BANCS 接続システム（3）
――ホストオンライン処理に要求される信頼性・パフォーマンスの実現

Realization of Reliability and Performance required for Mission Critical System

平 野 敬 幸, 安 室 秀 則

要 約 BANCS-AP ミドルは，BANCS システムにおいて，業務ロジックと BANCS 基盤の中間に位置し，業務ロジックの起動と業務遂行に必要な仕組みを提供する．また，システム要件を満たすため，BANCS-AP ミドルには可用性，信頼性，保守性，利便性が求められる．本稿では，BANCS-AP ミドルを構成する BANCS-AP スタータと Control クラスが，どのように要件を満たすための仕組みを実装しているかについて述べる．

Abstract BANCS-AP Middle perceived as a cross between the application logic and BANCS system's infrastructure offers a mechanism necessary to activate the business logic and carries out business practices while realizing high reliability. To satisfy the system requirements, high availability, reliability, maintainability as well as user-friendliness were the key issues.

This paper discusses how BANCS-AP Starter and Control Class, constituting BANCS-AP Middle, provide the necessary functions.

1. はじめに

2001 年 10 月に本番を開始した，M 銀行の「新対外（BANCS）システム」（以下，BANCS システム）は，Microsoft Windows 2000 Datacenter Server（以下，W2KDCS）を搭載した Unisys Enterprise Server ES 7000（以下，ES 7000）環境でメインフレームと同等の信頼性・パフォーマンスを求められたシステムである．そこで，BANCS-AP ミドルでは，容易に業務ロジックを開発するために利便性の高い API を提供し，ハイパフォーマンス，高信頼性そして耐障害性の実現を目的に開発を行った．本稿では，BANCS-AP スタータの制御と，Control クラスの役割を中心に BANCS-AP ミドルの仕組みについて述べる．

なお，BANCS システム全体で共通なシステム概要とシステム要件に関しては本誌掲載の論文「BANCS 接続システム（1）―プロジェクト成功の鍵：POC 実践報告（山岸重雄著）」（以下，山岸論文）に記述されている．

2. BANCS-AP ミドルの概要

図 1 は，1 サーバ内で稼働する BANCS システムの構成要素である．点線で囲んだ部分が，BANCS-AP ミドルを示している．BANCS 基盤コンポーネントは，BANCS システムを構成するサブシステムの一つとして共有メモリ操作，名前付きパイプによるプロセス間通信，データベース操作，ロギング機能などの仕組みを取り入れ，BANCS-AP ミドルを効率よく実行させるライブラリ群である．BANCS システム内で，BANCS-AP ミドルと関連する常駐プロセスの特徴は次のとおりである．

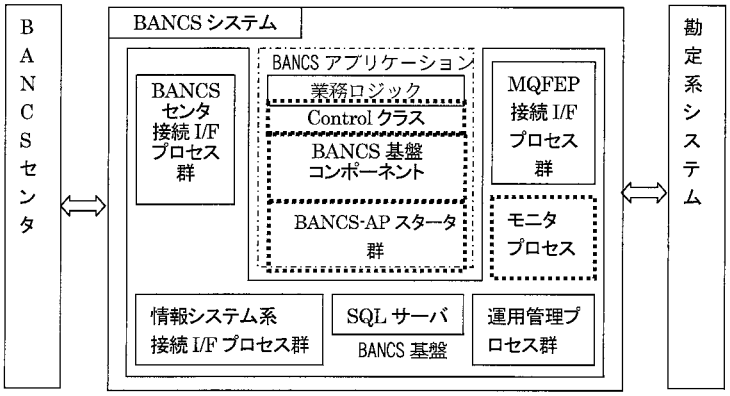


図 1　BANCS システム構成要素

1）　BANCS-AP スタータは，1 ノードで最大秒 100 件のトランザクション処理を可能にし，信頼性や耐障害性を確保するための仕組みを実装している．スレッドプーリング，データベースとのコネクションプーリングなどの機能を実装することにより，パフォーマンス要件の実現を可能にした．また，User Mode Process Dump を使用したプロセスのメモリダンプ機能，処理ごとのトレース機能などを備えており，障害発生時の保守性の向上を実現している．

