Clojure 简介

当Lisp遇上了Java

内容概要

- Clojure语法简介
- 初探函数式编程
- 学习资料

稍等，我们先装一个Clojure

- http://leiningen.org/#install

- 在线clojure：http://clojure.guangyu.me
- (Powered by tryclj.com)

什么是Clojure

- 函数式编程
- Lisp变体
- 运行在Java虚拟机上

- 简单 强大

Clojure语法简介

语法格式

- (函数 参数1 参数2 ...)
- (宏 参数1 参数2 ...)
- (Clojure保留字 参数1 参数2 ...)

- 总之，全是小括号

（不不，其实我们还有中括号，大括号 ...）

先学学算术

- `(+ 1 2)`

```
-> 3
```

- `(> 2 1)`

```
-> true
```

- 练习：__ 应该填什么让整个式子为true?

```
(= (- 10 (* 2 3)) __)
```

数据结构

- 向量
  `[1 2 3 4]`
- 列表
  `'(1 2 3 4)`
- 集合
  `#{1 2 3 4}`
- 映射表
  `{1 "a" 2 "b"}`
- 空
  nil
- 大整数
  1000000000000N
- 大小数
  1.0000000000001M
- 字符
  \a
- 关键字
  :a
- 正则表达式
  #"\s"

条件判断

- `(if (even? 10) "An even number" "An odd number")`

- ```
  (cond
    (= 1 n) do-something
    (= 2 n) do-something-too
    :else do-nothing)
  ```
  ```

列表简单操作

- (conj [1 2 3] 4)
-> (1 2 3 4)

- (conj '(1 2 3) 4)
-> (4 1 2 3)

- (conj #{1 2 3} 3)
-> #{1 3 2}

- (conj {1 "a" 2 "b"} [3 "c"])
-> {3 "c", 1 "a", 2 "b"}

- (into [] '(1 2 3))

- concat cons assoc

- (first [1 2 3])
-> 1

- (first {1 "a" 2 "b"})
-> [1 "a"]

- last butlast rest next

- rest与next区别（之一）
- (rest '())
-> ()
- (next '())
-> nil

定义一个常量

- `(def 常量名 值)`

- 临时定义/改变一个常量
- `(let [常量1 值1 常量2 值2 ...] expr)`

- `(def a 1)`
- `(let [a 2] a)`

`-> 2`

- `a`

`-> 1`

- 变量很难用，应该尽量用常量（今天我不讲）

初探函数式编程

纯函数

- 纯天然，不含添加剂，无副作用
- 满足下面两点：
  - 返回值仅取决于输入参数
  - 执行过程对环境不造成影响

- 其中影响包括但不限于
  - 屏幕输出
  - 文件操作
  - 网络连接

- 衍生规则
  - 没有变量！没有循环！没有蛀牙！

定义一个函数

- `(defn fn-name [arg1 arg2] expr)`
-
```
(defn fn-name
  ([arg]
   expr1)
  ([arg1 arg2]
   expr2))
```
- 可变参数
- `(defn fn-name [arg1 & argn] expr)`

匿名函数

- `(fn name? [arg1 arg2] expr)`
- `#(* % 3)`
- `#(+ %1 %2 %&)`
- `(partial + 5)`

- 让我们随便试几个例子

尾递归

```
(defn triangle-number
  ([n]
    (triangle-number n 0))
  ([n acc]
    (if (zero? n)
      acc
      (triangle-number (dec n)
                       (+ acc n)))))

(triangle-number 100000000)
-> StackOverflowError   clojure.lang.Numbers
$LongOps.isZero (Numbers.java:435)
```

尾递归

```
(defn triangle-number-recur
  ([n]
   (triangle-number-recur n 0))
  ([n acc]
   (if (zero? n)
     acc
     (recur (dec n) (+ acc n)))))

(triangle-number-recur 100000000)
-> 5000000050000000
```

返回函数的函数

```
(defn plus-n
  [n]
  (fn [x] (+ x n)))

(def plus-5 (plus-n 5))
(plus-5 2)
-> 7
```

函数与列表

- (apply + '(1 2 3)) 相当于 (+ 1 2 3)

- (map inc [1 2 3])
-> (2 3 4)
- (map str ["A" "B" "C"] ["a" "b" "c"])
-> ("Aa" "Bb" "Cc")

- (filter even? (range 10))
-> (0 2 4 6 8)

- ([1 2 3] 2)
-> 3
- (:a {:a 1 :b 2})
-> 1
- ({:a 1 :b 2} :a)
-> 1

reduce

- `(defn plus2 [a b] (+ a b))`
- `(reduce plus2 [1 2 3 4])`

`-> 10`

- 运算过程:
- `(plus2 1 2)`

`-> 3`

- `(plus2 3 3)`

`-> 6`

- `(plus2 6 4)`

`-> 10`

范例：Project Euler 001

- 题目：求出1000以下3或5的倍数的和

```
(use 'clojure.set)
(defn sum-of-3-5
  [n]
  (apply + (set (union (range 0 n 3)
                       (range 0 n 5)))))
```

对比：C语言实现

```
int pe001 (int n)
{
  int i, sum=0;

  for (i=0; i<n; i++) {
    if ((i%3 == 0) || (i%5 == 0))
      sum += i;
  }

  return sum;
}
```

for

```
(for [x [1 2 3] y [2 3 4] :when (even? (* x y))] (list x y))
-> ((1 2) (1 4) (2 2) (2 3) (2 4) (3 2) (3 4))
```

- 刚才范例的另一种实现

```
(defn sum-of-3-5-
  [n]
  (apply + (for [x (range n)
                 :when (or (zero? (rem x 3))
                           (zero? (rem x 5)))]
             x)))
```

- 对比：Python的for

```
sum([x for x in range(1000) if x % 3 == 0 and x % 5 == 0])
```

函数的组合

- 写一个函数，实现下面的功能
- ((my-func + max min) 1 2 3 4)

-> (10 4 1)

```
(defn my-juxt
  [& func-list]
  (fn [& arg-list]
    (map #(apply % arg-list) func-list)))
```

学习资料

- http://braveclojure.com
- http://www.4clojure.com
- http://clojure-euler.wikispaces.com/
- 《Clojure程序设计》
- 《Clojure编程》