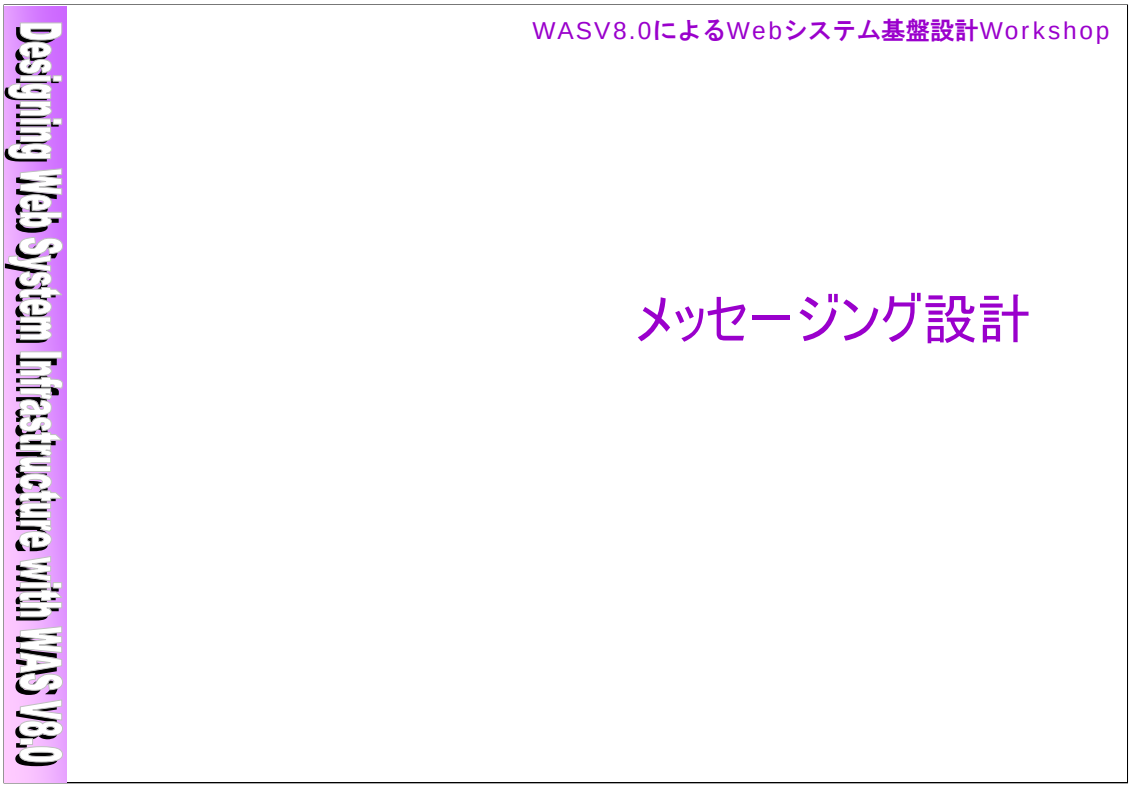

Designing Web System Infrastructure with WAS V8.0
WASV8.0によるWebシステム基盤設計Workshop
メッセージング設計

Designing Web System Infrastructure with WAS V8.0

当セッションの目的

- WASとメッセージング・サービスとの接続に必要な知識の習得
  - メッセージング技術の基礎を理解する。
  - JMSおよびMDBをWAS上で動かすために必要なメッセージング基盤設計のポイントを理解する。
  - WASが提供するメッセージング・サービスの仕組みを理解する。
  - WebSphere MQへの接続方法およびトポロジー構成について理解する。

© 2012 IBM Corporation

2

Designing Web System Infrastructure with WAS V8.0

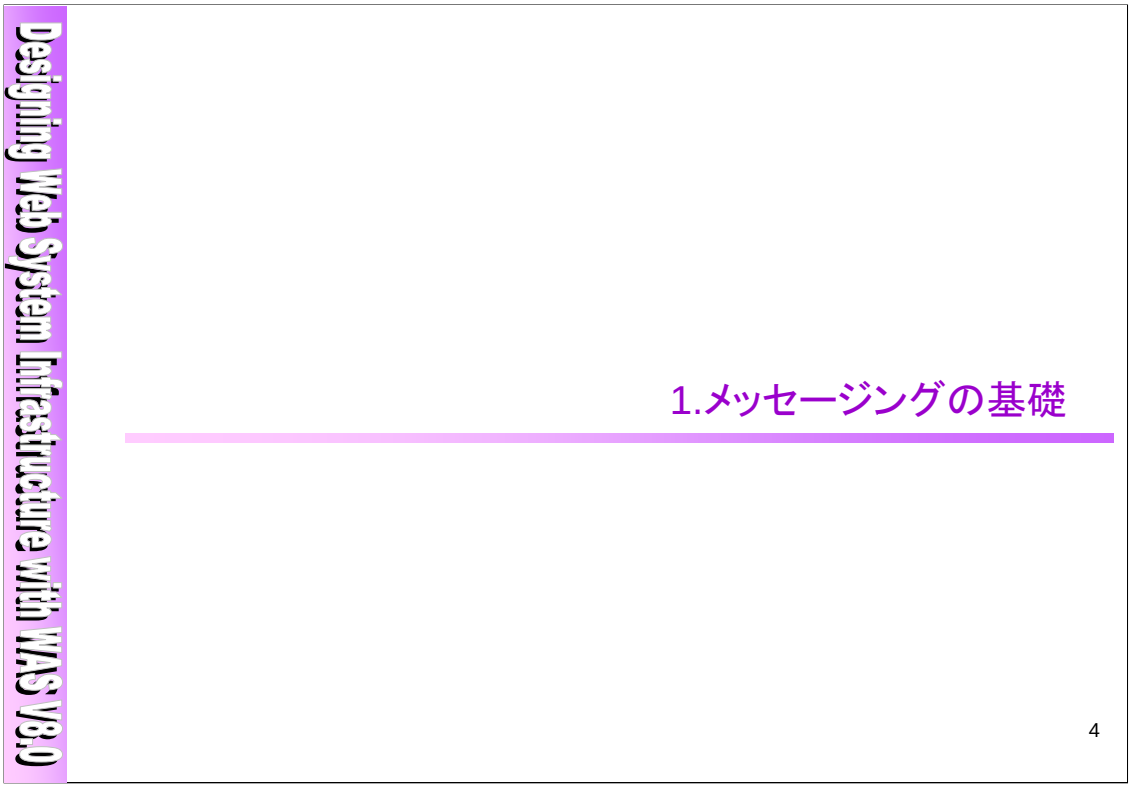
Agenda

1. メッセージングの基礎
2. JMS (Java Message Service)
3. MDB (Message Driven Bean)
4. サービス統合バス(SIBus)とメッセージング・エンジン
5. WebSphere MQ との接続
6. WebSphere MQとSIBusの選択基準

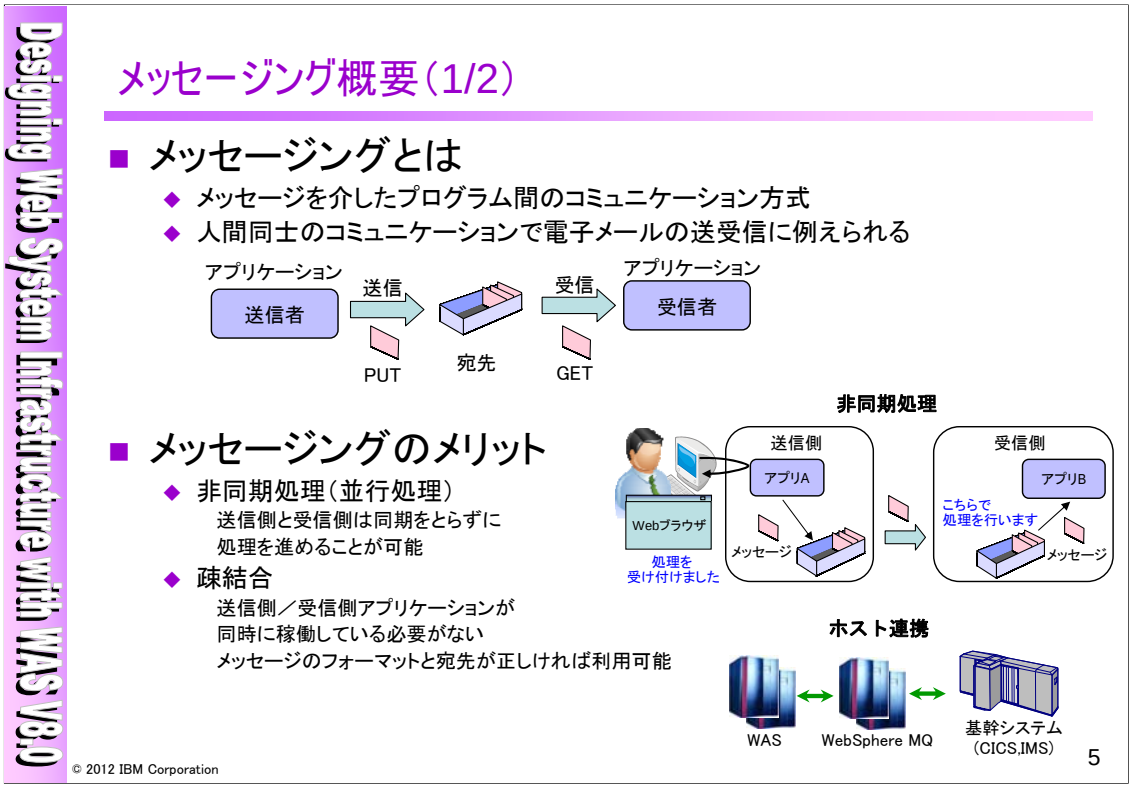
まとめ・参考文献

© 2012 IBM Corporation

3



この節では、メッセージングの基礎について解説します。



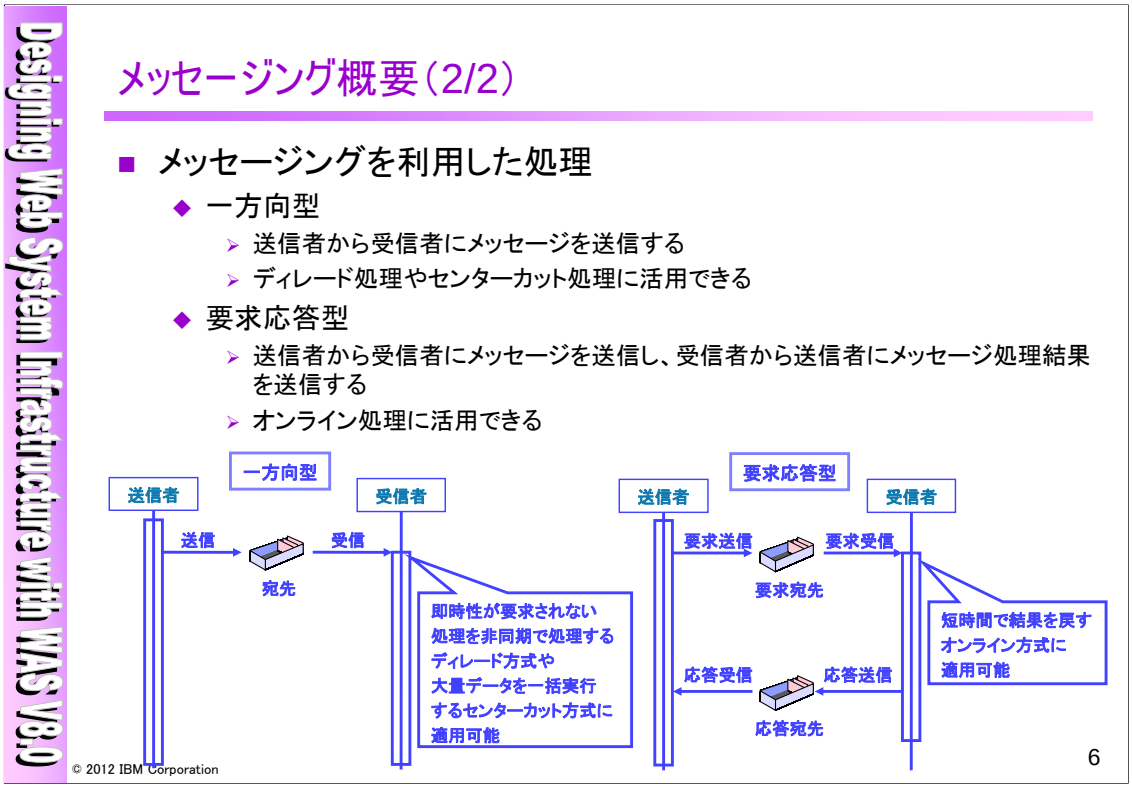
キューを介したメッセージングを活用すると、同期処理では難しかったいくつかの処理が簡単になります。

・並行処理

メッセージングを使うと、メッセージ送信側と受信側では同期をとりません。メッセージ送信側はメッセージ送信後、処理を継続できますし、メッセージ受信側はメッセージ受信後、すぐに処理を開始します。

・疎結合

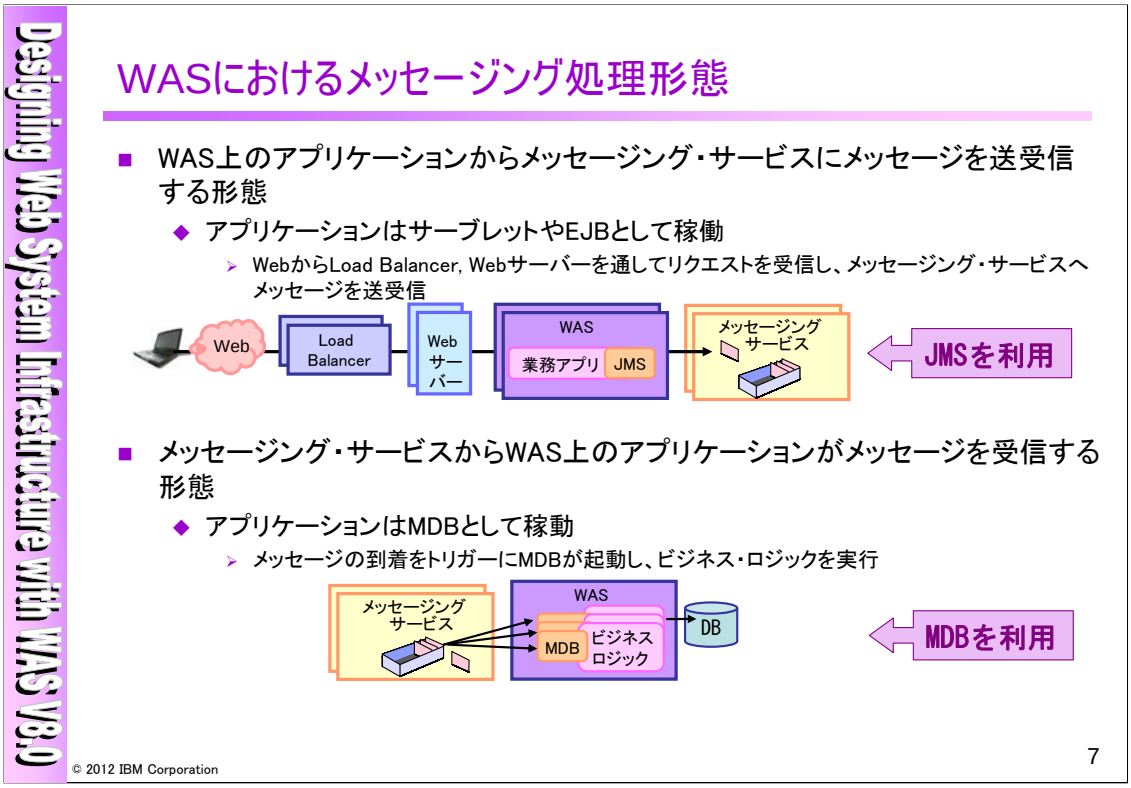
送信アプリケーションと受信アプリケーションが従うべき規約が少ないことです。メッセージのフォーマットと利用する宛先が正しければ、利用が可能になるというメリットがあります。この疎結合という考え方が、ESB（Enterprise Service Bus）でも活用されています。ESBについては参考資料の「SIBusとESB」を参照してください。



メッセージングの利用形態は様々ですが、つきつめると、一方向型のメッセージングと、要求応答型のメッセージングに分類することができます。

一方向型は、サービスを呼び出しておき、後から結果を取得するようなディレード(遅延)処理や、大量件数を処理するセンターカット(一括)処理に活用できます。

要求応答型は、受信者(サービス)が送信者(リクエスター)に処理結果を戻すような、オンライン処理に活用できます。ただし、メッセージ伝達の負荷を考えると、EJBやWebサービスと比較して、パフォーマンスは劣るかもしれません。パフォーマンスの観点からメッセージングを利用して要求応答を行うメリットを生かせるのは、送信側と受信側が並行して処理を行うような場合です。ただし、送信者側と受信者側のトランザクションが分離してしまうので、障害ケースのインダウト処理について、入念に検討する必要があります。



WASにおけるメッセージング処理形態は2つの形態に分類することができます。

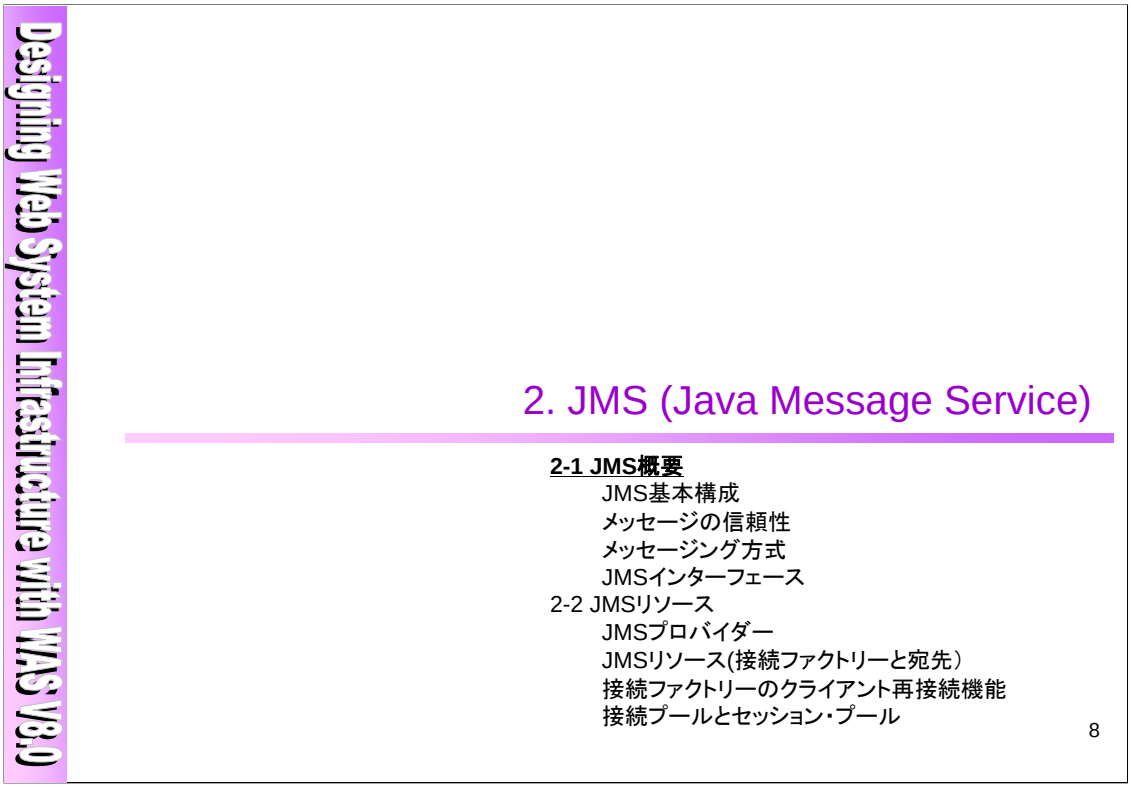
(1) WAS上で稼働するアプリケーションからメッセージング・サービスにメッセージを送信する形態 (JMS)

Webブラウザーから負荷分散装置、Webサーバーを経由してリクエストを受け、WebSphere MQ(MQ)などのメッセージング・サービスへメッセージを送受信します。WAS上のアプリケーションはサーブレットやEJBとして稼働し、JMSインターフェースを利用します。

(2) メッセージング・サービスからWAS上で稼働するアプリケーションがメッセージを受信する形態 (MDB)

メッセージング・サービスへのメッセージ到着をトリガーにしてWAS上のアプリケーションが稼働してビジネス・ロジックを実行します。

アプリケーションはメッセージ駆動型のEJB であるMDBを利用します。

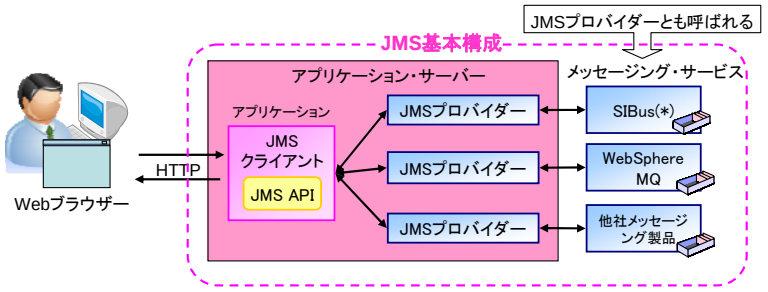


この節では、JMSについて説明します。

Designing Web System Infrastructure with WAS V8.0

JMS概要(1/4)　JMS基本構成

- JMS (Java Message Service)とは・・・・
  - Java EE標準のメッセージング・インターフェイス
    - WAS V8.0はJava EE6.0に準拠しており、JMS1.1が利用できる
  - JMSで実装するとメッセージング・サービスに依存しないアプリケーションを構築可能
  - JMSクライアント：JMSインターフェースによりメッセージング・サービスを利用するアプリケーション
  - JMSプロバイダー：JMSインターフェースを提供するJava EE側のコンポーネント
  - メッセージング機能はメッセージング・サービスが独自に実装可能
    - SIBus（サービス統合バス）はWASが提供するメッセージング・サービス



(*)Service Integration Bus(サービス統合バス)の略。WAS上で稼動する

© 2012 IBM Corporation

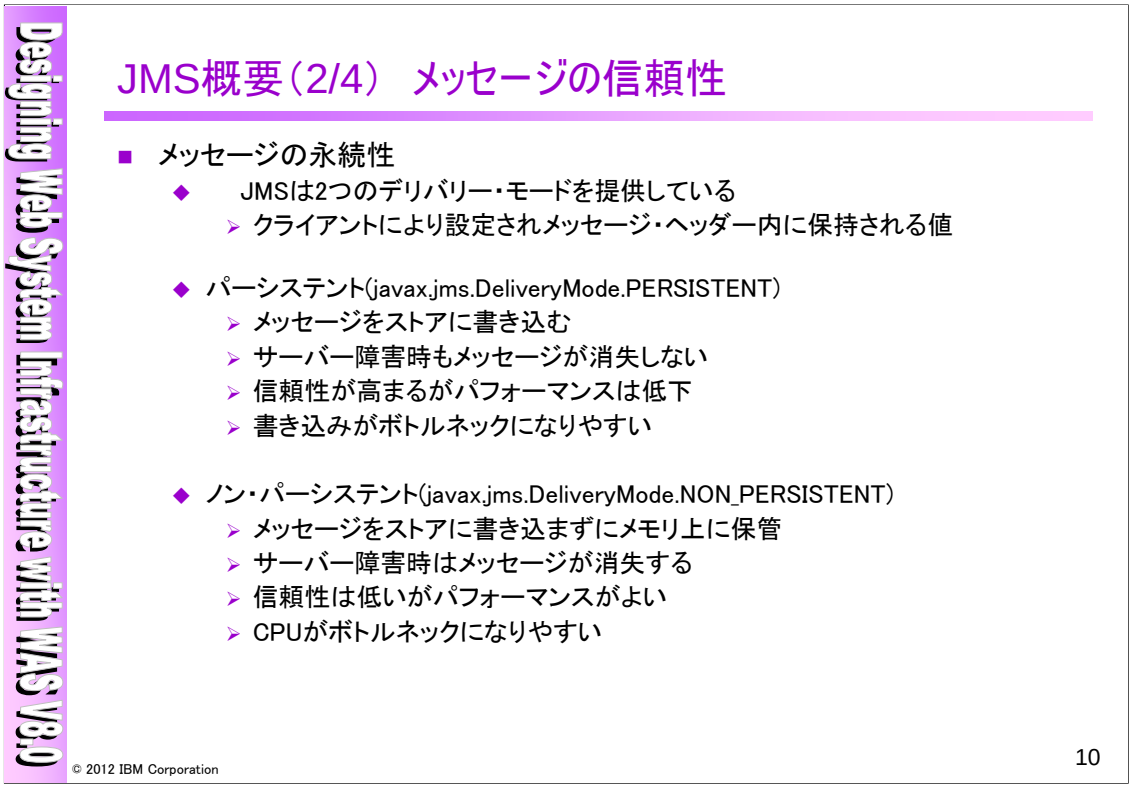
9

JMSはJava EE標準のメッセージング・インターフェイスです。標準のインターフェイスでコーディングすることにより、メッセージング・サービスに依存しないアプリケーションを開発できます。

JMSインターフェイスを利用してメッセージング・サービスを利用するアプリケーションを、JMSクライアントと呼びます。

JMSプロバイダーとはJMSのインターフェースを提供するJava EE側のコンポーネントを指していて、アプリケーションがどのメッセージング・サービス上の宛先に対して送受信を行うかという接続機能を提供します。MQやSIBusなどのメッセージング・サービスそのものをJMSプロバイダーと呼ぶ場合もあります。

JMSはインターフェイス定義が中心で、メッセージングの機能については触れていません。その為、メッセージング機能の実装は、各ベンダーに任されています。

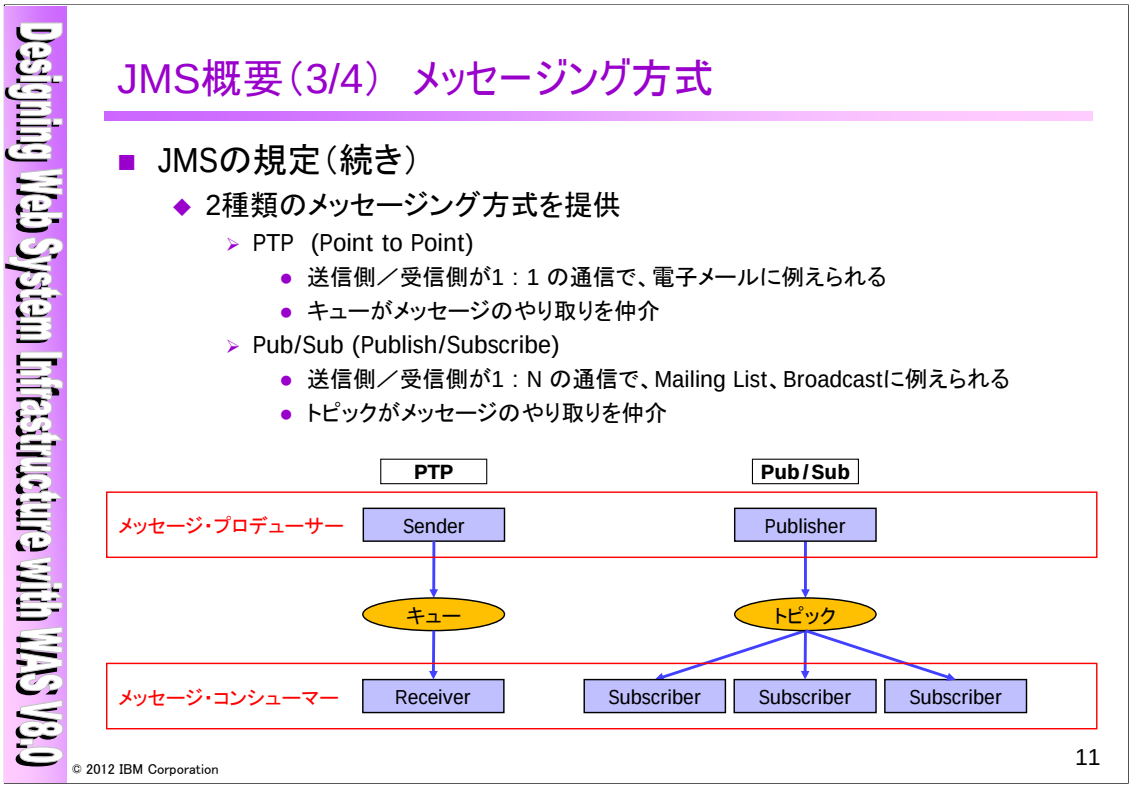


また、メッセージの信頼性は、パフォーマンスに関わる重要な選択です。

・パーシステント：メッセージをストア(*)に書き込むことで、高い信頼性を提供します。これに対して、ストアに書き込む分、メッセージングのパフォーマンスが低下します。一般的に、ストアへの書き込みがボトル・ネックになりやすい設定です。

・非パーシステント：メッセージをストア(*)に書き込まないので、メッセージを保持しているサーバーに障害が発生した場合には、メッセージを消失する可能性があります。信頼性が低い代わりに、メッセージングのパフォーマンスは向上します。一般的に、CPUがボトルネックになりやすい設定です。

(*)WASのSIBus(メッセージング・エンジン)のメッセージ・ストアにはデータ・ストアとファイル・ストアの2種類があります。（後述）



JMSでは、2種類のメッセージングを利用するための、インターフェイスを定義しています。

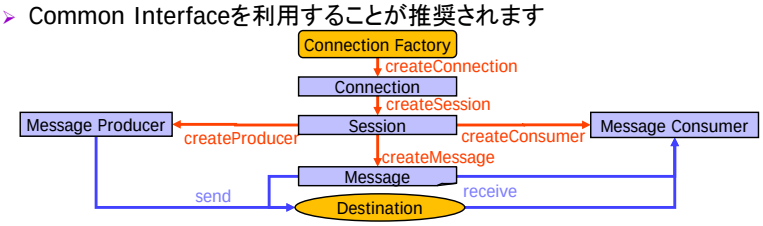
PTP(Point To Point)は、メールに例えられる1対1の通信形態です。メッセージのやり取りはキュー(Queue)を介して行われます。送信者(Sender)は、キューに対してメッセージを送信します。受信者(Receiver)は、必要なときにキューからメッセージを受け取り、処理を行います。

Pub/Sub(Publish and Subscribe)は、メーリング・リストに例えられる1対多の通信形態です。発行者(Publisher)は、トピック(Topic)に対してメッセージを送信します。全ての購読者(Subscriber)は、前もってトピックに購読依頼を出しておくことで、メッセージを受け取り、処理を行います。

Designing Web System Infrastructure with WAS V8.0

JMS概要（4/4）　JMSインターフェース

- JMS 1.1のインターフェース　・・・WAS V6.x/V7.0/V8.0が準拠
  - JMS 1.0.2までは、PTP型とPub/Sub型固有インターフェイスでコーディング
  - JMS 1.1からCommon Interfaceにより、宛先の種類に依存しないコーディングが可能
    - PTP型とPub/Sub型を1UOW（作業単位）で扱うことが可能
    - Common Interfaceを利用することが推奨されます



JMS Common Interface	PTP specific Interface	Pub/Sub specific interface
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver,QueueBrowser	TopicSubscriber

© 2012 IBM Corporation

12

WAS V6.x/7.0/8.0では、JMS 1.1で定義された、JMS Common Interfaceを利用することで、PTPであるか、Pub/Subであるかに依存しないアプリケーションを開発できます。JMSを利用する手順は概略以下の通りです。

・接続ファクトリー(ConnectionFactory)：

JMSプロバイダーの接続を管理するクラスです。アプリケーションからはlookupすることで取得できます。

・宛先(Destination)：

メッセージング・サービス上のキューやトピックのような、メッセージング・ポイントを表すクラスです。

・接続(Connection)：

メッセージング・サービスとの接続を表すクラスです。接続ファクトリーから取得できます。

・セッション(Session)：

メッセージング・サービスとのやり取りを管理するクラスです。接続から取得できます。トランザクションのコミット/ロールバックなども管理します。また、メッセージ供給者(MessageProducer)、メッセージ消費者(MessageConsumer)、各種メッセージ・クラスのファクトリーの役割も果たします。

・メッセージ供給者(MessageProducer)：

送信アプリケーションが利用するインターフェイスです。

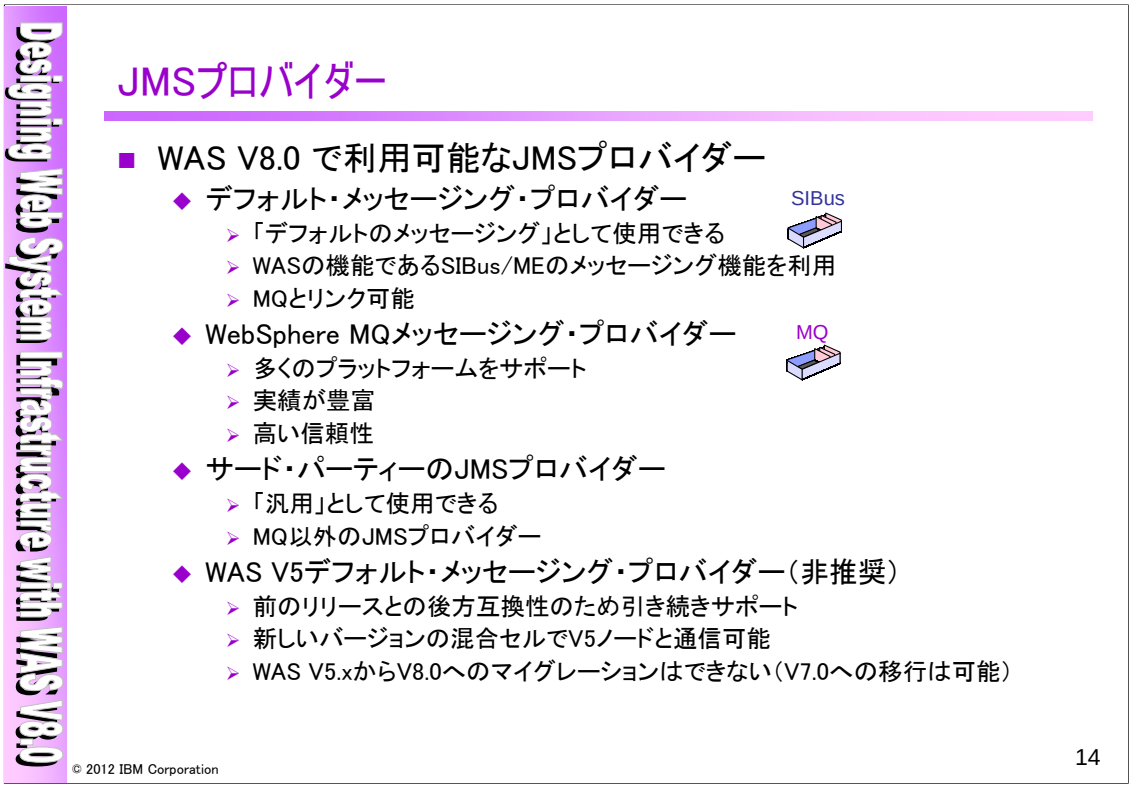
・メッセージ消費者(MessageConsumer)：

受信アプリケーションが利用するインターフェイスです。

・メッセージ(Message)：

メッセージングに利用するメッセージ本体です。Messageにはいくつかの種類があり、目的によって使い分けることができます。

続きまして、WAS上でJMSクライアントを稼働させるため、WASの基盤側で必要なJMSリソース設定についてお話します。



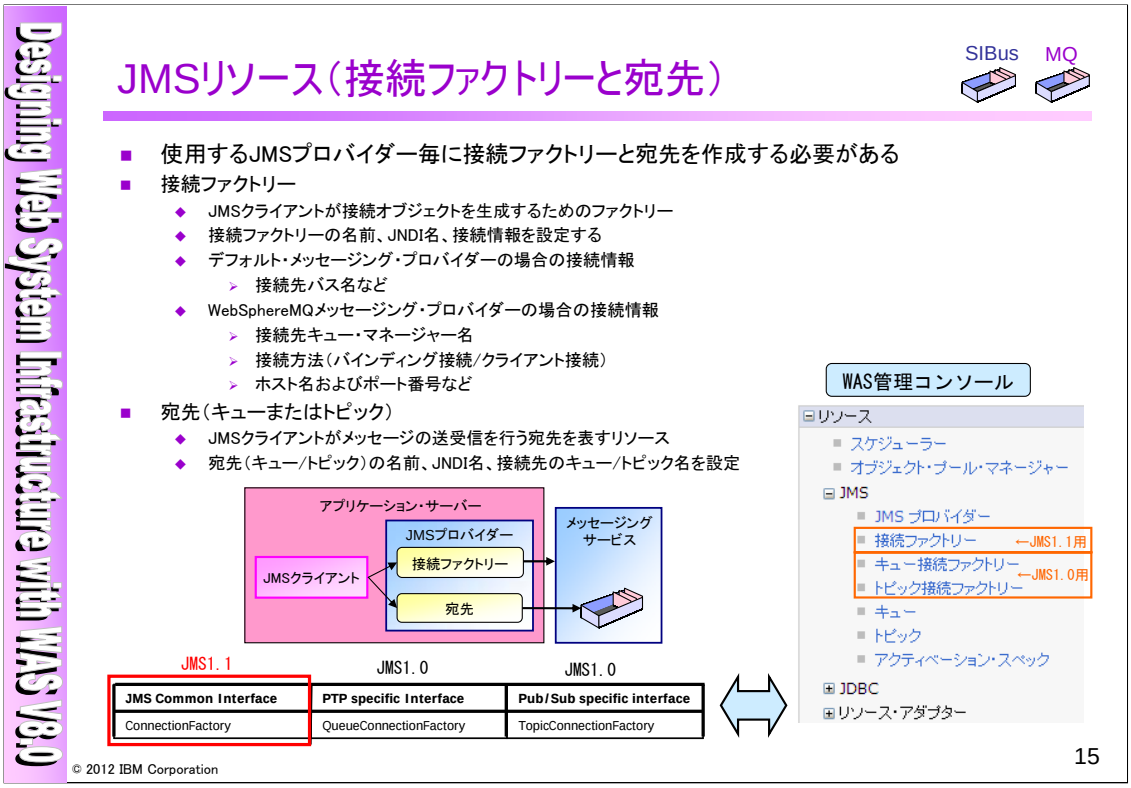
WAS V8.0で利用可能なJMSプロバイダーです。

・SIBus(Service Integration Bus：サービス統合バス)のメッセージング機能
「デフォルトのメッセージング」は、WAS V8.0で利用可能なSIBusのメッセージング機能を利用します（SIBusについては後述）。WAS V8.0はJava EE 6.0を実装しているので、Java EE 6.0に含まれるJMS1.1を利用可能です。

WebSphere MQ(以下MQ)は、多くのプラットフォームをサポートするIBM製品です。豊富な実績と、高い信頼性が評価されています。

JMS1.1 仕様をサポートする任意のサード・パーティー製メッセージング・プロバイダーを構成することもできます。

V5 デフォルト・メッセージングは、WAS V5.xの組み込みメッセージングです。



JMSクライアントをWAS上で稼働させる際に必要な構成として、WAS管理コンソール上でJMSリソースを設定する必要があります。尚、wsadminコマンドでも設定できます。

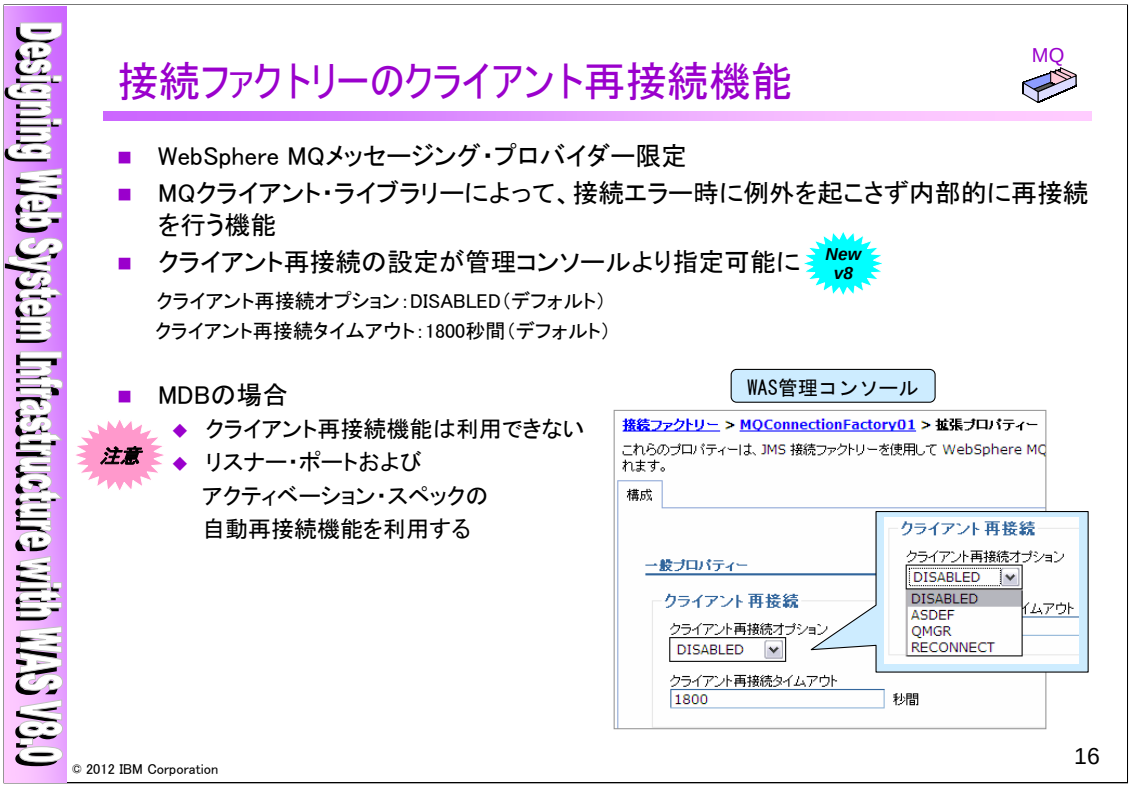
接続先のメッセージング・サービスに対応したJMSプロバイダーに対して、接続ファクトリーおよび宛先を作成します。

