プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 9

Java と Ruby

Java

- C/C++言語に似た構文
- ブロック構造言語; 変数は静的束縛
- 静的な型システム、型安全性
- 信頼性、安全性を重視

Ruby

- スクリプト言語
- 動的言語 (静的型付けをしない)
- クラス定義は宣言ではなく実行文
- 開発効率を重視

共通すること: オブジェクト指向であること (動的ルックアップ、情報隠蔽、継承、サブタイピング)

静的 vs 動的

1. 宣言された変数はすべて，式の中にあらわれるかどうか.
2. 宣言された変数はすべて，プログラム中で使われるかどうか.
3. 実行中に使用されるスタックの量.
4. MiniML 言語のプログラム (ペアを表すデータ構造を持つ) に対して，実行中に使用されるヒープの量.
5. MiniC 言語や MiniML 言語のプログラムに対して，プログラムの型が整合的であるかどうか.

C, ML では、1,5 は静的に決められる。2,3,4 は決められない (実行時の情報)。

静的 vs 動的

コンパイラとインタープリタの差 (再訪問)

- 多くのコンパイラは、プログラム全体 (あるいはプログラムの 1 つのモジュール全体) を読みこみ、可能な限り多くの静的情報を取得して、それを生かした効率良いコードを生成する。
- 多くのインタープリタは、そのようなことをしない。

静的情報を使って効率がよくなる例:

- 局所変数が、スタックフレーム中のどこにあるかを解析すれば、実行時には、「局所変数がどこにあるかを探す」必要がなくなる。
- 「a+b」という式で、a,b が整数型とわかっていれば、整数型かどうかのチェックはいらず、いきなり機械語の「加算」命令を使える。

現在の Optimizing Compilter は、上記以外にも様々な効率化をはかっている。

静的型付け vs 動的型付け

- Java, ML, C は静的型付け: コンパイル時にプログラムの型の整合性を検査 (あるいは推論) する。実行時には (ほとんど) 型の整合性を検査しないですむ。
- Ruby, Perl, Python, Lisp, Scheme などは、動的型付け: コンパイル時には型の整合性はチェックしない。実行時に型が整合しない演算を行えば、エラーを発生させる。

これらの帰結:

- 前者は、より高い信頼性を得やすい。
- 後者は、より柔軟なプログラミングが可能。

今日の演習の課題

- Java のサブタイピングと、動的ルックアップの基礎を理解する。
- Ruby を Java と比較して、動的型付け言語について考察する。

課題. 授業ホームページの指示にしたがって，Java と Ruby のサンプルプログラムを走らせ，継承，サブタイピング，動的ルックアップ，および Java の overload/override について理解を深めなさい.
レポートは次週まで。