

Chapter 1 プログラミングとJavaについて

1.1 プログラミングについて

■プログラミングとは何でしょうか？

文章を作成したり、表計算をするような場合は、コンピュータに用意されているソフトウェアを買ってきてそれを使いますね。授業（セミナー）を聞いたり、マニュアルなどを読んで、そのソフトウェアの使い方を覚えることになります。あるいは、自分で試行錯誤しながら使い方を学んで行きます。

しかし、ソフトウェアで用意されていない動作をコンピュータさせたい場合はどうでしょうか？例えば、自分の思い通りに表示をしたいとか、理学書や経済学に載っている計算方法に基づいて、ファイルに保存してあるデータに加工を施したい場合があるでしょう。もちろん、データの加工や分析であれば、表計算ソフトウェアを使ってもある程度のことはできると思います。製品として用意されているソフトウェアや、ネットワーク上で誰かが作ったソフトウェアなどを探し出しても、ある程度のことはできると思います。しかし、求める機能にぴったりと合うソフトウェアがなければ、結局は目的とする動作をコンピュータに実行させることはできなくなってしまいます。

それから、普段利用しているソフトウェア自体について考えてみましょう。ワードプロセッサにせよ、表計算ソフトウェアにせよ、それは結局「他の誰かが作成してくれた」ソフトウェアを、お金を払ったり、ネットワークからダウンロード（コピー）してきて使っているに過ぎません。それらのソフトウェアは一体何なのでしょう？

ソフトウェアというものは、コンピュータにさせたい動作を予め記述しておいたものを意味しています。ですから、誰かが時間を掛けてワードプロセッサや表計算ソフトウェアを作ってくれたのです。皆さんはそれを利用しているに過ぎません。ですから、コンピュータを動かすときには、必ず誰かがソフトウェアを作っておく必要があります。自分自身でそれらのソフトウェアを作る機会は、それほどはないと思います。しかし、誰かが作らないと、世の中に存在しないのです。

さて、ソフトウェア（Software）というのは総称として使われることが多い用語です。個々のソフトウェアのことをプログラム（Program）と呼んでいます。例えば、ワードプロセッサは一つのプログラムです。コンピュータは、プログラムがなければ何もできません。自分の思い通りのことをさせるためには、コンピュータにプログラムを与える必要があります。もちろん、そのプログラムは買ってきたものかも知れませんが、ダウンロードしたものかも知れませんが、自分で作ったものかも知れません。

★プログラムはどこから入手するか？

- ・誰かが作ったものを買ってくる
- ・ネットワーク上で無料でダウンロードする
- ・自分で作成する

コンピュータは、与えられたプログラム通りに動作します。ここで注意しなければならないのは、コンピュータは、プログラムで指示された以外のことは一切やってくれないことです。人間ならば、例えば、食事の後には気を使って皿まで洗ってくれたりするかも知れませんが、コンピュータはそういう指示をしない限り何もしてくれません。プログラムを作る人は、実はそういう配慮をコンピュータにさせるかどうかもちいち予め記述しておかなければならないのです。

プログラムを作成することは、プログラミング（Programming）と呼ばれています。プログラムを作成することを、「プログラムを組む」と言ったりもします。ちなみに、プログラムを作る人のことを、プログラマー（Programmer）と呼ぶこともあります。また、プログラムの作成を総称してプログラム開発（Program Development）と呼んでいます。

コンピュータに自分の思い通りのことをさせようと思えば、プログラミングするしかないのです。既存のソフトウェアを用いれば、ある程度までのことはできます。しかし、自分が要求したい目的に特化した動作をコンピュータにさせたいときには、やはりプログラミングしなければならないのです。例えば、自分の考案したゲームをコンピュータ上で実現するためには、プログラムを作成する（自作するなり、他の人に作らせるなり）必要があります。

また、ワードプロセッサや表計算ソフトウェアを作っている設計者は、それらのソフトウェアを利用するユーザの要求の中で最も必要とされるという機能を最大公約数的に実現しているだけに過ぎません。例えば、ワードプロセッサでルビが必要だと設計者が判断すれば、その機能をプログラミングします。日本以外では、ルビの概念がありませんから、設計者が日本市場によほど興味が無い限り、その機能はプログラミングされないことになります。ルビが必要なユーザはどうすればいいのでしょうか？

