http://clojure.org/

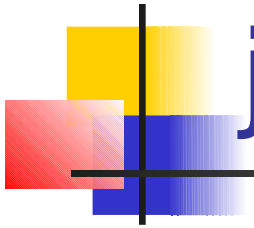
Clojure 第一眼

涧泉

2013-4-28

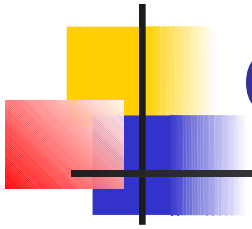
免责声明

- 文中知识点仅凭个人记忆和理解，不保证正确性。



joke

- 你喜欢函数式编程？是啊。我也是，你喜欢 Erlang 、 Lua 、 Lisp 还是 Haskell ？ Lisp 。我也是，你喜欢 Common Lisp 、 Emacs Lisp 、 Scheme 还是 Clojure ？ Scheme 。我也是，你喜欢 MIT Scheme 、 MzScheme 、 DrScheme 还是 Guile ？MIT Scheme 。我也是，你喜欢 7.5 版还是 7.4 版？ 7.5 。滚！异端
- 转自 @weibo



clojure

- LISP

  简单语法、 Symbol 、宏
- JVM

  字节码、解释执行、 Java 交互
- & More

  函数式、动态类型、并发支持

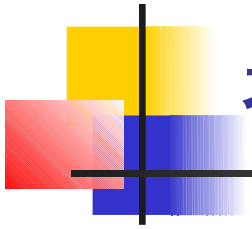
  通用编程语言

REPL（交互式控制台）

- Read Eval Print Loop

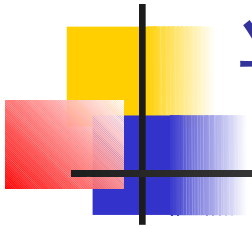
  读取表达式，求值，输出，循环
- 解释执行 REPL
- idea 、 eclipse 、 apt-get

```
hanyong@han:~$ clojure
Clojure 1.4.0
user=> (+ 2 3)
5
user=> (println "hello world")
hello world
nil
user=>
```



表达式

- 2 + 3 = ?
- (+ 2 3)
- (+ 2 3 4)
- (println “hello world”)
- nil
- (class “abc”)



语法 & 语义

- 语法

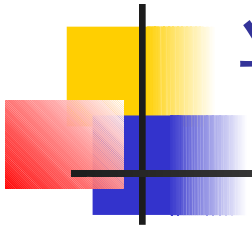
  原子、集合

  语法树？
- 语义

  special forms => (if)

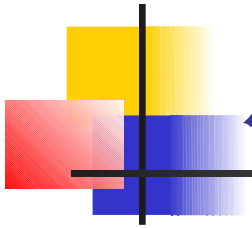
  宏 => (for)

  函数 => (println)



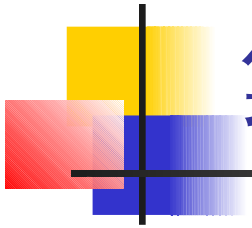
语法

- 原子
- 集合



原子

- user=> \a ; 字符
- \a
- user=> "string" ; 字符串
- "string"
- user=> 456 ; 整数
- 456
- user=> true false ; bool
- true
- false
- user=> :key ; keyword
- :key
- user=> nil
- nil



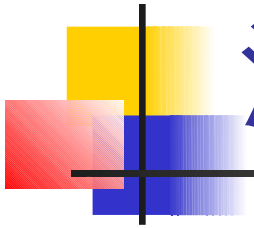
集合

- user=> [1 2 :is true "abc"]
- [1 2 :is true "abc"]

- user=> {:x 5, "pos" [3 2], [99 99] :good}
- {[99 99] :good, :x 5, "pos" [3 2]}

- user=> (list 44 :fast "joke")
- (44 :fast "joke")

- user=> #{"abc" false {:x 8} nil 42.5}
- #{nil 42.5 {:x 8} "abc" false}

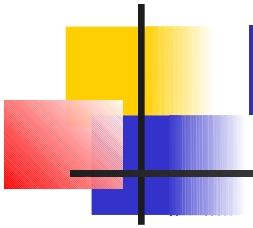


注释和可选分隔符

- user=> ; (print "hello") 这里只是注释

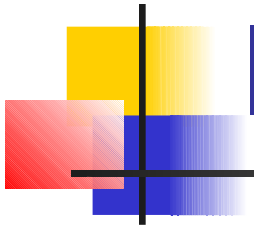
- user=> 1,456,,,,,,,,,72.3,,,,,,,,,,,,
- 1
- 456
- 72.3

