プログラム言語論

亀山幸義

筑波大学 情報科学類

スクリプト言語、DSL

スクリプト言語

Scripting Language (Language for Scripting)
歴史的には:

- スクリプト＝台本
- Computer Science では，(アプリケーションソフトウェアに対する)「指示や命令の列」
- スクリプト言語は，コマンドラインで実行されるような簡易な命令を記述できる言語で，インタープリタで実行されるものを意味することが多かった．
- 本格的なプログラム言語の対義語．
- 代表例: Unix の shell (shell のプログラムを shell script と言う)
- その他: Unix の awk, sed, ...

現在:

- PHP, Perl, Python, Ruby, JavaScript ... など多数の汎用の言語。
- スクリプト言語とは一体何だろう?

スクリプト言語

現在では,「スクリプト言語とは何か？」「この言語はスクリプト言語か否か」という問に正確に答えるのは難しい．その代わりに...

- 動的言語 (dynamic language): 通常は静的 (コンパイル時) に行われること (たとえば，関数やクラスの定義，型付けなど) の多くを動的に (実行時に) 行う言語.「この言語は動的言語か」を明確に決めることはできないが,「Ruby は Java に比べて動的である」ということは言える．
- 動的型付け言語: 型付けが静的に行われないで，実行時に行われる言語．(1+"abc" は実行時エラー)
- 領域特化言語 (domain specific language, DSL): 特定の領域 (domain) の問題を解くために適した (簡潔であり，汎用ではない) 言語．(例: Java で turtle graphics をするプログラムを書いたとき，そのプログラムのユーザは,「右へ 3cm 行け」「左に 90 度回転せよ」といった命令を発行して graphics を行う．後者の命令群から構成される言語が,この際の DSL である．)

スクリプト言語 (続き)

「スクリプト言語」の説明：

- プログラムを「すぐ書ける」言語 (生産性の高い言語, プロトタイピングに適した言語)
- インタープリタで動かすことが多い言語。
- 実行性能よりも、書きやすさ (速く書けること) を重視。

→ ??? (この説明でわかるわけがない。)

プログラム言語: 汎用 vs 特化

この授業で取り上げてきた言語はほとんどすべて汎用 (general purpose) プログラム言語

ドメイン特化言語 (Domain Specific Language, DSL)
特定の問題領域 (数値計算、数式処理、データベース利用，推論、グラフィクス etc.) に特化した言語。

汎用言語の役割と DSL の役割は相補的 (どちらか一方だけでは、すべてのニーズに対応できない)。

お勧めの読みもの: "Purpose-Built Languages", ACM Queue, Mike Shapiro (インターネット上で無料閲覧可能)。
`http://queue.acm.org/detail.cfm?id=1508217`

ドメイン特化言語

例

データベースを操る言語: SQL

数式処理は大得意な言語: Mathematica

パーサ (構文解析器) を作ってくれるシステム: yacc (言語は BNF 程度)

電卓の言語: 整数、+, −, ···

ロボットの関節を動かす言語: ···

···

実は、色々な場面に応じて、数限りない「言語」が作り出されて、使われている。

ドメイン特化言語の実現

2 段階の実現方法:

1 つ 1 つの DSL に対して、素晴しいコンパイラを作るわけには (なかなか) 行かない。

汎用言語で、DSL 処理系 (インタープリタ) を記述する。

ユーザの書いた (DSL 語による) プログラムは、その処理系が実行する。

DSL の 2 つのタイプ:

外部 DSL (external DSL): ホスト言語 (汎用言語) と別の構文

内部 DSL (embedded DSL): ホスト言語の中に埋めこまれる

大きな問題:

DSL 処理系をどうやって (手軽に) 書くか。

DSL 処理系の実行性能をどうやって (手軽に) あげるか。

ドメイン特化言語の実現

DSL 処理系の実行性能 (Abstraction Overhead の削減)

メタプログラミング・アプローチ

メタプログラミングとマルチステージ言語

メタプログラム=プログラムを作るプログラム

```
make_mul 5 ==>  (引数を 5 回かけるプログラム)
```

方法 0: 文字列として生成して、それを実行
方法 1: Scheme における eval 命令

```
(define (make_mul_sub n var)
   (if (= n 0) 1
      `(* ,var ,(make_mul_sub (- n 1) var))))
(define (make_mul n)
   `(lambda (x) ,@(make_mul_sub n 'x)))
((eval (make_mul 5)) 2)
```


メタプログラミングとマルチステージ言語

方法 2: マルチステージ言語 MetaOCaml

```
let rec power1 n var =
   if n=0 then .<1>.
   else   .< .~var * .~(power1 (n-1) var)>.
let power n =
   .<fun x -> .~(power1 n .<x>.)>.
(power 5)
=> .<fun x -> (x * (x * (x * (x * (x * 1)))))>.
(.! (power 5)) 2
=> 32
```

生成するプログラムだけでなく、生成されたプログラムも型の整合性が静的に (生成前に) 保証される。

インタープリタとコンパイラの違い (3)

power 関数のステージ化:

- 第 1 ステージ: n をもらって、x^n を計算するコードを生成。
- 第 2 ステージ: x^n のコードと x の値をもらって、コード実行。

コンパイラ==インタープリタをステージ化したもの

- 第 1 ステージ (コード生成): プログラムをもらって、そのプログラムのコードを生成。
- 第 2 ステージ (コード実行): コードと、プログラムに対する入力をもらって、コードを実行。

MSP による DSL 処理系の高速化

Walid Taha, Gentle Introduction to Multi-Stage Programming
実現されている MSP 言語:

- MetaOCaml
- Scala light-weight modular staging