

# JEUS EJB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 EJB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 .....	xv
<b>第1章 EJBの紹介 .....</b>	<b>1</b>
1.1. 概要 .....	1
1.2. 構成要素 .....	2
1.3. EJBの環境および設定 .....	3
1.3.1. ディレクトリーの構造 .....	4
1.3.2. XML設定ファイル .....	5
1.3.3. 関連ツール .....	7
1.4. EJBの基本設定 .....	7
<b>第2章 EJBエンジン .....</b>	<b>11</b>
2.1. 概要 .....	11
2.2. 主要機能 .....	11
2.3. EJBエンジンのディレクトリー構造 .....	12
2.4. EJBエンジンの設定 .....	14
2.4.1. Basic設定 .....	14
2.4.2. Active Management設定 .....	17
2.4.3. Timer Service設定 .....	20
2.5. システム・ログの設定 .....	20
2.6. EJBエンジンの制御およびモニタリング .....	20
2.7. EJBエンジンのチューニング .....	21
2.7.1. レゾリューション設定のチューニング .....	21
2.7.2. 高速デプロイ .....	22
2.7.3. 最大性能のためのシステム・ログの設定 .....	22
2.7.4. アクティブ管理の不使用 .....	22
2.7.5. HTTP呼び出しモードの使用 .....	23
<b>第3章 EJBモジュール .....</b>	<b>25</b>
3.1. 概要 .....	25
3.2. EJBモジュールの管理 .....	25
3.3. EJBモジュールの組み立て(Assembling) .....	28
3.3.1. EJBクラスのコンパイル .....	28
3.3.2. Deployment Descriptors(DD)の作成 .....	29
3.3.3. EJB JARファイルのパッケージング .....	32
3.4. EJBモジュールのデプロイ .....	33
3.4.1. デプロイ .....	33
3.4.2. デプロイされたEJBモジュールのディレクトリー構造 .....	38
3.5. EJBモジュールの制御およびモニタリング .....	39
3.5.1. EJBモジュールの制御 .....	39
3.5.2. EJBモジュールのモニタリング .....	43
<b>第4章 EJBの共通特性 .....</b>	<b>47</b>

4.1.	概要 .....	47
4.2.	EJBの設定 .....	49
4.2.1.	基本的な環境設定 .....	50
4.2.2.	スレッド・チケットの設定 .....	52
4.2.3.	外部参照の設定とマッピング .....	54
4.2.4.	HTTP呼び出し(Invoke)の環境設定 .....	57
4.2.5.	JEUS RMIの設定 .....	58
4.2.6.	EJBのセキュリティー設定 .....	58
4.3.	EJBのモニタリング .....	63
4.4.	EJBのチューニング .....	65
4.4.1.	TTP(Thread Ticket Pool)設定のチューニング .....	65
<b>第5章</b>	<b>EJBの相互運用性およびRMI/IIOP .....</b>	<b>67</b>
5.1.	概要 .....	67
5.1.1.	トランザクション相互運用(OTS) .....	68
5.1.2.	セキュリティー相互運用(CSiv2) .....	68
5.2.	相互運用の設定 .....	68
5.2.1.	COSネーミング・サービスの設定 .....	69
5.2.2.	相互運用性の有効設定 .....	69
5.2.3.	CSiv2セキュリティーの相互運用設定 .....	69
5.2.4.	EJB RMI/IIOPの設定 .....	70
5.3.	RMI/IIOPクライアント .....	71
5.3.1.	JEUS MS(Managed Server) .....	71
5.3.2.	他ベンダーのWAS .....	72
5.3.3.	スタンドアローン・クライアント .....	73
5.4.	既知の問題点(Known Issues) .....	73
<b>第6章</b>	<b>EJBのクラスタリング .....</b>	<b>75</b>
6.1.	概要 .....	75
6.2.	主要機能 .....	77
6.2.1.	ロードバランシング .....	77
6.2.2.	フェイルオーバー(EJBのフェイルオーバー) .....	78
6.2.3.	多重呼び出し不変メソッドを使用したEJBのフェイルオーバー .....	78
6.2.4.	セッション複製(Session Replication) .....	79
6.3.	EJBクラスタリングの設定 .....	79
6.3.1.	アノテーションを使用したクラスタリングの設定 .....	80
6.3.2.	xmlを使用したクラスタリングの設定 .....	82
6.3.3.	ステートフル・セッションBeanのクラスタリング設定 .....	84
6.4.	EJBフェイルオーバーの制限 .....	84
<b>第7章</b>	<b>セッションBean .....</b>	<b>87</b>
7.1.	ステートレス・セッションBean .....	87
7.1.1.	Thread Ticket Pool(TTP)とオブジェクト・マネージメント .....	87
7.1.2.	Webサービスのエンドポイント .....	88
7.2.	ステートフル・セッションBean .....	88

7.2.1.	Thread Ticket Pool(TTP)とオブジェクト・マネージメント .....	88
7.2.2.	プーリング・セッションBean .....	90
7.2.3.	Beanプールの設定 .....	90
7.2.4.	セッション・データ保持メカニズムの設定 .....	91
7.3.	共通設定 .....	91
7.3.1.	オブジェクト・マネージメント関連の設定 .....	91
<b>第8章</b>	<b>エンティティBean .....</b>	<b>95</b>
8.1.	概要 .....	95
8.2.	主要機能 .....	97
8.2.1.	共通機能 .....	97
8.2.2.	BMP & CMP 1.1 .....	102
8.2.3.	CMP 1.1/2.0 .....	102
8.2.4.	CMP 2.0 .....	108
8.3.	エンティティEJBの設定 .....	109
8.3.1.	共通設定 .....	110
8.3.2.	CMP 1.1/2.0 .....	115
8.3.3.	CMP 2.0 .....	122
8.3.4.	DB Insert Delayの設定(CMP Only) .....	130
8.4.	エンティティEJBのチューニング .....	131
8.4.1.	共通 .....	131
8.4.2.	BMP & CMP 1.1 .....	132
8.4.3.	CMP 1.1/2.0 .....	133
8.4.4.	CMP 2.0 .....	134
8.5.	CMP 2.0 エンティティBeanの完全な例 .....	134
<b>第9章</b>	<b>メッセージ駆動型Bean(MDB) .....</b>	<b>141</b>
9.1.	概要 .....	141
9.2.	MDBの設定 .....	141
9.2.1.	基本環境の設定 .....	141
9.2.2.	JMSの設定 .....	142
9.2.3.	JNDI SPIの環境設定 .....	147
<b>第10章</b>	<b>EJBタイマー・サービス .....</b>	<b>149</b>
10.1.	タイマー・サービスの設定 .....	149
10.1.1.	永続的なタイマー・サービスの設定(EJBエンジン) .....	149
10.1.2.	永続的なタイマーの処理(jeus-ejb-dd.xml) .....	151
10.1.3.	Cluster-Wide Timer Serviceの設定 .....	152
10.2.	タイマー・モニタリング .....	153
10.3.	タイマー・サービス使用時の注意事項 .....	155
10.3.1.	永続的なタイマーとJDBCコネクション .....	155
<b>第11章</b>	<b>EJBクライアント .....</b>	<b>157</b>
11.1.	概要 .....	157
11.2.	EJBアクセスのためのクライアント・プログラミング .....	157

11.3.	初期コンテキストの設定 .....	158
11.3.1.	JVM属性を使用したネーミング属性値の設定 .....	159
11.3.2.	ハッシュテーブルを利用したネーミング属性の設定 .....	160
<b>第12章</b>	<b>拡張機能 .....</b>	<b>163</b>
12.1.	作業領域(WorkArea)サービス .....	163
12.1.1.	UserWorkAreaインターフェース .....	163
12.1.2.	PropertyModeタイプ .....	164
12.1.3.	例外 .....	164
12.1.4.	ネストされたUserWorkArea .....	165
12.1.5.	UserWorkAreaを使用するアプリケーション・プログラムの開発 .....	165
<b>付録 A.</b>	<b>基本JavaタイプとDBフィールドのマッピング .....</b>	<b>171</b>
A.1.	概要 .....	171
A.2.	Tiberoのフィールドと列型のマッピング .....	171
A.3.	Oracleのフィールドと列型のマッピング .....	172
A.4.	Sybaseのフィールドと列型のマッピング .....	173
A.5.	MSSQLのフィールドと列型のマッピング .....	174
A.6.	DB2のフィールドと列型のマッピング .....	175
A.7.	Cloudscapeのフィールドと列型のマッピング .....	176
A.8.	Informixのフィールドと列型のマッピング .....	176
<b>付録 B.</b>	<b>Instant EJB QL APIリファレンス .....</b>	<b>179</b>
B.1.	概要 .....	179
B.2.	The EJBInstanceFinderインターフェース .....	179
B.3.	The EJBInstanceFinderメソッド .....	179
<b>用語集</b>	<b>.....</b>	<b>181</b>
<b>索引</b>	<b>.....</b>	<b>183</b>



# 目次

[図 1.1]	EJB実装の主な構成要素 .....	2
[図 1.2]	EJBエンジンの設定 .....	8
[図 2.1]	EJBエンジンのディレクトリー構造 .....	13
[図 2.2]	EJBエンジンの設定 - Basic設定 .....	15
[図 2.3]	EJBエンジンの設定 - Active Management設定 .....	18
[図 3.1]	EJBモジュールの管理手順 .....	26
[図 3.2]	Java EE EJBモジュールのJARファイルの構造 .....	27
[図 3.3]	アプリケーションのデプロイ - アプリケーション・リストの照会 .....	34
[図 3.4]	アプリケーションのデプロイ - 属性の設定 .....	35
[図 3.5]	アプリケーションのデプロイ - 結果 .....	37
[図 3.6]	デプロイされたEJBモジュールのディレクトリー構造 .....	38
[図 3.7]	デプロイされたアプリケーションのリスト .....	40
[図 3.8]	アプリケーションのアンデプロイ - 属性の設定 .....	41
[図 3.9]	アプリケーションのアンデプロイ - 結果 .....	41
[図 4.1]	TTPのプロセス .....	53
[図 5.1]	相互運用性の有効設定 .....	69
[図 5.2]	CSlv2セキュリティの相互運用設定 - Interop Ssl Config .....	70
[図 6.1]	EJB2クラスタリングのアーキテクチャー .....	76
[図 6.2]	EJBクラスタリングのアーキテクチャー .....	76
[図 7.1]	ステートレス・セッションBeanのTTPとBeanプール .....	87
[図 7.2]	ステートフル・セッションBeanのコネクション・プールとTTP、Beanプール .....	89
[図 8.1]	JEUS EJBエンジンでのエンティティBeanのオブジェクトとインスタンスの管理 .....	97
[図 8.2]	ejbLoad()の周期的な呼び出しシナリオ .....	99
[図 8.3]	EXCLUSIVE_ACCESSモードのシナリオ .....	100
[図 8.4]	SINGLE_OBJECTエンジン・モード .....	100
[図 8.5]	MULTIPLE_OBJECTエンジン・モード .....	101
[図 10.1]	永続的なタイマー・サービスの設定 - 基本設定 .....	150
[図 10.2]	永続的なタイマー・サービス - 詳細設定 .....	150
[図 10.3]	Cluster-Wide Timer Serviceの設定 .....	152
[図 10.4]	タイマー・モニタリング .....	153
[図 10.5]	タイマー動作のキャンセル .....	154
[図 12.1]	ネストされたUserWorkAreaでの登録情報 .....	165



## 表目次

[表 4.1]	JEUS EJBの設定可能な特性およびコンポーネント .....	48
[表 4.2]	ejb-jar.xmlとJEUS EJB DDファイルの参照タグとその関係 .....	54
[表 5.1]	KeystoreとTruystoreファイルに関連するJVMの-Dパラメータ .....	70
[表 8.1]	エンティティBeanの種類別の設定 .....	95
[表 8.2]	3つのエンジン・モードの長短所 .....	101
[表 8.3]	EmployeeBeanインスタンスのためのEJBフィールド .....	130
[表 8.4]	結果セット .....	130
[表 8.5]	エンティティBeanのエンジン・タイプの選択とクラスタリングの使用有無 .....	131
[表 A.1]	TiberoのためのEJB CMPフィールドとDB列型のマッピング .....	171
[表 A.2]	OracleのためのEJB CMPフィールドとDB列型のマッピング .....	172
[表 A.3]	SybaseのためのEJB CMPフィールドとDB列型のマッピング .....	173
[表 A.4]	MSSQLのためのEJB CMPフィールドとDB列型のマッピング .....	174
[表 A.5]	DB2のためのEJB CMPフィールドとDB列型のマッピング .....	175
[表 A.6]	CloudscapeのためのEJB CMPフィールドとDB列型のマッピング .....	176
[表 A.7]	InformixのためのEJB CMPフィールドとDB列型のマッピング .....	176



# 例目次

[例 1.1]	XMLヘッダー : <<domain.xml>> .....	6
[例 1.2]	XMLヘッダー : <<ejb-jar.xml >> .....	6
[例 1.3]	XMLヘッダー : <<jeus-ejb-dd .xml>> .....	7
[例 3.1]	EJB標準のDD : <<ejb-jar.xml>> .....	30
[例 3.2]	JEUS EJB DD : <<jeus-ejb-dd.xml>> .....	31
[例 4.1]	ステートフル・セッションBeanクラスとDD : <<CounterEJB.java>> .....	52
[例 4.2]	ステートフル・セッションBeanクラスとDD : <<jeus-ejb-dd.xml>> .....	52
[例 4.3]	BMP Beanの設定 : <<jeus-ejb-dd.xml>> .....	53
[例 4.4]	外部参照をJNDIにマッピング : <<CounterEJB.java>> .....	55
[例 4.5]	外部参照をJNDIにマッピング : <<ejb-jar.xml>> .....	55
[例 4.6]	外部参照をJNDIにマッピング : <<jeus-ejb-dd.xml>> .....	56
[例 4.7]	HTTP呼び出しの環境設定 : <<jeus-ejb-dd.xml>> .....	57
[例 4.8]	ロール割り当て設定 : <<jeus-ejb-dd.xml>> .....	59
[例 4.9]	Run-as Identify設定 : <<jeus-ejb-dd.xml>> .....	60
[例 4.10]	セキュリティー設定 : <<CustomerBean.java>> .....	60
[例 4.11]	セキュリティー設定 : <<EmployeeServiceBean.java>> .....	60
[例 4.12]	セキュリティー設定 : <<ejb-jar.xml>> .....	61
[例 4.13]	セキュリティー設定 : <<jeus-ejb-dd.xml>> .....	62
[例 4.14]	セキュリティー設定 : <<accounts.xml>> .....	62
[例 5.1]	EJB RMI/IIOPの設定 : <<jeus-ejb-dd.xml>> .....	71
[例 5.2]	corbanameルックアップの使用 .....	71
[例 5.3]	PROVIDER URLの使用 .....	72
[例 5.4]	Servlet EJB Injection .....	72
[例 5.5]	RMI/IIOP EJBのマッピング : <<jeus-web-dd.xml>> .....	72
[例 5.6]	スタンドアローン・クライアントの使用 .....	73
[例 5.7]	NullPointerExceptionが発生する場合 .....	74
[例 6.1]	アノテーションを使用したクラスタリングの設定 : <<CounterEJB.java>> .....	80
[例 6.2]	xmlを使用したクラスタリングの設定 : <<jeus-ejb-dd.xml>> .....	82
[例 7.1]	Beanプールの設定 : <<jeus-ejb-dd.xml>> .....	90
[例 7.2]	オブジェクト・マネージメントの設定 : <<jeus-ejb-dd.xml>> .....	91
[例 8.1]	Oracle DBでの主キー生成の設定 : <<jeus-ejb-dd.xml>> .....	105
[例 8.2]	MS SQLサーバーでの主キー自動生成の設定 : <<jeus-ejb-dd.xml>> .....	106
[例 8.3]	その他DBの主キー自動生成の設定 : <<jeus-ejb-dd.xml>> .....	107
[例 8.4]	エンティティEJBの基本かつ共通項目の設定 : <<jeus-ejb-dd.xml>> .....	110
[例 8.5]	オブジェクト・マネージメント関連の設定 : <<jeus-ejb-dd.xml>> .....	110
[例 8.6]	ejbLoad()とejbStore()永続性の最適化設定 : <<jeus-ejb-dd.xml>> .....	114
[例 8.7]	ejbLoad()とejbFind() CM永続性の最適化設定 : <<jeus-ejb-dd.xml>> .....	115
[例 8.8]	DBスキーマ情報の設定 : <<jeus-ejb-dd.xml>> .....	116
[例 8.9]	One-to-one/One-to-manyリレーションシップの設定 : <<jeus-ejb-dd.xml>> .....	122
[例 8.10]	Many-to-manyリレーションシップ・マッピングの設定 : <<jeus-ejb-dd.xml>> .....	123

[例 8.11]	インスタントEJB QLの設定 : <<jeus-ejb-dd.xml>> .....	124
[例 8.12]	XML DDファイルに「>」と「<」文字を挿入 : <<ejb-jar.xml>> .....	127
[例 8.13]	DB Insert Delayの設定 : <<jeus-ejb-dd.xml>> .....	130
[例 8.14]	リモート・インターフェース : <<Book.java>> .....	134
[例 8.15]	ホーム・インターフェース : <<BookHome.java>> .....	135
[例 8.16]	Beanの実装 : <<BookEJB.java>> .....	135
[例 8.17]	Java EE EJB DD : <<ejb-jar.xml>> .....	136
[例 8.18]	JEUS EJB DD : <<jeus-ejb-dd.xml>> .....	138
[例 9.1]	基本環境設定 : <<jeus-ejb-dd.xml>> .....	142
[例 9.2]	JMSの設定 : <<MyMDB.class>> .....	142
[例 9.3]	JMSの設定 : <<jeus-ejb-dd.xml>> .....	143
[例 9.4]	JNDI SPIの環境設定: <<jeus-ejb-dd.xml>> .....	147
[例 10.1]	永続的なタイマーの処理 : <<jeus-ejb-dd.xml>> .....	151
[例 11.1]	<<HelloClient.java>> .....	157
[例 12.1]	<<UserWorkAreaインターフェース>> .....	163
[例 12.2]	UserWorkAreaへのアクセス : <<UserWorkAreaSampleSenderBean.java>> .....	166
[例 12.3]	新しいUserWorkAreaの開始 : <<UserWorkAreaSampleSenderBean.java>> .....	166
[例 12.4]	作業領域に登録情報を設定 : <<UserWorkAreaSampleSenderBean.java>> .....	167
[例 12.5]	作業領域に設定された登録情報の取得 : <<UserWorkAreaSampleReceiverBean.java>> .	168
[例 12.6]	UserWorkAreaの終了 : <<UserWorkAreaSampleSenderBean.java>> .....	168

# このガイドについて

## 対象読者

本書は、JEUS<sup>®</sup>(以下、JEUS)のシステム管理者とJEUSでJava EE Enterprise JavaBeans(EJB)のデプロイを目的としている開発者を対象としています。

## 前提知識

本書ではEJB概念を始めとし、JEUSにおけるEJBエンジン、EJBモジュール、EJBクライアントに関する内容まで記述しています。本書を理解するには、事前に以下の内容を把握しておく必要があります。

- Java EE仕様についての知識

- EJBの基本概念

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

本書では、EJBに関する基本的な内容については記述していないので、別途の関連資料を参考にしてください。エンティティBeanは、EJB 3.0からはJPAに代替されました。『JEUS JPAガイド』を参照してください。

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。

# 本書の構成

本書は、計12章と2つの付録で構成されています。

- [「第1章 EJBの紹介」](#)

EJBの構成要素と特性および簡単なインストールおよび環境設定方法について説明します。

- [「第2章 EJBエンジン」](#)

EJBエンジンの概念とJEUS EJBのルート・レベルの概念構造、設定、モニタリング、チューニングについて説明します。

- [「第3章 EJBモジュール」](#)

EJBモジュールの構造と管理方法について説明します。

- [「第4章 EJBの共通特性」](#)

EJBの共通する特性について紹介します。

- [「第5章 EJBの相互運用性およびRMI/IIOP」](#)

EJBの相互運用性およびRMI/IIOPについて説明します。

- [「第6章 EJBのクラスタリング」](#)

EJBクラスタリングの概念と主要機能の設定方法について説明します。

- [「第7章 セッションBean」](#)

ステートレス・セッションBeanとステートフル・セッションBeanについて説明します。

- [「第8章 エンティティBean」](#)

EJBと設定、チューニングする際に必要な情報について説明します。

- [「第9章 メッセージ駆動型Bean\(MDB\)」](#)

EJBエンジンでMDBを使用する際の注意事項について説明します。

- [「第10章 EJBタイマー・サービス」](#)

JEUS EJBで提供するタイマー・サービスと、同サービスを使用するための設定について説明します。

- [「第11章 EJBクライアント」](#)

EJBクライアント・プログラミングの例と初期コンテキストの設定方法について説明します。



- [「第12章 拡張機能」](#)

JEUSでEJBを開発するときに使用できる拡張機能について説明します。

- [「付録 A. 基本JavaタイプとDBフィールドのマッピング」](#)

Javaのフィールド型とJEUSでサポートする主なDBベンダーのデータベース列型との基本マッピングについて説明します。

- [「付録 B. Instant EJB QL APIリファレンス」](#)

Instant EJB QL APIリファレンスでメソッドとインターフェースを使用する方法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
<b>参考</b>	参照/注意事項
<b>注</b>	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 JPAガイド	JEUSに統合されたTopLink Essentialの使用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 アプリケーションクライアントガイド	Java EEクライアントとJEUS間の相互運用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

- EJB 3.2 Specification
- XML Reference - domain.xmlの設定  
JEUS\_HOME/docs/manuals/xml-reference/index.html
- XML Reference - jeus-ejb-dd.xmlの設定  
JEUS\_HOME/docs/manuals/xml-reference/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 EJBの紹介

本章では、EJBの構成要素と特性および簡単なインストールと環境設定方法について説明します。

## 1.1. 概要

JEUSは、JSR 318 Enterprise JavaBeans™ 3.2(以下、EJB 3.2)をサポートしています。EJB 3.2標準(以下、EJB標準)には、複雑かつ重要な高性能ビジネス・ロジックを開発する際に、開発者の手間を最小限にしつつ、移植性の優れたビジネス・コンポーネントが作成できる構造が記述されています。

EJB 3.2におけるEJBの定義は以下のとおりです。

「EJBアーキテクチャーはオブジェクト指向の分散エンタープライズ・アプリケーションの開発および分散デプロイのためのコンポーネントです。EJBによって作成されたアプリケーションは拡張性、トランザクション処理、同時ユーザー処理における安全性が保証されます。EJB仕様に準拠しているアプリケーションは、一度作成されると、いかなるサーバー・プラットフォームにもデプロイすることができます。」

JEUSは、上述した「サーバー・プラットフォーム」の役割をします。

本書では、JEUSに実装されたEJBとエンタープライズ環境における必須拡張機能について説明します。

---

### 参考

より詳しい仕様の実装情報は、『JEUS 紹介ガイド』を参照してください。

---

JEUS EJBでは、クラスタリングを利用して性能や安定性を高めるため、標準以外に2つの追加機能を提供しています。

- 性能を改善するためのロード・バランシング(負荷分散)機能
- 安定性を拡張するためのフェイルオーバー機能

同機能を使用するには、2つ以上のManaged Server(以下、MS)がクラスターで構成されている必要があります。

---

### 参考

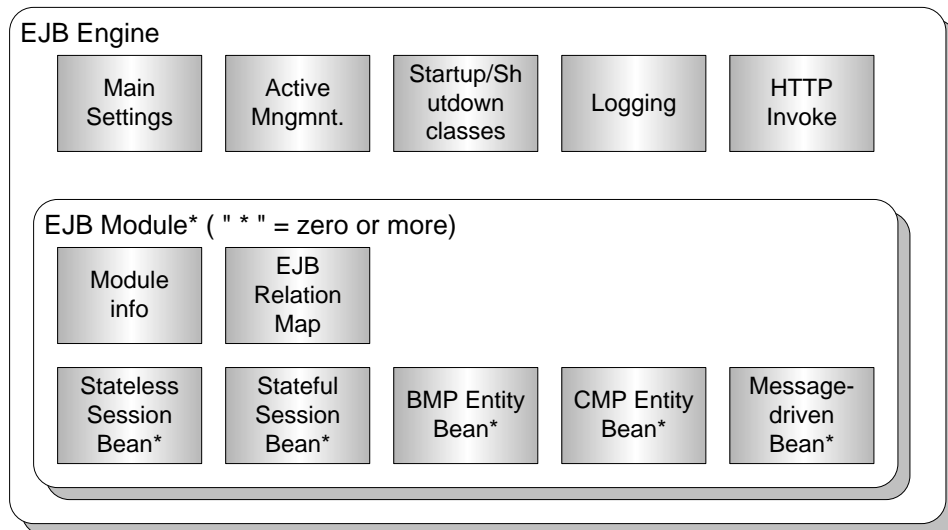
EJBクラスタリングについての詳細内容は、「[第6章 EJBのクラスタリング](#)」を、MSクラスタリングの詳細内容は、『JEUS ドメインガイド』の「[第5章 JEUSクラスタリング](#)」を参照してください。

---

## 1.2. 構成要素

以下は、EJB実装の主な構成要素です。

**[図 1.1] EJB実装の主な構成要素**



### ● EJBエンジン

EJB 3.2で説明するEJBコンテナに該当しており、デプロイされたEJBモジュールの実行環境を提供します。EJBエンジンについては、「[第2章 EJBエンジン](#)」で詳しく説明します。

### ● EJBモジュール

各EJBコンポーネントをグループ化して管理する単位です。JEUSではEJBコンポーネントをデプロイ、制御する際にEJBモジュールを基本単位としているため、EJBモジュールには必ず1つ以上のEJBコンポーネントが含まれています。EJBモジュールについての詳細内容は、「[第3章 EJBモジュール](#)」で説明します。

### ● EJBコンポーネント

EJBモジュール内に属しており、実際のビジネス・コンポーネントです。各EJBコンポーネントの共通した特性については、「[第4章 EJBの共通特性](#)」を参照してください。

以下は、EJBコンポーネントの5種類です。各種類に関する詳細内容は、それぞれの章を参照してください。

- ステートレス・セッションBean : 「[第7章 セッションBean](#)」
- ステートフル・セッションBean : 「[第7章 セッションBean](#)」
- BMPエンティティBean : 「[第8章 エンティティBean](#)」
- CMP 1.1 & CMP 2.x Entity Bean : 「[第8章 エンティティBean](#)」
- メッセージ駆動型Bean : 「[第9章 メッセージ駆動型Bean\(MDB\)](#)」

---

#### 参考

エンティティBeanは、EJB 3.0からJPAに代替されました。詳細内容については、『JEUS JPAガイド』を参照してください。

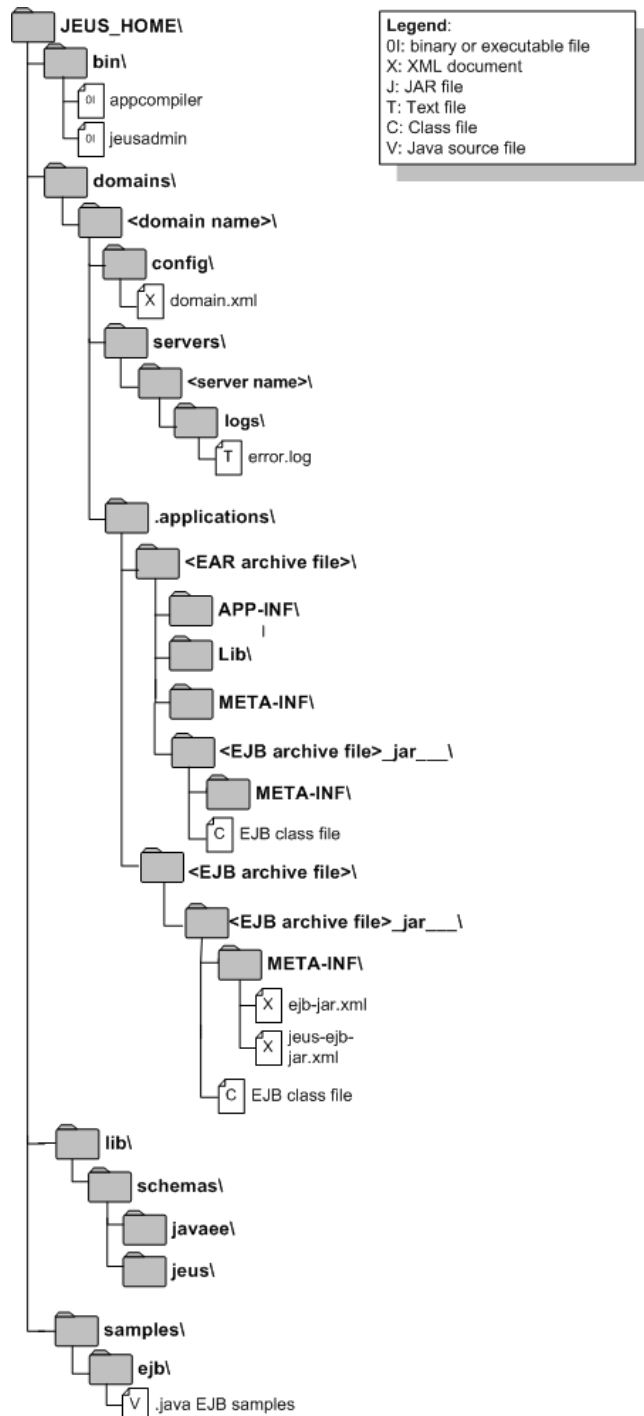
---

## 1.3. EJBの環境および設定

JEUS EJBの環境設定要素は、EJBに関連するディレクトリー構造、EJB設定ファイル、各種ツールなどで構成されます。さらに、EJBと関連するシステム・プロパティやその特性についても記述します。

## 1.3.1. ディレクトリーの構造

以下は、JEUS EJBのディレクトリーとファイルを示しています



bin

管理者がEJBを管理するときに使用できる実行スクリプトが保存されます。jeusadmin、appcompilerなどがあります。

domains/<domain name>/config

現在のJEUSシステムの主要設定ディレクトリーです。このディレクトリーにはdomain.xmlファイルが存在しています。

domains/<domain name>/servers/<server name>/logs

EJBエンジンのログ・ファイルが存在しています。EJBログのファイル・ハンドラーを別途指定した場合、別途のファイルで生成されます。ハンドラーが存在しない場合は、サーバーのログ設定に従います。

domains/<domain name>/application/

ドメインで使用するアプリケーションのアーカイブ・ファイルをコピー、解凍したディレクトリーです。アプリケーションのアーカイブ・ファイルが解凍されたディレクトリーにはEJB実装クラスとヘルパー・クラスが含まれています。

ファイル	説明
<EAR archive file>	EAR application-id名のディレクトリーの下位にEARアプリケーションの構成ファイルがそのままコピーされます
<EJB archive file>	EJB module-id名のディレクトリーの下位にEJBモジュールを構成するファイルがそのままコピーされます

lib/schemas

EJBと関連するスキーマ・ファイルが以下のディレクトリー別に格納されます。

下位ディレクトリー	説明
javaee	EJBと関連するすべてのJava EE XMLスキーマ・ファイルが存在します
jeus	EJBと関連するすべてのJEUS XMLスキーマ・ファイルが存在します

samples/ejb

EJBの実装例コードと下位ディレクトリーが含まれています。

#### 参考

ディレクトリー・リストで使用された"<",">"(例 "<server name>/")の括弧の中に使われた文字は実際のシステム設定値を示します。たとえば、実際のシステムが"server1"の場合、JEUS MSの値も"server1"と設定されます。したがって、"<server name>/"は実際のシステムで"server1/"というディレクトリーに指定されます。

## 1.3.2. XML設定ファイル

以下は、JEUS EJBの管理と設定に関連するXML設定ファイルです。XML設定ファイルの内容は、JEUSで定義されたXMLヘッダーで始まる必要があります。設定ファイルのXMLスキーマ・ファイルは、

JEUS\_HOME/lib/schemas/jeus/ディレクトリーに格納されます。さらに、ルート要素はJEUS XMLスキーマの名前空間を既存の名前空間に指定します。

- **domain.xml** (jeus-domain.xsd)

EJBエンジンに対する設定を行います。詳細説明は、『JEUS サーバガイド』の「[第2章 EJBエンジン](#)」を参照してください。

– 場所

```
JEUS_HOME/domains/<domain name>/config
```

– **domain.xml**のXMLヘッダー

**[例 1.1] XMLヘッダー : <<domain.xml>>**

```
<?xml version="1.0"?>
<domain version="8.0" xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

- **ejb-jar.xml** (ejb-jar\_3\_1.xsd)

主なEJBエンジン設定ファイルです。詳細内容は、EJB 3.2仕様を参照してください。

– 場所

```
META-INF/ 標準 JARファイル内の構造
```

– **ejb-jar.xml**のXMLヘッダー

ejb-jar.xmlのXMLヘッダーはEJB標準に含まれています。

**[例 1.2] XMLヘッダー : <<ejb-jar.xml >>**

```
<?xml version="1.0"?>
<ejb-jar version="3.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd">
```

- **jeus-ejb-dd.xml** (jeus-ejb-dd.xsd)

JEUS EJBモジュールのためのJEUS Deployment Descriptors(以下、DD)情報を設定します。詳細内容は、「[第3章 EJBモジュール](#)」を参照してください。

– 場所

```
META-INF/ 標準 JARファイル内の構造
```

– jeus-ejb-dd.xmlのXMLヘッダー

**[例 1.3] XMLヘッダー : <<jeus-ejb-dd.xml>>**

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="8.0">
```

---

#### 参考

本書で使用されるすべてのタグは、XMLスキーマの設定順に作成されています。タグの順序については、『JEUS XMLリファレンスガイド』を参照してください。JEUSでは、自動ソート機能を提供しているため、XML設定ファイルの作成時に順序を必ず守る必要はありません。

上記ファイルの例が本書で使用される際は、便宜上標準ヘッダーを省略しています。実際のXML設定ファイルには標準ヘッダーが含まれます。

---

### 1.3.3. 関連ツール

以下は、EJBエンジン、EJBモジュール、EJBコンポーネントを管理するために使用されるツールです。

- **コンソール・ツール(jeusadmin)**

JEUSのコンソール・ツールは、EJBエンジンを制御、モニタリングするために使用されます。詳細内容は、『JEUS リファレンスガイド』の「4.2.7. EJBエンジン関連コマンド」を参照してください。

- **WebAdmin**

WebAdminはJEUSのGUIベースの管理ツールとして、コンソール・ツールで可能なすべての作業をサポートします。WebAdminの詳細内容は、『JEUS WebAdminガイド』を参照してください。

---

#### 参考

EJB 2.xのためのappcompilerについては、『JEUS リファレンスガイド』の「4.3. appcompiler」を参照してください。

---

## 1.4. EJBの基本設定

本節では、JEUSでEJBを使用するためのインストール手順について説明します。

以下は、server1という名前のサーバーにEJBエンジンを設定する手順についての説明です。

1. JEUSが正常にインストールされているのかを確認し、環境変数を確認します。
2. COMMAND画面を実行し、JEUS\_HOME/binで以下のように入力するとDASが起動されます。

```
$ startDomainAdminServer -domain domain1 -u administrator -p password1 -verbose
. . .
[2016.08.06 21:33:15][2] [adminServer-1] [SERVER-0248] The JEUS server is STARTING.
. . .
[2016.08.06 21:33:16][3] [adminServer-1] [Security-0082] The JEUS security manager started.
. . .
[2016.08.06 21:33:16][3] [adminServer-1] [SERVER-0173] The JNDI naming server started.
. . .
[2016.08.06 21:33:28][2] [launcher-10] [Launcher-0034] The server[adminServer]
initialization completed successfully[pid : 2380].
[2016.08.06 21:33:28][0] [launcher-1] [Launcher-0040] Successfully started the server. The
server state is now RUNNING.
```

## 参考

上記の[3]は、ログ・レベルのうちCONFIGを意味します。したがって、CONFIG以上のログ・レベルが設定されている場合のみ確認できます。

- WebAdminで**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストからサーバー(MS)を選択すると、サーバー設定画面へ移動し、**[Engine] > [Ejb Engine] > [Basic]**メニューでEJBエンジンに関する設定ができます。設定しない場合は、デフォルト値で動作します。

### [図 1.2] EJBエンジンの設定

The screenshot shows the JEUS WebAdmin interface. On the left, a sidebar contains a navigation menu with options like Domain, Session, Clusters, Servers (selected), Applications, Security, Resources, Monitoring, and Console. Below the menu is a 'システム状態' (System Status) section showing the status of various components (Failed, Standby, Running, Shutdown, Suspended, Other). The main content area is titled 'EJB Engine' and has a 'HISTORY' button. Below the title, there's a description of the EJB Engine and its configuration options. The 'Basic' tab is selected, showing settings for 'Resolution' (300000) and 'Use Dynamic Proxy For Ejb2' (true). A description of the EJB Engine and its configuration options is provided.



