

非同期 Web サービスのためのメッセージング基盤設計

— SOAP/JMS プロトコルによる WAS-MB 連携とその高可用性・拡張性 —

石橋 香代子 平岩 梨果 神野 稚子

Designing Messaging Infrastructure for Asynchronous Web Services

- High Availability and Scalability for WAS-MB Integration Using SOAP/JMS -

Kayoko Ishibashi, Rika Hiraiwa, and Wakako Jinno

Web サービス連携において SOAP/JMS (Java Message Service) を通信プロトコルとする非同期処理を行うことは、接続先の状態にかかわらずメッセージ送信を可能にする有用なアプローチである。しかし、その実現のためにはメッセージ基盤設計が必要とされる。本論文では Enterprise Service Bus として WebSphere® Message Broker (MB) を採用したときの、WebSphere Application Server のメッセージング・エンジン (ME) 採用の優位性を考察し、その可用性、拡張性の高いメッセージング基盤設計の実現方法を述べる。特に ME の拡張性を実現するためには、Web サービス・リクエスター／プロバイダー側のメッセージ処理が正しく分散されるように考慮する必要がある。そのための構成上のヒントや考慮点を、検証結果を交えながら紹介する。

Asynchronous messaging using SOAP/JMS (Java Message Service) transport protocol is an effective approach that enables Web services to transmit messages without being affected by the status of the system being connected to. In this paper, we discuss the advantages of using Messaging Engines (MEs) in WebSphere® Application Server when WebSphere Message Broker is used for an Enterprise Service Bus. In addition, we describe how to design highly available and scalable messaging infrastructures using Messaging Engines. Especially with regard to scalability, messaging workload should be properly balanced in MEs both on the Web services requester side and on the provider side. We present configuration tips and considerations together with our test results.

Words & Phrases : SOAP/JMS, メッセージング・エンジン, MDB, MQ リンク, Web サービス
SOAP/JMS, Messaging Engine, MDB, MQ Link, Web Services

1. はじめに

SOA の普及に伴い、Web サービスで開発されるアプリケーションは増加の一途をたどっている。Web サービス連携を行う処理形態としては、大きく分けて同期および非同期に分類できるが、非同期連携は夜間に受注を受け付け、翌朝に在庫の引当を行うような時間差を吸収する処理形態に適する。Web サービスの特徴として、SOAP メッセージを使用するため特定の通信プロトコルに依存しないという点が挙げられる。Web サービスの通信プロトコルとして HTTP (以下、SOAP/HTTP) を使用することが一般的だが、先に挙げたような非同期連携を Web サービスで行う際には、メッセージング機能を提供する API である Java Message Service (JMS) [1] プロトコル (以下、SOAP/JMS) を適用することで実現できる [2]。また、SOA のアーキテクチャーでは Enterprise Service Bus (以下、ESB) を採用するケースも多くなってい

る。Web サービス連携の処理形態を考えた場合、リクエスターとプロバイダーで直接連携するケースもあるが、それぞれの関係をより疎にし、サービスの透過性を高めるために ESB を採用することは、優位性があるといえる。また ESB 製品の選定については、連携するサービスの特徴、ESB の責務によって決定することが重要である。IBM の ESB 製品として、WebSphere Data Power、WebSphere ESB、WebSphere Message Broker (以下、MB) がある。WebSphere Data Power は主に SOAP/HTTP による接続を焦点としパフォーマンスに利点がある。WebSphere ESB は WebSphere 基盤との親和性や Java プログラムによる柔軟な仲介処理が特徴である。MB は WebSphere MQ をベースとしたレガシー接続の仲介処理を行ってきた製品であり、三つの製品のうち最も実績があり、IBM ESB 製品ではアドバンスド製品と位置付けられる。本論文では、Web サービス実行環境として、WebSphere Application Server (以下、WAS) を、ESB と

提出日:2007年5月12日 再提出日:2008年9月22日

してMBを取り上げる。WAS-MBの連携には接続パターンが複数あり、それらの比較検討をするだけでも労力を要する。本論文では複数の接続パターンを整理した上で、WASのメッセージング・エンジン（以下、ME）を採用した構成に注目する。この構成は多くの利点が享受できるが、構成上の難易度が高く実現性についてあまり議論されていない。本論文を通して、検証結果を交えた実装上のヒントや考慮点を明らかにしていく。

さらにSOA化を前提とした場合、連携するWebサービス数やSOAPメッセージによる通信量の増加を考慮する必要があり、通常のWebアプリケーションで使用する以上に、可用性や拡張性といった非機能要件の実装が必須となる。

