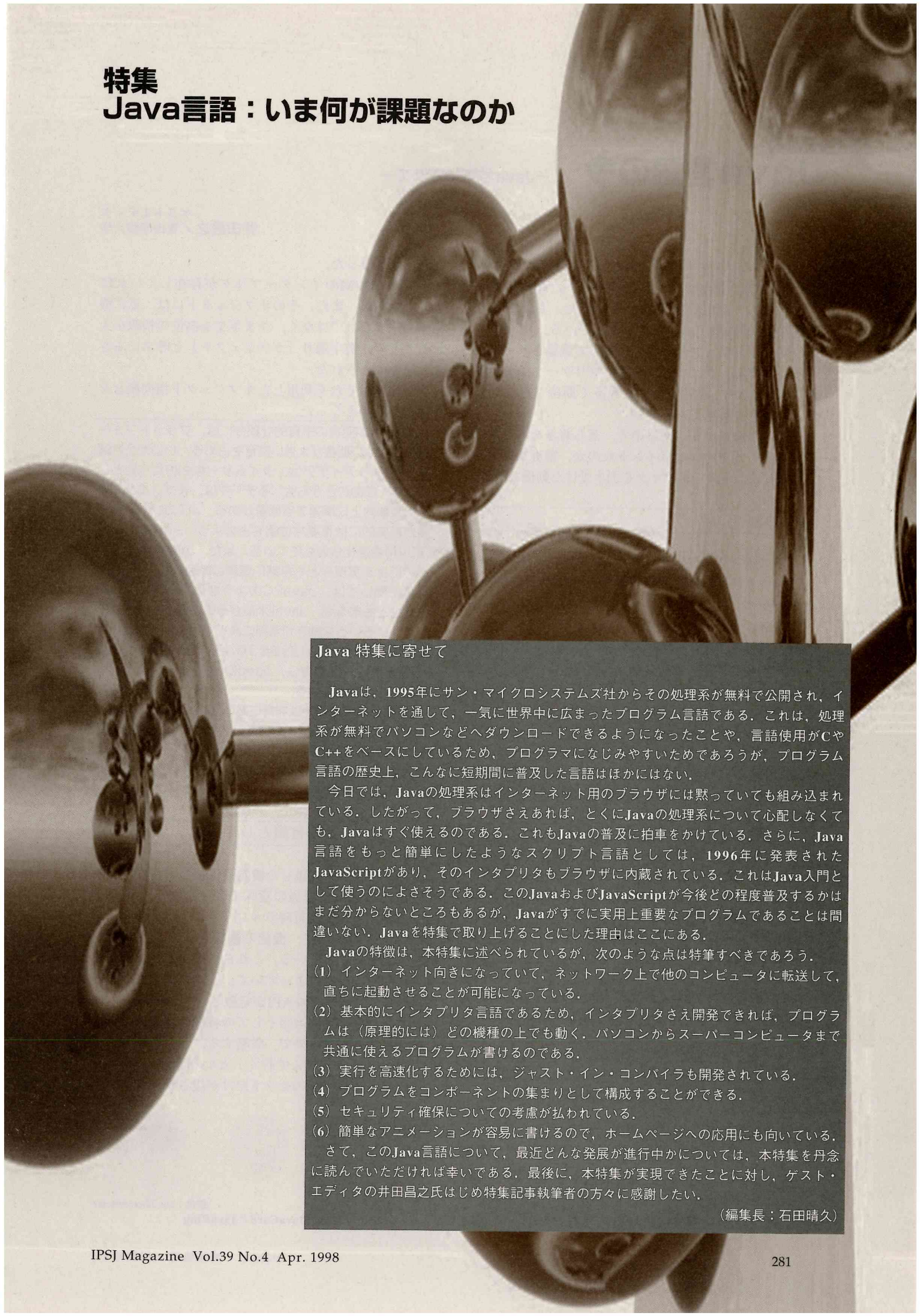
特集
Java言語：いま何が課題なのか



Java 特集に寄せて

Javaは，1995年にサン・マイクロシステムズ社からその処理系が無料で公開され，インターネットを通して，一気に世界中に広まったプログラム言語である．これは，処理系が無料でパソコンなどへダウンロードできるようになったことや，言語使用がCやC++をベースにしているため，プログラマになじみやすいためであろうが，プログラム言語の歴史上，こんなに短期間に普及した言語はほかにはない．

今日では，Javaの処理系はインターネット用のブラウザには黙っていても組み込まれている．したがって，ブラウザさえあれば，とくにJavaの処理系について心配しなくても，Javaはすぐ使えるのである．これもJavaの普及に拍車をかけている．さらに，Java言語をもっと簡単にしたようなスクリプト言語としては，1996年に発表されたJavaScriptがあり，そのインタプリタもブラウザに内蔵されている．これはJava入門として使うのによさそうである．このJavaおよびJavaScriptが今後どの程度普及するかはまだ分からないところもあるが，Javaがすでに実用上重要なプログラムであることは間違いない．Javaを特集で取り上げることにした理由はここにある．

Javaの特徴は，本特集に述べられているが，次のような点は特筆すべきであろう．

(1) インターネット向きになっていて，ネットワーク上で他のコンピュータに転送して，直ちに起動させることが可能になっている．

(2) 基本的にインタプリタ言語であるため，インタプリタさえ開発できれば，プログラムは（原理的には）どの機種の上でも動く．パソコンからスーパーコンピュータまで共通に使えるプログラムが書けるのである．

(3) 実行を高速化するためには，ジャスト・イン・コンパイラも開発されている．

(4) プログラムをコンポーネントの集まりとして構成することができる．

(5) セキュリティ確保についての考慮が払われている．

(6) 簡単なアニメーションが容易に書けるので，ホームページへの応用にも向いている．

さて，このJava言語について，最近どんな発展が進行中かについては，本特集を丹念に読んでいただければ幸いである．最後に，本特集が実現できたことに対し，ゲスト・エディタの井田昌之氏はじめ特集記事執筆者の方々に感謝したい．

（編集長：石田晴久）

特集

Java言語の今 —Java特集に寄せて—

ゲストエディタ
井田昌之／青山学院大学

「Java」という言葉が，何か妖しい魅力のある魔法の玉手箱のようなひびきを持っていた時代があった．今なお，新しい発表が続き，また，さまざまな実社会での喧騒がJavaを取り巻いている．Javaはワールドワイドウェブと密接に関連して議論されることが多い．事実，ウェブベースのアプリケーション環境ということが，当面Javaに最も多く期待されていることのようにも見える．

さまざまな声が聞こえる中で，落ち着きを取り戻して，さぁいったい何がポイントなのか，整理を始めよう．これが，ゲストエディタを引き受けた動機である．

Javaの原点に戻って

多くの人々をとらえたJavaの魅力は，つきつめると「プログラムが飛んで行く」ということにあるのではないか？　ネットワーク上のどこかに存在しているプログラムが，それを要求しているクライアントの所へ飛んで行き，そこで実行されるのである．クライアントはどんな種類のものであってもよい．ネットワークでつながれたコンピュータ間でプログラムをやりとりできる，そしてそれらが通信／協調しながら仕事をする仕組みが可能になる．この枠組みは，そもそものセットトップボックスのシステム用に考えられたその発端においても，爆発的な人気を得たウェブにおけるアプレットとしての利用においても，共通して考えられて来た夢の基盤であった．

多数のウェブブラウザがJavaに対応しており，それを利用したホームページも相当数にのぼる．ウェブを介してアプリケーションを実行する仕組みも多く利用されるようになった．たとえば，ハッブル宇宙望遠鏡へのJavaの採用により，研究者は世界各地にいたままで，ウェブブラウザを通して，望遠鏡からのデータをのぞき，処理をすることができる．

同時に，ネットワークコンピュータへの期待もある．JavaStationである．また，JavaRing，JavaCardあるいはいわゆる情報家電などが開発の途上にある（図-1）．

クライアント側だけではない．Java Web ServerなどJavaによるサーバの構築，さらに，もっと普遍的な分散モデルなどへと研究と開発は進んでいる．

Java仮想機械

Javaプログラムがどこでも実行できるという「魔法」は，Javaのオブジェクトプログラムは，Java仮想機械（JVM）の機械語の形をとる，という事実により可能となった．

少くともJVMのインタープリタが存在していればよいのである．また，そのオブジェクトには，単に機械語というだけではなく，さまざまな高位の情報を入れることで，文字通り「オブジェクト」と呼ぶにふさわしいものとされた．

開発環境もそれを利用したオブジェクト指向概念を利用したものとなった．

Javaへの期待の本質的な部分には，プラットフォーム独立性への期待が大きい割合を占める．いいかえれば，JVMというアイデアは，タイムリーなものだったといえよう．このことから生じるテーマは，まず，コンパイラの性能向上に関連する問題がある．JITコンパイラ技術が現実的には重要な位置を占めよう．HotSpotと呼ぶJVM高速化技術もでている．また，JVMのハードウェアによる実現なども自然に議論の範囲となっている．後者に関しては，JavaPCのような既存PCのJava端末化などもあるが，JavaChipがその中心的な話題であろう．いろいろな動きが活発に続くだろう．一例として，最近のニュースでは1998年3月4日にIBMはPicoJavaIのライセンスを取得し，JavaChipによる民生機器への対応を発表している．

JVMによる互換性は非常に高いが，feature levelでの互換性については まだ改善の余地がある．標準の制定が最終的には問題を解決することとなろうが，当面は，技術進歩の途上である点もあり，雑音がまだ生じることが十分考えられる．

オブジェクト指向言語として

ネットワークを意識した機能強化などがされているが，Javaは完全に普通の意味でのプログラミング言語である．また，実行時ライブラリの概念があり，CやC++などのように，機能の豊富さは，ライブラリ機能の具備によっている．これらの機能は，オブジェクト指向的な概念によっていて，クラスライブラリと呼ばれる．さまざまなAPIが定義されている．

オブジェクト指向言語としての特徴の詳細については誌面に限りがあるので，省略する．

「プログラムが飛んで行く」という状況において，プログラムは最小限のセットだけが渡され，ライブラ

提供：Sun Microsystems

図-1　JavaCardとJavaRing

リ機能はそれを呼び出したクライアント側のものが使われる．飛んで行った先の機能が利用されるのである．ここに分散オブジェクト技術への展開の基礎がある．

機能の豊富さを裏付けるものは，多様なAPIの設定とその実現である．ネットワーク機能，メディア機能といったレベルから，電子コマースのためのAPIといった応用領域のインタフェース仕様まで広く論議され，そのAPIに従うライブラリが複数利用可能になり，JavaのOpenプラットフォームとしての位置を確実なものにしている．

オブジェクト指向としての性質は，まず開発環境，そして，コンポネント指向の開発概念，言語としての特徴という点に加え，実行時でのオブジェクトの処理にまで渡っている．たとえば，Serializationの概念とその具備は，ファイルやシリアルストリームにおけるオブジェクトの表現を単なるデータ表現以上のものにしている．

永続オブジェクトに関してもJavaSpacesという技術が始まっている．GUIに関しても今まであった低レベルのAWTというツールに，Netscape社のIFCを統合したGUIコンポネントとしてJFCが登場している．

コンポネント指向の開発概念は，JavaBeansと呼ぶ技術を中核としており，全体的なソフトウェア生産性の向上はいうまでもなく，既存ソフトウェア財産（レガシーシステム）との連係においても，また，他の技術体系との連係においても，役立つことが期待されている．

JDBCによるデータベース対応，JavaIDLによるCORBA対応その他，多数の関連技術とのリンケージが存在／提案されている．

Javaに何を期待するのか？

Javaの言語仕様は，さまざまな期待を受けて，拡張と改良の要求がある．また，関連するさまざまな技術領域においてJavaの利用が考えられ，Javaの実行環境，開発環境に対しては新しいさまざまな技術が提案され，開発されている．工業化と普及のためには，標準化も重要であり，1997年11月にはいくつかの手順を経て，Sun Microsystems社は，ISO/IEC JTC1でのJava言語標準に関するPAS Submitterになった．参照処理系としてのJDKの配布，Webによる情報の公開，The Java Language Specificationの刊行，今までのオープンな歩みが評価されたものといえる．

こんなことを話し合ったことがある．

「言語の安定にはだいたい5年かかるだろう．今，Javaは3年くらい経った．もう一息．」

別の言い方をすれば，「Java言語はまだ成長する．言語仕様においてもまたその環境，応用に関しても．」

特集の内容

この特集では，以上のような点から，7編の解説論文と1編のエッセイを置くこととした．それらは選ばれた著者が大いに語ってもらうことで，この成長途上にある時期での特集を意味のあるものとすることとした．

「Java言語仕様は成長中である」ということから，その中核メンバの1人であるGuy Steele氏（Sun Laboratories）にその最先端の話題を語ってもらった．「国際化」は重要な関心事であり，風間氏（NTT）に，「オブジェクト指向」については，一般的なオブジェクト指向プログラミングについては別の機会に譲ることとし，平野氏（電総研）に依頼して分散オブジェクトの視点をまとめてもらった．これら3つが言語仕様およびその基本機能に関するものである．

次に，応用分野に関してだが，2つの解説を収めた．まず，Javaの応用の新展開に関して，次にJavaBeansに関してである．前者は松岡氏（東工大）および高木氏（名工大）に，後者は山口氏（日本Sun）に依頼した．

JavaはNC（ネットワークコンピュータ）との関連でも議論されている．この特集では，それについてそのままではなく，関連するJavaOSとJavaChipについてとりあげた．理由はNCおよびそれに盛り込まれる機能はいまだ多分に実験の域を出ないと判断していることによる．JavaChipについては藤田氏（日本Sun）に依頼し，JavaOSについては井田が担当した．

この特集では初学者用の解説は含まれていない．許された誌面に限りがあり，その中では現在の動向に関する正確さの高いそして生き生きとした解説を含めることに重点をおいたからである．けれども，ヒントとなるURL集を囲み記事として含めたし，各文では話題となっているキーワードについては含めるように努力したので，各解説の全ページを追うことで，Javaの全体像にふれることができるはずである．

Javaを学ぶには

Javaの言語仕様は，Addison Wesley社のThe Java Seriesにある．特に，The Java Language Specification（0-201-63451-1），The Java Virtual Machine（0-201-63452-X），The Java API Vol1（0-201-63453-8），Vol.2（0-201-63459-7）には設計者らによる基本的な定義が記されている．邦訳も順次出されているようである．入門書は多数あり，筆者も「Newはやわかり Java」（共立）を著しているが，自分で手にとって，最もなじみやすいものを選ぶとよい．ウェブサイトとしては，http://www.javasoft.com/あるいはhttp://java.sun.com/からもっとも中心的な情報が得られる．また，Java関連のプロダクト／出版物については，「ポートフォリオJava Computing編」（日本サン・マイクロシステムズ 1998.01）という冊子があることを知った．多くの学術ジャーナルおよび商業誌が，今後もJavaに関連する論文／解説／紹介記事を掲載することであろう．

この特集を出発点として，Javaに関する研究あるいは開発がいっそう進展することを期待する．

（平成10年3月11日受付）

特集

1. Java言語仕様設計上の考慮点

Javaは，高セキュリティ，高能力のフルスケールのプログラミング言語として，ネットワークプログラムに使われてきた．その進展と共に，たくさんの，面白い，普通とは違ったアプリケーションが出現してきた．これらのアプリケーションは，言語それ自身とAPIの2種類のサポートに支えられている．今後も広がるさまざまなアプリケーション領域をサポートするように，Java言語自身に要求されるものは，何なのだろうか？

Cや，C++と比較した時，Javaの良い点として，よく見られるプログラミングエラーをかなり早い段階で，たとえばコンパイル時に，キャッチできるということがある．また，陽なチェックを埋め込むことで，実行時にエラーをキャッチし，そして，プログラムの動作を中断させることなく障害をリポートすることができることがある．

Javaは，コンパイル時に厳密な型分析をして，プログラミングエラーをキャッチできる．ある型を持つとコンパイル時に宣言されている変数に対して，コンパイラはその変数の値がその型に本当に属するものであることを保証する．Cや，C++でのように，プログラマは，キャスト演算子を挿入することで，ある型から別の型へ，式を変換させる指示をさせることができるが，もしJavaコンパイラが，そのキャストが常に正しいものだと検証できなければ，実行時に陽にテストするコードを生成して，型の正しさが保持されるようにする．

また，C プログラムは割り付けられた記憶領域をフリーにしたのちに，（誤って）そのメモリの内容をそれがまだ持っているポインタを介して変更すると，トラブルに会う．Javaは，自動ガベジコレクションを用いているので，これが起きない．記憶域を，陽にフリーにする方法はない．また，記憶域の内容はすでにそれを参照することがなくなった後でのみ回収する．

ここまでのところ，Java言語の設計のこうした側面は大変うまくいっている．けれども，この言語は大きくいって，3つの領域での拡張の要求を受けている．サブセット，型システム，数値計算である．なぜ現在のJava言語ではこうした目的には不十分だと感じている人がいるのか，なぜ，変更プロポーザルがあるのか，これらのプロポーザルはJavaの全体の設計哲学にどうやってうまくフィットさせるのかなどを見ていく．

Guy L. Steele Jr.
Sun Microsystems Laboratories

訳・編集：井田昌之
青山学院大学

言語設計への3つの拡張要求

1）サブセッティング

■アリゾナ効果？

ある言語仕様を，それ1つだけであらゆるアプリケーションに対して，あらゆるプログラミング上の問題を解決するように設計するのは，困難なことである．Javaは，すべての目的に対してよい言語になるように，と試みてはいない．

アプリケーション全般に対応しようとする言語は，普通私がアリゾナ効果と呼んでいるもので悩むことになる．これは，アメリカ合衆国の初期の頃の話に由来する（この話は歴史的事実，というよりも比喩のために用いている）．その時代，アレルギーとか喘息あるいはその他の呼吸器関係の疾患を持っている人たちには，アリゾナへの引っ越しが有効な治療手段だと考えられていた．砂漠性気候で，また，花粉をまくような植物が大変少なかったからだ．そして，合衆国の他の地域，特に東海岸からたくさんの人たちが，アリゾナに移住した．しばらくそこに住むと，その人たちはその砂漠性気候は無味乾燥なものに思え，今まで住んでいた所を恋しく思うようになった．そこで，花や他の植物を移植して，新しい家もまた，以前の家の懐かしい雰囲気を持つようにしようとした．まもなくその人たちは，昔持っていた問題をまた持つようになる．そして，以前と同じように不幸になるというのである．

