



Clojure 初相识

刘家财

<http://liujiacai.net/>

提纲

- ◆ Clojure 特性介绍
 - ◆ functional / dynamic
 - ◆ Hosted on JVM
 - ◆ designed for concurrency
 - ◆ Another Lisp dialect
- ◆ Clojure 开发环境搭建
 - ◆ Emacs + Cider; IntelliJ + Cursive
 - ◆ Lein; boot-clj

特性介绍

Functional/Dynamic

- ◆ It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.——图灵奖得主 Alan J. Perlis
- ◆ 函数是一等成员
- ◆ Reified language construct.

Functional/Dynamic

- ◆ It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.——图灵奖得主 Alan J. Perlis
- ◆ 函数是一等成员
- ◆ Reified language construct.

Hosted on JVM

Clojure source

```
(defn foo 1)
(defn bar 2)
(println
 (+ foo bar))
```

compile

Java bytecode

```
const #1 foo
const #2 bar

static {} {
  invoke
}
```

load/eval

JVM

Clojure AOT Compilation

Hosted on JVM

- ◆ 不用担心类库没有
- ◆ 需具备一定的Java知识
- ◆ 其他实现：
 - ◆ ClojureScript: targets javascript
 - ◆ Ferret: targets ISO C++11

Designed for Concurrency

- ◆ 不可变的数据结构: list/vector/map/set
- ◆ 内置的并发操作函数: pmap, future...
- ◆ clojure.core.reducers
- ◆ Lock-free STM 软件事务模型
- ◆ core/async 同步方式写异步代码

STM

```
(def account (ref 100))
```

```
(defn transfer [acc1 acc2 amount]  
  (dosync  
    (alter acc1 - amount)  
    (alter acc2 + amount)))
```

经典案例：转账

Another Lisp dialect

```
(defn hello [greeting]  
  (println "Hello " greeting))
```

```
(hello "world")
```

括号语言

Another Lisp dialect

- ◆ what you see is what the compiler see.
- ◆ 直接操作抽象语法树 (AST)
- ◆ Code writing code

环境搭建

构建工具



leín ✓



boot

构建工具

- ◆ lein : 声明式, 一个大map, 类似 maven
- ◆ boot: 函数组合, 更加灵活; 类似 gradle
- ◆ 不用小看构建工具, 编程大部分时间花费在环境问题上

```

:min-lein-version "2.5.0"

:uberjar-name "hivez.jar"

:cljsbuild {:builds {:app {:source-paths ["src/cljs"]
                           :compiler {:output-to "resources/public/js/app.js"
                                       :output-dir "resources/public/js/out"
                                       :source-map "resources/public/js/out.js.map"
                                       :preamble ["react/react.min.js"]
                                       :externs ["react/externs/react.js"
                                                "resources/public/js/extern/leaflet.js"
                                                "resources/public/js/extern/leaflet.draw.js"]}
                           :optimizations :none
                           :pretty-print true}}}}

:profiles {:dev {:repl-options {:init-ns hivez.server
                               :nrepl-middleware [cemerick.piggieback/wrap-cljs-repl]}
            :plugins [[lein-figwheel "0.1.4-SNAPSHOT"]]
            :figwheel {:http-server-root "public"
                      :port 3449
                      :css-dirs ["resources/public/css"]}
            :env {:is-dev true}
            :cljsbuild {:builds {:app {:source-paths ["env/dev/cljs"]}}}}

:uberjar {:hooks [leiningen.cljsbuild]
          :env {:production true}
          :omit-source true
          :aot :all
          :cljsbuild {:builds {:app
                               {:source-paths ["env/prod/cljs"]
                                :compiler
                                {:optimizations :none
                                 :pretty-print false}}}}}})

```

project.clj

```
(defn dev-handler []
  (-> server/handler (reload/wrap-reload)
    (file/wrap-file "target/public")
    (file-info/wrap-file-info)))

(deftask dev-cljs
  "Build cljs for development."
  []
  (comp (watch)
    (speak)
    (reload :on-jsload (symbol "lokate.app/go!"))
    (cljs :source-map true
      :optimizations :none
      :output-to "public/js/main.js"))))

(deftask dev-serve
  "Start server for development."
  []
  (with-post-wrap fileset (server/run (dev-handler))))

(deftask dev
  "Build cljs and start server for development."
  []
  (comp
    (dev-cljs)
    (dev-serve)))

(deftask prod
  "Build application uberjar with http-kit main."
  []
  (comp (cljs :unified true
    :source-map true
    :optimizations :none
    :output-to "public/js/main.js")
    (aot :all true)
    (uber)
    (jar :file "lokate.jar" :main 'lokate.server)))
```

build.boot

IntelliJ+Cursive

- ◆ <https://www.jetbrains.com/idea/download>
- ◆ <https://cursive-ide.com/> (只能用lein)

