Java オブジェクト管理技術の動向

鬼塚 真[†1] 大場 克哉[†2] 小島 秀登[†3]
斉藤 信也[†4] 高橋 肇[†5]

ウェブブラウザ及び Java 言語の普及によりアプリケーションの世界は大きく変化し，現在注目を集めている移動エージェントや分散コンピューティング等においても Java 言語は核技術として重要性が高い．更に，既存のプログラミング言語においてデータの共有が重要であったように，Java 言語においてもオブジェクトを永続化して共有する機構が重要である．

上記の背景から，本論文では 3 つの項目，1) アプリケーションインタフェース，2) オブジェクトの永続化指定インタフェース，3) Java 言語と DBMS 内部言語との接点，から Java オブジェクトの管理技術について検討・比較する．更に代表的な Java オブジェクトの管理システムについて，その設計方針と特徴的な機能について概説する．

The Interfaces and Implementations for Java Object Management

MAKOTO ONIZUKA,[†1] KATSUYA OBA,[†2] HIDETO KOJIMA,[†3]
SHINYA SAITO[†4] and HAJIME TAKAHASHI [†5]

The spread of web browsers and Java has influenced application development environment, and Java is also a core technology in mobile agent and distributed computing. In addition, it is important to share persistent objects in Java applications similarly in other existing programming languages.

In this paper, we discuss and compare the interfaces and implementations for Java object management from three viewpoints, 1) an application interface, 2) an interface to decide objects persistent. 3) a connecting point between Java and DBMS internal interfaces, We also survey famous Java object management systems and summarize their design policies and features.

1. はじめに

ウェブブラウザの普及によってインターネット上に蓄積された情報の参照が容易になり，その後の Java 言語の普及によってインターネット上の Java アプレットや Java オブジェクトの共有が促進された．これらのことにより既存のアプリケーションの世界は大きく変化し，Java 言語によるアプリケーション開発が台頭してきた．また，現在注目を集めている移動エージェント技術[1),2)] や分散コンピューティング[3)] 等においても Java 言語は核技術として重要性が高く，今後も Java 言語及び周辺環境は進化していくものと考えられる．一方，従来のプログラミング言語により実装されるアプリケーションにおいて，データを共有することは重要であり，ファイルやデータベース管理システム (DBMS) を用いて実現されてきた．同様に Java 言語により実装されるアプリケーションにおいても，オブジェクトの共有は重要な機能の 1 つであると考えられる．

上記の背景から，本論文では Java オブジェクトの管理技術について動向を調査し，そのインタフェース及び実装技術について検討・比較する．特に Java 言語と関連が深いと考えられる 3 つの項目，1) アプリケーションインタフェース，2) オブジェクトの永続化指定インタフェース，3) Java 言語と DBMS 内部言語の接点，について検討・比較する．また，代表的な Java オブジェクトの管理システムとして GemStone/J, O2, Objectivity/DB, ObjectStore(PSE/PSE Pro 含む), Oracle8i, PJama を取り上げ，それらの設計方針及び

†1 NTT サイバースペース研究所
NTT Cyber Space Laboratories
†2 オージス総研
Osaka Gas Information System Research Institute
†3 日本総研
The Japan Research Institute
†4 オブジェクト デザイン ジャパン
Object Design Japan
†5 日本電子計算
Japan Information Processing Service

特徴的な機能について概説する．

1.1 本論文の構成

本論文の構成を示す．2ではJavaの利点・欠点及びJavaオブジェクトの管理技術について概説し，Javaオブジェクトを管理するデータベースの重要性について確認する．3では上述した3つの項目からJavaオブジェクト管理技術について検討・比較する．4では代表的なJavaオブジェクトの管理システムについてそれらの設計方針，3で述べた仕様・実装技術との対応関係，その他特徴的な機能について整理する．

2. Javaについて

本章ではJavaの利点と欠点について整理することでJavaの重要性を再確認し，その後Javaオブジェクトの管理技術について概説する．

2.1 Javaの重要性について

Javaには以下の4つの利点があると筆者らは考える．

オブジェクト指向言語 Java自身が洗練されたオブジェクト指向言語である．特徴的な点としてインタフェース継承，ガーベジコレクション，スレッド，エラーハンドリング，ポインタの排除，充実したライブラリ群，等の機能や環境を有する．これらによって，従来のオブジェクト指向言語と比較して，アプリケーションプログラムの実装が簡易化されている．

移植性 Javaコンパイラは仮想計算機(Java virtual machine, 以降JVM)で動作するバイトコードを生成する．このためJVMが搭載されていれば，ウェブブラウザやオペレーティングシステムを問わずJavaプログラムを実行可能である．

セキュリティ JVMにセキュリティ保護機構が実装されている．このためJVMを搭載するウェブブラウザにJavaアプレットをダウンロードして実行しても，Javaアプレットを実行する計算機のセキュリティが保護される．

分散オブジェクト 遠隔オブジェクトのメソッド起動(Remote method invocation)のAPIが提供されている．この機能によって分散アプリケーションの構築が簡易化されている．

2.2 Javaの欠点について

Javaには主に以下の2つの欠点があると筆者らは考える．

性能 Javaのバイトコードのロード時にJIT(Just in time compiler)を用いてバイトコードをネイティブコードへ変換する等の方策により，ロード後の性能改善が図られてきている．しかし，大量データを扱う際のJVMのメモリ管理部の問題や，ガーベジコレクションの処理コスト等の性能的問題は依然として残っている．

機能 他のオブジェクト指向言語と比較して不足している機能(多重継承，ポインタ，パラメータ化クラス(クラステンプレート))があり，このためソースコードの書き方に制約がある．

2.3 Javaオブジェクトの永続化の方法

従来のプログラミング言語により実装されるアプリケーションでは，ファイルやDBMSを用いてデータを永続化していた．同様に，Javaにより実装されるアプリケーションにおいても，オブジェクトの永続化は重要な機能の1つであると考えられる．Javaオブジェクトを永続化する方法としては以下の3つの方法が挙げられる．

