愛知教育大学教育実践総合センター紀要
第3号, pp. 83～ 87 (March, 2000)

Java 言語による Lisp 言語処理系の実装

安 本 太 一
(愛知教育大学数理科学選修)

Implementation of a Lisp system written in Java

Taichi YASUMOTO
(Department of Mathematical Sciences, Aichi University of Education)

要約: マルチプラットホーム向けLisp言語処理系を実装した。この処理系は，Java言語で記述されており，コンピュータのハードウェアやオペレーティングシステムを問わずに動作する。WWWブラウザ上でアプレットとして動作させることも可能である。学校のコンピュータ室のように多数のコンピュータで使用する場合は，WWWサーバ上に処理系のファイルを配置しておくだけでよく，個々のパソコンにインストールする労力は不要である。Java言語で記述した処理系の実行効率を向上させるため，Lisp言語用バイトコードインタプリタによるコンパイラ方式の試みや，非局所脱出の2通りの実装方法（例外処理とスレッド）の比較を行った。

キーワード: プログラミング教育，Lisp言語，コンピュータの活用

1 はじめに

プログラミングは，例えば，コンピュータの活用という形で，高等学校の数学Cのなかでとりあげられている。しかし，高等学校の数学Cの教科書をみる限り，使用されているプログラミング言語は古典的なBASIC言語であり，構造化や抽象化といった近代的な概念とは無縁なものになっている。これらの概念は，数学的な見方や考え方と関連しているのにもかかわらず，コンピュータ活用の段階になって無視されているといっても過言ではない。本稿では，Lisp言語の処理系の実装について報告する。Lisp言語は，数学的な裏付けを有するプログラミング言語である。報告する処理系は，コンピュータのハードウェアやオペレーティングシステムを問わずに動作し，インストールのコストも低いので学校の現場に適している。まず，次章でプログラミング教育の必要性とLisp言語採用の提案について述べ，3章で学校の現場を考慮したLisp言語処理系の設計方針を述べる。4章で処理系を実装するにあたって留意した点について述べ，5章でこれらの点に焦点をあてて実装した処理系を評価する。

2 プログラミング教育とLisp言語

「幼稚園，小学校，中学校，高等学校，盲学校，聾学校及び養護学校の教育課程の基準の改善について」(教育課程審議会平成10年7月29日答申)において，情報化への対応が明示されたように，近年，学校においてコンピュータ活用技術の教育が推進されてきた。この答申において，コンピュータや情報通信ネットワーク等を含め情報手段を活用できる基礎的な資質や能力を培う必要性が述べられている。コンピュータ活用技術の円滑な修得や今後多様化するハードウェア・ソフトウェアへの対応を考えると，コンピュータの表面的な使い方だけでは不足であり，論理的な思考が欠かせない。答申とほぼ同時期に開催された「衛星通信を活用した教育情報ネットワークの在り方に関する調査研究協力者会議」(平成10年6月)においても，21世紀の学力の基礎・基本のひとつとして，論理・数理を掲げている [1,2]。

論理的な思考のトレーニングとして，プログラミングを採用することが望ましいように思われる。プログラムに論理的な誤りがあると正しく動作しないことを経験すると，コンピュータは論理の積み重ねの上にたって動作していていることが理解できるからである。

プログラミング言語にはBASIC言語やC言語などさまざまなものがあるが，教育向けに適するものとしては次の用件を満たすことが必要である。

1. 構文が簡潔であること。
   論理的な思考のトレーニングが目的であるので，プログラミング言語の理解に時間を要するのは不適である。
2. インタプリタ方式の実行が可能なこと。
   対話的なプログラミングにより，プログラムの高い生産性を得ることは，生徒への負担軽減や授業時間の制約という観点から重要である。

学校教育の現場ではインタプリタ方式の言語としてBASIC言語が広く用いられてきたが，構文の簡潔さを考えるとLisp言語の方が優位である。Lisp言語のプログラムを構成する式（フォーム）は，リストフォーム，変数，データの3種類に大別されるという簡潔さは，BASIC言語は有していない。Lisp言語処理系は，フォームを読んで，それを評価し，結果を表示するという動作をするので，入出力文を知らなくてもある程度プログラミングが可能である。プログラミングの教科