2）　Control クラスは，業務ロジックの記述にあたって，オブジェクト指向やスレッドセーフに関して特に注意を払わなくても，安全なマルチスレッドプログラミングができる仕組みである．業務ロジックは，この Control クラスを継承してメインの処理を記述することで，Control クラスが提供する各種データアクセス，プロセス間通信，ログ採取機能などを行うメソッドをライブラリと同じ感覚で使用できる．また，業務ロジックの信頼性の確保と，パフォーマンス向上のため，業務処理で必要なデータ領域を属性として保持させたことにより，業務処理の中でのメモリ操作を行わなくてもよく，簡易性の高いアプリケーション構築を可能にした．

3）　モニタプロセスは，1 プロセスが存在し，電文送信タイムアウトの管理，Microsoft SQL Server 2000（以下，SQL Server）の稼働確認，共有メモリのガベージ処理などを行う．

3.　BANCS-AP スタータ

3.1　BANCS-AP スタータ概要

BANCS-AP スタータは以下のような構成要素を持つ（図 2）．

1）　レシーバスレッド

BANCS-AP スタータはプロセスごとに名前付きパイプを複数持ち，それぞれに読込み専用のレシーバスレッドを割り当て，同時に複数のプロセスからの入力を受け付ける．

2）　業務処理スレッド

各入力電文の業務処理はスレッドごとに並列に行い，業務処理を行うスレッド

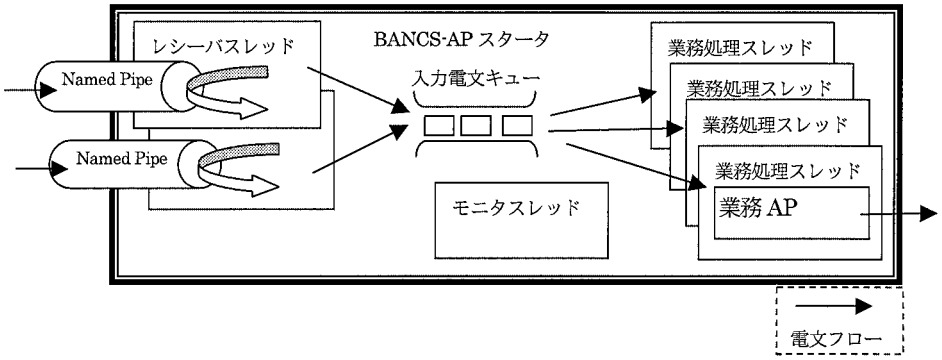


図 2　BANCS-AP スタータの内部構成

を予めプールしておく．スレッドプール処理の詳細は 3.3 節で述べる．

3）　モニタスレッド

プロセス内部のスレッドの状態を監視し，障害の検知やリカバリ処理を行う．スレッド監視機能の詳細は 3.3 節で述べる．

3.2　BANCS-AP スタータに求められるシステム要件

BANCS-AP スタータの役割を簡単に表現すると，他のプロセスから送られてきた電文を受け取り，その電文を業務 AP（アプリケーション）に受け渡すことである．しかし，金融系のミッションクリティカル業務である本システムにおいては，単に業務 AP に電文を渡すだけではなく，次のようなシステム要件が求められる．

1）　Availability（可用性）

・プロセス/スレッドの障害を他に波及させないようにする．

・SW（ソフトウェア）的に障害を切り離し，代替プロセス/スレッドにより継続処理可能にする．

・プロセス/スレッドの障害により縮退した構成を，逆に動的に追加できるようにする．

・システムの構成変更・運用変更に柔軟に対応できるようにする．

2）　Reliability（信頼性）

・要求される処理効率を確保する．

・データ量の増加により処理効率が劣化しないようにする．

3）　Maintainability（障害発生時の保守性）

・発生障害に対する自己解析可能な情報のロギングを行う．

・プロセス/スレッド間での制御の流れ/関係が認識できるようにする．

・発生障害の局所化を行う．

以上のような要求を満たすための処理の詳細について次節以降で述べる．

3.3　スレッドプーリング

3.3.1　スレッドプール処理（図 3）

BANCS-AP スタータでは，スレッド生成のオーバヘッドを考え，業務処理スレッドをあらかじめ一定数プールしておく．電文を受信した際，スレッドプール内の空いているスレッドに電文をディスパッチする．これにより電文受信から業務 AP の起動