- user=> {1, :x 5, :y 6, true} ； 分隔符被忽略
- {1 :x, 5 :y, 6 true}



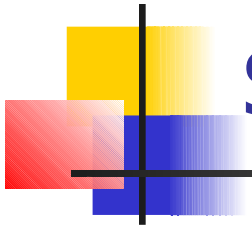
list 和 quote(1)

- clojure 对每个表达式求值
- 递归求值
- 求值 list 表达式发生函数 (语义) 调用
- quote 禁止求值
- 简写 (')
- 表达式自闭合性



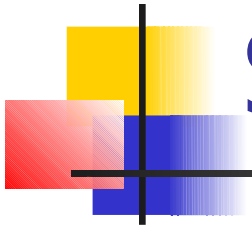
list 和 quote(2)

- user=> (println "hello")
- hello
- nil
- user=> (quote (println "hello"))
- (println "hello")
- user=> (= [(+ 2 3)] [5])
- true
- user=> (= '[(+ 2 3)] '[5])
- false



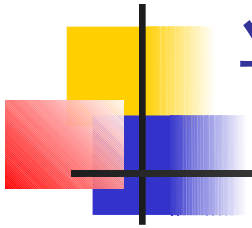
symbol

- user=> x
- CompilerException java.lang.RuntimeException: Unable to resolve symbol: x in this context, compiling: (NO_SOURCE_PATH:0)
- user=> (def x)
- #'user/x
- user=> x
- #<Unbound Unbound: #'user/x>
- user=> (def x 3)
- #'user/x
- user=> x
- 3
- user=> (print x)
- 3nil
- user=> (println x)
- 3
- nil
- user=> (println 'x "=" x)
- x = 3
- nil
- user=> (class x)
- java.lang.Long
- user=> (class 'x)
- clojure.lang.Symbol



Symbol 命名约定

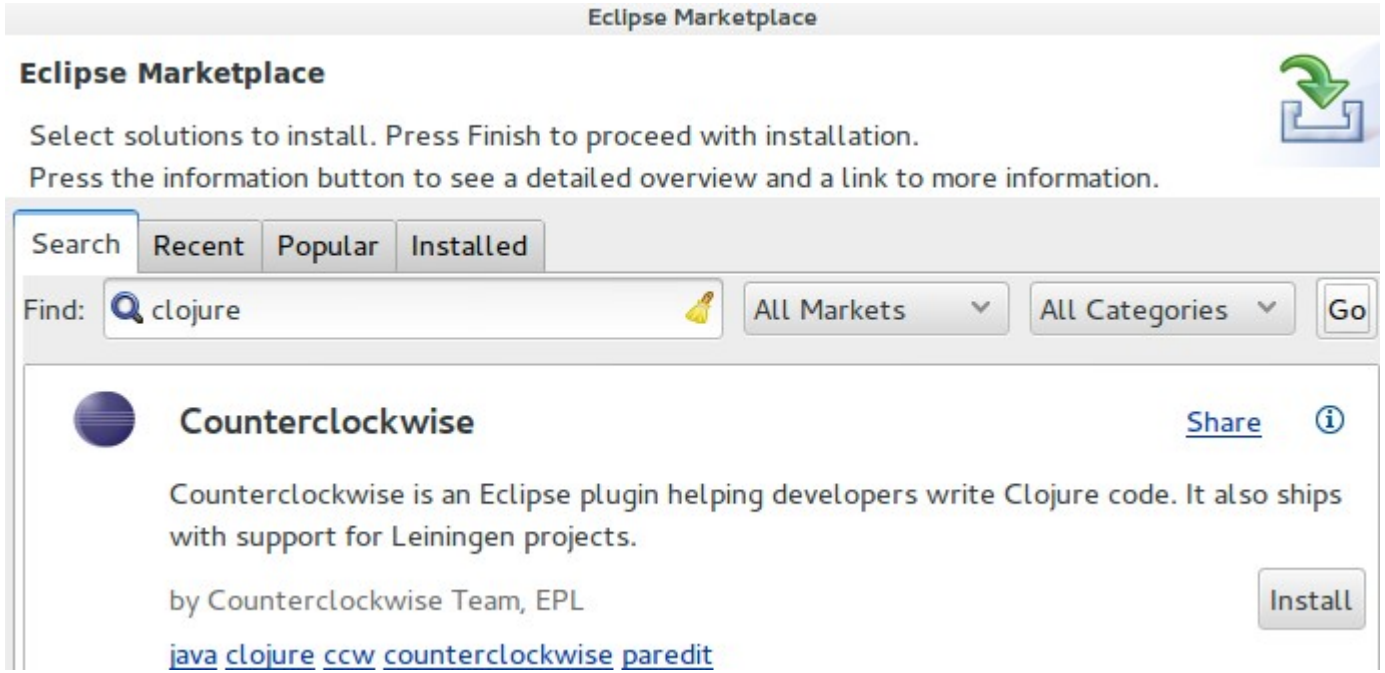
- user=> (empty? [])
- true
- user=> (vector? [])
- true
- user=> (every? integer? [1 2 3 4])
- true
- user=> (every? integer? [1 2 3 4 "ab"])
- false
- user=> (not-any? integer? [1 2 3 4 "ab"])
- false
- user=> (not-any? integer? ["ab"])
- true