---

## 参考

JEUSが提供するツールを使ったMS設定の変更は、DASが起動されている状態で可能です。

---

4. 別のコマンド画面で以下のようなコマンドを実行します。(ホスト情報はDASの情報です。)

```
$ jeusadmin -host localhost:9736
User name:
```

5. インストールするときに設定した管理者のユーザー名とパスワードを入力します。

```
User name: administrator
Password:
```

6. コマンド画面で以下のように入力して実行するとサーバーが起動されます。

```
$ startManagedServer -domain domain1 -server server1 -u administrator -p
password1 -verbose
...
[2016.08.07 14:10:10][2] [server1-1] [SERVER-0248] The JEUS server is STARTING.
...
[2016.08.07 14:10:17][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
```

7. helpコマンドを入力すると、EJBエンジンに対するモニタリングと管理に使用できるコマンドが照会されます。

```
[DAS]domain1.adminServer>help -g EJB
[ EJB]_____
cancel-ejb-timer          Cancels active EJB timers.
ejb-timer-info            Lists the active EJB timers. If no options are
                           specified, a list of all EJB modules that
                           contain timers will be displayed.
modify-active-management  Modify the active management configuration.
modify-check-resolution   Set the check resolution of the EJB engine on the
                           server.
show-active-management    Shows the active management of the EJB engine on
                           the server.
show-check-resolution     Shows the check resolution of the EJB engine on
                           the server.
To show detailed information for a command, use 'help [COMMAND_NAME]'.
ex) help connect
```

8. 終了するときは、コマンド画面にstop-server(サーバー終了) → local-shutdown(DAS終了) → exit(ツール終了)の順に入力します。

```
[DAS]domain1.adminServer>stop-server server1  
Server [server1] was successfully stopped.  
[DAS]domain1.adminServer>local-shutdown  
The server [adminServer] has been shut down successfully.  
offline>exit
```

9. すべてのJEUSシステムが終了されます。

---

#### 参考

インストールと設定についての詳細内容は、『JEUS インストール&スタートガイド』と、『JEUS サーバガイド』を参照してください。

---

## 第2章 EJBエンジン

本章では、EJBエンジンに関する基本的な内容とJEUS EJBのルート・レベルの概念である構造、設定、運用、モニタリング、チューニングについて説明します。

### 2.1. 概要

EJBエンジンはEJBの運用環境を提供します。本書での**EJBエンジン**は、EJB標準で使用するEJBコンテナと同様な概念です。EJBモジュール、EJBデプロイについての詳細情報は、「[第3章 EJBモジュール](#)」と「[第4章 EJBの共通特性](#)」を参照してください。

### 2.2. 主要機能

以下は、EJBエンジンの主要機能についての説明です。

- **EJBエンジンとManaged Server(MS)**

1つのMSには1つのEJBエンジンが存在します。しかし、1つのドメインには複数のMSが存在できるため、1つのドメインに複数のEJBエンジンが存在できます。

一般的に複数のマシンやCPU上において、複数のMSにEJBエンジンが設定され、MSはDASによってクラスター化されますが、このような設定をEJBクラスタリングといいます。これは、システムの性能向上や高い安定性、セキュリティ性が維持できる効率的なシステム構造です。EJBクラスタリングについての詳細内容は、「[第6章 EJBのクラスタリング](#)」を参照してください。

- **EJBエンジンの基本設定**

EJBエンジンは、`domain.xml`の`<ejb-engine>`で設定します。詳細内容は、「[2.4. EJBエンジンの設定](#)」を参照してください。

- **EJBエンジンのロギング**

`domain.xml`の設定によってMSにログを残すことが可能です。特にEJBエンジンのログ(`jeus.ejb`)のみ残すこともできますが、そのときMSログにもEJBエンジン・ログが残ります。

ログーのハンドラーにファイル・ハンドラーが指定されると、指定したファイル名でログ・ファイルが作成されます。さらに、コンソール・ハンドラーを使用すると、すべてのログ・メッセージを画面に残すこともできます。一般的には、ログ・メッセージがパイプを介してMSが開始されたコマンド画面に出力されます。

---

## 参考

そのほか、ユーザーが作成したハンドラーが登録できます。詳細内容は、『JEUS サーバガイド』の「第8章 ログイン」を参照してください。

---

### ● Active Management

Active ManagementはEJBモジュールに問題が発生した場合、EJBエンジンがEメールで通知する機能です。

たとえば、Beanクラスの誤った実装によって無限ループに陥ったり、デッドロックのような深刻なエラーが発生したりする場合、EJBエンジンがこれを検知して通知するようになっています。追加的にエラー処理ポリシーが設定できるため、異常現象が起こった場合は、Eメール通知や該当のEJBエンジンが動作するMSを自動的に再起動することができます。

Active Managementについての詳細内容は、「[2.4. EJBエンジンの設定](#)」を参照してください。

### ● HTTP Invoke

リモート・クライアントがJNDIサービスを利用してEJBインスタンスを検索するとき、クライアントはEJBメソッドが呼び出せるRMIスタブを取得します。通常スタブは、RMIランタイムを利用してEJBとのリモート通信を行います。RMI通信はTCPソケットをベースにしており、この方式はRMI通信ポートが別途必要なため、ファイアウォールが存在する環境では問題になる可能性があります。このとき必要な通信モードが**HTTP Invokeモード**です。

同モードを使用する場合、リモート・クライアントのRMI要求をHTTPで囲んでWebエンジンに転送し、WebエンジンはRMI要求を処理するサーブレット(jeus.rmi.http.ServletHandler)に渡します。このサーブレットは、RMIランタイムに要求を渡して実際のEJBメソッドを呼び出した後、その結果をHTTP応答で囲んでリモート・クライアントに伝達します。HTTP Invokeモードは、EJBエンジンまたはEJBコンポーネント別に設定できます。

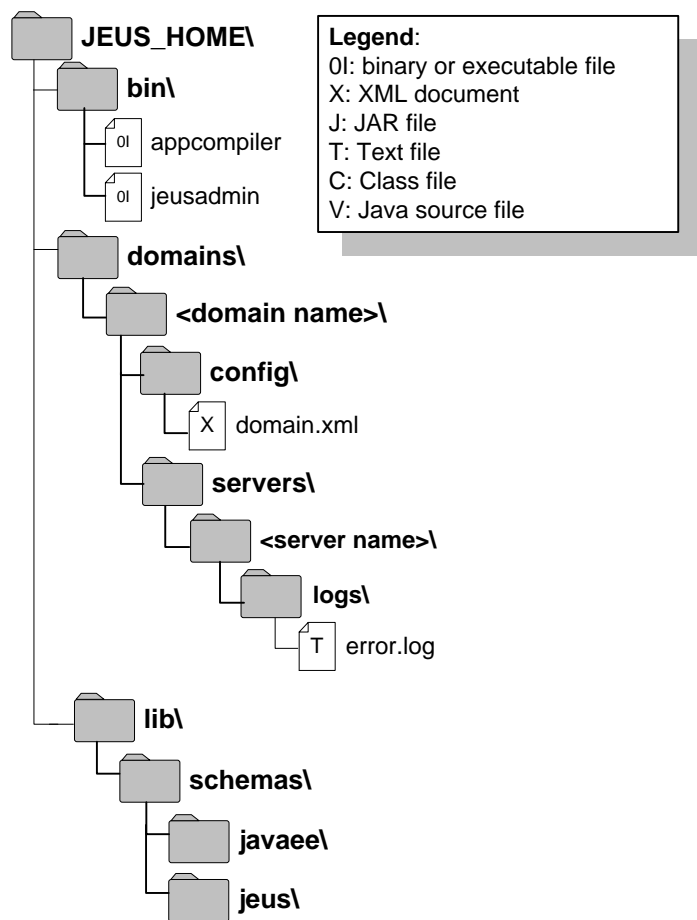
domain.xmlの<invoke-http>にHTTP Invokeモードが設定されると、EJBエンジン内のすべてのモジュールに適用されます。EJBモジュールのjeus-ejb-dd.xmlでは、特定のEJBコンポーネントのみ適用できます。その際、jeus-ejb-dd.xmlの設定がdomain.xmlの設定より優先します。

domain.xmlとjeus-ejb-dd.xmlの設定についての詳細内容は、「[4.2.4. HTTP呼び出し\(Invoke\)の環境設定](#)」を参照してください。

## 2.3. EJBエンジンのディレクトリー構造

以下は、EJBエンジンを管理するときに使用するディレクトリーとファイルの一覧です。

【図 2.1】 EJBエンジンのディレクトリー構造



基本ディレクトリーに関する内容は、「[1.3.1. ディレクトリーの構造](#)」と同様です。以下は、EJBエンジンで主に使われるディレクトリーの説明です。

bin

EJBエンジンを管理するツールが存在するディレクトリーです。

EJBエンジン管理 ツール	説明
appcompiler	EJBをデプロイするために必要なクラスを作成、コンパイルし、高速デプロイを実行します
jeusadmin	EJBエンジンを制御、モニタリングするために使用します

domains/<doamin name>/servers/<server name>/logs

EJBエンジンのログ・ファイルが存在するディレクトリーです。EJBログ・ファイルのファイル・ハンドラーを指定した場合は別途ファイルで作成されます。ハンドラーが存在しない場合は、サーバーのログ設定に従います。

## 2.4. EJBエンジンの設定

本節では、WebAdminを使用してEJBエンジンを設定する方法について説明します。

WebAdminを使用したEJBエンジンの設定は、以下の3つで分けられます。WebAdminを使って設定された内容は、JEUS\_HOME/domains/<domain name>/configに存在するdomain.xmlファイルに保存されます。

- Basic
- Active Management
- Timer Service

---

### 参考

1つのMSIには1つのEJBエンジンが存在します。MSの追加方法についての詳細内容は、『JEUS サーバガイド』の「第2章 JEUSの設定」を参照してください。

---

### 2.4.1. Basic設定

WebAdminを使用したEJBエンジンのBasic設定手順は以下のとおりです。

1. WebAdminの[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択してサーバー設定画面へ移動した後、[**Engine**] > [**Ejb Engine**] > [**Basic**]メニューを選択します。
2. [**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。

3. [Ejb Engine]画面でEJBエンジンに関する基本的な項目を設定し、[確認]ボタンをクリックします。

[図 2.2] EJBエンジンの設定 - Basic設定

domain1

Domain

Session

Clusters

Servers

Applications

Security

Resources

Monitoring

Console

システム状態

0 Failed

0 Standby

1 Running

0 Shutdown

0 Suspended

0 Other

Runtime Info

LOCK & EDIT

指定したドメインの項目を変更、追加、削除する機能です。

運用マニュアル

## EJB Engine

HISTORY

EJBエンジンは、J2EE EJBアプリケーションが動作するための環境を提供します。J2EEの仕様でのEJBコンテナに相当する機能です。サーバの起動時に実行され、1つのサーバには1つのEJBエンジンのみサポートされます。

ヘルプ

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Active Management Time Service

動的設定 必須項目 このページの設定を変更したい場合は、左のメニューから[Lock & Edit]ボタンをクリックしてください。TIP

Resolution 300000 ms EX 100000

[デフォルト:300000] EJBエンジンの状態をチェックする周期です。Active Managementを設定した場合、ブロックされたスレッド数をチェックして設定されている動作を行います。各Beanが<bean-pool>と<connect-pool>を使用する場合、プールのアイドル数を減らす作業をリサイジングといいます。<resizing-period>を過ぎているのをチェックしてからリサイジングを行います。さらに、ステートフルセッションBeanがクライアントからの<passivation-timeout>以上の要求をチェックし、必要に応じてパッシベーションを実行します。

Use Dynamic Proxy For Ejb2 true EX true

[デフォルト:true] (Since JEUS v6.0 Fix#7)JEUS v6.0 Fix#6以上のバージョンからは、EJB 2.xスタイルのBeanにおいて、既存のRMIスタブ方式の代わりに動的プロキシ方式を使用することができます。動的プロキシ方式とは、クライアント別に直接RMIスタブクラスをロードして使用するのではなく、すでに作成されているRMIスタブに対する動的プロキシ(java.lang.reflect.Proxy)を利用してメソッドを呼び出す方式です。この動的プロキシ方式を利用することにより、ローカル呼び出しの最適化(Local Call Optimization:同じJVMで呼び出されるEJBリモート呼び出しがRMIを介さずにローカル呼び出しを使うことによって性能が異なる機能)を使用することができます。ただし、JEUS v6.0 Fix#6より旧バージョンのJEUSライブラリを使用しているクライアントがEJBをリモートで呼び出す場合は互換性のトラブルが発生するため、このオプションをfalseに設定します。同オプションは、EJBモジュール別のjeus-ejb-dd.xmlの<module-info><use-dynamic-proxy-for-ejb2>に設定できますが、現在EJBエンジンのすべてのEJBモジュールは、EJB 2.xに対して動的プロキシ方式を使用せずにリモートクライアントとの互換性を維持するため、Use Dynamic Proxy For Ejb2をfalseに設定し、EJBモジュール別の設定を無視するようにすることができます。デフォルト値はtrueで、EJB 2.xスタイルのBeanに対して動的プロキシ方式を使用します。

### 詳細設定

すべてを開く

Enable User Notify ☐ EX true

#### Async Service

Thread Min 0 EX 10

Thread Max 30 EX 100

Access Timeout 300000 ms EX 180000

Invoke Http ☐

確認 再設定 削除

以下は、**基本情報**および**詳細設定**の設定項目についての説明です。

- 基本情報

項目	説明
Resolution	アクティブ・マネージメントとパッシベーションのチェック周期を設定します。すなわち、ブロックされたスレッドを検知する周期と、パッシベーション・タイムアウトの間、クライアントから要求を受けないBeanを検知する周期です
Use Dynamic Proxy For Ejb2	既存のRMIスタブ方式の代わり、動的プロキシ(Dynamic Proxy)方式を使用します

- Ejb Engine

項目	説明
Enable User Notify	オプションを設定するとEJBの例外がサーバーに設定したユーザー・ログに残ります。ユーザー・ログについては、『JEUS サーバガイド』の「第8章 ログイン」を参照してください

- Async Service

Asynchronous Invocation Serviceのための設定です。

項目	説明
Thread Min	維持するスレッドの最小数を設定します
Thread Max	維持するスレッドの最大数を設定します
Access Timeout	非同期(Async)メソッドの実行後、一定の時間が過ぎてもクライアントがゲットしない場合は、Futureオブジェクトを削除します。  クライアントのミスでゲットしなかった場合に備え、メモリー・リークの発生を防ぐための設定です

- Invoke Http

EJB RMIスタブがRMIランタイム・ポートにアクセスできない状況の場合に設定します。

項目	説明
Url	RMIスタブから呼び出されるRMIハンドラー・サーブレットのURIを入力します  – URIには、プロトコル、IPアドレス、ポートを除いたサーブレット・リクエスト・パスのみ入力します。すなわち、rmiHandlerServlet.warのjeus-web-dd.xmlに設定した<context-path>と、web.xmlに設定した<url-pattern>を続けて入力します。jeus-web-dd.xmlを作成しない場合は、<context-path>のデフォルト値であるwarの名前として、例ではrmiHandlerServletになります



項目	説明
	<ul style="list-style-type: none"> <li>– プロトコルはHTTP、IPはRMIランタイムと同様なアドレスで見なされます。HTTP-RMIリクエストを受けたWebサーバーとWebエンジンは、RMIランタイムと同じマシンに存在する必要があります。そうすると、RMIランタイムのアドレスはRMIスタブに通知されます。Webサーバーのポートは必ず次の&lt;http-port&gt;に設定します</li> <li>– JEUSが提供するrmiHandlerServlet.warには、jeus-web-dd.xmlが含まれていません。そのため、コンテキストはモジュール名と同様なrmiHandlerServletを使用します。また、web.xmlには&lt;url-pattern&gt;がサーブレット・ハンドラーとして設定されています。デフォルト値以外の設定が使用したい場合は、jeus-web-dd.xmlを作成してweb.xmlを修正した後、rmiHandlerServlet.warをデプロイします</li> </ul>
Http Port	HTTP-RMI要求を受けるポート番号を設定します。WebサーバーおよびWebエンジンには、RMIハンドラー・サーブレットがデプロイされ、実行されている必要があります

4. 設定内容を動的に反映するために[**Activate Changes**]ボタンをクリックします。

## 2.4.2. Active Management設定

Active Management設定は、エンジン再起動の条件とEメール通知機能で分けられます。エンジン再起動の条件は、EJBエンジンが再起動する前まで許可できる、ブロックされたEJBスレッドの最大数で決まります。

WebAdminを使用したEJBエンジンのActive Management設定手順は以下のとおりです。

1. WebAdminの[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択してサーバー設定画面へ移動した後、[**Engine**] > [**Ejb Engine**] > [**Active Management**]メニューを選択します。
2. [**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. [**Active Management**]画面で各項目を設定し、[**確認**]ボタンをクリックします。

[図 2.3] EJBエンジンの設定 - Active Management設定

Active Management

HISTORY

EJBエンジンをモニタリングして不具合を処理し、その結果をEメールで管理者に通知します。 [Performance Recommendation]: EJBエンジンのアクティブマネージメントよりは、サーブレットエンジンに含まれているWebコネクションを使用することをお勧めします。

ヘルプ ?

Basic

Resource

Engine

Web Engine

Jms Engine

Ejb Engine

Basic

Active Management

Time Service

動的設定

必須項目

確認

再設定

削除

Max Blocked Thread

100

EX 200

[デフォルト:-1] ブロックされたスレッドの最大数を設定します。この設定値よりEJBにブロックされたスレッド数が多い場合はコンテナを再起動します。この値が小さい場合、EJBエンジンが頻繁に再起動されることになるので注意が必要です。

Max Idle Time

300000

ms

EX 180000

[デフォルト:300000] EJBスレッドがブロックされていると判断する状態を定義します。この設定値は時間を意味しており、特定のスレッドがこの時間を超えてアイドル状態を維持している場合、ブロックされていると判断します。

詳細設定

すべてを開く

Email Notify

Smtip Host Address

mail,foo,com

From Address

admin@mail,foo,com

Sender Id

admin@mail,foo,com

Sender Password

myPassword

EX (DES)FQrLbQ/D8O1IDV571L28rw==

To Address

toAddress@mail,foo,com

Property

Cc Address

ccAddress@mail,foo,com

Bcc Address

bccAddress@mail,foo,com

確認

再設定

削除

以下は、**基本情報**および**詳細設定**の設定項目についての説明です。

- 基本情報

項目	説明
Max Blocked Thread	EJBエンジンが再起動する前まで許可できる、ブロックされたEJBスレッドの最大数です。この設定値が小さい場合、EJBエンジンが頻繁に再起動されることになるので注意が必要です。  デフォルト値に設定すると、ブロックされたスレッド数に制限がないことを意味します。すなわち、EJBエンジンはブロックされたスレッドの影響で再起動されることはありません
Max Idle Time	指定された時間の間、スレッドがブロックされた状態でリクエストを受けず、アイドル状態になっている場合は「ブロックされたスレッド・リスト」に追加されます。  この設定は、エンジンでブロックされたスレッドを判断する基準となります

- Email Notify

- 再起動の条件によってEJBエンジンが開始される場合、Eメール通知を転送するための情報を設定します。
- 以下は、下位設定項目についての説明です。

項目	説明
Smtpt Host Address	メッセージを転送するときに使用するSMTPのアドレスです。このアドレスは、ホスト名やIPアドレスで設定します
From Address	Eメール送信者のアドレスを設定します
Sender Id	SMTPアドレスを利用して認証を受けるIDを設定します
Sender Password	SMTPアドレスを利用して認証を受けるIDのパスワードを設定します
To Address	Eメール受信者のアドレスを設定します
Property	基本的なSMTPプロパティ以外に追加で必要なプロパティがある場合は、key、value型で設定します
Cc Address	Cc(カーボンコピー)に指定してEメールを受信する人のアドレスを入力します
Bcc Address	Bcc(ブラインドカーボンコピー)に指定してEメールを受信する人のアドレスを入力します

4. 設定内容を動的に反映するために[**Activate Changes**]ボタンをクリックします。

### 2.4.3. Timer Service設定

EJBタイマー・サービスとは、EJBが一定の時間または周期的にコールバックが受けられるようにするサービスです。

基本的な使用方法是EJB仕様に説明されているため、本節では、JEUS EJBが提供するタイマー・サービスと同機能を使用するための設定についてのみ説明します。詳細内容は、「[第10章 EJBタイマー・サービス](#)」を参照してください。

## 2.5. システム・ログの設定

EJBエンジンのシステム・ロギングとユーザー・ロギングは、WebAdminの以下のメニューで設定できます。

- システム・ロギング設定

WebAdminの[**Servers**] > **サーバー選択** > [**Basic**] > [**System Logging**]メニューで設定します。

- ユーザー・ロギング設定

WebAdminの[**Servers**] > **サーバー選択** > [**Basic**] > [**User Logging**]メニューで設定します。

システム・ロギング設定はEJBエンジンのみならず、すべてのエンジンに適用される共通した設定なので詳細内容については、『*JEUS サーバガイド*』の「[第8章 ロギング](#)」を参照してください。

## 2.6. EJBエンジンの制御およびモニタリング

EJBエンジンの制御は別のJEUSエンジン(サーブレットまたはJMS)の制御と類似しています。

WebAdminおよびコンソール・ツールを使って、EJBエンジンの実行環境と状態情報をモニタリングできます。WebAdminを使用したEJBエンジンの制御およびモニタリング方法は以下のとおりです。

- EJBエンジンの制御

WebAdminの[**Servers**]メニューを選択すると表示されるサーバー・リストから目的のサーバー名を指定した後、[**Engine**] > [**Ejb Engine**]を選択します。

- EJBエンジンのモニタリング

WebAdminの[**RUNTIME INFO**]ボタンをクリックした後、EJBエンジンの制御と同様な方法で、モニタリングするEJBエンジンを指定します。

---

#### 参考

1. EJBエンジンのモニタリングは、コンソール・ツールに比べてWebAdminが詳細かつ完璧なエンジン状態情報を提供するため、WebAdminを使用することをお勧めします。WebAdminの詳細内容については、『*JEUS WebAdminガイド*』を参照してください。

2. コンソール・ツールを使用して基本的な実行環境の情報が確認できます。詳細内容については、『JEUS リファレンスガイド』の「4.2.7. EJBエンジン関連コマンド」を参照してください。

---

## 2.7. EJBエンジンのチューニング

EJBエンジンの全体的な性能向上のために設定を変更することができます。本節では、EJBエンジンの性能関連設定について簡単に説明します。

以下は、チューニングのために必要な事項です。

- レゾリューション設定のチューニング
- 高速デプロイ(Fast Deploy)機能の使用
- 最大性能のためのシステム・ログの設定
- アクティブ管理(Active Management)の不使用
- HTTP呼び出しモードの使用

---

### 参考

EJBエンジンの情報やヒントについては、『JEUS XMLリファレンスガイド』の「11. domain.xml EJBエンジンの設定」を参照してください。XML/スキーマのうち、「P」と表示されているのは性能と関連するものです。

---

### 2.7.1. レゾリューション設定のチューニング

レゾリューションはEJBエンジンの状態をチェックする周期として、2つの周期が使用されます。

- ブロックされたスレッド数をチェックした後、EJBエンジンを再起動するアクティブ管理のチェック周期です。
- すべての下位コンポーネント(例、Beanプール)をチェックし、各Beanが無効対象なのかをチェックするパッシベーションのチェック周期です。

レゾリューションの値が大きいほどシステム・メモリーやその他のリソースの回収周期が長くなり、資源の活用率は低下しますが、エンジンの性能は向上されます。一方、値を小さく指定すると最新の状態が保たれますが、エンジン全体の性能は低下します。

したがって、メモリーのサイズや用途に合わせてレゾリューション値を適切に設定する必要があります。

---

#### 参考

レゾリューション値の設定によって、<passivation-timeout>または<disconnect-timeout>が適時に発生しない場合がありますので注意が必要です。

---

### 2.7.2. 高速デプロイ

高速デプロイ・オプションはEJBに設定せずアプリケーション別に設定しますが、性能に大きな影響を与えます。

エンジンが起動されるとき、デプロイが必要なEJBモジュールがすでにコンパイルされており、RMIスタブとスケルトンが存在する場合は、デプロイ・コマンドを使用するとき-fastオプションを追加します。これは、エンジンがEJBモジュールをデプロイする際にRMIクラスを生成しないようにするためです。

EJBモジュールとデプロイの詳細については、「[第3章 EJBモジュール](#)」を参照してください。

### 2.7.3. 最大性能のためのシステム・ログの設定

性能を改善するために、システム・ログを以下の3つの方法で調整できます。

- ファイル・ハンドラーを使ってロギングを迅速に行います。
- ファイル・ハンドラーのバッファ・サイズを大きく設定します。
- ログ・レベルを「SEVERE」に設定します。

---

#### 注

上記の提案は、安定的な運用環境でのみ適用してください。開発環境では開発を容易に行うため、上記値と逆に設定する必要があります。すなわち、コンソール・ハンドラーを使用し、かつバッファ・サイズを小さく設定し、ログ・レベルを「FINE」に指定します。

---

### 2.7.4. アクティブ管理の不使用

EJBエンジン・レベルのアクティブ管理を使用すると、設定によっては性能の低下を引き起こす可能性があるため、基本的には使用しません。その代わりに、サーブレット・エンジンに定義されたアクティブ管理を使用します。したがって、通常はアクティブ管理の設定を省略する場合があります。

## 2.7.5. HTTP呼び出しモードの使用

クライアント数が多い場合にHTTP呼び出しモードを使用すると、性能向上が期待できます。WebサーバーがJEUSへの要求を一定に調整するため、クライアントの要求どおりにスレッドを作成するRMIに比べ、コネクション数が少なく作成されます。ただし、HTTPプロトコルを使用すると、むしろ性能低下を引き起こす可能性があるので注意が必要です。





# 第3章 EJBモジュール

本章では、EJBモジュールの構造と管理方法について説明します。

## 3.1. 概要

EJBモジュールは、EJBエンジンにデプロイできる基本単位です。また、EJBコンポーネントをグループ化してその設定情報をDDで表現します。EJBをデプロイするときは、たった1つのEJBコンポーネントをデプロイ場合であっても必ずEJBモジュールにパッケージングします。

---

### 参考

EJB 2.1以下のコンポーネントの場合は、EJBモジュールにDD(ejb-jar.xml)が存在する必要がありますが、3.0以上のコンポーネントからはアノテーションをサポートしており、DDは選択事項に変更されました。

---

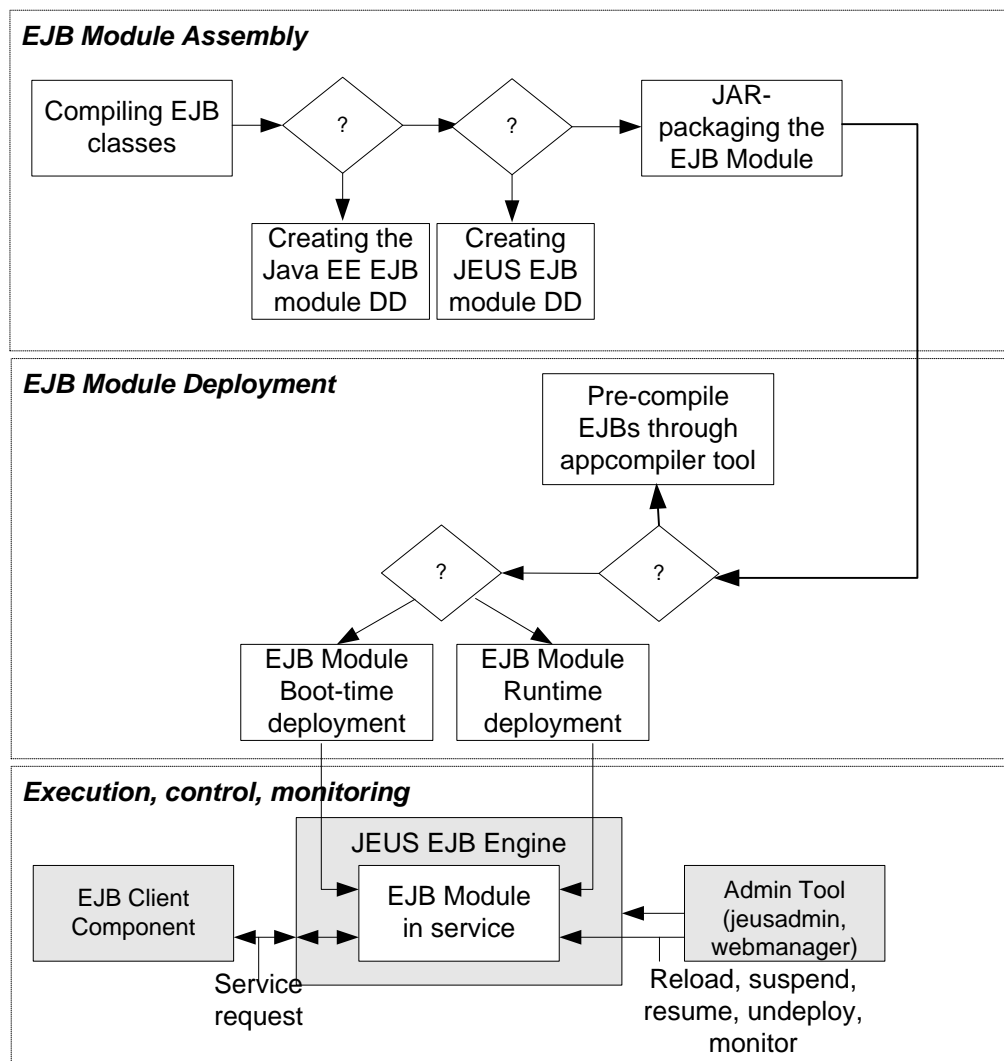
## 3.2. EJBモジュールの管理

以下は、JEUS内部でEJBモジュールを管理するための4つの主要作業です。作業の詳細内容については各節を参照してください。

- アセンブリー : 「[3.3. EJBモジュールの組み立て\(Assembling\)](#)」
- デプロイメント : 「[3.4. EJBモジュールのデプロイ](#)」
- 制御(undeploy, redeploy, stop-application, start-application) : 「[3.5.1. EJBモジュールの制御](#)」
- 監視 : 「[3.5.2. EJBモジュールのモニタリング](#)」

EJBモジュールを管理する手順は以下のとおりです。

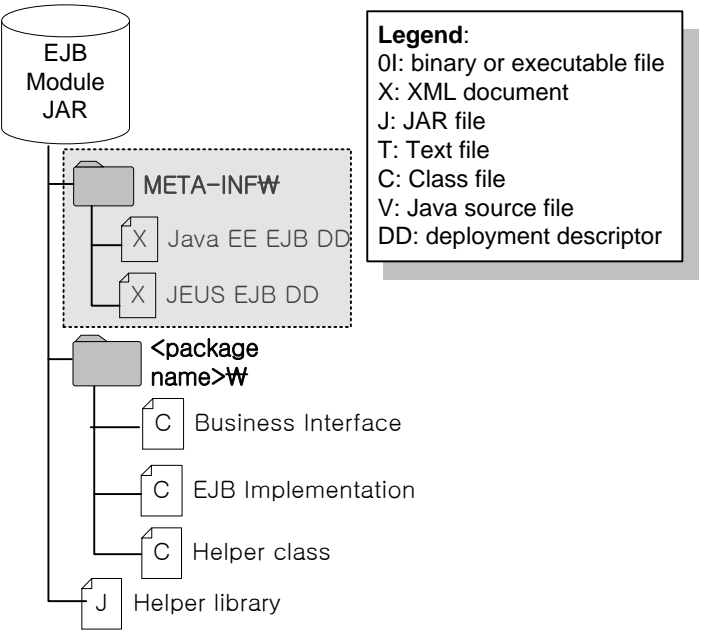
**[図 3.1] EJBモジュールの管理手順**



# EJBモジュールのJARファイルの構造

以下は、標準EJBモジュールのJARファイルの構造についての説明です。

[図 3.2] Java EE EJBモジュールのJARファイルの構造



## META-INF

区分	説明
ejb-jar.xml	実際の標準EJB DDファイルです。アノテーションで設定情報を表現した場合には存在しない場合もあります。ただし、EJB 2.xスタイルの場合は必ず必要です
jeus-ejb-dd.xml	JEUSでEJBをデプロイするとき必要なDDファイルです(選択事項)

## <package name>/

EJBクラスを含んでいます。EJBソースコードに定義されているものと同様、Javaパッケージ構造を反映する必要があります。たとえば、1つのEJBが「mypackage」というパッケージに属していると宣言されている場合、EJBクラスはEJB JARの「mypackage」ディレクトリーの下位に位置する必要があります。

以下のようなEJBコンポーネントが含まれています。

区分	説明
インターフェース	クライアントがアクセスするためのEJB 2.xスタイルの3.0以上のインターフェースです
エンタープライズBeanクラス	インターフェースに合わせて実際の業務ロジックを実装したBeanクラスです

MANIFEST.MF Class-Path Entryに明示したライブラリーに該当します。

### 3.3. EJBモジュールの組み立て(Assembling)

[\[図 3.1\]](#)のようにEJBモジュールを組み立てる手順です。

#### 1. EJBクラスのコンパイル

開発者が実装したEJBソース・ファイルをコンパイルします。

#### 2. DDの作成

- ejb-jar.xml DDの作成
- jeus-ejb-dd.xml DDの作成

#### 3. EJB JARファイルのパッケージング

JARファイルにEJBクラスとDDをパッケージングします。

次の節からは、「counter」例を利用してEJBモジュールの組み立て手順を説明します。「counter」は、JEUS\_HOME/samples/ejb/basic/statefulSession/ディレクトリーに存在するステートフル・セッションBeanとして、以下のような2つのJavaソース・ファイルが存在します。

- Counter.java(business interface)
- CounterEJB.java(bean implementation)

---

#### 参考

ステートフル・セッションBeanや他の種類のBeanに関する詳細情報は、[「第7章 セッションBean」](#)、[「第9章 メッセージ駆動型Bean\(MDB\)」](#)を参照してください。

---

#### 3.3.1. EJBクラスのコンパイル

EJBソース・ファイルからEJBモジュールを組み立てる最初の手順は、JDKのjavacコンパイラを使って開発者が実装したEJBソース・ファイルをコンパイルすることです。基本的にクラス・パスはJEUS\_HOME/lib/system/javaee.jarを使用します。jeus-ejb-dd.xmlに対応されるアノテーションを使用した場合は、JEUS\_HOME/lib/system/jeusapi.jarを追加し、また別のhelperクラスがあれば、それもすべて追加します。

以下は、「counter」Beanで使用するEJB JavaファイルをUNIX環境でコンパイルする実行例です。

```
$ javac -classpath JEUS_HOME/lib/system/javaee.jar *.java
```

上記のコマンドを実行するとクラス・ファイルが生成されます。

### 3.3.2. Deployment Descriptors(DD)の作成

EJBモジュールをデプロイするときに使用されるDDには、以下の2つがあります。

- 標準EJB DD

ejb-jar.xmlと命名され、EJBモジュールの「META-INF/」ディレクトリーに格納されます。開発者がEJBコンポーネントを作成するときは、設定情報の一部または全部をアノテーションで作成することができます。アノテーション情報を無視してDD設定のみを適用したい場合は、ejb-jar.xmlの<metadata-complete>をtrueに設定します。アノテーションとDDを混用した場合は、DDがアノテーションに優先します。

- JEUS EJB DD

jeus-ejb-dd.xmlと命名され、EJBモジュールの「META-INF/」ディレクトリーに格納されます。このファイルの詳細情報については、以下の[「jeus-ejb-dd.xml DDの作成」](#)を参照してください。

### 標準EJB DD(ejb-jar.xml)の作成

DDのフォーマットはEJB標準に従います。ほとんどはアノテーションで可能ですが、必要な場合はDDを使用します。

```
package ejb.basic.statefulSession;
@Remote
public interface Counter
{
    public void increase();
    ...
}
```

```
package ejb.basic.statefulSession;
@Stateful
@EJB(name="counter")
@TransactionManagement( TransactionManagementType.BEAN)
public class CounterEJB implements Counter {
    private int count = 0;

    public void increase()
    {
        count++;
    }
}
```

```
}  
...  
}
```

以下の例は、「counter」Beanの簡単なEJB標準のDD(ejb-jar.xml)を表したものです。

### [例 3.1] EJB標準のDD : <<ejb-jar.xml>>

```
<?xml version="1.0"?>  
<ejb-jar version="3.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd">  
  <enterprise-beans>  
    <session>  
      <ejb-name>counter</ejb-name>  
      <business-remote>ejb.basic.statefulSession.Counter</business-remote>  
      <ejb-class>ejb.basic.statefulSession.CounterEJB</ejb-class>  
      <session-type>Stateful</session-type>  
      <transaction-type>Bean</transaction-type>  
    </session>  
  </enterprise-beans>  
  <assembly-descriptor/>  
</ejb-jar>
```

---

#### 参考

EJB 3.0以上からは、ejb-jar.xmlのすべての内容はアノテーションで表現できるため、ejb-jar.xmlは選択的に使用します。

---

## JEUS EJB DD(jeus-ejb-dd.xml)の作成

JEUS EJB DDが必要な理由は、EJBエンジンにデプロイするための基本設定および例外リファレンスをJEUSサービスにマッピングするためです。そのほか、インスタンス・プーリングとエンティティBean、DBテーブルとのマッピング設定もjeus-ejb-dd.xmlで行います。

---

#### 参考

このような設定はEJBモジュールをJEUSに含わせてデプロイするために必要な設定ですが、一部はアノテーションで設定できます。jeus-ejb-dd.xmlが存在しない場合もデフォルト値で設定されるので、別途の環境設定が必要な場合のみ同作業を行います。

---

JEUS EJB DDは標準EJB DDの追加設定です。標準EJB DDと同様、JEUS EJB DDもEJBモジュールに適用されます。以下は、JEUS EJB DDの全般的な構成です。

### [例 3.2] JEUS EJB DD : <<jeus-ejb-dd.xml>>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    <role-permission>
      <!-- See chapter 「4.2.6. EJBのセキュリティー設定」 -->
      . . .
    </role-permission>
  </module-info>
  <beanlist>
    <jeusbean>
      <!-- See chapters 「第4章 EJBの共通特性」
      and 「第7章 セッションBean」 -->
      . . .
    </jeusbean>
    <jeusbean>
      <!-- See chapters 「第4章 EJBの共通特性」
      and 「第7章 セッションBean」 -->
      . . .
    </jeusbean>
    . . .
  </beanlist>
  <ejb-relation-map>
    <!-- See chapter 「第8章 エンティティBean」 -->
  </ejb-relation-map>
  <message-destination>
    <jndi-info>
      <ref-name>ejb/AccountEJB</ref-name>
      <export-name>ACCEJB</export-name>
    </jndi-info>
  </message-destination>

  <library-ref>
    <library-name>jsf</library-name>
    <specification-version>1.2</specification-version>
    <implementation-version>1.2</implementation-version>
  </library-ref>

</jeus-ejb-dd>
```

JEUS EJB DDの主要素は以下のとおりです。

要素	説明
<module-info>	全EJBモジュールに適用される包括的な情報を設定します。ロール割り当て (Role Assignment) についての情報は、「4.2.6. EJBのセキュリティー設定」を参照してください

要素	説明
<beanlist>	モジュールに含まれるEJBを宣言します。下位タブの名前と内容はBeanの種類(セッションBean、エンティティBean、MDB)によって詳細設定が異なります。beanlistとその下位タグは、「 <a href="#">第4章 EJBの共通特性</a> 」、「 <a href="#">第7章 セッションBean</a> 」、「 <a href="#">第9章 メッセージ駆動型Bean(MDB)</a> 」で詳しく説明します
<ejb-relation-map>	CMP 2.0 BeanのためのCMRマッピングを定義したEJBのリレーショナル・マッピング設定です。CMPの使用はこれ以上推奨しませんので、JPAを使用してください
<message-destination>	ejb-jar.xmlの<message-destination>に宣言されたメッセージ・デスティネーションとJNDIに登録された実際のデスティネーション・オブジェクトをマッピングします
<library-ref>	アプリケーションで使用する共有ライブラリー(shared library)の情報を設定します

### 3.3.3. EJB JARファイルのパッケージング

EJB JARファイルのパッケージングするときはjarユーティリティを使用し、以下の事項を想定します。

- ejb/basic/statefulSessionディレクトリーにcounter EJBクラス・ファイルが存在する
- クラスにアノテーションで設定し、ejb-jar.xmlとjeus-ejb-dd.xmlは存在しない

EJB JARファイルのパッケージング手順は以下のとおりです。

1. 以下のようにEJBクラスを1つのJARファイルにパッケージングします。

```
$ jar cf countermod.jar ejb/basic/statefulSession
```

countermod.jarというJava EE EJBモジュールが作成されます。

2. クライアントでこのEJBを使用するためにはインターフェース・ファイルが必要です。クライアントで使用するためにデプロイするcounterbeanclient.jarは、以下のように作成します。

```
$ jar cf counterbeanclient.jar ejb/basic/statefulSession/Counter.class
```

#### 参考

counterbeanclient.jarはリモート・クライアントで使用するためのデプロイ用のライブラリーです。



## 3.4. EJBモジュールのデプロイ

本節では、EJBモジュールのデプロイに関する全般的な内容について説明します。

### 3.4.1. デプロイ

一般的にはデプロイ時にEJBクラスをコンパイルしますが、appcompilerを使用してEJBクラスを事前にコンパイルすると、デプロイ時の速度を高めることができます。また、自動デプロイ(Auto Deploy)機能を使って簡単にEJBモジュールをデプロイすることもできます。デプロイについての詳細内容は、『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。

[図 3.1]でJEUSは以下の方法でEJBモジュールをデプロイします。

- appcompilerツールの使用

appcompilerツールを使用するとスタブとスケルトン・ファイルを事前に作成するため、デプロイ速度が速くなります。

- EJBモジュールのブートタイム・デプロイ

EJBエンジンが開始するたびにEJBモジュールが自動的にデプロイされるようにします。

- EJBモジュールのランタイム・デプロイ

EJBエンジンが開始された後、すなわちEJBエンジンが運用されているときにEJBモジュールをデプロイします。

コンソール・ツールとWebAdminを使用してEJBモジュールをEJBエンジンにデプロイします。

### appcompilerツールの使用

EJB 2.xモジュールの場合、appcompilerツールを使用するとスタブとスケルトン・ファイルが事前に作成され、デプロイ速度が速くなります。appcompilerを使用する場合、コンソール・ツールのデプロイ・コマンドを実行するとき-fastオプションを使用します。同オプションを使用しない場合、appcompilerは無効になります。appcompilerツールについての詳細内容は、『JEUS リファレンスガイド』の「4.3. appcompiler」を参照してください。

## EJBモジュールのブートタイム・デプロイ

MSが起動されるときにデプロイされることをブートタイム・デプロイといいます。DASを通じて一度デプロイされたアプリケーションはMSを起動するたびにブートタイム・デプロイが行われます。アプリケーションのデプロイ設定に関する詳細内容は、『JEUS アプリケーション&デプロイメントガイド』の「第4章 アプリケーションの作成およびデプロイ」を参照してください。

1つのドメインに複数のアプリケーションが登録できます。

## EJBモジュールのランタイム・デプロイ

ランタイム・デプロイとは、実行されているEJBエンジンにEJBモジュールをデプロイすることです。コンソール・ツールの`deploy`コマンドを使って実行されているEJBエンジンにモジュールをインストールすることができます。

「countmod」EJBモジュールのデプロイ手順について説明します。モジュールをデプロイするために以下の事項を想定します。

- JEUSのdomain1というDASとserver1というMSがすでに起動されている
- countmodモジュールがJEUS\_HOME/apphome下位のcountmod.jarまたはcountmodディレクトリ形式で存在する

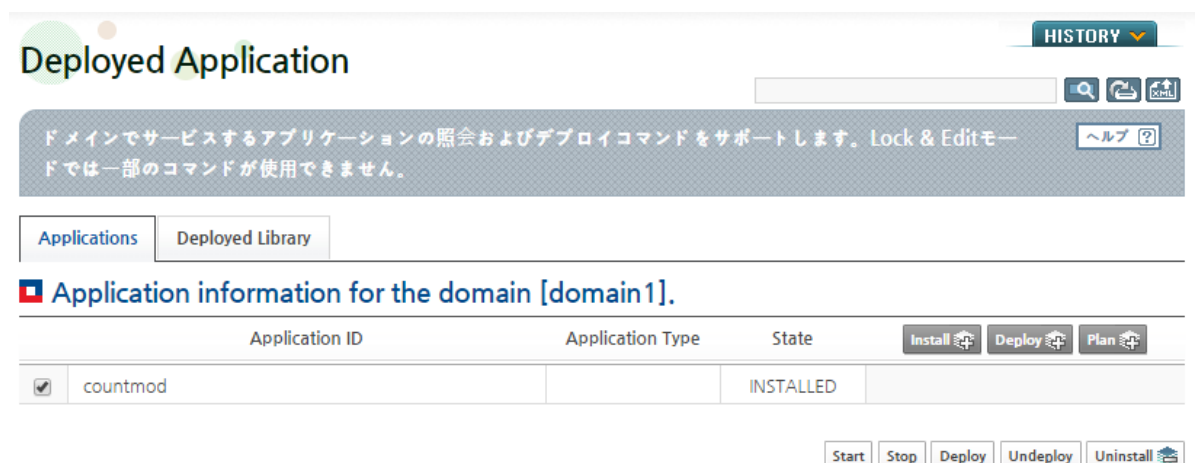
EJBモジュールのランタイム・デプロイは、コンソール・ツールまたはWebAdminを使用することができます。

### ● WebAdminの使用

WebAdminを使用してデプロイする手順は以下のとおりです。

1. WebAdminの[Applications]を選択すると、アプリケーション・リストの照会画面が表示されます。アプリケーション・リストの照会画面から、デプロイするアプリケーションの[deploy]ボタンをクリックします。

【図 3.3】アプリケーションのデプロイ - アプリケーション・リストの照会



2. [LOCK & EDIT]ボタンをクリックして、設定変更モードに切り替えます。
3. [Deploy]画面にてアプリケーションのデプロイ属性を設定し、[確認]ボタンをクリックします。

[図 3.4] アプリケーションのデプロイ - 属性の設定

Deploy

動的設定

必須項目

Targets

Target All Servers

☐

ドメインに存在するすべてのサーバにアプリケーションをデプロイするかどうかを設定します。

Target Server

☐ adminServer -- RUNNING(00:44:16)

☐ server1 -- SHUTDOWN

☐ server2 -- SHUTDOWN

アプリケーションをデプロイするサーバを指定します。

Target Cluster

選択できる項目が存在しません。  
アプリケーションをデプロイするクラスタを指定します。

Virtual Host

WebアプリケーションをサービスするWeb仮想ホストを指定します。

Dependent Libraries

アプリケーションが使用するライブラリを定義します。各ライブラリはアプリケーションより先にデプロイされている必要があります。この値は、デプロイ作業の終了と同時にDASのxmlに書き込まれます。ユーザが任意で変更してはなりません。

Dependent Library

アプリケーションが使用するライブラリを定義します。デプロイされたライブラリから選択します。

ID	Version
該当する内容が存在しません。	

詳細設定

すべてを開く

Upgrade	<input type="checkbox"/>
Only Distribute	<input type="checkbox"/>
Concurrent	<input type="checkbox"/>

Options

Classloading	<div></div>
Use Fast Deploy	<input type="checkbox"/>
Keep Generated	<input type="checkbox"/>
Shared	<input type="checkbox"/>
Staging	<input type="checkbox"/>
Security Domain Name	<div></div>
Auto Redeploy Interval	<div>ms</div>
Plan	<div></div>
Context Path	<div></div>
Node JAVA Context	<input type="checkbox"/>

確認

キャンセル

## 参考

WebAdminでのアプリケーション・デプロイ設定の詳細内容については、『*JEUS アプリケーション&デプロイメントガイド*』の「第4章 アプリケーションの作成およびデプロイ」のデプロイ項目を参照してください。

- デプロイされたアプリケーションは、「**Command**」列に[stop]、[undeploy]、[redeploy]、[add-target]、[remove-target]ボタンが作成され、制御が可能になります。各ボタンの機能と制御についての詳細内容は、「[3.5.1. EJBモジュールの制御](#)」を参照してください。

[図 3.5] アプリケーションのデプロイ - 結果



### ● コンソール・ツールの使用

デプロイの手順は以下のとおりです。

1. コンソール・ツールを実行します。コンソール・ツールについての詳細説明は、『JEUS リファレンスガイド』の「4.2.2. ローカル・コマンド」を参照してください。