シリアライザを用いる Javaのシリアライザを用いてメモリ上のJavaオブジェクト群をまとめてストリーム化してファイルに格納する方法．メモリ上のJavaアプレットやJavaアプリケーションの状態を保存し，逆に保存されたファイルの情報からメモリ上の状態を復元することが可能である．しかし，DBMSが有する高度な機能である，トランザクション管理・検索処理の機能等は有していないため，永続オブジェクトを扱う多くのアプリケーションにとって十分な機能とはいえない．このため本論文ではシリアライザについては議論しない．

RDBMSを用いる JDBC[4], SQLJ[5], JavaBlend[6], EJB[7),8)]等を利用してリレーショナルデータベース管理システム(RDBMS)へJavaオブジェクトを格納する方法．詳しくは次章で述べる．

ODBMSを用いる Java Binding[9], EJB等を利用してオブジェクト指向データベース管理システム(ODBMS)へJavaオブジェクトを格納する方法．詳しくは次章で述べる．

3. 比較項目

本章では，永続的なJavaオブジェクトを管理する技術について，特にJava言語と関連が深いと考えられる以下の3つの項目から検討・比較する．

アプリケーションインタフェース アプリケーション側へ提供されるインタフェース(API)についての仕様と技術に関する項目．但し，API全般に関して本項目だけで説明することは困難であると考え，オブジェクトの永続化指定インタフェースに関しては，本項目とは独立に下記の第2番目の項目で整理することとする．

オブジェクトの永続化指定インタフェース APIの

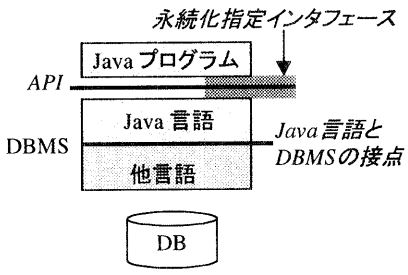


図1 各比較項目の位置づけ
Fig. 1 Comparison viewpoints

一部の機能である，オブジェクトを永続化指定するインタフェースに関する項目．型と永続性の独立性に関してはこの項目において整理する．

Java 言語と DBMS 内部言語の接点 DBMS 内部において永続の Java オブジェクトを扱うためには，Java 言語側から DBMS を実装する言語側の関数を起動できなければならない．本項目は，Java 言語と DBMS 内部言語の接点を実装する技術に関する．本項目は上述した API とは異なり，コーディング時にプログラマに影響しないが，Java プログラム実行時の性能やソースコードの移植性という点でプログラマに影響を及ぼす．

3.1 アプリケーションインタフェース

本章では，アプリケーション側へ提供されるインタフェースについての仕様と技術を比較・整理する．比較の際に重要な観点は以下の3点であると考える．

観点 1: 型管理

アプリケーションとデータベースとで型を独立して管理するのか，統合して管理するのかという観点である．型を独立して管理する場合，プログラマにとって以下の利点がある．

利点 1 特定のプログラミング言語にデータベースが依存しないため，データベースに格納された永続オブジェクトを複数のプログラミング言語から共有することが容易である．

利点 2 既存のデータベースを利用したアプリケーションを構築することが容易である．

一方欠点は次の通りである☆．

欠点 1 類似する型情報の管理をアプリケーション側とデータベース側の2箇所で管理しなければならない．

欠点 2 アプリケーションとデータベースとでは型システムが異なるため，アプリケーション側とデータベース側との型の対応を記述しなければならないし，更に実行時において変換のための余分なコストがかかる．

欠点 3 型情報が二重に管理されるため，より多くのメモリ空間を消費する．

また，型を統合して管理する場合についての利点・欠点は，型を独立して管理する場合の正反対になるため省略する．

観点 2: 格納する DBMS の種類

データの格納する部分に RDBMS を使うのか，ODBMS を使うのかという観点である．RDBMS を利用する場合，プログラマにとって以下の利点がある．

利点 1 OLTP 等の RDBMS が適するアプリケーションでは高性能を発揮できる．

一方欠点は次の通りである．

欠点 1 Java の操作モデルと RDB の操作モデルに表現能力のギャップがあるため，操作の変換手続きを記述しなければならないし，更に実行時において変換のための余分なコストがかかる☆☆．典型的な例としては，Java 言語側でのオブジェクト参照 (トラバース) 処理，属性値の参照処理，属性値の更新処理が，RDB 側ではそれぞれ JOIN 操作を含む検索，キー値を条件にした検索，キー値を条件にした更新処理，に変換されるため処理が高価になってしまう．

また，ODBMS を利用する場合についての利点・欠点は，RDBMS を利用する場合の正反対になるため省略する．

観点 3: 問合わせ言語

問合わせ言語として RDB に基づく問い合わせ言語 (SQL) が提供されているのか，オブジェクト指向のデータモデルに適合したオブジェクト問い合わせ言語 (OQL) が提供されているのかという観点である．SQL を利用する場合，プログラマにとって以下の利点がある．

利点 1 問合わせの言語仕様が簡略であるため，JOIN 操作を除けば最適化機構も簡略であり，最適化プラン作成のコストが少なく且つ最適なプランが選択される可能性が高い．

一方 OQL を利用する場合の利点は次の通りである．

利点 1 オブジェクト指向の概念である，継承，オブジェクト参照 (トラバース)，関連，等を指定した

☆ これらの欠点はアプリケーションとデータベースとの型システムの違いから生じていて，一般的には2つの言語の間でインタフェースを取る際に生じるインピーダンス・ミスマッチ問題として分類される [10]．

☆☆ この欠点はアプリケーションとデータベースとの操作モデルの違いから生じていて，これもインピーダンス・ミスマッチ問題として分類される [10]．

表1 APIの比較
Table 1 Application interfaces comparison

仕様/製品	型管理	DBMS	問合わせ言語
JDBC	独立	R	SQL
SQLJ	独立	R	SQL
Java Blend	統合	R	OQL
Java Binding	統合	O	OQL
EJB	統合	R,O	SQL,OQL

(SQLでは記述できない)問い合わせ文を記述することが可能である.

利点2 SQLではクラスの外延にのみ検索が実行可能であるが,OQLではコレクションクラスの要素に対して検索処理が実行可能である.オブジェクト指向設計されたアプリケーションでは,クラスの外延に対する検索よりもコレクションクラスの要素に対しての検索がより頻繁に用いられるため,この機能の重要性は高い.