■プログラミングを学ぶ意味

多くの人は書店で見掛けて、この本を購入したのかも知れませんが、中には酷いことに（著者にとってありがたいことに）この本を教科書として指定され、プログラミングの意味を知らずに強制的に授業で勉強する学生の諸君もいることと思います。ここで、一つプログラミングを学ぶ意味があるかどうかについて考えてみましょう。

★実行されるテキスト

プログラミングを学ぶ理由の一つは、テキストとして記述されたものが、コンピュータの世界では実行までできることが今までの世界と決定的に異なることにあります。そのような新しい方法を学ぶことは学問として意味があることだと思います（注1）。考えてみてください。例えば、実世界では紙に書かれたスケジュール表は、その通りに自分（や他人）が行動しない限り、ただの情報（紙切れ）でしかないのです。ところが、コンピュータ上では記述されたテキストが、一旦プログラムとして解釈されれば、コンピュータが自動的に実行してしまうのです。極端な例にしまえば、小説を書けば、コンピュータ上で仮想的にそこに描かれた世界が実現されるということになります。これは、一つの面白いパラダイムです。



図1-1 欲しいニンジンプログラムする

また、テキストとして完結した（Completed）仕様（Specification）を作るためには、プログラミング言語のような人工的な記述方法を利用することが最適であると言われています。特に、このような記述方法の場合、コンピュータによって簡単な間違いを指摘してくれる（皆さんは、これからたびたびエラーメッセージに遭遇する羽目になるでしょう）のは、コンピュータによる支援（エラーとして叱られるので、あまりサポートしてくれるとは思えないでしょう）として考えることができます。例えば、法律を作る場合、いろいろな場合を想定しなければなりません。これはプログラムの設計法と共通するものがあります。ただし、プログラムの場合、コンピュータによって実行できる形に変換するときに、ちょっとした綴り間違いや、仕様のミスを自動的にチェックしてくれるという利点があります。一方、法律を作る場合は、作ったものを何度も読み返して自分でミスを発見する以外方法がありません。

しかし、どうも「いいことだけを言って騙されている」ような気持ちになります。私の教えている学生も、同じことを言っていました。本当に騙されているのでしょうか？

確かに、プログラミング教育を受けても、物理や数学などと同じように後で役に立たないのに習っただけに終わる学生も少なくありません。大学の教養課程や高校で一体プログラミングを教えることがどの程度必要なのか、どのように教えるべきかについては、今までも議論されてきましたし、これからも長く議論していく必要があるでしょう。

★アルゴリズムと論理性

実行されるテキストを記述できること、あるいはコンピュータが支援してくれる形で完結した仕様を作成できること以外に、プログラミングを学ぶということは、さらに別の側面も持っているのではないのでしょうか。

それは、効率の良い作業の手順を考える、あるいは論理的に考えるための礎を作るという側面です。そういう作業を行なう手順のことは、アルゴリズム（Algorithm）と呼ばれています。例えば、皿を洗うときには、まず水道の蛇口をひねって水を出します。その水が皿に当たるようにします。そして、スポンジを取り出して、皿をごしごしとやりま

す。これも、簡単な一つのアルゴリズムです。もちろん、他のやり方もあるでしょう。そのような物事を実行する手順の可能性を探る能力も、プログラミングの経験から培われるのではないのでしょうか。

特にプログラミングにおいては、変数と式、繰返し（条件分岐）、そして呼出しという三つの道具を用いて手順を記述していきます。これらの道具を用いると、非常に柔軟に物事の手順が記述できます。これは、本書の中心的なテーマですので、以降の章でJavaを使って詳細に説明していきます。

プログラミングの経験は、頭の中で考えたことを実際に動かしてみることができるということを「実行されるテキスト」の項で書きましたが、完成させることができるのです。日常の経験の中で、何かを一定の形で完成させるというのは現代社会ではなかなか難しいことです。ものごとを完成させる喜びを、ある程度手近に体験できることは、眠っていた知的興奮を醒まさせるよい手段かも知れません。

しかも、論理的に整理されていないとプログラムは「動いてくれない」のです。自分の考えで欠落していたものは何か、なぜ思ったとおりに動いてくれないかを考えることを経験できる数少ないチャンスかも知れません。そもそも、自分がコンピュータにさせたい要求自体に問題があるのかも知れません。動いたプログラムを考察するよりも、動かないプログラムを考察する方がずっと奥深いことです。考えて会得する、造語してしまえば「考得する」ための教材なのです。

