ソフトウェア工学 140－17
（2003．3．7）

トランザクションモニタとしてのJ2EEの評価

今城哲二　尾花学　小川秀樹
日立製作所 ソフトウェア事業部

本論文では，J2EE™について次の議論を行っている．
（1）Javaの使用頻度が増えてきており，5年後には日本全体で15%～20%の割合になると予想している．それを可能にするためには，Javaのビジネスアプリケーション基盤であるJ2EEの課題を解決する必要がある．
（2）現在多くのビジネスアプリケーションはメインフレームのOLTP環境で稼動している．OLTPとJ2EEを比較することにより，何がJ2EEの課題であるか指摘できるはずである．また，J2EEは，OLTPの主流の方式であるCICS方式とIMS方式の中で，前者に類似している．この長所・短所も課題を示唆するであろう．
（3）OLTPの役割や機能は，AP間連携，安定性能の確保，信頼性の確保，運用性・保守性の確保の4つに集約できる．これらとJ2EEの比較をサーベイ中である．入力キュー，出力キュー，ジャーナル，スケジューリングなどの課題が見つかっている．J2EEにOLTP機能を持たせるやり方として，J2EEの機能拡張以外にJ2EE/OLTP連携方式がある．
（4）Java成功の要因はJava VMの存在である．しかし，VMがJavaの構造上の問題を引き起こしている．メモリリークなどがその代表である．Unicodeも，Javaの仕様である．これら，構造上あるいは本来の仕様は，長所であり短所ともなっている．これは“Write Once. Run anywhere.”の“コスト”である．

1. はじめに

ビジネスアプリケーション業務をJavaで開発するケースが増えている．例えば，ある大手システムベンダのSIでの顧客への納入量（ソースプログラムの行数ベース）をみると（表1），全体からみたJavaの利用割合は，2000年度は1%，2001年度は5%と増えている．2002年度も10%までには至らないが相当増える見込みである．中小ベンダや一般ユーザでは，大手システムベンダと比べCOBOLやVisual BASICの使用頻度が高く，そこではJavaの普及は表1の数値より相当下回っているが，表1は日本でのJavaの使用状況の実態と今後のトレンドを示唆する数値として参考になる．

今後プログラム言語の使用状況がどのように変わっていくか，公表されている具体的な数値が少なくその根拠を明確に説明するのは難しいが，我々は日本全体の現状と5年後は表2のように想定・予想している．5年後のJava利用予想は全体の15～20%である．現場でのプログラミングは開発よりも保守が多い．表1や表2には保守時のプログラミング行数も含まれる．そこではCOBOLが多く使われているので，相対的に新規開発での COBOL の割合は小さくなる．すなわち，5 年後の新規開発での Java の割合は 15~20%よりもある程度大きくなると予想できる．

Java のビジネスアプリケーション業務のプラットフォームは J2EE（Java2, Enterprise Edition）である．表 2 の予想のように Java すなわち J2EE が本格的にビジネスアプリケーション業務で使われるためには，多くの課題が存在するはずである．現在，Java を用いた業務の構築・保守現場では多くの苦労をしており[1][2]，この苦労が今と変わらず 5 年後も続くならば全体の 15~20%という増加予想の実現は不可能である．本論文の問題意識は，ビジネスアプリケーション業務が長い間，多くは今でも，メインフレームの中

表1　ある大手システムベンダのプログラム開発量
（2000~2001年度，顧客に納入した行数）

プログラム言語	使用比率	備考
COBOL(PL/I)を含む	50%前後	横ばい
C/C++	15～25%	横ばい又は減少傾向
Visual Basic	15%前後	横ばい又は減少傾向
Java	1%→5%	増加傾向
その他（4GL，…）	10～15%	

表2　プログラム言語使用状況（日本全体）

プログラム言語	現在の推定	5年後の予測（2007年）
COBOL（PL/Iを含む）	60～70%（PL/Iは1～数%）	40～50%（PL/Iは減少）
Java	3%以下	15%～20%
Visual Basic と C#(.NET)	20%台（Visual Basic）	20%～30%（.NET上のVBとC#）
C/C++	10%以下	数%
その他（4GL，…）	10%以下	10%以下