まず2章でWebサービス連携における通信プロトコルについて説明し、3章以降は特にSOAP/JMSに注目する。3章ではSOAP/JMSの仕組みについて、4章ではWAS-MBの接続パターンの中から有用性の高い構成について説明する。5章では、高可用性の実現方式について説明し、WAS-MB間のチャンネル断絶時に有効な復旧方法を比較検討する。6章では、MEの拡張構成を実現するための着眼点とその実現方法を紹介する。以上の内容を踏まえ、最後に、SOAP/JMSを使用したメッセージング基盤を採用する価値について述べる。

2. Web サービス連携における通信プロトコル

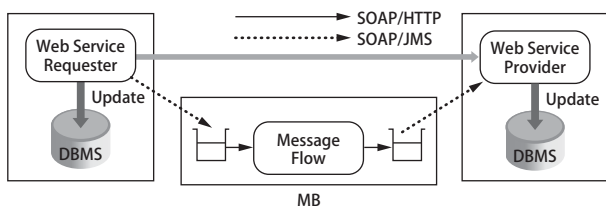


図 1. Web サービスによる業務連携

二つのWebサービスが連携し、かつそれぞれのビジネス・ロジックからリソース（DBMS）を更新するという処理（図1）において、適用するプロトコルについて検討する。まずこの処理に対する要件は、業務ごとに下記の二つに大別できる。

- 要件 1：厳密な整合性を要求される
- 要件 2：整合性よりも、サービス時間帯の影響を受けないことを要求される

上記の要件を満たすための実現方式として、それぞれ下記の二つの実現方式が検討できる。

- 要件 1：SOAP/HTTPによるWeb Services Atomic Transaction（以下、WS-AT）の採用
- 要件 2：SOAP/JMSによる非同期連携処理の採用

まず、SOAP/HTTP上でWS-ATを採用する方式は、下記の利点、考慮点が考えられる。

- (1) 利点
 - 厳密なACID属性^{*1}の順守により、障害時の確実なリカバリー処理が可能になる
 - 要求応答型の処理により障害が即時に検知可能である
- (2) 考慮点
 - 同期処理で厳密なACID属性を保つため、プロバイダーが稼働していない場合には処理が継続できない
 - 呼び出すWebサービス数に比例してDBMSのロック保持期間が長期化する

それに対しSOAP/JMS方式では、下記の利点、考慮点が考えられる。

- (1) 利点
 - 一方向型の処理ではプロバイダー側の状態にかかわらずリクエスター側の処理が完了できる
 - 一度送信されたメッセージは、プロバイダー側の処理完了まで永久保持することが可能なため、非常に信頼性の高い処理が保証できる。また確実なメッセージの転送を、ミドルウェアに任せることができる。
- (2) 考慮点
 - 最終的に複数リソースに対する更新が行われるが、UOW（Unit Of Work）が分割されるため、一時的に両リソースの不整合が生じる状態が発生する（ACID属性中Isolation / 分離性の不成立）。
 - プロバイダー側で、複数スレッド（MDBなどを使用）でキューのメッセージを処理する場合、メッセージの処理順序が維持されない。単一スレッドによる順序性の保持か、複数スレッドによるパフォーマンスのトレード・オフとなる。
 - WASv6.1では、SOAP/JMSは標準仕様ではなくIBM独自実装のため、他製品との相互接続性がない。ただし、WASv7ではW3Cで標準化が進められているSOAP/JMS1.0 [3]を採用しており、今後はSOAP/JMSにおける相互運用性も期待できる。

どちらの方式も、ともに正常終了時の最終的な処理結果は同じであるが、UOWの範囲や実現できる非機能要件は大きく異なる。特にSOAP/JMS方式の考慮点として挙げた一時的に発生するリソースの不整合については、これを許容できる業務であることが重要なポイントである。しかしこの制限が許容できる場合、この方式の利点がWebサービス間の関係を疎にし、関心の分離を実現できることに注目したい。これは前述したWS-ATにおける二つの考慮点を克服している。なお、この原子性および分離性を緩やかにするといった特性を同様に持つ実現方式としてロング・ランニング・トランザクションが考えられる。これは、分割された複数のUOWを一連の論理的なトランザクションとして考え、障害時などにはロールバックするための補正処理などを組み込むことを前提にしている[4]。その反面、SOAP/JMSによるメッセージ送信はシステム間の1回のメッセージ送信を焦点として考える。この送信方法は利点にも挙げたようにミドルウェアが通信を保証することが可能なため、本論文ではその優位性を生かすための基盤設計を3章以降で説明していく。