```
$ jeusadmin -host localhost:9736
```

2. ユーザー名とパスワードを入力します。

```
User name: administrator
Password:
Attempting to connect to localhost:9736.
The connection has been established to Domain Administration Server adminServer
in the domain domain1.
JEUS8 Administration Tool
To view help, use the 'help' command.
[DAS]domain1.adminServer>
```

3. **deploy**コマンドを実行して「countmod」EJBモジュールをデプロイします。

```
[DAS]domain1.adminServer> deploy countmod -servers server1
```

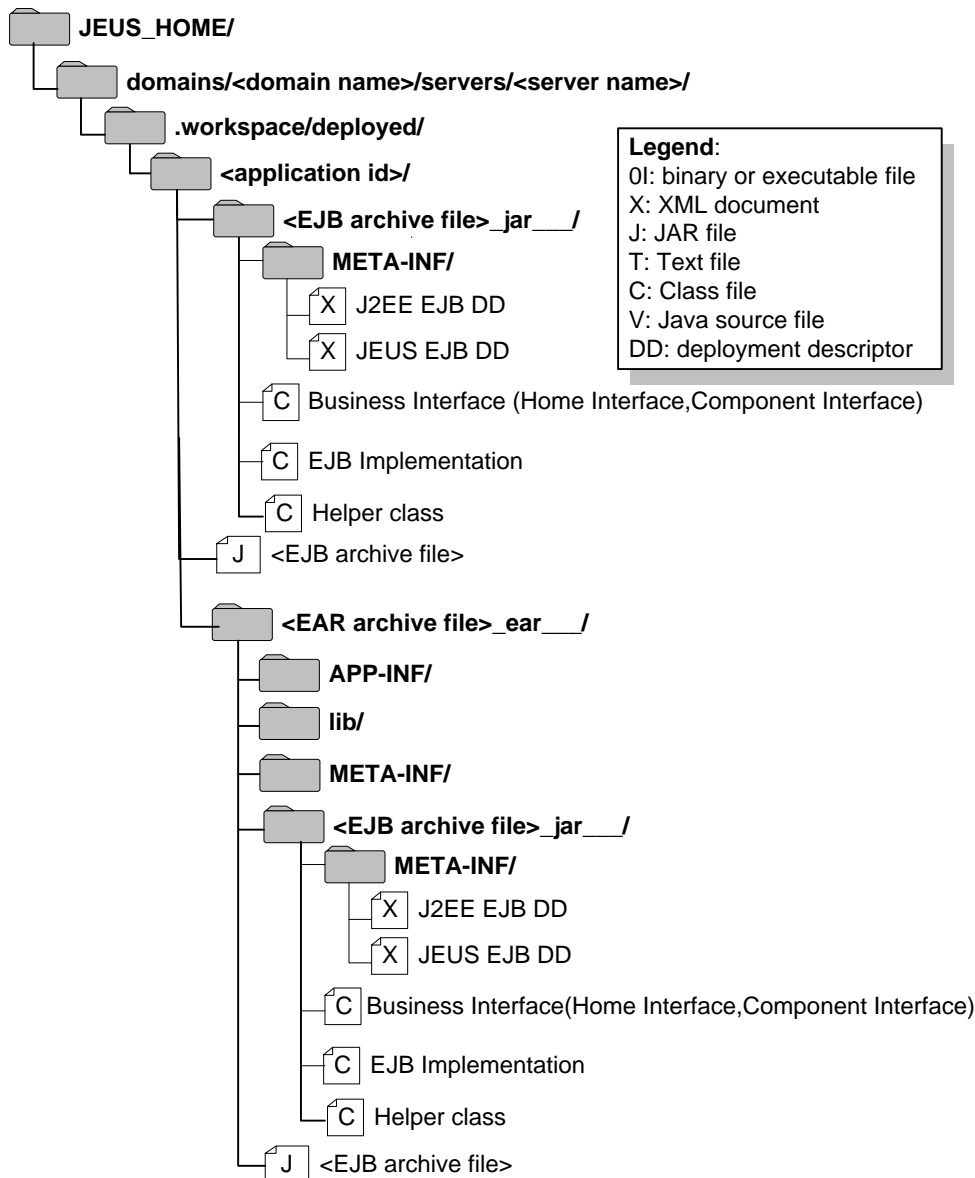
4. デプロイを確認するため、以下のように**application-info**コマンドを実行すると「countmod」モジュールが含まれたすべてのデプロイ済みのアプリケーションが出力されます。

```
[DAS]domain1.adminServer> application-info
```

### 3.4.2. デプロイされたEJBモジュールのディレクトリー構造

EJBモジュールがデプロイされた後、クラス・ファイルと設定ファイルは以下のような構造でJEUSインストール・ディレクトリーにデプロイされます。

[図 3.6] デプロイされたEJBモジュールのディレクトリー構造



JEUS\_HOME/domains/<domain name>/servers/<server name>/.workspace/deployed/ディレクトリーは<server name>に該当するMSにデプロイされたアプリケーションが存在するディレクトリーです。同ディレクトリーには、デプロイされたEJBモジュールのEJB実装クラスのみならず、helperクラスが存在します。

デプロイ方式によってデプロイされるファイルが格納されるディレクトリーは異なります。以下は、デプロイ方式についての説明です。

デプロイ方式	説明
EARデプロイ方式	EARファイルがデプロイされる場合は、EARファイル名で生成されたディレクトリーの下位にEARに含まれたモジュールがそれぞれ格納されます
スタンドアローン・デプロイ方式	JARファイルがデプロイされる場合は、JARファイル名で生成されたディレクトリーの「<jarファイル名>_jar_____」の下位に格納されます
Exploded EARデプロイ方式	EARファイルをアーカイブ形式ではなく、解凍した状態でデプロイする方式です
Explodedスタンドアローン・デプロイ方式	JARファイルをアーカイブ形式ではなく、解凍した状態でデプロイする方式です

Exploded EARデプロイ方式とExplodedスタンドアローン方式は、webhomeディレクトリーにコピーされず、元の場所を参照します。JEUSではアーカイブ・ファイルが存在するスタンドアローン・デプロイメントを**COMPONENTタイプ**、解凍したものを**EXPLODED COMPONENTタイプ**といいます。

#### 参考

以降からは、スタンドアローン・モジュールについてのみ説明します。EARデプロイ方式やデプロイの全般に関する詳細説明は、『JEUS アプリケーション&デプロイメントガイド』のデプロイ項目を参照してください。

## 3.5. EJBモジュールの制御およびモニタリング

デプロイされたEJBモジュールに対し、実行状態を制御するかモニタリングすることができます。([図 3.1]を参照)

### 3.5.1. EJBモジュールの制御

デプロイされたEJBモジュールに対し、stop-application、start-application、redploy-application、undeployコマンドを使用してEJBモジュール内の実行状態を制御することができます。EJBモジュールの制御には、以下の2つの方法があります。

- WebAdminを使用したEJBモジュールの制御
- コンソール・ツールを使用したEJBモジュールの制御

本節では、undeployコマンドの使用例を通じてEJBモジュールを制御する方法について説明します。

## WebAdminの使用

WebAdminを使用してEJBモジュールが制御できます。

以下は、WebAdminを使用したアンデプロイ手順についての説明です。

1. WebAdminの[**Applications**]メニューをクリックすると、[**Deployed Application**]画面にアプリケーション・リストが照会されます。デプロイされたアプリケーションの「**Command**」列には以下のようなコントロール・ボタンが作成されます。各ボタンはコンソール・ツールのオプションとそれぞれ対応されます。

アプリケーションをアンデプロイするときは、照会されたアプリケーション・リストから目的のアプリケーションの「**Command**」列に存在する[**undeploy**]ボタンをクリックします。

**[図 3.7] デプロイされたアプリケーションのリスト**

Application ID	Application Type	State	Command
countmod	EJB	RUNNING	Redeploy Add Target Remove Target

Start Stop Deploy Undeploy Uninstall



2. 以下のように[Undeploy]画面が表示されます。リクエスト中のサービスが完了されるまでの待機時間である「Timeout」項目を設定し、[確認]ボタンをクリックするとアンデプロイが実行されます。

【図 3.8】アプリケーションのアンデプロイ - 属性の設定

The image shows a dialog box titled "Undeploy" with a close button (X) in the top right corner. It contains four configuration sections:

- Timeout:** A text input field. Below it, a description reads: "アンデプロイをするとき、要求中のサービスが存在する場合は、その処理が完了するまでアンデプロイ要求が待機される時間です。"
- Graceful:** A dropdown menu. Below it, a description reads: "ドメインに旧アプリケーションと新アプリケーションが両方存在する場合、中止するアプリケーションを指定します。このオプションはWebアプリケーションの場合のみ適用されます。"
- Force:** A checkbox. Below it, a description reads: "アプリケーションをグレースフルアンデプロイせずにアンデプロイします。"
- Concurrent:** A checkbox. Below it, a description reads: "MSで選択したアプリケーションを同時にアンデプロイします。"

At the bottom right, there are two buttons: "確認" (Confirm) and "キャンセル" (Cancel).

#### 参考

「Graceful」項目は、グレースフル再デプロイが行われるとき、旧アプリケーションと新アプリケーションのうち、いずれをアンデプロイするかを決める設定です。

3. 正常にアンデプロイされると、以下のような結果メッセージが表示され、該当するアプリケーションの「State」と「Command」が変更されたことが確認できます。

【図 3.9】アプリケーションのアンデプロイ - 結果

The image shows a window titled "Deployed Application" with a "HISTORY" dropdown in the top right. Below the title bar, there is a search bar and icons for search, print, and refresh. A message box states: "ドメインでサービスするアプリケーションの照会およびデプロイコマンドをサポートします。Lock & Editモードでは一部のコマンドが使用できません。" with a "ヘルプ" (Help) button. Below this, a log entry shows: "Undeploying [countmod] (This may take time due to graceful undeployment) ..... undeploy the application for the application [countmod] succeeded."

Below the log, there are two tabs: "Applications" (selected) and "Deployed Library". Under the "Applications" tab, the title is "Application information for the domain [domain1].". Below this is a table:

	Application ID	Application Type	State	Install	Deploy	Plan
<input type="checkbox"/>	countmod	EJB	INSTALLED			

At the bottom right, there are buttons: Start, Stop, Deploy, Undeploy, and Uninstall.

## コンソール・ツールの使用

コンソール・ツールでは以下のようなコマンドを使ってEJBモジュールを制御します。すべてのコマンドには、EJBモジュールのIDをパラメータとして取得します。

- **redeploy-application**

redeploy-applicationコマンドは、指定したEJBモジュールをリロードするコマンドです。現在有効になっているEJBモジュールをEJB実行環境でアンデプロイしてから、再びディスクから読み込みデプロイします。

アンデプロイが実行されリロードが行われると、アンデプロイされたEJBモジュールのすべてのトランザクション状態が失われます。これは、アンデプロイまたは再デプロイされるモジュールに関連するすべてのEJBトランザクションがロールバックされることを意味します。

```
redeploy-application <application-id>
```

- **stop-application**

stop-applicationコマンドは指定したEJBモジュールを一時停止させ、クライアントのリクエストが一時的にEJBモジュールにアクセスできない状態にします。start-applicationコマンドを使用してEJBモジュールにアクセス可能な状態に戻します。

```
stop-application <application-id>
```

- **start-application**

stop-applicationコマンドによって停止されていたEJBモジュールを再び有効にします。

```
start-application <application-id>
```

- **undeploy**

undeployコマンドは、指定したモジュールをEJBエンジンの実行メモリーから除去し、EJBクライアントがEJBにアクセスできないようにします。ただし、物理的なファイルの削除は行いません。

```
undeploy <application-id>
```

以下は、コンソール・ツールを使ってEJBモジュールをアンデプロイする方法です。

1. コンソール・ツールを実行します。

```
$ jeusadmin -host localhost:9736
```

2. ユーザー名とパスワードを入力します。

```
User name: administrator
Password:
```

```
Attempting to connect to localhost:9736.  
The connection has been established to Domain Administration Server adminServer  
in the domain domain1.  
JEUS8 Administration Tool  
To view help, use the 'help' command.  
[DAS]domain1.adminServer>
```

3. 以下のように**undeploy**コマンドが正常に実行されると、「countermod」EJBモジュールがアンデプロイされます。

```
[DAS]domain1.adminServer> undeploy countermod
```

4. アンデプロイを確認するには、以下のように**application-info**コマンドを実行します。「countermod」モジュールを除いてデプロイされたすべてのモジュールが照会されます。

```
[DAS]domain1.adminServer> application-info
```

---

#### 参考

コンソール・ツールについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2. jeusadmin」を参照してください。

---

### 3.5.2. EJBモジュールのモニタリング

コンソール・ツールを使用してEJBモジュールをモニタリングできます。コンソール・ツールでapplication-infoコマンドを実行すると以下のようにデプロイされ、有効になったEJBモジュールから状態と運用情報が照会できます。application-infoコマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.7. EJBエンジン関連コマンド」を参照してください。

#### ● モジュール情報の照会

モジュール情報を照会します。

```
[DAS]domain1.adminServer>application-info -server server1 -id countermod -detail  
General information about the EJB module [countermod].  
=====
```

Module Name	Unique Module Name
countermod	countermod

```
=====
```

Beans				
=====				
+-----+-----+-----+-----+				
Bean Name	Type	Local Export Name	Remote Export Name	
+-----+-----+-----+-----+				
Count	StatelessSessionBean		Count	
+-----+-----+-----+-----+				
=====				

## ● Beanの照会

モジュール(module name)に含まれたEJBリストを照会します。

[DAS]domain1.adminServer>application-info -server server1 -id countermod -bean Count					
Module name : countermod					
Bean name : Count					
=====					
+-----+-----+-----+-----+-----+					
Name	(Count)	WaterMark(High:Low	Bound(Upper:L	Time(Max:Min:T	
		:Cur)	ower)	otal)	
+-----+-----+-----+-----+-----+					
create	times(0)				
+-----+-----+-----+-----+-----+					
comitted	transactio				
	n(0)				
+-----+-----+-----+-----+-----+					
total-remote-t		thread(100:100:100)			
hread					
+-----+-----+-----+-----+-----+					
timed-rb	transactio				
	n(0)				
+-----+-----+-----+-----+-----+					
remove	times(0)				
+-----+-----+-----+-----+-----+					
active-bean		bean(0:0:0)			
+-----+-----+-----+-----+-----+					
request	request(0)				
+-----+-----+-----+-----+-----+					
total-bean		bean(0:0:0)			
+-----+-----+-----+-----+-----+					
rolledback	transactio				
	n(0)				
+-----+-----+-----+-----+-----+					
active-thread		thread(0:0:0)			
+-----+-----+-----+-----+-----+					
MethodReadyCou		bean(0:0:0)			

nt				
+-----+	+-----+	+-----+	+-----+	+-----+
=====				

### 参考

EJBモジュールのモニタリングはWebAdminを使用することもできますが、コンソール・ツールを使用するとより詳しい情報が照会できます。WebAdminを使用した場合は、**[Monitoring]**メニューの**[JNDI]**または**[EJBTimer]**メニューを通じてEJBモジュールの一部情報がモニタリングできます。モニタリングについての詳細方法は、『*JEUS サーバガイド*』の「4.2.2. バインドされたオブジェクトの確認」と「[10.2. タイマー・モニタリング](#)」を参照してください。



## 第4章 EJBの共通特性

本章では、JEUSにおけるEJBの共通特性について説明します。

JEUSでEJBを使用するために必要な追加情報のみ記述しており、EJB標準については省略しています。  
EJBデプロイの詳細説明については、「[第3章 EJBモジュール](#)」を参照してください。

### 4.1. 概要

本節では、EJBの種類と種類別のDD設定について説明します。

### EJBの種類

JEUSは、EJB仕様により以下のEJBをサポートします。

- ステートレス・セッションBean :「[第7章 セッションBean](#)」
- ステートフル・セッションBean :「[第7章 セッションBean](#)」
- BMPエンティティBean :「[第8章 エンティティBean](#)」
- CMP 1.1 & CMP 2.x Entity Bean :「[第8章 エンティティBean](#)」
- メッセージ駆動型Bean :「[第9章 メッセージ駆動型Bean\(MDB\)](#)」

本章では、すべてのBeanに適用される共通した特性と設定について説明します。各EJBの追加的な詳細情報については該当する章を参照してください。

---

#### 参考

エンティティBeanはEJB 3.0からはJPAに代替されたため(『JEUS JPAガイド』を参照)、エンティティBeanの使用は推奨しませんが、下位互換性を保持するために引き続きサポートしています。

---

## EJBの種類別のDD設定

JEUS EJB DD(Deployment Descriptors)はjeus-ejb-dd.xmlと命名されたXMLファイルです。EJB JARファイルのMETA-INFの下位に存在します。JEUS EJB DDについての簡単な内容は、「[第3章 EJBモジュール](#)」を参照してください。

以下は、JEUS EJB DDで設定可能な機能とコンポーネントの簡単な情報です。(' @ 'はアノテーションで設定可能なオブジェクトです。)表の左側の項目は、実際に設定可能なJEUS EJB DDのXMLタグです。以外の列の6項目は、EJBの種類別の設定事項です。

**[表 4.1] JEUS EJBの設定可能な特性およびコンポーネント**

設定可能なオブジェクト	Stateless	Stateful	BMP	CMP1.1	CMP2.0	MDB
1. EJB name @	√	√	√	√	√	√
2. Export name @	√	√	√	√	√	
3. Local export name @	√	√	√	√	√	
4. Export port	√	√	√	√	√	
5. Export IIOP switch	√	√	√	√	√	
6. use-access-control	√	√	√	√	√	√
7. Run-as Identify	√	√	√	√	√	√
8. Security CSI Interop.	√	√	√	√	√	√
9. Env. Refs @	√	√	√	√	√	√
10. EJB refs @	√	√	√	√	√	√
11. Resource Refs @	√	√	√	√	√	√
12. Resource env. Refs @	√	√	√	√	√	√
13. Thread ticket pool settings	√	√	√	√	√	√
14. Clustering settings @	√	√	√	√	√	
15. HTTP Invoke	√	√	√	√	√	
16. JEUS RMI	√	√	√	√	√	
17. Pooling Bean Switch		√				
18. Object management	√	√	√	√	√	√
19. Persistence Optimize			√	√	√	
20. CM persistence opt.				√	√	
21. Schema info				√	√	
22. Relationship map					√	
23. Connect. fact. name						√
24. MDB resource adaptor						√



設定可能なオブジェクト	Stateless	Stateful	BMP	CMP1.1	CMP2.0	MDB
25. Destination @						√
26. JNDI SPI settings						√
27. Max messages						√
28. Activation Config @						√
29. Durable Timer Service	√		√	√	√	√

上記の表は以下のように分けられます。

- 項目1から16までは、6つのEJB種類に共通して適用されます。(MDBの2~5と14~16は除外)各項目については本章で説明します。
- 項目7(Run-as Identify)と8(security interoperability)、14(clustering)は、すべてのEJB種類に共通します。それぞれ「[4.2.6. EJBのセキュリティー設定](#)」,「[第5章 EJBの相互運用性およびRMI/IIOP](#)」,「[第6章 EJBのクラスタリング](#)」で別途説明します。
- 項目17は、ステートフル・セッションBeanにのみ適用されます。「[第7章 セッションBean](#)」で詳しく説明します。
- 項目18(object management)は、6つのEJB種類すべてに共通して適用されますが、Beanの種類によって適用内容が若干異なります。基本的な内容については、「[第7章 セッションBean](#)」で説明し、各章でその相違点について記述します。
- 項目19から22までは、エンティティBeanにのみ適用されます。詳細については、「[第8章 エンティティBean](#)」で説明します。
- 項目23から28までは、MDBにのみ適用されます。詳細については、「[第9章 メッセージ駆動型Bean\(MDB\)](#)」で説明します。
- 項目29は、EJB仕様で提供するタイマー・サービスが使用できるBean(ステートフル・セッションBeanを除くすべてのBean)で使用可能です。詳細については、「[第10章 EJBタイマー・サービス](#)」で説明します。

## 4.2. EJBの設定

すべてのEJB設定は、JEUS EJB DDファイルのjeus-ejb-dd.xmlファイル内に設定されます。上位XMLタグはBeanの種類を問わず<jeus-bean>を使用します。

次の節では、上記の5種類のEJB XMLの親タグに適用できる設定について説明します。<beanlist>の外に存在するタグについては、「[第3章 EJBモジュール](#)」に記述されています。特定のXMLの親タグ(Beaの種類)についての説明は、該当する章を参照してください。

---

## 参考

クラスタリングの環境設定は、「[第6章 EJBのクラスタリング](#)」を参照してください。

---

### 4.2.1. 基本的な環境設定

以下は、主に使用されるJEUS専用の設定であり、jeus-ejb-dd.xmlに宣言されています。同ファイルについては、「[第3章 EJBモジュール](#)」を参照してください。

- **<ejb-name>**

- 標準ejb-jar.xmlで<ejb-name>または実装クラスの@Stateless、@Stateful、@MessageDrivenの「name」値と同じ名前です。

- **<export-name>**

- グローバルJNDIに登録されるシステム内部において一意の名前である必要があります。
- DDファイルにこの要素が存在しない場合、Annotation @Statelessまたは@StatefulのmappedName attribute値が適用されるか、ejb-jar.xmlの<mapped-name>値が適用されます。

設定しない場合はデフォルト値のリモート・ビジネス・インタフェース名が使用されます。ただし、リモート・ビジネス・インタフェースが1つのみ存在し、リモート・ホーム・インタフェースが存在しない場合や、リモート・ビジネス・インタフェースが存在せず、リモート・ホーム・インタフェースが存在する場合のみデフォルト値が適用されます。以外の場合はJNDI名が決められずデプロイに失敗します。

- 順番は以下のとおりです。

1. jeus-ejb-dd.xmlの<export-name>
2. ejb-jarの<mapped-name>
3. @EJBのmappedName

- **<local-export-name>**

- JNDIに登録されるシステム内部において一意の名前である必要があります。
- DDファイルにこの要素が存在しない場合、Annotation @EJB mappedName attribute値に適用されます。リモート・ビジネス・インタフェースが存在する場合は、この値に「\_Local」が付きます。

両方とも存在しない場合は、デフォルト値のローカル・ビジネス・インタフェース名が使用されます。ただし、ローカル・ビジネス・インタフェースが1つのみ存在し、ローカル・ホーム・インタフェースが存在しない

場合や、ローカル・ビジネス・インタフェースが存在せず、ローカル・ホーム・インタフェースが存在する場合のみデフォルト値が適用されます。以外の場合はJNDI名が見つからず例外が発生します。

– 順番は以下のとおりです。

1. jeus-ejb-dd.xmlの<local-export-name>
2. Remote Business Interfaceが存在すると、ejb-jar.xmlの<mapped-name>+ \_Local  
Remote Business Interfaceが存在しない場合は、ejb-jar.xmlの<mapped-name>
3. Remote Business Interfaceが存在すると、@EJBのmappedName + \_Local  
Remote Business Interfaceが存在しない場合は、@EJBのmappedName

● <export-port>

- RMIサービス・ポートを明示的に設定する必要がある場合のみ使用します。
- ファイアウォールを使用しており、特定のポートにのみシステム・アクセスを許可する場合に有用です。このタグの設定は、ファイアウォール設定の「allowed」に設定されたポートと同じものである必要があります。値を指定しない場合は、以下のようにシステム・プロパティによって一括にポートが適用されます。

ポート	説明
jeus.ejb.exportPort	jeus-ejb-ddに特定のEJBに対する<export-port>が指定されていない、別のEJBのエクスポート・ポートを指定します
jeus.rmi.usebaseport	値がtrueの場合は管理対象サーバーのベース・ポートを使用し、falseの場合は管理対象サーバーのベース・ポート+7を使用します

● <export-iiop>

- 有効の場合、BeanのインターフェースがIIOPスタブとスケルトンとしてCOSネーミング・サーバーにエクスポートできるようにします。これは、IIOPにアクセス可能なすべてのクライアントBeanにアクセスできるようにします。

● <use-access-control>

- ブーリアン設定はBeanメソッドを実行するとき、Java EEのプリンシパルによってメソッドの実行がJava SEセキュリティ・マネージャーのモニタリングを受けるか否かを決めます。詳細内容は、『JEUSSセキュリティガイド』とJACC仕様書を参照してください。(デフォルト値: false)

以下は、上記の設定がステートフル・セッションBeanクラスとDDに適用された例です。対応されるアノテーションとDDの要素は太字で表記しています。

**[例 4.1] ステートフル・セッションBeanクラスとDD : <<CounterEJB.java>>**

```
package ejb.basic.statefulSession;

@Stateful(name="counter", mappedName="counterEJB")
public class CounterEJB implements Counter, CounterLocal
{
    private int count = 0;

    public void increase()
    {
        count++;
    }
    ...
}
```

**[例 4.2] ステートフル・セッションBeanクラスとDD : <<jeus-ejb-dd.xml>>**

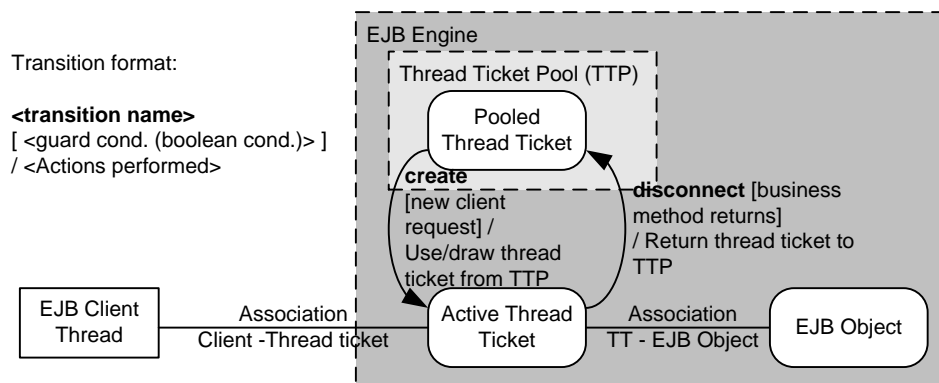
```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            <ejb-name>counter</ejb-name>
            <export-name>counterEJB</export-name>
            <local-export-name>counterEJB_Local</local-export-name>
            <export-port>7654</export-port>
            <export-iiop/>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

## 4.2.2. スレッド・チケットの設定

以下の図は、EJB Thread Ticket Pool(以下、TTP)の概念です。

基本的な動作は、クライアントから要求を受けると、新しいRMIスレッドをクライアント要求に割り当てることです。このとき、割り当てられたRMIスレッド数が設定されている最大値より多い場合は、当該要求にスレッドを割り当てずに待機します。

【図 4.1】 TTPのプロセス



上図で確認できるように、TTPはThread Ticket(以下、TT)を有します。1つのTTは1つのクライアント・スレッドがEJBエンジンにアクセスできるように許可します。クライアントから要求を受け取ると、TTPから1つのTTが取得されクライアントに割り当てられます。これは、各クライアントのスレッドは、EJBインスタンスに要求を渡す1つのEJBオブジェクトに割り当てられることを意味します。すなわち、EJBオブジェクトはEJBエンジンに存在する1つのオブジェクトとして、EJBクライアントとEJBインスタンスとの接続を管理します。

TTPに存在するTT数よりクライアントからの要求が多い場合、新しい要求はTTがフリーになってプールにリターンされるまで待機します。クライアントの要求がTTを待機した時間が10分を超えると、時間超過によってRemoteExceptionが発生します。

EJB処理が終了するか、(ステートレス・セッションBean)コネクション・タイムアウトが発生し、クライアントの接続が切断されるとTTがTTPに返されます。また、TTPはリモート呼び出しが可能なEJBコンポーネントを1つずつ持ちます。以外にコネクション・プールとBeanプールがあります。MDBにおいてはTTPではなくBeanプールを設定します。

## 参考

コネクション・プールとBeanプールについての詳細内容は、「[第7章 セッションBean](#)」と「[第8章 エンティティBean](#)」を参照してください。MDBでのBeanプールの使用については、「[第9章 メッセージ駆動型Bean\(MDB\)](#)」を参照してください。

EJBのTTは、BeanのルートXMLタグである<thread-max>で設定します。この値は、各EJBを実行するスレッドの最大許可数、すなわちTTの最大数です。デフォルト値は100です。待機しているクライアントの要求数がこの値より多い場合、新しい要求は先行作業が完了し、プールにTTが返されるまで待機します。待機から10分が経過するとRemoteExceptionが発生します。

以下は、BMP Beanに上記の設定を適用して例です。

## 【例 4.3】 BMP Beanの設定 : <<jeus-ejb-dd.xml>>

```

<jeus-ejb-dd>
  . . .

```

```

    <beanlist>
      <jeus-bean>
        . . .
        <thread-max>200</thread-max>
        . . .
      </jeus-bean>
      . . .
    </beanlist>
    . . .
  </jeus-ejb-dd>

```

### 4.2.3. 外部参照の設定とマッピング

Beanが使用する外部参照はアノテーションを利用して設定するか、またはejb-jar.xmlで使用した参照名とマッピングされたjeus-ejb-dd.xmlのJNDIエクスポート名を使用して設定できます。宣言されたすべての外部参照は、JNDIエクスポート名とマッピングされる必要があります。

アノテーションやejb-jar.xmlに宣言された外部参照は、各参照に該当するjeus-ejb-dd.xmlのタグに実際システムのJNDIエクスポート名でマッピングされる必要があります。

以下は、ejb-jar.xmlとJEUS EJB DDファイルの参照タグについての説明です。

**[表 4.2] ejb-jar.xmlとJEUS EJB DDファイルの参照タグとその関係**

タグ	EJB DD XMLタグ	説明
<env-entry>	<env>	EJB環境設定を定義するか無視します
<ejb-ref>	<ejb-ref>	EJB参照を実際のエクスポート名にバインドします
<resource-ref>	<res-ref>	外部リソース管理の参照を実際のエクスポート名にバインドします
<resource-env-ref>	<res-env-ref>	管理されているオブジェクト参照を実際のエクスポート名にバインドします
<message-destination-ref>	<message-destination-ref>	メッセージ・デスティネーションの参照を実際のエクスポート名にバインドします
<service-ref>	<service-ref>	Webサービス・エンドポイントの参照を実際のエクスポート名にバインドします

すべての<env>タグには、以下の下位タグの設定が必要です。

タグ	説明
<name>	env名です。ejb-jar.xmlに定義された<env-entry-name>タグの値と同じ値を使用します

タグ	説明
<type>	値のデータ型に該当する完全なJavaクラス名です。デフォルトのデータ型は、wrapperクラスを使用します
<value>	envの実際値を宣言します

他のタグ(<ejb-ref>、<res-ref>、<res-env-ref>)は、複数の<jndi-info>タグを持ちます。このタグは、JNDIエクスポート名に参照をマッピングするため、以下のような下位タグを設定する必要があります。

タグ	説明
<ref-name>	EJBソース・コードとejb-jar.xmlで定義された参照名です。ejb-jar.xmlにおいて<ejb-ref>と同格のタグは<ejb-ref>です。<resource-ref>は<res-ref>と、<resource-env-ref>は<res-env-ref>と同格です
<export-name>	JNDIにエクスポートされるときに使用される名前です

## 参考

<ref-name>と<export-name>タグは、必ずejb-jar.xmlに設定された名前と同じである必要があります。

以下は、外部参照をJNDIにマッピングする方法として、アノテーションとXMLの例です。

### [例 4.4] 外部参照をJNDIにマッピング : <<CounterEJB.java>>

```
@Stateful

@EJB(name="AccountEJB", beanInterface=ejb.Account.class, mappedName="ACCEJB")
@Resource(name="jdbc/DBDS", type=javax.sql.DataSource.class, mappedName="ACCOUNTDB")
public class CounterEJB implements Counter{
    @Resource(name="minAmount")
    private int minAccount = 100;

    . . .
}
```

### [例 4.5] 外部参照をJNDIにマッピング : <<ejb-jar.xml>>

```
<ejb-jar.xml>
. . .
<enterprise-beans>
    <session>
        . . .
        <env-entry>
            <env-entry-name>minAmount</env-entry-name>
            <env-entry-type>java.lang.Integer</env-entry-type>
```

```

        </env-entry>
        <ejb-ref>
            <ejb-ref-name>AccountEJB</ejb-ref-name>
            <ejb-ref-type>Session</ejb-ref-type>
            <remote>ejb.Account</remote>
        </ejb-ref>
        <resource-ref>
            <res-ref-name>jdbc/DBDS</res-ref-name>
            <res-ref-type>javax.sql.DataSource</res-ref-type>
        </resource-ref>
        . . .
    </session>
</enterprise-beans>
. . .
</ejb-jar.xml>

```

#### [例 4.6] 外部参照をJNDIにマッピング : <<jeus-ejb-dd.xml>>

```

<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <env>
                <name>minAmount</name>
                <type>java.lang.Integer</type>
                <value>100</value>
            </env>
            <ejb-ref>
                <jndi-info>
                    <ref-name>AccountEJB</ref-name>
                    <export-name>ACCEJB</export-name>
                </jndi-info>
            </ejb-ref>
            <res-ref>
                <jndi-info>
                    <ref-name>jdbc/DBDS</ref-name>
                    <export-name>ACCOUNTDB</export-name>
                </jndi-info>
            </res-ref>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>

```



# 4.2.4. HTTP呼び出し(Invoke)の環境設定

EJBモジュールのうち、特定のEJBにHTTP呼び出しを設定して使用するには、jeus-ejb-dd.xmlの<jeus-bean>下位に<invoke-http>を設定する必要があります。EJBエンジンでHTTP呼び出しが必要な場合は、EJBMain.xmlで<invoke-http>を追加します。(「第2章 EJBエンジン」で説明) この設定は、すべてのモジュールに適用されますが、各モジュールのDDファイルに<invoke-http>が存在する場合はファイルの設定を使用します。

以下は、jeus-ejb-dd.xmlのHTTP呼び出しの環境設定の例です。

**[例 4.7] HTTP呼び出しの環境設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
<beanlist>
  <jeus-bean>
    . . .
    <invoke-http>
      <url>
        /mycontext/RMIServletHandler
      </url>
      <http-port>
        80
      </http-port>
    </invoke-http>
    . . .
  </jeus-bean>
</beanlist>
. . .
</jeus-ejb-dd>
```

<invoke-http>設定には、2つの下位タグが存在します。

タグ	説明
<url>	HTTP-RMIスタブから呼び出されるRMIハンドラー・サーブレット(jeus.rmi.http.ServletHandler)のURIを入力します。このURIでは、プロトコル、IP、ポートを除いたサーブレット要求パスのみ入力します。  プロトコルはHTTPに、IPはRMIランタイムと同様なアドレスと見なされます。すなわち、HTTP-RMI要求を受けたWebサーバーとWebエンジンがRMIランタイムと同じマシンに存在する必要があります。そうすると、RMIランタイムのアドレスはRMIスタブに通知されます。Webサーバーのポートは必ず次の<http-port>に設定します
<http-port>	HTTP-RMI要求を受けるポート番号を設定します。WebサーバーおよびWebエンジンには、RMIハンドラー・サーブレットがデプロイされ、実行されている必要があります

---

#### 参考

HTTP呼び出しinvokeが正常に動作するには、RMIハンドラー・サーブレットがデプロイされている必要があります。RMIハンドラー・サーブレットを実装したクラスは、jeus.rmi.http.ServletHandlerです。jeus.rmi.http.ServletHandlerは、コンテキスト内の<invoke-http>に設定されたURLと同様にデプロイされる必要があります。デプロイについての詳細内容は、『JEUS Webエンジンガイド』を参照してください。

---

### 4.2.5. JEUS RMIの設定

JEUSでは、性能を高めたJEUS RMI通信を提供します。このRMIは非ブロッキングIOでも通信ができるため、クライアント数が多い場合に有効です。JEUS RMIを使用するには、クライアント・ライブラリーのJEUS\_HOME/lib/client/jclient.jarをクラス・パスに設定するだけで、別途の作業は不要です。

JEUS RMIの使用有無は、jeus-ejb-dd.xmlにBean別に設定できます。JEUS全体にシステム・プロパティ(-Djeus.ejb.rmi.jeus=true)として適用することも可能です。その際、DDの設定が優先します。

JEUS RMIをブロッキングIOにするか、非ブロッキングIOにするかの設定や、チャンネルとソケットの使用有無など、通信レイヤーの詳細はシステム・プロパティとして設定できます。詳細説明については、『JEUS リファレンスガイド』を参照してください。

### 4.2.6. EJBのセキュリティー設定

本節では、JEUSでEJBのためのセキュリティー設定方法について説明します。こちらでのセキュリティーとは、認証と権限付与のことをいいます。

JEUS EJBのセキュリティー設定と関連する項目は以下のとおりです。

- EJBモジュールのロール割り当て(Role Assignment)
- EJB Run-as Identifyの設定

---

#### 参考

EJB設定で使用するJEUSのユーザー一覧は、accounts.xmlに定義されています。詳細設定については、『JEUS セキュリティガイド』を参照してください。

---

#### 4.2.6.1. セキュリティー設定

以下は、状況別にセキュリティーを設定する方法です。

## ロール割り当て(Role Assignment)設定

EJBモジュールのロール割り当て設定とは、標準ejb-jar.xmlファイルに開発者が定義したロールとJEUSセキュリティ・ドメインのプリンシパル・グループ名をマッピングすることです。jeus-ejb-dd.xmlファイルの<module-info>の<role-permission>タグで設定します。

以下は、ロール割り当てを設定したXMLの例です

### [例 4.8] ロール割り当て設定 : <<jeus-ejb-dd.xml>>

```
<jeus-ejb-dd>
  <module-info>
    <role-permission>
      <role>manager</role>
      <principal>peter</principal>
    </role-permission>
  </module-info>
  <beanlist>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>
```

以下は、必須の下位タグについての説明です。

タグ	説明
<role>	EJBに設定された(アノテーションまたはejb-jar.xmlを利用)論理的なロール名を指定します
<principal>	accounts.xmlで定義された実際のユーザー名を指定します。role name-user nameのペアが、開発者が定義したロールとJEUSプリンシパル間のマッピングを定義します。1つのロール名に複数のユーザー名が設定できます

## Run-as Identify設定

Run-as Identify設定は、開発者がejb-jar.xmlの<run-as>で定義したロール名とJEUSセキュリティ・ドメイン設定で指定した実際プリンシパルのマッピングを提供します。(例: accounts.xmlファイルにて)

このプリンシパルは一般的に簡単なユーザー名を使用します。jeus-ejb-dd.xmlでBeanの設定レベルと同様な<run-as-identity>タグで設定します。

以下は、Run-as Identifyを設定したXMLの例です。

**[例 4.9] Run-as Identify設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        <jeus-bean>
            . . .
            <run-as-identity>
                <principal-name>peter</principal-name>
            </run-as-identity>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

タグ	説明
<principal-name>	ejb-jar.xmlファイルの<run-as>タグで与えられたロールにマッピングするユーザー名です。accounts.xmlに設定されている必要があります

## 4.2.6.2. セキュリティー設定の例

以下は、role-permissionのロールとrun-asのロールが実際のJEUSユーザーでマッピングされる例です。

JavaクラスとXML内容の一部は、3つの設定ファイル(ejb-jar.xml、jeus-ejb-dd.xml、accounts.xml)から取得したものです。これらのファイルで同様である必要のある部分は太字で強調しています。

**[例 4.10] セキュリティー設定 : <<CustomerBean.java>>**

```
@Stateless(name="CustomerEJB")
@RolesAllowed("CUSTOMER")
public class CustomerBean implements Customer {
    . . .
}
```

**[例 4.11] セキュリティー設定 : <<EmployeeServiceBean.java>>**

```
@Stateful(name="EmployeeService")
@RunAs("ADMIN")
public class EmployeeServiceBean implements EmployeeService{
    . . .
}
```

**[例 4.12] セキュリティー設定 : <<ejb-jar.xml>>**

