Scala og Clojure

Eivind Barstad Waaler

BEKK

10.05.2010

Clojure

- Nytt JVM språk – 2007
- Versjon 1.1.0
- Dynamisk typet
- Funksjonell programmering
- Kompilert til bytecode
- REPL
- Lisp

Clojure – Lisp

- LISt Processing
- `(<fn> <arg1> <arg2> ...)`
- Homoiconic (data $\longleftrightarrow$ kode)
- Makroer!

```clojure
; Definer noen verdier
(def x 5)
(def y (* 6 7))

; Skriv dem ut..
(println x y)
```

Fibonacci – FP “Hello world!”

```
(defn fib [n]
  (if (<= n 1)
    n
    (+ (fib (- n 1)) (fib (- n 2)))
  )
)
```

Få spesialformer

```clojure
(defn halfOrSame [n]
  (if (> n 2)
    (/ n 2)
    n))
```

```java
public static int halfOrSame(int n) {
  if (n > 2)
    return n / 2;
  else
    return n;
}
```

```scala
def halfOrSame(n: Int) = if (n > 2) n / 2 else n
```

Funksjonell Programmering

```
(reduce + [1 2 3 4])

(map (fn [x] (* 2 x)) [1 2 3 4])

(map #(* 2 %) [1 2 3 4])
```

Lagre data uten klasser?

- Kraftige datastrukturer
- Immutable + persistent
- Collections – Maps, Lists, Vectors, Sets
- Bra match for NoSQL?

Data – Maps, Keywords and Structs

```
(def inventors {"Scala" "Odersky", "Clojure" "Hickey"})

(inventors "Scala")

(def inventors {:Scala "Odersky", :Clojure "Hickey"})

(:Scala inventors)

(defstruct lang :name :inventor)

(def scala (struct lang "Scala" "Odersky"))

(:inventor scala)

(def clj (struct-map lang :name "Clojure" :year 2007))
```

Makroer – Kode og data

- Utvid kompilatoren med bruker-kode
- To steg:
  1. Makro kjøres (macro expansion)
  2. Kode kompileres
- Quote, unquote – egne symboler

```
(defmacro unless [expr form]
  (list 'if expr nil form))

(unless false (println "funker finfint"))
```

Enkle makroer – Scala by-name parametre

- Enkler makroer – kun utsatt evaluering
- Scala by-name parametre
- Evalueres ved bruk, ikke ved overføring

```
def unless(cond: => Boolean)(body: => Unit) {
  if(!cond) body
}

unless(false) {
  println("Funker fint dette og!")
}
```

Makro med “binding”

```
(def bit-bucket-writer
  (proxy [java.io.Writer] []
    (write [buf] nil)
    (close []    nil)
    (flush []    nil)))

(defmacro noprint [& forms]
  `(binding [*out* bit-bucket-writer]
     ~@forms))

(println "Regular println!")
(noprint (println "Noprint println!"))
```

Skriv kode baklengs :)

```clojure
(defmacro rev [& body]
  (let [rev# (reverse body)]
    `(do ~@rev#)))

(def result
  (rev
    (* x y)
    (def y 4)
    (def x 3)))

(println "Result:" result)
```

Hastighet

- Dynamiske JVM språk trege
- Clojure tricks
    1. Type hints
    2. Memoization

```
(+ (int 42) (int 35))

(defn len [x]
  (. x (length)))

(defn len2 [#^String x]
  (. x (length)))
```

Hastighet – memoization

```clojure
(defn slow-double [n]
  (Thread/sleep 100)
  (* n 2))

(def mem-double (memoize slow-double))

(def values [1 2 1 2 1 2])

(time (dorun (map slow-double values)))
; "Elapsed time: 602.931 msecs"

(time (dorun (map mem-double values)))
; "Elapsed time: 200.744 msecs"
```

Samtidighet

- Immutable er bra
- Hvordan dele endringer av data?
    - Atoms – atomisk verdi
    - Refs – STM (Software Transactional Memory)
    - Agents – Scala actors?

Multimethods

- Dynamisk dispatch – polymorfisme “on steroids”

```
(defmulti my-print class)

(defmethod my-print String [s]
  (.write *out* s))

(defmethod my-print Number [n]
  (.write *out* (.toString n)))

(my-print "test")
(my-print 123)
(my-print 23.4)
```

Konklusjon

- Clojure er gøy!
- Veldig konsis kode
- Makroer rocker – DSL heaven
- Bra for samtidighet
- Kompilator optimalisering tilgjengelig