*1 トランザクションを矛盾なく処理する4つの特性、原子性（Atomicity）、一貫性（Consistency）、分離性（Isolation）、永続性（Durability）の頭文字をとってACID属性という

3. SOAP/JMS の仕組みとルーティング実現方針

3.1 SOAP/JMS のメッセージ送信方式

まず、図2において、SOAP/JMS のメッセージ送信の仕組みを説明する。SOAP/JMS ではメッセージの送受信には JMS 宛先 (Queue または Topic) が使用される。Web サービスの実装は EJB (Stateless Session Bean / 図2中 "SLSB") で実装され、開発ツールにより生成される Message-Driven Bean (MDB) を介して通信される。

MDB はエンタープライズ・アプリケーション・アーカイブ (EAR) 内に一つ存在する。メッセージ内の SOAP ヘッダー中の targetService プロパティに呼び出す対象となるサービス名が指定されており、MDB はそこで設定されたサービス名に対応する Web サービスを呼び出す。しかしこの仕組みでは、リクエスターはプロバイダーが存在するキューに正しく送信するために宛先のキューを意識する必要があるという課題が存在する。この課題は ESB である MB の採用で解決できる。

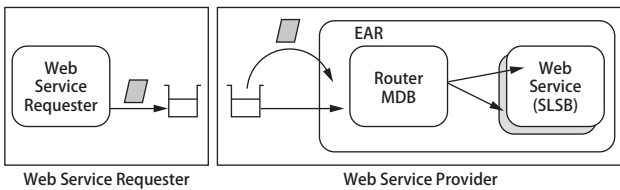


図2. SOAP/JMS の仕組み

3.2 WAS から MB を介してのルーティング

受信したメッセージを MB が正しいプロバイダーと結びつけたキューへとルーティングするためには、メッセージの中からその情報を特定する必要がある。それには、Web サービス名が格納される SOAP ヘッダーの targetService プロパティの利用が有効である。リクエスター側の WAS にて設定することにより SOAP ヘッダーの targetService プロパティが MQRFH2 ヘッダーとして MB に渡されるためメッセージ・フロー内でこれを抽出しルーティングすればよい。サービス名からルーティング先のキューを特定できるようなネーミング・ルールを徹底する必要があるが、この方法の利用によりアプリケーションに影響を与えずにルーティングが可能となる。

4. SOAP/JMS における WAS-MB 接続パターン

WAS-MB の接続は、WAS-MQ の接続とほぼ同等と考えられ、さまざまな接続形態が存在する。接続形態は、キューの配置によって次の3パターンに分類できる。

<パターン1> MQ が MB 側のみ存在

WAS 側からリモートの MQ キュー・マネージャー (以下 QMgr) へ MQ クライアント接続するパターン。

利点：追加の MQ ライセンス・管理が必要ない

欠点：MB 側でサービス停止している場合は WAS 側にてエラーが発生する

<パターン2> ME (MQ) が WAS 側のみ存在

ME に MB の JMS1.1Provider 接続機能 (v6.0 以降で提供) を使用するパターン。

利点：MB 側で MQ の管理が必要ない。ただし、MB 基盤の使用は MQ スキルが前提であることが多く、また MQ ライセンスも MB に含まれるため、享受できるメリットは少ない

欠点：WAS 側でサービス停止している場合は MB フロー側でエラーが発生する

<パターン3> WAS/MB 間の ME (MQ)-MQ 接続

WAS/MB 側の両方にキューを配置する構成で、この構成には2通り存在する。

<パターン3-1> MQ-MQ 接続

WAS 側に MQ を導入し、MQ QMgr 接続を行うパターン。

利点：

- WAS/MB がお互いのサービス停止に影響されない
- 実績が最も多い

欠点：WAS 側の追加 MQ ライセンスが必要

<パターン3-2> ME-MQ (MQ リンク) 接続

WAS 側に ME、MB 側に MQ を配置し、ME/QMgr 間接続に MQ リンクを使用するパターン。

利点：

- WAS/MB がお互いのサービス停止に影響されない
- 追加の MQ ライセンスが必要ない

欠点：

- ME/MQ 両方の管理が必要
- 実績が少ない

どのパターンにおいても利点・欠点があるが、SOAP/JMS の接続を実現するためには、どのような WAS-MB 構成が適しているだろうか。2章で述べた「プロバイダー側の状態にかかわらずリクエスター側の処理が完了」できるという SOAP/JMS の利点を最大限に活かすことを考えた場合、WAS/MB のそれぞれのローカルにキューを配置する<パターン3>構成が WAS と MB がお互いのサービス停止に影響されず望ましい。また、<パターン3>構成の2通りのうち、実績から判断すると<パターン3-1>が有用であるが、将来 Web サービス化が進み、MB による連携対象の WAS が増加するほど WAS 側に追加 MQ ライセンスが必要となることが問題となる。そこで本論文では、WAS 側の MQ の代わりに ME を使用し、MB の QMgr と接続する構成を提唱する。ME は WAS ライセンスの一部のため、将来の連携対象の WAS の増加に伴いコスト面で大きな利点があり、有力なメッセージング基盤構成である。この構成はまだ実績が少ないのが現状だが、筆者らが実際に検証した構成を適用することにより実績面の不安は解消できる。

