Langage Update (Clojure)

LL Tiger 2010/07/31

Toshiaki Maki

(Twitter:@making)

Language Update初登場なので

今回はClojureの言語紹介をします

スライド50ページもありますw

Agenda

- Clojure?
- Immutable
- Concurrency
- Program as Data
- etc

Clojure?

- new Lisp dialect
  - not CommonLisp , Scheme
  - run on JVM
- Functional
- Immutable
- Concurrency
  - STM
  - agent system
- Program as Data
  - Macro
  - DSL
- Java Interoperability

プログラミングClojure (訳:川合史朗)の出版で

吹き荒れたClojure旋風

TLがClojureの話題で一色!!

空前のClojureブーム到来!!

そんな時代がClojure
にもありました orz

気を取り直して、再び興味をもってもらえるように紹介します(> <)

Clojure History

- 2007年スタート
- Author: Rich Hickey

http://en.wikipedia.org/wiki/Rich_Hickey より

- 2009/05/04 1.0.0リリース
- 2009/12/31 1.1.0リリース ←現在安定版
- 2010/07/13 1.2.0 βeta1 リリース
- 2010/07/30 1.2.0 RC1 リリース ←イマココ

- The Eclipse Public License

Position in JVM Languages

	Functional		Object Oriented
Native to the JVM	Clojure	Scala	Groovy
Ported to the JVM	Armed Bear CL Kawa		JRuby Jython Rhino

Stuart Sierraの発表資料より引用

Syntax (primitive)

clojure type	example	java type
string	“hoge”	String
character	¥h	Character
regex	#”ho*”	Pattern
integer	124	Integer/Long/BigInteger
double	1.2345	Double
	1.2345M	BigDecimal
ratio	3/4	N/A
boolean	true	Boolean
nil	nil	null
symbol	hoge, +	N/A
keyword	:hoge, ::hoge	N/A

Syntax (data structure)

type	example
list	(1 2 3)
vector	[1 2 3]
map	{:hoge 100 :foo 200} or {:hoge 100, :foo 200}
set	#{:hoge :foo}

Hello World

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

(hello "World")    ; -> "Hello World"
```

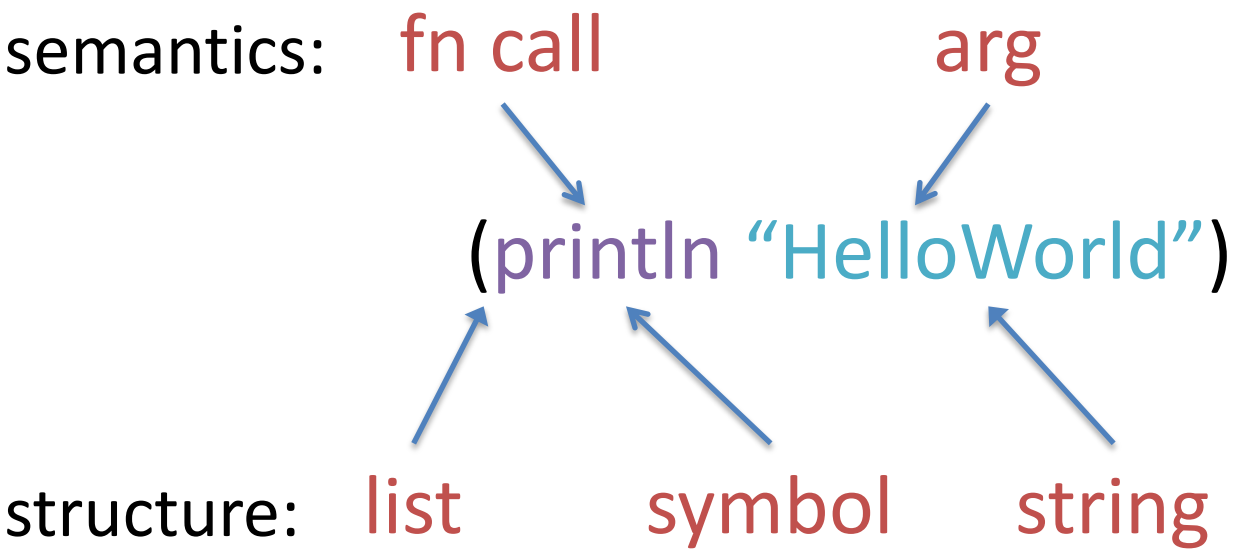
Javaの

