クライアントサーバシステムにおけるJavaの適用

Application of Java in Client/server System

宮　脇　　亨

要　約　Web技術を中心としたビジネスシステムが増えてきている．インターネット上では，企業の広報活動をはじめ，業務システムも稼働し始めている．このようなシステムにおいてJavaをどう適用していくかは，システム開発における重要なポイントである．

本稿では，まず検証モデルを設定し，システムで採用するJavaの技術要素の選択と解説をおこなう．そして，検証モデルの実証評価をもとに，Javaのクライアントサーバシステムへの適用について論じる．

Abstract　The Web technology-based business systems have been rapidly increased, and also the mission-critical systems including the public relations of companies are going to work over the Internet. It is an important matter in the system development how users will apply Java in such a business system.

This paper describes the establishment of an inspection model for the system development first, the selection and commentary of the technology elements of Java to be adopted by a system.

Then it describes the application of Java to the client/server system, depending on the demonstrative evaluation of the proposed inspection model.

1. はじめに

インターネットの普及により，Web技術を使ったビジネスシステムが増加してきている．企業にとっては，Web技術をベースにすることで広範囲の利用者を対象にしたサービスを提供することができる．そのため，ホームページを開設し企業アピールをおこなうといった広報活動から，Webクライアントを使った業務システムにまで適用されるケースが多くなってきている．

このような状況に応じて，システムの形態もWeb技術を中心としたものに変化してきている．たとえば，クライアントサーバ型のシステムでも，一般的にいわれる2階層型のシステムや3階層型のシステムに，Web技術を融合した新しいシステムモデルが登場してきている．

一方，インターネットとともに発展してきた技術としてJavaが注目を浴びている．Javaは，JavaAppletに代表されるように，Webシステムを意識した技術が多いからである．また，最近では，クライアントサイドだけでなく，サーバサイドで使用される技術としても注目を浴びている．これは，Javaのポータビリティや言語仕様の優秀さ（セキュリティ，ネットワークとの親和性など）が，ビジネスロジックを記述する言語としてこそ発揮されるのではないかという期待を示している．したがって，今後，Web技術を中心とした企業システムに，いかにJava技術を適用していくかは，システム開発における重要なポイントとなると考えられる．

本稿では，エンタープライズシステムへのJavaの適用を視野に入れつつ，まず，Javaの基本的な技術が，Web技術を含めた新しいシステムモデルに対してどう適用

されるのかを検証する．そのために，いわゆる 2 階層型のクライアントサーバシステム（以降，2 層モデルと呼ぶ）と Web 技術を融合させた検証モデルを設定し，システムで採用する Java の技術要素の選択と解説をおこなう．そして，検証モデルの実証評価をもとに，Java のクライアントサーバシステムへの適用について論じる．

2. 技術要素の選択

検証モデルとして想定しているシステムは，2 層モデルに Web 技術を加えたものである．

2 層モデルは，物理的にも論理的にも 2 層構造で構成されている．ここに Web 技術を加えて考えると，クライアントとデータベースの間に論理的に中間的な層を設ける必要がでてくる．したがって，Web 技術を加えた検証モデルは，論理的には 3 層構造であり物理的には 2 層構造をもったシステムとする．本稿では，この検証モデルを Web クライアントサーバシステム（以降，Web モデル）と仮に呼ぶことにする（図 1）．

Web モデルの特徴は論理的にシステム構造が分割されていることである．たとえば，図 1 に示す通り，サービス層とデータベース管理層を論理的に分けている．このことはシステムを拡張－すなわち物理構成を 3 層に拡張する際に分割が容易であることを示している．

