HITACHI
Inspire the Next

# Cosminexus V11 アプリケーションサーバ 機能解説 拡張編

解説書

3021-3-J08-30

# 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

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

### ■ マイクロソフト製品のスクリーンショットの使用について

マイクロソフトの許可を得て使用しています。

### ■ 発行

2022 年 3 月 3021-3-J08-30

### ■ 著作権

# 変更内容

## 変更内容(3021-3-J08-30) uCosminexus Application Server 11-20，uCosminexus Client 11-20，uCosminexus Developer 11-20，uCosminexus Service Architect 11-20，uCosminexus Service Platform 11-20

| 追加・変更内容 | 変更個所 |
|---|---|
| Java EE 8 に対応した。 | 1.1 |
| アプリケーションサーバが対応している標準仕様に，JAX-RS 2.1，JPA 2.2，JSON-B 1.0 を追加した。 | 1.1.3 |
| アプリケーションサーバ 11-20 での主な機能変更についての説明を追加した。 | 1.4 |
| Windows Server 2016 の記述を削除した。また，Windows Server 2022 の記述を追加した。 | 2.2.3(3)(b) |
| 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。

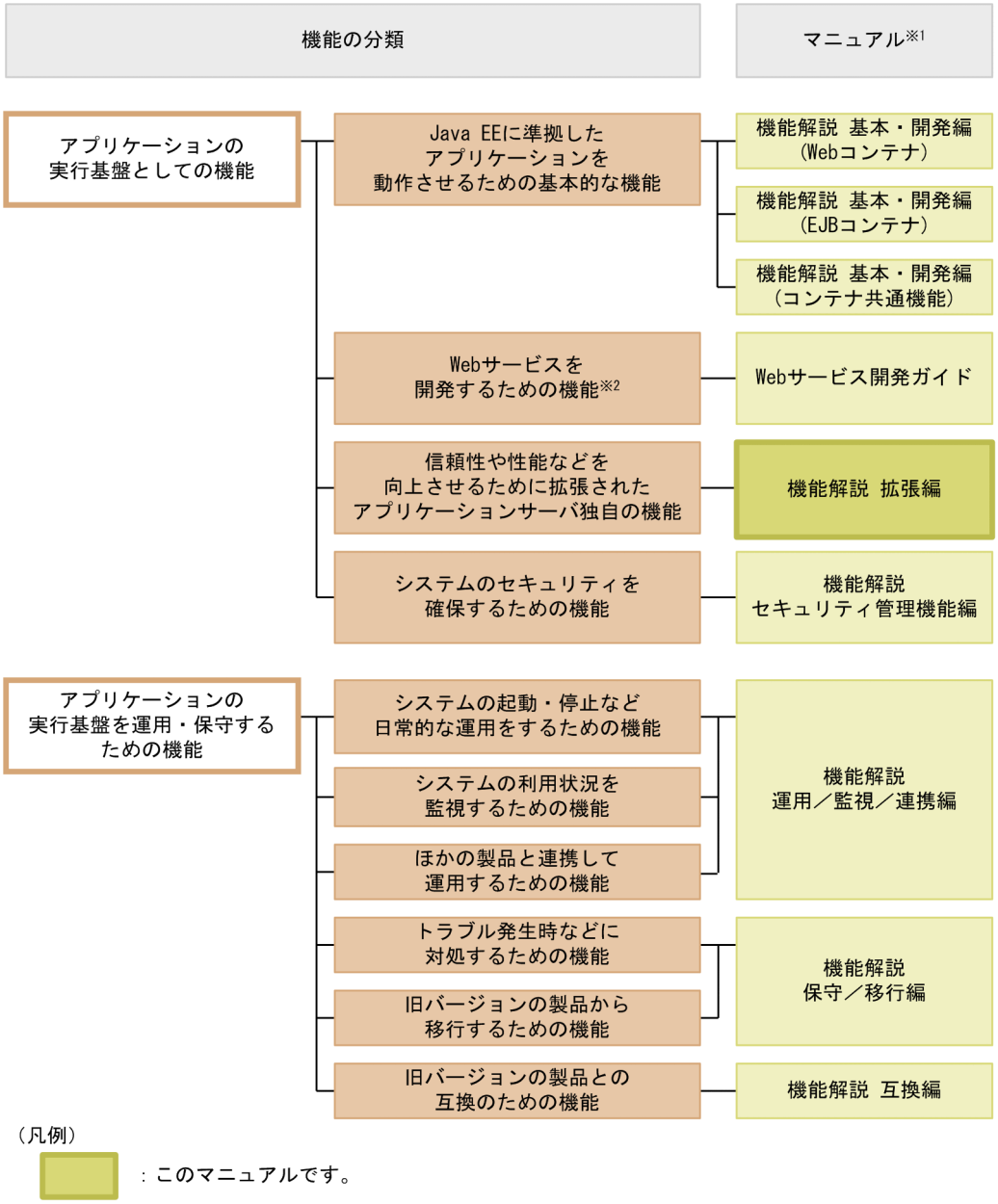
アプリケーションサーバの機能は，大きく分けて，次の二つに分類できます。

- 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 | |
| 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※2 |
| | 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 | |
| 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 | |
| | 監査ログ出力機能 | |
| | データベース監査証跡連携機能 | |

| 分類 | 機能 | マニュアル※1 |
|---|---|---|
| | 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） | |
| | 監査ログの収集および一元管理（JP1/Audit Management - Manager との連携） | |
| | 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） | |
| | 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） | |
| 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 |
| | 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　システムの目的と機能の対応

アプリケーションサーバでは，構築・運用するシステムの目的に合わせて，適用する機能を選択する必要があります。

この節では，アプリケーションサーバで拡張された各機能をどのようなシステムの場合に使用するとよいかを示します。機能ごとに，次の項目への対応を示しています。

- **信頼性**
  高い信頼が求められるシステムの場合に使用するとよい機能です。
  アベイラビリティ（安定稼働性）およびフォールトトレランス（耐障害性）を高める機能や，ユーザ認証などのセキュリティを高めるための機能が該当します。
- **性能**
  性能を重視したシステムの場合に使用するとよい機能です。
  システムのパフォーマンスチューニングで使用する機能などが該当します。
- **運用・保守**
  効率の良い運用・保守をしたい場合に使用するとよい機能です。
- **拡張性**
  システム規模の拡大・縮小および構成の変更への柔軟な対応が必要な場合に使用するとよい機能です。
- **そのほか**
  そのほかの個別の目的に対応するための機能です。

また，アプリケーションサーバで拡張された機能には，Java EE 標準機能とアプリケーションサーバが独自に拡張した機能があります。機能を選択するときには，必要に応じて，Java EE 標準への準拠についても確認してください。

### 1.2.1　バッチアプリケーション実行時に使用する機能

バッチアプリケーション実行時に使用する機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

表 1-2　バッチアプリケーション実行時に使用する機能とシステムの目的の対応

| 機能 | | システムの目的 | | | | | Java EE 標準への準拠 | | 参照先 |
|---|---|---|---|---|---|---|---|---|---|
| | | 信頼性 | 性能 | 運用・保守 | 拡張性 | そのほか | 標準 | 拡張 | |
| バッチアプリケーション実行機能 | バッチアプリケーションの実行 | － | － | － | － | － | － | ○ | 2.3.1, 2.3.2 |

| 機能 | | システムの目的 | | | | | Java EE 標準への準拠 | | 参照先 |
|---|---|---|---|---|---|---|---|---|---|
| | | 信頼性 | 性能 | 運用・保守 | 拡張性 | そのほか | 標準 | 拡張 | |
| | バッチアプリケーションの強制停止 | − | − | ○ | − | − | − | ○ | 2.3.3 |
| | バッチアプリケーション情報の一覧表示 | − | − | ○ | − | − | − | ○ | 2.3.4 |
| | バッチアプリケーションのログ出力 | − | − | ○ | − | − | − | ○ | 2.3.5 |
| EJB アクセス機能 | Enterprise Bean の呼び出し | − | − | − | ○ | − | ○ | ○ | 2.4 |
| | JNDI による EJB ホームオブジェクト・ビジネスインタフェースのリファレンスのルックアップ | − | − | − | ○ | − | ○ | ○ | |
| | トランザクションの実装 | − | − | − | ○ | − | ○ | ○ | |
| | RMI-IIOP 通信のタイムアウト | − | − | − | ○ | − | ○ | ○ | |
| | RMI-IIOP スタブ，インタフェースの取得 | − | − | − | ○ | − | ○ | ○ | |
| ネーミング管理が提供する機能 | JNDI 名前空間へのオブジェクトのバインドとルックアップ | − | − | − | ○ | − | ○ | ○ | 2.5※ |
| | J2EE リソースへの別名付与（ユーザ指定名前空間機能） | − | − | − | ○ | − | − | ○ | |
| | ラウンドロビンポリシーによる CORBA ネーミングサービスの検索 | − | − | − | ○ | − | − | ○ | |
| | ネーミング管理機能でのキャッシング | − | ○ | − | − | − | − | ○ | |
| | CORBA ネーミングサービスの切り替え | − | − | − | ○ | − | − | ○ | |
| リソース接続とトランザクション管理が提供する機能 | コネクションプーリング | − | ○ | − | − | − | ○ | ○ | 2.7, 2.8 |
| | 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数調節機能 | − | ○ | − | − | − | − | ○ | |
| | コネクションシェアリング・アソシエーション | − | ○ | − | − | − | ○ | − | |

| 機能 | | システムの目的 | | | | | Java EE 標準への準拠 | | 参照先 |
|---|---|---|---|---|---|---|---|---|---|
| | | 信頼性 | 性能 | 運用・保守 | 拡張性 | そのほか | 標準 | 拡張 | |
| | ステートメントプーリング | − | ○ | − | − | − | − | ○ | |
| | ライトトランザクション | − | ○ | − | − | − | − | ○ | |
| | 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 | ○ | − | − | − | − | − | ○ | |
| | コネクションの取得リトライ | ○ | − | − | − | − | − | ○ | |
| | コネクションプールの情報表示 | ○ | − | − | − | − | − | ○ | |
| | コネクションプールのクリア | ○ | − | − | − | − | − | ○ | |
| | トランザクションタイムアウトとステートメントキャンセル | ○ | − | − | − | − | ○ | − | |
| | 障害調査用 SQL の出力 | − | − | ○ | − | − | − | ○ | |
| | オブジェクトの自動クローズ | ○ | − | − | − | − | ○ | − | |
| | リソースへの接続テスト | − | − | ○ | − | − | − | ○ | |
| GC 制御機能 | | − | ○ | − | − | − | − | ○ | 2.9 |
| コンテナ拡張ライブラリ | | − | − | − | ○ | − | − | ○ | 2.10 |
| JavaVM の機能 | | − | − | ○ | − | − | − | ○ | 2.11 |

（凡例） ○：対応する　　−：対応しない

注

「Java EE 標準への準拠」の「標準」と「拡張」の両方に○が付いている機能は，Java EE 標準の機能にアプリケーションサーバ独自の機能が拡張されていることを示します。「拡張」だけに○が付いている機能はアプリケーションサーバ独自の機能であることを示します。

注※

バッチアプリケーションの場合，別名が付けられるのは J2EE リソースだけです。Enterprise Bean の説明は該当しません。

## 1.2.2　CTM による Enterprise Bean のスケジューリング機能

CTM による Enterprise Bean のスケジューリング機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

### 表 1-3　CTM による Enterprise Bean のスケジューリング機能とシステムの目的の対応

| 機能 | システムの目的 | | | | | Java EE 標準への準拠 | | 参照先 |
|---|---|---|---|---|---|---|---|---|
| | 信頼性 | 性能 | 運用・保守 | 拡張性 | そのほか | 標準 | 拡張 | |
| リクエストの流量制御 | ○ | ○ | − | − | − | − | ○ | 3.4 |
| リクエストの優先制御 | ○ | ○ | − | − | − | − | ○ | 3.5 |
| リクエストの同時実行数の動的変更 | ○ | ○ | ○ | − | − | − | ○ | 3.6 |
| リクエストの閉塞制御 | ○ | − | ○ | − | − | − | ○ | 3.7 |
| リクエストの負荷分散 | ○ | ○ | − | ○ | − | − | ○ | 3.8 |
| リクエストのキューの滞留監視 | ○ | − | ○ | − | − | − | ○ | 3.9 |
| CTM のゲートウェイ機能を利用した TPBroker／OTM クライアントとの接続 | − | − | − | ○ | − | − | ○ | 3.10 |

（凡例）　○：対応する　　−：対応しない

注

「Java EE 標準への準拠」の「拡張」だけに○が付いている機能はアプリケーションサーバ独自の機能であることを示します。

## 1.2.3　そのほかの拡張機能

そのほかの拡張機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

### 表 1-4　そのほかの拡張機能とシステムの目的の対応

| 機能 | システムの目的 | | | | | Java EE 標準への準拠 | | 参照先 |
|---|---|---|---|---|---|---|---|---|
| | 信頼性 | 性能 | 運用・保守 | 拡張性 | そのほか | 標準 | 拡張 | |
| バッチアプリケーションのスケジューリング | ○ | ○ | − | − | − | − | ○ | 4 章 |
| セッションフェイルオーバ機能 | ○ | − | ○ | − | − | − | ○ | 5 章，6 章 |
| 明示管理ヒープ機能を使用した FullGC の抑止 | ○ | − | − | − | − | − | ○ | 7 章 |

| 機能 | システムの目的 | | | | | Java EE 標準への準拠 | | 参照先 |
|---|---|---|---|---|---|---|---|---|
| | 信頼性 | 性能 | 運用・保守 | 拡張性 | そのほか | 標準 | 拡張 | |
| アプリケーションのユーザログ出力 | － | － | ○ | － | － | － | ○ | 8章 |

（凡例） ○：対応する　　－：対応しない

注

「Java EE 標準への準拠」の「標準」と「拡張」の両方に○が付いている機能は，Java EE 標準の機能にアプリケーションサーバ独自の機能が拡張されていることを示します。「拡張」だけに○が付いている機能はアプリケーションサーバ独自の機能であることを示します。

## 1.3　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。

### 1.3.1　分類の意味

このマニュアルでは，各機能の説明を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。

### 1.3.2　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「2.1　アプリケーション属性ファイル（cosminexus.xml）の指定内容」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。

## 1.4　アプリケーションサーバ 11-20 での主な機能変更

この節では，アプリケーションサーバ 11-20 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 11-20 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。

### 1.4.1　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 1-5　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 |

# 2 バッチサーバによるアプリケーションの実行

バッチサーバは，バッチアプリケーションを実行するためのサーバです。この章では，バッチサーバで提供する機能，およびバッチアプリケーションの作成方法について説明します。

なお，バッチアプリケーションのスケジューリング機能を使用したバッチアプリケーションの実行については，「4.　バッチアプリケーションのスケジューリング」を参照してください。

## 2.1　この章の構成

Java で開発したバッチジョブを実行するためのアプリケーションをバッチアプリケーションといいます。バッチアプリケーションは，常駐型の JavaVM プロセスであるバッチサーバで実行します。

バッチサーバによるアプリケーションの実行の概要については，「2.2　バッチアプリケーション実行環境の概要」を参照してください。また，バッチアプリケーションでのリソース接続の概要については，「2.6 リソース接続とトランザクション管理の概要」を参照してください。

アプリケーションサーバで提供するバッチサーバの機能と参照先を次の表に示します。

### 表 2-1　アプリケーションサーバで提供するバッチサーバの機能

| 機能名 | 参照先 |
|---|---|
| バッチアプリケーション実行機能 | 2.3 |
| EJB アクセス機能 | 2.4 |
| ネーミング管理機能 | 2.5 |
| リソース接続機能 | 2.7 |
| トランザクション管理 | 2.8 |
| GC 制御機能 | 2.9 |
| コンテナ拡張ライブラリ | 2.10 |
| JavaVM の機能 | 2.11 |
| Java アプリケーションからの移行 | 2.12 |
| JP1/AJS との連携 | 2.13 |

表 2-1 の機能のほかに，バッチサーバでは，バッチアプリケーションのスケジューリング機能を提供しています。以降，この機能をスケジューリング機能といいます。スケジューリング機能については，「4. バッチアプリケーションのスケジューリング」を参照してください。