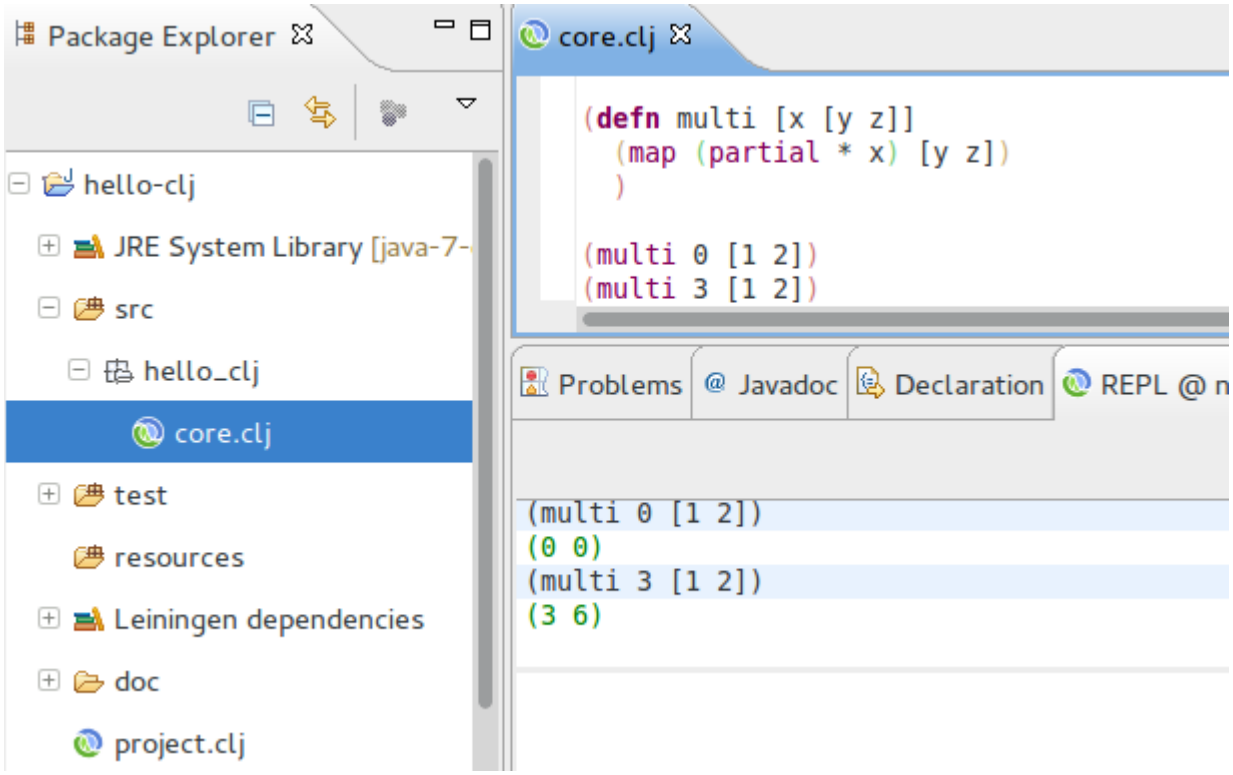
语法小结

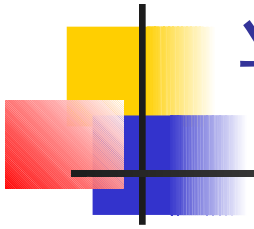
- 分词规则和数据结构？
- 语法树？
- json++ ？
- Q&A？

eclipse 安装


Eclipse Marketplace
Eclipse Marketplace
Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.
Search
Recent
Popular
Installed
Find:
clojure
All Markets
All Categories
Go
Counterclockwise
Share
Counterclockwise is an Eclipse plugin helping developers write Clojure code. It also ships with support for Leiningen projects.
by Counterclockwise Team, EPL
Install
java clojure ccw counterclockwise paredit