書には，Lisp言語を採用しているが，Lisp言語の説明にページ数をさいていないものも実際に存在する[3]。

3 Lisp言語処理系作成の基本方針

今回Lisp言語処理系を作成するにあたって，次の点に留意した。

1.特定のプラットフォームに依存せずに動作すること。
学校や教室ごとに，さまざまなメーカーのコンピュータが整備されている。また，コンピュータに搭載されているオペレーティングシステムの種類も様々である。

2.インストールが，オペレーティングシステムや既にインストールされているソフトウェアに，副作用を及ぼさないこと。
アプリケーションソフトウェアによっては，インストールの際に，オペレーティングシステムの設定を変更したり，ライブラリを置き換えてしまうものがある。その結果，オペレーティングシステムの動作が不安定になったり，既にインストールされているアプリケーションソフトウェアが動作しなくなる場合があるが，このような事態は避けなければならない。

3.インストールが容易なこと。
学校に配備されている多数のコンピュータの管理は，教員の大きな負担となっている。多数のコンピュータがあることを考えると，インストールの作業は可能な限り簡潔であることが望まれる。また，バージョンアップ等に伴う更新作業が簡潔であることが望ましいのはいうまでもない。

以上の点を勘案した結果，Java言語[4]を用いて，Lisp言語処理系を実装することにした。

Java言語で記述されたプログラムのバイトコードは，JVM（Java Virtual Machine）が実装されていれば，ハードウェアやオペレーティングシステムを問わずに実行可能である。GUIやネットワークを使用するプログラムであっても，構わない。Windows95／98，MacOS，Windows3.1，OS／2およびさまざまなUNIX（Linuxを含む）において，JVMが実装されていることから，上記1.の条件を満たす。Java言語で記述されたプログラムの実行形式であるバイトコードは，プラットフォームに依存しないものなので，オペレーティングシステムやネイティブアプリケーション（プラットフォーム固有の機械語やライブラリによって動作するアプリケーション）に影響を与える可能性そのものが全くない。むしろ，JVMのインストールが，オペレーティングシステムやネイティブアプリケーションに副作用を及ぼす可能性がある。しかしながら，最近では，オペレーティングシステムのインストール時にJVMがインストールされるようになっていることや，WWWブラウザにJVMが組み込まれている現状を考えると，JVMのインストールはあまり問題にならないと考えれるので，上記2.の条件を満たす。

Java言語で開発されたプログラムは，JVMを内蔵したWWWブラウザによって実行可能である。このような形式で実行するプログラムをアプレットという。図1に示すように，WWWサーバー上にLisp言語処理系のアプレットを配置し，ブラウザ上で実行することが可能である。個々のパソコンに処理系をインストールする必要はなく，WWWブラウザでアプレットを起動するURLを指定するだけで実行が可能である。バージョンアップは，サーバー上のアプレットを更新するだけでよい。Lisp言語処理系をスタンドアロンで実行する必要があるときは，処理系を構成する複数のクラスファイル（クラスに対応するJavaバイトコードからなるファイル）をjarアーカイブファイルとして1つにまとめてしまえば，インストールは1つのファイルをコピーするだけで完了する。したがって，いずれの実行形態であっても，上記3.の条件を満たす。

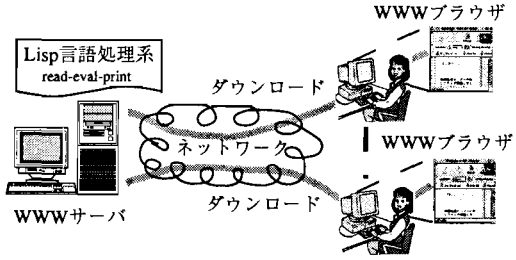

図1:WWWブラウザ上で動作するLisp言語処理系

なお，実装するLisp言語処理系の言語仕様は，国際標準のISLisp[5]に準拠するものとした。教育向けということを考えるとScheme[6]という選択肢もあるが，ISLispもその言語仕様がScheme並にコンパクトであることと，JISによる標準化がなされている現状[7]を踏まえて，ISLispを選択することにした。

