HITACHI
Inspire the Next

Cosminexus V11 アプリケーションサーバ 機能解説 互換編

解説書

3021-3-J12-50

前書き

■ 対象製品

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の前書きの対象製品の説明を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ 発行

2024 年 2 月 3021-3-J12-50

変更内容

変更内容(3021-3-J12-50) uCosminexus Application Server 11-40，uCosminexus Client 11-40，uCosminexus Developer 11-40，uCosminexus Service Architect 11-40，uCosminexus Service Platform 11-40

追加・変更内容	変更個所
アプリケーションサーバ 11-40 での主な機能変更についての説明を追加した。	1.3
JDK11 または JDK17 を使用する環境ごとに，指定できるモード（推奨モードまたは V9 互換モード）についての説明を追加した。	2.2
@OneToOne および@ManyToOne での LAZY フェッチを使用する場合の注意事項について説明を変更した。	8.20.1
マニュアル訂正の内容を反映した。	-

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルをお読みになる際の前提情報については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」のはじめにの説明を参照してください。

目次

1 アプリケーションサーバの機能

この章では，アプリケーションサーバの機能の分類と目的，および機能とマニュアルの対応について説明します。また，このバージョンで変更した機能についても説明しています。

1.1 機能の分類

アプリケーションサーバは，Java EE 8 に対応した J2EE サーバを中心としたアプリケーションの実行環境を構築したり，実行環境上で動作するアプリケーションを開発したりするための製品です。Java EE の標準仕様に準拠した機能や，アプリケーションサーバで独自に拡張された機能など，多様な機能を使用できます。目的や用途に応じた機能を選択して使用することで，信頼性の高いシステムや，処理性能に優れたシステムを構築・運用できます。

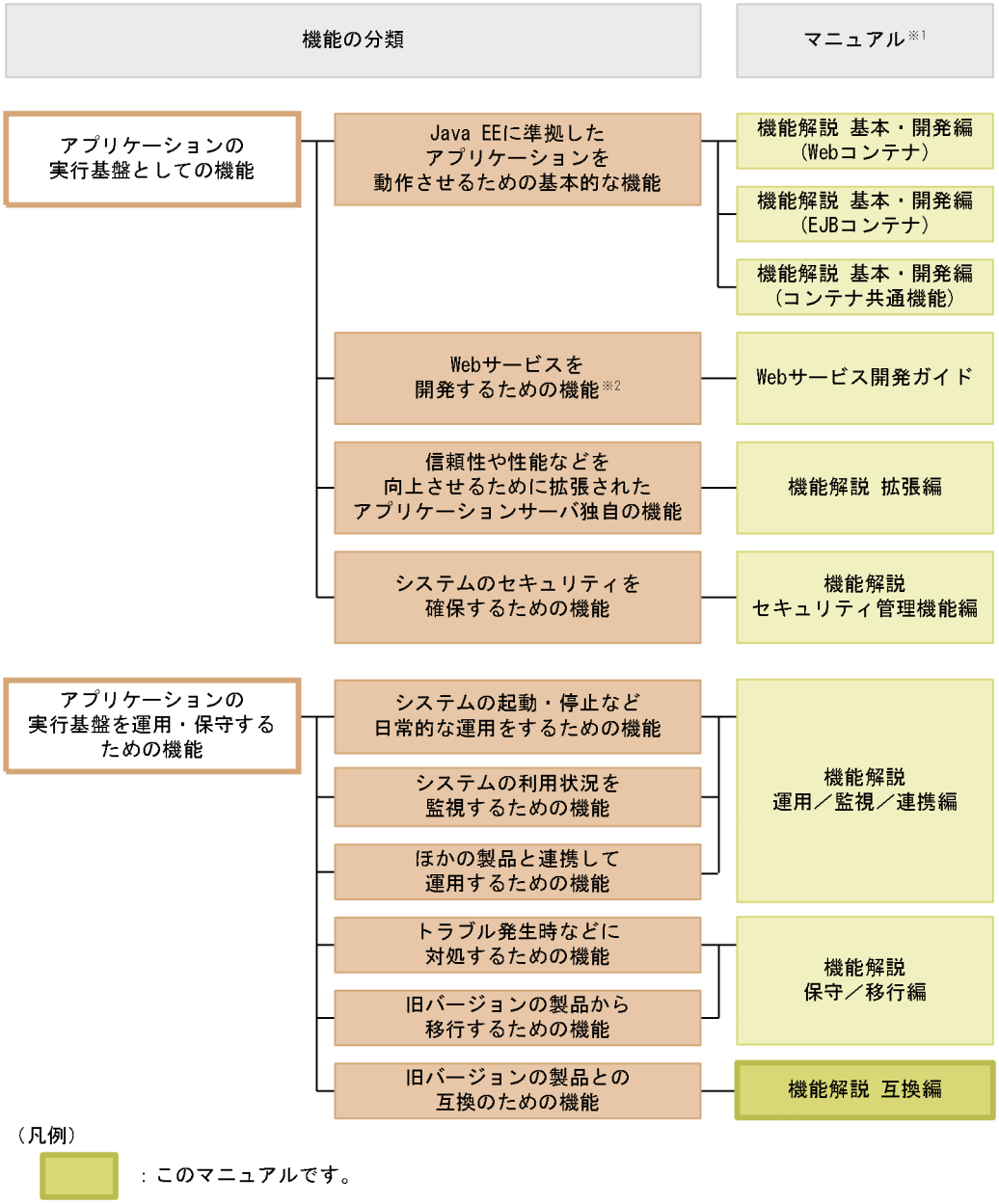
アプリケーションサーバの機能は，大きく分けて，次の二つに分類できます。

- アプリケーションの実行基盤としての機能
- アプリケーションの実行基盤を運用・保守するための機能

二つの分類は，機能の位置づけや用途によって，さらに詳細に分類できます。アプリケーションサーバのマニュアルは，機能の分類に合わせて提供しています。

アプリケーションサーバの機能の分類と対応するマニュアルについて，次の図に示します。

図 1-1　アプリケーションサーバの機能の分類と対応するマニュアル



注※1

マニュアル名称の「アプリケーションサーバ」を省略しています。

注※2

アプリケーションサーバでは，SOAP Web サービスと RESTful Web サービスを実行できます。目的によっては，マニュアル「アプリケーションサーバ Web サービス開発ガイド」以外の次のマニュアルも参照してください。

SOAP アプリケーションを開発・実行する場合

- アプリケーションサーバ SOAP アプリケーション開発の手引

SOAP Web サービスや SOAP アプリケーションのセキュリティを確保する場合

- XML Security - Core ユーザーズガイド

- アプリケーションサーバ Web サービスセキュリティ構築ガイド

XML の処理について詳細を知りたい場合

- XML Processor ユーザーズガイド

ここでは，機能の分類について，マニュアルとの対応と併せて説明します。

1.1.1 アプリケーションの実行基盤としての機能

アプリケーションとして実装されたオンライン業務やバッチ業務を実行する基盤となる機能です。システムの用途や求められる要件に応じて，使用する機能を選択します。

アプリケーションの実行基盤としての機能を使用するかどうかは，システム構築やアプリケーション開発よりも前に検討する必要があります。

アプリケーションの実行基盤としての機能について，分類ごとに説明します。

(1) アプリケーションを動作させるための基本的な機能（基本・開発機能）

アプリケーション（J2EE アプリケーション）を動作させるための基本的な機能が該当します。主に J2EE サーバの機能が該当します。

アプリケーションサーバでは，Java EE 8 に対応した J2EE サーバを提供しています。J2EE サーバでは，標準仕様に準拠した機能のほか，アプリケーションサーバ独自の機能も提供しています。

基本・開発機能は，機能を使用する J2EE アプリケーションの形態に応じて，さらに三つに分類できます。アプリケーションサーバの機能解説のマニュアルは，この分類に応じて分冊されています。

それぞれの分類の概要を説明します。

- **Web アプリケーションを実行するための機能（Web コンテナ）**
  Web アプリケーションの実行基盤である Web コンテナの機能，および Web コンテナと Web サーバが連携して実現する機能が該当します。
- **Enterprise Bean を実行するための機能（EJB コンテナ）**
  Enterprise Bean の実行基盤である EJB コンテナの機能が該当します。また，Enterprise Bean を呼び出す EJB クライアントの機能も該当します。
- **Web アプリケーションと Enterprise Bean の両方で使用する機能（コンテナ共通機能）**
  Web コンテナ上で動作する Web アプリケーションおよび EJB コンテナ上で動作する Enterprise Bean の両方で使用できる機能が該当します。

(2) Web サービスを開発するための機能

Web サービスの実行環境および開発環境としての機能が該当します。

アプリケーションサーバでは，次のエンジンを提供しています。

- JAX-WS 仕様に従った SOAP メッセージのバインディングを実現する JAX-WS エンジン
- JAX-RS 仕様に従った RESTful HTTP メッセージのバインディングを実現する JAX-RS エンジン

(3) 信頼性や性能などを向上させるために拡張されたアプリケーションサーバ独自の機能（拡張機能）

アプリケーションサーバで独自に拡張された機能が該当します。バッチサーバ，CTM，データベースなど，J2EE サーバ以外のプロセスを使用して実現する機能も含まれます。

アプリケーションサーバでは，システムの信頼性を高め，安定稼働を実現するための多様な機能が拡張されています。また，J2EE アプリケーション以外のアプリケーション（バッチアプリケーション）を Java の環境で動作させる機能も拡張しています。

(4) システムのセキュリティを確保するための機能（セキュリティ管理機能）

アプリケーションサーバを中心としたシステムのセキュリティを確保するための機能が該当します。不正なユーザからのアクセスを防止するための認証機能や，通信路での情報漏えいを防止するための暗号化機能などがあります。

1.1.2 アプリケーションの実行基盤を運用・保守するための機能

アプリケーションの実行基盤を効率良く運用したり，保守したりするための機能です。システムの運用開始後に，必要に応じて使用します。ただし，機能によっては，あらかじめ設定やアプリケーションの実装が必要なものがあります。

アプリケーションの実行基盤を運用・保守するための機能について，分類ごとに説明します。

(1) システムの起動・停止など日常的な運用をするための機能（運用機能）

システムの起動や停止，アプリケーションの開始や停止，入れ替えなどの，日常運用で使用する機能が該当します。

(2) システムの利用状況を監視するための機能（監視機能）

システムの稼働状態や，リソースの枯渇状態などを監視するための機能が該当します。また，システムの操作履歴など，監査で使用する情報を出力する機能も該当します。

(3) ほかの製品と連携して運用するための機能（連携機能）

JP1 やクラスタソフトウェアなど，ほかの製品と連携して実現する機能が該当します。

(4) トラブル発生時などに対処するための機能（保守機能）

トラブルシューティングのための機能が該当します。トラブルシューティング時に参照する情報を出力するための機能も含みます。

(5) 旧バージョンの製品から移行するための機能（移行機能）

旧バージョンのアプリケーションサーバから新しいバージョンのアプリケーションサーバに移行するための機能が該当します。

(6) 旧バージョンの製品との互換のための機能（互換機能）

旧バージョンのアプリケーションサーバとの互換用の機能が該当します。なお，互換機能については，対応する推奨機能に移行することをお勧めします。

1.1.3 機能とマニュアルの対応

アプリケーションサーバの機能解説のマニュアルは，機能の分類に合わせて分冊されています。

機能の分類と，それぞれの機能について説明しているマニュアルとの対応を次の表に示します。

表 1-1 機能の分類と機能解説のマニュアルの対応

分類	機能	マニュアル※1
基本・開発機能	Web コンテナ	基本・開発編(Web コンテナ)
	JSF および JSTL の利用	
	JAX-RS 2.1 の利用	
	WebSocket	
	NIO HTTP サーバ	
	サーブレットおよび JSP の実装	
	セッションマネージャの指定機能	
	EJB コンテナ	基本・開発編(EJB コンテナ)
	EJB クライアント	
	Enterprise Bean 実装時の注意事項	
	ネーミング管理	基本・開発編(コンテナ共通機能)
	リソース接続とトランザクション管理	
	OpenTP1 からのアプリケーションサーバの呼び出し（TP1 インバウンド連携機能）	

分類	機能	マニュアル※1
	JPA 2.2 の利用	
	CJMS プロバイダ	
	JavaMail の利用	
	アプリケーションサーバでの CDI の利用	
	アプリケーションサーバでの Bean Validation の利用	
	Java Batch	
	JSON-P	
	JSON-B	
	Concurrency Utilities	
	アプリケーションの属性管理	
	アノテーションの使用	
	J2EE アプリケーションの形式とデプロイ	
	コンテナ拡張ライブラリ	
	ライブラリ競合回避機能	
	パッケージ名変換機能	
拡張機能	バッチサーバによるアプリケーションの実行	拡張編
	CTM によるリクエストのスケジューリングと負荷分散	
	バッチアプリケーションのスケジューリング	
	J2EE サーバ間のセッション情報の引き継ぎ（セッションフェイルオーバ機能）	
	データベースセッションフェイルオーバ機能	
	明示管理ヒープ機能を使用した FullGC の抑止	
	アプリケーションのユーザログ出力	
	CORBA/OTM ゲートウェイ機能	
セキュリティ管理機能	統合ユーザ管理による認証	セキュリティ管理機能編
	アプリケーションの設定による認証	
	SSL/TLS 通信での TLSv1.2 の使用	
	API による直接接続を使用する負荷分散機の運用管理機能からの制御	
運用機能	システムの起動と停止	運用／監視／連携編
	J2EE アプリケーションの運用	
監視機能	稼働情報の監視（稼働情報収集機能）	
	リソースの枯渇監視	

分類	機能	マニュアル※1
	監査ログ出力機能	
	データベース監査証跡連携機能	
	運用管理コマンドによる稼働情報の出力	
	Management イベントの通知と Management アクションによる処理の自動実行	
	CTM の稼働統計情報の収集	
	コンソールログの出力	
連携機能	JP1 と連携したシステムの運用	
	システムの集中監視（JP1/IM との連携）	
	ジョブによるシステムの自動運転（JP1/AJS との連携）	
	クラスタソフトウェアとの連携	
	ホスト単位管理モデルを対象にした系切り替えシステム（クラスタソフトウェアとの連携）	
	1:1 系切り替えシステム（クラスタソフトウェアとの連携）	
	相互系切り替えシステム（クラスタソフトウェアとの連携）	
	N:1 リカバリシステム（クラスタソフトウェアとの連携）	
保守機能	トラブルシューティング関連機能	保守／移行編
	性能解析トレースを使用した性能解析	
	製品の JavaVM（以降，JavaVM と略す場合があります）の機能	
移行機能	旧バージョンのアプリケーションサーバからの移行	
	推奨機能への移行	
互換機能	基本・開発機能の互換機能	互換編※2
	拡張機能の互換機能	

注※1　マニュアル名称の「アプリケーションサーバ 機能解説」を省略しています。

注※2　このマニュアルです。

1.2　このマニュアルに記載している機能の説明

ここでは，このマニュアルで機能を説明するときの分類の意味と，分類を示す表の例について説明します。

1.2.1　分類の意味

このマニュアルでは，各機能について，次の五つに分類して説明しています。マニュアルを参照する目的によって，必要な個所を選択して読むことができます。

- 解説
  機能の解説です。機能の目的，特長，仕組みなどについて説明しています。機能の概要について知りたい場合にお読みください。
- 実装
  コーディングの方法や DD の記載方法などについて説明しています。アプリケーションを開発する場合にお読みください。
- 設定
  システム構築時に必要となるプロパティなどの設定方法について説明しています。システムを構築する場合にお読みください。
- 運用
  運用方法の説明です。運用時の手順や使用するコマンドの実行例などについて説明しています。システムを運用する場合にお読みください。
- 注意事項
  機能を使用するときの全般的な注意事項について説明しています。注意事項の説明は必ずお読みください。

1.2.2　分類を示す表の例

機能説明の分類については，表で説明しています。表のタイトルは，「この章の構成」または「この節の構成」となっています。

次に，機能説明の分類を示す表の例を示します。

機能説明の分類を示す表の例

表 X-1　この章の構成（○○機能）

分類	タイトル	参照先
解説	○○機能とは	X.1
実装	アプリケーションの実装	X.2

分類	タイトル	参照先
	DD および cosminexus.xml※での定義	X.3
設定	実行環境での設定	X.4
運用	○○機能を使用した運用	X.5
注意事項	○○機能使用時の注意事項	X.6

注※

cosminexus.xml については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「16. アプリケーションの属性管理」を参照してください。

ポイント

cosminexus.xml を含まないアプリケーションのプロパティ設定

cosminexus.xml を含まないアプリケーションでは，実行環境へのインポート後にプロパティを設定，または変更します。設定済みのプロパティも実行環境で変更できます。

実行環境でのアプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。サーバ管理コマンドおよび属性ファイルでのアプリケーションの設定については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「3.5.2　J2EE アプリケーションのプロパティの設定手順」を参照してください。

属性ファイルで指定するタグは，DD または cosminexus.xml と対応しています。DD または cosminexus.xml と属性ファイルのタグの対応についてはマニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「2.　アプリケーション属性ファイル（cosminexus.xml)」を参照してください。

なお，各属性ファイルで設定するプロパティは，アプリケーション統合属性ファイルでも設定できます。

1.3　アプリケーションサーバ 11-40 での主な機能変更

この節では，アプリケーションサーバ 11-40 での主な機能の変更について，変更目的ごとに説明します。

説明内容は次のとおりです。

- アプリケーションサーバ 11-40 で変更になった主な機能と，その概要を説明しています。機能の詳細については参照先の記述を確認してください。「参照先マニュアル」および「参照個所」には，その機能についての主な記載個所を記載しています。
- 「参照先マニュアル」に示したマニュアル名の「アプリケーションサーバ」は省略しています。

1.3.1　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 1-2　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Java SE 17 への対応	Java SE 17 の機能が使用できるようになりました。	システム設計ガイド	2.1
		機能解説 保守／移行編	9 章
メモリ管理方式に ZGC を追加	メモリ管理方式に ZGC を追加しました。	システム設計ガイド	7.17
接続できるデータベースに Azure SQL Database を追加	DB Connector を使用して接続できるデータベースに Azure SQL Database を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	3.6.17

2 V9 互換モードの概要

Version9 以前との互換性を重視するシステム向けの，「V9 互換モード」の概要について説明します。

2.1　V9 互換モードと推奨モード

アプリケーションサーバ 11-00 では，Servlet 3.1 や WebSocket 1.0 などの Java EE 7 新仕様の対応に当たり，09-87 以前との互換性を維持できない機能変更をしています。そのため，09-87 以前に設計されたシステムを 11-00 以降に移行する場合，幾つかの設計変更が必要です。

そこで，11-00 の新機能は使用しないで，09-87 以前との互換性を重視するシステム向けに，「V9 互換モード」を提供しています。V9 互換モードでセットアップされた J2EE サーバでは，09-87 相当の動作となり，11-00 以降で廃止された機能も一部を除き使用できます。

なお，V9 互換モードを使用しないモードは「推奨モード」と表記します。推奨モードでは，11-00 の新機能が使用できますが，11-00 以降で廃止された機能は使用できません。

2.2　インストール環境によって使用できるモード

アプリケーションサーバでは，インストール環境によって使用できるモードが異なります。インストール環境によって使用できるモードについて，次の表に示します。

表 2-1 インストール環境によって使用できるモード

インストール環境	推奨モード	V9 互換モード
JDK11 を使用する環境	○	○
JDK17 を使用する環境	○	×

（凡例）

○：使用できる

×：使用できない

3 V9 互換モードの使用方法

V9 互換モードを使用するには，次に示す二つの方法があります。

・J2EE サーバの新規作成時に V9 互換モードを指定する方法

・アプリケーションサーバ 09-87 以前で作成済みの，既存の J2EE サーバを更新インストールで移行する方法

この章では，これらの使用方法について説明します。

3.1　J2EE サーバの新規作成時に V9 互換モードを指定する方法

J2EE サーバの新規作成時に V9 互換モードを指定する方法には，次の四つの場合があります。

- Smart Composer 機能を使用する場合
- 運用管理ポータルを使用する場合
- J2EE サーバのコマンドを使用する場合
- 開発環境インスタントセットアップ機能を使用する場合

3.1.1　Smart Composer 機能を使用する場合

Smart Composer 機能を使用して，V9 互換モードの論理 J2EE サーバを含むシステムを新規作成する場合は，簡易構築定義ファイルの論理 J2EE サーバに指定するパラメタで，J2EE サーバの互換モードを設定するパラメタ「manager.j2ee.compat」の値に「V9」を指定してください。

簡易構築定義ファイルでの指定例を次に示します。

(簡易構築定義ファイルでの指定例)

```
<?xml version="1.0" encoding="UTF-8"?>
<model-definition
xmlns="http://www.cosminexus.com/mngsvr/schema/ModelDefinition-2.5">
  <web-system>
    <name>MyCompatWebSystem</name>
    <tier>
          :
      <configuration>
        <logical-server-type>j2ee-server</logical-server-type>
        <param>
          <param-name>manager.j2ee.compat</param-name>
          <param-value>V9</param-value>
        </param>
           :
```

詳細は，マニュアル「アプリケーションサーバ リファレンス定義編(サーバ定義)」の「4.11.1　J2EE サーバの互換モードを設定するパラメタ」を参照してください。

3.1.2　運用管理ポータルを使用する場合

運用管理ポータルを使用して，V9 互換モードの論理 J2EE サーバを新規作成する場合は，［運用管理ドメインの構成定義］の［J2EE サーバの追加］画面で，［互換モード］に「V9 互換モード」を指定してください。

［運用管理ドメインの構成定義］の［J2EE サーバの追加］画面を次に示します。

詳細は，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「9.9.1　J2EE サーバの追加」を参照してください。

3.1.3　J2EE サーバのコマンドを使用する場合

J2EE サーバのセットアップをする cjsetup コマンドを使用して，V9 互換モードの J2EE サーバを新規作成する場合は，cjsetup コマンドのコマンドライン引数に「-compat V9」を指定してください。

cjsetup コマンドの実行例を次に示します。

(cjsetup コマンドの実行例)

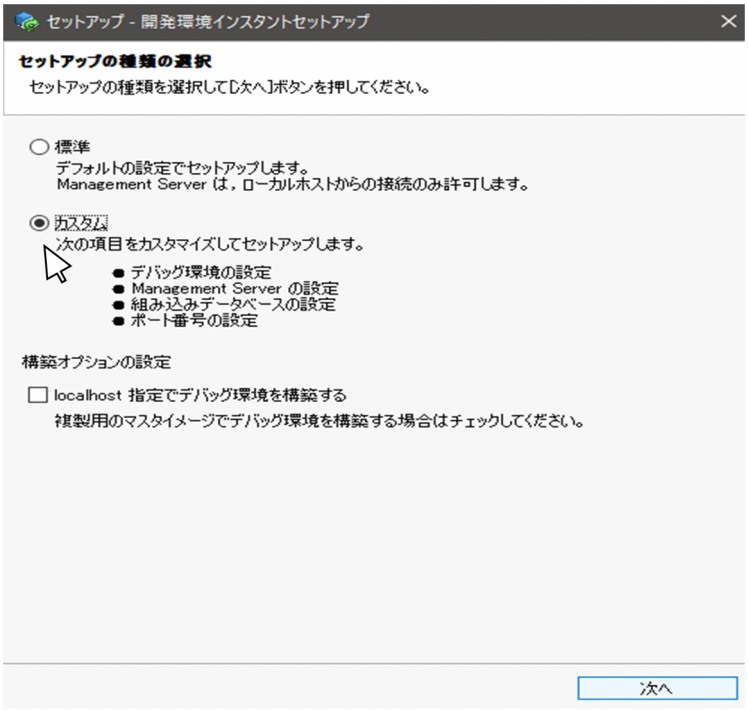
```
cjsetup MyCompatJ2EEServer -compat V9
```

詳細は，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.2　J2EE サーバを操作するコマンド」の「cjsetup（J2EE サーバのセットアップとアンセットアップ)」を参照してください。

3.1.4　開発環境インスタントセットアップ機能を使用する場合

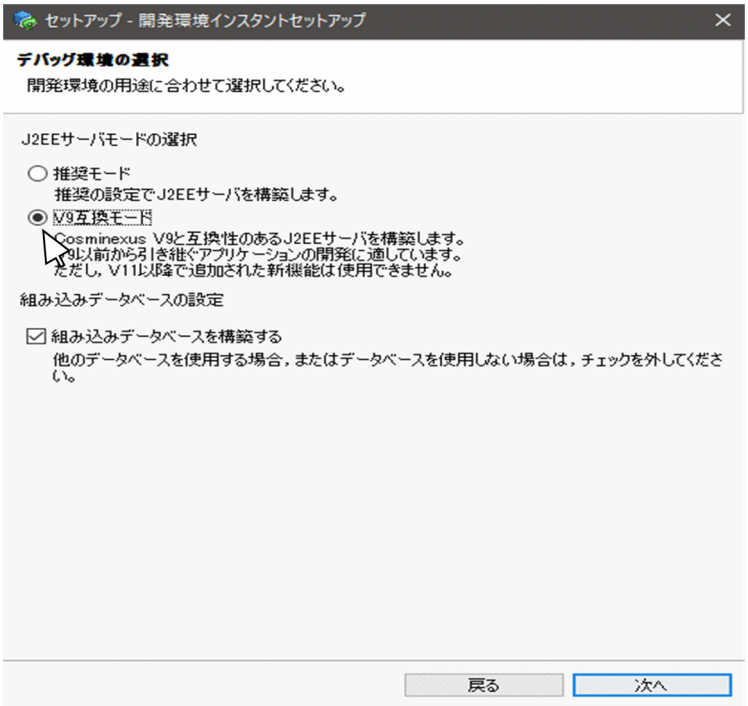
uCosminexus Developer または uCosminexus Service Architect の開発環境インスタントセットアップ機能を使用して，V9 互換モードの J2EE サーバを用いた開発環境をセットアップする場合は，次の手順で V9 互換モードを指定してください。

1. [セットアップの種類の選択] ページで「カスタム」を選択し，[次へ] をクリックします。



［デバッグ環境の選択］ページが表示されます。

2. ［デバッグ環境の選択］ページで［J2EE サーバモードの選択］の「V9 互換モード」を選択し，［次へ］をクリックします。



以降の手順は，推奨モードの場合と同じです。詳細は，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「2.3.5　デバッグ環境のカスタムセットアップ」を参照してください。

3.2　既存の J2EE サーバを更新インストールで移行する方法

09-87 以前に J2EE サーバがセットアップ済みの状態から，11-00 以降に更新インストールをした場合，セットアップ済みの既存の J2EE サーバは自動的に V9 互換モードとなります。

更新インストールによる旧バージョンからの移行作業については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「10.2.2　更新インストールの場合」を参照してください。

3.3　J2EE サーバの互換モードを確認する方法

セットアップ済みの J2EE サーバがどのモードで動作しているかは，J2EE サーバの起動時にメッセージログに出力されるメッセージ KDJE60001-I で確認できます。推奨モードの場合と V9 互換モードの場合のメッセージ KDJE60001-I を次に示します。

推奨モードの場合

```
KDJE60001-I
The J2EE server will start as 'Java EE Container Mode'. (container version = V11)
```

V9 互換モードの場合

```
KDJE60001-I
The J2EE server will start as 'Java EE Container Mode'. (container version = V9)
```

3.4　使用上の注意事項

- セットアップ済みの J2EE サーバに対して，推奨モードから V9 互換モードへの切り替えや，V9 互換モードから推奨モードへの切り替えはできません。J2EE サーバをアンセットアップしてから再度セットアップしてください。
- Smart Composer 機能や運用管理ポータルを用いて構築したシステムから論理 J2EE サーバだけを削除し，実サーバ名が同一でモードが異なる論理 J2EE サーバを作成する場合は，論理 J2EE サーバを作成する前に cjsetup コマンドを使用して，J2EE サーバをアンセットアップしてください。

  cjsetup コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.2　J2EE サーバを操作するコマンド」の「cjsetup（J2EE サーバのセットアップとアンセットアップ)」を参照してください。

4 V9 互換モードと推奨モードの機能

V9 互換モードと推奨モードの機能について説明します。

4.1　V9 互換モードと推奨モードの機能差異

アプリケーションサーバ 11-00 以降で，V9 互換モードだけで使用できる互換機能と，推奨モードだけで使用できる新機能を次に示します。ここに記載していない機能は，V9 互換モードと推奨モードの両方で使用できます。

表 4-1　各機能のモードごとの使用可否

機能名	推奨モード	V9 互換モード	記載個所
NIO HTTP サーバ機能	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「7.　NIO HTTP サーバ」
Web サーバ連携機能（リダイレクタ機能）	×	○	このマニュアルの「5.　Web サーバ連携」
インプロセス HTTP サーバ機能	×	○	このマニュアルの「6.　インプロセス HTTP サーバ」
Servlet 3.0 非同期サーブレット機能	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「8.　サーブレットおよび JSP の実装」
Servlet 3.1/Servlet 4.0	○	×	
WebSocket 1.1	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「5.　WebSocket」
Java Batch 1.0	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「11.　Java Batch」
Concurrency Utilities for Java EE 1.0	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.　Concurrency Utilities」
JSON-P 1.1	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「12.　JSON-P」
JSON-B 1.0	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.　JSON-B」
CDI 2.0	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「9.　アプリケーションサーバでの CDI の利用」
Bean Validation 2.0	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「10.　アプリケーションサーバでの Bean Validation の利用」
JSF 2.3	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「3.　JSF および JSTL の利用」
JAX-RS 2.1	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.　JAX-RS 2.1 の利用」
Cosminexus JAX-RS エンジン（JAX-RS 1.1）	×	○	このマニュアルの「7.　Cosminexus JAX-RS エンジン（JAX-RS 1.1）」

機能名	推奨モード	V9 互換モード	記載個所
JPA 2.2 対応プロバイダ	○	×	マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「6. JPA 2.2 の利用」
JPA 1.0 対応プロバイダ（CJPA プロバイダ機能）	×	○	このマニュアルの「8. CJPA プロバイダ」
HTTP レスポンス圧縮機能	×	○	このマニュアルの「9.2 HTTP レスポンス圧縮機能」
EJB 2.1 と Servlet 2.4 でのアノテーションの利用	△	△	このマニュアルの「18. 基本・開発機能の互換機能（EJB 2.1 と Servlet 2.4 でのアノテーションの利用）」
クラスタコネクションプール機能	△	△	このマニュアルの「19. クラスタコネクションプール機能を使用するための設定」
スレッドの非同期並行処理	△	△	このマニュアルの「20. スレッドの非同期並行処理」

（凡例）

○：使用できます。

△：使用できますが，互換機能のため推奨しません。

×：使用できません。

5 Web サーバ連携

この章では，Web サーバ連携に関する機能について説明します。

5.1　この章の構成

アプリケーションサーバでは，Web サーバと連携するためのリダイレクタの機能を提供しています。**リダイレクタ**とは，Web コンテナで提供しているライブラリです。リダイレクタを Web サーバに登録することで，Web サーバあての HTTP リクエストのうち特定のリクエストを，指定した Web コンテナに処理させたり，複数の Web コンテナにリクエストを振り分けて処理させたりできるようになります。

Web サーバ連携に関する機能と参照先を次の表に示します。

表 5-1　Web サーバ連携に関する機能と参照先

機能	参照先
Web サーバ（リダイレクタ）によるリクエストの振り分け	5.2
URL パターンでのリクエストの振り分け	5.3
ラウンドロビン方式によるリクエストの振り分け	5.4
POST データサイズでのリクエストの振り分け	5.5
通信タイムアウト（Web サーバ連携）	5.6
IP アドレス指定（Web サーバ連携）	5.7
エラーページのカスタマイズ（Web サーバ連携）※	5.8
ドメイン名指定でのトップページの表示	5.9
Web コンテナへのゲートウェイ情報の通知	5.10
Web コンテナ単位での同時実行スレッド数の制御	5.11
リダイレクタとの通信用オブジェクト	5.12
Explicit ヒープのチューニング	5.13

注※

Web サーバとして HTTP Server を使用する場合だけ使用できる機能です。Microsoft IIS を使用する場合，この機能は使用できません。

また，Web サーバと連携した場合，HTTP Server の SSL による認証やデータ暗号化，および Microsoft IIS の SSL による認証やデータ暗号化の機能も使用できます。HTTP Server の SSL による認証やデータ暗号化については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「7.2.2 HTTP Server の SSL の設定」を参照してください。Microsoft IIS の SSL の設定方法については，Microsoft IIS の SSL のヘルプを参照してください。なお，この機能が使用できるのは Web リダイレクタ環境でだけです。

Web サーバと連携する場合に必要な環境設定

Web サーバと連携する場合，次に示す環境設定が必要です。

- Smart Composer を使用する場合

「付録 D　HTTP Server の設定に関する注意事項」を参照して，HTTP Server の環境設定をしてください。

- **Smart Composer を使用しない場合**

  次のどちらかの Web サーバの環境を設定してください。

- HTTP Server（付録 D）
- Microsoft IIS（付録 E）

5.2　Web サーバ（リダイレクタ）によるリクエストの振り分け

この節では，リダイレクタを使用したリクエストの振り分けについて説明します。

この機能は，Web サーバ連携機能を使用する場合に使用できます。リクエストの振り分けをするには，Web サーバ/リダイレクタが動作しているホストに対して，振り分けを定義する必要があります。

この節の構成を次の表に示します。

表 5-2　この節の構成（Web サーバ（リダイレクタ）によるリクエストの振り分け）

分類	タイトル	参照先
解説	リダイレクタを使用したリクエスト振り分けの仕組み	5.2.1
	リクエストの振り分け方法を設定するユーザ定義ファイル（Smart Composer 機能を使用する場合）	5.2.2
	リクエストの振り分け方法を設定するユーザ定義ファイル（Smart Composer 機能を使用しない場合）	5.2.3
注意事項	Web サーバ連携時の注意事項	5.2.4

注　「実装」,「設定」，および「運用」について，この機能固有の説明はありません。

なお，使用する Web サーバの種類によって，必要な定義が異なります。また，Web サーバの種類が HTTP Server の場合は，使用するシステムの構築支援機能の種類によっても定義が異なります。使用する Web サーバの種類と定義を次の表に示します。

表 5-3　使用する Web サーバの種類と定義

Web サーバの種類	システムの構築支援機能の種類	定義
HTTP Server	Smart Composer 機能	• 簡易構築定義ファイルの定義 • workers.properties の定義 • mod_jk.conf の定義
	Smart Composer 機能以外	• workers.properties の定義 • mod_jk.conf の定義
Microsoft IIS	－	• workers.properties の定義 • uriworkermap.properties の定義 • isapi_redirect.conf の定義

（凡例）－：該当しない

インプロセス HTTP サーバを使用する場合のリダイレクトによるリクエストの振り分けについては，「6.7 リダイレクトによるリクエストの振り分け」を参照してください。

5.2.1　リダイレクタを使用したリクエスト振り分けの仕組み

リダイレクタを使用すると，Web サーバあての HTTP リクエストのうち，特定のリクエストを，指定した Web コンテナに処理させたり，複数の Web コンテナにリクエストを振り分けて処理させたりできます。

リダイレクタを使用してリクエストを振り分ける場合，**ワーカプロセス**※という Web サーバの背後で動作する Web コンテナの実行プロセスを使用します。ワーカプロセスは，リダイレクタを経由して，サーブレット，JSP を含むリクエストを処理するプロセスです。Web サーバとワーカプロセスとの間のデータ交換は，TCP/IP に基づき，ユーザが設定する特定のポート番号を通じて実行されます。リダイレクタの設定は，ワーカという Web コンテナを抽象化した設定単位を使用して行います。ワーカには，単体の J2EE サーバを表すワーカと，クラスタ構成の J2EE サーバを表すワーカがあります。J2EE サーバへリクエストを転送するワーカを転送先ワーカと呼びます。転送先ワーカはワーカタイプが ajp13 のワーカです。

注※

ワーカプロセスは，実際には，J2EE サーバとなります。

(1)　リクエストの転送パターン

リダイレクタからワーカプロセスへのリクエストの転送には，次に示すパターンがあります。

- 一つの Web サーバから一つのワーカプロセスへの転送
- 一つの Web サーバから複数のワーカプロセスへの転送

なお，Web サーバとワーカプロセスは，同一のマシン上にあっても，異なるマシン上にあってもかまいません。

リダイレクタからワーカプロセスへのリクエスト転送パターンを次の図に示します。

図 5-1　一つの Web サーバから一つのワーカプロセスへの転送

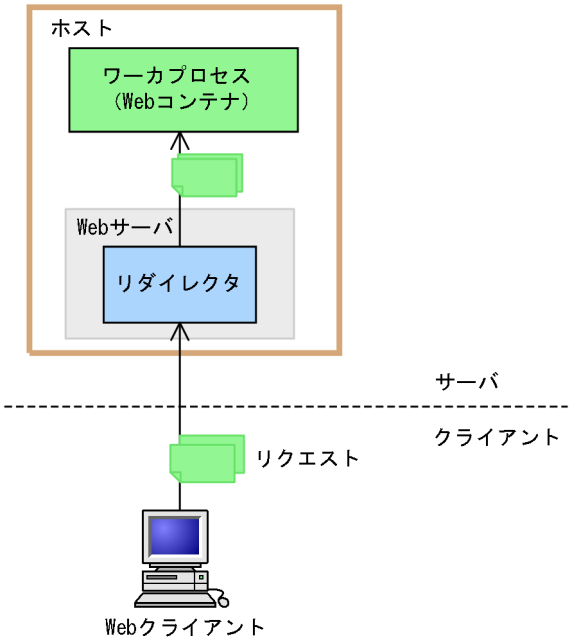
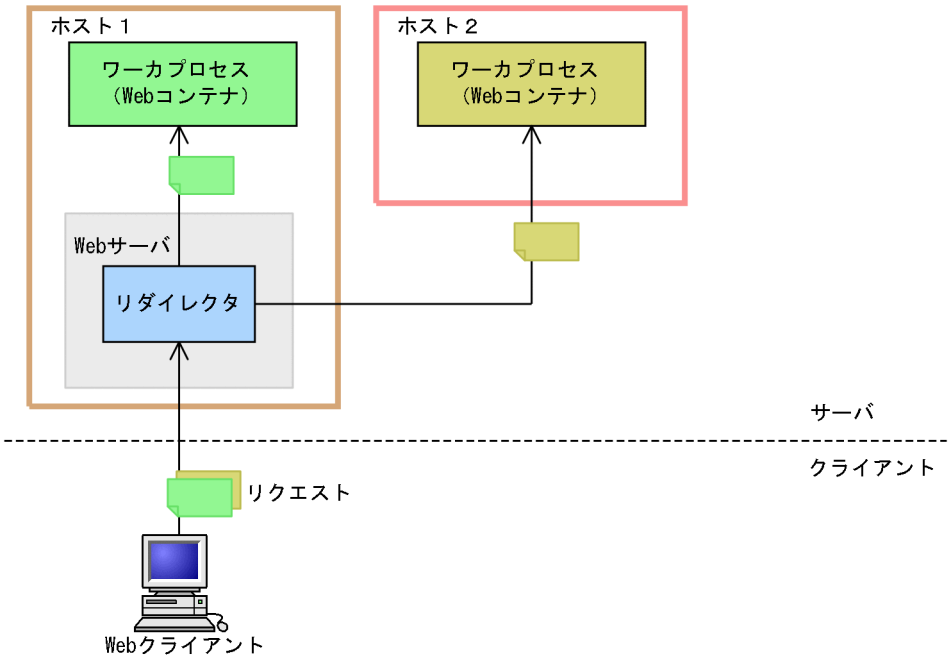


図 5-2　一つの Web サーバから複数のワーカプロセスへの転送



複数の Web コンテナへリクエストを振り分けるには，Web サーバに登録したリダイレクタに，複数の Web コンテナのワーカプロセスを振り分け先として定義します。

(2)　リクエストの振り分け方法

リダイレクタを使用してリクエストを振り分ける方法には，次の 3 種類があります。

- URL パターンによって振り分ける方法

  一つの Web コンテナに特定の処理をさせたい場合，および複数の Web コンテナに処理を振り分けたい場合に使用します。

  ワーカプロセスが一つの場合と複数の場合に使用できます。
- ロードバランサを使用してラウンドロビン方式で振り分ける方法

  複数の Web コンテナに処理を振り分けたい場合に使用します。
- POST データサイズによって振り分ける方法

  複数の Web コンテナに処理を振り分けたい場合に使用します。この振り分け方法は，Web サーバに HTTP Server を使用している場合にだけ設定できます。

  なお，次の機能を使用している場合，この振り分け方法は適用できません。

  - セッションフェイルオーバ機能
  - ラウンドロビン方式によるリクエストの振り分け

ワーカプロセスを作成するには，**ワーカ定義ファイル**と呼ばれるファイル（workers.properties）に，次の属性を定義します。

- ワーカ名

- ワーカのタイプ
- ワーカが動作しているホスト名，または IP アドレス
- ワーカが受け付けるポート番号

標準で提供される workers.properties には，あらかじめ次に示すワーカが定義されています。Web サーバと Web コンテナを同一のホスト上で動作させる場合には，これらのパラメタを変更する必要はありません。

ワーカの属性	パラメタ
ワーカ名	worker1
ワーカのタイプ	ajp13
ホスト名	localhost
ポート番号	8007

ワーカプロセスの定義方法の詳細については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

5.2.2　リクエストの振り分け方法を設定するユーザ定義ファイル（Smart Composer 機能を使用する場合）

リクエストを振り分けるためには，次のユーザ定義ファイルをテキストエディタなどで編集して，ワーカ，URL パターンとワーカのマッピング，リダイレクタの動作を設定します。

- **簡易構築定義ファイル**

  ワーカの定義，およびワーカごとのパラメタ，URL パターンとワーカのマッピングを設定します。URL パターンでのリクエスト振り分けを設定する場合に使用します。

  ラウンドロビン方式によるリクエスト振り分け，および POST データサイズでのリクエスト振り分けの場合，簡易構築定義ファイルでは設定できません。ラウンドロビン方式によるリスクエスト振り分け，および POST データサイズでのリクエスト振り分けを設定したワーカを含む環境を運用管理ポータルで構築後，cmx_export_model コマンドで簡易構築定義ファイルを出力した場合，設定内容として出力される物理ティアの種類は free-tier となります。

- **workers.properties（ワーカ定義ファイル）**

  ワーカの定義，およびワーカごとのパラメタを設定します。ラウンドロビン方式によるリクエスト振り分け，および POST データサイズでのリクエスト振り分けを設定する場合に使用します。

- **mod_jk.conf（リダイレクタ動作定義ファイル）**

  HTTP Server へのリクエストでどの URL パターンが Web コンテナへ転送されるかという URL パターンとワーカのマッピング，リダイレクタの動作を設定します。ラウンドロビン方式によるリクエスト振り分け，および POST データサイズでのリクエスト振り分けを設定する場合に使用します。

簡易構築定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。ワーカ定義ファイルの詳細については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。リダイレクタ動作定義ファイルの詳細については，「13.2.1　isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）」を参照してください。

5.2.3　リクエストの振り分け方法を設定するユーザ定義ファイル（Smart Composer 機能を使用しない場合）

リクエストを振り分けるためには，次のユーザ定義ファイルをテキストエディタなどで編集して，ワーカ，URL パターンとワーカのマッピング，リダイレクタの動作を設定します。

設定するファイルは，使用する Web サーバによって異なります。共通のユーザ定義ファイルと，Web サーバごとのユーザ定義ファイルを分けて説明します。なお，ユーザ定義ファイルのうち，httpsd.conf の詳細については，マニュアル「HTTP Server」を参照してください。ほかのユーザ定義ファイルの詳細については，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

(1)　共通のユーザ定義ファイル

HTTP Server，Microsoft IIS を使用する場合で共通のユーザ定義ファイルを次に示します。

- workers.properties（ワーカ定義ファイル）
  ワーカの定義，およびワーカごとのパラメタを設定します。
  ファイルの格納場所を次に示します。
  - Windows の場合
    <Application Server のインストールディレクトリ>¥CC¥web¥redirector¥workers.properties
  - UNIX の場合
    /opt/Cosminexus/CC/web/redirector/workers.properties
- usrconf.properties（ユーザプロパティファイル）
  リダイレクタからのリクエストを Web コンテナで受信するときの通信タイムアウトを設定します。
  ファイルの格納場所を次に示します。
  - Windows の場合
    <Application Server のインストールディレクトリ>¥CC¥server¥usrconf¥ejb¥<サーバ名称>¥usrconf.properties
  - UNIX の場合
    /opt/Cosminexus/CC/server/usrconf/ejb/<サーバ名称>/usrconf.properties

(2) HTTP Server を使用する場合のユーザ定義ファイル

HTTP Server を使用する場合のユーザ定義ファイルを次に示します。

- mod_jk.conf（リダイレクタ動作定義ファイル）
  HTTP Server でのリダイレクタの動作を設定します。
  ファイルの格納場所を次に示します。
  - Windows の場合
    <Application Server のインストールディレクトリ>¥CC¥web¥redirector¥mod_jk.conf
  - UNIX の場合
    /opt/Cosminexus/CC/web/redirector/mod_jk.conf
- httpsd.conf（HTTP Server 定義ファイル）
  HTTP Server の動作環境を定義するディレクティブ（Web サーバの実行環境を定義するパラメタ）を設定します。
  ファイルの格納場所を次に示します。
  - Windows の場合
    <Application Server のインストールディレクトリ>¥httpsd¥conf¥httpsd.conf
  - UNIX の場合
    /opt/hitachi/httpsd/conf/httpsd.conf

(3) Microsoft IIS を使用する場合のユーザ定義ファイル

Microsoft IIS を使用する場合のユーザ定義ファイルを次に示します。

- uriworkermap.properties（マッピング定義ファイル）
  Microsoft IIS での URL パターンとワーカのマッピングを設定します。
  ファイルの格納場所を次に示します。
  <Application Server のインストールディレクトリ>¥CC¥web¥redirector¥uriworkermap.properties
- isapi_redirect.conf（リダイレクタ動作定義ファイル）
  Microsoft IIS でのリダイレクタの動作を設定します。
  ファイルの格納場所を次に示します。
  <Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.conf

(4) 注意事項

ユーザ定義ファイルに関する注意事項を次に示します。

- 上書きインストールの場合，ユーザ定義ファイルは上書きされません。

- アップグレードインストールの場合のワーカ定義ファイル，および Web サーバ定義ファイルの処理については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「10. 旧バージョンのアプリケーションサーバからの移行（J2EE サーバモードの場合)」を参照してください。
- リダイレクタの定義ファイルの 1 行の最大文字数は，1,023 文字です。この文字数内で定義してください。
- 次のユーザ定義ファイルで，同じ名称のパラメタが複数指定されている場合は，最初に指定されたパラメタの値を使用して動作します。
  - isapi_redirect.conf
  - workers.properties
  - uriworkermap.properties

5.2.4 Web サーバ連携時の注意事項

Web サーバと連携するときの注意事項について説明します。

(1) Web コンテナが送受信できるリクエストヘッダおよびレスポンスヘッダの上限値

Web サーバと連携する場合，Web コンテナで送受信できるリクエストヘッダおよびレスポンスヘッダには上限があります。上限はそれぞれ 16KB です。16KB を超えるヘッダの送受信はできないので注意してください。

(2) HTTP Server を使用するときの注意事項

Web サーバ連携時は，HTTP Server の仮想ホスト機能を使用できません。

仮想ホストごとに別のリダイレクトを行いたい場合は，運用管理ポータルまたは Smart Composer を使用して，同一ホスト上に複数の HTTP Server を構築し，それぞれでリダイレクタを設定してください。

(3) Microsoft IIS を使用するときの注意事項

Microsoft IIS を使用するときの注意事項を説明します。

- Microsoft IIS で複数の Web サイトを構築している場合，これらの Web サイトと同時に連携することはできません。複数の Web サイトを構築している場合は，Web サイトごとにリダイレクタの設定をしてください。
- Microsoft IIS 用リダイレクタでは，Web コンテナに転送するリクエストについては，リクエスト URL 情報を変更します。変更したリクエスト URL は ISAPI フィルタ内で使用します。

  このため，Microsoft IIS 用リダイレクタ以降に実行される ISAPI フィルタでは，Microsoft IIS が最初に受信したリクエスト URL を取得できません。Microsoft IIS が受信したリクエスト URL を ISAPI

フィルタで取得したい場合は，ISAPI フィルタの優先順位を Microsoft IIS 用リダイレクタよりも高く設定する必要があります。なお，優先順位を調整するために，Microsoft IIS 用リダイレクタの優先順位を「中」または「低」に変更する必要がある場合は，Microsoft IIS 用リダイレクタ動作定義ファイルの filter_priority キーで指定します。filter_priority キーについては，「13.2.1　isapi_redirect.conf (Microsoft IIS 用リダイレクタ動作定義ファイル)」を参照してください。

- Microsoft IIS と連携する場合，次の HTTP リクエストヘッダは Web クライアントで指定していても，Web アプリケーションでは受信できません。
  - tomcaturl
  - tomcatquery
  - tomcatworker
  - tomcattranslate

  これらの HTTP リクエストヘッダはリダイレクタで使用されます。
- Microsoft IIS と連携する場合，POST データサイズによるリクエストの振り分けは設定できません。

5.3　URL パターンでのリクエストの振り分け

この節では，URL パターンでのリクエストの振り分けについて説明します。

HTTP リクエストに含まれる URL パターンによって，リクエストを振り分けます。これによって，特定の処理だけを Web コンテナに実行させたり，複数の Web コンテナに処理内容に応じて処理を振り分けたりできます。

この節の構成を次の表に示します。

表 5-4　この節の構成（URL パターンでのリクエストの振り分け）

分類	タイトル	参照先
解説	URL パターンでのリクエストの振り分けの概要	5.3.1
	URL パターンの種類と適用されるパターンの優先順位	5.3.2
設定	実行環境での設定（Smart Composer 機能を使用する場合）	5.3.3
	実行環境での設定（Smart Composer 機能を使用しない場合）	5.3.4

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

5.3.1　URL パターンでのリクエストの振り分けの概要

Web コンテナに転送するリクエストは，URL パターンとワーカプロセスのマッピングによって定義します。リダイレクタは設定された URL パターンに従って，リクエストを転送する Web コンテナを切り替えます。

例えば，「『/examples/』という URL を含む HTTP リクエストを Web コンテナで処理する」という定義や，「『/examples1/』という URL を含む HTTP リクエストは Web コンテナ A で，『/examples2/』という URL を含む HTTP リクエストは Web コンテナ B で処理する」などの定義ができます。

リダイレクタによるリクエストの振り分けの例を，次の図に示します。

図 5-3　リダイレクタによるリクエストの振り分け（特定のリクエストを Web コンテナに転送する場合）

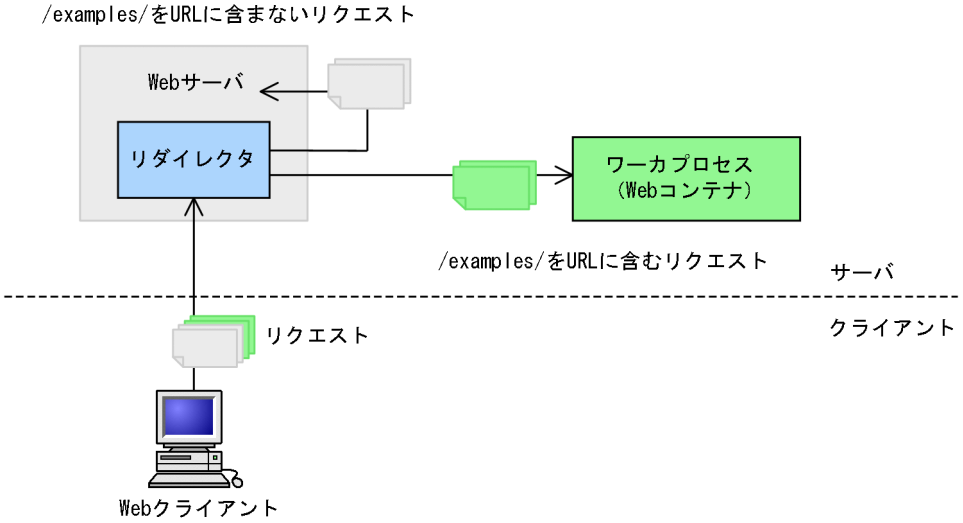
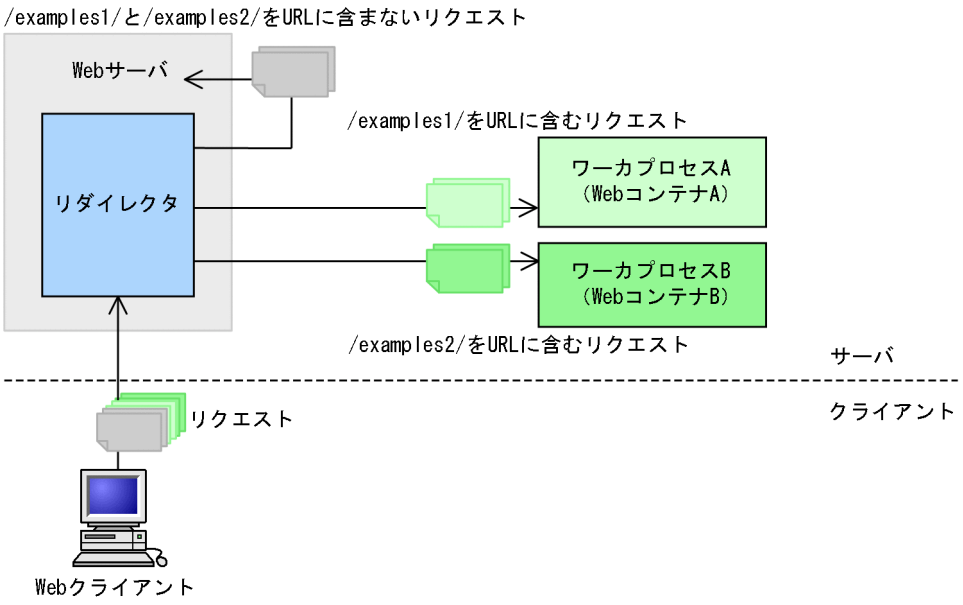


図 5-4　リダイレクタによるリクエストの振り分け（リクエストを複数の Web コンテナに振り分けて転送する場合）



5.3.2　URL パターンの種類と適用されるパターンの優先順位

リダイレクタの URL マッピングで指定できる URL パターンの種類，および適用されるパターンの優先順位について説明します。

(1)　URL パターンの種類

リダイレクタの URL マッピングには次の 4 種類の URL パターンを指定できます。

- **完全パス指定**

  完全に一致するパターンです。

  URL の形式：

  /<パス>

  ルートだけ指定する場合は，「/」。

  /<パス>に指定できる文字：

  次の文字を使用した，1 文字以上の文字列を指定します。

  半角英数字，「/」，「*」，「-」，「.」，「_」，「~」，「!」，「$」，「&」，「'」，「(」，「)」，「+」，「,」，「=」，「:」，「@」

  例：

  URL パターンが"/examples/jsp/index.jsp"で，URL が"/examples/jsp/index.jsp"の場合，一致となります。

- **パス指定**

  パスが一致するパターンです。

  URL の形式：

  /<パス>/*

  すべてのリクエストを対象にする場合の形式は，「/*」。

  /<パス>に指定できる文字：

  次の文字を使用した，1 文字以上の文字列を指定します。

  半角英数字，「/」，「*」，「-」，「.」，「_」，「~」，「!」，「$」，「&」，「'」，「(」，「)」，「+」，「,」，「=」，「:」，「@」

  例：

  URL パターンが"/examples/*"で，URL が"/examples/jsp/index.jsp"などの場合，一致となります。

- **拡張子指定**

  拡張子が一致するパターンです。指定されたパス以下のすべての階層に適用されます。

  URL の形式：

  /<パス>/*.<拡張子>

  すべてのパスを対象にする場合の形式は，「/*.<拡張子>」。

  <パス>，および<拡張子>に指定できる文字：

  次の文字を使用した，1 文字以上の文字列を指定します。

  半角英数字，「/」，「*」，「-」，「.」，「_」，「~」，「!」，「$」，「&」，「'」，「(」，「)」，「+」，「,」，「=」，「:」，「@」

  例：

  URL パターンが"/examples/*.jsp"で，URL が"/examples/jsp/index.jsp"などの場合，一致となります。

- **サフィックス指定**

  サフィックスが一致するパターンです。指定されたパス以下のすべての階層に適用されます。

  URL の形式：

  /<パス>/*<サフィックス>

  すべてのパスを対象にする場合の形式は，「/*<サフィックス>」。

  <パス>，および<サフィックス>に指定できる文字：

  次の文字を使用した，1 文字以上の文字列を指定します。

  半角英数字，「/」，「*」，「-」，「.」，「_」，「~」，「!」，「$」，「&」，「'」，「(」，「)」，「+」，「,」，「=」，「:」，「@」

  例：

  URL パターンが"/examples/servlet/*Servlet"で，URL が"/examples/servlet/HelloServlet"などの場合，一致となります。

  注意事項

  URL パターンの指定に関する注意事項を次に示します。

  - URL パターンの先頭に「/」以外を指定しないでください。URL パターンの先頭に「/」以外を指定した場合，Windows では KDJE41012-E が出力され，そのマッピングは無視されます。それ以外の OS では，メッセージが出力されて，HTTP Server の開始に失敗します。
  - 「*」をワイルドカードとして使用する場合，URL パターンの「/*」より前には指定できません。URL パターン内で最初の「*」の直前の文字が「/」ではない場合，その URL パターンは「完全パス指定」として扱われます。その場合，URL 内に「/*」となる個所があっても，ワイルドカードとして扱われません。
  - 同じ URL パターンのマッピングを複数記述しないでください。複数記述した場合，次のような動作となります。

    「完全パス指定」，および「パス指定」の場合，先に書いた URL パターンのマッピングが有効となります。「拡張子指定」，または「サフィックス指定」の場合は，あとに書いたものが有効となります。
  - <パス>，<拡張子>，<サフィックス>には，指定できる文字以外の文字を使用しないでください。指定できる文字以外の文字が URL パターンに含まれる場合，文字の種類によっては Web コンテナに転送されないことがあります。
  - <拡張子>または<サフィックス>の文字列の長さは 1 文字以上で指定してください。1 文字以上でない場合，「拡張子指定」では KDJE41041-W が出力され，そのマッピングは無視されます。「サフィックス指定」では，「パス指定」の URL パターンとして扱われます。

(2) 適用される URL パターンの優先順位

四つの URL パターンへのマッピングのうち，最優先される URL パターンは，「完全パス指定」です。「完全パス指定」に一致しない場合，次に示す順序でパスの一致が判定され，適用される URL パターンが決定します。

1. 「完全パス指定」に一致しない場合

   「パス指定」「拡張子指定」「サフィックス指定」のうち，最長一致のものが適用されます。最長一致とは，先頭(「/」)から「*」の上位パスまでができるだけ長いものに一致することを示します。

   次のように二つのマッピングが定義されている場合を例に，説明します。

   マッピング定義：

   /examples/* worker1

   /examples/jsp/* worker2

   この場合，URL が"/examples/jsp/index.jsp"の場合には worker2 のマッピングが適用され，URL が"/examples/test/index.jsp"の場合には worker1 のマッピングが適用されます。

2. 1.の条件に加えて，最長一致する「パス指定」「拡張子指定」「サフィックス指定」が複数存在する場合

   「パス指定」より，「拡張子指定」または「サフィックス指定」が優先されます。

   次のように二つのマッピングが定義されている場合を例に，説明します。

   マッピング定義：

   /examples/jsp/* worker1

   /examples/jsp/*.jsp worker2

   この場合，URL が"/examples/jsp/index.jsp"の場合には worker2 のマッピングが適用され，URL が"/examples/jsp/test.html"の場合には worker1 のマッピングが適用されます。

3. 1.および 2.の条件に加えて，最長一致する「拡張子指定」「サフィックス指定」が複数存在する場合

   あとに指定した URL パターンが優先されます。

   次のように二つのマッピングが定義されている場合を例に，説明します。

   マッピング定義：

   /examples/*.jsp worker1

   /examples/*jsp worker2

   この順番に指定されていた場合，URL が"/examples/jsp/index.jsp"の場合には worker2 のマッピングが適用されます。

注意事項

適用されるパターンの優先順位の判定に関する注意事項を次に示します。

- リクエスト URL にクエリ（URL の「?」以降の文字列）がある場合，その部分は URL パターンとの比較には使用されません。

  例：

リクエスト URL が"/examples/jsp/index.jsp?query=foo"の場合，比較に使用される URL は"/examples/jsp/index.jsp"となります。

- リクエスト URL にパスパラメタ（URL の「;」以降の文字列）がある場合，その部分は URL パターンとの比較には使用されません。

  例：

  リクエスト URL が"/examples/jsp/index.jsp;jsessionid=0000"の場合，比較に使用される URL は"/examples/jsp/index.jsp"となります。
- リクエスト URL はパスの正規化をしてから，URL パターンとの一致が判定されます。

  例：

  リクエスト URL が"/examples/../examples/./jsp//index.jsp"の場合，比較に使用される URL は"/examples/jsp/index.jsp"となります。
- URL パターンの正規化はしません。そのため，「./」，「../」などを含む URL パターンを定義しても，リクエストと一致することはありません。
- Windows の場合，「拡張子指定」の URL パターンの拡張子について，大文字小文字は区別されないで判定されます。

5.3.3　実行環境での設定（Smart Composer 機能を使用する場合）

HTTP リクエストに含まれる URL パターンによってリクエストを振り分けることで，特定の処理だけを Web コンテナに実行させたり，複数の Web コンテナに処理内容に応じて振り分けたりできます。なお，URL パターンによる振り分けは，原則として Web アプリケーション単位で振り分けます。URL パターンは，リダイレクタの動作として定義します。

(1)　設定の手順

URL パターンによるリクエストの振り分けは，次の手順で設定します。

1. 簡易構築定義ファイルで，ワーカ，および URL パターンとワーカのマッピングを定義します。

   ワーカ名のリスト，ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名などを設定します。

   また，すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。

2. Web サーバの環境を設定して，Web サーバを再起動します。

   Web サーバの設定については，「付録 D　HTTP Server の設定に関する注意事項」を参照してください。

(2)　簡易構築定義ファイルでの設定

URL パターンによるリクエストの振り分けの定義は，簡易構築定義ファイルで，論理 Web サーバ（web-server）の<configuration>タグ内に定義します。

簡易構築定義ファイルでの URL パターンによるリクエストの振り分けの定義について次の表に示します。

表 5-5　簡易構築定義ファイルでの URL パターンによるリクエストの振り分けの定義

分類	パラメタ	説明
ワーカの定義	worker.list	一つまたは複数のワーカ名のリストを指定します。
	worker.<ワーカ名>.host	ワーカのホスト名，または IP アドレスを指定します。
	worker.<ワーカ名>.port	ワーカのポート番号を指定します。
	worker.<ワーカ名>.type	ワーカのタイプ（ajp13，lb または post_size_lb）を指定します。
	worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカのコネクション数を指定します。Windows の場合にだけ指定できます。
	worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。
	worker.<ワーカ名>.delegate_error_code	エラーページの作成を Web サーバに委任する場合に利用するエラーステータスコードを指定します。
URL パターンとワーカのマッピングの定義	JkMount	URL パターンと，worker.list で指定されているワーカのどれかとの組み合わせを指定します。

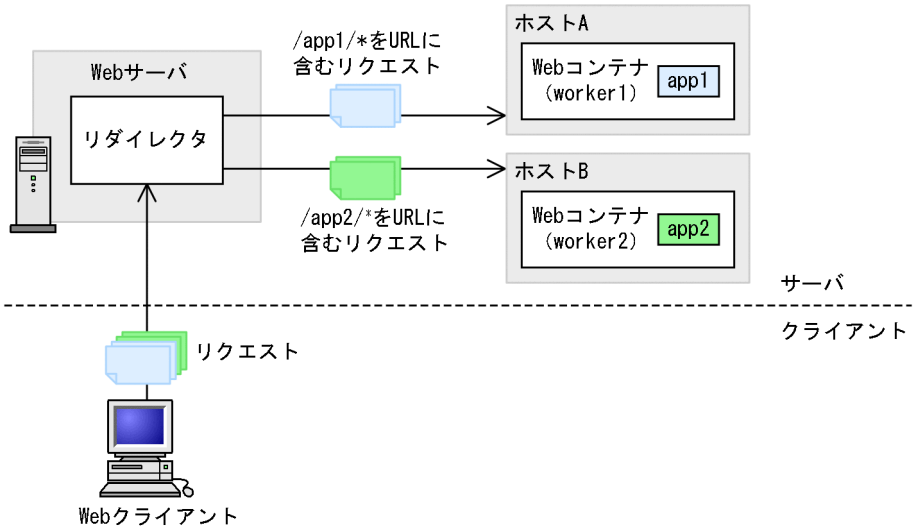
注　<ワーカ名>には，worker.list パラメタで指定したワーカの名称を定義します。

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

(3)　設定例

URL パターンでのリクエスト振り分けの例を次の図に示します。

図 5-5　URL パターンでのリクエスト振り分けの例



この例では，ホスト A には Web アプリケーション「app1」，ホスト B には Web アプリケーション「app2」がそれぞれデプロイされています。処理したい Web アプリケーション名をリクエストの URL パターンに含めることで，「/app1/*」という URL パターンを持つリクエストはホスト A で，「/app2/*」という URL パターンを持つリクエストはホスト B で処理できるようにします。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。

簡易構築定義ファイルの例を次に示します。ここでは，複数の Web コンテナへリクエストを振り分けるので，Web サーバに登録したリダイレクタに，複数の Web コンテナのワーカを振り分け先として指定します。また，「/app1/*」という URL パターンと「worker1」，「/app2/*」という URL パターンと「worker2」とをマッピングします。

なお，この構成を実現するためには，複数の Web システムに分割して構築する必要があります。

簡易構築定義ファイルの例

```
：
<param>
  <param-name>JkMount</param-name>
  <param-value>/app1/* worker1</param-value>
  <param-value>/app2/* worker2</param-value>
</param>
<param>
  <param-name>worker.list</param-name>
  <param-value>worker1,worker2</param-value>
</param>
<param>
  <param-name>worker.worker1.port</param-name>
  <param-value>8007</param-value>
</param>
<param>
  <param-name>worker.worker1.host</param-name>
  <param-value>hostA</param-value>
</param>
<param>
  <param-name>worker.worker1.type</param-name>
  <param-value>ajp13</param-value>
</param>
<param>
  <param-name>worker.worker1.cachesize</param-name>
  <param-value>64</param-value>
</param>
<param>
  <param-name>worker.worker2.port</param-name>
  <param-value>8007</param-value>
</param>
<param>
  <param-name>worker.worker2.host</param-name>
  <param-value>hostB</param-value>
</param>
<param>
  <param-name>worker.worker2.type</param-name>
  <param-value>ajp13</param-value>
</param>
<param>
```

```
  <param-name>worker.worker2.cachesize</param-name>
  <param-value>64</param-value>
</param>
 :
```

5.3.4　実行環境での設定（Smart Composer 機能を使用しない場合）

HTTP リクエストに含まれる URL パターンによってリクエストを振り分けることで，特定の処理だけを Web コンテナに実行させたり，複数の Web コンテナに処理内容に応じて振り分けたりできます。なお，URL パターンによる振り分けは，原則として Web アプリケーション単位で振り分けます。URL パターンは，リダイレクタの動作として定義します。

(1)　設定の手順

URL パターンによるリクエストの振り分けは，次の手順で設定します。

1. workers.properties でワーカを定義します。

   ワーカ名のリスト，ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名などを設定します。

   標準で提供される workers.properties には，デフォルト値が定義されています。コメントとして定義されているデフォルトの定義を使用する場合は，該当する行の先頭の「#」を削除してください。

   workers.properties（ワーカ定義ファイル）については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

2. HTTP Server を使用する場合は，mod_jk.conf で URL パターンとワーカのマッピングを定義します。Microsoft IIS を使用する場合は，uriworkermap.properties で URL パターンとワーカのマッピングを定義します。

   すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

   uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）については，「13.2.3 uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）」を参照してください。

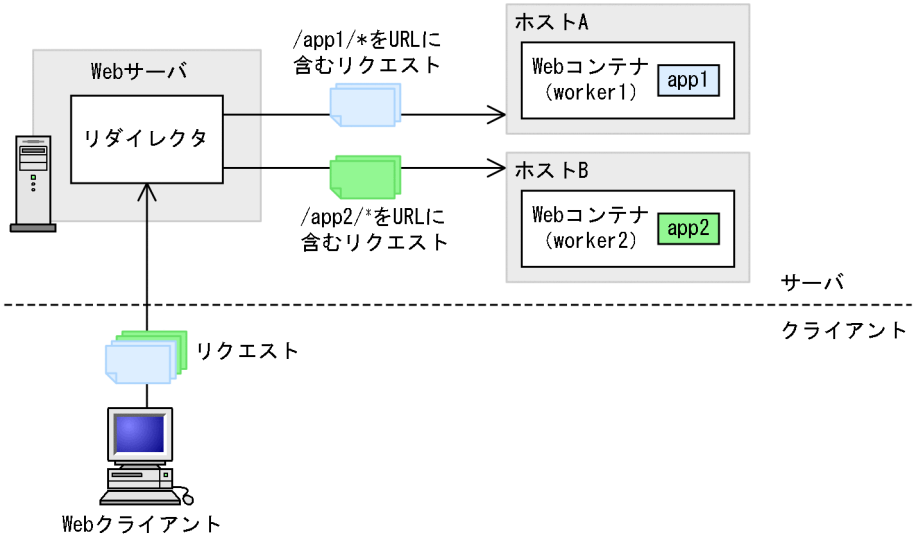
3. Web サーバの環境を設定して，Web サーバを再起動します。

   Web サーバの設定については，「付録 D　HTTP Server の設定に関する注意事項」，または「付録 E Microsoft IIS の設定」を参照してください。

(2)　設定例

URL パターンでのリクエスト振り分けの例を次の図に示します。

図 5-6　URL パターンでのリクエスト振り分けの例



この例では，ホスト A には Web アプリケーション「app1」，ホスト B には Web アプリケーション「app2」がそれぞれデプロイされています。処理したい Web アプリケーション名をリクエストの URL パターンに含めることで，「/app1/*」という URL パターンを持つリクエストはホスト A で，「/app2/*」という URL パターンを持つリクエストはホスト B で処理できるようにします。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。

workers.properties の例を次に示します。ここでは，複数の Web コンテナへリクエストを振り分けるので，Web サーバに登録したリダイレクタに，複数の Web コンテナのワーカを振り分け先として指定します。

workers.properties の例（Windows の場合）

```
worker.list=worker1,worker2

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
```

workers.properties の例（UNIX の場合）

```
worker.list=worker1,worker2

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
```

mod_jk.conf および uriworkermap.properties の例を次に示します。ここでは,「/app1/*」という URL パターンと「worker1」,「/app2/*」という URL パターンと「worker2」とをマッピングします。

mod_jk.conf の例（HTTP Server の場合）

```
JkMount /app1/* worker1
JkMount /app2/* worker2
```

uriworkermap.properties の例（Microsoft IIS の場合）

```
/app1/*=worker1
/app2/*=worker2
```

5.4　ラウンドロビン方式によるリクエストの振り分け

この節では，ラウンドロビン方式によるリクエストの振り分けについて説明します。

この節の構成を次の表に示します。

表 5-6　この節の構成（ラウンドロビン方式によるリクエストの振り分け）

分類	タイトル	参照先
解説	ラウンドロビン方式によるリクエストの振り分けの概要	5.4.1
	ラウンドロビン方式によるリクエストの振り分けの例	5.4.2
	ラウンドロビン方式によるリクエストの振り分けの定義	5.4.3
設定	実行環境での設定（Smart Composer 機能を使用する場合）	5.4.4
	実行環境での設定（Smart Composer 機能を使用しない場合）	5.4.5
注意事項	ラウンドロビン方式によるリクエストの振り分けに関する注意事項	5.4.6

注 「実装」および「運用」について，この機能固有の説明はありません。

Web コンテナがクラスタ構成で配置されている場合，リダイレクタを使用して，ラウンドロビン方式でそれぞれの Web コンテナにリクエストを振り分けられます。リダイレクタは，各リクエストに付けられたセッション ID を参照することで，同一の Web クライアントから来たリクエストが常に同一の Web コンテナへ転送されるように，リクエストを振り分けます。ただし，セッションのクッキー名をデフォルトの「JSESSIONID」からほかの名称に変更した場合，同一の Web クライアントから来たリクエストを同一の Web コンテナに転送する動作は保証されません。

また，振り分け先の Web コンテナの処理性能が異なる場合などには，負荷パラメタを定義することで，ホストごとの負荷の割合を調整することもできます。この方式でリクエストを振り分ける場合は，振り分け処理をする各 Web コンテナに，それぞれ同じ Web アプリケーションがデプロイされていることが前提になります。

5.4.1　ラウンドロビン方式によるリクエストの振り分けの概要

クラスタ構成でのラウンドロビン方式によるリクエスト振り分けには，**ロードバランサ**というワーカ定義を使用します。ロードバランサには，振り分け先となるワーカプロセスのリストが定義されています。この定義を基に，ワーカプロセスに対するラウンドロビン方式の振り分けが実行されます。

振り分けは HTTP リクエスト単位で実行されます。ただし，同じセッションに属する HTTP リクエストは，前回の振り分け先と同じワーカに振り分けられます。

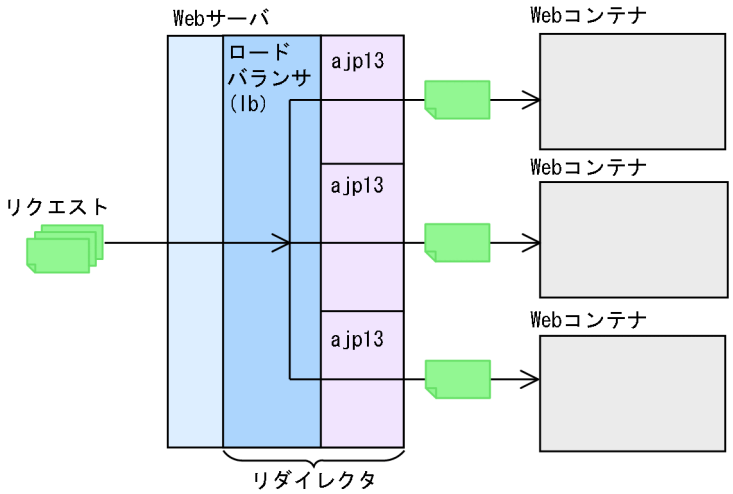
参考

Web サーバ連携時に，リダイレクタの設定でラウンドロビン方式によるリクエストの振り分けを指定すると，HttpSession のセッション ID にワーカ名が付加されます。また，サーバ ID を付加するかどうかの設定に関係なく，HttpSession のセッション ID にサーバ ID は付加されません。

5.4.2　ラウンドロビン方式によるリクエストの振り分けの例

ロードバランサを使用したリクエストの振り分けの例を次の図に示します。

図 5-7　ロードバランサを使用したリクエストの振り分けの例



5.4.3　ラウンドロビン方式によるリクエストの振り分けの定義

標準で提供される workers.properties には，あらかじめ次に示すロードバランサが定義されています。

```
#worker.list=loadbalancer1

#worker.loadbalancer1.type=lb
#worker.loadbalancer1.balanced_workers=worker1,worker2
```

worker.loadbalancer1.type では，ワーカのタイプを設定します。
worker.loadbalancer1.balanced_workers では，振り分け対象となるワーカプロセス名を設定します。
workers.properties には，loadbalancer1 として，それぞれ「lb」と「worker1，worker2」が定義されています。

ただし，この定義はコメントとして記述されています。このため，上記のロードバランサを利用する場合は，workers.properties の該当する行の先頭の「#」を削除してください。

リクエスト振り分けの比率は，振り分け対象となる各ワーカ定義の lbfactor パラメタに定義できます。lbfactor が大きければ大きいほどそのワーカプロセスに転送されるリクエストの割合は大きくなります。

例えば，worker1 と worker2 という二つのワーカプロセスでリクエストを振り分ける場合，ワーカ定義の lbfactor パラメタに次に示すように定義されているとします。

- worker1 の lbfactor パラメタ：2.0
- worker2 の lbfactor パラメタ：1.0

この場合，worker1 は worker2 の 2 倍の数の Web クライアントを担当することになります。

5.4.4　実行環境での設定（Smart Composer 機能を使用する場合）

振り分け先となるワーカのリストをロードバランサに定義することで，ラウンドロビン方式でワーカにリクエストを振り分けます。

振り分け先となる各ワーカに負荷分散値を設定して，リクエスト振り分けの比率を定義すると，ホストごとの負荷の割合を調整できます。リダイレクタは，この比率で HTTP リクエスト単位にラウンドロビンでリクエストを振り分けるので，比率が高いワーカほど転送されるリクエストの割合が多くなります。ただし，同じセッションに属する HTTP リクエストは前回と同じワーカに振り分けられます。

(1)　設定の手順

ラウンドロビン方式によるリクエスト振り分けは，次の手順で設定します。

1. workers.properties でロードバランサ，およびワーカを定義します。

   ロードバランサの定義

   ワーカ名のリスト，ワーカのタイプ（「lb」を設定），負荷分散の対象となるワーカのリストなどを設定します。

   各ワーカの定義

   ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名，負荷分散値などを設定します。

   標準で提供される workers.properties には，デフォルト値が定義されています。コメントとして定義されているデフォルトの定義を使用する場合は，該当する行の先頭の「#」を削除してください。

   workers.properties（ワーカ定義ファイル）については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

2. mod_jk.conf で URL パターンとワーカのマッピングを定義します。

   すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

3. Web サーバの環境を設定して，Web サーバを再起動します。

Web サーバの設定については，「付録 D　HTTP Server の設定に関する注意事項」を参照してください。

(a)　注意事項

- リダイレクタで負荷分散を使用する場合に，あるワーカで障害を検出すると，障害を検出してから 60 秒間はそのワーカはリダイレクト先ワーカの選択肢から除外されます。そのため，障害が回復しても，最大で 60 秒間はそのワーカが使用されない場合があります。
- UNIX の場合，HTTP Server のサーバプロセスを負荷に応じて生成・消滅させているときは，workers.properties の最初に定義したワーカによって多く割り振られます。また，サーバプロセス数を固定にしても，リクエストがどのサーバプロセスに割り振られるかは不定のため，同じ負荷分散値を指定してもラウンドロビンにならないことがあります。サーバプロセスは，消滅しなければ負荷に応じて増えてもよいため，サーバプロセスが短時間で生成・消滅しないようにディレクティブを指定する必要があります。

  HTTP Server の httpsd.conf のディレクティブが次に示す条件を満たすように，指定してください。

条件	意味
MaxSpareServers ≧ MaxClients	サーバプロセスは MaxClients まで増加し，処理終了後も常駐します。
MaxRequestsPerChild 10000	HTTP リクエストを 10,000 回処理後，リフレッシュのためにサーバプロセスを終了させます（10,000 は推奨値）。振り分け先の J2EE サーバ数に対して十分に大きな値を指定します。
StartServers = MaxClients	最初に全サーバプロセスを起動しておく場合に指定します。

(b)　ディレクティブの指定例

```
StartServers 256
MaxClients 256
MaxSpareServers 256
MaxRequestsPerChild 10000
```

(2)　workers.properties および mod_jk.conf での設定

ラウンドロビン方式によるリクエストの振り分けの設定は，workers.properties および mod_jk.conf で定義します。workers.properties および mod_jk.conf で定義するキーを次の表に示します。

表 5-7　workers.properties および mod_jk.conf で定義するキー（ラウンドロビン方式によるリクエストの振り分けの場合）

ファイルの種類	キー名	説明
workers.properties	worker.list	一つまたは複数のワーカ名のリストを指定します。
	worker.<ワーカ名>.host	ワーカのホスト名，または IP アドレスを指定します。
	worker.<ワーカ名>.port	ワーカのポート番号を指定します。

ファイルの種類	キー名	説明
	worker.<ワーカ名>.type	ワーカのタイプを指定します。ロードバランサには「lb」を，負荷分散の対象となるワーカには「ajp13」を指定します。
	worker.<ワーカ名>.balanced_workers	負荷分散の対象となるワーカのリストを指定します。
	worker.<ワーカ名>.lbfactor	負荷分散値を指定します。
	worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカのコネクション数を指定します。 Windows の場合にだけ指定できます。
	worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。
	worker.<ワーカ名>.delegate_error_code	エラーページの作成を Web サーバに委任する場合に利用するエラーステータスコードを指定します。
mod_jk.conf	JkMount	URL パターンと，worker.list で指定されているワーカのどれかとの組み合わせを指定します。

注　<ワーカ名>には，worker.list キー，または worker.<ワーカ名>.balanced_workers キーで指定したワーカの名称を定義します。

ワーカの種類ごとに指定できるパラメタを次の表に示します。

表 5-8　ワーカの種類ごとに指定できるキー

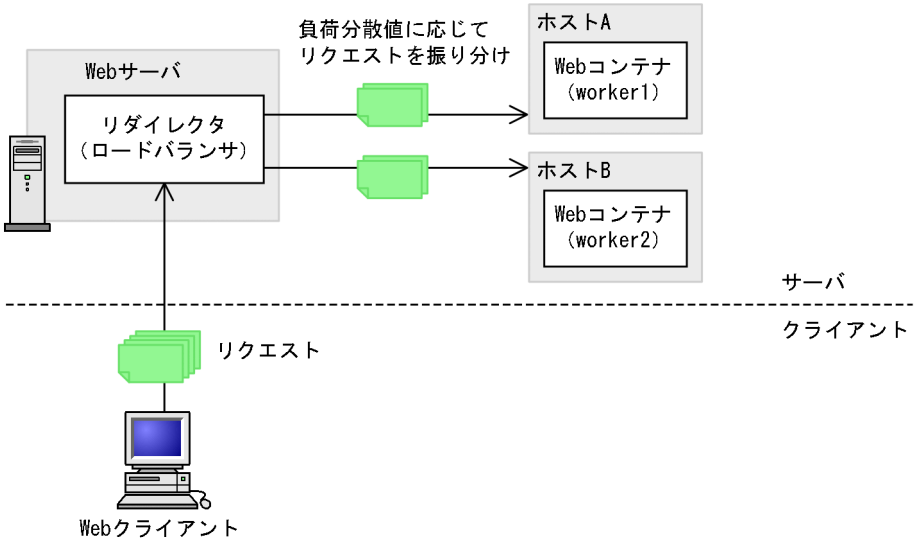
キー名	ワーカの種類（worker.<ワーカ名>.type キーの指定値）	
	ロードバランサ （「lb」を指定）	ワーカ （「ajp13」を指定）
worker.<ワーカ名>.host	×	○
worker.<ワーカ名>.port	×	○
worker.<ワーカ名>.type	○	○
worker.<ワーカ名>.balanced_workers	○	×
worker.<ワーカ名>.lbfactor	×	△
worker.<ワーカ名>.cachesize	×	△
worker.<ワーカ名>.receive_timeout	×	△
worker.<ワーカ名>.delegate_error_code	×	△

（凡例）○：指定できる　×：指定できない　△：任意に指定できる

(3)　設定例

ラウンドロビン方式によるリクエスト振り分けの例を次の図に示します。

図 5-8　ラウンドロビン方式によるリクエスト振り分けの例



この例では，「/examples」以下のリクエストがホスト A，ホスト B に同じ比率で振り分けます。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。

workers.properties の例を次に示します。ここでは，ロードバランサとワーカを定義します。リクエストを同じ比率で振り分けるので，「worker1」と「worker2」の負荷分散値はどちらも「1」を指定します。

workers.properties の例（Windows の場合）

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1,worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.lbfactor=1
```

workers.properties の例（UNIX の場合）

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1,worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.lbfactor=1
```

```
worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.lbfactor=1
```

mod_jk.conf の例を次に示します。ここでは，ロードバランサ名「loadbalancer1」を指定します。

mod_jk.conf の例

```
JkMount /examples/* loadbalancer1
```

5.4.5　実行環境での設定（Smart Composer 機能を使用しない場合）

振り分け先となるワーカのリストをロードバランサに定義することで，ラウンドロビン方式でワーカにリクエストを振り分けます。

振り分け先となる各ワーカに負荷分散値を設定して，リクエスト振り分けの比率を定義すると，ホストごとの負荷の割合を調整できます。リダイレクタは，この比率で HTTP リクエスト単位にラウンドロビンでリクエストを振り分けるので，比率が高いワーカほど転送されるリクエストの割合が多くなります。ただし，同じセッションに属する HTTP リクエストは前回と同じワーカに振り分けられます。

(1)　設定の手順

ラウンドロビン方式によるリクエスト振り分けは，次の手順で設定します。

1. workers.properties でロードバランサ，およびワーカを定義します。

   ロードバランサの定義

   ワーカ名のリスト，ワーカのタイプ（「lb」を設定），負荷分散の対象となるワーカのリストなどを設定します。

   各ワーカの定義

   ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名，負荷分散値などを設定します。

   標準で提供される workers.properties には，デフォルト値が定義されています。コメントとして定義されているデフォルトの定義を使用する場合は，該当する行の先頭の「#」を削除してください。

   workers.properties（ワーカ定義ファイル）については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

2. HTTP Server を使用する場合は，mod_jk.conf で URL パターンとワーカのマッピングを定義します。Microsoft IIS を使用する場合は，uriworkermap.properties で URL パターンとワーカのマッピングを定義します。

   すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）については，「13.2.3 uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）」を参照してください。

3. Web サーバの環境を設定して，Web サーバを再起動します。

Web サーバの設定については，「付録 D　HTTP Server の設定に関する注意事項」，または「付録 E Microsoft IIS の設定」を参照してください。

(a)　注意事項

- リダイレクタで負荷分散を使用する場合に，あるワーカで障害を検出すると，障害を検出してから 60 秒間はそのワーカはリダイレクト先ワーカの選択肢から除外されます。そのため，障害が回復しても，最大で 60 秒間はそのワーカが使用されない場合があります。
- Microsoft IIS を使用してリダイレクタが動作するワーカプロセスを複数に設定した場合，ワーカを 2 以上に設定していると，最初のリダイレクト先が workers.properties の最初に定義したワーカに多く割り振られることがあります。

  また，リクエストがどのワーカプロセスに割り振られるかは Microsoft IIS の制御によるため，同じ負荷分散値を指定してもラウンドロビンにならないことがあります。

  この場合，アプリケーションプールの数を一つにすることで，最初のリダイレクトの割り振り先もラウンドロビンになります。
- UNIX の場合，HTTP Server のサーバプロセスを負荷に応じて生成・消滅させているときは，workers.properties の最初に定義したワーカによって多く割り振られます。また，サーバプロセス数を固定にしても，リクエストがどのサーバプロセスに割り振られるかは不定のため，同じ負荷分散値を指定してもラウンドロビンにならないことがあります。サーバプロセスは，消滅しなければ負荷に応じて増えてもよいため，サーバプロセスが短時間で生成・消滅しないようにディレクティブを指定する必要があります。

  HTTP Server の httpsd.conf のディレクティブが次に示す条件を満たすように，指定してください。

条件	意味
MaxSpareServers ≧ MaxClients	サーバプロセスは MaxClients まで増加し，処理終了後も常駐します。
MaxRequestsPerChild 10000	HTTP リクエストを 10,000 回処理後，リフレッシュのためにサーバプロセスを終了させます（10,000 は推奨値）。振り分け先の J2EE サーバ数に対して十分に大きな値を指定します。
StartServers = MaxClients	最初に全サーバプロセスを起動しておく場合に指定します。

(b)　ディレクティブの指定例

```
StartServers 256
MaxClients 256
MaxSpareServers 256
MaxRequestsPerChild 10000
```

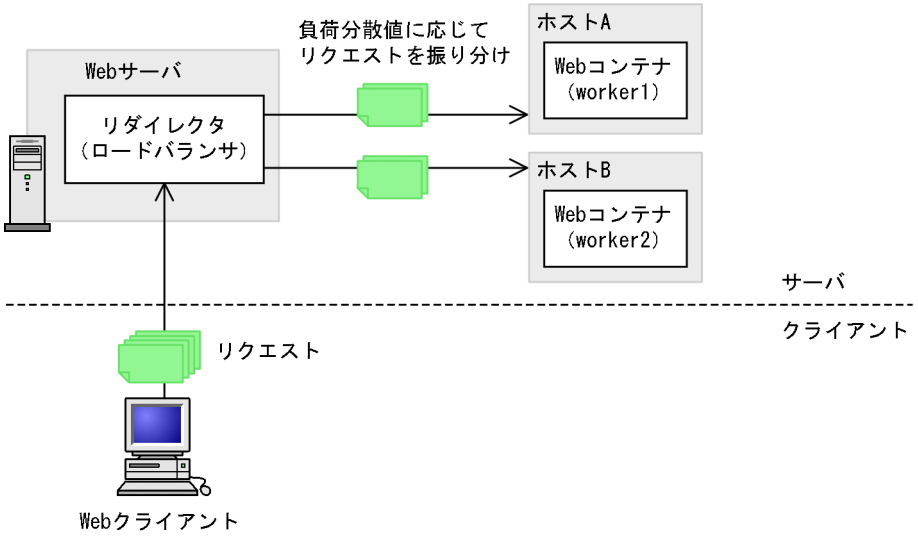
(2)　workers.properties および mod_jk.conf での設定

workers.properties および mod_jk.conf での設定については，Smart Composer を使用する場合と同様です。「5.4.4(2)　workers.properties および mod_jk.conf での設定」を参照してください。

(3)　設定例

ラウンドロビン方式によるリクエスト振り分けの例を次の図に示します。

図 5-9　ラウンドロビン方式によるリクエスト振り分けの例



この例では，「/examples」以下のリクエストがホスト A，ホスト B に同じ比率で振り分けます。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。

workers.properties の例を次に示します。ここでは，ロードバランサとワーカを定義します。リクエストを同じ比率で振り分けるので，「worker1」と「worker2」の負荷分散値はどちらも「1」を指定します。

workers.properties の例（Windows の場合）

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1,worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.lbfactor=1
```

workers.properties の例（UNIX の場合）

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1,worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.lbfactor=1
```

mod_jk.conf および uriworkermap.properties の例を次に示します。ここでは，ロードバランサ名「loadbalancer1」を指定します。

mod_jk.conf の例（HTTP Server の場合）

```
JkMount /examples/* loadbalancer1
```

uriworkermap.properties の例（Microsoft IIS の場合）

```
/examples/*=loadbalancer1
```

5.4.6　ラウンドロビン方式によるリクエストの振り分けに関する注意事項

リダイレクタのラウンドロビン方式によるリクエストの振り分けに関する注意事項を次に示します。

- **J2EE アプリケーション停止中の Web コンテナへのリクエストの送信**

  ラウンドロビン方式でリクエストを振り分ける場合，J2EE アプリケーションが停止中であっても Web コンテナへリクエストが送信されます。このため，J2EE アプリケーションの変更は，すべての Web コンテナをシステムから隔離した状態で実施する必要があります。

- **負荷分散機によるヘルスチェックの無効**

  負荷分散機と，ラウンドロビン方式によるリクエストの振り分けを併用した場合，J2EE サーバに障害が発生したときも，リダイレクタで正常な Web コンテナへ転送されます。このため，負荷分散機で J2EE サーバの障害が検知できなくなり，Web コンテナの監視が行えません。

5.5　POST データサイズでのリクエストの振り分け

この節では，POST データサイズでのリクエストの振り分けについて説明します。

この節の構成を次の表に示します。

表 5-9　この節の構成（POST データサイズでのリクエストの振り分け）

分類	タイトル	参照先
解説	POST データサイズでのリクエストの振り分けの概要	5.5.1
	POST データサイズによるリクエストの振り分けの例	5.5.2
	リクエストの振り分け条件	5.5.3
	POST データサイズによるリクエストの振り分けの定義	5.5.4
設定	実行環境での設定（Smart Composer 機能を使用する場合）	5.5.5
	実行環境での設定（Smart Composer 機能を使用しない場合）	5.5.6

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

5.5.1　POST データサイズでのリクエストの振り分けの概要

Web コンテナがクラスタ構成で配置されている場合，リダイレクタを使用して，POST データサイズごとにそれぞれの Web コンテナにリクエストを振り分けられます。この機能を使用すると，処理に時間が掛かる長大な POST データのリクエストを，特定の Web コンテナに転送できます。これによって，長大な POST データ以外のリクエストのスループットの低下や，レスポンス時間の低下を防ぐことができます。この方式でリクエストを振り分ける場合は，振り分け処理をする各 Web コンテナに，それぞれ Web アプリケーションがデプロイされていることが前提になります。ただし，各 J2EE サーバにデプロイする Web アプリケーションが同一である必要はありません。

クラスタ構成での POST データサイズによるリクエストの振り分けには，**POST リクエスト振り分けワーカ**というワーカ定義を使用します。POST リクエスト振り分けワーカには，振り分け先となるワーカプロセスのリストが定義されています。この定義を基に，ワーカプロセスに対する POST データサイズの振り分けが実行されます。POST リクエスト振り分けワーカの振り分け先となるワーカプロセスを**POST リクエスト転送先ワーカ**といいます。

POST リクエスト転送先ワーカへの振り分けは，HTTP リクエスト単位で実行されます。

注意事項

HTTP Cookie または URL 書き換えによる制御でセッションが管理されている場合でも，POST データサイズによる振り分けを設定しているときは，POST データサイズによって，リクエストの

振り分け先が決定されます。このため，同一クライアントからのリクエストの場合でも，HttpSession のセッション ID は引き継がれません。

例えば，J2EE サーバ 1 で HttpSession のセッション ID を生成したあと，J2EE サーバ 2 へリクエストが転送された場合，J2EE サーバ 2 で新たに HttpSession のセッション ID が生成されます。この場合，再度 J2EE サーバ 1 にアクセスすると，クライアントでは J2EE サーバ 2 で生成された HttpSession のセッション ID を使用しているため，J2EE サーバ 1 で新たに HttpSession のセッション ID が生成されます。このため，HttpSession のセッションは引き継がれません。

なお，J2EE サーバ 1 で HttpSession のセッション ID を生成したあと，J2EE サーバ 2 へリクエストが転送されても，J2EE サーバ 2 で HttpSession のセッション ID が生成されない場合，再度 J2EE サーバ 1 にアクセスすると，J2EE サーバ 1 で生成済みの HttpSession のセッション ID がそのまま使用されます。

5.5.2 POST データサイズによるリクエストの振り分けの例

POST データサイズでリクエストを振り分ける場合，POST リクエスト振り分けワーカに転送されるリクエストが限定できるかどうかによって，POST データサイズの上限値に設定する値が異なります。

- POST リクエスト振り分けワーカに転送されるリクエストが限定できる場合

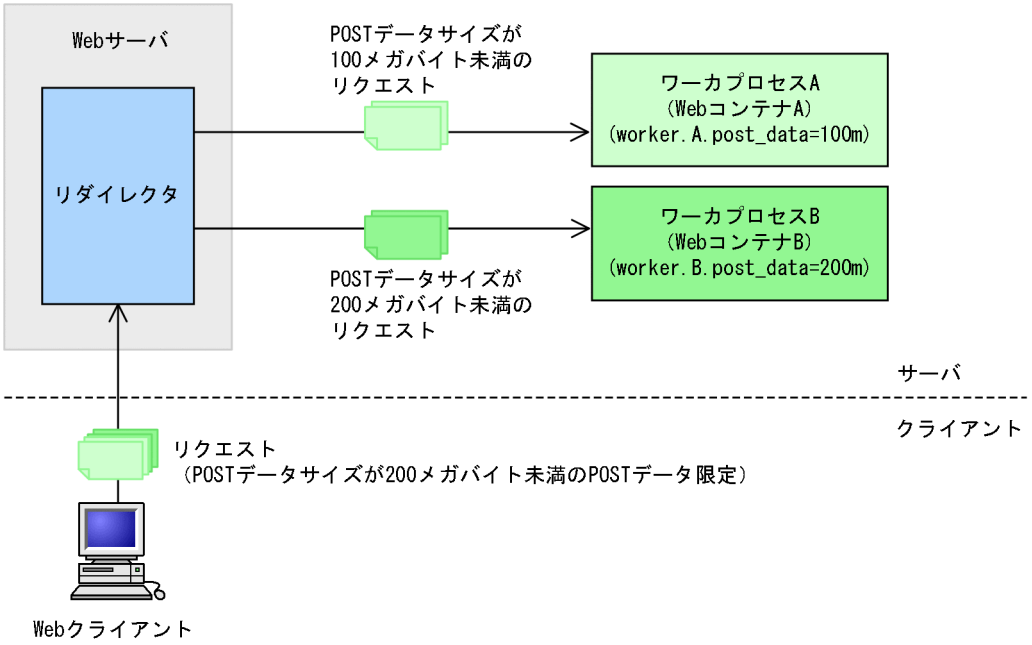
  ここでは，次の条件を満たすリクエストが，POST リクエスト振り分けワーカに転送されるものとして説明します。

  - POST データのリクエストである。
  - リクエストの POST データサイズが 200 メガバイト未満である。

  リクエストが限定できる場合，処理したい長大な POST データの範囲も限定できます。長大な POST データのリクエストを処理するワーカに，特定の POST データサイズのリクエストが転送されるように，それぞれのリクエスト転送先ワーカで上限値を設定します。

  リクエストが限定できる場合の，POST データサイズによるリクエストの振り分けの例を次の図に示します。

図 5-10　POST データサイズによるリクエストの振り分けの例（リクエストが限定できる場合）



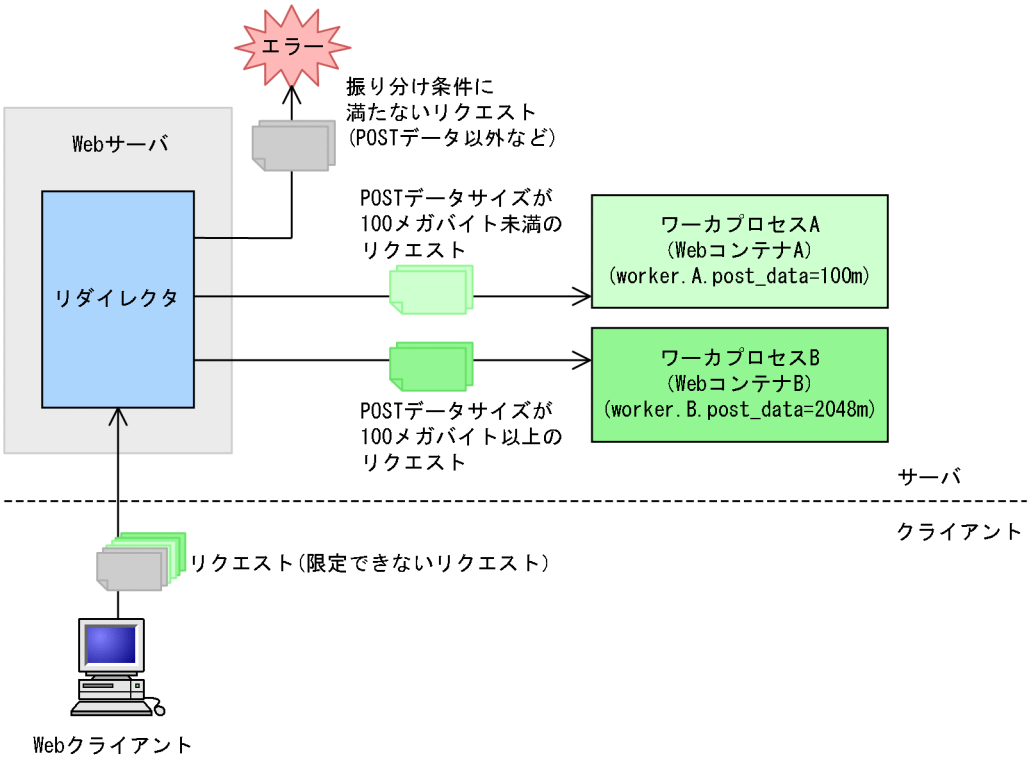
この図では，POST リクエスト転送先ワーカを二つ用意し，POST データサイズが 100 メガバイト以上 200 メガバイト未満のリクエストがワーカプロセス B に転送されるように，それぞれに POST データサイズの上限値を設定しています。リクエストの POST データサイズが上限値未満の場合に，それぞれの POST リクエスト転送先ワーカに振り分けられます。リクエストの POST データサイズが，複数の POST リクエスト転送先ワーカに当てはまる場合，POST データサイズの上限値が最も小さい POST リクエスト転送先ワーカに，リクエストは振り分けられます。例えば，POST データサイズが 80 メガバイトのリクエストの場合は，どちらのワーカにも該当しますが，ワーカプロセス A に振り分けられます。

- POST リクエスト振り分けワーカに転送されるリクエストが限定できない場合

  リクエストを限定できない場合，長大な POST データを処理するワーカの POST データサイズの上限値には，最大値を設定します。

  リクエストが限定できない場合の，POST データサイズによるリクエストの振り分けの例を次の図に示します。

図 5-11　POST データサイズによるリクエストの振り分けの例（リクエストが限定できない場合）



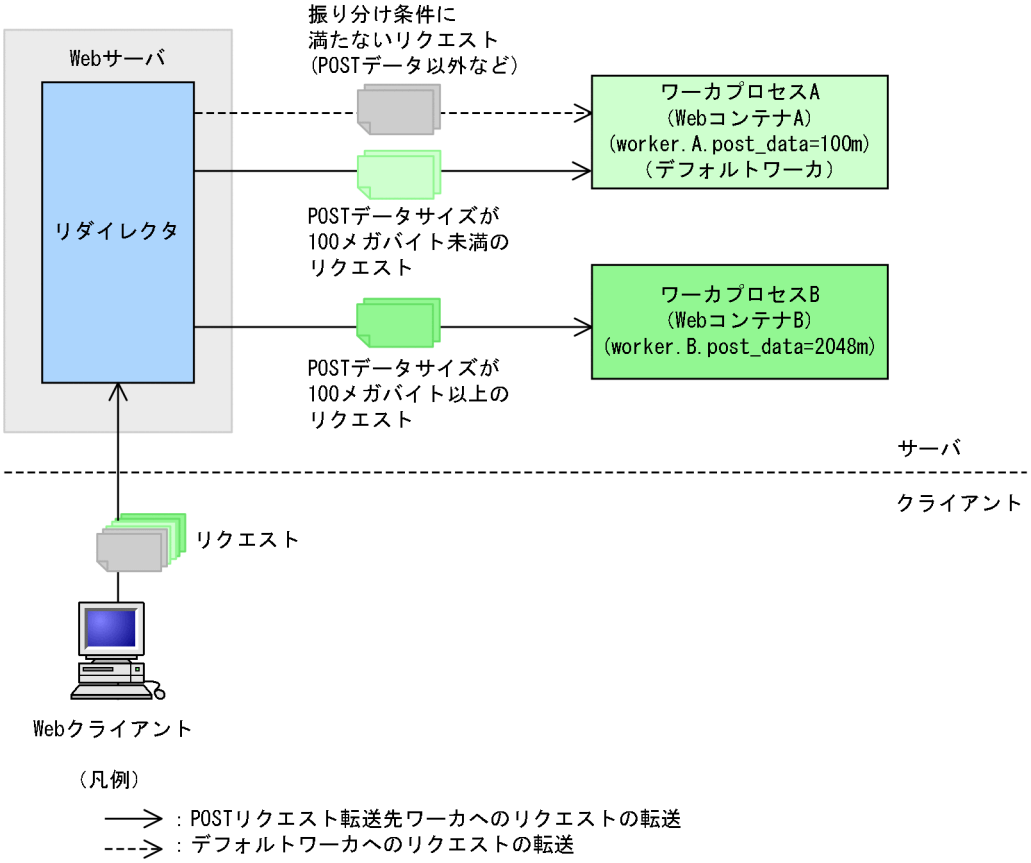
この図では，POST リクエスト転送先ワーカを二つ用意し，それぞれに POST データサイズの上限値を設定しています。長大な POST データのリクエストすべてをワーカプロセス B で処理するように，ワーカプロセス B の POST データサイズの上限値には最大値を設定しています。ワーカプロセス A の上限値以上（100 メガバイト以上）の POST データのリクエストは，すべてワーカプロセス B に転送されます。なお，この場合，POST データ以外のリクエストや，POST データサイズが参照できないリクエストなどが転送されると，リクエスト振り分けワーカで振り分けられないため，リダイレクタによってエラーステータスコード 400 が返されてエラーとなります。

POST データサイズでリクエストを振り分ける場合，リクエストの振り分け条件に満たないリクエストが POST リクエスト振り分けワーカに転送されると，リダイレクタによってエラーステータスコード 400 が返されてエラーとなります。リクエストの振り分け条件については，「5.5.3　リクエストの振り分け条件」を参照してください。

リクエストの振り分け条件に満たないリクエストも処理したい場合は，そのリクエストを転送するワーカプロセスを設定する必要があります。リクエストの振り分け条件に満たないリクエストを転送するワーカプロセスを**デフォルトワーカ**といいます。なお，デフォルトワーカの設定は任意です。

リクエストが限定できない場合に，リクエストの振り分け条件に満たないリクエストをデフォルトワーカに転送する例を次の図に示します。

図 5-12　POST データサイズによるリクエストの振り分けの例（デフォルトワーカを設定した場合）



この図では，リクエストの振り分け条件に満たないリクエストを，デフォルトワーカのワーカプロセス A に転送するように設定しています。

5.5.3　リクエストの振り分け条件

POST リクエスト転送先ワーカに振り分けられるリクエストは，次の条件を満たしている必要があります。

POST リクエスト転送先ワーカに振り分けられるリクエストの条件

- リクエストのメソッドが POST であること。
- リクエストに Content-Length ヘッダがある（ボディデータがチャンク形式でない）こと。
- リクエストの Content-Length ヘッダの値が，ワーカに設定している POST データサイズ未満であること。

どれか一つでも条件を満たさないリクエストは，デフォルトワーカに振り分けられます。デフォルトワーカが設定されていない場合は，エラーとなり，エラーステータスコード 400 のエラーが返されます。

5.5.4　POST データサイズによるリクエストの振り分けの定義

標準で提供される workers.properties には，あらかじめ次に示す POST リクエスト割り分けワーカが定義されています。

```
#worker.list=postsizelb1
#worker.postsizelb1.type=post_size_lb
#worker.postsizelb1.post_size_workers=worker1,worker2
#worker.postsizelb1.default_worker=worker1
```

worker.postsizelb1.type では，ワーカのタイプを設定します。worker.postsizelb1.post_size_workers では，振り分け対象となる POST リクエスト転送先ワーカのワーカプロセス名を設定します。worker.postsizelb1.default_worker では，デフォルトワーカを設定します。workers.properties には，postsizelb1 として，ワーカのタイプに「post_size_lb」，POST リクエスト転送先ワーカに「worker1，worker2」，デフォルトワーカに「worker1」が定義されています。

ただし，この定義はコメントとして記述されています。このため，この定義の POST リクエスト割り分けワーカを利用する場合は，workers.properties の該当する行の先頭の「#」を削除してください。

リクエストを振り分ける POST データサイズは，workers.properties のワーカ定義の post_data パラメタで設定します。

例えば，標準提供の postsizelb1 の定義を利用して，worker1 と worker2 という二つの POST リクエスト転送先ワーカに，それぞれ次のように POST データサイズを定義しているとします。

- worker1 の post_data パラメタ：100m
- worker2 の post_data パラメタ：200m

この場合，worker1 には 100 メガバイト未満のリクエストが，worker2 には 100 メガバイト以上 200 メガバイト未満のリクエストが振り分けられます。リクエスト振り分けワーカが，200 メガバイト以上のリクエストを振り分けた場合，そのリクエストはデフォルトワーカに設定されている worker1 に転送されます。

5.5.5　実行環境での設定（Smart Composer 機能を使用する場合）

振り分け先となるワーカのリストを POST リクエスト振り分けワーカに定義することで，POST データサイズでワーカにリクエストを振り分けます。

振り分け先となる POST リクエスト転送先ワーカに，POST データサイズの上限値を設定して，リクエスト振り分け先を定義します。これによって，特定のホストに，処理に時間が掛かる長大な POST データサイズのリクエストの処理を振り分けられます。リダイレクタは，HTTP リクエスト単位に POST データサイズの上限値で振り分けるので，長大な POST データ以外のリクエストのスループットの低下や，レスポンス時間の低下を防ぐことができます。なお，POST データサイズの上限値が設定されている場合は，同じセッションに属する HTTP リクエストであっても，POST データサイズの値が優先されます。

(1) 設定の手順

POST データサイズでのリクエスト振り分けは，次の手順で設定します。

1. workers.properties で POST リクエスト振り分けワーカ，および POST リクエスト転送先ワーカを定義します。

   POST リクエスト振り分けワーカの定義

   ワーカ名のリスト，ワーカのタイプ（「post_size_lb」を設定），POST データサイズによる振り分けの対象となるワーカのリストなどを設定します。必要に応じて，デフォルトワーカも設定します。

   各 POST リクエスト転送先ワーカの定義

   ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名，POST データサイズの上限値などを設定します。

   標準で提供される workers.properties には，デフォルト値が定義されています。コメントとして定義されているデフォルトの定義を使用する場合は，該当する行の先頭の「#」を削除してください。

   workers.properties（ワーカ定義ファイル）については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

2. mod_jk.conf で URL パターンとワーカのマッピングを定義します。

   すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

3. Web サーバの環境を設定して，Web サーバを再起動します。

   Web サーバの設定については，「付録 D　HTTP Server の設定に関する注意事項」を参照してください。

(2) workers.properties および mod_jk.conf での設定

POST データサイズでのリクエストの振り分けの設定は，workers.properties および mod_jk.conf で定義します。workers.properties および mod_jk.conf で定義するキーを次の表に示します。

表 5-10　workers.properties および mod_jk.conf で定義するキー（POST データサイズでのリクエストの振り分けの場合）

ファイルの種類	キー名	説明
workers.properties	worker.list	一つまたは複数のワーカ名のリストを指定します。
	worker.<ワーカ名>.host	ワーカのホスト名，または IP アドレスを指定します。
	worker.<ワーカ名>.port	ワーカのポート番号を指定します。
	worker.<ワーカ名>.type	ワーカのタイプを指定します。POST リクエスト振り分けワーカには「post_size_lb」を，振り分け対象となる POST リクエスト転送先ワーカには「ajp13」を指定します。

ファイルの種類	キー名	説明
	worker.<ワーカ名>.post_size_workers	POST データサイズによる振り分けの対象となるワーカのリストを指定します。
	worker.<ワーカ名>.post_data	リクエストの POST データサイズの上限値（バイト単位）を指定します。
	worker.<ワーカ名>.default_worker	リクエストの振り分け先に該当するワーカが，クラスタ構成内の POST リクエスト転送先ワーカにない場合に，リクエストを転送するワーカ（デフォルトワーカ）を指定します。
	worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカのコネクション数を指定します。 Windows の場合にだけ指定できます。
	worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。
	worker.<ワーカ名>.delegate_error_code	エラーページの作成を Web サーバに委任する場合に利用するエラーステータスコードを指定します。
mod_jk.conf	JkMount	URL パターンと，worker.list で指定されているワーカのどれかとの組み合わせを指定します。

注　<ワーカ名>には，worker.list キー，または worker.<ワーカ名>.post_size_workers キーで指定したワーカの名称を定義します。

ワーカの種類ごとに指定できるキーを次の表に示します。

表 5-11　ワーカの種類ごとに指定できるキー

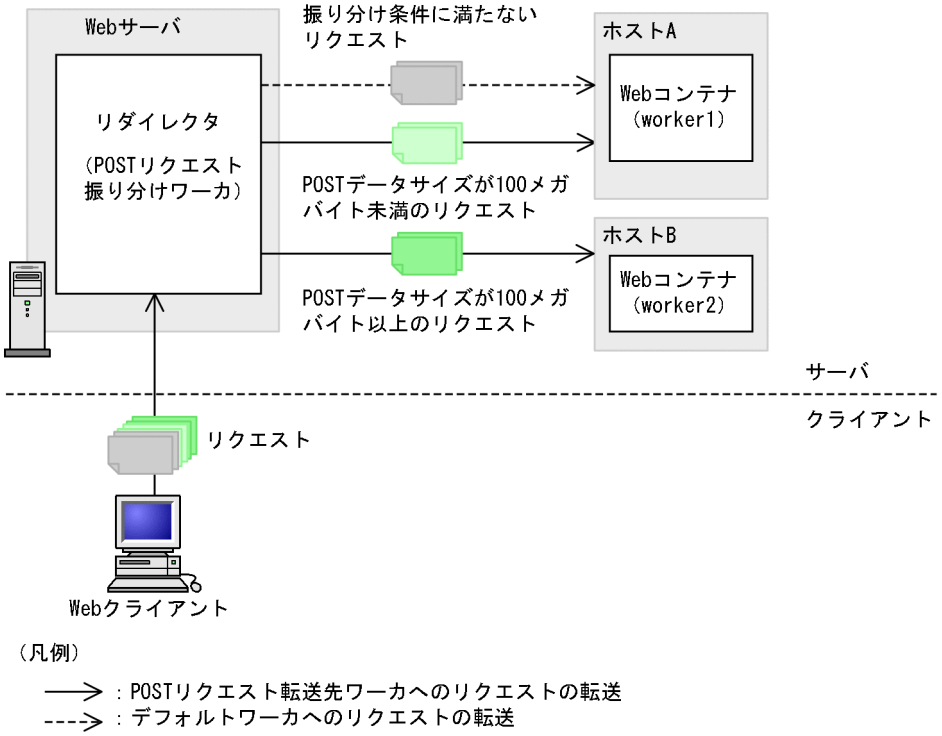
キー名	ワーカの種類（worker.<ワーカ名>.type キーの指定値）	
	POST リクエスト振り分けワーカ（「post_size_lb」を指定）	POST リクエスト転送先ワーカ（「ajp13」を指定）
worker.<ワーカ名>.host	×	○
worker.<ワーカ名>.port	×	○
worker.<ワーカ名>.type	○	○
worker.<ワーカ名>.post_size_workers	○	×
worker.<ワーカ名>.post_data	×	○
worker.<ワーカ名>.default_worker	△	×
worker.<ワーカ名>.cachesize	×	△
worker.<ワーカ名>.receive_timeout	×	△
worker.<ワーカ名>.delegate_error_code	×	△

（凡例）○：指定できる。 ×：指定できない。 △：任意に指定できる。

(3) 設定例

POST データサイズでのリクエスト振り分けの例を次の図に示します。ここでは，リクエストを限定できない場合を例に説明します。

図 5-13 POST データサイズでのリクエスト振り分けの例



この例では，「/examples」以下のリクエストのうち，POST データサイズが 100 メガバイト未満のリクエストをホスト A，POST データサイズが 100 メガバイト以上 200 メガバイト未満のリクエストをホスト B に振り分けます。リクエストの振り分け条件に満たないリクエストは，デフォルトワーカに設定したホスト A に振り分けます。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。リクエストの振り分け条件については，「5.5.3 リクエストの振り分け条件」を参照してください。

workers.properties の例を次に示します。ここでは，POST リクエスト振り分けワーカ，POST リクエスト転送先ワーカおよびデフォルトワーカを定義します。POST データサイズの上限値は，「worker1」に「100m」，「worker2」に「2048m」（POST データサイズの上限値の最大値）を指定します。デフォルトワーカには，「worker1」を指定します。

workers.properties の例（Windows の場合）

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1,worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
```

```
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.post_data=2048m
```

workers.properties の例（UNIX の場合）

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1,worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.post_data=2048m
```

mod_jk.conf の例を次に示します。ここでは，POST リクエスト振り分けワーカ「postsizelb1」を指定します。

mod_jk.conf の例

```
JkMount /examples/* postsizelb1
```

参考

POST データサイズの上限値は，httpsd.conf（HTTP Server 定義ファイル）の LimitRequestBody ディレクティブでも設定できます。LimitRequestBody ディレクティブの詳細については，マニュアル「HTTP Server」を参照してください。

5.5.6 実行環境での設定（Smart Composer 機能を使用しない場合）

振り分け先となるワーカのリストを POST リクエスト振り分けワーカに定義することで，POST データサイズでワーカにリクエストを振り分けます。

振り分け先となる POST リクエスト転送先ワーカに，POST データサイズの上限値を設定して，リクエスト振り分け先を定義します。これによって，特定のホストに，処理に時間が掛かる長大な POST データサイズのリクエストの処理を振り分けられます。リダイレクタは，HTTP リクエスト単位に POST データサイズの上限値で振り分けるので，長大な POST データ以外のリクエストのスループットの低下や，レスポ

ンス時間の低下を防ぐことができます。なお，POST データサイズの上限値が設定されている場合は，同じセッションに属する HTTP リクエストであっても，POST データサイズの値が優先されます。

注意事項

Microsoft IIS と連携する場合，POST データサイズによるリクエストの振り分けは設定できません。

(1) 設定の手順

POST データサイズでのリクエスト振り分けは，次の手順で設定します。

1. workers.properties で POST リクエスト振り分けワーカ，および POST リクエスト転送先ワーカを定義します。

   POST リクエスト振り分けワーカの定義

   ワーカ名のリスト，ワーカのタイプ（「post_size_lb」を設定），POST データサイズによる振り分けの対象となるワーカのリストなどを設定します。

   必要に応じて，デフォルトワーカも設定します。

   各 POST リクエスト転送先ワーカの定義

   ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名，POST データサイズの上限値などを設定します。

   標準で提供される workers.properties には，デフォルト値が定義されています。コメントとして定義されているデフォルトの定義を使用する場合は，該当する行の先頭の「#」を削除してください。

   workers.properties（ワーカ定義ファイル）については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

2. mod_jk.conf で URL パターンとワーカのマッピングを定義します。

   すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

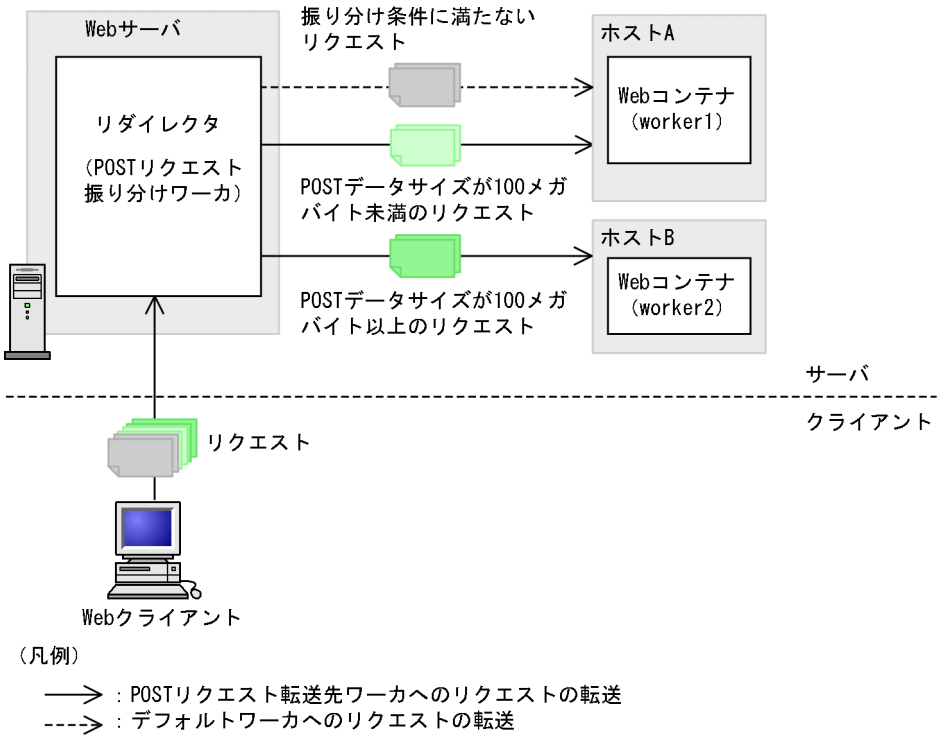
3. Web サーバの環境を設定して，Web サーバを再起動します。

   Web サーバの設定については，「付録 D　HTTP Server の設定に関する注意事項」を参照してください。

(2) 設定例

POST データサイズでのリクエスト振り分けの例を次の図に示します。ここでは，リクエストを限定できない場合を例に説明します。

図 5-14　POST データサイズでのリクエスト振り分けの例



この例では，「/examples」以下のリクエストのうち，POST データサイズが 100 メガバイト未満のリクエストをホスト A，POST データサイズが 100 メガバイト以上 200 メガバイト未満のリクエストをホスト B に振り分けます。リクエストの振り分け条件に満たないリクエストは，デフォルトワーカに設定したホスト A に振り分けます。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。リクエストの振り分け条件については，「5.5.3　リクエストの振り分け条件」を参照してください。

workers.properties の例を次に示します。ここでは，POST リクエスト振り分けワーカ，POST リクエスト転送先ワーカおよびデフォルトワーカを定義します。POST データサイズの上限値は，「worker1」に「100m」，「worker2」に「2048m」（POST データサイズの上限値の最大値）を指定します。デフォルトワーカには，「worker1」を指定します。

workers.properties の例（Windows の場合）

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1,worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
```

```
worker.worker2.cachesize=64
worker.worker2.post_data=2048m
```

workers.properties の例（UNIX の場合）

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1,worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.post_data=2048m
```

mod_jk.conf の例を次に示します。ここでは，POST リクエスト振り分けワーカ「postsizelb1」を指定します。

mod_jk.conf の例

```
JkMount /examples/* postsizelb1
```

参考

POST データサイズの上限値は，httpsd.conf（HTTP Server 定義ファイル）の LimitRequestBody ディレクティブでも設定できます。LimitRequestBody ディレクティブの詳細については，マニュアル「HTTP Server」を参照してください。

5.6　通信タイムアウト（Web サーバ連携）

この節では，Web サーバ連携での通信タイムアウトについて説明します。

Web サーバ連携機能を使用する場合，クライアント－Web サーバ間，および Web サーバ－Web コンテナ間での，リクエスト受信およびレスポンス送信に，通信のタイムアウトを設定できます。ネットワークの障害やアプリケーションの障害発生などで応答待ち状態になった場合，通信タイムアウトを設定していると，タイムアウトの発生によって障害の発生を検知できます。

この節の構成を次の表に示します。

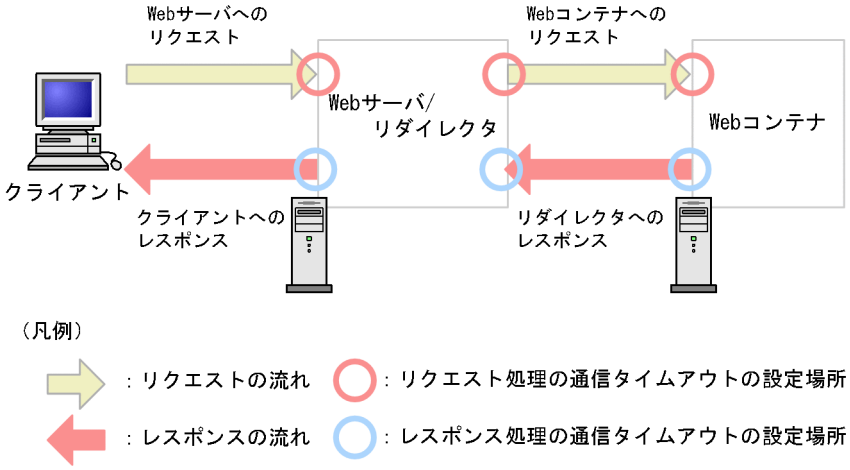
表 5-12　この節の構成（通信タイムアウト（Web サーバ連携））

分類	タイトル	参照先
解説	リクエスト送受信時の通信タイムアウト	5.6.1
	レスポンス送受信時の通信タイムアウト	5.6.2
設定	通信タイムアウトの設定	5.6.3
	リクエスト送受信時の通信タイムアウトの設定（Smart Composer 機能を使用する場合）	5.6.4
	リクエスト送受信時の通信タイムアウトの設定（Smart Composer 機能を使用しない場合）	5.6.5
	レスポンス送受信時の通信タイムアウトの設定（Smart Composer 機能を使用する場合）	5.6.6
	レスポンス送受信時の通信タイムアウトの設定（Smart Composer 機能を使用しない場合）	5.6.7

注　「実装」，「運用」および「注意事項」について，この機能固有の説明はありません。

Web サーバ連携機能を使用する場合，通信タイムアウトは，次に示す図中の四つの矢印が示す通信に対して設定します。さらに，リダイレクター－Web コンテナ間の通信では，リダイレクタ側，Web コンテナ側の両方で通信タイムアウトを設定できます。リクエスト送信の場合は，リダイレクタ側でのリクエスト送信時および Web コンテナ側でのリクエスト受信時に設定できます。また，レスポンス送信の場合は，Web コンテナ側でのレスポンス送信時およびリダイレクタ側でのレスポンス受信時に設定できます。タイムアウトを設定できる通信について次の図に示します。

図 5-15　タイムアウトを設定できる通信

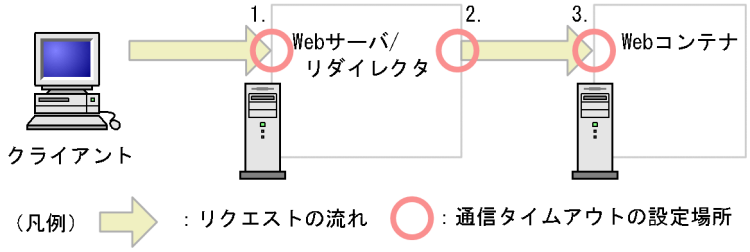


通信タイムアウトの設定について，リクエストの受信とレスポンスの送信に分けて説明します。

5.6.1　リクエスト送受信時の通信タイムアウト

ここでは，Web サーバ連携機能を使用する場合のリクエスト送受信時の通信タイムアウト設定について説明します。リクエスト送受信時の，通信タイムアウトの設定場所を次の図に示します。

図 5-16　リクエスト送受信時の通信タイムアウトの設定場所



Web サーバ連携機能を使用する場合，図中の○印の場所に通信タイムアウトを設定します。通信タイムアウトを設定する場所について説明します。なお，図中の項番は次の説明の項番と対応しています。

1. Web サーバでのリクエスト受信時（クライアント－Web サーバ）

   クライアントからのリクエストを，Web サーバで受信するときに，通信タイムアウトを設定します。通信タイムアウトは，Web サーバで設定します。

   Web サーバでのリクエスト受信時に通信タイムアウトを設定することによって検知できる障害については，「5.6.1(1)　Web サーバでのリクエスト受信時」を参照してください。

2. リダイレクタでのリクエスト送信時（リダイレクター－Web コンテナ）

   リダイレクタから Web コンテナにリクエストを送信するときに，通信タイムアウトを設定します。通信タイムアウトは，リダイレクタで設定します。

   リダイレクタでのリクエスト送信時に通信タイムアウトを設定することによって検知できる障害については，「5.6.1(2)　リダイレクタでのリクエスト送信時」を参照してください。

3. Web コンテナでのリクエスト受信時（リダイレクタ－Web コンテナ）

リダイレクタからのリクエストを，Web コンテナで受信するときに，通信タイムアウトを設定します。通信タイムアウトは，Web コンテナで設定します。

Web コンテナでのリクエスト受信時に通信タイムアウトを設定することによって検知できる障害については，「5.6.1(3)　Web コンテナでのリクエスト受信時」を参照してください。

(1)　Web サーバでのリクエスト受信時

Web サーバでは，クライアントから転送されたリクエストの受信処理に，タイムアウト時間を設定できます。設定した通信タイムアウトによって，クライアント側で障害が発生したことを検知できます。検知できる障害を次に示します。

検知できる障害

- クライアントが稼働するホストがダウンした
- クライアント－Web サーバ間でネットワーク障害が発生した
- クライアントアプリケーションで障害が発生した

(2)　リダイレクタでのリクエスト送信時

リダイレクタでは，Web コンテナへのリクエスト送信処理にタイムアウト時間を設定できます。設定したタイムアウト時間を超えてタイムアウトが発生した場合には HTTP Server のエラーログにメッセージが出力されます。

(a)　設定できる通信タイムアウトと検知できる障害

リダイレクタでのリクエスト送信処理では，次に示す二つの時間にタイムアウトが設定できます。

- Web コンテナにリクエストを送信するための，コネクションを確立する時間
- Web コンテナにリクエストを送信する時間

設定した通信タイムアウトによって，Web コンテナ側またはネットワーク上で障害が発生したことを検知できます。検知できる障害を次に示します。

検知できる障害

- Web コンテナが稼働するホストがダウンした
- リダイレクタ－Web コンテナ間でネットワーク障害が発生した

(b)　リクエスト送信処理のリトライ

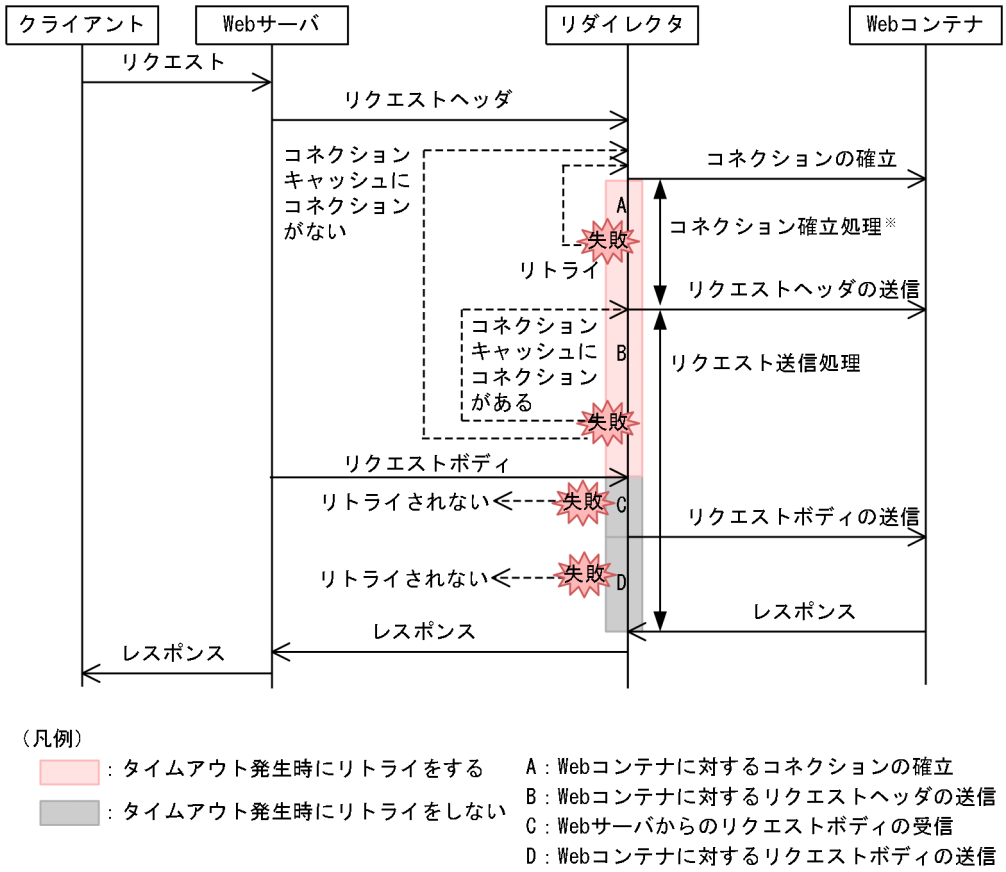
リダイレクタからのリクエスト送信処理が一時的にできなくなった場合，リクエスト送信処理のリトライができます。一時的にリクエスト送信ができない場合とは次のような場合です。

- ネットワークの一時的な障害が発生した場合

- コネクション確立時に，Web コンテナにリクエストが集中している状態で，確立要求が一時的にリスンキューからあふれた場合
- Web コンテナの起動完了を待っている場合

リトライできる処理は，コネクションの確立処理，およびリクエストヘッダの送信処理となります。リトライ処理の流れを次の図に示します。

図 5-17　Web コンテナへのリクエスト送信時のリトライ処理の流れ



（凡例）

：タイムアウト発生時にリトライをする

：タイムアウト発生時にリトライをしない

A：Webコンテナに対するコネクションの確立
B：Webコンテナに対するリクエストヘッダの送信
C：Webサーバからのリクエストボディの受信
D：Webコンテナに対するリクエストボディの送信

注※　コネクション確立は，コネクションキャッシュにコネクションがない場合だけ実施されます。

リトライは，図中の A または B でタイムアウトが発生した場合に実施されます。図中 C または D の部分で処理に失敗し，タイムアウトが発生した場合，リトライは実施されません。コネクションはクローズされ，クライアントにエラーが返ります。なお，図中 C または D の部分で処理の失敗とは，Web サーバからのリクエストボディの受信に失敗した場合，または Web コンテナへのリクエストボディの送信に失敗した場合を指します。

ポイント

図中の C または D の処理では，Web コンテナ側ですでにリクエストに対する処理が開始されている可能性があります。Web サーバからのリクエストボディの受信に失敗した場合，および Web コンテナへのリクエストボディの送信に失敗した場合は，リトライを実施すると二重送信となるおそれがあるため，リトライはされません。

次に，図中 A および図中 B で処理に失敗した場合のリトライについて説明します。

- **図中 A の部分で処理に失敗した場合のリトライ**

  リダイレクタから Web コンテナにコネクションの確立要求を出したあと，Web コンテナが稼働するホストの電源が切断されたり，ネットワーク障害が発生したりした場合，次のように動作します。

  1. 設定したタイムアウトの時間が経過すると，コネクション確立でタイムアウトが発生したことを示すメッセージが出力されます。
  2. 指定した回数だけコネクションの確立をリトライします。

  なお，リトライ回数分実施してもコネクションの確立ができなかった場合，リクエスト送信に失敗したことを示すメッセージが出力され，クライアントにエラー（ステータスコード 500）が返されます。

- **図中 B の部分で処理に失敗した場合のリトライ**

  コネクションの確立が成功したあと，または Web コンテナにリクエストヘッダの送信をしたあと，Web コンテナが稼働するホストの電源が切断されたり，ネットワーク障害が発生したりした場合，次のように動作します。

  1. 設定したタイムアウトの時間が経過すると，リクエスト送信でタイムアウトが発生したことを示すメッセージが出力されます。
  2. リクエストヘッダの送信処理に使用されたコネクションをクローズします。
  3. 指定した回数だけリクエストヘッダの送信をリトライします。

     なお，このとき，コネクションキャッシュにコネクションがあるかどうかで動作が異なります。

     ・コネクションキャッシュにコネクションがある場合

     コネクションキャッシュ中のコネクションを使用して，リクエストヘッダの送信処理からリトライします。

     ・コネクションキャッシュにコネクションがない場合

     再度，コネクションの確立を実施してから，リクエストヘッダの送信処理をリトライします。

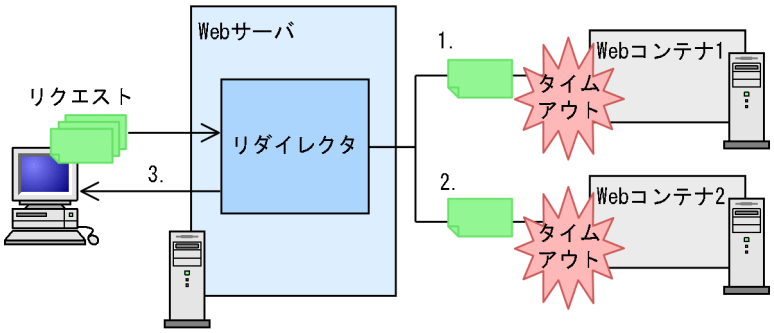
  なお，リトライ回数分実施してもリクエストヘッダの送信ができなかった場合，リクエスト送信に失敗したことを示すメッセージが出力され，クライアントにステータスコード 500 が返されます。

- **ロードバランサを使用してリクエストの振り分けをしている場合のリトライ動作**

  ロードバランサを使用したリクエストの振り分けをしている場合のリトライ動作について，次の図で説明します。

  なお，この図では，リクエストは，Web コンテナ 1，Web コンテナ 2 の順で振り分けられ，リトライ回数は 3 回が設定されていることとします。

図 5-18　ロードバランサを使用してリクエストの振り分けをしている場合のリトライ動作



図のリトライ動作について説明します。

1. Web コンテナ 1 へのリクエスト送信

   リクエストは Web コンテナ 1 に送信されます。Web コンテナ 1 へのコネクションの確立またはリクエストヘッダの送信処理に失敗すると，リトライをします。リトライは 3 回まで実施されます。

2. Web コンテナ 2 へのリクエスト送信

   Web コンテナ 1 でのリトライに 3 回失敗すると，リクエストは Web コンテナ 2 に転送されます。リクエストが転送されると，リトライは再度 1 からカウントされるため，Web コンテナ 2 に対しても，最大で 3 回リトライが実行されます。

3. クライアントへのエラー通知

   Web コンテナ 2 に対しても，コネクションの確立またはリクエストヘッダの送信処理が 3 回失敗すると，クライアントにエラー（ステータスコード 500）が返ります。

> **参考**
>
> リクエストの転送は，設定されている Web コンテナの数だけ実施されます。

(3)　Web コンテナでのリクエスト受信時

Web コンテナでは，リダイレクタから転送されたリクエストの受信処理に，タイムアウト時間を設定できます。設定した通信タイムアウトによって，リダイレクタ側で障害が発生したことを検知できます。検知できる障害を次に示します。

検知できる障害

- Web サーバが稼働するホストがダウンした
- Web サーバ－Web コンテナ間のネットワーク障害が発生した
- クライアント－Web サーバ間での処理が終わる前に，Web コンテナ側でタイムアウトが発生した

  これは，Web コンテナが要求したデータサイズ分の読み込みを，クライアント－Web サーバ間で行っているときに，クライアント－Web サーバ間の通信速度が十分ではないなどの理由によって，データの読み込みが終了する前に，Web コンテナで設定した通信タイムアウトが発生したことを意味します。

通信タイムアウトが発生する条件と発生後の動作について次の表に示します。

表 5-13　通信タイムアウトが発生する条件と発生後の動作

通信タイムアウトが発生する条件	発生後の動作
リクエスト受信時に次の条件をすべて満たしている場合 • リクエストにボディデータが存在する。 • ボディデータの形式がチャンク形式ではない。 • 読み込み処理に入ったあとで，リダイレクタが稼働するホスト，またはリダイレクターWeb コンテナ間のネットワークに障害が発生した。	• リクエスト処理は実行されない。 • メッセージログに KDJE39188-E のメッセージを出力する。
サーブレット(JSP)内での API※使用時に，次の条件をすべて満たしている場合 • リクエストにボディデータが存在する。 • ボディデータの形式がチャンク形式ではない。 • 読み込み処理に入ったあとで，リダイレクタが稼働するホスト，またはリダイレクターWeb コンテナ間のネットワークに障害が発生した。	• java.lang.IllegalStateException が発生する。 • リダイレクタとのコネクションがクローズされ，それ以降のデータの読み込み，書き込みができなくなる。 • メッセージログに KDJE39188-E のメッセージを出力する。
サーブレット(JSP)内で javax.servlet.ServletInputStream クラスまたは javax.servlet.ServletRequest クラスの getReader メソッドによって得られた java.io.BufferedReader クラスを使用して POST データを読み込んだ際に，次の条件をすべて満たしている場合 • リクエストにボディデータが存在する。 • 読み込み処理に入ったあとで，リダイレクタが稼働するホスト，またはリダイレクターWeb コンテナ間のネットワークに障害が発生した。	• java.net.SocketTimeoutException が発生する。 • リダイレクタとのコネクションがクローズされ，それ以降のデータの読み込み，書き込みができなくなる。 • メッセージログに KDJE39188-E のメッセージを出力する。

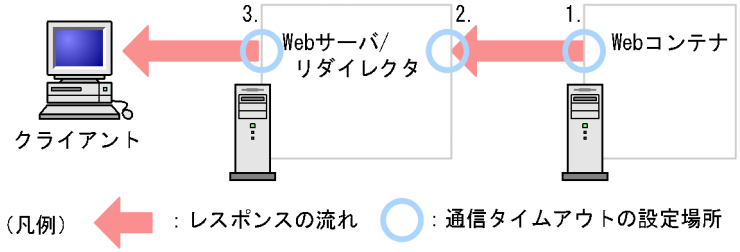
注※

javax.servlet.ServletRequest の，getParameter メソッド，getParameterMap メソッド，getParameterNames メソッド，getParameterValues メソッドを使用している場合を指します。

5.6.2　レスポンス送受信時の通信タイムアウト

ここでは，Web サーバ連携機能を使用する場合のレスポンス送受信時の通信タイムアウト設定について説明します。レスポンス送受信時の，通信タイムアウトの設定場所を次の図に示します。

図 5-19　レスポンス送信時の通信タイムアウトの設定場所（Web サーバ連携機能を使用する場合）



Web サーバ連携機能を使用する場合，図中の○印の場所に通信タイムアウトを設定します。通信タイムアウトを設定する場所について説明します。なお，図中の項番は次の説明の項番と対応しています。

1. Web コンテナでのレスポンス送信時（Web コンテナ－リダイレクタ）

   Web コンテナからリダイレクタにレスポンスを送信するときに，通信タイムアウトを設定します。通信タイムアウトは，Web コンテナで設定します。

   Web コンテナでのレスポンス送信時に通信にタイムアウトを設定することによって検知できる障害については，「5.6.2(1)　Web コンテナでのレスポンス送信時」を参照してください。

2. リダイレクタでのレスポンス受信時（Web コンテナ－リダイレクタ）

   Web コンテナからのレスポンスを，リダイレクタで受信するときに，通信タイムアウトを設定します。通信タイムアウトは，リダイレクタで設定します。

   リダイレクタでのレスポンス受信時に通信タイムアウトを設定することによって検知できる障害については，「5.6.2(2)　リダイレクタでのレスポンス受信時」を参照してください。

3. Web サーバでのレスポンス送信時（Web サーバ－クライアント）

   Web サーバからクライアントにレスポンスを送信するときに，通信タイムアウトを設定します。通信タイムアウトは，Web サーバで設定します。

   Web サーバでのレスポンス送信時に通信にタイムアウトを設定することによって検知できる障害については，「5.6.2(3)　Web サーバでのレスポンス送信時」を参照してください。

(1)　Web コンテナでのレスポンス送信時

Web コンテナでは，リダイレクタへのレスポンス送信処理に対して，タイムアウト時間を設定できます。設定した通信タイムアウトによって，リダイレクタ側で障害が発生したことを検知できます。検知できる障害を次に示します。

検知できる障害

- リダイレクタが稼働するホストがダウンした
- Web コンテナ－リダイレクタ間でネットワーク障害が発生した

また，通信タイムアウトが発生すると，KDJE39507-E（レスポンス送信処理でタイムアウト発生）がメッセージログに出力されます。通信タイムアウトが発生する条件と発生後の動作について次の表に示します。

表 5-14　通信タイムアウトが発生する条件と発生後の動作

通信タイムアウトが発生するタイミング	通信タイムアウト発生後のメソッドの動作	通信タイムアウト発生後のサーブレットまたは JSP の動作
サーブレット内で，javax.servlet.ServletResponse クラスの getOutputStream メソッドによって得られた javax.servlet.ServletOutputStream クラスのメソッドを使用してレスポンスデータをクライアントに送信したとき	java.net.SocketTimeoutException の例外が発生する。	リダイレクタとのコネクションがクローズされるため，リクエストデータおよびレスポンスデータの送受信ができなくなる。
サーブレット内で，javax.servlet.ServletResponse クラスの getWriter メソッドによって得られた java.io.PrintWriter クラスのメソッドを使用して，レスポンスデータをクライアントに送信したとき	送信処理が中断されて，リターンする。	• java.io.PrintWriter クラスの checkError メソッドが true を返す。 • リダイレクタとのコネクションがクローズされるため，リクエストデータおよびレスポンスデータの送受信ができなくなる。
JSP 内で，javax.servlet.jsp.JspWriter クラスのメソッドを使用してレスポンスデータをクライアントに送信したとき	java.net.SocketTimeoutException の例外が発生する。	リダイレクタとのコネクションがクローズされるため，リクエストデータおよびレスポンスデータの送受信ができなくなる。
静的コンテンツのレスポンスデータをクライアントに送信したとき	－	－

（凡例）－：該当しない

(2)　リダイレクタでのレスポンス受信時

リダイレクタは，Web コンテナにリクエストを送信すると，Web コンテナからのレスポンスを待ちます。このレスポンスの待ち時間に，タイムアウトを設定できます。設定した通信タイムアウトによって，Web コンテナ側で障害が発生したことを検知できます。検知できる障害を次に示します。

検知できる障害

- Web コンテナが稼働するホストがダウンした
- Web コンテナ－リダイレクタ間でネットワーク障害が発生した
- Web コンテナ内の Web アプリケーションで障害が発生した

Web アプリケーションで発生する障害には次のようなものがあります。

Web アプリケーションで発生する障害

- サーブレット/JSP 処理内の無限ループによって，レスポンスが返されない
- サーブレット/JSP の延長で Enterprise Bean，データベースなどが呼び出され，それによる応答待ちをしている
- Web アプリケーションでデッドロックが発生している

- アクセスピーク時にサーバの処理が追いつかないで滞っている

リダイレクタでの通信タイムアウト後の動作

通信タイムアウトが発生すると，リダイレクタは，Web コンテナとのコネクションを切断し，クライアントにステータスコード 500 のエラーを返します。

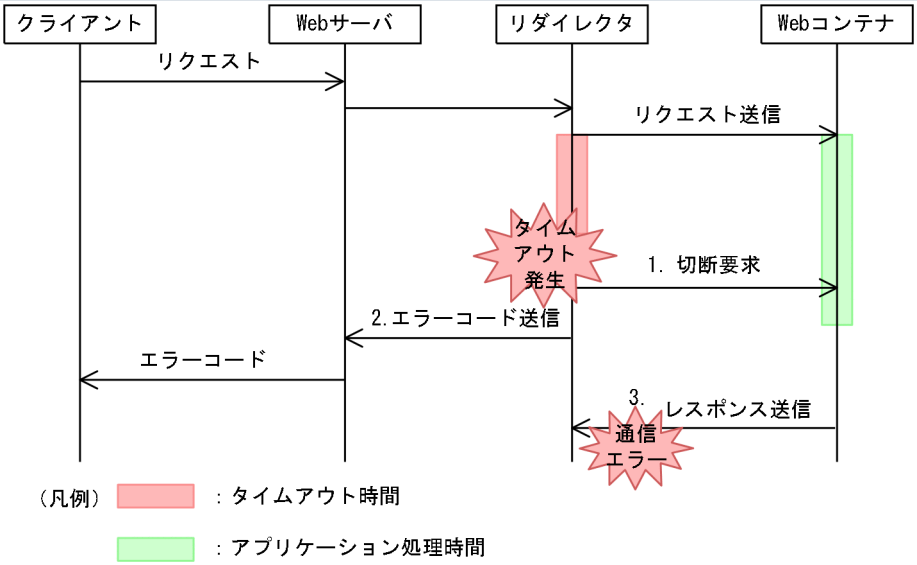
ポイント

アプリケーション処理中にタイムアウトが発生したときの動作

Web コンテナでの処理中にリダイレクタがタイムアウトしていても，Web コンテナでは，リダイレクタがタイムアウトしていることを検知できません。

リダイレクタでのタイムアウトを検知できるのは，Web コンテナの処理が完了し，リダイレクタへレスポンスを転送するときとなります。しかし，この時点では，すでにリダイレクタが Web コンテナとのコネクションを切断しているため，レスポンス送信はエラーとなります。アプリケーション処理中にタイムアウトが発生したときの動作について次の図に示します。

図 5-20　アプリケーション処理中にタイムアウトが発生したときの動作



図について説明します。

1. リダイレクタでタイムアウトが発生すると，Web コンテナとのコネクションを切断する要求を出します。
2. リダイレクタは Web サーバに対して，エラーコードを送信します。
3. Web コンテナでのアプリケーションの処理が完了すると，リダイレクタに対して，レスポンスを送信しますが，1.で，すでにリダイレクタと Web コンテナのコネクションは切断されているため，通信エラーが発生します。

(3) Web サーバでのレスポンス送信時

Web サーバでは，クライアントへのデータ送信処理に対して，タイムアウト時間を設定できます。設定した通信タイムアウトによって，クライアント側で障害が発生したことを検知できます。検知できる障害を次に示します。

検知できる障害

- クライアントが稼働するホストがダウンした
- クライアント－Web サーバ間のネットワーク障害が発生した
- クライアントアプリケーションで障害が発生した

5.6.3 通信タイムアウトの設定

ここでは，クライアント－Web サーバ間，および Web サーバ（リダイレクタ）－Web コンテナ間の通信タイムアウトの設定について説明します。

通信タイムアウトは，リクエストの送受信時，またはレスポンスの送受信時に設定します。通信タイムアウトの設定は，Smart Composer の使用の有無によって設定方法が異なります。通信タイムアウトの設定方法の参照先について，次の表に示します。

表 5-15 通信タイムアウト（Web サーバ連携）の設定方法の参照先

設定するタイミング	Smart Composer の使用	
	使用する	使用しない
リクエスト送受信時	5.6.4	5.6.5
レスポンスの送受信時	5.6.6	5.6.7

なお，通信タイムアウトは，EJB を呼び出す EJB クライアント側でも設定できます。EJB クライアント側での設定は，J2EE アプリケーション開発時に設定します。設定方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「2.11.5 RMI-IIOP 通信のタイムアウト」を参照してください。

5.6.4 リクエスト送受信時の通信タイムアウトの設定（Smart Composer 機能を使用する場合）

リクエスト送受信時の通信タイムアウトは，クライアント－Web サーバ間，リダイレクター－Web コンテナ間に設定します。

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1) Web サーバでのリクエスト受信時の設定

クライアントからのリクエストを Web サーバで受信するときの通信タイムアウトは，Web サーバに設定します。Web サーバには，HTTP Server を使用できます。

(a) 設定方法

クライアントから転送されたリクエストの受信処理の通信タイムアウトを，httpsd.conf に設定してください。

- リクエスト受信処理の待ち時間

  Timeout ディレクティブに，クライアントからのリクエスト受信処理の待ち時間（秒）を設定します。Timeout ディレクティブのデフォルト値は 300 秒です。なお，ここに設定する値は，クライアントへのデータ送信処理に対する通信タイムアウトと共用です。クライアントへのデータ送信処理に対する通信タイムアウトについては，「5.6.6(3) Web サーバでのレスポンス送信時の設定」を参照してください。

(b) 設定時の注意

Web サーバでのリクエスト受信時に設定するタイムアウト値については，クライアント－Web サーバ間のネットワーク構成，およびトラフィックの状態を考慮し，障害が発生したと判断できる時間を設定してください。

(2) リダイレクタでのリクエスト送信時の設定

リダイレクタから Web コンテナにリクエストを送信するときには，Web コンテナとのコネクションを確立してからリクエストを送信します。リダイレクタから Web コンテナにリクエストを送信するときの通信タイムアウトは，コネクション確立時，およびリクエスト送信時に設定できます。また，コネクションの確立およびリクエストヘッダの送信に失敗した場合のリトライ回数を設定できます。

(a) 設定方法

リダイレクタからの，コネクション確立およびリクエスト送信処理の通信タイムアウトおよびリトライ回数を，簡易構築定義ファイルに設定してください。

- コネクション確立の通信タイムアウト

  論理 Web サーバ（web-server）の<configuration>タグ内に，JkConnectTimeout パラメタで，Web コンテナへのコネクション確立処理の待ち時間（秒）を設定します。JkConnectTimeout パラメタのデフォルト値は 30 秒です。

- リクエスト送信処理のタイムアウト

  論理 Web サーバ（web-server）の<configuration>タグ内に，JkSendTimeout パラメタで，Web コンテナへのリクエスト送信処理の待ち時間（秒）を設定します。JkSendTimeout パラメタのデフォルト値は 100 秒です。

- コネクション確立およびリクエスト送信のリトライ回数

論理 Web サーバ（web-server）の<configuration>タグ内に，JkRequestRetryCount パラメタで，Web コンテナへのコネクション確立およびリクエスト送信のリトライ回数を設定します。JkRequestRetryCount パラメタのデフォルト値は 3 回です。

(b) 設定時の注意

リダイレクタでのリクエスト送信時に設定する通信タイムアウト値，およびリトライ回数については，次の点に考慮して設定してください。

- リトライ回数には，初回のコネクション確立およびリクエスト送信が含まれます。このため，初回のコネクション確立またはリクエスト送信処理で失敗した場合，コネクション確立またはリクエスト送信処理のリトライは，2 回目としてカウントされます。
- コネクション確立およびリクエスト送信処理の通信タイムアウト時間に 0 を指定した場合，または TCP が持つコネクション確立およびデータ送信の再送タイマより長い時間を設定した場合は，通信タイムアウト時間は TCP の持つタイムアウト時間が適用されます。

(3) Web コンテナでのリクエスト受信時の設定

リダイレクタからのリクエストを Web コンテナで受信するときの通信タイムアウトは，Web コンテナに設定します。

(a) 設定方法

リダイレクタから転送されたリクエストの受信処理の通信タイムアウトを，簡易構築定義ファイルに設定してください。

- リクエスト受信処理のタイムアウト

  論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.connector.ajp13.receive_timeout パラメタで，リダイレクタからのリクエスト受信処理の待ち時間（秒）を設定します。パラメタのデフォルト値は 100 秒です。

(b) 設定時の注意

Web コンテナでのリクエスト受信時に設定するタイムアウト値については，次の点に考慮して設定してください。

- Web サーバのリクエスト受信時のタイムアウトに設定した時間より，大きい値を設定してください。

  Web サーバのリクエスト受信時のタイムアウトに設定した時間より，小さい値を設定しているときに，クライアントや，クライアント－Web サーバ間でネットワーク障害が発生すると，Web サーバより先に Web コンテナでタイムアウトが発生してしまいます。この場合，障害が発生したのが，Web サーバ側であるのか，クライアント側であるのかが判別できなくなります。
- クライアントからのデータ受信が必要な場合，クライアントとの通信速度を考慮して，データ受信ができる時間を設定してください。

- リダイレクタでの Web コンテナへのデータの送信時に障害が発生した場合，TCP の再送タイマでのタイムアウトによって障害が検知されます。
  なお，UNIX の場合，タイムアウトまでの時間は，OS によって異なります。

5.6.5　リクエスト送受信時の通信タイムアウトの設定（Smart Composer 機能を使用しない場合）

リクエスト送受信時の通信タイムアウトは，クライアント－Web サーバ間，リダイレクター－Web コンテナ間に設定します。

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1)　Web サーバでのリクエスト受信時の設定

クライアントからのリクエストを Web サーバで受信するときの通信タイムアウトは，Web サーバに設定します。Web サーバには，HTTP Server または Microsoft IIS のどちらかを使用できます。

(a)　設定方法

クライアントから転送されたリクエストの受信処理の通信タイムアウトを，次のファイルに設定してください。

- httpsd.conf（HTTP Server の場合）
  Timeout ディレクティブに，クライアントからのリクエスト受信処理の待ち時間（秒）を設定します。Timeout ディレクティブのデフォルト値は 300 秒です。なお，ここに設定する値は，クライアントへのデータ送信処理に対する通信タイムアウトと共用です。クライアントへのデータ送信処理に対する通信タイムアウトについては，「5.6.6(3)　Web サーバでのレスポンス送信時の設定」を参照してください。
- isapi_redirect.conf（Microsoft IIS の場合）
  receive_client_timeout キーに，クライアントからのリクエスト受信処理の待ち時間（秒）を設定します。

(b)　設定時の注意

Web サーバでのリクエスト受信時に設定するタイムアウト値については，クライアント－Web サーバ間のネットワーク構成，およびトラフィックの状態を考慮し，障害が発生したと判断できる時間を設定してください。

(2)　リダイレクタでのリクエスト送信時の設定

リダイレクタから Web コンテナにリクエストを送信するときには，Web コンテナとのコネクションを確立してからリクエストを送信します。リダイレクタから Web コンテナにリクエストを送信するときの通

信タイムアウトは，コネクション確立時，およびリクエスト送信時に設定できます。また，コネクションの確立およびリクエストヘッダの送信に失敗した場合のリトライ回数を設定できます。

(a) 設定方法

リダイレクタからの，コネクション確立およびリクエスト送信処理の通信タイムアウトおよびリトライ回数を，次のファイルに設定してください。

- mod_jk.conf（HTTP Server の場合）
  - コネクション確立の通信タイムアウト

    JkConnectTimeout キーに，Web コンテナへのコネクション確立処理の待ち時間（秒）を設定します。JkConnectTimeout キーのデフォルト値は 30 秒です。
  - リクエスト送信処理のタイムアウト

    JkSendTimeout キーに，Web コンテナへのリクエスト送信処理の待ち時間（秒）を設定します。JkSendTimeout キーのデフォルト値は 100 秒です。
  - コネクション確立およびリクエスト送信のリトライ回数

    JkRequestRetryCount キーに，Web コンテナへのコネクション確立およびリクエスト送信のリトライ回数を設定します。JkRequestRetryCount キーのデフォルト値は 3 回です。
- isapi_redirect.conf（Microsoft IIS の場合）
  - コネクション確立処理の通信タイムアウト

    connect_timeout キーに，Web コンテナへのコネクション確立処理の待ち時間（秒）を設定します。connect_timeout キーのデフォルト値は 30 秒です。
  - リクエスト送信処理のタイムアウト

    send_timeout キーに，Web コンテナへのリクエスト送信処理の待ち時間（秒）を設定します。send_timeout キーのデフォルト値は 100 秒です。
  - コネクション確立およびリクエスト送信のリトライ回数

    request_retry_count キーに，Web コンテナへのコネクション確立およびリクエスト送信のリトライ回数を設定します。request_retry_count キーのデフォルト値は 3 回です。

(b) 設定時の注意

リダイレクタでのリクエスト送信時に設定する通信タイムアウト値，およびリトライ回数については，次の点に考慮して設定してください。

- リトライ回数には，初回のコネクション確立およびリクエスト送信が含まれます。このため，初回のコネクション確立またはリクエスト送信処理で失敗した場合，コネクション確立またはリクエスト送信処理のリトライは，2 回目としてカウントされます。
- コネクション確立およびリクエスト送信処理の通信タイムアウト時間に 0 を指定した場合，または TCP が持つコネクション確立およびデータ送信の再送タイマより長い時間を設定した場合は，通信タイムアウト時間は TCP の持つタイムアウト時間が適用されます。

(3) Web コンテナでのリクエスト受信時の設定

リダイレクタからのリクエストを Web コンテナで受信するときの通信タイムアウトは，Web コンテナに設定します。

(a) 設定方法

リダイレクタから転送されたリクエストの受信処理の通信タイムアウトを，次のファイルに設定してください。

- usrconf.properties

  webserver.connector.ajp13.receive_timeout キーに，リダイレクタからのリクエスト受信処理の待ち時間（秒）を設定します。

(b) 設定時の注意

Web コンテナでのリクエスト受信時に設定するタイムアウト値については，次の点に考慮して設定してください。

- Web サーバのリクエスト受信時のタイムアウトに設定した時間より，大きい値を設定してください。

  Web サーバのリクエスト受信時のタイムアウトに設定した時間より，小さい値を設定しているときに，クライアントや，クライアント－Web サーバ間でネットワーク障害が発生すると，Web サーバより先に Web コンテナでタイムアウトが発生してしまいます。この場合，障害が発生したのが，Web サーバ側であるのか，クライアント側であるのかが判別できなくなります。
- クライアントからのデータ受信が必要な場合，クライアントとの通信速度を考慮して，データ受信ができる時間を設定してください。
- リダイレクタでの Web コンテナへのデータの送信時に障害が発生した場合，TCP の再送タイマでのタイムアウトによって障害が検知されます。

  なお，UNIX の場合，タイムアウトまでの時間は，OS によって異なります。

5.6.6 レスポンス送受信時の通信タイムアウトの設定（Smart Composer 機能を使用する場合）

レスポンス送受信時の通信タイムアウトは，Web コンテナ－リダイレクタ間，Web サーバ－クライアント間に設定します。

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1) Web コンテナでのレスポンス送信時の設定

Web コンテナからリダイレクタにレスポンス送信処理するときの通信タイムアウトは，Web コンテナに設定します。Web コンテナからのレスポンス送信時の待ち時間は，簡易構築定義ファイルに設定してください。

- **レスポンス送信処理のタイムアウト**

  論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.connector.ajp13.send_timeout パラメタで，Web コンテナからのレスポンス送信時のタイムアウト時間（秒）を設定します。パラメタのデフォルト値は 600 秒です。

(2)　リダイレクタでのレスポンス受信時の設定

Web コンテナからのレスポンスを受信するときの通信タイムアウトをリダイレクタに設定します。

(a)　設定方法

Web コンテナからのレスポンス待ち時間を，簡易構築定義ファイルに設定してください。

- **レスポンス受信処理のタイムアウト**

  論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，worker.<ワーカ名>.receive_timeout パラメタで，ワーカごとのレスポンス待ち時間（秒）を設定します。パラメタのデフォルト値は 3600 秒です。J2EE アプリケーション単位で通信タイムアウトを設定したい場合は，J2EE アプリケーションごとにワーカを定義してマッピングしてください。

(b)　設定時の注意

リダイレクタでのレスポンス受信時のタイムアウトでは，Web アプリケーションの障害を検知できます。そのため，運用状況によっては，Web アプリケーションの処理に必要となる時間を考慮して，障害検知と判断できる値を通信タイムアウトに設定する必要があります。

リダイレクタでのレスポンス受信時に設定するタイムアウト値については，次の点に考慮して設定してください。

- 通信タイムアウトには，Web アプリケーションの処理に必要な時間より長い時間を設定してください。

  設定したタイムアウトの時間が，Web アプリケーションの処理時間より短い場合，Web アプリケーションは正常に処理されていても，リダイレクタ側ではタイムアウトが発生したと判断して，クライアントにエラーが返されます。

- Web コンテナのピーク時の，同時実行スレッド数の制御による待ち時間を考慮してください。

  同時実行スレッド数の制御によって，Web コンテナのピーク時には，処理待ちリクエストの発生が考えられます。このため，同時実行スレッド数の制御を設定している場合，リクエストの処理時間が延長されることも考慮して，通信タイムアウトを設定しておく必要があります。Web コンテナ単位の同時実行スレッド数の制御の設定については，「5.11　Web コンテナ単位での同時実行スレッド数の制御」を，Web アプリケーション単位や URL グループ単位の同時実行スレッド数の制御の設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.15　Web アプリケーション単位での同時実行スレッド数の制御」や「2.16　URL グループ単位での同時実行スレッド数の制御」を参照してください。

- Web コンテナでのリダイレクタへのデータの送信時に障害が発生した場合，TCP の再送タイマでのタイムアウトによって障害が検知されます。

なお，UNIX の場合，タイムアウトまでの時間は，OS によって異なります。

(3) Web サーバでのレスポンス送信時の設定

クライアントにレスポンスを送信するときの通信タイムアウトは，Web サーバに設定します。

(a) 設定方法

クライアントからのレスポンス待ち時間を，httpsd.conf に設定してください。

- **データ送信処理の通信タイムアウト**

  Timeout ディレクティブに通信タイムアウトを設定してください。なお，Timeout ディレクティブで指定する通信タイムアウトは，リクエスト受信処理に対する通信タイムアウトおよびレスポンス送信処理に対する通信タイムアウトの両方の時間として指定します。このため，リクエスト受信処理の通信タイムアウトとレスポンス送信処理の通信タイムアウトで異なる時間を設定することはできません。

  クライアントからのリクエスト受信処理に対する通信タイムアウトについては，「5.6.4(1) Web サーバでのリクエスト受信時の設定」を参照してください。

(b) 設定時の注意

Web サーバでのレスポンス送信時に設定するタイムアウト値については，クライアント－Web サーバ間のネットワーク構成，およびトラフィックの状態を考慮し，クライアントとデータの送受信をするのに十分な時間を設定してください。

また，Web サーバでのレスポンス送信時のタイムアウトは，Web コンテナでのレスポンス送信時のタイムアウト値よりも小さな値を設定してください。Web サーバでのレスポンス送信時のタイムアウトを，Web コンテナでのレスポンス送受信時のタイムアウト値よりも大きな値を設定しているときに，クライアント－Web サーバ間で障害が発生すると，Web サーバでのクライアントへのレスポンス送信のタイムアウトよりも先に，Web コンテナでのリダイレクタへのレスポンス送信でタイムアウトが発生するおそれがあります。この場合，障害が発生したのが，クライアント－Web サーバ間か，リダイレクター－Web コンテナ間か判別できなくなります。

5.6.7 レスポンス送受信時の通信タイムアウトの設定（Smart Composer 機能を使用しない場合）

レスポンス送受信時の通信タイムアウトは，Web コンテナーリダイレクタ間，Web サーバークライアント間に設定します。

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1) Web コンテナでのレスポンス送信時の設定

Web コンテナからリダイレクタにレスポンス送信処理するときの通信タイムアウトは，Web コンテナに設定します。Web コンテナからのレスポンス送信時の待ち時間は，次のファイルに，を設定してください。

- usrconf.properties

  webserver.connector.ajp13.send_timeout キーに，Web コンテナからのレスポンス送信時のタイムアウト時間（秒）を設定します。デフォルト値は 600 秒です。

(2) リダイレクタでのレスポンス受信時の設定

Web コンテナからのレスポンスを受信するときの通信タイムアウトをリダイレクタに設定します。

(a) 設定方法

次のファイルに，Web コンテナからのレスポンス待ち時間を設定してください。

- workers.properties

  worker.<ワーカ名>.receive_timeout キーに，ワーカごとのレスポンス待ち時間（秒）を設定します。J2EE アプリケーション単位で通信タイムアウトを設定したい場合は，J2EE アプリケーションごとにワーカを定義してマッピングしてください。

(b) 設定時の注意

リダイレクタでのレスポンス受信時のタイムアウトでは，Web アプリケーションの障害を検知できます。そのため，運用状況によっては，Web アプリケーションの処理に必要となる時間を考慮して，障害検知と判断できる値を通信タイムアウトに設定する必要があります。

リダイレクタでのレスポンス受信時に設定するタイムアウト値については，次の点に考慮して設定してください。

- 通信タイムアウトには，Web アプリケーションの処理に必要な時間より長い時間を設定してください。

  設定したタイムアウトの時間が，Web アプリケーションの処理時間より短い場合，Web アプリケーションは正常に処理されていても，リダイレクタ側ではタイムアウトが発生したと判断して，クライアントにエラーが返されます。

- Web コンテナのピーク時の，同時実行スレッド数の制御による待ち時間を考慮してください。

  同時実行スレッド数の制御によって，Web コンテナのピーク時には，処理待ちリクエストの発生が考えられます。このため，サーバ管理コマンドを使用して同時実行スレッド数の制御を設定している場合，リクエストの処理時間が延長されることも考慮して，通信タイムアウトを設定しておく必要があります。Web コンテナ単位の同時実行スレッド数の制御の設定については，「5.11 Web コンテナ単位での同時実行スレッド数の制御」を，Web アプリケーション単位や URL グループ単位の同時実行スレッド数の制御の設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.15 Web アプリケーション単位での同時実行スレッド数の制御」や「2.16 URL グループ単位での同時実行スレッド数の制御」を参照してください。

- Web コンテナでのリダイレクタへのデータの送信時に障害が発生した場合，TCP の再送タイマでのタイムアウトによって障害が検知されます。

  なお，UNIX の場合，タイムアウトまでの時間は，OS によって異なります。

(3) Web サーバでのレスポンス送信時の設定

クライアントにレスポンスを送信するときの通信タイムアウトを Web サーバに設定できます。

(a) 設定方法

次のファイルに，クライアントからのレスポンス待ち時間を設定してください。

- httpsd.conf（HTTP Server の場合）

  Timeout ディレクティブに通信タイムアウトを設定してください。なお，Timeout ディレクティブで指定する通信タイムアウトは，リクエスト受信処理に対する通信タイムアウトおよびレスポンス送信処理に対する通信タイムアウトの両方の時間として指定します。このため，リクエスト受信処理の通信タイムアウトとレスポンス送信処理の通信タイムアウトで異なる時間を設定することはできません。

  クライアントからのリクエスト受信処理に対する通信タイムアウトについては，「5.6.4(1) Web サーバでのリクエスト受信時の設定」を参照してください。

- MinFileBytesPerSec プロパティ（Microsoft IIS の場合）

  MinFileBytesPerSec プロパティに通信タイムアウトを設定してください。MinFileBytesPerSec プロパティには，Web サーバからクライアントへ送信するレスポンスデータのスループット（バイト/秒）を指定します。レスポンスデータのスループットが MinFileBytesPerSec に設定した値を下回った場合に，通信タイムアウトが発生します。なお，通信タイムアウトのデフォルトは 240 バイト/秒です。

  設定の詳細については，Microsoft IIS のマニュアルを参照してください。

(b) 設定時の注意

Web サーバでのレスポンス送信時に設定するタイムアウト値については，クライアント－Web サーバ間のネットワーク構成，およびトラフィックの状態を考慮し，クライアントとデータの送受信をするのに十分な時間を設定してください。

また，Web サーバでのレスポンス送信時のタイムアウトは，Web コンテナでのレスポンス送信時のタイムアウト値よりも小さな値を設定してください。Web サーバでのレスポンス送信時のタイムアウトを，Web コンテナでのレスポンス送受信時のタイムアウト値よりも大きな値を設定しているときに，クライアント－Web サーバ間で障害が発生すると，Web サーバでのクライアントへのレスポンス送信のタイムアウトよりも先に，Web コンテナでのリダイレクタへのレスポンス送信でタイムアウトが発生するおそれがあります。この場合，障害が発生したのが，クライアント－Web サーバ間か，リダイレクター－Web コンテナ間か判別できなくなります。

5.7　IP アドレス指定（Web サーバ連携）

この節では，Web サーバ連携での IP アドレス指定による Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 5-16　この節の構成（IP アドレス指定（Web サーバ連携））

分類	タイトル	参照先
解説	バインド先アドレス設定機能	5.7.1
設定	実行環境での設定（J2EE サーバの設定）	5.7.2
注意事項	Web サーバ連携での IP アドレス指定をする場合の注意事項	5.7.3

注　「実装」および「運用」について，この機能固有の説明はありません。

5.7.1　バインド先アドレス設定機能

Web コンテナでは，Web サーバ連携で利用する IP アドレスを明示的に指定できます。これを，**バインド先アドレス設定機能**といいます。この機能を使用することで，複数の物理ネットワークインタフェースを持つホスト，または一つの物理ネットワークインタフェースに対して，ホストで実行する場合に，特定の一つの IP アドレスだけを使用するよう設定できます。

バインドする IP アドレスは，J2EE サーバのプロパティをカスタマイズして設定します。J2EE サーバの動作設定のカスタマイズについては，「5.7.2　実行環境での設定（J2EE サーバの設定）」を参照してください。

5.7.2　実行環境での設定（J2EE サーバの設定）

Web サーバ連携での IP アドレス指定を使用する場合，J2EE サーバの設定が必要です。

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web サーバ連携での IP アドレス指定の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.ajp13.bind_host

　Web サーバ連携機能を使用する場合の IP アドレスまたはホスト名を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

5.7.3 Web サーバ連携での IP アドレス指定をする場合の注意事項

Web サーバ連携での IP アドレス指定をする場合の注意事項を次に示します。

- ホスト名または IP アドレスを設定した場合は，指定された IP アドレスへの接続要求しか受け付けません。IP アドレスを設定する代わりに，ワイルドカードアドレスを指定することで，そのホスト上の任意の IP アドレスへの接続を受け付けることができます。デフォルトでは，ワイルドカードアドレスを使用する設定になっています。なお，ワイルドカードアドレスを使用する場合は，次のことに注意してください。
  - 指定されたホスト名が，hosts ファイルまたは DNS などで解決できない場合は，ワイルドカードアドレスを使用してサーバを起動します。
  - 指定されたホスト名，または IP アドレスがリモートホストの場合，ワイルドカードアドレスを使用してサーバを起動します。

5.8　エラーページのカスタマイズ（Web サーバ連携）

クライアントから，存在しないリソースや例外が発生したサーブレットなどにアクセスがあると，Web コンテナはエラーステータスコードを返します。クライアント側では，Web コンテナから返されたエラーステータスコードに対応するエラーページが表示されます。アプリケーションサーバでは，クライアントで表示されるエラーページの代わりに，ユーザが作成したページをクライアントに表示させることができます。これを，**エラーページのカスタマイズ**と呼びます。

この節では，Web サーバと連携した場合の，Web サーバの機能を使用したエラーページのカスタマイズについて説明します。

この節の構成を次の表に示します。

表 5-17　この節の構成（エラーページのカスタマイズ（Web サーバ連携））

分類	タイトル	参照先
解説	エラーページのカスタマイズの概要	5.8.1
	エラーページのカスタマイズの仕組み	5.8.2
設定	実行環境での設定（Smart Composer 機能を使用する場合）	5.8.3
	実行環境での設定（Smart Composer 機能を使用しない場合）	5.8.4
注意事項	エラーページのカスタマイズの注意事項	5.8.5

注　「実装」および「運用」について，この機能固有の説明はありません。

注意事項

Web サーバの機能を使用したエラーページのカスタマイズは，Web サーバ連携機能を使用する場合に使用できる機能です。Web サーバが HTTP Server の場合だけ使用できます。Microsoft IIS の場合は使用できません。

5.8.1　エラーページのカスタマイズの概要

エラーページをカスタマイズするには，Servlet 仕様で規定されている web.xml の<error-page>タグを使用する方法と，Web サーバの機能を使用する方法があります。ただし，リダイレクタが Web コンテナとの通信に失敗した場合など，リダイレクタがエラーを返す場合のエラーページは，web.xml の<error-page>タグを使用する方法ではカスタマイズできません。リダイレクタがエラーを返す場合のエラーページのカスタマイズは，Web サーバの機能を使用する方法で行ってください。エラー発生場所とエラーページのカスタマイズ方法の対応を表に示します。

表 5-18　エラー発生場所とエラーページのカスタマイズ方法の対応

エラー発生場所	カスタマイズ方法	
	Web サーバの機能を使用する方法	web.xml の<error-page>タグを使用する方法
Web コンテナ	○	○
リダイレクタ	○	×

（凡例）　○：カスタマイズできる　×：カスタマイズできない

なお，Web コンテナでエラーが発生する条件と，対応するエラーステータスコードについては「付録 C エラーステータスコード」を参照してください。

5.8.2　エラーページのカスタマイズの仕組み

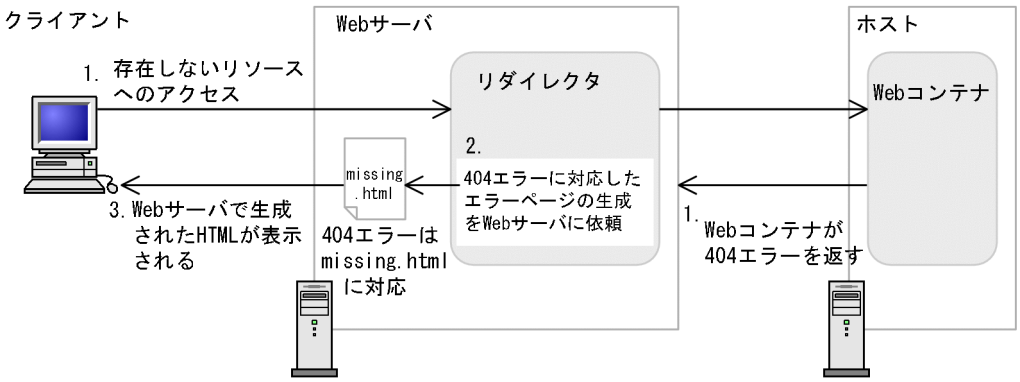
エラーページのカスタマイズの処理の仕組みを，Web コンテナでエラーが発生した場合，およびリダイレクタでエラーが発生した場合のそれぞれについて説明します。

Web コンテナでエラーが発生した場合

Web コンテナからのエラーステータスコードは，リダイレクタが受信します。リダイレクタは，エラーページの作成を Web サーバに委任し，Web サーバでエラーステータスコードに対応したユーザ作成のページをクライアントに送信します。これによって，クライアントでは，ユーザが作成したページが表示されます。

エラーページのカスタマイズの処理の流れを次の図に示します。

図 5-21　ユーザ作成のエラーページ表示の流れ（Web サーバの機能を使用する場合）



図中の 1.～3.について説明します。

1. クライアントから存在しないリソースへのアクセスがあると，Web コンテナでは，404 エラーを Web サーバに返します。
2. リダイレクタは 404 エラーを受け取ると，設定情報※を基にして，404 エラーに対応するエラーページの生成を Web サーバに依頼します。
3. Web サーバでは，設定情報※を基に 404 エラーに対応するエラーページ「missing.html」を，クライアントに返します。

リダイレクタでエラーが発生した場合

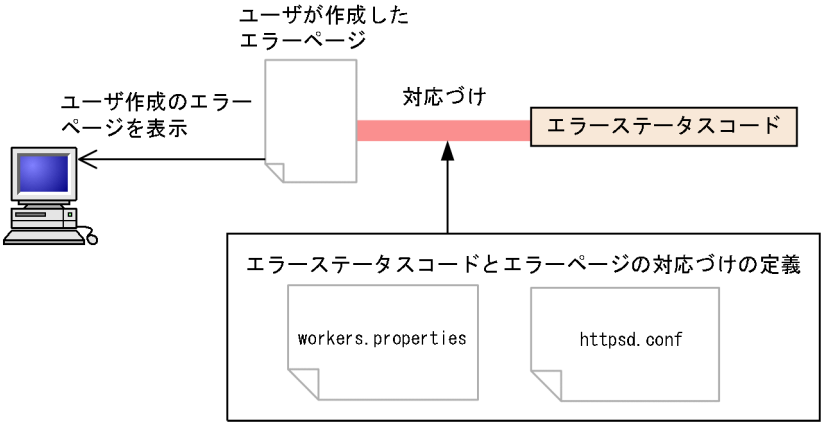
リダイレクタでエラーが発生すると，リダイレクタは設定情報を基にして，発生したエラーに対応するエラーページの生成を Web サーバに依頼します。Web サーバでは，設定情報※を基にエラーステータスコードに対応したユーザ作成のページをクライアントに送信します。これによって，クライアントでは，ユーザが作成したページが表示されます。

注※

エラーページをカスタマイズするためには，あらかじめ，エラーステータスコードとエラーページの対応づけをしておく必要があります。

対応づけの概要を次の図に示します。

図 5-22　エラーステータスコードとエラーページの対応づけ（Web サーバの機能を使用する場合）



エラーが発生したときに，エラーステータスコードが表示されたエラーページの代わりにユーザが作成したエラーページを表示させるためには，ユーザが作成したエラーページを特定のエラーステータスコードと対応づけます。該当するエラーステータスコードのエラーが発生した場合には，Web サーバ（HTTP Server）に設定された情報を基に，エラーステータスコードに対応したエラーページをクライアントに送信します。

5.8.3　実行環境での設定（Smart Composer 機能を使用する場合）

ここでは，エラーページのカスタマイズの設定について説明します。

(1)　設定方法

エラーステータスコードとエラーページとの対応づけは，次のファイルで定義します。

- 簡易構築定義ファイル

  論理 Web サーバ（web-server）の<configuration>タグ内に，worker.<ワーカ名>.delegate_error_code パラメタで，エラーページに対応づけたいエラーステータスコードを設定します。

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

- httpsd.conf

  ErrorDocument ディレクティブで，エラーステータスコードと，対応するエラーページのファイル名を対応づけます。

  httpsd.conf（HTTP Server 定義ファイル）については，マニュアル「HTTP Server」を参照してください。

(a)　エラーステータスコード設定時の注意事項

簡易構築定義ファイルでエラーステータスコードを指定する場合の注意事項を次に示します。

- エラーステータスコードは，ワーカ単位で指定します。
- エラーステータスコードを指定できるワーカのタイプは，「ajp13」だけです。ワーカのタイプが「lb」（ラウンドロビンに基づく負荷分散をするときの設定），および「post_size_lb」（POST データサイズに基づく振り分けをするときの設定）の場合，指定した内容は無視されます。
- 指定できるエラーステータスコードは，次の表のとおりです。これ以外のエラーステータスコードにエラーページを対応づけることはできません。

表 5-19　エラーページと対応づけできるエラーステータスコード

エラーステータスコード	説明句
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type

エラーステータスコード	説明句
416	Requested Range Not Satisfiable
417	Expectation Failed
422	Unprocessible Entity
423	Locked
424	Failed Dependency
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported
507	Insufficient Storage
510	Not Extended

(b) ErrorDocument ディレクティブ指定時の注意事項

httpsd.conf で ErrorDocument ディレクティブを指定する場合の注意事項を次に示します。

- ErrorDocument ディレクティブにローカル URL を使用する場合には，リダイレクタが Web コンテナに転送しない URL を指定する必要があります。
- ルートコンテキストの使用時など，リダイレクタの設定で「/*」という URL パターンをワーカにマッピングしている場合，すべてのリクエストが Web コンテナに転送されます。そのため，ErrorDocument ディレクティブには，Web コンテナ上のリソースを，完全 URL を使用して設定してください。

  ルートコンテキストを使用する場合で，エラーステータスコード 404 発生時に，Web コンテナ上のルートコンテキスト配下の error404.jsp を表示させるための設定例を次に示します。hostA は，Web サーバ稼働ホストです。

  （例）

  ```
  ErrorDocument 404 http://hostA/error404.jsp
  ```

  また，このとき Web コンテナが停止している場合には，リダイレクタはエラーステータスコード 500 のエラーを返します。このため，Web コンテナが停止している際のエラーページをカスタマイズするには，ErrorDocument ディレクティブで，エラーステータスコード 500 に対して別の Web サーバのリソースを，完全 URL で指定する必要があります。

(2) 設定例

エラーページのカスタマイズの例を次に示します。

簡易構築定義ファイルの例

```
 :
<param>
  <param-name>worker.list</param-name>
  <param-value>worker1</param-value>
</param>
<param>
  <param-name>worker.worker1.type</param-name>
  <param-value>ajp13</param-value>
</param>
<param>
  <param-name>worker.worker1.host</param-name>
  <param-value>host1</param-value>
</param>
<param>
  <param-name>worker.worker1.delegate_error_code</param-name>
  <param-value>404</param-value>
</param>
 :
```

worker.<ワーカ名>.delegate_error_code パラメタに，エラーステータスコード「404（Not Found)」を定義しています。

httpsd.conf の例

```
# httpsd.confの記述#
#           :
ErrorDocument 404 /missing.html
```

エラーステータスコードと，対応するエラーページのファイル名を対応づけます。エラーステータスコード「404（Not Found)」のエラーが発生した場合に，missing.html ファイルを表示するようになります。

ErrorDocument ディレクティブの詳細については，マニュアル「HTTP Server」を参照してください。

5.8.4　実行環境での設定（Smart Composer 機能を使用しない場合）

ここでは，エラーページのカスタマイズの設定について説明します。

(1)　設定方法

エラーステータスコードとエラーページとの対応づけは，次のファイルで定義します。

- workers.properties

  worker.<ワーカ名>.delegate_error_code キーに，エラーページに対応づけたい，エラーステータスコードを設定します。

  workers.properties（ワーカ定義ファイル）については，「13.2.4　workers.properties（ワーカ定義ファイル)」を参照してください。

- httpsd.conf

ErrorDocument ディレクティブで，エラーステータスコードと，対応するエラーページのファイル名を対応づけます。

httpsd.conf（HTTP Server 定義ファイル）については，マニュアル「HTTP Server」を参照してください。

workers.properties 設定時の注意事項

- エラーステータスコードは，ワーカ単位で指定します。
- エラーステータスコードを指定できるワーカのタイプは，「ajp13」だけです。ワーカのタイプが「lb」（ラウンドロビンに基づく負荷分散をするときの設定）の場合，指定した内容は無視されます。
- 指定できるエラーステータスコードは，次の表のとおりです。これ以外のエラーステータスコードにエラーページを対応づけることはできません。