なお、WAS は v6.1、MB は v5.0 で検証を行っている。

MB の最新バージョンは v6.1 であるが、設計方針は v6.1 以降でも適用可能である。

5. WAS-MB 構成の高可用性の実現

WAS-MB 構成の高可用性を実現するためには、図 3 に示すように、WAS-MB 構成に参加するすべてのコンポーネント (WAS, MB とその接続) の可用性を確保する必要がある。この内 WAS の可用性はクラスター、MB の可用性は HACMP™ などの高可用性ソフトウェアで実現され、事例も多く存在するため、本論文ではメッセージング関連のコンポーネントのみに注目して記述する。

5.1 WAS (ME) の高可用性

ME の基本構成は、クラスター内の 1 メンバーで ME を稼働させる Active/Stand-by 構成となる。WASv6.0 以降提供された HA マネージャーによる高可用性機能により ME の障害時にほかのクラスター・メンバー上の ME が起動される (図 3 (b) (e))。ME のメッセージ情報やトランザクションのステータスが格納されているメッセージ・ストアを外部 DB に配置し共有することで引き継ぎを可能としている。

5.2 ME-MQ 接続を実現する MQ リンクと高可用性

ME-MQ 接続は ME に MQ リンク [5] を定義することで実現できる (図 3 (c), (d))。MQ リンクにより ME に仮想の QMgr が定義され、MQ 側からは ME が MQ の QMgr として認識される。通常の MQ QMgr 同士の接続と同様、送信/受信チャンネルを定義することで MQ の QMgr と接続できる。

送信チャンネルの定義では、接続先の IP アドレスとポートを指定するため、障害時にそれらの情報が変更されると、チャンネルが切断されてしまう。そのため、障害時にもチャンネル接続を自動で再開できる設計が必要となる。

リクエスター側のアプリケーション・サーバー (以下, AS)

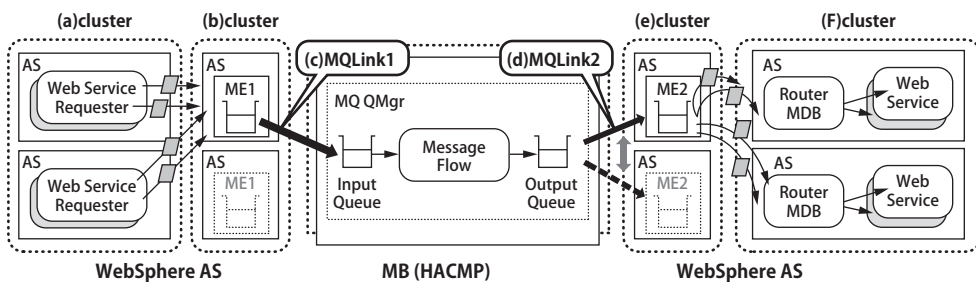


図 3. WAS-MB 構成における高可用性の実現

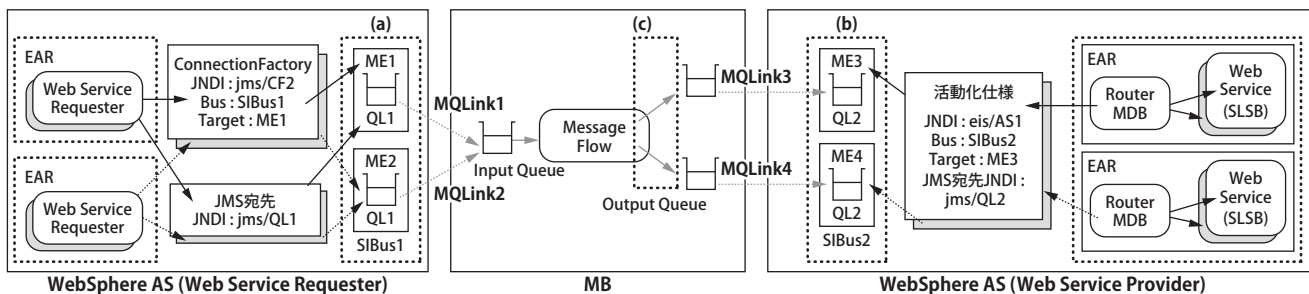


図 4. ME 拡張時の負荷分散の実現

からのリクエスト (図 3 (c)) は MB の可用性が HACMP で確保され、IP アドレスが引き継がれるため、チャンネルの再接続も自動で行われ特に問題はない。