■Switch文はよくない？

Javaの設計者達は，古いプログラミング言語の難しい機能を組み込まないよう

にしてきた．もっともそれらは，必ずしも成功してはいない．私自身の好みでいうと，switch文は，パスカルでのcase文に置き換えられるべき難しい機能だと思っている．けれども，多分全体としてのトレードオフがあり，このことは非常に取るに足らない部分だったのだろう．おそらく，この基本的な文／式を，C言語から大変違ったものにすると，Javaをプログラマが最初に受け入れるのを難しくしたからだろう．Javaは，なるべく小さく，そして，すべての処理系製作者がこの言語に完全に対応させるのを容易にさせるように考えられている．これは，大変重要なことである．つまり，移植性とプラットフォーム独立という2つが，Javaの大変重要なゴールなのである．そしてすべての処理系製作者が言語全体を完全にサポートする時にだけ，そのゴールは達成できる．

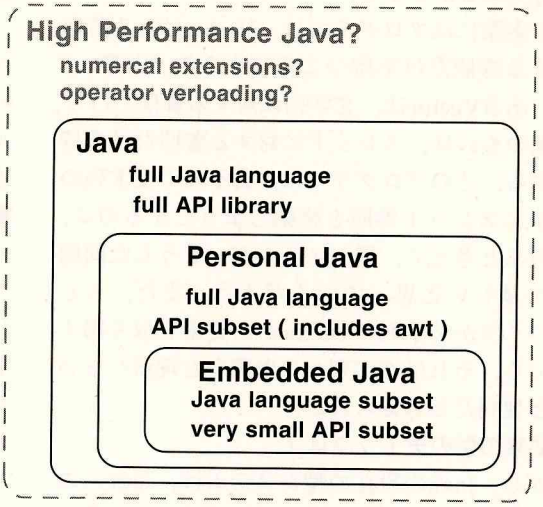


図-1　Java言語仕様の階層

■Javaは大きすぎる？

しかし，小さいはずのJavaでさえもいくつかのアプリケーションに対しては，大きすぎる言語である．たとえば，スマートカードのメモリは小さく，ディスプレイスクリーンも持っていない．この2つの理由で，JavaのAWTウィンドウソフトウェアを，スマートカードが完全にサポートすることを求めるのは，よいことではない．

これは，決して新しい話ではない．私が，X3J11（C言語）の標準化委員会のメンバであった1983年から1984年に，そこでは「I/O関数は，その標準の必須の部分にあるべきかどうか」，という議論をしきりにしていた．それは大変よく使われる機能であり，ほとんどすべてのCプログラマが使っていることが認められるが，同時にI/Oサポートはたくさんのメモリを使う大変複雑な部分であるので，別の代表は常に次のように聞くのである．「エレベータコントローラの場合はどうだい？」

別の言い方でいうと，C言語の重要なアプリケーション領域は組込みシステムであり，そうしたシステムはオフィシャルな標準言語で書くべきだが，その組込み環境では意味を持たない機能をサポートするように要求されるべきではないというのである．

■サブセッティングと仕様の階層

サブセッティングは，大変注意深く扱わなければならない．言語の定義は，移植性に問題を起こすフラグメント化をさけるようにするべきだからである．現在，Java Softでは，Embedded Java（http://java.sun.com/products/embeddedjava/）や，Personal Java（http://java.sun.com/products/personaljava/）を定義している．それらは，組込み環境および，個人用の装置環境に対して，Java言語の機能のサブセットを提供するものである．個人用の装置環境とは，ネットワークベースのモバイル消費者用電子機器などである．これらのものは，すでにJava言語全体を利用しているブラウザや，コマーシャルアプリケーションや，システムプログラムなどに対して混乱を与えない，別のアプリケーション領域としてよく定義されているようである（図-1）．

にもかかわらず，注意をする必要がある．それは，2つあるいは3つの別のセットというのは，管理できるが，たとえば，10とか15のセットというのは多すぎる．そのようにプログラミング言語を定義するというのは誤りだと私は信じている．

2）型システム

現在のJavaの型システムはほとんどの部分で成功したといえる．十分に強力であり，プログラムの振舞いに便利な制約を与え，また，多くのプログラムの操作上の誤りを正しくさせ，また，多くのよく見られるプログラミングエラーを実行時にではなく，コンパイル時に検出し，報告することができるようにしている．これによって，プログラマはアプリケーションユーザに見せるのではなく，自分でエラーメッセージを見ることができる．この型システムは，アプリケーションプログラマが毎日効率よく使うように，十分簡単にできている．

■Vectorはあらゆるオブジェクトを要素にできる

プログラミングの際に，Javaの型システムがうまくいっていない部分にしばしば出会うことがある．集合データ構造がその例である．一番簡単な例は，Vectorクラスである．Vectorは，配列を拡張したようなものである．Vectorは，あらゆるJavaオブジェクトへの参

照を中に持つことができる．

けれども実際にはプログラマは，Vectorはある特定の型に対する参照だけを持つようにすることが多い．たとえば，あるVectorは，文字列に対する参照だけを，あるいは別のものは，スレッドに対する参照だけを持つようにする．そのプログラマは，おそらく文字列のVectorの中にスレッド参照を格納しようとするのは，エラーであると考えて，型システムが，そうした問題を解決して欲しいと思っているだろう．また，もしJavaコンパイラが文字列のVectorから要素を取り出す時はいつでも，それは文字列への参照だと理解していてくれると便利だと考えられる．

■Vector要素の型のチェックは？

残念ながら，Javaの現在の型システムは，これらのどちらの仕事もしてはくれない．その結果この2番目の問題の方を解決するには，キャストを用いてプログラムするように強いられる．そして，最初の問題を避けるには，キャストを用いるかあるいは，自分で十分大きな注意を払うことになる．

文字列のVectorの要素を取り出す時には，

```
String s=
myVectorOfStrings.elementAt (3)；
```

の代わりに，プログラマは次のように書かなければならない．

```
String s= (String)
myVectorOfStrings.elementAt (3)；
```

ある要素を文字列のVectorに格納しようとする時，プログラマは，次のようなキャストを書くか，

```
myVectorOfStrings.setElementAt
((String) aThing, 3) ；
```

あるいは，大変注意深くなって，aThingの値が本当に文字列への参照であるかを，別の手段によって確認する必要がある．

キャストを用いる場合の問題は，実行時のオーバーヘッドではない．それは，心配の一番小さな部分である．問題は，コードの中の表現力，そして，コンパイル時に単純なエラーを見つけることに失敗する可能性，ということである．

■型システムの拡張提案

その結果，Javaに対して型システムを拡張する方向でのプロポーザルがいくつか存在している（たとえば，1997POPL，1997PLDI，1997OOPSLAなどのACMの会議の報告集を見よ）．これらは，大別して2つの種類に分けられる．1つは仮想型もう1つはパラメタ型である．パラメタ型のポイントを**表-1**に示す．

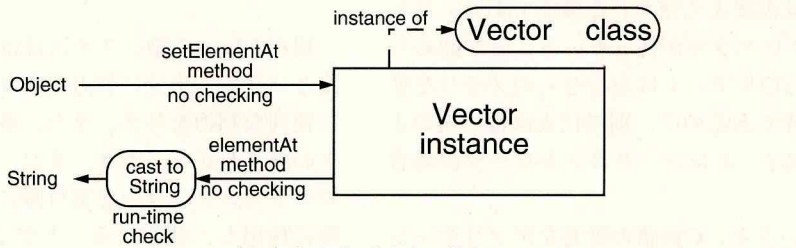
仮想型は大変一般的なものであって，熟練者の手にかかれば，大変柔軟性の高いものである．Betaプログラミング言語で，それらをかなり使ってきた経験がある．けれども，それらは一般的な利用のためには，ちょっと難解である．

パラメタ型は，おそらくもう少しなじみやすいものだろう．たとえば，C++へのテンプレート，あるいは，Adaのgenerics facilityは，この範疇に入る．

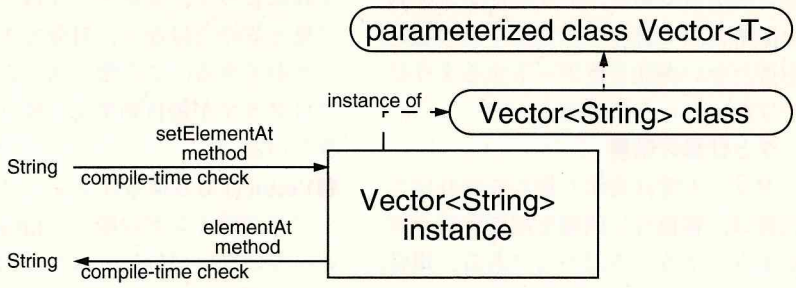
ここまでの議論に関しては**図-2**に示した対比を参照してほしい．パラメタ型の導入により，コンパイル時のチェックが可能となり，同時にcastも不要になる．

表-1 Parameterized Types

- Better compile-time type checking
- Support generic "collection" libraries
- Reduce use of casts
- Similar to C++ templates
- Also support generic numerical types (complex, interval, vector and matrix)



(a) 何もサポートがない場合



(b) パラメタ型を導入する一例

図-2 パラメタ型の導入

おすすめURL集　http://www.javasoft.com/などとは一味違ったさまざまな内容を持っているURLのいくつかを拾い出してみた．その内容についての保証はしないが，少なくともどのようなことが今議論されているかは把握することができる．

(1) Java FAQ What's New（日本語）　http://www.webcity.co.jp/info/andoh/java/javanew.html　Javaとその周辺の話題についての最新情報へのポインタがほぼ毎日掲載される．新製品，フリーソフトウェアの情報からマスメディアのJava関連情報まで．
(2) Java House Mailing List Homepage（日本語）　http://java-house.center.nitech.ac.jp/ml/　Javaに関する話題を対象としたメイリングリストJava Houseのアーカイブ．「トピックス」のページには，Javaプログラミングについてのよくある質問と回答が生の議論へのポインタの形で数百件整理されている．全文検索機能も付いている．
(3) お薦めJava本（日本語）http://www.webcity.co.jp/info/andoh/java/javabooklist.html　日本人読者によって投票されたJava関連書籍一覧．投票によるお薦め度付き．
(4) JavaWorld Book List　http://www.javaworld.com/javaworld/books/jw-books-index.html　分野別に整理されたJava関連洋書の一覧．
(6) Java Programing Information（日本語）　http://www.ingrid.org/java/　Javaプログラミングに関する総合リンク集．スレッド，国際化対応プログラミングや，Servlet，分散オブジェクトなどの話題についてまとめられている．
(5) JavaWorld - Java Jumps　http://www.javaworld.com/javaworld/common/jw-jumps.html　厳選されたJava関連Webページへのリンク集．
(7) じゃばじゃば（日本語）　http://www.asahi-net.or.jp/~DP8T-ASM/java/　Javaの基本的なプログラミングのうちつまづきやすいところが重点的に解説されている．オブジェクト指向設計の基礎，コーディングレベルでの最適化，デザインレベルでの最適化などの解説もある．
(8) マルチメディア・プログラミング（日本語）　http://www.wakhok.ac.jp/~tatsuo/kougi97/　稚内北星学園短期大学の植田教授による講義のページ．

高木浩光（名工大）編

■うまく仕様を拡張するには

1997年の12月の時点では，Javaにパラメタ型を採用するかどうか，あるいは，もっと全体的にどんな設計を考慮するか，ということにはなおも，合意がない．設計上の手ごわい考慮点をクリアする必要があるのもやっかいである．次のようなものである．

①それは単純である必要がある．たとえば，文字列へのベクタを参照している変数を簡単に宣言できなければならない．理想的にはその宣言は，単語vector，単語stringそして区切り記号1，2文字くらいですませるべきである．

②それは次の問題を解決する必要がある．たとえば，以下のような宣言が与えられたとしよう．

Vector<String> v = new Vector<String>（20）；

その時にコンパイラは次の型

v.elementAt（3）

が，stringであると判断できるようにするべきである．同様に，そのコンパイラはv.setElementAt（aThing,3）において，aThingの型はString型にアサインできると，判断できるようになっていてほしい．

③それはJava言語の残りの部分にきれいにフィットする必要がある．たとえば，現在のJava言語ではStackクラスはVectorのサブクラスである．Stack<String>をVector<String>のクラスだと期待できると便利だろう．

④その処理系は，メモリ空間や処理時間をやたらに多く必要とするべきでない．

⑤それはコード互換のアップグレードパスがあるべきだ．Vectorクラスを使っている既存のコードがそのまま使えるのが理想だ．

⑥それはJVM互換のアップグレードパスがあるべきだ．パラメタ型を用いているコードが，現在存在するブラウザ上で，実行できるような処理系戦略が存在するはずである（これは，改良されたJVMの処理系はさらによりよいエラーチェックをしたり，よりよい性能を与えるという可能性を禁止していない）．

表-2　Numerical Computing

· Performance	Compilers
· Functionality	More IEEE support?
· Extensibility	Parameterized types
· Notation	Operator overloading

あるただ1つの設計がこれらすべての制約に対して満足を与えるかどうかは，まだはっきりしていない．もし1つ見つかったとしても次の問題が残る．それを採用すべきかどうかである．私たちは「アリゾナ効果」が我々をおびやかさないようにしなければならない．

3）数値計算

■さまざまな希望

Javaを拡張して，浮動小数点演算を多用する科学アプリケーションにも対応させる提案がされてきている．もともとJavaの設計は速度ではなく，互換性と単純性を強調していた．その結果，正しいJavaの処理系は，すべてのハードウェア上で定められた浮動小数点演算結果を与えることができるが，ほとんどすべてのハードウェアには，ある種のちょっと奇妙な部分がある．それが，処理系作成者があらゆる場合に効率よく正しい結果を出すのを困難にしている．

この問題とは別に，Javaでの表現性を高くしようという提案もある．それによって，数値演算コードを書きやすくしたり，保守しやすくするためである．

どんな拡張が必要かどうかはまだよく分からないが，High Performance Fortran（Fortran 95の拡張）の設計がモデルとして役立つかもしれない．

■4つの問題点

数値計算に対して，現在のJavaの設計には次の4つの問題点がある（**表-2**）．

1. 性能．移植性の点から，Javaはすべての浮動小数点演算に対して，ある特定の再現可能なマシン独立な振舞いをするように求めている．この動作はIEEE754浮動小数点標準に合致しなければならない．さらに実際にはもっと具体的で制限のあるものにしている．というのは，IEEE754はあるいくつかの点では意図的に明確な定義をしていないからだ（たとえば不当な演算に対して生成されるNaNの値，式の中で中間の値が計算される時の正確な精度の規定など）．IEEE754だけでは，多数のハードウェアアーキテクチャの間で99%の互換性があるが，100%の互換性ではない．残りの1%がなおも大きな問題を作り出す．そこで，Javaの現在の設計はある種の妥協である．それは単純で実現性があり，理解もしやすいものだが，Javaの仕様のいくつかの部分は，現在のポピュラーなハードウェアアーキテクチャのすべてでは直接サポートされていないので，したがって，トリッキーなソフトウェアエミュレーションが必要となり，その結果，性能を低下させる．

2. 機能．IEEE754では定義されているが，Javaでは現在サポートされていない機能がある．フラグ情報へのアクセスがその例である．オーバーフロー，アンダーフロー，ゼロ除算などである．

3. 拡張性．数値プログラムでのデータ操作は単純な浮動小数点データだけではない．複素数やベクタ，行列，複素数の行列，有理数，インターバル，などさまざまな数値オブジェクトがある．

4. 表記法．普通のオブジェクト指向でのメソッド記法で十分用を足りているし，多くの場合快適なのだが，同時に数値演算では ＋，－，＊，／，不等号など間置記法が一般的である．Javaは単純な整数や浮動小数点演算に対して間置記法を組み込んでる．そこで疑問が生じる．それを他の数値および数学的なオブジェクトに拡張すべきかどうか．これらの点をどのようにして考えるべきかについて，順に見ていく．

・*Javaは将来への道筋を示す．*
・*地球上にはすでに何百万ものコンピュータがある．*
・*Leave Java alone !*

■性能とは？

まず最初に性能について．インタプリタの使用は性能上のボトルネックではないと暗黙のうちに仮定してきた．現在存在する多くのJava処理系はバイトコードインタプリタを用いているが，そうでないものもある．JIT（Just In Timeつまりロード時）のコンパイル技法を用いていたり，伝統的なスタティックなコンパイラもある．1998年の終わりまでにこれらのコンパイラは確かに性能において，最もよいC++コンパイラなどとたちうちできるようなよいコードを生成するだろう．問題というのはJavaの定義と現在のハードウェアアーキテクチャの間のミスマッチをどう扱うかということだ．考え方は次のようなものがあり，議論をよんでいる．

1. ポジションA.

『Javaはいつも将来のための道筋である．』

移植性が最も重要である．そして正確性がまずなければならない．したがって，我々は，Javaを定義されたその通りに実現するべきだ．そして将来のハードウェアの設計はJavaを正しく，そして効率よくサポートするように変化することを含むべきである．

2. ポジションB.

『地球上にはすでに何百万ものコンピュータが使われている．』

今日もまた，毎日も，多数のコンピュータが使われている．それらはJavaの現在の定義をサポートするのは困難である．仕様をちょっとゆるやかにすることは，ほとんどのプログラムにとってはおそらく害がないだろう．そしてそれらをより早く実行させることで，インストールされたハードウェアをもっと効率よく使えるだろう．そのハードウェアはすでに多額の金額の投資がなされているものである．我々は何年も待つべきではない．ハードウェアのベースが置き換えられるまで待つべきではない．そして今，よい性能を持つべきだ．

3. ポジションC.

