

# JEUS JCAガイド

JEUS v8.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

## Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

## Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(JEUS®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りがないことを保証しません。

## Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

JEUS®は、TmaxSoft Co., Ltd.の登録商標です。

Java and Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Java、Solarisは、Oracle Corporation及びその子会社、関連会社の登録商標です。

Microsoft, Windows, and Windows NT are registered trademarks or trademarks of Microsoft Corporation.

Microsoft、Windows、Windows NTは、Microsoft Corporationの登録商標または商標です。

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

HP-UXは、Hewlett Packard Enterprise Companyの登録商標です。

---

AIX is a registered trademark of International Business Machines Corporation.

AIXは、International Business Machines Corporationの登録商標です。

UNIX is a registered trademark of X/Open Company, Ltd.

UNIXは、X/Open Company, Ltd.の登録商標です。

Linux is a registered trademark of Linus Torvalds.

Linuxは、Linus Torvaldsの登録商標です。

Other products and company names are trademarks or registered trademarks of their respective owners.

その他、記載されている会社名、製品名などは、各社の商号、商標または登録商標です。

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (TM, ®).

本マニュアルに記載されている会社名、システム名、製品名などには必ずしも商標表示(TM、®)を付記しておりません。

### **Open Source Software Notice**

Some modules or files of this product are subject to the terms of the following licenses. : APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory : \${INSTALL\_PATH}/lib/licenses

この製品の一部ファイルまたはモジュールは、APACHE2.0、CDDL1.0、EDL1.0、OPEN SYMPHONY SOFTWARE1.1、TRILEAD-SSH2、Bouncy Castle、BSD、MIT、SIL OPEN FONT1.1のライセンスに準拠します。

### **文書情報**

文書名: JEUS JCAガイド

発行日: 2016年10月14日

ソフトウェアバージョン: JEUS v8.0

ガイドバージョン: v2.1.1

---



# 目次

このガイドについて .....	xi
<b>第1章 JCA標準 .....</b>	<b>1</b>
1.1. 概要 .....	1
1.2. コネクタ・アーキテクチャ .....	1
1.2.1. アウトバウンド .....	2
1.2.2. インバウンド .....	3
1.3. リソース・アダプターとJDBCドライバ .....	4
1.4. CCI(Common Client Interface) .....	5
<b>第2章 アウトバウンド管理 .....</b>	<b>7</b>
2.1. コネクション・プールとコネクション管理 .....	7
2.1.1. アプリケーションのコネクション・リクエストのパターン .....	7
2.1.2. リソース・アダプターとコネクション・プールの関係 .....	9
2.1.3. コネクション・プールのメリット .....	9
2.1.4. コネクション・プールの設定 .....	10
2.1.5. コネクション・プールの機能 .....	14
2.2. トランザクション管理 .....	16
2.2.1. ローカル・トランザクション・リソースのグローバル・トランザクション(XA)への参加機能 .....	16
2.2.2. グローバル・トランザクション(XA)とコネクション・シェアリング .....	17
2.3. コネクション・プールの設定例 .....	17
2.3.1. コネクション・ファクトリーが1つの場合 .....	18
2.3.2. コネクション・ファクトリーが2つ以上の場合 .....	19
2.4. コネクション・プールのモニタリングおよび制御 .....	22
2.4.1. コネクション・プールの制御 .....	22
2.4.2. コネクション・プールのモニタリング .....	24
<b>第3章 インバウンド管理 .....</b>	<b>27</b>
3.1. ワーク・マネージャーの管理 .....	27
3.1.1. 基本概念 .....	27
3.1.2. ワーク・マネージャーの設定 .....	28
3.2. メッセージ・インフロー .....	29
<b>第4章 リソース・アダプター .....</b>	<b>31</b>
4.1. セキュリティー管理 .....	31
4.1.1. コネクション認証 .....	31
4.2. パッケージング .....	32
4.3. デプロイ .....	32
4.3.1. SHAREDモードのクラス・ローディング .....	33
4.3.2. 再デプロイ .....	33
4.4. リソース・アダプターをリソースとして登録 .....	33
4.4.1. WebAdminの使用 .....	34

付録 A. 環境設定の注意事項 .....	37
索引 .....	39

## 図目次

[図 1.1]	コネクタ・アーキテクチャの概要 .....	2
[図 1.2]	インバウンドのメッセージ・インフロー .....	3
[図 2.1]	JCAコネクション・プールの画面 .....	22
[図 2.2]	JCAコネクション・プールの生成 .....	23
[図 2.3]	JCAコネクション・プールの生成確認 .....	24
[図 2.4]	JCAコネクション・プールのモニタリング .....	24
[図 2.5]	JCAコネクション・プールに存在するコネクションの詳細情報確認 .....	25





## 例目次

[例 2.1]	コネクション・リクエスト : <<ejb-jar.xml>> .....	8
[例 2.2]	コネクション・リクエスト : <<jeus-ejb-dd.xml>> .....	8
[例 2.3]	コネクション・ファクトリーが1つの場合のコネクション・プールの設定 : <<ra.xml>> .....	18
[例 2.4]	コネクション・ファクトリーが1つの場合のコネクション・プールの設定 : <<jeus-connector-dd.xml>> .....	19
[例 2.5]	コネクション・ファクトリーが2つの場合のコネクション・プールの設定 : <<ra.xml>> .....	19
[例 2.6]	コネクション・ファクトリーが2つの場合のコネクション・プールの設定 : <<jeus-connector-dd.xml>> .....	21
[例 3.1]	ワーク・マネージャーの設定 : <<jeus-connector-dd.xml>> .....	28
[例 3.2]	MDB連動リソースの設定 : <<jeus-ejb-dd.xml>> .....	30



# このガイドについて

## 対象読者

本書では、全社情報システム(または、企業情報システム、以下、EIS(Enterprise Information System))をJEUS<sup>®</sup>(以下、JEUS)と連携するためのリソース・アダプターの開発者およびJEUSでリソース・アダプターを使用する開発者を対象としています。JEUSのJ2EE Connector Architecture標準(以下、JCA標準)実装の特徴について説明した後、リソース・アダプターをJEUSに合わせてパッケージングする方法について説明します。

## 前提知識

本書を理解するためには、事前に以下の内容を把握しておく必要があります。

- コネクタ・アーキテクチャおよびリソース・アダプターの基本的な内容、用語、概念
- Javaの基本概念と使用方法([J2EE Connector Architecture Specification - version 1.7](#))

JEUSの基本的な使用方法と製品を理解するには、以下のガイドについてあらかじめ熟知することをお勧めします。

- JEUS 紹介ガイド
- JEUS インストール & スタートガイド

本書のすべてのサンプルと環境構成は、UNIXスタイルに準拠します。Microsoft Windows<sup>™</sup>(以下、Windows)など他の環境で作業を行う場合は、次のような事項を考慮してください。たとえば、Windowsプラットフォームでは、ディレクトリー区切り子をUNIXスタイルのスラッシュ(/)からWindowsスタイルのバックスラッシュ(\)に変えて使用してください。また、環境変数もWindowsスタイル(%%)に変更して使用してください。本書で触れているJEUS\_HOMEは、JEUSがインストールされているディレクトリーです。

## 制限事項

本書の内容は、Java標準に準拠して作成されていますが、本書で触れているJava EEやJava仕様については詳しく取り上げていません。関連内容についてはJava関連ドキュメントを参照してください。

---

## 参考

JCA標準についての概略的な内容は、<http://www.oracle.com/technetwork/java/index.html>に紹介されています。さらに詳しい内容については、JCA 1.7標準あるいはそれに準ずる書籍を参照してください。

---

## 本書の構成

本書は、計4章と付録で構成されています。

- 「[第1章 JCA標準](#)」

JCA標準およびリソース・アダプターについて説明します。

- 「[第2章 アウトバウンド管理](#)」

アウトバウンド・コミュニケーションにおいてJEUSの役割と機能について説明します。

- 「[第3章 インバウンド管理](#)」

インバウンド・コミュニケーションにおいてJEUSの役割と機能について説明します。

- 「[第4章 リソース・アダプター](#)」

リソース・アダプターのセキュリティ管理、パッケージング、デプロイおよびリソース・アダプターをコネクタ・リソースとして登録する方法について説明します。

- 「[付録 A. 環境設定の注意事項](#)」

jeus-connector-dd.xmlファイルを設定する際の注意事項について説明します。

## 表記上の規則

表記	意味
<<AaBbCc123>>	プログラム・ソースコードのファイル名
<Ctrl>+C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
ハイパーリンク	メール・アカウント、Webサイト
>	メニューの実行順
+----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
<div>参考</div>	参照/注意事項
<div>注</div>	注意事項
[図 1.1]	図の名前
[表 1.1]	表の名前
AaBbCc123	Javaコード、XMLドキュメント
[ <i>command argument</i> ]	オプション・パラメータ
< xyz >	「<」と「>」の間の内容は実際に使用される特定の名前または値で置き換えられる
	構文の中の相互に排他的な選択項目の選択肢を示す 例) A B: AとBのいずれかを選択
...	パラメータ、値、または他の情報が繰り返される
\${ }	環境変数

## システム要件

	要求事項
プラットフォーム	Solaris 9, 10, 11
	HP-UX 11.x, 11i, 11iV2
	IBM AIX 5L, 6L, AIX 7L
	MS Windows 2008, 2012, Vista, 7, 8
ハードウェア	最小2GB以上、推奨20GBのハードディスク容量
	推奨1GB以上のメモリー容量
JDK	JDK 7, JDK 8

## 関連文書

ガイド	説明
JEUS 紹介ガイド	JEUSサーバーについて全般的に紹介し、JEUSのアーキテクチャーを含む各構成要素について記述しています
JEUS インストール&スタートガイド	JEUSについて紹介し、JEUSのインストールおよび開始方法について記述しています
JEUS サーバガイド	JEUSシステムおよびサーバーの概要とシステムの管理方法について記述しています
JEUS アプリケーション&デプロイメントガイド	Java EEアプリケーションをJEUSにデプロイするための様々な方法とツールについて記述しています
JEUS リファレンスガイド	JEUSを使用するために必要な詳細設定とJEUSの使用方法について記述しています
JEUS WebAdminガイド	JEUSのWeb管理ツールであるWebAdminを利用したJEUSの設定および制御、モニタリング、クラスタリング、リソースの設定および管理について記述しています