で稼動してきたということから，メインフレーム技術と比較することにより，J2EEの問題点とそれを解決する課題が発見できないかという点にある．メインフレームのビジネスアプリケーション業務は，バッチ及びオンライン業務として実行されており，後者はOLTP（Online Transaction Processing）システムとして結実している．このシステムの管理プログラムをトランザクションモニタという．まだJavaでのバッチ業務はほとんど使われていないので，本論文では，オンライン業務についてJ2EEとOLTPとを比較し，J2EEがOLTPから学ぶべきことがないかを検討する．

本論文の構成は次のとおりである．第2章では先ずJava成功の要諦となった事項とそのトレードオフについて説明する．次にJ2EEの構成についてメインフレームのソフトウェア群の構成との対応を示し，それらが外形的には類似していることを説明する．第3章では，安定性能や信頼性の確保などOLTPはどのような設計思想に基づいて設計されているかについて議論する．第4章で，OLTPと比べたJ2EE問題点を指摘する．第5章では，それら問題の改善施策として，OLTPとJ2EEの連携方式について述べる．第6章と第7章で，関連研究とまとめを述べる．

2. Java技術とメインフレーム技術との対応

2.1 Java成功の要諦とVM採用のコスト

Java言語は，当初は組込みシステムの記述用のプログラム言語として設計されたので，C++をベースにしているがそれを大幅に簡潔にしかつ安全にした言語となっている．すなわち，次の式がJavaを適確に表現している．αの基調が“簡潔”で，βの基調が“安全”である．Java言語はその後あまり拡張されていないので，この良き性格は保持されたままである．この簡潔と安全が，Java成功の第一の要諦であった．

$$\text{Java} = \text{C++} - \alpha + \beta$$

C++から削除・追加された代表的な言語要素は，削除はポインタ演算，GOTO文，多重継承，文字列のNULLターミネータで終わる文字配列などであり，追加はガーベージコレクション，1ソース1公開ファイル，文字列（string）オブジェクトなどである．

Java成功の第2の要諦は，クライアントでプログラムが必要時にロードするというアプレットの存在であった．Java生誕時は，当時注目を浴びていたクライアントサーバシステムでは，多くのクライアントにそれぞれプログラム配布が必要であり，その運用が大変という欠陥が実感されてきたときだったので，この機能はプログラム配布で疲れていた運用管理者の人気を集めた．なお，現在この問題の解はWeb環境である．

Java成功の第3の要諦で，Javaの最も本質的特徴は，コンパイラの目的プログラムを中間語コードのバイトコードにし，目的コードの実行はVM（Virtual Machine）で行うことにしたことである（図1）．これにより，Java VMを搭載したハードウェア／OSでは，原則としてリコンパイルすることなく，バイトコードの実行が可能となり，“Write once. Run anywhere.™”が実現できた．

この第3の要諦のトレードオフとしてコストがかかるという問題がある．第一のコストは，VMではインタプリタでプログラムが実行されるので“実行時間がかかる”ということである．これを軽減する技術として，JIT（Just in time）コンパイラやHOTSPOTという技術が出現した．また，ハードウェア性能の向上も顕著で，通常の（そう負荷の高くない）ビジネスアプリケーションでは性能が問題になることは少なくなった．しかし，実行時の動作がVMが中核となっているので，“トラブル時の原因究明が難しい”という，第2のコストがJavaをビジネスアプリケーションの基幹系業務で使うときの深刻な問題としてクローズアップしてきている．

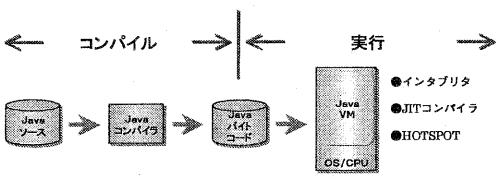

図1 JavaコンパイラとJava VM

2.2 J2EEの構成とメインフレームとの対応

Java技術はWeb環境の普及にともない，アプレットからサーブレット，JSP，さらにEJBと，クライアントサイドの技術から次第にサーバサイドの技術に発展し，J2EEとして集大成された．これを，3階層に分けて示したのが図2である．時間軸としては，図2の左から右に段階的に発展した．この結果，J2EEでのアプリケーションの実現方式として複数の方式（例えばDBにアクセスする方式として，EJBを介するやり方と介さないやり方）が存在し，システム設計の自由度が高いという（長所及び短所）をもたらした．この図2に対応するメインフレーム業務を図3に示す．こ