接続ファクトリーはJMSクライアントが接続オブジェクトを生成するためのファクトリーです。接続ファクトリーの定義では、名前とJNDI名や接続情報を指定します。接続ファクトリーで指定する接続情報についてはJMSプロバイダー毎に異なります。

宛先はJMSクライアントがメッセージの送受信を行う宛先を表すリソースです。メッセージング方式に応じてキューまたはトピックを作成します。

宛先の定義では、名前とJNDI名、キュー名を指定します。

JMSクライアントは接続ファクトリーと宛先をJNDIで指定することでメッセージの送受信を行います。



WebSphere MQ メッセージング・プロバイダーの接続ファクトリーにはクライアント再接続機能が提供されています。

このクライアント再接続機能はMQクライアント・ライブラリーによって提供されており、接続エラーが発生した際に例外を起こさずに内部的に再接続を行います。

この機能を利用することでアプリケーションにエラーが返らず、同一または異なるキュー・マネージャーへ再接続が可能となります。

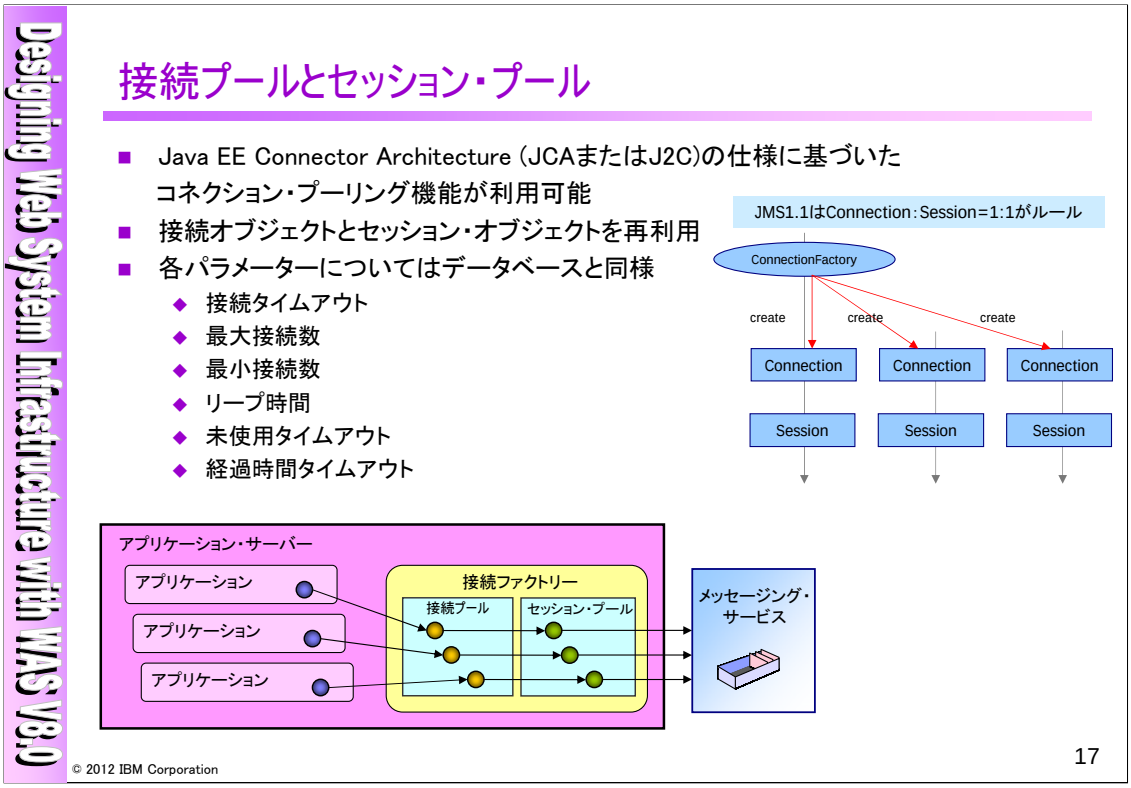
クライアント再接続オプションは以下から選択します。

DISABLED：自動再接続なし

ASDEF：mqclient.iniの設定に従う

QMGR：最初に接続されていたキュー・マネージャーにのみ自動再接続

RECONNECT：自動再接続を行う

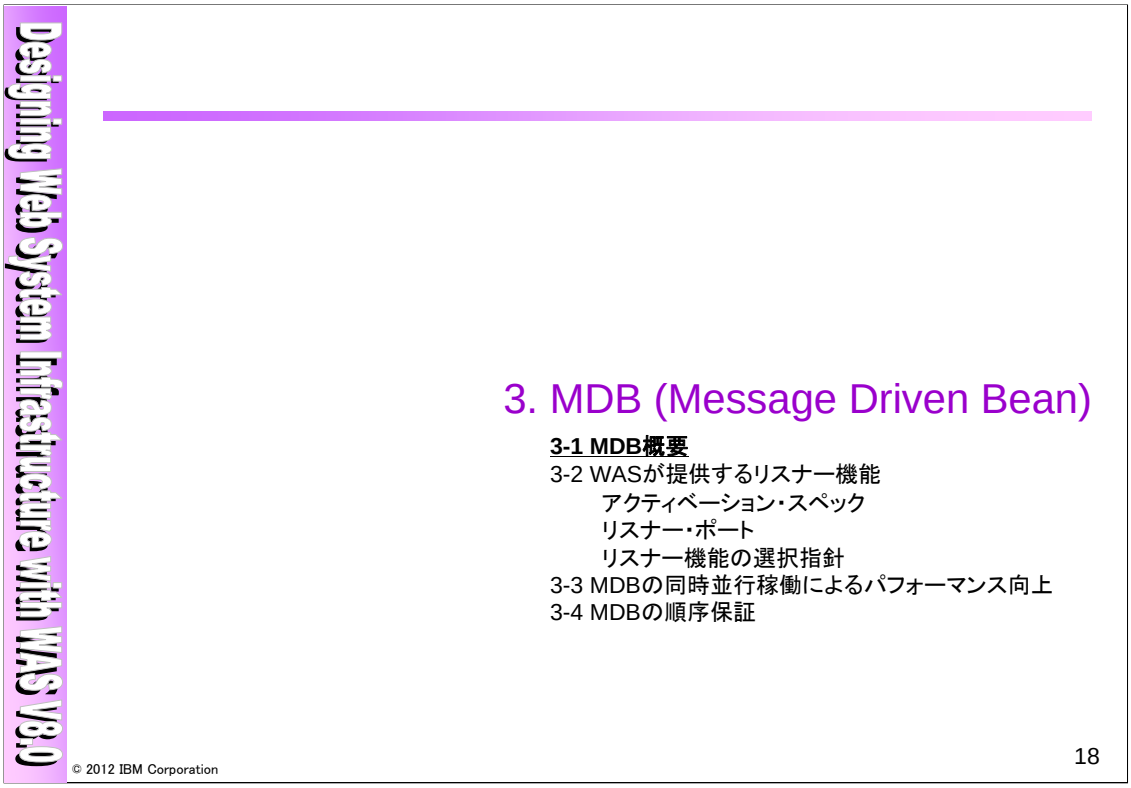


JMSではDB接続の場合と同様にJavaEE Connector Architecture (JCAまたはJ2C)の仕様に基づいたコネクション・プーリング機能が利用可能です。

接続オブジェクトは接続プールに保管し、セッション・オブジェクトはセッション・プールに保管され再利用されます。

コネクション・プーリングにおいて指定する各パラメーターはデータベースと同様です。

接続タイムアウト
最大接続数
最小接続数
リープ時間
未使用タイムアウト
経過時間タイムアウト

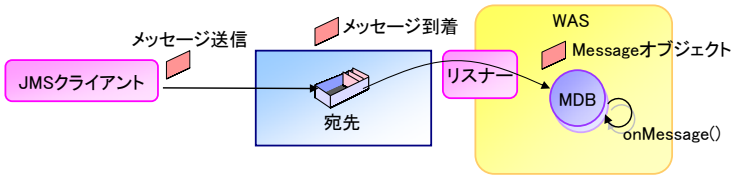


この節では、MDBについて説明します。

Designing Web System Infrastructure with WAS V8.0

MDB概要

- MDB(Message Driven Bean)とは・・・・
  - Java EE標準のメッセージ駆動型処理を行うEJB
    - Web環境で非同期メカニズムを実現ために利用される
    - 宛先にメッセージが到着した時にEJBコンテナーによって呼び出される
      - 他のEJBとは異なり、EJBクライアントから直接呼び出すことはできない
    - 並列にメッセージ受信およびビジネス・ロジックの実行が可能
      - ビジネス・ロジックは、void onMessage(Message) に記述
      - 受信メッセージは、Message オブジェクトとして参照可能
    - EJBコンテナーが起動時に複数インスタンスを作成し、プールすることが可能
  - トランザクション管理機能のサポート
    - コンテナー管理トランザクション（CMT）
      - コンテナーが自動的にトランザクションを開始、終了する
      - メッセージ受信とビジネス・ロジック内のリソース更新とを1UOW（作業単位）として処理できる
    - ビーン管理トランザクション（BMT）
      - コーディングで明示的にトランザクションの開始、終了を制御する

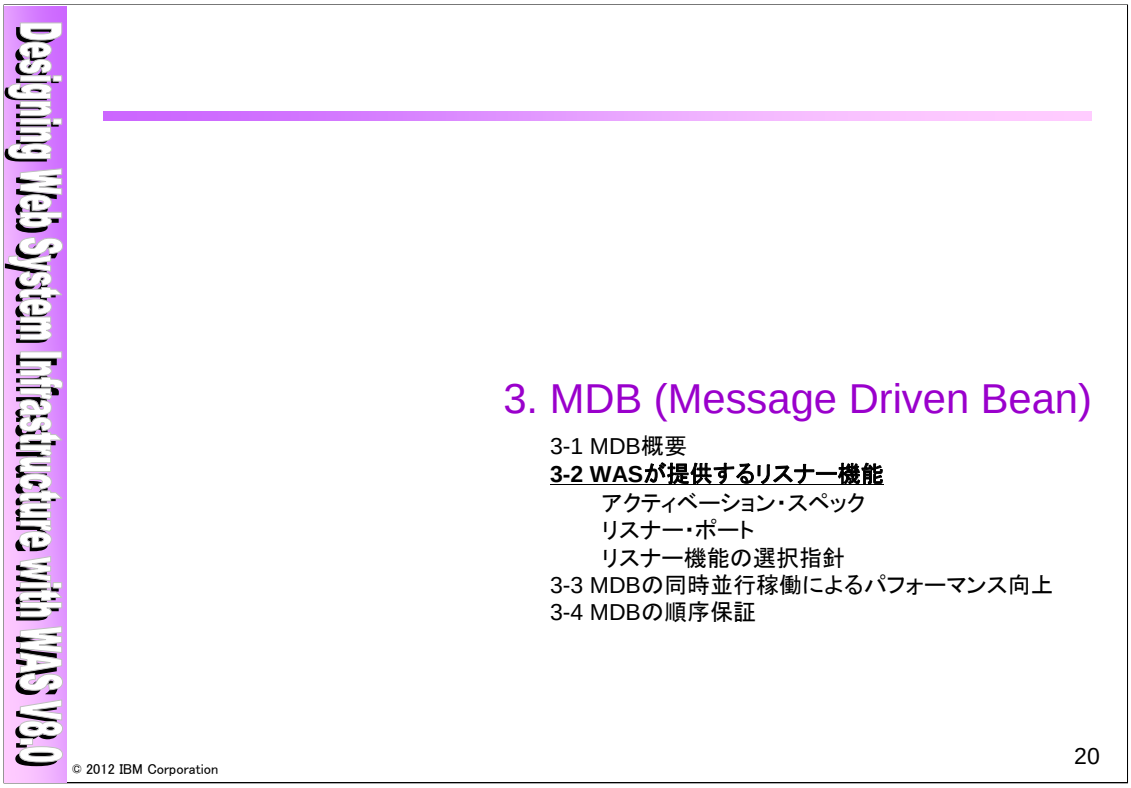


© 2012 IBM Corporation

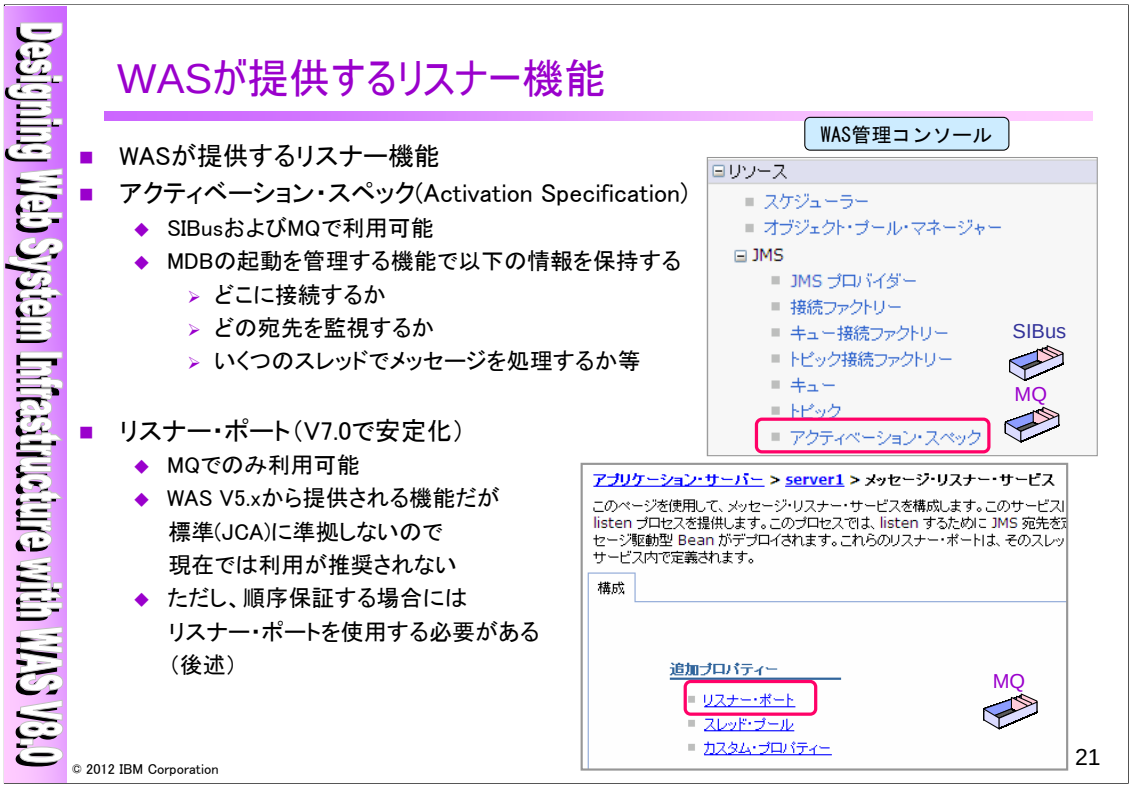
MDBは、Java EE標準のメッセージ駆動型処理を行うEJBです。JMSの宛先にメッセージが入ると、EJBコンテナーによりonMessage()メソッドが呼び出されます。複数のMDBを起動し、並行してメッセージを処理することができます。一度作成されたMDBインスタンスは、EJBコンテナーがプーリングするので、次回起動時の初期化処理を短縮されます。

また、MDBはトランザクションをサポートしており、CMTとBMTのどちらでも動作させることができます。

参考資料に「サンプル・コード（MDB）」、「MDBのデプロイ方法」がありますので、参照してください。

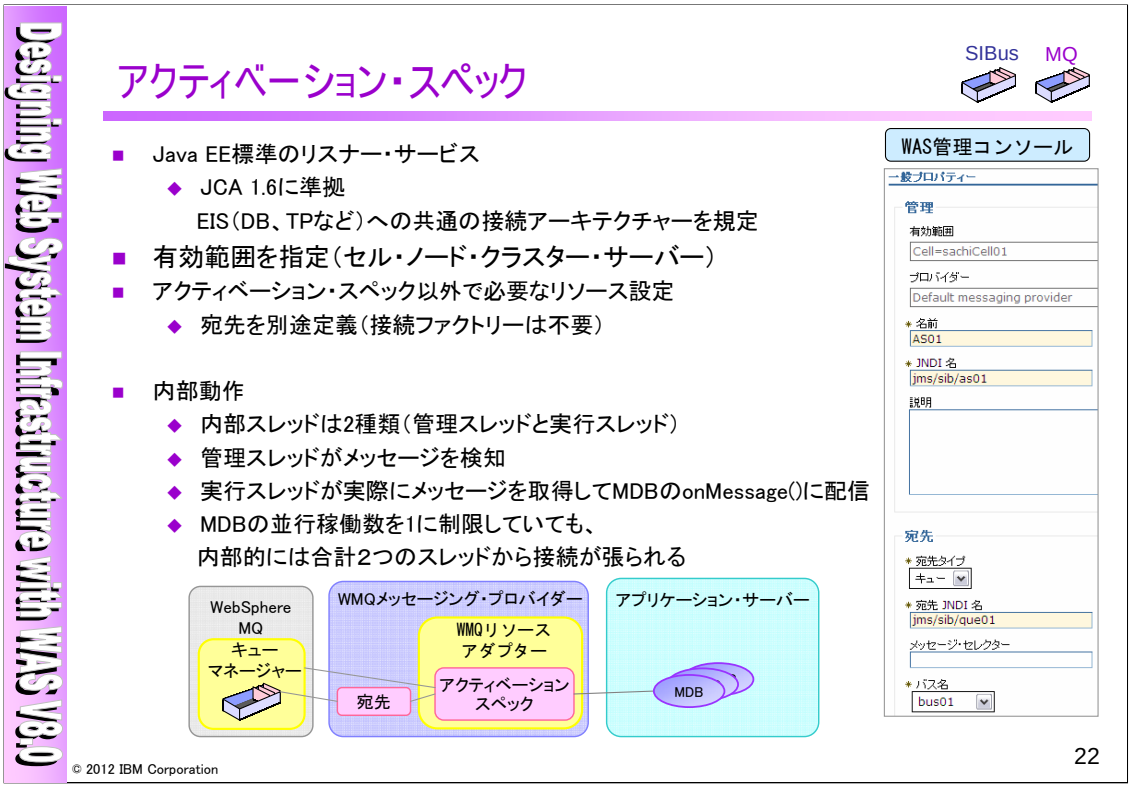


次にWASが提供するリスナー機能について説明します。



MDBがどのように起動するかという設定は、リスナー機能に記述します。
WASが提供するリスナー機能としてアクティベーション・スペックとリスナー・ポートがあります。

V7.0リリース時点において、リスナー・ポートは「非推奨」機能でしたが、その後「安定化されたフィーチャー」に変更されました。
「安定化されたフィーチャー」は「非推奨」とは異なり、将来のバージョンでも引き続き提供される機能です。
詳細については、以下リンク先を参照してください。
WAS V8.0 Information Center: 安定化されたフィーチャー
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rmig_stabfeat.html



アクティベーション・スペックはJava EE標準のリスナー・サービスです。

Java EE Connector Architecture 1.6 に準拠しています。JCAはメッセージングに限らずEIS(Enterprise Information System)への共通の接続アーキテクチャーを規定しています。

アクティベーション・スペックはセル・ノード・クラスター・サーバーの有効範囲を指定した上で、名前、JNDI名、宛先JNDI名、宛先タイプ、接続情報等を設定します。

WASが提供しているメッセージ基盤のサービス統合バス(SIBus)の場合は、アクティベーション・スペックを利用する必要があります。後述のリスナー・ポートは使用できません。WebSphere MQはアクティベーション・スペック、リスナー・ポートの両方を利用できます。

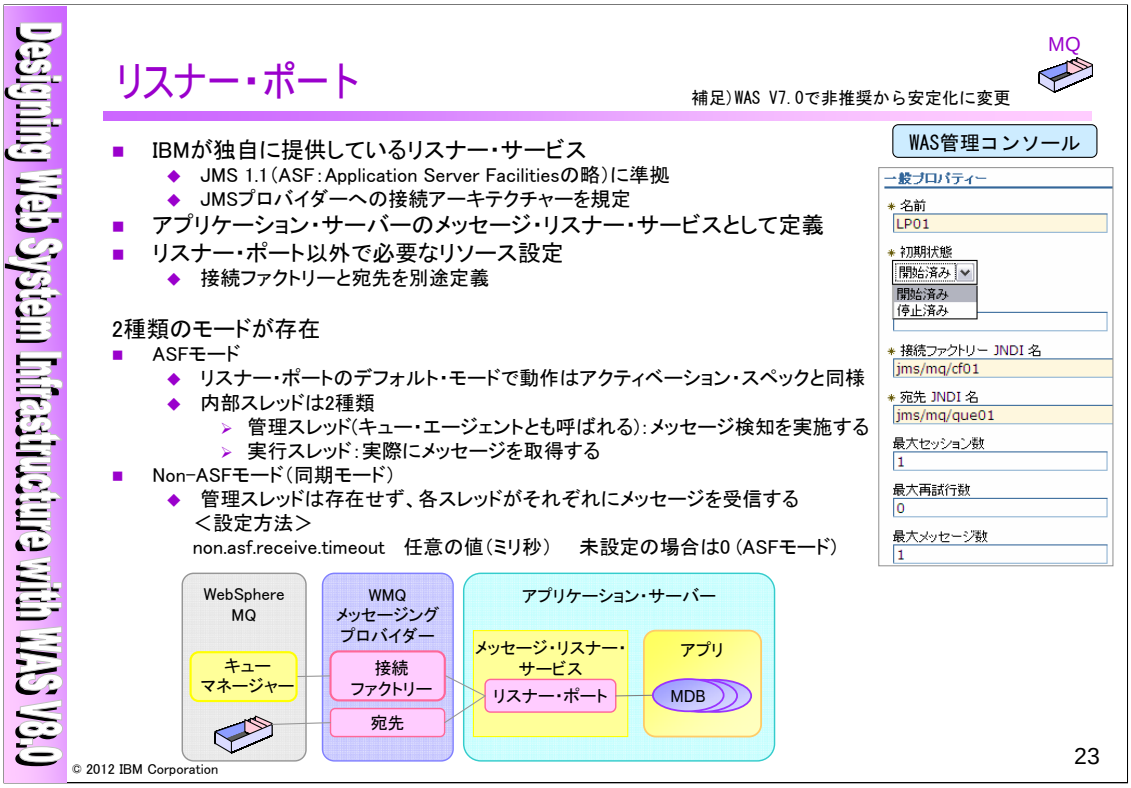
以前はアクティベーション・スペックにアプリケーション・サーバー起動時の初期状態を指定することができませんでしたが、WAS V7.0 FP13以降およびWAS V8.0では初期状態を指定できるようになりました。アクティベーション・スペックのカスタム・プロパティーに以下の設定を行います。

管理コンソールにて[リソース]-[JMS]-[アクティベーション・スペック]-[カスタム・プロパティー]を選択してください。

”WAS_EndpointInitialState”：ACTIVEあるいはINACTIVE

アクティベーション・スペックは内部動作として2種類のスレッドが使用されており、それらが連携してメッセージ配信を実施しています。

これらの2つのスレッドはキュー・マネージャーからは別個のアプリケーションとして見えます。管理スレッドは到着したメッセージを検知しますが、それを受信(GET)するわけでありません。そのかわりにメッセージIDを実行スレッドに転送して、メッセージを受信するためにセッションをスレッド上で実行します。そして受信したメッセージオブジェクトをMDBのonMessage()メソッドに配信します。



次にリスナー・ポートについて説明します。

先ほども申し上げた通り、MDBのリスナー機能はアクティベーション・スペックを利用することが今後推奨されますが、リスナー・ポートはMQ-MDBの構成で順序保証を実施するために必要となりますので、ここで詳しく説明しています。

リスナー・ポートはIBMが独自に提供しているリスナー・サービスで、JMS 1.1（ASF：Application Server Facilitiesの略）に準拠しています。

アプリケーション・サーバーのメッセージ・リスナー・サービスとしてリスナー・ポートを定義します。アクティベーション・スペックのように有効範囲の指定はできず、アプリケーション・サーバーに紐づく定義体です。クラスター構成の場合は、メンバーサーバー毎にリスナー・ポートの定義が必要になります。

また、MDBをWAS上で稼働させるにはリスナー・ポートのほかに接続ファクトリーと宛先を定義します。

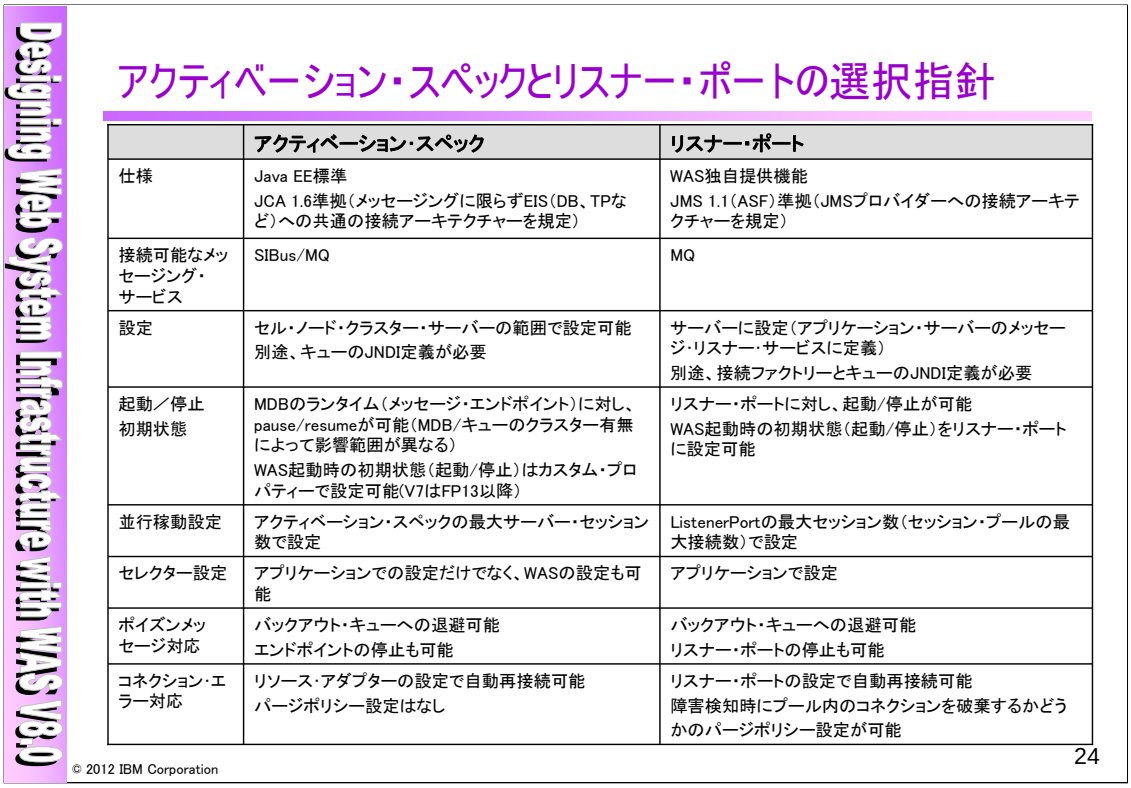
リスナー・ポートはMQと接続する際にのみ利用可能で、SIBusとの接続には利用することができませんので注意してください。

リスナー・ポートにはASFモード（デフォルト）とNon-ASFモードが存在します。

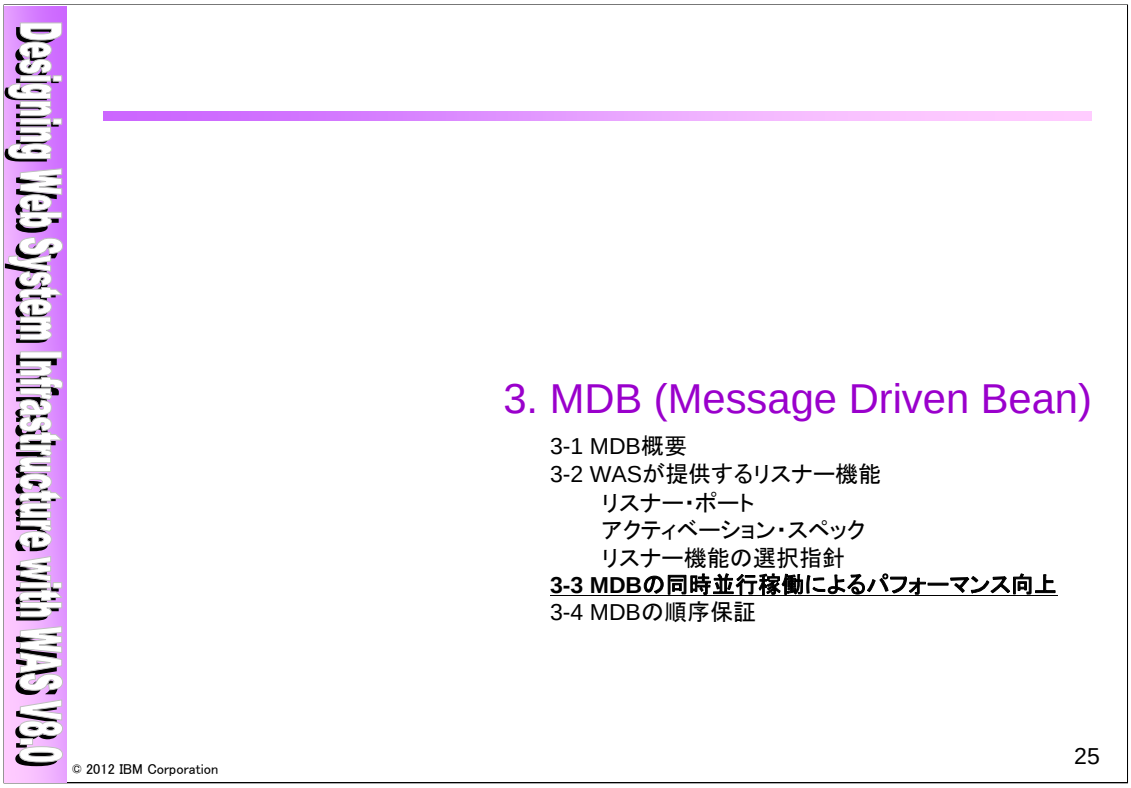
ASFモードはアクティベーション・スペックと同じ内部動作をとります。管理スレッドと実行スレッドが連携してMDBのonMessage()メソッドにメッセージを配信します。それに対して、Non-ASFモードは管理スレッドが存在せず、各実行スレッドがそれぞれにメッセージを受信します。リスナー・ポート起動時に指定された数のスレッドが起動され、それぞれが受信待機状態となります。受信待機状態のスレッドはメッセージの有無に関わらずアクティブのままでスレッド・プールにスレッドが戻されることはありません。

Non-ASFモードへの設定方法はアプリケーション・サーバーのメッセージ・リスナー・サービスのカスタム・プロパティーに以下の設定を行ってください。

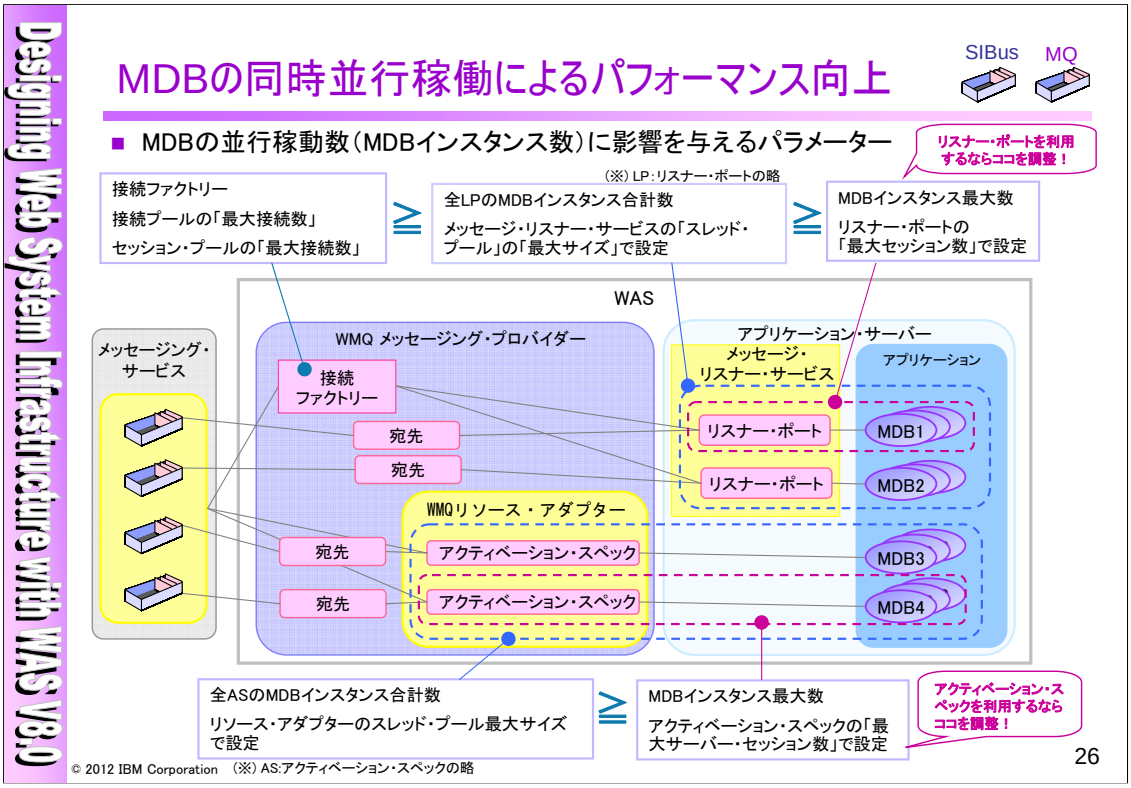
non.asf.receive.timeout　任意の値（ミリ秒）　未設定の場合は0(ASFモード)



アクティベーション・スペックとリスナー・ポートの選択指針を表にまとめています。



MDBの同時並行稼働数ついて解説します。



MDBを同時に並行稼働させることでパフォーマンスを向上させることができます。

MDBを並行稼働数を調整する場合の方針

•リスナー・ポートの最大セッション数あるいはアクティベーション・スペックの最大サーバー・セッション数で調整

•順序保証が必要な場合は1に設定する

•1つのアプリケーション・サーバーに複数のリスナー・ポートが定義され、それぞれにMDBが紐づいているようなケースはメッセージ・リスナー・サービスの「スレッド・プール」の「最大サイズ」で調整する

•接続ファクトリーの接続プールの最大接続数では制限をかけない

リスナー・サービスのパラメーターの考え方については、下記ガイドが参考になります。

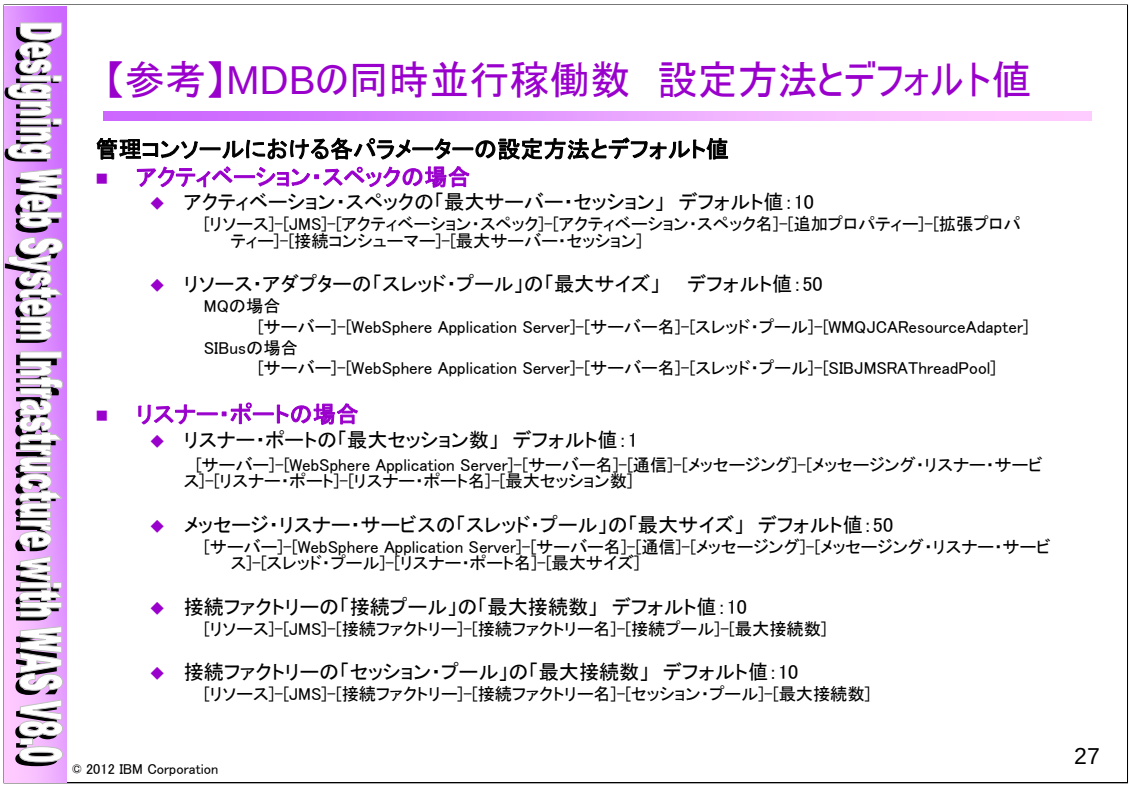
How the maximum sessions property on the listener port affects WebSphere Application Server performance

http://www.ibm.com/developerworks/websphere/library/techarticles/0602_kesavan/0602_kesavan.html

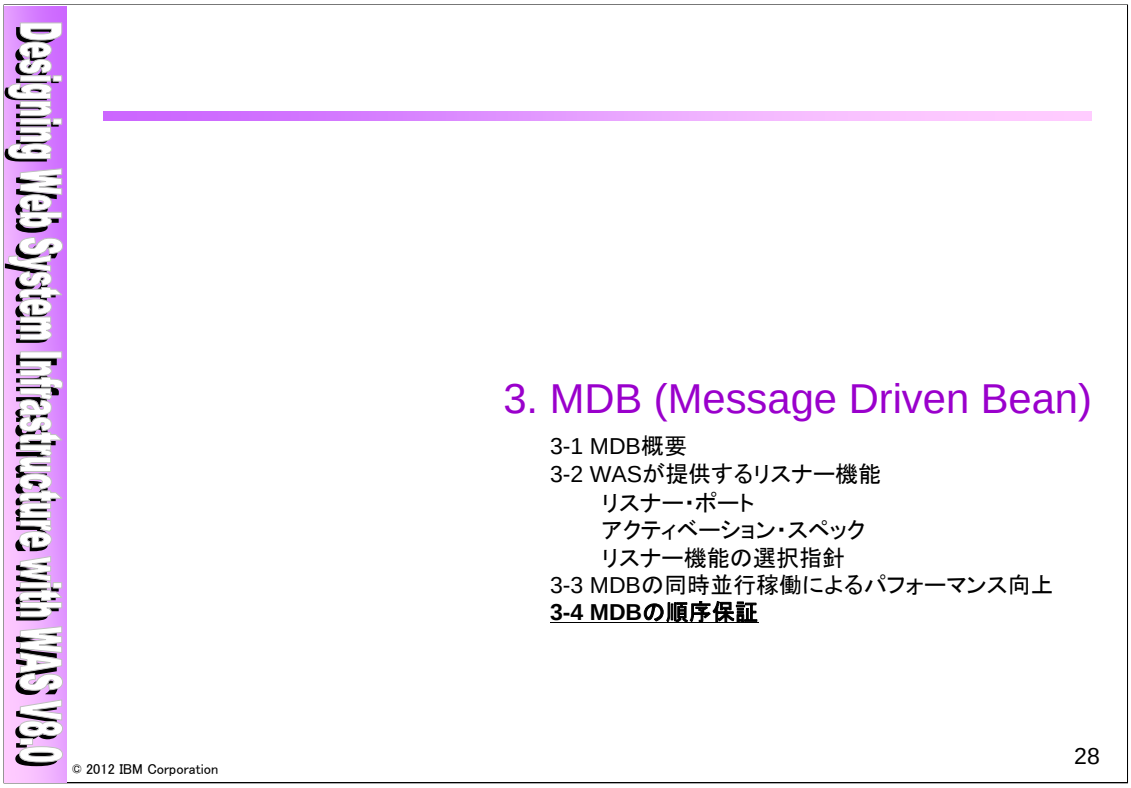
また、スレッド数設定指針としては、下記ガイドをご参照ください。

Get the most out of high performance message-driven beans and WebSphere Application Server

http://www.ibm.com/developerworks/websphere/library/techarticles/0508_parkinson/0508_parkinson.html