4 実装

Lisp言語処理系実装の一般的な話は多くの文献で述べられているので，Java言語を用いた実装に固有なことに焦点をあてて述べる。

4.1 データ表現

Lisp言語は，動的な型付けを行う。C言語による典型的な実装では，Lisp言語の各データ型オブジェクトに対応する構造体を包括する共用体を定義し，変数はこの共用体によって実現する。Java言語による今回の実装では，あるクラスをスーパークラスとして，Lisp言語の各データ型オブジェクトはこのスーパークラスのサブクラスとした。整数，浮動小数点数，文字列のオブジェクトは，Java言語に組込み型のint,

double, Stringをそのまま利用するという実装も考えられるが，コンス（cons）などの他のLispデータ型のオブジェクトと扱いが異なってしまい，処理系の実装として統一がとれないので採用しなかった。

4.2 ごみ集め

Lisp言語処理系では，各オブジェクトに対して動的なメモリ割当てを行っている。処理系が使用できるメモリには限りがあるので，参照されなくなったオブジェクトが使用していたメモリ領域を回収して再利用しなければならない。Lisp言語処理系では，そのようなメモリ管理を自動的に行う“ごみ集め”という機能を装備することになっている。

JVMにはごみ集めの機能が装備されている。Java言語のクラス定義として実現された各データ型のオブジェクトは，このごみ集めの管理下におかれるので，Lisp言語オブジェクトのためのごみ集めを実装する必要がない。このことは，Lisp言語処理系の実装コストの低減だけではなく，Lisp言語で記述されたプログラムからGUIやネットワーク等のライブラリの利用が容易になったことを意味する。従来，C言語等のごみ集めが標準装備されていない環境では，これらのライブラリが使用しているメモリ管理とLisp言語処理系のメモリ管理が協調せずに併存することになるため，これらのライブラリを使用することが困難な場合があった。ソースプログラムが入手できないライブラリの場合は，ライブラリが使用するメモリがLisp言語処理系のメモリ管理下におかれるように修正できないからである。

4.3 バイトコードインタプリタ

今回作成したLisp言語処理系は，インタプリタ方式を基本としているが，Lisp言語用バイトコードインタプリタによるコンパイラ方式も試験的に実装している。

Java言語よって記述されたプログラムのJavaバイトコードは，JVMによってインタプリタ実行されるので，その実行速度が速いとはいえない。C言語で記述された等価なプログラムと比べて，4倍から10倍遅いという報告もある[8]。プログラム実行中にJavaバイトコードをネイティブコードに変換するJITも普及しているが，Lispプログラムの字句解析，構文解析および局所変数の解析を事前に行い，これらの解析に伴うオーバヘッドを減らすことは重要である。

Lisp言語用バイトコードインタプリタの命令は，通産省情報処理技術者試験のアセンブラに採用されているCOMET [9]に似たものである。Lisp言語のデータ型を扱うために，適宜，拡張や変更を行った。例えば，汎用レジスタ（GR）をLispのオブジェクトを参照するものとしたり，CONS, CARおよびCDRといった命令を追加している。

Lisp言語用バイトコードインタプリタとのインタフェースとして，次の特殊形式および関数を追加した。

- バイトコードによる関数定義(特殊形式)
  (defbfun ＜記号＞ ＜リスト1＞ ＜リスト2＞)
  ＜記号＞は関数名，＜リスト1＞は関数本体のバイトコードの列，＜リスト2＞はこの関数で使用するLispデータの列。
- バイトコードの実行(関数)
  (exbcode ＜リスト1＞ ＜リスト2＞)
  ＜リスト1＞は実行するバイトコードの列，＜リスト2＞はバイトコードが使用するLispデータの列。