```
public static void hello (String s) {
    System.out.println(“Hello ” + s);
}
```

```
hello(“World”);
```

に相当

Function Call



Java Interoperability

	Java	Clojure
import package	Import java.util.Date;	(import ‘java.util.Date)
new Instance	new Date();	(Date.)
invoke method	date.toString();	(.toString date)
static method	System.getenv(“PATH”);	(System/getenv “PATH”)

```
StringBuilder sb = new StringBuilder();
sb.append(“Lightweight”);
sb.append(“Language”);
sb.append(“Tiger”);
sb.append(“2010”);
```

```
(let [sb (StringBuilder.)]
  (.append sb “Lightweight”)
  (.append sb “Language”)
  (.append sb “Tiger”)
  (.append sb “2010”))
```

Agenda

- Clojure?
- Immutable
- Concurrency
- Program as Data
- etc

Immutable

- Clojureのデータは不変

```
(def ll-info {:name “LL Tiger” :year 2010})
(assoc ll-info :place “Nissho-Hall”)
;; -> {:place "Nissho-Hall", :name "LL Tiger", :year 2010}
ll-info
;; -> {:name "LL Tiger", :year 2010}
```

Mapに値を追加しても

元のMapはそのまま

Immutable

メモリ

Key	Value
:name	LL Tiger
:year	2010
:place	Nissho-Hall

ll-infoをassocされたmap

immutableなので共通部分を共有できる
→効率的なデータ構造

Immutable

- 関数型プログラミングに適したスタイル
  - 入力値に対して出力値の計算に集中できる
  - 並行処理にも有利

Agenda

- Clojure?
- Immutable
- Concurrency
- Program as Data
- etc

Concurrency

並行プログラミングを容易にする4種類の強力なAPI

	同期的	非同期的
協調的	今回はrefのみ紹介	
非協調的	atom	agent
スレッドローカル	var	

ref

- 状態を扱えるようにする
  - Clojureでは値がimmutableなので、関数適用しても元の値は変更されない
  - refを使用すると状態を扱える
- 複数の状態をall or nothingで変更
  - 複数の変数をtransactionalに変更できる

without “ref”

加算/減算処理を普通に考えると

```
(def a 0)
(def b 1)
(inc a) ; aの値を+1する関数
(dec b) ; bの値を-1する関数
(println a b) ; -> 1 0 ??
```

a,bはimmutableなので実際は「0 1」が出力される
a,b更新の一貫性も保証されない

STEP1/4

更新する値をrefでくるむ（変更可能な参照）

```
(def a (ref 0))
(def b (ref 1))
(inc a)
(dec b)
(println a b)
```

状態を保持する
変数を生成

まだ実行してもエラー

STEP2/4

alterでrefの中身を更新

```
(def a (ref 0))
(def b (ref 1))
(alter a inc) ; aにincを適用して更新
(alter b dec) ; bにdecを適用して更新
(println a b)
```

まだ実行してもエラー

(alter *reference update-fn & args…*)形式
(ref-set *reference new-value*)でも可

STEP3/4

derefでrefの中身を参照

```
(def a (ref 0))
(def b (ref 1))
(alter a inc)
(alter b dec)
(println (deref a) (deref b))
```

まだ実行してもエラー

リーダマクロ@を使って略記できる

(println @a @b)で可

STEP4/4

atomicにしたい範囲をdosyncで囲む

