Chapter 1. プログラムとプログラミング言語Java

１．１．プログラムとは何か？

プログラム（Program）とは、コンピュータに何をさせるのかを記述した手順になっています。そのためには、効率の良い作業の手順を考えるとか、あるいは論理的に考えないといけません。作業を行なう手順のことは、一般的には、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。これをコンピュータに、この場合はロボットかも知れませんが、実行させるように記述したのがプログラムということになります。

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（Programmer）と呼ぶこともあります。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。プログラム全体を総称してソフトウェアと呼ぶことは御存知でしょう。

記述された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。

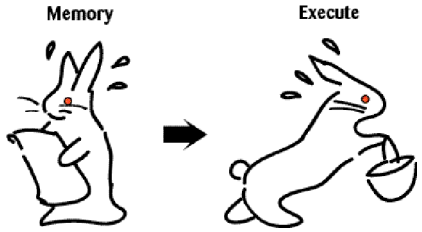

図1-1　プログラムに書かれた命令を実行するコンピュータ

しかし、様々なレベルでのプログラムが存在しています。なぜでしょうか？同じ事を記述するのに、どのように命令を記述するのかが異なるからです。簡単に言ってしまえば、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、人工的な疑似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

（InstructionCode）と呼び、このような命令コードの連なりのことを機械語（MachineLanguage）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大変でしょう。そこで、プログラムを記述し実行するのに様々な形態が生まれるようになってきました。

図1-2 プログラミング言語と機械語

１．２．プログラミング言語の実行方式

前の節で紹介しましたように、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、幾つかの方式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。

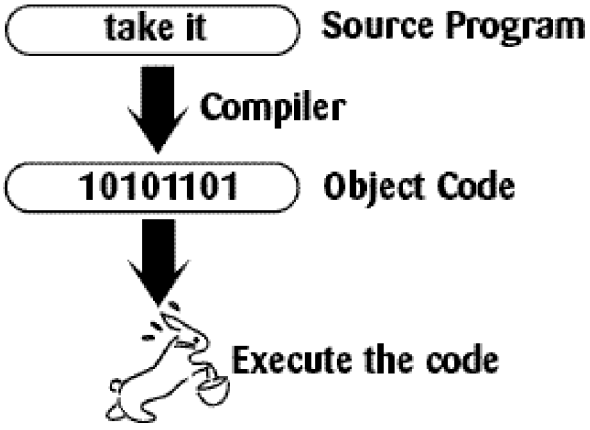

図1-3 コンパイラ方式

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、コンパイル方式ではコンピュータが用いているプロセッサ（CPU）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。

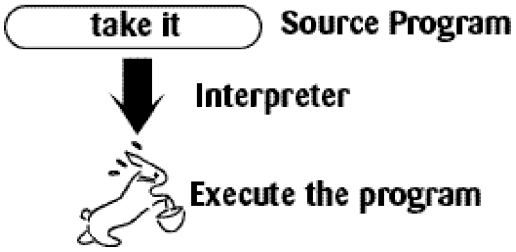

図1-4 インタープリタ方式

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。しかし、コンピュータやオペレーティングシステムが違っ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（IntermediateCode）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。

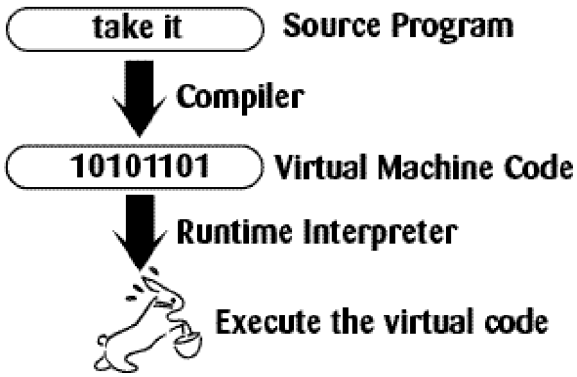

図1-5 中間コード方式

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、翻訳の大きな作業をしませんので、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の環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とその実行方式

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表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++, 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

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、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（1.1版から少し崩れましたが）専門家の間でも評価が高い言語です。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のプログラム実行方式について