## 参考文献

- [J2EE Connector Architecture Specification - version 1.7](#)
- XML Reference - jeus-connector-dd.xmlの設定  
JEUS\_HOME/docs/reference/schema/index.html

## お問合せ先

### Korea

TmaxSoft Co., Ltd.  
45, Jeongjail-ro, Bundang-gu,  
Seongnam-si, Gyeonggi-do, 13613  
South Korea  
Tel: +82-31-8018-1000  
Fax: +82-31-8018-1115  
Email: [info@tmax.co.kr](mailto:info@tmax.co.kr)  
Web (Korean): <http://www.tmaxsoft.com>  
TechNet: <http://technet.tmaxsoft.com>

### USA

TmaxSoft Inc.  
101 North Wacker Drive, Suite 2014,  
Chicago, IL 60606  
U.S.A  
Tel: +1-312-525-8330  
Email: [info@tmaxsoft.com](mailto:info@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/us\\_en/home](http://www.tmaxsoft.com/us_en/home)

### Japan

TmaxSoft Japan Co., Ltd.  
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073  
Japan  
Tel: +81-3-5765-2550  
Fax: +81-3-5765-2567  
Email: [info@tmaxsoft.co.jp](mailto:info@tmaxsoft.co.jp)  
Web (Japanese): <http://www.tmaxsoft.co.jp>

## China

Beijing TmaxSoft System Software Co., Ltd.  
Room103, No.2 Huizhong Building, Seven Street Shangdi,  
Haidian District, Beijing, 100085  
P.R.China  
Tel: +86-10-6298-8827  
Email: [info@tmaxsoft.com.cn](mailto:info@tmaxsoft.com.cn)  
Web (Chinese): [http://www.tmaxsoft.com/cn\\_en/home\\_cn\\_en](http://www.tmaxsoft.com/cn_en/home_cn_en)

## Brazil

Tmax Brasil Sistemas e Serviços Ltda.  
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel  
Alphaville Barueri, Sao Paulo, 06472-001  
Brazil  
Tel: +55-11-4191-3100  
Fax: +55(11) 4191-3705 (extension#112)  
Email: [info.bra@tmaxsoft.com](mailto:info.bra@tmaxsoft.com)  
Web (Portuguese): [http://www.tmaxsoft.com/br\\_en/home\\_br\\_en](http://www.tmaxsoft.com/br_en/home_br_en)

## Russia

Tmax Rus L.L.C.  
Leninsky prospekt, 113/1 (Park Place Moscow),  
Office 318e, Moscow, 117198  
Russia  
Tel: +7(495)970-01-35  
Email: [info.rus@tmaxsoft.com](mailto:info.rus@tmaxsoft.com)  
Web (Russian): [http://www.tmaxsoft.com/ru\\_ru/home\\_ru\\_ru](http://www.tmaxsoft.com/ru_ru/home_ru_ru)



## Singapore

Tmax Singapore Pte. Ltd.  
430 Lorong 6, Toa Payoh #10-02,  
OrangeTee Building, 319402  
Singapore  
Tel: +65-6259-7223  
Fax: +65-6258-7112  
Email: [info.sg@tmaxsoft.com](mailto:info.sg@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/sg\\_en/home\\_sg\\_en](http://www.tmaxsoft.com/sg_en/home_sg_en)

## United Kingdom

TmaxSoft UK Ltd.  
215 Knyvett House, Watermans Business Park,  
The Causeway, Staines TW18 3BAB  
United Kingdom  
Tel: +44-1784-895005  
Email: [info.uk@tmaxsoft.com](mailto:info.uk@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/gb\\_en/home\\_gb\\_en](http://www.tmaxsoft.com/gb_en/home_gb_en)

## Canada

TmaxSoft Canada, Inc.  
2425 Matheson Blvd East, 8th floor,  
Unit 824 Mississauga, ON, L4W 5K4  
Canada  
Tel: +1-905-361-2888  
Email: [info.canada@tmaxsoft.com](mailto:info.canada@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/ca\\_en/home\\_ca\\_en](http://www.tmaxsoft.com/ca_en/home_ca_en)

## Australia

TmaxSoft Proprietary Limited  
L32, 101 Miller Street, North Sydney 2060  
Australia  
Tel: +91-9845-330-704  
Email: [info.aus@tmaxsoft.com](mailto:info.aus@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/au\\_en/home\\_au\\_en](http://www.tmaxsoft.com/au_en/home_au_en)

## India

TmaxSoft Technologies Private Limited  
Sobha Alexander Plaza, 3rd Floor,  
16/2 Commissariat Road, Bangalore-560025  
India  
Tel: +91-9845-330-704  
Email: [info.india@tmaxsoft.com](mailto:info.india@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/in\\_en/home\\_in\\_en](http://www.tmaxsoft.com/in_en/home_in_en)

## Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office  
Windowist Tower. Eski Buyukdere Cad. No:26,  
Maslak 34467 Istanbul  
Turkey  
Tel: +90-544-553-6045  
Email: [cslee@tmaxsoft.com](mailto:cslee@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/tr\\_en/home\\_tr\\_en](http://www.tmaxsoft.com/tr_en/home_tr_en)

# 第1章 JCA標準

本章では、JCA標準およびリソース・アダプターについて説明します。

## 1.1. 概要

JCA(Java EE Connector Architecture)は、J2EE 1.3でJCA 1.0で紹介された後、J2EE 1.4ではインフロー・メッセージの処理や様々な拡張機能が含まれ、JCA 1.5に発展しました。

現在はJCA 1.7が最新バージョンとしてリリースされており、リソース・アダプターのアノテーション使用およびセキュリティ・インフローなどの機能を追加でサポートしています。

## 1.2. コネクター・アーキテクチャー

コネクター・アーキテクチャーは、メインフレーム、ERP(Enterprise Resource Planning)、TPモニター、レガシー・データベース・システムなどを包括する企業情報システム(EIS(Enterprise Information System))との連動のために定義された標準です。標準が存在しない場合は、EISベンダーとWeb Application Server(以下、WAS)ベンダーの間に別途のカスタム・ドライバを実装する必要があるため、N by M問題をもたらします。この問題は、Java EE環境の移植性や拡張性に深刻な制約を与えます。

同問題を解決するため、JCA標準はコネクター・アーキテクチャーという概念を導入し、リソース・アダプターという相互連動に必要なアダプターを定義しました。EISベンダーがリソース・アダプターを提供することで、コネクター・アーキテクチャーによって複数のWASにおいてコード・レベルを修正せずに相互互換性を持って動作することができます。リソース・アダプターについての詳細内容は、「[1.3. リソース・アダプターとJDBCドライバ](#)」を参照してください。

---

### 参考

コネクター・アーキテクチャーの詳しい内容については記述しないため、JCA 1.7標準やそれに準ずる書籍を参照してください。

---

コネクター・アーキテクチャーは、大きくアウトバウンドとインバウンドで分けられます。「アウト」と「イン」は、WASを基準にしてEISとのコミュニケーション方向性を表した言葉です。

- アウトバウンド(Outbound)

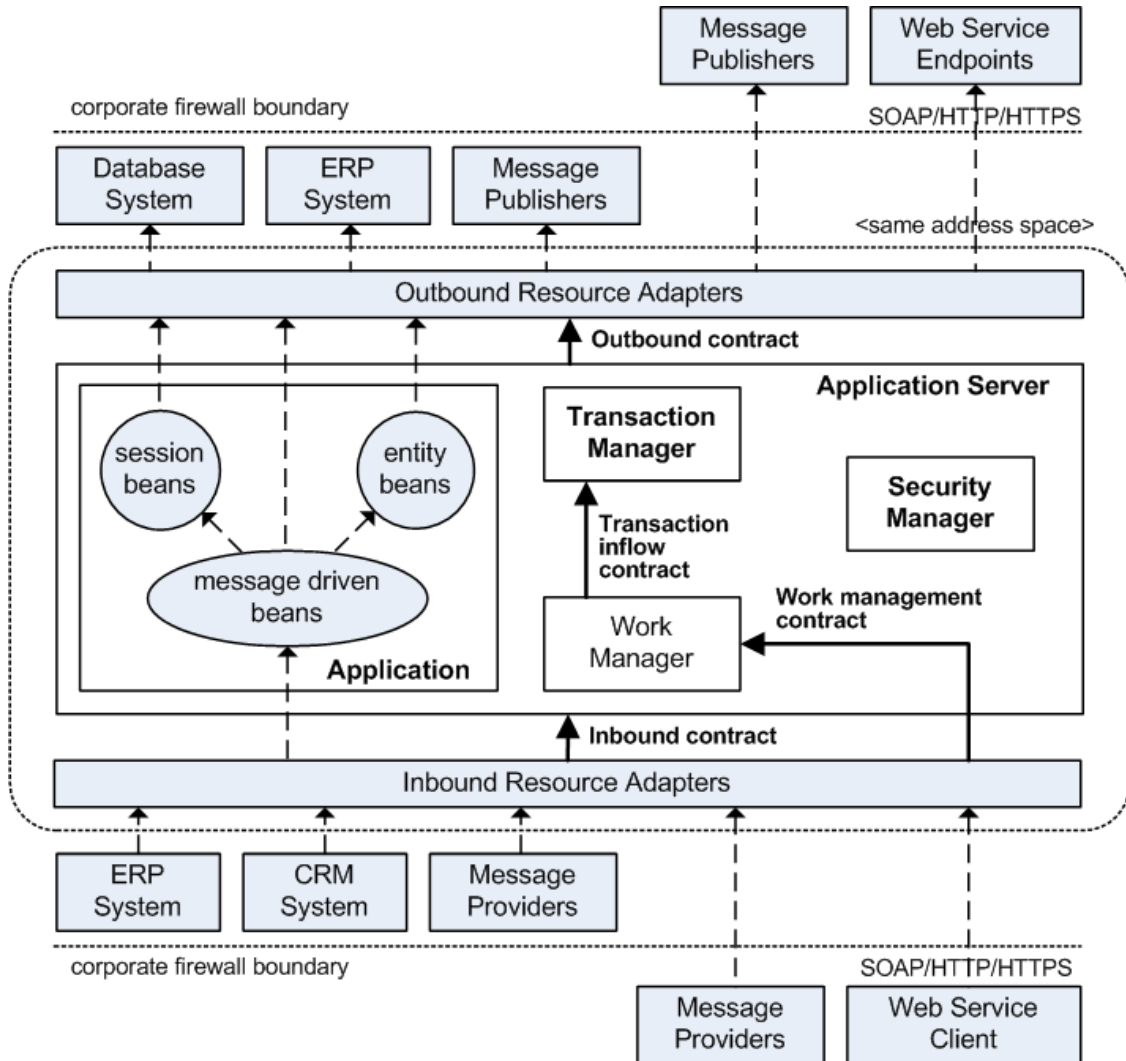
WASにデプロイされたアプリケーションからEISへのフローを意味します。

- インバウンド(Inbound)