Project: hello-cljsc

- hello-cljsc ~/dev/hello-cljsc
 - .idea
 - dev-resources
 - images
 - projectFilesBackup
 - resources
 - src
 - hello_cljsc
 - core.clj
 - target
 - test
 - .gitignore
 - .nrepl-port
 - hello-cljsc.iml
 - project.clj
 - README.md
- External Libraries
 - < 1.7 > /Library/Java/Java
 - Leiningen: args4j:2.0.26
 - Leiningen: clojure-complet
 - Leiningen: com.google.codi
 - Leiningen: com.google.guar
 - Leiningen: com.google.java
 - Leiningen: com.google.java
 - Leiningen: com.google.prot
 - Leiningen: org.clojure/cloj
 - Leiningen: org.clojure/cloj
 - Leiningen: org.clojure/data
 - Leiningen: org.clojure/goog
 - Leiningen: org.clojure/goog
 - Leiningen: org.clojure/tool
 - Leiningen: org.clojure/tool
 - tools.reader-0.8.10.jar
 - clojure.tools
 - reader
 - reader.clj
 - META-INF
 - Leiningen: org.json/json:20
 - Leiningen: org.mozilla/rhin

```
(reader/read-string "(+ 1 [2 3] {1 2} #{1 2} #_[1 23 3])")

;; Reading a string will result in Clojure data structures that we can be manip
;; regular Clojure code!
(map type (reader/read-string "(+ 1 [2 3] {1 2} #{1 2})"))

;; =====
;; Utilities

;; First, we define a series of utility helper functions which will simplify
;; our interactions with the ClojureScript analyzer and compiler.

;; A simple helper to emit ClojureScript compiled to JavaScript
;; as a string.
(defn emit-str [ast]
  (with-out-str (c/emit ast)))

;; A simple helper which allows us to read ClojureScript source from a string
;; instead of having to bother with files.
(defn string-reader [s]
  (clojure.lang.LineNumberingPushbackReader. (java.io.StringReader. s)))

;; A simple helper that takes a stream and returns a lazy sequences of
;; read forms.
(defn forms-seq [stream]
  (let [rdp (readers/indexing-push-back-reader stream 1)
        forms-seq* (fn forms-seq* []
                     (lazy-seq
                      (if-let [form (reader/read rdp nil nil)]
                        (cons form (forms-seq*))))))]
    (forms-seq*)))

;; Getting a seq of s-expressions.
(forms-seq (string-reader "(+ 1 2)"))

;; Evaluate the following expressions.

;; form-seq will return a seq containing two forms.
(forms-seq (string-reader "(+ 1 2) (+ 3 4)"))

;; The first form is (+ 1 2)
(first (forms-seq (string-reader "(+ 1 2) (+ 3 4)")))

;; The first form is a list.
(first (forms-seq (string-reader "(fn [x y]\n(+ x y)")))

;; The first form in (fn [x y] (+ x y)) is a symbol
(ffirst (forms-seq (string-reader "(fn [x y]\n(+ x y)")))

;; The second form in (fn [x y] (+ x y)) is a vector
(second (first (forms-seq (string-reader "(fn [x y]\n(+ x y)")))))
```

REPL Local: hello-cljsc.core

```
(map type (reader/read-string "(+ 1 [2 3] {1 2} #{1 2})"))
=>
(clojure.lang.Symbol
 java.lang.Long
 clojure.lang.PersistentVector
 clojure.lang.PersistentArrayMap
 clojure.lang.PersistentHashSet)
(defn emit-str [ast]
  (with-out-str (c/emit ast)))
=> #'hello-cljsc.core/emit-str
(defn emit-str [ast]
  (with-out-str (c/emit ast)))
=> #'hello-cljsc.core/emit-str
(defn string-reader [s]
  (clojure.lang.LineNumberingPushbackReader. (java.io.StringReader. s)))
=> #'hello-cljsc.core/string-reader
(defn forms-seq [stream]
  (let [rdp (readers/indexing-push-back-reader stream 1)
        forms-seq* (fn forms-seq* []
                     (lazy-seq
                      (if-let [form (reader/read rdp nil nil)]
                        (cons form (forms-seq*))))))]
    (forms-seq*)))
=> #'hello-cljsc.core/forms-seq
(forms-seq (string-reader "(+ 1 2)"))
=> ((+ 1 2))
(forms-seq (string-reader "(+ 1 2) (+ 3 4)"))
=> ((+ 1 2) (+ 3 4))
(first (forms-seq (string-reader "(+ 1 2) (+ 3 4)")))
=> (+ 1 2)
```

