# オープン系ミッションクリティカル業務構築を支援するMIDMOST技術基盤

## Technical Bases of MIDMOST, Supporting Mission-critical Business Application Building on Open Architectures

馬場定行, 宮野将彰, 佐藤則之

**要　約**　MIDMOSTはWindows 2000 DatacenterServerおよびAdvancedServer上にて稼働するミドルウェアであり，ミッションクリティカル業務構築の支援を目的としたシステムである．このシステムは将来的にはプラットフォームフリーでアプリケーションを稼働させるという考えに基づいて設計されている．これにより今後も発展するオープン化技術に対し，アプリケーションの改修を最小限に抑制した上で継続的に最新技術を取り入れられる環境を提供している．

本稿では，このMIDMOSTの特徴および機能について，その主な技術基盤を説明する．

**Abstract**　MIDMOST is a middleware which works on Windows 2000 Datacenter Server or Advanced Server, and aims at the support of mission-critical business application development.

This system is designed based on the thought that in the future an application will work without platform dependency.

MIDMOST provides the environment that the latest technology is adopted continuously while an application needs little modification.

This paper explains main technological bases of the features and functionalities of MIDMOST.

## 1. はじめに

MIDMOSTはWindows 2000 DatacenterServer（以下，DCS）およびWindows 2000 AdvancedServer（以下，ADS）上にて稼働するミッションクリティカルなアプリケーション構築の支援を目的としたトランザクション制御ミドルウェアである．MIDMOSTは，基本的なトランザクション制御に加え，業務処理ロジック構築支援やシステム運用・保守支援を重視したミドルウェアであり，使用者に対しオールインワンで機能提供する．

本稿では，MIDMOSTが提供する機能のうち特に，リソース制御機能，通信制御機能，障害追及支援機能について記述する．

## 2. 概　　要

MIDMOSTがアプリケーション開発者にとって有用なミドルウェアの条件として考慮されている点は，OS/DBMS/通信ソフトウェアなどに対し統一的なアプリケーションインタフェース（API）を提供することと，マルチスレッド制御，例外処理制御のようなプログラミングにおいて基本ソフトウェアへの依存度の高い制御部分を使用者から隠蔽し，高効率なリソース制御を行なうフレームワークを提供することである．APIについては基本的なトランザクション制御機能の他に，出力メッセージ保存・回復・照会機能，タイマスケジュール機能といったトランザクション処理補完の機能や，日数計算（営業日算出，月初月末日算出等），Win 32 APIをラッピングした関数などのAP共通ルーチンの機能，ログ出力や実行トレースなどの障

害追及資料採取の機能を有し，一般的な TP モニタにはない業務ロジック構築支援を行なう．障害追及資料の採取については，API 提供だけでなく，コーディングルールにより開発生産性・保守性の向上を考慮している．

また，通信制御における複雑な宛先管理や並列サーバ構成下でのロードバランス制御等について，最適なメッセージのルーティング機能はオープン環境下では必要な要件である．MIDMOST は，業務量の伸びによって，自由に業務処理を実行するサーバを追加・変更し，常に快適なサーバ環境を維持できるメッセージルーティング機能を提供する．システム運用の面においても，24 時間連続稼働を支援するプログラム入替え機能や，構成管理変更時の整合性・差分情報検査機能等をもつ．

図 1 に示す通り，MIDMOST のシステム構成要素はアプリケーション層に特化している．プレゼンテーション層，データベース層には，それぞれデファクトスタンダードなソフトウェアと豊富な補完ツールがすでに存在し，オープン系システムにおいてカスタマイズが容易な部分である．それに比較するとアプリケーション層は業務への依存度が高く，システム開発にかかる生産性と保守性が重視される部分である．MIDMOST はこの部分を広くカバーするためのミドルウェアとして位置づけられる．