以下の章では,JDBC, SQLJ, Java Blend, Java Binding, EJBのAPIについて,上記の3つの観点を基準に比較・整理する(表を1に示す).

3.1.1 JDBC

JDBCはJava言語からRDBへのSQLの発行及びその結果を扱うためのライブラリの仕様等を規定している.実際にはJDBCの仕様に対応したJDBCドライバを用いてRDBへアクセスすることができる.表1に示すとおり,JDBCでは1)アプリケーションとデータベースの型を独立に管理,2)格納するDBMSとしてはRDBMSを利用,3)上記1),2)に基づき問合わせ言語はSQL,である.

このためJDBCを用いる場合,上述した独立した型管理,RDBMS利用,SQL利用の各々の利点・欠点があてはまる.

3.1.2 SQLJ

SQLJはJava言語に対する埋め込みSQLの規格である.SQLJにより記述されたプログラムは,プリプロセッサによって処理され,JDBCを用いてRDBへアクセスするプログラムに変換される.表1に示すとおり,各観点についてSQLJはJDBCと同様であるため,利点・欠点についてもJDBCと同様となる.

3.1.3 Java Blend

Java BlendはJDBCドライバ上に構築されているソフトウェアであり,Java言語でのデータモデル・操作をRDBのデータモデル・操作へ変換する.表1に示すとおり,Java Blendでは1)アプリケーションとデータベースの型を統合して管理,2)格納するDBMSとしてはRDBMSを利用,3)問合わせ言語はOQL,である.

Java Blendを用いる場合,上述の統合した型管理,RDBMS利用,OQL利用の各々の利点・欠点の他に,利点としてアプリケーションとデータベースの間のデータマッピング規則をカスタマイズすることで既存のRDBを流用することが可能である,ことが挙げられる.

3.1.4 Java Binding

Java BindingはODMGにより標準化が進められているインターフェース仕様であり,クラスライブラリ,例外処理,検索言語(OQL)等を規定している.Java Bindingは次の3つの方針に従って整理されている[9].

(1) Java言語とデータベースは1つの統合的な型システムを共有する
(2) Javaの文法を拡張しない
(3) 推移的永続性(3.2.2で詳しく述べる)によりオブジェクトを永続化する

加えて表1に示したように,Java Bindingでは1)格納するDBMSとしてODBMSを利用,2)問合わせ言語はOQL,である.

このためJava Bindingを用いる場合,上述の統合した型管理,ODBMS利用,OQL利用の各々の利点・欠点があてはまる.

3.1.5 Enterprise JavaBeans(EJB)

EJBは,サーバ側の再利用可能なJavaコンポーネントモデルを規定している.EJBは以下の3つの基本要素から構成されている.

Enterprise Bean クラスに相当する.EJB 1.0では,非永続的であるsession beanと永続的であるentity beanの2種類が定義可能である☆,と規定している.

EJBコンテナ Enterprise Beanをラッピングするものであり,Enterprise Beanに対する操作は全てEJBコンテナを経由して実行される.

EJBサーバ EJBコンテナを経由してEnterprise Beanを実行するアプリケーションサーバに相当する.この部分でDBMSの機能である,トランザクション管理機能,スケーラビリティの確保,セキュリティ管理等が実現される.

また表1に示すとおり,EJBでは1)アプリケーションとデータベースの型を統合して管理,2)格納するDBMSとしてはRDBMSまたはODBMSを利用,

☆ 厳密にはsession beanのメソッドの内部で永続化処理をすることで,session beanを永続化することも可能である.

3) 問合わせ言語は SQL または OQL，である．

EJB を用いる場合，RDBMS を利用した場合は RDBMS 利用の利点・欠点があてはまり，一方 ODBMS を利用した場合は ODBMS 利用の利点・欠点があてはまる．またその他に下記の利点がある．

利点 1 EJB の環境ではビジネスロジックと分散管理のためのロジックが分離されるため，アプリケーションサーバとして EJB サーバは高性能・高スケーラビリティを実現しやすく，且つアプリケーションプログラマはビジネスロジックに集中することができる．

利点 2 コンポーネントベースの開発環境であるので，コンポーネントの再利用の促進が期待できる．

一方 EJB を用いる場合の欠点を以下に挙げる．

欠点 1 現状では EJB のコンポーネントは基本的なものだけであり，コンポーネントの充実にまだ時間がかかると考えられる．

欠点 2 EJB 自身が大規模システムを前提に考えられているため，サーバ側で必要とするリソースが大きくなる傾向がある．この結果，特に小規模システムでは性能に悪影響を及ぼしてしまうことも考えられる．

3.2 オブジェクトの永続化指定インタフェース

本章では，オブジェクトを永続化指定するインタフェースについて比較・整理する．オブジェクトの永続化指定方法は 1) クラスの永続可能化，2) インスタンスの永続化，の 2 段階の工程に詳細化される．各々について以下で詳細に述べる．

3.2.1 クラスの永続可能化

クラスの永続可能化の工程では，永続化可能であるクラスを決定し，同時にそのクラスのインスタンスを永続化する実装方式も決定する．クラスを永続可能にする手段は大きく下記の 2 種類がある．

(1) ソースコード上で永続可能であることを陽に指定する方法 (実装継承，インタフェース継承，デリゲーション，永続化可能クラスライブラリ等を利用する方法)．この方法は更に細分化され，Java の標準クラスを永続可能化する場合と，アプリケーション (AP) で新規に定義されるクラスを永続可能化する場合とがある．

(2) ソースコード上ではクラスが永続可能であることを指定しないで，内部的な処理で永続化可能性を付与する方法 (プリプロセッサ，ポストプロセッサ，JVM 拡張等を行う方法)．プリプロセスやポストプロセス方式では，実際には (1) と等価な処理をコードの変換時に埋め込む DBMS もある．

(1) の方法は，ソースコードの透明性は保たれるが，何らかの指定方法によりクラスを陽に永続可能化しなければならないため，プログラマの負担は大きい．(2) の方法は，ソースコードの変更が不要であるためプログラマの負担は少ないが，必要以上のクラスを永続化可能であると扱ってしまう可能性がある．例えば，永続可能化する必要の無いクラスを永続可能化した場合，非永続であるオブジェクトにおいても永続化のための情報を管理してしまうため，メモリやディスク空間が無駄になり且つ処理速度も劣化してしまうという問題がある．