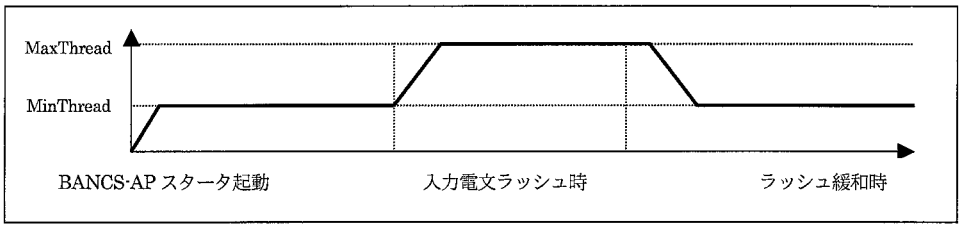


図 3　スレッドプール処理

までの時間を短縮することができる．

さらに，運用中に一時的にトラフィックが集中し多くのスレッドが必要とされる場合も想定される．そこで，BANCS-AP スタータでは，プールするスレッド数を動的に状況に応じてコントロールする事ができるよう実装した．電文を受信した際，空いているスレッドがなく，スレッド数が構成パラメタ MaxThread で指定された数より少なければ，新たにスレッドを生成して業務を割当てる．そして，ラッシュ状態でなくなり一定時間，起動されないスレッドがいる時は，スレッド数を再び減らしてリソースの無駄を最小限に抑える．

また，テストや運用時にパフォーマンスの再調整を行うため，プールしておくスレッド数の最小値や最大値等は，プロセスごとに実行時に決定する事ができる．

3.3.2　スレッド監視機能

BANCS-AP スタータでは，プールしている業務処理スレッドが正常に動作している事を確認するために，スレッド監視処理を行う必要がある．

プールしているスレッド数は動的に変動するために，以下のような方法を用いている．

1）各 BANCS-AP スタータは内部に業務処理スレッドの状況を監視するためにスレッドステータスリストを持つ. スレッドステータスリストの各ノードは, プールされている業務処理スレッドごとにあり, 各スレッドの処理情報を保持している．

2）業務処理スレッドは，各処理を行う前に，その処理の種別とタイムスタンプを更新する．

3）業務処理スレッドは起動・終了する際に，リストに対してノードの追加・削除を行い，動的構成変更に対応する．

4 ）モニタスレッドは一定間隔でリストを検索してタイムスタンプをチェックしハングアップしているスレッドを検出する．異常スレッドを発見した場合は，警告ログ出力，ダンプ出力，スレッド強制終了等を行い障害リカバリを実施する．

3.4　プロセス障害対応

3.4.1　プロセス障害種別

BANCS システムを構成する常駐プロセスで発生する障害は，以下の 3 種類に分類することができる．

① プロセスで発生したアプリケーション例外

② プロセス自身で検知した，処理続行不能（危険）な障害

③ プロセス外（クラスタサービス）から検知される，プロセスのハングアップ

これらの障害が発生した際,重要なのは即座に業務が再開できるようにすることと,現象を解析するための情報を残すことである．解析情報を残すために，BANCS-APスタータではUser Mode Process Dumpを用いて，ダンプファイルを生成している．ダンプファイルを生成する仕組みは，上記の①～③で異なるため，それぞれの場合の仕組みについて次節以降に述べる．

3.4.2 アプリケーション例外対応

プロセスでアプリケーション例外が発生した際の処理方法として，Windows 2000ではデフォルトでDr.Watsonが用いられるが，以下のような問題がある．

1） ダンプファイル名が制御できない．

Dr. Watsonの設定ではダンプファイル名はシステムで一つしか設定できないため,稼働中に複数回の例外が発生するとダンプファイルが上書きされてしまい,最後に発生した事象以外の解析ができなくなってしまう．