## 2.2　バッチアプリケーション実行環境の概要

この節では，バッチアプリケーション実行環境の概要について説明します。

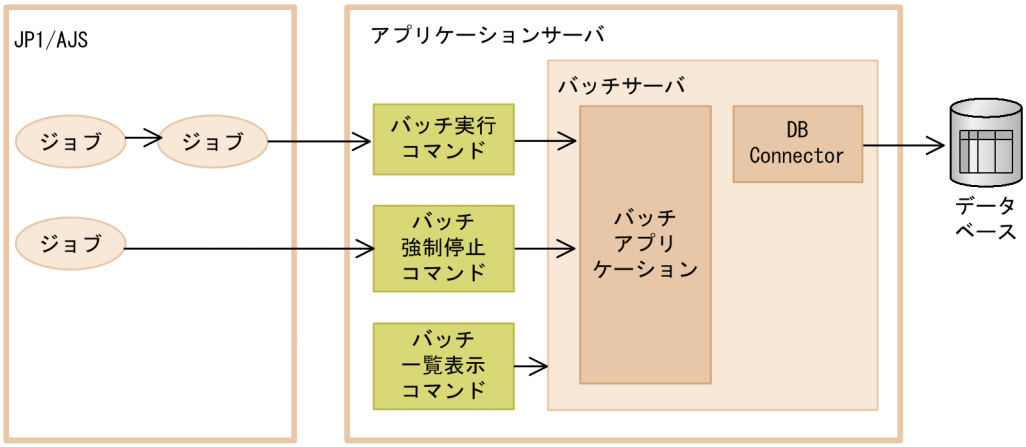
バッチアプリケーションは，バッチ処理を実装した Java アプリケーションです。バッチアプリケーション実行環境は，バッチアプリケーションを実行するための環境です。常駐型の JavaVM プロセスであるバッチサーバで構成されます。アプリケーションサーバでは，コマンドを使用して，バッチサーバ上のバッチアプリケーションを実行します。バッチサーバで同時に実行できるバッチアプリケーションは一つだけです。

バッチサーバではバッチアプリケーションを実行する機能として，バッチサービスを提供しています。バッチ実行コマンド（cjexecjob コマンド）を実行すると，バッチサービスはバッチアプリケーションの情報を基に，バッチアプリケーションの実行を開始します。また，バッチ強制停止コマンド（cjkilljob コマンド）を実行すると，実行中のバッチアプリケーションに対して強制停止を実行します。バッチ一覧表示コマンド（cjlistjob）を実行すると，バッチアプリケーションの情報を出力します。

バッチアプリケーションの実行環境は，JP1/AJS と連携できます。バッチ実行コマンドをあらかじめ JP1/AJS のジョブとして定義しておくことで，JP1/AJS からバッチアプリケーションを実行できます。バッチ強制停止コマンドも，JP1/AJS のジョブとして定義できます。

バッチアプリケーション実行の流れを次の図に示します。

図 2-1　バッチアプリケーション実行の流れ




### 2.2.1　バッチアプリケーションを実行するシステム

バッチアプリケーションを実行するシステムには，バッチサーバが必要です。また，バッチアプリケーションを実行するシステムは，次の製品と連携する運用もできます。

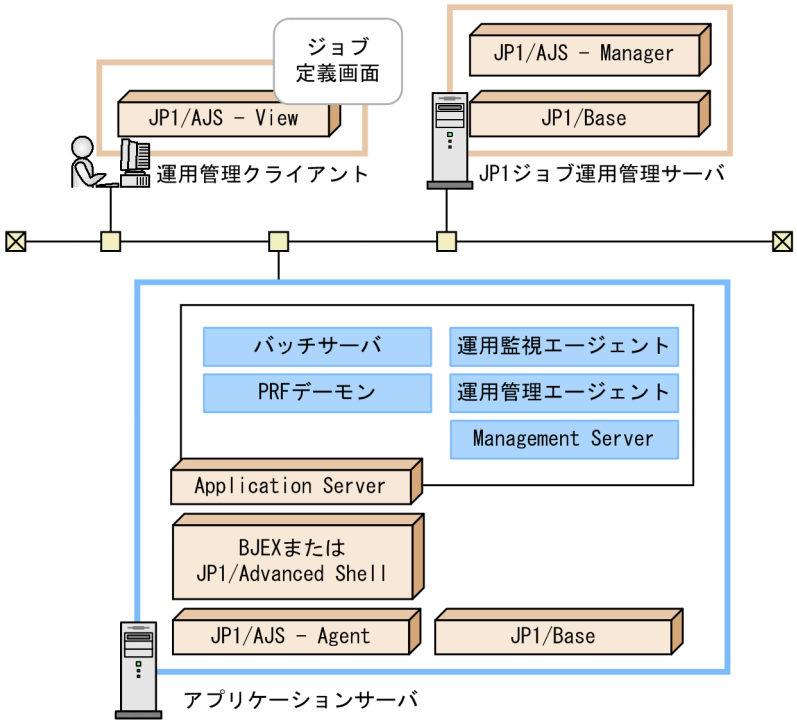
- JP1/AJS
- BJEX または JP1/Advanced Shell

これらの製品と連携すると，バッチサーバの起動・停止や，バッチアプリケーションの開始をジョブとして定義して，バッチアプリケーションを自動実行できます。また，BJEX，または JP1/Advanced Shell と連携すると，バッチアプリケーションの実行コマンドの戻り値を使用したジョブステップの条件付き実行機能を使用したり，ジョブを強制終了したときにバッチアプリケーションを自動的に停止させたりできます。

ここでは，バッチアプリケーションを実行するシステムのシステム構成について説明します。スケジューリング機能を使用したシステムについては，「4.　バッチアプリケーションのスケジューリング」を参照してください。

バッチアプリケーションを実行するシステムの構成例を次の図に示します。

図 2-2　バッチアプリケーションを実行するシステムの構成例




この図の場合，バッチアプリケーションを実行するシステムは，次の製品と連携しています。

- JP1/AJS
- BJEX または JP1/Advanced Shell

これらの製品と連携しない場合は，図中の運用管理クライアント，JP1 ジョブ運用管理サーバ，ならびにバッチサーバの BJEX，JP1/Advanced Shell，JP1/AJS - Agent および JP1/Base は必要ありません。

## 2.2.2　バッチサーバおよびバッチアプリケーションの操作の流れ

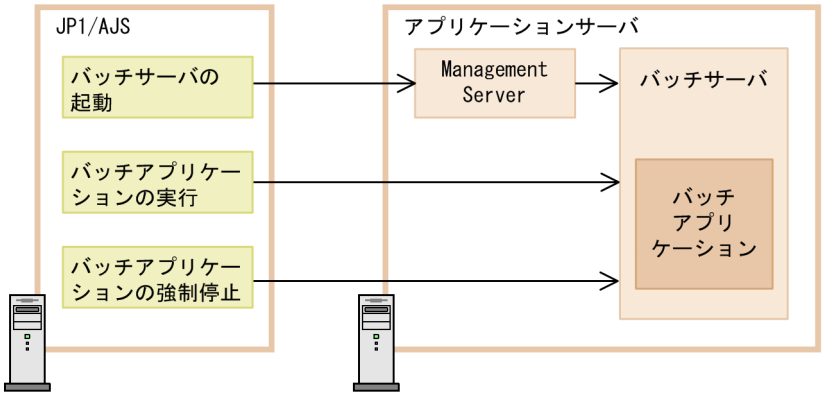
ここでは，システム構成ごとに，バッチサーバおよびバッチアプリケーションの操作の流れについて説明します。

## (1) JP1/AJS と連携するシステム

JP1/AJS と連携するシステムでは，バッチサーバの起動や，バッチアプリケーションの実行および強制停止は，JP1/AJS から操作します。JP1/AJS では，あらかじめバッチサーバやバッチアプリケーションの操作をジョブとして定義します。

バッチサーバおよびバッチアプリケーション操作の流れを次の図に示します。

図 2-3　バッチサーバおよびバッチアプリケーション操作の流れ（JP1/AJS 連携）




バッチサーバは，JP1/AJS からアプリケーションサーバの Management Server を経由して起動します。一方，バッチアプリケーションの実行や強制停止は，JP1/AJS から直接バッチアプリケーションに対して実行します。JP1/AJS ではこれらの操作を UNIX ジョブまたは PC ジョブとしてあらかじめ定義しておきます。

JP1/AJS でのジョブ定義については，「2.13.1　JP1/AJS と連携するための設定」を参照してください。

**参考**

Management Server を配置しない構成もできます。ただし，配置しない構成の場合，バッチアプリケーションの強制停止に失敗したときには，手動でバッチサーバの再起動が必要になります。Management Server を使用してバッチサーバを監視すると，トラブル発生時にバッチサーバを自動再起動できるため，Management Server を使用する運用をお勧めします。

## (2) JP1/AJS，BJEX，および JP1/Advanced Shell と連携するシステム

次の製品と連携するシステムでは，バッチサーバの起動や，バッチアプリケーションの実行および強制停止は，JP1/AJS，BJEX，または JP1/Advanced Shell から操作します。

- JP1/AJS
- BJEX または JP1/Advanced Shell

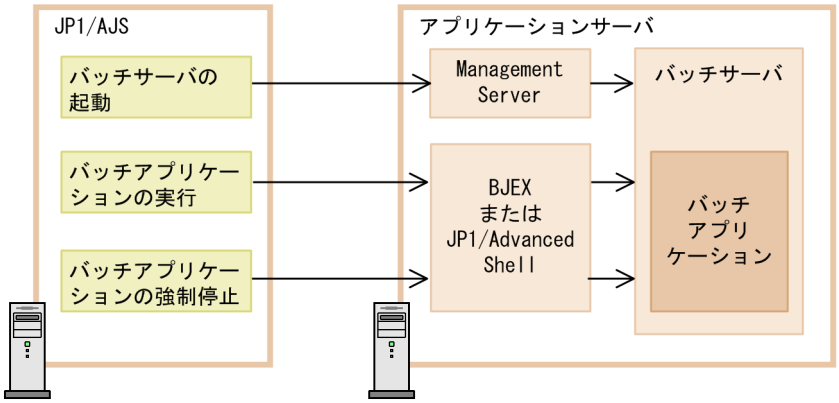
JP1/AJS，BJEX，および JP1/Advanced Shell では，あらかじめバッチサーバやバッチアプリケーションの操作をジョブとして定義します。

**注意事項**

BJEX と連携する場合は，JP1/AJS とも連携する必要があります。JP1/Advanced Shell と連携する場合，JP1/AJS と連携する必要はありません。

バッチサーバおよびバッチアプリケーション操作の流れを次の図に示します。

図 2-4　バッチサーバおよびバッチアプリケーション操作の流れ（JP1/AJS，BJEX，および JP1/Advanced Shell 連携）




バッチサーバは，JP1/AJS からアプリケーションサーバの Management Server を経由して起動します。バッチアプリケーションの実行，およびバッチアプリケーションの強制停止は，JP1/AJS から BJEX または JP1/Advanced Shell を経由してバッチアプリケーションに対して実行します。このため，JP1/AJS，BJEX，および JP1/Advanced Shell で，次に示す操作をあらかじめジョブに定義します。

- バッチサーバの起動

  JP1/AJS の UNIX ジョブまたは PC ジョブとして定義します。

- バッチアプリケーションの実行

  JP1/AJS の UNIX ジョブまたは PC ジョブとして，BJEX のジョブ定義 XML ファイルまたは JP1/Advanced Shell のジョブ定義スクリプトファイルを指定します。また，BJEX のジョブ定義 XML ファイルまたは JP1/Advanced Shell のジョブ定義スクリプトファイルにバッチアプリケーションの実行を定義します。

- バッチアプリケーションの強制停止

  実行中の UNIX ジョブまたは PC ジョブを JP1/AJS から強制終了した場合，その命令を受けた BJEX または JP1/Advanced Shell は，バッチアプリケーションを自動的に停止させます。

JP1/AJS，BJEX，および JP1/Advanced Shell でのジョブ定義については，「2.13.2　JP1/AJS，BJEX，および JP1/Advanced Shell と連携するための設定」を参照してください。

**参考**

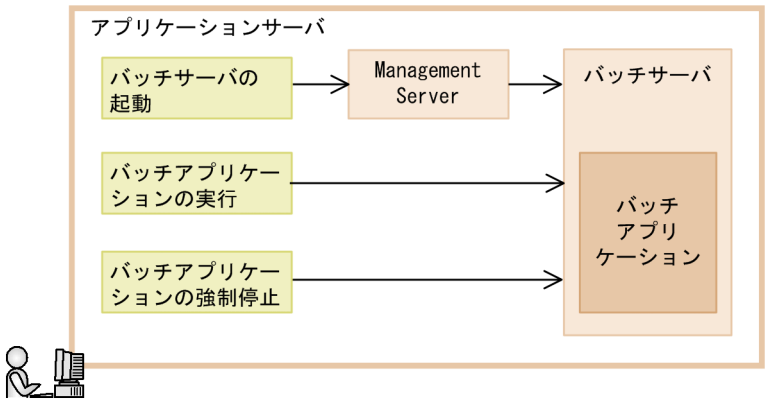
- Management Server を配置しない構成もできます。ただし，配置しない構成の場合，バッチアプリケーションの強制停止に失敗したときには，手動でバッチサーバの再起動が必要になります。Management Server を使用してバッチサーバを監視すると，トラブル発生時にバッチサーバを自動再起動できるため，Management Server を使用する運用をお勧めします。
- バッチサーバでは BJEX のジョブログ出力機能を使用できます。ただし，ジョブログ出力機能で出力されるログには，cjexecjob コマンドの CPU 使用時間およびメモリ使用量が出力されます。Java バッチアプリケーション自体のジョブステップの CPU 使用時間およびメモリ使用量は出力できません。

## (3) JP1/AJS，BJEX，および JP1/Advanced Shell と連携しないシステム

JP1/AJS，BJEX，および JP1/Advanced Shell と連携しないシステムでは，バッチサーバの起動や，バッチアプリケーションの実行および強制停止は，コマンドで直接実行します。

バッチサーバおよびバッチアプリケーション操作の流れを次の図に示します。

図 2-5 バッチサーバおよびバッチアプリケーション操作の流れ




バッチサーバは，Smart Composer 機能で提供するコマンドを使用して，アプリケーションサーバの Management Server を経由して起動します。一方，バッチアプリケーションの実行や強制停止は，バッチアプリケーション実行機能が提供するコマンド（バッチ実行コマンドおよびバッチ強制停止コマンド）を使用して直接バッチアプリケーションに対して実行します。

**参考**

Management Server を配置しない構成もできます。ただし，配置しない構成の場合，バッチアプリケーションの強制停止に失敗したときには，手動でバッチサーバの再起動が必要になります。Management Server を使用してバッチサーバを監視すると，トラブル発生時にバッチサーバを自動再起動できるため，Management Server を使用する運用をお勧めします。

## 2.2.3　バッチアプリケーションの実行環境の構築と運用

ここでは，バッチアプリケーションの実行環境の構築方法と，運用方法について説明します。また，バッチアプリケーションの実行環境と連携できるプログラムについても説明します。

### (1)　バッチアプリケーションの実行環境の構築

バッチアプリケーションの実行環境は，Smart Composer 機能，サーバ管理コマンドを使用して構築します。バッチアプリケーションの実行環境の構築手順を次に示します。

1. Smart Composer 機能を使用してシステムを構築します。
   簡易構築定義ファイルでシステム構成を定義し，Smart Composer 機能で提供するコマンドを使用して，システムを一括構築します。
2. サーバ管理コマンドを使用して，リソースアダプタを設定します。
   バッチアプリケーションからデータベースに接続する場合だけ実施します。

Smart Composer 機能，サーバ管理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 システム構築・運用ガイド」の「4.6　バッチアプリケーションを実行するシステムの構築」を参照してください。

**注意事項**

バッチサーバを複数構築する場合，サーバで使用する TCP/IP のポート番号を重複しないよう変更する必要があります。また，バッチサーバは J2EE サーバで使用している TCP/IP のポート番号を使用します。複数のバッチサーバを同時に起動する場合，およびバッチサーバと J2EE サーバを同時に起動する場合は，使用するポート番号が重複しないように設定してください。ポート番号については，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「3.15　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」を参照してください。

**参考**

バッチアプリケーションの実行環境は運用管理ポータルを使用して構築することもできます。運用管理ポータルを使用したバッチアプリケーションの実行環境の構築については，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」の「5.　バッチアプリケーションを実行するシステムの構築と削除」を参照してください。

### (2)　バッチアプリケーションの実行環境の運用

バッチアプリケーションの実行環境は次の順序で運用します。

1. システムの起動
   Smart Composer 機能で提供するコマンドを使用して，バッチサーバを含むシステム全体を起動します。バッチアプリケーションからリソースに接続する場合は，DB Connector も開始します。