その反面、受信チャンネルを構成しているプロバイダー側 AS の ME (図 3 (e)) フェイル・オーバーでは接続対象の ME が変更になるため、チャンネルの切断が発生してしまう (図 3 (d))。ここでは、チャンネル接続を自動回復させる方法を記述する。

① IP エイリアスを利用する方法

IP エイリアスを利用する方法 [5] を実現するには、以下の課題および制約があるため、注意が必要である。

- ポートの引き継ぎは行われなため、稼働系/待機系 ME のポートを一致させる必要がある。
- フェイル・オーバーと同じタイミングで上記ロジックを記述したスクリプトを実行する必要があるが、そのタイミングの決定が困難。

② MQ 送信チャンネルの属性を変更する方法

MQ 側送信チャンネルの接続先情報を変更する方法も有効である。この方法は MQ のサポートパックである MR01 をチャンネル Exit として実装することで実現できる。チャンネルの再試行のたびに宛先リストに記述されたサーバーへ順次つなぎ換えを行うことで、チャンネル接続の回復を可能にする [6]。

この方法は、①で言及した課題および制約はなく、基本ロジック部分は提供されているため実装に要する工数を削減できる。また実際に検証し正しく稼働することも確認できており、①と比べ優位性の高い方法であるといえる。

6. メッセージング・エンジンの拡張性の実現

5.1 で述べたように ME の基本構成ではクラスター内で

ME を一つ Active/Stand-by 形式で稼働させる。しかし大量のメッセージの送受信により一つの ME では処理能力の限界を迎えることも考えられる。その場合には複数の ME を稼働させることが必要となるが、WAS では、クラスター内に Active/Stand-by の ME を

複数構成し、見かけ上 Active/Active 形式で ME を稼働させる ME パーティショニング構成 [7] をとることが可能である。この機能による ME の拡張性を見据えた設計とその考慮点について述べる。

6.1 負荷分散の実現

ME を拡張すると、宛先となる ME が複数存在することになるため、どのように複数の ME に負荷分散を行うかが問題となる。ここでは、(1) リクエスターが ME にメッセージを送信するときの負荷分散 (図 4 (a))、および (2) MB がプロバイダー側 ME にメッセージを送信するときの負荷分散 (図 4 (b)) について述べる。

(1) リクエスター側 ME

図 4 に示すように、Web サービス・リクエスターは、JMS プロバイダーとの接続情報が定義された接続ファクトリー (以下、CF) と JMS 宛先情報を使用し、JMS プロバイダー上のキューに接続する。一方、MDB が JMS プロバイダー上のキューに接続する際は、JMS プロバイダーとの接続情報と JMS 宛先情報が定義された活動化仕様を介して接続する。

CF では、「接続の接近性」にて ME を検索する際の検索範囲を指定できる。デフォルトは「バス」であり、SIBus 内の全 ME が対象となる。接続の際はこの検索範囲の中で、より近い ME がより離れた ME に優先して選択される。

ME のフェイル・オーバー時には一時的に複数の ME が同じ接近性となる場合があるが、その場合接続する ME はランダムに決定される。そのため、一つの ME へ接続が集中する可能性が生じるが、1 度接続が確立されると、フェイル・バック後も ME の再接続は行われない。このことは、複数の ME に負荷分散されない可能性を示唆する。それを回避するには、CF にて対象となる ME を指定し、複数の CF を Web サービス・リクエスターが使い分けの必要がある。このことは、サーバー・スコープで JNDI が同じ CF を複数定義し、Web サービス・リクエスターが稼働する AS ごとに CF の接続先 ME を切り替えることで実現可能である。

(2) プロバイダー側 ME

プロバイダー側 ME を拡張した場合、MB からのメッセージの負荷分散手段が必要となる。

通常、SIBus に対して一つの MQ リンクを構成すれば、MB 側の MQ と接続することができる。受信したメッセージは SIBus 内を自動でルーティングされ、宛先のキューが定義された ME へ送られるからである。しかし、ME パーティショニング構成をとった場合、すべての ME 上に同名のキューが配置される。その場合、MQ リンクを構成した ME 上に宛先のキューが必ず存在することになり、はじめにメッセージを受け取ったその ME に負荷が集中してしまう (図 5)。そのため、ME パーティショニング構成では、すべての ME に MQ リンクを構成する必要がある (図 4 の MQLink3, MQLink4)。MB のメッセージ・フロー中に複数の Output キューにメッセージを送るロジックを追加し、複数 MQ リンクにメッセージを分散することで、ME への負荷分散が可能となる (図 4 (c))。