2） プロセスを救えない．

Dr. Watsonではプロセスが終了した際にダンプを生成するため，例外捕捉をするとダンプが生成されない．そのため例外捕捉を行わず解析情報を得ようとすると，プロセスに複数ある業務処理スレッドの一つだけで例外が発生した場合でも，プロセス自体が終了してしまうので，他のスレッドで処理中の電文処理も消失してしまう．

BANCSシステムでは，解析情報を確実に残すことと，業務処理を可能な限り続けるという，システムの要求があるため，Dr.Watsonの機能だけでは不十分である．そこで，Windows NT OEM Support Toolsで提供されている，User Mode Process Dumpを用いた．User Mode Process Dumpは以下の方法でクラッシュダンプファイルを作成する事ができる．

1） コマンドライン出力

コマンドラインから，UserDump.exeにダンプを生成したいプロセスのプロセスIDと生成したいダンプファイル名を指定して実行し，クラッシュダンプファイルを生成する．

2） Exception Monitoring機能

サービスとして起動されているUser Mode Process Dumpにあらかじめ，ダンプを作成したいプロセスを登録しておくと,そのプロセスで例外が発生した際,自動的にクラッシュダンプファイルが生成される．この際，ダンプファイルの出力場所，モニタする例外の種類をプロセスごとに指定する事ができる．またダンプファイル名はプロセス名と例外が発生したスレッドのスレッドIDから構成される．

Exception Monitoring機能の大きなメリットとしては，ダンプ出力処理の容易性とダンプ出力の確実性がある．User Mode Process Dumpにプロセス名を一度登録しておけば,（マシン，プロセスの両方の）起動や再起動，一度に複数の同一プロセスが起動した場合などにも対応し，再登録の必要はなく，exeの入れ替えを行っても同様である．また，Exception Monitoringではファーストチャンスと呼ばれるタイミング（ユーザ定義の例外ハンドラに飛ぶ前）で例外のハンドリングが行われ，ダン

プ出力を行う．このためスタックが著しく破壊され例外ハンドラで処理を行えないような場合でもダンプが生成される．

ただし，Exception　Monitoring 機能で監視するプロセスの処理効率が若干落ちるという問題があるが，テストの結果，許容範囲と判断した．

以上のようなことから，BANCS-AP スタータでは，Exception Monitoring 機能を使い，次のように例外処理を行う（図 4）．

① 業務処理スレッド単位で例外捕捉を行う．

② 業務処理スレッドで例外が発生した際は，プロセスをモニタしている User Mode Process Dump でそれを検知し，クラッシュダンプファイルを生成する．