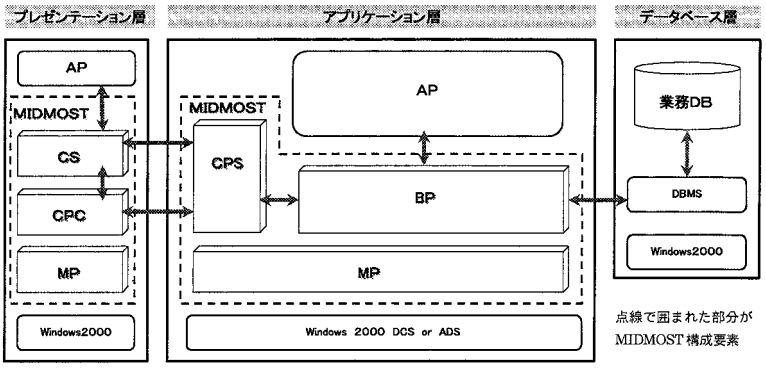


**図 1　MIDMOST システム構成要素**

1)　CS（Communication Service）

クライアントまたは WEB サーバ上のアプリケーションから呼び出されるインタフェースであり，MIDMOST への通信を行なうための API 群である．CS による接続は，2 種類の方式（直接サーバ内の CPS と接続する方式と，CPC を介して CPS と接続する方式）を選択できる．

2)　CPC（Communication Process Client）

クライアントまたは WEB サーバ上のアプリケーションに搭載される常駐プロセスであり，同一サーバ上で稼働する MIDMOST システムごとに 1 プロセス存在する．同一業務を複数のサーバで処理するような並列サーバ構成の場合，要求メッセージを負荷分散する機能や，多種の業務別サーバとの間でメッセージをルーティングする機能を保有する．

3)　CPS（Communication Process Server）

サーバに搭載される常駐プロセスであり，同一サーバ上で稼働する MIDMOST システムごとに 1 プロセス存在する．CPS は自ノード宛のメッセージを受信すると該メッセー

ジを処理する BP（後述）に引き渡し，さらに BP 内で処理された結果を受信すると，CS/CPC 向けに送信する．また CPS は他ノード宛にメッセージを送信する場合，CPC と同じく負荷分散機能やメッセージルーティング機能を保有する．

4） BP（Business Process）

サーバに搭載される常駐プロセスであり，同一サーバ上で稼働する MIDMOST システムごとに複数のプロセスを常駐させることができる．この BP は業務処理が動作するプロセスであり，使用者が DLL として実装した業務処理は，BP が提供している業務処理スレッド上で稼働する．

使用者は効率上の最適化，および業務の分割単位を考慮して BP の数を決定する．

5） MP（Monitor Process）

サーバに搭載される常駐プロセスであり，同一サーバ上で稼働する MIDMOST システムごとに 1 プロセス存在する．DBMS の動作監視，メモリテーブルのロック監視，タイマスケジュール・出力メッセージのリカバリ等の監視と制御を行なうほか，MIDMOST およびノード内のシステム稼働状況の統計情報採取を行なう．

## 3. 通信制御機能

MIDMOST はサーバ間の通信を行なうための機能を標準で提供している．本章ではメッセージルーティング機能について論理宛先管理を中心に記述する．

### 3.1 メッセージルーティング機能

クライアント・サーバ環境で，クライアント AP から業務処理を呼び出す時には，業務処理の物理的なサーバを認識して，そのサーバに対して接続を確立してから，通信を行わなければならない．しかし状況に応じては，複数サーバで業務処理の配置変更を行う必要がある．この場合はサーバの変更に応じた各クライアントの変更も必要となり，システム管理者の保守作業負荷は多大なものとなっている．これを解消するには，サーバの配置が変更されても，クライアント AP にその変更を意識させない仕組みが求められる．このためには，通信の宛先を，サーバという物理的なものから論理的なものに変える必要がある．つまり，「物理宛先と論理宛先の対応づけを行い，クライアント AP と業務処理との接続を確立する」ことが必要となる．この機能を実現したのが，CPS と CPC が提供するメッセージルーティング機能である．

### 3.2 CPS における論理宛先管理

前述したが各サーバ上には一つの CPS が動作している．サーバ A の CPS を CPS(A)，サーバ B の CPS を CPS（B）とした時，CPS（B）が既に正常起動しており，後から CPS（A）が起動した場合の各 CPS の動作を CPS（A）を中心に説明する．

1） CPS 起動時

図 2 に示す通り，CPS（A）は起動時に，ノード内のプロセスを管理するプロセス振分テーブルから稼働するプロセスに関する情報を取得し，それぞれのプロセスが持つ論理宛先を抽出して，自ノード論理宛先振分リストを作成する．そして，その論理宛先をキーとして，論理宛先の閉塞状態を管理する閉塞テーブルから，それぞれの論理宛先の閉塞状況を取得する．取得した自ノード論理宛先マップをブロードキャストにより他ノードの CPS

(B) に伝達する.