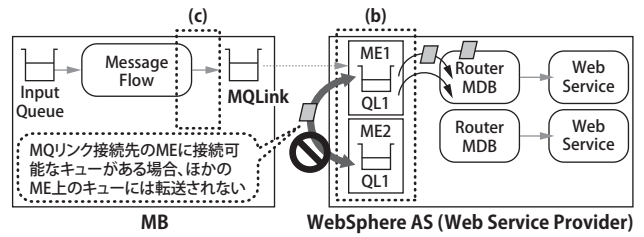


図 5. MQ QMgr と SIBus の接続

6.2 MDB を使用する構成での ME の拡張性

MDB は起動時に接続先のキューが決定されるという特性上、ME の拡張性にも影響を与えるため注意が必要となる。ME の拡張性に関して、パーティショニングという観点で語られている資料は多いが、MDB と組み合わせで使用した場合の構成について言及された資料がない。ここでは MDB の拡張性にも触れながら、考慮点について述べていく。

(1) コンポーネントごとの拡張性の実現 -ME/MDB リモート配置

一般的に、メッセージング処理を実現する ME とビジネス・ロジックを処理する MDB では拡張要件は異なると考えられ、それぞれのニーズに応じた拡張が求められる。現時点でのガイド [7] では、ME/MDB の同一 AS 上での稼働 (図 6 (a)) が推奨されているが、この構成では ME のローカルにある MDB のみがメッセージを取得し、クラスター内で稼働しているほかの MDB ではメッセージを処理できなくなるため、クラスター内の MDB を有効に使用することができないという重大な欠点がある。ME/MDB を別 AS 上に稼働させた場合 (図 6 (c))、上記欠点はなく、クラスター内の MDB すべてでメッセージを処理できる。さらにニーズに応じた拡張が可能となるため、利点が大きい構成であるといえる。

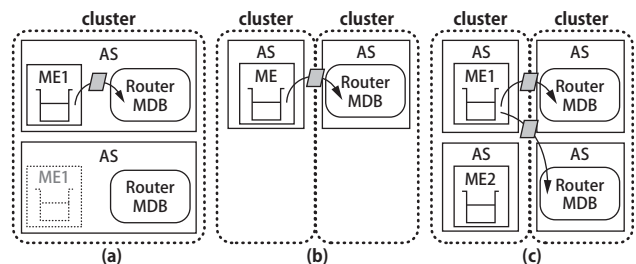


図 6. ME/MDB の構成

(2) ME ローカル/リモート配置によるパフォーマンス比較

別 AS 上で稼働させる場合の最も大きな懸念事項として、プロセス間の通信によるパフォーマンス劣化が挙げられる。そこで、図 6 に示す三つのパターンについて、MDB の処理時間を計測した。矢印が検証時のメッセージの流れを表す。(a) は同一 AS 上で稼働させた場合の構成、(b)、(c) は別 AS 上に稼働させた場合の構成である。別 AS 上に稼働させた場合、(c) のように複数の MDB により並行処理

