集中企画

石川和也

Javaのすべて
その実像に迫る

いま「Java」が注目されている。多くのインターネットの利用者が使っている「ネットスケープナビゲーター2.0」でも採用されていることは有名だし、さらに米国マイクロソフト社がライセンス供与を受けている。また、先月号で紹介した米国オラクル社が提唱している「ネットワークコンピューター（NC）」もJavaを基本とすると発表している。

そもそもは米国サン・マイクロシステムズが開発した開発者向けの「プログラミング言語」がなぜこれだけ社会的に騒がれるのだろうか？

今月の集中企画では、Javaの実像に迫ることにしよう。

まずパート1ではプログラムを書かない人でもJavaのなんたるかを理解していただき、パート2ではプログラミング言語としての特徴を解説、そしてパート3では開発環境を紹介、パート4では実際のアプレットを作ってみることにしたい。

PART 1
Javaはなぜこれほどまでに話題になるのか

「Java」はワープロや表計算ソフトのようなアプリケーションではなく、アプリケーションソフトを開発するのに使う「プログラミング言語」の1つだ。プログラミング言語と一言でいっても、多くの読者の人には馴染みがないと思うので、ここではまず、プログラミング言語とはなにかということから解説し、いま、なぜJavaが注目されているのかについて、アプローチしていこう。

Javaはプログラム開発のための「言葉」

◆◆◆◆◆◆

パーソナルコンピュータの黎明期には、「パソコンを使う」ということは、「プログラムを書く」ということでもあった。しかしアプリケーションソフトのパッケージが登場し、一般の利用者はプログラムなど書けなくても、コンピュータを使えるようになってきた。

それにもかかわらず、Java言語というプログラミング言語が注目されている。それはなぜなのだろうか？

プログラミング言語とは、コンピュータの利用者ではなく、プログラムを開発するエンジニアがプログラムを作成するのに使う「言葉」だ。おおざっぱにいえば、この「言葉」は人間の言葉に近い「コンピュータ語」だ。

一般にアプリケーションを開発する人がもっともよく使っているプログラミング言語は「C（シー）言語」や「C++（シー・プラスプラス）言語」というもので、この言語を使いこなすには、それなりのコンピュータの知識が要求される。

しかし、C言語について知らなくても、ワープロや表計算ソフトが使えるように、Java言語について知らなくても、Java言語で作成されたアプリケーション（アプレット）を使うことはできる。つまり、Java言語自体は開発者が使うもので、逆にいえば、Java言語を知らなくても利用者はアプレットを使うことができる。

Javaはアニメーションのための言語ではない

◆◆◆◆◆◆

もし、すでにネットスケープナビゲーター2.0を使っているようなら、http://www.javasoft.com/を見てみよう。ここにはJava言語で作られたアプレットのデモンストレーションが用意されている。確かにいままでのWWWブラウザーの中ではこうしたアニメーションを使うことはできなかったから、初めて見た人はきっと驚くことだろう。Java言語がこれだけ有名になった一因は、いままでのWWWではできなかった表現ができるようになったことだろう。しかし、Javaはあくまで「プログラミング言語」なので、誰でも簡単にプログラムを作れるというわけではない。

最近では、ネットスケープナビゲーターで絵を動かすものとして「ショックウェーブ・フォー・ディレクター」という米国マクロメディア社の開発したソフトが登場した。これは、従来CD-ROMを使ったマルチメディアタイトルを制作するためのアプリケーションである「ディレクター」で開発したデータ（作品）をWWWブラウザー上で再生するための「ネットスケープ・プラグイン」だ。ディレクターなら、難しいプログラミングなしに、絵を動かしたり、音を入れたりできる。

もちろん、ショックウェーブなどを使わなくてもネットスケープナビゲーターの機能の「プッシュプル」の機能を使い、コマ送りのイラストを次々と送り出して、あたかも絵が動いているかのようにもできる。

もし、WWWブラウザー上で絵を動かしたいだけなら、Java言語なんて難しいものには手をださず、ショックウェーブを使う方法を選んだほうがいいだろう。Java言語の本来の目的はそこにはない。

WWWの可能性を拡大するJava

◆◆◆◆◆◆

Java言語はネットワークを前提としたコンピュータ環境に劇的な革新をもたらすといわれている。なぜC言語やC++言語といったプログラミング言語の一種といえるJava

Javaアプレットのデモンストレーション。多くの人がWWW上のアニメーション制御機能だと勘違いした
URL http://www.nasoft.com/

ショックウェーブの作品例。ディレクターはあくまで画面指向のマルチメディア作品の開発環境だ
URL http://www.jtnet.ad.jp/WWW/SANRIO/s_gallery/xo_demo.html

言語がこれだけ注目されるのだろうか？ それに答えるまえに、WWWの仕組みについてもう一度触れておくことにしよう。

すでに多くの人はネットスケープナビゲーターなどのWWWブラウザーを使って世界中のコンピュータに入っている情報を見ることができていると思う。WWWはインターネット上で情報を共有する単純で効果的な手段を提供したが、WWW用の文書を作るのに使われるHTML（ハイパーテキスト・マークアップ・ランゲージ）は、論文や書籍のような「文書」の形での情報提供を中心に考えられていたために、表現できるのは「固定された文書」に限られていた。

もちろん、ヘルパーアプリケーションを使えば動画や音声も再生できたが「動いている」動画の内容はすでに1度録画され、固定された内容である。書籍に代表されるように、これまでの文書は作成されたときと同じ内容で固定されている。株価情報のように常に更新される内容の文書には「固定された文書」では役に立たない。

しかし、ネットスケープナビゲーターにJava言語を解釈し、動かすエンジンが組み込まれた結果、文書を表現する以外にアプリケーションプログラムを実行することが可能になり状況に応じて変化する「ダイナミック」な文書が表現できるようになった。さらにその実行環境、つまりWWWブラウザーのウィンドウの中が、ウィンドウズやマッキントッシュなどのデスクトップのようにプログラムが動く環境になったのだ。

Javaの使命はプログラム言語としての移植性と信頼性

WWWはネットスケープナビゲーターなどのブラウザーからの要求に応じて、対応する文書をネットワークを通じてブラウザーに送る。ブラウザーが要求するのは文字や画像などが中心だが、扱うのはあくまでファイル単位になっていた。

米国サン・マクロシステムズ社はここに着目した。つまり「WWWで文字や画像の代わりにプログラムを送ったらどうだろう？ そして送ったプログラムがWWWブラウザーの中で動いたとしたら…」という極めて単純なことだった。そこで、サン・マイクロシステムズ社内で研究開発が進んでいた情報家電向けのシステム（グリーンプロジェクトといわれていた）を開発するための、手軽で移植性や信頼性に優れたプログラミング言語を組み込むことにしたのだ。サン・マイクロシステムズは、この言語をブラシュアップし、移植性や信頼性などを実現した「Java言語」を開発した。

Java言語を使えばウィンドウズでも、マックでも動く

MS-DOS時代のコンピュータを知っている人なら、アプリケーションソフトウェアを買うときに、「MS-DOS用」というだけではなく、NECのPC-9801シリーズ用とか、IBM用とか、東芝のダイナブック用とかというように、対応するマシンによって選ばなければならなかった。

この理由はプログラムはCPUが理解できても、表示画面の性能が違うなど、ハードウェアの違いもあるので、同じプログラムは動かなかったからだ。もちろん違うCPUを使っているマシン、たとえばマッキントッシュでは動かない。しかし、ウィンドウズが登場して問題は改善された。同じインテル社のCPUを使っていれば、ハードウェアの違いは関係なくなり、「ウィンドウズ用」というアプリケーションを買えば、ウィンドウズの動いているマシンならどれでも動くようになった。

しかし、ウィンドウズ用のアプリケーションは相変わらずマッキントッシュでは動かないし、マッキントッシュ用のアプリケーションはウィンドウズでは動かない。もちろんUNIXとの関係も同様だ。