表 5-20　エラーページと対応づけできるエラーステータスコード

エラーステータスコード	説明句
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
422	Unprocessible Entity
423	Locked

エラーステータスコード	説明句
424	Failed Dependency
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported
507	Insufficient Storage
510	Not Extended

ErrorDocument ディレクティブ指定時の注意事項

- ErrorDocument ディレクティブにローカル URL を使用する場合には，リダイレクタが Web コンテナに転送しない URL を指定する必要があります。
- ルートコンテキストの使用時など，リダイレクタの設定で「/*」という URL パターンをワーカにマッピングしている場合，すべてのリクエストが Web コンテナに転送されます。そのため，ErrorDocument ディレクティブには，Web コンテナ上のリソースを，完全 URL を使用して設定してください。

  ルートコンテキストを使用する場合で，エラーステータスコード 404 発生時に，Web コンテナ上のルートコンテキスト配下の error404.jsp を表示させるための設定例を次に示します。hostA は，Web サーバ稼働ホストです。

  ErrorDocument 404 http://hostA/error404.jsp

  また，このとき Web コンテナが停止している場合には，リダイレクタはエラーステータスコード 500 のエラーを返します。このため，Web コンテナが停止している際のエラーページをカスタマイズするには，ErrorDocument ディレクティブで，エラーステータスコード 500 に対して別の Web サーバのリソースを，完全 URL で指定する必要があります。

(2) 設定例

エラーページのカスタマイズの例を次に示します。

workers.properties の例

```
# ワーカ定義ファイルの記述
worker.list=worker1

worker.worker1.type=ajp13
worker.worker1.host=host1
worker.worker1.port=8007
worker.worker1.delegate_error_code=404
```

worker.<ワーカ名>.delegate_error_code キーに，エラーステータスコード「404（Not Found)」を定義しています。

httpsd.conf の例

```
# httpsd.confの記述#
#          :
ErrorDocument 404 /missing.html
```

エラーステータスコードと，対応するエラーページのファイル名を対応づけます。エラーステータスコード「404（Not Found)」のエラーが発生した場合に，missing.html ファイルを表示するようになります。

ErrorDocument ディレクティブの詳細については，マニュアル「HTTP Server」を参照してください。

5.8.5 エラーページのカスタマイズの注意事項

Web サーバ連携機能を使用する場合に，Web サーバの機能を使用してエラーページをカスタマイズするときの注意事項を次に示します。

- エラーページのカスタマイズは，Web サーバが HTTP Server の場合だけ使用できます。このため，Microsoft IIS と連携している場合に，workers.properties にエラーページのカスタマイズを設定しても，無効となります。
- Servlet 2.3 仕様に対応した Web アプリケーションでは，サーブレットの仕様で規定されている web.xml の<error-page>タグによるエラーページのカスタマイズ機能を使用している場合，Web コンテナは<error-page>タグに記述されているページへのアクセス結果をステータスコードとして返却します。このため，<error-page>タグに記述されているページへのアクセスでエラーが発生しない場合には，この機能は動作しません。
- エラーページのカスタマイズをするための設定が，ワーカ定義（workers.properties または簡易構築定義ファイル）または httpsd.conf のどちらか一方にしかない場合，設定されているエラーが Web コンテナで発生しても，ユーザが設定したファイルは表示されません。

  エラーページの生成を委任するエラーステータスコードがワーカ定義だけに指定されている（httpsd.conf の ErrorDocument ディレクティブを定義していない）場合，そのエラーステータスコードのエラーが発生したときに返却されるエラーページは，HTTP Server が自動生成したページになります。

5.9　ドメイン名指定でのトップページの表示

デプロイした Web アプリケーションにアクセスするとき，URL にドメイン名を指定するだけで，index.html や index.jsp などの，Web アプリケーションのトップページを表示させることができます。この機能は，Web サーバ連携機能を使用する場合に使用できます。ここでは，index.html や index.jsp などのファイルを，welcome ファイルと呼びます。

この節では，ドメイン名指定でのトップページの表示について説明します。

この節の構成を次の表に示します。

表 5-21　この節の構成（ドメイン名指定でのトップページの表示）

分類	タイトル	参照先
解説	ドメイン名指定でのトップページの表示について	5.9.1
設定	実行環境での設定（Smart Composer 機能を使用する場合）	5.9.2
	実行環境での設定（Smart Composer 機能を使用しない場合）	5.9.3

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

5.9.1　ドメイン名指定でのトップページの表示について

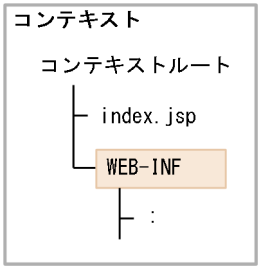
ドメイン名だけでトップページを表示させるには，welcome ファイルを**ルートコンテキスト**に配置する必要があります。ルートコンテキストとは，コンテキストルート※が空文字（コンテキストルートに名称が指定されていない）のコンテキストです。

注※

Web アプリケーションをまとめた管理単位を，**コンテキスト**といいます。このコンテキストのルートパスを**コンテキストルート**といいます。Web アプリケーションにアクセスするときは，コンテキストルートを URL 上に指定します。

コンテキストとコンテキストルートについて，次の図に示します。

図 5-23　コンテキストとコンテキストルート



ドメイン名だけでトップページを表示させるには，次の設定が必要になります。

- リダイレクタの設定

ルートコンテキストには，Web サーバ経由でアクセスします。このため，リダイレクタの URL マッピング定義で，該当する URL がリダイレクトされるように設定する必要があります。mod_jk.conf（HTTP Server の場合）または uriworkermap.properties（Microsoft IIS の場合）に設定します。

- **アプリケーションの設定**

  インポートした J2EE アプリケーションのコンテキストルートに空文字を指定します。

(1) 注意事項

ドメイン名指定でのトップページの表示機能を使用する上での注意事項を次に示します。

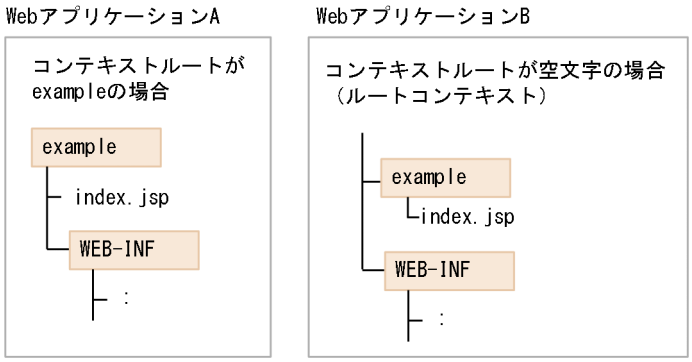
- **コンテキストルートとルートコンテキストが同じ階層を持つ場合にアクセスされる階層**

  コンテキストルートとルートコンテキストが同じ階層を持つ場合，コンテキストルートの階層にアクセスされます。例を次に示します。

  例

  Web アプリケーション A がコンテキストルート example を持ち，Web アプリケーション B のコンテキストルートが空文字で，両方の Web アプリケーション内に「example」という階層を持つ場合の例を説明します。

  図 5-24 コンテキストルートとルートコンテキストが同じ階層を持つ場合にアクセスされる階層を持つ例

  この場合，「http://<ホスト名>/example」にアクセスすると，コンテキストルートを持つ，Web アプリケーション A の example/index.jsp が実行されます。

  ただし，ディレクトリ内に，forward や include がある場合で，例えばディレクトリ内の forward にアクセスすると，ルートコンテキストの index.jsp に forward されます。

- **Web アプリケーション内の構成**

  URL の先頭に「ejb」や「web」を使用することはできません。

  使用できない例

  http://<ホスト名>:<ポート番号>/ejb/

  http://<ホスト名>:<ポート番号>/web/

  このため，ルートコンテキストとしてデプロイする Web アプリケーションでは，「ejb」または「web」が先頭になるような構成にしないでください。

- メッセージ本文での表示方法
  コンソールやログファイルに出力されるメッセージでは，コンテキストルートは空文字で表示されます。

5.9.2　実行環境での設定（Smart Composer 機能を使用する場合）

ここでは，ドメイン名指定でトップページを表示するための設定について説明します。

デプロイした Web アプリケーションにアクセスするとき，URL にドメイン名を指定するだけで，index.html や index.jsp などの，Web アプリケーションのトップページを表示させることができます。

(1)　設定方法

ドメイン名指定でトップページを表示するための設定方法を次に示します。

1. ルートコンテキストを設定します。
   ルートコンテキストとは，コンテキストルートに名称が指定されていないコンテキストのことです。ルートコンテキストの指定は，動作モードによって異なります。
   J2EE アプリケーションのコンテキストルート定義については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「9.11.1　J2EE アプリケーションのコンテキストルート定義」を参照してください。

2. リダイレクタで，ルートコンテキストへのリクエストの振り分けを設定します。
   ルートコンテキストへのリクエストの振り分けは，簡易構築定義ファイルに指定します。
   簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

(2)　設定例

ルートコンテキストへのリクエストの振り分けの設定例について説明します。

URL にドメイン名を指定するだけで，Web アプリケーションのトップページを表示させるためには，リダイレクタの URL マッピング定義で，ルートコンテキストにリクエストが振り分けられるように設定してください。例えば，ルートコンテキストは「worker1」に，「/examples」は worker2 に振り分ける場合には，次のように指定します。

簡易構築定義ファイルの例

```
 :
<param>
  <param-name>JkMount</param-name>
  <param-value>/* worker1</param-value>
  <param-value>/examples/* worker2</param-value>
</param>
 :
```

5.9.3　実行環境での設定（Smart Composer 機能を使用しない場合）

ここでは，ドメイン名指定でトップページを表示するための設定について説明します。

デプロイした Web アプリケーションにアクセスするとき，URL にドメイン名を指定するだけで，index.html や index.jsp などの，Web アプリケーションのトップページを表示させることができます。

(1)　設定方法

ドメイン名指定でトップページを表示するための設定方法を次に示します。

1. ルートコンテキストを設定します。

   ルートコンテキストとは，コンテキストルートに名称が指定されていないコンテキストのことです。ルートコンテキストの指定は，動作モードによって異なります。

   サーバ管理コマンドを使用して J2EE アプリケーションのプロパティを定義するときにルートコンテキストを指定します。コンテキストルートとしてルートコンテキストを設定する場合は，空文字を指定します。J2EE アプリケーションのコンテキストルート定義については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「9.11.1　J2EE アプリケーションのコンテキストルート定義」を参照してください。

2. リダイレクタで，ルートコンテキストへのリクエストの振り分けを設定します。

   ルートコンテキストへのリクエストの振り分けは，Web サーバに HTTP Server を使用している場合は mod_jk.conf に，Microsoft IIS を使用している場合は uriworkermap.properties に指定します。

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

   uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）については，「13.2.3　uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）」を参照してください。

(2)　設定例

ルートコンテキストへのリクエストの振り分けの設定例について説明します。

URL にドメイン名を指定するだけで，Web アプリケーションのトップページを表示させるためには，リダイレクタの URL マッピング定義で，ルートコンテキストにリクエストが振り分けられるように設定してください。例えば，ルートコンテキストは「worker1」に，「/examples」は worker2 に振り分ける場合には，次のように指定します。

mod_jk.conf の例（HTTP Server の場合）

```
JkMount /* worker1
JkMount /examples/* worker2
```

uriworkermap.properties の例（Microsoft IIS の場合）

```
/*=worker1
/examples/*=worker2
```

5.10　Web コンテナへのゲートウェイ情報の通知

この節では，Web コンテナへのゲートウェイ情報の通知について説明します。

ゲートウェイ指定機能によって，Web コンテナにゲートウェイ情報を通知し，welcome ファイルや Form 認証画面に正しくリダイレクトできるようになります。

この節の構成を次の表に示します。

表 5-22　この節の構成（Web コンテナへのゲートウェイ情報の通知）

分類	タイトル	参照先
解説	ゲートウェイ指定機能	5.10.1
設定	実行環境での設定（Smart Composer 機能を使用する場合）	5.10.2
	実行環境での設定（Smart Composer 機能を使用しない場合）	5.10.3
注意事項	Web コンテナにゲートウェイ情報を通知する場合の注意事項	5.10.4

注 「実装」および「運用」について，この機能固有の説明はありません。

5.10.1　ゲートウェイ指定機能

クライアントと，Web サーバとの間に，SSL アクセラレータや負荷分散機などのゲートウェイを配置している場合で，welcome ファイルや Form 認証画面への遷移時などに Web コンテナが自動的にリダイレクトするとき，Web コンテナではゲートウェイの情報を得ることができず，転送先の URL を正しく作成できないことがあります。

これを解決するために，**ゲートウェイ指定機能**を使用します。ゲートウェイ指定機能によって，Web コンテナにゲートウェイ情報を通知し，welcome ファイルや Form 認証画面に正しくリダイレクトできるようになります。

ゲートウェイ指定機能は次のような場合に使用できます。

- **クライアントと，Web サーバとの間に SSL アクセラレータを配置する場合**
  クライアントから SSL アクセラレータへのアクセスが HTTPS の場合でも，SSL アクセラレータから Web サーバへのアクセスは HTTP となるため，Web コンテナは HTTP によるアクセスであると認識します。このため，welcome ファイルや Form 認証画面へのリダイレクト先 URL のスキームは HTTP となります。
  この場合，ゲートウェイ指定機能を使用して，スキームを常に https と見なすように指定することで，正しくリダイレクトできるようになります。
- **Host ヘッダのないリクエストに対して，リクエストを受けた Web サーバ以外へリダイレクトする必要がある場合**

Host ヘッダのないリクエストをリダイレクトする場合，リダイレクト先 URL のホスト名・ポート番号は，リクエストを受けた Web サーバのホスト名・ポート番号となります。

ゲートウェイ指定機能は，Web サーバの前に負荷分散機を配置している場合などで，クライアントがアクセスする URL のホスト名・ポート番号が，リクエストを受けた Web サーバと異なるときに使用します。これによって，クライアントからアクセスするホスト名・ポート番号が指定されるので，正しくリダイレクトできるようになります。

なお，Web サーバと連携する場合で，かつ一つのリダイレクタに複数の異なる経路でアクセスする場合(複数のゲートウェイから Web コンテナに HTTP リクエストが転送される場合など)，ゲートウェイ指定機能を使用できません。Web サーバと連携する場合，ゲートウェイ指定機能を使用するには，Web コンテナへのアクセス経路は一つになる構成にする必要があります。

5.10.2　実行環境での設定（Smart Composer 機能を使用する場合）

ここは，ゲートウェイ指定機能を使用するための設定について説明します。

クライアントと Web サーバとの間に，SSL アクセラレータや負荷分散機などのゲートウェイを配置している場合，ゲートウェイ指定機能を使用することで，Web コンテナにゲートウェイ情報を通知し，Web アプリケーションのトップページや Form 認証画面に正しくリダイレクトできるようになります。

(1)　設定方法

ゲートウェイ指定機能を使用するための設定方法を次に示します。

1. ゲートウェイのホスト名・ポート番号・リダイレクト先 URL のスキームを，リダイレクタごとに指定します。

2. Web サーバを再起動します。

なお，ゲートウェイのホスト名・ポート番号・リダイレクト先 URL のスキームは，簡易構築定義ファイルに指定します。論理 Web サーバ（web-server）の<configuration>タグ内に，次のパラメタで指定します。

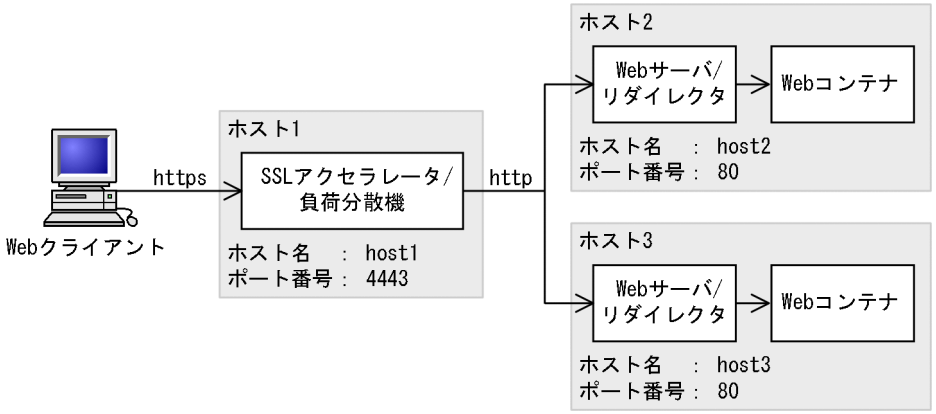
- ホスト名：JkGatewayHost
- ポート番号：JkGatewayPort
- リダイレクト先 URL のスキーム：JkGatewayHttpsScheme

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

(2)　設定例

ゲートウェイ指定機能の設定例を次に示します。

図 5-25　ゲートウェイ指定機能の設定例



この例では，クライアントと Web サーバの間に SSL アクセラレータを配置しています。クライアントから SSL アクセラレータへのアクセスが HTTPS の場合でも，SSL アクセラレータから Web サーバへのアクセスは HTTP となるため，Web コンテナは HTTP によるアクセスであると認識します。このため，Web アプリケーションのトップページや Form 認証画面へのリダイレクト先 URL のスキームは HTTP となります。この場合，ゲートウェイ指定機能を使用して，スキームを常に HTTPS と見なすように指定することで，正しくリダイレクトできるようになります。

簡易構築定義ファイルの例を次に示します。ここでは，リダイレクト先 URL のスキームを常に HTTPS と見なすように JkGatewayHttpsScheme パラメタで「On」を指定します。

簡易構築定義ファイルの例

```
 :
<param>
  <param-name>JkGatewayHost</param-name>
  <param-value>host1</param-value>
</param>
<param>
  <param-name>JkGatewayPort</param-name>
  <param-value>4443</param-value>
</param>
<param>
  <param-name>JkGatewayHttpsScheme</param-name>
  <param-value>On</param-value>
</param>
 :
```

5.10.3　実行環境での設定（Smart Composer 機能を使用しない場合）

ここでは，ゲートウェイ指定機能を使用するための設定について説明します。

クライアントと Web サーバとの間に，SSL アクセラレータや負荷分散機などのゲートウェイを配置している場合，ゲートウェイ指定機能を使用することで，Web コンテナにゲートウェイ情報を通知し，Web アプリケーションのトップページや Form 認証画面に正しくリダイレクトできるようになります。

(1)　設定方法

ゲートウェイ指定機能を使用するための設定方法を次に示します。

1. ゲートウェイのホスト名・ポート番号・リダイレクト先 URL のスキームを，リダイレクタごとに指定します。

2. Web サーバを再起動します。

なお，ゲートウェイのホスト名・ポート番号・リダイレクト先 URL のスキームは，Web サーバに HTTP Server を使用している場合は mod_jk.conf に，Microsoft IIS を使用している場合は isapi_redirect.conf に指定します。指定するキーはそれぞれ次のとおりです。

mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，「13.2.2　mod_jk.conf (HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

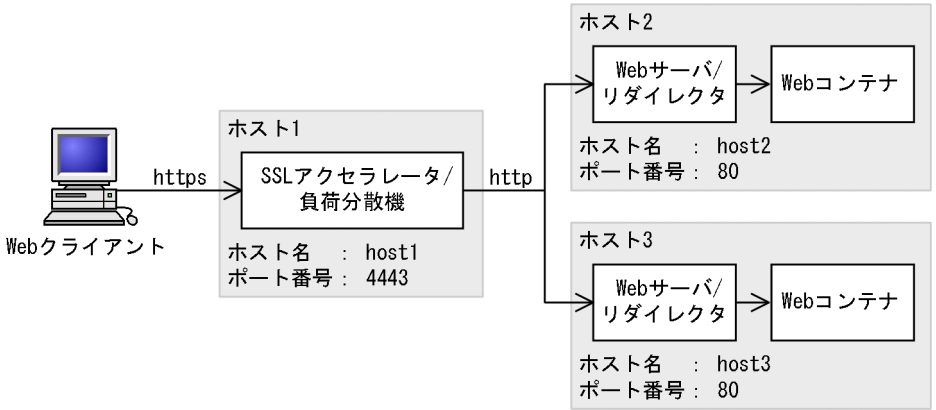
isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）については，「13.2.1 isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル)」を参照してください。

- mod_jk.conf の場合
  ホスト名：JkGatewayHost キー
  ポート番号：JkGatewayPort キー
  リダイレクト先 URL のスキーム：JkGatewayHttpsScheme キー
- isapi_redirect.conf の場合
  ホスト名：gateway_host キー
  ポート番号：gateway_port キー
  リダイレクト先 URL のスキーム：gateway_https_scheme キー

(2)　設定例

ゲートウェイ指定機能の設定例を次に示します。

図 5-26　ゲートウェイ指定機能の設定例



この例では，クライアントと Web サーバの間に SSL アクセラレータを配置しています。クライアントから SSL アクセラレータへのアクセスが HTTPS の場合でも，SSL アクセラレータから Web サーバへのアクセスは HTTP となるため，Web コンテナは HTTP によるアクセスであると認識します。このため，Web アプリケーションのトップページや Form 認証画面へのリダイレクト先 URL のスキームは HTTP となります。この場合，ゲートウェイ指定機能を使用して，スキームを常に HTTPS と見なすように指定することで，正しくリダイレクトできるようになります。

mod_jk.conf，および isapi_redirect.conf の例を次に示します。ここでは，リダイレクト先 URL のスキームを常に HTTPS と見なすように mod_jk.conf の JkGatewayHttpsScheme キーで「On」，isapi_redirect.conf の gateway_https_scheme キーで「true」を指定します。

mod_jk.conf の例（HTTP Server の場合）

```
JkGatewayHost host1
JkGatewayPort 4443
JkGatewayHttpsScheme On
```

isapi_redirect.conf の例（Microsoft IIS の場合）

```
gateway_host=host1
gateway_port=4443
gateway_https_scheme=true
```

5.10.4　Web コンテナにゲートウェイ情報を通知する場合の注意事項

ゲートウェイ指定機能を使用する上での注意事項を次に示します。

リダイレクト先 URL のホスト名，およびポート番号の指定について

通常，ブラウザは Host ヘッダを付けてリクエストを送信するため，リダイレクト先 URL のホスト名やポート番号を指定する必要はありません。

なお，リクエストに Host ヘッダがあるかどうかは，javax.servlet.http.HttpServletRequest クラスの getHeader メソッドに，引数「Host」を指定して呼び出すことで確認できます。

サーブレット API の動作について

ゲートウェイ指定機能の利用によって，一部のサーブレット API の動作が変わります。Web アプリケーションで API を利用するときには注意が必要です。

なお，動作が変わるサーブレット API については，「9.4.2(1)　ゲートウェイ指定機能を使用する場合の注意」を参照してください。

web.xml の<transport-guarantee>タグについて

ゲートウェイ指定機能でスキームを HTTPS と見なすように設定した場合，Web サーバへのリクエストが HTTP であっても HTTPS であると見なされます。このため，web.xml の<transport-guarantee>タグで INTEGRAL や CONFIDENTIAL を指定しても HTTPS の URL へリダイレクトされないので注意してください。

Cookie の Secure 属性について

ゲートウェイ指定機能でスキームを HTTPS と見なすように設定している場合に，Web コンテナが生成したセッション ID を，Cookie によってクライアントに返すとき，その Cookie には Secure 属性が付与されます。

ゲートウェイを通さない Web サーバへの通信

リダイレクタでゲートウェイ指定機能を有効にした場合，その Web サーバに SSL アクセラレータや負荷分散機などのゲートウェイを通さないとき，直接 HTTP 通信はできません。

5.11　Web コンテナ単位での同時実行スレッド数の制御

この節では，Web コンテナ単位での同時実行スレッド数の制御について説明します。

この節の構成を次の表に示します。

表 5-23　この節の構成（Web コンテナ単位での同時実行スレッド数の制御）

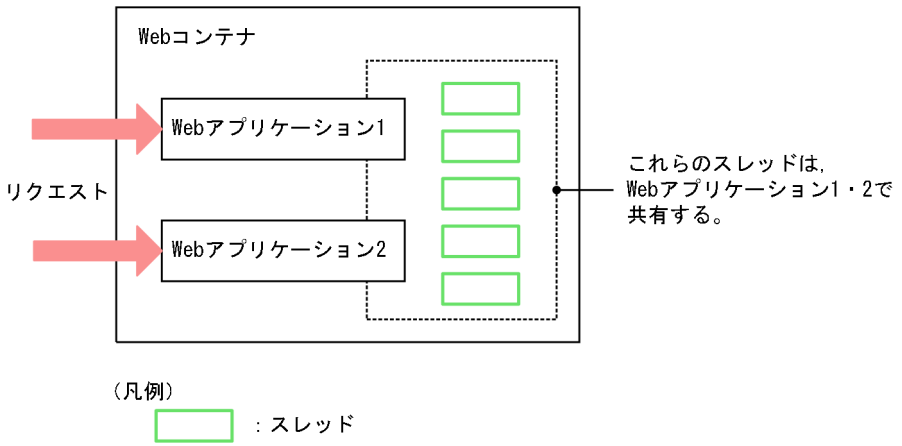
分類	タイトル	参照先
解説	同時実行スレッド数の制御の仕組み（Web コンテナ単位）	5.11.1
設定	実行環境での設定（J2EE サーバの設定）	5.11.2

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

5.11.1　同時実行スレッド数の制御の仕組み（Web コンテナ単位）

Web コンテナ単位での同時実行スレッド数の制御の仕組みについて，次の図で説明します。

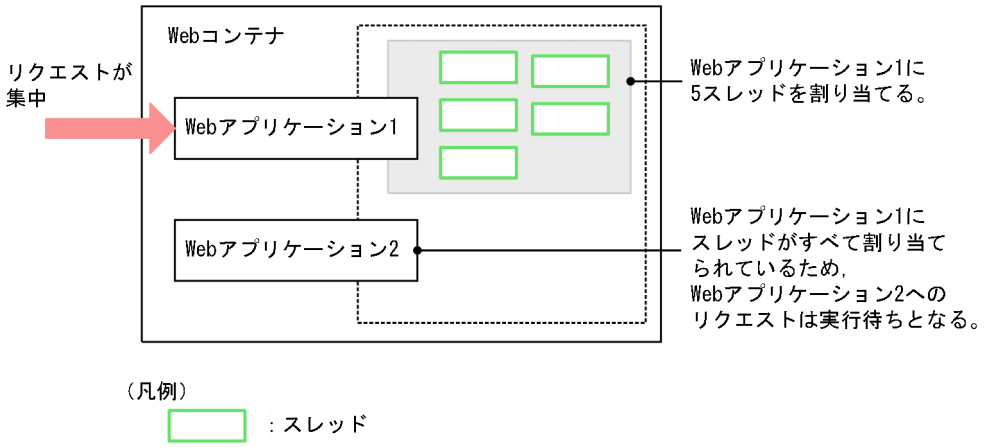
図 5-27　Web コンテナ単位での同時実行スレッド数制御



例えば，Web コンテナに二つの Web アプリケーションがデプロイされていて，同時実行スレッド数に「5」を設定している場合，二つの Web アプリケーションで同時に実行できるスレッド数は 5 となります。

Web コンテナ単位に同時実行スレッド数を設定することで，Web コンテナにデプロイされた複数の Web アプリケーションのうち，一つの Web アプリケーションにアクセスが集中した場合でも，アクセスが集中している Web アプリケーションにスレッドを割り当てることができます。この仕組みについて次の図で説明します。

図 5-28　アクセスが集中したときのスレッドの扱い（Web コンテナ単位の場合）



図のように，Web コンテナに Web アプリケーションが二つデプロイされていて，同時実行スレッド数に「5」が設定されている場合に，Web アプリケーション 1 にリクエストが集中すると，5 スレッドすべてが Web アプリケーション 1 に割り当てられます。

一方，Web アプリケーション 2 に対するリクエストは，Web アプリケーション 1 のリクエスト処理が完了するまで，Web コンテナ単位の実行待ちキューにためられます。なお，Web コンテナ単位の実行待ちキューにためられたリクエストは，リクエスト処理の完了後，順次実行されます。

5.11.2　実行環境での設定（J2EE サーバの設定）

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web コンテナ単位での同時実行スレッド数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，次のどちらかのパラメタを指定します。

- webserver.connector.ajp13.max_threads

  Web コンテナ全体の最大同時実行スレッド数を指定します。このパラメタは Web サーバ連携の場合に指定します。

- webserver.connector.inprocess_http.max_execute_threads

  Web コンテナ全体の最大同時実行スレッド数を指定します。このパラメタはインプロセス HTTP サーバを使用する場合に指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

5.12　リダイレクタとの通信用オブジェクト

Explicit メモリブロック領域の確保や解放は，Web コンテナによって実行されます。Web コンテナとリダイレクタ間の通信に使用する通信用オブジェクトは，通常は常設コネクションとして使い回され，Web サーバが起動している間は保持されます。

Web コンテナとリダイレクタの間で障害が発生した場合などに通信の切断・再接続が発生すると，通信用オブジェクトは破棄・再生成されます。このとき，破棄された通信用オブジェクトが Tenured 領域に残ります。

これを防ぐため，J2EE サーバでは，Web コンテナとリダイレクタ間の通信用オブジェクトを Explicit ヒープに配置します。これによって，不要なオブジェクトが Tenured 領域に残ることを防ぎ，FullGC の発生を抑止します。

通信の確立と切断のタイミングに対応する Explicit メモリブロックの配置の流れについて説明します。

通信が確立されたとき

通信が確立されると，コネクション 1 個に対して 1 個の Explicit メモリブロックが作成されます。作成された Explicit メモリブロックに，リダイレクタとの通信用オブジェクトが配置されます。

通信が切断されたとき

通信が切断されると，Explicit メモリブロック内に配置されたリダイレクタとの通信用オブジェクトごと，1 個の Explicit メモリブロックの解放が予約されます。

通信が切断された直後は，解放が予約された状態です。解放が予約された Explicit メモリブロックは，そのあとで CopyGC または FullGC が実行されたときに実際に解放されます。このとき，解放が予約されたすべての領域が解放されます。

5.13　Explicit ヒープのチューニング

ここでは，Explicit ヒープのチューニングについて説明します。

5.13.1　Explicit ヒープのメモリサイズの見積もり（J2EE サーバが使用するメモリサイズの見積もり）

Explicit ヒープをチューニングする前提として，明示管理ヒープ機能を使用するための設定が必要です。明示管理ヒープ機能は，JavaVM の起動オプションとして-XX:+HitachiUseExplicitMemory が指定されている場合に有効になります。J2EE サーバの場合，明示管理ヒープ機能はデフォルトで使用される設定になっています。また，Tenured 領域のメモリサイズ増加の要因となるオブジェクトが Explicit ヒープに配置されるように設定されています。このため，J2EE サーバが配置するオブジェクトに必要な Explicit ヒープのメモリサイズを必ず見積もってください。

明示管理ヒープ機能は，Explicit ヒープのメモリサイズを適切に見積もった上で使用しないと，効果が出ません。

J2EE サーバでは，Tenured 領域のメモリサイズ増加の要因になる，次のオブジェクトを Explicit ヒープに配置します。

- リダイレクタとの通信用オブジェクト
- HTTP セッションに関するオブジェクト

リダイレクタとの通信用オブジェクトが利用する Explicit ヒープのメモリサイズは，定義ファイルでの設定値から算出できます。見積もり方法については，「5.13.2　リダイレクタとの通信用オブジェクトで利用するメモリサイズ」を参照してください。

HTTP セッションに関するオブジェクトが利用する Explicit ヒープのメモリサイズは，実際にアプリケーションを動作させて情報を取得した上で見積もります。見積もり方法については，マニュアル「アプリケーションサーバ システム設計ガイド」の「7.11.2　HTTP セッションに関するオブジェクトで利用するメモリサイズ」を参照してください。

Explicit ヒープのメモリサイズを見積もった結果，HTTP セッションに関するオブジェクトが利用する Explicit ヒープのメモリサイズが極端に大きい場合は，マニュアル「アプリケーションサーバ システム設計ガイド」の「付録 A　HTTP セッションで利用する Explicit ヒープの効率的な利用」を参考にして，アプリケーションの設計を見直してください。V9 互換モードで HTTP セッションを見積もる際には，リダイレクタとの通信用オブジェクトを Explicit ヒープに配置しないように，J2EE サーバのユーザプロパティ「ejbserver.server.eheap.ajp13.enabled」に false を指定してください。

5.13.2　リダイレクタとの通信用オブジェクトで利用するメモリサイズ

リダイレクタとの通信用オブジェクトで利用する Explicit ヒープのメモリサイズは，次の式で見積もります。

```
リダイレクタとの通信用オブジェクトで利用するExplicitヒープのメモリサイズ
=1コネクションで使用するメモリサイズ※×リダイレクタとのコネクション数
```

注※

1 コネクションで使用するメモリサイズは，明示管理ヒープ機能の自動配置機能の使用の有無によって異なります。明示管理ヒープ機能の自動配置機能の使用の有無による 1 コネクションで使用するメモリサイズを次に示します。

表 5-24　明示管理ヒープ機能の自動配置機能の使用の有無による 1 コネクションで使用するメモリサイズ

項番	明示管理ヒープ機能の自動配置機能使用の有無	1 コネクションで使用するメモリサイズ
1	○	144 キロバイト
2	×	128 キロバイト

（凡例）

○：明示管理ヒープ機能の自動配置機能を使用する。

×：明示管理ヒープ機能の自動配置機能を使用しない。

Web サーバと Web コンテナが 1：1 のシステム構成では，「リダイレクタとのコネクション数」として，Web サーバで設定する最大接続数を使用してください。

最大接続数の設定個所は，Web サーバおよび使用する OS の種類によって異なります。設定個所を次の表に示します。

表 5-25　最大接続数の設定個所

Web サーバ	OS	設定個所
HTTP Server	Windows	httpsd.conf（HTTP Server 定義ファイル）の ThreadsPerChild ディレクティブ
	UNIX	httpsd.conf（HTTP Server 定義ファイル）の MaxClients ディレクティブ
Microsoft IIS	Windows	Web サイトのプロパティの「パフォーマンス」タブで設定する「最大接続数」

5.13.3　稼働情報による見積もり

J2EE サーバのテストを実施する場合，または運用開始後の J2EE サーバによる実際の Explicit ヒープ使用状況は，稼働情報で確認できます。ここでは，稼働情報による確認手順について説明します。

稼働情報の出力内容，出力するための設定，および稼働情報ファイルの出力先については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「3.3　稼働情報ファイルの出力機能」を参照してください。

(1)　稼働情報を使用した見積もりの考え方

稼働情報を使用した見積もりでは，システムに必要な Explicit ヒープ領域のメモリサイズは次のようになります。

1. HTTP セッションで使用する Explicit ヒープ領域のメモリサイズ
2. 1.の領域を除いた，内部（コンテナ）で使用する Explicit ヒープ領域のメモリサイズ
3. アプリケーションおよび JavaVM で使用する Explicit ヒープ領域のメモリサイズ
4. JavaVM が Explicit メモリブロックを管理するために使用する Explicit ヒープ領域のメモリサイズ（Java ヒープの Survivor 領域のサイズ×2）

1.～3.のメモリサイズは，稼働情報で確認できます。4.は，明示管理ヒープの自動解放機能が有効な場合に，「Java ヒープの Survivor 領域のサイズ×2」のメモリサイズを使用します。

1.～3.で示す各 Explicit ヒープ領域を使用するものの例を表で示します。なお，1.～3.は表中の項番 1～3 に対応しています。

表 5-26　Explicit ヒープ領域を使用するものの具体例

項番	Explicit ヒープ領域	Explicit ヒープ領域を使用するものの具体例
1	HTTP セッションで使用する Explicit ヒープ領域	HTTP セッション
2	コンテナで使用する Explicit ヒープ領域	• リダイレクタとの通信用オブジェクト • HTTP セッション管理用オブジェクト
3	アプリケーションおよび JavaVM で使用する Explicit ヒープ領域	• アプリケーション • JavaVM

(2)　見積もりに使用する稼働情報取得時の注意

見積もりに使用する稼働情報は，本番環境，または本番環境と同等の環境で取得してください。

次に示す項目が本番環境と異なる場合は，稼働情報を使って適切なメモリサイズを見積もることはできません。

- 各定義ファイルに設定するプロパティ，およびオプションに指定する値
- サーバに登録されている Web アプリケーションの数
- リダイレクタとのコネクションの数
- 業務アプリケーションが処理するデータのサイズ

- 一定時間内に処理するデータの数

また，見積もりのために稼働情報を取得する場合，Explicit ヒープ領域を使い切った状態にならないよう Explicit ヒープ領域サイズの最大値を設定するオプションを指定してください。

Explicit ヒープ領域の最大サイズが不十分な状態で稼働情報を取得した場合，Explicit ヒープ領域を使い切った状態になるおそれがあります。Explicit ヒープ領域を使い切った状態で取得した稼働情報では，適切な見積もりはできません。Explicit ヒープ領域を使い切った状態かどうかは，稼働情報の EHeapSize.HighWaterMark の値が，Explicit ヒープ領域の最大サイズの値と同じ値になっているかどうかで確認できます。稼働情報の EHeapSize.HighWaterMark の値と Explicit ヒープ領域の最大サイズの値が同じだった場合，Explicit ヒープ領域を使い切っている状態となります。

(3) 見積もり方法

稼働情報を基にした見積もり方法を次に示します。

(a) HTTP セッションで利用する Explicit ヒープ領域のメモリサイズ

HTTP セッションで利用するメモリサイズは，マニュアル「アプリケーションサーバ システム設計ガイド」の「7.11.2 HTTP セッションに関するオブジェクトで利用するメモリサイズ」で示した HTTP セッションで利用する Explicit ヒープのメモリサイズの式で求めます。このとき，式に含まれる「1 セッションで使用するメモリサイズ」を稼働情報で確認できます。

1 セッションで使用するメモリサイズは，稼働情報に出力された「Explicit メモリブロックの最大サイズ」に該当します。「Explicit メモリブロックの最大サイズ」には，稼働情報収集間隔の間に解放された Explicit メモリブロックのうち，最大のものの利用済みサイズが出力されます。そのため，Explicit ヒープを見積もる際は，64 キロバイト単位で切り上げて見積もってください。さらに，明示管理ヒープ機能の自動配置機能を使用する場合は，16 キロバイトを加算して見積もってください。

また，見積もりをする際は，次に示す値を参考にしてください。なお，システムで必要なセッション数は，「Explicit メモリブロックの個数」に該当します。

- HTTP セッションで取得した Explicit メモリブロックの最大サイズ
  （HTTPSessionEMemoryBlockMaxSize.HighWaterMark の値）
- HTTP セッションで取得した Explicit メモリブロックの個数
  （HTTPSessionEMemoryBlockCount.HighWaterMark の値）

(b) コンテナで利用する Explicit ヒープ領域のメモリサイズ

コンテナで使用する Explicit ヒープ領域のメモリサイズは，稼働情報の「コンテナで利用する Explicit ヒープサイズ」が該当します。見積もりに使用する稼働情報は取得した値の中で最大値（ContainerEHeapSize.HighWaterMark の値）を使用してください。

(c) アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズ

アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズは，稼働情報の「アプリケーションで利用する Explicit ヒープサイズ」の値が該当します。見積もりに使用する稼働情報は取得した値の中で最大値（ApplicationEHeapSize.HighWaterMark）を使用してください。

(4) 稼働情報の確認手順

稼働情報の確認手順について説明します。ここでは，(3)による稼働情報の見積もり式を例に説明します。なお，稼働情報ファイルの出力内容については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「3.3 稼働情報ファイルの出力機能」を参照してください。

見積もり式

```
必要なExplicitヒープ領域のメモリサイズ
＝(HTTPSessionEMemoryBlockMaxSize.HighWaterMarkを64キロバイト単位に切り上げた値
×HTTPSessionEMemoryBlockCount.HighWaterMark)
＋ContainerEHeapSize.HighWaterMark
＋ApplicationEHeapSize.HighWaterMark
＋JavaヒープのSurvivor領域のサイズ
×2(明示管理ヒープ自動解放機能が有効な場合だけ)
```

それぞれの値の確認方法について説明します。

(a) HTTP セッションで利用する Explicit ヒープ領域のメモリサイズ

HTTP セッションで利用する Explicit ヒープ領域のメモリサイズは，JavaVM の稼働情報ファイルに出力される HTTPSessionEMemoryBlockMaxSize.HighWaterMark，および HTTPSessionEMemoryBlockCount.HighWaterMark の値を使って確認します。

HTTP セッションで利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例を次の図に示します。

図 5-29　HTTP セッションで利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例

Date(+0900)	HTTPSessionEMemoryBlockMaxSize.StartTime(+0900)	HTTPSessionEMemoryBlockMaxSize.HighWaterMark	HTTPSessionEMemoryBlockMaxSize.LowWaterMark	HTTPSessionEMemoryBlockMaxSize.Current	HTTPSessionEMemoryBlockCount.StartTime(+0900)	HTTPSessionEMemoryBlockCount.HighWaterMark	HTTPSessionEMemoryBlockCount.LowWaterMark	HTTPSessionEMemoryBlockCount.Current
2009/11/18 10:44:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:45:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:46:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:47:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:48:31	43:30.8	16	0	0	43:30.8	10	0	10
2009/11/18 10:49:31	43:30.8	290664	0	0	43:30.8	29	10	24
2009/11/18 10:50:31	43:30.8	403304	0	403304	43:30.8	33	23	25
2009/11/18 10:51:31	43:30.8	408424	0	0	43:30.8	35	24	31
2009/11/18 10:52:31	43:30.8	408424	0	0	43:30.8	38	24	30
2009/11/18 10:53:31	43:30.8	402280	0	402280	43:30.8	35	22	24
2009/11/18 10:54:31	43:30.8	405352	0	0	43:30.8	43	24	40
2009/11/18 10:55:31	43:30.8	404328	0	0	43:30.8	47	29	38
2009/11/18 10:56:31	43:30.8	407400	0	0	43:30.8	49	32	41
2009/11/18 10:57:31	43:30.8	404328	0	0	43:30.8	51	34	35
2009/11/18 10:58:31	43:30.8	402280	0	0	43:30.8	48	33	43
2009/11/18 10:59:31	43:30.8	396136	0	0	43:30.8	48	31	39
2009/11/18 11:00:31	43:30.8	**410472** 1	0	0	43:30.8	46	29	34
2009/11/18 11:01:31	43:30.8	407400	0	0	43:30.8	43	27	33
2009/11/18 11:02:31	43:30.8	408424	0	0	43:30.8	52	31	39
2009/11/18 11:03:31	43:30.8	408424	0	0	43:30.8	55	39	51
2009/11/18 11:04:31	43:30.8	406376	0	0	43:30.8	**57** 2	39	41
2009/11/18 11:05:31	43:30.8	407400	0	0	43:30.8	56	36	47
2009/11/18 11:06:31	43:30.8	409448	0	0	43:30.8	52	34	47
2009/11/18 11:07:31	43:30.8	395112	0	0	43:30.8	51	31	43
2009/11/18 11:08:31	43:30.8	393064	0	0	43:30.8	43	9	9
2009/11/18 11:09:31	43:30.8	0	0	0	43:30.8	13	9	13
2009/11/18 11:10:31	43:30.8	0	0	0	43:30.8	13	13	13

HTTPSessionEMemoryBlockMaxSize.HighWaterMark の最大値は，図中 1.で示している 11:00:31 に取得した 410472 バイト（400.85 キロバイト）です。

この値を 64 キロバイト単位に切り上げると，448 キロバイトとなります。
HTTPSessionEMemoryBlockCount.HighWaterMark の最大値は図中 2.で示している 11:04:31 に取得した 57 です。

これら二つの値を掛け合わせた値が，HTTP セッションで利用する Explicit ヒープ領域のメモリサイズとなります。

(b)　コンテナで利用する Explicit ヒープ領域のメモリサイズ

コンテナで利用する Explicit ヒープ領域のメモリサイズは，JavaVM の稼働情報ファイルに出力される ContainerEHeapSize.HighWaterMark の値を使って確認します。

コンテナで利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例を次の図に示します。

図 5-30　コンテナで利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例

Date(+0900)	ContainerEHeapSize.StartTime(+0900)	ContainerEHeapSize.HighWaterMark	ContainerEHeapSize.LowWaterMark	ContainerEHeapSize.Current
2009/11/18 10:44:31	43:30.8	262144	0	262144
2009/11/18 10:45:31	43:30.8	262144	262144	262144
2009/11/18 10:46:31	43:30.8	262144	262144	262144
2009/11/18 10:47:31	43:30.8	262144	262144	262144
2009/11/18 10:48:31	43:30.8	1572864	262144	1572864
2009/11/18 10:49:31	43:30.8	5505024	1572864	5505024
2009/11/18 10:50:31	43:30.8	**6815744**	5505024	6815744
2009/11/18 10:51:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:52:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:53:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:54:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:55:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:56:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:57:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:58:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:59:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:00:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:01:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:02:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:03:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:04:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:05:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:06:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:07:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:08:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:09:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:10:31	43:30.8	**6815744**	6815744	6815744

1

ContainerEHeapSize.HighWaterMark の最大値は図中 1.で示している 10:50:31 以降に取得している 6815744 バイト（6656 キロバイト）です。

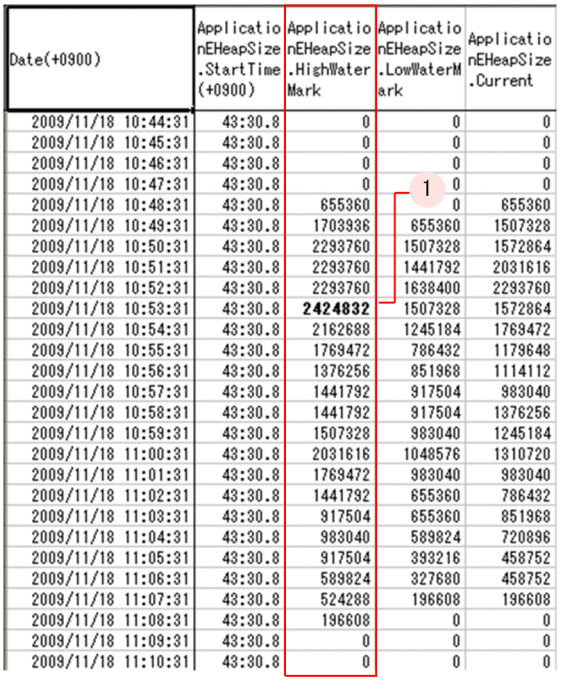
これがコンテナで利用する Explicit ヒープ領域のメモリサイズとなります。

(c)　アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズ

アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズは JavaVM の稼働情報ファイルに出力される ApplicationEHeapSize.HighWaterMark の値を使って確認します。

アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例を次の図に示します。

図 5-31　アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例



Date(+0900)	ApplicationEHeapSize.StartTime(+0900)	ApplicationEHeapSize.HighWaterMark	ApplicationEHeapSize.LowWaterMark	ApplicationEHeapSize.Current
2009/11/18 10:44:31	43:30.8	0	0	0
2009/11/18 10:45:31	43:30.8	0	0	0
2009/11/18 10:46:31	43:30.8	0	0	0
2009/11/18 10:47:31	43:30.8	0	0	0
2009/11/18 10:48:31	43:30.8	655360	0	655360
2009/11/18 10:49:31	43:30.8	1703936	655360	1507328
2009/11/18 10:50:31	43:30.8	2293760	1507328	1572864
2009/11/18 10:51:31	43:30.8	2293760	1441792	2031616
2009/11/18 10:52:31	43:30.8	2293760	1638400	2293760
2009/11/18 10:53:31	43:30.8	**2424832**	1507328	1572864
2009/11/18 10:54:31	43:30.8	2162688	1245184	1769472
2009/11/18 10:55:31	43:30.8	1769472	786432	1179648
2009/11/18 10:56:31	43:30.8	1376256	851968	1114112
2009/11/18 10:57:31	43:30.8	1441792	917504	983040
2009/11/18 10:58:31	43:30.8	1441792	917504	1376256
2009/11/18 10:59:31	43:30.8	1507328	983040	1245184
2009/11/18 11:00:31	43:30.8	2031616	1048576	1310720
2009/11/18 11:01:31	43:30.8	1769472	983040	983040
2009/11/18 11:02:31	43:30.8	1441792	655360	786432
2009/11/18 11:03:31	43:30.8	917504	655360	851968
2009/11/18 11:04:31	43:30.8	983040	589824	720896
2009/11/18 11:05:31	43:30.8	917504	393216	458752
2009/11/18 11:06:31	43:30.8	589824	327680	458752
2009/11/18 11:07:31	43:30.8	524288	196608	196608
2009/11/18 11:08:31	43:30.8	196608	0	0
2009/11/18 11:09:31	43:30.8	0	0	0
2009/11/18 11:10:31	43:30.8	0	0	0

ApplicationEHeapSize.HighWaterMark の最大値は図中 1.で示している 10:53:31 に取得した 2424832 バイト（2368 キロバイト）です。

(d)　必要な Explicit ヒープ領域のメモリサイズ

(a)～(c)で示した稼働情報から求められる，必要な Explicit ヒープ領域のメモリサイズは次のようになります。

```
448(キロバイト)×  57  ＋  6656(キロバイト)  ＋  2368(キロバイト)
＝  34560(キロバイト)≒  34メガバイト
```

明示管理ヒープの自動解放機能が有効な場合，「Java ヒープの Survivor 領域のサイズ×2」を加算した値が，最終的な Explicit ヒープ領域の見積もりサイズとなります。

6 インプロセス HTTP サーバ

この章では，インプロセス HTTP サーバ機能の設定について説明します。

6.1　この章の構成

アプリケーションサーバでは，Web サーバ機能としてインプロセス HTTP サーバを提供しています。**インプロセス HTTP サーバ**とは，J2EE サーバのプロセス内で提供される Web サーバ機能です。Web サーバを経由しないで，HTTP リクエストを J2EE サーバのプロセスが直接受信することによって，Web サーバ連携時よりも優れた処理性能で Web サーバの機能を利用できます。

インプロセス HTTP サーバの機能と参照先を次の表に示します。

表 6-1　インプロセス HTTP サーバの機能と参照先

機能	参照先
インプロセス HTTP サーバの概要	6.2
Web クライアントからの接続数の制御	6.3
リクエスト処理スレッド数の制御	6.4
Web クライアントからの同時接続数の制御によるリクエストの流量制御	6.5
同時実行スレッド数の制御によるリクエストの流量制御	6.6
リダイレクトによるリクエストの振り分け	6.7
Persistent Connection による Web クライアントとの通信制御	6.8
通信タイムアウト（インプロセス HTTP サーバ）	6.9
IP アドレス指定（インプロセス HTTP サーバ）	6.10
アクセスを許可するホストの制限によるアクセス制御	6.11
リクエストデータのサイズの制限によるアクセス制御	6.12
有効な HTTP メソッドの制限によるアクセス制御	6.13
HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ	6.14
エラーページのカスタマイズ（インプロセス HTTP サーバ）	6.15
Web コンテナへのゲートウェイ情報の通知	6.16
ログ・トレースの出力	6.17
URI のデコード機能	6.18
インプロセス HTTP サーバのログ取得の設定	6.19
cjtracesync（インプロセス HTTP サーバ用トレースファイルの同期）	6.20
SOAP アプリケーション運用時の注意事項	6.21

6.2　インプロセス HTTP サーバの概要

この節では，インプロセス HTTP サーバの概要について説明します。

この節の構成を次の表に示します。

表 6–2　この節の構成（インプロセス HTTP サーバの概要）

分類	タイトル	参照先
解説	インプロセス HTTP サーバの使用	6.2.1
	インプロセス HTTP サーバで使用できる機能	6.2.2
設定	実行環境での設定（J2EE サーバの設定）	6.2.3

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.2.1　インプロセス HTTP サーバの使用

インプロセス HTTP サーバは，J2EE サーバのプロセス内で提供される Web サーバ機能です。

Web サーバを経由しないで，HTTP リクエストを J2EE サーバのプロセスが直接受信することによって，Web サーバ連携時よりも優れた処理性能で Web サーバの機能を利用できます。このため，性能を重視したシステムでは，インプロセス HTTP サーバを使用することを推奨します。

ただし，HTTP Server および Microsoft IIS と比べて提供機能に差異があるため，機能差異を確認した上で，インプロセス HTTP サーバを使用するかどうか検討してください。また，インプロセス HTTP サーバと Web サーバ連携機能を同時に使用することはできません。システム設計時に，あらかじめどちらの機能を使用するか選択しておく必要があります。

なお，インプロセス HTTP サーバを使用する場合，次のことが前提となります。

- インプロセス HTTP サーバは不正アクセスが想定される外部ネットワークからのアクセスが可能な DMZ に配置しないで，ファイアウォール内の内部ネットワークに配置する必要があります。インターネットなどの外部ネットワークからアクセスされるシステムでは，DMZ にプロキシサーバを配置して，内部ネットワークのインプロセス HTTP サーバに転送するようにシステムを構築する必要があります。
- インプロセス HTTP サーバでは，HTTP だけがサポートされています。HTTPS はサポートされていません。HTTPS を使用する場合，SSL アクセラレータ，または HTTP Server のリバースプロキシを使用することが前提となります。
- インプロセス HTTP サーバを使用してアクセスできるのは，J2EE サーバ上にデプロイされた Web アプリケーションだけです。なお，静的コンテンツだけをデプロイすることはできませんが，リダイレクトによるリクエストの振り分け，およびエラーページのカスタマイズを実行する場合だけ，Web アプリケーションに含まれない静的コンテンツを指定できます。

また，インプロセス HTTP サーバを使用する場合，次のことに注意してください。

- Web クライアントとインプロセス HTTP サーバとの TCP コネクションの接続中に，cjstopsv コマンドを実行して J2EE サーバを停止した場合，Web クライアントがインプロセス HTTP サーバとの TCP コネクションを切断するか，usrconf.properties（J2EE サーバ用ユーザプロパティファイル）の webserver.connector.inprocess_http.persistent_connection.timeout キーで指定するタイムアウトが発生するまで，J2EE サーバが停止しません。Web クライアントからの TCP コネクション切断や，タイムアウト時間に関係なく J2EE サーバを停止したい場合は，cjstopsv コマンドに「-f」オプションを指定して J2EE サーバを強制停止してください。
- リクエストヘッダに含まれる次のヘッダは非サポートです。※1
  - Cookie2
  - Expect※2
  - Keep-Alive
  - Trailer

  注※1

  インプロセス HTTP サーバではヘッダを読み込みますが，非サポートのヘッダの名前や値に応じた個別の処理はしないで，そのまま J2EE アプリケーションの処理を継続します。

  注※2

  Expect ヘッダの値が""100-continue""のリクエストを受信した場合，HTTP ステータスコード 100 のレスポンスを返さないで，そのまま J2EE アプリケーションの処理を継続します。
- リクエストヘッダに含まれる Cookie ヘッダの値が""（ダブルクォーテーション）で囲まれている場合，""（ダブルクォーテーション）も値の一部として扱います。
- リクエストヘッダに含まれる Transfer-Encoding ヘッダの値の""gzip""は非サポートです。受信したリクエストの Transfer-Encoding ヘッダの該当する値は無視します。

インプロセス HTTP サーバの設定は，J2EE サーバのプロパティをカスタマイズして設定します。J2EE サーバの動作設定のカスタマイズについては，「6.2.3　実行環境での設定（J2EE サーバの設定）」を参照してください。

ポイント

インプロセス HTTP サーバは，デフォルトでは使用しない設定になっています。

6.2.2　インプロセス HTTP サーバで使用できる機能

インプロセス HTTP サーバで使用できる機能と参照先を次の表に示します。

表 6-3　インプロセス HTTP サーバで使用できる機能と参照先

機能名		参照先
Web クライアントからの接続数の制御		6.3
Web クライアントからのリクエスト処理スレッド数の制御		6.4
リクエストの流量制御	Web クライアントからの同時接続数の制御	6.5
	同時実行スレッド数の制御※	6.6
リダイレクトによるリクエストの振り分け		6.7
Web クライアントとの通信制御	Persistent Connection による通信制御	6.8
	インプロセス HTTP サーバで設定できる通信タイムアウト	6.9
	インプロセス HTTP サーバで利用する IP アドレス指定※	6.10
Web クライアントからのアクセス制御	アクセスを許可するホストの制限	6.11
	リクエストデータのサイズ制限	6.12
	有効な HTTP メソッドの制限	6.13
Web クライアントへのレスポンスのカスタマイズ	HTTP レスポンスヘッダのカスタマイズ	6.14
	エラーページのカスタマイズ	6.15
Web コンテナへのゲートウェイ情報の通知※		6.16
ログ・トレースの出力		6.17

注※
インプロセス HTTP サーバを使用しない場合（Web サーバ連携機能を使用する場合）との機能差異はありません。

6.2.3　実行環境での設定（J2EE サーバの設定）

ここでは，インプロセス HTTP サーバの設定手順について説明します。

J2EE サーバのプロセス内で提供される Web サーバ機能を使用して，HTTP リクエストを受信するためには，インプロセス HTTP サーバ機能を使用する構成でシステムを構築する必要があります。

設定手順を次に示します。

1. インプロセス HTTP サーバ機能の使用を有効にします。
   インプロセス HTTP サーバの仕様の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.enabled パラメタで「true」を指定します。デフォルトでは「false」が指定されています。簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

2. Web クライアントからの接続数の制御とリクエスト処理スレッド数の制御を設定します。

サーバを稼働するホストの性能やクライアントからのアクセス状況に合わせてリクエスト処理スレッド数を調節することで，インプロセス HTTP サーバのパフォーマンスを向上できます。設定については，「6.3　Web クライアントからの接続数の制御」および「6.4　リクエスト処理スレッド数の制御」を参照してください。

設定時の留意点を次に示します。

- サーバ起動直後から大量のリクエストを処理する必要がある場合は，サーバ起動時に作成するリクエスト処理スレッド数に大きな値を指定してください。
- 予備スレッドの最大数を大きくすると急なアクセス増加にも迅速に対応できますが，リソースを多く消費するため注意してください。

3. Web クライアントからのアクセス制御の設定をします。

クライアントからの接続や送信されるリクエストに対するセキュリティを強化することで，外部からの不正アクセスやサーバへの攻撃を防ぐことができます。設定については，「6.11　アクセスを許可するホストの制限によるアクセス制御」，「6.12　リクエストデータのサイズの制限によるアクセス制御」，および「6.13　有効な HTTP メソッドの制限によるアクセス制御」を参照してください。

4. 必要に応じて，インプロセス HTTP サーバで使用できる各機能についても設定してください。

インプロセス HTTP サーバで使用できる機能については，「6.2.2　インプロセス HTTP サーバで使用できる機能」を参照してください。

6.3　Web クライアントからの接続数の制御

Web クライアントからの接続数とリクエスト処理スレッド数を制御して，リクエスト処理スレッド数を最適化することによって，J2EE サーバの負荷を一定に抑え，安定した高いスループットを維持できます。リクエスト処理スレッド数の制御については，「6.4　リクエスト処理スレッド数の制御」を参照してください。

この節では，Web クライアントからの接続数の制御について説明します。

この節の構成を次の表に示します。

表 6-4　この節の構成（Web クライアントからの接続数の制御）

分類	タイトル	参照先
解説	Web クライアントからの接続数の制御の概要	6.3.1
設定	実行環境での設定（J2EE サーバの設定）	6.3.2

注　「実装」，「運用」および「注意事項」について，この機能固有の説明はありません。

インプロセス HTTP サーバでは，一度に接続できる Web クライアントの数を設定することで，インプロセス HTTP サーバで作成するリクエスト処理スレッド数を制御できます。また，処理を実行していないリクエスト処理スレッドを予備スレッドとして一定数プールしておくことで，リクエスト処理スレッドの追加・削除に掛かる処理を最小限に抑えられます。

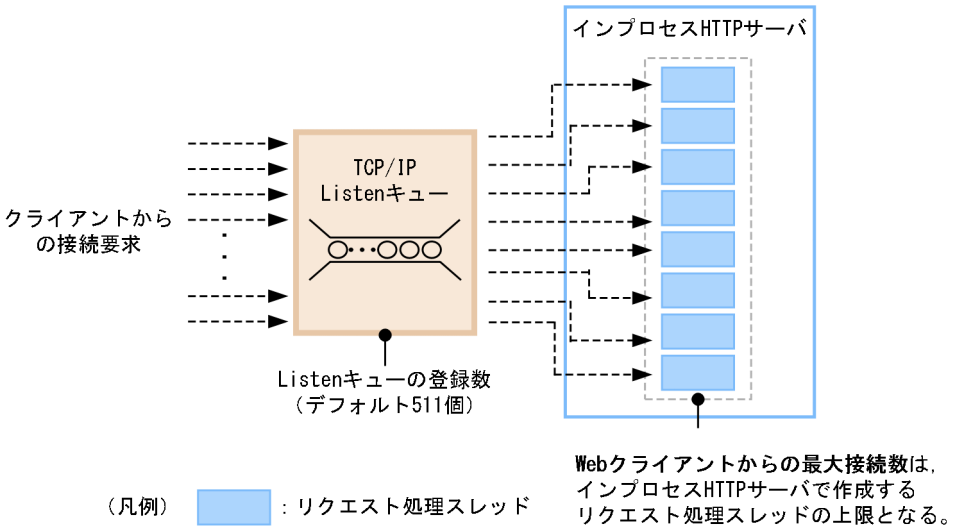
6.3.1　Web クライアントからの接続数の制御の概要

インプロセス HTTP サーバでは，一度に接続する Web クライアントやプロキシサーバの数の最大値を設定して，リクエスト処理スレッド数を制御します。インプロセス HTTP サーバでは Web クライアントからの接続数分のリクエスト処理スレッドを作成するため，Web クライアントからの接続数の最大値は，インプロセス HTTP サーバで作成するリクエスト処理スレッド数の上限となります。

なお，クライアントからの接続要求は，TCP/IP の Listen キューに登録されて，リクエスト処理スレッドに渡されます。接続数の上限を超えたクライアントからの接続要求は，Listen キューに蓄えられます。Listen キューに蓄えられたクライアントからの接続要求が指定した最大値を超えた場合，クライアントはサーバへの接続に失敗します。

Web クライアントからの接続数の制御の概要を次の図に示します。

図 6-1　Web クライアントからの接続数の制御の概要



6.3.2　実行環境での設定（J2EE サーバの設定）

Web クライアントからの接続数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Web クライアントからの接続数の制御の定義について次の表に示します。

表 6-5　簡易構築定義ファイルでの Web クライアントからの接続数の制御の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.max_connections	Web クライアントやプロキシサーバとの接続数の最大値を指定します。インプロセス HTTP サーバでは，Web クライアントからの接続数分のリクエスト処理スレッドを作成するため，ここで指定した値がリクエスト処理スレッド数の上限になります。
webserver.connector.inprocess_http.backlog	Web クライアントからの接続数の上限を超えた HTTP リクエストは，Listen キューに蓄えられます。ここでは，Listen キューの登録数の最大値を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

6.4　リクエスト処理スレッド数の制御

Web クライアントからの接続数とリクエスト処理スレッド数を制御して，リクエスト処理スレッド数を最適化することによって，J2EE サーバの負荷を一定に抑え，安定した高いスループットを維持できます。Web クライアントからの接続数の制御については，「6.3　Web クライアントからの接続数の制御」を参照してください。

この節では，リクエスト処理スレッド数の制御について説明します。この節の構成を次の表に示します。

表 6-6　この節の構成（リクエスト処理スレッド数の制御）

分類	タイトル	参照先
解説	リクエスト処理スレッド数の制御の概要	6.4.1
設定	実行環境での設定（J2EE サーバの設定）	6.4.2

注　「実装」，「運用」および「注意事項」について，この機能固有の説明はありません。

6.4.1　リクエスト処理スレッド数の制御の概要

インプロセス HTTP サーバでは，起動時にリクエスト処理スレッドを作成したあと，リクエスト処理スレッドの状態，およびスレッド数を定期的に監視します。インプロセス HTTP サーバにリクエストが集中している場合は，リクエスト処理スレッドを追加して十分な予備スレッドをプールしておき，リクエストが少ない場合は余分にプールしている予備スレッドを削除します。

リクエスト処理スレッド数の制御は次のように実行します。

1. J2EE サーバ起動時に，指定した数分のリクエスト処理スレッドを作成します。
2. J2EE サーバの稼働中はリクエスト処理スレッド数を監視します。
3. 監視のタイミングで，予備スレッド数が指定した最小値より少ない場合は，リクエスト処理スレッドを追加して予備スレッドとしてプールします。また，予備スレッド数が指定した最大値より多い場合は，余分な予備スレッドを削除します。

   なお，J2EE サーバ起動時に作成したスレッド数を維持することもできます。起動時に作成したスレッド数を維持する場合，リクエスト処理スレッドと予備スレッドの総数が起動時に作成したスレッド数以下の場合，予備スレッド数が最大値を超えていても予備スレッドは削除されません。例えば，起動時に作成したスレッド数が 8 で，予備スレッド数の最大値が 5 の場合，リクエスト処理スレッドが 2 で，予備スレッド数が 6 のときは，予備スレッドは削除されません。

次に，リクエスト処理スレッド数の遷移について，例を使用して説明します。

- **遷移例 1**

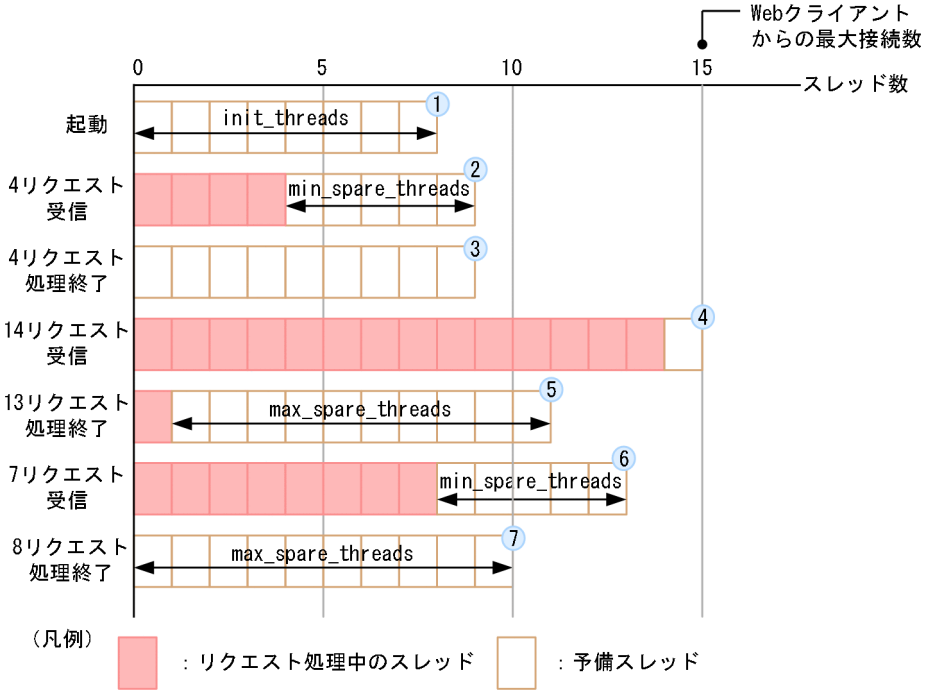
  次の内容が設定されていることとします。

  - Web クライアントからの最大接続数：15

- Listen キューの登録数：100
- J2EE サーバ起動時に作成するリクエスト処理スレッド数：8
- 予備スレッドの最小値：5
- 予備スレッドの最大値：10
- J2EE サーバ起動時に作成したスレッド数の維持：無効

リクエスト処理スレッド数の遷移例を次の図に示します。

図 6-2　リクエスト処理スレッド数の遷移例



図中の項番 1～7 について説明します。

1. J2EE サーバ起動時に，指定した数分（8 スレッド）のリクエスト処理スレッドを作成します。
2. 4 リクエストを受信した時点で，予備スレッドが 4 スレッドとなり，最小値より少ないため 1 スレッド追加します。
3. 4 リクエストの処理が終了した時点で，予備スレッドが 9 スレッドとなり，最大値より少なく，最小値より多いため現状を維持します。
4. 14 リクエストを受信した時点で，予備スレッド数が最小値より少なくなっていますが，Web クライアントからの最大接続数に達しているため，予備スレッドを 1 スレッドだけ追加します。
5. 13 リクエストの処理が終了した時点で，予備スレッド数が最大値を超えているため，4 スレッド削除します。
6. 7 リクエストを受信した時点で，予備スレッド数が最小値より少ないため，2 スレッド追加します。
7. 8 リクエストの処理が終了した時点で，予備スレッド数が最大値を超えているため，3 スレッド削除します。

- **遷移例 2**

予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にすることによって，一度作成したリクエスト処理スレッドを削除しないで使い続けることができます。

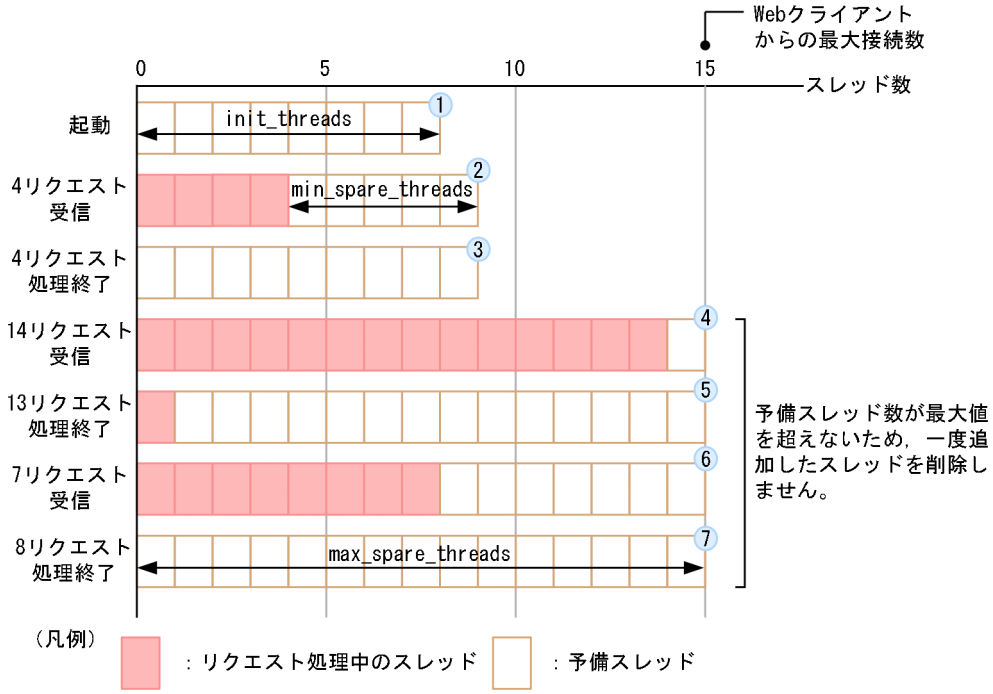
予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にした場合のリクエスト処理スレッド数の遷移例について説明します。

なお，ここでは，次の内容が設定されていることとします。

- Web クライアントからの接続数の最大値：15
- Listen キューの登録数の最大値：100
- J2EE サーバ起動時に作成するリクエスト処理スレッド数：8
- 予備スレッドの最小値：5
- 予備スレッドの最大値：15
- J2EE サーバ起動時に作成したスレッド数の維持：無効

予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にした場合のリクエスト処理スレッド数の遷移例を次の図に示します。

図 6-3　予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にした場合のリクエスト処理スレッド数の遷移例



図中の項番 1～7 について説明します。

1. J2EE サーバ起動時に，指定した数分（8 スレッド）のリクエスト処理スレッドを作成します。
2. 4 リクエストを受信した時点で，予備スレッドが 4 スレッドとなり，最小値より少ないため 1 スレッド追加します。
3. 4 リクエストの処理が終了した時点で，予備スレッドが 9 スレッドとなり，最大値より少なく，最小値より多いため現状を維持します。

4. 14 リクエストを受信した時点で，予備スレッド数が最小値より少なくなっていますが，リクエスト処理スレッドの総数が Web クライアントからの接続数の最大値を超えないように，予備スレッドを 1 スレッドだけ追加します。
5. 13 リクエストの処理が終了した時点で，予備スレッドが 14 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。
6. 7 リクエストを受信した時点で，予備スレッドが 7 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。
7. 8 リクエストの処理が終了した時点で，予備スレッド数が 15 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。

- **設定例 3**

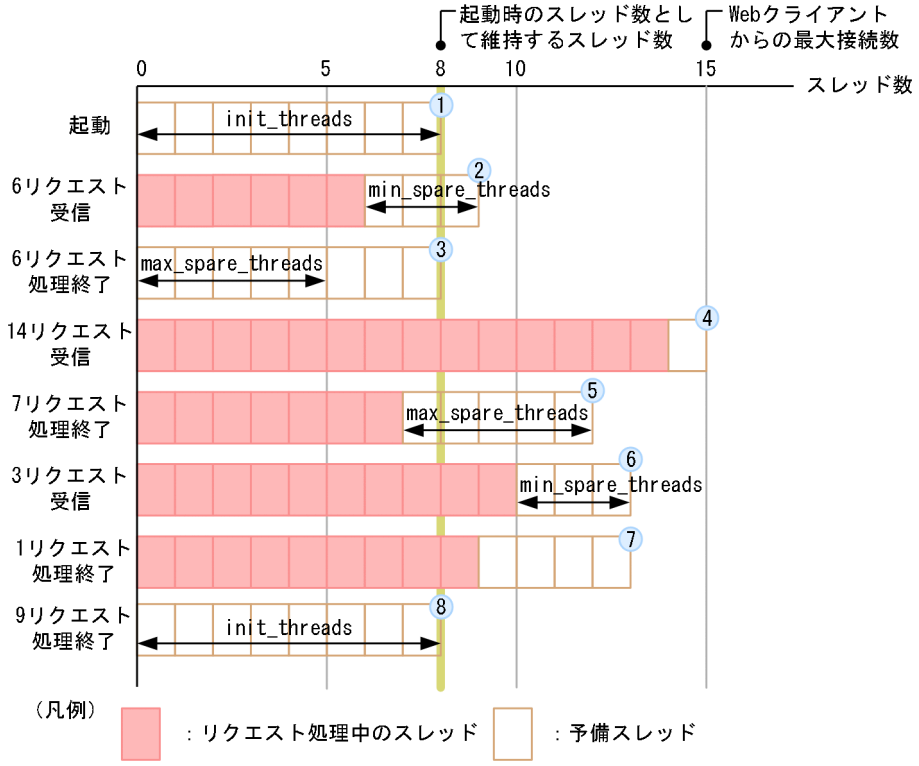
  サーバ起動時に作成されたスレッド数を維持する場合のリクエスト処理スレッド数の遷移例について説明します。

  なお，ここでは，次の内容が設定されていることとします。

  - Web クライアントからの接続数の最大値：15
  - Listen キューの登録数の最大値：100
  - J2EE サーバ起動時に作成するリクエスト処理スレッド数：8
  - 予備スレッドの最小値：3
  - 予備スレッドの最大値：5
  - J2EE サーバ起動時に作成したスレッド数の維持：有効

  サーバ起動時に作成されたスレッド数を維持する場合のリクエスト処理スレッド数の遷移例を次の図に示します。

図 6-4　J2EE サーバ起動時に作成したスレッドを維持する場合のリクエスト処理スレッド数の遷移例



図中の項番 1～8 について説明します。

1. J2EE サーバ起動時に，指定した数分（8 スレッド）のリクエスト処理スレッドを作成します。
2. 6 リクエストを受信した時点で，予備スレッドが 2 スレッドとなり，最小値より少ないため 1 スレッド追加します。
3. 6 リクエストの処理が終了した時点で，予備スレッドが 9 スレッドとなり，最大値を超えていますが，サーバ起動時に作成したスレッド数を維持するため，1 スレッドだけ削除します。
4. 14 リクエストを受信した時点で，予備スレッド数が最小値より少なくなっていますが，リクエスト処理スレッドの総数が Web クライアントからの接続数の最大値を超えないように，予備スレッドを 1 スレッドだけ追加します。
5. 7 リクエストの処理が終了した時点で，予備スレッドが 8 スレッドとなり，最大値を超えたため，3 スレッド削除します。
6. 3 リクエストを受信した時点で，予備スレッドが 2 スレッドとなり，最小値より少ないため，1 スレッド追加します。
7. 1 リクエストの処理が終了した時点で，予備スレッド数が 4 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。
8. 9 リクエストの処理が終了した時点で，予備スレッド数が 13 スレッドとなり，最大値より多くなっていますが，J2EE サーバ起動時のスレッド数を維持するため，5 スレッドだけ削除します。

6.4.2　実行環境での設定（J2EE サーバの設定）

リクエスト処理スレッド数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリクエスト処理スレッド数の制御の定義について次の表に示します。

表 6-7　簡易構築定義ファイルでのリクエスト処理スレッド数の制御の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.init_threads	J2EE サーバ起動時に作成するリクエスト処理スレッド数を指定します。
webserver.connector.inprocess_http.min_spare_threads	予備スレッドの最小値を指定します。予備スレッド数が指定した最小値より少ない場合は，リクエスト処理スレッドが追加されて予備スレッドとしてプールされます。
webserver.connector.inprocess_http.max_spare_threads	予備スレッドの最大値を指定します。予備スレッド数が指定した最大値より多い場合は，余分な予備スレッドが削除されます。予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にすることによって，一度作成したリクエスト処理スレッドを削除しないで使い続けることができます。
webserver.connector.inprocess_http.keep_start_threads	J2EE サーバ起動時に作成したリクエスト処理スレッドを維持するかどうかを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

6.5　Web クライアントからの同時接続数の制御によるリクエストの流量制御

この節では，Web クライアントからの同時接続数の制御によるリクエストの流量制御について説明します。

この節の構成を次の表に示します。

表 6-8　この節の構成（Web クライアントからの同時接続数の制御によるリクエストの流量制御）

分類	タイトル	参照先
解説	Web クライアントからの同時接続数の制御	6.5.1
設定	実行環境での設定（J2EE サーバの設定）	6.5.2

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.5.1　Web クライアントからの同時接続数の制御

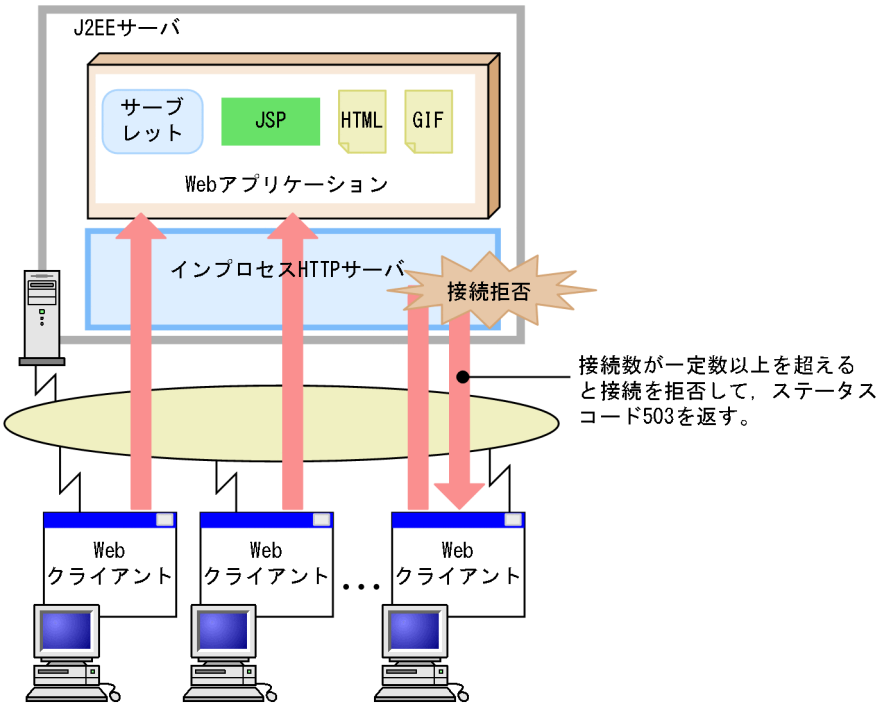
インプロセス HTTP サーバでは，Web クライアントからの最大接続数の設定とあわせて，接続を拒否するリクエスト数を設定することによって，Web クライアントからの同時接続数を制御します。

Web クライアントからの接続数の増加や，J2EE アプリケーションなどの影響で J2EE サーバの負荷が高くなった場合に，Web クライアントからのリクエストの受け付けを拒否して，即座にエラーを返すことで，Web クライアントは即座にレスポンスを受信できます。これによって，J2EE サーバの負荷を一定に抑え，リクエストに対するレスポンスタイムを維持できます。

Web クライアントからの最大接続数から接続を拒否するリクエスト数を引いた値が，Web クライアントからの接続を承諾する数となります。

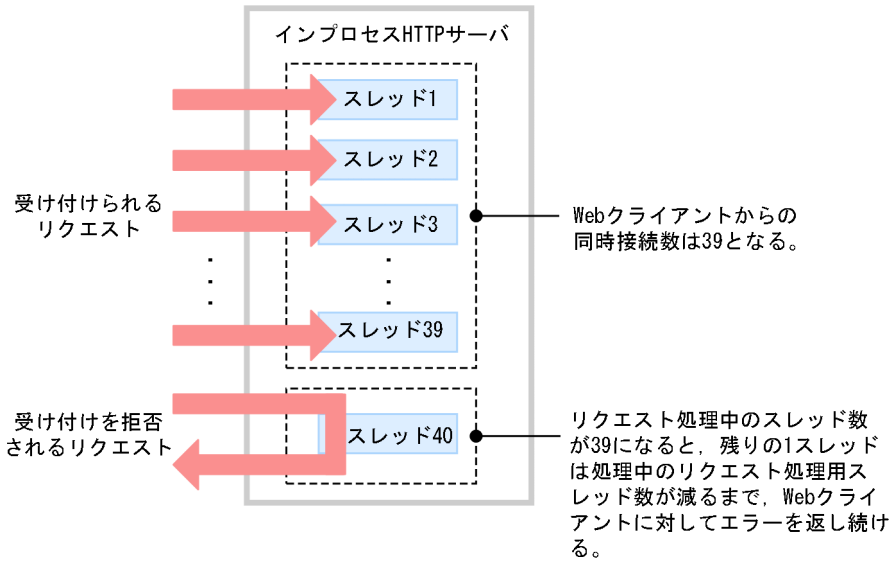
Web クライアントからの同時接続数の制御の概要を，次の図に示します。

図 6-5　Web クライアントからの同時接続数の制御の概要



例えば，Web クライアントからの最大接続数を 40 として，接続を拒否するリクエスト数を 1 とする場合，インプロセス HTTP サーバに接続し同時にリクエストを処理できる Web クライアントの数は 39 となります。リクエスト処理中のスレッド数が 39 になると，残りの 1 スレッドは処理中のリクエスト処理スレッド数が減るまで受信したリクエストに対してエラーを返し続けます。Web クライアントからの同時接続数の制御の例を次の図に示します。

図 6-6　Web クライアントからの同時接続数の制御の例



Web クライアントからの同時接続数の制御によって，接続を拒否したリクエストに対しては，ステータスコード 503 のエラーを Web クライアントに返します。このときクライアントに返すエラーページをカスタマイズすると，レスポンスメッセージのカスタマイズや，ほかのサーバへのリダイレクトができます。

Web クライアントへのレスポンスのカスタマイズ（インプロセス HTTP サーバの場合）の設定については，「6.14　HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ」，および「6.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）」を参照してください。

注意事項

フレームを使用したページや，画像を挿入したページを表示する際，Web クライアントから複数のリクエストを受信する場合があります。その場合，Web クライアントからの同時接続数の制御によって表示されるページが部分的にエラーになることがあります。

6.5.2　実行環境での設定（J2EE サーバの設定）

Web クライアントからの同時接続数の制御をする場合，J2EE サーバの設定が必要です。

Web クライアントからの同時接続数の制御の設定方法および設定例について説明します。

(1)　設定方法

Web クライアントからの同時接続数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.rejection_threads

接続を拒否するリクエスト数を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

注意事項

フレームを使用したページや，画像を挿入したページを表示する際，Web クライアントから複数のリクエストを受信する場合があります。その場合，Web クライアントからの同時接続数の制御によって表示されるページが部分的にエラーになることがあります。

(2)　設定例

Web クライアントからの同時接続数の制御の設定例について説明します。

次に，リクエスト処理スレッド数の上限が 40，接続を拒否するリクエスト処理スレッド数が 1 の場合の設定例を示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.max_connections</param-name>
  <param-value>40</param-value>
```

```
</param>
<param>
  <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
  <param-value>1</param-value>
</param>
 :
```

この設定例では，接続後に同時にリクエストを処理できる Web クライアントの数は 39 となります。リクエスト処理中のスレッド数が 39 に達すると，残りの 1 スレッドは Web クライアントに対してエラーを返し続けます。

Web クライアントからの同時接続数の制御によって，接続を拒否したリクエストに対しては，ステータスコード 503（Service Unavailable）のエラーを Web クライアントに返します。このときクライアントに返すエラーページをカスタマイズすると，レスポンスメッセージのカスタマイズや，ほかのサーバへのリダイレクトができます。それぞれの場合の設定例を次に示します。なお，エラーページのカスタマイズについては，「6.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）」を参照してください。

- **レスポンスメッセージをカスタマイズする場合**

  Web クライアントからの接続を拒否した場合に，レスポンスボディとして特定のファイルを Web クライアントに返すための設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
  <param-value>3</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
  <param-value>REJECTION_1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.status</param-name>
  <param-value>503</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.file</param-name>
  <param-value>C:/data/busy.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.file.content_type=text/html; charset</param-name>
  <param-value>ISO-8859-1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.request_url</param-name>
  <param-value>/*</param-value>
</param>
 :
```

この設定例では，エラーステータスコードに「503」，対応するエラーページのファイルに「C:/data/busy.html」，レスポンスの Content-Type ヘッダに「text/html; charset=ISO-8859-1（Media-Type は text/html で，ISO-8859-1 文字セットを使用）」，URL パターンに「/*」を指定しています。このため，エラーステータスコード「503」が発生した場合には，リクエストの URI に関係なく，「C:/data/busy.html」のファイルの内容をレスポンスとして返します。

- ほかのサーバへリダイレクトする場合

接続を拒否したリクエストをほかのサーバへリダイレクトする場合の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
  <param-value>3</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
  <param-value>REJECTION_1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.status</param-name>
  <param-value>503</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.redirect_url</param-name>
  <param-value>http://host1/busy.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.request_url</param-name>
  <param-value>/*</param-value>
</param>
<param>
 :
```

この設定例では，エラーステータスコードに「503」，対応するエラーページのファイルに「http://host1/busy.html」，URL パターンに「/*」を指定しています。このため，エラーステータスコード「503」が発生した場合には，リクエストの URI に関係なく，すべてのリクエストが「http://host1/busy.html」の URL にリダイレクトされます。

6.6　同時実行スレッド数の制御によるリクエストの流量制御

この節では，同時実行スレッド数の制御によるリクエストの流量制御について説明します。

この節の構成を次の表に示します。

表 6–9　この節の構成（同時実行スレッド数の制御によるリクエストの流量制御）

分類	タイトル	参照先
解説	同時実行スレッド数の制御によるリクエストの流量制御の概要	6.6.1
設定	実行環境での設定（J2EE サーバの設定）	6.6.2

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.6.1　同時実行スレッド数の制御によるリクエストの流量制御の概要

Web コンテナでは，マルチスレッドでサーブレットのリクエストを処理します。このとき，同時に実行できるスレッド数に上限を設定することで，スラッシングなどによるパフォーマンスの低下を防止できます。また，適切なスレッド数を設定することで，アクセス状況に従ったパフォーマンスのチューニングができます。

6.6.2　実行環境での設定（J2EE サーバの設定）

Web コンテナでの同時実行スレッド数の制御をする場合，J2EE サーバの設定が必要です。

ここでは，Web コンテナでの同時実行スレッド数の制御の設定について説明します。

同時に実行するスレッド数を制御するには，Web コンテナ単位で制御する方法，Web アプリケーション単位で制御する方法，および URL グループ単位で制御する方法の 3 種類あります。

(1)　Web コンテナ単位での制御

Web コンテナ単位で同時実行スレッド数を制御する場合，Web コンテナ全体で同時に実行できるスレッドの最大数を設定します。ここで設定したスレッド数は，Web コンテナにデプロイされているすべての Web アプリケーションで共用するスレッド数となります。

Web コンテナ単位での同時実行スレッド数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.max_execute_threads

Web コンテナ全体で同時に実行できるスレッドの最大数を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

(2)　Web アプリケーション単位での制御

Web アプリケーション単位で同時実行スレッド数を制御する場合，Web コンテナでのスレッド数制御も同時に設定する必要があります。

Web アプリケーション単位での同時実行スレッド数の制御は，簡易構築定義ファイルおよびサーバ管理コマンドで設定します。設定方法は，Web サーバ連携機能を使用する場合と同じです。Web サーバ連携機能を使用する場合の，Web アプリケーション単位での同時実行スレッド数の設定方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.15　Web アプリケーション単位での同時実行スレッド数の制御」を参照してください。

(3)　URL グループ単位での制御

URL グループ単位で同時実行スレッド数を制御する場合，Web アプリケーション単位での同時実行スレッド数制御，および Web コンテナでのスレッド数制御も同時に設定する必要があります。

URL グループ単位での同時実行スレッド数の制御は，サーバ管理コマンドで設定します。設定方法は，Web サーバ連携機能を使用する場合と同じです。Web サーバ連携機能を使用する場合の，URL グループ単位での同時実行スレッド数の設定方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.16　URL グループ単位での同時実行スレッド数の制御」を参照してください。

6.7　リダイレクトによるリクエストの振り分け

この節では，リダイレクトによるリクエストの振り分けについて説明します。

インプロセス HTTP サーバでは，HTTP リクエストに含まれる URL パターンによってリクエストを振り分けます。また，振り分けたリクエストに対するレスポンスをカスタマイズしてクライアントに返すこともできます。

URL パターンによるリクエストの振り分けと，レスポンスのカスタマイズの処理について説明します。また，リダイレクトによるリクエストの振り分けの設定の概要についても説明します。

この節の構成を次の表に示します。

表 6–10　この節の構成（リダイレクトによるリクエストの振り分け）

分類	タイトル	参照先
解説	URL パターンによるリクエストの振り分け	6.7.1
	レスポンスのカスタマイズ	6.7.2
設定	実行環境での設定（J2EE サーバの設定）	6.7.3
注意事項	リダイレクトによるリクエストの振り分けに関する注意事項	6.7.4

注 「実装」および「運用」について，この機能固有の説明はありません。

6.7.1　URL パターンによるリクエストの振り分け

インプロセス HTTP サーバでは，インプロセス HTTP サーバあての HTTP リクエストのうち，特定の URL へのリクエストを，指定した Web コンテナに振り分けて処理させることができます。これによって，システム構成の変更などの理由で Web アプリケーションを別な J2EE サーバに移動する場合に，変更前の URL へのリクエストを，変更後の URL に転送できます。

また，インプロセス HTTP サーバでのリダイレクトによる振り分けでは，特定の Web アプリケーション，および Web アプリケーション内の特定のサーブレット/JSP に対するリクエストを，一時的にほかの Web サーバにリダイレクトする場合にも使用できます。インプロセス HTTP サーバでは，リクエストされたサーブレット/JSP などのリソースが実際にあるかどうかに関係なく，リダイレクトを実行します。リダイレクトは，サーブレット/JSP よりも優先されます。そのため，サーブレット/JSP へのリクエストがリダイレクトされる URL と一致した場合，サーブレット/JSP は実行されません。

6.7.2　レスポンスのカスタマイズ

特定の URL へのリクエストに対して，特定のファイルをレスポンスとして返すようにカスタマイズすることもできます。リダイレクトする URL へのリクエストに対するレスポンスのステータスコードが 300～

307 の場合，レスポンスボディを自動生成してクライアントにレスポンスを返します。また，指定したファイルをレスポンスボディとして使用することもできます。ファイルを指定する場合，レスポンスの Content-Type ヘッダもあわせて指定します。

レスポンスボディが自動生成されるステータスコードの詳細，およびリダイレクトによるリクエストの振り分けの設定（インプロセス HTTP サーバの場合）については，「6.7.3　実行環境での設定（J2EE サーバの設定）」を参照してください。

6.7.3　実行環境での設定（J2EE サーバの設定）

ここでは，リダイレクトによるリクエストの振り分けの設定について説明します。

(1)　概要

インプロセス HTTP サーバでは，HTTP リクエストに含まれる URL パターンによってリクエストを振り分けることができます。また，振り分けたリクエストに対するレスポンスをカスタマイズして特定のファイルをクライアントに返すこともできます。リダイレクトする URL へのリクエストに対するレスポンスのステータスコードが 300 番台の場合，レスポンスボディを自動生成してクライアントにレスポンスを返します。また，指定したファイルをレスポンスボディとして使用することもできます。ファイルを指定する場合，レスポンスの Content-Type ヘッダもあわせて指定します。

自動生成されるレスポンスボディを次に示します。

```
<HTML><HEAD>
<TITLE>ステータスコードおよび説明句</TITLE>
</HEAD><BODY>
<H1>ステータスコードおよび説明句</H1>
</BODY></HTML>
```

レスポンスボディが自動生成されるステータスコードおよび説明句を次に示します。

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 305 Use Proxy
- 307 Temporary Redirect

リクエスト処理時にレスポンスボディとして使用するファイルの読み込みに失敗した場合，ステータスコードに 300 番台が指定されていれば，レスポンスボディを自動生成してクライアントに返します。ステータスコードに 200 が指定されていれば，ステータス 500 エラーをクライアントに返します。

(2) 設定方法

リダイレクトによるリクエストの振り分けの定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリダイレクトによるリクエストの振り分けの定義について次の表に示します。

表 6-11 簡易構築定義ファイルでのリダイレクトによるリクエストの振り分けの定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.redirect.list	リダイレクト定義名を指定します。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.request_url	リダイレクトするリクエストの URL を「/」（スラッシュ）で始まる絶対パスで指定します。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.redirect_url	リクエストのリダイレクト先の URL を指定します。なお，ステータスコードに 200 を指定した場合は，URL を指定できません。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.status	リダイレクトを実行する場合のレスポンスのステータスコードを指定します。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.file	特定のファイルをレスポンスとしてクライアントに返す場合にレスポンスボディとして使用するファイルを指定します。なお，ステータスコードに 200 を指定した場合は，使用するファイルを指定する必要があります。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.file.content_type	特定のファイルをレスポンスとしてクライアントに返す場合にレスポンスボディとして使用するファイルの Content-Type ヘッダを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

注意事項

- HTTP レスポンスが HTML ファイルの場合，その HTML からほかのファイル（画像ファイルなど）を参照していると，正しくブラウザに表示されないことがあります。
- リダイレクト URL の指定値がリクエスト URL の指定値と一致する場合，クライアントによってはリダイレクトを実行し続けることがあります。
- URL の書き換えによるセッション管理をしている場合，リクエスト URL と同じ Web アプリケーション内へのリダイレクトを実行してもセッションを引き継ぐことはできません。

(3) 設定例

リダイレクトによるリクエストの振り分けの設定例について説明します。

- 設定例 1

  リダイレクトによるリクエストの振り分けの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.redirect.list</param-name>
  <param-value>REDIRECT_1,REDIRECT_2</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.request_url</param-n
ame>
  <param-value>/index.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.redirect_url</param-
name>
  <param-value>http://host1/new_dir/index.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.status</param-name>
  <param-value>302</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.request_url</param-n
ame>
  <param-value>/old_dir/*</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.redirect_url</param-
name>
  <param-value>http://host1/new_dir/</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.status</param-name>
  <param-value>301</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.file</param-name>
  <param-value>C:/data/301.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.file.content_type</p
aram-name>
  <param-value>text/html; charset=ISO-8859-1</param-value>
</param>
 :
```

この設定例では，リダイレクト定義名として，「REDIRECT_1」と「REDIRECT_2」を使用しています。「REDIRECT_1」では，「/index.html」へのリクエストを「http://host1/new_dir/index.html」にステータスコード「302」でリダイレクトします。「REDIRECT_2」では，「/old_dir/」以下へのリクエストを「http://host1/new_dir/」以下にステータス「301」でリダイレクトします。また，レスポンスボディとして「C:/data/301.html」を使用し，Content-Type ヘッダとして「text/html; charset=ISO-8859-1」を使用します。

- **設定例 2**

リクエスト URL としてワイルドカードを使用し，リダイレクト URL に「/」で終わる値を指定した場合，レスポンスの Location ヘッダに設定される値は，「リダイレクト URL に指定した値」＋「実際のリクエスト URL のワイルドカード以降のパス」になります。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.redirect.list</param-name>
  <param-value>REDIRECT_3</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_3.request_url</param-n
ame>
  <param-value>/dir1/*</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_3.redirect_url</param-
name>
  <param-value>http://host/dir2/</param-value>
</param>
 :
```

この設定例の場合，実際のリクエスト URL が「/dir1/subdir1/index.html」のときには，Location ヘッダには「http://host/dir2/subdir1/index.html」が設定されます。なお，リクエスト URL の指定でワイルドカードを使用した場合でも，リダイレクト URL が「/」で終わらないときは，Location ヘッダの値は，リダイレクト URL と同じ値になります。

リダイレクトが行われる際，実際のリクエスト URL にクエリ文字列が付加されていた場合，Location ヘッダに設定される値は，リダイレクト URL の指定値にクエリ文字列を付加した値となります。また，リダイレクト URL にはクエリ文字列の付いた値を指定できます。この場合，実際のリクエスト URL にもクエリ文字列が付いているときは，Location ヘッダに設定される値は，リダイレクト URL の指定値の後ろにリクエストのクエリ文字列が付加された値となります。

6.7.4　リダイレクトによるリクエストの振り分けに関する注意事項

リダイレクトによるリクエストの振り分けに関する注意事項を次に示します。

- HTTP レスポンスが HTML ファイルの場合，その HTML からほかのファイル（画像ファイルなど）を参照していると，正しくブラウザに表示されない場合があります。
- リダイレクト URL の指定値とリクエスト URL の指定値が一致すると，クライアントによってはリダイレクトを実行し続けることがあるため注意してください。
- URL の書き換えによるセッション管理をしている場合，Web アプリケーション内へのリダイレクトを実行してもセッションを引き継ぐことはできません。

6.8　Persistent Connection による Web クライアントとの通信制御

この節では，Persistent Connection による Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 6-12　この節の構成（Persistent Connection による Web クライアントとの通信制御）

分類	タイトル	参照先
解説	Persistent Connection による通信制御	6.8.1
設定	実行環境での設定（J2EE サーバの設定）	6.8.2

注　「実装」および「運用」について，この機能固有の説明はありません。

6.8.1　Persistent Connection による通信制御

Persistent Connection は，Web クライアントとインプロセス HTTP サーバ間で確立した TCP コネクションを持続して，複数の HTTP リクエスト間で使用し続けるための機能です。Persistent Connection を使用することによって，Web クライアントと Web サーバ間でのコネクション接続に掛かる時間を短縮し，処理時間の短縮と通信トラフィックの軽減を図れます。

インプロセス HTTP サーバでは，次に示す項目を設定することで，Persistent Connection による通信制御を実現します。

- Persistent Connection 数の上限値

  Persistent Connection 数の上限値を設定することで，一つの TCP コネクションで連続してリクエストを処理できる Web クライアント数を制御します。TCP コネクション数が指定した上限値を超えた場合，リクエスト処理終了後に切断します。これによって，新規リクエストを処理するスレッドが確保でき，リクエスト処理スレッドを特定のクライアントに占有されることを防げます。

- Persistent Connection のリクエスト処理回数の上限値

  Persistent Connection のリクエスト処理回数の最大値を設定することで，同じ Web クライアントから連続してリクエスト要求があった場合の処理を制御します。

  Persistent Connection のリクエスト処理回数が指定した上限値を超えた場合，リクエスト処理終了後にコネクションを切断します。これによってリクエスト処理スレッドを特定のクライアントに占有し続けられることを防げます。

- Persistent Connection のタイムアウト

  Persistent Connection のリクエスト待ち時間にタイムアウトを設定することで，Persistent Connection のリクエスト待ち時間を制御します。指定したタイムアウト時間を超えてリクエスト処理要求がない場合は，TCP コネクションを切断します。これによって，使用されていない状態で TCP コネクションが占有され続けることを防げます。また，Persistent Connection のリクエスト待ち時間に

0 を指定してタイムアウトをしない設定にしている場合でも，リクエスト処理回数の上限値を超えたリクエスト要求があるとコネクションが切断されます。

なお，コネクションを切断された Web クライアントは，接続のリトライを実行して，再度リクエストを送信します。

6.8.2 実行環境での設定（J2EE サーバの設定）

Persistent Connection による通信制御を使用する場合，J2EE サーバの設定が必要です。

ここでは，Persistent Connection による通信制御の設定および設定例について説明します。

(1) J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Persistent Connection による通信制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Persistent Connection による通信制御の定義について次の表に示します。

表 6-13 簡易構築定義ファイルでの Persistent Connection による通信制御の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.persistent_connection.max_connections	一つの TCP コネクションで連続してリクエストを処理できる Web クライアント数を制御するため，Persistent Connection 数の上限値を指定します。
webserver.connector.inprocess_http.persistent_connection.max_requests	同じ Web クライアントから連続してリクエスト要求があった場合の処理を制御するため，Persistent Connection のリクエスト処理回数の上限値を指定します。
webserver.connector.inprocess_http.persistent_connection.timeout	Persistent Connection のリクエスト待ち時間を制御するため，Persistent Connection のタイムアウトを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

(2) 設定例

Persistent Connection による通信制御の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.persistent_connection.max_connections</para
m-name>
  <param-value>5</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.persistent_connection.max_requests</param-n
ame>
```

```
  <param-value>100</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.persistent_connection.timeout</param-name>
  <param-value>15</param-value>
</param>
 :
```

この設定例では，TCP コネクション数が「5」を超えた場合，またはリクエスト処理回数が「100」を超えた場合には，リクエスト処理終了後に TCP コネクションを切断します。また，タイムアウト時間「15」秒を過ぎてもリクエスト処理要求がない場合は，TCP コネクションを切断します。

6.9　通信タイムアウト（インプロセス HTTP サーバ）

この節では，インプロセス HTTP サーバでの通信タイムアウトによる Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 6-14　この節の構成（通信タイムアウト（インプロセス HTTP サーバ））

分類	タイトル	参照先
解説	通信タイムアウトの概要	6.9.1
設定	実行環境での設定（J2EE サーバの設定）	6.9.2

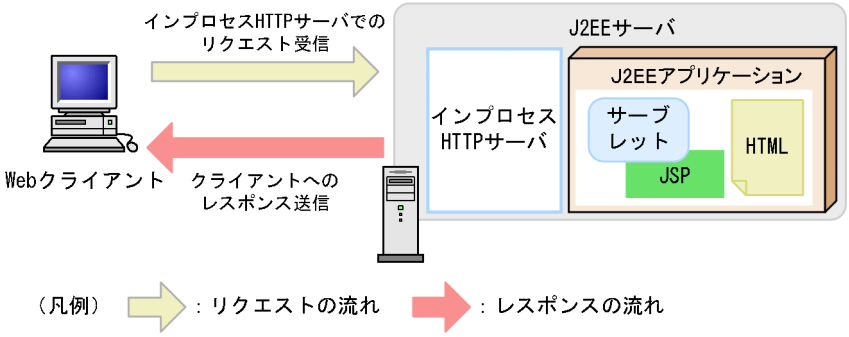
注　「実装」，「運用」および「注意事項」について，この機能固有の説明はありません。

6.9.1　通信タイムアウトの概要

インプロセス HTTP サーバを使用する場合，Web クライアントとインプロセス HTTP サーバ間での，リクエスト受信およびレスポンス送信に，通信タイムアウトを設定できます。ネットワークの障害やアプリケーションの障害などが発生し応答待ち状態になった場合，通信タイムアウトを設定していると，タイムアウトの発生によって障害の発生を検知できます。

インプロセス HTTP サーバを使用する場合，次に示す図中の二つの矢印が示す通信に対してタイムアウトを設定します。

図 6-7　タイムアウトを設定できる通信（インプロセス HTTP サーバを使用する場合）



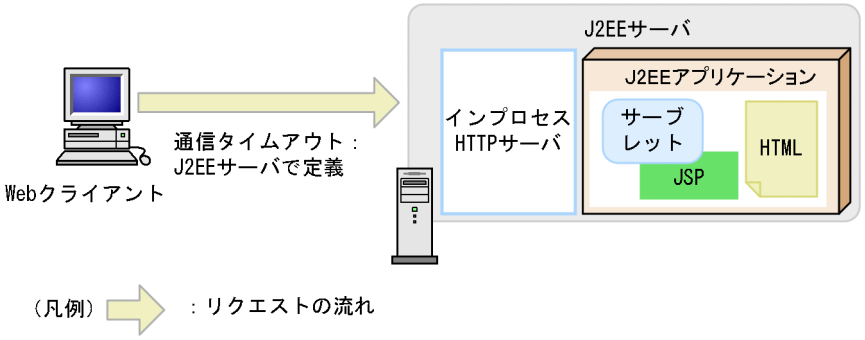
図に示すように，通信タイムアウトはリクエスト受信とレスポンス送信に対して設定します。通信タイムアウトの設定について，リクエストの受信とレスポンスの送信に分けて説明します。

なお，Web クライアントからのリクエスト受信，および Web クライアントへのレスポンス送信でタイムアウトが発生した場合，Web クライアントまたはネットワークで障害が発生したと見なされて，Web クライアントとの接続が切断されるため，レスポンスは返されません。

(1)　リクエスト受信時の通信タイムアウト

リクエスト受信時の通信タイムアウトの設定場所を次の図に示します。

図 6-8　リクエスト受信時の通信タイムアウトの設定場所（インプロセス HTTP サーバを使用する場合）



インプロセス HTTP サーバを使用する場合，Web クライアントとインプロセス HTTP サーバ間の通信に，タイムアウトを設定します。

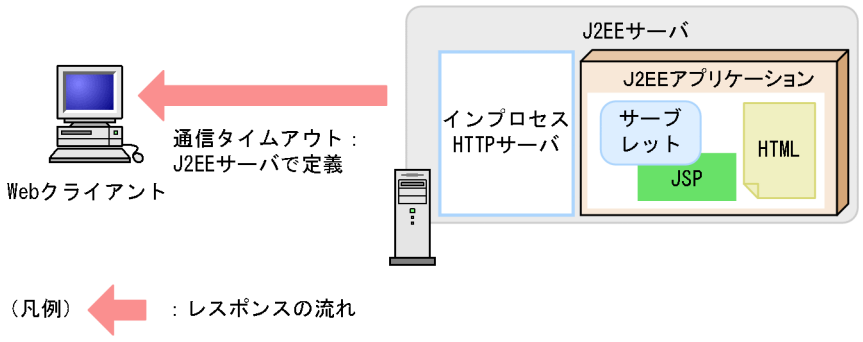
Web クライアントとインプロセス HTTP サーバ間の通信にタイムアウトを設定することによって，クライアント側で次に示す障害が発生したことを検知できます。

- Web クライアントが稼働するホストがダウンした
- Web クライアントとインプロセス HTTP サーバ間でネットワーク障害が発生した
- クライアントアプリケーションで障害が発生した

(2)　レスポンス送信時の通信タイムアウト

レスポンス送信時の通信タイムアウトの設定場所を次の図に示します。

図 6-9　レスポンス送信時の通信タイムアウトの設定場所（インプロセス HTTP サーバを使用する場合）



インプロセス HTTP サーバを使用する場合，インプロセス HTTP サーバと Web クライアント間の通信に，タイムアウトを設定します。

インプロセス HTTP サーバと Web クライアント間の通信にタイムアウトを設定することによって，次に示す障害を検知できます。

- Web クライアントが稼働するホストがダウンした
- Web クライアントとインプロセス HTTP サーバ間でネットワーク障害が発生した
- クライアントアプリケーションで障害が発生した

6.9.2　実行環境での設定（J2EE サーバの設定）

インプロセス HTTP サーバの通信タイムアウトの設定をする場合，J2EE サーバの設定が必要です。

ここでは，インプロセス HTTP サーバの通信タイムアウトの設定および設定例について説明します。

通信タイムアウトは，リクエストの受信時，またはレスポンスの送信時に設定します。リクエスト受信時およびレスポンス送信時の通信タイムアウトの設定についてそれぞれ説明します。

(1)　リクエスト受信時の通信タイムアウトの設定

リクエスト受信時の通信タイムアウトは，クライアントーインプロセス HTTP サーバ間に設定します。

インプロセス HTTP サーバでのリクエスト受信時の通信タイムアウトは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.receive_timeout
　クライアントからのリクエスト受信処理の待ち時間を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

(2)　レスポンス送信時の通信タイムアウトの設定

レスポンス送信時の通信タイムアウトは，インプロセス HTTP サーバークライアント間に設定します。

インプロセス HTTP サーバでのレスポンス送信時の通信タイムアウトは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.send_timeout
　クライアントへのレスポンス送信処理の待ち時間を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細は，「第 3 編　リファレンス（V9 互換モード）」を参照してください。

(3) 設定例

インプロセス HTTP サーバの通信タイムアウトの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.receive_timeout</param-name>
  <param-value>300</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.send_timeout</param-name>
  <param-value>600</param-value>
</param>
 :
```

この設定例では，リクエスト受信のタイムアウトに「300」秒，リクエスト送信のタイムアウトに「600」秒を設定しています。

6.10　IP アドレス指定（インプロセス HTTP サーバ）

この節では，インプロセス HTTP サーバでの IP アドレス指定による Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 6-15　この節の構成（IP アドレス指定（インプロセス HTTP サーバ））

分類	タイトル	参照先
解説	バインド先アドレス設定機能	6.10.1
設定	実行環境での設定（J2EE サーバの設定）	6.10.2
注意事項	インプロセス HTTP サーバでの IP アドレス指定をする場合の注意事項	6.10.3

注　「実装」および「運用」について，この機能固有の説明はありません。

参考

インプロセス HTTP サーバを使用しない場合（Web サーバ連携機能を使用する場合）との機能差異はありません。機能の解説については，「5.7　IP アドレス指定（Web サーバ連携）」を参照してください。

6.10.1　バインド先アドレス設定機能

Web コンテナでは，インプロセス HTTP サーバで利用する IP アドレスを明示的に指定できます。これを，**バインド先アドレス設定機能**といいます。この機能を使用することで，複数の物理ネットワークインタフェースを持つホスト，または一つの物理ネットワークインタフェースに対して，ホストで実行する場合に，特定の一つの IP アドレスだけを使用するように設定できます。

6.10.2　実行環境での設定（J2EE サーバの設定）

インプロセス HTTP サーバの IP アドレスの設定をする場合，J2EE サーバの設定が必要です。

ここでは，インプロセス HTTP サーバの IP アドレスの設定について説明します。

インプロセス HTTP サーバの IP アドレスは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.bind_host

インプロセス HTTP サーバで利用するホスト名または IP アドレスを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

6.10.3　インプロセス HTTP サーバでの IP アドレス指定をする場合の注意事項

インプロセス HTTP サーバでの IP アドレス指定をする場合の注意事項を次に示します。

- ホスト名または IP アドレスを設定した場合は，指定された IP アドレスへの接続要求しか受け付けません。IP アドレスを設定する代わりに，ワイルドカードアドレスを指定することで，そのホスト上の任意の IP アドレスへの接続を受け付けることができます。デフォルトでは，ワイルドカードアドレスを使用する設定になっています。なお，ワイルドカードアドレスを使用する場合は，次のことに注意してください。
  - 指定されたホスト名が，hosts ファイルまたは DNS などで解決できない場合は，ワイルドカードアドレスを使用してサーバを起動します。
  - 指定されたホスト名，または IP アドレスがリモートホストの場合，ワイルドカードアドレスを使用してサーバを起動します。

6.11　アクセスを許可するホストの制限によるアクセス制御

J2EE サーバへの不正アクセスを防ぐため，アクセスできるホストを制限できます。デフォルトでは，全ホストからのアクセスが可能な状態になっています。アクセスを許可するホストのホスト名や IP アドレスを設定しておくことで，特定のホストからのアクセスだけを許可して，不正アクセスを防止できます。

この節では，アクセスを許可するホストの制限によるアクセス制御について説明します。

表 6-16　この節の構成（アクセスを許可するホストの制限によるアクセス制御）

分類	タイトル	参照先
解説	アクセスを許可するホストの制限	6.11.1
設定	実行環境での設定（J2EE サーバの設定）	6.11.2

注 「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.11.1　アクセスを許可するホストの制限

ホストを制限するには，アクセスを許可するホストの，ホスト名，または IP アドレスを設定します。このとき，ホスト名や IP アドレスの代わりにアスタリスク（*）を指定すると，全ホストからのアクセスを許可する設定になります。アクセスを許可するホストをホスト名で指定した場合，J2EE サーバ起動時にホスト名が解決されます。なお，ローカルホストは明記しなくても常にアクセスが許可されます。通常，外部ネットワークからアクセスされるシステムでは，プロキシサーバの IP アドレスを指定します。

アクセスを許可するホストをホスト名で指定した場合の注意事項を次に示します。

注意事項

- hosts ファイル，または DNS などでホスト名を解決できるようにする必要があります。解決できない場合はデフォルトの設定でサーバが起動されます。
- ホスト名の解決を J2EE サーバの起動時にするため，サーバの起動に時間が掛かることがあります。また，起動後に変更された IP アドレスは反映されないことがあります。

6.11.2　実行環境での設定（J2EE サーバの設定）

アクセスを許可するホストの制限の設定をする場合，J2EE サーバの設定が必要です。

ここでは，アクセスを許可するホストの制限の設定方法と設定例について説明します。

(1)　設定方法

アクセスを許可するホストの制限は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.permitted.hosts

アクセスを許可するホストのホスト名や IP アドレスを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

注意事項

アクセスを許可するホストをホスト名で指定した場合の注意事項を次に示します。

- hosts，または DNS などでホスト名を解決できるようにする必要があります。解決できない場合はデフォルトの設定（全ホストからのアクセスが可能な状態）で J2EE サーバが起動されます。
- J2EE サーバの起動時にホスト名を解決するため，J2EE サーバの起動に時間が掛かることがあります。また，起動後に変更された IP アドレスは反映されないことがあります。

(2)　設定例

アクセスを許可するホストの制限の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.permitted.hosts</param-name>
  <param-value>host1,host2</param-value>
</param>
 :
```

この設定例では，「host1」と「host2」からのアクセスだけを許可し，ほかのホストからのアクセスを許可しません。

6.12　リクエストデータのサイズの制限によるアクセス制御

インプロセス HTTP サーバでは，一定のサイズ以下のリクエストデータだけを受け付けることによって，不正なリクエストデータの受け付けを拒否し，サーバへの負荷を抑え，安定した稼働を維持できます。

この節では，リクエストデータのサイズの制限によるアクセス制御について説明します。

この節の構成を次の表に示します。

表 6-17　この節の構成（リクエストデータのサイズの制限によるアクセス制御）

分類	タイトル	参照先
解説	リクエストデータのサイズの制限	6.12.1
設定	実行環境での設定（J2EE サーバの設定）	6.12.2

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.12.1　リクエストデータのサイズの制限

インプロセス HTTP サーバでは，一定のサイズ以下のリクエストデータだけを受け付けることによって，不正なリクエストデータの受け付けを拒否し，サーバへの負荷を抑え，安定した稼働を維持できます。

次に示す項目を設定することで，リクエストデータのサイズ制限によるアクセス制御を実現します。

- リクエストラインの長さの制限

  リクエストラインの長さの上限値を設定することで，アクセスを制御します。リクエストラインの長さには，HTTP メソッド，URI（クエリ文字列を含む），HTTP のバージョン，およびリクエストラインの終わりを示す改行文字（2 バイト）が含まれます。

  受信したリクエストラインの長さが上限値を超えた場合，Web クライアントに対して，ステータスコード 414 のエラーを返します。

- HTTP ヘッダ数の制限

  HTTP リクエストに含まれる HTTP ヘッダ数の上限値を設定することで，アクセスを制御します。

  受信した HTTP リクエストに含まれる HTTP ヘッダ数が上限値を超えた場合，Web クライアントに対してステータスコード 400 のエラーを返します。

- リクエストヘッダサイズの制限

  HTTP リクエストのリクエストヘッダのサイズに上限値を設定することによって，アクセスを制御します。

  受信した HTTP リクエストの HTTP ヘッダのサイズが上限値を超えた場合，Web クライアントに対してステータスコード 400 のエラーを返します。

- リクエストボディサイズの制限

HTTP リクエストのボディサイズに上限値を設定することによって，アクセスを制御します。インプロセス HTTP サーバでは，リクエストヘッダに含まれる Content-Length ヘッダの値で判断します。

HTTP リクエストのボディサイズが上限値を超えた場合，Web クライアントに対してステータスコード 413 のエラーを返します。

なお，チャンク形式でリクエストボディが送信された場合，サーブレット内部では指定された上限値までのデータを読み込みます。上限値を超えるとサーブレットで例外（IOException）が発生しますが，サーブレットの処理は続行されます。リクエストを送信したクライアントには，指定された上限値までのデータを読み込んだ結果に基づいて，アプリケーションが作成したレスポンスを返します。

ポイント

SSL アクセラレータや負荷分散機などのゲートウェイ機器や，プロキシサーバを配置していて，ゲートウェイ機器やプロキシサーバにリクエストデータサイズの制御機能がある場合は，その制御機能の設定値以下の値を設定する必要があります。

6.12.2　実行環境での設定（J2EE サーバの設定）

リクエストデータのサイズ制限の設定をする場合，J2EE サーバの設定が必要です。

ここでは，リクエストデータのサイズ制限の設定方法と設定例について説明します。

(1)　設定方法

J2EE サーバの設定は，簡易構築定義ファイルで実施します。リクエストデータのサイズ制限の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリクエストデータのサイズ制限の定義について次の表に示します。

表 6-18　簡易構築定義ファイルでのリクエストデータのサイズ制限の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.limit.max_request_line	リクエストラインの上限値を指定します。
webserver.connector.inprocess_http.limit.max_headers	HTTP リクエストに含まれる HTTP ヘッダ数の上限値を指定します。
webserver.connector.inprocess_http.limit.max_request_header	HTTP リクエストのリクエストヘッダのサイズの上限値を指定します。
webserver.connector.inprocess_http.limit.max_request_body	HTTP リクエストのボディサイズの上限値を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

ポイント

SSL アクセラレータや負荷分散機などのゲートウェイ機器や，プロキシサーバを配置していて，ゲートウェイ機器やプロキシサーバにリクエストデータサイズの制御機能がある場合は，その制御機能の設定値以下の値を設定する必要があります。

(2) 設定例

リクエストデータのサイズ制限の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_request_line</param-name>
  <param-value>1024</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_headers</param-name>
  <param-value>100</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_request_header</param-name>
  <param-value>8192</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_request_body</param-name>
  <param-value>16384</param-value>
</param>
 :
```

6.13　有効な HTTP メソッドの制限によるアクセス制御

この節では，有効な HTTP メソッドの制限によるアクセス制御について説明します。

この節の構成を次の表に示します。

表 6–19　この節の構成（有効な HTTP メソッドの制限によるアクセス制御）

分類	タイトル	参照先
解説	有効な HTTP メソッドの制限	6.13.1
設定	実行環境での設定（J2EE サーバの設定）	6.13.2

注 「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.13.1　有効な HTTP メソッドの制限

インプロセス HTTP サーバでは，HTTP リクエストで有効な HTTP メソッドを制限することによって，無効な HTTP メソッドを含むリクエストの受け付けを拒否します。これによって，サーバ上のリソースへの不正アクセスを防止できます。デフォルトでは，DELETE メソッド，HEAD メソッド，GET メソッド，OPTIONS メソッド，POST メソッド，および PUT メソッドの使用を許可しています。

HTTP メソッドを制限するには，有効な HTTP メソッドのメソッド名を設定します。有効な HTTP メソッドとして設定する値は，RFC2616 に規定されている値を使用する必要があります。ただし，メソッド名の文字列にアスタリスク（*）は使用できません。メソッド名の代わりにアスタリスク（*）を指定すると，すべてのメソッドの使用を許可する設定になります。

無効な HTTP メソッドを含むリクエストを受信した場合，Web クライアントに対してステータスコード 405 のエラーを返します。

なお，静的コンテンツに対して OPTIONS メソッドを含むリクエストを送信した場合，レスポンスに含まれる Allow ヘッダには静的コンテンツに対してデフォルトで有効なメソッド（GET メソッド，POST メソッド，TRACE メソッド，および OPTIONS メソッド）から，インプロセス HTTP サーバで無効なメソッドを除いたメソッドが返されます。サーブレット/JSP の場合，Web アプリケーションの実装に依存します。

6.13.2　実行環境での設定（J2EE サーバの設定）

有効な HTTP メソッドの制限の設定をする場合，J2EE サーバの設定が必要です。

ここでは，有効な HTTP メソッドの制限の設定方法と設定例について説明します。

(1) 設定方法

有効な HTTP メソッドの制限は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.enabled_methods

有効な HTTP メソッドのメソッド名を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

ポイント

静的コンテンツに対して OPTIONS メソッドを含むリクエストを送信した場合，静的コンテンツに対してデフォルトで有効なメソッド（GET メソッド，POST メソッド，TRACE メソッド，および OPTIONS メソッド）から，インプロセス HTTP サーバで無効なメソッドを除いたリクエストが返されます。サーブレット/JSP の場合，Web アプリケーションの実装に依存します。

(2) 設定例

有効な HTTP メソッドの制限の設定例を次に示します。なお，この設定例はデフォルトの設定です。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.enabled_methods</param-name>
  <param-value>GET,HEAD,POST,PUT,DELETE,OPTIONS</param-value>
</param>
 :
```

この設定例では，GET メソッド，HEAD メソッド，POST メソッド，PUT メソッド，DELETE メソッド，および OPTIONS メソッドに対してアクセスを許可し，TRACE メソッドに対してアクセスを拒否します。

6.14 HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ

この節では，HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズについて説明します。

この節の構成を次の表に示します。

表 6-20 この節の構成（HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ）

分類	タイトル	参照先
解説	HTTP レスポンスヘッダのカスタマイズ	6.14.1
設定	実行環境での設定（J2EE サーバの設定）	6.14.2

注 「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.14.1 HTTP レスポンスヘッダのカスタマイズ

ここでは，HTTP レスポンスヘッダのカスタマイズについて説明します。

インプロセス HTTP サーバでは，HTTP レスポンスの Server ヘッダに自動設定する情報をカスタマイズできます。デフォルトでは，「CosminexusComponentContainer」が自動設定されます。

Server ヘッダに自動設定する値は，RFC2616 に規定されている値を使用する必要があります。サーブレット/JSP で Server ヘッダを使用する設定にしている場合はその設定が優先されます。

6.14.2 実行環境での設定（J2EE サーバの設定）

HTTP レスポンスヘッダのカスタマイズの設定をする場合，J2EE サーバの設定が必要です。

ここでは，HTTP レスポンスヘッダのカスタマイズの設定方法と設定例について説明します。

(1) 設定方法

HTTP レスポンスヘッダのカスタマイズは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.response.header.server

HTTP レスポンスの Server ヘッダに自動設定する文字列を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

(2)　設定例

HTTP レスポンスヘッダのカスタマイズの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.response.header.server</param-name>
  <param-value>GyoumuServer/1.0</param-value>
</param>
 :
```

この設定例では，Server ヘッダの値として「GyoumuServer/1.0」を指定しています。

6.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）

クライアントから，存在しないリソースへのアクセスなどがあると，インプロセス HTTP サーバはエラーステータスコードとエラーページを返し，クライアントにはインプロセス HTTP サーバが生成したエラーページが表示されます。インプロセス HTTP サーバでは，このエラーページの代わりに，ユーザが作成したページをクライアントに表示させることができます。これを，**エラーページのカスタマイズ**と呼びます。

この節では，エラーページを使用した Web クライアントへのレスポンスのカスタマイズについて説明します。

この節の構成を次の表に示します。

表 6-21　この節の構成（エラーページのカスタマイズ（インプロセス HTTP サーバ））

分類	タイトル	参照先
解説	カスタマイズできるエラーページ	6.15.1
実装	エラーページのカスタマイズを実行する場合に必要な実装	6.15.2
設定	実行環境での設定（J2EE サーバの設定）	6.15.3
注意事項	エラーページのカスタマイズに関する注意事項	6.15.4

注　「運用」について，この機能固有の説明はありません。

6.15.1　カスタマイズできるエラーページ

存在しないリソースへのアクセスなどのエラーが発生したときに，エラーステータスコードが表示されたエラーページの代わりにユーザが作成したエラーページをクライアントに表示させることができます。

インプロセス HTTP サーバによるエラーページのカスタマイズを利用することで，特定のステータスコードに対応するエラーページのカスタマイズや，リクエスト URL に対応するエラーページのカスタマイズを Web コンテナ単位にまとめて制御できます。また，次に示すような Web アプリケーションによるエラーページのカスタマイズが実行できない場合でも，エラーページをカスタマイズできます。

- リクエストに対応するコンテキストがない場合（ステータスコード 404）
- 停止処理中のコンテキストでリクエスト処理を実行しようとした場合（ステータスコード 503）
- インプロセス HTTP サーバがエラーステータスコードを返す場合

インプロセス HTTP サーバが返すエラーステータスコードについては「付録 C.3　インプロセス HTTP サーバが返すエラーステータスコード」を参照してください。

インプロセス HTTP サーバでは，次に示すエラーページのカスタマイズができます。

- **ステータスコードに対応するエラーページのカスタマイズ**
  ステータスコード 400 番台，500 番台に対応するエラーページをカスタマイズできます。

ステータスコードに対応するエラーページをカスタマイズすることによって，ステータスコードに対応するファイルの送信と，ステータスコードに対応するリダイレクトを実行できます。

- ステータスコードに対応するファイルの送信

  カスタマイズするステータスコードに対して，レスポンスボディとして特定のファイルをクライアントに返すことができます。この場合，レスポンスの Content-Type ヘッダの値を指定します。

  なお，リクエスト処理時にファイルの読み込みに失敗した場合，デフォルトのエラーページが使用されます。

- ステータスコードに対応するリダイレクト

  カスタマイズするステータスコードに対して，特定の URL にリダイレクトできます。リダイレクトする場合，レスポンスのステータスコードに 302 を設定して，Location ヘッダにリダイレクト URL を指定します。

  ステータスコードに対応するファイルを送信する場合，リダイレクトは実行できません。

- **リクエスト URL に対応するエラーページのカスタマイズ**

  リクエスト URL を指定して，特定の URL のエラーページをカスタマイズできます。リクエスト URL を指定した場合，指定した URL と一致するリクエスト処理でエラーが発生した場合だけ，カスタマイズしたエラーページをクライアントに返します。

6.15.2 エラーページのカスタマイズを実行する場合に必要な実装

インプロセス HTTP サーバによるエラーページのカスタマイズを実行する場合，javax.servlet.http.HttpServletResponse インタフェースの sendError メソッドを使用してレスポンスのステータスコードを設定する必要があります。なお，setStatus メソッドを使用した場合（JSP で setStatus メソッドを使用した場合など），インプロセス HTTP サーバによるカスタマイズが実行されないことがあります。ただし，sendError メソッドを使用しても，Web アプリケーションが次に示すどちらかの条件に該当する場合，インプロセス HTTP サーバによるエラーページのカスタマイズは実行されません。

- sendError メソッドの実行時に例外が発生した場合
- Web アプリケーションでエラーページのカスタマイズの設定をしていて，エラー発生時にその設定によるエラーページの実行が正常終了した場合※

注※

エラーページの実行が正常終了した場合とは，次の条件を満たす場合のことです。

- エラーページで catch されない例外が発生していない。
- ステータスコードが 400～599 以外で終了している。

6.15.3 実行環境での設定（J2EE サーバの設定）

エラーページのカスタマイズを実行する場合，J2EE サーバの設定が必要です。

ここでは，エラーページのカスタマイズの設定方法と設定例について説明します。

(1) 設定方法

J2EE サーバの設定は，簡易構築定義ファイルで実施します。エラーページのカスタマイズの定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのエラーページのカスタマイズの定義について次の表に示します。

表 6-22 簡易構築定義ファイルでのエラーページのカスタマイズの定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.error_custom.list	エラーページのカスタマイズ定義名を指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.status	エラーステータスコードに対応づけるエラーページのカスタマイズをする場合に，エラーページをカスタマイズするエラーステータスコードを指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.file	エラーステータスコードに対応するファイルを送信する場合に，レスポンスボディとしてクライアントに返すファイルを指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.file.content_type	webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.file パラメタに指定したファイルをレスポンスボディとしてクライアントに送信する際の Content-Type ヘッダの値を指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.redirect_url	エラーステータスコードに対応するリダイレクトをする場合に，リダイレクト先の URL を指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.request_url	リクエスト URL に対応づけるエラーページのカスタマイズをする場合に，エラーページのカスタマイズを適用するリクエスト URL を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3 簡易構築定義ファイル」を参照してください。

(2) 設定例

エラーページのカスタマイズの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
  <param-value>ERR_CUSTOM_1,ERR_CUSTOM_2</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.status</param-nam
e>
  <param-value>404</param-value>
</param>
<param>
```

```
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.file</param-name>
  <param-value>C:/data/404.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.file.content_type
</param-name>
  <param-value>text/html; charset=ISO-8859-1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.status</param-nam
e>
  <param-value>503</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.redirect_url</par
am-name>
  <param-value>http://host1/503.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.request_url</para
m-name>
  <param-value>/dir1/*</param-value>
</param>
 :
```

この設定例では，エラーページのカスタマイズ定義名として，「ERR_CUSTOM_1」と「ERR_CUSTOM_2」を使用しています。「ERR_CUSTOM_1」では，レスポンスのステータスコードが「404」の場合には，「C:/data/404.html」をクライアントに返します。Content-Type ヘッダの値は，「text/html; charset=ISO-8859-1」を使用します。また，「ERR_CUSTOM_2」では，リクエストが「/dir1/」から始まる URL で，レスポンスのステータスコードが「503」の場合に，「http://host1/503.html」にリダイレクトします。

6.15.4　エラーページのカスタマイズに関する注意事項

インプロセス HTTP サーバによるエラーページのカスタマイズに関する注意事項を次に示します。

- HTTP レスポンスが HTML ファイルの場合，その HTML からほかのファイル（画像ファイルなど）を参照していると，エラーページが正しく表示されない場合があります。
- ブラウザの設定によっては，ステータスコードがエラーを示し，レスポンスボディのサイズが小さい場合，レスポンスボディがブラウザ固定のメッセージに置き換えられて表示されることがあるため注意してください。特にエラーページをカスタマイズしなかった場合に表示されるデフォルトのエラーページは，レスポンスボディのサイズが小さいため注意してください。
- リダイレクト URL の指定値とリクエスト URL の指定値が一致して，リダイレクト後に同じエラーステータスが発生すると，クライアントによってはリダイレクトを実行し続けることがあるため注意してください。
- ステータスコードに対応するリダイレクトを実施する場合，エラー発生時にリクエスト URL に付加されていたクエリ文字列はリダイレクト URL に付加されません。また，URL の書き換えによるセッショ

ン管理をしている場合，エラーの発生したページと同じ Web アプリケーション内へのリダイレクトを実施してもセッションを引き継ぐことはできません。

6.16　Web コンテナへのゲートウェイ情報の通知

この節では，Web コンテナへのゲートウェイ情報の通知について説明します。

この節の構成を次の表に示します。

表 6-23　この節の構成（Web コンテナへのゲートウェイ情報の通知）

分類	タイトル	参照先
解説	ゲートウェイ指定機能	6.16.1
設定	実行環境での設定（J2EE サーバの設定）	6.16.2
注意事項	Web コンテナにゲートウェイ情報を通知する場合の注意事項	6.16.3

注　「実装」および「運用」について，この機能固有の説明はありません。

参考

インプロセス HTTP サーバを使用しない場合（Web サーバ連携機能を使用する場合）との機能差異はありません。機能の解説については，「5.10　Web コンテナへのゲートウェイ情報の通知」を参照してください。

6.16.1　ゲートウェイ指定機能

クライアントとインプロセス HTTP サーバとの間に，SSL アクセラレータや負荷分散機などのゲートウェイを配置している場合で，welcome ファイルや Form 認証画面への遷移時などに Web コンテナが自動的にリダイレクトするとき，Web コンテナではゲートウェイの情報を得ることができないで，転送先の URL を正しく作成できないことがあります。

これを解決するために，**ゲートウェイ指定機能**を使用します。ゲートウェイ指定機能によって，Web コンテナにゲートウェイ情報を通知し，welcome ファイルや Form 認証画面に正しくリダイレクトできるようになります。

ゲートウェイ指定機能は次のような場合に使用できます。

- **クライアントとインプロセス HTTP サーバとの間に SSL アクセラレータを配置する場合**

  クライアントから SSL アクセラレータへのアクセスが HTTPS の場合でも，SSL アクセラレータから Web サーバへのアクセスは HTTP となるため，Web コンテナは HTTP によるアクセスであると認識します。このため，welcome ファイルや Form 認証画面へのリダイレクト先 URL のスキームは HTTP となります。

  この場合，ゲートウェイ指定機能を使用して，スキームを常に https と見なすように指定することで，正しくリダイレクトできるようになります。

- **Host ヘッダのないリクエストに対して，リクエストを受けたインプロセス HTTP サーバ以外へリダイレクトする必要がある場合**

  Host ヘッダのないリクエストをリダイレクトする場合，リダイレクト先 URL のホスト名・ポート番号は，リクエストを受けた Web サーバのホスト名・ポート番号となります。

  ゲートウェイ指定機能は，Web サーバまたはインプロセス HTTP サーバの前に負荷分散機を配置している場合などで，クライアントがアクセスする URL のホスト名・ポート番号が，リクエストを受けた Web サーバまたはインプロセス HTTP サーバと異なるときに使用します。これによって，クライアントからアクセスするホスト名・ポート番号が指定されるので，正しくリダイレクトできるようになります。

なお，インプロセス HTTP サーバを使用する場合，一つの Web コンテナに複数の異なる経路でアクセスする場合（複数のゲートウェイから Web コンテナに HTTP リクエストが転送される場合など），ゲートウェイ指定機能を使用できません。インプロセス HTTP サーバを使用する場合，ゲートウェイ指定機能を使用するには，Web コンテナへのアクセス経路は一つになる構成にする必要があります。

6.16.2　実行環境での設定（J2EE サーバの設定）

Web コンテナにゲートウェイ情報を通知するための設定をする場合，J2EE サーバの設定が必要です。

ここでは，Web コンテナにゲートウェイ情報を通知するための設定方法と設定例について説明します。

(1)　設定方法

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web コンテナにゲートウェイ情報を通知するための設定の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Web コンテナにゲートウェイ情報を通知するための設定の定義について次の表に示します。

表 6-24　簡易構築定義ファイルでの Web コンテナにゲートウェイ情報を通知するための設定の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.gateway.https_scheme	リダイレクト先 URL のスキームを指定します。
webserver.connector.inprocess_http.gateway.host	ゲートウェイのホスト名を指定します。
webserver.connector.inprocess_http.gateway.port	ゲートウェイのポート番号を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

(2) 設定例

ゲートウェイ指定機能の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.gateway.host</param-name>
  <param-value>host1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.gateway.port</param-name>
  <param-value>4443</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.gateway.https_scheme</param-name>
  <param-value>true</param-value>
</param>
 :
```

この設定例では，ゲートウェイ指定機能を使用して，スキームを常に HTTPS と見なすように設定しています。

6.16.3 Web コンテナにゲートウェイ情報を通知する場合の注意事項

ゲートウェイ指定機能を使用する上での注意事項を次に示します。

リダイレクト先 URL のホスト名，およびポート番号の指定について

通常，ブラウザは Host ヘッダを付けてリクエストを送信するため，リダイレクト先 URL のホスト名やポート番号を指定する必要はありません。

なお，リクエストに Host ヘッダがあるかどうかは，javax.servlet.http.HttpServletRequest クラスの getHeader メソッドに，引数「Host」を指定して呼び出すことで確認できます。

サーブレット API の動作について

ゲートウェイ指定機能の利用によって，一部のサーブレット API の動作が変わります。Web アプリケーションで API を利用するときには注意が必要です。

なお，動作が変わるサーブレット API については，「9.4.2(1) ゲートウェイ指定機能を使用する場合の注意」を参照してください。

web.xml の<transport-guarantee>タグについて

ゲートウェイ指定機能でスキームを HTTPS と見なすように設定した場合，Web サーバへのリクエストが HTTP であっても HTTPS であると見なされます。このため，web.xml の<transport-guarantee>タグで INTEGRAL や CONFIDENTIAL を指定しても HTTPS の URL へリダイレクトされないので注意してください。

Cookie の Secure 属性について

ゲートウェイ指定機能でスキームを HTTPS と見なすように設定している場合に，Web コンテナが生成したセッション ID を，Cookie によってクライアントに返すとき，その Cookie には Secure 属性が付与されます。

6.17　ログ・トレースの出力

この節では，インプロセス HTTP サーバが出力するログ・トレースについて説明します。

この節の構成を次の表に示します。

表 6–25　この節の構成（ログ・トレースの出力）

分類	タイトル	参照先
解説	インプロセス HTTP サーバが出力するログ・トレース	6.17.1
設定	インプロセス HTTP サーバのアクセスログのカスタマイズ	6.17.2

注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

6.17.1　インプロセス HTTP サーバが出力するログ・トレース

インプロセス HTTP サーバでは，アプリケーション開発のサポート，運用時の性能解析，および障害発生時のトラブルシュートのために，次の表に示すログおよびトレースを出力します。

表 6–26　インプロセス HTTP サーバが出力するログ・トレース

ログ・トレースの種類	説明
アクセスログ	Web クライアントからのリクエスト，およびレスポンスの処理結果を出力します。Web クライアントとの通信の解析に使用します。 アクセスログを解析することによって，Web クライアントからリクエストされたファイルや，インプロセス HTTP サーバの性能情報，およびセッショントラッキング情報などを分析できます。
性能解析トレース	インプロセス HTTP サーバでリクエストを送受信するときの性能解析情報や，障害発生時のトラブルシュートのための情報を出力します。 性能解析トレースは，CSV 形式などに変換して，ほかの J2EE サーバの各機能が出力する性能解析情報とあわせて，システム全体のボトルネックの解析などに使用できます。性能解析トレースの詳細については「14.1　性能解析トレースの概要」を参照してください。
スレッドトレース	保守用のトレースです。
通信トレース	保守用のトレースです。

6.17.2　インプロセス HTTP サーバのアクセスログのカスタマイズ

インプロセス HTTP サーバでは，アプリケーション開発のサポート，運用時の性能解析，および障害発生時のトラブルシュートのために，アクセスログ，性能解析トレース，スレッドトレース，および通信トレースを出力します。これらのファイルは，ファイル面数やファイルサイズなどを変更できます。また，アクセスログでは，ログの出力形式をカスタマイズできます。

ここでは，インプロセス HTTP サーバのアクセスログの出力形式のカスタマイズについて説明します。インプロセス HTTP サーバのアクセスログおよびトレースファイルのファイル面数やファイルサイズなどの変更については「15.2　インプロセス HTTP サーバのログ取得の設定」を参照してください。

(1)　カスタマイズの手順

アクセスログの出力形式のカスタマイズの手順を次に示します。

1. アクセスログのフォーマット名を定義します。

   フォーマットを新規作成する場合には，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.logger.access_log.format_list パラメタで，新規に作成するフォーマット名を追加します。

   設定例

```
 :
<param>
  <param-name>webserver.logger.access_log.format_list</param-name>
  <param-value>formatA</param-value>
</param>
 :
```

   フォーマットを新規作成する場合には，デフォルトで提供されているアクセスログのフォーマットを参考にしてください。フォーマットについては，「6.17.2(2)　アクセスログのフォーマット」を参照してください。

2. アクセスログの出力形式を定義します。

   簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.logger.access_log.<フォーマット名>パラメタで，<フォーマット名>で指定したフォーマットでのアクセスログの出力形式を定義します。出力形式の定義では，アクセスログのフォーマットの引数を指定します。

   設定例

```
 :
<param>
  <param-name>webserver.logger.access_log.formatA</param-name>
  <param-value>%h %u %t &quot;%r&quot; %&gt;s HostHeader=&quot;%{host}i&quot;</param-v
alue>
</param>
 :
```

   指定できるフォーマットの引数については，「6.17.2(3)　アクセスログのフォーマットの引数」を参照してください。

3. アクセスログの出力に使用するフォーマットを指定します。

   簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.logger.access_log.inprocess_http.usage_format パラメタで，アクセスログの出力に使用

するフォーマット名を指定します。ここで指定したフォーマットで，アクセスログが出力されるようになります。

設定例

```
 :
<param>
  <param-name>webserver.logger.access_log.inprocess_http.usage_format</param-name>
  <param-value>formatA</param-value>
</param>
 :
```

(2) アクセスログのフォーマット

アプリケーションサーバでは，インプロセス HTTP サーバのアクセスログのフォーマットとして，common（デフォルトフォーマット）と combined（拡張フォーマット）という 2 種類のフォーマットを提供しています。フォーマットを新規作成する場合には，これらのフォーマットを参考にしてください。

(a) 出力形式

アクセスログの出力形式について説明します。なお，△は，半角スペースを表します。また，ここでは，表記の都合上，複数行にわたっていますが，実際には 1 行で出力されます。

デフォルトフォーマットの出力形式を次に示します。

```
Webクライアントのホスト名またはIPアドレス△リモートログ名△認証ユーザ名
△Webクライアントからのリクエストの処理を開始した時刻△リクエストライン
△最終ステータスコード△HTTPヘッダを除く送信バイト数
```

拡張フォーマットの出力形式を次に示します。

```
Webクライアントのホスト名またはIPアドレス△リモートログ名△認証ユーザ名
△Webクライアントからのリクエストの処理を開始した時刻△リクエストライン
△最終ステータスコード△HTTPヘッダを除く送信バイト数
△"Refererヘッダの内容"△"User-Agentヘッダの内容"
```

下線部分が，デフォルトフォーマットと拡張フォーマットの差異です。拡張フォーマットでは，デフォルトフォーマットの出力内容に加えて，「Referer ヘッダの内容」と「User-Agent ヘッダの内容」が出力されます。

(b) 出力例

デフォルトフォーマットでのアクセスログの出力例を次に示します。

```
10.20.30.40 - user [20/Dec/2004:15:45:01 +0900] "GET /index.html HTTP/1.1" 200 8358
10.20.30.40 - user [20/Dec/2004:15:45:01 +0900] "GET /left.html HTTP/1.1" 200 2358
10.20.30.40 - user [20/Dec/2004:15:45:01 +0900] "GET /right.html HTTP/1.1" 200 4358
```

拡張フォーマットでのアクセスログの出力例を次に示します。

```
10.20.30.40 - - [18/Jan/2005:13:06:10 +0900] "GET / HTTP/1.0" 200 38 "-" "Mozilla/4.0 (compa
tible; MSIE 6.0; Windows NT 5.1)"
10.20.30.40 - - [18/Jan/2005:13:06:25 +0900] "GET /demo/ HTTP/1.0" 500 684 "-" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)"
```

(3) アクセスログのフォーマットの引数

フォーマットの出力形式を定義するときに指定する，アクセスログのフォーマットの引数を次の表に示します。

表 6-27 アクセスログのフォーマットの引数の一覧

フォーマットの引数	出力内容	出力例
%%	%記号	%
%a	Web クライアントの IP アドレス	10.20.30.40
%A	J2EE サーバの IP アドレス	10.20.30.100
%b	HTTP ヘッダを除く送信バイト数 (0 バイトの場合は「-」となる)	2048
%B	HTTP ヘッダを除く送信バイト数 (0 バイトの場合は「0」となる)	1024
%h	Web クライアントのホスト名または IP アドレス (ホスト名を取得できない場合は IP アドレスとなる)	10.20.30.40
%H	リクエストプロトコル	HTTP/1.1
%l	リモートログ名※1 (常に「-」になる)	-
%m	リクエストメソッド	GET
%p	Web クライアントからのリクエストを受け付けたポート番号	80
%q	クエリ文字列 (「?」から始まる。クエリ文字列がない場合は空文字となる)	?id=100&page=15
%r	リクエストライン	GET /index.html HTTP/1.1
%>s	最終ステータスコード (内部リダイレクトされた値は出力しない)	200
%S※2	ユーザのセッション ID (セッション ID がない場合は「-」となる)	00455AFE4DA4E7B7789F247B8FE5D605
%t	Web クライアントからのリクエストの処理を開始した時刻 (単位：秒，出力形式：dd/MMM/YYYY:HH:mm:ss Z)	[18/Jan/2005:13:06:10 +0900]
%T	Web クライアントのリクエストの処理に掛かった時間 (単位：秒)	2

フォーマットの引数	出力内容	出力例
%d	Web クライアントからのリクエストの処理を開始した時刻 (単位：ミリ秒，出力形式：dd/MMM/YYYY:HH:mm:ss.nnn Z (nnn はミリ秒))	[18/Jan/2005:13:06:10.152 +0900]
%D	Web クライアントのリクエストの処理に掛かった時間 (単位：ミリ秒)	2000
%u	Basic 認証ユーザ名，Form 認証ユーザ名 (認証ユーザ名がない場合は「-」となる)	user
%U	リクエストファイルパス	/index.html
%v	J2EE サーバのローカルホスト名	server
%{foo}i※3	リクエストヘッダ foo の内容 (foo ヘッダが存在しない場合は「-」となる)	%{Host}i の場合 www.example.com:8888
%{foo}c	Web クライアントが送信した Cookie 情報で Cookie の名前が foo の内容 (Cookie の名前に foo がない場合は「-」となる)	%{JSESSIONID}c の場合 00455AFE4DA4E7B7789F247B8FE5D605
%{foo}o※3	レスポンスヘッダ foo の内容 (foo ヘッダが存在しない場合は「-」となる)	%{Server}o の場合 CosminexusComponentContainer

注※1

リモートログ名は，RFC 1413 で規定されている Identification プロトコルで取得できる Web クライアント側のユーザ名です。

注※2

%S で表示されるセッション ID は，Cookie 名「JSESSIONID」の値です。

注※3

一度の HTTP リクエストまたは HTTP レスポンスで同じヘッダ名を複数回送信する場合があります。この場合，最初に読み込んだヘッダの内容を出力します。

注意事項

フォーマットの引数の指定に誤りがある場合には，デフォルトフォーマットが使用されます。デフォルトフォーマットが使用される例を次に示します。

- フォーマットの引数の一覧にない文字列（例：%G など）を指定した場合
- リクエストヘッダの内容，レスポンスヘッダの内容，Cookie 名で 0 文字（%{}i など）を指定した場合

参考

デフォルトフォーマットと拡張フォーマットをフォーマットの引数で記述すると，次のような形式になります。

- デフォルトフォーマット
  %h %l %u %t "%r" %>s %b
- 拡張フォーマット
  %h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-Agent}i"

6.18　URI のデコード機能

URI のデコード機能は，Web サーバ連携，およびインプロセス HTTP サーバで利用できます。

この節では，URI のデコード機能について説明します。

この節の構成を次の表に示します。

表 6-28　この節の構成（URI のデコード機能）

分類	タイトル	参照先
解説	URI のデコード機能の概要	6.18.1
設定	実行環境での設定（J2EE サーバの設定）	6.18.2
注意事項	URI のデコード機能を使用する場合の注意事項	6.18.3

注　「実装」および「運用」について，この機能固有の説明はありません。

6.18.1　URI のデコード機能の概要

URI のデコード機能を使用すると，アプリケーションサーバでは，リクエスト URI のサーブレットパスおよび追加のパス情報に含まれる URL エンコードされた文字列がデコードされます。ただし，コンテキストパスはデコードされません。

デコードされた URI を使用しない Web アプリケーションを実行するときは，URI のデコード機能を使用しないように設定するか，Web アプリケーション側で対応する必要があります。

URI のデコード機能の使用時に影響がある Servlet API，デコードされた文字列を使用する機能，デコード時に使用される文字コード，および文字列のデコードと正規化の実行順序を次に記述します。

(1)　URI のデコード機能の使用時に影響がある Servlet API

URI のデコード機能を使用する場合，javax.servlet.http.HttpServletRequest インタフェースの次のメソッドでは，デコードされた URI が戻り値となります。

- getPathInfo メソッド
- getPathTranslated メソッド
- getServletPath メソッド

ただし，getRequestURI メソッドおよび getRequestURL メソッドでは，デコードされていない URL が戻り値となります。

(2) デコードされた文字列を使用する機能

URI のデコード機能を使用する場合，次に示すマッチング処理で，デコードされた文字列が使用されます。

- サーブレットおよび JSP の URL パターンとのマッチング
- デフォルトマッピングとのマッチング
- 静的コンテンツとのマッチング
- フィルタの URL パターンとのマッチング
- web.xml の<error-page>タグ，または JSP の page ディレクティブの errPage 属性で指定するエラーページとのマッチング
- アクセスを制限する URL パターンとのマッチング
- ログイン認証の URL 判定
- リクエストのフォワードおよびインクルード
- HTTP レスポンス圧縮フィルタの URL パターンとのマッチング
- URL グループ単位の同時実行スレッド数制御の URL パターンとのマッチング

ただし，コンテキストパスはデコードされないで，元の文字列のまま扱われるため，コンテキストルートと一致しない場合は「404 Not Found」が戻り値となります。

また，アプリケーションサーバの次の機能では，デコード後の文字列でのマッチングは行われません。

- インプロセス HTTP サーバのエラーページのカスタマイズ機能
- インプロセス HTTP サーバのリダイレクトによるリクエストの振り分け機能

(3) デコード時に使用される文字コード

URI のデコード機能を使用する場合，デコード時に使用される文字コードは UTF-8 です。

(4) 文字列のデコードと正規化の実行順序

クライアントから送信されたリクエスト URI で，マッチングに利用される URL は，デコードされたあとで正規化されます。

6.18.2 実行環境での設定（J2EE サーバの設定）

URI のデコード機能を使用する場合，J2EE サーバの設定が必要です。

J2EE サーバの設定は，簡易構築定義ファイルで実施します。URI のデコード機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の

webserver.http.request.uri_decode.enabled に指定します。このパラメタでは，URI のデコード機能を使用するかどうかを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

6.18.3　URI のデコード機能を使用する場合の注意事項

ここでは，URI のデコード機能を使用する場合の注意事項について説明します。

(1)　文字列のデコードと正規化の実行順序

サーブレットパスと URL パターンのマッチングでは，デコードされたあとで正規化された URI が使用されます。

コンテキストパスとコンテキストルートのマッチングでは，デコードされないで正規化された URI が使用されます。

(2)　リクエストの属性

フォワード時またはインクルード時にリクエストに追加される属性にも，デコードされた値が格納されるものがあります。フォワード時またはインクルード時にリクエストに設定される各属性について，格納される値がデコードされるかどうかを次の表に示します。

処理	属性	格納される値のデコードの実行
フォワード	javax.servlet.forward.request_uri	×
	javax.servlet.forward.context_path	×
	javax.servlet.forward.servlet_path	○
	javax.servlet.forward.path_info	○
	javax.servlet.forward.query_string	×
インクルード	javax.servlet.include.request_uri	×
	javax.servlet.include.context_path	×
	javax.servlet.include.servlet_path	○
	javax.servlet.include.path_info	○
	javax.servlet.include.query_string	×

(凡例)

○：デコードされる

×：デコードされない

各属性および各属性に格納される値については，サーブレット仕様書を参照してください。

(3) HTTP セッションの引き継ぎ

コンテキストパスはデコードされないで，元の文字列のまま扱われるため，HTTP セッションは引き継がれます。

6.19　インプロセス HTTP サーバのログ取得の設定

ここでは，インプロセス HTTP サーバのログ取得で設定できる項目について説明します。

インプロセス HTTP サーバでは，アプリケーション開発のサポート，運用時の性能解析，および障害発生時のトラブルシューティングのために，アクセスログ，性能解析トレース，スレッドトレース，および通信トレースを出力します。これらのファイルでは，ファイル面数やファイルサイズを簡易構築定義ファイルで変更できます。

インプロセス HTTP サーバのログ取得で変更できる設定項目と，項目に対応する簡易構築定義ファイルのパラメタを次の表に示します。

表 6-29　インプロセス HTTP サーバのログ取得の設定項目

ログおよびトレース	項目	対応する簡易構築定義ファイルのパラメタ
アクセスログ	アクセスログの出力の有無	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.enabled （デフォルトではアクセスログが出力されます）
	アクセスログのファイル名	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.filename
	アクセスログのファイルサイズ	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.filesize
	アクセスログのファイル面数	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.filenum
	アクセスログのフォーマット名	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.format_list※
	アクセスログの出力形式	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.<フォーマット名>※
	アクセスログ出力時のフォーマット	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.usage_format※
性能解析トレース	－	ほかの性能解析トレースと同様に，日常的なシステム運用の作業で，cprfed コマンドを実行するときに取得条件などを指定します。性能解析トレースファイルの取得については，「14. 性能解析トレース」を参照してください。
スレッドトレース	スレッドトレースのファイル面数	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.thread_trace.inprocess_http.filenum
	スレッドトレースのファイルサイズ	スレッドトレースのファイルサイズ＝（((A+B）×32,786）＋ 32,914）バイト A ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の

ログおよびトレース	項目	対応する簡易構築定義ファイルのパラメタ
		webserver.connector.inprocess_http.max_connections パラメタの値 B ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.send_timeout パラメタの値が 0 の場合 0，0 以外の場合 1
通信トレース	通信トレースのファイル面数	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.communication_trace.inprocess_http.filenum
	通信トレースのファイルサイズ	通信トレースのファイルサイズ＝（((A+B）×172,050）＋128）バイト A ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.max_connections パラメタの値 B ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.send_timeout パラメタの値が 0 の場合は 0，0 以外の場合は 1

（凡例）－：該当しない

注※　アクセスログでは，これらのキーでフォーマットを定義することで，ログの出力形式をカスタマイズできます。インプロセス HTTP サーバのアクセスログのカスタマイズについては，「6.17.2　インプロセス HTTP サーバのアクセスログのカスタマイズ」を参照してください。

6.20 cjtracesync（インプロセス HTTP サーバ用トレースファイルの同期）

この節では，インプロセス HTTP サーバ用トレースファイルを同期させる cjtracesync コマンドについて説明します。

形式

```
cjtracesync [-h] [-thr|-comm] <サーバ名称>
```

機能

インプロセス HTTP サーバ使用時に，スレッドトレースおよび通信トレースの情報を共有メモリの情報と同期させます。インプロセス HTTP サーバを使用している場合に，Management Server の機能を利用しないで障害情報を取得するときには，情報を取得する直前に，このコマンドを実行してください。

引数

-h
　コマンドの使用方法が表示されます。

-thr
　スレッドトレース情報を取得する場合に指定します。-thr オプション，-comm オプションのどちらも指定されていない場合，スレッドトレースおよび通信トレースの両方の情報を更新します。

-comm
　通信トレース情報を取得する場合に指定します。-thr オプション，-comm オプションのどちらも指定されていない場合，スレッドトレースおよび通信トレースの両方の情報を更新します。

<サーバ名称>
　トレース情報を取得する J2EE サーバのサーバ名称を指定します。サーバ名称は必ず指定してください。

戻り値

0：
　正常終了しました。

1：
　異常終了しました。

9：
　管理者特権がないため，コマンドが実行できません（Windows の場合）。

6.21 SOAP アプリケーション運用時の注意事項

SOAP アプリケーション運用時の注意事項について説明します。

6.21.1 インプロセス HTTP サーバを使用している J2EE サーバを停止する場合の注意

インプロセス HTTP サーバを使用している場合，J2EE サーバを停止しようとしても停止しないか，停止までに長時間掛かることがあります。

これは，次の条件がすべて重なる場合に発生します。

- J2EE サーバを停止しようとした場合
- SOAP サービス側の Web サーバにインプロセス HTTP サーバを使用している場合
- webserver.connector.inprocess_http.persistent_connection.timeout の値を大きく設定しているか，0 を設定している場合
- 運用中の SOAP サービスにコネクションプーリングが有効な SOAP クライアントが接続中であるか，一度でも接続したことがある場合

設定内容および停止方法ごとに，発生する可能性のある現象を次に示します。

- インプロセス HTTP サーバの webserver.connector.inprocess_http.persistent_connection.timeout の値を大きく設定している場合

  cjstopsv コマンドで J2EE サーバを停止するときに KDJE39514-W が出力され，最大でこのプロパティに設定した秒数分の時間が掛かることがあります。
- インプロセス HTTP サーバの webserver.connector.inprocess_http.persistent_connection.timeout の値に 0 を設定している場合

  J2EE サーバが停止しないことがあります。
- Management Server を使用して停止した場合

  論理サーバの停止に失敗したことを表すメッセージ（KEOS20011-E）が出力され，強制停止することがあります。

cjstopsv コマンド実行時に KDJE39514-W が出力されて，J2EE サーバの停止に時間が掛かる場合は，cjstopsv コマンドに-f オプションを付けて再度実行し，J2EE サーバを停止してください。

Management Server を使用して停止した場合，停止監視時間を設定しているときは，停止監視時間が経過すると強制停止されます。停止監視時間を設定していないときは，次の手順で J2EE サーバを停止してください。

1. 運用管理ポータルで［論理サーバの起動/停止］アンカーをクリックします。

2. [サーバビュー] タブの [論理 J2EE サーバ] − [J2EE サーバ] − [<J2EE サーバ名>] をクリックします。

3. [起動/停止] タブをクリックします。

4. [強制停止] ボタンをクリックします。

7 Cosminexus JAX-RS エンジン（JAX-RS 1.1）

V9 互換モードでの Cosminexus JAX-RS エンジン（JAX-RS 1.1）について説明します。

7.1　V9 互換モードでの Cosminexus JAX-RS エンジン（JAX-RS 1.1）

V9 互換モードでは，JAX-RS 機能は Cosminexus JAX-RS エンジン（JAX-RS 1.1）だけ使用できます。Cosminexus JAX-RS エンジン（JAX-RS 1.1）については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」を参照してください。

8 CJPA プロバイダ

この章では，CJPA プロバイダの機能概要，アプリケーションの実装方法，および実行環境の設定について説明します。

8.1　この章の構成

CJPA プロバイダとは，JPA 仕様書に定められた API や内部処理を実装し，ライブラリ形式でユーザに提供するアプリケーションサーバが提供する JPA プロバイダです。

この章では，CJPA プロバイダの機能について説明します。この章の構成を次の表に示します。

表 8-1　この章の構成（CJPA プロバイダの機能）

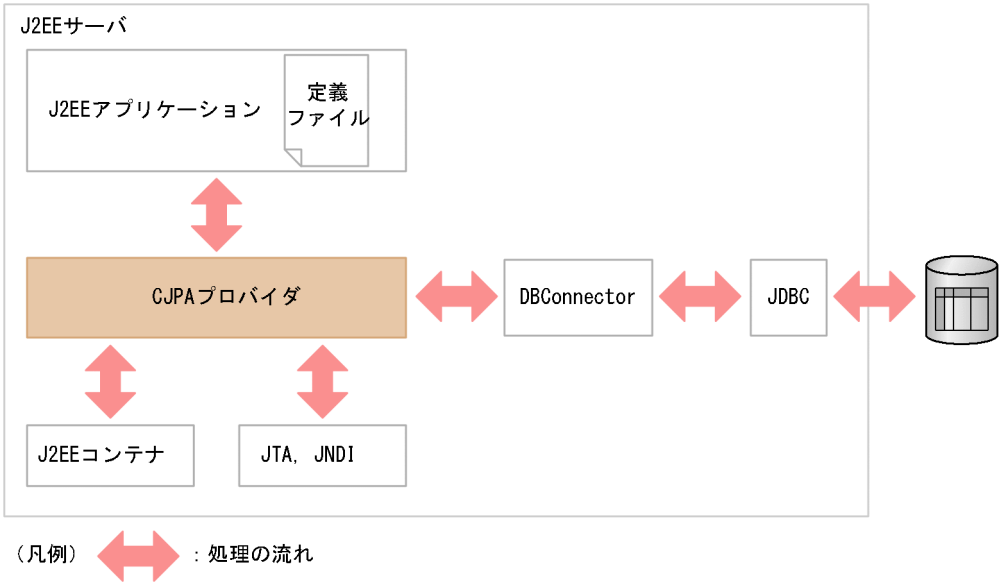
分類	タイトル	参照先
解説	CJPA プロバイダとは	8.2
	エンティティを使用したデータベースの更新	8.3
	EntityManager によるエンティティの操作	8.4
	データベースと Java オブジェクトとのマッピング情報の定義	8.5
	エンティティのリレーションシップ	8.6
	エンティティオブジェクトのキャッシュ機能	8.7
	プライマリキー値の自動採番	8.8
	クエリ言語によるデータベース操作	8.9
	楽観的ロック	8.10
	JPQL での悲観的ロック	8.11
実装	エンティティクラスの作成	8.12
	エンティティクラスの継承方法	8.13
	EntityManager および EntityManagerFactory の使用方法	8.14
	コールバックメソッドの指定方法	8.15
	クエリ言語を利用したデータベースの参照および更新方法	8.16
	JPQL の記述方法	8.17
	persistence.xml の定義	8.18
設定	実行環境での設定	8.19

注　「運用」について，この機能固有の説明はありません。

8.2　CJPA プロバイダとは

CJPA プロバイダとは，アプリケーションサーバが提供する JPA プロバイダです。CJPA プロバイダの位置づけを次の図に示します。

図 8-1　CJPA プロバイダの位置づけ



CJPA プロバイダは J2EE サーバの一機能です。JTA および JNDI を使用して，データベースを操作します。ここでは，CJPA プロバイダの機能概要について説明します。

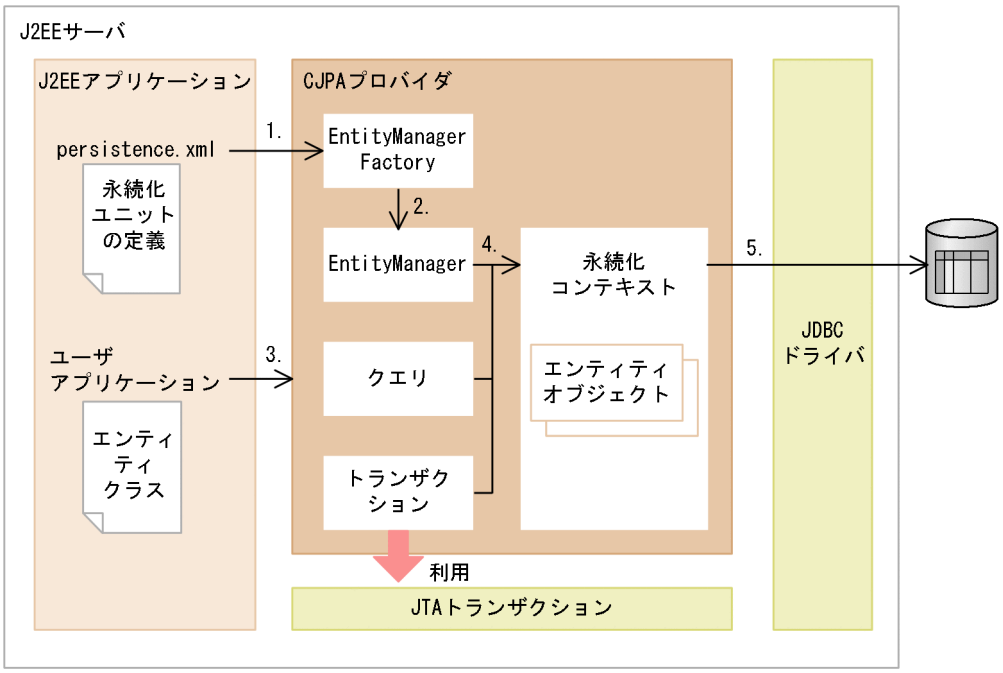
8.2.1　CJPA プロバイダでの処理

CJPA プロバイダでは，次の機能を提供しています。

- エンティティオブジェクトとデータベースのマッピング
- エンティティオブジェクトの管理
- JPQL によるデータの操作
- JDBC を通じてのデータベースへのアクセス

JPA を使用するには，ユーザが作成したアプリケーションで CJPA プロバイダが提供するインタフェースを操作します。CJPA プロバイダで実行する処理を次の図に示します。

図 8-2　CJPA プロバイダで実行される処理



図について説明します。

1. persistence.xml に定義されている永続化ユニットの情報から，CJPA プロバイダは EntityManagerFactory を生成します。
   永続化ユニットとは，マッピング対象となるクラス，O/R マッピングの情報，データソースなどを集めた論理的なグループです。永続化ユニットの詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「5.4.4　永続化ユニットとは」を参照してください。
2. 生成された EntityManagerFactory から EntityManager を生成します。
3. ユーザアプリケーションから，EntityManager，クエリ，またはトランザクションを操作します。
4. EntityManager，クエリ，トランザクションから永続化コンテキストを通して，エンティティオブジェクトを操作します。
5. エンティティオブジェクトの変更を JDBC ドライバを介してデータベースに反映します。

なお，EntityManager には，次の 2 種類があります。

- コンテナ管理 EntityManager
  J2EE コンテナによって管理される EntityManager です。
- アプリケーション管理 EntityManager
  アプリケーションによって管理される EntityManager です。

EntityManager の種類の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「5.5.1　EntityManager と永続化コンテキスト」を参照してください。

また，トランザクションではトランザクションのコミットやロールバックなどを管理します。CJPA プロバイダでは，トランザクションとしてリソースローカルトランザクションを使用できます。JTA トランザクションと連携する場合，J2EE コンテナが提供する機能を使用します。

8.2.2　CJPA プロバイダが提供する機能

CJPA プロバイダでは次に示す機能を提供しています。

表 8–2　CJPA プロバイダが提供する機能

機能	機能の概要	標準/拡張※	参照先
エンティティを使用したデータベースの更新	ユーザのアプリケーションから生成したエンティティによって，データベースのデータを更新します。	標準	8.3 8.4
データベースと Java オブジェクトとのマッピング情報の定義	Java オブジェクトとデータベースとのマッピング情報は，アノテーションまたは O/R マッピングファイルで定義できます。	標準	8.5
エンティティのリレーションシップの設定	データベースのテーブル間の関連をエンティティで設定します。	標準	8.6
エンティティオブジェクトのキャッシュ	EntityManager のエンティティオブジェクトの内容をメモリに保持する機能です。同じ内容のエンティティオブジェクトが呼ばれた場合，メモリのエンティティオブジェクトを使用します。	拡張	8.7
プライマリキーの自動採番	エンティティオブジェクトを使用してデータベースにデータを挿入するときに，CJPA プロバイダはプライマリキーを自動的に格納します。ユーザがプライマリキーを指定しなくても，一意の値が格納されます。	標準	8.8
クエリ言語によるデータベース操作	クエリ言語を使用してデータベースを操作できます。CJPA プロバイダでは，クエリ言語として JPQL または SQL を使用できます。	標準	8.9
楽観的ロック	データベースの排他方法の一つです。データベースのデータがほかのトランザクションから更新されていないかを確認してからデータを更新します。楽観的ロックではデータをロックしないため，デッドロックは発生しません。	標準	8.10
JPQL での悲観的ロック	データベースの排他方法の一つです。ほかのトランザクションから更新されないように，レコードを占有ロックします。悲観的ロックは JPQL を使用した場合だけ実行されます。	拡張	8.11

注※　JPA1.0 仕様に基づいた機能かどうかを示します。
　標準：JPA1.0 仕様に基づいた機能であることを示します。
　拡張：CJPA プロバイダ独自の機能であることを示します。

これらの機能の詳細については，表中の「参照先」に示す節を参照してください。

また，CJPA プロバイダでは，PRF トレースを取得できます。PRF トレースの取得ポイントについては「14.　性能解析トレース」を参照してください。

8.2.3　CJPA プロバイダを使用するための前提条件

ここでは，CJPA プロバイダを使用するための前提条件について説明します。

(1)　使用できるコンポーネント

CJPA プロバイダを利用するコンポーネントは，EJB の場合は EJB 3.0 以降，Web コンポーネントの場合は Servlet 2.5 以降となります。

アプリケーションサーバでは，EJB，Web アプリケーションで JPA を使用できます。また，Web アプリケーションからユーザスレッドを使用した場合も，JPA を使用できます。なお，次に示す環境またはライブラリでは JPA を使用できません。

- EJB クライアントアプリケーション環境
- J2EE アプリケーションクライアント環境
- コンテナ拡張ライブラリ

JPA を使用できるコンポーネントを次の表に示します。

表 8-3　JPA を使用できるコンポーネント

コンポーネント		JPA の使用
EJB	Stateless Session Bean（EJB3.0 以降）※1	○
	Stateful Session Bean（EJB3.0 以降）※1	○
	Stateless Session Bean（EJB3.0 より前）	×
	Stateful Session Bean（EJB3.0 より前）	×
	インターセプタ	○
	Message-driven Bean	×
	Entity Bean	×
Web アプリケーション	サーブレット，フィルタ，イベントリスナ（Servlet2.5 以降）	○
	JSP，JSP のタグハンドラ，JSP のイベントリスナ，JSP のタグライブラリイベントリスナ※2（Servlet2.5 以降）	○
	サーブレット，フィルタ，イベントリスナ（Servlet2.5 より前）	×

コンポーネント		JPA の使用
	JSP，JSP のタグハンドラ，JSP のイベントリスナ，JSP のタグライブラリイベントリスナ（Servlet2.5 より前）	×

（凡例）○：使用できる　×：使用できない

注※1　アプリケーションサーバでは EJB3.0 の ejb-jar.xml を使用した JPA の定義はできません。このため，@PersistenceUnit や@PersistenceContext などのアノテーションを使用して，永続化ユニットや永続化コンテキストの参照を定義してください。

注※2　アプリケーションサーバでは，JSP タグライブラリイベントリスナでのアノテーションの使用はできません。JSP タグライブラリイベントリスナで JPA の機能を使用する場合は，web.xml の<persistence-unit-ref>タグや<persistence-context-ref>タグを使用して永続化ユニットや永続化コンテキストの参照を定義してください。

注意事項

Servlet 2.5 以降の web.xml の metadata-complete 属性に true が設定されている場合，Web コンポーネントのアノテーションが読み込まれません。このため，アノテーションを使用して，永続化コンテキストまたは永続化ユニットの参照を定義することはできません。ただし，web.xml に参照を定義することはできます。

(2)　接続できるデータベース

CJPA プロバイダを使用する場合，次に示すデータベースに接続できます。

- Oracle
- HiRDB

(3)　使用できる DB Connector

CJPA プロバイダは，DB Connector を使用してデータベースを更新します。CJPA プロバイダでは次の表に示す DB Connector を使用できます。

表 8-4　CJPA プロバイダで使用できる DB Connector

使用するデータベース	トランザクションタイプ	使用できる DB Connector
Oracle	LocalTransaction NoTransaction	DBConnector_Oracle_CP.rar
	XATransaction	DBConnector_Oracle_XA.rar
Oracle RAC	LocalTransaction NoTransaction	DBConnector_Oracle_CP.rar DBConnector_CP_ClusterPool_Root.rar DBConnector_Oracle_CP_ClusterPool_Member.rar
HiRDB	LocalTransaction NoTransaction	DBConnector_HiRDB_Type4_CP.rar
	XATransaction	DBConnector_HiRDB_Type4_XA.rar

(4) 使用できない環境

次の環境では CJPA プロバイダを使用できません。

- バッチアプリケーションの実行環境
- EJB クライアントアプリケーションの実行環境

(5) メソッドキャンセル機能の使用について

CJPA プロバイダでは，OneToOne および ManyToOne の LAZY フェッチの場合，アクセサメソッドに独自のバイナリコードを埋め込みます。これによって，アクセサメソッドがメソッドキャンセルの対象になります。このため，OneToOne リレーションシップ，または ManyToOne リレーションシップで LAZY フェッチが指定されている場合は，エンティティクラス，埋め込み可能クラス，およびマップドスーパークラスを保護区に登録する必要があります。

なお，保護区に登録しない場合は，埋め込んだバイナリコード上でメソッドキャンセルが起こるおそれがあります。保護区に登録しない場合の動作は保証しません。

保護区に登録されている場合でも，登録されていない場合でも，メソッドタイムアウトで下記のようなスタックトレースが出力されます。ただし，保護区に登録しているとメソッドキャンセルは発生しません。

次に示す例では，ユーザが呼び出した EntityClass1.getMappingClass2 メソッドの延長で，埋め込んだ EntityClass1._toplink_getmappingClass2 が呼ばれています。そこでメソッドタイムアウトが発生していますが，メソッドキャンセルは発生しません。