MDBの並行稼動数に影響を与える各パラメーターの管理コンソールでの設定方法とデフォルト値をまとめました。



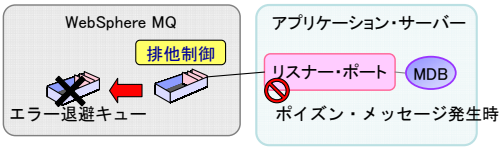
MDBの順序保証について説明します。

Designing Web System Infrastructure with WAS V8.0

MDBでの厳密なメッセージの順序保証

MQ

順序保証を行う場合はNon-ASFリスナー・ポートを使用してください



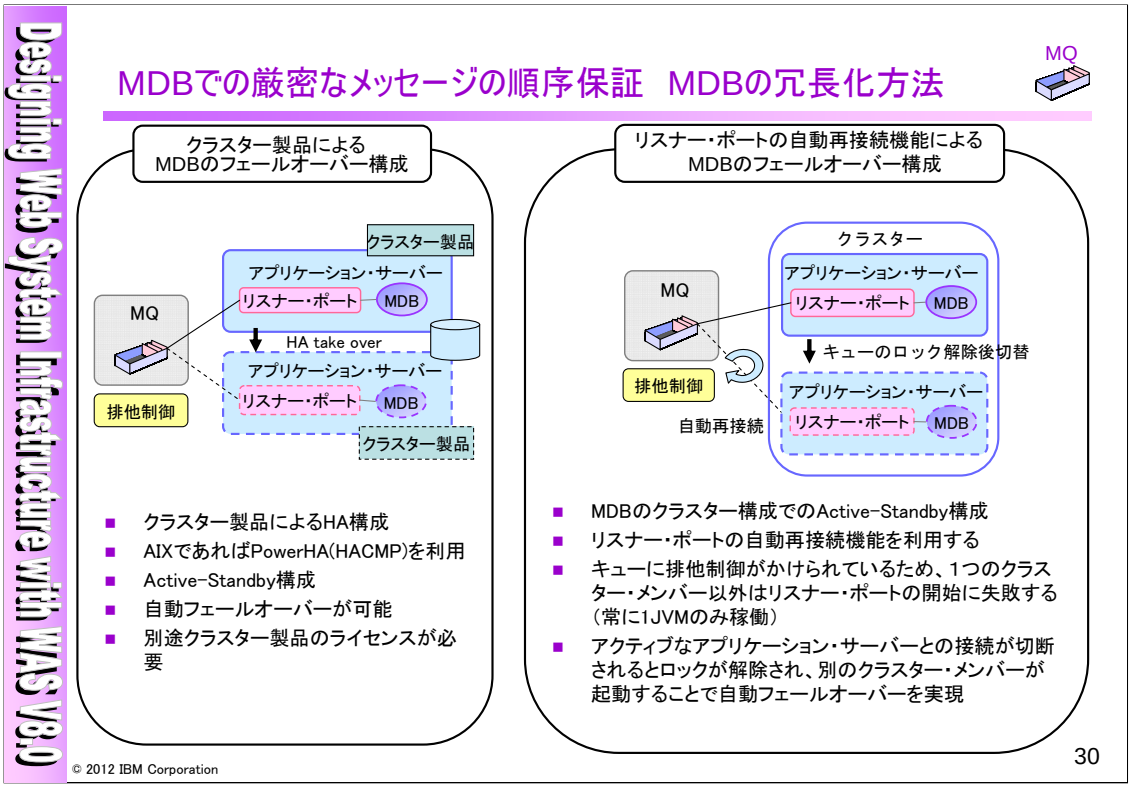
メッセージ処理の経路を１本にすることがポイント

- 必要な設定
  - MDBの稼働インスタンスを一時点で1つに制限する
    - リスナー・ポートの最大セッション数を1に設定
    - クラスター構成にしたとしても1JVMしか起動できない
  - キューを排他制御
    - MQのキュー属性「DEFSOPT」を「EXCL」に設定
    - キューに排他制御がかかるとASFリスナー・ポートおよびアクティベーション・スペックはエラーになる
  - バックアウトを繰り返すメッセージ（ポイズン・メッセージ）の対応
    - リスナー・ポートを停止させる
      - WAS側でリスナー・ポートの「最大再試行数」を0に設定
    - エラー退避キューは設けない
      - MQ側でバックアウトしきい値(BOTHRESH)を0に設定
- SPOFとなるため冗長構成を検討する必要がある（後述）
- 2フェーズ・コミット利用時は厳密な順序保証のため特別な考慮が必要 （後述）

© 2012 IBM Corporation

29

MDBにおける厳密なメッセージの順序保証を行うためには、Non-ASFモードのリスナー・ポートを使用する必要があります。



MDBにおいて順序保証をする場合は、通常のクラスター構成でActive-Active構成にできないため、アプリケーション・サーバーのトポロジーが大きく変わります。稼働できるアプリケーション・サーバーが1つであるため、障害時はアプリケーション・サーバーの再起動あるいは、別のアプリケーション・サーバーを起動することでMDBの処理を継続させることになります。

MDBの冗長化方法を検討する場合で自動フェールオーバー構成をとりたい場合、以下の2つの方法が考えられます。

・クラスター製品によるMDBのフェールオーバー構成

クラスター製品を利用してActive-Standby構成にすることでフェールオーバーを実現します。

・リスナー・ポートの自動再接続機能によるフェールオーバー構成

MDBをクラスター構成にして、Active-Standby構成にします。MQ側の排他制御とリスナー・ポートの自動再接続機能を利用することで自動フェールオーバーが可能です。

以下の設定を行います。

設定方法

メッセージ・リスナー・サービスのカスタム・プロパティー

・MAX.RECOVERY.RETRIES（再接続の回数）

できる限り大きな値を設定

設定可能な最大値：2147483647

・RECOVERY.RETRY.INTERVAL（再接続の間隔）

適切な値を設定

Designing Web System Infrastructure with WAS V8.0

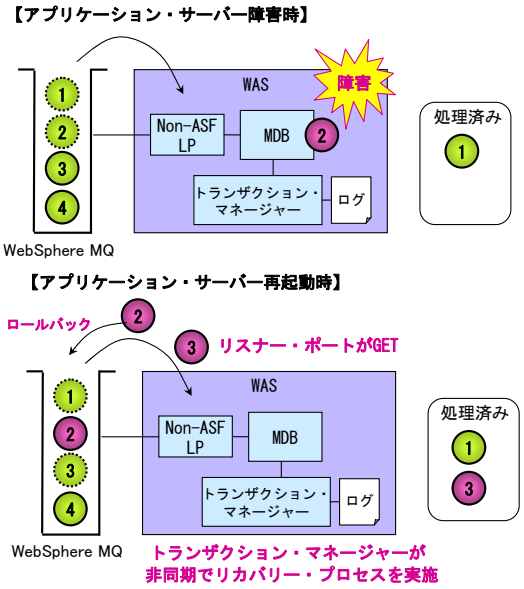
2フェーズ・コミット利用時の処理順序がずれるケース

MQ

- アプリケーション・サーバー障害時のトランザクション・リカバリーを行うシナリオ
- アプリケーション・サーバー再起動の際にトランザクション・リカバリーより先にMDBの処理が開始されるとメッセージの処理順序がずれる可能性がある（発生する確率は低い）

対応策

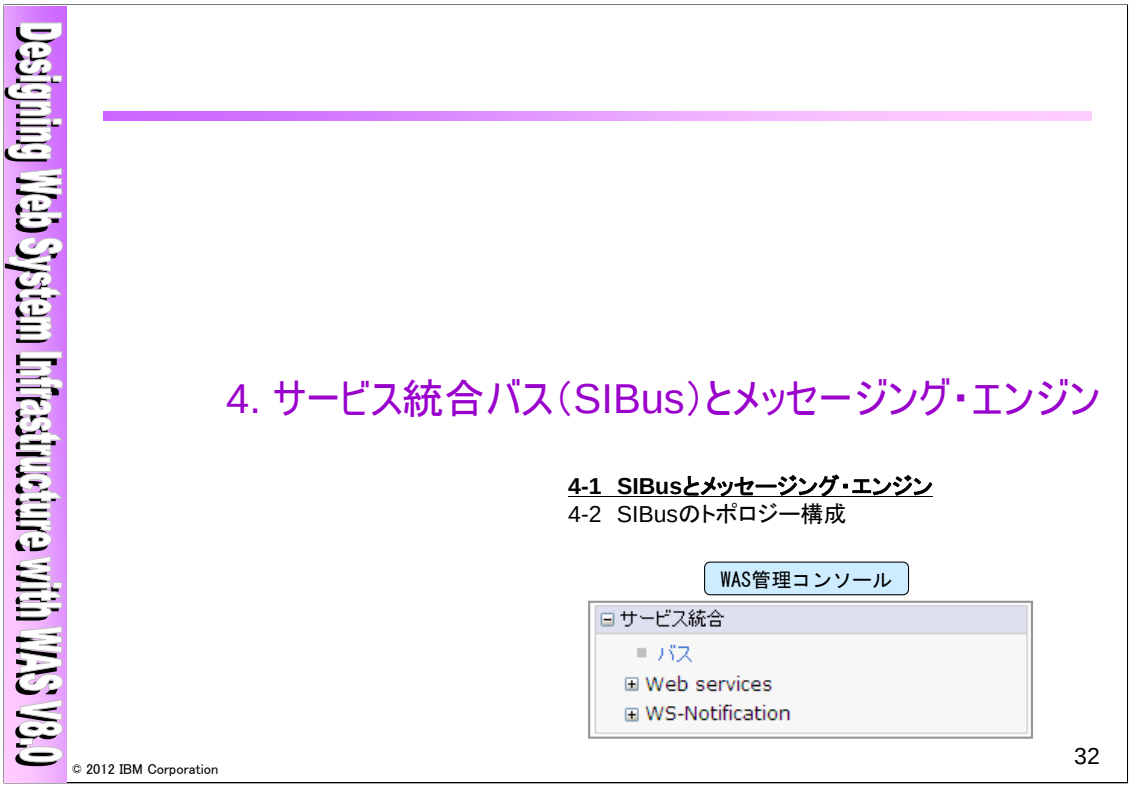
- 方法1
  リスナー・ポートの初期状態を停止状態に設定
- 方法2
  デフォルト・メッセージング・プロバイダーの”厳密なメッセージの順序付け(Strict message ordering)”を利用するとトランザクション・リカバリー・シナリオにおいても順序を保証することができる。SIBusのMQリンク機能（後述）を利用するため構成が複雑になる



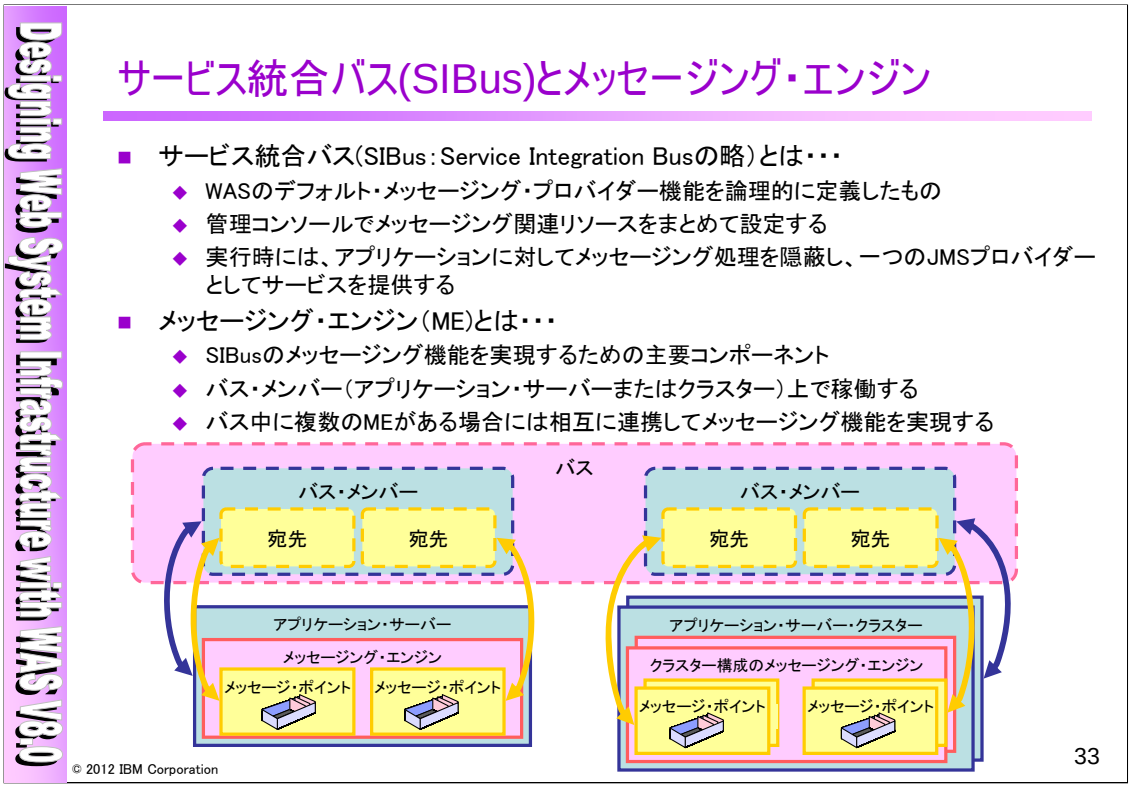
© 2012 IBM Corporation

31

2フェーズ・コミット利用時において処理順序がずれるケースがあり、特別な考慮が必要になります。



この節では、サービス統合バス（SIBus ）とメッセージング・エンジンについて解説します。



SIBus (Service Integration Bus)は、メッセージング・プロバイダーを論理的に定義したもの(Logical Entity)であり、実体としてのメッセージング・プロバイダー機能はMEにより提供されています。

別の言い方をすると、SIBusは仮想的なメッセージング・プロバイダーで、機能の実装はMEにより提供されています。アプリケーションは、表面的には、SIBusのインスタンスであるバスに接続し、宛先を介してメッセージのやり取りを行います。ただし、内部的には、アプリケーションはMEに接続して、メッセージ・ポイントを介してメッセージのやり取りを行っています。

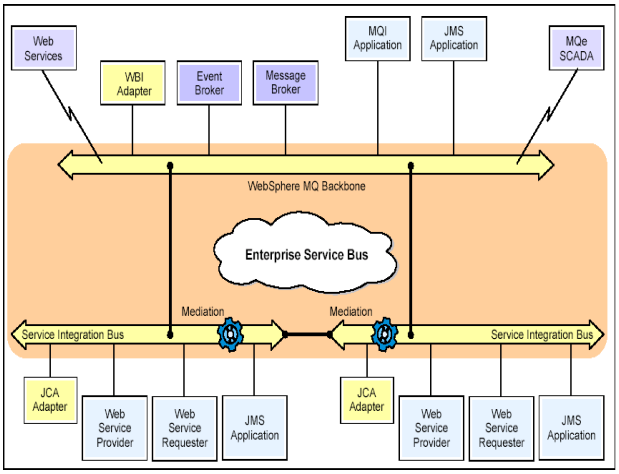
メッセージング・プロバイダーを利用するために、WAS管理者はバスを作成し、バス・メンバーを登録して、そのバス・メンバー上に宛先を作成します。これにより、バス・メンバーとして登録したアプリケーション・サーバーあるいはクラスターにはMEが生成され、宛先に関連付けられたバス・メンバーのMEにはメッセージ・ポイントが生成されます。アプリケーション・サーバーとMEは、同一プロセス内で動作します。

WAS管理者は抽象化されたバスや宛先だけでなく、実体であるMEやメッセージ・ポイントを意識しなくてはなりませんが、アプリケーションを開発する上では、バスに接続し宛先にメッセージを送信する、という考え方に基づいてコーディングできます。

Designing Web System Infrastructure with WAS V8.0

SIBus機能一覧

- メッセージング機能
- JMSプロバイダー機能
- 高可用性機能
- 負荷分散機能
- 外部バスとの接続機能
  - SIBusリンク
  - MQリンク
- メディエーション機能
  - メッセージ・フォーマット変換
  - ルーティング
- SIBWS機能
  - セキュリティー機能



© 2012 IBM Corporation

34

SIBusは以下のような機能を提供します。

・メッセージング機能

SIBusでは、メッセージング・サービスを実装することで、疎結合を実現しています。

・JMSプロバイダー機能

SIBusをJMSプロバイダーとして利用する機能できます。

・高可用性

HAマネージャーと連携して、SIBusのメッセージング・サービスの実体として扱われるメッセージング・エンジン(以下ME)の可用性を高めることができます。

・負荷分散機能

メッセージを送信する際に、複数のキューに振り分けることで、負荷を分散することができます。

・外部バスとの接続機能

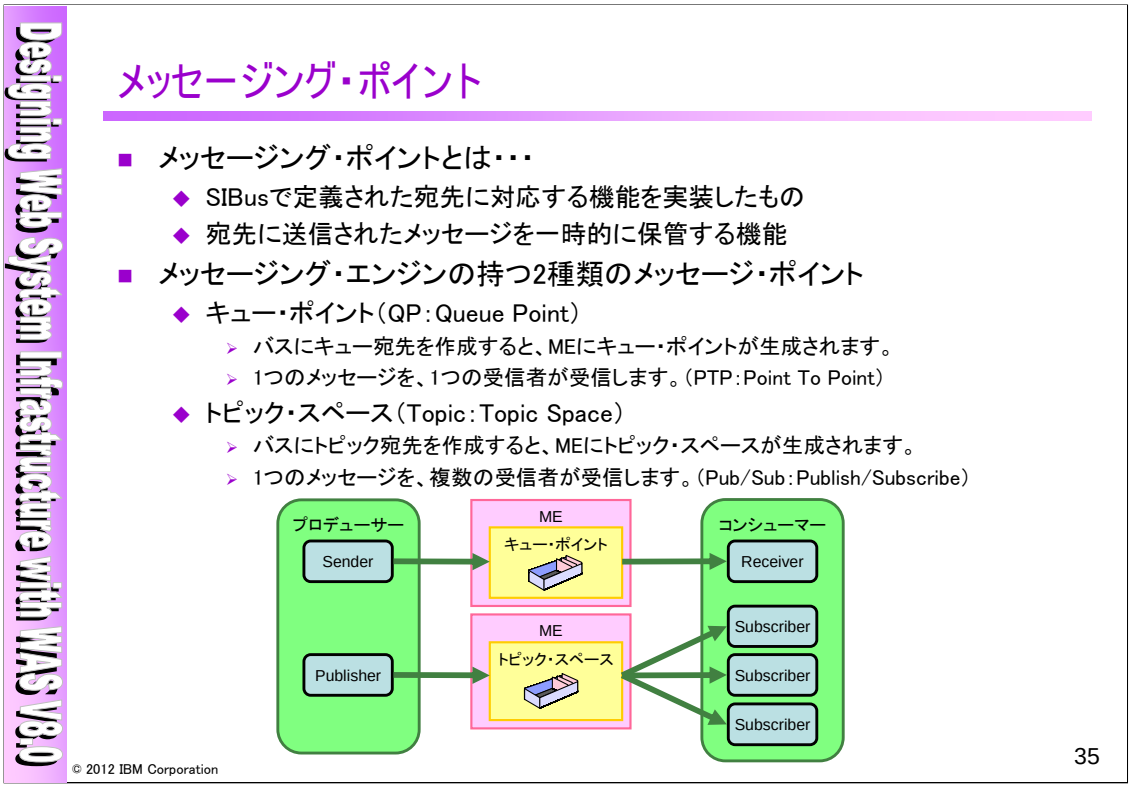
前のリリースの組み込みメッセージングでは、MQとチャネル接続できない、といった制約がありました。SIBusではMQとのチャネル接続が可能です。

・メディエーション機能(Mediation)

メッセージのフォーマット変換や、ルーティングを行うことができます。メディエーションを利用すると、サービスの変更を最小限にとどめたり、サービスの繋ぎ替えが可能になります。

・SIBWS機能

WASに標準で付属のSIBus Web Service enablementを導入することで、Webサービスも SIBusに統合できます。この機能をSIBWSと呼びます。

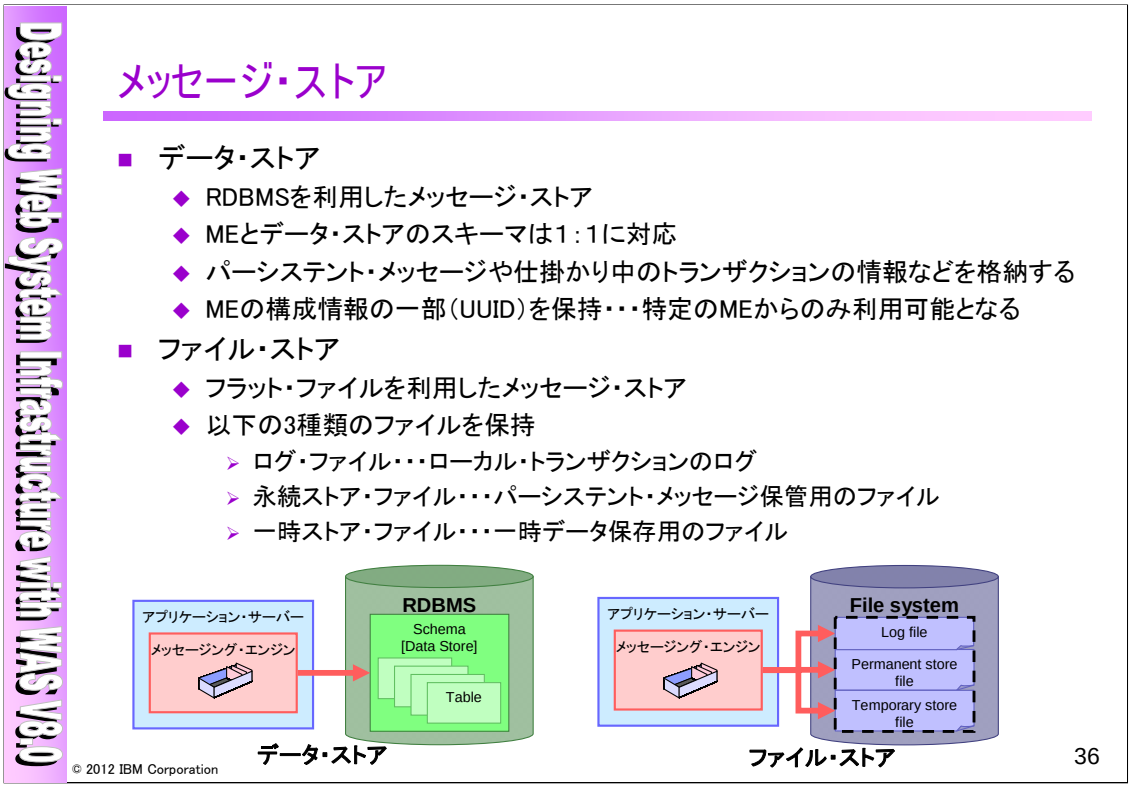


バスの宛先には、PTPでやり取りを行う為のキュー宛先と、Pub/Subを行う為のトピック宛先が利用できます。

キュー宛先を作成すると、宛先に関連付けられたバス・メンバーのMEにキュー・ポイントが生成されます。
PTPの場合、メッセージを送る側を送信者（Sender）、メッセージを受け取る側を受信者（Receiver）と呼びます。

トピック宛先を作成すると、宛先に関連付けられたバス・メンバーのMEにトピック・スペースが生成されます。
Pub/Subの場合、メッセージを送る側を発行者（Publisher）、メッセージを受け取る側を購読者（Subscriber）と呼びます。

キュー・ポイントやトピック・スペースなど、宛先の実体にあたるオブジェクトを、MEではメッセージ・ポイントと呼びます。



MEを利用するには、メッセージ・ストアが必要です。メッセージ・ストアには、RDBMSを利用したデータ・ストアと、フラット・ファイルを利用したファイル・ストアの2種類があります。

データ・ストアでは、1つのMEに対して、1つのスキーマを割り当てます。データ・ストアには、パーシステント・メッセージの他に、仕掛かり中のトランザクションの情報などを格納します。また、MEの構成情報の一部（UUID）を保持しているため、特定のMEからのみ利用可能となっています。

ファイル・ストアでは、以下の3種類のファイルを保持しています。

◆Log file（ログ・ファイル）

MEを構成すると、初回起動時に設定されたログ・サイズ分、ディスク領域を確保します。先頭から末尾に向かって、ローカル・トランザクションのログを書き出します。使い切ってしまうと、古いログを先頭から上書きしていきます。（TMログとは異なります。）ME上での保持期間が短いメッセージは、メッセージの信頼性に依っては、ログのみに書き出されて、永続ストアや一時ストアには書き出されません。

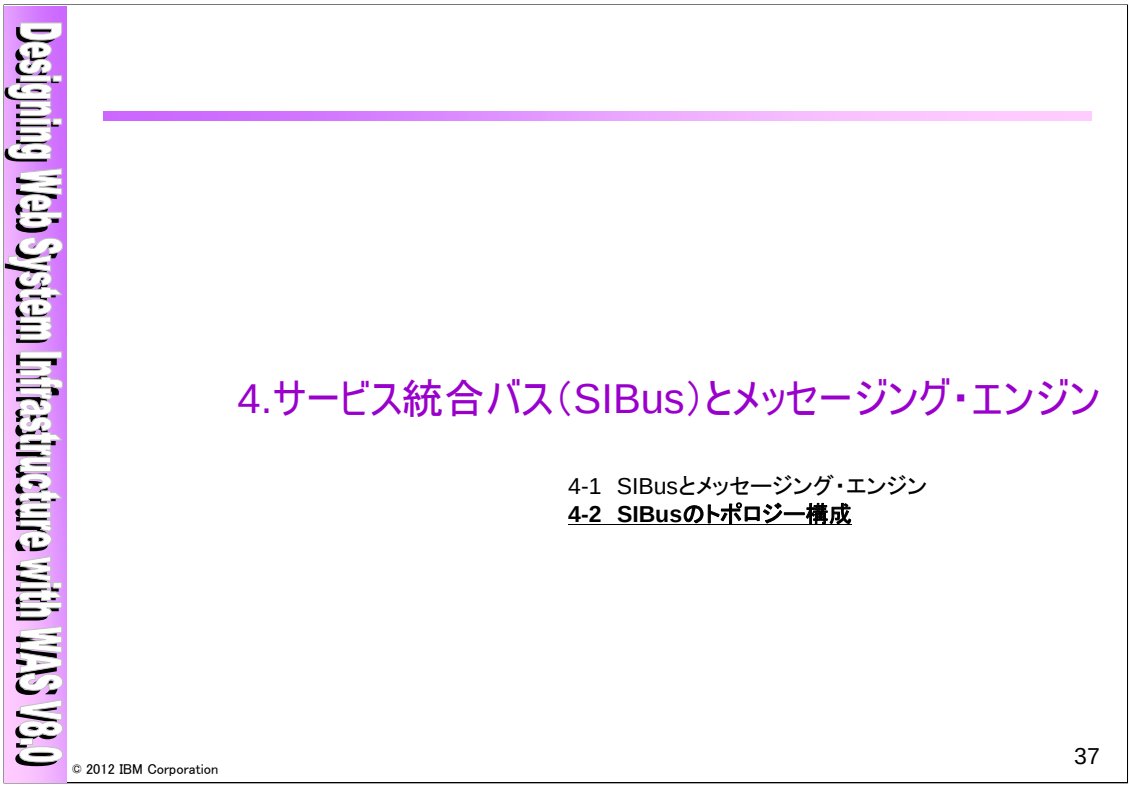
◆Permanent store file（永続ストア・ファイル）

MEを構成すると、初回起動時に設定された最小永続ストア・サイズ分、ディスク領域を確保します。ファイルに収まらなくなると、最大永続ストア・サイズに指定したサイズまで、必要な分ずつ拡張していきます。

◆Temporary store file（一時ストア・ファイル）

MEを構成すると、初回起動時に設定された最小一時ストア・サイズ分、ディスク領域を確保します。ファイルに収まらなくなると、最大一時ストア・サイズに指定したサイズまで、必要な分ずつ拡張していきます。

データ・ストアとファイル・ストアの比較と注意点については巻末の参考資料をご参照ください。

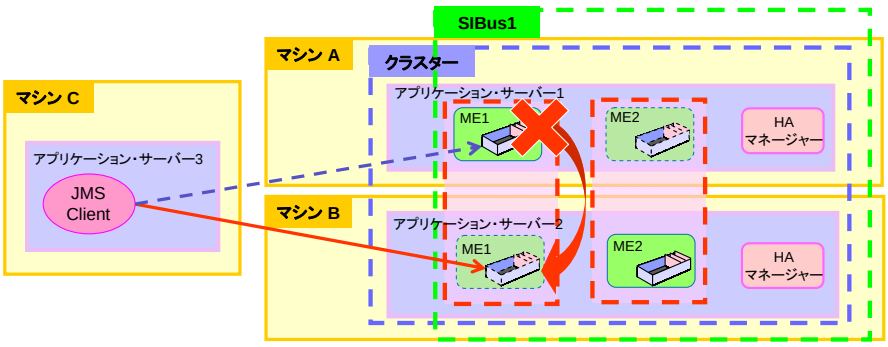


次にSIBusの代表的なトポロジー構成について紹介します。

Designing Web System Infrastructure with WAS V8.0

SIBusのトポロジー構成

- HAマネージャーによるMEのフェールオーバー構成(Active-Standby)
  - クラスター構成のメッセージング・エンジンのフェールオーバーが可能
  - メッセージ・ストアを共有ディスク上に配置することでパーシステント・メッセージの引継ぎが可能になる
- MEのパーティショニング（クラスタリング）機能による負荷分散構成(Active-Active)
  - バス・メンバー内に複数のメッセージング・エンジンを起動し、処理の負荷分散が可能



© 2012 IBM Corporation

38

SIBusのトポロジー構成として、高可用性、スケーラビリティー、パフォーマンスを向上させるために以下のような構成をとることが可能です。

・HAマネージャーによるフェールオーバー構成

各JVM上で稼働しているHAマネージャーによりメッセージングエンジンの引継ぎが可能になります。メッセージエンジンをクラスター構成にすることでクラスター間でのフェールオーバーが可能になります。（Active-Standby構成）

・MEのパーティショニング機能による負荷分散構成

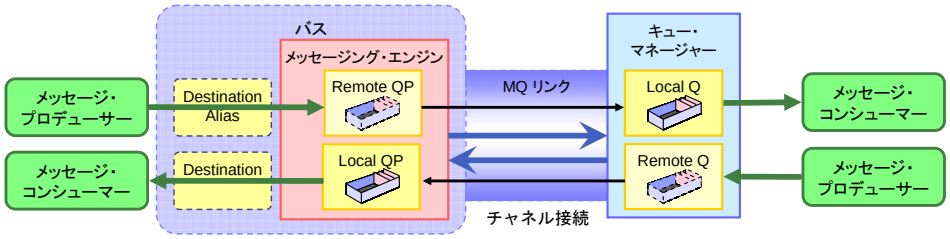
メッセージングエンジンはパーティショニング（クラスタリング）機能を利用すればメッセージ処理の負荷分散を行うことが可能です。

同時に複数のメッセージング・エンジンを稼働させることが可能です。（Active-Active構成）

Designing Web System Infrastructure with WAS V8.0

SIBusの外部接続　(SIBusリンクとMQリンク)

- 2種類の外部バス接続が存在する
  - SIBusリンクはSIBusとSIBusを接続する
  - MQリンクはSIBusとMQのキュー・マネージャーを接続する
- MQのキュー・マネージャーとの連携
  - キュー・マネージャーを外部バスとして登録できる
  - メッセージング・エンジンとキュー・マネージャーをチャネル接続（MQの機能）することで、バス-MQ間におけるメッセージのやり取りを実現している
  - このチャネル接続のことを、MQリンクと呼ぶ
  - メッセージング・エンジンは、キュー・マネージャーにとって、リモート・キュー・マネージャーであるように振舞う



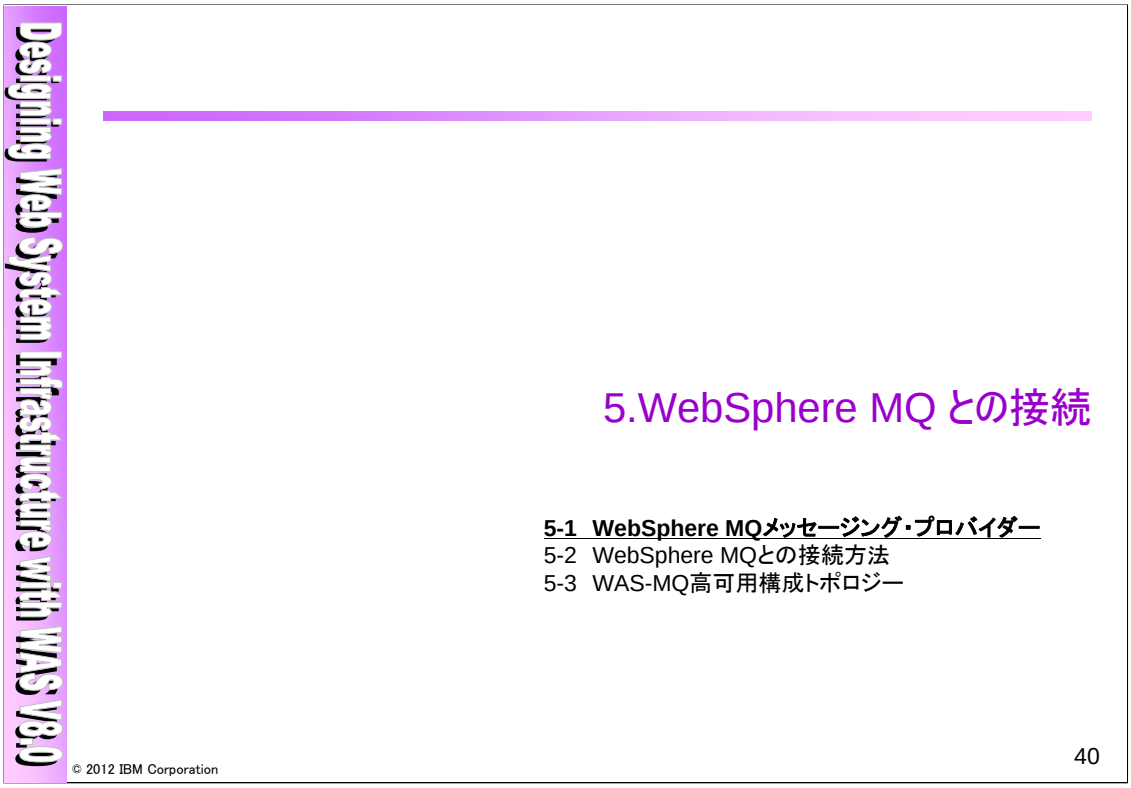
© 2012 IBM Corporation

39

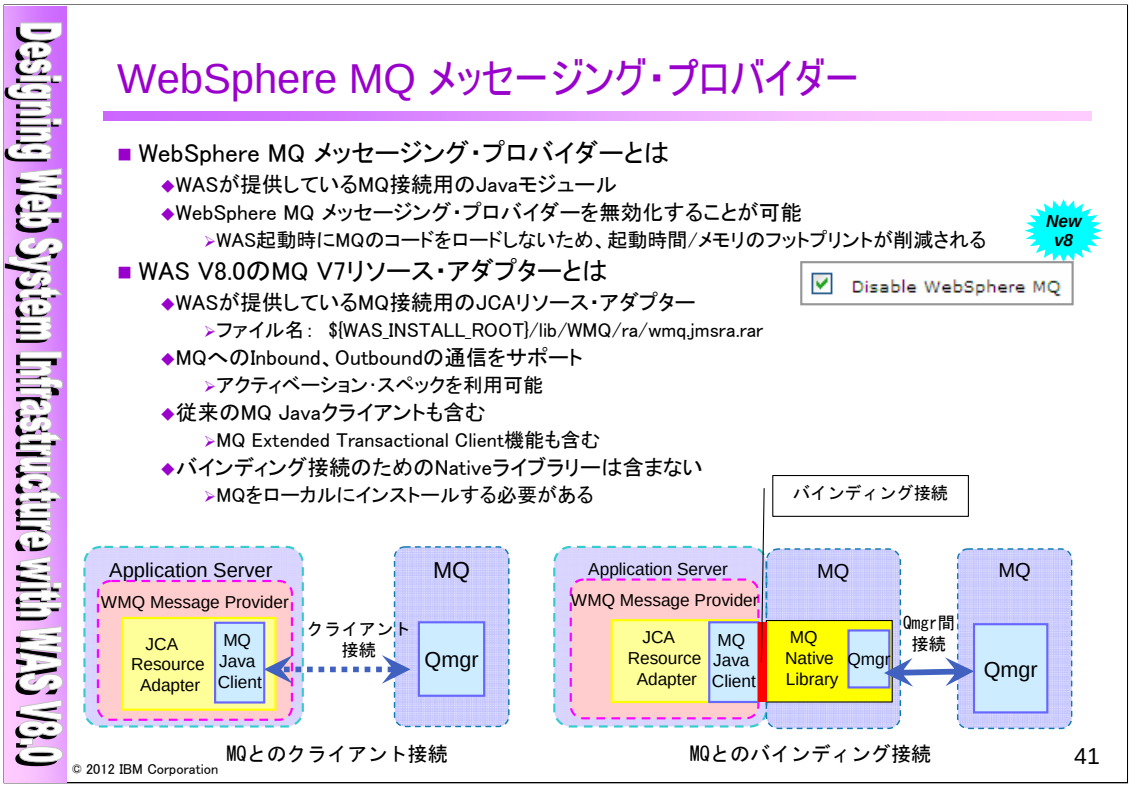
MQリンクは、MEがキュー・マネージャーとして振舞うことで、MQのキュー・マネージャーとチャネル接続する機能です。SIBusの世界からはキュー・マネージャーが外部バスに見えるのに対して、MQの世界からはMEがリモートキュー・マネージャーに見えます。バスからキュー・マネージャーにメッセージを送信するには、キュー・マネージャーのローカル・キューに対して、バスに別名宛先を定義します。キュー・マネージャーからバスにメッセージを送信するには、バスの宛先に対して、キュー・マネージャーにリモート・キューを定義します。

MQリンクを利用する環境を構築するのには、WASに加えてMQの知識が重要です。MQリンクを利用する場合は、TCTなどによりシステムの妥当性を検討してください。

外部バス接続の作成方法については、参考資料に「外部バス接続の作成方法」がありますので、参照してください。



この節では、WebSphere MQとの接続ということでJMSプロバイダー、接続方法、WAS-MQの高可用構成トポロジーついて解説します。



WebSphere MQ messaging providerとはWASが提供しているMQ接続用のJavaモジュールです。その主要なコンポーネントであるMQ V7リソース・アダプターは、MQ接続用のJCAリソース・アダプターです。具体的なファイル名としては、