インターネットには、いろいろなコンピュータがつながっていることはご存知のとおりだ。文字や画像は、あらかじめ決まった形式があるので、その形式の文字や画像を使っているページなら、どのようなマシ

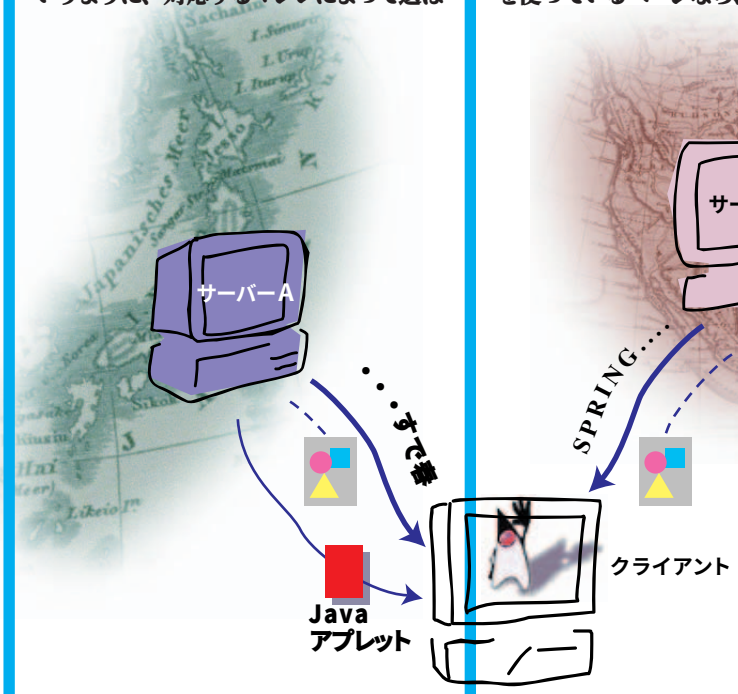

図1：WWWは世界中にちらばる文字データや画像を配送する仕組みだ。この仕組みを使ってプログラムが配送できたら画期的だ

ンやオペレーティングシステムを使っていても同じように表示できる。

プログラムもそれと同様に、1つのプログラムだけ作るだけで、インターネットにつながっているどんなハードウェアを使っていても同じように動いたら画期的だ。そうすればユーザーは便利になるし、エンジニアも同じ機能のアプレットを複数の機種に対応させるために作り替えなくてよい。

インターネットにつながるのは、パソコンやワークステーションだけではない。今後はさまざまな装置のつながる可能性があるので、なおさらどんなハードウェアでも動くことが望まれるのだ。

Javaは「言語」とそれを解釈する「エンジン」で構成される

そこでJava言語の意味が出てくる。「Java」は、プログラムを書く「Java言語」とJava言語で作られたアプレットを実行するための「エンジン（インタープリターという）」で構成される。エンジンはネットスケープナビゲーターなどのアプリケーションにあらかじめ組み込まれており、その組み込まれたところにアプレットがロードされると、そのアプレットが動きだす。

Java言語が注目されているのは、このようにJava言語を使ってアプレットを1つだけ作れば、マッキントッシュだろうと、ウィンドウズだろうと、UNIXだろうと、Java言語のアプレットを動かす「エンジン」さえ組み込まれていれば、どのような環境でも動かすことができるというところだ。

現在はネットスケープナビゲーターやHotJavaブラウザーなどの一部のアプリケーションにしか組み込まれていないが、電子手帳などの他の装置でも、エンジンを組み込めば同じアプレットが動くことになる。

Javaのアプレットはコンパクトだ

このように、オペレーティングシステムやハードウェアなどに依存しないのは、Java言語で作られたアプレットが「バイト・コード」というJavaの「エンジン」が理解する形式（これを「中間言語」という。人間の言葉でもなければ、コンピュータが直接理解できる言葉でもない「中間的な」言語になっているからだ）になっているからだ。

また、Java言語で作成されたアプレットは「必要に応じて、部品化されたプログラムをネットワークでダウンロードして、実行する」という機能があるので、関係するプログラムの全部を一度にダウンロードするのではなく、必要に応じて必要なプログラムだけをダウンロードして、動かすことができる。たとえばワープロソフトは編集、検索、文字の配置などのさまざまな機能を持っているが、これらの機能をそれぞれ個別の「部品（パーツ）」として切り分け、必要な部分を読み込んで実行させることができる。そのため、ハードウェア自体もたくさんのメモリーやディスク容量を必要としない。

たとえばJava言語はエンジンであるインタープリターのサイズが100キロバイト程度、標準的なクラスライブラリという部品を含めても500キロバイト程度とコンパクトになっている。そのため、最近話題の500ドルコンピュータなどでも動作する可能性が十分にあるといわれている。

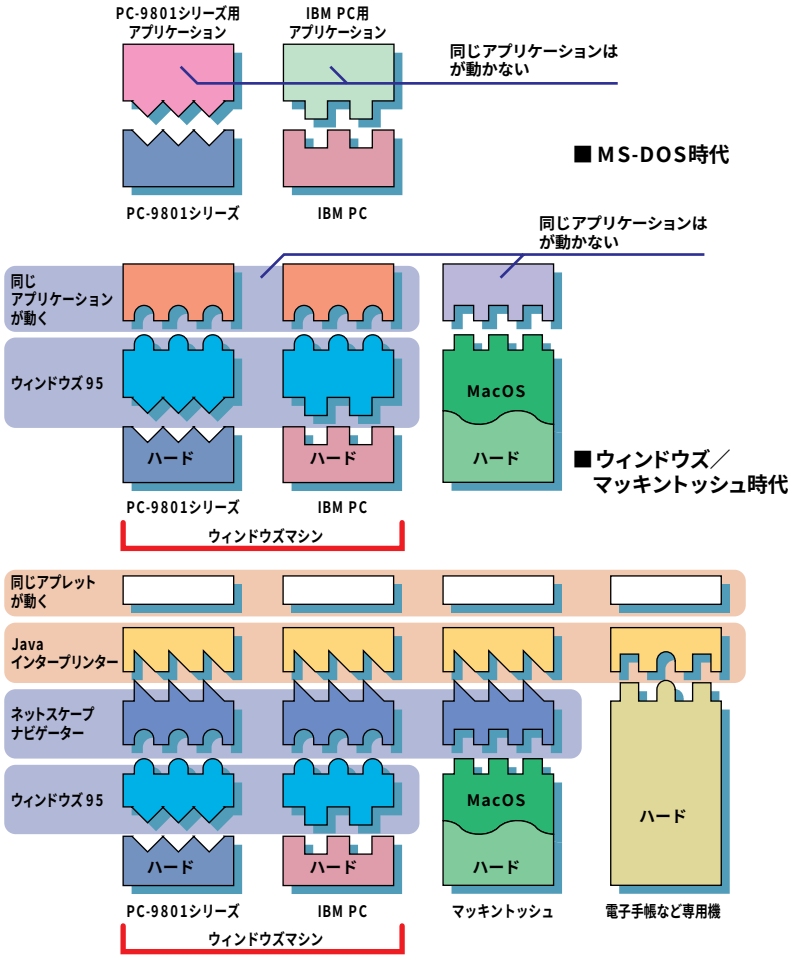

図2：Javaはハードウェアやオペレーティングシステムに依存しないので、インターネットでプログラムを配送するのに向いている

WWWとの組み合わせで威力を発揮する

◆◆◆◆◆◆

Java言語は、まず「HotJava」というWWWブラウザーに組み込まれた。このHotJavaというブラウザーはサン・マイクロシステムズ社が開発したWWWブラウザーだ。HotJava自体も大部分がJava言語で作成されている。HotJavaはWWWが備えている「ネットワーク上での部品の識別（URL）」機能と「ネットワーク経由での配送（ハイパーリンクとHTTP）」機能を活用して、すでにある環境下で手軽にネットワークコンピューティングが実現できることを具体的に見せた。ネットスケープコミュニケーションズ社はこの可能性にいち早く気付き、自社の「ネットスケープナビゲーター」へJava言語を取り込むことを決めた。