れは，ビジネスアプリケーションの典型的なオンライン業務の図である．すなわち，J2EEはビジネスアプリションのオンライン業務のプラットフォームとなることを目的に考えられたもので，ほぼ実用段階に達している．しかし，後で説明する問題点もかかえており，今後の発展もJavaの仕様制定団体JCPで検討されているので，J2EEの将来性を強調するときには，“未完の大器”と呼ぶこともある．

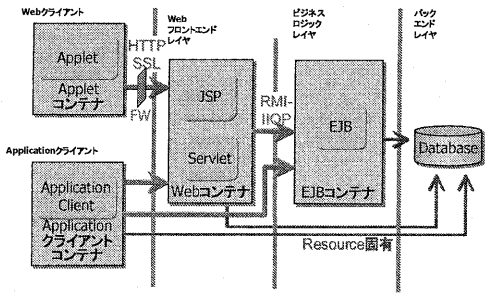

図2　J2EEの3階層モデル

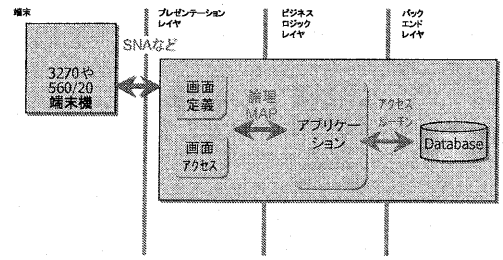

図3　オンライン業務の3階層モデル

レイヤ		メインフレームでの例	J2EE(Java™2 Platform, Enterprise Edition)	
アプリケーション開発	画面	XMAPなど 定義ツール	JSP(JavaServer Pages)	
	言語	COBOL, PL/I	Java [言語]	
データベース		IMS, DB2, XDMなど	JDBC(Java Database Connectivity) 各種オープンDBMS(HiRDB, Oracleなど)	EJB
オンライントランザクション		IMS, CICS, DCCM, XDMなど	JTA(Java Transaction API) JTS(Java Transaction Service)	EJB
通信基盤		通信管理	Java IDL(Java Interface Definition Language) Java RMI(Java remote Method Invocation)	EJB
OS	システムコール	スーパバイザコール	Javaクラスライブラリ	
	カーネル	MVS, VOS3など	Java VM (Intel Pentium, SPARC, Power, PA-RISCなど各種CPU)	
ハードウェア		3x0シリーズ, Mシリーズなど		

表3　メインフレームと対応させたJ2EE構成要素

表3に，メインフレームと対応させたJ2EEの構成要素を示す．このような対応がつくということは，機能的にはビジネスアプリケーションのオンライン業務がJ2EEでも可能であることの別証であるといえる．この表ではメインフレームのOSに，Javaクラスライブラリと Java VMを対応させている．これは，Javaの世界ではOSやハードウェアを隠蔽し，ユーザアプリケーションからはすべて自分自身のインターフェースで見えるようにしていることを意味している．

2.3　J2EEのCICS方式との類似性

OSを隠蔽するアプローチは，OLTPとして著名なIBMのCICSに似ている．それと対照的なのは，IMSである．日立では前者の方式をTMS-3V，後者はXDMやTMS-4V/SPが採用している．

CICS方式とIMS方式の相違は，オペレーティングシステム置き換え（replace- the-operating-system）アプローチと，オペレーティングシステム利用（use-the-operating-system）アプローチといわれている[3]．IMS方式では，OSのプログラム制御やスケジューリング機能，メモリ保護機能，ファイル機能などはそのまま使用しており，異なるトランザクションのAPが同時に実行されるときは異なるアドレス空間で実行される．一方，CICS方式では一つのアドレス空間の中で複数のトランザクションのAPをマルチタスク（メインフレームのOS用語でマルチプロセスとほぼ同義）で同時に実行する．それらの同時実行APを整合性をとって管理するために，APからOS機能を直接使用するのは禁止しており，CICS方式が提供するOSの代替機能のみ使用するようにしている．