『Javaにそこまで要求するのか？　大きなお世話だ．ほっておいて．』

すべての処理系が浮動小数点数の複雑な扱いをサポートする必要はない．高性能を必要とするならFortranを使いなさい．

■IEEE754は絶対か

次に，機能．Javaが現在省いているIEEE754の機能なしで書くのは難しいプログラムが存在する．同時に一方で，IEEE754は低レベルの単純な逐次型プロセッサ，アセンブリ言語のレベルを提示していて，隠された状態（副作用）に依存したり，制御構造（そして高水準プログラミング言語の残りの部分との関係）が指定されていないトラップ機構に依存したりしている．副作用依存性は，並列ハードウェアアーキテクチャや，パイプラインアーキテクチャなどの現在の高性能のためのゴールに対して障害となる．オーバーフローなどを検出できるのは望ましいことである一方，IEEE754プログラミングモデルを絶対視するべきだとはいえない．

■再びアリゾナ効果

そこで次に拡張性．アリゾナ！新しい組込みデータ

型をJavaに何十も加えるというのは，確かに望ましいことではない．それは処理系をもっと機能あふれるものにするだろうが，同時に言語のサブセットとして現在の形を保とうという強い要求が出てくるだろう．それは入れようとしている数値計算機能を落とすということである．

複合型のデータ型，たとえばベクタ，行列，複素数，ユーザ定義のインターバルなどをJavaのクラスとして許すようにするとか，また，よいJavaコンパイラがそうしたデータ構造上のオープンコードを可能にしたり，おそらくは多くの場合にヒープの割り付けを避けるようなそうした機能やキーワードを与えるというようなプロポーザルはたくさん研究されている．コア言語をどちらかといえば単純なものに保つ一方で，現在のJavaの処理系と互換であり，サードパーティの数値計算サポートライブラリ市場を開くものとなるようなアプローチも考えられる．パラメタ型はそうした数値ライブラリを開発するのにふさわしいかもしれない．

■オペレータオーバーローディングは却下された

最後に表記法．これは技術的にはすべての中で最も単純なものである．それはすべてコンパイル時に扱うことができる．したがって，特定の実行時のサポートを必要としない．また，技術的には些細なことである．Javaの設計者達はもともとオペレータオーバーローディングを考えていた．そしてそれを却下した．彼らのよく考えた結果の意見というのは次のようなものである．

オペレータオーバーローディングはまったく数値計算コードをクリアなものにするが，同時に非数値演算コードをより読みにくくする．そしてそれは多くのC++プログラマによってひどく悪い方法で乱用されてきた．Javaに対して意図された応用領域では，あまり，数値計算コードを考えてはいなかった．そしてオペレータオーバーローディングをやるにはためらいがあったようだ．

技術的な微妙な点として取り上げたいのは以下のようなことである．ほとんどすべての人は，

```
a+b
```

は，次のことを意味すると合意できるだろう．

```
a.plus(b)
```

つまり，オーバーロードされたオペレータは単にメソッド呼び出しの簡略化された形式なのである．これはaの型が参照型である場合に，意味を持つ．また，aもbも参照型でないが共にプリミティブ型であるとき，Javaに現在組み込まれたルールがそうしたケースをカバーする．けれどももし，aがプリミティブ型であって，bが参照型であった場合はどうだろうか？

提案1：そうしたケースは禁止する．これは大変単純で，説明が容易である．不幸なことに，それはよく使われる数学的な表記方法を禁止することになる．和は定数を最後に書くことになる．(a+3)．一方で，積は定数を最初に書くことになる．(3＊v)．古い習慣はなかなか死なないのだ．

提案2：aが参照である場合にa＋bはa.plus(b)を意味するようにする．一方，bが参照である場合にはaはbの前に評価される点を除いてb.reversePlus(a)を意味するようにする．提案1より多少よりよいものになる．けれども解決できない別の問題がある．数学的なベクタクラスが定義されていて，その時に行列クラスを定義したい人がいたとする．これは行列をベクタでかけるメソッドを含むことになってくる．m＊vはm.multiply(v)のような何かを意味することになる．けれども，ベクタを行列でかけるというのはどうだろう．この乗算は可換ではない．したがって，我々は常にm＊vと書くことを強要できず，v＊mは何か別のことを意味するようになる．けれども，ベクタクラスは行列クラスを書くプログラマのコントロールの元にはない．行列クラスがv＊mを扱う方法が存在する必要がある．けれども，ベクタクラスがv＊mを本当に処理しようとしたらどうなるだろう？（それは，ベクタを任意のオブジェクトで乗算するようにできるということになっていくのだろうか）．

提案3：これが現在の私の好みである．もしaが参照であり，bがそうでない場合にはa＋bはa.plus(b)を意味する．bが参照であり，aがそうでなければ，a＋bはb.reversePlus(a)を意味する．もし，a，b共に参照であればa＋bは，a.plus(b)とb.reversePlus(a)の両方を意味する．つまり，これら2つのメソッドの1つだけが最も特定的に適用可能なメソッドを持つべきであり，もし両方ともが持っていれば，a＋bはあいまいとなり，したがってコンパイル時のエラーとなる．

さらに，これらの提案が議論していないいくつかの点がある．たとえば，ある種の機械では，拡張（80ビット）あるいは4倍（128ビット）精度の浮動小数点型というものをサポートしている．けれども，そうした型をJavaに加え，それを必須の型にすると，それをサポートしていないマシン上では貧弱な性能となり，処理系の頭痛の種となるだろう．そうした型をオプションとして加えることは移植性を破壊させる．どうしたらいいのだろう？

処理系戦略

Java言語が変更されるか，あるいは拡張されるか，にかかわらず，なおも処理系の技術や戦略の改良について，たくさんの部分が残ってる．

言語のサブセッティングに関する論点とは別に，性能やコスト特性が異なる処理系がさまざまに存在する必要がある．たとえば，ソフトウェアあるいはハードウェアに対するリアルタイム応答性など特殊な要求も

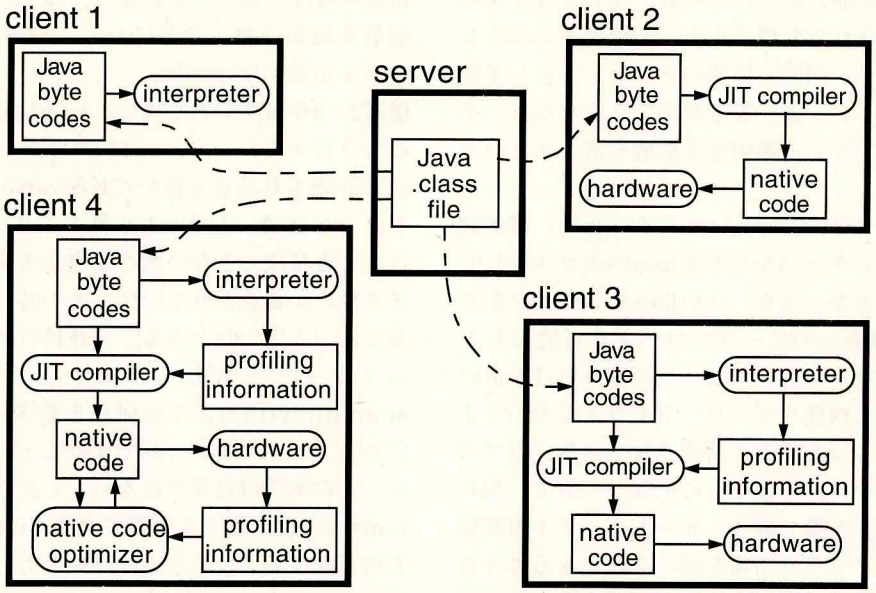


図-3　さまざまな実行形態

いくつかある．単純な例としては，マルチメディアの配達というのは，予測できない制約されない中断時間には耐えられないということがある．それらは単純な記憶管理（ガベジコレクション）の多くのアルゴリズムなども関連する．一方で，スマートカードなどの小さな組込みのシステムなどは，長い休止を避けるような洗練された記憶管理技法をサポートするほど，十分なメモリを持っていないかもしれない．

on the flyコンパイル

もう1つの重要な研究領域は，「on the fly」コンパイルである．この言葉が意味するのは，単純なJITコンパイルではなく，実行時のコンパイルということである．JITコンパイルはバイトコードを，そのクラスがロードされる時に機械語に変換することである．

実行時のコンパイルには，Javaの実行性能を最大にするのに重要な2つの側面がある．まず第1は，Javaの処理系はコンパイルリソースを割り付ける最適戦略を決定するために，実行時プロフィルのフィードバックを用いることができるということである．これは頻繁に実行されないコードに対する最適化の努力を多大に払わないで済むようにさせる．第2の点は，Javaの処理系は実行時の情報の収集と，キャッシングの戦略を用いて，そのコードの実体の特定の動作の振舞いについて，コンパイルされたコードを最適化できるということである．たとえば，原理的には複数のメソッド定義のどれかを呼び出すような仮想的なメソッド呼び出しを，そのプログラム実行の間で常にある1つの定義だけを呼び出すようにできるかもしれない．コンパイルされたコードはまずそうしたケースに際して，インラインテストを用いることで，賭けをするようにカスタマイズできる．つまり，そのメソッドの差し立てを平均よりももっと速くすることができる．その結果，on the flyコンパイルを用いるJavaコードは，最もよいスタティック・スタンドアローン・コンパイルで生成されたコードよりも速く実行できるかもしれない．

これらを図-3にまとめた．client1から4へ順に，高度なものとなる．

パラレリズム

3番目の領域は，パラレリズムである．Javaはプログラムの中の並列性を移植性のある仕方で陽に提供する，という点で，かなり普通のプログラミング言語とは違ったものになっている．Javaバーチャルマシンの構造の中に，並列性に関するチャンスがあることを指摘しておきたい．

今仮に，JVMは2つの（あるいはそれ以上の）プロセッサを使えるとしよう．それらはユーザスレッドを実行するためにすべて割り付けられるべきだろうか？

おそらく1つのプロセッサは専用にするべきだろう．あるいは少なくともコンパイルの作業に使えるようにするべきだ．また，記憶域管理についてはどうだろうか？ユーザプロセスと同時にガベジコレクタが実行するようなコンカレントガベジコレクションを本当に使うのはなおも難しいことかもしれない．なぜならば，極端に注意深く設計されたハードウェアのサポートなしでは，同期のためのオーバーヘッドは無視できないものになっているからだ．しかし，パラレルガベジコレクタを設計するのは，それほど困難さはないかもしれない．その場合，ユーザプロセスはガベジコレクションの間，停止しなければならない．けれども，ガベジコレクションアルゴリズム自身は，多くのプロセッサを同時に利用できるのである．

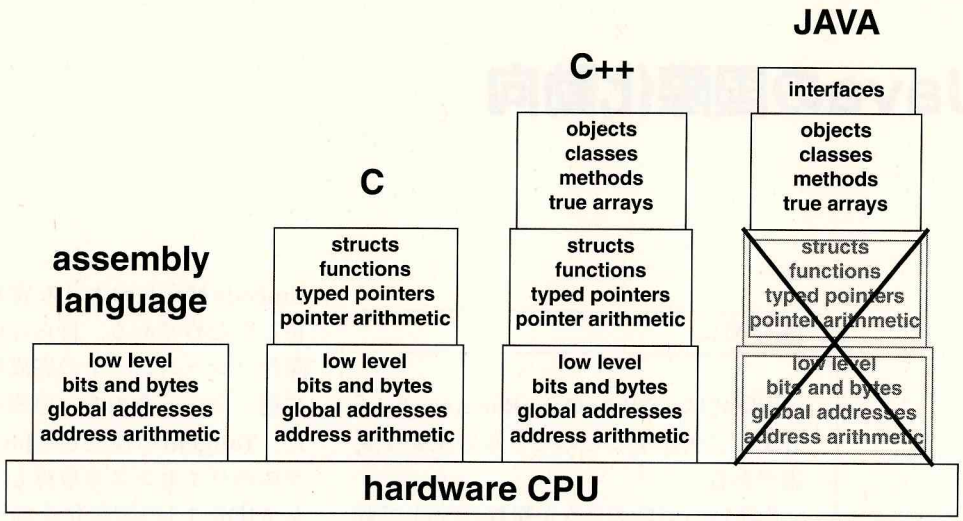


図-4　システムプログラミングの進化

結論

Javaは成功したものだと信じている．そして複雑さをうまく管理できるようにしていると思うので，より広く使われ続けるだろう．古いプログラミング・パラダイムを捨て，注意深く選んだいくつかの新しいパラダイムをサポートすることで，アプリケーションの作成と展開の過程を簡単化した．

システムプログラミング言語の進化を調べることも有益である．以下のようになる（**図-4**）．

①アセンブリ言語は大変に低レベルだった．その機械が行うことすべてをすることができた．けれどもその結果のコードは抽象度が足りず，また移植性がなかった．その基本的なデータモデルはビットを語に編成したものだった．プログラマはすべてのアドレス計算をコードしなければならなかった．

②C言語はシステムプログラムにおいて大きくステップを進めるものだった．オペレーティングシステム全体を含むたくさんのコードは，おおよそ移植できるものとなった．それは移植性があるというより，容易に移植できると呼ぼう．

C言語はいくつかの便利なデータ抽象化を提供していた．アセンブリ言語のビットの処理はバイトの配列に編成された．それは次に，レコード構造，あるいはバイトではなくデータ項目の要素からなる配列へ自動的にうまく分けられていった．そしてもし必要であれば，常に低レベルのビットを参照することができた．キャストを1つか2つすることでコンパイラはすべてのものにアクセスすることを許した．もちろん，ビッグインディアンとリトルインディアンの機械の両者でうまく動くようなコードを書くのは困難なことだった．またこの言語は必要な秩序を保つ手助けはほとんど与えていなかった．

③C++はレコード構造上に，オブジェクト指向プログラミングをサポートした．これは大変高レベルの抽象度の組み立てを許し，また真の移植性にそれ自身を導くような秩序のあるプログラミングスタイルを許した．けれども，キャストをすると，その下にあるレコード構造を，バイトの配列を，あるいは生のビットなどを，なおも参照することができた．

④Javaは完全に高レベルのオブジェクト指向設計まで上り詰めて，その下のはしごを全部蹴飛ばして捨てたようなものだ．

Javaでは，すべてのものはオブジェクトであって，それと2，3の秩序のあるプリミティブ型だけだ．オブジェクトの生のビットを見ることは許されていない．コードは真に移植性がある．その型システムによって高レベルの抽象化が強いられているので，キャストはあるが限られていて，その抽象度をはみ出すことはできない．コードは安全であり保守も容易である．JVMが，常にビットの組は「参照のように偶然見えるような整数」ではなく，参照であることを知らせるので，自動的に記憶域管理をすることができる．クラスの振舞いは，APIによって，記述される．JavaはAPIの処理系に関する健全な競争型のマーケットを生成してきた．

抽象データ型，オブジェクト，定義済みのインタフェース，自動記憶域管理，などは古いアイディアだが，Javaと共にそれらはより広範に受け入れられ，そして，実社会で毎日，何百，何十万の人々に用いられている．

（平成10年2月16日受付）

特集

2. Javaの国際化動向

風間一洋
NTT光ネットワーク
システム研究所

はじめに

国際化は，“Write Once, Run Anywhere”の実現に欠かすことのできない重要な要素である．

あるJavaプログラムが複数のOS上で動作しても，日本語の入力やアラビア語の処理ができないように使用できる国や言語が制限されるとしたら，とても“Write Once, Run Anywhere”とはいえない．また，国際化が基本クラスとは独立して開発され，国際化に対して通常のプログラミングとは別の作業が要求されるとしたら，国際化されていないプログラムが大量に生まれることになる．

これらの問題を避けるためには，国際化機能を基本クラスに不可分に組み込み，国際化をデフォルトにすると共に，国際化された高レベルAPIを提供して国際化プログラムを作る作業を，より簡単にする必要がある．Javaの国際化は，この基本理念に基づいた現時点で最も進歩的な国際化フレームワークの1つである．本稿では，Javaの国際化におけるさまざまな特徴と今後の方向性について述べる．

JDK 1.1の国際化

非英語圏の言語を扱うプログラムを作成するためには，国際化されたJDK 1.1が公開されるまで，長い間待たねばならなかった．たとえば，Javaのα版では国際化は考慮されず，β版でRuntime#getLocalizedfInputStream (), などのメソッド名が予約され，JDK 1.0.2でようやくjava.lang.Characterクラスに Latin-1の文字に対応するコードが加えられた程度であった．

この遅延の原因は，JavaSoft社に国際化フレームワークの設計経験を持つ技術者が少なかったことと，「国際化＝Unicode対応」のような安易な誤解が存在したためである．自ら作成していた国際化フレームワークの完成度が要求水準に達しなかったので，急遽方針を切り替え，Taligentの作成したJavaの国際化クラスのライセンスを取得し，彼らと協力してJDK 1.1の国際化を創り上げたと聞く．

Javaの国際化フレームワークは，以下の目標に基づいて設計されている[1)]．

- 国際化プログラムのデフォルト化
- オブジェクト指向設計
- マルチリンガル処理のサポート
- プラットフォーム独立性
- Unicodeのサポート
- 以前のバージョンに対する互換性の保証

ただし，国際化作業ははじまったばかりであり，現時点ですべての機能が実装されてはいるわけではない．

ロケールモデル

ロケール（Locale）は，一般的に同じ言語や慣習を共有している地域的，政治的，または文化的な領域を表し，Javaでは言語コード，国コード，バリアントコードという属性を持つ java.util.Localeクラスのインスタンスとして扱われる．

Javaのロケールモデルは，既存の国際化と比較して，次のような特徴を持っている．

- 図-1のようにプログラムやウィンドウの一部分に違うロケールを混在できる

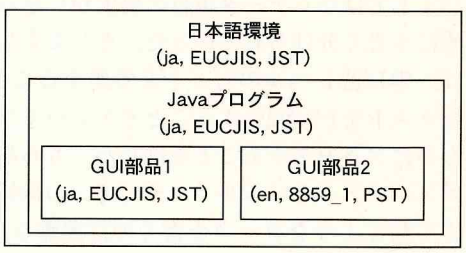


図-1　ノングローバルロケール

(ノングローバルロケール).
• ロケールを動的に変更できる.
• エンコーディングは外部との情報交換に関係するだけで,内部ではUnicodeに統一されるので,エンコーディング名はロケール情報とは独立して扱われる.
• 明示的な指定がなければデフォルト値を使用するので,特に意識しなくてもグローバルアプリケーションが容易に作成できる.

この点については,エンコーディング名,タイムゾーンについても同じである.

ただし,JDK 1.1の時点では必ずしも実装が伴っておらず,入力や表示においてロケールの指定が正しく扱われなかったり,appletやservlet単位のロケールの指定ができないなど,多くの制約がある.

JavaとUnicode

ーUnicodeについて

Unicodeは,世界中から収集した多くの文字(script)を16ビット単位の最小構成要素であるUnicode文字で表現する規格であり[2),3)],次のような特徴を持つ.

• 複数のUnicode文字を合成して,図-2のように文字を動的に合成(Dynamic Composition)できる.

$$A + ¨ \rightarrow Ä$$

図-2 Dynammic Composition

• 図-3のように,同じ文字列を表すUnicode文字シーケンス(equivalent sequence)が複数存在する.

$$B + Ä = B + A + ¨$$

図-3 Equivalent Sequence

• 中国,台湾,日本,韓国の漢字をうち字形や意味が類似している文字を統合し,同じコードポイントを与えている(Han Unification).言語の区別がないので,Unicode文字シーケンスとは別に,その言語情報がなければ,正しくテキスト処理や文字の表示ができない.