```
<?xml version="1.0"?>
<ejb-jar version="3.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd">
  <enterprise-beans>
    <session>
      <ejb-name>CustomerEJB</ejb-name>
      . . .
      <security-role-ref>
        <role-name>CUSTOMER</role-name>
      </security-role-ref>
      <security-identity>
        <use-caller-identity />
      </security-identity>
    </session>
    <session>
      <ejb-name>EmployeeService</ejb-name>
      . . .
      <security-identity>
        <run-as>
          <role-name>ADMIN</role-name>
        </run-as>
      </security-identity>
    </session>
    . . .
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>CUSTOMER</role-name>
    </security-role>
    <method-permission>
      <role-name>CUSTOMER</role-name>
      <method>
        <ejb-name>CustomerEJB</ejb-name>
        <method-name>*</method-name>
        <method-params />
      </method>
    </method-permission>
    . . .
  </assembly-descriptor>
  . . .
</ejb-jar>
```

上記の例では、2つのセッションBeanを宣言しました。

- 1つ目のセッションBean(CustomerEJB)は、「CUSTOMER」ロールに連結されるセキュリティー・ロールを持ちます。
- 2つ目のセッションBean(EmployeeService)は、「ADMIN」という<run-as>ロールを宣言しています。「CUSTOMER」と「ADMIN」ロールは、EJBのデプロイ時に実際システムのプリンシパルとマッピングされる必要があります。

以下のXML DDファイルは、「CUSTOMER」と「ADMIN」ロールが実際システムのプリンシパル名(それぞれ「customer」と「peter」)にマッピングされる例です。

**[例 4.13] セキュリティー設定 : <<jeus-ejb-dd.xml>>**

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    . . .
    <role-permission>
      <principal>customer</principal>
      <role>CUSTOMER</role>
    </role-permission>
  </module-info>
  <beanlist>
    <jeus-bean>
      <ejb-name>EmployeeService</ejb-name>
      <run-as-identity>
        <principal-name>peter</principal-name>
      </run-as-identity>
    </jeus-bean>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>
```

以下の例は、ユーザーがaccounts.xmlに設定されている内容です。

**[例 4.14] セキュリティー設定 : <<accounts.xml>>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>customer</ name>
      <password>{SHA}b1SUPYpdjb8QDcq+ozfbIEZx5oY=</name>
      <group>test</group>
    </user>
```

```

        <user>
            <name>peter</ name>
            <password>{SHA}McbQlyhI3yiOG1HGTg8DQVWkyhg=</name>
            <group>test</group>
        </user>
    </users>
</accounts>

```

上記の例で、「customer」と「peter」というユーザーの定義が確認できます。これらのユーザーは、jeus-ejb-dd.xmlファイルを通じてejb-jar.xmlファイルの「CUSTOMER」と「ADMIN」のロールと連携されます。

## 4.3. EJBのモニタリング

JEUSではEJBモジュールに含まれた個別のEJBを制御する方法は提供しません。ただし、コンソール・ツールを使用したBean別のモニタリングは可能です。

コンソール・ツールでは、application-infoコマンドを使ってBeanの名前、エクスポート名、状態など、個別EJBの情報が照会できます。以下の例では、「countermod」というEJBがデプロイされていると仮定します。

以下のようにコマンドを実行すると、EJBモジュール内の各Beanの名前とエクスポート名が確認できます。

```

[DAS]domain1.adminServer>application-info -server server1 -id countermod -detail
General information about the EJB module [countermod].
=====
+-----+-----+-----+
| Module Name | Unique Module Name |
+-----+-----+-----+
| countermod | countermod          |
+-----+-----+-----+
=====

Beans
=====

+-----+-----+-----+-----+
| Bean Name | Type | Local Export Name | Remote Export Name |
+-----+-----+-----+-----+
| Count | StatelessSessionBean | | Count |
+-----+-----+-----+-----+
=====

```

以下のようにコマンドを実行すると、指定したEJBモジュールに登録されている個別Beanの状態が確認できます。

```
[DAS]domain1.adminServer>application-info -server server1 -id countermod -bean
Count
Module name : countermod
Bean name : Count
=====
```

Name	(Count)	WaterMark(High:Low:Cur)	Bound(Upper:Lower)	Time(Max:Min:Total)
create	times(0)			
comitted	transactio n(0)			
total-remote-t hread		thread(100:100:100)		
timed-rb	transactio n(0)			
remove	times(0)			
active-bean		bean(0:0:0)		
request	request(0)			
total-bean		bean(0:0:0)		
rolledback	transactio n(0)			
active-thread		thread(0:0:0)		
MethodReadyCou nt		bean(0:0:0)		

```
=====
```

## 参考

application-infoコマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.6.3. application-info」を参照してください。



## 4.4. EJBのチューニング

以下は、EJBの運用性能を高める(または、システム・リソースの無駄遣いを防ぐため)ために、すべてのEJB種類に共通して適用可能な設定です。

- TTPの最大値を適切に修正します。
- 複数のEJBエンジンにBeanをクラスタリングして設定します。「第6章 EJBのクラスタリング」を参照してください。
- 適切なJDBCコネクション・プールを設定します。設定についての詳細内容は、『JEUS サーバガイド』を参照してください。
- EJB 2.xの場合、EJBモジュールのデプロイ時間を短縮するために、deployコマンドに-fastオプションとappcompilerを使用します。詳細内容については、「第3章 EJBモジュール」を参照してください。

上記の基本的な最適化オプションのみならず、Beanの種類によって様々なチューニング方法があります。EJB種類によるチューニングと最適化方法については、該当する章を参照してください。

### 4.4.1. TTP(Thread Ticket Pool)設定のチューニング

EJBが高性能を発揮するには、以下のルールを適用してTTPを正しく設定する必要があります。

- 最大プール・サイズを大きく設定すると、TTが多く生成されるため、より多くのクライアント要求が処理できます。しかし、大きく設定した最大プール・サイズにより、多数のクライアントがEJBエンジンにアクセスする場合は多くのメモリー・リソースが消耗されます。
- 特定の時間帯にBeanのサービス使用を要求するクライアントが急増する場合、最大プール・サイズから最小プール・サイズを引いた値の分だけ増やし、新しいTTが「batch」で生成されるように設定します。

TTPの設定は、特定のEJBに対してアクセス可能な量が調整できる役割をします。

---

#### 参考

最大プール・サイズはTTPにおいて最も重要なパラメータです。そのため、性能を改善するためにはこの値を優先して調整し、チューニングします。この値は<thread-max>に設定します。

---



# 第5章 EJBの相互運用性およびRMI/IIOP

本章では、EJBの相互運用性およびRMI/IIOPについて説明します。

## 5.1. 概要

JEUSは、他のWASや異機種システム間のリモートEJB呼び出しをサポートするためにRMI/IIOPプロトコルを提供します。RMI/IIOPはOMG(<http://omg.org>)で指定したCORBA(Common Object Request Broker Architecture)をベースにしており、EJBの相互運用性(Interoperability)のためにEJB仕様で定めた標準方式です。JEUS間または他のWASからのリモート呼び出しをサポートする場合、RMI/IIOP相互運用機能が使用できます。RMI/IIOPリモート呼び出しは、他のWASや異機種システムをサポートする方式の1つです。他のWASからJEUSのEJBを呼び出し、トランザクションを連動する(または逆に)別の方法も存在しますが、本章ではRMI/IIOP方式について説明します。

相互運用は、他システムがJEUSのEJBを呼び出す方法と、JEUSが他WASのEJBを呼び出す2つの方法があります。すなわち、JEUSはRMI/IIOPサーバーまたはクライアントになったり、その両方になったりできます。これらのすべての場合に対応できるORB(Object Request Broker)ランタイムが必要であり、トランザクションとセキュリティの相互運用も提供する必要があります。

JEUSには、CORBAネーミング・サービスを提供するためのCOSネーミング・サーバーおよびスタブ・クラス無しで動的スタブをメモリーで自動生成するORBランタイムが組み込まれています。トランザクションの相互運用のためにはOTS(Object Transaction Service)仕様を、セキュリティの相互運用のためにはCSlv2(Common Secure Interoperability version 2)仕様を実装しています。

CORBAおよびRMI/IIOPの詳しい内容については、以下の関連リンクを参照してください。

- [CORBA 2.3.1 Specification](#)
- [Glossary of Java IDL Terms](#)
- [Java RMI over IIOP](#)
- [OTS 1.2.1 Specification](#)
- [CSlv2 Specification](#)

### 5.1.1. トランザクション相互運用(OTS)

トランザクションの相互運用はOTS(Object Transaction Service)仕様をベースにしています。オブジェクトはトランザクションの影響を受けるCORBAオブジェクトを意味します。

OTSを使用する場合、ORBにトランザクションが処理できるインターセプター(リスナーの一種)を追加することになります。追加されたインターセプターがIIOPプロトコル・ヘッダーのトランザクション関連部分を処理し、クライアントまたはサーバーの役割をするJEUSは別のリモート・システム(別のWASまたはJEUS)間のトランザクション伝播(Transaction propagation)をサポートします。

---

#### 参考

現在サポートしているOTSの仕様バージョンは1.2です。

---

### 5.1.2. セキュリティー相互運用(CSIv2)

CSIv2は相互運用においてセキュリティーを保証するための仕様です。CSIを使用すると、OTSと同様セキュリティー処理が可能なインターセプターがORBに追加され、インターセプターがセキュリティーに関するヘッダーおよびその他の処理を行います。

セキュリティーの相互運用は、EJB参照のIOR(Interoperable Object Reference)にCSIのためのセキュリティー情報が追加されることで始まります。ネーミング・サーバーにIORが登録されると、これを使用するクライアントのORBとJEUS MSがIIOPプロトコルのセキュリティー・レベルでネゴシエーションを処理します。またJEUS MSは、別のEJB Beanを呼び出す場合、CSIサービスが必要であればクライアントとして動作します。

---

#### 参考

セキュリティーの相互運用についての詳細内容は、CORBAのCSIv2仕様を参照してください。

---

## 5.2. 相互運用の設定

相互運用はデフォルトで設定されていないため、使用する際には関連しているサーバーの設定が必要です。

JEUSのEJBをRMI/IIOPにエクスポートするには、MSのEnable Interop設定を有効にし、デプロイされるEJBにIIOPエクスポート関連の設定が必要となります。一方、JEUSをRMI/IIOPクライアントとしてのみ使用する場合は、該当するMSのEnable Interop設定のみ有効にします。

相互運用に関連する事項は、WebAdminを使用して設定できます。

## 5.2.1. COSネーミング・サービスの設定

MSで提供する基本的なJNDIネーミング・サービス以外に、CORBAオブジェクトのためのCOSネーミング・サービスが追加できます。COSネーミング・サービスには、RMI/IIOPにエクスポートされるEJBの参照(スタブ)が保存され、これをEJBクライアントからルックアップして使用します。

COSネーミング・サービスは必要な時点で自動的に開始されます。MS JVMで動作し、「**BASEPORT+4(例: 9740)**」をサービス・ポートとして使用します。

### 参考

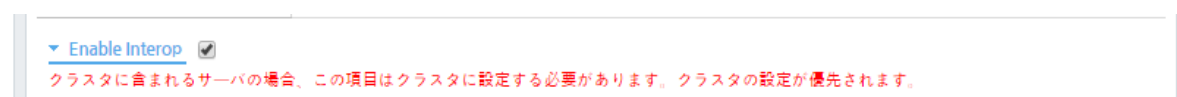
「**BASEPORT**」は、WebAdminで[**Servers**]メニューのサーバー・リストからサーバーを選択した後、[**Resource**] > [**Listener**]メニューの「**base**」項目に設定されたリスナーの「**Listen Port**」です。

## 5.2.2. 相互運用性の有効設定

RMI/IIOPクライアントおよびサーバーに必要な相互運用機能を使用するには、相互運用性の属性を有効にする必要があります。この属性を有効にすると、ORBが初期化されます。

相互運用性の属性を有効にするには、WebAdminの[**Servers**]メニューを選択した後、検索されたサーバー・リストから目的のサーバーをクリックします。[**Server**]画面の**詳細設定**で「**Enable Interop**」を設定します。

[図 5.1] 相互運用性の有効設定



CSlv2機能を使用するためにはより詳しい設定が必要です。以下の「[5.2.3. CSlv2セキュリティの相互運用設定](#)」を参照してください。

## 5.2.3. CSlv2セキュリティの相互運用設定

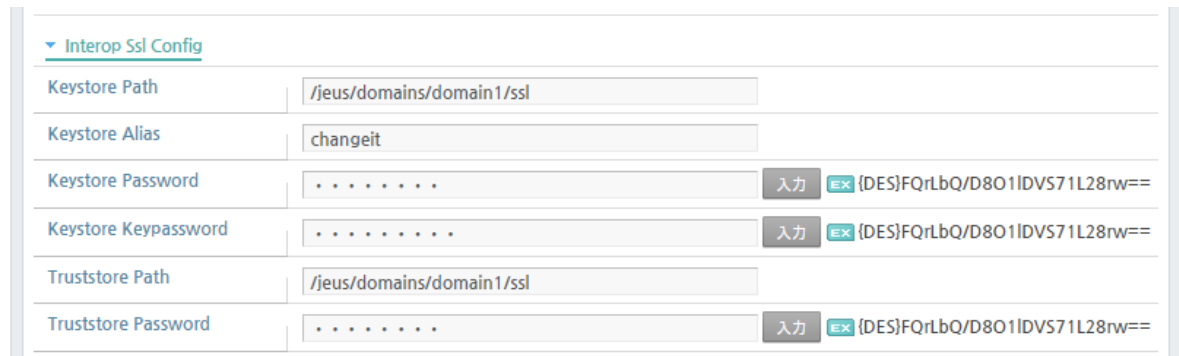
CSlv2機能を実行するために、2つの追加的なセキュリティ環境ファイルと、JEUSセキュリティ・マネージャーの情報を利用します。追加的に設定するセキュリティ環境ファイルは、**Keystore**と**Truststore**ファイルです。

セキュリティ環境ファイル	説明
Keystoreファイル	X.509のためのキーを保存します。Oracle社が提供するX.509キーストアを実装したファイルです。Secure Socket Layer(SSL)が呼び出されると、このファイルがクライアントに転送されます
Truststoreファイル	X.509クライアント側の証明設定ファイルです。キーストアと同じ形式です

このファイルのパスと必要情報は、WebAmdinまたはシステム・プロパティを使用して設定できます。

CSlv2関連の事項は、**Enable Interop**の**Interop Ssl Config**に設定します。

**【図 5.2】CSlv2セキュリティの相互運用設定 - Interop Ssl Config**



CSlv2関連の設定は、実行スクリプトのシステム・プロパティを使用することも可能です。ただし、システム・プロパティより、WebAdmin設定が優先されます。

以下は、実行スクリプトのオプションです。設定値は「-D」パラメータを使用します。

**【表 5.1】KeystoreとTruststoreファイルに関連するJVMの-Dパラメータ**

パラメータ	説明
jeus.ssl.keystore	Keystoreファイルまで絶対パスです (デフォルト値: DOMAIN_HOME/config/keystore)
jeus.ssl.keypass	Keystoreファイルに与えられたパスワード値です(デフォルト値: jeuskeypass)
jeus.ssl.truststore	Truststoreファイルまで絶対パスです (デフォルト値: DOMAIN_HOME/config/truststore)
jeus.ssl.trustpass	Truststoreファイルに与えられたパスワードです(デフォルト値: jeustrustpass)

信頼するノード同士においては、呼び出し者の認証(Authorization)および承認(Authorization)が不要な場合もあります。

jeus.ejb.csi.trusthostsシステム・プロパティにIPアドレスを設定すると、信頼するノード同士の不要なセキュリティ・チェックを省くことができます。JEUSセキュリティ・マネージャーは不特定のアクセスに対し、anonymous principalを表す「guest」を使用します。

## 5.2.4. EJB RMI/IIOPの設定

EJBをRMI/IIOPにエクスポートするには、jeus-ejb-dd.xmlにEJB別に<export-iiop>を設定します。<export-iiop>が設定されると、EJBホーム参照がCOSネーミング・サービスに与えられた<export-name>で

登録されます。このように登録すると、COSネーミング・サービスを利用して外部からEJB参照を取得して使用することができます。以下は、EJB RMI/IIOPを設定したjeus-ejb-dd.xmlの例です。

**[例 5.1] EJB RMI/IIOPの設定 : <<jeus-ejb-dd.xml>>**

```
...
<jeus-bean>
  <ejb-name>CalcEJB</ejb-bean>
  <export-name>ejb/Calc</export-name>
  <export-iiop>
    <only-iiop>true</only-iiop>
  </export-iiop>
</jeus-bean>
...
```

以下は、設定タグについての説明です。

タグ	説明
<only-iiop>	EJBをRMIとRMI/IIOPを使用してそれぞれエクスポートするか、RMI/IIOPのみ使用するかを設定します。前者の場合、COSネーミング・サーバーにはRMI/IIOPスタブが登録され、JEUSネーミング・サーバーにはRMIスタブが登録されます

**参考**

現在はEJBホームとEJBオブジェクト参照のみRMI/IIOPを使用してエクスポートできます。EJB 3.0ビジネス・ビューはRMI/IIOPが使用できません。

## 5.3. RMI/IIOPクライアント

RMI/IIOPクライアントは、別のJEUSや他ベンダーのWAS、またはスタンドアローン・クライアントになることができます。本節では、RMI/IIOPクライアントでEJBを使用する方法について説明します。

### 5.3.1. JEUS MS(Managed Server)

JEUS MSで運用されるアプリケーションにおいて、JEUSや他ベンダーのWAS上に存在RMI/IIOP EJBを呼び出す方法は以下のとおりです。

まず、COSネーミング・サーバーからEJBホーム参照(スタブ)をルックアップします。直接ルックアップする場合は、JNDI APIのcorbaname URLが使用できます。

**[例 5.2] corbanameルックアップの使用**

```
InitialContext ctx = new InitialContext();
Object obj = ctx.lookup("corbaname:iiop:1.2@192.168.11.22:9740#ejb/Calc");
```

```
CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();
```

URLは「corbaname:iiop:1.2@<host>:<port>#<name>」形式であり、<host>と<port>はCOSネーミング・サーバーのアドレスです。また、直接COSネーミング・サーバーをPROVIDER URLに指定し、初期コンテキスト(InitialContext)を取得して使用することも可能です。

以下の例のように、PROVIDER URLを指定する方式は様々です。1つ方式を選択して指定します。

#### [例 5.3] PROVIDER URLの使用

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "iiop://192.168.11.22:9740");
// env.put(Context.PROVIDER_URL, "iiopname://192.168.11.22:9740");
// env.put(Context.PROVIDER_URL, "corbaname:iiop:1.2@192.168.11.22:9740");
// env.put(Context.PROVIDER_URL, "corbaloc:iiop:1.2@192.168.11.22:9740");

InitialContext ctx = new InitialContext(env);
Object obj = ctx.lookup("ejb/Calc");

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();
```

依存性注入(Dependency Injection)、または「java:comp/env/」形式の論理的なJNDI名を使用するクライアントの場合、jeus-web-dd.xmlのようなアプリケーションのJEUS DDファイルで、<export-name>をcorbaname URLにマッピングします。

#### [例 5.4] Servlet EJB Injection

```
@EJB(name="ejb/Calc")
private CalcHome calcHome;
```

#### [例 5.5] RMI/IIOP EJBのマッピング : <<jeus-web-dd.xml>>

```
<ejb-ref>
  <jndi-info>
    <ref-name>ejb/Calc</ref-name>
    <export-name>corbaname:iiop:1.2@192.168.11.22:9740#ejb/Calc</export-name>
  </jndi-info>
</ejb-ref>
```

## 5.3.2. 他ベンダーのWAS

他ベンダーのWASで運用されるアプリケーションからJEUSのRMI/IIOP EJBを呼び出す方法は、[「5.3.1. JEUS MS\(Managed Server\)」](#)の方式とほぼ同じです。



この場合、JEUSのように別途のORB設定が必要であるか否かを確認し、必要な場合は設定を行います。詳細内容については、該当する製品の関連文書を参照してください。

### 5.3.3. スタンドアローン・クライアント

スタンドアローン・クライアントは、別途のORBランタイムおよび関連設定がされていないため、まず基本的に必要なjclient.jarクライアント・ライブラリー以外にも、JEUSシステム・ライブラリーに存在するcorba-omgapi.jarとcorba-orb.jarをクラス・パスに追加し、jeus.client.interop=trueシステム・プロパティを追加します。

EJBホーム参照(スタブ)を取得する方式は、「[5.3.1. JEUS MS\(Managed Server\)](#)」と類似しています。ただし、JEUSで提供するJNDIプロバイダーを追加的に指定します。

#### [例 5.6] スタンドアローン・クライアントの使用

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JEUSContextFactory");
env.put(Context.PROVIDER_URL, "iiop://192.168.11.22:9740");

InitialContext ctx = new InitialContext(env);
Object obj = ctx.lookup("ejb/Calc");

CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);
Calc calcRef = home.create();
```

## 5.4. 既知の問題点(Known Issues)

既知の問題点は以下のとおりです。

- **サーバーに接続できない場合、再接続を1分間無限にトライする現象**

JEUSに組み込まれたORBからサーバーに接続できない場合、別途のスリープ・タイム(sleep time)無しで60秒間再接続します。そのため、複数のログ・メッセージが出力されるか、CPUの使用率が高まる問題が生じます。これを防ぐためには、クライアントで以下のようなオプションをシステム・プロパティとして設定する必要があります。(デフォルト値: 60000、単位: ms)

```
com.sun.corba.ee.transport.ORBCommunicationsRetryTimeout=1
```

- **PortableRemoteObject.narrow()の呼び出し後、NullPointerExceptionが発生する現象**

クライアントで以下のように呼び出す場合、homeオブジェクトがnullを返し、NullPointerExceptionが発生する場合があります。

#### [例 5.7] NullPointerExceptionが発生する場合

```
CalcHome home = (CalcHome)PortableRemoteObject.narrow(obj, CalcHome.class);

// null is returned
Calc calcRef = home.create(); // NullPointerException
```

この問題は、CORBAスタブ・クラスが見つからなかった場合に発生します。JEUS MSで**Enable Interop**設定に誤りがある場合です。同設定を行うと正常なORBが初期化され、スタブを動的に自動生成します。外部クライアントの場合、動的スタブを生成する機能がないとこのようなエラーが発生する場合があります。

スタンドアローン・クライアントの場合、システム・プロパティに以下のように設定されているのを確認します。

```
jeus.client.interop=true
```

- **WebLogicでドット(.)が含まれた名前のJEUS EJBが見つからない現象**

WebLogicでは「com.acme.CalcHome」のようにドット(.)が含まれた形式で<export-name>を指定した場合、ルックアップに失敗します。WebLogicはドット(.)をスラッシュ(/)と同様に認識し、要求が「com/acme/CalcHome」のように送信されるため、ドット(.)が含まれた名前は使用しないようにしてください。

# 第6章 EJBのクラスタリング

本章では、EJBクラスタリングの概念と主要機能の設定方法について説明します。

## 6.1. 概要

EJBのフェイルオーバーとロードバランシング機能を使用するためには、各Beanが複数のEJBエンジンにデプロイされ、クラスタリングを構成する必要があります。クラスタリングは、コンポーネント・レベル(個別Bean)で実行され、ステートレス/ステートフル・セッションBeanとエンティティBeanで使用できます。メッセージ駆動型Bean(MDB)はクラスタリングの対象ではありません。

JEUS EJBクラスタリングには、以下の2つの機能があります。

- ロードバランシング

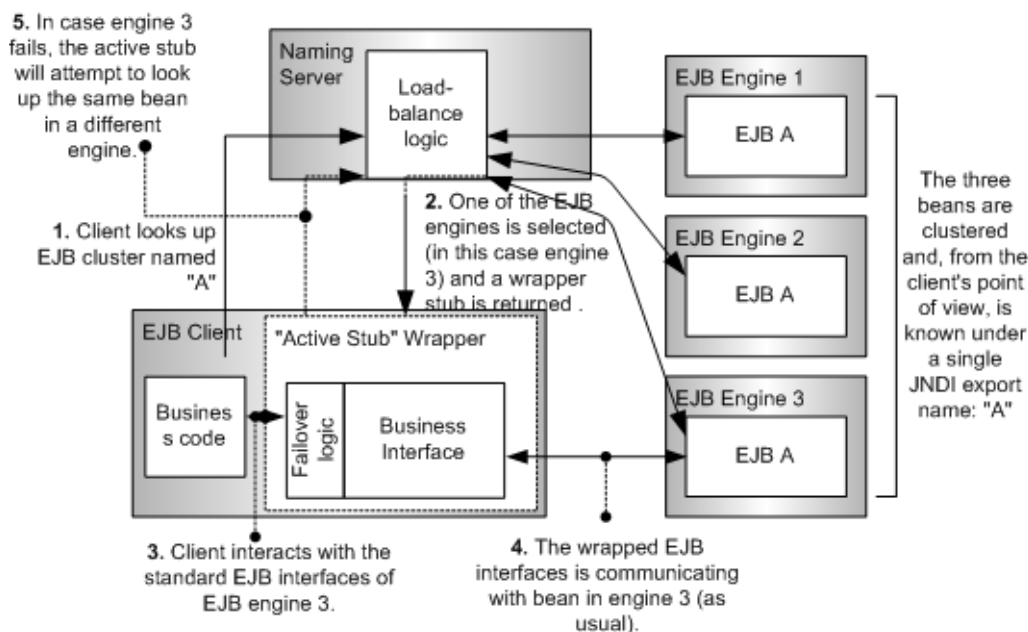
EJBに対する要求を分散させ、Beanの全体的な応答速度を高めます。

- フェイルオーバー

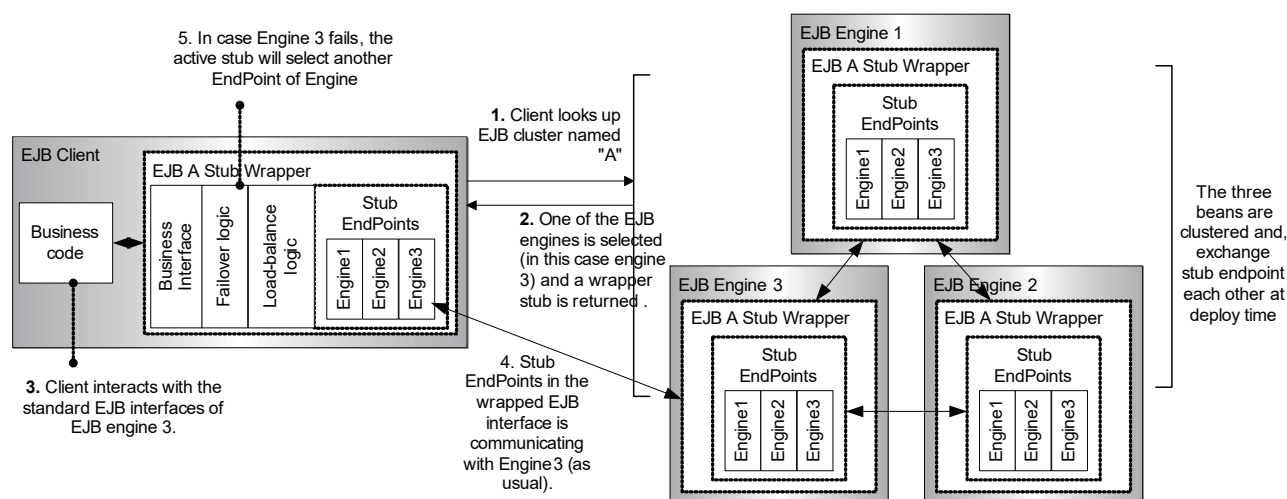
Beanメソッドの実行前と実行中に、1つのEJBインスタンスまたはEJBエンジンが停止すると、別のEJBエンジンのEJBインスタンスで要求が処理されます。

以下の図は、EJBクラスタリングの主要機能であるフェイルオーバーとロードバランシングの動作方式です。

【図 6.1】 EJB2クラスタリングのアーキテクチャー



【図 6.2】 EJBクラスタリングのアーキテクチャー



クラスタリングするモジュールをデプロイすると、ネーミング・サーバーにすべて同じ名前でバインドされます。クライアントがその中から1つの名前を使って実行するとロードバランシングとフェイルオーバーが可能になります。したがって、同じモジュールであってもネーミング・サーバーに異なる名前でバインドしてデプロイすると当該モジュールはクラスタリングされません。

## 参考

ステートフル・セッションBeanは、フェイルオーバーのためにJEUSの分散セッション・マネージャーを使用します。1つのEJBエンジンには1つのセッション・マネージャーが存在するため、すべてのBeanのクラスタリング領域は一致する必要があります。たとえば、EJB engine1とEJB engine2をBean AIにし、EJB

engine1とEJB engine3をBean Bにすると、セッション・マネージャーはBean AとBean BがEJB engine1、EJB engine2、EJB engine3にクラスタリングされていると誤って解釈をします。

---

## 6.2. 主要機能

以下は、EJBクラスタリングの主要機能についての説明です。

### 6.2.1. ロードバランシング

クライアントがルックアップまたは注入を使用してBean Aを要求すると、ネーミング・サーバーは3つのEJBエンジンに存在する3つのBeanからランダムで1つを選択して返します。この方式を使用すると、3つのBeanは同じメソッドから同様な呼び出し要求を受けることになるため、1つのエンジンがすべての要求に対応するよりは、およそ3倍のシステム向上が期待できます(ロードバランシングの場合に発生する小規模のリソース消費は考慮しない場合)。クライアントはBeanの種類によって以下のように動作します。

- EJB 2 - ルックアップ時に選択されたEJBエンジンを継続して呼び出します。JEUS 7以前のバージョンで使用していた`jeus.jndi.clusterlink.selection-policy`プロパティは削除されました。そのため、ルックアップ時には基本的に同じEJBエンジンのBeanを優先的に呼び出すか、あるいは、ランダムで選択されたEJBエンジンからルックアップします。
- EJB 3 - ステートレス：メソッド呼び出しごとにロードバランシングを実行します。(ただし、最初の呼び出しはルックアップ時に選択されたEJBエンジンに送られます)。JEUS 7以前のバージョンでは、ルックアップを通じてのみロードバランシングを行ったため、呼び出しは一度ルックアップしたEJBエンジンへのみ可能でした。一方、JEUS 8からはルックアップしたBeanの中にクラスタリングに属している他のEJBエンジンと通信できるエンドポイントが存在しており([図 6.2](#)を参照)、このエンドポイントを使ってメソッド呼び出しレベルのロードバランシングが可能となります。

以下は、設定可能なロードバランシング・ポリシーです。

- RoundRobin: クラスタに属しているサーバーを順番に一度ずつ選択
- Random: サーバーをランダムに選択
- LocalLinkPreference: 同じサーバーを優先的に選択

ロードバランシング・ポリシーは、システム・プロパティ(`jeus.ejb.cluster.selection-policy`)で設定できます。詳しい内容は、『*JEUS リファレンスガイド*』の「1.8. EJBのシステム・プロパティ」を参照してください。

- EJB 3 - ステートフル：ルックアップ時に選択されたEJBエンジンを継続して呼び出します。ステートフルBeanは状態を保存するためにセッションを使用するので、メソッド呼び出しごとにロードバランシングを実行すると、セッション・コピーによるオーバーヘッドが発生し、性能が低下される可能性があります。(セッションの共有方法または設定についての詳細は、『*JEUS セッション管理ガイド*』の「第2章 分散セッション・サーバー」を参照してください)

## 6.2.2. フェイルオーバー(EJBのフェイルオーバー)

フェイルオーバーとは、1つのEJBサービスに障害が発生してもサービスを正常に提供する機能です。(例: OS障害、ネットワークの停止、EJBエンジンの障害など)

JEUSシステムが処理できる障害復旧には、以下の2つの方法があります。

- **クライアントから要求が受信されたとき、使用可能なエンジンのBeanに未処理要求を転送する方法**

問題が発生したEJBエンジンの除いて、ロードバランシング・アルゴリズムを実行します。新しいクライアントの要求を処理するには、使用可能なエンジンのBeanを選択します。

JEUSでは、フェイルオーバーの再ルーティングがクライアント側のEJBスタブで処理されます。このスタブを、動的スタブ、あるいは標準EJBインターフェースを囲んだwrapperといいます。wrapperを使用するときの特徴は、現在接続されているBeanまたはEJBエンジンに問題があるか否かを判断するロジックの有無です。このような問題が検知されたら、起動中のスタブが自動的にJNDIサーバーに接続し、動作しているEJBエンジンの代わりに新しいスタブを要求します。([図 6.1]または[図 6.2]の5番)

- **実行中のBeanに何らかの理由でランタイム・エラーが生じた場合、処理中の要求を別のBeanに再転送する方法**

このような状況でのフェイルオーバー方法には限界があります。Beanが要求を処理している最中に問題が検知されると、処理の進行状況や、どのランタイム・エラーを発生させたのかが把握できません。単純にクラスターに存在する別のBeanのメソッドを呼び出すことは、また別の問題を引き起こす可能性があります。

この問題を解決する方法として、DBフィールドを1つずつ増加させるメソッドを有するBean Instance Aを想定してみます。1つが増加された直後に問題が発生する場合、単純に別のBean「B」に存在する同じビジネス・メソッドを呼び出すと、また1つ増加されます。したがって、DBに一貫性のない誤った値を転送することになります。このような場合は、多重呼び出し不変(Idempotent)メソッドを使用すると安全に復元することができます。多重呼び出し不変メソッドを利用したEJBのフェイルオーバーについての詳細内容は、「6.2.3. 多重呼び出し不変メソッドを使用したEJBのフェイルオーバー」を参照してください。

上記の2つのシナリオの相違点は、エラーが検知されるタイミングです。すなわち、リモート・ビジネス・メソッドを呼び出す前なのか、あるいはBeanが要求を処理している最中なのかです。

## 6.2.3. 多重呼び出し不変メソッドを使用したEJBのフェイルオーバー

多重呼び出し不変メソッドはネガティブな副作用のない「getter」メソッドです。メソッドの実行中にいかなる状態(例: インスタンス変数、DBフィールドなど)も変更されないことを保証します。

したがって、「6.2.2. フェイルオーバー(EJBのフェイルオーバー)」の2番目の方法が持つ限界は多重呼び出し不変メソッドで解決できます。しかし、多重呼び出し不変メソッドを使用しない場合は、ランタイム・エラーが発生したメソッドを再実行するよりは、例外を発生させることをお勧めします。EJBのフェイルオーバーをより

うまく動作させるには、多重呼び出し不変メソッドの使用が望ましいのですが、メソッドが多重呼び出し不変メソッドであるか否かを判断する基準が特にないため、ビジネス・メソッドの状態を的確に識別してから設定する必要があります。

## 6.2.4. セッション複製(Session Replication)

ステートフル・セッションBeanの場合、セッションのバックアップのためにJEUSセッション・マネージャーを使用します。通常、ビジネス・メソッドの呼び出し単位でセッションの状態が変化するため、JEUSではメソッド呼び出しが発生し、結果が返される時点ごとにJEUSセッション・マネージャーにセッションのバックアップを要求します。このようなバックアップ作業を**セッション複製(Session Replication)**といいます。

JEUSセッション・マネージャーはセッションを同期的(Sync)/非同期的(Async)に複製できます。JEUSではこれを複製モード(Replication Mode)と称します。

- 同期的(Sync)複製モード

セッション・バックアップが完了されるまでBeanが待機するため、フェイルオーバー時に常に最新のセッションが複製されているというメリットがあります。しかし、セッション・マネージャーでネットワーク障害などによってタイムアウトが発生し、処理が中断される場合、Beanもその分待機するデメリットがあります。

- 非同期的(Async)複製モード

パフォーマンスの面での問題はありますが、セッション複製が直ちに行われなため、その間にフェイルオーバーが実行されると最新のセッションが取得できなくなります。

上記の2つの方法にはそれぞれ長短所があるため、JEUSではBeanと各ビジネス・メソッドの特性に合わせてユーザーが設定できるようになっています。また、セッション複製が不要なメソッドの設定も可能です。詳しい設定方法は、「[6.3. EJBクラスタリングの設定](#)」を参照してください。

---

### 参考

JEUSでは、クラスタリングに参加したステートフル・セッションBeanは基本的に同期的(Sync)複製モードでセッション複製が行われますが、ユーザー設定で変更することも可能です。

---

## 6.3. EJBクラスタリングの設定

EJBクラスタリングはBeanクラスにアノテーションで設定するか、jeus-ejb-dd.xmlに設定することができます。設定する事項には、クラスタリングで構成されるBeanと、そのBeanの多重呼び出し不変メソッド、Beanまたは各メソッドのセッション複製モードなどがあります。

本節では、例を利用してアノテーションとDD(xml)ファイルにクラスタリングを設定する方法について説明します。

### 6.3.1. アノテーションを使用したクラスタリングの設定

クラスタリングに参加するBeanクラスまたはメソッドに、以下のようなアノテーションを使用して設定します。

**[例 6.1] アノテーションを使用したクラスタリングの設定 : <<CounterEJB.java>>**

```
package ejb.basic.statelessSession;

import javax.ejb.Stateful;
import jeus.ejb.Clustered;
import jeus.ejb.Replication;
import jeus.ejb.ReplicationMode;

@Stateful(name="counter", mappedName="COUNTER")
@Clustered(version = ClusteringVersion.JEUS8)
@Replication(ReplicationMode.SYNC)
@CreateIdempotent
public class CounterEJB implements Counter, CounterLocal {
    private int count = 0;

    public int increaseAndGet() {
        return ++count;
    }

    @Replication(ReplicationMode.NONE)
    public void doNothing(int a, String b) {
    }

    @Idempotent
    public int getResult() {
        return count;
    }

    @Idempotent
    @jeus.ejb.Replication(ReplicationMode.ASYNC)
    public int getResultAnother() {
        return count;
    }
    ...
}
```



以下は、クラスター別の設定についての説明です。

クラス	説明
@jeus.ejb.Clustered	<p>Beanのクラスタリング全体を有効または無効にします。2つのバージョンがあり、デフォルトはClusteringVersion.JEUS8です</p> <ul style="list-style-type: none"><li>– ClusteringVersion.JEUS7 : JEUS 8以前のバージョンとの互換性のために使用します。この値を使用すると、JEUS 7でJEUS 8にあるEJBを呼び出すことができます</li><li>– ClusteringVersion.JEUS8 : JEUS 8から新しく導入された方式のクラスタリングを使用します。JEUS 7に比べ、ルックアップ、ビジネス・メソッド呼び出しの速度が向上されました</li></ul>
@jeus.ejb.Idempotent	<ul style="list-style-type: none"><li>– Beanまたはビジネス・メソッドが多重呼び出し不変であるか否かを宣言します</li><li>– valueをfalseに設定すると、非多重呼び出し不変になります</li><li>– リモートBeanオブジェクトを生成する最中に例外状況が発生する場合、ステートレス・セッションBeanに対しては常に多重呼び出し不変で再トライし、ステートフル・セッションBeanの場合は、非多重呼び出し不変でアプリケーションに例外を発生させます</li></ul>
@jeus.ejb.Createldempotent	<ul style="list-style-type: none"><li>– セッションBeanを生成するとき、多重呼び出し不変として定義するか否かを宣言します</li><li>– valueをfalseに設定すると、非多重呼び出し不変になります。このアノテーションを記述しない場合は、ステートレス・セッションBeanを常に多重呼び出し不変として見なします</li></ul>
@jeus.ejb.Replication	<ul style="list-style-type: none"><li>– Bean別、またはビジネス・メソッド別にセッション複製モードが設定できます</li><li>– こちらでは、<b>jeus.ejb.ReplicationMode</b>という列挙型(enum)のクラスを使用します。このクラスについては、JEUS API(javadoc)ドキュメントを参照してください。同アノテーションは、@Replication(ReplicationMode.SYNC)のような形式で使用できます。ユーザーはBean別に設定できますが、何にも設定しない場合のデフォルトは、<b>ReplicationMode.SYNC</b>です。メソッドを設定しない場合は、基本的にBeanに設定された値に従います。ただし、@Idempotentの場合のデフォルトはReplicationMode.NONEであり、ビジネス・ホームに定義されるcreate()はBeanの設定に従います</li></ul>

## 6.3.2. xmlを使用したクラスタリングの設定

JEUS EJBモジュールDDファイル(jeus-ejb-dd.xml)には、クラスタリングに参加するそれぞれのBeanのために<clustering>タグの下位に以下のような設定が適用できます。

**[例 6.2] xmlを使用したクラスタリングの設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            <ejb-name>counter</ejb-name>
            <export-name>COUNTER</export-name>
            . . .
            <clustering>
                <enable-clustering>true</enable-clustering>
                <clustering-version>jeus8</clustering-version>
                <ejb-remote-idempotent-method>
                    <method-name>getResult</method-name>
                </ejb-remote-idempotent-method>
                <ejb-remote-idempotent-method>
                    <method-name>getResultAnother</method-name>
                </ejb-remote-idempotent-method>
                <create-idempotent>true</create-idempotent>
                <replication>
                    <bean-mode>sync</bean-mode>
                    <methods>
                        <method>
                            <method-name>doNothing</method-name>
                            <method-params>
                                <method-param>int</method-param>
                                <method-param>java.lang.String</method-param>
                            </method-params>
                            <mode>none</mode>
                        </method>
                        <method>
                            <method-name>getResultAnother</method-name>
                            <method-params>
                                <method-param>void</method-param>
                            </method-params>
                            <mode>async</mode>
                        </method>
                    </methods>
                </replication>
            </clustering>
        </jeus-bean>
    </beanlist>
</jeus-ejb-dd>
```

```

        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>

```

以下は、**<clustering>**の下位設定タグについての説明です。

タグ	説明
<enable-clustering>	Beanのクラスタリング全体を有効または無効にします
<clustering-version>	クラスタリングのバージョンを指定します。指定可能な値は、jeus7とjeus8です。デフォルト値はjeus8です。その他の詳しい内容は、@jeus.ejb.Clusteredと同様です
<ejb-remote-idempotent-method>	Beanメソッドの中で、多重呼び出し不変メソッドを宣言します( <a href="#">「6.3.1. アノテーションを使用したクラスタリングの設定」</a> の@jeus.ejb.Idempotentを参照)
<ejb-remote-idempotent-exclude-method>	Beanメソッドの中で、多重呼び出し不変メソッドとして宣言したものから除外したいメソッドを宣言します。<ejb-remote-idempotent-method>より優先順位は高く、使用方法は同様です
<ejb-home-idempotent-method>	2.xスタイルのホーム・インターフェースに定義されたメソッドの中から、多重呼び出し不変メソッドを宣言します( <a href="#">「6.3.1. アノテーションを使用したクラスタリングの設定」</a> の@jeus.ejb.CreateIdempotentを参照)。使用方法は<ejb-remote-idempotent-method>と同様です
<ejb-home-idempotent-exclude-method>	2.xスタイルのホーム・インターフェースに定義されたメソッドの多重呼び出し不変メソッドとして宣言したものから除外したいメソッドを宣言します。<ejb-home-idempotent-method>より優先されます。使用法は<ejb-remote-idempotent-method>と同様です
<create-idempotent>	セッションBeanを生成するとき、多重呼び出し不変として定義するか否かを宣言します
<replication>	Beanレベルのセッション複製モード、またはメソッド別の複製モードを設定します。詳細内容については、 <a href="#">「6.2.4. セッション複製(Session Replication)」</a> の <a href="#">[例 6.2]</a> を参照してください