EISからWASのアプリケーションへのフローを意味します。

このようなアウトバウンドおよびインバウンド・コミュニケーションのために、WASとリソース・アダプターが行うべき作業を定義したものがコネクタ・アーキテクチャです。

【図 1.1】コネクタ・アーキテクチャの概要



### 1.2.1. アウトバウンド

アウトバウンドを表す代表的な用語がコネクションです。WASのアプリケーションがEISにリクエストしてその結果を取得したい場合、コネクションを確立してリクエストを送信することになります。

代表的な例として、サーブレットやEJBコンポーネントからデータベースにSQLを送信する場合があります。これは、[図 1.1](#)のセッションBeansまたはエンティティBeansでアウトバウンド・リソース・アダプターを介してEISにリクエストするフローに該当します。

JEUSでは、このようなコネクションの効率的な管理のために以下の機能を提供します。

- コネクション・プールの提供

- コネクション・プールを利用してコネクションを毎回生成せずに再使用し、性能の効率性を高めます。
- 負荷状況では、WASからリソースに送るリクエストが無限に増えないように制御します。

- トランザクション管理

- グローバル・トランザクション(XA)が必要な場合には、自動的にトランザクション・マネージャーとの連動をサポートします。
- グローバル・トランザクション(XA)の場合、コネクション・シェアリング(コネクション共有)をサポートします。

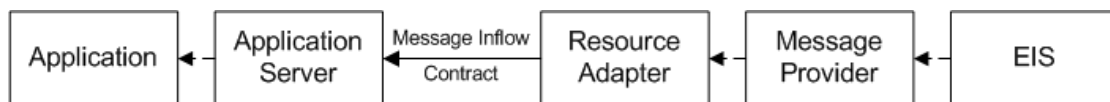
以外にも、JEUSでは様々な付加機能を提供しています。アウトバウンドについての詳細内容は、「[第2章 アウトバウンド管理](#)」を参照してください。

## 1.2.2. インバウンド

JCA標準は、JMS(Java Message Service)を含む多様な種類のメッセージ・ベースのサービスと連動できるメカニズムを定義しています。この場合、リソース・アダプターはメッセージ・プロバイダーの役割をし、エンド・ポイント・アプリケーションはメッセージ・コンシューマーの役割をします。送信されるメッセージのタイプはWASとは無関係であり、実際にメッセージが送信されるメソッドを定義したメッセージ・インターフェースはリソース・アダプターが任意で決めることができます。

以下は、インバウンドのメッセージ・インフローの構成です。

**[図 1.2] インバウンドのメッセージ・インフロー**



メッセージ・プロバイダーは、メッセージ・コンシューマーの直接的な要求に対してのみメッセージを送信しており、両領域の間にはいかなるコンテキストの送信、および共有も行われません。EISで発生したトランザクションあるいは認証に関する情報をWASに渡すためには、ワーク・マネージャーを使用してコンテキストを送信する必要があります。

JEUSでのインバウンド・コミュニケーション管理についての詳細内容は、「[第3章 インバウンド管理](#)」または「[3.2. メッセージ・インフロー](#)」を参照してください。

---

### 参考

コンテキストの送信についての詳しい内容は、JCA標準を参照してください。

---

## ワーク・マネージャー

一般的に、アウトバウンド・メッセージのみ処理するリソース・アダプターの場合は別途のスレッドは不要ですが、インバウンド・メッセージを処理するリソース・アダプターの場合はスレッドが必要になることもあります。そのときJCA標準では、リソース・アダプター内で任意でJavaスレッドを生成せずに、WASが提供するワーク・マネージャーを利用することをお勧めしています。特に、EISでスタートしたトランザクションや認証情報を連動してジョブを実行する場合には、ワーク・マネージャーを使用する必要があります。

JEUSのワーク・マネージャー実装はスレッド・プールをベースにします。リソース・アダプターがワーク・インスタンスを送信すると、スレッド・プールでワーカー・スレッドを取得して作業を行います。リソース・アダプターのワーカー・スレッドから作業の開始、終了、例外イベントを取得したい場合は、イベント・リスナーを渡すことができます。また、ジョブが実行されるときに必要なトランザクション、認証情報などをコンテキストに渡すことも可能です。

JEUSのワーク・マネージャーの実装については、「[3.1. ワーク・マネージャーの管理](#)」を参照してください。ワーク・マネージャーの基本概念については、JCA標準を参照してください。

## 1.3. リソース・アダプターとJDBCドライバ

JDBCがリレーショナル・データベース(RDB)との連動を標準として定義したものなら、リソース・アダプターはRDBを含めた様々な情報システムとの連動標準を定義しています。すなわち、リソース・アダプターはWASと連動するための外部リソースに対し、JDBCドライバより一般的かつ広範な定義がなされたドライバです。

リソース・アダプターとJDBCドライバの大きな相違点は、リソース・アダプターはJava EE標準で定義した概念で、JDBCドライバはJava SE向けに作られた概念であることです。そのため、JDBCドライバはスタンドアローンJavaアプリケーションで直接使用することができます。

また、JDBCドライバはJARファイルをシステム・クラス・パスに設定して使用しており、ドライバをアップデートするためには、現在実行中のJVMを終了する必要があります。一方、リソース・アダプターはJava EEで定義したアプリケーションなので、RARファイルにパッケージングされており、WASにデプロイすることが可能です。

リソース・アダプターのバージョンをアップグレードするには、WASを中断せずにリデプロイします。リソース・アダプターもJDBCドライバのようにJARファイルに変更した後、システム・クラス・パスに設定して使用できますが、基本的にはリデプロイが可能なJava EEアプリケーションです。

以下は、リソース・アダプターとJDBCドライバーを比較したものです。

	リソース・アダプター	JDBCドライバー
概念	Java EE標準で定義した概念	Java SEのための概念(JDBCドライバーはスタンドアローンJavaアプリケーションで直接使用可能)
連動	リレーショナル・データベース(RDB)を含め、様々な情報システムの連動標準を定義	リレーショナル・データベースとの連動を標準として定義
デプロイ	RARファイルにパッケージングするようになっており、WASにデプロイ可能	JARファイルをシステム・クラス・パスに設定して使用
アップデート	WASを中断せずにアップデート可能	現在実行中のJVMを終了した後にアップデート

リソース・アダプターはJDBCドライバーに比べて一般的かつ広範な定義をしたドライバーであり、より容易な使用が可能です。

## 1.4. CCI(Common Client Interface)

CCIIは、アウトバウンド・リソース・アダプターを統一されたAPIで提供するためのものです。JCA標準はリソース・アダプターを開発する際にCCIを使用することをお勧めしています。

CCIIは、サービスの呼び出しをインタラクション(対話)に概念化しており、サービス方式に関する内容については、インタラクション仕様(Interaction Spec)と呼ばれるEISに従属的なインスタンスで抽象化します。サービス呼び出しのパラメータと戻り値は、レコード型で提供されます。

インタラクション仕様は、一般的にサービス関数名(FunctionName)とサービス形式(InteractionVerb)を含めることをお勧めします。CCIIは同期送信、同期受信、同期送受信の3つのサービス形式をサポートしており、非同期サービスはサポートしません。

---

### 注

CCIIはアプリケーションとリソース・アダプター間のインターフェースも統一するとの趣旨で作られたインターフェースです。WASは、アプリケーションとリソース・アダプター間のインターフェースには関与しておらず、基本的にはjava.lang.Objectとして扱います。

---

以下は、CCIを利用してEISのサービスを呼び出すための準備プロセスです。

```
// get Connection to EIS by lookup ConnectionFacotry in JNDI
javax.naming.Context nc = new InitialContext();
javax.resource.cci.ConnectionFactory cf =
(ConnectionFactory)nc.lookup("java:/comp/env/eis/ConnectionFactory");
javax.resource.cci.Connection cx = cf.getConnection();
```

```
// create Interaction
javax.resource.cci.Interaction ix = cx.createInteraction();

// create Interaction Spec
com.wombat.cci.InteractionSpecImpl spec = .....
spec.setFunctionName("<EIS_FUNCTION_NAME>");
spec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE)
. . .
```

サービスに必要なデータをレコードに保存して、以下のようにサービスを呼び出します。

```
// call EIS service
boolean ret = ix.execute(spec, input, output);
```

---

### 参考

CCIの詳細内容については、JCA標準またはCCI関連の書籍を参照してください。

---



## 第2章 アウトバウンド管理

本章では、アプリケーションからEISに渡るアウトバウンド・コミュニケーションにおけるJEUSの役割や機能について、コネクション・プールとトランザクションの連動を中心にして説明します。

### 2.1. コネクション・プールとコネクション管理

アウトバウンド・コミュニケーションにおいて、アプリケーションがEISにリクエストし、その結果を取得する場合にはコネクションが必要となります。JEUSでは、アプリケーションがコネクションを効率的に使用できるようにコネクション・プールを提供します。

#### 2.1.1. アプリケーションのコネクション・リクエストのパターン

アプリケーションはコネクション・ファクトリーをリクエストした後、当該ファクトリーからコネクションを取得することができます。これは、JDBCコネクションを取得するときにデータソースをルックアップした後、データソースからコネクションを取得する方法と同じです。

以下は、アプリケーションのコネクション・リクエスト・パターンの例です。

```
// obtain the initial JNDI Naming context
Context initctx = new InitialContext();
// perform JNDI lookup to obtain the connection factory
javax.resource.cci.ConnectionFactory connectionFactory =

(javax.resource.cci.ConnectionFactory)initctx.lookup("java:comp/env/eis/sampleEIS");
javax.resource.cci.Connection conn = connectionFactory.getConnection();
try {
    // do some works
} finally {
    conn.close();
}
```

---

#### 参考

上記の例ではCCIを例にしていますが、実際のコネクション・ファクトリーには特別に定義されたインターフェースが存在しておらず、リソース・アダプターで独自の定義し、これをアプリケーションが使用することになります。