```
message-id       message(LANG=ja)
KDJE52703-W      A timeout occurred while the user program was executing. (threadID = 237949
87, rootAPInfo = 10.209.11.124/5964/0x4828eb62000128e0, application = JPA_JavaEE_TP, bean =
TestBean, method = doTest)
    jpa.test.annotation.onetoone.entity.EntityClass1._toplink_getmappingClass2(EntityClass1.
java)
                        locals:
                          (jpa.test.annotation.onetoone.entity.EntityClass1) this = <0x11e358
78> (jpa.test.annotation.onetoone.entity.EntityClass1)
                         at jpa.test.annotation.onetoone.entity.EntityClass1.getMappingClass
2(EntityClass1.java:34)
                        locals:
                          (jpa.test.annotation.onetoone.entity.EntityClass1) this = <0x11e358
78> (jpa.test.annotation.onetoone.entity.EntityClass1)
```

保護区への登録については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.5　criticalList.cfg（保護区リストファイル)」を参照してください。

(6) アノテーションの参照抑止機能の使用について

CJPA プロバイダを使用する場合，アノテーションの参照抑止機能は使用できません。アノテーションの参照抑止機能が有効になっていると，永続化コンテキストおよび永続化ユニットの参照を定義できません。

8.2.4　DB Connector のコネクション数の見積もり

CJPA プロバイダを使用する場合に必要となる DB Connector のリソースの見積もりについて説明します。

CJPA プロバイダでは，DB Connector のコネクションを取得します。CJPA プロバイダが使用するコネクション数の見積もり式を次に示します。

コネクション数の見積もり式

CJPA プロバイダのアプリケーションで利用されるコネクション数

＝ JPA 機能を利用したアプリケーションの同時実行数※

注※　JPA 機能を利用したアプリケーションで，複数の永続化ユニットの EntityManager を利用する場合や，ビジネスメソッドの呼び出しで EntityManager が利用するトランザクションが異なる場合，EntityManager ごとにそれぞれコネクションが必要となります。

注意事項

ユーザが JPA 機能とは別にコネクションを取得する場合，コネクションシェアリング機能を利用することでコネクション数を削減できます。コネクションシェアリングの詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.3 コネクションシェアリング・アソシエーション」を参照してください。

8.3　エンティティを使用したデータベースの更新

CJPA プロバイダでは，エンティティを使用したデータベースの更新ができます。

JPA を使用するには，ユーザはデータベースのテーブルを Java のオブジェクトとして扱うためのエンティティクラスを作成する必要があります。エンティティクラスを使用すると，CJPA プロバイダを通してデータベースのデータを操作します。このとき，CJPA プロバイダが提供する EntityManager インタフェースを介して，データベースを操作します。

エンティティクラスを使用したデータベースの操作は次のとおりです。

1. エンティティクラスのインスタンス（エンティティオブジェクト）を生成します。
2. EntityManager インタフェースの引数に，エンティティオブジェクトを渡し，データベースを操作します。

   この操作をするためには，エンティティクラスを作成する必要があります。

EntityManager とはエンティティを操作して，状態を制御するためのオブジェクトです。エンティティの操作には，persist 操作，remove 操作，merge 操作，flush 操作，および refresh 操作があります。これらの操作の詳細については，「8.4.1　エンティティの状態遷移」を参照してください。

8.4 EntityManager によるエンティティの操作

エンティティの状態は，CJPA プロバイダで提供する EntityManager で制御されます。ここでは，EntityManager でのエンティティの操作について説明します。

8.4.1 エンティティの状態遷移

エンティティには状態があります。エンティティに対して操作するとエンティティの状態は遷移します。ここでは，エンティティの状態の種類，エンティティに対して実行する操作，およびエンティティの操作と状態遷移について説明します。

(1) エンティティの状態の種類

エンティティの状態には，new，managed，detached，removed の 4 種類があります。EntityManager のメソッドを利用してエンティティを操作することで，エンティティの状態が変更されます。

エンティティのそれぞれの状態について次の表に示します。

表 8-5 エンティティの状態

エンティティインスタンスの状態	説明
new	新しく生成された状態のエンティティです。 新しく生成されたエンティティインスタンスは，永続化アイデンティティを持ちません。このため，永続化コンテキストに関連づいていません。
managed	永続化コンテキストに関連づけられた永続化アイデンティティを持ち，永続化コンテキストで管理されている状態です。
detached	永続化コンテキストに関連づけられていない永続化アイデンティティを持つ状態です。
removed	永続化アイデンティティを持ち，永続化コンテキストに関連づけられている状態です。また，エンティティインスタンスがデータベースから削除されることが予定されている状態でもあります。

(2) エンティティに対する操作

ユーザがデータベースのレコードを検索すると，CJPA プロバイダは取得したデータをエンティティのフィールドに保持します。また，ユーザがデータベースの内容を更新する場合は，永続化コンテキストに登録されているエンティティの状態を変更して，トランザクションをコミットすることでデータベースにエンティティの状態を反映します。このように，エンティティに対して操作することで，データベースの情報を更新します。

エンティティに対する操作の種類を次の表に示します。

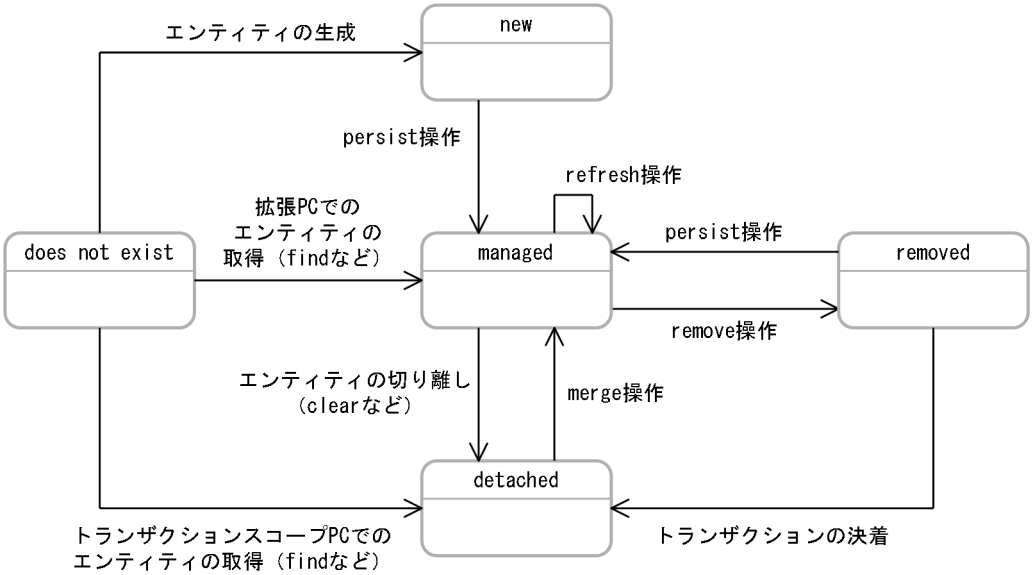
表 8-6　エンティティに対する操作の種類

操作	説明
flush	エンティティオブジェクトの内容をデータベースに反映します。
merge	EntityManager の管理対象外となっているエンティティオブジェクトを EntityManager の管理対象とします。
persist	エンティティオブジェクトを管理して，永続化します。
refresh	エンティティオブジェクトにデータベースの内容を反映します。
remove	エンティティオブジェクトを削除予定状態にします。

(3)　エンティティに対する操作と状態遷移

エンティティインスタンスに対する操作と状態遷移について次の図で説明します。

図 8-3　エンティティインスタンスに対する操作と状態遷移



エンティティに対する操作と状態遷移について次の表にまとめます。

表 8-7　エンティティの状態遷移

操作	状態			
	new	managed	detached	removed
persist	managed※1	managed	managed※1	managed
remove	new	removed	例外※2，※3	Removed
merge	• コピー元 new • コピー先 managed	• コピー元 managed • コピー先 managed	• コピー元 detached • コピー先 managed	例外※3，※4

操作	状態			
	new	managed	detached	removed
refresh	例外※2，※4	managed※5	例外※2，※4	例外※2，※4
commit	−	※6	−	detached
rollback	−	detached	−	detached
flush	−	managed	−	detached
clear	−	detached	−	detached

（凡例）−：該当しない

注※1　persist 操作で例外が発生した場合，状態は遷移しないで元の状態になります。

注※2　発生する例外は IllegalArgumentException です。

注※3　データベース上に対応する行が存在しない場合，操作は無視されます。

注※4　例外時には状態は遷移しないで，元のままとなります。

注※5　データベース上に対応する行が存在しない場合，EntityNotFoundException が発生します。

注※6　トランザクションスコープの永続化コンテキストの場合は detached になります。拡張された永続化コンテキストの場合は managed になります。

また，persist，remove，merge，refresh をトランザクション外で実行した場合の動作は，永続化コンテキストの種類によって異なります。

- トランザクションスコープの永続化コンテキストの場合
  TransactionRequiredException となります。
- 拡張された永続化コンテキストの場合
  状態が遷移して，次のトランザクション決着のタイミングでデータベースに状態が反映されます。

(4)　エンティティに対する操作の伝播

エンティティがリレーションシップを持つ場合に，リレーションシップを表すアノテーションの cascade 属性を指定すると，エンティティに対する操作が関連するエンティティに対して伝播されます。cascade 属性には，次の表に示す値を指定できます。

表 8-8　cascade 属性の種類

cascade 属性の種類	説明
CascadeType.ALL	persist，remove，merge，refresh の操作が関連先に伝播されます。
CascadeType.PERSIST	persist 操作が関連先に伝播されます。
CascadeType.REMOVE	remove 操作が関連先に伝播されます。
CascadeType.MERGE	merge 操作が関連先に伝播されます。
CascadeType.REFRESH	refresh 操作が関連先に伝播されます。

8.4.2　エンティティに対する persist 操作

エンティティに対する persist 操作を実行するには，EntityManager の persist メソッドを呼び出します。EntityManager の persist メソッドを呼び出したり，persist 処理がカスケードされたりすると，エンティティはデータベースへの永続化や永続化コンテキストでの管理対象になります。

次の表に，persist 操作後のエンティティの状態をエンティティの状態ごとに示します。

表 8–9　Persist 操作でのエンティティの状態

エンティティの状態	persist 操作後のエンティティの状態
new※	managed に遷移します。managed に遷移したエンティティはトランザクションのコミット時またはコミット前にデータベースに追加されます。または，flush 操作の実行の結果としてデータベースに追加されます。
managed	persist 操作は無視されます。ただし，エンティティからほかのエンティティへのリレーションシップの cascade 属性に PERSIST または ALL が指定されていると，このエンティティが参照するエンティティに persist 操作が伝播されます。
detached※	データベースにエンティティと対応する行が存在しない場合は，managed に遷移します。対応する行が存在する場合，EntityExistsException が発生します。
removed	managed に遷移します。

注※　CJPA プロバイダの場合，エンティティの状態が new または detached の場合，データベース上にエンティティに対応するデータが存在するかどうかでエンティティの状態遷移の結果が異なります。CJPA プロバイダの場合，次のことに注意してください。

- データベースに永続化されていないエンティティを引数に指定した場合は，managed に遷移します。
- データベースに引数で渡されたエンティティと重複する行が存在し，そのエンティティが永続化コンテキストで管理されている場合，persist 処理時に EntityExistsException が発生します。
- データベースに引数で渡されたエンティティと重複する行が存在するが，永続化コンテキストでエンティティが管理されていない場合，flush やコミット時に EntityExistsException またはほかの PersistenceException が発生します。

注意事項

エンティティをデータベースに永続化したり，データベースからエンティティの情報を読み込んだりするタイミングで，エンティティは永続化コンテキストに登録されます。エンティティを managed に遷移したあとに，永続化処理をロールバックした場合は，エンティティは永続化コンテキストで管理されていないので注意してください。

8.4.3　エンティティに対する remove 操作

エンティティに対する remove 操作を実行するには，EntityManager の remove メソッドを呼び出します。EntityManager の remove メソッドを呼び出したり，remove 処理がカスケードされたりすると，エ

ンティティは removed 状態になります。removed 状態のエンティティはトランザクションのコミット処理でデータベースから削除されます。

次の表に，remove 操作後のエンティティの状態をエンティティの状態ごとに示します。

表 8–10　remove 操作でのエンティティの状態

エンティティの状態	状態遷移の結果
new	remove 操作は無視されます。ただし，エンティティからほかのエンティティへのリレーションシップの cascade 属性に REMOVE または ALL が指定されていると，このエンティティが参照するエンティティに remove 操作が伝播されます。
managed	removed に遷移します。エンティティからほかのエンティティへのリレーションシップの cascade 属性に REMOVE または ALL が指定されていると，このエンティティが参照するエンティティに remove 操作が伝播されます。
detached	次のように遷移します。 • 引数で渡された detached 状態のエンティティに対応する行がデータベースに存在する場合，remove 操作時に IllegalArgumentException がスローされます。 • CJPA プロバイダの場合，データベースに対応する行が存在しないときには，remove 操作は無視されます。ただし，ほかのエンティティへのリレーションシップの cascade 属性に REMOVE または ALL が指定されていれると，このエンティティが参照するエンティティに remove 操作が伝播されます。
removed	remove 操作は無視されます。ほかのエンティティへの伝播もしません。

注 1　removed 状態のエンティティはトランザクションのコミット時，トランザクションのコミット前，または flush 操作時の実行結果としてデータベースから削除されます。

注 2　エンティティが削除されたあと，エンティティの内容は remove 処理が呼び出された直後の内容になります。ただし，エンティティの生成直後の場合を除きます。

8.4.4　データベースからのエンティティの取得

EntityManager の find メソッドまたは getReference メソッドの呼び出しによって，引数で指定した主キーに対応するエンティティをデータベースから取得できます。

EntityManager の find メソッドまたは getReference メソッドで返されるエンティティは，トランザクションスコープの永続化コンテキストを利用します。このため，トランザクション内でメソッドを呼び出した場合は managed 状態になります。また，トランザクション外で find メソッドまたは getReference メソッドを呼び出した場合は detached 状態になります。

find 操作または getReference 操作で取得したエンティティの状態を次の表に示します。

表 8–11　find 操作または getReference 操作で取得したエンティティの状態

永続化コンテキスト	取得したエンティティの状態
トランザクションスコープの永続化コンテキスト（トランザクション外）	detached

永続化コンテキスト	取得したエンティティの状態
トランザクションスコープの永続化コンテキスト（トランザクション内）	managed
拡張された永続化コンテキスト（トランザクション外）	managed
拡張された永続化コンテキスト（トランザクション内）	managed

なお，CJPA プロバイダの場合，find 操作または getReference 操作では次の点が JPA 仕様と異なります。なお，このほかに JPA 仕様との機能差はありません。

- JPA 仕様では，EntityManager の getReference メソッドでは実際のインスタンスではなく，引数で与えられた主キーに対応するエンティティに Proxy を返すことができます。ただし，CJPA プロバイダでは，@Basic に対する Lazy ローディングをサポートしていません。このため，CJPA プロバイダでは，getReference の戻り値として，Proxy ではなくエンティティインスタンスを返します。

  なお，リレーションシップに対する Lazy ローディングのサポート範囲については，「8.4.5(2)　データベースからのエンティティ情報の読み込み」を参照してください。
- CJPA プロバイダでは，find メソッドと getReference メソッドの間で存在しないエンティティを引数に指定した場合，find メソッドでは null 値が返ります。また，getReference メソッドでは EntityNotFoundException が発生します。

8.4.5　データベースとの同期

トランザクションのコミットまたは flush メソッドの実行時に，エンティティの状態がデータベースに書き込まれます。一方，明示的に refresh を呼び出さないかぎり，メモリにロードされたエンティティの状態のリフレッシュは実行されません。

ここでは，データベースへのエンティティ情報の書き込みと，データベースからのエンティティ情報の読み込みについて説明します。

(1)　データベースへのエンティティ情報の書き込み

flush 操作またはトランザクションのコミットでのエンティティの状態遷移について説明します。次の表ではエンティティ A の状態ごとに状態遷移の結果を示しています。

表 8-12　flush 操作またはトランザクションのコミットでのエンティティインスタンスの状態遷移

エンティティ A の状態	状態遷移の結果
new	flush 操作は無視されます。
managed	データベースと同期されます。
removed	エンティティ A はデータベースから削除されます。
detached	flush 操作は無視されます。

また，managed 状態のエンティティ A がエンティティ B に対してリレーションシップを持っている場合，flush 処理の延長で次の表に示す条件に従って，persist 操作がカスケードされます。

表 8-13　関連するエンティティ B への flush 処理およびコミットでの persist 操作のカスケード

エンティティ B へのリレーションシップの cascade 属性の指定	エンティティ B の状態	結果
PERSIST または ALL が指定されている	－	persist 操作がエンティティ B にカスケードされます。
PERSIST または ALL が指定されていない	new	• flush 操作の場合 IllegalStateException が発生してトランザクションはロールバックにマークされます。 • コミット処理の場合 IllegalStateException が発生してトランザクションのコミットに失敗します。
	managed	データベースと同期化されます。
	removed	• flush 操作の場合 IllegalStateException が発生してトランザクションはロールバックにマークされます。 • コミット処理の場合 IllegalStateException が発生してトランザクションのコミットに失敗します。
	detached	• エンティティ A がリレーションシップの所有者の場合 リレーションシップの変更はデータベースと同期されます。 • エンティティ B がリレーションシップの所有者の場合 例外が発生します。CJPA プロバイダでは，この場合の動作はサポート対象外となります。

（凡例）－：該当しない

なお，トランザクションの外で flush メソッドを呼び出すと，TransactionRequiredException が発生します。

ポイント

リレーションシップとデータベースに対する永続化の関連

- 双方向のリレーションシップを持つ managed 状態のエンティティは，リレーションシップの所有者側で保持される参照を基に永続化されます。アプリケーション開発者は，変更が発生したときに所有者側と被所有者側でそれぞれ，メモリ上の状態に矛盾がないようにエンティティを保持するようアプリケーションを作成してください。
- 単方向の OneToOne，OneToMany の場合，エンティティで定義しているリレーションシップの関係とデータベースのテーブル間の関係が合っていることは，開発者の責任で保証してください。

(2) データベースからのエンティティ情報の読み込み

EntityManager の refresh メソッドを呼び出すと，それまでに行われたエンティティの変更は破棄され，データベースの内容でエンティティの状態を上書きします。このとき，データベース上に対応する行が存在しない場合は EntityNotFoundException が発生します。

エンティティが managed 状態以外の場合に，EntityManager の refresh メソッドを呼び出すと IllegalArgumentException が発生します。

(a) データベースからのエンティティ情報を読み込むタイミング

refresh メソッドもしくは find メソッドの実行時，またはクエリの発行時に，データベースからエンティティが読み込まれます。このときに，関連するエンティティもあわせて読み込むことができます。これをフェッチ戦略といいます。フェッチ戦略は各リレーションシップの fetch 属性で指定します。fetch 属性には次の 2 種類のどちらかを指定します。

- FetchType.EAGER
  データベースからエンティティの情報を読み込むときに，関連するフィールドやエンティティの情報を読み込みます。
- FetchType.LAZY
  フィールドまたは関連先に初めてアクセスしたときにデータベースからの読み込みが実行されます。これを，Lazy ローディングといいます。

FetchType.EAGER を指定するとデータベースからエンティティの情報を読み込むたびに，関連するフィールドやエンティティの情報を読み込みます。このため，不要な関連先のエンティティの取得を回避したい場合は，FetchType.LAZY を指定してください。

CJPA プロバイダでの fetch 属性のサポート範囲を次の表に示します。

表 8-14　リレーションシップごとの fetch 属性のサポート範囲

リレーションシップのアノテーション	サポート範囲
@ManyToMany	デフォルトは FetchType.LAZY です。
@OneToMany	デフォルトは FetchType.LAZY です。
@OneToOne	デフォルトは FetchType.EAGER です。なお，LAZY を指定したときには，(b)を参照してください。
@ManyToOne	デフォルトでは FetchType.EAGER です。なお，LAZY を指定したときには，(b)を参照してください。
@Basic	fetch 属性は無視されます。デフォルトの FetchType.EAGER が常に適用されます。

(b) @OneToOne および@ManyToOne での LAZY フェッチ

@OneToOne および@ManyToOne のリレーションシップに対してフェッチ戦略に LAZY を指定した場合，エンティティクラスのローディング時に，LAZY を指定したフィールドの getter メソッドにバイナリコードを埋め込みます。

getter メソッドが呼び出されると，埋め込んだバイナリコードの処理を通して，データベースから関連先のエンティティを取得します。getter メソッドがバイナリの埋め込み対象になるため，リレーションの対象となるフィールドにアクセスするための getter メソッドを設定しておく必要があります。また，その getter メソッドに@OneToOne または@ManyToOne を設定しておく必要があります。

注意事項

getter メソッドでは，リレーションシップの targetEntity の型から，関連先のエンティティの型を決定します。targetEntity に指定するクラスの型は，フィールド・プロパティの型にキャストできる必要があります。

8.4.6 永続化コンテキストからのエンティティの切り離しと merge 操作

永続化コンテキストから切り離されたエンティティを detached 状態のエンティティといいます。エンティティは次のタイミングで detached 状態になります。

- トランザクションスコープの永続化コンテキストでのトランザクションをコミットしたとき
- トランザクションのロールバックを実行したとき
- 永続化コンテキストをクリアしたとき（EntityManager.clear()を呼び出したとき）
- EntityManager をクローズしたとき
- エンティティのシリアライズおよびエンティティの値渡しをしたとき

切り離されたエンティティのインスタンスは，永続化や取得をした永続化コンテキストの外で存在し続けます。エンティティの状態とデータベースの状態は同期されません。

(1) detached 状態のエンティティへのアクセス

アプリケーションは永続化コンテキストの終了後でも，detached 状態のエンティティにアクセスできます。この場合，エンティティのフィールドおよび関連先は，フェッチ済みである必要があります。detached 状態のエンティティでフェッチされていないフィールドおよび関連先のエンティティにはアクセスできません。なお，フィールドに指定する@Basic には常に FetchType.EAGER が適用されるため，関連先のエンティティやフィールドの情報が取得済みとなります。

リレーションシップでは detached 状態のエンティティからアクセスするには，次に示すどれかの条件を満たす必要があります。

- find()で取得されたエンティティインスタンスであること
- クエリを使って取得されたエンティティまたは FETCH JOIN 節で明示的に要求されたエンティティであること
- プライマリキーでない永続状態のインスタンスをアプリケーションでアクセス済みであること
- fetch=EAGER と指定された関連をたどって別の有効なエンティティから取得済みのエンティティであること

利用できないインスタンスへのアクセス，および利用できるインスタンスの無効なステートへのアクセスを行った場合は，例外が発生します。ただし，CJPA プロバイダでは，FetchType.LAZY を指定した場合，EntityManager の終了後に detached 状態になっていても，フェッチされていないフィールドおよび関連先エンティティにアクセスすると，データベースから値を取得して，内容を参照することができる場合があります。

(2) エンティティの merge 処理

EntityManager の merge メソッドの呼び出し，または merge 処理のカスケードによって，detached 状態のエンティティを EntityManager で管理される永続化コンテキストにマージできます。

次の表に merge 処理でのエンティティの状態遷移を，エンティティ A の状態ごとに示します。

表 8–15　merge 処理でのエンティティの状態遷移の結果

エンティティ A の状態	状態遷移の結果
new	managed 状態のエンティティ A'が新しく作成されて，エンティティ A の状態がエンティティ A'にコピーされます。merge の引数のエンティティは new のままであることに注意してください。
managed	merge 操作は無視されます。しかし，エンティティ A からほかのエンティティへのリレーションシップの cascade 属性に，MERGE または ALL が指定されている場合，エンティティ A が参照するエンティティに merge 操作が伝播されます。
detached	エンティティ A の状態が，同じ ID を持つすでに存在している管理されたエンティティ A'にコピーされます。または，エンティティ A のコピーである新しい managed 状態のエンティティが作成されます。merge の引数のエンティティは detached 状態のままであることに注意してください。
removed	merge 操作によって，IllegalArgumentException が発生します。または，トランザクションのコミットに失敗します。エンティティ A の状態は removed 状態のままであることに注意してください。

注 1　エンティティ A からエンティティ B へのリレーションが cascade=MERGE または cascade=ALL で指定されていると，すべてのエンティティ B は再帰的にエンティティ B'としてマージされます。エンティティ A'にはエンティティ B'がセットされます。なお，エンティティ A が managed 状態の場合，エンティティ A とエンティティ A'は同じであることに注意してください。

注 2　エンティティ A のリレーションに cascade=MERGE または cascade=ALL が指定されていない状態でエンティティ B を参照している場合，エンティティ A がエンティティ A'にマージされるときには，エンティティ A'から関連をたどるとエンティティ B と同じ永続化アイデンティティを持つ管理されたエンティティ B'の参照にたどり着きます。

JPA 仕様では，フェッチされていないことを意味する LAZY にマークされたフィールドはマージのときに無視されます。ただし，CJPA プロバイダでは，@Basic は EAGER として動作するため，リレーションシップでないすべてのフィールドはマージ処理の対象となります。

また，Version 列がエンティティで使われている場合は，マージ操作とそのあとに呼ばれるフラッシュおよびコミット処理時に，エンティティのバージョンチェックを実行します。Version 列が存在しない場合，merge 操作ではエンティティのバージョンチェックは実行されません。詳細については，「8.10.1　楽観的ロックの処理」を参照してください。

なお，CJPA プロバイダでは，ほかベンダとの間で，エンティティのマージ処理によって永続化コンテキストに戻すという処理はサポートしていません。

(3)　注意事項

- CJPA プロバイダの場合，トランザクションスコープの永続化コンテキストでは，managed 状態のエンティティはトランザクションのコミットによって detached 状態になります。一方，拡張された永続化コンテキストでは，managed 状態のエンティティは管理されたままとなります。
- トランザクションスコープでも拡張された永続化コンテキストでも，トランザクションをロールバックすると，すべての存在している managed 状態のインスタンスと removed 状態のインスタンスは，detached 状態になります。インスタンスの状態は，トランザクションがロールバックされた時点の状態です。
- トランザクションをロールバックすると，永続化コンテキストの状態はロールバックした時点の状態になるため，データベースの状態と矛盾します。なお，CJPA プロバイダの場合は，バージョン属性の状態と生成された状態は矛盾した状態になるため，merge 操作などを行うと例外になることがあります。

8.4.7　managed 状態のエンティティ

EntityManager の contains()メソッドはエンティティのインスタンスがカレントの永続化コンテキストで管理されているかを取得するために利用できます。

ここでは，contains()メソッドの戻り値の条件について説明します。

- contains()メソッドが true を返す条件
  - エンティティがデータベースから取得されていて，EntityManager から削除されていない場合，または切り離されていない場合
  - エンティティのインスタンスが生成され，persist メソッドがそのエンティティに対して実行されている場合，または persist 操作がそのエンティティに伝播されている場合
- contains()メソッドが false を返す条件
  - エンティティのインスタンスが切り離されている場合
  - remove メソッドがエンティティに対して実行されている場合，または remove 操作がエンティティに伝播されている場合
  - エンティティのインスタンスが生成され，persist メソッドがそのエンティティに対して実行されていない場合，または persist 操作がそのエンティティに伝播されていない場合

実際のデータベースでの insert および delete 処理がトランザクションの決着まで遅延されます。これに対して，persist や remove の伝播は contains メソッドではすぐに反映されることに注意してください。

なお，エンティティのインスタンスが単一の永続化コンテキストでだけ管理されていることは，アプリケーション側で保証してください。CJPA プロバイダでは，同じ Java のインスタンスを複数の永続化コンテキストで管理した場合の動作は保証しません。

8.5 データベースと Java オブジェクトとのマッピング情報の定義

CJPA プロバイダでは，データベースと Java オブジェクトをマッピングするための情報を定義できます。マッピング情報はアノテーションまたは O/R マッピングファイルで定義します。

- **アノテーションを使用した定義**
  アプリケーションのエンティティクラスに直接マッピング情報を定義します。
- **O/R マッピングファイルを使用した定義**
  O/R マッピングファイルとは，マッピング情報を記載するための XML 形式のファイルです。タグを使用してマッピング情報を定義します。

アノテーションと O/R マッピングファイルの両方でマッピング情報を定義している場合，O/R マッピングファイルの定義が優先されます。このため，アノテーションを使用して定義したマッピング情報を変更する場合に O/R マッピングファイルを使用すると，アプリケーションを変更することなくマッピング情報を変更できます。

アプリケーションの作成方針に合わせて，アノテーションを使用するか，O/R マッピングファイルを使用するか，またはアノテーションと O/R マッピングファイルを併用するかを決定してください。

8.6　エンティティのリレーションシップ

エンティティでは，データベースのテーブル間の関連をリレーションシップで表現できます。CJPA プロバイダでは，エンティティのリレーションシップを設定できます。

8.6.1　リレーションシップの種類

リレーションシップには関連と方向があります。関連には，OneToOne，ManyToOne，OneToMany，および ManyToMany があります。また，それぞれの関連の方向には，単方向と双方向があります。関連と方向を組み合わせると，リレーションシップの種類には次の 7 種類があります。

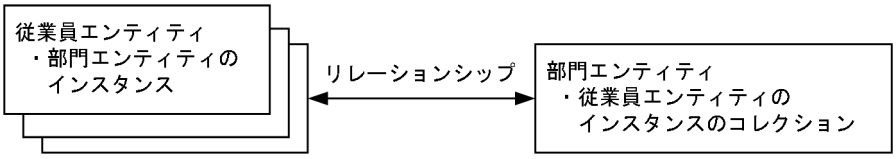
- 単方向の OneToOne リレーションシップ
- 単方向の ManyToOne リレーションシップ
- 単方向の OneToMany リレーションシップ
- 単方向の ManyToMany リレーションシップ
- 双方向の OneToOne リレーションシップ
- 双方向の ManyToOne/OneToMany リレーションシップ
- 双方向の ManyToMany リレーションシップ

ある会社の従業員と部門を例にリレーションシップについて説明します。従業員と部門は次のような関連があります。

- エンティティでは，従業員と部門をそれぞれ表現します。
- 従業員は必ずどこかの部門に所属するものとします。
- 部門では複数の従業員を保持します。
- 従業員から部門を参照したり，部門から従業員を参照したりします。

この例の場合，エンティティの双方向の ManyToOne/OneToMany リレーションシップで表現します。この場合，Many が従業員，One が部門となります。従業員のエンティティと部門のエンティティの関連を次の図に示します。

図 8-4　従業員のエンティティと部門のエンティティの関連



なお，エンティティには，関連先のエンティティに操作を伝播させる設定ができます。この設定をカスケードといいます。カスケードが設定されていると，エンティティに対して操作した場合，操作したエンティ

ティとリレーションシップがあるエンティティにも自動的に同様の操作が実行されます。カスケードを利用すると，ユーザは関連先のエンティティへの操作の手間を省くことができます。

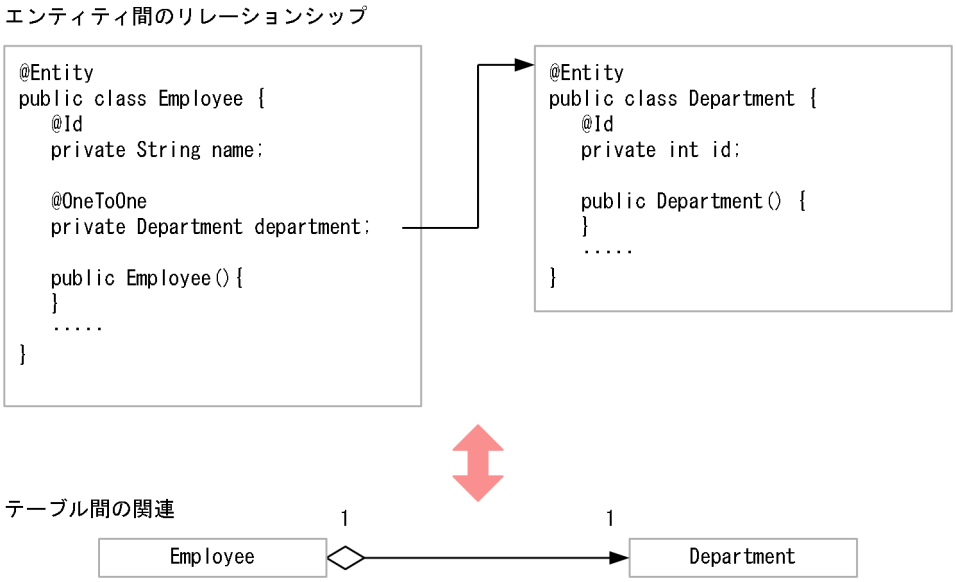
8.6.2　リレーションシップのアノテーション

エンティティではテーブル間の関連をリレーションシップとして扱います。リレーションシップをエンティティで扱う場合，エンティティでフィールドとしてほかのエンティティへの参照を保持します。その上で，次に示すリレーションシップのアノテーションを，エンティティを参照する永続化プロパティまたはインスタンス変数に設定してください。

- OneToOne（1 対 1）
- OneToMany（1 対多）
- ManyToOne（多対 1）
- ManyToMany（多対多）

エンティティのリレーションシップとテーブル間の関連を次の図に示します。

図 8-5　エンティティのリレーションシップとテーブル間の関連



エンティティへの参照に対して，リレーションシップのアノテーションが指定されていない場合は動作を保証しません。なお，コレクションを使用した参照で，generic 型を利用しない場合，リレーションシップの target として，対象のエンティティを指定する必要があります。

なお，アノテーションを使用する代わりに，O/R マッピングファイルでリレーションシップを定義することもできます。ただし，O/R マッピングファイルで定義している場合で，同じ定義をアノテーションに指定しているときには，アノテーションの定義は上書きされるので注意してください。

リレーションシップのアノテーションでのデフォルトマッピング

OneToOne，OneToMany，ManyToOne，ManyToMany のリレーションシップのアノテーションを適用した場合，デフォルトマッピングの規則が適用されます。O/R マッピングファイルでリレーションシップを指定した場合も同様に，デフォルトマッピングが適用されます。

リレーションシップのアノテーションのデフォルトマッピングを使用する場合，データベースのテーブルおよびリレーションが，「8.6.4　デフォルトマッピング（双方向のリレーションシップ）」または「8.6.5　デフォルトマッピング（単方向のリレーションシップ）」にあるデフォルトと異なると，実行時にデータベース問い合わせで SQL 文が正しく作成できません。例外が発生します。

8.6.3　リレーションシップの方向

リレーションシップには，双方向のリレーションシップと単方向のリレーションシップがあります。双方向のリレーションシップを扱う場合，リレーションシップには所有者と被所有者があります。単方向のリレーションシップには所有者だけがあります。リレーションシップの所有者は，データベースのリレーションシップの更新を決定できます。

双方向のリレーションシップでは，次のルールが適用されます。

- 双方向のリレーションシップの被所有者は，@OneToOne，@OneToMany，@ManyToMany の mappedBy 要素によって，所有者を指定します。mappedBy 要素では，所有者のエンティティで，被所有者側を参照しているプロパティまたはフィールドの名前を指定します。
- ManyToOne/OneToMany 双方向のリレーションシップでは，many 側を所有者にしてください。そのため，mappedBy 要素は@ManyToOne では指定できません。
- OneToOne の双方向のリレーションシップでは，外部キーを含む側のエンティティが所有者になります。
- ManyToMany の双方向のリレーションシップは，どちらが所有者でもかまいません。

リレーションシップのアノテーションには cascade 属性があります。cascade 属性を指定すると，エンティティに対する操作を参照先のエンティティに対しても伝播させることができます。ただし，リレーションシップのアノテーションの cascade 属性に REMOVE を指定できるのは，OneToOne または OneToMany のときだけです。ほかのリレーションに対して cascade=REMOVE を適用した場合の動作は保証しません。エンティティがリレーションシップを持つ場合の cascade 属性については，「8.4.1(4) エンティティに対する操作の伝播」を参照してください。

注意事項

CJPA プロバイダでは，実行時のリレーションシップの一貫性を保つためのチェックは実施しません。このため，アプリケーションの実行時にリレーションシップを更新する場合に，リレーションシップに矛盾が発生するような更新を行っても，警告や例外は発生しません。

なお，OneToMany，ManyToMany などのコレクションのリレーションシップでは，データベースから値をフェッチした場合に，関連するエンティティが存在しないときには，リレーションシップの値として空のコレクションを返します。

8.6.4 デフォルトマッピング（双方向のリレーションシップ）

ここでは，双方向のリレーションシップのデフォルトマッピングについて説明します。

(1) 双方向の OneToOne リレーションシップ

次に示す条件の場合に適用される双方向の OneToOne リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A はエンティティ B の単体のインスタンスを参照して，@OneToOne を設定します。
- エンティティ B はエンティティ A の単体のインスタンスを参照して，@OneToOne を設定します。@OneToOne の mappedBy 属性にエンティティ A でエンティティ B を参照する永続化プロパティ（またはフィールド）名を指定します。
- エンティティ A は，リレーションシップの所有者です。

適用されるデフォルトマッピング

- エンティティ A は，テーブル A にマップされます。
- エンティティ B は，テーブル B にマップされます。
- テーブル A は，テーブル B の外部キーを持つ必要があります。なお，外部キー列の名前は，次のようになります。

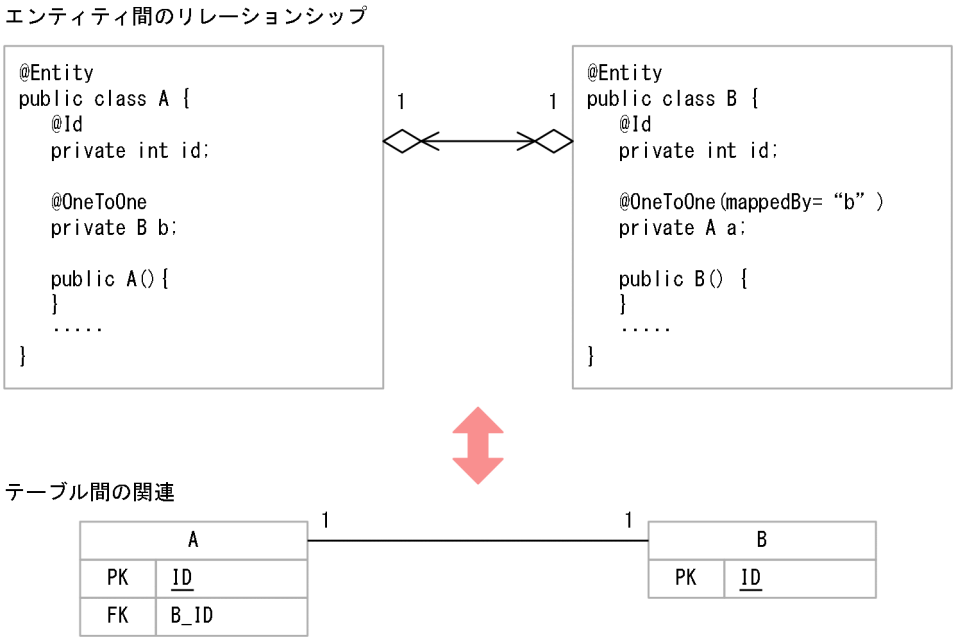
  外部キー列の名前

  エンティティ A のリレーションシッププロパティ（またはフィールド）の名前_テーブル B のプライマリキー列の名前

  注　斜体は可変値を表します。

また，外部キー列は，テーブル B のプライマリキーと同じ型を持ち，ユニークキー制約があります。

図 8-6　双方向の OneToOne でのデフォルトマッピング



(2)　双方向の ManyToOne/OneToMany リレーションシップ

次に示す条件の場合に適用される双方向の ManyToOne/OneToMany リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A はエンティティ B の単体のインスタンスを参照して，@ManyToOne（または，O/R マッピングファイルの該当する XML タグ）を設定します。
- エンティティ B は，エンティティ A のコレクションを参照して，@OneToMany（または，O/R マッピングファイルの該当する XML タグ）を設定します。@OneToMany には mappedBy 属性を指定します。mappedBy 属性は，エンティティ A で，エンティティ B を参照するために設定した永続プロパティ（またはフィールド）名を指定します。
- エンティティ A は，リレーションシップの所有者です。

適用されるデフォルトマッピング

- エンティティ A はテーブル A にマップされます。
- エンティティ B はテーブル B にマップされます。
- テーブル A は，テーブル B に対する外部キーを持つ必要があります。外部キー列の名前は，次のようになります。

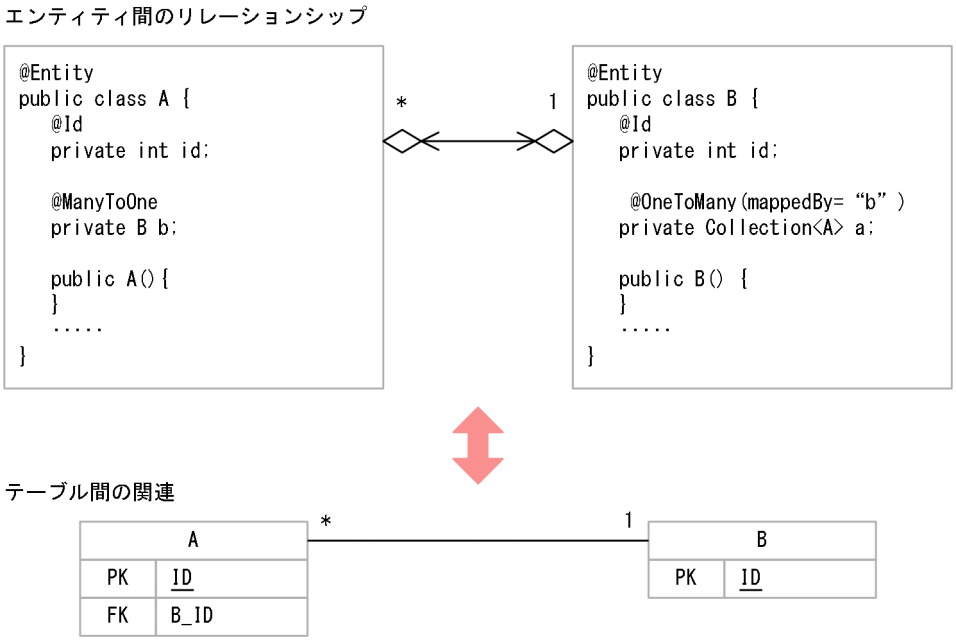
  外部キー列の名前

  エンティティ A のリレーションシッププロパティ（またはフィールド）の名前_テーブル B のプライマリキー列の名前

  注　斜体は可変値を表します。

外部キー列はテーブル B のプライマリキーと同じ型を持ちます。

図 8-7　双方向の ManyToOne/OneToMany でのデフォルトマッピング



(3)　双方向の ManyToMany リレーションシップ

次に示す条件の場合に適用される双方向の ManyToMany リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A は，エンティティ B のコレクションを参照します。コレクションには，@ManyToMany（または，O/R マッピングファイルで該当する XML 要素）を設定します。
- エンティティ B は，エンティティ A のコレクションを参照します。コレクションには，@ManyToMany（または，O/R マッピングファイルで該当する XML タグ）を設定して，mappedBy 属性を指定します。mappedBy 属性には，エンティティ A で，エンティティ B を参照するために設定した永続化プロパティ（またはフィールド）名を指定します。
- エンティティ A は，リレーションシップの所有者側です。

適用されるデフォルトマッピング

- エンティティ A は，テーブル A にマップされます。
- エンティティ B は，テーブル B にマップされます。
- テーブル A，B のほかに，所有者側のテーブルの名前が最初にくる A_B という名前の結合表が必要です。この結合表は，二つの外部キー列を持ちます。

  一つ目の外部キー列はテーブル A を参照し，テーブル A のプライマリキーと同じ型を持ちます。この外部キー列の名前は，次のようになります。

  外部キー列の名前

  エンティティ B のリレーションシップのプロパティ（またはフィールド）の名前_テーブル A のプライマリキーの名前

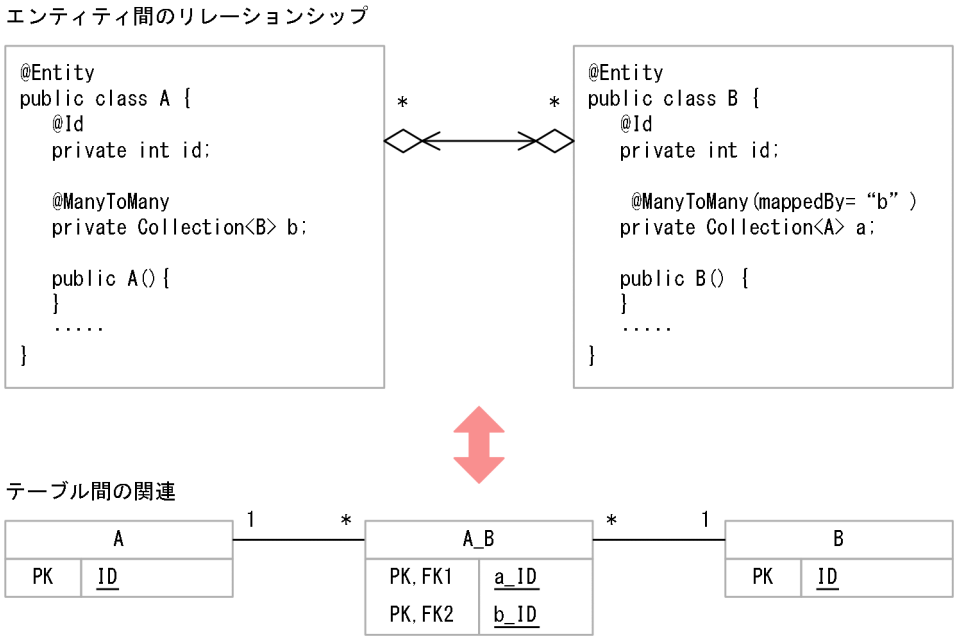
また，もう一つの外部キー列は，テーブル B を参照し，テーブル B のプライマリキーと同じ型を持ちます。この外部キー列の名前は，次のようになります。

外部キー列の名前

エンティティ A のリレーションシップのプロパティ（またはフィールド）の名前_テーブル B のプライマリキーの名前

注　斜体は可変値を表します。

図 8-8　双方向 ManyToMany でのデフォルトマッピング



8.6.5　デフォルトマッピング（単方向のリレーションシップ）

単方向のリレーションシップには，Single-Valued リレーションシップと Multi-Valued リレーションシップがあります。それぞれのリレーションシップについて説明します。

- **単方向の Single-Valued リレーションシップ**

  単方向の Single-Valued リレーションシップとは，単体のインスタンスを参照し，所有者だけが存在するリレーションシップのことです。

  単方向 Single-Valued リレーションシップには，単方向の OneToOne リレーションシップと単方向の ManyToOne リレーションシップがあります。

- **単方向の Multi-Valued リレーションシップ**

  単方向の Multi-Valued リレーションシップとは，コレクションの形でエンティティを参照し，所有者だけが存在するリレーションシップのことです。

  単方向 Multi-Valued リレーションシップには，単方向の OneToMany リレーションシップと単方向の ManyToMany リレーションシップがあります。

ここでは，単方向のリレーションシップのデフォルトマッピングについて説明します。

(1) 単方向の Single-Valued リレーションシップ

単方向の OneToOne リレーションシップと単方向の ManyToOne リレーションシップのデフォルトマッピングについて説明します。

(a) 単方向の OneToOne リレーションシップ

次に示す条件の場合に適用される単方向の OneToOne リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A は，エンティティ B の単体のインスタンスを参照して，@OneToOne（または，O/R マッピングファイルで該当する XML タグ）を設定します。
- エンティティ B からは，エンティティ A を参照しません。
- エンティティ A がリレーションシップの所有者となります。

適用されるデフォルトマッピング

- エンティティ A は，テーブル A にマップされます。
- エンティティ B は，テーブル B にマップされます。
- テーブル A は，テーブル B に対する外部キーを持つ必要があります。外部キー列の名前は，次のようになります。

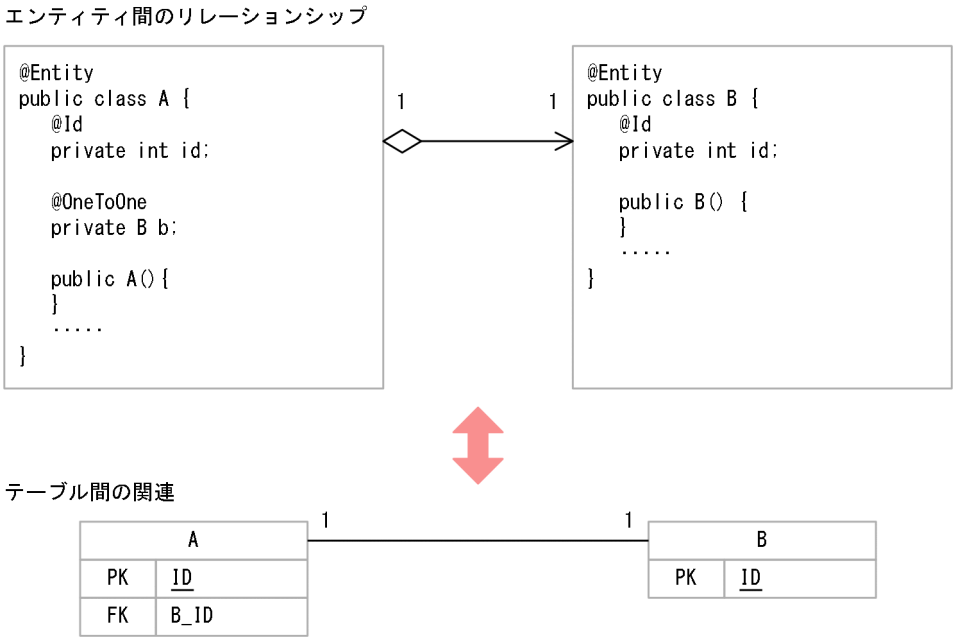
  外部キー列の名前

  エンティティ A のリレーションシップのプロパティ（または，フィールド）の名前_テーブル B のプライマリキー列の名前

  注　斜体は可変値を表します。

  外部キー列は，テーブル B のプライマリキーと同じ型を持ち，ユニークキー制約です。

図 8-9　単方向の OneToOne でのデフォルトマッピング



(b)　単方向の ManyToOne リレーションシップ

次に示す条件の場合に適用される単方向の ManyToOne リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A は，エンティティ B の単体のインスタンスを参照し，@ManyToOne（または，O/R マッピングファイルで該当する XML タグ）を設定します。
- エンティティ B からは，エンティティ A を参照しません。

適用されるデフォルトマッピング

- エンティティ A は，テーブル A にマップされます。
- エンティティ B は，テーブル B にマップされます。
- テーブル A は，テーブル B に対する外部キーを持つ必要があります。外部キー列の名前は，次のようになります。

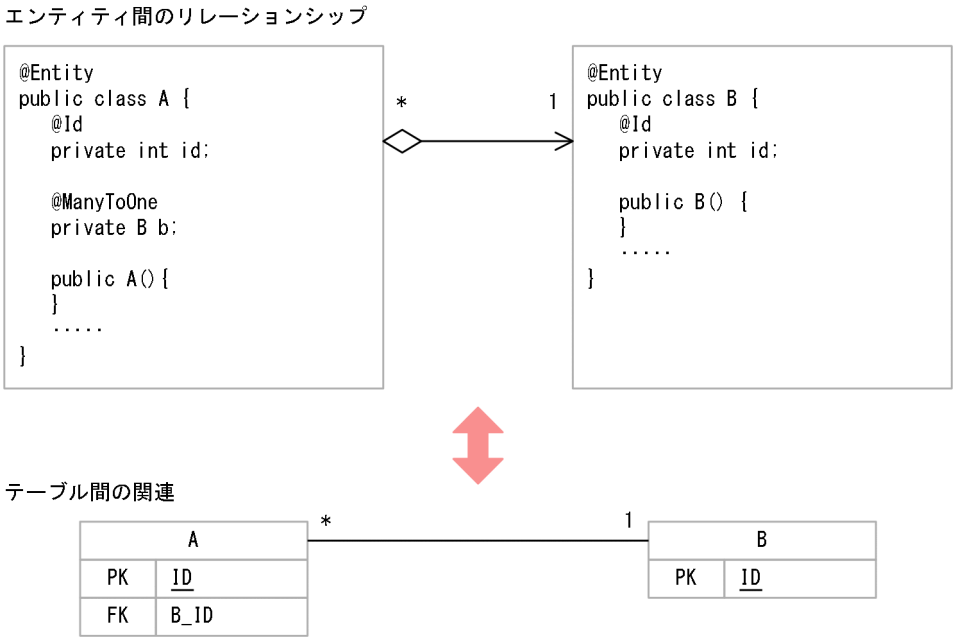
  外部キー列の名前

  エンティティ A のリレーションシップのプロパティ（またはフィールド）の名前_テーブル B のプライマリキー列の名前

  注　斜体は可変値を表します。

外部キー列は，テーブル B のプライマリキーと同じ型を持つ必要があります。

図 8-10　単方向の ManyToOne でのデフォルトマッピング



(2)　単方向の Multi-Valued リレーションシップ

単方向の OneToMany リレーションシップと単方向の ManyToMany リレーションシップのデフォルトマッピングについて説明します。

(a)　単方向の OneToMany リレーションシップ

次に示す条件の場合に適用される単方向の OneToMany リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A は，エンティティ B のコレクションを参照します。コレクションに@OneToMany（または，O/R マッピングファイルで該当する XML タグ）を設定します。
- エンティティ B からは，エンティティ A を参照しません。
- エンティティ A がリレーションシップの所有者です。

適用されるデフォルトマッピング

- エンティティ A は，テーブル A にマップされます。
- エンティティ B は，テーブル B にマップされます。
- テーブル A，B のほかに，所有者側のテーブルの名前が最初にくる A_B という名前の結合表が必要となります。この結合表は，二つの外部キー列を持ちます。

  一つ目の外部キー列は，テーブル A を参照し，テーブル A のプライマリキーと同じ型を持ちます。この外部キー列の名前は，次のようになります。

  外部キー列の名前

  エンティティ A の名前_テーブル A のプライマリキー列の名前

また，もう一つの外部キー列は，テーブル B を参照し，テーブル B の外部キーと同じ型を持ち，ユニークキー制約があります。この外部キー列の名前は，次のようになります。

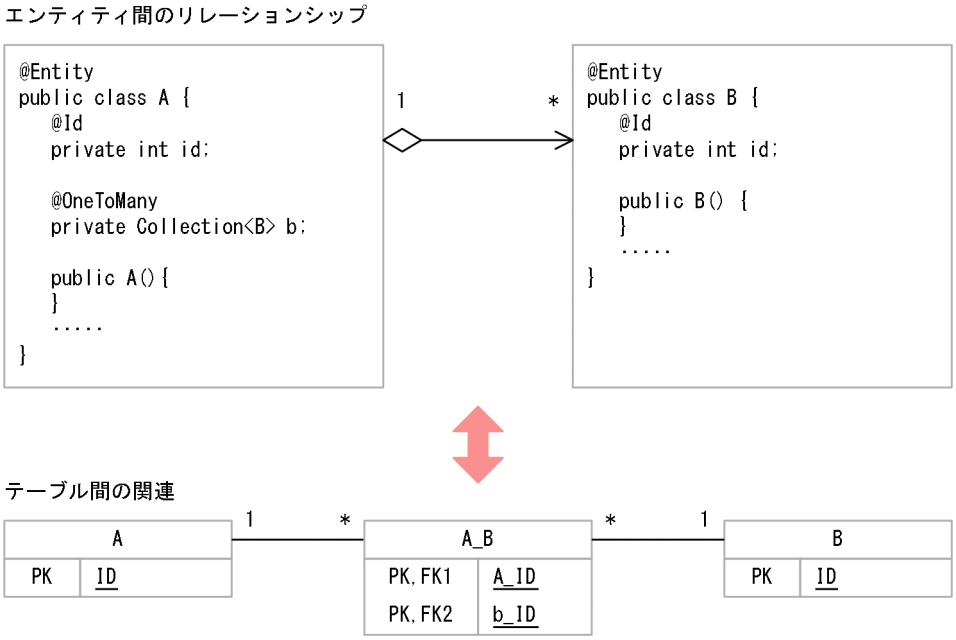
外部キー列の名前

エンティティ A のリレーションシッププロパティ（またはフィールド）の名前_テーブル B のプライマリキー列の名前

注　斜体は可変値を表します。

> **注意事項**
>
> この例のような結合表を使用した単方向の OneToMany リレーションシップは，CJPA プロバイダ以外の JPA プロバイダではサポートされていない可能性があります。単方向の OneToMany リレーションシップで作成したアプリケーションを CJPA プロバイダ以外の JPA プロバイダで動作させる場合は注意してください。

図 8-11　単方向 OneToMany でのデフォルトマッピング



(b)　単方向の ManyToMany リレーションシップ

次に示す条件の場合に適用される単方向の ManyToMany リレーションシップのデフォルトマッピングについて説明します。

条件

- エンティティ A は，エンティティ B のコレクションを参照します。コレクションに@ManyToMany（または，O/R マッピングファイルで該当する XML タグ）を設定します。
- エンティティ B は，エンティティ A を参照しません。
- 所有者はエンティティ A です。

適用されるデフォルトマッピング

- エンティティ A は，テーブル A にマップされます。
- エンティティ B は，テーブル B にマップされます。
- テーブル A，B のほかに，所有者側のテーブルの名前が最初にくる A_B という名前の結合表が必要となります。この結合表は，二つ外部キー列を持ちます。

  外部キー列の一つは，テーブル A を参照し，テーブル A の外部キーと同じ型を持ちます。この外部キー列の名前は，次のようになります。

  外部キー列の名前

  エンティティ A の名前_テーブル A のプライマリキーの名前

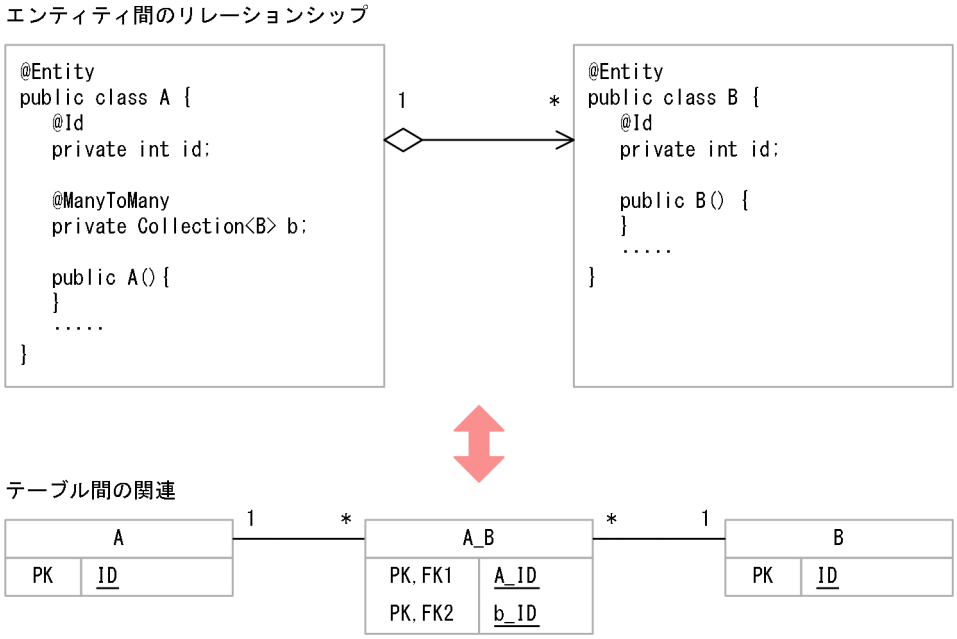
  また，もう一つの外部キー列は，テーブル B を参照し，テーブル B のプライマリキーと同じ型を持ちます。外部キー列の名前は，次のようになります。

  外部キー列の名前

  エンティティ A のリレーションシップのプロパティ（またはフィールド）の名前_テーブル B のプライマリキーの名前

  注　斜体は可変値を表します。

図 8-12　単方向 ManyToMany でのデフォルトマッピング



8.7　エンティティオブジェクトのキャッシュ機能

エンティティオブジェクトのキャッシュ機能とは，アプリケーションで使用したエンティティオブジェクトをメモリ内で保持するための機能です。エンティティオブジェクトのキャッシュ機能を使用している場合に，同じエンティティオブジェクトが操作されると，CJPA プロバイダ内にキャッシュされているエンティティオブジェクトが使用されます。データベースから再度データを読み込むことがないので，データベースへのアクセスが最小限になり，処理性能の負荷を軽減できます。なお，この機能は，CJPA プロバイダ独自の機能です。

ここでは，エンティティオブジェクトのキャッシュ機能について説明します。

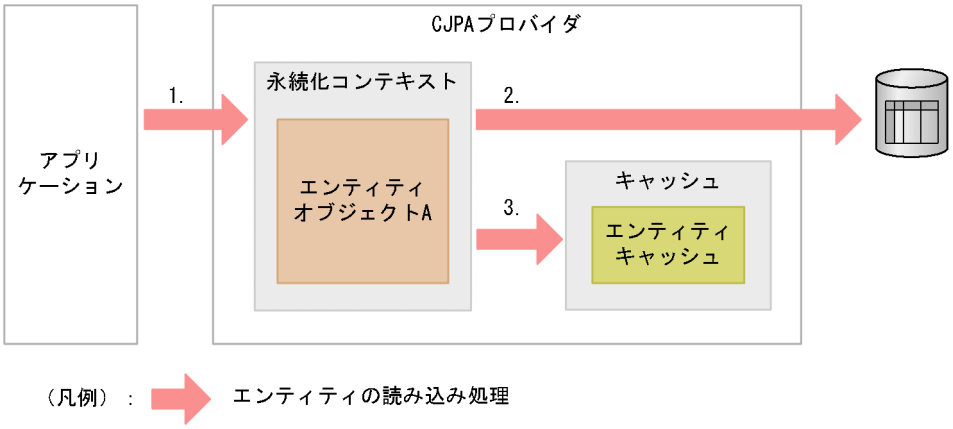
8.7.1　キャッシュ機能の処理

キャッシュ機能を使用している場合，同じエンティティに対して読み込みをすると，データベースではなく，キャッシュからデータが取得されるようになります。キャッシュ機能の処理の流れ，キャッシュの登録および更新のタイミング，およびキャッシュの更新処理の流れを説明します。

(1)　キャッシュ機能の処理の流れ

キャッシュ機能の処理の流れについて次の図に示します。

図 8-13　キャッシュ機能の処理の流れ



上記の図について説明します。

- **エンティティの読み込み処理**

  find などのメソッドで最初にエンティティを読み込むときは次の流れで処理されます。

  1. エンティティの読み込み処理をします。
  2. データベースからデータを取得します。
  3. 取得したデータのエンティティオブジェクトをキャッシュに登録します。

エンティティがリレーションシップを持つ場合でリレーションシップ先のエンティティを取得するとき，目的のエンティティがキャッシュに存在すると，データベースにアクセスしないでキャッシュのエンティティを参照します。

キャッシュは，永続化コンテキスト単位で存在します。

(2)　キャッシュの登録および更新のタイミング

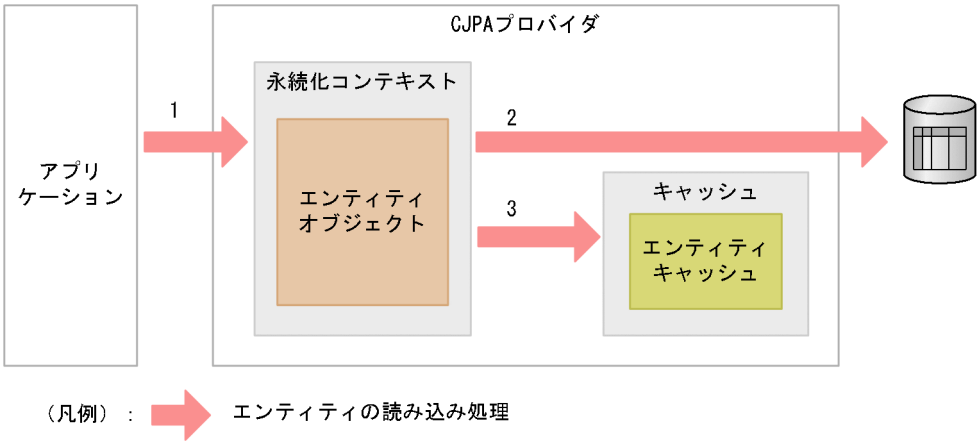
キャッシュへの登録および更新は次に示すタイミングで実施されます。

- キャッシュの登録のタイミング
  対象となるキャッシュが存在しない状態でのエンティティオブジェクトの読み込み時（find 操作）
- キャッシュの更新のタイミング
  - エンティティのリフレッシュ処理時（refresh 操作）
  - トランザクションのコミット時
  - 楽観的ロックの処理で例外が発生した場合
    なお，楽観的ロックの処理で OptimisticLockException 例外が発生した場合には，キャッシュに登録されているエンティティオブジェクトは削除されます。

(3)　キャッシュの更新処理の流れ

キャッシュの更新処理の流れについて次の図に示します。

図 8-14　キャッシュの更新処理



図について説明します。

1. エンティティオブジェクトに対して，次の操作を実施します。
   - refresh 操作
   - トランザクションのコミット
2. refresh 操作の場合は，データベースにアクセスします。

3. 永続化コンテキストにあるデータをキャッシュに登録して，キャッシュのデータを更新します。

なお，JPQL を実行した場合にもキャッシュは登録されます。登録されるタイミングは，対象となるキャッシュが存在しない状態で JPQL を実行した場合です。JPQL でもキャッシュのデータを使用しますが，キャッシュのデータの有無に関係なく，データベースのアクセスが発生します。このため，キャッシュによる処理性能の向上は期待できません。詳細については，(4)を参照してください。

キャッシュは，エンティティのオブジェクト単位で情報を保持しているため，クエリの実行時に返されるオブジェクトがエンティティ自身の場合には，キャッシュへの更新が実行されます。それ以外のフィールドを指定するような場合では，更新はされません。キャッシュの更新が有効になる場合と，ならない場合の JQPL の例を次に示します。

- キャッシュの更新が有効に行われる JPQL の例

```
SELECT emp FROM Employee AS emp
```

- キャッシュの更新がされない JPQL の例

```
SELECT emp.id, emp.name, emp.address FROM Employee AS emp
```

(4) JPQL とキャッシュの関係

JPQL でエンティティオブジェクト全体を取得するような場合，キャッシュに登録されている情報が存在すればキャッシュの情報を取得します。CJPA プロバイダのキャッシュ機能では，対象となるエンティティを特定するための ID であるプライマリキーが必要になります。プライマリキーを取得するには，JPQL の結果を取得する必要があり，この際にデータベースへのアクセスが発生します。データベースへのアクセスの結果から，プライマリキーを抽出して，キャッシュからエンティティオブジェクトを取得します。また，キャッシュに対象となるデータが存在しない場合は，データベースの情報からエンティティを作成します。

JPQL では，キャッシュのデータの有無に関係なく必ずデータベースへのアクセスが発生します。そのため，JPQL の場合，キャッシュによる性能向上は望めません。

8.7.2 キャッシュの参照形態とキャッシュタイプ

キャッシュの参照形態には次の 3 種類があります。

- **強参照**
  GC による回収の対象になりません。
- **弱参照（java.lang.ref.WeakReference）**
  弱可到達の場合，GC による回収の対象になります。なお，弱可到達かどうかは，java.lang.ref.WeakReference の仕様に依存します。
  CJPA プロバイダで弱参照のキャッシュが回収対象にならない例を次に示します。
  - 永続化コンテキストに登録されているエンティティオブジェクトのキャッシュ

- 弱可到達でないキャッシュにリレーションシップで参照されているキャッシュ
- エンティティの継承戦略を使用している場合，継承関係にある別のエンティティオブジェクトのキャッシュが，弱可到達でないキャッシュ

- ソフト参照（java.lang.ref.SoftReference）

  メモリ残量の低下時にキャッシュアウトする参照形態です。

  リソースの消費率や生存期間に応じて，GC による回収の対象になります。回収されるタイミングや，回収対象となるオブジェクトの選択方法などの仕様は，JavaVM に依存します。

どの形態でキャッシュを参照するかによってキャッシュタイプの種類が異なります。キャッシュの参照形態とキャッシュタイプの対応を次の表に示します。

表 8-16　キャッシュの参照形態とキャッシュタイプの対応

キャッシュの参照形態	キャッシュタイプ
強参照	Full
強参照＋弱参照※1	HardWeak
弱参照＋ソフト参照※1	SoftWeak
弱参照	Weak
なし※2	None

注※1　参照形態を組み合わせます。

注※2　エンティティオブジェクトをキャッシュしません。

CJPA プロバイダでは，キャッシュのタイプを選択できます。アプリケーションの設計や環境によってタイプを選択してください。キャッシュタイプは，persistence.xml で指定します。persistence.xml については「12.2　persistence.xml」を参照してください。

次にキャッシュタイプについて説明します。

(1)　Full

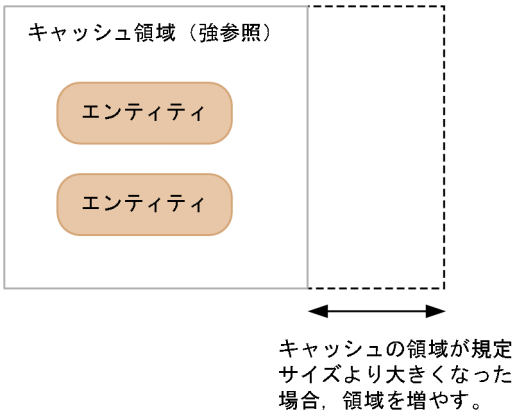
すべてのエンティティを強参照でキャッシュします。

キャッシュタイプに Full を指定した場合，データベースへのアクセスが少なくなるため，処理に対する負荷が少なくなります。ただし，メモリを占有し続けるため，メモリへの負荷が高くなります。

Full は，エンティティオブジェクトの存続期間が長く，頻繁なアクセスを必要とする少数のエンティティオブジェクトで作成される場合に指定します。また，多数のエンティティオブジェクトを読み取る場合は，メモリの負荷が高くなるので，複数レコードを一括更新する場合の使用はお勧めしません。

Full を指定した場合，指定されたキャッシュサイズで強参照領域を確保します。強参照領域が規定のサイズを超えた場合，Hashtable の仕様に基づいて領域を増やします。Full を指定した場合のキャッシュ内のイメージを次の図に示します。

図 8-15　Full の場合のキャッシュ内イメージ



(2)　HardWeak

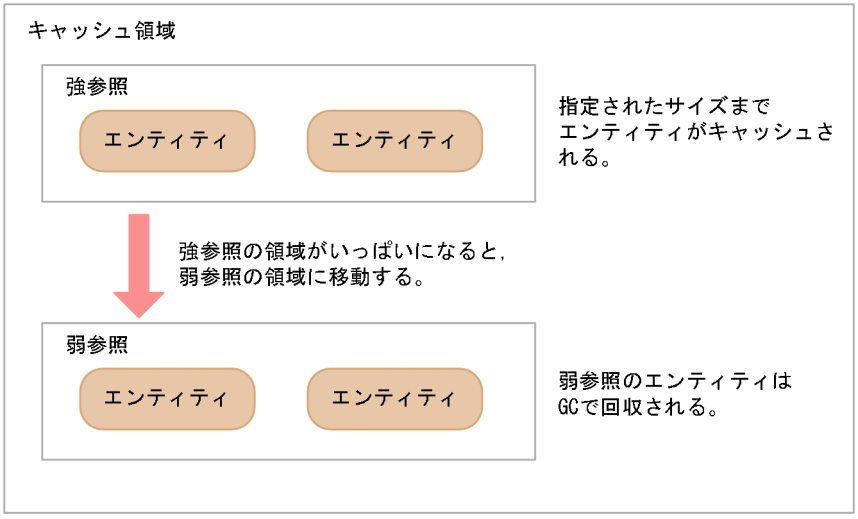
強参照と弱参照でキャッシュします。

エンティティオブジェクトをリストで保持するときには，強参照を使用します。キャッシュサイズに指定した値だけ，強参照領域を固定長で作成します。キャッシュサイズが指定された値に達すると，古いエンティティオブジェクトを弱参照に移動します。このとき，キャッシュへの登録が最も長く使用されていないエンティティオブジェクトから順に弱参照に移動されます。弱参照に移動したエンティティオブジェクトが使用されると，再度，強参照の領域に格納されます。

HardWeak を指定すると，存続期間の長いエンティティオブジェクトを使用して，キャッシュで使用されるメモリを効率良く制御できます。

SoftWeak を使用したシステムでメモリ不足の状態が頻繁に発生する場合には，ソフト参照の利点を生かすことができないため，HardWeak を使用してください。HardWeak の場合のキャッシュ内イメージを次の図に示します。

図 8-16　HardWeak の場合のキャッシュ内イメージ



HardWeak の場合，強参照のキャッシュ領域では強参照でキャッシュを保持します。また，弱参照のキャッシュ領域では弱参照でキャッシュを保持します。

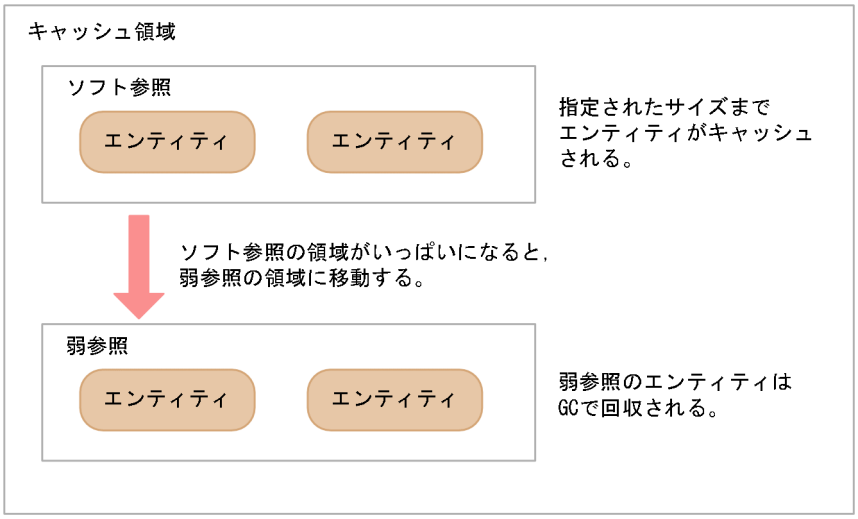
(3) SoftWeak

ソフト参照と弱参照でキャッシュします。

ソフト参照を使用して，エンティティオブジェクトをリストで保持し，キャッシュサイズに指定した値だけ，ソフト参照領域を固定長で作成します。キャッシュサイズが指定された値に達すると，古いエンティティオブジェクトを弱参照の領域に移動します。このとき，キャッシュへの登録が最も長く使用されていないエンティティオブジェクトから順に弱参照に移動されます。弱参照に移動したエンティティオブジェクトが使用されると，再度，強参照の領域に格納されます。

SoftWeak を指定すると，存続期間の長いエンティティオブジェクトを使用して，キャッシュで使用されるメモリを効率良く制御できます。このため，キャッシュ機能を使用する場合は，SoftWeak を指定することをお勧めします。SoftWeak の場合のキャッシュ内イメージを次の図に示します。

図 8-17　SoftWeak の場合のキャッシュ内イメージ



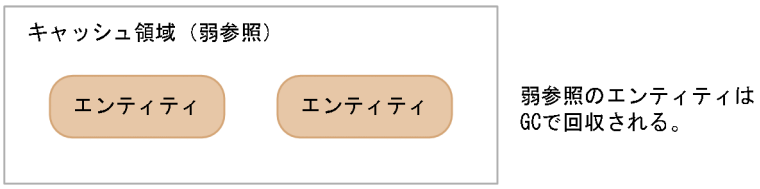
SoftWeak の場合，ソフト参照のキャッシュ領域ではソフト参照でキャッシュを保持します。また，弱参照のキャッシュ領域では弱参照でキャッシュを保持します。

(4) Weak

すべてのエンティティを弱参照でキャッシュします。

このため，すべてのエンティティオブジェクトが GC の対象になります。Weak を指定するとメモリへの負荷が少なくなりますが，GC によってキャッシュが削除されるおそれがあります。エンティティオブジェクトのキャッシュ機能を重視しないシステムで使用してください。Weak の場合のキャッシュ内イメージを次の図に示します。

図 8-18　Weak の場合のキャッシュ内イメージ



Weak の場合，弱参照でエンティティをキャッシュします。

(5)　NONE

エンティティオブジェクトはキャッシュされません。エンティティオブジェクトをデータベースから読み取ったあと，すぐに破棄する場合に使用します。

8.7.3　キャッシュ機能の有効範囲

キャッシュのデータは永続化コンテキスト単位で保持されます。このため，別の永続化コンテキストでエンティティオブジェクトが呼び出されていても，キャッシュからデータを取得できません。

なお，キャッシュのデータは，永続化コンテキストが生成されたときから破棄されるまでの間保持されます。

8.7.4　キャッシュ機能を使用するときの注意事項

ここでは，エンティティオブジェクトのキャッシュ機能を使用するときの注意事項を説明します。

(1)　クエリでデータを更新または削除した場合の注意

アプリケーション内で JPQL やネイティブクエリを使用して，データを更新したり，削除したりした場合，キャッシュの内容は更新されません。refresh 操作などを行って，データベースの内容を取得し直してください。

次に示す操作の場合，キャッシュのデータを読み込むため，データベースへの更新がされません。

1. データをエンティティオブジェクトに読み込む。
   キャッシュにもエンティティのデータが登録されます。
2. 1.で読み込んだデータを含む削除クエリを実行する。
   削除クエリの実行でデータは削除されますが，キャッシュは削除されません。
3. 1.と同じデータをエンティティオブジェクトに読み込む。
   1.と同じデータのため，キャッシュにあるデータを読み込みます。
4. 3.のデータを flush する。

データベースには該当する行がないため，追加または更新処理ができません。

(2)　キャッシュを使用する永続化コンテキストが複数ある場合の注意

キャッシュを使用することで，データベースへのアクセス頻度を下げることができます。ただし，一方でキャッシュによるデータのタイムラグが発生して，楽観的ロック例外の発生頻度が高くなるおそれもあります。

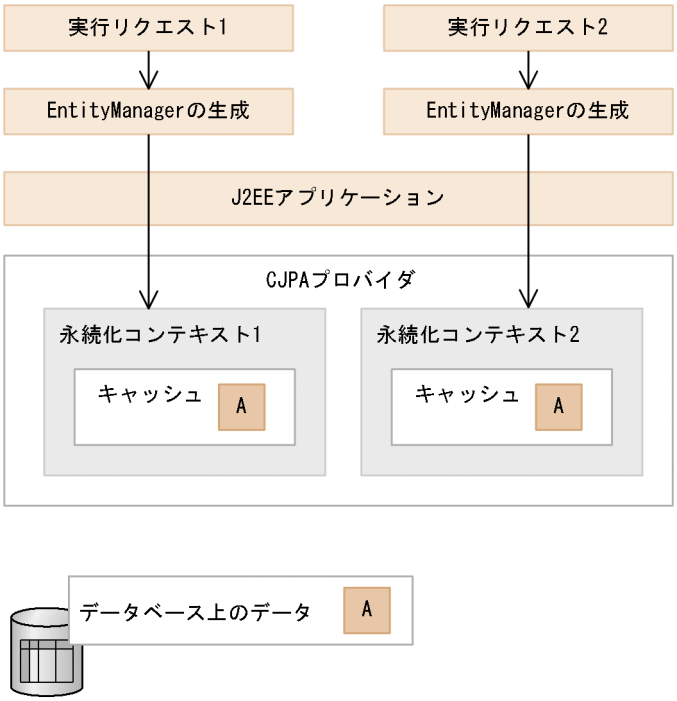
キャッシュは永続化コンテキスト単位に存在します。そのため，永続化コンテキストと対で存在するEntityManager が複数かつ同時期に生成され，同一プライマリキーのエンティティを同時に操作すると，一方でデータを更新しても他方でタイミング良く，更新後のデータを参照できないことがあります。これによって，楽観的ロック例外が発生しやすくなります。

次に，楽観的ロック例外が発生する仕組みと対処方法について説明します。

(a)　楽観的ロック例外が発生する例

ここでは，次の図に示す環境で，永続化コンテキスト単位にキャッシュが存在する場合を例に説明します。

図 8-19　この例で説明する環境

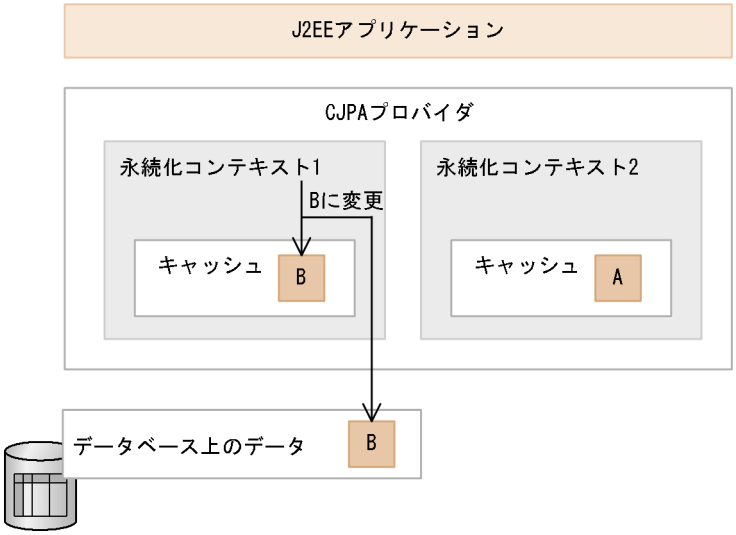


図の状態では，キャッシュとデータベースには不整合はなく，すでにキャッシュに A というデータが格納されているものとします。

1. 永続化コンテキスト 1 でデータを A から B に変更します。

   このとき，キャッシュとデータベースの内容が等しいため例外は発生しません。

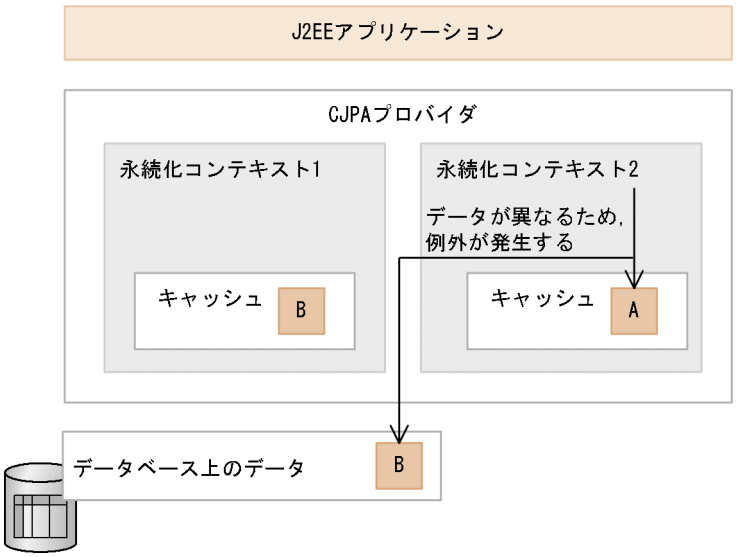
図 8-20　永続化コンテキスト 1 でのデータの変更



2. 1.の処理が終了後，永続化コンテキスト 2 で A のデータを変更します。

キャッシュのデータは変更されていないため，データベースのデータとキャッシュに不整合があります。このため，楽観的ロックによる例外が発生します。

図 8-21　永続化コンテキスト 2 でのデータの変更



このような環境の場合，キャッシュを使用すると，更新されないキャッシュが残ってしまい，楽観的ロックによる例外が発生することがあります。

(b)　対処方法

楽観的ロック例外が発生した場合は，キャッシュ内の該当オブジェクトは削除されます。このため，find メソッドを実行するか，または refresh メソッドを実行して関連するすべてのデータを再度データベースから取得してください。これによって，キャッシュのデータとデータベースを同期できます。

(3) キャッシュの登録および更新タイミングに関する注意事項

- JPQL を使用して悲観的ロックを取得する場合，クエリの実行時に返されるエンティティオブジェクトはキャッシュに登録されません。
- ネイティブクエリを利用して，@SqlResultSetMapping でエンティティを戻り値とするクエリを実行した場合，戻り値のエンティティオブジェクトはキャッシュに格納されます。
- エンティティの refresh 操作で，エンティティオブジェクトの読み込み後にほかのスレッドで OptimisticLockException が発生してキャッシュが削除されると，リフレッシュ処理を実行してもキャッシュにエンティティオブジェクトは登録されません。

8.8 プライマリキー値の自動採番

プライマリキー採番機能とは，エンティティオブジェクトを使用してレコードを挿入する際に，プライマリキー値を自動で生成する機能です。この機能によって，ユーザがプライマリキー値を指定しなくても一意の値が格納されるようになります。CJPA プロバイダでは，プライマリキー採番機能を提供しています。

プライマリキー値の生成方法

プライマリキー値の生成方法には次の 4 種類があります。

- TABLE

  プライマリキー値を保存しておくためのテーブルを使用して，プライマリキー値を生成する方法です。

- SEQUENCE

  データベースのシーケンスオブジェクトを使用して，プライマリキー値を生成する方法です。ただし，データベースに HiRDB を使用している場合，CJPA プロバイダでは TABLE と同じ処理を実施します。

- IDENTITY

  データベースの identity 列を利用してプライマリキー値を生成する方法です。ただし，CJPA プロバイダでは，使用しているデータベースの種類によって動作が異なります。

  HiRDB の場合，TABLE と同じ処理を実施します。

  Oracle の場合，SEQUENCE と同じ処理を実施します。

- AUTO

  使用しているデータベースに適した生成方法を選択します。CJPA プロバイダでは，HiRDB の場合，Oracle の場合ともに TABLE を選択します。

プライマリキー値が採番されるタイミング

CJPA プロバイダの場合，flush 操作またはトランザクションのコミットのタイミングでプライマリキー値が採番されます。

プライマリキー値の生成方法が SEQUENCE の場合の例

プライマリキー値の生成方法が SEQUENCE の場合の例を示します。この例では，事前に EMP_SEQ という名称のシーケンスオブジェクトが作成されているものとします。

```
@Entity
public class Employee {
・・・
    @SequenceGenerator(
        name=”EMPLOYEE_GENERATOR”,
        sequenceName=”EMP_SEQ”

    )
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE,
 generator=”EMPLOYEE_GENERATOR”)
    @Column(name=”EMPLOYEE_ID”)
    public Integer getId() {
        return id;
```

```
    }
 . . .
}
```

8.9　クエリ言語によるデータベース操作

JPA を使用してデータベースのデータを操作するには，javax.persistence.Query インタフェースを通して実施します。javax.persistence.Query インタフェースを使用すると，複数のレコードに対してまとめて，検索，更新，削除などの操作ができます。javax.persistence.Query インタフェースを使用するには，ユーザはクエリ言語を使用してデータベースを操作します。

CJPA プロバイダでは，クエリ言語として JPQL および SQL を使用できます。JPQL および SQL についてそれぞれ説明します。

- JPQL

  JPA 仕様で定義されているクエリ言語です。データベースに依存しない言語で，エンティティクラスを対象に操作します。

- SQL

  データベースに依存したクエリ言語です。ネイティブクエリともいいます。データベースのデータを対象に操作します。

JPQL や SQL によるデータベース操作については，「8.16　クエリ言語を利用したデータベースの参照および更新方法」を参照してください。また，JPQL の文法については，「8.17　JPQL の記述方法」を参照してください。

8.10　楽観的ロック

CJPA プロバイダでは，EntityManager と永続化コンテキストを利用して永続化対象のエンティティを管理します。変更されたエンティティの情報は，flush メソッドまたはトランザクションのコミット処理が実行されたときにデータベースに反映されます。複数のトランザクションで同時にデータベースの同じ行を更新するおそれがある場合，データの整合性を保証する必要があります。データの整合性を保証するために，CJPA プロバイダでは楽観的ロックを提供しています。

楽観的ロックとは，データの更新処理を始めてから更新が完了するまでの間に，ほかから更新されていないことを確認するためのロック機能です。楽観的ロックはデータベースをロックしないので，デッドロックなどが発生しないという利点があります。

ここでは，楽観的ロックについて説明します。

8.10.1　楽観的ロックの処理

楽観的ロックを使用すると，CJPA プロバイダはデータベースのデータがほかのアプリケーションから更新されていないかをユーザに代わってチェックします。データベースのデータが更新されていると，CJPA プロバイダは例外を発生させて，ユーザにデータが更新されていることを通知します。また，トランザクションをロールバックにマークします。

(1)　データ更新の有無のチェック方法

データが更新されているかどうかは，データベースのテーブル上に用意したバージョン列の更新の有無によってチェックします。データベース上のデータが更新されると，バージョン列のバージョン番号が更新されます。これによって，ほかのアプリケーションなどからデータベースが更新されたことがわかります。データベースを更新するときのバージョン列の状態と動作について次の表に示します。

表 8-17　データベースを更新するときのバージョン列の状態と動作

テーブルのバージョン列の状態	動作
バージョン列の値が更新されていない場合	CJPA プロバイダは，エンティティの情報をデータベースに反映します。このとき，データベースのバージョン列の値を更新します。
バージョン列の値が更新されている場合	ほかのアプリケーションなどからデータベースのデータが更新されていることを表します。このため，CJPA プロバイダは OptimisticLockException を発生させて，トランザクションをロールバックにマークします。

このように，バージョン列の状態によって，エンティティを読み込んだ状態からデータベースを更新するまでに，ほかのトランザクションがデータを更新していないことを保証できます。

(2) 永続化フィールドおよびリレーションシップのバージョンチェック

楽観的ロックを使用するには，エンティティの永続化フィールドおよびリレーションシップの両方をバージョンチェックの対象にします。バージョンチェックの対象とするには，エンティティにバージョン列に対応する Version フィールド（プロパティ）を設定してください。Version フィールドは@Version または O/R マッピングファイルの<version>タグを使用して設定します。

エンティティのバージョンチェックは次のどちらかのタイミングで実行されます。

- エンティティの状態が変更され，その変更をデータベースに書き込むとき
- merge 処理によって，エンティティが managed 状態に変更されたとき※

  注※

  バージョンチェックは merge 実行時だけでなく，flush またはトランザクションのコミット時にもチェックが実行されます。

バージョンチェックによってエンティティのバージョンが古いことが判明した場合，
OptimisticLockException が発生します。また，トランザクションはロールバックにマークされます。

(3) flush 操作またはトランザクションの決着時のバージョンチェック

エンティティを flush 操作またはトランザクションの決着時にバージョンチェックの対象にできます。バージョンチェックの対象とするには，EntityManager の lock メソッドにエンティティを指定します。EntityManager の lock メソッドを使用することで，トランザクションでのバージョンチェック対象にエンティティを追加したり，バージョン列の更新方針を変更したりできます。

CJPA プロバイダでは，バージョン列の更新タイミング（lock メソッドの LockModeType）として LockModeType.READ および LockModeType.WRITE の二つをサポートしています。バージョン列の更新タイミングの指定内容にかかわらず，CJPA プロバイダではトランザクションの決着時に次に示す二つの事象が起きないことを保証します。

- ダーティリード（Dirty Read）

  トランザクション「T1」が行を変更します。次に，「T1」がコミットやロールバックを行う前に別のトランザクション「T2」が同じ行を読み込み，変更した値を取得します。最終的に「T2」がコミットに成功します。

  「T1」がコミットかロールバックをするかどうかが重要ではなく，「T2」のコミットの前またはあとのどちらで「T1」のコミットかロールバックが行われるかが重要になります。
- 繰り返し不可能な読み込み（un-Repeatable Read）

  トランザクション「T1」が行を読み込みます。次に，「T1」がコミットする前に，もう一方のトランザクション「T2」がその行の変更や削除を実施します。最終的に両方のトランザクションはコミットに成功します。

LockModeType として LockModeType.WRITE を指定すると，エンティティの状態変更がない場合でもバージョン列は強制的に更新されます。バージョン列の更新は flush またはトランザクションのコミッ

トが呼び出されたタイミングで実行されます。なお，バージョン列の更新前にエンティティが削除された場合，バージョン列の更新は省略されることもあります。

8.10.2　楽観的ロックに失敗した場合の例外処理

明示的に楽観的ロックの実行を確認したい場合は，flush()メソッドを呼び出します。flush()メソッドの呼び出しで楽観的ロック例外が発生した場合，例外処理をキャッチしてリカバリ処理を行うことができます。楽観的ロックに問題がない場合，flush()メソッドによってエンティティのバージョンのチェックおよびVersion 列が更新されます。

例外処理の記述例を次に示します。

```
try{
  em.flush();
} catch (OptimisticLockException e){
  // 例外処理
}
```

この例に示すように，例外処理の中で OptimisticLockException をラップすることで楽観的ロックの例外をアプリケーション例外として見せることができます。ただし，この場合のトランザクションはロールバックにマークされているので，コミットできないことに注意してください。

なお，CJPA プロバイダでは，例外の原因となったエンティティを OptimisticLockException に格納しません。OptimisticLockException の getEntity()メソッドは常に null 値を返します。

8.10.3　楽観的ロックを使用する際の注意事項

ここでは，楽観的ロックを使用する際の注意事項について説明します。

(1)　Version フィールドの設定時の注意事項

Version フィールドの設定時の注意事項を次に示します。

- Version フィールドが設定されていない場合，そのエンティティに対するバージョンチェックは実施しません。その場合，エンティティとデータベースの間の整合性を保つようなアプリケーションをユーザ責任で作成してください。
- トランザクションに Version フィールドが設定されているエンティティと設定されていないエンティティが含まれている場合，Version フィールドが設定されているエンティティに対してだけバージョンチェックが実行されます。なお，エンティティにバージョンが含まれていないことはトランザクションの決着処理には影響ありません。
- ユーザは Version フィールドの値を参照できますが，更新はしないでください。ただし，バルク更新処理では Version フィールドの値を更新できます。

(2) lock メソッド使用時の注意

lock メソッド使用時の注意事項を次に示します。

- Version フィールド（プロパティ）を持たないエンティティに対する EntityManager の lock メソッドはサポートしていません。Version フィールド（プロパティ）を持たないエンティティを指定して，lock(entity, LockModeType.READ)または lock(entity, LockModeType.WRITE)が呼ばれた場合は，PersistenceException が発生します。
- Version フィールド（プロパティ）を持つエンティティの状態が更新される場合は，lock メソッドの呼び出し有無にかかわらず，ダーティリード，および繰り返し不可能な読み込みの現象は発生しません。
- CJPA プロバイダでは LockModeType.READ が指定された場合，エンティティに該当するデータベースの値に変更がないかを確認するために，UPDATE 文を発行します。このため，データベースに対して UPDATE 文による排他が掛かります。UPDATE 文は，flush 処理の実行時およびトランザクションのコミット時に発行されます。エンティティオブジェクトの状態に変更がない場合でも発行されます。ただし，エンティティが削除された場合は発行されません。

(3) HiRDB でのクライアント単位の排他制御について

CJPA プロバイダの楽観的ロックは，データベースの Isolation レベルが Read Committed でアクセスされることを想定したロック方式です。データベースが HiRDB の場合，Isolation レベルがデフォルトの設定では Repeatable Read であるため，Read Committed に変更する必要があります。

クライアント単位の Isolation レベルは，クライアント環境変数のデータ保証レベルの PDISLLVL パラメタで設定します。デフォルトの値は Repeatable Read (2)です。このため，Read Committed (1) に変更してください。変更例を次に示します。

変更例：PDISLLVL=1

クライアント環境変数は Connector 属性ファイルの<config-property>タグで environmentVariables プロパティの値に指定するか，または HiRDB のクライアント環境変数グループの設定ファイルに追加してください。

クライアント環境変数のデータ保証レベルをデフォルト設定の Repeatable Read で動作させた場合は，共有モードで排他が掛かります。このため，find メソッドなどの参照系 SQL の発行と flush メソッドなどの更新系 SQL の発行を組み合わせることによって，デッドロックが発生しやすくなるので注意してください。

8.11　JPQL での悲観的ロック

悲観的ロックとは，複数のトランザクションによってデータベース上の同じレコードを更新するときに，対象となるレコードを占有ロックする方法です。あるトランザクションがあるレコードに対して悲観的ロックを掛けると，ほかのトランザクションはそのレコードを参照または更新できません（ただし，Oracle の場合，参照はできます）。悲観的ロックは JPQL を使用しているときだけ使用できます。

悲観的ロックを使用すると，ロックを取得したトランザクションが終了するまでロックの解放待ちが発生します。このため，楽観的ロックよりも同時実行性はありませんが，楽観的ロックで発生するトランザクションのコミット時のエラーを回避できます。

悲観的ロックの指定方法

悲観的ロックは，CJPA プロバイダがサポートするクエリヒントを利用することで実現します。悲観的ロックは，Query メソッドの setHint()メソッドまたは@NamedQuery の属性に@QueryHint を指定して実行します。

悲観的ロック機能の実装例

悲観的ロック機能の実装例を次に示します。

実装例 1

Query メソッドの setHint()メソッドで悲観的ロックを指定する場合の例を次に示します。

```
Query query = manager.createQuery("SELECT emp FROM  Employee  AS emp");
query.setHint("cosminexus.jpa.pessimistic-lock","Lock");
```

実装例 2

@NamedQuery の属性の@QueryHint で悲観的ロックを指定する場合の例を次に示します。

```
@NamedQuery(
name="employee_list",
query="SELECT emp FROM  Employee AS emp",
hints={ @QueryHint(name="cosminexus.jpa.pessimistic-lock", value="Lock") }
)
@Entity
public class Employee{
. . .
}
```

注意事項

悲観的ロックが使用できるのは JPQL の時だけです。createNativeQuery メソッドや@NamedNativeQuery の hints 属性に@QueryHint を指定しても有効にはなりません。また，O/R マッピングファイルの<named-native-query>タグの hint 属性に指定した場合も有効にはなりません。なお，CJPA プロバイダでの悲観的ロックの排他仕様は，使用するデータベースの仕様に準じます。

8.12　エンティティクラスの作成

JPA を利用したアプリケーションを作成するには，アプリケーションでエンティティクラスを定義します。エンティティクラスは，データベースのテーブルのレコードを Java のオブジェクトとして扱うためのクラスです。ユーザがアプリケーションで new オペレーションを行うと，エンティティクラスのインスタンスが生成されます。

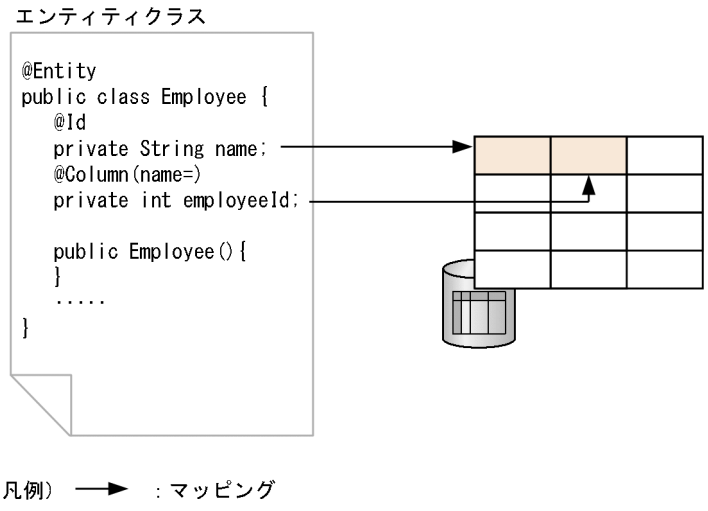
ここでは，JPA アプリケーションの作成について説明します。

8.12.1　エンティティクラスとデータベースの対応の定義

エンティティクラスは，データベースのテーブルの行に対応します。また，エンティティクラスが保持するフィールドは，テーブルのカラムの値に対応します。ユーザがエンティティクラスのインスタンスに対してフィールドの値を更新すると，CJPA プロバイダによって対応するデータベースのテーブルのカラムも更新されます。そのため，ユーザはデータベースに対して SQL を発行することなく，データベースの状態を変更できます。

エンティティクラスとデータベースのテーブルとのマッピングについて次の図に示します。

図 8-22　エンティティクラスとデータベースのテーブルとのマッピング



エンティティのフィールドとデータベースのカラムの対応関係は，アノテーションまたは O/R マッピングファイルで定義します。アノテーションと O/R マッピングファイルのどちらでも定義はできます。ただし，両方で定義している場合は，O/R マッピングファイルの設定がアノテーションよりも優先されます。アノテーションと O/R マッピングファイルとで同じ設定項目に対して異なる値を設定している場合は，アノテーションの値を O/R マッピングファイルの値で上書きします。

8.12.2　エンティティクラスの作成要件

JPA を使用したアプリケーションを作成する場合には，JPA 仕様で決められたエンティティクラスの作成要件やデータベースのマッピング要件を守る必要があります。作成要件を次に示します。

- エンティティクラスは@Entity または O/R マッピングファイルの<entity>タグに指定する必要があります。
- エンティティクラスは引数なしのコンストラクタを持ちます。
- enum やインタフェースはエンティティクラスとしてはいけません。
- エンティティクラスのクラス階層のルートとなるエンティティまたはマップドスーパークラスでは，プライマリキーを持つ必要があります。プライマリキーは，エンティティ階層で必ず一つ定義してください。
- エンティティクラスのインスタンスを値渡しでメソッドの引数として渡す場合，Serializable インタフェースを実装する必要があります。
- データベースのカラムの状態はエンティティのインスタンス変数で表現し，インスタンス変数は JavaBean のプロパティに対応します。なお，インスタンス変数は，クライアントから直接アクセスして値を変更してはいけません。アクセサメソッド（getter／setter メソッド）や，ビジネスメソッド経由で値を変更してください。
- エンティティの永続インスタンス変数は，private，protected，または package から参照できるアクセスレベルにしてください。
- 引数なしのコンストラクタは，public か protected で宣言してください。
- エンティティクラスは final にしないでください。また，エンティティクラスの永続化インスタンス変数とすべてのメソッドも final にしてはいけません。

CJPA プロバイダでは，これらの条件に合わない場合，例外が発生するおそれがあります。なお，例外が発生しない場合でもこれらの条件に合わないエンティティクラスを作成している場合の動作は保証しません。

また，CJPA プロバイダの場合，エンティティクラスでは，データベースの一つのカラムを複数のフィールドに対応づけないでください。この条件に合わない場合，アプリケーション実行時に例外が発生することがあります。例外が発生しない場合でも，動作は保証しません。

8.12.3　エンティティクラスのフィールドに対するアクセス方法の指定

CJPA プロバイダは，エンティティの状態をデータベースに書き込んだり，データベースの状態をエンティティとして読み込んだりするときに，エンティティクラスのフィールドにアクセスします。このときのアクセス方法をアクセスタイプと呼びます。アクセスタイプには，プロパティとフィールドの 2 種類があります。また，アクセスタイプはアノテーションまたは O/R マッピングファイルで指定します。アクセスタイプと指定方法について次の表に示します。

表 8-18　アクセスタイプと指定方法

アクセスタイプ	説明	指定方法	
		アノテーション	O/R マッピングファイル
プロパティ	getter メソッドを経由して取得する方法です。	フィールドの getter メソッドにアノテーションを指定します。	<access>タグに PROPERTY を指定します。
フィールド	インスタンス変数を直接参照する方法です。	フィールドにアノテーションを指定します。	<access>タグに FIELD を指定します。

アクセスタイプがプロパティの場合，エンティティが保持するプロパティを**永続化プロパティ**といいます。また，アクセスタイプがフィールドの場合，エンティティのフィールドを**永続化フィールド**といいます。

アクセスタイプに関連する注意事項

アクセスタイプを指定するときには，次の点を考慮してください。

- アクセスタイプがフィールドの場合，CJPA プロバイダは永続化フィールドに直接アクセスします。@Transient を設定されていないインスタンス変数は，永続化の対象となります。
- アクセスタイプがプロパティの場合，CJPA プロバイダはアクセサメソッドを利用して永続化プロパティの値を取得します。@Transient がアクセサメソッドに設定されていないプロパティは，永続化の対象となります。
- アクセスタイプがプロパティの場合，setter メソッドにマッピングアノテーションは設定できません。CJPA プロバイダの場合，setter メソッドに設定したマッピングアノテーションは無視されます。
- @Transient または O/R マッピングファイルで<transient>タグが付与されたフィールドとプロパティには，マッピングアノテーションは設定できません。設定した場合には，アプリケーションの開始時に例外が発生します。
- プロパティのアクセサメソッドは public または protected にしてください。これは，JPA 仕様で禁止されていますが，アプリケーションサーバの JPA 機能ではチェックしません。また，private であっても例外も発生しません。
- 永続化フィールドと永続化プロパティのアクセサメソッドの両方にマッピングアノテーションが適用された場合，永続化プロパティのアクセサメソッドに設定されたアノテーションはすべて無視されます。

8.12.4　アクセサメソッドの作成

ここでは，アクセサメソッドのシグネチャ規則，およびアクセサメソッドへのビジネスロジックの追加について説明します。

(1)　アクセサメソッドのメソッドシグネチャ規則

CJPA プロバイダが永続化プロパティに対してアクセスする場合，プロパティのアクセサメソッドは，次に示す JavaBeans と同じメソッドシグネチャ規則に従っている必要があります。

- T getProperty()
- void setProperty(T t)

boolean の戻り値を返すプロパティの場合，getter メソッドは isProperty という名前にすることもできます。なお，CJPA プロバイダを使用する場合，getter メソッドと setter メソッドのどちらかしかないときには，アプリケーション開始時に例外が発生します。

また，永続化フィールド，永続化プロパティでコレクション値を扱う場合は，次に示すコレクション値をインタフェースで定義してください。

- java.util.Collection
- java.util.Set
- java.util.List
- java.util.Map

永続化プロパティがコレクション値となる場合，アクセサメソッドのメソッドシグネチャはこれらのインタフェースのどれかにしてください。または，これらのコレクションのジェネリックな型を使用することもできます（例：Set<T>）。

(2) アクセサメソッドへのビジネスロジックの追加

アクセサメソッドは，プロパティの setter/getter 処理に加えて，値を検証するなどのビジネスロジックを含むことができます。アクセスタイプがプロパティの場合，CJPA プロバイダがアクセサメソッドを呼び出すタイミングでビジネスロジックが動作します。

ただし，この場合には，次の点に注意してください。

- CJPA プロバイダの実行時に，永続状態をロードして格納するアクセサメソッドが呼び出される順番は定義されていません。このため，getter に含まれるロジックの実行順序は未定となります。
- アクセスタイプがプロパティで，永続化プロパティで lazy フェッチが指定されている場合で，移植性を求めるには，エンティティの内容が CJPA プロバイダにフェッチされるまで，エンティティの内容にアクセスしないことをお勧めします。
- アクセスタイプがプロパティの場合で，ビジネスロジックとして値を変更するロジックを追加したときには，CJPA プロバイダではデータの一貫性は保証しません。

プロパティアクセサメソッドで Runtime 例外が発生すると，現在のトランザクションはロールバックにマークされます。CJPA プロバイダがエンティティの永続状態の内容を読み込んだり，格納する際のアクセサメソッドで例外が発生したりした場合，トランザクションはロールバックされます。また，アプリケーション例外をラップする PersistenceException 例外が発生します。

8.12.5　エンティティの永続化フィールドおよび永続プロパティの型

エンティティの永続化フィールド／永続化プロパティは，次に示す型にしてください。

- Java のプリミティブ型
- java.lang.String
- ほかのシリアライズ型
- プリミティブ型のラッパークラス
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- ユーザが定義するシリアライズな型
- byte[]
- Byte[]
- char[]
- Character[]
- enum
- エンティティタイプとエンティティタイプで複製できるコレクション
- 埋め込みできるクラス

CJPA プロバイダを使用する場合は，上記以外の型が指定されたときの動作は保証しません。アプリケーションの実行時に例外が発生するおそれもあります。

ただし，CJPA プロバイダを使用している場合，エンティティのインスタンス変数の型は，データベースのカラムの型と対応づけしている必要があります。Java の型とデータベースの型の対応づけは，JDBC ドライバが行います。CJPA プロバイダでは，Oracle 接続時には Oracle が提供する Oracle JDBC Thin Driver を JDBC ドライバとして利用します。HiRDB 接続時には HiRDB Type4 JDBC Driver を JDBC ドライバとして利用します。ユーザは JDBC ドライバがサポートする型の変換に合わせてエンティティのインスタンス変数の型を決定してください。

8.12.6　エンティティでのプライマリキーの指定

エンティティでは，プライマリキーをクラス階層の中で必ず指定してください。プライマリキーを指定する場合には，次に示すルールに従ってください。

- 単体の（複合型でない）プライマリキーは，永続化フィールドまたは永続化プロパティに@Id を指定するか，O/R マッピングファイルで指定します。これによって，エンティティのフィールドとの対応づけをしてください。
- 複合型のプライマリキーの場合，@EmbeddedId として単一のフィールドでプライマリキークラスを指定するか，または@IdClass と@Id によって，フィールドのセットとして複合プライマリキーを指定します。
- 複合型のプライマリキーの場合，プライマリキークラスと呼ばれるプライマリキーを含むクラスを作成します。

これらの条件に従っていない場合は，アプリケーション開始時に例外が発生します。

また，アプリケーションでエンティティのプライマリキーの値を変更しないでください。アプリケーションでプライマリキーの値を変更した場合，実行時に例外が発生します。

(1)　プライマリキーの型

単体または複合型のプライマリキーは，次に示す型のどれかにしてください。

- Java のプリミティブ型
- プリミティブをラッパした型
- java.lang.String
- java.util.Date
- java.sql.Date

なお，近似値型（例えば浮動小数点数型）をプライマリキーとして指定すると，CJPA プロバイダの場合，丸め誤差が発生したり，equals メソッドの結果に信頼性がないという問題が発生したりします。このため，CJPA プロバイダではプライマリキーに近似値型を使用した場合の動作は保証しません。java.util.Date がプライマリキーとしてフィールド／プロパティで使用される場合，temporal type 属性は DATE 型として指定する必要があります。

(2)　複合型のプライマリキー

エンティティでは複合型のプライマリキーを扱うことができます。エンティティで複合型のプライマリキーを指定するには，埋め込み型クラスを利用する方法と@IdClass を利用する方法の 2 とおりがあります。それぞれの方法を説明します。

(a)　埋め込み型クラスを利用する方法

埋め込み型クラスを利用するには，@Embeddable を付与したクラスを作成し，そのクラスのフィールドとして複合型のプライマリキーを定義します。エンティティクラスでは，@Embeddable を付与したクラスの型のフィールドを定義して，@EmbeddedId をアノテートします。エンティティクラスの例と埋め込み型クラスの例を次に示します。

- エンティティクラスの例

```
@Entity
public class Employee {
    private EmployeePK employeePK;

    public Employee(){
    }

    @EmbeddedId
    public EmployeePK getEmployeePK(){
      retrun this.employeePK;
    }

    public void setEmployeePK(EmployeePK employeePK){
      this.employeePK = employeePK;
    }
     . . .
}
```

- 埋め込み型クラスの例

```
@Embeddable
public class EmployeePK {
    private String name;
    private int employeeId;

    public EmployeePK(){
    }

    public boolean equals(Object obj){
       . . .
    }

    public int hashCode(){
       . . .
    }
     . . .
}
```

埋め込み型クラスについては「(3)　埋め込み型クラス」を参照してください。なお，アノテーションの代わりに O/R マッピングファイルを利用することもできます。

(b)　@IdClass を利用する方法

@IdClass を利用するには，エンティティクラスでプライマリキーに対応する複数のインスタンス変数を定義して@Id を付与します。また，@IdClass を使用してプライマリキークラスを指定します。プライマ

リキークラスでは，エンティティで定義したプライマリキーと同じ名前と型を持つフィールドまたはプロパティを定義します。エンティティクラスの例とプライマリキークラスの例を次に示します。

- エンティティクラスの例

```
@Entity
@IdClass(EmployeePK.class)
public class Employee {
    private String name;
    private int employeeId;

    public Employee(){
    }

    @Id
    public String getName(){
      retrun this.name;
    }

    public void setName(String name){
      this.name = name;
    }

    @Id
    public int getEmployeeId(){
      retrun this.employeeId;
    }

    public void setName(int employeeId){
      this.employeeId = employeeId;
    }
     . . .
}
```

- プライマリキークラスの例

```
public class EmployeePK implements Serializable {
    private String name;
    private int employeeId;

    public EmployeePK(){
    }

    public boolean equals(Object obj){
       . . .
    }

    public int hashCode(){
       . . .
    }
     . . .
}
```

埋め込み型クラスの場合と同様に，アノテーションの代わりにO/Rマッピングファイルを利用することもできます。なお，プライマリキークラスのアクセスタイプは，プライマリキーに対応するエンティティクラスのアクセスタイプで決定します。

複合型のプライマリキーを扱うためには，埋め込み型クラスまたは@IdClass のどちらかを使用します。ただし，次に示すルールに従ってください。

- プライマリキークラスは，public で引数のないコンストラクタを持たなければならなりません。
- 永続化プロパティを使用する場合，プライマリキークラスのプロパティは public または protected にしてください。
- プライマリキークラスは，直列化可能にしてください。
- プライマリキークラスでは equals と hashCode メソッドを定義します。マップされているデータベース上で主キーが等しい場合には equals で true を返し，hashCode の値を等しくする必要があります。
- 複合プライマリキーは，埋め込みクラスとしてマップされているか，エンティティクラスの複数のフィールド／プロパティをマップされている必要があります。
- プライマリキークラスがエンティティクラスの複合フィールド／プロパティにマップされる場合，プライマリキークラスのプライマリキーのフィールド／プロパティの名前と，エンティティクラスの名前を一致させてください。また，型も同じにしてください。

CJPA プロバイダでは，これらの条件を満たさない場合動作は保証しません。また，条件を満たさない場合はアプリケーションの開始時に例外が発生することもあります。

(3) 埋め込み型クラス

永続化対象の幾つかのフィールドをまとめたクラスを用意すると，エンティティのフィールドとして保持できます。このようなクラスを埋め込み型クラスといいます。

埋め込み型クラスは，エンティティに埋め込まれてエンティティと同じデータベースのテーブルにマップされます。このため，エンティティとは異なり，プライマリキーを持ちません。

ユーザが埋め込み型クラスを利用する場合，埋め込むクラスには@Embeddable を設定します。また，埋め込まれる側のエンティティクラスでは，埋め込み先のフィールド，プロパティに@Embedded を指定します。なお，アノテーションの代わりに O/R マッピングファイルで同様に定義することもできます。

埋め込み型クラスは複合型のプライマリキーを定義するために利用することもできます。この場合，埋め込まれるエンティティクラスでは，@Embedded の代わりに@EmbeddedId を設定します。

埋め込み型クラスでは，次に示す作成要件を必ず守ってください。

1. 埋め込み型クラスは，必ず@Embeddable で定義するか，O/R マッピングファイルの<embeddable>タグで定義してください。
2. enum やインタフェースを埋め込み型クラスとしないでください。
3. 埋め込み型クラスを含むエンティティクラスを detached オブジェクトとして値渡ししている場合，Serializable インタフェースを実装してください。
4. 埋め込み型クラスおよび埋め込み型クラスの永続化インスタンス変数と，すべてのメソッドでは final にしないでください。

5. 引数なしのコンストラクタを持つ必要があります。
6. 引数なしコンストラクタは，public か protected で宣言してください。
7. 埋め込み型クラスのインスタンス変数は，private，protected，または package から参照できなければなりません。
8. 埋め込み型クラスの永続化インスタンス変数は，クライアントから直接アクセスされないようにしてください。エンティティのアクセサメソッド（getter／setter メソッド）や，ほかのビジネスメソッドでアクセスしないでください。

CJPA プロバイダでは，1.および 2.の条件を満たさない場合には例外が発生して，アプリケーションの開始に失敗します。また，3.から 8.についても例外が発生するおそれがありますが，例外が発生しない場合でも動作は保証しません。

埋め込み型クラスのアクセスタイプは，埋め込まれた側のエンティティクラスのアクセスタイプで決定します。

埋め込み型クラスを利用する場合，埋め込みの階層は一つにしてください。また，複数のエンティティから埋め込み型クラスのオブジェクトを共有することはできません。これらの条件を満たさない場合，CJPA プロバイダでは動作を保証しません。

8.12.7 永続化フィールドおよび永続化プロパティのデフォルトマッピング規則

リレーションシップ以外の永続化フィールドまたは永続化プロパティに対して，O/R マッピング情報が指定されていない場合は，次に示すデフォルトのマッピング規則が適用されます。

- @Embeddable がアノテートされたクラスの場合，フィールド／プロパティは@Embedded 側のエンティティでの指定に従ってデータベースにマップされます。
- 永続化フィールド／永続化プロパティの型が次のどれかである場合，@Basic が定義されているのと同じ方法でマップされます。
  - Java のプリミティブ型
  - プリミティブ型のラッパ
  - java.lang.String
  - java.math.BigInteger
  - java.math.BigDecimal
  - java.util.Date
  - java.util.Calendar
  - java.sql.Date
  - java.sql.Time

- java.sql.Timestamp
- byte[]
- Byte[]
- char[]
- Character[]
- enum
- Serializable を実装した任意の型

なお，上記以外の型が指定されている場合の動作は保証しません。

8.13　エンティティクラスの継承方法

エンティティクラスの継承には次に示す特長があります。

- エンティティクラスは抽象クラス，具象クラスのどちらでも使用できます。抽象クラスおよび具象クラスともに，@Entity が定義できます。また，エンティティとしてマップでき，抽象クラスおよび具象クラスの両方に対してクエリを発行できます。
- エンティティクラスはほかのエンティティクラスから継承できます。
- エンティティクラスは非エンティティクラスを継承できます。また，非エンティティクラスはエンティティクラスを継承できます。

ここでは，エンティティクラスの継承クラスの種類と継承マッピング戦略について説明します。

8.13.1　継承クラスの種類

エンティティクラスの継承クラスの種類には，抽象エンティティクラス，マップドスーパークラス，および非エンティティクラスがあります。それぞれ説明します。

(1)　抽象エンティティクラス

抽象クラスはエンティティクラスとして定義できます。抽象エンティティクラスは，直接インスタンスを作成できない点が具象エンティティクラスとは異なります。抽象エンティティクラスは，エンティティとしてマップすることも，サブクラスのエンティティを操作・取得するためにクエリの対象に指定することもできます。

サブクラスでは，プロパティのアクセサメソッドをオーバーライドできます。ただし，永続化フィールドおよび永続化プロパティの O/R マッピング情報をサブクラスでオーバーライドした場合，動作は保証しません。サブクラスで O/R マッピング情報をオーバーライドする場合は，@AssociationOverride や @AttributeOverride などを使用してください。

抽象エンティティクラスは，@Entity または O/R マッピングファイルで指定します。

(2)　マップドスーパークラス

マップドスーパークラスとは，エンティティクラスのスーパークラスになることができるクラスです。永続化フィールドやマッピング情報を定義でき，さらにこれらを継承するような構成にできます。

マップドスーパークラスは，特定のテーブルを指定できないため，@Table の指定はできません。そのため，マップドスーパークラスをエンティティにすることはできません。マップドスーパークラスはエンティティクラスとは異なり，問い合わせができないため，EntityManager やクエリの操作の引数に渡すこともできません。また，リレーションシップの対象にすることもできません。

マップドスーパークラスは，抽象クラスとしても，具象クラスとしても定義できます。クラスをマップドスーパークラスとして定義する場合は，@MappedSuperclass，または O/R マッピングファイルに<mapped-superclass>タグを定義します。マッピング情報は，継承されたエンティティクラスに対して提供されます。

マップドスーパークラスとして設計したクラスは，マッピングがサブクラスにだけ提供できるマッピングであることを除いて，エンティティクラスと同様の方法でマップできます。

エンティティがサブクラスとして適用されると，マップドスーパークラスで指定された情報が，サブクラスに継承されます。@AssociationOverride や O/R マッピングファイルで対応する XML 要素を使用して，マッピング情報をサブクラス上でオーバーライドすることができます。

(3) エンティティ継承階層構造の非エンティティのクラス

エンティティクラスは非エンティティクラスのスーパークラスを持つことができます。スーパークラスは具象クラスおよび抽象クラスとして定義できます。

非エンティティクラスのスーパークラスには次に示す特長があります。

- 振る舞いだけを継承します。
- 状態は永続ではありません。
- 継承したすべての状態は，継承したエンティティクラスでも永続ではありません。
- 永続でない状態は，EntityManager の制御の対象ではありません。
- スーパークラスのアノテーションは無視されます。

非エンティティクラスを EntityManager や Query インタフェースのメソッドの引数にしたり，マッピング情報にしたりしないでください。CJPA プロバイダの場合，アプリケーションの実行時に例外が発生します。

8.13.2 継承マッピング戦略

エンティティを継承した場合，クラス階層をテーブルにマッピングする方法を継承マッピング戦略として指定できます。継承マッピング戦略は，@Inheritance または O/R マッピングファイルの<entity>タグの<inheritance>タグを使用して指定します。

継承マッピング戦略には次の 3 種類があります。

- SINGLE TABLE 戦略

  SINGLE TABLE 戦略は，エンティティクラスの継承階層にあるすべてのクラスが一つのテーブルにマップする戦略方法です。

- JOINED 戦略

  JOINED 戦略は，クラス階層の最上位は単一のテーブルにマップする戦略方法です。

- TABLE PER CLASS 戦略

  エンティティクラスのクラス階層中の各クラスを別々のテーブルにマップする戦略方法です。

ただし，CJPA プロバイダでは，TABLE PER CLASS 戦略は使用できません。CJPA プロバイダを使用している場合で，TABLE PER CLASS 戦略を指定したときには，アプリケーションの起動時に例外が発生します。

注意事項

CJPA プロバイダを使用する場合，次のことに注意してください。

- クラス階層中で継承戦略を複数組み合わせて指定することはできません。また，複数の継承戦略を組み合わせているかどうかのチェックも実施しません。指定した場合の動作は保証しません。
- @DiscriminatorColumn で指定したカラムをエンティティのフィールドに定義した場合，persist 操作をしてコミットしてもエンティティのフィールドに設定した値はデータベースに反映されません。@DiscriminatorValue で指定した値またはそのデフォルト値が反映されます。また，コミット後には，コミット前にフィールドに設定されていた値が格納されたままとなるため注意が必要です。

これらは javax.persistence.DiscriminatorType 列挙型の値で指定されます。それぞれの戦略について説明します。

(1) SINGLE TABLE 戦略

SINGLE TABLE 戦略とは，エンティティクラスの継承階層にあるすべてのクラスが一つのテーブルにマップする戦略方法です。そのため，テーブルではクラスを識別するためのカラムとして，識別カラムを持つ必要があります。

識別カラムは@DiscriminatorColumn，または O/R マッピングファイルで指定します。なお，デフォルトの識別カラム名は「DTYPE」です。データベースに識別カラムがない場合，アプリケーションの実行時に例外が発生します。

識別カラムに格納される値を指定したい場合には，@DiscriminatorValue または O/R マッピングファイルの<entity>タグ下の<discriminator-value>タグを使用して指定します。

注意事項

SINGLE TABLE 戦略を利用する場合，サブクラスのフィールドに対応するテーブルのカラムでは，null 値を指定できるようにする必要があります。

(2) JOINED 戦略

JOINED 戦略とは，クラス階層の最上位は単一のテーブルにマップする戦略方法です。各サブクラスは，スーパークラスから継承されていないサブクラス特有のフィールドと，スーパークラス表の主キーの外部キーとして機能する主キー列を持つ別々のテーブルで示されます。

JOINED 戦略の場合も，SINGLE TABLE の場合と同様にスーパークラスがマップされるテーブルに識別カラムを持つ必要があります。

注意事項

JOINED 戦略を利用する場合，サブクラスのインスタンスの生成で複数回の結合の実行が必要になります。このため，階層構造が深くなると，性能が劣化するおそれがあります。また，クラス階層にわたる範囲でクエリを発行する場合，JOIN が必要です。

8.14　EntityManager および EntityManagerFactory の使用方法

ここでは，アプリケーションから利用する EntityManager および EntityManagerFactory の使用方法について説明します。

8.14.1　EntityManager でのエンティティのライフサイクル管理

EntityManager はデータベースに対して次の操作をするためのインタフェースを持つオブジェクトです。

- エンティティを登録したり，削除したりする。
- プライマリキーによってエンティティを検索する。
- エンティティをわたったクエリを発行する。

EntityManager に対してエンティティを登録すると，トランザクションのコミットなどの適切なタイミングでエンティティの状態がデータベースで永続化されます。

また，EntityManager はエンティティの集合を表す永続化コンテキストと関連を持ちます。EntityManager にエンティティが登録されると，エンティティは特定の永続化コンテキストに属します。また，EntityManager は，エンティティのライフサイクルを管理します。

EntityManager で管理するエンティティの集合は永続化ユニットという単位で定義します。永続化ユニットはアプリケーションの設定ファイルである persistence.xml で定義します。

永続化コンテキストと永続化ユニットの注意事項について説明します。

- 永続化コンテキスト内ではエンティティは一意となるようにします。このため，同一の永続化コンテキスト内では，データベースの同じ行を表すエンティティは一つとしてください。なお，永続化コンテキストが異なる場合は，データベースの同じ行を表すエンティティを複数持つことができます。この場合のデータベース上での排他方法については，「8.10　楽観的ロック」または「8.11　JPQL での悲観的ロック」を参照してください。
- 永続化ユニットはそれぞれ単一のデータベースにマッピングされます。定義の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「5.8.2(3)　<jta-data-source>タグ，<non-jta-data-source>タグ」を参照してください。

8.14.2　EntityManager および EntityManagerFactory の設定方法

アプリケーションで利用する EntityManager および EntityManagerFactory は，アノテーションまたは DD で設定します。

- EntityManagerFactory の設定
  - アノテーションの場合：@PersistenceUnit で設定します。

  - DD の場合：<persistence-unit-ref>タグで設定します。
- EntityManager の設定
  - アノテーションの場合：@PersistenceContext で設定します。
  - DD の場合：<persistence-context-ref>タグで設定します。

なお，アノテーションの詳細については「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」，DD に設定するタグの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

8.14.3　EntityManager の API に関する注意事項

EntityManager が提供する API についての注意事項を次に示します。なお，EntityManager の API については「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

- トランザクションスコープ永続化コンテキストの EntityManager が使用されるとき，persist，merge，remove，refresh メソッドは，トランザクションコンテキスト内で実行する必要があります。トランザクションコンテキストがない場合は，javax.persistence.TransactionRequiredException がスローされます。
- find メソッドと getReference メソッドは，トランザクションコンテキストでの実行を要求しません。このため，トランザクションスコープ永続化コンテキストの EntityManager が使用された場合，結果のエンティティは detached 状態になります。また，拡張された永続化コンテキストの EntityManager が使用された場合，結果のエンティティは managed 状態になります。
- createQuery メソッドの引数が有効な JPQL の文字列でなければ，IllegalArgument 例外を送出し，クエリの実行が失敗します。
- 実行するネイティブクエリが，接続先のデータベースの仕様に合わない場合，または定義された結果のセットがクエリの結果と互換性がない場合，クエリの実行は失敗し，クエリの実行時に PersistenceException 例外がスローされます。
- EntityManager インタフェースのメソッドからランタイム例外が送出されると，カレントのトランザクションはロールバックにマークされます。
- EntityManager から取得した Query オブジェクト，および EntityTransaction オブジェクトは，EntityManager がオープンしている間は有効です。

8.15　コールバックメソッドの指定方法

エンティティのライフサイクルを受け取るために，メソッドをライフサイクルのコールバックメソッドに指定できます。コールバックメソッドとして定義されたメソッドは永続化に関するライフサイクルイベントに対応して呼び出されます。

ここでは，コールバックメソッドの指定個所，実装方法，および呼び出し順序について説明します。

8.15.1　コールバックメソッドの指定個所

コールバックメソッドは，次の場所に指定できます。

- エンティティクラスまたはマップドスーパークラス内
- エンティティクラスまたはマップドスーパークラスに関連づけられたエンティティリスナクラス

エンティティリスナクラスとは，コールバックメソッドを実装するための専用のクラスです。エンティティリスナクラスを使用すると，コールバックメソッドの実装部分を分離できます。

コールバックメソッドは，アノテーションまたは O/R マッピングファイルで指定します。ただし，デフォルトのコールバックメソッドは，O/R マッピングファイルで指定します。アノテーションでは指定できません。なお，デフォルトのコールバックメソッドとは，永続化ユニット内のすべてのエンティティに適用されるエンティティリスナを指します。

コールバックリスナの指定方法について説明します。

(1)　アノテーションでのコールバックメソッドの指定

コールバックメソッドの指定にアノテーションを使用する場合は，次の表にあるアノテーションをメソッドに設定してください。ライフサイクルイベントに応じてメソッドを呼び出すことができます。

表 8-19　アノテーションを使用したコールバックメソッドの指定

アノテーション	実行される内容
@PostLoad	エンティティが永続化コンテキストにロードされたあとか，または refresh 操作が適用されたあとに，コールバックメソッドが実行されます。キャッシュからエンティティを読み込んだあと，またはデータベースに SELECT 文を発行したタイミングで実行されます。
@PrePersist @PreRemove	EntityManager がエンティティの persist または remove 操作を行う前にコールバックメソッドが呼び出されます。マージ操作が適用され，新しく managed 状態のインスタンスが作成される場合，managed 状態のインスタンスへの PrePersist コールバックメソッドは，エンティティの状態がコピーされたあとに呼び出されます。これらの PrePersist または PreRemove のコールバックは操作がカスケードされるすべてのインスタンスに対しても呼び出されます。PrePersist または PreRemove メソッドは常に persist，merge，または remove 操作の一部として同期を取って呼び出されます。

アノテーション	実行される内容
@PostPersist @PostRemove	PostPersist および PostRemove コールバックメソッドは，エンティティが persist または remove 操作で，永続化されたあとまたは削除されたあとで呼び出されます。 これらのコールバックは操作がカスケードされたすべてのエンティティに対しても呼び出されます。PostPersist または PostRemove メソッドは，それぞれデータベースの insert または delete 操作が行われたあとに呼び出されます。これらのデータベースへの操作は persist，merge，もしくは remove 操作の直後，または flush メソッドが呼び出されたあとになります。ただし，トランザクションの終了時になることもあります。生成されたプライマリキーは PostPersist メソッドで利用できます。
@PreUpdate @PostUpdate	PreUpdate および PostUpdate のコールバックはそれぞれエンティティデータのデータベースへの update 操作の前後に呼び出されます。 これらのデータベースへの操作はエンティティの状態が更新された場合，または状態がデータベースにフラッシュされた場合に実行されます。ただし，データベースへの操作は，トランザクションの終了時になることもあります。一つのトランザクション内で，エンティティを永続化してから更新したり，エンティティを更新してから削除したりした場合には，PreUpdate や PostUpdate のコールバックが発生しないことがあります。

エンティティリスナクラスを利用する場合，エンティティに対して@EntityListener を指定してエンティティリスナクラスを指定する必要があります。指定方法の例を次に示します。

```
@Entity
@EntityListeners(CallbackListener.class)
public class Employee implements Serializable{
. . .
}
```

(2) O/R マッピングファイルでのコールバックリスナの指定

O/R マッピングファイルを使用してコールバックメソッドを指定する場合は次のように指定します。

- エンティティリスナのクラスおよびそのクラスのコールバックメソッドを指定する場合，O/R マッピングファイルの<entity-listener>タグを使用します。ライフサイクルリスナメソッドは，<entity-listener>タグ下の，<pre-persist>タグ，<post-persist,>タグ，<pre-remove>タグ，<post-remove>タグ，<pre-update>タグ，<post-update>タグ，および<post-load>タグを使用して指定します。
- エンティティリスナクラスのコールバックメソッドを指定する場合は，コールバックイベントごとに最大一つのメソッドを<entity-listener>タグ以下のタグを使用して指定できます。
- <persistence-unit-defaults>タグの<entity-listeners>タグの下位タグに対して，O/R マッピングファイルの<entity-listener>タグを指定するとデフォルトコールバックメソッドが指定できます。
- <entity>タグまたは<mapped-subclass>タグにある<entity-listeners>タグの下位タグに，<entity-listener>タグを指定すると，エンティティまたはマップドスーパークラスとそのサブクラスに対するコールバックリスナの指定になります。
- <entity-listeners>タグに指定したリスナの順番でコールバックリスナは呼び出されます。リスナの呼び出し順序については，「8.15.3　コールバックメソッドの呼び出し順序」を参照してください。

8.15.2　コールバックメソッドの実装

ユーザは必要に応じてコールバックメソッドを実装します。エンティティクラスやマップドスーパークラス内に実装するコールバックメソッドとエンティティリスナクラスのコールバックメソッドでコールバックメソッドのシグネチャは異なります。

エンティティクラスやマップドスーパークラスで定義されるコールバックメソッドは次に示すシグネチャになります。

```
void <METHOD>
```

また，エンティティリスナクラスで定義されるコールバックメソッドは次に示すシグネチャになります。

```
void <METHOD>(Object)
```

引数の Object には，コールバックメソッドが実行されるエンティティのインスタンスを指定します。

(1)　コールバックメソッド使用時の注意事項

コールバックメソッドについては，次に示す注意事項があります。これらの条件を満たさない場合，アプリケーションの開始時に例外が発生して，アプリケーションの開始に失敗します。

- public で引数がないコンストラクタを持たなければなりません。
- コールバックメソッドでは public，private，protected，およびパッケージレベルのアクセスができます。ただし，static や final は使用できません。
- 一つのクラスが同じライフサイクルイベントに対して複数のライフサイクルコールバックメソッドを持つことはできません。ただし，同じメソッドが複数のコールバックイベントで使用されることがあります。

(2)　コールバックメソッドに適用されるルール

コールバックメソッドについては次に示すルールが適用されます。

- コールバックメソッドでは，未チェックまたは実行時例外の送出が許可されています。トランザクション中に実行したコールバックメソッドでスローされた実行時例外は，トランザクションをロールバックさせます。コールバックメソッドが複数指定されている場合，実行時例外がスローされたあとには残りのコールバックメソッドは実行されません。
- コールバックメソッドでは，JNDI，JDBC，JMS，Enterprise Bean を実行できます。
- コールバックメソッドで次の操作をしないでください。
  - EntityManager を呼び出す。
  - クエリ操作を実行する。
  - ほかのエンティティインスタンスにアクセスする。

- リレーションシップを更新する。

このような方法で使用した場合，動作を保証しません。

- Java EE 環境でコールバックメソッドが呼び出された場合，エンティティのコールバックリスナは呼び出すコンポーネントのネーミングコンテキストを共有します。また，エンティティのコールバックメソッドは，コールバックメソッドが呼び出された時点の呼び出し元コンポーネントのトランザクションとセキュリティコンテキストで呼び出されます。

8.15.3 コールバックメソッドの呼び出し順序

エンティティに対して複数のコールバックメソッドを定義した場合，呼び出し順序は次の規則に従います。

1. O/R マッピングファイルに定義した順番でデフォルトリスナが呼び出されます。
   明示的に@ExcludeDefaultListeners または O/R マッピングファイルの<exclude-default-listeners>タグを指定しないかぎり，デフォルトリスナは永続化ユニット内のすべてのエンティティに適用されます。
2. @EntityListeners で指定された順番にコールバックメソッドが呼び出されます。
   なお，O/R マッピングファイルを使用すると，次の操作ができます。
   - エンティティに対するコールバックメソッドの呼び出し順序を指定する。
   - アノテーションでの指定順序をオーバーライドする。
3. エンティティ（またはマップドスーパークラス）に指定されたコールバックメソッドが呼び出されます。

(1) 継承階層内での呼び出し順序

エンティティクラスやマップドスーパークラスの継承階層の中で複数回のエンティティリスナを定義した場合，呼び出し順序は次のようになります。

1. デフォルトコールバックリスナがあれば，最初に呼び出されます。
2. エンティティリスナクラスのコールバックメソッドがスーパークラスに指定されているリスナから順番に呼び出されます。このとき，@EntityListener などの指定があれば，その順序に従います。
3. すべてのエンティティリスナのコールバックメソッドが呼び出されたあとで，エンティティ（またはマップドスーパークラス）に定義されたコールバックメソッドがスーパークラスに指定されているリスナから順番に呼び出されます。

コールバックメソッドをサブクラスでオーバーライドした場合には，オーバーライドされたメソッドは呼び出されません。オーバーライドしたコールバックメソッドが異なるライフサイクルのイベントを指定している場合，またはライフサイクルコールバックメソッドではない場合，オーバーライドされた側のメソッドは呼び出されます。また，メソッドのコールバックメソッドの設定はオーバーライドされます。

(2) コールバックメソッドの除外について

- @ExcludeDefaultListeners または O/R マッピングファイルの<exclude-default-listeners>タグを指定すると，デフォルトエンティティリスナはエンティティクラス（または mapped superclass）とそのサブクラスでは呼び出されません。
- @ExcludeSuperclassListeners または O/R マッピングファイルの<exclude-superclass-listeners>タグがエンティティクラスや mapped superclass に適用されると，そのクラスとそのサブクラスではリスナのコールバックメソッドは呼び出されません。@ExcludeSuperclassListeners または O/R マッピングファイルの<exclude-superclass-listeners>タグは，デフォルトのエンティティリスナの呼び出しを除外することにはなりません。
- O/R マッピングファイルを使用して，entity や mapped superclass への除外されたデフォルト/スーパークラスリスナを明示的に指定すると，エンティティやそのサブクラスに適用されることになります。

8.16　クエリ言語を利用したデータベースの参照および更新方法

クエリとは，データベースに対する処理要求（問い合わせ）を文字列として表したものです。データベース上のデータの検索，更新，削除などの命令をシステムに発行するときに使用します。クエリを実行するには，javax.persistence.Query インタフェースを使用します。クエリには，JPQL とネイティブクエリの 2 種類があります。ここでは，クエリを利用したデータベースの参照および更新方法について説明します。

8.16.1　JPQL でのデータベースの参照および更新方法

JPQL は，データベースを検索・更新したり，データベースが持っている集合関数などの機能を利用したりするためのクエリ言語です。SQL がテーブルを対象としたクエリ言語であるのに対して，JPQL はエンティティクラスを対象とした JPA 仕様で定義されているクエリ言語です。

クエリはアノテーションまたは O/R マッピングファイルで定義できます。クエリと同じ永続化ユニットでエンティティが定義されている場合，エンティティの集合を表す抽象スキーマ型をクエリで使用できます。また，パス式を使用すると，永続化ユニット内で定義されたリレーションシップをわたってクエリを使用できます。パス式の詳細については，「8.17.4(2)　パス式」を参照してください。

アプリケーションに JPQL を記述して実行すると，次の順序で接続先のデータベースに対して SQL が発行されます。

1. JPQL が実行されると，CJPA プロバイダによって JPQL の内容が解釈される。
2. 対象となるエンティティクラス内に記述されたアノテーションや O/R マッピングファイルの情報を基に，接続先のデータベース製品固有の SQL 文に組み立てて発行する。

ここでは，JPQL の使用方法について説明します。

(1)　Query オブジェクトの取得方法

JPQL を使用して Query オブジェクトを取得するためには，CJPA プロバイダが提供する次の EntityManager インタフェースのメソッドを使用します。EntityManager インタフェースのメソッドについて説明します。

(a)　Query createQuery(String JPQL 文)

createQuery の記述例を次に示します。引数には実行する JPQL 文を指定します。

```
Query q = em.createQuery(
                  ”SELECT c  ” +
                  ”FROM Customer c ” +
                  ” WHERE c.name LIKE ”Smith”);
```

(b) Query createNamedQuery(String クエリ名)

あらかじめ名前を付けて定義しておくことのできるクエリを名前付きクエリといいます。名前付きクエリは，@NamedQuery を任意のエンティティクラスに付与して定義します。@NamedQuery の name 属性にクエリ名を指定して，query 属性には JPQL 文を指定します。

CJPA プロバイダの場合，同じ名称の名前付きクエリを複数指定することはできません。同じ名称の名前付きクエリを複数指定した場合には，警告メッセージ KDJE55535-W を出力します。CJPA プロバイダで指定した場合，どのクエリが動作するかは保証しません。

次に，@NamedQuery の定義例を示します。この例では，@NamedQuery を使用してあらかじめ findAllCustomersWithName という名前でクエリを登録しています。アプリケーションの createNamedQuery メソッドに登録した名前付きクエリ名を渡すことで，事前に登録されているクエリを取得して利用します。

- @NamedQuery でのクエリ名の登録

```
@NamedQuery(
      name="findAllCustomersWithName",
      query="SELECT c FROM Customer c WHERE c.name LIKE :custName"
)
@Entity
public class Customer {
・・・
}
```

- createNamedQuery メソッドでの名前付きクエリの記述例

```
@Stateless
public class MySessionBean {
・・・
  @PersistenceContext
  public EntityManager em;
   ・・・
  public void doSomething() {
     ・・・
    Query q = em.createNamedQuery("findAllCustomersWithName")
                                      .setParameter("custName", "Smith");
  }
}
```

なお，同一の永続化ユニット内では，ほかのエンティティで定義した名前付きクエリを使用することもできます。

(2) パラメタの指定方法

JPQL では，クエリを生成する際に，WHERE 節に記述する条件式にパラメタを使用して，動的に値を設定することができます。パラメタの値は Query インタフェースの setParameter メソッドで設定します。パラメタには，位置パラメタと名前付きパラメタがあります。それぞれについて説明します。

位置パラメタ

WHERE 節の中のパラメタを組み込みたい位置に，「?」と数値の組み合わせを記述します。パラメタの値は，Query インタフェースの setParameter メソッドで設定します。位置パラメタの記述形式と記述例を次に示します。

- 記述形式

```
Query setParameter(int 位置, Object 値)
```

- 記述例

```
Query q = em.createQuery(
        ”SELECT c FROM Customer c WHERE c.balance < ?1”)
.setParameter(1, 20000);
```

名前付きパラメタ

WHERE 節の中のパラメタを組み込みたい位置に，「:」と任意の文字列（ただし，0～9 の文字を除く）の組み合わせを記述します。パラメタの値は，Query インタフェースの setParameter メソッドで設定します。名前付きパラメタの記述形式と記述例を次に示します。

- 記述形式

```
Query setParameter(String パラメタ名, Object 値)
```

パラメタ名の先頭には「:」を指定しないでください。また，名前付きパラメタは大文字・小文字を区別します。

- 記述例

```
Query q = em.createQuery(
        ”SELECT c FROM Customer c WHERE c.name LIKE :custName”)
        .setParameter(”custName”, ”John”);
```

なお，パラメタを使用するときには次のことに注意してください。

- 一つのクエリで位置パラメタと名前付きパラメタを混在して使用しないでください。CJPA プロバイダの場合，混在して使用したときの動作は保証しません。
- setParameter メソッドには，java.util.Date 型または java.util.Calendar 型のオブジェクトをパラメタ値として設定するときに，次のように指定できます。なお，パラメタ値の時間タイプは TemporalType 列挙型で指定する必要があります。詳細については Java のドキュメントを参照してください。
  - Query setParameter(int 位置, Date 日付, TemporalType 時間タイプ)
  - Query setParameter(String パラメタ名, Date 日付, TemporalType 時間タイプ)
  - Query setParameter(int 位置, Calendar カレンダ, TemporalType 時間タイプ)
  - Query setParameter(String パラメタ名, Calendar カレンダ, TemporalType 時間タイプ)

(3) クエリ結果の取得および実行

生成したクエリを実行して，そのクエリ結果を返したり，更新クエリを実行したりするためには，Query インタフェースの次のメソッドを使用します。それぞれのメソッドについて説明します。

(a) Object getSingleResult()

このメソッドは，クエリ結果を単一のオブジェクトとして返す場合に使用します。

メソッドを実行するとデータを検索します。検索の結果，ヒットした単一の行をエンティティオブジェクトに格納し，Object 型で返します。Object 型の戻り値は対象のエンティティクラスにキャストする必要があります。

複数の行がヒットした場合は，NonUniqueResultException 例外が発生します。ヒットする行がなかった場合は，NoResultException 例外が発生します。

(b) List getResultList()

このメソッドは，クエリ結果をリストとして返す場合に使用します。

メソッドを実行するとデータを検索します。検索の結果，ヒットした複数の行をエンティティオブジェクトに格納し，リストで返します。実行結果として複数の行が返されること想定しているため，ヒットする行が一つもなかった場合は，空のリストが返ります。

(c) int executeUpdate()

このメソッドは，更新クエリを実行する場合に使用します。

メソッドを実行するとテーブルから複数の行を一斉に削除または更新するクエリを実行します。実行結果にはヒットした行の件数が返ります。

8.16.2 ネイティブクエリでのデータベースの参照および更新方法

CJPA プロバイダでは，JPQL 以外のクエリ言語として，データベース固有のネイティブクエリを直接記述して，データベースの参照・更新などを実行できます。

ここでは，ネイティブクエリでの使用方法について説明します。

(1) Query オブジェクトの取得方法

ネイティブクエリを使用して Query オブジェクトを取得するためには，次に示す CJPA プロバイダが提供する EntityManager インタフェースのメソッドを使用します。

(a) Query createNativeQuery(String SQL 文)

createNativeQuery の記述例を次に示します。引数には実行するネイティブクエリを指定します。

```
Query q = em.createNativeQuery(
                    ”SELECT o.id, o.quantity, o.item ” +
                    ”FROM Order o, Item i ” +
                    ”WHERE (o.item = i.id) AND (i.name = ’widget’)”);
```

(b)　Query createNativeQuery(String SQL 文，Class 結果格納クラス)

createNativeQuery の記述例を次に示します。第 1 引数には実行するネイティブクエリ，第 2 引数には実行結果を格納するクラスオブジェクトを指定します。

```
Query q = em.createNativeQuery(
                    ”SELECT o.id, o.quantity, o.item ” +
                    ”FROM Order o, Item i ” +
                    ”WHERE (o.item = i.id) AND (i.name = ’book’)”,
                    com.hitachi.Order.class);
```

この例の場合，クエリが実行されると「book」という名前のアイテムに対するすべての Order エンティティのコレクションを返します。

なお，SELECT 節で指定したクエリの結果と引数に指定したクラスオブジェクトの整合性がない場合は例外が発生します。

(c)　Query createNativeQuery(String SQL 文，String 結果セットマッピング名)

第 1 引数には実行するネイティブクエリ，第 2 引数には実行結果を格納する結果セットマッピング名を指定します。結果セットマッピングは，@ SqlResultSetMapping で指定します。なお，結果セットマッピングについては，「(2)　結果セットマッピング」を参照してください。

@SqlResultSetMapping の定義例と createNativeQuery の記述例を次に示します。

- @SqlResultSetMapping の定義例

```
@SqlResultSetMapping(name=”BookOrderResults”,
      entities=@EntityResult(entityClass=com.hitachi.Order.class))
```

- createNativeQuery の記述例

```
Query q = em.createNativeQuery(
                    ”SELECT o.id, o.quantity, o.item ” +
                    ”FROM Order o, Item i ” +
                    ”WHERE (o.item = i.id) AND (i.name = ’book’)”,
                    ”BookOrderResults”);
```

この例の場合，クエリが実行されると「book」という名前のアイテムに対するすべての Order エンティティのコレクションを返します。@SqlResultSetMapping を使用することで，「8.16.1(1)　Query オブジェクトの取得方法」の@NamedQuery を使用した場合の記述例と同じ結果を得ることができます。

なお，SELECT 節で指定したクエリ結果と引数に指定した@SqlResultSetMapping 設定の整合性がない場合は例外が発生します。

(d) Query createNamedQuery(String クエリ名)

ネイティブクエリの場合も，JPQL と同じように createNamedQuery メソッドを使用できます。ネイティブクエリの場合は，引数に名前付きネイティブクエリ名を指定してください。

名前付きネイティブクエリは，@NamedNativeQuery を任意のエンティティクラスに付与して定義します。引数のクエリ名には，@NamedNativeQuery の name 属性で指定した名前を使用します。

CJPA プロバイダの場合，同じ名称の名前付きクエリを複数指定することはできません。同じ名称の名前付きクエリを複数指定した場合には，警告メッセージ KDJE55522-W を出力します。CJPA プロバイダで指定した場合，どのクエリが動作するかは保証しません。

次に，createNamedQuery メソッドの使用例を示します。この例では，@NamedNativeQuery を使用してあらかじめ findBookOrder という名前でクエリを登録しています。アプリケーションの createNamedQuery メソッドに登録した名前付きクエリ名を渡すことで，事前に登録されているクエリを取得して利用します。

- @NamedNativeQuery でのクエリ名の登録

```
@NamedNativeQuery( name="findBookOrder",
       query="SELECT o.id, o.quantity, o.item " +"
             "FROM Order o, Item i " +
             "WHERE (o.item = i.id) AND (i.name = 'book')")
)
@Entity
public class Order {
 . . .
}
```

- createNamedQuery メソッドでの名前付きネイティブクエリの記述例

```
@Stateless
public class MySessionBean {
   . . .
  @PersistenceContext
  public EntityManager em;
   . . .
  public void doSomething() {
     . . .
    Query q = em.createNamedQuery("findBookOrder ")
  }
}
```

なお，同一の永続化ユニット内では，ほかのエンティティで定義した名前付きネイティブクエリを使用することもできます。

(2) 結果セットマッピング

結果セットマッピングとは，ネイティブクエリの実行結果を任意のエンティティクラスにマッピングして受け取ったり，スカラ値で受け取ったりするための機能です。

結果セットマッピングでは，ネイティブクエリの実行結果として取得した各カラム値のマッピング情報を@SqlResultSetMapping を指定して任意のエンティティクラスに対して付与します。

(a) @SqlResultSetMapping の記述形式

@SqlResultSetMapping の記述形式を次に示します。

```
@SqlResultSetMapping(
  name= 結果セットマッピングの名前,
  entities= 結果をマッピングするためのエンティティクラス指定(@EntityResultの配列),
  columns= 結果をマッピングするためのカラム指定(@ColumnResultの配列）)
```

name 属性

結果セットマッピング名を指定します。

entities 属性

@EntityResult の配列を指定します。@EntityResult の記述形式を次に示します。

```
@EntityResult(
    entityClass= 結果をマッピングするためのクラスを指定,
    fields= 結果をマッピングするためのフィールド指定(@FieldResultの配列）)
```

@EntityResult の entityClass 属性には，カラム値を格納するエンティティクラスを指定します。また，field 属性には，@FieldResult の配列を指定します。

@EntityResult の記述形式を次に示します。

```
@FiledResult(
    name= クラスの永続プロパティ(またはフィールド)の名前,
    column= SELECT節のカラムの名前(または別名）)
```

@FieldResult の name 属性には，@EntityResult の entityClass 属性に指定したエンティティクラスの永続化フィールド名を指定します。また，column 属性にはカラム名を指定します。

columns 属性

columns 属性は，エンティティクラスに格納しないでスカラ値として受け取るために，@ColumnResult の配列を指定します。スカラ値を取り出す必要がない場合は指定する必要はありません。
@ColumnResult の name 属性には，値を取り出すカラム名を指定してください。@ColumnResult の記述形式を次に示します。

```
@ColumnResult(
    name= SELECT節のカラムの名前(または別名）)
```

なお，カラム名は AS で指定したエリアス名でも指定できます。SELECT 節に同じ名前の複数の列を含む場合は，列の別名を使用してください。

(b) 使用例

従業員テーブル（Employee）と部門テーブル（Department）から従業員番号 12003 のクエリ結果を任意のエンティティクラス（EmployeeSetmap）にマッピングして，スカラ値（EMP_MONTHLY_SALARY カラム）を受け取る例を次に示します。

結果セットマッピング名（NativeQuerySetMap）を createNativeQuery の第 2 引数に指定して，結果セットマッピングを実行します。

- 結果セットマッピングを使用したネイティブクエリの記述例

```
query = em.createNativeQuery(
                ”SELECT e.EMPLOYEE_ID AS EMP_EMPLOYEE_ID, ” +
                ”e.EMPLOYEE_NAME AS EMP_EMPLOYEE_NAME, ” +
                ”d.DEPARTMENT_NAME AS DEP_DEPARTMENT_NAME, ” +
                ”e.MONTHLY_SALARY AS EMP_MONTHLY_SALARY  ” +
                ”FROM EMPLOYEE e, DEPARTMENT d ” +
                ”WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID ” +
                ”AND e.EMPLOYEE_ID = 12003”,
                ”NativeQuerySetMap”);
```

- ネイティブクエリの実行結果を格納する任意のエンティティクラス

```
@Entity
public class EmployeeSetmap implements Serializable {
:
@Id
public int getEmployeeId() { return employeeId; }
public String getEmployeeName() { return employeeName; }
public String getDepartmentName() { return departmentName; }
:
}
```

- @ SqlResultSetMapping の記述例

```
@SqlResultSetMapping(
       name=”NativeQuerySetmap”,
       entities={ @EntityResult(
                        entityClass=EmployeeSetmap.class,
                        fields={ @FiledResult(
                                    name=”employeeId”,
                                    column=”EMP_EMPLOYEE_ID”),
                                @FiledResult(
                                    name=”employeeName”,
                                    column=”EMP_EMPLOYEE_NAME”),
                                @FiledResult(
                                    name=”departmentName”,
                                    column=”DEP_DEPARTMENT_NAME”) } ) },
       columns={ @ColumnResult(
                        name=”EMP_MONTHLY_SALARY”) }
```

なお，@ SqlResultSetMapping を実行してネイティブクエリを実行した結果の Object 型配列は，次のようになります。

Object[0]

EmployeeSetmap クラスのオブジェクト

（EMP_EMPLOYEE_ID カラム，EMP_EMPLOYEE_NAME カラム，DEP_DEPARTMENT_NAME カラムの値が各フィールドに格納される）

Object[1]
　EMP_MONTHLY_SALARY カラムの値

(3)　パラメタの指定方法

ネイティブクエリは，JPQL と同様に，パラメタによって動的に値を設定することができます。WHERE 節の中のパラメタを組み込みたい位置に，「?」と数値の組み合わせを記述します。パラメタの値は，Query インタフェースの setParameter メソッドで設定します。ただし，ネイティブクエリでは，JPQL の名前付きパラメタは使用できません。

パラメタの記述形式を次に示します。

```
Query setParameter(int 位置, Object 値)
```

(4)　ネイティブクエリ結果の取得および実行

ネイティブクエリ結果の取得および実行は，JPQL と同様に，Query インタフェースの次のメソッドを使用します。

- Object getSingleResult()
- List getResultList()
- int executeUpdate()

これらのメソッドの詳細については，「8.16.1　JPQL でのデータベースの参照および更新方法」を参照してください。

8.16.3　クエリ結果件数の範囲指定

複数件あるクエリ結果の中から任意に指定した件数だけを取得したり，開始位置で指定したクエリ結果だけを取得したりすることができます。これらの情報を取得するには，Query インタフェースのメソッドを使用します。メソッドについて次に説明します。

(1)　setMaxResults メソッド

複数件あるクエリ結果の中から任意に指定した件数だけを取得するには Query インタフェースの setMaxResults メソッドを使用します。setMaxResults メソッドの記述形式を次に示します。

```
Query setMaxResults(int検索結果の最大数)
```

メソッドの引数には，検索結果の最大数を指定します。

(2) setFirstResult メソッド

開始位置で指定したクエリ結果だけを取得するには setFirstResult メソッドを使用します。setFirstResult メソッドの記述形式を次に示します。

```
Query setFirstResult(int検索結果の開始位置)
```

メソッドの引数に検索結果の開始位置を指定します。なお，開始位置は 0 から始まる数値を指定してください。

(3) Query インタフェースのメソッドの使用例

setMaxResults メソッドおよび setFirstResult メソッドの使用例を示します。この例では，従業員データ（Employee）から，月給（e.monthlySalary）の多い順に 10 番目から 5 人分の Employee オブジェクトを取得します。

```
Query query = em.createQuery( "SELECT e FROM Employee AS e " +
                              "ORDER BY e.monthlySalary DESC" )
                              .setFirstResult(9)
                              .setMaxResults(5);
List resultList = query.getResultList();
```

(4) 注意事項

setFirstResult メソッドを使用して開始位置を指定した場合，引数に指定した値によって，getResultList メソッドおよび getSingleResult メソッドを呼び出してから結果の値を取得するまでの時間が変わります。通常，引数に指定した開始位置の値に比例して，結果の値が戻ってくるまでの時間が掛かります。

8.16.4 フラッシュモードの指定

エンティティオブジェクトに対して実行された未コミット操作をクエリがどのように扱うかを指定できます。この指定をフラッシュモードの指定といいます。

フラッシュモードは，javax.persistence.Query インタフェースの setFlushMode メソッドで設定します。CJPA プロバイダでは，設定できる値は AUTO だけです。COMMIT は設定できません。

FlushModeType.AUTO の場合

トランザクション内でクエリが実行されたとき，クエリ結果に影響を及ぼす永続化コンテキストにあるすべてのエンティティの変更内容がクエリ結果に反映されます。

この設定は，EntityManager インタフェースの setFlushMode メソッドのフラッシュモードに関係なくクエリに適用されます。

8.16.5　クエリヒントの指定

クエリ実行時に，ベンダに依存するヒントとしてクエリヒントを指定できます。CJPA プロバイダでは，悲観的ロックを使用するときに指定します。

クエリヒントは次に示す場所に指定します。

- Query オブジェクトの setHint()メソッドの引数
- @NamedQuery の引数の@Hint
- O/R マッピングファイルの<named-query>タグの下位要素である<hint>タグ

クエリヒントに指定できる範囲以外の値が設定された場合は例外が発生します。なお，指定する値は大文字と小文字を区別しません。

例外が発生するタイミングは，クエリヒントが指定された個所によって異なります。次に例外発生のタイミングを示します。

- setHint()メソッドの場合，アプリケーションのクエリ実行時
- アノテーションの場合，デプロイ時
- O/R マッピングファイルの場合，アプリケーション開始時

CJPA プロバイダがサポートするクエリヒントについては，アノテーションを使用する場合は「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」，O/R マッピングファイルを使用する場合は「12.3　O/R マッピングファイル」を参照してください。

8.16.6　クエリの実行時の注意事項

ここでは，クエリ実行時の注意事項について説明します。

- setMaxResults メソッドまたは setFirstResult メソッドで，コレクション同士が FETCH JOIN を含むクエリを実行した場合，結果は保証されません。
- executeUpdate メソッド以外の Query メソッドは，トランザクション内で実行する必要はありません。特に getResultList と getSingleResult メソッドはトランザクション内で実行する必要はありません。
- トランザクションスコープ永続化コンテキストの EntityManager で，クエリが実行された場合の結果のエンティティは，detached 状態になります。拡張永続化コンテキストの EntityManager でクエリが実行された場合，すべて managed 状態になります。
- Query インタフェースのメソッドからスローされる NoResultException と NonUniqueResultException 以外の実行時例外はカレントのトランザクションをロールバックします。

8.17　JPQL の記述方法

ここでは，JPQL の記述方法について説明します。

8.17.1　JPQL の構文

JPQL 文には，SELECT 文と UPDATE 文，DELETE 文があります。

JPQL 文は，動的に指定したり，アノテーションや O/R マッピングファイルのタグで静的に定義したりすることができます。また，JPQL のすべての文でパラメタの指定ができます。

JPQL は型付き言語で，すべての式は型を持ちます。式の型は，式の構成や識別変数で定義された抽象スキーマ型，永続化フィールドとリレーションシップを評価する型，およびリテラル型で構成されています。なお，構文の文法については，「付録 G　JPQL の BNF」を参照してください。

注意事項

CJPA プロバイダの場合，BNF 構文に準拠しない JPQL を使用すると，例外が発生するおそれがあります。例外が発生しない場合でも，動作は保証しません。また，BNF 構文に準拠した JPQL を使用した場合でも，使用するデータベースで該当する機能をサポートしていない場合の動作は保証しません。

(1)　抽象スキーマ型

JPQL では，エンティティを対象にクエリを発行します。そのため，クエリでは対象となるエンティティの抽象スキーマを定義する必要があります。エンティティの抽象スキーマ型は，アノテーションまたは O/R マッピングファイルによって提供されるエンティティクラスと O/R マッピングの情報によって定義されます。

抽象スキーマ型とは，エンティティクラスのことを指します。エンティティクラスを構成するものとして，フィールドや，アノテーションなどの O/R マッピング情報があります。

エンティティの抽象スキーマ型は次に示すフィールドを持ちます。

- **ステートフィールド**
  ステートフィールドは，エンティティクラスの永続化フィールド，または永続化プロパティで，リレーションシップによる関連が存在しないフィールド，またはプロパティです。
- **関連フィールド**
  関連フィールドは，エンティティクラスでリレーションシップによって関連づいた永続化フィールド，または永続化プロパティです。リレーションシップが OneToMany または ManyToMany の場合，フィールドはコレクションになります。

(2) 抽象スキーマ名

JPQL では，エンティティを示すために，その抽象スキーマ型を指定する必要があります。抽象スキーマ型を示すための名前を抽象スキーマ名といいます。抽象スキーマ名は@Entity の name 属性（または O/R マッピングファイルの<entity>タグの name 属性）で定義します。name 属性が指定されなかった場合は，エンティティクラスの（パッケージ名抜きの）クラス名となります。

抽象スキーマ名は，永続化ユニット単位で固有となります。

(3) クエリのドメイン

クエリのドメインは，永続化ユニット内で定義されたすべてのエンティティの抽象スキーマ型を参照できます。抽象スキーマ型の中で定義されている関連フィールドによって，ほかの関連するエンティティの抽象スキーマ型を参照できます。

8.17.2 SELECT 文

SELECT 文には次の節があります。

- SELECT 節
  検索するオブジェクトの型か集合関数による値を指定します。SELECT 節は必ず指定します。
- FROM 節
  検索が適用される範囲を指定します。FROM 節は必ず指定します。
- WHERE 節
  検索結果を絞るために使用します。WHERE 節は省略できます。
- GROUP BY 節
  検索結果をグループ化するために使用します。GROUP BY 節は省略できます。
- HAVING 節
  グループ化されたものをフィルタリングするために使用します。HAVING 節は省略できます。
- ORDER BY 節
  検索結果の順序化をするために使用します。ORDER BY 節は省略できます。

SELECT 文に，SELECT 節および FROM 節の指定がない場合は例外が発生します。

8.17.3 SELECT 節

SELECT 節ではクエリの結果を表します。一つ以上の値がクエリの SELECT 節から返ります。SELECT 節には，次に示す要素をコンマで区切って指定します。なお，コンマで区切られた一つのまとまりを select 式と呼びます。

- 抽象スキーマの識別子または識別子を付与した永続化フィールド
- パス式
- 集合関数
- コンストラクタ式

なお，DISTINCT キーワードは，重複する値をクエリ結果から除くときに指定してください。

SELECT 節の記述例を次に示します。

```
SELECT e.employeeName, e.monthlySalary
FROM Employee AS e
WHERE e.monthlySalary ＜ 150000
```

(1) コンストラクタ式

コンストラクタ式は，一つ以上の Java のインスタンスを返す SELECT 節の select 式で使用されます。生成名は完全修飾名を指定します。

コンストラクタ式は，エンティティのカラムの一部または全部や，関連する別のエンティティのカラムと組み合わせた形で取得できます。なお，この結果を格納するクラスはエンティティである必要はありません。

コンストラクタ式の構文は，select 式に NEW 演算子を付けて指定します。格納するクラスがエンティティの場合，エンティティクラスのインスタンスの状態は new になります。コンストラクタ式の記述例を次に示します。

```
SELECT NEW com.hitachi.jpa.test.entity.EmployeeTmp
(e.employeeId, e.employeeName, d.departmentName)
FROM Department AS d, d.employees AS e
WHERE e.employeeId = 12003
```

(2) 集合関数

SELECT 節では集合関数を使用できます。使用できる集合関数を次の表に示します。

表 8-20 JPQL の SELECT 節で使用できる集合関数

集合関数	引数	結果の型	適用される値がない場合の結果
AVG	数値型のステートフィールド※	Double 型	null
MAX	順序を指定できるフィールド型（数値型，文字列型，文字型，または日付型）※	適用するフィールドの型	null
MIN	順序を指定できるフィールド型（数値型，文字列型，文字型，または日付型）※	適用するフィールドの型	null
SUM	数値型のステートフィールド※	• 整数型の場合：Long 型	null

集合関数	引数	結果の型	適用される値がない場合の結果
		• 浮動小数点型の場合：Double 型 • BigInteger 型の場合：BigInteger 型 • BigDecimal 型の場合：BigDecimal 型	
COUNT	識別変数（引数にパス式を指定する場合，ステートフィールドか関連フィールドを指定する）	Long 型	0

注　DISTINCT が指定されているかどうかにかかわらず，集合関数が適用される前に null 値は除去されます。

注※　引数にパス式を指定する場合，関連フィールドを指定することはできません。

なお，集合関数式の構文の詳細については，「付録 G　JPQL の BNF」を参照してください。

(3)　SELECT 節の実行結果

クエリの SELECT 節で定義されるクエリ結果の型は，次のどれか一つになります。なお，複数ある場合は順番に並べられます。

- エンティティ抽象スキーマ型
- フィールド型
- 集合関数の結果
- コンストラクタ式の結果

SELECT 節の結果の型は，SELECT 節に含まれる select 式の結果の型によって定義されます。複数の select 式が SELECT 節で使われている場合，クエリの結果は Object[]型となります。この結果の要素は SELECT 節で指定された順番に一致し，各 select 式の結果の型と一致します。

8.17.4　FROM 節

ここでは FROM 節について説明します。

(1)　範囲変数宣言と識別変数

範囲変数宣言とは，FROM 節内でエンティティクラスの論理的な名前を記述し，そのあとに AS と識別子を指定する宣言です（AS は省略できます）。この範囲変数宣言の識別子を識別変数といいます。範囲変数宣言と識別変数の例を次に示します。

```
SELECT ・・・（省略）・・・

FROM  Department AS dep
```

```
WHERE ・・・（省略）・・・
```

「Department AS dep」の部分が範囲変数宣言です。また，「dep」が識別変数となります。次に，範囲変数宣言の構文を示します。

```
range_variable_declaration ::=
abstract_schema_name [AS] identification_variable
```

範囲変数宣言での識別変数の構文は，SQL の構文と同じです。構文について説明します。

- キーワード AS の使用は任意です。
- 識別変数を省略することはできません。ただし，抽象スキーマと識別変数の間に指定する AS は省略できます。識別変数は FROM 節で指定します。
- 予約済みの識別子は使用できません。使用した場合は例外が発生します。
- 同じ永続化ユニットのほかのエンティティと同じ名前は使用できません。CJPA プロバイダでは，同じ名前のエンティティを使用した場合の動作は保証しません。
- 識別変数は大文字，小文字の区別をしません。
- 抽象スキーマ名と同一の名前は指定できません。CJPA プロバイダでは，同じ名前の抽象スキーマを指定した場合の動作は保証しません。
- Java 識別子文字で始まり，ほかのすべての文字は Java 識別子の部分文字となる必要があります。それ以外の文字が指定された場合は，例外が発生します。最初の文字は，Character.isJavaIdentifierStart メソッドの戻り値が true になる文字にします（アンダースコア（_）とドルマーク（$）文字を含む）。最初以外の文字は，Character.isJavaIdentifierPart メソッドの戻り値が true になる文字にします（ただし，クエスチョンマーク（?）は，JPQL の予約語のため使用できません）。
- JPQL の予約語は次のとおりです。

  SELECT，FROM，WHERE，UPDATE，DELETE，JOIN，OUTER，INNER，LEFT，GROUP，BY，HAVING，FETCH，DISTINCT，OBJECT，NULL，TRUE，FALSE，NOT，AND，OR，BETWEEN，LIKE，IN，AS，UNKNOWN，EMPTY，MEMBER，OF，IS，AVG，MAX，MIN，SUM，COUNT，ORDER，BY，ASC，DESC，MOD，UPPER，LOWER，TRIM，POSITION，CHARACTER_LENGTH，CHAR_LENGTH，BIT_LENGTH，CURRENT_TIME，CURRENT_DATE，CURRENT_TIMESTAMP，NEW，EXISTS，ALL，ANY，SOME

  なお，「UNKNOWN」は JPA1.0 では利用していませんが，CJPA プロバイダでは予約語となっているので注意してください。

(2) パス式

パス式は，識別変数のあとにピリオド（.）を付加し，ステートフィールドまたは関連フィールドを続けるための式です。このため，パス式の型は，ステートフィールドまたは関連フィールドの型になります。

パス式をたどって得た関連フィールドから，さらにパス式を組み立てることができます。ただし，基になるパス式の型がコレクション関連フィールドである場合，パス式を組み立てることはできません。コレクション型からパス式を作ることは，構成的に誤りとなります。

なお，パス式の途中の関連フィールドが null 値の場合，パスは値がないとみなされるので，クエリの結果には影響はありません。

パス式は，inner join を使用する構文で使用できます。パス式の構文の詳細については，「付録 G　JPQL の BNF」を参照してください。

参考

関連フィールドの種類を次に示します。

- コレクション関連フィールド（collection_valued_association_field）とは，関連フィールドがコレクションで指定されているものです。OneToMany または ManyToMany の関係で示されます。
- 非コレクション関連フィールド（single_valued_association_field）とは，関連フィールドが single-valued で指定されているものです。OneToOne または ManyToOne の関係で示されます。
- エンベッデドクラスフィールドは，embedded クラスに対応するエンティティのフィールド名です。

(3)　Joins 式

Joins 式は FROM 節で使用できます。使用できる Joins 式を次の表に示します。

表 8–21　FROM 節で使用できる Joins 式

Joins 式	内容	BNF 構文の構文名※1
Inner Joins	関係するフィールドで，二つのエンティティクラスを結合し，関連を持っているエンティティオブジェクトだけを抽出します。	join, join_spec
Left Outer Joins	関係するフィールドで，二つのエンティティクラスを結合し，関連を持っているエンティティオブジェクトおよび，関連を持っていないエンティティオブジェクトも抽出します。	join, join_spec
Fetch Joins	関係のあるフィールドで，二つエンティティクラスを結合します。なお，エンティティ間にはリレーションシップによる関連があるため，Select 節中には一つのエンティティクラスだけを指定します。※2	fetch_join

注※1　BNF 構文の構文名の詳細については，「付録 G　JPQL の BNF」を参照してください。

注※2　Fetch Joins でエンティティを取得した場合，右側で指定した関連先のエンティティの情報をクエリ実行と同時に取得します。これによって，フェッチ戦略に依存しないで関連先の情報を取得できます。Fetch Joins の記述例を次に示します。

```
SELECT emp FROM Employee AS emp JOIN FETCH emp.company
```

Join 式を使用する場合の注意事項

Join 式を使用する場合の注意事項について説明します。

Inner Joins の注意事項

INNER キーワードは，任意で使われます。

Left Outer Joins の注意事項

OUTER キーワードは，任意で使われます。

Fetch Joins の注意事項

- Fetch Joins では，一つのエンティティで二つのエンティティ情報を指定します。指定するエンティティと，そのエンティティと関連する別のエンティティ情報を結合します。

  なお，CJPA プロバイダの場合，一つのエンティティ情報で結合するため，リレーションシップを持つエンティティを指定してください。リレーションシップを持たないエンティティを指定した場合は例外が発生します。
- JOIN FETCH の右側で参照される関係は，クエリの結果として返されるエンティティに属する関係となる必要があります。CJPA プロバイダの場合，エンティティに属さないときは例外が発生します。
- JOIN FETCH の右側で参照されるエンティティには，識別子を指定できません。このため，クエリ内で参照することはできません。

(4) コレクションメンバの宣言

コレクションメンバ宣言の識別変数は，予約された識別子 IN を使って宣言します。コレクションメンバ式で定義された識別変数は，パス式を使ってコレクションの値を取得できます。コレクションメンバ式の記述例を次に示します。

```
SELECT emp.employeeId, emp.employeeName, dep.departmentName
FROM Department AS dep, IN (dep.employees) AS emp
WHERE dep.departmentId = 3
```

なお，コレクションメンバ式の構文の詳細については，「付録 G　JPQL の BNF」を参照してください。

(5) 注意事項

ここでは，FROM 節での注意事項について説明します。

- 識別変数の影響

  FROM 節で宣言された識別変数が WHERE 節の中で使用されなくても，宣言された識別変数がクエリの結果に反映されます。
- 多態性の注意事項

JPQL はポリフォリズムです。FROM 節で明確に参照している特定のエンティティクラスのインスタンスだけでなく，そのサブクラスも対象になります。このため，クエリによって取得できるインスタンスは，クエリ条件を満たすサブクラスのインスタンスを含みます。

8.17.5 WHERE 節

WHERE 節は式を満たすオブジェクトまたは変数を検索するために使われる条件式で構成されています。WHERE 節では，select 文の結果や update や delete 操作の範囲を特定します。

(1) WHERE 節で使用できる条件式

WHERE 節で使用できる条件式を次の表に示します。

表 8-22 WHERE 節で使用できる条件式

式	内容	BNF 構文の構文名※
BETWEEN	フィールドで指定した範囲に含まれていることを評価します。	between_expression
IN	フィールドで指定したどれかの値と一致することを評価します。	in_expression
LIKE	フィールドでワイルドカード記号を展開したあとの文字列と一致することを評価します。	like_expression
IS [NOT] NULL	null 値かどうかをテストします。	null_comparison_expression
IS [NOT] EMPTY	指定したコレクション値が空かどうかをテストします。	empty_collection_comparison_expression
[NOT] MEMBER [OF]	指定したコレクション値がコレクションのメンバであるかどうかをテストします。空のコレクションを表す場合，MEMBER OF 式の値は FALSE，NOT MEMBER OF 式の値は TRUE となります。	collection_member_expression
EXISTS	サブクエリの結果を判定します。一つ以上の値がある場合は true，それ以外は false を返します。	exists_expression
ALL	サブクエリで返されたすべての値と比較します。	all_or_any_expression
ANY (SOME)	サブクエリで返されたどれかの値と比較します。	all_or_any_expression
サブクエリ	select による結果を記述できます。	subquery
関数式	「(2) 関数式」を参照してください。	該当する関数構文を参照

注※ BNF 構文の構文名の詳細については，「付録 G JPQL の BNF」を参照してください。

WHERE 節を使用する場合の注意事項について説明します。

- IN 式
  - 評価対象のフィールドが null 値か unknown の場合，式の値は unknown になります。

- IN 式に対する値の集合を定義したコンマで区切られたリストでは，少なくとも一つ以上の要素が必要です。CJPA プロバイダの場合，一つ以上の要素がないと例外が発生します。
- 評価対象のフィールドは，文字列，数値，または enum 値を持つ必要があります。
- リテラルや入力パラメタ値，およびサブクエリの結果は，評価対象のフィールドと同じ抽象スキーマ型にしてください。CJPA プロバイダの場合，同じ抽象スキーマ型でないと例外が発生します。

- LIKE 式
  - BNF 構文で示される string_expression または pattern_value の値が，null または unknown の場合，LIKE 式の値は unknown になります。
  - BNF 構文で示される escape_character が指定されている場合で，値が null のとき，LIKE 式の値は unknown になります。
  - 指定するパターン値は，アンダースコア（_）が 1 文字に相当し，パーセント（%）文字が連続文字（空の連続文字も含む）に相当します。なお，ほかの文字は検索文字列を示します。
  - オプションのエスケープ文字は，文字列リテラルかまたは character 文字です。パターン文字列のアンダースコアとパーセント文字を標準の文字として解釈させる場合に使用します。
- IS [NOT] NULL 式

  指定したコレクション値が null または unknown の場合，IS [NOT] EMPTY 式の値は unknown になります。
- [NOT] MEMBER [OF]式

  指定した値が null または unknown の場合，戻り値は unknown になります。
- ALL 式または ANY (SOME)式
  - 条件演算が true でも false でもない場合は，unknown になります。
  - 一緒に使用できる比較演算子は，=，<，<=，>，>=，<>です。
  - サブクエリの結果の型は比較演算子の型と同じにする必要があります。CJPA プロバイダの場合，型が異なると例外が発生します。
  - SOME は，ANY の同義語です。※

  注※　この注意事項は ANY (SOME)式の場合だけ該当します。
- サブクエリ

  WHERE 節や HAVING 節で使用します。FROM 節では使用できません。

(2)　関数式

JPQL では次の表に示す関数を WHERE 節や HAVING 節で使用できます。関数式の引数の値が null か unknown の場合，関数式の値は unknown になります。

表 8-23　JPQL で使用できる関数

分類	関数	返り値	引数についての補足説明
文字列関数	CONCAT	引数の連結した文字列	－
	SUBSTRING	引数で指定した開始位置と長さの文字列	2 番目と 3 番目引数は，返されるサブストリングの開始位置および長さを整数で指定します。文字列の先頭の位置は 1 です。
	TRIM	特定の文字を取り除いた文字列	取り除く文字がない場合は，スペース（空白）と見なされます。任意の trim 文字は character の入力文字列です。トリムの方法が指定されない場合は BOTH となります。
	LOWER	小文字化した文字列	－
	UPPER	大文字化した文字列	－
	LENGTH	文字列長の整数値	－
	LOCATE	指定された位置から検索し，与えられた文字列で最初に文字列を見つけた位置の整数値 （文字列が見つからない場合は 0）	1 番目引数は検索する文字列，2 番目引数は検索される文字列を示します。任意となる 3 番目の引数は，検索を開始する文字の位置を示します（デフォルトは先頭から検索）。文字列の先頭位置は 1 です。
算術関数	ABS	関数の引数と同じ型（Integer, Float，または Double）の絶対値	引数として数値を渡します。
	SQRT	引数で渡された数値の平方根（実数）	引数として数値を渡します。
	MOD	引数で渡された二つの数値の剰余（整数）	二つの整数を渡します。
	SIZE	コレクションの要素数（整数）	コレクションを渡します。
日付時間関数	CURRENT_DATE	データベースの日付	－
	CURRENT_TIME	データベースの時刻	－
	CURRENT_TIMESTAMP	データベースのタイムスタンプ	－

（凡例）－：該当しない

(3)　注意事項

WHERE 節の注意事項について説明します。

(a)　演算子の優先順位

演算子の優先順位を次に示します。

1. ピリオド（.）
2. 算術演算子

   単項演算（+，-），乗算と除算（*，/），加算と減算（+，-）
3. 比較演算子

   =，>，>=，<，<=，<>（not equal），[NOT] BETWEEN，[NOT] LIKE，[NOT] IN，IS [NOT] NULL，IS [NOT] EMPTY，[NOT] MEMBER [OF]
4. 論理演算子

   NOT，AND，OR

(b) 条件式の注意事項

- 条件式は，条件式，比較式，論理演算子，評価結果が boolean 値になるパス式，boolean リテラル，boolean 型の入力パラメタで構成されます。
- 算術式は，比較式で使用できます。算術式は，ほかの算術式，四則演算，結果が数値になるパス式，数値リテラル，数値型の入力パラメタで構成されます。
- 算術操作は，数値昇格（numeric promotion）を使用します。
- 式の評価順序を示すために括弧で囲むことができます。
- 集合関数は，HAVING 節の条件式だけで使用できます。
- 条件式の中では，シリアライズ形式または lobs としてマップされるフィールドを使用しないでください。

(c) リテラルの注意事項

- 文字リテラルは，シングルクォテーションで囲みます（例：'literal'）。シングルクォテーションを含む文字列リテラルは，二つシングルクォテーションを使用します。
- JavaString リテラルのように，クエリの文字列リテラルは Unicode 文字のエンコードを使用します。
- Java のエスケープ表記の使用は，クエリの文字列リテラルではサポートしません。
- enum のリテラルは，Java の enum リテラル構文のリテラルが使用できます。enum リテラルには enum クラス名が必要です。
- boolean リテラルは，TRUE と FALSE である。このリテラルは大文字小文字を区別しません。

(d) 識別変数の注意事項

- 識別変数は，クエリの FROM 節で宣言された識別子であるため，ほかの節では宣言できません。FROM 節以外で宣言した場合は例外が発生します。SELECT 文，または DELETE 文の WHERE か HAVING 節で使用されるすべての識別変数は，FROM 節で定義されたものを使用します。
- 識別変数の範囲は，WHERE 節と HAVING 節で決定されます。識別変数は，エンティティの抽象スキーマタイプのコレクションメンバやインスタンスは指定できますが，コレクション自体は指定できません。コレクションを指定した場合は例外が発生します。

(e) Path 式の注意事項

WHERE か HAVING 節内でのコレクションのパス式で，条件式の一部として IS [NOT] EMPTY 式や [NOT] MEMBER [OF]式，または SIZE 操作への引数以外を使用しないでください。

(f) 入力パラメタの注意事項

- 入力パラメタは，クエリの WHERE 節か HAVING 節で使用できます。
- 一つのクエリで位置パラメタと名前付きパラメタを混在して使用しないでください。混在した場合の動作は保証されません。
- 入力パラメタの値が null の場合，入力パラメタを含む比較操作か算術操作で返される値は unknown になります。

なお，位置パラメタと名前付きパラメタの使い方については，「8.16.1(2) パラメタの指定方法」を参照してください。

8.17.6 GROUP BY 節および HAVING 節

GROUP BY 節では，クエリの結果をグループごとにまとめます。また，HAVING 節では，クエリ結果をさらに絞り込むための条件指定ができます。グループを指定した上で，HAVING 節の条件を指定してください。

クエリに WHERE 節と GROUP BY 節の両方が含まれる場合，WHERE 節が最初に実行され，その後，GROUP BY 節で形式を整えて HAVING 節に従ってフィルタリングします。

SELECT 節に現れる集合関数以外のアイテムは，GROUP BY 節にも指定する必要があります。グルーピングでは，null 値も含めて条件として扱います。GROUP BY 節および HAVING 節の注意事項について説明します。

- エンティティによるグルーピングはできますが，シリアライズしたフィールドや，lob フィールドを含むことはできません。CJPA プロバイダでは，指定した場合には例外が発生します。
- HAVING 節にはグループアイテムに対する検索条件を指定するため，グループアイテムに適用する集合関数を指定する必要があります。CJPA プロバイダでは，検索条件の指定がない場合は例外が発生します。
- GROUP BY 節がない状態で HAVING 節は使用しないでください。使用した場合は例外が発生します。

GROUP BY 節および HAVING 節の記述例を次に示します。

```
SELECT e.department.departmentId
FROM Employee AS e
GROUP BY e.department.departmentId
HAVING COUNT(e.department.departmentId) <= 2
```

なお，GROUP BY 節および HAVING 節の構文については，「付録 G　JPQL の BNF」を参照してください。

8.17.7　ORDER BY 節

ORDER BY 節ではオブジェクトや値が順序付けされてクエリの結果を返します。ORDER BY 節の記述例を次に示します。

```
SELECT e
FROM Employee AS e
ORDER BY e.monthlySalary DESC
```

ORDER BY 節について説明します。

- 一つ以上の order 項目が指定された場合，orderby 項目要素の左から右へ順に優先度を決め，いちばん左の orderby 項目がいちばん高い優先度を持ちます。
- ASC は昇順のときに，DESC は降順のときに指定します。なお，デフォルト値は ASC です。
- null 値がある場合の順番は SQL のルールを適用します。
- ORDER BY 節が使用された場合，クエリの結果の順番はクエリメソッドの結果に保存されます。

なお，ORDER BY 節の構文については，「付録 G　JPQL の BNF」を参照してください。

また，ORDER BY 節では次に示す条件を満たす必要があります。

- ORDER BY 節で指定する order 項目は順序付けができること。
- ORDER BY 節で指定する order 項目は SELECT 節の select 式からたどれること。

これらの条件に合わない場合，CJPA プロバイダでは例外が発生するおそれがあります。ただし，例外が発生しない場合でも動作は保証しません。

8.17.8　バルク UPDATE 文およびバルク DELETE 文

複数のレコードを一括して更新することをバルク更新といいます。バルク更新をするには，バルク UPDATE 文およびバルク DELETE 文を使用します。バルク UPDATE と DELETE の操作は，単体のエンティティクラスまたはサブクラスと合わせたエンティティクラスに適用されます。UPDATE 文では UPDATE 節に，DELETE 文では FROM 節に識別変数を定義して，1 種類のエンティティを指定します。

バルク UPDATE 文およびバルク DELETE 文を使用するときの注意時について説明します。

- delete 操作は，指定されたクラスとそのサブクラスのエンティティにだけ適用されます。関係するエンティティへのカスケードはされません。

- update 操作で指定される更新値（new_value）は，割り当てられたフィールドの型との互換性が必要です。互換性がない場合は例外が発生します。
- バルク UPDATE 文は，楽観的ロックを掛けないでデータベースの更新操作を実行するため，楽観的ロックに関する処理は実行しません。このため，バージョン列の値を手動で参照・更新する必要があります。

注意事項

バルク UPDATE 文やバルク DELETE 文を実行するときは，データベースと活性化している永続化コンテキストのエンティティとの間で不一致になるため注意が必要です。バルク UPDATE 文とバルク DELETE 文での操作は，トランザクションと切り離したところで実行するか，トランザクションの開始時に実行する必要があります。

バルク UPDATE 文およびバルク DELETE 文の記述例を次に示します。

```
UPDATE EMPLOYEE
SET MONTHLY_SALARY = MONTHLY_SALARY + 1000
WHERE DEPARTMENT_ID = 3

DELETE FROM EMPLOYEE e
WHERE e.EMPLOYEE_NAME IS NULL AND e.monthlySalary  IS EMPTY
```

なお，バルク UPDATE 文，バルク DELETE 文の構文については，「付録 G　JPQL の BNF」を参照してください。

8.17.9　JPQL 使用時の注意事項

ここでは，JPQL 使用時の注意事項について説明します。

(1)　null 値の注意事項

- クエリ結果の null 値
  - クエリの結果の値が，null 値を持つ関連フィールドまたはステートフィールドに対応する場合，その null 値がクエリメソッドの結果として返されます。
  - IS NOT NULL の構文は，クエリの結果の集合から null 値を除去するために使用できます。
  - Java の数値系プリミティブ型によって定義されるフィールドには，クエリの結果として null 値を生成できません。
- 比較，条件式での null 値
  - 参照の対象がデータベースに存在しない場合，その値は null とみなされます。null 値を検索するには，比較条件 IS NULL および IS NOT NULL だけが使用できます。

- null 値を IS NULL や IS NOT NULL 以外の条件で使用して，その結果が null 値に依存する場合，結果は unknown になります。
- null 値の比較または算術操作の結果は，常に unknown 値になります。
- 二つの null 値の比較結果は unknown 値になります。
- unknown 値のある比較または算術操作の結果は，常に unknown 値になります。
- boolean 演算子は 3 値論理を使用します。AND の場合，OR の場合，および NOT の場合の 3 値論理式をそれぞれ表に示します。

表 8–24　AND の 3 値論理式（A AND B の結果）

A	B		
	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

表 8–25　OR の 3 値論理式（A OR B の結果）

A	B		
	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

表 8–26　NOT の 3 値論理式（NOT A の結果）

A	NOT A の結果
True	False
False	True
Unknown	Unknown

(2)　HiRDB で JPQL を使用する場合の注意事項

- 次に示す JPQL の関数の引数に，位置パラメタや名前パラメタを指定することはできません。

  TRIM，SQRT，ABS，LENGTH，LOWER，MOD，LOCATE，UPPER，CONCAT，SUBSTRING，IS [NOT] NULL

  これらの関数の引数に位置パラメタや名前パラメタを指定した場合，動作は保証されません。なお，位置パラメタや名前パラメタを指定した場合は HiRDB の SQL 構文エラーとなり，SQLException 例外を含んだ PersistenceException 例外をスローすることがあります。

  これらの関数の機能を使用したい場合は，ネイティブクエリを使用してください。

- 四則演算子（＋，－，＊，／）の両側に？パラメタは指定できません。また，比較演算子（=，>，>=，<，<=，<>）の両側に？パラメタを指定したり，片側に？パラメタを指定したりしてもう一方にリテラルを指定することはできません。指定した場合の動作は保証されません。指定した場合はHiRDB の SQL 構文エラーとなり，SQLException 例外を含んだ PersistenceException 例外をスローすることがあります。

  JPQL の中で四則演算や比較演算しないで，JPQL を使用する前に四則演算や比較演算の操作をしてから JPQL を使用するようにしてください。

HiRDB で JPQL を使用する場合に使用できないクエリの記述例を次に示します。

使用できない例 1：関数の引数に位置パラメタを指定する

```
Query query1 = em.createQuery(
"SELECT  o FROM TestEntity o "+
                "WHERE o.name=TRIM(LEADING FROM ?1)")
             .setParameter(1, "  SatoTaro");
```

使用できない例 2：四則演算子に位置パラメタを指定する

```
int no_A=2;
int no_B=4;
Query query2 = em.createQuery(
"SELECT  o FROM TestEntity o WHERE o.id = ?1 + ?2")
               .setParameter(1, no_A)
               .setParameter(2, no_B);
```

使用できない例 3：比較演算子に位置パラメタを指定する