ISO/IEC 10646と,それに対応するJIS X 0221はUnicodeを元にした規格である[4)].このUCS(Universal Multiple-Octet Coded Character Set)として,16ビットのUCS-2と32ビットのUCS-4があり,UCS-2はUCS-4のBMP(Basic Multilingual Plane)に相当する.Unicode 2.0ではBMPだけが定義されているが,UCS-4の各面の割り当て作業は開始されており,今後JISで定義する第3水準および第4水準[5)]は,UCS-4の第2面にCJK統合漢字以外の漢字として追加され,これは将来のUnicodeにも追加される.

非常に冗長かつ複雑な仕様のために,Unicode文字を処理する場合には,さまざまなデータや正規化処理が必要である.プログラム中で直接操作するのは困難なだけでなく,たいていの場合は国際化を損なう結果になるので,システムが提供するメソッドを利用して間接的にアクセスすることが望ましい.

ーUnicodeとJava言語仕様

Javaは言語設計の時点からUnicode対応が考慮されていた[6)]ので,char型は16ビットであり,文字リテラルのUnicodeエスケープ(Unicode Escapes)表現など,ソースコード中のUnicodeの使用が配慮されていた.このため,Netscape Navigator 3やInternet Explorer 3がいち早くUnicodeに対応し,JDK 1.1を待たずに日本語アプレットが作成できた.

しかし,正しくはUnicode文字を16ビット単位で扱えるのは,UCS-2だけを扱う場合と,UCS-4をUTF-16形式で扱う場合に限定される.UTF-16は,Unicode2.0のサロゲート(surrogates)と同じであり,1文字を16ビット単位可変長で表現する.

将来的にUTF-16を使用するか,char型を32ビット化することになると思われるが,32ビット化は文字列処理が容易でも,既存のJava処理系の変更が必要になり,逆にUTF-16は,基本クラスや既存のプログラムの書き直しが必要になるだけでなく,プログラムの記述そのものが複雑になってしまうなど,どちらの方法も何らかの問題を抱えている.

ーUnicodeと国際化

国際化とは,プログラムから言語や文化的慣習,エンコーディングに依存する部分を分離して,それらの条件に依存しないようにするためのフレームワークを与える手法である.このような国際化の完成度を,CSI(Code Set Independence)/CSD(Code Set Dependence)という指標で判断することがある[15)].

Unicodeの採用と国際化は直接の関係はなく,Javaの国際化フレームワーク設計のたくさんある選択枝の1つにすぎない.また,プログラム中にUnicodeを意識したり,Unicodeに依存するコードが存在するべきでない.たとえ,Unicodeが将来大規模な変更を受けても,すでに作成されたプログラムが影響を受けないように実装するのが望ましい.

このような国際化プログラムを実現するためには,今まで述べてきた問題がおこらないように,次の点に注意して実装するとよい[7),8)].

• プログラム中ではStringなどの内部情報を隠した形で扱う.さらに,Unicodeのコードポイントを直接指定しない,16ビットの配列を作りUnicodeのコードポイントをインデックスとして使わない,charを受け渡しするメソッドを定義しないように注意する.
• Stringを正規化するjava.lang.String#equals ()はバイナリ比較であり,同じ文字列を表すUnicodeシーケンスを正しく比較できないので,java.lang.Collator#equals ()を使用するか,java.lang.CollationKeyクラス

を使用してバイナリ比較できるように正規化しておく．

さらにJDK 1.2では文字列のある範囲に属性（言語タグ，文法注釈，読み，フォント，スタイルなど）を設定し，それをAttributedCharacterIteratorインタフェースに従ってアクセスするjava.text.AttributedStringクラスが追加される．このクラスを使用すれば，各文字に情報を提供した側がどのように情報を格納しているかについて知る必要なしにアクセスできる．

しかし，現時点で用意されている国際化APIでは機能が不足する部分や，バグのために望む動作をしない部分も多く，どうしても上記のコーディングが必要になる場合もある．そのような場合は，問題になると思われるコードを1カ所に集めることで，将来の書き換えを容易にするとよい．

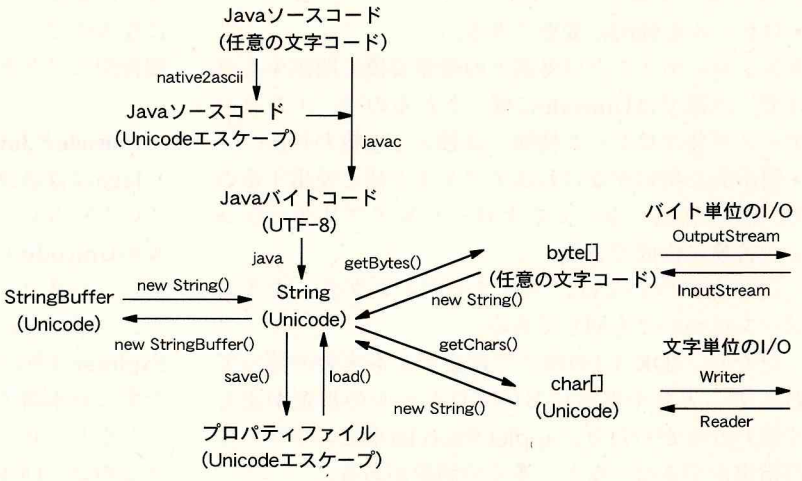


図-4　エンコーディング変換

エンコーディング変換の問題

Javaが内部でUnicodeで処理しても，ユーザがUnicodeを扱うとは限らない．むしろ，JISやEUC，Shift-JISといった既存のエンコーディングを扱う方が多いので，外部との入出力時にエンコーディング変換をおこなう必要がある．

国際化フレームワークのエンコーディング変換は，オリジナルテキストをそのまま内部に保持するフレームワークと，内部コードに変換してから保持するフレームワークに分類でき，前者の例がXウィンドウシステムやCopland[9]であり，後者の例がMule[10]やJavaである．

JavaではStringオブジェクトを中心に，図-4のようにエンコーディング変換が行われている．

内部で扱うエンコーディングを統一すれば実装は容易になるが，エンコーディング変換への依存度が高くなると共に，問題が生じた場合の対処は難しくなる．ISO 2022ベースの内部コードを採用したMuleでは既存のエンコーディングに対して親和性がよく，問題はおこらないが，日本のさまざまなエンコーディングとUnicodeの変換には，さまざまな問題が存在する[11]．

- 似た形の文字が複数あり，変換候補が絞りにくい．たとえば，「～」に似た文字は5つ存在する．「‖」，「…」，「－」も同じ問題を持つ．
- 互換性のために全角・半角文字の領域が確保されているが，特に元のエンコーディングには片方しかない「¢」，「£」，「¬」は，変換候補が2種類ある．
- 日本では，今までISO/IEC 646-IRV（ASCII）とJIS X0201で同じコードポイントに割り当てられていた「\」と「¥」，「~」と「‾」を明確に区別してこなかったが，Unicodeでは別の文字としてコードポイントが確保されているので，複数の変換候補がある．特に円記号が深刻であり，Unicodeの円記号に変換するとjavacなどのコンパイラで，バックスラッシュに変換すると金額を扱うビジネスソフトで問題が起こるので，¥問題（yen sign problem）と呼ばれる．実際には，全角文字があるので，さらに変換の選択枝は増える．

このような変換テーブルの作成方針は各プラットフォームによって異なる．またMicrosoftのような「NEC拡張文字セット」なども同時に受け入れ可能な単一の変換テーブル（Unicodeへは多対一の変換になる）を採用する例や，Apple Computerのように状況に応じて複数の変換テーブルを使い分ける例もある．

つまり，同じShift-JISファイルがプラットフォームごとに別のUnicodeシーケンスになってしまう．現時点では互いに他の変換には対応していない．このために，Windows 95のJavaプログラムのテキストフィールドから「～」を入力すると他の文字に化けることがある．今後コンバータやjava.text.Collatorクラスを改良すると同時に，文字のマッピングの標準化が必須であろう．

JDK 1.2の変更点

JDK 1.1では，開発期間を短縮するためにベースプラットフォームの国際化機能を最大限に利用した．しかし，AWTのような各プラットフォームのGUI部品を内部で使用するツールキットでは，提供される機能が各プラットフォームの機能集合の最大公約数に制限されるだけでなく，プラットフォーム間の挙動の違いが問題になる．たとえば，インプットメソッドやアウトプットメソッドの部分で次のような制限があった．

• プラットフォーム依存の機能を使用できるGUI部品が限定される．たとえば，日本語入力ができるのはTextFieldとTextAreaだけである．
• 動的変更ができない．たとえば，日本語入力はデフォルトロケールが日本語の場合だけ可能であり，GUI部品ごとにロケールを設定しても，デフォルトロケールがそのまま使われる．
• 使える機能が限定される．たとえば，アラビア語のように右から左，左から右の書き方向の異なるテキストの混在をBIDI（Bi-directionalの短縮形）と呼ぶが，これはすべてのプラットフォームで利用できないので提供されない．

そこで，JDK 1.2では，各プラットフォームの国際化機能に頼らずに，低レベルから独自にGUI部品や国際化機能を実装したJFC (Java Foundation Classes) を採用することで，高機能でポータブルな国際化フレームワークを実現することを目指している．

これによって，JDK 1.1に存在した制約の大部分が取り除かれるだけでなく，今まで作成するのが難しかった日本語ワープロや，マルチリンガルアプリケーションを作成するための機能が提供される．

—Input Method Framework

Input Method Framework[12] では，**図-5**のようにプログラムがインプットメソッドにアクセスするためのInput Method APIと，インプットメソッドを実装するためのInput Method Engine APIが提供される．なお，IIIMP（Internet-Intranet Input Method Protocol）[13] は，特定のプラットフォームに依存しないインプットメソッドプロトコルである．

JDK 1.2の段階ではプラットフォームのインプットメソッドだけを使えるInput Method APIだけが提供される予定である．

—Java 2D API

Java 2Dでは，ヘブライ語やアラビア語のように書き方向が異なるテキストの混在がサポートされ，国際テキストを扱えるようになる[14]．ただし，文字の前後関係が決定されないと，実際に文字をどのようにレンダリングするかが決定されなくなるので，行単位でバッファリングしてから実際の描画処理を行うようになる．

ほかに英語やアラビア語の合字（ligature）もサポートされ処理が複雑になるために，カレットの表示移動，文字の選択やハイライト処理などがサポートされる．

さいごに

本稿では，JDK 1.1およびJDK 1.2の特徴と，現在抱えている問題について述べてきた．

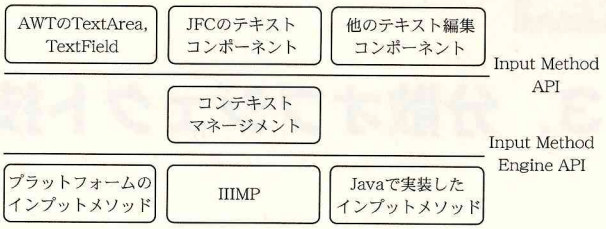


図-5　Input Method Framework

ただし，国際化の実装のバグは，実際にそのロケールを使用している人間でないと発見しにくい．そこで，正しく日本語を扱うためには，国際化機能を使いながら問題を洗い出し，それらを積極的にフィードバックしなければならない．

最後に，Unicodeを採用した国際化フレームワークの実装例は少なく，Unicode 2.0の仕様をすべて実装した例はまだない．特に異なるプラットフォーム上で動作するJavaでは，ほかでは発見されないようなさまざまな問題点も明らかになるだろう．Javaの国際化が，Unicodeのアンバランスな設計が原因で発生するさまざまな問題点を隠蔽し，すべての仕様を妥当な実行性能で実装できるかどうかは，Unicodeの有用性について討論する上で注目すべき事例である．

参考文献
1) JavaSoft: JDK 1.1 Internationalization Overview (1998). (http://www.javasoft.com:80/products/jdk/1.1/docs/guide/intl/intlTOC.doc.html)
2) The Unicode Consortium: The Unicode Standard, Version 2.0, Addison-Wesley (1996).
3) Lunde, K.: 日本語情報処理, ソフトバンク (1995).
4) 日本工業標準調査会: 国際符合化文字集合（UCS）－第一部体系及び基本多言語面 JIS X 0221-1995（ISO/IEC 10646-1:1993），日本規格協会 (1995).
5) 芝野耕司: JIS漢字の拡張計画 7ビット及び8ビットの2バイト情報交換用符合化漢字集合－第3水準及び第4水準 (1996). http://www.tiu.ac.jp/JCS/
6) Gosling, J., Joy, B. and Steele, G. L.: The Java Language Specification, Addison-Wesley (1996).
7) Campione, M. and Walrath, K. : Writing Global Programs, The Java Tutorial. (http://java.sun.com/docs/books/tutorial/intl/)
8) Davis, M. : Java Cookbook: Creating Global Applications (1997). http://www.ibm.com/java/education/globalapps/ (Mark Davis著，村田稔樹 訳，風間一洋監訳: Java Cookbook: グローバルアプリケーションの作成 (1997). http://www.oki.co.jp/OKI/RDG/JIS/java/GlobalJava/index.htm)
9) McConnell, J.: Internationalization on Copland, Eighth International Unicode/ISO 10646 Conference (1996).
10) 錦見美貴子，高橋直人，戸村　哲，半田剣一，桑理聖二，向川信一，吉田智子: マルチリンガル環境の実現X Window/Wnn/Mule/WWWブラウザでの多国語環境, プレンティスホール (1996).
11) TOG/JVC CDE/Motif技術検討WG: Unicodeとユーザ定義文字・ベンダ定義文字に関する問題点と解決策 (1997).
12) JavaSoft: Input Method Framework Design Specification (1997).
13) Hiura, H.: Platform Independent Input Method Protocol for the Internet, Network Computer（NC）and Java, Tenth International Unicode Conference (1997).
14) Davis, M., Felt, D. and Raley, J.: International Text in JDK 1.2 (1997). http://www.ibm.com/java/education/international-text/
15) Hiura, H.: Unicode and Internationalization with UNIX and the X-Window's System, Eighth International Unicode Conference (1996).

（平成10年2月25日受付）

特集

3. 分散オブジェクト技術

平野　聡
電子技術総合研究所／
HORB Open &
Ring Server Project

分散処理の動機

ネットワーク上にある複数の計算機にまたがって「1つのJavaプログラム」を動作させる技術がJava用分散オブジェクト技術（Java DOT）です．本稿では，現在広く普及しつつあるJava DOTの全体像を捉えるために，Java DOTによって分散システムの記述がいかに簡単になるか，Java DOTの基本原理，そして，Java DOTの現状とその動向について述べます．

解説を具体的にするために，ひとりの脳科学の実験家に登場してもらいましょう．彼は情報処理に永くたずさわっており現在は脳機能の解明を仕事としています．ここ数カ月，脳のサンプルを薄くスライスし，記憶の痕跡を示す特殊な印が現れているかを顕微鏡で観察することが日課です．彼は10年前からMacを愛用していますが，彼の顕微鏡はWindowsのパソコンでコントロールされています（**図-1**）．顕微鏡から取り込んだ画像データの解析には計算センターのスーパーコンを使います．現在は顕微鏡のワンショットごとにいちいちファイル転送をしているため，手間がかかります．もっとインタラクティブに，「Windowsの計算機で取り込んだ脳の画像データをスーパーコンピュータで処理し，その結果を即座に手元のMacで3次元画像をくるくると回して表示する」にはどうしたらよいでしょうか．実験家は簡単なプログラムを書くことはできますので，これを「1つの小さなプログラム」として書くことが希望です．

この例は小さな例ですが，生産ライン管理システム，在庫管理システム，受発注システムなど，ネットワークで結合されたたくさんの計算機上で，一連の処理を行うプログラムを協調・分散して動作させる分散処理は私たちに身近なものです．計算機にはWindows，Mac，Unix，メインフレームなど多様な機種があります．これらの異機種を混在して相互運用することが可能な，分散言語や分散OSを実現することが，情報処理 技術開発の大きな目標でした．Java以前の分散処理技術や参考文献についてはタネンバウムの著書[1]を参照してください．この本にはいろいろな分散処理技術が紹介されていますが，残念ながら，実験家の環境で共通して利用できるものは1つもありません．Javaの出現前の時点で利用可能なプログラミング手法を使うならば，おそらくは，Windows，スーパーコン，Mac それぞれのマシンのために1つずつプログラムを書いて，プログラム間はファイル転送を行うか，もう少しプログラム能力があれば，それぞれのプログラムをTCP/IPのソケットで接続してデータ転送を行うようなプログラムを書いていたでしょう．

Java用分散オブジェクト技術の概要

1995年，Javaの出現によって，計算機の機種に依存せずにプログラムを書くことが可能になりました．Write Once, Run Anywhere. どの機種の上でも，同じプログラムが動作します．では，実験家もJavaを使えば，彼の分散システムを「1つの小さなプログラム」として簡単に書くことができるでしょうか？　否，依然として計算機ごとにプログラムを書いて，ソケットでデータ転送を行う必要があります．なぜなら，Javaのプログラムは依然として1つの計算機の1つのプロセスの中でしか動かないからです．これを克服するために，Java用のDOTが一斉に開発され，またたく間に実際のシステム開発に利用されるようになりました．

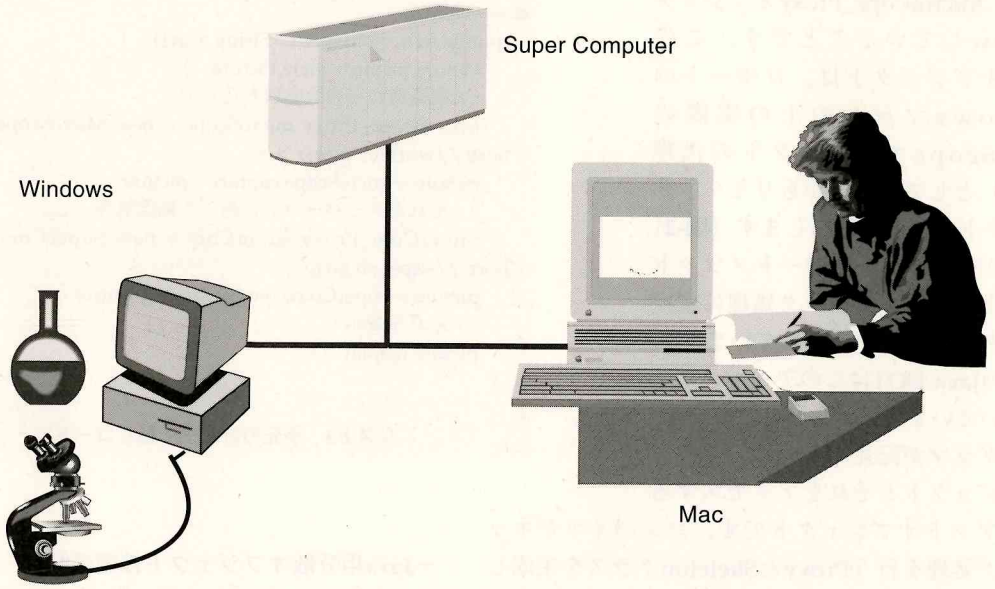


図-1　ある実験家の実験システム

Javaには大域変数やポインタがない，メモリ管理もガベジコレクションによって自動化されているといった，CやC++で取り扱いが難しかった問題がなくなっているために分散処理に向いています．そのため，長い間研究段階に留まっていた分散処理技術を実際に使える技術として実現することが比較的容易でした．Javaはもはやあらゆる機種の上で動作します．Java DOTを使えば，実験家は望み通りのプログラムを書くことができます．