CICS方式の欠点は，APの不良でOLTP管理プログラム（トランザクションモニタ）や別トランザクションのAPのメモリを破壊したときや，APが誤ってOS機能を直接使ってしまったときのトラブルシュートが難しいことである．これは，Java VMにも当てはまる構造的な問題である．また，ユーザの設計の自由度はIMS方式と比べCICS方式の方が大きいので，設計者の技量の差が大きくなり，オンラインシステム構築の難易度がIMS方式よりも高い．CICS方式の長所もいろいろあったが，CICS方式の構造的な問題と構築が比較的難しいことが主な要因で，日本のメインフレームではIMS方式の方がCICS方式よりもSEや顧客に高く評価されるようになった．大手銀行の第2次勘定系オンラインシステムまでは，当時(1970年代後半)主流のOSが多重空間をサポートしていなかったこともあり，CICS方式の採用が一般であった．多重空間

OSが当り前となった1985年前後に開発された第3次オンラインシステムでは，IMS方式の方がより安全ということと，クロスメモリ機能などOSの性能改善の努力などで，大手銀行の大部分はIMS方式を採用した．

3. OLTPの設計思想

本章では，メインフレームのOLTPだけでなく，UNIXやPCサーバなどのオープンシステムを含めた，現代が必要とするOLTPの一般的な設計思想について議論する．

3.1 OLTPの概要

OLTPは，オンラインシステムを構築するための性能・信頼性・スケーラビリティ等の高度なシステム要件を満たすための各種機能を提供している．その中でもOLTPの最も重要な役割は，単純なクライアントサーバシステム（CSS）やWebシステムと違い，安定した応答性能を確保するところにある（図4参照）．つまり，

・トラフィックが一時的に過負荷状態になっても，安定的なレスポンスを保証し
・限られた資源で最大限のパフォーマンスを出すことを実現するもの

がOLTPである．逆に言えば，よく制御されていないトランザクションシステムは，

・トラフィックが一時的に多くなると，スラッシングなどによるオーバヘッドでパニック状態になってしまう
・ユーザ数（端末数やクライアント数）がある規模以上になると必要となるプロセスやメモリが膨大に必要となりシステムとして成り立たなくなる

などの問題があると考えられる．

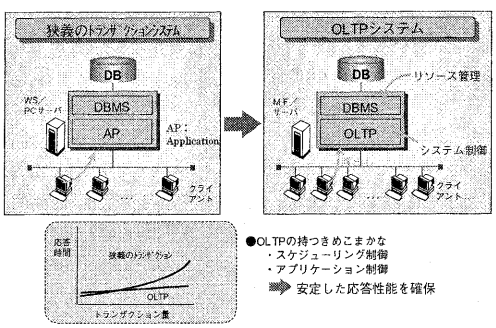

図4　OLTPによる安定した応答性能の確保

3.2　OLTPの役割

OLTPの役割や提供されるサービスは，OLTP製品ごとのシステム稼働環境や考え方によって力点の所在に相違があるが，本質的には次の点に集約される．

・　AP（Application）間連携
・　安定性能の確保
・　信頼性の確保
・　運用性・保守性の確保

しかし，各々の役割，機能の重要さの重みが変わると設計思想が異なってくる．そのため，各々の実現方式に変化または追加がでてくる．ときにはまったく異なっているように見えても，他の環境からみれば，似たような機能ということも多い．ここでは，できるだけ本質的なものについて，それぞれ整理してみる．

3.2.1　AP間連携

AP間連携はシステムの分散化，ネットワーク化に伴い，重要な要素となってきている．ここでは，大きく分けて，次の2点のサポートが必要になる．

(1) ＡＰ間通信

同一OLTP内，あるいは，異なるOLTP間でのAP連携は，同期型，非同期型など，いろいろな接続形態はあるが，RPC（Remote Procedure Call）など基本的には下位のプロトコルとは独立のインタフェースを提供するとともに，リカバリやスケジューリングなどOLTPの特徴的機能が必要となる．たとえば，図5のような接続形態の異なる２つのアプリケーション間でのデータの同期保証などである．

(2) 多種プロトコルサポート

システム環境，ネットワーク環境に応じて各種多様なプロトコルのサポートが必要になる．個別にアプリケーション毎にサポートしていたのではたまらないため，接続性及び開発・運用のしやすさが重要である．プロトコルでは，国際標準や業界標準などなんらかの