以下は、上記で指定したBeanクラスタリングが動作するための注意点です。

- クラスタリングに参加するすべてのBeanは、<clustering>下位のすべての情報が同様である必要があります。
- クラスタリングを構成するためには、目的のBeanの<export-name>をすべて同様に設定します。

### 6.3.3. ステートフル・セッションBeanのクラスタリング設定

クラスタリング環境においてステートフル・セッションBeanを実行するには、セッション・マネージャーを追加設定する必要があります。

以下は、セッション・マネージャーの主要設定であるパッシベーション・タイムアウトについての説明です。その他の項目については、『JEUS セッション管理ガイド』の「2.10.3. 分散型セッション・サーバーの設定」を参照してください。

項目	説明
Passivation Timeout	Beanの情報をパッシベーションするための設定です。  EJBエンジンのレプリケーションに設定された秒ごとに、パッシベーションまたは切断(disconnect)されたBeanの有無をチェックします。そのとき、最後にBeanにアクセスした時間が設定時間(ms)を超過したBeanはパッシベーションの対象となります。  セッション設定のうち、EJBパッシベーション・タイムアウトに関連する設定は、Trigger-Timeoutです

## 6.4. EJBフェイルオーバーの制限

クライアントでEJB参照(スタブ)を毎回ルックアップせず、ルックアップしたEJB参照をキャッシュして繰り返し使用する場合、(インジェクションを使用する場合も含む)、有効なEJBエンド・ポイントが存在しているにもかかわらずフェイルオーバーされない場合があります。これは、ルックアップする時点で存在していたEJBエンド・ポイントのリストに該当するMSがすべて起動されておらず、その後に起動された新しいMSのみ存在する場合に発生します。

フェイルオーバーが必要な時点から内部的にルックアップして新しいEJB参照をクライアントに渡します。しかし、使用中のEJB参照がルックアップされるときデプロイされていたMSがすべてダウンすると、現在使用中のEJB参照がルックアップされる時点ではデプロイされませんが、その後デプロイされ、フェイルオーバーする時点でサービス可能な新しいEJBエンド・ポイントが存在してもフェイルオーバーされません。

MSのダウンとは、異常終了やEJBエンド・ポイントのアンデプロイなどによって、既存のすべてのEJBエンド・ポイントがサービスできない状態を意味します。また、新しいEJBエンド・ポイントがデプロイされる場合、遅れて(EJB参照をルックアップしてすでに使用している場合)新しいMSがクラスターに追加されるか、ダウンしたMSを再起動するか、アンデプロイされたEJBエンド・ポイントを再デプロイする場合などが含まれます。

2つのMSでアクティブ/バックアップ・クラスタリングを構成する場合に発生し得る現象です。さらに、以下ののようなシナリオで問題が発生する場合があります。

A、B、CというMSが存在しており、クライアントが初めてルックアップし、継続してキャッシュする場合です。

1. A、B、Cにそれぞれデプロイされており、初めてルックアップした結果としてAのEJBを取得します。

2. AのEJBを使用中にAが異常終了され、内部的にBのEJBをルックアップして引き続きサービスを受けます。
3. BのEJBがアンデプロイされ、CのEJBをルックアップして使用します。
4. このとき、BのEJBがデプロイされ、直後にCが異常終了しました。

この場合、CのEJB参照がルックアップされるとき、BのEJBエンドポイントはアンデプロイされており、CのEJB参照をルックアップした後にデプロイされたため、CがダウンされたときのBは運用中であったもののフェイルオーバーはされません。しかし、このときBがデプロイまたはアンデプロイされたのではなく、異常起動または異常終了された場合は、正常にフェイルオーバーされます。異常終了の場合はフェイルオーバーの時点で再起動するか否かをチェックするため、フェイルオーバーが正常に行われるのです。

フェイルオーバーが正常に行われない場合は、再びルックアップを実行し、新しいEJB参照を取得します。そうすると、新しいエンドポイントの一覧を取得できるため、このような問題を避けることができます。



# 第7章 セッションBean

本章では、ステートレス・セッションBeanとステートフル・セッションBeanについて説明します。

ステートレス・セッションBeanは、「第4章 EJBの共通特性」で説明した内容以外に特別な内容はないため、主にステートフル・セッションBeanについて説明します。

## 7.1. ステートレス・セッションBean

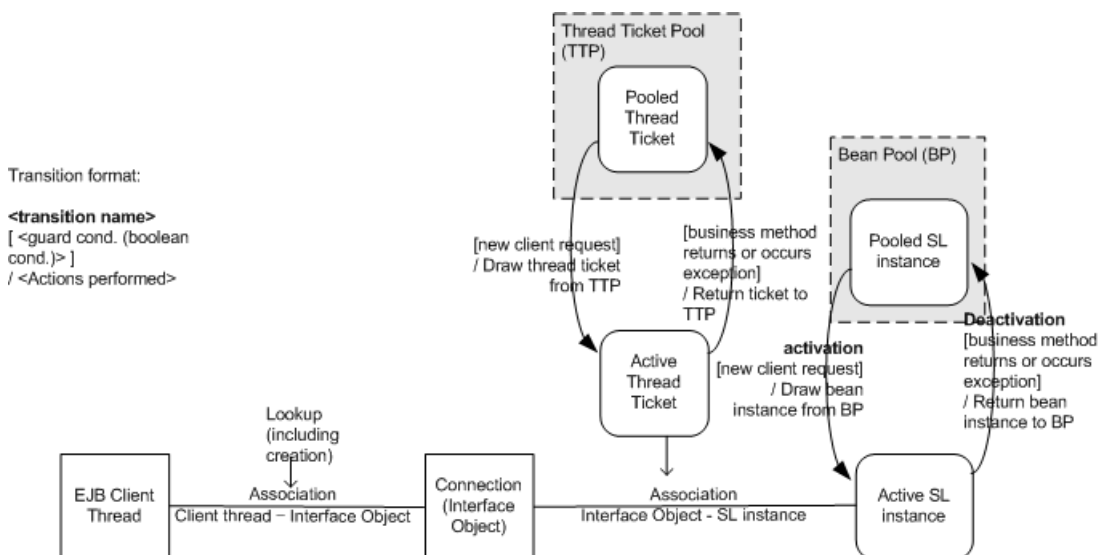
本節では、セッションBeanのTTP(Thread Ticket Pool)について説明します。

### 7.1.1. Thread Ticket Pool(TTP)とオブジェクト・マネージメント

ステートレス・セッションBeanは、クライアントの要求に関する状態情報を持っておらず、すべてのクライアントがコネクションを共有するので、コネクション・プールを設定する必要はありません。しかし、状態情報を持っていないことによりBeanインスタンスが再使用できるため、Beanプールを設定することができます。

以下は、ステートレス・セッションBeanのTTPとBeanプールの関係図です。

【図 7.1】ステートレス・セッションBeanのTTPとBeanプール



1. 作成(create)時点でクライアントと接続したコネクションに要求を送信すると、TTPでTT(Thread Ticket)を発行します。TTが発行されなかった場合は、発行されるまで待機します。待機時間が10分を超えるとリモート例外が発生し、要求が実行されません。(「第4章 EJBの共通特性」を参照)

2. TTが発行されたクライアントの要求を実行するため、BeanプールからBeanインスタンスを取得してコネクションと接続します。
3. ステートレス・セッションBeanインスタンスの処理が終了したら、TTはTTPに、BeanインスタンスはBeanプールに返され、次のクライアントの要求を待ちます。すなわち、要求ごとにTTとBeanインスタンスを割り当てし、要求処理が終了したら返されます。

Beanプール数とTTP数の意味の違いは、同時実行できるローカル・クライアントとリモート・クライアントの要求数に関連していることです。すなわち、リモート呼び出しはTTPからTTを取得した後、Beanプールからインスタンスを取得します。一方、ローカル呼び出しはクライアント・スレッドで実行されるため、TTの発行は不要で、Beanプールからインスタンスのみ取得します。

したがって、jeus-ejb-dd.xmlの<thread-max>値を小さく設定すると、設定値以上のリモート呼び出しは受けられませんが、Beanプールの最大値は無制限であるため、ローカル呼び出しは処理できます。

### 7.1.2. Webサービスのエンドポイント

EJB 2.1以降から、ステートレス・セッションBeanはWebサービス形式でエクスポートできます。

---

#### 参考

詳細内容については、『JEUS Webサービスガイド』を参照してください。

---

## 7.2. ステートフル・セッションBean

本節では、以下のステートフル・セッションBeanの追加設定の概念について説明します。

- オブジェクト・マネージメントの設定：Beanインスタンス・プールとコネクション・プール
- Beanインスタンス・プーリングのオプション
- セッション・マネージャーを利用した状態保持メカニズム

### 7.2.1. Thread Ticket Pool(TTP)とオブジェクト・マネージメント

ステートフル・セッションBeanは、クライアントの要求に関連した状態情報を保持する必要があるため、ステートレス・セッションBeanとは違ってコネクション・プールを設定し、活用する必要があります。これは、エンティティBeanにも同様に適用されます。詳細内容については、「[第8章 エンティティBean](#)」を参照してください。

以下は、ステートフル・セッションBeanのコネクション・プールとTTP、Beanプールの関係図です。



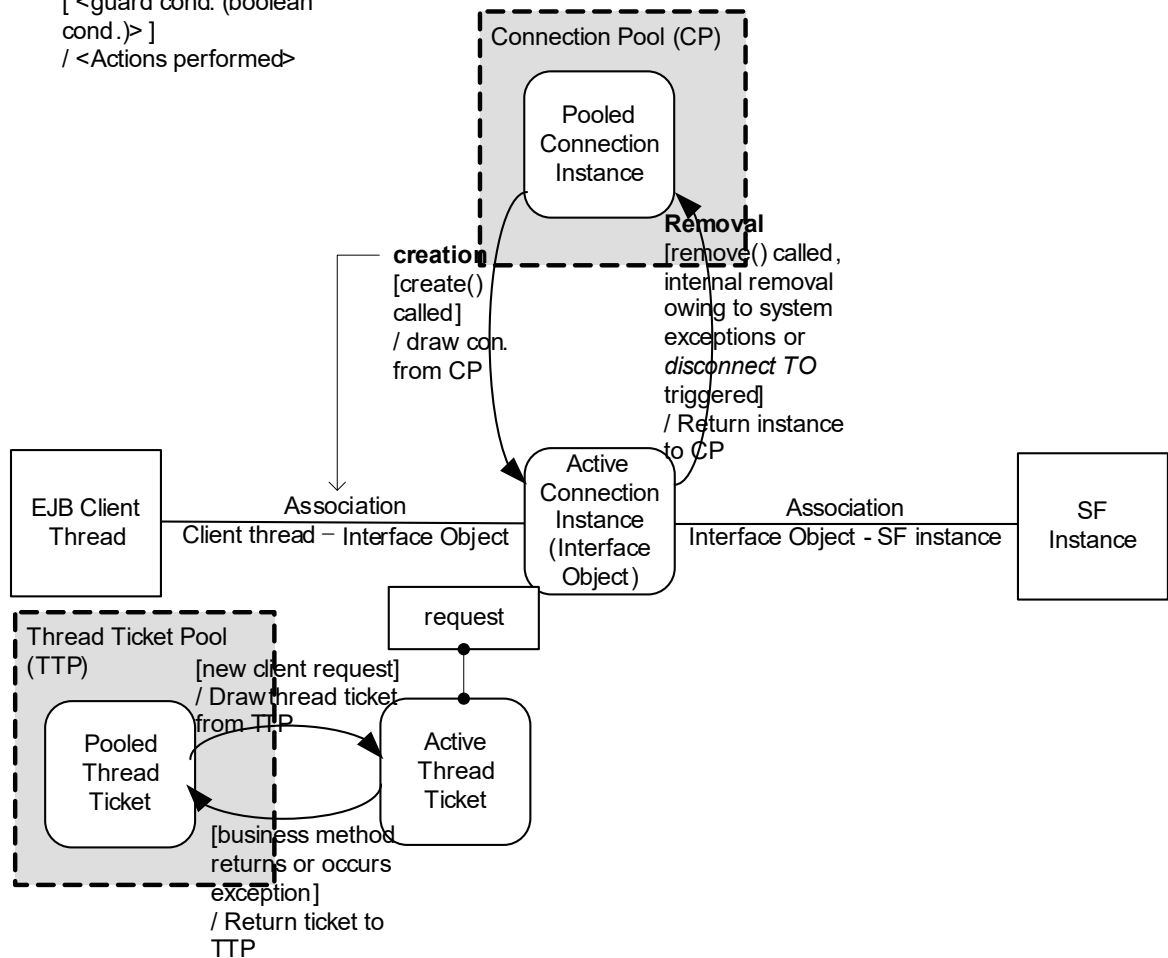
【図 7.2】ステートフル・セッションBeanのコネクション・プールとTTP、Beanプール

Transition format

<transition name>

[ <guard cond. (boolean  
cond.)> ]

/ <Actions performed>



クライアントがステートフル・セッションBeanを生成すると、新しいBeanインスタンス(SFインスタンス)が作成され、コネクション・プールからコネクションを取り出し、2つを接続します。Beanインスタンスと接続されたコネクションをクライアントに渡すと、このコネクションは現在のクライアントに割り当てられ、別のクライアントとは共有されません。したがって、クライアントが削除しない限りコネクション・プールには返されません。コネクション・プールに返されるときには接続されていたBeanインスタンスとの接続を切断します。

当該コネクションに要求するとTTPからTTが発行され、TTを受けた要求のみ処理されます。ステートレス・セッションBeanとは違って、要求ごとに新しいBeanインスタンスがBeanプールから割り当てられるのではなく、該当するクライアントのためのBeanインスタンスが固定されています。

上述のように、このBeanインスタンスはクライアントがBeanを削除すると、コネクションがコネクション・プールに返され、コネクションとの接続が切断されると同時に削除されます。基本的に、ステートフル・セッションBeanには状態があるため、Beanインスタンスを再使用するBeanプールは使用しません。

## 7.2.2. プーリング・セッションBean

EJB標準によると、ステートフル・セッションBeanのインスタンスは別のクライアント・セッションで再使用できません。しかし、Beanが削除されるとき正常に初期化されれば、Beanインスタンスを再使用することも可能です。JEUSでは、性能改善やリソースの無駄遣いを減らすため、Beanインスタンスを毎回作成せずに再使用する方法を提供します。

Beanプールを使用すると、Beanが削除されるとき当該BeanインスタンスがBeanプールに返されます。Beanの削除はクライアントが明示的にremoveを呼び出すか、長時間要求がないためタイムアウトされた場合に発生します。その際、PRE DESTROYコールバックが呼び出されますが、同状況ではBeanの初期化が正常に実装された場合のみ使用してください。

## 7.2.3. Beanプールの設定

ステートフル・セッションBeanをプーリングBeanに切り替えるには、jeus-ejb-dd.xmlの<pooling-bean>値をtrueに設定します。

**[例 7.1] Beanプールの設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
  . . .
  <beanlist>
    <jeus-bean>
      <ejb-name>teller</ejb-name>
      <export-name>TELLEREJB</export-name>
      <local-export-name>LOCALTELLEREJB</local-export-name>
      <export-port>7654</export-port>
      <export-iiop>true</export-iiop>
      <object-management>
        <pooling-bean>true</pooling-bean>
        <bean-pool>
          <pool-min>10</pool-min>
          <pool-max>200</pool-max>
          <resizing-period>1800000</resizing-period>
        </bean-pool>
        . . .
      </object-management>
      . . .
    </jeus-bean>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>
```

## 7.2.4. セッション・データ保持メカニズムの設定

ステートフル・セッションBeanは状態を持ちます。この状態はクライアント・セッションの間に維持される必要があります。すなわち、同じクライアントからのいかなる要求にも状態が保持される必要があります。これがステートフル・セッションBeanの基本的な特性です。

一般的にランタイム時の状態はインスタンス変数で保存されます。しかし、ステートフル・セッションBeanがパッシベーションされると、システムのリソースを保持するために運用環境からこのインスタンス変数を削除する必要があります。しかし、Beanが再び有効になったら状態を復元するためにデータを取得する必要が生じ、第2のリポジトリが必要となります。

JEUSに含まれた分散式セッション・マネージャーを第2のリポジトリとして使用します。パッシベーションされるとき、クラスター化されたステートフル・セッションBeanは、フェイルオーバーのためにトランザクションのコミットが正常に実行されるとき、その情報がセッション・マネージャーに渡されます。セッション・マネージャーはDASで設定されます。

設定についての詳細内容は、「[6.3.3. ステートフル・セッションBeanのクラスタリング設定](#)」を参照してください。

## 7.3. 共通設定

本節では、ステートレス・セッションBeanとステートフル・セッションBeanの共通した設定項目について説明します。これらの項目は、JEUS EJBモジュールDDの<jeus-bean>タグの下位に設定されます。

### 7.3.1. オブジェクト・マネージメント関連の設定

ステートレス・セッションBeanは状態がないため、1つのコネクションだけ使用しても構いません。したがってBeanプールのみ使用できます。一方、ステートフル・セッションBeanは状態を持っており、コネクション・プールとBeanプールを両方とも使用できます。以下のようにプールを設定して使用すると、毎回インスタンスを生成する負荷を減らすことができます。

以下は、オブジェクト・マネージメントを設定したXMLの例です。

**[例 7.2] オブジェクト・マネージメントの設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
  . . .
  <beanlist>
    <jeus-bean>
      . . .
      <object-management>
        <bean-pool>
          <pool-min>10</pool-min>
          <pool-max>200</pool-max>
        </bean-pool>
      </object-management>
    </jeus-bean>
  </beanlist>
</jeus-ejb-dd>
```

```

        <connect-pool>
            <pool-min>10</pool-min>
            <pool-max>200</pool-max>
        </connect-pool>
        <passivation-timeout>10000</passivation-timeout>
        <disconnect-timeout>1800000</disconnect-timeout>
    </object-management>
    <jeus-bean>
        . . .
    <beanlist>
        . . .
</jeus-ejb-dd>

```

以下は、<object-management>の下位設定タグについての説明です。

#### ● <bean-pool>

- EJB Beanインスタンス・プールの動作方式を指定します。
- 下位タグは以下のとおりです。

タグ	説明
<pool-min>	プールを初期化する際に作成する初期Beanインスタンスの数です(デフォルト値: 0)
<pool-max>	インスタンスの使用が完了された後、プールに保存可能なインスタンスの最大数です(デフォルト値: 100)
<resizing-period>	プールのサイズを再調整する時間です。「リサイジング」は使用されないBeanインスタンスがプールの最小値まで削除されることを意味します(デフォルト値: 5分、単位: ms)

#### ● <connect-pool>

- クライアントとBeanインスタンスを接続するコネクションの保持数を指定します。(同オプションは、JEUS 6 Fix#8からはセッションBeanで使いません。)
- 下位タグは以下のとおりです。

タグ	説明
<pool-min>	プールを初期化する際に作成する初期コネクション数です(デフォルト値: 0)
<pool-max>	コネクションの使用が完了された後、プールに保存可能なコネクションの最大数です(デフォルト値: 100)

タグ	説明
<resizing-period>	プールのサイズを再調整する時間です。「リサイジング」は使用されないBeanインスタンスがプールの最小値まで削除されることを意味します(デフォルト値: 5分、単位: ms)

#### ● <passivation-timeout>

- 指定された時間の間、クライアントからの要求を受けなかったステートフル・セッションBeanをパッシベーションするときに使用します。すなわち、設定時間が過ぎてもクライアントからの要求がない場合、該当のBeanはパッシベーションの対象となります。

パッシベーションの対象となるBeanをチェックする周期は、EJBエンジンに設定したレゾリューションに従います。パッシベーションが実行されると、メモリーから当該Beanインスタンスが削除され、インスタンスの状態はファイルに保存されます。内部的に分散セッション・サーバーを使用します。

- 同設定は様々な場所で設定可能であり、優先順位は以下のとおりです。
  - 特定のセッションBeanにのみ適用 : jeus-ejb-dd.xmlの<passivation-timeout>
  - すべてのステートフル・セッションBeanに適用 : システム・プロパティのjeus.ejb.stateful.passivate
  - すべてのEJB Beanに(エンティティBeanとセッションBeanの両方)適用 : システム・プロパティのjeus.ejb.all.passivate
  - すべてのEJB Beanに(エンティティBeanとセッションBeanの両方)適用 : DASの<node>/<session-router-config>/<session-router>/<file-db>/<passivation-to>

上記の設定がない場合は、デフォルト値で設定されます。(デフォルト値: 300000(5分)、単位: ms)

#### ● <disconnect-timeout>

- 指定された時間の間、クライアントからの要求を受けなかった場合、クライアントとステートフル・セッションBeanインスタンスのコネクションを切断するときに使用します。コネクションはそれぞれのクライアントとインスタンスとの接続を切断し、コネクション・プールに返されます。

したがって、クライアントは当該コネクションにはそれ以上の要求ができず、使用中のBeanインスタンスは削除されるか、Beanプールに返されます。

- 同設定は様々な場所で設定可能であり、優先順位は以下のとおりです。
  - 特定のセッションBeanにのみ適用(優先順位は順序どおり)
    - ejb-jar.xmlの<stateful-timeout>

b. @StatefulTimeout

c. **(Deprecated)** jeus-ejb-dd.xml <disconnect-timeout>

2. すべてのステートフル・セッションBeanに適用：システム・プロパティのjeus.ejb.stateful.disconnect

3. すべてのEJB Beanに(エンティティBeanとセッションBeanの両方)適用：システム・プロパティのjeus.ejb.all.disconnect

上記の設定がない場合は、システム・プロパティのjeus.ejb.all.disconnectのデフォルト値で設定されます。(デフォルト値: 3600000(1時間)、単位: ms)

---

#### 参考

<passivation-timeout>と<disconnect-timeout>に使用される時間はBeanインスタンスに最後にアクセスした時点から測定されるため、<disconnect-timeout>を<passivation-timeout>より長く設定します。また、<passivation-timeout>はEJBエンジンに設定されるレゾリュションより大きく設定する必要があります。

タイムアウト値を長く設定すると、長時間(およそ10分以上、またはタイムアウトが中断された場合)メモリー内の多くのインスタンスが有効状態で留まるので、システム・リソースを浪費することになります。一方、短すぎるタイムアウト値は(数秒)パッシベーション、アクティベーションなどの作業が頻繁に発生するため、性能の低下やディスコネクト作業によってセッションの遺失可能性が生じます。

---

# 第8章 エンティティBean

エンティティBeanは、EJB 3.0からはJPAに代替されました。Beanを新しく開発する場合はJPAを使用することをお勧めします。ただし、既存のユーザーのためにJEUS EJBエンジンはエンティティBeanをサポートしています。

本章では、JEUS EJBエンジンでエンティティBeanを設定、チューニングするために必要なすべての情報について説明します。

## 8.1. 概要

EJB仕様に準拠し、JEUS EJBエンジンは以下のような3つのエンティティBeanをサポートします。

- Bean Managed Persistence Entity Bean (BMP)
- EJB 1.1仕様に準拠したContainer Managed Persistence Entity Bean (CMP 1.1)
- EJB 2.0仕様に準拠したContainer Managed Persistence Entity Bean (CMP 2.0)

JEUSのエンティティBeanは種類を問わず、すべて「[第4章 EJBの共通特性](#)」で説明した機能と設定コンポーネントを共有するので、JEUS EJBエンジンでエンティティBeanを設定して使用する前に、関連した章の内容を理解しておく必要があります。

Beanの種類によって3つの設定が存在します。

- 3つの種類にすべて適用される共通した設定
- **CMP 1.1とCMP 2.0**にのみ適用され、BMPには適用されない設定
- **CMP 2.0**にのみ適用される設定

以下は、エンティティBeanの種類別の設定です。

**[表 8.1] エンティティBeanの種類別の設定**

設定可能なエンティティ・オプション	BMP	CMP 1.1	CMP 2.0
1. EJB name	√	√	√
2. Export name	√	√	√
3. Local export name	√	√	√

設定可能なエンティティ・オプション	BMP	CMP 1.1	CMP 2.0
4. Export port	√	√	√
5. Export IIOP switch	√	√	√
6. use-access-control	√	√	√
7. Run-as Identity	√	√	√
8. Security CSI Interop.	√	√	√
9. Env. refs	√	√	√
10. EJB refs	√	√	√
11. Resource Refs	√	√	√
12. Resource env. Refs	√	√	√
13. Thread ticket pool settings	√	√	√
14. Clustering settings	√	√	√
15. HTTP invoke	√	√	√
16. JEUS RMI	√	√	√
17. Object management	√	√	√
18. Persistence Optimize	√	√	√
19. CM persistence opt.		√	√
20. Schema info		√	√
21. Relationship map			√

上記のように項目1から16までは、すべてのEJBに共通して適用されます。詳細については、「[第4章 EJBの共通特性](#)」を参照してください。

項目17は、セッションBeanとエンティティBeanの両方に該当しますが、使用方法において異なる部分があるのでその相違点について説明します。また、項目18から21まではエンティティBeanにのみ該当する内容なので、本章で詳しく説明します。

以外に、以下のようなJEUSエンティティEJBに関連する内容についても説明します。

- Default Primary Keyクラス
- EJB QL言語に追加されたJEUSの特定項目
- JEUS Instant EJB QL API



## 参考

JEUSのEJBエンティティBean設定において最も重要な部分は性能チューニングです。そのため、エンジン・モードとサブ・モードなど性能に関連する部分について詳しく説明します。本章の後半に登場する「[8.4. エンティティEJBのチューニング](#)」は、本節の内容を簡単にまとめたものです。

## 8.2. 主要機能

本節では、エンティティBeanの共通機能および各Beanの主要機能について説明します。

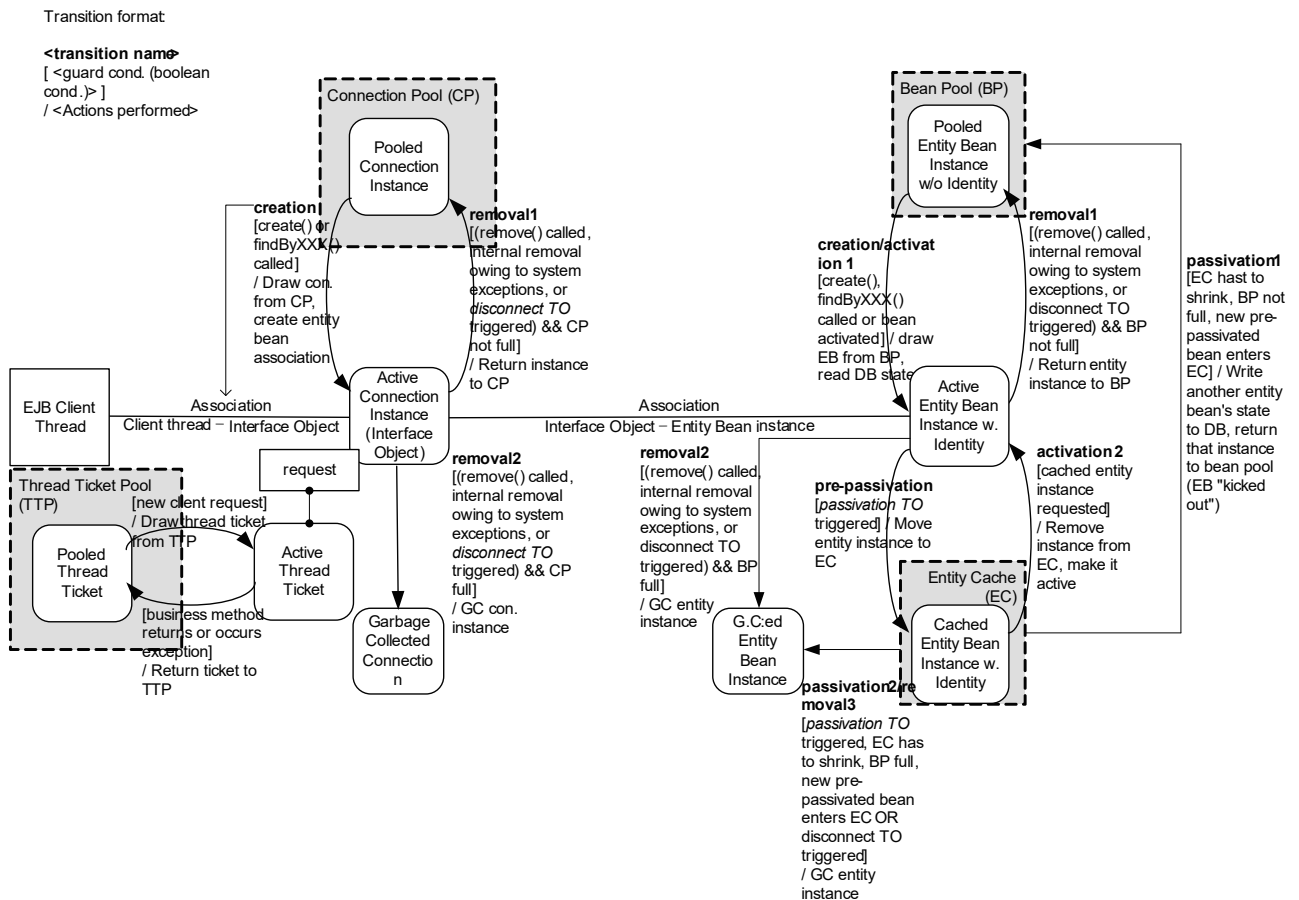
### 8.2.1. 共通機能

3つのエンティティBeanは種類を問わずJEUSエンティティBeanの共通機能を有します。

#### 8.2.1.1. エンティティBeanオブジェクトの管理およびエンティティ・キャッシュ

以下の図は、EJBエンジンでエンティティBeanオブジェクトが管理されるプロセスを表しています。

【図 8.1】 JEUS EJBエンジンでのエンティティBeanのオブジェクトとインスタンスの管理



「object management」とは、EJBエンジンの内部的なBeanインスタンス・プールとコネクション・プールを意味します。このプールはBeanごとに設定され、性能向上とシステム・リソースの効率的な管理のために使用されます。ステートフル・セッションBeanの動作と類似しているため、本節では相違点についてのみ説明します。詳細内容については、[「第7章 セッションBean」](#)を参照してください。

エンティティBeanは、コネクション・プール以外にエンティティ・キャッシュを使用します。エンティティEJBがパッシベーションされる時(パッシベーション・タイムを過ぎた場合)、実際にそのBeanインスタンスは直ちにパッシベーションされません。(ステートフル・セッションBeanの場合にはパッシベーションされます。)その代わりに、エンティティ・キャッシュに渡され、すべてのエンティティBeanインスタンスが有効状態で保持されます。(すなわち、識別可能な有効状態を保持します。)別のプールと同様、エンティティ・キャッシュもランタイム・メモリーに保持され、有効状態からパッシベーションされるまでの間の状態でランタイム・キャッシュで動作します。

以下は、エンティティBeanインスタンスが実際にパッシベーションされる場合です。

エンティティ・キャッシュがいっぱいの場合にパッシベーションされます。新しいエンティティBeanがキャッシュに入ろうとすると、同キャッシュにすでにBeanインスタンスがいっぱいの場合、キャッシュ内のBeanインスタンスが強制的に押し出されます。押し出されたエンティティBeanインスタンスはパッシベーションされ、ディスクに書き込まれます。同BeanインスタンスはBeanプールに返されるか、Beanプールがいっぱいの場合はページされます。結果的に新しいエンティティBeanインスタンスは、強制的に押し出されたBeanインスタンスのスペースに格納されます。

キャッシュ・サイズは<persistence-optimize>/<entity-cache-size>で調整できます。必ずフルになったときにキャッシュ・サイズを減らすのではなく、この設定はヒントとして使われます。また、強制的にページされるBeanインスタンスを選択する方法もキャッシュに留まった時間順ではありません。

このようなキャッシュを使用すると、多くの時間が必要なパッシベーションを効果的に管理できるメリットがあります。また、パッシベーションされたエンティティBeanを再び有効化するときDBにアクセスする必要がなくなり、キャッシュに存在するインスタンス(存在する場合)を使って効果的に管理できます。

オブジェクト・プールの標準パッシベーション・タイムアウト値が時間的な制約をコントロールするパッシベーション値であるなら、エンティティ・キャッシュはメモリーの制約をコントロールするパッシベーション設定値になります。これがエンティティ・キャッシュを使用する主な理由です。

---

#### 注

エンティティ・キャッシュの設定は、「object management」の設定ではなく、一般的なエンティティBeanの永続性(Persistence)の最適化設定です。詳細内容については、[「8.3. エンティティEJBの設定」](#)を参照してください。

---

### 8.2.1.2. エンジン・モード・セレクションを利用したejbLoad()永続性の最適化

エンティティBeanに対し、ejbLoad()メソッドを呼び出すタイミングを決めるのはEJBエンジン(コンテナ)です。このメソッドはDBの状態情報やBeanインスタンスのランタイム状態情報を同期化するためにBMPとCMPで

使用します。ejbLoad()メソッドはDBから1つのレコードを読み取り、その結果をBean内部の特性値を設定するために使用されます。

ejbLoad()メソッドはエンティティBeanのライフサイクルの間、少なくとも1回は呼び出されます。場合によっては、Beanインスタンスの呼び出し直前にejbLoad()メソッドの呼び出しが必要な場合があります。

DBでBeanの状態情報を読み込むシナリオは以下のとおりです。

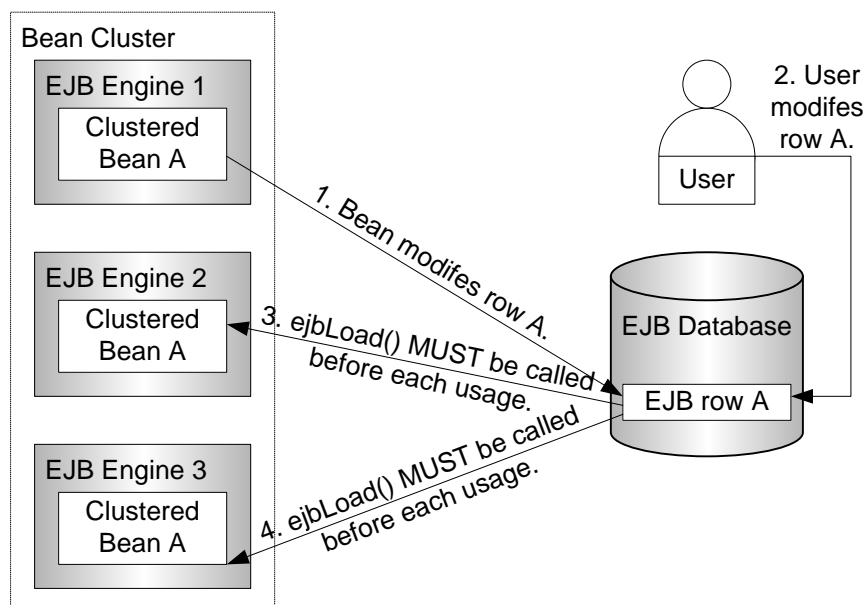
- エンティティBeanが複数のEJBエンジンにクラスターされている場合、その中の1つのBeanがDBの1つのレコードを変更することができます。すなわち、クラスターされたエンティティBeanは相互通信ができないため、特定のエンティティBeanインスタンスの状態情報が最新のものであることが保証できません。

- 外部要素(ユーザーまたはシステム)がEJBと連動しているDBを変更する場合があります。

この場合、EJBエンジンの特定のEJBインスタンスは、DBの内容が正常に反映されない状態になります。

以下の図は、エンティティBeanがクラスターされている場合と外部でBeanのDB行を修正した場合、ejbLoad()が周期的に呼び出される2つのシナリオです。

**【図 8.2】 ejbLoad()の周期的な呼び出しシナリオ**

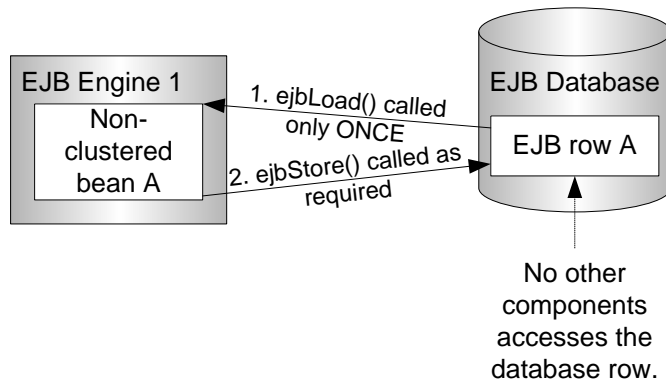


上記のシナリオ以外は(クラスターリングを使用せず、外部要素によるEJBのDBデータが変更されないと保証できる場合)エンティティBeanのejbLoad()動作を最適化することができます。

EXCLUSIVE\_ACCESSエンジン・モードをBeanに適用すると最適化が可能です。このモードを使用すると、ejbLoad()はBeanがインスタンス化される時1度だけ呼び出されます。これによりDBのアクセス量を50%ほど減らすことができます。

以下の図は、EXCLUSIVE\_ACCESSモードが使用されるシナリオです。EXCLUSIVE\_ACCESSは1つのBeanがDB行を使用し、ほとんどのejbLoad()は最適化されます。

**[図 8.3] EXCLUSIVE\_ACCESSモードのシナリオ**



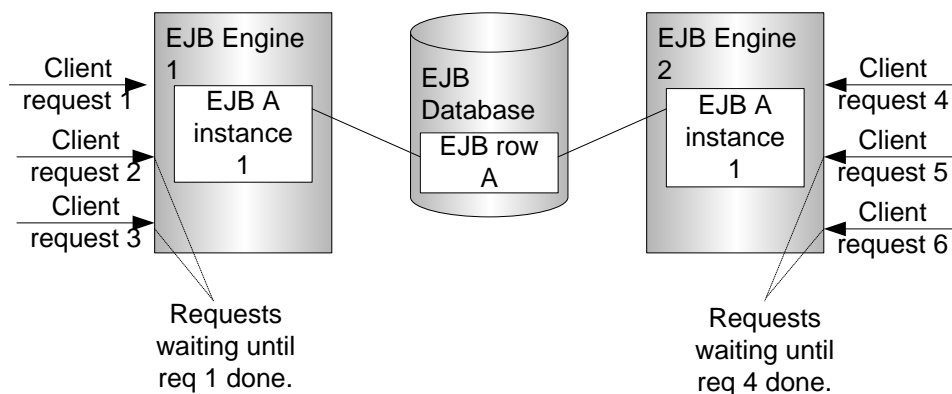
Beanをクラスタリングするか、外部要素によるEJBデータの変更が必要な場合は、SINGLE\_OBJECTまたはMULTIPLE\_OBJECTエンジン・モードを使用します。2つのモードのうち1つが使用されると、EJBエンジンはEJBクライアントの要求が受信されるたびにejbLoad()メソッドを呼び出します。これによって各エンティティBeanの性能は低下しますが、すべての状態情報がDBによって管理されるため、別のエンジンとBeanに負荷を分散させることができます。

以下は、SINGLE\_OBJECT MULTIPLE\_OBJECTモードの相違点についての説明です。

- SINGLE\_OBJECTエンジン・モード

SINGLE\_OBJECTでは、各EJBエンジンの1つのBeanインスタンスが、すべてのクライアントの要求を処理します。すなわち、最初の要求が処理されている間に送られた要求は待機することになります。

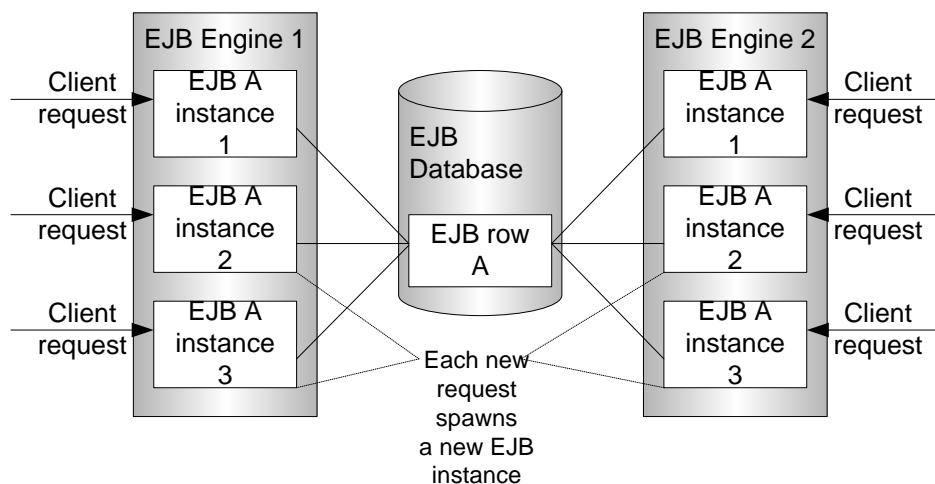
**[図 8.4] SINGLE\_OBJECTエンジン・モード**



- MULTIPLE\_OBJECTエンジン・モード

MULTIPLE\_OBJECTモードでは、このような制約は適用されず、EJBエンジンが新しいエンティティBeanインスタンスを生成して、新しいEJBクライアントの要求を処理します。つまり、MULTIPLE\_OBJECTエンジン・モードでは新しいEJBインスタンスが割り当てられます。