```
(def a (ref 0))
(def b (ref 1))
(dosync
  (alter a inc)
  (alter b dec))
(println @a @b)  ; -> 1 0
```

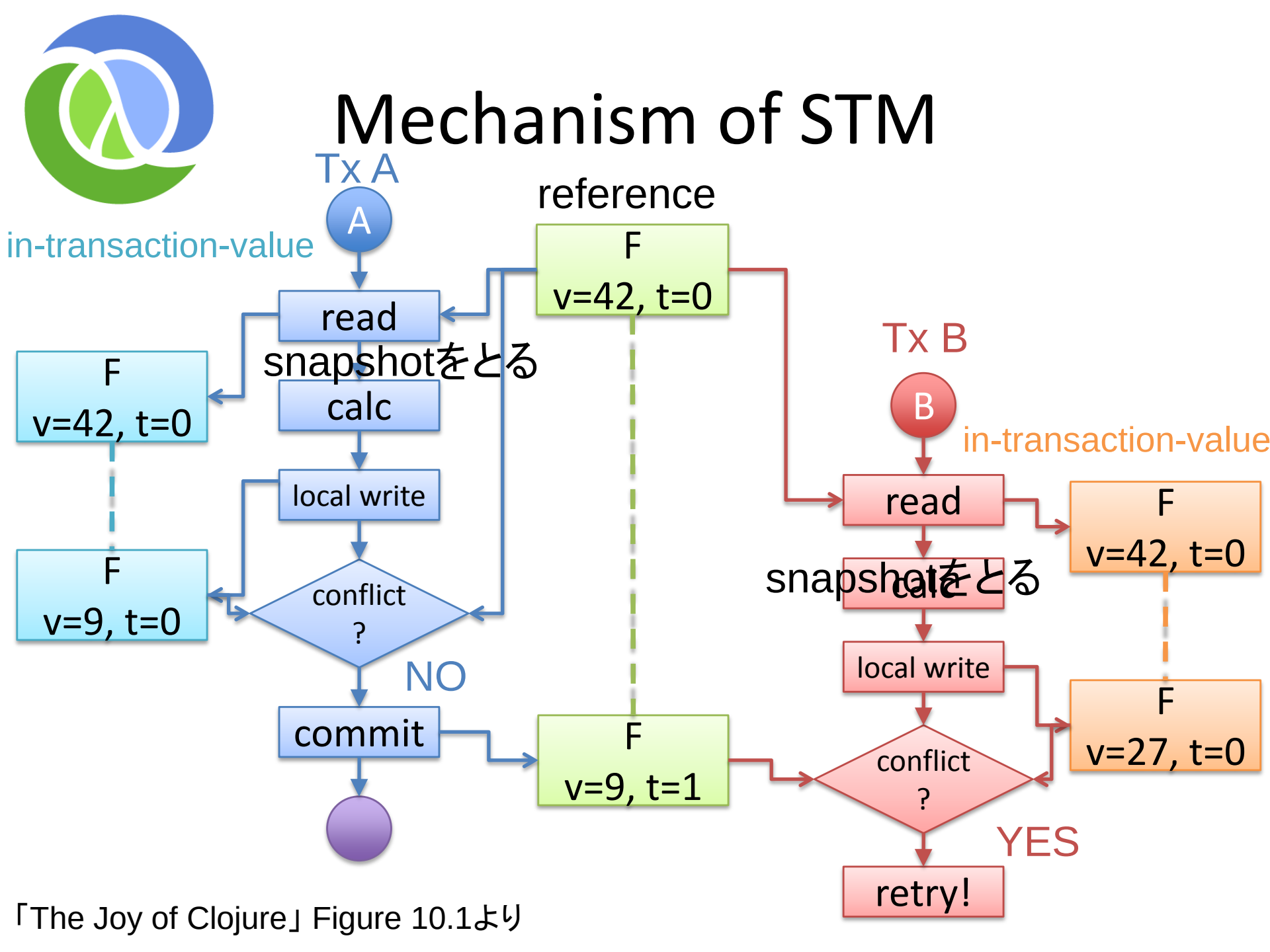
トランザクション範囲

STM

- Software Transactional Memory
- ACID特性のうちACIを提供
  - Atomic
    - トランザクション内の処理は不可分
  - Consistency
    - トランザクション内の処理の一部で失敗した場合、全体が失敗
  - Isolation
    - トランザクション内で他のトランザクションの途中結果はわからない
  - Durability
    - トランザクション内での変更は永続的である（STMはメモリ上のはなしなので、Dはサポートされない）
- 言語レベルでインメモリDBみたいなものをもっているイメージ

Implementaion of STM

- MultiVersion Concurrency Control
  - 値の変更はコピーした値に対して行い、タイムスタンプなどで楽観排他して整合性をとる
- Snapshot Isolation
  - トランザクションごとに値をコピーし、並行性を高める
  - Oracle/PostgereSQLなどのRDBMSでは、SERIALIZABLE分離レベルで使われている
- より詳しくは
  - http://d.hatena.ne.jp/marblejenka/20100626/1277528587



「The Joy of Clojure」Figure 10.1より

Concurrency

- 簡単なセマンティクスで複数の処理をatomicにできる
- デッドロックやスレッドアンセーフなバグを心配する必要がない

(他の言語だと複雑になるような)
並行処理が簡単に書ける！

Agenda

- Clojure?
- Immutable
- Concurrency
- Program as Data
- etc

Program as Data

- Clojure (というかLisp)ではプログラム自体がデータ構造
- 式を評価する前にカッコの中を好きに操作できる
  - Macro
  - カッコ遊び
- 好きに操作することでDSLを作るのも容易

Macro

dotoマクロ