2. バッチアプリケーションの実行

   cjexecjob コマンドを使用してバッチアプリケーションを開始します。

3. バッチサーバの停止

   Smart Composer 機能で提供するコマンドを使用して，バッチサーバを含むシステム全体を停止します。

> **参考**
>
> 運用管理ポータルを使用したバッチアプリケーションの実行環境の起動および停止については，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」の「6.1　システムの起動と停止」を参照してください。

JP1/AJS と連携する場合は，JP1/AJS からバッチサーバおよびバッチアプリケーションを開始できます。また，JP1/AJS，BJEX，および JP1/Advanced Shell と連携する場合は JP1/AJS からバッチサーバを，BJEX または JP1/Advanced Shell からバッチアプリケーションを開始できます。

システムの起動および停止の詳細については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「2.6　システムの起動と停止の設定」を参照してください。バッチアプリケーションの開始方法については，「2.3.2　バッチアプリケーションの実行」を参照してください。

バッチアプリケーションを実行するシステムでは，次の運用機能が使用できます。

### (a)　システムの日常運用を支援する機能

システムの起動・停止のほかに，バッチサーバの稼働状況や，バッチサーバのリソースの使用状況を監視できます。システムの日常運用を支援する機能の概要について説明します。

- **稼働情報の監視（稼働情報収集機能）**

  バッチサーバの稼働状態を定期的に監視し，サーバ性能やリソースの情報などの稼働情報を取得します。

- **運用管理コマンドによる稼働情報の出力**

  運用管理コマンドを使用して，運用管理ドメイン内の論理サーバを監視し，稼働情報を取得します。

- **リソースの枯渇監視**

  バッチサーバを対象に，メモリやスレッドなどのリソースを監視します。監視対象のリソースについての情報は，一定間隔でファイルに出力されます。また，監視対象のリソースの状態が設定したしきい値を超えた場合には，アラートが発生します。アラートが発生すると，メッセージを出力し，Management Server に対してイベントを通知します。

- **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**

  バッチサーバが稼働中に出力するすべてのメッセージを契機にして Management イベントを発行できます。Management Server 側では，Management イベントが通知されたときの動作を定義しておくことで，Management イベントが発生すると自動的にアクションを実行できるようになります。

- **CTM の稼働統計情報の収集**

  バッチアプリケーションのスケジューリング機能を使用する場合に，CTM から出力された稼働統計情報を収集できます。この情報を基に，CTM の処理性能を分析できます。

システムの日常運用を支援する機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.1　システムの日常運用を支援する機能」を参照してください。

### (b)　システムの保守を支援する機能

バッチサーバなど，運用管理エージェントによって起動されるプロセスの情報をコンソールログとして出力できます。コンソールログ出力の概要について説明します。

- コンソールログ出力

  運用管理エージェントが起動したプロセスの標準出力や標準エラー出力などのコンソール出力情報をコンソールログに出力できます。コンソールログ出力については，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「11.　コンソールログの出力」を参照してください。

また，バッチアプリケーションのログをユーザログとして出力できます。ユーザログ出力は拡張機能の一つです。ユーザログ出力の概要について説明します。

- ユーザログ出力

  バッチアプリケーションで例外が発生した場合に，メッセージおよびログをトレース共通ライブラリ形式で出力できます。ユーザログについては，「8.　アプリケーションのユーザログ出力」を参照してください。

### (c)　システムの監査を支援する機能

システムの構築者や運用者がアプリケーションサーバのプログラムに対して実行した操作や履歴を出力できます。また，バッチアプリケーションがデータベースにアクセスした際に使用したアカウントを記録できます。システムの監査を支援する機能の概要について説明します。

- 監査ログの出力

  監査ログに，システムの構築者や運用者がアプリケーションサーバのプログラムに対して実行した操作，およびその操作に伴うプログラムの動作の履歴を出力することで，システムの監査に利用できます。

- データベース監査証跡機能との連携

  データベースが提供するデータベース監査証跡機能と連携することで，バッチアプリケーションがデータベースにアクセスした際に使用したアカウントを記録できます。

システムの監査を支援する機能については，次の個所を参照してください。

- 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「6.　監査ログ出力機能」
- 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「7.　データベース監査証跡連携機能」

### (d)　システムの保守のための機能

バッチサーバの障害を検知したときに，トラブルシューティングの資料を取得できます。システムの保守のための機能の概要について説明します。

- トラブルシューティング

  障害検知時コマンドを使用すると，Management Server が論理サーバの障害を検知した場合に，トラブルシューティングの資料を取得できます。また，アプリケーションサーバの構成ソフトウェアの snapshot ログを出力，収集できます。

  例えば，システムにトラブルが発生した場合には，トラブルシューティング情報として snapshot ログを自動的に収集できます。

- 性能解析トレースを使用したシステムの性能解析

  性能解析トレースは，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を収集する機能です。この情報を基に，システム性能およびボトルネックを分析できます。

システムの保守のための機能については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」を参照してください。

## (3) ほかのプログラムとの連携

バッチアプリケーションを実行するシステムでは，次に示すプログラムと連携できます。

- JP1 との連携
- クラスタソフトウェアとの連携

JP1 との連携については，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「12. JP1 と連携したシステムの運用」を参照してください。クラスタソフトウェアとの連携については，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「15. クラスタソフトウェアとの連携」を参照してください。

### (a) JP1 連携による運用管理機能の概要

JP1 連携による運用管理機能の概要について説明します。

- システムの集中監視（JP1/IM との連携）

  業務システム全体のリソースの状態を集中監視することで，稼働性能を把握，調査したり，トラブルの発生を検知して，原因を究明して対策したりできます。この機能は JP1/IM と連携して実現できます。

- ジョブによるシステムの自動運転（JP1/AJS との連携）

  サーバやアプリケーションの開始や停止のスケジュールをあらかじめ定義して自動化することで，効率的なリソースの配分や業務の効率化，省力化を図れます。JP1/AJS と連携することで，カスタムジョブによるシステムの自動運転が実現できます。

- 監査ログの収集および一元管理（JP1/Audit Management - Manager との連携）

  システムの監査で使用する監査ログを，自動で収集して，一括管理できます。この機能は JP1/Audit Management - Manager と連携して実現できます。

### (b) クラスタソフトウェアとの連携による系切り替え機能の概要

クラスタソフトウェアとの連携による系切り替え機能の概要について説明します。連携できるクラスタソフトウェアは，Windows Server Failover Cluster※（Windows の場合），および HA モニタ（AIX・Linux の場合）です。

注※

使用できる OS は，Windows Server 2019 Standard/Datacenter および Windows Server 2022 Standard/Datacenter となります。

- **1:1 系切り替えシステム**

  実行系と待機系が 1 対 1 になるシステム構成です。バッチアプリケーション実行環境の場合，アプリケーションサーバの 1:1 系切り替えシステムでの運用をサポートしています。実行系サーバでの障害発生時またはシステムの保守時に，あらかじめ待機させておいたサーバに自動的に切り替えて，業務処理を続行するための機能です。これによって，システムの不稼働時間を短縮し，クライアントの業務処理への影響を減らせます。

  なお，バッチアプリケーション実行環境の場合，運用管理サーバを配置しないため，運用管理サーバの 1:1 系切り替えシステムは使用できません。

- **相互系切り替えシステム**

  1:1 系切り替えシステムの構成で，2 台のサーバがそれぞれ現用系として動作しながら，互いの予備系になるシステムです。アプリケーションサーバの相互系切り替えシステムでの運用をサポートしています。

- **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**

  ホスト単位管理モデルの実行系 N 台と予備系 1 台を配置したシステム構成です。ホスト単位管理モデルのアプリケーションサーバの系切り替えシステムの運用をサポートしています。

## 2.2.4 マルチバイト文字の使用について

次に示す個所でマルチバイト文字を使用する場合は，これらの個所すべてでマルチバイト文字のエンコードを同じにしてください。

- バッチアプリケーション用オプション定義ファイル（usrconf.cfg）内にマルチバイト文字を使用する場合
- バッチサーバ用オプション定義ファイル（usrconf.cfg）内にマルチバイト文字を使用する場合
- cjexecjob コマンドの引数にマルチバイト文字を指定する場合
- バッチアプリケーションのソースコードで，java.lang.System.out または java.lang.System.err にマルチバイト文字を出力する場合

また，バッチサーバと cjexecjob コマンドを実行するコンソールの環境変数 LANG には，対応する文字エンコードを表示できるようにしてください。

## 2.3　バッチアプリケーション実行機能

バッチアプリケーション実行機能は，バッチサーバで提供する機能の一つです。バッチアプリケーション実行機能は，バッチアプリケーションを実行したり，バッチアプリケーションが出力したデータをログ出力機能に出力したりします。

この節では，バッチアプリケーション実行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 2-2　この節の構成（バッチアプリケーション実行機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | バッチアプリケーション実行機能の概要 | 2.3.1 |
| | バッチアプリケーションの実行 | 2.3.2 |
| | バッチアプリケーションの強制停止 | 2.3.3 |
| | バッチアプリケーション情報の一覧表示 | 2.3.4 |
| | バッチアプリケーションのログ出力 | 2.3.5 |
| | バッチアプリケーションで使用するコマンドの実行について | 2.3.6 |
| 実装 | バッチアプリケーションの実装（バッチアプリケーションの作成規則） | 2.3.7 |
| | バッチアプリケーションの実装（リソースに接続する場合） | 2.3.8 |
| | バッチアプリケーションの実装（EJB にアクセスする場合） | 2.3.9 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.3.10 |
| 注意事項 | バッチアプリケーション作成時の注意 | 2.3.11 |

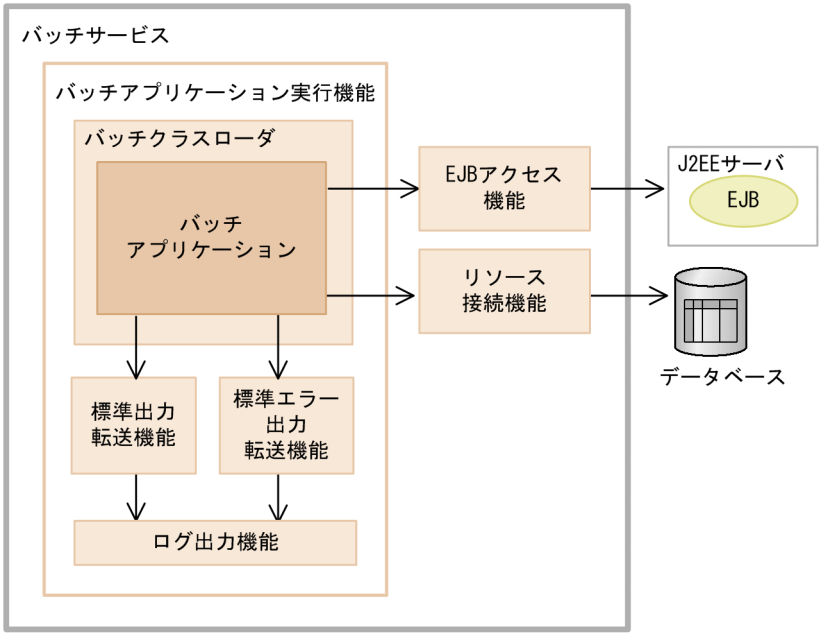
注　「運用」について，この機能固有の説明はありません。

### 2.3.1　バッチアプリケーション実行機能の概要

バッチアプリケーション実行機能とは，バッチアプリケーションを実行するための機能です。バッチアプリケーションは，バッチアプリケーション実行機能で提供されているバッチクラスローダ上で実行されます。また，実行中のバッチアプリケーションが出力した内容は，ログ出力機能に出力されます。

バッチアプリケーション実行機能について次の図に示します。

図 2-6　バッチアプリケーション実行機能の概要




また，バッチアプリケーション実行機能は，EJB アクセス機能やリソース接続機能と連携できます。

- EJB アクセス機能と連携すると，バッチアプリケーションからほかの J2EE サーバの EJB にアクセスできます。
- リソース接続機能と連携すると，バッチアプリケーションからデータベースに接続できます。

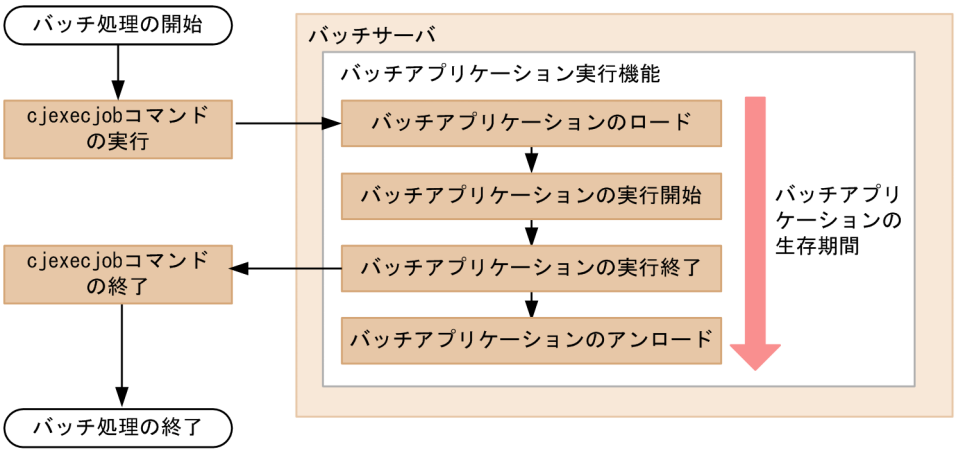
EJB アクセス機能については「2.4.1　EJB アクセスで使用できる機能」を，リソース接続機能については「2.7　リソース接続機能」を参照してください。

次に，バッチアプリケーションのライフサイクルとバッチアプリケーションを実行するクラスローダについて説明します。

## (1)　バッチアプリケーションのライフサイクル

バッチアプリケーションは，cjexecjob コマンドを使用して開始します。次の図を使用して，バッチアプリケーションのライフサイクルについて説明します。

図 2-7　バッチアプリケーションのライフサイクル




1. cjexecjob コマンドを実行すると，バッチアプリケーションはバッチクラスローダによってロードされます。
2. バッチアプリケーションがバッチサーバ上で実行されます。
3. バッチアプリケーションの処理が終了します。

   バッチアプリケーションの処理終了後に，バッチアプリケーションをロードしたバッチクラスローダが GC されます。
4. バッチアプリケーションのクラスがアンロードされます。

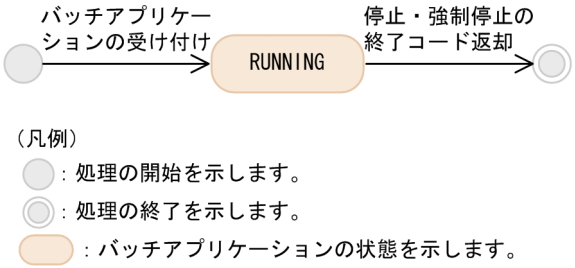
**注意事項**

バッチアプリケーションは cjexecjob コマンドが実行されるたびにバッチクラスローダにロードされ，処理が完了するとクラスがアンロードされます。常駐形式のバッチアプリケーションをバッチサーバ上で動作させることは推奨しません。

## (2)　バッチアプリケーションの状態遷移

バッチアプリケーションの状態遷移を次の図に示します。

図 2-8　バッチアプリケーションの状態遷移（スケジューリング機能を使用しない場合）



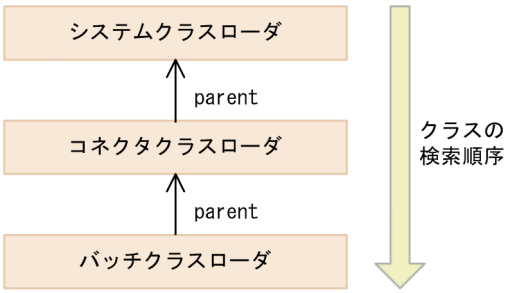

「RUNNING」は，バッチアプリケーションがバッチサーバ上にあって実行中の状態です。

バッチアプリケーションの状態は，バッチアプリケーション情報から確認できます。バッチアプリケーション情報の表示方法については，「2.3.4　バッチアプリケーション情報の一覧表示」を参照してください。

## (3)　バッチアプリケーションを実行するクラスローダ

バッチアプリケーション実行時には，バッチサーバ上でバッチアプリケーション用のクラスローダが生成されます。バッチアプリケーションはクラスローダ上で実行されます。バッチアプリケーション用のクラスローダの構成を次の図に示します。