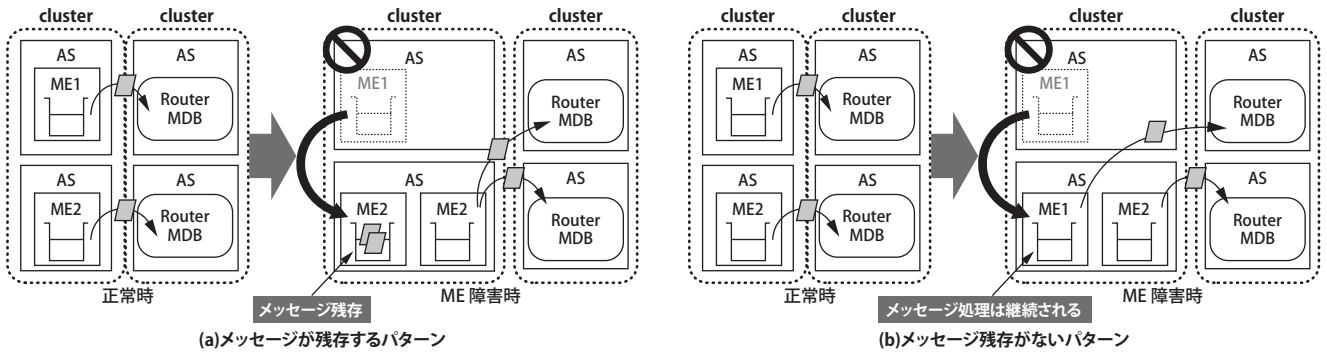


図 7. ME 障害時の挙動

させることが可能であるが、(a) との比較を容易にするため、(b) の構成でも検証を行った。検証では、ME にメッセージを 10,000 件滞留させ、MDB が全件処理する時間を計測した。なお、MDB の同時並行稼働インスタンス数を示す最大並行エンドポイント数は「10」としている。

表 1. MDB の 10,000 件メッセージ処理時間合計

MDB 内の処理	構成 (a)	構成 (b)	構成 (c)
(A) 処理なし	37sec	86sec	96sec
(B) sleep2 秒	32min 32sec	33min 36sec	16min 59sec

表 1 は測定結果である。(A) は MDB で何も処理を行わない場合の結果を示しているが、リモート構成になることで 2 倍以上のパフォーマンス劣化があることが分かる (a と b)。MDB の数が増えるとパフォーマンスが落ちているが (b と c)、これは一つのキューに複数の MDB がアクセスすることによる競合が原因であると考えられる。(B) では MDB に 2 秒の sleep を入れており、より実際の処理に近いケースとなっている。このケースでは、c が約 1/2 と圧倒的に短く、リモート接続によって発生していたパフォーマンス劣化は、全体の処理時間を考えると非常に小さいことが分かる。これは、MDB の処理時間が加わったことによって、キューへのアクセス時の競合が緩和されることも一因となっている。

以上の検証から、ME/MDB が別 AS 上で稼働する構成におけるパフォーマンス上の懸念はないと判断できる。前述したようにリモート構成における利点は大きく、それらを受受したい場合こちらの構成を採用されたい。

6.3 メッセージ残存回避の実現

MDB を使用する構成の場合、メッセージの残存を回避するため、障害時においてもすべての ME が MDB から接続されていることが必須となる。ここでは、ME/MDB を別 AS 上で稼働させた構成での (1) ME が稼働する AS、(2) MDB が稼働する AS、それぞれの障害時におけるメッセージ残存回避方法について記述する。

(1) ME が稼働する AS 障害時

ME が稼働する AS の障害時には、フェイル・オーバー後の ME へ接続する MDB が必要となる。MDB の接続対象となる ME は活動化仕様により決定されるが (図 4)、デフォルトでは、CF と同様「接続の接近性」から決定される。ME がフェイル・オーバーすると、MDB からは同じ接近性の ME が複数あることとなるが (図 7 ME 障害時の ME1, ME2)、その場合、フェイル・オーバー後の ME が MDB に接続されない可能性がある (図 7 (a))。

これを回避するには、WASv6.1 の機能拡張である活動化仕様の「ターゲット」プロパティの使用が有効である。これにより接続先 ME の指定が可能となるため、MDB が接

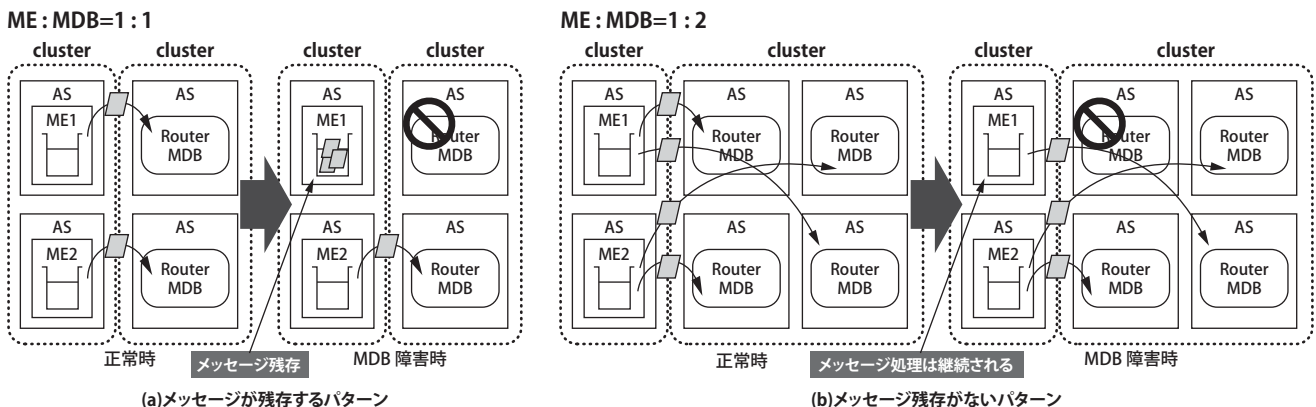


図 8. MDB 障害時の挙動

続する ME が固定され、フェイル・オーバー後もメッセージ処理が可能となる (図 7 (b)).