```
(doto (StringBuilder.)
  (.append “Lightweight”)
  (.append “Language”)
  (.append “Tiger”)
  (.append “2010”))
```

コンパイル時に
1つ目の式の結果を一時変数に束縛し、
2つ目以降の式の2番目の要素に挿入した形で評価する

```
(let [sb (StringBuilder.)]
  (.append sb “Lightweight”)
  (.append sb “Language”)
  (.append sb “Tiger”)
  (.append sb “2010”))
```

DSL

```
(def langs '("Perl" "PHP" "Python" "Ruby" "Clojure"
"HTML5" "Scala"))

;; シーケンスから条件を満たすものをソート後、変換して返却するfrom構文
(from lang in langs                          ← langsの中から
      where (.startsWith lang "P")           ← Pから始まるものを
      orderby (count lang)                   ← 文字列長昇順で
      select (.toLowerCase lang))            ← 小文字に変換して
                                               取り出す

;; -> ("php" "perl" "python")
```

元ネタ http://www.slideshare.net/pcalcado/lisp-macros-in-20-minutes-featuring-clojure-presentation

DSL

- macro定義で実現

```
(defmacro from [var _ coll _ condition _ ordering _ desired-map]
  `(map (fn [~var] ~desired-map)
        (sort-by (fn [~var] ~ordering)
                 (filter (fn [~var] ~condition) ~coll))))
```

DSL

```
(from lang in langs where (.startsWith lang "P")
        orderby (count lang) select (.toLowerCase lang))

(defmacro from [var _ coll _ condition _ ordering _ desired-map]
  `(map (fn [~var] ~desired-map)
          (sort-by (fn [~var] ~ordering)
                   (filter (fn [~var] ~condition) ~coll))))
```

macro expand