---

上記のアプリケーションがEJBの場合、アノテーションあるいはejb-jar.xmlの<resource-ref>を以下のとおり設定する必要があります。

**[例 2.1] コネクション・リクエスト : <<ejb-jar.xml>>**

```
<ejb-jar>
. . .

<enterprise-beans>
. . .
  <session>
    . . .
    <resource-ref>
      <res-ref-name>eis/sampleEIS</res-ref-name>
      <res-type>
        javax.resource.cci.ConnectionFactory
      </res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
    </resource-ref>
    . . .
```

jeus-ejb-dd.xmlには「eis/sampleEIS」が実際にマッピングされるJNDI名を設定します。

以下例の「sampleConnectionPool」は、リソース・アダプターをデプロイするとき、jeus-connector-dd.xmlに記述するコネクション・プールのJNDI名です。

**[例 2.2] コネクション・リクエスト : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
  <beanlist>
    <jeus-bean>
      . . .
      <res-ref>
        <jndi-info>
          <ref-name>eis/sampleEIS</ref-name>
          <export-name>sampleConnectionPool</export-name>
        </jndi-info>
      </res-ref>
    </jeus-bean>
    . . .
```

---

**参考**

jeus-connector-dd.xmlについての詳細内容は、「[第4章 リソース・アダプター](#)」を参照してください。

---

## 2.1.2. リソース・アダプターとコネクション・プールの関係

アプリケーションがJNDIを介してコネクション・ファクトリーをリクエストすると、JEUSはリソース・アダプターにコネクション・ファクトリーの生成をリクエストします。

JEUSがコネクション・ファクトリーの生成をリクエストするときは、リソース・アダプターにコネクション・マネージャーを渡します。このコネクション・マネージャーの実装が、**JEUSコネクション・プール**です。

リソース・アダプターがコネクション・ファクトリーを生成してJEUSに渡すと、JEUSはこれをアプリケーションに渡します。アプリケーションが受け取ったコネクション・ファクトリーにコネクションをリクエストすると、リソース・アダプターはJEUSから取得したコネクション・マネージャーにコネクションをリクエストします。そうすると、JEUSコネクション・マネージャーはプーリングしていた物理的なコネクション(あるいはマネージド・コネクション)を1つ抽出してコネクション・ハンドルを生成した後、リソース・アダプターに返します。その後、アダプターがアプリケーションに同コネクションを渡します。物理的なコネクションとコネクション・ハンドルは、JCA標準に定義されている概念です。

アプリケーションは、コネクションが不要になったときには必ずリターンする必要があります。これを一般的にはコネクションを閉じると表現します。アプリケーションが正常にコネクションを閉じない場合は、常にコネクション・リークが発生することになります。

JEUSが渡したコネクション・マネージャーの使用有無はリソース・アダプターが決めます。リソース・アダプターがコネクション・マネージャーを使用せず、内部的にコネクションを生成した場合、JEUSはコネクションの管理およびアプリケーションとEIS間のコミュニケーション・プロセスについては関与しません。

---

### 参考

1. コネクション・マネージャーについての詳細内容は、`javax.resource.spi.ConnectionManager`と`javax.resource.spi.ManagedConnectionFactory`のJavadocおよびJCA標準を参照してください。
  2. 物理的なコネクションとコネクション・ハンドルの詳細については、JCA標準の「Chapter 6. Connection Management」を参照してください。
- 

## 2.1.3. コネクション・プールのメリット

JCAで提供するJEUSコネクション・プールの機能はDBコネクション・プールと類似していますが、以下のようなメリットがあります。

- 高性能

データベースとのコネクションのみならず、外部リソースとのコネクション生成には時間がかかります。しかし、コネクション・プールの物理的なコネクションは毎回新たに生成するのではなく、事前に生成しておくか再使用するため、コネクション生成のオーバーヘッドが減少されます。また、アプリケーションがコネクションの使用を終えて閉じた場合にも、実際に切断するのではなくプールにリターンさせ、コネクションの中断によるオーバーヘッドを減らすことができます。

- 同時接続管理

同時にリソースをリクエストするクライアントの数を制御することが可能です。負荷状況でリソースに処理が集中し、サービスが不能になることを防ぐことができます。

---

#### 参考

DBコネクション・プールについての詳細内容は、『*JEUS サーバガイド*』の「第6章 DBコネクション・プールとJDBC」を参照してください。

---

## 2.1.4. コネクション・プールの設定

JCAコネクション・プールはリソース・アダプターをデプロイする際、jeus-connector-dd.xmlを記述して設定します。

このとき、アプリケーションでコネクション・プールを使用するために必要となるJNDI名は、JEUSで決められた規則に従って自動的に決定されます。しかし、ユーザが直接JNDI名を指定する場合にはjeus-connector-dd.xmlに記述する必要があります。jeus-connector-dd.xmlの<jeus-connector-dd>下位の<connection-pool>を使用して設定します。jeus-connector-dd.xmlをRARに含ませる方法は、「[第4章 リソース・アダプター](#)」を参照してください。

JCA標準により、1つのリソース・アダプターには複数のコネクション・プールを設定することができます。このようなコネクション・プールの基準となるのは、ra.xmlの<connection-definition>と、その下位要素の<connectionfactory-interface>です。たとえば、<connection-definition>が5つ設定されていると、JEUSには最小5つのコネクション・プールが構成されます。

---

#### 注

ra.xmlに定義する<connectionfactory-interface>は重複できません。重複する値が存在する場合は、ra.xmlの有効性チェックに失敗し、デプロイできません。

---

以下は、jeus-connector-dd.xmlの各タグについての説明です。実際の設定例は、「[2.3. コネクション・プールの設定例](#)」を参照してください。

- <connectionfactory-interface>

- ra.xmlに<connection-definition>が2つ以上の場合は、必ずこの値を設定します。設定しないと、コネクション・プールが生成できないため、デプロイ・エラーが発生します。
- ra.xmlの各<connection-definition>に定義された<connectionfactory-interface>値をそのまま入力します。<connectionfactory-interface>の詳しい内容は、「[第4章 リソース・アダプター](#)」を参照してください。

- **<export-name>**

- コネクション・プールのJNDI名です。
- サーバーにおいて一意の名前である必要があり、必須項目です。

- **<transaction-support>**

- コネクション・プールがサポートするトランザクション・タイプを以下から設定します。同設定は、ra.xmlの設定に優先します。
  - NoTransaction
  - LocalTransaction
  - XATransaction

- **<user>**

- コネクションを生成するときに必要なログインをコネクション・プールに委ねる場合に設定するユーザーIDです。

- **<password>**

- コネクションを生成するときに必要なログインをコネクション・プールに委ねる場合に設定するパスワードです。
- 暗号化して保存するときは、以下の形式で入力します。

```
{algorithm}ciphertext
```

例)

```
{DES}FQrLbQ/D8O1lDVS71L28rw==
```

- **<use-lazy-transaction-enlistment>**

- JCA標準で言及しているトランザクションの最適化機能の1つである「Lazy Transaction Enlistment」オプションの使用有無を決定します。(デフォルト値: false)

- **<pool-management>**

コネクション・プールの主なオプションは、<connection-pool>の<pool-management>下位に存在します。

- **<min>** : コネクション数のデフォルト値です。(デフォルト値: 2)

- **<max>** : プーリングするコネクション数の最大値です。(デフォルト値: 10)
- **<step>** : コネクション数を増やす必要がある場合に、コネクションの増分を設定します。(デフォルト値: 1)
- **<period>** : 設定された間隔ごとにコネクション・プールのサイズを<min>値に合わせます。アイドル・コネクションが存在しない場合は、<min>値に調整しません。既存の<pooled-timeout>の代わりにこの設定を使用します。(デフォルト値: 10分、単位: ms)
- **<wait-connection>** : アイドル・コネクションが存在せず、コネクション・プールのサイズも最大値に到達された場合において、コネクション・リクエストを処理する方法を決定します。

タグ	説明
<wait-connection>	<ul style="list-style-type: none"> <li>- true: システムは使用可能なコネクションを取得するために待機します。待機後にもコネクションが取得できなかった場合は例外(exception)を発生させます</li> <li>- false : コネクションを新たに作成してアプリケーションに提供し、アプリケーションがこれを返したらプーリングせずに閉じて破棄します。このように使い捨てという意味で、JEUSでは「ディスポーザブル・コネクション (disposable Connection)」と言います(デフォルト値)</li> </ul>
<wait-timeout>	ユーザーがコネクションのために待機する時間です、ユーザーがこの時間の間待機しても、いかなるコネクションも使用できない場合は例外が発生します。<wait-connection>がtrueの場合のみ有効です(デフォルト値: 10秒、単位: ms)

- **<use-match-connection>** : コネクション・マッチの使用有無を決定します。(デフォルト値: false)
- **<allow-disposable-connection-when-match-failed>** : コネクション・マッチに失敗した場合、使い捨てのコネクションを使用するか否かを設定します。コネクション・マッチをしない場合、この値は無効になります。(デフォルト値: false)
- **<connection-validation>** : コネクションの有効性チェックについての設定です。

タグ	説明
<enabled>	<p>コネクションの有効性チェック機能を使用するか否かを決定します。</p> <p>同設定値をtrueにしても、リソース・アダプターがjavax.resource.spi.ValidatingManagedConnectionFactoryを実装した場合のみ使用できません</p>

タグ	説明
<period>	コネクションの有効性チェックの周期を設定します。設定された周期に従って、アイドル状態にあるコネクションの有効性をチェックします(単位 : ms)
<non-validation-interval>	コネクション単位で有効性をチェックするとき、最後にコネクションを使用した時間とチェックするときの時間の差が、設定した時間より短い場合はチェックを行いません。同設定によって、有効性チェックによるオーバーヘッドを減らすことができます(単位: ms)
<validation-retrial-count>	有効性チェックは、デストロイ・ポリシーがFailedConnectionOnlyの場合は1回実行します。また、AllConnectionsの場合は1つのコネクションに対して実行してみて、それが失敗すると別のコネクションに対して再度行うので計2回実行することになります。  有効性チェックが足りないと判断される場合は、チェック回数を増やすこともできます
<destroy-policy-on-validation>	コネクションの有効性チェックに失敗した場合、当該コネクション・プールに存在するコネクションを処理するポリシーを決めるオプションです  <ul style="list-style-type: none"> <li>FailedConnectionOnly : 有効性チェックに失敗した物理的コネクションのみ閉じます(デフォルト値)</li> <li>AllConnections : 有効性チェックに失敗した場合、プールに存在している別のコネクションをもう一度チェックし、それでも失敗したら当該コネクション・プールの全コネクションを閉じます。アプリケーションが使用していたコネクションもすべて閉じられます</li> </ul>

- **<action-on-connection-leak>** : コンポーネント(主にStatelessコンポーネント - サーブレット/JSP、ステートレス・セッションBeans、MDB)で利用したコネクションに対するロギングやリターン・アクションを設定します。設定しなかった場合の基本動作は、サーバーに設定したAction On Resource Leakに従います。(デフォルト値 : Warning)
- **<connection-trace>** : コネクションをモニタリングするためのオプションです。現在基本機能は、どのアプリケーションがコネクションを使用しているのかが分かるように、getConnectionするときのスタック情報を表示します。さらに、サーバーに設定したAction On Resource Leakが動作する場合でも同情報を表示します。

オプション	説明
enabled	コネクション・トレース機能をon/offするオプションです
get-connection-trace	アプリケーションがコネクションを取得するとき(getConnection呼出し)のスタック・トレースを保存しておくオプションです(デフォルト値: true)
local-transaction-trace	アプリケーションがリソース・アダプターとローカル・トランザクション作業を行うときの情報を追跡するか否かを選択するオプションです。

オプション	説明
	<get-connection-trace>と一緒に使用すると、ローカル・トランザクションのコミットまたはロールバックが正常に行われていないアプリケーションを追跡するときに役立ちます

- **<max-use-count>** : 1つのコネクションに対して設定された数を全部使用すると、当該コネクションを捨て、新たなコネクションを確立することになります。(デフォルト値: 0、コネクションをチェンジしないという意味)
- **<pool-destroy-timeout>** : コネクション・プールをデストロイするときに待機する時間です。リソース・アダプターをアンデプロイするとき、プールをデストロイすることになりますが、コネクションを閉じながらリソースとネットワーク通信を行う場合、ハングアップする可能性があります。そのため、設定された時間の間待機してもデストロイが実行されなかったら、これを無視して引き続きアンデプロイを行います。(デフォルト値: 10秒)
- **<property>**
  - ManagedConnectionFactoryに適用するプロパティを追加します。ra.xmlに設定された値を置き換えるか追加するときに使用します。

---

#### 参考

Connection Match、Lazy Transaction Enlistmentについての詳細内容は、JCA標準を参照してください。

---

## 2.1.5. コネクション・プールの機能

コネクション・プールはコネクションの有効性チェックが可能で、コネクション・リークが処理できる機能を持っています。

### コネクションの有効性チェック

アプリケーションがコネクションをリクエストした場合、リソース・アダプターを使用してコネクションの有効性(Connection Validation)についてチェックを要求する機能です。同機能は、コネクションの内部的なエラーによる切断、ファイアウォールによるソケット通信の切断現象などをチェックするときに有効です。チェックが失敗すると物理的コネクションを新たに生成し、これに対するハンドルをアプリケーションに返します。



有効性チェックの作業が頻繁に行われ、オーバーヘッドが発生した場合は、<non-validation-interval>を設定します。同設定は、コネクション・チェックを実行するときの時間と、最後にコネクションを使用した時間との差が既に設定したインターバル内なら、無条件で有効だと判断してチェックが行われないようにします。

たとえば、インターバルを5秒(5000ms)に設定した場合、コネクションの最後の使用から5秒以内なら、そのコネクションは無条件で有効だと判断されます。

以下は、<non-validation-interval>の設定例です。