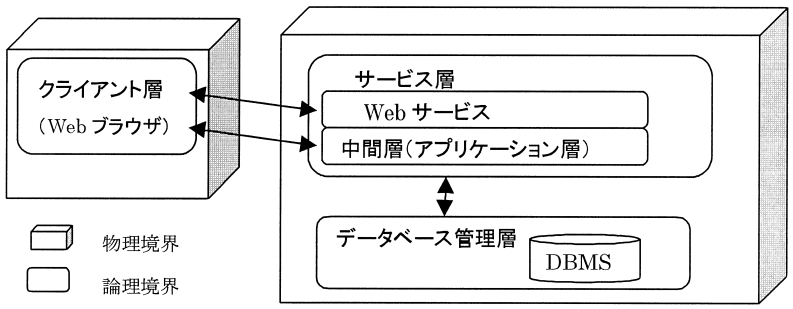

図 1　検証モデルについて

この検証モデルにおいて，まずシステム構成上重要なポイントとなる技術を抽出し，Java でどう実現するかを検討する．検討すべき項目は以下の通りである．

1） Web 技術をどう利用するか
2） データベースへのアクセス方法はどうするか
3） 通信方法はどうするか

技術の選択基準は，どれに重きをおくかで違ってくるが，今回の方針としては，Web 技術の利用方法を優先し，次にデータアクセス方法を決定していく．最後に通信方法について決定する．（実際のシステム開発は，顧客要求などに応じて優先度をつける必要がある）

2.1 Web 技術をどう利用するか

2 層モデルは，クライアント層とデータベース層によって構成されており，すでに

このモデルをターゲットにした開発ツールも多い．この種の開発ツールでは，クライアントとデータベースを直接結びつけて開発していく．しかし，Web 技術を利用していくことを考えると，クライアントプログラムを直接データベースと接続するのではなく中間層を介して接続されることになる．

一方，Web クライアントとして HTML ベースのクライアントを考えると，2 層モデルのクライアントに含まれていたデータアクセス，データチェックなどのロジックを組み込むことには限界があり，必然的に中間層へロジックを移すことになる．

一般的に 2 層モデルのクライアントは Fat　Client と呼ばれているが，これはビジネスロジック相当が GUI 制御とともにクライアントプログラム内に含まれているからである．これに対して，Web ベースで考えるクライアント（Thin　Client）では，ビジネスロジックそのものがクライアントになくサーバ側（Web サーバ）に集中している．そのため，過去にビジネスシステム（特に基幹系のシステム）が実現してきたクライアントに比べると，あまりに貧弱な（機能的に劣っているかのような）クライアントとして映ってしまう．しかし，Web クライアントの重要な特性としては，ブラウザがあればどんなクライアントでも（PC でも UNIX でも…）稼働できる点がある．各クライアントにプログラムを配布しないでも利用可能であることは，管理コストの削減になるだけでなく，不特定多数のユーザを相手にビジネス可能であるということである．

以上のことを踏まえたうえで，Web モデルにおけるクライアントの役割（システム内で果たす役割分担）について考えてみると，Web クライアントの持つべき特性としては，Fat でもない，Thin でもない中間的なものが妥当ではないかと思われる．つまり，Fat　Client に詰め込まれていたロジックのうち，ビジネスロジック相当をサーバで処理するが，GUI 制御や入力チェックなどの最低限実現したい機能はクライアントが実装するということである（図 2）．

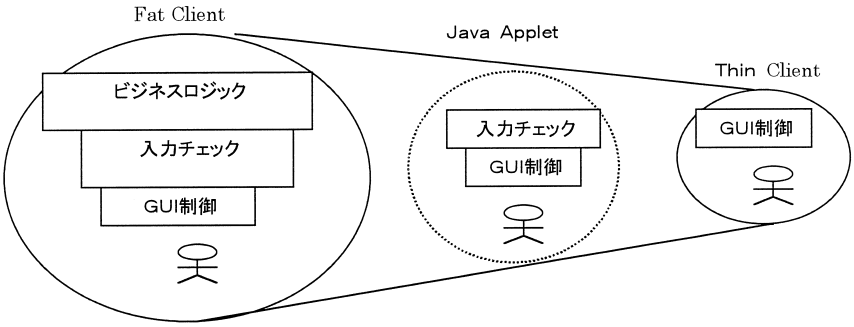

図 2　FAT と THIN の中間に位置する Web クライアントのイメージ

現在，この中間的な Web クライアントとして，JavaApplet は位置づけることが可能であると考える．なぜなら，JavaApplet は，Web サーバよりネットワークを介してプログラムをダウンロード可能であり，ブラウザ上で稼働する．また GUI としての表現力も Swing コンポーネントの利用でビジネスシステムとしての要件を果たす