```
int cmp_no=3;
Query query3 = em.createQuery(
"SELECT  o FROM TestEntity o WHERE o.id = ?1 AND ?1 < 9")
               .setParameter(1, cmp_no);
```

8.17.10　クエリ使用時に発生する例外

CJPA プロバイダの場合，クエリ関連のアノテーションで文法的に間違いがあると，アプリケーションをデプロイするとき例外が発生します。また，クエリ関連のメソッドの引数が不正であったり，指定された文字列が有効な JPQL の文字列でなかったりすると，IllegalArgumentException 例外が発生するか，クエリの実行が失敗します。

ネイティブクエリを使用しているデータベースのクエリに対して有効でない場合，または定義された結果のセットがクエリの結果と互換性がない場合，クエリの実行は失敗します。CJPA プロバイダでは，クエリの実行時に PersistenceException 例外がスローされます。

(1) EntityManager 内のクエリ関連インタフェースの API で発生する例外

EntityManager 内のクエリ関連インタフェースの API で発生する例外を次に示します。

- createQuery メソッドのクエリ文字列が正しくない場合は，IllegalArgumentException 例外がスローされます。
- createNamedQuery メソッドに指定された名前でクエリが定義されていない場合は，IllegalArgumentException 例外がスローされます。

(2) Query インタフェースの API で発生する例外

Query インタフェースの API で発生する例外を次に示します。

- getResultList メソッドで，JPQL の UPDATE 文または DELETE 文が呼び出された場合は，IllegalStateException 例外がスローされます。
- getSingleResult メソッドで，クエリ結果がない場合は，NoResultException 例外がスローされます。なお，クエリ結果が二つ以上ある場合は，NonUniqueResultException 例外がスローされます。JPQL の UPDATE 文，または DELETE 文が呼び出された場合は，IllegalStateException 例外がスローされます。
- executeUpdate メソッドで，JPQL の SELECT 文が呼び出された場合は，IllegalStateException 例外がスローされます。このとき，トランザクションがない場合は，TransactionRequiredException 例外がスローされます。
- setHint メソッドの第 2 引数が実装に対して正しくない場合は，IllegalArgumentException 例外がスローされます。
- setParameter メソッドの引数の位置がクエリの位置パラメタと一致しない場合，または引数のパラメタ名がクエリ文字列のパラメタと一致しない場合は，IllegalArgumentException 例外がスローされます。
- setMaxResults または setFirstResult メソッドの引数が負数の場合は，IllegalArgumentException 例外がスローされます。

なお，API の詳細については，Java のドキュメントを参照してください。

8.18　persistence.xml の定義

この節では，persistence.xml の定義について，CJPA プロバイダ独自の機能であるエンティティオブジェクトのキャッシュ機能の定義と，データソースの指定の注意を説明します。

8.18.1　エンティティオブジェクトのキャッシュ機能の定義

CJPA プロバイダで提供するエンティティオブジェクトのキャッシュ機能の定義は，persistence.xml の<property>タグ内に指定します。persistence.xml でのエンティティオブジェクトのキャッシュ機能の定義について次の表に示します。

表 8-27　persistence.xml でのエンティティオブジェクトのキャッシュ機能の定義

指定するプロパティ	設定内容
cosminexus.jpa.cache.size.<ENTITY>	エンティティをキャッシュする場合のキャッシュサイズを指定します。
cosminexus.jpa.cache.size.default	エンティティをキャッシュする場合のキャッシュサイズのデフォルトを指定します。
cosminexus.jpa.cache.type.<ENTITY>	エンティティのキャッシュタイプを指定します。
cosminexus.jpa.cache.type.default	エンティティのキャッシュタイプのデフォルトを指定します。
cosminexus.jpa.target-database	接続するデータベースの名前を指定します。

タグの詳細は「12.2.2　<property>タグに指定できる CJPA プロバイダ独自のプロパティ」を参照してください。

参考

ここで説明しているプロパティは，CJPA プロバイダ独自のプロパティです。persistence.xml には，このほかに JPA 仕様で定義されたプロパティを指定することができます。ただし，CJPA プロバイダの場合，JPA 仕様で規定されている javax から始まるプロパティは使用できません。

8.18.2　データソースの指定の注意

persistence.xml でのデータソースの指定では，アプリケーションサーバの機能であるユーザ指定名前空間機能を使用して，リソースアダプタに別名を付けることができます。persistence.xml の設定でリソースアダプタに別名を設定した場合は，Connector 属性ファイルでもリソースアダプタの別名の定義が必要です。詳細については，「8.19　実行環境での設定」を参照してください。

8.19　実行環境での設定

CJPA プロバイダを使用する場合，J2EE サーバの設定および DB Connector の設定が必要です。

8.19.1　J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。CJPA プロバイダが出力するログファイルの設定は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの CJPA プロバイダが出力するログファイルの設定について次の表に示します。

表 8–28　簡易構築定義ファイルでの CJPA プロバイダが出力するログファイルの設定

指定するパラメタ	設定内容
ejbserver.logger.channels.define.JPAOperationLogFile.filenum	稼働ログの面数を指定します。
ejbserver.logger.channels.define.JPAMaintenanceLogFile.filenum	保守ログの面数を指定します。
ejbserver.logger.channels.define.JPAOperationLogFile.filesize	稼働ログのサイズを指定します。
ejbserver.logger.channels.define.JPAMaintenanceLogFile.filesize	保守ログのサイズを指定します。
cosminexus.jpa.logging.level.operation.<category>	稼働ログのログレベルを指定します。

簡易構築定義ファイルおよび指定するパラメタの詳細は，「11.2.1　J2EE サーバ用ユーザプロパティを設定するパラメタ」を参照してください。

8.19.2　DB Connector の設定

DB Connector の設定は，サーバ管理コマンドおよび Connector 属性ファイルで実施します。

Connector 属性ファイルでの設定について次の表に示します。

表 8–29　Connector 属性ファイルでの設定

指定するタグ	設定内容
<resource-external-property>－<optional-name>タグ	persistence.xml のデータソースの指定でリソースアダプタの別名を使用している場合，リソースアダプタの別名を設定します。persistence.xml でリソースアダプタの別名を使用しない場合は設定不要です。

指定するタグ	設定内容
<resource-external-property>－<res-auth>タグ	persistence.xml のデータソースの指定でリソースアダプタの別名を使用している場合，リソースの認証方式として必ず「Container」を指定してください。
<resource-external-property>－<res-sharing-scope>タグ	persistence.xml のデータソースの指定でリソースアダプタの別名を使用している場合，リソース接続を共有するかどうかの指定として必ず「Shareable」を指定してください。
<property>－<property-name>タグ <property>－<property-value>タグ	データベースへの接続情報を指定します。 • ユーザ名の指定 <property-name>タグに User を，<property-value>タグにデータベース接続で使用するユーザ名を指定します。 • パスワードの指定 <property-name>タグに Password を，<property-value>タグにデータベース接続で使用するパスワードを指定します。
<connection-definition>－<transaction-support>タグ	トランザクションのサポートレベルを指定します。ここに指定する値は，persistence.xml の<persistence-unit>タグの transaction-type 属性に指定した値と合わせてください。

Connector 属性ファイルおよび指定するタグの詳細は「19.6　Connector 属性ファイル」を参照してください。

8.20　アプリケーション開発時の注意事項

8.20.1　Cosminexus 09-60 から 11-30 の環境，または Cosminexus 11-40 以降かつ JDK 11 を使用している環境で，@OneToOne および@ManyToOne での LAZY フェッチを使用する場合の注意事項

Cosminexus 09-60 から 11-30 の環境，または Cosminexus 11-40 以降かつ JDK 11 を使用している環境で，アプリケーションを開発する場合，次の条件が重なるエンティティクラスは Java SE 6 のソースファイルとして作成してください。またアプリケーションコンパイル時に Java SE 6 のクラスファイルを生成するように設定し，コンパイルしてください。

- エンティティクラスに@OneToOne または@ManyToOne が含まれる。
- @OneToOne または@ManyToOne の fetch 属性に FetchType.LAZY を指定している。

注　javac コマンドを使用してコンパイルする場合，「-target」および「-source」を使用することで，Java SE 6 のクラスファイルを生成できます。

8.21　JPA アプリケーションでトラブルが発生した場合

JPA を使用したアプリケーションでトラブルが発生した場合の解析手順について説明します。

ここでは，代表的な障害の例とその場合の障害原因の解析手順を説明します。

なお，CJPA プロバイダが出力する各ログ間の情報は，時刻，スレッド ID／プロセス ID，および PersistenceUnit 名を基にして対応づけます。取得したログでの原因の特定ができない場合で，再現性があるときには，ログレベルを上げて資料を取得してください。

8.21.1　ユーザアプリケーションでの例外発生

ユーザアプリケーションで例外が発生した場合のユーザが実施する解析手順を説明します。例外が発生するタイミングは，アプリケーションの起動時（デプロイを含む）またはアプリケーションの実行時が考えられます。それぞれのタイミングで例外が発生したときの解析手順について説明します。

(1)　アプリケーション起動時の例外発生

アプリケーションの起動時に例外が発生した場合の解析手順を次に示します。

1. メッセージログを確認する。

   メッセージログに出力されたメッセージを参照します。メッセージマニュアルを参照して，出力されたメッセージ ID の対処方法を確認します。また，対策を検討します。

2. アプリケーションの定義を確認する。

   メッセージログの内容に従って，アプリケーション内のアノテーションの定義情報，persistence.xml，O/R マッピングファイルの設定内容を見直し，設定内容に問題がないかどうかを確認してください。設定内容の確認で原因が判明しない場合，出力されたメッセージの対処方法が保守員連絡となっている場合は，保守員に連絡してください。

(2)　アプリケーション実行時の例外発生

アプリケーション実行時に例外が発生した場合の解析手順を次に示します。

1. メッセージログを確認する。

   メッセージログに出力されたメッセージから障害の事象を確認します。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。

2. 例外ログを確認する。

   例外ログに出力されたスタックトレースから，例外が発生した個所を特定します。例外が発生した原因がアプリケーションの処理内容に問題があるかどうかを確認します。アプリケーションの処理に問題がある場合は，アプリケーションを修正してください。

メッセージログの対処方法が保守員連絡になっている場合，アプリケーションの処理に問題がない場合などの原因が判明しないときには，保守員に連絡してください。

8.21.2　性能面での障害発生

特定の処理に時間が掛かっているなど，性能面で障害が発生した場合の解析手順を説明します。性能面で問題が発生するタイミングは，アプリケーション起動時またはアプリケーションの実行時が考えられます。

アプリケーションの起動時に性能面で問題がある場合には，保守員に連絡してください。アプリケーションの実行時に性能面で問題がある場合には，次に示す手順で問題の個所を特定してください。

1. PRF トレースを確認する。

   PRF トレースの内容から処理に時間が掛かっている個所を特定します。ユーザ実装個所と CJPA プロバイダのどちらに障害の原因があるかを特定します。ユーザ実装個所に問題がある場合にはアプリケーションを修正するなどの対処を実施してください。CJPA プロバイダの内部処理で処理に時間が掛かっている場合には保守員に連絡してください。

2. 稼働ログを確認する。

   発行した SQL を確認したい場合には，稼働ログを確認します。

   例えば，次に示す場合などに該当するかどうかは，発行している SQL を参照することで確認できます。

   - 継承戦略に JOINED を利用したため，エンティティの読み込みで JOIN 操作が多発している。
   - コレクションで管理しているエンティティを操作したため，その延長で各要素に対して SELECT が発生している。

なお，CJPA プロバイダで出力される稼働ログについては，「15.1.1　CJPA プロバイダの稼働ログ」を参照してください。

8.21.3　トラブルシューティングで使用する資料

システムで障害が発生した場合，CJPA プロバイダで必要とするトラブルシューティングの資料と資料の取得元を次の表に示します。

表 8-30　CJPA プロバイダで必要とするトラブルシューティングの資料と資料の取得元

トラブルシューティング資料	取得元	資料のファイル名称
メッセージログ	snapshot(1 次)	メッセージログ
例外ログ	snapshot(1 次)	障害発生時の例外情報
CJPA プロバイダの稼働ログ	snapshot(1 次)	CJPA の稼働ログ
J2EE サーバの定義情報	snapshot(1 次)	• J2EE サーバ用オプション定義ファイル

トラブルシューティング資料		取得元	資料のファイル名称
			• J2EE サーバ用ユーザプロパティファイル • J2EE サーバ用セキュリティポリシーファイル
J2EE サーバの作業ディレクトリに含まれるアプリケーションの定義情報	persistence.xml	snapshot(2 次)	• EJB サーバ作業ディレクトリの内容 • Web コンテナ作業ディレクトリの内容
	O/R マッピングファイル		
	EJB，WAR の属性ファイル		
DB Connector の Connector 属性ファイル		snapshot(2 次)	Web コンテナ作業ディレクトリの内容
DB Connector のログ		snapshot(1 次)	J2EE リソースアダプタとしてデプロイして使用するリソースアダプタの稼働ログ
性能解析トレース		snapshot(1 次)	PRF デーモンおよび PRF コマンドのログ
J2EE サーバのスレッドダンプ		スレッドダンプ取得コマンド	
接続するデータベースの SQL トレース （ただし，トレース情報を採取している場合だけ）	• HiRDB の場合 SQL トレース	マニュアル「HiRDB SQL リファレンス」を参照してください。	
	• Oracle の場合 SQL トレース init.ora で sql_trace=true と設定時に取得されるトレースファイル	Oracle のマニュアルを参照してください。	

8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲

javax.persistence パッケージに含まれるアノテーションの一覧およびアノテーションを指定する際の注意事項について説明します。

なお，マッピング情報はアノテーションの代わりに O/R マッピングファイルで指定することもできます。アノテーションと O/R マッピングファイルとの対応については，「8.22.59　アノテーションと O/R マッピングとの対応」を参照してください。

アノテーションを指定する際の注意事項

- アプリケーションサーバの JPA 機能では，DDL 出力機能に関連する属性に対応していません。
- アノテーションで同じカラム名を複数指定する場合は，大文字および小文字をそろえて指定してください。
- フィールド名またはメソッド名をカラム名に割り当てた場合，アプリケーションサーバの JPA 機能では文字列を大文字として扱います。対応するアノテーションでカラム名を指定する場合は，大文字にしてください。
- アクセスタイプは，アノテーションを付与する場所によって決まります。ただし，アクセスタイプがフィールドとプロパティで混在した場合は，フィールドの設定が有効になります。
- プロパティ名は，アクセサメソッドの get または set（is）を除いた文字列によって次のように決まります。
  - ・最初の二文字が大文字の場合，そのままの文字列になります。
  - ・最初の二文字が大文字ではない場合，最初の文字を小文字に変換した文字列になります。
  - ・一文字の場合，最初の文字を小文字に変換した文字列になります。

アノテーション一覧

アノテーションの区分	アノテーション名	概要
エンティティのアノテーション	@Entity	クラスがエンティティであることを示します。
テーブル・カラム関連のアノテーション	@Column	永続化フィールドまたは永続化プロパティと，データベース上のカラムとのマッピングを指定します。
	@JoinColumn	エンティティクラス間の関連づけで，結合表のための外部キーカラムまたは外部キーカラムから参照された，結合先テーブルのカラムを指定します。
	@JoinColumns	@JoinColumn を複数同時に記述する場合に使用します。
	@JoinTable	次のクラスに設定する結合表を指定するアノテーションです。 • ManyToMany リレーションシップを指定する場合の所有者側のクラス

アノテーションの区分	アノテーション名	概要
		• 片方向の OneToMany リレーションシップを持つクラス
	@PrimaryKeyJoinColumn	ほかのテーブルと結合する場合に，外部キーとして使われるカラムを指定します。
	@PrimaryKeyJoinColumns	@PrimaryKeyJoinColumn を複数同時に記述する場合に使用します。
	@SecondaryTable	エンティティクラスにセカンダリテーブルを指定します。
	@SecondaryTables	@SecondaryTable を複数同時に記述する場合に使用します。
	@Table	エンティティクラスにプライマリテーブルを指定します。
	@UniqueConstraint	プライマリテーブルまたはセカンダリテーブルに対して，CREATE 文を生成する場合にユニーク制約を含めることを指定します。 なお，このアノテーションは，CJPA プロバイダには対応していません。
ID 関連のアノテーション	@EmbeddedId	埋め込み可能クラスの複合プライマリキーであることを指定します。
	@GeneratedValue	プライマリキーカラムにユニークな値を自動で生成，付与する方法を指定します。
	@Id	エンティティクラスのプライマリキーのプロパティ，またはフィールドであることを指定します。
	@IdClass	エンティティクラスの複数のフィールドまたはプロパティへマップされた複合プライマリキークラスを指定します。
	@SequenceGenerator	プライマリキーを作成するシーケンスジェネレータの設定を指定します。
	@TableGenerator	プライマリキーを作成するジェネレータの設定を指定します。
ロックのアノテーション	@Version	楽観的ロック機能を使用するために用いる version フィールドまたは version プロパティを指定します。
マッピング関連のアノテーション	@Basic	最も単純なデータベースのカラムへのマッピングの型を示します。
	@Embeddable	埋め込みクラスであることを指定します。
	@Embedded	埋め込み先のエンティティクラス内で，埋め込みクラスのインスタンス値を示す永続化プロパティまたは永続化フィールドであることを指定します。

アノテーションの区分	アノテーション名	概要
	@Enumerated	永続化フィールドまたは永続化プロパティを列挙型として指定します。
	@Lob	データベースがサポートしている large オブジェクト型の永続化フィールドまたは永続化プロパティであることを指定します。
	@MapKey	OneToMany リレーションシップ，または ManyToMany リレーションシップで，被所有者側のエンティティクラスが java.util.Map 型で示される場合にマップ内のオブジェクト識別に用いられるマップキーを指定します。
	@OrderBy	エンティティの情報を取得するとき，コレクションが評価した順番を指定します。
	@Temporal	時刻を表す型（java.util.Date および java.util.Calendar）を持つ永続化プロパティまたは永続化フィールドに指定します。
	@Transient	永続化しないエンティティクラス，マップドスーパークラス，または埋め込みクラスのフィールドまたはプロパティであることを指定します。
リレーション関連のアノテーション	@ManyToMany	指定したクラスが ManyToMany リレーションシップであることを示し，所有者側のエンティティクラスから被所有者側のエンティティクラスへの複数の関連を指定します。
	@ManyToOne	指定したクラスが ManyToOne リレーションシップであることを示し，被所有者側のエンティティクラスへの関連を指定します。
	@OneToMany	指定したクラスが OneToMany リレーションシップであることを示し，所有者側のエンティティクラスから被所有者側のエンティティクラスへの複数の関連を指定します。
	@OneToOne	指定したクラスが OneToOne リレーションシップであることを示し，エンティティクラス間の一つの関連を指定します。
継承・オーバーライド関連のアノテーション	@AssociationOverride	マップドスーパークラスや埋め込みクラスで指定された，ManyToOne リレーションシップまたは OneToOne リレーションシップで使用する設定をオーバーライドします。
	@AssociationOverrides	@AssociationOverride を複数同時に記述する場合に使用します。
	@AttributeOverride	次に示すマッピング情報をオーバーライドします。 • @Basic が指定された（またはデフォルトで適用された）プロパティ，フィールド • @Id で指定されたプロパティ，フィールド

アノテーションの区分	アノテーション名	概要
	@AttributeOverrides	@AttributeOverride を複数同時に記述する場合に使用します。
	@DiscriminatorColumn	SINGLE_TABLE 戦略または JOINED 戦略で使用する識別用カラムを指定します。 エンティティクラスの継承で，スーパークラスとなるエンティティクラスに付与します。
	@DiscriminatorValue	SINGLE_TABLE 戦略または JOINED 戦略で使用する識別用カラムの値を指定します。
	@Inheritance	エンティティクラス階層で使われる継承マッピング戦略を指定します。
	@MappedSuperclass	マップドスーパークラスであることを指定します。
クエリ関連のアノテーション	@ColumnResult	SQL のクエリ結果をエンティティクラスとマッピングするためのカラムを指定します。
	@EntityResult	SQL のクエリ結果をマッピングするエンティティクラスを指定します。
	@FieldResult	SQL のクエリ結果をマッピングするフィールドを指定します。
	@NamedNativeQueries	@NamedNativeQuery を複数同時に記述する場合に使用します。
	@NamedNativeQuery	SQL で名前付きクエリを指定します。
	@NamedQueries	@NamedQuery を複数同時に記述する場合に使用します。
	@NamedQuery	JPQL の名前付きクエリを指定します。
	@QueryHint	データベース固有のクエリのヒントを指定します。
	@SqlResultSetMapping	SQL のクエリの結果セットマッピングを指定します。
	@SqlResultSetMappings	@SqlResultSetMapping を複数同時に記述する場合に使用します。
イベント・コールバック関連のアノテーション※	@EntityListeners	エンティティクラスまたはマップドスーパークラスで使用されるコールバックリスナクラスを指定します。
	@ExcludeDefaultListeners	次に示すクラスに対して，デフォルトリスナを抑止するアノテーションです。 • エンティティクラス • マップドスーパークラス • エンティティクラスまたはマップドスーパークラスのサブクラス

アノテーションの区分	アノテーション名	概要
	@ExcludeSuperclassListeners	次に示すクラスに対して，スーパークラスリスナを抑止するアノテーションです。 • エンティティクラス • マップドスーパークラス • エンティティクラスまたはマップドスーパークラスのサブクラス
	@PostLoad	データベースに SELECT 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。
	@PostPersist	データベースに INSERT 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。
	@PostRemove	データベースに DELETE 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。
	@PostUpdate	データベースに UPDATE 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。
	@PrePersist	データベースに INSERT 文を発行する前に呼び出されるコールバックメソッドであることを示すアノテーションです。
	@PreRemove	データベースに DELETE 文を発行する前に呼び出されるコールバックメソッドであることを示すアノテーションです。
	@PreUpdate	データベースに UPDATE 文を発行する前に呼び出されるコールバックメソッドであることを示すアノテーションです。
EntityManager と EntityManagerFactory のリファレンス関連のアノテーション	@PersistenceContext	コンテナ管理の EntityManager を定義します。詳細はマニュアル「アプリケーションサーバ リファレンス API 編」の「2.7　javax.persistence パッケージ」を参照してください。
	@PersistenceContexts	@PersistenceContext を複数同時に記述する場合に使用します。詳細はマニュアル「アプリケーションサーバ リファレンス API 編」の「2.7 javax.persistence パッケージ」を参照してください。
	@PersistenceProperty	コンテナ管理の EntityManager にプロパティを設定します。詳細はマニュアル「アプリケーションサーバ リファレンス API 編」の「2.7 javax.persistence パッケージ」を参照してください。

アノテーションの区分	アノテーション名	概要
	@PersistenceUnit	EntityManagerFactory への永続化ユニットを定義します。詳細はマニュアル「アプリケーションサーバ リファレンス API 編」の「2.7 javax.persistence パッケージ」を参照してください。
	@PersistenceUnits	@PersistenceUnit を複数同時に記述する場合に使用します。詳細はマニュアル「アプリケーションサーバ リファレンス API 編」の「2.7 javax.persistence パッケージ」を参照してください。

注※　コールバックメソッドの詳細については，「8.15　コールバックメソッドの指定方法」を参照してください。

8.22.1　@AssociationOverride

(1)　説明

マップドスーパークラスや埋め込みクラスで指定された ManyToOne リレーションシップまたは OneToOne リレーションシップで使用する設定をオーバーライドするアノテーションです。

@AssociationOverride が指定されていない場合，外部キーカラムはオリジナルマッピングと同様にマッピングされます。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2)　属性

@AssociationOverride の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	オーバーライドする関連マッピングを持つフィールドまたはプロパティの名前を指定する属性です。
joinColumns	必須	@JoinColumn の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a)　name 属性

型

String

説明

オーバーライドする関連マッピングを持つフィールドまたはプロパティの名前を指定する属性です。

デフォルト値

なし

(b) joinColumns 属性

型

JoinColumn[]

説明

@JoinColumn の配列を指定する属性です。

マッピングの型はマップドスーパークラスまたは埋め込みクラスの定義が適用されます。

指定できる値は，@JoinColumn の配列で指定できる範囲です。詳細は，「8.22.24　@JoinColumn」を参照してください。

デフォルト値

なし

8.22.2 @AssociationOverrides

(1) 説明

@AssociationOverride を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2) 属性

@AssociationOverrides の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@AssociationOverride の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

AssociationOverride[]

説明

@AssociationOverride の配列を指定する属性です。

指定できる値は，@AssociationOverride の配列で指定できる範囲です。詳細は，「8.22.1 @AssociationOverride」を参照してください。

デフォルト値

なし

8.22.3 @AttributeOverride

(1) 説明

次に示すマッピング情報をオーバーライドするアノテーションです。

- @Basic で指定した（デフォルトが適用された）プロパティまたはフィールド
- デフォルトが適用されたプロパティまたはフィールド
- @Id で指定されたプロパティまたはフィールド

マップドスーパークラスや埋め込みクラスに定義された@Column の設定をオーバーライドするために，マップドスーパークラスを継承したエンティティクラスや埋め込みクラスのフィールドまたはプロパティに適用します。

@AttributeOverride が指定されていない場合，カラムはオーバーライド前のオリジナルマッピングでマップされます。

継承関係を持たない単体のエンティティクラスに@AttributeOverride を定義した場合，動作はしますが動作の保証はできません。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2) 属性

@AttributeOverride の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	マッピングがオーバーライドされるフィールドまたはプロパティの名前を指定する属性です。
column	必須	オーバーライドする@Column を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

マッピングがオーバーライドされるフィールドまたはプロパティの名前を指定する属性です。

デフォルト値

なし

(b) column 属性

型

Column

説明

オーバーライドする@Column を指定する属性です。

マッピングの型は埋め込み可能クラス，またはマップドスーパークラスの定義が適用されます。

指定できる値は，@Column で指定できる範囲です。詳細は，「8.22.6 @Column」を参照してください。

デフォルト値

なし

8.22.4 @AttributeOverrides

(1) 説明

@AttributeOverride を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2) 属性

@AttributeOverrides の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@AttributeOverride の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

AttributeOverride[]

説明

@AttributeOverride の配列を指定する属性です。

指定できる値は，@AttributeOverride の配列で指定できる範囲です。詳細は，「8.22.3 @AttributeOverride」を参照してください。

デフォルト値

なし

8.22.5 @Basic

(1) 説明

最も単純なデータベースのカラムへのマッピングの型を示すアノテーションです。

次に示す永続化する型のプロパティまたはインスタンス変数に適用できます。

- Java のプリミティブ型
- プリミティブ型のラッパークラス
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- Byte[]
- char[]
- Character[]
- enums
- ユーザが定義するシリアライズ型

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Basic の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
fetch	任意	フェッチ戦略の指定値を指定する属性です。
optional	任意	フィールドまたはプロパティに null 値を使用できるかどうかを指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) fetch 属性

型

FetchType

説明

フェッチ戦略の指定値を指定する属性です。

指定できる値は，FetchType.EAGER または FetchType.LAZY です。

なお，CJPA プロバイダでは，fetch 属性は無視され，デフォルトの FetchType.EAGER が常に適用されます。fetch 属性の詳細は，「8.4.5　データベースとの同期」を参照してください。

デフォルト値

FetchType.EAGER

8.22.6 @Column

(1) 説明

永続化フィールドまたは永続化プロパティと，データベース上のカラムとのマッピングを指定するアノテーションです。

永続化プロパティまたは永続化フィールドに明示的に@Column を指定しない場合でも，@Column が指定されたように永続化フィールドまたは永続化プロパティは扱われます。この場合@Column の各属性値にはデフォルト値が適用されます。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Column の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	カラム名を指定する属性です。
unique	任意	プロパティがユニークキーであるかどうかを指定する属性です。

属性名	任意/必須	属性の説明
		なお，この属性は，CJPA プロバイダには対応していません。
nullable	任意	データベースのカラムに null 値を指定できるかどうかを指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
insertable	任意	@Column で指定したカラムを SQL の INSERT 文に含むかどうかを指定する属性です。
updatable	任意	@Column で指定したカラムを SQL の UPDATE 文に含むかどうかを指定する属性です。
columnDefinition	任意	CREATE 文を出力するとき，カラムに付加する制約を DDL で記載する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
table	任意	カラムを含むテーブル名を指定する属性です。
length	任意	カラムの長さを指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
precision	任意	カラムの精度を指定する属性です。カラムが数値型の場合に指定します。 なお，この属性は，CJPA プロバイダには対応していません。
scale	任意	カラムのスケールを指定する属性です。カラムが数値型の場合に指定します。 なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

カラム名を指定する属性です。

指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

このアノテーションを指定したプロパティ名またはフィールド名

(b) insertable 属性

型

boolean

説明

@Column で指定したカラムを SQL の INSERT 文に含むかどうかを指定する属性です。指定できる値は，true または false です。

それぞれの値の意味は次のとおりです。

true：@Column で指定したカラムを SQL の INSERT 文に含みます。

false：@Column で指定したカラムを SQL の INSERT 文に含みません。

デフォルト値

true

(c) updatable 属性

型

boolean

説明

@Column で指定したカラムを SQL の UPDATE 文に含むかどうかを指定する属性です。指定できる値は，true または false です。

それぞれの値の意味は次のとおりです。

true：@Column で指定したカラムを SQL の UPDATE 文に含みます。

false：@Column で指定したカラムを SQL の UPDATE 文に含みません。

デフォルト値

true

(d) table 属性

型

String

説明

カラムを含むテーブル名を指定します。

指定できるテーブル名は，データベースの仕様に依存します。

デフォルト値

プライマリテーブル名

8.22.7 @ColumnResult

(1) 説明

SQL のクエリ結果をエンティティクラスとマッピングするためのカラムを指定するアノテーションです。

適用可能要素は，@SqlResultSetMapping の columns です。

(2) 属性

@ColumnResult の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	SELECT 節のカラムの名またはカラムの別名を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

SELECT 節のカラム名またはカラムの別名を指定する属性です。

指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

なし

8.22.8 @DiscriminatorColumn

(1) 説明

SINGLE_TABLE 戦略または JOINED 戦略で使用する識別用カラムを指定するアノテーションです。エンティティクラスの継承で，スーパークラスとなるエンティティクラスに付与します。

適用可能要素は，クラスです。

(2) 属性

@DiscriminatorColumn の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	識別用カラム名を指定する属性です。
discriminatorType	任意	識別用カラムの型を指定する属性です。
columnDefinition	任意	CREATE 文を出力するとき，識別用カラムに付加する制約を DDL で記載する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
length	任意	識別用カラムが文字列の場合，その長さを指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

識別用カラム名を指定する属性です。

指定できるカラム名は，データベースの仕様に依存します。

name 属性の値は，@Column の name 属性と同じ値（大文字および小文字を合わせた値）を指定してください。

デフォルト値

"DTYPE"

(b) discriminatorType 属性

型

DiscriminatorType

説明

識別用カラムの型を指定する属性です。

指定できる値は，次のとおりです。

- DiscriminatorType.STRING
- DiscriminatorType.CHAR
- DiscriminatorType.INTEGER

デフォルト値

DiscriminatorType.STRING

8.22.9 @DiscriminatorValue

(1) 説明

SINGLE_TABLE 戦略または JOINED 戦略で使用する識別用カラムの値を指定するアノテーションです。スーパークラスまたはサブクラスに指定できます。

適用可能要素は，クラスです。

なお，次の点に注意して指定してください。

- @DiscriminatorValue の設定は継承されません。@DiscriminatorValue をエンティティクラスごとに設定する必要があります。

- @DiscriminatorValue の設定は，@DiscriminatorColumn の discriminatorType の型と length の長さに一致させる必要があります。
- @DiscriminatorColumn の discriminatorType が INTEGER の場合は，次の点に注意してください。
  - @DiscriminatorValue には，先頭に 0 や空白を含まない整数だけを指定してください。
  - @DiscriminatorValue は省略できません。省略した場合の動作は保証しません。
- @DiscriminatorColumn の discriminatorType が INTEGER 以外の場合は，@DiscriminatorValue を省略できます。このとき，value の値がエンティティのクラス名であるものとして動作します。

(2) 属性

@DiscriminatorValue の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	識別用のカラムに設定する値を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

String

説明

識別用カラムに設定する値を指定する属性です。

指定できる値は，識別用カラムの型に依存します。

デフォルト値

エンティティ名

8.22.10 @Embeddable

(1) 説明

埋め込みクラスであることを示すアノテーションです。

埋め込みクラスとは，エンティティクラス内にフィールドとして埋め込むことができるクラスのことです。

適用可能要素は，クラスです。

(2) 属性

@Embeddable の属性はありません。

8.22.11 @Embedded

(1) 説明

エンティティクラス内で，埋め込みクラスのインスタンス値を示す永続化プロパティまたは永続化フィールドであることを示すアノテーションです。

埋め込みクラス内で宣言されたカラムマッピングをオーバーライドしたい場合は，@AttributeOverrideまたは@AttributeOverrides を使用します。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Embedded の属性はありません。

8.22.12 @EmbeddedId

(1) 説明

埋め込みクラスの複合プライマリキーであることを示すアノテーションです。

エンティティが所有する埋め込み可能クラスの永続化プロパティまたは永続化フィールドに付与します。

@EmbeddedId を利用する場合，@EmbeddedId を複数指定したり，@EmbeddedId 以外に@Id を指定したりしてはいけません。

@Transient を埋め込みクラスのフィールドに付与した場合，そのフィールドは複合プライマリキーの対象になりません。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@EmbeddedId の属性はありません。

8.22.13 @Entity

(1) 説明

クラスがエンティティであることを示すアノテーションです。

エンティティクラスのクラス名は，パッケージ名を含まないクラス名になります。指定するときは，次の内容に注意してください。

- エンティティ名は永続化ユニット内でユニークな名称にしてください。
- JPQL の予約文字を設定してはいけません。設定した場合の動作は保証しません。

適用可能要素は，クラスです。

(2) 属性

@Entity の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	エンティティクラスに対する論理的な名前を指定する属性です。なお，JPQL では，抽象スキーマ名となります。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

エンティティクラスに対する論理的な名前を指定する属性です。なお，JPQL では，抽象スキーマ名となります。

指定できる値は，JPQL の仕様に依存します。

デフォルト値

@Entity を指定したクラスのクラス名

8.22.14 @EntityListeners

(1) 説明

エンティティクラスまたはマップドスーパークラスで使用されるコールバックリスナクラスを指定するアノテーションです。

適用可能要素は，クラスです。

(2) 属性

@EntityListeners の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	コールバックリスナクラスを指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

Class[]

説明

コールバックリスナクラスを指定する属性です。

指定できる値は，クラスです。

デフォルト値

なし

8.22.15 @EntityResult

(1) 説明

SQL のクエリ結果をマッピングするエンティティクラスを指定するアノテーションです。

適用可能要素は，@SqlResultSetMapping の entities 属性です。

(2) 属性

@EntityResult の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
entityClass	必須	結果のクラスを指定する属性です。
fields	任意	@FieldResult の配列を指定する属性です。
discriminatorColumn	任意	エンティティインスタンスの型を決定する SELECT 節の中で，識別用カラムの名前または別名を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) entityClass 属性

型

Class

説明

結果のクラスを指定する属性です。

指定できる値は，クラス名です。

デフォルト値

なし

(b) fields 属性

型

FieldResult[]

説明

@FieldResult の配列を指定する属性です。

指定できる値は，@FieldResult の配列で指定できる範囲です。詳細は，「8.22.19 @FieldResult」を参照してください。

デフォルト値

空の配列

(c) discriminatorColumn 属性

型

String

説明

エンティティインスタンスの型を決定するための SELECT 節の中で，識別用カラムの名前または別名を指定する属性です。

指定できる値は，テーブルに指定されたカラムの名前または別名です。

デフォルト値

空の文字列

8.22.16 @Enumerated

(1) 説明

永続化フィールドまたは永続化プロパティを列挙型として指定するアノテーションです。

@Basic とともに使用できます。列挙型には ORDINAL（数値型）と STRING（文字列型）が指定できます。

次の場合，列挙型は ORDINAL（数値型）が指定されます。

- value 属性に列挙される型が指定されていない場合
- @Enumerated が指定されていない場合

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Enumerated の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	任意	列挙型をマッピングするのに使われる型を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

EnumType

説明

列挙型をマッピングするのに使われる型を指定する属性です。

指定できる値は，次のどちらかの値です。

- EnumType.ORDINAL：数値型
- EnumType.STRING：文字列型

デフォルト値

EnumType.ORDINAL

8.22.17 @ExcludeDefaultListeners

(1) 説明

次に示すクラスに対して，デフォルトリスナを抑止するアノテーションです。

- エンティティクラス
- マップドスーパークラス
- エンティティクラスまたはマップドスーパークラスのサブクラス

なお，デフォルトリスナを指定できるのは，XML ディスクリプタだけです。

適用可能要素は，クラスです。

(2) 属性

@ExcludeDefaultListeners の属性はありません。

8.22.18 @ExcludeSuperclassListeners

(1) 説明

次に示すクラスに対して，スーパークラスリスナを抑止するアノテーションです。

- エンティティクラス
- マップドスーパークラス
- エンティティクラスまたはマップドスーパークラスのサブクラス

適用可能要素は，クラスです。

(2) 属性

@ExcludeSuperclassListeners の属性はありません。

8.22.19 @FieldResult

(1) 説明

SQL のクエリ結果をマッピングするフィールドを指定するアノテーションです。

適用可能要素は，@EntityResult の field 属性です。

(2) 属性

@FieldResult の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	クラスの永続化フィールドまたは永続化プロパティの名前を指定する属性です。
column	必須	SELECT 節のカラム名またはカラムの別名を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

クラスの永続化フィールドまたは永続化プロパティの名前を指定する属性です。

デフォルト値

なし

(b) column 属性

型

String

説明

SELECT 節のカラム名またはカラムの別名を指定する属性です。

指定できるカラム名または別名は，データベースの仕様に依存します。

デフォルト値

なし

8.22.20 @GeneratedValue

(1) 説明

プライマリキーカラムにユニークな値を自動で生成，付与する方法を指定するアノテーションです。@Id を持つエンティティクラスまたはマップドスーパークラスのプライマリキーのフィールドまたはプロパティに適用します。

プライマリキー値の生成方法には，次の 4 種類の方法があります。なお，選択する生成方法に基づいて，あらかじめ基礎テーブルやデータベースシーケンスオブジェクトを用意しておく必要があります。それぞれの生成方法の詳細は，strategy 属性の説明を参照してください。

- GenerationType.AUTO
- GenerationType.IDENTITY
- GenerationType.SEQUENCE
- GenerationType.TABLE

適用可能要素は，メソッドとフィールドです。

(2) 属性

@GeneratedValue の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
strategy	任意	エンティティクラスのプライマリキー値を生成する方法を指定する属性です。
generator	任意	使用する@SequenceGenerator または@TableGenerator で設定される name 属性を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) strategy 属性

型

GenerationType

説明

エンティティクラスのプライマリキー値を生成する方法を指定する属性です。

指定できる値は，次の 4 種類です。

- GenerationType.AUTO

  データベースごとに最も適切な手順を選択して，プライマリキー値を生成します。

  データベースが Oracle または HiRDB の場合は，GenerationType.TABLE と同じ処理をします。

- GenerationType.IDENTITY

  データベースの identity 列を利用して，プライマリキー値を生成します。

  データベースが Oracle の場合は，GenerationType.SEQUENCE と同じ処理をします。

  データベースが HiRDB の場合は，GenerationType.TABLE と同じ処理をします。

- GenerationType.SEQUENCE

  データベースのシーケンスオブジェクトを使用して，プライマリキー値を生成します。

  データベースが HiRDB の場合は，GenerationType.TABLE と同じ処理をします。

- GenerationType.TABLE

  プライマリキー値を保持しておくためのテーブルを使用して，プライマリキー値を生成します。

デフォルト値

GenerationType.AUTO

(b) generator 属性

型

String

説明

使用する@SequenceGenerator または@TableGenerator で設定される name 属性を指定する属性です。

デフォルト値

strategy 属性の値によって，次の名前が仮定されます。

- GenerationType.AUTO の場合
  "SEQ_GEN"
- GenerationType.SEQUENCE の場合
  "SEQ_GEN_SEQUENCE"
- GenerationType.TABLE の場合
  "SEQ_GEN_TABLE"

8.22.21 @Id

(1) 説明

エンティティクラスのプライマリキーのプロパティまたはフィールドであることを示すアノテーションです。

@Id は，エンティティクラスまたはマップドスーパークラスで適用されます。

@Id を指定したフィールドまたはプロパティに対してマップされたデータベース上のカラムは，プライマリテーブルのプライマリキーカラムであると仮定されます。プライマリキーカラムのカラム名を@Column を用いて指定していない場合，プライマリキーカラムのカラム名は@Id を指定したフィールドまたはプロパティの名前になります。

なお，@Id を指定したフィールドに@Version を指定した場合は，@Id が無効になります。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Id の属性はありません。

8.22.22 @IdClass

(1) 説明

エンティティクラスの複数のフィールドまたはプロパティへマップされた複合プライマリキークラスを指定するアノテーションです。

このアノテーションは，マップドスーパークラスまたはエンティティクラスに適用されます。

エンティティクラスのプライマリキーのフィールドまたはプロパティと，複合プライマリキークラスのフィールドまたはプロパティの名前と型は一致させる必要があります。このアノテーションで指定した名前および型は，@Id が付いたエンティティのプライマリキーのプロパティまたはフィールドの型および名前に対応させてください。

適用可能要素は，クラスです。

(2) 属性

@IdClass の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	複合プライマリキークラスを指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

Class

説明

複合プライマリキークラスを指定する属性です。

指定できる値は，クラス名です。

デフォルト値

なし

8.22.23 @Inheritance

(1) 説明

エンティティの継承階層で使われる継承マッピング戦略を指定するアノテーションです。

@Inheritance は継承階層の親であるエンティティクラスに指定されます。

CJPA プロバイダで使用できる継承マッピング戦略には，次の 2 種類があります。

- SINGLE_TABLE（クラス階層ごとの単体テーブル）
- JOINED（結合サブクラス戦略）

継承マッピング戦略については，「8.13.2 継承マッピング戦略」を参照してください。

適用可能要素は，クラスです。

(2) 属性

@Inheritance の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
strategy	任意	継承マッピング戦略の種類を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) strategy 属性

型

InheritanceType

説明

エンティティで使用する継承マッピング戦略の種類を指定する属性です。

指定できる値は，次の 2 種類です。

- InheritanceType.SINGLE_TABLE：継承階層にあるすべてのクラスを一つのテーブルにマッピングする戦略です。
- InheritanceType.JOINED：継承階層の最上位（親クラス）は単一の表にマッピングされ，各サブクラスはサブクラス特有のマッピングをする戦略です。

デフォルト値

InheritanceType.SINGLE_TABLE

8.22.24 @JoinColumn

(1) 説明

エンティティクラス間の関連づけをする場合に，結合表のための外部キーカラムまたは外部キーカラムによって参照された，結合先テーブルのカラム名を指定するアノテーションです。必ず結合先テーブルのプライマリキーとなっているカラムを指定してください。

複数の外部キーカラムがある場合は@JoinColumns を使用し，それぞれの関連に対して@JoinColumn を指定してください。なお，複数の@JoinColumn を指定した場合は，name 属性および referencedColumnName 属性をそれぞれのアノテーションで指定してください。

@JoinColumn を明示的に指定しない場合，関連を示す永続化プロパティまたは永続化フィールドに単体の外部キーカラムを指定しているように扱われます。また，@JoinColumn の各属性値にデフォルト値が適用されます。

また，一つのカラムがフィールドに対して行った変更と，カスケード操作の関連づけによる変更が同時に実施されたことによって，整合性が取れなくなる場合があります。このため，name 属性および referencedColumnName 属性に指定したカラムをエンティティのフィールドに定義する場合は，insertable 属性および updatable 属性を false に指定する必要があります。この指定によって，フィールドに対して行った変更だけがデータベースに反映されますが，カスケード操作の関連づけによる変更はデータベースに反映されません。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@JoinColumn の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	対象テーブルを結合するために使用する外部キーカラム名を指定する属性です。
referencedColumnName	任意	name 属性で指定した外部キーカラムによって参照された結合先テーブルのカラム名を指定する属性です。
unique	任意	プロパティがユニークキーであるかどうかを指定する属性です。 なお，この属性は，CJPA プロバイダでは対応していません。
nullable	任意	データベースのカラムに null 値を指定できるかどうかを指定する属性です。 なお，この属性は，CJPA プロバイダでは対応していません。
insertable	任意	@JoinColumn で指定したカラムを SQL の INSERT 文に含むかどうか指定する属性です。
updatable	任意	@JoinColumn で指定したカラムを SQL の UPDATE 文に含むかどうか指定する属性です。
columnDefinition	任意	CREATE 文を出力する場合，外部カラムに追加する規約を DDL で記載する属性です。 なお，この属性は，CJPA プロバイダでは対応していません。
table	任意	外部キーカラムを含むテーブル名を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

対象テーブルを結合するために使用する外部キーカラム名を指定する属性です。

外部キーカラムは，エンティティのリレーションシップの種別ごとに存在する個所が異なります。エンティティのリレーションシップの種別ごとに外部キーカラムの存在個所を示します。

- OneToOne リレーションシップまたは ManyToOne リレーションシップの場合
  自エンティティのテーブル内
- ManyToMany リレーションシップの場合
  @JoinTable の結合表内

なお，指定できる値は，データベースのカラム名の仕様に依存します。

デフォルト値

- 自エンティティ内に単体の外部キーカラムが指定され，name 要素の値に何も指定しない場合
  <自エンティティ内の関連プロパティまたはフィールドの名前>_<参照されるプライマリキー列の名前>
- 参照している関連プロパティやフィールドがない場合（例：@JoinTable を使用している場合）
  <参照しているエンティティの名前>_<参照されるプライマリキー列の名前>

(b) referencedColumnName 属性

型

String

説明

name 属性で指定した外部キーカラムによって参照された結合先テーブルのカラム名を指定する属性です。

結合先テーブルのカラム名は次の個所に存在します。

- リレーションシップのアノテーションを使った場合
  参照されるテーブル内
- @JoinTable で使った場合
  所有者側エンティティのエンティティテーブル内

注

結合が逆結合の一部で定義される場合は，被所有者側のエンティティクラスのテーブル内になります。

なお，指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

外部キーによって参照されるテーブルのプライマリキーのカラム名

注

単体の外部キーカラムが指定される場合は，デフォルトを適用します。

(c) insertable 属性

型

boolean

説明

@JoinColumn で指定したカラムを SQL の INSERT 文に含むかどうかを指定する属性です。指定できる値は，true または false です。

それぞれの値の意味は次のとおりです。

true：@JoinColumn で指定したカラムを SQL の INSERT 文に含みます。

false：@JoinColumn で指定したカラムを SQL の INSERT 文に含みません。

デフォルト値

true

(d) updatable 属性

型

boolean

説明

@JoinColumn で指定したカラムを SQL の UPDATE 文に含むかどうかを指定する属性です。指定できる値は，true または false です。

それぞれの値の意味は次のとおりです。

true：@JoinColumn で指定したカラムを SQL の UPDATE 文に含みます。

false：@JoinColumn で指定したカラムを SQL の UPDATE 文に含みません。

デフォルト値

true

(e) table 属性

型

String

説明

外部キーカラムを含むテーブル名を指定する属性です。

指定できるテーブル名は，データベースの仕様に依存します。

デフォルト値

プライマリテーブル名

8.22.25 @JoinColumns

(1) 説明

同じ関連を示す@JoinColumn を複数同時に記述する場合に指定するアノテーションです。また，@JoinColumns は複合外部キーのマッピングを定義します。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@JoinColumns の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@JoinColumn の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

JoinColumn[]

説明

@JoinColumn の配列を指定する属性です。

指定できる値は，@JoinColumn の配列で指定できる範囲です。

デフォルト値

なし

8.22.26 @JoinTable

(1) 説明

次のクラスに設定する結合表を指定するアノテーションです。

- ManyToMany リレーションシップを指定する場合の所有者側のクラス
- 片方向の OneToMany リレーションシップを持つクラス

name 属性が指定されない場合，結合表の名前は次の名称になります。

<所有者側テーブル名>_<被所有者側テーブル名>

適用可能要素は，メソッドとフィールドです。

(2) 属性

@JoinTable の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	結合表のテーブル名を指定する属性です。
catalog	任意	結合表のテーブルのカタログ名を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
schema	任意	結合表のテーブルのスキーマ名を指定する属性です。
joinColumns	任意	所有者側エンティティのプライマリテーブルを参照する，結合表の外部キーカラムを指定する属性です。
inverseJoinColumns	任意	被所有者側エンティティのプライマリテーブルを参照する，結合表の外部キーカラムを指定する属性です。
uniqueConstraints	任意	テーブルのユニーク規制を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

結合表のテーブル名を指定する属性です。

指定できるテーブル名は，データベースの仕様に依存します。

デフォルト値

<所有者側テーブル名>_<被所有者側テーブル名>

(b) schema 属性

型

String

説明

テーブルのスキーマ名を指定する属性です。

指定できる値は，データベースのスキーマ名の仕様に依存します。

デフォルト値

使用するデータベースのデフォルトのスキーマ

(c) joinColumns 属性

型

JoinColumn[]

説明

所有者側エンティティのプライマリテーブルを参照する，結合表の外部キーカラムを指定する属性です。@JoinColumn の配列を指定します。@JoinColumn の name 属性には結合表の外部キーカラム名，referencedColumnName 属性には所有者側の参照カラム名をそれぞれ指定します。

指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

@JoinColumn の外部キー

(d) inverseJoinColumns 属性

型

JoinColumn[]

説明

被所有者側エンティティのプライマリテーブルを参照する，結合表の外部キーカラムを指定する属性です。@JoinColumn の配列を指定します。@JoinColumn の name 属性には結合表の外部キーカラム名，referencedColumnName 属性には外部キーカラムで参照された結合先テーブルのカラムをそれぞれ指定します。

指定できる値は，データベースのカラム名の仕様に依存します。

デフォルト値

@JoinColumn の外部キーカラム

8.22.27 @Lob

(1) 説明

データベースがサポートしている large オブジェクト型の永続化フィールドまたは永続化プロパティであることを指定するアノテーションです。@Basic とともに使用できます。

@Lob にはバイナリ型（Blob）とキャラクタ型（Clob）があります。@Lob の型は，永続化フィールドまたは永続化プロパティの型によって決まります。文字列およびキャラクタ型の場合は Clob になり，そのほかの場合は Blob になります。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Lob の属性はありません。

8.22.28　@ManyToMany

(1)　説明

ManyToMany リレーションシップの関係を持つ所有者側のエンティティクラスから被所有者側のエンティティクラスへの複数の関連を指定するアノテーションです。

ManyToMany リレーションシップは，双方向，単方向に関係なく，所有者側と被所有者側を持ちます。関係が双方向の場合，結合表の指定はどちらの側でも指定できます。

なお，Generics を使用して Collection 要素型が指定されている場合，被所有者側のエンティティクラスを指定する必要はありません。そのほかの場合は，必ず指定してください。

また，@ManyToMany を指定した場合は，次に示すアノテーションの設定に注意してください。

- @OneToMany のための同じアノテーションの属性は，@ManyToMany と同じ属性を持ちます。
- 所有者側と被所有者側のクラスで@ManyToMany を定義したプロパティまたはフィールドが同じ名前の場合は，@JoinTable のデフォルト設定（joinColumns 属性，inverseJoinColumns 属性の指定がない状態）を使用しないでください。
- 双方向関係の場合には，所有者側の情報を基に結合表の値が更新されます。mappedBy 属性を指定したエンティティクラスでマッピング情報を変更しても，その情報は結合表には反映されません。

適用可能要素は，メソッドとフィールドです。

(2)　属性

@ManyToMany の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
targetEntity	任意	被所有者側のエンティティクラスを指定する属性です。
cascade	任意	カスケード対象となるオペレーションを指定する属性です。
fetch	任意	フェッチ戦略の指定値を指定する属性です。
mappedBy	任意	被所有者側のエンティティクラスの要素に付与して，所有者側のエンティティクラスで関係を保持しているフィールドまたはプロパティの名前を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a)　targetEntity 属性

型

Class

説明

被所有者側のエンティティクラスを指定する属性です。

Generics を使って定義されているコレクションプロパティの場合は，任意で指定します。それ以外の場合は必ず指定してください。

デフォルト値

コレクションのパラメタ化された型

注　Generics を使って定義されているときにだけ設定されます。

(b)　cascade 属性

型

CascadeType[]

説明

カスケード対象となるオペレーションを指定する属性です。

指定できる値を次に示します。

- CascadeType.ALL：所有者側のエンティティクラスの persist，remove，merge，refresh の操作が関連先にカスケードされます。
- CascadeType.MERGE：所有者側のエンティティクラスの merge 操作が関連先にカスケードされます。
- CascadeType.PERSIST：所有者側のエンティティクラスの persist 操作が関連先にカスケードされます。
- CascadeType.REFRESH：所有者側のエンティティクラスの refresh 操作が関連先にカスケードされます。
- CascadeType.REMOVE：所有者側のエンティティクラスの remove 操作が関連先にカスケードされます。

デフォルト値

カスケード対象なし

(c)　fetch 属性

型

FetchType

説明

データベースからのデータのフェッチ戦略を定義する属性です。フェッチ戦略については，「8.4.5 データベースとの同期」を参照してください。

指定できる値は，次の 2 種類です。

- EAGER 戦略：データが EAGER にフェッチされなければならない要求
- LAZY 戦略：データが最初にアクセスされるときに，LAZY にフェッチされる要求

デフォルト値

FetchType.LAZY

(d) mappedBy 属性

型

String

説明

被所有者側のエンティティクラスの要素に付与し，所有者側のエンティティクラスで関係を保持しているフィールドまたはプロパティの名前を指定する属性です。

この属性を指定した場合，関係は双方向になります。双方向関係の場合には所有者側の情報を基に結合表の値は更新されます。被所有者側のエンティティクラス（mappedBy 属性を指定したエンティティクラス）でマッピング情報を変更しても，その情報は結合表には反映されません。

デフォルト値

なし

8.22.29 @ManyToOne

(1) 説明

@ManayToOne が指定されたクラスが ManyToOne リレーションシップであることを示し，所有者側のエンティティクラスから被所有者側のエンティティクラスへの関連を指定するアノテーションです。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@ManyToOne の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
targetEntity	任意	被所有者側のエンティティクラスを指定する属性です。
cascade	任意	カスケード対象となるオペレーションを指定する属性です。
fetch	任意	フェッチ戦略の指定値を指定する属性です。
optional	任意	すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を設定できるかどうかを指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) targetEntity 属性

型

Class

説明

被所有者側のエンティティクラスを指定する属性です。

デフォルト値

アノテーションが付与されているフィールドやプロパティの型

(b) cascade 属性

型

CascadeType[]

説明

カスケード対象となるオペレーションを指定する属性です。

指定できる値を次に示します。

- CascadeType.ALL：所有者側のエンティティクラスの persist，remove，merge，および refresh の操作が関連先にカスケードされます。
- CascadeType.MERGE：所有者側のエンティティクラスの merge 操作が関連先にカスケードされます。
- CascadeType.PERSIST：所有者側のエンティティクラスの persist 操作が関連先にカスケードされます。
- CascadeType.REFRESH：所有者側のエンティティクラスの refresh 操作が関連先にカスケードされます。
- CascadeType.REMOVE：所有者側のエンティティクラスの remove 操作が関連先にカスケードされます。

デフォルト値

カスケード対象なし

(c) fetch 属性

型

FetchType

説明

データベースからのデータのフェッチ戦略を定義する属性です。フェッチ戦略については，「8.4.5 データベースとの同期」を参照してください。

指定できる値は，次の 2 種類です。

- EAGER 戦略：データが EAGER にフェッチされなければならない要求

- **LAZY 戦略**：データが最初にアクセスされるときに，LAZY にフェッチされる要求

デフォルト値

FetchType.EAGER

(d) optional 属性

型

boolean

説明

すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を指定できるかどうかを指定する属性です。指定できる値は，次のとおりです。

- **true**：すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を設定できます。
- **false**：すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を指定できません。

デフォルト値

true

8.22.30 @MapKey

(1) 説明

OneToMany リレーションシップまたは ManyToMany リレーションシップで被所有者側のエンティティクラスが java.util.Map 型で示される場合に，マップ内のオブジェクト識別に用いられるマップキーを指定するアノテーションです。

name 属性が指定されない場合，関連づけられたエンティティのプライマリキーはマップキーとして使われます。

プライマリキーが複合プライマリキーである場合に@IdClass としてマップされるときは，マップキーに複合プライマリキーを使います。

プライマリキー以外の永続化フィールドまたは永続化プロパティがマップキーとして使われる場合は，そのマップキーと関連したユニークキー制約を持つことができます。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@MapKey の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	マップキーとして使われる被所有者側のエンティティクラスの永続化フィールドまたは永続化プロパティの名前を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

マップキーとして使われる被所有者側のエンティティクラスの永続化フィールドまたは永続化プロパティの名前を指定する属性です。

デフォルト値

被所有者側のエンティティクラスのプライマリキーフィールドまたはプロパティの名前

8.22.31 @MappedSuperclass

(1) 説明

マップドスーパークラスであることを指定するアノテーションです。

マップドスーパークラスは，継承するためのクラスであるため，このクラスに対応するテーブルを持ちません。マップドスーパークラスは，サブクラスへのマッピングと関連マッピング情報が継承されることを除いて，エンティティと同じ方法でテーブルにマップされます。

@AttributeOverride を使うことでマッピング情報をサブクラスでオーバーライドできます。

適用可能要素は，クラスです。

(2) 属性

@MappedSuperclass の属性はありません。

8.22.32 @NamedNativeQueries

(1) 説明

@NamedNativeQuery を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラスです。

(2) 属性

@NamedNativeQueries の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@NamedNativeQuery の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

NamedNativeQuery[]

説明

@NamedNativeQuery の配列を指定する属性です。

指定できる値は，@NamedNativeQuery 配列で指定できる範囲です。詳細は，「8.22.33 @NamedNativeQuery」を参照してください。

デフォルト値

なし

8.22.33 @NamedNativeQuery

(1) 説明

SQL で名前付きクエリを指定するアノテーションです。エンティティクラスとマップドスーパークラスに適用できます。

適用可能要素は，クラスです。

(2) 属性

@NamedNativeQuery の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	名前付きクエリの名前を指定する属性です。
query	必須	SQL の文字列を指定する属性です。
hints	任意	@QueryHint の配列を指定する属性です。
resultClass	任意	SQL の結果を反映するクラスを指定する属性です。

属性名	任意/必須	属性の説明
resultSetMapping	任意	@SqlResultSetMapping の name 属性で指定した名前を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

名前付きクエリの名前を指定する属性です。

指定できる値は，文字列です。

デフォルト値

なし

(b) query 属性

型

String

説明

SQL の文字列を指定する属性です。

指定できる SQL は，使用するデータベースの仕様に依存します。

デフォルト値

なし

(c) hints 属性

型

QueryHint[]

説明

@QueryHint の配列を指定する属性です。

指定できる値は，@QueryHint の配列で指定できる範囲です。詳細は，「8.22.48　@QueryHint」を参照してください。

デフォルト値

空の配列

(d) resultClass 属性

型

Class

説明

SQL の結果を反映するクラスを指定する属性です。

resultClass 属性は，クエリの実行結果をマッピングしたいクラスがあるときに指定します。resultClass 属性と resultSetMapping 属性は同時に指定しないでください。

指定できる値は，クラス名です。

デフォルト値

void.class

(e) resultSetMapping 属性

型

String

説明

結果セットを定義した@SqlResultSetMapping の name 属性で指定した名前を指定する属性です。

任意の結果セットに SQL の結果をマッピングしたいときに指定します。

resultClass 属性と resultSetMapping 属性は同時に指定しないでください。

指定できる範囲は，@SqlResultSetMapping の name 属性で指定できる範囲です。詳細は，「8.22.52(2)(a) name 属性」を参照してください。

デフォルト値

空の文字列

8.22.34 @NamedQueries

(1) 説明

@NamedQuery を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラスです。

(2) 属性

@NamedQueries の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@NamedQuery の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

NamedQuery[]

説明

@NamedQuery の配列を指定する属性です。

指定できる値は，@NamedQuery の配列で指定できる範囲です。詳細は，「8.22.35 @NamedQuery」を参照してください。

デフォルト値

なし

8.22.35 @NamedQuery

(1) 説明

JPQL の名前付きクエリを指定するアノテーションです。エンティティクラスとマップドスーパークラスに適用できます。

適用可能要素は，クラスです。

(2) 属性

@NamedQuery の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	名前付きクエリ名を指定する属性です。
query	必須	JPQL のクエリ文字列を指定する属性です。
hints	任意	@QueryHint の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

名前付きクエリ名を指定する属性です。

指定できる値は，文字列です。

デフォルト値

なし

(b) query 属性

型

String

説明

JPQL のクエリ文字列を指定する属性です。

指定できる値は，JPQL の仕様に依存します。

デフォルト値

なし

(c) hints 属性

型

QueryHint[]

説明

@QueryHint の配列を指定する属性です。

指定できる値は，@QueryHint の配列で指定できる範囲です。詳細は，「8.22.48 @QueryHint」を参照してください。

デフォルト値

空の配列

8.22.36 @OneToMany

(1) 説明

OneToMany リレーションシップの関係を持つ所有者側のエンティティクラスから被所有者側のエンティティクラスへの複数の関連を指定するアノテーションです。

@OneToMany のための同じアノテーションの属性は，@ManyToMany と同じ属性を持ちます。

なお，Generics を使用して Collection 要素型が指定されている場合，被所有者側のエンティティクラスを指定する必要はありません。そのほかの場合は，必ず指定してください。

また，双方向の関係にする場合は，非所有者側に必ず mappedBy 属性を指定してください。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@OneToMany の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
targetEntity	任意	被所有者側のエンティティクラスを指定する属性です。
cascade	任意	カスケード対象となるオペレーションを指定する属性です。
fetch	任意	フェッチ戦略の指定値を指定する属性です。
mappedBy	任意	被所有者側のエンティティクラスの要素に付与し，所有者側のエンティティクラスで関係を保持しているフィールドまたはプロパティの名前を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) targetEntity 属性

型

Class

説明

被所有者側のエンティティクラスを指定する属性です。

Generics を使って定義されているコレクションプロパティの場合は，任意で指定します。それ以外の場合は必ず指定してください。

デフォルト値

コレクションのパラメタ化された型

注 Generics を使って定義されているときにだけ設定されます。

(b) cascade 属性

型

CascadeType[]

説明

カスケード対象となるオペレーションを指定する属性です。

指定できる値を次に示します。

- CascadeType.ALL：所有者側のエンティティクラスの persist，remove，merge，および refresh の操作が関連先にカスケードされます。
- CascadeType.MERGE：所有者側のエンティティクラスの merge 操作が関連先にカスケードされます。
- CascadeType.PERSIST：所有者側のエンティティクラスの persist 操作が関連先にカスケードされます。

- CascadeType.REFRESH：所有者側のエンティティクラスの refresh 操作が関連先にカスケードされます。
- CascadeType.REMOVE：所有者側のエンティティクラスの remove 操作が関連先にカスケードされます。

デフォルト値

カスケード対象なし

(c) fetch 属性

型

FetchType

説明

データベースからのデータのフェッチ戦略を定義する属性です。フェッチ戦略については，「8.4.5 データベースとの同期」を参照してください。

指定できる値は，次の 2 種類です。

- EAGER 戦略：データが EAGER にフェッチされなければならない要求
- LAZY 戦略：データが最初にアクセスされるときに，LAZY にフェッチされる要求

デフォルト値

FetchType.LAZY

(d) mappedBy 属性

型

String

説明

被所有者側のエンティティクラスの要素に付与し，所有者側のエンティティクラスで関係を保持しているフィールドまたはプロパティの名前を指定する属性です。

この属性を指定した場合，関係は双方向になります。

デフォルト値

なし

8.22.37 @OneToOne

(1) 説明

指定されたクラスが OneToOne リレーションシップであることを示し，エンティティクラス間の一つの関連を指定するアノテーションです。

双方向の関係にする場合は，非所有者側に必ず mappedBy 属性を指定してください。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@OneToOne の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
targetEntity	任意	被所有者側のエンティティクラスを指定する属性です。
cascade	任意	カスケード対象となるオペレーションを指定する属性です。
fetch	任意	フェッチ戦略の指定値を指定する属性です。
optional	任意	すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を設定できるかどうかを指定する属性です。
mappedBy	任意	被所有者側のエンティティクラスの要素に付与し，所有者側のエンティティクラスで関係を保持しているフィールド名を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) targetEntity 属性

型

Class

説明

被所有者側のエンティティクラスを指定する属性です。

デフォルト値

アノテーションが付与されているフィールドやプロパティの型

(b) cascade 属性

型

CascadeType[]

説明

カスケード対象となるオペレーションを指定する属性です。

指定できる値を次に示します。

- CascadeType.ALL：所有者側のエンティティクラスの persist，remove，merge，refresh の操作が関連先にカスケードされます。
- CascadeType.MERGE：所有者側のエンティティクラスの merge 操作が関連先にカスケードされます。

- CascadeType.PERSIST：所有者側のエンティティクラスの persist 操作が関連先にカスケードされます。
- CascadeType.REFRESH：所有者側のエンティティクラスの refresh 操作が関連先にカスケードされます。
- CascadeType.REMOVE：所有者側のエンティティクラスの remove 操作が関連先にカスケードされます。

デフォルト値

カスケード対象なし

(c) fetch 属性

型

FetchType

説明

データベースからのデータのフェッチ戦略を定義する属性です。フェッチ戦略については，「8.4.5 データベースとの同期」を参照してください。

指定できる値は，次の 2 種類です。

- EAGER 戦略：データが EAGER にフェッチされなければならない要求
- LAZY 戦略：データが最初にアクセスされるときに，LAZY にフェッチされる要求

デフォルト値

FetchType.EAGER

(d) optional 属性

型

boolean

説明

すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を指定できるかどうかを指定する属性です。指定できる値は，次のとおりです。

- true：すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を設定できます。
- false：すべての非プリミティブ型のフィールドおよびプロパティの値に null 値を指定できません。

デフォルト値

true

(e) mappedBy 属性

型

String

説明

被所有者側のエンティティクラスの要素に付与し，所有者側のエンティティクラスで関係を保持しているフィールド名を指定します。指定した場合，関係は双方向になります。

デフォルト値

なし

8.22.38 @OrderBy

(1) 説明

エンティティの情報を取得するとき，コレクションに保持される順番を指定するアノテーションです。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@OrderBy の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	任意	プライマリキー以外のフィールドまたはプロパティを基にした順番でエンティティを取得したい場合に指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

String

説明

プライマリキー以外のフィールドまたはプロパティを基にした順番でエンティティクラスを取得するときに指定する属性です。コンマ区切りで，順番を指定したいフィールドまたはプロパティを指定します。

取得する順番は，フィールドまたはプロパティのあとに指定します。指定できる値は次のとおりです。指定しなかった場合は，昇順になります。

- ASC：昇順
- DESC：降順

value 属性内で指定されるフィールドまたはプロパティには，比較演算できる値が格納されているカラムを指定します。

デフォルト値

エンティティクラスのプライマリキーによる昇順

8.22.39　@PostLoad

(1)　説明

キャッシュからエンティティを読み込んだあと，またはデータベースに SELECT 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2)　属性

@PostLoad の属性はありません。

8.22.40　@PostPersist

(1)　説明

データベースに INSERT 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2)　属性

@PostPersist の属性はありません。

8.22.41　@PostRemove

(1)　説明

データベースに DELETE 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2)　属性

@PostRemove の属性はありません。

8.22.42 @PostUpdate

(1) 説明

データベースに UPDATE 文を発行したあとに呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2) 属性

@PostUpdate の属性はありません。

8.22.43 @PrePersist

(1) 説明

データベースに INSERT 文を発行する前に呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2) 属性

@PrePersist の属性はありません。

8.22.44 @PreRemove

(1) 説明

データベースに DELETE 文を発行する前に呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2) 属性

@PreRemove の属性はありません。

8.22.45　@PreUpdate

(1)　説明

データベースに UPDATE 文を発行する前に呼び出されるコールバックメソッドであることを示すアノテーションです。エンティティクラス，マップドスーパークラス，またはエンティティリスナクラスのメソッドに適用されます。

適用可能要素は，メソッドです。

(2)　属性

@PreUpdate の属性はありません。

8.22.46　@PrimaryKeyJoinColumn

(1)　説明

ほかのテーブルと結合する場合に外部キーとして使われるカラムを指定するアノテーションです。次の場合に使用します。

- 継承マッピング戦略の JOINED 戦略で，エンティティのスーパークラスのプライマリキーとサブクラスのプライマリキーが異なる名前の場合
- @SecondaryTable で，プライマリテーブルとセカンダリテーブルを結合する場合※
- OneToOne リレーションシップで，被所有者側のエンティティクラスのプライマリキーが外部キーとして使用されている場合

注※

この場合は，@SecondaryTable 内で使用します。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2)　属性

@PrimaryKeyJoinColumn の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	対象テーブルを結合するためのカラム名を指定する属性です。
referencedColumnName	任意	name 属性で指定したカラムによって参照される結合先テーブルのプライマリキーのカラム名を指定する属性です。
columnDefinition	任意	CREATE 文を出力するときにカラムに付加する制約を DDL で記載する属性です。

属性名	任意/必須	属性の説明
		なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

対象テーブルを結合するためのカラム名を指定する属性です。

指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

- JOINED 戦略を使用した場合
  スーパークラスのプライマリテーブルのプライマリキーのカラム名
- @SecondaryTable を使用した場合
  プライマリテーブルのプライマリキーのカラム名
- OneToOne リレーションシップを使用した場合
  対象となるエンティティのテーブルのプライマリキーのカラム名

(b) referencedColumnName 属性

型

String

説明

name 属性で指定したカラムによって参照される結合先テーブルのプライマリキーのカラム名を指定する属性です。@Column の name 属性の文字列と同じ値を指定してください。なお，指定する文字列は大文字，小文字もそろえてください。

指定できるカラム名は，データベースの仕様に依存します。

OneToOne リレーションシップでカラムの指定をプライマリキーにしなくてもユニークキー制約があれば動作してしまいますが，動作の保証はしません。

デフォルト値

- JOINED 戦略を使用した場合
  スーパークラスのプライマリテーブルのプライマリキーのカラム名
- @SecondaryTable を使用した場合
  プライマリテーブルのプライマリキーのカラム名
- OneToOne リレーションシップを使用した場合

対象となるエンティティのテーブルのプライマリキーのカラム名

8.22.47 @PrimaryKeyJoinColumns

(1) 説明

@PrimaryKeyJoinColumn を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2) 属性

@PrimaryKeyJoinColumns の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@PrimaryKeyJoinColumn の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

PrimaryKeyJoinColumn[]

説明

@PrimaryKeyJoinColumn の配列を指定する属性です。

指定できる値は，@PrimaryKeyJoinColumn で指定できる範囲です。詳細は，「8.22.46 @PrimaryKeyJoinColumn」を参照してください。

デフォルト値

なし

8.22.48 @QueryHint

(1) 説明

データベース固有のクエリのヒントを指定するアノテーションです。

悲観的ロックやエンティティのキャッシュ機能の設定ができます。

適用可能要素は，@NamedQuery，または@NamedNativeQuery の hints 要素です。

(2) 属性

@QueryHint の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	ヒントの名前を指定する属性です。
value	必須	ヒントの値を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

使用するヒントの名前を指定する属性です。指定できる値は次のとおりです。

cosminexus.jpa.pessimistic-lock

悲観的ロックを使用するかどうかを指定するヒントの名前です。

デフォルト値

なし

(b) value 属性

型

String

説明

ヒントの値を指定する属性です。name 属性で指定したヒントの名前に合わせて，次の値を指定します。

name 属性で cosminexus.jpa.pessimistic-lock を指定した場合の指定値

- NoLock：悲観的ロックを使用しない場合に指定します。
- Lock：悲観的ロックを使用する場合に指定します。対象となるテーブルがすでにロックされている場合は，解放されるまで待ちます。このとき発行される SQL を使用するデータベースごとに示します。

  Oracle の場合：SELECT.... FOR UPDATE

  HiRDB の場合：SELECT....WITH EXCLUSIVE LOCK
- LockNoWait：悲観的ロックを使用する場合に指定します。対象となるテーブルがすでにロックされている場合は，例外が発生します。このとき発行される SQL を使用するデータベースごとに示します。

  Oracle の場合：SELECT.... FOR UPDATE NO WAIT

  HiRDB の場合：SELECT....WITH EXCLUSIVE LOCK NO WAIT

デフォルト値

name 属性で cosminexus.jpa.pessimistic-lock を指定した場合

NoLock

8.22.49 @SecondaryTable

(1) 説明

エンティティクラスにセカンダリテーブルを指定するアノテーションです。

エンティティクラスがデータベース上の複数のテーブルにわたってマッピングされる場合に指定します。

@SecondaryTable をエンティティクラス内で指定しない場合，エンティティクラスのすべての永続化プロパティまたは永続化フィールドは，プライマリテーブルで指定されたテーブルとマッピングします。

適用可能要素は，クラスです。

(2) 属性

@SecondaryTable の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	セカンダリテーブル名を指定する属性です。
catalog	任意	セカンダリテーブルのカタログ名を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
schema	任意	セカンダリテーブルのスキーマ名を指定する属性です。
pkJoinColumns	任意	セカンダリテーブルがプライマリテーブルと結合するために使用する外部キーカラムを指定する属性です。
uniqueConstraints	任意	テーブルでのユニークキー制約を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

セカンダリテーブル名を指定する属性です。

指定できるテーブル名は，データベースの仕様に依存します。

デフォルト値

なし

(b) schema 属性

型

String

説明

セカンダリテーブルのスキーマ名を指定する属性です。

指定できるスキーマ名は，データベースの仕様に依存します。

デフォルト値

使用するデータベースのデフォルトのスキーマ名

(c) pkJoinColumns 属性

型

PrimaryKeyJoinColumn[]

説明

セカンダリテーブルの外部キーカラムを指定する属性です。@PrimaryKeyJoinColumn の配列で指定します。

この要素が指定されない場合，セカンダリテーブルの外部キーカラムはプライマリテーブルのプライマリキーカラムと同じ名前と型を持ちます。このため，セカンダリテーブルはプライマリテーブルのプライマリキーカラムを参照します。

デフォルト値

指定できる値は，@PrimaryKeyJoinColumn の配列で指定できる範囲です。詳細は，「8.22.46 @PrimaryKeyJoinColumn」を参照してください。

8.22.50 @SecondaryTables

(1) 説明

@SecondaryTable を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラスです。

(2) 属性

@SecondaryTables の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@SecondaryTable の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

SecondaryTable[]

説明

@SecondaryTable の配列を指定する属性です。

指定できる値は，@SecondaryTable の配列で指定できる範囲です。詳細は，「8.22.49 @SecondaryTable」を参照してください。

デフォルト値

なし

8.22.51 @SequenceGenerator

(1) 説明

プライマリキーを作成するシーケンスジェネレータの設定を指定するアノテーションです。
@SequenceGenerator を使用する場合，次の設定が必要です。

- @GeneratedValue の strategy 属性に GenerationType.SEQUENCE を指定します。
- @GeneratedValue の generator 属性で指定された名前を@SequenceGenerator の name 属性に設定します。

シーケンスジェネレータは，エンティティクラス，またはプライマリキーのフィールドもしくはプロパティで指定されます。シーケンスジェネレータ名のスコープは永続化ユニットで有効です。

シーケンスオブジェクトを作成する場合，順序の番号間の増分間隔（INCREMENT BY）と生成する順序番号の初期値（START WITH）には，正の整数を指定します。初期値（START WITH）に 1 を指定した場合，プライマリキーは 1 から生成されます。負の値を指定した場合の動作は保証しません。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2) 属性

@SequenceGenerator の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	@GeneratedValue アノテーションの generator 属性で指定された名前を指定する属性です。
sequenceName	任意	既存のプライマリキー値または事前に定義したプライマリキー値を取得するためのデータベースシーケンスオブジェクトの名前を指定する属性です。
initialValue	任意	シーケンスオブジェクトが，プライマリキー値生成を開始する初期値を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。値を指定した場合は無視されます。
allocationSize	任意	シーケンスからプライマリキー値を割り当てるサイズを指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

@GeneratedValue アノテーションの generator 属性で指定された名前を指定する属性です。

指定できる値は，文字列です。

デフォルト値

なし

(b) sequenceName 属性

型

String

説明

既存のプライマリキー値または事前に定義したプライマリキー値を取得するためのデータベースシーケンスオブジェクトの名前を指定する属性です。

指定できるシーケンスオブジェクト名は，データベースの仕様に依存します。

デフォルト値

@GeneratedValue の generator 属性の指定値

(c) allocationSize 属性

型

int

説明

シーケンスからプライマリキー値を割り当てるサイズを指定する属性です。指定できるシーケンスオブジェクト名は，データベースの仕様に依存します。

指定できるサイズは，int 型の 1 以上の数値です。シーケンスオブジェクトの増分間隔と同じ値を指定します。異なる値を指定した場合の動作は保証しません。

なお，この属性は，実行時に利用する最大値を指定できます。シーケンス番号の管理領域として取得するため，大きな値を指定した場合は，実行時に java.lang.OutOfMemoryError 例外が発生します。

デフォルト値

50

8.22.52 @SqlResultSetMapping

(1) 説明

SQL のクエリの結果セットマッピングを指定するアノテーションです。

適用可能要素は，クラスです。

(2) 属性

@SqlResultSetMapping の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	結果セットマッピングの名前を指定する属性です。
entities	任意	@EntityResult の配列を指定する属性です。
columns	任意	@ColumnResult の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

結果セットマッピングの名前を指定する属性です。

指定できる値は，文字列です。

デフォルト値

なし

(b) entities 属性

型

EntityResult[]

説明

@EntityResult の配列を指定する属性です。

指定できる値は，@EntityResult の配列で指定できる範囲です。詳細は，「8.22.15 @EntityResult」を参照してください。

デフォルト値

空の配列

(c) columns 属性

型

ColumnResult[]

説明

@ColumnResult の配列を指定する属性です。

指定できる値は，@ColumnResult の配列で指定できる範囲です。詳細は，「8.22.7 @ColumnResult」を参照してください。

デフォルト値

空の配列

8.22.53 @SqlResultSetMappings

(1) 説明

@SqlResultSetMapping を複数同時に記述する場合に指定するアノテーションです。

適用可能要素は，クラスです。

(2) 属性

@SqlResultSetMappings の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	@SqlResultSetMapping の配列を指定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

SqlResultSetMapping[]

説明

@SqlResultSetMapping の配列を指定する属性です。

指定できる値は，@SqlResultSetMapping の配列で指定できる範囲です。詳細は，「8.22.52 @SqlResultSetMapping」を参照してください。

デフォルト値

なし

8.22.54 @Table

(1) 説明

エンティティクラスに，プライマリテーブルを指定するアノテーションです。

明示的にエンティティクラスに@Table を指定しない場合でも，@Table が指定されたようにエンティティクラスは扱われます。その場合，@Table の各属性値にはデフォルト値が適用されます。

エンティティがマッピングするテーブルを複数指定する場合は，@SecondaryTable または @SecondaryTables を使用してください。

適用可能要素は，クラスです。

(2) 属性

@Table の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	任意	テーブル名を指定する属性です。
catalog	任意	テーブルのカタログ名を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
schema	任意	テーブルのスキーマ名を指定する属性です。
uniqueConstraints	任意	テーブルでのユニークキー制約を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

テーブル名を指定する属性です。

指定できるテーブル名は，データベースの仕様に依存します。

デフォルト値

エンティティ名

(b) schema 属性

型

String

説明

テーブルのスキーマ名を指定する属性です。

指定できるスキーマ名は，データベースの仕様に依存します。

デフォルト値

使用するデータベースのデフォルトのスキーマ名

8.22.55 @TableGenerator

(1) 説明

プライマリキーを作成するジェネレータの設定を指定するアノテーションです。

@TableGenerator を使用する場合，次の設定が必要です。

- @GeneratedValue の strategy 属性に GenerationType.TABLE を指定します。
- @GeneratedValue の generator 属性で指定された名前を@TableGenerator の name 属性に設定します。

テーブルジェネレータは，エンティティクラス，またはプライマリキーのフィールドまたはプロパティに指定されます。ジェネレータ名のスコープは永続化ユニットで有効です。

エンティティはプライマリキー値を生成するために，ジェネレータテーブルの行を使用します。

シーケンスを管理するテーブルを作成する場合，初期値には正の整数を指定します。初期値に 0 を指定した場合，プライマリキーは 1 から生成されます。

適用可能要素は，クラス，メソッド，およびフィールドです。

(2) 属性

@TableGenerator の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
name	必須	プライマリキー値のためのジェネレータ名を指定する属性です。
table	任意	生成されるプライマリキー値を確保するテーブルの名前を指定する属性です。
catalog	任意	生成されるプライマリキー値を確保するテーブルのカタログ名を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。
schema	任意	生成されるプライマリキー値を確保するテーブルのスキーマ名を指定する属性です。
pkColumnName	任意	生成されるプライマリキー値を確保するテーブルのプライマリキーカラム名を指定する属性です。
valueColumnName	任意	生成された最終値を確保するカラム名を指定する属性です。
pkColumnValue	任意	生成されるプライマリキー値を確保するテーブルのプライマリキー値を指定する属性です。
initialValue	任意	生成された最新の値を確保するカラムを初期化するために使う値を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。値を指定した場合は無視されます。
allocationSize	任意	ジェネレータからプライマリキー値を割り当てるサイズを指定する属性です。
uniqueConstraints	任意	生成されるプライマリキー値を確保するテーブル上でのユニークキー制約を指定する属性です。 なお，この属性は，CJPA プロバイダには対応していません。値を指定した場合は無視されます。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) name 属性

型

String

説明

プライマリキー値のためのジェネレータ名を指定する属性です。

指定できる値は，文字列です。

デフォルト値

なし

(b) table 属性

型

String

説明

生成されるプライマリキー値を確保するテーブルの名前を指定する属性です。

指定できるテーブル名は，データベースの仕様に依存します。

デフォルト値

"SEQUENCE"

(c) schema 属性

型

String

説明

生成されるプライマリキー値を確保するテーブルのスキーマ名を指定する属性です。

指定できるスキーマ名は，データベースの仕様に依存します。

デフォルト値

使用するデータベースのデフォルトのスキーマ名

(d) pkColumnName 属性

型

String

説明

生成されるプライマリキー値を確保するテーブルのプライマリキーカラム名を指定する属性です。

指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

"SEQ_NAME"

(e) valueColumnName 属性

型

String

説明

生成された最終値を確保するカラム名を指定する属性です。

指定できるカラム名は，データベースの仕様に依存します。

デフォルト値

"SEQ_COUNT"

(f) pkColumnValue 属性

型

String

説明

生成されるプライマリキー値を確保するテーブルのプライマリキー値を指定する属性です。

指定できる値は，生成されたプライマリキーのカラムの型に依存します。

デフォルト値

name 属性で指定された文字列

(g) allocationSize 属性

型

int

説明

ジェネレータからプライマリキー値を割り当てるサイズを指定する属性です。

指定できる値は，int 型の 1 以上の数値です。

なお，この属性は，実行時に利用する最大値を指定できます。シーケンス番号の管理領域として取得するため，大きな値を指定した場合は，実行時に java.lang.OutOfMemoryError 例外が発生します。

デフォルト値

50

8.22.56 @Temporal

(1) 説明

時刻を表す型（java.util.Date および java.util.Calendar）を持つ永続化プロパティまたは永続化フィールドに指定するアノテーションです。@Basic とともに使用できます。

ただし，@Version と@Temporal は同時に指定できません。どちらかのアノテーションだけを指定してください。

適用可能要素は，メソッドとフィールドです。

(2) 属性

@Temporal の属性の一覧を次の表に示します。

属性名	任意/必須	属性の説明
value	必須	データベース型に対応する TemporalType 列挙型に設定する属性です。

CJPA プロバイダで対応する属性の詳細を次に示します。

(a) value 属性

型

TemporalType

説明

データベース型に対応する TemporalType 列挙型に設定する属性です。

指定できる値は，次の 3 種類です。

- TemporalType.DATE：java.sql.Data と同じです。
- TemporalType.TIME：java.sql.Time と同じです。
- TemporalType.TIMESTAMP：java.sql.Timestamp と同じです。

デフォルト値

なし

8.22.57 @Transient

(1) 説明

次に示す永続化しないクラスのフィールドまたはプロパティであることを示すアノテーションです。

- エンティティクラス
- マップドスーパークラス
- 埋め込みクラス

適用可能要素は，メソッドとフィールドです。

@Transient を定義したフィールドの値は永続化されませんが，永続化コンテキストに保持されるため，設定した値を取得できます。ただし，別の EntityManager からは値を取得できません。

(2) 属性

@Transient の属性はありません。

8.22.58　@Version

(1)　説明

楽観的ロック機能を使用するために用いる version フィールドまたは version プロパティを指定するアノテーションです。

version フィールドまたは version プロパティでサポートする型を次に示します。

- int
- java.lang.Integer
- short
- java.lang Short
- long
- java.lang Long
- java.sql.Timestamp

このアノテーションを使用する場合は，次のことに注意してください。

- @Version と@Temporal は同時に指定できません。どちらかのアノテーションだけを指定してください。
- version プロパティをプライマリテーブル以外のテーブルには指定しないでください。
- @Version で指定されたフィールドまたはプロパティは，アプリケーションによって更新してはいけません。
- SQL を使用して複数のレコードを一度に更新するバルク更新の場合は，version フィールドまたは version プロパティを自動更新しません。このため，バルク更新をする場合に楽観的ロックを使用するときは，手動で参照および更新をする必要があります。
- エンティティクラスに対しては，version フィールドまたは version プロパティは一つだけ設定できます。複数の version フィールドまたは version プロパティを設定した場合，一つだけ有効となります。設定が有効になる順番は不定です。

適用可能要素は，メソッドとフィールドです。

(2)　属性

@Version の属性はありません。

8.22.59　アノテーションと O/R マッピングとの対応

アノテーションと O/R マッピングファイルとの対応を次の表に示します。

表 8-31　アノテーションと O/R マッピングファイルの対応

アノテーション	O/R マッピングの要素
@AssociationOverride	<association-override>
@AssociationOverrides	−
@AttributeOverride	<attribute-override>
@AttributeOverrides	−
@Basic	<basic>
@Column	<column>
@ColumnResult	<column-result>
@DiscriminatorColumn	<discriminator-column>
@DiscriminatorValue	<discriminator-value>
@Embeddable	<embeddable>
@Embedded	<embedded>
@EmbeddedId	<embedded-id>
@Entity	<entity>
@EntityListeners	<entity-listeners>
@EntityResult	<entity-result>
@Enumerated	<enumerated>
@ExcludeDefaultListeners	<exclude-default-listeners>
@ExcludeSuperclassListeners	<exclude-superclass-listeners>
@FieldResult	<field-result>
@GeneratedValue	<generated-value>
@Id	<id>
@IdClass	<id-class>
@Inheritance	<inheritance>
@JoinColumn	<join-column>
@JoinColumns	−
@JoinTable	<join-table>
@Lob	<lob>
@ManyToMany	<many-to-many>
@ManyToOne	<many-to-one>
@MapKey	<map-key>

アノテーション	O/R マッピングの要素
@MappedSuperclass	<mapped-superclass>
@NamedNativeQueries	−
@NamedNativeQuery	<named-native-query>
@NamedQueries	−
@NamedQuery	<named-query>
@OneToMany	<one-to-many>
@OneToOne	<one-to-one>
@OrderBy	<order-by>
@PostLoad	<post-load>
@PostPersist	<post-persist>
@PostRemove	<post-remove>
@PostUpdate	<post-update>
@PrePersist	<pre-persist>
@PreRemove	<pre-remove>
@PreUpdate	<pre-update>
@PrimaryKeyJoinColumn	<primary-key-join-column>
@PrimaryKeyJoinColumns	−
@QueryHint	<hint>
@SecondaryTable	<secondary-table>
@SecondaryTables	−
@SequenceGenerator	<sequence-generator>
@SqlResultSetMapping	<sql-result-set-mapping>
@SqlResultSetMappings	−
@Table	<table>
@TableGenerator	<table-generator>
@Temporal	<temporal>
@Transient	<transient>
@UniqueConstraint	<unique-constraint>
@Version	<version>

(凡例)

−：該当しません。

9 Web コンテナ

この章では，Web コンテナ（サーブレットと JSP を実行するためのサーバ基盤）について，V9 互換モードだけで使用できる機能や注意事項を説明します。Web コンテナの機能は，サーブレットまたは JSP を使用した J2EE アプリケーションを実行する場合に使用します。

9.1　リクエストおよびレスポンスのフィルタリング機能

この節では，リクエストおよびレスポンスのフィルタリング機能について説明します。

この節の構成を次の表に示します。

表 9-1　この節の構成（リクエストおよびレスポンスのフィルタリング機能）

分類	タイトル	参照先
解説	アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）	9.1.1
	フィルタ連鎖の推奨例	9.1.2
実装	DD での定義	9.1.3
設定	実行環境での設定（Web アプリケーションの設定）	9.1.4

注 「運用」および「注意事項」について，この機能固有の説明はありません。

アプリケーションサーバで使用できるフィルタリングの機能には，Servlet 仕様で規定されている機能と，アプリケーションサーバで提供する機能があります。どちらの機能も，サーブレット/JSP のリクエストやレスポンスに対してフィルタリングをする機能です。

Servlet 仕様で規定されているフィルタリング機能では，サーブレット/JSP を実行する前のリクエスト，またはサーブレット/JSP を実行したあとのレスポンスをラップします。これによって，データの変更，リソースに対するトレースの取得などができます。

また，アプリケーションサーバが提供するフィルタリング機能では，セッション情報を引き継いだり，HTTP レスポンスを圧縮したりできます。アプリケーションサーバでは，これらのフィルタリング機能を使用するためにサーブレットフィルタを提供しています。アプリケーションサーバが提供するサーブレットフィルタを **built-in フィルタ**といいます。次項でアプリケーションサーバが提供する built-in フィルタについて説明します。

9.1.1　アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）

アプリケーションサーバでは，次に示す機能を使用するためのサーブレットフィルタ（built-in フィルタ）を提供しています。

- HTTP レスポンスの圧縮

  HTTP リクエストに対する HTTP レスポンスを圧縮するために，アプリケーションサーバは built-in フィルタとして HTTP レスポンス圧縮フィルタを提供しています。

built-in フィルタの種類を次の表に示します。また，Web アプリケーションに built-in フィルタを組み込むことによって使用できる機能の参照先をあわせて示します。

表 9–2　built-in フィルタの種類

built-in フィルタの種類	機能の説明	参照先マニュアル	参照個所
HTTP レスポンス圧縮フィルタ	サーブレット，JSP，および静的コンテンツへの HTTP リクエストに対する HTTP レスポンスを gzip 形式に圧縮します。	このマニュアル	9.2

なお，Servlet 3.0 以降では，web.xml ではなく API でのフィルタ定義ができますが，built-in フィルタは API でのフィルタ定義はできません。

以降で，built-in フィルタの HTTP リクエスト・HTTP レスポンスへの作用，built-in フィルタの動作条件に関する制約について説明します。

(1)　HTTP リクエストおよび HTTP レスポンスへの作用

built-in フィルタは，クライアント側から送信される HTTP リクエストのリクエストヘッダ，およびリクエストボディに対して作用して，情報の削除，追加，変更などを実施する場合があります。また，サーバから送信される HTTP レスポンスのレスポンスヘッダ，およびレスポンスボディに対して同様に作用する場合もあります。built-in フィルタの HTTP リクエストおよび HTTP レスポンスへの作用を次の表に示します。

表 9–3　built-in フィルタの HTTP リクエストおよび HTTP レスポンスへの作用

built-in フィルタの種類	HTTP リクエストへの作用	HTTP レスポンスへの作用
HTTP レスポンス圧縮フィルタ	－	レスポンスボディが圧縮される場合，Content-Encoding ヘッダに「gzip」を指定します。また，レスポンスボディを gzip 形式で圧縮します。

（凡例）　－：該当なし

(2)　動作条件に関する制約

ユーザのフィルタと，built-in フィルタを同時に使用する場合，フィルタ連鎖の中で built-in フィルタが呼び出される順序に制約があります。

built-in フィルタごとに次に示す制約について説明します。

- 位置に関する制約
  フィルタ連鎖中の built-in フィルタの位置（呼び出される順序）に関する制約です。
- 動作条件に関する制約
  built-in フィルタが動作する上での前提条件などの動作条件に関する制約です。
- 前後に配置するほかのサーブレットフィルタに対する制約
  built-in フィルタの前後に配置するサーブレットフィルタに対する制約です。

(a) HTTP レスポンス圧縮フィルタに対する制約

HTTP レスポンス圧縮フィルタに対する制約を次の表に示します。

表 9-4 HTTP レスポンス圧縮フィルタに対する制約

制約の種類	説明
位置に関する制約	−
動作条件に関する制約	−
前後に配置するほかのサーブレットフィルタに対する制約	javax.servlet.http.HttpServletResponse クラスの setHeader メソッド，addHeader メソッド，setIntHeader メソッド，または addIntHeader メソッドを使用して，Content-Length ヘッダ，または Content-Encoding ヘッダの設定を変更するサーブレットフィルタと同時に使用する場合，そのサーブレットフィルタは HTTP レスポンス圧縮フィルタよりあとに配置する必要があります。

（凡例） −：該当なし

9.1.2 フィルタ連鎖の推奨例

フィルタ連鎖の推奨例を次に示します。次に示す順序で連鎖するように配置してください。

なお，例で説明するユーザのフィルタは，リクエストボディおよびレスポンスボディに作用する一般的なフィルタを想定しています。

- HTTP レスポンス圧縮フィルタ，およびユーザのフィルタ（FilterA）を使用する場合
  1. HTTP レスポンス圧縮フィルタ
  2. ユーザのフィルタ（FilterA）
- HTTP レスポンス圧縮フィルタ，およびユーザのフィルタ（FilterD）を使用する場合
  1. HTTP レスポンス圧縮フィルタ
  2. ユーザのフィルタ（FilterD）

9.1.3 DD での定義

リクエストおよびレスポンスのフィルタリング機能の定義は，web.xml に指定します。

DD でのリクエストおよびレスポンスのフィルタリング機能の定義については，次の場所を参照してください。

- HTTP レスポンス圧縮フィルタについては，「9.2 HTTP レスポンス圧縮機能」を参照してください。

9.1.4　実行環境での設定（Web アプリケーションの設定）

リクエストおよびレスポンスのフィルタリング機能を定義する場合，Web アプリケーションの設定が必要です。cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。リクエストおよびレスポンスのフィルタリングの定義には，フィルタ属性ファイルおよび WAR 属性ファイルを使用します。

フィルタ属性ファイルおよび WAR 属性ファイルで指定するタグは，DD と対応しています。DD（web.xml）での定義については，「9.1.3　DD での定義」を参照してください。

9.2　HTTP レスポンス圧縮機能

この節では，HTTP レスポンス圧縮機能について説明します。

HTTP レスポンス圧縮機能とは，サーブレット，JSP，および静的コンテンツへの HTTP リクエストに対する HTTP レスポンスを，gzip 形式に圧縮する機能です。この機能を使用して HTTP レスポンスを圧縮することで，Web コンテナと Web クライアント（ブラウザなど）との間の HTTP レスポンス通信に掛かる時間を削減できます。

この機能は，Web アプリケーションに組み込んで動作させるサーブレットフィルタとして提供されています。これを，**HTTP レスポンス圧縮フィルタ**といいます。

この節の構成を次の表に示します。

表 9-5　この節の構成（HTTP レスポンス圧縮機能）

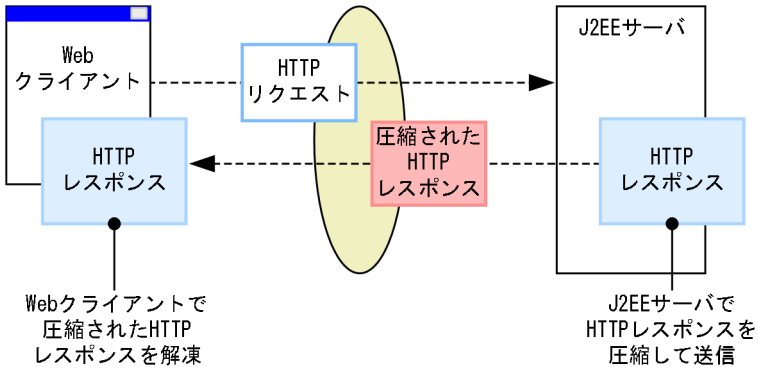
分類	タイトル	参照先
解説	HTTP レスポンス圧縮フィルタの概要	9.2.1
	HTTP レスポンス圧縮フィルタを使用するための条件	9.2.2
実装	HTTP レスポンス圧縮フィルタを使用するアプリケーションの実装	9.2.3
	DD での定義	9.2.4
	DD の定義例	9.2.5
設定	実行環境での設定（Web アプリケーションの設定）	9.2.6

注　「運用」および「注意事項」について，この機能固有の説明はありません。

9.2.1　HTTP レスポンス圧縮フィルタの概要

HTTP レスポンス圧縮機能を有効にすると，HTTP レスポンスのレスポンスボディを gzip 形式に圧縮します。HTTP レスポンス圧縮機能の概要を次の図に示します。

図 9-1　HTTP レスポンス圧縮機能の概要



HTTP レスポンス圧縮機能を有効にするためには，アプリケーションサーバが提供する HTTP レスポンス圧縮フィルタを Web アプリケーションに組み込む必要があります。HTTP レスポンス圧縮機能を適用する場合，Web アプリケーションの DD（web.xml）に HTTP レスポンス圧縮フィルタのフィルタ定義，およびフィルタマッピングの定義を追加します。J2EE サーバにデプロイ済みの Web アプリケーションに適用する場合は，サーバ管理コマンドを使用して，HTTP レスポンス圧縮フィルタのフィルタ定義，およびフィルタマッピングの定義を追加します。

9.2.2 HTTP レスポンス圧縮フィルタを使用するための条件

ここでは，HTTP レスポンス圧縮フィルタを使用する場合の条件や注意事項について説明します。

(1) 前提条件

HTTP レスポンス圧縮機能を使用にするには，次の前提条件を満たしている必要があります。

- Web クライアントの gzip 形式の対応

  HTTP レスポンス圧縮機能を有効にした場合，gzip 形式で圧縮された HTTP レスポンスを Web クライアントで解凍する必要があります。そのため，Web クライアントが gzip 形式に対応していることが前提となります。Web クライアントが gzip 形式に対応していない場合は，HTTP レスポンス圧縮機能を有効にしていても，HTTP レスポンスは圧縮されません。

- Web クライアントの HTTP/1.1 の対応

  HTTP レスポンス圧縮機能では，Web クライアントが gzip 形式に対応しているかどうかを HTTP リクエストの Accept-Encoding ヘッダに指定された値によって判定しています。そのため，Web クライアントは Accept-Encoding ヘッダの仕様が規定されている HTTP/1.1 に対応している必要があります。

(2) 必要なメモリ量

HTTP レスポンス圧縮機能に必要なメモリ量は次の計算式で求められます。

HTTP レスポンス圧縮機能に必要なメモリ量（バイト）=HTTP レスポンス圧縮機能が有効となる HTTP リクエストの同時実行数×レスポンスの圧縮しきい値（バイト）

圧縮しきい値とは，HTTP レスポンスボディのサイズによって，HTTP レスポンスを圧縮するかどうかを判定するためのしきい値です。HTTP レスポンスボディのサイズが圧縮しきい値に定義したサイズを超える場合にだけ，HTTP レスポンスを圧縮します。なお，圧縮しきい値は，HTTP リクエストに対して設定します。

圧縮しきい値は，DD（web.xml）に定義します。圧縮しきい値を定義することで，HTTP レスポンスのサイズが小さい場合に，HTTP レスポンスの圧縮に掛かる時間が，通信に掛かる時間よりも大きくならないようにします。

圧縮しきい値は，圧縮するリソースの種別と通信回線の速度によって適切なサイズが決まります。圧縮しきい値に定義するサイズは実測で求め，適切なサイズを定義することを推奨します。

(3) HTTP レスポンス圧縮機能を有効にする場合の条件

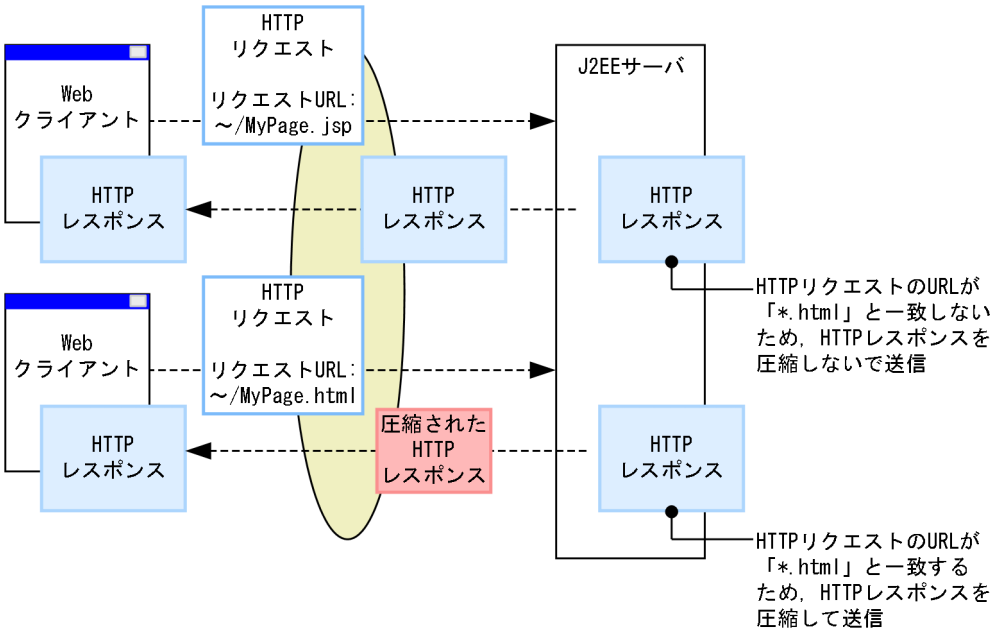
HTTP レスポンス圧縮機能が有効になる条件を指定できます。指定できる条件を次に示します。

- HTTP リクエストの URL パターン

  HTTP レスポンス圧縮フィルタを実装している Web アプリケーションに対するリクエストの URL が，指定した URL パターンと一致する場合，そのリクエストに対するレスポンスを圧縮します。

  HTTP レスポンスの圧縮を実行する HTTP リクエストの URL パターンとして「*.html」を指定した場合の例を次の図に示します。

  図 9-2 HTTP レスポンスの圧縮を実行する HTTP リクエストの URL パターンとして「*.html」を指定した場合の例



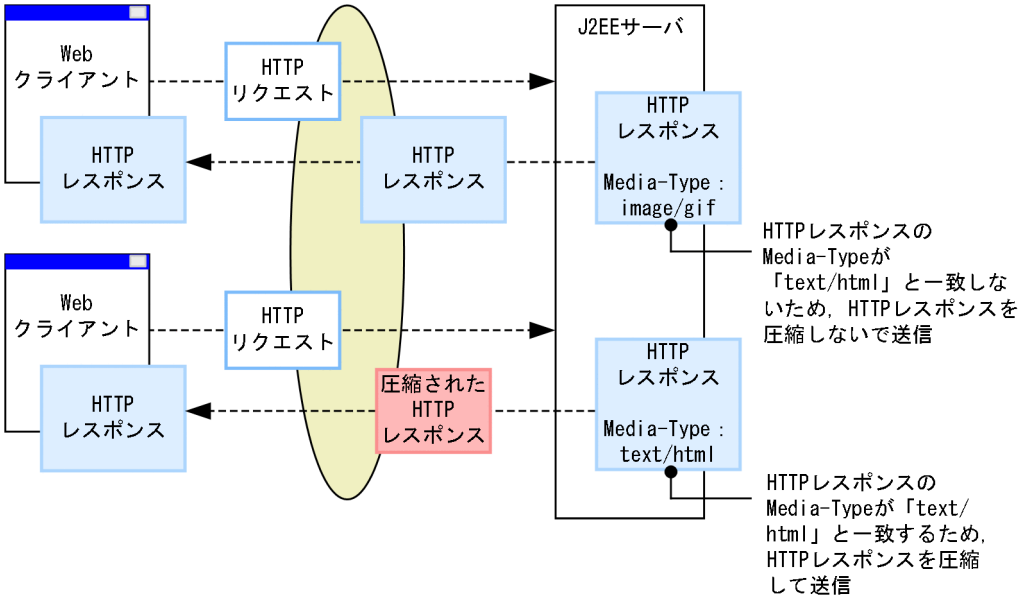
- HTTP レスポンスの Media-Type

  HTTP レスポンスの Content-Type ヘッダに含まれる Media-Type の値が，指定した値と一致する場合，HTTP レスポンスを圧縮します。

  HTTP レスポンスの Media-Type は，サーブレット/JSP の場合は J2EE アプリケーションが setContentType メソッドによって設定した値です。静的コンテンツの場合は拡張子に対応づけられた MIME タイプとなります。

  HTTP レスポンスの圧縮を実行する HTTP レスポンスの Media-Type として「text/html」を指定した場合の例を次の図に示します。

図 9-3　HTTP レスポンスの圧縮を実行する HTTP レスポンスの Media-Type として「text/html」を指定した場合の例

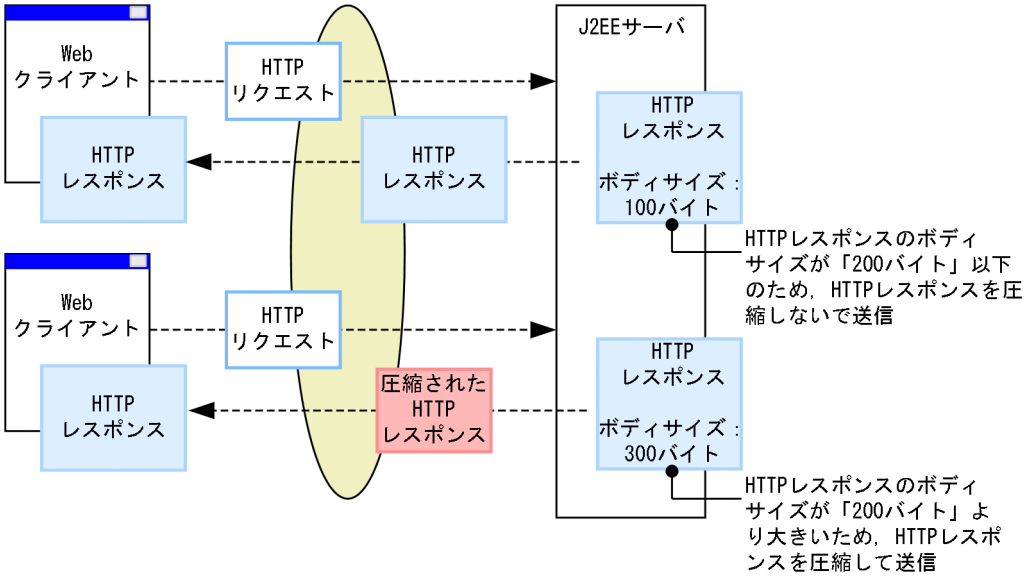


- HTTP レスポンスのボディサイズ

  HTTP レスポンスの圧縮を実行するためしきい値を設定して，ボディサイズがこのしきい値を超える場合，HTTP レスポンスを圧縮します。

  HTTP レスポンスの圧縮を実行する HTTP レスポンスのボディサイズとして「200 バイト」を指定して HTTP レスポンス圧縮機能を有効にした場合の例を次の図に示します。

図 9-4　HTTP レスポンスの圧縮を実行する HTTP レスポンスのボディサイズとして「200 バイト」を指定した場合の例



(4) 注意事項

HTTP レスポンス圧縮フィルタの定義に関する注意事項

HTTP レスポンス圧縮フィルタを使用する場合，built-in フィルタの HTTP リクエストおよび HTTP レスポンスへの作用やフィルタ連鎖の順序制約を考慮して，Web アプリケーションに HTTP レスポンス圧縮フィルタを組み込む必要があります。built-in フィルタの詳細については，「9.1.1 アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）」を参照してください。

なお，Servlet 3.0 以降では，web.xml ではなく API でのフィルタ定義ができますが，built-in フィルタは API でのフィルタ定義はできません。

エラーページに関する注意事項

HTTP レスポンス圧縮機能を使用する Web アプリケーションでは，次の機能を使用してエラーページをカスタマイズできます。

- Web サーバの機能を使用したエラーページのカスタマイズ
- インプロセス HTTP サーバによるエラーページのカスタマイズ
- web.xml の<error-page>タグを使用したエラーページのカスタマイズ

web.xml の<error-page>タグを使用したエラーページを使用する場合は，エラーページに静的コンテンツ，またはレスポンスから javax.servlet.ServletOutputStream を取得して使用するサーブレットを指定してください。

HTTP レスポンス圧縮機能では圧縮後のデータの出力にレスポンスオブジェクトから取得した javax.servlet.ServletOutputStream を使用します。そのため，エラーページを生成するサーブレット，または JSP でレスポンスオブジェクトからは，java.io.PrintWriter を取得できません。

9.2.3 HTTP レスポンス圧縮フィルタを使用するアプリケーションの実装

ここでは，HTTP レスポンス圧縮フィルタを使用するアプリケーションを開発するときの注意事項について説明します。

(1) ほかのフィルタと併用する場合の呼び出し順序

HTTP レスポンス圧縮フィルタは，HTTP レスポンスのヘッダに設定するほかのフィルタよりも前に呼び出される必要があります。javax.servlet.http.HttpServletResponse の setHeader メソッド，addHeader メソッド，setIntHeader メソッド，および addIntHeader メソッドを使用して，Content-Length ヘッダ，Content-Encoding ヘッダを設定するほかのフィルタを使用する場合は，HTTP レスポンス圧縮フィルタよりも後ろに配置してください。

(2) HTTP レスポンスのバッファに関する注意事項

HTTP レスポンス圧縮機能を有効にした場合，HTTP レスポンスのバッファよりも前に圧縮しきい値に指定したサイズのバッファが設けられます。HTTP レスポンスのバッファには，出力したデータが圧縮しきい値を超えたときに書き込まれます。

圧縮後のデータサイズが HTTP レスポンスのバッファサイズを超えなければ，Web クライアントに HTTP レスポンスが書き出されることはありません。出力したデータのサイズが圧縮しきい値を超える前に，HTTP レスポンスを Web クライアントに書き出す必要がある場合は，明示的にレスポンス出力用ストリーム※の flush メソッドを呼び出す必要があります。ただし，出力したデータのサイズが圧縮しきい値を超える前に flush メソッド，または javax.servlet.ServletResponse クラスの flushBuffer メソッドを呼び出した場合は，出力したデータは圧縮されないで Web クライアントに書き出されます。

注※

レスポンス出力用ストリームとは，次に示すオブジェクトを指します。

- javax.servlet. ServletResponse クラスの getOutputStream メソッドによって取得した javax.servlet.ServletOutputStream
- javax.servlet. ServletResponse クラスの getWriter メソッドによって取得した java.io.PrintWriter
- JSP で暗黙的に利用できる javax.servlet.jsp.JspWriter

(3) HTTP レスポンスのレスポンスヘッダに関する注意事項

HTTP レスポンス圧縮機能によって HTTP レスポンスのレスポンスボディが圧縮される場合，この HTTP レスポンスの Content-Encoding ヘッダには gzip，Vary ヘッダには Accept-Encoding が指定されます。Content-Length ヘッダには何も指定されません。

したがって，javax.servlet.ServletResponse クラスの setContentLength メソッドを使用する場合と javax.servlet.http.HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※を使用する場合は，次の点に注意してください。

- 次のどちらかの API を使用して Content-Length ヘッダ，Content-Encoding ヘッダを設定するフィルタを追加している場合は，レスポンス圧縮用フィルタよりもあとに実行されるように DD（web.xml）で定義してください。
  - ServletResponse クラスの setContentLength メソッド
  - HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※
- HTTP レスポンスのレスポンスボディが圧縮される場合，ServletResponse クラスの setContentLength メソッド，および HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※を使用しても，HTTP レスポンスの Content-Length ヘッダは付加されません。Content-Length ヘッダが付加されていない HTTP レスポンスは Web コンテナによって chunk 形式でクライアントに送信されます。
- HTTP レスポンスのレスポンスボディが圧縮される場合，HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※を使用しても Content-Encoding ヘッダには値が設定されません。レスポンスボディが圧縮される場合は，Web コンテナによって Content-Encoding ヘッダに gzip が指定されます。

注※

レスポンスヘッダを追加・変更する API とは，javax.servlet.http.HttpServletResponse クラスの次のメソッドを指します。

- setHeader メソッド
- addHeader メソッド
- setIntHeader メソッド
- addIntHeader メソッド

(4) HTTP レスポンスのデータ出力に関する注意事項

javax.servlet.ServletResponse クラスの getOutputStream メソッド，または getWriter メソッドによって ServletOutputStream または PrintWriter を取得し，HTTP レスポンスを出力する場合は，次の点に注意してください。

- ServletOutputStream または PrintWriter を使用して，HTTP レスポンスのバッファにデータを書き出している状態で，ServletResponse の setContentType メソッドを呼び出した場合，圧縮するように指定された Media-Type を持つ HTTP レスポンスであっても圧縮されません。

  ただし，圧縮する Media-Type にアスタリスク（*）が指定されているときは圧縮されます。
- Media-Type を指定して JSP の出力を圧縮するためには，page ディレクティブの contentType 属性を指定するか，または JSP のバッファを超える前に ServletResponse クラスの setContentType メソッドを呼び出す必要があります。

(5) アプリケーションで HTTP レスポンスを圧縮する場合の注意事項

アプリケーションで圧縮した HTTP レスポンスには，HTTP レスポンス圧縮機能が有効にならないように設定してください。アプリケーションで圧縮した HTTP レスポンスに対して，HTTP レスポンス圧縮機能を有効にした場合の動作は保証できません。

9.2.4 DD での定義

ここでは，HTTP レスポンス圧縮機能を使用する場合に必要な DD の定義について説明します。

HTTP レスポンス圧縮機能を有効にするためには，Web アプリケーションの DD に，フィルタ定義およびフィルタマッピング定義を追加する必要があります。HTTP レスポンス圧縮機能は，フィルタマッピング定義によってマッピングされた URL パターンを持つリソースに対してリクエストがあった場合にだけ有効となります。

HTTP レスポンス圧縮機能の定義は，web.xml に指定します。

DD での HTTP レスポンス圧縮機能の定義について次の表に示します。

表 9-6　DD での HTTP レスポンス圧縮機能の定義

設定項目	指定するタグ	設定内容
フィルタ定義	<filter>タグ内の<filter-name>タグ	追加するフィルタ名称を指定します。設定値は，固定値です。 **設定値（固定値）** com.hitachi.software.was.web.ResponseCompressionFilter
	<filter>タグ内の<filter-class>タグ	追加するフィルタのクラス名を指定します。設定値は，固定値です。 **設定値（固定値）** com.hitachi.software.was.web.ResponseCompressionFilter
URL パターンと HTTP レスポンス圧縮規則のマッピング	<filter-class><init-param>タグ内の<param-name>タグ，<param-value>タグ	URL パターンと HTTP レスポンス圧縮機能のマッピングを指定します。 詳細については，「9.2.4(1)　URL パターンと HTTP レスポンス圧縮規則のマッピング（url-mapping）」を参照してください。
HTTP レスポンス圧縮規則		HTTP レスポンスの圧縮規則として，圧縮規則名，Media-Type と圧縮しきい値を指定します。 詳細については，「9.2.4(2)　HTTP レスポンス圧縮規則」を参照してください。
フィルタマッピング定義	<filter-mapping>タグ内の<filter-name>タグ	フィルタ名称を指定します。設定値は，固定値です。 **設定値（固定値）** com.hitachi.software.was.web.ResponseCompressionFilter
	<filter-mapping>タグ内の<url-pattern>タグ	URL パターンやサーブレットクラスと，HTTP レスポンス圧縮フィルタのマッピングを指定します。HTTP レスポンス圧縮機能は，フィルタマッピング定義によって，マッピングされた URL パターンを持つリソースに対してリクエストがあった場合だけ有効となります。 設定値は，任意です。

次に示す DD の定義例を基に，DD の定義内容を説明します。

```
      :
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /*=rule1;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       *:1000;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
```

```
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

<filter>タグで囲まれた部分がフィルタ定義，<filter-mapping>タグで囲まれた部分がフィルタマッピング定義となります。フィルタ定義の<filter-name>タグおよび<filter-class>タグに指定する値は，次のパッケージ名で固定です。

```
com.hitachi.software.was.web.ResponseCompressionFilter
```

次に，DD の<init-param>タグ内に定義する内容を示します。

(1) URL パターンと HTTP レスポンス圧縮規則のマッピング（url-mapping）

HTTP レスポンス圧縮機能が有効となる URL パターンと，指定した URL パターンに適用する HTTP レスポンス圧縮規則名のマッピングを指定します。

パラメタ指定時の規則および URL パターンのマッピング規則を示します。

(a) パラメタ指定時の規則

- URL パターンは，大文字と小文字が区別されます。
- HTTP レスポンス圧縮規則名は，大文字と小文字が区別されます。
- URL パターンと HTTP レスポンス圧縮規則名は，半角のイコール（=）で区切ります。
- 複数指定する場合は，半角のセミコロン（;）で区切ります。
- 複数の URL パターンに該当する場合は，先に指定したものが使用されます。
- パラメタ値の改行，タブ，スペースは無視されます。
- パラメタ値の末尾にある半角のセミコロン（;）は無視されます。

(b) URL パターンのマッピング規則

url-mapping パラメタに定義する URL パターンには，パス一致，拡張子一致，および完全一致のマッピング規則が適用されます。それぞれのマッピング規則を次に示します。

- **パス一致**

  URL パターンとして「/」で始まり「/*」で終わる文字列を指定した場合は，リクエスト URL のコンテキストルートからの相対パスが URL パターンの「*」を除く文字列で始まっているときに，一致したとみなされます。また，URL パターンとして「/*」を指定した場合は，すべてのリクエスト URL が一致したとみなされます。

  （例）

  URL パターンが「/jsp/*」では，リクエスト URL のコンテキストルートからの相対パスが「/jsp/index.jsp」の場合に一致したとみなされます。

- **拡張子一致**

URL パターンとして「*.」で始まる文字列を指定した場合は，リクエスト URL の拡張子と URL パターンの「*.」に続く文字列と同じであるときに，一致したとみなされます。

（例）

URL パターンが「*.jsp」では，リクエスト URL のコンテキストルートからの相対パスが「/jsp/index.jsp」の場合に一致したとみなされます。

- 完全一致

URL パターンとして「/」で始まる上記以外の文字列を指定した場合は，リクエスト URL のコンテキストルートからの相対パスがこの URL パターンと完全に同じであるときに，一致したとみなされます。

（例）

URL パターンが「/jsp/index.jsp」では，リクエスト URL のコンテキストルートからの相対パスが「/ jsp/index.jsp」の場合に一致したとみなされます。

(2) HTTP レスポンス圧縮規則

圧縮する HTTP レスポンスの Media-Type と圧縮しきい値を指定します。

Media-Type

HTTP レスポンス圧縮機能では，条件として指定した Media-Type が HTTP レスポンスに含まれる場合に，HTTP レスポンスを圧縮します。

圧縮しきい値

圧縮しきい値は，100 から 2,147,483,647 までの整数値で指定します。デフォルトの設定では，すべての Media-Type に対して圧縮しきい値「100」が適用されます。

圧縮しきい値とは，HTTP レスポンスボディのサイズによって，HTTP レスポンスを圧縮するかどうかを判定するためのしきい値です。HTTP レスポンス圧縮機能では，HTTP レスポンスボディのサイズが圧縮しきい値に定義したサイズを超える場合に，HTTP レスポンスを圧縮します。

圧縮しきい値を定義することで，HTTP レスポンスのサイズが小さい場合に，HTTP レスポンスの圧縮に掛かる時間が，通信に掛かる時間よりも大きくならないようにします。

圧縮しきい値は，圧縮するリソースの種別と通信回線の速度によって適切なサイズが決まるので，圧縮しきい値に定義するサイズは実測で求め，適切なサイズを定義することをお勧めします。

パラメタ指定時の規則を示します。

- Media-Type にアスタリスク（*）を指定した場合は，すべての Media-Type を表します。ただし，Media-Type ごとの指定がある場合は，Media-Type ごとの指定が優先されます。
- Media-Type は，大文字と小文字が区別されません。
- Media-Type と圧縮しきい値は，半角のコロン（:）で区切ります。
- 複数指定する場合は半角のセミコロン（;）で区切ります。
- 同じ Media-Type を複数指定した場合はあとに指定したものが使用されます。
- パラメタ値の改行，タブ，スペースは無視されます。

- パラメタ値の末尾にある半角のセミコロン（;）は無視されます。

(3) 注意事項

HTTP レスポンス圧縮フィルタの設定時の注意事項を次に示します。

- レスポンス圧縮用フィルタの初期化時に，フィルタの初期化パラメタの妥当性をチェックします。ここで初期化パラメタに定義された値に問題がある場合は，フィルタの初期化処理でエラーとなり，Web アプリケーションは開始しません。
- リクエスト URL が url-mapping パラメタに指定した URL パターンに一致しても，レスポンス圧縮用フィルタがマッピングされている URL パターンに一致しない場合は，url-mapping パラメタに指定したレスポンス圧縮規則が適用されることはありません。
- レスポンス圧縮フィルタに複数の URL パターンをマッピングする場合は，リクエスト URL が同時に複数の URL パターンに一致しないようにフィルタマッピングの定義を記述する必要があります。複数の URL パターンに対して異なるレスポンス圧縮機能を設定する場合は，url-mapping パラメタに複数の URL パターンを指定してください。
- Web アプリケーションのバージョンが Servlet 2.4 以降の場合，<filter-mapping>の<dispatcher>タグは指定しないでください。また，<dispatcher>タグで「FORWARD」，「INCLUDE」，「ERROR」を指定した場合，Web アプリケーションの開始時にエラーが発生し，アプリケーションが開始できないので，注意してください。

9.2.5 DD の定義例

ここでは，次に示すそれぞれの場合を例に，HTTP レスポンス圧縮機能を使用する DD の定義例を示します。

- HTTP レスポンスのボディサイズで圧縮条件を指定する場合
- URL パターンで圧縮条件を指定する場合
- HTTP レスポンスの Media-Type で圧縮条件を指定する場合
- 圧縮条件を組み合わせて指定する場合

(1) HTTP レスポンスのボディサイズで圧縮条件を指定する場合

HTTP レスポンスのボディサイズで圧縮条件を指定する場合の定義例を示します。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
```

```
      <param-value>
        /*=rule1;
      </param-value>
    </init-param>
    <init-param>
      <param-name>rule1</param-name>
      <param-value>
        *:1000;
      </param-value>
    </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
    <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例では，URL パターンが/*のアクセスに対する HTTP レスポンスで，ボディサイズが 1,000 バイトを超えるものを圧縮します。

(2) URL パターンで圧縮条件を指定する場合

URL パターンで圧縮条件を指定する場合の定義例を示します。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
    <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
    <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
    <init-param>
      <param-name>url-mapping</param-name>
      <param-value>
        /app/dir/*=rule1;
        *.html=rule1;
      </param-value>
    </init-param>
    <init-param>
      <param-name>rule1</param-name>
      <param-value>
        *:100;
      </param-value>
    </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
    <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
    <url-pattern>/app/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例では，URL パターンが/app/*のアクセスに対する HTTP レスポンスで次の条件を満たすものを圧縮します。

- HTTP リクエストの URL パターンが/app/dir/*で，ボディサイズが 100 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが*.html で，ボディサイズが 100 バイトを超える HTTP レスポンス

(3) HTTP レスポンスの Media-Type で圧縮条件を指定する場合

HTTP レスポンスの Media-Type で圧縮条件を指定する場合の定義例を示します。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /*=rule1;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       text/html:500;
       application/pdf:1000;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例は URL パターンが/*のアクセスに対する HTTP レスポンスで次の条件を満たすものを圧縮します。

- Media-Type が text/html でボディサイズが 500 バイトを超える HTTP レスポンス
- Media-Type が application/pdf でボディサイズが 1,000 バイトを超える HTTP レスポンス

(4) 圧縮条件を組み合わせて指定する場合

次に示す例のように，圧縮条件を組み合わせて定義することもできます。

```
<web-app>
      :
```

```
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /app/dir1/*=rule1;
       /app/dir2/*=rule2;
       *.html=rule3;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       *:500;
       application/pdf:1000;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule2</param-name>
     <param-value>
       application/pdf:2000;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule3</param-name>
     <param-value>
       *:100;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/app/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例では，URL パターンが/app/*のアクセスに対する HTTP レスポンスで次の条件を満たすものを圧縮します。

- HTTP リクエストの URL パターンが/app/dir1/*で Media-Type が application/pdf 以外，ボディサイズが 500 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが/app/dir1/*で Media-Type が application/pdf，ボディサイズが 1,000 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが/app/dir2/*で Media-Type が application/pdf，ボディサイズが 2,000 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが*.html でボディサイズが 100 バイトを超える HTTP レスポンス

9.2.6　実行環境での設定（Web アプリケーションの設定）

フィルタリングによる HTTP レスポンスの圧縮を使用する場合，Web アプリケーションの設定が必要です。

なお，Web アプリケーションの設定は，cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。フィルタリングによる HTTP レスポンスの圧縮の定義には，フィルタ属性ファイルおよび WAR 属性ファイルを使用します。

フィルタ属性ファイルおよび WAR 属性ファイルで指定するタグは，DD と対応しています。DD（web.xml）での定義については，「9.2.4　DD での定義」および「9.2.5　DD の定義例」を参照してください。

9.3 Web コンテナに関する注意事項

V9 互換モードの Web コンテナで取り扱える POST データの最大サイズは 2GB です。また，Web コンテナの設定や，Web コンテナのフロントにある Web サーバやリダイレクタの設定によって，取り扱える POST データの最大サイズは制限されます。

9.4　サーブレットおよび JSP の実装時の注意事項

この節では，サーブレットおよび JSP の実装時の注意事項について説明します。

この節の構成を次の表に示します。

表 9-7　この節の構成（サーブレットおよび JSP の実装時の注意事項）

タイトル	参照先
サーブレットおよび JSP 実装時共通の注意事項	9.4.1
サーブレット実装時の注意事項	9.4.2
EL2.2 仕様で追加，変更された仕様についての注意事項	9.4.3

9.4.1　サーブレットおよび JSP 実装時共通の注意事項

アプリケーションサーバ上で動作するアプリケーションのプログラムとして，サーブレットおよび JSP を実装するときの共通の注意事項を示します。

(1)　Web アプリケーションの動作の前提となる J2EE アプリケーションのバージョン

Web アプリケーションの動作の前提となる J2EE アプリケーションが準拠する J2EE 仕様のバージョンについて，Web アプリケーションのバージョンごとに次の表に示します。

表 9-8　J2EE アプリケーションが準拠する J2EE 仕様のバージョン

J2EE アプリケーションが準拠する J2EE 仕様のバージョン	Web アプリケーションに対応する Servlet 仕様				
	3.0	2.5	2.4	2.3	2.2
Java EE 6	○	○	○	○	○
Java EE 5	×	○	○	○	○
J2EE1.4	×	×	○	○	○
J2EE1.3	×	×	△	○	○
J2EE1.2	×	×	△	△	○

（凡例）

○：使用できる

△：使用できる（J2EE アプリケーションのインポート時に J2EE 仕様のバージョンが 1.4 に更新されるため）

×：使用できない

(2) Web アプリケーションのサポート範囲

Web アプリケーションのバージョンは，web.xml に記述する Servlet 仕様のバージョン情報で識別されます。上位のバージョンの Web アプリケーションは下位のバージョンの機能を使用できます。下位のバージョンの Web アプリケーションは上位のバージョンの機能を使用できません。

Web アプリケーションのバージョンごとに，使用できる機能範囲を次の表に示します。

表 9-9 Web アプリケーションのサポート範囲

Web アプリケーションのバージョン	Servlet					JSP					タグライブラリ※1			
	3.0	2.5	2.4	2.3	2.2	2.2	2.1	2.0	1.2	1.1	2.1	2.0	1.2	1.1
3.0	○	○	○	○	○	×※2	○	○	○	○	○	○	○	○
2.5	×	○	○	○	○	×	○	○	○	○	○	○	○	○
2.4	×	×	○	○	○	×	×	○	○	○	×	○	○	○
2.3	×	×	×	○	○	×	×	×	○	○	×	×	○	○
2.2	×	×	×	×	○	×	×	×	×	○	×	×	×	○

（凡例）○：使用できる　×：使用できない

注※1

タグライブラリのバージョンとは，タブライブラリ・ディスクリプタ（TLD ファイル）のバージョンを表します。

注※2

JSP 2.2 に対応した web.xml は読み込めますが，JSP 2.2 で追加されたタグについては無視されます。

なお，下位のバージョンの Web アプリケーションから上位のバージョンの機能を使用した場合，エラーが発生することがあります。発生するエラーについてバージョンごとに次に示します。

表 9-10 Servlet 2.2，2.3，2.4 または 2.5 仕様に対応する Web アプリケーションから Servlet 3.0 の機能を使用した場合のエラー

仕様	使用する機能	エラー時の処理
Servlet 3.0	新規 API の呼び出し	Servlet 3.0 仕様で追加された API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。
	新規アノテーションの使用	アノテーションを使用してエラーとなった場合の処理については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「17. アノテーションの使用」を参照してください。
JSP 2.2	EL の新規 API	EL2.2 の API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。

Servlet 2.2 仕様，Servlet 2.3 仕様，Servlet 2.4 仕様および Servlet 2.5 仕様から，Servlet 3.0 仕様に Web アプリケーションのバージョンアップする場合の作業，および注意事項については，マニュアル「ア

プリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「8.2.12　既存の Web アプリケーションを Servlet 3.0 仕様にバージョンアップする場合の留意点」を参照してください。

表 9–11　Servlet 2.2，2.3，または 2.4 仕様に対応する Web アプリケーションから Servlet 2.5 の機能を使用した場合のエラー

仕様	使用する機能	エラー時の処理
Servlet 2.5	新規 API の呼び出し	Servlet 2.5 仕様で追加された API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。
	新規アノテーションの使用	アノテーションを使用してエラーとなった場合の処理については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「17.　アノテーションの使用」を参照してください。
JSP 2.1	新規ディレクティブの属性※1	サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	TLD 2.1	Web アプリケーション開始時に次に示す TLD ファイルが存在した場合，メッセージログに KDJE39293-W のメッセージが出力され，処理されません。 • web.xml の<taglib>要素内の<tablib-location>要素に指定された TLD ファイル • /WEB-INF/lib ディレクトリ以下の Jar ファイル内の/META-INF ディレクトリ以下に配置された TLD ファイル Web アプリケーション開始時にこれら以外の TLD ファイルが存在した場合は，JSP コンパイル時にメッセージログに KDJE39293-W のメッセージが出力され，処理されません。 アプリケーションへの初回アクセス時などに JSP コンパイルが発生した場合は，サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	EL の追加機能	• JSP 2.1 で追加された Enum 型については，専用の処理は実施されないで，一般のクラスと同様に処理されます。 ただし，JSP 2.1 仕様で非推奨となった JSP 2.0 仕様の EL の API を使用した場合，Web アプリケーションのバージョンに関係なく，JSP 2.0 仕様の EL の機能範囲で処理されます。 • #{}の書式の EL は文字列として表示されます。 • "¥#"はエスケープシーケンスとして扱われません。「"¥#"」という文字列として表示されます。

注※1

JSP ページで<jsp:directive.XXX />形式で XXX に JSP 仕様で未定義のディレクティブを指定した場合，または<jsp:XXX>形式で XXX に JSP 仕様で未定義のスタンダードアクションを指定した場合，定義内容はそのまま出力されます。

注※2

KDJE39145-E には JSP のコンパイルエラーの詳細，KDJE39186-E にはトランスレーションエラーの発生通知が出力されます。

Servlet 2.2 仕様，Servlet 2.3 仕様および Servlet 2.4 仕様から，Servlet 2.5 仕様に Web アプリケーションのバージョンアップする場合の作業，および注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「8.2.13　既存の Web アプリケーションを Servlet 2.5 仕様にバージョンアップする場合の留意点」を参照してください。

表 9–12　Servlet 2.2 または 2.3 仕様に対応する Web アプリケーションから Servlet 2.4 仕様の機能を使用した場合のエラー

仕様	使用する機能	エラー時の処理
Servlet 2.4	新規 API の呼び出し	Servlet 2.4 仕様で追加された API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。
	新規リスナ登録	Web アプリケーションの開始時に KDJE39297-W のメッセージがメッセージログに出力され，そのリスナ定義は無視されます。
JSP 2.0	新規ディレクティブ新規スタンダードアクション※1	サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	タグファイル	TLD を使用しない場合 taglib ディレクティブで新規属性となる tagdir 属性が不正として，サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。 TLD を使用する場合 TLD 2.0 を使用したエラーになります。
	TLD 2.0	次に示す TLD ファイルは，Web アプリケーション開始時にチェックされます。該当する場合，メッセージログに KDJE39293-W のメッセージが出力され，無視されます。 • web.xml の<taglib><tablib-location>に指定された TLD ファイル • /WEB-INF/lib 下の Jar ファイル内の/META-INF 以下に配置された TLD ファイル これら以外の TLD ファイルは，JSP コンパイル時にチェックされます。初回アクセス時など，JSP ファイルのコンパイル時は，サーブレットログに KDJE 39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	シンプル・タグ・ハンドラ	サーブレットログに KDJE39145-E のメッセージを，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。

注※1

JSP ページで<jsp:directive.XXX />形式で XXX に JSP 仕様で未定義のディレクティブを指定した場合，または<jsp:XXX>形式で XXX に JSP 仕様で未定義のスタンダードアクションを指定した場合，定義内容はそのまま出力されます。

注※2

KDJE39145-E には JSP のコンパイルエラーの詳細，KDJE39186-E にはトランスレーションエラーの発生通知が出力されます。

Servlet 2.2 仕様および Servlet 2.3 仕様から，Servlet 2.4 仕様に Web アプリケーションのバージョンアップする場合の作業，および注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「8.2.15　既存の Web アプリケーションを Servlet 2.4 仕様にバージョンアップする場合の留意点」を参照してください。

なお，Servlet 2.2 仕様に対応する Web アプリケーションから Servlet 2.3 の機能を使用しても，アプリケーションのインポート時に Servlet 2.3 仕様に準拠した Web アプリケーションに書き換えられるため，正常に処理され，エラーは通知されません。

(3)　トランザクションと JDBC コネクション利用時の注意

サーブレット，JSP でトランザクションを利用する場合，該当するサービスメソッドで JDBC コネクションを取得し，該当するサービスメソッドが終了する前に解放してください。トランザクションが開始しているサーブレットおよび JSP では，次に示す JDBC コネクションの使用はサポートされません。

- サーブレット，JSP のサービスメソッドが生成したスレッド上の JDBC コネクションを使用する。
- サーブレット，JSP のサービスメソッドから呼び出した別のサーブレット，JSP のサービスメソッドで JDBC コネクションを使用する。
- サーブレット，JSP のサービスメソッドの init メソッドで取得した JDBC コネクションを使用する。
- インスタンス変数に格納された JDBC コネクションを使用する。※

  注※

  SingleThreadModel のサーブレットおよび JSP を使用した場合は，インスタンス変数に JDBC コネクションを格納できます。

(4)　パッケージ名の指定に関する注意

不正なパッケージ名が指定されたクラスをサーブレットおよび JSP で使用した場合，ブラウザからアクセスしたときにステータスコード 500 のエラーになります。例えば，作成したクラスファイルを正しく配置して，ブラウザからアクセスしても，パッケージ名の宣言に不正があった場合は，該当クラスが見つかりません。この場合，ステータスコード 500 のエラーが返されます。

(5)　Cookie 利用時の注意

- 日本語などの 2 バイトコードを含む Cookie は使用しないでください。使用した場合，サーブレットおよび JSP で利用している HTTP セッションが失われる場合があります。
- Cookie でセッション管理をする場合，ホスト名による URL でアクセスしたサーブレットまたは JSP で生成されたセッションは，ホスト名の代わりに IP アドレスを指定した URL でアクセスしたサーブレットまたは JSP に引き継がれません（逆も同様です）。

(6) 特別な意味を持つ入力値の表示に関する注意

フォームなどで「<」や「>」などの特別な意味を持つ文字の入力値をそのまま表示した場合，悪意のあるユーザが<SCRIPT>，<OBJECT>，<APPLET>，<EMBED>のスクリプトなどを実行できるタグを使用して，重大なセキュリティ上の問題を引き起こすおそれがあります。アプリケーション開発者は，ユーザから入力されたデータに対して必ず検査をする処理を追加して，特別な意味を持つ文字を排除する必要があります。

(7) コミット後のエラーページの表示に関する注意

サーブレットまたは JSP でレスポンスがコミットされたあとは，例外などのエラーが発生したとしても，次に示すエラーページはブラウザに表示されません。

- web.xml で指定したエラーページ
- JSP の page ディレクティブの errorPage 属性で指定したエラーページ

レスポンスのコミットは，ユーザが ServletResponse クラスの flushBuffer メソッドなどを明示的に呼び出してコミットする場合以外にも，レスポンスのバッファが満杯になって自動的に Web コンテナがコミットすることがあります。

サーブレットまたは JSP でコミットされているかどうかを調べるには，ServletResponse クラスの isCommitted メソッドを使用します。また，バッファサイズの変更は，サーブレットの場合は ServletResponse クラスの setBufferSize メソッドで，JSP の場合は page ディレクティブの buffer 属性の指定で実施できます。

(8) PrintWriter，JSPWriter クラス利用時の性能向上について

PrintWriter クラスおよび JSPWriter クラスの print メソッドと println メソッドの呼び出し回数を少なくすることで，アクセス回数を減らし，性能を向上できます。例えば，StringBuffer クラスを使用し，最後に println メソッドを呼び出すようにして，print および println メソッドの呼び出し回数を削減します。

(9) javax.servlet.error.XXXXX によるエラー情報参照時の注意

Servlet 2.3 仕様で定義されている javax.servlet.error.XXXXX 属性は，web.xml の<error-page>タグに指定されたサーブレットまたは JSP 内でそのエラーページを実行する要因となったエラー情報を参照するためのものです。web.xml の<error-page>タグに指定されたサーブレットまたは JSP 以外からは，これらの属性を参照しないでください。

(10) ファイルアクセス時の注意

ファイルにアクセスする場合は，必ず絶対パスを指定してください。相対パスを指定すると，J2EE サーバは実行ディレクトリからの相対パスによって目的のパスを検索しようとします。ServletContext クラスの getRealPath メソッドで相対パスを指定すると，WAR ファイルを展開したディレクトリでの相対パスが取得されます。

また，ファイルにアクセスする場合は，必ずファイルをクローズしてください。WAR ファイル展開ディレクトリでファイルにアクセスしてクローズしないと，J2EE サーバで正常にアンデプロイできなくなります。WAR ファイルの展開ディレクトリ下のパスを指定していない場合でもファイルをクローズしていないと，J2EE サーバの起動中にファイルを削除できないなどの現象が発生します。

(11) 例外発生時のエラーページの設定について

JSP，サーブレットへのアクセスで例外が発生した場合，Web コンテナのデフォルトの処理では例外のステータスコードをブラウザに返します。このデフォルトの処理を変える場合は JSP の errorPage の指定や web.xml でエラーページを設定してください。

(12) クラスローダの取得に関する注意

J2EE アプリケーション内のコードから Component Container のクラスローダを取得して，次に示すメソッドを使用する場合に，java.net.JarURLConnection クラスが使用されます。

- getResource(String).openConnection().getInputStream()
- getResource(String).openStream()

これらのメソッドが呼び出される過程で java.net.JarURLConnection クラスの openConnection メソッドが呼び出され，該当する URL に指定された JAR ファイルがオープンされます。この JAR ファイルは close メソッドを明示的に呼ばないかぎり，オープンされたままになり削除できません。これらのメソッドは J2EE アプリケーション内で使用しないでください。また，JAR ファイルに対する操作が必要で java.net.JarURLConnection クラスの openConnection メソッドを使用する場合には，java.net.JarURLConnection の getJarFile メソッドが返す JarFile インスタンスの close メソッドを必ず呼び出すようにしてください。

(13) URLConnection クラス使用時の注意

java.net.URLConnection クラスは setUseCaches(boolean)メソッドを使用して，指定された URL に対してコネクションを取得するときにキャッシュの情報を利用するかどうかを指定できます。
URLConnection クラスに対して setUseCaches(false)メソッドを指定した場合に，コネクションごとに対象のオブジェクトが生成されます。J2EE アプリケーション内のコードから使用する場合には，J2EE サーバの JavaVM がメモリ不足となるおそれがあります。

(14) ネイティブライブラリのロードに関する注意

System.loadLibrary メソッドを使用して，サーブレットおよび JSP からネイティブライブラリをロードしないでください。サーブレットおよび JSP でネイティブライブラリをロードすると，JNI 仕様の制約によって，java.lang.UnsatisfiedLinkError が発生することがあります。ネイティブライブラリのロードが必要な場合は，System.loadLibrary メソッドを呼び出すコンテナ拡張ライブラリを作成し，サーブレットおよび JSP からコンテナ拡張ライブラリを参照するように実装してください。コンテナ拡張ライブラリの作成については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「19. コンテナ拡張ライブラリ」を参照してください。

(15)　ユーザスレッドの使用方法

アプリケーションを構成するサーブレットおよびJSPからスレッドを生成して，使用できます。ユーザがプログラムの中で明示して生成するスレッドのことを，**ユーザスレッド**といいます。

ユーザスレッドは，生成後の動作のしかた（ライフサイクル）によって，次の二つに分けられます。

- サービスメソッドやinitメソッドの範囲内で動作させる。
- サービスメソッドやinitメソッドのバックグラウンドで動作させる。

ユーザスレッドの使用条件

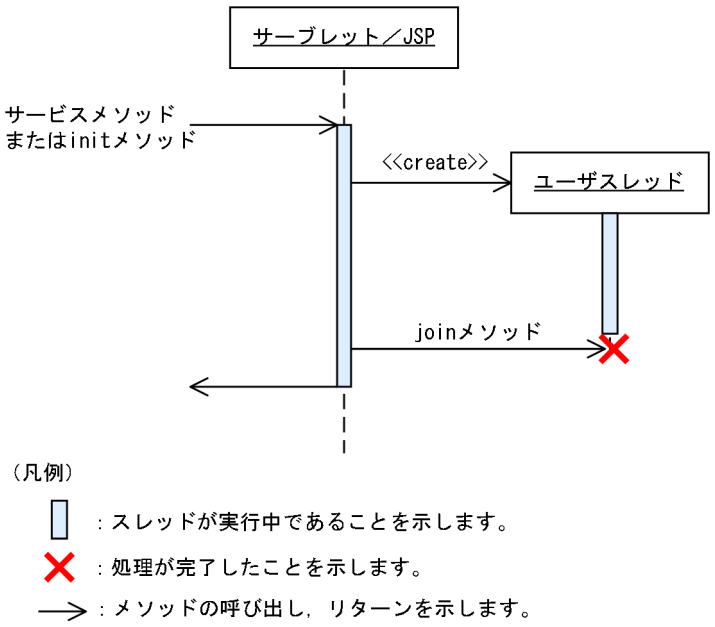
- ユーザスレッドは，Enterprise Beanでは使用できません（EJB仕様で，Enterprise Beanからのスレッドの生成が禁止されているため）。

ユーザスレッドを使用する場合のライフサイクルについて説明します。

(a)　サービスメソッドやinitメソッドの範囲内で動作させる場合

サービスメソッドやinitメソッドでユーザスレッドの処理を完了させるモデルです。このモデルの処理の流れを次の図に示します。

図 9-5　サービスメソッドやinitメソッドの範囲内で動作させる場合の処理

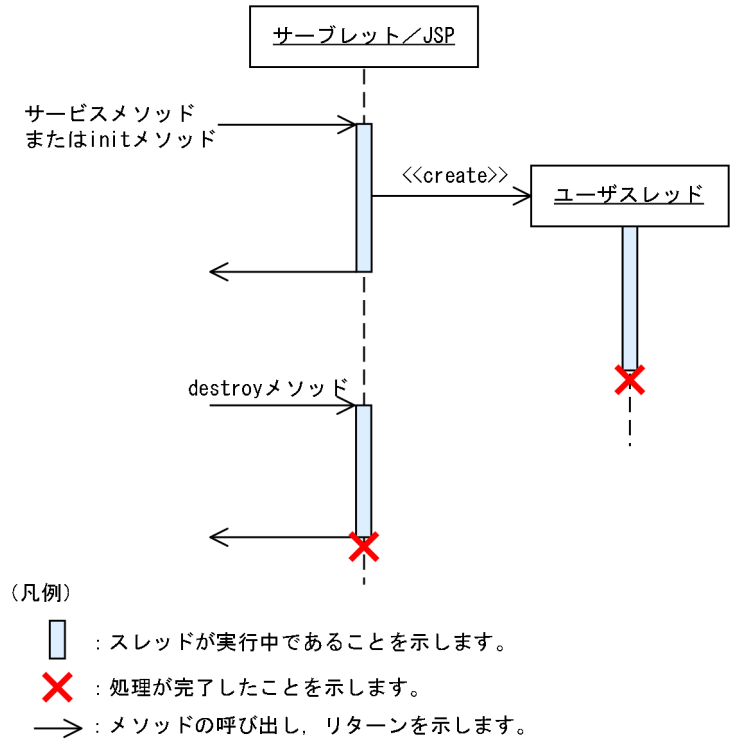


サービスメソッドやinitメソッドの呼び出しの範囲内で，ユーザスレッドを生成します。サービスメソッドやinitメソッドでは，joinメソッドによってユーザスレッドの処理が完了するのを待ってから，リターンします。

(b)　サービスメソッドやinitメソッドのバックグラウンドで動作させる場合

サービスメソッドやinitメソッドでユーザスレッドを生成し，その後ユーザスレッドをバックグラウンドで動作させるモデルです。このモデルの処理の流れを次の図に示します。

図 9-6　サービスメソッドや init メソッドのバックグラウンドで動作させる場合の処理



ユーザスレッドを生成したサービスメソッドや init メソッドは，ユーザスレッドを生成したあと，処理の完了を待たないでリターンします。ただし，アプリケーションを停止したあとは，ユーザスレッドから J2EE サービスを利用できなくなります。したがって，アプリケーションの停止によって javax.servlet.ServletContextListener の contextDestroyed メソッドか，JSP または Servlet の destroy メソッドでユーザスレッドを停止すれば問題ありません。

(16)　セッション情報の永続化について

Web コンテナではセッション情報の永続化はサポートされません。Web コンテナではセッション情報は，正常，異常に関係なく Web コンテナが終了すると失われます。セッション情報を保持したい場合は，セッションフェイルオーバ機能を使用してください。

また，web.xml で<distributable>タグを指定した場合，および Serializable でないオブジェクトをセッション情報として登録した場合も IllegalArgumentException は発生しません。

(17)　init メソッドおよび destroy メソッドをオーバーライドしていない場合に出力されるメッセージ

init メソッドおよび destroy メソッドをオーバーライドしていないサーブレットを初期化または終了すると，次の形式のログがサーブレットログに出力されます。

- メッセージ ID：KDJE39037-I
- メッセージ本文：path="aa....aa" :bb....bb: init※

aa....aa

「/」から始まるコンテキストパスを表します。

bb....bb

web.xml の<servlet-name>タグで指定したサーブレット名を表します。デフォルトマッピングのサーブレットの場合は，「org.apache.catalina.INVOKER.<クラス名>」となります。

注※

init メソッドの場合は「init」，destroy メソッドの場合は「destroy」となります。出力されるメッセージは，それぞれ javax.servlet.GenericServlet クラスの init メソッドおよび destroy メソッドで出力されるログです。したがって，init メソッドまたは destroy メソッドをオーバーライドしたサーブレットではこれらのメッセージは出力されません。

また，JSP の場合は，page ディレクティブの extends 属性で指定する JSP の基底クラスで init メソッドおよび destroy メソッドをオーバーライドしなかった場合，同様のメッセージが出力されます。その場合，サーブレット名は"com.hitachi.software.web.servlet-name.jsp"となります。JSP で page ディレクティブの extends 属性を指定しなかった場合は，init メソッドのログだけが出力され，destroy メソッドのログは出力されません。

ただし，サーブレットの場合も JSP の場合も，init メソッドおよび destroy メソッドをオーバーライドしてスーパークラスの init メソッドおよび destroy メソッドを呼ぶときは，このメッセージを出力します。

(18) javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性について

javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性について，Servlet で例外をスローした場合と JSP ファイルで例外をスローした場合の二つに分けて説明します。

(a) Servlet で例外をスローした場合

Servlet でスローした例外クラスが java.lang.Error，またはその派生クラスの場合

javax.servlet.ServletException クラスの例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。Servlet でスローした例外は，javax.servlet.ServletException クラスの getRootCause メソッドで取得できます。

Servlet でスローした例外クラスが java.lang.Error，またはその派生クラス以外のクラスの場合

Servlet でスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

(b) JSP ファイルで例外をスローした場合

- エラーページが JSP ファイルの場合

  web.xml の<error-page>タグでエラーページを指定した場合

  web.xml の<error-page>タグでエラーページを指定した場合について，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

JSP ファイルでスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

JSP 1.2

JSP ファイルでスローした例外クラスが次のクラスのどれかであれば，JSP ファイルでスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

・java.io.IOException，またはその派生クラス

・java.lang.RuntimeException，またはその派生クラス

・javax.servlet.ServletException，またはその派生クラス

JSP ファイルでスローした例外クラスがこれら以外の場合，javax.servlet.ServletException クラスの例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。JSP ファイルでスローした例外は，javax.servlet.ServletException クラスの getRootCause メソッドで取得できます。

page ディレクティブの errorPage 属性でエラーページを指定した場合

page ディレクティブの errorPage 属性でエラーページを指定した場合について，エラーページで page ディレクティブの isErrorPage 属性に true を指定した場合と false を指定した場合に分けて示します。

エラーページで page ディレクティブの isErrorPage 属性に true を指定した場合

JSP ファイルでスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

エラーページで page ディレクティブの isErrorPage 属性に false を指定した場合

javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に値は設定されません。

- エラーページが Servlet の場合

web.xml の<error-page>タグでエラーページを指定した場合

エラーページが JSP ファイルの場合の，web.xml の<error-page>タグでエラーページを指定した場合と同様です。

page ディレクティブの errorPage 属性でエラーページを指定した場合

javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に値は設定されません。

(19) バイナリデータを含む Web アプリケーションの操作について

バイナリデータを含む Web アプリケーションでは，次のことに注意してください。

- クライアントから送信されたバイナリデータへのリクエストを実行する場合

  バイナリデータへのリクエストで適用されるフィルタ内で，レスポンスオブジェクトからの PrintWriter を取得しないでください。

- クライアントから送信されたリクエストを処理するサーブレットまたは JSP がディスパッチする場合次の場所では，レスポンスオブジェクトからの PrintWriter を取得しないでください。
  - バイナリデータへのリクエストで適用されるフィルタ内
  - バイナリデータにディスパッチするサーブレットまたは JSP 内

> **参考**
>
> バイナリデータとは，拡張子にマッピングされた MIME タイプが"text/"から始まっていない静的コンテンツ，またはマッピングが存在しない静的コンテンツです。

(20) レスポンスの文字エンコーディングに関する注意

JSP またはサーブレットのレスポンスボディの文字エンコーディングが，UTF-16（16 ビット UCS 変換形式）の場合，ブラウザによって正しく表示できない場合があります。その場合は，JSP またはサーブレットの文字エンコーディングに，UTF-16BE（16 ビット UCS 変換形式のビッグエンディアンバイト順），または UTF-16LE（16 ビット UCS 変換形式のリトルエンディアンバイト順）を使用してください。

(21) javax.servlet.ServletRequest インタフェースの getServerName メソッドおよび getServerPort メソッドの戻り値について

getServerName メソッド，および getServerPort メソッドの戻り値について説明します。

Servlet 2.4 仕様以降では，Host ヘッダの有無によって，getServerName メソッド，および getServerPort メソッドの戻り値が異なります。Servlet 2.4 仕様以降での getServerName メソッド，および getServerPort メソッドの戻り値を次の表に示します。

表 9-13 getServerName メソッド，および getServerPort メソッドの戻り値（Servlet 2.4 仕様以降の場合）

Host ヘッダの有無	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
あり	Host ヘッダの「:」より前の部分	Host ヘッダの「:」よりあとの部分
なし	解決されたサーバ名または IP アドレス	クライアントとの接続を受け付けたサーバのポート番号

アプリケーションサーバでは，getServerName メソッド，および getServerPort メソッドの戻り値は，HTTP リクエストと，使用する機能の組み合わせによって得られます。なお，HTTP 1.1 のリクエストに Host ヘッダが含まれない場合，HTTP 1.1 仕様に従って，400 エラーとなります。また，HTTP 1.1 仕様では，リクエストラインのリクエスト URI が絶対 URI の場合，ホストにはリクエスト URI のホストを使用して，Host ヘッダの内容は無視するように定義されています。なお，Servlet 仕様では明記されていませんが，HTTP 仕様に従って，リクエストラインの URI に含まれるホスト名を優先するようになっています。

HTTP リクエストと，使用する機能の組み合わせによって得られる，getServerName メソッド，および getServerPort メソッドの戻り値を次の表に示します。なお，ゲートウェイ指定機能を使用している場合の，getServerName メソッド，および getServerPort メソッドの戻り値については，表 9-13 を参照してください。

表 9-14　getServerName メソッド，および getServerPort メソッドの戻り値（アプリケーションサーバの場合）

HTTP リクエスト		使用する機能	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類			
あり	絶対 URI	Web サーバ連携	リクエストラインのホスト名	リクエストラインのポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	Host ヘッダのホスト名	Host ヘッダのポート番号
		インプロセス HTTP サーバ	Host ヘッダのホスト名	Host ヘッダのポート番号
なし	絶対 URI	Web サーバ連携	リクエストラインのホスト名	リクエストラインのポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	Web サーバのホスト名または IP アドレス※2	Web サーバのポート番号
		インプロセス HTTP サーバ	J2EE サーバのホスト名または IP アドレス※1	インプロセス HTTP サーバのポート番号

注※1　java.net.InetAddress.getLocalHost メソッド，または getHostName メソッドの戻り値となります。

注※2　HTTP Server を使用する場合，ServerName ディレクティブに指定した値となります。ServerName ディレクティブについては，マニュアル「HTTP Server」を参照してください。

ゲートウェイ指定機能使用時の getServerName メソッド，および getServerPort メソッドの戻り値を次の表に示します。

表 9-15　ゲートウェイ指定機能使用時の getServerName メソッド，および getServerPort メソッドの戻り値（アプリケーションサーバの場合）

HTTP リクエスト		使用する機能	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類			
あり	絶対 URI	Web サーバ連携	リクエストラインのホスト名	リクエストラインのポート番号

HTTP リクエスト		使用する機能	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類			
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	Host ヘッダのホスト名	Host ヘッダのポート番号
		インプロセス HTTP サーバ	Host ヘッダのホスト名	Host ヘッダのポート番号
なし	絶対 URI	Web サーバ連携	ゲートウェイ指定機能で指定したホスト名	ゲートウェイ指定機能で指定したポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	ゲートウェイ指定機能で指定したホスト名	ゲートウェイ指定機能で指定したポート番号
		インプロセス HTTP サーバ	ゲートウェイ指定機能で指定したホスト名	ゲートウェイ指定機能で指定したポート番号

(22) javax.servlet.ServletException クラスのコンストラクタで指定した根本原因の例外の取得について

アプリケーションサーバでは，コンストラクタ ServletException(String, Throwable)または ServletException(Throwable)で指定した根本原因の例外を getCause メソッドで取得できます。なお，getRootCause メソッドでも取得できます。ただし，07-60 以前のバージョンでは getCause メソッドには null を返します。

javax.servlet.ServletException クラスのコンストラクタで指定した根本原因の例外の取得について，互換用のパラメタおよび注意事項について説明します。

- 互換用のパラメタ

  07-60 以前と同じ動作にする場合，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で互換用のパラメタ webserver.servlet_api.exception.getCause.backcompat に true を指定します。パラメタの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

  互換用のパラメタ webserver.servlet_api.exception.getCause.backcompat の指定値による getCause メソッドおよび getRootCause メソッドの戻り値の違いを次の表に示します。

表 9-16　webserver.servlet_api.exception.getCause.backcompat の指定値による getCause メソッドおよび getRootCause メソッドの戻り値の違い

メソッド	webserver.servlet_api.exception.getCause.backcompat の指定値	
	true	false
getCause()	×	○
getRootCause()	○	○

（凡例）○：根本原因の例外を返す　×： null を返す

なお，この互換用のパラメタの指定内容は，javax.servlet.jsp.JspException クラスの getCause メソッドおよび getRootCause メソッドの動作にも適用されます。

- 注意事項

  根本原因の例外を getCause メソッドの実装によって取得できる場合，コンストラクタ ServletException(String, Throwable)または ServletException(Throwable)で生成した ServletException オブジェクトに対して initCause(Throwable)を呼び出すことはできません。initCause(Throwable)を呼び出した場合，java.lang.IllegalStateException 例外がスローされます。

(23)　javax.servlet.ServletOutputStream オブジェクトに対する flush メソッドの実行についての注意

アプリケーションサーバでは，javax.servlet.ServletResponse オブジェクトから取得する javax.servlet.ServletOutputStream オブジェクトに対して，close メソッドを実行したあとで flush メソッドを実行しても，java.io.IOException 例外をスローしません。

(24)　リクエスト URI の正規化

アプリケーションサーバでは，リクエスト URI に含まれる文字列は，正規化されたあと，次に示すマッチング処理で使用されます。

- コンテキストパスとコンテキストルートのマッチング
- サーブレットおよび JSP の URL パターンとのマッチング
- デフォルトマッピングとのマッチング
- 静的コンテンツとのマッチング
- フィルタの URL パターンとのマッチング
- web.xml の<error-page>タグ，または JSP の page ディレクティブの errPage 属性で指定するエラーページとのマッチング
- アクセスを制限する URL パターンとのマッチング
- ログイン認証の URL 判定
- リクエストのフォワードおよびインクルード

- HTTP レスポンス圧縮フィルタの URL パターンとのマッチング
- URL グループ単位の同時実行スレッド数制御の URL パターンとのマッチング
- インプロセス HTTP サーバのエラーページのカスタマイズ
- インプロセス HTTP サーバのリダイレクトによるリクエストの振り分け

(25) javax.servlet.http.HttpServletRequest インタフェースの getRequestURI メソッドおよび getRequestURL メソッドの戻り値について

javax.servlet.http.HttpServletRequest インタフェースの getRequestURI メソッドおよび getRequestURL メソッドでは，正規化された URL が戻り値となります。

(26) welcome ファイルに URL マッピングされた Servlet または JSP の指定

リクエスト URL が URL マッピングされた Servlet または JSP と一致しないで，welcome ファイルに転送される必要がある場合，Web コンテナでは次のように転送先の welcome ファイルが選択されます。

まず，指定された welcome ファイル名から静的コンテンツや JSP ファイルの候補が優先して選択されます。該当するものがない場合，URL マッピングされた Servlet または JSP の候補が選択されます。

welcome ファイルに関する注意事項について説明します。

- **welcome ファイル転送方式による制約**

  welcome ファイルの転送は，HTTP リダイレクト（HTTP ステータスコード 302 でブラウザがリダイレクトする）によって実現しています。この転送方式には制約があるため，URL 設計の際に次のことに注意してください。

  - POST リクエストを受け付けた際，ブラウザから送信されたリクエストボディの情報を転送先の welcome ファイルに引き継げません。POST された情報がフォーム入力形式（Content-Type が application/x-www-form-urlencoded）の場合だけ，Web コンテナが生成する welcome ファイル転送先 URL のクエリ文字列に情報を付与する形で引き継げます。ただし，この場合も，リクエストボディの情報が多い場合に転送先 URL が長くなり過ぎる，ブラウザのアドレスバーにクエリ文字列として情報がそのまま見える，などについて考慮が必要です。
  - 転送先の welcome ファイルのサーブレットが doGet メソッドを実装していない場合，ブラウザに「400 Bad Request」（HTTP/1.1 以外の場合）または「405 Method Not Allowed」（HTTP/1.1 の場合）が表示されます。
  - Web アプリケーションから javax.servlet.RequestDispatcher インタフェースの include メソッドを呼び出した際，インクルードする対象の URL として welcome ファイルが存在するディレクトリを指定していても，転送先の welcome ファイルのコンテンツは挿入されません。
- **JSP 事前コンパイル済み環境での welcome ファイルの追加**

JSP 事前コンパイル済みの Web アプリケーションに，welcome ファイルに指定した JSP ファイルを追加する場合，JSP ファイルの追加後に JSP 事前コンパイルを再度実行する必要があります。JSP 事前コンパイルを再度実行しなかった場合，正しく welcome ファイル転送処理が実行されません。

- **サーブレットクラスが参照できないサーブレットの welcome ファイルの指定**

  サーブレットクラスが参照できないサーブレットを welcome ファイルに指定しないでください。サーブレットクラスが参照できないサーブレットを指定した場合，正しく welcome ファイル転送処理が実行されません。

- **ディレクトリが存在しないパスへの welcome ファイル要求**

  Web アプリケーション内のリソースとして存在しないディレクトリのパスに対するリクエストの場合，リクエスト URL の末尾が「/」であっても welcome ファイル転送処理は実行されません。

(27) サーブレット，フィルタ，リスナの開始・終了順序

Web アプリケーションを開始すると，リクエストの受付を開始する前に次の順序で初期化処理をすることが Servlet 2.4 仕様で明確化されました。アプリケーションサーバでは，Servlet2.3 以前の Web アプリケーションでも同じ順序で初期化処理をします。Web アプリケーション開始時のサーブレット，フィルタ，およびリスナは次の順序で開始されます。

1. リスナの開始（インスタンスの生成※1，@PostConstruct アノテーションのメソッドおよび ServletContextListener の contextInitialized メソッドの呼び出し※2）
2. フィルタの開始（インスタンスの生成※1，@PostConstruct アノテーションのメソッドおよび init メソッドの呼び出し）
3. load-on-startup タグで指定された Servlet/JSP の開始（インスタンスの生成※1，@PostConstruct アノテーションのメソッドおよび init メソッドの呼び出し）

注※1　Servlet 3.0 以降では，API 呼び出しによって動的にサーブレット，フィルタ，リスナを追加できますが，インスタンスを指定する API 呼び出しによって定義を追加したサーブレット，フィルタ，リスナについては，インスタンス生成済みのため，Web コンテナではインスタンスを生成しません。

注※2　リスナの contextInitialized()メソッドの呼び出しで例外が発生しても，KDJE39103-E のメッセージを出力して Web アプリケーションの開始処理を継続します。

なお，web.xml の<load-on-startup>要素によって Web アプリケーション開始時の初期化処理実行を指定しなかったサーブレットについては，初回リクエスト実行時にサーブレットのインスタンスの生成および init()メソッドを呼び出します。

このとき，サーブレットのインスタンスの生成および init()メソッドはフィルタより前に呼び出します。

Web アプリケーション終了時のサーブレット，フィルタ，およびリスナは次の順序で終了します。

1. 開始済みの Servlet/JSP の終了(destroy メソッド，@PreDestroy アノテーションのメソッド呼び出し)
2. フィルタの終了(destroy メソッド，@PreDestroy アノテーションのメソッド呼び出し)

3. リスナの終了(@PreDestroy アノテーションのメソッド呼び出し)

(28) Web アプリケーション内の静的コンテンツへのアクセス

Web アプリケーション内の静的コンテンツへのアクセス時に使用できるメソッドは，GET，HEAD，POST，TRACE，OPTIONS のどれかです。

POST メソッドを使用した場合，GET メソッド使用時と同様，静的コンテンツの内容を応答します。

(29) 文字エンコーディングに関する注意

同じ Web アプリケーション内では，web.xml で指定したエラーページと，HTTP レスポンスに文字エンコーディングを使用するサーブレットおよび JSP に，同じ文字エンコーディングを使用してください。

(30) クエリ文字列にイコール（"="）以降だけ指定した場合の戻り値について

リクエストのクエリ文字列にイコール（"="）以降しか指定していない場合（例えば，http://localhost/application/getparam.jsp?=param のような場合），使用している Web サーバ種別によって，javax.servlet.ServletRequest インタフェースのリクエストパラメタを取得する Servlet API の戻り値が異なります。

Web サーバ種別ごとの Servlet API の戻り値を次に示します。

- Web サーバ連携機能または簡易 Web サーバを使用している場合
  - getParameter メソッド
    空文字（""）を指定するとイコール（"="）以降に指定されたパラメタを返します。
  - getParameterMap メソッド
    空文字（""）をキーとするパラメタを含む java.util.Map オブジェクトを返します。
  - getParameterNames メソッド
    空文字（""）を含む java.util.Enumeration オブジェクトを返します。
  - getParameterValues メソッド
    空文字（""）を指定するとイコール（"="）以降に指定されたパラメタを返します。
- インプロセス HTTP サーバを使用している場合
  - getParameter メソッド
    空文字（""）を指定しても null を返します。
  - getParameterMap メソッド
    空の java.util.Map オブジェクトを返します。
  - getParameterNames メソッド
    空の java.util.Enumeration オブジェクトを返します。
  - getParameterValues メソッド

空文字（""）を指定しても null を返します。

(31) javax.servlet.http.HttpServletResponse インタフェースの containsHeader メソッドについて

以下のレスポンスヘッダは Web コンテナにより自動的にレスポンスにセットされる場合があります。このようなレスポンスヘッダは，javax.servlet.http.HttpServletResponse インタフェースの containsHeader メソッドで，レスポンスにセットされているかどうかを確認できません。

- Web サーバ連携の場合
  - Content-Length
  - Content-Type
  - Set-Cookie
- インプロセス HTTP サーバの場合
  - Connection
  - Content-Language
  - Content-Length
  - Content-Type
  - Date
  - Server
  - Set-Cookie
  - Transfer-Encoding

(32) アプリケーションサーバのライブラリに関する注意

アプリケーションサーバのライブラリを J2EE アプリケーションに含めると，ライブラリのバージョン不整合などが原因で，アプリケーションのインポートや開始，実行で不正な動作になることがあります。そのため，製品が使用方法として明示している場合を除いて，アプリケーションサーバのライブラリは J2EE アプリケーションに含めないようにしてください。

9.4.2 サーブレット実装時の注意事項

サーブレットを実装するときの注意事項を示します。

(1) ゲートウェイ指定機能を使用する場合の注意

Web コンテナにゲートウェイ情報を通知し，welcome ファイルや FORM 認証画面に正しくリダイレクトする**ゲートウェイ指定機能**を使用できます。ゲートウェイ指定機能については，「5.10　Web コンテナへのゲートウェイ情報の通知」を参照してください。

ゲートウェイ指定機能を使用する場合，一部のサーブレット API の動作が変わります。ゲートウェイ指定機能使用時のサーブレット API の注意事項を，使用するメソッドごとに示します。

- javax.servlet.http.HttpServletResponse クラスの sendRedirect メソッド
  引数に相対 URL を指定し，Host ヘッダのないリクエストの場合，リダイレクト先 URL のホスト名とポート番号は，ゲートウェイ指定機能で指定した値となります。引数に相対 URL を指定し，ゲートウェイ指定機能でスキームを https と見なすように設定した場合，リダイレクト先 URL のスキームは常に「https」となります。
- javax.servlet.ServletRequest インタフェースの getRequestURL メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合，戻り値は常に「https://」で始まる URL となります。
- javax.servlet.ServletRequest インタフェースの getServerName メソッド
  ゲートウェイ指定機能でリダイレクト先 URL のホスト名を指定し，リクエストに Host ヘッダがない場合，戻り値は指定した値となります。
- javax.servlet.ServletRequest インタフェースの getServerPort メソッド
  ゲートウェイ指定機能でリダイレクト先 URL のポート番号を指定し，リクエストに Host ヘッダがない場合，戻り値は指定した値となります。ゲートウェイ指定機能でリダイレクト先 URL のホスト名を指定し，ポート番号を省略した場合，戻り値はリクエストのスキームが http であれば「80」，https であれば「443」となります。
- javax.servlet.ServletRequest インタフェースの getScheme メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合，戻り値は常に「https」となります。
- javax.servlet.ServletRequest インタフェースの isSecure メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合，戻り値は常に「true」となります。
- javax.servlet.ServletRequest インタフェースの getAttribute メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合でも，次の属性は取得できません。
  - javax.servlet.request.cipher_suite（Web サーバに Microsoft IIS を使用している場合は，ゲートウェイ指定機能の使用に関係なく取得できません）
  - javax.servlet.request.key_size
  - javax.servlet.request.X509Certificate

9.4.3　EL2.2 仕様で追加，変更された仕様についての注意事項

EL2.2 仕様で追加，変更された仕様を，アプリケーションサーバ上で使用するときの注意事項を示します。EL2.2 仕様については，EL2.2 仕様書を参照してください。

- アプリケーションサーバでは EL2.2 をサポートしますが，EL2.1 の処理系と EL2.2 の処理系の両方を保持します。使用する処理系は簡易構築定義ファイルの次のパラメタで切り替えられます。このパラメタは簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

  ```
  webserver.jsp.el2_2.enabled
  ```

  webserver.jsp.el2_2.enabled パラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.11.2　J2EE サーバ用ユーザプロパティを設定するパラメタ」を参照してください。

- 一つの Bean 内に同じ数のパラメタを持つメソッドが複数定義された場合，最初に定義されたメソッドに型変換されます。そのため，メソッドを呼び出すときの引数の型が，最初に定義されたメソッドに合う場合は正しく呼び出されますが，引数の型が合わない場合は ELException が発生します。

9.5　V9 互換モードで事前コンパイルをする場合の注意事項

V9 互換モードで JSP の事前コンパイルをする場合の注意事項を次に示します。

- JSP の事前コンパイルコマンド（cjjspc）は推奨モード用と V9 互換モード用でインストール先が異なります。V9 互換モード用の事前コンパイルコマンドのインストール先については，「17.　Web アプリケーションで使用するコマンド」を参照してください。
- 推奨モードで事前コンパイルした結果を V9 互換モードで実行できません。使用した場合の動作は未サポートです。

10 J2EE サーバで使用するファイル

この章では，J2EE サーバで使用するファイルの形式，格納先，機能，指定できるキーなどについて説明します。ここでは，推奨モードと異なる内容だけ説明します。

10.1.1　usrconf.properties（J2EE サーバ用ユーザプロパティファイル）

(1)　cosminexus.jpa から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

キー名称	内容	省略値	VR
cosminexus.jpa.logging.level.operation.<category>	J2EE サーバの JPA 機能を利用する場合に，JPA プロバイダ稼働ログのカテゴリごとのログレベルを指定します。カテゴリ名，およびログレベルは大文字と小文字を区別します。 未設定の場合，稼働ログには出力されません。セキュリティや性能面に対する影響があるため，出力レベルの設定には注意してください。 Off を指定した場合： JPA プロバイダ稼働ログに対するログを出力しません。 Information を指定した場合： JPA プロバイダ稼働ログに JPA の稼働情報を出力します。 Detail を指定した場合： JPA プロバイダ稼働ログに JPA の詳細な稼働情報および，Information で出力される情報を出力します。	Off	08-00
cosminexus.jpa.exception.logging.sql※	JPA プロバイダが SQL 文を実行してデータベースから例外を受け取った場合に，例外メッセージに例外が発生した原因となった SQL 文を含むどうかを指定します。 Off を指定した場合： JPA プロバイダが実行した SQL 文と?パラメタ（プレースホルダ）への指定値を例外メッセージに含みません。 Information を指定した場合： JPA プロバイダが実行した SQL 文を例外メッセージに含みます。 Detail を指定した場合： JPA プロバイダが実行した SQL 文と?パラメタ（プレースホルダ）への指定値を例外メッセージに含みます。	Off	08-00

注※

このプロパティで指定した値は，例外ログの出力内容にも反映されます。

Information，および Detail の場合は，SQL 文や?パラメータ（プレースホルダ）の内容が例外ログにも出力されるため，セキュリティに注意してください。開発時や保守のために必要に応じて値を指定してください。

バイナリデータを扱う場合，?パラメータ（プレースホルダ）にはバイナリオブジェクトのハッシュ値が出力されます。

SQL 発行の準備が完了する以前に通信異常によるコネクション取得エラーなどが発生した場合，?パラメータ（プレースホルダ）値を取得できない場合があります。

(2) ejbserver.jpa から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

キー名称	内容	省略値	VR
ejbserver.jpa.defaultJtaDsName	デフォルトの JTA データソースの参照を指定します。このプロパティは persistence.xml の jta-data-source を指定しなかった場合，または空白文字を指定した場合に使用されます。	なし	08-00
ejbserver.jpa.defaultNonJtaDsName	デフォルトの非 JTA データソースの参照を指定します。このプロパティは persistence.xml の non-jta-data-source を指定しなかった場合または空白文字を指定した場合に使用されます。	なし	08-00
ejbserver.jpa.defaultProviderClassName	デフォルトの JPA プロバイダクラス名を指定するプロパティです。このプロパティは，persistence.xml の provider を指定しなかった場合または空白文字が指定した場合に使用されます。	com.hitachi.software.jpa.PersistenceProvider	08-00
ejbserver.jpa.disable※	アプリケーションサーバの JPA 機能を使用する場合に指定します。 true を指定した場合： アプリケーションサーバの JPA 機能は無効になります。 false を指定した場合： アプリケーションサーバの JPA 機能は有効になります。	false	08-20
ejbserver.jpa.overrideJtaDsName	persistence.xml の jta-data-source に指定した値，および ejbserver.jpa.defaultJtaDsName に指定した値より優先して使用する JTA データソースの参照を指定します。	なし	08-00
ejbserver.jpa.overrideNonJtaDsName	persistence.xml の non-jta-data-source に指定した値，および ejbserver.jpa.defaultNonJtaDsName に指定した値より優先して使用する非 JTA データソースの参照を指定します。	なし	08-00
ejbserver.jpa.overrideProvider	persistence.xml の provider に指定した値，および ejbserver.jpa.defaultProviderClassName に指定した値より優先して使用する JPA プロバイダクラス名を指定します。	なし	08-00
ejbserver.jpa.emfprop.<property key>	JPA プロバイダ独自のプロパティのキーを指定します。すべての永続化ユニットのデプロイ時に，「ejbserver.jpa.emfprop.」プリフィックスを除去したプロパティが JPA プロバイダに渡されます。	なし	08-00

注※　ejbserver.jpa.disable=true を指定した場合の注意事項

アプリケーションに persistence.xml が含まれる場合，アプリケーションサーバはアプリケーション開始時に persistence.xml を読み込まなくなります。また，アプリケーションのリロード機能を利用している場合，persistence.xml が更新検知の対象ではなくなります。

アプリケーションがアプリケーションサーバの管理する永続化コンテキストまたは永続化ユニットを利用している場合，アプリケーションの開始ができなくなります。

(3) ejbserver.logger から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

「関連情報」とは，指定したキーに関する情報の参照先です。マニュアル名称の「アプリケーションサーバ」を省略しています。

キー名称	内容	省略値	VR	関連情報
ejbserver.logger.channels.define.<チャネル名>※.filenum	J2EE サーバのログファイルの面数を，1～16の整数で指定します。	• チャネル名がWebAccessLogFileの場合 16 • チャネル名がMaintenanceLogFile またはWebServletLogFileの場合 4 • チャネル名が上記以外の場合 2	－	
ejbserver.logger.channels.define.<チャネル名>※.filesize	J2EE サーバのログファイルのサイズ（単位：バイト）を，4096～2147483647 の整数で指定します。	• チャネル名がWebAccessLogFileの場合 2097152 • チャネル名がMaintenanceLogFile の場合 16777216	－	

キー名称	内容	省略値	VR	関連情報
		• チャネル名がWebServletLogFileの場合 4194304 • チャネル名が上記以外の場合 1048576		

（凡例）

－：08-00 より前のバージョンを示します。

空欄：関連情報はありません。

注※

チャネル名称として次に示す名称を設定できます。

- アプリケーションサーバシステムのログ出力用のチャネル

  MessageLogFile，MaintenanceLogFile，ExceptionLogFile，ConsoleLogFile，EJBContainerLogFile，WebContainerLogFile，WebServletLogFile，UserOutLogFile，UserErrLogFile，WebAccessLogFile，JPAOperationLogFile，JPAMaintenanceLogFile

- リソース枯渇監視ログ出力用のチャネル

  MemoryWatchLogFile，FileDescriptorWatchLogFile，ThreadWatchLogFile，ThreaddumpWatchLogFile，RequestQueueWatchLogFile，HttpSessionWatchLogFile，ConnectionPoolWatchLogFile

資料の取得については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「2.3　資料の取得」を参照してください。

(4)　ejbserver.server から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

「関連情報」とは，指定したキーに関する情報の参照先です。マニュアル名称の「アプリケーションサーバ」を省略しています。

キー名称	内容	省略値	VR	関連情報
ejbserver.server.eheap.ajp13.enabled	リダイレクタとの通信用オブジェクトをExplicit ヒープに配置するかどうかを指定します。	true	08-00	

キー名称	内容	省略値	VR	関連情報
	true を指定した場合： 　リダイレクタとの通信用オブジェクトを Explicit ヒープに配置します。 false を指定した場合： 　リダイレクタとの通信用オブジェクトを Java ヒープ領域に配置します。 ただし，JavaVM オプション HitachiUseExplicitMemory を無効にした場合，このプロパティは無効（false 指定時と同じ挙動）となります。			

（凡例）

空欄：関連情報はありません。

(5) webserver.connector から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

「関連情報」とは，指定したキーに関する情報の参照先です。マニュアル名称の「アプリケーションサーバ」を省略しています。

キー名称	内容	省略値	VR	関連情報
webserver.connector.ajp13.backlog	リダイレクタからの接続要求の最大の待ち行列数を指定します。1～2147483647 の整数で指定します。 有効な最大値は実行するプラットフォームで指定できる Socket の Listen キューの最大値となります。実際の Listen キューの最大値は OS によって異なるため，詳細は各 OS の listen 関数についてのマニュアルを参照してください。 このキーに指定した値は，java.net.ServerSocket クラスのコンストラクタの backlog 引数に設定されます。ただし，この指定値が OS の制限値を超えた場合は，OS の制限値が設定されたものと解釈され，エラーにはなりません。制限値は OS によって異なります。制限値を拡張する方法については，OS のマニュアルを参照してください。	100	－	
webserver.connector.ajp13.bind_host	Web サーバ連携で使用する IP アドレスまたはホスト名称を指定します。 IP アドレス，またはホスト名称の前後の半角スペースは無視されます。値を指定しない場合は，ワイルドカードアドレスが使用されます。	なし	－	「5.7　IP アドレス指定（Web サーバ連携）」

キー名称	内容	省略値	VR	関連情報
	このプロパティを指定するときは，ワーカホスト名称にもローカルホスト名称または IP アドレスを指定する必要があります。 同一ホストで実行している Web サーバと Web サーバ連携をしている構成で，次のどちらかの設定をしている場合，Web コンテナは Web サーバからのリクエストを受信できません。 • webserver.connector.ajp13.bind_host プロパティにローカルホスト名称，または IP アドレスを指定し，リダイレクタのワーカホスト名称に localhost などのループバックアドレスを指定している。 • webserver.connector.ajp13.bind_host プロパティに localhost などのループバックアドレスを指定し，リダイレクタのワーカホスト名称にローカルホスト名称，または IP アドレスを指定している。			
webserver.connector.ajp13.max_threads	Web コンテナがリクエストを処理する同時実行数を指定します。※1 1～1024 の整数で指定します。 指定されたリクエストの同時実行数分のスレッドがサーバ起動時に生成されます。	10	－	「機能解説 基本・開発編 (Web コンテナ)」の「2.14　Web コンテナ単位での同時実行スレッド数の制御」
webserver.connector.ajp13.port	Web サーバとの通信に使用するポート番号を指定します。 1～65535 の整数で指定します。 すでにほかのアプリケーションで使用または確保されているポート番号は指定できません。また，複数の J2EE サーバで Web サーバとの通信に使用するポートのポート番号に同じ値を設定しないでください。同一のポート番号を指定した J2EE サーバは cjstartsv コマンドで複数起動できません。	8007	－	
webserver.connector.ajp13.receive_timeout	リクエスト受信処理のリダイレクタへのデータ要求処理で，リダイレクタからの応答を待つ時間（通信タイムアウト値）を 0～3600 の整数（単位：秒）で指定します。 0 を指定した場合は，リダイレクタからの応答を受け取るまで待ち続け，タイムアウトは発生しません。	600（秒）	－	
webserver.connector.ajp13.send_timeout	レスポンス送信処理のタイムアウト値を 0～3600 の整数（単位：秒）で指定します。	600	－	

キー名称	内容	省略値	VR	関連情報
	数値以外の文字列や範囲外の数値を指定した場合は，メッセージを出力し，デフォルト値を使用します。 タイムアウト値に 0，または TCP の持つデータ送信の再送タイマより長い時間を設定した場合，タイムアウト値は TCP の持つタイムアウト値になります。その場合，不正なタイムアウト値が指定されたことを示すメッセージは出力されません。			
webserver.connector.inprocess_http.backlog	Web クライアントからの接続要求を格納する TCP リスンキューの長さを 1～2147483647 の整数で指定します。 有効な指定値の最大値や実際に設定される TCP リスンキューの長さは OS によって異なります。 数値以外の文字列，範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	511	－	
webserver.connector.inprocess_http.bind_host	インプロセス HTTP サーバで使用する IP アドレスまたはホスト名を指定します。 IP アドレスまたはホスト名の前後の半角スペースは無視されます。値を指定しない場合は，ワイルドカードアドレスが使用されます。 指定されたホスト名または IP アドレスが解決できない場合，ローカルではないホストのホスト名または IP アドレスを指定した場合はメッセージが出力され，ワイルドカードアドレスが使用されます。	なし	－	
webserver.connector.inprocess_http.enabled	インプロセス HTTP サーバ機能を有効にするかどうかを指定します。 true を指定した場合： インプロセス HTTP サーバ機能を有効にします。 false を指定した場合： インプロセス HTTP サーバ機能を無効にします。 true または false 以外の文字列を指定した場合，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。 なお，インプロセス HTTP サーバ機能が有効の場合は，Web サーバ連携は使用できません。	false	－	
webserver.connector.inprocess_http.enabled_methods	アクセスを許可する HTTP メソッドを指定します。	GET,HEAD,POST,PUT,DELETE,OPTIONS	－	

キー名称	内容	省略値	VR	関連情報
	複数のメソッドを指定する場合は，コンマ（,）で区切ります。メソッド名にはHTTP/1.1で定義されたメソッドを指定します。 また，アスタリスク（*）を指定した場合は，すべてのメソッドが許可されます。 HTTPメソッドでは大文字，小文字が区別されるため，このプロパティに指定する値も区別されます。 メソッド名にはRFC2616で規定されている値を使用する必要があります。ただし，文字列"*"をメソッド名として指定できません。 各メソッド名の前後の空白文字※2は無視されます。不正な値，空文字列または空白文字※2を指定した場合は，メッセージが出力され，デフォルト値が設定されます。			
webserver.connector.inprocess_http.error_custom.list	エラーページカスタマイズ機能で使用するエラーページカスタマイズ定義名を指定します。 指定できる値の最大長は1024文字で，英数字（A～Z，a～z，0～9）またはアンダースコア（_）で構成される文字列で指定します。定義名一つの文字列長は1～32文字です。 定義名を複数指定する場合は，コンマ（,）で区切ります。コンマの前後の空白文字※2は無視されます。また，同じエラーページカスタマイズ定義名を複数回指定できません。 不正な値を指定した場合，メッセージが出力され，すべてのエラーページカスタマイズ定義は無効となります。	なし	－	
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file	エラーページカスタマイズ機能で，エラーページカスタマイズ時のレスポンスボディとして使用するファイルを絶対パスで指定します。 パスの区切り記号には”/”を使用します。 webserver.connector.inprocess_http.error_custom.listで設定されていないエラーページカスタマイズ定義名を使用してこのプロパティを設定した場合，プロパティは無効になります。 webserver.connector.inprocess_http.error_custom.listで指定したエラーページカスタマイズ定義名について，このプロパティ，またはwebserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.redirect_urlのどちらか一方だけを必ず指定してください。両方のプロパティを指定した場合，どちらも指定しなかった場合，絶対パスで指定しなかった場合，または存在しないファイルや読み取り権限のないファイルを指定した場	なし	－	

キー名称	内容	省略値	VR	関連情報
	合は，メッセージが出力され，このエラーページカスタマイズ定義は無効となります。 空文字列または空白文字※2 を指定した場合，プロパティは無効になります。			
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file.content_type	エラーページカスタマイズ機能で，エラーページカスタマイズ時のレスポンスの Content-Type ヘッダの値を指定します。 webserver.connector.inprocess_http.error_custom.list で設定されていないエラーページカスタマイズ定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file が設定されていない場合，プロパティは無効になります。	text/html	－	
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.redirect_url	エラーページカスタマイズ機能で，リダイレクト URL を絶対パスで指定します。 webserver.connector.inprocess_http.error_custom.list で設定されていないエラーページカスタマイズ定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 webserver.connector.inprocess_http.error_custom.list で指定したエラーページカスタマイズ定義名について，このプロパティ，または webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file のどちらか一方だけを必ず指定してください。 値が正しいかどうかのチェックは行われないため，実際に動作させて確認する必要があります。	なし	－	
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.request_url	エラーページカスタマイズ機能で，エラーページカスタマイズを適用するリクエスト URL をスラッシュ（/）で始まる絶対パスで指定します。ワイルドカード（*）はスラッシュの直後に 1 回だけ指定できます。"*"は必ずワイルドカードと解釈されるため，通常の文字としては使用できません。 また，このプロパティで指定した値と webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.status で指定した値は，ほかのエラーページカスタマイズ定義と完全に一致してはいけません。 webserver.connector.inprocess_http.error_custom.list で設定されていないエラーページカ	/*	－	

キー名称	内容	省略値	VR	関連情報
	スタマイズ定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 不正な値を指定した場合，メッセージが出力され，このエラーページカスタマイズ定義は無効となります。			
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.status	エラーページカスタマイズ機能で，エラーページのカスタマイズを行うレスポンスのステータスコードを 400～599 の整数で指定します。 このプロパティで指定した値，および webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.request_url で指定した値は，ほかのエラーページカスタマイズ定義と完全に一致してはいけません。 webserver.connector.inprocess_http.error_custom.list で設定されていないエラーページカスタマイズ定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 webserver.connector.inprocess_http.error_custom.list で指定したエラーページカスタマイズ定義名について，このプロパティは必ず指定してください。指定しなかった場合，または不正な値を指定した場合，メッセージが出力され，このエラーページカスタマイズ定義は無効となります。	なし	－	
webserver.connector.inprocess_http.gateway.host	ゲートウェイのホスト名または IP アドレスを指定します。Host ヘッダのないリクエストに対して welcome ファイルなどにリダイレクトするとき，Location ヘッダに指定する URL のホスト名部分が指定値となります。	なし	－	
webserver.connector.inprocess_http.gateway.port	ゲートウェイのポート番号を 1～65535 の整数で指定します。 Host ヘッダのないリクエストに対して welcome ファイルなどにリダイレクトするとき，Location ヘッダに指定する URL のポート番号部分が指定値となります。 webserver.connector.inprocess_http.gateway.host が指定されていない場合は，このプロパティの指定は無視されます。 また，webserver.connector.inprocess_http.gateway.host を指定し，このプロパティを省略した場合は下記の値が設定されます。 • webserver.connector.inprocess_http.gateway.https_scheme に true を指定している場合：443	なし	－	

キー名称	内容	省略値	VR	関連情報
	• webserver.connector.inprocess_http.gateway.https_scheme に false を指定している場合，または未指定の場合：80 数値以外の文字列，または範囲外の数値を指定した場合は，メッセージが出力され，指定されなかったものとみなされます。			
webserver.connector.inprocess_http.gateway.https_scheme	クライアントからのリクエストのスキームは https で，SSL アクセラレータなどによって Web サーバへのスキームが http となる場合に true を指定します。 true を指定した場合： Web サーバへのリクエストのスキームが https とみなされます。 false を指定した場合： 何もしません。 true または false 以外の文字列を指定した場合，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	false	−	
webserver.connector.inprocess_http.hostname_lookups	インプロセス HTTP サーバで受信したリクエストに対して，Web コンテナがホスト名のルックアップの逆引きをしてクライアントの IP アドレスをホスト名に変換するかどうかを指定します。 ただし，ホスト名の逆引きをすると，スループットが低下します。 ホスト名を解決しなかった場合，javax.servlet.ServletRequest インタフェースの getRemoteHost()メソッドの結果や，ログファイルに出力するクライアントの IP アドレスは，ドット（.）で区切られた書式の IP アドレスとなります。 true を指定した場合： IP アドレスをホスト名に変換します。 false を指定した場合： IP アドレスをホスト名に変換しません。 true または false 以外の文字列を指定した場合，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	false	−	
webserver.connector.inprocess_http.init_threads	サーバ起動時に生成するインプロセス HTTP サーバのリクエスト処理スレッド数を 1～1024 の整数で指定します。	10	−	

キー名称	内容	省略値	VR	関連情報
	指定する値は，Web クライアントとの最大接続数（webserver.connector.inprocess_http.max_connections に指定した値）以下である必要があります。これを超える値を指定した場合，メッセージが出力され，Web クライアントとの最大接続数が値として設定されます。 また，有効な最大値は実行する OS によって異なります。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。			
webserver.connector.inprocess_http.keep_start_threads	サーバ起動時に作成したスレッド数を維持するかどうかを指定します。 true を指定した場合： サーバ起動時に作成したスレッド数を維持します。プールに保持された予備スレッド数が，予備スレッド数の最大数（webserver.connector.inprocess_http.max_spare_threads に指定した値）を超えた状態でも，サーバ起動時に作成したスレッド数を下回りません。 false を指定した場合： サーバ起動時に作成したスレッド数を維持しません。予備スレッドとして保持する最大数，最小数に従って調節します。 また，サーバ起動時に作成したスレッド数が予備スレッド数の最小数（webserver.connector.inprocess_http.min_spare_threads に指定した値）よりも小さい場合，このプロパティの設定に関係なく予備スレッドの最小数に指定した値でスレッド数が維持されます。 このプロパティに false を指定した場合，サーバ起動時に作成したスレッドは予備スレッド数の最大数以下になるように調整されます。サーバ起動時に予備スレッドの最大数より大きい数のリクエスト処理スレッドを作成した場合，予備スレッドの最大数を超えたスレッドは，サーバ起動後に 1 秒間隔で一つずつ破棄されます。 true または false 以外の文字列を指定した場合，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	false	−	
webserver.connector.inprocess_http.limit.max_headers	HTTP リクエストに含まれる HTTP ヘッダの個数の上限を 0～32767 の整数で指定します。	100	−	

キー名称	内容	省略値	VR	関連情報
	上限値を設定しない場合は，0 を指定してください。 このプロパティで指定した HTTP ヘッダの個数に満たない場合でも，webserver.connector.inprocess_http.limit.max_request_header で指定したサイズを超えた場合はエラーとなります。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。			
webserver.connector.inprocess_http.limit.max_request_body	HTTP リクエストのリクエストボディの最大サイズ（単位：バイト）を-1～2147483647 の整数で指定します。上限値を設定しない場合は，-1 を指定してください。また，リクエストボディがチャンク形式で送信された場合，チャンクヘッダのサイズも指定するサイズに含める必要があります。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	-1	－	
webserver.connector.inprocess_http.limit.max_request_header	HTTP リクエストのリクエストヘッダの最大サイズ（単位：バイト）を 7～65536 の整数で指定します。 このプロパティで設定したリクエストヘッダの最大サイズに満たない場合でも，webserver.connector.inprocess_http.limit.max_headers で指定した HTTP ヘッダを超えた場合はエラーとなります。 また，HTTP ヘッダの終わりを示す改行文字(CR(0x0d)+LF(0x0a)の 2 バイト）も指定するサイズに含める必要があります。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	16384	－	
webserver.connector.inprocess_http.limit.max_request_line	リクエストラインの最大長（単位：バイト）を-1 または 7～8190 の整数で指定します。上限値を指定しない場合は，-1 を指定してください。リクエストラインは，HTTP メソッドやクエリ文字列を含む，URI および HTTP バージョンを含みます。 指定する値は，リクエストヘッダの最大サイズ（webserver.connector.inprocess_http.limit.max_request_header に指定した値）以下である必要があります。これを超える値を指定した場合は，メッセージが出力され，リクエストヘッ	8190	－	

キー名称	内容	省略値	VR	関連情報
	ダの最大サイズがリクエストラインの最大長として設定されます。 また，リクエストラインの終わりを示す改行文字（CR(0x0d)+LF(0x0a)の 2 バイト）も指定するサイズに含める必要があります。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。			
webserver.connector.inprocess_http.max_connections	Web クライアントとの最大接続数を 1～1024 の整数で指定します。有効な最大値は実行する OS によって異なります。 このパラメタに指定した値がリクエスト処理スレッドの最大値になります。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	100	－	
webserver.connector.inprocess_http.max_execute_threads	Web コンテナがリクエストを処理する同時実行数を 1～1024 の整数で指定します。 指定する値は，Web クライアントとの最大接続数（webserver.connector.inprocess_http.max_connections に指定した値）以下である必要があります。これを超える値を指定した場合は，メッセージが出力され，Web クライアントとの最大接続数が値に設定されます。 数値以外の文字列や範囲外の数値，空文字列または，空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	10	－	
webserver.connector.inprocess_http.max_spare_threads	プールに保存する予備スレッドの最大数を 1～1024 の整数で指定します。 指定する値は，Web クライアントとの最大接続数（webserver.connector.inprocess_http.max_connections に指定した値）以下である必要があります。これを超える値を指定した場合は，メッセージが出力され，Web クライアントとの最大接続数が値に設定されます。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	20	－	
webserver.connector.inprocess_http.min_spare_threads	プールに保持する予備スレッドの最小数を 1～1024 の整数で指定します。 設定する値はプールに保持する予備スレッドの最大数（webserver.connector.inprocess_http.max_	5	－	

キー名称	内容	省略値	VR	関連情報
	spare_threads に指定した値）以下である必要があります。プールに保持する予備スレッドの最大数を超える値を設定した場合は，メッセージが出力され，プールに保持する予備スレッドの最大数がプールに保持する予備スレッドの最小数として設定されます。 数値以外の文字列や範囲外の数値，空文字列，または空白文字※2 を指定した場合は，メッセージが出力されデフォルト値が設定されます。			
webserver.connector.inprocess_http.permitted.hosts	インプロセス HTTP サーバへのアクセスを許可するホストの IP アドレス（10 進表記），またはホスト名を指定します。複数指定する場合は，IP アドレスまたはホスト名の間をコンマ（,）で区切ります。アクセス制限をしない場合はアスタリスク（*）だけを指定します。 なお，ローカルホスト（"localhost"に関連づけられたアドレス※3）は明記しなくても常にアクセスが許可されます。 空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。 指定されたホスト名が解決できない場合は，メッセージが出力され，ローカルホスト（"localhost"に関連づけられたアドレス※3）からのアクセスだけが許可されます。 なお，IP アドレスまたはホスト名の前後の半角スペースは無視されます。	*	－	
webserver.connector.inprocess_http.persistent_connection.max_connections	Persistent Connection で保持する TCP コネクションの最大数を 0～1024 の整数で指定します。 設定する値は Web クライアントとの最大接続数（webserver.connector.inprocess_http.max_connections に指定した値）以下である必要があります。Web クライアントとの最大接続数を超える値を設定した場合は，メッセージが出力され，Web クライアントとの最大接続数が Persistent Connection で保持する TCP コネクションの最大数として設定されます。 数値以外の文字列や範囲外の数値を指定した場合は，メッセージが出力され，webserver.connector.inprocess_http.max_connections に指定した値がデフォルト値として設定されます。また，空文字列または空白文字※2 を指定した場合は，webserver.connector.inprocess_http.max_c	webserver.connector.inprocess_http.max_connections に指定した値	－	

キー名称	内容	省略値	VR	関連情報
	onnections に指定した値がデフォルト値として設定されます。			
webserver.connector.inprocess_http.persistent_connection.max_requests	Persistent Connection による TCP コネクションを持続したままの連続接続回数の上限を 0～2147483647 の整数で指定します。上限値を設定しない場合は，0 を指定してください。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	100	－	
webserver.connector.inprocess_http.persistent_connection.timeout	Persistent Connection で TCP コネクションを持続した状態での，リクエスト待ち時間（単位：秒）を 0～3600 の整数で指定します。0 を指定した場合，タイムアウトしません。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	3	－	
webserver.connector.inprocess_http.port	インプロセス HTTP サーバが使用するポート番号を 1～65535 の整数で指定します。すでにほかのアプリケーションで使用されているポート番号は指定できません。ほかのアプリケーションで使用されているポート番号や確保されているポート番号を指定した場合，メッセージが出力され，J2EE サーバが起動されません。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	80	－	
webserver.connector.inprocess_http.receive_timeout	Web クライアントからのリクエスト受信で，タイムアウトするまでの時間（単位：秒）を 0～3600 の整数で指定します。0 を指定した場合，タイムアウトしません。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	300	－	
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.file	リダイレクト機能で，リダイレクト時のレスポンスボディとして使用するファイルを絶対パスで指定します。パスの区切り記号には”/”を使用します。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status に 200 を指定した場合，必ずこのプロパティを指定します。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status に 200 を指定し，このプロパティが指定されていない場合，メッセージが出力され，このリダイレクト定義は無効となります。	なし	－	

キー名称	内容	省略値	VR	関連情報
	webserver.connector.inprocess_http.redirect.list で設定されていないリダイレクト定義名を使用してこのプロパティを設定した場合，空文字列または空白文字※2 を指定した場合，プロパティは無効となります。 絶対パスでない値を指定した場合は，メッセージが出力され，このリダイレクト定義は無効となります。存在しないファイルや読み取り権限のないファイルを指定した場合は，J2EE サーバ起動時にメッセージが出力され，このリダイレクト定義は無効となります。			
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.file.content_type	リダイレクト機能で，リダイレクト時のレスポンスの Content-Type ヘッダの値を指定します。 webserver.connector.inprocess_http.redirect.list で設定されていないリダイレクト定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.file を設定していない場合，プロパティは無効となります。	text/html	－	
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.redirect_url	リダイレクト機能で，リダイレクト URL を絶対 URL で指定します。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status に 200 を指定した場合，このプロパティは設定できません。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status に 200 を指定し，このプロパティを指定した場合，メッセージが出力され，リダイレクト定義は無効となります。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status に 200 以外を指定した場合，必ずこのプロパティを指定します。 webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status に 200 以外を指定し，このプロパティを指定しなかった場合，メッセージが出力され，そのリダイレクト定義は無効となります。 webserver.connector.inprocess_http.redirect.list で設定されていないリダイレクト定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 値が正しいかどうかのチェックは行われないため，実際に動作させて確認する必要があります。	なし	－	

キー名称	内容	省略値	VR	関連情報
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.request_url	リダイレクト機能で，リダイレクトを行うリクエスト URL をスラッシュ（/）で始まる絶対パスで指定します。ワイルドカード（*）はスラッシュの直後に 1 回だけ指定できます。ワイルドカードは 0 文字以上の任意の文字列を表します。"*"は必ずワイルドカードと解釈されるため，通常の文字としては使用できません。また，ほかのリダイレクト定義で指定した値と同じ値は指定できません。 webserver.connector.inprocess_http.redirect.list で指定したリダイレクト定義名について，必ずこのプロパティを指定する必要があります。指定しなかった場合，メッセージが出力され，そのリダイレクト定義は無効となります。 webserver.connector.inprocess_http.redirect.list で設定されていないリダイレクト定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 不正な値を指定した場合は，メッセージが出力され，そのリダイレクト定義は無効となります。	なし	－	
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status	リダイレクト機能で，リダイレクト時のレスポンスのステータスコード（200，300，301，302，303，305，307）を指定します。 webserver.connector.inprocess_http.redirect.list で設定されていないリダイレクト定義名を使用してこのプロパティを設定した場合，プロパティは無効となります。 不正な値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	302	－	
webserver.connector.inprocess_http.redirect.list	リダイレクト機能で使用するリダイレクト定義名を指定します。 このプロパティに指定できる値の最大長は 1024 文字です。リダイレクト定義名は，英数字（A～Z，a～z，0～9）またはアンダースコア（_）で構成される文字列で指定します。また，リダイレクト定義名一つの文字列長は 1 文字～32 文字です。 リダイレクト定義名を複数指定する場合は，コンマ（,）で区切ります。コンマの前後の空白文字※2 は無視されます。同じリダイレクト定義名は複数回指定できません。 不正な値を指定した場合は，メッセージが出力され，すべてのリダイレクト定義は無効となります。	なし	－	

キー名称	内容	省略値	VR	関連情報
webserver.connector.inprocess_http.rejection_threads	アクセスを拒否するリクエスト処理スレッドの数を 0～1023 の整数で指定します。指定する値は，リクエスト処理スレッドの最大数（webserver.connector.inprocess_http.max_connections に指定した値）よりも小さくする必要があります。Web クライアントとの最大接続数以上の値を設定した場合は，メッセージが出力され，Web クライアントとの最大接続数よりも 1 小さい値がアクセスを拒否するリクエスト処理スレッド数として設定されます。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	1	－	
webserver.connector.inprocess_http.response.header.server	レスポンスに自動的に付加する Server ヘッダの値を指定します。 空文字列または空白文字※2 を指定した場合，メッセージが出力され，デフォルト値が設定されます。	CosminexusComponentContainer	－	
webserver.connector.inprocess_http.send_timeout	Web クライアントへのレスポンス送信で，タイムアウトするまでの時間（単位：秒）を 0～3600 の整数で指定します。0 を指定した場合，タイムアウトは有効になりません。 数値以外の文字列や範囲外の数値，空文字列または空白文字※2 を指定した場合は，メッセージが出力され，デフォルト値が設定されます。	300	－	
webserver.connector.limit.max_fileupload_count	ファイルアップロードを実施するリクエストに含まれるアップロードファイル数の上限値を-1～2147483647 の整数で指定します。 上限値を設定しない場合は，-1 を指定します。 数値以外の文字列や範囲外の数値，空文字列または空白文字を指定した場合は，メッセージが出力され，デフォルト値が設定されます。 ファイル数が上限値を超えたことを検知すると，ファイルシステムへのファイル作成を実施する前にエラーとし KDJE39606-E を出力します。 アップロードファイル数が指定した上限値を超える場合，次の API 呼び出し時に IllegalStateException をスローします。 • javax.servlet.http.HttpServletRequest.getPart(String) • javax.servlet.http.HttpServletRequest.getParts() また，リクエストのアップロードファイル数が指定した上限値を超える場合，次の API を実行	10000	11-00-10	

キー名称	内容	省略値	VR	関連情報
	してもマルチパートで送信されたリクエストパラメタは取得できません。 • javax.servlet.ServletRequest.getParameter(String) • javax.servlet.ServletRequest.getParameterMap() • javax.servlet.ServletRequest.getParameterNames() • javax.servlet.ServletRequest.getParameterValues(String)			

（凡例）

－：08-00 より前のバージョンを示します。

注※1

- 有効な最大値は実行するプラットフォームに依存します。
- Web サーバに到着するリクエストの一部が Web コンテナに転送されるため，Web サーバの最大同時接続数は，URL グループ単位，Web アプリケーション単位およびデフォルトの実行待ちキューサイズの総和＋ Web コンテナ単位の最大同時実行スレッド数より大きく設定する必要があります。

  また，データベース操作をするサーブレットや JSP については，データベースコネクションの数よりも多くの多重度は得られないため，Web コンテナの同時実行数を増やす場合は，利用できるデータベースコネクションの数も増やす必要があります。

  性能のチューニング時には，次に示す関係を常に考慮して，各パラメタの値を調整してください。

  〈Webサーバの最大同時接続数〉 ＞ 〈URLグループ単位，Webアプリケーション単位およびデフォルトの実行待ちキューサイズの総和〉＋〈Webコンテナ単位の最大同時実行スレッド数〉

  〈Webコンテナ単位の最大同時実行スレッド数〉 ≧ 〈データベースコネクションの数〉

  Web コンテナ単位での同時実行スレッド数の制御については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.14　Web コンテナ単位での同時実行スレッド数の制御」を参照してください。

  Web サーバでの処理の同時接続数については Web サーバのマニュアルを参照してください。

注※2

空白文字とは，半角スペース，タブ，LF（0x0a），CR（0x0d）または FF（0x0c）のことを指します。

注※3

J2EE サーバ起動時に"localhost"に関連づけられたアドレスになります。

(6)　webserver.container から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

「関連情報」とは，指定したキーに関する情報の参照先です。マニュアル名称の「アプリケーションサーバ」を省略しています。

キー名称	内容	省略値	VR	関連情報
webserver.container.ac.logEnabled	Web コンテナの保守用のトレースログを出力するかどうかを指定します。 true を指定した場合： トレースログを出力します。 false を指定した場合： トレースログを出力しません。	false	－	

（凡例）

－：08-00 より前のバージョンを示します。

10.1.2 server.policy（J2EE サーバ用セキュリティポリシーファイル）

(1) 形式

J2SE のセキュリティポリシーファイル形式に従います。

(2) ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥server¥usrconf¥ejb¥<サーバ名称>¥
- UNIX の場合
  /opt/Cosminexus/CC/server/usrconf/ejb/<サーバ名称>/

(3) 機能

J2EE サーバを実行する JavaVM のセキュリティポリシーを指定します。

J2EE サーバの稼働中に，このファイルの内容を変更した場合，変更した内容は次に J2EE サーバを起動したときに反映されます。

(4) 記述例

使用されるポリシーファイルの内容を次に示します。