そんなこんなで、高校や大学の教養課程で良いアルゴリズムを創作できるようなプログラミング教育は行なうことは（著者自身は）意義深いものだと思っています。もちろん、独学でも同じことは言えると思います。ただ、現在のプログラミング教育はそのような教育効果を挙げていないのかも知れません。

注1 これは慶應義塾大学の「大岩 元」教授から聞いたものですので、大岩説と呼びます。

1.2 プログラムの実行方式について

■ストアド・プログラム方式

現在のほとんどのコンピュータは、あらかじめプログラムを与えておき、後からそのプログラムを起動するという方式を採用しています。この方式のことをストアド・プログラム方式（Stored Program Method）と呼んでいます。ストアド・プログラム方式を採用しているコンピュータのことをフォン・ノイマン型（Von Neumann）コンピュータと呼ぶこともあります。

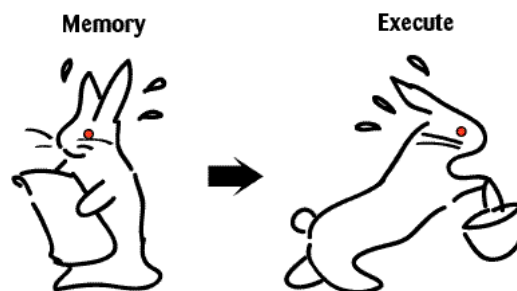


図1-2 スタッド・プログラム方式

■プログラムのレベル

プログラムというのは、具体的にはコンピュータに与えられる命令（Instruction）の連なり（Sequence）のことを指しています。コンピュータは、プログラム中に書かれている命令を見ながら、一つ一つ命令を実行していきます。

しかし、様々なレベルでのプログラムが存在しています。何故でしょうか？同じ事を記述するのに、どのように命令を記述するのが異なるからです。簡単に言ってしまうと、コンピュータが理解しやすいのか、それとも人間が理解しやすいのかによって、レベルの差が生まれてきます。

★プログラムのレベル分け

人間が理解できるプログラム

→人工的な疑似自然言語を使って記述されている

コンピュータが理解できるプログラム

→コンピュータが直接実行できるような命令で記述されている

プログラムを記述するために用いられる、人工言語のことをプログラミング言語（Programming Language）と呼んでいます。また、コンピュータが直接実行できるような命令のことを命令コード（Instruction Code）と呼び、このような命令コードの連なるのことを機械語（Machine Language）と呼んでいます。機械語は、16進数（基本的には2進数）で表現されています。機械語を人間がコード化（Coding）するのは、非常な手間が掛かります。また、コード化された16進数から、どのような命令であるかを理解するのも、異常な手間がかかります（注1）。



図1-3 プログラミング言語と機械語

機械語の命令コードと1対1に対応した簡易人工言語のことをアセンブリ言語（Assembly Language）と呼んでいます。アセンブリ言語を命令コードに変換するソフトウェアは、アセンブラ（Assembler）と呼ばれていました。その逆に、命令コードをアセンブリ言語に逆変換するソフトウェアは、ディスアセンブラ（Disassembler）と呼ばれていました。これらのソフトウェアは、以下に出てくるコンパイラというソフトウェアに比べれば、ずっと簡単なソフトウェアで、まだコンピュータの処理速度があまり速くなかった時代には、頻繁に用いられていました。

注1 昔のプログラマは、頭の中にコード表が埋め込まれていましたので、ある命令がどのような16進数で表現されるのかすぐに変換することができました。逆に、16進数を見て、それがどのような命令であるのかも一瞬で思い浮かべることができました。このような人は、「歩くアセンブラ・ディスアセンブラ」と呼ばれていました。

■プログラミング言語の実行方式

プログラミング言語は、人間には読みやすいのですが、そのままではコンピュータは直接実行することができません。コンピュータが直接受け取って実行できるのは、機械語だけなのです。そのために、プログラミング言語で記述されたプログラムを実行するには、幾つかの方法がありますが代表的な次の3つの方式を紹介します。

★コンパイラ方式

プログラミング言語で記述されたプログラムを「予め」機械語へ翻訳する方式です。翻訳するソフトウェアのことをコンパイラ（Compiler）と呼びます。実際の実行は、翻訳された機械語の方のプログラムを実行する形で行なわれます。