CPS (B) は，自身の持つ他ノード論理宛先振分リストに，受信したノードの情報を追加するとともに，自ノードの IP アドレス，自身が接続を監視しているポート番号，およびそのノード上の自ノード論理宛先マップを応答メッセージとして CPS(A)に送信する. CPS (A) は，自身の他ノード論理宛先振分リストに追加し，受け取った IP アドレスとポート番号を用いて接続を確立し,他ノード接続マップにそのソケット識別子を設定する.

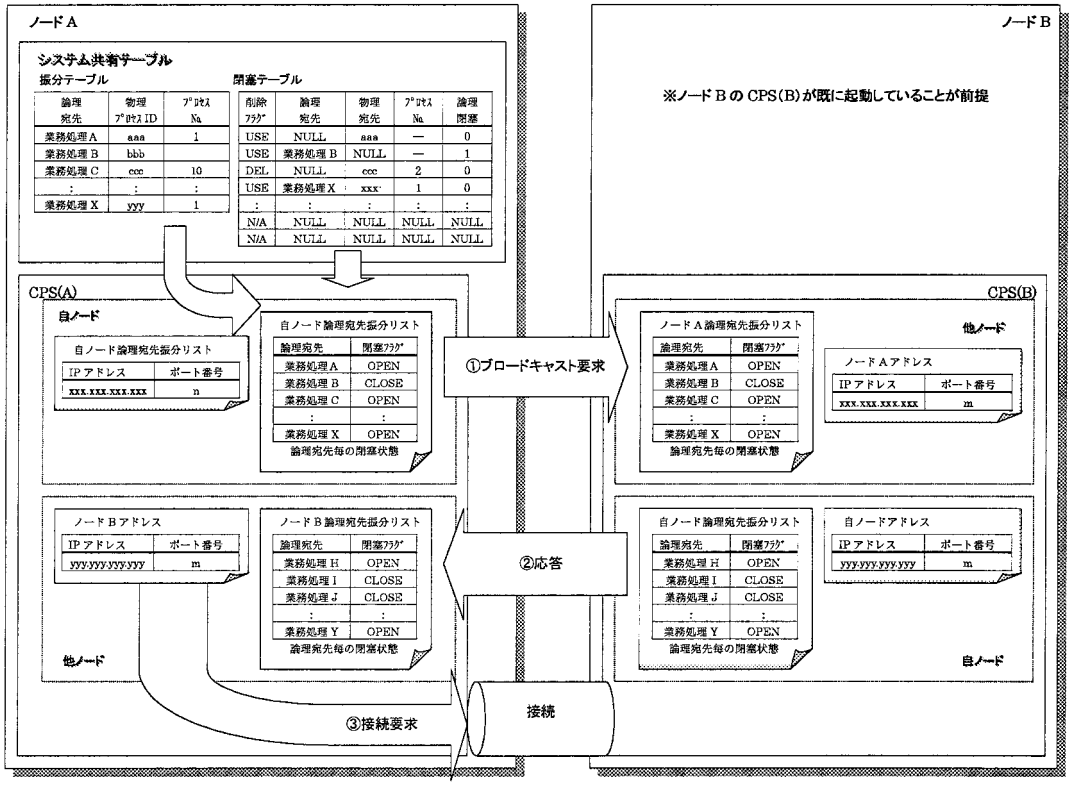


**図 2　CPS 間の論理宛先振分リストの配布**

2)　論理宛先状態変更時

業務処理の配置の変更が行われた場合，他ノード論理宛先振分リストも変更状況に応じて同期を取って更新されなければならないが，この維持管理は次のように行っている.