もちろん、Java言語はWWWとだけ組み合わせて利用されるための言語ではないが汎用的なネットワークを使った配送システムであるWWWと世界的な標準となっているネットスケープナビゲーターを従来のハードウェアやオペレーティングシステムなどと同様に「プラットフォーム」と見立て、その上で動作するネットワークアプリケーションを動作させることはJava言語にとっては有効なアプローチだろう。

つまり、絵を動かして、音を鳴らすだけではなく、コンピュータではプログラムさえ書けばなんでもできるように、Java言語でもプログラムさえ書けば、なんでもできるわけだ。プログラミング技術自体は難しいが、最初に紹介したショックウェーブやプッシュプルを使うよりも高い柔軟性と可能性があるといえる。いずれはネットスケープナビゲーターの中でワードプロセッサーが動くようになるかもしれない。

Java言語が騒がれる理由とは

◆◆◆◆◆◆

では、いよいよ最初の疑問に答えることにしよう。「Java言語はなぜいまこれだけ騒がれているのだろうか？」。

大きく分けると2つの理由が考えられるだろう。1つは、いままで説明したように、Javaのインタープリターを採用したWWWブラウザーやネットワーク機能をもったコンピュータがあれば、ネットワークからアプレットを読み込んで実行できるようになると、現在繰り広げられている「ウィンドウズ対マッキントッシュ」というような論争に終止符を打ち、どちらでも好きなほうを選べばアプリケーションが共通になってしまう「かもしれない」ということだ。

つまり、アプリケーションを動かすための基盤となるオペレーティングシステムで大きなシェアをとったマイクロソフトが業界の覇権を握ったように、Javaというアプリケーション実行環境（つまりはオペレーティングシステムと同じくらいの意味を持つということ）で大きなシェアを握った会社がインターネット時代のコンピュータ業界の覇権を握る可能性があるということを意味している。

もう1つの理由は、インターネットとそれを使うWWWやブラウザーという仕組みを使ってアプリケーションを配信し、しかもそのアプリケーションがさらに通信をしながらいろんな仕事をすることになると、いままで想像していなかったような利用方法が開発されるのではないかという期待だ。

一方で、このような思惑ばかりが先行してしまい、その周囲が異様に盛り上がりすぎているという意見もある。そもそもこの言語を開発したサン・マイクロシステムズのビル・ジョイ氏はあくまで「C言語、C++言語に対抗できる新しいプログラミング言語」と位置づけていたものが、インターネットでアプレットを配布するというアイデアにより、いっきに表舞台に立ってしまった感もある。

結果として、インターネットの可能性を拡大するものではあるので、大きな期待はよせるものの過剰に騒ぎすぎることは逆効果でもあるだろう。

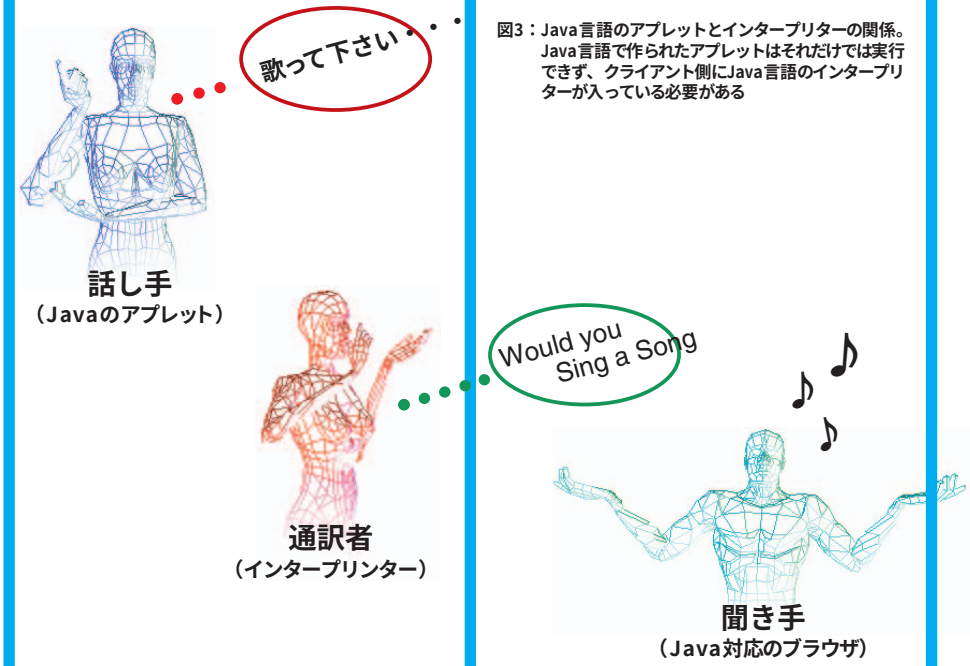

図3：Java言語のアプレットとインタープリターの関係。Java言語で作られたアプレットはそれだけでは実行できず、クライアント側にJava言語のインタープリターが入っている必要がある

PART 2
プログラミング言語としての特徴

Java言語はオブジェクト指向言語として新たに開発された。文法的にはC言語やC++言語に非常によく似ている。現在、多くのアプリケーションプログラムやシステムがC言語やC++言語を使って開発されているので、多くのプログラマの人々にとっては馴染みやすいだろう。ここではJava言語とC言語、C++言語との違いについて見てみよう。

「ポインター」がない

Java言語にはポインターがない。そのため、メモリーをアドレス指定によって直接は操作できない。ポインターはプログラミングの柔軟性を向上させるが、弊害もある。

ポインター操作を誤り、配列の範囲外にアクセスしてしまうことはよくあることだ。また、初期化していないポインターの内容でアクセスをし、予期しないメモリー領域を破壊してしまうこともある。このようなポインターに起因するバグはコンパイラーも検出しないし、実行時のスタックやメモリーの内容によってエラーが発生する場所が違うので、問題の個所の特定が非常に困難だ。

Java言語ではポインター操作自体をなくすことで、ポインター操作の誤りから発生するプログラムのバグを未然に防止する。

限定された型変換

ポインター同様に型変換（キャスト）もバグの温床になっている。Java言語では型変換はサポートしているが、同一のクラスから派生したクラス間での型変換に限定されている。そして型変換の正当性はコンパイル時にチェックされ、不正な型変換はできない。

構造体、共用体がない

また、複数のデータをひとまとめにして管理する構造体（structやunion）もない。オブジェクト指向ではデータとその操作（メソッド）を組にしてオブジェクトとして定義している。構造体を作成するということは、新しいデータ型を作り出すことになるから、同時にそのデータを操作するメソッドを作成し、それらをクラスとして合わせて定義する。

規定された基本データ型

C言語では基本的なデータタイプ（int、char、short、logn、float、doule）のサイズは言語仕様上は明確に決められていなかった。これはC言語でそれぞれのハードウェアアーキテクチャにとって自然なデータサイズを提供できるというメリットがあったが、一方で移植性の悪いプログラムを生む原因にもなった。特に最も頻繁に利用されるint型のデータサイズがCPUアーキテクチャに依存するために、16ビットアーキテクチャで設計されているウィンドウズと、32ビットアーキテクチャが一般的なUNIX間でプログラムの移植をする場合に苦労の原因となっていた。

Java言語ではこれらの基本データ型のサイズを明確に規定するとともに、格納されるデータフォーマットについても規定している。このなかで文字型が16ビットで格納されるコードセットがユニコード（unicode）という点はCプログラマ（特にUNIX）が戸惑う点かもしれない。また、「unsigned（符号なし）」という修飾詞がない。そのため右シフト演算（>>）時にキャリアビットを無視する符号なしの右シフト演算子（>>>）が追加されている。