③ 次にユーザ定義の例外変換関数を呼び出す．例外変換関数では，例外発生ポインタと例外番号を保持したユーザ定義の C＋＋例外に変換し，スローする．

④ 業務処理スレッドで例外をキャッチし，例外発生のログを出力し，処理していた電文をバイナリログとして出力する．

⑤ 業務処理スレッドを終了する前に，必要であれば，領域の開放や DB のトランザクションの破棄（ロールバック）を行う．

⑥ 障害発生スレッドを終了する．

⑦ 代替スレッドが起動される．

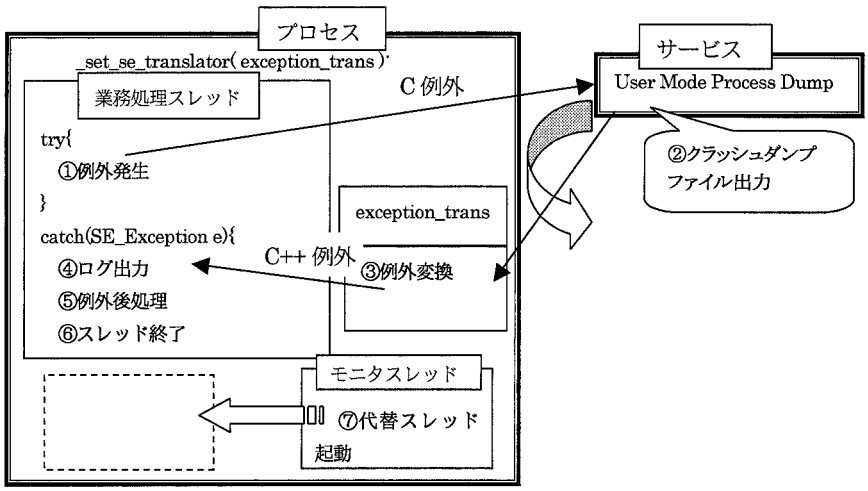


図 4　例外処理の流れ

3.4.3　プロセス自身で検知した，処理続行不能な障害

プロセス内の処理で，何らかの原因で正常な処理が続行不可能だと判断した場合は，即座に終了する必要がある．その際，プロセス自身で CreateProcess を使いコマンドラインから UserDump に，自身のプロセス ID と出力ファイル名を指定してクラッシュダンプファイルを生成する．この際，ダンプファイル名にはプロセス名，発生時間とプロセス ID を含めて，重複するのを防いでいる．

3.4.4　プロセス外（クラスタサービス）から検知される，プロセスのハングアップ

BANCS システムでは，基盤システムとして MS クラスタサービスを使ったリソー

ス管理を行っている．リソース管理等の詳細は本稿の記述範囲ではないため詳細は述べないが，システム内の常駐プロセスがハングアップしているとクラスタサービスのカスタムリソース DLL に検知された場合は，プロセスを終了する前に，User Mode Process Dump を用いてクラッシュダンプを生成する．

4. Control クラス

BANCS-AP スタータによりシステム実行時のパフォーマンスと信頼性の向上が実現された．しかし，それは実行時に発生した例外的な問題によってシステム全体が致命的な状態に陥ることを防ぐという範囲にとどまる．つまり，BANCS-AP スタータのみでは電文を正しく送り届けるという本来の目的に対しての信頼性の向上には限界がある．やはり，システム全体の信頼性を高めるのに一番効果的な方法は，システムを開発する段階から誤りの少ないプログラムを作るようにしていくことである．BANCS システムではアプリケーションコードを BANCS-AP ミドルと業務ロジックに分離し，業務ロジックの開発者を，電文の送受信処理，マルチスレッディング，例外処理などの問題から解放して開発の負担を減らすことで，信頼性の高いコードを生み出すように設計されている．

Control クラスは BANCS-AP スタータと業務ロジックとのインタフェースとなる役割を担っている．考慮すべきインタフェースは BANCS-AP スタータから業務ロジックを呼び出すためのインタフェースである BANCS 基盤フレームワークと，業務ロジックからデータベースアクセス，プロセス間通信，ディスクや共有メモリを用いた実行状態の共有などの機能を呼び出すためのインタフェースである BANCS 基盤 API がある．本章では，これら二つのインタフェースの観点から Control クラスについて述べる．

4.1 BANCS 基盤フレームワーク

Control クラスは BANCS-AP ミドルの中で，業務ロジック構築のフレームワークとしての性格を持っており，イベントドリブン型のプログラミングスタイルを採っている．BANCS-AP スタータはプロセス起動時，スレッド起動時，電文受信時などのイベントが発生するたびに Control クラスに定義されたコールバックメソッドを呼び出すことにより，業務ロジックの処理を取り込んでいる（図 5）．