図 2-9　バッチアプリケーションを実行するクラスローダの構成




図のそれぞれのクラスローダについて説明します。

- システムクラスローダ

  アプリケーションサーバの構成ソフトウェアが提供するクラスやコンテナ拡張ライブラリのクラスをロードします。

  - 生成タイミング：J2EE サーバ起動時
  - 破棄タイミング：J2EE サーバ停止時
- コネクタクラスローダ

  リソースアダプタに含まれるクラスをロードします。バッチサーバ内に一つだけ存在します。

  - 生成タイミング：J2EE サーバ起動時
  - 破棄タイミング：J2EE サーバ停止時
- バッチクラスローダ

  バッチアプリケーションに含まれるクラスをロードします。バッチアプリケーションを実行するスレッドのコンテキストクラスローダは，バッチクラスローダです。

  - 生成タイミング：バッチアプリケーション実行時
  - 破棄タイミング：バッチアプリケーション終了時

なお，バッチクラスローダ生成時には，バッチクラスローダが生成されたことを示すメッセージが出力されます（メッセージ KDJE55013-I）。また，バッチクラスローダのファイナライズ処理が実行されたことを示すメッセージも出力されます（メッセージ KDJE55014-I）。

クラスローダの破棄についての注意事項は，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「付録 B.1　デフォルトのクラスローダ構成」を参照してください。なお，クラスローダの破棄のタイミング，クラスローダの破棄時に出力されるメッセージについては適宜読み替えてください。

## 2.3.2　バッチアプリケーションの実行

バッチアプリケーションは，cjexecjob コマンドで開始します。バッチアプリケーションの main メソッドの実行が終わると，バッチサーバは FullGC を実行します。ここでは，バッチアプリケーションの開始方法と，バッチアプリケーションの開始および終了時の処理について説明します。

なお，実行中のバッチアプリケーションを停止する場合には，強制停止をします。バッチアプリケーションの強制停止の方法については，「2.3.3　バッチアプリケーションの強制停止」を参照してください。

### (1)　バッチアプリケーションの開始方法

ここでは，バッチアプリケーションの開始方法について説明します。

バッチアプリケーションを開始するには cjexecjob コマンドを使用します。cjexecjob コマンドを実行するには，次の四つの方法があります。

1. cjexecjob コマンドを直接実行する方法
   JP1/AJS，BJEX，および JP1/Advanced Shell を使用しない場合はこの方法で開始します。
2. cjexecjob コマンドを JP1/AJS のジョブとして定義しておき，JP1/AJS から実行する方法
   JP1/AJS だけを使用する場合はこの方法で開始します。
3. cjexecjob コマンドを BJEX のジョブステップとして定義しておき，JP1/AJS から BJEX のジョブを実行する方法
   JP1/AJS および BJEX を使用する場合はこの方法で開始します。
4. JP1/Advanced Shell のジョブ定義スクリプトファイルから，JP1/Advanced Shell が提供する adshjava コマンドを使用して実行する方法
   JP1/Advanced Shell を使用する場合はこの方法で開始します。なお，この方法では，JP1/Advanced Shell を直接実行することも，JP1/AJS 経由で JP1/Advanced Shell を実行することもできます。

2.，3.，および 4.の方法で，バッチアプリケーションを開始するときの，JP1/AJS，BJEX，および JP1/Advanced Shell のジョブの定義については，「2.13　JP1/AJS との連携」を参照してください。

なお，JP1/AJS，BJEX，および JP1/Advanced Shell からバッチアプリケーションを実行する際には，あらかじめバッチサーバを起動しておいてください。

### (2)　バッチアプリケーションの開始処理

cjexecjob コマンドにバッチアプリケーションのクラス名とクラスパスを指定すると，cjexecjob コマンドに指定したバッチアプリケーションが開始します。バッチアプリケーション開始時の処理を次に示します。

1. バッチアプリケーションの開始処理を開始することを示すメッセージ（KDJE55000-I）を出力します。
2. バッチアプリケーションの main メソッドを開始することを示すメッセージ（KDJE55001-I）を出力します。

3. public static void main(String[])メソッドまたは public static int main(String[])メソッドを実行します。

バッチアプリケーションの開始時には，cjexecjob コマンドに指定した実行クラスの public static void main(String[])メソッドまたは public static int main(String[])メソッドが呼び出されます。メソッドの引数には，cjexecjob コマンドのクラス名のあとに指定した引数を設定します。

#### バッチアプリケーションの開始に失敗する場合

バッチアプリケーションに main メソッドが定義されていない場合などには，バッチアプリケーションの開始は失敗します。なお，バッチアプリケーションの開始に失敗すると，バッチサーバと cjexecjob コマンドは次のように動作します。

- バッチサーバの動作

  メッセージを出力し，バッチアプリケーション開始に失敗した情報とメッセージ文字列を cjexecjob コマンドに返します。

- cjexecjob コマンドの動作

  バッチサーバから受け取ったメッセージ文字列を出力し，異常終了します。なお，コマンドの戻り値は 1 です。

次の表に，バッチアプリケーションの開始に失敗する条件，およびバッチサーバが出力するメッセージを示します。

表 2-3　バッチアプリケーションの開始に失敗する条件

| バッチアプリケーションの開始に失敗する条件 | バッチサーバが出力するメッセージ |
|---|---|
| usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）の読み込みに失敗した。 | KDJE55035-E |
| cjexecjob コマンドに指定したクラスが存在しない。 | KDJE55006-E |
| public static void main(String[])メソッドまたは public static int main(String[])メソッドが定義されていない。 | |
| public static void main(String[])メソッドまたは public static int main(String[])メソッドのどちらともシグニチャが異なる。 | |
| cjexecjob コマンドに指定したクラスのロード時に，java.lang.NoClassDefFoundError が発生した。 | KDJE55007-E |
| public static void main(String[])メソッドまたは public static int main(String[])メソッド呼び出し時に必要なクラスが見つからない。 | |
| static{}ブロック内でエラーが発生した。 | |
| 上記以外の問題で main メソッドが実行できない。 | KDJE55008-E |

## (3)　バッチアプリケーションの終了処理

バッチアプリケーションは，main メソッドの実行が終わると処理が終了します。バッチアプリケーション終了時に実行される処理を次に示します。

1. バッチアプリケーション終了処理を開始することを示すメッセージ（KDJE55002-I）を出力します。
2. バッチアプリケーションの終了処理が完了したことを示すメッセージ（KDJE55003-I）を出力します。
3. FullGC を実行します。
4. cjexecjob コマンドにバッチアプリケーションの終了コードを送信します。

次の表に，バッチアプリケーションの終了条件と，そのときのバッチサーバや cjexecjob コマンドの動作を示します。

### 表 2-4　バッチアプリケーションの終了条件

| バッチアプリケーションの終了条件 | バッチサーバの動作 | cjexecjob コマンドの動作 |
|---|---|---|
| main メソッドを最後まで実行した。 | KDJE55002-I 出力して，バッチアプリケーションの実行を終了する。終了後に KDJE55003-I を出力する。 | 正常終了する。<br>戻り値：0 |
| public static void main(String[])メソッドで return 文を実行した。 | | |
| public static int main(String[])メソッドで return <終了コード>を実行した。 | | 正常終了する。<br>戻り値：return に指定した終了コード |
| main メソッドの外に，java.lang.Throwable または java.lang.Throwable を継承したクラスがスローされた。 | KDJE55009-E を出力する。例外のスタックトレースを例外ログに出力する。バッチアプリケーションの実行を終了する。 | 例外のスタックトレースを標準エラー出力に出力する。バッチアプリケーションの実行を異常終了する。<br>戻り値：1 |
| バッチサーバが終了した（バッチサーバの強制終了または予期しない JavaVM のダウン）。 | なし。 | KDJE55021-E を出力して，バッチアプリケーションの実行を異常終了する。<br>戻り値：1 |

なお，バッチアプリケーション実行中に［Ctrl］＋［C］やシグナルなどによって cjexecjob コマンドを終了しても，バッチアプリケーションの実行は終了しません。バッチアプリケーションの実行を強制停止したい場合は，cjkilljob コマンドを実行してください。ただし，cjkilljob コマンドで強制終了した場合，cjexecjob コマンドの終了コードは不定となります。バッチ強制停止コマンドについては，「2.3.3　バッチアプリケーションの強制停止」を参照してください。

## (4)　バッチアプリケーション実行時の注意事項

バッチアプリケーションから EJB または DB Connector を呼び出して使用する場合，バッチアプリケーションの開始時には，使用する EJB および DB Connector があるかどうかの確認は実施しません。バッチアプリケーションから参照している EJB または DB Connector がない場合は，バッチアプリケーション実行中に実行時エラーになります。バッチアプリケーションを開始する前に，参照先の EJB があるかどうかを確認してください。また，DB Connector を使用してバッチアプリケーションからデータベースに接続する場合は，バッチサーバで DB Connector を開始しておいてください。

## 2.3.3　バッチアプリケーションの強制停止

実行中のバッチアプリケーションを必要に応じて停止させることができます。これを，バッチアプリケーションの強制停止といいます。ここでは，バッチアプリケーションの強制停止について説明します。

### (1)　バッチアプリケーションの強制停止方法

バッチアプリケーションを強制停止するには cjkilljob コマンドを使用します。cjkilljob コマンドを実行するには，次の三つの方法があります。

1. cjkilljob コマンドを直接実行する方法
   JP1/AJS を使用しない場合はこの方法で開始します。
2. cjkilljob コマンドを JP1/AJS のリカバリージョブとして定義しておき，ジョブやジョブネットの強制停止の延長で実行する方法
   BJEX または JP1/Advanced Shell の使用の有無に関係なく，JP1/AJS を使用する場合はこの方法で開始します。
   2.の方法で，JP1/AJS のリカバリージョブとしてバッチアプリケーションを強制停止するときの JP1/AJS のジョブの定義については，「2.13　JP1/AJS との連携」を参照してください。
3. BJEX または JP1/Advanced Shell の強制停止の延長で，バッチアプリケーションを強制停止する方法
   BJEX または JP1/Advanced Shell を使用してバッチアプリケーションを実行している場合，BJEX または JP1/Advanced Shell を強制終了することで，これらの製品を経由して実行されたバッチアプリケーションも自動的に停止されます。この方法では，JP1/AJS のリカバリージョブを定義する必要はありません。

なお，バッチアプリケーションの強制停止に失敗した場合，バッチサーバの強制停止が実行されます。このため，複数のアプリケーションを続けて実行する場合は，バッチサーバの再起動が必要になります。バッチアプリケーションの強制停止失敗に備えて，あらかじめバッチサーバを自動再起動するよう設定しておくことをお勧めします。バッチサーバの自動再起動は，Management Server の運用監視を使用して実現できます。詳細は，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「2.4　障害発生時の自動再起動」を参照してください。

### (2)　バッチアプリケーションの強制停止処理

実行中のバッチアプリケーションを強制停止するには，cjkilljob コマンドを使用します。cjkilljob コマンドを使用すると，バッチアプリケーションを実行しているスレッドに対してメソッドキャンセルを実行してバッチアプリケーションを強制停止します。

メソッドキャンセルとは，実行中のメソッドをキャンセルする機能です。ただし，メソッドを実行している領域によって，キャンセルできるメソッドとできないメソッドがあります。メソッドをキャンセルできる領域のことを非保護区，メソッドをキャンセルできない領域を保護区といいます。実行中のメソッドが非保護区の場合に，メソッドキャンセルが実行されます。バッチアプリケーションの強制停止で実行されるメソッドキャンセルは，J2EE アプリケーション実行時間の監視機能で実行されるメソッドキャンセルと

同じです。メソッドキャンセ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 運用／監視／連携編」の「5.3.4　メソッドキャンセルとは」を参照してください。

バッチアプリケーション強制停止時に実行される処理を次に示します。

1. バッチアプリケーションの強制停止処理を開始することを示すメッセージ（KDJE55004-I）を出力します。
2. public static void main(String[])メソッドまたは public static int main(String[])メソッドのメソッドキャンセルを実行します。

   なお，メソッドキャンセルに失敗した場合，KDJE55017-E を出力して強制停止が失敗し，バッチサーバが強制停止します。強制停止が失敗した場合は，バッチサーバを再起動してください。
3. バッチアプリケーションの強制停止処理が完了したことを示すメッセージ（KDJE55005-I）を出力します。
4. FullGC を実行します。
5. cjexecjob コマンドにバッチアプリケーションの終了コードを送信します。

次の表に，バッチアプリケーションの強制停止条件を示します。

表 2-5　バッチアプリケーションの強制停止の条件

| バッチアプリケーションの強制停止の条件 | バッチサーバの動作 | cjexecjob コマンドの動作 |
|---|---|---|
| バッチアプリケーション実行中にバッチ強制停止コマンドを実行した。 | KDJE55004-I 出力して，バッチアプリケーションの実行を強制停止を開始する。強制停止の完了時には KDJE55005-I を出力する。強制停止に失敗したときは KDJE55017-E を出力する。 | バッチ強制終了時の正常パス<br>戻り値：1 |

なお，JP1/Advanced Shell の adshjava コマンドによってバッチアプリケーションを実行した場合は，JP1/Advancel Shell のジョブを強制終了することで，JP1/Advanced Shell が自動的に cjkilljob コマンドを実行してバッチアプリケーションを強制停止します。

## (3)　バッチアプリケーションの強制停止実行時の注意事項

バッチアプリケーションの強制停止に失敗すると，バッチサーバが強制停止されます。複数のバッチアプリケーションを連続して実行する場合，バッチサーバの強制停止後に実行するバッチアプリケーションを開始する前に，バッチサーバを再起動する必要があります。このため，Management Server を利用して，バッチサーバが自動的に再起動するように設定してください。詳細は，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「5.3.4　メソッドキャンセルとは」を参照してください。

## 2.3.4　バッチアプリケーション情報の一覧表示

実行中のバッチアプリケーションの状態や，バッチ実行コマンドの開始時刻などをバッチアプリケーション情報として一覧表示できます。ここでは，バッチアプリケーション情報の一覧表示について説明します。

### (1)　バッチアプリケーション情報の一覧表示の方法

バッチアプリケーション情報の一覧を表示するには，JP1/AJS を使用しているかどうかに関係なく，cjlistjob コマンドを直接実行します。

バッチアプリケーション情報は，バッチサーバ単位に取得できます。cjlistjob コマンドの引数には，バッチアプリケーション情報を取得したいバッチサーバ名を指定します。

### (2)　バッチアプリケーション情報の一覧表示処理

cjlistjob コマンドを実行すると，引数に指定したバッチサーバで実行中のバッチアプリケーションの情報が取得できます。バッチアプリケーション情報は，標準出力に出力されます。

取得できるバッチアプリケーション情報を次の表に示します。

表 2-6　取得できるバッチアプリケーション情報

| 取得できるバッチアプリケーション情報の項目 | 内容 |
|---|---|
| バッチアプリケーションの状態 | 「running」が出力されます。running は，バッチアプリケーションの状態が RUNNING であることを示します。詳細は，「2.3.1(2)　バッチアプリケーションの状態遷移」を参照してください。 |
| バッチアプリケーション名 | cjexecjob コマンドに指定したバッチアプリケーションのクラス名が出力されます。 |
| 性能解析トレースのルートアプリケーション情報 | 性能解析トレースのルートアプリケーションの通信番号が出力されます。<br>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，バッチアプリケーションの状態を確認できます。 |
| バッチ実行コマンドの実行時刻 | cjexecjob コマンドを実行した時刻が次の形式で出力されます。なお，△は，半角スペースを表します。<br>yyyy/mm/dd△hh:mm:ss.ssssss<br>yyyy：西暦年，mm：月，dd：日，hh：時，mm：分，ss：秒，ssssss：マイクロ秒 |

なお，バッチアプリケーションがない場合，cjlistjob コマンドを実行しても何も出力されません。この場合，cjlistjob コマンドは正常終了します。

cjlistjob コマンドの出力形式と出力例を次に示します。なお，△は，半角スペースを表します。

**cjlistjob コマンドの出力形式**

```
<バッチアプリケーションの状態>△<バッチアプリケーション名>△<性能解析トレースのルートアプリケーション情報>△<バッチ実行コマンドの実行時刻>
```

cjlistjob コマンドの出力例

```
running com.xxx.mypackage.batchApp1 0x0000000000123456 2008/04/14 17:27:35.689012
```