${WAS_INSTALL_ROOT}/lib/WMQ/ra/wmq.jmsra.rar

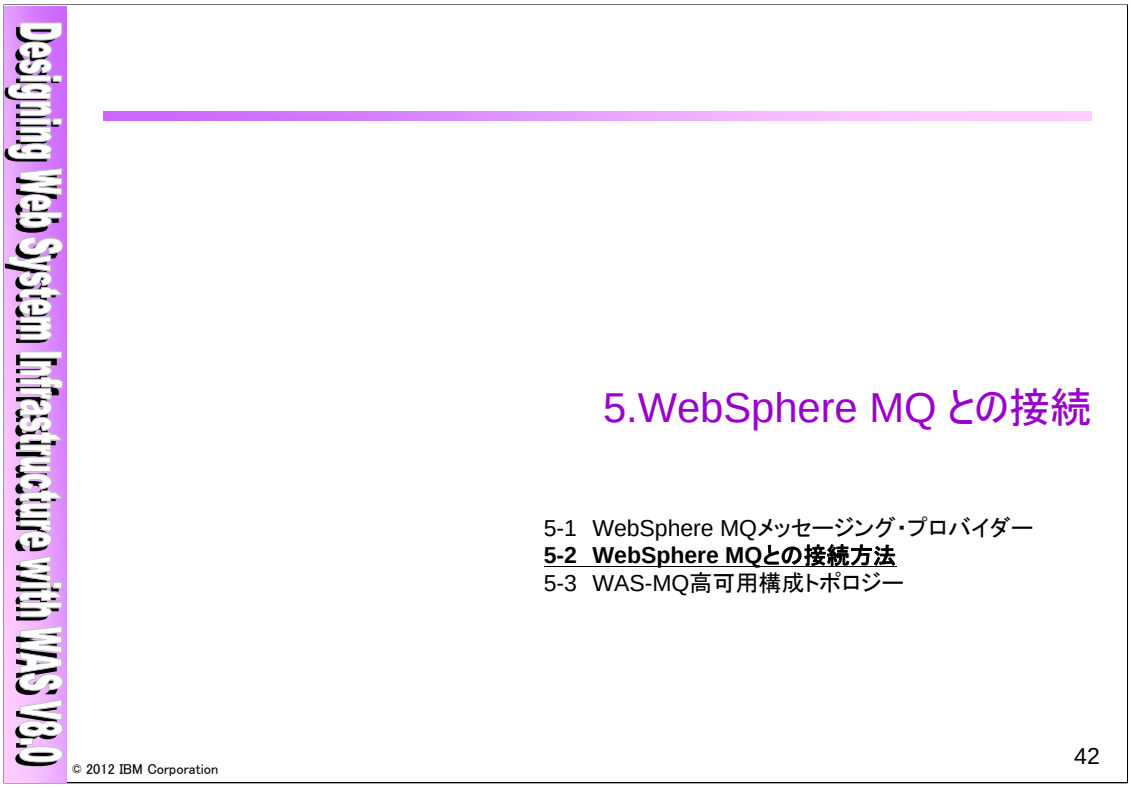
となります。

これにより、MQへのInbound、Outboundの通信をサポートすることができ、アクティベーション・スペックを利用することができるようになりました。

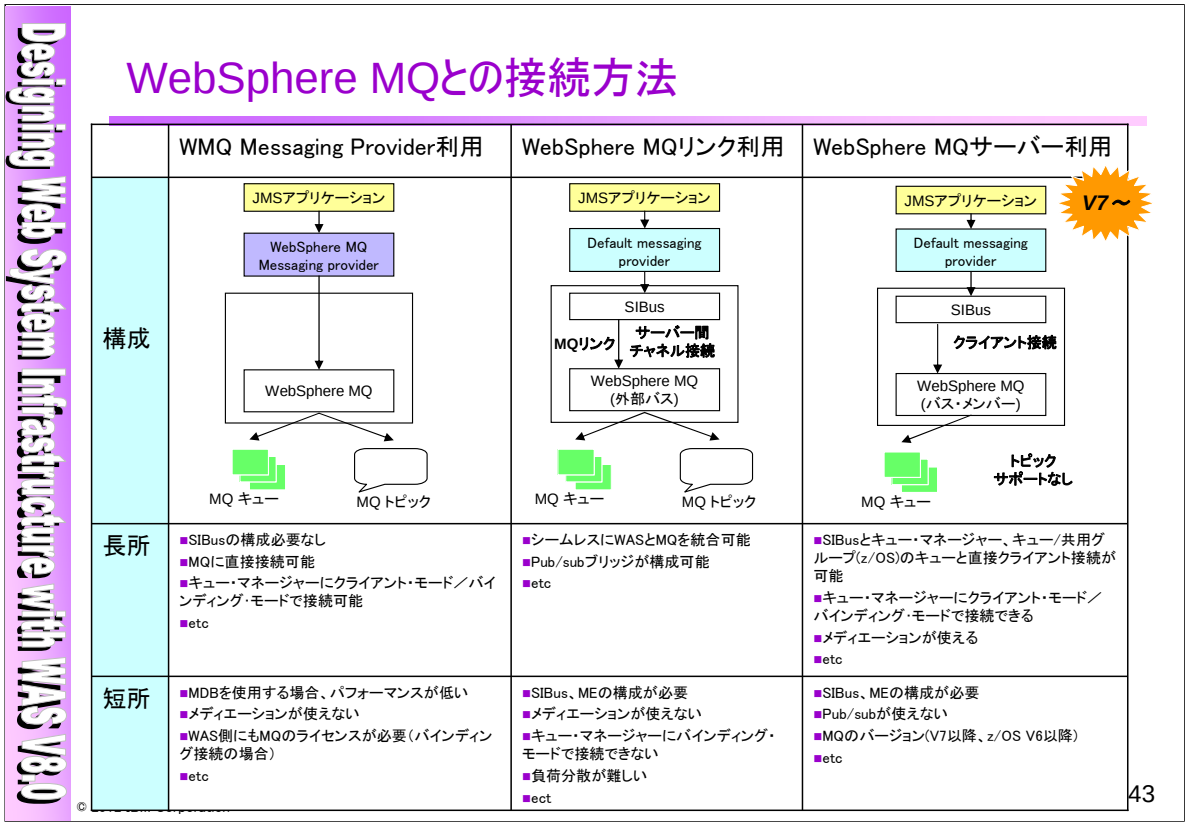
また、MQ V7リソース・アダプターは従来のMQ Javaクライアントも含むので、それが含んでいたMQ Extended Transactional Client機能も利用可能です。ただし、Nativeライブラリは含んでいませんので、バインディング接続を行うためには、MQをローカルにインストールする必要があります。さらに、バインディング接続の設定を行うためには、リソース・アダプターの「Native library path」にMQ提供のNativeライブラリのディレクトリを指定するか、WebSphere変数「MQ_INSTALL_ROOT」にMQ導入ディレクトリを指定する必要があります。

※WAS V8.0では「MQ_INSTALL_ROOT」はNativeライブラリを指定するために使用されます。

※「Native library path」設定のほうが優先されます。



次にWebSphere MQとの接続方法を紹介します。



WASとMQとの接続方法には3通りあります。

MQ Messaging Providerを利用する方法は前述した通りですが、これ以外にMQ と接続する方法は、MQリンクを利用する方法とMQサーバーを利用する方法があります。

WAS V8 InfoCenter: WebSphere MQ との相互協調処理: 主なフィーチャーの比較

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/cmm_comp_feat.html

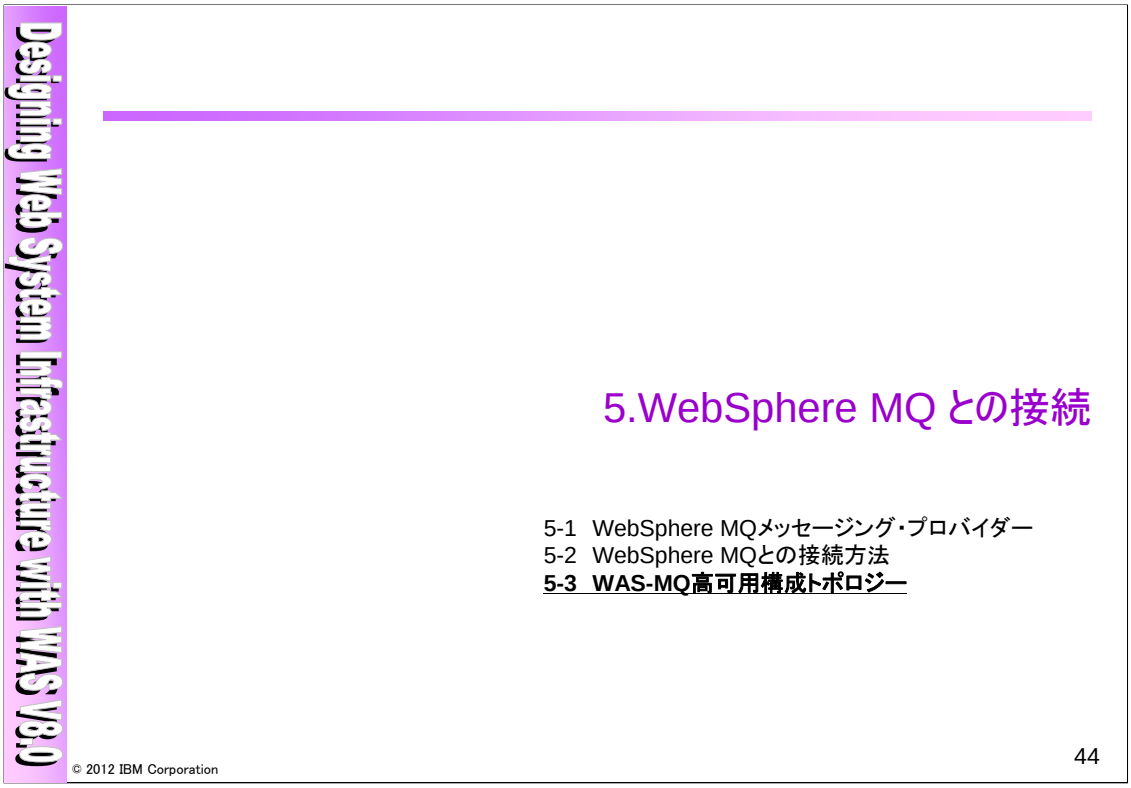
WAS V8 InfoCenter: WebSphere MQ との相互運用: アーキテクチャーの比較

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/cmm_comp_arch.html

WAS V8 InfoCenter: WebSphere MQ との相互協調処理

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/cmm_interop.html

WebSphere MQサーバー利用の場合の構成方法については、参考資料に「MQサーバーをSIBusメンバーとして追加する方法」がありますので、参照してください。

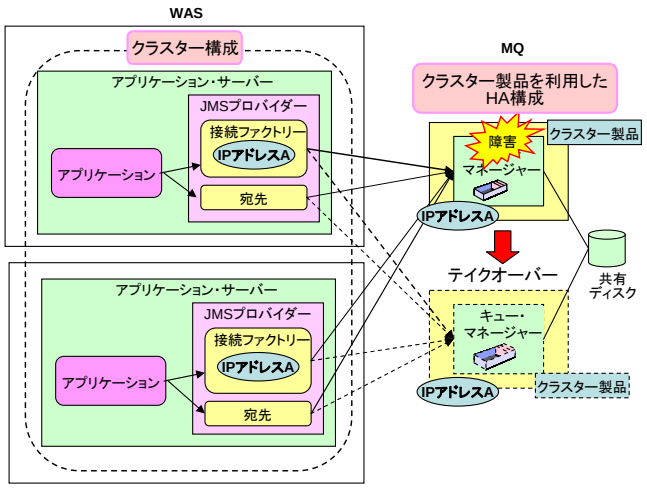


WAS-MQ高可用構成トポロジーについて解説します。

Designing Web System Infrastructure with WAS V8.0

WAS-MQ高可用構成トポロジー (1)

- WASはクラスター構成
- MQは高可用性製品を利用したフェールオーバー構成
  - PowerHA(HACMP)などのクラスター製品を使用したActive-Standby構成
- WASが接続すべきキュー・マネージャーはひとつだけ



© 2012 IBM Corporation

45

WASとMQが連携するシステムにおける高可用構成トポロジーを2つご紹介します。

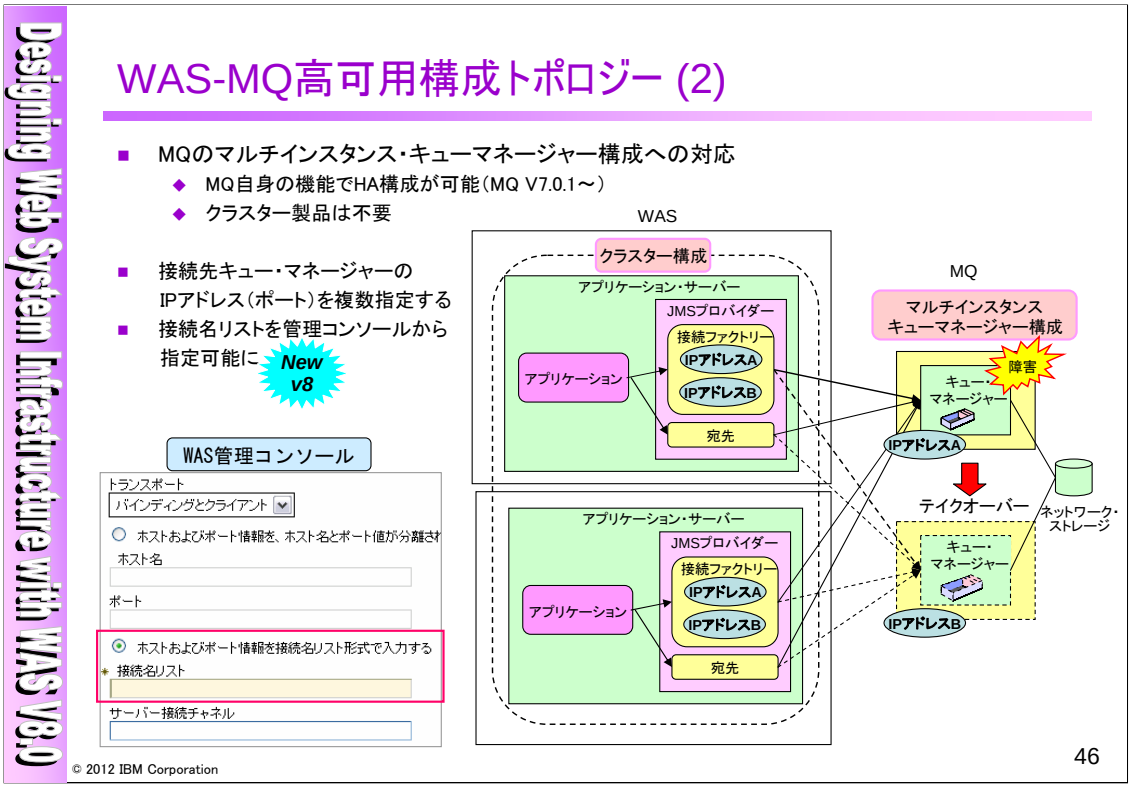
(1) WASはクラスター構成/MQはクラスター製品を利用したHAフェールオーバー構成

WAS側のアプリケーション・サーバーはクラスター構成をとるのが一般的です。

MQ側の冗長構成方法としてクラスター製品(PowerHAやMSCSなど)を利用したHA構成(Active-Standby構成)をとるのが一般的です。

MQ障害時、クラスター製品の機能で自動フェールオーバーされ、ネットワークアドレスの引き継ぎも行われます。

WASからみると、単一のIPアドレスで接続しますので、接続ファクトリーではひとつのIPアドレスを指定します。



(2) WASはクラスター構成、MQはマルチインスタンス・キュー・マネージャー構成

WAS側のアプリケーション・サーバーは前ページと同様にクラスター構成をとります。

MQ側のキュー・マネージャーはMQ V7.0.1より提供されているマルチインスタンス・キュー・マネージャー構成をとります。

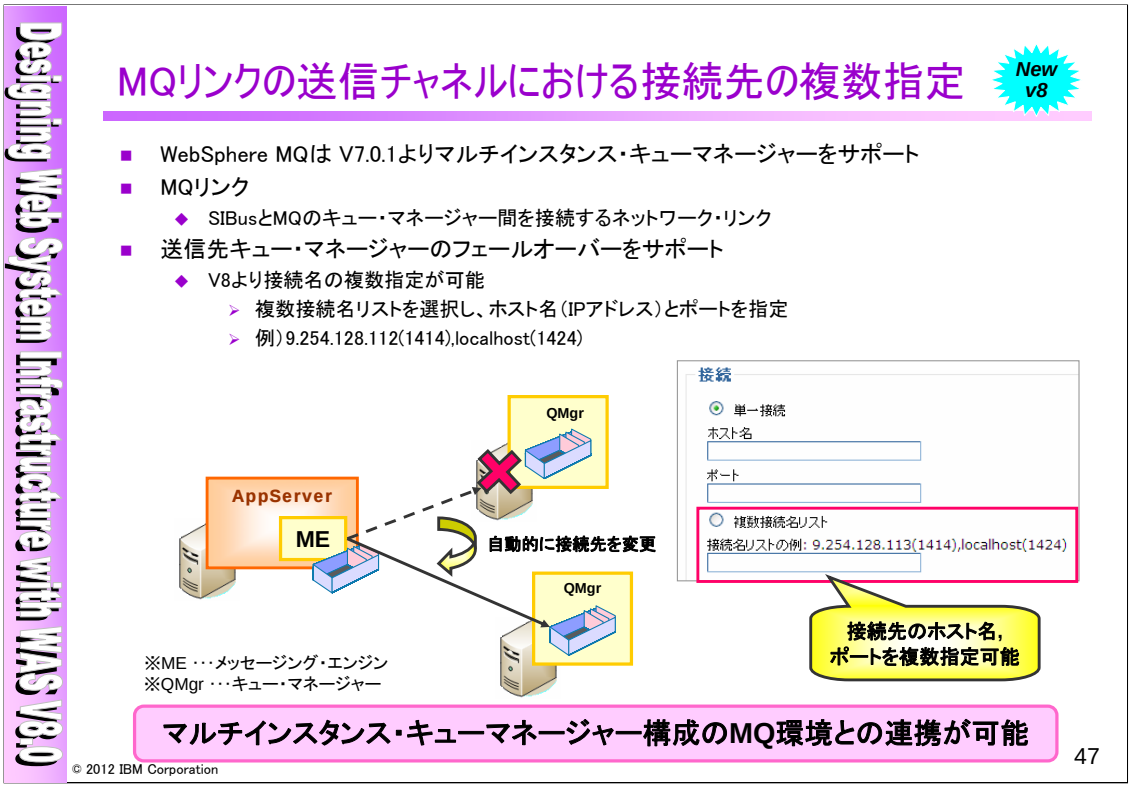
マルチインスタンス・キュー・マネージャーはMQ自身の機能で自動フェールオーバーが可能になりますので、クラスター製品は不要です。

ただし、ネットワークアドレスは引き継がれず、ActiveとStandbyでIPアドレスが異なりますので、WAS側でも複数IPアドレスへの接続ができるよう構成する必要があります。

接続先キュー・マネージャーを切り替える方法としては、接続名リスト（IPアドレスとポート番号をカンマ(,)で区切った接続リスト）で指定する方法とチャネル定義テーブルを利用する方法があります。チャネル定義テーブルはクライアント接続のみ利用可能です。

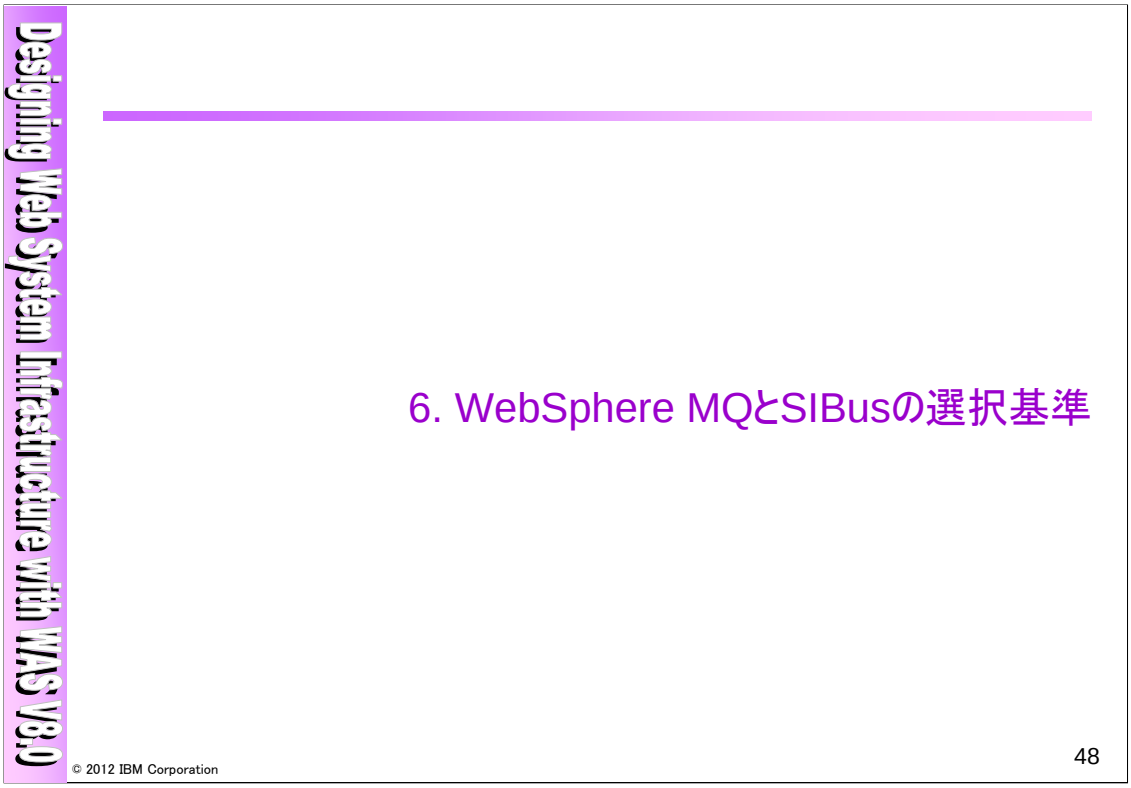
WAS V8.0より管理コンソールから接続名リストを指定可能になっています。（以前のバージョンではカスタムプロパティーに設定する必要がありました。）

また、V8.0の新機能としてMQリンクの送信チャネルにおける接続先が指定可能になっています。（次頁参照）

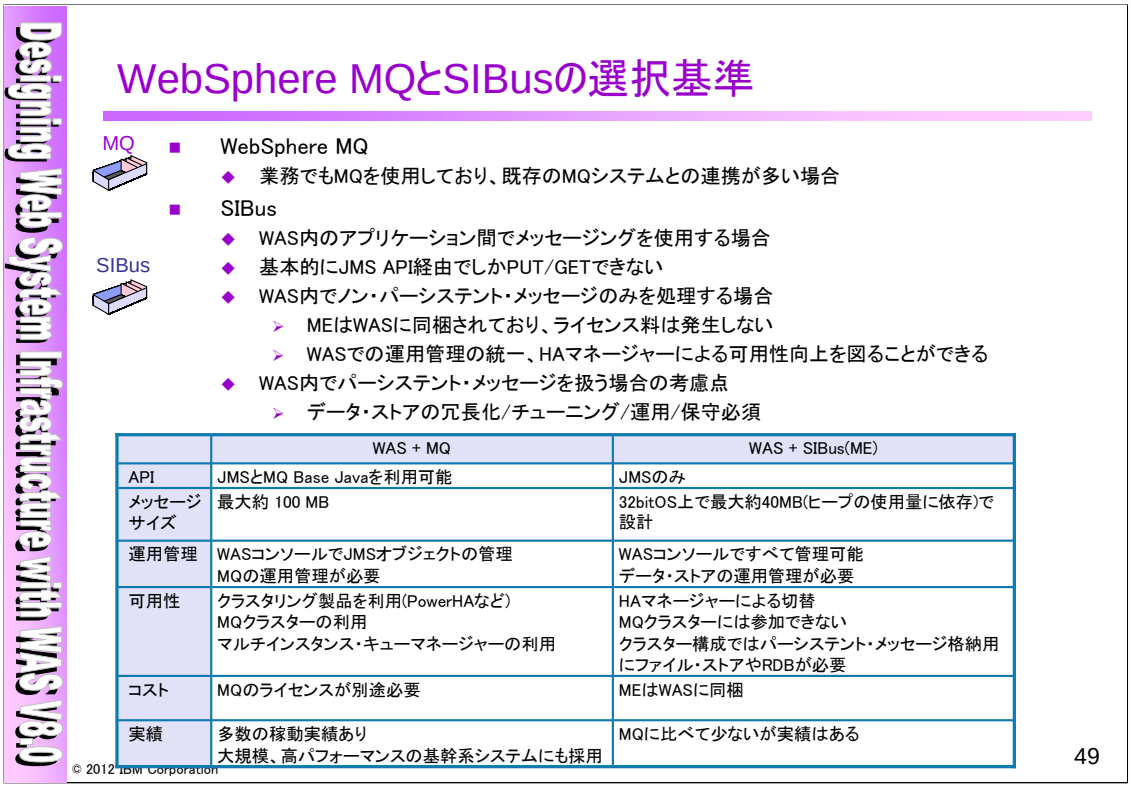


WAS V8.0の新機能として、MQリンクの送信チャネルにおいて接続先を複数指定することができるようになりました。

この事でマルチインスタンス・キューマネージャー構成のMQ環境との連携が可能になっています。



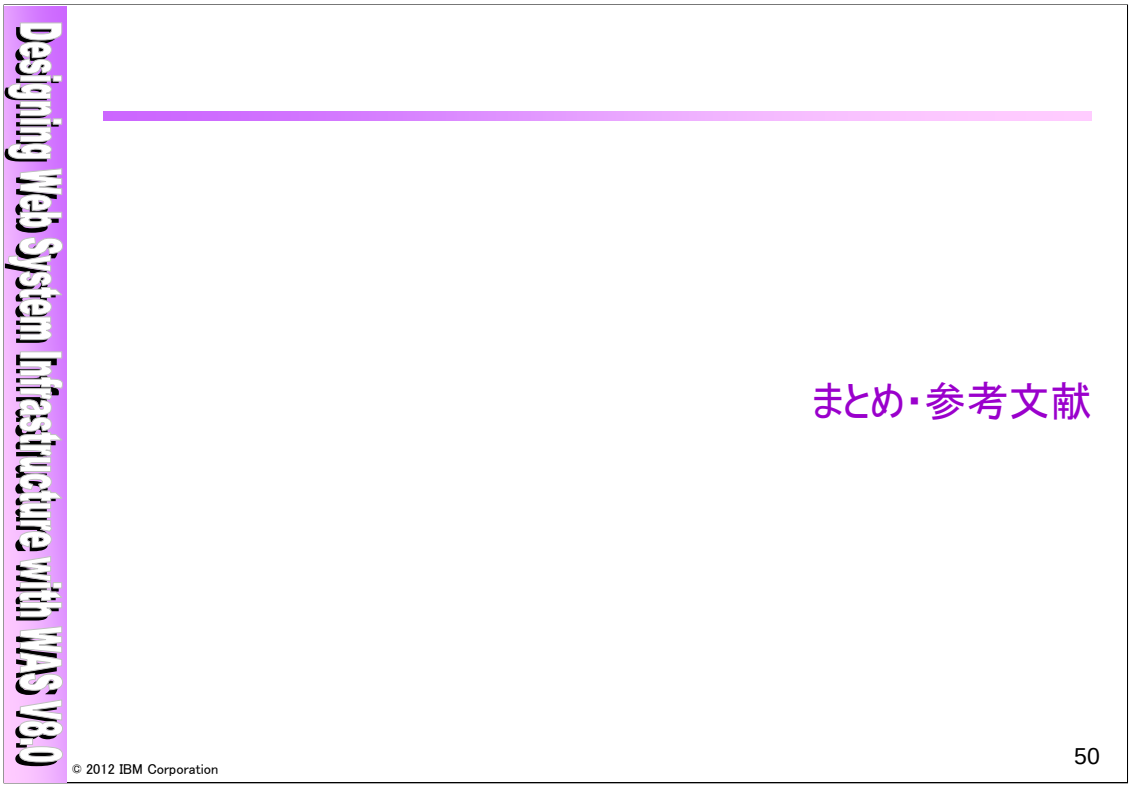
この節では、WebSphere MQとSIBusの選択基準について解説します。

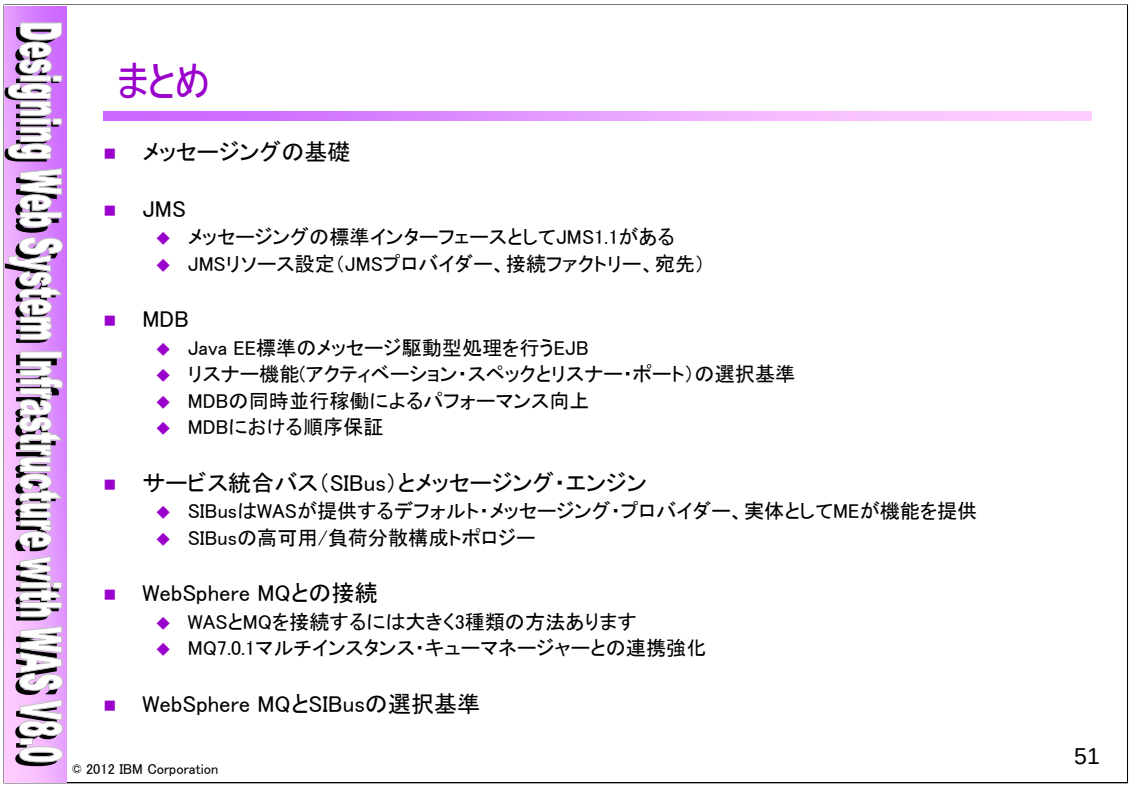


WebSphereMQとSIBusの選択基準についてまとめています。

WAS V8.0 Information Center: WebSphere Application Server と WebSphere MQ メッセージングとの比較

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rmj_jmsp_sibvmq.html


Designing Web System Infrastructure with WAS V8.0
まとめ・参考文献
© 2012 IBM Corporation
50



このページには、本セッションで解説した内容がまとめてあります。

Designing Web System Infrastructure with WAS V8.0

参考文献 (1/2)

- Information Center
  - WebSphere Application Server V8.0
    http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp
  - WebSphere Application Server V7.0
    http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp
  - WebSphere MQ V7.0
    http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp
- ワークショップ資料
  - WebSphere Application Server V8.0 アナウンスメント・ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was8_ws/
  - WAS V7.0 最新動向ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was7_update/
  - WebSphere Application Server V7.0によるWebシステム基盤設計ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was7_guide/index.html
  - WebSphere Application Server V6.1による基幹システム設計ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was61_guide/index.html
  - WebSphere Application Server V6による基幹システム設計ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/was/was6_guide/
  - MQ-WAS連携ワークショップ
    http://www.ibm.com/developerworks/jp/websphere/library/wmq/was_mq_ws/

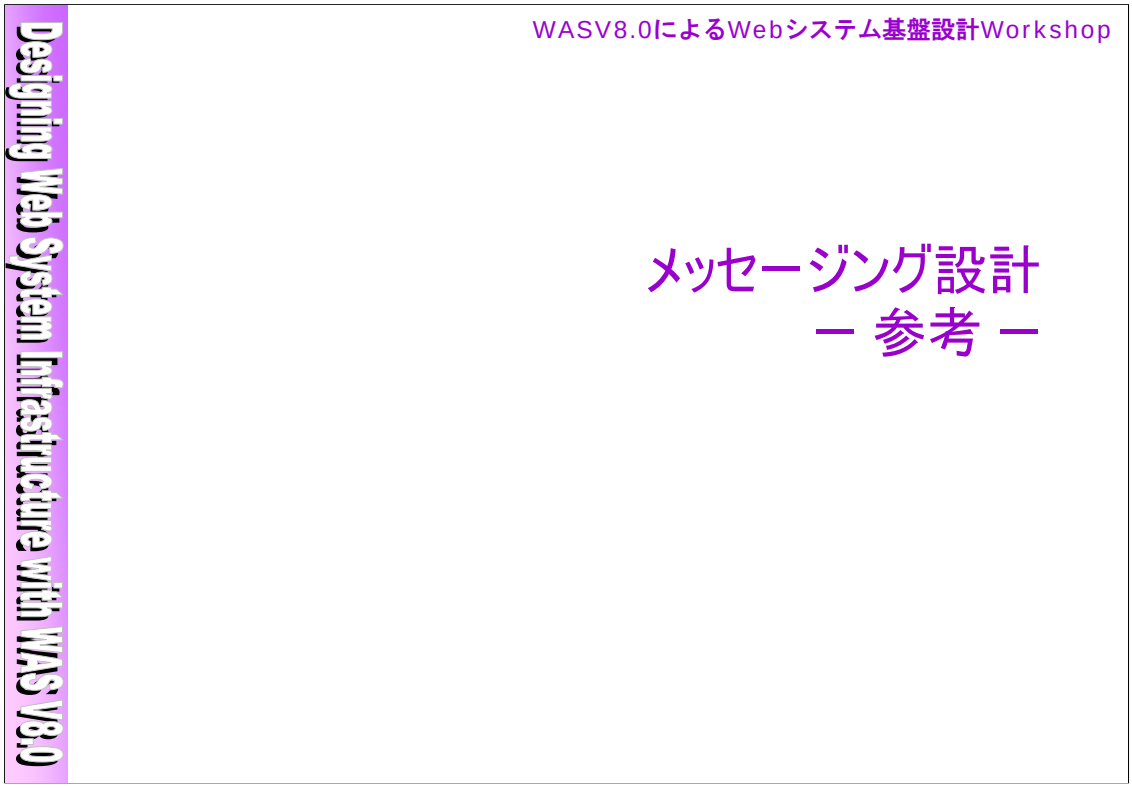
© 2012 IBM Corporation

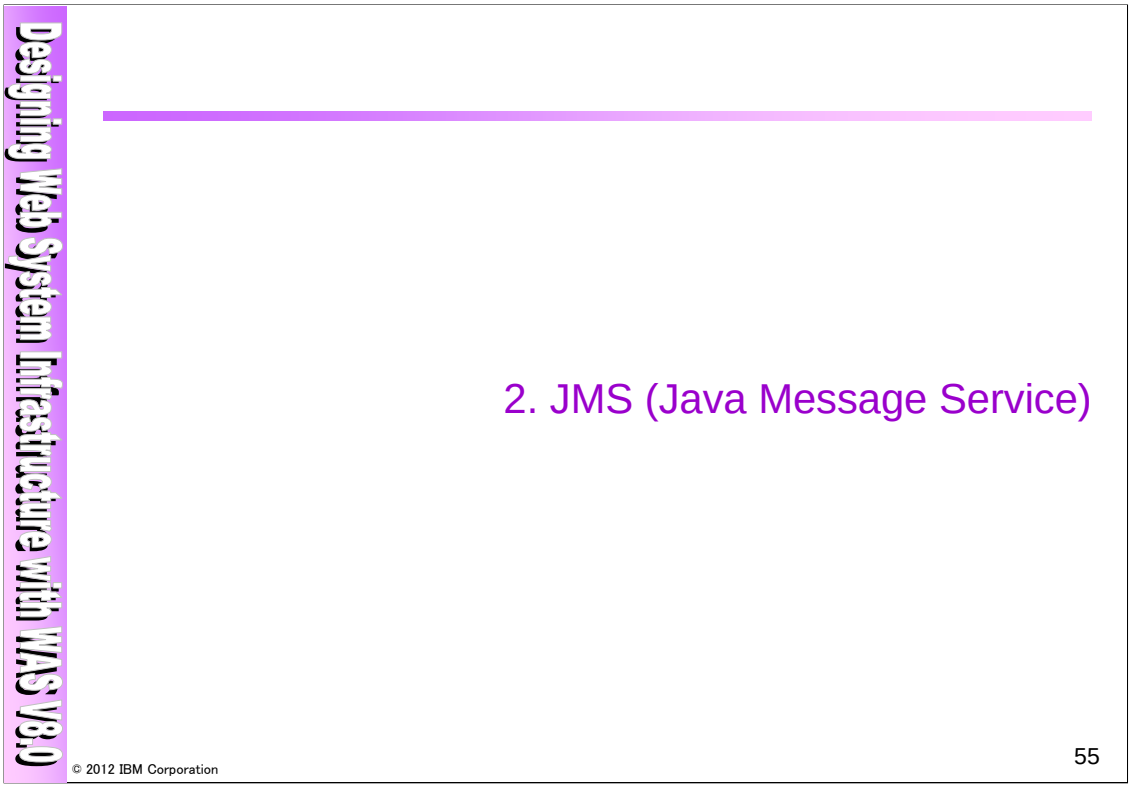
Designing Web System Infrastructure with WAS V8.0

参考文献 (2/2)

- その他
  - WebSphere Application Server V7.0 管理ガイド
    http://www.ibm.com/developerworks/jp/websphere/library/was/was7_adminguide/
  - WAS虎の巻: 第4回「JMSとメッセージング・サービス接続」
    http://www.ibm.com/developerworks/jp/websphere/library/was/toranomaki/4.html
  - Delivery ordering with the WebSphere MQ mesaging provider and message-driven bean (MDB) applications
    http://www-01.ibm.com/support/docview.wss?uid=swg21446463&myns=swgws&mynp=OCSSEQTP&mync=E
  - When to use ASF and non-ASF modes to process messages in WebSphere Application Server
    http://www.ibm.com/developerworks/websphere/library/techarticles/0611_titheridge/0611_titheridge.html
  - PM18782: SUPPORT EQUIVALENT TO LISTENERPORT INITIALSTATE=STOPPED WHEN USING ACTIVATION SPECS
    http://www-01.ibm.com/support/docview.wss?uid=swg1PM18782
  - Planning Service Integration Bus and WebSphere MQ Usage
    http://www-01.ibm.com/support/docview.wss?uid=swg21497341

© 2012 IBM Corporation

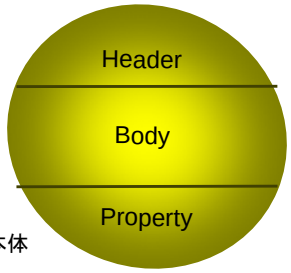

Designing Web System Infrastructure with WAS V8.0
WASV8.0によるWebシステム基盤設計Workshop
メッセージング設計
－ 参考 －


Designing Web System Infrastructure with WAS V8.0
2. JMS (Java Message Service)
© 2012 IBM Corporation
55

Designing Web System Infrastructure with WAS V8.0

JMS概要　メッセージ・フォーマット

- JMSの規定
  - メッセージ・フォーマット仕様
    - メッセージ・ヘッダー（Header）
      - 宛先など、主にメッセージング・プロバイダーが必要とする情報を保持する領域です。
    - メッセージ・プロパティー（Property）
      - 名前：値の組み合わせで、アプリケーションが必要な情報を保持する領域です。
    - メッセージ本体（Body）
      - 扱うデータ形式毎にインターフェイスが定義されています。
      - Selectorで条件指定には利用できません。
  - メッセージ本体（Body）の種類
    - TextMessage：
      - テキストを保持するメッセージ本体
    - MapMessage：
      - JavaのMapクラスのようにキーと値を保持するメッセージ本体
    - ObjectMessage：
      - Serialize可能なオブジェクトを保持するメッセージ本体
    - etc…



© 2012 IBM Corporation

J2EE 1.4やJavaEE 5.0/6.0の仕様は、JMS 1.1の仕様を含みます。JavaEE 6.0に準拠しているWAS V8.0でも、JMS 1.1が利用可能です。JMSでは、主にメッセージングに必要なインターフェイスや、メッセージの仕様が規定されています。メッセージング・プロバイダー内部のメッセージのやり取りについて規定しているものではありません。

メッセージングに必要なインターフェイスとしては、PTP型通信やPub/Sub型通信を行う上で必要な、接続状態、セッション情報などを操作するためのインターフェイスが定義されています。

メッセージの仕様では、メッセージの構造として、メッセージ・ヘッダー、メッセージ・プロパティー、メッセージ本体が、それぞれ何をするための物なのかを記述されています。