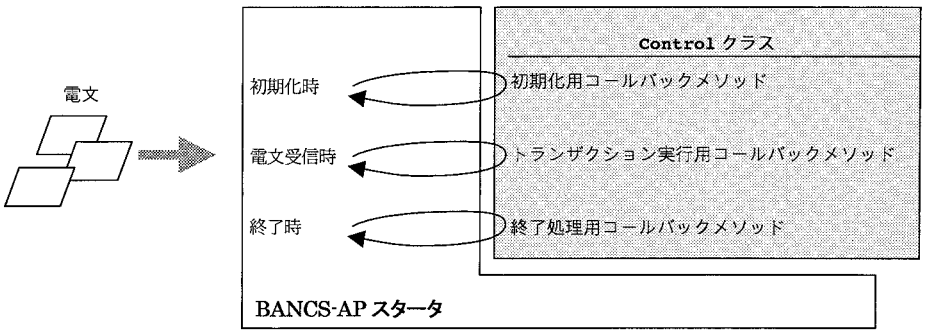
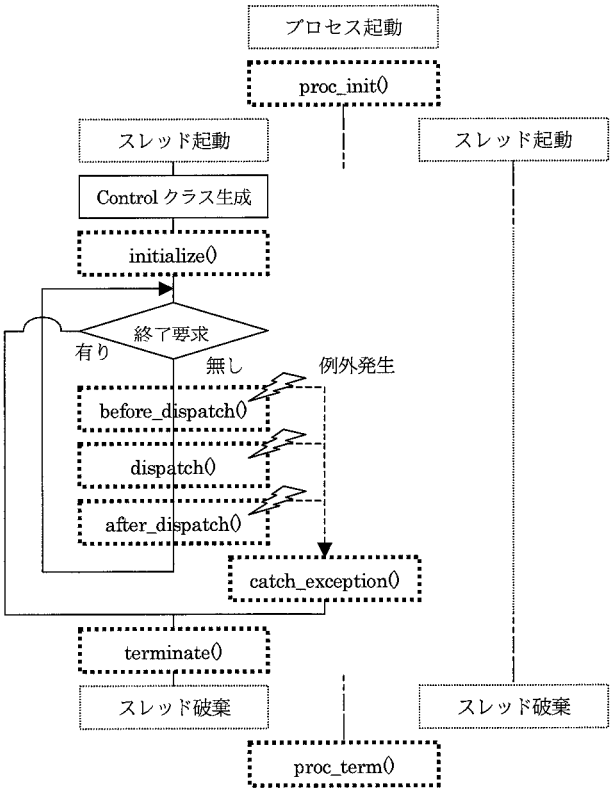


図 5　BANCS-AP スタータと Control クラスの関係

BANCS-AP スタータから呼び出されるコールバックメソッドは 8 種類存在し，業務ロジックの開発者は Control クラスを継承してコールバックメソッドを必要に応じてオーバライドするだけで，マルチスレッド環境に BANCS システムの業務ロジックを組み込むことが可能となる．BANCS 基盤フレームワークを構成するコールバックメソッドの呼び出しタイミングを示す（図 6）．点線で囲んだ部分がコールバックメソッドであり，その概略を以下に示す．



- proc_init()　プロセス起動時に 1 度だけ呼ばれる Control クラスのインスタンス全体に影響する初期化処理を行う
- proc_term()　プロセス終了時に 1 度だけ呼ばれる Control クラスのインスタンス全体に影響する終了処理を行う
- initialize()　スレッド起動時に 1 度だけ呼び出され，Control クラスのインスタンスの初期化処理を行う
- terminate()　スレッド終了時に 1 度だけ呼び出され，Control クラスのインスタンスの終了処理を行う
- dispatch()　電文を受信するごとに呼び出され，業務処理を行う．必ずオーバライドする必要がある
- before_dispatch()　電文を受信するごと dispatch ()の直前に呼び出される．電文処理に先立って共通な処理があれば行う
- after_dispatch()　dispatch ()終了後に呼び出される．電文処理後に共通な処理があれば行う
- catch_exception()　dispatch，before_dispatch，after_dispatch で例外が発生した時に呼び出される

図 6　BANCS 基盤フレームワークのフロー

4.2 BANCS 基盤 API

BANCS 基盤 API は Control クラスのメソッドとして定義されており，開発作業の軽減と保守性の向上のために以下のような特徴を有している．

① BANCS 基盤 API を使用するにあたり，特別な初期化処理や終了処理は不要で，BANCS 基盤フレームワークを構成するコールバックメソッド内から平易に呼び出すことができる．

② スレッド間でのリソースの競合問題を API 内で吸収し，かつ，業務ロジックの記述に際して必要なリソースをあらかじめ BANCS 基盤 API の一部として確保しておくことにより，業務ロジック開発者のリソース管理の負担を軽減させる．

③ コードの正当性の確認やデバッグの効率を高めるのに役立つように，API の呼び出しトレースを逐次記録する．

次に BANCS 基盤 API の中から，コードの信頼性を向上させるためのコード正当性の確認・デバッグ作業の効率化を支援するロギングに関連する機能について述べる．