一分散オブジェクトの記述

では，実際に実験家のシステムを記述してみることにしましょう．同じJavaを用いてもDOTの種類によって記述の方法はまったく異なります．ここでは，記述が簡潔なHORB[2)~3)]を用います．まず，計算機間でやりとりされる画像データをオブジェクトにしてみましょう．**リスト1**は画像データであるPictureクラスです．画像データを格納する変数と，画像処理，表示を行うメソッドを持っています．

顕微鏡はWindowsに接続されており，顕微鏡画像をイメージとして取り込むことができます．スーパーコンは画像データオブジェクトを高速に画像解析します．これらのオブジェクトを**リスト2**に示します．実験家のMacの上で動作するクライアント側のコードの主要な部分が**リスト3**です．mainから始まるこのプログラムは，win1.etl.go.jpというWindowsマシン上に顕微鏡オブジェクトを作って，脳のスライスの顕微鏡画像を取り込みます．次にスーパーコンであるsuper.etl.go.jpにスーパーコンオブジェクトを作り，脳画像を送り込んで画像処理を行います．処理が終了したらディスプレイに表示して終了します．このプログラムはそれぞれ数行からなる合計4つのクラスから構成されており，「小さな1つのプログラム」という条件に合格します．実験家はわずか1時間で完成したこのプログラムによって，毎日数時間の実験時間を節約できるようになりました．

一分散オブジェクトの基本原理

リスト3のプログラムは，みたところ通常のJavaプログラムとほとんど違いがありません．違うのは，MicroScopeオブジェクトをnewによって生成する代

```
class Picture {
   Image image;          // 画像データ
   void imageProcessing () {...} // 画像解析
   void display () {...}        // 結果の画像の表示
}
```

リスト1　画像データオブジェクト

```
class MicroScope {
   Picture capture (Picture data) {
      // 顕微鏡から画像を取り込む
      data.image = 取り込み ();
      return data;
   }
}
```

```
class SuperCom {
   Picture computation (Picture data) {
      data.imageProcessing ();
      return data;
   }
}
```

リスト2　顕微鏡オブジェクトとスーパーコンオブジェクト

わりに，MicroScope_Proxyオブジェクトをnewしていることです．このProxyオブジェクトは，リモートのWindowsマシンの上の実際のMicroScopeオブジェクトの代理（proxy）として，あたかもリモートオブジェクトのごとく動作します（**図-2**）．Proxyの内部には，リモートメソッドの呼び出しをネットワーク処理に変換するスタブコードが含まれています[4]．すべてのJava DOTはこのアーキテクチャを用いています．

```
class Client {
    public static void main（String argv[]）{
        Picture picture = new Picture（）；
        // 顕微鏡から画像を取り込んで
        MicroScope_Proxy microScope = new MicroScope_Proxy
("horb://win1.etl. go.jp/")；
        picture = microScope.capture（picture）；
        // それをスーパーコンに送って画像処理
        SuperCom_Proxy superCom = new SuperCom_Proxy
("horb://super.etl.go.jp")；
        picture = SuperCom.computation（picture）；
        // それを表示
        picture.display（）；
    }
}
```

リスト3　手元の計算機で動くコード

プログラマが記述するのは，リモートオブジェクトとそれをアクセスするクライアントオブジェクトです．コンパイラがネットワーク処理を行うProxyとSkeletonクラスを生成します．ProxyとSkeletonは，分散オブジェクトの通信路であるORBを介してリモートメソッドの実行を行います．このプログラムで中心的な役割を果たしているのがリスト1の画像データオブジェクトです．このオブジェクトはMac上で生成された後，Windowsに送られたり，スーパーコンに送られたりして仕事を進めていきます．このオブジェクト転送の機能は，分散システムの拡張性や再利用性を著しく高めます．たとえば，新しい画像解析アルゴリズムを使用したい場合は，画像データオブジェクトを継承するサブクラスに新アルゴリズムを実装すれば，他のクラスは一切変更する必要がありません．また，画像データオブジェクト自身に次の処理を決する処理を行わせればモバイルエージェントとなり，システムは自律的な動作を行うようになります．このようなオブジェクト転送機能は，複雑な構造のオブジェクトを異なる計算機にそのままの形で転送するオブジェクト・シリアライゼーションに基づいています．オブジェクト・シリアライゼーションはJDK1.1からJavaの標準機能になりましたが，すべてのJava DOTがオブジェクト転送の機能を備えているわけではありません．

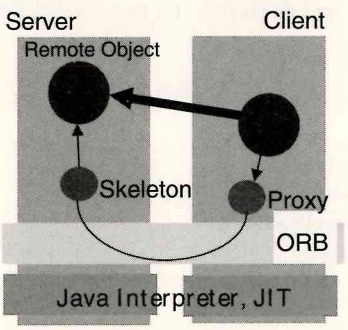


図-2　分散オブジェクト技術のアーキテクチャ

―Java用分散オブジェクト技術の利点

以上のJava DOTの利点をまとめると以下のようになります．

- 分散オブジェクト技術がネットワークコンピューティングのフレームワークとして機能します．プログラマがデータ転送やセキュリティを記述する必要がありません．プログラマはアプリケーションの記述に専念できます．
- 通信処理でバグが出ないため，開発期間が大幅に短縮されます．
- アプリケーションの拡張性や再利用性が高まります．
- 異機種が混在するネットワーク上で相互運用可能であり，プログラムも機種にかかわらずポータブルです．
- コンパイラやORB自身がJavaで記述されているDOTでは，ORB自身も機種を選びません．

分散オブジェクトの現在と将来

Java DOTは急激な勢いで進化しています．本章ではJava DOTの現状と進化の方向について概観します．

―分散オブジェクト技術の分類

現在のJava DOTを周辺技術も含めた分散オブジェクトフレームワークとして分類すると以下の3種類に分けることができます．

(1) OMG CORBA

OMG（Object Management Group）は分散オブジェクト技術の確立を目的に，1989年に創立された世界最大の技術コンソーシアムです．CORBAのAはアーキテクチャを表しており，CORBAの仕様はOMG各社の分散オブジェクト製品の共通のアーキテクチャを意味しています．現在公開されているCORBA 2.1は，ORBコア[5]，共通サービス[6]，共通ファシリ

ティ[7]から構成されています．CORBAはアーキテクチャですから，その実装は各ベンダに依存しており，CORBA製品は準拠のレベル，実現する機能の範囲，独自拡張の度合い，相互運用性などが非常に多岐に渡っています．文献10）によれば，CORBA準拠のORBは製品，フリーなど合わせて40種に達するそうです．ベンダが違えばCORBA準拠製品同士でも相互運用性がない状態が永く続いていましたが，CORBA2から仕様化されたIIOP（Internet Inter-ORB Protocol）によって，相互の通信ができるようになりました．1996, 7年はJava上にIIOPを実装したCORBA製品のリリースが相次ぎました．JDK1.2にはJavaIDLが標準で装備されています．CORBAはプログラミング言語に依存せずにC，C++，COBOL， Javaなどの間で相互にオブジェクト呼び出しができるよう，リモートオブジェクトの定義をOMG IDL（Interface Definition Langu-age）言語で行います．IDLコンパイラによってIDLからProxyが生成され，通常のプログラミング言語で書いた実装のコードと共に実行します．OMG IDL言語はJavaが生まれる前に定義された言語ですから，そのオブジェクトモデルはJavaのオブジェクトモデルと異なっており，プログラミングを難しくしている面があります．オブジェクト転送は，現在，参照渡しのみが利用可能ですが，現在策定中のCORBA3によって，オブジェクトの値渡しができるようになる予定です[18]．OMGはビジネスオブジェクト，モーバイルエージェント，電子商取引など広範囲に渡って仕様を策定中です．これらの仕様がカバーする範囲の多くはJava APIと重なっており，システム設計時にどちらを選ぶかが問題となっています．

(2) Windows環境を前提とするMicrosoftのDCOM

Microsoftのオブジェクト技術であるActiveXコンポーネントは，現在オブジェクト市場のほとんどを占めており，Windowsのユーザは毎日その恩恵を被っています．ActiveXのオブジェクトモデルはCOM（Common Object Model）と呼ばれ，やはり言語に依存しません．オブジェクトはMicrosoft ODL（Object Definition Langu-age）言語によって記述し，通常のプログラミング言語で書いた実装のコードと共に実行するのはCORBAと同様です．COMを分散化する技術であるDCOM（Distributed COM）[8]は，膨大なCOM資産を無変更でリモート実行することを可能としています．Microsoftは次期Windows NTに向けてWindows の分散アーキテクチャDNA[9]を策定しています．DNAはオブジェクト・ディレクトリ，トランザクション，電子商取引などを含み，JavaからWindowsのAPIでシステム開発のすべてが記述できるようになるとされています．

(3) Java環境を前提とするJava専用分散オブジェクト技術

Javaの豊富なAPIの利用を前提とするJava専用DOTを用いれば，CORBAやWindowsに依存しない分散オブジェクトフレームワークの構築が可能です．IDLを書くことなくJavaだけでリモートオブジェクトの記述が可能なので，IDLやODLにある技術的な制約を受けません．自然なオブジェクトの記述が可能となるため，プログラマの負担は大幅に軽減されます．CORBAやDCOMにはないオブジェクト転送も可能です．新たにシステムを開発する場合には，有力な選択肢となっています．SunのRMI[11]はJDK1.1から標準装備となったJava DOTで，最も基本的な分散オブジェクト機能を提供します．Microsoftが同社のWWWブラウザにRMIなどのAPIを実装しなかったため，両社の間で訴訟となっています．HORB[2),3)]は，筆者によって1995年に公開された世界で最初のJava DOTです．相互運用性や記述性など，永かった分散処理の夢を初めて実現しました．記述が簡潔で性能がよいこと，また，広く普及しているJDK1.0ベースのWWWブラウザでオブジェクト転送が可能なのはHORBだけであることから，Javaの初期から現在まで広く使われています．ObjectSpaceのVoyager[12]はJavaのReflectionを有効に使ったモーバイルエージェント機能を有するJava DOTで，高い人気があります．Spaceというオブジェクトグループに対してグループコミュニケーションをする機能が特徴です．

一分散オブジェクト技術発展の動向

ここではJava DOTの動向のうち，主要と思われる3つについて述べます．

(1) 分散オブジェクトフレームワークからアプリケーションフレームワークへ

ビジネスシステムのアーキテクチャは，サーバ上のデータベースをクライアントからアクセスする2層構造から，ビジネスロジックをアプリケーションサーバ内に分離した3層構造へと変わりつつあります．アプリケーションサーバとデータベース間はJDBCで，アプリケーションサーバとクライアント間はDOTで接続します[13]．さらに，業務ごとのビジネスロジックを実装したコンポーネントを，アプリケーションサーバ上にあらかじめ用意している，アプリケーションフレームワークが形を現してきました．IBMのSan Francisco Frameworks[14]はその一例です．アプリケーションフレームワークを用いると，業務内容にあったビジネス・コンポーネントを選択し，それとクライアントのGUIをカスタマイズすることがシステム開発となります．システムを1から作成し，人月単位で報酬を得ているシステム開発のスタイルが大きく変わるでしょう．

(2) 共通バスとしてのIIOPの採用

CORBA3になるとIIOPがオブジェクト転送機能を

備えるため，Java専用DOTも機能をあまり損なうことなくIIOPを使うことができるようになります．すると，IIOPを介して既存システムのCORBAと接続することが可能となります．そのため，VoyagerはすでにIIOPを実装していますし，RMIとHORBもIIOPの実装を表明しています．

(3) オブジェクトモデルのJava化

システム開発の記述言語にJavaが占める割合が高くなるため，CORBAやCOMもJavaへの対応を急いでいます．CORBAでは，IDLを書くことなしに，Javaのインタフェース定義からIDLを生成する仕様が検討されています[17]．VisiGenicのVisiBroker[15]はすでにjava2idlコマンドによってそのような機能を実現しています．Microsoft COMの次世代版のCOM+[16]では，ODLなしでJavaやC++などの言語で直接リモートオブジェクトを定義できるようになります．また，COMは実装の継承をサポートしていませんが，COM+では実装の継承は単一継承，インタフェースの継承は多重継承とJavaのオブジェクトモデルに適合されています．COM+用C++もimportやimplementsなどJava風のプリミティブが追加されます．

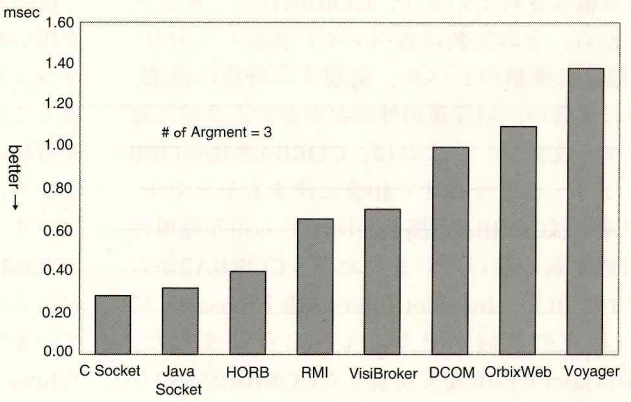


図-3 リモートメソッド呼び出しのベンチマーク

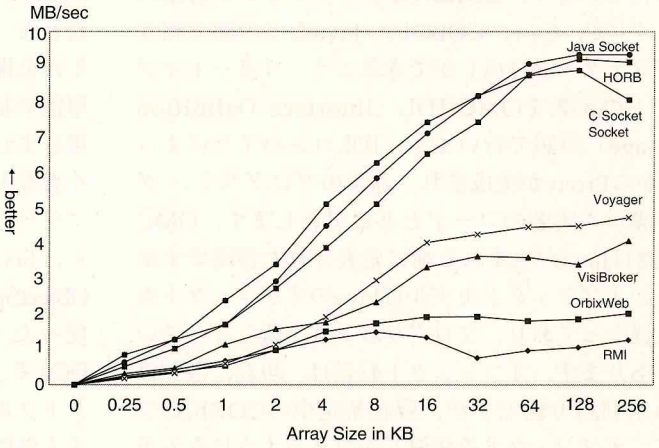


図-4 データ転送速度のベンチマーク

一現在の処理系間での性能比較

現在のJava DOTは性能が低いものが多く，システム開発上の問題点の1つとなっています．現状を調べるために代表的なJava DOTのベンチマークを行いましたので紹介します[3]．**図-3**にリモートメソッド呼び出しのベンチマーク結果を示します．リモートメソッドはint foo（int，int，int）です．**図-4**はvoid foo（byte[]）をリモート実行した際のデータ転送速度の測定結果です．測定環境は100Base-TXで接続された2台のPentiumII 266MHz Windows NTを用いました．Java実行系にはいずれもSunとMicrosoftのJITを使用しています．ソケットと同程度の速度を実現しているのはHORBだけでした．JavaはCと比べてまだ実行速度が遅く，Cとソケットのプログラムから Java DOTに移行するには十分な注意が必要です．DOTのベンダは性能向上の努力が求められています．

参考文献

1) Tanenbaum, A.S.: Distributed Operating Systems, Prentice Hall (1995). 水野他訳, 分散オペレーティングシステム, プレンティスホール (1996).
2) Hirano, S.: HORB: Distributed Execution of Java Programs, Worldwide Computing and Its Applications, Springer Lecture Notes in Computer Science 1274, pp.29-42 (1997).
3) Hirano, S., Yasu, Y. and Igarashi, H.: Performance Evaluation of Popular Distributed Object Technologies for Java, Proc. of ACM 1998 Workshop on High-Performance Network Computing (1998), http://www.cs.ucsb.edu/conferences/java98
4) Shapiro, M.: Structure and Encapsu-lation in distributed Systems: The Proxy Principle, ICDCS, pp.198-205 (1986).
5) Object Management Group, The Common Object Request Broker: Architecture and Specification 2.1 (1997), http://www.omg.org
6) OMG, Common Services Architecture (1995).
7) OMG, Common Facilities Architecture (1995).
8) Brown, N. and Kindel, C.: Distributed Component Object Model Protocol－DCOM/1.0, Internet Draft (1996).
9) http://www.microsoft.com/sitebuilder/dna/
10) 鈴木: CORBA技術の全貌と今後の動向, Dr.Dobb's Journal Japan, pp.28-43 (Aug. 1997).
11) JavaSoft, Java Remote Method Invocation Specification, 1997, http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html
12) Glass, G.: ObjectSpace Voyager－the Agent ORB for Java, Worldwide Computing and Its Applications (1998), http://ci.etl.go.jp/wwca98/
13) Duan, N.: Distributed Database Access in a Corporate Environment Using Java, 5th Int. WWW Conference (1996).
14) http://www.developer.ibm.com:8080/welcome/java/sfindex.html
15) http://www.visigenic.com
16) Kirtland, M.: Object-Oriented Software Development Made Simple with COM+ Runtime Services, Microsoft Systems Journal, Vol.12, No.11 (1997).
17) IBM, et al.: Java to IDL Mapping, OMG TC Document orbos/98-01-07.
18) BEA, et al.: Objects by Value, OMG TC Document orbos/98-01-01.