定義された文字列

C言語では文字列は文字データの配列（NULL文字で終了する）として定義されている。そのため、文字列の操作はメモリー上に並んだ配列に対する操作として実現される。文字列の複製や連結を実現するためには文字配列に対する処理（strcpy, strcat等）をする必要があった。

Java言語では文字列は「String」や「StringBuffer」オブジェクトとして定義される。これらのクラスでは文字列の複製や連結が定義されており、「＋」演算子によって2つのString（StringBuffer）オブジェクトを連結できる。さらに「＋」演算子で連結をする場合に整数型の変数などは

表1：C言語とJava言語の比較

項目	C++言語	Java言語
ポインター	ある	ない
構造体	ある	ない（クラスを定義）
文字列操作	文字型の配列操作関数 strcat()	String, StringBufferオブジェクトの操作 str + str_b
多重継承	ある	なし
アクセス制御	static, auto	public, private, private protected,protected
ネットワーク	ある	ある（セキュリティ制限有）
配列	境界チェックなし	境界チェックあり
例外処理	signal（オプション）	try { ～ } catch （例外）{例外処理}（必須）
関数の引数	可変長ANSIコンパイラは型のチェックを行う	メソッドで定義されたものだけ型と数が一致する必要あり

printf関数での出力書式指定と同様に対応する文字列に変換して連結される。

また、実行時に文字列領域をチェックするので、領域外に不正にアクセスすることも防げる。

ポインターを使わない関数呼び出し

C言語では関数への引数は「値呼び出し（call by value）」で行われるので、データはコピーして渡される。呼び出された関数は常に複製されたデータを操作しているので、引数はローカル変数（初期化された）と同様に扱える。呼び出し側の変数の内容を変更させたい場合には、変数のポインターを引数として渡す。

Java言語では「参照呼出し（call[pass] by reference）」で引数が渡されるのでポインターによって操作する必要はない。

また、C言語では関数に渡す引数の数は自由だが、Java言語では関数（メソッド）定義時に指定された型および数を持つ呼び出ししか許可されない。定義されていない呼び出しをするとコンパイル時にエラーとなる。

PART 3
Java言語のプログラムの開発環境

ここではJavaに興味を持たれた方が実際にプログラムを作る際の手順について説明するとともに、自作のJavaアプレットやインターネットで提供されているJavaプログラムを動作させるために必要な手順を説明する。

Javaプログラムの種類

Java言語で作成するプログラムは大きく分けると2種類ある。1つは「アプレット」と呼ばれネットスケープナビゲーターやHotJavaといったJava対応のWWWブラウザーで動作するアプリケーションだ。アプレットはJavaのインタープリターを組み込んであるWWWブラウザーで動作するので、実行させるアプレット名や表示サイズ、引数を指定するためのHTMLファイルを必要とする。また、数値演算やデータベースへのアクセス処理などの一般的なプログラム処理に加えて、描画や文字列の表示、マウスやキーボードからの入力処理といった画面表示や利用者との対話をするアプリケーションの開発ができる。

Javaアプリケーションというと WWWブラウザー上で動作するアプレットが一般的だが、WWWブラウザーがなくても単独で動作するアプリケーションの構築もできる。これを「スタンドアローン」という。この「スタンドアローン」のアプリケーションを実行させるためには、Java言語で書いたプログラムの実行環境であるJavaインタープリターとともに動作させる。

Javaアプリケーションの開発

Javaで書かれたプログラムは、いわゆる「バイトコード」という中間言語，つまりマシンコードでもなければ、高級言語でもない、その中間的な形式に変換され、仮想マシン上で実行される。仮想マシンとは、CPUやオペレーティングシステムに直接は依存しないプログラムの実行環境のことだ。そのためBASIC言語とは違い、プログラムを作成（編集）しながらの実行はできない。C言語やC++言語などのコンパイラー言語と同じように、ソースコードを書いたらコンパイルをして、バイトコードに変換しなければならない。そのためのJavaの開発環境としては、いくつかの商品が発売されはじめているが、ここではサン・マイクロシステムズ社のJDK（Java Development Kit）での開発方法について説明しよう。

JDKの概要

JDKは、Sunワークステーション（ソラリス2.*x*）用、ウィンドウズ95/NT用、そしてマッキントッシュ用という3つのプラットフォームが用意されている。

このJDKには、Javaコンパイラーである「javac」、デバッガーである「jdb」、そして「jdoc」などが入っている。これらのツールは「http://www.javasoft.com/devcorner.html」でそれぞれのプラットフォームに対応したJDKを入手できる。

JDKのインストール

JDKを入手し、適当なディレクトリに展開したら、JDKを利用するための環境設定をする。このなかで環境変数「CLASSPATH」が設定がされていないと、コンパイル時や実行時に「クラスが見つからない」というエラーが発生する場合がある。

プログラム開発の手順

「アプレット」と「スタンドアローン」では開発手順は若干異なる。まず「スタンドアローン」の開発手順から紹介しよう。Javaでのプログラム開発はC言語での開発と同じで、編集→コンパイル→テスト→実行を繰り返す。

プログラムの編集（作成）は一般的なテキストエディタを利用する。ただし、ファイル名には決まりがあり、プログラムのソースファイルはそのファイルに書いてあるクラス名に拡張子に「.java」を付けたファイル名になる。ファイル名での大文字と小文字の区別は厳格に「ある」。Spining-Globe.javaとspiningglobe.javaは別のクラスを定義しているファイルとして扱われる。特にウィンドウズ95/NTの場合は注意が必要だ。

サンプルプログラム

C言語を習ったときに最初に書いたように、「Hello World」プログラムを書いてみよう。これは説明するまでもないが、画面に「Hello World」という文字を表示するだけのものだ。

```
HelloWorld.java
class HelloWorld {
public static void main(String args[]) {
System.out.println("Hello World");
}
}
```

スタンドアローンのプログラムはクラスの集合として書く。そしてクラス定義にある「mainメソッド（関数）」がインタープリターから最初に呼び出される。

コンパイル

Javaソースコードを書いたら、つぎにコンパイルをしてバイトコードに変換する。コンパイルは「javac」コマンドを使う。

```
C> javac HelloWorld.java
```

この場合、ソースファイル名の拡張子は省略できず、また大文字と小文字は区別されるので注意が必要だ。

コンパイラーはクラス定義をバイトコードに変換した「クラスファイル」を作成する。クラスファイルは各クラスの名前に「.class」拡張子を付けたファイル名となる。1つのJavaプログラム内に複数のクラスが定義されていた場合、コンパイラーはそれぞれのクラス名を持つクラスファイルとして別々のファイルを作成する。

Javaコンパイラーが作成したクラスファイルはCコンパイラーが作成したバイナリとは異なり、できあがったバイトコードのファイルだけでは動かない。動かすためにはJavaインタープリターを使って作成されたファイルを実行させることになる。JDKに入っているインタープリターは「java」コマンドで起動でき、その引数としてクラス名を指定する（「.class」が付いたクラスファイル名ではない）。

```
C> Java HelloWor
```

アプレットの作成

つぎにアプレットを作成してみることにしよう。基本的にはスタンドアローンと同様の手順で作成するが、ホームページにアプレットを貼り付けるためのHTMLファイルも並行して作成する必要がある。

プログラムの作成方法とコンパイル方法についてはスタンドアローンと変わりはなく、クラス名に「.java」拡張子を付けたファイルを作成しそれをコンパイルしてクラスファイルを作成する。アプレットのプログラムは「Appletクラス（正確にはjava.applet.Appletクラス）」のサブクラスとして定義する。スタンドアローンでは最初に起動されるクラスを識別するためにmainメソッドを書く必要があったが、アプレットではHTMLによって最初に呼び出されるクラスを指定しているのでmainメソッドを書く必要はない。その代わりにブラウザによってアプレットが呼び出された場合の処理や終了時の処理、さらには別のページに移動したときの処理などを書けるようになっている。