1） メッセージログ

実行中に発生したエラー，警告，情報，デバッグ情報などのメッセージを簡易な用法でログファイルに出力する機構を開発者に提供している．ログファイルはあらかじめ用途ごとに 3 種類を用意しており，ファイル番号によって出力先を指定する．3 種類のログファイルは，標準ログ，運用監視対象ログ，破棄電文記録ログである．ログの出力指定は，メッセージと共にログ出力レベル（エラー，警告，情報，デバッグ情報）を埋め込むことによって行う．メッセージログは確実なログ収集を目的としているため，アプリケーション起動時に出力対象とするログ出力レベルを指定し，実行時に動的変更をすることはできない．

2） スナップダンプ

アプリケーション実行中にログ出力レベルを変更できないメッセージログを補完するかたちで，ログの出力を比較的柔軟な方法で行うようにするために用意されたのがスナップダンプである．スナップダンプは実行時の出力内容を動的に変更することが可能であり，カテゴリごとに割り振られたスナップ ID によりコード内のログメッセージを特定し，ログ出力レベルにより出力を制御する．外部ユーティリティであるスナップダンプ制御コマンドにより出力レベルを操作することができる．

3） API トレース

各 BANCS 基盤 API では，API 呼び出しの際に API 名と引数情報を API トレースとして記録している．これらの情報は Control クラス内に確保されており，後述するエラーダンプの際には API トレースの情報も同時にログファイルに出力されるため，エラーが発生した時点のコードの実行状況が把握でき，問題解決を助ける効果がある．API トレースはスナップダンプの機能を利用することにより，実行時の動的な出力制御が可能となっている．

4） エラーダンプ

dispatch（）の戻り値として FALSE が返された場合，業務ロジックで対処不可能な事象が発生したということで，BANCS 基盤フレームワークは Control クラス自身のメモリイメージをログファイルへ出力する．Control クラスには BANCS 基盤 API への引数に用いるリソースが含まれており，エラーが発生した状況を確実に記録しておくことができる．

以上のような豊富なロギング機能を提供することにより，開発・テスト・デバッグのサイクルを効率よく繰り返すことが可能になり，結果として信頼性の高いプログラムコードが生み出だされる．

5. トランザクションの信頼性の向上

BANCS システムのように多くの電文を高速に中継するようなシステムの場合には，予期せぬエラーによる処理の中断やスレッド障害などにより，電文を正しく処理できない状態に陥っても電文の欠損を防がなければならない．また，ネットワーク等の障害による応答電文の未着，接続先システムのダウンなどの状況においてもトランザクションの一貫性を保つ必要がある．

BANCS-AP ミドルでは実行状態のトランザクション連動，タイムアウト処理そして実行状況のリカバリを採用して電文処理の信頼性向上を図っている．

1） トランザクション連動

BANCS-AP ミドルはデータベースを用いたトランザクション処理を行うと同時に共有メモリテーブルを用いて処理速度の向上を図っている．通常はデータベースの内容と共有メモリテーブルの内容は同期しているが，障害発生などの要因により両者の内容に一貫性がなくなる場合がある．このような状態を回避するために，Control クラスはトランザクション連動と呼ばれるトランザクション管理機能を提供する．

トランザクション連動とは Control クラス内に確保されたトランザクション・オブジェクトと呼ばれるインスタンスを用いて，トランザクション中に一連の BANCS 基盤 API の呼出しで設定されるパラメータ情報をもとに共有メモリテーブルを操作することである．トランザクション・オブジェクトはビギン API 呼出し時に生成され，コミット API 呼出し時に解放を行う．コミット API を呼び出さずに dispatch（）を終了した時やロールバック API が呼び出された時にはトランザクション連動が実行され，共有メモリテーブルの内容が復元される．

2） タイムアウト処理

電文の中継処理というのは，電文を受信してトランザクション処理を行ったのち，電文を送信するという流れである．この流れが往復することで 1 電文の処理が完了する．つまり 2 回のトランザクションが対にならなければならない．そのため往路で電文を送信した後に，ネットワークや接続先システムの障害が発生した場合にはトランザクションの取り消しが必要になる．