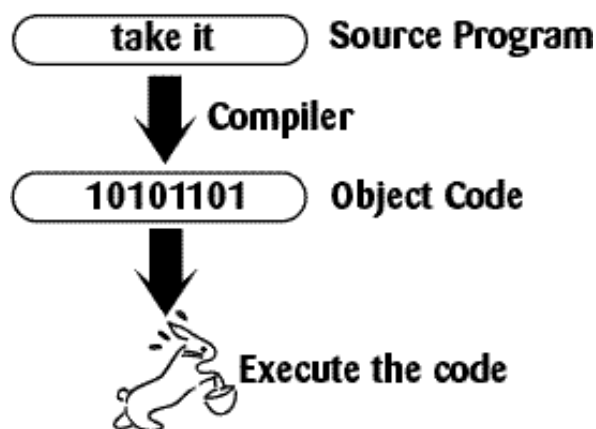


図1-4 コンパイラ方式

プログラミング言語で記述された翻訳される前のプログラムのことを、ソーステキスト（Source Text）あるいはソースプログラム（Source Program）と呼ぶことがあります。また、翻訳された後の機械語の方のプログラムのことを、オブジェクトコード（Object Code）と呼ぶことがあります。

この方式では、翻訳された機械語のプログラムが実行されますので、高速な実行が約束されます。特に、事前に翻訳しますから機械語をより高速に実行できる技術があります。この技術のことを最適化（Optimization）と呼びます。ただし、コンピュータの種類や、オペレーティングシステムの種類別に機械語が違うので、別のコンピュータに持っていったときは、もう一度コンパイルしなおす必要があります。

★インタープリタ方式

プログラミング言語で記述されたプログラムをその場で翻訳しながら実行する方式です。その場で翻訳しながら実行するソフトウェアのことをインタープリタ（Interpreter）と呼びます。

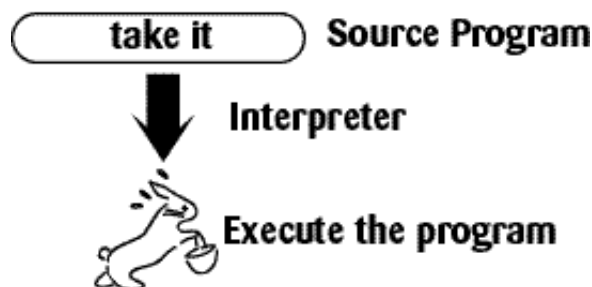


図1-5 インタープリタ方式

この方式では、その場で翻訳するので実行は低速になります。コンピュータやオペレーティングシステムが違っててもインタープリタさえあれば、プログラムを実行することができます。

★中間コード（仮想マシン）方式

プログラミング言語で記述されたプログラムを予め中間コードに翻訳しておく方式です。この場合に、中間コードに翻訳するソフトウェアのこともコンパイラと呼びます。中間コードに翻訳されたプログラムは、ランタイム・インタープリタ（Runtime Interpreter）によって実行されます。

中間コード（Intermediate Code）とは、機械語ではないのですが、機械語に近い形の命令コードになっています。仮想的なマシン（Virtual Machine）を想定・設計し、そのマシンが実行できる命令コードのことを指します。

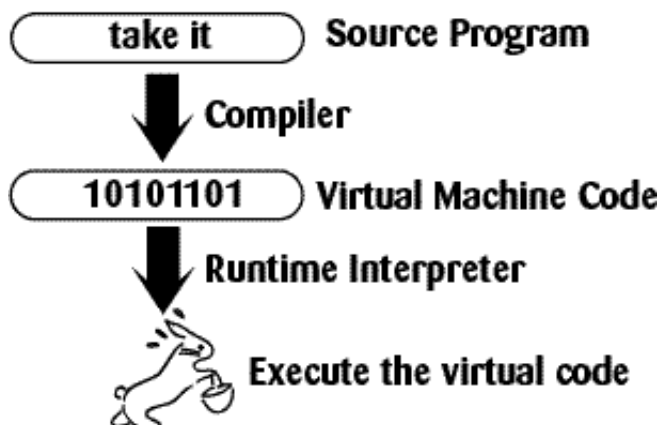


図1-6 中間コード方式

ランタイム・インタープリタは、仮想マシンを模擬的に実行する、つまりシミュレーション（Simulation）するソフトウェアとなっています。これに対して、通常のコンパイラ方式では、機械語のプログラムを、コンピュータが直接実行しています。機械語は、それぞれのコンピュータに固有の命令コードになっていますから、中間コードに対して、機械語のことをネイティブコード（Native Code）と呼ぶことがあります。