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れば結構です。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が、興味のない人は混乱するかも知れませんので、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（RuntimeEnvironment）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でした。オペレーティングシステムやメモリを保護する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、データを記憶しておく領域の大きさが実行時に動的（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、オペレーティングシステムにそのような領域を確保する作業をプログラマがでプログラミングしていました。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。ですから、動的に変わるのを気にせずにプログラムできるモデルの方が利用しやすいと言えます。このようなモデルを採用している言語の場合は、よく高級プログラミング言語と呼ばれています。高級プログラミング言語の場合は、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。

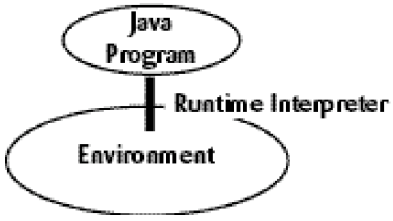

図1-6 Javaの実行環境

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。

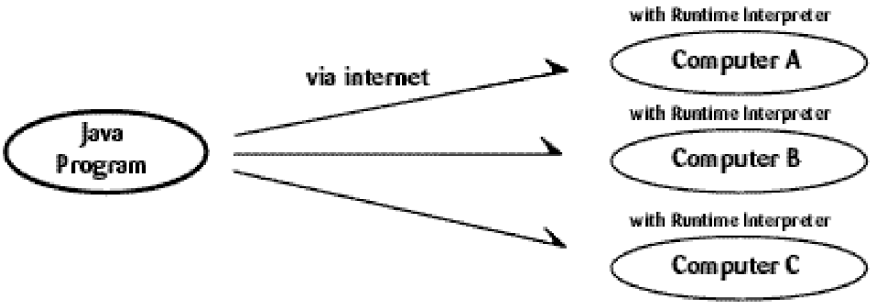

図1-7 ネットワーク配信されるJavaプログラムの実行方式

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などの昔の版のものは、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した。現在では、オペレーティングシステム上に共通のものが用意されています。

Javaのラインタイム・インタープリタのことを、Javaの仮想マシンということで、JavaVMあるいはJVM（Java Virtual Machine）と一般的に呼ばれています。Mac OS Xも含む、ほとんどのオペレーティングシステム上で、JavaVMが実装されていますので、Javaで書かれたプログラムは、基本的には一度記述したら、どのコンピュータでも実行できるようになっています。

なお、これとは反対に、これ以上他の機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翻訳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。Javaの場合は、記述されたJavaのプログラムから直接機械語に翻訳するコンパイラを、JIT（JustIn Time）コンパイラと呼んでいます。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。

１．４．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Mac OS X上で稼働する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、特定のオペレーティングシステム上だけで稼働しています。ですから、オペレーティングシステムが替わるとそれ用のアプリケーションを用意する必要があります。

Safari, Netscape Navigator, Internet Explorer, Microsoft Excel, Microsoft Word, AppleWorks, iMovie, Mail, Adobe Illustrator, Adobe Photoshop, FinalCut Pro, etc.

Javaは、ランタイム・インター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。しかし、JavaVMの実行環境があっていれば、オペ

レーティングシステムごとに開発する必要はなくなりました。

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・インター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。

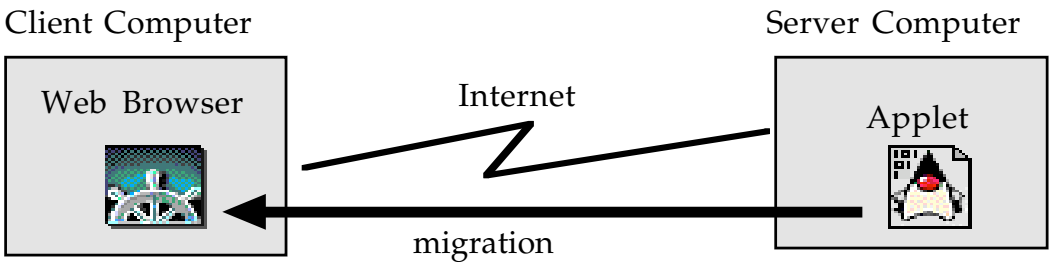

図1-8 アプレットの実行方式

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の昔の版のものでは、ソフトウェアの中に独自のJavaVMが含まれていました。現在は、Safariなども含め、ほとんどWebブラウザで、オペレーティングシステムで用意されたJavaVMを利用する場合がほとんどです。本書では、アプリケーションも取り扱いますが、このアプレットの開発を基本としたJavaプログラミングを説明していきます。