トランザクションの取り消し処理は業務ロジックの一部として実装されるべきである．BANCS-AP ミドルでは，それをサポートするためにタイマ監視機構を

提供している．電文送信時に送信電文と共にタイマ登録を行うと，タイムアウト発生時に自動的にタイムアウト発生時の業務ロジックが起動される．

3） リカバリ

BANCS システムの起動時にはデータベース，共有メモリテーブルやその他の実行状況を保持している要素は必ずしも互いに一貫した状態になっているとは限らない．フェイルオーバが発生した際も同様である．そこで，現在のデータベース状態を一貫性保持のための基準として，データベースの情報をもとにその他要素の状態を再設定する処理を実行状況のリカバリ機能として提供する．このリカバリ機能は BANCS システムの起動時及びフェイルオーバ発生時に使用される．

6. おわりに

本稿では，ES 7000 と W2KDCS 上で稼働する，ミッションクリティカルなシステムである BANCS システムの中で，アプリケーションの中核となる BANCS-AP ミドルの可用性，信頼性，保守性，利便性とパフォーマンス要件を実現するためのさまざまな仕組みについて述べた．

今後，ES 7000 と W2KDCS を使用したミッションクリティカルなシステム構築が増えることが予想される．ただし，ES 7000 と W2KDCS 上に，従来のメインフレームで稼働していたようなミッションクリティカルなオンライン・リアルタイムシステムを構築する上で二つ問題がある．

一つは，ユニシス HMP　IX シリーズの OS 2200 で提供されているような TIP/XIS レベルのミドルウェアの欠如である．当社が販売促進している Windows サーバ上での業務アプリケーション構築では，マイクロソフト社および他のサードベンダ製品の基盤ソフトウェアを利用した開発が行なわれている．金融のようなミッションクリティカルな業務アプリケーションの開発において，トランザクションの整合性・一貫性を保証するには，これら基盤ソフトウェアの組み合せだけでは機能不足，連携不備な部分がある．BANCS-AP スタータは，スケジューリング・メモリ管理機能や MS クラスタサービスを利用したホットスタンバイ相当の機能を実現し，Windows で利用可能なミッションクリティカルなシステム構築用ミドルウェアの仕組みを実現している．

もう一つは，従来のミッションクリティカルなオンライン・リアルタイムシステム開発経験者に Windows 上でのプログラミングに長じた人材が不足していることである．C＋＋言語でオブジェクト指向のわかるプログラマを多数同時に集めるのも容易ではない．Control クラスは，そのような，オブジェクト指向やマルチプログラミングにおけるスレッドセーフの考え方をあまり理解しなくても，結果的にオブジェクト指向でかつスレッドセーフのプログラムが作れる仕組みの提供を目指して用意したものである．すなわち，BANCS-AP スタータが，電文処理スレッドを開始するに当たって，事前にその電文処理用の Control サブクラスを割り当て，電文処理用のメソッドを呼び出す．従って，アプリケーションプログラマは，電文処理用のメソッドをオーバライドして電文処理ロジックを C 言語の感覚で記述すれば，結果的にオブジェクト指向のメリットである親クラスのメソッド（BANCS 基盤 API）を継承して使用

でき，かつ，Control クラスの属性として定義された作業領域も，各電文処理スレッドごとに，スレッド固有な領域として確保されているので，この作業領域を用いる限り，意識せずにスレッドセーフなプログラミングができる．

以上の様な成果を基に，現在 BANCS-AP ミドルの仕組みをさらに発展させて，Windows サーバ上に W2KDCS（または Windows 2000 Advanced Server）を使用したミッションクリティカルなシステムを構築する上で必要な，汎用性に富んだ信頼性の高いミドルソフトウェアを新たに開発済みである．

執筆者紹介　平　野　敬　幸（Yoshiyuki Hirano）
1969 年生，1993 年法政大学工学部土木工学科卒業．同年日本ユニシス(株)入社．総合開発支援ツール TIPPLER の開発，大規模分散システム SYSTEMν［nju：］の開発を経て，BANCS システムの開発に従事．現在，ファウンデーションサービ部ミドルウェアサービス室に所属．

安　室　秀　則（Hidenori Yasumuro）
1975 年生，2000 年筑波大学大学院理工学研究科修了．同年日本ユニシス(株)入社．BANCS システムの開発に従事．現在，ファウンデーションサービ部ミドルウェアサービス室に所属．