```
// (1)
// Grant all permissions to anything loaded from the
// EJB server itself

grant codeBase "file:${ejbserver.install.root}/lib/*" {
permission java.security.AllPermission;
```

```
};
grant codeBase "file:${tpbroker.java.home}/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/DABJ/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/manager/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/c4web/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/c4web/exlib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/jaxws/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/jaxrs/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/jaxp/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/CTM/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/PRF/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/wss/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/XMLSEC/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${ejbserver.install.root}/sfo/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${hntrlib.home}/classes/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/common/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${ejbserver.install.root}/weld/lib/*" {
permission java.security.AllPermission;
};

// (2)
// Grant all permissions to the container generated stubs and
// implementation classes
grant codeBase "file:${ejbserver.http.root}/ejb/${ejbserver.serverName}/containers/-" {
permission java.security.AllPermission;
};

// (3)
```

```
// Grant all permissions to imported resource (datasource) implementations
// implementation classes
grant codeBase "http://*/ejb/${ejbserver.serverName}/import/resjars/-" {
permission java.security.AllPermission;
};

// (4)
// Grant permissions to resource adapters
//
grant codeBase "file:${ejbserver.http.root}/ejb/${ejbserver.serverName}/rarjars/-" {

// For uCosminexus TP1 Connector & TP1/Client/J
permission java.util.PropertyPermission "*", "read, write";

// For uCosminexus TP1 Connector & TP1/Client/J & Cosminexus Reliable Messaging
permission java.io.FilePermission "<<ALL FILES>>", "read, write, delete";
permission java.net.SocketPermission "*", "connect,listen,accept";

// For TP1/Message Queue - Access
permission java.lang.RuntimePermission "loadLibrary.*";

// For TP1/Message Queue - Access & Cosminexus Reliable Messaging
permission java.lang.RuntimePermission "modifyThreadGroup";
permission java.lang.RuntimePermission "modifyThread";

// For DB Connector
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// For authentication (from J2EE RI server.policy file)
permission javax.security.auth.PrivateCredentialPermission "* * ¥"*¥"", "read";

// For Cosminexus Reliable Messaging
permission javax.security.auth.AuthPermission "modifyPrivateCredentials";
permission java.lang.RuntimePermission "getenv.HRMDIR";

// For Cosminexus SOA FTP Inbound Adapter
permission java.lang.RuntimePermission "getClassLoader";
permission java.lang.RuntimePermission "setContextClassLoader";
permission java.lang.RuntimePermission "accessDeclaredMembers";
};

// (5)
// Grant permissions to JSP/Servlet
//
grant codeBase "file:${ejbserver.http.root}/web/${ejbserver.serverName}/-" {
permission java.lang.RuntimePermission "loadLibrary.*";
permission java.lang.RuntimePermission "queuePrintJob";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
permission java.net.SocketPermission "*", "connect";
permission java.io.FilePermission "<<ALL FILES>>", "read, write";
permission java.util.PropertyPermission "*", "read";
permission javax.security.auth.AuthPermission "getSubject";
permission javax.security.auth.AuthPermission "createLoginContext.*";
};

// (6)
// Grant permissions to Cosminexus Service Coordinator
```

```
//
grant codeBase "file:${cosminexus.home}/CSC/lib/*" {
permission java.security.AllPermission;
};

// (7)
// Grant permissions to custom login modules
//
grant codeBase "file:${cosminexus.home}/manager/modules/-" {
permission java.io.FilePermission "<<ALL FILES>>", "read";
permission javax.security.auth.AuthPermission "modifyPrincipals";
permission javax.security.auth.AuthPermission "modifyPublicCredentials";
};

// (8)
// Grant minimal permissions to everything else:
//  EJBs
//  client implementation classes
grant  {
permission java.util.PropertyPermission "*", "read";
permission java.lang.RuntimePermission "queuePrintJob";
permission java.net.SocketPermission "*", "connect";
};
```

記述例の（1）～（8）について説明します。

(1)

J2EE サーバが使用するクラスファイルに対して次の権限を許可します。

- すべてのアクセス権を許可

(2)

J2EE サーバが生成するスタブとスケルトンなどのクラスファイルに対して次の権限を許可します。

- すべてのアクセス権を許可

(3)

J2EE サーバが使用するリソースのクラスファイルに対して次の権限を許可します。

- すべてのアクセス権を許可

(4)

J2EE サーバが使用するリソースアダプタのクラスファイルに対して次の権限を許可します。

- すべてのプロパティ情報に対して読み取り，および書き込みを許可
- すべてのファイルに対して読み取り，書き込み，および削除を許可
- すべてのソケット通信に対してネットワークへの接続，接続での待機，および接続の受け付けを許可
- すべてのライブラリのロードを許可
- スレッドグループの変更を許可
- スレッドの変更を許可
- すべてのリフレクション操作を許可

- 任意の Subject が所有する，すべての非公開 Credential へのアクセスを許可
- Subject に関連づけられた非公開 Credential の Set の変更を許可
- 環境変数 HRMDIR の値の取得を許可
- クラスローダの取得を許可
- コンテキストクラスローダの設定を許可
- クラスの宣言されたメンバへのアクセスを許可

注意事項

- リソースアダプタ内の JAR ファイルの展開先である J2EE サーバ管理下のディレクトリが記載されています。
- J2EE サーバで動作するすべてのリソースアダプタが有効範囲です。

(5)

JSP／サーブレットのクラスファイルに対して次の権限を許可します。

- すべてのライブラリのロードを許可
- 印刷ジョブ要求を許可
- スレッドの変更を許可
- スレッドグループの変更を許可
- すべてのソケット通信に対してネットワーク接続を許可
- すべてのファイルに対して読み取り，および書き込みを許可
- すべてのプロパティ情報の読み取りを許可
- Subject の参照を許可
- あらゆる名称で LoginContext クラスのインスタンス化を許可

注意事項

次の定義は，07-00 より前の環境で構築したサーバには記載されていません。Web アプリケーションでユーザスレッドを作成する必要がある場合，追加してください。

- permission java.lang.RuntimePermission "modifyThread";
- permission java.lang.RuntimePermission "modifyThreadGroup";

(6)

Service Coordinator のクラスファイルに対し次の権限を許可します。

- すべてのアクセス権を許可

(7)

統合ユーザ管理のカスタムログインモジュールに対し次の権限を許可します。

- すべてのファイルに対する読み込みを許可
- Subject に Principal および Credential の追加を許可

(8)

すべてのクラスファイルに対して次の権限を許可します。

- すべてのプロパティ情報の読み取りを許可
- 印刷ジョブ要求を許可
- すべてのソケット通信に対してネットワーク接続を許可

(5) 注意事項

- 構文が不正または適切なアクセス権限が設定されていない server.policy ファイルを使用した場合，java.lang.StackOverflowError または java.lang.OutOfMemoryError が発生して J2EE サーバが異常終了することがあります。
- サーバをセットアップするときに生成された server.policy ファイルには，J2EE サーバを動作させるために最低限必要な権限が記述されています。生成された server.policy ファイルの記述行の削除および変更はしないでください。

11 Smart Composer 機能で使用するファイル

この章では，Smart Composer 機能で使用するファイルの形式，格納先，機能，指定できるキーなどについて説明します。ここでは，推奨モードと異なる内容だけ説明します。

11.1　論理 Web サーバで指定できるパラメタ

ここでは，論理 Web サーバで指定できるパラメタについて説明します。

11.1.1　HTTP Server 用リダイレクタ動作定義を設定するパラメタ

HTTP Server 用リダイレクタ動作定義を設定するパラメタについて，次の表に示します。

「省略値」とは，パラメタの指定がない場合に仮定される値です。「VR」とは，パラメタが導入・変更されたアプリケーションサーバのバージョンです。なお，「param-name 指定値」に対応する param-value の指定内容については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

注

AllText パラメタを指定する場合，HTTP Server 用リダイレクタ動作定義を設定するパラメタの指定を有効にするには，設定ファイルの内容に次の記述を含めてください。

```
Include ”<Application Serverのインストールディレクトリ>/CC/web/redirector/servers/<論理Webサーバ>/mod_jk.conf”
```

表 11-1　HTTP Server 用リダイレクタ動作定義を設定するパラメタ

param-name 指定値	指定可能値	省略値	VR
JkConnectTimeout	0～3600 の整数で指定します。	30	07-00
JkGatewayHost	次のどれかを指定します。 • IPv4 アドレス • ホスト名 なお，ホスト名には，英数字，アンダースコア「_」，ピリオド「.」，およびハイフン「-」で指定した 255 文字以内の文字列を指定してください。	なし	07-50
JkGatewayHttpsScheme	次のどちらかを指定します。 • On • Off このパラメタについては，「5.10 Web コンテナへのゲートウェイ情報の通知」を参照してください。	Off	06-50
JkGatewayPort	1～65535 の整数で指定します。	なし	07-50
JkLogFileDir	任意の文字列を指定します。	logs	06-50
JkLogFileNum	1～64 の整数で指定します。	8	06-50 07-50

param-name 指定値	指定可能値	省略値	VR
JkLogFileSize	4096～2147483647 の整数で指定します。	4194304	06-50 07-50
JkLogLevel	次のどれかを指定します。 • emerg • error • info • debug	error	06-50
JkModulePriority	次のどれかを指定します。 • REALLY_FIRST（整数値の-10 に対応） • FIRST（整数値の 0 に対応） • MIDDLE（整数値の 10 に対応） • LAST（整数値の 20 に対応） • REALLY_LAST（整数値の 30 に対応） • -10～30 の整数	FIRST	07-00
JkMount	次の形式で指定します。 <URL パターン> <ワーカ名> <ワーカ名>にはワーカの定義で worker.list に指定した名称を使用してください。ワーカの定義と worker.list については「11.1.2 ワーカ定義を設定するパラメタ」を参照してください。 注 同一の J2EE サーバを複数のワーカ名で定義しないでください。そのようなワーカを JkMount に指定した場合の動作は保証されません。	combined-tier と http-tier の場合※ /* [J2EE サーバ名] 注 同時に［J2EE サーバ名］が worker（タイプ：ajp13）として設定されます。	07-50
JkPrfId	英数字を 31 文字以内で指定します。 「TSC」,「tsc」または「CTM」や「ctm」で始まる文字列を指定した場合，エラーになります。	なし free-tier 以外の場合，値は自動で設定されます。	07-50
JkRequestRetryCount	1～16 の整数で指定します。	3	07-00
JkSendTimeout	0～3600 の整数で指定します。	100	07-00
JkTraceLog	次のどちらかを指定します。 • On • Off	On	06-50
JkTraceLogFileDir	任意の文字列を指定します。	logs	06-50
JkTraceLogFileNum	1～64 の整数で指定します。	4	06-50

param-name 指定値	指定可能値	省略値	VR
			07-50
JkTraceLogFileSize	4096～2147483647 の整数で指定します。	16777216	06-50 07-50
JkTranslateBackcompat	次のどちらかを指定します。 • On • Off	Off	06-50

注

キーを複数指定した場合，最後に指定した値が有効になります。

注※

なお，combined-tier と http-tier でこのパラメタを省略した場合，次のように関連するパラメタに値が設定されます。

```
<param>
<param-name>JkMount</param-name>
<param-value>/* [J2EEサーバ名]</param-value>
</param>
<param>
<param-name>worker.list</param-name>
<param-value>[J2EEサーバ名]</param-value>
</param>
<param>
<param-name>worker.[J2EEサーバ名].host</param-name>
<param-value>[ホスト名]</param-value>
</param>
<param>
<param-name>worker.[J2EEサーバ名].port</param-name>
<param-value>[ポート番号]</param-value>
</param>
<param>
<param-name>worker.[J2EEサーバ名].type</param-name>
<param-value>ajp13</param-value>
</param>
```

11.1.2　ワーカ定義を設定するパラメタ

ワーカ定義を設定するパラメタについて，次の表に示します。

「省略値」とは，パラメタの指定がない場合に仮定される値です。「VR」とは，パラメタが導入・変更されたアプリケーションサーバのバージョンです。なお，「param-name 指定値」に対応する param-value の指定内容については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。

注

AllText パラメタを指定する場合，ワーカ定義を設定するパラメタの指定を有効にするには，設定ファイルの内容に次の記述を含めてください。

```
Include ”<Application Serverのインストールディレクトリ>/CC/web/redirector/servers/<論理Webサーバ>/mod_jk.conf”
```

(1) ワーカ定義ファイルに指定できるキー

ワーカ，およびワーカごとの各パラメタを定義します。このキーに不正な値を設定した場合，動作は保証されません。

param-name 指定値	指定可能値	省略値	VR
worker.list	任意の文字列を指定します。	なし	07-50
worker.<ワーカ名>.<パラメタ>	定義パラメタについては，「(2) ワーカごとの定義パラメタ」を参照してください。	なし	07-50

(2) ワーカごとの定義パラメタ

param-name 指定値	指定可能値	省略値	VR
worker.<ワーカ名>.balanced_workers	任意の文字列を指定します。	なし	07-50
worker.<ワーカ名>.cachesize※1	1～2147483647 の整数で指定します。	64	07-50
worker.<ワーカ名>.default_worker	ワーカ名をコンマ「,」で区切って指定します。 先頭と末尾のスペースは無視されます。	なし	07-50
worker.<ワーカ名>.delegate_error_code	400～417，422～424，500～505，507，510 をコンマ「,」で区切って指定します。	なし	07-50
worker.<ワーカ名>.host	指定できる値を次に示します。 • IPv4 アドレス※2 • ホスト名※2 • @myhost	なし	07-50
worker.<ワーカ名>.lbfactor※1	0～9999999999 の整数で指定します。	1	07-50
worker.<ワーカ名>.port	1～65535 の整数で指定します。	なし	07-50
worker.<ワーカ名>.post_data	数字（0～9）の 10 文字以内の文字列と m，M，k，K が 0～1 回連続する文字列を指定します。	なし	07-50
worker.<ワーカ名>.post_size_workers	任意の文字列を指定します。	なし	07-50
worker.<ワーカ名>.receive_timeout※1	0～3600 の整数で指定します。	3600	07-50
worker.<ワーカ名>.type	次のどれかを指定します。 • ajp13 • lb	なし	07-50

param-name 指定値	指定可能値	省略値	VR
	• post_size_lb		

注

ワーカを新規に定義する場合は，次のパラメタを必ず定義してください。

JkMount については「11.1.1　HTTP Server 用リダイレクタ動作定義を設定するパラメタ」を参照してください。

・worker.list

・worker.＜ワーカ名＞.host

・worker.＜ワーカ名＞.port

・worker.＜ワーカ名＞.type

・JkMount

注※1

combined-tier または http-tier で構築した場合，省略値は J2EE サーバ名で定義されたワーカ名のワーカに適用されます。省略値を変更したい場合は，上記の「注」に示すワーカを定義してください。

注※2

IPv4 アドレスまたはホスト名を指定する場合は，ホストの定義の<host-name>タグの値を指定してください。異なる値を指定した場合，警告メッセージ（KEOS24195-W）が出力され，意図しない設定となることがあります。

11.2　論理 J2EE サーバで指定できるパラメタ

ここでは，論理 J2EE サーバで指定できるパラメタについて説明します。

指定できるパラメタと詳細についての参照先を次の表に示します。

表 11–2　使用するサーバと指定するパラメタの参照先の対応

使用するサーバ	指定するパラメタ
J2EE サーバ	J2EE サーバ用ユーザプロパティを設定するパラメタ（11.2.1 参照）

11.2.1　J2EE サーバ用ユーザプロパティを設定するパラメタ

J2EE サーバ用ユーザプロパティを設定するパラメタについて説明します。

「param-name 指定値」に対応する param-value の指定内容については，「10.1.1　usrconf.properties (J2EE サーバ用ユーザプロパティファイル)」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。「省略値」とは，パラメタの指定がない場合に仮定される値です。「VR」とは，パラメタが導入・変更されたアプリケーションサーバのバージョンです。

なお，「param-value の指定内容」が記載されているパラメタは，usrconf.properties（J2EE サーバ用ユーザプロパティファイル）と指定方法が異なるパラメタです。

J2EE サーバ用ユーザプロパティを設定するパラメタを指定する場合の指定形式を次に示します。

指定形式

```
<param-name>パラメタ</param-name>
<param-value>値</param-value>
```

usrconf.properties（J2EE サーバ用ユーザプロパティファイル）に指定できるパラメタのうち，この項の表に記載していないパラメタは，次の形式で指定してください。

指定形式

```
<param-name>ex.properties</param-name>
<param-value>パラメタ=値</param-value>
```

値を複数指定する場合は，複数の<param-value>を指定します。

値を複数指定する場合の指定形式

```
<param-name>ex.properties</param-name>
<param-value>パラメタ=値</param-value>
<param-value>パラメタ=値</param-value>
```

なお，この項に記載があるパラメタのうち，次のパラメタは，この形式で指定することもできます。それ以外のパラメタをこの形式で指定した場合，動作が保証されません。また，<param-name>パラメタ</param-name>の形式とこの形式の両方で同じパラメタを指定した場合，動作が保証されません。

- 運用管理ポータルの画面に対応しないパラメタ
- 運用管理ポータルの画面に対応し，画面名が「システムプロパティの設定」，設定個所が「システムプロパティ」であるパラメタ

パラメタと運用管理ポータルの画面の対応については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」を参照してください。

(1) cosminexus.jpa から始まるパラメタ

cosminexus.jpa から始まるパラメタについて次の表に示します。「param-name 指定値」に対応する param-value の指定内容については，「10.1.1(1) cosminexus.jpa から始まるキー」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。

表 11–3 cosminexus.jpa から始まるパラメタ

param-name 指定値	指定可能値	省略値	VR
cosminexus.jpa.logging.level.operation.<category>	指定できる文字列を次に示します。 • Off • Information • Detail	Off	08-00
cosminexus.jpa.exception.logging.sql	指定できる文字列を次に示します。 • Off • Information • Detail	Off	08-00

(2) ejbserver.jpa から始まるパラメタ

ejbserver.jpa から始まるパラメタについて次の表に示します。「param-name 指定値」に対応する param-value の指定内容については，「10.1.1(2) ejbserver.jpa から始まるキー」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。

表 11–4 ejbserver.jpa から始まるパラメタ

param-name 指定値	指定可能値	省略値	VR
ejbserver.jpa.defaultJtaDsName	任意の文字列を指定します。	なし	08-00
ejbserver.jpa.defaultNonJtaDsName	任意の文字列を指定します。	なし	08-00
ejbserver.jpa.defaultProviderClassName	任意の文字列を指定します。	com.hitachi.software.jpa.Persiste	08-00

param-name 指定値	指定可能値	省略値	VR
		nceProvider	
ejbserver.jpa.disable	指定できる文字列を次に示します。 • true • false	false	08-50
ejbserver.jpa.overrideJtaDsName	任意の文字列を指定します。	なし	08-00
ejbserver.jpa.overrideNonJtaDsName	任意の文字列を指定します。	なし	08-00
ejbserver.jpa.overrideProvider	任意の文字列を指定します。	なし	08-00
ejbserver.jpa.emfprop.<property key>	任意の文字列を指定します。	なし	08-00

(3) ejbserver.logger から始まるパラメタ

ejbserver.logger から始まるパラメタについて次の表に示します。「param-name 指定値」に対応する param-value の指定内容については，「10.1.1(3) ejbserver.logger から始まるキー」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。

表 11-5 ejbserver.logger から始まるパラメタ

param-name 指定値	指定可能値	省略値	VR
ejbserver.logger.channels.define.JPAOperationLogFile.filenum	1～16 の整数で指定します。	2	08-00
ejbserver.logger.channels.define.JPAOperationLogFile.filesize	4096～2147483647（単位：バイト）の整数で指定します。	1048576	08-00
ejbserver.logger.channels.define.JPAMaintenanceLogFile.filenum	1～16 の整数で指定します。	2	08-00
ejbserver.logger.channels.define.JPAMaintenanceLogFile.filesize	4096～2147483647（単位：バイト）の整数で指定します。	1048576	08-00
ejbserver.logger.channels.define.WebAccessLogFile.filenum	1～16 の整数で指定します。	16	07-50
ejbserver.logger.channels.define.WebAccessLogFile.filesize	4096～2147483647（単位：バイト）の整数で指定します。	4194304	07-50

(4) ejbserver.server から始まるパラメタ

ejbserver.server から始まるパラメタについて次の表に示します。「param-name 指定値」に対応する param-value の指定内容については，「10.1.1(4) ejbserver.server から始まるキー」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。

表 11-6　ejbserver.server から始まるパラメタ

param-name 指定値	指定可能値	省略値	VR
ejbserver.server.eheap.ajp13.enabled	指定できる文字列を次に示します。 • true • false	true	08-00

(5)　webserver.connector から始まるパラメタ

webserver.connector から始まるパラメタについて次の表に示します。「param-name 指定値」に対応する param-value の指定内容については，「10.1.1(5)　webserver.connector から始まるキー」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。

表 11-7　webserver.connector から始まるパラメタ

param-name 指定値	指定可能値	省略値	VR
webserver.connector.ajp13.backlog	1～2147483647 の整数で指定します。	100	06-50
webserver.connector.ajp13.bind_host	指定できる文字列を次に示します。 • ホスト名※ • IPv4 アドレス※ • @myhost	なし	07-50
webserver.connector.ajp13.max_threads	1～1024 の整数で指定します。	10	06-50
webserver.connector.ajp13.port	1～65535 の整数で指定します。	8007	06-50 07-00
webserver.connector.ajp13.receive_timeout	0～3600（単位：秒）の整数で指定します。	600	06-50 07-00
webserver.connector.ajp13.send_timeout	0～3600（単位：秒）の整数で指定します。	600	07-00
webserver.connector.inprocess_http.backlog	1～2147483647 の整数で指定します。	511	07-50
webserver.connector.inprocess_http.bind_host	指定できる文字列を次に示します。 • ホスト名※ • IPv4 アドレス※ • @myhost	なし	07-50
webserver.connector.inprocess_http.enabled	指定できる文字列を次に示します。 • true • false	false	07-50
webserver.connector.inprocess_http.enabled_methods	指定できる値を次に示します。 • GET	GET,HEAD,POST,P	07-50

param-name 指定値	指定可能値	省略値	VR
	• HEAD • POST • PUT • DELETE • OPTIONS • TRACE • CONNECT • PATCH • LINK • UNLINK • アスタリスク「*」	UT,DELETE,OPTIONS	
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file	ファイル名	なし	07-50
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file.content_type	任意の文字列を指定します。	text/html	07-50
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.redirect_url	任意の文字列を指定します。	なし	07-50
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.request_url	任意の文字列を指定します。	/*	07-50
webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.status	400～599 の整数で指定します。	なし	07-50
webserver.connector.inprocess_http.error_custom.list	英数字，およびアンダースコア「_」を使って 32 文字以内で指定します。また，複数指定する場合はコンマ「,」で区切って指定します。	なし	07-50
webserver.connector.inprocess_http.gateway.host	次のどれかを指定します。 • ホスト名 • IPv4 アドレス • @myhost	なし	07-50
webserver.connector.inprocess_http.gateway.https_scheme	指定できる文字列を次に示します。 • true • false	false	07-50
webserver.connector.inprocess_http.gateway.port	1～65535 の整数で指定します。	なし	07-50
webserver.connector.inprocess_http.init_threads	1～1024 の整数で指定します。	10	07-50

param-name 指定値	指定可能値	省略値	VR
webserver.connector.inprocess_http.keep_start_threads	指定できる文字列を次に示します。 • true • false	false	07-50
webserver.connector.inprocess_http.limit.max_headers	0～32767 の整数で指定します。	100	07-50
webserver.connector.inprocess_http.limit.max_request_body	-1～2147483647（単位：バイト）の整数で指定します。	-1	07-50
webserver.connector.inprocess_http.limit.max_request_header	7～65536（単位：バイト）の整数で指定します。	16384	07-50
webserver.connector.inprocess_http.limit.max_request_line	指定できる値（単位：バイト）を示します。 • -1 • 7～8190	8190	07-50
webserver.connector.inprocess_http.max_connections	1～1024 の整数で指定します。	100	07-50
webserver.connector.inprocess_http.max_execute_threads	1～1024 の整数で指定します。	10	07-50
webserver.connector.inprocess_http.max_spare_threads	1～1024 の整数で指定します。	20	07-50
webserver.connector.inprocess_http.min_spare_threads	1～1024 の整数で指定します。	5	07-50
webserver.connector.inprocess_http.permitted.hosts	次のどれかを指定します。 • ホスト名 • IPv4 アドレス • @myhost • アスタリスク「*」	*	07-50
webserver.connector.inprocess_http.persistent_connection.max_connections	0～1024 の整数で指定します。	100	07-50
webserver.connector.inprocess_http.persistent_connection.max_requests	0～2147483647 の整数で指定します。	100	07-50
webserver.connector.inprocess_http.persistent_connection.timeout	0～3600（単位：秒）の整数で指定します。	3	07-50
webserver.connector.inprocess_http.port	1～65535 の整数で指定します。	80	07-50
webserver.connector.inprocess_http.receive_timeout	0～3600（単位：秒）の整数で指定します。	300	07-50
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.file	ファイル名を指定します。	なし	07-50

param-name 指定値	指定可能値	省略値	VR
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.file.content_type	任意の文字列を指定します。	text/html	07-50
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.redirect_url	任意の文字列を指定します。	なし	07-50
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.request_url	任意の文字列を指定します。	なし	07-50
webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.status	指定できる値を次に示します。 • 200 • 300 • 301 • 302 • 303 • 305 • 307	302	07-50
webserver.connector.inprocess_http.redirect.list	英数字，およびアンダースコア「_」を使って 32 文字以内で指定します。また，複数指定する場合はコンマ「,」で区切って指定します。	なし	07-50
webserver.connector.inprocess_http.rejection_threads	0～1023 の整数で指定します。	1	07-50
webserver.connector.inprocess_http.response.header.server	任意の文字列を指定します。	CosminexusComponentContainer	07-50
webserver.connector.inprocess_http.send_timeout	0～3600（単位：秒）の整数で指定します。	300	07-50

注※

ホスト名または IPv4 アドレスを指定する場合は，ホストの定義の<host-name>タグの値を指定してください。異なる値を指定した場合，警告メッセージ（KEOS24186-W）が出力され，意図しない設定となることがあります。

(6)　webserver.container から始まるパラメタ

webserver.container から始まるパラメタについて次の表に示します。「param-name 指定値」に対応する param-value の指定内容については，「10.1.1(6)　webserver.container から始まるキー」についての説明を参照してください。また，参照する場合はキーをパラメタに読み替えてください。

表 11–8　webserver.container から始まるパラメタ

param-name 指定値	指定可能値	省略値	VR
webserver.container.ac.logEnabled	指定できる文字列を次に示します。	false	06-50

param-name 指定値	指定可能値	省略値	VR
	• true • false		

12 JPA で使用するファイル

この章では，CJPA プロバイダで使用するファイルの形式，格納先，機能，指定できるキーなどについて説明します。

12.1　CJPA プロバイダで使用するファイルの一覧

CJPA プロバイダで使用するファイルの一覧を，次の表に示します。

表 12–1　CJPA プロバイダで使用するファイルの一覧

ファイル名	概要	参照先
persistence.xml	アプリケーションサーバの JPA 機能の永続化ユニット情報を設定するためのファイルです。	12.2
O/R マッピングファイル	O/R マッピング情報を設定するためのファイルです。	12.3

12.2 persistence.xml

persistence.xml の構成を次に示します。

タグ名	出現パターン	説明
<persistence>	1 回	ルートタグを表します。
└ <persistence-unit>	0 回以上	永続化ユニットの定義をします。
├ <description>	0 または 1 回	永続化ユニットに関する説明を記述します。
├ <provider>	0 または 1 回	javax.persistence.spi.PersistenceProvider の実装クラス名を指定します。
├ <jta-data-source>	0 または 1 回	JTA トランザクションに対応したデータソースの参照を指定します。
├ <non-jta-data-source>	0 または 1 回	JTA トランザクションには対応していないデータソースの参照を指定します。
├ <mapping-file>	0 回以上	O/R マッピングファイルを指定します。
├ <jar-file>	0 回以上	entity クラス，embeddable クラス，および mappedsuper クラスを含む JAR ファイル名を記述します。
├ <class>	0 回以上	entity クラス，embeddable クラス，および mappedsuper クラスを記述します。
├ <exclude-unlisted-classes>	0 または 1 回	Persistence クラスを指定します。
└ <properties>	0 または 1 回	CJPA プロバイダ独自のプロパティを定義します。
└ <property>	0 回以上	各種のプロパティを定義します。

それぞれのタグの詳細については，「12.2.1 persistence.xml の詳細」を参照してください。

12.2.1 persistence.xml の詳細

(1) <persistence>

永続化ユニットの定義を開始することを示すルートタグです。

<persistence>タグには，XML 名前空間を指定する xmlns 要素を設定する必要があります。

指定できる属性を次の表に示します。

表 12–2　<persistence>の属性

属性名	型	任意／必須	説明
version	persistence:versionType	必須	XML スキーマのバージョン"1.0"を指定します。

(2)　<persistence-unit>

永続化ユニットを定義します。指定できる属性を次の表に示します。

表 12–3　<persistence-unit>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	永続化ユニットの名前を指定します。 name 属性で指定する名前は，永続化ユニットがパッケージングされる範囲内で一意な名前である必要があります。 重複した名前の永続化ユニットが定義された場合は，CJPA プロバイダでは動作は保証しません。 Java EE 環境で同名の永続化ユニットを定義した場合，コンテナ側で警告メッセージを出力します。 Java EE 環境では，name 属性に指定する値は，空文字であってはなりません。空文字を指定した場合には，コンテナ側で例外が発生します。
transaction-type	persistence:persistence-unit-transaction-type	任意	EntityManager が使用するトランザクションを指定します。 JTA 　JTA トランザクションを使用します。 RESOURCE_LOCAL 　JTA トランザクションを使用しないで，独自にトランザクションを管理します。 transaction-type 属性の値を指定しなかった場合は，デフォルトである「JTA」が適用されます。 Java EE 環境で指定できる属性の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「5.8.1　<persistence-unit>タグに指定する属性」を参照してください。

(3)　<description>

永続化ユニットに関する説明を記述します。

(4)　<provider>

javax.persistence.spi.PersistenceProvider の実装クラス名を指定します。

CJPA プロバイダを使用する場合，com.hitachi.software.jpa.PersistenceProvider を指定します。ほかの JPA プロバイダが存在する場合など，明示的に指定する場合に記述します。なお，値を指定しない場合は，コンテナの挙動に依存します。

注意事項

<provider>タグの途中に空白が入ると，要素を指定していない場合と同じ処理をします。

(5) <jta-data-source>

JTA トランザクションに対応したデータソースの参照を指定します。

<persistence-unit>タグの transaction-type に指定した値が JTA の場合に指定します。なお，値を指定しない場合は，コンテナの挙動に依存します。

(6) <non-jta-data-source>

JTA トランザクションには対応していないデータソースの参照を指定します。<persistence-unit>タグの transaction-type に指定した値が RESOURCE_LOCAL の場合に指定します。

transaction-type に指定した値が JTA の場合，CJPA プロバイダでは<non-jta-data-source>タグに値が指定されても無視します。値を指定しない場合は，コンテナの挙動に依存します。

(7) <mapping-file>

O/R マッピングファイルを指定します。

指定したファイルはクラスパスで指定している場所に格納されている必要があります。O/R マッピングファイルを使用しない場合，または，orm.xml を所定の位置に配置して使用する場合は，記述する必要はありません。

指定したファイルが見つからない場合，アプリケーション開始に失敗します。

(8) <jar-file>

entity クラス，embeddable クラス，mappedsuper クラスを含む JAR ファイル名を記述します。JAR ファイルのパスは，永続化ユニットのルートから相対パスで指定します。

指定したファイルが見つからない場合，コンテナの挙動に依存します。

(9) <class>

entity クラス，embeddable クラス，および mappedsuper クラスを記述します。

指定したクラスが見つからない場合，コンテナの挙動に依存します。

なお，<class>タグに指定された値が entity クラス，embeddable クラス，および mappedsuper クラスであるかどうかのチェックは CJPA プロバイダでは実施しません。そのため，entity クラス，embeddable クラス，および mappedsuper クラス以外のクラスを指定した場合に例外は発生しないで，動作します。

(10) <exclude-unlisted-classes>

Persistence クラスを定義します。

指定できる値と，値を指定した場合の挙動を次に示します。

true

class 要素，jar-file 要素，および mapping-file 要素によって明示的に指定されたクラスだけ Persistence クラスとして扱います。

false

exclude-unlisted-class 要素を指定していない場合，永続化ユニットのルート以下の class ファイルに対して JPA 対象のクラスであるかどうかを検索します。

(11) <properties>

CJPA プロバイダ独自のプロパティを定義します。この要素以下に property 要素を指定してプロパティを定義します。

(12) <property>

各種のプロパティを定義します。

プロパティの詳細については，「12.2.2 <property>タグに指定できる CJPA プロバイダ独自のプロパティ」を参照してください。

指定できる属性を次の表に示します。

表 12-4 <property>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	プロパティの名前。
value	xsd:string	必須	プロパティの値。

12.2.2 <property>タグに指定できる CJPA プロバイダ独自のプロパティ

persistence.xml の<property>タグに指定する CJPA プロバイダ独自のプロパティについて説明します。

CJPA プロバイダでは，EntityManagerFactory の createEntityManager()の引数として，指定できるプロパティはありません。なお，persistence.xml ファイルのプロパティの属性値は，数値であっても引用符（ダブルクォーテーションまたはシングルクォーテーション）で囲って指定してください。

注意事項

CJPA プロバイダでは，JPA 仕様で規定されている javax から始まるプロパティを<property>タグに指定できません。JPA 仕様で規定されている javax から始まるプロパティを，persistence.xml の<property>タグに指定した場合の動作は保証しません。

また，JPA 仕様で規定されている javax から始まるプロパティはシステムプロパティとして指定しないでください。JPA 仕様で規定されている javax から始まるプロパティを次に示します。

- javax.persistence.transactionType
- javax.persistence.jtaDataSource
- javax.persistence.nonJtaDataSource
- javax.persistence.provider

CJPA プロバイダ独自のプロパティについて説明します。

これらのプロパティは，persistence.xml ファイルの<property>タグに指定します。

プロパティの値に指定可能範囲外の値が設定された場合，デプロイ時に例外が発生します。また，大文字と小文字は区別されます。

CJPA プロバイダ独自のプロパティについて表で示します。

表 12-5　CJPA プロバイダ独自のプロパティ

プロパティ名	内容	指定可能値	デフォルト
cosminexus.jpa.cache.size.<ENTITY>	<ENTITY>で指定されたエンティティのキャッシュに設定する初期化サイズを指定します。 キャッシュのタイプが Full の場合，エンティティのキャッシュに対する初期化サイズを設定します。 キャッシュタイプが HardWeak または SoftWeak の場合，エンティティのキャッシュの最大サイズを設定します。 キャッシュサイズの指定は，全体で呼び出されるエンティティの最大 ID 数（レコード数）を目安にして設定します。	0～2147483647	1000

プロパティ名	内容	指定可能値	デフォルト
cosminexus.jpa.cache.size.default	エンティティをキャッシュする場合のデフォルトのキャッシュサイズを設定します。 キャッシュのタイプが Full の場合，エンティティのキャッシュに対する初期化サイズを設定します。 キャッシュタイプが HardWeak または SoftWeak の場合，エンティティのキャッシュの最大サイズを設定します。 キャッシュサイズの指定は，全体で呼び出されるエンティティの最大 ID 数（レコード数）を目安にして設定します。	0～2147483647	1000
cosminexus.jpa.cache.type.<ENTITY>	<ENTITY>で指定されたエンティティのキャッシュのタイプを指定します。	指定できる文字列は次のとおりです。 • Full • Weak • HardWeak • SoftWeak • NONE	SoftWeak
cosminexus.jpa.cache.type.default	キャッシュのデフォルトタイプを指定します。	指定できる文字列は次のとおりです。 • Full • Weak • HardWeak • SoftWeak • NONE	SoftWeak
cosminexus.jpa.target-database	接続するデータベースの名前を指定します。各データベース固有の処理を切り分けるために，指定値によりデータベース固有部分を実装したクラスを読み込みます。 Auto を指定した場合は，CJPA プロバイダへの初回リクエスト時に，persistence.xml の<jta-data-source>または<non-jta-data-source>に指定したリソースの情報から，自動でデータベースを識別します。このため，データベースの名前を指定した場合に比べて若干処理時間が長くなります。 通常は Auto を使用しないでデータベースの名前を指定することを推奨します。	指定できる文字列を次に示します。 • Auto • HiRDB • Oracle	Auto

12.3 O/R マッピングファイル

O/R マッピングファイルの構成を次に示します。

タグ名	出現パターン	説明
<entity-mappings>	1 回	ルートタグ。
├ <description>	0 または 1 回	説明を付加します。
├ <persistence-unit-metadata>	0 または 1 回	PersistenceUnit 全体に関する定義を指定します。
│ ├ <xml-mapping-metadata-complete>	0 または 1 回	永続化ユニットのマッピングメタデータを抑止するかどうかを指定します。
│ └ <persistence-unit-defaults>	0 または 1 回	永続化ユニットのデフォルト値を指定します。
│ ├ <schema>	0 または 1 回	スキーマを定義します。
│ ├ <catalog>	0 または 1 回	カタログを定義します。
│ ├ <access>	0 または 1 回	アクセスタイプを指定します。
│ ├ <cascade-persist>	0 または 1 回	カスケード永続化オプションを追加します。
│ └ <entity-listeners>	0 または 1 回	永続化ユニットのデフォルトエンティティリスナを定義します。
│ └ <entity-listener>	0 回以上	エンティティリスナを指定します。
│ ├ <pre-persist>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
│ ├ <post-persist>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
│ ├ <pre-remove>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
│ ├ <post-remove>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
│ ├ <pre-update>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。

タグ名	出現パターン	説明
│ │ │ │ ├		
│ │ │ │ ├ <post-update>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
│ │ │ │ └ <post-load>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
├ <package>	0 または 1 回	同一マッピングファイル内の要素や属性に記載されたクラスのパッケージを指定します。
├ <schema>	0 または 1 回	スキーマを定義します。
├ <catalog>	0 または 1 回	カタログを定義します。
├ <access>	0 または 1 回	アクセス方法を定義します。
├ <sequence-generator>	0 回以上	シーケンスジェネレータを追加します。
├ <table-generator>	0 回以上	テーブルジェネレーターを定義します。
│ └ <unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
│ └ <column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
├ <named-query>	0 回以上	名前付きクエリを定義します。
│ ├ <query>	1 回	クエリ文字列を指定します。
│ └ <hint>	0 回以上	クエリにヒントを付加します。
├ <named-native-query>	0 回以上	名前付きネイティブクエリを定義します。
│ ├ <query>	1 回	クエリ文字列を指定します。
│ └ <hint>	0 回以上	クエリにヒントを付加します。
├ <sql-result-set-mapping>	0 回以上	SQL 結果セットマッピングを定義します。

タグ名	出現パターン	説明
<entity-result>	0 回以上	ネイティブ SQL のクエリ結果をマッピングするための Entity クラスを指定します。
<field-result>	0 回以上	ネイティブ SQL のクエリ結果をマッピングするためのフィールドを指定します。
<column-result>	0 回以上	ネイティブ SQL のクエリ結果をマッピングするためのカラムを指定します。
<mapped-superclass>	0 回以上	永続化ユニットのマップドスーパークラスを定義します。
<description>	0 または 1 回	永続化ユニットのマップドスーパークラスの説明を付加します。
<id-class>	0 または 1 回	マップドスーパークラスで指定した @IdClass を上書きします。
<exclude-default-listeners>	0 または 1 回	マップドスーパークラスとそのサブクラスのデフォルトエンティティリスナを抑止するかどうかを定義します。
<exclude-superclass-listeners>	0 または 1 回	マップドスーパークラスとそのサブクラスのスーパークラスリスナを抑止するかどうかを定義します。
<entity-listeners>	0 または 1 回	コールバックリスナクラスを指定します。
<entity-listener>	0 回以上	エンティティリスナを指定します。
<pre-persist>	0 または 1 回	ライフサイクルコールバックメソッドを指定します。
<post-persist>	0 または 1 回	
<pre-remove>	0 または 1 回	
<post-remove>	0 または 1 回	
<pre-update>	0 または 1 回	

タグ名	出現パターン	説明
│ │ │ │		
│ │ │ ├ <post-update>	0 または 1 回	
│ │ │ └ <post-load>	0 または 1 回	
│ ├ <pre-persist>	0 または 1 回	マップドスーパークラスで，対応するアノテーションによるライフサイクルコールバックメソッドを定義します。
│ ├ <post-persist>	0 または 1 回	
│ ├ <pre-remove>	0 または 1 回	
│ ├ <post-remove>	0 または 1 回	
│ ├ <pre-update>	0 または 1 回	
│ ├ <post-update>	0 または 1 回	
│ ├ <post-load>	0 または 1 回	
│ └ <attributes>	0 または 1 回	要素自体の定義はありません。
│ ├ <id>	0 回以上※1	フィールドやプロパティで指定したマッピングを上書きします。
│ │ ├ <column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
│ │ ├ <generated-value>	0 または 1 回	プライマリキー値生成戦術を指定します。
│ │ ├ <temporal>	0 または 1 回	DATE，TIME，TIMESTAMP 型へマッピングをするときに指定します。
│ │ ├ <table-generator>	0 または 1 回	テーブルジェネレータを追加します。

タグ名	出現パターン	説明
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<sequence-generator>	0 または 1 回	シーケンスジェネレータを追加します。
<embedded-id>	0 または 1 回※1	フィールドやプロパティで指定したマッピングを上書きします。
<attribute-override>	0 回以上	プロパティやフィールドのマッピングを上書きします。
<column>	1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
<basic>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
<lob>	0 または 1 回※2	Lob 型へマッピングをするときに指定します。
<temporal>	0 または 1 回※2	DATE，TIME，TIMESTAMP 型へマッピングをするときに指定します。
<enumerated>	0 または 1 回※2	列挙型へマッピングをするときに指定します。
<version>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
<temporal>	0 または 1 回	DATE，TIME，TIMESTAMP 型へマッピングをするときに指定します。

タグ名	出現パターン	説明
├ <many-to-one>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
├ <join-column>	0 回以上※3	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
├ <join-table>	0 または 1 回※3	多対多や片方向の一対多で使用する結合表を指定します。
├ <join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
├ <inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
└ <unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
└ <column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
└ <cascade>	0 または 1 回	カスケード可能な操作を指定します。
├ <cascade-all>	0 または 1 回	すべての操作をカスケードします。
├ <cascade-persist>	0 または 1 回	persist 操作をカスケードします。
├ <cascade-merge>	0 または 1 回	merge 操作をカスケードします。
├ <cascade-remove>	0 または 1 回	remove 操作をカスケードします。
└ <cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
├ <one-to-many>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
├ <order-by>	0 または 1 回	関連をコレクションに保持する際に付ける順序を指定します。

タグ名	出現パターン	説明
<map-key>	0 または 1 回	Map 型の関連としてマップキーを指定します。
<join-table>	0 または 1 回※4	多対多や片方向の一対多で使用する結合表を指定します。
<join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<join-column>	0 回以上※4	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<cascade>	0 または 1 回	カスケード可能な操作を指定します。
<cascade-all>	0 または 1 回	すべての操作をカスケードします。
<cascade-persist>	0 または 1 回	persist 操作をカスケードします。
<cascade-merge>	0 または 1 回	merge 操作をカスケードします。
<cascade-remove>	0 または 1 回	remove 操作をカスケードします。
<cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
<one-to-one>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<primary-key-join-column>	0 回以上※5	ほかのテーブルに JOIN する外部キーとして使用される，プライマリキーカラムを指定します。

タグ名	出現パターン	説明
<join-column>	0 回以上※5	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<join-table>	0 または 1 回※5	多対多や片方向の一対多で使用する結合表を指定します。
<join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<cascade>	0 または 1 回	カスケード可能な操作を指定します。
<cascade-all>	0 または 1 回	すべての操作をカスケードします。
<cascade-persist>	0 または 1 回	persist 操作をカスケードします。
<cascade-merge>	0 または 1 回	merge 操作をカスケードします。
<cascade-remove>	0 または 1 回	remove 操作をカスケードします。
<cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
<many-to-many>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<order-by>	0 または 1 回	関連をコレクションに保持する際に付ける順序を指定します。
<map-key>	0 または 1 回	Map 型の関連としてマップキーを指定します。
<join-table>	0 または 1 回	多対多や片方向の一対多で使用する結合表を指定します。

タグ名	出現パターン	説明
<join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<cascade>	0 または 1 回	カスケード可能な操作を指定します。
<cascade-all>	0 または 1 回	すべての操作をカスケードします。
<cascade-persist>	0 または 1 回	persist 操作をカスケードします。
<cascade-merge>	0 または 1 回	merge 操作をカスケードします。
<cascade-remove>	0 または 1 回	remove 操作をカスケードします。
<cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
<embedded>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<attribute-override>	0 回以上	プロパティやフィールドのマッピングを上書きします。
<column>	1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
<transient>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<entity>	0 回以上	永続化ユニットのエンティティを定義します。
<description>	0 または 1 回	永続化ユニットのエンティティの説明を付加します。

タグ名	出現パターン	説明
├ <table>	0 または 1 回	Entity クラスの@Table（デフォルト値も含む）を上書きします。
│ └ <unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
│ └ <column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
├ <secondary-table>	0 回以上	Entity クラスのすべての@SecondaryTable と@SecondaryTables（デフォルト値も含む）を上書きします。
│ ├ <primary-key-join-column>	0 回以上	Entity クラスのすべての@PrimaryKeyJoinColumn と@PrimaryKeyJoinColumns（デフォルト値も含む）を上書きします。
│ └ <unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
│ └ <column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
├ <primary-key-join-column>	0 回以上	Entity クラスのすべての@PrimaryKeyJoinColumn と@PrimaryKeyJoinColumns（デフォルト値も含む）を上書きします。
├ <id-class>	0 または 1 回	Entity クラスに指定された@IdClass を上書きします。
├ <inheritance>	0 または 1 回	Entity クラスの@Inheritance（デフォルト値も含む）を上書きします。
├ <discriminator-value>	0 または 1 回	Entity クラスの@DiscriminatorValue（デフォルト値も含む）を上書きします。
├ <discriminator-column>	0 または 1 回	Entity クラスの@DiscriminatorColumn（デフォルト値も含む）を上書きします。

タグ名	出現パターン	説明
<sequence-generator>	0 または 1 回	プライマリキーを作成するシーケンスジェネレータの設定を指定します。
<table-generator>	0 または 1 回	プライマリキーを作成するジェネレータの設定を指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<named-query>	0 回以上	名前付きクエリを定義します。
<query>	1 回	クエリ文字列を指定します。
<hint>	0 回以上	クエリにヒントを付加します。
<named-native-query>	0 回以上	名前付きネイティブクエリを定義します。
<query>	1 回	クエリ文字列を指定します。
<hint>	0 回以上	クエリにヒントを付加します。
<sql-result-set-mapping>	0 回以上	SQL 結果のマッピングを定義します。
<entity-result>	0 回以上	ネイティブ SQL のクエリ結果をマッピングするための Entity クラスを指定します。
<field-result>	0 回以上	ネイティブ SQL のクエリ結果をマッピングするためのフィールドを指定します。
<column-result>	0 回以上	ネイティブ SQL のクエリ結果をマッピングするためのカラムを指定します。
<exclude-default-listeners>	0 または 1 回	Entity クラスとそのサブクラスのデフォルトエンティティリスナを抑止します。
<exclude-superclass-listeners>	0 または 1 回	Entity クラスとそのサブクラスのスーパークラスリスナを抑止します。

タグ名	出現パターン	説明
<entity-listeners>	0 または 1 回	Entity クラスの@EntityListeners を上書きします。
<entity-listener>	0 回以上	エンティティリスナを指定します。
<pre-persist>	0 または 1 回	マップドスーパークラスで，対応するアノテーションによるライフサイクルコールバックメソッドの定義を上書きします。
<post-persist>	0 または 1 回	
<pre-remove>	0 または 1 回	
<post-remove>	0 または 1 回	
<pre-update>	0 または 1 回	
<post-update>	0 または 1 回	
<post-load>	0 または 1 回	
<pre-persist>	0 または 1 回	Entity クラスで，対応するアノテーションによるライフサイクルコールバックメソッドの定義を上書きします。
<post-persist>	0 または 1 回	
<pre-remove>	0 または 1 回	
<post-remove>	0 または 1 回	
<pre-update>	0 または 1 回	
<post-update>	0 または 1 回	

タグ名	出現パターン	説明
├ <post-load>	0 または 1 回	
├ <attribute-override>	0 回以上	Entity クラスの@AttributeOverride もしくは@AttributeOverrides で定義された値に追加されます。
└ <column>	1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
├ <association-override>	0 回以上	Entity クラスの@AssociationOverride もしくは@AssociationOverrides で定義された値に追加されます。
└ <join-column>	1 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
└ <attributes>	0 または 1 回	要素自自体の定義はありません。
├ <id>	0 回以上※1	フィールドやプロパティで指定したマッピングを上書きします。
├ <column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
├ <generated-value>	0 または 1 回	プライマリキー値生成戦術を指定します。
├ <temporal>	0 または 1 回	DATE，TIME，TIMESTAMP 型へマッピングを行うときに指定します。
├ <table-generator>	0 または 1 回	テーブルジェネレータを追加します。
└ <unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
└ <column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
└ <sequence-generator>	0 または 1 回	プライマリキーを作成するシーケンスジェネレータの設定を指定します。

タグ名	出現パターン	説明
│ │ │ │		
│ │ ├ <embedded-id>	0 または 1 回※1	フィールドやプロパティで指定したマッピングを上書きします。
│ │ │ └ <attribute-override>	0 回以上	Entity クラスの@AttributeOverride もしくは@AttributeOverrides で定義された値に追加されます。
│ │ │ │ └ <column>	1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
│ │ ├ <basic>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
│ │ │ ├ <column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
│ │ │ ├ <lob>	0 または 1 回※2	Lob 型へマッピングをするときに指定します。
│ │ │ ├ <temporal>	0 または 1 回	DATE，TIME，TIMESTAMP 型へマッピングをするときに指定します。
│ │ │ └ <enumerated>	0 または 1 回※2	列挙型へマッピングをするときに指定します。
│ │ ├ <version>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
│ │ │ ├ <column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
│ │ │ └ <temporal>	0 または 1 回	DATE，TIME，TIMESTAMP 型へマッピングをするときに指定します。
│ │ ├ <many-to-one>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。

タグ名	出現パターン	説明
<join-column>	0 回以上※4	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<join-table>	0 または 1 回※4	多対多や片方向の一対多で使用する結合表を指定します。
<join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<cascade>	0 または 1 回	カスケード可能な操作を指定します。
<cascade-all>	0 または 1 回	すべての操作をカスケードします。
<cascade-persist>	0 または 1 回	persist 操作をカスケードします。
<cascade-merge>	0 または 1 回	merge 操作をカスケードします。
<cascade-remove>	0 または 1 回	remove 操作をカスケードします。
<cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
<one-to-many>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<order-by>	0 または 1 回	関連をコレクションに保持する際に付ける順序を指定します。
<map-key>	0 または 1 回	Map 型の関連としてマップキーを指定します。
<join-table>	0 または 1 回※3	多対多や片方向の一対多で使用する結合表を指定します。

タグ名	出現パターン	説明
│ │ │ │		
│ │ │ │ ├ <join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
│ │ │ │ ├ <inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
│ │ │ │ └ <unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
│ │ │ │ └ <column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
│ │ │ ├ <join-column>	0 回以上※3	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
│ │ │ └ <cascade>	0 または 1 回	カスケード可能な操作を指定します。
│ │ │ ├ <cascade-all>	0 または 1 回	すべての操作をカスケードします。
│ │ │ ├ <cascade-persist>	0 または 1 回	persist 操作をカスケードします。
│ │ │ ├ <cascade-merge>	0 または 1 回	merge 操作をカスケードします。
│ │ │ ├ <cascade-remove>	0 または 1 回	remove 操作をカスケードします。
│ │ │ └ <cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
│ │ ├ <one-to-one>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
│ │ │ ├ <primary-key-join-column>	0 回以上※5	Entity クラスのすべての @PrimaryKeyJoinColumn と @PrimaryKeyJoinColumns（デフォルト値も含む）を上書きします。
│ │ │ ├ <join-column>	0 回以上※5	所有者側エンティティに対応する結合表の外部キーカラムを指定します。

タグ名	出現パターン	説明
<join-table>	0 または 1 回※5	多対多や片方向の一対多で使用する結合表を指定します。
<join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<cascade>	0 または 1 回	カスケード可能な操作を指定します。
<cascade-all>	0 または 1 回	すべての操作をカスケードします。
<cascade-persist>	0 または 1 回	persist 操作をカスケードします。
<cascade-merge>	0 または 1 回	merge 操作をカスケードします。
<cascade-remove>	0 または 1 回	remove 操作をカスケードします。
<cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
<many-to-many>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<order-by>	0 または 1 回	関連をコレクションに保持する際に付ける順序を指定します。
<map-key>	0 または 1 回	Map 型の関連としてマップキーを指定します。
<join-table>	0 または 1 回	多対多や片方向の一対多で使用する結合表を指定します。
<join-column>	0 回以上	所有者側エンティティに対応する結合表の外部キーカラムを指定します。

タグ名	出現パターン	説明
<inverse-join-column>	0 回以上	被所有者側エンティティに対応する結合表の外部キーカラムを指定します。
<unique-constraint>	0 回以上	DDL にユニーク制約を付加します。
<column-name>	1 回以上	ユニーク制約を付加するカラムの名前を指定します。
<cascade>	0 または 1 回	カスケード可能な操作を指定します。
<cascade-all>	0 または 1 回	すべての操作をカスケードします。
<cascade-persist>	0 または 1 回	persist 操作をカスケードします。
<cascade-merge>	0 または 1 回	merge 操作をカスケードします。
<cascade-remove>	0 または 1 回	remove 操作をカスケードします。
<cascade-refresh>	0 または 1 回	refresh 操作をカスケードします。
<embedded>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<attribute-override>	0 回以上	Entity クラスの@AttributeOverride もしくは@AttributeOverrides で定義された値に追加されます。
<column>	1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
<transient>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
<embeddable>	0 回以上	永続化ユニットの埋め込み可能クラスを定義します。
<description>	0 または 1 回	永続化ユニットの埋め込み可能クラスの説明を付加します。
<attributes>	0 または 1 回	要素自体の定義はありません。

タグ名	出現パターン	説明
├ <basic>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。
│ ├ <column>	0 または 1 回	Persistent フィールドのプロパティに，カラムのマッピングを指定します。
│ ├ <lob>	0 または 1 回※2	Lob 型へマッピングをするときに指定します。
│ ├ <temporal>	0 または 1 回※2	DATE，TIME，TIMESTAMP 型へマッピングをするときに指定します。
│ └ <enumerated>	0 または 1 回※2	列挙型へマッピングを行うときに指定します。
└ <transient>	0 回以上	フィールドやプロパティで指定したマッピングを上書きします。

注※1

<id>タグおよび<embedded-id>タグのどちらか一つを指定します。

注※2

<lob>タグ，<temporal>タグ，および<enumerated>タグのどれか一つを指定します。

注※3

<join-column>タグ，および<join-table>タグのどちらか一つを指定します。

注※4

<join-table>タグ，および<join-column>タグのどちらか一つを指定します。

注※5

<primary-key-join-column>タグ，<join-column>タグ，および<join-table>タグのどれか一つを指定します。

12.3.1 entity-mappings 以下の要素

(1) <entity-mappings>

ルートタグ。

指定できる属性を次の表に示します。

表 12–6 <entity-mappings>の属性

属性名	型	任意／必須	説明
version	orm:versionType	必須	JPA のバージョンを指定します。

(2) <package>

package 要素は，同一マッピングファイル内の要素や属性に記載されたクラスのパッケージを指定します。package 要素は，パッケージ名付きのクラス名がクラスに指定され，package 要素に指定されたパッケージ名と異なる場合に上書きされます。

(3) <schema>

schema 要素は，同一マッピングファイル内に記載されたエンティティにだけ適用されます。

schema 要素は，次の要素や属性によって上書きされます。

- マッピングファイル内に記載された Entity クラスの@Table や@SecondaryTable に明確に指定された schema 要素。
- entity 要素に定義された table 要素や secondary-table 要素の schema 属性。

(4) <catalog>

catalog 要素は，同一マッピングファイル内に記載されたエンティティにだけ適用されます。

catalog 要素は，次の要素や属性によって上書きされます。

- マッピングファイル内に記載された Entity クラスの@Table や@SecondaryTable に明確に指定された catalog 要素。
- entity 要素に定義された table 下位要素や secondary-table 下位要素の catalog 属性。

注意事項

catalog 要素はデータベースによって，存在しない場合があります。CJPA プロバイダでサポートしている Oracle と HiRDB では catalog 要素は存在しません。そのため，catalog 要素を指定できません。指定した場合は，アプリケーション実行時に例外が発生します。

(5) <access>

access 要素は，同一マッピングファイル内に記載された，管理されたクラスで適用されます。

access 要素は，次のアノテーションや属性によって上書きされます。

- Entity クラスのアノテーションの指定位置によって決まるアクセスタイプ。
- entity 要素，mapped-superclass 要素，および embeddable 要素で定義された access 属性。

注　指定値には，PROPERTY または FIELD を指定します。エンティティクラスのフィールドに対するアクセス方法の指定については，「8.12.3　エンティティクラスのフィールドに対するアクセス方法の指定」を参照してください。

(6) <sequence-generator>

シーケンスジェネレータを追加します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–7 <sequence-generator>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.51 @SequenceGenerator」の name 属性を参照してください。
sequence-name	xsd:string	任意	「8.22.51 @SequenceGenerator」の sequenceName 属性を参照してください。
initial-value	xsd:int	任意	「8.22.51 @SequenceGenerator」の initialValue 属性を参照してください。
allocation-size	xsd:int	任意	「8.22.51 @SequenceGenerator」の allocationSize 属性を参照してください。

12.3.2 persistence-unit-metadata 以下の要素

(1) <persistence-unit-metadata>

PersistenceUnit 全体に関する定義を指定します。

(2) <xml-mapping-metadata-complete>

xml-mapping-metadata-complete 要素を指定すると，永続化ユニットのマッピングメタデータは抑止され，クラスで指定したアノテーションは無視されます。

xml-mapping-metadata-complete 要素が指定され，XML 要素が省略されると，デフォルト値が有効になります。

注意事項

アノテーションが指定されている場合，この要素を指定することによって，KDJE55532-W のメッセージが出力されることがあります。

(3) <persistence-unit-defaults>

永続化ユニットのデフォルト値を指定します。

(4) <schema>

schema 要素は，永続化ユニット内のすべての Entity クラス，テーブルジェネレータ，および結合表に適用されます。

schema 要素は，次の要素や属性によって上書きされます。

- entity-mappings 要素の schema 要素。
- Entity クラスの@Table や@SecondaryTable に明確に指定された schema 属性。
- entity 要素の table 要素や secondary-table 要素に指定された schema 属性。
- @TableGenerator や table-generator 要素に明確に指定された schema 要素。
- @JoinTable や join-table 要素に明確に指定された schema 要素。

(5) <catalog>

catalog 要素は，永続化ユニット内のすべての Entity クラス，テーブルジェネレータ，結合表に適用されます。

catalog 要素は，次の要素や属性によって上書きされます。

- entity-mappings 要素の catalog 要素。
- Entity クラスの@Table や@SecondaryTable に明確に指定された catalog 属性。
- entity 要素の table 要素や secondary-table 要素に指定された catalog 属性。
- @TableGenerator や table-generator 要素に明確に指定された catalog 要素。
- @JoinTable や join-table 要素に明確に指定された catalog 要素。

注意事項

CJPA プロバイダでサポートしている Oracle と HiRDB では catalog は存在しません。そのため，catalog を指定できません。指定した場合，アプリケーション実行時に例外が発生します。

(6) <access>

access 要素は，永続化ユニット内のすべての管理されたクラスで適用されます。

access 要素は，次のアノテーションや要素や属性によって上書きされます。

- Entity クラスのアノテーションの指定位置によって決まるアクセスタイプ。
- entity-mappings 要素の access 要素。
- entity 要素や mapped-superclass 要素や embeddable 要素で定義された access 属性。

注意事項

指定値には，PROPERTY または FIELD を指定します。エンティティクラスのフィールドに対するアクセス方法の指定については，「8.12.3　エンティティクラスのフィールドに対するアクセス方法の指定」を参照してください。

(7)　<cascade-persist>

cascade-persist 要素は，永続化ユニット内のすべてのリレーションシップで適用されます。

cascade-persist 要素の指定は，アノテーションか O/R マッピングファイルで指定された値に加えてすべてのリレーションシップにカスケード永続化オプションを追加します。

注意事項

cascade-persist 要素を指定すると，上書きや無効化することができません。

(8)　<entity-listeners>

entity-listeners 要素は，永続化ユニットのデフォルトエンティティリスナを定義します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

12.3.3　table-generator 以下の要素

(1)　<table-generator>

table-generator 要素によって定義されたジェネレータは，永続化ユニットに適用されます。

定義されたジェネレータは，アノテーションで定義されたジェネレータに追加されます。アノテーションで同名のジェネレータが定義されたら，table-generator 要素で定義したジェネレータが上書きされます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–8　<table-generator>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.55　@TableGenerator」の name 属性を参照してください。
table	xsd:string	任意	「8.22.55　@TableGenerator」の table 属性を参照してください。

属性名	型	任意／必須	説明
catalog	xsd:string	任意	「8.22.55　@TableGenerator」の catalog 属性を参照してください。
schema	xsd:string	任意	「8.22.55　@TableGenerator」の schema 属性を参照してください。
pk-column-name	xsd:string	任意	「8.22.55　@TableGenerator」の pkColumnName 属性を参照してください。
value-column-name	xsd:string	任意	「8.22.55　@TableGenerator」の valueColumnName 属性を参照してください。
pk-column-value	xsd:string	任意	「8.22.55　@TableGenerator」の pkColumnValue 属性を参照してください。
initial-value	xsd:int	任意	「8.22.55　@TableGenerator」の initialValue 属性を参照してください。
allocation-size	xsd:int	任意	「8.22.55　@TableGenerator」の allocationSize 属性を参照してください。

12.3.4　named-query 以下の要素

(1)　<named-query>

named-query 要素によって定義された名前付きクエリは，永続化ユニットに適用されます。

定義された名前付きクエリは，アノテーションで定義された名前付きクエリに追加されます。アノテーションで同名の名前付きクエリが定義されたら，named-query 要素で定義した名前付きクエリが上書きされます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–9　<named-query>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.35　@NamedQuery」の name 属性を参照してください。

12.3.5　named-native-query 以下の要素

(1)　<named-native-query>

named-native-query 要素によって定義された名前付きネイティブクエリは，永続化ユニットに適用されます。

定義された名前付きネイティブクエリは，アノテーションで定義された名前付きネイティブクエリに追加されます。アノテーションで同名の名前付きネイティブクエリが定義されたら，O/R マッピングファイルの named-native-query 要素で定義した名前付きネイティブクエリを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–10　<named-native-query>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.33　@NamedNativeQuery」の name 属性を参照してください。
result-class	xsd:string	任意	「8.22.33　@NamedNativeQuery」の resultClass 属性を参照してください。
result-set-mapping	xsd:string	任意	「8.22.33　@NamedNativeQuery」の resultSetMapping 属性を参照してください。

12.3.6　sql-result-set-mapping 以下の要素

(1)　<sql-result-set-mapping>

sql-result-set-mapping 要素によって定義された SQL 結果セットマッピングは，永続化ユニットに適用されます。

定義された SQL 結果セットマッピングは，アノテーションで定義された SQL 結果セットマッピングに追加されます。アノテーションで同名の SQL 結果セットマッピングが定義されたら，O/R マッピングファイルの sql-result-set-mapping 要素で定義した SQL 結果セットマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–11　<sql-result-set-mapping>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.52　@SqlResultSetMapping」の name 属性を参照してください。

12.3.7　mapped-superclass 以下の要素

次に示す要素と属性は，要素や属性の対象であるマップドスーパークラスだけに適用されます。

(1)　<mapped-superclass>

mapped-superclass 要素は，永続化ユニットのマップドスーパークラスを定義します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–12　<mapped-superclass>の属性

属性名	型	任意／必須	説明
class	xsd:string	必須	マップドスーパークラスのクラス名
access	orm:access-type	任意	access 属性は，マップドスーパークラスのアクセスタイプを定義します。access 属性は，マップドスーパークラスに与えた persistence-unit-defaults 要素（デフォルトとして指定した要素）や entity-mappings 要素（永続化ユニット全体に有効な要素）で指定したアクセスタイプを上書きします。※1
metadata-complete※2	xsd:boolean	任意	metadata-complete 属性が mapped-superclass 要素自身に指定されたら，マップドスーパークラスや，マップドスーパークラスのフィールドやプロパティに指定されたアノテーションは無視されます。 metadata-complete が mapped-superclass 要素に指定され，XML 要素が省略されると，デフォルト値が有効になります。

注※1　指定値には，PROPERTY または FIELD を指定します。エンティティクラスのフィールドに対するアクセス方法の指定については，「8.12.3　エンティティクラスのフィールドに対するアクセス方法の指定」を参照してください。

注※2　アノテーションが指定されている場合，この要素を指定することによって，KDJE55532-W のメッセージが出力されることがあります。

(2) <id-class>

id-class 要素は，マップドスーパークラスで指定した@IdClass を上書きします。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–13 <id-class>の属性

属性名	型	任意／必須	説明
class	xsd:string	必須	「8.22.22 @IdClass」の value 属性を参照してください。

(3) <exclude-default-listeners>

exclude-default-listeners 要素は，@ExcludeDefaultListeners がマップドスーパークラスに指定されたかどうかに関係なく適用されます。

exclude-default-listeners 要素は，マップドスーパークラスとそのサブクラスのデフォルトエンティティリスナを抑止します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(4) <exclude-superclass-listeners>

exclude-superclass-listeners 要素は，@ExcludeSuperclassListeners がマップドスーパークラスに指定されたかどうかに関係なく適用されます。

exclude-superclass-listeners 要素は，マップドスーパークラスとそのサブクラスのスーパークラスリスナを抑止します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(5) <entity-listeners>

entity-listeners 要素は，マップドスーパークラスの@EntityListeners を上書きします。

ほかの方法で抑止されない場合，これらのリスナはマップドスーパークラスとそのサブクラスで適用します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(6) <pre-persist>，<post-persist>，<pre-remove>，<post-remove>，<pre-update>，<post-update>，<post-load>

これらの要素は，マップドスーパークラスで，対応するアノテーションによるライフサイクルコールバックメソッドの定義を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–14　<pre-persist>，<post-persist>，<pre-remove>，<post-remove>，<pre-update>，<post-update>，<post-load>の属性

属性名	型	任意／必須	説明
method-name	xsd:string	必須	対象のメソッド名。

(7) <id>

id 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–15　<id>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	プライマリキープロパティ，もしくはフィールドであることを指定します。

(8) <embedded-id>

embedded-id 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–16　<embedded-id>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	複合プライマリキーであることを指定します。

(9)　<basic>

basic 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–17　<basic>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	型をマッピングするメソッドおよびフィールドを指定します。
fetch	orm:fetch-type	任意	Fetch 戦略の値を指定します。 詳細は「8.22.5　@Basic」の fetch 属性を参照してください。
optional	xsd:boolean	任意	フィールド（プロパティ）が null を使用するかどうかを指定します。 詳細は「8.22.5　@Basic」の optional 属性を参照してください。

(10)　<version>

version 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–18　<version>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	version プロパティもしくはフィールドであることを指定します。

(11) <many-to-one>

many-to-one 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–19 <many-to-one>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	多対一リレーションシップのメソッドもしくはフィールドであることを指定します。
target-entity	xsd:string	任意	関連する Entity クラスを指定します。 詳細は「8.22.29 @ManyToOne」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	Fetch 戦略の値を指定します。 詳細は「8.22.29 @ManyToOne」の fetch 属性を参照してください。
optional	xsd:boolean	任意	すべての非プリミティブなフィールドおよびプロパティの値に null を指定するかどうかを定義します。 詳細は「8.22.29 @ManyToOne」の optional 属性を参照してください。

(12) <one-to-many>

one-to-many 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–20 <one-to-many>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	一対多リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	関連する Entity クラスを指定します。詳細は「8.22.36 @OneToMany」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	Fetch 戦略の値を指定します。

属性名	型	任意／必須	説明
			詳細は「8.22.36　@OneToMany」の fetch 属性を参照してください。
mapped-by	xsd:string	任意	被所有者側の Entity クラスの要素に付与し，所有者側の Entity クラスで関係を保持しているフィールド（プロパティ）名を指定します。 詳細は「8.22.36　@OneToMany」の mappedBy 属性を参照してください。

(13)　<one-to-one>

one-to-one 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–21　<one-to-one>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	一対一リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	「8.22.37　@OneToOne」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	「8.22.37　@OneToOne」の fetch 属性を参照してください。
optional	xsd:boolean	任意	「8.22.37　@OneToOne」の optional 属性を参照してください。
mapped-by	xsd:string	任意	「8.22.37　@OneToOne」の mappedBy 属性を参照してください。

(14)　<many-to-many>

many-to-many 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–22　<many-to-many>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	多対多リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	「8.22.28　@ManyToMany」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	「8.22.28　@ManyToMany」の fetch 属性を参照してください。
mapped-by	xsd:string	任意	「8.22.28　@ManyToMany」の mappedBy 属性を参照してください。

(15)　<embedded>

embedded 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–23　<embedded>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	埋め込みオブジェクトであるプロパティまたはフィールド。

(16)　<transient>

transient 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–24　<transient>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	永続的でないプロパティまたはフィールド。

12.3.8　entity 以下の要素

次に示す要素と属性は，下位要素や属性の対象である Entity クラスにだけ適用されます。

(1)　<entity>

entity 要素は，永続化ユニットのエンティティを定義します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–25　<entity>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	name 属性は，エンティティの名前を定義します。name 属性は，@Entity の name 要素によって定義されたエンティティ名を，明確に指定されたかデフォルトかに関係なく上書きします。 Entity クラスの名前を上書きした場合の動作は保証しません。
class	xsd:string	必須	エンティティのクラス名。
access	orm:access-type	任意	access 属性は，Entity クラスのアクセスタイプを定義します。access 属性は，Entity クラスに与えた persistence-unit-defaults 要素（デフォルトとして指定した要素）や entity-mappings 要素（永続化ユニット全体に有効な要素）で指定したアクセスタイプを上書きします。※1
metadata-complete※2	xsd:boolean	任意	metadata-complete 属性が entity 要素自身に指定されたら，Entity クラスや，Entity クラスのフィールドやプロパティに指定されたアノテーションは無視されます。 metadata-complete が entity 要素に指定され，XML 要素が省略されると，デフォルト値が有効になります。

注※1　指定値には，PROPERTY または FIELD を指定します。エンティティクラスのフィールドに対するアクセス方法の指定については，「8.12.3　エンティティクラスのフィールドに対するアクセス方法の指定」を参照してください。

注※2　アノテーションが指定されている場合，この要素を指定することによって，KDJE55532-W のメッセージが出力されることがあります。

(2)　<table>

table 要素は，Entity クラスの@Table（デフォルト値も含む）を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–26　<table>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.54　@Table」の name 属性を参照してください。
catalog	xsd:string	任意	「8.22.54　@Table」の catalog 属性を参照してください。
schema	xsd:string	任意	「8.22.54　@Table」の schema 属性を参照してください。

(3)　<secondary-table>

secondary-table 要素は，Entity クラスのすべての@SecondaryTable と@SecondaryTables（デフォルト値も含む）を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–27　<secondary-table>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.49　@SecondaryTable」の name 属性を参照してください。
catalog	xsd:string	任意	「8.22.49　@SecondaryTable」の catalog 属性を参照してください。
schema	xsd:string	任意	「8.22.49　@SecondaryTable」の schema 属性を参照してください。

(4)　<primary-key-join-column>

primary-key-join-column 要素は，Entity クラスのすべての@PrimaryKeyJoinColumn と @PrimaryKeyJoinColumns（デフォルト値も含む）を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–28　<primary-key-join-column>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.46　@PrimaryKeyJoinColumn」の name 属性を参照してください。

属性名	型	任意／必須	説明
referenced-column-name	xsd:string	任意	「8.22.46　@PrimaryKeyJoinColumn」のreferencedColumnName 属性を参照してください。
column-definition	xsd:string	任意	「8.22.46　@PrimaryKeyJoinColumn」のcolumnDefinition 属性を参照してください。

(5)　<id-class>

id-class 要素は，Entity クラスに指定された@IdClass を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-29　<id-class>の属性

属性名	型	任意／必須	説明
value	xsd:string	必須	「8.22.22　@IdClass」の value 属性を参照してください。

(6)　<inheritance>

inheritance 要素は，Entity クラスの@Inheritance（デフォルト値も含む）を上書きします。

inheritance 要素は，Entity クラスと，そのサブクラスで適用します（アノテーションや XML 要素で指定したサブクラスが，別の方法で上書きされなかった場合）。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-30　<inheritance>の属性

属性名	型	任意／必須	説明
strategy	orm:inheritance-type	任意	「8.22.23　@Inheritance」の strategy 属性を参照してください。

(7)　<discriminator-value>

discriminator-value 要素は，Entity クラスの@DiscriminatorValue（デフォルト値も含む）を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(8)　<discriminator-column>

discriminator-column 要素は，Entity クラスの@DiscriminatorColumn（デフォルト値も含む）を上書きします。

discriminator-column 要素は，Entity クラスと，そのサブクラスで適用します（アノテーションや XML 要素で指定したサブクラスが，別の方法で上書きされなかった場合）。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–31　<discriminator-column>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.8　@DiscriminatorColumn」の name 属性を参照してください。
discriminator-type	orm:discriminator-type	任意	「8.22.8　@DiscriminatorColumn」の discriminatorType 属性を参照してください。
column-definition	xsd:string	任意	「8.22.8　@DiscriminatorColumn」の columnDefinition 属性を参照してください。
length	xsd:int	任意	「8.22.8　@DiscriminatorColumn」の length 属性を参照してください。

(9)　<sequence-generator>

sequence-generator 要素によって定義されたジェネレータは，アノテーションによって定義されたジェネレータや，O/R マッピングファイルで定義されたほかのジェネレータに加えられます。アノテーションで同名のジェネレータが定義されたら，sequence-generator 要素で定義されたジェネレータで上書きします。

sequence-generator 要素によって定義されたジェネレータは，永続化ユニットに適用されます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–32　<sequence-generator>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.51　@SequenceGenerator」の name 属性を参照してください。
sequence-name	xsd:string	任意	「8.22.51　@SequenceGenerator」の sequenceName 属性を参照してください。
initial-value	xsd:int	任意	「8.22.51　@SequenceGenerator」の initialValue 属性を参照してください。
allocation-size	xsd:int	任意	「8.22.51　@SequenceGenerator」の allocationSize 属性を参照してください。

(10)　<table-generator>

table-generator 要素によって定義されたジェネレータは，アノテーションによって定義されたジェネレータや，O/R マッピングファイルで定義されたほかのジェネレータに加えられます。アノテーションで同名のジェネレータが定義されたら，table-generator 要素で定義されたジェネレータで上書きします。

table-generator 要素によって定義されたジェネレータは，永続化ユニットに適用されます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–33　<table-generator>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.55　@TableGenerator」の name 属性を参照してください。
table	xsd:string	任意	「8.22.55　@TableGenerator」の table 属性を参照してください。
catalog	xsd:string	任意	「8.22.55　@TableGenerator」の catalog 属性を参照してください。
schema	xsd:string	任意	「8.22.55　@TableGenerator」の schema 属性を参照してください。
pk-column-name	xsd:string	任意	「8.22.55　@TableGenerator」の pkColumnName 属性を参照してください。
value-column-name	xsd:string	任意	「8.22.55　@TableGenerator」の valueColumnName 属性を参照してください。

属性名	型	任意／必須	説明
pk-column-value	xsd:string	任意	「8.22.55　@TableGenerator」の pkColumnValue 属性を参照してください。
initial-value	xsd:int	任意	「8.22.55　@TableGenerator」の initialValue 属性を参照してください。
allocation-size	xsd:int	任意	「8.22.55　@TableGenerator」の allocationSize 属性を参照してください。

(11)　<named-query>

named-query 要素によって定義された名前付きクエリは，アノテーションによって定義された名前付きクエリや，O/R マッピングファイルで定義されたほかの名前付きクエリに加えられます。アノテーションで同名の名前付きクエリが定義されたら，named-query 要素で定義された名前付きクエリで上書きします。

named-query 要素によって定義された名前付きクエリは，永続化ユニットに適用されます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–34　<named-query>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.35　@NamedQuery」の name 属性を参照してください。

(12)　<named-native-query>

named-native-query 要素によって定義された名前付きネイティブクエリは，アノテーションによって定義された名前付きネイティブクエリや，O/R マッピングファイルで定義されたほかの名前付きネイティブクエリに加えられます。アノテーションで同名の名前付きネイティブクエリが定義されたら，named-native-query 要素で定義された名前付きネイティブクエリで上書きします。

named-native-query 要素によって定義された名前付きネイティブクエリは，永続化ユニットに適用されます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–35　<named-native-query>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.33　@NamedNativeQuery」の name 属性を参照してください。
result-class	xsd:string	任意	「8.22.33　@NamedNativeQuery」の resultClass 属性を参照してください。
result-set-mapping	xsd:string	任意	「8.22.33　@NamedNativeQuery」の resultSetMapping 属性を参照してください。

(13)　<sql-result-set-mapping>

sql-result-set-mapping 要素によって定義された SQL 結果のマッピングは，アノテーションによって定義された SQL 結果のマッピングや，O/R マッピングファイルで定義されたほかの SQL 結果のマッピングに加えられます。アノテーションで同名の SQL 結果のマッピングが定義されたら，sql-result-set-mapping 要素で定義された SQL 結果のマッピングで上書きします。

sql-result-set-mapping 要素によって定義された SQL 結果のマッピングは，永続化ユニットに適用されます。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–36　<sql-result-set-mapping>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.52　@SqlResultSetMapping」の name 属性を参照してください。

(14)　<exclude-default-listeners>

exclude-default-listeners 要素は，@ExcludeDefaultListeners が Entity クラスに指定されたかどうかに関係なく適用されます。

exclude-default-listeners 要素は，Entity クラスとそのサブクラスのデフォルトエンティティリスナを抑止します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(15) <exclude-superclass-listeners>

exclude-superclass-listeners 要素は，@ExcludeSuperclassListeners が Entity クラスに指定されたかどうかに関係なく適用されます。

exclude-superclass-listeners 要素は，Entity クラスとそのサブクラスのスーパークラスリスナを抑止します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(16) <entity-listeners>

entity-listeners 要素は，Entity クラスの@EntityListeners を上書きします。

ほかの方法で抑止されない場合，これらのリスナは Entity クラスとそのサブクラスで適用します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(17) <pre-persist>，<post-persist>，<pre-remove>，<post-remove>，<pre-update>，<post-update>，<post-load>

これらの要素は，Entity クラスで，対応するアノテーションによるライフサイクルコールバックメソッドの定義を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–37　<pre-persist>，<post-persist>，<pre-remove>，<post-remove>，<pre-update>，<post-update>，<post-load>の属性

属性名	型	任意／必須	説明
method-name	xsd:string	必須	対象のメソッド名。

(18) <attribute-override>

attribute-override 要素は，Entity クラスの@AttributeOverride または@AttributeOverrides で定義された値に加えられます。attribute-override 要素は，同一の属性名の AttributeOverride 要素を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–38　<attribute-override>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.3　@AttributeOverride」の name 属性を参照してください。

(19)　<association-override>

association-override 要素は，Entity クラスの@AssociationOverride または@AssociationOverrides で定義された値に加えられます。association-override 要素は，同一の属性名の AssociationOverride 要素を上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–39　<association-override>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.1　@AssociationOverride」の name 属性を参照してください。

(20)　<id>

id 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–40　<id>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	プライマリキープロパティまたはフィールド。

(21)　<embedded-id>

embedded-id 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–41　<embedded-id>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	複合プライマリキー。

(22)　<basic>

basic 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–42　<basic>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	型をマッピングするメソッドまたはフィールド。
fetch	orm:fetch-type	任意	「8.22.5　@Basic」の fetch 属性を参照してください。
optional	xsd:boolean	任意	「8.22.5　@Basic」の optional 属性を参照してください。

(23)　<version>

version 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–43　<version>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	version プロパティまたはフィールド。

(24)　<many-to-one>

many-to-one 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–44　<many-to-one>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	多対一リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	「8.22.29　@ManyToOne」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	「8.22.29　@ManyToOne」の fetch 属性を参照してください。
optional	xsd:boolean	任意	「8.22.29　@ManyToOne」の optional 属性を参照してください。

(25)　<one-to-many>

one-to-many 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–45　<one-to-many>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	一対多リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	「8.22.36　@OneToMany」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	「8.22.36　@OneToMany」の fetch 属性を参照してください。
mapped-by	xsd:string	任意	「8.22.36　@OneToMany」の mappedBy 属性を参照してください。

(26)　<one-to-one>

one-to-one 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–46　<one-to-one>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	一対一リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	「8.22.37　@OneToOne」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	「8.22.37　@OneToOne」の fetch 属性を参照してください。
optional	xsd:boolean	任意	「8.22.37　@OneToOne」の optional 属性を参照してください。
mapped-by	xsd:string	任意	「8.22.37　@OneToOne」の mappedBy 属性を参照してください。

(27)　<many-to-many>

many-to-many 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–47　<many-to-many>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	多対多リレーションシップのメソッドまたはフィールド。
target-entity	xsd:string	任意	「8.22.28　@ManyToMany」の targetEntity 属性を参照してください。
fetch	orm:fetch-type	任意	「8.22.28　@ManyToMany」の fetch 属性を参照してください。
mapped-by	xsd:string	任意	「8.22.28　@ManyToMany」の mappedBy 属性を参照してください。

(28)　<embedded>

embedded 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-48　<embedded>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	埋め込みオブジェクトであるプロパティまたはフィールド。

(29)　<transient>

transient 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-49　<transient>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	永続的でないプロパティまたはフィールド。

12.3.9　embeddable 以下の要素

次に示す要素と属性は，要素や属性の対象である埋め込み可能クラスだけに適用されます。

(1)　<embeddable>

embeddable 要素は，永続化ユニットの埋め込み可能クラスを定義します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-50　<embeddable>の属性

属性名	型	任意／必須	説明
class	xsd:string	必須	埋め込み可能クラスのクラス名。
access	orm:access-type	任意	access 属性は，埋め込み可能クラスのアクセスタイプを定義します。access 属性は，埋め込み可能クラスに与えた persistence-unit-defaults 要素（デフォルトとして指定した要素）や entity-mappings 要素（永続化ユニット全体に有効な要素）で指定したアクセスタイプを上書きします。※1

属性名	型	任意／必須	説明
metadata-complete※2	xsd:boolean	任意	metadata-complete 属性が embeddable 要素自身に指定されたら，埋め込み可能クラスや，埋め込み可能クラスのフィールドやプロパティに指定されたアノテーションは無視されます。 metadata-complete が embeddable 要素に指定され，XML 要素が省略されると，デフォルト値が有効になります。

注※1　指定値には，PROPERTY または FIELD を指定します。エンティティクラスのフィールドに対するアクセス方法の指定については，「8.12.3　エンティティクラスのフィールドに対するアクセス方法の指定」を参照してください。

注※2　アノテーションが指定されている場合，この要素を指定することによって，KDJE55532-W のメッセージが出力されることがあります。

(2)　<basic>

basic 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–51　<basic>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	型をマッピングするメソッドまたはフィールド。
fetch	orm:fetch-type	任意	「8.22.5　@Basic」の fetch 属性を参照してください。
optional	xsd:boolean	任意	「8.22.5　@Basic」の optional 属性を参照してください。

(3)　<transient>

transient 要素は，フィールドやプロパティで指定したマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–52　<transient>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	永続的でないプロパティまたはフィールド。

12.3.10　その他の要素

(1)　<description>

説明を付加します。

(2)　<entity-listener>

エンティティリスナを指定します。

指定できる属性を次の表に示します。

表 12-53　<entity-listener>の属性

属性名	型	任意／必須	説明
class	xsd:string	必須	エンティティリスナのクラス名。

(3)　<pre-persist>，<post-persist>，<pre-remove>，<post-remove>，<pre-update>，<post-update>，<post-load>

ライフサイクルコールバックメソッドを指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-54　<pre-persist>，<post-persist>，<pre-remove>，<post-remove>，<pre-update>，<post-update>，<post-load>の属性

属性名	型	任意／必須	説明
method-name	xsd:string	必須	対象のメソッド名。

(4)　<unique-constraint>

DDL にユニーク制約を付加します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(5)　<column-name>

ユニーク制約を付加するカラムの名前を指定します。

@UniqueConstraint の columnNames 属性と対応しています。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(6)　<query>

クエリ文字列を指定します。

@NamedQuery の query 属性および@NamedNativeQuery の query 属性と対応しています。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(7)　<hint>

クエリにヒントを付加します。

@NamedQuery の hints 属性および@NamedNativeQuery の hints 属性と対応しています。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-55　<hint>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.48　@QueryHint」の name 属性を参照してください。
value	xsd:string	必須	「8.22.48　@QueryHint」の value 属性を参照してください。

(8)　<entity-result>

ネイティブ SQL のクエリ結果をマッピングするための Entity クラスを指定します。

@SqlResultSetMapping の entities 属性と対応しています。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–56　<entity-result>の属性

属性名	型	任意／必須	説明
entity-class	xsd:string	必須	「8.22.15　@EntityResult」の entityClass 属性を参照してください。
discriminator-column	xsd:string	任意	「8.22.15　@EntityResult」の discriminatorColumn 属性を参照してください。

(9)　<field-result>

ネイティブ SQL のクエリ結果をマッピングするためのフィールドを指定します。

@EntityResult の fields 属性と対応しています。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–57　<field-result>の属性

型	属性名	任意／必須	説明
xsd:string	name	必須	「8.22.19　@FieldResult」の name 属性を参照してください。
xsd:string	column	必須	「8.22.19　@FieldResult」の column 属性を参照してください。

(10)　<column-result>

ネイティブ SQL のクエリ結果をマッピングするためのカラムを指定します。

@SqlResultSetMapping の columns 属性と対応しています。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–58　<column-result>の属性

型	属性名	任意／必須	説明
xsd:string	name	必須	「8.22.7　@ColumnResult」の name 属性を参照してください。

(11) <attributes>

attributes 要素自体の機能はありません。

(12) <column>

Persistent フィールドまたはプロパティに，カラムのマッピングを指定します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-59 <column>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.6 @Column」の name 属性を参照してください。
unique	xsd:boolean	任意	「8.22.6 @Column」の unique 属性を参照してください。
nullable	xsd:boolean	任意	「8.22.6 @Column」の nullable 属性を参照してください。
insertable	xsd:boolean	任意	「8.22.6 @Column」の insertable 属性を参照してください。
updatable	xsd:boolean	任意	「8.22.6 @Column」の updatable 属性を参照してください。
column-definition	xsd:string	任意	「8.22.6 @Column」の columnDefinition 属性を参照してください。
table	xsd:string	任意	「8.22.6 @Column」の table 属性を参照してください。
length	xsd:int	任意	「8.22.6 @Column」の length 属性を参照してください。
precision	xsd:int	任意	「8.22.6 @Column」の precision 属性を参照してください。
scale	xsd:int	任意	「8.22.6 @Column」の scale 属性を参照してください。

(13) <generated-value>

プライマリキー値生成戦術を指定します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-60　<generated-value>の属性

属性名	型	任意／必須	説明
strategy	orm:generation-type	任意	「8.22.20　@GeneratedValue」の strategy 属性を参照してください。
generator	xsd:string	任意	「8.22.20　@GeneratedValue」の generator 属性を参照してください。

(14)　<temporal>

DATE，TIME，TIMESTAMP 型へマッピングを行うときに指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(15)　<table-generator>

テーブルジェネレータを追加します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-61　<table-generator>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.55　@TableGenerator」の name 属性を参照してください。
table	xsd:string	任意	「8.22.55　@TableGenerator」の table 属性を参照してください。
catalog	xsd:string	任意	「8.22.55　@TableGenerator」の catalog 属性を参照してください。
schema	xsd:string	任意	「8.22.55　@TableGenerator」の schema 属性を参照してください。
pk-column-name	xsd:string	任意	「8.22.55　@TableGenerator」の pkColumnName 属性を参照してください。
value-column-name	xsd:string	任意	「8.22.55　@TableGenerator」の valueColumnName 属性を参照してください。
pk-column-value	xsd:string	任意	「8.22.55　@TableGenerator」の pkColumnValue 属性を参照してください。

属性名	型	任意／必須	説明
initial-value	xsd:int	任意	「8.22.55　@TableGenerator」の initialValue 属性を参照してください。
allocation-size	xsd:int	任意	「8.22.55　@TableGenerator」の allocationSize 属性を参照してください。

(16)　<attribute-override>

プロパティやフィールドのマッピングを上書きします。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-62　<attribute-override>の属性

属性名	型	任意／必須	説明
name	xsd:string	必須	「8.22.3　@AttributeOverride」の name 属性を参照してください。

(17)　<lob>

Lob 型へマッピングを行うときに指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(18)　<enumerated>

列挙型へマッピングを行うときに指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(19)　<join-column>

テーブルを結合するために，所有者側エンティティに対応する結合表の外部キーカラムを指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-63　<join-column>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.24　@JoinColumn」の name 属性を参照してください。
referenced-column-name	xsd:string	任意	「8.22.24　@JoinColumn」の referencedColumnName 属性を参照してください。
unique	xsd:boolean	任意	「8.22.24　@JoinColumn」の unique 属性を参照してください。
nullable	xsd:boolean	任意	「8.22.24　@JoinColumn」の nullable 属性を参照してください。
insertable	xsd:boolean	任意	「8.22.24　@JoinColumn」の insertable 属性を参照してください。
updatable	xsd:boolean	任意	「8.22.24　@JoinColumn」の updatable 属性を参照してください。
column-definition	xsd:string	任意	「8.22.24　@JoinColumn」の columnDefinition 属性を参照してください。
table	xsd:string	任意	「8.22.24　@JoinColumn」の table 属性を参照してください。

(20)　<join-table>

多対多（many-to-many）や片方向の一対多（one-to-many）で使用する結合表を指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-64　<join-table>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.26　@JoinTable」の name 属性を参照してください。
catalog	xsd:string	任意	「8.22.26　@JoinTable」の catalog 属性を参照してください。
schema	xsd:string	任意	「8.22.26　@JoinTable」の schema 属性を参照してください。

(21)　<inverse-join-column>

テーブルを結合するために，被所有者側エンティティに対応する結合表の外部キーカラムを指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12-65　<inverse-join-column>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.24　@JoinColumn」の name 属性を参照してください。
referenced-column-name	xsd:string	任意	「8.22.24　@JoinColumn」の referencedColumnName 属性を参照してください。
unique	xsd:boolean	任意	「8.22.24　@JoinColumn」の unique 属性を参照してください。
nullable	xsd:boolean	任意	「8.22.24　@JoinColumn」の nullable 属性を参照してください。
insertable	xsd:boolean	任意	「8.22.24　@JoinColumn」insertable 属性を参照してください。
updatable	xsd:boolean	任意	「8.22.24　@JoinColumn」の updatable 属性を参照してください。
column-definition	xsd:string	任意	「8.22.24　@JoinColumn」の columnDefinition 属性を参照してください。
table	xsd:string	任意	「8.22.24　@JoinColumn」の table 属性を参照してください。

(22)　<cascade>

カスケードできる操作を指定します。

機能および属性の詳細については，「8.22　javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(23)　<cascade-all>

すべての操作をカスケードします。

(24)　<cascade-persist>

persist 操作をカスケードします。

(25)　<cascade-merge>

merge 操作をカスケードします。

(26) <cascade-remove>

remove 操作をカスケードします。

(27) <cascade-refresh>

refresh 操作をカスケードします。

(28) <order-by>

関連をコレクションに保持する際に付ける順序を指定します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

(29) <map-key>

Map 型の関連としてマップキーを指定します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–66 <map-key>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.30 @MapKey」の name 属性を参照してください。

(30) <primary-key-join-column>

ほかのテーブルに JOIN する外部キーとして使われる，プライマリキーカラムを指定します。

機能および属性の詳細については，「8.22 javax.persistence パッケージに含まれるアノテーションのサポート範囲」を参照してください。

指定できる属性を次の表に示します。

表 12–67 <primary-key-join-column>の属性

属性名	型	任意／必須	説明
name	xsd:string	任意	「8.22.46 @PrimaryKeyJoinColumn」の name 属性を参照してください。

属性名	型	任意／必須	説明
referenced-column-name	xsd:string	任意	「8.22.46　@PrimaryKeyJoinColumn」の referencedColumnName 属性を参照してください。
column-definition	xsd:string	任意	「8.22.46　@PrimaryKeyJoinColumn」の columnDefinition 属性を参照してください。

12.4　クエリヒント

CJPA プロバイダでは，O/R マッピングファイルの named-query 要素の下位要素である hint 要素にクエリヒントを指定できます。なお，クエリヒントは，@NamedQuery アノテーションの引数の@Hint アノテーションにも指定できます。アノテーションに指定するクエリヒントについては，「8.22.48 @QueryHint」を参照してください。

O/R マッピングファイルの named-query 要素の下位要素である hint 要素に指定できるクエリヒントを次の表に示します。

表 12–68　CJPA プロバイダで使用できるクエリヒント

キー名称	説明	指定可能値	デフォルト
cosminexus.jpa.pessimistic-lock	悲観的ロックを使用するかどうかを指定します。	NoLock 悲観的ロックを使用しません。 Lock 悲観的ロックを使用します。 対象となるテーブルがすでにロックされている場合，解放されるまで待ちます。 • Oracle の場合 SELECT … FOR UPDATE を発行します。 • HiRDB の場合 SELECT … WITH EXCLUSIVE LOCK を発行します。 LockNoWait 悲観的ロックを使用します。 対象となるテーブルがすでにロックされている場合，例外が発生します。 • Oracle の場合 SELECT … FOR UPDATE NO WAIT を発行します。 • HiRDB の場合 SELECT … WITH EXCLUSIVE LOCK NO WAIT を発行します。	NoLock

注　指定できるデータ型は String です。

注意事項

O/R マッピングファイルに指定したクエリヒントに，指定可能範囲外の値が設定された場合は，アプリケーションを開始したタイミングで例外が発生します。なお，値の大文字と小文字は区別しません。

13 Web サーバ連携で使用するファイル

この章では，Web サーバ連携で使用するファイルの形式，格納先，機能，指定できるキーなどについて説明します。

13.1　Web サーバ連携で使用するファイルの一覧

Web サーバ連携で使用するファイルの一覧を，次の表に示します。

表 13–1　Web サーバ連携で使用するファイルの一覧

ファイル名	分類	概要	参照先
isapi_redirect.conf	Microsoft IIS 用リダイレクタ動作定義ファイル	Microsoft IIS 用リダイレクタの動作を定義します。	13.2.1
mod_jk.conf	HTTP Server 用リダイレクタ動作定義ファイル	HTTP Server 用リダイレクタの動作を定義します。	13.2.2
uriworkermap.properties	Microsoft IIS 用マッピング定義ファイル	Microsoft IIS へのリクエストでどの URL パターンが Web コンテナに転送されるかを定義します。	13.2.3
workers.properties	ワーカ定義ファイル	リダイレクタの動作を定義します。	13.2.4

13.2 Web サーバ連携で使用するファイルの詳細

13.2.1 isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）

(1) 形式

次のようにキーを指定します。

```
〈キー名称〉=〈値〉
```

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。
- 値が存在しない行を定義した場合，その行は無視されます。キー名称として定義されていないパラメタを定義しても無視されます。
- 「キー名称=値」として 1023 文字までが有効となります。超えた部分は切り捨てられます。

(2) ファイルの格納先

<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.conf

(3) 機能

Microsoft IIS 用リダイレクタの動作パラメタを定義します。

(4) 指定できるキー

指定できるキーとデフォルト値を次に示します。このキーに不正な値を指定した場合，動作は保証されません。なお，複数のキーを指定した場合，最後に指定した値が有効になります。

キー名称	内容	デフォルト値
connection_sharing	ワーカ間でコネクションを共有するかどうかを指定します。 true を指定した場合： 同一ホスト，同一ポートへ接続するワーカは，ワーカ定義が異なってもコネクションを共有します。 false を指定した場合： ワーカ定義が異なるワーカ間のコネクションは共有しません。	true
connect_timeout	リクエスト送信時の Web コンテナに対するコネクション確立のタイムアウト値を 0～3600 の整数（単位：秒）で指定します。	30

キー名称	内容	デフォルト値
	数値以外の文字列や範囲外の数値を指定した場合は，メッセージを出力し，デフォルト値を使用します。 タイムアウト値に 0，または TCP の持つデータ送信の再送タイマより長い時間を設定した場合，TCP の持つタイムアウト値になります。その場合，不正なタイムアウト値が指定されたことを示すメッセージは出力されません。	
filter_priority	リダイレクタを ISAPI フィルタとして登録する際の，優先順序を指定します。 指定できる文字列を次に示します。 • high（優先順序を「高」にする） • medium（優先順序を「中」にする） • low（優先順序を「低」にする）	high
gateway_host	ゲートウェイのホスト名または IP アドレスを指定します。 Host ヘッダのないリクエストを welcome ファイルなどにリダイレクトする場合，Location ヘッダに指定する URL のホスト名部分が指定値になります。	なし
gateway_https_scheme	クライアントからのリクエストのスキームが https であり，かつ SSL アクセラレータなどを介すことで Web サーバへのスキームが http となる場合に，true を指定します。 true を指定した場合，Web サーバへのリクエストのスキームが https であるとみなされます。false を指定した場合は，何も実行されません。	false
gateway_port	ゲートウェイのポート番号を指定します。Host ヘッダのないリクエストに対して，welcome ファイルなどにリダイレクトするとき，Location ヘッダに指定する URL のポート番号部分が指定値になります。このパラメタを指定した場合は，gateway_host も必ず指定してください。 gateway_host を指定してこのパラメタを省略した場合，http によるアクセスの場合は 80，https によるアクセスの場合は 443 が使用されます。	なし
log_file_dir	ログファイルの出力先ディレクトリを指定します。 相対パスで指定した場合： 　<Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のディレクトリ名を指します。 絶対パスで指定した場合： 　記述されたディレクトリ名を指します。 なお，出力先として指定したディレクトリのアクセス権に Users グループに対する書き込み権限を設定する必要があります。アクセス権を設定していない場合，ログファイルが出力されません。 log_file_prefix と trace_log_file_prefix に同じ値を指定した場合は，このキーに trace_log_file_dir とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	logs
log_file_num	リダイレクタ用ログファイルの最大数を指定します。この数を超えると古いログファイルは上書きされます。 1～64 の整数値を指定します。	5
log_file_prefix	ログファイル名のプリフィックスです。実際のログファイル名は，このキーの指定値に<通番>.log が付けられたものになります。	isapi_redirect

キー名称	内容	デフォルト値
	log_file_dir と trace_log_file_dir に同じ値を指定した場合，このキーに trace_log_file_prefix とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	
log_file_size	リダイレクタ用ログファイルの一つ当たりのサイズをバイト単位で指定します。 4096～16777216 の整数値を指定します。	4194304
log_level	リダイレクタ用ログファイルの出力レベルを指定します。指定するログレベルを一つだけ指定します。 debug，info，error を指定できます。	error
prf_id	PRF デーモン起動時に PRF 識別子に指定した文字列を指定します。	PRF_ID
receive_client_timeout	クライアントから POST データを受信するときのタイムアウト時間を秒単位で指定します。 60～3600 の整数値を 60（秒）の倍数で指定します。 指定された値が 60 の倍数になっていない場合は，60 の倍数に切り上げた値がタイムアウト時間になります。	300
request_retry_count	リクエスト送信時の Web コンテナに対するコネクション確立，およびリクエスト送信のリトライ回数を，1～16 の整数（単位：回）で指定します。 リトライ回数には，初回のコネクション確立およびリクエスト送信処理も含まれます。 タイムアウトが発生した場合，リトライするケースは次のとおりです。 • コネクション確立時にタイムアウトした場合 • リクエストヘッダ送信時にタイムアウトした場合 上記処理以降の，リクエストボディの送信時にタイムアウトが発生した場合は，リトライは行いません。 リトライ回数に，範囲外の値や，整数値でない値などの異常値を設定した場合，デフォルト値が設定されます。	3
send_timeout	リクエスト送信のタイムアウト値を 0～3600 の整数（単位：秒）で指定します。 数値以外の文字列や範囲外の数値を指定した場合は，メッセージを出力し，デフォルト値を使用します。 タイムアウト値に 0，または TCP の持つデータ送信の再送タイマより長い時間を設定した場合，TCP の持つタイムアウト値になります。その場合，不正なタイムアウト値が指定されたことを示すメッセージは出力されません。	100
trace_log	リダイレクタの保守用トレースログを出力するかどうかを指定します。出力する場合は true を，出力しない場合は false を指定します。	true
trace_log_file_dir	保守用トレースログファイルの出力先ディレクトリを指定します。 相対パスで記述した場合： 　<Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のディレクトリ名を指します。 絶対パスで記述した場合： 　記述されたディレクトリ名を指します。	logs

キー名称	内容	デフォルト値
	なお，出力先として指定したディレクトリのアクセス権に Users グループに対する書き込み権限を設定する必要があります※。アクセス権を設定していない場合，ログファイルが出力されません。 log_file_prefix と trace_log_file_prefix に同じ値を指定した場合は，このキーに log_file_dir とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	
trace_log_file_num	保守用トレースログファイルの最大数を指定します。この数を超えると古いログファイルは上書きされます。 1～64 の整数値を指定します。	4
trace_log_file_prefix	保守用トレースログファイル名のプリフィックスを指定します。実際のログファイル名は，このキーの指定値に<通番>.log が付加されたものになります。 log_file_dir と trace_log_file_dir に同じ値を指定した場合は，このキーに log_file_prefix とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	iis_rd_trace
trace_log_file_size	保守用トレースログファイルの一つ当たりのサイズをバイト単位で指定します。 4096～16777216 の整数値を指定します。	16777216
worker_file	ワーカ定義ファイルの位置とファイル名を指定します。 相対パスで指定した場合： 　<Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のファイル名を指します。 絶対パスで指定した場合： 　記述されたファイル名を指します。	workers.properties
worker_mount_file	マッピング定義ファイルの位置とファイル名を指定します。 相対パスで指定した場合： 　<Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のファイル名を指します。 絶対パスで指定した場合： 　記述されたファイル名を指します。	uriworkermap.properties

注※　Microsoft IIS と連携する場合

新規インストール時には，デフォルトのログ出力先ディレクトリは存在しません。ディレクトリを作成してアクセス権を設定するか，一つ上のディレクトリ redirector へアクセス権を設定してください。

また，リダイレクタのログ出力先ディレクトリを変更し，そのパスが途中までしか存在しない場合，存在する最下層のディレクトリに対してアクセス権を設定するか，指定したパスをすべて作成し，アクセス権を設定してください。

(5)　記述例

```
gateway_host=hostA
gateway_https_scheme=true
gateway_port=443

log_level=error
log_file_size=4194304
log_file_num=5
```

```
log_file_dir=logs
log_file_prefix=isapi_redirect
prf_id=prfid
trace_log=true
trace_log_file_size=16777216
trace_log_file_num=4
trace_log_file_dir=logs
trace_log_file_prefix=iis_rd_trace
receive_client_timeout=300
worker_file=workers.properties
worker_mount_file=uriworkermap.properties
```

(6) 注意事項

このファイルを編集してリダイレクタのユーザ定義を変更した場合，Web サーバを再起動する必要があります。変更した定義は，Web サーバを再起動したあとに反映されます。

13.2.2 mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）

(1) 形式

次のようにキーを指定します。

```
〈キー名称〉〈値〉
```

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。
- キー名称と値は半角スペースで区切ります。また，値を複数指定する場合も半角スペースで区切ります。
- 指定する値がファイルパスでスペースを含む場合は，パス全体を""（ダブルクォーテーション）で囲む必要があります。
- コメントの記述方法，および記述できる文字の種類などの記述形式は，HTTP Server の仕様に従います。
- キー名称の大文字小文字は区別しません。

(2) ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥redirector¥mod_jk.conf
- UNIX の場合
  /opt/Cosminexus/CC/web/redirector/mod_jk.conf

(3) 機能

リダイレクタの動作を定義します。

(4) 指定できるキー

- モジュール定義

  Web コンテナと HTTP Server 間の通信を処理するためにライブラリを定義します。

  形式

  `LoadModule jk_module <ライブラリ名>`

  ライブラリ名を絶対パスで指定します。なお，複数指定はできません。

  指定例

  Windows の場合

  LoadModule jk_module "<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥mod_jk.dll"

  UNIX の場合

  LoadModule jk_module /opt/Cosminexus/CC/web/redirector/mod_jk.so

  注意事項

  モジュール定義を指定するキーは，ほかのキーよりも前に記述する必要があります。

- マッピング定義

  HTTP Server へのリクエストでどの URL パターンが Web コンテナに転送されるかを定義します。

  形式

  `JkMount <URLパターン> <ワーカ名>`

  workers.properties の worker.list で指定されているワーカのどれかを記述します。URL パターンとワーカ名の組み合わせを複数記述できます。このファイルに不正な値を設定した場合，動作は保証されません。また，先頭がスラッシュ（/）から始まらない URL パターンを指定した場合，リダイレクタ起動時にエラーメッセージがログに出力され，指定した内容は無視されます。

  拡張子の長さが 0 文字の拡張子指定の URL パターンを指定した場合（URL パターンの末尾が「/*.」の場合），メッセージ KDJE41041-W がログに出力され，指定した内容は無視されます。

  JkMount キーについては，「5.4 ラウンドロビン方式によるリクエストの振り分け」，および「5.5 POST データサイズでのリクエストの振り分け」を参照してください。

- リダイレクタ定義

  次に示すキーを指定できます。ただし，このキーに不正な値を指定した場合，動作は保証されません。

  なお，「関連情報」とは，指定したキーに関する情報の参照先です。マニュアル名称の「アプリケーションサーバ」を省略しています。

キー名称	内容	デフォルト値	関連情報
JkConnectTimeout	リクエスト送信時の Web コンテナに対するコネクション確立のタイムアウト値	30	

キー名称	内容	デフォルト値	関連情報
	を，0～3600 の整数（単位：秒）で指定します。 数値以外の文字列や範囲外の数値を指定した場合は，メッセージを出力し，デフォルト値を使用します。 タイムアウト値に 0，または TCP の持つデータ送信の再送タイマより長い時間を設定した場合，TCP の持つタイムアウト値になります。その場合，不正なタイムアウト値が指定されたことを示すメッセージは出力されません。		
JkGatewayHost	ゲートウェイのホスト名または IP アドレスを指定します。 Host ヘッダのないリクエストに対して welcome ファイルなどへリダイレクトする場合，Location ヘッダに指定する URL のホスト名部分が指定値になります。	なし	「5.10　Web コンテナへのゲートウェイ情報の通知」
JkGatewayHttpsScheme	クライアントからのリクエストのスキームが https であり，かつ SSL アクセラレータなどを介すことで Web サーバへのスキームが http となる場合に，On を指定します。 On を指定した場合，Web サーバへのリクエストのスキームが https であるとみなされます。Off を指定した場合，何も実行されません。	Off	「5.10　Web コンテナへのゲートウェイ情報の通知」
JkGatewayPort	ゲートウェイのポート番号を指定します。Host ヘッダのないリクエストを welcome ファイルなどにリダイレクトする場合，Location ヘッダに指定する URL のポート番号部分が指定値となります。 このパラメタを指定した場合は，必ず JkGatewayHost も指定してください。 JkGatewayHost を指定してこのパラメタを省略した場合，http によるアクセスであれば 80，https によるアクセスであれば 443 が使用されます。	なし	「5.10　Web コンテナへのゲートウェイ情報の通知」
JkLogFileDir	ログファイルの出力先ディレクトリを指定します。Windows の場合，相対パスまたは絶対パスで指定してください。UNIX の場合，絶対パスで指定してください。 相対パスで指定した場合： <Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のディレクトリ名を指します。	• Windows の場合 logs • UNIX の場合 /opt/Cosminexus/CC/web/	

キー名称	内容	デフォルト値	関連情報
	絶対パスで指定した場合： 記述されたディレクトリ名を指します。 なお，出力先として指定したディレクトリのアクセス権に HTTP Server の実行アカウントの書き込み権限を設定する必要があります※1。アクセス権を設定していない場合，ログファイルが出力されません。 JkLogFilePrefix と JkTraceLogFilePrefix に同じ値を指定した場合は，このキーに JkTraceLogFileDir とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	redirector/logs	
JkLogFileNum	リダイレクタ用ログファイルの最大数を指定します。この値を超えると古いログファイルは上書きされます。次の範囲の整数値を指定してください。 • Windows の場合 1≦JkLogFileNum≦16 • UNIX の場合 1≦JkLogFileNum≦64	5	
JkLogFilePrefix	ログファイル名のプリフィックスです。実際のログファイル名は，指定値に<通番>.log が付加されたものとなります。 JkLogFileDir と JkTraceLogFileDir に同じ値を指定した場合は，このキーに JkTraceLogFilePrefix とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	hws_redirect	
JkLogFileSize	リダイレクタ用ログファイルの一つ当たりのサイズ（バイト）を指定します。次の範囲の整数値を指定してください。 • Windows の場合 4096≦JkLogFileSize≦2147483647 • UNIX の場合 4096≦JkLogFileSize≦16777216	4194304	
JkLogLevel	リダイレクタ用ログファイルのメッセージログ，および保守用トレースログのログ出力レベルを指定します。指定するログレベルを一つだけ指定します。debug，info，error（デフォルト値），および emerg を指定できます。なお，emerg は，Windows の場合だけ指定できる値です。	error	

キー名称	内容	デフォルト値	関連情報
	これら以外の値を指定した場合は，errorが指定されたものとして動作します。		
JkModulePriority	HTTP Server へリダイレクタ以外の外部モジュールを登録する場合に，ほかの外部モジュールに対するリダイレクタの実行順位を指定します。 指定できる値を次に示します。 • -10～30 の整数 • REALLY_FIRST（整数値の-10 に対応） • FIRST（整数値の 0 に対応） • MIDDLE（整数値の 10 に対応） • LAST（整数値の 20 に対応） • REALLY_LAST（整数値の 30 に対応） 実行順位は指定する値が小さい程高くなります。	FIRST	
JkOptions	リクエスト URL の URL デコードを行うかどうかを指定します。なお，このキーは UNIX 用です。 ForwardURICompatUnparsed（デフォルト）： リクエスト URL の URL デコードを行いません。 ForwardURICompat： リクエスト URL の URL デコードを行います。 02-00 では URL デコードを行っていたため，URL デコードによって変換される文字列を含む URL を使用する場合，02-00 と同様に URL デコードを行う必要があるときだけ`ForwardURICompat` を指定してください。	ForwardURICompatUnparsed	
JkPrfId	PRF デーモン起動時に PRF 識別子に指定した文字列を指定します。	PRF_ID	
JkRequestRetryCount	リクエスト送信時の Web コンテナに対するコネクション確立，およびリクエスト送信のリトライ回数を，1～16 の整数（単位：回）で指定します。 リトライ回数には，初回のコネクション確立およびリクエスト送信処理も含まれます。 タイムアウトが発生した場合，リトライするケースは次のとおりです。	3	

キー名称	内容	デフォルト値	関連情報
	• コネクション確立時にタイムアウトした場合 • リクエストヘッダ送信時にタイムアウトした場合 上記処理以降の，リクエストボディの送信時にタイムアウトが発生した場合は，リトライは行いません。 リトライ回数に，範囲外の値や，整数値でない値などの異常値を設定した場合，デフォルト値が設定されます。		
JkSendTimeout	リクエスト送信のタイムアウト値を 0～3600 の整数（単位：秒）で指定します。 数値以外の文字列や範囲外の数値を指定した場合は，メッセージを出力し，デフォルト値を使用します。 タイムアウト値に 0，または TCP の持つデータ送信の再送タイマより長い時間を設定した場合，TCP の持つタイムアウト値になります。その場合，不正なタイムアウト値が指定されたことを示すメッセージは出力されません。	100	
JkTraceLog	リダイレクタの保守用トレースログを出力するかどうかを指定します。出力する場合は On（デフォルト値）を，出力しない場合は Off を指定します。	On	
JkTraceLogFileDir	保守用トレースログファイルの出力先ディレクトリを指定します。Windows の場合，相対パスまたは絶対パスで指定してください。UNIX の場合，絶対パスで指定してください。 相対パスで記述した場合： <Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のディレクトリ名を指します。 絶対パスで記述した場合： 記述されたディレクトリ名を指します。 JkLogFilePrefix と JkTraceLogFilePrefix に同じ値を指定した場合は，このキーに JkLogFileDir とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	• Windows の場合 logs • UNIX の場合 /opt/Cosminexus/CC/web/redirector/logs	
JkTraceLogFileNum	保守用トレースログファイルの最大数を整数値で指定します。 • Windows の場合	4	

キー名称	内容	デフォルト値	関連情報
	1～16 • UNIX の場合 1～64 この数を超えると古いログファイルは上書きされます。		
JkTraceLogFilePrefix	保守用トレースログファイル名のプリフィックスを指定します。実際のログファイル名は，このキーの指定値に<通番>.log が付加されたものとなります。 JkLogFileDir と JkTraceLogFileDir に同じ値を指定した場合は，このキーに JkLogFilePrefix とは異なる値を指定する必要があります。同じ値が指定された場合，リダイレクタは動作しません。	hws_rd_trace	
JkTraceLogFileSize	保守用トレースログファイルの一つ当たりのサイズを整数値（単位：バイト）で指定します。 • Windows の場合 4096～2147483647 • UNIX の場合 4096～16777216	16777216	
JkTranslateBackcompat	05-05 以前のバージョンの互換用のキーです。 HTTP Server へのリクエストが Web コンテナに転送させる URL パターンであった場合，リダイレクタよりあとに実行されるモジュールの translate_handler 関数が HTTP Server から呼び出されるようにするかどうかを指定します。 On を指定した場合，リダイレクタの次に実行されるモジュールの translate_handler 関数が呼び出されます。 Off を指定した場合，リダイレクタよりあとに実行されるモジュールの translate_handler 関数は呼び出されません。	Off	
JkWorkersFile	ワーカ定義ファイルのファイル名を指定します。Windows の場合，相対パスまたは絶対パスで指定してください。UNIX の場合，絶対パスで指定してください。 相対パスで指定した場合： <Application Server のインストールディレクトリ>¥CC¥web¥redirector 以下のファイル名を指します。	• Windows の場合 workers.properties • UNIX の場合 /opt/Cosminex	

キー名称	内容	デフォルト値	関連情報
	絶対パスで指定した場合： 記述されたファイル名を指します。	us/CC/web/redirector/workers.properties※2	

注

複数のキーを指定した場合，最後に指定した値が有効になります。

注※1　HTTP Server と連携する場合

新規インストール時には，デフォルトのログ出力先ディレクトリは存在しません。ディレクトリを作成してアクセス権を設定するか，一つ上のディレクトリ redirector にアクセス権を設定してください。

また，リダイレクタのログ出力先ディレクトリを変更し，そのパスが途中までしか存在しない場合，存在する最下層のディレクトリに対してアクセス権を設定するか，指定したパスをすべて作成し，アクセス権を設定してください。

注※2

/opt/Cosminexus/CC/web/redirector/workers.properties が存在しない場合，HTTP Server は起動しません。

(5)　記述例

- Windows の場合

```
LoadModule jk_module "C:¥Program Files¥Hitachi¥Cosminexus¥CC¥web¥redirector¥mod_jk.dll"※

JkGatewayHost hostA
JkGatewayHttpsScheme On
JkGatewayPort 443

JkLogLevel error
JkLogFileSize 4194304
JkLogFileNum 5
JkLogFileDir logs
JkLogFilePrefix hws_redirect
JkPrfId prfid
JkTraceLog On
JkTraceLogFileSize 16777216
JkTraceLogFileNum 4
JkTraceLogFileDir logs
JkTraceLogFilePrefix hws_rd_trace
JkTranslateBackcompat Off
JkWorkersFile workers.properties

JkMount /examples/* worker1
```

注※

LoadModule の指定は，ファイル上では 1 行で記述してください。

- UNIX の場合

```
LoadModule jk_module /opt/Cosminexus/CC/web/redirector/mod_jk.so
JkWorkersFile /opt/Cosminexus/CC/web/redirector/workers.properties
```

```
JkLogLevel error
JkLogFileSize 4194304
JkLogFileNum 5
JkLogFileDir /opt/Cosminexus/CC/web/redirector/logs
JkLogFilePrefix hws_redirect

JkTraceLog On
JkTraceLogFileSize 16777216
JkTraceLogFileNum 4
JkTraceLogFileDir /opt/Cosminexus/CC/web/redirector/logs
JkTraceLogFilePrefix hws_rd_trace

JkMount /examples/* worker1
```

(6) 注意事項

- Windows の場合，リダイレクタのユーザ定義を変更するには，Web サーバを再起動する必要があります。変更した定義は，Web サーバを再起動したあとに反映されます。
- UNIX の場合，リダイレクタのユーザ定義を変更して，変更内容を反映させるには，次の操作が必要です。

ファイルサイズまたはファイル面数を変更する場合

1. Web サーバを停止します。
2. ログファイルおよび HNTRLib が使用する管理ファイルを，移動または削除します。
   ＜ HNTRLib が使用する管理ファイル＞
   メッセージログファイルの場合：<JkLogFilePrefix の設定値>.mm
   保守用トレースログファイルの場合：<JkTraceLogFilePrefix の設定値>.mm
3. Web サーバを起動します。

なお，HNTRLib が使用する管理ファイルのデフォルトの格納場所は次のとおりです。

<Application Server のインストールディレクトリ>/CC/web/redirector/logs/mmap

ファイルサイズまたはファイル面数を変更しない場合

Web サーバを再起動します。

13.2.3 uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）

(1) 形式

次のようにキーを指定します。

```
〈キー名称〉 = 〈値〉
```

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。
- 値が存在しない行を定義した場合，その行は無視されます。キー名称として定義されていないパラメタを定義しても無視されます。
- 「キー名称=値」として 1023 文字までが有効となります。超えた部分は切り捨てられます。

(2) ファイルの格納先

<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥uriworkermap.properties

(3) 機能

uriworkermap.properties は，Microsoft IIS へのリクエストでどの URL パターンが Web コンテナに転送されるかを定義します。

(4) 指定できるキー

workers.properties の worker.list で指定されているワーカのどれかを記述します。URL パターンとワーカ名の組み合わせを複数記述できます。このファイルに不正な値を設定した場合，動作は保証されません。また，先頭がスラッシュ（/）から始まらない URL パターンを指定した場合，リダイレクタ起動時にエラーメッセージがログに出力され，指定した内容は無視されます。

拡張子の長さが 0 文字の拡張子指定の URL パターンを指定した場合（URL パターンの末尾が「/*.」の場合），メッセージ KDJE41041-W がログに出力され，指定した内容は無視されます。

```
<URLパターン> = <ワーカ名>
```

(5) 記述例

```
/examples/*=worker1
```

(6) 注意事項

リダイレクタのユーザ定義を変更した場合，Web サーバを再起動する必要があります。変更した定義は，Web サーバを再起動したあとに反映されます。

13.2.4　workers.properties（ワーカ定義ファイル）

(1)　形式

次のようにキーを指定します。

```
〈キー名称〉=〈値〉
```

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。
- 値が存在しない行を定義した場合，その行は無視されます。キー名称として定義されていないパラメタを定義しても無視されます。
- 「キー名称=値」として 1023 文字までが有効となります。超えた部分は切り捨てられます。
- ワーカ名に使用できる文字は半角英数字，アンダースコア「_」，およびハイフン「-」です。これら以外の文字を指定した場合の動作は保証しません。

(2)　ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥redirector¥workers.properties
- UNIX の場合
  /opt/Cosminexus/CC/web/redirector/workers.properties

(3)　機能

ワーカを定義し，ワーカごとにパラメタを設定して，リダイレクタの動作を定義します。

(4)　指定できるキー

ワーカ定義ファイルに指定できるキーと，ワーカごとの定義パラメタについて説明します。

(a)　ワーカ定義ファイルに指定できるキー

ワーカ，およびワーカごとの各パラメタを定義します。このキーに不正な値を設定した場合，動作は保証されません。

キー名称	内容	デフォルト値
worker.list	ワーカ名のリストを指定します。複数ある場合は，コンマ（,）で区切ります。一つ以上のワーカ名を必ず指定する必要があります。	なし

キー名称	内容	デフォルト値
	mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル），または uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）で指定するワーカ名を指定します。	
worker.<ワーカ名>.<パラメタ>	ワーカごとの定義パラメタを指定します。worker.list に記述されたワーカごとに設定します。 定義パラメタについては，「13.2.4(4)(b)　ワーカごとの定義パラメタ」を参照してください。	なし

(b)　ワーカごとの定義パラメタ

定義できるパラメタ	内容	デフォルト値	関連情報
worker.<ワーカ名>.balanced_workers	負荷分散の対象となるワーカのリストを指定します。複数ある場合は，コンマ（,）で区切ります。	なし	
worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカとのコネクション数を，1～2147483647 の整数で指定します。なお，このパラメタは Windows 用です。 ワーカとのコネクションは，この設定値以内の場合はリダイレクタ内に保持し，該当ワーカへの通信に再利用され，接続先の J2EE サーバが終了するまで解放しません。リクエストの多重度が設定値を超えた場合は，設定値を超えたリクエストだけをリクエスト単位にワーカとのコネクションの確立，解放を行います。 この値は次に示す式に従ってメモリを消費します。 （式） **メモリ消費量=（worker.<ワーカ名>.cachesize の値）×10KB**	64	
worker.<ワーカ名>.default_worker	デフォルトワーカのワーカ名を指定します。 POST リクエスト転送先ワーカに指定したワーカと同じワーカを指定した場合，指定したワーカには，POST データサイズによる振り分け条件を満たすリクエスト，およびデフォルトワーカとして条件を満たすリクエストが転送されます。 なお，このパラメタに POST リクエスト転送先ワーカに指定していないワーカを指定し，そのワーカに worker.<ワーカ名>.post_data が指定されている場合，worker.<ワーカ名>.post_data 定義は無視されます。 ワーカ名の前後の空白文字（スペース，タブ，およびフォームフィード）は無視されます。 このパラメタの指定を省略した場合や空文字列を指定した場合など，転送条件を満たすワーカが存在しないリクエストに対してエラーが返されます。	なし	

定義できるパラメタ	内容	デフォルト値	関連情報
worker.<ワーカ名>.delegate_error_code	エラーページの委任機能を利用するエラーステータスコードを指定します※1。複数指定する場合はコンマ（,）で区切って指定します。	なし	
worker.<ワーカ名>.host※2	ワーカのホスト名，または IP アドレスを指定します。	なし	
worker.<ワーカ名>.lbfactor	リクエストの負荷分散値を指定します。設定する値は 0 よりも大きくしてください。なお，値として小数値を指定することもできます。	1	
worker.<ワーカ名>.port※2	ワーカのポート番号を，1～65535 の整数で指定します。 すでにほかのアプリケーションで使用，または確保されているポート番号は指定できません。	なし	
worker.<ワーカ名>.post_data	<ワーカ名>に指定したワーカに転送するリクエストの Content-Length ヘッダの値の上限値に 1 を加えた値を次のように指定します。 • 1～2147483648 の整数（単位：バイト） • 1～2097152 の整数に「k」または「K」を付加した値（単位：キロバイト） • 1～2048 の整数に「m」または「M」を付加した値（単位：メガバイト） Content-Length ヘッダの値が指定値未満のリクエストを，<ワーカ名>に指定したワーカに転送します。 worker.<ワーカ名>.post_size_workers パラメタに複数のワーカを設定している場合，リクエストの Content-Length ヘッダの値が指定値未満，かつ指定値が最も小さいワーカにリクエストは転送されます。 worker.<POST リクエスト振り分けワーカのワーカ名>.post_size_workers パラメタで指定した，ほかのワーカと同じ値は設定しないでください。 値の前後の空白文字（スペース，タブ，フォームフィード）は無視されます。	なし	
worker.<ワーカ名>.post_size_workers	POST リクエスト転送先ワーカのワーカ名のリストを指定します。複数指定する場合は，コンマ(,）で区切って指定します。ただし，同じワーカ名は指定できません。 ワーカ名の前後の空白文字（スペース，タブ，フォームフィード）は無視されます。	なし	
worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。レスポンスデータを待つ時間を 0～3600 の整数値（単位：秒）で指定します。0 を指定した場合，レスポン	3600	「5.6　通信タイムアウト（Web サーバ連携）」

定義できるパラメタ	内容	デフォルト値	関連情報
	スを受け取るまで待ち続け，通信タイムアウトにはしません。		
worker.<ワーカ名>.type	ワーカのタイプを次に示すタイプから指定します。タイプごとの設定できるパラメタについては，「13.2.4(4)(c)　worker.<ワーカ名>.type ごとの定義パラメタ」を参照してください。なお，このパラメタはワーカごとに必ず指定する必要があります。 ajp13： 外部プロセスで動作している Web コンテナへリクエストを転送するワーカです。 ajp12： 旧バージョンとの互換性を保つためのワーカです。ajp13 が指定されたものとして動作します。 lb： ラウンドロビンに基づく負荷分散機能を持つワーカです。 post_size_lb： POST リクエスト振り分けワーカです。HTTP Server 使用時だけ指定できます。	なし	

注※1

指定できるコードがコメント文で記載されています。必要に応じてコメントを外してください。

注※2

複数のワーカで同一の Web コンテナのホスト名およびポート番号を指定することもできます。

(c)　worker.<ワーカ名>.type ごとの定義パラメタ

定義できるパラメタ	ワーカのタイプ		
	ajp13	lb	post_size_lb
worker.<ワーカ名>.balanced_workers	×	○	×
worker.<ワーカ名>.cachesize※1	△	×	×
worker.<ワーカ名>.default_worker	×	×	△
worker.<ワーカ名>.delegate_error_code	△	×	×
worker.<ワーカ名>.host	○	×	×
worker.<ワーカ名>.lbfactor	△	×	×
worker.<ワーカ名>.port	○	×	×
worker.<ワーカ名>.post_data	×/○※2	×	×
worker.<ワーカ名>.post_size_workers	×	×	○

定義できるパラメタ	ワーカのタイプ		
	ajp13	lb	post_size_lb
worker.<ワーカ名>.receive_timeout	△	×	×

（凡例）

○：必ず指定します。

△：任意に指定します。

×：指定できません。

注 1

ajp12 で指定できるパラメタは，ajp13 と同じです。

注 2

UNIX の場合，必ず指定する項目に値を指定しなかったとき，または指定した値が不正のとき，HTTP Server は起動しません。

注※1

Windows の場合だけ有効です。UNIX の場合，パラメタは無視されます。

注※2

POST リクエスト転送先ワーカでは必須です。

(5) 記述例

```
worker.list=worker1
worker.worker1.port=8007
worker.worker1.host=localhost
worker.worker1.type=ajp13
#worker.worker1.cachesize=64
#worker.worker1.receive_timeout=3600
#worker.worker1.delegate_error_code=400,401,402,403,404,405,406,407,408,409,410,411,412,413,
414,415,416,417,422,423,424,500,501,502,503,504,505,507,510

#----------------------------------------------------------------
# Example setting for Loadbalancer.
#----------------------------------------------------------------
#worker.list=loadbalancer1
#
#worker.loadbalancer1.type=lb
#worker.loadbalancer1.balanced_workers=worker1,worker2
#
#worker.worker1.port=8007
#worker.worker1.host=host1
#worker.worker1.type=ajp13
#worker.worker1.cachesize=64
#worker.worker1.lbfactor=1
#worker.worker1.receive_timeout=3600
#worker.worker1.delegate_error_code=400,401,402,403,404,405,406,407,408,409,410,411,412,413,
414,415,416,417,422,423,424,500,501,502,503,504,505,507,510

#
#worker.worker2.port=8007
#worker.worker2.host=host2
#worker.worker2.type=ajp13
#worker.worker2.cachesize=64
#worker.worker2.lbfactor=1
```

```
#worker.worker2.receive_timeout=3600
#worker.worker2.delegate_error_code=400,401,402,403,404,405,406,407,408,409,410,411,412,413,
414,415,416,417,422,423,424,500,501,502,503,504,505,507,510
#-------------------------------------------------------------
# Example setting for post data size based worker.
#-------------------------------------------------------------
#worker.list=postsizelb1#worker.postsizelb1.type=post_size_lb
#worker.postsizelb1.post_size_workers=worker1,worker2
#worker.postsizelb1.default_worker=worker1
#
#worker.worker1.port=8007
#worker.worker1.host=host1
#worker.worker1.type=ajp13
#worker.worker1.post_data=100m
#worker.worker1.receive_timeout=3600
#worker.worker1.delegate_error_code=400,401,402,403,404,405,406,407,408,409,410,411,412,413,
414,415,416,417,422,423,424,500,501,502,503,504,505,507,510
#
#worker.worker2.port=8007
#worker.worker2.host=host2
#worker.worker2.type=ajp13
#worker.worker2.post_data=2048m
#worker.worker2.receive_timeout=3600
#worker.worker2.delegate_error_code=400,401,402,403,404,405,406,407,408,409,410,411,412,413,
414,415,416,417,422,423,424,500,501,502,503,504,505,507,510
```

(6) 注意事項

このファイルを編集してリダイレクタのユーザ定義を変更した場合，Web サーバを再起動する必要があります。変更した定義は，Web サーバを再起動したあとに反映されます。

14 性能解析トレース

性能解析トレースは，クライアントからのリクエストを処理するときにアプリケーションサーバの各機能が出力する性能解析情報（トレース情報），およびアプリケーションの処理が出力する性能解析情報（トレース情報）を収集する機能です。

この情報を基に，システムおよびアプリケーションの処理性能を解析できます。この章では，性能解析トレースを使用して，システムおよびアプリケーションの性能を解析する方法について説明します。ここでは，V9 互換モードと推奨モードで差異がある項目について説明します。ここに記載がない場合は，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.　性能解析トレースを使用した性能解析」を参照してください。なお，性能解析トレースによって性能解析情報が取得されるポイント（トレース取得ポイント）や，情報の取得範囲（PRF トレース取得レベル）については，「14.5　性能解析トレースのトレース取得ポイントと PRF トレース取得レベルの概要」から「14.13　CDI のトレース取得ポイント」を参照してください。

14.1　性能解析トレースの概要

性能解析トレースの格納場所は次のとおりです。

- Windows の場合
  <環境変数 PRFSPOOL の設定ディレクトリ>¥utt¥prf¥<PRF 識別子>
- UNIX の場合
  $PRFSPOOL/utt/prf/<PRF 識別子>

性能解析トレースの収集方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.　性能解析トレースを使用した性能解析」を参照してください。性能解析トレースには，セッショントレースも出力されている場合があります。

14.2　アプリケーションサーバの性能解析トレースの概要

トレースを出力するリダイレクタ，Web コンテナなどを，**機能レイヤ**といいます。性能解析トレースでは，次の機能レイヤの入り口と出口でトレース情報を出力します。必要に応じて，各機能レイヤ内の処理のうち，性能に影響を与える処理ごとにも，トレース情報を出力します。アプリケーションの実行環境と該当する機能レイヤのうち，V9 互換モードと推奨モードで差異がある項目を次の表に示します。

表 14–1　アプリケーションの実行環境と該当する機能レイヤ

機能レイヤ	アプリケーションの実行環境	
	J2EE アプリケーションの実行環境	バッチアプリケーションの実行環境
リダイレクタ	○	－
CJPA プロバイダ	○	－

（凡例）○：該当する　－：該当しない

参考

- PRF トレースには，キー情報（ルートアプリケーション情報）が付与されます。J2EE アプリケーションの場合，リダイレクタまたは EJB クライアントで取得した情報になります。
- J2EE アプリケーションのトランザクションがタイムアウトした場合やリダイレクタでのレスポンス受信時にタイムアウトが発生した場合，性能解析トレースに出力されるルートアプリケーション情報を使用して，タイムアウトしたトランザクションやリクエストを特定できます。

14.3　性能解析トレースファイルの出力情報（性能解析トレースの場合）

性能解析トレースでは，各機能レイヤのトレース情報を収集します。

各機能レイヤが出力する情報のうち，V9 互換モードと推奨モードで差異がある項目を次の表に示します。

表 14–2　性能解析トレースファイルに出力する情報（性能解析トレースの場合）

トレース情報のヘッダ	説明	値の範囲
Thread	トレース情報を取得したプロセス内スレッドのスレッド ID，およびスレッドのハッシュ値※1。	スレッド ID：20 けた以内の 10 進数が出力されます。 ハッシュ値：10 けた以内の 10 進数が出力されます。
ProcessName	プロセス名。	32 文字以内のプロセスを表す文字列※2 が出力されます。
OPT※3	取得ポイントごとの付加情報。	514 文字以内の 16 進数値文字列が出力されます。

注※1

CTM およびリダイレクタで取得したトレース情報には，スレッドのハッシュ値は出力されない場合があります。

注※2

リダイレクタの場合，プロセス名称は次のように決定されます。

- リダイレクタの場合

  Web サーバとして HTTP Server を使用している場合は，"RD-{Web サーバ待ち受けポート番号}"がプロセス名称となります。Microsoft IIS を使用している場合は，"Redirector"がプロセス名称となります。

注※3

機能レイヤによっては，「OPT」に入り口時刻が出力されるトレース取得ポイントがあります。入り口時刻とは，そのトレース取得ポイントのトレースに対応する入り口トレースが出力された時間を示します。例えば，リダイレクタのトレース取得ポイントの場合，Web サーバへの HTTP レスポンスボディ情報の送信完了時（0x8104）のトレース取得ポイントで出力される入り口時刻は，Web コンテナからの HTTP レスポンスボディ情報の送信開始時（0x8004）の時刻になります。

なお，入り口時刻は，「1970 年 1 月 1 日 00:00:00」からの通算時間が 16 バイトの値で出力されます。前半の 8 バイトの値は秒単位を，後半の 8 バイトの値はマイクロ秒単位を示します。ただし，リダイレクタの性能解析トレースの場合は，後半の 8 バイトはミリ秒単位を示します。

14.4　Web サーバのレスポンスタイムの解析

アプリケーションサーバの処理性能は，クライアントからデータベースなどの EIS までの一連の処理，およびその処理結果がクライアントに返却されるまでのリクエストの一連の処理で，リダイレクタや Web コンテナなどの機能レイヤから出力されるトレース情報を基に解析できます。

ここでは，Web サーバがクライアントからリクエストを受けてから返却するまでに掛かった時間を解析する方法を，例を使用して説明します。

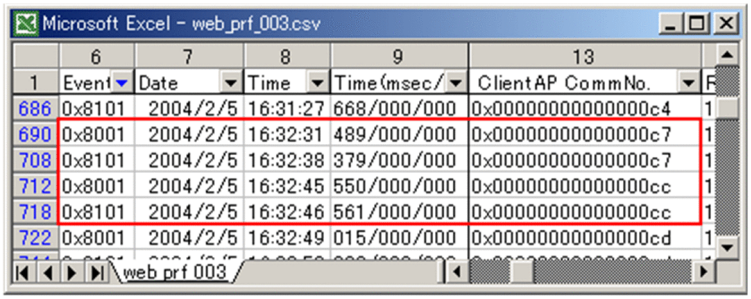
リダイレクタに割り当てられたイベント ID「0x8001」および「0x8101」をキーにして，Web サーバで収集した性能解析トレースファイルをフィルタリングします。フィルタリングに使用するイベント ID が示すトレース取得ポイントは次のとおりです。

表 14–3　フィルタリングに使用するイベント ID が示すトレース取得ポイント

イベント ID	トレース取得ポイント
0x8001	Web コンテナへのリクエストヘッダ情報の送信
0x8101	Web コンテナからのレスポンス完了通知の受信

イベント ID「0x8001」および「0x8101」をキーにして，性能解析トレースをフィルタリングした例を次に示します。

図 14–1　性能解析トレースファイルをフィルタリングした例



「0x8001」および「0x8101」のトレース取得時刻から，リクエストごとのレスポンスタイムを解析できます。図 14-1 の例では，クライアントアプリケーション情報の通信番号が「0x00000000000000c7」と「0x00000000000000cc」のリクエストのレスポンスタイムを比較すると，「0x00000000000000c7」のリクエストの方が処理に時間が掛かっていることがわかります。

14.4.1　タイムアウトが発生したリクエストの特定

ここでは，リダイレクタでのレスポンス受信時にタイムアウトが発生した場合に，性能解析トレースを利用してタイムアウトが発生したリクエストを特定する方法について説明します。

リダイレクタでのレスポンス受信時にタイムアウトが発生した場合，メッセージ KDJE41019-E が出力されます。このメッセージには，次の情報が出力されます。

- リクエストの URI
- Web サーバとの通信に使用する Web コンテナが動作するホストの IP アドレス
- Web サーバとの通信に使用する Web コンテナのポート番号
- Web サーバとの通信に使用する Web コンテナのファイルディスクリプタ
- 性能解析トレースのルートアプリケーション情報

メッセージに出力された性能解析トレースのルートアプリケーション情報を，性能解析トレースファイルに出力されたルートアプリケーション情報と突き合わせて確認することで，性能解析トレース中のどこでリクエストのタイムアウトが発生したかを確認できます。

タイムアウトが発生したリクエストは，該当するルートアプリケーション情報を出力しているリクエスト処理内の，次の表に示すイベント ID のトレース情報で確認できます。オペレーション名に出力されている URI で，該当するリクエストを特定してください。

表 14-4　タイムアウトしたリクエストを確認するために使用できる性能解析トレース

イベント ID	説明
0x8000	リクエスト処理要求の取得後に出力された情報です。 オペレーション名として，リクエストの URI が出力されます。
0x8100	リダイレクタ処理の完了後に出力された情報です。 オペレーション名として，リクエストの URI が出力されます。

14.4.2　ルートアプリケーション情報を利用したログ調査

J2EE アプリケーションまたはバッチアプリケーション内で，アプリケーションサーバの API を使用すると，性能解析情報のルートアプリケーション情報の文字列表現を任意のタイミングでログファイルに出力できます。

ルートアプリケーション情報の文字列表現は，次の形式で出力されます（最大 48 文字）。