１．５．Javaのクラスとクラスライブラリ

Javaのプログラムは、クラス（Class）というまとまりで記述されます。１つのプログラムは、複数のクラスから構成されますが、単純なプログラムの場合は、１つのプログラムが１つのクラスに対応しています。自分で記述したプログラム、すなわちクラスを実行するのには、JavaVM以外に、他のクラスを必要とします。他のクラスが用意した手続き（Method）を用いて、ディスプレイに何かを表示したり、マウスやキーボードの入力を得ることになります。このような実行環境で用意されているプログラムの集まりのことを、ライブラリ（Library）と呼んでいます。Javaの場合は、クラスの集まりになっていますので、クラスライブラリ（Class Library）、あるいはパッケージ（Package）と呼んでいます。

１．５．１．代表的なクラスライブラリ

１つのJavaのプログラムを実行するために環境として、JavaVMとクラスライブラリが必要なことがわかりました。Javaを開発したSunMicrosystemsでは、どの環境でも用いることができる標準的なクラスライブラリを定義しています。大きくは、次の３つにわかれています。

J2SE（Standard Edition）	一般のアプリケーション、アプレットを動かすためのもの
J2EE（Enterprize Edition）	サーバなどで動かすためのもの
J2ME（Micro Edition）	組込み機器で動かすためのもの

本書では、この中でJ2SEに含まれているクラスライブラリを使ってプログラミングしていきます。MacOSXに関係するいくつかの代表的なクラスライブラリを以下に紹介していきます。

★Javaの標準的なクラスライブラリ
J2SEに入っているクラスライブラリで、どの環境でも実装されています（Java2DはMacOS 9までの環境では実装されていませんでした）ので、実行環境を気にせずに、一般的に用いることができます。

JFC（Java Foundation Class）	共通に必要なクラスライブラリ
AWT（Abstract Window Toolkit）	ウィンドウ、ボタンなどのGUIの表示のもの
Swing	AWTの拡張版
Java2D	2次元のグラフィックスの表示のもの

★その他の拡張されたクラスライブラリ
上記以外で、一般によく使われているクラスライブラリを取り上げてみました。ただし、Java3Dは、現在のところ、Mac OS Xではライセンス問題の関係で実装されていません。

SAX, Relax	XMLを扱うためのもの
Java3D	３次元グラフィックスの表示のもの

★Mac OS Xで使われるクラスライブラリ
Mac OS Xでよく使われるクラスライブラリを取り上げてみました。QuickTime for Javaや、OpenGL for Javaは、Mac OS Xだけでなく、WindowsやLinux用にも開発されていますので、MacOS固有のものではありません。

Cocoa Java	Objective-C言語用のCocoaというライブラリをJava用に移植したもの
QuickTime for Java	映像・音声のマルチメディアを扱うQuickTimeを制御するもの
OpenGL for Java	C言語用のOpenGL（３次元グラフィックス）をJavaで扱うためのもの

本書では、標準的なクラスライブラリとして、JFC、AWT、Swing、Java2Dを利用し、QuickTimeforJavaも少しだけ触れたいと思います。Cocoa Javaは、Interface Builderなどの開発ツールも連携して利用するのですが、かなりプログラミングの手法が異なりますので、別の本に譲りたいと思います。

１．５．２．クラスライブラリの形式とフォルダ

Javaのクラスライブラリは、下記の２つの形式で格納されています。共に、複数のクラスを１つのファイルの中に入れるというアーカイブ（Archive）の形になっています。

Zip形式	Windowsでよく用いられているアーカイブ形式で、.zipという拡張子がつきます。
Jar形式	Javaで定義されたアーカイブ形式で、.jarという拡張子がつきます。

どちらかの形式のファイルが、以下のフォルダか、そのサブフォルダに置かれています。一度覗いてみておいてください。

標準のクラスライブラリが置かれているフォルダ
　/System/Library/Frameworks/JavaVM.frameworks/Classes

Apple社のクラスライブラリが置かれているフォルダ
　/System/Library/Java

拡張用のクラスライブラリが置かれているフォルダ

/System/Library/Frameworks/JavaVM.frameworks/Home/lib/ext	JDK 1.3まで
/Library/Java/Extensions	JDK 1.4より