```
<non-validation-interval>5000</non-validation-interval>
```

---

#### 注

リソース・アダプターでコネクションの有効性チェックを行うには、必ずjavax.resource.spi.Validating ManagedConnectionFactoryを実装する必要があります。リソース・アダプターにチェックを要求したとき、外部リソースからの応答がないため待ち続ける状況が発生することもあります。現在のWASではこのようなトラブルを解決できる方法がないため、リソース・アダプターからタイムアウト・オプションを提供する必要があります。

---

ユーザーは有効性チェックに失敗した場合、コネクション・プールに存在する残りのコネクションに対して以下のうち1つのデストロイ・ポリシーを設定することができます。

ポリシー	説明
FailedConnectionOnly	有効性チェックに失敗したコネクションのみ破棄します(デフォルト値)
AllConnections	有効性チェックの失敗後すぐにコネクションを破棄するのではなく、再度プールからコネクションを取り出し、チェックを要求します。それでも失敗したらプールに存在するすべてのコネクションを破棄します

以下は、有効性チェックに失敗した場合、プールに存在するコネクションに対するデストロイ・ポリシーを設定する例です。

```
<destroy-policy-on-validation>AllConnections</destroy-policy-on-validation>
```

---

#### 参考

有効性チェックの回数を追加する場合には、<validation-retrial-count>を使用します。

---

## コネクション・リーク処理

JEUSは、サーバーまたはコネクション・プール別にコネクション・リークに関するアクションを設定することができます。

サーブレット/JSP、ステートレス・セッションBean、メッセージ・ドリブンBeanのようにスタートとエンドが明確なコンポーネントの場合、使用したコネクションを正常に返したのかを確認することができます。コネクションを返していない場合、呼び出しマネージャーを利用してロギングを残すか、強制的にクローズするアクションが設定できます。ただし、JCAコネクションの場合、呼び出しマネージャーの自動クローズ・アクションに対して例外が存在します。

JCAコネクション・プールでは、JDBCの`java.sql.Connection`のようにどのメソッドが`close`の役割をするのが把握できないため、直接コネクションを閉じることはできません。ただし、リソース・アダプターの`ra.xml`に設定された`<connection-definition>`の`<connection-interface>`が以下のような場合は、`close`メソッドがすでに把握できており、メソッドにどのパラメータも存在しないため直接コネクションを閉じます。

以下は、JCAコネクションの呼び出しマネージャーの自動クローズ・アクションの例外メソッドです。

- `java.sql.Connection`
- `javax.resource.cci.Connection`

上記の`<connection-interface>`でない場合には直接コネクションを閉じることはできませんが、`javax.resource.spi.ManagedConnection`に定義された`cleanup`メソッドを呼び出した後、当該コネクションを強制的にプールに返します。

`cleanup`メソッドにはリソース・アダプターが以下のような動作をするように定義されています。

```
The cleanup should invalidate all connection handles  
that had been created using this ManagedConnection instance.
```

---

#### 参考

呼び出しマネージャーについての詳細内容は、『*JEUS サーバガイド*』の「2.3.1.2. Action On Resource Leakの設定」を参照してください。

---

## 2.2. トランザクション管理

本節では、JEUSでトランザクション・タイプがローカル・トランザクション、グローバル・トランザクションであるコネクション・プールを管理する方法について説明します。

### 2.2.1. ローカル・トランザクション・リソースのグローバル・トランザクション(XA)への参加機能

原則的にローカル・トランザクションはアプリケーションとリソース・アダプター間のトランザクションであるため、グローバル・トランザクション(XA)が存在しない状況でリソース・アダプターにコネクションを要求する場合、JEUSが関与することはできません。

しかし、グローバル・トランザクション内でローカル・トランザクション・タイプのコネクション・プールを使用する場合、リソース・アダプターのManagedConnectionから取得できるjavax.resource.spi.LocalTransactionを、まるでXAリソースであるかのようにエミュレーションすることができます。

グローバル・トランザクションをサポートしないリソースであっても、リソース・アダプターがローカル・トランザクション・タイプをサポートしていれば、最大1つのリソースがグローバル・トランザクションに参加できます。これは、JDBCでXAエミュレーション機能を使用するコネクション・プール・データソースとほぼ同様です。

---

#### 参考

データソースについての詳細内容は、『JEUS サーバガイド』の「6.4. データソース設定」を参照してください。

---

## 2.2.2. グローバル・トランザクション(XA)とコネクション・シェアリング

同じグローバル・トランザクションにおいて、1つのリソースに対して常に1つのコネクションのみ使用することをコネクション・シェアリング(コネクション共有)と言います。JEUSでは、常にコネクション・シェアリングをサポートしており、基本的に何も設定されていない場合でも共有するようになっています。

コネクション・シェアリング機能が不要な場合は、アプリケーション別にejb-jar.xml, web.xmlの<resource-ref>にUnshareableと設定します。

```
<resource-ref>
  <res-ref-name>jca/pool</res-ref-name>
  <res-type>javax.resource.cci.ConnectionFactory</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

---

#### 参考

1. コネクション・シェアリングの詳細内容は、JCA標準1.7の「7.9 Connection Sharing」を参照してください。
  2. DBコネクション・プールでもコネクション・シェアリングを提供しています。『JEUS サーバガイド』の「6.11. グローバル・トランザクション(XA)とコネクションの共有」を参照してください。
- 

## 2.3. コネクション・プールの設定例

アウトバウンド・リソース・アダプターを使用して、1つまたはそれ以上のアウトバウンド・コネクションを設定することができます。

## 2.3.1. コネクション・ファクトリーが1つの場合

以下は、javax.resource.cci.ConnectionFactoryというコネクション・ファクトリーを1つ持つra.xmlの例です。  
ほとんどの場合、コネクション・ファクトリーは1つです。

**[例 2.3] コネクション・ファクトリーが1つの場合のコネクション・プールの設定 : <<ra.xml>>**

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/j2ee" version="1.5"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd">
  <display-name>ConnectionManagementSample1</display-name>
  <vendor-name>TmaxSoft</vendor-name>
  <eis-type>TestResource</eis-type>
  <resourceadapter-version>2.0</resourceadapter-version>
  <license>
    <license-required>false</license-required>
  </license>
  <resourceadapter>
    <outbound-resourceadapter>
      <connection-definition>
        <managedconnectionfactory-class>
          com.tmax.SampleManagedConnectionFactory
        </managedconnectionfactory-class>
        <connectionfactory-interface>
          javax.resource.cci.ConnectionFactory
        </connectionfactory-interface>
        <connectionfactory-impl-class>
          com.tmax.SampleConnectionFactory
        </connectionfactory-impl-class>
        <connection-interface>
          javax.resource.cci.Connection
        </connection-interface>
        <connection-impl-class>
          com.tmax.SampleConnection
        </connection-impl-class>
      </connection-definition>
      <transaction-support>
        LocalTransaction
      </transaction-support>
      <authentication-mechanism>
        <authentication-mechanism-type>
          BasicPassword
        </authentication-mechanism-type>
        <credential-interface>
          javax.resource.spi.security.PasswordCredential
        </credential-interface>
      </authentication-mechanism>
    </outbound-resourceadapter>
  </resourceadapter>
</connector>
```