ことが可能であるからである．

Web ベースのクライアントサーバシステムを考えた場合，この Web クライアントの位置づけは重要であると考える．そして，JavaApplet が Web クライアントとしての特性を満たすことができる技術であることによって，今回の検証モデルでは，まず JavaApplet の利用を決定する．

2.2 データベースアクセスの方法を決定する

次にデータアクセスの方法を考える．

Java からデータベースへアクセスするには，JDBC を経由して行なう．JDBC では，RDBMS（以降，DBMS とする）へのインタフェースが規定されている．したがって，このインタフェースに対してプログラミングすればよい．あとは，これに対応した JDBC ドライバを選択することでデータベースへのアクセスが可能になる．

JDBC ドライバには，表 1 に示す通り 4 種類あるため，それぞれの特徴を考えながら採用するドライバのタイプを決定する必要がある．また，JDBC ドライバの選択は，JavaApplet との関連が強い．JavaApplet は，セキュリティの関係上ローカル資源へのアクセスを許していないからである．（Java 2 ではポリシーの設定で可能であるが信頼できるサイトからロードされるのが前提である）

こういった Web クライアントとデータベースアクセスの関連については，Web モデルにおいて重要なポイントである．今回の Java の検証モデルにおける JavaApplet の採用と JDBC ドライバの選択は，Web モデルにおけるひとつの実現案を示しているといえる．

表 1　JDBC ドライバの種類と特徴

ﾀｲﾌﾟ	分類	仕組み	特徴
Ⅰ	JDBC－ODBC ブリッジ型	JDBC API を ODBC API に変換し、ODBC ドライバを使用してアクセスする。	ODBC 資産を継承できる JDK1.1 より標準添付 ローカル資源をアクセス
Ⅱ	Native-API & partlyJava Driver 型	JDBC API を DBMS 固有のデータベース API に変換し、DBMS 固有の通信によりアクセス	DBMS 固有 ローカル資源アクセス
Ⅲ	Net-Protocol & All-JavaDriver 型	JDBC API を独立したネットプロトコルへ変換し、中間層へアクセス。中間層で DBMS 固有のプロトコルに変換してアクセスする。	中間層は既存のミドルウエアを利用可能。 システムが複雑になる。 クライアント部分が Java で記述されている。（ローカル資源アクセスなし） 提供ベンダ依存
Ⅳ	Native-Protocol & All-JavaDriver 型	JDBC API を DBMS 固有のプロトコルに変換し、直接 DBMS との通信を行なう。アクセスのための機能はすべて Java で書かれている。	シンプルなシステム構成 ダウンロードサイズが大きくなる。 ローカル資源アクセスなし DBMS ベンダ固有

さて，JavaApplet を前提として採用する JDBC ドライバの検討をおこなう．

まず，候補としてタイプ III，IV が上がってくる．これはローカル資源にアクセスしないで済むタイプだからである（JavaApplet の制約）．ただ，ドライバを提供（タ

イプ III の場合は中間層も）するベンダに DBMS が制限されていることを考えると，ターゲットとして考えている DBMS を選択できない可能性もある．ビジネスシステムにおいて DBMS の選択は重要であるので，候補としたタイプ III はいったん保留する．また，タイプ IV はドライバを JavaApplet とともにダウンロードすることから，サイズが大きくなるのでこれも保留する．逆に，タイプ I の場合は，ODBC 経由のため選択できる DBMS が多いのが特徴である．ローカル資源へのアクセス（ODBC ドライバ）は行なう必要はあるものの，タイプ I をタイプ III のように中間層を設ける方法で対応することができる．

今回は検証モデルであることも考えて，DBMS を自由に選択できるタイプ I を使用することにする．すなわち，JavaApplet から Java のサーバアプリケーション（中間層）を経由して DBMS へアクセスする方法である．これなら，ODBC の資産を利用することができ，しかも，Java のサーバアプリケーション層の評価にもつながる．論理的に 3 層構造であり，Web モデルに適合している．