アプレットの「Hello World」プログラムは下のリストのとおりだ。

コンパイルはつぎのようなコマンドラインで行う。

```
C> javac HelloWorldApplet.java
```

Java言語で書いた「Hello World」（アプレット）

```
HelloWorldApplet.java
import java.awt.Graphics;
import java.applet.Applet;
class HelloWorldApplet extends Applet {
public void init(){
}
public void start() {
}
public void stop() {
}
public void destory(){
}
public void paint(Graphics g) {
g.drawString ("Hello World", 10, 10);
}
}
```

これらはなくてもよい。ここでは例としてあげている。

アプレットを埋め込んだHTMLファイルの例

```
<HTML>
<HEAD>
<TITLE>Javaのサンプルプログラム</TITLE>
</HEAD>
<BODY>
<APPLET CODE="HelloWorldApplet.class"
WIDTH=100 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

HTMLファイルの作成

アプレットをホームページに貼り付ける方法はイメージ（画像）ファイルをホームページに貼り付けるのと同様にHTMLファイルで貼り付けたいアプレットのクラス名を指定する。アプレットの貼り付けには「<APPLET」タグを使う。

貼り付けには、拡張子「.class」を含んだクラスファイル名を指定する。CODEBASEが省略された場合には、HTMLファイルと同じディレクトリにあるアプレットが呼び出される。URLも指定できるが、セキュリティ上の理由から、アプレットをロードしたホスト以外のホストを指定しても実行時に拒否される。

なお、次のパート4以降ではHTMLファイルを作成する必要があるが、この記事は言語仕様を中心とするため、省略している。

アプレットの実行

アプレットを実行させるためには、WWWブラウザーでアプレットを貼り付けているHTMLファイルを呼び出す。現時点ではネットスケープナビゲーター2.0とHotJavaブラウザーが対応している。ただしネットスケープナビゲーターとHotJavaブラウザーでは実行できるアプレットのバージョンが異なる。またJDKにはアプレットのテスト用に「appletviewer」が用意されており、これを利用するとアプレットだけのテストができる。

ネットスケープナビゲーターでのアプレットの実行

ネットスケープナビゲーターを使ってJavaのアプレットを実行する場合には、「option」メニューの「Network and Security」内の「disable java（Javaを無効にする）」が指定されていないことを確認する。また、β版ではURLとして「http://...」のみが有効だった時期があった。そのためWWWサーバーがない状態ではテストが困難だったが、現在ではURLに「file://」（もしくは「FILE」メニューの「File Open」から）を使用できるので、JDKとネットスケープナビゲーターを入手するだけでJava言語を使ったプログラムの開発ができるようになった。

PART 4
プログラム開発の実際

それでは実際にJavaプログラムのソースコードを見ながら、Javaプログラミングのエッセンスをつぎの4つに分けて解説しよう。

1. 文字と画像の表示
2. キーボード／マウスといったユーザーとの対話に必要なイベント処理
3. スレッドを利用した動きのあるアニメーション
4. クライアント・サーバー型のネットワークアプリケーション

このほかにJDKにはサンプルアプレットのソースコードが提供されているので参考になるだろう。

これらのプログラムはJDK 1.0を利用してウィンドウズ95の環境下で動作の確認をしてある。アプレットの実行はネットスケープナビゲーター2.0、JDKに付属しているアプレットビューアの両方での動作を確認してある。ただし、「ネットワークアプリケーション」については「appletviewer」でのみ動作する。

なお、ここでの解説はC++言語での開発経験のある方を対象としているので、メソッド、クラス、継承などの用語については、あらためて解説しない。他のオブジェクト指向関係の資料を参考にして欲しい。

表2：Java言語の修飾子

クラス定義	
public	パッケージとして別のクラスからもアクセスが可能なクラス（一般的）
abstruct	abstructメソッドを含んでいるクラス定義
final	これ以上派生させることのできないクラス定義

変数およびメソッド	
public	どのクラスからでも参照可能
protected	同じパッケージのクラスもしくは他のパッケージで定義されているサブクラスから参照可能
private protected	所属するクラスのサブクラスからのみ参照可能
private	所属するクラス定義内でのみ参照可能
static	インスタンスには所属せず、クラスに1つだけ存在する。 staticメソッドからインスタンス変数は参照できない

文字と画像の表示

まずは、描画プログラムを例に基本的なアプレットプログラムの構成を見ていこう（プログラム❶）。アプレットは「Applet」

クラスのサブクラスとして定義する。サブクラスの定義は定義するクラス名の後ろに「extends親クラス名」と記述する。

```
class Draw extends Applet {
// クラスの定義
}
```

Appletクラスは自分で作成したものではなく、JDKで提供されているクラスを使う。このように別に定義されているクラスを使用するために「importパッケージ名」文を使う。Javaでは複数のクラス定義をひとまとめにしたものを「パッケージ」と呼ぶ。C言語などでのライブラリと似ている。パッケージは環境変数CLASSPATHで指定されるディレクトリから参照される。

それでは実際のクラス定義を見てみよう。クラス定義の中は「変数（データ）」定義と「メソッド（関数）」定義に分けられる。変数の定義はクラス定義内のどこでも行え、初期化が必要であれば「＝」で初期値を定義する。

配列はオブジェクトとして扱われるので、「new」演算子によって実際の領域を確保する必要がある。配列の添え字は0から始まり（要素数-1）までが有効で、範囲外にアクセスしようとすると、実行時エラーになる。

```
Color colors[3];   //この記述は許されない
Color colors[] = new Color[3];   //このように定義する
Color colors[] = {Color.red, Color.green, Color.bule};      // 初期化をする
```

クラス定義や変数、メソッドには有効範囲や定義の性格を決めるための修飾子がある。それぞれの定義文の先頭に修飾子を付加する。

アプレットの親となるAppletクラスには、アプレットの動作が始まりと終わりに呼び出される4つの基本的なメソッドを持っている。これらのメソッドはデフォルトでは何の動作もしないので、派生したクラス内で同じ名前のメソッドを定義（これをオーバライドと呼ぶ）することでアプレットの開始と終了の処理を定義できる。ここでは、initメソッドだけをオーバライドし、イメージファイルの読み込み処理をしている。

アプレットで描画処理を行うための3つのメソッドが用意されている。repaintメソッドはオーバライドすることはできないが、paintメソッド、repaintメソッドは必要に応じて表示処理を記述する。

paint()　アプレット内の表示領域全体を描画するリフレッシュ処理をする。この処理ではバックグラウンドを消してから書き直すので、上書きを行いたい場合には、下記のupdateを利用する。アプレットが起動した際に呼び出されるので、存在しない変数などを参照しないように注意が必要だ。

update()　アプレットの表示領域に上書きで表示を行う。

repaint()　画面表示を行うために他のメソ

表3：Appletクラスの4つのメソッド

init()	アプレットがブラウザーに読み込まれたときにだけ実行される。例のような描画に必要なイメージの読み込みなど、データ初期化などをすることが一般的だ。
start ()	アプレットが起動されたときに呼び出される。ブラウザー上でホームページ間を移動しアプレットのあるホームページに戻ってきたときにも呼び出される。
stop()	アプレットが停止されたときに呼び出される。ブラウザーで別のホームページに移動したときに呼び出される。アニメーションや音声などを再生している場合にはそれらを停止させるために利用することが多い。
destory()	アプレットが破棄されたときに呼び出される。メモリー解放はシステム側で行われるが、ネットワーク接続の終了処理などが行われることが多い。

プログラム❶　文字と画像の表示プログラム