CPS は一定間隔毎に，プロセス振分テーブルと閉塞テーブルから論理宛先を抽出し，プロセス上に持っている自ノード論理宛先振分リストとの比較を行う．相違があったときは，自ノード論理宛先振分リストを更新するとともに，論理宛先の差分をブロードキャストで通知する．また接続している CPC に対しても論理宛先の差分を個別に送信する．差分情報を受信した CPS/CPC は，受け取った差分情報にしたがって，自身が持つ他ノード論理宛先振分リストを更新する．このようにして，各 CPS/CPC は他ノード上にある論理宛先の閉塞状況を維持している.

3)　CPS 終了時

また，他ノードにある CPS が終了したときには，その CPS との通信を行うスレッドが

終了を検知して，他ノード論理宛先振分リスト中の該当ノードのエントリを削除し，そのCPS とのソケット識別子を閉じる．一方，サーバ障害やネットワーク障害等は，通信が行われないと検知することができない．そのため，CPS はヘルスチェックメッセージを一定間隔毎に他ノード CPS に送信する．構成定義で指定されたヘルスチェックタイムアウト時間内に応答メッセージを受信しなかった場合は，サーバ障害やネットワーク障害が起きたものとして，他ノード振分マップ中のそのノードのエントリを削除し，そのノードの CPS とのソケット識別子を閉じる．このようにして，現在稼働中ではないノードに対して，メッセージを送信しないようにしている．

4） 論理宛先振分エラー時

このように，各 CPS は他ノード論理宛先振分リストの維持を行っているが，他ノードからのブロードキャストメッセージの受信もれや，他ノードからの論理宛先マップを受信する前に他ノード宛メッセージ送信の要求を受ける，などの理由で，送信メッセージに指定した論理宛先が，他ノード論理宛先振分リスト上に見つからないことが起こりうる．この場合は，そのメッセージの論理宛先を指定して，論理宛先取得要求をブロードキャストし，他ノードからの応答を一定時間待つ．待機時間内に他ノードから応答があった場合には，他ノード論理宛先振分情報を更新し，メッセージを送信する．もし時間内に他ノードから応答がなかった場合は, ユーザが不正な論理宛先を指定したエラーとして処理をする．

### 3.3 CPC における論理宛先管理

CPC も CPS と同様にルーティング機能を提供するが，サーバと異なるセグメントにあるクライアント上で動作させるので，ブロードキャストメッセージが届かないことが想定される．このため CPS とは異なり次のような動作を行う．

1） CPC 起動時

CPC は起動時に，接続する CPS の IP アドレスを取得し，自身が監視している UDP ポート番号を送信する．そして，メッセージを受信した CPS は，受信した UDP のポート番号を設定する．自ノード論理宛先振分リストメッセージを受信した CPS と同様に，自ノードの IP アドレス，自身が接続を監視しているポート番号，およびそのノード上の自ノード論理宛先振分リストを応答メッセージとして CPC に送信する（図 3）．

2） 論理宛先振分エラー時

CPC は，稼働しているすべての CPS と接続しているとは限らない．したがって，メッセージ送信の要求を受けた CPC は，指定された論理宛先を取得できない可能性がある．よって，CPC は自身が持っている他ノード論理宛先振分リストから指定された論理宛先が見つからなかった場合には宛先取得を要求せず，転送された CPS によって論理宛先の解決が行なわれることを期待して，接続している中から 1 CPS を選択しメッセージを転送する（図 4）．

3） CPS 障害時

CPC は，CPS が稼働するサーバの障害を検知したとき，当該ノードの CPS に一定時間間隔で指定された回数分再接続を試みる．なぜなら，CPS 同士は自ノード論理宛先振分リストのブロードキャストメッセージを受信することによって障害から回復したことを検知できるが，CPC　はブロードキャストメッセージを受信できないため，CPC 自身が接