Java Binding では Java の型と永続性を独立させるために (2) を採用しているが，多くの DBMS では，クラスの種類に応じてこの (1),(2) の方法を組み合わせることで効率的に永続オブジェクトの管理を実現している．

3.2.1.1 永続可能であることを指定する方法

実装継承 アプリケーションにより新規に定義されるクラスを永続化する際に，永続化のための特定のクラスを実装継承することで永続化可能性を付与する方法．実装継承による方法では，永続化に必要な情報 (OID, ロック情報) は 1) 永続オブジェクト内，2) デリゲート (委譲) して永続オブジェクトとは別の場所，のいずれかにより管理される．1) の方法ではオブジェクトの大きさが増大してしまうが，ロック取得等の永続オブジェクトの処理が簡略であるという利点がある．一方 2) の方法はオブジェクトの大きさは増大せず，且つ永続オブジェクトの処理が比較的簡略であるため，1) より優れた方法であるといえる．

しかし実装継承を用いることは適切な方法とはいえない．その理由は，Java では実装の多重継承がサポートされていないため，永続化を指定したクラスでは他のクラスを実装継承できないという問題が生じるためである．

インタフェース継承 上述の実装継承の欠点を解消するため，実装継承ではなくインタフェース継承を用

表 2 永続化可能クラスの指定方法
Table 2 Persistence-capable class specification methods

ソースコード	クラス種別	方法
指定する	AP 定義	実装継承
		インタフェース継承
	標準	デリゲーション
		永続化可能クラスライブラリ
指定しない	全て	プリプロセッサ
	全て	ポストプロセッサ
	全て	JVM 拡張

いてクラスに永続化可能性を付与する方法が考えられる．実装継承と異なる点は，実装を継承しないため永続化に必要な属性やメソッドを継承先のクラスで実装しなければならないことである．☆

デリゲーション Java の標準クラス (String, Array 等) に対し永続化可能性を付与したい場合，それらは既に定義済みであるため上述の継承を用いた方法を適用できない．この問題を解消する方法は2つあり，1つは次の項目で述べる永続化可能クラスライブラリを用いる方法で，もう1つはデリゲーションを用いる方法である．デリゲーションによる方法では，永続化可能性が付与されたクラスの属性の型として標準クラスの参照を用いた場合に，その標準クラスに永続化可能性を付与するという方法である．デリゲーションによる方法では，永続化に必要な情報を永続オブジェクトとは別の場所に保持するためオブジェクトの大きさは増大しない．しかし，永続オブジェクト側で永続情報へのリンクが管理できないため，ロック取得等の永続オブジェクトの処理の際に，永続オブジェクトから永続化情報を操作する処理が高価になってしまうという欠点がある．

永続化可能クラスライブラリ Java の標準クラスに対応した永続化可能クラスをライブラリとして提供する方法．ライブラリの実装方法としては，上述の継承と同様に 1) 永続化に必要な情報は永続オブジェクト内に格納する，2) 永続オブジェクトとは別の場所に格納する，のいずれかの方法により実現される．本方法の欠点は，標準クラスを永続化可能クラスへ書き換えなければならないことである．

3.2.1.2 永続可能であることを指定しない方法

Java プログラム変換 (プリプロセッサ) Java プログラムを変換して，永続化オブジェクトに対する手続き等を加える方法．欠点としてはソースコードが変換されてしまうため，デバッガでプリプロセス前のソースコードを利用してデバッグすることができなくなってしまうことである．

バイトコード変換 (ポストプロセッサ) Java プログラムから生成される Java のバイトコードを変換して，永続化オブジェクトに対する手続き等を加える方法．プリプロセッサ方式とは異なりデバッグ情報もバイトコードへ適切に埋め込むことで，デバッガも通常通り利用することができる．またバイトコードを利用するため，Java の標準ライブラリクラスのようにソースコードの無いクラスも永続化可能にできるという利点がある．

JVM 拡張 JVM を拡張する方法．永続オブジェクトを揮発オブジェクトと同様に扱えるように JVM を拡張しているため DBMS 自体の実装が複雑になるが，多様なチューニングによる性能改善が可能である．しかし，Java の大きな特徴である移植性が損なわれてしまう．

3.2.2 インスタンスの永続化

インスタンスの永続化の工程では，永続化可能なクラスのインスタンスが永続であることを決定する．インスタンスを永続化にする手段は大きく下記の2種類がある．

推移的に永続化 (推移的永続性) 名前つきオブジェクト (named object) を起点とし，そのクラス参照属性またはクラス参照のコレクションクラス属性により推移的に参照されうるオブジェクト群を全て永続化する方法．型とは独立に永続化の範囲を決定できるという利点があるが，逆にコミットの実行時に永続オブジェクトの判定処理が高価になる傾向がある．Java Binding ではこの方式を採用している．

常に永続化 生成されたオブジェクトを常に永続化する方法．永続オブジェクトの管理が簡略であるため処理が高速に行えるという利点があるが，同一クラスにおいて永続と非永続のオブジェクトの混在が許されないという制約がある．EJB の entity bean はこの方式に相当する．

3.3 Java 言語と DBMS 内部言語の接点

永続の Java オブジェクトを扱うためには，Java 言語側から DBMS を実装する言語側の関数を起動できなければならない．この Java 言語と DBMS 内部言語の接点の実装方法は以下の3つに分類できる．

Java DBMS DBMS 自体を Java で実装することでシームレスに結合する方法 (図2 左参照)．

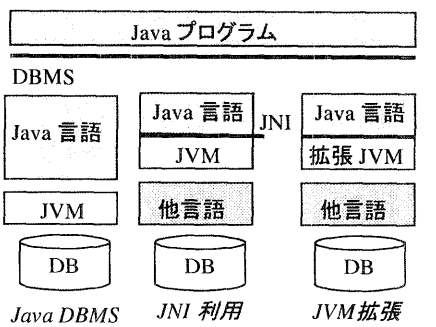


図2 Java 言語と DBMS 内部言語の接点
Fig. 2 A connecting point between Java and DBMS internal interfaces