2.3 通信方法はどうするか

JavaApplet から Java のサーバアプリケーション（中間層）までの通信手段が問題である（中間層から DBMS へのアクセスは ODBC ドライバに依存している）．

Java と Java の通信であれば，JavaRMI（Remote Method Invocation）が使用できる．JavaRMI（以降 RMI と呼ぶ）は，Java による分散オブジェクト技術であり，これも Java の持つ豊富なパッケージの一つである．RMI を使えば，ソケットレベルの制御を隠蔽し，データ交換のエンコードやデコードをしなくてすむ．さらに，サーバオブジェクトを代理オブジェクトを介してシームレスに操作できるので，プログラミング上，ローカルなオブジェクトとリモートオブジェクトを意識しないで実装可能である．

分散オブジェクトという観点からは，CORBA による通信も考えられる．CORBA は言語非依存であり，Java と C++ といった組み合わせも可能である．Java には JavaIDL という ORB が提供されており，この JavaIDL を介して別の ORB への接続を行なうこともできる．ただし，CORBA は汎用的なフレームワークであるため，RMI と比較してやや冗長的なコードを書かなければならないことと IDL を記述する必要がある．また，RMI にはオブジェクトの実体を値渡しできるメリットがあり[*1]，今回の検証モデルでは，RMI により通信をおこなうものとする．

3. 技 術 検 証

3.1 JavaApplet

JavaApplet が，Web クライアントとして位置づけられ，Fat と Thin の間に位置することはすでに述べた．その観点から検証モデルでは，ビジネスアプリケーションとして，ある程度の表現力をもった GUI と入力チェックなどを組み込んだ．Java には，AWT（AbstractWindowsKit）と Swing コンポーネント（以降，Swing とする）が用意されているので，両方を使用して評価をおこなった．

Swing は，Java 2 より標準で用意されている GUI コンポーネントである．AWT では，GUI 作成のための基本的な部品しか提供していないのに対して，Swing では

豊富な部品が用意されており表現力豊かな画面を構築可能である．したがって，Java 2 以降の開発では，GUI に Swing を採用するシステムが多くなると予想される．しかし，JavaApplet への適用を考えると，ブラウザに搭載されている JVM が Java 2 に対応していなければ使用することはできない．イントラネットの範囲であれば，JavaPlug-in を導入させるといったルールも適用可能であるが，不特定のユーザを対象にするインターネットシステムでは，ブラウザに搭載されている JVM に依存するため，Swing ではなく AWT を選択することにならざるを得ない．

AWT は，Swing に比べると表現力にはやや乏しいが，HTML ベースの画面よりは複雑な画面を構成できる上に，GUI 制御機能や入力チェックなどの機能を盛り込むことができる．さらにスクリプト言語と違い，ロジックが見えないという点でも有利である．したがって，今後も AWT の利用価値はしばらくあると考えられる．

ただ，AWT では稼働プラットホーム上の部品（ネイティブな部品）を使用するため，たとえば，Windows 95 ではリソースを食いつぶしたりすることがある．このため，設計上の注意点として必要以上にオブジェクトを生成しないようにしなければならない．また，プラットホーム間で GUI 部品の微妙な配置など完全な互換性が保たれないといった点もある．非互換についてはテストにより最大公約数的なレイアウトを検討していかなければならない．

一方，Swing コンポーネントでは，AWT と違い部品を Java 自身で描画している．そのため，AWT のようなリソースの問題は発生しない．また，部品の非互換に関しても解消される．ただ，Java で部品描画を行なうということは，メモリを大量に消費するため，低スペックな PC ではレスポンスが悪くなり重たくなる．AWT で実現していた部品のレスポンスに比べると反応が若干遅く感じる点は否めない．

Swing か AWT かの選択に関しては，クライアントの稼働環境を十分に考慮し，決定する必要がある．いずれにしても，Web クライアントとして考えれば，あまり重い処理をクライアントに搭載せず，どれだけ Thin Client に近づけられるかといった点がポイントとなろう．