メッセージ・フォーマットの詳細については、「メッセージ・ヘッダー」、「メッセージ・プロパティー」、「メッセージ本体」を参照してください。

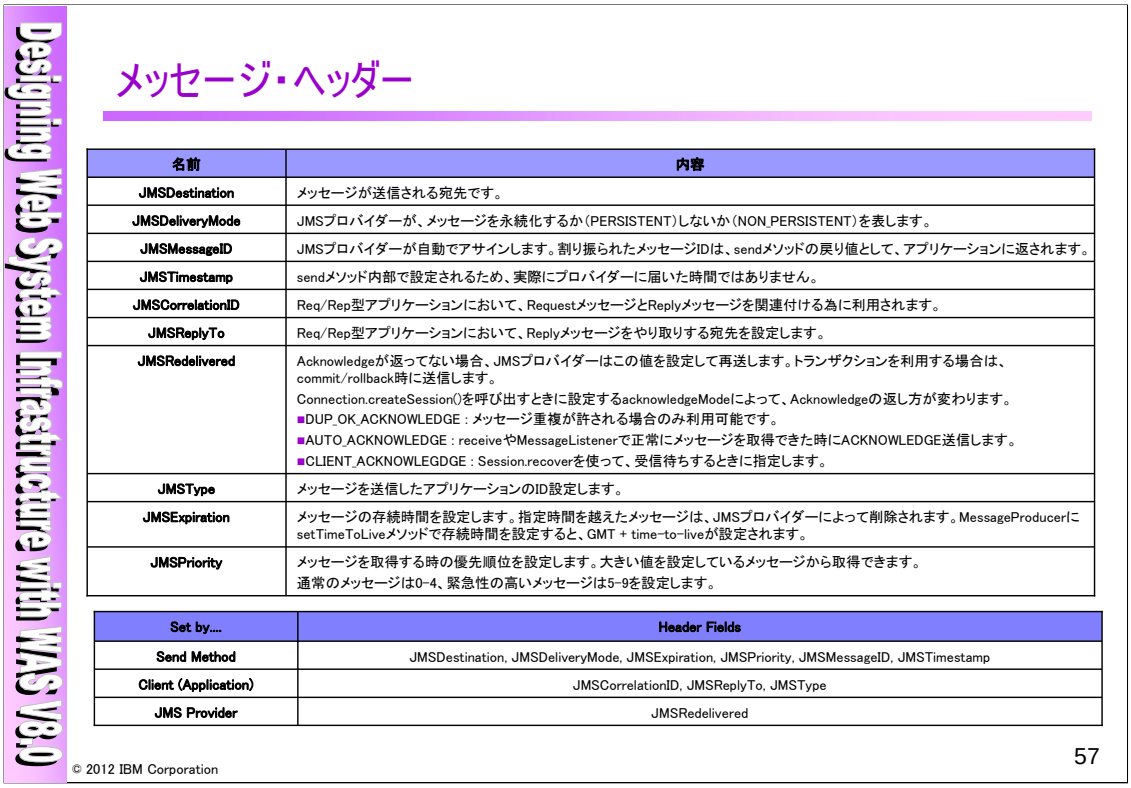
その他に、JMSではメッセージのフォーマットや、メッセージの信頼性が定義されています。

メッセージの型のうち、使われる機会が多いのは、以下の種類です。

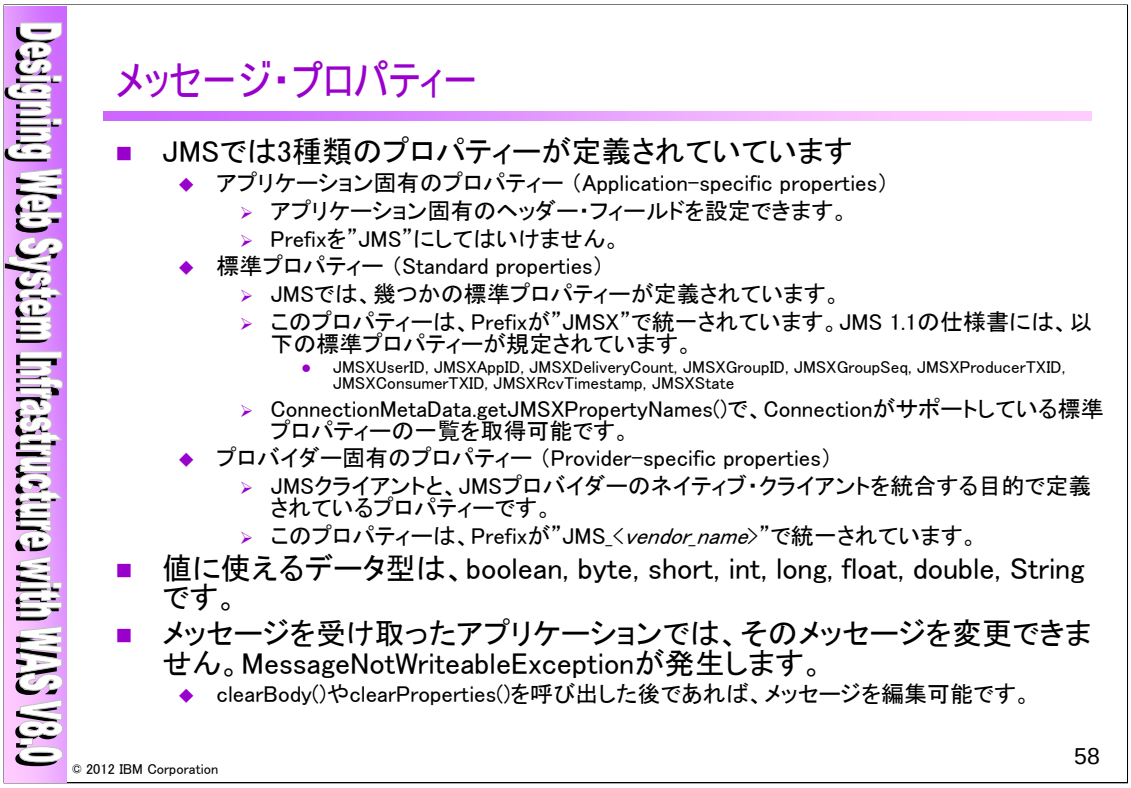
・TextMessage：メッセージ本体がテキストを保持します。テキストしか送れない為、受信後にメッセージを解析する処理が必要になります。

・MapMessage：メッセージ本体が、java.util.Mapのように、キーと値を保持しています。その為、送信アプリケーションと受信アプリケーションでは、どのキーがどんな値を保持しているかのみを知っていれば、通信可能になります。

・ObjectMessage：Serialize可能なオブジェクトを、メッセージ本体として送信することができます。MapMessageのように、受け取ったメッセージを解析する必要が無いので、社内システムなど、クラス定義を配布可能な範囲内で利用するには最適です。



メッセージ・ヘッダーは”JMS”からはじまる変数名が与えられています。ヘッダー情報は、sendメソッド実行時に設定されるもの、アプリケーションで設定するべきもの、JMSプロバイダーが自動的に設定するものがあります。



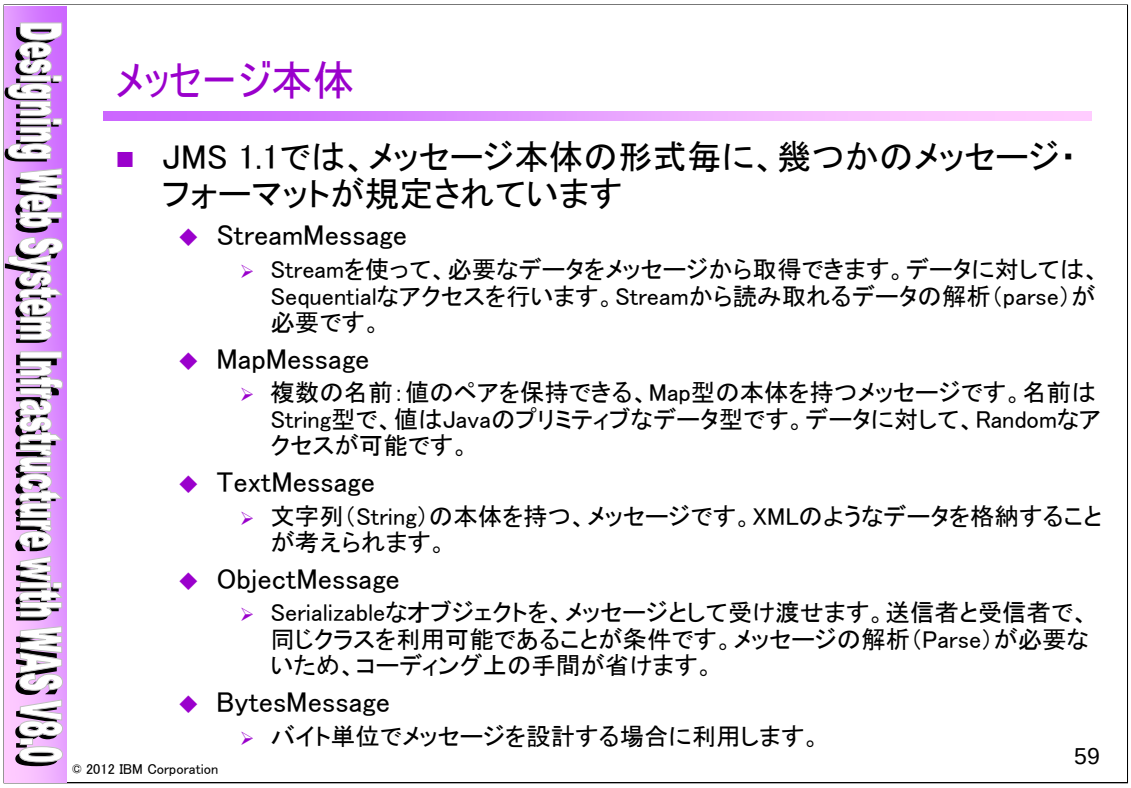
メッセージ・ヘッダー以外にも、メッセージに関する追加情報を、メッセージ・プロパティーとして登録できます。メッセージ・プロパティーには、3種類のプロパティーが定義されています。

アプリケーション固有プロパティーは、JMSの開発者が自由に利用できるプロパティーです。ただし、”JMS”ではじまる名前は、JMSの仕様で予約されているので、利用することができません。

標準プロパティーは、JMSで規定されているプロパティーです。後ほど出てくるJMSXDeliveryCount（再送回数）も、標準プロパティーに含まれます。アプリケーションが利用するわけではなく、自動的に設定/利用されている項目がほとんどです。

プロバイダー固有プロパティーは、JMSプロバイダーとして利用しているアプリケーションから送信されたメッセージと、JMS以外のメッセージング・プロバイダーとして利用しているアプリケーションから送信されたメッセージの整合性を保つのが目的で、定義されているプロパティーです。

メッセージ・プロパティーの名前はString型で、値はboolean、byte、short、int、long、float、double、Stringなどの、プリミティブなデータ型です。メッセージを受け取ったアプリケーションでは、clearProperties()を行わない限り変更できません。



本体のデータ形式にあわせて、5種類のメッセージ・フォーマットが規定されています。

StreamMessageは、メッセージ本体からデータを取得する時に利用します。Streamで得られたデータは、アプリケーションで解析する必要がありますので、アプリケーションのコードは、多少複雑になる可能性があります。

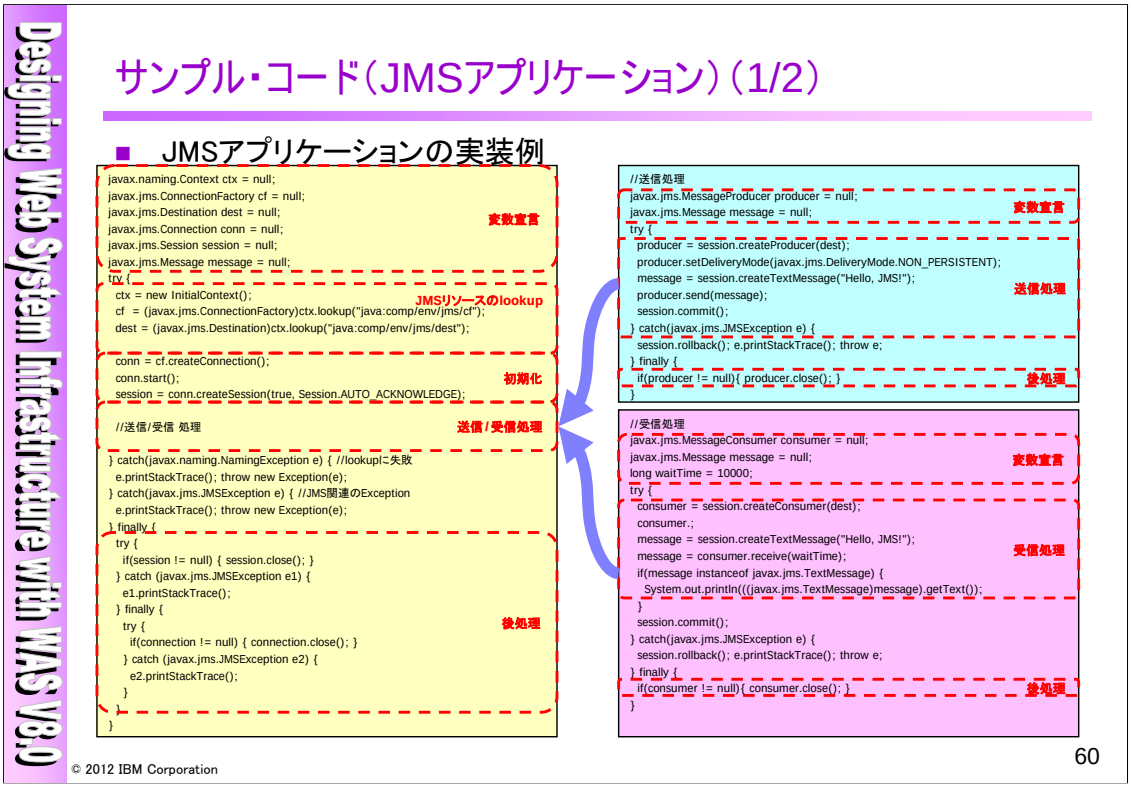
MapMessageは、名前：値ペアを複数保持させたい場合に利用します。得られたメッセージは、ほとんどの場合解析する必要がありませんので、アプリケーションのコードは比較的簡単なります。

TextMessageは、文字列の本体を持つシンプルなメッセージです。複数の値を受け渡すには、メッセージ・フォーマットを決定して、それに基づくメッセージの解析処理を行う必要があります。メッセージを解析する必要があるため、アプリケーションのコードも複雑になる可能性があります。

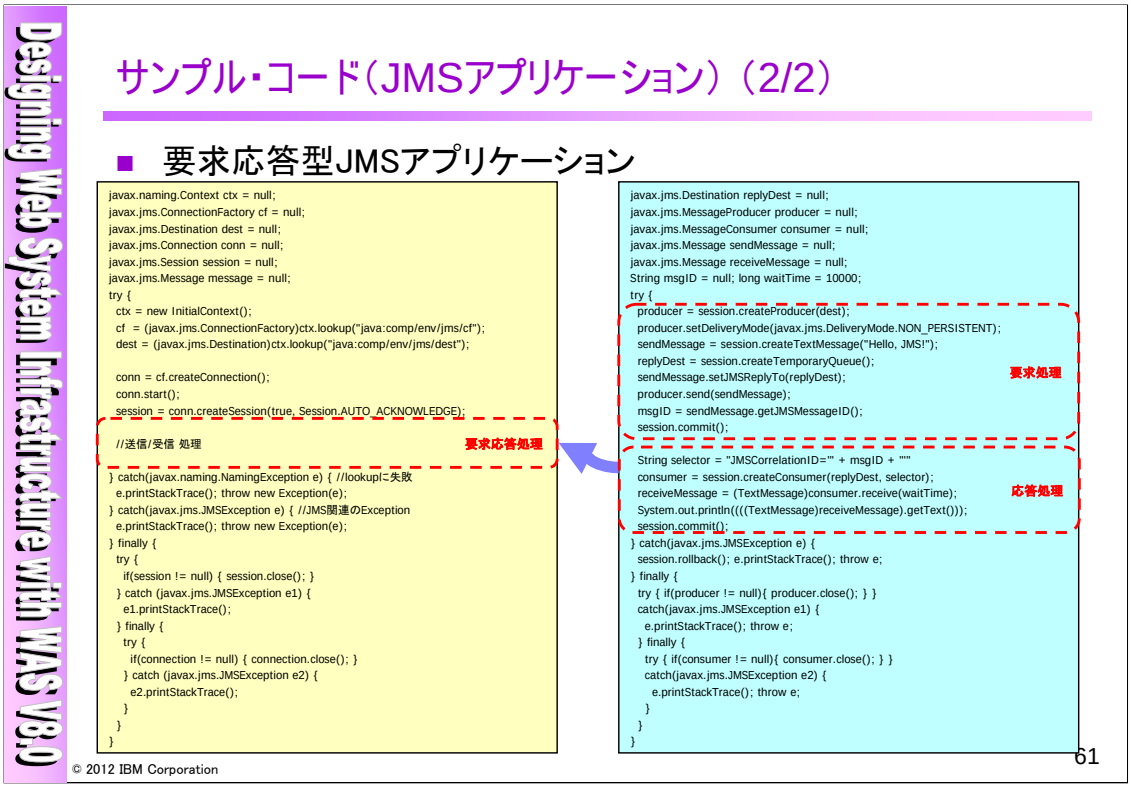
ObjectMessageは、Serializableのオブジェクトをメッセージ本体に設定して受け渡せます。送信者と受信者は、同じSerializableクラスを利用しなければなりません。メッセージの解析が必要ないので、アプリケーション・コードは比較的簡単になります。

ByteMessage

バイト単位でメッセージを設計する場合に利用します。メッセージの解析ロジックが、複雑になる可能性があります。



JMSアプリケーションのサンプルコードです。

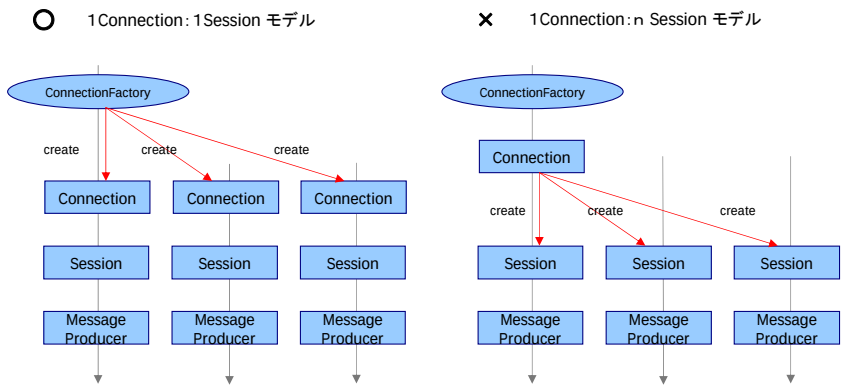


要求応答型JMSアプリケーションのサンプル・コードです。

Designing Web System Infrastructure with WAS V8.0

JMS1.1アプリケーション実装での注意点

- 1Connection：1Session
  - JMS1.1から、1Connection オブジェクトに対してSessionオブジェクトを1つだけ作成する1:1モデルを使用する制約が追加
    - ServletやEJBではクラス変数としてConnectionFactoryのみ宣言し、スレッド単位のメソッド内でConnectionとSessionの宣言を行う



© 2012 IBM Corporation

62

JMSプロバイダーに接続する際、次の手順を踏みます。

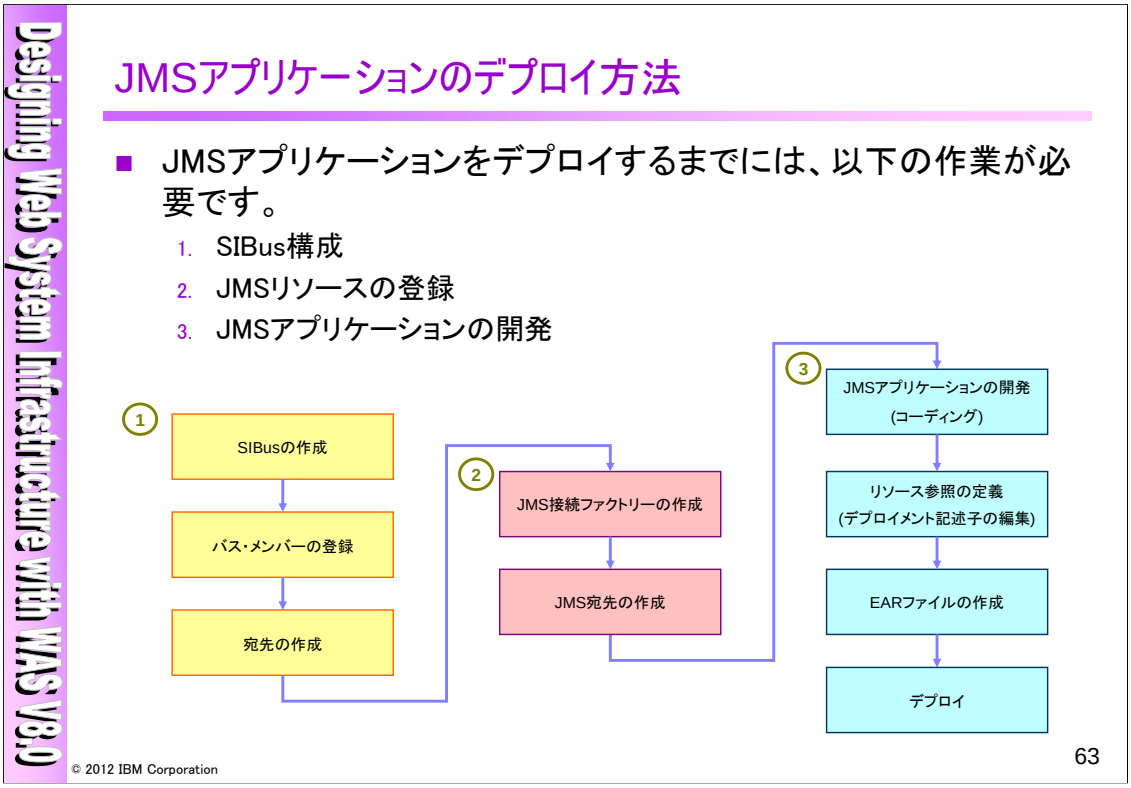
1.ConnectionFactoryオブジェクトをJNDIネームスペースからlookUp

2.ConnectionFactoryオブジェクトからConnectionオブジェクトを生成

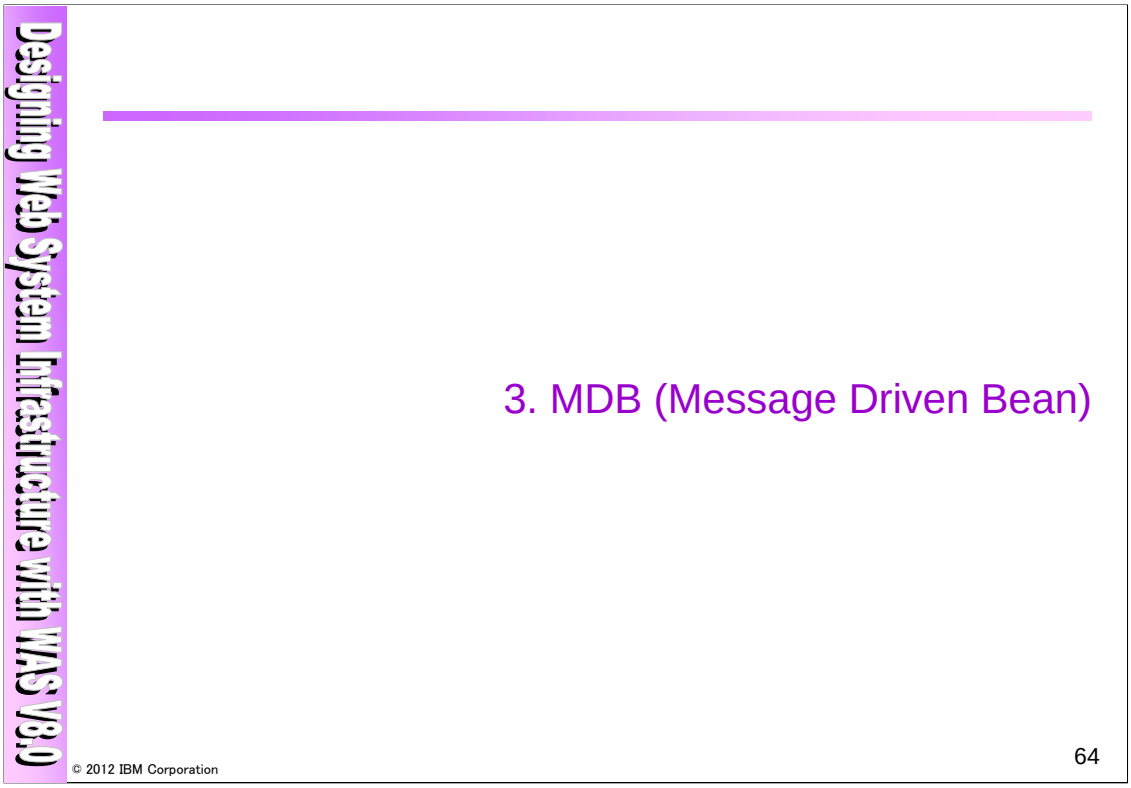
3.ConnectionオブジェクトからSessionオブジェクトを生成

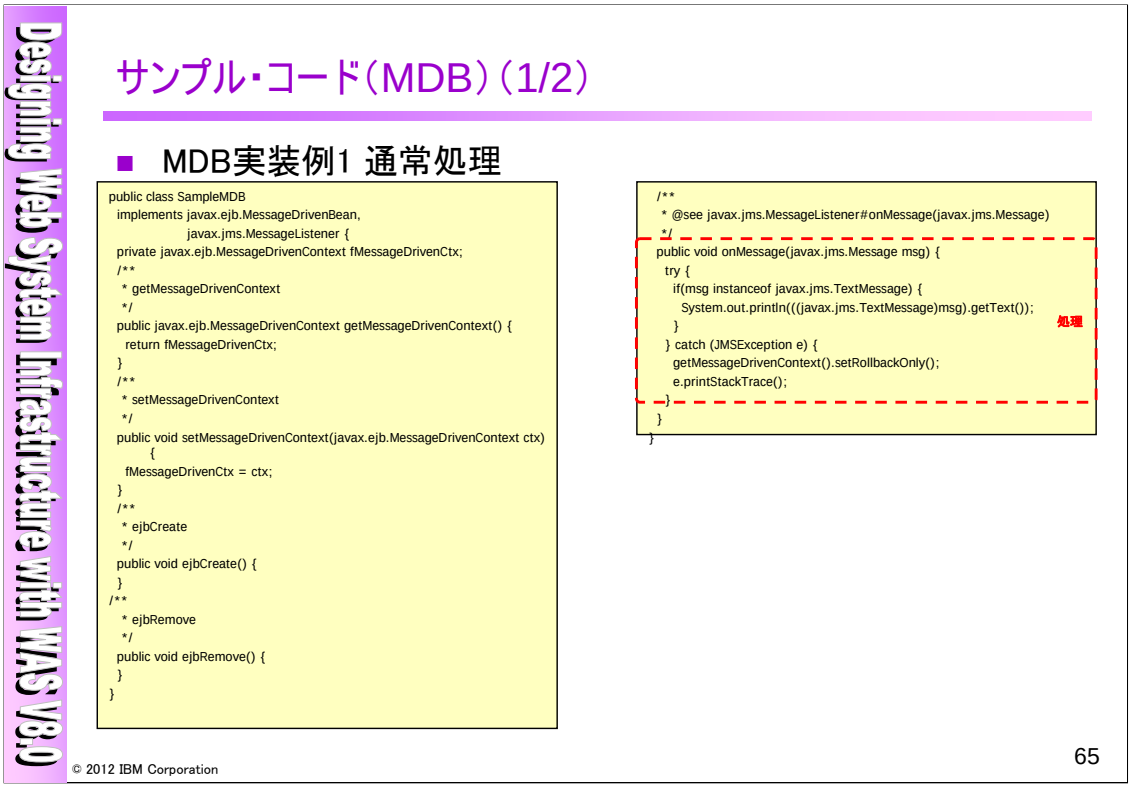
JMS1.1より前の仕様は、Connectionオブジェクトと Sessionオブジェクトの使用法について明確な規定がありませんでしたが、JMS1.1の仕様で「1つのConnectionオブジェクトに対し、1つのSessionオブジェクトを生成する」という制約が記述されました。その影響でJMS 1.0.2のSessionがスレッド・セーフだったのに対し、JMS 1.1のSessionはスレッド・セーフではなくなっています。

1つのConnectionオブジェクトから複数個のSessionオブジェクトを作成するモデルで実装されているアプリケーションは、改変を検討する必要があります。

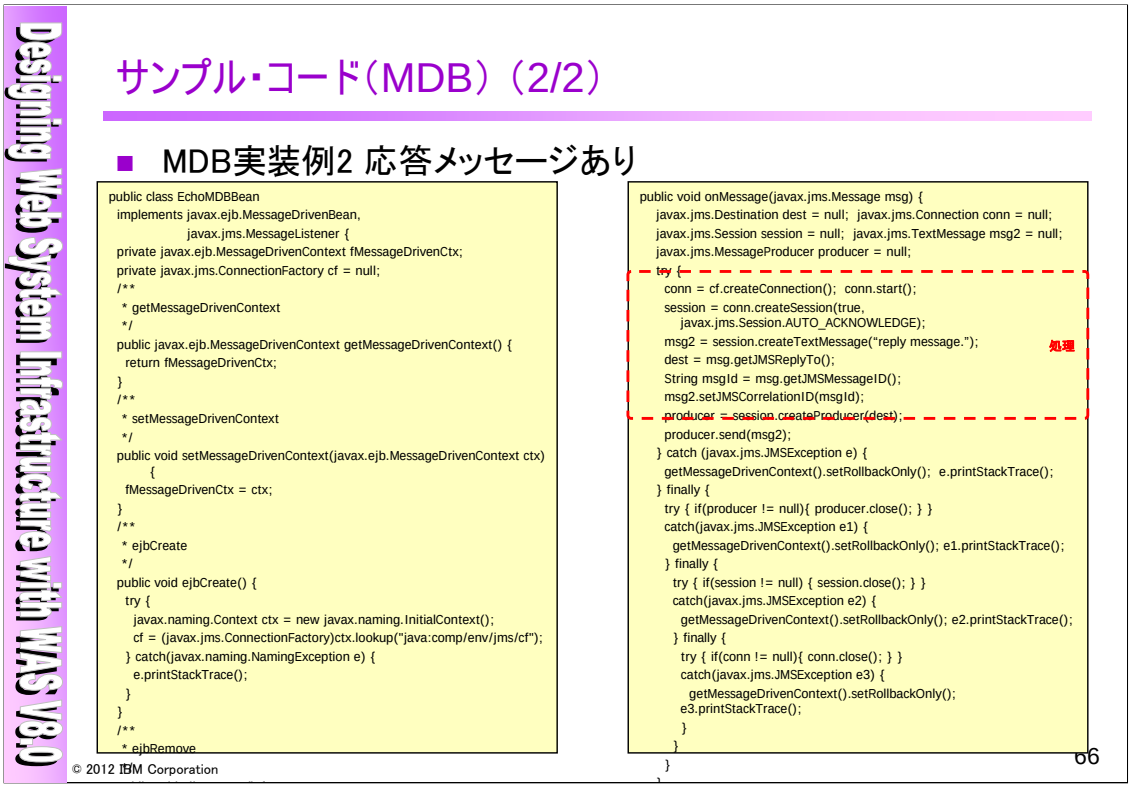


JMSアプリケーションをサーバーで稼動させるためには、SIBusの構成、JMSリソースの登録、アプリケーションの開発/デプロイを行う必要があります。

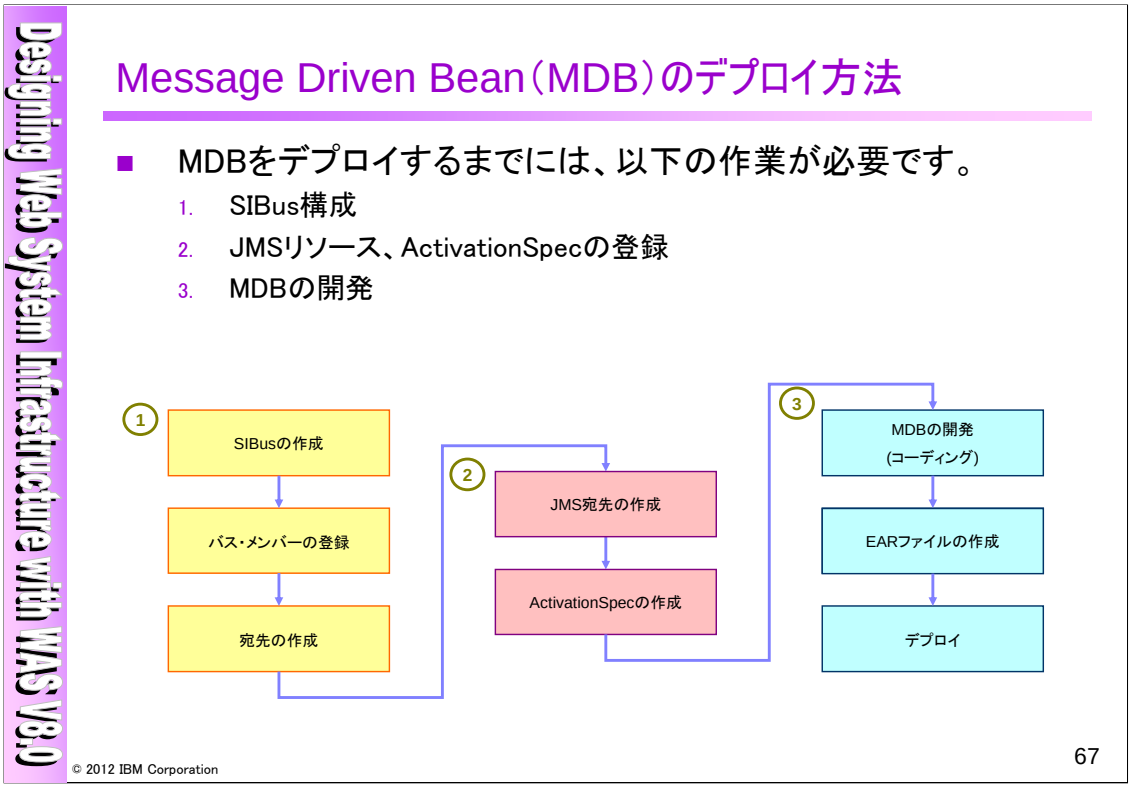

Designing Web System Infrastructure with WAS V8.0
3. MDB (Message Driven Bean)
© 2012 IBM Corporation
64



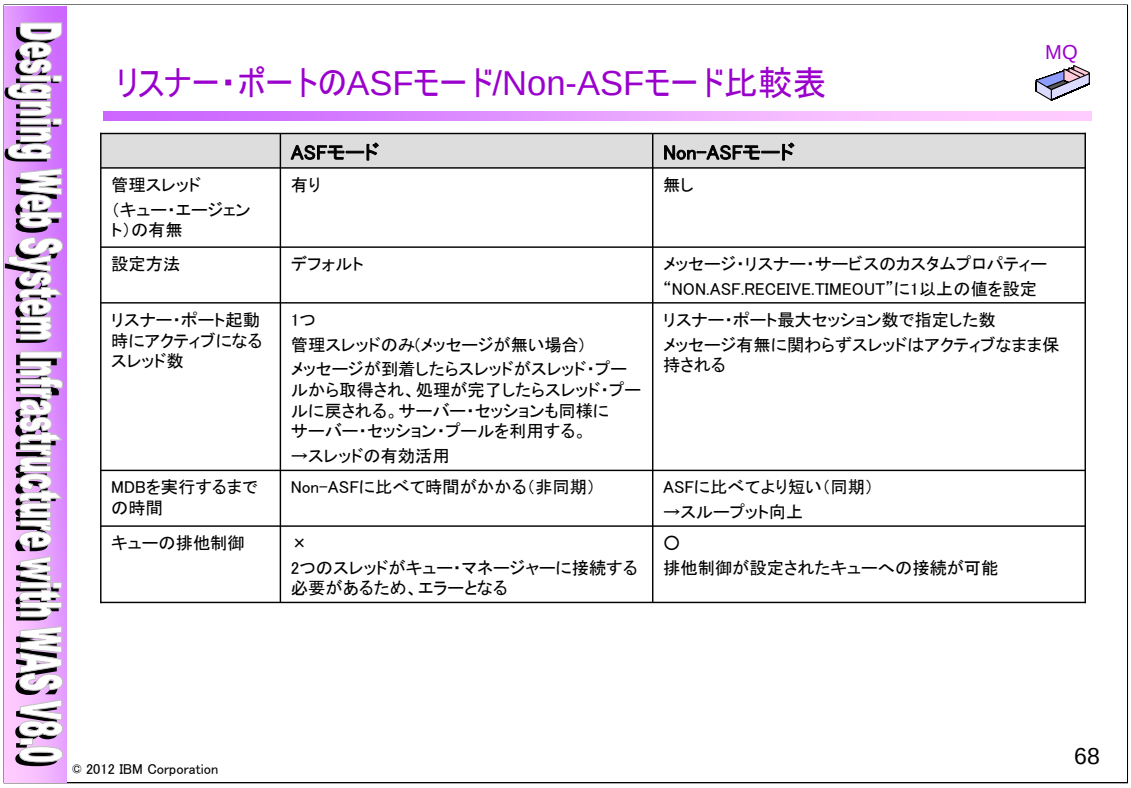
Container Managed Transaction（CMT）で稼動するMDBのサンプル・コードです。TextMessageの本体をSystem.outに出力しています。



Container Managed Transaction（CMT）で稼動するMDBのサンプル・コードです。要求メッセージのJMSMessageIDを、応答メッセージのJMSCorrelationIDに設定しています。

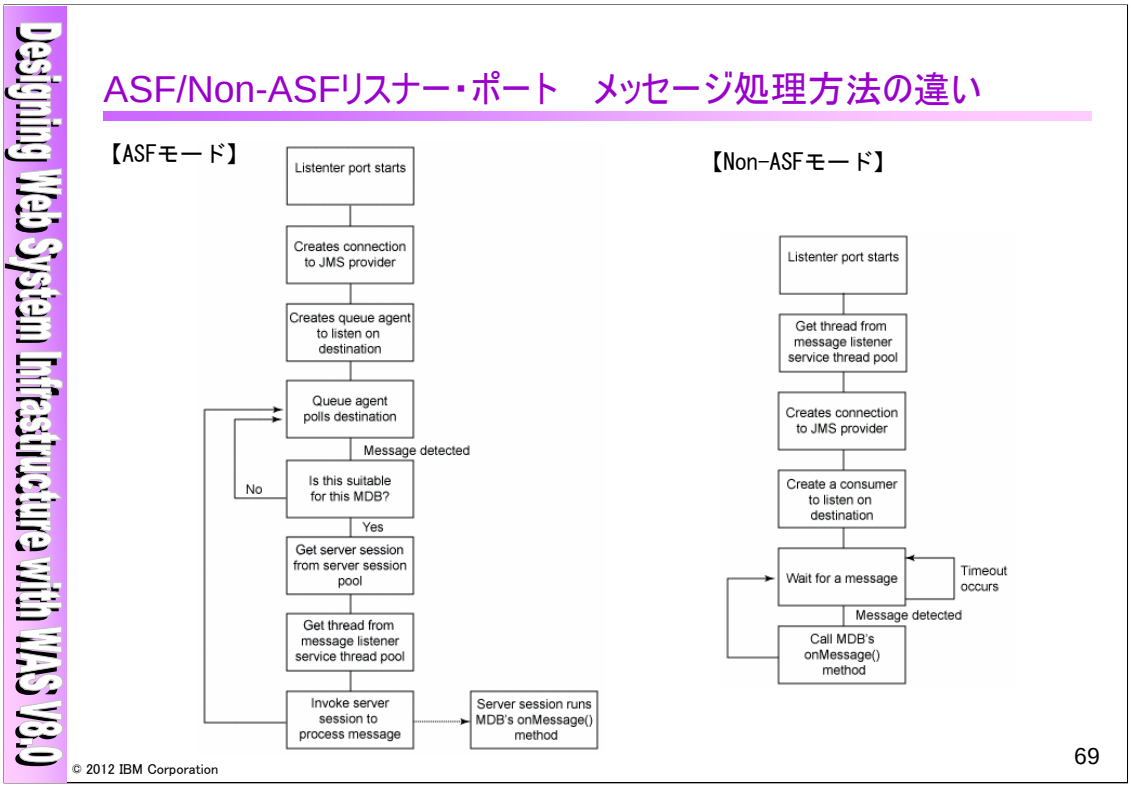


MDBをサーバーで稼動させるためには、SIBusの構成、JMSリソースやActivationSpecの登録、アプリケーションの開発/デプロイを行う必要があります。