☆ EJB の場合 entity bean クラスはインタフェース継承によって実装され，EJB コンテナまたは entity bean において永続化に必要な情報が管理される．

利点としては，Javaの利点を引き継ぐため移植性が高いことが挙げられる．逆にJavaの欠点も引き継ぐため，ヒープを大量に使用した際のJVMの性能劣化やガーベジコレクションによる処理遅延等の問題があるため，大規模のデータベースを扱うことに適さない．

JNIを用いる方法 DBMS自体はC++等Java以外のプログラミング言語で記述しておき，標準APIであるJNI(Java native interface)を用いて，Java言語側からDBMSの内部言語側の関数を起動する方法(図2中参照)．

利点としては，100% Javaのクライアントアプリケーションをサポートしながら，DBMS側ではC++等のプログラミング言語によってデータベースサーバを構築することが可能であるため，高性能・高スケーラビリティを実現可能である．更にJavaの言語に依存しない形式で永続オブジェクトを格納するため，他のプログラミング言語からも永続オブジェクトを共有することが可能である．しかし，JNI自身のオーバヘッドが大きく，特に規模の小さいデータベースの場合はJNIの処理量が大きいウェイトを占めてしまい，性能が劣化するという問題がある．

JVMを拡張する方法 DBMS自体はC++等Java以外のプログラミング言語で記述しておき，JVMを拡張することで，JVM内部からDBMSの内部関数を起動する方法(図2右参照)．

多様なチューニングが可能であるため，上述したJava DBMSやJNIでの性能的問題を改善することが可能である．しかし，Javaの大きな特徴である移植性が損なわれてしまう．

4. Java DBMSの製品の概要

本章では，代表的なJavaオブジェクトの管理システムとしてGemStone/J, O2, Objectivity/DB, ObjectStore(PSE/PSE Pro含む)を取り上げ，それらの設計方針，3で述べた仕様・実装技術との対応関係，その他特徴的な機能について整理する．また，その他としてOracle8i, PJamaについて概略を述べる．他にもJavaオブジェクトの管理システムはいくつかあるが，それらについては文献[14]を参照されたい．

4.1 GemStone/J

GemStone/Jの設計方針を以下に示す．

大規模Java開発・運用環境の提供 GemStone/Jでは，サーバ上でJVMを搭載することにより，Javaクライアント環境とサーバ環境間でのパーティショニングを実現し，Persistent Cache Architecture(PCA)と呼ばれる共有オブジェクト・リポジトリにて，Javaオブジェクトを永続化することでオブジェクトを共有化している(図3)．GemStone Systems社では，このアプリケーション・モデルをDistributed JavaBeans(DJB)と呼んでおり，他にEJB及びCORBAでの利用もGemStone/JのJVMをとおしてPCAにて同様に共有化される．同様に，JDBCを用いることでRDBに格納されているデータをPCAにオブジェクトとして共有化することが可能である．GemStone/JのJVMはJava2対応のJVMであり，Pure Java環境のプログラムを実行が可能である．PCAは20億オブジェクトの格納ができ，Java2のセキュリティ，GemStone/Jのフォールトトレラントな各種サービス，CORBAによる分散により，Javaによる大規模で高可用性な環境を提供する．

前章で論じたJava DBMSの比較の項目について整理すると，

(1) APIはEJBをサポートする．

(2) 標準クラスの永続可能化についてはGemStone/J内蔵のJVMにて行い，アプリケーション定義クラスについてはインタフェース継承による方法により実現している．

(3) Java言語とDBMS内部言語の接点はJVMの拡張により実装している．

その他，特徴的な機能としては，分散環境をサポートする2フェーズ・コミット機能や，トランザクション・マネージャによるTPモニタ機能，JDBC接続時のコネクション・プーリング機能を提供している．以上のように，GemStone/Jは，Javaアプリケーション・サーバとODBMSを融合させた製品である．

今後は，よりEJBに関連した機能を付加していく予定である．具体的には，Java2, Java Server Pages(JSP), Javaサーブレット，EJB, CORBAのORB等のモデルやRDBMSデータ，オブジェクト・トランザクション・モニタ(OTM)機能を，内蔵のオブジェクト・リポジトリにて統合，共有化を行うことにより，Javaによる大規模なアプリケーションの開発・運用環境を提供していく予定である．

4.2 O2

O2の設計方針は次の3点である．

高性能・高スケーラビリティ 大規模・多ユーザのデータベース構築時にも高性能を発揮し，高い信頼性と可用性も併せ持つデータベースエンジンを提供．

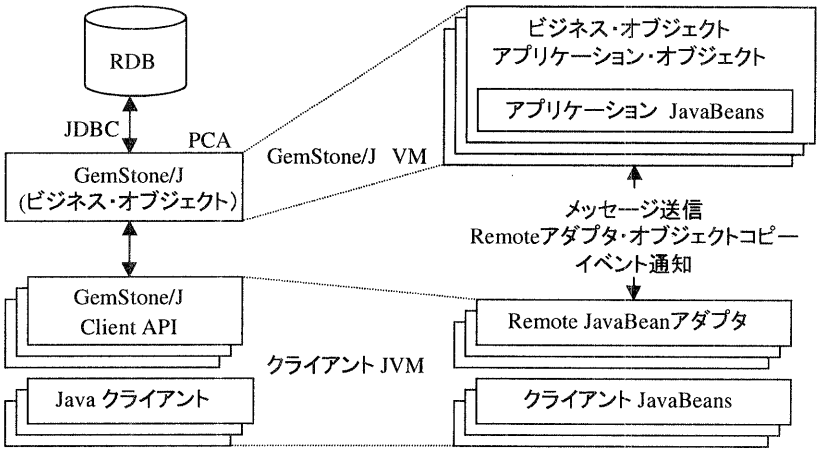


図 3 Distributed JavaBeans アーキテクチャ
Fig. 3 Distributed JavaBeans architecture

オープン化と標準への準拠 ODMG, JDBC 標準準拠の他，分散トランザクションの標準である JTS(Java Transaction Service)[11], XA 等にも準拠．