```

        </credential-interface>
    </authentication-mechanism>
    <reauthentication-support>
        false
    </reauthentication-support>
</outbound-resourceadapter>
</resourceadapter>
</connector>

```

jeus-connector-dd.xmlを設定しないと、ra.xmlの<connection-definition>に該当するコネクション・プールが生成できません。そのため、jeus-connector-dd.xmlの<export-name>を必ず設定します。

以下は、jeus-connector-dd.xmlの設定例です。各タグについては、「[2.1.4. コネクション・プールの設定](#)」を参照してください。

**[例 2.4] コネクション・ファクトリーが1つの場合のコネクション・プールの設定 : <<jeus-connector-dd.xml>>**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <connection-pool>
        <export-name>jcapool</export-name>
        <pool-management>
            <min>10</min>
            <max>50</max>
            <period>600000</period> <!-- 10 minutes -->
            <wait-connection>
                <wait-connection>true</wait-connection>
                <wait-timeout>30000</wait-timeout>
            </wait-connection>
            <action-on-connection-leak>Warning</action-on-connection-leak>
        </pool-management>
    </connection-pool>
</jeus-connector-dd>

```

## 2.3.2. コネクション・ファクトリーが2つ以上の場合

以下は、javax.sql.DataSourceとjavax.resource.cci.ConnectionFactoryをコネクション・ファクトリーとして設定したra.xml例です。

**[例 2.5] コネクション・ファクトリーが2つの場合のコネクション・プールの設定 : <<ra.xml>>**

```

<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/j2ee" version="1.5"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

```

```

    http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd">
<display-name>ConnectionManagementSample1</display-name>
<vendor-name>TmaxSoft</vendor-name>
<eis-type>TestResource</eis-type>
<resourceadapter-version>2.0</resourceadapter-version>
<license>
    <license-required>false</license-required>
</license>
<resourceadapter>
    <outbound-resourceadapter>
        <connection-definition>
            <managedconnectionfactory-class>
                com.tmax.DataSourceManagedConnectionFactory
            </managedconnectionfactory-class>
            <connectionfactory-interface>
                javax.sql.DataSource
            </connectionfactory-interface>
            <connectionfactory-impl-class>
                com.tmax.JeusDataSource
            </connectionfactory-impl-class>
            <connection-interface>
                java.sql.Connection
            </connection-interface>
            <connection-impl-class>
                com.tmax.JeusConnection
            </connection-impl-class>
        </connection-definition>
        <connection-definition>
            <managedconnectionfactory-class>
                com.tmax.SampleManagedConnectionFactory
            </managedconnectionfactory-class>
            <connectionfactory-interface>
                javax.resource.cci.ConnectionFactory
            </connectionfactory-interface>
            <connectionfactory-impl-class>
                com.tmax.SampleConnectionFactory
            </connectionfactory-impl-class>
            <connection-interface>
                javax.resource.cci.Connection
            </connection-interface>
            <connection-impl-class>
                com.tmax.SampleConnection
            </connection-impl-class>
        </connection-definition>
        <transaction-support>
            NoTransaction
        </transaction-support>
    </outbound-resourceadapter>
</resourceadapter>

```

```

        <authentication-mechanism>
            <authentication-mechanism-type>
                BasicPassword
            </authentication-mechanism-type>
            <credential-interface>
                javax.resource.spi.security.PasswordCredential
            </credential-interface>
        </authentication-mechanism>
        <reauthentication-support>
            false
        </reauthentication-support>
    </outbound-resourceadapter>
</resourceadapter>
</connector>

```

以下とおり、jeus-connector-dd.xmlでra.xmlの<connection-definition>に該当するコネクション・プールを生成するには、必ず**<connectionfactory-interface>**を設定します。設定しないと、デプロイする際にコネクション・プールが生成できないというエラーが発生します。

**[例 2.6] コネクション・ファクトリーが2つの場合のコネクション・プールの設定 : <<jeus-connector-dd.xml>>**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <connection-pool>
        <export-name>jdbcpool</export-name>
        <connectionfactory-interface>
            javax.sql.DataSource
        </connectionfactory-interface>
        <pool-management>
            <min>5</min>
            <max>20</max>
        </pool-management>
    </connection-pool>
    <connection-pool>
        <export-name>ccipool</export-name>
        <connectionfactory-interface>
            javax.resource.cci.ConnectionFactory
        </connectionfactory-interface>
        <pool-management>
            <min>1</min>
            <max>10</max>
        </pool-management>
    </connection-pool>
</jeus-connector-dd>

```

## 2.4. コネクション・プールのモニタリングおよび制御

JCAコネクション・プールのモニタリングおよび制御は、コンソール・ツール(jeusadmin)またはWebAdminを使用します。JCAコネクション・プールのモニタリングおよび制御方法を確認するには、リソース・アダプター・モジュールをデプロイする必要があります。

本節の例では、JCAコネクション・プールのJNDIエクスポート名が「my\_rar\_cp」であるリソース・アダプター・モジュールがデプロイされていると想定します。JCAコネクション・プールのJNDIエクスポート名の設定については、「[2.1.4. コネクション・プールの設定](#)」を参照してください。リソース・アダプター・モジュールのデプロイ方法については、『JEUS アプリケーション&デプロイメントガイド』を参照してください。

### 参考

本節では、WebAdminを使用したモニタリングおよび制御方法について説明します。コンソール・ツールを使用したモニタリングおよび制御方法については、『JEUS リファレンスガイド』の「4.2.12. コネクション・プールのモニタリングおよび制御コマンド」を参照してください。

### 2.4.1. コネクション・プールの制御

以下は、WebAdminを使用してコネクション・プールを制御する方法についての説明です。

1. WebAdminの左側のメニューから[Monitoring] > [Connection Pools]を選択すると、コネクション・プール画面が表示されます。画面にてリソース・アダプター・モジュールをデプロイしたサーバーを選択すると、リソース・アダプター・モジュールに設定したコネクション・プールが照会されます。

【図 2.1】 JCAコネクション・プールの画面



jeus\_domain

Domain

Session

Clusters

Servers

Applications

Security

Resources

Monitoring

Thread

Transaction

MBean

### Connection Pool

HISTORY

サーバに作成されたJDBC/JCAコネクションプールを制御し、ランタイム状態を照会します。

adminServer

server1

**The connection pool information on the server [server1].**

Connection Pool ID	JNDI Export Name	Min	Max	Active	Idle	Disposable	Total	Wait	Enabled
my_rar_cp *	my_rar_cp	2	30	0	0	0	0	false	true

\* : has not been created, total = active + idle + disposable

stmt



2. コネクション・プールをクリックすると、コネクション・プールの詳細情報や制御ボタンが表示されます。コネクション・プールが生成されていない場合は、[create]をクリックしてコネクション・プールを生成することができます。

[図 2.2] JCAコネクション・プールの生成

jeus\_domain

Domain

Session

Clusters

Servers

Applications

Security

Resources

Monitoring

Thread

Transaction

MBean

JNDI

Web

Servers

JMS

Connection Pool

EJB Timer

System Info

Connection Pool

サーバに作成されたJDBC/JCAコネクションプールを制御し、ランタイム状態を照会します。

adminServer

server1

Information about connections in the server [server1]'s connection pool [my\_rar\_cp].

Connection ID	State	State Time(sec)	Use Count	Type
該当する内容が存在しません。				

Enable Shrink Disable Refresh Create

The connection pool information on the server [server1].

Connection Pool ID	JNDI Export Name	Min	Max	Active	Idle	Disposable	Total	Wait	Enabled
my_rar_cp *	my_rar_cp	2	30	0	0	0	0	false	true

\* : has not been created, total = active + idle + disposable

以下は、コネクション・プールのボタンについての説明です。

ボタン	説明
[enable]	コネクション・プールを有効にします
[disable]	コネクション・プールを無効にします
[shrink]	コネクション・プールのコネクション数を最小値に調整します
[refresh]	コネクション・プールのコネクションを新しいコネクションに取り替えます

3. **[create]**をクリックすると、以下のようにコネクション・プールの生成が確認できます。

**[図 2.3] JCAコネクション・プールの生成確認**

**jeus\_domain**

Domain  
Session  
Clusters  
Servers  
Applications  
Security  
Resources  
Monitoring

Thread  
Transaction  
MBean  
JNDI  
Web  
Servers  
JMS  
Connection Pool  
EJB Timer  
System Info  
Server Log  
Statistic  
Patch Info

### Connection Pool

サーバに作成されたJDBC/JCAコネクションプールを制御し、ランタイム状態を照会します。

adminServer  
server1

**Information about connections in the server [server1]'s connection pool [my\_rar\_cp].**

Connection ID	State	State Time(sec)	Use Count	Type
my_rar_cp-2	idle	9.083	0	pooled
my_rar_cp-1	idle	9.196	0	pooled

Enable Shrink Disable Refresh Create

**The connection pool information on the server [server1].**

Connection Pool ID	JNDI Export Name	Min	Max	Active	Idle	Disposable	Total	Wait	Enabled
my_rar_cp	my_rar_cp	2	30	0	2	0	2	false	true

\* : has not been created, total = active + idle + disposable

## 2.4.2. コネクション・プールのモニタリング

以下は、WebAdminを使用してコネクション・プールのモニタリングする方法についての説明です。

- WebAdminの左側のメニューから**[Monitoring] > [Connection Pools]**を選択すると、コネクション・プール画面が表示されます。画面にてリソース・アダプター・モジュールをデプロイしたサーバーを選択すると、リソース・アダプター・モジュールに設定したコネクション・プールが照会されます。

**[図 2.4] JCAコネクション・プールのモニタリング**

**jeus\_domain**

Domain  
Session  
Clusters  
Servers  
Applications  
Security  
Resources  
Monitoring

Thread  
Transaction  
MBean  
JNDI

### Connection Pool

サーバに作成されたJDBC/JCAコネクションプールを制御し、ランタイム状態を照会します。

adminServer  
server1

**The connection pool information on the server [server1].**

Connection Pool ID	JNDI Export Name	Min	Max	Active	Idle	Disposable	Total	Wait	Enabled
my_rar_cp	my_rar_cp	2	30	0	2	0	2	false	true

\* : has not been created, total = active + idle + disposable

以下は、コネクション・プール・リストの項目についての説明です。リストのコネクション・プールはモニタリング情報が一緒に表示されます。

項目	説明
Connection Pool ID	JCAコネクション・プールの場合、JNDIエクスポート名と同じです
JNDI Export Name	コネクション・プールのJNDIリポジトリの登録名です
Min	コネクション・プールのコネクションの最小値です
Max	コネクション・プールのコネクションの最大値です
Active	使用中のコネクション数です
Idle	使用可能なアイドル状態のコネクション数です
Disposable	コネクション・プールにアイドル状態のコネクションが存在しない場合に生成する使い捨てコネクション数です。使い捨てコネクションはWait列がfalseの場合のみ生成されます
Total	Active、Idle、Disposableの合計です
Wait	コネクション・プールに使用可能なアイドル状態のコネクションが存在しない場合、使い捨てコネクションを生成するか否かです
Enabled	コネクション・プールの有効有無です

2. 特定のコネクション・プールをクリックすると、当該コネクション・プールに存在するコネクションの詳細情報が確認できます。

**[図 2.5] JCAコネクション・プールに存在するコネクションの詳細情報確認**

The screenshot shows the JBoss Monitoring console interface. On the left is a navigation menu with 'jeus\_domain' at the top and various system components listed below, including 'Monitoring' which is expanded. The main content area is titled 'Connection Pool' and shows the configuration for 'my\_rar\_cp' on 'server1'. It includes a table of active connections and a summary table of pool statistics.

**Information about connections in the server [server1]'s connection pool [my\_rar\_cp].**

Connection ID	State	State Time(sec)	Use Count	Type
my_rar_cp-2	idle	68.552	0	pooled
my_rar_cp-1	idle	68.665	0	pooled

**The connection pool information on the server [server1].**

Connection Pool ID	JNDI Export Name	Min	Max	Active	Idle	Disposable	Total	Wait	Enabled
my_rar_cp	my_rar_cp	2	30	0	2	0	2	false	true

■ \* : has not been created, total = active + idle + disposable

以下は、コネクション・プールの詳細情報の項目についての説明です。

項目	説明
Connection ID	コネクション識別子です
State	コネクションの状態です <ul style="list-style-type: none"><li>– idle : 使用可能な状態です</li><li>– active : 使用中の状態です</li></ul>
State Time	ステートの持続時間です
Use Count	コネクションの使用回数です
Type	使い捨てコネクションであるか否です <ul style="list-style-type: none"><li>– disposable : 使い捨てコネクションの場合です</li><li>– pooled : 使い捨てコネクションではない場合です</li></ul>

## 第3章 インバウンド管理

本章では、EISからアプリケーションに渡るインバウンド・コミュニケーションにおけるJEUSの役割や機能について説明します。主に、ワーク・マネージャー、Message Driven Beans(MDB)とリソース・アダプター間の連動について説明します。

### 3.1. ワーク・マネージャーの管理

本節では、ワーク・マネージャーの基本概念と設定方法について説明します。

#### 3.1.1. 基本概念

WASからバックグラウンドでジョブワークを実行するか、別のアプリケーションに情報を渡したいとき、リソース・アダプターでJavaスレッドを生成することになります。しかし、リソース・アダプター内で直接Javaスレッドが生成されることは、WASの観点からは好ましいことではありません。そのため、リソース・アダプターで特定のジョブワークを実行したい場合は、それをWASに渡して直接的なスレッドの管理はWASが代行に行うということがワーク・マネージャーの基本概念です。このときのジョブワークは、`javax.resource.spi.work.Work`のインターフェースを実装したインスタンスで表します。

JEUSでは、スレッド・プールをベースにしてワーク・マネージャーを提供します。スレッド・プールはリソース・アダプターがワーク・マネージャーを実際に使用するときには作られるため、`javax.resource.spi.BootstrapContext`を利用して常に有効なワーク・マネージャー・インスタンスを渡します。

JEUSのワーク・マネージャー設定はスレッド・プール・スタイルになっており、何も設定しなかった場合は、`jeus-connector-dd.xsd`スキーマに定義されているデフォルト値で生成されます。

---

#### 参考

ワーク・マネージャーおよびワークについての詳細は、JCA標準1.7の「10. Work Management」を参照してください。

---

JDKスレッド・プールは、最小値(JDKではコアサイズという)の分だけスレッドを増やし、以降はすべてキューに蓄積します。キューがフルになったらスレッドを最大値まで増やす方式です。

JEUSのスレッド・プールは、JDKで提供されるスレッド・プール(`java.util.concurrent.ThreadPoolExecutor`)の動作と類似しています。しかし、JEUSのスレッド・プールはスレッドを増やす条件をJDK方式より緩和しており、ジョブワークの量に合わせてスレッドを適切に増やす方式を取っています。したがって、リソース・アダ

プターがワーク・マネージャーを頻繁に使用する場合は、最小値、最大値以外にも<keep-alive-time>と<queue-size>を適切に調整する必要があります。

### 3.1.2. ワーク・マネージャーの設定

ワーク・マネージャーは内部的にスレッド・プールを使用するため、スレッド・プールの設定と類似しています。ワーク・マネージャーはリソース・アダプターに含まれるjeus-connector-dd.xmlの <worker-pool>に設定します。

以下は、ワーク・マネージャーの設定例です。

#### [例 3.1] ワーク・マネージャーの設定 : <<jeus-connector-dd.xml>>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <worker-pool>
    <min>0</min>
    <max>5</max>
    <keep-alive-time>60000</keep-alive-time>
    <shutdown-timeout>-1</shutdown-timeout>
  </worker-pool>
</jeus-connector-dd>
```