この方式は、ちょうどコンパイラ方式とインタープリタ方式の合の子みたいなものになっています。どうしてこのような方式が生まれてきたのでしょうか？まず、予め中間コードまで変換しておきますので、インタープリタ方式に比べて、その場でいちいち翻訳しなくて済むので、ある程度高速にプログラムを実行することができます。そして、中間コードに変換されたプログラムは、ランタイム・インタープリタが用意されていれば、どのコンピュータに持って

いっても実行できます。実は、コンパイラやインタープリタに比べたら、ランタイム・インタープリタはかなり小さいソフトウェアなのです。ですから、いろいろなコンピュータ、あるいはオペレーティングシステム的环境下で用意することができるのです。

1.3 プログラミング言語Javaについて

■プログラム言語とJava

この本の中で紹介していくJavaというものは一つのプログラミング言語です。プログラミング言語には、いろいろな流派がありまして、その主要なものとして次のような流派があります。ここでは、流派の名前と、それに属する主要な言語を図中に挙げておくに留めます。というのは、これ以外にも一杯プログラミング言語が存在するからです。各国・地方で言語が異なるように、あるいは一人一人のしゃべり方が異なるように、新たなプログラミング言語を企画する人の人数分だけプログラミング言語が存在すると言っても過言ではないでしょう。それらを一つ一つを紹介するだけで一冊の本になってしまいます。

★プログラミング言語の流派

- ・命令型言語 (Imperative Programming Language) Fortran, Cobol, Pascal, C, C++
- ・関数型言語 (Functional Programming Language) Lisp, Scheme
- ・論理型言語 (Logical Programming Language) Prolog
- ・オブジェクト指向型言語 (Object Oriented Programming Language) Smalltalk, Java