言語独立&データベース独立 異なるプログラミング言語間でデータベース中のオブジェクトを共有でき，また O2 用コードを RDB に対して使用することもできる．

既に論じた Java DBMS の比較の項目から整理すると，

(1) API は Java Binding である．

(2) クラスの永続可能化については標準クラスは永続化可能クラスライブラリを用いる方法により提供され，アプリケーション定義クラスについてはバイトコード変換（ポストプロセッサ）による方法を用いて実現している．またオブジェクトの永続化については，推移的永続性により実現している．

(3) Java 言語と DBMS 内部言語の接点は JNI を用いて実装している．

以上のような方式の採用に加えて次の特長を有する．

- 開発者は Java クラスの記述を扱うだけで良く，データベース内の永続オブジェクトをどのようにして読み書きするかを気にする必要が無い．
- システムは Java のクラス定義からデータベーススキーマを自動的に生成する．
- オブジェクトのリード／ライトのためのデータベースアクセスは，バイトコードのポストプロセスにより必要な個所に透過的に加えられる．
- スキーマの進化が O2 Engine によりサポートされているため，既存の永続オブジェクトを失うことなく，Java クラスのどんな修正も可能である．
- O2 の最適化されたクライアント／サーバアーキテクチャによって，作成されたアプリケーションを効率的に実行できる．
- オブジェクトは安全なマルチユーザ環境でリアルタイムに共有できる．
- Java のバイトコード自体もデータベースに保存できる．

4.3 Objectivity/DB

Objectivity/DB の設計方針は以下の 2 点である．

高スケーラビリティ 大規模データベース構築時にも安定した性能を提供する．

高可用性 ミッションクリティカルシステムでの使用に耐えうる耐障害性，高可用性を提供する．

前章で論じた Java DBMS の比較の項目について整理する．

(1) API は Java Binding であり，ODMG2.0 対応に加えて，Objectivity/DB が提供する Java API が提供されている．Objectivity/DB による Java API では，必ずしも推移的永続性によらないオブ

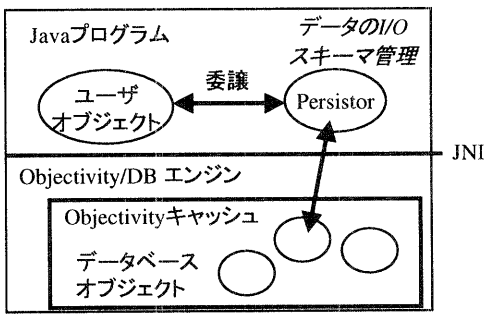


図 4 Objectivity/DB for Java アーキテクチャ
Fig. 4 Objectivity/DB for Java architecture

表 3 Java オブジェクト管理システムの比較
Table 3 Java object management systems comparison

製品	API	クラスの永続化指定	DBMS 接点
GemStone/J	EJB	標準: JVM 拡張 AP 定義: インタフェース継承	JVM 拡張
O2	Java Binding	標準: 永続可能化クラスライブラリ AP 定義: ポストプロセッサ	JNI
Objectivity/DB	Java Binding, 独自 API	標準: デリゲーション AP 定義: 実装継承, インタフェース継承	JNI
ObjectStore	Java Biding	ポストプロセッサ, 標準: デリゲーション AP 定義: インタフェース継承	JNI
ObjectStore PSE	Java Binding	同上	Java DBMS
Oracle8i	JDBC, SQLJ	RDB 側で管理	JNI(JDBC)
PJama	独自 API	JVM 拡張	JVM 拡張

ジェクトの永続化，ストレージ階層の概念の導入，高速検索のための各種機能など，ODMG に定義されていない機能を含む Objectivity/DB の全機能を利用可能である．これらは特に，大規模データベースに効率的にアクセスするために重要である．

(2) アプリケーション定義クラスの永続可能化については，実装継承およびインタフェース継承による方法を提供している．実装継承による方法では，3.2.1.1 の 2) デリゲート方式を採用することにより，オブジェクトサイズの問題を解決している．すなわち，データベースに対する処理は，Persistor と呼ばれるオブジェクトへ制御を委譲し，これがデータベースに格納されているオブジェクトに対する一種のプロキシオブジェクトとして機能する (図 4)．データベースのスキーマは，永続可能クラスの定義をもとに実行時に動的に登録される．したがって，あらかじめ DDL（データ定義言語）をプリプロセスしたり，あるいはソースコード，バイトコードをポストプロセスする必要はない．スキーマ進化機能により，ソースコードレベルとデータベーススキーマ情報は常に最新のものに保たれ，メンテナンス性の向上も期待できる．なお，標準クラスの永続可能化にはデリゲーション方式を採用している．

(3) Java 言語と DBMS 内部言語の接点は JNI を用いて実装している．DBMS にとって，キャッシュを効率的に管理することがその性能にとって最も重要な要素であり，DBMS 本体はこれが可能な環境，プログラミング言語で記述されるべきだと考えるからである．

その他，特徴的な機能は以下のとおりである．

- レプリケーション機能により，耐障害性，高可用性を実現している．各レプリカに重み付けすることによって，障害発生時に全レプリカへのアクセスができなくても，(重み付け後の) 過半数のデータベースへアクセス可能であれば，更新を許し，障害復旧後に自動的に再同期を行う．また，レプリカを適切に配置することにより，データアクセスの高速化も同時に図ることが可能である．
- データベースクライアントにおけるマルチスレッド，マルチトランザクションをサポートしている．これは CORBA, EJB 等と組合せて分散オブジェクトシステムを構築する際に，キャッシュを効率的に利用するという観点から有効である．
- Objectivity/DB の対応言語である Java, C++, Smalltalk 間でインターオペラビリティを提供している．

今後は，上述の設計方針に基づき Java による大規模ミッションクリティカルシステム構築をサポートするための機能を追加，強化していく予定である．執筆時点で項目として上がっているのは，

- オブジェクト指向上流 CASE ツール (Rational Rose) 上で定義された UML(Unified Modeling Language) によるオブジェクト指向モデルを利用した，グラフィカルなデータベーススキーマ管理の実現．
- CORBA トランザクションサービス，イベントサービス等をサポートする，CORBA との統合のためのフレームワークの提供．
- 各種アプリケーションサーバー製品 (EJB 等) との統合のためのフレームワークの提供．
- Collection クラスのスケーラビリティの強化．