以下は、<worker-pool>のタグについての説明です。

タグ	説明
<min>	ワーク・マネージャーが管理するスレッド数の最小値です(デフォルト値: 0)
<max>	ワーク・マネージャーが管理するスレッド数の最大値です(デフォルト値: 5)
<keep-alive-time>	最小値以外のスレッドの場合、設定された時間の間に使用されないと自動的にスレッド・プールから削除されます。(デフォルト値: 1分)  <pooled-timeout>を代替する設定です
<queue-size>	スレッド・プールで必要としているキューのサイズを指定します(デフォルト値: 4096)
<pre-allocation>	ワーク・マネージャーが初期化されるときに<min>値に設定されている数のスレッドを事前に作成しておきます (デフォルト値: true)
<shutdown-timeout>	リソース・アダプターをアンデプロイしながらワーク・マネージャーが終了されるとき、設定された時間の間待機してから終了します。待機する間は新しい要求は受け付けません。すなわち、グレースフル・シャットダウンをサポートします(デフォルト値 : -1、待機せずに終了します)

---

#### 参考

jeus-connector-dd.xmlをRARに含ませる方法については、「[第4章 リソース・アダプター](#)」を参照してください。

---

## 3.2. メッセージ・インフロー

JCA標準に従って、リソース・アダプターからJEUSにデプロイされたアプリケーションにインバウンド・コミュニケーションを行うには、MDBを実装する必要があります。なお、MDBを利用して別のEJBコンポーネントを呼び出すことをお勧めしています。

[\[図 1.2\]](#)によると、WASからアプリケーションに渡るフローにおいて、MDBが第1次ゲートウェイの役割をしています。

本節では、JEUSでMDBとリソース・アダプターを連動する方法について説明します。

---

#### 参考

メッセージ・インフローの詳細内容については、JCA標準1.7の「13. Message Inflow」、または関連する書籍を参照してください。

---

以下は、MDBの例です。

```
@MessageDriven(
    activationConfig =
    {
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue")
        , @ActivationConfigProperty(propertyName = "DestinationProperties",
            propertyValue = "imqDestinationName=Queue")
        , @ActivationConfigProperty(propertyName = "ProviderIntegrationMode",
            propertyValue = "jndi")
        , @ActivationConfigProperty(propertyName = "ConnectionFactoryJndiName",
            propertyValue = "XAConnectionFactory")
        , @ActivationConfigProperty(propertyName = "DestinationJndiName",
            propertyValue = "jms/QUEUE1")
    }
)

public class TestMsgBean implements javax.jms.MessageListener {
    ...

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void onMessage(Message message) {
```

```
...
}
}
```

EJB 3.0からはejb-jar.xmlの代わりにアノテーションで設定することができます。MDBでは、**@ActivationConfigProperty**を使用してリソース・アダプターが要求するプロパティを必ず設定する必要があります。これについては、リソース・アダプターが提供するマニュアルを参照してください。

MDBと連動するリソース・アダプターをjeus-ejb-dd.xmlの<mdb-resource-adapter-name>に設定します。

**[例 3.2] MDB連動リソースの設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
<beanlist>
  <jeus-bean>
    <ejb-name>TestMsgBean</ejb-name>
    <connection-factory-name>QueueConnectionFactory</connection-factory-name>

    <destination>jms/QUEUE1</destination>
    <mdb-resource-adapter-name>app#ra</mdb-resource-adapter-name>
    . . .
  </jeus-bean>
. . .
```

タグ	説明
<mdb-resource-adapter-name>	連動するリソース・アダプターを指定します。そのとき、リソース・アダプターがスタンドアローン・モードか、EARに含まれているかによってリソース・アダプターの名前が異なってきます  – スタンドアローン・モジュールの場合: 該当するモジュール名  – EARに含まれたモジュールの場合: EAR名 + 「#」 + 該当するモジュール名  上記の例は、リソース・アダプターがEARに属するモジュールの場合です。EARの名前が「app」、リソース・アダプター・モジュールの名前が「ra」です

**注**

MDBをデプロイするとき、リソース・アダプターが事前にデプロイされている必要があります。



## 第4章 リソース・アダプター

本章では、JEUSで提供されるセキュリティ管理機能やリソース・アダプターにjeus-connector-dd.xmlを追加する方法およびデプロイ時の注意事項について説明します。

---

### 参考

実際のリソース・アダプターのパッケージングについては、JCA標準1.7の「20 Packaging Requirements」を参照してください。

---

## 4.1. セキュリティ管理

本節では、JEUSが提供しているリソース・アダプターの認証および権限チェック機能について説明します。

### 4.1.1. コネクション認証

JCA標準に明示されているようにejb-jar.xml, web.xmlなどに記述されている内容をベースにして、コネクションの認証者を判別します。