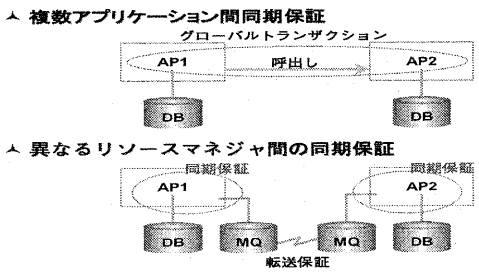

図5　OLTPでの複数AP間の同期保証

標準に従わなければないないことが多い．

3.2.2 安定性能の確保

いくら性能が良くても，ある規模を超すと急に性能が悪くなったり，どうなっているか判らないといった状態にならないことが重要である．そのために，次の２点が必要である．

(1) 安定した性能保証

クライアントから見て，APのスレッドやプロセス，ノードを共有するため，ロードバランス，キュー優先スケジュール，ＡＰ起動・停止管理などの機能を持つスケジューリング機能により，瞬間的なトラフィック変動による性能悪化，スラッシングを防止し，偏差の少ない安定性能を提供する（図６参照）．

また，このようなきめ細かな制御で安定した性能が可能になることで，システム設計時に性能コストやシステム構成を見積もることが可能になる．

(2) ある程度以上の負荷を想定した処理構造

負荷が増大したとき，それに比例して増加するシステム制御のオーバヘッドをできるかぎり排除した高性能化を実現する．具体的には，以下のような処理が考えられる．

・ＡＰやＤＢなど資源の初期化処理の事前実行
・トランザクション同期処理の一括化
・排他処理の改善

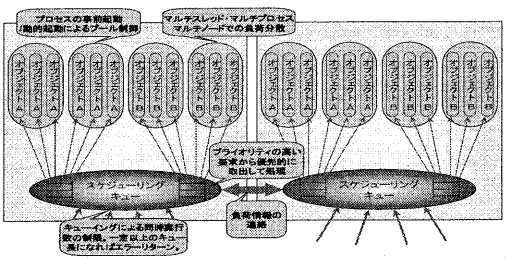

図６ OLTP の安定性能を実現するための施策

3.2.3 信頼性の確保

障害はハードウェア，ＡＰ，システム，データ，ネットワーク，クライアントなど多岐にわたるが，それぞれのレベルで信頼性機能(RASIS)が必要になる(図７)．

(1) Reliability

品質も含めて，とにかくダウンしないこと．誤操作，ネットワーク障害，ＡＰミスなどでダウンとならないようにする．

(2) Availability

タイムアウト監視などによるすばやい異常検知と，二重化（多重化）制御や特定サービスの閉塞により，異常部分だけを切り離して，決してオンラインシステム全体をストップさせたりしないことが重要である．また，その後障害が回復したら，二重系を復帰したり閉塞を解除して，元の状態に戻ることができる必要がある．

不幸にしてシステムダウンしてしまったら，とにかく早く回復してサービスを再開できるようにする必要がある．このためOLTPは，一定間隔ごとにシステムの状態をチェックポイントとして取得したり，待機システムを用意しておき，障害と同時にそちらに切替える方式（ホットスタンバイ）を採ったりする．

(3) Integrity：データの保全／整合性．同期点管理

データの保全，整合性のため，同期点管理，通番管理などによる回復が重要である．DBのデータについては，DBMSが主な役割を担うが，OLTPではDBMSを含めたシステム全体のIntegrityを制御をする．

(4) Security

オープンなネットワーク環境では，不当なアクセス防止や，データの機密保護が重要である．

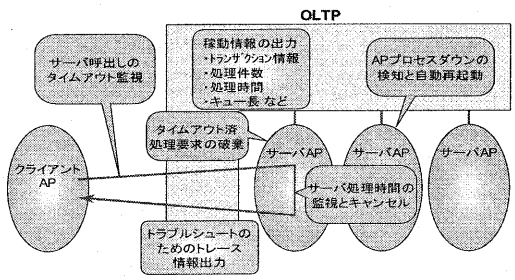

図７ OLTP の信頼性機能（RASIS）

3.2.4 運用性，保守性の確保

一定規模以上のシステムでは，全部を見渡せるプロフェッショナルなシステム運用者は，ほとんど期待できない．それを前提として，以下の運用，保守機能が必要である．

・ 容易かつ柔軟なシステム定義
・ 自動運転
・ 動的構成変更（特に24時間運転対応）
・ トラブルシュート機能

・ ＡＰメンテナビリティ（バージョン管理等）
・ 各種統計情報，トランザクション履歴の取得
・ バッチを考慮した運用サポート

4 J2EEのいくつかの問題点

4.1 OLTPとの比較による問題点

第3章で述べたOLTPの設計思想と比べ，そこで述べた項目（要件）がJ2EEでどう実現されているかについて，二つ例を示す．

(1) 入力キュー，出力キュー，ジャーナル

OLTPの性能安定と信頼性確保の基本機能ともいうべき，入力キュー，出力キュー，ジャーナルの機能がJ2EEでは定式化されておらず，それらはそれぞれのベンダでの機能追加あるいはユーザでの運用に委ねられている