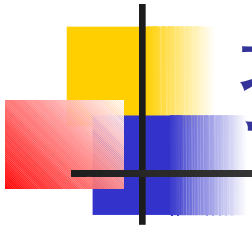
eclipse 开发


Package Explorer
hello-clj
JRE System Library [java-7-
src
hello_clj
core.clj
test
resources
Leiningen dependencies
doc
project.clj
core.clj
(defn multi [x [y z]]
(map (partial * x) [y z])
)
(multi 0 [1 2])
(multi 3 [1 2])
Problems
Javadoc
Declaration
REPL @ n
(multi 0 [1 2])
(0 0)
(multi 3 [1 2])
(3 6)



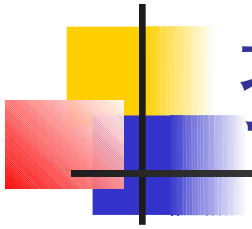
语义

- 所有语义都使用 list 表达式语法 (code is data ?)
- Special forms，宏，函数
- 控制结构，函数定义，函数
- 自定义语义，DSL?



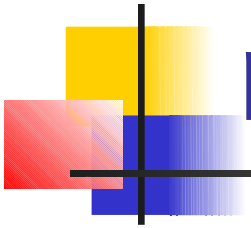
控制结构 (1)

- (def symbol init?) ； 符号定义，全局
- (do exprs*) ； 复合表达式
- (if test then else?) ； 分支
- (let [bindings*] exprs*) ； 局部推导
- (loop [bindings*] exprs* (.. recur ..)) ； 递归



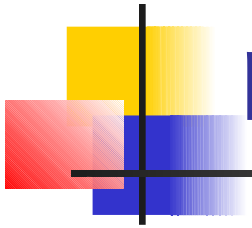
控制结构 (2)

- (for [e [elems*] :when test] expr) ；列表变换
- (doto obj (.method params*)*)；连续操作
- (-> expr (f params*)*)；连续首参数
- (->> expr (f params*)*)；连续末参数



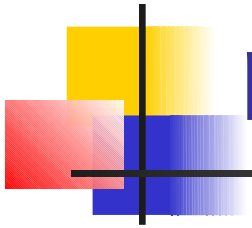
函数定义

- (fn [params*] exprs*)
- (defn name [params*] exprs*)
- #(action exprs*)
- 可变参数 [x & more]
- 参数结构 [x [y z]]
- 参数个数重载



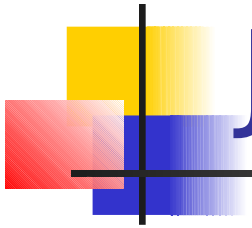
函数

- + - * quot rem /
- println class
- filter map partial
- apply concat lazy-cat
- http://clojuredocs.org/quickref/Clojure Core



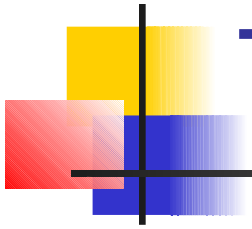
函数示例

```
(defn two
  ([] [])
  ([x & more]
    (lazy-cat [(* x 2)] (apply two more))
    ))

(two)
(two 1 2 3)

(defn multi [x [y z]]
  (map (partial * x) [y z])
  )

(multi 0 [1 2])
(multi 3 [1 2])
```



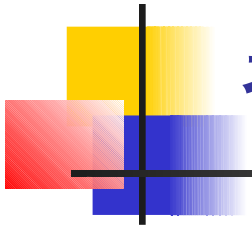
Java 交互

- (StringBuilder. "init-value")；构造函数
- (.append sb "str")；方法调用
- (import java.util.Random)
- (Math/PI) / (Math/abs -1)；静态成员



TODO

- 项目管理 (leiningen)
- 宏
- 多方法、 Java 接口
- 高阶函数、闭包
- 并行支持
- Q&A ?



参考

- 尔术 《clojure make fun》
- The Blackstag Blog http://blackstag.com/blog.posting?id=6
- 为什么 Lisp 语言如此先进 http://www.ruanyifeng.com/blog/2010/10/why_lisp_is_superior.html