（平成10年2月23日受付）

エッセイ Javaとプログラム言語研究

松岡　聡／東京工業大学

Javaの登場により，欧米ではプログラム言語研究に新展開が見えつつある．言語としてのJavaは，Algol/Pascal，CLU，Smalltalk，Modula-3，C++などの設計思想を取捨選択したもので，良好な設計ではあるが，技術的に目新しいところはない．しかし一方で，その効率的な実行サポート，それによるインターネットを含む新しい計算環境への対応・発展，という点で，従来からのプログラム言語研究の諸分野に新たな研究課題を生み出している：

－JITに代表される，効率的な動的コンパイラ技術
－マルチスレッド，同期操作の効率化
－ガーベジコレクションアルゴリズムの速度・質の向上
－セキュリティ機能のプログラム言語への取り込み
－超広域高性能計算，分散エージェント計算系など，広域計算をターゲットとした計算系
－型システムの新たな応用：さまざまなパラメタ型の追加の提案，Genericity,Parasitic Methods，バイトコードの型システムなど
－その他のモービルコード関係のテーマ：デバッグ，分散コードの一貫性維持，など

これらの課題は，従来からの基礎研究がJavaによりその適用が現実化したり，理論・技術の拡張が必要となったものである．たとえば，Javaでのガーベジコレクションアルゴリズムやパラメタ型の提案などは，既存の理論や技法の応用であるが，Javaの普及により研究の具体性や実用時の信頼性が増している，といえる．また，超広域高性能計算・動的コンパイラ・バイトコードの型システムのようなテーマは，Javaに特化した研究ではないものの，Javaの誕生によりより盛んになったり，価値を高めた，といえよう．ここ1～2年のOOPSLA/ECOOP/POPL/PLDIなどのオブジェクト指向・プログラム言語研究の主要学会では，このような論文の発表が目立ち始め，また，ACM Workshop on Java for High-Performance Network Computingなど，専門のワークショップもいくつか開催されている

しかし，我が国のこれらの分野の貢献は驚くほど少ない．無論，貢献が皆無ではなく，IBM東京基礎研究所のAglets・東芝のPlangentなどのエージェントシステム，我々のNinfプロジェクトの一環であるNinfletや早稲田大学のJavaによる超広域高性能計算システム，などがある．また，コンパイル時に自己反映計算を行う筑波大のOpenJavaや電総研のEPP，神戸大のJavaベースの並列言語の研究，および富士通と我々の共同研究のOpenJITなどがある．また，電総研のHORB，PFUのTeikadeなど，従来の要素研究を高い完成度でパッケージングし，海外でも広く知られ，さまざまな研究のベースとして用いられているソフトウェアシステムも存在する．さらに，IBM東京基礎研究所のJITコンパイラは世界的にも大変高速な処理系の1つである．しかし，これらは少数の例外で，欧米と比較すると，過去のソフトウェア研究と同様，そのアクティビティは低い．これは，我が国でのJavaの普及度や知名度の高さからすると，極めて異例であると考える．

これはなぜか？　筆者は，1つの理由はJavaに対するソフトウェア研究におけるプロフェッショナリズム－つまり，技術を客観的にサーベイし評価し応用する専門家としての能力－の希薄さに起因する，と主張したい．たとえば，先年，通称「Java・・・大賞」が開催され，筆者はその技術部門の審査員長を務めたが，各審査員の多くの作品に対する評価は，「『既存のソフトウェア技術を単にJavaで実装しました』であり，新たなソフトウェア技術はあまり認められない．」といった厳しいものであった．また，国内のJava関係の論文の査読時に，従来のソフトウェア研究のサーベイや技術比較をほとんど行わず，Java以前の先行研究の認識が欠如している（非プロフェッショナル的な）論文が目に付くのも事実である．

ソフトウェア研究におけるプロフェッショナルは，ソフトウェアの技術の現状に対する広範な認識のみならず，ソフトウェアの基礎的な理論や技術に関する深い知見が必要となる．逆に，それらなしには最先端の研究開発はおぼつかない．喩えていえば，自動車の趣味人は書店に並ぶ車の一般紙を購読するが，最先端の自動車を設計するプロフェッショナルになるには，それらを幾ら熟読しても不可能である．むしろ，大学の機械工学などで数学，力学，機械設計学，材料学，熱力学，電子回路などを専門知識として学び，さらに各自動車メーカに膨大に蓄積された技術およびノウハウを習得して，初めて自動車工学のプロフェッショナルが育成される．同様に，上記のJavaに関する分野で最先端の成果を上げられるプロフェッショナルとしてのソフトウェア研究者の育成のためには，数学・計算機科学の種々の基礎教育，並びにコンパイラ，言語処理系，分散システム，オブジェクト指向などに関する深い専門教育が不可欠である．逆に，やみくもにJava技術のみを若い技術者や研究者がフォローするのは，プロフェッショナルの養成にはつながらないと考える．

新編集体制による「情報処理」で，くしくも新しい計算機科学の潮流の代表格としてJavaを特集することになった．しかし，我が国からJavaに関する，あるいはJavaの到来によって促された新しい研究がなされるためには，特集の内容は単に商業誌のようなJava技術の現状の表層的紹介にとどまらず，「情報処理」ならではの，ソフトウェア技術のプロフェッショナルとしての観点からの技術的背景，過去の研究成果，Javaへの応用の仕方，などの解説が強く望まれる．その点に関して，今回の特集は一種の試金石になったといえよう．今後も「情報処理」の特集が特定のテーマの表層的な紹介にならず，プロフェッショナルな視点からの深い知見に満ちた解説となることを望みたい．

（平成10年2月24日受付）

特集

4. 応用の新展開

ーメタコンピューティングへの応用ー

高木浩光
名古屋工業大学

松岡　聡
東京工業大学

メタコンピューティング

1997年10月，RSA Data SecurityのThe RSA Data Security Secret-Key Challenge[1]において，distributed.netのチーム[2]が広域分散処理によって56bit RC5を破ったことが話題となった．このプロジェクトでは，問題を分割しクライアントに配布するサーバを中心に据え，部分問題を解くクライアントプログラムをWindowsやMacOSをも含む各種OS用のバイナリ形式で用意し，ボランティアを募って彼らにこれをダウンロードして実行してもらうことで，多数のクライアントに結果を報告させるという方法をとった．56bit RC5は250日ほどで破られ，この間50万もの異なるIPアドレスの計算機が参加したという．

このような，ネットワーク上の多数の計算機を包括的に1つの仕事のために使用するというアイデアは，「メタコンピューティング」[3]として知られ，近年，ネットワーキング技術の進歩とインターネットの爆発的な拡大により，現実のものとして再び注目されてきている．distributed.netの例のようなパソコンの利用のほかにも，スーパーコンピュータを利用した本格的なプロジェクトも立ち上がってきており，たとえば「I-WAY」プロジェクト[4),5]は，北米の17の組織にあるスーパーコンピュータを互いに接続して，この上で，60の参加グループにより開発されるさまざまな分野の科学技術計算アプリケーションを実行させようというものである．

ここ数年でメタコンピューティングが特に注目されるようになったもう1つの理由に，Javaの登場がある．distributed.netのプロジェクトがあらゆるOS用のバイナリを用意したように，メタコンピューティングの成功には異機種混成の環境にどう対処するかが1つの鍵となる．アプリケーションをJavaで記述しておくことで，1つのコードの提供だけで，Java VM（Virtual Machine）の動作する任意の環境でそれを動作させることができる．この点に着目してJava AppletによってRC5を破ろうというプロジェクトがいくつか存在する[6),7]．一般に，Javaによる計算はC言語などにより書かれた同じプログラムに比べて実行速度が遅いと考えられ，このような目的にはそぐわないのではないかと考えられがちであるが，これらのプロジェクトでは次のようにその意義を主張している．第1にJavaの実行性能は高度なJust-In-Time（JIT）コンパイラの登場により将来はFORTRAN，C並みに高速化されると期待できる．第2に，Java Appletを動作させられるWWWブラウザさえあれば，処理系の面倒なインストール作業を必要とせずに，誰でも簡単に実行させられるため，より多くのボランティアを募ることができる．

Javaによる汎用メタコンピューティングシステム

第3はJavaのセキュリティ機能にある．ここで，セキュリティの意義を明確にするために，メタコンピューティングの運用形態を**表-1**のように分類してみた．PVM（Parallel Virtual Machine）[8]やMPI（Message Passing Interface）[9]を通信ライブラリとして使用した，NOW（Network Of Workstations）[11]上の仮想並列計算機システム，またCondor[10]などのシステムは，使用する各計算機に利用アカウントを必要とするものであるため，計算機資源提供者とアプリケーション利用者は同一であり，それぞれ特定

の者に限られる．これに対して，Ninf[12]およびNetSolve[13]などのシステムは，利用アカウントを持たない不特定の者に対して，リモート手続き呼び出しによる計算機資源の利用を提供するものであるが，実行されるアプリケーションはあらかじめ管理者によってインストールされているものに限られる．一方，冒頭に述べたdistributed.netのようなアプローチは，不特定の者の計算機資源を利用しているが，ボランティア達がこのプロジェクトを信頼（提供されているクライアントプログラムが自分の計算機に害を及ぼさないと信頼）できる必要があるため，アプリケーション作成者と利用者は特定の者に限られる．

表-1　メタコンピューティング運用形態の分類

代表的なシステム	計算機資源提供者	アプリケーション作成者	アプリケーション利用者
PVM, MPI, Condor	特定	特定	特定
Ninf, NetSolve	特定	特定	不特定
distributed.net	不特定	特定	特定
Javelin, Ninflet	不特定	不特定	不特定

これに対して，計算機資源提供者，アプリケーション作成者，利用者それぞれが不特定であることを許すシステムがJavaをベースとしていくつか提案されている．不特定多数の計算機資源での任意のプログラムの実行を現実的なものとするためには，プログラムが安全に（提供された計算機に害を与えないように）実行されることが保証される必要があるが，Javaがちょうどこの特性を備えている．Javaは，

- ファイルやネットワークへのアクセス，スレッドに対する操作などの可否を制御するjava.lang.SecurityManagerクラス．
- 不正なメモリ番地へのアクセスを起こさないことを保証する，排除されたポインタ演算や添字チェックされる配列といった言語仕様．
- 不正なコードが含まれていないかチェックするbytecode verifier．
- 異なるClassLoaderからロードされた同名のクラスを別クラスとして扱う仕組み．

といったセキュリティのための機構を持っており，これらに基づくJavaの安全性は，その1つの応用である「Applet」のフレームワークにより実証されている．このような特徴から，不特定多数による不特定多数のためのメタコンピューティングシステムを実現する手段として，Javaが注目されている．

Appletによるメタコンピューティングシステム

このような観点に基づいて提案されたJava Appletによるメタコンピューティングシステムに，Javelin[14]，Charlotte[15]，JET[16]，IceT[17]，POPCORN[18]，Bayanihan[19]などがある．Javelinは，クライアント，ブローカ，ホストの3者から構成され（**図-1**），クライアントがアプリケーション利用者，ホストが計算機資源提供者に相当する．ブローカは，クライアントからの要求を適切なホストに割り当てる役割を担っている．クライアントは目的のアプリケーションをJava Appletとして記述し，そのクラスファイルをどこかのWWWサーバに置いておく（図中（2））．一方ホストはJava VMを持つWWWブラウザである．ブローカは小さなAppletとサーバプログラムから構成されている．クライアントは，あるアプリケーションを実行させたいとき，そのURLをブローカに登録する（図中（3））．一方ホストは，その計算機資源を提供したいときに，ブローカのWWWページにアクセスする（図中（1））．ブローカは，クライアントから要求されたAppletのURLを，利用可能なホストに渡してそのページを開くよう指示する（図中（4））．ホストのWWWブラウザはこの指示を受けるとそのURLのページを開く．その結果，クライアントのAppletがホスト上でロードされて実行される（図中（5），（6））．

Javelin以外のシステムも基本的には同様の仕組みを持つが，それぞれ次のような特色を持っている．POPCORNでは，アプリケーション利用者をBuyer，計算機資源提供者をSellerと呼び，Javelinのブローカに相当する部分をMarketと呼ぶ．MarketはCPU時間を売買する市場であり，提供した計算機資源で実際に計算が行われるとpopcoinという単位の通貨を獲得でき，次の機会にこれを支払って自分のアプリケーションを他のSellerの計算機上で実行することができる．この仕組みによって，信頼関係のない不特定の者の間でこのシステムを利用しても，公平に互いの資源を共有することができる．この仕組みにより，夜間の空いている時間帯に計算機資源を地球の裏側の昼間の地域に貸しだし，昼間は逆に借りるといった運用が現実的なものとなる．

Charlotteは，プログラミングモデルとして，fork-join型の並列プログラミングに基づいたDSM（Distributed Shared Memory）を用意している．このDSMはJavaの純粋なクラスライブラリとして用意されており，特別なVMの拡張やコンパイラを必要としていないため，そのままWWWブラウザの上で動作可能となっている．具体的には各プリミティブ型に対応するDSM版のクラス（たとえばfloatに対応するDfloatクラス）が用意されており，そのインスタンスに対するget()とset(data)でデータの読み出しと書き込みをするものとし，このget()，set(data)メソ

ッドが分散共有メモリとしてのコヒーレンス管理を行っている．Charlotteではコンシステンシモデルとして CR&EW（Concurrent Read and Exclusive Write）を採用している．

Bayanihanは，ボランティアによる計算を指向しており，できるだけ多くのボランティアを集めるためか，計算の過程を見せることに重点が置かれている．Javelinにおけるブローカに相当する仕組みはなく，問題ごとに個別のサーバが用意され，ボランティア達がこの問題のページにアクセスすることで計算が行われる．システムはいわゆるMaster-Workerパターンに基づいており，中央のサーバプログラムがMasterとして，AppletがWorkerとして動作する．Workerには対応するGUIのフレームワークが用意されており，プログラマは，目的のアプリケーションの計算ルーチンと，その結果を表示するメソッドを実装する．

また国内においても，昨年開催された「Javaに関する技術・応用・表現大賞」の技術部門に，早稲田大学から同様にJava Appletによる分散処理を実現する作品が応募されている[20]．

専用のJavaフレームワークによるメタコンピューティング

筆者らもJavaによるメタコンピューティングシステムとしてNinflet[21]を提案している．Ninfletは Appletベースではなく，専用のスタンドアロンアプリケーションとして実装している．Ninfletは独自のClassLoaderとSecurityManagerを実装しており，Appletベースでは実現不可能な細かなセキュリティポリシーを持っている．たとえば，AWTによるウィンドウの作成や，スレッド操作を禁止している．

Ninfletシステムは，Server，Dispatcher，Clientの3つから構成される．このうちServerとDispatcherはJavelinにおけるホスト，ブローカに相当する．計算資源提供者は，WWWブラウザを操作するのではなく，Serverをデーモンプロセスとして常時稼働させておく．Serverでは，DispatcherからのNinflet発行要求の受け入れの許可／拒否の条件を設定することができる．計算資源提供者は，たとえば，自分がその計算機を使用していない間だけNinfletの受け入れを許可するよう設定することができる．不許可に設定されている場合，DispatcherはそのServerにNinfletのインスタンス化を依頼しない．タイマー機能により，不在にする夜間，休日だけ許可するといったことや，スクリーンセーバが動作している間だけ許可するといったことも可能である．受け入れ設定が不許可に切り替えられたとき，すでに実行中のNinfletが存在した場合，そのNinfletに対しては退去命令が発行される．退去命令が発行されたとき，

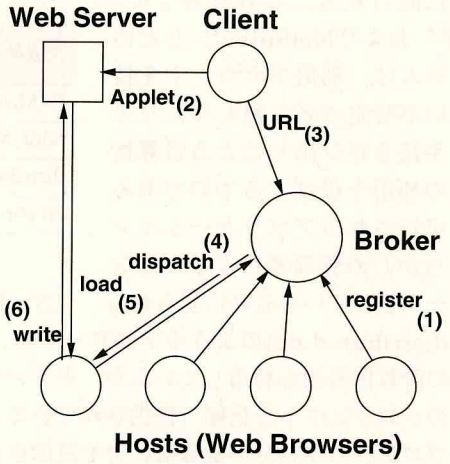


図-1 Java Appletを利用したメタコンピューティングシステムの基本構成

Ninfletは計算の途中結果を退避させることができる．退避は，Dispatcherにより指示されたところに対してなされる．退避されたNinfletは，Dispatcherにより再割当される別のServer上で再実行を開始することができる．

Appletベースのシステムには，計算資源提供者がインストール作業を必要としないという利点があったが，継続して資源提供をするには常時WWWブラウザを動作させて指定のページを開いておかなければならない．これに対しNinfletシステムは，いったんインストールがすめば，常時バックグランドプロセスとして動作し，計算機所有者を煩わせることがない．したがって，個人の計算機だけでなく，ワークステーションあるいはPCクラスタ上に仮想並列計算機を構築するためにも利用することができる．

Javaによる他の分散・並列計算システム

他にも，不特定の者のためのメタコンピューティングシステムではないが，Javaによる分散・並列処理をサポートするシステムがいくつか提案されている．JPVM[22]は，PVM風のメッセージパッシングインタフェースを実現するJavaのクラスライブラリであり，PVMとは互換性がない．JavaPVM[23]は，Java で書かれたPVMへのインタフェースであり，PVM と互換性がある．ただしnative methodを使用して実装されている．JAVAR[24),25)]，JAVAB[26]はループなどに内在する並列性を，Javaのスレッドを用いた並列プログラムに変換する．JAVARはソースコードレベル変換するプリプロセッサで，JAVABはバイトコードからバイトコードへ変換する．

ARMI[27]はRMIにおいて非同期メソッド呼び出しを実現する．非同期メソッド呼び出しはHORB[28]でも提供されているが，HORBでは，1つのリモートリ

ファレンスオブジェクトに対して同時に1つしか非同期メソッド呼び出しをすることができない．これに対し，ARMIでは呼び出しごとにfutureオブジェクトを返すことにより，同時に複数の呼び出しが可能となっている．

Javaによるメタコンピューティングの将来

科学技術計算のためにJavaを利用することは果たして妥当だろうか? 現状ではJava VMの実行速度はこのために十分なものとはいえない．にもかかわらず，Javaをそのような目的で利用する気運は高まっており，専門のワークショップもいくつか開催されている[29),30)]．

たしかに，Javaを用いる理由がセキュリティやプログラミングの容易さだけにあるのなら，別の専用言語を設計するのもよいだろう．しかし，普及度，今後の発展速度，プログラマの習熟度，関心度などを考えると，新たな言語を興すよりJavaを活用する方が現実的と考えられる．実行速度の問題は，HotSpotなどの先進的なJITコンパイラや，FORTRANなどにおける既存の最適化技術の適用によって改善されるだろう．ただし，Java固有の問題も多く存在する．たとえば，キャッシュを意識したブロッキングやプリロード系のソースレベルでの最適化に効果が出ない場合があることが我々の実験によって確認されている．また，UMA-SMP，NUMA-SMP，MPPなどの環境でいかにして最適化するかといった課題もある．