```
(map (fn [lang] (.toLowerCase lang))
        (sort-by (fn [lang] (count lang))
                 (filter (fn [lang] (.startsWith lang "P")) langs)))
```

langsの中から

小文字に変換した結果を取り出すを

Program as Data

- プログラム自体がデータ構造なので、評価前に自由に操作できる
  - 式の途中に値を追加したり
  - テンプレートを作って埋め込むとか
- 定型処理を省ける
- DSLも簡単に作れる

Agenda

- Clojure?
- Immutable
- Concurrency
- Program as Data
- etc

Development

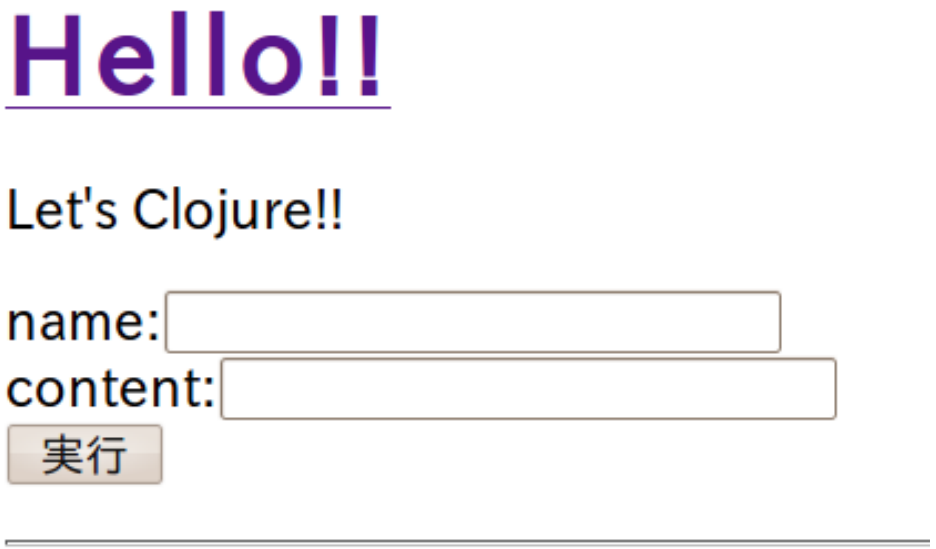
- Leiningen (http://github.com/technomancy/leiningen)
  - デファクトスタンダードなビルドツール
  - ライブラリ(jar)の依存関係解決
  - mavenベース
- Clojars (http://clojars.org/)
  - leiningenと相性の良いmavenレポジトリサイト

leiningenで開発
→Clojarsにjarをデプロイ
→leiningenでClojarsから取得して使ってもらう

Clojure on GAE

- Blankプロジェクト
  - http://github.com/making/clj-gae-blank

```
$ git clone git://github.com/making/clj-gae-blank.git
$ cd clj-gae-blank
$ lein compile
# 開発版サーバ起動
$ dev_appserver war
```



Clojure on GAE

- 逆引きClojure
  - http://rd.clojure-users.org

Clojure on Android

Activityの定義

```
(defactivity Main
  (:create (.setContentView context R$layout/main)
           (on-click (view-by-id R$id/about_button)
                     (start-activity About))
           (on-click (view-by-id R$id/public_timeline_button)
                     (start-activity PublicTimeline))))
```

http://github.com/remvee/clj-android

Othre Products

- Ring (http://github.com/mmcgrana/ring)
  - Webアプリケーションの抽象層
  - WSGI(Python)、Rack(Ruby)に相当するもの
- Compojure (http://github.com/weavejester/compojure)
  - Sinatra(Ruby)に似たWebアプリケーションフレームワーク
  - Ring上に構成
  - 逆引きClojureで採用
- Incanter (http://incanter.org/)
  - 統計/グラフ
  - R言語風プラットフォーム

Community

- Clojure-Users (http://clojure-users.org)
  - Tokyo.clj
    - 月1回の勉強会/ハッカソン
    - 次回は8/28
      - 登録はここから http://twtvite.com/tokyoclj5
  - clojure-ja (http://groups.google.co.jp/group/clojure-ja)
    - Clojureユーザー用日本語ML
- Chaton Clojure (http://practical-scheme.net/chaton/clojure/)
  - Shiroさんが答えてくれる!

Next

- http://github.com/stuarthalloway/clojure-presentations/downloads
  - Stuart HallowayによるClojure入門
  - Clojureの重要なエッセンスが網羅

ご清聴ありがとうございました