3H－3

ネットワーク対応オブジェクト指向Lispの処理系

小池 龍信　船戸 潤一　中西 正和
慶應義塾大学大学院 理工学研究科 計算機科学専攻

はじめに

Javaの登場が更にインターネットの普及を爆発的に促進している。その重要なファクタとして、アーキテクチャに依存しないというJavaの特徴が挙げられる。これは計算機のハードウェアやOSの制約にとらわれずに、共通のプログラムを動作させることができる、という環境である。本研究では、Javaインタプリタ上で実行可能なLispシステムであり、ネットワーク対応オブジェクト指向Lispである、**Java lisp** (jlisp) を実装し、その仕様及び、有効性について考察する。

1　jlisp言語仕様

オブジェクト指向Lispであるjlispのプログラムは、クラス定義の集合で構成される。jlispにおけるクラスは、内部状態を表すデータと、そのデータを操作するメソッドの集まりである。また、jlispで扱える基本データタイプは通常のLisp同様に、数値アトム、シンボルアトム、2進木リストであり、スコープはレキシカルスコープを採用している。

1.1　クラスの定義

クラスはスペシャルフォーム **defclass** により以下のような構文で定義される。

```
(defclass ClassName (SuperClass)
  ((var1 init1) (var2 init2) …)
  ((defmethod MethodName1 …)
   (defmethod MethodName2 …) …))
```

- ClassName:クラスの名前。このクラスを定義したファイル名は、クラス名+.lspとする。
- (SuperClass):直接継承するクラスのリスト。ただしjlispは単一継承であるため、リストの要素は1つである。
- (var*n* init*n*):インスタンス変数名とその初期値のリスト。初期値は定数に限らず一般のLisp式でも与えられる。
- (defmethod MethodName*n* …):スペシャルフォーム **defmethod** によってメソッドを定義する。定義方法は通常のLispの関数定義と同様である。

1.2　メッセージ送信のためのスペシャルフォーム

カプセル化されたインスタンス間でのメッセージ送信のために、以下のスペシャルフォームを用意している。

- **slot-value**:インスタンス変数のアクセス。
  `(slot-value instance variable)`
- **method**:インスタンスメソッドの起動。
  `(method instance MethodName arg1 …)`
- **make-instance**:インスタンスの生成。
  `(make-instance ClassName)`

1.3　メインメソッドとlispクラス

アプリケーションプログラムの実行を開始する**lispmain**メソッドを、**lisp**クラスの中に1つ定義しなければならない。これはC言語のmain()関数、Javaのmain(String argv[])メソッドに相当するものである。lispクラスは、lispmainメソッドが定義されているという点以外、他のjlispのクラスと相違点はない。

2.　システム構成

jlispはその言語仕様に基づいて作成されたソースファイルから、jlispコンパイラによってオブジェクトコードを生成する。jlispコンパイラの構成は以下のとおりである。

1. **バイトコード変換フェーズ**
   オブジェクトの動作が記述されたLispソースコードを、Javaインタプリタによって解釈、実行されるJava byte code[1]に変換する。
2. **クラスファイルの生成フェーズ**
   Javaクラスファイルフォーマット[1]に従って、バイトコードを含んだクラスファイルを生成する。

Lisp On Java

Tatsunobu KOIKE Junichi FUNATO Masakazu NAKANISHI
Department of Computer Science, Faculty of Science and Technology, Keio University 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223, Japan

3. jlisp の実現方法

Lisp で記述されたソースコードを Java インタプリタ上で実行するには、その Lisp 式の振舞いを忠実に Java のバイトコードで表現しなければならない。

3.1 jlisp のデータ構造

Lisp ではどのようなデータでも動的に扱うことが許される一方、Java ではデータの型宣言とその整合性が厳しく要求され、変数の動的扱いは許されていない。そこで、**Lispobj** クラスを Java で用意し、jlisp におけるすべてのデータはそのクラスのサブクラス (以下参照) のインスタンスとして扱うものとする。

- **Atom クラス**
- **Symbol クラス**
- **Specialsym クラス**: t、nil を表すオブジェクト
- **Inumber クラス**: 整数値を表すオブジェクト
- **Dnumber クラス**: 実数値を表すオブジェクト
- **Cell クラス**
- **Lisp_Lang_Object クラス**: ユーザクラスの親クラス

これらの継承関係は図 1 のとおりである。これにより、すべての入力データは内部的には (バイトコード上では)Lispobj 型と宣言されたものとして扱う。

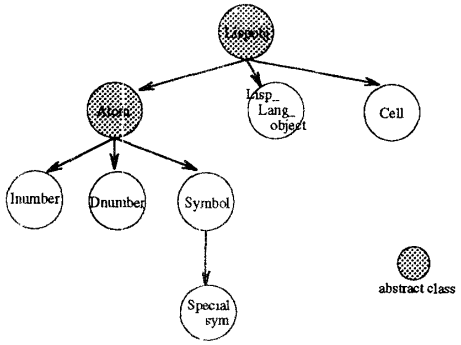

図 1: データ構造の継承関係

3.2 データオブジェクトを生成、操作するクラス

jlisp のすべてのデータを 3.1 節のクラスのインスタンスとして扱うために、入力データからオブジェクトを生成したり、データオブジェクト同士の演算を行なう手続きが必要となる。

Changetoobject クラス

このクラスには、数値アトム、シンボルアトムやリストなどの入力データから、それぞれに対応した 3.1 節のクラスのインスタンスを生成するメソッドが予め定義されている。

Function クラス

このクラスには、データオブジェクトの算術演算やリスト処理を行なうメソッドが予め定義されている。

3.1 節の jlisp のデータ構造を実現するクラスと合わせて、必要に応じてこれらのメソッド呼び出しを jlisp コンパイラがバイトコードで記述することにより、Lisp 式を Java バイトコードで表現できる。

4. 結論及び今後の展望

4.1 結論

本研究では、以下のような特徴をもつ Lisp システムを実現した。

- クラスの定義のみで構成される、というオブジェクト指向のモデルをそのまま実現している。
- jlisp プログラミングにおいては、リスト構造を扱うことができ、複雑なデータ構造を扱えるという Lisp の利点をそのまま反映することができる一方、その実行においては、OS の制約を受けずアーキテクチャに依存しない、という Java の利点をも兼ね備えている。故にネットワーク上で共通の Lisp プログラムの実行を可能にする。
- jlisp が生成するクラスファイルは、Java のそれと同一の仕様をもつため、jlisp、Java 間で相互に再利用性の向上をはかることができ、ソフトウェアの開発、管理を jlisp、Java の両言語で行なうことが可能である。よって、Lisp ユーザと Java ユーザの共同開発を促進し、オブジェクト指向プログラミングの可能性を更に広げる。

4.2 今後の展望

- jlisp は単一継承を採用しているため、複数のクラスを直接継承できない。そこでより多くのメソッドをクラスが継承できるようなインターフェイス機能を提供することで、再利用性の向上が計られると予想される。
- 生成したバイトコードに対し、最適化を施すことにより、実行の効率化が期待される。

参考文献

[1] Tim Lindholm, Frank Yellin. *The Java Virtual Machine Specification*, *ADDISON-WESLEY*, 1996.

[2] 井田昌之, 元吉文男, 大久保清貴. *bit* 別冊 *Common Lisp* オブジェクトシステム, 共立出版, 1989.

[3] 上野亜紀子. オブジェクト指向 *Lisp* の設計および実装, 慶應義塾大学 学士論文, 1995.