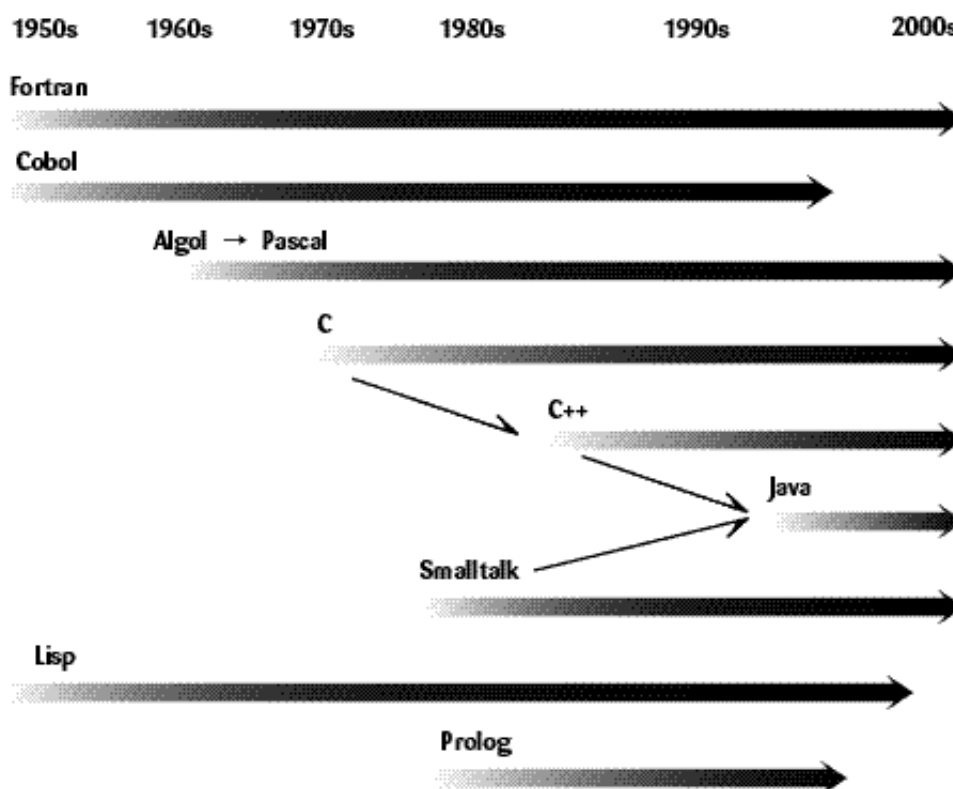


図1-7 プログラミング言語の流派図

図を見ればわかるとおり、Javaは、C言語の流れを汲む、オブジェクト指向プログラミング言語となっています。1995年にワークステーションの老舗 Sun Microsystems社が世に送り出した、最新鋭のプログラミング言語と言っていいでしょう。言語の設計が整っているので専門家の間でも評価が高い言語です。特に、WWWプロジェクトやインターネットの加熱と相まって、アプレット（後で触れます）用の実用プログラミング言語として、あるいはコンピュータやオペレーティングシステムの種類に依存しないプログラミング言語として使われています。

■Javaの実行方式について

Javaは、紹介した3つの方式のうちで、中間コード方式を採用しています。はい、これだけわかっていたら結構です。次に進んでください。以下になぜJavaが中間コード方式を採用しているかの理由を書いています。興味のない人は混乱するかも知れませんが、読まなくて結構です。

中間コード方式は、オブジェクト指向プログラミング言語での標準的な実行の形態となっています。なぜなら、オブジェクト指向言語では、プログラム実行時にプログラムが実行される環境を必要とし、環境に対して操作を行なうプログラミング言語であるからです。ここでいう環境（Environment）あるいは実行環境（Runtime Environment）とは、プログラムから利用できる機構のことを指したり、プログラムの実行中の途中結果を記憶している機構のことを指しています。

コンパイラ方式を採用している言語では、環境に対して実行時に余分にデータを記憶していく場所を確保したりすることが、非常に低レベル（機械語レベル）の形でしか用意されていません。そのために、プログラマは、危険（注1）を承知でそのようなプログラミングを行なわなければなりません。オペレーティングシステムや特殊なハードウェアを用意することによって、危険を回避できるのですが、それでもプログラミングとしてはあまり洗練された方式ではありませんでした。

オブジェクト指向言語は、そのような問題を洗練された方式で解決しました。その反面プログラムを実行するまで、どの程度データを記憶しておく領域が実行時に動的（Dynamic）にどんどん変わっていきます。静的（Static）なデータ記憶領域を中心に考えられているコンパイラ方式では対処できません（注2）。そのため、実行時の環境をお膳立てすることができるインタープリタ方式か中間コード方式を取らざるを得ないのです。しかし、インタープリタ方式では実行速度が遅いため、最終的には中間コード方式が採用されることが多いのです。



図1-8 Javaの実行環境

また、プログラム自身がネットワークで別の場所に運ばれて実行されるような環境を想定しているJavaのようなプログラミング言語では、インタープリタ方式か中間コード方式を採らざるを得ません。インタープリタ方式では、ちょっと遅さが問題になりますし、プログラムを実行させるソフトウェアも大きくなります。予め中間コードまで変換しておき、それをネットワークを介して別の場所に転送した方が効率的ですし、受け取る側のコンピュータでも小さなランタイム・インタープリタさえ用意しておけば、それを実行させることができます（注3）。

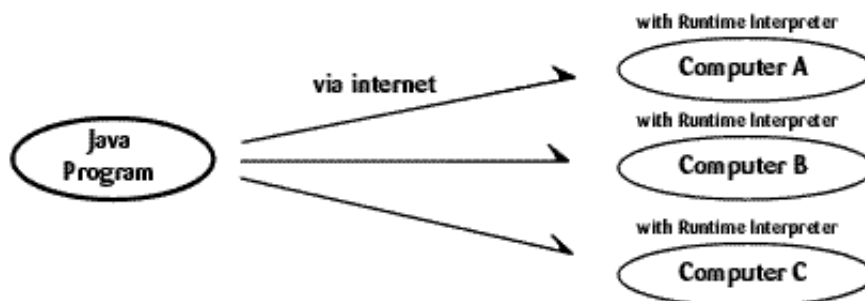


図1-9 ネットワーク配信されるJavaプログラムの実行方式

Javaで書かれたプログラムは、HTMLで記述されたページと同じようにインターネット上を転送されます。Webページを見るためのブラウザ（Browser）、例えばNetscape NavigatorやMicrosoft Internet Explorerなどは、Javaの中間コードを実行するために、ランタイム・インタープリタをその内部に持っていますし、オペレーティングシステム上に用意されているものを利用することもできます。