この例は，cjlistjob コマンドの引数に指定したバッチサーバでは，2008/4/14 17:27:35.689012 に開始したバッチアプリケーションが実行中であることを示しています。

## 2.3.5 バッチアプリケーションのログ出力

バッチサーバでは，バッチアプリケーションの実行ログを出力します。実行ログには，実行中のバッチアプリケーションが標準出力や標準エラー出力に出力する内容を，バッチ実行コマンド単位で出力します。これらの情報は障害発生時の調査情報として利用できます。

バッチアプリケーションが java.lang.System.out および java.lang.System.err に書き出したデータは，バッチサーバによってそれぞれ次の場所に出力されます。

- バッチアプリケーションが java.lang.System.out に書き出したデータ
  バッチサーバの標準出力転送機能で，次の場所に出力されます。
  - バッチサーバのユーザ出力ログ
  - バッチサーバの標準出力
  - cjexecjob コマンドの標準出力
- バッチアプリケーションが java.lang.System.err に書き出したデータ
  バッチサーバの標準エラー出力転送機能で，次の場所に出力されます。
  - バッチサーバのユーザエラーログ
  - バッチサーバの標準エラー出力
  - cjexecjob コマンドの標準エラー出力

また，cjexecjob コマンド，cjkilljob コマンドおよび cjlistjob コマンドで出力するメッセージは，メッセージのレベルによってそれぞれ次の場所に出力されます。

I（Information）
　標準出力に出力します。ただし，コマンドの使用方法を示すメッセージ（KDJE55029-I，KDJE55030-I，KDJE55052-I）の出力先は標準エラー出力になります。

E（Error）および W（Warning）
　標準エラー出力に出力されます。

バッチアプリケーションを実行するための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「3.3　バッチ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。また，メッセージのレベルについては，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)」の「7.1　メッセージの記述形式」を参照してください。

## 2.3.6　バッチアプリケーションで使用するコマンドの実行について

ここでは，バッチアプリケーションで使用するコマンドの実行について説明します。

バッチアプリケーションで使用するコマンドには次の 3 種類があります。

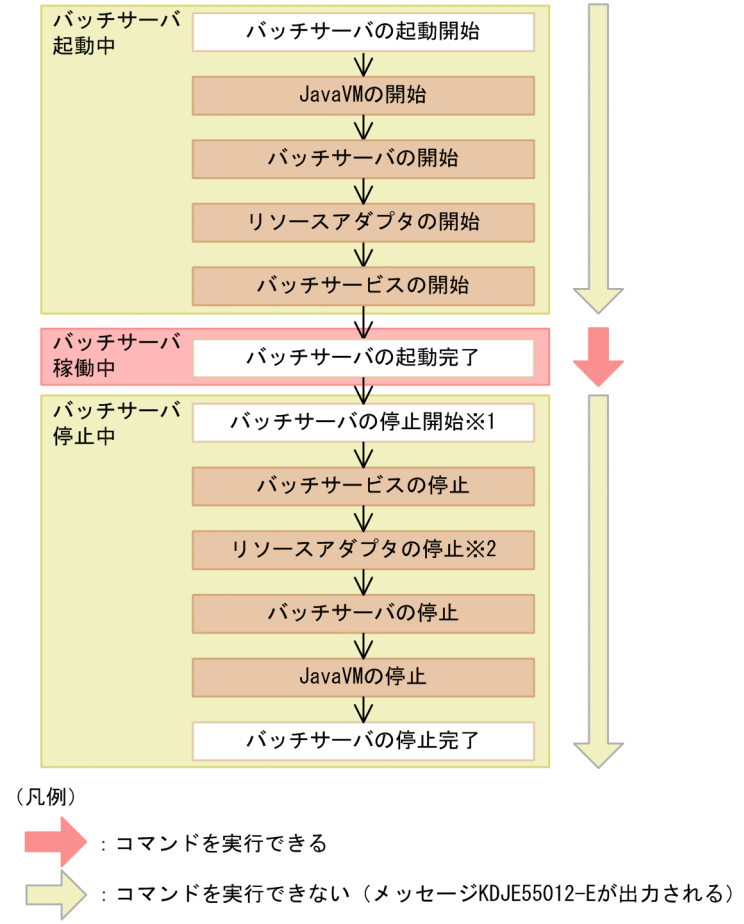
- cjexecjob コマンド（バッチ実行コマンド）
  バッチアプリケーションを実行するためのコマンドです。
- cjkilljob コマンド（バッチ強制停止コマンド）
  実行中のバッチアプリケーションを強制停止するためのコマンドです。
- cjlistjob コマンド（バッチ一覧表示コマンド）
  バッチアプリケーション情報を一覧表示するためのコマンドです。

これらのコマンドは，バッチサーバの状態によって実行できないことがあります。バッチサーバの状態とコマンドの実行について説明します。なお，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「3.3　バッチ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。

### (1)　バッチサーバの状態とコマンドの実行

cjexecjob コマンド，cjkilljob コマンドおよび cjlistjob コマンドは，バッチサーバの状態によって実行できないことがあります。バッチサーバの状態とコマンドの実行可否を次の図に示します。

図 2-10　バッチサーバの状態とコマンドの実行可否




注※1　実行中のバッチアプリケーションがある場合はバッチアプリケーションが終了するまで待機します。
注※2　開始状態のリソースアダプタがある場合はリソースアダプタを停止します。

なお，バッチサーバの停止完了後は，cjexecjob コマンド，cjkilljob コマンドおよび cjlistjob コマンドは実行できません。メッセージ KDJE55010-E が出力されます。

また，バッチサーバでほかのコマンドを処理している場合，コマンドの種類によっては実行できない場合があります。バッチサーバでコマンドを処理しているときの，コマンドの実行可否を次の表に示します。

表 2-7　バッチサーバでコマンドを処理しているときのコマンドの実行可否

| 実行するコマンド | | 処理中のコマンド | | | |
|---|---|---|---|---|---|
| | | cjexecjob | cjkilljob | cjlistjob | サーバ管理コマンド |
| cjexecjob | | × | × | ○ | ○ |
| cjkilljob | | ○ | × | ○ | ○ |
| cjlistjob | | ○ | ○ | ○ | ○ |
| サーバ管理コマンド | cjstoprar | × | × | ○ | △※1 |
| | cjstoprar 以外のコマンド | ○ | ○ | ○ | △※1 |

| 実行するコマンド | 処理中のコマンド | | | |
|---|---|---|---|---|
| | cjexecjob | cjkilljob | cjlistjob | サーバ管理コマンド |
| cjstopsv または cmx_stop_target | ○※2 | ○※2 | ○※2 | △※1 |
| cjdumpsv | ○ | ○ | ○ | ○ |

（凡例）○：実行できる　×：実行できない　△：コマンドの種類によって異なる

注※1　サーバ管理コマンドの種類によって動作が異なります。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「3.2.2　サーバ管理コマンドの排他制御」を参照してください。

注※2　実行中のバッチアプリケーションがある場合は，メッセージ KDJE55033-I を出力し，バッチアプリケーションの終了を待機します。

### (2)　コマンド処理中にバッチサーバが異常終了した場合

バッチサーバで cjexecjob コマンド，cjkilljob コマンドおよび cjlistjob コマンドを処理している場合に，バッチサーバが異常終了したときは，メッセージ KDJE55021-E が出力されます。バッチサーバの状態を確認してから再度コマンドを実行してください。

### (3)　コマンド実行時の注意事項

コマンド実行時の注意事項を次に示します。

- cjexecjob コマンド，cjkilljob コマンドおよび cjlistjob コマンド実行時に，バッチサーバがない場合，メッセージ KDJE55010-E を出力してコマンドは異常終了します。
- 簡易構築定義ファイルの ejbserver.ctm.enabled パラメタと，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.ctm.enabled キーの指定値が一致していない場合，次のコマンド実行時にエラーが発生することがあります。
  - cjexecjob コマンド実行時，メッセージ KDJE55067-E を出力してコマンドが異常終了することがあります。
  - cjlistjob コマンド実行時，バッチアプリケーション情報が出力されないことがあります。

## 2.3.7　バッチアプリケーションの実装（バッチアプリケーションの作成規則）

バッチアプリケーションとは，バッチ処理の内容を実装した Java アプリケーションです。ここでは，バッチアプリケーションの作成規則について説明します。

### (1)　バッチアプリケーションのファイル形式

バッチアプリケーションは，JavaVM で規定しているクラスファイル形式にします。なお，複数のクラスを使用する場合は次のこともできます。

- クラスファイルを配置したディレクトリをクラスパスに含める。

- クラスファイルをアーカイブした JAR ファイルをクラスパスに含める。

## (2) バッチアプリケーションに実装できる処理

バッチアプリケーションには，Java で記述できる処理を実装できます。ただし，ファイルの操作やバッチアプリケーション内で使用するスレッドなどについて，使用時の注意事項があります。アプリケーション作成時の注意については，「2.3.11　バッチアプリケーション作成時の注意」を参照してください。

## (3) バッチ処理の開始

バッチ処理の開始メソッドとして，次のどちらかのメソッドをバッチアプリケーションに定義してください。

- public static void main(String[])
- public static int main(String[])

main メソッドの戻り値の型と修飾子が異なる場合，バッチアプリケーションは実行できません。なお，main メソッドには throws を指定できます。main メソッドの引数には，cjexecjob コマンドに指定した引数が文字列配列で渡されます。

また，JavaVM 終了メソッドを使用できる設定にした場合は，バッチアプリケーションの開始時に，バッチサーバによってバッチアプリケーション実行開始スレッドが作成され，スレッドグループ（batchThreadGroup）に登録されます。JavaVM 終了メソッドは，簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタに「true」を指定した場合に使用できます。ejbserver.batch.application.exit.enabled パラメタの設定については，「2.3.10　実行環境での設定（バッチサーバの設定）」を参照してください。

## (4) バッチ処理の終了

バッチアプリケーションが次のどちらかの状態になると処理が終了します。

- cjexecjob コマンドの引数に指定したクラスの main メソッドの実行が終了する。
- 例外やエラーが main メソッドの外にスローされる。

また，次のどれかの状態になると，バッチアプリケーションのスレッド（batchThreadGroup に属するスレッド）が終了します。

- JavaVM 終了メソッドを呼び出す。
- main メソッドがリターンする。
- main スレッドで発生した例外がキャッチされない。

バッチアプリケーション終了時に使用できる終了処理を次の表に示します。

表 2-8　バッチアプリケーション終了時に使用できる終了処理

| バッチアプリケーションの終了方法 | | 使用できる終了処理 | |
|---|---|---|---|
| | | java.io.deleteOnExit | シャットダウンフック |
| JavaVM 終了メソッドの呼び出しによる終了 | java.lang.System.exit(int)の呼び出しによる終了 | ○ | ○ |
| | java.lang.Runtime.exit(int)の呼び出しによる終了 | ○ | ○ |
| | java.lang.Runtime.halt(int)の呼び出しによる終了 | ○ | × |
| [Ctrl] ＋ [C] による終了 | | × | × |
| main メソッドのリターンによる終了 | | ○ | ○ |
| main スレッドでの例外発生による終了 | | ○ | ○ |

（凡例）○：使用できる　×：使用できない

なお，JavaVM 終了メソッドは，簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタに「true」を指定した場合に使用できます。ejbserver.batch.application.exit.enabled パラメタの設定については，「2.3.10　実行環境での設定（バッチサーバの設定）」を参照してください。

## 2.3.8　バッチアプリケーションの実装（リソースに接続する場合）

ここでは，リソースに接続するバッチアプリケーションの作成方法について説明します。新規にバッチアプリケーションを作成する場合と，既存のバッチアプリケーションから移行する場合について説明します。

### (1)　新規にバッチアプリケーションを作成する場合

新規にバッチアプリケーションを作成する場合，リソースへの接続には DB Connector を使用することをお勧めします。DB Connector とは，アプリケーションサーバで提供するデータベースに接続するためのリソースアダプタです。DB Connector を使用したリソースに接続する方法を次に示します。

1. バッチサーバで DB Connector を設定します。

   ユーザ指定名前空間機能を使用して，DB Connector のオブジェクトに別名を付けて JNDI 名前空間に登録します。バッチアプリケーションからデータベースに接続するときには，必ずユーザ指定名前空間機能を使用してください。

   別名は，DB Connector をバッチサーバにデプロイしたあと，Connector 属性ファイルで設定します。次の設定例のように，Connector 属性ファイルの<resource-external-property>タグに<optional-name>タグを追加して別名を設定します。

   **設定例**

```
<connector-runtime>
      :
```

```
    <resource-external-property>
        <optional-name>DB Connectorの別名</optional-name>
    </resource-external-property>
</connector-runtime>
```

DB Connector の別名の付け方については，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。

また，DB Connector の設定の流れについては，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　リソース接続」を参照してください。

2. 1.で設定した別名で DB Connector をルックアップし，コネクションファクトリ（javax.sql.DataSource インタフェース）を取得します。

取得したコネクションファクトリから java.sql.Connection を取得します。コーディング例を次に示します。

```
String dbName = <DB Connectorの別名>;
     InitialContext ic = new InitialContext();
     DataSource ds = (DataSource) ic.lookup(dbName);
     Connection con =  ds.getConnection();
```

3. 取得した java.sql.Connection を使用して，リソースに接続します。

JDBC ドライバの提供する java.sql.Connection と API は同じです。

**注意事項**

DB Connector を使用する場合，バッチサーバで DB Connector を開始してからバッチアプリケーションを開始してください。

## (2)　既存のバッチアプリケーションから移行する場合

既存のバッチアプリケーション（Java アプリケーション）から移行する場合，リソースに接続する方法は次の 2 種類があります。

- Application Server で提供する DB Connector を使用したリソース接続に変更する。
- JDBC ドライバを使用してリソースに接続する（接続方法を変更しない）。

DB Connector を使用しない場合，バッチアプリケーションのコードを修正する必要はありません。ただし，DB Connector が提供する機能および GC 制御機能は利用できません。ここでは，リソースの接続方法を DB Connector に変更する場合の移行方法と，JDBC ドライバを使用する場合（接続方法を変更しない場合）の移行方法を説明します。

### (a)　DB Connector を使用したリソース接続に変更する

DB Connector を使用する場合，DB Connector から java.sql.Connection を取得するようバッチアプリケーションを変更してください。変更方法を次に示します。

1. バッチサーバで DB Connector を設定します。

ユーザ指定名前空間機能を使用して，DB Connector のオブジェクトに別名を付けて JNDI 名前空間に登録します。バッチアプリケーションからデータベースに接続するときには，必ずユーザ指定名前空間機能を使用してください。

別名は，DB Connector をバッチサーバにデプロイしたあと，Connector 属性ファイルで設定します。次の設定例のように，Connector 属性ファイルの<resource-external-property>タグに<optional-name>タグを追加して別名を設定します。

**設定例**

```
<connector-runtime>
      :
   <resource-external-property>
       <optional-name>DB Connectorの別名</optional-name>
   </resource-external-property>
</connector-runtime>
```

DB Connector の別名の付け方については，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。

また，DB Connector の設定の流れについては，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　リソース接続」を参照してください。

2. バッチアプリケーション内のリソース接続処理のコードを DB Connector を使用するよう変更します。

変更前のバッチアプリケーションを次に示します。下線部分は Connection 取得処理です。この処理を「変更後のバッチアプリケーション」の下線部分の処理に変更してください。「変更後のバッチアプリケーション」の下線部分は，DB Connector の Connection 取得処理です。

- 変更前のバッチアプリケーション

```
Class.forName(”oracle.jdbc.driver.OracleDriver”);
Connection con = DriverManager.getConnection(uri,”user”,”pass”);
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeBatch();
con.commit();
```

- 変更後のバッチアプリケーション

```
String dbName = <DB Connectorの別名>
InitialContext ic = new InitialContext();
DataSource ds = (DataSource)ic.lookup(dbName);
Connection con = ds.getConnection();
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeBatch();
con.commit();
```

DB Connector から取得した java.sql.Connection は，JDBC ドライバの java.sql.Connection と同様に使用できます。このため，java.sql.Connection の取得方法だけを変更すれば，ほかのバッチアプリケーションのコードを変更する必要はありません。