defbfunによって定義された関数は，インタプリタ実行される通常の関数（バイトコードによって定義されていない関数）から呼び出されることも可能であるし，逆にインタプリタ実行される通常の関数を呼び出すことが可能である。現時点では，Lispの関数をLisp言語用バイトコードに変換するコンパイラを実装していないので，利用者がハンドコンパイルしなければならない。

```
// throwするときに投げるExceptionの定義
class CatchThrowException extends Exception{
  LispObj tagname, value;  //タグ，値
}
…
  // (throw 〈タグ〉 〈式〉)の処理を
  // 行う部分
  void throw(LispObj arguments) throws
    CatchThrowException{
      〈式〉を評価し，その値をvalueとする．
      tagnameとvalueを与えて生成した
      CatchThrowExceptionのインスタンスを
      投げる．
  }
…
  // (catch 〈タグ〉 〈式1〉…〈式n〉)の処理を
  // 行う部分
  void catch(LispObj arguments) throws
    CatchThrowException{
      try{
        環境の保存
        〈式1〉…〈式n〉を評価
      }catch(CatchThrowException e){
        CatchThrowExceptionをとらえる．
        〈タグ〉が一致していれば
          環境を復元し，
          eの中の値を返して終了．
        〈タグ〉が一致してなければ
          受け取ったeを再び投げる．
      }
  }
…
```

図2: Javaの例外処理によるcatchとthrowの実現

4.4 非局所脱出

Lisp 言語の非局所脱出構造（catch と throw）を，2通りの方法で実現した。一つは Java 言語の例外処理を用いる方法，他の一方は Java 言語のスレッドを用いる方法である。

4.4.1 例外処理を用いた実現

例外処理を用いる方法の概略を，図2に示す。Exception クラスを継承した Catch Throw Exception というクラスを定義した。このクラスのインスタンスは，（thorw ＜タグ＞ ＜式＞)の＜タグ＞と＜式＞に対応するスロットを持つ。（Lisp 言語の）catch 式の本体を，Java 言語の try の中で評価し，throw 式の評価によって投げられた Catch Throw Exception のインスタンスを（Java 言語の）catch で捕える。＜タグ＞が一致しない場合は，再び，そのインスタンスを上位に投げる。

4.4.2 スレッドを用いた実現

スレッドを用いる方法の概略を，図3に示す。基本的には，（catch ＜タグ＞ ＜式1＞…＜式 n＞)の＜式1＞…＜式 n ＞を評価するためのスレッドを生成する。このスレッドは，＜タグ＞が一致する（ throw ＜タグ＞ ＜式＞)が評価されたら，対応する＜タグ＞を有している catch 式の評価に合流する。catch 式の評価中に別の catch 式が評価される場合は，スレッドがさらに生成される。

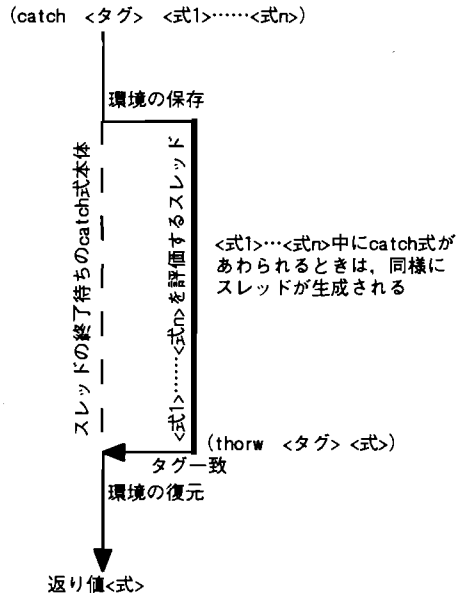

図3: Java のスレッドによる catch と throw の実現

5 評価

今回実装した Lisp 言語処理系がWWWブラウザ上で動作している様子を，図4に示す。利用者は，本処理系が配置されているWWWサーバの URL を指定することによって，（インストールすることなしに）本処理系を利用することが可能である。なお，定量的な評価ではないが，JIT コンパイラを備えていてスタンドアロンアプリケーションとして実行する場合，micro SPARCII 70MHz 程度の低速な CPU でも，学習を目的とするのであれば実用上差し支えない程度のレスポンスが得られていることを付しておく。