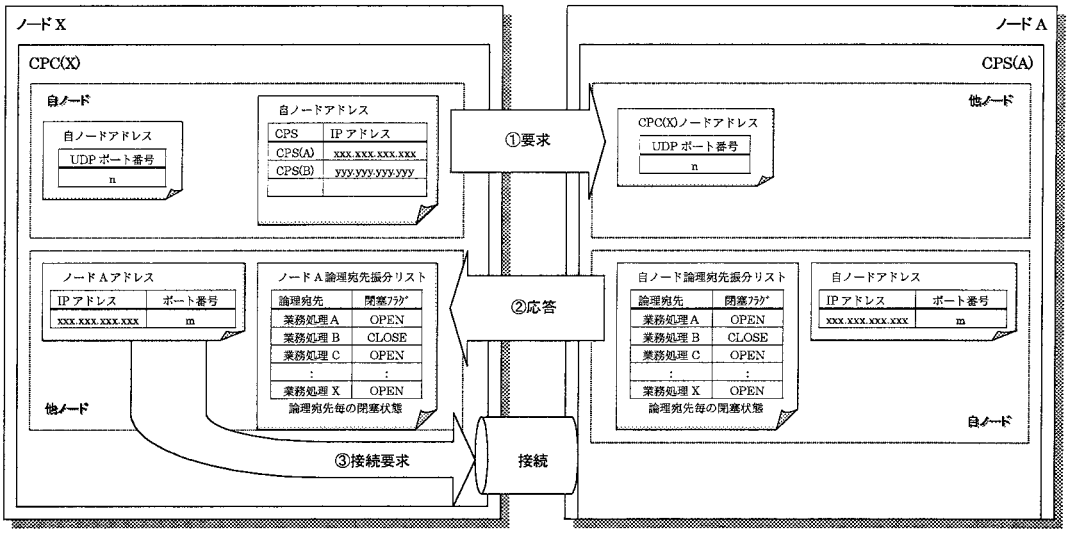


**図 3　CPC への論理宛先振分リストの配布**

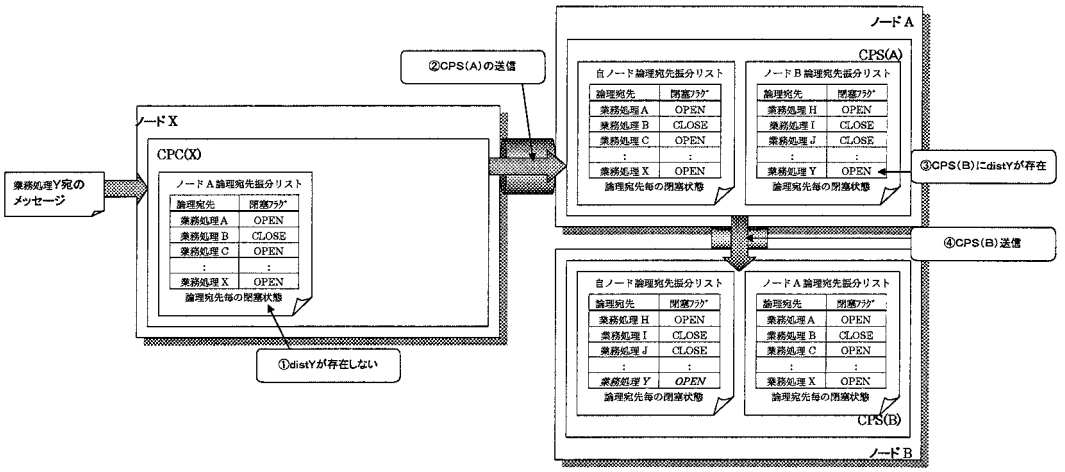


**図 4　CPC と CPS のルーティング**

続しにいかなければならないからである．もし指定回数以内に再接続できなければ，その CPS との接続は放棄される．

## 4.　リソース制御機能

MIDMOST は，CPU やメモリ等のリソース制御する幾つかの機能を保有している．本章ではその代表的な二つの機能について記載する．

### 4.1　スレッドプーリング

BP では業務処理を一つのスレッド上で実行するため，通常は業務処理が必要な時にスレッドを獲得して初期化し，業務処理終了時にスレッドを解放することになる．しかし同一の業務処理が頻繁に発生するシステムを考えた場合，一つ一つの業務処理が各々スレッドの生成/初期化/開放を行なうことは資源（CPU，I/O）の浪費であり，結果的にシステム全体のパフォーマンス低下を招く恐れがある．

そこで BP が提供する業務処理スレッドでは，定常時は業務処理ごとに一定数のスレッドを保有しておき，ラッシュ時には最大数までスレッドを生成する「スレッドプーリング」サービスを提供している．ユーザは事前に業務処理ごとの定常スレッド数と最大スレッド数を設定しておくことにより，本機能を使用することができる．