```
import  java.applet.Applet;     // アプレットの作成に必要
import  java.awt.*;             // 描画処理に必要

public class Draw extends Applet {      // クラス定義
        Image                   image;
        Color colors[] = {Color.red, Color.blue, Color.green};  // 初期化された配列
        int num_colors = colors.length;  // 配列の要素数の取得
        String string = "Hello World!";
        char    chars[];

        public void init () {
                // HTML 内の APPLETタグで指定されたイメージファイルを読み込む
                image = getImage (getDocumentBase (), getParameter ("image"));

                // Stringクラスの内容をキャラクタ配列に書き込む
                chars = new char[string.length()];
                string.getChars(0, string.length(), chars, 0);
        }

        // 描画処理。毎回背景をクリアしてから表示が行われる。
        public void   paint (Graphics g) {
                int x, y;
                FontMetrics     metrics = g.getFontMetrics();

                // 1文字毎に色を変えて表示を行う。
                x = 10; y = 30;
                for (int i = 0; i < string.length(); i++) {
                        g.setColor (colors[i % num_colors]);    // 色の指定
                        g.drawChars (chars, i, 1, x, y);              // 文字表示
                        x += (metrics.charWidth(chars[i]) + 1); // 文字幅を基に文字送り
                }

                // 四角を書く
                g.drawRect (10, 50, 30, 30);
        g.fillRect (60, 50, 30, 30);

                // 円を書く
        g.drawArc (10, 120, 30, 30, 0, 360);
        g.fillArc (60, 120, 30, 30, 0, 360);

                // イメージを書く
                g.drawImage (image, 10, 160, this);
        }
}
```

ッドから呼び出される。再表示を行うためにupdateメソッドを呼び出す。updateメソッドが定義されていない場合にはpaintメソッドを呼び出す。

キーボード／マウスのイベント処理

◆◆◆◆◆◆

つぎのような条件を満たすアプリケーションを作成してみる（プログラム❷）。

❶ マウス操作でブラウザーの画面上に4角形を書く。マウスがクリックされたポイントからドラッグし、ボタンが離された位置まで四角を表示する。ドラッグ中の表示の変化も表示する。

❷ キーボードから入力された文字をブラウザー上のマウスでクリックされた位置に表示する。

まず、ブラウザ上で表示をするので、「Applet」クラスから派生させる。イベントと呼ばれるマウスやキーボードからの入力を処理するためのパッケージが「java.awt.Event」として用意されているので、これをimportする。それ以外のグラフィック関連のパッケージについては「java.awt.*」のようにパッケージ名にワイルドカードを使うことで、必要なパッケージをコンパイラが判断して読み込むように指示する。

今回はstartメソッドを定義した。ここで、背景色をわかりやすくするために黄色に定義した。initメソッド内では、描画に必要な準備が整っていない場合があることと、別のホームページから戻ってきたときに背景を表示し直す必要があるので、ここで定義した。

一般的な入力イベントを取り扱うメソッドには以下のようなメソッドがある。

マウスのボタンの数はプラットフォームによって異なることがある。マウス関係のイベントはどのボタンが操作されたかに関係なく呼び出される。そのためボタンを区

プログラム❷　マウスとキーボードからの入力を処理するプログラム

```
import java.applet.Applet;
import java.awt.Event;     // イベント処理用のパッケージ
import java.awt.*;

public class EventSample extends Applet {
            Color   fgColor;
              int           upperX, upperY, downX;
              int           lowerX, lowerY, downY;

            static final    int maxChars = 1024;    // 定数と同様に使える
            char    inputChar[];
              int           numChars;

            boolean isText = false;

            public void init () {
              inputChar = new char[maxChars];
              upperX = upperY = lowerX = lowerY = 0;
            }

            public void start () {
                        setBackground(Color.yellow);    // 背景を設定
            }

            // 表示処理。指定によって文字と四角をかき分ける
            public void     paint (Graphics g) {
                        g.setColor (fgColor);
                        if (isText) {
                                    // 文字の表示
                                    g.drawChars (inputChar, 0, numChars, upperX,
upperY);
                                    isText = false;
                        } else {
                                    // 四角の表示
                                    g.drawRect (upperX, upperY, lowerX - upperX,
lowerY - upperY);
                        }
            }

            // キーが話されたときの処理
            public boolean  keyUp (Event event, int key) {
                        if (numChars == maxChars)
                                    numChars = 0;
                        fgColor = Color.red;
                        inputChar[numChars++] = (char) key;     // 文字型へ変換
                        isText = true;
                        repaint ();
                        if ((char)key == '\n') {
                                    numChars = 0;
                        }
                        return true;
            }

            // マウスが押されたら初期座標を移動
            public boolean  mouseDown (Event event, int x, int y) {
                        numChars = 0;
                        upperX = lowerX = downX = x;
                        upperY = lowerY = downY = y;
                        return true;
            }

            // マウスが離されたら四角を決定
            public boolean  mouseUp (Event event, int x, int y) {
                        fgColor = Color.black;
                        updatePosition (x, y);
                        repaint ();
                        return true;
            }

            // マウスがドラッグされている間は緑で表示
            public boolean  mouseDrag (Event event, int x, int y) {
                        fgColor = Color.green;
                        updatePosition (x, y);
                        repaint ();
                        return true;
            }

            // このクラス定義でのみ参照できるメソッド
            // 座標位置を更新する
            private void    updatePosition (int x, int y) {
                        if (x < downX) {
                                    upperX = x;
                                    lowerX = downX;
                        } else {
                                    upperX = downX;
                                    lowerX = x;
                        }

                        if (y < downY) {
                                    upperY = y;
                                    lowerY = downY;
                        } else {
                                    upperY = downY;
                                    lowerY = y;
                        }
```

別した処理をしたい場合は、メソッドに渡される引数のイベントオブジェクトを検査することで、どのボタンが操作されたかを判断する必要がある。今回は押されたボタンの種類に関わらずに処理をしている。

mouseDown、mouseUp、mouseDragの各メソッドをオーバライドしてマウス処理をしている。それぞれのメソッドにはマウスが操作された位置がアプレット領域の左上を（0，0）として渡されるので、マウスの操作に応じて四角を表示するために必要な座標の計算をするための内部だけで有効な（private）メソッドで定義されているupdatePositionメソッドを呼び出す。そしてrepaintメソッドを呼び出して表示処理をする。

キーボード入力はkeyUpメソッドを定義することで受け取れる。ここではpaintメソッドで表示をするので、表示内容は常にクリアされてから表示されてしまう。そのため、これまでに入力した文字を覚えておくための配列を用意している。9行目で配列の宣言をしているが、その領域の最大値を宣言するために12行目で「maxChars」という変数を定義している。この変数は「static final int」と定義することで定数のように、変更できない変数として定義できる。

マウス処理とキーボード処理で表示処理のためのpaint処理を共有するので、文字表時か四角表示のどちらかを区別するための変数を設けている。それぞれのイベント処理で表示をすることも（がんばれば）可能だが、画面のリフレッシュ処理等を考えると表示の処理は集中させる方がよいだろう。

スレッドを利用した動きのあるアニメーション

典型的なJavaアプレットというと「手を振っているDUKE」だろう。一般にアニメーションは、「パラパラまんが」のように、少しずつ違う絵をつぎつぎと表示して画像

表4：入力イベントを処理するメソッド

mouseDown	マウスのボタンを押したとき
mouseUp	マウスのボタンを離したとき
mouseDrag	マウスをドラッグ（ボタンを押したままで移動）したとき
mouseEnter	マウスがアプレットの領域に入ったとき
mouseExit	マウスがアプレットの領域から出たとき
mouseMove	マウスを移動したとき
keyDown	キーを押したとき
keyUp	キーを離したとき

プログラム❸　スレッドを処理するプログラム

```
import  java.applet.*;
import  java.awt.*;

// アプレットのサブクラスでスレッド処理を行うには「Runnable」インターフェースをつかう
public class    Animation extends Applet implements Runnable {
                Image       img[] = new Image[10];
                Thread      tid = null;
                int                 index = 0;