```
IPアドレス/プロセスID/通信番号
（例：10.209.15.130/1234/0x0000000000000001）
```

API を使用してログファイルにルートアプリケーション情報を出力すると，ログファイルと性能解析トレースファイルを突き合わせた調査ができるようになります。

Web コンテナでは，新しいリクエストを受け付けた時点で，リクエスト単位で新しいルートアプリケーション情報が割り当てられます。新しいルートアプリケーション情報が割り当てられるトレースポイントを次に示します。

表 14-5　新しいルートアプリケーション情報が割り当てられるトレースポイント

条件	イベント ID（処理内容）	処理内容
インプロセス HTTP サーバを使用している場合	0x8211	リクエスト取得時，リクエストヘッダ解析完了時
Web サーバと連携している場合※	0x8200	リクエスト取得時，リクエストヘッダ解析完了時

注※　Web サーバと連携している場合，リダイレクタで取得したルートアプリケーション情報が割り当てられます。リダイレクタまたはリダイレクタが使用する PRF トレース出力ライブラリのロードに失敗してルートアプリケーション情報が発行されなかった場合，Web コンテナで新しいルートアプリケーション情報が割り当てられます。

また，次のトレース取得ポイントでは，「IP アドレス/プロセス ID/通信番号」が「0.0.0. 0/0/0x0000000000000000」と出力されることがあります。

- 0x8212
  インプロセス HTTP サーバ使用時の Web クライアントからのデータ読み込み開始時
- 0x8312
  インプロセス HTTP サーバ使用時の Web クライアントへのデータ書き込み開始時

「IP アドレス/プロセス ID/通信番号」が「0.0.0. 0/0/0x0000000000000000」と出力されるのは，次の場合です。

- HTTP リクエストヘッダを受信した場合
- HTTP リクエストではない，不正なデータを受信した場合
- リクエスト処理中に例外が発生した場合

なお，ルートアプリケーション情報の文字列表現での出力に使用する API は，com.hitachi.software.ejb.application.prf パッケージで提供されている，CprfTrace クラスの getRootApInfo メソッドです。

J2EE アプリケーションまたはバッチアプリケーション開発時のルートアプリケーション情報の取得の実装については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.7.9　性能解析トレースファイルとスレッドダンプを対応づけた問題個所の調査」を参照してください。API の詳細については，マニュアル「アプリケーションサーバ リファレンス API 編」を参照してください。

14.5　性能解析トレースのトレース取得ポイントと PRF トレース取得レベルの概要

ここでは，トレース取得ポイントと PRF トレース取得レベルについて説明します。

14.5.1　トレース取得ポイント

トレース取得ポイントは大きく分けて，J2EE サーバの開始・終了時でのトレース取得と，各機能レイヤでの処理中のトレース取得およびアプリケーションのメソッドの開始・終了時でのトレース取得があります。

(1)　J2EE サーバの開始・終了時のトレース取得

J2EE サーバの開始処理の完了時，および J2EE サーバの終了処理の開始時に，トレース情報を取得できます。取得できるイベント ID と参照先を次に示します。

- イベント ID

  0x8FFE～0x8FFF
- 参照先

  J2EE サーバの開始・終了時のトレース取得の詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「8.24　J2EE サーバの開始・終了時のトレース取得ポイント」を参照してください。

(2)　各機能レイヤでのトレース取得

取得できるイベント ID と機能レイヤの対応を次の表に示します。

表 14-6　取得できるイベント ID と機能レイヤ

イベント ID	機能レイヤ	図中の番号※	参照先
0x1101～0x1102 0x1301～0x1302 0x1401～0x1406 0x2002～0x2003 0x2101～0x2104 0x3000～0x3008	CTM	5	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.3
0x8000～0x8004 0x8100～0x8104	リダイレクタ	1	14.6
0x8200～0x8203 0x8206～0x8225 0x8300～0x8303 0x8306～0x8307	Web コンテナ	2	14.7，14.8，14.9，14.10

イベント ID	機能レイヤ	図中の番号※	参照先
0x8311～0x8325 0x8E01～0x8E06			
0x8401～0x840A 0x8425～0x8428 0x842D～0x8434 0x8453～0x8454 0x8460～0x846D 0x8470～0x8477 0x8490～0x8491 0x84A0～0x84D9 0x8C41	EJB コンテナ	6	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.9
0x8603～0x861C	JNDI	4	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.10
0x8435～0x843F 0x8811～0x8820 0x8C41	JTA	7	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.11
0x8B00～0x8B01 0x8B80～0x8B89 0x8B8A～0x8C03 0x8C10～0x8C13 0x8C20～0x8C41 0x8C60～0x8C65 0x8C80～0x8C93 0x8CC0～0x8CD9 0x8D00～0x8D19 0x8D42～0x8D4D 0x8D60～0x8D63 0x8D80～0x8D89 0x8D8A～0x8D8F 0x8D90～0x8D99	DB Connector，JCA コンテナ	8	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.12
0x8E01～0x8E06	RMI	3	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.13
0x9400～0x9413	OTS	9	マニュアル「アプリケーション

イベント ID	機能レイヤ	図中の番号※	参照先
			サーバ 機能解説 保守／移行編」の 8.14
0x9C00～0x9C03	標準出力／標準エラー出力／ユーザログ	－	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.15
0x9D00，0x9D01	DI	10	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.16
0xA100，0xA101	バッチアプリケーション実行機能	11	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.17
0xA500～0xA5AF	JPA	12	14.11
0xA300～0xA305 0xA310～0xA315 0xA320～0xA323 0xA330～0xA331 0xA340～0xA363 0xA370～0xA381 0xA390～0xA3C3	CJPA プロバイダ	13	14.12
0x842F 0x8430～0x8432 0x8825 0x8826 0x8B86，0x8B87 0x8B8A～0x8B93 0xAA00～0xAA06 0xAA08～0xAA0D 0xAA10～0xAA19	TP1 インバウンド連携機能	14	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.19
0xA600～0xA60F 0xA610～0xA619 0xA61E，0xA61F 0xA620～0xA62F 0xA630～0xA63F 0xA640～0xA64F 0xA650～0xA65F	CJMS プロバイダ	15	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.20

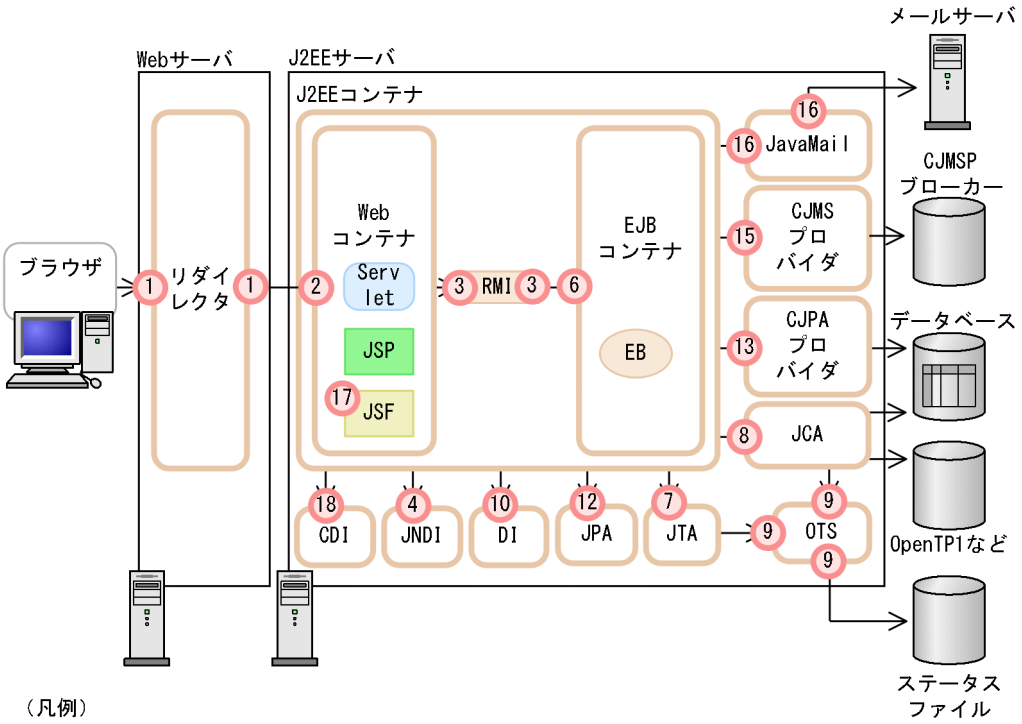
イベント ID	機能レイヤ	図中の番号※	参照先
0xA660～0xA667 0xA61A，0xA61B 0xA668～0xA66F 0xA670～0xA67F 0xA686～0xA68D 0xA692～0xA69B 0xA69E，0xA69F 0xA6A0，0xA6A1 0xA6A6～0xA6AB			
0xAD00～0xAD15 0xAD80～0xAD93	JavaMail	16	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.21
0xAF00～0xAF13	JSF	17	マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の 8.22
0xb002～0xb009	CDI	18	14.13

（凡例）－：該当しない

注※　図 14-2～図 14-5 中の番号と対応しています。

PRF トレースを出力する機能レイヤとトレース取得ポイントについて，システムの構成ごとに次の図に示します。

図 14-2　機能レイヤとトレース取得ポイント（Web クライアント構成の場合）



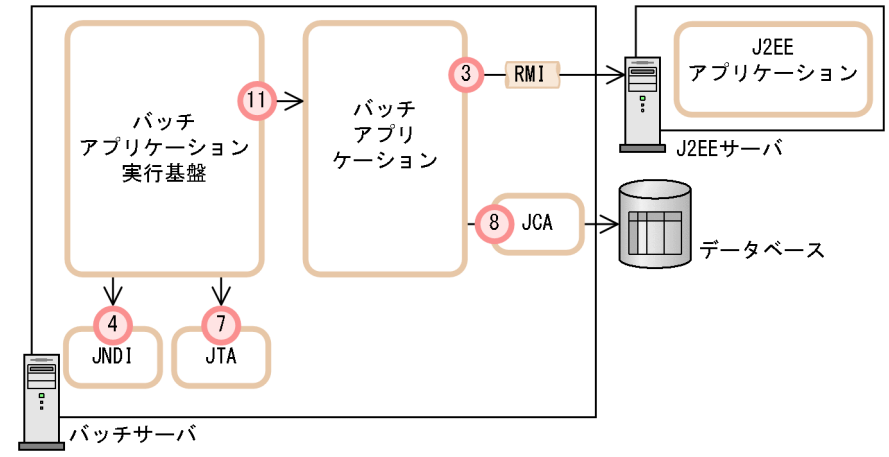
（凡例）

：PRFトレースを出力する機能レイヤ

：トレース取得ポイント

EB　：Enterprise Bean

図 14-3　機能レイヤとトレース取得ポイント（バッチアプリケーションを実行するシステムの場合）



（凡例）

：PRFトレースを出力する機能レイヤ

：トレース取得ポイント

図 14-4　機能レイヤとトレース取得ポイント（EJB クライアント/TPBroker クライアント/TPBroker OTM クライアント構成（CTM 使用）の場合）

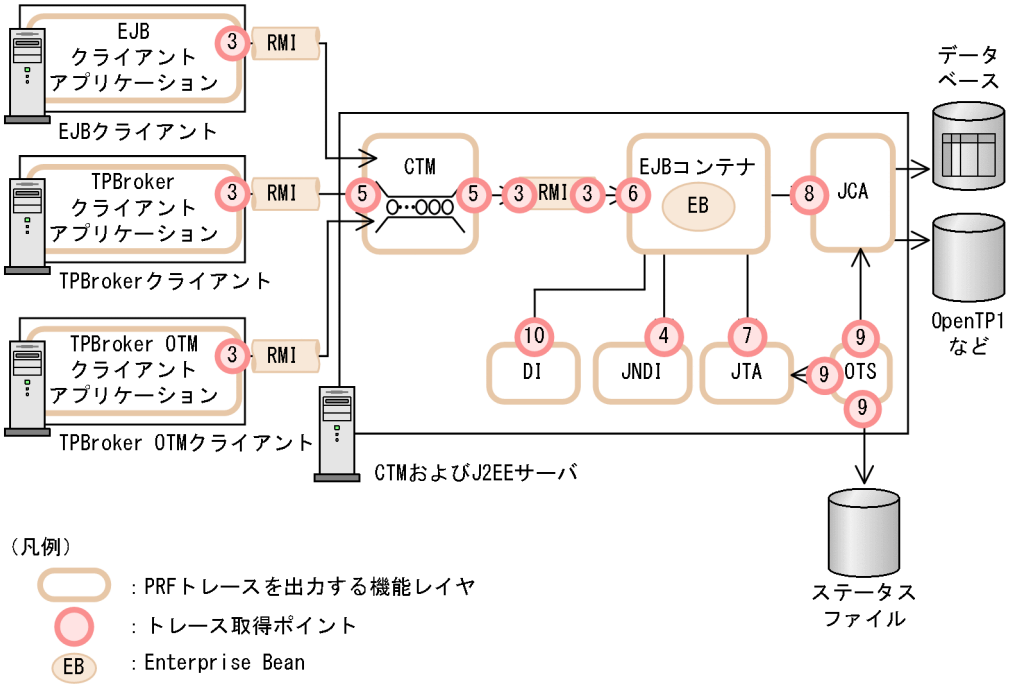
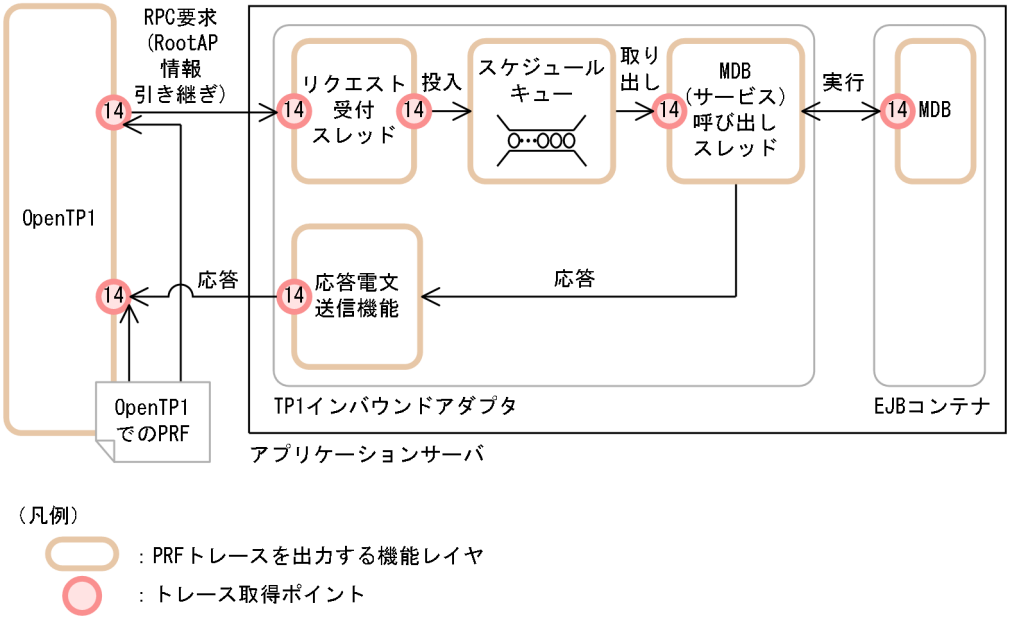


図 14-5　機能レイヤとトレース取得ポイント（TP1 インバウンド連携機能の場合）



トレース取得ポイントは，各機能レイヤ内でさらに詳細に分かれており，トレース取得ポイントによって，PRF トレース取得レベルが異なります。各機能レイヤの詳細なトレース取得ポイント，および PRF トレース取得レベルについては，表 14-6 の参照先を参照してください。

参考

表 14-6 に示した機能レイヤのほか，Application Server のプロセス，構成ソフトウェアおよび関連プログラムでも，PRF トレースが取得できるものがあります。

表 14-6 に示した以外で PRF トレースを取得できる機能レイヤとイベント ID の対応を，次の表に示します。

表 14-7　取得できるイベント ID と機能レイヤに示した以外で PRF トレースを取得できるイベント ID と機能レイヤの対応

イベント ID	機能レイヤ
0x9000～0x90FF 0xA400～0xA4FF	SOAP 通信基盤
0x9100～0x91FF	• TP1 Connector • TP1/Client/J
0x9200～0x92FF	TP1/MQ Access
0x9300～0x93FF	Reliable Messaging
0x9800～0x9B6E	HCSC サーバ
0x9E00～0x9EFF	Service Coordinator Interactive Workflow
0x9F00～0x9FFF	HCSC サーバ（Object Access アダプタ）
0xA000～0xA0FF	HCSC サーバ（ファイルアダプタ）
0xA200～0xA2FF	HCSC サーバ（Message Queue アダプタ）
0xAB00～0xABFF 0xAC00～0xACFF	HCSC サーバ（FTP アダプタ）
0xA400～0xA4FF	JAX-WS エンジン
0xE000～0xE0FF	Elastic Application Data store

(3)　アプリケーションのメソッドの開始・終了時トレース取得

アプリケーションのメソッドの開始時，および終了時にトレース情報を取得できます。取得できるトレース情報を次に示します。

- メソッドの開始時刻，終了時刻
- 識別 ID
- パッケージ名，クラス名，メソッド名
- メソッドで実行された最後の行の行番号
- 発生した例外またはエラーのクラス名

トレース情報の詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「8.25 アプリケーションのトレース取得ポイント」を参照してください。

(4) 各トレースのリターンコード

各トレースのリターンコードは入口トレースの場合，常に「0」が出力されます。

出口トレースおよび呼び出し後トレースの場合，次のように出力されます。

正常終了：0

異常終了：0 以外

14.5.2 PRF トレース取得レベル

性能解析トレースでは，次の 4 種類の PRF トレース取得レベルを設定してトレースを出力できます。レベルによって，トレース取得ポイントの数が異なります。トレース取得ポイント，および PRF トレース取得レベルについては，表 14-6 の参照先を参照してください。

- **標準レベル**
  各機能レイヤの境界（入り口と出口）を識別できるトレース情報を出力します。
- **詳細レベル**
  標準レベルの出力内容に加えて，各機能レイヤ内処理のトレース情報も出力します。
- **保守レベル**
  障害発生時などの保守情報を取得するためのレベルです。
- **抑止レベル**
  トレース情報の出力を抑止するためのレベルです。RMI の機能レイヤにだけ設定できます。

Management Server を利用して運用している場合は，簡易構築定義ファイルで，すべての機能レイヤに対して共通のレベルを設定して，トレース情報を出力します。

なお，次節以降は，トレース取得レベルが標準レベルと詳細レベルのトレース取得ポイントについて説明します。保守レベルは，障害発生時などの保守情報を取得するためのレベルのため，通常は収集する必要はありません。

14.6　リダイレクタのトレース取得ポイント

ここでは，リダイレクタのトレース取得ポイントと，取得できるトレース情報について説明します。

なお，PRF トレース取得レベルを「詳細」にしている場合，リクエスト処理のトレースとセッショントレースが出力されます。

14.6.1　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。なお，「0x8003」のポイントで，セッショントレースは出力されません。

表 14–8　リダイレクタでのトレース取得ポイントの詳細

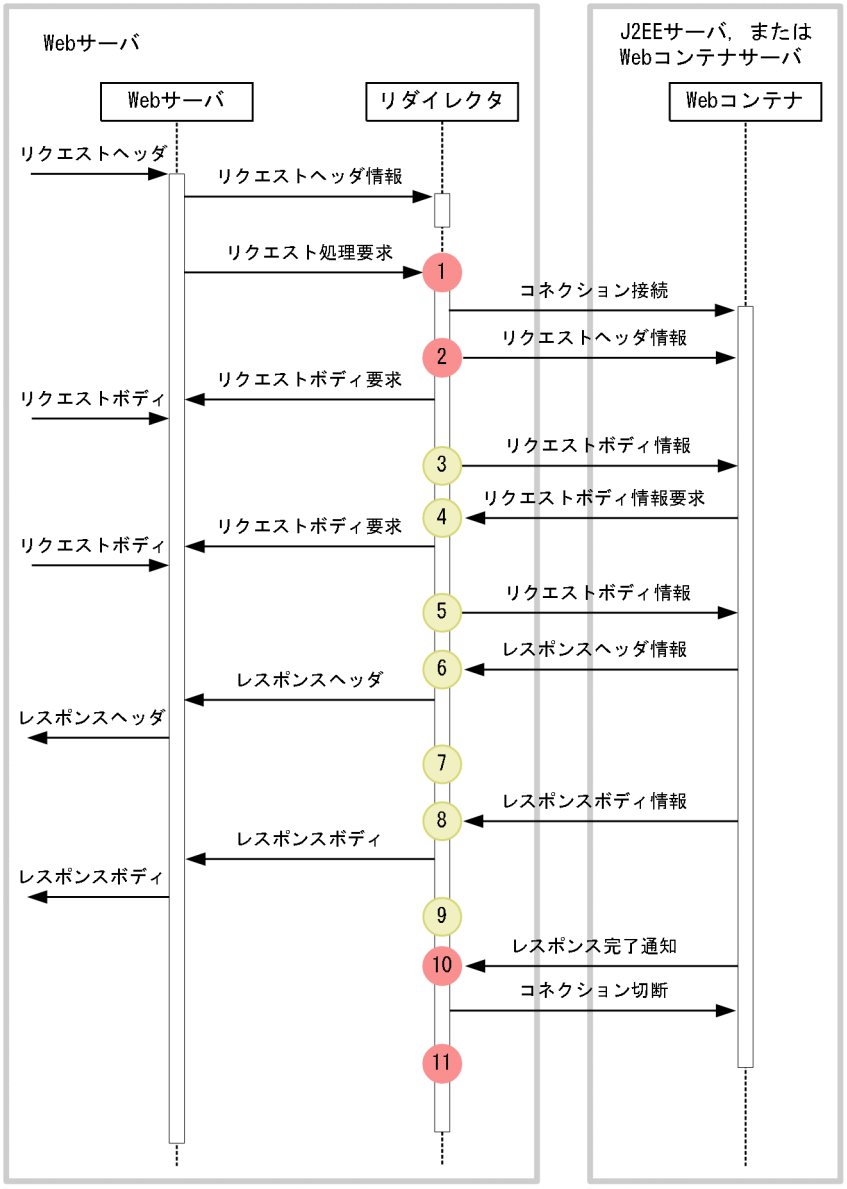
イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8000	1	リクエスト処理要求の取得直後	A／B
0x8001	2	Web コンテナへのリクエストヘッダ情報の送信直後	A／B
0x8002	4	Web コンテナからの HTTP リクエストボディ情報の要求受信直後	B
0x8003	6	Web コンテナからの HTTP レスポンスヘッダ情報の受信直後	B
0x8004	8	Web コンテナからの HTTP レスポンスボディ情報の送信開始直後	B
0x8100	11	リダイレクタ処理完了直後	A／B
0x8101	10	Web コンテナからのレスポンス完了通知の受信直後	A／B
0x8102	3，5	Web コンテナへの HTTP リクエストボディ情報の送信直前	B
0x8103	7	HTTP レスポンスヘッダ情報の設定完了直後	B
0x8104	9	Web サーバへの HTTP レスポンスボディ情報の送信完了直後	B

（凡例）B：詳細　A／B：標準と詳細で異なる情報を取得

注※　図 14-6 中の番号と対応しています。

リダイレクタでのトレース取得ポイントを次の図に示します。

図 14-6　リダイレクタのトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

コネクションは，通常，常設コネクションのために使い回されるため，リクエストのたびに切断されることはありません。コネクションは，通信で例外が発生した場合，または常設コネクションの上限数に達していた場合だけ切断されます。

次に示すポイントでは，トレース情報が出力される場合が制限されます。

3 のポイント

ボディデータの形式がチャンク形式ではない場合にトレース情報が出力されます。ボディデータの形式がチャンク形式の場合，2 と 4 のポイント間の Web サーバへのリクエストボディ要求および Web コンテナへのリクエストボディ情報が送信されません。

4 と 5 のポイント

Web コンテナからリクエストボディ情報の要求があった場合だけトレース情報が出力されます。また，4，5，8，9 のポイントでは，リクエストボディ情報とレスポンスボディ情報が複数回クライアントに送信されることがあるため，トレース情報も複数回出力されることがあります。

また，次に示すポイントでは，無効なセッション ID を取得する場合や，セッション ID を取得しない場合があります。

1〜5 のポイント

セッション ID を取得できます。ただし，リクエストヘッダの Cookie または URL からセッション ID を取得するため，無効なセッション ID（J2EE アプリケーションで破棄された HttpSession の ID または有効期限切れで破棄された HttpSession の ID）を取得する場合があります。

また，有効なセッション ID を取得した場合も，J2EE アプリケーションでセッションが破棄されることがあります。

7〜11 のポイント

セッションを生成した場合だけ，セッション ID を取得できます。

14.6.2 取得できるトレース情報

リダイレクタで取得できるトレース情報を次の表に示します。

表 14–9 リダイレクタで取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション※2
1	0x8000	A／B	クライアントのアドレス：HTTP メソッド	URI	※3
2	0x8001	A／B	コンテナのアドレス：ポート番号	−	※3
3，5	0x8102	B	送信サイズ	−	<入り口時刻><セッション ID 文字数:セッション ID:取得方法>
4	0x8002	B	要求サイズ	−	<セッション ID 文字数:セッション ID:取得方法>
6	0x8003	B	−	−	−
7	0x8103	B	−	−	※4

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション※2
8	0x8004	B	送信サイズ	－	<セッション ID 文字数:セッション ID:取得方法>
9	0x8104	B	送信できたサイズ	－	<入り口時刻><セッション ID 文字数:セッション ID>
10	0x8101	A／B	コンテナのアドレス：ポート番号	－	※5
11	0x8100	A／B	クライアントのアドレス：HTTP メソッド	URI	※6

（凡例）B：詳細　A／B：標準と詳細で異なる情報を取得　－：該当なし

注※1　図 14-6 中の番号と対応しています。

注※2　セッションの Cookie の名前を「JSESSIONID」以外の文字列に変更した場合，セッション ID 文字数，セッション ID，および取得方法は出力されません。

注※3　レベルが「標準」の場合，何も出力されません。レベルが「詳細」の場合，セッション ID 文字数:セッション ID:取得方法が出力されます。

注※4　正常に処理された場合，入り口時刻とセッション ID が表示されます。例外が発生した場合，入り口時刻に加え，4 バイトの保守情報，およびセッション ID 文字数:セッション ID が表示されます。

注※5　レベルが「標準」の場合，入り口時刻が表示されます。レベルが「詳細」の場合，入り口時刻に加え，セッション ID 文字数:セッション ID が表示されます。

注※6　レベルが「標準」の場合，常に，入り口時刻とステータスコードが表示されます。ただし，Microsoft IIS 使用時に例外が発生した場合は，入り口時刻とステータスコードに加え，4 バイトの保守情報が表示されます。レベルが「詳細」の場合，レベルが「標準」の場合の情報に加え，セッション ID 文字数:セッション ID が表示されます。

14.7　Web コンテナのトレース取得ポイント（リクエスト処理のトレース）

ここでは，Web コンテナのトレース取得ポイントと，取得できるトレース情報について説明します。なお，Web コンテナでは，リクエスト処理のトレースとセッショントレースが出力されます。ここでは，リクエスト処理のトレースについて説明します。また，インプロセス HTTP サーバを使用したときのトレースポイントと取得できるトレース情報についても説明します。

14.7.1　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–10　Web コンテナでのトレース取得ポイントの詳細（リクエスト処理のトレース）

イベント ID	図中の番号※1	トレース取得ポイント		レベル
0x8200※2	1	リクエスト取得・リクエストヘッダ解析完了直後	Web サーバ経由の場合	A
0x8201			簡易 Web サーバ経由の場合	A
0x8202※3	3	サーブレット／JSP の呼び出し直前		A
0x8203	2	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前 （web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）		B
0x8206	4	RequestDispatcher 経由のサーブレット／JSP 呼び出し直前		B
0x8207	3	静的コンテンツの呼び出し		B
0x8300	8	リクエスト処理完了	Web サーバ経由の場合	A
0x8301			簡易 Web サーバ経由の場合	A
0x8302	6	サーブレット／JSP の処理完了直後		A
0x8303	7	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後 （web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）		B
0x8306	5	RequestDispatcher 経由のサーブレット／JSP 処理完了直後		B
0x8307	6	静的コンテンツの処理完了直後（DefaultServlet）		B

（凡例）A：標準　B：詳細

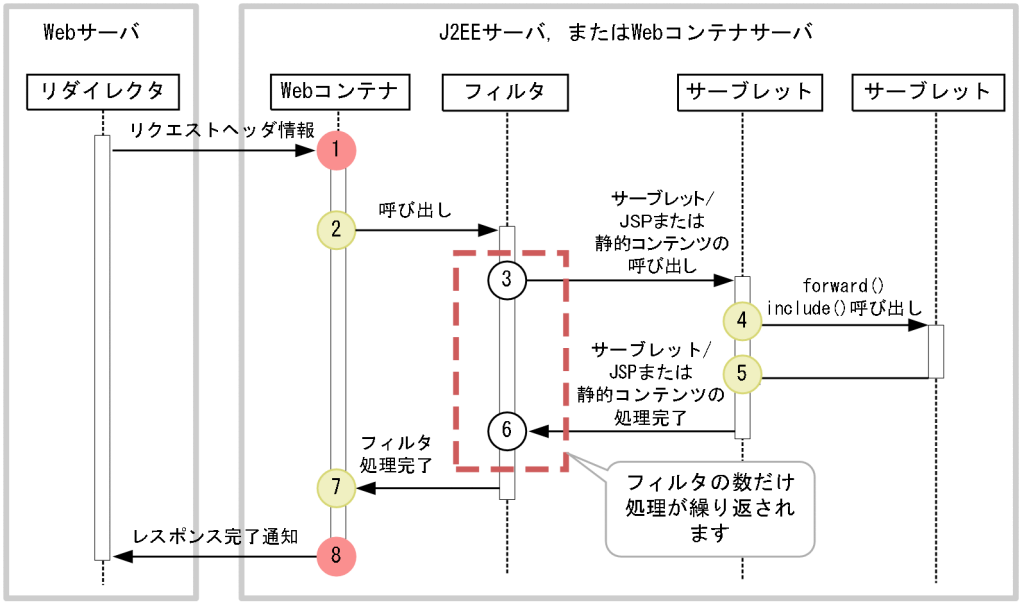
注※1　図 14-7 中の番号と対応しています。

注※2　POST リクエストの場合，クライアントから POST データが来ないと，この区間で遅延することがあります。

注※3　JSP のコンパイルが必要な場合は，JSP のコンパイルを実行したあと，トレースを取得します。

Web コンテナでのトレース取得ポイントを次の図に示します。

図 14-7　Web コンテナのトレース取得ポイント（リクエスト処理のトレース）



（凡例）

：トレース取得ポイントを示します。PRF トレース取得レベルは「標準」です。

：トレース取得ポイントを示します。PRF トレース取得レベルは「詳細」です。

：トレース取得ポイントを示します。
サーブレットの呼び出しの場合，PRF トレース取得レベルは「標準」です。
静的コンテンツの呼び出しの場合，PRF トレース取得レベルは「詳細」です。

14.7.2　取得できるトレース情報

Web コンテナで取得できるトレース情報を次の表に示します。

表 14-11　Web コンテナで取得できるトレース情報（リクエスト処理のトレース）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
	0x8201	A	HTTP メソッド	URI	－
2	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
3	0x8202	A	クラス名または JSP ファイル名	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバル

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					セッション ID 文字数:グローバルセッション ID>
	0x8207	B	−	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
4	0x8206	B	クラス名	ディスパッチのタイプ コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
5	0x8306	B	クラス名	ディスパッチのタイプ コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
6	0x8302	A	クラス名または JSP ファイル名	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
7	0x8303	B	クラス名	−	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><例外名:セッション ID 文字数:セッション ID>
8	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻>ステータスコード • 例外発生時 <入り口時刻><ステータスコード><例外名>
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8301	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-7 中の番号と対応しています。

参考

SOAP クライアント以外のリクエストを受信した場合，トレース情報のキー情報である「クライアントアプリケーション情報」には，常に 0 が表示されます。SOAP クライアントのリクエストを受信した場合だけ，「クライアントアプリケーション情報」が出力されます。

14.7.3 トレース取得ポイントと PRF トレース取得レベル（インプロセス HTTP サーバを使用した場合）

インプロセス HTTP サーバを使用した場合のイベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–12 インプロセス HTTP サーバでのトレース取得ポイントの詳細

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0x8202※2	5	サーブレット／JSP 呼び出し直前	A／B
0x8203	4	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前 （web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8206	6	RequestDispatcher 経由のサーブレット／JSP 呼び出し直前	B
0x8207	5	静的コンテンツの呼び出し直前	B
0x8211	3	リクエスト取得時／リクエストヘッダ解析完了直後	A／B
0x8212	1	Web クライアントからのデータ読み込み開始直前	B
0x8213	8	Web クライアントへのデータ書き込み開始直前	B
0x8302	10	サーブレット／JSP 処理完了直後	A
0x8303	11	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後 （web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8306	7	RequestDispatcher 経由のサーブレット／JSP 処理完了直後	B
0x8307	10	静的コンテンツの処理完了直後（DefaultServlet）	B
0x8311	12	リクエスト処理完了直後	A／B
0x8312	2	Web クライアントからのデータ読み込み完了直後	B
0x8313	9	Web クライアントへのデータ書き込み完了直後	B

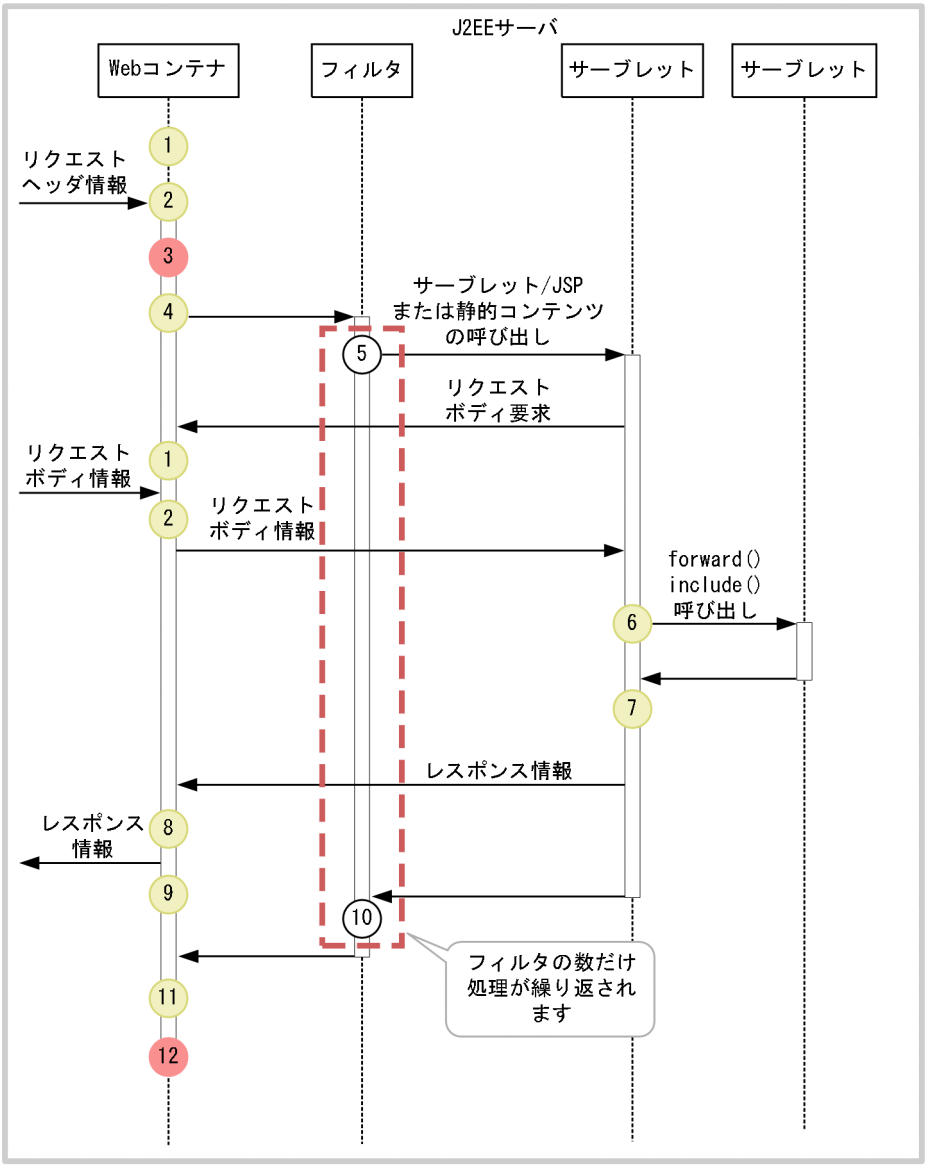
（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※1　図 14-8 中の番号と対応しています。

注※2　JSP のコンパイルが必要な場合は，JSP のコンパイルを実行したあと，トレースを取得します。

インプロセス HTTP サーバを使用した場合のトレース取得ポイントを次の図に示します。

図 14-8　インプロセス HTTP サーバのトレース取得ポイント



（凡例）
- ：トレース取得ポイントを示します。PRF トレース取得レベルは「標準」です。
- ：トレース取得ポイントを示します。PRF トレース取得レベルは「詳細」です。
- ：トレース取得ポイントを示します。
  サーブレットの呼び出しの場合，PRF トレース取得レベルは「標準」です。
  静的コンテンツの呼び出しの場合，PRF トレース取得レベルは「詳細」です。

なお，1 および 2 のポイントはリクエスト情報をクライアントから複数回受信することがあるため，トレース情報も複数回出力されることがあります。

8 および 9 のポイントはレスポンス情報が複数回クライアントに送信されることがあるため，トレース情報も複数回出力されることがあります。

フィルタからリクエストボディ情報の要求や，レスポンスの送信を行った場合も，1 および 2 または 8 および 9 のトレースは出力されます。

14.7.4 取得できるトレース情報

インプロセス HTTP サーバで取得できるトレース情報を次の表に示します。

表 14–13 インプロセス HTTP サーバで取得できるトレース情報（リクエスト処理のトレース）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8212	B	要求サイズ	−	−
2	0x8312	B	読み込んだサイズ	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
3	0x8211	A	HTTP メソッド	URI	−
		B			<セッション ID 文字数:セッション ID:取得方法>
4	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
5	0x8202	A	クラス名または JSP ファイル名	−	−
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8207	B	−	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
6	0x8206	B	クラス名	ディスパッチのタイプ コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
7	0x8306	B	クラス名	ディスパッチのタイプ コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><例外名:セッション ID 文字数:セッション ID>
8	0x8213	B	書き込みサイズ	−	−
9	0x8313	B	書き込んだサイズ	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
10	0x8302	A	クラス名または JSP ファイル名	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
11	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
12	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-8 中の番号と対応しています。

14.8 Web コンテナのトレース取得ポイント（セッショントレース）

ここでは，Web コンテナのトレースのトレース取得ポイントと，取得できるトレース情報について説明します。なお，Web コンテナでは，リクエスト処理のトレースとセッショントレースが出力されます。ここでは，セッショントレース，およびグローバルセッションについてのトレース取得ポイントと取得できるトレース情報について説明します。

14.8.1 トレース取得ポイントと PRF トレース取得レベル（セッショントレース）

セッショントレースに関連するトレースの，イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。なお，「0x8203」，「0x8202」，「0x8207」，「0x8206」および「0x8300」では，グローバルセッションについての情報も出力されます。

表 14–14 Web コンテナでのトレース取得ポイントの詳細（セッショントレース）

イベント ID	図中の番号※1	トレース取得ポイント	レベル※2
0x8200	1	リクエスト取得・リクエストヘッダ解析完了直後（Web サーバ経由の場合）	A／B
0x8202	4，9	サーブレット／JSP 呼び出し直前	A／B
0x8203	2，3	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前 （web.xml の<filter-mapping>タグの<dispatcher>タグを省略した場合，または<dispatcher>タグで"REQUEST"を指定したフィルタを呼び出した場合）	B
0x8206	7	RequestDispatcher 経由のサーブレット／JSP 呼び出し直前	B
0x8207	4，9	静的コンテンツ呼び出し直前（DefaultServlet）	B
0x8208	5	セッション生成後	B
0x8209	6	セッション破棄後	B
0x8210	17	セッションタイムアウト後	B
0x8211	1	リクエスト取得・リクエストヘッダ解析完了直後（インプロセス HTTP サーバ経由の場合）	A／B
0x8214	8	フォワード時に実行されるフィルタの呼び出し直前 （web.xml の<filter-mapping>タグの<dispatcher>タグで"FORWARD"を指定したフィルタを呼び出した場合）	B
0x8215	8	インクルード時に実行されるフィルタの呼び出し直前 （web.xml の<filter-mapping>タグの<dispatcher>タグで"INCLUDE"を指定したフィルタを呼び出した場合）	B
0x8216	2	エラーページに転送される際に実行されるフィルタの呼び出し直前	B

イベント ID	図中の番号※1	トレース取得ポイント	レベル※2
		(web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタを呼び出した場合)	
0x8300	16	リクエスト処理完了直後（Web サーバ経由の場合）	A／B
0x8302	10，13	サーブレット／JSP 処理完了直後	A／B
0x8303	14，15	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後 (web.xml の<filter-mapping>タグの<dispatcher>タグで""REQUEST""を指定したフィルタの処理が完了した場合)	B
0x8306	12	RequestDispatcher 経由のサーブレット／JSP 処理完了直後	B
0x8307	10，13	静的コンテンツ処理完了直後（DefaultServlet）	B
0x8311	16	リクエスト処理完了直後（インプロセス HTTP サーバ経由の場合）	A／B
0x8314	11	フォワード時に実行されるフィルタの処理完了直後 (web.xml の<filter-mapping>タグの<dispatcher>タグで"FORWARD"を指定したフィルタの処理が完了した場合)	B
0x8315	11	インクルード時に実行されるフィルタの処理完了直後 (web.xml の<filter-mapping>タグの<dispatcher>タグで"INCLUDE"を指定したフィルタの処理が完了した場合)	B
0x8316	15	エラーページに転送される際に実行されるフィルタの処理完了直後 (web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタの処理が完了した場合)	B

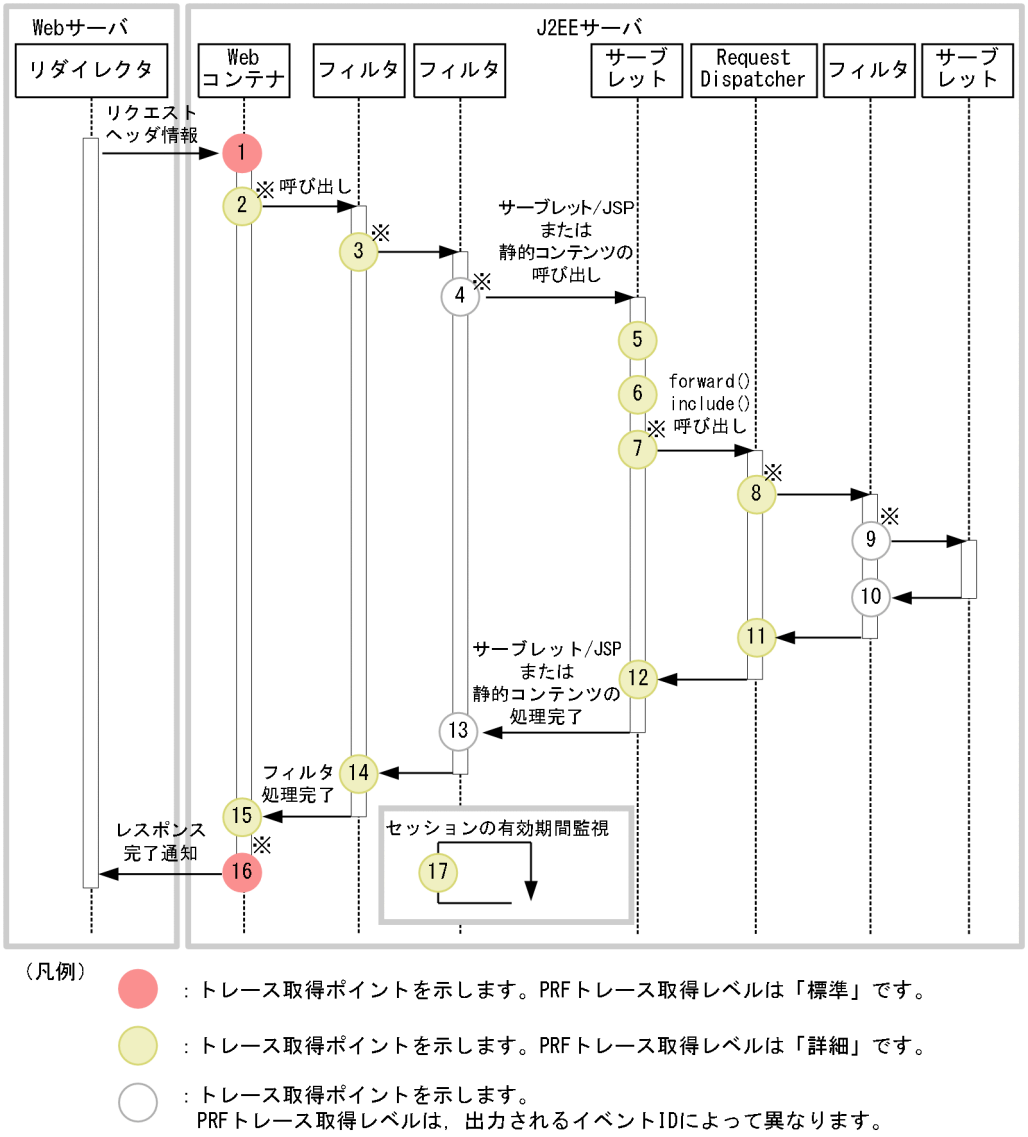
（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※1　図 14-9 中の番号と対応しています。

注※2　セッショントレースについての情報はレベルが「詳細」の場合だけ出力されます。

Web コンテナでのセッショントレースのトレース取得ポイントを次の図に示します。

図 14-9　Web コンテナのトレース取得ポイント（セッショントレース）



注※
グローバルセッションIDも取得できるトレース取得ポイントを示します。

それぞれのポイントで取得できるセッション ID について説明します。

1 のポイント

セッション ID を取得できます。ただし，リクエストヘッダの Cookie または URL からセッション ID を取得するため，無効なセッション ID（J2EE アプリケーションで破棄された HttpSession の ID または有効期限切れで破棄された HttpSession の ID）を取得する場合があります。

また，有効なセッション ID を取得した場合も，J2EE アプリケーションでセッションが破棄される場合があります。

2，3，4，7，8，9 のポイント

トレース取得ポイントで有効なセッション ID を取得できます。ただし，J2EE アプリケーションでセッションが破棄される場合があります。

また，これらのポイントでは，グローバルセッション ID も取得できます。取得できるグローバルセッション ID の内容は，トレース取得ポイントごとに異なります。

- 2 のポイントは，一つのリクエストで最初にイベント ID「0x8203」が出力されるトレース取得ポイントです。このトレース取得ポイントでは，Web クライアントからリクエストとして送信されたグローバルセッション ID が取得できます。ただし，このポイントでは，すでに無効になっているグローバルセッション ID が出力される場合もあります。
- 3，4，7，8，9 のポイントで出力される，イベント ID が「0x8216」「0x8202」「0x8203」「0x8206」「0x8207」「0x8214」「0x8215」のトレースには，その時点で有効なグローバルセッション ID が取得できます。

5 のポイント

J2EE アプリケーションでセッションが生成された場合だけ，トレース取得ポイントで有効なセッション ID を取得できます。ただし，J2EE アプリケーションでセッションが破棄される場合があります。

6 のポイント

J2EE アプリケーションでセッションが破棄された場合だけ，トレース取得ポイントで無効になったセッション ID を取得できます。ただし，J2EE アプリケーションでセッションが破棄される場合があります。

10，11，12，13 のポイント

トレース取得ポイントで有効なセッション ID を取得できます。ただし，J2EE アプリケーションでセッションが破棄される場合があります。

14，15 のポイント

トレース取得ポイントで有効なセッション ID を取得できます。なお，このトレース取得ポイントでリクエスト処理が完了すると，以降の J2EE アプリケーションでセッションが破棄されることはありません。

16 のポイント

トレース取得ポイントで有効なセッション ID を取得できます。なお，このトレース取得ポイントでリクエスト処理が完了すると，以降の J2EE アプリケーションでセッションが破棄されることはありません。

また，グローバルセッションを生成した場合，リクエスト処理が終了した時点で有効なグローバルセッション ID を取得できます。

17 のポイント

有効期限を超えたセッションが破棄された場合だけ，無効になったセッション ID を取得できます。

14.8.2 取得できるトレース情報

セッショントレースについて，Web コンテナで取得できるトレース情報を次の表に示します。なお，イベント ID が「0x8202」，「0x8203」，「0x8206」，「0x8207」，「0x8214」，「0x8215」，「0x8300」および「0x8311」のトレース取得ポイントでは，グローバルセッションについての情報も出力されます。

表 14–15　Web コンテナで取得できるトレース情報（セッショントレース）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	－ <セッション ID 文字数:セッション ID:取得方法>
		B			
4，9	0x8202	A	クラス名（JSP 呼び出し時は JSP ファイル名）	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
2，3	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
7	0x8206	B	クラス名	ディスパッチのタイプ コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
4，9	0x8207	B	－	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
5	0x8208	B	コンテキストルート名	セッション有効期間	<セッション ID 文字数:セッション ID>
6	0x8209	B	コンテキストルート名	セッション生成時刻	<セッション ID 文字数:セッション ID>
17	0x8210	B	コンテキストルート名	セッション有効期間:セッション生成時刻	<セッション ID 文字数:セッション ID>
1	0x8211	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
8	0x8214	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
8	0x8215	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
2	0x8216	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
16	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
10，13	0x8302	A	クラス名（JSP 呼び出し時は JSP ファイル名）	－	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
14，15	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
12	0x8306	B	クラス名	ディスパッチのタイプ コンテキストルート名	• 正常時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
10，13	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
16	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
11	0x8314	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
11	0x8315	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
15	0x8316	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>

（凡例）A：標準　B：詳細　－：該当なし
注※　図 14-9 中の番号と対応しています。

14.9　Web コンテナのトレース取得ポイント（フィルタのトレース）

ここでは，フォワード時，またはインクルード時に呼び出されるフィルタを設定した場合の Web コンテナのトレースのトレース取得ポイントと，取得できるトレース情報について説明します。

なお，フォワード時，またはインクルード時に呼び出されるフィルタを設定した場合の Web コンテナでは，正常に処理が終了した場合と障害が発生した場合で取得できるトレース情報が異なります。ここでは，それぞれの場合について説明します。

なお，JSP の page ディレクティブで errorPage 属性を使用しエラーページを設定して，JSP で例外が発生した場合，リクエストの forward でエラーページが表示されます。したがって，JSP でエラーページを表示する場合も，forward 時のトレースが出力されます。

14.9.1　正常に処理が終了した場合の Web コンテナのトレース取得ポイント（フィルタのトレース）

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–16　正常終了時の Web コンテナでのトレース取得ポイントの詳細（フィルタのトレース）

イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8200	1	リクエスト取得・リクエストヘッダ解析完了直後（Web サーバ経由）	A／B
0x8201	1	リクエスト取得時・リクエストヘッダ解析完了直後（簡易 Web サーバ経由）	A
0x8202	3	サーブレット／JSP の呼び出し直前	A／B
0x8202	6	サーブレット／JSP の呼び出し直前	A／B
0x8203	2	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8206	4	RequestDispatcher 経由のサーブレット／JSP 呼び出し直前	B
0x8207	6	静的コンテンツ呼び出し直前（DefaultServlet）	B
0x8211	1	リクエスト取得時・リクエストヘッダ解析完了直後（インプロセス HTTP サーバ経由）	A／B
0x8214	5	フォワード時に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグで"FORWARD"を指定したフィルタ）	B

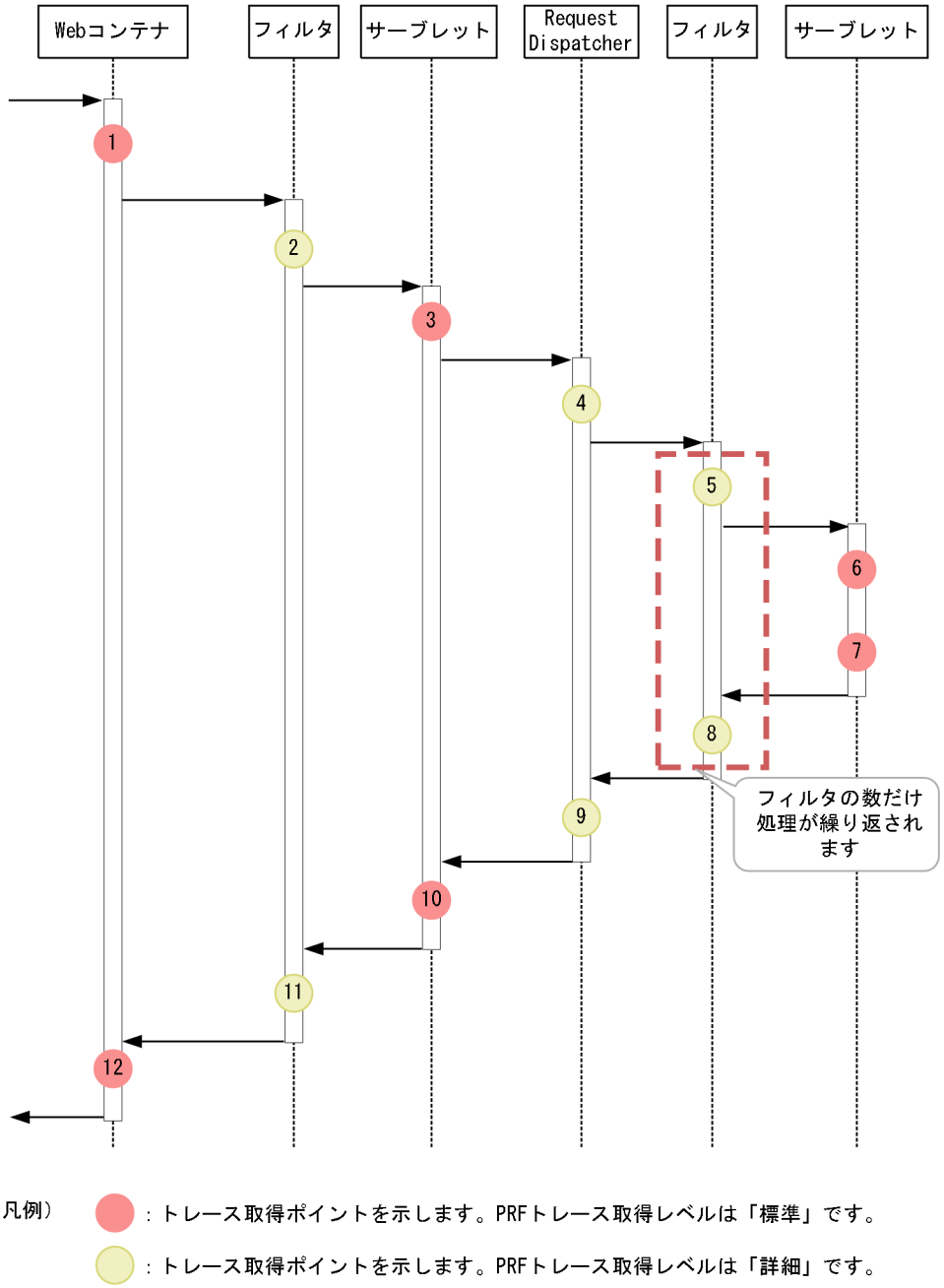
イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8215	5	インクルード時に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグで"INCLUDE"を指定したフィルタ）	B
0x8300	12	リクエスト処理完了直後（Web サーバ経由）	A／B
0x8301	12	リクエスト処理完了直後（簡易 Web サーバ経由）	A
0x8302	7	サーブレット／JSP の処理完了直後	A／B
0x8302	10	サーブレット／JSP の処理完了直後	A／B
0x8303	11	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8306	9	RequestDispatcher 経由のサーブレット／JSP 処理完了直後	B
0x8307	7	静的コンテンツ処理完了直後(DefaultServlet)	B
0x8311	12	リクエスト処理完了直後（インプロセス HTTP サーバ経由）	A／B
0x8314	8	フォワード時に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"FORWARD"を指定したフィルタ）	B
0x8315	8	インクルード時に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"INCLUDE"を指定したフィルタ）	B

（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※　図 14-10 中の番号と対応しています。

フォワード時，またはインクルード時に呼び出されるフィルタを設定した場合の Web コンテナでのトレース取得ポイントを次の図に示します。

図 14-10　正常終了時の Web コンテナでのトレース取得ポイント（フィルタのトレース）



(2)　取得できるトレース情報

フォワード時，またはインクルード時に呼び出されるフィルタを設定した場合の Web コンテナで取得できるトレース情報を次の表に示します。

表 14-17　正常終了時の Web コンテナで取得できるトレース情報（フィルタのトレース）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	−

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
		B			<セッション ID 文字数:セッション ID:取得方法>
	0x8201	A	HTTP メソッド	URI	－
	0x8211	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
2	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
3	0x8202	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
4	0x8206	B	クラス名	ディスパッチのタイプコンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
5	0x8214	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8215	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
6	0x8202	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8207	B	－	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
7	0x8302	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	• 正常時 <入り口時刻> • 例外発生時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
8	0x8314	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8315	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
9	0x8306	B	クラス名	ディスパッチのタイプコンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
10	0x8302	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
11	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
12	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8301	A	HTTP メソッド	URI	• 正常時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>
	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-10 中の番号と対応しています。

14.9.2　例外が発生した場合の Web コンテナのトレース取得ポイント（フィルタのトレース）

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–18　例外が発生した場合の Web コンテナでのトレース取得ポイントの詳細（フィルタのトレース）

イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8200	1	リクエスト取得・リクエストヘッダ解析完了直後（Web サーバ経由）	A／B
0x8201	1	リクエスト取得時・リクエストヘッダ解析完了直後（簡易 Web サーバ経由）	A
0x8202	3	サーブレット／JSP の呼び出し直前	A／B
0x8202	8	サーブレット／JSP の呼び出し直前	A／B
0x8203	2	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8206	6	RequestDispatcher 経由のサーブレット／JSP 呼び出し直前	B
0x8207	8	静的コンテンツ呼び出し直前（DefaultServlet）	B

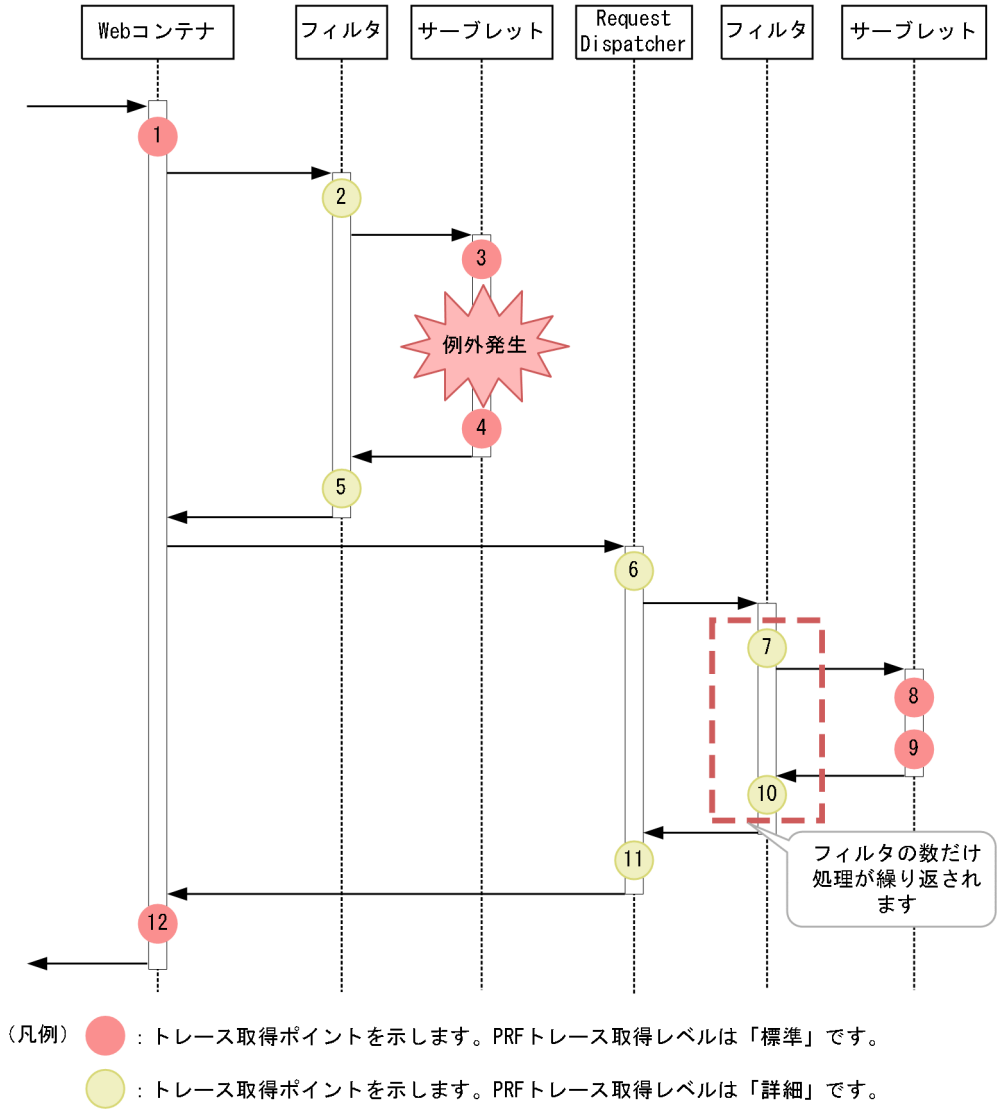
イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8211	1	リクエスト取得時・リクエストヘッダ解析完了直後（インプロセス HTTP サーバ経由）	A／B
0x8216	7	エラーページに転送される際に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタ）	B
0x8300	12	リクエスト処理完了直後（Web サーバ経由）	A／B
0x8301	12	リクエスト処理完了直後（簡易 Web サーバ経由）	A
0x8302	4，9	サーブレット／JSP の処理完了直後	A／B
0x8303	5	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8306	11	RequestDispatcher 経由のサーブレット／JSP 処理完了直後	B
0x8307	9	静的コンテンツ処理完了直後(DefaultServlet)	B
0x8311	12	リクエスト処理完了直後（インプロセス HTTP サーバ経由）	A／B
0x8316	10	エラーページに転送される際に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタ）	B

（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※　図 14-11 中の番号と対応しています。

例外が発生した場合の Web コンテナでのトレース取得ポイントを次の図に示します。

図 14-11　例外が発生した場合の Web コンテナでのトレース取得ポイント



(2)　取得できるトレース情報

フォワード時，またはインクルード時に呼び出されるフィルタを設定した場合の Web コンテナで取得できるトレース情報を次の表に示します。

表 14-19　例外が発生した場合の Web コンテナで取得できるトレース情報（フィルタのトレース）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	−
		B			<セッション ID 文字数:セッション ID:取得方法>
	0x8201	A	HTTP メソッド	URI	−

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0x8211	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
2	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
3	0x8202	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
4	0x8302	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
5	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
6	0x8206	B	クラス名	ディスパッチのタイプコンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
7	0x8216	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
8	0x8202	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8207	B	－	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
9	0x8302	A	クラス名（JSP 呼び出しの場合は JSP ファイル名）	－	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	－	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
10	0x8316	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
11	0x8306	B	クラス名	ディスパッチのタイプ コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					• 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
12	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8301	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-11 中の番号と対応しています。

14.10 Web コンテナのトレース取得ポイント（データベースセッションフェイルオーバ機能のトレース）

ここでは，データベースセッションフェイルオーバ機能を使用した場合のトレース取得ポイントと，取得できるトレース情報について説明します。

14.10.1 HTTP セッションを作成するリクエスト処理のトレース取得ポイントと取得できるトレース情報（データベースセッションフェイルオーバ機能のトレース）

HTTP セッションを作成するリクエスト処理のトレース取得ポイントと取得できるトレース情報について説明します。

(1) トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-20 HTTP セッションを作成するリクエスト処理のトレース取得ポイントの詳細（データベースセッションフェイルオーバ機能）

イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8200	1	リクエスト取得・リクエストヘッダ解析完了直後（Web サーバ経由）	A／B
0x8202	2	サーブレット／JSP 呼び出し直前	A／B
0x8203	2	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8207	2	静的コンテンツ呼び出し直前（DefaultServlet）	B
0x8211	1	リクエスト取得・リクエストヘッダ解析完了直後（インプロセス HTTP サーバ経由）	A／B
0x8219	6	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のシリアライズ開始直前	A
0x8222	8	データベースセッションフェイルオーバ機能による Web アプリケーション処理後のデータベースアクセス開始直前	A
0x8223	3	データベースセッションフェイルオーバ機能による HTTP セッション作成時のデータベースアクセス開始直前	A
0x8300	10	リクエスト処理完了直後（Web サーバ経由）	A／B
0x8302	5	サーブレット／JSP 処理完了直後	A／B

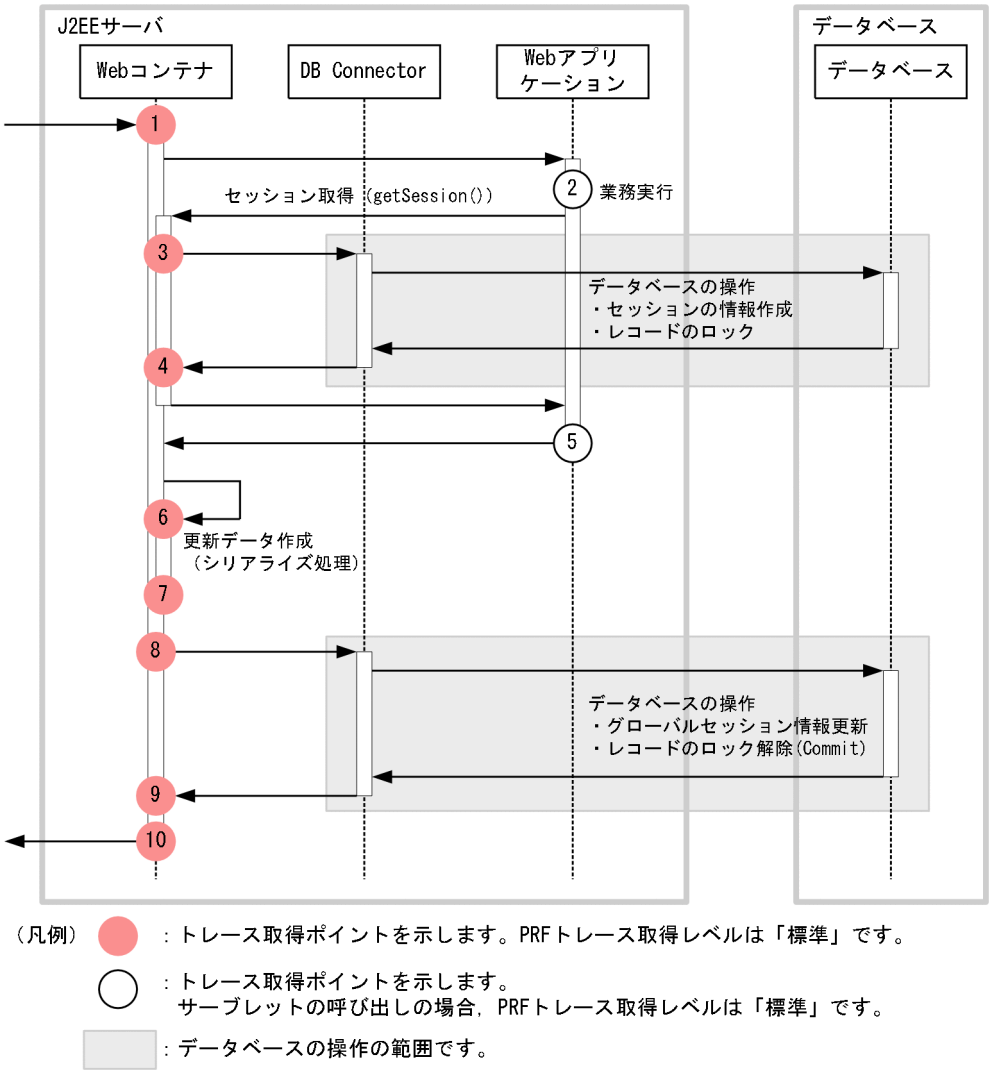
イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8303	5	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8307	5	静的コンテンツ処理完了直後（DefaultServlet）	B
0x8311	10	リクエスト処理完了直後（インプロセス HTTP サーバ経由）	A／B
0x8316	5	エラーページに転送される際に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタ）	B
0x8319	7	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のシリアライズ終了直後	A
0x8322	9	データベースセッションフェイルオーバ機能による Web アプリケーション処理後のデータベースアクセス終了直後	A
0x8323	4	データベースセッションフェイルオーバ機能による HTTP セッション作成時のデータベースアクセス終了直後	A

（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※　図 14-12 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-12　HTTP セッションを作成するリクエスト処理のトレース取得ポイント（データベースセッションフェイルオーバ機能）



(2)　取得できるトレース情報

HTTP セッションを作成するリクエスト処理で取得できるトレース情報を次の表に示します。

表 14-21　HTTP セッションを作成するリクエスト処理で取得できるトレース情報（データベースセッションフェイルオーバ機能）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
	0x8211	A	HTTP メソッド	URI	－

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
		B			<セッション ID 文字数:セッション ID:取得方法>
2	0x8202	A	クラス名 (JSP 呼び出し時は JSP ファイル名)	−	−
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8207	B	−	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
3	0x8223	A	−	−	−
4	0x8323	A	完全性保障モードが有効な場合 排他を取得したレコード番号 完全性保障モードが無効な場合 -1	−	• 正常時 <入り口時刻><セッション ID 文字数:作成した HTTP セッションのセッション ID> • 例外発生時 <入り口時刻><例外名>
5	0x8302	A	クラス名 (JSP 呼び出し時は JSP ファイル名)	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8316	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
6	0x8219	A	リクエスト URL	−	<セッション ID 文字数:セッション ID>
7	0x8319	A	リクエスト URL	シリアライズ後の HTTP セッションの属性情報のサイズ(バイト)	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
8	0x8222	A	−	−	<セッション ID の文字数:HTTP セッションのセッション ID>
9	0x8322	A	完全性保障モードが有効な場合 排他を解放したレコード番号 完全性保障モードが無効な場合 -1	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
10	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID >

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-12 中の番号と対応しています。

14.10.2　HTTP セッションを更新するリクエスト処理のトレース取得ポイントと取得できるトレース情報（データベースセッションフェイルオーバ機能のトレース）

HTTP セッションを更新するリクエスト処理のトレース取得ポイントと取得できるトレース情報について説明します。

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-22　HTTP セッションを更新するリクエスト処理のトレース取得ポイントの詳細（データベースセッションフェイルオーバ機能）

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0x8200	1	リクエスト取得・リクエストヘッダ解析完了直後（Web サーバ経由）	A／B

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0x8202	6	サーブレット／JSP 呼び出し直前	A／B
0x8203	6	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8207	6	静的コンテンツ呼び出し直前（DefaultServlet）	B
0x8211	1	リクエスト取得・リクエストヘッダ解析完了直後（インプロセス HTTP サーバ経由）	A／B
0x8219※2	8	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のシリアライズ開始直前	A
0x8220	4	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のデシリアライズ開始直前	A
0x8221	2	データベースセッションフェイルオーバ機能による Web アプリケーション処理前のデータベースアクセス開始直前	A
0x8222※2	10	データベースセッションフェイルオーバ機能による Web アプリケーション処理後のデータベースアクセス開始直前	A
0x8300	12	リクエスト処理完了直後（Web サーバ経由）	A／B
0x8302	7	サーブレット／JSP 処理完了直後	A／B
0x8303	7	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8307	7	静的コンテンツ処理完了直後（DefaultServlet）	B
0x8311	12	リクエスト処理完了直後（インプロセス HTTP サーバ経由）	A／B
0x8316	7	エラーページに転送される時に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタ）	B
0x8319※2	9	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のシリアライズ終了直後	A
0x8320	5	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のデシリアライズ終了直後	A
0x8321	3	データベースセッションフェイルオーバ機能による Web アプリケーション処理前のデータベースアクセス終了直後	A
0x8322※2	11	データベースセッションフェイルオーバ機能による Web アプリケーション処理後のデータベースアクセス終了直後	A

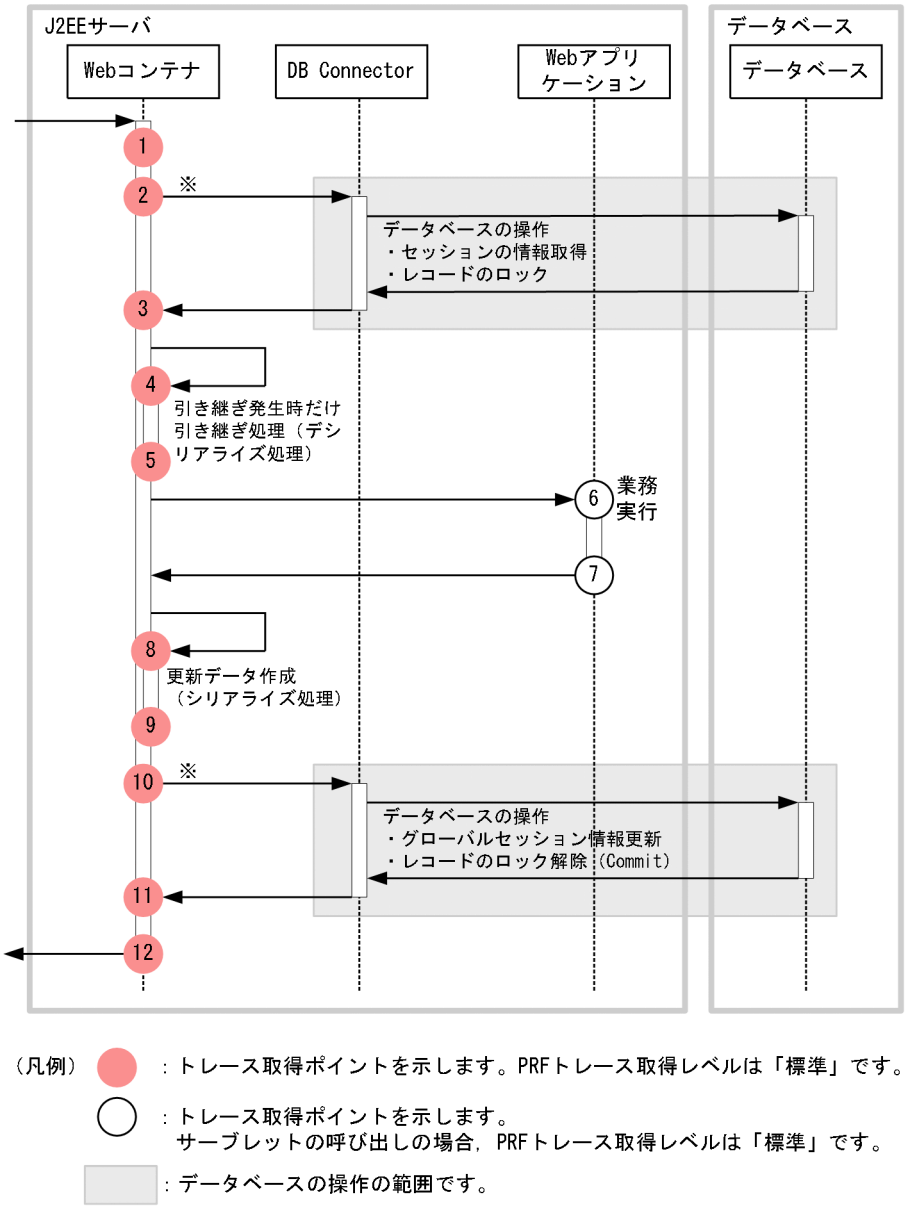
（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※1　図 14-13 中の番号と対応しています。

注※2　HTTP セッションの参照専用リクエストの場合，出力されません。

トレース取得ポイントを次の図に示します。

図 14-13　HTTP セッションを更新するリクエスト処理のトレース取得ポイント（データベースセッションフェイルオーバ機能）



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

：トレース取得ポイントを示します。
サーブレットの呼び出しの場合，PRFトレース取得レベルは「標準」です。

：データベースの操作の範囲です。

注※　実際のDB Connector呼び出しは，複数回発生します。

(2)　取得できるトレース情報

HTTP セッションを更新するリクエスト処理で取得できるトレース情報を次の表に示します。

表 14–23　HTTP セッションを更新するリクエスト処理で取得できるトレース情報（データベースセッションフェイルオーバ機能）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
	0x8211	A	HTTP メソッド	URI	－
		B			<セッション ID 文字数:セッション ID:取得方法>
2	0x8221	A	－	－	<セッション ID の文字数:リクエストで受信したセッション ID>
3	0x8321	A	完全性保障モードが有効な場合 排他を取得したレコード番号 完全性保障モードが無効な場合 -1	－	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
4	0x8220	A	リクエスト URL	デシリアライズ前の HTTP セッションの属性情報のサイズ（バイト）	<セッション ID 文字数:セッション ID>
5	0x8320	A	リクエスト URL	－	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
6	0x8202	A	クラス名 (JSP 呼び出し時は JSP ファイル名)	－	－
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8207	B	－	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
7	0x8302	A	クラス名 (JSP 呼び出し時は JSP ファイル名)	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8316	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
8	0x8219	A	リクエスト URL	−	<セッション ID 文字数:セッション ID>
9	0x8319	A	リクエスト URL	シリアライズ後の HTTP セッションの属性情報のサイズ(バイト)	• 正常時 <入り口時刻> • 例外発生時

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
					<入り口時刻><例外名>
10	0x8222	A	－	－	<セッション ID の文字数:HTTP セッションのセッション ID>
11	0x8322	A	完全性保障モードが有効な場合 排他を解放したレコード番号 完全性保障モードが無効な場合 -1	－	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
12	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-13 中の番号と対応しています。

14.10.3 HTTP セッションを無効化するリクエスト処理のトレース取得ポイントと取得できるトレース情報（データベースセッションフェイルオーバ機能のトレース）

HTTP セッションを無効化するリクエスト処理のトレース取得ポイントと取得できるトレース情報について説明します。

(1) トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-24 HTTP セッションを無効化するリクエスト処理のトレース取得ポイントの詳細（データベースセッションフェイルオーバ機能）

イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8200	1	リクエスト取得・リクエストヘッダ解析完了直後（Web サーバ経由）	A／B
0x8202	6	サーブレット／JSP 呼び出し直前	A／B
0x8203	6	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの呼び出し直前（web.xml の<filter-mapping>タグの<dispatcher>タグを省略，または<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8207	6	静的コンテンツ呼び出し直前（DefaultServlet）	B
0x8211	1	リクエスト取得・リクエストヘッダ解析完了直後（インプロセス HTTP サーバ経由）	A／B
0x8220	4	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のデシリアライズ開始直前	A
0x8221	2	データベースセッションフェイルオーバ機能による Web アプリケーション処理前のデータベースアクセス開始直前	A
0x8224	7	データベースセッションフェイルオーバ機能による HTTP セッション無効化時のデータベースアクセス開始直前	A
0x8300	10	リクエスト処理完了直後（Web サーバ経由）	A／B
0x8302	9	サーブレット／JSP 処理完了直後	A／B
0x8303	9	リクエストを受信したサーブレット／JSP の実行前に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"REQUEST"を指定したフィルタ）	B
0x8307	9	静的コンテンツ処理完了直後（DefaultServlet）	B
0x8311	10	リクエスト処理完了直後（インプロセス HTTP サーバ経由）	A／B
0x8316	9	エラーページに転送される際に実行されるフィルタの処理完了直後（web.xml の<filter-mapping>タグの<dispatcher>タグで"ERROR"を指定したフィルタ）	B

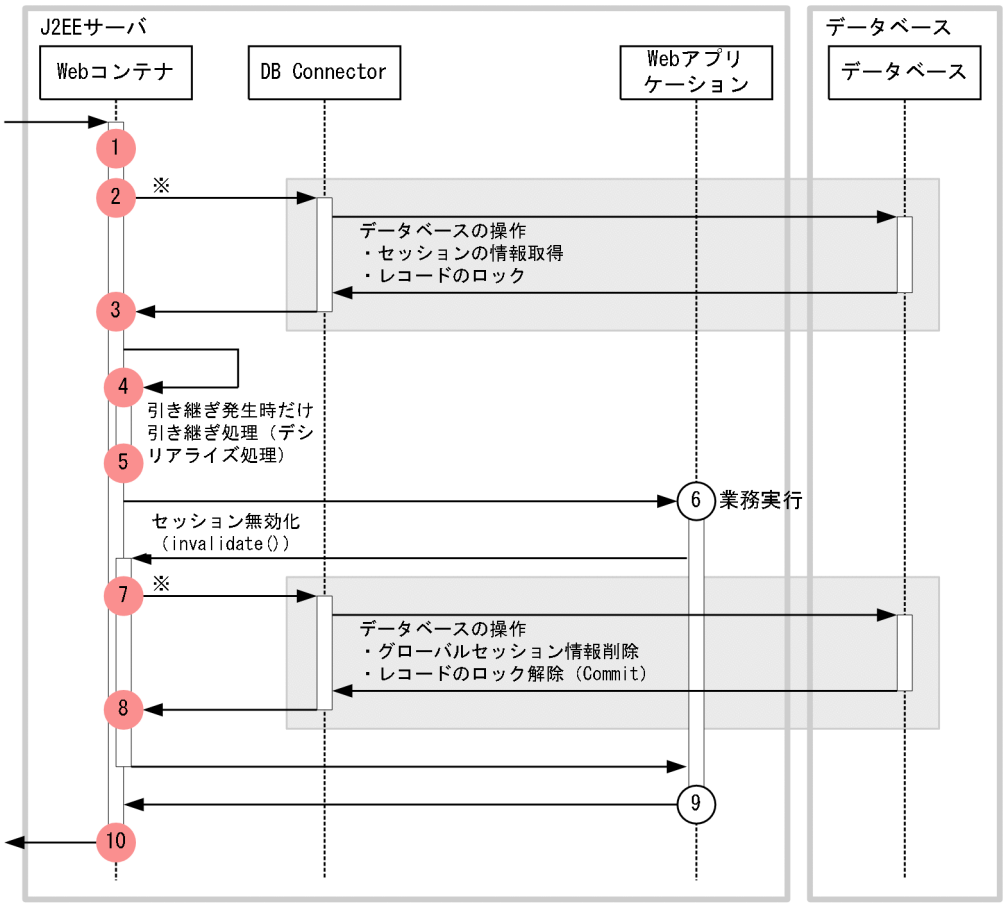
イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8320	5	データベースセッションフェイルオーバ機能による HTTP セッションの属性情報のデシリアライズ終了直後	A
0x8321	3	データベースセッションフェイルオーバ機能による Web アプリケーション処理前のデータベースアクセス終了直後	A
0x8324	8	データベースセッションフェイルオーバ機能による HTTP セッション無効化時のデータベースアクセス終了直後	A

（凡例）A：標準　B：詳細　A／B：標準と詳細で異なる情報を取得

注※　図 14-14 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-14　HTTP セッションを無効化するリクエスト処理のトレース取得ポイント（データベースセッションフェイルオーバ機能）



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

：トレース取得ポイントを示します。
サーブレットの呼び出しの場合，PRFトレース取得レベルは「標準」です。

：データベースの操作の範囲です。

注※　実際のDB Connector呼び出しは，複数回発生します。

(2) 取得できるトレース情報

HTTP セッションを無効化するリクエスト処理で取得できるトレース情報を次の表に示します。

表 14–25 HTTP セッションを無効化するリクエスト処理で取得できるトレース情報（データベースセッションフェイルオーバ機能）

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8200	A	HTTP メソッド	URI	−
		B			<セッション ID 文字数:セッション ID:取得方法>
	0x8211	A	HTTP メソッド	URI	−
		B			<セッション ID 文字数:セッション ID:取得方法>
2	0x8221	A	−	−	<セッション ID の文字数:リクエストで受信したセッション ID>
3	0x8321	A	完全性保障モードが有効な場合 排他を取得したレコード番号 完全性保障モードが無効な場合 -1	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
4	0x8220	A	リクエスト URL	デシリアライズ前の HTTP セッションの属性情報のサイズ（バイト）	<セッション ID 文字数:セッション ID>
5	0x8320	A	リクエスト URL	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
6	0x8202	A	クラス名 (JSP 呼び出し時は JSP ファイル名)	−	−
		B		コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8207	B	−	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0x8203	B	クラス名	コンテキストルート名	<セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
7	0x8224	A	−	−	<セッション ID の文字数:無効化した HTTP セッションのセッション ID>
8	0x8324	A	完全性保障モードが有効な場合 排他を解放したレコード番号 完全性保障モードが無効な場合 -1	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
9	0x8302	A	クラス名 (JSP 呼び出し時は JSP ファイル名)	−	• 正常時 <入り口時刻> • 例外発生時 <入り口時刻><例外名>
		B		コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8307	B	−	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
	0x8303	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0x8316	B	クラス名	コンテキストルート名	• 正常時 <入り口時刻><セッション ID 文字数:セッション ID> • 例外発生時 <入り口時刻><例外名:セッション ID 文字数:セッション ID>
10	0x8300	A	HTTP メソッド	URI	• 正常時 <入り口時刻><ステータスコード> • 例外発生時 <入り口時刻><ステータスコード><例外名>
		B			• 正常時 <入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID> • 例外発生時 <入り口時刻><ステータスコード><例外名:セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>
	0x8311	A	HTTP メソッド	URI	<入り口時刻><ステータスコード>
		B			<入り口時刻><ステータスコード><セッション ID 文字数:セッション ID:グローバルセッション ID 文字数:グローバルセッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-14 中の番号と対応しています。

14.10.4　有効期限監視で HTTP セッションを無効化するリクエスト処理のトレース取得ポイントと取得できるトレース情報（データベースセッションフェイルオーバ機能のトレース）

有効期限監視で HTTP セッションを無効化するリクエスト処理のトレース取得ポイントと取得できるトレース情報について説明します。

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–26　有効期限監視で HTTP セッションを無効化するリクエスト処理のトレース取得ポイントの詳細（データベースセッションフェイルオーバ機能）

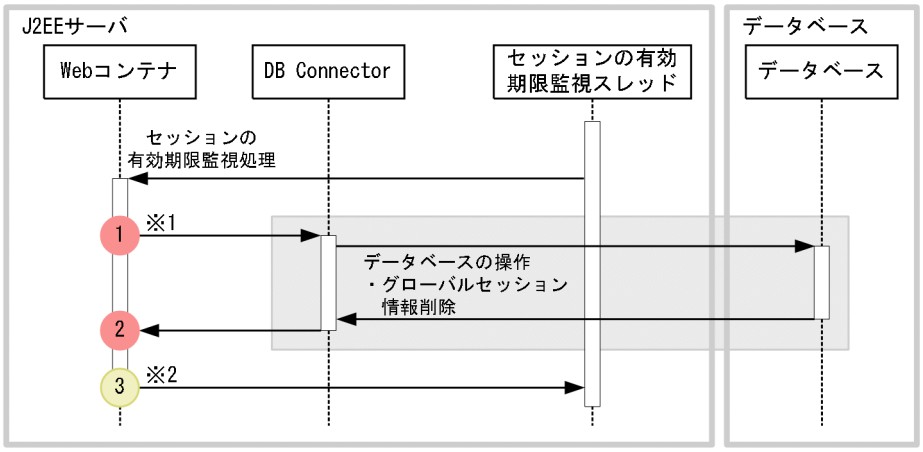
イベント ID	図中の番号※	トレース取得ポイント	レベル
0x8210	3	セッションタイムアウト後	B
0x8225	1	データベース上のグローバルセッション情報の有効期限監視処理開始直前	A
0x8325	2	データベース上のグローバルセッション情報の有効期限監視処理終了直後	A

（凡例）A：標準　B：詳細

注※　図 14-15 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-15　有効期限監視で HTTP セッションを無効化するリクエスト処理のトレース取得ポイント（データベースセッションフェイルオーバ機能）



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

：データベースの操作の範囲です。

注※1　実際のDB Connector呼び出しは，複数回発生します。

注※2　削除したHTTPセッションの個数分出力します。

(2)　取得できるトレース情報

有効期限監視で HTTP セッションを無効化するリクエスト処理で取得できるトレース情報を次の表に示します。

表 14-27　有効期限監視で HTTP セッションを無効化するリクエスト処理で取得できるトレース情報（データベースセッションフェイルオーバ機能）

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0x8225※2	A	−	−	−
2	0x8325※2	A	グローバルセッション情報の有効期限チェックを実施した場合 　無効化したグローバルセッションの個数 グローバルセッション情報の有効期限チェックを実施しなかった場合 　現在，有効期限チェックを担当している J2EE サーバの J2EE サーバ名（IP アドレス）	−	• 正常時 　<入り口時刻> • 例外発生時 　<入り口時刻><例外名>

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
3	0x8210	B	コンテキストルート名	セッション有効期間：セッション生成時刻	<セッション ID 文字数:セッション ID>

（凡例）A：標準　B：詳細　－：該当なし

注※1　図 14-15 中の番号と対応しています。

注※2　完全性保障モードが無効な場合は，出力されません。

14.11　JPA でのトレース取得ポイント

ここでは，JPA のトレース取得ポイントと，取得できるトレース情報について説明します。

14.11.1　アプリケーション管理の永続化コンテキストを利用した場合のトレース取得ポイントと取得できるトレース情報

アプリケーション管理の永続化コンテキストを利用した場合のトレース取得ポイントと取得できるトレース情報について説明します。

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–28　アプリケーション管理の永続化コンテキストを利用した場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA500	1	EntityManagerFactory#createEntityManagerFactory()の処理開始	A
0xA501	2	EntityManagerFactory#createEntityManagerFactory()の処理終了	A
0xA502	1	EntityManagerFactory#createEntityManagerFactory(Map map)の処理開始	A
0xA503	2	EntityManagerFactory#createEntityManagerFactory(Map map)の処理終了	A
0xA504	1	EntityManagerFactory#isOpen()の処理開始	A
0xA505	2	EntityManagerFactory#isOpen()の処理終了	A
0xA506	1	EntityManagerFactory#close()の処理開始	A
0xA507	2	EntityManagerFactory#close()の処理終了	A
0xA508	1	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA509	2	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50A	1	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA50B	2	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理終了	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA50C	1	EntityManager#contains(Object entity)の処理開始	A
0xA50D	2	EntityManager#contains(Object entity)の処理終了	A
0xA50E	1	EntityManager#lock(Object entity, LockModeType lockMode)の処理開始	A
0xA50F	2	EntityManager#lock(Object entity, LockModeType lockMode)の処理終了	A
0xA510	1	EntityManager#merge(T entity)の処理開始	A
0xA511	2	EntityManager#merge(T entity)の処理終了	A
0xA512	1	EntityManager#persist(Object entity)の処理開始	A
0xA513	2	EntityManager#persist(Object entity)の処理終了	A
0xA514	1	EntityManager#refresh(Object entity)の処理開始	A
0xA515	2	EntityManager#refresh(Object entity)の処理終了	A
0xA516	1	EntityManager#remove(Object entity)の処理開始	A
0xA517	2	EntityManager#remove(Object entity)の処理終了	A
0xA518	1	EntityManager#clear()の処理開始	A
0xA519	2	EntityManager#clear()の処理終了	A
0xA51A	1	EntityManager#flush()の処理開始	A
0xA51B	2	EntityManager#flush()の処理終了	A
0xA51C	1	EntityManager#createQuery(String qlString)の処理開始	A
0xA51D	2	EntityManager#createQuery(String qlString)の処理終了	A
0xA51E	1	EntityManager#createNamedQuery(String name)の処理開始	A
0xA51F	2	EntityManager#createNamedQuery(String name)の処理終了	A
0xA520	1	EntityManager#createNativeQuery(String sqlString)の処理開始	A
0xA521	2	EntityManager#createNativeQuery(String sqlString)の処理終了	A
0xA522	1	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理開始	A
0xA523	2	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理終了	A
0xA524	1	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理開始	A
0xA525	2	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理終了	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA526	1	EntityManager#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA527	2	EntityManager#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA528	1	EntityManager#getFlushMode()の処理開始	A
0xA529	2	EntityManager#getFlushMode()の処理終了	A
0xA52A	1	EntityManager#joinTransaction()の処理開始	A
0xA52B	2	EntityManager#joinTransaction()の処理終了	A
0xA52C	1	EntityManager#getTransaction()の処理開始	A
0xA52D	2	EntityManager#getTransaction()の処理終了	A
0xA52E	1	EntityManager#getDelegate()の処理開始	A
0xA52F	2	EntityManager#getDelegate()の処理終了	A
0xA530	1	EntityManager#isOpen()の処理開始	A
0xA531	2	EntityManager#isOpen()の処理終了	A
0xA532	1	EntityManager#close()の処理開始	A
0xA533	2	EntityManager#close()の処理終了	A
0xA534	1	EntityTransaction#begin()の処理開始	A
0xA535	2	EntityTransaction#begin()の処理終了	A
0xA536	1	EntityTransaction#commit()の処理開始	A
0xA537	2	EntityTransaction#commit()の処理終了	A
0xA538	1	EntityTransaction#rollback()の処理開始	A
0xA539	2	EntityTransaction#rollback()の処理終了	A
0xA53A	1	EntityTransaction#getRollbackOnly()の処理開始	A
0xA53B	2	EntityTransaction#getRollbackOnly()の処理終了	A
0xA53C	1	EntityTransaction#setRollbackOnly()の処理開始	A
0xA53D	2	EntityTransaction#setRollbackOnly()の処理終了	A
0xA53E	1	EntityTransaction#isActive()の処理開始	A
0xA53F	2	EntityTransaction#isActive()の処理終了	A
0xA540	1	Query#executeUpdate()の処理開始	A
0xA541	2	Query#executeUpdate()の処理終了	A
0xA542	1	Query#getResultList()の処理開始	A

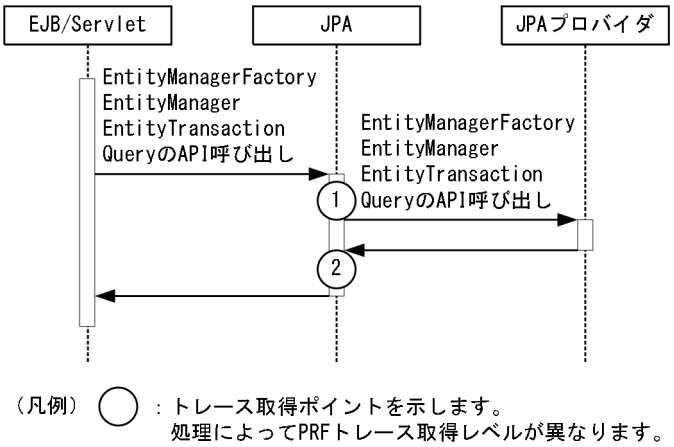
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA543	2	Query#getResultList()の処理終了	A
0xA544	1	Query#getSingleResult()の処理開始	A
0xA545	2	Query#getSingleResult()の処理終了	A
0xA546	1	Query#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA547	2	Query#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA548	1	Query#setFirstResult(int startPosition)の処理開始	B
0xA549	2	Query#setFirstResult(int startPosition)の処理終了	B
0xA54A	1	Query#setMaxResults(int maxResult)の処理開始	B
0xA54B	2	Query#setMaxResults(int maxResult)の処理終了	B
0xA54C	1	Query#setHint(String hintName, Object value)の処理開始	B
0xA54D	2	Query#setHint(String hintName, Object value)の処理終了	B
0xA54E	1	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA54F	2	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA550	1	Query#setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA551	2	Query#setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA552	1	Query#setParameter(int position, Object value)の処理開始	B
0xA553	2	Query#setParameter(int position, Object value)の処理終了	B
0xA554	1	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA555	2	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA556	1	Query#setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA557	2	Query#setParameter(String name, Date value, TemporalType temporalType)の処理終了	B
0xA558	1	Query#setParameter(String name, Object value)の処理開始	B
0xA559	2	Query#setParameter(String name, Object value)の処理終了	B

（凡例）A：標準　B：詳細

注※　図 14-16 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-16　アプリケーション管理の永続化コンテキストを利用した場合のトレース取得ポイント



(2)　取得できるトレース情報

アプリケーション管理の永続化コンテキストを利用した場合に取得できるトレース情報を次の表に示します。

表 14-29　アプリケーション管理の永続化コンテキストを利用した場合に取得できるトレース情報

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA500	A	−	−	−
	0xA502	A	−	−	−
	0xA504	A	−	−	−
	0xA506	A	−	−	−
	0xA508	A	entity のクラス名	−	−
	0xA50A	A	entity のクラス名	−	−
	0xA50C	A	entity のクラス名	−	−
	0xA50E	A	entity のクラス名	lockMode の値	−
	0xA510	A	entity のクラス名	−	−
	0xA512	A	entity のクラス名	−	−
	0xA514	A	entity のクラス名	−	−
	0xA516	A	entity のクラス名	−	−
	0xA518	A	−	−	−
	0xA51A	A	−	−	−
	0xA51C	A	−	−	−
	0xA51E	A	name	−	−
	0xA520	A	−	−	−

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA522	A	resultClass のクラス名	−	−
	0xA524	A	resultSetMapping	−	−
	0xA526	A	flushMode の値	−	−
	0xA528	A	−	−	−
	0xA52A	A	−	−	−
	0xA52C	A	−	−	−
	0xA52E	A	−	−	−
	0xA530	A	−	−	−
	0xA532	A	−	−	−
	0xA534	A	−	−	−
	0xA536	A	−	−	−
	0xA538	A	−	−	−
	0xA53A	A	−	−	−
	0xA53C	A	−	−	−
	0xA53E	A	−	−	−
	0xA540	A	−	−	−
	0xA542	A	−	−	−
	0xA544	A	−	−	−
	0xA546	A	flushMode の値	−	−
	0xA548	B	startPosition の値	−	−
	0xA54A	B	maxResult の値	−	−
	0xA54C	B	hintName	value のクラス名	−
	0xA54E	B	position の値	−	−
	0xA550	B	position の値	−	−
	0xA552	B	position の値	−	−
	0xA554	B	name の値	−	−
	0xA556	B	name の値	−	−
	0xA558	B	name の値	−	−
2	0xA501	A	−	−	• 正常時
	0xA503	A	−	−	

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA505	A	−	−	<入り口時刻> • 例外発生時 <入り口時刻><例外名>
	0xA507	A	−	−	
	0xA509	A	−	−	
	0xA50B	A	−	−	
	0xA50D	A	−	−	
	0xA50F	A	−	−	
	0xA511	A	−	−	
	0xA513	A	−	−	
	0xA515	A	−	−	
	0xA517	A	−	−	
	0xA519	A	−	−	
	0xA51B	A	−	−	
	0xA51D	A	−	−	
	0xA51F	A	−	−	
	0xA521	A	−	−	
	0xA523	A	−	−	
	0xA525	A	−	−	
	0xA527	A	−	−	
	0xA529	A	−	−	
	0xA52B	A	−	−	
	0xA52D	A	−	−	
	0xA52F	A	−	−	
	0xA531	A	−	−	
	0xA533	A	−	−	
	0xA535	A	−	−	
	0xA537	A	−	−	
	0xA539	A	−	−	
	0xA53B	A	−	−	
	0xA53D	A	−	−	
	0xA53F	A	−	−	

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA541	A	－	－	
	0xA543	A	－	－	
	0xA545	A	－	－	
	0xA547	A	－	－	
	0xA549	B	－	－	
	0xA54B	B	－	－	
	0xA54D	B	－	－	
	0xA54F	B	－	－	
	0xA551	B	－	－	
	0xA553	B	－	－	
	0xA555	B	－	－	
	0xA557	B	－	－	
	0xA559	B	－	－	

（凡例）A：標準　B：詳細　－：該当なし

注※　図 14-16 中の番号と対応しています。

14.11.2　コンテナ管理の永続化コンテキストを利用した場合のトレース取得ポイントと取得できるトレース情報

コンテナ管理の永続化コンテキストを利用した場合のトレース取得ポイントと取得できるトレース情報について説明します。ここでは，次の四つの場合に分けて説明します。

- トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合
- トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合
- トランザクション外で生成された Query をトランザクション外で利用した場合
- 拡張永続化コンテキストを利用した場合

(1)　トレース取得ポイントと PRF トレース取得レベル

(a)　トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-30　トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA508	1	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA509	6	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50A	1	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA50B	6	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50C	1	EntityManager#contains(Object entity)の処理開始	A
0xA50D	6	EntityManager#contains(Object entity)の処理終了	A
0xA50E	1	EntityManager#lock(Object entity, LockModeType lockMode)の処理開始	A
0xA50F	6	EntityManager#lock(Object entity, LockModeType lockMode)の処理終了	A
0xA510	1	EntityManager#merge(T entity)の処理開始	A
0xA511	6	EntityManager#merge(T entity)の処理終了	A
0xA512	1	EntityManager#persist(Object entity)の処理開始	A
0xA513	6	EntityManager#persist(Object entity)の処理終了	A
0xA514	1	EntityManager#refresh(Object entity)の処理開始	A
0xA515	6	EntityManager#refresh(Object entity)の処理終了	A
0xA516	1	EntityManager#remove(Object entity)の処理開始	A
0xA517	6	EntityManager#remove(Object entity)の処理終了	A
0xA518	1	EntityManager#clear()の処理開始	A
0xA519	6	EntityManager#clear()の処理終了	A
0xA51A	1	EntityManager#flush()の処理開始	A
0xA51B	6	EntityManager#flush()の処理終了	A
0xA51C	1	EntityManager#createQuery(String qlString)の処理開始	A
0xA51D	6	EntityManager#createQuery(String qlString)の処理終了	A
0xA51E	1	EntityManager#createNamedQuery(String name)の処理開始	A
0xA51F	6	EntityManager#createNamedQuery(String name)の処理終了	A
0xA520	1	EntityManager#createNativeQuery(String sqlString)の処理開始	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA521	6	EntityManager#createNativeQuery(String sqlString)の処理終了	A
0xA522	1	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理開始	A
0xA523	6	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理終了	A
0xA524	1	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理開始	A
0xA525	6	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理終了	A
0xA526	1	EntityManager#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA527	6	EntityManager#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA528	1	EntityManager#getFlushMode()の処理開始	A
0xA529	6	EntityManager#getFlushMode()の処理終了	A
0xA52A	1	EntityManager#joinTransaction()の処理開始	A
0xA52B	6	EntityManager#joinTransaction()の処理終了	A
0xA52C	1	EntityManager#getTransaction()の処理開始	A
0xA52D	6	EntityManager#getTransaction()の処理終了	A
0xA52E	1	EntityManager#getDelegate()の処理開始	A
0xA52F	6	EntityManager#getDelegate()の処理終了	A
0xA530	1	EntityManager#isOpen()の処理開始	A
0xA531	6	EntityManager#isOpen()の処理終了	A
0xA532	1	EntityManager#close()の処理開始	A
0xA533	6	EntityManager#close()の処理終了	A
0xA540	1	Query#executeUpdate()の処理開始	A
0xA541	6	Query#executeUpdate()の処理終了	A
0xA542	1	Query#getResultList()の処理開始	A
0xA543	6	Query#getResultList()の処理終了	A
0xA544	1	Query#getSingleResult()の処理開始	A
0xA545	6	Query#getSingleResult()の処理終了	A
0xA546	1	Query#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA547	6	Query#setFlushMode(FlushModeType flushMode)の処理終了	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA548	1	Query#setFirstResult(int startPosition)の処理開始	B
0xA549	6	Query#setFirstResult(int startPosition)の処理終了	B
0xA54A	1	Query#setMaxResults(int maxResult)の処理開始	B
0xA54B	6	Query#setMaxResults(int maxResult)の処理終了	B
0xA54C	1	Query#setHint(String hintName, Object value)の処理開始	B
0xA54D	6	Query#setHint(String hintName, Object value)の処理終了	B
0xA54E	1	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA54F	6	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA550	1	Query#setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA551	6	Query#setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA552	1	Query#setParameter(int position, Object value)の処理開始	B
0xA553	6	Query#setParameter(int position, Object value)の処理終了	B
0xA554	1	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA555	6	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA556	1	Query#setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA557	6	Query#setParameter(String name, Date value, TemporalType temporalType)の処理終了	B
0xA558	1	Query#setParameter(String name, Object value)の処理開始	B
0xA559	6	Query#setParameter(String name, Object value)の処理終了	B
0xA560	4	JPA プロバイダの EntityManager の find(Class<T> entityClass, Object primaryKey)の処理開始	B
0xA561	5	JPA プロバイダの EntityManager の find(Class<T> entityClass, Object primaryKey)の処理終了	B
0xA562	4	JPA プロバイダの EntityManager の getReference(Class<T> entityClass, Object primaryKey)の処理開始	B
0xA563	5	JPA プロバイダの EntityManager の getReference(Class<T> entityClass, Object primaryKey)の処理終了	B
0xA564	4	JPA プロバイダの EntityManager の contains(Object entity)の処理開始	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA565	5	JPA プロバイダの EntityManager の contains(Object entity)の処理終了	B
0xA566	4	JPA プロバイダの EntityManager の lock(Object entity, LockModeType lockMode)の処理開始	B
0xA567	5	JPA プロバイダの EntityManager の lock(Object entity, LockModeType lockMode)の処理終了	B
0xA568	4	JPA プロバイダの EntityManager の merge(T entity)の処理開始	B
0xA569	5	JPA プロバイダの EntityManager の merge(T entity)の処理終了	B
0xA56A	4	JPA プロバイダの EntityManager の persist(Object entity)の処理開始	B
0xA56B	5	JPA プロバイダの EntityManager の persist(Object entity)の処理終了	B
0xA56C	4	JPA プロバイダの EntityManager の refresh(Object entity)の処理開始	B
0xA56D	5	JPA プロバイダの EntityManager の refresh(Object entity)の処理終了	B
0xA56E	4	JPA プロバイダの EntityManager の remove(Object entity)の処理開始	B
0xA56F	5	JPA プロバイダの EntityManager の remove(Object entity)の処理終了	B
0xA570	4	JPA プロバイダの EntityManager の clear()の処理開始	B
0xA571	5	JPA プロバイダの EntityManager の clear()の処理終了	B
0xA572	4	JPA プロバイダの EntityManager の flush()の処理開始	B
0xA573	5	JPA プロバイダの EntityManager の flush()の処理終了	B
0xA574	4	JPA プロバイダの EntityManager の createQuery(String qlString)の処理開始	B
0xA575	5	JPA プロバイダの EntityManager の createQuery(String qlString)の処理終了	B
0xA576	4	JPA プロバイダの EntityManager の createNamedQuery(String name)の処理開始	B
0xA577	5	JPA プロバイダの EntityManager の createNamedQuery(String name)の処理終了	B
0xA578	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString)の処理開始	B
0xA579	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString)の処理終了	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA57A	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, Class resultClass)の処理開始	B
0xA57B	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, Class resultClass)の処理終了	B
0xA57C	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, String resultSetMapping)の処理開始	B
0xA57D	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, String resultSetMapping)の処理終了	B
0xA57E	4	JPA プロバイダの EntityManager の setFlushMode(FlushModeType flushMode)の処理開始	B
0xA57F	5	JPA プロバイダの EntityManager の setFlushMode(FlushModeType flushMode)の処理終了	B
0xA580	4	JPA プロバイダの EntityManager の getFlushMode()の処理開始	B
0xA581	5	JPA プロバイダの EntityManager の getFlushMode()の処理終了	B
0xA582	4	JPA プロバイダの EntityManager の joinTransaction()の処理開始	B
0xA583	5	JPA プロバイダの EntityManager の joinTransaction()の処理終了	B
0xA584	4	JPA プロバイダの EntityManager の isOpen()の処理開始	B
0xA585	5	JPA プロバイダの EntityManager の isOpen()の処理終了	B
0xA586	4	JPA プロバイダの Query の executeUpdate()の処理開始	B
0xA587	5	JPA プロバイダの Query の executeUpdate()の処理終了	B
0xA588	4	JPA プロバイダの Query の getResultList()の処理開始	B
0xA589	5	JPA プロバイダの Query の getResultList()の処理終了	B
0xA58A	4	JPA プロバイダの Query の getSingleResult()の処理開始	B
0xA58B	5	JPA プロバイダの Query の getSingleResult()の処理終了	B
0xA58C	4	JPA プロバイダの Query の setFlushMode(FlushModeType flushMode)の処理開始	B
0xA58D	5	JPA プロバイダの Query の setFlushMode(FlushModeType flushMode)の処理終了	B
0xA58E	4	JPA プロバイダの Query の setFirstResult(int startPosition)の処理開始	B
0xA58F	5	JPA プロバイダの Query の setFirstResult(int startPosition)の処理終了	B
0xA590	4	JPA プロバイダの Query の setMaxResults(int maxResult)の処理開始	B

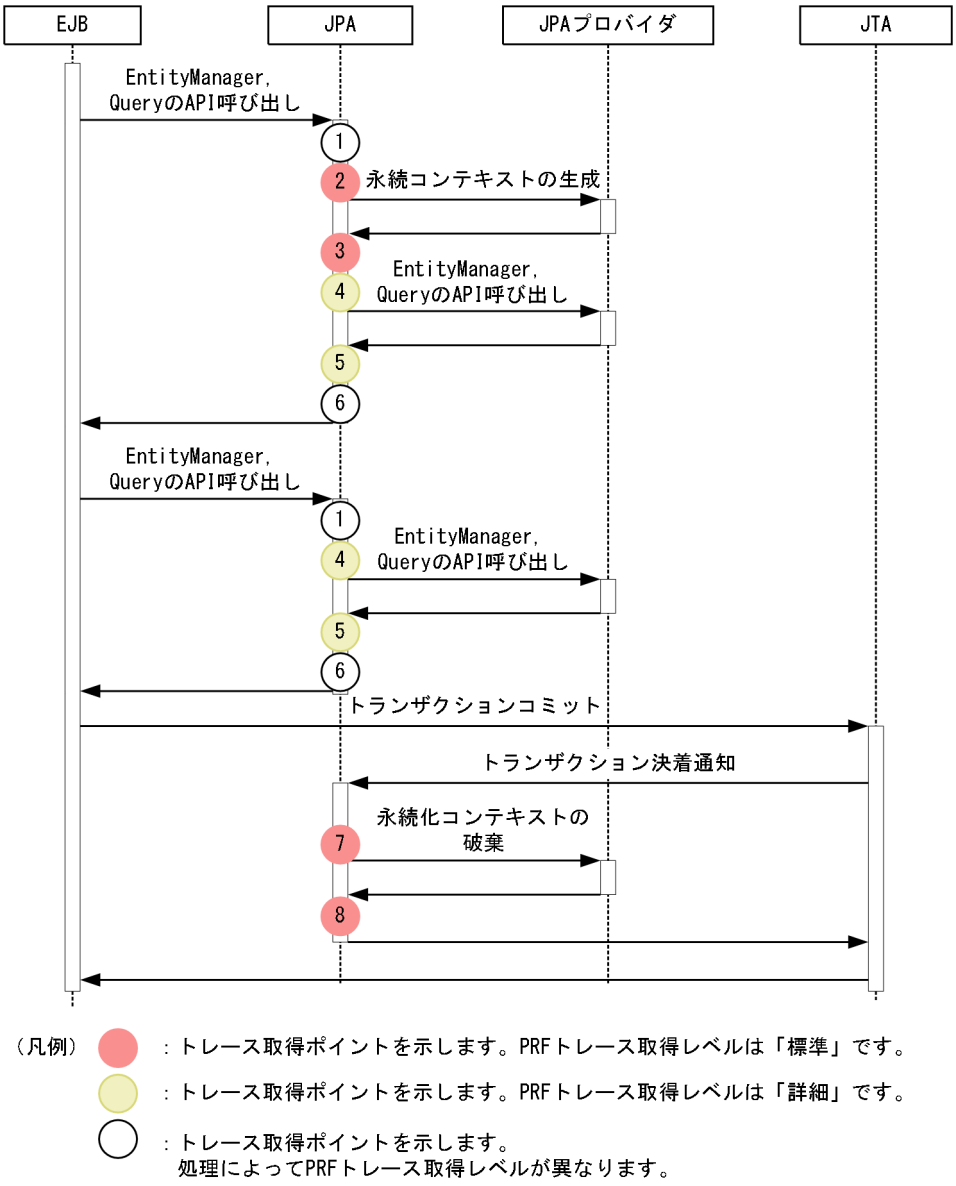
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA591	5	JPA プロバイダの Query の setMaxResults(int maxResult)の処理終了	B
0xA592	4	JPA プロバイダの Query の setHint(String hintName, Object value)の処理開始	B
0xA593	5	JPA プロバイダの Query の setHint(String hintName, Object value)の処理終了	B
0xA594	4	JPA プロバイダの Query の setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA595	5	JPA プロバイダの Query の setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA596	4	JPA プロバイダの Query の setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA597	5	JPA プロバイダの Query の setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA598	4	JPA プロバイダの Query の setParameter(int position, Object value)の処理開始	B
0xA599	5	JPA プロバイダの Query の setParameter(int position, Object value)の処理終了	B
0xA59A	4	JPA プロバイダの Query の setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA59B	5	JPA プロバイダの Query の setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA59C	4	JPA プロバイダの Query の setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA59D	5	JPA プロバイダの Query の setParameter(String name, Date value, TemporalType temporalType)の処理終了	B
0xA59E	4	JPA プロバイダの Query の setParameter(String name, Object value)の処理開始	B
0xA59F	5	JPA プロバイダの Query の setParameter(String name, Object value)の処理終了	B
0xA5A0	2	トランザクションスコープの永続化コンテキストを利用した際の永続化コンテキストの生成処理開始	A
0xA5A1	3	トランザクションスコープの永続化コンテキストを利用した際の永続化コンテキストの生成処理終了	A
0xA5A2	7	トランザクションスコープの永続化コンテキストを利用した際の永続化コンテキストの破棄処理開始	A
0xA5A3	8	トランザクションスコープの永続化コンテキストを利用した際の永続化コンテキストの破棄処理終了	A

（凡例）A：標準　B：詳細

注※　図 14-17 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-17　トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

：トレース取得ポイントを示します。
処理によってPRFトレース取得レベルが異なります。

(b)　トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–31　トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA508	1	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA509	8	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50A	1	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA50B	8	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50C	1	EntityManager#contains(Object entity)の処理開始	A
0xA50D	8	EntityManager#contains(Object entity)の処理終了	A
0xA50E	1	EntityManager#lock(Object entity, LockModeType lockMode)の処理開始	A
0xA50F	8	EntityManager#lock(Object entity, LockModeType lockMode)の処理終了	A
0xA510	1	EntityManager#merge(T entity)の処理開始	A
0xA511	8	EntityManager#merge(T entity)の処理終了	A
0xA512	1	EntityManager#persist(Object entity)の処理開始	A
0xA513	8	EntityManager#persist(Object entity)の処理終了	A
0xA514	1	EntityManager#refresh(Object entity)の処理開始	A
0xA515	8	EntityManager#refresh(Object entity)の処理終了	A
0xA516	1	EntityManager#remove(Object entity)の処理開始	A
0xA517	8	EntityManager#remove(Object entity)の処理終了	A
0xA518	1	EntityManager#clear()の処理開始	A
0xA519	8	EntityManager#clear()の処理終了	A
0xA51A	1	EntityManager#flush()の処理開始	A
0xA51B	8	EntityManager#flush()の処理終了	A
0xA526	1	EntityManager#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA527	8	EntityManager#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA528	1	EntityManager#getFlushMode()の処理開始	A
0xA529	8	EntityManager#getFlushMode()の処理終了	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA52A	1	EntityManager#joinTransaction()の処理開始	A
0xA52B	8	EntityManager#joinTransaction()の処理終了	A
0xA52C	1	EntityManager#getTransaction()の処理開始	A
0xA52D	8	EntityManager#getTransaction()の処理終了	A
0xA52E	1	EntityManager#getDelegate()の処理開始	A
0xA52F	8	EntityManager#getDelegate()の処理終了	A
0xA530	1	EntityManager#isOpen()の処理開始	A
0xA531	8	EntityManager#isOpen()の処理終了	A
0xA532	1	EntityManager#close()の処理開始	A
0xA533	8	EntityManager#close()の処理終了	A
0xA560	4	JPA プロバイダの EntityManager の find(Class<T> entityClass, Object primaryKey)の処理開始	B
0xA561	5	JPA プロバイダの EntityManager の find(Class<T> entityClass, Object primaryKey)の処理終了	B
0xA562	4	JPA プロバイダの EntityManager の getReference(Class<T> entityClass, Object primaryKey)の処理開始	B
0xA563	5	JPA プロバイダの EntityManager の getReference(Class<T> entityClass, Object primaryKey)の処理終了	B
0xA564	4	JPA プロバイダの EntityManager の contains(Object entity)の処理開始	B
0xA565	5	JPA プロバイダの EntityManager の contains(Object entity)の処理終了	B
0xA566	4	JPA プロバイダの EntityManager の lock(Object entity, LockModeType lockMode)の処理開始	B
0xA567	5	JPA プロバイダの EntityManager の lock(Object entity, LockModeType lockMode)の処理終了	B
0xA568	4	JPA プロバイダの EntityManager の merge(T entity)の処理開始	B
0xA569	5	JPA プロバイダの EntityManager の merge(T entity)の処理終了	B
0xA56A	4	JPA プロバイダの EntityManager の persist(Object entity)の処理開始	B
0xA56B	5	JPA プロバイダの EntityManager の persist(Object entity)の処理終了	B
0xA56C	4	JPA プロバイダの EntityManager の refresh(Object entity)の処理開始	B

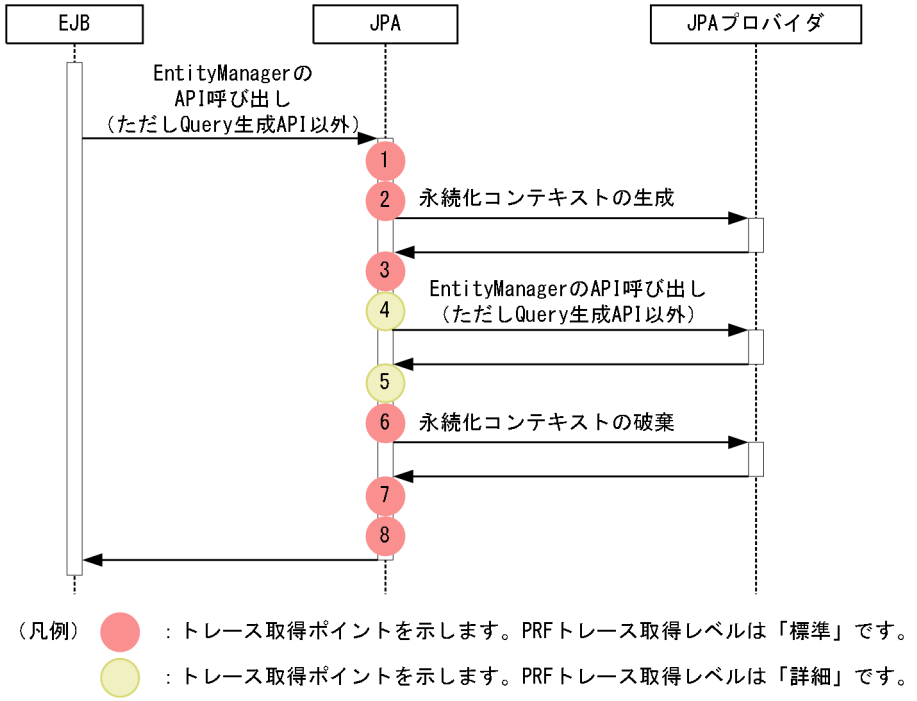
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA56D	5	JPA プロバイダの EntityManager の refresh(Object entity)の処理終了	B
0xA56E	4	JPA プロバイダの EntityManager の remove(Object entity)の処理開始	B
0xA56F	5	JPA プロバイダの EntityManager の remove(Object entity)の処理終了	B
0xA570	4	JPA プロバイダの EntityManager の clear()の処理開始	B
0xA571	5	JPA プロバイダの EntityManager の clear()の処理終了	B
0xA572	4	JPA プロバイダの EntityManager の flush()の処理開始	B
0xA573	5	JPA プロバイダの EntityManager の flush()の処理終了	B
0xA57E	4	JPA プロバイダの EntityManager の setFlushMode(FlushModeType flushMode)の処理開始	B
0xA57F	5	JPA プロバイダの EntityManager の setFlushMode(FlushModeType flushMode)の処理終了	B
0xA580	4	JPA プロバイダの EntityManager の getFlushMode()の処理開始	B
0xA581	5	JPA プロバイダの EntityManager の getFlushMode()の処理終了	B
0xA582	4	JPA プロバイダの EntityManager の joinTransaction()の処理開始	B
0xA583	5	JPA プロバイダの EntityManager の joinTransaction()の処理終了	B
0xA584	4	JPA プロバイダの EntityManager の isOpen()の処理開始	B
0xA585	5	JPA プロバイダの EntityManager の isOpen()の処理終了	B
0xA5A4	2	トランザクションスコープの EntityManager の Query 生成以外の API を，トランザクションの存在しない状況で利用した際の永続化コンテキストの生成処理開始	A
0xA5A5	3	トランザクションスコープの EntityManager の Query 生成以外の API を，トランザクションの存在しない状況で利用した際の永続化コンテキストの生成処理終了	A
0xA5A6	6	トランザクションスコープの EntityManager の Query 生成以外の API を，トランザクションの存在しない状況で利用した際に生成した永続化コンテキストの破棄処理開始	A
0xA5A7	7	トランザクションスコープの EntityManager の Query 生成以外の API を，トランザクションの存在しない状況で利用した際に生成した永続化コンテキストの破棄処理終了	A

（凡例）A：標準　B：詳細

注※　図 14-18 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-18　トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合のトレース取得ポイント



(c)　トランザクション外で生成された Query をトランザクション外で利用した場合

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-32　トランザクション外で生成された Query をトランザクション外で利用した場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA51C	1	EntityManager#createQuery(String qlString)の処理開始	A
0xA51D	6	EntityManager#createQuery(String qlString)の処理終了	A
0xA51E	1	EntityManager#createNamedQuery(String name)の処理開始	A
0xA51F	6	EntityManager#createNamedQuery(String name)の処理終了	A
0xA520	1	EntityManager#createNativeQuery(String sqlString)の処理開始	A
0xA521	6	EntityManager#createNativeQuery(String sqlString)の処理終了	A
0xA522	1	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理開始	A
0xA523	6	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理終了	A
0xA524	1	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理開始	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA525	6	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理終了	A
0xA540	1，11	Query#executeUpdate()の処理開始	A
0xA541	6，16	Query#executeUpdate()の処理終了	A
0xA542	1，11	Query#getResultList()の処理開始	A
0xA543	6，16	Query#getResultList()の処理終了	A
0xA544	1，11	Query#getSingleResult()の処理開始	A
0xA545	6，16	Query#getSingleResult()の処理終了	A
0xA546	1，7	Query#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA547	6，10	Query#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA548	1，7	Query#setFirstResult(int startPosition)の処理開始	B
0xA549	6，10	Query#setFirstResult(int startPosition)の処理終了	B
0xA54A	1，7	Query#setMaxResults(int maxResult)の処理開始	B
0xA54B	6，10	Query#setMaxResults(int maxResult)の処理終了	B
0xA54C	1，7	Query#setHint(String hintName, Object value)の処理開始	B
0xA54D	6，10	Query#setHint(String hintName, Object value)の処理終了	B
0xA54E	1，7	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA54F	6，10	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA550	1，7	Query#setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA551	6，10	Query#setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA552	1，7	Query#setParameter(int position, Object value)の処理開始	B
0xA553	6，10	Query#setParameter(int position, Object value)の処理終了	B
0xA554	1，7	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA555	6，10	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA556	1，7	Query#setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA557	6，10	Query#setParameter(String name, Date value, TemporalType temporalType)の処理終了	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA558	1，7	Query#setParameter(String name, Object value)の処理開始	B
0xA559	6，10	Query#setParameter(String name, Object value)の処理終了	B
0xA574	4	JPA プロバイダの EntityManager の createQuery(String qlString)の処理開始	B
0xA575	5	JPA プロバイダの EntityManager の createQuery(String qlString)の処理終了	B
0xA576	4	JPA プロバイダの EntityManager の createNamedQuery(String name)の処理開始	B
0xA577	5	JPA プロバイダの EntityManager の createNamedQuery(String name)の処理終了	B
0xA578	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString)の処理開始	B
0xA579	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString)の処理終了	B
0xA57A	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, Class resultClass)の処理開始	B
0xA57B	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, Class resultClass)の処理終了	B
0xA57C	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, String resultSetMapping)の処理開始	B
0xA57D	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, String resultSetMapping)の処理終了	B
0xA586	4，12	JPA プロバイダの Query の executeUpdate()の処理開始	B
0xA587	5，13	JPA プロバイダの Query の executeUpdate()の処理終了	B
0xA588	4，12	JPA プロバイダの Query の getResultList()の処理開始	B
0xA589	5，13	JPA プロバイダの Query の getResultList()の処理終了	B
0xA58A	4，12	JPA プロバイダの Query の getSingleResult()の処理開始	B
0xA58B	5，13	JPA プロバイダの Query の getSingleResult()の処理終了	B
0xA58C	4，8	JPA プロバイダの Query の setFlushMode(FlushModeType flushMode)の処理開始	B
0xA58D	5，9	JPA プロバイダの Query の setFlushMode(FlushModeType flushMode)の処理終了	B
0xA58E	4，8	JPA プロバイダの Query の setFirstResult(int startPosition)の処理開始	B
0xA58F	5，9	JPA プロバイダの Query の setFirstResult(int startPosition)の処理終了	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA590	4，8	JPA プロバイダの Query の setMaxResults(int maxResult)の処理開始	B
0xA591	5，9	JPA プロバイダの Query の setMaxResults(int maxResult)の処理終了	B
0xA592	4，8	JPA プロバイダの Query の setHint(String hintName, Object value)の処理開始	B
0xA593	5，9	JPA プロバイダの Query の setHint(String hintName, Object value)の処理終了	B
0xA594	4，8	JPA プロバイダの Query の setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA595	5，9	JPA プロバイダの Query の setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA596	4，8	JPA プロバイダの Query の setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA597	5，9	JPA プロバイダの Query の setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA598	4，8	JPA プロバイダの Query の setParameter(int position, Object value)の処理開始	B
0xA599	5，9	JPA プロバイダの Query の setParameter(int position, Object value)の処理終了	B
0xA59A	4，8	JPA プロバイダの Query の setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA59B	5，9	JPA プロバイダの Query の setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA59C	4，8	JPA プロバイダの Query の setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA59D	5，9	JPA プロバイダの Query の setParameter(String name, Date value, TemporalType temporalType)の処理終了	B
0xA59E	4，8	JPA プロバイダの Query の setParameter(String name, Object value)の処理開始	B
0xA59F	5，9	JPA プロバイダの Query の setParameter(String name, Object value)の処理終了	B
0xA5A8	2	トランザクションスコープの EntityManager の Query 生成 API または Query による検索・更新後に同一 Query オブジェクトの API を，トランザクションの存在しない状況で利用した際の永続化コンテキストの生成処理開始	A
0xA5A9	3	トランザクションスコープの EntityManager の Query 生成 API または Query による検索・更新後に同一 Query オブジェクトの API	A

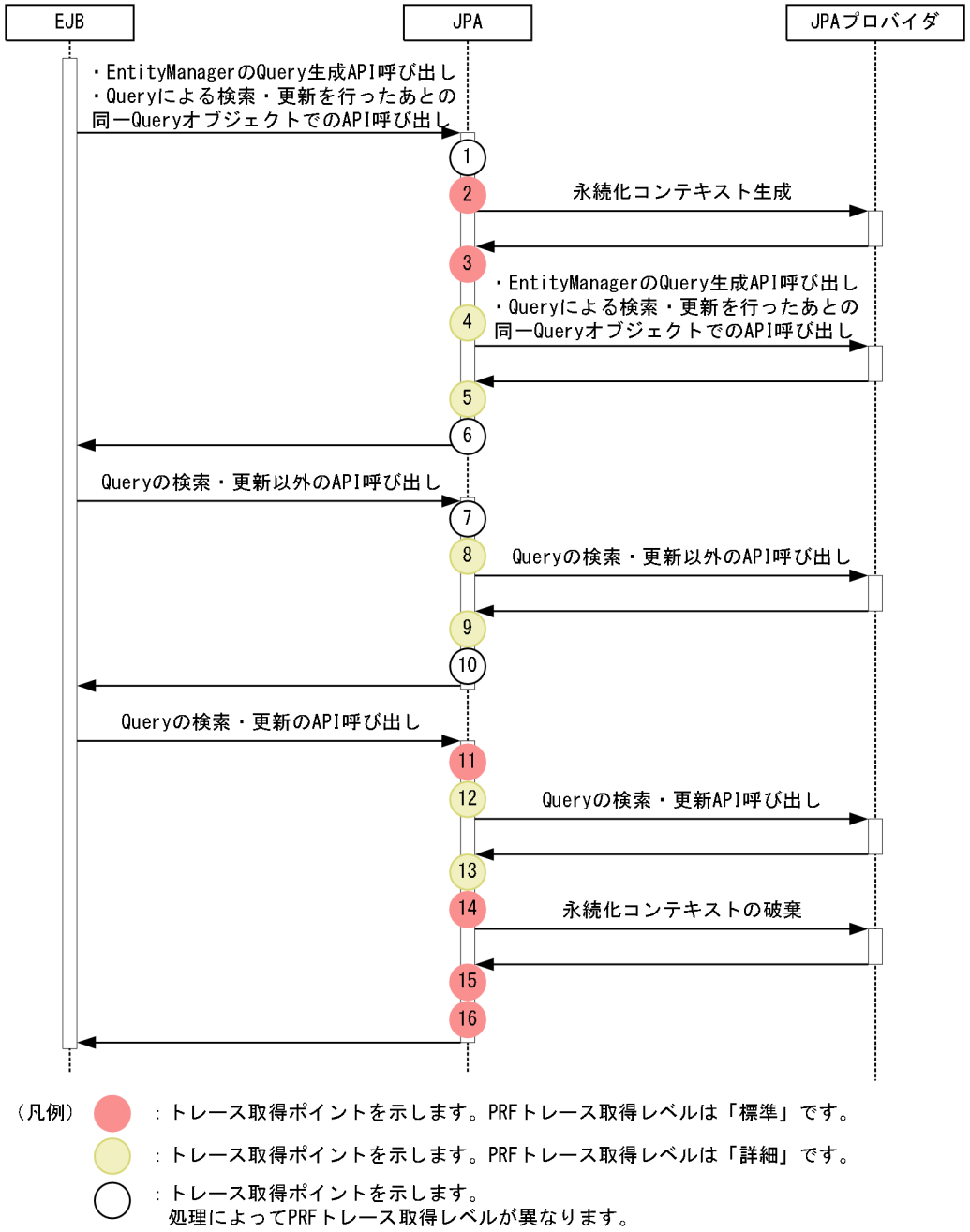
イベント ID	図中の番号※	トレース取得ポイント	レベル
		を，トランザクションの存在しない状況で利用した際の永続化コンテキストの生成処理終了	
0xA5AA	14	トランザクションスコープの EntityManager の Query 生成 API または Query による検索・更新後に同一 Query オブジェクトの API を，トランザクションの存在しない状況で利用した際に生成した永続化コンテキストの破棄処理開始	A
0xA5AB	15	トランザクションスコープの EntityManager の Query 生成 API または Query による検索・更新後に同一 Query オブジェクトの API を，トランザクションの存在しない状況で利用した際に生成した永続化コンテキストの破棄処理終了	A

（凡例）A：標準　B：詳細

注※　図 14-19 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-19　トランザクション外で生成された Query をトランザクション外で利用した場合のトレース取得ポイント



(d)　拡張永続化コンテキストを利用した場合

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-33　拡張永続化コンテキストを利用した場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA508	3	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理開始	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA509	6	EntityManager#find(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50A	3	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理開始	A
0xA50B	6	EntityManager#getReference(Class<T> entityClass, Object primaryKey)の処理終了	A
0xA50C	3	EntityManager#contains(Object entity)の処理開始	A
0xA50D	6	EntityManager#contains(Object entity)の処理終了	A
0xA50E	3	EntityManager#lock(Object entity, LockModeType lockMode)の処理開始	A
0xA50F	6	EntityManager#lock(Object entity, LockModeType lockMode)の処理終了	A
0xA510	3	EntityManager#merge(T entity)の処理開始	A
0xA511	6	EntityManager#merge(T entity)の処理終了	A
0xA512	3	EntityManager#persist(Object entity)の処理開始	A
0xA513	6	EntityManager#persist(Object entity)の処理終了	A
0xA514	3	EntityManager#refresh(Object entity)の処理開始	A
0xA515	6	EntityManager#refresh(Object entity)の処理終了	A
0xA516	3	EntityManager#remove(Object entity)の処理開始	A
0xA517	6	EntityManager#remove(Object entity)の処理終了	A
0xA518	3	EntityManager#clear()の処理開始	A
0xA519	6	EntityManager#clear()の処理終了	A
0xA51A	3	EntityManager#flush()の処理開始	A
0xA51B	6	EntityManager#flush()の処理終了	A
0xA51C	3	EntityManager#createQuery(String qlString)の処理開始	A
0xA51D	6	EntityManager#createQuery(String qlString)の処理終了	A
0xA51E	3	EntityManager#createNamedQuery(String name)の処理開始	A
0xA51F	6	EntityManager#createNamedQuery(String name)の処理終了	A
0xA520	3	EntityManager#createNativeQuery(String sqlString)の処理開始	A
0xA521	6	EntityManager#createNativeQuery(String sqlString)の処理終了	A
0xA522	3	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理開始	A

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA523	6	EntityManager#createNativeQuery(String sqlString, Class resultClass)の処理終了	A
0xA524	3	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理開始	A
0xA525	6	EntityManager#createNativeQuery(String sqlString, String resultSetMapping)の処理終了	A
0xA526	3	EntityManager#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA527	6	EntityManager#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA528	3	EntityManager#getFlushMode()の処理開始	A
0xA529	6	EntityManager#getFlushMode()の処理終了	A
0xA52A	3	EntityManager#joinTransaction()の処理開始	A
0xA52B	6	EntityManager#joinTransaction()の処理終了	A
0xA52C	3	EntityManager#getTransaction()の処理開始	A
0xA52D	6	EntityManager#getTransaction()の処理終了	A
0xA52E	3	EntityManager#getDelegate()の処理開始	A
0xA52F	6	EntityManager#getDelegate()の処理終了	A
0xA530	3	EntityManager#isOpen()の処理開始	A
0xA531	6	EntityManager#isOpen()の処理終了	A
0xA532	3	EntityManager#close()の処理開始	A
0xA533	6	EntityManager#close()の処理終了	A
0xA540	3	Query#executeUpdate()の処理開始	A
0xA541	6	Query#executeUpdate()の処理終了	A
0xA542	3	Query#getResultList()の処理開始	A
0xA543	6	Query#getResultList()の処理終了	A
0xA544	3	Query#getSingleResult()の処理開始	A
0xA545	6	Query#getSingleResult()の処理終了	A
0xA546	3	Query#setFlushMode(FlushModeType flushMode)の処理開始	A
0xA547	6	Query#setFlushMode(FlushModeType flushMode)の処理終了	A
0xA548	3	Query#setFirstResult(int startPosition)の処理開始	B
0xA549	6	Query#setFirstResult(int startPosition)の処理終了	B
0xA54A	3	Query#setMaxResults(int maxResult)の処理開始	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA54B	6	Query#setMaxResults(int maxResult)の処理終了	B
0xA54C	3	Query#setHint(String hintName, Object value)の処理開始	B
0xA54D	6	Query#setHint(String hintName, Object value)の処理終了	B
0xA54E	3	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA54F	6	Query#setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA550	3	Query#setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA551	6	Query#setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA552	3	Query#setParameter(int position, Object value)の処理開始	B
0xA553	6	Query#setParameter(int position, Object value)の処理終了	B
0xA554	3	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA555	6	Query#setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA556	3	Query#setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA557	6	Query#setParameter(String name, Date value, TemporalType temporalType)の処理終了	B
0xA558	3	Query#setParameter(String name, Object value)の処理開始	B
0xA559	6	Query#setParameter(String name, Object value)の処理終了	B
0xA560	4	JPA プロバイダの EntityManager の find(Class<T> entityClass, Object primaryKey)の処理開始	B
0xA561	5	JPA プロバイダの EntityManager の find(Class<T> entityClass, Object primaryKey)の処理終了	B
0xA562	4	JPA プロバイダの EntityManager の getReference(Class<T> entityClass, Object primaryKey)の処理開始	B
0xA563	5	JPA プロバイダの EntityManager の getReference(Class<T> entityClass, Object primaryKey)の処理終了	B
0xA564	4	JPA プロバイダの EntityManager の contains(Object entity)の処理開始	B
0xA565	5	JPA プロバイダの EntityManager の contains(Object entity)の処理終了	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA566	4	JPA プロバイダの EntityManager の lock(Object entity, LockModeType lockMode)の処理開始	B
0xA567	5	JPA プロバイダの EntityManager の lock(Object entity, LockModeType lockMode)の処理終了	B
0xA568	4	JPA プロバイダの EntityManager の merge(T entity)の処理開始	B
0xA569	5	JPA プロバイダの EntityManager の merge(T entity)の処理終了	B
0xA56A	4	JPA プロバイダの EntityManager の persist(Object entity)の処理開始	B
0xA56B	5	JPA プロバイダの EntityManager の persist(Object entity)の処理終了	B
0xA56C	4	JPA プロバイダの EntityManager の refresh(Object entity)の処理開始	B
0xA56D	5	JPA プロバイダの EntityManager の refresh(Object entity)の処理終了	B
0xA56E	4	JPA プロバイダの EntityManager の remove(Object entity)の処理開始	B
0xA56F	5	JPA プロバイダの EntityManager の remove(Object entity)の処理終了	B
0xA570	4	JPA プロバイダの EntityManager の clear()の処理開始	B
0xA571	5	JPA プロバイダの EntityManager の clear()の処理終了	B
0xA572	4	JPA プロバイダの EntityManager の flush()の処理開始	B
0xA573	5	JPA プロバイダの EntityManager の flush()の処理終了	B
0xA574	4	JPA プロバイダの EntityManager の createQuery(String qlString)の処理開始	B
0xA575	5	JPA プロバイダの EntityManager の createQuery(String qlString)の処理終了	B
0xA576	4	JPA プロバイダの EntityManager の createNamedQuery(String name)の処理開始	B
0xA577	5	JPA プロバイダの EntityManager の createNamedQuery(String name)の処理終了	B
0xA578	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString)の処理開始	B
0xA579	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString)の処理終了	B
0xA57A	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, Class resultClass)の処理開始	B

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA57B	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, Class resultClass)の処理終了	B
0xA57C	4	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, String resultSetMapping)の処理開始	B
0xA57D	5	JPA プロバイダの EntityManager の createNativeQuery(String sqlString, String resultSetMapping)の処理終了	B
0xA57E	4	JPA プロバイダの EntityManager の setFlushMode(FlushModeType flushMode)の処理開始	B
0xA57F	5	JPA プロバイダの EntityManager の setFlushMode(FlushModeType flushMode)の処理終了	B
0xA580	4	JPA プロバイダの EntityManager の getFlushMode()の処理開始	B
0xA581	5	JPA プロバイダの EntityManager の getFlushMode()の処理終了	B
0xA582	4	JPA プロバイダの EntityManager の joinTransaction()の処理開始	B
0xA583	5	JPA プロバイダの EntityManager の joinTransaction()の処理終了	B
0xA584	4	JPA プロバイダの EntityManager の isOpen()の処理開始	B
0xA585	5	JPA プロバイダの EntityManager の isOpen()の処理終了	B
0xA586	4	JPA プロバイダの Query の executeUpdate()の処理開始	B
0xA587	5	JPA プロバイダの Query の executeUpdate()の処理終了	B
0xA588	4	JPA プロバイダの Query の getResultList()の処理開始	B
0xA589	5	JPA プロバイダの Query の getResultList()の処理終了	B
0xA58A	4	JPA プロバイダの Query の getSingleResult()の処理開始	B
0xA58B	5	JPA プロバイダの Query の getSingleResult()の処理終了	B
0xA58C	4	JPA プロバイダの Query の setFlushMode(FlushModeType flushMode)の処理開始	B
0xA58D	5	JPA プロバイダの Query の setFlushMode(FlushModeType flushMode)の処理終了	B
0xA58E	4	JPA プロバイダの Query の setFirstResult(int startPosition)の処理開始	B
0xA58F	5	JPA プロバイダの Query の setFirstResult(int startPosition)の処理終了	B
0xA590	4	JPA プロバイダの Query の setMaxResults(int maxResult)の処理開始	B
0xA591	5	JPA プロバイダの Query の setMaxResults(int maxResult)の処理終了	B

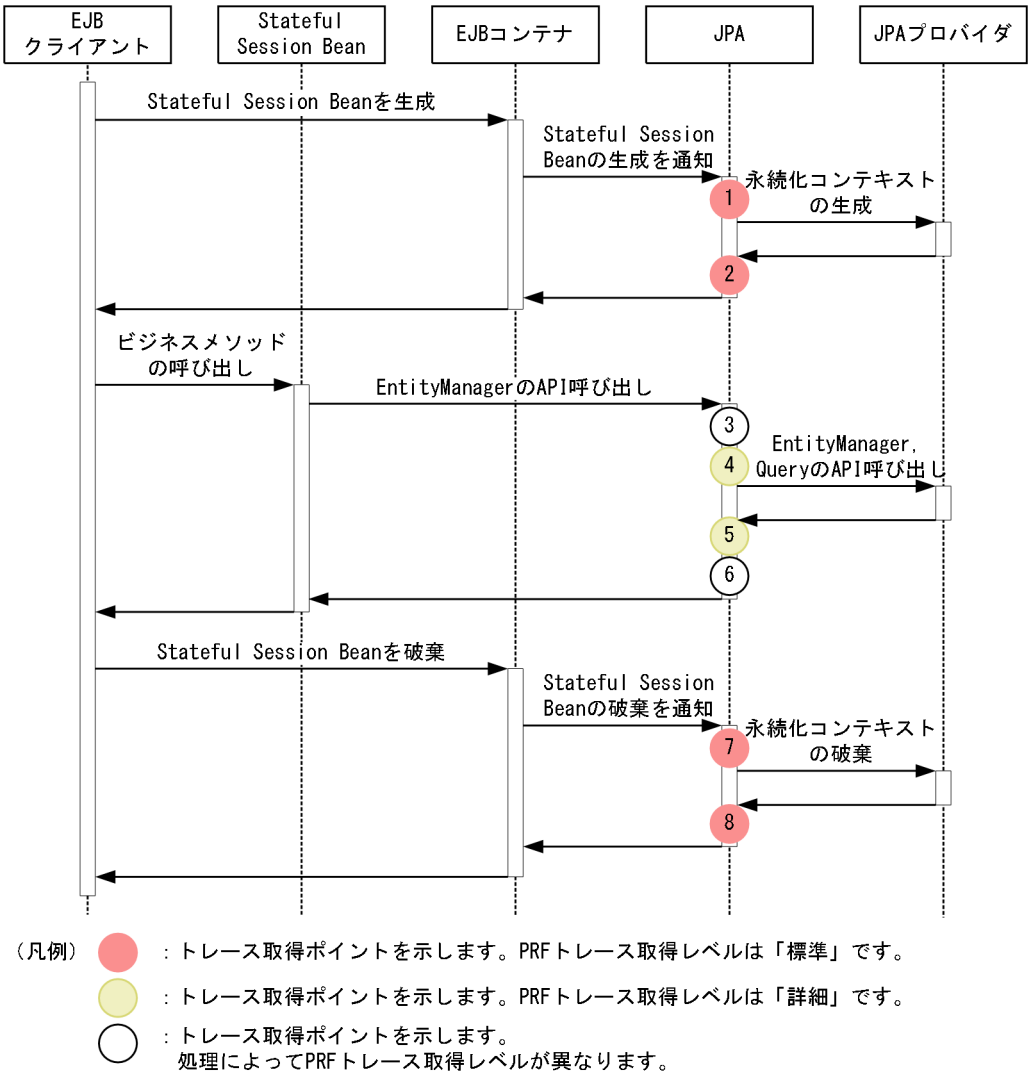
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA592	4	JPA プロバイダの Query の setHint(String hintName, Object value)の処理開始	B
0xA593	5	JPA プロバイダの Query の setHint(String hintName, Object value)の処理終了	B
0xA594	4	JPA プロバイダの Query の setParameter(int position, Calendar value, TemporalType temporalType)の処理開始	B
0xA595	5	JPA プロバイダの Query の setParameter(int position, Calendar value, TemporalType temporalType)の処理終了	B
0xA596	4	JPA プロバイダの Query の setParameter(int position, Date value, TemporalType temporalType)の処理開始	B
0xA597	5	JPA プロバイダの Query の setParameter(int position, Date value, TemporalType temporalType)の処理終了	B
0xA598	4	JPA プロバイダの Query の setParameter(int position, Object value)の処理開始	B
0xA599	5	JPA プロバイダの Query の setParameter(int position, Object value)の処理終了	B
0xA59A	4	JPA プロバイダの Query の setParameter(String name, Calendar value, TemporalType temporalType)の処理開始	B
0xA59B	5	JPA プロバイダの Query の setParameter(String name, Calendar value, TemporalType temporalType)の処理終了	B
0xA59C	4	JPA プロバイダの Query の setParameter(String name, Date value, TemporalType temporalType)の処理開始	B
0xA59D	5	JPA プロバイダの Query の setParameter(String name, Date value, TemporalType temporalType)の処理終了	B
0xA59E	4	JPA プロバイダの Query の setParameter(String name, Object value)の処理開始	B
0xA59F	5	JPA プロバイダの Query の setParameter(String name, Object value)の処理終了	B
0xA5AC	1	拡張スコープの永続化コンテキストを利用した際の永続化コンテキストの生成処理開始	A
0xA5AD	2	拡張スコープの永続化コンテキストを利用した際の永続化コンテキストの生成処理終了	A
0xA5AE	7	拡張スコープの永続化コンテキストを利用した際の EntityManager.close()の処理開始	A
0xA5AF	8	拡張スコープの永続化コンテキストを利用した際の EntityManager.close()の処理終了	A

（凡例）A：標準　B：詳細

注※　図 14-20 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-20　拡張永続化コンテキストを利用した場合のトレース取得ポイント



(2)　取得できるトレース情報

(a)　トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合

トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合に取得できるトレース情報を次の表に示します。

表 14-34　トランザクションスコープの永続化コンテキストをトランザクション内で利用した場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA508	A	entity のクラス名	−	−
	0xA50A	A	entity のクラス名	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA50C	A	entity のクラス名	−	−
	0xA50E	A	entity のクラス名	lockMode の値	−
	0xA510	A	entity のクラス名	−	−
	0xA512	A	entity のクラス名	−	−
	0xA514	A	entity のクラス名	−	−
	0xA516	A	entity のクラス名	−	−
	0xA518	A	−	−	−
	0xA51A	A	−	−	−
	0xA51C	A	−	−	−
	0xA51E	A	name	−	−
	0xA520	A	−	−	−
	0xA522	A	resultClass のクラス名	−	−
	0xA524	A	resultSetMapping	−	−
	0xA526	A	flushMode の値	−	−
	0xA528	A	−	−	−
	0xA52A	A	−	−	−
	0xA52C	A	−	−	−
	0xA52E	A	−	−	−
	0xA530	A	−	−	−
	0xA532	A	−	−	−
	0xA540	A	−	−	−
	0xA542	A	−	−	−
	0xA544	A	−	−	−
	0xA546	A	flushMode の値	−	−
	0xA548	B	startPosition の値	−	−
	0xA54A	B	maxResult の値	−	−
	0xA54C	B	hintName	value のクラス名	−
	0xA54E	B	position の値	−	−
	0xA550	B	position の値	−	−
	0xA552	B	position の値	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA554	B	name の値	−	−
	0xA556	B	name の値	−	−
	0xA558	B	name の値	−	−
2	0xA5A0	A	−	−	−
3	0xA5A1	A	−	−	※2
4	0xA560	B	entity のクラス名	−	−
	0xA562	B	entity のクラス名	−	−
	0xA564	B	entity のクラス名	−	−
	0xA566	B	entity のクラス名	lockMode の値	−
	0xA568	B	entity のクラス名	−	−
	0xA56A	B	entity のクラス名	−	−
	0xA56C	B	entity のクラス名	−	−
	0xA56E	B	entity のクラス名	−	−
	0xA570	B	−	−	−
	0xA572	B	−	−	−
	0xA574	B	−	−	−
	0xA576	B	name	−	−
	0xA578	B	−	−	−
	0xA57A	B	resultClass のクラス名	−	−
	0xA57C	B	resultSetMapping	−	−
	0xA57E	B	flushMode の値	−	−
	0xA580	B	−	−	−
	0xA582	B	−	−	−
	0xA584	B	−	−	−
	0xA586	B	−	−	−
	0xA588	B	−	−	−
	0xA58A	B	−	−	−
	0xA58C	B	flushMode の値	−	−
	0xA58E	B	startPosition の値	−	−
	0xA590	B	maxResult の値	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA592	B	hintName	value のクラス名	−
	0xA594	B	position の値	−	−
	0xA596	B	position の値	−	−
	0xA598	B	position の値	−	−
	0xA59A	B	name の値	−	−
	0xA59C	B	name の値	−	−
	0xA59E	B	name の値	−	−
5	0xA561	B	−	−	※2
	0xA563	B	−	−	※2
	0xA565	B	−	−	※2
	0xA567	B	−	−	※2
	0xA569	B	−	−	※2
	0xA56B	B	−	−	※2
	0xA56D	B	−	−	※2
	0xA56F	B	−	−	※2
	0xA571	B	−	−	※2
	0xA573	B	−	−	※2
	0xA575	B	−	−	※2
	0xA577	B	−	−	※2
	0xA579	B	−	−	※2
	0xA57B	B	−	−	※2
	0xA57D	B	−	−	※2
	0xA57F	B	−	−	※2
	0xA581	B	−	−	※2
	0xA583	B	−	−	※2
	0xA585	B	−	−	※2
	0xA587	B	−	−	※2
	0xA589	B	−	−	※2
	0xA58B	B	−	−	※2
	0xA58D	B	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA58F	B	−	−	※2
	0xA591	B	−	−	※2
	0xA593	B	−	−	※2
	0xA595	B	−	−	※2
	0xA597	B	−	−	※2
	0xA599	B	−	−	※2
	0xA59B	B	−	−	※2
	0xA59D	B	−	−	※2
	0xA59F	B	−	−	※2
6	0xA509	A	−	−	※2
	0xA50B	A	−	−	※2
	0xA50D	A	−	−	※2
	0xA50F	A	−	−	※2
	0xA511	A	−	−	※2
	0xA513	A	−	−	※2
	0xA515	A	−	−	※2
	0xA517	A	−	−	※2
	0xA519	A	−	−	※2
	0xA51B	A	−	−	※2
	0xA51D	A	−	−	※2
	0xA51F	A	−	−	※2
	0xA521	A	−	−	※2
	0xA523	A	−	−	※2
	0xA525	A	−	−	※2
	0xA527	A	−	−	※2
	0xA529	A	−	−	※2
	0xA52B	A	−	−	※2
	0xA52D	A	−	−	※2
	0xA52F	A	−	−	※2
	0xA531	A	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA533	A	−	−	※2
	0xA541	A	−	−	※2
	0xA543	A	−	−	※2
	0xA545	A	−	−	※2
	0xA547	A	−	−	※2
	0xA549	B	−	−	※2
	0xA54B	B	−	−	※2
	0xA54D	B	−	−	※2
	0xA54F	B	−	−	※2
	0xA551	B	−	−	※2
	0xA553	B	−	−	※2
	0xA555	B	−	−	※2
	0xA557	B	−	−	※2
	0xA559	B	−	−	※2
7	0xA5A2	A	−	−	−
8	0xA5A3	A	−	−	※2

（凡例）A：標準　B：詳細　−：該当なし

注※1　図 14-17 中の番号と対応しています。

注※2　正常に処理された場合，入り口時刻が表示されます。例外が発生した場合，入り口時刻と例外が表示されます。

(b)　トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合

トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合に取得できるトレース情報を次の表に示します。

表 14-35　トランザクションスコープの永続化コンテキストに関連づいているエンティティマネージャをトランザクション外で利用した場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA508	A	entity のクラス名	−	−
	0xA50A	A	entity のクラス名	−	−
	0xA50C	A	entity のクラス名	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA50E	A	entity のクラス名	lockMode の値	−
	0xA510	A	entity のクラス名	−	−
	0xA512	A	entity のクラス名	−	−
	0xA514	A	entity のクラス名	−	−
	0xA516	A	entity のクラス名	−	−
	0xA518	A	−	−	−
	0xA51A	A	−	−	−
	0xA526	A	flushMode の値	−	−
	0xA528	A	−	−	−
	0xA52A	A	−	−	−
	0xA52C	A	−	−	−
	0xA52E	A	−	−	−
	0xA530	A	−	−	−
	0xA532	A	−	−	−
2	0xA5A4	A	−	−	−
3	0xA5A5	A	−	−	※2
4	0xA560	B	entity のクラス名	−	−
	0xA562	B	entity のクラス名	−	−
	0xA564	B	entity のクラス名	−	−
	0xA566	B	entity のクラス名	lockMode の値	−
	0xA568	B	entity のクラス名	−	−
	0xA56A	B	entity のクラス名	−	−
	0xA56C	B	entity のクラス名	−	−
	0xA56E	B	entity のクラス名	−	−
	0xA570	B	−	−	−
	0xA572	B	−	−	−
	0xA57E	B	flushMode の値	−	−
	0xA580	B	−	−	−
	0xA582	B	−	−	−
	0xA584	B	−	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
5	0xA561	B	−	−	※2
	0xA563	B	−	−	※2
	0xA565	B	−	−	※2
	0xA567	B	−	−	※2
	0xA569	B	−	−	※2
	0xA56B	B	−	−	※2
	0xA56D	B	−	−	※2
	0xA56F	B	−	−	※2
	0xA571	B	−	−	※2
	0xA573	B	−	−	※2
	0xA57F	B	−	−	※2
	0xA581	B	−	−	※2
	0xA583	B	−	−	※2
	0xA585	B	−	−	※2
6	0xA5A6	A	−	−	−
7	0xA5A7	A	−	−	※2
8	0xA509	A	−	−	※2
	0xA50B	A	−	−	※2
	0xA50D	A	−	−	※2
	0xA50F	A	−	−	※2
	0xA511	A	−	−	※2
	0xA513	A	−	−	※2
	0xA515	A	−	−	※2
	0xA517	A	−	−	※2
	0xA519	A	−	−	※2
	0xA51B	A	−	−	※2
	0xA527	A	−	−	※2
	0xA529	A	−	−	※2
	0xA52B	A	−	−	※2
	0xA52D	A	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA52F	A	−	−	※2
	0xA531	A	−	−	※2
	0xA533	A	−	−	※2

（凡例）A：標準　B：詳細　−：該当なし

注※1　図 14-18 中の番号と対応しています。

注※2　正常に処理された場合，入り口時刻が表示されます。例外が発生した場合，入り口時刻と例外が表示されます。

(c)　トランザクション外で生成された Query をトランザクション外で利用した場合

トランザクション外で生成された Query をトランザクション外で利用した場合に取得できるトレース情報を次の表に示します。

表 14–36　トランザクション外で生成された Query をトランザクション外で利用した場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA51C	A	−	−	−
	0xA51E	A	name	−	−
	0xA520	A	−	−	−
	0xA522	A	resultClass のクラス名	−	−
	0xA524	A	resultSetMapping	−	−
1，11	0xA540	A	−	−	−
	0xA542	A	−	−	−
	0xA544	A	−	−	−
1，7	0xA546	A	flushMode の値	−	−
	0xA548	B	startPosition の値	−	−
	0xA54A	B	maxResult の値	−	−
	0xA54C	B	hintName	value のクラス名	−
	0xA54E	B	position の値	−	−
	0xA550	B	position の値	−	−
	0xA552	B	position の値	−	−
	0xA554	B	name の値	−	−
	0xA556	B	name の値	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA558	B	name の値	−	−
2	0xA5A8	A	−	−	−
3	0xA5A9	A	−	−	※2
4	0xA574	B	−	−	−
	0xA576	B	name	−	−
	0xA578	B	−	−	−
	0xA57A	B	resultClass のクラス名	−	−
	0xA57C	B	resultSetMapping	−	−
4，12	0xA586	B	−	−	−
	0xA588	B	−	−	−
	0xA58A	B	−	−	−
4，8	0xA58C	B	flushMode の値	−	−
	0xA58E	B	startPosition の値	−	−
	0xA590	B	maxResult の値	−	−
	0xA592	B	hintName	value のクラス名	−
	0xA594	B	position の値	−	−
	0xA596	B	position の値	−	−
	0xA598	B	position の値	−	−
	0xA59A	B	name の値	−	−
	0xA59C	B	name の値	−	−
	0xA59E	B	name の値	−	−
5	0xA575	B	−	−	※2
	0xA577	B	−	−	※2
	0xA579	B	−	−	※2
	0xA57B	B	−	−	※2
	0xA57D	B	−	−	※2
5，13	0xA587	B	−	−	※2
	0xA589	B	−	−	※2
	0xA58B	B	−	−	※2
5，9	0xA58D	B	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA58F	B	−	−	※2
	0xA591	B	−	−	※2
	0xA593	B	−	−	※2
	0xA595	B	−	−	※2
	0xA597	B	−	−	※2
	0xA599	B	−	−	※2
	0xA59B	B	−	−	※2
	0xA59D	B	−	−	※2
	0xA59F	B	−	−	※2
6	0xA51D	A	−	−	※2
	0xA51F	A	−	−	※2
	0xA521	A	−	−	※2
	0xA523	A	−	−	※2
	0xA525	A	−	−	※2
6，16	0xA541	A	−	−	※2
	0xA543	A	−	−	※2
	0xA545	A	−	−	※2
6，10	0xA547	A	−	−	※2
	0xA549	B	−	−	※2
	0xA54B	B	−	−	※2
	0xA54D	B	−	−	※2
	0xA54F	B	−	−	※2
	0xA551	B	−	−	※2
	0xA553	B	−	−	※2
	0xA555	B	−	−	※2
	0xA557	B	−	−	※2
	0xA559	B	−	−	※2
14	0xA5AA	A	−	−	−
15	0xA5AB	A	−	−	※2

（凡例）A：標準　B：詳細　−：該当なし

注※1　図 14-19 中の番号と対応しています。

注※2　正常に処理された場合，入り口時刻が表示されます。例外が発生した場合，入り口時刻と例外が表示されます。

(d)　拡張永続化コンテキストを利用した場合

拡張永続化コンテキストを利用した場合に取得できるトレース情報を次の表に示します。

表 14–37　拡張永続化コンテキストを利用した場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA5AC	A	−	−	−
2	0xA5AD	A	−	−	※2
3	0xA508	A	entity のクラス名	−	−
	0xA50A	A	entity のクラス名	−	−
	0xA50C	A	entity のクラス名	−	−
	0xA50E	A	entity のクラス名	lockMode の値	−
	0xA510	A	entity のクラス名	−	−
	0xA512	A	entity のクラス名	−	−
	0xA514	A	entity のクラス名	−	−
	0xA516	A	entity のクラス名	−	−
	0xA518	A	−	−	−
	0xA51A	A	−	−	−
	0xA51C	A	−	−	−
	0xA51E	A	name	−	−
	0xA520	A	−	−	−
	0xA522	A	resultClass のクラス名	−	−
	0xA524	A	resultSetMapping	−	−
	0xA526	A	flushMode の値	−	−
	0xA528	A	−	−	−
	0xA52A	A	−	−	−
	0xA52C	A	−	−	−
	0xA52E	A	−	−	−
	0xA530	A	−	−	−
	0xA532	A	−	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA540	A	−	−	−
	0xA542	A	−	−	−
	0xA544	A	−	−	−
	0xA546	A	flushMode の値	−	−
	0xA548	B	startPosition の値	−	−
	0xA54A	B	maxResult の値	−	−
	0xA54C	B	hintName	value のクラス名	−
	0xA54E	B	position の値	−	−
	0xA550	B	position の値	−	−
	0xA552	B	position の値	−	−
	0xA554	B	name の値	−	−
	0xA556	B	name の値	−	−
	0xA558	B	name の値	−	−
4	0xA560	B	entity のクラス名	−	−
	0xA562	B	entity のクラス名	−	−
	0xA564	B	entity のクラス名	−	−
	0xA566	B	entity のクラス名	lockMode の値	−
	0xA568	B	entity のクラス名	−	−
	0xA56A	B	entity のクラス名	−	−
	0xA56C	B	entity のクラス名	−	−
	0xA56E	B	entity のクラス名	−	−
	0xA570	B	−	−	−
	0xA572	B	−	−	−
	0xA574	B	−	−	−
	0xA576	B	name	−	−
	0xA578	B	−	−	−
	0xA57A	B	resultClass のクラス名	−	−
	0xA57C	B	resultSetMapping	−	−
	0xA57E	B	flushMode の値	−	−
	0xA580	B	−	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA582	B	−	−	−
	0xA584	B	−	−	−
	0xA586	B	−	−	−
	0xA588	B	−	−	−
	0xA58A	B	−	−	−
	0xA58C	B	flushMode の値	−	−
	0xA58E	B	startPosition の値	−	−
	0xA590	B	maxResult の値	−	−
	0xA592	B	hintName	value のクラス名	−
	0xA594	B	position の値	−	−
	0xA596	B	position の値	−	−
	0xA598	B	position の値	−	−
	0xA59A	B	name の値	−	−
	0xA59C	B	name の値	−	−
	0xA59E	B	name の値	−	−
5	0xA561	B	−	−	※2
	0xA563	B	−	−	※2
	0xA565	B	−	−	※2
	0xA567	B	−	−	※2
	0xA569	B	−	−	※2
	0xA56B	B	−	−	※2
	0xA56D	B	−	−	※2
	0xA56F	B	−	−	※2
	0xA571	B	−	−	※2
	0xA573	B	−	−	※2
	0xA575	B	−	−	※2
	0xA577	B	−	−	※2
	0xA579	B	−	−	※2
	0xA57B	B	−	−	※2
	0xA57D	B	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA57F	B	−	−	※2
	0xA581	B	−	−	※2
	0xA583	B	−	−	※2
	0xA585	B	−	−	※2
	0xA587	B	−	−	※2
	0xA589	B	−	−	※2
	0xA58B	B	−	−	※2
	0xA58D	B	−	−	※2
	0xA58F	B	−	−	※2
	0xA591	B	−	−	※2
	0xA593	B	−	−	※2
	0xA595	B	−	−	※2
	0xA597	B	−	−	※2
	0xA599	B	−	−	※2
	0xA59B	B	−	−	※2
	0xA59D	B	−	−	※2
	0xA59F	B	−	−	※2
6	0xA509	A	−	−	※2
	0xA50B	A	−	−	※2
	0xA50D	A	−	−	※2
	0xA50F	A	−	−	※2
	0xA511	A	−	−	※2
	0xA513	A	−	−	※2
	0xA515	A	−	−	※2
	0xA517	A	−	−	※2
	0xA519	A	−	−	※2
	0xA51B	A	−	−	※2
	0xA51D	A	−	−	※2
	0xA51F	A	−	−	※2
	0xA521	A	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA523	A	−	−	※2
	0xA525	A	−	−	※2
	0xA527	A	−	−	※2
	0xA529	A	−	−	※2
	0xA52B	A	−	−	※2
	0xA52D	A	−	−	※2
	0xA52F	A	−	−	※2
	0xA531	A	−	−	※2
	0xA533	A	−	−	※2
	0xA541	A	−	−	※2
	0xA543	A	−	−	※2
	0xA545	A	−	−	※2
	0xA547	A	−	−	※2
	0xA549	B	−	−	※2
	0xA54B	B	−	−	※2
	0xA54D	B	−	−	※2
	0xA54F	B	−	−	※2
	0xA551	B	−	−	※2
	0xA553	B	−	−	※2
	0xA555	B	−	−	※2
	0xA557	B	−	−	※2
	0xA559	B	−	−	※2
7	0xA5AE	A	−	−	−
8	0xA5AF	A	−	−	※2

（凡例）A：標準　B：詳細　−：該当なし

注※1　図 14-20 中の番号と対応しています。

注※2　正常に処理された場合，入り口時刻が表示されます。例外が発生した場合，入り口時刻と例外が表示されます。

14.12　CJPA プロバイダのトレース取得ポイント

ここでは，CJPA プロバイダのトレース取得ポイントと，取得できるトレース情報について説明します。

14.12.1　EntityManagerFactory の取得／解放処理のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–38　EntityManagerFactory の取得／解放処理のトレース取得ポイントの詳細

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0xA304※2	3	EntityManagerFactory#close()の処理開始	B
0xA305※2	4	EntityManagerFactory#close()の処理終了	B
0xA320※3	1	PersistenceProvider#createEntityManagerFactory(String, Map)の処理開始	A
0xA321※3	2	PersistenceProvider#createEntityManagerFactory(String, Map)の処理終了	A
0xA322※3	1	PersistenceProvider#createContainerEntityManagerFactory(PersistenceUnitInfo, Map)の処理開始	A
0xA323※3	2	PersistenceProvider#createContainerEntityManagerFactory(PersistenceUnitInfo, Map)の処理終了	A

（凡例）A：標準　B：詳細

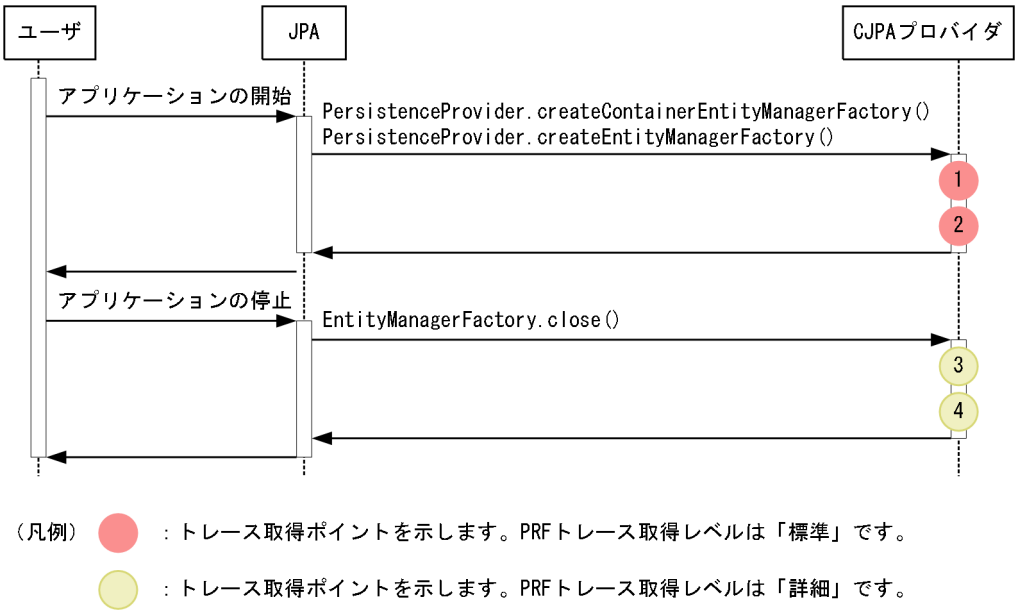
注※1　図 14-21 中の番号と対応しています。

注※2　javax.persistence.EntityManagerFactory の場合のトレース取得ポイントです。

注※3　javax.persistence.spi.PersistenceProvider の場合のトレース取得ポイントです。

トレース取得ポイントを次の図に示します。

図 14-21　EntityManagerFactory の取得／解放処理のトレース取得ポイント



(2)　取得できるトレース情報

EntityManagerFactory の取得／解放処理で取得できるトレース情報を次の表に示します。

表 14-39　EntityManagerFactory の取得／解放処理で取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1※2	0xA320※3	A	−	−	−
	0xA322	A	−	−	−
2※2	0xA321※3	A	−	−	※4
	0xA323	A	−	−	※4
3※5	0xA304	B	−	−	−
4※5	0xA305	B	−	−	※4

（凡例）A：標準　B：詳細　−：該当なし

注※1　図 14-21 中の番号と対応しています。

注※2　javax.persistence.spi.PersistenceProvider の場合のトレース取得ポイントです。

注※3　CJPA プロバイダを使用した場合は出力されません。

注※4　例外が発生した場合，例外が表示されます。

注※5　javax.persistence.EntityManagerFactory の場合のトレース取得ポイントです。

14.12.2　EntityManager の取得処理のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-40　javax.persistence.EntityManagerFactory の場合のトレース取得ポイントの詳細

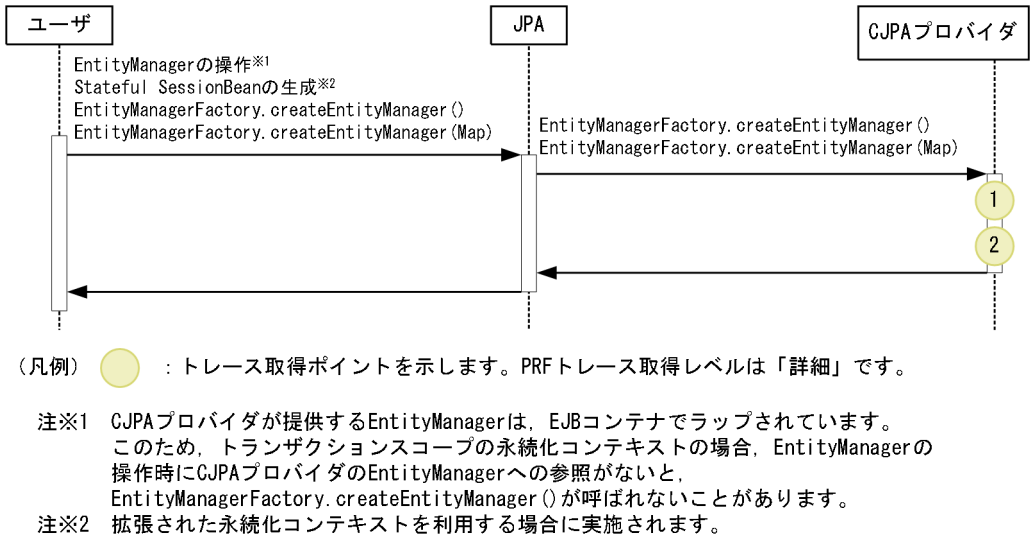
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA300	1	EntityManagerFactory#createEntityManager()の処理開始	B
0xA301	2	EntityManagerFactory#createEntityManager()の処理終了	B
0xA302	1	EntityManagerFactory#createEntityManager(Map)の処理開始	B
0xA303	2	EntityManagerFactory#createEntityManager(Map)の処理終了	B

（凡例）B：詳細

注※　図 14-22 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-22　javax.persistence.EntityManagerFactory の場合のトレース取得ポイント



(2)　取得できるトレース情報

EntityManager の取得処理で取得できるトレース情報を次の表に示します。

表 14–41　javax.persistence.EntityManagerFactory の場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA300	B	−	−	−
	0xA302	B	−	−	−
2	0xA301	B	−	−	※2
	0xA303	B	−	−	※2

（凡例）B：詳細　−：該当なし

注※1　図 14-22 中の番号と対応しています。

注※2　例外が発生した場合，例外が表示されます。

14.12.3　EntityManager の操作のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–42　javax.persistence.EntityManager の場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA340	1	EntityManager#persist(Object)の処理開始	B
0xA341	2	EntityManager#persist(Object)の処理終了	B
0xA342	1	EntityManager#merge(T)の処理開始	B
0xA343	2	EntityManager#merge(T)の処理終了	B
0xA344	1	EntityManager#remove(Object)の処理開始	B
0xA345	2	EntityManager#remove(Object)の処理終了	B
0xA346	1	EntityManager#find(Class<T>, Object)の処理開始	B
0xA347	2	EntityManager#find(Class<T>, Object)の処理終了	B
0xA348	1	EntityManager#getReference(Class<T>, Object)の処理開始	B
0xA349	2	EntityManager#getReference(Class<T>, Object)の処理終了	B
0xA34A	1	EntityManager#flush()の処理開始	B
0xA34B	2	EntityManager#flush()の処理終了	B
0xA34C	1	EntityManager#lock(Object, LockModeType)の処理開始	B

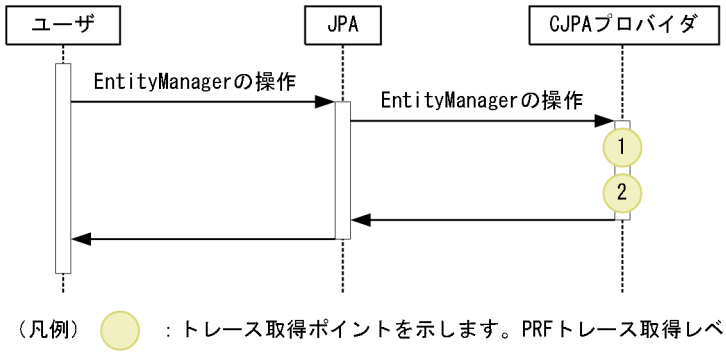
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA34D	2	EntityManager#lock(Object, LockModeType)の処理終了	B
0xA34E	1	EntityManager#refresh(Object)の処理開始	B
0xA34F	2	EntityManager#refresh(Object)の処理終了	B
0xA350	1	EntityManager#clear()の処理開始	B
0xA351	2	EntityManager#clear()の処理終了	B
0xA352	1	EntityManager#contains(Object)の処理開始	B
0xA353	2	EntityManager#contains(Object)の処理終了	B
0xA35E	1	EntityManager#joinTransaction()の処理開始	B
0xA35F	2	EntityManager#joinTransaction()の処理終了	B

（凡例）B：詳細

注※　図 14-23 の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-23　javax.persistence.EntityManager の場合のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

(2)　取得できるトレース情報

EntityManager の操作で取得できるトレース情報を次の表に示します。

表 14-43　javax.persistence.EntityManager の場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA340	B	−	−	−
	0xA342	B	−	−	−
	0xA344	B	−	−	−
	0xA346	B	−	−	−
	0xA348	B	−	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA34A	B	−	−	−
	0xA34C	B	−	−	−
	0xA34E	B	−	−	−
	0xA350	B	−	−	−
	0xA352	B	−	−	−
	0xA35E	B	−	−	−
2	0xA341	B	−	−	※2
	0xA343	B	−	−	※2
	0xA345	B	−	−	※2
	0xA347	B	−	−	※2
	0xA349	B	−	−	※2
	0xA34B	B	−	−	※2
	0xA34D	B	−	−	※2
	0xA34F	B	−	−	※2
	0xA351	B	−	−	※2
	0xA353	B	−	−	※2
	0xA35F	B	−	−	※2

（凡例）B：詳細　−：該当なし

注※1　図 14-23 中の番号と対応しています。

注※2　例外が発生した場合，例外が表示されます。

14.12.4　EntityManager の解放処理のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–44　javax.persistence.EntityManager の場合のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA360	1	EntityManager#close()の処理開始	B

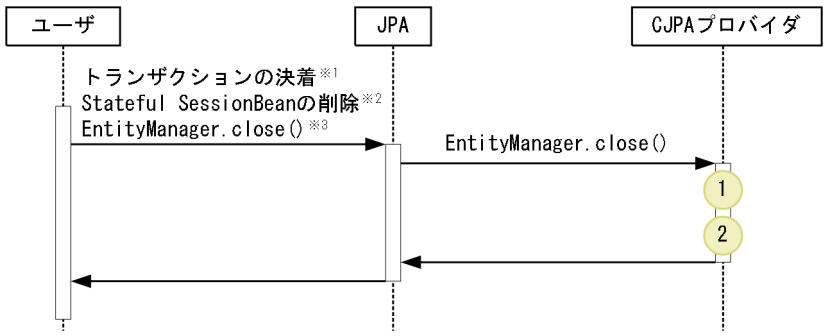
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA361	2	EntityManager#close()の処理終了	B

（凡例）B：詳細

注※　図 14-24 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-24　javax.persistence.EntityManager の場合のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

注※1　コンテナ管理のEntityManagerでトランザクションスコープの永続化コンテキストの場合

注※2　コンテナ管理のEntityManagerで拡張された永続化コンテキストの場合

注※3　アプリケーション管理のEntityManagerの場合

(2)　取得できるトレース情報

EntityManager の解放処理で取得できるトレース情報を次の表に示します。

表 14-45　javax.persistence.EntityManager の場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA360	B	－	－	－
2	0xA361	B	－	－	※2

（凡例）B：詳細　－：該当なし

注※1　図 14-24 の番号と対応しています。

注※2　例外が発生した場合，例外が表示されます。

14.12.5　Query の操作のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-46　Query 操作のトレース取得ポイントの詳細

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0xA354※2	1	EntityManager#createQuery(String)の処理開始	B
0xA355※2	2	EntityManager#createQuery(String)の処理終了	B
0xA356※2	1	EntityManager#createNamedQuery(String)の処理開始	B
0xA357※2	2	EntityManager#createNamedQuery(String)の処理終了	B
0xA358※2	1	EntityManager#createNativeQuery(String)の処理開始	B
0xA359※2	2	EntityManager#createNativeQuery(String)の処理終了	B
0xA35A※2	1	EntityManager#createNativeQuery(String, Class)の処理開始	B
0xA35B※2	2	EntityManager#createNativeQuery(String, Class)の処理終了	B
0xA35C※2	1	EntityManager#createNativeQuery(String, String)の処理開始	B
0xA35D※2	2	EntityManager#createNativeQuery(String, String)の処理終了	B
0xA370※3	5	Query#getResultList()の処理開始	B
0xA371※3	6	Query#getResultList()の処理終了	B
0xA372※3	5	Query#getSingleResult()の処理開始	B
0xA373※3	6	Query#getSingleResult()の処理終了	B
0xA374※3	5	Query#executeUpdate()の処理開始	B
0xA375※3	6	Query#executeUpdate()の処理終了	B
0xA376※3	3	Query#setParameter(String, Object)の処理開始	B
0xA377※3	4	Query#setParameter(String, Object)の処理終了	B
0xA378※3	3	Query#setParameter(String, Date, TemporalType)の処理開始	B
0xA379※3	4	Query#setParameter(String, Date, TemporalType)の処理終了	B
0xA37A※3	3	Query#setParameter(String, Calendar, TemporalType)の処理開始	B
0xA37B※3	4	Query#setParameter(String, Calendar, TemporalType)の処理終了	B
0xA37C※3	3	Query#setParameter(int, Object)の処理開始	B
0xA37D※3	4	Query#setParameter(int, Object)の処理終了	B
0xA37E※3	3	Query#setParameter(int, Date, TemporalType)の処理開始	B
0xA37F※3	4	Query#setParameter(int, Date, TemporalType)の処理終了	B
0xA380※3	3	Query#setParameter(int, Calendar, TemporalType)の処理開始	B

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0xA381※3	4	Query#setParameter(int, Calendar, TemporalType)の処理終了	B

（凡例）B：詳細

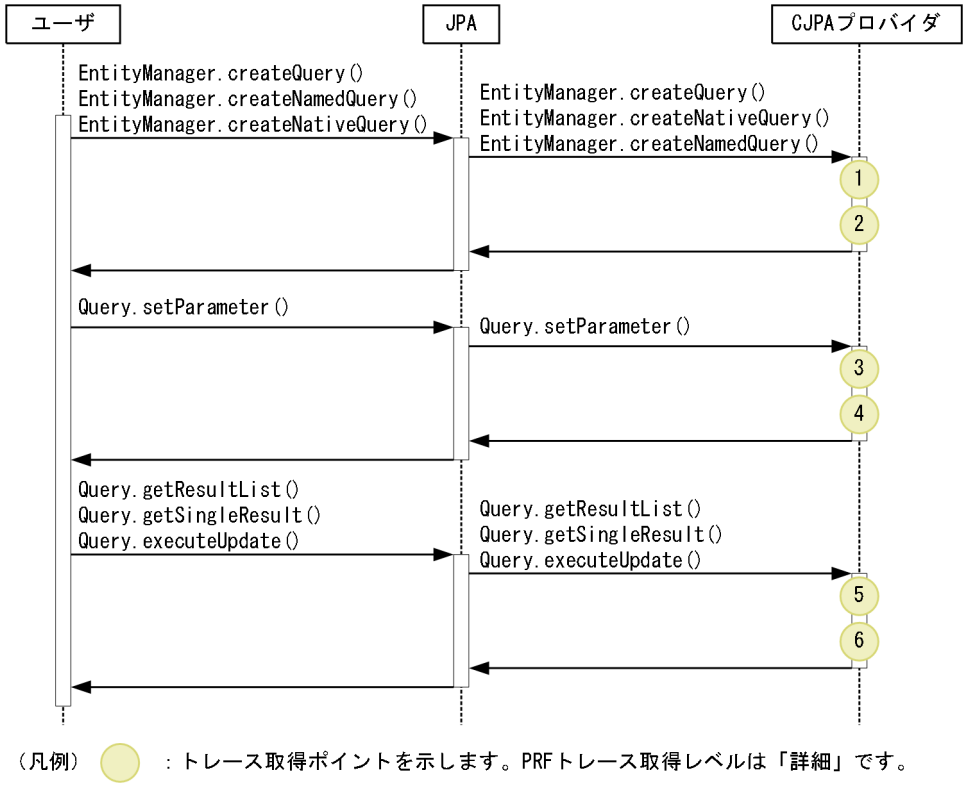
注※1　図 14-25 中の番号と対応しています。

注※2　javax.persistence.EntityManager の場合のトレース取得ポイントです。

注※3　javax.persistence.Query の場合のトレース取得ポイントです。

トレース取得ポイントを次の図に示します。

図 14-25　Query 操作のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRF トレース取得レベルは「詳細」です。

(2)　取得できるトレース情報

Query の操作で取得できるトレース情報を次の表に示します。

表 14-47　Query 操作で取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1※2	0xA354	B	−	−	−
	0xA356	B	−	−	−
	0xA358	B	−	−	−
	0xA35A	B	−	−	−

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA35C	B	−	−	−
2※2	0xA355	B	−	−	※3
	0xA357	B	−	−	※3
	0xA359	B	−	−	※3
	0xA35B	B	−	−	※3
	0xA35D	B	−	−	※3
3※4	0xA376	B	−	−	−
	0xA378	B	−	−	−
	0xA37A	B	−	−	−
	0xA37C	B	−	−	−
	0xA37E	B	−	−	−
	0xA380	B	−	−	−
4※4	0xA377	B	−	−	※3
	0xA379	B	−	−	※3
	0xA37B	B	−	−	※3
	0xA37D	B	−	−	※3
	0xA37F	B	−	−	※3
	0xA381	B	−	−	※3
5※4	0xA370	B	−	−	−
	0xA372	B	−	−	−
	0xA374	B	−	−	−
6※4	0xA371	B	−	−	※3
	0xA373	B	−	−	※3
	0xA375	B	−	−	※3

（凡例）B：詳細　−：該当なし

注※1　図 14-25 中の番号と対応しています。

注※2　javax.persistence.EntityManager の場合のトレース取得ポイントです。

注※3　例外が発生した場合，例外が表示されます。

注※4　javax.persistence.Query の場合のトレース取得ポイントです。

14.12.6　EntityTransaction の操作のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–48　EntityTransaction の操作のトレース取得ポイントの詳細

イベント ID	図中の番号※1	トレース取得ポイント	レベル
0xA310※2	3	EntityTransaction#begin()の処理開始	B
0xA311※2	4	EntityTransaction#begin()の処理終了	B
0xA312※2	5	EntityTransaction#commit()の処理開始	B
0xA313※2	6	EntityTransaction#commit()の処理終了	B
0xA314※2	5	EntityTransaction#rollback()の処理開始	B
0xA315※2	6	EntityTransaction#rollback()の処理終了	B
0xA362※3	1	EntityManager#getTransaction()の処理開始	B
0xA363※3	2	EntityManager#getTransaction()の処理終了	B

（凡例）B：詳細

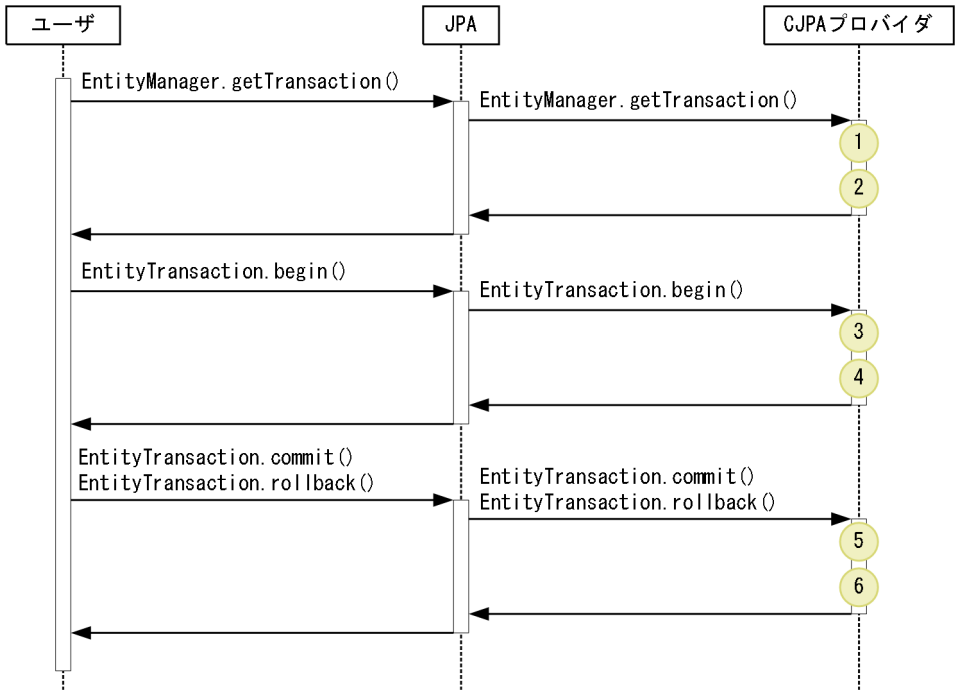
注※1　図 14-26 中の番号と対応しています。

注※2　javax.persistence.EntityTransaction の場合のトレース取得ポイントです。

注※3　javax.persistence.EntityManager の場合のトレース取得ポイントです。

トレース取得ポイントを次の図に示します。

図 14-26　EntityTransaction の操作のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

(2)　取得できるトレース情報

EntityTransaction の操作で取得できるトレース情報を次の表に示します。

表 14-49　EntityTransaction の場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1※2	0xA362	B	－	－	－
2※2	0xA363	B	－	－	※3
3※4	0xA310	B	－	－	－
4※4	0xA311	B	－	－	※3
5※4	0xA312	B	－	－	－
	0xA314	B	－	－	－
6※4	0xA313	B	－	－	※3
	0xA315	B	－	－	※3

（凡例）B：詳細　－：該当なし

注※1　図 14-26 中の番号と対応しています。

注※2　javax.persistence.EntityManager の場合のトレース取得ポイントです。

注※3　例外が発生した場合，例外が表示されます。

注※4　javax.persistence.EntityTransaction の場合のトレース取得ポイントです。

14.12.7　ユーザへのコールバックメソッドのトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-50　ユーザへのコールバックメソッドでのトレース取得ポイントの詳細

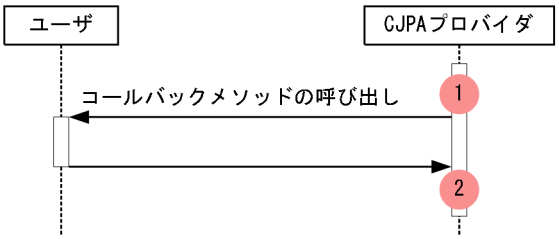
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA390	1	PrePersist()コールバックメソッドの呼び出し直前	A
0xA391	2	PrePersist()コールバックメソッドからリターンした直後	A
0xA392	1	PostPersist()コールバックメソッドの呼び出し直前	A
0xA393	2	PostPersist()コールバックメソッドからリターンした直後	A
0xA394	1	PreRemove()コールバックメソッドの呼び出し直前	A
0xA395	2	PreRemove()コールバックメソッドからリターンした直後	A
0xA396	1	PostRemove()コールバックメソッドの呼び出し直前	A
0xA397	2	PostRemove()コールバックメソッドからリターンした直後	A
0xA398	1	PreUpdate()コールバックメソッドの呼び出し直前	A
0xA399	2	PreUpdate()コールバックメソッドからリターンした直後	A
0xA39A	1	PostUpdate()コールバックメソッドの呼び出し直前	A
0xA39B	2	PostUpdate()コールバックメソッドからリターンした直後	A
0xA39C	1	PostLoad()コールバックメソッドの呼び出し直前	A
0xA39D	2	PostLoad()コールバックメソッドからリターンした直後	A

（凡例）A：標準

注※　図 14-27 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-27　ユーザへのコールバックメソッドのトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

(2)　取得できるトレース情報

ユーザへのコールバックメソッドで取得できるトレース情報を次の表に示します。

表 14-51　ユーザへのコールバックメソッドで取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA390	A	ユーザが指定したコールバックメソッド名	－	－
	0xA392	A		－	－
	0xA394	A		－	－
	0xA396	A		－	－
	0xA398	A		－	－
	0xA39A	A		－	－
	0xA39C	A		－	－
2	0xA391	A	ユーザが指定したコールバックメソッド名	－	※2
	0xA393	A		－	※2
	0xA395	A		－	※2
	0xA397	A		－	※2
	0xA399	A		－	※2
	0xA39B	A		－	※2
	0xA39D	A		－	※2

（凡例）A：標準　－：該当なし

注※1　図 14-27 中の番号と対応しています。

注※2　例外が発生した場合，例外が表示されます。

14.12.8　エンティティクラスのバイナリ変換のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-52　エンティティクラスのバイナリ変換でのトレース取得ポイントの詳細

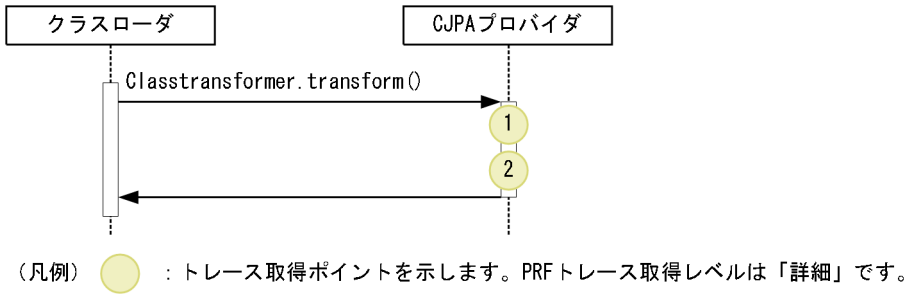
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA330	1	エンティティクラスに対するバイナリ変換処理の開始直前	B
0xA331	2	エンティティクラスに対するバイナリ変換処理の完了直後	B

（凡例）B：詳細

注※　図 14-28 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-28　エンティティクラスのバイナリ変換でのトレース取得ポイント



ポイント

JPA を使用するアプリケーションでは，エンティティクラス以外のクラスの場合でも，そのクラスがクラスローダからローディングされるタイミングでこれらのトレース情報が取得されます。

(2)　取得できるトレース情報

エンティティクラスのバイナリ変換で取得できるトレース情報を次の表に示します。

表 14-53　エンティティクラスのバイナリ変換で取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA330	B	−	−	−
2	0xA331	B	−	−	※2

（凡例）B：詳細　−：該当なし

注※1　図 14-28 中の番号と対応しています。
注※2　例外が発生した場合，例外が表示されます。

14.12.9　トランザクションマネージャとのトランザクション連携処理のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

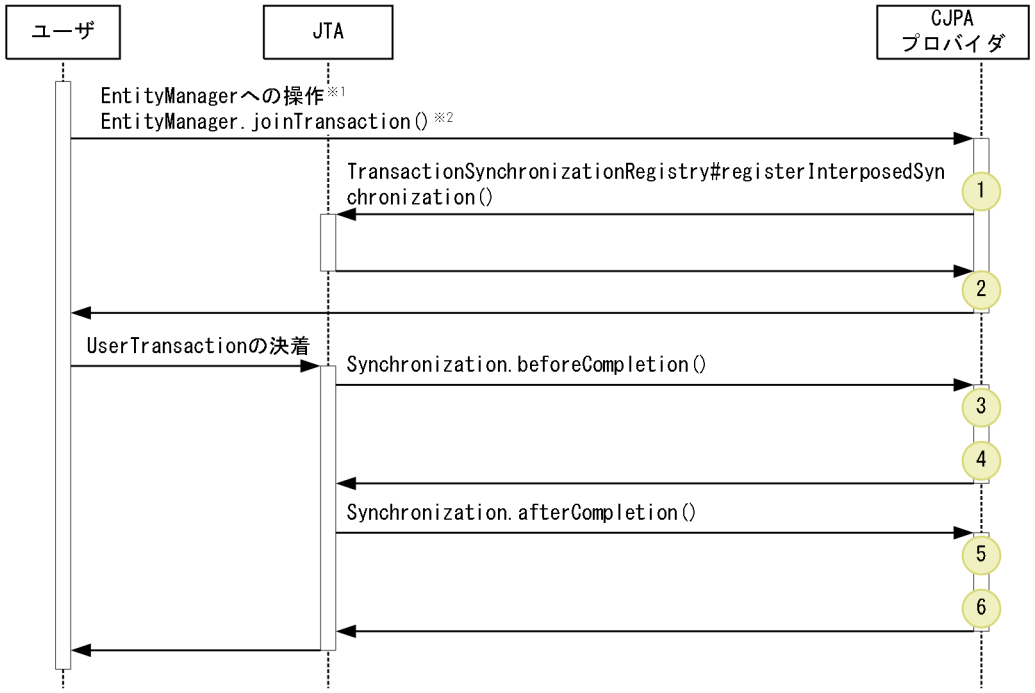
表 14–54　トランザクションマネージャとのトランザクション連携処理のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA39E	1	永続化コンテキストとトランザクションを関連づける処理の開始直前	B
0xA39F	2	永続化コンテキストとトランザクションを関連づける処理の完了直後	B
0xA3A0	3	JTA トランザクション決着の前処理の開始直前	B
0xA3A1	4	JTA トランザクション決着の前処理の終了直後	B
0xA3A2	5	JTA トランザクション決着の後処理の開始直前	B
0xA3A3	6	JTA トランザクション決着の後処理の終了直後	B

（凡例）B：詳細
注※　図 14-29 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-29　トランザクションマネージャとのトランザクション連携処理のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

注※1　UserTransactionの開始後にEntityManagerへの操作を行った場合です。

注※2　コンテナが内部的に，EntityManager.joinTransaction()を呼び出すこともあります。

(2)　取得できるトレース情報

トランザクションマネージャとのトランザクション連携処理で取得できるトレース情報を次の表に示します。

表 14-55　トランザクションマネージャとのトランザクション連携処理で取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA39E	B	－	－	－
2	0xA39F	B	－	－	※2
3	0xA3A0	B	－	－	－
4	0xA3A1	B	－	－	※2
5	0xA3A2	B	－	afterCompletion()の引数で渡されたトランザクションのステータス(javax.transaction.Status の int 値)	－
6	0xA3A3	B	－	－	※2

（凡例）B：詳細　－：該当なし

注※1　図 14-29 中の番号と対応しています。

注※2　例外が発生した場合，例外が表示されます。

14.12.10　DB Connector のコネクション操作のトレース取得ポイントと取得できるトレース情報

(1)　トレース取得ポイントと PRF トレース取得レベル

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14–56　DB Connector のコネクション操作のトレース取得ポイントの詳細

イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA3A4	1	DataSource#getConnection()の呼び出し直前	B
0xA3A5	2	DataSource#getConnection()からリターンした直後	B
0xA3A6	1	DataSource#getConnection(String, String)の呼び出し直前	B
0xA3A7	2	DataSource#getConnection(String, String)からリターンした直後	B
0xA3A8	1	DriverManager#getConnection(String, Properties)の呼び出し直前	B
0xA3A9	2	DriverManager#getConnection(String, Properties)からリターンした直後	B
0xA3AA	1	DriverManager#getConnection(String, String, String)の呼び出し直前	B
0xA3AB	2	DriverManager#getConnection(String, String, String)からリターンした直後	B
0xA3AC	9	Connection#close()の呼び出し直前	B
0xA3AD	10	Connection#close()からリターンした直後	B
0xA3AE	7	Connection#commit()の呼び出し直前	B
0xA3AF	8	Connection#commit()からリターンした直後	B
0xA3B0	3	Connection#createStatement()の呼び出し直前	B
0xA3B1	4	Connection#createStatement()からリターンした直後	B
0xA3B2	3	Connection#prepareCall(String)の呼び出し直前	B
0xA3B3	4	Connection#prepareCall(String)からリターンした直後	B
0xA3B4	3	Connection#prepareCall(String, int, int)の呼び出し直前	B
0xA3B5	4	Connection#prepareCall(String, int, int)からリターンした直後	B
0xA3B6	3	Connection#prepareStatement(String)の呼び出し直前	B
0xA3B7	4	Connection#prepareStatement(String)からリターンした直後	B
0xA3B8	3	Connection#prepareStatement(String, int, int)の呼び出し直前	B

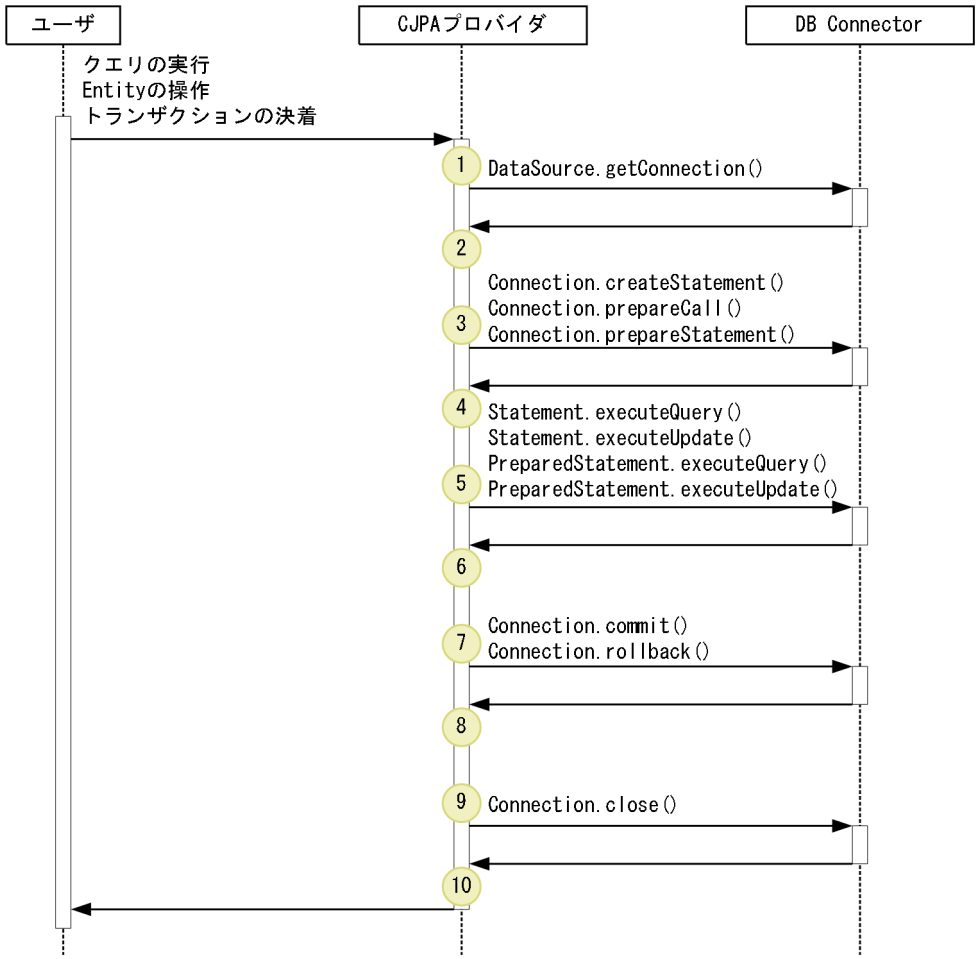
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xA3B9	4	Connection#prepareStatement(String , int, int)からリターンした直後	B
0xA3BA	7	Connection#rollback()の呼び出し直前	B
0xA3BB	8	Connection#rollback()からリターンした直後	B
0xA3BC	5	Statement#executeQuery(String)の呼び出し直前	B
0xA3BD	6	Statement#executeQuery(String)からリターンした直後	B
0xA3BE	5	Statement#executeUpdate(String)の呼び出し直前	B
0xA3BF	6	Statement#executeUpdate(String)からリターンした直後	B
0xA3C0	5	PreparedStatement#executeUpdate()の呼び出し直前	B
0xA3C1	6	PreparedStatement#executeUpdate()からリターンした直後	B
0xA3C2	5	PreparedStatement#executeQuery()の呼び出し直前	B
0xA3C3	6	PreparedStatement#executeQuery()からリターンした直後	B

（凡例）B：詳細

注※　図 14-30 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-30　DB Connector のコネクション操作のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

(2)　取得できるトレース情報

DB Connector のコネクション操作で取得できるトレース情報を次の表に示します。

表 14-57　DB Connector のコネクション操作で取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xA3A4	B	−	−	−
	0xA3A6	B	−	−	−
	0xA3A8	B	−	−	−
	0xA3AA	B	−	−	−
2	0xA3A5	B	−	−	※2
	0xA3A7	B	−	−	※2
	0xA3A9	B	−	−	※2

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
	0xA3AB	B	−	−	※2
3	0xA3B0	B	−	−	−
	0xA3B2	B	−	−	−
	0xA3B4	B	−	−	−
	0xA3B6	B	−	−	−
	0xA3B8	B	−	−	−
4	0xA3B1	B	−	−	※2
	0xA3B3	B	−	−	※2
	0xA3B5	B	−	−	※2
	0xA3B7	B	−	−	※2
	0xA3B9	B	−	−	※2
5	0xA3BC	B	−	−	−
	0xA3BE	B	−	−	−
	0xA3C0	B	−	−	−
	0xA3C2	B	−	−	−
6	0xA3BD	B	−	−	※2
	0xA3BF	B	−	−	※2
	0xA3C1	B	−	−	※2
	0xA3C3	B	−	−	※2
7	0xA3AE	B	−	−	−
	0xA3BA	B	−	−	−
8	0xA3AF	B	−	−	※2
	0xA3BB	B	−	−	※2
9	0xA3AC	B	−	−	−
10	0xA3AD	B	−	−	※2

（凡例）B：詳細　−：該当なし

注※1　図 14-30 中の番号と対応しています。

注※2　例外が発生した場合，例外が表示されます。

14.13　CDI のトレース取得ポイント

ここでは，CDI のトレース取得ポイントと，取得できるトレース情報について説明します。

14.13.1　CDI のトレース取得ポイントと取得できるトレース情報

CDI のトレース取得ポイントと取得できるトレース情報について説明します。ここでは，次の二つの場合に分けて説明します。

- JSF と CDI を組み合わせて利用する場合
- サーブレットと CDI を組み合わせて利用する場合

(1)　トレース取得ポイントと PRF トレース取得レベル

(a)　JSF と CDI を組み合わせて利用する場合

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-58　JSF と CDI を組み合わせて利用する場合のトレース取得ポイントの詳細

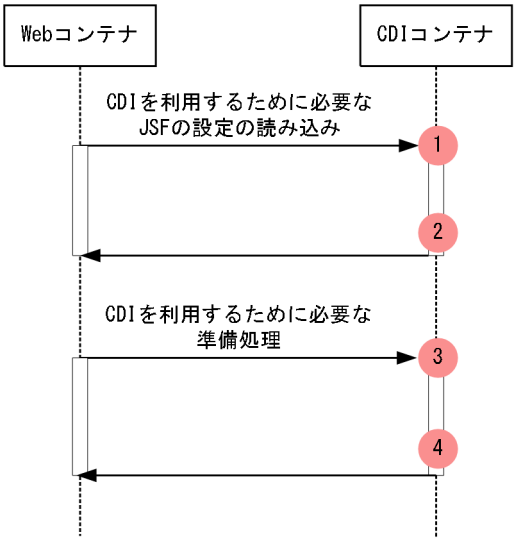
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xb002	1	CDI を利用するために必要な JSF の設定の読み込み処理開始	A
0xb003	2	CDI を利用するために必要な JSF の設定の読み込み処理終了（正常終了）	A
0xb004	3	CDI を利用するために必要な JSF の準備の処理開始	A
0xb005	4	CDI を利用するために必要な JSF の準備の処理終了（正常終了）	A
0xb006	5	EL 評価準備の処理開始	B
0xb007	6	EL 評価準備の処理終了（正常終了）	B

（凡例）A：標準　B：詳細

注※　図 14-31 および図 14-32 中の番号と対応しています。

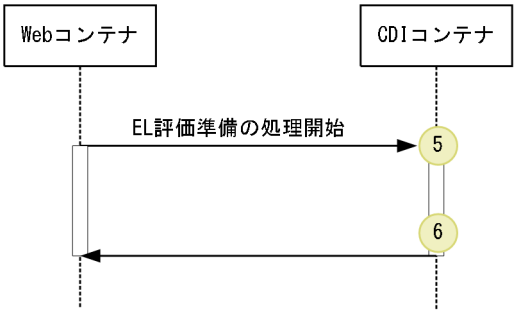
トレース取得ポイントを次の図に示します。

図 14-31　JSF と CDI を組み合わせて利用する場合のトレース取得ポイント（JSF の設定の読み込みと準備処理時）



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

図 14-32　JSF と CDI を組み合わせて利用する場合のトレース取得ポイント（EL 評価処理時）



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「詳細」です。

(b)　サーブレット・フィルタ・リスナと CDI を組み合わせて利用する場合

イベント ID，トレース取得ポイント，および PRF トレース取得レベルについて，次の表に示します。

表 14-59　サーブレット・フィルタ・リスナと CDI を組み合わせて利用する場合のトレース取得ポイントの詳細

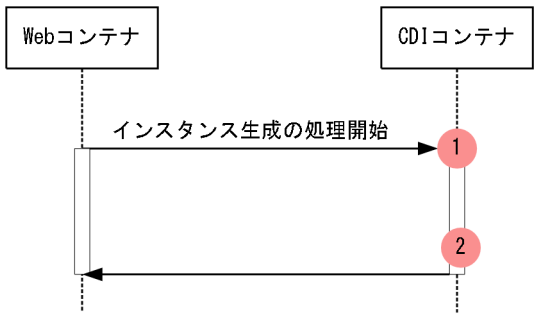
イベント ID	図中の番号※	トレース取得ポイント	レベル
0xb008	1	サーブレット・フィルタ・リスナのインスタンス生成の処理開始	A
0xb009	2	サーブレット・フィルタ・リスナのインスタンス生成の処理終了（正常終了）	A

（凡例）A：標準

注※　図 14-33 中の番号と対応しています。

トレース取得ポイントを次の図に示します。

図 14-33　サーブレット・フィルタ・リスナと CDI を組み合わせて利用する場合のトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。PRFトレース取得レベルは「標準」です。

(2)　取得できるトレース情報

(a)　JSF と CDI を組み合わせて利用する場合

JSF と CDI を組み合わせて利用する場合に取得できるトレース情報を次の表に示します。

表 14-60　JSF と CDI を組み合わせて利用する場合に取得できる場合に取得できるトレース情報

図中の番号※1	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xb002※2	A	WeldFacesConfigProvider	−	Web コンテナのコンテキスト情報
2	0xb003※2	A	WeldFacesConfigProvider	−	入口時刻
3	0xb004※2	A	WeldApplicationFactory	−	−
4	0xb005※2	A	WeldApplicationFactory	−	入口時刻
5	0xb006※3	B	WeldApplication	−	−
6	0xb007※3	B	WeldApplication	−	入口時刻

（凡例）A：標準　B：詳細　−：該当なし

注※1　図 14-31 および図 14-32 中の番号と対応しています。

注※2　CDI を利用するために必要な JSF の設定の読み込みと，CDI を利用するために必要な JSF の準備のトレース情報はアプリケーション開始時に取得します。

注※3　EL 評価準備のトレース情報は，FacesServlet が初期化処理時と JSF で指定されている EL 式の評価実行時に取得します。

(b)　CDI からサーブレットを呼び出す場合

CDI からサーブレットを呼び出す場合に取得できるトレース情報を次の表に示します。

表 14-61　CDI からサーブレットを呼び出す場合に取得できるトレース情報

図中の番号※	イベント ID	レベル	取得できる情報		
			インタフェース名	オペレーション名	オプション
1	0xb008	A	CDIServiceImpl	－	次の情報が出力されます。 • Managed クラス • クラス名 • コンテキストルート
2	0xb009	A	CDIServiceImpl	－	入口時刻

（凡例）A：標準　－：該当なし

注※　図 14-33 中の番号と対応しています。

なお，トレース情報は，サーブレット・フィルタ・リスナのインタフェース生成時に取得します。

15 出力されるログ情報とログ取得の設定

この章では，出力されるログ情報とログ取得の設定について説明します。

15.1　機能ごとに出力されるログ情報

特定の機能を利用したときに出力されるログ情報について説明します。

ここでは，次の機能を利用したときに出力されるログ情報について説明します。

- CJPA プロバイダの稼働ログ

15.1.1　CJPA プロバイダの稼働ログ

ここでは，CJPA プロバイダが出力する稼働ログのメッセージについて，ログ出力のカテゴリごとに説明します。

(1)　カテゴリが SQL の場合

```
SQL [aa....aa] SQL = bb....bb
```

aa....aa：永続化ユニット名

bb....bb：発行する SQL 文

説明

JPA が SQL を発行します。

```
SQL [aa....aa] PARAM = bb....bb
```

aa....aa：永続化ユニット名

bb....bb：?パラメタ(プレースホルダ)の設定値

説明

SQL 文の?パラメタ（プレースホルダ）に値を設定します。

bb....bb には，?パラメタ（プレースホルダ）に設定された値を文字表現に変換した結果が出力されます。文字や数値の場合はそのまま出力されます。ただし，例えばバイナリデータが java の byte 型配列に格納されていると，byte 型配列に対して toString メソッドを実行して，「[B@f7ba93」のような表記で出力されます。

```
SQL [aa....aa] RETURN = bb....bb
```

aa....aa：永続化ユニット名

bb....bb：戻り値

説明

PreparedStatement#executeUpdate()の戻り値です。戻り値はSQL文を実行した結果の行数です。詳細についてはJavadocの「java.sql.PreparedStatement」を参照してください。

(2) カテゴリがTRANSACTIONの場合

```
TRN [aa....aa] JPA processing was bound to a JTA transaction. (status = bb....bb)
```

aa....aa：永続化ユニット名

bb....bb：JTAトランザクションに参加したときのトランザクションの状態

説明

JTAトランザクションに参加しました。トランザクションの状態（ステータス）を示す文字列を出力します。状態を示す文字列の詳細については，Javadocの「javax.transaction.Status」を参照してください。

```
TRN [aa....aa] A JTA transaction was committed. (status = bb....bb)
```

aa....aa：永続化ユニット名

bb....bb：JTAトランザクションが決着したときのトランザクションの状態

説明

JTAトランザクションの決着が通知されました。JTAから渡される，トランザクションが決着したときの状態（ステータス）を示す文字列を出力します。状態を示す文字列の詳細については，Javadocの「javax.transaction.Status」を参照してください。