今後，Javaによるメタコンピューティングを実用的なものとするには，そのアーキテクチャデザインの研究を進めると同時に，数値計算に特化したJITコンパイラや，数値計算に向いたJava言語の拡張およびAPIの拡張の研究を進める必要がある．我々も，自己反映計算によりこれらの問題を解決するOpenJITシステム[31)]の提案・研究開発を始めている．

参考文献
1) The RSA Data Security Secret-Key Challenge, http://www.rsa.com/rsalabs/97challenge/
2) DISTRIBUTED.NET -- Fastest Computer on Earth, http://www.distributed.net/
3) Catlett, C. and Smarr, L.: Metacomputing, Communications of the ACM, 35, pp.44-52 (1992).
4) DeFanti, T., Foster, I., Papka, M., Stevens, R. and Kuhfuss, T.: Overview of the I-WAY: Wide Area Visual Supercomputing, International Journal of Supercomputer Applications, 10 (1996).
5) Stevens, R.: The I-WAY and the Future of Distributed Supercomputing, http://www.crpc.rice.edu/CRPC/newsletters/fal95/director.html
6) RC5 Java Breaking Effort, http://rc5.mit.edu/
7) RC5 and Java, http://www.hewgill.com/rc5/
8) Sunderam, V. S.: PVM: A Framework for Parallel Distributed Computing, Technical Report ORNL/TM-11375d, Department of Math and Computer Science, Emory University (1990).
9) MPI: A Message-Passing Interface Standard, The International Journal of Supercomputer Applications and High Performance Computing Systems (1994).
10) Lizkow, M., Livny, M. and Mutka, M. W.: Condor - A Hunter of Idle Workstations, Proceedings of the 8th International Conference of Distributed Computing Systems (1988). http://www.cs.wisc.edu/condor/
11) Anderson, T. E., Culler, D. E. and Patterson, D.: A Case for NOW (Network of Workstations), IEEE Micro, Vol.15 (1) (Feb. 1995).
12) 中田, 高木, 松岡, 長嶋, 佐藤, 関口: Ninfによる広域分散並列計算, 並列処理シンポジウムJSPP'97論文集, pp.281-288 (1997). http://ninf.etl.go.jp/
13) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, In Proceedings of Supercomputing'96 (1996). http://www.cs.utk.edu/netsolve/
14) Cappello, P., Christiansen, B., Ionescu, M. F., Neary, M. O., Schauser, K. E. and Wu, D.: Javelin: Internet-Based Parallel Computing Using Java, Concurrency: Practice and Experience (Nov. 1997).
15) Baratloo, A., Karaul, M., Kedem, Z. and Wyckoff, P.: Charlotte: Metacomputing on the Web, In Proc. of the 9th International Conference on Parallel and Distributed Computing Systems (Sep. 1996). http://www.cs.nyu.edu/milan/charlotte/
16) Silva, L. M., Pedroso, H. and Silva, J. G.: The Design of JET: A Java Library for Embarrassingly Parallel Applications, Proceedings WOTUG'20－Parallel Programming and Java Conference, IOS Press, pp.210-228 (1997).
17) Gray, P. A. and Sunderam, V. S.: The IceT Environment for Parallel and Distributed Computing, In Proc. of 1997 International Scientific Computing in Object-Oriented Parallel Environments Conference (1997).
18) Camiel, N., London, S., Nisan, N. and Regev, O.: The POPCORN Project: Distributed Computation over the Internet in Java, In 6th International World Wide Web Conference (Apr. 1997). http://www.cs.huji.ac.il/~popcorn/
19) Sarmenta, L. F. G., Hirano, S. and Ward, S. A.: Web-Based Volunteer Computing Using Java, In Proc. of the 2nd International Conference on Worldwide Computing and its Applications (Mar. 1998). http://www.cag.lcs.mit.edu/bayanihan/
20) 福森, 浜中, 菅原, 吉川, 中山: JAVAアプレットによる分散処理の実現, Javaに関する技術・応用・表現大賞'97 (Aug. 1997). http://www.java-fj.or.jp/event/grandprix/97/
21) Takagi, H., Matsuoka, S., Nakada, H., Sekiguchi, S., Satoh, M. and Nagashima, U.: Ninflet: a Migratable Parallel Objects Framework using Java, ACM 1998 Workshop on Java for High-Performance Network Computing (Feb. 1998).
22) Ferrari, A.: JPVM－The Java Parallel Virtual Machine, http://www.cs.virginia.edu/~ajf2j/jpvm.html
23) JavaPVM－The Java to PVM Interface, http://www.isye.gatech.edu/chmsr/JavaPVM/
24) Bik, A. J. C. and Gannon, D. B.: Automatically Exploiting Implicit Parallelism in Java, Concurrency, Practice and Experience, Vol.9 (6), pp.579-619 (1997).
25) JAVAR－A Prototype Restructuring Compiler for Java, http://www.extreme.indiana.edu/~ajcbik/JAVAR/index.html
26) JAVAB－A Prototype Bytecode Parallelization Tool, http://www.extreme.indiana.edu/~ajcbik/JAVAB/index.html
27) Raje, R., Williams, J. and Boyles, M.: An Asynchronous Remote Method Invocation (ARMI) Mechanism for Java, Concurrency: Practice and Experience, Vol.9 (11), pp.1207-1211 (1997). http://klingon.cs.iupui.edu/~rraje/html/armi.html
28) Hirano, S.: HORB: Distributed Execution of Java Programs, Worldwide Computing and Its Applications, Springer Lecture Notes in Computer Science 1274, pp.29-42 (1997). http://www.horb.org/
29) Java for Scientific Computing, http://www.npac.syr.edu/projects/javaforcse/
30) ACM Workshop on Java for High-Performance Network Computing, http://www.cs.ucsb.edu/conferences/java98/
31) 情報処理振興事業協会（IPA）高度情報化支援ソフトウェア育成事業, 自己反映計算に基づくJava言語用の開放型Just-In-Timeコンパイラ OpenJITの研究開発, http://matsulab.is.titech.ac.jp/OpenJIT/

（平成10年2月26日受付）

特集

5. JavaBeansとコンポーネントウェア

山口　浩
日本サン・マイクロシステムズ（株）

はじめに

ソフトウェアを通常の工業製品と同様，いくつかの部品を組み合わせることで構築することはソフトウェア工学における大きな課題であった．これに対する試みは，古くはFortranで使用する科学技術計算ライブラリのようなサブルーチンの集合として現れたが，その後オブジェクト指向プログラミングが登場した1980年代にはさまざまなクラスライブラリという形になり，さらにウィンドウ環境が一般的になった1990年頃からは，GUIビルダーと呼ばれる開発ツールで使用される部品☆として現れるようになった．これらの部品は，当初は個々のGUI ビルダーに依存した設計であったが，しだいに部品間のインタフェースが標準化されるようになり，いくつかの業界標準も登場するようになってきた．コンポーネントウェア☆2はこのようなソフトウェア部品を中心とした部品の組み立てによって，ソフトウェアを構築する技術を指す言葉として用いられている[1]．一方，Java言語は1995年に発表されて以来，その最大の特徴であるプラットフォーム非依存性によりさまざまな分野で注目を浴びてきたが，特にソフトウェア部品をJava によって作成するというアイデアは，JavaBeansという形で急速に実現されることとなった．本稿では，JavaBeansを用いたアプリケーション開発とその背景にある技術を紹介するとともに，現在開発作業が行われている新しい機能についても概説する．

JavaBeansを用いたアプリケーション開発

JavaBeansの仕様書[2] では，JavaBeans（Beans）を「再利用可能なソフトウェア・コンポーネントでありビルダーツールによって視覚的に操作できるもの」と定義している．これはJavaBeansがGUIビルダーのようなビルダーツールを利用してアプリケーションの作成を行うユーザを対象としたソフトウェア部品であり，利用にあたってはJavaのプログラミング言語としての知識は仮定しないことを前提としている．また，JavaBeans はビルダーツールによって視覚的に操作できる必要はあるが，それ自体は必ずしも目に見える部品でなくてもかまわない点にも注意しておく．上述の仕様書では，JavaのプログラムがJavaBeansであるための条件を，実装すべきクラス，メソッド，あるいは最終的な部品形態としてのパッケージング形式等について定義しており，これに従って作成されたJavaBeansはJavaをサポートするすべてのプラットフォーム上でビルダーツールを用いて自由に組み合わせることが可能となる．

すでに，各社からさまざまな JavaBeans対応のビルダーツールが登場しているが☆3，**図-1**に米国サン・マイクロシステムズが開発したビルダーツール JavaStudio[8] を例として示す．

JavaStudioのメインウィンドウには個々の部品（Beans）に対応したアイコンがいくつかのカテゴリーに分類されて並んでおり，ユーザは必要な部品をここからGUIウィンドウにマウスを使ったドラッグアンドドロップで配置することによりアプリケーションを作成する．この時，デザインウィンドウには，これらの部品に対応するアイコンが表示されており，マウス操作でそれらの間にリンクを張ることで部品間に処理の流れ等の論理的な関係を定義することができる．

あらかじめ用意されている部品としては，テキストを表示するラベル，入力フィールド，ボタンといったGUIの基本部品に加え，マルチメディア関連やデータベース操作を行うもの，あるいはループや条件判断，データの分岐を行うような「制御」のための部品など，約50種類がある．さらに，ユーザが作成したり購入したりしたBeansを新たな部品として追加登録することも行えるようになっている．これらの操作は，マウスによる簡単な操作や適当なテキストの入力を行うだけで実行でき，プログラミング言語に関する知識はほとんど必要としない．実際にJavaStudioを用いてアプリケーションの作成を行ってみると，それが電子機器の回路図を書き上げる作業に類似していることに気付くであろう．しかもこうして作成したプログラムは，Javaアプレットとしてさまざまな OS上のWWWブラウザで動かしたり，あるいはスタンドアローン型のアプリケーションとして，さまざまなJava環境で動作させたりすることができる．さらにこれを新しいBeanとして部品化することも可能である．このような特徴は従来のGUIビルダーには見られないものであり，JavaBeansの持つ大きな特徴でもある．

☆ ボタンやラベル等の部品でウィジェットあるいはガジェットと呼ばれることもある．

☆2 コンポーネント・ソフトウェアとも呼ばれる．ほかにIBM社，Apple Computer社によるOpenDoc やMicrosoft社によるOLE/COMなどが知られている．

☆3 JavaBeans対応の開発ツールについては，http://java.sun.com/beans/tools.htmlに一覧がある．

JavaBeansで用いられるコンポーネントウェアとしての基盤技術

JavaBeans を特徴づける重要な属性は「プロパティ」,「メソッド」,「イベント」である.「プロパティ」はBeansの公開された変数であり,適当なメソッドを呼び出すことで,参照や変更が可能な属性である.「メソッド」は通常のJavaのパブリックなメソッドであるが,Beansとして使用するものをこれらのメソッドの一部に限定することも可能である.最後に「イベント」はBeans間の通信に用いられる仕掛けであり,1つのコンポーネントで,ある事象が発生したことを,他のコンポーネントに知らせるための通信手段として提供されている.事象を発生するイベントソースに,その事象に関心を持つコンポーネントがリスナーオブジェクトを登録しておき,事象の発生時にそのオブジェクトの適当なメソッドが呼び出される仕組み(デリゲーションイベントモデル)によって実現されている.

さて,ビルダーツールは個々のBeansが持つさまざまな属性を知るために「イントロスペクション」と呼ばれる機能を用いる.これはJavaのCore Reflectionを用いた機能で,Javaのクラスファイルに定義されている変数やメソッドのシグネチャを調べたり,あるいは指定したメソッドを適当な引数とともに実行したりする一種のメタプログラミングを行う強力な仕掛けである.さらに,ユーザがこれらのBeansの属性を変更するためのインタフェースはCustomizer APIとして用意されているが,Beansの提供者はビルダーツールのためにカスタマイザを書くことで,カスタマイズのための優れたユーザインタフェースを実現することができる.一般にカスタマイズを行った結果は,あるクラスのインスタンスとなるが,これをそのまま保存する仕組みも提供されている.これにはオブジェクトのシリアリゼーションと呼ばれる機能が使われている.これらは,いずれもJDK1.1で導入された一般的な機能であるが,JavaBeansの誕生にはこれらのさまざまな機能をうまく組み合わせて利用することが必須であったことは容易に想像されるであろう.

JavaBeansの新しい機能

現在のJavaBeansの仕様に基づくBeans開発ツールやビルダーツールは,ほとんどが単体で動くJavaの世界に閉じたアプリケーションのためのものであるが,他のコンポーネントウェアとの接続やBeans間のより高いレベルの結合,あるいは分散環境への対応といった多くの課題が残されており,現在もさまざまな機能拡張のための作業が進行中である.これらのうちから主要なものを**表-1**に紹介する.今年の夏頃にリリースが予定されているJDK 1.2では,GUI機能の拡張やサーバ環境での利用等を踏まえた機能拡張が行われるが,表-1のうちのいくつかは基本機能として盛り込まれる予定である.JavaBeansの大きな目的の1つは,企業内システムで用いられるような複数のプラットフォームで動作する大規模なアプリケーション開発をBeansベースで行うことであるが,これらの機能を盛り込むことで,それが現実のものとなる日もそう遠くはないと考えている.

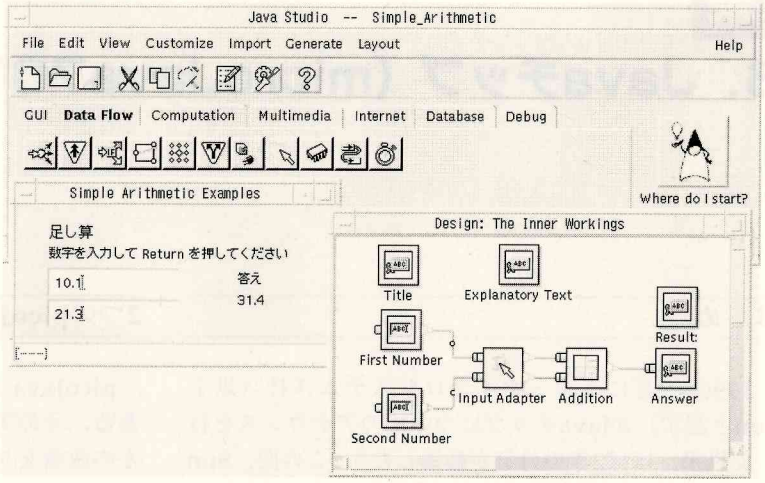


図-1 JavaStudioによるアプリケーション開発

表-1 JavaBeans 関連の拡張機能

新しい仕様	内容
Enterprise JavaBeans[3]	C/S システムにおいてサーバ側で動作するJava プログラムを作成するために拡張された JavaBeans 仕様
Drag and Drop[4]	Java アプリケーション間および通常のアプリケーションとのドラッグアンドドロップ操作
Activation Framework[5]	データソースの型や処理方式の決定方法の一般化
Runtime Containment[6]	JavaBeans が自分自身を含むコンテナに関する情報を実行時に得るためのインタフェース
InfoBus[7]	コンテナ内の複数のJavaBeans が相互にデータ交換を行うためバスアーキテクチャ

参考文献
1) 青山幹雄: コンポーネントウェアの挑戦, bit, Vol.28 (Mar., Apr. 1996).
2) JavaSoft: JavaBeans 1.01 (July 1997).
3) JavaSoft: Enterprise JavaBeans Version 0.9 (Feb. 1998).
4) JavaSoft: Proposal for a Drag and Drop subsystem for the Java Foundation Classes (Dec. 1997).
5) JavaSoft: JavaBeans Activation Framework Specification (Dec. 1997).
6) JavaSoft: A Draft Proposal to define an Extensibile Runtime Containment and Services Protocol for JavaBeans (Version 0.98) (Jan. 1998).
7) JavaSoft: InfoBus 1.1 Specification, Draft 6 (Dec. 1997).
8) SunSoft: JavaStudio 1.0, http://www.sun.co.jp/workshop/js/
文献2) ~7) は米国サン・マイクロシステムズのJavaホームページ (http://java.sun.com) からダウンロード可能である.

