

混ぜるな危険！？ JavaとLLをマッシュアップせよ

鈴木 雄介
ITアーキテクト

ブログ: アークランプ (<http://www.arclamp.jp/>)



自己紹介


フリーランス
鈴木雄介



(株)エーティーエルシステムズ
チーフソフトウェアアーキテクト
兼 テクノロジーディレクター
稚内北星学園大学
客員助教授


雑誌: JavaWorld、DBマガジン、WEB+DB PRESS、開発の現場など
書籍: Web2.0キーワードブック
Web: @IT、SDC
講演: JavaOne Tokyo、デブサミ2006、SOAフォーラム2006

ブログ: アークランプ (<http://www.arclamp.jp/>)



狙い


LLって、なんだ？
Java環境でLLを使う必要がある？じゃ、どう使う？
そもそも、なんでLLがいるんだ？



Agenda

- LLを理解する
- LLの言語仕様
- Java環境の言語
- Java環境でのLL
- LLが求められる本質
- まとめ

LLを理解する



プログラミング言語とは何か

- プログラミング言語はソフトウェアを"表現"するための手法
- ソフトウェアを表現するのは人
 - システム開発における生産手段は人そのもの
- つまり、プログラミング言語とはソフトウェア・エンジニアにとっての道具といえる

道具としての言語を理解する 1/3

- Java
 - 静的型付け、コンパイラチェック
 - 厳格なクラス定義
- LL
 - 動的型付け、ランタイム
 - 柔軟な関数定義
- これだけでは理解したことにならない

道具としての言語を理解する 2/3

- 「技術と道具は、いかなる仕事が可能であるかを左右するだけでなく、いかに行われるかを左右する。逆に、仕事とその構造、組織、コンセプトは、技術と道具の発展に影響を与える。その影響はあまりにも大きく、**仕事とのかかわりがわからなければ、技術の発展も道具の発展も理解できない**といってよい。」
 - P.F.ドラッカー『テクノロジストの条件』/第1章/仕事と道具/P.23

道具としての言語を理解する 3/3

- では、LLによって、いかなる仕事をいかに行うようになるのか？
- また、LLという道具は、どのような仕事、構造、組織、コンセプトから生まれたのか？
- LLを使う3人の言葉を聞いてみる
 - ポール・グラハム(ハッカー)
 - DHH(RubyOnRails作者)
 - 伊藤直也(はてな)

LLとは 1人/3人

- ポール・グラハム(ハッカー)曰く
 - 「ハッカーと画家に共通することは、どちらもものを創る人間だということだ。作曲家や建築家や作家と同じように、**ハッカーと画家がやろうとしているのは、良いものを創るということだ。**」
 - 「作家や画家や建築家が、創りながら作品を理解してゆくのと同じで、**プログラマはプログラムを書きながら理解してゆくべきなんだ。**」
- CONT.

LLとは 1人/3人 CONT.

- 「この気づきは、ソフトウェアの設計に大きな意味を持つ。まず何よりも、これはプログラミング言語は柔軟でなければならないということの意味する。**プログラミング言語はプログラムを考えるためのものであって、既に考えたプログラムを書き下すためのものじゃない。**それはペンではなく鉛筆であるべきなんだ。」
- 「静的な型付けは、私が大学で教わったようにプログラムするなら良い考えだと思う。でも私の知るハッカー達はそんなふうにはプログラムしない。**我々に必要なのは、落書きしたりぼかしたり塗りつぶしたりできる言語**であって、型の紅茶茶碗を膝に置きながら厳しいコンパイラおばさんと丁寧な会話をするような言語じゃない。」
 - 「ハッカーと画家」より
 - <http://www.shiro.dreamhost.com/scheme/trans/hp-j.html>

LLとは 2人/3人

- DHH(Ruby on Rails作者)曰く
 - 「Rubyは**美しいコードを書くことができる**、プログラムをハッピーにする言語だと感じた」
 - なぜRubyを選んだのですか、という問いに答えて
 - <http://itpro.nikkeibp.co.jp/article/NEWS/20060620/241346/>より