3.2 JDBC

JDBC によるデータベースアクセスするためには，図 3 に示すような手順を踏む．

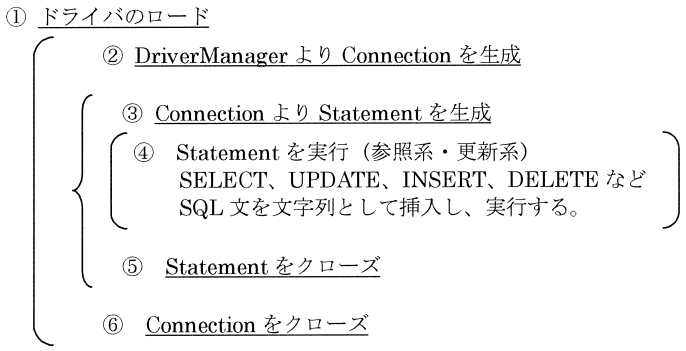

図 3　データベースアクセス手順

JDBC は，SQL 文を直接記述する比較的低レベルなインタフェースである．つまり，JDBC では実行する SQL 文字列の内容には関知せず，データベースに送るだけである．データベースが SQL 文字列を解釈できないようなケースがあった場合は例外が発生する．このことは，プログラムの実行時にはじめて SQL 文の正当性が評価されることを示している．

検証モデルでは，論理的に 3 層構造をとったことで，サーバプログラムにデータベースとのコネクションや SQL 文を隠蔽させるようにし，クライアントからは SQL 文そのものを記述しないようにした（図 4）．したがって，クライアントはデータベースのスキーマの変更に対して直接影響を受けることは少なくなり，システムの柔軟性はある程度確保できた．しかし，サーバ側には，直接 SQL 文を記述しているクラスが存在しており，コンパイル時点で SQL 文の正当性が評価できないため問題は残っている．

この問題への解決策としては，ストアドプロシジャ（以降，SP）による対応が考えられる．SP を使用すれば，事前に構文チェックが行なわれているので，SQL 文の正当性の評価を行なえる．また，呼出しもプロシージャ名だけで済む．しかし，SP は DBMS 上に格納されるため，ポータビリティという点で，新たな問題点が発生する．

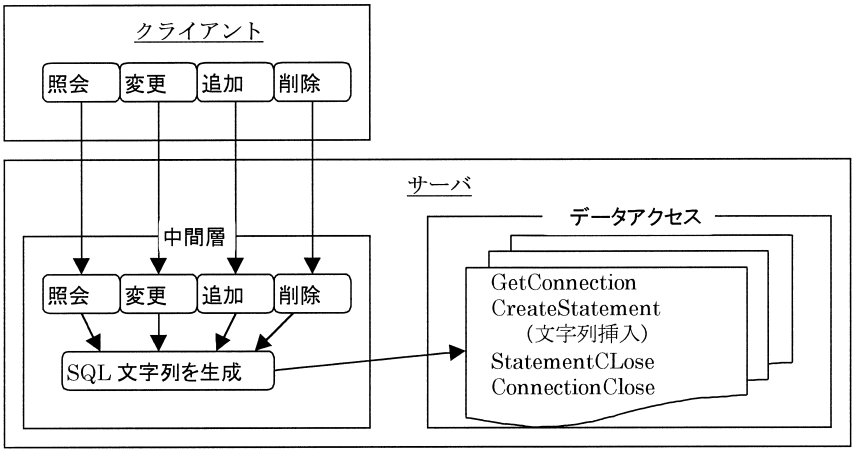

図 4　データベースアクセスに関するアプリケーション構造

今回は， Statement 文の間で複数の SQL を連続して実行する必要はなかったので，クライアントからの要求に応じて，その都度 Connection を作成し，SQL 文を一回実行し，クローズするようにした．実際にはトランザクションを明示的に使用（Commit/Rollback）し，詳細な制御をおこなうことは必須であり，また Connection を管理(プール)し，アクセスしてきたユーザを割り当てていくようなことが必要になる．

3.3　RMI

RMI は，Java の分散オブジェクト製品である．JDK 1.1 よりコア API として提供されている．RMI による分散アプリケーションのイメージは，図 5 に示す通りであ