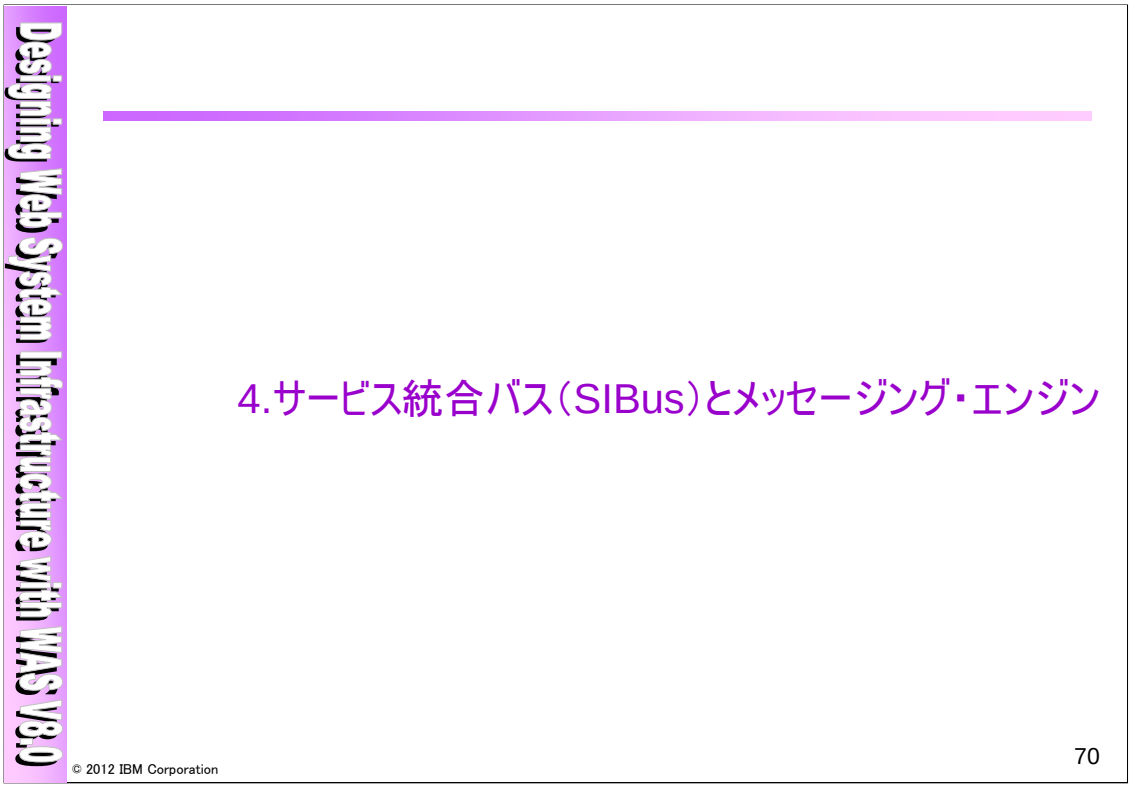
	ASFモード	Non-ASFモード
管理スレッド（キュー・エージェント）の有無	有り	無し
設定方法	デフォルト	メッセージ・リスナー・サービスのカスタムプロパティー“NON.ASF.RECEIVE.TIMEOUT”に1以上の値を設定
リスナー・ポート起動時にアクティブになるスレッド数	1つ 管理スレッドのみ(メッセージが無い場合) メッセージが到着したらスレッドがスレッド・プールから取得され、処理が完了したらスレッド・プールに戻される。サーバー・セッションも同様にサーバー・セッション・プールを利用する。 →スレッドの有効活用	リスナー・ポート最大セッション数で指定した数 メッセージ有無に関わらずスレッドはアクティブなまま保持される
MDBを実行するまでの時間	Non-ASFに比べて時間がかかる（非同期）	ASFに比べてより短い（同期） →スループット向上
キューの排他制御	× 2つのスレッドがキュー・マネージャーに接続する必要があるため、エラーとなる	○ 排他制御が設定されたキューへの接続が可能

リスナー・ポートのASFモードとNon-ASFモードの違いについて表にまとめました。



リスナー・ポートのASFモードとNon-ASFモードのメッセージ処理方法をフローチャートで説明しています。

ASFリスナー・ポートの動作はアクティベーション・スペックと同様です。

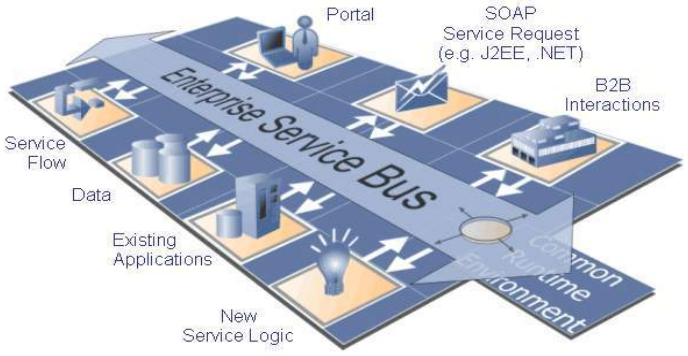


この節では、SIBusとメッセージング・エンジンについて解説します。

Designing Web System Infrastructure with WAS V8.0

SIBusとESB

- ESB概要（参考）
  - ESBは、分散システムを構築するためのアプローチであるSOAの通信基盤
  - IBMのBusiness Integration Reference Architectureでは、各種サービスを緩やかに連携するための仲介および通信を行うレイヤーに位置づけられている
    - サービス間の連携に必要な作業を代わりに行うことで疎結合を実現する



© 2012 IBM Corporation
71

ESBは各種サービスを疎結合するための仲介および通信を行うレイヤーに位置づけられています。同期・非同期通信、複数のプロトコルをサポートする通信経路です。

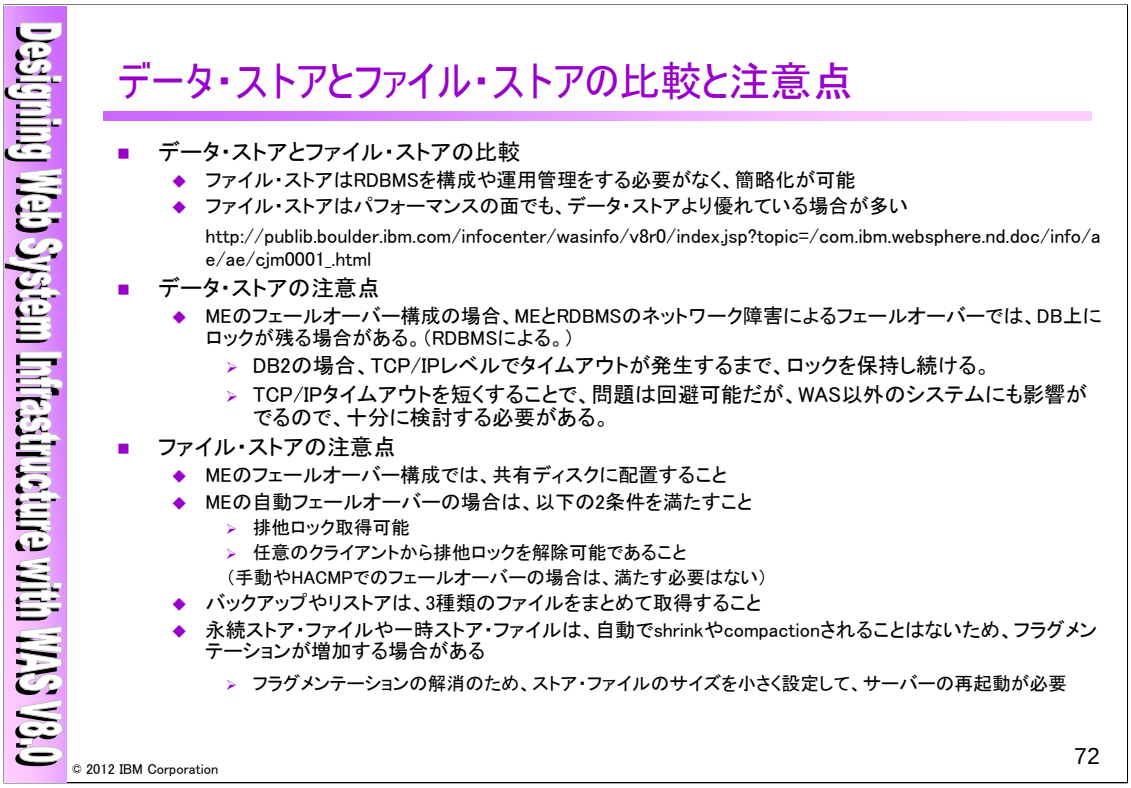
各種サービスが抽象的なインターフェースを持つことで、そのサービスの実装や稼動するプラットフォームを意識せず利用することができます。これはSOAがめざす組み合わせによるシステムの構築への大きな前進と言えます。しかしこのままでサービスを連携するにはいくつか考慮点があります。

•まずサービス連携に当たって、連携相手のネットワーク上のロケーションなどの情報が必要です。例えばWebサービスの場合、サービス・リクエスターの中でサービス・プロバイダを表すWSDLのロケーションを記述する必要があります。

•またサービス・リクエスターがサービス・プロバイダを呼び出すには、双方が同じトランスポート・プロトコルを使用する必要があります。

•さらにリクエスターの要求によって、別のサービスへルーティングしたい場合、サービス・リクエスターの中でそのルーティングの制御ロジックを記述する必要があります。

上記のように個々のサービスが明確に定義されているインターフェースを持っていても、連携する際、通信レベルでお互いに依存することになります。ESBの役割は、上記のようなサービス間の連携に必要な作業を個々のサービスの代わりに行なうことで、サービス間の通信レベルでの依存をできるだけ取り除き、疎結合を実現することです。



メッセージング・エンジンのメッセージ・ストアとしてデータ・ストアとファイル・ストアを選択することができます。

データ・ストアと比較して、ファイル・ストアには、いくつかのメリットが得られます。

・構成簡略化:ファイル・ストアを利用すると、OS上のファイルとしてデータを出力する為、RDBMSが不要です。

・運用効率向上:RDBMSの運用管理を行う必要がなくなります。

・パフォーマンス向上:データ・ストアにおけるME-RDBMS間のトランザクションは、ファイル・ストアではありませんので、処理が効率化され、パフォーマンスの向上が見込めます。ただし、パフォーマンスについては、データ・ストアに利用するDBのパフォーマンスに依存します。

データ・ストアの注意点です。

MEのフェール・オーバー構成の場合、MEとRDBMSのネットワーク障害によるフェール・オーバーでは、DB上にロックが残る場合があります。(RDBMSによります。)

DB2の場合、TCP/IPレベルでタイムアウトが発生するまで、ロックを保持し続けます。TCP/IPタイムアウトを短くすることで、問題は回避可能ですが、WAS以外のシステムにも影響がでますので、十分に検討する必要があります。

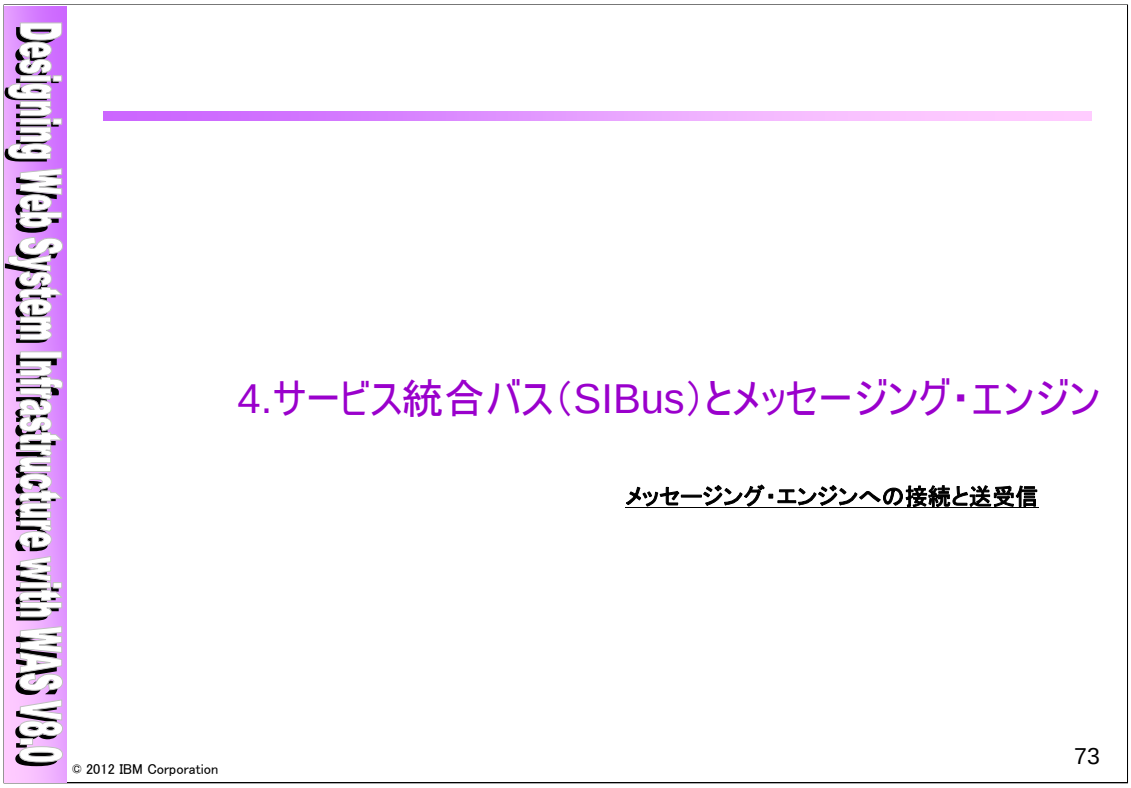
次に、ファイル・ストア利用時の注意点です。

MEのフェールオーバー構成をとる場合、ファイル・ストアは共有ディスクに配置し、MEを引き継ぐアプリケーション・サーバーからアクセスできるように構成する必要があります。その際、各アプリケーション・サーバーが、MEのファイル・ストアに同じファイルパスでアクセスできるように構成します。

MEが起動すると、アプリケーション・サーバーは、ログ・ファイルとストア・ファイルに排他ロックをかけます。1つファイル・ストアに1つのMEしかアクセスしないよう制御する仕組みになっています。このため、フェール・オーバー後にもファイルにアクセスできるファイル・システムである必要があります。

詳細については、以下のInformation Center「ファイル・ストアの高可用性についての考慮事項 」を参照してください。

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cjm1460_.html

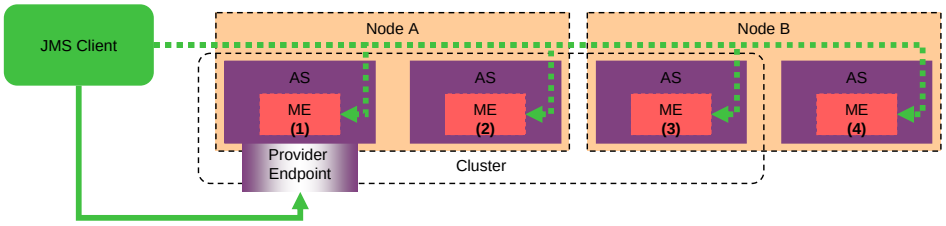


この節では、SIBusとメッセージング・エンジンについて解説します。

Designing Web System Infrastructure with WAS V8.0

接続先メッセージング・エンジンの選択

- JMS接続ファクトリーのデフォルト設定では、バス内の複数のMEのうちブートストラップ・エンドポイントから最も近いMEに接続します
  - 「近い」とは、同一サーバー、同一クラスター（同一ノード優先）、同一セルの順です。
  - 接続の接近性を指定することで、接続先のMEを限定できます。
    - 同一バス（デフォルト）、クラスター、ホスト、サーバーの指定が可能です。
  - ターゲット・タイプは、バス・メンバー名、メッセージング・エンジン・グループ名、メッセージング・エンジン名のいずれかで指定できます。
    - ターゲット重要度を「必須」にすると、ターゲット・タイプの指定に該当しないMEに対しては接続しません。



74

© 2012 IBM Corporation

アプリケーションがどのMEに接続するかは、メッセージが目的のメッセージ・ポイントに到達するまでの経路に影響します。

JMS接続ファクトリーは、設定したブートストラップ・エンドポイントから、最も近いMEを選択します。

(1)同一サーバー

(2)同一クラスター＆同一ノード

(3)同一クラスター

(4)同一セル

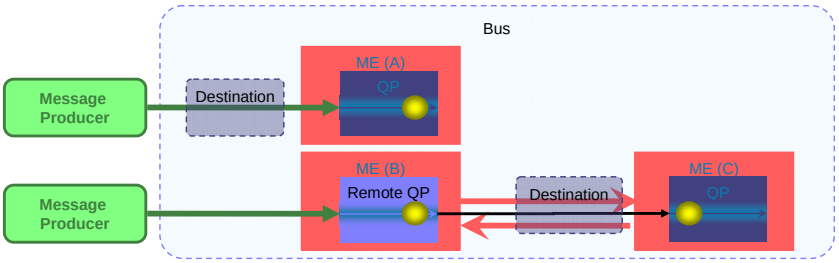
ブートストラップ・エンドポイントが指定されていない場合は、アプリケーションが稼動するサーバーのブートストラップ・エンドポイントに問い合わせます。

ターゲットを指定している場合、接続先がそのターゲットに優先、もしくは限定されます。ターゲット・タイプとしては、バス・メンバー名、メッセージング・エンジン名、メッセージング・エンジン・グループ名が選択可能です。

Designing Web System Infrastructure with WAS V8.0

メッセージの送信

- 接続先のMEに、宛先に対応するキュー・ポイント（QP）が存在する場合、アプリケーションからのメッセージは直接そのキュー・ポイントに格納されます
- 宛先に対応するキュー・ポイントが存在しない場合、送信者からのメッセージは、一旦接続先のMEに保管されます
  1. 接続先ME（B）は、受け取ったメッセージをリモート・キュー・ポイント（Remote QP）に格納します。
  2. リモート・キュー・ポイントは、管理コンソールでMEのランタイム表示を行うことで、格納されているメッセージを確認可能です。
  3. 送信者は、ME（B）に接続している場合、ME（C）が停止していても、メッセージを送信できます。



75

© 2012 IBM Corporation

メッセージを送信する際、宛先に対応するキュー・ポイントが接続先MEに存在する場合は、直接そのキュー・ポイントにメッセージを届けることができます。

これに対して、宛先に対応するキュー・ポイントがその時点では存在しない場合は、直接そのキュー・ポイントにメッセージを届けられません。

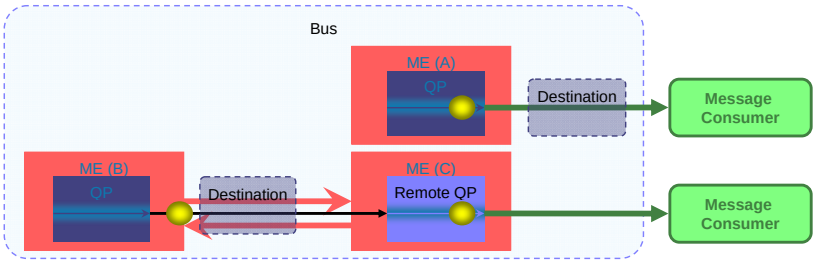
(1)接続先MEにリモート・キュー・ポイントを生成して、一旦そこにメッセージを格納します。
(2)アプリケーションには、リモート・キュー・ポイントにメッセージを格納できた時点で処理が戻ります。
(3)リモート・キュー・ポイントから目的キュー・ポイントにメッセージが送信されます。

目的のキュー・ポイントにメッセージが届いた時点では、ME（B）上のリモート・キュー・ポイントにメッセージが残ったままです。このメッセージが完全に消去されるのは、受信アプリケーションが、ME（C）のQPからメッセージを受け取って、トランザクションをcommitした時点です。

Designing Web System Infrastructure with WAS V8.0

メッセージの受信

- 接続先のMEに、宛先のキュー・ポイントが存在する場合、アプリケーションは、直接そのキュー・ポイントのメッセージが取得できます
- 宛先に対応するキュー・ポイントが存在しない場合、受信者からの要求は、接続先MEを経由して、キュー・ポイントが存在するMEに配信されます
  1. 接続先ME（C）は、メッセージ要求をME（B）のキュー・ポイントに送信します。
  2. メッセージ要求がME（B）で受信されると、ME（B)のキュー・ポイントのメッセージが、ME（C）上のリモート・キュー・ポイントに送信されます。
  3. Message Consumerは、ME（C）からメッセージを受信します。



76

© 2012 IBM Corporation

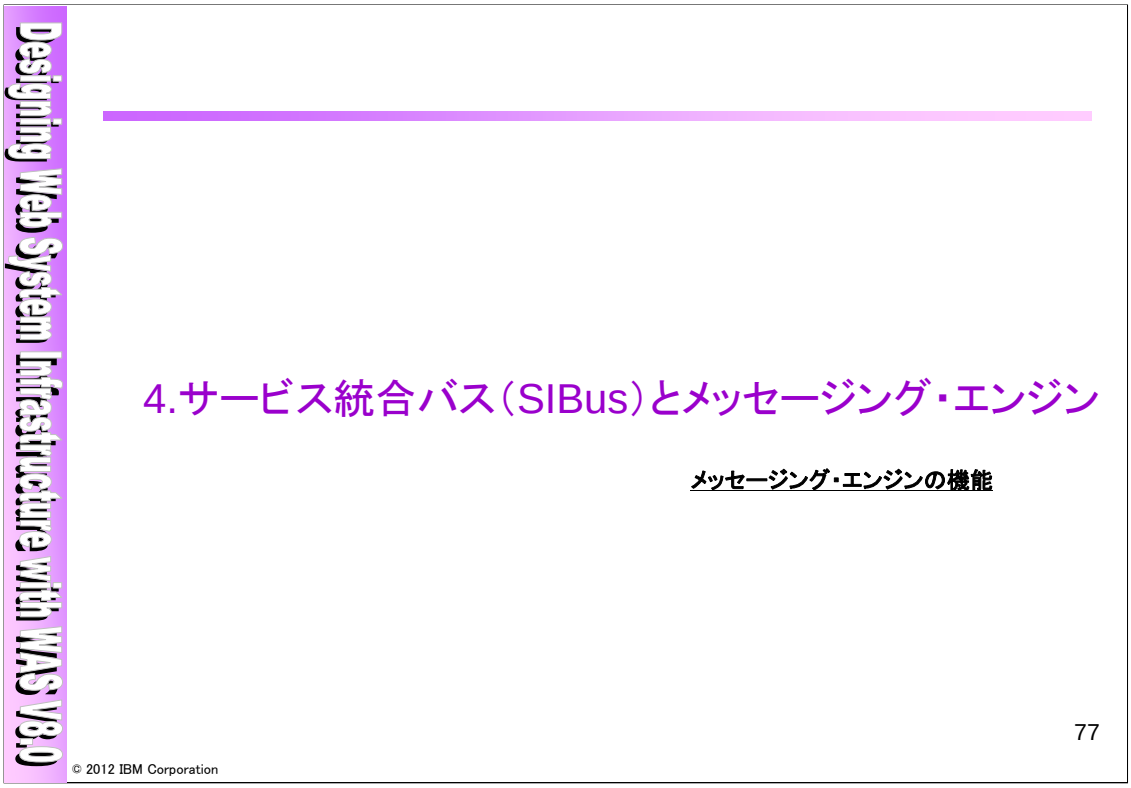
メッセージを受信するとき、接続先MEにメッセージ・ポイントが存在する場合は、そこから直接メッセージを取得できます。

これに対して、接続先MEにメッセージ・ポイントが存在しない場合は、直接メッセージを取得できません。

(1)接続先ME（C）はリモート・キュー・ポイントを作成します。
(2)メッセージ要求（※）を他のME（B）に送信します。
(3)ME（B）にメッセージが存在すると、そのメッセージは、ME（C）のリモート・キュー・ポイントに届きます。
(4)アプリケーションはリモート・キュー・ポイントからメッセージを取得します。

※メッセージ要求には、メッセージ・セレクター（後述）が含まれます。

リモート・キュー・ポイントでメッセージを取得した時点では、ME（B）のQPには、メッセージが残ったままです。このメッセージが完全に消去されるのは、受信アプリケーションが受信トランザクションをcommitした時点です。



この節では、SIBusとメッセージング・エンジンについて解説します。

Designing Web System Infrastructure with WAS V8.0

メッセージ・セレクター

- メッセージ・セレクター（Selector）とは、メッセージを選別して受信する為の機能
  - メッセージ受信側がセレクターで条件を指定します。
  - 条件に適合しないメッセージは、宛先に残されます。
  - このため、メッセージの順序性は保証されなくなります。
  - 条件指定に利用できるのは、メッセージ・ヘッダーとプロパティーのみです。メッセージ本体に対しては条件指定できません。
  - 条件指定方法
    - Javaの基本データ型を利用可能です
    - 構文はSQL方式の条件指定が可能です。条件はAND/ORで連結可能です。
  - 指定場所
    - JMSクライアントのコード中で、Message Consumer作成時に指定する

```
String selector = "JMSCorrelationID='" + msgId + "'";
MessageConsumer consumer = session.createConsumer(replyqueue, selector);
```

    - （MDBの場合は）ActivationSpecのプロパティーとして管理コンソールから指定する（以下の図）
      または、MDBのデプロイメント・ディスクリプターで指定する

78

© 2012 IBM Corporation

宛先に多様なメッセージが格納されている場合、目的のメッセージのみを取得するために、メッセージ・セレクターという機能が実装されています。これは、JMS 1.1に規定されていてる条件指定方法で、SQLの条件指定によく似ています。（SQL97のサブセットです。）

セレクターで条件指定に利用できるのは、メッセージ・ヘッダーとメッセージ・プロパティーで、メッセージ本体については、条件指定には利用できません。

Javaの基本データ型を条件指定に利用可能です。

文字列はシングル・クォート(') でくくります。エンコードはUnicodeです。

'literal' , 'literal''s'

数値はJavaのlong型と同じ範囲をサポートします。

浮動小数は、Javaのdouble型と同じ範囲をサポートします。

7E3 , -57.9E2

7. , -95.7 , +6.2

Boolean型はTRUEかFALSEで指定します。

SQL方式の条件指定が可能です。条件指定は、AND/ORで連結可能です。

"NumberOfOrders > 1"

"age BETWEEN 15 AND 19" == "age >= 15 AND age <= 19"

"age NOT BETWEEN 15 AND 19"

"Contry IN('UK', 'US', 'France')

"phone LIKE '12%3' : 123,12293はfalse、1234はtrue

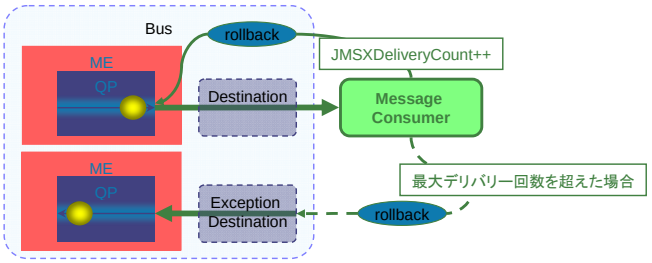
"word LIKE 'l_se' : "lose"はtrue、"loose"はfalse

"pop_name IS NULL"

Designing Web System Infrastructure with WAS V8.0

例外宛先

- 一般的な受信アプリケーションは、メッセージの処理に失敗した場合、トランザクションをrollbackします。これにより、メッセージは元の宛先に戻されます。
- MEでは、宛先に最大デリバリー失敗数を設定可能です。
  - JMSXDeliveryCountが最大デリバリー失敗数に満たない場合、トランザクションをrollbackすることで、メッセージは宛先に戻ります（バックアウト）。
  - JMSXDeliveryCountが最大デリバリー失敗数を超えた場合、トランザクションをrollbackすると、メッセージを例外宛先に退避します。
- バックアウトされてしまうようなメッセージを、ポイズン・メッセージと呼びます。
  - ポイズン・メッセージが他のメッセージ処理の邪魔をしない為に、JMSXDeliveryCountが最大デリバリー失敗数を超えると、メッセージを例外宛先にバックアウトします。



79

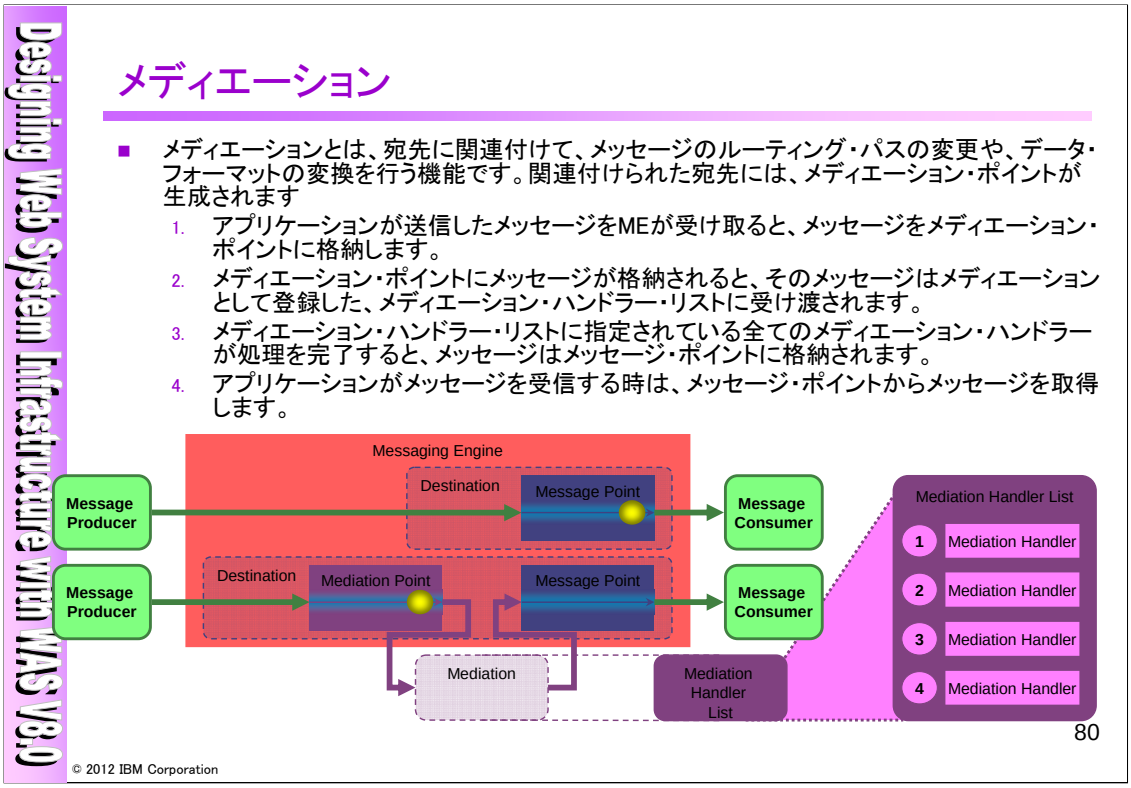
（注意） JMSXDeliveryCountは、メッセージ・ヘッダーに含まれる値です。

© 2012 IBM Corporation

アプリケーションは、メッセージ処理中に問題が発生すると、多くの場合トランザクションをrollbackします。トランザクションがrollbackされると、そのトランザクション内で取得したメッセージは、元の宛先に戻されます。この時、そのメッセージのJMSXDeliveryCountが1足されます。宛先に戻されたメッセージは、再度処理されますが、何回繰り返してもうまく処理できないメッセージが存在します。このようなメッセージを、ポイズン・メッセージと呼びます。

ポイズン・メッセージが他のメッセージの邪魔をしない為に、JMSXDeliveryCountが最大デリバリー失敗数を超えると、メッセージを例外宛先にバックアウトします。この最大デリバリー失敗数と、例外宛先は、宛先に対して設定できます。例外宛先のデフォルトは、システム例外宛先です。システム例外宛先とは、バス・メンバーを登録したときに、自動的に作成される宛先です。

例外宛先にたまったメッセージの処理方法は、アプリケーション設計の段階で決めることをおすすめします。



メディエーションは、宛先に関連付けることで、その宛先に届いたメッセージのルーティング・パスを変更したり、メッセージの中身を操作する機能です。

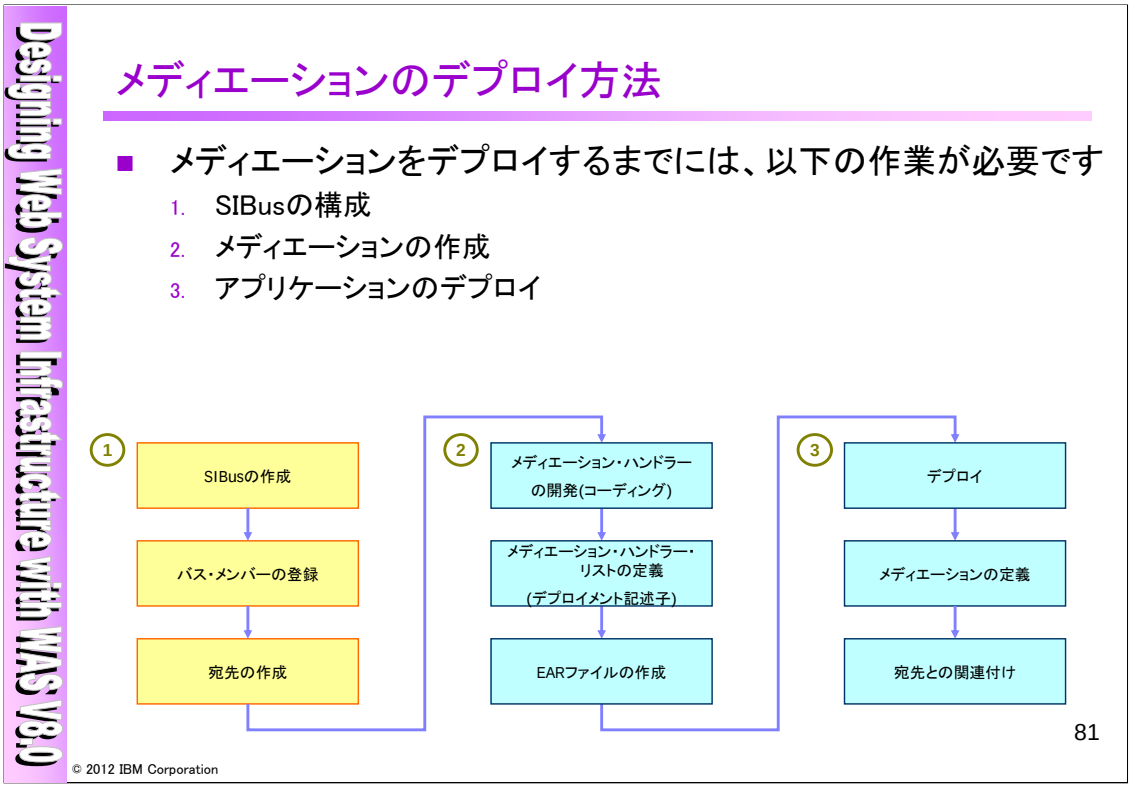
メディエーションが関連付けられた宛先にメッセージを送信すると、メディエーション・ポイントにメッセージが格納されます。通常の受信アプリケーションは、メディエーション・ポイントにあるメッセージは取得できません。メディエーションのみが取得できます。メディエーションに関連付けられたメディエーション・ハンドラー・リストが呼び出され、メディエーション・ハンドラーが一つずつ呼び出されます。全ての処理が成功すると、メッセージはメッセージ・ポイントに格納されます。ここではじめて、受信アプリケーションがメッセージを取得できます。

WAS V8 Information Center: メッセージ・ヘッダーの使用

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjy1502.html

WAS V8 Information Center: メッセージ・ペイロードの使用

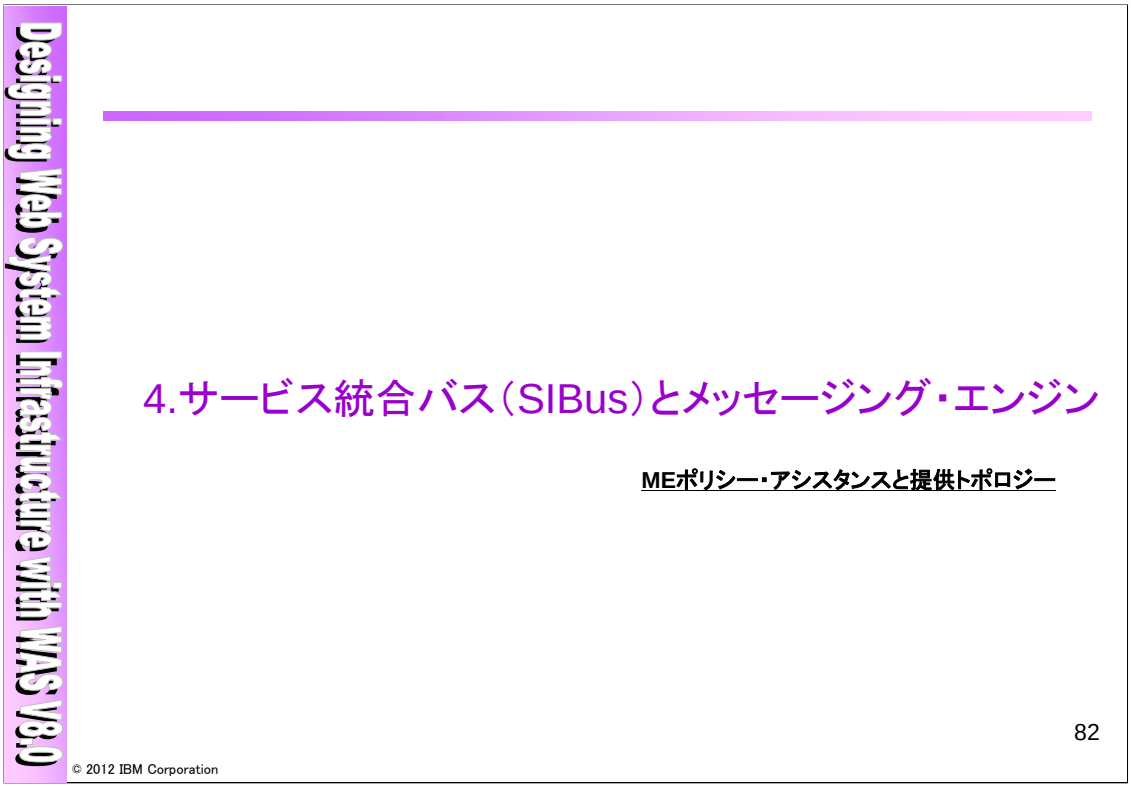
http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjy1501.html



メディエーションをサーバーで稼動させるためには、SIBusの構成、JMSリソースや、アプリケーションの開発/デプロイ、宛先とのを行う必要があります。

メディエーション・ハンドラーは、com.ibm.websphere.sib.mediation.handler.MediationHandlerインターフェイスを実装します。

WASのメディエーションは、全て手でコーディングする必要がありますが、WebSphere ESBにおけるメディエーションは、より充実した機能をGUIを使って簡単に作成できます。



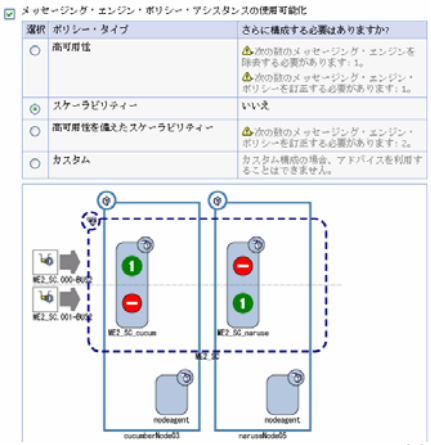
この節では、SIBusとメッセージング・エンジンについて解説します。

Designing Web System Infrastructure with WAS V8.0

メッセージング・エンジン・ポリシー・アシスタンス

V7～

- メッセージング・エンジンの可用性/スケーラビリティー/パフォーマンスを向上
- ポリシー・アシスタンスとは
  - SIBusにバス・メンバーとしてクラスターを登録するときに呼ばれ、以下の機能を提供します
    - コア・グループ・ポリシーの自動設定
    - タイプに従って、アドバイスを提供する
    - ME環境をグラフィカルに表示する
- ポリシー・タイプは4種類を提供
  - 高可用性ポリシー
  - スケーラビリティー・ポリシー
  - 高可用性を備えたスケーラビリティー ポリシー
  - カスタム・ポリシー



83

© 2012 IBM Corporation

MEポリシー・アシスタンスは、WAS V7.0で追加された機能です。

これはSIBusにバス・メンバーとしてクラスターを登録するときに呼ばれ、コア・グループ・ポリシーの自動設定や、設定に必要なアドバイスを提供したり、ME環境をグラフィカルに表示するなどの機能を持っています。