LLとは 3人/3人

- 伊藤直也(はてな)曰く
 - 「独創的なソフトウェアを**「創りながら創る」**という方法においては**静的型付けの言語よりも動的型付けの言語の方が有利**だと思うし、スーツを着て設計書を書いてからものを作るなんて作業は、今の僕には耐え難い作業です。」
 - 「僕やはてながPerlを選ぶ理由」より
 - <http://d.hatena.ne.jp/naoya/20050518/1116425594>

13

LLを理解する

- LLは試行錯誤をしながらソフトウェアを創るために最適化された道具

14

LLの言語仕様

15

LLの言語仕様 1/3

- 試行錯誤をするためには直感的で書きやすい言語の法が良い
- 簡単なサンプルコード
 - Java vs スクリプト言語 (Groovy)
 - 1~5の配列において、各要素を足しこんでいく

16

LLの言語仕様 2/3

```
import java.util.ArrayList;
import java.util.List;

public class Sample {
    public static void main(String[] args) {
        int total = 0;
        List list = new ArrayList();
        list.add(new Integer(1));
        list.add(new Integer(2));
        list.add(new Integer(3));
        list.add(new Integer(4));
        list.add(new Integer(5));
        for (int i = 0; i < list.size(); i++) {
            total += ((Integer)list.get(i)).intValue();
        }
        System.out.println(total);
    }
}
```

18行

17

LLの言語仕様 3/3

```
def total = 0
[1..5].each() { total += it }
println total
```

そのまま読んでみよう

3行

18

JavaとLLの言語仕様

- Javaは手堅く手順を示している
- LLはデータ構造と制御を同時に表現している
 - より右脳的?
 - ぱっとみて全体を認識できる
 - 情報の加工に向いている
- LLは、思考を手順に変換する手間が省けるので直感的
 - 最初はなれないと気持ち悪いけど

19

道具としてのLLの仕様 おまけ

JavaSE5で書いてみました

```
import java.util.ArrayList;
import java.util.List;

public class SampleTiger {
    public static void main(String[] args) {
        int total = 0;
        List<Integer> list = new ArrayList<Integer> ();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        for (int it : list) {
            total += it;
        }
        System.out.println(total);
    }
}
```

ジェネリクス
オートボックス
拡張FOR文
オートアンボックス

18行

20

Java環境の言語

21

Java環境の言語

- では、Java環境で言語としてのLLは使えるのか?
- JavaとLLの仕事から見た違い
 - Java
 - 型を通じて理解できる
 - コードの可読性重視(誰が書いても同じ)
 - 同時多人数開発、IDEサポート、メンテ効率向上
 - LL
 - 俺様規約
 - 俺様コード
 - 自分主義、自分ツール、自分効率向上

22

Java環境の言語

- 単純な結論
 - エンタープライズはJava、Web2.0はLL
 - エンタープライズ・システムでLLは使えない
- 確かにLLはJavaの代替にはならない
 - Javaはエンタープライズ・システムでビジネスロジックを記述する「ゴールデン・スタンダード」
- しかし、LLはエンタープライズ・システムの中で使える「部分」(作業、レイヤー、コンポーネント)があるはず

23

Java環境の言語

- ちなみに、すでにエンタープライズ・システムではたくさんの言語を使っている
 - SQL
 - XML
 - JSP (含むHTML)
 - EL(式言語)
- LLも、この延長として捉えればよい

24

Java環境の言語: SQL

- RDBMSにおいてデータの操作や定義をおこなうための言語
- JavaではJDBC経由でSQLを流し込んで利用
 - ORMによるラッピングが進んでいるが、インピーダンス・ミスマッチの問題から、やはり大事な言語
 - JPA (EJB3.0)にもEJB-QLとして残る
- 関係を示しながらデータ操作を行うのに最適

```
select name from emp, section
where emp.secionId = sesion.id and
section.name='開発部'
```

25

Java環境の言語: XML

- 1998年2月にW3CによってXML1.0が勧告
- Javaでは、1999年ごろからデプロイ・ディスクリプタとして利用開始
 - 階層化した構成管理のため。プロパティファイルから脱却
 - その後はWebServices、SOAの流れ中でデータの記述言語として利用される
- 階層化するデータ構造を表現するのに最適

```
<pets>
  <pet type="dog">ぼち</pet >
  <pet type="cat">たま</pet >
</pets>
```