る．

クライアントからサーバ（リモートオブジェクト）にアクセスするためには，①まず，サーバ上のオブジェクトが RMI レジストリに登録されていなければならない．②次に，LookUp という操作により，リモートオブジェクトの参照を得る．RMI レジストリでは，リモートオブジェクト名を指定すれば，登録してあるオブジェクトの参照を返す．③最後に，得たオブジェクトの参照によりメソッドを呼び出すことができる．この時，クライアント側にはスタブが，サーバ側にはスケルトンがあり，相互に引数の受け渡し等を実現してくれる．たとえば，サーバ側にあるオブジェクト（リモートでない）を引数として渡す場合は，そのオブジェクトを Java の Serialization 技術を使って転送する．

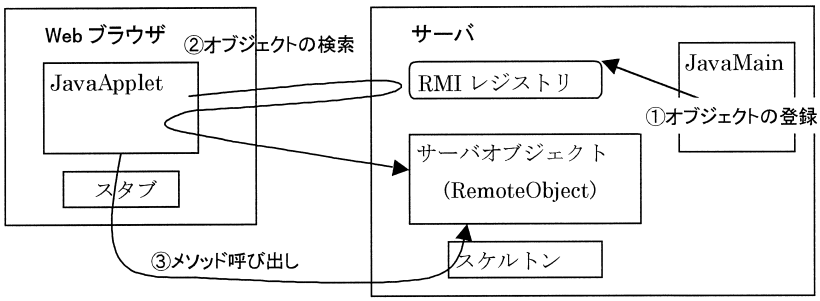

図 5　RMI による分散アプリケーションイメージ

以上のような手順を踏めば，リモート空間にあるオブジェクトをあたかもローカルなオブジェクト（JavaObject モデル）のように利用可能となる．

ただし，以下の点で違いがあるので注意が必要である．

- リモートオブジェクトのクライアントはリモートインタフェースに働きかけるのであり，これらのインタフェースの実装に働きかけることはない．
- リモートメソッド呼び出しに対するリモートでない引数と，その結果の返り値は参照渡しではなくコピー渡しになる．
- リモートオブジェクト呼び出しの失敗に関しては，その性質としてローカルオブジェクト呼び出しの失敗よりも複雑である．したがって，より多くの例外処理を扱う必要がある．

リモートオブジェクトとなるクラスには，インタフェースの記述が必要である．このインタフェースをもとにスタブ・スケルトンが生成されるからである．作業としても通常の javac によるコンパイル以外に rmic によるコンパイルが必要となる．しかし，インタフェースを記述することは，リモート空間のオブジェクトにアクセスするための作業だけでなく，クライアントとサーバをアプリケーション的に分離できるメリットがある．つまり，インタフェース定義により，クライアントはサーバ上の実装（Implementation）より独立できるため，クライアント実装担当とサーバ側の実装担当が別々に作業できるのである．サーバ側のリモートインタフェースさえ決まっていれば，クライアントは，サーバの実装状況に関係なく並行して開発できる．たとえば，

サーバの実装ができあがるまでは，テスト用の実装によりクライアントは開発を進めることができる．クライアントにとっては，サーバ側の実装が，テスト用であろうが本番用であろうが，関係なく開発作業に専念できる．インタフェースが変更されない限り，サーバ側の実装を意識する必要が無いのである．

4. 検証モデル評価

検証モデルの評価は，選択した技術に対しておこなった．評価ポイントは，以下の3点である．

1） Web クライアントとしての JavaApplet の有効性

2） Java とデータベースアクセス

3） RMI による効果

1）については，Web クライアントとしての JavaApplet は選択肢として有力であると考えられるが，実装するクライアントシステムが，Fat 寄りか，Thin 寄りかを判断して選択していくべきであろう．Fat であることは，ダウンロードサイズが大きくなるデメリットは当然のことながら，稼働するアプリケーションが重く（メモリ消費が多く）なることで十分なレスポンスが得られない可能性があるからであり，Thin であることは，細かな GUI 制御や入力チェックなどがクライアントサイドで十分にできないからである．