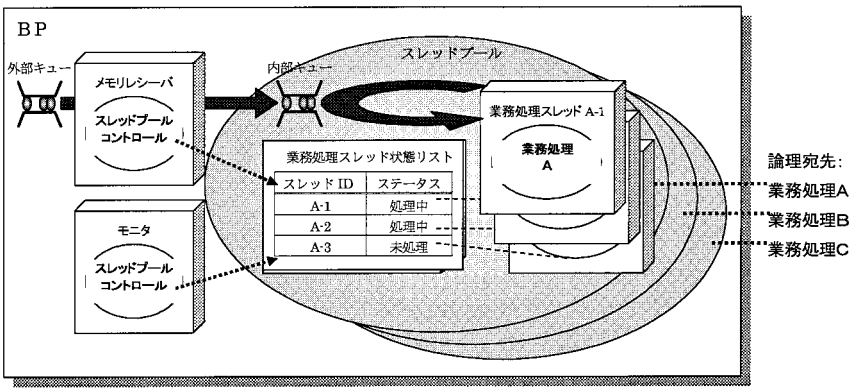


**図 5　スレッドプーリング構造**

BP 内に存在する業務処理スレッドは一種類ではなく，図 5 の業務処理 A/B/C のように，通常は業務処理やその設定情報などが異なるものが複数存在する．また一種類の業務処理はスレッド一つで構成される場合と，同一種類のスレッドが多重化して構成される場合がある．

このような特性をもつスレッド群を管理するため，「スレッドプール」という考え方を導入している．スレッドプールとは，同一種類のスレッドの集まりである．これにより一つの業務処理の多重化を実現している．

さらにスレッドプールの管理をするために，「スレッドプールコントロールオブジェクト」が存在し，スレッドプールをメンバとして保持することで，全体の属性・全体に対する管理を行っている．これにより一つの BP 内に複数の業務処理を実装することが可能となる．BP は，スレッドプールコントロールオブジェクトにより，自身の起動時に予め各業務処理を定常スレッド数分生成しておく．

BP 起動中にスレッドプールコントロールオブジェクトを保有しているのはメモリレシーバとモニタである．メモリレシーバは入力メッセージを受信し，該メッセージの処理対象である業務処理のスレッドプールが保有する内部キューにメッセージを PUT する．この時，スレッドプールコントロールオブジェクトにより業務処理が全て処理中であり，かつ処理中の業務処理スレッド数が最大スレッド数より少ない場合に，業務処理スレッドを追加する．

一方モニタはスレッドプールコントロールオブジェクトにより業務処理スレッドのステータスを定期的に監視しており，起動している業務スレッド数の内，処理中でないスレッドが存在し，かつ定常スレッド数より多い場合には，処理中でないスレッドを終了させる．

つまり業務処理の多重度は，スレッドプールコントロールオブジェクトにより入力メッセージの発生頻度に応じて動的に制御され，リソース消費の最適化が図られることになる．メッセージラッシュ時の業務処理スレッド数の増減は，図 6 のようになる．

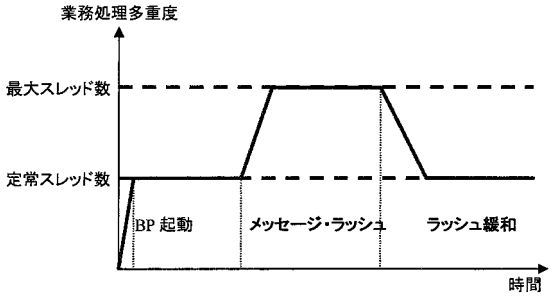


**図 6　業務処理スレッド数の増減**

### 4.2　コネクションプーリング

通常の業務処理ではデータベースを使用することが一般的である．ただし各業務処理内で各々データベースとのコネクションを確立/解放することは，通常 DBMS に対するコストを発生させ，業務処理全体のパフォーマンスを低下させることが考えられる．

そのため BP では DBMS とのコネクションを予め一定数保持しておき業務処理が必要な時に利用できる「コネクションプーリング」サービスを提供している（図 7）．