26

Java環境の言語: JSP

- 1999年にJavaServer Pages 1.0がリリース
 - Webアプリの普及に伴いHTML出力に向く仕組みが必要に
 - Servletのラップとして準備される
 - スクリプトレットからタグリブへ
 - でも、タグリブで制御を記述するのはたいへん。SWITHもELSEもない
- 構造の中に手順を埋め込むのに最適(問題あるけど)

```
<title><%= hoge %></title>
<c:forEach begin="1" end="9" var="i">
  <c:out value="${i}"/>
</c:forEach>
```

27

Java環境の言語: EL(式言語)

- 2002年、JSTL (JavaServer Pages Standard Tag Library) 1.0に含まれる形で利用開始
 - タグリブでの記述があまりにもめんどろなので利用されるようになり標準仕様
 - いわゆる簡易なスクリプト言語(インタプリタ型)
- オブジェクトの参照や、ちょっとした演算に最適

```
<c:out value="${mymodel.title}"/>
<c:out value="${mylist[0] + mymap['key'] }"/>
```

28

分業

- そもそも、なぜこんなにもたくさんの言語があるのか?
- 専門化にあわせて道具を特化させることで生産性を高めてきた
 - SQLのJava化は成功しなかった
 - マルチ言語化は産業としての当然の進歩
- 分業は「労働の生産性の飛躍的な向上」の源泉
 - アダム・スミス/国富論(1776年)

29

協業

- ただし分業にも限界がある
 - 「作業の細分化と専門化が進むとともに、それらを統合するのがむずかしくなり、コスト高になってきた。<中略>どこかの段階で、**統合コストが超専門化の利点を上回る**ようになる。」
 - トフラー/『富の未来』/第4章/基礎的条件の深部/P.69
- いかに統合コストを上げずに作業を分離するかが重要
 - ヒント: インターフェース指向。SOAの基本的なアイデア

30

Java環境でのLL

31

Java環境でのLL

- では、スクリプト言語の特性を考えると、どの部分に使えるのだろうか？
- 僕の考え
 - アウターゼリー
 - アプリケーションの外側で使う。自動生成やバッチ処理
 - **インナーゼリー**
 - **アプリケーションの内側で使う。以下参照**
 - スーパーグラー
 - アプリケーションを結合する。

32

インナーゼリー

- Webアプリケーションのプレゼンテーション層に使う
 - 分業のキーワード
 - プレーンHTML+スクリプト言語+Java
 - 協業のキーワード
 - インターフェース(SOA的)
 - DI

33

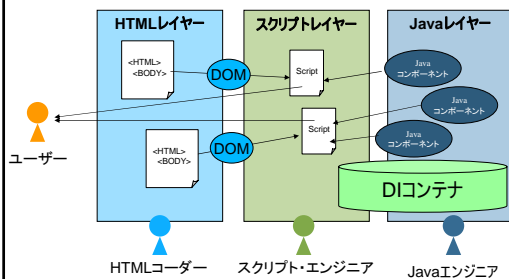
Sarugau JS

- Sarugau JS (さるごー じえいえす)
 - <http://www.sarugau.org/>
 - 2006年4月より公開
 - オープンソース(CC帰属2.1日本)
- サーバーサイドのDHTMLエンジン
- 非定型的でデザインが重視されるような部分での利用を想定。
 - デザイナやHTMLコーダーといかに協業するか

34

Sarugau JS

- スクリプトレイヤーがデータとビューを結びつける



35

Sarugau JS

- 現在は、JavaScript版を公開中
 - ちょっと使いにくい
 - JavaのJavaScriptライブラリ「Rhino」が、Javaとの統合を、それほど強く意識してない
 - JavaSE6に標準で搭載されるスクリプトエンジン実装
- Groovy版を開発中
 - <http://groovy.codehaus.org/>
 - Javaとの統合が意識された言語
 - JSR241 (The Groovy Programming Language)として標準策定作業中

36

Sarugau JS: サンプル

- サンプル
 - HTMLのタイトルを書き換えて、リストを表示する
- 用意するもの
 - プュアなHTML
 - DAO (Javaのデータアクセスオブジェクト)
 - 設定ファイル (DIコンテナ)
 - スクリプト (Groovy)