私見ではあるが，JavaApplet は，HTML ベースで設計可能なレベルの画面に入力チェック等を加えた程度の Thin クライアントに採択するのが実用的ではないかと思っている．今後も，Web ベースシステムが増えていくことが予想されるが，適用するビジネスシステムの特性を考慮し，本当に Web クライアントを使用するのかどうかといった点についても考慮する必要がでてくるだろう．

2）については，現在，Java からデータベースへのアクセスは JDBC 経由しか選択できない．そのため，SQL 文を文字列として直接記述することは避けられない．SQL 文をできるだけ汎用的に扱えるようなクラスを用意するか，事前にデータベースのスキーマとチェックし SQL 文の正当性を評価できるような仕組みが必要であると思われる．この点に関しては，SQLJ への期待は大きい．SQLJ は Java による埋め込み型 SQL であり，ANSI/ISO で標準として承認された仕様である．少なくとも，SQL 言語のセマンティクスとシンタックスがチェックされた後に JDBC を使用したプログラムへの変換を自動的に行なえる．このことは，データベースアクセスプログラムの生産性の向上を生むばかりでなく，品質，パフォーマンスへの期待が広がる．

3）については，通信制御部分の隠蔽だけでなく，リモートオブジェクトをシームレスに操作できることの効果は大きい．実際にネットワークプログラミングを意識する必要はないため，実装担当者は本来のアプリケーションロジックに専念できる．また，RMI による例をみるように，インタフェースと実装の分離は，作業の独立性を確保し，生産性をあげるメリットがある．Java の場合，言語レベルで Interface が規定されており，このことでオブジェクト指向の良い点（カプセル化，多相性）を実現することができるので，設計上の重要なポイントとなる．

RMI が Java のインタフェースを記述するだけで良いのに比べると，CORBA では

IDL を別途記述する必要がある．さらに IDL の型マッピングについても配慮しなければならない．言語独立性と他システムとの連携を含めた相互運用性を考えれば CORBA の選択によるメリットは大きいが，開発上のオーバーヘッドを考えると Java–Java 間の接続に関しては，RMI が適していると考えられる．

5. おわりに

今回の検証モデルにより，Web 技術を融合したクライアントサーバシステムに Java の基本的な技術をどう適用していくべきかの実際を評価できた．また，Web クライアントはどうあるべきかといった点についても考えることができた．Java を使えば，ある程度複雑なロジックをもつ Web クライアントが構築可能であり，データベースアクセスについても JDBC を介しておこなうことが可能であった．また，RMI により簡単にクライアントサーバ間の通信をおこなうことができた．

これらの要素技術は，Java の中心的な技術であり，将来的に中規模から大規模システムへ Java を適用する際にも重要な技術である．たとえば，RMI は，Java–Java 間の通信だけでなく，CORBA オブジェクトとの通信が可能になる．（RMI over IIOP）また，JDBC についても拡張が進んでいる．

したがって，今後は，Java のエンタープライズ API である EJB や JavaServlet，JavaServerPages（JSP）などを中心に，本格的なビジネスアプリケーションにとって必要な技術（トランザクション管理，データベースロック，大量データの扱いなど）について評価を行なっていかなければならない．また，CORBA を含めたシステムのスケーラビリティにも注目し，エンタープライズレベルのシステムへの適応を模索していく必要がある．

＊ 1　CORBA 2.0 の仕様では，オブジェクトコピーはサポートされていない．CORBA 3.0 で採用される予定である．

参考文献　［1］　萩本順三，福村真奈美，不破康人共著，『最新オブジェクト指向技術応用実践』，エーアイ出版，1998. 1，4 章 Java とデータベース，5 章分散オブジェクト．
［2］　宮脇亨，『クライアントサーバシステムにおける Java の適用について』，テクニカルシンポジウム 98，1998. 10.
［3］　植田龍男，『Java 入門実践編―RMI を活用した分散オブジェクト』，月刊 JavaWorld，IDG コミニケーションズ，1998. 11

執筆者紹介　宮　脇　　亨（Tooru Miyawaki）
1961 年生．1984 年神戸商科大学商経学部管理科学科卒業．同年日本ユニシス（株）入社．地域金融機関の営業店システム開発・適用に従事．1997 年 Java プロジェクトに参加し，現在，生産技術部情報技術室に所属．