**注意事項**

DB Connector を使用する場合，バッチサーバで DB Connector を開始してからバッチアプリケーションを実行してください。

### (b)　JDBC ドライバを使用してリソースに接続する

JDBC ドライバを使用する場合，バッチアプリケーションのコードを修正する必要はありません。ただし，使用する JDBC ドライバのライブラリをバッチサーバのクラスパスに追加する必要があります。詳細は，使用する JDBC ドライバの設定に従ってください。次に，JDBC ドライバのライブラリをバッチサーバのクラスパスに追加する方法を示します。バッチサーバのクラスパスに追加するには，usrconf.cfg（バッチサーバ用オプション定義ファイル）に次の記述を追加します。

add.class.path=<JDBC ドライバのライブラリのフルパス>

なお，usrconf.cfg（バッチサーバ用オプション定義ファイル）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(サーバ定義)」の「3.2.1　usrconf.cfg（バッチサーバ用オプション定義ファイル)」を参照してください。

## (3)　リソースに接続するバッチアプリケーションの注意

リソースに接続するバッチアプリケーションを作成するときには，次のことに注意してください。

### (a)　バッチアプリケーション実行時の注意

バッチアプリケーション実行中には，DB Connector の停止や設定変更をしないでください。DB Connector の停止や設定変更は，バッチアプリケーションが終了してから実施します。

### (b)　コネクションのクローズ

バッチサーバでは，コネクションの自動クローズは実行されません。このため，使用したコネクションは必ずクローズするよう，アプリケーションに実装してください。

### (c)　JTA のローカルトランザクションの使用

バッチアプリケーションの中で，JTA のローカルトランザクションを使用できます。JTA のローカルトランザクションは，次に示す方法で使用します。

1. 次のどちらかの方法で UserTransaction オブジェクトを取得します。
   - ネーミングサービスからルックアップして取得する。
     ルックアップ名：HITACHI_EJB/SERVERS/<サーバ名称>/SERVICES/UserTransaction
   - com.hitachi.software.ejb.ejbclient.UserTransactionFactory クラスの getUserTransaction メソッドを使用して取得する。
2. UserTransaction オブジェクトの begin()メソッドを呼び出して，トランザクションを開始します。

3. リソースに接続します。

4. UserTransaction オブジェクトの commit()または rollback()を呼び出して，トランザクションを決着します。

UserTransaction インタフェースの使用方法の詳細については，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「3.4.8　UserTransaction インタフェースを使用する場合の処理概要と留意点」を参照してください。

UserTransaction 使用時の注意事項を次に示します。

- UserTransaction は main スレッドだけ使用できます。ユーザスレッドでは使用できません。
- main スレッドでトランザクションの開始および決着を実施してください。
- スレッド生成時にトランザクションは引き継がれません。
- スレッド間でコネクションやコネクションから得られたインスタンス（ステートメントなど）を渡せません。このインスタンスを使用した場合は動作が不正になります。

### (d)　トランザクションの決着

バッチアプリケーション内でトランザクションを開始した場合は，バッチアプリケーション内で必ず決着処理を実施してください。トランザクションの決着処理を実施しないでバッチアプリケーションを終了すると，タイムアウト時間が経過したあとにトランザクションがロールバックされます。

この場合，簡易構築定義ファイルの ejbserver.batch.application.exit.enabled パラメタの指定値によって，次に実行するバッチアプリケーションでトランザクションを開始する時（javax.transaction.UserTransaction#begin()）の挙動が異なります。

**ejbserver.batch.application.exit.enabled パラメタで「true」を指定している場合**

次に実行するバッチアプリケーションでトランザクションを開始できます（javax.transaction.UserTransaction#begin()を受け付けます）。

**ejbserver.batch.application.exit.enabled パラメタで「false」を指定している場合**

次に実行するバッチアプリケーションでトランザクションを開始できません。この場合，javax.transaction.NotSupportedException が発生し，詳細情報として「KDJE31009-E No nested transaction is supported.」が出力されます。

トランザクションを開始できない状態から回復する場合は，バッチサーバを再起動してください。

ejbserver.batch.application.exit.enabled パラメタの設定については，「2.3.10　実行環境での設定（バッチサーバの設定)」を参照してください。

## 2.3.9　バッチアプリケーションの実装（EJB にアクセスする場合）

バッチアプリケーションから J2EE アプリケーションの EJB にアクセスできます。EJB にアクセスするバッチアプリケーションを作成する場合，アクセスする EJB を次に示す名称でルックアップして使用できます。

- 自動的にバインドされる名称（Portable Global JNDI 名または HITACHI_EJB から始まる名称）
- ユーザ指定名前空間機能を使用した別名

EJB にアクセスする場合，次に示す手順でバッチアプリケーションを準備します。

1. バッチアプリケーションからアクセスする EJB の準備
   バッチアプリケーションからアクセスする EJB を含む J2EE アプリケーションを開始状態にします。
2. バッチアプリケーションの実装
   バッチアプリケーション内に，EJB を使用するためのコードを実装します。
3. バッチアプリケーションの実行
   2.で作成したバッチアプリケーションを実行します。

それぞれの手順の詳細を次に説明します。

### (1)　EJB の準備

バッチアプリケーションからアクセスする EJB を持つ J2EE アプリケーションを用意します。また，J2EE アプリケーションを実行するための J2EE サーバも用意します。J2EE サーバの構築については，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「4.1　Web サーバを別ホストに配置してマシン性能を向上するシステムの構築」を参照してください。

構築した J2EE サーバ上で，J2EE アプリケーションを開始します。cjgetstubsjar コマンドを使用して，開始した J2EE アプリケーションの RMI-IIOP スタブおよびインタフェースを取得しておきます。

なお，バッチアプリケーションから EJB にアクセスする場合，別名によるルックアップをするときは，事前にユーザ指定名前空間機能を使用して EJB の別名を設定しておいてください。EJB の別名の設定については，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(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。

### (2)　バッチアプリケーションの実装

「2.3.9(1)　EJB の準備」で設定した EJB を取得するためのコードを，バッチアプリケーションに実装します。コードの例を次に示します。

```
String EjbName = <EJBのルックアップ名>;
InitialContext ic = new InitialContext();
Object objref = ic.lookup(EjbName);
<ホームインタフェースクラス名> home =
   (<ホームインタフェースクラス名>) PortableRemoteObject.narrow(objref,
```

```
                                <ホームインタフェースクラス名>.class);
<EJBオブジェクトクラス名> ejbobj = home.create();
```

ホームインタフェースおよびEJBオブジェクトファイルはあらかじめ準備しておいてください。バッチアプリケーションのコンパイル時および実行時にクラスパスに含める必要があります。

### (3) バッチアプリケーションの実行

バッチアプリケーションを実行する場合，クラスパスに「2.3.9(1) EJBの準備」で取得したスタブや「2.3.9(2) バッチアプリケーションの実装」で使用したインタフェースファイルをフルパスで指定します。

EJBを検索するネーミングサービスのURLは，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）のjava.naming.provider.urlの値として指定します。

ただし，リソース接続機能とEJBアクセス機能を同時に使用する場合は，ネーミングサービス切り替え機能を使用して，EJBをルックアップするネーミングサービスを指定してください。この場合，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）のjava.naming.provider.urlは指定しないでください。ネーミングサービス切り替え機能については，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「2.10 CORBAネーミングサービスの切り替え」を参照してください。

## 2.3.10 実行環境での設定（バッチサーバの設定）

バッチアプリケーション実行機能を使用する場合，バッチサーバの設定が必要です。バッチサーバの設定は，簡易構築定義ファイルで実施します。

**注意事項**

デフォルトの設定では，スケジューリング機能を使用しない設定（false）になっています。スケジューリング機能を使用しない場合は，次のパラメタおよびキーの設定を変更しないでください。

- 簡易構築定義ファイルの論理J2EEサーバ（j2ee-server）のejbserver.ctm.enabledパラメタ
- usrconf.cfg（バッチアプリケーション用オプション定義ファイル）のbatch.ctm.enabledキー

バッチアプリケーション実行機能の定義は，簡易構築定義ファイルの論理J2EEサーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのバッチアプリケーション実行機能の定義を次の表に示します。

## 表 2-9　簡易構築定義ファイルでのバッチアプリケーション実行機能の定義

| 項目 | 指定するパラメタ | 指定内容 | 必須または任意 |
|---|---|---|---|
| バッチサーバとしてサーバを構築するための設定 | batch.service.enabled | バッチサーバとして構築するために，必ず true を指定してください。 | 必須 |
| SecurityManager を使用しない設定 | use.security | SecurityManager は使用しません。パラメタの値には必ず false を指定してください。 | 必須 |
| ライトトランザクション機能を有効にするための設定 | ejbserver.distributedtx.XATransaction.enabled | グローバルトランザクションは使用できません。ローカルトランザクションを使用します※1。パラメタの値には必ず false を指定してください。なお，このパラメタはデフォルトの設定が false のため，変更しないでください。 | 必須 |
| 明示管理ヒープ機能を使用しない設定 | add.jvm.arg | バッチアプリケーションで明示管理ヒープ機能を実装していない場合は，明示管理ヒープ機能を無効にすることをお勧めします。明示管理ヒープ機能を無効にするには，パラメタの値に-XX:-HitachiUseExplicitMemory を指定してください。デフォルトの設定の場合，明示管理ヒープ機能は有効（-XX:+HitachiUseExplicitMemory）です。 | 任意 |
| 実サーバ名の設定 | realservername | バッチサーバの実サーバ名を指定します。実サーバ名は，同一ホスト内でユニークになるように指定してください。省略した場合は，論理サーバ名が設定されます。 | 任意 |
| JavaVM 終了メソッド呼び出し時の JavaVM の動作設定 | ejbserver.batch.application.exit.enabled | 次の JavaVM 終了メソッドをバッチアプリケーションで呼び出した時に，JavaVM を終了するかどうかを指定します。<br>• java.lang.System.exit(int)<br>• java.lang.Runtime.exit(int)<br>• java.lang.Runtime.halt(int)<br>デフォルトは「true」（JavaVM を終了しないでバッチアプリケーションのスレッドを終了する）です。<br>「true」を指定，または設定を省略した場合には，JavaVM 終了メソッドの呼び出し時に，バッチアプリケーションのスレッド（batchThreadGroup に属するスレッド）が終了され，JavaVM は終了されません。<br>「false」を指定した場合には，JavaVM 終了メソッドの呼び出し時に，バッチサーバごと JavaVM が終了されます。このため，バッチアプリケーションでは，JavaVM 終了メソッドおよびシャットダウンフックが使用できません。※2 | 任意 |

（凡例）必須：必ず指定する　任意：必要に応じて指定する

注　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)」を参照してください。

注※1　バッチサーバの場合，ローカルトランザクションに最適化された環境を提供する，ライトトランザクション機能を使用します。ライトトランザクション機能は，ejbserver.distributedtx.XATransaction.enabled パラメタに「false」を指定すると有効になります。

注※2　ejbserver.batch.application.exit.enabled パラメタで「false」を指定した場合には，JavaVM 終了メソッドおよびシャットダウンフックが使用できません。次に示す手段で対処してください。

- JavaVM 終了メソッドに対する対処

  public static int main(String[])メソッドにバッチ処理の内容を記述します。終了コードを返す場合は，return <終了コード>を使用します。ただし，return を使用する場合は，finally ブロックが実行されます。

- シャットダウンフックに対する対処

  バッチアプリケーション終了時に実施したい処理がある場合は，main メソッドの finally ブロック内に処理を記述してください。

## 2.3.11　バッチアプリケーション作成時の注意

ここでは，バッチアプリケーション作成時に注意が必要となる処理や，バッチアプリケーションでは使用できない機能について説明します。これらの内容を確認の上，バッチアプリケーションを作成してください。

### (1)　注意が必要な処理

次に示す処理は，バッチアプリケーションを作成する際に注意が必要です。

#### (a)　ファイルやディレクトリの操作

バッチアプリケーションでは，次に示すファイルやディレクトリは操作しないでください。

- Application Server のインストールディレクトリ以下のファイルやディレクトリ

  Application Server のインストールディレクトリ以下のファイルやディレクトリについては，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 B　インストール後のディレクトリ構成」を参照してください。

- バッチサーバの作業ディレクトリ以下のファイルやディレクトリ

  バッチサーバの作業ディレクトリについては，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 C.2　バッチサーバの作業ディレクトリ」を参照してください。

また，バッチアプリケーションでファイルやディレクトリを扱う場合，ファイルやディレクトリのパスに相対パスを使用できません。cjexecjob コマンドを実行したディレクトリからの相対パスを取得したい場合は，ejbserver.batch.currentdir の値を使用してください。ejbserver.batch.currentdir 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」の「ejbserver.batch.currentdir プロパティ」を参照してください。

次に，バッチアプリケーションの修正例を示します。

修正前

```
File f = new File(”DataFile.txt”);
```

修正後

```
File f = new File(System.getProperty(”ejbserver.batch.currentdir”) + System.getProperty(”
file.separator”) + ”DataFile.txt”);
```

## (b) スレッドの使用

バッチサーバは，バッチアプリケーションが作成および開始したスレッドの終了を待ちません。バッチアプリケーション内でスレッドを使用する場合は，バッチアプリケーションを終了する前に，開始したすべてのユーザスレッドを完了するように実装してください。また，ユーザスレッドはメソッドキャンセルの対象外です。

なお，簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタで「true」を指定した場合には，次の点に注意してください。

- スレッドグループ（ThreadGroup）を作成できません。
- バッチアプリケーションに java.lang.Thread クラスのインタフェースである UncaughtExceptionHandler を継承したハンドラを登録していると，JavaVM 終了メソッド呼び出し時に登録したハンドラの処理が実行されます。このとき，uncaughtException メソッドの引数に JP.co.Hitachi.soft.jvm.SpecialThrowable の例外が渡されることがあります。

バッチアプリケーションが作成したスレッドが残っていると，バッチアプリケーションのクラスや使用したリソースは解放されません。このため，次にバッチアプリケーションを開始しようとすると，バッチアプリケーションの開始に失敗するおそれがあります。また，ユーザスレッド内では，次のバッチサーバの機能を呼び出すことはできません。

- バッチアプリケーション実行機能
- EJB アクセス機能
- ネーミング管理が提供する機能
- リソース接続とトランザクション管理が提供する機能
- GC 制御機能
- コンテナ拡張ライブラリが提供する機能

## (c) JavaVM 終了時のリソースの自動クローズ

バッチサーバではサーバの JavaVM 上でバッチアプリケーションを実行します。このため，JavaVM 終了による自動的なリソースのクローズ処理を期待した実装をしている場合は，メモリやファイルディスクリプタのリークが発生します。例えば，次の場合にリークが発生します。

- ZIP ファイルまたは JAR ファイルをオープンしている場合，明示的にクローズしないと C ヒープ領域がリークします。

- バッチサーバの usrconf.properties に ejbserver.batch.application.exit.enabled=false を指定した場合，java.io.File.deleteOnExit()を使用しても，バッチサーバが停止するまでファイルは削除されません。バッチサーバが停止するまで C ヒープ領域がリークします。

この問題を回避するためには，リソースが正しくクローズされるようにバッチアプリケーションを実装してください。

また，ファイルやソケットなども明示的にクローズしていないと，リソース解放のタイミングが不定になります。これによって，次回以降のバッチアプリケーションの実行に影響を及ぼすおそれがあります。ファイルやソケットは明示的にクローズするようにしてください。

なお，バッチサーバの場合，コネクションの自動クローズは使用できません。バッチアプリケーション内で必ずコネクションをクローズしてください。

### (d)　JavaVM 終了メソッドの使用

簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタに「true」を指定すると，次の JavaVM 終了メソッドが使用できます。

- java.lang.System.exit(int)
- java.lang.Runtime.exit(int)
- java.lang.Runtime.halt(int)

ejbserver.batch.application.exit.enabled パラメタの設定については，「2.3.10　実行環境での設定（バッチサーバの設定）」を参照してください。なお，JavaVM 終了メソッドを使用する場合の注意事項については，「2.3.11(3)　JavaVM 終了メソッド使用時の注意」を参照してください。

### (e)　シャットダウンフックの使用

簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタに「true」を指定すると，次の場合にシャットダウンフックが使用できます。

- JavaVM 終了メソッドを呼び出した場合
- main メソッドがリターンした場合
- main スレッドで発生した例外がキャッチされなかった場合

ejbserver.batch.application.exit.enabled パラメタの設定については，「2.3.10　実行環境での設定（バッチサーバの設定）」を参照してください。