[図 8.5] MULTIPLE\_OBJECTエンジン・モード



以下は、エンジン・モードを選択するときの長短所についての説明です。

[表 8.2] 3つのエンジン・モードの長短所

区分	EXCLUSIVE	SINGLE	MULTIPLE
クラスタリングできるのか?	No	Yes	Yes
外部エンティティがDBにアクセスできるのか?	No	Yes	Yes
DBコネクションが効率的に使用されるのか?(ejbLoad()の呼び出し頻度)	Yes	No	No
長時間のトランザクションの場合、適切なオプションをサポートするのか?	No	No	Yes
1つのエンジンに適合した、単一処理の有効なモードは?	Not applicable	Yes	No
1つのエンジンでEJBを同時処理するために要求される効果的な多重処理モードは?	Not applicable	No	Yes

3つのエンジン・モードのうち、適切なモードを選択するための基準は以下のとおりです。

- エンティティBeanに多くの要求が送られるとき、十分なEJBエンジンが設定されている場合は、SINGLE\_OBJECTモードを選択してエンジンにBeanをクラスタリングで構成します。
- エンティティBeanに多くの要求が送られるとき、限定されたEJBエンジンが設定されている場合は、MULTIPLE\_OBJECTモードを選択してエンジンにBeanをクラスタリングで構成します。
- MULTIPLE\_OBJECTとSINGLE\_OBJECTを選択するときには考慮する事項の1つは、1つのトランザクションを実行するために必要な総時間です。長時間かかるトランザクションの場合は、MULTIPLE\_OBJECTを選択することをお勧めします。

- エンティティBeanに少量の要求が送られるときは、EXCLUSIVE\_ACCESSモードを選択し、1つのEJBエンジンにBeanをデプロイします。

EJBクラスタリングについての詳細情報は、「[第6章 EJBのクラスタリング](#)」を参照してください。

## 8.2.2. BMP & CMP 1.1

本節では、ejbStore()呼び出しを最適化する方法について説明します。

---

### 参考

本節で説明するejbStore()呼び出しの最適化方法は、BMPとCMP 1.1 Beanにのみ適用されます。CMP 2.0は別の方法で最適化します。

---

### 8.2.2.1. Non-modifyingメソッドによるejbStore()永続性の最適化

ejbStore()はEJBエンジンによってDBのBean状態情報が保持される必要があると判断される場合に呼び出されます。通常、DBへのアクセスは各トランザクションのコミット、またはBeanインスタンスがパッシベーションされる前に行われます。

しかし、トランザクション中にBeanの状態に変更がなければ、ejbStore()の呼び出しは必要ありません。

JEUS EJBエンジンはBeanの状態変更を自動的に予測することができないため、開発者またはデプロイヤーは、ejbStore()を呼び出すタイミングをEJBエンジンが判断できるようにヒントを提供する必要があります。このヒントは、Non-modifyingメソッドの一覧で提供されます。Non-modifyingメソッドは読み取りのみ許可するメソッド(getter)として、エンティティBeanのDB内部の状態は変更しません。

Non-modifyingメソッドの一覧を確認して、EJBエンジンはejbStore()メソッドの呼び出し有無を決めます。トランザクション中、またはBeanが有効化された状態にNon-modifyingメソッドのみ実行されると、EJBエンジンはejbStore()呼び出しが不要だと判断し、その手順をスキップします。これは、全体的な性能が向上されることを意味します。

## 8.2.3. CMP 1.1/2.0

本節では、CMP 1.1/2.0の主要機能について説明します。

### 8.2.3.1. ejbLoad()、ejbFind()、CM永続性の最適化

「[8.2.1.2. エンジン・モード・セレクションを利用したejbLoad\(\)永続性の最適化](#)」と「[8.2.2.1. Non-modifyingメソッドによるejbStore\(\)永続性の最適化](#)」でejbLoad()とejbStore()メソッドの呼び出しタイミングについて説明しました。BMPとCMPは、ejbLoad()とejbStore()メソッドを使用するため、BMPとCMPエンティティBeanにのみ適用されました。

本節では、性能向上のためにejbLoad()とejbFind()メソッドを最適化する方法について説明します。ejbLoad()、ejbFind()の実装方法はEJBエンジンから影響を受けるため、CMP Bean(1.1と2.0バージョン)に限定されます。BMPにおいて同メソッドの実装と動作方式は開発者によって決まります。

EJBエンジンがCMP ejbLoad()、ejbFind()を呼び出すと、以下のようなエンジンの下位モードを選択して動作方式を最適化することができます。3つのモードについての詳細内容は省略します。

モード	説明
ReadLocking	ejbLoad()を呼び出すとき、EJBエンジンがDBの共有ロック(Shared Lock)を有することができます。このタイプのロックは、別のBeanが同じレコードを読み込むことはできますが、書き込みはできません
WriteLocking	ejbLoad()を呼び出すとき、EJBエンジンがDBの排他ロック(Exclusive Lock)を有することができます。別のBeanはDBに読み書きできません
WriteLockingFind	ejbLoad()とejbFind()を呼び出すとき、EJBエンジンがDBの排他ロック(Exclusive Lock)を有することができます。別のBeanはDBに読み書きできません

上記の3つのモードが使用される状況は以下のとおりです。

- CMP Beanが、書き込みより読み込みを頻繁に行う場合は、常にReadLockingモードを選択します。
- CMP Beanが、読み込みより書き込みを頻繁に行う場合は、WriteLockingモードか、WriteLockingFindモードを選択します。

CMPでは、エンジン・モード、Non-modifyingメソッド、エンジン内の下位モードが規則に基づいて設定される必要があります。パラメータの設定については、「[8.4. エンティティEJBのチューニング](#)」を参照してください。

さらに、CMP Beanではjava.sql.ResultSetのフェッチ・サイズ、EJBエンジンが起動されときのキャッシング有無、EJB Beanインスタンスのキャッシング・タイミングを決める<init caching>が設定できます。

### 8.2.3.2. エンティティBeanのスキーマ情報

EJBエンジンがBeanの永続性を管理するには(CMP)、エンティティBeanインスタンス・フィールドやDBの表、列とのマッピングを宣言して設定する方法が必要です。また、DBソースも指定します。この情報はEJB XML環境に<schema info>で記述します。

CMP 1.1のこのタグにはEJBエンジンがCMP 1.1 finderメソッドを生成するとき、基本的に使用されるSQL文も含まれます。

### 8.2.3.3. 主キー(Primary Key)の自動生成

開発の際、場合によっては主キーを生成せずに、EJBインスタンスを生成する場合があります。この場合、JEUS EJBエンジンが新しいEJBインスタンスに割り当てする固有の主キーを生成するように設定することができます。

JEUS EJBエンジンはDBと連動して同機能を提供しており、CMP 1.1とCMP 2.0に適用されます。この機能を使用すると主キーを気にせず簡単にEJBを開発することができます。

以下は、主キーの自動生成機能の使用方法和適用手順についての説明です。

1. 開発者は習慣的にパラメータなしでCMP1.1/CMP2.0のcreate()を呼び出す場合がありますが、そうするとcreate()メソッドがEJBの主キーとなるパラメータを持っていないことになります。

```
InitialContext ctx = new InitialContext();
BookHome bHome = (BookHome) ctx.lookup("bookApp");
Book b = bHome.create("JEUS EJB Guide", "Park Sungho");
// Start using "b".
. . .
```

上記の例で、EJBホームであるBookHomeを検索し、EJB Homeからパラメータが2つのcreate()を呼び出します。("JEUS EJB Guide"と"Park Sungho")このパラメータは新しいBookのタイトルと著者名を意味します。著者は異なるが、同じタイトルのBookが存在する可能性があるため、このパラメータを主キーとして使用するの是不適です。

この機能をサポートするため、EJBエンジン("A")は必ず固有の主キーを生成して新しいBookに割り当てます。そして複数のエンジン("B"、"C")がクラスタリングされた状態でも同じことが同時に発生する可能性があるため、注意が必要です。

各エンジンは広範なクラスタリング中にも固有の主キーが取得できる必要があるため、1つの中央リポジトリが必要です。このリポジトリは1つの列とデータで構成されたDBとして実装されます。

この1つの値はEJBエンジンと連動して増加し続ける主キー・カウンター(または、主キージェネレーター)です。DBは同方法を利用して、常にクラスタリングされたEJBエンジンが読み取れる新しいかつ固有な主キーを保持することができます。

2. 主キーが存在しないcreate()が呼び出されると、EJBエンジンの「A」は中央リポジトリ(主キーDB)にアクセスします。同リポジトリから整数の主キーを取得し、Bookインスタンスに割り当てます。
3. EJBエンジンの「A」はリポジトリに予め指定されている値の分だけ主キーを増やします。デフォルト値(キー・キャッシュ・サイズ)は「1」ですが、1以上の値を使用することをお勧めします。(例:20)

キー・キャッシュ・サイズが「20」の場合、EJBエンジンの「A」は「19」(20-1)を割り当てます。次に主キーを読み込む別のEJBエンジン(「B」と「C」)は、最後に読み取られた値より20大きい値を読み込むことになります。

すなわちEJBエンジン「A」は、19個までは外部の主キー・リポジトリにアクセスせず、EJBインスタンスが生成できることになるため、性能向上に大きく寄与します。エンジン「A」は、割り当てられた主キーのカウンターを保持し、この値を完全消費した場合のみ外部から新しい主キーを読み込みます。内部的なカウンターや主キーを生成するときに使われます。



以下は、主キー自動生成の特徴です。

- OracleとMS SQLサーバーは主キー・シーケンスを自動的に提供します。

主キー自動設定についての詳細内容は、『[Oracle DBにおける主キー生成の設定](#)』と『[MS SQLサーバでの主キー自動生成の設定](#)』を参照してください。

- 非Oracleと非MS SQLサーバーは主キー表を直接作成する必要があります。

詳細内容については、『[その他DBの主キー自動生成](#)』を参照してください。

- 主キー自動生成機能を使用するには、DBがTransaction Isolation Level - TRANSACTION\_SERIALIZABLEをサポートする必要があります。

- 主キーはEJBで定義される必要があります。

主キーは、java.lang.Integerタイプの単一キーであり、EJB DDに宣言される必要があります。

---

#### 参考

主キー自動生成機能は、TRANSACTION\_SERIALIZABLE(Isolation Level)をサポートする必要があります。すなわち、Oracle、MS SQLサーバー以外に対しては、この分離レベル(Isolation Level)をサポートしているか否かを確認してください。

---

以下は、各DBに設定する方法です。

## Oracle DBでの主キー生成の設定

Oracle DBでの主キーの自動生成は、Oracleで使用するシーケンス・オブジェクトを使います。主キーの自動生成機能を設定する前にシーケンス・オブジェクトが生成されている必要があります。シーケンス設定についての内容は、Oracleの関連文書を参照してください。

以下は、Oracle DBで主キーを生成するjeus-ejb-dd.xmlの例です。

**[例 8.1] Oracle DBでの主キー生成の設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <schema-info>
```

```

        . . .
        <data-source-name>MYORACLEDB</data-source-name>
        <auto-key-generator>
            <generator-type>
                Oracle
            </generator-type>
            <generator-name>
                my_generator
            </generator-name>
            <key-cache-size>
                20
            </key-cache-size>
        </auto-key-generator>
    </schema-info>
    . . .
</jeus-bean>
. . .
</beanlist>
. . .
</jeus-ejb-dd>

```

上記の例では、主キーの生成タイプを「Oracle」に設定しており、<generator-name>を「my\_generator」にして<key-cache-size>を「20」に設定しています。

---

#### 参考

<key-cache-size>はOracleのシーケンス・オブジェクトのSEQUENCE INCREMENT値と一致する必要があります。

Oracle DBは<data-source-name>から選択します。(こちらでは「MYORACLEDB」が使われていますが、JEUSMain.xmlのDBコネクション・プールのJNDI名です。)

設定についての詳細内容は、『JEUS サーバガイド』を参照してください。

---

## MS SQLサーバーでの主キー自動生成の設定

MS SQLサーバーは自動的に固有のIDENTITY列に主キーを保持するため、主キーの自動生成を簡単に設定することができます。

以下は、MS SQLサーバーで主キーの自動生成を設定したjeus-ejb-dd.xmlの例です。

### [例 8.2] MS SQLサーバーでの主キー自動生成の設定 : <<jeus-ejb-dd.xml>>

```

<jeus-ejb-dd>
    <beanlist>
        . . .
    </beanlist>
</jeus-ejb-dd>

```

```

    <jeus-bean>
        . . .
        <schema-info>
            . . .
            <data-source-name>MSSQLDB</data-source-name>
            <auto-key-generator>
                <generator-type>
                    MSSQL
                </generator-type>
            </auto-key-generator>
        </schema-info>
    </jeus-bean>
    . . .
</beanlist>
</jeus-ejb-dd>

```

<generator-type>に「MSSQL」を入力します。MS SQLサーバーは<data-source-name>から選択されます。

## **その他DBの主キー自動生成の設定**

OracleとMS SQLサーバー以外に別のDBを利用して主キーを自動生成する場合、必ず以下の手順に従います。

1. DBに表を1つ生成します。表は1つの列と行で構成し、値は「0」に設定します。

以下は、主キー値を管理する表です。

Primary Key generator table PrimKeyTable	
	PrimKeyGeneratorColumn (主キー値が保存される列)
Row 1	0 (主キー値の初期値)

上記の表は以下のSQLで生成されます。

```

CREATE table PrimKeyTable (PrimKeyGeneratorColumn int);
INSERT into PrimKeyTable VALUES (0);

```

2. 表を生成し、jeus-ejb-dd.xmlを以下のように修正します。

**[例 8.3] その他DBの主キー自動生成の設定 : <<jeus-ejb-dd.xml>>**

```

<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>

```

```

    . . .
    <schema-info>
        . . .
        <data-source-name>MYDB</data-source-name>
        <auto-key-generator>
            <generator-type>
                USER_KEY_table
            </generator-type>
            <generator-name>
                PrimKeyTable
            </generator-name>
            <sequence-column>
                PrimKeyGeneratorColumn
            </sequence-column>
            <key-cache-size>
                20
            </key-cache-size>
        </auto-key-generator>
    </schema-info>
    . . .
</jeus-bean>
. . .
</beanlist>
. . .
</jeus-ejb-dd>

```

上記の例で、タイプは「USER\_KEY\_table」、表名は「PrimKeyTable」です。列名は「PrimKeyGeneratorColumn」、キー・キャッシュ・サイズは「20」に設定しています。

DBを使用するために設定する<data-source-name>タグの値は、JEUSMain.xmlのDBコネクション・プールの<export-name>と一致する必要があります。

---

## 参考

設定についての詳細内容は、『JEUS サーバガイド』を参照してください。

---

## 8.2.4. CMP 2.0

本節では、CMP 2.0の主要機能について説明します。

### 8.2.4.1. Entity Bean Relationship Mapping

EJB 2.0仕様にはDBリレーションシップをサポートするため、CMPリレーションシップの概念が紹介されています。CMP 2.0 Beanとリレーションシップを使用するときは、デプロイヤーがJEUS EJB DDに追加情報を提供する必要があります。

## 8.2.4.2. JEUS EJB QL Extension

CMP 2.0 Beanはejb-jar.xmlのEJB QL文と連携されているホーム・インターフェースのfindByXXXX()メソッドを持つ必要があります。JEUSは標準EJB QL言語に加え、いくつかの追加事項をサポートします。追加事項は、ejb-jar.xmlに使用されるか、Instant EJB QL要求に使用できます。

「[8.3.3.3. JEUS EJB QLの拡張機能\(Extension\)](#)」には、上記の例が含まれたejb-jar.xmlが含まれています。

## 8.2.4.3. Instant EJB QL

クライアント・コード内でCMP Beanの集合を検索する場合、Beanのホーム・インターフェースに宣言されているfindByXXXX()メソッドに依存するしかありません。検索方法が複雑な場合は、findByXXXX()だけでは検索できない場合があります。

JEUSではこのような場合に備え、クライアント・コードでEJB QL selectの問合せ文が定義できる非標準インターフェースを提供しています。このインターフェースは、jeus.ejb.Bean.objectbase.EJBInstanceFinderと呼ばれています。JEUS EJB DDにBeanのインスタンスQLを有効にした場合、エンティティBeanのホーム・インターフェースで実装します。同インターフェースはユーザーのEJB QLをホーム・インターフェースに渡すメソッドであるfindWithInstantQL(String ejbQLString)を持っています。このメソッドは問合せに該当するEJB インターフェースのjava.util.Collectionオブジェクトを返します。

同メソッドの例は、「[8.2.3.3. 主キー\(Primary Key\)の自動生成](#)」を、該当するAPIについての説明は、「[付録 B. Instant EJB QL APIリファレンス](#)」を参照してください。

## 8.3. エンティティEJBの設定

本節では、以下のJEUSエンティティEJBの設定方法について説明します。

- すべてのエンティティBeanに適用可能な設定
  - 基本かつ共通の設定
  - オブジェクト・プール
  - エンティティ・キャッシュの設定を含む永続性の最適化
- CMPIにのみ適用可能な設定
  - CM永続性の最適化
  - スキーマ情報の設定
- CMP 2.0にのみ適用可能な設定
  - リレーションシップ・マッピングの設定

これらの特性は、JEUS EJBモジュールDD(jeus-ejb-dd.xml)の<beanlist>タグの下位の<jeus-bean>に設定されます。

## 8.3.1. 共通設定

以下は、すべてのエンティティBeanに適用できる基本設定です。

### 8.3.1.1. 基本かつ共通項目の設定

「第4章 EJBの共通特性」で説明したJEUSでサポートするすべてのBeanに適用できる設定は、全種類のエンティティBeanにも適用可能です。

以下は、BMPエンティティBeanの基本的なXMLタグの例です。

**[例 8.4] エンティティEJBの基本かつ共通項目の設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
<beanlist>
  <jeus-bean>
    <ejb-name>account</ejb-name>
    <export-name>ACCOUNTEJB</export-name>
    <local-export-name>
      LOCALACCOUNTEJB
    </local-export-name>
    <export-port>7654</export-port>
    <export-iiop>true</export-iiop>
    . . .
  </jeus-bean>
  . . .
</beanlist>
. . .
</jeus-ejb-dd>
```

### 8.3.1.2. オブジェクト・マネージメント関連の設定

以下は、オブジェクト・マネージメント関連の設定がされているXMLの例です。<jeus-bean>の<object-management>タグ内で構成されます。

**[例 8.5] オブジェクト・マネージメント関連の設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
```

```

<beanlist>
  <jeus-bean>
    . . .
    <object-management>
      <bean-pool>
        <pool-min>10</pool-min>
        <pool-max>200</pool-max>
      </bean-pool>
      <connect-pool>
        <pool-min>10</pool-min>
        <pool-max>200</pool-max>
      </connect-pool>
      <capacity>5000</capacity>
      <passivation-timeout>10000</passivation-timeout>
      <disconnect-timeout>1800000</disconnect-timeout>
    </object-management>
  </jeus-bean>
  . . .
</beanlist>
. . .
</jeus-ejb-dd>

```

以下は、設定タグについての説明です。

- **<bean-pool>**

- EJB Beanインスタンス・プールの動作方式を指定します。
- 下位タグは以下のとおりです。

タグ	説明
<pool-min>	プールを初期化するときに生成する初期Beanインスタンスの数とプールに保持するBeanインスタンスの最小数(リサイジング作業時にプールに残すインスタンス数)です(デフォルト値: 0)
<pool-max>	<p>インスタンスの使用後、プールへの保存可否を決めるプールが保持できるBeanインスタンスの最大数です。セッションBeanは&lt;pool-max&gt;設定値まで作成できるため、それ以上の要求が同時に送られるとEJB例外が発生します。</p> <p>一方、エンティティBeanはBeanインスタンスの作成には制限がなく、使用後にプールに保持できる数を制限します(デフォルト値: 100)</p>
<resizing-period>	プールのサイズを再調整する時間です。「リサイジング」とは、使用されないBeanインスタンスがプールの最小値まで削除されることを意味します(デフォルト値: 5分、単位: ms)

- **<connect-pool>**

- クライアントとBeanインスタンスを接続するコネクションをいくつまで保持するかを設定します。当該オプションはセッションBeanでは使用しません。
- 下位タグは以下のとおりです。

タグ	説明
<pool-min>	プールに保持するコネクションの最小数です。リサイジング時にプールに残すインスタンス数です(デフォルト値: 0)
<pool-max>	コネクションの使用後、プールへの保存可否を決めるプールが保持できるコネクションの最大数です(デフォルト値: 100)
<resizing-period>	プールのサイズを再調整する時間です。「リサイジング」とは、使用されないコネクションがプールの最小数まで削除されることを意味します(デフォルト値: 5分、単位: ms)

- **<capacity>**

- 生成されると予想されるBeanインスタンスの最大数です。エンティティBeanにのみ使われます。この値は、EJBと連携される内部クライアント・セッション・データの効率的な構成のために使用されます。(デフォルト値: 10000)

- **<passivation-timeout>**

- 指定された時間の間にクライアントの要求を受けないBeanをパッシベーションします。

設定時間が過ぎるまでクライアントの要求を受けなかったBeanインスタンスがパッシベーションの対象となります。パッシベーション対象のBeanインスタンスがメモリーに残っている数を<persistence-optimize>/<entity-cache-size>で調整することができます。

- ステートフル・セッションBeanの場合はエンティティ・キャッシュが存在しないため、設定時間の間に要求がないとパッシベーションされます。一方、エンティティBeanの場合は、エンティティ・キャッシュに留まってからパッシベーションされます。パッシベーションされるBeanをチェックする周期は、EJBエンジンに設定したレゾリューションに従います。(デフォルト値: 5分)

パッシベーションが実行されると、該当するBeanインスタンスがメモリーから削除されます。

- この設定は様々な場所で設定可能であり、優先順位は以下のとおりです。

1. 特定のセッションBeanにのみ適用 : jeus-ejb-dd.xmlの<passivation-timeout>
2. すべてのエンティティBeanに適用 : システム・プロパティーのjeus.ejb.entity.passivate



3. すべてのEJB Beanに(エンティティBeanとセッションBean)適用 : システム・プロパティの  
jeus.ejb.all.passivate

設定しない場合はデフォルト値で設定されます。(デフォルト値: 300000(5分)、単位: ms)

● <disconnect-timeout>

- 指定された時間の間にクライアントの要求を受けなかった場合、クライアントとBeanインスタンスの接続を切断します。切断された接続は接続・プールに返されます。

そのため、クライアントは当該接続にこれ以上の要求ができず、使用中のBeanインスタンスは削除されるか、Beanプールを使用している場合はBeanプールに返されます。

- この設定は様々な場所で設定可能であり、優先順位は以下のとおりです。

1. 特定のセッションBeanにのみ適用 : jeus-ejb-dd.xmlの<disconnect-timeout>
2. すべてのエンティティBeanに適用 : システム・プロパティのjeus.ejb.stateful.disconnect
3. すべてのEJB Beanに(エンティティBeanとセッションBean)適用 : システム・プロパティの  
jeus.ejb.all.disconnect

設定しない場合は、システム・プロパティjeus.ejb.all.disconnectのデフォルト値である3600000(1時間)で設定されます。(単位: ms)

---

**参考**

<passivation-timeout>と<disconnect-timeout>に使用される時間はBeanインスタンスにアクセスした最終時点から測定されます。そのため、<disconnect-timeout>を<passivation-timeout>より長く設定する必要があります。また、<passivation-timeout>はEJBエンジンに設定されるレゾリュションより大きく設定します。

タイムアウトを長く設定すると、長時間(およそ10分以上、またはタイムアウトが中断された場合)の間、メモリ内に多くのインスタンスが有効状態で留まることになるため、システム・リソースの浪費になります。一方、タイムアウト値が短すぎる(数秒)場合は、パッシベーション、アクティベーションなどの作業が頻繁に行われるため、性能が低下する場合があります。また、切断(disconnect)によってセッションを遺失する可能性もあります。

---

### 8.3.1.3. ejbLoad()とejbStore()永続性の最適化設定

永続性メソッド呼び出しの最適化は、jeus-ejb-dd.xmlの各Bean DDに設定されています。

以下は、ejbLoad()、ejbStore()メソッドの永続性の最適化を設定したXMLの例です。<jeus-bean>の<persistence-optimize>タグ内で構成されます。

**[例 8.6] ejbLoad()とejbStore()永続性の最適化設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
  . . .
  <beanlist>
    . . .
    <jeus-bean>
      . . .
      <persistence-optimize>
        <engine-type>SINGLE_OBJECT</engine-type>
        <non-modifying-method>
          <method-name>myBusinessmethod</method-name>
          <method-params>
            <method-param>
              java.lang.String
            </method-param>
            <method-param>int</method-param>
            <method-param>double</method-param>
          </method-params>
        </non-modifying-method>
        <non-modifying-method>
          <method-name>myBusinessmethod2</method-name>
          <method-params>
            <method-param>
              java.lang.String
            </method-param>
            <method-param>int</method-param>
          </method-params>
        </non-modifying-method>
        <entity-cache-size>500</entity-cache-size>
        <update-delay-till-tx>
          false
        </update-delay-till-tx>
        <include-update>
          true
        </include-update>
      </persistence-optimize>
    </jeus-bean>
    . . .
  </beanlist>
  . . .
</jeus-ejb-dd>
```

以下は、設定タグについての説明です。

タグ	説明
<engine-type>	以下のいずれかに設定します。各設定の相違点については、「 <a href="#">8.2. 主要機能</a> 」を参照してください <ul style="list-style-type: none"> <li>EXCLUSIVE_ACCESS (デフォルト値)</li> <li>SINGLE_OBJECT</li> <li>MULTIPLE_OBJECT</li> </ul>
<non-modifying-method>	ejbStore()呼び出しの必要性に関するヒントをEJBエンジンに提供します。このタグは、CMP 2.0 Beanには適用されません
<entity-cache-size>	パッシブセッションの対象となるエンティティBeanインスタンスのための内部キャッシュ・サイズを指定します。キャッシュに設定可能なエンティティBeanインスタンスの最大サイズまで指定できます。サイズが大きいほど良い性能を出しますが、システム・リソース(主メモリー)はその分多く使われます(デフォルト値: 2000)
<update-delay-till-tx>	ブーリアン値で、トランザクションがコミットされるまで、EJBデータのアップデートやインサートが遅延されるか否かを示します。  この値が「false」の場合、すべてのアップデート、インサート作業は即実行されるため、同じトランザクション内ではcommit()が呼び出される前でも変更内容が確認できますが、性能には悪影響を及ぼします(デフォルト値: true)
<include-update>	<schema-info><jeus-query><include-updates>のデフォルト値を指定します。  この値が「true」の場合、finderメソッドが呼び出される間に生成されたアップデートがコミットされるため、finderメソッドが実行される間にアップデートされた情報が照会できます(デフォルト値: false)

## 8.3.2. CMP 1.1/2.0

以下は、CMP 1.1/2.0の設定についての説明です。

### 8.3.2.1. ejbLoad()とejbFind() CM永続性の最適化設定

CMP 1.1と2.0 Beanには、ejbLoad()永続性の最適化を設定することができます。詳細内容については、「[8.3.1.3. ejbLoad\(\)とejbStore\(\)永続性の最適化設定](#)」を参照してください。

以下は、ejbLoad、ejbFind CM永続性の最適化を設定したXMLの例です。<jeus-bean>の<cm-persistence-optimize>タグ内に設定されます。

**[例 8.7] ejbLoad()とejbFind() CM永続性の最適化設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
```

```

<beanlist>
    . . .
    <jeus-bean>
        . . .
        <cm-persistence-optimize>
            <subengine-type>WriteLocking</subengine-type>
            <fetch-size>80</fetch-size>
            <init-caching>true</init-caching>
        </cm-persistence-optimize>
    </jeus-bean>
    . . .
</beanlist>
. . .
</jeus-ejb-dd>

```

以下は、設定タグについての説明です。

タグ	説明
<subengine-type>	以下のいずれかに設定します <ul style="list-style-type: none"> <li>– ReadLocking (デフォルト値)</li> <li>– WriteLocking</li> <li>– WriteLockingFind</li> </ul>
<fetch-size>	結果セット(ResultSet)を使用して一度に取得できるレコード数を指定します(デフォルト値: 10)
<init-caching>	ブーリアン値が有効になると、EJBエンジンはDB表の各レコードの対し、事前にインスタンス化されたEJBエンティティ・インスタンスを生成します。この作業はエンジンが起動されるときに実行されます。無効の場合、EJBインスタンスはホーム・インターフェースのcreate()、find-ByXXXX()、または類似したメソッド呼び出しによってのみ生成されます(デフォルト値: false)

### 8.3.2.2. DBスキーマ情報の設定

CMP 1.1/2.0ではDB表と列に対応する、EJBインスタンス・フィールドとのマッピングを含むDBスキーマ情報を設定します。

以下は、DBスキーマ情報を設定したXMLの例です。<jeus-bean>の<schema-info>タブ内に構成されます。

**[例 8.8] DBスキーマ情報の設定 : <<jeus-ejb-dd.xml>>**

```

<jeus-ejb-dd>
    . . .

```

```

<beanlist>
    . . .
    <jeus-bean>
        . . .
        <schema-info>
            <table-name>ACCOUNT</table-name>
            <cm-field>
                <field>id</field>
                <column-name>ID</column-name>
                <type>NUMERIC</type>
                <exclude-field>false</exclude-field>
            </cm-field>
            <cm-field>
                <field>customer_addr</field>
                <column-name>customer_address</column-name>
                <type>VARCHAR(30)</type>
                <exclude-field>false</exclude-field>
            </cm-field>
            <creating-table>
                <use-existing-table/>
            </creating-table>
            <deleting-table>true</deleting-table>
            <prim-key-field>
                <field>id</field>
            </prim-key-field>
            <jeus-query>
                <method>
                    <method-name>findByAddress</method-name>
                    <method-params>
                        <method-param>java.lang.String</method-param>
                    </method-params>
                </method>
                <sql>customer_address=?</sql>
            </jeus-query>
            <jeus-query>
                <query-method>
                    <method-name>findByTitle</method-name>
                    <method-params>
                        <method-param>java.lang.String</method-param>
                    </method-params>
                </query-method>
            <jeus-ql>
                SELECT OBJECT(b) FROM Book b
                WHERE b.title = ?1 ORDERBY b.price
            </jeus-ql>
        </jeus-query>
        <db-vendor>oracle</db-vendor>
    </jeus-bean>
</beanlist>

```

```

        <data-source-name>MYDB</data-source-name>
    </schema-info>
    . . .
</jeus-bean>
    . . .
</beanlist>
    . . .
</jeus-ejb-dd>

```

以下は、設定タグについての説明です。

- **<table-name>**

- 現在のEJBにマッピングされる関係データベース(RDB)の表の名前です。
- 設定されていない場合は、EJB module name、EJB nameを使用して表の名前が任意で設定されます。
- 一部のDBMSが持つ表名の長さに制限があるため、15字以内の表名のみ許可されます。

- **<cm-field>**

- Container Managed Fieldのマッピングは、それぞれの列とフィールドの関係のために存在します。
- 下位に以下のタグを設定します。

タグ	説明
<field>	DBの列にマッピングされるEJBフィールド名を指定します
<column-name>	指定されたDBの表名と同じものです。指定されていない場合は、EJBフィールド名を使用します
<type>	<p>DBの列に定義されたデータ型(例: VARCHAR(25)、NUMERICなど)を基準にします。タグが指定されていない場合は、デフォルト型が使用されます。(使用されるタイプはDBによって異なります。)このマッピングについては、「<a href="#">付録 A. 基本JavaタイプとDBフィールドのマッピング</a>」を参照してください。</p> <p>Oracle DBMSの場合は以下の値が使用できます。これらのタイプは、画像のような大きいデータ型に適しています</p> <ul style="list-style-type: none"> <li>– CLOB(Character Large Object) : java.lang.Stringに適してします</li> <li>– BLOB(Binary Large Object) : serializableされたフィールドに適してします</li> </ul>
<exclude-field>	ruelに設定すると、<field>タグで指定されたフィールドに対してaccessorメソッドが生成されず、クライアントから当該フィールドにアクセスできません。同オプショ

タグ	説明
	<p>ンは、CMP 2.0 Beanでのみ動作し、移行(マイグレーション)のために使用されます</p> <p>(デフォルト値: false)</p>

#### ● <creating-table>

- Beanが開始される時点でこのBeanが使用する表がDBに存在するの否かをチェックし、下位タグに従って既存の表を使用するか、新規作成、またはエラーを発生させます。
- チェックする方法は、<schema-info>/<table-name>と同じ名前で、<schema-info>/<cm-field>に明示したフィールドが存在するのチェックします。また、リレーションを管理するBeanの場合、<ejb-relation-map>/<jeus-relationship-role>/<column-map>に明示した外部キーの存在有無もチェックします。
- 以下の下位タグを設定します。

タグ	説明
<use-existing-table>	すでにDBに表が存在する場合はそのまま使用し、存在しない場合のみ生成します
<force-creating-table>	DBに表が存在してもその表を削除し、新しい表を生成します

下位タグを指定しない場合、すでに表が存在している場合はエラーが発生し、存在しない場合は表を生成します。

- タグが存在しない場合は、MS JVMパラメータの「-Djeus.ejb.checktable」設定によって表のチェック有無を決めます。

設定値	説明
true	表をチェックする際、該当する表が存在しないとエラーが発生します(デフォルト値)
false	Beanが開始する時点で表のチェックを実行しないため、実際の表を使用するときにエラーが発生します

#### ● <deleting-table>

- このタグが有効の場合、EJBモジュールがアンデプロイされるとき、指定された表がDBから削除されます。

- このオプションは、実際の状況において慎重に使われる必要があるため、JEUSのシステム・プロパティを使用して別途指定したときのみ動作します。すなわち、JEUSMain.xmlのCommandオプションに「-Djeus.ejb.enable.configDeleteOption=true」と設定した場合のみ動作します。

また、<creating-table>設定が存在しないのに<deleting-table>設定を使用することはほとんどないため、<creating-table>設定が存在しない場合は<deleting-table>設定が実行されません。

## ● <prim-key-field>

- 複合キーを主キーとして使用する場合、選択的に使われます。すなわち、ejb-jar.xmlの<prim-key-class>にユーザーが実装した主キー・クラスを指定した場合、複合キーが選択的に使用できます。基本的には、<prim-key-class>に宣言されたすべてのパブリック・フィールドが主キーを構成しますが、ユーザーがフィールドを選択して主キーを構成したい場合はこの項目を設定します。

- 以下の要素を正しく使用してください。

### • 複合キー

要素	説明
ejb-jar.xml/<prim-key-class>	ユーザーが実装した主キー・クラスです(必須項目)
ejb-jar.xml/<primkey-field>	宣言できません。<prim-key-class>に宣言したタイプは複数のフィールドを含むタイプなので、1つのフィールド名とマッピングできません(選択項目)
jeus-ejb-dd.xml/<prim-key-field>	主キークラスのすべてのパブリック・フィールドを主キーとして構成せず、必要なフィールドを選択して宣言します(選択項目)

### • 単一キー

要素	説明
ejb-jar.xml/<prim-key-class>	java.lang.String, java.lang.Integerなど、ejb-jar.xml/<primkey-field>のタイプです(必須項目)
ejb-jar.xml/<primkey-field>	主キーに該当するエンティティBeanのフィールド名です。  <primkey-field>に宣言したフィールド名と<prim-key-class>に宣言したタイプがエンティティBeanクラスに宣言されたフィールド名、タイプとマッピングされる必要があります(選択項目)
jeus-ejb-dd.xml/<prim-key-field>	複数のフィールドが存在する場合の選択時に必要な項目なので、宣言は不要です(選択項目)



- **<jeus-query>**

- CMP 1.1エンティティBeanの場合、finderメソッドに対して必要なSQL文を必ず明示します。
- CMP 2.0の場合には、ejb-jar.xmlに指定されたEJB QLがオーバーライディング(overriding)できます。  
  
<jeus-query>を設定すると、クエリー・メソッド(findXXX)でEJB QLとJEUS EJB QLの拡張が使用できます。これは、ejb-jar.xmlの<query>タグと類似しています。[「8.3.3.3. JEUS EJB QLの拡張機能\(Extension\)」](#)で詳しく説明します。同タグは、BEA WebLogicアプリケーション・サーバーをJEUS 7に容易にマイグレーションするのが目的です。
- SQLを定義するためにSQL文が使用されるfinderメソッドの名前、パラメータ、SQL文が設定される必要があります。これは、finderメソッドを生成するときに使われます。  
  
指定されたSQL文には、WHERE句の次に登場する部分のみ含まれます。他の要素は自動で与えられます。こちらでは「?」文字が指定可能で、これはfinderメソッドのパラメータ値の代わりになります。ANSI SQLのWHERE句の文法が使用できます。(このSQLはWHEREに含まれます。)
- <include-updates>が「true」に設定されると、finderメソッドを呼び出す間に変更された内容がDBにコミットされるため、finderメソッドで変更された内容を確認することができます。

- **<db-vendor>**

- Beanのために使用されるDBのベンダーを指定します。  
  
ベンダーを最適化するために、EJBエンジンがベンダー名を使用します。指定された値が実際の性能にどれほどの影響を及ぼすかはまだ確認されていません。使用可能なベンダーの種類は、[「付録 A. 基本 JavaタイプとDBフィールドのマッピング」](#)を参照してください。一般的な例として「oracle」が挙げられます。
- またこの値は、Javaオブジェクトとベンダーが指定したDBフィールド型との正確なマッピングを判断するために使用されます。

- **<data-source-name>**

- DBと接続するときに使用するDBコネクション・プールのJNDI名を指定します。コネクション・プールは、一般的にドメインに設定され、MS JVMIによって実行されます。

- **<auto-key-generator>**

- CMP 2.0で自動的に主キーを生成するために使用しており、下位タグが存在します。下位タグについての詳細説明は、[「8.2.3.3. 主キー\(Primary Key\)の自動生成」](#)を参照してください。

### 8.3.3. CMP 2.0

本節では、CMP 2.0の設定についての説明です。

#### 8.3.3.1. リレーションシップ・マッピングの設定

2つのBeanの間にリレーションシップ・マッピングを設定する方法は、以下のとおり場合によって異なります。

- One-to-one/One-to-many Relationship Mapping
- Many-to-many Relationship Mapping

#### One-to-one/One-to-manyリレーションシップ・マッピング

2つのBeanの間にOne-to-one/One-to-manyリレーションシップを設定するときは<ejb-relation-map>タグ内に設定します。

以下は、「employee-manager」という関係を持つ2つのBean同士にMany-to-one(= One-to-many)関係を設定した例です。<ejb-relation-map>タグに設定します。

**[例 8.9] One-to-one/One-to-manyリレーションシップの設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
    . . .
    <ejb-relation-map>
        <relation-name>employee-manager</relation-name>
        <jeus-relationship-role>
            <relationship-role-name>employee-to-manager</relationship-role-name>
            <column-map>
                <foreign-key-column>manager-id</foreign-key-column>
                <target-primary-key-column>man-id</target-primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </ejb-relation-map>
    . . .
</jeus-ejb-dd>
```

以下は、設定タグについての説明です。

- <relation-name>

ejb-jar.xmlファイルに定義されています。

- <jeus-relationship-role>

JEUSリレーションシップの2つの役割をリレーションシップの目的別に1つずつ設定します。各リレーションシップの役割に必要な情報は以下のとおりです。

– **<relationship-role-name>**

ejb-jar.xmlの<ejb-relationship-role-name>タグに定義されています。

– **<column-map>**

2つの下位タグである<foreign-key-column>と<target-primary-key-column>を有します。

タグ	説明
<foreign-key-column>	リレーションシップで外部キーの役割をする列名です
<target-primary-key-column>	外部キーの列がマッピングされる表の主キー・フィールドです。キーが複数の列で構成されている場合、複数の列対列のマッピングが指定できます

上記のXMLの例では、employee表のmanager-idという外部キーの列がemployee-to-managerのMany-to-one関係を確立するため、manager表の主キー・フィールドとマッピングされています。

One-to-oneマッピングもMany-to-oneと類似した方法で設定されます。ただし、One-to-one関係では主キーの列と外部キーの列が表のどこに存在しても構いません。また、One-to-manyでは「many」側の表が外部キーの列を持つ必要があります。

## Many-to-manyリレーションシップ・マッピング

2つのBean同士にMany-to-many関係を確立するためには、One-to-oneの関係で設定された情報以外に、<table-name>と<jeus-relationship-role>の情報が追加的に設定される必要があります。

以下は、Many-to-many関係を示す「student-course」の例です。

### [例 8.10] Many-to-manyリレーションシップ・マッピングの設定 : <<jeus-ejb-dd.xml>>

```
<jeus-ejb-dd>
    . . .
    <ejb-relation-map>
        <relation-name>student-course</relation-name>
        <table-name>STUDENTCOURSEJOIN</table-name>
        <jeus-relationship-role>
            <relationship-role-name>student</relationship-role-name>
            <column-map>
                <foreign-key-column>
                    student-id <!--This column in join table-->
                </foreign-key-column>
                <target-primary-key-column>
                    stu-id <!--This column in student table-->
                </target-primary-key-column>
            </column-map>
        </jeus-relationship-role>
    </jeus-relationship-role>
```

```

        <relationship-role-name>course</relationship-role-name>
        <column-map>
            <foreign-key-column>
                course-id <!--This column in join table-->
            </foreign-key-column>
            <target-primary-key-column>
                cou-id <!--This column in course table-->
            </target-primary-key-column>
        </column-map>
    </jeus-relationship-role>
</ejb-relation-map>
. . .
</jeus-ejb-dd>

```

以下は、設定タグについての説明です。

タグ	説明
<table-name>	Many-to-many関係の結合表の名前を指定します
<jeus-relationship-role>	2つのJEUSリレーションシップ・ロールで、2つのBeanが使用する表の主キー列に結合表の2つの外部キー列をマッピングします。

上記の例で、2つのJEUSリレーションシップ・ロールがどのように使用されているのかが確認できます。各ロールは結合表の2つの列のうち1つと、実際にBeanが使用する表の主キーの列がマッピングされるように宣言しています。