```
TRN [aa....aa] An EntityTransaction started.
```

aa....aa：永続化ユニット名

説明

EntityTransactionを開始しました。

TRN [aa....aa] An EntityTransaction was committed.

aa....aa：永続化ユニット名

説明

EntityTransaction をコミットしました。

TRN [aa....aa] An EntityTransaction was rolled back.

aa....aa：永続化ユニット名

説明

EntityTransaction をロールバックしました。

15.2　インプロセス HTTP サーバのログ取得の設定

ここでは，インプロセス HTTP サーバのログ取得で設定できる項目について説明します。

インプロセス HTTP サーバでは，アプリケーション開発のサポート，運用時の性能解析，および障害発生時のトラブルシューティングのために，アクセスログ，性能解析トレース，スレッドトレース，および通信トレースを出力します。これらのファイルでは，ファイル面数やファイルサイズを簡易構築定義ファイルで変更できます。

インプロセス HTTP サーバのログ取得で変更できる設定項目と，項目に対応する簡易構築定義ファイルのパラメタを次の表に示します。

表 15–1　インプロセス HTTP サーバのログ取得の設定項目

ログおよびトレース	項目	対応する簡易構築定義ファイルのパラメタ
アクセスログ	アクセスログの出力の有無	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.enabled （デフォルトではアクセスログが出力されます）
	アクセスログのファイル名	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.filename
	アクセスログのファイルサイズ	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.filesize
	アクセスログのファイル面数	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.filenum
	アクセスログのフォーマット名	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.format_list※
	アクセスログの出力形式	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.<フォーマット名>※
	アクセスログ出力時のフォーマット	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.access_log.inprocess_http.usage_format※
性能解析トレース	－	ほかの性能解析トレースと同様に，日常的なシステム運用の作業で，cprfed コマンドを実行するときに取得条件などを指定します。性能解析トレースファイルの取得については，「14. 性能解析トレース」を参照してください。
スレッドトレース	スレッドトレースのファイル面数	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.thread_trace.inprocess_http.filenum
	スレッドトレースのファイルサイズ	スレッドトレースのファイルサイズ＝（((A+B）×32,786）＋ 32,914）バイト A ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の

ログおよびトレース	項目	対応する簡易構築定義ファイルのパラメタ
		webserver.connector.inprocess_http.max_connections パラメタの値 B ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.send_timeout パラメタの値が 0 の場合 0，0 以外の場合 1
通信トレース	通信トレースのファイル面数	論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.logger.communication_trace.inprocess_http.filenum
	通信トレースのファイルサイズ	通信トレースのファイルサイズ＝（((A+B）×172,050）＋128）バイト A ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.max_connections パラメタの値 B ＝論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.send_timeout パラメタの値が 0 の場合は 0，0 以外の場合は 1

（凡例）－：該当しない

注※　アクセスログでは，これらのキーでフォーマットを定義することで，ログの出力形式をカスタマイズできます。インプロセス HTTP サーバのアクセスログのカスタマイズについては，「6.17.2　インプロセス HTTP サーバのアクセスログのカスタマイズ」を参照してください。

16 システム設計ガイド（V9 互換モード）

この章では，J2EE アプリケーションを実行するシステムのパフォーマンスをチューニングする方法について説明します。

パフォーマンスチューニングによって動作環境を最適化することで，システムの性能を最大限に生かせるようになります。

バッチアプリケーション実行基盤のパフォーマンスチューニングについて検討する場合は，マニュアル「アプリケーションサーバ システム設計ガイド」の「9. パフォーマンスチューニング（バッチアプリケーション実行基盤）」を参照してください。

16.1　パフォーマンスチューニングで考慮すること

この節では，J2EE アプリケーション実行基盤のパフォーマンスチューニングで考慮することについて説明します。

16.1.1　パフォーマンスチューニングの観点

J2EE アプリケーション実行基盤のパフォーマンスチューニングは，次の観点で実施します。

- 同時実行数の最適化
- Enterprise Bean の呼び出し方法の最適化
- データベースアクセス方法の最適化
- タイムアウトの設定
- Web アプリケーションの動作の最適化
- CTM の動作の最適化
- そのほかの項目のチューニング

それぞれのポイントについて説明します。

(1)　同時実行数の最適化

同時実行数の最適化は，処理を多重化して CPU の処理能力を最大限に引き出して，システムのスループットを向上させることを目的とします。しかし，次のような場合，多重化しただけではスループットが向上しません。場合によっては，スループットが低下するおそれがあります。

- 入出力処理，排他処理などのボトルネックがある場合
- 最大スループットに到達している場合
- CPU の利用率が飽和した状態で多重度以上の負荷を掛けた場合
- 実行待ちキューのサイズが不適切な場合
- 階層的な最大実行数の設定が不適切な場合

パフォーマンスチューニングでは，これらを考慮しながら適切なチューニングを実施して，同時実行数の最適化を図ります。

(2)　Enterprise Bean の呼び出し方法の最適化

Enterprise Bean の呼び出し方法の最適化は，同じ J2EE アプリケーションや同じ J2EE サーバ内のコンポーネントを呼び出すときに，ローカルインタフェースやリモートインタフェースのローカル呼び出し機能を利用することで，不要なネットワークアクセスを削減することを目的とします。

次の機能を利用することで，RMI-IIOP 通信によって発生する不要なネットワークアクセスを削減できます。

- ローカルインタフェースの利用
- リモートインタフェースのローカル呼び出し機能の利用

また，引数や戻り値の渡し方を参照渡しにすることで，さらに処理性能を向上できる場合があります。パフォーマンスチューニングでは，アプリケーションやシステムの特徴によってこれらの機能を有効に活用して，処理性能の向上を図ります。

(3) データベースアクセス方法の最適化

データベースアクセス方法の最適化は，処理に時間が掛かるコネクションやステートメントを事前に生成しておくことで，データベースアクセス時のオーバーヘッドを削減することを目的とします。

パフォーマンスチューニングでは，次に示す機能を有効に活用することで，データベースアクセス処理を最適化し，スループットを向上させます。

- コネクションプーリング
- ステートメントプーリング（PreparedStatement および CallableStatement のプーリング）

(4) タイムアウトの設定

タイムアウトの設定は，システムのトラブル発生を検知して，リクエストの応答が返らなくなることを防ぎ，適宜リソースを解放することを目的とします。

設定できるタイムアウトには，次の種類があります。

- Web フロントシステムのタイムアウト
- バックシステムのタイムアウト
- トランザクションのタイムアウト
- データベースのタイムアウト

(5) Web アプリケーションの動作の最適化

Web アプリケーションの動作の最適化は，コンテンツの配置方法の検討やキャッシュの利用によって不要なネットワークアクセスを削減して処理速度を速めたり，負荷分散によってシステムのスループットの向上を図ったりすることを目的とします。

なお，Web サーバとして，リダイレクタモジュールを組み込んだ Web サーバと連携する場合と，インプロセス HTTP サーバを使用している場合で，チューニングできる項目が異なります。

Web サーバと連携する場合には，次の処理ができます。

- 静的コンテンツと Web アプリケーションでの処理の振り分け

- 静的コンテンツのキャッシュ
- セッション情報に応じたリクエストの振り分け

インプロセス HTTP サーバを使用する場合は，次の処理ができます。

- 静的コンテンツと Web アプリケーションの配置の切り分け
- 静的コンテンツのキャッシュ

(6) CTM の動作の最適化

CTM の動作の最適化は，CTM で使用するプロセス間の通信間隔を最適化して通信負荷を軽減したり，トラブル発生時に迅速に検知して対処したりすることで，システムとしての性能を向上させることを目的とします。また，CTM によってリクエストの処理に優先順位を付けることで，重要なリクエストをすばやく処理するようにもチューニングできます。

(7) そのほかの項目のチューニング

アプリケーションサーバでは，(1)～(6)で説明した項目以外にも，チューニングできる項目があります。必要に応じてチューニングを実施してください。

16.1.2 アプリケーションの種類ごとにチューニングできる項目

チューニング項目は，アプリケーションの種類によって異なります。アプリケーションに含まれるコンポーネントごとのチューニング項目について，次に示します。

表 16–1 サーブレットと JSP で構成されるアプリケーション（Web アプリケーション）のチューニング項目

チューニング項目	利用できる機能	参照先
リクエスト処理スレッド数の最適化（インプロセス HTTP サーバを使用する場合）	リクエスト処理スレッド数の制御（インプロセス HTTP サーバを使用する場合）※	16.3.1
同時実行数の最適化	Web アプリケーションでの同時実行スレッド数制御（Web コンテナ単位，Web アプリケーション単位，または URL グループ単位）	16.3.2
データベースアクセス方法の最適化	コネクションプーリング	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.5.1」
	ステートメントプーリング	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.5.2」

チューニング項目	利用できる機能	参照先
タイムアウトの設定	Web フロントシステムでのタイムアウトの設定	16.4.2
	J2EE アプリケーションのメソッド実行時間に対するタイムアウトの設定	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.7」
Web アプリケーションの動作の最適化	静的コンテンツと Web アプリケーションの配置の切り分け	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.7.1」
	静的コンテンツのキャッシュ	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.7.2」
	リダイレクタによるリクエストの振り分け（Web サーバ連携の場合）	5.2
そのほかの項目のチューニング	Persistent Connection の制御（インプロセス HTTP サーバを使用する場合）	16.6

注※　Web サーバと連携する場合は，Web サーバの機能を使用してチューニングしてください。

表 16–2　Enterprise Bean で構成されるアプリケーションのチューニング項目

チューニング項目	利用できる機能	参照先
同時実行数の最適化	Stateless Session Bean のインスタンスプーリング	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.3.5」
	Stateful Session Bean のセッション制御	
	Message-driven Bean のインスタンスプーリング	
	CTM による同時実行数制御※（CTM を使用している場合）	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.3.6」
Enterprise Bean の呼び出し方法の最適化	ローカルインタフェースの使用	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.4.1」
	リモートインタフェースのローカル呼び出し最適化	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.4.2」
	リモートインタフェースの参照渡し	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.4.3」

チューニング項目	利用できる機能	参照先
データベースアクセス方法の最適化	コネクションプーリング	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.5.1」
	ステートメントプーリング	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.5.2」
タイムアウトの設定	バックシステムでのタイムアウトの設定	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.3」
	トランザクションタイムアウトの設定	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.4」
	データベースでのタイムアウトの設定	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.6」
	J2EE アプリケーションのメソッド実行時間に対するタイムアウトの設定	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.7」

注※　Stateless Session Bean だけが対象です。

また，CTM を使用したシステムの場合に設定できる，CTM の動作のチューニング項目を次の表に示します。CTM は，アプリケーションが Stateless Session Bean で構成されている場合に使用できます。

表 16–3　CTM の動作についてのチューニング項目

チューニング項目	利用できる機能	参照先
CTM の動作の最適化	CTM ドメインマネジャおよび CTM デーモンの稼働状態を監視する間隔のチューニング	マニュアル「アプリケーションサーバ システム設計ガイド」の「8.8.1」
	負荷状況監視間隔のチューニング	「アプリケーションサーバ システム設計ガイド」の「8.8.2」
	CTM デーモンのタイムアウト閉塞の設定	「アプリケーションサーバ システム設計ガイド」の「8.8.3」

チューニング項目	利用できる機能	参照先
	CTM で振り分けるリクエストの優先順位の設定	「アプリケーションサーバ システム設計ガイド」の「8.8.4」

16.2　チューニングの方法

この節では，チューニングの方法について説明します。チューニングの方法は，設定対象の種類によって異なります。

16.2.1　J2EE サーバおよび Web サーバ（リダイレクタを含む）のチューニング

J2EE サーバおよび Web サーバ（リダイレクタを含む）のチューニングには，Smart Composer 機能の簡易構築定義ファイルを使用します。簡易構築定義ファイルでは，<configuration>タグ下の<logical-server-type>に設定対象とする論理サーバの種類（J2EE サーバまたは Web サーバ）を指定して，<param>タグ下でパラメタ名とその値を設定します。簡易構築定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

参考

Smart Composer 機能を使用できない場合，Web サーバ（リダイレクタを含む）のチューニングはファイルを編集して定義します。

Smart Composer 機能を使用できない場合に，Web サーバ（リダイレクタを含む）のチューニングに使用するファイルについて，次の表に示します。

表 16-4　Smart Composer 機能を使用できない場合に Web サーバ（リダイレクタを含む）のチューニングに使用するファイル

対象	チューニング方法
Web サーバ	httpsd.conf の編集
Web サーバ（リダイレクタ）	mod_jk.conf（HTTP Server の場合）の編集
	isapi_redirect.conf（Microsoft IIS の場合）の編集
	workers.properties（ワーカの設定の場合）の編集

mod_jk.conf の詳細については，「13.2.2　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。isapi_redirect.conf の詳細については，「13.2.1 isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）」を参照してください。workers.properties の詳細については，「13.2.4　workers.properties（ワーカ定義ファイル）」を参照してください。httpsd.conf の詳細については，マニュアル「HTTP Server」を参照してください。

16.3　同時実行数を最適化する

この節では，アプリケーションのリクエストの同時実行数を最適化するための考え方とチューニング方法について説明します。

16.3.1　Web サーバでのリクエスト処理スレッド数を制御する

Web フロントシステムの場合，Web ブラウザなどのクライアントからのリクエストは，Web サーバが作成するリクエスト処理スレッドによって処理されます。リクエスト処理スレッド数を適切に制御することで，処理性能の向上が図れます。

ここでは，Web サーバでのリクエスト処理スレッド数を制御する目的と，チューニングの指針について説明します。

なお，ここでは，インプロセス HTTP サーバを使用している場合のチューニングの方法について説明します。

参考

Web サーバ連携時に HTTP Server を使用している場合は，HTTP Server の設定で同様のチューニングができます。詳細は，マニュアル「HTTP Server」を参照してください。

また，Smart Composer 機能を使用してシステムを構築する場合，Web サーバでのリクエスト処理スレッド数の設定は，抽象パラメタの利用によって設定できます。抽象パラメタとは，互いに関連のあるパラメタを一つにまとめたパラメタです。抽象パラメタを利用することで，Web サーバのリクエスト処理スレッド数の設定を，同時実行スレッド数などの関連するパラメタと一緒に定義できます。抽象パラメタについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「付録 I　Smart Composer 機能で使用できる抽象パラメタ（V9 互換モードの場合）」を参照してください。

(1)　リクエスト処理スレッド数を制御する目的

リクエスト処理スレッド数を J2EE サーバが動作しているホストの性能やクライアントからのアクセス状況に合わせてチューニングすることで，性能向上を図れます。

リクエスト処理スレッドの生成は，負荷が高い処理です。リクエスト処理スレッドをあらかじめ生成してプールしておくことで，Web ブラウザなどのクライアントからのリクエスト処理要求時の負荷を軽くして，処理性能を高めることができます。

インプロセス HTTP サーバを使用する場合，J2EE サーバ起動時にリクエスト処理スレッドをまとめて生成してプールしておき，Web ブラウザなどのクライアントからリクエスト処理要求があった場合にそれを利用するようにできます。これによって，リクエスト処理要求時の処理性能の向上を図れます。なお，プー

ルしているスレッドの数を監視しておくことで，プールしているスレッド数が少なくなった場合はさらに追加生成して，プールに確保しておくこともできます。

ただし，使用しないスレッドを大量にプールしておくと，むだなリソースを消費します。このため，システムの処理内容に応じて，プールするリクエスト処理スレッド数を適切に制御し，場合によっては不要なスレッドを削除することが必要です。

リクエスト処理スレッドの制御では，これらを考慮して，パラメタに適切な値を設定してください。

(2) 設定の指針

リクエスト処理スレッド数の制御では，次のパラメタを使用してチューニングできます。

- J2EE サーバ起動時に生成するリクエスト処理スレッドの数
- Web クライアントとの接続数（リクエスト処理スレッド数）の上限数
- 接続数の上限を超えた場合の TCP/IP の Listen キュー（バックログ）の最大値
- 予備スレッド数の最大数および最小数
- J2EE サーバ起動時に生成したリクエスト処理スレッド数を維持するかどうかの選択

これらのパラメタを設定するときには，次の点に留意してください。

- 提供するサービスの内容によっては，J2EE サーバ起動直後から大量のリクエストを処理する必要があります。この場合は，J2EE サーバ起動時に生成するリクエスト処理スレッドの数に，大きな値を指定してください。
- 予備スレッド数の最大数を大きくしておくと，クライアントからのアクセスが急に増加した場合にも，処理性能を下げることなく迅速に対応できます。ただし，多くの予備スレッドをプールしておくと，多くのリソースが消費されます。このため，急な増加が予想されるアクセス数を見積もって，適切な数の予備スレッドがプールされるように，注意して設定してください。
- 一定数のリクエスト処理スレッドは確保した状態で，それを超えるリクエスト処理スレッドの増減を最大数と最小数を指定して制御したい場合は，J2EE サーバ起動時に生成したリクエスト処理スレッド数を維持する設定にしてください。これによって，システムとして最低確保しておきたい数のリクエスト処理スレッドを確保した状態で，クライアントからのアクセスピーク時のリクエスト処理スレッド数の増減に対応できます。未使用のリクエスト処理スレッド数が予備スレッド数の最大値を超えている場合も，J2EE サーバ起動時に生成した数のリクエスト処理スレッドは維持されます。
- 一度作成したスレッドを削除しないでプールし続けたい場合は，予備スレッド数の最大数を，Web クライアントとの最大接続数と同じ値にしてください。

このほか，Web アプリケーションの同時実行スレッド数との関係についても留意してください。Web アプリケーションの同時実行スレッド数については，「16.3.2 Web アプリケーションの同時実行数を制御する」を参照してください。

16.3.2　Web アプリケーションの同時実行数を制御する

Web アプリケーションの同時実行数制御では，Web フロントシステムの場合に，Web サーバが Web ブラウザなどのクライアントから受け付けたリクエストの処理を，同時に幾つのスレッドで実行するかを制御します。

同時実行スレッド数は，URL グループ単位，Web アプリケーション単位，または Web コンテナ単位で制御できます。同時実行スレッド数は，Web サーバ連携の場合，インプロセス HTTP サーバを使用する場合，どちらの場合も制御できます。

(1)　同時実行スレッド数制御の違い

Web コンテナ単位，Web アプリケーション単位，および URL グループ単位の同時実行スレッド数制御の違いは次のとおりです。

Web コンテナ単位

Web コンテナ全体で同時にリクエストを処理するスレッド数を設定できます。

Web アプリケーション単位

Web コンテナ上で動作する Web アプリケーションごとに，同時にリクエストを処理するスレッド数を設定できます。

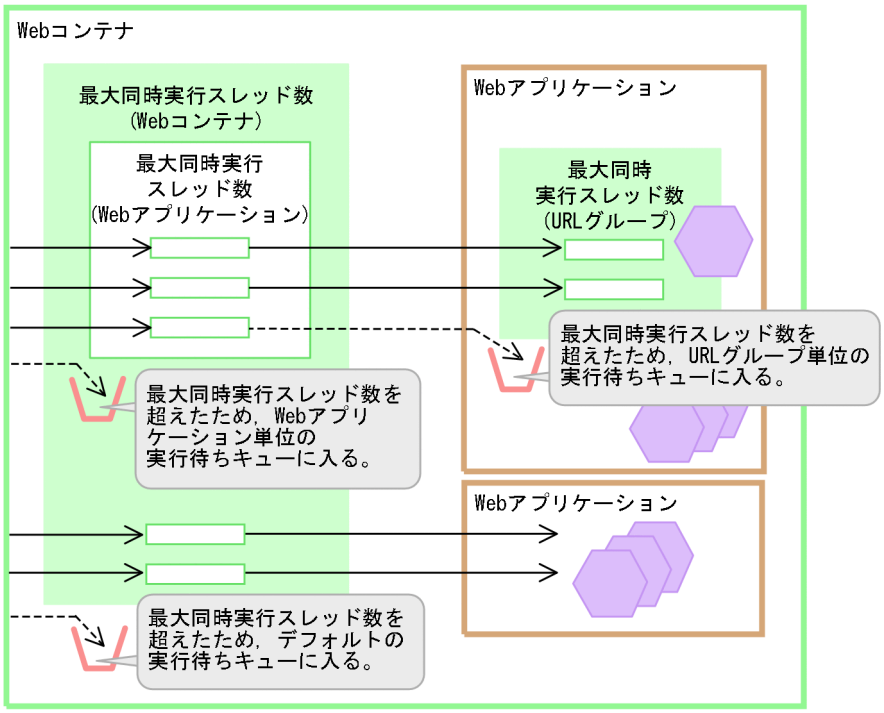
URL グループ単位

リクエストを Web アプリケーション内の特定の業務処理（業務ロジック）に対応する URL に振り分ける場合，振り分け先 URL の処理ごとに，同時にリクエストを処理するスレッド数を設定できます。

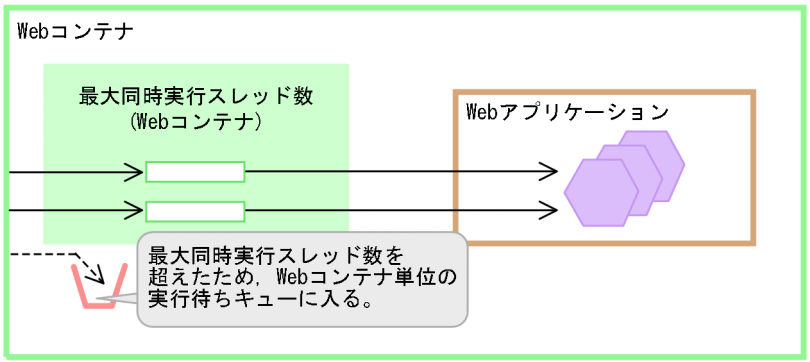
Web コンテナ単位，Web アプリケーション単位，および URL グループ単位の同時実行スレッド数の関係を次の図に示します。

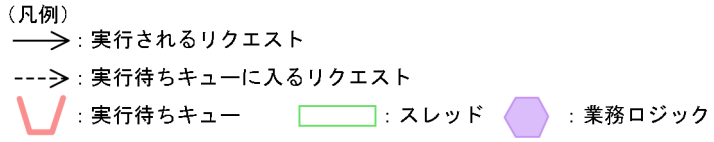
図 16-1　Web コンテナ単位，Web アプリケーション単位，および URL グループ単位の同時実行スレッド数の関係

●Webアプリケーション単位での同時実行スレッド数の制御が有効な場合



●Webアプリケーション単位での同時実行スレッド数の制御が無効な場合





注意事項

Web コンテナ単位の最大同時実行スレッド数の制御は，Web アプリケーション単位での同時実行スレッド数の制御を無効にした場合だけ有効になります。

Web アプリケーション単位での同時実行スレッド数の制御を有効にする場合，Web コンテナ単位の最大同時実行スレッド数のチェックは，Web アプリケーション単位での同時実行スレッド数制御で実施されます。詳細はマニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コ

ンテナ)」の「2.15.6　同時実行スレッド数および実行待ちキューサイズの設定例（Web アプリケーション単位)」を参照してください。

Web アプリケーションに対するリクエストの実行は，Web コンテナ単位，Web アプリケーション単位，および URL グループ単位に設定した同時実行スレッド数に制限されます。Web コンテナ単位，Web アプリケーション単位および URL グループ単位に設定した同時実行スレッド数を超えるリクエストは，それぞれの実行待ちキューに入ります。

(2)　選択の指針

同時実行スレッド数制御の単位を選択するときの指針について説明します。

なお，同時実行スレッド数を制御する機能の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.13　同時実行スレッド数の制御の概要」を参照してください。

- **Web アプリケーション単位の選択の指針**

  Web アプリケーション単位の同時実行数を制御することで，J2EE サーバが TCP 接続要求だけではなく，Web アプリケーションの実行待ちキューを管理できるようになります。このため，J2EE サーバ上で実行する Web アプリケーションが一つだけの場合でも，Web アプリケーション単位の同時実行スレッド数を設定することをお勧めします。

  Web アプリケーション単位での同時実行スレッド数の設定は，Web コンテナ単位で設定する場合に比べて，次のような利点があります。

  - Web アプリケーションごとの同時実行スレッド数に上限を設けることで，特定の業務に対応する Web アプリケーションへのリクエストが増大した場合に，その Web アプリケーションが Web コンテナ全体の処理能力を占有しないようにできます。これによって，ほかの業務も滞りなく実行できます。
  - CPU や I／O 処理に掛かる負荷が異なる複数の Web アプリケーションが Web コンテナ上にある場合，それぞれの条件に適した同時実行スレッド数が設定できます。
  - Web アプリケーションごとにリクエストの実行待ちキューサイズが設定できるので，Web アプリケーションの特徴に応じた実行待ちキュー管理ができます。また，Web アプリケーション単位の実行待ちキュー以上のリクエストが送信された場合には，クライアントに HTTP レスポンスコードで通知できます。

  なお，Web アプリケーション単位の同時実行スレッド数は，稼働中の J2EE サーバでも動的に変更できます。稼働中の J2EE サーバで実行する Web アプリケーションの同時実行スレッド数の動的変更の手順については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.17.2　同時実行スレッド数の動的変更の流れ」を参照してください。

- **URL グループ単位の選択の指針**

  Web アプリケーション単位で同時実行スレッド数を制御している場合に，さらに業務ロジック単位で同時実行スレッド数の制御をしたいときには，URL グループ単位で同時実行スレッド数を制御します。

Web アプリケーションが次のような業務ロジックを含む場合，URL グループ単位の設定を検討してください。

- ほかの処理に影響を受けないで優先して実行したい業務ロジック
- ほかの処理に比べて処理時間が掛かる，または CPU や I／O の負荷が大きい業務ロジック

URL グループ単位での同時実行スレッド数の設定は，Web アプリケーション単位だけの設定に比べて，次のような利点があります。

- 重要度が高い業務ロジック（URL グループ）には，確実に実行するためのスレッド数を割り当てられます。これによって，ほかの業務ロジックに対するリクエスト数が増大した場合も，Web アプリケーション全体の同時実行スレッド数をその業務ロジックに占有されないで，重要度が高い業務ロジックを実行できます。
- 処理時間が掛かる業務ロジック（URL グループ）の同時実行数に上限を設けることで，特定の業務ロジックによって Web アプリケーション全体の同時実行数が占有されないように制御できます。
- Web アプリケーション内に CPU や I／O の負荷が異なる複数の業務ロジック（URL グループ）がある場合は，業務ロジックに応じた同時実行数が設定できます。
- Web アプリケーション内の業務ロジック（URL グループ）ごとにリクエストの待ち行列長（実行待ちキューのキューサイズ）が設定できるので，業務ロジックの特徴に応じた実行待ちキューを管理できます。また，この URL グループ単位の実行待ちキューがあふれた場合，クライアントに HTTP レスポンスコード 503（Service Temporarily Unavailable）を通知できます。

16.4　タイムアウトを設定する

アプリケーションサーバのシステムでは，トラブル発生時にリクエストの応答が戻ってこない状態になることを防ぐために，幾つかのポイントにタイムアウトを設定できます。

この節では，システム全体でタイムアウトが設定できるポイントと，設定する場合の指針について説明します。

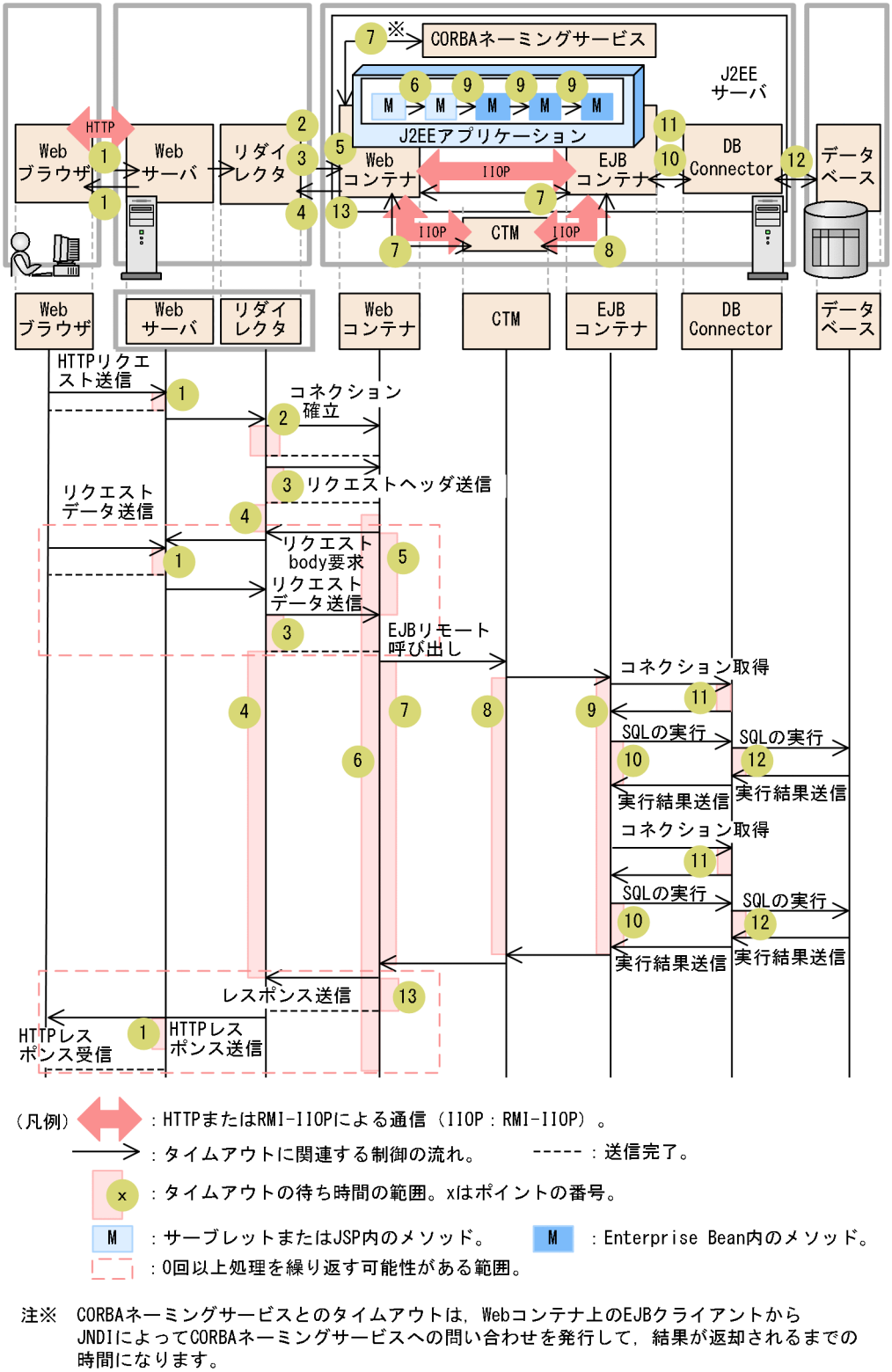
参考

TP1 インバウンド連携機能を使用して OpenTP1 からアプリケーションサーバを呼び出す場合，この節で説明する内容のほか，OpenTP1 側の設定を考慮したタイムアウトの設定が必要です。詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「4. OpenTP1 からのアプリケーションサーバの呼び出し（TP1 インバウンド連携機能)」を参照してください。

16.4.1　タイムアウトが設定できるポイント

J2EE アプリケーションを実行するシステムでは，次の図に示すポイントにタイムアウトが設定できます。なお，次の図は，クライアントが Web ブラウザの場合です。また，Web サーバ連携をする場合とインプロセス HTTP サーバを使用する場合で，ポイントが異なります。

図 16-2　タイムアウトが設定できるポイント（Web サーバ連携の場合）



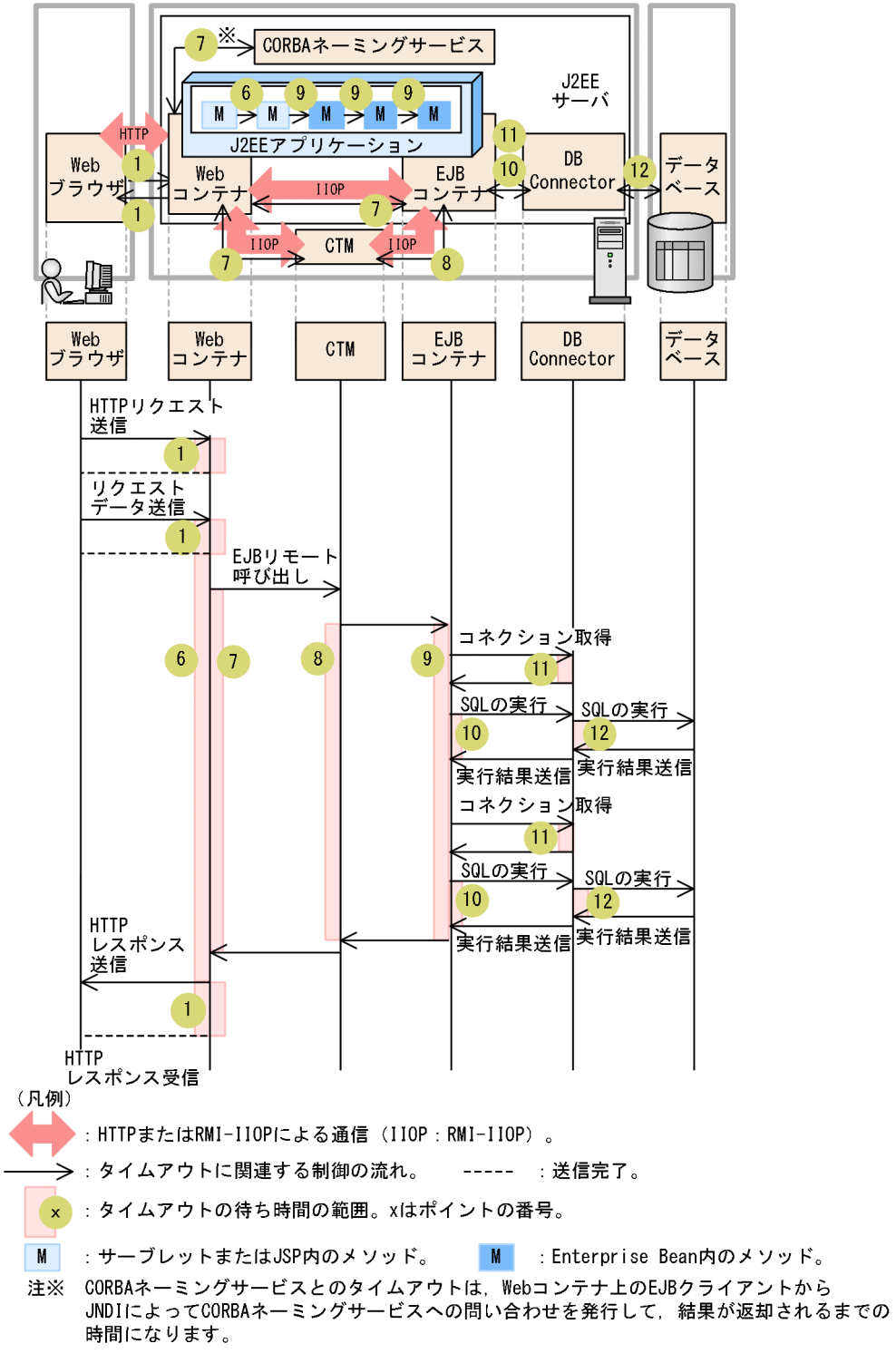
（凡例）
：HTTPまたはRMI-IIOPによる通信（IIOP：RMI-IIOP）。
：タイムアウトに関連する制御の流れ。　----- ：送信完了。
x ：タイムアウトの待ち時間の範囲。xはポイントの番号。
M ：サーブレットまたはJSP内のメソッド。　M ：Enterprise Bean内のメソッド。
：0回以上処理を繰り返す可能性がある範囲。

注※　CORBAネーミングサービスとのタイムアウトは，Webコンテナ上のEJBクライアントから
JNDIによってCORBAネーミングサービスへの問い合わせを発行して，結果が返却されるまでの
時間になります。

なお，クライアントが EJB クライアントの場合は，Web コンテナを EJB クライアントに置き換えてください。EJB クライアントからデータベースまでの範囲のタイムアウトが設定できます。

また，インプロセス HTTP サーバを使用する場合は，リダイレクタは該当しません。このため，タイムアウトを設定するポイントとして，ポイント 2～5 と 13 は該当しません。インプロセス HTTP サーバを使用する場合にタイムアウトが設定できるポイントについて，次の図に示します。

図 16-3　タイムアウトが設定できるポイント（インプロセス HTTP サーバの場合）



（凡例）

：HTTPまたはRMI-IIOPによる通信（IIOP：RMI-IIOP）。

：タイムアウトに関連する制御の流れ。　-----　：送信完了。

x　：タイムアウトの待ち時間の範囲。xはポイントの番号。

M　：サーブレットまたはJSP内のメソッド。　M　：Enterprise Bean内のメソッド。

注※　CORBAネーミングサービスとのタイムアウトは，Webコンテナ上のEJBクライアントからJNDIによってCORBAネーミングサービスへの問い合わせを発行して，結果が返却されるまでの時間になります。

それぞれのポイントに設定するタイムアウトは，次の表に示すような用途で使い分けられます。

表 16–5　各ポイントに設定するタイムアウトの目的とデフォルトのタイムアウト設定

ポイント	タイムアウトの種類	主な用途
1	サーバ側で設定するクライアントからのリクエスト受信およびクライアントへのデータ送信のタイムアウト	Web サーバ連携の場合 通信路の障害，または Web サーバの障害の検知 インプロセス HTTP サーバの場合 通信路の障害，または不正なクライアントからのアクセスの検知
2	リダイレクタ側で設定する Web コンテナへのリクエスト送信処理のうち，コネクション確立のタイムアウト	通信路の障害または Web コンテナの障害検知
3	リダイレクタ側で設定する Web コンテナへのリクエスト送信処理のうち，リクエストヘッダおよびリクエストボディ送信のタイムアウト	通信路の障害または Web コンテナの障害検知
4	リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
5	Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト	通信路の障害または Web サーバの障害検知
6	Web アプリケーションで設定するメソッドの実行時間のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど）
7	EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI ネーミングサービス呼び出しのタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
8※	EJB クライアント側で設定する CTM からの Enterprise Bean 呼び出しのタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
9	EJB で設定するメソッドの実行時間のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど）
10	EJB コンテナ側で設定するデータベースのトランザクションタイムアウト	データベースサーバの障害（サーバダウンまたはデッドロックなど）の検知，またはリソースの長時間占有防止
11	DB Connector で設定するコネクション取得時のタイムアウト	コネクション取得時の障害検知（通信路の障害またはリソース枯渇）
12	データベースのタイムアウト	データベースサーバの障害（サーバダウンまたはデッドロックなど）の検知，またはリソースの長時間占有防止
13	Web コンテナ側で設定するリダイレクタへのレスポンス送信のタイムアウト	通信路の障害またはリダイレクタの障害検知

注※　CTM を使用している場合にだけ存在するポイントです。CTM を利用しない構成の場合，ポイント 7 の範囲は Web コンテナから EJB コンテナに EJB リモート呼び出しを実行してから，EJB コンテナから Web コンテナに実行結果が送信されるまでの間になります。

これらのタイムアウトの基本的な設定指針は次のとおりです。

- タイムアウト値の設定は，呼び出し元（Web クライアントまたは EJB クライアント）に近いほど大きな値を設定するのが原則です。このため，次の関係で設定することを推奨します。
  - ポイント 1<ポイント 5
  - ポイント 4>ポイント 6>ポイント 7
  - ポイント 7=ポイント 8>ポイント 9>ポイント 10
  - ポイント 10>ポイント 11
  - ポイント 9>ポイント 12
  - ポイント 1<ポイント 13
- 4，7，10，12 のポイントのタイムアウト値を設定する場合は，呼び出し処理に通常どの程度の時間が掛かっているかを見極めた上で，呼び出す処理（業務）ごとに算出して設定してください。

なお，1～13 のポイントは，システムでの位置づけによって，次の三つに分けられます。

- Web フロントシステムで意識する必要があるポイント（1～6，および 13）

  詳細は，「16.4.2　Web フロントシステムでのタイムアウトを設定する」を参照してください。
- バックシステムで意識する必要があるポイント（7～9）

  詳細は，マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.3　バックシステムでのタイムアウトを設定する」を参照してください。
- データベース接続時に意識する必要があるポイント（10～12）

  このポイントは，さらにトランザクションでのタイムアウト，DB Connector でのタイムアウト，およびデータベースでのタイムアウトに分けて意識する必要があります。

  詳細は，マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.4　トランザクションタイムアウトを設定する」，「8.6.6　データベースでのタイムアウトを設定する」を参照してください。

それぞれのポイントでの設定については，「16.4.3　タイムアウトを設定するチューニングパラメタ」，またはマニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.8　タイムアウトを設定するチューニングパラメタ」を参照してください。

参考

それぞれのポイントのデフォルト値は次のとおりです。

ポイント	デフォルト値
1	300 秒
2	30 秒
3	100 秒
4	3,600 秒
5	600 秒

ポイント	デフォルト値
6	設定されていません。タイムアウトしません。
7	設定されていません。レスポンスを待ち続けます。
8	ポイント 7 と同じ値が Enterprise Bean 呼び出し時に自動的に引き継がれて設定されます。
9	設定されていません。タイムアウトしません。
10	180 秒
11	タイムアウトの設定個所ごとに異なります。 • 物理コネクション確立時のタイムアウト：8 秒 • コネクション枯渇時のコネクション取得要求のタイムアウト：30 秒 • コネクション障害検知時のタイムアウト：5 秒
12	データベースの種類とタイムアウトの設定個所ごとに異なります。 HiRDB の場合 ロック解放待ちタイムアウト：180 秒 レスポンスタイムアウト：0 秒（HiRDB クライアントは HiRDB サーバからの応答があるまで待ち続けます） リクエスト間隔タイムアウト：600 秒 Oracle の場合（グローバルトランザクションを使用するとき） ロック解放待ちタイムアウト：60 秒 SQL Server の場合 メモリ取得待ちタイムアウト：-1（-1 を指定した場合の動作は，SQL Server のドキュメントを参照してください） ロック解放待ちタイムアウト：-1（ロックが解放されるまで待ち続けます） XDM/RD E2 の場合 ロック解放待ちタイムアウト：なし（タイムアウト時間を監視しません） SQL 実行 CPU 時間タイムアウト：10 秒 SQL 実行経過時間タイムアウト：0 秒（タイムアウト時間を監視しません） トランザクション経過時間タイムアウト：600 秒 レスポンスタイムアウト：0 秒（HiRDB クライアントは XDM/RD E2 サーバからの応答があるまで待ち続けます）
13	600 秒

16.4.2 Web フロントシステムでのタイムアウトを設定する

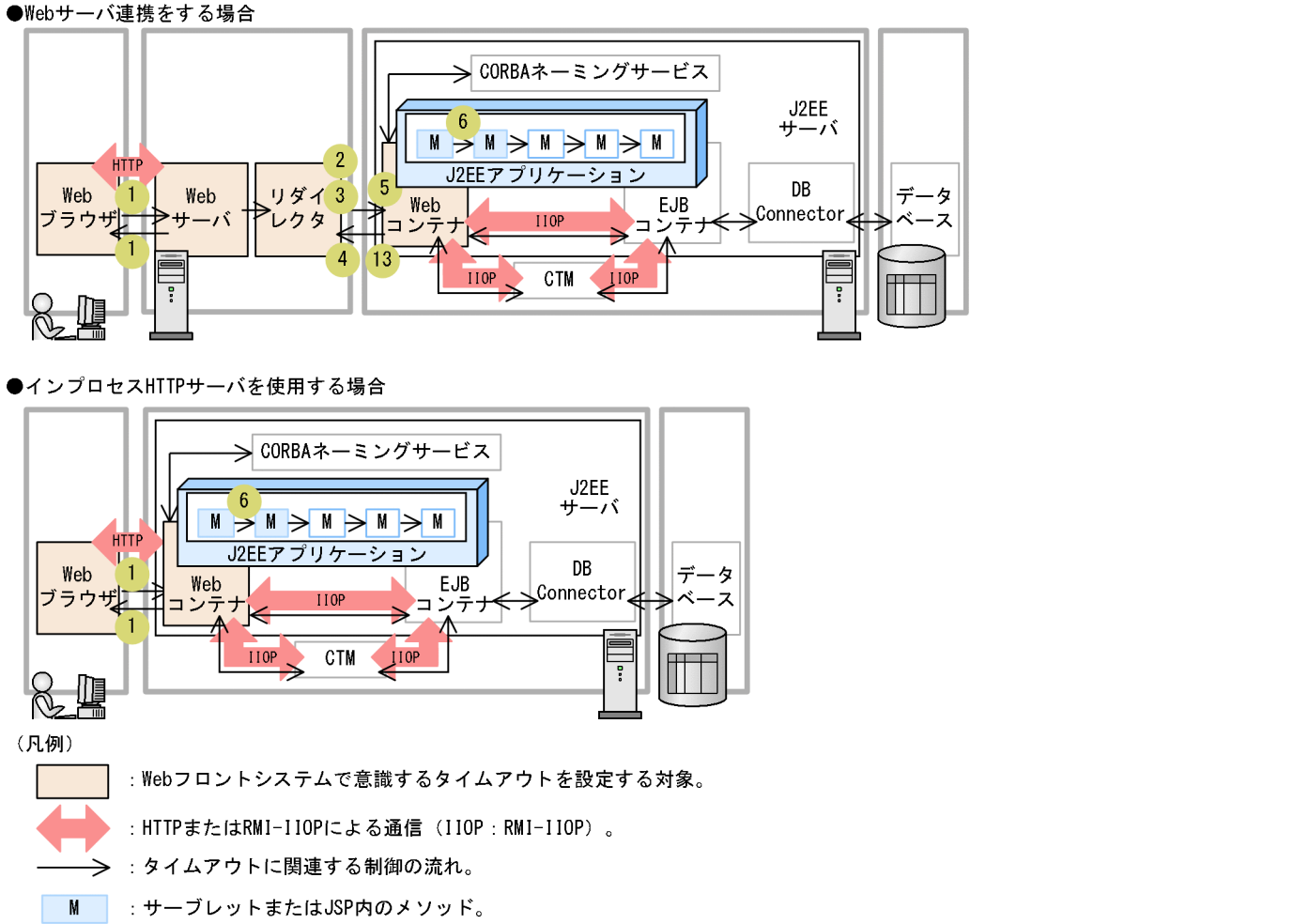
ここでは，Web フロントシステムでのタイムアウトの設定について説明します。

Web フロントシステムのタイムアウトを設定する場合は，システム全体のタイムアウトのうち，次の図に示す，1～6 および 13 のポイントについて意識する必要があります。この番号は，図 16-2 または図 16-3 と対応しています。

ポイント

インプロセス HTTP サーバを使用する場合，設定できるのは 1 と 6 のポイントです。2～5，および 13 のポイントは該当しません。

図 16-4　Web フロントシステムの場合に意識するタイムアウトのポイント（1～6，および 13 のポイント）



- Web サーバでのクライアントからのリクエスト受信，およびクライアントへのデータ送信待ち時間（1 のポイント）

  Web ブラウザからの要求が滞った場合に，タイムアウトによってリダイレクタのリソースを解放します。また，Web ブラウザへの応答が滞った場合（Web ブラウザが受信しない場合）に，タイムアウトによってリダイレクタおよび J2EE サーバ内の Web コンテナのリソースを解放します。

  Web サーバ連携の場合，これらの待ち時間には，同じ値が設定されます。

  インプロセス HTTP サーバを使用する場合，クライアントからのリクエスト受信待ち時間と，クライアントへのデータ送信待ち時間には，それぞれ異なる値を指定できます。

- Web サーバに登録したリダイレクタでの Web コンテナへのリクエスト送信待ち時間（2 および 3 のポイント）

リダイレクタから Web コンテナへのリクエスト送信時に，Web コンテナ自体のトラブル，またはリダイレクタと Web コンテナ間の通信路でのトラブルによって制御が戻らなくなった場合に，タイムアウトによってリダイレクタのリソースを解放します。また，同時に Web ブラウザにエラーを通知します。Web サーバと連携する場合にだけ設定できるポイントです。

ポイント 2 は Web コンテナとのコネクション確立の待ち時間，ポイント 3 は Web コンテナへのリクエスト送信処理の待ち時間です。

- **Web サーバに登録したリダイレクタでの Web コンテナからのデータ受信待ち時間（4 のポイント）**

  J2EE アプリケーションで何かのトラブルが発生して制御が戻らなくなった場合に，タイムアウトによってリダイレクタのリソースを解放します。また，同時に Web ブラウザにエラーを通知します。Web サーバと連携する場合にだけ設定できるポイントです。

  ポイント

  設定の単位はワーカです。このため，業務によって処理に掛かる時間が異なる場合は，業務に対応する Web アプリケーション単位でワーカを定義してタイムアウトを設定することをお勧めします。

- **Web コンテナでのリダイレクタからのデータ受信待ち時間（5 のポイント）**

  ブラウザからの要求が滞ったときに，J2EE サーバ（Web コンテナ）のリソースを解放します。Web サーバと連携する場合にだけ設定できるポイントです。

- **Web コンテナ上でのリクエスト処理待ち時間（6 のポイント）**

  J2EE アプリケーションの実行時間監視機能を利用します。

  マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.7　J2EE アプリケーションのメソッドタイムアウトを設定する」を参照してください。

- **Web コンテナからリダイレクタへのレスポンス送信待ち時間（13 のポイント）**

  Web コンテナからリダイレクタへのレスポンス送信時に，リダイレクタ自体のトラブル，またはリダイレクタと Web コンテナ間の通信路でのトラブルによって制御が戻らなくなった場合に，タイムアウトによって Web コンテナのリソースを解放します。また，同時に Web ブラウザにエラーを通知します。Web サーバと連携する場合にだけ設定できるポイントです。

16.4.3　タイムアウトを設定するチューニングパラメタ

ここでは，タイムアウトの設定で使用するチューニングパラメタの設定方法についてまとめて示します。

(1)　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト

図 16-2 または図 16-3 のポイント 1 のタイムアウトを設定するチューニングパラメタです。使用する Web サーバによって設定個所が異なります。

Web サーバ連携の場合は，Web サーバ単位に設定します。ファイル編集によって設定します。

表 16–6　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ（Web サーバ連携の場合）

設定項目	設定個所
クライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト	httpsd.conf の Timeout ディレクティブ

注　Web サーバとして Microsoft IIS を使用しているときには，isapi_redirect.conf の receive_client_timeout キーを編集してください。

インプロセス HTTP サーバの場合は，J2EE サーバ単位に設定します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 16–7　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ（インプロセス HTTP サーバの場合）

設定項目	設定対象	設定個所（パラメタ名）
クライアントからのリクエスト受信のタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.receive_timeout
クライアントへのデータ送信のタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.send_timeout

(2)　リダイレクタ側で設定する Web コンテナへのデータ送信のタイムアウト

図 16-2 のポイント 2，およびポイント 3 のタイムアウトを設定するチューニングパラメタです。リダイレクタ側で設定するタイムアウトのチューニングパラメタについて説明します。なお，これらのチューニングパラメタは，Web サーバ連携の場合だけ指定できます。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 16–8　リダイレクタ側で設定するタイムアウトのチューニングパラメタ

ポイント	設定項目	設定対象	設定個所（パラメタ名）※
2	リクエスト送信時の Web コンテナに対するコネクション確立のタイムアウト	論理 Web サーバ（web-server）	JkConnectTimeout
3	リクエスト送信のタイムアウト	論理 Web サーバ（web-server）	JkSendTimeout

注※　Web サーバとして Microsoft IIS を使用しているときには，isapi_redirect.conf の connect_timeout キーを編集してください。

(3) リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト

図 16-2 のポイント 4 のタイムアウトを設定するチューニングパラメタです。

リダイレクタのワーカ定義単位で設定します。リダイレクタ側で設定するタイムアウトのチューニングパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 16–9 リダイレクタ側で設定するタイムアウトのチューニングパラメタ

設定項目	設定対象	設定個所（パラメタ名）
レスポンスデータ待ちの通信タイムアウト	論理 Web サーバ（web-server）	worker.<ワーカ名>.receive_timeout

Web サーバ連携の場合だけ指定できます。

(4) Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト

図 16-2 のポイント 5 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 16–10 Web コンテナ側で設定するタイムアウトのチューニングパラメタ

設定項目	設定対象	設定個所（パラメタ名）
リダイレクタからの応答待ちのタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.receive_timeout

Web サーバ連携の場合だけ指定できます。

(5) Web コンテナ側で設定するリダイレクタへのデータ受信のタイムアウト

図 16-2 のポイント 13 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 16–11　Web コンテナ側で設定するタイムアウトのチューニングパラメタ

設定項目	設定対象	パラメタ名
レスポンス送信処理のタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.send_timeout

Web サーバ連携の場合だけ指定できます。

16.5 Web アプリケーションの動作を最適化する

この節では，Web アプリケーションのパフォーマンスチューニングの方法について説明します。Web フロントシステムの場合に検討してください。

ここでは，次の 3 種類のチューニング方法について説明します。

- 静的コンテンツと Web アプリケーションの配置を切り分ける
- 静的コンテンツをキャッシュする
- リダイレクタを使用してリクエストを振り分ける（Web サーバ連携の場合）

16.5.1 Web アプリケーションの動作を最適化するためのチューニングパラメタ

ここでは，Web アプリケーションの動作を最適化するために使用するチューニングパラメタの設定方法についてまとめて示します。

(1) 静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ

静的コンテンツと Web アプリケーションの配置の切り分けは，Web サーバの動作を定義するファイルのパラメタとして指定します。設定個所，ファイルおよびパラメタは，使用する Web サーバの種類によって異なります。

Web サーバ連携で HTTP Server を使用している場合は，リダイレクタモジュールを使用して切り分けます。インプロセス HTTP サーバを使用している場合は，リバースプロキシサーバ（HTTP Server）に配置しているリバースプロキシモジュールを使用して切り分けます。

設定方法および設定個所を次に示します。

表 16–12 静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ

使用する Web サーバ	設定方法	設定個所
HTTP Server （リダイレクタモジュールを使用した切り分け※1）	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 Web サーバ（web-server） **パラメタ名** JkMount

使用する Web サーバ	設定方法	設定個所
インプロセス HTTP サーバ （リバースプロキシモジュールを使用した切り分け）	ファイル編集	**定義ファイル** httpsd.conf **設定対象** リバースプロキシサーバ **パラメタ名** ProxyPass ディレクティブ※2

注※1　Web サーバとして Microsoft IIS を使用しているときは，uriworkermap.properties で設定します。

注※2　httpsd.conf の詳細については，マニュアル「HTTP Server」を参照してください。

(2)　静的コンテンツをキャッシュするためのチューニングパラメタ

静的コンテンツをキャッシュするためのチューニングパラメタについて説明します。これらのチューニングパラメタは，Web コンテナ単位または Web アプリケーション単位に設定します。

Web コンテナ単位に設定するチューニングパラメタの設定方法について，次の表に示します。これらの項目は，Smart Composer 機能で設定します。

表 16–13　静的コンテンツをキャッシュするためのチューニングパラメタ（Web コンテナ単位で設定する項目）

設定項目	設定個所
静的コンテンツのキャッシュを使用するかどうかの選択	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.static_content.cache.enabled
Web アプリケーション単位のメモリサイズの上限値の設定	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.static_content.cache.size
キャッシュする静的コンテンツのファイルサイズの上限値の設定	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.static_content.cache.filesize.threshold

Web アプリケーション単位に設定するチューニングパラメタについて示します。Web アプリケーション単位に設定する項目は，web.xml を直接編集するか，サーバ管理コマンドを使用して設定します。デプロイ前の Web アプリケーションに設定する場合は，web.xml を編集してください。デプロイ後の Web アプリケーションに設定する場合は，サーバ管理コマンド（cjsetappprop）を使用してください。

設定内容を次の表に示します。

表 16–14　静的コンテンツをキャッシュするためのチューニングパラメタ（Web アプリケーション単位で設定する項目）

設定項目	設定内容※
静的コンテンツのキャッシュを使用するかどうかの選択	<param-name>タグ com.hitachi.software.web.static_content.cache.enabled <param-value>タグ （設定値）
Web アプリケーション単位のメモリサイズの上限値の設定	<param-name>タグ com.hitachi.software.web.static_content.cache.size <param-value>タグ （設定値）
キャッシュする静的コンテンツのファイルサイズの上限値の設定	<param-name>タグ com.hitachi.software.web.static_content.cache.filesize.threshold <param-value>タグ （設定値）

注

（設定値）に設定できる値の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.19.2　DD での定義（Web アプリケーション単位での設定)」を参照してください。

注※

web.xml を直接編集する場合，<web-app>タグ内に<context-param>タグを追加して，<context-param>タグ内に<param-name>タグおよび<param-value>タグを追加します。

サーバ管理コマンドを使用する場合，WAR 属性ファイルの<hitachi-war-property>タグ内に<context-param>タグを追加して，<context-param>タグ内に<param-name>タグおよび<param-value>タグを追加します。

(3)　リダイレクタによってリクエストを振り分けるためのチューニングパラメタ

リダイレクタによってリクエストを振り分けるためのチューニングパラメタは，Web サーバの動作を定義するファイルのパラメタとして指定します。

なお，この定義は，Web サーバ連携の場合だけできます。インプロセス HTTP サーバを使用している場合は定義できません。

設定方法および設定個所を次に示します。

表 16–15　リダイレクタによってリクエストを振り分けるためのチューニングパラメタ

設定項目	設定方法	設定個所
URL パターンのマッピング定義※	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** JkMount

注※　Web サーバとして Microsoft IIS を使用しているときは，uriworkermap.properties で設定します。

16.6　そのほかの項目のチューニング

ここでは，前の節までに説明した項目以外のチューニング項目について説明します。

ここで説明するのは，次の項目です。

- Persistent Connection についてのチューニング

この項目は，Web フロントシステムの場合で，インプロセス HTTP サーバを使用するときにチューニングを検討してください。

HTTP/1.1 では，Web クライアントと Web サーバ間で確立した TCP コネクションを持続して，複数の HTTP リクエスト間で使用し続けるための Persistent Connection が定義されています。Persistent Connection を使用することによって，Web クライアントと Web サーバ間でコネクション接続に掛かる時間を短縮し，通信トラフィックを軽減できます。

ただし，Persistent Connection を使用すると，特定の Web クライアントがリクエスト処理スレッドを占有することになるため，サーバ全体の処理性能が低下することがあります。このため，Persistent Connection を有効に活用し，かつサーバ処理性能を維持できるようにチューニングする必要があります。

インプロセス HTTP サーバを使用する場合，Persistent Connection について，次の項目がチューニングできます。

- Persistent Connection 数の上限値

  この上限値を超える TCP コネクションについては，リクエスト処理終了後に切断されます。これによって，新規接続を処理するスレッドが確保でき，リクエスト処理スレッドを特定のクライアントに占有されることを防げます。
- Persistent Connection のリクエスト処理回数の上限値

  同じ Web クライアントから連続してリクエスト要求があった場合も，この上限値を超えると，リクエスト処理終了後に一度 TCP コネクションが切断されます。

  これによってリクエスト処理スレッドを特定のクライアントに占有され続けることを防げます。
- Persistent Connection のタイムアウト

  Persistent Connection のリクエスト待ち時間にタイムアウトを設定できます。指定したタイムアウト時間を超えてリクエスト処理要求がない場合は，TCP コネクションが切断されます。これによって，使用されていない状態で TCP コネクションが占有され続けることを防げます。

これらの項目は，Smart Composer 機能で使用する簡易構築定義ファイルのパラメタとして指定します。Persistent Connection について設定するチューニングパラメタについて次の表に示します。

表 16–16　Persistent Connection について設定するチューニングパラメタ

設定項目	設定個所
Persistent Connection 数の上限値	定義ファイル 　簡易構築定義ファイル

設定項目	設定個所
	設定対象 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.connector.inprocess_http.persistent_connection.max_connections
リクエスト処理回数の上限値	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.connector.inprocess_http.persistent_connection.max_requests
タイムアウト	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.connector.inprocess_http.persistent_connection.timeout

なお，各パラメタの詳細については，「11.2　論理 J2EE サーバで指定できるパラメタ」を参照してください。

16.7　アプリケーションサーバが使用する TCP/UDP のポート番号

デフォルト値が「(浮動)」のポートは，ポート番号を明示的に固定しない場合にアプリケーションサーバによって自動的に番号が付けられるポートです。

アプリケーションサーバが使用する TCP/UDP のポート番号の説明を次の表に示します。なお，ご使用の OS によっては，ネットワーク単位ではなくホスト単位でファイアウォールが設定されているものがあります。これらのファイアウォールでは，localhost（127.0.0.1）以外との通信は，同一ホスト内でもファイアウォールのフィルタリングの対象になる場合があります。この場合は，ホスト内でしか通信しないポートであっても，フィルタで通信を許可する設定にしてください。

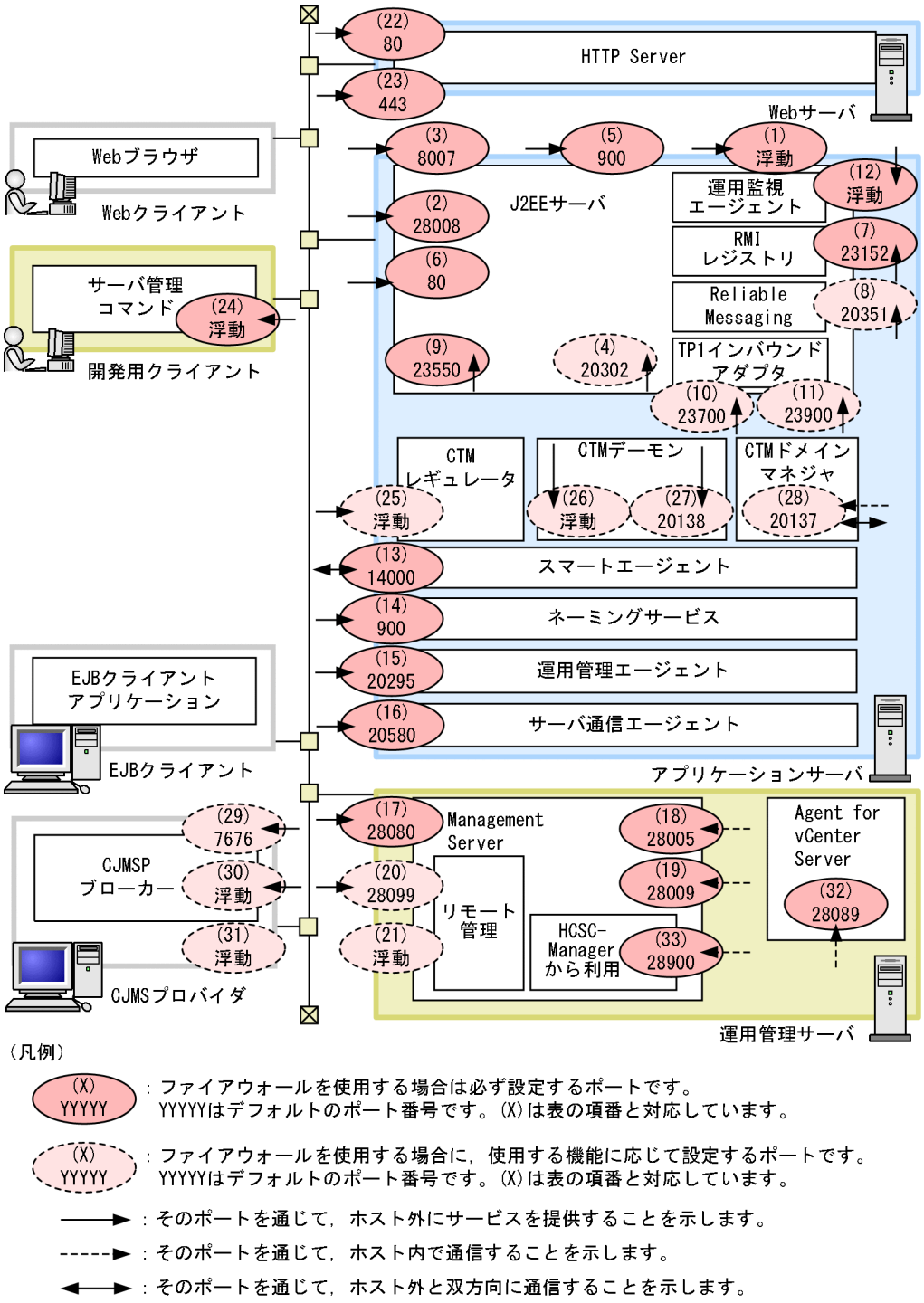
表 16–17　アプリケーションサーバが使用する TCP/UDP のポート番号

項番	プロセス	説明	デフォルト値
(1)	J2EE サーバ	EJB コンテナのリクエスト受付ポート。	（浮動）
(2)		管理用通信ポート。	28008
(3)		Web サーバ（リダイレクタ）からのリクエスト受付ポート。	8007
(4)		トランザクションサービス使用時のトランザクションリカバリ処理通信ポート。 トランザクションサービス使用時に必要です。	20302
(5)		インプロセスで起動するネーミングサービスのリクエスト受付ポート。	900
(6)		インプロセス HTTP サーバのリクエスト受付ポート。 インプロセス HTTP サーバを使用するときに必要です。	80
(7)		RMI レジストリのリクエスト受付ポート。	23152
(8)		共有キューを使用して複数システム間でのアプリケーション連携をする場合のイベント受信用ポート。	20351
(9)		稼働情報取得時のリクエスト受付ポート。	23550
(10)		OpenTP1 からの RPC 要求を待ち受けるポート。	23700
(11)		OpenTP1 からの同期点要求を待ち受けるポート。	23900
(12)	運用監視エージェント	運用監視エージェントの通信用ポート。	（浮動）
(13)	スマートエージェント	スマートエージェントの通信用ポート環境変数。 UDP による双方向通信に必要です。	14000
(14)	ネーミングサービス	ネーミングサービスのリクエスト受付ポート引数（TPBroker が利用）。	900
(15)	運用管理エージェント	運用管理エージェントが Management Server との通信に使用するポート。	20295
(16)	サーバ通信エージェント	サーバ通信エージェントが仮想サーバマネージャとの通信に使用するポート。	20580
(17)	Management Server	Management Server の http ポート。	28080
(18)		Management Server の終了要求ポート。	28005

項番	プロセス	説明	デフォルト値
		ホスト内通信に必要です。	
(19)		Management Server の内部通信ポート。 ホスト内通信に必要です。	28009
(20)		Manager リモート管理機能への接続ポート。	28099
(21)		Manager リモート管理機能へのクライアント接続ポート。	（浮動）
(22)	HTTP Server	HTTP Server の http ポート。	80
(23)		HTTP Server の https ポート。	443
(24)	サーバ管理コマンド	サーバ管理コマンドが J2EE サーバと通信するポート。	（浮動）
(25)	CTM レギュレータ	CTM レギュレータが EJB クライアントからのリクエストを受け付けるポートの基底値。基底値＋プロセス数だけ使用します。 CTM 使用時に必要です。	（浮動）
(26)	CTM デーモン	CTM デーモンが EJB クライアントからのリクエストを受け付けるポート。 CTM 使用時に必要です。	（浮動）
(27)		CTM デーモンがほかのデーモンや J2EE サーバなどと通信するポート。 CTM 使用時に必要です。	20138
(28)	CTM ドメインマネジャ	CTM ドメインマネジャがほかの CTM ドメインマネジャと通信するポート。 CTM 使用時に，TCP および UDP 通信（ブロードキャスト）をするために必要です。	20137
(29)	CJMSP ブローカー	CJMS プロバイダのブローカーがリソースアダプタやコマンドからのリクエストを受け付けるためのポート。	7676
(30)		CJMS プロバイダのブローカーがリソースアダプタとコネクションを確立するためのポート。	（浮動）
(31)		CJMS プロバイダのブローカーがコマンドとコネクションを確立するためのポート。	（浮動）
(32)	Management Server	08-50 モードの仮想サーバマネージャ（Management Server）が vCenter Server の接続処理を行うための，内部で起動するプロセス（Agent for vCenter Server）のポート。	28089
(33)		HCSC-Manager から利用される Management Server の内部通信用ポート。	28900

アプリケーションサーバのプロセスが使用する TCP/UDP のポート番号について，次の図に示します。(x)は表の項番と対応しています。

図 16-5　アプリケーションサーバが使用する TCP/UDP のポート番号



（凡例）

(X) YYYYY：ファイアウォールを使用する場合は必ず設定するポートです。
YYYYYはデフォルトのポート番号です。(X)は表の項番と対応しています。

(X) YYYYY：ファイアウォールを使用する場合に，使用する機能に応じて設定するポートです。
YYYYYはデフォルトのポート番号です。(X)は表の項番と対応しています。

――►：そのポートを通じて，ホスト外にサービスを提供することを示します。

-----►：そのポートを通じて，ホスト内で通信することを示します。

◄――►：そのポートを通じて，ホスト外と双方向に通信することを示します。

これ以外の凡例については，マニュアル「アプリケーションサーバ システム設計ガイド」の「3.2　システム構成の説明について」を参照してください。

ポート番号の指定個所を次の表に示します。表の項番は図中の項番と対応しています。

表 16–18　アプリケーションサーバが使用する TCP/UDP のポート番号の指定個所

項番	定義ファイル	設定対象	パラメタ名※1
(1)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	vbroker.se.iiop_tp.scm.iiop_tp.listener.port
(2)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	ejbserver.http.port
(3)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.port
(4)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	ejbserver.distributedtx.recovery.port
(5)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.port
(6)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	ejbserver.rmi.naming.port
(7)	Connector 属性ファイル	Reliable Messaging	<config-property>タグに指定する RMSHPort※2
(8)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	ejbserver.rmi.remote.listener.port
(9)	Connector 属性ファイル	TP1 インバウンドアダプタ	<config-property>タグに指定する scd_port※3
(10)	Connector 属性ファイル	TP1 インバウンドアダプタ	<config-property>タグに指定する trn_port※3
(11)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	mngagent.connector.port
(12)	簡易構築定義ファイル	論理スマートエージェント（smart-agent）	smartagent.port
(13)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	ejbserver.naming.port
(14)	adminagent.properties	運用管理エージェント	adminagent.adapter.port キー
(15)	sinaviagent.properties※4	サーバ通信エージェント	sinaviagent.port キー
(16)	mserver.properties	Management Server	webserver.connector.http.port キー
(17)	mserver.properties	Management Server	webserver.shutdown.port キー
(18)	mserver.properties	Management Server	webserver.connector.ajp13.port キー
(19)	mserver.properties	Management Server	com.cosminexus.mngsvr.management.port キー
(20)	mserver.properties	Management Server	com.cosminexus.mngsvr.management.listen.port キー
(21)	簡易構築定義ファイル	論理 Web サーバ（web-server）	Listen

項番	定義ファイル	設定対象	パラメタ名※1
(22)	簡易構築定義ファイル	論理 Web サーバ（web-server）	Listen
(23)	usrconf.properties（サーバ管理コマンド用システムプロパティファイル）	サーバ管理コマンド	vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー
(24)	簡易構築定義ファイル	論理 CTM（component-transaction-monitor）	ctm.RegOption
(25)	簡易構築定義ファイル	論理 CTM（component-transaction-monitor）	ctm.EjbPort
(26)	簡易構築定義ファイル	論理 CTM（component-transaction-monitor）	ctm.port
(27)	簡易構築定義ファイル	論理 CTM ドメインマネジャ（ctm-domain-manager）	cdm.port
(28)	config.properties	CJMSP ブローカー	imq.portmapper.port キー
(29)	config.properties	CJMSP ブローカー	imq.jms.tcp.port キー
(30)	config.properties	CJMSP ブローカー	imq.admin.tcp.port キー
(31)	vmx.properties	08-50 モードの仮想サーバマネージャ（Management Server）	vmx.vcenterserver.agent.port キー
(32)	mserver.properties	Management Server	ejbserver.naming.port キー

注※1　設定ファイルが簡易構築定義ファイルの場合は，<configuration>タグ内の<param-name>の指定値を指します。

注※2　RMSHPort は，リソースアダプタ Reliable Messaging のプロパティ定義で指定するコンフィグレーションプロパティです。RMSHPort については，マニュアル「Reliable Messaging」の「6.　コンフィグレーションプロパティ」を参照してください。

注※3　scd_port および trn_port は，リソースアダプタ TP1 インバウンドアダプタのプロパティ定義で指定するコンフィグレーションプロパティです。scd_port および trn_port については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「4.12.2　リソースアダプタの設定」を参照してください。

注※4　サーバ通信エージェントの詳細については，サーバ通信エージェントのドキュメントを参照してください。

参考

運用管理ポータルまたはファイル編集によってアプリケーションサーバを構築している場合の TCP/UDP のポートの設定個所を次に示します。

表 16-19　運用管理ポータルまたはファイル編集によってアプリケーションサーバを構築している場合の TCP/UDP のポートの設定個所

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(1)	[EJB コンテナの設定] 画面の「オプション」の「通信ポート番号」	usrconf.properties の vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(2)	[J2EE サーバの基本設定] 画面の「コンテナの設定」の「簡易 Web サーバのポート番号」	usrconf.properties の ejbserver.http.port キー
(3)	[Web コンテナの設定] 画面の「Web サーバとの接続」の「ポート番号」	usrconf.properties の webserver.connector.ajp13.port キー
(4)	[トランザクションの設定] 画面の「トランザクションに関する設定」の「JTA リカバリの固定ポート番号」	usrconf.properties の ejbserver.distributedtx.recovery.port キー
(5)	[ネーミングの設定] 画面の「インプロセス選択時の設定」の「ポート番号」	usrconf.properties の ejbserver.naming.port キー
(6)	[Web コンテナの設定] 画面の「インプロセス HTTP サーバ機能の使用」の「ポート番号」	usrconf.properties の webserver.connector.inprocess_http.port キー
(7)	[通信の設定] 画面の「RMI レジストリの設定」の「ポート番号」	usrconf.properties の ejbserver.rmi.naming.port キー
(8)	Connector 属性ファイルの<config-property>タグに指定する RMSHPort※1	Connector 属性ファイルの<config-property>タグに指定する RMSHPort※1
(9)	[通信の設定] 画面の「RMI レジストリの設定」の「通信ポート番号」	usrconf.properties の ejbserver.rmi.remote.listener.port キー
(10)	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの<config-property>タグに指定する scd_port	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの<config-property>タグに指定する scd_port
(11)	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの<config-property>タグに指定する trn_port	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの<config-property>タグに指定する trn_port
(12)	[J2EE コンテナの設定] 画面の「運用監視エージェントの設定」の「ポート番号」	mngagent.<実サーバ名>.properties の mngagent.connector.port キー
(13)	[スマートエージェントの設定] 画面の「スマートエージェントに関する設定」の「監視ポート番号」	環境変数 OSAGENT_PORT
(14)	• CORBA ネーミングサービスをインプロセスで起動する場合 [J2EE サーバの基本設定] 画面の「利用するネーミングサービスの設定」の「インプロセス用のポート番号」 • CORBA ネーミングサービスをアウトプロセスで起動する場合 [ホスト内のサーバの設定] 画面の「ネーミングサービスの設定」の「ネーミングサービスのポート番号」	• CORBA ネーミングサービスをインプロセスまたはアウトプロセスで自動起動する場合 usrconf.properties の ejbserver.naming.port キー • CORBA ネーミングサービスを手動起動する場合 nameserv コマンドのコマンド引数に「-Dvbroker.se.iiop_tp.scm.iiop_tp.listener.port=<ポート番号>」を指定。
(15)	adminagent.properties の adminagent.adapter.port キー	adminagent.properties の adminagent.adapter.port キー

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(16)	sinaviagent.properties の sinaviagent.port キー※2	sinaviagent.properties の sinaviagent.port キー
(17)	[ネットワークの設定] 画面の「Management Server 接続 HTTP ポート番号」	mserver.properties の webserver.connector.http.port キー
(18)	[ネットワークの設定] 画面の「Management Server 終了要求受信ポート番号」	mserver.properties の webserver.shutdown.port キー
(19)	[ネットワークの設定] 画面の「Management Server 内部通信用ポート番号」	mserver.properties の webserver.connector.ajp13.port キー
(20)	mserver.properties の com.cosminexus.mngsvr.management.port キー	mserver.properties の com.cosminexus.mngsvr.management.port キー
(21)	mserver.properties の com.cosminexus.mngsvr.management.listen.port キー	mserver.properties の com.cosminexus.mngsvr.management.listen.port キー
(22)	[ホスト内のサーバの設定] 画面の「J2EE サーバの設定」の「ポート番号」の「http」	httpsd.conf の Listen ディレクティブまたは Port ディレクティブ
(23)	[ホスト内のサーバの設定] 画面の「J2EE サーバの設定」の「ポート番号」の「https」	httpsd.conf の Listen ディレクティブまたは Port ディレクティブ
(24)	usrconf.properties（サーバ管理コマンド用システムプロパティファイル）の vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー	usrconf.properties（サーバ管理コマンド用システムプロパティファイル）の vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー
(25)	[レギュレータの設定] 画面の「CTM レギュレータの設定」の「設定ファイル」	ctmregltd コマンドまたは ctmstart コマンドの引数-CTMEjbPort
(26)	[スケジューリングの設定] 画面の「詳細設定」の「EJB リクエスト受信ポート番号」	ctmstart コマンドの引数-CTMEjbPort
(27)	[CTM の基本設定] 画面の「基本設定」の「ポート番号」	ctmstart コマンドの引数-CTMPort
(28)	[CTM ドメインマネジャの基本設定] 画面の「ポート番号」	ctmdmstart コマンドの引数-CTMPort
(29)	config.properties の imq.portmapper.port キー	config.properties の imq.portmapper.port キー
(30)	config.properties の imq.jms.tcp.port キー	config.properties の imq.jms.tcp.port キー
(31)	config.properties の imq.admin.tcp.port キー	config.properties の imq.admin.tcp.port キー
(32)	vmx.properties の vmx.vcenterserver.agent.port キー	vmx.properties の vmx.vcenterserver.agent.port キー
(33)	mserver.properties の ejbserver.naming.port キー	mserver.properties の ejbserver.naming.port キー

注※1　RMSHPort は，リソースアダプタ Reliable Messaging のプロパティ定義で指定するコンフィグレーションプロパティです。RMSHPort については，マニュアル「Reliable Messaging」の「6.　コンフィグレーションプロパティ」を参照してください。

注※2　サーバ通信エージェントの詳細については，サーバ通信エージェントのドキュメントを参照してください。

注意事項

サーバの待ち受けポートの注意事項（UNIX の場合）

UNIX の場合，次の条件がすべて重なるときは，待ち受けをしていない TCP ポートに対して，接続に成功してしまうことがあります。

- 待ち受けをしていないポートに対して接続試行を実行する
- 接続対象が自ホストであり，かつ一時ポート番号の範囲（OS が動的に割り当てるポートの範囲）である

この現象が発生した場合，想定したプロセスとの通信ができないで，タイムアウトなどが発生します。この現象を回避するためには，サーバの待ち受けポートに一時ポート番号の範囲以外の値を指定してください。一時ポート番号の範囲は，次のファイルで確認できます。

AIX の場合

最小値（32768）：no -o tcp_ephemeral_low

最大値（65535）：no -o tcp_ephemeral_high

Linux の場合

/proc/sys/net/ipv4/ip_local_port_range

なお，サーバの待ち受けポートの設定方法は，各 OS のマニュアルを参照してください。

17 Web アプリケーションで使用するコマンド

この章では，Web アプリケーションで使用するコマンドについて説明します。ここでは，推奨モードと異なる内容だけ説明します。

17.1 Web アプリケーションで使用するコマンドの詳細

Web アプリケーションで使用するコマンドの格納先を次に示します。

コマンドの格納先

Web アプリケーションで使用するコマンドは，次のディレクトリに格納されています。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥bin¥V9
- UNIX の場合
  /opt/Cosminexus/CC/web/bin/V9

17.2 cjjspc（JSP の事前コンパイル）

推奨モードと V9 互換モードでは，JSP の事前コンパイルのコマンドが異なります。V9 互換モードで JSP の事前コンパイルを実行する場合は，V9 互換モード用の cjjspc コマンドを使用してください。cjjspc コマンドの入力形式，機能などについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル）」を参照してください。

18 基本・開発機能の互換機能（EJB 2.1 と Servlet 2.4 でのアノテーションの利用）

この章では，基本・開発機能の互換機能のうち，EJB 2.1 と Servlet 2.4 でのアノテーションの利用について説明します。

この機能は互換機能のため推奨しません。Java EE 標準の，EJB3.0/Servet2.5 以降でのアノテーションの利用を検討してください。

18.1　EJB 2.1 と Servlet 2.4 でのアノテーションの利用

アプリケーションサーバでは，EJB 2.1 と Servlet 2.4 でアプリケーションサーバ独自のアノテーションを利用できます。この節では，EJB 2.1 と Servlet 2.4 でのアノテーションの利用について説明します。

この節の構成を次の表に示します。

表 18–1　この節の構成（EJB 2.1 と Servlet 2.4 でのアノテーションの利用）

分類	タイトル	参照先
解説	ロード対象のクラスとロード時に必要なクラスパス	18.1.1
	EJB 2.1 と Servlet 2.4 でのアノテーション参照抑止機能	18.1.2
実装	javax.annotation パッケージに含まれるアノテーションのサポート範囲	18.1.3
	javax.ejb パッケージに含まれるアノテーションのサポート範囲	18.1.4
設定	実行環境での設定（J2EE サーバ単位の設定）	18.1.5
	アノテーション参照抑止機能の設定変更	18.1.6
注意事項	アノテーションの利用時の注意事項	18.1.7

注　「運用」について，この節での説明はありません。

18.1.1　ロード対象のクラスとロード時に必要なクラスパス

アノテーション情報は，次のような操作を実行したときに読み込まれます。

- リソースの追加（cjaddapp）※
- インポート時（cjimportapp）
- インポート時（cjimportres）
- インポート時（cjimportlibjar）※
- 開始時（cjstartapp）
- リプレース時（cjreplaceapp）
- リロード時（cjreloadapp）

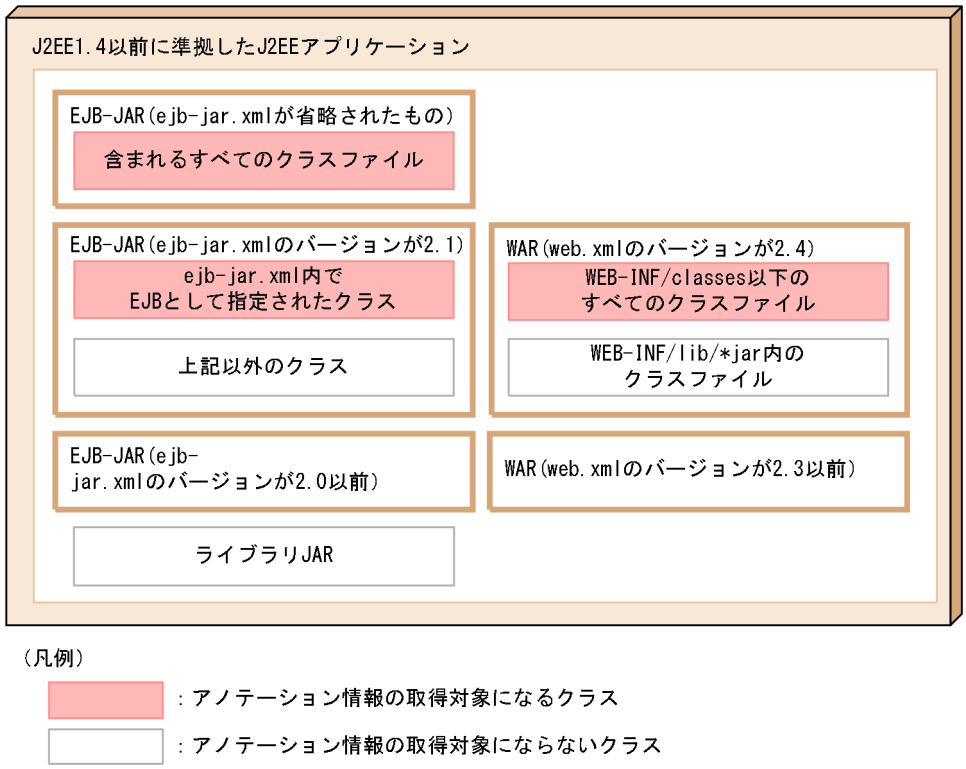
注※

対象となるのはライブラリ JAR のクラスです。

アノテーション情報を読み込むためには，まずクラスをロードする必要があります。

J2EE1.4 以前に準拠した J2EE アプリケーションでアノテーション情報を読み込むためにロードするクラスを次の図に示します。

図 18-1　アノテーション情報を読み込むためにロードするクラス



EJB-JAR（ejb-jar.xml のバージョンが 2.1）に含まれるクラスファイルまたは WAR（web.xml のバージョンが 2.4）に含まれるクラスファイルからアノテーション情報を読み込む場合のアノテーションは，アプリケーションサーバ独自の仕様です。また，アプリケーションサーバでは，アプリケーションサーバ独自の仕様によって，J2EE 1.4 の場合でも ejb-jar.xml の省略とアノテーションの使用ができます。

なお，J2EE 1.4 以前に準拠した J2EE アプリケーションの場合，EJB-JAR（ejb-jar.xml のバージョンが 2.1，または ejb-jar.xml が省略されたもの）や WAR（web.xml のバージョンが 2.4）からライブラリ JAR を使用することはできません。指定しても無視されます。

アノテーション情報を取得するためにクラスをロードするかどうかは，各モジュールのバージョン，および J2EE サーバで設定されているプロパティに依存します。

モジュールのバージョンが EJB-JAR（2.1）または WAR（2.4）の場合のアノテーション情報を取得するかどうかの条件を次の表に示します。

表 18-2　モジュールのバージョンが EJB-JAR（2.1）または WAR（2.4）の場合

モジュール		J2EE サーバのプロパティ ejbserver.deploy.applications.metadata_complete	
		false またはプロパティを省略時	true
EJB-JAR（2.1）		○	−
WAR（2.4）	WEB-INF/classes 以下	○	−
	WEB-INF/lib/JAR ファイル	−	−

（凡例）

○：アノテーション情報を取得する

－：アノテーション情報を取得しない

18.1.2　EJB 2.1 と Servlet 2.4 でのアノテーション参照抑止機能

アノテーション参照抑止機能は，J2EE アプリケーション中のアノテーションの参照（解析処理）を実施するかどうかを設定する機能です。アノテーション参照抑止機能を使用すると，アノテーションを使用しない場合に，アノテーションの解析処理を抑止することができます。

EJB 2.1 または Servlet 2.4 でのアノテーションの利用は，アプリケーションサーバで拡張された機能です。このため，アノテーションを利用している場合に，アノテーションの参照を抑止したいときは，アプリケーションサーバ独自のアノテーション参照抑止機能を使用します。

なお，EJB 2.1 または Servlet 2.4 でのアノテーションの参照抑止機能は，J2EE サーバ単位だけで設定できます。モジュール単位の参照抑止は設定できません。また，J2EE サーバ単位のアノテーションの参照抑止機能を有効にした場合，アノテーションを記述した EJB 2.1 または Servlet 2.4 のモジュールを含む J2EE アプリケーションは使用できなくなります。

18.1.3　javax.annotation パッケージに含まれるアノテーションのサポート範囲

javax.annotation パッケージのアノテーションの適用範囲を説明します。ここでは，WAR ファイル（Servlet 2.4 対応）および EJB-JAR ファイル（EJB2.1 対応）のコンポーネントごとに記述できるアノテーションを説明します。

(1)　EJB-JAR ファイル（EJB2.1 対応）

EJB-JAR ファイルに記述できるアノテーションの一覧を示します。

表 18–3　EJB-JAR ファイル（EJB2.1 対応）に記述できるアノテーション（javax.annotation パッケージ）

アノテーション名	Enterprise Bean				例外クラス	その他のクラス
	インタフェース	Session Bean	Entity Bean	Message-driven Bean		
@Resource※	－	○	－	×	－	－
@Resources※	－	○	－	×	－	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

(2) WAR ファイル（Servlet 2.4 対応）

WAR ファイルに記述できるアノテーションの一覧を示します。

表 18–4 WAR ファイル（Servlet 2.4 対応）に記述できるアノテーション（javax.annotation パッケージ）

アノテーション名	Servlet 仕様			JSP 仕様				その他のクラス
	サーブレット	サーブレットフィルタ	イベントリスナ	JSP ファイル	タグハンドラ		タグライブラリイベントリスナ	
					クラシックタグハンドラ	シンプルタグハンドラ		
@Resource※1	○	○	○	－	○	○※2	×	－
@Resources※1	○	○	○	－	○	○	×	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※1

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

注※2

Servlet2.4 仕様の Web アプリケーションではメソッドおよびフィールドに対する DI ができません。

18.1.4 javax.ejb パッケージに含まれるアノテーションのサポート範囲

javax.ejb パッケージのアノテーションの適用範囲を説明します。ここでは，WAR ファイル（Servlet 2.4 対応）および EJB-JAR ファイル（EJB2.1 対応）のコンポーネントごとに記述できるアノテーションを説明します。

(1) WAR ファイル（Servlet 2.4 対応）

WAR ファイルに記述できるアノテーションの一覧を示します。

表 18–5　WAR ファイル（Servlet 2.4 対応）に記述できるアノテーション（javax.ejb パッケージ）

アノテーション名	Servlet 仕様			JSP 仕様				その他のクラス
	サーブレット	サーブレットフィルタ	イベントリスナ	JSP ファイル	タグハンドラ		タグライブラリイベントリスナ	
					クラシックタグハンドラ	シンプルタグハンドラ		
@EJB※1	○	○	○	－	○	○※2	×	－
@EJBs※1	○	○	○	－	○	○	×	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※1

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

注※2

Servlet2.4 仕様の Web アプリケーションではメソッドおよびフィールドに対する DI ができません。

(2)　EJB-JAR ファイル（EJB2.1 対応）

EJB-JAR ファイルに記述できるアノテーションの一覧を示します。

表 18–6　EJB-JAR ファイル（EJB2.1 対応）に記述できるアノテーション（javax.ejb パッケージ）

アノテーション名	Enterprise Bean				例外クラス	その他のクラス
	インタフェース	Session Bean	Entity Bean	Message-driven Bean		
@EJB※	－	○	－	×	－	－
@EJBs※	－	○	－	×	－	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

18.1.5　実行環境での設定（J2EE サーバ単位の設定）

EJB 2.1 または Servlet 2.4 のモジュールに対するアノテーション参照抑止機能の定義は，J2EE サーバ単位に設定します。設定した内容は，その J2EE サーバにデプロイされているすべての J2EE アプリケーションに適用されます。

J2EE サーバ単位でアノテーションの参照を抑止する設定は，簡易構築定義ファイルを編集して行います。論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，次のパラメタで設定します。

キー名称	内容	省略値	導入されたバージョン
ejbserver.deploy.applications.metadata_complete	アノテーションを記述できるアプリケーションのうち，次のアプリケーションでアノテーション参照抑止機能を有効にするかどうかを指定します。 • EJB 2.1 • Servlet 2.4 true を指定した場合： 　アノテーションを参照しません。 false を指定した場合： 　アノテーションを参照します。	false	07-50

このパラメタは，usrconf.properties（J2EE サーバ用ユーザプロパティファイル）にも指定できます。

アノテーションの参照を抑止する設定をした場合，設定した J2EE サーバの起動時に，アノテーションの参照を抑止することを示すメッセージが出力されます。J2EE サーバ起動前または起動後にアノテーションの参照を抑止しているかどうかの設定を確認する場合は，簡易構築定義ファイルの<configuration>タグ内のパラメタ「ejbserver.deploy.applications.metadata_complete」に設定された値を参照してください。

注意事項

J2EE サーバで ejbserver.deploy.applications.metadata_complete に true を指定した場合，アノテーションを記述した EJB 2.1 または Servlet 2.4 のモジュールを含む J2EE アプリケーションは使用できなくなります。

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

18.1.6　アノテーション参照抑止機能の設定変更

アノテーション参照抑止機能の設定を変更する場合について説明します。

(1) モジュール単位の設定（DD の metadata-complete 属性の値）を変更する場合

DD の metadata-complete 属性を変更する場合は，metadata-complete 属性を変更したアプリケーションを作成して，J2EE サーバに再度インポートする必要があります。

metadata-complete 属性を変更した場合，リデプロイ機能（-replaceDD オプションを指定した cjreplaceapp コマンド）によってアプリケーションを入れ替えようとすると，エラーになり，入れ替え処理は中断されます。

(2) J2EE サーバ単位の設定（簡易構築定義ファイルの ejbserver.deploy.applications.metadata_complete の値）を変更する場合

アプリケーションインポート後に ejbserver.deploy.applications.metadata_complete の値を変更する場合について説明します。

true から false に変更した場合は，アプリケーションのアノテーションが参照されるようになります。特に対処は不要です。

false から true に変更した場合は，アノテーションが参照されなくなります。これに伴い，アプリケーションが正常に動作しなくなることがあります。アプリケーションが正常に動作しなくなった場合は，次の対処を実施してください。

アーカイブ形式のアプリケーションの場合

アノテーション参照抑止機能の設定を見直してください。または，J2EE アプリケーションの実装および DD の設定を見直して，アノテーションを参照しなくても正しく動作するように設定を変更してください。変更後，アプリケーションを再度インポートしてください。

展開ディレクトリ形式のアプリケーションの場合

アノテーション参照抑止機能の設定を見直してください。または，J2EE アプリケーションの実装および DD の設定を見直して，アノテーションを参照しなくても正しく動作するように設定を変更してください。変更後，アプリケーションを再開始してください。

18.1.7 アノテーションの利用時の注意事項

EJB 2.1，Servlet 2.4 のモジュールに定義されたアノテーションの情報は，DD に定義しても上書きはされません。

これらのモジュールに定義できるのは，アプリケーションサーバ独自のアノテーションです。DD によって上書きすることはできません。

19 クラスタコネクションプール機能を使用するための設定

この章では，クラスタコネクションプール機能について説明します。

この機能は互換機能のため推奨しません。Oracle RAC の機能による代替を検討してください。

19.1　クラスタコネクションプール機能

この節では，クラスタコネクションプール機能について説明します。

この節の構成を次の表に示します。

表 19-1　この節の構成（クラスタコネクションプール機能）

分類	タイトル	参照先
解説	Oracle RAC を使用した Oracle への接続	19.1.1
	クラスタコネクションプールの概要	19.1.2
	使用するリソースアダプタ	19.1.3
	クラスタコネクションプールの動作	19.1.4
	手動によるコネクションプールの停止・開始の流れ	19.1.5
設定	コネクションプールをクラスタ化するために必要な設定	19.1.6

注　「実装」および「運用」について，この機能固有の説明はありません。

クラスタコネクションプール機能は，データベースをクラスタ構成にしているシステムの場合に，最適な動作をするための機能です。Oracle RAC を使用した Oracle への接続の場合に使用できます。クラスタコネクションプール機能を使用することで，障害発生時やメンテナンス時の可用性の低下を防ぐことができます。クラスタコネクションプール機能の使用時に，データベースノードに障害が発生した場合，およびデータベースノードのメンテナンスを行う場合の動作について説明します。

- **データベースノードに障害が発生した場合**

  OS，ハード，ソフト障害など，コネクションが取得できない状況になった場合，障害が発生しているデータベースノードに接続しているコネクションプールを自動で一時停止できます（自動一時停止機能）。J2EE アプリケーションからリソースアダプタにコネクションの取得要求が出された場合でも，一時停止されたコネクションプールにはコネクションの取得要求が出されないため，TCP/IP のタイムアウトまで処理が中断することはありません。これによって，J2EE アプリケーションは，ほかの正常なデータベースノードに接続しているコネクションプールからコネクションを取得し，業務を継続できます。

  また，データベースノードの障害が回復した時に，自動でコネクションプールを再開できます（自動再開機能）。コネクションプールが再開されると，自動的に回復したデータベースノードから再びアクセスされるため，データベースノード回復時に cjclearpool コマンドを実行して，コネクションプールを削除する必要はありません。

- **データベースノードのメンテナンスを行う場合**

  データベースノードのメンテナンスを行う場合，コマンドを使用して任意のタイミングでメンバコネクションプールを一時停止できます（手動一時停止機能）。これによって，そのデータベースノードを切り離して，メンテナンスを行えます。

また，メンテナンスが終了して再開する時に，コマンドを使用して任意のタイミングでコネクションプールを再開できます（手動再開機能）。

なお，クラスタコネクションプール機能では，コネクション取得時に障害を検知した場合，正常なほかのメンバリソースアダプタからコネクションを取得します。その際，アプリケーションでエラーは発生しません。

この節では，Oracle RAC 機能を使用してクラスタ化した Oracle との接続方法，およびコネクションプールをクラスタ化する場合のコネクションプール（クラスタコネクションプール）の特徴，および機能について説明します。

19.1.1　Oracle RAC を使用した Oracle への接続

Oracle RAC を使用した Oracle への接続方法は，Oracle のバージョン，または負荷分散に使用する機能によって異なります。なお，接続できるトランザクションの種類はローカルトランザクションです。

Oracle のバージョン，負荷分散に使用する機能および使用する RAR ファイルの対応については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6.7　Oracle と接続する場合の前提条件と注意事項」を参照してください。

負荷分散に使用する機能ごとに，Oracle への接続方法について説明します。

(1)　アプリケーションサーバの機能を使用した接続

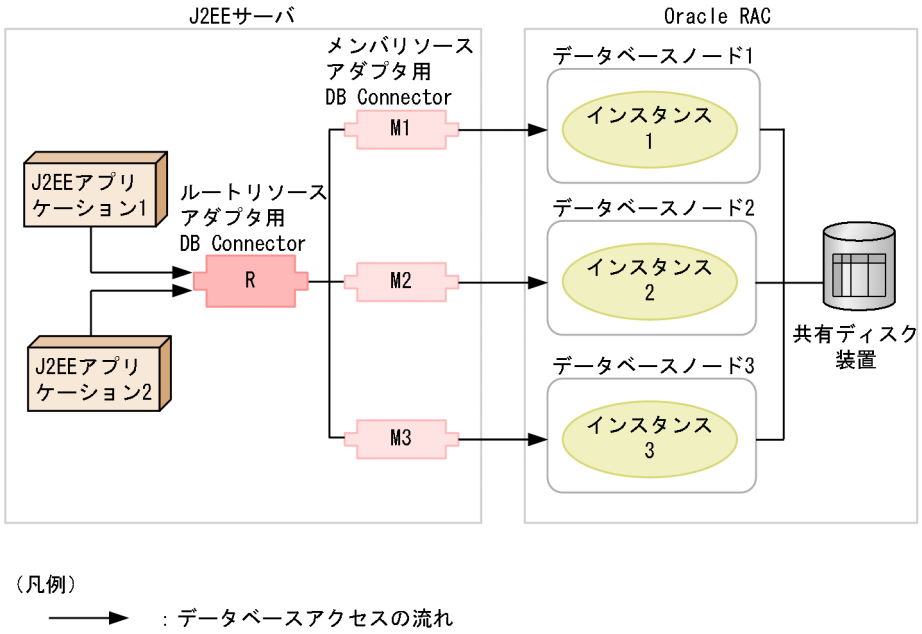
アプリケーションサーバのクラスタコネクションプール機能を使用して，Oracle RAC に接続します。アプリケーションサーバがデータベースアクセスの負荷を分散します。

クラスタコネクションプール機能の詳細については，19.1.2 以降の説明を参照してください。

(a)　負荷分散処理の流れと設定

クラスタコネクションプール機能を使用した場合の負荷分散処理の流れと設定を次の図に示します。

図 19-1　クラスタコネクションプール機能を使用した接続



この図では，3 台構成の Oracle RAC システムで，データベースノード 1 にはインスタンス 1，データベースノード 2 にはインスタンス 2，データベースノード 3 にはインスタンス 3 があります。データベースに接続するための設定を次に示します。

1. インスタンス 1，2，3 に対応したメンバリソースアダプタ用 DB Connector M1，M2，M3 を生成します。また，メンバリソースアダプタに振り分ける機能を持つルートリソースアダプタ用 DB Connector R も生成します。
2. J2EE アプリケーションはルートリソースアダプタ用 DB Connector R に関連づけます。

この設定によって，J2EE アプリケーション 1，2 からのデータベースアクセスはデータベースノード 1，2，3 に分散されます。

(b)　データベース障害発生時と回復時の動作

データベース障害が発生した場合，アプリケーションサーバが障害を検知します。障害が発生したデータベースに対応するメンバリソースアダプタが閉塞され，残っているインスタンスで処理が続行されます。

データベース障害が回復するとアプリケーションサーバは自動で閉塞を解除します。また，手動でも閉塞を解除できます。

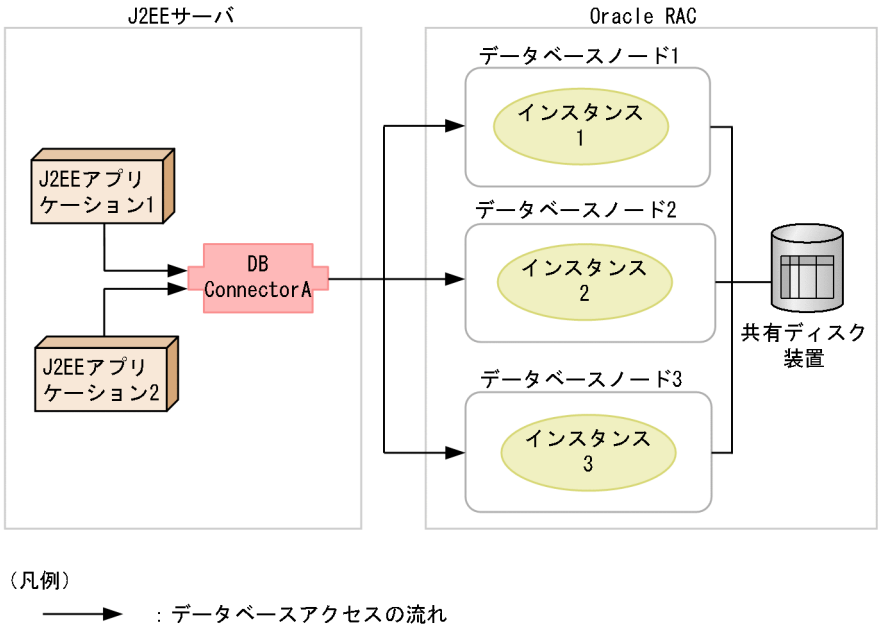
(2)　Oracle の機能を使用した接続

DB Connector から Oracle RAC に接続し，Oracle RAC の機能でデータベースアクセスの負荷を分散します。

(a)　負荷分散処理の流れと設定

Oracle RAC の機能を使用した場合の負荷分散処理の流れと設定を次の図に示します。

図 19-2　Oracle RAC の機能を使用した接続



この図では，3 台構成の Oracle RAC システムで，データベースノード 1 にはインスタンス 1，データベースノード 2 にはインスタンス 2，データベースノード 3 にはインスタンス 3 があります。データベースに接続するための設定について説明します。

1. それぞれのインスタンスに分散して接続する DB Connector A を生成します。
2. DB Connector A は，グローバルデータベース名またはサービス名を使用するように設定します。
3. J2EE アプリケーション 1，および 2 を DB Connector A に関連づけます。

この設定によって，J2EE アプリケーション 1，および 2 からのデータベースアクセスはデータベースノード 1，2，3 に分散されます。

(b)　データベース障害発生時と回復時の動作

データベース障害が発生した場合，Oracle RAC 機能によって障害が発生したインスタンスが切り離され，残っているインスタンスで処理が続行されます。

アプリケーションサーバのコネクションプールを使用している場合，データベース回復時には次のどちらかの操作を実行してください。コネクションプールがクリアされ，これ以降のアクセスが正常に分散されます。

- cjclearpool コマンドを実行する。
- J2EE サーバを再起動する。

19.1.2　クラスタコネクションプールの概要

クラスタ化されたコネクションプールのことをクラスタコネクションプールといいます。ここでは，クラスタコネクションプールの構成，およびクラスタコネクションプールで使用できる機能について説明します。

(1)　クラスタコネクションプールの構成

クラスタコネクションプールは，各データベースノードと接続するメンバリソースアダプタと，それら複数のメンバリソースアダプタを束ねるルートリソースアダプタで構成されています。ルートリソースアダプタとメンバリソースアダプタについて説明します。

- ルートリソースアダプタ

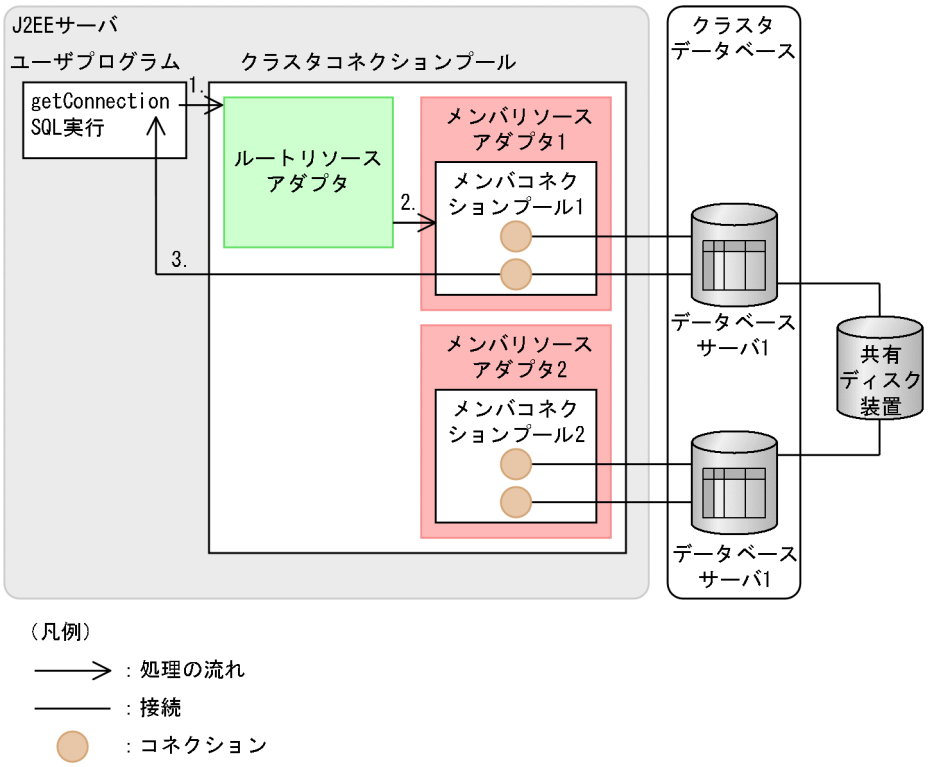
  クラスタ化したコネクションプール（クラスタコネクションプール）を使用するときに，J2EE アプリケーションからアクセスされるリソースアダプタです。ルートリソースアダプタは，メンバリソースアダプタを束ねる役割を持ちます。ルートリソースアダプタでは，ルートリソースアダプタに対する処理要求をメンバリソースアダプタに振り分けます。なお，ルートリソースアダプタは，コネクションプールを持ちません。

- メンバリソースアダプタ

  クラスタ化されている個々のデータベースノードに接続するリソースアダプタです。メンバリソースアダプタは，必ずルートリソースアダプタを経由してアクセスされます。メンバリソースアダプタのコネクションプールのことを，メンバコネクションプールといいます。

コネクションプールをクラスタ化しているときのコネクション取得時の処理の流れを次の図に示します。

図 19-3　コネクション取得時の処理の流れ



1. J2EE アプリケーションは，ルートリソースアダプタに対しコネクション取得要求をする。

2. ルートリソースアダプタは，メンバコネクションプールを一つ選択し，コネクション取得要求を出す。

3. メンバコネクションプールからコネクションが選択され，J2EE アプリケーションに返す。

(2) 前提条件

クラスタコネクションプール機能を利用できるデータベースは，Oracle RAC 機能を使用している場合だけです。使用できる JDBC ドライバは，Oracle JDBC Thin Driver です。

(3) データベース接続で使用できる J2EE コンポーネントおよび機能

クラスタコネクションプール機能を使用している場合に，データベース接続で使用できる J2EE コンポーネントおよび機能を次の表に示します。

表 19-2 データベース接続で使用できる J2EE コンポーネントおよび機能（クラスタコネクションプール機能）

項目		Oracle（クラスタコネクションプール機能を使用した場合）
J2EE コンポーネント	Servlet/JSP	○
	Stateless Session Bean	○
	Stateful Session Bean	○
	Singleton Session Bean	○
	Entity Bean（BMP）	○
	Entity Bean（CMP1.1）	×
	Entity Bean（CMP2.0）	×
	Message-driven Bean	×
使用できる機能	コネクションプーリング	○
	コネクションプールのウォーミングアップ	○
	コネクションプールの情報表示（cjlistpool コマンド）	○
	コネクションプールのクリア	○
	リソースへの接続テスト	○
	コネクションの障害検知	○
	ステートメントプーリング	○
	ステートメントキャンセル	○※
	ステートメント setQueryTimeout メソッド	○※

項目		Oracle（クラスタコネクションプール機能を使用した場合）
	コネクション ID の PRF トレース出力	○
	障害調査用 SQL の出力	○

（凡例）○：使用できる　×：使用できない

注※　Oracle に接続する場合の注意事項があります。注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6.7　Oracle と接続する場合の前提条件と注意事項」を参照してください。

注意事項

リソースアダプタを使用する場合，J2EE アプリケーションからリソースアダプタへのリファレンスを解決しておく必要があります。リソースアダプタを使用している J2EE アプリケーションをカスタマイズするときに，J2EE アプリケーションからリソースアダプタへのリファレンスを解決しておいてください。

(4)　使用できるリソース接続とトランザクション管理の機能

ルートリソースアダプタ，およびメンバリソースアダプタで使用できる機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.3.4　リソースアダプタの機能」を参照してください。

19.1.3　使用するリソースアダプタ

ルートリソースアダプタおよびメンバリソースアダプタで実行できるコマンド，設定の概要，および注意事項について説明します。

(1)　実行できるコマンド

ルートリソースアダプタおよびメンバリソースアダプタごとの，コマンドの実行可否を次の表に示します。なお，これ以外のコマンドについてはどちらのリソースアダプタでも実行できます。

表 19-3　ルートリソースアダプタおよびメンバリソースアダプタのコマンドの実行可否

コマンド	リソースアダプタの種類	
	ルートリソースアダプタ	メンバリソースアダプタ
cjstartrar	○	○
cjstoprar	○	○
cjtestres	○	○
cjlistrar	○	○

コマンド	リソースアダプタの種類	
	ルートリソースアダプタ	メンバリソースアダプタ
cjclearpool	×	○
cjlistpool	×	○
cjsuspendpool	×	○
cjresumepool	×	○

（凡例） ○：実行できる　　×：実行できない

(2) DB Connector の設定

ルートリソースアダプタおよびメンバリソースアダプタの DB Connector で設定する項目を次の表に示します。

表 19-4 ルートリソースアダプタおよびメンバリソースアダプタの DB Connector で設定する項目

設定項目	リソースアダプタの種類	
	ルートリソースアダプタ	メンバリソースアダプタ
トランザクションサポートレベル	×	○※1
ログ取得の可否	○	○
データベース接続情報	−	○
DB Connector 固有の設定（ステートメントプールなど）	−	○
セキュリティ情報（ユーザ名，パスワード）	×	○※2
コネクションプールサイズ	×	○
クラスタコネクションプール固有の設定	○	−

（凡例）○：設定が必要　×：設定は不要　−：設定項目がない

注※1　一つのクラスタコネクションプールを構成するメンバリソースアダプタのトランザクションサポートレベルは，すべて同じにする必要があります。

注※2　一つのクラスタコネクションプールを構成するメンバリソースアダプタのユーザ名は，すべて同じにする必要があります。

DB Connector の設定の詳細については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2　データベースと接続するための設定」，または「19.6　Connector 属性ファイル」を参照してください。

(3) ルートリソースアダプタについての注意事項

- cjclearpool コマンドは実行できません。実行した場合は，メッセージが出力され，コマンドが異常終了します。クラスタコネクションプール内のコネクションを cjclearpool コマンドでクリアする場合は，メンバリソースアダプタに対してコマンドを実行してください。
- cjlistpool コマンドは実行できません。実行した場合は，メッセージが出力され，コマンドが異常終了します。クラスタコネクションプール内のコネクションプールの情報を表示する場合は，cjlistpool コマンドの-resname にメンバリソースアダプタの表示名を指定するか，-resall を指定して実行してください。
- 一つのクラスタコネクションプール内で，コンテナ管理でのサインオンとコンポーネント管理でのサインオンを混在して使用できません。コンポーネント管理でのサインオンを使用する場合は，ルートリソースアダプタに属するすべてのメンバリソースアダプタのユーザ名を空白にしてください。
- ルートリソースアダプタは，ルートリソースアダプタに属するすべてのメンバリソースアダプタが開始状態の場合に開始できます。これ以外の場合に，ルートリソースアダプタを開始すると，コンソールまたは画面にエラーメッセージが出力され，リソースアダプタの開始に失敗します。

(4) メンバリソースアダプタについての注意事項

- メンバリソースアダプタでは，次の機能が前提となります。これらの機能は，デフォルトで有効となり，無効にすることはできません。
  - コネクションプーリング

    コネクションプールの最大値に 0 より大きい値を設定してください。0 以下の値を設定した場合，コネクションプールの最大値と最小値にデフォルト値が指定されたものとして動作します。
  - コネクション取得時のコネクションの障害検知

    コネクション取得時のコネクションの障害検知およびコネクション障害検知のタイムアウトは，設定値に関係なく有効となります。
  - コネクション枯渇時のコネクション取得待ち

    コネクション枯渇時のコネクション取得待ちは，設定値に関係なく有効となります。
  - loginTimeout

    loginTimeout のプロパティに 0 よりも大きい値を設定してください。0 以下の値を設定した場合はデフォルト値が指定されたものとして動作します。
- メンバリソースアダプタでは，次の機能を使用できません。これらの機能は，デフォルトで無効となり，有効にすることはできません。
  - コネクションの取得リトライ
  - J2EE リソースのユーザ指定名前空間
- メンバコネクションプール内のコネクションは，すべて同じデータベースノードに接続している必要があるため，データベースで接続先を変更する機能は使用しないでください。次にその機能の例を挙げます。なお，各機能を無効にする設定については，オラクルのマニュアルを参照してください。

- クライアント・ロード・バランシング機能
- 接続時フェイルオーバー機能
- データベースサービス
- リスナによる負荷分散機能

- メンバリソースアダプタにアクセスするときは，必ずルートリソースアダプタを経由します。そのため，次の個所には設定できません。
  - J2EE アプリケーションのリソースリファレンス
  - CMP Entity Bean のマッピング定義
- メンバリソースアダプタは，所属するルートリソースアダプタが停止状態の場合に停止できます。開始状態のときに，メンバリソースアダプタを停止すると，コンソールまたは画面にエラーメッセージが出力され，リソースアダプタの停止に失敗します。
- メンバコネクションプールの閉塞および一時停止時，コネクションを破棄するために，コネクションプールから未使用コネクションを取り除きます。この際，コネクションの破棄が完了していない時点でコネクションプールが再開されると，コネクションプール内のコネクションとコネクションプールから取り除いた未使用コネクションの総数が，コネクションプールのコネクション数の最大値を一時的に超える場合があります。

19.1.4　クラスタコネクションプールの動作

コネクションプールをクラスタ化した場合に使用できる機能と動作について説明します。クラスタコネクションプールでは次の機能を実行できます。

- コネクションプールの一時停止
- コネクションプールの再開

また，コネクションプールの状態，およびコネクションの取得要求を受けたときの，コネクションプールの選択方式についても説明します。

(1)　コネクションプールの一時停止

メンバコネクションプールを閉塞および一時停止できます。一時停止を実行すると，メンバコネクションプールが閉塞し一時停止します。

J2EE アプリケーションがルートリソースアダプタにコネクション取得要求をした場合，閉塞または一時停止したメンバコネクションプールにはコネクション取得要求は出されません。

次の場合にメンバコネクションプールを一時停止してください。

- データベースノードに障害が発生した場合
- データベースノードのメンテナンスをする場合

なお，一時停止の方法には次の 2 とおりがあります。

- 自動一時停止
- 手動一時停止

> **参考**
>
> 一時停止処理では，J2EE アプリケーションで使用し終わったコネクションを破棄し，コネクションプール内のすべてのコネクションを破棄してから，コネクションプールを一時停止します。ネットワーク障害やデータベースノード障害が発生した場合，ネットワークのタイムアウトまで待ってからコネクションを破棄するため，閉塞してから一時停止するまでに時間が掛かることがあります。

(a)　自動一時停止

データベースノードの障害時にメンバコネクションプールを自動で一時停止できます。障害を検知すると，メンバコネクションプールは自動的に閉塞状態になり，そのあと一時停止します。

コネクションの取得時に次の現象が発生した場合，データベースノードに障害が発生したと判断し，メンバコネクションプールが自動的に一時停止します。

- コネクション取得時のコネクションの障害検知がタイムアウトした場合
- 物理コネクションの取得に失敗した場合
- 物理コネクションの取得がタイムアウトした場合
- コネクション管理スレッドが枯渇した場合

この機能は，デフォルトで有効となっています。有効/無効の設定は，ルートリソースアダプタのプロパティとして設定します。リソースアダプタの設定については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「5.4　リソースアダプタのプロパティ定義」を参照してください。

(b)　手動一時停止

データベースノードのメンテナンスをする場合などに，cjsuspendpool コマンドを実行することで，コネクションプールを手動で一時停止できます。手動一時停止は，メンバコネクションプールが開始状態，開始予約状態，自動一時停止状態，および自動一時停止予約状態の場合に実行できます。メンバコネクションプールが自動一時停止状態の場合に cjsuspendpool コマンドを実行すると，手動一時停止状態になります。これによって，自動一時停止後に自動再開させない運用ができます。手動での一時停止手順については，「19.1.5(2)　コネクションプールの一時停止」を参照してください。コマンドの詳細については，「cjsuspendpool（メンバコネクションプールの一時停止）」を参照してください。

> **注意事項**
>
> cjsuspendpool コマンドで手動一時停止したコネクションプールは，自動で再開されません。手動で再開してください。

(2) コネクションプールの再開

一時停止したメンバコネクションプールを再開できます。J2EE アプリケーションからルートリソースアダプタにコネクション取得要求をした時に，再開したメンバコネクションプールには再びコネクションの取得要求が出されるようになります。

なお，コネクションプールを再開するには，未使用のコネクション管理スレッドの個数が，コネクションプールのコネクション数の最大数以上必要です。

次の場合にコネクションプールを再開してください。

- データベースノードの障害が回復した場合
- データベースノードのメンテナンスが終了した場合

再開の方法には次の 2 とおりがあります。

- 自動再開
- 手動再開

> **参考**
>
> - コネクションプールが停止されている場合に，未使用のコネクション管理スレッドの個数がコネクションプールのコネクション数の最大数になると，メッセージが出力されます。cjresumepool コマンドの実行に失敗した場合は，出力されたメッセージを確認してから再度コマンドを実行してください。
> - コネクションプールのウォーミングアップ機能が有効な場合には，コネクションプールの設定で定義した最小値までコネクションをプールしてから，メンバコネクションプールを開始します。コネクションをプーリングしているときにコネクションの生成に失敗した場合は，メンバコネクションプールは一時停止状態に戻ります。また，コネクションプールのウォーミングアップ機能が無効な場合は，すぐに開始状態になります。

(a) 自動再開

自動一時停止したメンバコネクションプールを自動で再開できます。

自動一時停止したメンバコネクションプールでは，データベースノードの状態をチェックするために，一定間隔で物理コネクションの取得要求を出します。このとき，コネクションの取得に成功すると，データベースノードが回復したと判断し，自動的に再開処理が行われます。また，自動一時停止したメンバコネクションプールを自動再開させない運用にするには，自動一時停止後に手動一時停止してください。再開時には，手動再開してください。

なお，ルートリソースアダプタが停止状態の場合，自動再開処理は行われません。ただし，メンバコネクションプールが自動再開中状態の場合にルートリソースアダプタを停止したときは，実行中の自動再開処理が継続されます。

この機能は，デフォルトで有効となっています。有効/無効の切り替えや，データベースノードの状態をチェックする間隔の設定は，ルートリソースアダプタのプロパティとして設定します。リソースアダプタの設定については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「5.4 リソースアダプタのプロパティ定義」を参照してください。

(b) 手動再開

自動一時停止または手動一時停止したコネクションプールを手動で再開できます。手動再開するには，cjresumepool コマンドを使用します。手動再開は，コネクションプールが次の状態の場合に実行できます。

- 自動一時停止状態
- 手動一時停止状態
- 自動一時停止予約状態
- 手動一時停止予約状態

手動での手動再開手順については，「19.1.5(3) コネクションプールの再開」を参照してください。コマンドの詳細については，「cjresumepool（メンバコネクションプールの再開始）」を参照してください。

(3) コネクションプールの状態

コネクションプールの状態は，メンバコネクションプールの場合だけ存在するものです。なお，コネクションプールの状態は，J2EE サーバやリソースアダプタの再起動後も維持されます。

メンバコネクションプールの状態は次の方法で確認できます。手動一時停止または手動再開をする場合は，コマンドを実行する前にコネクションプールの状態を確認してください。

- cjlistrar コマンド

  cjlistrar コマンドは，デプロイされているすべてのリソースアダプタについて，リソースアダプタ名とリソースアダプタの状態を標準出力に出力します。-clusterpool を指定した場合には，メンバコネクションプールの状態も表示できます。

  コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistrar（リソースアダプタの一覧表示）」を参照してください。

- cjlistpool コマンド

  cjlistpool コマンドは，コネクションプールの情報を表示するコマンドです。メンバリソースアダプタの場合は，メンバコネクションプールの状態も表示できます。メンバリソースアダプタのコネクション情報の表示例については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.4 コネクションプールの情報表示」を参照してください。

  コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistpool（コネクションプールの一覧表示）」を参照してください。

ここでは，メンバコネクションプールの状態について説明します。

メンバコネクションプールの状態は，J2EE サーバやメンバリソースアダプタを再起動しても維持されます。コネクションプールの状態を確認する方法については，「19.1.5(1)　コネクションプールの状態の確認」を参照してください。

なお，メンバリソースアダプタ以外のコネクションプールには状態はありません。

(a)　コネクションプールの状態遷移

メンバコネクションプールの状態遷移を次の図に示します。

図 19-4　メンバコネクションプールの状態遷移（手動で一時停止を実行する場合）

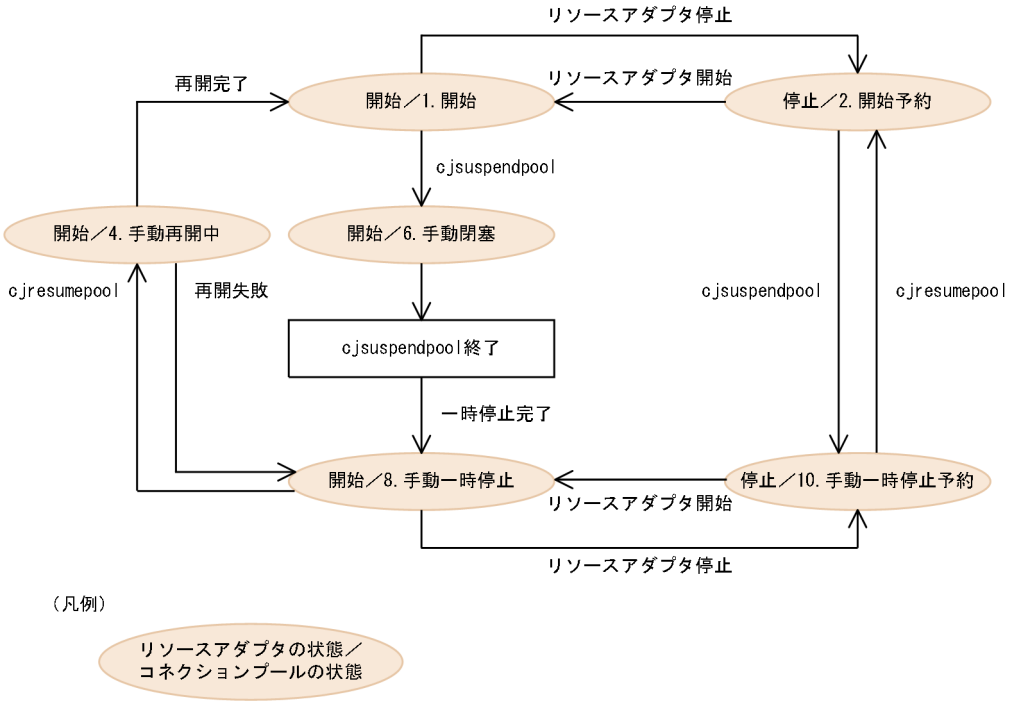
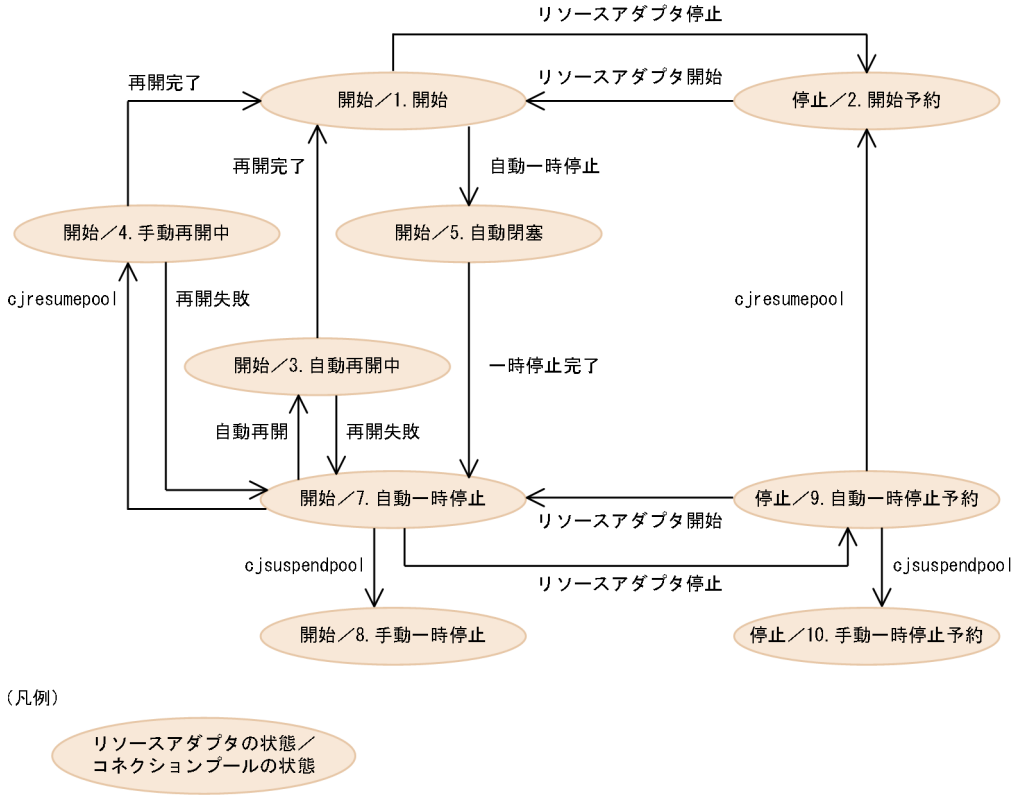


図 19-5　メンバコネクションプールの状態遷移（自動で一時停止が実行される場合）



なお，J2EE サーバの開始時にコネクションプールが再開中状態，または閉塞状態の場合，コネクションプールは一時停止状態に遷移します。

それぞれの状態の詳細について次の表に示します。

表 19-5　メンバコネクションプールの状態

番号	状態	メッセージに表示される状態	説明
1	開始状態	running	コネクションプールが処理を受け付けている状態です。ルートリソースアダプタへのコネクション取得要求時には，開始状態のコネクションプールだけに処理が行われます。
2	開始予約状態	runningReserved	コネクションプールが開始状態のときにリソースアダプタが停止された状態です。開始予約状態のコネクションプールは，リソースアダプタの開始時に開始状態になります。リソースアダプタをデプロイした直後は，この状態になります。
3	自動再開中状態	resumingAutomatically	自動再開機能によって，コネクションプールが再開処理を行っ

番号	状態	メッセージに表示される状態	説明
			ている状態です。再開処理が成功すると開始状態になります。再開処理が失敗すると，自動一時停止状態に戻ります。自動再開中状態のコネクションプールには，ルートリソースアダプタへのコネクション取得要求時に処理が出されません。
4	手動再開中状態	• resumingManuallyFromSuspendedAutomatically • resumingManuallyFromSuspendedManually • resumingManually	自動一時停止状態，または手動一時停止状態のときにcjresumepool コマンドを実行して，コネクションプールが再開処理を行っている状態です。再開処理が成功すると，開始状態になります。再開処理が失敗すると，コマンド実行前の状態に戻ります。手動再開中状態のコネクションプールには，ルートリソースアダプタへのコネクション取得要求時に処理が出されません。
5	自動閉塞状態	blockedAutomatically	自動一時停止機能によって，コネクションプールが閉塞されて，一時停止処理を行っている状態です。一時停止処理が完了すると，自動一時停止状態になります。自動閉塞状態のコネクションプールには，ルートリソースアダプタへのコネクション取得要求時に処理が出されません。
6	手動閉塞状態	blockedManually	cjsuspendpool コマンドによって，コネクションプールが閉塞され，一時停止処理を行っている状態です。一時停止処理が完了すると，手動一時停止状態になります。手動閉塞状態のコネクションプールには，ルートリソースアダプタへのコネクション取得要求時に処理が出されません。
7	自動一時停止状態	suspendedAutomatically	自動一時停止機能によって，コネクションプールが一時停止された状態です。コネクションプール内にコネクションは存在しません。自動一時停止状態の

番号	状態	メッセージに表示される状態	説明
			コネクションプールには，ルートリソースアダプタへのコネクション取得要求時に処理が出されません。
8	手動一時停止状態	suspendedManually	cjsuspendpool コマンドによって，コネクションプールが一時停止された状態です。コネクションプール内にコネクションは存在しません。手動一時停止状態のコネクションプールには，ルートリソースアダプタへのコネクション取得要求時に処理が出されません。
9	自動一時停止予約状態	suspendedAutomaticallyReserved	コネクションプールが自動一時停止状態のときにリソースアダプタを停止した状態です。自動一時停止予約状態のコネクションプールは，リソースアダプタの開始時に自動一時停止状態になります。なお，このとき，コネクションプールのウォーミングアップ機能は無効になります。
10	手動一時停止予約状態	suspendedManuallyReserved	コネクションプールが手動一時停止状態のときにリソースアダプタを停止した状態です。手動一時停止予約状態のコネクションプールは，リソースアダプタの開始時に手動一時停止状態になります。なお，このとき，コネクションプールのウォーミングアップ機能は無効になります。

注　番号は，図 19-4 および図 19-5 中の番号を示します。

(b)　コネクションプールの状態によるコマンド実行の可否

コネクションプールの状態によって，実行できるコマンドと実行できないコマンドがあります。コネクションプールの状態ごとに，各コマンドの実行の可否を次に示します。

表 19-6　コネクションプールの状態によるコマンド実行の可否

コマンド	コネクションプールの状態							
	開始	開始予約	自動再開中/手動再開中	自動閉塞/手動閉塞	自動一時停止	手動一時停止	自動一時停止予約	手動一時停止予約
cjstartrar	×	○	×	×	×	×	○	○
cjstoprar	○	×	△※1	△※2	○	○	×	×
cjclearpool	○	×	×	×	×	×	×	×
cjlistrar	○	○	○	○	○	○	○	○
cjsuspendpool	○	○	×	×	○	×	○	×
cjresumepool	×	×	×	×	○	○	○	○
cjstopsv(通常停止時)	○	○	△※1	△※2	○	○	○	○
cjstopsv(強制停止時)	○	○	○	○	○	○	○	○

（凡例）　○：実行できる　△：制限あり　×：実行できない

注※1　コマンドは受け付けられますが，開始または一時停止状態になってから処理が実行されます。

注※2　コマンドは受け付けられますが，一時停止状態になってから処理が実行されます。

(4)　コネクションプールの選択方式

J2EE アプリケーションがルートリソースアダプタにコネクションの取得要求を出した時に，メンバコネクションプールが一つ選択されます。このときメンバコネクションプールが選択される方式は，ラウンドロビン方式です。

選択対象のコネクションプールは，開始状態のメンバコネクションプールです。コネクションに空きがあるメンバコネクションプールが優先的に選択されます。

ただし，コネクション枯渇状態のメンバコネクションプールしかない場合は，コネクション取得要求が待ち状態になります。さらに，コネクション取得待ちのタイムアウトが発生すると，コネクションの取得に失敗します。また，コネクション取得要求が待ち状態の場合にコネクションプールが閉塞されたときは，コネクション取得要求が再開され，次の優先度のメンバコネクションプールからコネクションの取得を試みます。すべてのメンバコネクションプールからコネクションが取得できない場合には，コネクションの取得に失敗します。

なお，開始状態のメンバコネクションプールがない場合にコネクション取得要求があったときには，コネクションの取得に失敗します。

また，各メンバコネクションプールの最大サイズを設定する場合，次の指針に従って設定します。

メンバコネクションプールの最大サイズ（数）＝システムで許容される最大同時接続数÷データベースノードの数

19.1.5　手動によるコネクションプールの停止・開始の流れ

ここでは，クラスタコネクションプールの流れについて説明します。データベースに発生した障害に対応する場合や，データベースをメンテナンスする場合，一部のコネクションプールを手動で停止，再開始することで，システム全体を止めることなくデータベースをメンテナンスできます。一部のコネクションプールを手動で停止，再開始して，データベースをメンテナンスする作業は次の流れで行います。

1. コネクションプールの状態を確認する（(1)参照）
   サーバ管理コマンドを使用して実行します。

2. コネクションプールを一時停止する（(2)参照）
   サーバ管理コマンドを使用して実行します。

3. コネクションプールを再開する（(3)参照）
   サーバ管理コマンドを使用して実行します。

4. コネクションプールの状態を確認する（(1)参照）
   サーバ管理コマンドを使用して実行します。

(1)　コネクションプールの状態の確認

クラスタ構成のデータベースをメンテナンスする前，およびメンテナンスしたあとに，コネクションプールの状態を確認します。

コネクションプールの状態の詳細については「19.1.4(3)　コネクションプールの状態」を参照してください。

コネクションプールの状態の確認は，cjlistrar コマンドで実行できます。

cjlistrar コマンドの実行形式と実行例を次に示します。

実行形式

```
cjlistrar <サーバ名> -clusterpool
```

実行例

```
cjlistrar MyServer -clusterpool
```

(2)　コネクションプールの一時停止

ここでは，コネクションプールを一時停止する手順について説明します。

コネクションプールの一時停止は，cjsuspendpool コマンドで実行できます。cjsuspendpool コマンドを実行すると，コネクションプールが閉塞されて一時停止状態になり，コネクション取得要求を受け付けなくなります。

cjsuspendpool コマンドは，コネクションプールが次の状態の場合に実行できます。

- 開始
- 開始予約
- 自動一時停止
- 自動一時停止予約

コネクションプールの状態を確認する方法については，「19.1.5(1)　コネクションプールの状態の確認」を参照してください。

cjsuspendpool コマンドの実行形式と実行例を次に示します。

実行形式

```
cjsuspendpool ＜サーバ名＞ -resname ＜一時停止対象となるメンバリソースアダプタの表示名＞
```

実行例

```
cjsuspendpool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

(3)　コネクションプールの再開

ここでは，コネクションプールを再開する手順について説明します。

コネクションプールの再開は，cjresumepool コマンドで実行できます。cjresumepool コマンドを実行すると，コネクションプールは手動再開中状態になり，再開処理が実行されます。再開処理が完了すると，コネクションプールは開始状態になり，コネクション取得要求を受け付けるようになります。

cjresumepool コマンドは，コネクションプールが次の場合に実行できます。

- 使用されていないコネクション管理スレッドが，コネクションプールのコネクションの最大値以上ある
- コネクションプールが，手動一時停止，手動一時停止予約，自動一時停止，または自動一時停止予約の状態である

コネクションプールの状態を確認する方法については，「19.1.5(1)　コネクションプールの状態の確認」を参照してください。

cjresumepool コマンドの実行形式と実行例を次に示します。

実行形式

```
cjresumepool ＜サーバ名＞ -resname ＜再開対象となるメンバリソースアダプタの表示名＞
```

実行例

```
cjresumepool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

なお，cjresumepool コマンドを実行したあと，コネクションプールの状態を確認して，再開処理が正しく実行されたかどうかを確認してください。コネクションプールの状態を確認する方法については，「19.1.5(1)　コネクションプールの状態の確認」を参照してください。

19.1.6　コネクションプールをクラスタ化するために必要な設定

コネクションプールをクラスタ化するためには，DB Connector のプロパティ設定が必要になります。DB Connector のプロパティ設定については，「19.3　データベースと接続するための設定（クラスタコネクションプールの場合）」を参照してください。

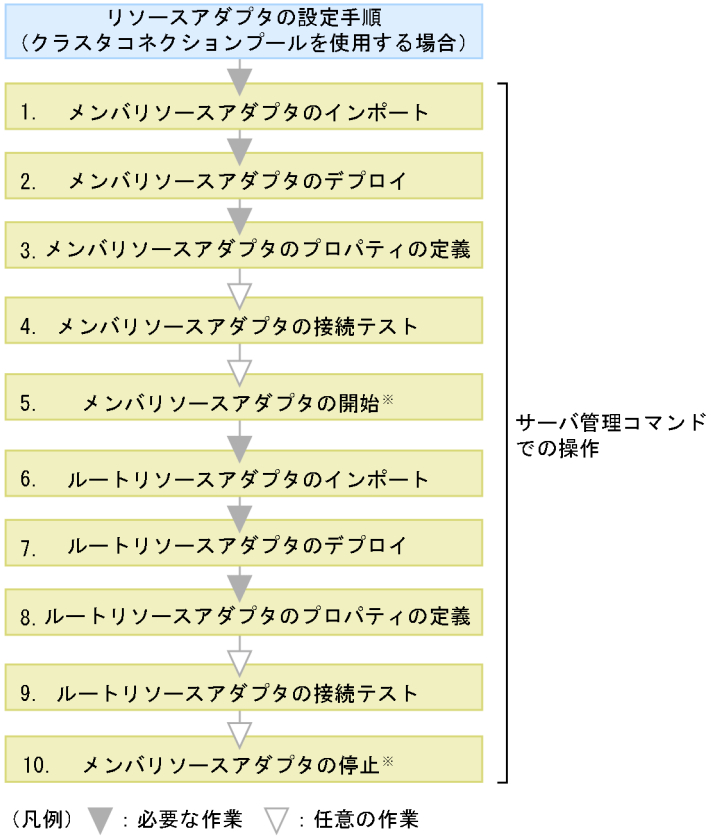
19.2　リソース接続

この節では，クラスタコネクションプール機能のリソースアダプタの設定の流れ（クラスタコネクションプールを使用する場合）について説明します。

19.2.1　リソースアダプタの設定の流れ（クラスタコネクションプールを使用する場合）

コネクションプールをクラスタ化している場合の，データベースに接続するときのリソースアダプタの設定の流れを次の図に示します。

図 19-6　クラスタコネクションプールでのリソースアダプタの設定の流れ



注※　ルートリソースアダプタの接続テストを実施する場合に必要な作業です。

図中の 1.～10.について説明します。

1. サーバ管理コマンドを使用してメンバリソースアダプタをインポートします。

   cjimportres コマンドを使用して，メンバリソースアダプタをインポートします。

   インポートするリソースアダプタについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.3.2　リソースアダプタの種類」を参照してください。

2. サーバ管理コマンドを使用してメンバリソースアダプタをデプロイします。

cjdeployrar コマンドを使用して，メンバリソースアダプタをデプロイします。

3. サーバ管理コマンドを使用してメンバリソースアダプタのプロパティを定義します。

cjgetrarprop コマンドで Connector 属性ファイルを取得し，ファイル編集後に，cjsetrarprop コマンドで編集内容を反映させます。

メンバリソースアダプタとルートリソースアダプタのプロパティ定義では設定できる項目が異なります。メンバリソースアダプタおよびルートリソースアダプタで設定できるプロパティ定義の項目については，「19.1　クラスタコネクションプール機能」を参照してください。

使用する機能ごとに設定するリソースアダプタのプロパティについては，それぞれ次の個所を参照してください。

- リソース接続とトランザクション管理機能を使用するための設定

  マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.4.12　実行環境での設定」

- パフォーマンスチューニングのための機能

  マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.10 実行環境での設定」

- フォールトトレランスのための機能

  マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.13 実行環境での設定」

- J2EE リソースの別名の設定

  マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.6.6　J2EE リソースの別名の設定」

4. サーバ管理コマンドを使用してメンバリソースアダプタの接続テストを実施します。

cjtestres コマンドを使用して，メンバリソースアダプタの接続テストを実施します。

メンバリソースアダプタの接続テストでの検証内容については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.17　リソースへの接続テスト」を参照してください。

また，1.～4.までの流れを，メンバリソースアダプタの数だけ繰り返します。

5. サーバ管理コマンドを使用してメンバリソースアダプタを開始します。

ルートリソースアダプタの接続テストを実施する場合には，あらかじめメンバリソースアダプタを開始しておいてください。cjstartrar コマンドを使用して，メンバリソースアダプタを開始します。

6. サーバ管理コマンドを使用してルートリソースアダプタをインポートします。

cjimportres コマンドを使用して，ルートリソースアダプタをインポートします。

インポートするリソースアダプタについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.3.2　リソースアダプタの種類」を参照してください。

7. サーバ管理コマンドを使用してルートリソースアダプタをデプロイします。

cjdeployrar コマンドを使用してルートリソースアダプタをデプロイします。

8. サーバ管理コマンドを使用してルートリソースアダプタのプロパティを定義します。

cjgetrarprop コマンドで Connector 属性ファイルを取得し，ファイル編集後に，cjsetrarprop コマンドで編集内容を反映させます。

9. サーバ管理コマンドを使用してルートリソースアダプタの接続テストを実施します。

cjtestres コマンドを使用して，ルートリソースアダプタの接続テストを実施します。

ルートリソースアダプタの接続テストでの検証内容については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.17　リソースへの接続テスト」を参照してください。

10. サーバ管理コマンドを使用してメンバリソースアダプタを停止します。

ルートリソースアダプタの接続テストを実施した場合には，メンバリソースアダプタを停止してください。cjstoprar コマンドを使用して，メンバリソースアダプタを停止します。

サーバ管理コマンドでの操作については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「3.　サーバ管理コマンドの基本操作」を参照してください。また，コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.4　J2EE サーバで使用するリソース操作コマンド」を参照してください。属性ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.　リソースの設定で使用する属性ファイル」を参照してください。

注意事項

コネクションプールをクラスタ化する場合，J2EE アプリケーションからルートリソースアダプタへのリファレンスを解決しておく必要があります。ルートリソースアダプタを使用している J2EE アプリケーションのプロパティを定義するときに，J2EE アプリケーションからルートリソースアダプタへのリファレンスを解決しておいてください。

19.3　データベースと接続するための設定（クラスタコネクションプールの場合）

DB Connector で，Oracle RAC のクラスタ化されたデータベースに接続する場合，コネクションプールをクラスタ化して使うことができます。クラスタコネクションプールの詳細については，「19.1　クラスタコネクションプール機能」を参照してください。

この節では，DB Connector のコネクションプールをクラスタ化するための設定について説明します。

19.3.1　クラスタコネクションプールの概要

クラスタ化されたコネクションプールをクラスタコネクションプールといいます。ルートリソースアダプタとメンバリソースアダプタで構成されます。メンバリソースアダプタのコネクションプールをメンバコネクションプールといいます。

DB Connector のクラスタコネクションプールを使用するための作業と状態の制御方法を次に示します。

(1)　クラスタコネクションプールの設定

クラスタコネクションプールを使用するためには，クラスタコネクションプールを使用できるリソースアダプタの設定が必要です。

クラスタコネクションプール用の DB Connector の設定は，次の手順で実施します。

なお，手順 1，2 は，必要なメンバリソースアダプタの数だけ繰り返してください。

1. メンバリソースアダプタ用 DB Connector を設定します。
   メンバリソースアダプタ用 DB Connector を，次の手順で設定します。
   - メンバリソースアダプタ用の DB Connector の RAR ファイルをインポートします。
   - プロパティを定義します。
   - DB Connector をデプロイします。
   - 接続を確認します。
     正しく接続できるかどうかは，接続テストで確認できます。
2. メンバリソースアダプタ用 DB Connector を開始します。
3. ルートリソースアダプタ用 DB Connector を設定します。
   ルートリソースアダプタ用 DB Connector を，次の手順で設定します。
   - ルートリソースアダプタ用の DB Connector の RAR ファイルをインポートします。
   - プロパティを定義します。

- DB Connector をデプロイします。
- 接続を確認します。
  正しく接続できるかどうかは，接続テストで確認できます。

4. ルートリソースアダプタ用 DB Connector を開始します。

ルートリソースアダプタは，J2EE アプリケーションから直接アクセスされるので，デプロイしたルートリソースアダプタは，J2EE アプリケーションのプロパティ設定でリファレンスを解決する必要があります。詳細については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「9.3.3 リソースアダプタのリファレンス定義」を参照してください。

参考

DB Connector のプロパティを新規に設定する場合，Component Container が提供しているテンプレートファイルが利用できます。

Connector 属性ファイルのテンプレートファイルは，次に示すディレクトリに格納されています。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥templates¥
- UNIX の場合
  /opt/Cosminexus/CC/admin/templates/

このテンプレートファイルを使用すると，DB Connector をインポートする前に，Connector 属性ファイルを編集しておくことができます。なお，テンプレートファイルはコピーして使用してください。

Connector 属性ファイルのテンプレートファイル名については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.13　Connector 属性ファイルのテンプレートファイル」を参照してください。

なお，すでにプロパティが設定されている DB Connector のプロパティを変更する場合は，テンプレートファイルは使用しないでください。インポートした DB Connector の Connector 属性を取得して，Connector 属性ファイルを編集してください。

(2)　クラスタコネクションプールの状態と実行できる操作

メンバコネクションプールは，データベースの障害や保守などの場合，手動で一時停止できます。一時停止状態になると，ルートリソースアダプタへのコネクション取得要求時に処理が実行されません。

また，一時停止したメンバコネクションプールは，手動で再開できます。ルートリソースアダプタへのコネクション取得要求時には，開始状態のコネクションプールだけ処理が実行されます。

コネクションプールの状態制御については，「19.1.4　クラスタコネクションプールの動作」を参照してください。

(3)　リソースアダプタの種類による機能差異

リソースアダプタの種類で，使用できる機能を次に示します。

表 19-7　リソースアダプタの種類による機能一覧

機能	リソースアダプタの種類	
	ルートリソースアダプタ	メンバリソースアダプタ
コネクションプーリング	×	◎
コネクションプールのウォーミングアップ	×	○
コネクションシェアリング・アソシエーション	×	○
ステートメントプーリング	×	○
DataSource オブジェクトのキャッシング	○	×
DB Connector のコンテナ管理でのサインオンの最適化	×	○
コネクション障害検知	×	◎
コネクション障害検知のタイムアウト	×	◎
コネクション枯渇時のコネクション取得待ち	×	◎
コネクション取得リトライ	×	×
コネクションプール情報表示	×	○
コネクションプールクリア	×	○
コネクション自動クローズ	×	○
コネクションスイーパ	×	○
接続テスト	○	○
コネクションプール数調節機能	×	○
コネクションプール情報の表示	×	○
コネクションプールの一時停止※	×	○
コネクションプールの再開※	×	○
J2EE リソースのユーザ指定名前空間機能	○	×

（凡例）　◎：必ず有効になる　○：使用できる　×：使用できない

注※　コネクションプールをクラスタで使用しない場合，コネクションプールの一時停止および再開は実行できません。

DB Connector の属性を設定するリソースアダプタの種類を次に示します。

表 19-8　リソースアダプタの種類と属性設定

設定項目	リソースアダプタの種類	
	ルートリソースアダプタ	メンバリソースアダプタ
トランザクションサポートレベル	×	○
ログ取得可否	○	○
データベース接続情報	－	○
DB Connector 固有の設定（ステートメントプールなど）	－	○
セキュリティ情報（ユーザ名，パスワード）	×	○
コネクションプールサイズ	×	○
クラスタコネクションプール固有の設定	○	－

（凡例）　○：設定要　×：設定不要　－：設定項目なし

19.3.2　メンバリソースアダプタ用 DB Connector の設定

メンバリソースアダプタ用 DB Connector を，次の手順で設定します。

1. メンバリソースアダプタ用の DB Connector をインポートします。
2. プロパティを定義します。
3. メンバリソースアダプタ用の DB Connector をデプロイします。
4. 接続を確認します。

(1)　メンバリソースアダプタ用の DB Connector のインポート

次に示すコマンドを実行してメンバリソースアダプタ用の DB Connector をインポートします。

(a)　実行形式

```
cjimportres ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type rar -f <ファイルパス>
```

<ファイルパス>には，RAR ファイルを指定してください。

RAR ファイルは，次のディレクトリに格納されています。

- Windows の場合
  <Application Serverのインストールディレクトリ>¥CC¥DBConnector¥ClusterPool¥
- UNIX の場合

/opt/Cosminexus/CC/DBConnector/ClusterPool/

メンバリソースアダプタとしてインポートする RAR ファイルについて説明します。

DBConnector_Oracle_CP_ClusterPool_Member.rar

クラスタコネクションプールのメンバリソースアダプタです。ローカルトランザクションまたはトランザクションなし（トランザクションサポートレベルに LocalTransaction または NoTransaction を指定する）で使用します。Oracle JDBC Thin Driver の ConnectionPoolDataSource を使用して，Oracle に接続します。

J2EE アプリケーションのリソースリファレンスに設定して使用することはできません。

(b) 実行例

```
cjimportres MyServer -type rar -f ”c:¥Program Files¥Hitachi¥Cosminexus¥CC¥DBConnector¥ClusterPool¥DBConnector_Oracle_CP_ClusterPool_Member.rar”
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

(2) メンバリソースアダプタ用の DB Connector のプロパティ定義

メンバリソースアダプタ用の DB Connector のプロパティを定義します。プロパティを定義する手順については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2.2 DB Connector のプロパティ定義」を参照してください。ここでは，メンバリソースアダプタ用の DB Connector のプロパティの設定項目を説明します。

(a) メンバリソースアダプタ用 DB Connector の一般情報

設定できる DB Connector の一般情報属性（<outbound-resourceadapter>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
トランザクションサポートのレベル※	○	<transaction-support>
再認証のサポート有無	○	<reauthentication-support>

（凡例） ○：必須

注※ 一つのクラスタコネクションプールを構成するメンバリソースアダプタのトランザクションサポートレベルは，すべて同じにしてください。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1 Connector 属性ファイルの指定内容」を参照してください。

(b) メンバリソースアダプタ用コンフィグレーションプロパティ

メンバリソースアダプタ用 DB Connector のコンフィグレーションプロパティ（<config-property>タグ）と設定内容は，対応するリソースアダプタ（DBConnector_Oracle_CP.rar）と同じです。対応する

リソースアダプタのコンフィグレーションプロパティについては，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2.2　DB Connector のプロパティ定義」を参照してください。

(c)　実行時プロパティ

メンバリソースアダプタ用 DB Connector の実行時プロパティ（<outbound-resourceadapter> - <connection-definition> - <connector-runtime>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<property-name>
プロパティのデータ型	<property-type>
プロパティの値	<property-value>

定義するプロパティの数だけ，上記の設定を繰り返してください。

プロパティ名（<property-name>）には，次の項目を設定します。

プロパティ項目	プロパティ名（<property-name>）の設定項目
ユーザ名※	User
パスワード※	Password
コネクションプールにプールするコネクションの最小値	MinPoolSize
コネクションプールにプールするコネクションの最大値	MaxPoolSize
ログを出力するかどうかの選択	LogEnabled
コネクションの最終利用時刻から，コネクションを自動破棄（コネクションスイーパ）するかを判定するまでの時間	ConnectionTimeout
コネクションの自動破棄（コネクションスイーパ）が動作する間隔	SweeperInterval
コネクション枯渇時のコネクション取得要求をキューで管理する場合の待ち時間の最大値	RequestQueueTimeout
コネクションプール監視のアラート出力を有効にするかどうかの選択	WatchEnabled
コネクションプールを監視する間隔	WatchInterval
コネクションプール使用状態を監視するしきい値	WatchThreshold
コネクションプール監視結果のファイルを出力するかどうかの選択	WatchWriteFileEnabled
コネクション数調節機能が動作する間隔	ConnectionPoolAdjustmentInterval
コネクションプールのウォーミングアップ機能を有効にするかどうかの選択	Warmup

注※　一つのクラスタコネクションプールを構成するメンバリソースアダプタのユーザ名は，すべて同じにしてください。

注意事項

メンバリソースアダプタでは，次の項目は設定の有無に関係なく，常に「有効」になります。

プロパティ項目	有効値	プロパティ名（<property-name>）の設定項目
コネクションプールにプールするコネクションの最小値	コネクションプーリング機能は常に有効 「MinPoolSize」，「MaxPoolSize」に「0」を設定しても，デフォルト値の「10」が仮定されます。	MinPoolSize
コネクションプールにプールするコネクションの最大値		MaxPoolSize
プール内のコネクションに障害が発生しているかどうかをチェックする方法の選択	常に「1」（コネクション取得時の障害検知）	ValidationType
コネクション枯渇時にコネクション取得要求をキューで管理するかどうかの選択	常に「true」	RequestQueueEnable
ネットワーク障害検知機能のタイムアウトを有効にするかどうかの選択	常に「true」	NetworkFailureTimeout

また，実行時プロパティのコネクションリトライ回数（「RetryCount」）とコネクションリトライ待ち時間（「RetryInterval」）の設定に関係なく，コネクション取得リトライ機能は，常に無効となります。

(3) メンバリソースアダプタ用の DB Connector のデプロイ

メンバリソースアダプタ用の DB Connector は，デプロイすると J2EE リソースアダプタとして使用できます。なお，デプロイしたあとで，プロパティを定義することもできます。デプロイ後に定義する場合は，該当するメンバリソースアダプタ用の DB Connector が所属するルートリソースアダプタと，メンバリソースアダプタ用の DB Connector を停止した状態で実行してください。プロパティを定義する方法については，「19.3.2(2) メンバリソースアダプタ用の DB Connector のプロパティ定義」を参照してください。

次に示すコマンドを実行してメンバリソースアダプタ用の DB Connector をデプロイします。

実行形式

```
cjdeployrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜メンバリソースアダプタ用のDB Connector表示名＞
```

実行例

```
cjdeployrar MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

cjdeployrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeployrar（リソースアダプタのデプロイ）」を参照してください。

(4) メンバリソースアダプタ用の DB Connector の接続テスト

メンバリソースアダプタ用の DB Connector に設定した情報が正しいかどうか，接続テストで検証します。

次に示すコマンドを実行して，メンバリソースアダプタ用の DB Connector の接続テストを実施します。

実行形式

```
cjtestres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜メンバリソースアダプタ用のDB Connectorの表示名＞
```

実行例

```
cjtestres -type rar -resname DB_Connector_for_Oracle_ClusterPool_Member
```

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト）」を参照してください。

注意事項

一度接続テストをしたメンバリソースアダプタ用の DB Connector は，J2EE サーバを再起動するまで削除できません。メンバリソースアダプタ用の DB Connector を削除する場合は，そのメンバリソースアダプタ用の DB Connector が所属するルートリソースアダプタとメンバリソースアダプタ用の DB Connector を停止してから，J2EE サーバを再起動してください。

19.3.3 ルートリソースアダプタ用 DB Connector の設定

ルートリソースアダプタ用 DB Connector を，次の手順で設定します。

1. ルートリソースアダプタ用の DB Connector をインポートします。
2. プロパティを定義します。
3. ルートリソースアダプタ用の DB Connector をデプロイします。
4. 接続を確認します。

(1) ルートリソースアダプタ用の DB Connector のインポート

次に示すコマンドを実行してルートリソースアダプタ用の DB Connector をインポートします。

(a) 実行形式

```
cjimportres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -f ＜ファイルパス＞
```

＜ファイルパス＞には，RAR ファイルを指定してください。

RAR ファイルは，次のディレクトリに格納されています。

- Windows の場合
  ＜Application Serverのインストールディレクトリ＞¥CC¥DBConnector¥ClusterPool¥
- UNIX の場合

/opt/Cosminexus/CC/DBConnector/ClusterPool/

ルートリソースアダプタとしてインポートする RAR ファイルについて説明します。

DBConnector_CP_ClusterPool_Root.rar

クラスタコネクションプールのルートリソースアダプタです。属するメンバリソースアダプタがローカルトランザクションまたはトランザクションなし（トランザクションサポートレベルに LocalTransaction または NoTransaction を指定する）で，データベースに接続する場合に使用します。J2EE アプリケーションのリソースリファレンスに設定して使用します

(b) 実行例

```
cjimportres MyServer -type rar -f ”c:¥Program Files¥Hitachi¥Cosminexus¥CC¥DBConnector¥ClusterPool¥DBConnector_CP_ClusterPool_Root.rar”
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

(2) ルートリソースアダプタ用の DB Connector のプロパティ定義

ルートリソースアダプタ用の DB Connector のプロパティを定義します。プロパティを定義する手順については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2.2 DB Connector のプロパティ定義」を参照してください。ここでは，ルートリソースアダプタ用の DB Connector のプロパティの設定項目を説明します。

ルートリソースアダプタ用の DB Connector で，有効なプロパティを次に示します。下記以外の項目は，設定しても無視されます。

- ルートリソースアダプタ用の DB Connector の説明（<description>）
- ルートリソースアダプタ用の DB Connector 名称（<display-name>）
- コンフィグレーションプロパティ（<outbound-resourceadapter> - <connection-definition> - <config-property>）
- 実行時プロパティ（<outbound-resourceadapter> - <connection-definition> - <connector-runtime>）のログを出力するかどうかの選択（<LogEnabled>）
- 別名情報（<outbound-resourceadapter> - <connection-definition> - <connector-runtime> - <resource-external-property>）

コンフィグレーションプロパティの設定項目を次に示します。

項目	対応するタグ
コンフィグレーションプロパティ名	<config-property-name>
コンフィグレーションプロパティのデータ型	<config-property-type>

項目	対応するタグ
コンフィグレーションプロパティの値	<config-property-value>

定義するコンフィグレーションプロパティの数だけ，上記の設定を繰り返してください。

DBConnector_CP_ClusterPool_Root.rar を使用する場合の，コンフィグレーションプロパティの設定項目と設定例を次の表に示します。

表 19-9　DBConnector_CP_ClusterPool_Root.rar を使用する場合のコンフィグレーションプロパティの設定例

項目名	設定例
algorithm	RoundRobin
enableAutoPoolSuspend	true
enableAutoPoolResume	true
dbCheckInterval	30
memberResourceAdapterName1	DB_Connector_for_Oracle_ClusterPool_Member1
memberResourceAdapterName2	DB_Connector_for_Oracle_ClusterPool_Member2
logLevel	ERROR

コンフィグレーションプロパティ名（<config-property-name>）の「memberResourceAdapterName[n]」プロパティは，メンバリソースアダプタの表示名を設定します。memberResourceAdapterName[n]は，デフォルトでは優先度 2 まで定義してあります。さらに，メンバリソースアダプタを指定する場合には，プロパティを追加してください。優先度 n は，1～100 の範囲で指定してください。n は，連続している必要はありません。

(3)　ルートリソースアダプタ用の DB Connector のデプロイ

ルートリソースアダプタ用の DB Connector は，デプロイすると J2EE リソースアダプタとして使用できます。なお，デプロイしたあとで，プロパティを定義することもできます。デプロイ後に定義する場合は，該当するルートリソースアダプタ用の DB Connector を停止した状態で実行してください。プロパティを定義する方法については，「19.3.3(2)　ルートリソースアダプタ用の DB Connector のプロパティ定義」を参照してください。

次に示すコマンドを実行してルートリソースアダプタ用の DB Connector をデプロイします。

実行形式

```
cjdeployrar ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <ルートリソースアダプタ用のDB Connector表示名>
```

実行例

```
cjdeployrar MyServer -resname DB_Connector_for_ClusterPool_Root
```

cjdeployrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeployrar（リソースアダプタのデプロイ）」を参照してください。

(4) ルートリソースアダプタ用の DB Connector の接続テスト

ルートリソースアダプタ用の DB Connector に設定した情報が正しいかどうか，接続テストで検証します。

次に示すコマンドを実行して，ルートリソースアダプタ用の DB Connector の接続テストを実施します。

実行形式

```
cjtestres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜ルートリソースアダプタ用のDB Connectorの表示名＞
```

実行例

```
cjtestres -type rar -resname DB_Connector_for_ClusterPool_Root
```

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト）」を参照してください。

注意事項

一度接続テストをしたルートリソースアダプタ用の DB Connector は，J2EE サーバを再起動するまで削除できません。ルートリソースアダプタ用の DB Connector を削除する場合は，ルートリソースアダプタ用の DB Connector を停止してから，J2EE サーバを再起動してください。

19.3.4 メンバリソースアダプタ用 DB Connector の開始と停止

(1) メンバリソースアダプタ用の DB Connector の開始

次に示すコマンドを実行してメンバリソースアダプタ用の DB Connector を開始します。

実行形式

```
cjstartrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜メンバリソースアダプタ用のDB Connectorの表示名＞
```

実行例

```
cjstartrar MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

cjstartrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartrar（リソースアダプタの開始）」を参照してください。

(2) メンバリソースアダプタ用の DB Connector の停止

次に示すコマンドを実行して，メンバリソースアダプタ用の DB Connector を停止します。

実行形式

```
cjstoprar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜メンバリソースアダプタ用のDB Connectorの表示名＞
```

実行例

```
cjstoprar MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

cjstoprar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstoprar（リソースアダプタの停止)」を参照してください。

注意事項

メンバリソースアダプタが所属するルートリソースアダプタを停止してから，メンバリソースアダプタを停止してください。

19.3.5 ルートリソースアダプタ用 DB Connector の開始と停止

(1) ルートリソースアダプタ用の DB Connector の開始

次に示すコマンドを実行してルートリソースアダプタ用の DB Connector を開始します。

実行形式

```
cjstartrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜ルートリソースアダプタ用のDB Connectorの表示名＞
```

実行例

```
cjstartrar MyServer -resname DB_Connector_for_ClusterPool_Root
```

cjstartrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartrar（リソースアダプタの開始)」を参照してください。

注意事項

ルートリソースアダプタに所属するメンバリソースアダプタを開始してから，ルートリソースアダプタを開始してください。

(2) ルートリソースアダプタ用の DB Connector の停止

次に示すコマンドを実行して，ルートリソースアダプタ用の DB Connector を停止します。

実行形式

```
cjstoprar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜ルートリソースアダプタ用のDB Connectorの表示名＞
```

実行例

```
cjstoprar MyServer -resname DB_Connector_for_ClusterPool_Root
```

cjstoprar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstoprar（リソースアダプタの停止)」を参照してください。

注意事項

J2EE アプリケーション中の J2EE リソースがルートリソースアダプタ用の DB Connector を参照している場合は，J2EE アプリケーションを停止してから，ルートリソースアダプタ用の DB Connector を停止してください。

19.3.6　コネクションプールの状態の確認

クラスタコネクションプールで使用しているコネクションプールの状態は，次の二つの方法で参照できます。

- cjlistrar コマンドで，デプロイされているすべてのリソースアダプタの，リソースアダプタ名とメンバコネクションプールの状態を参照します。
- cjlistpool コマンドで，メンバコネクションプールの情報を参照します。
  メンバコネクションプールの情報を参照して，必要に応じてメンバコネクションプールを閉塞/一時停止/再開します。コネクションプールの状態定義については，「19.1.4　クラスタコネクションプールの動作」を参照してください。

(1)　コネクションプールの状態の参照

デプロイされているすべてのリソースアダプタについて，リソースアダプタ名とリソースアダプタの状態が表示されます。リソースアダプタがクラスタコネクションプールのメンバリソースアダプタの場合には，コネクションプールの状態も表示されます。

次に示すコマンドを実行して，コネクションプールの状態を参照します。

実行形式

```
cjlistrar ［＜サーバ名＞］ ［-nameserver ＜プロバイダURL＞］ -clusterpool
```

実行例

```
cjlistrar MyServer -clusterpool
```

cjlistrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistrar（リソースアダプタの一覧表示)」を参照してください。

(2) メンバコネクションプールの情報の参照

コネクションプールの情報が表示されます。リソースアダプタがクラスタコネクションプールのメンバリソースアダプタの場合には，メンバコネクションプールの情報も表示されます。

次に示すコマンドを実行して，すべてのリソースアダプタのコネクションプール情報を参照します。

実行形式

```
cjlistpool ［＜サーバ名＞］ ［-nameserver ＜プロバイダURL＞］ -resall
```

実行例

```
cjlistpool MyServer -resall
```

特定のリソースアダプタについて，コネクションプールの情報を表示する場合は次のコマンドを実行します。

実行形式

```
cjlistpool ［＜サーバ名＞］ -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjlistpool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

リソースアダプタのコネクションプール情報が表示されます。

cjlistpool コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistpool（コネクションプールの一覧表示)」を参照してください。

注意事項

cjlistpool コマンドは，ルートリソースアダプタに対しては実行できません。

19.3.7 コネクションプールの一時停止

データベースの障害や保守などの場合，メンバコネクションプールを手動で一時停止できます。一時停止したコネクションプールは，コネクション取得要求を受け付けません。

次のコマンドを実行して，クラスタコネクションプールのメンバコネクションプールを一時停止します。

実行形式

```
cjsuspendpool ［＜サーバ名＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜一時停止対象となるメンバリソースアダプタの表示名＞
```

実行例

```
cjsuspendpool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

cjsuspendpool コマンドの詳細については，「cjsuspendpool（メンバコネクションプールの一時停止）」を参照してください。

注意事項

cjsuspendpool コマンドを使用して，手動で一時停止したコネクションプールは，自動再開はできません。cjresumepool コマンドで手動再開してください。

19.3.8　コネクションプールの再開

一時停止したメンバコネクションプールは手動で再開できます。J2EE アプリケーションがルートリソースアダプタにコネクション取得要求をしたときに，再開したコネクションプールは，再びコネクション取得要求を受け付けることができるようになります。

次のコマンドを実行して，クラスタコネクションプールのメンバコネクションプールを再開します。

実行形式

```
cjresumepool ［<サーバ名>］ ［-nameserver <プロバイダURL>］ -resname <再開対象となるメンバリソースアダプタの表示名>
```

実行例

```
cjresumepool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

cjresumepool コマンドの詳細については，「cjresumepool（メンバコネクションプールの再開始）」を参照してください。

19.4　設定する項目と操作の概要

次のリソースアダプタの設定でのアプリケーション設定操作の概要を示します。

- データベースと接続するための設定（クラスタコネクションプールの場合）

19.4.1　データベースと接続するための設定（クラスタコネクションプールの場合）

DB Connector のコネクションプールをクラスタ化で使用する場合に必要な作業です。

表 19-10　データベースと接続するための設定（クラスタコネクションプールの場合）の概要

設定項目	内容	参照先
メンバリソースアダプタ用 DB Connector の設定	メンバリソースアダプタ用 DB Connector をインポート，設定，接続テストまで実行します。	19.3.2
ルートリソースアダプタ用 DB Connector の設定	ルートリソースアダプタ用 DB Connector をインポート，設定，接続テストまで実行します。	19.3.3
メンバリソースアダプタ用 DB Connector の開始と停止	メンバリソースアダプタ用 DB Connector を開始します。また，開始状態にあるメンバリソースアダプタ用 DB Connector を停止します。	19.3.4
ルートリソースアダプタ用 DB Connector の開始と停止	ルートリソースアダプタ用 DB Connector を開始します。また，開始状態にあるルートリソースアダプタ用 DB Connector を停止します。	19.3.5
コネクションプールの状態の確認	メンバコネクションプールの状態を参照します。	19.3.6
コネクションプールの一時停止	メンバコネクションプールを一時停止の状態にします。	19.3.7
コネクションプールの再開	一時停止の状態のメンバコネクションプールを再開します。	19.3.8

19.5　J2EE サーバで使用するリソース操作コマンド

ここでは，J2EE サーバで使用するリソース操作コマンドのうち，cjresumepool（メンバコネクションプールの再開始）および cjsuspendpool（メンバコネクションプールの一時停止）について説明します。

cjresumepool（メンバコネクションプールの再開始）

形式

```
cjresumepool [<サーバ名称>] [-nameserver <プロバイダURL>]
             -resname <リソースアダプタ表示名>
             [-resname <リソースアダプタ表示名> ...]
```

機能

指定したクラスタコネクションプールのメンバコネクションプールを再開します。一度に複数のメンバコネクションプールの再開もできます。再開に失敗したメンバコネクションプールがあった場合でも，すべてのメンバコネクションプールに対して再開を試みます。一つでも再開できなかった場合は，異常終了します。

リソースアダプタが開始状態の場合，このコマンドを実行すると，メンバコネクションプールは手動再開中状態になります。次に，メンバコネクションプールのウォーミングアップ機能が有効な場合は，メンバコネクションプール内にコネクションをプールし，その後，メンバコネクションプールは開始状態となります。

リソースアダプタが停止状態の場合，このコマンドを実行すると，メンバコネクションプールは開始予約状態になります。

引数

<サーバ名称>

接続先 J2EE サーバ名称を指定します。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

-nameserver <プロバイダ URL>

CORBA ネーミングサービスへのアクセスプロトコル，CORBA ネーミングサービスが稼働しているホスト名，およびそれが使用しているポート番号を次に示す形式で指定します。

```
<プロトコル名称>::<ホスト名称>:<ポート番号>
```

指定内容の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.1.2　プロバイダ URL について」を参照してください。

-resname <リソースアダプタ表示名>

再開対象となるメンバリソースアダプタの表示名を指定します。

クラスタコネクションプールのメンバリソースアダプタでないリソースアダプタを指定した場合，異常終了となります。

この引数は，指定したリソースアダプタの開始/停止状態に関係なく，このコマンド自体を実行できます。ただし，メンバコネクションプールの状態によって，コマンドを実行できる場合とできない場合があります。詳細は次の表を参照してください。

メンバコネクションプールの状態	コマンドの実行可否
自動一時停止状態 手動一時停止状態 自動一時停止予約状態 手動一時停止予約状態	実行可能
開始状態 開始予約状態 自動再開中状態 手動再開中状態 自動閉塞状態 手動閉塞状態	実行不可

入力例

```
cjresumepool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドが実行できません。

3：

タイムアウトエラーが発生しました。

9：

管理者特権がないため，コマンドが実行できません（Windows Server 2012，Windows Server 2008，Windows 8，Windows 7 または Windows Vista の場合）。

注意事項

- コマンドの引数にサーバ名称を指定する場合，コマンド名の直後にする必要があります。そのほかの引数は，サーバ名称より後ろ（サーバ名称を省略した場合はコマンド名の後ろ）にあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<アプリケーション名> -name）やオプション

名と値を非対応にすること（例：-nameserver <アプリケーション名> -name <プロバイダ URL>）はできません。

- コマンドの引数にサーバ名称を指定する場合，cjsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。cjsetup コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetup（J2EE サーバのセットアップとアンセットアップ)」を参照してください。

cjsuspendpool（メンバコネクションプールの一時停止）

形式

```
cjsuspendpool [<サーバ名称>][-nameserver <プロバイダURL>]
              -resname <リソースアダプタ表示名>
              [-resname <リソースアダプタ表示名> ...]
```

機能

クラスタコネクションプールのメンバコネクションプールを一時停止します。一度に複数のメンバコネクションプールの一時停止もできます。一時停止に失敗したメンバコネクションプールがある場合でも，すべてのメンバコネクションプールに対して一時停止を試みます。どれか一つでも一時停止できなかった場合は異常終了となります。

リソースアダプタが開始状態のときに，このコマンドを実行すると，メンバコネクションプールは手動閉塞状態になり，コマンドは終了します。その後，J2EE サーバでは一時停止処理が行われます。一時停止処理が完了すると，メンバコネクションプールは手動一時停止状態になります。

リソースアダプタが停止状態のときに，このコマンドを実行すると，メンバコネクションプールは手動一時停止予約状態になります。

引数

<サーバ名称>

接続先 J2EE サーバ名称を指定します。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

-nameserver <プロバイダ URL>

CORBA ネーミングサービスへのアクセスプロトコル，CORBA ネーミングサービスが稼働しているホスト名，およびそれが使用しているポート番号を次に示す形式で指定します。

```
<プロトコル名称>::<ホスト名称>:<ポート番号>
```

指定内容の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.1.2　プロバイダ URL について」を参照してください。

-resname <リソースアダプタ表示名>

一時停止対象となるメンバリソースアダプタの表示名を指定します。

クラスタコネクションプールのメンバリソースアダプタの表示名以外を指定した場合，異常終了となります。

この引数は，指定したリソースアダプタの開始/停止状態に関係なく，このコマンド自体を実行できます。ただし，メンバコネクションプールの状態によって，コマンドを実行できる場合とできない場合があります。詳細は次の表を参照してください。

メンバコネクションプールの状態	コマンドの実行可否
開始状態 開始予約状態 自動一時停止状態 自動一時停止予約状態	実行可能
手動一時停止状態 手動一時停止予約状態 自動再開中状態 手動再開中状態 自動閉塞状態 手動閉塞状態	実行不可