```
<resource-ref>
  <res-ref-name>jca/pool</res-ref-name>
  <res-type>javax.resource.cci.ConnectionFactory</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
  <res-auth>Container</res-auth>
</resource-ref>
```

- <res-auth>

コンテナまたはアプリケーションのうちアプリケーション・コンポーネント別の認証者を設定します。

- Container(デフォルト値)

コンテナによるコネクション認証として<res-auth>を「Container」に設定した場合は、jeus-connector-dd.xmlにユーザー名とパスワードを設定します。そのとき、パスワードには暗号化された値が使用できます。

以下は、パスワードの設定例です。

```
{DES}FQrLbQ/D8O1lDVS71L28rw==
```

このように設定したユーザー名とパスワード情報は、新たにコネクションを生成するときにリソース・アダプターへ渡す認証情報として使用されます。ユーザーがjeus-connector-dd.xmlに何の認証情報も設定しなかった場合は、空のjavax.security.auth.Subjectオブジェクトをリソース・アダプターに渡します。

---

#### 参考

パスワードの暗号化についての詳細は、『JEUS ドメインガイド』の「第7章 セキュリティー管理」を参照してください。

---

#### – Application

<res-auth>を「Application」に設定した場合は、アプリケーションがコネクションをリクエストするとき、JEUSはコネクション認証には関わりません。その代わりに、アプリケーションとリソース・アダプター間に認証情報を共有することになります。同情報は、javax.resource.spi.ConnectionRequestInfoを実装したリソース・アダプター・クラスを利用します。

## 4.2. パッケージング

JEUSにリソース・アダプターをデプロイするには、ra.xml以外にもJEUSが必要とする別途のDD(Deployment Descriptor)を使用してjeus-connector-dd.xmlファイルを作成する必要があります。同ファイルには、以下の内容が設定されます。

- ワーク・マネージャーの設定: [「3.1.2. ワーク・マネージャーの設定」](#)
- アウトバインド・コネクション・プールの設定: [「2.1.4. コネクション・プールの設定」](#)

設定した後、RARファイルのMETA-INFディレクトリーにjeus-connector-dd.xmlを格納します。

```
xxx.rar/META-INF
```

## 4.3. デプロイ

リソース・アダプターは、以下のとおり2つの形式でデプロイできます。

- スタンドアローン・モジュール：JEUSのすべてのアプリケーションで使用できます。
- Java EEアプリケーション(EAR)に属するモジュール：EAR内でのみ使用できます。

---

#### 参考

JEUSでアプリケーションをデプロイする方法の詳細は、『JEUS アプリケーション&デプロイメントガイド』を参照してください。

---

### 4.3.1. SHAREDモードのクラス・ローディング

JCA標準上、スタンドアローン・モジュールでデプロイするリソース・アダプターはすべてのアプリケーションが使用できる必要があるため、JEUSではSHAREDモードのクラス・ローディング方式をサポートしています。そのため、リソース・アダプターはユーザーの設定を問わず常にSHAREDモードでデプロイされます。その際、リソース・アダプターを使用するアプリケーションも、必ずSHAREDモードでデプロイします。

---

#### 参考

1. SHAREDモードのクラス・ローディング方式についての詳細は、『JEUS サーバガイド』の「1.4. クラス・ローダーの構造」を参照してください。
  2. デプロイ方法についての詳細は、『JEUS アプリケーション&デプロイメントガイド』を参照してください。
- 

### 4.3.2. 再デプロイ

リソース・アダプターは、一種のJDBCドライバーとしてJEUSに登録されます。JDBCドライバーは、JEUS\_HOME/lib/datasourceの下位にJARファイルをおくとサーバーのクラス・パスとして登録される方式です。そのため、途中でJARファイルをチェンジしても正常に反映できず、JEUSをシャットダウンする必要があります。

しかし、リソース・アダプターはJEUSが管理するアプリケーションなので、JEUSをシャットダウンせずにリソース・アダプターのバージョン・アップや再デプロイが可能です。

以下は、再デプロイ時の制限事項です。

- リソース・アダプター・モジュールを再デプロイするときは、それを使用するアプリケーションをすべて再デプロイする必要があります。

既存のリソース・アダプターを使用していたアプリケーションはすでにクラスをキャッシュしているため、再デプロイしたリソース・アダプターのクラスを検索しません。

- 現在JEUSでは、SHAREDモードでデプロイしたEJBモジュールを再デプロイする場合、それを使用していたWebモジュールもすべて自動で再デプロイしています。しかし、リソース・アダプター・モジュールに対しては、まだ自動再デプロイはサポートしていないため、手動でデプロイする必要があります。

## 4.4. リソース・アダプターをリソースとして登録

リソース・アダプター・モジュールは、1つの独立したアプリケーションというより、すべてのアプリケーションが共有して使用するドライバーのような概念です。そのような観点から、JEUSはドメインにリソース・アダプター・モジュールをコネクター・リソースとして登録して使用できる機能を提供します。

ドメインにリソース・アダプター・モジュールをコネクター・リソースとして登録する作業はWebAdminを使用し  
て行えます。その際、ドメインに登録されるリソース・アダプターの設定情報は、リソース・アダプター・モジュールのjeus-connector-dd.xmlの設定情報とほぼ同様な形式です。

リソース・アダプターの設定は、以下のような優先順位が適用されデプロイされます。

#### 1. WebAdminの設定

jeus-connector-dd.xml設定より優先順位が高いです。同設定が存在する場合、リソース・アダプターは  
デプロイの後、ドメインでコネクター・リソースとして使用できます。

#### 2. jeus-connector-dd.xmlの設定

WebAdminを使用したリソース・アダプターの設定が存在しない場合に適用されます。リソース・アダプター  
は、モジュール本来の役割のみ実行します。

### 4.4.1. WebAdminの使用

以下は、WebAdminを使用してリソース・アダプターをコネクター・リソースとしてドメインに登録する手順で  
す。

1. WebAdminの左側のメニューから**[Resources] > [External Source]**を選択すると、**[External Source]**  
画面が表示されます。**[LOCK & EDIT]**ボタンをクリックした後、**Connector**リストから**[Add]**ボタンをクリッ  
クすると、コネクター・リソースの設定画面に移ります。
2. コネクション・リソースの設定は、リソース・アダプターIDを基本として、ワーカー・プールとコネクション・プー  
ルの設定で構成されています。「**Resource Adapter Module Id**」にコネクター・リソースとして登録する  
リソース・アダプター・モジュールのIDを入力し、必要に応じて「**Worker Pool**」を調整した後、**[確認]**ボタ  
ンをクリックします。
3. コネクション・プールの設定が必要な場合は、コネクター・リソースとして追加したリソース・アダプターのID  
をクリックします。
4. 再びコネクター・リソースの画面設定に戻り、「**Connection Pool**」リソースの**[Add]**ボタンをクリックする  
と、コネクション・プールの設定を追加できます。
5. 必要なコネクション・プールの設定値を入力した後、**[確認]**ボタンをクリックします。
6. コネクター・リソースのドメイン登録を最終適用するため、**[Activate Changes]**ボタンをクリックします。
7. サーバーに登録された情報の適用が完了すると、結果メッセージが出力されます。

メッセージが出力されたらコネクタ・リソースの登録作業は完了されますが、同作業は動的に反映できないため、登録したコネクタ・リソースを使用するにはサーバーを再起動し、リソース・アダプター・モジュールを再デプロイする必要があります。



# 付録 A. 環境設定の注意事項

jeus-connector-dd.xmlを作成するときの必須考慮事項は以下のとおりです。

- **ra.xmlに<outbound-resourceadapter><connection-definition>の存在有無**

リソース・アダプターはインバウンド目的でのみ使用できるため、アウトバウンド設定は存在しない場合があります。そのときは、jeus-connector-dd.xmlを作成しなくても構いません。ただし、<connection-definition>が存在する場合は、それに合わせてコネクション・プールを生成する必要があるので、必ず jeus-connector-dd.xmlまたはdomain.xmlのexternal-sourceにコネクタを設定してください。

- **ra.xmlに<outbound-resourceadapter><connection-definition>が2つ以上存在する場合**

<connection-definition>が2つ以上設定された場合は、それに合わせて最小1つ以上のコネクション・プールを生成します。そのときの注意点は、それぞれの<connection-definition>設定に合わせて、<connectionfactory-interface>値をjeus-connector-dd.xmlまたはdomain.xmlのexternal-sourceにコネクタを設定する必要があります。詳しい内容は、「[2.3. コネクション・プールの設定例](#)」を参照してください。

コネクション・プールを生成しない場合は、ra.xmlを調整します。

- **ワーク・マネージャー設定を調整したい場合**

jeus-connector-dd.xmlを設定しなくても基本的にはリソース・アダプターにワーク・マネージャーを提供します。ワーク・マネージャーは、リソース・アダプターが必要なときに取得して使用するものなので、JEUSでは最初からこれを初期化しません。

その代わり、実際にリクエストするときにワーク・マネージャーを生成します。ワーク・マネージャーをスレッド・プール設定ともいえますので、スレッド数が調整したい場合にはjeus-connector-dd.xmlを作成してワーク・マネージャーを設定します。ワーク・マネージャーの詳しい設定方法については、「[3.1.2. ワーク・マネージャーの設定](#)」を参照してください。





# 索引

## シンボル

<connection-pool>  
    <connectionfactory-interface>, 10  
    <export-name>, 11  
    <password>, 11  
    <pool-management>, 11  
    <property>, 14  
    <transaction-support>, 11  
    <use-lazy-transaction-enlistment>, 11  
    <user>, 11  
<jeus-bean>  
    <mdb-resource-adapter-name>, 30  
<non-validation-interval>, 15  
<resource-ref>, 8, 17  
    <res-auth>, 31  
<validation-retrial-count>, 15  
<worker-pool>  
    <keep-alive-time>, 28  
    <max>, 28  
    <min>, 28  
    <pre-allocation>, 28  
    <queue-size>, 28  
    <shutdown-timeout>, 28

## A

AllConnections, 15

## C

CCI(Common Client Interface), 5  
Connector Architecture, 1

## D

Deploy  
    SHAREDモードのクラス・ローディング, 33

## E

ERP, 1

## F

FailedConnectionOnly, 15

## J

JCA, 1

## あ

アウトバウンド, 1  
インバウンド, 2

## か

コネクション・プール, 9  
コネクション・リーク, 15

## た

デプロイ  
    再デプロイ, 33

## わ

ワーク・マネージャー, 4