                // アニメーション用にイメージを読み込む。
                public void init() {
                                for (int i = 0; i < 10; i++)
                                                img[i] = getImage(getDocumentBase(), "images/T" + i + ".gif");
                }

                // アプレットが起動されたらスレッドを作成し起動させる。
                public void start() {
                                if (tid == null) {
                                                tid = new Thread(this);
                                                tid.start();
                                }
                }

                // アプレットが停止した起動しているスレッドも停止させる。
                public void stop() {
                                if (tid != null) {
                                                tid.stop();
                                }
                                tid = null;
                }

                // スレッドとなる処理。
                // 200ms 休止し次のイメージを表示させる
                public void run() {
                                while (true) {
                                                if (index == 10)
                                                                index = 0;
                                                repaint();
                                                try {
                                                                tid.sleep(200);
                                                } catch (InterruptedException e) {
                                                }
                                                index++;
                                }
                }

                // スレッドで指定されたページのイメージを表示させる
                public void paint(Graphics g) {
                                g.drawImage(img[index], 10, 10, this);
                }
}
```

表5：スレッド関連のメソッド

start	スレッドの開始。このメソッドが呼び出されることによりrunメソッドで記述されたスレッドとしての処理が起動される
stop	スレッドの停止
run	スレッドとして独立して動作する処理を記述
suspend	スレッド処理の一時停止
resume	停止しているスレッド処理の再開
sleep	スレッドの休眠（ミリ秒単位）

が動いているように見せかける。もう1つのやり方は、演算によってグラフィックスを処理して画像を動かす方法である。

この処理をするためには必要な画像の作成や表示という処理と一定時間経過するまで待つという2つの処理を同時にしなければならない。画像を単純に繰り返し表示しているだけでは、実際いつ画面上に表示結果が反映されるかわからないために、処理のタイミングに依存してしまうか、最悪の場合にはその処理から抜け出さないので、実際には画面上に何も表示されないかもしれない。

このような複数の処理をするためにJava言語では「スレッド」という概念が用意されている。スレッドのためには「Thread」クラスが提供されているが、今回はアニメーションを表示するためにアプレット上でスレッドを使わなければならない。ただしアプレットでスレッドを使うには注意が必要だ。

アプレットとして使うためには「Applet」クラスのサブクラスとして定義する必要がある（プログラム❸）。Java言語では複数の親を持つサブクラスは生成できない（多重継承ができない）。これを回避するために「Runable」インターフェイスが用意されているので、これを使ってクラス宣言をする。

スレッドにはいくつかのメソッドが用意されている。これをオーバライドすることでスレッドを操作する。

この例で説明しているstartメソッドとstopメソッドは、Appletクラス用のメソッドを定義している。アプレットが起動すると自分自身「Thread」を作成し、そのスレッドを開始させる。スレッドとして別の動作になる処理をrunメソッドで定義している。このメソッドは一定期間（200ミリ秒）だけ休眠し、つぎの画像の表示の表示処理を呼び出している。

このようにして表示処理と時間待ち処理を別々に動作させることができる。

プログラム❹　サーバー側プログラム

```
// ネットワークに必要なパッケージをロードする。
// アプレットではないので java.applet.* をロードする必要はない
import java.net.*;
import java.io.*;

// このプログラムはアプレットではなく「スタンド・アローン」で動作する。

// 入力を行うためのスレッドを定義
class serverIO extends Thread {
        DataInputStream       input;

        // コンストラクタを定義
        public serverIO(Socket so) {
                try {
                        // 入力を受けるためのデータストリームを作成
                        input = new DataInputStream (so.getInputStream());
                } catch (IOException e) {
                }
        }

        // スレッドのメイン処理
        // クライアントからの入力を受け取り、1文字ごとに表示を行う。
        public void     run() {
                try {
                        while (true) {
                                byte    c = input.readByte();
                                System.out.write((int)c);
                                System.out.flush();   // バッファをフラッシュ
                        }
                } catch (IOException e) {
                }
        }
}

// 起動されるクラス。 mainメソッドが呼び出される。
public class serverSide {
  static public void main(String args[]) {
  serverIO    tid;
    try {
      // 最初の引数をポート番号として受け口となるソケットを開く
      ServerSocket ss = new ServerSocket(Integer.parseInt(args[0]));
      while (true) {
        System.out.println ("Waiting ...");
        Socket so = ss.accept();  // クライアントからの接続を待つ。接続されるまで処理がブロックされる
        System.out.println ("client request accepted !!");

        // 接続が確立されたらクライアントにメッセージを送信
        PrintStream out = new PrintStream(so.getOutputStream());
        out.println("Connection Established");
        out.println("\tNice to meet you.");

        // 入力待ちのスレッドを作成/起動し、この処理は次のクライアントからの接続に備える。
        tid = new serverIO (so);
        tid.start ();
      }
    } catch (IOException e) {
    }
  }
}
```

クライアント・サーバー型のネットワークアプリケーション

◆◆◆◆◆◆

それでは最後にネットワークアプリケーションの例について見てみよう。Java言語ではクライアント・サーバー型のネットワークアプリケーションを開発することができる。言語としてはネットワークプロトコルには依存しない形で作成できるが、現在のところTCP/IPでの通信のみがサポートされている。

ネットワークアプリケーションを作成するためには「java.net」パッケージが用意されているので、これを使うとUNIXでのsocketライブラリを利用したネットワークアプリケーションと似たようなプログラムを作成できる。

今回はすでに作成した四角を書くアプレットに修正を加えたクライアントとサーバーの間でメッセージの交換をするアプリケーションを作成する。サーバーとクライアント間で特定のポートを予め決めておいてネットワークを接続する。サーバーを起動するとクライアントからの接続を受けるためのソケットが作成される。つぎにクライアントを起動し、サーバーとの接続が確立

されると、サーバーからメッセージが送られてくる。クライアント側でマウスボタンの操作によって四角が書かれるたびに、四角の座標がサーバーに伝えられる。

【サーバープログラム】

それでは、まずサーバー側のプログラムから見てみよう（プログラム❹）。今回の例の中でこのサーバーアプリケーションのみがアプレットクラスのサブクラスではない。デーモンと呼ばれるプログラムがUNIX上のさまざまなネットワークサービスを提供しているが、Java言語で書かれたアプリケーションも単独で動作するデーモンとして記述ができることを示す意味も含めてスタンドアローンとして記述した。もちろんアプレットとして記述することもできる。

通常、ネットワークアプリケーションは入出力処理とアプリケーションのメイン処理を非同期にするので、マルチスレッド処理が必須だ。ネットワークからのデータ読み込みは希望する長さ（バイト数）のデータが常に読み込めるとは限らないからだ。データが到着するまで読み込み処理にだけに専念するわけにいかない。UNIXではselectシステムコールを追加したりread/writeなどの入出力処理をブロックしない（データが読み込めても読み込めなくても呼び出し側にすぐ戻る）ように修正していたが、Java言語ではスレッドがサポートされているので割と楽に作成することができる。受信と送信、さらにはメイン処理を別々のスレッドとして記述することで単純化ができる。

この例ではサーバーはクライアントからの入力だけを監視しているので、送信用のスレッドは作成していない。クライアントからの接続が確立し、受信用スレッドを作成するとサーバーはつぎのクライアントからのリクエストを受け取る処理をする。このようにサーバープログラムを作成すると、複数のクライアントからの処理を同時にできる。

プログラム❺　クライアント側プログラム

```
import java.io.*;
import java.net.*;     // ネットワーク用のパッケージをロード

public class clientThread extends Thread {
      // ネットワークスレッド用の内部変数を定義し、初期値も設定する。
      Socket so = null;
      Thread tid = null;
      DataInputStream input = null;
      PrintStream out = null;

      // 指定されたホストのポートへ接続する。
      public void open(String host, int port) {
            try {
                  // まずは通信用のソケットを作成
                  so = new Socket(host, port);