(2) MDB が稼働する AS 障害時

一つの ME に対し二つ以上の MDB が接続する構成とすることで、MDB が稼働する AS の障害時にも、ME 上のメッセージ処理を継続することが可能となる。図 8 (b) に示すように、ME 拡張時には、必ず MDB が稼働する AS も同時に追加し、一つの MDB が障害を起こしてもほかの MDB により処理される構成にする。この場合、ノード障害も考慮し、二つの MDB は別ノードに配置することが望ましい。

7. おわりに

本論文では Web サービス連携において SOAP/JMS を通信プロトコルとする非同期処理を行う上で、ESB として MB を使用し、複数ある WAS-MB の接続パターンを整理した。さらに ME を採用する方式の利点を示し、このパターンにおける高可用性・拡張性を実現するための指針や考慮点について説明を行った。

なお本論文で紹介した実装方式は、可能な限り要件をシンプルにかつ確実に実現できる手法である。またこれらの手法をすべて検証し、本番業務に十分耐え得ることを実証した。

SOAP/JMS を使用した非同期連携が、SOA の浸透に十分貢献できる技術であることは論じてきたとおりであり、W3C による SOAP/JMS 標準化 [3] が進められるに伴いますますこの処理形態の採用も活性化すると考えられる。本論文が採用の活性化を支え、メッセージング基盤設計における一つの有用な指針となると考えている。

参考文献

- [1] Sun Developer Network, <http://java.sun.com/products/jms/> (2005-2-3).
- [2] Patterns:ServiceOriented Architecture and Web Services, <http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf> (2004-4).
- [3] SOAP over Java™ Message Service 1.0, <http://www.w3.org/Submission/SOAPJMS/> (2007-10-26).
- [4] 長妻令子: "分散サーバー上でのリアルタイム処理," ProVision, No.55, pp.101-107 (2007).
- [5] WAS V6 SIBus 構成ガイド,
<http://www.ibm.com/jp/software/websphere/developer/was/wv6/sibus/soapjms/> (2006-2-15).
- [6] WASv6.1 SIBusとWebSphereMQを繋ぐMQリンクの高可用性について (WAS-07-028),
<http://www-06.ibm.com/jp/domino01/mkt/cnpages1.nsf/page/default-0041D082> (2007-8-8).
- [7] WAS6.1MDBを使用したME高可用性における考慮点 (WAS-07-013), <http://www-06.ibm.com/jp/domino01/mkt/cnpages1.nsf/page/default-000B3D41>



日本アイ・ビー・エム株式会社
 ITS 製造ソリューションサービス
 製造第2サービス
 ITスペシャリスト

石橋 香代子 Kayoko Ishibashi

[プロフィール]

2005 年、日本 IBM 入社。電機メーカー様を中心とした製造業のお客様担当 SE として、UNIX 基盤系のデリバリーを担当。主に、SOA 関連の WebSphere 製品を使用したプロジェクトに参画し、インフラストラクチャー・アーキテクチャーの設計・構築に従事している。

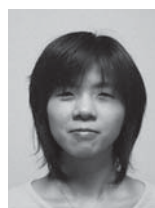


日本アイ・ビー・エム
 システムズ・エンジニアリング株式会社
 ソフトウェアテクノロジーセンター
 Webインフラストラクチャー
 主任ITスペシャリスト

平岩 梨果 Rika Hiraiwa

[プロフィール]

2001 年、日本 IBM 入社。入社以来、石油化学系・電機メーカーを中心とした製造業のお客様担当 SE としてデリバリーを担当。現在は SOA 関連の WebSphere 製品をサポートする活動や、SOA プロジェクトにおける ESB (WESB) アプリケーションの設計・構築に従事している。



日本アイ・ビー・エム
 システムズ・エンジニアリング株式会社
 ソフトウェアテクノロジーセンター
 ビジネス・インテグレーション
 主任ITスペシャリスト

神野 稚子 Wakako Jinno

[プロフィール]

1997 年、日本 IBM 入社。WebSphere Application Server 担当として多数の Web システム構築に携わる。2006 年より現在の BPM ソリューションのベースとなるワークフロー製品の担当となり、SOA ベースの BPM ソリューションの設計・構築に従事している。