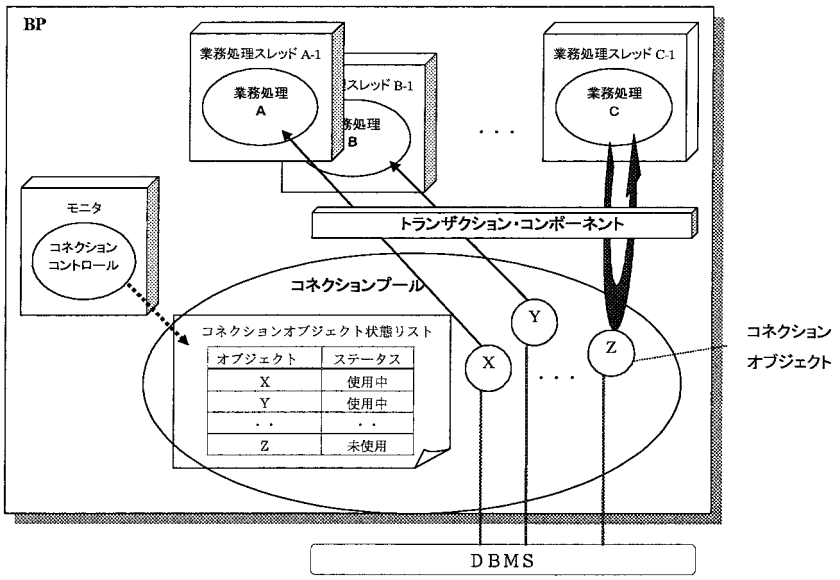


**図 7　コネクションプーリング構造**

DBMS とのコネクションの確立はモニタにより BP 起動時に行われ，コネクションオブジェクトとして BP 内にプールされる．

業務処理はトランザクションを開始した時点で，トランザクション・サービスがプール内のコネクションオブジェクトを取得することにより，コネクションを使用することができる．またトランザクションが終了した時点で，同コンポーネントによりコネクションオブジェクトをコネクションプールに戻すことにより，コネクション自体の再利用を実現しコスト発生の抑制を図っている．

また業務処理がコネクションオブジェクトを取得する際，既に未使用のオブジェクトが存在

しない場合には，モニタにより新規オブジェクトを追加する．このため一時的にラッシュ状態になるとコネクション数が増加するが，ラッシュが落ち着くと増加分は余剰コネクションとなる．そのためモニタは一定間隔でコネクションオブジェクト状態リストを監視し，余剰コネクションが存在する場合には削除することにより，リソース使用状態の最適化を図っている（図8）．

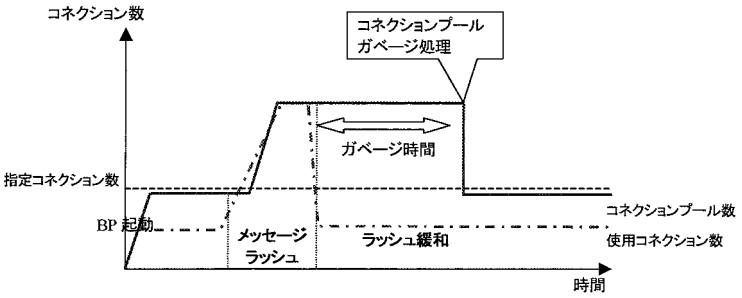


**図 8　コネクションプール数の増減**

## 5.　障害追及情報取得支援

MIDMOST では，アプリケーションに障害追及情報取得用の API を予め組み込んでおくことにより，情報の採取を可能にしている．本機能は，使用者のプログラム開発工程において，標準的なコーディング規約を設け，それに準拠することによって実現できる機能である．

1）　テキストログ・バイナリログ

テキストログ・バイナリログは，使用者が指定するログファイルに出力するインタフェースである．ログの出力レベルは 4 段階（Debug，Info，Warning，Error）あり，外部指示により開発環境と本番稼働環境でレベル設定を変更することでリソース使用の抑制を図ることができる．

2）　ステップトレース

ステップトレースは，プログラムの実行トレースを採取するインタフェースである．使用者はアプリケーションから呼び出す関数の出入口に，このインタフェースを記述する．これにより例外発生時およびアプリケーションの論理的エラー検知時には後述するプロセスダンプの採取により，どのステップまで処理が実行されていたか把握できる．