等である．

4.4 ObjectStore

ObjectStore の設計方針は以下の 3 点である．

高言語透過性 一時オブジェクトと永続オブジェクトのライフタイムの違いを除いて，それらに対して

同一のシンタックス及びセマンティクスでアクセスできるよう設計されている．このため，プログラマはオブジェクトの永続性をほとんど意識せずにコーディングすることが可能である．

小規模から大規模システムまでのサポート シングルユーザ・スタンドアローンに機能を限定したPSE及びPSE Proと，本格的なDBMS機能を実現したObjectStoreの複数製品を用意している．これらは統一されたAPIを提供しているので，アプリケーションのアップグレードも容易である．

高スケーラビリティ・高可用性 ObjectStoreについては，大規模かつミッション・クリティカル・システムに対応できるよう，微細度排他制御処理によるマルチ・ユーザ・アクセス，オンライン・バックアップ，差分バックアップ，ロール・フォーワード，レプリケーション，DBMS二重化等の機能を提供する．

前章で論じた項目について整理する．

(1) APIはJava bindingである．ODMG2.0の全機能に加えて，キャッシュを制御するための各種APIや，インデックスを用いて検索の最適化を制御するためのAPIが提供されており，高いスケーラビリティが実現されている．また，JDK1.2の仕様に準拠したコレクションライブラリが提供されている．

(2) オブジェクト永続化可能性の実装は，インタフェース継承とデリゲーションを併用している．また，ポストプロセッサを提供しており，通常はこれを用いて永続化可能にしたいクラスにアノテーションを行い，永続化のための実装を自動生成する．

(3) DBMS内部言語との接点は，製品によって実装方法を変えている．PSE/PSE Proは小規模DBを対象にした製品であるので，DBMS全体をJavaで記述している(100% Pure Javaの認定も取得している)．これにより高い移植性が実現できるので，デバイス等への組込み用途でも使用されている．一方，ObjectStoreはJNIを用いた実装を行っており，ネイティブ・コードレベルでの性能を実現している．

その他，特徴的な機能は以下の通りである．

- ローエンド商品のPSEはフリーウェアとして公開されており，誰もが無料で使用することができる．
- 各種周辺ツールやライブラリが充実している．マルチメディアデータ等を扱うライブラリ群であるObjectManagers，データベースの内容を閲覧するInspector，UMLやOMTを用いてデータベース設計を行うDatabase Designer，Webサーバと併用する際に使用するObjectForms等が提供されている．

将来計画については，ObjectStoreの次期メジャーバージョンであるR6.0では，スケーラビリティの大幅な改善が予定されている．これを実現するために，DBMSエンジンの基本設計に対しても大規模な見直しが行われている．また，マルチセッション及びマルチスレッド対応等の新機能の追加も予定されている．更にEJBとのインテグレーションも予定されている．またR6.0の他に，ObjectStoreを内部に組み込んだeXcelonと呼ばれる新製品も発表されている．これはXML[15]データをパース後のDOM[16]ツリーの形で格納する新しい概念のデータベースである．eXcelonに格納されているXMLデータへは，DOM API及びXQL[17]，XUL(XML Update Language)等を通じてアクセスする．

4.5 その他

上述以外のJavaオブジェクトを管理する代表的な製品であるOracle8i及びPJama(旧名称PJava)[12),13)]について整理する．

Oracle8iは，オブジェクト指向の機能をRDBMSへ取り込んだオブジェクトリレーショナルデータベース管理システム(ORDBMS)であり，Javaオブジェクトを永続化して管理する機能を有する．

既に論じたJava DBMSの比較の項目から整理すると，

(1) APIはSQLJ, JDBCである．

(2) クラスの永続可能化については，Javaとは独立にRDB側で実現される．

(3) Java言語とDBMS内部言語の接点は，JDBC内でJNIを利用することで実装していると考えられる．

一方，PJamaはJava言語のための永続的プログラミングシステムであり，設計方針は以下の3点である．

直交永続性 クラスに依存せずにオブジェクトを永続化できる．

推移的永続性 永続化対象のオブジェクトは推移的に決定される．

永続性独立 永続オブジェクト・揮発オブジェクトに関わらず，プログラムを統一的に記述できる．

既に論じたJava DBMSの比較の項目から整理すると，

(1) APIは独自APIである．

(2) クラスの永続可能化についてはJVM拡張により実現し，インスタンスの永続化については推移的永続性により実現している．

(3) Java言語とDBMS内部言語の接点はJVMの

拡張により実装している.

5. むすび

本論文では Java オブジェクトの管理技術について動向を調査し，そのインタフェース及び実装技術について検討・比較した．特に Java 言語と関連が深いと考えられる 3 つの項目，1) API, 2) オブジェクトの永続化指定インタフェース，3) Java 言語と DBMS 内部言語の接点，について検討・比較した．

1) の API に関しては，性能的な観点から考えた場合 Java Binding 及び EJB が有力であり，特に EJB はまだ成熟した技術ではないが，DBMS 技術を統合したアプリケーションサーバにおけるコンポーネントモデルを規定しており今後の発展が期待される．

2) のオブジェクトの永続化指定インタフェースに関しては，大きくクラスの永続可能化とインスタンスの永続化の 2 段階の工程に分けられた．前者のクラスの永続可能化については，更に 2 つに分類されソースコード上で永続可能であることを陽に指定する方法 (実装継承，インタフェース継承，デリゲーション，永続化可能クラスライブラリ) と，ソースコード上で永続可能であることを指定しない方法 (プリプロセッサ，ポストプロセッサ，JVM 拡張) とがあった．多くの DBMS では，クラスの種類に応じてこの 2 つの方法を組み合わせることで，効率的に永続オブジェクトの管理を実現している．一方，後者のインスタンスの永続化については，推移的永続性による方法と常に永続化する方法の 2 つの方法があった．推移的永続性による永続化の方法は Java Binding で採用しており，常に永続化する方法は EJB で採用されている．

3) の Java 言語と DBMS 内部言語の接点に関しては，DBMS 自身を Java で実装する方法，JNI を利用する方法，JVM を拡張する方法，の 3 つの方法があった．各方法はそれぞれ利点・欠点があるため，利用するアプリケーションに応じて適切な方法により実装される DBMS を選択することが良いと考えられる．

