

# Shake , 宏与 DSL

```
(sh/ifconfig wlan0)
```

- 孙宁
- 美味书签
- [github.com/sunng87](https://github.com/sunng87)

# 缘起

- <http://amoffat.github.com/sh/>
- ipython, iREPL
- 包装 subprocess , 优雅 API

# 别人如何做

- `clojure.java.shell`

```
(sh "ifconfig wlan0")
```

# 别人如何做

- pallet

```
(sh (ifconfig "wlan0"))
```

# 别人如何做

- conch

```
(program ifconfig)  
(ifconfig wlan0)
```

# 别人如何做

- sh

```
ifconfig("wlan0")
```

# Shake 如何做

```
(sh/ifconfig wlan0)
```



# Shake 的实现

```
(.start (ProcessBuilder. "ifconfig" "wlan0"))
```

# 从 API 到实现

- 无需声明 (~~conch~~)
- 没有外层的宏 (~~pallet~~)
- 参数都以 **symbol** 的方式传入 (~~java.shell~~)
- Shell 命令都是 Clojure 代码中可引用的 **var**

# 实现方案

- 宏
- 通过程序自动创建 var

# 宏

- 可替换展开的文本模式
  - Lisp
  - C
  - sweet.js
- Lisp: Programmable programming language
- Lisp: Homoiconicity (同像性)

# 宏与函数

- 宏对代码编程，函数对数据编程
- 宏接受字面量，符号 (symbol)
- 函数接受值
- 宏自外向内展开，函数自内向外求值

```
(let [x 1]
  (a-macro 5 x (+ 1 2)))
```

The diagram illustrates the arguments of the `a-macro` function. Three blue callout boxes point to the arguments: `5` is labeled `int`, `x` is labeled `symbol`, and `(+ 1 2)` is labeled `list`.

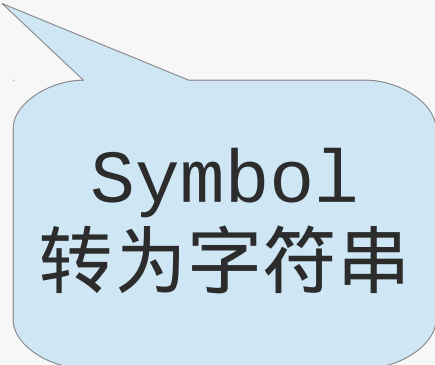
```
(let [x 1]
  (a-fn 5 x (+ 1 2)))
```

The diagram illustrates the arguments of the `a-fn` function. A single blue callout box points to the argument `5`, which is labeled `int`.

# shake 的宏

```
(ifconfig wlan0)
```

```
(defmacro ifconfig [& args]  
  `(start (ProcessBuilder.  
           (list "ifconfig"  
                 ~@(map str args)))))
```



Symbol  
转为字符串

# 自动创建 var 指向宏

- intern
- eval

写一段程序，生成一段宏；这段宏，生成另一段程序

# 自动创建 var

- intern



symbol

```
(intern *ns* (symbol "my-var")  
            (fn [] (...)))
```



value



# 其实宏也是函数

```
(defn dummy-macro [&form &env & args]  
  `(+ ~@args))
```

```
(.setMacro (var dummy-macro))
```

```
(macroexpand '(dummy-macro 1 2 x y (z)))  
(clojure.core/+ 1 2 x y (z))
```

dummy-macro

CompilerException

java.lang.RuntimeException:

Can't take value of a macro

# 自动创建 var

- eval

```
(eval `(defmacro my-var [] (...)))
```

```
(let [my-var "my-var"]  
  (eval `(defmacro ~(symbol my-var)  
                []  
                (...))))
```

对 macro 来说 eval 的方式更简单一些

# 放到一起

```
(doseq [exec (dir "/usr/bin")]  
  (eval `(defmacro ~(symbol exec)  
    [& args#]  
    (let [str-args#  
      (conj (map str args#)  
        ~exec)]  
      `( .start  
        (ProcessBuilder.  
          (list ~@str-args#)))))))
```

# 访问 clojure 的 var 和 binding

- 基于 symbol 的设计导致无法访问 clojure 的值
- 借鉴 shell 访问变量的方式
- \$PATH

```
(let [home "/home/nsun"]  
  (ls $home))
```

# 宏可以修改代码

- 修改传入的符号

```
(for [x# args#  
      :let [y# (if (.startsWith (str x#) "$")  
                    (symbol (subs (str x#) 1))  
                    (str x#))]]  
  y#)
```

```
(ls -l $home)  
(ProcessBuild. (list "ls" "-l" home))
```

# 限制和 TODO

- 受 reader 的限制：
  - (date +%s)
  - (xargs -0)
- 迟加载
- 管道
  - (`| (grep Hello $file-path) (wc -l)`)

项目地址：

<http://github.com/sunng87/shake>

欢迎 pull request

谢谢！