3）　スナップダンプ

使用者独自の情報をダンプ出力するインタフェースである．内容は任意のスナップダンプファイルに出力される．出力単位は，使用者が API にて指定したダンプ ID にて選択するほか，論理宛先単位，入力メッセージ単位に指定することができる．

4）　ユーザ領域ダンプ

アプリケーションのデータ領域をダンプ採取するインタフェースである．本 API を使用することで，アプリケーションのロジックにて異常を検出した場合に必要なデータ領域をダンプ出力することができる．

5）　プロセスダンプ

アプリケーション処理上の業務的なエラー発生時にプロセスダンプを出力するインタフェースである．本機能は WindowsNT　OEM　Support　Tools で提供される UserMode-

ProcessDump を使用して実現されている．ただしダンプ採取中はプロセスがフリーズするため，ダンプ採取要否を指定できるようにしている．

## 6. おわりに

本稿では，Windows 2000 DCS および ADS で動作する開発生産性の高いミッションクリティカル業務開発を，信頼性・可用性において高いレベルで実現するミドルウェアの基礎技術について述べた．

IT 技術は日々劇的に進歩しており，今後とも最新でかつ有効な技術を取り入れていくことは，オープン系開発において重要な成功要素であると認識している．

一方で，ミッションクリティカル業務のアプリケーション開発では，スクラップ＆ビルドのような短期開発・再構築を繰り返す開発手法は少ないため，一度構築したシステムは長期にわたって活用したいという要請が強い．今後，ミドルウェアに求められる要件は，「アプリケーションの改修を最小限に抑制した上で継続的に最新技術を取り入れられること」であると考える．

MIDMOST は，この要件を満たすとともに，さらに今後は Windows Server 2003 のみならず，UNIX/Linux にも順次搭載可能なミドルウェアとして発展させていく予定である．

---

**参考文献**

［1］ Jeffrey Richter, Advanced Windows　改訂第 4 版，(株)アスキー
［2］ Mark Russinovich, アーキテクチャ徹底解説　Microsoft Windows 2000　上，日経 BP ソフトプレス
［3］ John Paul Muler/Irfan Chaudhry, Microsoft Windows 2000　テクニカルリファレンス　パフォーマンスチューニングガイド，日経 BP ソフトプレス
［4］ W.Richard Strvens, UNIX ネットワークプログラミング　第 2 版　Vol.1　ネットワーク API：　ソケットと XTI，(株)ピアソン・エデュケーション
［5］ Microsoft Windows 2000 TCP/IP 実装の詳細，http://www.microsoft.com/japan/windows 2000/techinfo/howitworks/communications/networkbasics/tcpip_implement.asp?SD＝GN&LN＝ja&gssnb＝1
［6］ Microsoft Corporation, Microsoft ODBC 3.0　プログラマーズリファレンス　& SDK，アスキー出版局
［7］ Brian W.Kernighan/Dennis M.Ritchie，プログラミング言語 C　第 2 版，共立出版(株)
［8］ Bjarne Stroustrup，プログラミング言語　C＋＋　第 3 版，(株)アスキー
［9］ 日本ユニシス，技報　75 号　特集：Windows Data Center Ⅱ　ES 7000 で実現する大規模ミッションクリティカル・システム

**執筆者紹介**　馬　場　定　行（Sadayuki Baba）
1990 年日本ユニシス(株)入社．SE サービス，SBI 21/DE 開発業務を経て，MIDMOST 開発業務に従事．現在，システムサービス本部金融第二システム統括部開発三部に所属．

宮　野　将　彰（Masaaki Miyano）
1990年日本ユニシス(株)入社．SEサービスを経て，MIDMOST開発，適用支援に従事．現在，システムサービス本部金融第二システム統括部開発二部に所属．

佐　藤　則　之（Noriyuki Sato）
1992年日本ユニシス(株)入社．SPIRIT/STDL開発および保守業務を経て，MIDMOST開発業務に従事．現在，アドバンスドテクノロジ本部IT統括部サーバコンピューティング技術部に所属．