また，代表的な Java オブジェクトの管理システムとして GemStone/J, O2, Objectivity/DB, ObjectStore(PSE/PSE Pro 含む), Oracle8i, PJama を取り上げ，それらの設計方針及び特徴的な機能について概説した．

今後，情報家電の普及や通信と放送との融合が進むに従い，家庭内 LAN に代表されるように益々ネットワークが世の中に浸透し，その結果大量の情報またはオブジェクトがネットワーク上で管理され，それらを共有する枠組が重要になっていくものと考えられる．Java オブジェクトの管理技術が，その枠組における 1 核技術として位置付けられ，今後も発展していくことを期待したい．

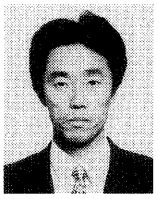
参考文献

1) Glass, G.: ObjectSpace Voyager — The Agent ORB for Java, *Lecture Notes in Computer Science*, Vol. 1368, pp. 38-55(1998).
2) 本位田 真一: モバイルエージェント技術, オブジェクト指向最前線'98, 朝倉書店, pp. 228-233(1998).
3) Matsuoka,S., Ogawa,H., Shimura,K., Kimura,Y., Hotta,K., and Takagi,H.: OpenJIT — A Reflective Java JIT Compiler, *Proc. of OOPSLA '98 Workshop on Reflective Programming in C++ and Java*, pp. 16-20(1998).
4) Sun microsystems Inc.: *The JDBC(tm) Data Access API*, URL http://java.sun.com/products/jdbc/.
5) Oracle: *SQLJ: Embedded SQL in Java*, URL http://www.oracle.com/java/sqlj/.
6) Sun microsystems Inc.: *Java(tm) Blend*, URL http://www.sun.com/software/javablend/.
7) Sun microsystems Inc.: *Enterprise JavaBeans(tm)*, URL http://java.sun.com/products/ejb/.
8) Barry & Associates: *EJB Application Servers*, URL http://www.odbmsfacts.com/ejb_servers/ejb_servers.html.
9) Cattell,R.G.G.and Barry,D.K.: *The object database standard: ODMG2.0*, Morgan Kaufmann Publishers, Inc.(1997).
10) Naqvi,K. and Tsur,S.: *A Logical Language for Data and Knowledge Bases*, Computer Science Press, New York(1989).
11) Sun microsystems Inc.: *Java(tm) Transaction Service (JTS)*, URL http://java.sun.com/products/jts/.
12) Sun microsystems Inc.: *The Forest Project: PJama: Orthogonal Persistence for Java*, URL http://www.sunlabs.com/research/forest/COM.Sun.Labs.Forest.doc.external_www.PJava.main.htm
13) Atkinson,M.P., Daynes,L., Jordan,M.J., Printezis,T., and Spence,S.: An Orthogonally Persistent Java(tm), *SIGMOD Record*, Vol.25, No.4, pp. 5-10(1996).
14) Schneider,M.: *Java / Databases*, URL http://www.rhein-neckar.de/cetus/oo_db_java.html.
15) Connolly,D.: *Extensible Markup Language (XML tm)*, URL http://www.w3.org/XML/.
16) Wood,L.: *Document Object Model (DOM)*, URL http://www.w3.org/DOM/.
17) Robie,J., Lapp,J. and Schach,D.: *XML Query*

Language (XQL),
URL http://www.w3.org/TandS/QL/QL98/pp/xql.html.
18) 吉川 和巳, 皆川 和史: オブジェクト指向DBをJavaで学ぶ, 知る, 使う, Java WORLD, Vol.2, No.9, pp. 68-91(1998).
19) 吉川 和巳: 徹底検証! エンタプライズ Java ソリューション, Java WORLD, Vol.2, No.12, pp. 34-71(1998).
20) Barry,D.K.: *The Object Database Handbook*, John Wiley & Sons, Inc.(1996).
21) ルーミス, メアリー E.S.: オブジェクトデータベースのエッセンス, アジソンウェスレイ・トッパン (1996).
22) Hueng,P.: *Comparison of four ooDBMS('98)*, URL http://www.icslab.agh.edu.pl/~jasper/oodb/.
23) Amirbekian,V.: *Comparison of O2, Objectivity, ObjectStore, Versant*,
URL http://galaxy.uci.agh.edu.pl/~vahe/products.htm
24) Chaudhri,A.B.: Experiences using object data management in the real world, *SIGMOD Record*, Vol.27, No.1, pp. 5-10(1998).
25) Chaudhri,A.B. and Loomis,M: *Object Database in Practice*, Prentice Hall, Inc.(1998).

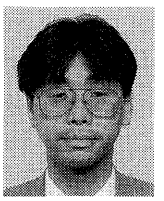
(平成 1999 年 3 月 20 日受付)
(平成 1999 年 6 月 27 日採録)

(担当編集委員 横田 一正)



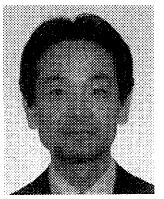
鬼塚 真 (正会員)

平3東工大・工・情工卒. 同年NTT入社. オブジェクト指向データベースの設計・性能評価, マルチメディアデータの検索モデル・管理システムの研究に従事. ACM 会員.



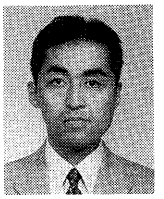
大場 克哉

平1阪大・文・美卒. 同年オージス総研入社. マルチメディアを利用した知的 CAI システムの設計・開発, オブジェクト指向データベース等, オブジェクト指向開発環境の利用に関連するコンサルティングに従事.



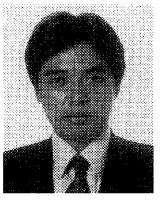
小島 秀登

平1慶応大・理工・電気工修士了. 現 (株) 日本総合研究所勤務. マルチメディアシステムの画像, 音声関係処理, ドライバ等の開発従事.



斉藤 信也

昭60富士ゼロックス情報システム入社. 平6オブジェクト デザイン ジャパン入社. オブジェクト指向DBMSの技術支援及びマーケティング活動に従事.



高橋 肇

昭57日大・文理卒. 同年日本電子計算入社. 平2より, オブジェクト指向データベース関連ビジネスに従事.