37

HTML

- プレーンなHTML。CSSのID属性をつける

```
<html>
<body>
  <ul id="list">
    <li>名前</li>
    <li>名前</li>
    <li>名前</li>
    <li>名前</li>
  </ul>
  <div>TOTAL:<span id="total">NN</span></div>
</body>
</html>
```



赤いところを動的に出力

38

DAO

- MyDAOとMyModel
 - DAOのモック実装
 - メソッドfindAllで、MyModelの配列を返す

```
public class MyDaoImpl implements MyDao {
  public List findAll() {
    List l = new ArrayList();
    l.add(new MyModel("Foo", 1));
    l.add(new MyModel("Bar", 2));
    l.add(new MyModel("ほげ", 3));
    l.add(new MyModel("たろう", 4));
    l.add(new MyModel("はなこ", 5));
    return l;
  }
}

public class MyModel {
  private String name;
  private int value;
  private Date create;
}
```

39

設定

- Groovyから参照できるように設定
 - Springframework

```
<bean id="groovyScriptRepository"
  class="...GroovyScriptRepositoryImpl">
  <property name="propertyMap">
    <map>
      <entry key="services">
        <map>
          <entry key="myDao">
            <bean class="...MyDaoImpl" />
          </entry>
        </map>
      </entry>
    </map>
  </property>
</bean>
```

40

スクリプト

```
import org.sarugau.js.script.groovy.builder.TagBuilder

def people = services.myDao.findAll() //DAOの呼び出し
def total = 0;
TagBuilder builder = new TagBuilder(document)
builder.writeById('list'){
  people.each() { v ->
    li(v.name+' : '+v.value) //LIタグの出力
    total += v.value //足しこみ
  }
}
builder.writeById('total'){
  textnode(total) //合計値の出力
}
```



41

スクリプト: サービスの呼び出し

- サービスの呼び出し
 - services.{登録名}.{メソッド名}
- スクリプトのグローバル変数として呼び出せる
- DIコンテナへのルックアップをしているようなもの
 - DIの機能はすべて利用可能

42

スクリプト:ビルダー

- TagBuilder
 - HTMLのタグを書き出すためのビルダー。タグと同じように階層化して記述可能

```
TagBuilder.writeById('タグ名') {
  内側のタグ([属性キー:値,...],テキスト) {
    内側のタグ([属性キー:値,...],テキスト) {}
  }
}
```
 - 内部にクロージャや制御処理を書くことができる

```
TagBuilder.タグ名() {
  [1,2,3,4,5].each() {内側のタグ(...){}}
}
```

43

Sarugau JSを使った感想

- Javaとスクリプトで作業が分離
 - 徹底したインターフェース指向にならざるをえない
 - 粒度設計や柔軟性など課題はSOAと同じ
- HTMLとスクリプトで作業が分離
 - HTMLを完全にデザイナーとHTMLコーダーにまかせられる
 - CSSのID属性は彼らの世界の規約
 - 今後はインフォメーション・アーキテクチャとの連携が課題
 - HTMLコーダー+スクリプトが書けるような人材なら、両方を記述可能

44

Sarugau JSを使った感想

- スクリプト部分のテスト
 - もっと単体テストを導入すればよさそう。そもそも、テストできないほど複雑なコードを書かないことが大事。Javaに押し付けてしまえばいい
 - IDEサポートは2008年ぐらい?
- パフォーマンス
 - ほとんど気にならない
 - キャッシュをうまく使えばいい。要は使いどころ

45

LLが求められる本質

46

なぜLLなのか

- LLを使う人はちゃんと設計しないのか?
 - そんなことはない。もちろん設計は重要
- 設計の幻想に対する回答
- プログラミングは「設計を元に製造すればよい」というものではない

47

近代工業の考え方

- 近代工業の組立ラインにおける生産性
 - 設計を元に、品質よく/コストを少なく/期限に間に合うように製造すること(量の生産性)
 - QCDの追求
 - 設計は事前の開発の結果うまれてきたもの
- テイラーの科学管理法(1911年)
 - 仕事を個々の動作を分解し、時間計測と改善を繰り返す
 - いかにも無駄なく作るか
 - 生産のプロセス化
 - トヨタ生産方式の原型