(2) スケジューリング機能

J2EEではリクエストの受付窓口が単一のためプライオリティ制御ができないなど，スケジューリング機能がOLTPと比べ弱体である．この結果，リクエストが増加したときに，到着頻度の低いリクエストが実行されなくなる可能性がある（図8）．

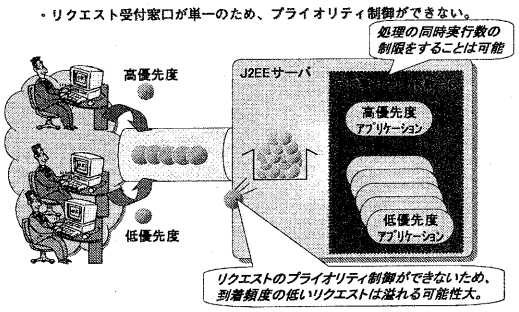

図8 複雑なスケジューリングが困難

これらの例のように，一つ一つの項目について議論していくべきだが，まだその観点からのJ2EEのサーベイは完了していない．現段階でいえることは，J2EE全体はオンライントランザクションモニタであるという考え方では統一的には設計されておらず，短期間で個々の機能仕様が設計され，それらを積み上げてJ2EEとして集大成されたということである．

このため，ユーザがJ2EEを使用するとき，アプリケーションのレベルに応じ，より上位にそれらの機能の利用ガイドとベストセレクションが必要となっている．その現段階のJ2EEの進化に対応した表現が，現在脚光を浴びている百花繚乱のフレームワークである．フレームワークを大別すると，サーブレットとJSPでの画面処理を中心としたものと，よりEJBを意識したものに分かれる．後者を発展させることにより，OLTPの思想を中核としたフレームワークが実現でき，何がJ2EE自身に欠けているかが解明されるはずである．それら不足機能のJ2EEでのサポートとフレームワークで実現した機能が一体化され，ミッションクリティカル向けのJ2EEの機能として一つのミドルレイヤを構成しることになるだろう．

4.2 構造的あるいは本来の問題点

第2章でもいくつか指摘したが，本節でも，いくつかのJavaの構造的あるいは本来の問題点を指摘する．なお，ここで指摘した事項は，見方を変えればJavaの長所ともいう事項であるので，長所のトレードオフとしてコストとして受け入れ（，場合によっては非常に高いものになることを覚悟して）Javaを利用するか，あるいはリスクが高すぎるのでJava自体の適用をあきらめるかの決断が必要である．

(1) Java VM内のメモリ所要量

実行アプリケーションによってメモリ所要量やライフサイクルが異なるため，一般にメモリ使用量の限界値を管理するのは困難である．このため，最終的にメモリ不足になったときは，システムが停止してしまう．システムのメモリ状態がダイナミックに変わるので，定量的な評価が難しく，トラブル再現も困難で，原因究明と対策に多大な時間をかかったという悪夢の経験を多くのベンダがしており，その経験の中から回避するノウハウを次第に積み重ねている．ただし，完全な対策は今のところない．

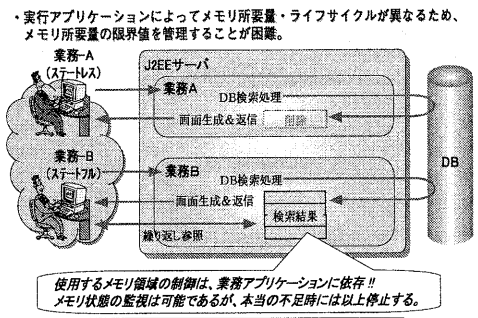

図9 Java VMでのメモリ使用量はアプリケーション依存

(2) インタプリタ形式での実行

Javaで記述されているアプリケーションのコンパイル結果のバイトコードを，Java VMが逐次解析し，実行するので，マシン語を生成している言語と比べ一般に性能は遅い．ただし，VM実行中に繰返し実行される部分が検出されるとそれはHOTSPOTとみなされ，JITコンパイラによりCPUに対応する実行コードが生成され，その部分の実行性能は大幅に改善される．このHOTSPOTでJITコンパイラが動くコストと改善された効果は経験的に，これがないときと比べシステム全体の性能は相当程度良くなっていることが多い．しかし，JITコンパイラが動いているときはそれに遭遇したリクエストは待たされてしまい，平等に安定した性能を実現というわけにはいかない．