## (2)　バッチアプリケーションで実装できない機能

次に示す機能はバッチアプリケーションでは使用できません。「対処方法」に示す手段で対応してください。

- **標準入力からの入力**
  java.lang.System.in などを使用した標準入力からの入力処理はできません。

対処方法

入力処理が必要な場合はファイルを使用してください。

- **JNI の使用**

  JNI 経由でのネイティブライブラリの実行機能は使用できません。

  対処方法

  JNI を使用する場合は，コンテナ拡張ライブラリ経由で使用してください。このとき，ネイティブライブラリのロードはコンテナ拡張ライブラリ内で実施します。

- **システムプロパティのセットの置き換え**

  次に示すメソッドは使用できません。

  - java.lang.System.setProperties(java.util.Properties)

  対処方法

  java.lang.System.setProperty(String, String)を使用します。

- **標準出力ストリームおよび標準エラー出力ストリームの割り当てのし直し**

  次に示すメソッドは使用できません。

  - java.lang.System.setOut(java.io.PrintStream)
  - java.lang.System.setErr(java.io.PrintStream)

  対処方法

  java.lang.System.out および java.lang.System.err を使用しないで，出力したい PrintStream オブジェクトを直接使用します。

## (3) JavaVM 終了メソッド使用時の注意

簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタに「true」を指定すると，バッチアプリケーションで JavaVM 終了メソッドを使用しても，JavaVM は終了されません。この場合，JavaVM 終了メソッドが呼び出したスレッドだけを終了できます。

ここでは，ejbserver.batch.application.exit.enabled パラメタに「true」を指定している場合に，JavaVM 終了メソッドを使用するときの注意事項について説明します。

### (a) Java 言語仕様との差異

バッチアプリケーションで使用する JavaVM 終了メソッドは，Java 言語仕様と仕様が異なります。Java 言語仕様との差異を次の表に示します。

表 2-10 Java 言語仕様との差異

| 項目 | Java 言語仕様の場合 | バッチアプリケーションの場合 |
|---|---|---|
| 終了対象 | JavaVM | JavaVM 終了メソッドを呼び出したスレッド |

| 項目 | Java 言語仕様の場合 | バッチアプリケーションの場合 |
|---|---|---|
| 呼び出し以降に記述された java ロジック | JavaVM 終了メソッドの呼び出し以降に記述されている処理は実行されません。 | 次の場合，JavaVM 終了メソッドの呼び出し以降に記述されている処理は実行されます。<br>• JavaVM 終了メソッドが try ブロックに記述された場合は，対応する finally ブロックが実行されます。※1<br>• スレッドに java.lang.Thread.UncaughtExceptionHandler が登録されていた場合は，UncaughtExceptionHandler が実行されます。※1 |
| 複数回呼び出し | 使用できません。 | 次の場合に複数回呼び出されます。<br>• JavaVM 終了メソッドを呼び出したスレッドと同一の finally ブロックから，JavaVM 終了メソッドを呼び出す場合<br>• バッチアプリケーションから起動された複数のユーザスレッド※2 から，JavaVM 終了メソッドを呼び出す場合 |

注※1　finally ブロック内および finally ブロック内で呼び出されたメソッド内で例外が発生し，その例外が finally ブロックでキャッチされないで finally ブロックの実行が途中で中断された場合は，スレッドが終了できないときがあります。

また，次の場合，スレッドの終了に時間が掛かったり，スレッドが終了できなかったりするときがあります。

- finally ブロック内および finally ブロック内で呼び出されたメソッド内に，時間が掛かる Java プログラム処理が記述された場合
- java.lang.Thread.UncaughtExceptionHandler に，時間が掛かる Java プログラム処理が記述された場合

なお，時間の掛かる Java プログラムの処理には，無限ループ，synchronized 文によるモニタ待ち，java.lang.wait()による待ちなどがあります。

注※2　ユーザスレッドとは，バッチアプリケーションが作成した子スレッドを示します。ユーザスレッドを使用している場合は，次の点に注意してください。

- run()メソッドの中で interruptedException がキャッチされると，ユーザスレッドは終了されないで残ります。
- ユーザスレッドが残っていても，main スレッドが終了していれば，次のアプリケーションの開始を受け付けられますが，メモリリークが発生します。

**参考**

JavaVM 終了メソッド実行時に JavaVM を終了したい場合は，簡易構築定義ファイルで ejbserver.batch.application.exit.enabled パラメタに「false」を指定します。「false」を指定すると，JavaVM 終了メソッド実行時に，バッチサーバごと JavaVM が終了します。

## (b)　JavaVM 終了メソッドを呼び出した場合の処理

バッチアプリケーションごとに，JavaVM 終了メソッドを呼び出した場合の処理を説明します。

### シングルスレッドで実装されたバッチアプリケーションで JavaVM 終了メソッドを呼び出した場合

main スレッドを終了し，バッチアプリケーションの実行を終了します。

JavaVM 終了メソッドが複数回呼び出された場合の動作を次の表に示します。

表 2-11　JavaVM 終了メソッドが複数回呼び出された場合の動作（シングルスレッドで実装されたバッチアプリケーションの場合）

| 項番 | 項目 | 動作 |
|---|---|---|
| 1 | バッチアプリケーション実行機能への終了通知 | 1 回目の JavaVM 終了メソッド呼び出し時にだけ終了を通知します。<br>2 回目以降の JavaVM 終了メソッド呼び出し時には通知しません。 |
| 2 | 終了コードの返却 | 1 回目の JavaVM 終了メソッド呼び出し時に引数に指定された終了コードが有効となります。<br>2 回目以降の JavaVM 終了メソッド呼び出し時に引数に指定された終了コードは無効となります。 |
| 3 | JavaVM 終了メソッドを呼び出したスレッド | JavaVM 終了メソッドの呼び出し回数に関係なく，スレッドが終了します。 |

**マルチスレッドで実装されたバッチアプリケーションで JavaVM 終了メソッドを呼び出した場合**

JavaVM 終了メソッドを呼び出したスレッドが終了します。それ以外のスレッドの処理は，JavaVM 終了メソッドがどのスレッドから呼び出されたかによって，異なります。

- main スレッドで JavaVM 終了メソッドを呼び出した場合，main メソッドがリターンした場合，または main スレッドで発生した例外がキャッチされなかった場合

  バッチアプリケーション実行機能が，すべての実行中のユーザスレッドに対して，java.lang.Thread クラスの interrupt を実行します。

- ユーザスレッドで JavaVM 終了メソッドを呼び出した場合

  バッチアプリケーション実行機能が，次に示すスレッド以外の，すべての実行中のユーザスレッドに対して，java.lang.Thread クラスの interrupt を実行します。

  ・JavaVM 終了メソッドを呼び出したユーザスレッド

  ・main スレッド

  バッチアプリケーション実行機能は，ユーザスレッドを呼び出している main スレッドに対して，メソッドキャンセルを実行します。メソッドキャンセルが成功した場合，main スレッドが終了し，バッチアプリケーションが終了します。メソッドキャンセルが失敗した場合は，バッチサーバごと JavaVM が終了します。

  ユーザスレッドでの JavaVM 終了メソッド呼び出しは，メソッドキャンセルが失敗することがあるため，推奨しません。

どちらの場合も，main スレッドが終了すると，ユーザスレッドが残っているかどうかに関係なく，次のバッチアプリケーションの開始を受け付けられます。

## 2.4　EJB アクセス機能

バッチアプリケーションからほかの J2EE アプリケーションにある EJB にアクセスできます。この機能を EJB アクセスといいます。この節では，EJB アクセスで使用できる機能（EJB アクセス機能）について説明します。

なお，EJB にアクセスするバッチアプリケーションの作成方法については，「2.3.9　バッチアプリケーションの実装（EJB にアクセスする場合）」を参照してください。

この節の構成を次の表に示します。

表 2-12　この節の構成（EJB アクセス機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | EJB アクセスで使用できる機能 | 2.4.1 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.4.2 |

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。

### 2.4.1　EJB アクセスで使用できる機能

EJB アクセスで使用できる機能について，次の表に示します。それぞれの機能の詳細については，参照先の説明を参照してください。

表 2-13　EJB アクセスで使用できる機能

| 分類 | | 機能 | 説明 | 参照先マニュアル※ | 参照個所 |
|---|---|---|---|---|---|
| JNDI | 基本機能 | JNDI 名前空間へのオブジェクトのバインドとルックアップ | 自動的にバインドされる名称（Portable Global JNDI 名または HITACHI_EJB から始まる名称），またはユーザ指定名前空間を利用して，EJB ホームオブジェクトおよびビジネスインタフェースのリファレンスをネーミングサービスからルックアップできます。 | 基本・開発編(コンテナ共通機能) | 2.3 |
| | 拡張機能 | ラウンドロビンポリシーによる CORBA ネーミングサービスの検索 | 複数のネーミングサービスと J2EE サーバで構成されるシステムに対して，バッチアプリケーションからのルックアップをラウンドロビンで実行できます。これによって，負荷分散を実現できます。 | | 2.7 |
| | | ネーミング管理機能でのキャッシング | ネーミングサービスからルックアップしたオブジェクトをメモリ上に保持（キャッシュ）できます。キャッシュの利用によって，ネーミングサービスへのアクセスの性能上のコストを削減できます。 | | 2.8 |

| 分類 | 機能 | 説明 | 参照先マニュアル※ | 参照個所 |
|---|---|---|---|---|
| EJB | Enterprise Beanの実行 | EJBコンテナで実行されているEnterprise Beanをバッチアプリケーションから呼び出せます。ただし，呼び出し方法はリモート呼び出しだけ使用できます。ローカル呼び出しはできません。 | 基本・開発編(EJBコンテナ) | 2.2 |
| | Enterprise Beanの呼び出し | | | 3.4 |
| | RMI-IIOPスタブ，インタフェースの取得 | バッチアプリケーションから，TPBrokerのRMI-IIOPの機能を利用してアプリケーションを呼び出せます。 | | 3.7 |
| | EJBのリモートインタフェースの呼び出し | バッチアプリケーションからのEJB呼び出し実行時に通信障害が発生した場合に，送信動作を選択できます。 | | 2.13 |
| トランザクション | トランザクション管理 | バッチアプリケーションでトランザクションを開始・決着できます。ただし，バッチアプリケーションの場合，グローバルトランザクションは使用できません。 | 基本・開発編(コンテナ共通機能) | 3.4 |
| | EJBクライアントアプリケーションでのトランザクションの実装 | バッチアプリケーションでUserTransactionを取得し，トランザクションを開始・決着できます。UserTransactionの取得方法には次の2種類の方法があります。<br>1. UserTransactionFactoryクラスを使用する方法<br>2. ルックアップを使用する方法 | 基本・開発編(EJBコンテナ) | 3.5 |
| そのほか | EJBコンテナでのタイムアウトの設定 | バッチサーバとネーミングサービス間，およびバッチサーバとJ2EEサーバ間の通信で，RMI-IIOP通信のタイムアウトを設定できます。<br>バッチアプリケーションの場合，Stateful Session Beanのタイムアウト，Entity BeanのEJBオブジェクトのタイムアウト，およびインスタンス取得待ちのタイムアウトの説明は該当しません。 | 基本・開発編(EJBコンテナ) | 2.11 |
| | 性能解析トレースを使用したシステムの性能解析 | バッチアプリケーションの性能解析トレースを出力できます。 | 保守／移行編 | 7章 |
| | アプリケーションのユーザログ出力 | バッチアプリケーションのログを出力できます。 | このマニュアル | 8章 |

注※　参照先のマニュアル名称は，「アプリケーションサーバ 機能解説」を省略しています。

## 2.4.2　実行環境での設定（バッチサーバの設定）

EJBアクセス機能を使用する場合，バッチサーバの設定が必要です。

バッチサーバの設定は，簡易構築定義ファイルで実施します。EJBアクセス機能の定義は，簡易構築定義ファイルの論理J2EEサーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの EJB アクセス機能の定義を次の表に示します。

### 表 2-14　簡易構築定義ファイルでの EJB アクセス機能の定義

| 項目 | 指定するパラメタ | 設定内容 |
|---|---|---|
| RMI-IIOP 通信のタイムアウト | ejbserver.rmi.request.timeout | RMI-IIOP 通信のクライアントとサーバ間の通信タイムアウト時間を指定します。 |
| リモートインタフェースでの通信障害発生時の EJB クライアントの動作 | ejbserver.container.rebindpolicy | EJB メソッドの呼び出し時に通信障害が発生した場合のバッチサーバ側でのコネクションの再接続動作とリクエストの再送動作を指定します。 |
| バッチサーバの通信ポートと IP アドレスの固定 | vbroker.se.iiop_tp.scm.iiop_tp.listener.port | バッチサーバの通信ポートを指定します。 |
| | vbroker.se.iiop_tp.host | バッチサーバの使用する IP アドレスまたはホスト名を固定するかどうかを指定します。 |

注　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)」を参照してください。

## 2.5　ネーミング管理機能

**ネーミング管理**は，J2EE サービスで提供されている機能の一つです。J2EE サービスは，J2EE コンテナの部品機能として利用される機能です。ネーミング管理では，オブジェクト（Enterprise Bean に対応する EJB ホームオブジェクト，ビジネスインタフェースのリファレンスおよび J2EE リソース）の名前と格納場所を管理しています。ネーミング管理の機能を使用することで，バッチアプリケーションは，呼び出す Enterprise Bean またはリソースの格納場所を知らなくても，名前から必要なオブジェクトを利用できるようになります。この節では，バッチサーバで使用できるネーミング管理機能および設定方法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 2-15　この節の構成（ネーミング管理機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | バッチサーバで使用できるネーミング管理機能 | 2.5.1 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.5.2 |

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。

## 2.5.1　バッチサーバで使用できるネーミング管理機能

バッチサーバで使用できるネーミング管理機能を次の表に示します。ネーミング管理機能の詳細は，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「2.　ネーミング管理」を参照してください。

### 表 2-16　ネーミング管理機能

| 機能 | 説明 |
|---|---|
| JNDI 名前空間へのオブジェクトのバインドとルックアップ | オブジェクトを JNDI 名前空間の名前とバインドして管理します。バッチアプリケーションからは，バインドされた名前を使用してルックアップできます。バッチアプリケーションの場合，java:comp/env でのルックアップは使用できません。 |
| 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） | J2EE リソースに別名を付与できます。バッチアプリケーションからは，別名として設定した任意の名称でルックアップできます。なお，バッチアプリケーションからデータベースに接続する場合，J2EE リソースには必ず別名を設定してください。<br>J2EE リソースについては，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。バッチアプリケーションの場合，Enterprise Bean の説明は該当しません。 |

| 機能 | 説明 |
|---|---|
| ラウンドロビンポリシーによる CORBA ネーミングサービスの検索 | 複数の CORBA ネーミングサービスに登録されている同一名称（別名）の EJB ホームオブジェクトリファレンスを，ラウンドロビンポリシーに従ってルックアップできます。 |
| ネーミング管理機能でのキャッシング | ルックアップした EJB ホームオブジェクトリファレンスをキャッシングしておき，2 回目以降に同じオブジェクトをルックアップする場合の処理に掛かる時間を短くできます。 |
| CORBA ネーミングサービスの切り替え | ルックアップの対象にする JNDI 名前空間を，InitialContext クラスのインスタンスのプリフィックス判定によって切り替えられます。 |

ネーミング管理機能の JNDI では，CORBA オブジェクトリファレンス以外のオブジェクト（RMI-IIOP のリモートオブジェクトや JDBC データソースなどのオブジェクト）を次のように扱います。

- CORBA オブジェクトリファレンス以外の登録は，対象のオブジェクトを CORBA オブジェクトに変換し，CORBA オブジェクトリファレンスを CORBA ネーミングサービスへ登録することで実現しています。
- CORBA オブジェクト以外のオブジェクトの検索は，CORBA オブジェクトリファレンスを検索し，CORBA オブジェクトから逆変換して元のオブジェクトを取得することで実現しています。

## 2.5.2 実行環境での設定（バッチサーバの設定）

ネーミング管理機能を使用する場合，バッチサーバの設定が必要です。

バッチサーバの設定は，簡易構築定義ファイルで実施します。ネーミング管理機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのネーミング管理機能の定義を次の表に示します。