注1 実行時にコンピュータを暴走させる危険です。最近のパソコンでは、暴走させてもリセットボタン等でもう一度再起動すれば済むのですが、昔の大型コンピュータでは、そのような低レベルのプログラムを組むことは、一部の設計者にしか許されていませんでした。他の人は、予め固定された大きさのデータ記憶領域をうまく活用するようなプログラムを書かなければいけませんでした。

注2 コンパイラ方式を採用しながら、オブジェクト指向言語に分類されているC++というプログラミング言語もありますが、そのような無理がたたって、プログラムの動作がプログラマにとってわかりにくくなってしまったので、結局あまり普及していません。プログラマが想定できるプログラムの動きのことをプログラミングモデル (Programming Model) と呼ぶことがありますが、このモデルが簡単な方がプログラミングしやすいのです。

注3 これ以上他の機種種のコンピュータに移動しないで、特定のコンピュータ上で高速に動かしたいという要求もあるために、Javaの中間コードを特定のコンピュータの機械語に変換するソフトウェアや、Javaのソースプログラムを直接機械語に変換するソフトウェアも各社から提供されています。これらのソフトウェアは、ネイティブコードに変換するので、ネイティブ・コンパイラ (Native Compiler) と呼ばれています。もちろん、変換した後も動的な実行環境をサポートする部分が必要ですので、そのサポートする部分をオブジェクトコードに組み込んでいます。

■Javaをプログラミング言語として選ぶ意味

前に出てきましたように、プログラミング言語を学ぶ意味は、ものごとの手順を記述する、すなわちアルゴリズムを記述することにあります。Javaはそれに適した言語でしょうか？どちらかと言えば、BasicやPascalのような言語、オブジェクト指向言語ではSmalltalkのような言語、あるいは最近のものであれば、AppleScriptのような言語がそれに相応しいと考えられます (注1)。このようなプログラミング言語の特徴は、わずかの記述で最初の一步のプログラムが記述できることにあります。JavaやC言語の場合には、これらのプログラミング言語に比べると、かなり訳のわからない呪文や記号を書かなければなりません。

しかし、それにも拘わらずJavaがもてはやされる理由は、一つは特定のコンピュータやオペレーティングシステムを選ばないという特徴があるからです。さまざまな会社がJavaの支援を表明し、Javaが動かないシステムはなくなってきました。もちろん、すべての会社のコンピュータで最新のJavaが動くわけではないのですが、この本で扱う内容程度であれば、どのJavaのランタイム・インタプリタでも動くでしょう。

加えて、言語のサポートをSun Microsystemsというコンピュータ開発会社 (Vender) が強力に推し進めていることがあります。それは、言語の開発・実行環境を提供するだけでなく、さまざまな規格を作ったり、Javaを利用した開発の支援を行ったりしています。仕事と同じで、誰かが責任を持って推し進めてくれていると安心感が生まれます。他の開発会社も安心して参画できますので、これからCやC++に取って代わる標準のプログラミング言語として普及していくことでしょう。

そういう状況の中で、Javaをプログラミング言語として選ぶのは、総合的なバランスから見れば、有力な選択肢であると言えます。これからのプログラミング言語でオブジェクト指向の要素が入っていないものは現出しないでしょうから、Javaを知っておけば、他のプログラミング言語も理解できることでしょう。

注1 AppleScriptとVisual Basicは、それぞれApple社とMicrosoft社が、自社のオペレーティングシステム上で稼働する簡易言語 (Scripting Language) として用意されています。元のBasicやAppleScriptが教育用に適しており、Visual Basicが適していないのは、Visual Basicが多機能である反面、いろいろな機能を未整理で言語の中に取り込んでいるからです。Apple社の製品は、シンプルさを中心に設計されていますが、Microsoft社の製品は高機能な反面、複雑なものをそのままユーザに見せるという設計がされています。それぞれの会社の文化・戦略の違いによるものでしょう。