(平成10年2月27日受付)

特集

6. Javaチップ（microJava701）の高速化技術

藤田光章
日本サン・マイクロシステムズ（株）

はじめに

1996年2月にサン・マイクロシステムズ社（以下Sunと記す）がJavaチップについてのアナウンスを行って以来，ほぼ2年の月日が経過した☆．この間，SunからはpicoJava-Iを数社に対してライセンスし，またSun自身もpicoJavaのテクノロジーをベースとしたJavaチップmicroJava701の開発を進めてきた．

コアライセンスというリリース形態をとるpicoJava-Iは，Javaバイトコードを逐次解釈あるいはコンパイルすることなしに直接実行できるCPUコアであり，実体はデザイン情報である．実際の物理的なピンを持ったチップ（ICパッケージ）ではない．そしてこのCPUコアを使って作られたASIC (Application Specific Integrated Circuit)がJavaチップである．

同じCPUコアをベースとしたJavaチップでも，CPUコア以外の部分はその目的などに応じてさまざまな機能回路が付加されて1つのASIC（Javaチップ）となる．microJava701もそうしたJavaチップの1つとしてSunが開発中のものである．

SunはpicoJavaのテクノロジーについてはこれまでにいろいろな場所で資料やデータを公表してきた．昨年7月の東京ビッグサイトにて行われたJava Computing ExpoでのJava Processorのチュートリアルや，昨年10月に米国西海岸で行われたMicro Processor Forumなどである．それらに加えて最近は商業誌にも紹介記事[1),2)]が掲載されてきている．picoJavaコアあるいはJavaチップの基本的説明についてはそれらを参照していただくとして，本文ではmicroJava701で新たに開発されたCPUコアの高速化技術について解説する．

☆ Javaチップ，JavaOSなどの位置付けについては，http://java.sun.com/docs/white/platform/JavaPlatform.doc1.htmlにある（編著）．

2つのpicoJavaコア

picoJava-IはSunが現在ライセンスしているコアである．その実行性能をさらにアップするためにいくつかの改善を加えて最近開発したのがpicoJava-IIコアである．microJava701はこのpicoJava-IIをCPUコアとしてインプリメントしている．2つのコアのアーキテクチャ上の相違点の主なものを**表-1**にまとめた．

性能向上の要点

microJava701のコアであるpicoJava-IIでは，より高いクロック周波数で動作させて全体の性能向上をはかるために，パイプラインの段数を4段から6段に増やしている．これによりmicroJavaでは0.25ミクロンのCMOSプロセスを適用して200MHzで動作させることを設計目標として開発が進められている．またこれとあわせて行ったのが命令フォールディング機能の強化である．

picoJavaはスタック構造のアーキテクチャを持っており，バイトコードの直接実行においてはスタック最上部へのデータの移動（コピー）とスタック最上部での演算という命令のシーケンスが頻繁に発生する．これに対し，picoJava-Iでは命令フォールディング機能をそなえ，性能向上を図っている．従来のRISC，CISC型CPUとの処理速度比較においては，特に従来型CPUが1クロックサイクルで実行する汎用レジスタ間の演算と等価な動作が，スタック構造ではいくつかの動作の組合せとなるため性能面で不利であった．picoJava-IIではこの点を考慮し，picoJava-Iの命令フォールディング機能をさらに強化して，RISCマシンなどにおける汎用レジスタ間の演算と等価な動作をさせる場合の性能も大幅に改善している．

この性能改善は，C/C++からJavaへの移行過程と

表-1　picoJava-IとpicoJava-IIの相違点

	picoJava-I	picoJava-II
パイプライン段数	4	6
I-Cache Line Size	8 bytes	16 bytes
Instruction Buffer	12 byte deep	16 byte deep
I-Cache to I-Buffer	4 bytes	8 bytes
フォールディング	2命令まで	4命令まで

RISCマシン	バイト・コード	picoJava-IIのフォールディング
ADD R2,R1,#10	iload_1 bipush 10 iadd istore_2	iload_1+bipush 10+iadd+istore_2 （1クロック・サイクルで実行）
SUB R3,R1,R2	iload_1 iload_2 isub istore_3	iload_1+iload_2+isub+istore_3 （1クロック・サイクルで実行）

図-1　picoJava-IIの命令フォールディング

して，従来型CPU向けにC言語で書かれた実績のあるソフトウェアをバイトコードにコンパイルしてJavaチップ上で実行する場合に，従来プロセッサで得られていた性能と同等，あるいはそれ以上を実現する上できわめて重要である．

picoJava-IIにおける命令フォールディング機能の強化

従来型CPUのように汎用レジスタ（レジスタ・ファイル）を演算回路の対象として構成したマシンでは，演算対象のデータ（変数）をなるべく多く汎用レジスタに割り当てて高速なレジスタ間の演算をフルに活用することにより，処理の高速化を図るのが普通である．スタック構造のマシンでも演算の対象となる変数はスタック内に配置している．従来型CPUのレジスタ間演算と等価な演算をスタック構造のマシンで行う場合は，演算対象のデータを演算に先立ってスタック最上部にコピーし，演算の結果スタック最上部に残されたデータをまた変数に書き戻すというプロセスが必要となり，変数間の演算の前後にいくつものコピー動作のための命令を伴うので処理速度が遅くなる．picoJava-Iではもともとスタック最上部へのデータ移動とそれに続く演算に関する2つの命令を1クロック・サイクルで実行できるような命令フォールディング機能をそなえているが，RISCマシンなどで1クロック・サイクルで演算されるレジスタ間演算命令はバイトコードでは**図-1**のように4命令くらいに分解され，RISCマシンのレジスタ間演算と等価な処理においてRISCマシンと同等の性能を実現するには2命令のフォールディングでは不足する．picoJava-IIではこの点を考慮し，最大4命令までフォールディングできるように改善した．

図-2に示すように，picoJava-IIのフォールディングユニットでは，常に命令バッファ中の7バイトを分析し，命令の組合せ可能なものを最大4命令までフォールディングしている．

たとえば，A，B，Cをそれぞれスタック上のLocal Variableとした場合，次のようなバイトコードのシーケンスが7バイトのフォールデング可能な例である．

```
iload  <A>        2バイト命令
iload  <B>        2バイト命令
iadd              1バイト命令
istore <C>        2バイト命令
```

フォールディング・ユニットはバイトコードのシーケンス分析の結果を2つのオペランドアドレス（RS1，RS2）と演算コード（OPC），および演算結果の書き戻しアドレス（RD）の4つの情報に整理してレジスタ制御部に渡す．レジスタ制御部では与えられたRS1，RS2によりスタックキャッシュにアクセスして2つのオペランドデータ（OP1，OP2）を実行ユニットに渡している．この機能強化を実現するためには，5バイトを分析して2命令までのフォールディングを行っているpicoJava-Iに比べ，picoJava-IIでは命令デコードの部分で1クロック・サイクルあたり，より多くの命令をデコード回路に供給する必要があり，そのためにはメモリから命令キャッシュ，および命令キャッシュから命令バッファへのそれぞれのデータ転送能力を上げ，命令バッファの容量を増やすとともにレジスタ制御部におけるスタックへのアドレッシングを強化した．表-1に示したpicoJava-IとIIのアーキテクチャ上の相違点の多くはこのフォールディング機能の強化のためともいえる．

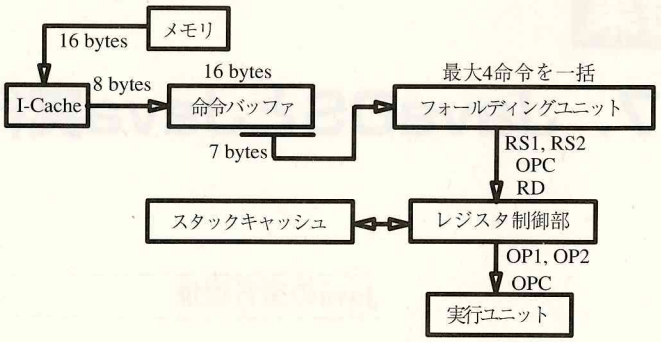


図-2　命令フォールディングのデータバス

表-2　Dhrystone MIPS2.1

PROCESSOR	MHz	MIPS	MIPS/MHz
Motorola PPC602	66	65	0.98
Hitachi SH7708	100	100	1.00
Sun microJava 701	200	200	1.00
Digital SA110	200	230	1.15
Sun microSPARCIIep	100	118	1.18
NEC R4300	133	160	1.20

性能評価

以上に述べた高速化技術をインプリメントした結果，microJava701の性能評価（予測値）ではJava CodeについてはCaffeineMark 3.0（embedded）：13,332，C CodeについてはDhrystone MIPS:200 @200MHzであった．

表-2はDhrystoneに関する他CPUとの比較であるが，RISCマシンとほぼ同等の性能が得られている．これらの性能予測値より，Java Codeの実行性能の高さはいうまでもないが，RISCマシンに比べてスタック構造のCPUは性能が出ないといわれてきた弱点も以上に述べた高速化技術により克服している．今後microJava701がチップとして完成して実チップ上での性能実測値が得られるようになれば性能測定プログラムのみならず多くのアプリケーションを実行することにより以上に述べた高速化技術の有効性が実証されるであろう．

参考文献
1）窪田和弘：Javaチップについて，bit, 97年11月号．
2）米Sun社のmicroJava701，バイトコードを高速実行，日経エレクトロニクス，1998.1.12 (no. 707).

（平成10年2月25日受付）

特集

7. JavaOSとJava実行環境

井田昌之
青山学院大学

Javaの実行環境

Javaで作成されたオブジェクトプログラムは，Java仮想機械のコードの形をとる．これをJava byte codeと呼ぶ．Java byte codeの実行は，**図-1**（1）にあるように任意のOS上でインタプリタを作動させる方式（たとえば多くのウェブブラウザの利用の場合）が基本的なものであった．これは性能的にはあまり高いものではない．そこで，いくつかの性能向上の手法が開発されている．

Javaのそもそもの発端であるセットトップボックスあるいは情報家電といった実行環境を考える場合には，汎用のOSは必ずしも必要がなく，**図-1**（2）のようにJavaコードの実行を目的としたJavaOSを置くことが提案されている．また，Javaの実行だけを目的とするネットワークコンピュータにおいても同じ形態が志向できる．

汎用のコンピュータ上の汎用のOSのもとでの実行を高速化させるには，byte codeに対してコンパイラを実行させて，その機械の機械語に変換してから実行する**図-1**（3）に示す方式が有効と考えられている．実行の直前にこれを行なうJust In Timeコンパイラと，前もって実行させておくスタティックコンパイラの2つにさらに分かれる．

Java byte codeの実行に対する究極的な方式は，byte codeを直接実行するコンピュータを用意することである．**図-1**（4）がそれであり，いわゆるJavaChipを用意し，その上にJavaOSを搭載する．もし，その環境におけるすべてのアプリケーションがJava環境ですむのであれば，この方式は最適な選択となり得る．一方，汎用性／拡張性などの点で，今後の推移を見守る必要がある．

以下に，JavaOSについて解説する．

JavaOSは何のため？

Java OSは，Javaアプリケーションの実行のためのOSである．他の言語で書かれたアプリケーションについては考えられていない．イントラネット，インターネットおよび組込み装置を意識して設計されている．また，JavaOSは，現在までのところ，JavaChipのためのOSに限定したものではない，複数のハードウェアプラットフォームをターゲットとしている．JavaOSは，こうした環境の中で，クライアント管理の労力を軽減できるOSとして考えられている．

JavaOSは，小さく効率のよいことが意図されており，4MBのROMと4MBのRAMを最小実行環境として設定している．普通のパソコンが32MBあるいは64MBの主記憶を実装する時代に，この小ささの強調は，「制限されたリソース環境」を意識したものである．

また，JavaOSは，いわゆるネットワークコンピュータのOSとして注目を浴びた．いったんネットワークに接続されれば，さまざまな機能を発揮するように意図されている．さらにその場合，HotJavaブラウザによるウェブへの対応，HotJava ViewsによるGUI環境がベースとして用意される．

JavaOSの概要は，http://java.sun.

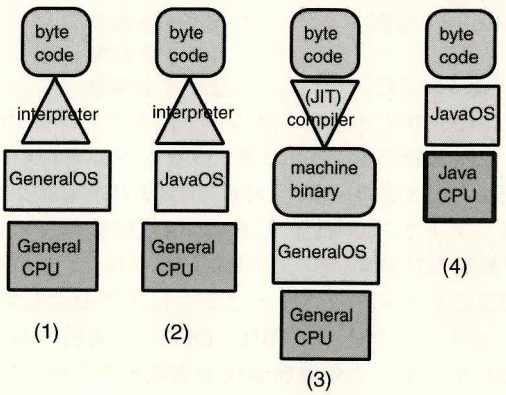


図-1　Javaの実行環境4態

com/marketing/collateral/os.htmlおよびhttp://java.sun.com/products/javaos/index.htmlなどから知ることができる．

JavaOSを搭載する機器は低価格で管理のいらないステーション（Zero Administration Client）というのが特徴といわれてきた．しかし，普通のデスクトップを必要とする環境では，それらが今までにもっていたさまざまなソフトウェア環境の移植あるいは互換性の保証が必要となる．ハードウェアリソースへの対応も重要になる．このため，JavaOSを搭載したコンピュータは汎用のコンピュータというよりも，専用の機器としての場合の方が，より有効性を発揮できるものと考えられる．

このような流れは自然に，組込み機器用のOSという色彩を強く持つものへと向かうことになる．いわゆる情報家電への期待である（図-2）．

JavaOSのアーキテクチャ

JavaOSはレイヤードアーキテクチャによっている．大きくはプラットフォーム依存部とJavaで書かれたプラットフォーム独立部の2層からなる．

(1) **プラットフォーム依存部**：プラットフォーム依存部は，その機械の機械語でコードされる．マイクロカーネルとJVMがその中にある．

マイクロカーネルには，ブート機能，割り込み処理，多重スレッド機能，トラップ，DMA機能がある．Sun Microsystemsからのライセンスによりリリースされている現在の版では，SPARC，x86，StrongARMがサポートされている．なお，これらの開発には，GCC 2.7.2をベースにした開発環境が想定されている．JVMは，Java bytecodeのインタプリタループと実行だが，このためにメモリ管理，スレッドの制御，クラスローディング，およびbytecode verifierがその中に含まれている．

(2) **プラットフォーム独立部**：プラットフォーム独立部には，以下のような機能が入れられている．

JavaOS Window system，JavaOS Graphics system，JavaOS device drivers，JavaOS Network Classes

JavaOS Window systemは，ボタン，メニュー，スクロールバーなどGUIコンポネントを供給する．また，オーバーラップウィンドウに対応している．

JavaOS Graphics systemは描画パッケージを供給する．

JavaOS device drivers（デバイスドライバ）はJavaで書かれ，移植性があり，拡張性がある．

Network Classesは，TCP/IPをサポートし，ネットワークトランスポートおよびルーティング機能を供給する．DNSおよびNIS機能があり，ホスト名参照，ユーザなどに共通した機構を用いることができる．RARP，DHCP，NFS client，SNMPが組み込まれている．SSLサポート，PPP，HCP起動時のベンダオプションデモ用のセットアップなどがあり，ネットワークに関連する機能がいろいろと具備されている．その詳細は省略する．

JavaOSは，マルチプログラミングではないが，マルチスレッドに対応している．つまり，アプリケーション実行中にネットワークへの対応をしたりすることはできる．

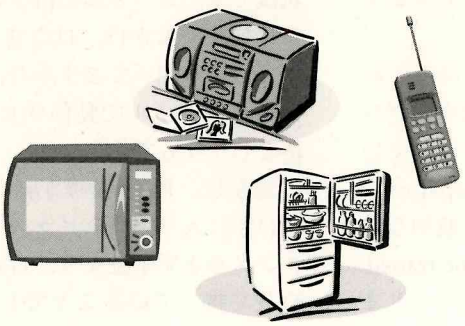


図-2　情報家電への期待

JavaOSの起動

JavaOSは，ROM，ネットワーク，そして，ハードディスクなどから起動できる．ネットワークコンピュータとしての利用の場合には，おそらくはネット上のサーバからシステムをダウンロードして起動することが考慮されるだろうし，組込み機器・ハンドヘルドの場合にはROMから立ち上がることが主に考慮されるだろう．

JavaOSでのJava言語仕様

JavaOSは，Javaの言語仕様の水準を特定しない．JavaOSで動作するアプリケーションは，Javaが動作できるブラウザ・OSでも動作する．けれども，Java Application Environment（JAE）という概念がある．たとえば，対象とする実行環境がEmbedded JavaあるいはPersonal Javaの範囲だとすると，グラフィック関連のサポートがなくなり，それらを要求するプログラムは動かなくなってしまう．どの範囲のクラスライブラリが必要なのかによって，実行環境の水準分けをする仕組みの必要性が出てくる．これがJAEである．

Embedded Java/Personal Javaという仕様水準の設定は，JavaOSの開発開始のあとから出てきた概念である．つまり，小さな応用には小さな機能しかいらないということを明確化できる領域があるということであり，歴史的な理解としてうなづけるものがある．しかし，仕様と対象領域の対応がはっきりとしたものとなるかは，今後の実用化により答えが出る．

（平成10年3月9日受付）

コラム　▼▽インターネット△▲

標準化と実装研究

楠本博之／慶應義塾大学

会誌のリニューアルにともない，石田編集長からコラムを連載してくれないかと頼まれました．「インターネット」に関する記事や読みものは，巷の雑誌にあふれており，わざわざ「情報処理」のような会誌でとりあげるからには，同じような内容では，おもしろくありません．かといって，あまり会誌を意識して，従来のスタイルを踏襲すると，リニューアルした意味がありませんので，執筆者にとってなかなか難しい連載になりそうです．1年間の予定ですが，どうぞおつき合いください．

さて，話題にことかかないインターネットですが，今回は，最近の研究関連の話題です．

みなさんもよくご存知のように，インターネットは急激に規模が拡大し，ホスト数とネットワークが増加し，ネットワークの経路制御情報の増大とネットワークアドレス不足が生じています．ホストを識別するためのアドレスは32ビット分の空間しかありません．また，インターネット上を流れるデータグラムをどちらの方向に転送するかを判断するための経路制御のための表も大きくなりすぎて，転送を行うルータの負荷が増大しています．

そして，移動計算機環境の支援やホストやネットワークの自動構成，セキュリティ機能といった，これまで研究が進められてきたプロトコルもうまく採り入れて抜本的に問題解決をはかるため，次世代インターネットプロトコルの研究開発が数年前から進められ，佳境に入っています．現在一般的に使われているインターネットプロトコル（IP）のバージョン番号が4であるのに対して，新しいプロトコルはバージョン番号が6になるので，IPv6と呼ばれます[1]．

IPv6の研究開発には，世界中の数多くの研究者や技術者がかかわっており，日本の研究者も深くかかわっています[2),3)]．6BONEと呼ばれるIPv6を使った研究開発のテストベッドバックボーンの運用も軌道にのっており，世界中で29カ国の組織がすでに接続しています．日本ではWIDEのグループが中心となって運用実験を行っています．IPv4の経路情報の爆発への反省から，集約可能アドレス（aggregatable address）構造をとれるようになっています．IPv6アドレス空間は，128ビットですが，aggregatable global unicast addressは，図のようなアドレス形式になっています．

FP（Format Prefix）は，この集約可能アドレス型に固有のプリフィックスで，001に決まっています．TLA ID（Top-Level Aggregation ID）が，一般的なトラフィックの通過（public transit）を提供する組織に割り当てられます．6BONEのために，上位16ビットが3ffeが実験的に割り当てられています．6BONEの中ではアドレス割り当ての登録などの実験をするために，NLA ID（Next-Level Aggregation ID）の上位8ビットを使ってpTLA（pseudo TLA）を決め，pTLA＝5がWIDEに割り当てられています．現在，6BONE上では経路制御プロトコルやIPv4とIPv6の相互乗り入れなどの実運用へ向けての重要な実験も進められており，移行へ向けての体制作りが行われています．

このように，新しいインターネットプロトコルのホストおよびルータ上への実装が進められているわけですが，IPのような基盤となるプロトコルが新しくなるのは，10年，20年に1回のことですから，今，IPv6の研究をする機会を得た人は幸運です．ひょっとすると自分のアイディアがこれから10数年もの間，世界中の人たちに使われるかもしれないわけで，いろいろなことに挑戦してほしいと思います．

もう1つの幸運は開発環境です．TCP/IPをネットワークプロトコルとしてBerkeley UNIXに採り入れる研究が進められた頃を考えると研究環境には非常に大きな差があります．オペレーティングシステムにかかわる研究とそのソースコードの利用可能性は，切り離せないものですが，当時はBSDのシステムが動く高価ミニコンピュータシステムが，ある意味で標準機でした．大学で導入できるところはかぎられており，あっても学科に1台，よくて研究室に1台という状況です．カーネルの再コンパイルに何時間もかけ，はたまたTSSで多人数で使ってますから，頻繁に違うカーネルで動作させたり，落としたりもできません．

現在は，PC互換機が非常に安価に手に入り，コンピュータサイエンスを志す学生なら，1台以上，自分で持っていることでしょう．自分のマシンでOSやネットワー