### 8.3.3.2. インスタントEJB QLの設定

クライアントにインスタントEJB QLの問合せを可能にするには、JEUS EJB DDの<enable-instant-ql>タグをtrueに設定します。詳細内容については、「[付録 B. Instant EJB QL APIリファレンス](#)」を参照してください。

**[例 8.11] インスタントEJB QLの設定 : <<jeus-ejb-dd.xml>>**

```

<jeus-ejb-dd>
. . .
<beanlist>
. . .
    <jeus-bean>
. . .
        <enable-instant-ql>true</enable-instant-ql>
    </jeus-bean>
. . .
</beanlist>
. . .
</jeus-ejb-dd>

```

### 8.3.3.3. JEUS EJB QLの拡張機能(Extension)

JEUS CMPに関連するEJB QLの拡張機能について説明します。

JEUS EJB QLの拡張機能には、新しいキーワードと副問合せ(subquery)、ejbSelect()メソッドによってリターンされる結果セット、集合関数、多様なEJB QLの問合せ、GROUP BYキーワードなどがあります。

CMP 2.0 BeanのfindByXXX()メソッドとejbSelectXXX()メソッドでは、そのホーム・インターフェースとejb-jar.xmlのEJB QL文が連携されている必要があります。JEUSは、標準EJB QL言語にいくつかの事項を追加します。この追加事項は、ejb-jar.xmlファイルやEJB QLで使用されます。同機能が使用されたejb-jar.xmlの例は、[「8.5. CMP 2.0 エンティティBeanの完全な例」](#)を参照してください。

---

#### 注

本書で適用されるEJB QLの拡張機能は、標準EJB 2.1 QLでサポートしていない項目です。そのため、同拡張機能を使用して作成されたEJBアプリケーションはJava EE標準に準拠していないため、移植性が低下します。すなわち、提供された拡張機能を使って作成したEJBコンポーネントは、別のJava EEサーバーにデプロイできません。JEUS EJB QLの拡張機能は、ほとんど標準SQLのように動作します。より詳しい情報については、SQLを参照してください。

---

### JEUS EJB QL拡張機能のキーワード追加

JEUSでは、以下のような非標準キーワードがEJB QL文で使用できます。

- ORACLEHINTキーワード

「oraclehint<標準Oracle hint string>」の文法を有します。

```
SELECT OBJECT(b) ORACLEHINT ' /*+ALL_ROWS*/ '
FROM Book b
```

このキーワードについての詳細情報は、Oracleの関連文書を参照してください。

### JEUS EJB QL拡張機能の副問合せ

JEUSのEJB QL副問合せとは、EJB QL問合せの1つとして、WHERE句に含まれた別の問合せのことをいいます。SQL問合せと副問合せの関係と似ています。

以下は、ネストされたEJB QL問合せの例です。平均以上の給料をもらっている社員を問合せします。

```
SELECT OBJECT(e)
FROM EmployeeBean AS e
WHERE e.salary >
      (SELECT AVG(e2.salary)
       FROM EmployeeBean AS e2)
```

- リターン・タイプ

JEUS EJB QL問合せに対し、以下のようなリターン・タイプがあります。

- Single value、もしくは<cmp-field>に対応するコレクション(上記の例において、e.salaryのような値)
- 集合関数を使用して作成された1つの戻り値(例: MAX(e.salary))
- 簡単な主キー(Compound主キーではない)を含むcmp-Bean(例: SELECT OBJECT(emp))

- 比較演算子

JEUS EJB QLは、副問合せを比較演算子の被演算子として使用可能です。JEUS EJB QLの比較演算子は以下のとおりです。

- [NOT] IN, [NOT] EXISTS, [NOT] UNIQUE )
- <, >, <=, >=, =, <>
- 上記の演算子とANY、ALL、SOMEの組合せ

以下は、マネージャーでない社員を抽出する例です。

```
SELECT OBJECT(employee)
FROM EmployeeBean AS employee
WHERE employee.id NOT IN
  (SELECT employee2.id
   FROM ManagerBean AS employee2)
```

以下は、副問合せの結果値の有無をテストする**NOT EXISTS**演算子の例です。(「NOT」は値の有無をテストするために削除される場合があります。)

```
SELECT OBJECT(cust)
FROM CustomerBean AS cust
WHERE NOT EXISTS
  (SELECT order.cust_num
   FROM OrderBean AS order
   WHERE cust.num = order.cust_num)
```

上記の問合せの結果は、注文していないすべての顧客一覧です。

上記の例は、副問合せの結果が上位問合せのパラメータとして反映される必要があるため、相関(correlated)問合せともいえます。一方、依存関係のない問合せを非相関(uncorrelated)問合せといいます。

以下は、**UNIQUE**演算子の使用例です。この演算子は、結果セットの行のうち重複されるものがあるか否かを確認します。行が0、あるいは1つであるか、副問合せの結果内に存在するすべてのレコードが一意的な場合は、trueを返します。同演算子にNOTを付けると、反対の結果が得られます。

```
SELECT OBJECT(cust)
FROM CustomerBean AS cust
WHERE UNIQUE
      (SELECT cust2.id FROM CustomerBean AS cust2)
```

以下は、「>」演算子の例です。他の演算子と同様、副問合せの結果値として使用できます。

```
SELECT OBJECT(employee)
FROM EmployeeBean AS employee
WHERE employee.salary >
      (SELECT AVG(employee2.salary)
       FROM EmployeeBean AS employee2)
```

上記のJEUS EJB QL問合せは、平均給料以上をもらっている社員の結果を返します。

参考までに、ANY、ALL、SOMEが使用されない場合は、1つの値のみ副問合せとしてリターンされる必要があります。(この場合は、平均給料) ANY、ALL、SOME演算子は、1つの値より多くの値をリターンする副問合せに使用します。同演算作業は、以下のように解釈できます。

- <Operand> <arithmetic operator> **ANY** <subquery>

副問合せの結果のうち、少なくとも1つの値がtrueをリターンすると、trueを返します。

- <Operand> <arithmetic operator> **SOME** <subquery>

「ANY」と同様な結果が得られます。

- <Operand> <arithmetic operator> **ALL** <subquery>

副問合せの結果のすべてがtrueをリターンした場合のみtrueを返します。

---

#### 参考

XML DDファイルに「>」と「<」文字を入力する場合は、CDATA部分に含ませます。そうしないと、XMLパーサーがその文字をXMLタグの指定文字として認識します。

---

#### [例 8.12] XML DDファイルに「>」と「<」文字を挿入 : <<ejb-jar.xml>>

```
. . .
<query>
  <description>method finds large orders</description>
```

```

<query-method>
    <method-name>findLargeOrders</method-name>
    <method-params></method-params>
</query-method>
<ejb-ql>
<![CDATA[SELECT OBJECT(o) FROM Order o WHERE o.amount > 1000]]>
</ejb-ql>
</query>
. . .

```

## JEUS EJB QL拡張機能の結果セット

JEUS EJBエンジンは、複数の列であるjava.sql.ResultSetオブジェクトを返すejbSelect()問合せをサポートします。したがって、ejbSelectメソッドのSELECT文の中にコンマ(,)で区切られたターゲット・フィールドの一覧を指定することができます。

```

SELECT employee.name, employee.id, employee.salary
FROM EmployeeBean AS employee

```

上記の問合せは、すべてのEmployeeBeanインスタンスのname、id、salaryで構成されたjava.sql.ResultSetを生成します。

この結果セットは、「name」、「id」、「salary」の列で構成されたレコードが含まれます。同結果セットの各レコードに該当するエンティティBeanインスタンスを表現します。結果セットは、<cmp-field>値(Beansまたはリレーションシップ・フィールドでない)のみを返します。

---

### 参考

ejbSelect()メソッドを使用してjava.sql.ResultSetタイプのオブジェクトを取得し、すべての作業が終了したときにResultSet.close()を呼び出します。このclose()メソッドは、EJBエンジンによって自動的に呼び出されません。

---

## JEUS EJB QL拡張機能の動的問合せ

JEUS EJB QLは、EJB QL問合せをプログラマ的に実装できるようにサポートします。これは、インスタントEJB QL機能によって可能になります。本節では、JEUS CMP 2.0でインスタントEJB QLをプログラミングするときに必要な基本的な事項について説明します。

以下のコードは、CMP 2.0ホーム・インターフェースのインスタントEJB QLの使用方法です。同コードは、純粋にクライアント側のコードです。このコードを動作させるには、CMP BeanにインスタントEJB QLを使用するように設定されている必要があります。詳細内容については、「[8.3. エンティティEJBの設定](#)」を参照してください。



```

import jeus.ejb.Bean.objectbase.EJBInstanceFinder;
import java.util.Collection;
. . .
Context initial = new InitialContext();
Object objref = initial.lookup("emp");
EmpHome home = (EmpHome)
    PortableRemoteObject.narrow(objref, EmpHome.class);
EJBInstanceFinder finder = (EJBInstanceFinder) home;
java.util.Collection foundBeans =
    finder.findWithInstantQL ("<EJB QL query sentence>");
// Use foundBeans collection...
. . .

```

上記例の「<EJB QL query sentence>」は、パラメータ無しの完全なEJB QL文で代替される必要があります。(「?」は使用できません。) このAPIについては、「[付録 B. Instant EJB QL APIリファレンス](#)」を参照してください。

## JEUS EJB QL拡張機能のGROUP BYキーワード

JEUS EJB QLは、**GROUP BY**と**GROUP BY HAVING**句を提供しており、以下のように動作します。

- **<SELECT statement> GROUP BY <grouping column(s)>**

SELECT文が実行されるとレコードが生成され、GROUP BY文は<grouping column(s)>に指定された列のセットを検索します。同じ値を持つ列内のすべてのレコードは1つのレコードに結合され、重複されるレコードは捨てられます。

```

SELECT employee.departmentName
FROM EmployeeBean AS employee
GROUP BY employee.departmentName

```

上記の問合せは、employee Beanの一意のdepartmentName一覧が結果値として得られます。

- **<SELECT statement> GROUP BY <grouping column(s)> HAVING <condition>**

同SELECT文は、上記例のような方式で動作しますが、<condition>に設定された事項を満足させる列のグループのみ含まれます。

```

SELECT employee.departmentName
FROM EmployeeBean AS employee
GROUP BY employee.departmentName
HAVING COUNT(employee.departmentName) >= 3

```

上記の問合せは、2つ以上のEmployeeBeanインスタンスが発見された一意のdepartmentName一覧を返します。(すなわち、3つ以上のemployeeを持つdepartmentの一覧)

たとえば、[表 8.3](#)のデータにEJB QL問合せを実行させると、[表 8.4](#)のような結果が出力されます。

**[表 8.3] EmployeeBeanインスタンスのためのEJBフィールド**

id	departmentName
johnsmith	R&D
peterwright	R&D
lydiajobs	R&D
catherinepeters	Marketing

**[表 8.4] 結果セット**

departmentName
R&D

この結果が出力される理由は、HAVING条件が少なくとも3つのレコードを有する列のグループが含まれる必要があると明示されているためです。GROUP BY、GROUP BY HAVING句の詳細な情報については、SQL文を参照してください。

#### 参考

JEUS EJB QLでは、GROUP BYとGROUP BY HAVING文を副問合せ、またはルートの間問合せで使用できます。GROUP BY HAVING文場合は、java.lang.Stringオブジェクトの集合がリターンされます。ルートのGROUP BYとGROUP BY HAVING文は、Beanオブジェクトがリターンできません。

## 8.3.4. DB Insert Delayの設定(CMP Only)

CMPでEJBが生成されるとき、EJBデータがバックエンドDBに記録される時点を設定できます。

メソッド	説明
ejbCreate	ejbCreate()の実行後、ejbPostCreate()を実行する前に新しいEJBデータを挿入します
ejbPostCreate	ejbCreate()とejbPostCreate()の完成後、新しいEJBデータを挿入します(デフォルト値)

DB Insert Delayの設定は、jeus-ejb-dd.xmlで<jeus-bean>の<database-insert-delay>タブ内に構成されます。

**[例 8.13] DB Insert Delayの設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
. . .
<beanlist>
. . .
  <jeus-bean>
. . .
```

```

        <database-insert-delay>ejbCreate</database-insert-delay>
        . . .
    </jeus-bean>
    . . .
</beanlist>
. . .
</jeus-ejb-dd>

```

# 8.4. エンティティEJBのチューニング

本節では、JEUSのエンティティEJBの性能をチューニング方法について詳しく説明します。チューニング方法は、「第4章 EJBの共通特性」と「第7章 セッションBean」の「オブジェクト・マネージメント」で記述した方法と共にエンティティBeanに適用できます。

## 8.4.1. 共通

以下は、すべてのエンティティBeanの性能をチューニングするために必要な共通設定です。

- エンジン・メイン・モードの選択とクラスタリングの適用可否
- エンティティ・キャッシュ・サイズの設定 -DBベンダーの設定

以下は、すべてのエンティティBeanの性能をチューニングするための説明です。

## エンジン・メイン・モードの選択とクラスタリングの適用有無

以下は、<engine-type>の選択時と、いくつかのEJBエンジンにEJBをクラスタリングするかを指定するときの判断基準となる規則です。

[表 8.5] エンティティBeanのエンジン・タイプの選択とクラスタリングの使用有無

1Beanあたりの要求頻度	多くのEJBエンジン、多数のCPU	少ないEJBエンジン、1つ以上のCPU	1つのEJBエンジン、1つのCPU
高	EJBをクラスタリングし、SINGLE_OBJECTを使用します	EJBをクラスタリングし、MULTIPLE_OBJECTを使用します	より多くのEJBエンジンを使用するようにします。あるいは、クラスタリングせず、MULTIPLE_OBJECTを使用します
中	EJBをクラスタリングし、SINGLE_OBJECTを使用します	EJBをクラスタリングし、MULTIPLE_OBJECTを使用します	EJBをクラスタリングせず、EXCLUSIVE_ACCESSを使用します

1Beanあたり の要求頻度	多くのEJBエンジン、多数の CPU	少ないEJBエンジン、1つ以上 のCPU	1つのEJBエンジン、1つのCPU
低	EJBをクラスタリングせず、 EXCLUSIVE_ACCESSを使用します	EJBをクラスタリングせず、 EXCLUSIVE_ACCESSを使用 します	EJBをクラスタリングせず、 EXCLUSIVE_ACCESSを使用 します

エンティティBeanのDBデータが外部のWASではなく、コンポーネントによって変更される環境においては、SINGLE\_OBJECTまたはMULTIPLE\_OBJECTを使用します。なお、フェイルオーバーのためにはBeanをクラスタリングする必要があります。

SINGLE\_OBJECTとMULTIPLE\_OBJECTモードのうち1つを選択するとき、最後に考慮する事項は対象Beanのトランザクションの処理時間です。(すなわち、Bean内部でトランザクションが開始、コミットされ、終了するまでの平均時間) 同時時間が長い場合はMULTIPLE\_OBJECTモードを選択し、このモードで使用可能な同時性を使用する必要があります。一方、時間が短い場合は、SINGLE\_OBJECTモードの使用をお勧めします。

上記の「規則」に従うと、EJBエンジンは最適の性能でejbLoad()を呼び出すことができます。

---

#### 参考

上記表の「高」、「中」、「低」の意味を明確に定義することはできないため、使用環境における最適の設定を工夫する必要があります。

---

## エンティティ・キャッシュ・サイズの設定

エンティティ・キャッシュは、パッシベーションされるエンティティBeanインスタンスのリポジトリの役割をします。エンティティBeanインスタンスが再アクティブ化されるとキャッシュから使用されます。

<entity-cache-size>サイズを大きく(数百個のインスタンスが格納できるほど)設定すると、パッシベーションされるBeanが再アクティブ化されるとき、新しいエンティティBeanインスタンスの生成は不要なので性能が向上します。しかし、メモリー管理が最も重要な事項である場合は、この値を小さく設定するか、使用しないようにします。(値が「0」の場合は、エンティティ・キャッシュを使用しません。)

## DBベンダーの設定

<db-vendor>タグを正しく設定します。場合によっては、Oracleのような他ベンダー製品のために、EJBエンジンがDBとの接続を最適化させます。このタグに使用できるDBMSベンダー名については、『JEUS XMLリファレンスガイド』の「第11章 domain.xml EJBエンジンの設定」を参照してください。

### 8.4.2. BMP & CMP 1.1

以下は、BMPとCMP 1.1のチューニングについての説明です。

## Non-modifyingメソッドの登録

DBに表現された状態情報を変更しないエンティティBeanのすべての業務メソッドは、JEUS EJBモジュールDDの<non-modifying-method>タグに登録される必要があります。これにより、EJBエンジンがejbStore()を呼び出す必要がなくなり、効率的に運用することができます。同チューニングは、BMPとCMP 1.1 Beanにのみ適用できます。

### 8.4.3. CMP 1.1/2.0

以下は、CMP 1.1/2.0のチューニングについての説明です。

## エンジンのサブ・モードの選択

CMP Beanの使用時に<subengine-type>を選択すると、EJBエンジンがejbLoad()、ejbFind()メソッドを最適化して使用することができます。<subengine-type>を選択する主な基準は以下のとおりです。

- CMP Beanが、書き込み作業より読み込み作業が多いと予想される場合は、ReadLockingモードを使用します。
- CMP Beanが、読み込み作業より書き込み作業が多いと予想される場合は、WriteLockingまたはWriteLockingFindモードを使用します。

## フェッチ・サイズの設定

多くのシステム・メモリーを使用しても高性能が出したい場合は、CMP Beanの<fetch-size>タグを高く設定します。そうすると、呼び出しごとに多くのデータを読み込むため、ネットワークを効果的に使用することができます。しかし、EJBエンジンが多くのデータをキャッシングしなければならないため、システムのリソースを多く消耗することになり、各呼び出しに対する待機時間が増加します。

一方、システム・メモリーの節約を優先して、遅いネットワーク速度でも構わない場合は、この値を低く設定します。デフォルト値は「10」です。(通常、高い値は「100」、低い値は「10」以下です。)

## 初期化キャッシングの設定

高い運用性能のために多くのシステム・メモリーが使用可能な場合は、CMP Beanの<init-caching>値をtrueに設定します。そうすると、EJBエンジンの起動時にすべてのDBレコードが読み込まれ、エンティティBeanインスタンスに渡されます。結果的にエンジンの起動時間は長くなりますが、最初のアクセス時間は短縮できます。

一方、システム・メモリーの節約またはエンジンの起動時間を優先する場合は、falseに設定します。DB表がEJBに多くマッピングされている場合は(数百、数千レコード)、<init-caching>がfalseに設定される必要があります。

ります。EJBがread-only(ejbLoad())が一度だけ呼び出される場合)の場合は、同オプションの使用をお勧めします。

#### 8.4.4. CMP 2.0

CMP 2.0をチューニングするためにEJB QLを使用しますが、EJB QLは効率的ではないので簡単な場合のみ使用してください。

### 8.5. CMP 2.0 エンティティBeanの完全な例

EJBコード、標準EJB DD、JEUS DDが同時に設定、動作できるかを示すため、本節ではCMP 2.0 Beanの完全な例を提示します。同例は、Bookを概念としてモデル化したBook EJBです。

すべての例にはアノテーションを付けておらず、CMP 2.0 Beanがjeus-ejb-dd.xmlに設定される方法のみ記述します。例をデプロイするためには、パッケージングの後、EJBにデプロイします。詳細内容については、「[第3章 EJBモジュール](#)」を参照してください。

JEUSのejb-jar.xmlでのEJB QLの使用例は、本節の[8.5節「Java EE EJB DD」](#)を参照してください。

## リモート・インターフェース

**[例 8.14] リモート・インターフェース : <<Book.java>>**

```
package test.book;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Book extends EJBObject {

    public String getTitle() throws RemoteException;
    public void setTitle(String title) throws RemoteException;
    public String getAuthor() throws RemoteException;
    public void setAuthor(String author) throws RemoteException;
    public double getPrice() throws RemoteException;
    public void setPrice(double price) throws RemoteException;
    public String getPublisher() throws RemoteException;
    public void setPublisher(String publisher)
        throws RemoteException;
    public String toBookString() throws RemoteException;
}
```

## ホーム・インターフェース

[例 8.15] ホーム・インターフェース : <<BookHome.java>>

```
package test.book;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import javax.ejb.EJBHome;
import java.util.*;

public interface BookHome extends EJBHome {

    public Book create(String code, String title, String author,
                      double price, String publisher)
        throws CreateException, RemoteException;
    public Book findByPrimaryKey(String code)
        throws FinderException, RemoteException;
    public Collection findByTitle(String title)
        throws FinderException, RemoteException;
    public Collection findInRange(String from, String to)
        throws FinderException, RemoteException;
    public Collection findAll()
        throws FinderException, RemoteException;
}
```

## Beanの実装

[例 8.16] Beanの実装 : <<BookEJB.java>>

```
package test.book;

import javax.ejb.EntityBean;
import javax.ejb.EntityContext;

public abstract class BookEJB implements EntityBean {

    public String ejbCreate(String code, String title,
                           String author, double price, String publisher) {
        setCode(code);
        setTitle(title);
        setAuthor(author);
        setPrice(price);
        setPublisher(publisher);
        return null;
    }
}
```

```

    }

    public void ejbPostCreate(String code, String title,
        String author, double price, String publisher) {}

    public abstract String getCode();
    public abstract void setCode(String code);
    public abstract String getTitle();
    public abstract void setTitle(String title);
    public abstract String getAuthor();
    public abstract void setAuthor(String author);
    public abstract double getPrice();
    public abstract void setPrice(double price);
    public abstract String getPublisher();
    public abstract void setPublisher(String publisher);

    public String toBookString() {
        return getCode() + "- [" + getTitle() + "] by " +
            getAuthor() + " | " + getPrice() + " | " +
            getPublisher();
    }

    public BookEJB() {}
    public void setEntityContext(EntityContext ctx) {}
    public void unsetEntityContext() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
}

```

## Java EE EJB DD

### [例 8.17] Java EE EJB DD : <<ejb-jar.xml>>

```

<?xml version="1.0"?>
<ejb-jar version="3.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd">
    <enterprise-Beans>
        <entity>
            <ejb-name>BookBean</ejb-name>
            <home>test.book.BookHome</home>

```



```

<remote>test.book.Book</remote>
<ejb-class>test.book.BookEJB</ejb-class>
<persistence-type>Container</persistence-type>
<prim-key-class>java.lang.String</prim-key-class>
<reentrant>false</reentrant>
<cmp-version>2.x</cmp-version>
<abstract-schema-name>Book</abstract-schema-name>
<cmp-field><field-name>code</field-name></cmp-field>
<cmp-field><field-name>title</field-name></cmp-field>
<cmp-field><field-name>author</field-name></cmp-field>
<cmp-field><field-name>price</field-name></cmp-field>
<cmp-field><field-name>publisher</field-name></cmp-field>
<primkey-field>code</primkey-field>
  <query>
    <query-method>
      <method-name>findByTitle</method-name>
      <method-params>
        <method-param>
          java.lang.String
        </method-param>
      </method-params>
    </query-method>
  <ejb-ql>
    SELECT OBJECT(b) FROM Book b
    WHERE b.title = ?1 ORDERBY b.price
  </ejb-ql>
</query>
  <query>
    <query-method>
      <method-name>findInRange</method-name>
      <method-params>
        <method-param>
          java.lang.String
        </method-param>
        <method-param>
          java.lang.String
        </method-param>
      </method-params>
    </query-method>
  <ejb-ql>
    SELECT OBJECT(b) FROM Book b
    WHERE b.title BETWEEN ?1 AND ?2
  </ejb-ql>
</query>
  <query>
    <query-method>
      <method-name>findAll</method-name>

```

```

        <method-params/>
    </query-method>
    <ejb-ql>
        SELECT OBJECT(b) ORACLEHINT '/*+ALL_ROWS*/' FROM Book b
    </ejb-ql>
</query>
</entity>
</enterprise-Beans>
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>BookBean</ejb-name>
            <method-name>*</method-name>
            <method-params/>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
</ejb-jar>

```

---

### 参考

太字で表示されている部分が非標準のJEUS専用EJB QLです。(ORDERBY、BETWEEN、ORACLEHINT)

---

## JEUS EJB DD

### [例 8.18] JEUS EJB DD : <<jeus-ejb-dd.xml>>

```

<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">

    <module-info/>
    <beanlist>
        <jeus-bean>
            <ejb-name>BookBean</ejb-name>
            <export-name>book</export-name>
            <export-port>0</export-port>
            <export-iiop><only-iiop>false</only-iiop></export-iiop>
            <persistence-optimize>
                <engine-type>EXCLUSIVE_ACCESS</engine-type>
                <non-modifying-method>
                    <method-name>getTitle</method-name>
                </non-modifying-method>
                <non-modifying-method>
                    <method-name>getAuthor</method-name>
                </non-modifying-method>
            </persistence-optimize>
        </jeus-bean>
    </beanlist>
</jeus-ejb-dd>

```

```

        </non-modifying-method>
    <non-modifying-method>
        <method-name>getPrice</method-name>
    </non-modifying-method>
    <non-modifying-method>
        <method-name>getPublisher</method-name>
    </non-modifying-method>
    <non-modifying-method>
        <method-name>toBookString</method-name>
    </non-modifying-method>
</persistence-optimize>
<schema-info>
    <table-name>Booktable</table-name>
    <cm-field><field>code</field></cm-field>
    <cm-field><field>title</field></cm-field>
    <cm-field><field>author</field></cm-field>
    <cm-field><field>price</field></cm-field>
    <cm-field><field>publisher</field></cm-field>
    <creating-table>
        <use-existing-table/>
    </creating-table>
    <deleting-table>true</deleting-table>
    <db-vendor>oracle</db-vendor>
    <data-source-name>datasource1</data-source-name>
</schema-info>
    <enable-instant-ql>true</enable-instant-ql>
</jeus-bean>
</beanlist>
</jeus-ejb-dd>

```



# 第9章 メッセージ駆動型Bean(MDB)

本章では、EJBエンジンでメッセージ駆動型Bean(以下、MDB(Message Driven Bean))を使用するときの注意事項について説明します。

## 9.1. 概要

EJBエンジンは、MDBインスタンスを同時に実行することが可能であり、メッセージ・ストリームを処理することができます。EJBエンジンは、MDBクラスに受信されるメッセージの順序を保証しないため、メッセージの到着順ではなく任意で送信します。

MDBも処理順を保証しておらず、予約するメッセージより取り消しメッセージが先に到着する場合があります。そのため、設計の際には処理順を考慮する必要があります。

このような同時処理は、BeanプールがMDBからインスタンスを取得し、リクエストに割り当てることで処理されます。これはステートレス・セッションBeanのThread Ticket Pool(TTP)の動作方法とほぼ同様です。したがって、MDBでは<thread-max>値ではなく、<pool-max>値を使用する必要があります。詳細内容については、「[第7章 セッションBean](#)」を参照してください。

## 9.2. MDBの設定

EJBエンジンがMDBを実行するためには若干の設定が必要です。MDBは固有の設定以外にJEUSの別のEJBと共通した設定があります。

本節では、以下の項目について簡単に説明します。

- JEUS EJB DDの基本MDBの設定([「第4章 EJBの共通特性」](#)で説明した設定の下位項目)
- JEUS EJB DDファイルに設定されているJEUS MDBのJMS専用の設定

### 9.2.1. 基本環境の設定

MDBは、別のEJBと類似した基本設定を使用します。各MDBには<ejb-name>、<run-as identity>、<security-interop>などが設定できます。

以下は、MDBの基本設定例です。

**[例 9.1] 基本環境設定 : <<jeus-ejb-dd.xml>>**

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            <ejb-name>order</ejb-name>
            <!--JMS settings goes here (see 「9.2.2. JMSの設定」 )-->
            <run-as-identity>
                . . .
            </run-as-identity>
            <security-interop>
                . . .
            </security-interop>
            <env>
                . . .
            </env>
            <ejb-ref>
                . . .
            </ejb-ref>
            <res-ref>
                . . .
            </res-ref>
            <res-env-ref>
                . . .
            </res-env-ref>
            <!-- No clustering of MDB -->
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

## 9.2.2. JMSの設定

MDBが処理するメッセージによって設定が異なります。JMSメッセージを処理する場合は、JMSコネクション・ファクトリーのエクスポート名が必要であり、コネクタ経由でメッセージを処理する場合は、リソース・アダプターが必要です。

以下は、MDBクラスのアノテーションとして設定した例と、それに対応されるXML文の一部です。

**[例 9.2] JMSの設定 : <<MyMDB.class>>**

```
@MessageDriven(
    activationConfig = {
```

```

        @ActivationConfigProperty(propertyName="destinationType",
                                   propertyValue="javax.jms.Queue"),
        @ActivationConfigProperty(propertyName="acknowledgeMode",
                                   propertyValue="Auto-acknowledge")
    },
    mappedName = "jms/QUEUE1"
)
public class MyMDB implements MessageListener {

    ...

}

```

**[例 9.3] JMSの設定 : <<jeus-ejb-dd.xml>>**

```

<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <ejb-name>MyMDB</ejb-name>
            <connection-factory-name>
                QueueConnectionFactory
            </connection-factory-name>
            <destination>jms/QUEUE1</destination>
            <max-message>15</max-message>
            <activation-config>
                <activation-config-property>
                    <activation-config-property-name>
                        destinationType
                    </activation-config-property-name>
                    <activation-config-property-value>
                        javax.jms.Queue
                    </activation-config-property-value>
                </activation-config-property>
                <activation-config-property>
                    <activation-config-property-name>
                        acknowledgeMode
                    </activation-config-property-name>
                    <activation-config-property-value>
                        Auto-acknowledge
                    </activation-config-property-value>
                </activation-config-property>
            </activation-config>
            . . .
        </jeus-bean>
    
```

```
        . . .  
    </beanlist>  
    . . .  
</jeus-ejb-dd>
```

以下は、設定タグについての説明です。

- **<connection-factory-name>**

- JMSコネクション・ファクトリーが使用するJNDIエクスポート名を設定します。
- 様々な場所で設定可能であり、優先順位は以下のとおりです。
  1. jeus-ejb-dd.xml/<jeus-bean>/<connection-factory-name>
  2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : jeus.connectionFactoryName
  3. @ActivationConfigProperty property name : jeus.connectionFactoryName

- **<mdb-resource-adapter>**

- JEUSコネクタを介してメッセージング・システムと接続される場合に使用するリソース・アダプターを設定します。当該リソース・アダプターのjeus-connector-dd.xmlに指定されているモジュールのIDを設定して使用します。

- **<destination>**

- JMSデスティネーションのJNDI名を設定します。
- 様々な場所で設定可能であり、優先順位は以下のとおりです。
  1. jeus-ejb-dd.xml/<jeus-bean>/<destination>
  2. ejb-jar.xmlの<message-driven>/<mapped-name>
  3. @MessageDrivenのmappedName
  4. ejb-jar.xmlの<message-driven>/<message-destination-link>

- **<jndi-spi>**

- MDBがデフォルト値(jeus.jndi.JNSContextFactory)ではなく、別のJNDI名サービスに登録されているJMSサービスを使用する場合、すなわち、JEUS MDBをIBM MQまたはSONIC MQのようなJEUS JMSサービス以外のものと接続するときに使います。



- **<max-messages>**

- 複数のJMSセッションを登録して処理する場合、1つのセッションにメッセージを割り当てる最大数を指定するときに使用します。
- キューに蓄積されている数が設定値より少ない場合、1つのセッションが継続して処理し、値を超える場合は別のセッションを使用します。すなわち、この値が10の場合、メッセージ数が10個を超えるまでは1つのセッションがメッセージを処理します。これはロードを減らすためです。詳細内容については、JMS仕様を参照してください。

- **<activation-config>**

- JMSやリソース・アダプターを設定するactivation configで、ejb-jar.xmlのactivation configをオーバーライドする場合に使用します。
- JMS MDBの場合、acknowledgeMode、messageSelector、destinationType、subscriptionDurabilityは基本的に認識されます。追加的に上述したjeus.connectionFactoryNameとjeus.clientId、jeus.subscriptionNameが使用できます。jeus.clientIdとjeus.subscriptionNameは、TopicのDurable Subscriptionのために提供されます。値を設定しない場合は、モジュール名、Bean名で設定されます。
- 様々な場所で設定可能であり、優先順位は以下のとおりです。
  1. jeus-ejb-dd.xmlの<activation-config>
  2. ejb-jar.xmlの<activation-config>
  3. @ActivationConfig

- **<ack-mode>**

- JMSセッションのAcknowledgeモードを設定します。
- 様々な場所で設定可能であり、優先順位は以下のとおりです。
  1. jeus-ejb-dd.xml/<jeus-bean>/<ack-mode>
  2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : acknowledgeMode
  3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : acknowledgeMode
  4. ejb-jar.xml/<message-driven-bean>/<acknowledge-mode>
  5. @ActivationConfigProperty property name : acknowledgeMode

- **< durable>**

- JMSのDurable Subscriberを設定します。
- 様々な場所で設定可能であり、優先順位は以下のとおりです。
  1. jeus-ejb-dd.xml/<jeus-bean>/<durable>
  2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : subscriptionDurability
  3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : subscriptionDurability
  4. ejb-jar.xml/<message-driven-bean>/<subscription-durability>
  5. @ActivationConfigProperty property name : subscriptionDurability

- **<msg-selector>**

- JMSのmessage selectorを設定します。
- 様々な場所で設定可能であり、優先順位は以下のとおりです。
  1. jeus-ejb-dd.xml/<jeus-bean>/<msg-selector>
  2. jeus-ejb-dd.xml/<jeus-bean>/<activation-config> property name : messageSelector
  3. ejb-jar.xml/<message-driven-bean>/<activation-config> property name : messageSelector
  4. ejb-jar.xml/<message-driven-bean>/<message-selector>
  5. @ActivationConfigProperty property name : messageSelector

---

#### 参考

上記設定についての詳細内容は、『JEUS MQガイド』、『JEUS JCAガイド』、『JMS標準』、『EJB標準』を参照してください。

---

### 9.2.3. JNDI SPIの環境設定

基本的にJEUS MDBは、JEUSネーミング・サービスを利用してJMSのコネクションをルックアップして使用します。

別のJMSサービス(IBM MQまたはSONIC MQ)を利用する場合、別のサービスからコネクションを取得する必要があります。このとき、JMSサービスを含む外部のネーミング・サービスを使用するMDBを設定することができます。MDBがJMSサービスをルックアップするためには、それぞれのMDBに<jndi-spi>を設定します。

以下は、JNDI SPIの環境設定例です。

#### [例 9.4] JNDI SPIの環境設定: <<jeus-ejb-dd.xml>>

```
<jeus-ejb-dd>
    . . .
    <beanlist>
        . . .
        <jeus-bean>
            . . .
            <jndi-spi>
                <mq-vendor>SONICMQ</mq-vendor>
                <initial-context-factory>
                    acme.jndi.ACMEContextFactory
                </initial-context-factory>
                <provider-url>
                    protocol://localhost:2345
                </provider-url>
            </jndi-spi>
            . . .
        </jeus-bean>
        . . .
    </beanlist>
    . . .
</jeus-ejb-dd>
```

以下は、<jndi-spi>下位の設定タグについての説明です。

タグ	説明
<mq-vendor>	MDBがコネクションを確立するMQ/JMSのベンダー名です(JNDIネーミング・サービスを利用して設定)。「IBMMQ」と「SONICMQ」が公式的なサポート・ベンダーです
<initial-context-factory>	ネーミング・サーバーのためのJNDI InitialContext Factoryクラス名を使ってJMSサービスに接続されます
<provider-url>	要求によってネーミング・サービスに接続するときに使用するURLです



# 第10章 EJBタイマー・サービス

EJBタイマー・サービスとは、EJBが指定時間または周期的にコールバックが受けられるサービスです。基本的な使用方法についてはEJB仕様に説明されています。本章では、JEUS EJBで提供するタイマー・サービスおよび設定について説明します。

## 10.1. タイマー・サービスの設定

JEUS EJBタイマー・サービスは、基本的にはEJB仕様をベースにしていますが、永続的にタイマーを管理する機能は、ユーザーや性能に応じて選択的に使用できます。

以下は、タイマー・サービスの設定です。

- EJBエンジンのタイマー・サービス

タイマー・サービスを使用するすべてのBeanに適用される共通的な設定と、永続的なタイマー・サービスを可能にする設定をします。

- jeus-ejb-dd.xml

各Beanがデプロイまたはアンデプロイされるとき、永続的なタイマーを管理する方法について設定します。

### 10.1.1. 永続的なタイマー・サービスの設定(EJBエンジン)

WebAdminを使用してEJBエンジンの永続的なタイマー・サービスが設定できます。

1. WebAdminの**[Servers]**メニューを選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから目的のサーバーを選択すると、サーバー設定画面へ移動します。サーバー設定画面にて**[Engine]** > **[Ejb Engine]** > **[Timer Service]**を選択します。
2. **[LOCK & EDIT]**ボタンをクリックして、設定変更モードに切り替えます。
3. **[Timer Service]**設定画面にて、基本情報と**詳細設定**の項目を設定した後、**[確認]**ボタンをクリックします。

[図 10.1] 永続的なタイマー・サービスの設定 - 基本設定

Timer Service

HISTORY

EJB タイマーサービスを設定します。

ヘルプ ?

Basic Resource Engine

Web Engine | Jms Engine | Ejb Engine

Basic Active Management Time Service

動的設定 \* 必須項目

確認 再設定 削除

Support Persistence	<input checked="" type="checkbox"/>	EX true [デフォルト: true] EJB タイマーの永続的なサポート有無を指定します。基本的にEJB タイマーは永続的ですが、この機能が不要な場合はfalseに設定し、一括で非永続的に指定することができます。
Max Retrial Count	1	EX 3 [デフォルト: 1] タイムアウトコールバックメソッドでエラーが発生した場合や、当該メソッドに関連するトランザクションがロールバックされた場合、再実行する最大数を示します。
Retrial Interval	5000	EX 5000 [デフォルト: 5000] 再実行する間隔です。

以下は、**詳細設定**にて永続的なタイマー・サービスを設定した例です。

[図 10.2] 永続的なタイマー・サービス - 詳細設定

詳細設定

すべてを開く

Thread Pool

Min 2

Max 30

Period 3600000 ms

Database Setting

Db Vendor oracle EX Oracle

Data Source Id \* datasource1

以下は、各設定項目についての説明です。

- Thread Pool

タイマー・サービスがtimeout()メソッドを実行するときに使用するスレッド・プールについての説明です。

項目	説明
Min	プーリングされるスレッドの最小値です
Max	プーリングされるスレッドの最大値です
Period	プーリングされるスレッドを整理する時間です

- Database Setting

永続的なタイマー・サービスを使用するときに設定する必須項目です。

永続的なタイマーはDBを使用するために内部的にCMP Beanを使っており、CMP Beanは以下の項目を利用して設定されます。各項目はCMP Beanのスキーマ設定と同じです。

項目	説明
Db Vendor	タイマーCMP Beanが使用するDBのベンダーを指定します
Data Source Id	タイマーCMP Beanが使用するデータソース・リソース名を指定します

4. 設定内容を動的に反映するため、**[Activate Changes]**ボタンをクリックします。

## 10.1.2. 永続的なタイマーの処理(jeus-ejb-dd.xml)

永続的なタイマー・サービスを使用するように設定されている場合、ユーザーはEJB Bean別に永続的なタイマー・サービスの使用有無が設定できます。さらに、デプロイする前にDBに残っている永続的なタイマーを使用するか、削除するかなどが設定できます。jeus-ejb-dd.xmlを利用して設定します。

---

### 参考

永続的なタイマー・サービスはWebAdminで設定できます。WebAdminでの詳しい設定方法については、[「10.1.1. 永続的なタイマー・サービスの設定\(EJBエンジン\)」](#)を参照してください。

---

以下は、jeus-ejb-dd.xmlに永続的なタイマー処理を設定した例です。**<jeus-bean>**の**<timer-service>**タグ内に設定します。

### [例 10.1] 永続的なタイマーの処理 : <<jeus-ejb-dd.xml>>

```
<jeus-ejb-dd>
. . .
<beanlist>
. . .
  <jeus-bean>
    . . .
    <timer-service>
      <support-persistence>
        true
      </support-persistence>
    </timer-service>
    . . .
  </jeus-bean>
. . .
```