(3) 実行時の文字コードはUnicode

Javaアプリケーションが扱う文字コードはUnicodeである．外部の文字コードと異なる場合，文字列を入力あるいは出力するたびにコード変換が多発する可能性がある（図10）．外部の文字コードがシフトJISやEUCなどUnicode以外のときは，その変更が困難なDBや入出力の事例が多いので，性能劣化の原因となることがしばしば起きている．また，変換時の文字化けや外字の問題なども発生している[4]．

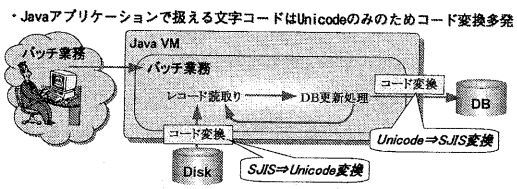

図10 Java実行時の文字コードはUnicode

5. J2EEとOLTPの連携方式

J2EEに不足しているOLTP機能を追加拡張するには，一つ一つその追加拡張の正当性と必要性を明確にし，仕様制定団体のJCPに問題を提起し，そこで正当性と必要性が認められてから，使用案を提示し，さらに審議・承認という過程が必要である．このやりかたでミッションクリティカルなビジネスアプリケーションに必要な機能の追加が始まっている．

ここでは，それとは異なるOLTP機能の利用方式を紹介する．

一つは，J2EEの1コンポーネントのJTA（Java Transaction API）の実装にOLTP技術を用いるやり方である．日立ではJTAのトランザクションモニタの実装に，国内で著名なOLTPであるOpenTP1技術をベースにしたTPBrokerのOTS機能を採用している（図11真中のJTAのボックス参照）．しかし，この方式はあくまでも下部構造を支える技術であって，ユーザからみえるJTAのAPIが変わるわけではない．

もう一つの方式は，J2EE1.3で追加された外部システムとの接続方式であるJCA（Java Connector Architecture）を用いて，OLTPシステムと接続する方式である．日立のAPサーバーであるCosminexusはJCAを介してOpenTP1のサーバー機能と接続している（図11右側下のJCAのボックス参照）．この接続の詳細は図12に示す．ここではCosminexus側にJCAとインターフェースを持つCosminexus OpenTP1 Connectorをサポートし，それがOpenTP1のサーバプログラム（SPP）と交信する．この接続により，J2EEで構築したシステムは，OpenTP1のクライアントとして動作することになる．この関係を示したのが図13である．

なお，現段階ではミッションクリティカルシステム（銀行勘定系システムなど5分以上システムダウンすると新聞などで報道される社会活動に影響度の高いシステムや，社内システムであっても24時間生産管理システムのように自社の存亡に影響の強いシステムなど）にJ2EEを使うのは時期尚早である．それらのシステムとで，オープンシステムでWeb環境を使用して実現するときは，図13のような使い方を推奨する．