ポリシー・タイプとして4種類のポリシーを提供していて、それぞれのトポロジーを効率よく構成することが可能です。

以下のページで、これらのトポロジーについて解説していきます。

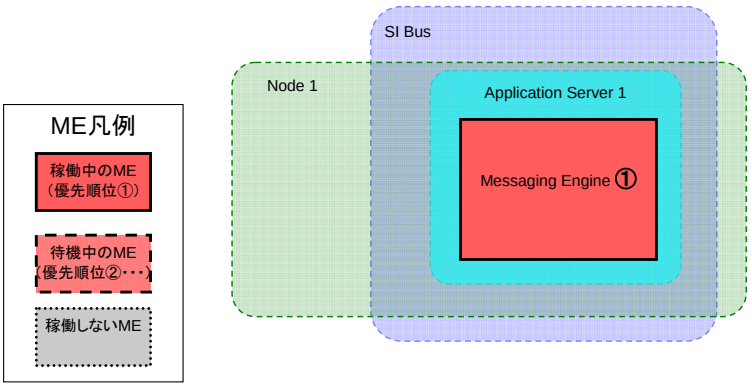
（注意）ポリシー・タイプの一つ「カスタム・ポリシー」は、ポリシー設定を手作業で行います。このポリシー・タイプによるトポロジーは、この資料では扱いません。

Designing Web System Infrastructure with WAS V8.0

単体構成のトポロジー

- 単体構成
  - 1つのアプリケーション・サーバーのみをバス・メンバーとして持つ状態
  - 1つのMEのみしか稼働しないので高可用性やスケーラビリティーは持たない
  - 1つのMEのみしか稼働しないのでSPOF（Single Point of Failure）となりうる

（注意）メッセージング・エンジン・ポリシー・アシスタンスの対象外



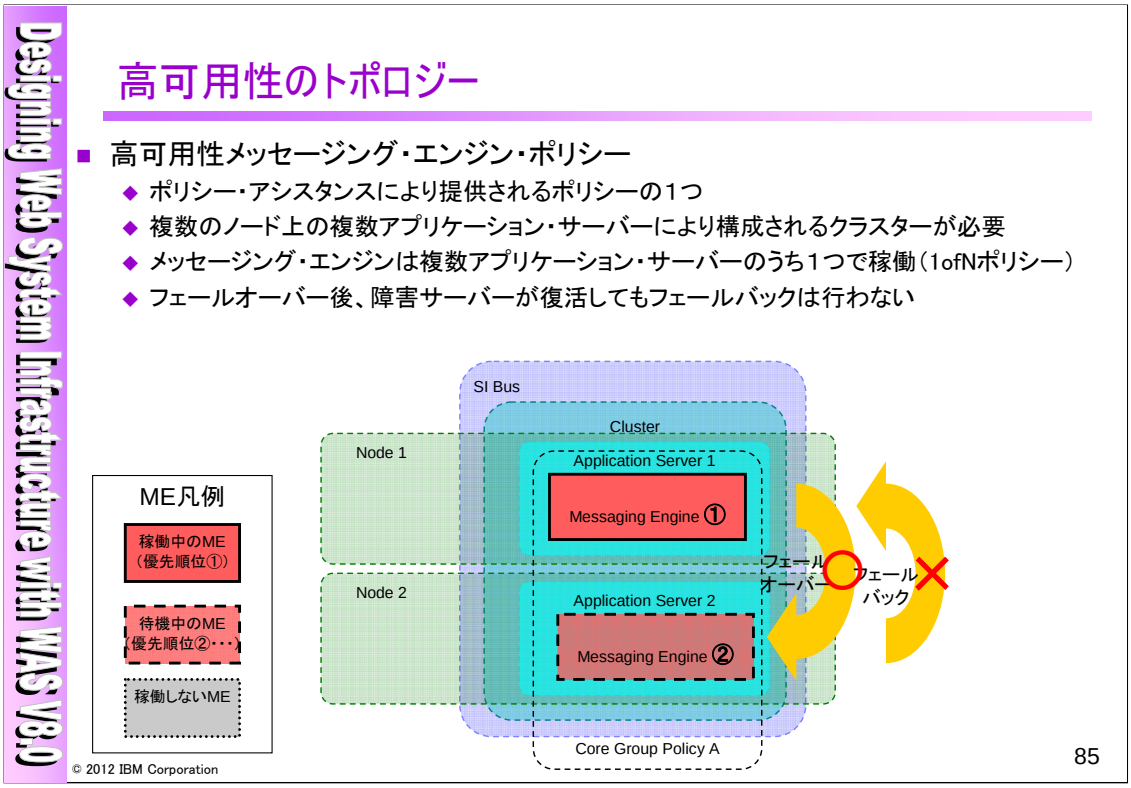
© 2012 IBM Corporation

84

このページでは、単体構成のトポロジーについて説明します。

これは1つのアプリケーション・サーバーのみをバス・メンバーとして登録した状態で、1つのMEのみしか存在しません。したがって、高可用性やスケーラビリティーは持ちません。SPOF (Single Point of Failure) となり得ます。

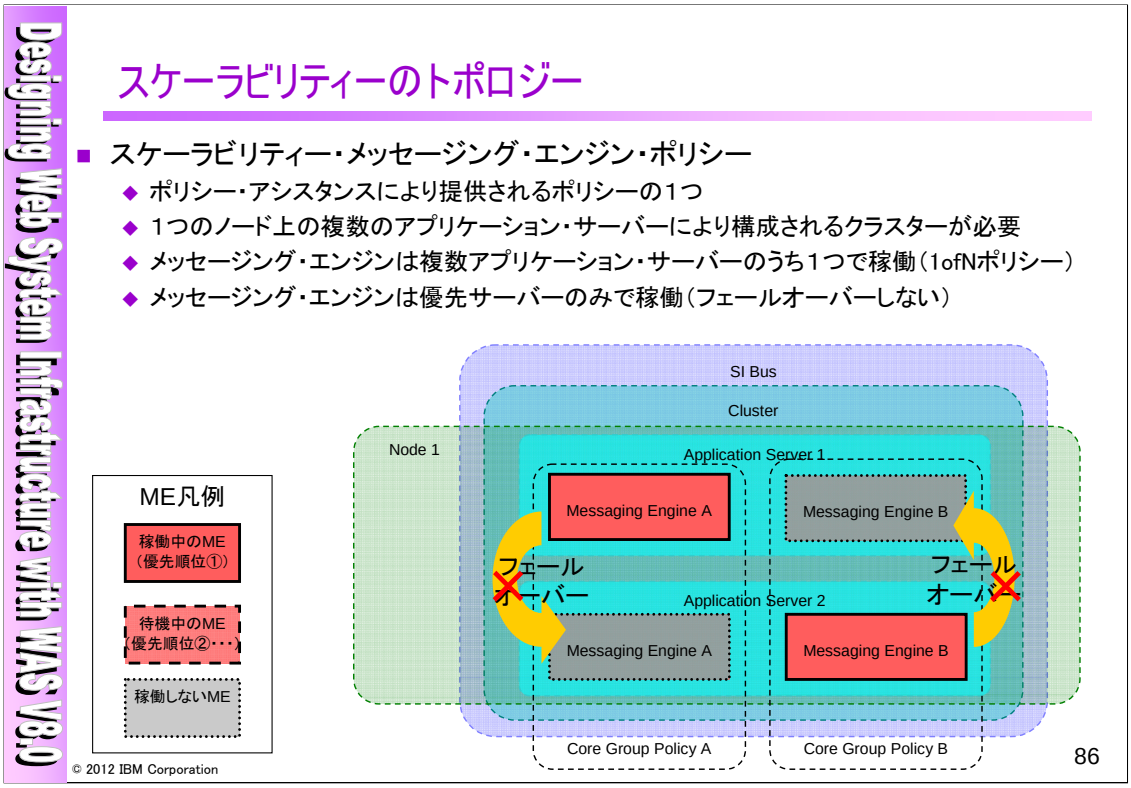
この場合にはメッセージング・エンジン・ポリシー・アシスタンスの対象外です。



このページでは、高可用性のトポロジーについて説明します。

メッセージング・エンジン・ポリシー・アシスタンスで提供する選択肢のひとつに「高可用性メッセージング・エンジン・ポリシー」があります。

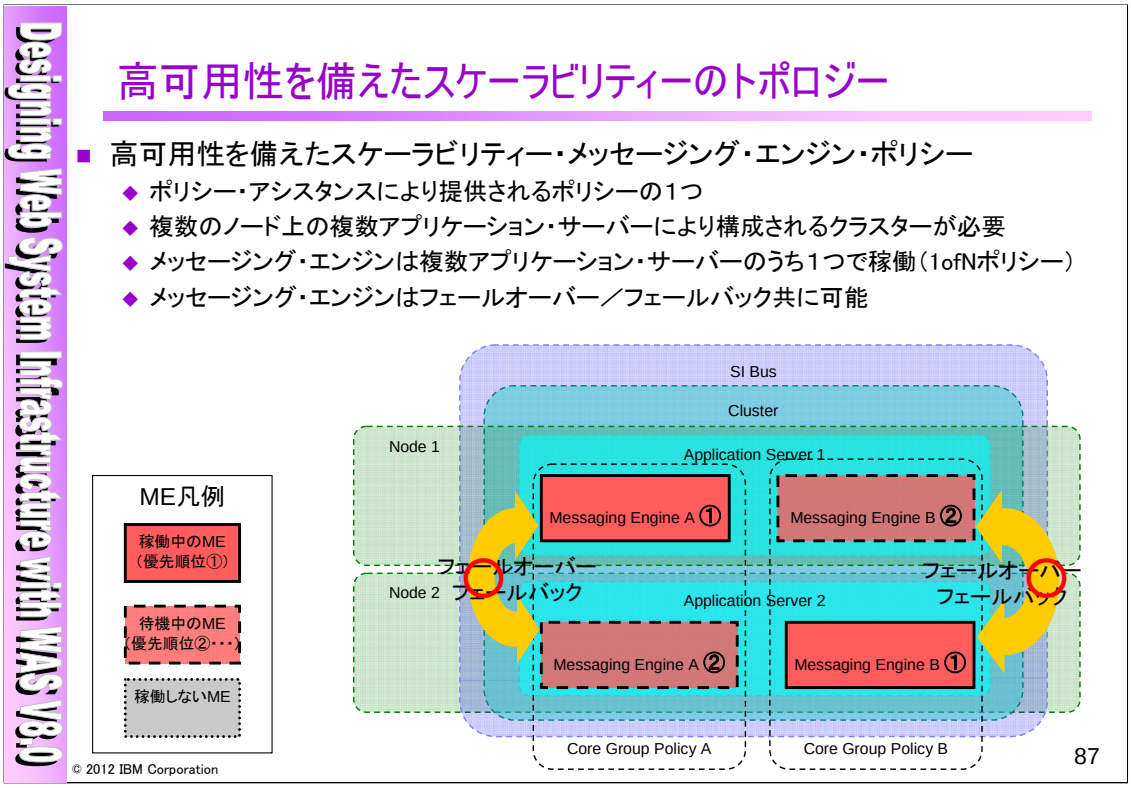
このポリシーで作成されるトポロジーを構成するには、複数のノード上の複数アプリケーション・サーバーにより構成されるクラスターが必要です。メッセージング・エンジンは複数アプリケーション・サーバーのうち1つで稼働（1ofNポリシー）します。フェール・オーバー後、障害サーバーが復活してもフェール・バックは行われません。



このページでは、スケーラビリティーのトポロジーについて説明します。

メッセージング・エンジン・ポリシー・アシスタンスで提供する選択肢のひとつに「スケーラビリティー・メッセージング・エンジン・ポリシー」があります。

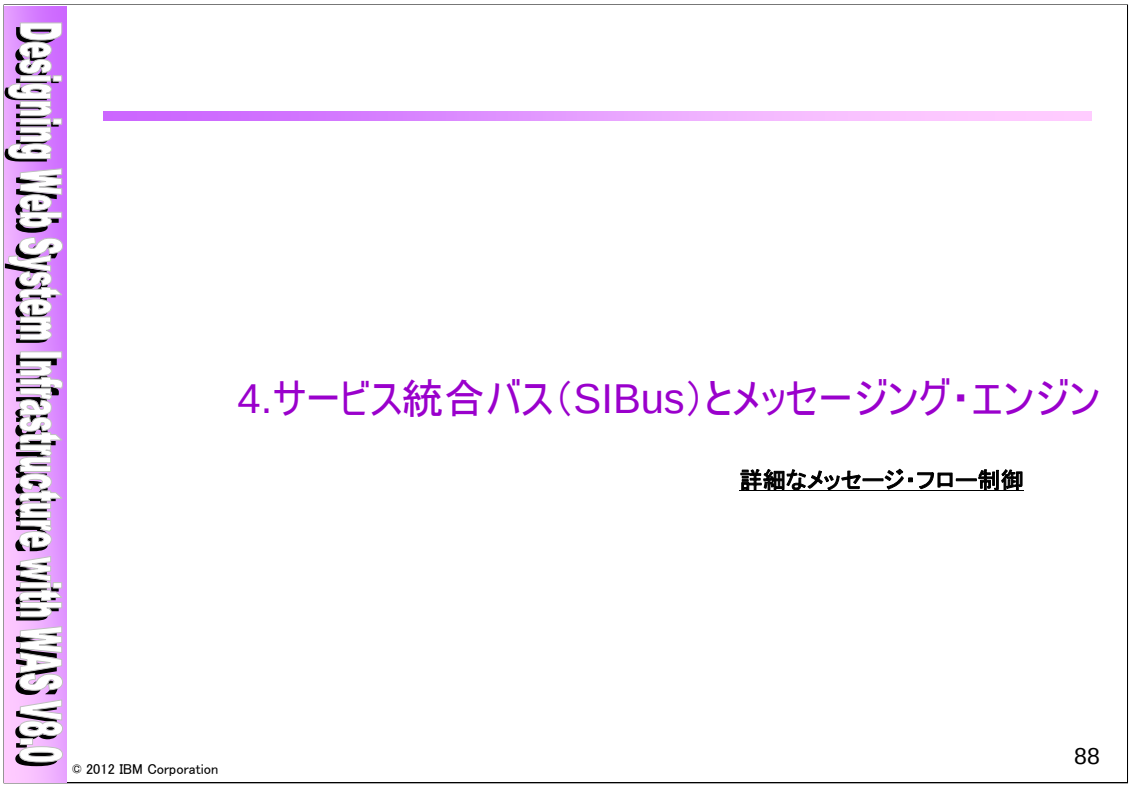
このポリシーで作成されるトポロジーを構成するには、1つ以上のノード上の複数アプリケーション・サーバーにより構成されるクラスターが必要です。メッセージング・エンジンは複数アプリケーション・サーバーのうち1つで稼働（1ofNポリシー）します。メッセージング・エンジンは優先サーバーのみで稼働し、フェール・オーバーはしません。



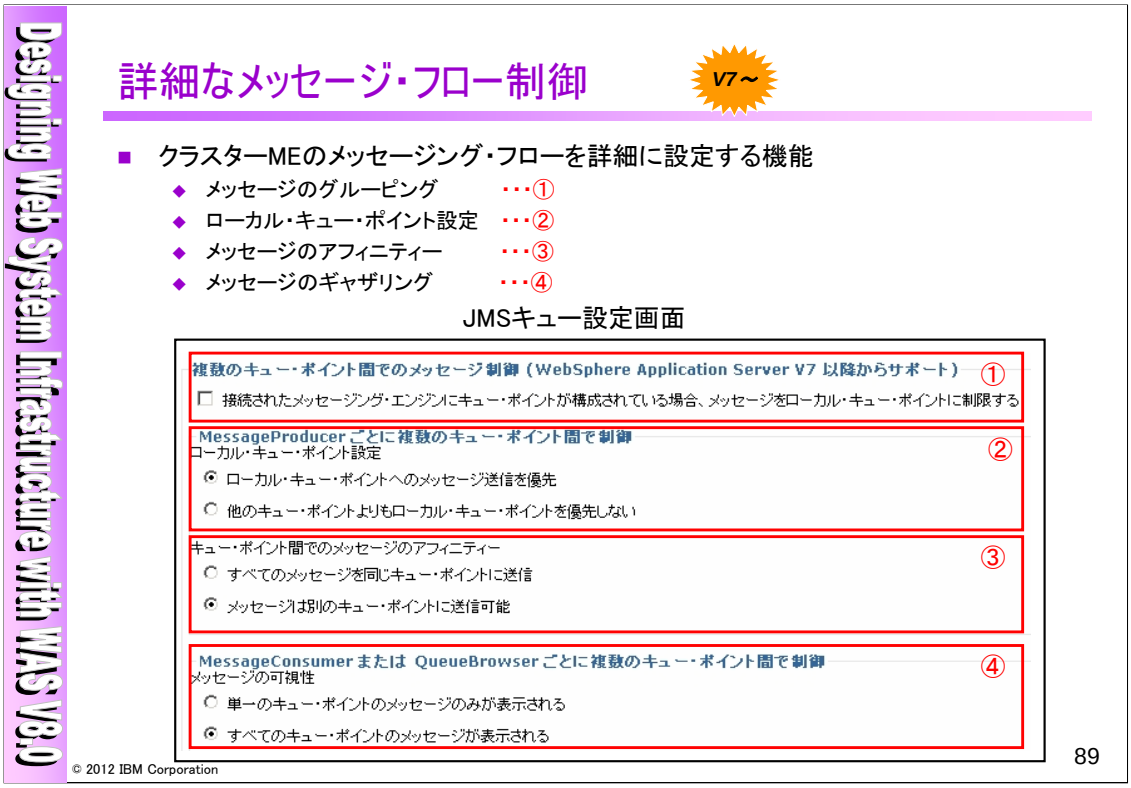
このページでは、高可用性を備えたスケーラビリティーのトポロジーについて説明します。

メッセージング・エンジン・ポリシー・アシスタンスで提供する選択肢のひとつに「高可用性を備えたスケーラビリティー・メッセージング・エンジン・ポリシー」があります。

このポリシーで作成されるトポロジーを構成するには、複数のノード上の複数アプリケーション・サーバーにより構成されるクラスターが必要です。メッセージング・エンジンは複数アプリケーション・サーバーのうち1つで稼働（1ofNポリシー）しますが、メッセージング・エンジンが複数セット稼働していますので、スケーラビリティーが実現されています。メッセージング・エンジンはフェール・オーバー／フェール・バック共に可能です。

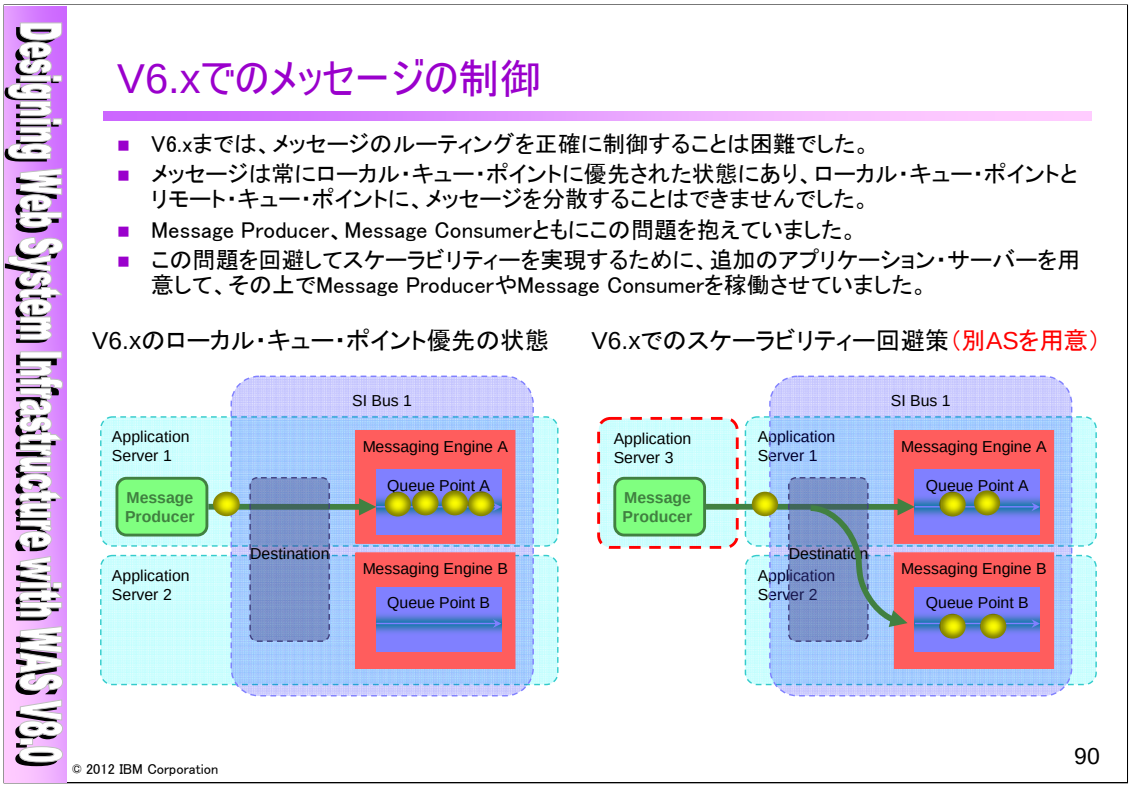


この節では、SIBusとメッセージング・エンジンについて解説します。



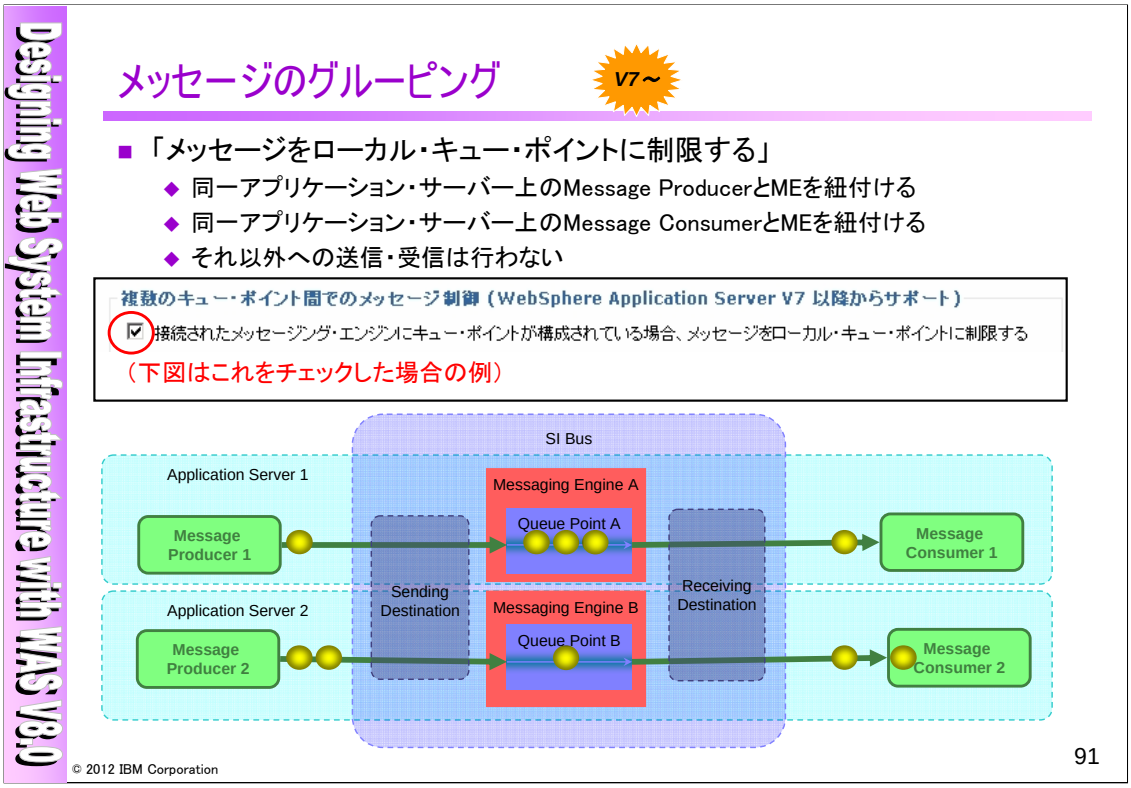
V7からの新機能として、JMSキューの設定により、メッセージ・フロー制御が細かく行えるようになりました。

①～④のそれぞれについて、次ページ以降で詳しく説明していきます。



V6.xまでは、メッセージのルーティングを正確に制御することは困難でした。

メッセージは常にローカル・キュー・ポイントに優先された状態にあり、ローカル・キュー・ポイントとリモート・キュー・ポイントに、メッセージを分散することはできませんでした。Message Producer、Message Consumerともにこの問題を抱えていました。このため、この問題を回避してスケーラビリティーを実現するために、追加のアプリケーション・サーバーを用意して、その上でMessage ProducerやMessage Consumerを稼働させる必要がありました。

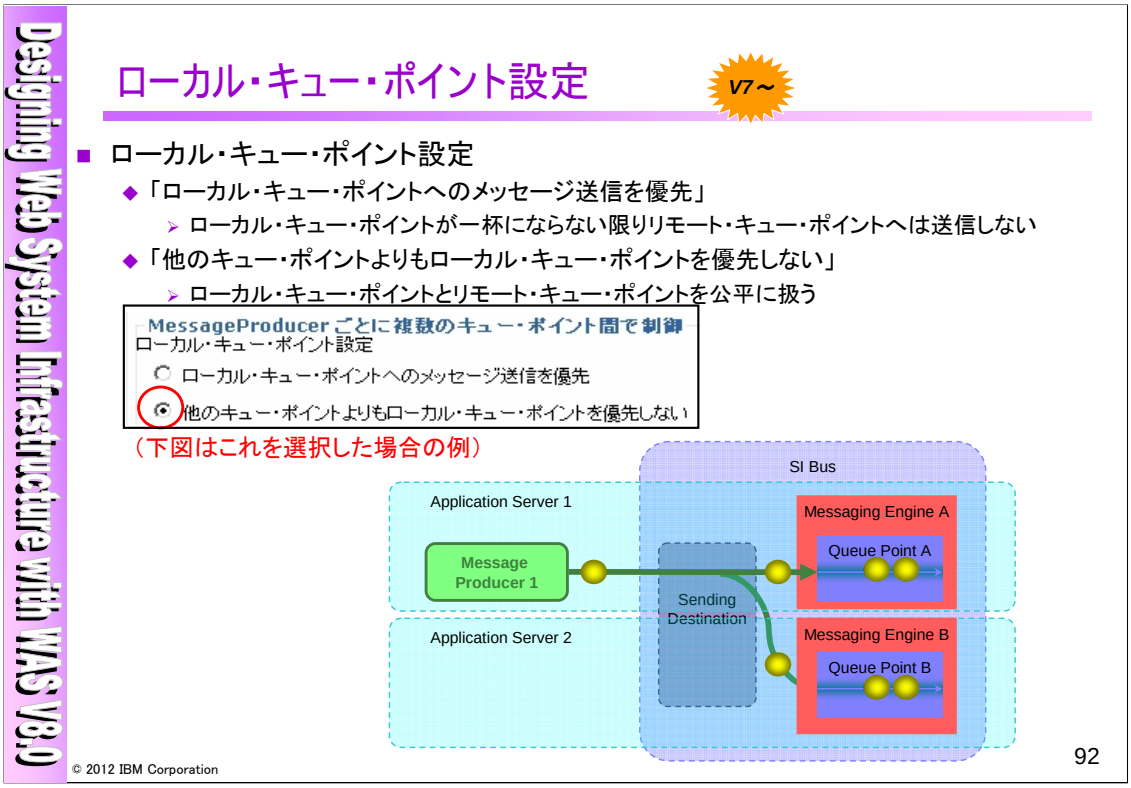


メッセージのグルーピングとは、同一アプリケーション・サーバー上（＊1）のMessage ProducerとMEを紐付けたり、Message ConsumerとMEを紐付けたりすることです。

「接続されたメッセージング・エンジンにキュー・ポイントが構成されている場合、メッセージをローカル・キュー・ポイントに制限する」のチェック・ボックスをチェックすることで、この機能を利用することができます。

上図のMessage Producer 1とMessage Producer 2はそれぞれ、5個ずつのメッセージ（黄色い丸）を送信しています。もしこの機能がオンになっていなければ、Messaging Engine Aのキュー・ポイントとMessaging Engine Bのキュー・ポイントのどちらに、メッセージがキューイングされるかは指定できません。（詳細な設定は、次のページのローカル・キュー・ポイントで設定します。）この例では、Message Producer 1の5個のメッセージがMessage Consumer 1で受信され、同様に、Message Producer 2の5個のメッセージがMessage Producer 2によって受信される様子を示しています。

（＊1）「メッセージをローカル・キュー・ポイントに制限する」機能では、Message Producer やMessage Consumerが、MEと同一アプリケーション・サーバー上である場合だけではなく、接続されたMEのキュー・ポイントにメッセージをキューイングするよう指定することができます。



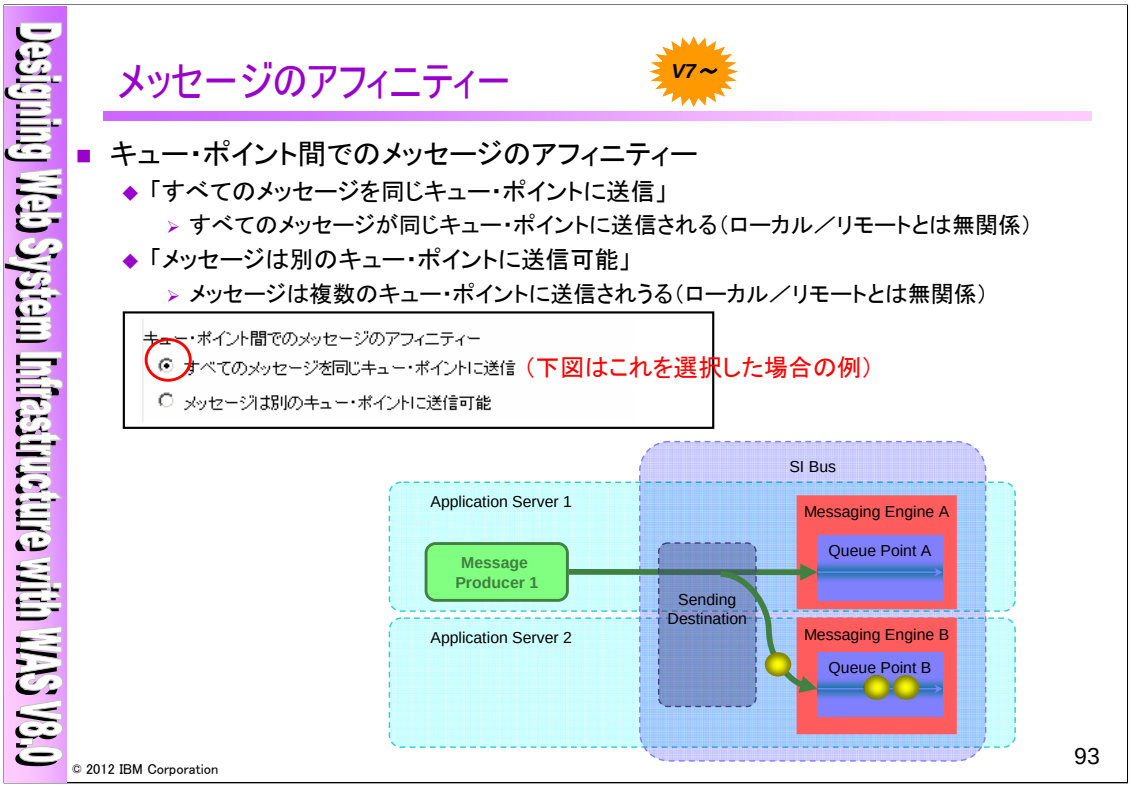
このページでは、ローカル・キュー・ポイントを優先するか、ローカルとリモートのキュー・ポイントを公平に扱うか、を選択する「ローカル・キュー・ポイント設定」について説明します。

前のページで説明した、「メッセージングのグルーピング」設定をオフ（デフォルト）にして、「ローカル・キュー・ポイント設定」で「他のキュー・ポイントよりもローカル・キュー・ポイントを優先しない」を選択すると、ローカルとリモートのキュー・ポイントを公平に扱うことが可能になります。

図中のMessage Producer 1が送信したメッセージは、ローカル・キュー・ポイントとリモート・キュー・ポイントに公平に振り分けられます。（＊1）

逆に、この設定にしない場合、つまり「ローカル・キュー・ポイントへのメッセージ送信を優先」を選択した場合には、ローカル・キュー・ポイントが一杯になるまで、リモート・キュー・ポイントには、キューイングされません。

（＊1）ここでの「公平は」、（あるMessage Producerが送信する複数のメッセージの）一つ目のメッセージが、ローカル・キュー・ポイントに優先的に振り分けられる訳ではない、という意味です。二つ目以降のメッセージの振り分けについては、次のページで説明します。

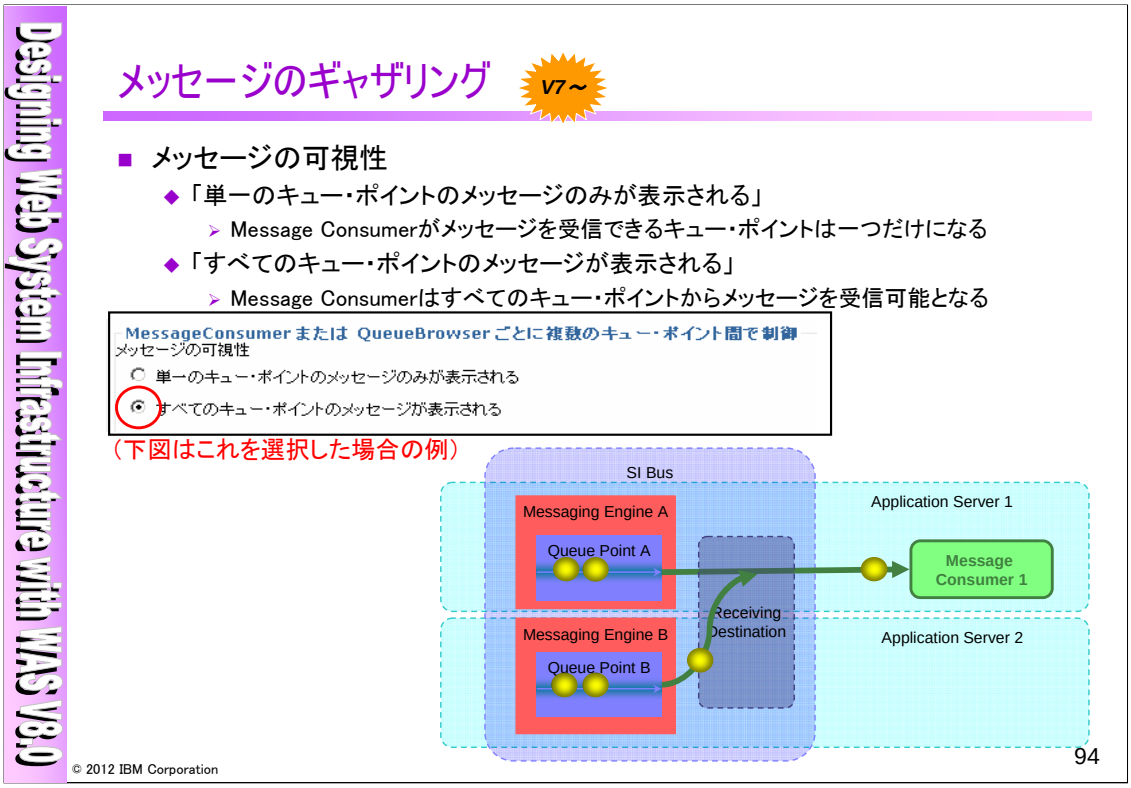


このページでは、「キュー・ポイント間でのメッセージのアフィニティー」設定について、説明します。

この設定は、Message Producerが複数のメッセージを送信するときに、すべてのメッセージを同じキュー・ポイントに送信するか、それとも、複数のキュー・ポイントに分けて送信するかを選択します。

前のページで説明した「ローカル・キュー・ポイント設定」とは異なり、キュー・ポイントが、ローカルかリモートかということとは関係がありません。「すべてのメッセージを同じキュー・ポイントに送信」が選択されていると、Message Producerは一つ目のメッセージが送信されたキュー・ポイントに、後続の複数のメッセージを送信します。「メッセージは別のキュー・ポイントに送信可能」の場合には、同じキュー・ポイントにはこだわりません。また、前のページで説明した「ローカル・キュー・ポイント設定」で、「ローカル・キュー・ポイントへのメッセージ送信を優先」を選択している場合には、（ローカル・キュー・ポイントが利用可能である限り）、ラウンド・ロビン方式の振り分けは行われません。

（注意）アフィニティーの範囲は、1つのMessage Producerに対応します。JMS1.1インターフェースでは、session（1つのトランザクションに対応）のsession#createProducer(destination)メソッドにより、MessageProducerオブジェクトが作成される点に注意してください。

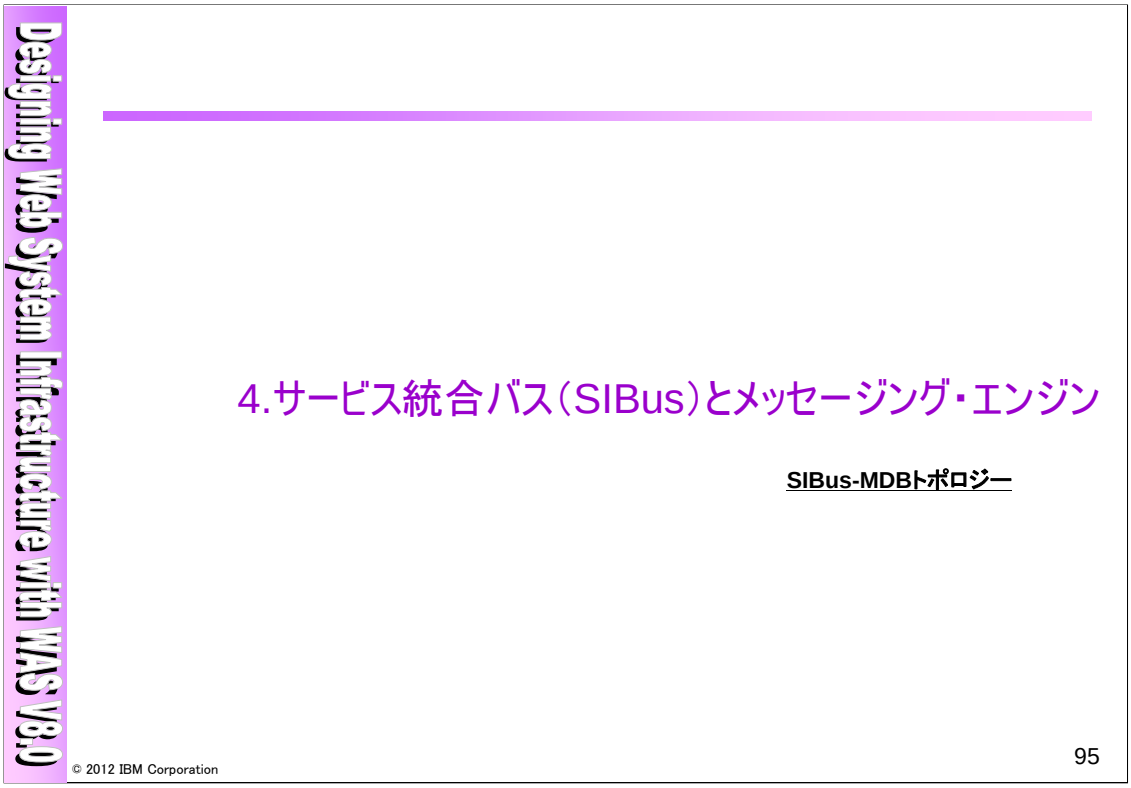


このページでは、「メッセージの可視性」設定について、説明します。

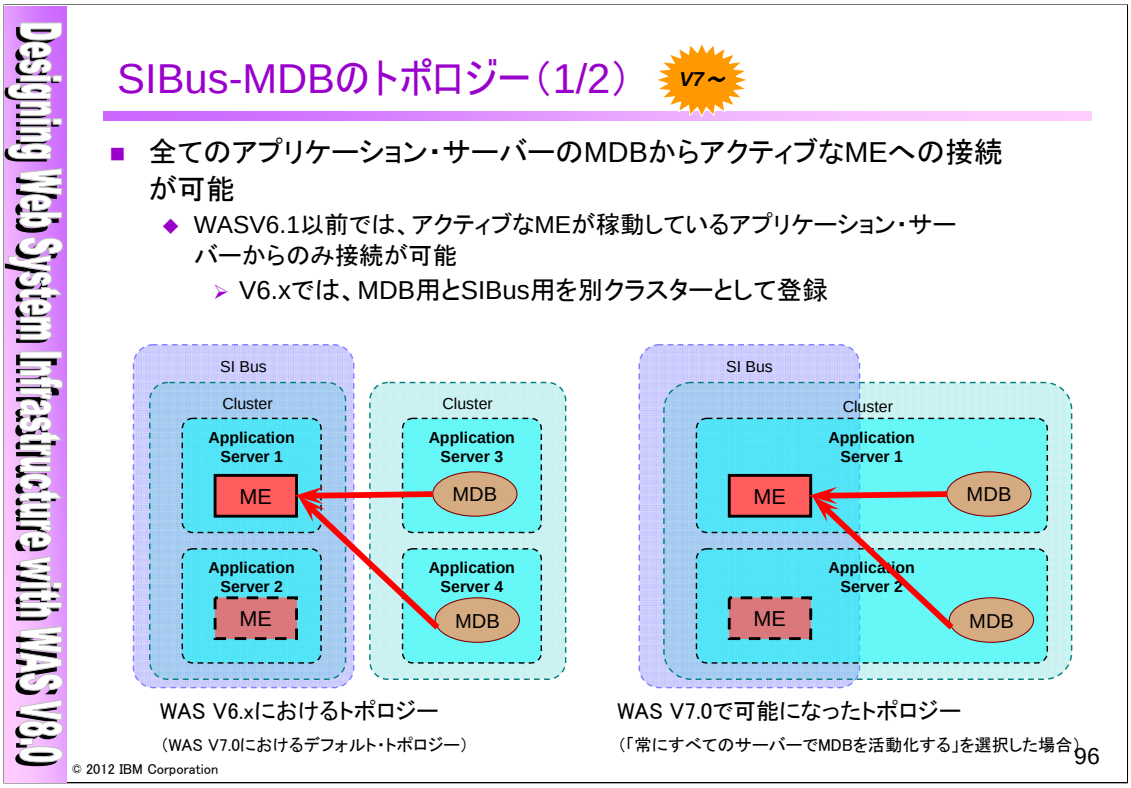
この設定は、Message Producerがメッセージを受信する際に、複数のキュー・ポイントからどの様に受信するかを指定します。