入力例

```
cjsuspendpool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドが実行できません。

3：

タイムアウトエラーが発生しました。

9：

管理者特権がないため，コマンドが実行できません（Windows Server 2012，Windows Server 2008，Windows 8，Windows 7 または Windows Vista の場合）。

注意事項

- コマンドの引数にサーバ名称を指定する場合，コマンド名の直後にする必要があります。そのほかの引数は，サーバ名称より後ろ（サーバ名称を省略した場合はコマンド名の後ろ）にあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<データソース表示名> -resname）やオプション名と値を非対応にすること（例：-nameserver <データソース表示名> -resname <プロバイダURL>）はできません。
- コマンドの引数にサーバ名称を指定する場合，cjsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。cjsetup コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetup（J2EE サーバのセットアップとアンセットアップ)」を参照してください。

19.6 Connector 属性ファイル

Connector 属性ファイルは，リソースアダプタの属性を取得，編集する場合に使用するファイルです。

19.6.1 DB Connector に設定する<config-property>タグに指定できるプロパティ

次に示す DB Connector に設定する<config-property>タグに指定できる値について説明します。

- クラスタコネクションプールを使用して Oracle に接続する場合（ルートリソースアダプタ）
- クラスタコネクションプールを使用して Oracle に接続する場合（メンバリソースアダプタ）

(1) クラスタコネクションプールを使用して Oracle に接続する場合（ルートリソースアダプタ）

- DBConnector_CP_ClusterPool_Root.rar
  トランザクション管理をしない場合，またはローカルトランザクションを使用する場合に使用します。
  指定できるプロパティについては，次の表を参照してください。

表 19-11 DBConnector_CP_ClusterPool_Root.rar を使用する場合に指定できるプロパティ

config-property-name	config-property-type	config-property-value
algorithm	java.lang.String	クラスタコネクションプール機能のコネクションプール選択方式を指定します。 • RoundRobin 最後に選択したコネクションプールの次の優先度のコネクションプールから優先度順に，コネクションの取得を試みます。優先度が最も低いコネクションプールに達した場合には，優先度が最も高いコネクションプールが選択されます。すべてのコネクションプールからコネクションが取得できない場合は，例外となります。 クラスタデータベースがアクティブ/アクティブ構成で，各インスタンスで負荷分散を図る場合に指定します。 デフォルト値は RoundRobin です。
dbCheckInterval	java.lang.Integer	一時停止状態のコネクションプールで，DB ノードの状態をチェックする間隔を，2～2147483647 の整数（単位：秒）で指定します。有効範囲外の値を指定した場合には，デフォルト値となります。デフォルト値は 30 です。 メンバリソースアダプタで Oracle JDBC Thin Driver を使用するときは，各メンバリソースアダプタの loginTimeout プロパティの値よりも長い時間を指定してください。

config-property-name	config-property-type	config-property-value
enableAutoPoolResume	java.lang.Boolean	コネクションプールの自動再開機能を有効とするか，無効とするかを指定します。 • true を設定した場合 コネクションプールの自動再開機能が有効となります。 • false を設定した場合 コネクションプールの自動再開機能が無効となります。 デフォルト値は true です。
enableAutoPoolSuspend	java.lang.Boolean	コネクションプールの自動一時停止機能を有効とするか，無効とするかを指定します。 • true を設定した場合 コネクションプールの自動一時停止機能が有効となります。 • false を設定した場合 コネクションプールの自動一時停止機能が無効となります。 デフォルト値は true です。
logLevel	java.lang.String	DB Connector が出力するログ・トレースのレベルを指定します。 次の値が指定できます。 • 0 または ERROR • 10 または WARNING • 20 または INFORMATION デフォルト値は，0 または ERROR です。
memberResourceAdapterName[n]	java.lang.String	優先度 n のメンバリソースアダプタの表示名を指定します。このプロパティは，デフォルトでは定義されていないので，必要に応じて追加してください。n は 1～100 の範囲で指定します。
memberResourceAdapterName1	java.lang.String	優先度 1 のメンバリソースアダプタの表示名を指定します。優先度は値が小さいほど高くなります。
memberResourceAdapterName2	java.lang.String	優先度 2 のメンバリソースアダプタの表示名を指定します。

(2) クラスタコネクションプールを使用して Oracle に接続する場合（メンバリソースアダプタ）

- DBConnector_Oracle_CP_ClusterPool_Member.rar
  トランザクション管理をしない場合，またはローカルトランザクションを使用する場合に使用します。
  指定できるプロパティについては，次の表を参照してください。

表 19-12　DBConnector_Oracle_CP_ClusterPool_Member.rar を使用する場合に指定できるプロパティ

config-property-name	config-property-type	config-property-value
CallableStatementPoolSize 「機能解説 基本・開発編(コンテナ共通機能)」－「3.14.4」	java.lang.Integer	コネクションプールに割り当てられるコネクションごとの CallableStatement のプール数を設定します。デフォルトは 10 です。 0 を指定した場合，ステートメントをプールしません。
CancelStatement	java.lang.Boolean	トランザクションタイムアウトやアプリケーション強制停止時に，Statement クラス，CallableStatement クラスおよび PreparedStatement クラスで実行中の SQL をキャンセルするかどうかを指定します。 • true を指定した場合 実行中の SQL をキャンセルします。 • false を指定した場合 実行中の SQL をキャンセルしません。 デフォルト値は true です。 専用サーバ接続をする場合は，false を指定してください。
ConnectionIDUpdate	java.lang.Boolean	コネクション ID を DataSource#getConnection メソッドごとに更新するかどうかを指定します。 • true を指定した場合 DataSource#getConnection メソッドのたびにコネクション ID を生成します。 • false を指定した場合 最初の DataSource#getConnection メソッドでコネクション ID を生成し，そのあとは更新しません。 デフォルト値は false です。
databaseName	java.lang.String	Oracle サーバ上の特定のデータベース名（SID）を指定します。設定された値は，Oracle JDBC Thin Driver の DataSource 系インタフェースの setDatabaseName メソッドに渡されます。
loginTimeout	java.lang.Integer	データベースへの接続試行のタイムアウトを，1～2147483647 の整数（単位：ミリ秒）で指定します。 有効範囲外の値を指定した場合には，デフォルト値が使用されます。デフォルト値は 8000 です。設定された値は秒単位に切り上げて Oracle JDBC Thin Driver の DataSource 系インタフェースの setLoginTimeout メソッドに渡されます。
logLevel	java.lang.String	DB Connector が出力するログ・トレースのレベルを指定します。 次の値が指定できます。 • 0 または ERROR • 10 または WARNING • 20 または INFORMATION デフォルトは，0 または ERROR です。

config-property-name	config-property-type	config-property-value
portNumber	java.lang.Integer	Oracle のサーバが要求をリスニングするポート番号を指定します。デフォルトは 1521 番ポートです。設定された値は，Oracle JDBC Thin Driver の DataSource 系インタフェースの setPortNumber メソッドに渡されます。
PreparedStatementPoolSize 「機能解説 基本・開発編(コンテナ共通機能)」－「3.14.4」	java.lang.Integer	コネクションプールに割り当てられるコネクションごとの PreparedStatement のプール数を設定します。デフォルトは 10 です。 0 を指定した場合，ステートメントをプールしません。
serverName	java.lang.String	Oracle サーバのホスト名または IP アドレスを指定します。設定された値は，Oracle JDBC Thin Driver の DataSource 系インタフェースの setServerName メソッドに渡されます。
url	java.lang.String	Oracle JDBC Thin Driver がデータベースに接続するために必要な JDBC URL を指定します。設定された値は Oracle JDBC Thin Driver の setURL メソッドに渡されます。 このプロパティに値が設定された場合，databaseName，portNumber，serverName で指定された値は無視されます。また，ユーザが url で指定を行う場合は JDBC URL に thin ドライバを指定します。 （例） jdbc:oracle:thin:@ServerA:1521:service1

20 スレッドの非同期並行処理

この章では，Timer and Work Manager for Application Servers に準拠した TimerManager および WorkManager によるスレッドの非同期並行処理について説明します。

この機能は互換機能のため推奨しません。Java EE 標準の，Concurrency Utilities for Java EE の使用を検討してください。

20.1　この章の構成

スレッドの非同期並行処理の機能と参照先を次の表に示します。

表 20–1　スレッドの非同期並行処理の機能

機能名	参照先
スレッドの非同期並行処理の概要	20.2
TimerManager を使用した非同期タイマ処理	20.3
WorkManager を使用した非同期スレッド処理	20.4

20.2　スレッドの非同期並行処理の概要

アプリケーションサーバでは，Java EE 環境で非同期タイマ処理や非同期スレッド処理などのスレッドの非同期並行処理を実行できます。

Java EE の標準仕様では，サーブレットから新しいスレッドを生成したり EJB がスレッドを管理したりできません。また，スレッドの非同期並行処理は原則的に推奨されていません。そのため，アプリケーションサーバでは，Java EE 環境でスレッドの非同期並行処理を実現するために，CommonJ が定めた Timer and Work Manager for Application Servers の仕様に基づいた API を提供します。

スレッドの非同期並行処理を実現するための API の概要を次に示します。

- TimerManager

  Timer for Application Servers の仕様に準拠した API です。この API を使用すると，実行間隔を指定してスレッドの非同期処理をスケジューリングできます。この機能を**非同期タイマ処理**と呼びます。

- WorkManager

  Work Manager for Application Servers の仕様に準拠した API です。この API を使用すると，スレッドの非同期処理ができます。この機能を**非同期スレッド処理**と呼びます。

TimerManager および WorkManager は，EJB またはサーブレットから使用できます。

アプリケーションサーバの，Timer and Work Manager for Application Servers への対応状況については，「20.2.3　Timer and Work Manager for Application Servers との対応」を参照してください。

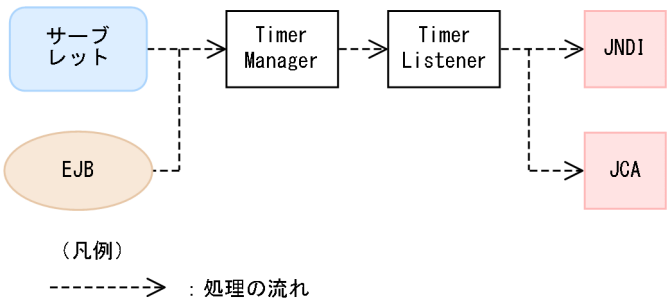
20.2.1　スレッドの非同期並行処理の流れ

スレッドの非同期並行処理を実行するには，EJB およびサーブレットから TimerManager または WorkManager をルックアップします。ここでは，TimerManager を使用した非同期タイマ処理，および WorkManager を使用した非同期スレッド処理の流れについて説明します。

TimerManager を使用した非同期タイマ処理の流れ

TimerManager を使用した非同期タイマ処理の流れを次の図に示します。

図 20–1　TimerManager を使用した非同期タイマ処理の流れ



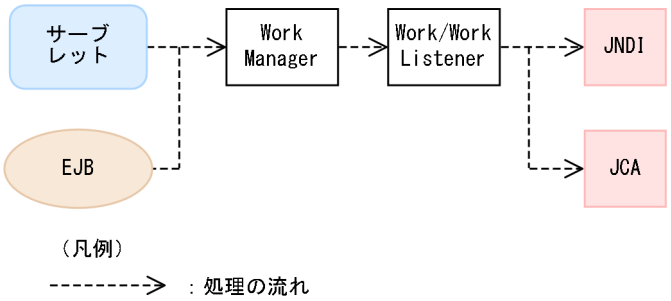
EJB およびサーブレットは，実行する非同期並行処理を呼び出す，スケジュール元となります。

TimerManager は，JNDI によるルックアップ時に作成されます。実行する処理の実体は，TimerManager が提供するリスナである TimerListener に実装します。TimerListener は，必要に応じて JNDI や JCA にアクセスして処理を実行します。

WorkManager を使用した非同期スレッド処理の流れ

WorkManager を使用した非同期スレッド処理の流れを次の図に示します。

図 20–2　WorkManager を使用した非同期スレッド処理の流れ



EJB およびサーブレットは，実行する非同期並行処理を呼び出す，スケジュール元となります。

WorkManager は，アプリケーション開始時に作成されます。JNDI によってルックアップされた場合は，アプリケーション開始時に作成された WorkManager を返します。実行する処理の実体は，WorkManager が提供する Work または WorkListener に実装します。Work または WorkListener は，必要に応じて JNDI や JCA にアクセスして処理を実行します。

20.2.2　スレッドの非同期並行処理で使用できる Java EE の機能

非同期並行処理として実行する処理の中では，Java EE の機能を使用できます。Java EE の機能を使用できる TimerManager および WorkManager の API を次に示します。

TimerManager

- TimerListener.timerExpired
  設定した時間に達したときに実行されるメソッドです。
- StopTimerListener.timerStop
  TimerManager.stop メソッドが実行されたとき，またはアプリケーションが停止したときに実行されるメソッドです。
- CancelTimerListener.timerCancel
  TimerManager.cancel メソッドが実行されたときに実行されるメソッドです。

WorkManager

- Work.run
  WorkManager で非同期に実行される処理メソッドです。
- WorkListener.workAccepted
  スケジュールした Work を WorkManager が受け付けるときに実行されるメソッドです。

- WorkListener.workCompleted
  スケジュールされた Work の run メソッドが終了した直後に実行されるメソッドです。
- WorkListener.workRejected
  スケジュールした Work を WorkManager が受け付けたあとに，スケジュール処理を継続できなくなった場合に実行されるメソッドです。
- WorkListener.workStarted
  スケジュールされた Work の run メソッドが実行される直前に実行されるメソッドです。

それぞれの API の詳細については，Timer and Work Manager for Application Servers の API 仕様を参照してください。

TimerManager および WorkManager で使用できる Java EE 機能を次の表に示します。

表 20–2　TimerManager および WorkManager で使用できる Java EE の機能

機能名	使用可否	参照先
Enterprise Bean の呼び出し	×	−
ネーミングサービス	○※	(1)
トランザクションサービスとリソース接続	○※	(2)
ログとトレースの出力	○	(3)
コンテナ拡張ライブラリの利用	○	(4)
メソッドキャンセル	×	−

（凡例）
○：使用できる
×：使用できない
−：該当なし

注※
ただし，一部の機能については使用できません。使用できる機能については，「参照先」の列に示す情報を参照してください。

次に，TimerManager および WorkManager で使用できる機能を詳細に分類して説明します。また，それぞれの機能を使用する場合の注意事項についても説明します。

(1)　ネーミングサービス

ネーミングサービスとして提供する機能が TimerManager および WorkManager で使用できるかどうかを次の表に示します。

表 20–3　ネーミングサービスの機能の使用可否

機能名	使用可否
JNDI を使用した DB Connector のルックアップ	○

機能名	使用可否
JNDI を使用した Java Mail のルックアップ	×
JNDI を使用した JavaBeans リソースのルックアップ	×
JNDI を使用した EntityManager のルックアップ	×
JNDI を使用した EntityManagerFactory のルックアップ	×
JNDI を使用した TimerManager のルックアップ	×※1
JNDI を使用した WorkManager のルックアップ	×※1
JNDI を使用したユーザトランザクションのルックアップ	○※2

（凡例）

○：使用できる

×：使用できない

注※1　TimerManager や WorkManager のスケジュールの延長で，さらに TimerManager や WorkManager を呼び出すことはできません。

注※2　スケジュール元が，CMT でトランザクションを管理する EJB の場合は，java:comp/UserTransaction でルックアップできません。必ず HITACHI_EJB/SERVERS/<J2EE サーバ名>/SERVICES/UserTransaction を使用してルックアップしてください。

注意事項

スケジュール元で取得した DB Connector やユーザトランザクションを WorkManager または TimerManager 中で使用しないでください。必ず実行した処理を実装した TimerListener または Work 内で取得してください。

(2)　トランザクションサービスとリソース接続

リソースアダプタには，DB Connector だけを使用できます。TimerManager および WorkManager で使用できる DB Connector を次の表に示します。

表 20-4　DB Connector の使用可否

DB Connector 名	使用可否
DBConnector_HiRDB_Type4_CP.rar	○
DBConnector_HiRDB_Type4_XA.rar	○
DBConnector_Oracle_CP.rar	○
DBConnector_Oracle_XA.rar	○
DBConnector_MySQL_CP.rar	×
DBConnector_PostgreSQL_CP.rar	×
DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar	×

DB Connector 名	使用可否
DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar	×
DBConnector_Oracle_CP_Cosminexus_RM.rar	×
DBConnector_Oracle_XA_Cosminexus_RM.rar	×
DBConnector_CP_ClusterPool_Root.rar	×
DBConnector_Oracle_CP_ClusterPool_Member.rar	×

（凡例）

○：使用できる

×：使用できない

DB Connector を使用する場合，トランザクションサポートレベルには，NoTransaction，LocalTransaction，または XATransaction を指定してください。DB Connector のコネクションを取得するには，DB Connector の別名を設定する必要があります。JNDI によるルックアップでは，設定した別名を使用して，DB Connector のコネクションを取得してください。DB Connector の別名を使用したコネクションの取得方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.6　Enterprise Bean または J2EE リソースへの別名付与（ユーザ指定名前空間機能)」を参照してください。

リソース接続およびトランザクションサービスとして提供する機能が TimerManager および WorkManager で使用できるかどうかを次の表に示します。

表 20–5　トランザクションサービスの機能の使用可否

機能名		使用可否
トランザクション (ユーザトランザクション)	ローカルトランザクション	○
	グローバルトランザクション	○
トランザクションの自動決着※1		△
トランザクションタイムアウト		○
コネクションプーリング	DB Connector によるコネクションプーリング	○
	コネクションプールのウォーミングアップ	○
	コネクション数調節	○
コネクションシェアリング※2		△
コネクションアソシエーション		×
DB Connector のステートメントプーリング		○
コネクションの障害検知		○
コネクション枯渇時のコネクション取得待ち		○
コネクション取得リトライ		○

機能名	使用可否
コネクション自動クローズ	×
コネクションスイーパ	○
障害調査用 SQL の出力	○

（凡例）

○：使用できる

△：一部の機能が使用できない

×：使用できない

注※1　ユーザトランザクションは，リスナの処理メソッドからリターンする前に決着する必要があります。トランザクションが決着していない場合，例外が発生しなくてもトランザクションはロールバックされて，メッセージ（KDJE43179-W）が出力されます。

注※2　シェアリングできるコネクションの範囲は，デフォルトで設定される「同一トランザクション」だけです。

注意事項

取得した DB Connector のコネクションは自動でクローズされないため，必ずメソッド内でコネクションをクローズするように設定してください。

(3)　ログとトレースの出力

ログとトレースを出力する機能が TimerManager および WorkManager で使用できるかどうかを次の表に示します。

表 20-6　ログとトレースの機能の使用可否

機能名	使用可否
ユーザログ	○
性能解析トレース	○

（凡例）

○：使用できる

参考

性能解析トレースのオペレーション名について

TimerManager および WorkManager の性能解析トレースでは，スケジュールごとに一意の番号を取得できます。この情報は，トレース情報のオペレーション名に出力されます。取得できるトレース情報の詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「8.　性能解析トレースのトレース取得ポイントと PRF トレース取得レベル」を参照してください。

(4) コンテナ拡張ライブラリの利用

コンテナ拡張ライブラリの機能は，TimerManager および WorkManager を使用しない場合と同様に使用できます。

20.2.3 Timer and Work Manager for Application Servers との対応

Timer and Work Manager for Application Servers の仕様で，ベンダ依存と規定されている仕様には，アプリケーションサーバでは対応していません。また，CommonJ が提供する API とアプリケーションサーバが提供する API には，仕様差があります。ここでは，アプリケーションサーバが対応していない Timer for Application Servers の仕様，アプリケーションサーバが対応していない Work Manager for Application Servers の仕様，および CommonJ とアプリケーションサーバで動作が異なる API について説明します。

(1) アプリケーションサーバが対応していない Timer for Application Servers の仕様

アプリケーションサーバが対応していない Timer for Application Servers の仕様について次の表に示します。

表 20–7 アプリケーションサーバが対応していない Timer for Application Servers の仕様（ベンダ依存機能）

アプリケーションサーバが対応していない仕様	備考
最大スケジューリング数のカスタマイズ	最大スケジューリング数は 50 です。変更できません。
• TimerManager のリスナを実装するクラス • Java EE のコンポーネントを継承しない一般の Java オブジェクト以外のオブジェクト（EJB やサーブレットなど）	javax.ejb.EnterpriseBean を継承するクラスをスケジュールした場合，エラーになります。それ以外の場合はエラーチェックをしません。
実行スレッドへのトランザクションコンテキストの継承項目	スケジュール元のトランザクション状態に関係なく，トランザクションなしとなります。CMT の NOT_SUPPORTED に相当します。
実行スレッドへの J2EE コンテキストの継承項目のカスタマイズ	継承される項目は一定です。
実行スレッドで使用できる Java EE の機能	使用できる機能については「20.2.2 スレッドの非同期並行処理で使用できる Java EE の機能」を参照してください。

なお，Timer for Application Servers では，J2EE アプリケーション中で使用できるコンポーネントについて規定されていません。アプリケーションサーバの場合に TimerManager で使用できるコンポーネントについては，「20.3.5 TimerManager を使用したアプリケーションの開発」を参照してください。

(2) アプリケーションサーバが対応していない Work Manager for Application Servers の仕様

アプリケーションサーバが対応していない Work Manager for Application Servers の仕様について次の表に示します。

表 20-8 アプリケーションサーバが対応していない Work Manager for Application Servers の仕様（ベンダ依存機能）

アプリケーションサーバが対応していない仕様	備考
WorkManager を使用した非同期スレッド処理のリモートでの実行	WorkItem をリモートで実行した場合，ローカルで実行されるダミーの RemoteWorkItem を返します。
最大スケジューリング数のカスタマイズ	最大スケジューリング数には制限がありません。 | • TimerManager のリスナを実装するクラス • Java EE のコンポーネントを継承しない一般の Java オブジェクト以外のオブジェクト（EJB やサーブレットなど）	javax.ejb.EnterpriseBean を継承するクラスをスケジュールした場合，エラーになります。それ以外の場合，エラーチェックはしません。
アプリケーションの開始以外のタイミングでの WorkManager の作成	WorkManager はアプリケーションの開始時にだけ作成されます。
実行スレッドへのトランザクションコンテキストの継承項目	スケジュール元のトランザクション状態に関係なく，トランザクションなしとなります。CMT の NOT_SUPPORTED に相当します。
実行スレッドへの J2EE コンテキストの継承項目のカスタマイズ	継承される項目は一定です。
実行スレッドで使用できる Java EE の機能	使用できる機能については「20.2.2　スレッドの非同期並行処理で使用できる Java EE の機能」を参照してください。

なお，Work Manager for Application Servers では，J2EE アプリケーション中で使用できるコンポーネントについて規定されていません。アプリケーションサーバの場合に WorkManager で使用できるコンポーネントについては，「20.4.4　WorkManager を使用したアプリケーションの開発」を参照してください。

(3) CommonJ とアプリケーションサーバで動作が異なる API

CommonJ とアプリケーションサーバで動作が異なる API を次の表に示します。

表 20-9 CommonJ とアプリケーションサーバで動作が異なる API

クラス	メソッド	アプリケーションサーバでの動作
commonj.timers.TimerManager	schedule(TimerListener listener,Date time)	listener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。

クラス	メソッド	アプリケーションサーバでの動作
	schedule(TimerListener listener,long delay)	listener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。
	schedule(TimerListener listener,Date firstTime,long period)	listener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。
	schedule(TimerListener listener,long delay,long period)	listener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。
	scheduleAtFixedRate(TimerListener listener,Date firstTime,long period)	listener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。
	scheduleAtFixedRate (TimerListener listener,long delay,long period)	listener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。
commonj.work.WorkManager	schedule(Work work)	work が null の場合，WorkException を返します。
	schedule(Work work,WorkListener wl)	work が null の場合，WorkException を返します WorkListener が javax.ejb.EnterpriseBean を継承している場合，IllegalArgumentException を返します。

20.3　TimerManager を使用した非同期タイマ処理

この節では，TimerManager を使用した非同期タイマ処理について説明します。

この節の構成を次に示します。

表 20–10　この節の構成（TimerManager を使用した非同期タイマ処理）

分類	タイトル	参照先
解説	TimerManager を使用したスレッドのスケジューリング方式	20.3.1
	TimerManager のライフサイクル	20.3.2
	TimerManager のステータス遷移	20.3.3
	TimerManager の多重スケジュール数	20.3.4
実装	TimerManager を使用したアプリケーションの開発	20.3.5

注 「設定」「運用」および「注意事項」について，この機能固有の説明はありません。

TimerManager を使用した非同期タイマ処理では，Java EE 環境で，実行間隔を指定してスレッドの非同期処理をスケジューリングできます。バックグラウンドでは，コンテナで管理されたスレッドを使用するため，安全にタスクを実行できます。

スケジューリングする処理は，TimerListener で実装します。スケジュール元となる EJB やサーブレットで TimerManager のメソッドを実行することで，TimerListener に実装した処理がスケジューリングされます。また，TimerManager の schedule メソッドから返された Timer を使用することで，スケジュールの応答やキャンセルができます。

TimerManager を使用するには，EJB 属性やサーブレット属性の<resource-ref>タグに，TimerManager に関する情報を定義します。EJB やサーブレットは，デプロイ時に<res-ref-name>タグに定義した名前でルックアップして TimerManager を使用します。

20.3.1　TimerManager を使用したスレッドのスケジューリング方式

TimerManager を使用したスレッドのスケジューリングには，次の二つの方式があります。

- 1 回だけ処理を実行する
- 一定の間隔で処理を繰り返し実行する

それぞれのスケジューリング方式の概要を次に示します。

(1)　1 回だけ処理を実行する

指定した時間に，1 回だけ処理を実行する方式です。処理が実行されると，TimerManager は破棄されます。

(2) 一定の間隔で処理を繰り返し実行する

一定の間隔で処理を繰り返し実行するには，次の二つの方法があります。

- fixed-rate（処理を開始する間隔を指定して繰り返し実行する）
- fixed-delay（処理の終了から次の処理の開始までの間隔を指定して繰り返し実行する）

スケジューリングされた処理は，TimerManager を停止するか，対応する Timer の cancel メソッドが実行されるまで，処理を実行し続けます。

それぞれの方法の概要を次に示します。

fixed-rate（処理を開始する間隔を指定して繰り返し実行する）

一定の間隔で処理を繰り返し開始していく方式です。fixed-rate では，次の内容を指定します。

最初の処理を開始するタイミング

次のどちらかの方法で指定します。

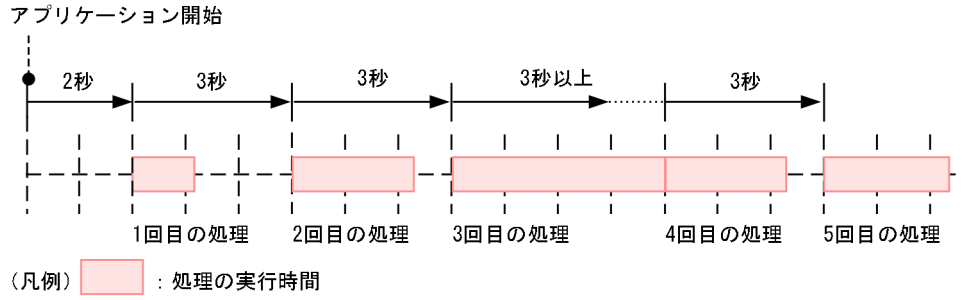
- 開始時刻を指定する場合
  scheduleAtFixedRate メソッドの引数である firstTime を Date 型で指定します。
- アプリケーションを開始してから処理を実行するまでの経過時間を指定する場合
  scheduleAtFixedRate メソッドの引数である delay を long 型で指定します。単位はミリ秒です。

前の処理を開始してから次の処理を開始するまでの間隔

scheduleAtFixedRate メソッドの引数である period を long 型で指定します。単位はミリ秒です。

fixed-rate の処理のイメージを次の図に示します。この図では，アプリケーションの開始から最初の処理を開始するまでの時間を 2 秒，前の処理を開始してから次の処理を開始するまでの時間を 3 秒としています。

図 20–3　fixed-rate の処理のイメージ



注　横軸は時間を示します。

fixed-rate の処理では，前に実行された処理時間が period に指定した時間より長かった場合，前に実行された処理が終わった直後に次の処理が開始されます。この図では，3 回目の処理時間が period で指定した時間である 3 秒より長かったため，3 回目の処理が終わった直後に 4 回目の処理が開始されています。

fixed-delay（処理の終了から次の処理の開始までの間隔を指定して繰り返し実行する）

前の処理が終了してから，一定の間隔を置いて処理を繰り返し開始していく方式です。fixed-delay では，次の内容を指定します。

最初の処理を開始するタイミング

次のどちらかの方法で指定します。

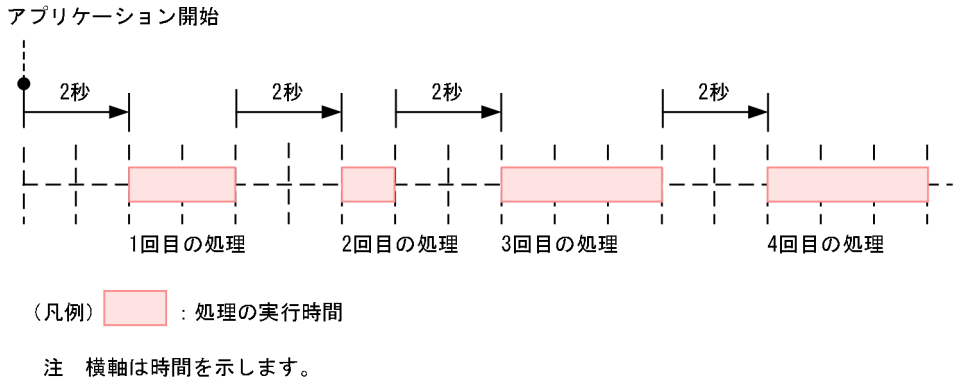
- 開始時刻を指定する場合
  schedule メソッドの引数である firstTime を Date 型で指定します。
- アプリケーションを開始してから処理を実行するまでの経過時間を指定する場合
  schedule メソッドの引数である delay を long 型で指定します。単位はミリ秒です。

前の処理が終了してから次の処理を開始するまでの間隔

schedule メソッドの引数である period を long 型で指定します。単位はミリ秒です。

fixed-delay の処理のイメージを次の図に示します。この図では，アプリケーションの開始から最初の処理を開始するまでの時間，および前の処理が終了してから次の処理を開始するまでの時間を 2 秒としています。

図 20–4　fixed-delay の処理のイメージ



20.3.2　TimerManager のライフサイクル

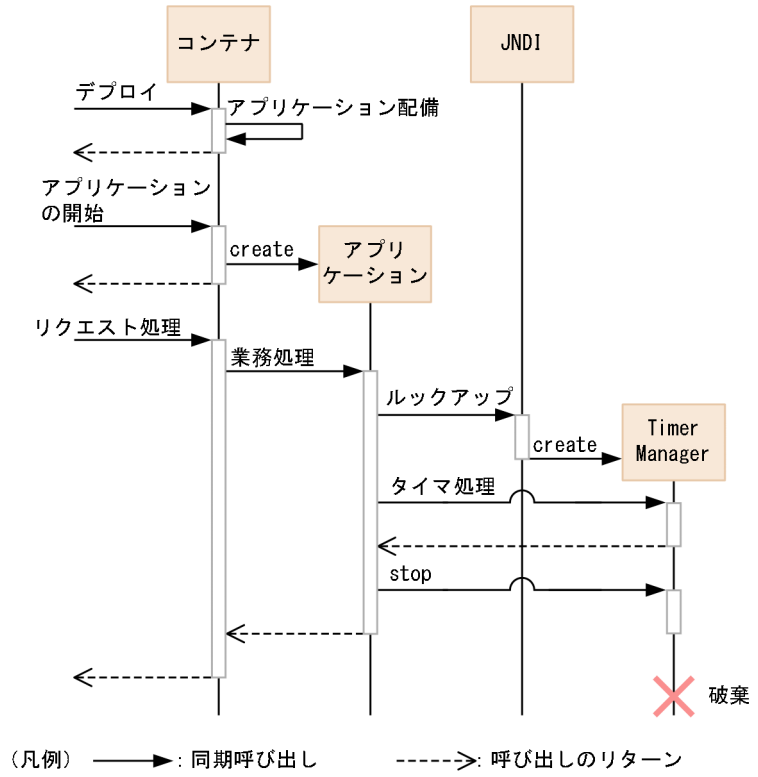
ここでは，TimerManager のライフサイクルについて説明します。

TimerManager は，アプリケーション中で JNDI によってルックアップされた時に作成されます。ルックアップされるたびに，新しい TimerManager が作成されます。作成された TimerManager は，アプリケーション中で stop メソッドを実行して，明示的に停止することをお勧めします。stop メソッドを実行しないで自動で TimerManager を停止することもできますが，その場合，TimerManager が停止するまでアプリケーションも停止しません。そのため，TimerManager の停止処理に伴って，アプリケーションの停止にも時間が掛かることがあります。

また，TimerManager は永続化されません。そのため，JavaVM が終了した場合，作成された TimerManager およびスケジュールされたタイマは破棄されます。

TimerManager のライフサイクルを次の図に示します。

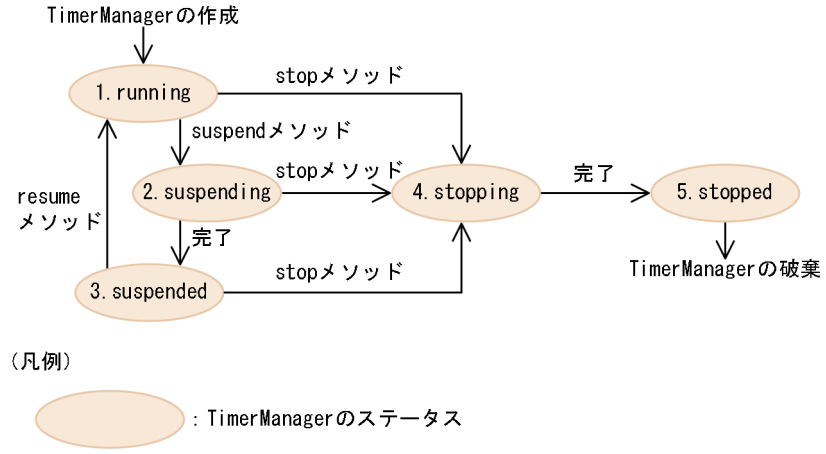
図 20-5　TimerManager のライフサイクル



20.3.3　TimerManager のステータス遷移

TimerManager は，サスペンド（一時停止），リジューム（再開）などによる閉塞や，停止の処理に伴って，ステータスが遷移します。そのときの TimerManager のステータスは，isStopped メソッド，isStopping メソッド，isSuspended メソッドなどを使用して確認できます。TimerManager のステータス遷移を次の図に示します。

図 20-6　TimerManager のステータス遷移



それぞれのステータスの詳細について次の表に示します。

表 20–11　TimerManager のステータス

図中の番号※	ステータス	説明
1	running	TimerManager が実行中の状態です。新規スケジュールの受付と実行ができます。
2	suspending	サスペンド実行中の状態です。サスペンドが実行されたときに，実行中のタスクが残っていることを示します。実行中のタスクがなくなるとステータスが suspended に遷移します。
3	suspended	すべてのタスクがサスペンドされている状態です。ステータスが suspended のときは，スケジューリングされたすべてのタスクは実行されません。ステータスが suspended のタスクは，リジュームされた時に実行されます。
4	stopping	TimerManager の停止処理が実行中の状態です。停止処理が実行されたときに，実行中のタスクが残っていることを示します。実行中のタスクがなくなるとステータスが stopped に遷移します。
5	stopped	TimerManager が停止した状態です。すべてのタスクが停止されて，それ以降実行されないことを示しています。一度停止した TimerManager は再開できません。

注※　番号は，図 20-6 の番号を示します。

20.3.4　TimerManager の多重スケジュール数

TimerManager でスケジュールされたタイマ処理では，スレッドプールで管理されたスレッドを使用します。タイマ処理がスケジュールされると，スレッドプールで管理されたスレッドが一つ割り当てられます。スレッドプールに空きスレッドがある場合は，空きスレッドが使用されます。スレッドプールに空きスレッドがない場合は，新しいスレッドが生成されて使用されます。スレッドプールに生成されたスレッドは，TimerManager が停止するまでプールされます。

同時に使用できるスレッドのインスタンスごとの最大数は，50 です。なお，スケジュールしたタイマ処理が待機中の場合も，スレッドは割り当てられます。そのため，同時にスケジュールできる処理の最大数は，タイマ処理の状態に関係なく 50 です。

すでに生成されたスレッドが最大数に達している場合，スケジュールされたタイマ処理はキューに格納されて，スレッドに空きができるまで待機します。空きスレッドができ次第，キューに格納されたタイマ処理が実行されます。

51 以上のスレッドを同時にスケジュールする場合は，複数の TimerManager を使用してください。

20.3.5　TimerManager を使用したアプリケーションの開発

ここでは，TimerManager を使用したアプリケーションの開発について説明します。

TimerManager を使用する場合の，アプリケーションを構成するコンポーネントの使用可否を次の表に示します。

表 20–12　TimerManager を使用する場合のアプリケーションを構成するコンポーネントの使用可否

コンポーネント				使用可否
EJB クライアント				×
リソースアダプタ				×
JavaBeans リソース				×
サーブレット／JSP※				○
EJB	Stateless Session Bean	EJB2.1 以前	CMT	○
			BMT	○
		EJB3.0		×
	Stateful Session Bean			×
	Entity Bean			×
	Message-driven Bean			×

（凡例）

○：使用できる

×：使用できない

注※　サーブレットリスナやフィルタでも使用できます。

TimerManager を使用するアプリケーションの開発の流れは次のとおりです。

1. スケジュール元となる EJB またはサーブレットの属性を定義する
2. TimerManager のリスナに実行する処理を実装する
3. スケジュール元となる EJB またはサーブレットを作成する
4. TimerManager を使用する J2EE アプリケーションをコンパイルする

それぞれの作業の詳細を次に示します。

(1)　スケジュール元となる EJB またはサーブレットの属性を定義する

TimerManager を使用する EJB またはサーブレットの属性を DD に定義します。TimerManager を使用するための定義は，アノテーションでは実施できません。

TimerManager を使用するために定義する必要がある属性を次の表に示します。

表 20-13　TimerManager を使用するために定義する必要がある属性

タグ名		説明
＜ルートタグ＞		－
├	<description>	任意で設定してください。
├	<res-ref-name>	JNDI ENC 名（JNDI ルックアップに使用する名前）を指定してください。
├	<res-type>	次の内容を設定してください。 「commonj.timers.TimerManager」
├	<res-auth>	設定した値は無視されます。
├	<res-sharing-scope>	「Unshareable」を設定してください。ただし，「Shareable」を設定しても，「Unshareable」設定時と同様に動作します（ルックアップされるたびに新しい TimerManager が作成されます）。
├	<mapped-name>	設定した値は無視されます。
├	<injection-target>	設定した値は無視されます。
└	<linked-to>	設定した値は無視されます。

サーブレットで TimerManager を使用する場合の web.xml の定義例を次に示します。

```
<web-app>
  <display-name>TimerManagerSample</display-name>
  <servlet>
    <servlet-name>SampleServlet</servlet-name>
    <display-name>SampleServlet</display-name>
    <servlet-class>SampleServlet</servlet-class>
  </servlet>
  . . .
  <resource-ref>
    <res-ref-name>timer/MyTimer</res-ref-name>
    <res-type>commonj.timers.TimerManager</res-type>
```

```
    <res-auth>Container</res-auth>
    <res-sharing-scope>Unshareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

TimerManager は，アプリケーション中で JNDI によるルックアップが実行されるたびに作成されます。作成された TimerManager は，stop メソッドの実行時，またはアプリケーション終了時に破棄されます。なお，TimerManager は必要に応じて複数定義することもできます。

(2) TimerManager のリスナに実行する処理を実装する

TimerManager を使用するには，実行する処理を実装したリスナを作成する必要があります。リスナのインタフェースには，必ず実装するものと，必要に応じて実装するものがあります。必ず実装するインタフェースと必要に応じて実装するインタフェースを次に示します。

必ず実装するインタフェース

- TimerListener

必要に応じて実装するインタフェース

- StopTimerListener
- CancelTimerListener

API の詳細については，Timer and Work Manager for Application Servers の API の仕様を参照してください。

TimerListener，StopTimerListener および CancelTimerListener を実装したクラスの例を次に示します。

```
public class MyTimerListener
        implements TimerListener,StopTimerListener, CancelTimerListener {
    private int count = 0;

        public MyTimerListener() {
        }
        public void timerStop(Timer timer) {
                System.out.println("Timer stopped: " + timer);
        }

        public void timerCancel(Timer timer) {
                System.out.println("Timer cancelled: " + timer);
        }

        public void timerExpired(Timer timer) {
                System.out.println("Timer expired !");
                    if(count++ > 10) {
                //規定回数に到達したためキャンセル
                timer.cancel();
            } else {
             System.out.println("The next timer will fire at : " +
                                timer.getScheduledExecutionTime());
            }
```

```
        }
}
```

(3) スケジュール元となる EJB またはサーブレットを作成する

TimerManager を使用するには，スケジュール元となる EJB またはサーブレットに，属性に定義した TimerManager の JNDI 名のルックアップ，および TimerManager の処理のスケジューリングを実装します。

属性に定義した TimerManager の JNDI を使用したルックアップ

属性に定義した TimerManager の JNDI 名をルックアップして TimerManager を取得します。ルックアップには java:comp/env を使用します。TimerManager を取得する例を次に示します。

```
InitialContext ic = new InitialContext();
TimerManager tm = (TimerManager)ic.lookup
                    (”java:comp/env/timer/MyTimer”);
```

TimerManager の処理のスケジューリング

TimerManager の処理のスケジューリングは，TimerManager の schedule メソッドを呼び出して実行します。TimerManager 処理のスケジューリングの例を次に示します。

```
InitialContext ctx = new InitialContext();
TimerManager mgr = (TimerManager)
              ctx.lookup(”java:comp/env/timer/MyTimer”);
TimerListener listener = new MyTimerListener();
mgr.schedule(listener, 1000*60,1000*10);
mgr.stop();
```

(4) TimerManager を使用する J2EE アプリケーションをコンパイルする

TimerManager を使用する J2EE アプリケーションをコンパイルする場合は，次の JAR ファイルを含めてください。

＜ Application Server のインストールディレクトリ＞¥CC¥lib¥ejbserver.jar

20.4　WorkManager を使用した非同期スレッド処理

この節では，WorkManager を使用した非同期スレッド処理について説明します。

この節の構成を次に示します。

表 20-14　この節の構成（WorkManager を使用した非同期スレッド処理）

分類	タイトル	参照先
解説	デーモン Work と非デーモン Work	20.4.1
	非デーモン Work で使用するスレッドプールとキューについて	20.4.2
	WorkManager，デーモン Work および非デーモン Work のライフサイクル	20.4.3
実装	WorkManager を使用したアプリケーションの開発	20.4.4
設定	実行環境での設定	20.4.5

注　「運用」および「注意事項」について，この機能固有の説明はありません。

WorkManager を使用した非同期スレッド処理では，Java EE 環境で，スレッドの非同期処理を実行できます。バックグラウンドでは，コンテナで管理されたスレッドを使用するため，安全にタスクを実行できます。

非同期で実行する処理は，Work で実装します。スケジュール元となる EJB やサーブレットで WorkManager の schedule メソッドを実行することで，Work に実装した処理がスケジューリングされます。また，WorkManager の schedule メソッドから返された WorkItem を使用することで，スケジュールの状態を確認できます。

WorkManager を使用するには，EJB 属性やサーブレット属性の<resource-ref>タグに，WorkManager に関する情報を定義します。EJB やサーブレットは，デプロイ時に<res-ref-name>タグに定義した名前でルックアップして WorkManager を使用します。

20.4.1　デーモン Work と非デーモン Work

WorkManager では，デーモン Work（長寿命 Work）と非デーモン Work（短寿命 Work）の 2 種類の Work を作成できます。それぞれの Work の概要を次に示します。

- **デーモン Work（長寿命 Work）**

  デーモン Work は，schedule メソッドの実行時に作成されて，サーブレットや EJB のリクエスト処理が終了しても実行され続けます。また，WorkManager の終了時に破棄されます。デーモン Work は，スレッドプールのスレッドではなく，常に新しく作成されたスレッドで実行されます。

- **非デーモン Work（短寿命 Work）**

  非デーモン Work は，schedule メソッドの実行時に作成されて，run メソッドの処理の終了時に破棄されます。非デーモン Work は，スレッドプールで管理されたスレッドおよびキューを使用します。

20.4.2　非デーモン Work で使用するスレッドプールとキューについて

非デーモン Work は，スレッドプールとキューを使用して処理されます。処理で使用されるスレッドプールおよびキューは，DD で定義した WorkManager 単位で作成されます。なお，スレッドプールにプールできるスレッドの最大サイズを設定します。次に，非デーモン Work がスケジュールされたとき，プールできるスレッドの最大サイズと，プール内のスレッド数の関係と動作について説明します。

- プール内のスレッドが，スレッドプールの最大スレッド数より少ない場合
  新しいスレッドを作成して，非デーモン Work を実行します。なお，スレッドは，スレッドプールの中に空きスレッドがあるかどうかに関係なく，生成されます。
- プール内に，スレッドプールの最大スレッド数で設定した数だけスレッドがある場合
  スレッドプールの中の空きスレッドを使用して非デーモン Work を実行します。空きスレッドがないときには，スケジュールされた非デーモン Work はキューに格納されます。キューに格納された非デーモン Work は，空きスレッドができると実行されます。

スレッドプールの最大スレッド数は，デフォルトで 10 となっています。最大スレッド数を変更したい場合は，「20.4.5　実行環境の設定」を参照してください。なお，キューサイズには制限はありません。

ポイント

WorkManager を停止しようとするときには，実行中の WorkManager およびキューに格納された WorkManager の処理がすべて終了されてから停止処理が開始されます。また，キュー格納時に WorkManager が停止された場合でも，キューに格納された WorkManager は実行されます。

20.4.3　WorkManager，デーモン Work および非デーモン Work のライフサイクル

ここでは，WorkManager，デーモン Work および非デーモン Work のライフサイクルについて説明します。

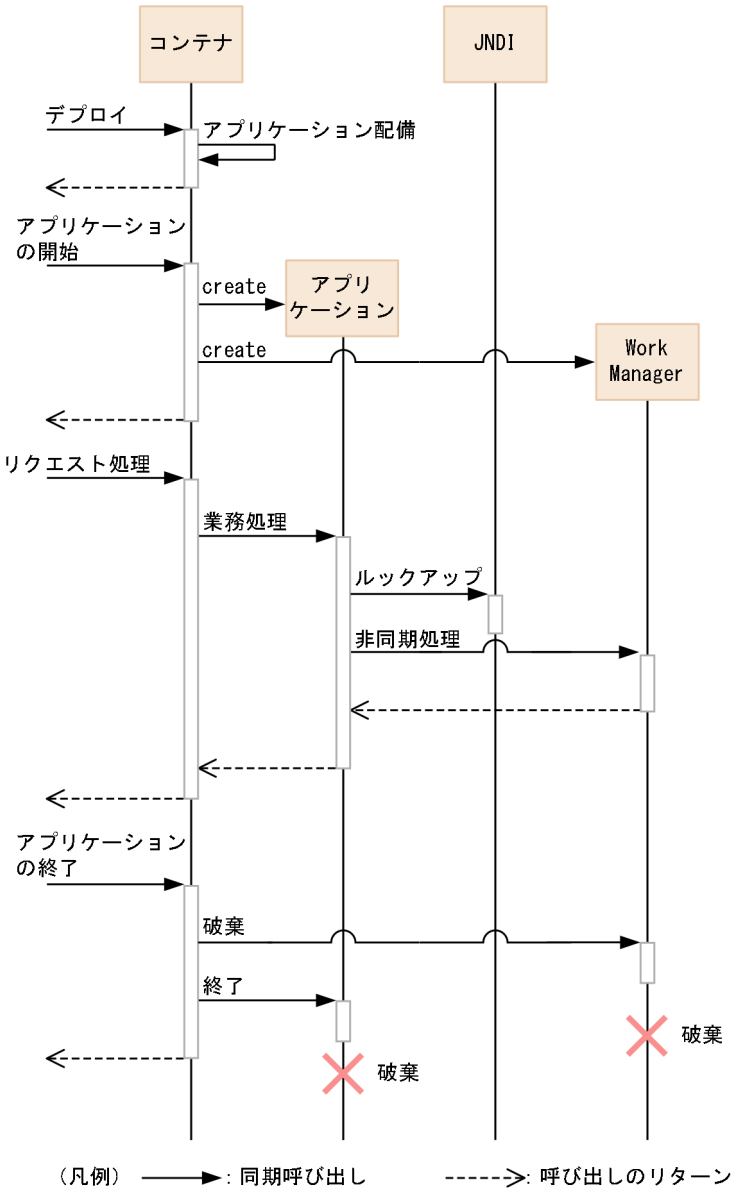
(1)　WorkManager のライフサイクル

WorkManager は，アプリケーション開始時に作成されます。アプリケーション中でルックアップされると，アプリケーション開始時に作成された WorkManager が返されます。複数回ルックアップされても，アプリケーション開始時に作成された同じ WorkManager が呼び出されます。WorkManager は，アプリケーションの停止時に破棄されます。

また，WorkManager は永続化されません。そのため，JavaVM が終了した場合，作成された WorkManager およびスケジュールされた非同期処理は破棄されます。

WorkManager のライフサイクルを次の図に示します。

図 20-7　WorkManager のライフサイクル

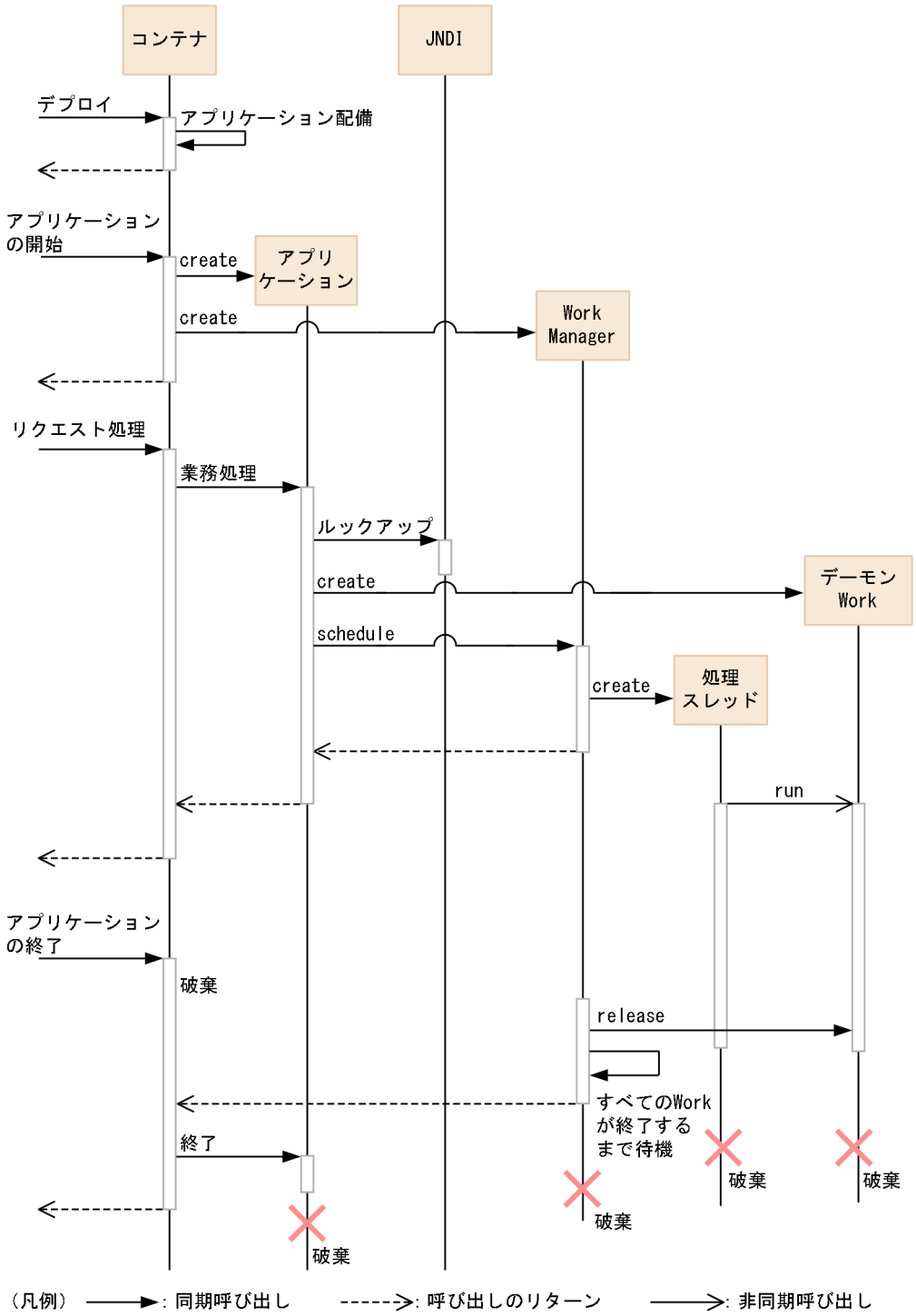


(2)　デーモン Work のライフサイクル

デーモン Work は，schedule メソッドの実行時に作成されます。また，WorkManager の停止時（WorkManager に対応するアプリケーションの停止時）に破棄されます。WorkManager の停止時には，WorkManager はデーモン Work の release メソッドを実行したあと，すべてのデーモン Work が終了するまで待機します。

デーモン Work のライフサイクルを次の図に示します。

図 20-8　デーモン Work のライフサイクル

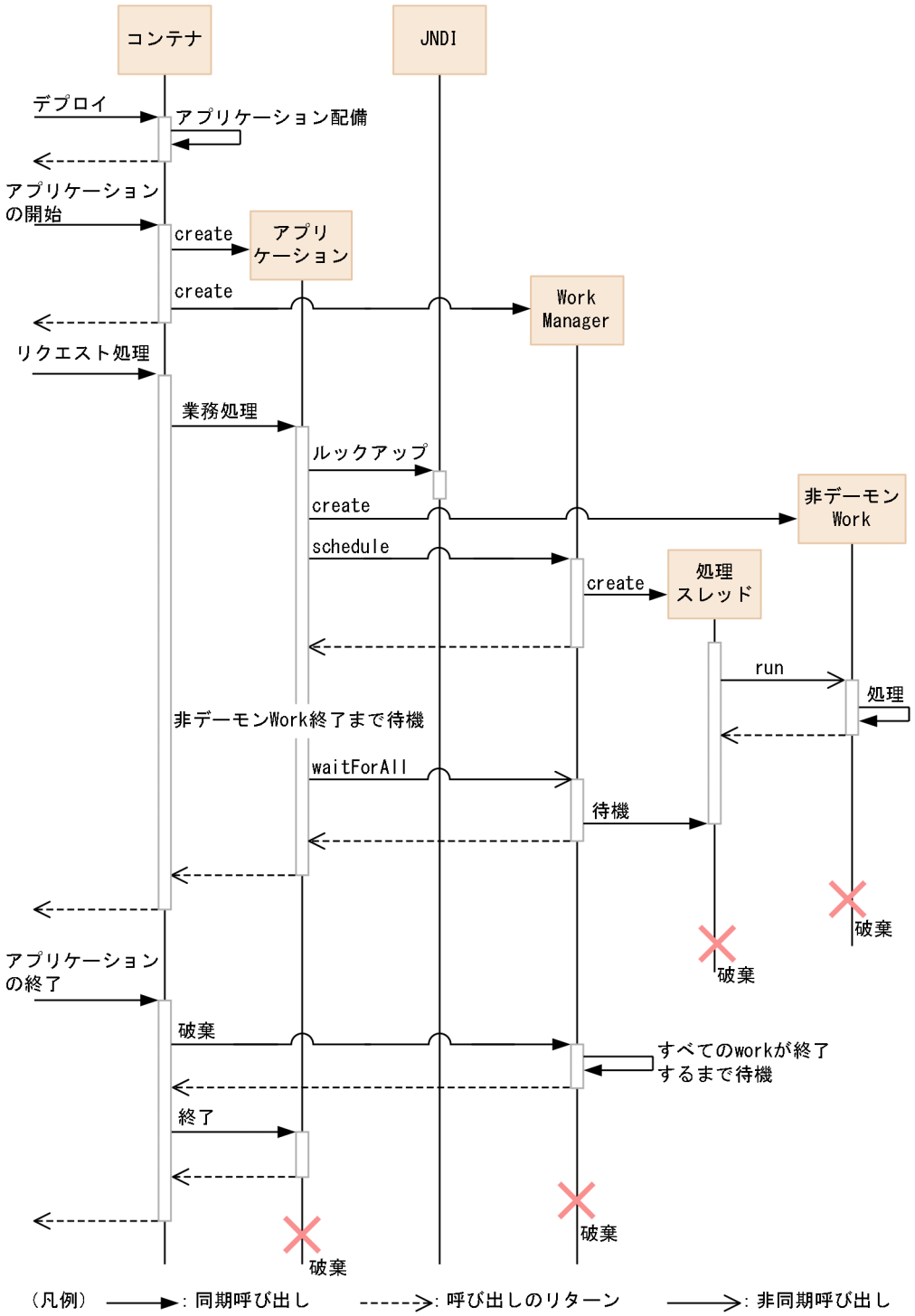


(3)　非デーモン Work のライフサイクル

非デーモン Work は，schedule メソッドの実行時に作成されます。また，run メソッドの処理が完了すると終了します。非デーモン Work の実行中の場合またはキューで実行待ちの場合に WorkManager を停止したとき（対応するアプリケーションを停止したとき），非デーモン Work が停止するまで待機してから終了します。

非デーモン Work のライフサイクルを次の図に示します。

図 20-9　非デーモン Work のライフサイクル



20.4.4　WorkManager を使用したアプリケーションの開発

ここでは，WorkManager を使用したアプリケーションの開発について説明します。

WorkManager を使用する場合の，アプリケーションを構成するコンポーネントの使用可否を次の表に示します。

表 20-15　WorkManager を使用する場合のアプリケーションを構成するコンポーネントの使用可否

コンポーネント				使用可否
EJB クライアント				×
リソースアダプタ				×
JavaBeans リソース				×
サーブレット／JSP※				○
EJB	Stateless Session Bean	EJB2.1 以前	CMT	○
			BMT	○
		EJB3.0		×
	Stateful Session Bean			×
	Entity Bean			×
	Message-driven Bean			×

（凡例）

○：使用できる

×：使用できない

注※　サーブレットリスナやフィルタでも使用できます。

WorkManager を使用するアプリケーションの開発の流れは次のとおりです。

1. スケジュール元となる EJB またはサーブレットの属性を定義する
2. Work およびリスナに実行する処理を実装する
3. スケジュール元となる EJB またはサーブレットを作成する
4. WorkManager を使用する J2EE アプリケーションをコンパイルする

それぞれの作業の詳細を次に示します。

(1)　スケジュール元となる EJB またはサーブレットの属性を定義する

WorkManager を使用する EJB またはサーブレットの属性を DD に定義します。属性は，EJB またはサーブレットの属性定義ファイルで定義します。アノテーションで定義することはできません。

WorkManager を使用するために定義する必要がある属性を次の表に示します。

表 20-16　WorkManager を使用するために定義する必要がある属性

タグ名	説明
＜ルートタグ＞	－

タグ名		説明
├	<description>	任意で設定してください。
├	<res-ref-name>	JNDI ENC 名（JNDI ルックアップに使用する名前）を指定してください。
├	<res-type>	次の内容を設定してください。 「commonj.work.WorkManager」
├	<res-auth>	設定した値は無視されます。
├	<res-sharing-scope>	「Shareable」を設定してください。ただし，「Unshareable」を設定しても，「Shareable」設定時と同様に動作します（WorkManager はアプリケーション開始時に作成されて，ルックアップ時には同じ WorkManager が返ります）。
├	<mapped-name>	設定した値は無視されます。
├	<injection-target>	設定した値は無視されます。
└	<linked-to>	設定した値は無視されます。

サーブレットで WorkManager を使用する場合の web.xml の定義例を次に示します。

```
<web-app>
  <display-name>WorkManagerSample</display-name>
  <servlet>
    <servlet-name>SampleServlet</servlet-name>
    <display-name>SampleServlet</display-name>
    <servlet-class>SampleServlet</servlet-class>
  </servlet>
  <resource-ref>
    <res-ref-name>wm/MyWorkManager</res-ref-name>
    <res-type>commonj.work.WorkManager</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
```

```
  </resource-ref>
</web-app>
```

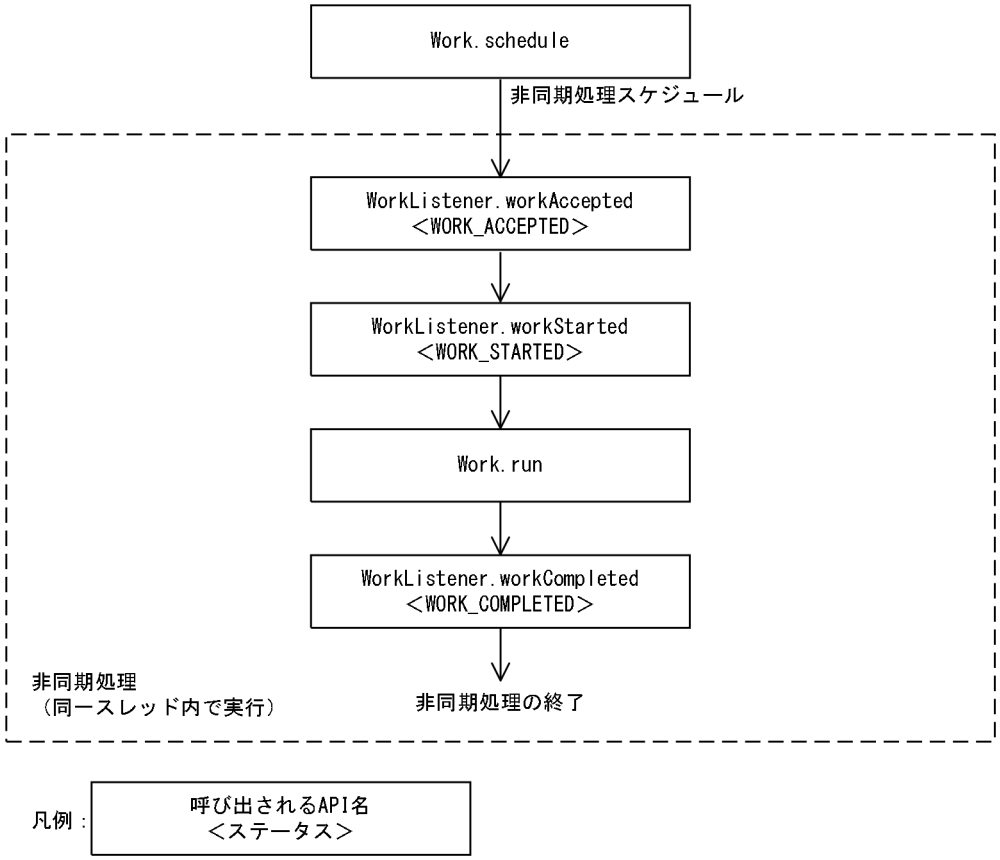
WorkManager は，アプリケーションの開始時に，属性の定義に従って自動で作成されます。属性で定義した数の WorkManager が作成されます。

(2) Work およびリスナに実行する処理を実装する

WorkManager を使用するには，実行する処理を実装した Work およびリスナを作成する必要があります。インタフェースには，処理の実体である run メソッドを持った Work インタフェースと，処理の受付，開始，終了などのタイミングで処理を実行するための WorkListener インタフェースがあります。このうち，Work は必ず実装してください。API の詳細については，Timer and Work Manager for Application Servers の API の仕様を参照してください。

WorkListener インタフェースの API が呼び出される流れとステータスの遷移を次に示します。

図 20–10　WorkListener インタフェースの API が呼び出される流れとステータスの遷移



WorkListener のメソッドおよび Work.run メソッドは，同一スレッド内で呼び出されます。そのため，それぞれのメソッドは，共通の Java EE コンテキストを使用できます。

デーモン Work と非デーモン Work に実装する処理の流れと実装例，および WorkListener の実装例を次に示します。

デーモン Work に実装する処理の流れと実装例

デーモン Work を使用するには，isDaemon メソッドが true を返すように Work を実装します。

WorkManager が終了すると，コンテナはデーモン Work を停止するために，release メソッドを実行します。そのため，release メソッドが実行されたら run メソッドの処理が終了するように実装してください。release メソッドが適切に実装されていない場合，WorkManager 停止時にデーモン Work が停止しないで，無限待ちになることがあるので注意してください。

デーモン Work の実装例を次に示します。

```
public class MyWork implements Work {
  private String name;
  private boolean isLoopContinue = true;
  public MyWork() {}

  public void release() {
    isLoopContinue = false;
  }

  public boolean isDaemon() {
    return true;
  }

  public void run() {
    while (isLoopContinue) {
      System.out.println("DaemonWork is executed");
   try {
     Thread.sleep(10000);
   } catch(InterruptedException e) {}
 }
  }
  public String toString() {
    return name;
  }
}
```

非デーモン Work に実装する処理の流れと実装例

非デーモン Work を使用するには，isDaemon メソッドが false を返すように Work を実装します。

非デーモン Work の処理は，スケジューリングした EJB やサーブレットの処理中に終了する必要があります。そのため，スケジュールした Work が終了するのを待って，EJB またはサーブレットの処理を終了するように実装してください。スケジュールした Work の終了を待つには，waitForAll メソッドまたは waitForAny メソッドを使用します。スケジュールした Work が終了しないうちに EJB やサーブレットの処理を終了した場合，スケジュールした EJB やサーブレットのライフサイクルを超えて Work の処理が実行されてしまいます。非デーモン Work がスケジュールしたリクエストのライフサイクルを超えて実行されないように，必ず waitForAll メソッドなどを使用してユーザプログラム中で処理を終了してください。

非デーモン Work の実装例を次に示します。

```
public class MyWork implements Work {
         private String name;
         private String data;
         public MyWork(String name) {
                  this.name = name;
```

```
}

          public void release() {}

          public boolean isDaemon() {
                    return false;
          }

          public void run() {
                    data = ”Hello, World. name=” + name;
          }

          public String getData() {
                    return data;
          }

          public String toString() {
                    return name;
          }
}
```

WorkListener の実装例

WorkListener の実装例を次に示します。

```
public class ExampleListener implements WorkListener {
    public void workAccepted(WorkEvent we) {
        System.out.println(”Work Accepted”);
    }

    public void workRejected(WorkEvent we) {
        System.out.println(”Work Rejected”);
    }

    public void workStarted(WorkEvent we) {
        System.out.println(”Work Started”);
    }

    public void workCompleted(WorkEvent we) {
        System.out.println(”Work Completed”);
    }
}
```

(3) スケジュール元となる EJB またはサーブレットを作成する

WorkManager を使用するには，スケジュール元となる EJB またはサーブレットに，属性に定義した WorkManager の JNDI 名のルックアップ，および WorkManager の処理のスケジューリングを実装します。

属性に定義した WorkManager の JNDI 名

属性に定義した WorkManager の JNDI 名をルックアップして WorkManager を取得します。ルックアップには java:comp/env を使用します。WorkManager を取得する例を次に示します。

```
InitialContext ic = new InitialContext();
WorkManager tm = (WorkManager)ic.lookup
                    ("java:comp/env/wm/MyWorkManager");
```

WorkManager の処理のスケジューリング

WorkManager の処理のスケジューリングは，WorkManager の schedule メソッドを呼び出して実行します。

複数の非デーモン Work をスケジューリングしたあと，すべての Work の終了を待つプログラムの例を次に示します。

```
MyWork work1 = new MyWork();
MyWork work2 = new MyWork();
InitialContext ctx = new InitialContext();
WorkManager mgr = (WorkManager) ctx.lookup("java:comp/env/wm/MyWorkManager");
WorkItem wi1 = mgr.schedule(work1, new ExampleListener());
WorkItem wi2 = mgr.schedule(work2);
Collection coll = new ArrayList();
coll.add(wi1);
coll.add(wi2);
mgr.waitForAll(coll, WorkManager.INDEFINITE);

System.out.println("work1 data: " + work1.getData());
System.out.println("work2 data: " + work2.getData());
```

(4) WorkManager を使用する J2EE アプリケーションをコンパイルする

WorkManager を使用する J2EE アプリケーションをコンパイルする場合は，次の JAR ファイルを含めてください。

＜ Application Server のインストールディレクトリ＞¥CC¥lib¥ejbserver.jar

20.4.5 実行環境の設定

非デーモン Work で使用するスレッドプールの最大スレッド数をデフォルト値の「10」から変更したい場合，J2EE サーバの設定が必要です。

J2EE サーバの設定は，簡易構築定義ファイルで実施します。スレッドプールの最大スレッド数の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。簡易構築定義ファイルでの設定を次の表に示します。

表 20-17 簡易構築定義ファイルでのスレッドプールの最大スレッド数を変更するための定義

指定するパラメタ	設定内容
ejbserver.commonj.WorkManager.non_daemon_work_threads	非デーモン Work で使用するスレッドプールの最大スレッド数を設定します。1～65535 の範囲※で設定してください。 デフォルト値は 10 です。

注※ 範囲以外の数値が指定された場合は，メッセージ KDJE34510-W が表示され，デフォルト値が適用されます。

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.3　簡易構築定義ファイル」を参照してください。

付録

付録 A　リダイレクタ機能のインストール

リダイレクタ機能をインストールする方法には，次の種類があります。

- 新規インストール

  リダイレクタ機能がインストールされていない環境に，リダイレクタ機能をインストールすることです。新規インストールでは，リダイレクタのユーザ定義は，インストーラがデフォルトのユーザ定義を作成します。

- 更新インストール

  リダイレクタ機能がインストール済みの環境に，リダイレクタ機能（同一製品）をインストールすることです。更新インストールでは，リダイレクタのユーザ定義は，引き継ぎます。

付録 A.1　リダイレクタ機能をインストールする（Windows の場合）

アプリケーションサーバに接続して，リダイレクタ機能をインストールします。インストールでは，製品の提供媒体であるインストーラを使用します。インストーラの使用方法については，製品の提供媒体のドキュメントを参照してください。

インストールの手順を次に示します。インストール作業には Administrator 権限が必要です。インストールするときの注意事項については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 I.1　Application Server をインストールおよびアンインストールするときの注意事項」を参照してください。

手順

1. 製品の提供媒体を CD-ROM ドライブにセットします。

   ［日立総合インストーラ］ダイアログに，「選択されたソフトウェアをインストールします。」と表示されます。

   ［日立総合インストーラ］ダイアログが表示されない場合，エクスプローラを使用して，CD-ROM ディレクトリの「HCD_INST.EXE」をダブルクリックしてください。

   注意事項

   製品の提供媒体をハードディスクにコピーしてからインストールする場合は，コピー先のパス名に";"（セミコロン），"."（ピリオド），および全角文字を含まないようにしてください。

2. Application Server を選択した状態で，［インストール実行］ボタンをクリックします。

   ［インストール処理開始の確認 － 日立総合インストーラ］ダイアログに，「インストールを開始します。よろしいですか？」と表示されます。

3. ［OK］ボタンをクリックします。

   ［uCosminexus Application Server セットアッププログラムへようこそ］ダイアログが表示されます。

4. [次へ] ボタンをクリックします。
   [インストール先の選択] ダイアログが表示されます。

5. 必要に応じて [インストール先のフォルダ] を選択して，[次へ] ボタンをクリックします。
   [機能の選択] ダイアログが表示されます。

6. 次に示すどちらかの左にあるボタンをクリックします。
   - 新規インストールの場合：[Redirector - Redirector をインストールするためのセットアップです。セットアップ可能なすべてのオプションをカスタマイズする事ができます。]
   - 更新インストールの場合：[Redirector - Redirector をインストールするためのセットアップです。再インストールするプログラム機能を選択できます。]

   [プログラムの選択] ダイアログが表示されます。

7. インストールする構成ソフトウェア（プログラム）を選択して，[次へ] ボタンをクリックします。
   次に示す構成ソフトウェアから選択します。
   - Component Container - Redirector（リダイレクタ機能）
   - HTTP Server（Web サーバ）
   - Performance Tracer（パフォーマンストレーサ）

   [ユーザ情報] ダイアログが表示されます。

8. [ユーザ名] および [会社名] を入力して [次へ] ボタンをクリックします。
   [プログラム フォルダの選択] ダイアログが表示されます。

9. 必要に応じて [プログラムフォルダ] を変更して，[次へ] ボタンをクリックします。
   [インストールの開始] ダイアログが表示されます。

10. 設定した内容を確認して，問題がなければ [次へ] ボタンをクリックします。
    インストールが開始されます。インストールが完了すると，[セットアップの完了] ダイアログが表示されます。

11. [完了] ボタンをクリックします。
    OS を再起動するかどうかを確認する画面が表示されます。

12. [はい] ボタンをクリックします。
    OS が再起動し，リダイレクタ機能のインストールが完了します。

付録 A.2　リダイレクタ機能をインストールする（UNIX の場合）

アプリケーションサーバに接続して，リダイレクタ機能をインストールします。

インストールでは，製品の提供媒体である PP インストーラを使用します。

インストールの手順を次に示します。インストール作業には root 権限が必要です。インストールするときの注意事項については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 I.1 Application Server をインストールおよびアンインストールするときの注意事項」を参照してください。

手順

1. アプリケーションサーバに root 権限（スーパーユーザ）でログインします。
2. PP インストーラ実行時の言語種別と，実行するターミナルの言語が一致しているかどうかを確認し，一致していない場合は一致させます。
3. 製品の提供媒体を CD-ROM ドライブにセットします。
4. 製品の提供媒体が CD-ROM の場合は，CD-ROM ファイルシステムをマウントします。

   コマンドの実行例を次に示します。下線部には，デバイススペシャルファイル名，および CD-ROM ファイルシステムのマウントディレクトリ名を指定します。なお，これらの名称は，OS，ハードウェア，および環境によって異なります。

   （AIX の実行例）

   ```
   mount -r -v cdrfs /dev/cd0 /cdrom
   ```

   （Linux の実行例）

   ```
   mount -r -o mode=0544  /dev/cdrom  /mnt/cdrom
   ```

5. セットアッププログラムを起動します。

   コマンドの実行例を次に示します。下線部には，CD-ROM ファイルシステムのマウントディレクトリ名を指定します。

   （AIX の実行例）

   ```
   /cdrom/aix/setup /cdrom
   ```

   （Linux の実行例）

   ```
   /mnt/cdrom/x64lin/setup /mnt/cdrom
   ```

   CD-ROM セットアッププログラムによって，PP インストーラと常駐プロセス自動起動プログラムがハードディスク上にインストールされ，PP インストーラが自動的に起動されます。

   注意事項

   CD-ROM のディレクトリ名やファイル名は，マシン環境によって記述した内容と見え方が異なることがあります。ls コマンドで確認し，表示されたファイル名をそのまま入力してください。

6. PP インストーラのメインメニューで，［I］キーを押します。

   PP インストール画面が表示されます。

7. プログラムにカーソルを移動させ，［スペース］キーを押します。

製品の共通モジュールと，次に示す構成ソフトウェアから必要なものを選択します。

- Component Container - Redirector（リダイレクタ機能）
- HTTP Server（Web サーバ）
- Performance Tracer（パフォーマンストレーサ）

選択したプログラムの左側には<@>が表示されます。なお，プログラムを選択してインストールすることもできます。

注意事項

製品の共通モジュールは，製品によって表示されるプログラム名が異なります。

8. **選択したプログラムの左側に<@>が表示されていることを確認して，［I］キーを押します。**
   画面の最下行に「Install PP? (y: install, n: cancel)==>」メッセージが表示されます。

9. **［y］キーまたは［Y］キーを押します。**
   インストールが開始されます。
   ［n］キーまたは［N］キーを押すと，インストールが中止されて PP インストール画面に戻ります。

10. **インストール終了を示すメッセージが出力されたら，［Q］キーを押します。**
    PP インストーラのメインメニューに戻ります。

11. **必要に応じて，PP インストーラのメインメニューで［L］キーを押して，インストール済みのプログラムを確認します。**
    PP 一覧表示画面が表示されます。［P］キーを押すと，インストール済みのプログラム一覧が「/tmp/hitachi_PPLIST」に出力されます。［Q］キーを押すと，PP インストーラのメインメニューに戻ります。

12. **PP インストーラのメインメニューで，［Q］キーを押します。**
    リダイレクタ機能のインストールが完了します。

注意事項

リダイレクタ機能を使用するため HTTP Server を起動する際は，次の環境変数を設定する必要があります。

LD_LIBRARY_PATH=/opt/Cosminexus/common/lib

付録 B　推奨手順以外の方法でパフォーマンスチューニングをする場合のチューニングパラメタ

ここでは，推奨手順以外の方法でパフォーマンスチューニングをする場合のチューニングパラメタについて説明します。

付録 B.1　タイムアウトを設定するチューニングパラメタ（推奨手順以外の方法）

ここでは，タイムアウトの設定で使用するチューニングパラメタの設定個所についてまとめて示します。

(1)　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト

Web サーバ連携の場合は，Web サーバ単位に設定します。インプロセス HTTP サーバの場合は，J2EE サーバ単位に設定します。

表 B-1　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ（推奨手順以外の方法）

使用する Web サーバ	設定方法	設定個所
Web サーバ連携	運用管理ポータル（HTTP Server の場合）	［Web サーバの設定］画面の「項目ごとに設定します。」の「追加ディレクティブ」の Timeout ディレクティブ
		［Web サーバの設定］画面の「設定ファイルの内容を直接設定します。」の「設定ファイルの内容」の Timeout ディレクティブ
	ファイル編集※	• HTTP Server の場合 httpsd.conf の Timeout ディレクティブ • Microsoft IIS の場合 isapi_redirect.conf の receive_client_timeout キー
インプロセス HTTP サーバ	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Web クライアントとの接続設定」の「通信タイムアウト」の「リクエスト受信」
		［通信・スレッド制御に関する設定］画面の「Web クライアントとの接続設定」の［通信タイムアウト］の［リクエスト送信］
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.receive_timeout キー
		usrconf.properties の webserver.connector.inprocess_http.send_timeout キー

注※　HTTP Server の定義ファイルである httpd.conf を編集して設定します。

(2) リダイレクタ側で設定するWebコンテナへのデータ送信のタイムアウト

リダイレクタ側で設定するタイムアウトのチューニングパラメタについて，次の表に示します。なお，これらのチューニングパラメタは，Webサーバ連携の場合だけ指定できます。

表B-2 リダイレクタ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

Webサーバの種類	設定方法	設定個所
HTTP Server	運用管理ポータル	［リダイレクタの設定］画面の「オプション」の「リクエスト送信コネクション確立タイムアウト時間」
	ファイル編集	mod_jk.conf の JkConnectTimeout パラメタ
Microsoft IIS	ファイル編集	isapi_redirect.conf の connect_timeout キー
HTTP Server	運用管理ポータル	［リダイレクタの設定］画面の「オプション」の「リクエスト送信タイムアウト時間」
	ファイル編集	mod_jk.conf の JkSendTimeout パラメタ
Microsoft IIS	ファイル編集	isapi_redirect.conf の send_timeout キー

(3) リダイレクタ側で設定するWebコンテナからのデータ受信のタイムアウト

リダイレクタのワーカ定義単位で設定します。リダイレクタ側で設定するタイムアウトのチューニングパラメタの設定方法と設定個所を次の表に示します。

表B-3 リダイレクタ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

設定方法	設定個所
運用管理ポータル	［ワーカの設定］画面の「リダイレクタで再利用するワーカとのコネクション数の定義」の「通信タイムアウト」
ファイル編集	workers.properties の worker.<ワーカ名>.receive_timeout キー

Webサーバ連携の場合だけ指定できます。

(4) Webコンテナ側で設定するリダイレクタからのデータ受信のタイムアウト

J2EEサーバ単位で設定します。Webコンテナ側で設定するタイムアウトのチューニングパラメタを次の表に示します。

表B-4 Webコンテナ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

設定方法	設定個所
運用管理ポータル	［Webコンテナの設定］画面の「Webサーバとの接続」の「タイムアウト時間」
ファイル編集	usrconf.properties の webserver.connector.ajp13.receive_timeout キー

Web サーバ連携の場合だけ指定できます。

(5) Web コンテナ側で設定するリダイレクタへのデータ受信のタイムアウト

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタを次の表に示します。

表 B-5 Web コンテナ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

設定方法	設定個所
運用管理ポータル	［Web コンテナの設定］画面の「Web サーバとの接続」の「レスポンス送信タイムアウト時間」
ファイル編集	usrconf.properties の webserver.connector.ajp13.send_timeout キー

Web サーバ連携の場合だけ指定できます。

(6) EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト

J2EE サーバ単位，EJB クライアントアプリケーション単位または API による呼び出し単位に設定します。

EJB クライアント側で設定するタイムアウトのチューニングパラメタ（RMI-IIOP 通信によるリモート呼び出し）を次の表に示します。

表 B-6 EJB クライアント側で設定するタイムアウトのチューニングパラメタ（RMI-IIOP 通信によるリモート呼び出し）（推奨手順以外の方法）

単位	設定方法	設定個所
J2EE サーバ単位	運用管理ポータル	［EJB コンテナの設定］画面の「サーバとの接続」の「タイムアウト時間」
	ファイル編集	usrconf.properties の ejbserver.rmi.request.timeout キー

EJB クライアントアプリケーション単位の設定，および API 単位の設定については，推奨手順の場合と違いがありません。

EJB クライアント側で設定するタイムアウトのチューニングパラメタ（ネーミングサービス呼び出し）を次の表に示します。

表 B-7　EJB クライアント側で設定するタイムアウトのチューニングパラメタ（ネーミングサービス呼び出し）（推奨手順以外の方法）

単位	設定方法	設定個所
J2EE サーバ単位	運用管理ポータル	［ネーミングの設定］画面の「利用するネーミングサービスの設定」の「タイムアウト時間」
	ファイル編集	usrconf.properties の ejbserver.jndi.request.timeout キー

EJB クライアントアプリケーション単位の設定については，推奨手順の場合と違いがありません。

(7)　EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト

J2EE サーバ単位，EJB クライアントアプリケーション単位または API による呼び出し単位に設定します。

なお，このタイムアウトの設定値には，「付録 B.1(6)　EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト」で指定した設定値と同じ値が引き継がれます。

(8)　EJB コンテナ側で設定するデータベースのトランザクションタイムアウト（DB Connector を使用した場合）

J2EE サーバ単位，Enterprise Bean，インタフェース，メソッド単位（CMT の場合），または API による呼び出し単位（BMT の場合）に設定します。

トランザクションタイムアウトのチューニングパラメタを次の表に示します。

表 B-8　トランザクションタイムアウトのチューニングパラメタ（推奨手順以外の方法）

単位	設定方法	設定個所
J2EE サーバ単位	運用管理ポータル	［トランザクションの設定］画面の「トランザクションに関する設定 」の「タイムアウト時間」
	ファイル編集	usrconf.properties の ejbserver.jta.TransactionManager.defaultTimeOut キー

Enterprise Bean，インタフェース，メソッド単位（CMT の場合）および API 単位（BMT の場合）のチューニングパラメタについては，推奨手順の場合と違いがありません。

(9)　データベースのタイムアウト

データベースのタイムアウトを設定するチューニングパラメタについては，推奨手順の場合と違いがありません。

付録 B.2　Web アプリケーションの動作を最適化するためのチューニングパラメタ（推奨手順以外の場合）

ここでは，Web アプリケーションの動作を最適化するために使用するチューニングパラメタの設定個所についてまとめて示します。

(1)　静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ

静的コンテンツと Web アプリケーションの配置の切り分けは，Web サーバの動作を定義するファイルのパラメタとして指定します。設定個所，ファイルおよびパラメタは，使用する Web サーバの種類によって異なります。

Web サーバの種類ごとの設定方法および設定個所を次に示します。

表 B-9　静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ（推奨手順以外の場合）

使用する Web サーバ	Web サーバの種類	設定方法	設定個所
Web サーバ連携（リダイレクタモジュールを使用した切り分け）	HTTP Server	運用管理ポータル	［論理 Web サーバの定義］画面の「マッピング定義」
		ファイル編集	mod_jk.conf のマッピング定義（JkMount パラメタ）
	Microsoft IIS	ファイル編集	uriworkermap.properties
インプロセス HTTP サーバ（リバースプロキシモジュールを使用した切り分け）	HTTP Server	運用管理ポータル	［Web サーバの設定］画面の「項目ごとに設定します。」の「追加ディレクティブ」の ProxyPass ディレクティブ
			［Web サーバの設定］画面の「設定ファイルの内容を直接設定します。」の「設定ファイルの内容」の ProxyPass ディレクティブ
		ファイル編集	httpsd.conf の ProxyPass ディレクティブ※

注※　httpsd.conf の詳細については，マニュアル「HTTP Server」を参照してください。

(2)　静的コンテンツをキャッシュするためのチューニングパラメタ

静的コンテンツをキャッシュするためのチューニングパラメタについて説明します。これらのチューニングパラメタは，Web コンテナ単位または Web アプリケーション単位に設定します。

Web コンテナ単位に設定するチューニングパラメタの設定方法および設定個所について，次の表に示します。

表 B-10　静的コンテンツをキャッシュするためのチューニングパラメタ（Web コンテナ単位で設定する項目）（推奨手順以外の場合）

設定項目	設定方法	設定個所
静的コンテンツのキャッシュを使用するかどうかの選択	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「静的コンテンツキャッシュ機能」
	ファイル編集	usrconf.properties の webserver.static_content.cache.enabled キー
Web アプリケーション単位のメモリサイズの上限値の設定	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「キャッシュサイズ」
	ファイル編集	usrconf.properties の webserver.static_content.cache.size キー
キャッシュする静的コンテンツのファイルサイズの上限値の設定	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「ファイルサイズ」
	ファイル編集	usrconf.properties の webserver.static_content.cache.filesize.threshold キー

Web アプリケーション単位に設定するチューニングパラメタについては，推奨手順の場合と違いがありません。

(3)　リダイレクタによってリクエストを振り分けるためのチューニングパラメタ

リダイレクタによってリクエストを振り分けるためのチューニングパラメタは，Web サーバの動作を定義するファイルのパラメタとして指定します。設定個所，ファイルおよびパラメタは，使用する Web サーバごとに異なります。

なお，この定義は，Web サーバ連携の場合だけできます。インプロセス HTTP サーバを使用している場合は定義できません。

Web サーバごとの設定方法および設定個所を次に示します。

表 B-11　リダイレクタによってリクエストを振り分けるためのチューニングパラメタ（推奨手順以外の場合）

Web サーバの種類	設定方法	設定個所
HTTP Server	運用管理ポータル	［論理 Web サーバの定義］画面の「マッピング定義」
	ファイル編集	mod_jk.conf のマッピング定義（JkMount パラメタ）
Microsoft IIS	ファイル編集	uriworkermap.properties

付録 B.3　Persistent Connection についてのチューニングパラメタ（推奨手順以外の方法）

Persistent Connection についてのチューニングパラメタについて説明します。

この項目は，Web フロントシステムの場合で，インプロセス HTTP サーバを使用するときにチューニングを検討してください。

表 B-12　Persistent Connection について設定するチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
Persistent Connection 数の上限値	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Persistent コネクション」の「上限値」
	usrconf.properties	webserver.connector.inprocess_http.persistent_connection.max_connections キー
リクエスト処理回数の上限値	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Persistent コネクション」の「リクエスト処理回数の上限値」
	usrconf.properties	webserver.connector.inprocess_http.persistent_connection.max_requests キー
タイムアウト	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Persistent コネクション」の「タイムアウト」
	usrconf.properties	webserver.connector.inprocess_http.persistent_connection.timeout キー

付録 C　エラーステータスコード

ここでは，Web コンテナ，リダイレクタ，およびインプロセス HTTP サーバが返すエラーステータスコードについて示します。

使用する Web サーバによって，エラーが発生する個所が異なります。エラーが発生する個所に応じたエラーステータスコードを参照してください。使用する Web サーバとエラーが発生する個所の対応について次の表に示します。

表 C-1　使用する Web サーバとエラーが発生する個所の対応

使用する Web サーバ	エラーが発生する個所		
	Web コンテナ	リダイレクタ	インプロセス HTTP サーバ
HTTP Server または Microsoft IIS	○	○	－
インプロセス HTTP サーバ	○	－	○

（凡例）○：エラーが発生する。　－：エラーは発生しない。

付録 C.1　Web コンテナが返すエラーステータスコード

クライアントから，存在しないリソースや例外が発生したサーブレットなどにアクセスがあると，Web コンテナはエラーステータスコードを返します。Web コンテナが返すエラーステータスコードと，エラーステータスコードを返す条件を次の表に示します。

表 C-2　Web コンテナが返すエラーステータスコードと条件

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	次のどれかに該当する場合，エラーステータスコード 400 が返ります。 • FORM 認証で使用するログインページとして指定されたリソースに対して，クライアントから直接リクエストを送信し，その結果表示されたログインページからユーザ認証に成功した場合 • 次の三つの条件をすべて満たしているアクセスの場合 1. HTTP のバージョンが"HTTP/1.0"のとき 2. アクセス対象となるサーブレットが javax.servlet.http.HttpServlet を継承しているとき 3. アクセス時の HTTP メソッドが該当するサーブレットでオーバーライドされていないとき • Content-Length ヘッダの値が 2147483647 より大きい，または 0 より小さいリクエストヘッダでアクセスされた場合 • Content-Length ヘッダの値が数値以外のリクエストヘッダでアクセスされた場合 • Content-Length ヘッダを複数含むリクエストヘッダでアクセスされた場合 • リクエスト URI を正規化できなかった場合

エラーステータスコード	エラーステータスコードを返す条件
401　Unauthorized	BASIC 認証を必要とするリソースに対して，次のようなアクセスがあった場合，エラーステータスコード 401 が返ります。 • 不正なユーザ名，またはパスワードでアクセスされた場合 • 認証情報を含まないでアクセスされた場合（Authorization ヘッダがないアクセス）。
403　Forbidden	次のどれかに該当する場合，エラーステータスコード 403 が返ります。 • BASIC 認証，または FORM 認証を必要とするリソースに対して，認可できないユーザ名でアクセスされた場合 • web.xml で，auth-constraint 要素に role-name 要素を指定しないで，すべてのアクセスを許可しないとするリソースにアクセスされた場合※1 • 静的コンテンツに対して，PUT または DELETE メソッドでアクセスされた場合 • web.xml の<transport-guarantee>要素で，INTEGRAL または CONFIDENTIAL が設定されているリソースに http でアクセスされた場合※2
404　Not Found	次のどちらかのアクセスがあった場合，エラーステータスコード 404 が返ります。 • 存在しないリソースにアクセスされた場合 • javax.servlet.UnavailableException が発生しているサーブレット，または JSP ファイルにアクセスされた場合※3
405　Method Not Allowed	次の三つの条件をすべて満たしているアクセスの場合，エラーステータスコード 405 が返ります。 • HTTP のバージョンが"HTTP/1.1"の場合 • アクセス対象となるサーブレットが javax.servlet.http.HttpServlet を継承している場合 • アクセス時の HTTP メソッドが該当するサーブレットでオーバーライドされていない場合
412　Precondition Failed	If-Match ヘッダ，または If-Unmodified-Since ヘッダで指定した条件に一致しない静的コンテンツへのアクセスの場合，エラーステータスコード 412 が返ります。
413　Request Entity Too Large	リクエストボディのサイズが上限値を超えた場合，エラーステータスコード 413 が返ります。
416　Requested Range Not Satisfiable	次のどれかに当てはまる不正な Range ヘッダの値を使用した静的コンテンツへのアクセスの場合，エラーステータスコード 416 が返ります。 • Range ヘッダの値が"byte"から始まっていない • 範囲定義に数字や"-"を使用していない • 指定範囲が妥当ではない
500　Internal Server Error	次のどれかに該当する場合，エラーステータスコード 500 が返ります。 • 例外が発生するサーブレットまたは JSP ファイルにアクセスされた場合※4 • コンパイルに失敗した JSP ファイルにアクセスされた場合 • 削除された静的コンテンツにアクセスされた場合※5 • 静的コンテンツへのアクセスで I/O エラーが発生した場合 • web.xml の定義が不正な状態で，<auth-constraint>要素で保護されたリソースにアクセスされた場合※6

エラーステータスコード	エラーステータスコードを返す条件
501　Not Implemented	静的コンテンツまたは javax.servlet.http.HttpServlet を継承したサーブレットに対して，GET，HEAD，POST，PUT，DELETE，OPTIONS，TRACE メソッド以外の HTTP メソッドでアクセスされた場合，エラーステータスコード 501 が返ります。
503　Service Unavailable	次のどれかに該当する場合，エラーステータスコード 503 が返ります。 • リクエストの実行待ちキューに空きがない場合※7 • javax.servlet.UnavailableException が発生しているサーブレットまたは JSP ファイルにアクセスされた場合※8 • 終了処理中の Web コンテナに対してアクセスされた場合 • 予期しないエラーまたは例外によって，異常な状態になった Web アプリケーションにアクセスされた場合

注※1　Web アプリケーションのバージョンが 2.4 以降の場合に該当します。

注※2　usrconf.properties の webserver.connector.redirect_https.port キーに転送先の https のポート番号を設定していない場合が該当します。

注※3　Web アプリケーションのバージョンが 2.4 以降の場合で，永久的に利用できないことを示す javax.servlet.UnavailableException が発生し，サーブレットおよび JSP ファイルで例外を catch していないときが該当します。

注※4　次のような場合が該当します。

- Web アプリケーションのバージョンが 2.4 以降の場合
  サーブレットまたは JSP で例外を catch していないとき
- Web アプリケーションのバージョンが 2.3 の場合
  web.xml の<error-page>タグまたは JSP ファイルの page ディレクティブでエラーページの指定がなく，サーブレットまたは JSP ファイルで例外を catch していないとき

注※5　Web アプリケーションのリロード機能，JSP ファイルの再コンパイル機能，または J2EE アプリケーションのリロード機能を使用しない場合が該当します。

注※6　web.xml で<auth-constraint>要素に<role-name>要素が定義され，<login-config>要素が定義されていない場合に該当します。この状態でアプリケーションを開始すると，KDJE39150-W の警告メッセージがコンソール画面，およびメッセージログに出力されます。

注※7　Web アプリケーション単位の同時実行スレッド数制御，または URL グループ単位の同時実行スレッド数制御を設定している場合が該当します。

注※8　次のような場合が該当します。

- Web アプリケーションのバージョンが 2.4 以降の場合
  一時的に利用できないことを示す，javax.servlet.UnavailableException が発生し，サーブレットまたは JSP で例外を catch していないとき
- Web アプリケーションのバージョンが 2.3 の場合
  web.xml の<error-page>タグまたは JSP ファイルの page ディレクティブでエラーページの指定がなく，サーブレットまたは JSP ファイルで例外を catch していないとき

付録 C.2　リダイレクタが返すエラーステータスコード

Web コンテナとのデータ送受信時にタイムアウトが発生したり，定義ファイルに記載誤りがあったりすると，リダイレクタはエラーステータスコードを返します。リダイレクタが返すエラーステータスコードと，エラーステータスコードを返す条件を，Web サーバの種類ごとに表に示します。

表 C-3　リダイレクタが返すエラーステータスコードと発生条件（HTTP Server の場合）

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	次のどれかに該当する場合，エラーステータスコード 400 が返ります。 • リクエストの Host ヘッダのポート番号が不正な場合 • リクエストのメソッドが POST ではない場合※1 • リクエストに Content-Length ヘッダがない（POST リクエストである場合，ボディがチャンク形式である）場合※1 • リクエストの Content-Length ヘッダの値が，POST リクエスト転送先ワーカに設定された POST データサイズの上限を超えた値である場合※1
500　Internal Server Error	次のどれかに該当する場合，エラーステータスコード 500 が返ります。 • mod_jk.conf の内容に記述誤りがある場合※2 • workers.properties の読み込みに失敗した場合，または内容に記述誤りがある場合※1 • リクエストヘッダが 16KB を超えている場合※3 • Web コンテナとのコネクション確立に失敗した場合 • Web コンテナとのコネクション確立でタイムアウトが発生した場合 • Web コンテナへのデータ送信時にエラーが発生した場合 • Web コンテナへのデータ送信時にタイムアウトが発生した場合 • Web コンテナからのデータ受信時にエラーが発生した場合 • Web コンテナからのデータ受信時にタイムアウトが発生した場合 • クライアントからの POST データの読み込みでタイムアウトが発生した場合 • リクエストにサポートしない HTTP メソッド※4 が指定されている場合

注※1　POST データサイズによる振り分けで，デフォルトワーカを指定していない場合が該当します。

注※2　Windows の場合だけ該当します。UNIX の場合は起動に失敗します。

注※3　HTTP Server のリクエストヘッダの制限に関する設定で合計 16KB 以上のリクエストヘッダを受信可能にしている場合，該当するおそれがあります。

注※4　HTTP メソッドのサポート可否については，表 C-5 を参照してください。

表 C-4　リダイレクタが返すエラーステータスコードと発生条件（Microsoft IIS の場合）

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	次のどちらかの場合，エラーステータスコード 400 が返ります。 • リクエスト URL に"%"を含み，さらに"%"のあとに続く 2 文字が 16 進数を表さない文字（A～F，a～f，または 0～9 以外の文字）の場合 • リクエストの Host ヘッダのポート番号が不正の場合
403　Forbidden	次のどちらかの場合，エラーステータスコード 403 が返ります。 • リクエスト URL が"hitachi_ccfj"から始まる場合※1 • リクエスト URL に"%2F"が含まれる場合※1
500　Internal Server Error	次のどれかに該当する場合，エラーステータスコード 500 が返ります。 • isapi_redirect.conf の内容に記述誤りがある場合 • workers.properties の読み込みに失敗した場合，または内容に記述誤りがある場合

エラーステータスコード	エラーステータスコードを返す条件
	• リクエストヘッダが 16KB を超えている場合※2 • Web コンテナとのコネクション確立に失敗した場合 • Web コンテナとのコネクション確立でタイムアウトが発生した場合 • Web コンテナへのデータ送信時にエラーが発生した場合 • Web コンテナへのデータ送信時にタイムアウトが発生した場合 • Web コンテナからのデータ受信時にエラーが発生した場合 • Web コンテナからのデータ受信時にタイムアウトが発生した場合 • クライアントからの POST データの読み込みでタイムアウトが発生した場合 • リクエストにサポートしない HTTP メソッド※3 が指定されている場合

注※1　大文字・小文字の区別はありません。

注※2　Microsoft IIS のリクエストヘッダの制限に関する設定で合計 16KB 以上のリクエストヘッダを受信可能にしている場合，該当するおそれがあります。

注※3　HTTP メソッドのサポート可否については，表 C-5 を参照してください。

リダイレクタでのリクエストの HTTP メソッドのサポート可否

リダイレクタでのリクエストの HTTP メソッドのサポート可否について，次の表に示します。

表 C-5　リダイレクタでのリクエストの HTTP メソッドのサポート可否

HTTP メソッド	サポート可否
OPTIONS	○
GET	○
HEAD	○
POST	○
PUT	○
DELETE	○
TRACE	○
CONNECT	×※
PROPFIND	○
PROPPATCH	○
MKCOL	○
COPY	○
MOVE	○
LOCK	○
UNLOCK	○
ACL	○

HTTP メソッド	サポート可否
REPORT	○
VERSION-CONTROL	○
CHECKIN	○
CHECKOUT	○
UNCHECKOUT	○
SEARCH	○
上記以外の HTTP1.1 で使用できるメソッド	×※

（凡例）○：サポートする　×：サポートしない

注※

サポートしない HTTP メソッドが指定されているリクエストには，リダイレクタがステータス 500 エラーを返します。また，メッセージ KDJE41001-E が出力されます。

付録 C.3　インプロセス HTTP サーバが返すエラーステータスコード

クライアントからのリクエストのサイズが上限値を超えたり，値が不正だったりした場合，インプロセス HTTP サーバはエラーステータスコードを返します。インプロセス HTTP サーバが返すエラーステータスコードとエラーステータスコードを返す条件を次の表に示します。

表 C-6　インプロセス HTTP サーバが返すエラーステータスコードと条件

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	リクエストの状態が次のどれかの場合，エラーステータスコード 400 が返ります。 • リクエストの HTTP のバージョンが 1.1 で，Host ヘッダがない場合 • リクエストの Host ヘッダのポート番号が不正な場合 • リクエストヘッダのサイズが上限値を超えた場合 • リクエストヘッダの数が上限値を超えた場合 • リクエスト URI が不正な場合 • リクエスト URI のデコードに失敗した場合 • リクエスト URI を正規化できなかった場合 • リクエストの Content-Length ヘッダの値が 2147483647 より大きい，または 0 より小さい場合 • リクエストの Content-Length ヘッダの値が数値以外の場合 • リクエストの Content-Length ヘッダが複数指定されている場合 • リクエストラインの HTTP のバージョンがサポートされていない場合
403　Forbidden	web.xml の<transport-guarantee>要素で，INTEGRAL または CONFIDENTIAL が設定されているリソースに http でアクセスされた場合，エラーステータスコード 403 が返ります。

エラーステータスコード	エラーステータスコードを返す条件
405　Method Not Allowed	許可しない HTTP メソッドからアクセスがあった場合，エラーステータスコード 405 が返ります。
413　Request entity too large	リクエストボディのサイズが上限値を超えた場合，エラーステータスコード 413 が返ります。
414　Request-URI too large	リクエストラインの長さが上限値を超えた場合，エラーステータスコード 414 が返ります。
500　Internal Server Error	リダイレクト機能によってステータスコード 200 でファイルを返す際にファイルの読み込みに失敗した場合，エラーステータスコード 500 が返ります。
501　Not implemented	リクエストの transfer-encoding ヘッダの値が，サポートされていない場合に，エラーステータスコード 501 が返ります。
503　Service Unavailable	流量制御の上限を超えてリクエスト処理を実行しようとした場合，エラーステータスコード 503 が返ります。

付録 D　HTTP Server の設定に関する注意事項

ここでは，HTTP Server の設定に関する注意事項について説明します。

付録 D.1　HTTP Server の再起動時の注意事項

HTTP Server の再起動に失敗した原因が，簡易構築定義ファイル（Smart Composer 機能を使用しない場合はリダイレクタ定義ファイル（mod_jk.conf），またはワーカ定義ファイル（workers.properties））にある場合，次のどちらかにメッセージが出力されます。

- HTTP Server の起動コマンド実行画面またはイベントログ
  - HTTP Server をコマンドプロンプトから起動，停止，または再起動するときは，起動コマンド実行画面に出力されます。
  - HTTP Server をサービスから起動，停止，または再起動するときは，イベントログに出力されます。Windows の場合だけが該当します。
- HTTP Server のエラーログファイル

  エラーログファイルは，デフォルトでは次の場所に出力されます。
  - Windows の場合

    <HTTP Server のインストールディレクトリ>¥logs¥error.log
  - UNIX の場合

    /opt/hitachi/httpsd/logs/error.log

それぞれに出力されるメッセージを表に示します。

表 D–1　HTTP Server の起動コマンド実行画面またはイベントログに出力されるメッセージ

メッセージ	原因・対策方法
JkWorkersFile file_name invalid※	JkWorkersFile キーに指定されたファイル名が無効です。JkWorkersFile キーに指定している値を修正したあと，Web サーバを再起動してください。
Can't find the workers file specified※	指定されたワーカ定義ファイルが見つかりません。JkWorkersFile キーに指定されているファイルの有無を確認したあと，Web サーバを再起動してください。
Content should start with /※	URL パターンの先頭文字が「/」（スラッシュ）以外です。JkMount キーに指定されている URL パターンの先頭文字を「/」（スラッシュ）に修正したあと，Web サーバを再起動してください。
JkOptions: Illegal option 'a...a'	JkOption キーに指定された値（a...a）は不正です。 JkOption キーに指定している値を修正したあと，Web サーバを再起動してください。
a...a takes b...b arguments,	a...a に示すキーに指定できる値の数は b...b です。 a...a に示すキーに指定している値の数を修正したあと，Web サーバを再起動してください。

メッセージ	原因・対策方法
a...a must be On or Off	a...a に示すキーに指定できる値は On または Off です。 a...a に示すキーに指定している値を On または Off にしたあと，Web サーバを再起動してください。
JkModulePriority: Invalid value specified.a...a	JkModulePriority キーに設定された値 a...a が不正です。JkModulePriority キーに正しい値を設定して，Web サーバを再起動してください。

注※　UNIX の場合だけ出力されるメッセージです。

表 D–2　HTTP Server のエラーログファイル

メッセージ	説明・対策方法
[時刻] [emerg] Memory error	メモリが不足しています。 システムが利用できるメモリ使用量を確保したあと，Web サーバを再起動してください。
[時刻] [emerg] Error while opening the workers※	次のどちらかの問題が発生しています。 • ワーカ定義ファイルのオープンに失敗しました。 JkWorkersFile キーに指定されているファイルのアクセス権に読み取り権限があることを確認したあと，Web サーバを再起動してください。 • ワーカ定義ファイルの内容が不正です。 ワーカ定義ファイルの内容を正しく修正したあと，Web サーバを再起動してください。
[時刻] [emerg] Error while checking the mod_jk.conf※	リダイレクタ定義ファイルに不適切な記述がありました。リダイレクタに出力されているメッセージを確認したあと，Web サーバを再起動してください。

注※　UNIX の場合だけ出力されるメッセージです。

付録 D.2　リダイレクタのログに関する注意事項

- ログ出力先ディレクトリに HTTP Server の実行アカウントの書き込み権限がない場合，リクエストは正常に処理されますが，ログは出力されません。
- Windows の場合，デフォルトの状態ではリダイレクタのログ出力先ディレクトリ（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥logs）がありません。このため，HTTP Server は，起動時に一つ上のディレクトリ（redirector ディレクトリ）のアクセス権を引き継いで，logs ディレクトリを生成してログを出力しようとします。このとき，HTTP Server の実行アカウントの書き込み権限がないとログは出力されません。この場合は，logs ディレクトリを生成してアクセス権を設定するか，または一つ上のディレクトリ（redirector ディレクトリ）にアクセス権を設定してください。

  また，リダイレクタのログ出力先ディレクトリを変更している場合に，指定しているパスが途中までしかないときは，存在する最下層のディレクトリに対してアクセス権を設定するか，または指定したパスに対応するディレクトリをすべて作成して，最下層のディレクトリにアクセス権を設定してください。

付録 D.3　HTTP Server のアップグレード時の注意事項

HTTP Server をアップグレードした場合，HTTP Server 用のリダイレクタの変更内容を反映するためには，HTTP Server を再起動する必要があります。HTTP Server の再起動の方法については，マニュアル「HTTP Server」を参照してください。

付録 E　Microsoft IIS の設定

ここでは，Microsoft IIS を使用して Web サーバと連携する場合の設定方法について説明します。

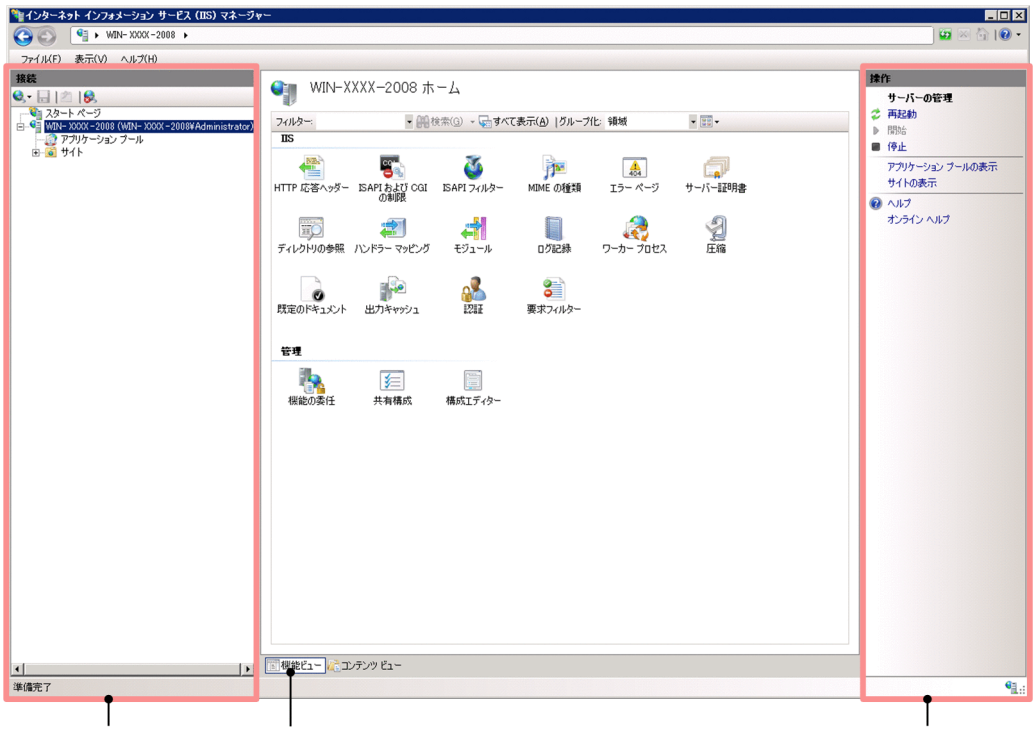
付録 E.1　Microsoft IIS 10.0 の設定

ここでは，Microsoft IIS を使用して Web サーバと連携する場合の Microsoft IIS 10.0 の設定方法について説明します。

J2EE サーバと Microsoft IIS との連携は，次に示す手順で設定します。

1. Microsoft IIS および役割サービスのインストール
2. ISAPI および CGI の制限の追加
3. ISAPI フィルタの追加
4. ハンドラマッピングの設定
5. 仮想ディレクトリの追加
6. Web サーバの認証機能の解除
7. [サーバー] ノードでのアプリケーションプールの設定
8. [サイト] ノードでのアプリケーションプールの設定
9. リダイレクタログ出力先ディレクトリへのアクセス権の設定
10. Web サイトの開始

> **参考**
>
> Microsoft IIS の設定で使用する，[インターネット インフォメーション サービス（IIS）マネージャー] の画面構成を次に示します。なお，以降で示す手順に従って，[接続] ウィンドウ，[機能ビュー]，[操作] ウィンドウなどで設定を実施してください。



(1) Microsoft IIS および役割サービスのインストール

Microsoft IIS を使用する場合，Microsoft IIS および用途に応じた役割サービスをインストールする必要があります。IIS および役割サービスをインストールする手順を次に示します。

Microsoft IIS がインストール済みの場合，不足している役割サービスがあれば追加インストールしてください（必要な役割サービスについては後述）。

1. ［サーバーマネージャー］を起動します。
   （自動起動していない場合，スタート画面から起動できます）

2. ［サーバーマネージャー］の［役割と機能の追加］をクリックします。

3. ウィザードが起動するので［次へ］をクリックします。

4. ［役割ベースまたは機能ベースのインストール］を選択し，［次へ］をクリックします。

5. 対象のサーバを選択し，［次へ］をクリックします。

6. ［サーバーの役割］では，［Web サーバー（IIS)］を選択し，［次へ］をクリックします。
   ［Web サーバー（IIS）に必要な機能を追加しますか？］のポップアップが出た場合，［管理ツールを含める（存在する場合)］にチェックを入れて［機能の追加］をクリックします。

7. ［機能の選択］では［次へ］をクリックします。

8. [Web サーバーの役割（IIS）] では［次へ］をクリックし，［役割サービスの選択］では最低限次の役割サービスを選択し，［次へ］をクリックします。
   - HTTP エラー
   - ディレクトリの参照
   - 既定のドキュメント
   - 静的なコンテンツ
   - ISAPI 拡張
   - ISAPI フィルター
   - IIS 管理コンソール

   指定した役割サービス以外も必要に応じて追加してください。特に［HTTP ログ］や［トレース］は障害発生時のトラブルシュートに有益です。
9. [インストール] をクリックします。
10. [閉じる] をクリックします。
    インストールが完了します。

(2) ISAPI および CGI の制限の追加

ISAPI および CGI の制限を追加する手順を次に示します。

1. [インターネット インフォメーション サービス（IIS）マネージャー] を起動します。
   ［サーバマネージャー］の［ツール］から起動できます。
2. [機能ビュー] のサーバの［ホーム］ページで，[ISAPI および CGI 制限] をダブルクリックします。
3. [ISAPI および CGI 制限] ページの［操作］ウィンドウで［追加］をクリックします。
4. [ISAPI または CGI の制限の追加] ダイアログで次の操作をします。
   - ［ISAPI または CGI パス］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）を指定します。
   - ［説明］に「ISAPI」と入力します。
   - ［拡張パスの実行を許可する］をチェックします。
5. [OK] ボタンをクリックします。
   ［ISAPI または CGI の制限の追加］ダイアログが閉じて，設定が反映されます。

(3) ISAPI フィルタの追加

ISAPI フィルタを追加する手順を次に示します。

1. [機能ビュー] のサイトの［ホーム］ページで［ISAPI フィルター］をダブルクリックします。

2. [ISAPI フィルター］ページの［操作］ウィンドウで［追加］をクリックします。

3. [ISAPI フィルターの追加］ダイアログで次の操作をします。
   - ［フィルター名］に「hitachi_ccfj」と入力します。
   - ［実行可能ファイル］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）を指定します。

4. [OK］ボタンをクリックします。
   ［ISAPI フィルターの追加］ダイアログが閉じて，設定が反映されます。

(4) ハンドラマッピングの設定

ハンドラマッピングを設定する手順を次に示します。

1. [機能ビュー] のサイトの［ホーム］ページで［ハンドラー マッピング］をダブルクリックします。

2. [ハンドラー マッピング］ページで「ISAPI-dll」を選択し，［操作］ウィンドウで［編集...］をクリックします。

3. [モジュール マップの編集］ダイアログで次の操作をします。
   - ［要求パス］に「*.dll」と入力します。
   - ［実行可能ファイル］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）を指定します。

   すでにハンドラマッピングが設定されている場合，［スクリプトマップの編集］ダイアログが起動しますが，操作内容は同じです。

4. [OK］ボタンをクリックします。

5. モジュールマップの編集を有効にするかどうかを確認するメッセージダイアログで，ISAPI 拡張機能を有効にするために，［はい］ボタンをクリックします。

6. [ハンドラー マッピング］ページで「ISAPI-dll」を選択し，［操作］ウィンドウで［機能のアクセス許可の編集...］をクリックします。

7. [機能のアクセス許可の編集］ダイアログで「読み取り」，「スクリプト」および「実行」のすべてのアクセス許可をチェックします。

8. [OK］ボタンをクリックします。
   ［機能のアクセス許可の編集］ダイアログが閉じて，設定が反映されます。

(5) 仮想ディレクトリの追加

「hitachi_ccfj」という名称の仮想ディレクトリを追加します。仮想ディレクトリを追加する手順を次に示します。

1. [接続] ウィンドウで［サイト］ノードを展開し，仮想ディレクトリを追加するサイトをクリックします。
2. [操作] ウィンドウで，［仮想ディレクトリの表示］をクリックします。
   ［仮想ディレクトリ］ページが表示されます。
3. [仮想ディレクトリ］ページの［操作］ウィンドウで，［仮想ディレクトリの追加...］をクリックします。
4. [仮想ディレクトリの追加] ダイアログで次の操作をします。
   - ［エイリアス］に「hitachi_ccfj」と入力します。
   - ［物理パス］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）が格納されているディレクトリを指定します。
5. [OK] ボタンをクリックします。
   ［仮想ディレクトリの追加］ダイアログが閉じて，設定が反映されます。

(6) Web サーバの認証機能の解除

Web コンテナの認証機能を使用する場合，次に示す Web サーバの認証機能の解除が必要になります。

- Active Directory クライアント証明書の認証
  Web コンテナが管理するリソースへアクセスする場合は解除してください。
- ダイジェスト認証
  Web コンテナの認証機能の使用の有無に関係なく，必ず解除してください。
- Windows 認証
  Web コンテナの認証機能（Basic 認証）を使用する場合は解除してください。
- APS.NET 偽装
  Web コンテナが管理するリソースへアクセスする場合は解除してください。

Active Directory クライアント証明書の認証，ダイジェスト認証，Windows 認証，フォーム認証および APS.NET 偽装を解除する手順を次に示します。

1. [接続] ウィンドウでサーバの［ホーム］ページを選択し，［認証］をダブルクリックします。
2. [認証] 画面で，[Active Directory クライアント証明書の認証] を選択し，［操作］ウィンドウで［無効にする］をクリックします（任意）。
3. [接続] ウィンドウで［サイト］ノードを展開し，「hitachi_ccfj」をクリックします。
4. [ホーム］ページで［認証］をダブルクリックします。

5. [認証] 画面で，[フォーム認証] を選択し，[操作] ウィンドウで [無効にする] をクリックします（任意）。

   Web コンテナが管理するリソースへアクセスする場合は解除してください。

6. [認証] 画面で，[ダイジェスト認証] を選択し，[操作] ウィンドウで [無効にする] をクリックします（必須）。

   Web コンテナの認証機能の使用の有無に関係なく，ダイジェスト認証は必ず無効にしてください。

7. [認証] 画面で，[Windows 認証] を選択し，[操作] ウィンドウで [無効にする] をクリックします（任意）。

   Web コンテナの Basic 認証を設定している場合は，必ず解除してください。

8. [認証] 画面で，[APS.NET 偽装] を選択し，[操作] ウィンドウで [無効にする] をクリックします（任意）。

   Web コンテナが管理するリソースへアクセスする場合は解除してください。

9. [OK] ボタンをクリックして各ウィンドウを閉じます。

(7) [サーバー] ノードでのアプリケーションプールの設定

[サーバー] ノードでのアプリケーションプールの設定手順を次に示します。

1. [接続] ウィンドウでサーバノードを展開し，[アプリケーション プール] をクリックします。

2. [アプリケーション プール] ページで，使用するアプリケーションプールを選択し，[操作] ウィンドウの [詳細設定] をクリックします。

3. [詳細設定] ダイアログで次の操作をします。

   [ワーカー プロセスの最大数] ※に，Microsoft IIS 内でリクエストを処理するプロセスの最大数を指定します。

   注※

   このマニュアルでは，Web コンテナの実行プロセスのことも「ワーカプロセス」と表記していますが，ここで設定する「ワーカー プロセス」とは Microsoft IIS 内でリクエストを処理するプロセスを指します。

4. [OK] ボタンをクリックします。

   [詳細設定] ダイアログが閉じて，設定が反映されます。

(8) [サイト] ノードでのアプリケーションプールの設定

サイトノードでのアプリケーションプールの設定手順を次に示します。

1. [接続] ウィンドウで [サイト] ノードを展開し，アプリケーションプールを指定するサイトをクリックします。

2. [操作] ウィンドウで，[詳細設定] をクリックします。

3. [詳細設定] ダイアログで，[アプリケーションプール] を入力します。
   [アプリケーションプール] に，「(7) [サーバー] ノードでのアプリケーションプールの設定」で設定したアプリケーションプールの名称を指定します。

4. [OK] ボタンをクリックします。
   [詳細設定] ダイアログが閉じて，設定が反映されます。

(9) リダイレクタログ出力先ディレクトリへのアクセス権の設定

リダイレクタのログ出力先ディレクトリには，Microsoft IIS のアプリケーションプールの実行アカウントに対する書き込み権限を付加しておく必要があります。エクスプローラから，リダイレクタのログ出力ディレクトリにアクセス権を設定してください。

アプリケーションプールの実行アカウントは，アプリケーションプールの［詳細設定］ダイアログの［ID］で指定します。デフォルトの ID を指定している場合は，「IIS_IUSRS」グループに対して書き込み権限を付加してください。

デフォルトの ID は次のとおりです。

- Microsoft IIS 10.0：「ApplicationPoolIdentity」

なお，新規インストール時には，デフォルトの状態ではリダイレクタのログ出力先ディレクトリ（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥logs）がありません。このため，logs ディレクトリを作成してアクセス権を設定するか，または一つ上のディレクトリ（redirector ディレクトリ）にアクセス権を設定してください。

また，リダイレクタのログ出力先ディレクトリを変更している場合に，指定しているパスが途中までしかないときは，存在する最下層のディレクトリに対してアクセス権を設定するか，または指定したパスに対応するディレクトリをすべて作成して，最下層のディレクトリにアクセス権を設定してください。

(10) Web サイトの開始

Microsoft IIS の Web サイトの開始の手順を次に示します。

1. [接続] ウィンドウで［サイト］ノードを展開し，開始するサイトをクリックします。

2. [操作] ウィンドウで，[開始] をクリックします。開始済みの場合は，[再起動] をクリックします。
   Web サイトが開始，または再起動します。

(11) 注意事項

ここでは，Microsoft IIS の設定時の注意事項について説明します。

(a) 構成設定の複数環境へのレプリケーションについての注意事項

Microsoft IIS では，構成設定を web.config ファイルに保存できます。また，保存した web.config ファイルを基に，xcopy を使用して複数環境に構成環境をレプリケーションできます。

ただし，リダイレクタを使用する構成環境の場合，xcopy での複数環境へのレプリケーションでリダイレクタの設定はできません。xcopy で複数環境を同一の構成設定にする場合も，リダイレクタの設定は環境ごとに手動で同期してください。

(b) エラーページのカスタマイズについての注意事項

Microsoft IIS でエラーページにカスタムエラーページを返す設定にしている場合，web.xml の<error-page>タグによるエラーページのカスタマイズは無効となることがあります。web.xml の<error-page>タグによるエラーページのカスタマイズを有効にしたい場合は，Microsoft IIS のエラーページのカスタムエラーページの設定を無効にしてください。

(c) ほかのフィルタと共存させる場合の注意事項

Microsoft IIS が受信したリクエストが，Web コンテナに転送するリクエストの場合，Microsoft IIS 用リダイレクタは ISAPI フィルタ内で使用するリクエスト URL 情報を変更します。そのため，Microsoft IIS 用リダイレクタよりあとに実行される ISAPI フィルタでは，Microsoft IIS が受信したリクエスト URL を取得できません。Microsoft IIS が受信したリクエスト URL をフィルタで取得する場合は，Microsoft IIS 用リダイレクタよりも順番が先になるように設定してください。順番を調整するために，Microsoft IIS 用リダイレクタの優先順位を「中」または「低」に変更する必要がある場合は，isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）の filter_priority キーで指定します。isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）の filter_priority キーの詳細については，「13.2.1 isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）」を参照してください。

付録 F　JPA プロバイダと EJB コンテナ間の規約

ここでは，JPA プロバイダと EJB コンテナ間の規約について説明します。

付録 F.1　ランタイムに関する規約

ランタイムに関する規約には，コンテナ側の責任と JPA プロバイダ側の責任があります。

(1)　コンテナ側の責任

- トランザクションスコープの永続化コンテキストに関する規約

  トランザクションスコープの永続化コンテキストが使用されている場合に，JTA トランザクションにエンティティマネージャが関連づいていないときには，コンテナは次の処理を実行します。

  - コンテナは次の場合に，EntityManagerFactory.createEntityManager を呼び出して新しいエンティティマネージャを作成します。

    JTA トランザクションのスコープ内のビジネスメソッドで，トランザクションスコープの永続化コンテキストを使用するエンティティマネージャのメソッドが初めて呼び出された場合

  - JTA トランザクションが決着（コミットまたはロールバック）したあとに，コンテナは EntityManager.close を呼び出して，エンティティマネージャをクローズします。

  また，コンテナは次の条件をすべて満たすと，TransactionRequiredException をスローします。

  - トランザクションスコープの永続化コンテキストが使用されている場合
  - トランザクションがアクティブでない場合
  - アプリケーションから EntityManager の persist，remove，merge，refresh メソッドが呼び出された場合

- 拡張永続化コンテキストに関する規約

  拡張永続化コンテキストが使用されている場合は，コンテナは次の処理を行います。

  - コンテナは次の場合に，EntityManagerFactory.createEntityManager を呼び出して新しいエンティティマネージャを作成します。

    Stateful Session Bean のインスタンスが作成された際に，拡張永続化コンテキストを使用するエンティティマネージャへの参照が定義されている場合

  - コンテナは次のタイミングで，EntityManager.close を呼び出してエンティティマネージャをクローズします。

    エンティティマネージャを作成した Stateful Session Bean，および同じ永続化コンテキストを引き継いだ Stateful Session Bean が削除されたとき

  - コンテナ管理のトランザクションを使用する Stateful Session Bean のビジネスメソッドの呼び出し時に，エンティティマネージャが JTA トランザクションと関連づけられていない場合には，コンテナはエンティティマネージャを JTA トランザクションに関連づけ，

EntityManager.joinTransaction を呼び出します。JTA トランザクションにすでに別のエンティティマネージャが関連づけられている場合には，コンテナは EJBException をスローします。

- ビーン管理のトランザクションを使用する Stateful Session Bean のビジネスメソッド内で UserTransaction.begin が呼び出された場合には，コンテナはエンティティマネージャを JTA トランザクションと関連づけ，EntityManager.joinTransaction を呼び出します。なお，ビーン管理のトランザクションについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「2.7.2　BMT」を参照してください。

- **コンテナ管理のエンティティマネージャに関する規約**

  コンテナ管理のエンティティマネージャを使用している場合に，アプリケーションが EntityManager.close を呼び出すと，コンテナは IllegalStateException をスローします。

  @PersistenceCotnext または DD の<persistence-context-ref>タグで，プロパティが指定された場合には，コンテナは EntityManagerFactory.createEntityManager(Map map)メソッドを使用してエンティティマネージャを作成し，指定されたプロパティを map 引数に含めて JPA プロバイダに渡します。

- **コネクションの自動クローズ機能について**

  アプリケーションサーバの場合，Session Bean や Web コンポーネントの延長で JPA プロバイダが取得したコネクションは，コンテナの自動クローズ機能によって自動的にクローズされることがあります。コネクションの自動クローズ機能とは，コネクションのリークを防止するための機能です。

  コネクションは，次の条件を満たすと自動クローズの対象となります。

  - Stateless Session Bean で取得したコネクションは，ビジネスメソッドからリターンするときに，自動クローズの対象となります。
  - Stateful Session Bean で取得したコネクションは，Stateful Session Bean が破棄されるときに，自動クローズの対象となります。
  - Web コンポーネントで取得したコネクションは，サービスメソッドからリターンするときに，自動クローズの対象となります。

  ただし，これらの条件に当てはまる場合でも，コネクションが JTA トランザクションに参加しているときには，JTA トランザクションがコミットするまで自動クローズが保留されます。

(2)　JPA プロバイダ側の責任

アプリケーションで使用するエンティティマネージャが，トランザクションスコープの永続化コンテキストを使用するように定義されているか，拡張永続化コンテキストを使用するように定義されているかは，JPA プロバイダには伝わりません。JPA プロバイダでの責任は，コンテナが要求したときにエンティティマネージャを作成し，トランザクションからトランザクションの決着の通知を受け取るための Synchronization をトランザクションに登録することです。

- コンテナが EntityManagerFactory.createEntityManager を呼び出したときには，JPA プロバイダは新しいエンティティマネージャを作成し，それをコンテナに返す必要があります。JTA トランザクションがアクティブである場合は，JPA プロバイダは Synchronization を JTA トランザクションに登録する必要があります。

- コンテナが EntityManager.joinTransaction を呼び出したときには，JPA プロバイダは Synchronization を JTA トランザクションに登録する必要があります。ただし，以前に joinTransaction が呼び出されていて，JTA トランザクションに Synchronization を登録済みの場合には，何もする必要はありません。
- JTA トランザクションがコミットされるときには，JPA プロバイダはすべての変更されたエンティティの状態を，データベースにフラッシュする必要があります。
- JTA トランザクションがロールバックされるときには，JPA プロバイダは，すべての managed 状態のエンティティをデタッチする必要があります。
- JPA プロバイダがトランザクションのロールバックの原因になる例外をスローする場合には，JPA プロバイダがトランザクションをロールバックにマークする必要があります。
- コンテナが EntityManager.close を呼び出した場合，そのエンティティマネージャが関係したすべての未解決トランザクションが決着されたあとに，JPA プロバイダはすべての確保したリソースを解放する必要があります。エンティティマネージャがすでにクローズされている場合には，JPA プロバイダは IllegalStateException をスローする必要があります。
- コンテナが EntityManager.clear を呼び出した場合，JPA プロバイダはすべての managed 状態のエンティティをデタッチする必要があります。

(3) javax.transaction.TransactionSynchronizationRegistry インタフェース

JPA プロバイダは，トランザクションに Synchronization を登録したり，トランザクションをロールバックにマークしたりするために，TransactionSynchronizationRegistry インタフェースを使用できます。TransactionSynchronizationRegistry のインスタンスは，JNDI を使用して「java:comp/TransactionSynchronizationRegistry」という名前でルックアップできます。

インタフェース定義を次に示します。

```
package javax.transaction;

/**
 * このインタフェースは，JPAプロバイダやリソースアダプタなど，
 * アプリケーションサーバのシステムレベルのコンポーネントから使用する
 * ためのものです。
 * このインタフェースを使って，
 * 特別な順序で呼ばれるシンクロナイゼーションの登録，
 * 現在のトランザクションへのリソースオブジェクトの登録，
 * 現在のトランザクションのコンテキストの取得，
 * 現在のトランザクションのステータスの取得，
 * 現在のトランザクションをロールバックにマークできます。
 *
 * このインタフェースはステートレスなサービスオブジェクトとして，
 * アプリケーションサーバによって実装されます。
 * 同じオブジェクトを複数のコンポーネントからマルチスレッドセーフに
 * 使用できます。
 *
 * 標準的なアプリケーションサーバでは，このインタフェースを実装した
```

```
 * インスタンスは，JNDIを使用して標準的な名前でルックアップできます。
 * 標準的な名前は
 * 「java:comp/TransactionSynchronizationRegistry」です。
 */
public interface TransactionSynchronizationRegistry {

    /**
     * このメソッドを呼び出した時に現在のスレッドに関連づいている
     * トランザクションを表現する一意のオブジェクトを返します。
     * このオブジェクトのhashCodeとequalsメソッドは
     * オーバーライドされており，
     * ハッシュマップのキーとして使用できます。
     * トランザクションが存在しない場合には，nullを返します。
     *
     * 同じアプリケーションサーバの同じトランザクションコンテキスト内で
     * このメソッドが呼ばれて返されたオブジェクトは，すべて同じhashCodeを
     * 持ち，equalメソッドでの比較結果はtrueとなります。
     *
     * toStringメソッドは，トランザクションコンテキストの情報を人が
     * 読みやすい形の文字列として返します。
     * ただし，toStringで返される文字列のフォーマットは規定されていません。
     * また，toStringの結果に関して，バージョン間での互換性は
     * 保障されていません。
     *
     * 取得したオブジェクトがシリアライズできる保障はなく，
     * JavaVMの外に出したときの動作は規定されていません。
     *
     * @return このメソッドが呼ばれたときにスレッドに関連づいている
     * トランザクションを一意に表現するオブジェクト
     */
    Object getTransactionKey();

    /**
     * このメソッドを呼び出した時のスレッドに関連づいているトランザクション
     * のリソースマップに，オブジェクトを追加またはリプレースします。
     * マップのキーは，コンフリクトが発生しないように，
     * このメソッドを呼び出す側で定義されたクラスである必要があります。
     * キーとして使用するクラスは，マップのキーとして適切なhashCodeとequals
     * メソッドを持っている必要があります。
     * マップのキーや値が，このクラスによって評価されたり，使用されたりする
     * ことはありません。
     * このメソッドの一般的な規約は，Mapのputメソッドと同じで，
     * キーはnull以外にする必要がありますが，値はnullにすることもできます。
     * すでにキーに関連づいた値が存在する場合，その値は置き換えられます。
     *
     * @param key マップのエントリのキー
     * @param value マップのエントリの値
     * @exception IllegalStateException アクティブなトランザクションが
     * 存在しない場合
     * @exception NullPointerException 引数のキーがnullである場合
     */
    void putResource(Object key, Object value);

    /**
     * このメソッドを呼び出した時のスレッドに関連づいているトランザクション
     * のリソースマップから，オブジェクトを取り出します。
     * キーは，同じトランザクション内で，事前にputResourceメソッドに
     * 指定したオブジェクトと同じである必要があります。
```

```
     * 指定されたキーが現在のリソースマップに存在しない場合は，
     * nullを返します。
     * このメソッドの一般的な規約は，Mapのputメソッドと同じで，
     * キーはnull以外にする必要がありますが，値はnullにすることもできます。
     * マップにキーが格納されていない場合や，キーに対してnullが格納
     * されている場合には，返り値はnullになります。
     * @param key マップのエントリのキー
     * @return キーに関連づけられた値
     * @exception IllegalStateException アクティブなトランザクションが
     * 存在しない場合
     * @exception NullPointerException 引数のキーがnullである場合
     */
    Object getResource(Object key);

    /**
     * 特別な順序で呼ばれるシンクロナイゼーションのインスタンスを登録します。
     * このメソッドで登録したSynchronizationのbeforeCompletionは，
     * すべてのSessionSynchronization.beforeCompletionや，
     * トランザクションに直接登録された
     * Synchronization.beforeCompletionが呼ばれたあとで，
     * 2フェーズコミット処理が開始される前に呼び出されます。
     * 同じように，このメソッドで登録した
     * SynchronizationのafterCompletionは，
     * 2フェーズコミット処理が完了したあとで，
     * すべてのSessionSynchronization.afterCompletionや，
     * トランザクションに直接登録されたSynchronization.afterCompletionが
     * 呼ばれる前に呼び出されます。
     *
     * beforeCompletionは，このメソッドを呼び出した時にスレッドに
     * 関連づいていたトランザクションのコンテキストで呼び出されます。
     * beforeCompletionでは，コネクタなどのリソースへのアクセスは
     * 許可されていますが，タイマーサービスやビーンのメソッドなどの
     * ユーザコンポーネントへのアクセスは許可されていません。
     * これは，呼び出し側によって管理されているデータや，
     * registerInterposedSynchronizationで登録された
     * ほかのSynchronizationによって
     * すでにフラッシュされたデータを変更してしまうおそれがあるためです。
     * 一般的なコンテキストは，registerInterposedSynchronizationを
     * 呼び出したコンポーネントのコンテキストになります。
     *
     * afterCompletionが呼び出される時のコンテキストは定義されていません。
     * なお，ユーザコンポーネントへのアクセスは許可されていません。
     * また，リソースをクローズすることはできますが，
     * リソースに対するトランザクショナルな操作はできません。
     *
     *トランザクションがアクティブでない場合にこのメソッドが呼び出されたとき，
     * IllegalStateExceptionがスローされます。
     *
     * このメソッドが2フェーズコミット処理の開始後に呼び出された場合は，
     * IllegalStateExceptionがスローされます。
     *
     * @param sync 登録するSynchronizationのインスタンス
     * @exception IllegalStateException アクティブなトランザクションが
     * 存在しない場合
     */
    void registerInterposedSynchronization(Synchronization sync);

    /**
```

```
     * このメソッドを呼んだ時にスレッドに関連づいているトランザクション
     * のステータスを返します。
     * このメソッドの返り値は,
     * TransactionManager.getStatus()の結果と同じです。
     *
     * @return このメソッドを呼んだときにスレッドに関連づいている
     * トランザクションのステータス
     */
    int getTransactionStatus();

    /**
     * このメソッドを呼んだ時にスレッドに関連づいているトランザクション
     * がロールバックされるようにマークします。
     *
     * @exception IllegalStateException アクティブなトランザクションが
     * 存在しない場合
     */
    void setRollbackOnly();

    /**
     * このメソッドを呼んだ時にスレッドに関連づいているトランザクション
     * がロールバックするようにマークされているかどうかを返します。
     *
     * @return ロールバックするようにマークされている場合はtrue
     * @exception IllegalStateException アクティブなトランザクションが
     * 存在しない場合
     */
    boolean getRollbackOnly();
}
```

付録 F.2　デプロイメントに関する規約

デプロイメントに関する規約には，コンテナ側の責任と JPA プロバイダ側の責任があります。

(1)　コンテナ側の責任

デプロイメント時に，コンテナはアプリケーション内の決められた場所にパッケージングされた persistence.xml を検索します。アプリケーション内に persistence.xml が存在する場合には，コンテナは persistence.xml に定義された永続化ユニットの定義を処理します。なお，コンテナが検索する場所については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「5.8 persistence.xml での定義」を参照してください。

コンテナは persistence.xml ファイルを persistence_1_0.xsd で検証します。検証の結果，エラーが発生した場合にはユーザに通知します。persistence.xml にプロバイダやデータソースの情報が指定されていない場合には，デフォルト値が使用されます。使用されるデフォルト値は次のとおりです。

- <provider>タグ

  簡易構築定義ファイルで指定したデフォルトの JPA プロバイダが使用されます。また，このタグを省略した場合で，簡易構築定義ファイルでデフォルトの JPA プロバイダが指定されていないときには，JPA プロバイダとして CJPA プロバイダが使用されます。

ポイント

簡易構築定義ファイルでデフォルトの JPA プロバイダを指定するには，論理 J2EE サーバの<param-name>タグに ejbserver.jpa.defaultProviderClassName を指定して，<param-value>タグにデフォルトの JPA プロバイダクラス名を指定します。

なお，簡易構築定義ファイルで，論理 J2EE サーバの<param-name>タグに ejbserver.jpa.overrideProvider パラメタが指定されているときは，ejbserver.jpa.overrideProvider パラメタの<param-value>タグに指定されている JPA プロバイダクラス名が，<provider>タグや ejbserver.jpa.defaultProviderClassName パラメタに指定した値よりも優先して使用されます。

簡易構築定義ファイルに指定するパラメタについては，「11.2.1　J2EE サーバ用ユーザプロパティを設定するパラメタ」を参照してください。

永続化ユニットで使用される JPA プロバイダを決定する優先順位を次の表に示します。

表 F-1　永続化ユニットで使用される JPA プロバイダを決定する優先順位

優先度	使用する JPA プロバイダ
1	簡易構築定義ファイルの ejbserver.jpa.overrideProvider プロパティに指定された値
2	persistence.xml の<provider>タグに指定された値
3	CJPA プロバイダ （簡易構築定義ファイルの ejbserver.jpa.defaultProviderClassName パラメタを使用して変更することもできる）

- <jta-data-source>タグ，<non-jta-data-source>タグ

  簡易構築定義ファイルの ejbserver.jpa.defaultJtaDsName パラメタまたは ejbserver.jpa.defaultNonJtaDsName パラメタに指定した値が使用されます。ただし，これらのプロパティにはデフォルト値がありません。

  ejbserver.jpa.overrideJtaDsName パラメタまたは ejbserver.jpa.overrideNonJtaDsName パラメタに値が指定されている場合は，<jta-data-source>タグ，<non-jta-data-source>タグに指定された値や，ejbserver.jpa.defaultJtaDsName パラメタ，ejbserver.jpa.defaultNonJtaDsName パラメタに指定された値よりも優先して使用されます。

  永続化ユニットで使用される JTA データソースおよび非 JTA データソースを決定するときの優先順位を次の表に示します。

表 F-2　永続化ユニットで使用される JTA データソースおよび非 JTA データソースを決定する優先順位

優先度	使用する JPA プロバイダ
1	簡易構築定義ファイルの ejbserver.jpa.overrideJtaDsName プロパティまたは ejbserver.jpa.overrideNonJtaDsName プロパティに指定された値
2	persistence.xml の<jta-data-source>または<non-jta-data-source>エレメントに指定された値
3	簡易構築定義ファイルの ejbserver.jpa.defaultJtaDsName プロパティまたは ejbserver.jpa.defaultNonJtaDsName プロパティに指定した値

コンテナが永続化ユニットのエンティティマネージャファクトリを作成するときには，JPA プロバイダにプロパティを渡すことがあります。

コンテナは，persistence.xml で永続化ユニットごとに定義された javax.persistence.spi.PersistenceProvider の実装クラスのインスタンスを作成し，createContainerEntityManagerFactory メソッドを呼び出して，コンテナ管理のエンティティマネージャを作成するための EntityManagerFactory を取得します。永続化ユニットのメタデータは，PersistenceUnitInfo オブジェクトとして，createContainerEntityManagerFactory メソッドの引数で JPA プロバイダに渡されます。コンテナは一つの永続化ユニット定義に対して，一つだけ EntityManagerFactory を作成し，その EntityManagerFactory から複数の EntityManager を作成します。

永続化ユニットが再デプロイされる場合には，コンテナはすでに取得した EntityManagerFactory の close メソッドを呼び出したあと，createContainerEntityManagerFactory を新しい PersistenceUnitInfo とともに呼び出します。

(2) JPA プロバイダ側の責任

JPA プロバイダは PersistenceProvider SPI を実装し，PersistenceProvider の createContainerEntityManagerFactory メソッドが呼ばれたときに，引数で渡される永続化ユニットのメタデータ（PersistenceUnitInfo）を使用して，EntityManagerFactory を作成し，コンテナに返す必要があります。

JPA プロバイダは，永続化ユニットに含まれるマネージドクラス（エンティティクラスなど）のメタデータアノテーションを処理します。また，永続化ユニットで O/R マッピングファイルが使用されている場合には，JPA プロバイダが解釈する必要があります。このとき，O/R マッピングファイルを orm_1_0.xsd を使用して検証し，エラーが発生した場合はユーザに通知する必要があります。

(3) javax.persistence.spi.PersistenceProvider インタフェース

JPA プロバイダは，javax.persistence.spi.PersistenceProvider インタフェースを実装する必要があります。このインタフェースはコンテナによって呼び出されるものであり，アプリケーションから呼び出すものではありません。PersistenceProvider の実装クラスは，public で引数のないコンストラクタを持っている必要があります。

javax.persistence.spi.PersistenceProvider インタフェースを次に示します。

```
package javax.persistence.spi;

/**
* JPAプロバイダによって実装されるインタフェースです。
* このインタフェースはEntityManagerFactoryを作成するために使用されます。
* Java EE環境ではコンテナによって呼び出され，
* JavaSE環境ではPersistenceクラスによって呼び出されます。
*/
public interface PersistenceProvider {
```

```
  /**
  * PersistenceクラスがEntityManagerFactoryを作成する時に呼び出されます。
  *
  * @param emName 永続化ユニットの名前
  * @param map JPAプロバイダによって使用されるプロパティのMap。
  * ここに指定されたプロパティは，persistence.xmlファイルの対応する
  * プロパティを上書きするため，またはpersistence.xmlファイルに
  * 指定されていないプロパティを指定するために使用されます。
  *（プロパティを指定する必要がない場合はnullが渡されます）
  * @return 永続化ユニットのEntityManagerFactory
  * JPAプロバイダが正しくない場合は，nullを返します。
  */
  public EntityManagerFactory createEntityManagerFactory(String
  emName, Map map);

  /**
  * コンテナがEntityManagerFactoryを作成する時に呼びだされます。
  *
  * @param info JPAプロバイダが使用するためのメタデータ
  * @param map JPAプロバイダが使用するためのインテグレーションレベルの
  * プロパティ（指定しない場合はnullが渡されます）
  * @return メタデータで指定された永続化ユニットのEntityManagerFactory
  */
  public EntityManagerFactory createContainerEntityManagerFactory(
  PersistenceUnitInfo info, Map map);
}
```

(4) javax.persistence.spi.PersistenceUnitInfo インタフェース

javax.persistence.spi.PersistenceUnitInfo インタフェースの定義を次に示します。

```
import javax.sql.DataSource;
/**
* このインタフェースはコンテナによって実装され，
* EntityManagerFactoryを作成する時にJPAプロバイダに渡されて使用される。
*/
public interface PersistenceUnitInfo {

  /**
  * @return persistence.xmlで定義された永続化ユニット名
  */
  public String getPersistenceUnitName();

  /**
  * @return persistence.xmlの<provider>エレメントに定義された，
  * JPAプロバイダ実装クラスの完全修飾クラス名
  */
  public String getPersistenceProviderClassName();

  /**
  * @return EntityManagerFactoryによって作成されるEntityManagerの
  * トランザクションのタイプを返す。
  * persistence.xmlのtransaction-type属性に指定されたタイプである。
  */
  public PersistenceUnitTransactionType getTransactionType();
```

```
/**
* @return JPAプロバイダが使用するためのJTAが有効な
* データソースを返す。
* persistence.xmlの<jta-data-source>エレメントで指定されたデータソースか,
* デプロイメント時にコンテナによって決定されたデータソースである。
*/
public DataSource getJtaDataSource();

/**
* @return JPAプロバイダがJTAトランザクションの外で
* データにアクセスするための，JTAが無効なデータソースを返す。
* persistence.xmlの<non-jta-data-source>エレメントで指定された
* データソースか，デプロイメント時にコンテナによって決定された
* データソースである。
*/
public DataSource getNonJtaDataSource();

/**
* @return JPAプロバイダがエンティティクラスのマッピングを
* 決定するためにロードする必要のある，マッピングファイル名のリスト
* マッピングファイルは，標準的なXML形式のマッピングフォーマット
* でなければならない。
* 一意の名前を持ち，アプリケーションのクラスパスからリソースとして
* ロードできるものでなければならない。
* それぞれのマッピングファイル名は，persistence.xmlの<mapping-file>タグ
* に指定されたものである。
*/
public List<String> getMappingFileNames();

/**
* JPAプロバイダが，永続化ユニットのマネージドクラスを検索する必要のある,
* JARファイルまたはJARファイルを展開したディレクトリのURLのリストを返す。
* それぞれのURLはpersistence.xmlファイルの<jar-file>タグに指定された
* ものである。
* URLはJARファイルまたはJARファイルを展開したディレクトリを指す
* ファイルURLか，またはJARのフォーマットのInputStreamを取得できる
* そのほかのURL形式である。
*
* @return JARファイルまたはディレクトリを指すURLオブジェクトのリスト
*/
public List<URL> getJarFileUrls();

/**
* 永続化ユニットルートであるJARファイルまたはディレクトリのURLを返す。
* （永続化ユニットルートがWEB-INF/classesディレクトリである場合,
* WEB-INF/classesディレクトリのURLを返す）
* URLはJARファイルまたはJARファイルを展開したディレクトリを指す
* ファイルURLか，またはJARのフォーマットのInputStreamを取得できる
* そのほかのURL形式である。
*
* @return JARファイルまたはディレクトリを指すURLオブジェクト
*/
public URL getPersistenceUnitRootUrl();

/**
* @return JPAプロバイダがマネージドクラスとして扱わなくてはならない
* クラス名のリスト。
* それぞれのクラス名はpersistence.xmlファイルの<class>タグに指定された
```

```
* ものである。
*/
public List<String> getManagedClassNames();

/**
* @return 永続化ユニットルートに配置されていて,
* 明示的にマネージドクラスであると指定されていないクラスを,
* マネージドクラスとして扱うかどうかを返す。
* この値は, persistence.xmlファイルの<exclude-unlisted-classes>タグに
* 指定されたものである。
*/
public boolean excludeUnlistedClasses();

/**
* @return Propertiesオブジェクト。
* それぞれのプロパティはpersistence.xmlファイルの<property>タグに
* 指定されたものである。
*/
public Properties getProperties();

/**
* @return JPAプロバイダがクラスやリソースをロードしたり,
* URLをオープンしたりするために使用できるClassLoader。
*/
public ClassLoader getClassLoader();

/**
* PersistenceUnitInfo.getClassLoaderメソッドによって返されるクラスローダで,
* 新しいクラスの定義や, クラスの再定義のたびに呼ばれる,
* JPAプロバイダのトランスフォーマーを登録する。
* このトランスフォーマーは, PersistenceUnitInfo.getNewTempClassLoaderメソッドで返される
* クラスローダでロードされるクラスには影響を与えない。
* クラスローディングのスコープ内で永続化ユニットが幾つ定義されていても,
* 同じクラスローディングのスコープ内でクラスが変換されるのは一度だけである。
*
* @param transformer コンテナがクラスの定義（再定義）時に呼び出す,
* JPAプロバイダのトランスフォーマー
*/
public void addTransformer(ClassTransformer transformer);

/**
* JPAプロバイダが一時的にクラスやリソースをロードしたり,
* URLをオープンしたりするために使用できるクラスローダの
* 新しいインスタンスを返す。
* このクラスローダのスコープやクラスパスは,
* PersistenceUnitInfo.getClassLoaderで返される
* クラスローダと完全に同じである。
* このクラスローダでロードしたクラスが,
* アプリケーションのコンポーネントから参照できるようになることはない。
* JPAプロバイダは, createContainerEntityManagerFactoryの
* 呼び出しの延長でだけ, このクラスローダを使用できる。
*
* @return 現在のクラスローダと同じスコープ・クラスパスを持つ一時的な
* クラスローダ
*/
public ClassLoader getNewTempClassLoader();
```

```
}
```

参考

アプリケーションサーバでは，永続化ユニットで JTA データソース，非 JTA データソースが定義されていない場合には，getJtaDataSource()または getNonJtaDataSource()は null を返します。永続化ユニットで JTA データソース，非 JTA データソースが定義されていない場合とは，次の状態を指します。

- persistence.xml で<jta-data-source>，<non-jta-data-source>が省略されていて，かつデフォルト値がシステムプロパティ ejbserver.jpa.defaultJtaDsName, ejbserver.jpa.defaultNonJtaDsName で定義されていない場合
- システムプロパティ ejbserver.jpa.overrideJtaDsName, ejbserver.jpa.overrideNonJtaDsName も定義されていない場合

付録 G　JPQL の BNF

ここでは，JPA1.0 仕様で規定されている JPQL の BNF について示します。

表 G-1　BNF 表記の規則

表記	内容
{ }	グループを表します。
[ ]	オプションの構文を表します。
*	0 以上を表します。
A|B	A または B を表します。

なお，背景色付きの**太字**部分はキーワードを表します。

次に，BNF を示します。

```
QL_statement ::= select_statement | update_statement | delete_statement
select_statement ::= select_clause from_clause [where_clause] [groupby_clause]
[having_clause] [orderby_clause]
update_statement ::= update_clause [where_clause]
delete_statement ::= delete_clause [where_clause]
from_clause ::=
FROM identification_variable_declaration
{, {identification_variable_declaration | collection_member_declaration}}*
identification_variable_declaration ::= range_variable_declaration { join | fetch_join }*
range_variable_declaration ::= abstract_schema_name [AS] identification_variable
join ::= join_spec join_association_path_expression [AS] identification_variable
fetch_join ::= join_spec FETCH join_association_path_expression
association_path_expression ::=
collection_valued_path_expression | single_valued_association_path_expression
join_spec::= [ LEFT [OUTER] | INNER ] JOIN
join_association_path_expression ::= join_collection_valued_path_expression |
join_single_valued_association_path_expression
join_collection_valued_path_expression::=
identification_variable.collection_valued_association_field
join_single_valued_association_path_expression::=
identification_variable.single_valued_association_field
collection_member_declaration ::=
IN (collection_valued_path_expression) [AS] identification_variable
single_valued_path_expression ::=
state_field_path_expression | single_valued_association_path_expression
state_field_path_expression ::=
{identification_variable | single_valued_association_path_expression}.state_field
single_valued_association_path_expression ::=
identification_variable.{single_valued_association_field.}* single_valued_association_field
collection_valued_path_expression ::=
identification_variable.{single_valued_association_field.}*collection_valued_association_fie
ld
state_field ::= {embedded_class_state_field.}*simple_state_field
update_clause ::= UPDATE abstract_schema_name [[AS] identification_variable]
SET update_item {, update_item}*
update_item ::= [identification_variable.]{state_field | single_valued_association_field} =
```

```
new_value
new_value ::=
simple_arithmetic_expression |
string_primary |
datetime_primary |
boolean_primary |
enum_primary
simple_entity_expression |
NULL
delete_clause ::= DELETE FROM abstract_schema_name [[AS] identification_variable]
select_clause ::= SELECT [DISTINCT] select_expression {, select_expression}*
select_expression ::=
single_valued_path_expression |
aggregate_expression |
identification_variable |
OBJECT(identification_variable) |
constructor_expression
constructor_expression ::=
NEW constructor_name ( constructor_item {, constructor_item}* )
constructor_item ::= single_valued_path_expression | aggregate_expression
aggregate_expression ::=
{ AVG | MAX | MIN | SUM } ([DISTINCT] state_field_path_expression) |
COUNT ([DISTINCT] identification_variable | state_field_path_expression |
single_valued_association_path_expression)
where_clause ::= WHERE conditional_expression
groupby_clause ::= GROUP BY groupby_item {, groupby_item}*
groupby_item ::= single_valued_path_expression | identification_variable
having_clause ::= HAVING conditional_expression
orderby_clause ::= ORDER BY orderby_item {, orderby_item}*
orderby_item ::= state_field_path_expression [ ASC | DESC ]
subquery ::= simple_select_clause subquery_from_clause [where_clause]
[groupby_clause] [having_clause]
subquery_from_clause ::=
FROM subselect_identification_variable_declaration
{, subselect_identification_variable_declaration}*
subselect_identification_variable_declaration ::=
identification_variable_declaration |
association_path_expression [AS] identification_variable |
collection_member_declaration
simple_select_clause ::= SELECT [DISTINCT] simple_select_expression
simple_select_expression::=
single_valued_path_expression |
aggregate_expression |
identification_variable
conditional_expression ::= conditional_term | conditional_expression OR conditional_term
conditional_term ::= conditional_factor | conditional_term AND conditional_factor
conditional_factor ::= [ NOT ] conditional_primary
conditional_primary ::= simple_cond_expression | (conditional_expression)
simple_cond_expression ::=
comparison_expression |
between_expression |
like_expression |
in_expression |
null_comparison_expression |
empty_collection_comparison_expression |
collection_member_expression |
exists_expression
between_expression ::=
```

```
arithmetic_expression [NOT] BETWEEN
arithmetic_expression AND arithmetic_expression |
string_expression [NOT] BETWEEN string_expression AND string_expression |
datetime_expression [NOT] BETWEEN
datetime_expression AND datetime_expression
in_expression ::=
state_field_path_expression [NOT] IN ( in_item {, in_item}* | subquery)
in_item ::= literal | input_parameter
like_expression ::=
string_expression [NOT] LIKE pattern_value [ESCAPE escape_character]
null_comparison_expression ::=
{single_valued_path_expression | input_parameter} IS [NOT] NULL
empty_collection_comparison_expression ::=
collection_valued_path_expression IS [NOT] EMPTY
collection_member_expression ::= entity_expression
[NOT] MEMBER [OF] collection_valued_path_expression
exists_expression::= [NOT] EXISTS (subquery)
all_or_any_expression ::= { ALL | ANY | SOME} (subquery)
comparison_expression ::=
string_expression comparison_operator {string_expression | all_or_any_expression} |
boolean_expression { =|<>} {boolean_expression | all_or_any_expression} |
enum_expression { =|<>} {enum_expression | all_or_any_expression} |
datetime_expression comparison_operator
{datetime_expression | all_or_any_expression} |
entity_expression { = | <> } {entity_expression | all_or_any_expression} |
arithmetic_expression comparison_operator
{arithmetic_expression | all_or_any_expression}
comparison_operator ::= = | > | >= | < | <= | <>
arithmetic_expression ::= simple_arithmetic_expression | (subquery)
simple_arithmetic_expression ::=
arithmetic_term | simple_arithmetic_expression { + | - } arithmetic_term
arithmetic_term ::= arithmetic_factor | arithmetic_term { * | / } arithmetic_factor
arithmetic_factor ::= [{ + | - }] arithmetic_primary
arithmetic_primary ::=
state_field_path_expression |
numeric_literal |
(simple_arithmetic_expression) |
input_parameter |
functions_returning_numerics |
aggregate_expression
string_expression ::= string_primary | (subquery)
string_primary ::=
state_field_path_expression |
string_literal |
input_parameter |
functions_returning_strings |
aggregate_expression
datetime_expression ::= datetime_primary | (subquery)
datetime_primary ::=
state_field_path_expression |
input_parameter |
functions_returning_datetime |
aggregate_expression
boolean_expression ::= boolean_primary | (subquery)
boolean_primary ::=
state_field_path_expression |
boolean_literal |
input_parameter |
```

```
enum_expression ::= enum_primary | (subquery)
enum_primary ::=
state_field_path_expression |
enum_literal |
input_parameter |
entity_expression ::=
single_valued_association_path_expression | simple_entity_expression
simple_entity_expression ::=
identification_variable |
input_parameter
functions_returning_numerics::=
LENGTH(string_primary) |
LOCATE(string_primary, string_primary[, simple_arithmetic_expression]) |
ABS(simple_arithmetic_expression) |
SQRT(simple_arithmetic_expression) |
MOD(simple_arithmetic_expression, simple_arithmetic_expression) |
SIZE(collection_valued_path_expression)
functions_returning_datetime ::=
CURRENT_DATE|
CURRENT_TIME |
CURRENT_TIMESTAMP
functions_returning_strings ::=
CONCAT(string_primary, string_primary) |
SUBSTRING(string_primary,
simple_arithmetic_expression, simple_arithmetic_expression)|
TRIM([[trim_specification] [trim_character] FROM] string_primary) |
LOWER(string_primary) |
UPPER(string_primary)
trim_specification ::= LEADING | TRAILING | BOTH
```

付録 H　各バージョンでの主な機能変更

ここでは，11-40 よりも前のアプリケーションサーバの各バージョンでの主な機能の変更について，変更目的ごとに説明します。11-40 での主な機能変更については，「1.3　アプリケーションサーバ 11-40 での主な機能変更」を参照してください。

説明内容は次のとおりです。

- アプリケーションサーバの各バージョンで変更になった主な機能と，その概要を説明しています。機能の詳細については「参照先マニュアル」の「参照個所」の記述を確認してください。「参照先マニュアル」および「参照個所」には，その機能についての 11-40 のマニュアルでの主な記載個所を記載しています。
- 「参照先マニュアル」に示したマニュアル名の「アプリケーションサーバ」は省略しています。

付録 H.1　11-30 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-1　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JDBC4.3 への対応	JDBC4.3 に対応しました。	アプリケーションサーバ & BPM/ESB 基盤 概説	4.6.2
接続できるデータベースに MySQL，PostgreSQL を追加	DB Connector を使用して接続できるデータベースに，MySQL，PostgreSQL を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	3.6
パッケージ名変換機能の追加	jakarta パッケージ名を前提とするアプリケーションをアプリケーションサーバで動作させる機能を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	20 章

付録 H.2　11-20 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-2　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JSF 2.3 への対応	JSF 2.3 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	3 章
JAX-RS 2.1 への対応	JAX-RS 2.1 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	4 章
WebSocket 1.1 への対応	WebSocket 1.1 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	5 章
Servlet 4.0 への対応	Servlet 4.0 に対応しました。これに伴い，NIO HTTP サーバで HTTP/2 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	8 章
CDI Managed Bean での JPA 利用	CDI Managed Bean での JPA 利用に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	5 章
JPA 2.2 への対応	JPA 2.2 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	6 章
CDI 2.0 への対応	CDI 2.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	9 章
WEB-INF/lib 内の CDI への対応	WAR ファイルの WEB-INF/lib 以下の JAR ファイルから CDI を利用できるようにしました。	機能解説 基本・開発編 (コンテナ共通機能)	9 章
BV 2.0 への対応	BV 2.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	10 章
JSON-P 1.1 への対応	JSON-P 1.1 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	12 章
JSON-B 1.0 への対応	JSON-B 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	13 章

付録 H.3　11-10 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 H-3　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ライブラリ競合回避機能の追加	クラス・リソースをロードするときの検索順序を変更し，ユーザアプリケーションに含まれるライブラリを優先して参照できるようにする機能を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	付録 B.4

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H–4　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JSP2.3 への対応	JSP2.3 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	8 章
コンテナ管理の EntityManager への対応	JPA 2.1 に対応したコンテナ管理の EntityManager に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	5 章
Interceptors 1.2 への対応	Interceptors 1.2 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	15 章

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H–5　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
セッションマネージャの指定機能の追加	クラウド環境利用時に HTTP セッションのセッションフェイルオーバを利用できるようにする機能を追加しました。	機能解説 基本・開発編 (Web コンテナ)	2.22

付録 H.4　11-00 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 H–6　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
開発環境の Windows Server 対応	クラウド環境上にアプリケーション開発環境を構築できるよう，uCosminexus Developer のサポート OS に Windows Server OS を追加しました。	−	−

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-7　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 3.0/3.1 への対応	Servlet 3.0 の非同期サーブレット，および Servlet 3.1 の非同期 I/O 系 API に対応しました。	機能解説 基本・開発編 (Web コンテナ)	8.1
EL 3.0 への対応	EL 3.0 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	2.3.3
JSF 2.2 への対応	JSF 2.2 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	3 章
JAX-RS 2.0 への対応	JAX-RS 2.0 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	4 章
WebSocket 1.0 への対応	WebSocket 1.0 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	5 章
NIO HTTP サーバ機能の追加	従来のリダイレクタ機能やインプロセス HTTP サーバ機能に代わり，非同期サーブレットや WebSocket などのノンブロッキング I/O 処理に対応したインプロセスの HTTP サーバとして，NIO HTTP サーバ機能を追加しました。	機能解説 基本・開発編 (Web コンテナ)	7 章
JPA 2.1 への対応	JPA 2.1 に対応し，JPA 2.1 対応の JPA プロバイダを利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	6 章
CDI 1.2 への対応	CDI 1.2 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	9 章
BV 1.1 への対応	Bean Validation 1.1 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	10 章
Java Batch 1.0 への対応	Batch Applications for the Java Platform （Java Batch） 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	11 章
JSON-P 1.0 への対応	Java API for JSON Processing（JSON-P） 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	12 章
Concurrency Utilities 1.0 への対応	Concurrency Utilities for Java EE 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	14 章
WebSocket 通信への対応	HTTP Server から J2EE サーバに WebSocket 通信を中継する機能を追加しました。	HTTP Server	4.15

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H-8　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
暗号化通信モジュールの変更	HTTP Server の暗号化通信モジュールとして mod_ssl を採用しました。	HTTP Server	5 章，付録 D

(4)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 H-9　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
V9 互換モードの追加	Version 9 以前の J2EE サーバから移行するユーザ向けに，Version 9 との互換性を維持するための V9 互換モードを追加しました。	機能解説 保守／移行編	10.3.4

付録 H.5　09-87 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-10　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Java SE 11 への対応	Java SE 11 の機能が使用できるようになりました。	機能解説 保守／移行編	9 章

付録 H.6　09-80 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-11　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JAX-RS 機能でのラムダ式の使用	web.xml のサーブレット初期化パラメタに指定したパッケージとそのサブパッケージに含まれるクラスで，ラムダ式が使用できるようになりました。	Web サービス開発ガイド	11.2
Java SE 9 への対応	Java SE 9 の機能が使用できるようになりました。	機能解説 保守／移行編	9 章

(2) 信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H-12 信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web サーバの Apache2.4 のサポート	Web サーバのベースバージョンとして Apache2.4 をサポートしました。	HTTP Server	6 章，付録 C
SSL 通信での楕円曲線暗号の利用	楕円曲線暗号を利用した SSL 通信ができるようになりました。	HTTP Server	5 章，付録 C
SSL ライブラリの変更	SSL 機能を提供する SSL ライブラリを OpenSSL に変更しました。	HTTP Server	5 章，付録 C

付録 H.7 09-70 での主な機能変更

(1) 標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-13 標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータルでの JSP コンパイルバージョンの追加	J2EE サーバでの JSP から生成されたサーブレットのコンパイル方法に「JDK1.7 の仕様に従ったコンパイル」と「JDK7 の仕様に従ったコンパイル」を追加する。	運用管理ポータル操作ガイド	10.8.4
		リファレンス 定義編(サーバ定義)	4.11.2
JDK8 でのメタスペース対応	JavaVM の起動で使用している Permanent 領域用のオプションを Metaspace 領域用のオプションに変更する。	システム構築・運用ガイド	付録 A.2
		運用管理ポータル操作ガイド	10.8.7
		リファレンス 定義編(サーバ定義)	5.2.1，5.2.2，8.2.3
統合ユーザ管理でのユーザ認証の SHA-2 対応	統合ユーザ管理でのユーザ認証のハッシュアルゴリズムとして SHA-224，SHA-256，SHA-384，SHA-512 を追加する。	機能解説 セキュリティ管理機能編	5.3.1，5.3.9，5.10.7，11.4.3，12.4.3，12.5.3，13.2，14.2.2
Red Hat Enterprise Linux Server 7 での自動起動と自動再起動および自動停止の追加	Red Hat Enterprise Linux Server 7 での Management Server と運用管理エージェントの自動起動と自動再起動および自動停止方法を追加する。	機能解説 運用／監視／連携編	2.6.3，2.6.4，2.6.5
		リファレンス コマンド編	7.2

(2)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 H–14　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
V9.7 へのバージョンアップ対応	バージョンアップ時の JavaVM の起動で使用している Permanent 領域用のオプションを Metaspace 領域用のオプションに変更する手順を追加する。	機能解説 保守／移行編	10.3.1，10.3.2，10.3.5

(3)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 H–15　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
snapshot ログの収集対象	snapshot ログの収集対象として JavaVM イベントログと Management Server のスレッドダンプを追加する。	機能解説 保守／移行編	付録 A.2

付録 H.8　09-60 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H–16　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
G1GC への対応	G1GC を選択できるようになりました。	システム設計ガイド	7.15
		リファレンス 定義編(サーバ定義)	14.5
圧縮オブジェクトポインタ機能への対応	圧縮オブジェクトポインタ機能を使用できるようになりました。	機能解説 保守／移行編	9.16

(2)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H–17　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ファイナライズ滞留解消機能の追加	ファイナライズ処理の滞留を解消でき，OS 資源の解放遅れなどの発生を抑止できるようになりました。	機能解説 保守／移行編	9.17

(3)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 H–18　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
ログファイルの非同期出力機能の追加	ログのファイル出力を非同期でできるようになりました。	リファレンス 定義編(サーバ定義)	14.2

付録 H.9　09-50 での主な機能変更

(1)　開発生産性の向上

開発生産性の向上を目的として変更した項目を次の表に示します。

表 H–19　開発生産性の向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Eclipse セットアップの簡略化	GUI を利用して Eclipse 環境をセットアップできるようになりました。	アプリケーション開発ガイド	1.1.5，2.4
ユーザ拡張性能解析トレースを使ったデバッグ支援	ユーザ拡張性能解析トレース設定ファイルを開発環境で作成できるようになりました。	アプリケーション開発ガイド	1.1.3，6.4

(2)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 H–20　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境でのシステム構成パターンの拡充	仮想化環境で使用できるティアの種類（http-tier，j2ee-tier および ctm-tier）が増えました。これによって，次のシステム構成パターンが構築できるようになりました。	仮想化システム構築・運用ガイド	1.1.2

項目	変更の概要	参照先マニュアル	参照個所
	• Web サーバと J2EE サーバを別のホストに配置するパターン • フロントエンド（サーブレット，JSP）とバックエンド（EJB）を分けて配置するパターン • CTM を使用するパターン		

(3) 標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H–21 標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JDBC 4.0 仕様への対応	DB Connector で JDBC 4.0 仕様の HiRDB Type4 JDBC Driver，および SQL Server の JDBC ドライバに対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	3.6.3
Portable Global JNDI 名での命名規則の緩和	Portable Global JNDI 名に使用できる文字を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	2.4.3
Servlet 3.0 仕様への対応	Servlet 3.0 の HTTP Cookie の名称，および URL のパスパラメタ名の変更が，Servlet 2.5 以前のバージョンでも使用できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.7
Bean Validation と連携できるアプリケーションの適用拡大	CDI や UAP でも Bean Validation を使って検証できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	10 章
JavaMail への対応	JavaMail 1.4 に準拠した API を使用したメール送受信機能を利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	8 章
javacore コマンドが使用できる OS の適用拡大	javacore コマンドを使って，Windows のスレッドダンプを取得できるようになりました。	リファレンス コマンド編	javacore (スレッドダンプの取得／Windows の場合)

(4) 信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H–22 信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
コードキャッシュ領域の枯渇回避	システムで使用しているコードキャッシュ領域のサイズを確認して，領域が枯渇する前にしきい値を変更して領域枯渇するのを回避できるようになりました。	システム設計ガイド	7.2.6
		機能解説 保守／移行編	5.7.2, 5.7.3

項目	変更の概要	参照先マニュアル	参照個所
		リファレンス 定義編(サーバ定義)	14.1, 14.2, 14.4
明示管理ヒープ機能の効率的な適用への対応	自動解放処理時間を短縮し，明示管理ヒープ機能を効率的に適用するための機能として，Explicit ヒープに移動するオブジェクトを制御できる機能を追加しました。 • Explicit メモリブロックへのオブジェクト移動制御機能 • 明示管理ヒープ機能適用除外クラス指定機能 • Explicit ヒープ情報へのオブジェクト解放率情報の出力	システム設計ガイド	7.14.6
		機能解説 拡張編	7.2.2, 7.6.5, 7.10, 7.13.1, 7.13.3
		機能解説 保守／移行編	5.5
クラス別統計情報の出力範囲拡大	クラス別統計情報を含んだ拡張スレッドダンプに，staticフィールドを基点とした参照関係を出力できるようになりました。	機能解説 保守／移行編	9.6

(5) 運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 H-23 運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
EADs セッションフェイルオーバ機能のサポート	EADs と連携してセッションフェイルオーバ機能を実現する EADs セッションフェイルオーバ機能をサポートしました。	機能解説 拡張編	5 章
WAR による運用	WAR ファイルだけで構成された WAR アプリケーションを J2EE サーバにデプロイできるようになりました。	機能解説 基本・開発編(Web コンテナ)	2.2.1
		機能解説 基本・開発編(コンテナ共通機能)	18.9
		リファレンス コマンド編	cjimportwar(WAR アプリケーションのインポート)
運用管理機能の同期実行による起動と停止	運用管理機能（Management Server および運用管理エージェント）の起動および停止を，同期実行するオプションを追加しました。	機能解説 運用／監視／連携編	2.6.1, 2.6.2, 2.6.3, 2.6.4
		リファレンス コマンド編	adminagentctl(運用管理

項目	変更の概要	参照先マニュアル	参照個所
			エージェントの起動と停止), mngautorun（自動起動および自動再起動の設定／設定解除), mngsvrctl (Management Server の起動／停止／セットアップ)
明示管理ヒープ機能での Explicit メモリブロックの強制解放	javagc コマンドで，Explicit メモリブロックの解放処理を任意のタイミングで実行できるようになりました。	機能解説 拡張編	7.6.1, 7.9
		リファレンス コマンド編	javagc (GC の強制発生)

(6) そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 H-24 そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
定義情報の取得	snapshotlog（snapshot ログの収集）コマンドで定義ファイルだけを収集できるようになりました。	機能解説 保守／移行編	2.3
		リファレンス コマンド編	snapshotlog (snapshot ログの収集)
cjenvsetup コマンドのログ出力	Component Container 管理者のセットアップ（cjenvsetup コマンド）の実行情報がメッセージログに出力されるようになりました。	システム構築・運用ガイド	4.1.4
		機能解説 保守／移行編	4.20
		リファレンス コマンド編	cjenvsetup (Compo

項目	変更の概要	参照先マニュアル	参照個所
			nent Container 管理者のセットアップ)
BIG-IP v11 のサポート	使用できる負荷分散機の種類に BIG-IP v11 が追加になりました。	システム構築・運用ガイド	4.7.2
		仮想化システム構築・運用ガイド	2.1
明示管理ヒープ機能のイベントログへの CPU 時間の出力	Explicit メモリブロック解放処理に掛かった CPU 時間が，明示管理ヒープ機能のイベントログに出力されるようになりました。	機能解説 保守／移行編	5.11.3
ユーザ拡張性能解析トレースの機能拡張	ユーザ拡張性能解析トレースで，次の機能を追加しました。 • トレース対象の指定方法を通常のメソッド単位の指定方法に加えて，パッケージ単位またはクラス単位で指定できるようになりました。 • 使用できるイベント ID の範囲を拡張しました。 • ユーザ拡張性能解析トレース設定ファイルに指定できる行数の制限を緩和しました。 • ユーザ拡張性能解析トレース設定ファイルでトレース取得レベルを指定できるようになりました。	機能解説 保守／移行編	7.5.2, 7.5.3, 8.25.1
Session Bean の非同期呼び出し使用時の情報解析向上	PRF トレースのルートアプリケーション情報を使用して，呼び出し元と呼び出し先のリクエストを突き合わせることができるようになりました。	機能解説 基本・開発編（EJB コンテナ）	2.17.3

付録 H.10　09-00 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 H-25　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での構築・運用の操作対象単位の変更	仮想化環境の構築・運用時の操作対象単位が仮想サーバから仮想サーバグループへ変更になりました。仮想サーバグループの情報を定義したファイルを使用して，複数の仮想サーバを管理ユニットへ一括で登録できるようになりました。	仮想化システム構築・運用ガイド	1.1.2

項目	変更の概要	参照先マニュアル	参照個所
セットアップウィザードによる構築環境の制限解除	セットアップウィザードを使用して構築できる環境の制限が解除されました。ほかの機能で構築した環境があってもアンセットアップされて，セットアップウィザードで構築できるようになりました。	システム構築・運用ガイド	2.2.7
構築環境の削除手順の簡略化	Management Server を使用して構築したシステム環境を削除する機能（mngunsetup コマンド）の追加によって，削除手順を簡略化しました。	システム構築・運用ガイド	4.1.37
		運用管理ポータル操作ガイド	3.6，5.4
		リファレンス コマンド編	mngunsetup（Management Server の構築環境の削除）

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H–26　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 3.0 への対応	Servlet 3.0 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	8 章
EJB 3.1 への対応	EJB 3.1 に対応しました。	機能解説 基本・開発編 (EJB コンテナ)	2 章
JSF 2.1 への対応	JSF 2.1 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	3 章
JSTL 1.2 への対応	JSTL 1.2 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	3 章
CDI 1.0 への対応	CDI 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	9 章
Portable Global JNDI 名の利用	Portable Global JNDI 名を利用したオブジェクトのルックアップができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	2.4
JAX-WS 2.2 への対応	JAX-WS 2.2 に対応しました。	Web サービス開発ガイド	1.1, 16.1.5, 16.1.7, 16.2.1, 16.2.6, 16.2.10, 16.2.12, 16.2.13,

項目	変更の概要	参照先マニュアル	参照個所
			16.2.14, 16.2.16, 16.2.17, 16.2.18, 16.2.20, 16.2.22, 19.1, 19.2.3, 37.2, 37.6.1, 37.6.2, 37.6.3
JAX-RS 1.1 への対応	JAX-RS 1.1 に対応しました。	Web サービス開発ガイド	1.1, 1.2.2, 1.3.2, 1.4.2, 1.5.1, 1.6，2.3, 11 章, 12 章, 13 章, 17 章, 24 章, 39 章

(3) 信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H–27 信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
SSL/TLS 通信での TLSv1.2 の使用	RSA BSAFE SSL-J を使用して，TLSv1.2 を含むセキュリティ・プロトコルで SSL/TLS 通信ができるようになりました。	－	－

（凡例）－：09-70 で削除された機能です。

(4) 運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 H–28 運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web コンテナ全体の実行待ちキューの総和の監視	Web コンテナ全体の実行待ちキューの総和を稼働情報に出力して監視できるようになりました。	機能解説 運用／監視／連携編	3 章

項目	変更の概要	参照先マニュアル	参照個所
アプリケーションの性能解析トレース（ユーザ拡張トレース）の出力	ユーザが開発したアプリケーションの処理性能を解析するための性能解析トレースを，アプリケーションの変更をしないで出力できるようになりました。	機能解説 保守／移行編	7 章
仮想化環境でのユーザスクリプトを使用した運用	任意のタイミングでユーザ作成のスクリプト（ユーザスクリプト）を仮想サーバ上で実行できるようになりました。	仮想化システム構築・運用ガイド	7.8
運用管理ポータル改善	運用管理ポータルの次の画面で，手順を示すメッセージを画面に表示するように変更しました。 • ［設定情報の配布］画面 • Web サーバ，J2EE サーバおよび SFO サーバの起動画面 • Web サーバクラスタと J2EE サーバクラスタの一括起動，一括再起動および起動画面	運用管理ポータル操作ガイド	10.10.1, 11.9.2, 11.10.2, 11.10.4, 11.10.6, 11.11.2, 11.12.2, 11.12.4, 11.12.6
運用管理機能の再起動機能の追加	運用管理機能（Management Server および運用管理エージェント）で自動再起動が設定できるようになり，運用管理機能で障害が発生した場合でも運用が継続できるようになりました。また，自動起動の設定方法も変更になりました。	機能解説 運用／監視／連携編	2.4.1, 2.4.2, 2.6.3, 2.6.4
		リファレンス コマンド編	mngautorun（自動起動および自動再起動の設定／設定解除）

(5) そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 H-29 そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
ログ出力時のファイル切り替え単位の変更	ログ出力時に，日付ごとに出力先のファイルを切り替えられるようになりました。	機能解説 保守／移行編	3.2.1
Web サーバの名称の変更	アプリケーションサーバに含まれる Web サーバの名称を HTTP Server に変更しました。	HTTP Server	－
BIG-IP の API（SOAP アーキテクチャ）を使用した直接接続への対応	BIG-IP（負荷分散機）で API（SOAP アーキテクチャ）を使用した直接接続に対応しました。 また，API を使用した直接接続を使用する場合に，負荷分散機の接続環境を設定する方法が変更になりました。	システム構築・運用ガイド	4.7.3，付録 J
		仮想化システム構築・運用ガイド	2.1，付録 C

項目	変更の概要	参照先マニュアル	参照個所
		機能解説 セキュリティ管理機能編	8.2，8.4，8.5，8.6，18.2.1，18.2.2，18.2.3

（凡例）－：マニュアル全体を参照する

付録 H.11　08-70 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 H-30　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータル改善	運用管理ポータルの画面で，リソースアダプタの属性を定義するプロパティ（Connector 属性ファイルの設定内容）の設定，および接続テストができるようになりました。また，運用管理ポータルの画面で，J2EE アプリケーション（ear ファイルおよび zip ファイル）を Management Server にアップロードできるようになりました。	ファーストステップガイド	3.5
		運用管理ポータル操作ガイド	－
page/tag ディレクティブの import 属性暗黙インポート機能の追加	page/tag ディレクティブの import 属性暗黙インポート機能を使用できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.3.7
仮想化環境での JP1 製品に対する環境設定の自動化対応	仮想サーバへのアプリケーションサーバ構築時に，仮想サーバに対する JP1 製品の環境設定を，フックスクリプトで自動的に設定できるようになりました。	仮想化システム構築・運用ガイド	7.7.2
統合ユーザ管理機能の改善	ユーザ情報リポジトリでデータベースを使用する場合に，データベース製品の JDBC ドライバを使用して，データベースに接続できるようになりました。 DABroker Library の JDBC ドライバによるデータベース接続はサポート外になりました。 簡易構築定義ファイルおよび運用管理ポータルの画面で，統合ユーザ管理機能に関する設定ができるようになりました。 また，Active Directory の場合，DN で日本語などの 2 バイト文字に対応しました。	機能解説 セキュリティ管理機能編	5 章，14.2.2
		運用管理ポータル操作ガイド	3.5，10.8.1
HTTP Server 設定項目の拡充	簡易構築定義ファイルおよび運用管理ポータルの画面で，HTTP Server の動作環境を定義するディレクティブ（httpsd.conf の設定内容）を直接設定できるようになりました。	システム構築・運用ガイド	4.1.21

項目	変更の概要	参照先マニュアル	参照個所
		運用管理ポータル操作ガイド	10.9.1
		リファレンス 定義編(サーバ定義)	4.10

（凡例）－：マニュアル全体を参照する

(2) 標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 H-31 標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ejb-jar.xml の指定項目の追加	ejb-jar.xml に，クラスレベルインターセプタおよびメソッドレベルインターセプタを指定できるようになりました。	機能解説 基本・開発編(EJB コンテナ)	2.15
パラレルコピーガーベージコレクションへの対応	パラレルコピーガーベージコレクションを選択できるようになりました。	リファレンス 定義編(サーバ定義)	14.5
Connector 1.5 仕様に準拠した Inbound リソースアダプタのグローバルトランザクションへの対応	Connector 1.5 仕様に準拠したリソースアダプタで Transacted Delivery を使用できるようにしました。これによって，Message-driven Bean を呼び出す EIS がグローバルトランザクションに参加できるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	3.16.3
TP1 インバウンドアダプタの MHP への対応	TP1 インバウンドアダプタを使用してアプリケーションサーバを呼び出す OpenTP1 のクライアントとして，MHP を使用できるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	4 章
cjrarupdate コマンドの FTP インバウンドアダプタへの対応	cjrarupdate コマンドでバージョンアップできるリソースアダプタに FTP インバウンドアダプタを追加しました。	リファレンス コマンド編	2.2

(3) 信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 H-32 信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
データベースセッションフェイルオーバ機能の改善	性能を重視するシステムで，グローバルセッション情報を格納したデータベースのロックを取得しないモードを選択できるようになりました。また，データベースを更新しない，参照専用のリクエストを定義できるようになりました。	機能解説 拡張編	6 章

項目	変更の概要	参照先マニュアル	参照個所
OutOfMemory ハンドリング機能の対象となる処理の拡大	OutOfMemory ハンドリング機能の対象となる処理を追加しました。	機能解説 保守／移行編	2.5.4
		リファレンス 定義編 (サーバ定義)	14.2
HTTP セッションで利用する Explicit ヒープの省メモリ化機能の追加	HTTP セッションで利用する Explicit ヒープのメモリ使用量を抑止する機能を追加しました。	機能解説 拡張編	7.11

(4) 運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 H–33 運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での JP1 製品を使用したユーザ認証への対応（クラウド運用対応）	JP1 連携時に，JP1 製品の認証サーバを利用して，仮想サーバマネージャを使用するユーザを管理・認証できるようになりました。	仮想化システム構築・運用ガイド	1.2.2，3 章，4 章，5 章，6 章，7.9

(5) そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 H–34 そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
負荷分散機への API（REST アーキテクチャ）を使用した直接接続の対応	負荷分散機への接続方法として，API（REST アーキテクチャ）を使用した直接接続に対応しました。 また，使用できる負荷分散機の種類に ACOS（AX2500）が追加になりました。	システム構築・運用ガイド	4.7.2，4.7.3
		仮想化システム構築・運用ガイド	2.1
		リファレンス 定義編 (サーバ定義)	4.2.4
snapshot ログ収集時のタイムアウトへの対応と収集対象の改善	snapshot ログの収集が指定した時間で終了（タイムアウト）できるようになりました。一次送付資料として収集される内容が変更になりました。	機能解説 保守／移行編	付録 A

付録 H.12 08-53 での主な機能変更

(1) 導入・構築の容易性強化

導入・構築の容易性強化を目的として変更された項目を次の表に示します。

表 H-35　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
さまざまなハイパーバイザに対応した仮想化環境の構築	さまざまなハイパーバイザを使用して実現する仮想サーバ上に，アプリケーションサーバを構築できるようになりました。 また，複数のハイパーバイザが混在する環境にも対応しました。	仮想化システム構築・運用ガイド	2 章，3 章，5 章

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更された項目を次の表に示します。

表 H-36　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
トランザクション連携に対応した OpenTP1 からの呼び出し	OpenTP1 からアプリケーションサーバ上で動作する Message-driven Bean を呼び出すときに，トランザクション連携ができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	4 章
JavaMail	POP3 に準拠したメールサーバと連携して，JavaMail 1.3 に準拠した API を使用したメール受信機能を利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	8 章

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更された項目を次の表に示します。

表 H-37　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JavaVM のトラブルシュート機能強化	JavaVM のトラブルシュート機能として，次の機能が使用できるようになりました。 • OutOfMemoryError 発生時の動作を変更できるようになりました。 • JIT コンパイル時に，C ヒープ確保量の上限値を設定できるようになりました。 • スレッド数の上限値を設定できるようになりました。 • 拡張 verbosegc 情報の出力項目を拡張しました。	機能解説 保守／移行編	4 章，5 章，9 章

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更された項目を次の表に示します。

表 H-38　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JP1/ITRM への対応	IT リソースを一元管理する製品である JP1/ITRM に対応しました。	仮想化システム構築・運用ガイド	1.3，2.1

(5)　そのほかの目的

そのほかの目的で変更された項目を次の表に示します。

表 H-39　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
Microsoft IIS 7.0 および Microsoft IIS 7.5 への対応	Web サーバとして Microsoft IIS 7.0 および Microsoft IIS 7.5 に対応しました。	－	－
HiRDB Version 9 および SQL Server 2008 への対応	データベースとして次の製品に対応しました。 • HiRDB Server Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 また，SQL Server 2008 に対応する JDBC ドライバとして，SQL Server JDBC Driver に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	3 章

（凡例）－：該当なし。

付録 H.13　08-50 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更された項目を次の表に示します。

表 H-40　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web サービスプロバイダ側での web.xml の指定必須タグの変更	Web サービスプロバイダ側での web.xml で，listener タグ，servlet タグ，および servlet-mapping タグの指定を必須から任意に変更しました。	リファレンス 定義編 (サーバ定義)	2.2.3
論理サーバのネットワークリソース使用	J2EE アプリケーションからほかのホスト上にあるネットワークリソースやネットワークドライブにアクセスするための機能を追加しました。	機能解説 運用／監視／連携編	1.2.3，5.2，5.7
サンプルプログラムの実行手順の簡略化	一部のサンプルプログラムを EAR 形式で提供することによって，サンプルプログラムの実行手順を簡略化しました。	ファーストステップガイド	3.5

項目	変更の概要	参照先マニュアル	参照個所
		システム構築・運用ガイド	付録 L
運用管理ポータルの画面の動作の改善	画面の更新間隔のデフォルトを「更新しない」から「3秒」に変更しました。	運用管理ポータル操作ガイド	7.4.1
セットアップウィザードの完了画面の改善	セットアップウィザード完了時の画面に，セットアップで使用した簡易構築定義ファイルおよび Connector 属性ファイルが表示されるようになりました。	システム構築・運用ガイド	2.2.6
仮想化環境の構築	ハイパーバイザを使用して実現する仮想サーバ上に，アプリケーションサーバを構築する手順を追加しました。	仮想化システム構築・運用ガイド	3 章，5 章

(2) 標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更された項目を次の表に示します。

表 H–41 標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
OpenTP1 からの呼び出しへの対応	OpenTP1 からアプリケーションサーバ上で動作する Message-driven Bean を呼び出せるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	4 章
JMS への対応	JMS 1.1 仕様に対応した CJMS プロバイダ機能を使用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	7 章
Java SE 6 への対応	Java SE 6 の機能が使用できるようになりました。	機能解説 保守／移行編	5.5, 5.8.1
ジェネリクスの使用への対応	EJB でジェネリクスを使用できるようになりました。	機能解説 基本・開発編 (EJB コンテナ)	4.2.18

(3) 信頼性の維持・向上

信頼性の維持・向上を目的として変更された項目を次の表に示します。

表 H–42 信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
明示管理ヒープ機能の使用性向上	自動配置設定ファイルを使用して，明示管理ヒープ機能を容易に使用できるようになりました。	システム設計ガイド	7.2, 7.7.3, 7.11.4, 7.12.1
		機能解説 拡張編	7 章
データベースセッションフェイルオーバ機能の URI 単位での抑止	データベースセッションフェイルオーバ機能を使用する場合に，機能の対象外にするリクエストを URI 単位で指定できるようになりました。	機能解説 拡張編	5.6.1

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での障害監視	仮想化システムで，仮想サーバを監視し，障害の発生を検知できるようになりました。	仮想化システム構築・運用ガイド	－

(4) 運用性の維持・向上

運用性の維持・向上を目的として変更された項目を次の表に示します。

表 H-43 運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
管理ユーザアカウントの省略	運用管理ポータル，Management Server のコマンド，または Smart Composer 機能のコマンドで，ユーザのログイン ID およびパスワードの入力を省略できるようになりました。	システム構築・運用ガイド	4.1.15
		運用管理ポータル操作ガイド	2.2，7.1.1，7.1.2，7.1.3，8.1，8.2.1，付録 E.2
		リファレンス コマンド編	1.4，mngsvrctl (Management Server の起動／停止／セットアップ)，mngsvrutil (Management Server の運用管理コマンド)，8.3，cmx_admin_passwd (Management Server の管理ユーザアカウントの設定)
仮想化環境の運用	仮想化システムで，複数の仮想サーバを対象にした一括起動・一括停止，スケールイン・スケールアウトなどを実行する手順を追加しました。	仮想化システム構築・運用ガイド	4 章，6 章

(5) そのほかの目的

そのほかの目的で変更された項目を次の表に示します。

表 H-44 そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
Tenured 領域内不要オブジェクト統計機能	Tenured 領域内で不要となったオブジェクトだけを特定できるようになりました。	機能解説 保守／移行編	9.8
Tenured 増加要因の基点オブジェクトリスト出力機能	Tenured 領域内不要オブジェクト統計機能を使って特定した，不要オブジェクトの基点となるオブジェクトの情報を出力できるようになりました。		9.9

項目	変更の概要	参照先マニュアル	参照個所
クラス別統計情報解析機能	クラス別統計情報を CSV 形式で出力できるようになりました。		9.10
論理サーバの自動再起動回数オーバー検知によるクラスタ系切り替え	Management Server を系切り替えの監視対象としているクラスタ構成の場合，論理サーバが異常停止状態(自動再起動回数をオーバーした状態，または自動再起動回数の設定が 0 のときに障害を検知した状態)になったタイミングでの系切り替えができるようになりました。	機能解説 運用／監視／連携編	18.4.3, 18.5.3, 16.2.2, 16.3.3, 16.3.4
ホスト単位管理モデルを対象とした系切り替えシステム	クラスタソフトウェアと連携したシステム運用で，ホスト単位管理モデルを対象にした系切り替えができるようになりました。		16 章
ACOS（AX2000，BS320）のサポート	使用できる負荷分散機の種類に ACOS（AX2000，BS320）が追加になりました。	システム構築・運用ガイド	4.7.2, 4.7.3, 4.7.5, 4.7.6，付録 J，付録 J.2
		リファレンス 定義編(サーバ定義)	4.2.4, 4.3.2, 4.3.4, 4.3.5, 4.3.6, 4.7.1
CMT でトランザクション管理をする場合に Stateful Session Bean（SessionSynchronization）に指定できるトランザクション属性の追加	CMT でトランザクション管理をする場合に，Stateful Session Bean（SessionSynchronization）にトランザクション属性として Supports，NotSupported および Never を指定できるようになりました。	機能解説 基本・開発編(EJB コンテナ)	2.7.3
OutOfMemoryError 発生時の運用管理エージェントの強制終了	JavaVM で OutOfMemoryError が発生したときに，運用管理エージェントが強制終了するようになりました。	機能解説 保守／移行編	2.5.5
スレッドの非同期並行処理	TimerManager および WorkManager を使用して，非同期タイマ処理および非同期スレッド処理を実現できるようになりました。	機能解説 拡張編	－

付録 H.14　08-00 での主な機能変更

(1)　開発生産性の向上

開発生産性の向上を目的として変更された項目を次の表に示します。

表 H-45　開発生産性の向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ほかのアプリケーションサーバ製品からの移行容易化	ほかのアプリケーションサーバ製品からの移行を円滑に実施するため，次の機能を使用できるようになりました。 • HTTP セッションの上限が例外で判定できるようになりました。 • JavaBeans の ID が重複している場合や，カスタムタグの属性名と TLD の定義で大文字・小文字が異なる場合に，トランスレーションエラーが発生することを抑止できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.3, 2.7.5
cosminexus.xml の提供	アプリケーションサーバ独自の属性を cosminexus.xml に記載することによって，J2EE アプリケーションを J2EE サーバにインポート後，プロパティの設定をしないで開始できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	16.3

(2)　標準機能への対応

標準機能への対応を目的として変更された項目を次の表に示します。

表 H-46　標準機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 2.5 への対応	Servlet 2.5 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	2.2, 2.5.4, 2.6，8 章
JSP 2.1 への対応	JSP 2.1 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	2.3.1, 2.3.3, 2.5, 2.6，8 章
JSP デバッグ	MyEclipse を使用した開発環境で JSP デバッグができるようになりました。※	機能解説 基本・開発編 (Web コンテナ)	2.4
タグライブラリのライブラリ JAR への格納と TLD のマッピング	タグライブラリをライブラリ JAR に格納した場合に，Web アプリケーション開始時に Web コンテナによってライブラリ JAR 内の TLD ファイルを検索し，自動的にマッピングできるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.3.4
application.xml の省略	J2EE アプリケーションで application.xml が省略できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	16.4
アノテーションと DD の併用	アノテーションと DD を併用できるようになり，アノテーションで指定した内容を DD で更新できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	17.5

項目	変更の概要	参照先マニュアル	参照個所
アノテーションの Java EE 5 標準準拠（デフォルトインターセプタ）	デフォルトインターセプタをライブラリ JAR に格納できるようになりました。また，デフォルトインターセプタから DI できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	16.4
@Resource の参照解決	@Resource でリソースの参照解決ができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	17.4
JPA への対応	JPA 仕様に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	6 章

注※　09-00 以降では，WTP を使用した開発環境で JSP デバッグ機能を使用できます。

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更された項目を次の表に示します。

表 H–47　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
セッション情報の永続化	HTTP セッションのセッション情報をデータベースに保存して引き継げるようになりました。	機能解説 拡張編	5 章，6 章
FullGC の抑止	FullGC の要因となるオブジェクトを Java ヒープ外に配置することで，フ FullGC 発生を抑止できるようになりました。	機能解説 拡張編	7 章
クライアント性能モニタ	クライアント処理に掛かった時間を調査・分析できるようになりました。	－	－

（凡例）－：09-00 で削除された機能です。

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更された項目を次の表に示します。

表 H–48　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータルでのアプリケーション操作性向上	アプリケーションおよびリソースの操作について，サーバ管理コマンドと運用管理ポータルの相互運用ができるようになりました。	運用管理ポータル操作ガイド	1.1.3

(5)　そのほかの目的

そのほかの目的で変更された項目を次の表に示します。

表 H-49　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
無効な HTTP Cookie の削除	無効な HTTP Cookie を削除できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.7.4
ネーミングサービスの障害検知	ネーミングサービスの障害が発生した場合に，EJB クライアントが，より早くエラーを検知できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	2.9
コネクション障害検知タイムアウト	コネクション障害検知タイムアウトのタイムアウト時間を指定できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3.15.1
Oracle11g への対応	データベースとして Oracle11g が使用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3 章
バッチ処理のスケジューリング	バッチアプリケーションの実行を CTM によってスケジューリングできるようになりました。	機能解説 拡張編	4 章
バッチ処理のログ	バッチ実行コマンドのログファイルのサイズ，面数，ログの排他処理失敗時のリトライ回数とリトライ間隔を指定できるようになりました。	リファレンス 定義編 (サーバ定義)	3.2.5
snapshot ログ	snapshot ログの収集内容が変更されました。	機能解説 保守／移行編	付録 A.1，付録 A.2
メソッドキャンセルの保護区公開	メソッドキャンセルの対象外となる保護区リストの内容を公開しました。	機能解説 運用／監視／連携編	付録 C
統計前のガーベージコレクション選択機能	クラス別統計情報を出力する前に，ガーベージコレクションを実行するかどうかを選択できるようになりました。	機能解説 保守／移行編	9.7
Survivor 領域の年齢分布情報出力機能	Survivor 領域の Java オブジェクトの年齢分布情報を JavaVM ログファイルに出力できるようになりました。	機能解説 保守／移行編	9.11
ファイナライズ滞留解消機能	JavaVM のファイナライズ処理の状態を監視して，処理の滞留を解消できるようになりました。	－	－
サーバ管理コマンドの最大ヒープサイズの変更	サーバ管理コマンドが使用する最大ヒープサイズが変更されました。	リファレンス 定義編 (サーバ定義)	5.2.1，5.2.2
推奨しない表示名を指定された場合の対応	J2EE アプリケーションで推奨しない表示名を指定された場合にメッセージが出力されるようになりました。	メッセージ(構築／運用／開発用)	KDJE42374-W

（凡例）－：09-00 で削除された機能です。

付録 I　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

記号

数字

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

う

え

か

き

く

ひ

ふ

ほ

ま

め

も

ゆ

よ

ら

り

る

れ

ろ

わ