■アプリケーションとアプレットについて

アプリケーション (Application) とは、特定のオペレーティングシステム上で稼働されるプログラムのことを言います。古い教科書では応用プログラムという呼び方をされていました。例えば、皆さんがご存じのような次のようなプログラムは、すべてアプリケーションです。これらのアプリケーションは、WindowsやMacOS、あるいはUNIX系 (Solaris, BSD, Linux, BeOSなど) のオペレーティングシステム上で稼働しています。

Javaは、ランタイム・インタプリタを必要としますが、最終的にアプリケーションとして動くプログラムを開発することができます。これは、今までのプログラミング言語でもまったく同じで、アプリケーションとして動くプログラムを開発するために用いられてきました（注1）。

Javaが際立って他のプログラミング言語と異なる点は、アプレット（Applet）という形態のプログラムを開発できることです。アプレットは、HTMLで書かれたWebページと共に、インターネットを介して、あるコンピュータから別のコンピュータに転送されてから実行されるプログラムのことです。もちろん、プログラムが実行されるコンピュータ上でランタイム・インタプリタが動いていないといけません。なお、このときのプログラムの転送のことを専門的には実行移動（migration）と呼ばれています。

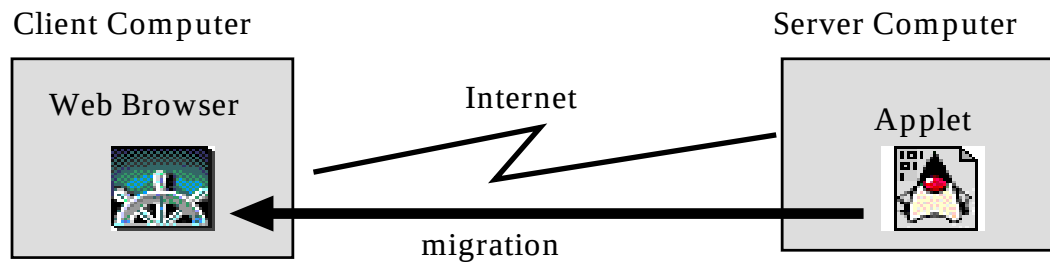


図1-10 アプレットの実行方式

前に説明したように、このアプレットの実行形態をサポートするために、WebブラウザのNetscape NavigatorやInternet Explorerの中には、ランタイム・インタプリタが含まれているのです。

注1 C言語は、アプリケーションだけではなくて、オペレーティングシステム自体も開発するために設計されたプログラミング言語です。このような言語の特徴は、かなり機械語に近い部分もプログラミング言語の中で記述できることにあります。そのような言語を低水準（low-level）プログラミング言語と呼んでいます。また、アルゴリズムの記述やアプリケーションの開発を目的とした言語のことは、高水準（high-level）プログラミング言語と呼ばれています。

1.4 まとめと課題

■まとめ

プログラミングを学ぶ意義

- ・コンピュータ内の仮想世界において実行されるテキストの可能性
- ・コンピュータ支援のもとに完結された仕様の作成
- ・論理的な考え方、アルゴリズムの孝得

プログラムのレベル

- ・プログラミング言語で書かれたソースプログラム
- ・機械語（命令コードの羅列）で書かれたオブジェクトコード

プログラムの実行形態

- ・コンパイル方式
- ・インタプリタ方式
- ・中間コード方式

Javaで書かれたプログラムの実行形態

- ・中間コード方式でランタイム・インタプリタを必要する
- ・オペレーティングシステム上で実行されるアプリケーション
- ・インターネットを介して転送されて実行されるアプレット

■課題

1-1

Fortran、Cobol、C、Pascal、Lisp、Smalltalk-80、Basicのそれぞれのプログラミング言語について、プログラムはどのような実行形態で実行されるのか調べなさい。また、余裕があればそうなった理由（歴史的な事情や機構など）も調べなさい。

1-2

なぜ、中間コードはネイティブコードに比べて遅いのか答えなさい。どのようにすれば、中間コードで記述されたプログラムを高速に実行することができるのか考えてみなさい。

1-3

歴史的にはコンパイラ方式の方がインタプリタ方式よりも先に考え出されました。どうしてそうなったのか考えてみなさい。ヒント：アセンブラのことを考えてみなさい。余裕があれば、この問題を主題にしてFortranとLispが開発された歴史を調べてみなさい。

1-4

学生であれば、どうしてプログラミングを教えるのか、あるいはなぜJava言語を選んだのかを担当の先生に失礼のないように質問してみなさい。