「単一のキュー・ポイントのメッセージのみが表示される」を選択した場合には、初めに接続したMEに構成されているキュー・ポイントのみから、受信を行います。

「すべてのキュー・ポイントのメッセージが表示される」を選択した場合には、受信宛先に関連付けられているすべてのキュー・ポイントから、受信を行います。
上図では、この場合を示しています。Message Consumer 1はローカル・キュー・ポイントからだけでなく、リモート・キュー・ポイントからもメッセージを受信しています。



この節では、SIBusとメッセージング・エンジンについて解説します。



WASの以前のバージョンでは、アプリケーション・サーバー1上のME1がアクティブな場合、アプリケーション・サーバー2からME1に接続することが出来ませんでした。この制約により、MDBが稼動しているアプリケーション・サーバーとMEが稼動しているアプリケーション・サーバーを別クラスターに構成するというような対応が必要でした。WAS V7.0ではこの制約がなくなり、全てのアプリケーション・サーバーのMDB (Message Driven Bean)からのアクティブなMEへの接続が可能になりました。この構成とする場合には、管理コンソールより、「リソース > JMS > アクティベーション・スペック」を選択し、“常に

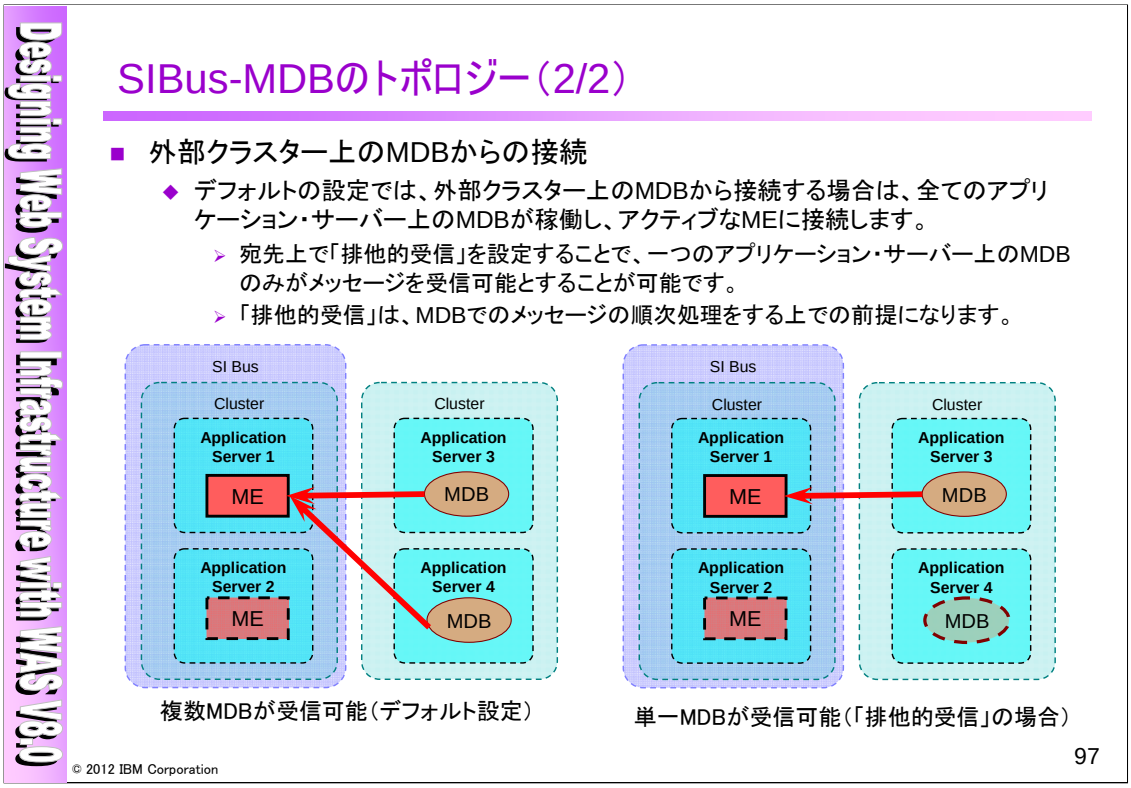
すべてのサーバーでMDBを活動化する”のチェックボックスをONにして下さい。(デフォルト値はOFFです。)

WAS V8 Information Center: JMS アクティベーション・スペック [設定]

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/SIBJMSActivationSpec_DetailForm.html

WAS V8 Information Center: メッセージ・ドリブン Bean がクラスター内で接続する方法

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cjn_mdb_endpt_overview.html



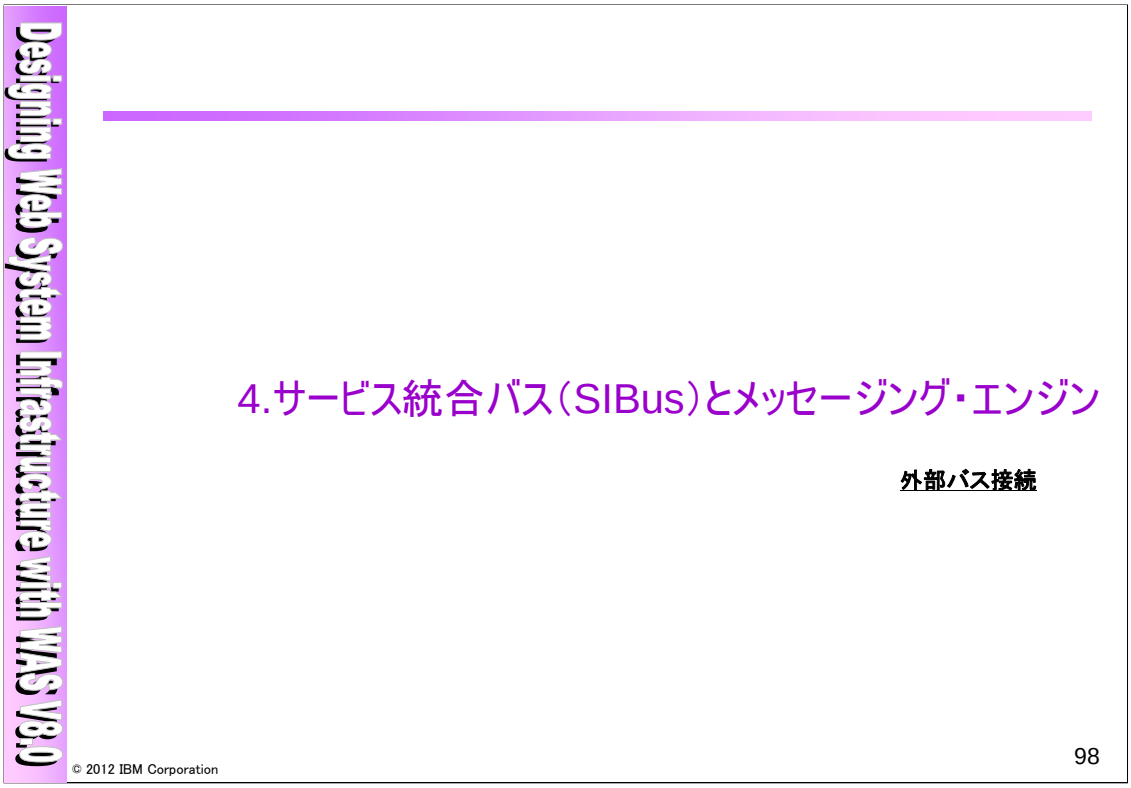
メッセージング・エンジンが稼働するアプリケーション・サーバー・クラスターとは別のクラスター上で稼働するMDBの受信方法を設定する場合の注意点です。

デフォルトの設定では、複数のアプリケーション・サーバーで稼働する複数のMDBがアクティブなMEに接続しますので、メッセージは複数のMDB上で処理されることになります。

単一のMDBでメッセージの処理を行いたい場合には、宛先の設定で「排他的受信」を有効にして、一つのMDBのみがメッセージを受信するように設定します。これは、MDBでメッセージの順次処理を行う場合の前提となる設定です。確実に順次処理を行うための方法については、以下のリンク先「メッセージ配列」を参照してください。

WAS V8 Information Center: メッセージの順序付け

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cjj8000_.html

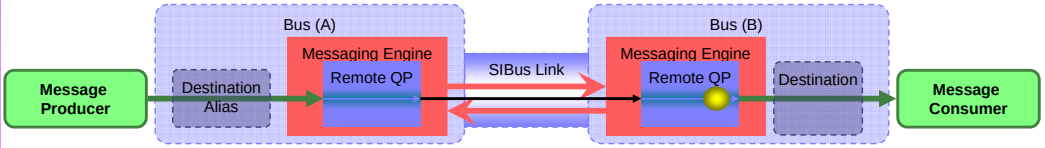


この節では、SIBusとメッセージング・エンジンについて解説します。

Designing Web System Infrastructure with WAS V8.0

外部バス接続（SIBusリンク）

- メッセージを他のバスに送信する為に、バス同士をリンクさせる機能が提供されています
  - バス同士をリンクさせるには、バスに対して、相互に外部バスを定義します
- 外部バス接続には2種類存在します
  - SIBus
  - WebSphere MQ（MQ）のキュー・マネージャー（Qmgr）
- SIBusリンク
  - 他のバスを外部バスとして登録できます。
  - ME同士を接続することで、バス間におけるメッセージのやりとりを実現しています。このME同士の接続を、SIBusリンクと呼びます。
  - 外部バスの宛先にメッセージを送信する場合、バスに別名宛先を作成して、そこにメッセージを送信することで、MEリンク経由で別バスにメッセージが送信されます。

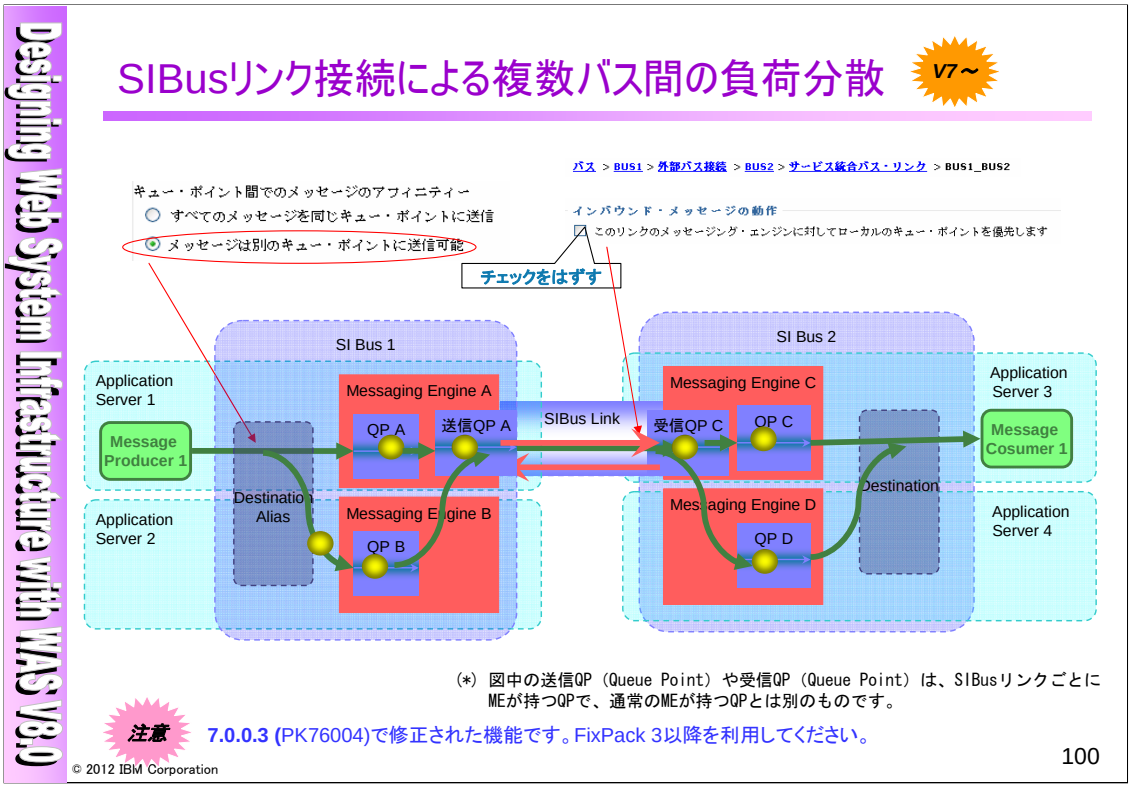


© 2012 IBM Corporation

99

同一バス内では、メッセージング・エンジン同士が連携して、メッセージのやり取りが可能なことは前述の通りです。SIBusでは、別のバスともメッセージのやり取りが可能です。外部バスとして利用可能なのは、SIBusと、MQのQmgrです。SIBus同士の接続をSIBusリンク、SIBus-Qmgr接続をMQリンクと呼びます。

SIBusリンクは、それぞれのバス内にある特定のME同士の接続です。バス間のメッセージのやり取りは、全てのMEを介して行われます。外部バスの宛先にメッセージを送信するためには、バスに別名宛先を定義する必要があります。



WAS V7.0より、SIBusリンク接続による複数バス間の負荷分散が可能となりました。

この構成を行うためには、別名宛先キューで、（前述の）「キュー・ポイント間でのメッセージのアフィニティー」で「メッセージは別のキュー・ポイントに送信可能」を設定します。さらに、SIBusリンクの「インバウンド・メッセージの動作」において、「このリンクのメッセージング・エンジンに対してローカルのキュー・ポイントを優先します」のチェックをオフにします。（デフォルトはオンです。）

これにより送信側SIBusでメッセージを分散させるだけでなく、受信側でもメッセージを分散させることができるようになります。

この機能は、Fipack3（PK76004）で修正されていますので、V7.0.0.3以降を利用してください。

（注意）図中の送信QP（Queue Point）や受信QP（Queue Point）は、SIBusリンクごとにMEが持つQPで、通常のMEが持つQPとは別のものです。

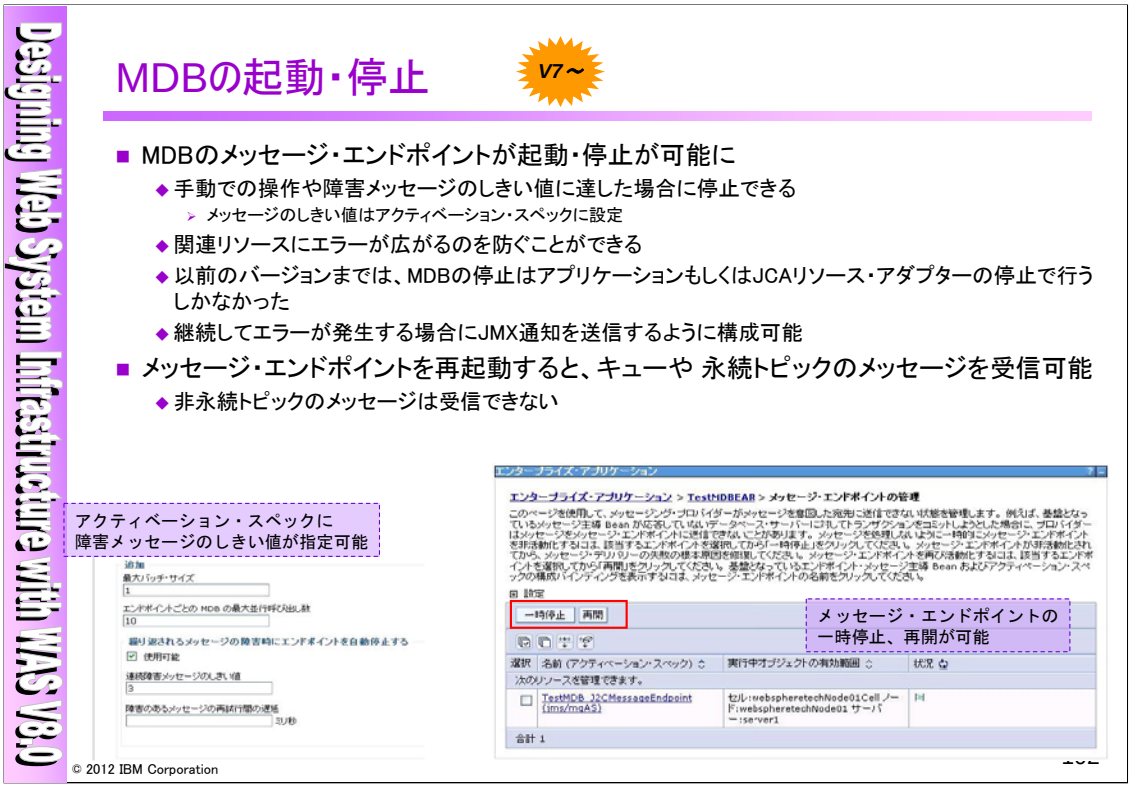
Designing Web System Infrastructure with WAS V8.0

4.サービス統合バス（SIBus）とメッセージング・エンジン

SIBus／MEの運用方法

© 2012 IBM Corporation

101



以前のバージョンでは、MDBを停止させるには、アプリケーション自身の停止、もしくはリソース・アダプターを停止させる必要がありました。WAS V7からメッセージ・エンドポイントを停止することができるようになりました。管理コンソールから、一時停止、再開が可能です。

エンタープライズ・アプリケーション > MDB名 > ランタイム・タブ > メッセージ・エンドポイントの管理からメッセージ・エンドポイントの起動・停止ができます。

アクティベーション・スペックに、障害メッセージのしきい値を設定することが可能です。障害メッセージのしきい値を超えると、エンドポイントを自動停止することができます。障害メッセージが大量に送信された場合に、例外宛先があふれるのを防いだり、他のリソースへの影響を抑えることができます。

リソース > JMS > アクティベーション・スペック > アクティベーション・スペック名　から障害メッセージのしきい値を設定できます。

WAS V8 Information Center:メッセージ・エンドポイントによるメッセージの管理

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tdat_msgendpoint.html

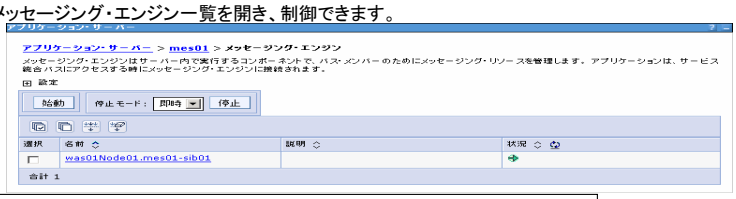
WAS V8 Information Center:システム・リソース問題からの MDB または SCAアプリケーションの保護

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjn_mdb_0001.html

Designing Web System Infrastructure with WAS V8.0

メッセージング・エンジン起動/停止

- メッセージング・エンジンは、管理コンソールかwsadminを使って、起動/停止できます。
  - 管理コンソール
    サービス統合 → バス → *バス名* → メッセージング・エンジン
    を選択して、メッセージング・エンジン一覧を開き、制御できます。



  - wsadmin

```
# 変数MEに、メッセージング・エンジンを設定します
set ME [$AdminControl completeObjectName type=SIBMessagingEngine,name=${MENAME},*]

# 開始します
$AdminControl invoke ${ME} start

# 停止します
$AdminControl invoke ${ME} stop

#即時に停止します
$AdminControl invoke ${ME} stop {"IMMEDIATE"}

# 強制停止します
$AdminControl invoke ${ME} stop {"FORCE"}
```

© 2012 IBM Corporation

MEには、2種類の停止モードがあります。

即時モードでは、MEは停止要求の発生時に実行されていたすべてのメッセージング作業の完了時に停止します。MEの停止を示す通知が各アプリケーションに送信されることはありません。停止コマンドが実行された後は、MEは新しいオペレーションの開始を許可しません。

既存の各接続については、MEは現行オペレーションが終了するのを待ちます。ただし、受信オペレーションのように、ME内でブロックするオペレーションの場合は除きます。この場合、オペレーションは中断されます。非同期コンシューマーの場合、たとえ現行メッセージの処理に任意の時間がかかっても、その完了が許可されます。MEはその後にアクティブなトランザクションを取り消し、その接続に関する以降のオペレーションを禁止します。すべての接続がこの無効化された状態になると、MEは停止します。

強制モードでは、MEは停止されて、すべての現行トランザクションが優先され、アプリケーションは強制的に切断されます。

強制モードは即時モードと似ていますが、停止コマンドが実行されると、MEが停止されることにより、アプリケーション・スレッドで実行中のメッセージング・オペレーションが中断されるという点が異なります。強制モードでは現行メッセージング・オペレーションの完了は許可されず、MEはオペレーションを中断してから、以降のオペレーションを禁止します。すべての接続がこの状態になると、MEが停止します。

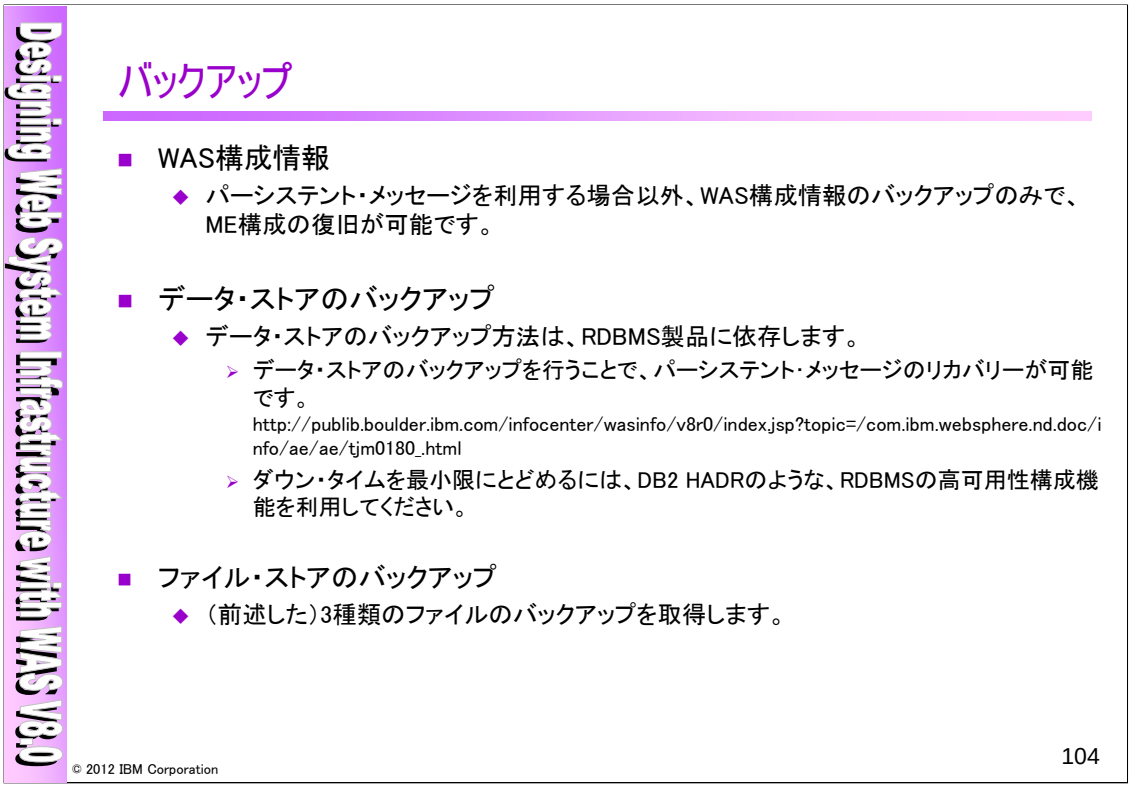
強制モードの目的は、できる限り短時間でMEのシャットダウンを完了することです。MEの以後の再始動時には、即時モードを使用してオペレーションを停止した場合よりも多くのリカバリー・アクションが必要になるため、処理に時間がかかることがあります。例えば、強制モードでの停止処理により、メッセージが未確定のままになる可能性があるため、これらのメッセージを未確定トランザクションの解決の説明に従って処理する必要があります。

stopServerコマンドでMEを停止した場合、即時モードでMEを停止します。

WAS V8 Information Center: メッセージング・エンジンの停止のメカニズム

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cjk0200_.html

なお、WAS V7.0からMDBの起動・停止が可能になりました。



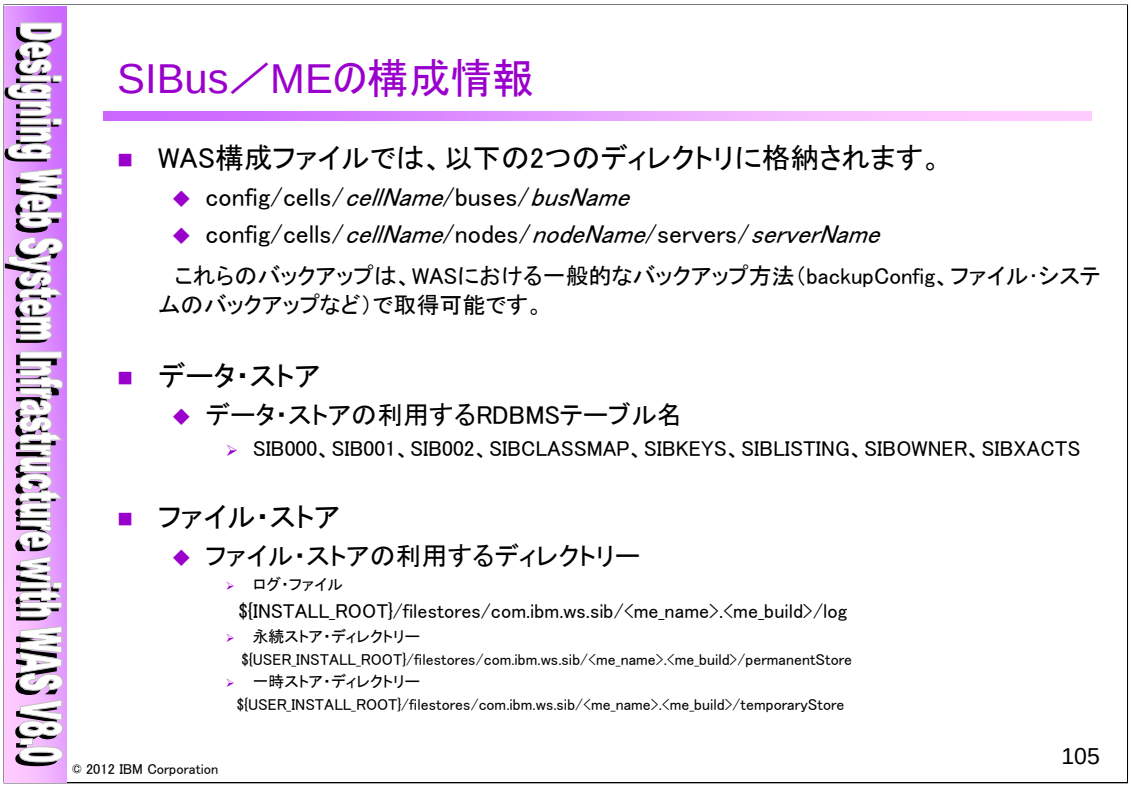
MEの構成情報は、通常のWAS構成情報に含まれますので、backupConfigコマンド等によって取得した通常のバックアップによって障害時の復旧が可能です。ただし、パーシステント・メッセージを利用している場合は、メッセージ・ストアのバックアップも必要です。

データ・ストアを構成するとき、表スペースなどの設定を行うためにはDDLを書き換える必要があります。その為には、sibDDLGeneratorコマンドを利用して、テーブル作成DDLをファイルに出力して、それを編集するのが適切です。また、このDDLのバックアップを取得しておくことで、データ・ストアの復旧に役立てることができます。

データ・ストアの表作成時に、表スペースの指定などを含めたい場合は、sibDDLGeneratorコマンドを利用して、一旦DDLをファイルに出力してから、編集します。復旧時には、このDDLを実行することで、表スペースの指定などについても、復旧できます。sibDDLGenerator.sh -system db2 -version 8.1 -platform aix -schema sib01mes01 -user db2inst1 -statementend “;” -create > sib01mes01crt.ddl

データ・ストアに保存されるメッセージは、揮発性が高い為、データ・ベースのバックアップを取ることはそれ程重要ではありません。データ・ベースの可用性を高める方が効果的です。DB2 HADRのようなRDBMSの高可用性機能を利用して、ダウン・タイムを短縮できます。

ファイル・ストアのバック・アップについては、ファイル・ストアの説明ページを参照してください。

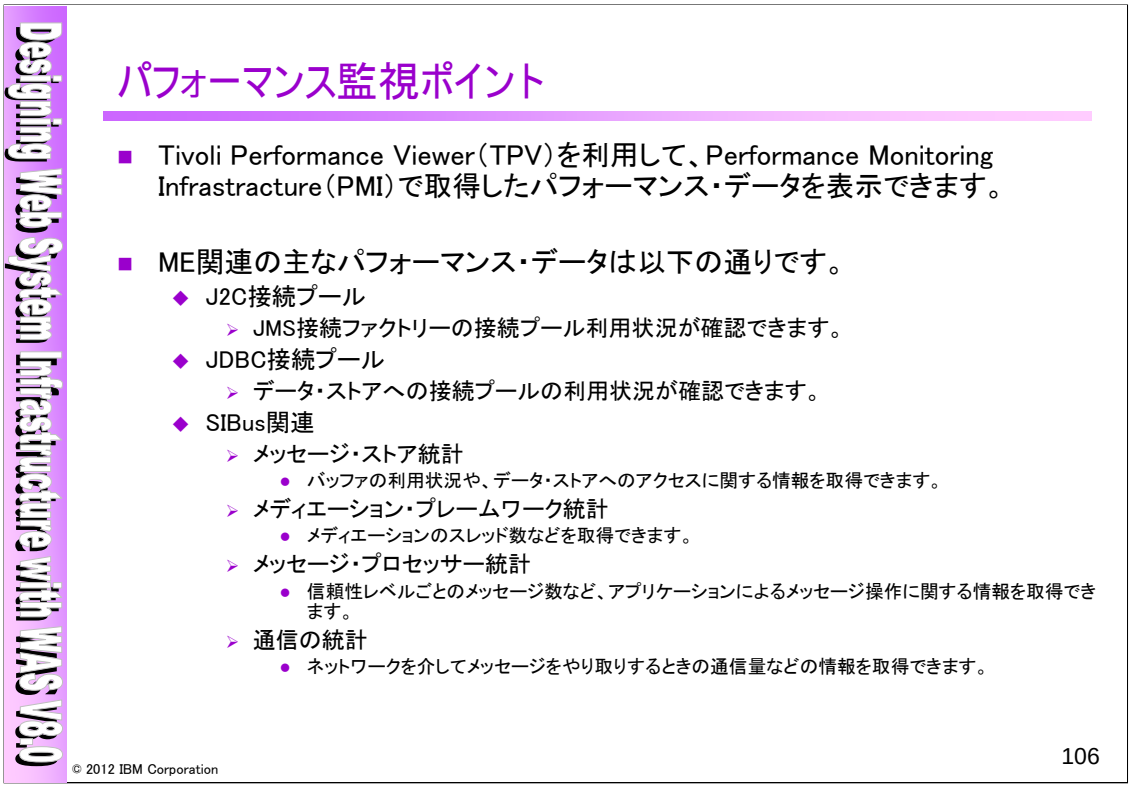


SIBus関連の構成情報は、WASの構成ファイルに含まれます。ファイル名としては以下の通りとなります。

config/cells/*cellName*/buses/*busName*
- sib-authorisations.xml
- sib-bus.xml
- sib-destinations.xml

config/cells/*cellName*/nodes/*nodeName*/servers/*serverName*
- sib-engines.xml
- sib-service.xml

その他に、パーシステント・メッセージを利用している場合は、データ・ストアのバックアップを取ることで、メッセージも復旧可能です。（ファイル・ストアの場合には、永続ストア・ファイルです。）



ME関連のコンポーネントは、TPVでリソースの利用状況を確認できます。

ME関連の主なパフォーマンス・データは以下の通りです。

・J2C接続プール：

JMS接続ファクトリーの接続プール利用状況が確認できます。

・JDBC接続プール：

データ・ストアへの接続プールの利用状況が確認できます。

・SIBus関連：

メッセージ・ストア統計：

バッファの利用状況や、データ・ストアへのアクセスに関する情報を取得できます。

メディエーション・プレームワーク統計：

メディエーションのスレッド数などを取得できます。

メッセージ・プロセッサー統計：

信頼性レベルごとのメッセージ数など、アプリケーションによるメッセージ操作に関する情報を取得できます。

通信の統計：

ネットワークを介してメッセージをやり取りするときの通信量などの情報を取得できます。

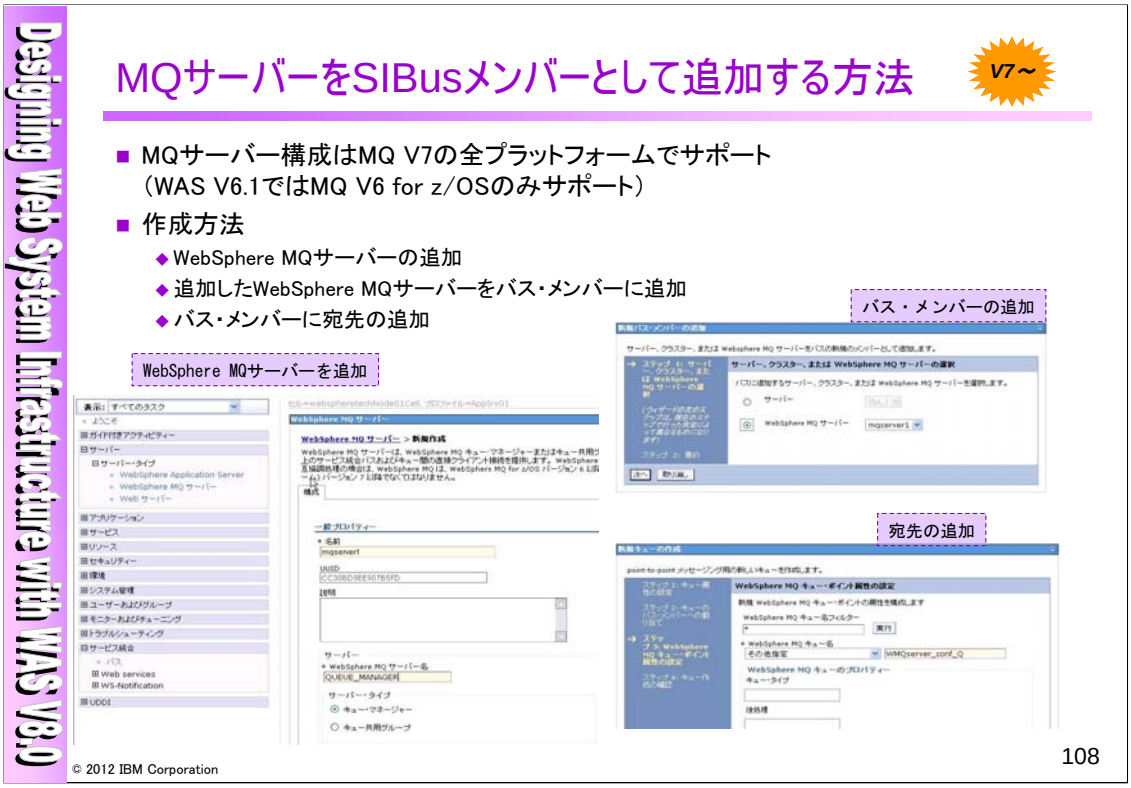
Designing Web System Infrastructure with WAS V8.0

5.WebSphere MQ との接続

MQサーバーをSIBusメンバーとして追加する方法

© 2012 IBM Corporation

107



MQサーバー構成が、MQ V7の全プラットフォームでサポートされました。MQサーバーを、バス・メンバーとして利用できます。

SIBusとキュー・マネージャーやキュー共用グループ(z/OS)のキューと直接クライアント接続が可能になりました。

作成方法は以下の手順で行います。

・WebSphere MQサーバーの追加 （サーバー > サーバー・タイプ > WebSphere MQサーバー から新規作成する）

・追加したWebSphere MQサーバーをバス・メンバーに追加 （サービス統合 > バス > バス名 > トポロジー > バス・メンバー からバス・メンバーに追加する）

・バス・メンバーに宛先の追加 （サービス統合 > バス > バス名 > 宛先 から宛先を作成する）

これまで、外部バス接続の作成と、MQリンクまたはSIBリンクの作成は別々になっており、名前などが同じであるか確認する必要がありました。WAS V7から外部バス接続を作成すると、MQリンクおよびSIBリンクの作成を続けて設定することが可能なため、名前の確認などを行う必要がなくなりました。

サービス統合 > バス > バス名 > 外部バス接続　から外部バス接続を作成します。

外部バス接続を作成後に、MEをリスタートする必要もなくなりました。動的に構成変更が可能です。

Designing Web System Infrastructure with WAS V8.0

ワークショップ、セッション、および資料は、IBMまたはセッション発表者によって準備され、それぞれ独自の見解を反映したものです。それらは情報提供の目的のみで提供されており、いかなる参加者に対しても法律的またはその他の指導や助言を意図したものではなく、またそのような結果を生むものでもありません。本講演資料に含まれている情報については、完全性と正確性を期するよう努力しましたが、「現状のまま」提供され、明示または暗示にかかわらずいかなる保証も伴わないものとします。本講演資料またはその他の資料の使用によって、あるいはその他の関連によって、いかなる損害が生じた場合も、IBMは責任を負わないものとします。本講演資料に含まれている内容は、IBMまたはそのサプライヤーやライセンス交付者からいかなる保証または表明を引きだすことを意図したものでも、IBMソフトウェアの使用を規定する適用ライセンス契約の条項を変更することを意図したものでもなく、またそのような結果を生むものでもありません。

本講演資料でIBM製品、プログラム、またはサービスに言及していても、IBMが営業活動を行っているすべての国でそれらが使用可能であることを暗示するものではありません。本講演資料で言及している製品リリース日付や製品機能は、市場機会またはその他の要因に基づいてIBM独自の決定権をもっていつでも変更できるものとし、いかなる方法においても将来の製品または機能が使用可能になると確約することを意図したものではありません。本講演資料に含まれている内容は、参加者が開始する活動によって特定の販売、売上高の向上、またはその他の結果が生じると述べる、または暗示することを意図したものでも、またそのような結果を生むものでもありません。パフォーマンスは、管理された環境において標準的なIBMベンチマークを使用した測定と予測に基づいています。ユーザーが経験する実際のスループットやパフォーマンスは、ユーザーのジョブ・ストリームにおけるマルチプログラミングの量、入出力構成、ストレージ構成、および処理されるワークロードなどの考慮事項を含む、数多くの要因に応じて変化します。したがって、個々のユーザーがここで述べられているものと同様の結果を得られると確約するものではありません。

記述されているすべてのお客様事例は、それらのお客様がどのようにIBM製品を使用したか、またそれらのお客様が達成した結果の実例として示されたものです。実際の環境コストおよびパフォーマンス特性は、お客様ごとに異なる場合があります。

IBM、IBM ロゴ、ibm.com、CICS、DB2、IMS、Tivoli、WebSphere、は、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。
他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。
現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。

JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会社の米国およびその他の国における商標または登録商標です。

© 2012 IBM Corporation