```

    </beanlist>
    . . .
</jeus-ejb-dd>

```

以下は、設定タグについての説明です。

タグ	説明
<support-persistence>	Beanのタイマーを永続的に管理するか否かを決めます

## 10.1.3. Cluster-Wide Timer Serviceの設定

クラスターの場合、すべてのEJBエンジンが同様なEJBタイマー・サービスを使用するため、共通した設定が必要となります。したがって、各MSに設定された値は無視されます。

以下は、WebAdminを使用してCluster-Wide Timer Serviceを設定する手順です。

1. WebAdminの**[Clusters]**メニューを選択すると、クラスター・リスト照会画面が表示されます。クラスター・リストから目的のクラスターを選択すると、クラスター設定画面へ移動します。
2. **[LOCK & EDIT]**ボタンをクリックして、設定変更モードに切り替えます。
3. クラスターの設定画面にて、**Cluster Wide Timer Service**領域の項目を設定し、**[確認]**ボタンをクリックします。

### [図 10.3] Cluster-Wide Timer Serviceの設定

Cluster Wide Timer Service

Cluster Wide Timerサービスに関する情報を設定します。

Database Setting
Cluster Wide Timerサービスが永続的に管理するタイマーハンドラを他ベンダのDBに保存したい場合に設定します。

Db Vendor	<div>oracle</div> <div> <div>Oracle</div> </div> <div>タイマーハンドラを保存するためのDBのベンダを示します。自動的にベンダを確認しますが、必要な場合は設定することも可能です。</div>
Data Source Id *	<div>datasource1</div> <div> タイマーサービスがタイマーの永続性を提供するために使用するDBのコネクションプール名です。リソース項目のデータソースにDBリソースとして設定されている必要があり、XA連動が可能なタイプのコネクションプールを設定します。 </div>

各設定項目についての説明は、「10.1.1. 永続的なタイマー・サービスの設定(EJBエンジン)」の「Database Setting」項目と同様です。

4. 設定内容を動的に反映するため、**[Activate Changes]**ボタンをクリックします。



## 10.2. タイマー・モニタリング

WebAdminおよびコンソール・ツールを使って、動作中のEJBタイマーに対するモニタリングおよび動作のキャンセルが可能です。

### WebAdminの使用

WebAdminを使用したモニタリングおよび動作のキャンセル方法は以下のとおりです。

- モニタリング

WebAdminで[Monitoring] > [EJB Timer]を選択すると、[EJB Timer]画面へ移動します。動作中のEJBタイマー・リストと情報が以下のように検索されます。

[図 10.4] タイマー・モニタリング

domain1

EJB Timer

HISTORY

現在サーバで動作しているEJBタイマーの情報を照会します。

Persistent Timer List: server[server1]

adminServer

server1

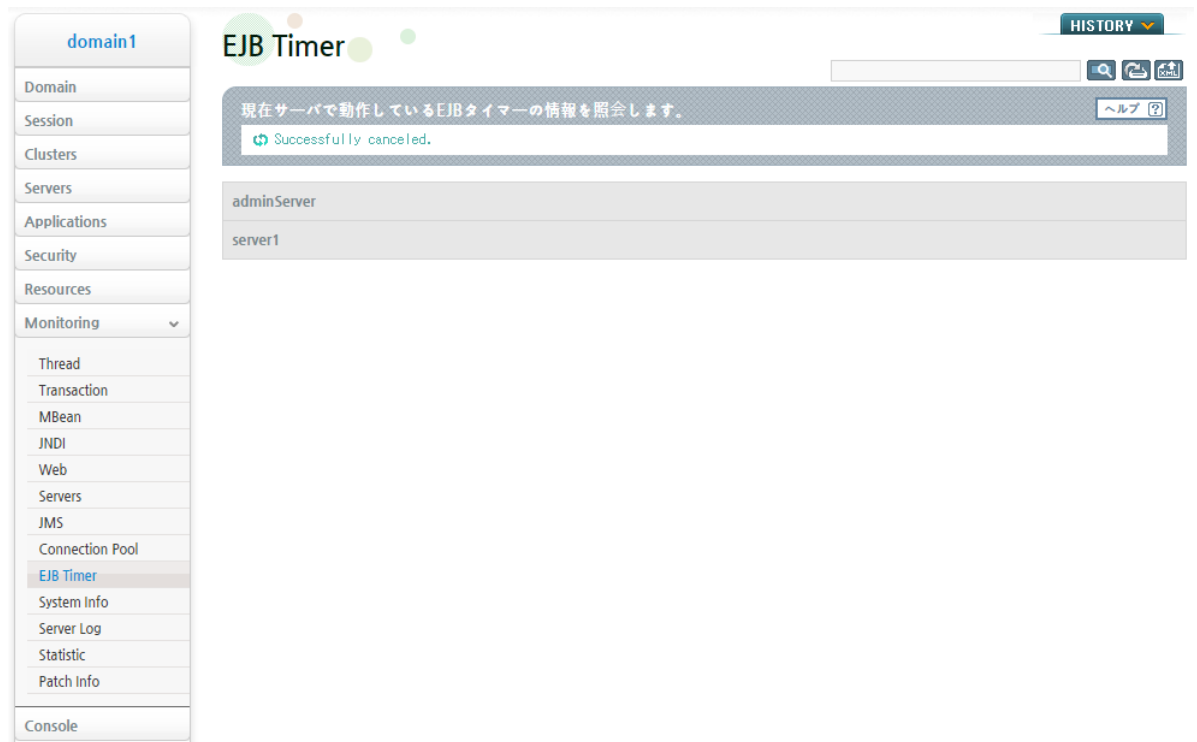
Module ID: singleAnno

ID	BEAN	METHOD	SCHEDULE	INFO	NEXT	Cancel
2	PersistenceTest Bean	print ()	sec=*,min=*,hour=*,dayOfMonth=*,dayOfWeek=*,month=*,year=*		2016-09-30 (金) 午後 04:20:30 KST	Cancel

- 動作のキャンセル

[EJB Timer]画面にて、特定のタイマーまたは「**Command**」列の[**cancel**]ボタンをクリックすると、動作中のタイマーをキャンセルすることができます。以下は、[**cancel**]ボタンをクリックして、EJBタイマーの動作をキャンセルした画面です。

[図 10.5] タイマー動作のキャンセル



## コンソール・ツールの使用

タイマーのモニタリングおよび動作のキャンセルはコンソール・ツールを使用することもできます。

- モニタリング

**ejb-timer-info**コマンドを使って特定サーバーのEJBタイマー情報が確認できます。

```
ejb-timer-info -server <server-name>
```

- 動作のキャンセル

**cancel-ejb-timer**コマンドを使って特定サーバーのEJBタイマーをキャンセルすることができます。

```
cancel-ejb-timer -server <server-name>
```

---

#### 参考

上記の2つのコマンドおよびEJBエンジン関連のコマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.7. EJBエンジン関連コマンド」を参照してください。

---

## 10.3. タイマー・サービス使用時の注意事項

### 10.3.1. 永続的なタイマーとJDBCコネクション

永続的なタイマーはDBに保存され、同タイマーを管理するCMP BeanはEJBエンジンのTime Service - Database Setting - Data Source Id値を使用します。

永続的なタイマーを使用するBeanが含まれたトランザクションはタイマーCMP Beanが使用するデータソースも管理します。これを考慮してデータソースをLocalXAResourceで使用するか、XAResourceで使用するかを決めます。

---

#### 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*』を参照してください。

---



# 第11章 EJBクライアント

本章では、EJBクライアント・プログラミングの例と初期コンテキスト(InitialContext)の設定方法について説明します。

## 11.1. 概要

EJBクライアントとは、EJBエンジンからデプロイされたEJBコンポーネントの業務ロジックを呼び出すすべてのアプリケーションのことをいいます。EJBクライアントには、サーブレット、アプレット、別のEJB、スタンドアローンJavaプログラムなどが含まれます。

本章では、スタンドアローンJavaプログラムを利用してEJBクライアントを作成、実行する際に必要な基本的な情報を提供します。

---

### 参考

クライアント・コンテナ・アプリケーションについては、『JEUS アプリケーションクライアントガイド』を参照してください。

---

## 11.2. EJBアクセスのためのクライアント・プログラミング

以下は、「HelloApp」というエクスポート名を持つEJBをルックアップし、sayHello()メソッドを呼び出す一般的なクライアント・アプリケーションの実装例です。

### [例 11.1] <<HelloClient.java>>

```
package hello;
import java.rmi.*;
import javax.ejb.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class HelloClient {
    private void run()
    {
        try {
            //JEUS InitialContextの取得
            InitialContext ctx = new InitialContext();

            //EJBの取得
```

```

        Hello hello = (Hello)ctx.lookup("HelloApp");
        //Business Methodの呼び出し
        String s = hello.sayHello();
    }
    catch (Exception e)
    { // Exception handling. }
}

public static void main(String args[])
{
    HelloClient hclient = new HelloClient();

    hclient.run();
}
}

```

上記の例で、初期コンテキストを取得する太字の「**InitialContext ctx = new InitialContext();**」部分が正常に動作するには、初期コンテキストの属性を設定する必要があります。初期コンテキストの設定については、「[11.3. 初期コンテキストの設定](#)」を参照してください。

## 11.3. 初期コンテキストの設定

初期コンテキストは、JNDI名前空間の命名されたすべてのオブジェクトを検索するときに使用するオブジェクトです。EJBを使用する際に必要な重要コンポーネントです。

EJBでは、同オブジェクトを使用してEJBエンジンのBeanインスタンスの参照を検索します。JEUSの初期コンテキスト・インスタンスを取得するには、初期コンテキストを取得する前に5つのネーミング・コンテキストの属性のうち、少なくとも1つを指定する必要があります。そうすると、クライアント・アプリケーションがJEUSネーミング・サービス・サーバー(JNSサーバー)と通信することができます。

JNDIの初期コンテキスト・オブジェクトがJVMの「-D」属性(推奨事項)とEJBクライアント・コードのハッシュテーブル(Hashtable)を使用して設定される方法と、それぞれの場合においてEJBクライアントを呼び出す方法について説明します。

以下は、JEUSでサポートする初期コンテキスト・オブジェクトを取得する前に設定する5つのネーミング・コンテキスト環境の属性です。

- **INITIAL\_CONTEXT\_FACTORY**(required)

ネーミング・コンテキストの属性は、「jeus.jndi.JNSContextFactory」の値が必要です。同値は、ネーミング・コンテキストを生成するときに使用するファクトリーのクラス名です。

- **URL\_PKG\_PREFIXES**

jeus.jndi.jns.urlで設定され、JEUS初期コンテキストのURLスキーマを使ってオブジェクトを検索します。

- **PROVIDER\_URL**

JEUSネーミング・サーバー(JNSサーバー)のIPアドレスを設定します。(デフォルト値: 127.0.0.1 (localhost))

- **SECURITY\_PRINCIPAL**

JEUSネーミング・サーバーの認証手順に必要なユーザー情報を設定します。

実行スレッドにユーザー情報が登録されていない場合、基本的にguestとして認証されます。ユーザー情報を指定すると、JEUSセキュリティ領域(Security realm)に定義されたユーザー名を有するユーザー情報が使用されます。

- **SECURITY\_CREDENTIALS**

認証手順で使用するユーザーのパスワードを設定します。認証に失敗した場合は、以前のユーザー情報が維持されます。認証に成功すると、新しいユーザー情報が使われます。

---

#### 参考

上記の基本的な属性以外に、特性を追加設定することができます。詳細については、『*JEUS サーバガイド*』の「第4章 JNDIネーミング・サーバー」を参照してください。

---

## 11.3.1. JVM属性を使用したネーミング属性値の設定

上述した属性は、JVMインタプリター(interpreter)の「-D」を使用して設定できます。同オプションをEJBクライアント・アプリケーションを開始するために使用するJavaコマンドに追加します。

以下は、各ネーミング属性を「-D」を使って設定する例です。

- **INITIAL\_CONTEXT\_FACTORY** (required)

```
-Djava.naming.factory.initial=jeus.jndi.JNSContextFactory
```

- **URL\_PKG\_PREFIXES**

```
-Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
```

- **PROVIDER\_URL**

```
-Djava.naming.provider.url=<host_address>
```

- **SECURITY\_PRINCIPAL**

```
-Djava.naming.security.principal=<username>
```

## ● SECURITY\_CREDENTIALS

```
-Djava.naming.security.credentials=<password>
```

以下は、クライアントを呼び出すときに「-D」オプションを使ってEJBクライアントのネーミング・コンテキストを設定する例です。「user1」というユーザー名と「password1」というパスワードが、192.168.10.10アドレスに登録されたJEUSネーミング・サーバーを利用してEJBスタブを検索するEJBクライアントにどのように使用されるのかを示します。

```
$ java -classpath
    %JEUS_HOME%\lib\client\jclient.jar
    -Djava.naming.factory.initial=jeus.jndi.JNSContextFactory
    -Djava.naming.factory.url.pkgs=jeus.jndi.jns.url
    -Djava.naming.provider.url=192.168.10.10
    -Djava.naming.security.principal=user1
    -Djava.naming.security.credentials=password1
    HelloClient
```

---

### 参考

JEUSにデプロイしたEJBが2.xスタイルの場合は、別途のクライアント用のライブラリーを有する必要があります。同ライブラリーは、通常appcompilerを使ってEJBをコンパイルするときに作成できます。

クラスFTPサーバーを使用する場合は、クライアントが自動的に必要なクラスをダウンロードして使います。EJB 3.0以上の場合はこの事項を考慮する必要はありません。

---

## 11.3.2. ハッシュテーブルを利用したネーミング属性の設定

ネーミング・コンテキストの属性を設定する別の方法として、ハッシュテーブルを使ってクライアント・コード内のネーミング属性データを直接ハッシュテーブルに入力する方法があります。

以下は、実装例です。

```
Context
{
    ctx = null; Hashtable ht = new Hashtable();
    ht.put(Context.INITIAL_CONTEXT_FACTORY, jeus.jndi.JNSContextFactory);
    ht.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
    ht.put(Context.PROVIDER_URL, "192.168.10.10");
    ht.put(Context.SECURITY_PRINCIPAL, "user1");
    ht.put(Context.SECURITY_CREDENTIALS, "password1");
    try { ctx = new InitialContext(ht); . . .
```

上記の例で、ハッシュテーブルが初期コンテキストのconstructorパラメータとして渡されていることに注目します。



以下の例は、クライアント・クラスにネーミング属性を設定した場合、EJBクライアントを実行する方法です。

```
$ java -classpath  
    $JEUS_HOME/lib/client/jclient.jar HelloClient
```

---

### 参考

ハッシュテーブルを使用した方法は、値をハードコーディングする方法なのでお勧めしません。

---



# 第12章 拡張機能

本章では、JEUSでEJBを開発するときに使用できる拡張機能について説明します。

## 12.1. 作業領域(WorkArea)サービス

作業領域サービスとは、プログラムが暗黙的(implicit)なコンテキストを継続して伝播するときに使用できる機能です。同機能を使用すると、セキュリティ・コンテキストやトランザクション・コンテキストのように、別途のパラメータで渡さなくてもローカルまたはリモート・メソッドを呼び出すときに継続して伝播されます。作業領域は、コンテキストを伝播するときやメソッドのパラメータが多くなる場合に使用できます。

作業領域は一種のユーザー保存空間として、名前と値のペア形式のマップで保存されます。作業領域が新しく開始されると、現在のスレッドと一緒に移動するため、呼び出されたメソッドやEJBのようなコンポーネントで引続き使用することができます。また、リモートEJBを呼び出す場合にも伝播(propagation)される特性があります。

作業領域サービスを利用するために、UserWorkAreaインターフェースであるjeus.workarea.UserWorkAreaを使用します。

---

### 参考

詳しいAPI情報については、JEUS API Javadocドキュメントのjeus.workareaパッケージを参照してください。

現在、リモート作業領域の伝播機能は、EJB 3.0ビジネス・インターフェースを使用する場合のみ有効です。EJBオブジェクトのリモート・コンポーネント・インターフェースを使用するか、JEUSの基本RMI呼び出しではなくIIOP呼び出しの場合は伝播できません。

---

### 12.1.1. UserWorkAreaインターフェース

UserWorkAreaインターフェースは、UserWorkAreaの開始、終了、操作に必要なすべてのメソッドを定義します。

#### [例 12.1] <<UserWorkAreaインターフェース>>

```
public interface UserWorkArea {
    public void begin(String name);
    public void complete()
        throws NoWorkAreaException, NotOriginatorException;
    public String getName();
}
```

```

public String[] retrieveAllKeys();
public void set(String key, java.io.Serializable value)
    throws NoWorkAreaException, NotOriginatorException,
        PropertyReadOnlyException;
public void set(String key, java.io.Serializable value, PropertyModeType mode)

    throws NoWorkAreaException, NotOriginatorException,
        PropertyReadOnlyException;
public java.io.Serializable get(String key);
public PropertyModeType getMode(String key);
public void remove(String key)
    throws NoWorkAreaException, NotOriginatorException,
        PropertyReadOnlyException;
}

```

---

### 参考

メソッドに定義した詳細情報については、Javadocを参照してください。

---

## 12.1.2. PropertyModeタイプ

作業領域に保存される値は、それぞれのPropertyModeを持ちます。

各PropertyModeタイプは以下のとおりです。

タイプ	説明
NORMAL	登録された値の修正、削除が可能です
READ_ONLY	UserWorkAreaに登録された値の修正、削除ができません

## 12.1.3. 例外

UserWorkAreaで定義された例外は以下のとおりです。

例外	説明
WorkAreaException	作業領域で発生する例外のルート例外です
NotOriginatorException	作業領域を開始していないスレッドが値を修正、削除、終了しようとする場合に発生します
PropertyReadOnlyException	PropertyModeがREAD_ONLYに設定された値を変更する場合に発生します

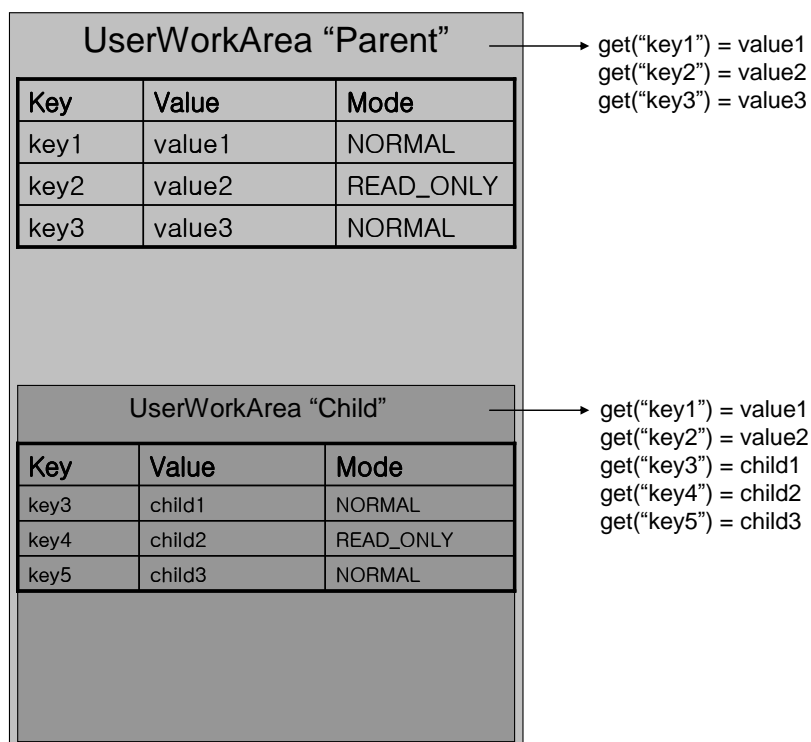
## 12.1.4. ネストされたUserWorkArea

UserWorkAreaはネスト可能です。すでにUserWorkAreaが存在している状態で、新しいUserWorkAreaを開始すると、当該UserWorkAreaは既存のUserWorkAreaにネストされます。

ネストされたUserWorkAreaは、上位のUserWorkAreaに保存された値をそのまま使用可能で、新しい値を追加することができます。ネストされたUserWorkAreaで追加された値は、当該UserWorkAreaでのみ有効で、完了すると消えます。

ネストされたUserWorkAreaでの登録情報は以下のとおりです。

[図 12.1] ネストされたUserWorkAreaでの登録情報



## 12.1.5. UserWorkAreaを使用するアプリケーション・プログラムの開発

本節では、UserWorkAreaを使用するEJBの例を作成し、UserWorkAreaインターフェースの使用方法について説明します。

EJBの例は、UserWorkAreaSampleSenderとUserWorkAreaSampleReceiverの2つのEJBで構成されています。送信側(Sender)でUserWorkAreaを生成してデータを伝播し、受信側(Receiver)でUserWorkAreaに設定された値を利用してメッセージを生成し、元の送信側に返します。

### 1. UserWorkAreaへのアクセス

UserWorkAreaを使用するために、まずJNDIでUserWorkAreaをルックアップします。

以下の例は、JNDIでUserWorkAreaをルックアップする方法です。

**[例 12.2] UserWorkAreaへのアクセス : <<UserWorkAreaSampleSenderBean.java>>**

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {

    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            //UserWorkAreaをJNDIから取得します。
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
        } catch (NamingException e) {
            // Do Something...
        }
        return message;
    }
}
```

**2. 新しいUserWorkAreaの開始**

JNDIから最初にルックアップしたUserWorkAreaにはいかなる情報も含まれていないため、新しいUserWorkAreaを開始する必要があります。UserWorkAreaの名前がnullの場合は、NullPointerExceptionが発生します。

**[例 12.3] 新しいUserWorkAreaの開始 : <<UserWorkAreaSampleSenderBean.java>>**

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {

    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            // 新しいUserWorkAreaを開始します。
            userWorkArea.begin("UserWorkArea1");
        } catch (NamingException e) {
            // Do Something...
        }
        return message;
    }
}
```

### 3. 作業領域に登録情報を設定

新しく開始したUserWorkAreaに登録情報を設定します。登録情報は<key、value、mode>で構成されます。「key」は文字列で、「value」は直列化(Serialization)が可能なオブジェクトです。

値を設定する際に現在のUserWorkAreaが存在しない場合は、NoWorkAreaExceptionが発生します。開始したUserWorkAreaでない場合はNotOriginatorExceptionが発生し、すでにREAD\_ONLYに設定された値を修正するとPropertyReadOnlyExceptionが発生します。

#### [例 12.4] 作業領域に登録情報を設定 : <<UserWorkAreaSampleSenderBean.java>>

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {

    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            userWorkArea.begin("UserWorkArea1");
            // userWorkAreaに値をNORMALに設定します。
            userWorkArea.set("name", "johan");
            // userWorkAreaに値をREAD_ONLYに設定します。
            userWorkArea.set("company", "TmaxSoft", PropertyModeType.READ_ONLY);

            UserWorkAreaSampleReceiver receiver
                = (UserWorkAreaSampleReceiver) ic.lookup("receiver");
            message = receiver.createMessage();
        } catch (NamingException e) {
            // Do Something...
        } catch (NoWorkAreaException e) {
            // Do Something...
        } catch (PropertyReadOnlyException e) {
            // Do Something...
        } catch (NotOriginatorException e) {
            // Do Something...
        }

        return message;
    }
}
```

### 4. 作業領域に設定された登録情報の取得

受信側から伝播された、UserWorkAreaから設定された登録情報を取得してメッセージを生成します。保存された情報を取得する際、UserWorkAreaに存在しないキーを使用するとnullがリターンされます。

**[例 12.5] 作業領域に設定された登録情報の取得 : <<UserWorkAreaSampleReceiverBean.java>>**

```
public class UserWorkAreaSampleReceiverBean implements UserWorkAreaSampleReceiver
{
    public String createMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            // UserWorkAreaに保存した値を取得します。
            String name = (String)userWorkArea.get("name");
            String company = (String)userWorkArea.get("company");
            message = "Hello " + name + " from " + company;

        } catch (NamingException e) {
            // Do Something...
        }

        return message;
    }
}
```

**5. UserWorkAreaの終了**

開始したUserWorkAreaを終了します。終了するときは必ず開始されたオリジネーター(Originator)でのみ可能です。それ以外で終了すると、NotOriginatorExceptionが発生します。

**[例 12.6] UserWorkAreaの終了 : <<UserWorkAreaSampleSenderBean.java>>**

```
public class UserWorkAreaSampleSenderBean implements UserWorkAreaSampleSender {

    public String getMessage() {
        InitialContext ic;
        String message = null;
        try {
            ic = new InitialContext();
            UserWorkArea userWorkArea
                = (UserWorkArea) ic.lookup("java:comp/UserWorkArea");
            userWorkArea.begin("UserWorkArea1");
            userWorkArea.set("name", "user1");
            userWorkArea.set("company", "TmaxSoft", PropertyModeType.READ_ONLY);

            UserWorkAreaSampleReceiver receiver
                = (UserWorkAreaSampleReceiver) ic.lookup("receiver");
            message = receiver.createMessage();
            userWorkArea.complete(); // UserWorkAreaを終了します。
        }
    }
}
```



```
        } catch (NamingException e) {  
            // Do Something...  
        } catch (NoWorkAreaException e) {  
            // Do Something...  
        } catch (PropertyReadOnlyException e) {  
            // Do Something...  
        } catch (NotOriginatorException e) {  
            // Do Something...  
        }  
  
        return message;  
    }  
}
```



# 付録 A. 基本JavaタイプとDBフィールドのマッピング

本付録では、JEUSの特性であるJavaフィールド型と、JEUSでサポートする主要DBベンダーの列型と基本マッピングについて説明します。

## A.1. 概要

基本マッピングは、CMP Beanの<schema-info><cm-field><type>がjeus-ejb-dd.xmlに指定されていないときに使用します。この場合、EJBシステムはBeanのCMPフィールドの中からJavaフィールドを検索してJavaタイプを取得します。<type>は以下の各節で登場する基本マッピングに指定されたものと同じであるとしています。

以下の表では、3の列型について記述しています。

- **Javaフィールド型**

検索によって発見されたEJBのCMPフィールド型

- **SQL型**

java.sql.Typesクラスで定義された一般的なSQL型

- **DBの列型**

発見されたJavaフィールド型をベースにして、新しい表を作成する際にJEUSが使用するDBの列型の名前

## A.2. Tiberoのフィールドと列型のマッピング

[表 A.1] TiberoのためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	Tibero DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
Boolean	BIT	SMALLINT

Javaフィールド型	SQL型(java.sql.Types)	Tibero DBの列型
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	LONG RAW
java.sql.Date	DATE	DATE
java.sql.Time	TIME	DATE
java.sql.Timestamp	TIMESTAMP	DATE
<Java object>	JAVA OBJECT	LONG RAW

## A.3. Oracleのフィールドと列型のマッピング

[表 A.2] OracleのためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	Oracle DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
Boolean	BIT	SMALLINT
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT

Javaフィールド型	SQL型(java.sql.Types)	Oracle DBの列型
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	LONG RAW
java.sql.Date	DATE	DATE
java.sql.Time	TIME	DATE
java.sql.Timestamp	TIMESTAMP	DATE
<Java object>	JAVA OBJECT	LONG RAW

## A.4. Sybaseのフィールドと列型のマッピング

[表 A.3] SybaseのためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	Sybase DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	BIT
Boolean	BIT	BIT
java.lang.Byte	TINYINT	TINYINT
Byte	TINYINT	TINYINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INT
Int	INTEGER	INT
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)

Javaフィールド型	SQL型(java.sql.Types)	Sybase DBの列型
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	IMAGE
java.sql.Date	DATE	DATETIME
java.sql.Time	TIME	DATETIME
java.sql.Timestamp	TIMESTAMP	Not supported
<Java object>	JAVA OBJECT	IMAGE

## A.5. MSSQLのフィールドと列型のマッピング

[表 A.4] MSSQLのためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	MSSQL DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	BIT
Boolean	BIT	BIT
java.lang.Byte	TINYINT	TINYINT
Byte	TINYINT	TINYINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INT
Int	INTEGER	INT
java.lang.Long	BIGINT	NUMERIC(22,0)
Long	BIGINT	NUMERIC(22,0)
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	FLOAT
Double	DOUBLE	FLOAT
byte[]	LONGVARBINARY	IMAGE

Javaフィールド型	SQL型(java.sql.Types)	MSSQL DBの列型
java.sql.Date	DATE	DATETIME
java.sql.Time	TIME	DATETIME
java.sql.Timestamp	TIMESTAMP	DATETIME
<Java object>	JAVA OBJECT	IMAGE

## A.6. DB2のフィールドと列型のマッピング

[表 A.5] DB2のためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	DB2 DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)
java.lang.Boolean	BIT	SMALLINT
Boolean	BIT	SMALLINT
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHARACTER(4)
Char	CHAR	CHARACTER(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	BIGINT
Long	BIGINT	BIGINT
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE
Double	DOUBLE	DOUBLE
byte[]	LONGVARBINARY	VARCHAR(255)
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
<Java object>	JAVA OBJECT	LONG VARCHAR

## A.7. Cloudscapeのフィールドと列型のマッピング

[表 A.6] CloudscapeのためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	Cloudscape DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	DECIMAL(15,5)
java.lang.Boolean	BIT	BOOLEAN
Boolean	BIT	BOOLEAN
java.lang.Byte	TINYINT	TINYINT
Byte	TINYINT	TINYINT
java.lang.Character	CHAR	CHAR(4)
Char	CHAR	CHAR(4)
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	LONGINT
Long	BIGINT	LONGINT
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	LONG BIT VARYING
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
<Java object>	JAVA OBJECT	LONG BIT VARYING

## A.8. Informixのフィールドと列型のマッピング

[表 A.7] InformixのためのEJB CMPフィールドとDB列型のマッピング

Javaフィールド型	SQL型(java.sql.Types)	Informix DBの列型
java.lang.String	VARCHAR	VARCHAR(255)
java.math.BigDecimal	NUMERIC	NUMERIC(15,5)



Javaフィールド型	SQL型(java.sql.Types)	Informix DBの列型
java.lang.Boolean	BIT	VARCHAR
Boolean	BIT	VARCHAR
java.lang.Byte	TINYINT	SMALLINT
Byte	TINYINT	SMALLINT
java.lang.Character	CHAR	CHAR
Char	CHAR	CHAR
java.lang.Short	SMALLINT	SMALLINT
Short	SMALLINT	SMALLINT
java.lang.Integer	INTEGER	INTEGER
Int	INTEGER	INTEGER
java.lang.Long	BIGINT	SERIAL8
Long	BIGINT	SERIAL8
java.lang.Float	REAL	REAL
Float	REAL	REAL
java.lang.Double	DOUBLE	DOUBLE PRECISION
Double	DOUBLE	DOUBLE PRECISION
byte[]	LONGVARBINARY	VARCHAR(255)
java.sql.Date	DATE	DATE
java.sql.Time	TIME	DATETIME HOUR TO SECOND
java.sql.Timestamp	TIMESTAMP	DATETIME YEAR TO SECOND
<Java object>	JAVA OBJECT	BYTE



# 付録 B. Instant EJB QL APIリファレンス

本付録では、Instant EJB QL APIリファレンスのメソッドとインターフェースを使用する方法について説明します。

## B.1. 概要

EJB QL APIは、EJBクライアント・アプリケーションの開発者がクライアントのコードに直接EJB QL問合せを指定して、EJB finderメソッドの制約が解決できるようにします。同APIは、1つのインターフェースと1つのメソッドで構成されています。

---

### 参考

このAPIは、標準EJB Entity finderメソッドに比べ固定的かつ非効率的なので、なるべく使用しないようにします。

---

## B.2. The EJBInstanceFinderインターフェース

### interface jeus.ejb.bean.objectbase.EJBInstanceFinder

このインターフェースは、jeus-ejb-dd.xmlファイルの<enable-instant-ql>要素がtrueに設定されている環境で、CMP 2.0エンティティBeanのホーム・インターフェースによって実装されます。同インターフェースは、クライアント・コード内に任意のEJB QL問合せを直接挿入できるようにします。

```
public abstract interface EJBInstanceFinder extends Remote
```

## B.3. The EJBInstanceFinderメソッド

### java.util.Collection findWithInstantQL

- 使用方法

「ejbQLQuery」パラメータで表現されたEJB QL問合せに該当するBeanのEJB集合を返します。

```
java.util.Collection findWithInstantQL (String ejbQLQuery)
```

- パラメータ

パラメータ	説明
string ejbQLQuery	有効なEJB QL文として、「?」が含まれてはなりません。この文法は、JEUSで定義したEJB QLの3つの追加事項の1つです

- 戻り値

戻り値	説明
java.util.Collection	問合せの応答に該当するBeanインターフェースの集合です

- 例外

- FinderException
- RemoteException

# 用語集

## 2次記憶装置

電源を供給しなくても永久的にデータを保存する補助記憶装置です。(例: ハードディスク)

## メイン・メモリー

ランタイム・メモリーとも言います。コンピューター内部の可用メモリー(RAM)です。

## Activation(アクティベーション)

パッシベーション(Passivation)の反対概念です。Beanインスタンスを補助記憶装置からメイン・メモリーに移すことを意味します。

## Active Management

JEUSでは、正常でない状況で自動的にEメールを送信することを意味します。

## Bean Pool(Beaプール)

EJBインスタンスのプールです。

## Bean Type(Beaタイプ)

J2EEの基本Beanタイプ(例: ステートレスBean、ステートフルBean、エンティティBean、メッセージ駆動型Bean)または開発者が定義したBeanです(基本J2EE Beanの下位Bean)。

## BMP

Bean管理パーシスタンス(Bea Managed Persistence)のエンティティBeanです。

## CMP

コンテナ管理パーシスタンスのエンティティBeanです。

## Connection Pool(コネクション・プール)

EJBオブジェクトのプールです。このオブジェクトはクライアントとEJB間の接続および通信をサポートします。

## DD

Deployment Descriptor(デプロイメント記述子)の略語です。

## EJB

Enterprise Java Beansの略語です。

## EJB Clustering(EJBクラスタリング)

複数のEJBエンジンに分散されているEJB Beanの設定です。クライアントからは1つのBeanのように見えますが、内部的には安定性と性能向上のために負荷分散が行われます。

## EJB Client(EJBクライアント)

EJBのメソッドを使用するコンポーネントです。(例: 独立したJavaアプリケーション、サーブレット、またはその他のEJB)

## EJB Engine(EJBエンジン)

J2EE仕様の「EJBコンテナ」です。EJBコンポーネントのベース環境を提供します。

## EJB Failover(EJBフェイルオーバー)

同じEJBアプリケーションを実行する複数のエンジンを使用することで安定性を高める際に使用します。

## EJB Module(EJBモジュール)

EJB JAR内にある1つ、あるいはそれ以上のEJBパッケージです。EJBエンジンにデプロイされているEJBはEJBモジュールを使用します。

## Entity Cache(エンティティ・キャッシュ)

エンティティBeanで非活性化されたBeanを維持するために使用される特別なキャッシュです。このキャッシュがフルになった場合のみBeanは補助記憶装置として非活性化されます。性能向上のために使用されます。

## Load Balancing(ロード・バランシング、負荷分散)

クラスタリングでの作業が片方に偏らないようにする技術です。同じEJBアプリケーションを実行するエンジンにクライアントの要求を均等に分散して割り当てることで、性能を向上させる場合に使用します。

## MDB

Message Driven Beanの略語です。

## Object Management(オブジェクト管理)

コネクション・プールとBeanプール環境を参照してください。

## Passivation(パッシベーション)

再び要求があるまで、EJBインスタンスをメイン・メモリーから補助記憶装置に移しておくことです。

## Round Robin(ラウンドロビン)

キューのデータ構造と同じです。キューに入れられた順に選択されます。これは、コンポーネント別に公正な負荷分散を保証します。

## SF

ステートフル・セッションBeanです。

## SL

ステートレス・セッションBeanです。

## Thread Ticket Pool(スレッド・チケット・プール)

アクセス・チケットのプールです。JEUSでEJBにアクセスするには、クライアントは必ずスレッド・チケット・プールからスレッド・チケットを取得する必要があります。

# 索引

## シンボル

- <ejb-relation-map>
  - <jeus-relationship-role>, 122, 124
  - <relation-name>, 122
  - <table-name>, 124
- <env>
  - <name>, 54
  - <type>, 55
  - <value>, 55
- <export-name>, 55
- <jeus-bean>
  - <ack-mode>, 145
  - <activation-config>, 145
  - <connection-factory-name>, 144
  - <destination>, 144
  - <durable>, 146
  - <jndi-spi>, 144
  - <max-messages>, 145
  - <mdb-resource-adapter>, 144
  - <msg-selector>, 146
- <jeus-bean><cm-persistence-optimize>
  - <fetch-size>, 116
  - <init-caching>, 116
  - <subengine-type>, 116
- <jeus-bean><object-management>
  - <bean-pool>, 92, 111
  - <capacity>, 112
  - <connect-pool>, 92, 112
  - <disconnect-timeout>, 93, 113
  - <passivation-timeout>, 93, 112
- <jeus-bean><persistence-optimize>
  - <engine-type>, 115
  - <entity-cache-size>, 115
  - <include-update>, 115
  - <non-modifying-method>, 115
  - <update-delay-till-tx>, 115

- <jeus-bean><schema-info>
  - <auto-key-generator>, 121
  - <cm-field>, 118
  - <creating-table>, 119
  - <data-source-name>, 121
  - <db-vendor>, 121
  - <deleting-table>, 119
  - <jeus-query>, 121
  - <prim-key-field>, 120
  - <table-name>, 118
- <jndi-spi>, 147
  - <initial-context-factory>, 147
  - <mq-vendor>, 147
  - <provider-url>, 147
- <ref-name>, 55
- <support-persistence>, 152

## A

- accounts.xml, 60
- Active Management, 12, 17
- appcompiler, 33
- appcompilerデプロイ, 33
- Async Service, 16
  - Access Timeout, 16
  - Thread Max, 16
  - Thread Min, 16

## C

- cancel-ejb-timerコマンド, 154
- CM永続性, 102
- CM永続性の最適化, 115
- COMPONENTタイプ, 39
- CounterEJB.java
  - @jeus.ejb.Clustered, 81
  - @jeus.ejb.CreateIdempotent, 81
  - @jeus.ejb.Idempotent, 81
  - @jeus.ejb.Replication, 81

## D

- DB Insert Delay, 130
- Deployment Descriptors, 29
- domain.xml, 6, 14

## E

- EARデプロイ方式, 39
- Ejb Engine, 16
  - Enable User Notify, 16
- EJB JAR, 27
- EJB RMI Stub, 12
- ejb-jar.xml, 6
  - <ejb-ref>, 54
  - <env-entry>, 54
  - <message-destination-ref>, 54
  - <resource-env-ref>, 54
  - <resource-ref>, 54
  - <service-ref>, 54
- ejb-jar\_3\_1.xsd, 6
- ejb-timer-infoコマンド, 154
- EJBInstanceFinder, 109
- ejbLoad(), 98
- EJBのチューニング, 65
- EJBのフェイルオーバー, 78
- EJBエンジン, 2, 11
- EJBコンテナ, 11
- EJBコンポーネント, 2
- EJBセキュリティー設定, 58
- EJBモジュール, 2, 25
- EJBモジュールの制御, 39
- EJBモジュールの制御およびモニタリング
  - application-info, 43
  - redeploy-application, 42
  - start-application, 42
  - stop-application, 42
  - undeploy, 42
- Email Notify, 19
  - Bcc Address, 19
  - Cc Address, 19
  - From Address, 19
  - Property, 19
  - Sender Id, 19
  - Sender Password, 19
  - Smtp Host Address, 19
  - To Address, 19
- Exploded EARデプロイ方式, 39
- Explodedスタンドアローン・デプロイ方式, 39

Eメール通知, 17

## H

- HTTP Invoke, 12
- HTTP invokeモード, 12
- HTTP呼び出しモード, 23

## I

- init caching, 103
- INITIAL\_CONTEXT\_FACTORY, 158, 159
- Instant EJB QL, 109, 124
- Invoke Http, 16
  - Http Port, 17
  - Url, 16

## J

- JEUS EJB DD, 29
  - <beanlist>, 32
  - <ejb-ref>, 54
  - <ejb-relation-map>, 32
  - <env>, 54
  - <library-ref>, 32
  - <message-destination-ref>, 54
  - <message-destination>, 32
  - <module-info>, 31
  - <res-env-ref>, 54
  - <res-ref>, 54
  - <service-ref>, 54
- JEUS EJB QLの拡張機能, 125
- jeus-domain.xsd, 6
- jeus-ejb-dd.xml, 6
  - <clustering>, 82
  - <clustering><clustering-version>, 83
  - <clustering><create-idempotent>, 83
  - <clustering><ejb-home-idempotent-exclude-method>, 83
  - <clustering><ejb-home-idempotent-method>, 83
  - <clustering><ejb-remote-idempotent-exclude-method>, 83
  - <clustering><ejb-remote-idempotent-method>, 83
  - <clustering><enable-clustering>, 83
  - <clustering><replication>, 83



<ejb-relation-map>, 122  
<export-iiop>, 70  
<export-iiop><only-iiop>, 71  
<invoke-http><http-port>, 57  
<invoke-http><url>, 57  
<jeus-bean><schema-info>, 116  
<module-info><role-permission>, 59  
<module-info><role-permission><principal>, 59  
<module-info><role-permission><role>, 59  
<run-as-identity>, 59  
<run-as-identity><principal-name>, 60

jeus-ejb-dd.xsd, 6

JMSの設定, 142

## K

Keystoreファイル, 69

## M

Managed Server, 11

Many-to-manyリレーションシップ・マッピング, 123

Max Blocked Thread, 19

Max Idle Time, 19

Max messages, 49

## N

Non-modifyingメソッド, 102

## O

One-to-one/One-to-manyリレーションシップ・マッピング,  
122

ORACLEHINT, 125

## P

Persistent Timer Service

Database Setting, 151

PropertyModeのタイプ

NORMAL, 164

READ\_ONLY, 164

PROVIDER\_URL, 159

## R

ReadLocking, 103

Resolution, 16

RMIランタイム, 12

Run-as Identify, 59

## S

SECURITY\_CREDENTIALS, 159, 160

SECURITY\_PRINCIPAL, 159

SEQUENCE INCREMENT, 106

## T

Thread Ticket Pool, 87

TRANSACTION\_SERIALIZABLE, 105

Truststoreファイル, 69

## U

URL\_PKG\_PREFIXES, 158, 159

Use Dynamic Proxy For Ejb2, 16

UserWorkAreaの例外

NotOriginatorException, 164

PropertyReadOnlyException, 164

WorkAreaException, 164

## W

WriteLocking, 103

WriteLockingFind, 103

## あ

永続性の最適化, 113

永続的なタイマー・サービスの設定

Thread Pool, 150

エンジン・モード, 98

エンジン再起動の条件, 17

エンティティEJBチューニング, 131

オブジェクト・マネージメント, 87, 88, 91, 110

## さ

作業領域サービス, 163

障害ポリシー, 12

状態保持メカニズム, 91

- 相関(correlated)問合せ, 126
- スキーマ情報, 103, 116
- スタンドアローン・デプロイ方式, 39
- ステートフル・セッションBeanのクラスタリング設定
  - Passivation Timeout, 84
- スレッド・チケット・プール, 52
- セッションEJBの設定, 91
- セッション複製(Session Replication), 79
  - 同期的(Sync)複製モード, 79
  - 非同期的(Async)複製モード, 79

## た

- 多重呼び出し不変メソッド, 78
- デプロイ, 33

## は

- 非相関(uncorrelated)問合せ, 126
- 標準EJB DD, 29
- フェイルオーバー, 75, 78
- ブートタイム・デプロイ, 33, 34
- プーリング・セッションBean, 90

## ら

- ランタイム・デプロイ, 33, 34
- リレーションシップ・マッピングの設定, 122