                  // 入出力用のデータストリームを作成
                  input = new DataInputStream(so.getInputStream());
                  out = new PrintStream (so.getOutputStream ());
            } catch (IOException e) {
              // socketの作成に失敗した場合のエラー処理
            }
      }

      // 終了時の処理。開いているソケットを閉じる
      public void close() {
            try {
                  if (so != null)
                        so.close();
                  so = null;
                  input = null;
                  out = null;
            } catch (IOException e) {
            }
      }

      // スレッドになる処理。
      // サーバからのメッセージを受け取り改行毎に画面に表示。
      public void run() {
            try {
                  String msg = "";
                  while (true) {
                        if (input == null)
                              continue;

                        char c = (char)input.readByte();
                        msg = msg + c;
                        if (c == '\n') {
                              System.out.println(msg);
                              msg = "";
                        }
                  }
            } catch (EOFException e) {
            } catch (IOException e) {
            }
      }

      // サーバとの接続が確立されていれば指定されたメッセージを送信
      public boolean       sendMessage(String msg) {
```

【クライアントプログラム】

クライアント側は2つのファイルから構成されている。以前に作ったマウスで四角を書くプログラムに処理を加えて、ネットワークオブジェクトを取り扱うようにしたものと（プログラム❺）、実際にサーバーと通信を行うためのネットワークオブジェクトを使用したものだ（プログラム❻）。

まずはネットワーク用のクラス定義から見てみよう。このクラスは他のクラスから呼び出されて使われることを想定しているので、特にアプレットとして作成する必要はなかった。そのため、「Thread」クラスから派生させた。ネットワークの接続をするためのopenメソッドと、接続を終了するcloseメソッド、さらにスレッドとなって処理を行うためのrunメソッドが基本となっている。今回はデータを送信だけをする必要があったので、メッセージを送信するためのsendMessageメソッドを定義している

プログラム❻　サーバーと通信するプログラム

```
/*
    イベントのテストで作成したマウスを使って
    四角を書くアプレットにネットワークアプレットを呼びだす。
*/
import java.applet.Applet;
import java.awt.Event;
import java.awt.*;

public class EventSample extends Applet {
    Color  fgColor;
    int    upperX, upperY, downX;
    int    lowerX, lowerY, downY;

    static final    int maxChars = 1024;
    char   inputChar[];
    int    numChars;

    boolean isText = false;

    String serverHost = "";                 // 接続先のホスト
    int    serverPort;   // ポート番号

    clientThread sock = null;               // ネットワーク用のスレッド

    public void init () {
        inputChar = new char[maxChars];
        upperX = upperY = lowerX = lowerY = 0;

        // このアプレットを呼び出したサーバに接続する。
        // 現在はセキュリティ上の理由から接続できるホストに
        // 制限がある。
        //      1. 自分自身
        //      2. アプレットを呼び出したサーバ
        //
        // ただし、appletviewerでは、テストの用にセキュリティを
        // はずすことができる。
        serverHost = getDocumentBase().getHost();
        serverPort = Integer.parseInt(getParameter("port"));

        // ネットワークスレッドを作成
        sock = new clientThread();
    }

    public void start () {
        // アプレットが起動されたらネットワークスレッドを開始させ
        sock.start();
        // ネットワークの接続を行う。
        sock.open(serverHost, serverPort);

        setBackground(Color.yellow);
    }

    public void    stop() {
        // アプレットが停止したらネットワークを閉じて
        sock.close();
        // ネットワークスレッドも停止させる。
        sock.stop();
    }

    public void    paint (Graphics g) {
        g.setColor (fgColor);
        if (isText) {
            g.drawChars (inputChar, 0, numChars, upperX, upperY);
            isText = false;
        } else {
            g.drawRect (upperX, upperY, lowerX - upperX, lowerY - upperY);
        }
    }

    public boolean       keyUp (Event event, int key) {
        if (numChars == maxChars)
            numChars = 0;
        fgColor = Color.red;
        inputChar[numChars++] = (char) key;
        isText = true;
        repaint ();
        if ((char)key == '\n') {
            numChars = 0;
        }
        return true;
    }

    public boolean       mouseDown (Event event, int x, int y) {
        numChars = 0;
        upperX = lowerX = downX = x;
        upperY = lowerY = downY = y;
        return true;
    }

    public boolean       mouseUp (Event event, int x, int y) {
        fgColor = Color.black;
        updatePosition (x, y);
        repaint ();

        // 画面上に表示した四角の座標をサーバに送信する。
        sock.sendMessage("Rectangle(" + upperX + ", " + upperY + ") - (" + lowerX + ", " + lowerY + ")");
        return true;
    }

    public boolean       mouseDrag (Event event, int x, int y) {
        fgColor = Color.green;
        updatePosition (x, y);
        repaint ();
        return true;
    }

    private void  updatePosition (int x, int y) {
        if (x < downX) {
            upperX = x;
            lowerX = downX;
        } else {
            upperX = downX;
            lowerX = x;
        }

        if (y < downY) {
            upperY = y;
            lowerY = downY;
        } else {
            upperY = downY;
            lowerY = y;
        }

    }
}
```

が、本来であればopen、close、run、read、writeなどのメソッドで定義し、再利用が可能になるように定義した方がよいだろう。今回はサンプルということでこのようになっている。

ネットワークの接続を確立するためにSocketオブジェクトを作成している。このオブジェクトの引数としてサーバー名とポー番号を指定しているが、Java言語ではセキュリティ上の制限からアプリケーションから接続できるホストは、アプレットを読み込んだホストか、実行している自分自身のホストに制限されている。

JDKでのアプレットテストプログラムであるappletviewerではセキュリティのレベルを「None（不可）」、「applet Host（アプレットを読み込んだホスト）」「Unrestricted（制限なし）」の3段階から選択することができる。

ネットワークのアプリケーションでは各種のエラーやイベント、割り込みといった例外処理が発生することが予想される。そのため、これらの例外に対しての処理を記述する必要がある。

＊ここで紹介したサンプルプログラムはインターネットマガジンのWWWサーバーからダウンロードできます。

URL http://home.impress.co.jp/magazine/inetmag/magnavi/ip9605/

［インターネットマガジン　バックナンバーアーカイブ］ ご利用上の注意

このPDFファイルは、株式会社インプレスR&D（株式会社インプレスから分割）が1994年～2006年まで発行した月刊誌『インターネットマガジン』の誌面をPDF化し、「インターネットマガジン　バックナンバーアーカイブ」として以下のウェブサイト「All-in-One INTERNET magazine 2.0」で公開しているものです。

http://i.impressRD.jp/bn

このファイルをご利用いただくにあたり、下記の注意事項を必ずお読みください。

- 記載されている内容（技術解説、URL、団体・企業名、商品名、価格、プレゼント募集、アンケートなど）は発行当時のものです。
- 収録されている内容は著作権法上の保護を受けています。著作権はそれぞれの記事の著作者（執筆者、写真の撮影者、イラストの作成者、編集部など）が保持しています。
- 著作者から許諾が得られなかった著作物は収録されていない場合があります。
- このファイルやその内容を改変したり、商用を目的として再利用することはできません。あくまで個人や企業の非商用利用での閲覧、複製、送信に限られます。
- 収録されている内容を何らかの媒体に引用としてご利用する際は、出典として媒体名および月号、該当ページ番号、発行元（株式会社インプレスR&D）、コピーライトなどの情報をご明記ください。
- オリジナルの雑誌の発行時点では、株式会社インプレス R&D（当時は株式会社インプレス）と著作権者は内容が正確なものであるように最大限に努めましたが、すべての情報が完全に正確であることは保証できません。このファイルの内容に起因する直接的および間接的な損害に対して、一切の責任を負いません。お客様個人の責任においてご利用ください。

このファイルに関するお問い合わせ先
株式会社インプレスR&D
All-in-One INTERNET magazine 編集部
im-info@impress.co.jp