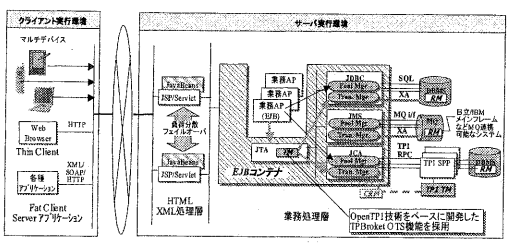

図11 J2EE と OLTP の接続

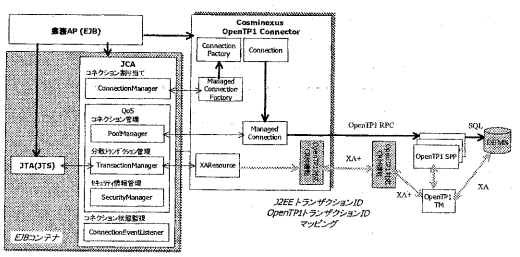

図12 J2EE と OLTP の接続の詳細

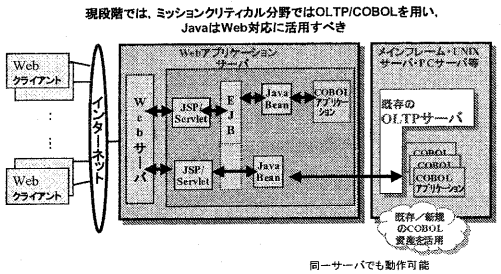

図 13　OLTP と J2EE の連携推奨パターン

6. 関連研究

Javaについての文献は多数あるので本論文では取り上げない．J2EEとメインフレームとの機能比較や，J2EEとOLTPを比較したものはまだ見当たらない．

OLTPは本来DBと並んでソフトウェア科学の基本分野であるが，ベンダの製品とユーザでの利用との相乗効果によって進歩してきたので，研究論文や研究書は少なく，実質的な基本文献は各ベンダの製品マニュアル，機関誌掲載論文あるいは社内文書である．これらは一般には参照が難しい．GrayとReuterの分厚いトランザクション処理の著書[3]は，1990年代はじめまでのこの分野の技術・研究・製品を総括した名著で文献リストも詳細である翻訳も発行された．[5]と[6]は，日本語の著書で，OLTPの概念とオープンシステムのOLTPについて言及している

7. むすび

本論文では，J2EEについて次の議論した．

（1）Javaの使用頻度が増えてきており，5年後には日本全体で15%～20%の割合になると予想した．それを可能にするためには，Javaのビジネスアプリケーション基盤であるJ2EEの課題を解決する必要がある．

（2）現在多くのビジネスアプリケーションはメインフレームのOLTP環境で稼動している．OLTPとJ2EEを比較することにより，何がJ2EEの課題であるか指摘できるはずである．また，J2EEは，OLTPの主流の方式であるCICS方式とIMS方式の中で，前者に類似している．この長所・短所も課題を示唆するであろう．

（3）OLTPの役割や機能は，AP間連携，安定性能の確保，信頼性の確保，運用性・保守性の確保の４つに集約できる．これらとJ2EEの比較をサーベイ中である．入力キュー，出力キュー，ジャーナル，スケジューリングなどの課題が見つかっている．J2EEにOLTP機能を持たせるやり方として，J2EEの機能拡張以外にJ2EE/OLTP連携方式がある．

（4）Java成功の要因はJava VMの存在である．しかし，VMがJavaの構造上の問題を引き起こしている．メモリリークなどがその代表である．Unicodeも，Javaの仕様である．これら，構造上あるいは本来の仕様は，長所であり短所ともなっている．これは“Write Once. Run anywhere.”の“コスト”である．

最後に，次の2つの著作物の出版が必要であることを主張して，本論文のしめくくりとする．

・Java VMの動作を分かり易く解説した日本語の著書．これがあればJava技術者全体のトラブルシュートの水準向上に役立つ．

・日本では，緑の窓口システムや銀行勘定系システムなどそこで使われているOLTPともども世界有数のシステムが実用化されているのに，それらがきちんと記述された書籍がないのは残念である．それらの開発の中心となった方々が次々と退職している．これらの日本の価値ある技術の記録を留め，そのノウハウが次代の技術者に伝承されるために，早急に“代表的な実用オンラインシステムとその基盤のOLTPについての技術報告書”を記述・発行するプロジェクトの発足が望まれる．

文　献

[1] J2EEのトラブル回避，日経オープンシステム，No.117，pp.138-161（2 2003年1月号）．

[2] EJB™コンポーネントに関するコンソーシアムパフォーマンス研究会編：パフォーマンスに関するHints & Tips集，http:www.ejbcons.gr.jp（2003年3月公開予定）．

[3] J. Gray & A. Reuter: Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, Inc. (1993). ・・・ 翻訳は，喜連川優（監訳）：トランザクション処理 概念と技法（上・下2巻），日経BP社（2001年）．

[4] 風間一洋：国際化と日本語処理，アスキー(2000)．

[5] 渡辺榮一，J.グレイ他：OLTPシステム――オンライントランザクション処理――，近代科学社(1995年）．

[6] 渡辺榮一編：オープンOLTPシステム入門，日経BP出版センター（1995年）．