48

ソフトウェア開発の生産性 1/2

- ソフトウェアは工業製品ではない
 - 量産コストはゼロ。組立ラインとは違う
- 生産性を上げる方法はまったく違う
 - ドラッカーの示した「知識労働の生産性を向上させる六つの条件」
 1. 仕事の目的を考える。
 2. 働く者自身が生産性向上の責任を担う
 3. 継続してイノベーションを行う
 4. みずから継続して学び、人に教える
 5. 知識労働の生産性は量よりも質の問題であることを認識する
 6. 知識労働者は、組織にとってコストではなく資本財であることを理解する

49

ソフトウェア開発の生産性 2/2

- 「六番目の条件以外は、**肉体労働の生産性向上の条件とはちよど逆**である。＜中略＞肉体労働では、質は制約に過ぎない。最低の基準があるだけである。＜中略＞**知識労働における仕事の質は制約どころではない。仕事の本質である。**」
- 「教師の仕事は、生徒の数では評価されない。何人の生徒が本当に学んだかが問題である。」
- 「知識労働の生産性は質を中心に据えなければならない。しかも最低を基準にはしていない。最高ではないにしても最適を基準にしなければならない。量の問題を考えるのはその後である」
 - P.F.ドラッカー／『テクノロジストの条件』／第5章／知識労働の生産性／P.80

50

芸術における定義との近さ

- むしろ芸術に近い
- 美しさは、設計ではなく創られたものに現れる
 - 「制作時にどれほどの見通しをもっていても、真に美しい作品を生み出したときには、真っ先に作者がその美にうたれるはず。人間の持ち分は、設計図を書いてそれを具現化することです。それ以上のことはできません。**美は、その仕事に対して与えられる恵み**なのです。恵みとして与えられる美は、その仕事が単に設計図を巧みに具現化したというだけでなく、真に『よい』ものであることを示しているのです。」
 - 佐々木健一／『美学への招待』／P.210より
 - 「ハッカーと画家」の共通点

51

LLが望まれる理由の深部

- 重要なのは出来上がるものの質
- 車で言えば新車開発
 - 車の開発者の仕事は、部品の量では評価されない。何台売れたかが問題である
- ソフトウェア開発も質を重視すべき
 - ソフトウェア・エンジニアの仕事は、コードの量では評価されない。どれだけビジネス価値に貢献したかが問題である
- そのためには**試行錯誤が絶対**に必要
 - 新車開発なら、モック、クレイモデル、CAD、シミュレーター
- もちろん、言語だけではなく開発手法そのものにも

52

まとめ

まとめ 1/3

- LLの言語仕様は極端だとしても、LLが必要とされる本質は真摯に考えるべき
 - 質を上げるための試行錯誤は重要
 - まだ、歴史が浅くてノウハウが蓄積されていない
- 「試行錯誤」を、いかにエンタープライズ・システムに取り入れるのか
 - 言語としても、開発手法としても、JavaとLLの融合
 - たとえばアジャイル

54

まとめ 2/3

- システム開発の再構築
 - 生産性のパラドックス
 - 「電球が発明されたのは1879年だが、電化が始まり、経済や生産性に大きな影響をあたえるようになるまでに、数十年の時を要したというのだ。なぜか？電動機を取り付けて古いテクノロジー（蒸気機関）を廃棄するだけではだめだからだ。製造の手順すべてを再構築しなければならない。」
 - トーマス・フリードマン／フラット化する世界（上）／第3章／三重の収束／P.293
 - 「旧大陸から新大陸への移動」
 - 丸山先生
- LLは、その1つの表れ
 - Web2.0サービスでの運用に対する考え方、スケールアウト戦略

55

まとめ 3/3

- マッシュアップ
 - (特にAPIが公開された)ウェブサービス同士を(できることならば)開発者ですら思いもよらない方法で組み合わせて、新たな価値を創造すること。
 - はてなキーワードより
 - <http://d.hatena.ne.jp/keyword/%A5%DE%A5%C3%A5%B7%A5%E5%A5%A2%A5%C3%A5%D7>
- あなたは新しい価値を創造することができる
 - ぜひトライを！

56

arclamp.jp

混ぜるな危険！？ JavaとLLをマッシュアップせよ

鈴木 雄介
ITアーキテクト

ブログ: アークランプ (<http://www.arclamp.jp/>)

57