Emacs + Cider (高级用户)

- ◆ <https://www.gnu.org/software/emacs/>
- ◆ <https://github.com/clojure-emacs/cider>
- ◆ 推荐配置：
 - ◆ <https://www.braveclojure.com/basic-emacs/>
- ◆ 我的Emcas配置 (Ruby/CL/Node/Python/Clojure) :
 - ◆ <https://github.com/jiacai2050/conf/tree/master/.emacs.d>

```

(ns cider.nrepl.middleware.info
  (:require [clojure.string :as str]
            [clojure.java.io :as io]
            [cider.nrepl.middleware.util.cljs :as cljs]
            [cider.nrepl.middleware.util.java :as java]
            [cider.nrepl.middleware.util.misc :as u]
            [clojure.repl :as repl]
            [cljs-tooling.info :as cljs-info]
            [clojure.tools.nrepl.transport :as transport]
            [clojure.tools.nrepl.middleware :refer [set-descriptor!]]
            [clojure.tools.nrepl.misc :refer [response-for]])

(defn maybe-protocol
  [info]
  (if-let [prot-meta (meta (:protocol info))]
    (merge info {:file (:file prot-meta)
                 :line (:line prot-meta)})
    info))

(def var-meta-whitelist
  [:ns :name :doc :file :arglists :macro :protocol :line :column :static :added :deprecated :resource-path])

(defn map-seq [x]
  (if (seq x)
    x
    nil))

(defn var-meta
  [v]
  (→ v meta maybe-protocol (select-keys var-meta-whitelist) map-seq))

(defn ns-meta
  [ns]
  (merge
   (meta ns)
   {:ns ns
    :file (→ (ns-publics ns)
              first
              second
              var-meta
              :file)
    :line 1}))

(defn resolve-var
  [ns sym]
  (if-let [ns (find-ns ns)]
    (try (ns-resolve ns sym)
         ;; Impl might try to resolve it as a class, which may fail
         (catch ClassNotFoundException _
               nil)
         ;; TODO: Preserve and display the exception info
         (catch Exception _
               nil)))
    nil))

(defn resolve-aliases
  [ns]
  (if-let [ns (find-ns ns)]
    (info.clj)
    nil))

--:--- info.clj Top of 9.0k (16,6) Git-master | Clojure cider[cider.nrepl.middleware.info] | --:--- *cider-repl cider-nrepl* All of 361 (5,33) (REPL, Paredit company-capf Projectile[cider

reset!: [[atom newval]]

[clojure.core/merge
 (& maps)]
 Added in 1.0
 Returns a map that consists of the rest of the maps conj-ed onto
 the first. If a key occurs in more than one map, the mapping from
 the latter (left-to-right) will be the mapping in the result.

--:--- *cider-doc* All of 243 (1,0) (Doc company Projectile[cider-nrepl])
: CIDER 0.6.0snapshot (package: 20141116.1221) [Java 1.7.0_17, Clojure 1.5.1, nREPL 0.2.6]
user>
cider.nrepl.middleware.info> (var-meta #'maybe-protocol)
{:column 1, :line 13, :arglists ([info]), :file "/Users/bozhidar/projects/cider-nrepl/src/cider/nrepl/middleware/info.clj", :name maybe-protocol, :ns #<Namespace cider.nrepl.middleware.info>}
cider.nrepl.middleware.info> [res]
reset!
reset-meta!
resolve
resolve-aliases
resolve-special
resolve-var
resource-path
response-for
rest
restart-agent
```

其他方式

- ◆ VIM: <https://github.com/tpope/vim-fireplace>
- ◆ Eclipse: <http://doc.ccw-ide.org/>
- ◆ Atom: <https://atom.io/packages/proto-repl>
- ◆ <http://clojure-doc.org/articles/content.html>

Thank You.



群名称: SICP读书群
群号: 119845407



公众号