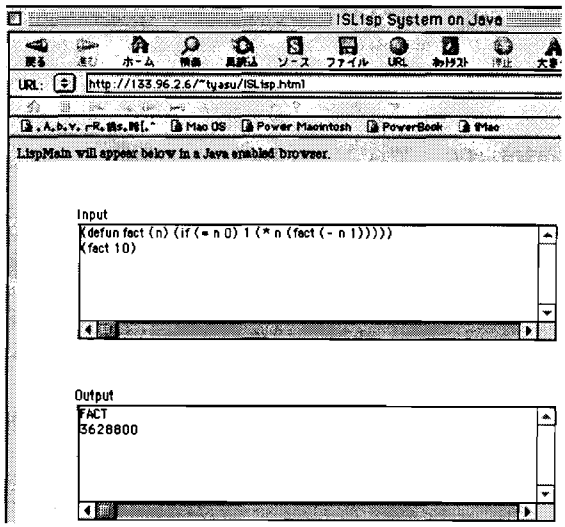

図4:実装した処理系のブラウザ上における動作例

Lisp 言語用バイトコードインタプリタによるコンパイル方式の性能を評価するために，リストの長さを求める再帰的な関数を定義して，その実行時間を計測した。使用したコンピュータは SunUltra5（Ultra SPARACIIi269MHz)，JDK のバージョンは JDK1.2 fcs0（JIT コンパイラ付き)，引数として与えたリストの長さは10である。インタプリタ方式で実行した場合は328ミリ秒，（Lisp 言語用バイトコードインタプリタによる）コンパイル方式により実行した場合は2ミリ秒を要した。この数値をみる限り性能が極めて悪いことから，インタプリタ方式は，学習を目的とする場合は別にしても，実用的でない。コンパイル方式の場合は，実際にはコンパイルに要する時間が利用者のコスト負担となることを考えると，Lisp 言語用バイトコードよりも，より高い実行性能が期待できる Java バイトコードに翻訳するコンパイラ方式を採用した方が良いように思われる。

先述した非局所脱出の実現方法を，上記と同じ環境で比較した。catch 式が3つ入れ子になっていて，3つめの catch 式から2つめの catch 式に throw するような関数を定義して，その実行時間を計測した。Java の例外処理機能を用いた場合は1ミリ秒，スレッド機能を用いた場合は5ミリ秒を要した。スレッド機能を用いた場合の方が大幅に遅くなっていることから，スレッドを生成するコストが高いことがうかがえる。スレッドを用いた場合は，実装の工夫により，catch 式が深く入れ子になっていても，例外処理を用いた実現

のようにthrowしたタグとの一致を確認しながら順次戻ることをしなくても，脱出した複数のcatch式に対応するスレッドを一括して終了させることが可能である。しかし，スレッド生成のコストが高いことを考えると，スレッドを用いた非局所脱出実現の方法は得策でないことがわかった。

6 まとめ

学校教育におけるプログラミング言語としてLisp言語を提案し，マルチプラットフォーム向けLisp言語処理系の実装について報告した。この処理系は，JVMが装備されていれば，ハードウェアやオペレーティングシステムを問わずに動作する。今後の課題としては，ISLispの機能のうちオブジェクト機能やコンディションシステムなどの未実装なものを完成すること，ISLispのプログラムをJavaバイトコードに翻訳するコンパイラの実装，アプレット実行時のためのFTPクライアントによるプログラム入力補助機能の実装があげられる。

謝辞

本研究は，原野直樹，青木恭一郎，小濱佳久の各氏の卒業研究に関わって行われた。各位に深謝する。

参考文献

[1] 坂元昂：教育改革に貢献する情報通信，情報通信学会誌，Vol.17, No.1, pp.1-9 (1999).

[2] 坂元昂：高度情報通信時代の教育改革，SCS教育工学特別講演資料 (1999).

[3] Abelson, H. and Sussman, G. : *Structures and Interpretation of Computer Programs*, The MIT Press (1985).

[4] *The Java Language Environment A White Paper*, Sun Microsystems (1995).

[5] *Information Technology - Programming Languages, their environments and system software interfaces - Programming Language ISLISP*, International Standard, Reference number ISO/IEC 13816:1997(E) (1997).

[6] Clinger, W. and Rees, J. eds. : *The Revised*^{4} *Report on the Algorithmic Language Scheme*, CIS-TR-90-02, University of Oregon (1990).

[7] JIS X 3012:1988 プログラミング言語ISLISP，日本規格化協会 (1998).

[8] 有賀妙子，竹岡尚三：Java 1.1プログラミング，ソフトバンク (1997).

[9] 玉井浩：徹底研究 COMET & CASL，サイエンス社 (1995).