### 表 2-17 簡易構築定義ファイルでのネーミング管理機能の定義

| 項目 | 指定するパラメタ | 設定内容 |
|---|---|---|
| 基本設定 | ejbserver.naming.host | CORBA ネーミングサービスのホスト名を指定します。※1 |
| | ejbserver.naming.port | CORBA ネーミングサービスのポート番号を指定します。※1 |
| ラウンドロビン検索※2 | ejbserver.jndi.namingservice.group.list | CORBA ネーミングサービスのグループを指定します。 |
| | ejbserver.jndi.namingservice.group.<Specify group name>.providerurls | 各グループに属する，CORBA ネーミングサービスのルート位置を指定します。 |
| | java.naming.factory.initial | InitialContextFactory の実装をデレゲートしているクラスを指定します。 |

| 項目 | 指定するパラメタ | 設定内容 |
|---|---|---|
| ネーミングのキャッシング | ejbserver.jndi.cache | ネーミングでのキャッシングを有効にするかどうかを指定します。 |
| | ejbserver.jndi.cache.interval | キャッシュクリアの間隔を指定します。 |
| | ejbserver.jndi.cache.interval.clear.option | キャッシュクリアの範囲を指定します。<br>キャッシュを定期的にクリアするときの設定例（物理ティアの定義の場合）を次に示します。<br>（例）<br><configuration><br><logical-server-type>j2ee-server</logical-server-type><br><param><br><param-name>ejbserver.jndi.cache</param-name><br><param-value>on</param-value><br></param><br><param><br><param-name>ejbserver.jndi.cache.interval</param-name><br><param-value>60</param-value><br></param><br><param><br><param-name>ejbserver.jndi.cache.interval.clear.option</param-name><br><param-value>check</param-value><br></param><br>：<br></configuration> |
| ネーミングサービスの通信タイムアウト | ejbserver.jndi.request.timeout | ネーミングサービスとの通信タイムアウト時間を指定します。 |

注　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。

注※1　デフォルトの設定では，バッチサーバは，ホスト名「localhost」，ポート番号「900」の CORBA ネーミングサービスをインプロセスで自動起動して使用します。

注※2　ラウンドロビン検索は，ユーザ指定名前空間機能を使用していることが前提になります。ユーザ指定名前空間機能を使用する場合，サーバ管理コマンドの動作設定のカスタマイズが必要です。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(コンテナ共通機能)」の「2.6.7　実行環境での設定」を参照してください。

## 2.6　リソース接続とトランザクション管理の概要

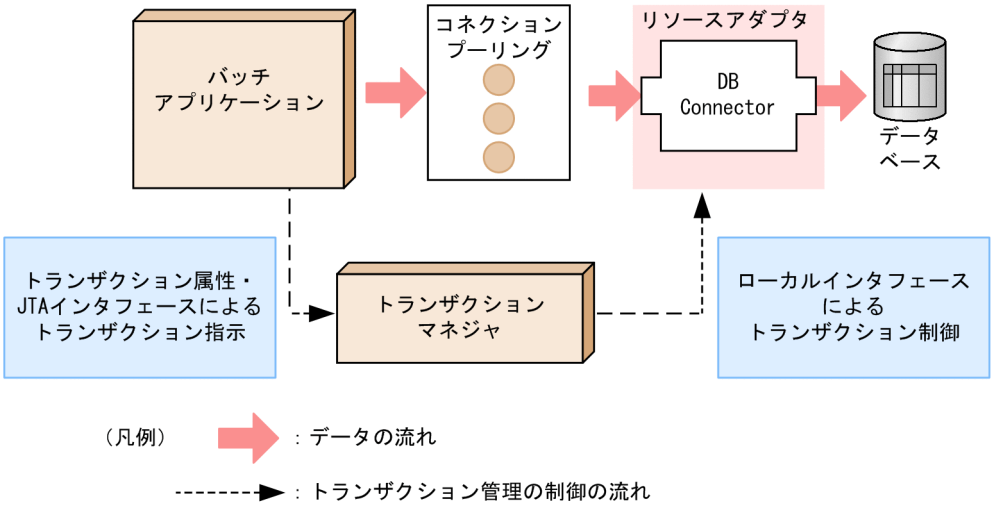
バッチアプリケーションでは，処理の延長でデータベースに接続できます。バッチアプリケーションからデータベースに接続するには，接続するリソースに対応したリソースアダプタをデプロイして使用します。アプリケーションサーバでは，データベースに接続するためのリソースアダプタである DB Connector を提供しています。

また，アプリケーションサーバでは，これらのリソースに効率的かつ信頼性の高い方法でアクセスするために，コネクションプーリングやトランザクション管理の機能を提供しています。コネクションプーリングを使用すると，リソースに対するコネクションをプーリングして，効率的にコネクションを使用できます。また，障害が発生したコネクションを適切にコネクションプールから取り除きます。また，トランザクション管理の機能を使用すると，トランザクションマネジャが，メソッドごとに指定するトランザクション属性や JTA インタフェース（UserTransaction）による指示に基づいて，リソースアクセスのトランザクションを適切に制御します。

なお，バッチアプリケーションではグローバルトランザクションは使用できません。

コネクションプーリング，およびトランザクション管理の機能を使用したリソースへの接続の例を次の図に示します。

図 2-11　コネクションプーリングおよびトランザクション管理の機能を使用したリソースへの接続の例




なお，リソースに接続するバッチアプリケーションの作成方法については，「2.3.8　バッチアプリケーションの実装（リソースに接続する場合）」を参照してください。

## 2.7　リソース接続機能

バッチアプリケーションではリソースとしてデータベースを使用できます。この節では，バッチアプリケーションからのデータベースへの接続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 2-18　この節の構成（リソース接続機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | 接続できるデータベース | 2.7.1 |
| | リソースへの接続方法 | 2.7.2 |
| | DB Connector（RAR ファイル）の種類 | 2.7.3 |
| | リソースアダプタの使用方法 | 2.7.4 |
| | リソースアダプタの設定方法 | 2.7.5 |
| | リソースアダプタの設定の流れ | 2.7.6 |
| 設定 | 実行環境での設定 | 2.7.7 |

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。

### 2.7.1　接続できるデータベース

バッチサーバからは次のデータベースに接続できます。ただし，バッチサーバではグローバルトランザクションは使用できません。

- HiRDB
- Oracle
- SQL Server※
- XDM/RD E2

注※

SQL Server と接続できるのは Windows の場合だけです。

これらのデータベースを利用するためにはリソースアダプタを使用します。リソースアダプタを利用するには，サーバ管理コマンドを使用して，リソースアダプタのプロパティ設定やインポートなどの作業が必要です。リソースアダプタの設定については，「2.7.7(2)　リソースアダプタの設定」を参照してください。

なお，リソースの設定をする前に，リソースの設定に関する注意事項を理解しておいてください。また，サーバ管理コマンドを使用する場合には，必要に応じて，サーバ管理コマンドの動作設定をカスタマイズしてください。リソース設定時の注意事項や，サーバ管理コマンドを使用するための動作設定については，

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「3.3　サーバ管理コマンドの動作設定のカスタマイズ」を参照してください。

また，次に示すデータベースとの接続に関する説明については，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　データベースへの接続」を参照してください。

- DB Connector による接続の概要
- データベースと JDBC ドライバの対応
- DB Connector がサポートする JDBC 仕様
- データベースと接続する場合の前提条件と注意事項※

注※

接続するデータベースの種類に応じて参照してください。

## 2.7.2　リソースへの接続方法

バッチアプリケーションからデータベースに接続するには，JDBC ドライバを直接使用するか，またはアプリケーションサーバで提供しているリソースアダプタを使用します。リソースアダプタを使用する場合は，DB Connector を使用します。バッチアプリケーションからデータベースに接続するときに使用できる機能を，接続方法ごとに次の表に示します。なお，DB Connector を使用すると次の表の機能に加えて，DB Connector が提供している機能も使用できます。DB Connector が提供している機能については，「2.7.4(1)　リソースアダプタの機能」を参照してください。

### 表 2-19　データベースに接続するときに使用できる機能

| 使用できる機能 | | | 接続方法 | |
|---|---|---|---|---|
| | | | DB Connector | JDBC ドライバ |
| SQL の実行 | | | ○ | ○ |
| トランザクションの利用 | Connection API によるトランザクション | | ○ | ○ |
| | JTA | 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 | × | × |
| GC 制御機能 | | | ○ | × |

（凡例）○：使用できる　×：使用できない

## 2.7.3　DB Connector（RAR ファイル）の種類

DB Connector を使用してデータベースに接続する場合，使用する JDBC ドライバに応じた RAR ファイルを使用します。RAR ファイルは，サーバ管理コマンドを使用して操作します。サーバ管理コマンドを使

用して RAR ファイルを操作する方法については，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. リソースアダプタの設定」を参照してください。

JDBC ドライバの種類とバッチサーバの場合に使用できる RAR ファイルについて次の表に示します。

表 2-20 JDBC ドライバと RAR ファイルの対応

| JDBC ドライバ | RAR ファイル | 説明 |
|---|---|---|
| HiRDB Type4 JDBC Driver | DBConnector_HiRDB_Type4_CP.rar | HiRDB，XDM/RD E2 への接続に使用する RAR ファイルです。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。 |
| Oracle JDBC Thin Driver | DBConnector_Oracle_CP.rar | Oracle への接続に使用する RAR ファイルです。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。 |
| SQL Server の JDBC ドライバ | DBConnector_SQLServer_CP.rar | SQL Server（Windows の場合だけ）への接続に使用する RAR ファイルです。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。 |

注 新規に，DB Connector の RAR ファイルを使用する場合，アプリケーションサーバで提供する Connector 属性ファイルのテンプレートファイルを使用して，プロパティを定義できます。Connector 属性ファイルのテンプレートファイルは，すべての DB Connector の RAR ファイルに対して提供しています。提供しているテンプレート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「4.1.13 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。

## 2.7.4 リソースアダプタの使用方法

リソースアダプタを使用してリソースと接続する場合，リソースアダプタを J2EE リソースアダプタとしてデプロイしてください。J2EE リソースアダプタとは，J2EE サーバ上に配置されたリソースアダプタです。デプロイ方法については，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「3.3.7 リソースアダプタの設定方法」を参照してください。

### (1) リソースアダプタの機能

バッチサーバの場合にデータベース接続で使用できる機能を次の表に示します。それぞれの機能の詳細については，参照先の説明を参照してください。

表 2-21 リソースアダプタの機能

| 機能 | 項目 | 説明 | 参照先マニュアル※1 | 参照個所 |
|---|---|---|---|---|
| パフォーマンスチューニングのための機能 | コネクションプーリング | コネクションをメモリ上にプールしておくことで，アプリケーションからの接続要求に対して高速に処理できます。 | 基本・開発編(コンテナ共通機能) | 3.14.1 |

| 機能 | 項目 | 説明 | 参照先マニュアル※1 | 参照個所 |
|---|---|---|---|---|
| | コネクションプールのウォーミングアップ | サーバ起動時またはリソースアダプタ起動時に，指定した数のコネクションを作成します。コネクションをプールしておくことで，コネクションプール開始直後の接続要求を高速に処理します。 | | 3.14.2 |
| | コネクション数調節機能 | 一定間隔で，プール内の不要なコネクションを段階的に減少させます。 | | 3.14.2 |
| | コネクションシェアリング・アソシエーション | コネクションを共有することで，コネクション取得処理に掛かる処理時間を短縮できます。<br>コネクションシェアリングでは，論理コネクションと接続先リソースのコネクションである物理コネクションを多対 1 で接続します。<br>バッチアプリケーションの場合，コネクションアソシエーションは使用できません。 | | 3.14.3 |
| | ステートメントプーリング | PreparedStatement および CallableStatement を使用する処理の場合に，これらのステートメントをプーリングしておくことで，同じステートメント作成に掛かる処理時間を短縮できます。 | | 3.14.4 |
| | DataSource オブジェクトのキャッシング | JNDI インタフェースを使用して，DataSource 型のオブジェクトの検索要求をする場合，DataSource オブジェクトをキャッシングできます。 | | 3.14.7 |
| | DB Connector のコンテナ管理でのサインオンの最適化 | コンテナ管理でサインオンをする場合，サインオンの動作を最適化できます。 | | 3.14.8 |
| フォールトトレランスのための機能 | コネクション障害検知 | プーリングされているコネクションにトラブルが発生していないかを検知できます。これによって，ユーザプログラムからのコネクション要求に対して，有効なコネクションだけを返却できます。 | | 3.15.1 |
| | コネクション枯渇時のコネクション取得待ち | コネクションプールに指定した最大数までコネクションがプールされていて，利用できるコネクションがない場合は，コネクション取得要求を待ち状態にできます。 | | 3.15.2 |
| | コネクション取得リトライ | 使用できるコネクションがコネクションプールにない場合や接続先リソースの物理コネクションの確立に失敗した場合に，自動的にコネクションの取得処理を再実行できます。 | | 3.15.3 |
| | コネクションプールの情報表示 | コマンドを使用して，コネクションプール内のコネクションの情報を表示できます。 | | 3.15.4 |
| | コネクションプールのクリア | データベースサーバにトラブルが発生してコネクションが切断された場合などに，不要なコネクションプールをコマンドで削除できます。 | | 3.15.5 |

| 機能 | 項目 | 説明 | 参照先マニュアル※1 | 参照個所 |
|---|---|---|---|---|
| | ステートメントキャンセル | 実行中の SQL 処理が返ってこない状態でトランザクションタイムアウトが発生した場合に，ステートメントをキャンセルできます。 | | 3.15.8 |
| | 障害調査用 SQL の出力 | デッドロックやスローダウンなどの障害が発生した場合に，発行した SQL をログに出力できます。ログは，障害要因の解析に利用できます。 | | 3.15.10 |
| | オブジェクトの自動クローズ | ユーザプログラムでオープンした Statement オブジェクトなどをクローズできなかった場合に，DB Connector によってオブジェクトを自動的にクローズできます。 | | 3.15.11 |
| リソースへの接続テスト | リソースへの接続テスト | 環境構築時に，リソースアダプタが正しく設定できているかどうかを確認できます。 | | 3.17 |
| 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） | J2EE リソースへの別名付与※2 | J2EE リソースに別名を付与できます。バッチアプリケーションからは，別名として設定した任意の名称でルックアップできます。 | | 2.6 |
| 性能解析トレースを使用したシステムの性能解析 | コネクション ID の PRF トレース | 各機能が出力する性能解析情報を収集します。この情報を基に，システム性能およびボトルネックを分析できます。 | 保守／移行編 | 8 章 |

注※1　参照先のマニュアル名称は，「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　バッチサーバの場合，リソースアダプタの別名は必ず使用します。

リソースアダプタの種類ごとに使用できる機能を次の表に示します。

## 表 2-22　リソースアダプタの種類ごとに使用できる機能

| 機能 | 項目 | リソースアダプタの種類 |
|---|---|---|
| | | DB Connector |
| パフォーマンスチューニングのための機能 | 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 | ○ |
| | コネクション数調節機能 | ○ |
| | 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 | ○ |

| 機能 | 項目 | リソースアダプタの種類 |
|---|---|---|
| | | DB Connector |
| フォールトトレランスのための機能 | コネクション障害検知 | ○ |
| | 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 | ○ |
| | コネクションプールの情報表示 | ○ |
| | コネクションプールのクリア | ○ |
| | ステートメントキャンセル | ○ |
| | 障害調査用 SQL の出力 | ◎ |
| | オブジェクトの自動クローズ | ○ |
| リソースへの接続テスト | リソースへの接続テスト | ○ |
| 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） | J2EE リソースへの別名付与※ | ○ |
| 性能解析トレースを使用したシステムの性能解析 | コネクション ID の PRF トレース | ○ |

（凡例）

◎：必ず有効になる

○：使用できる

注※　バッチアプリケーションの場合，コネクションアソシエーションは使用できません。

## (2)　リソースアダプタ以外の機能

ここでは，リソースアダプタ以外で実現される機能について説明します。ここで説明する機能は，リソースアダプタの種類に関係なく使用できます。

リソースアダプタ以外で実現される機能を，次の表に示します。それぞれの機能の詳細については，参照先の説明を参照してください。

### 表 2-23　リソースアダプタ以外の機能

| 機能 | 項目 | 説明 | 参照先マニュアル※ | 参照個所 |
|---|---|---|---|---|
| パフォーマンスチューニングのための機能 | ライトトランザクション | ローカルトランザクションに最適化された環境を提供します。ライトトランザクション機能は必ず有効にします。 | 基本・開発編(コンテナ共通機能) | 3.14 |
| フォールトトレランスのための機能 | トランザクションタイムアウト | トランザクション開始時間から一定時間経過した時点で呼び出し先のトランザクションをロールバックします。 | 基本・開発編(コンテナ共通機能) | 3.15 |

注※　参照先のマニュアル名称は，「アプリケーションサーバ 機能解説」を省略しています。

**注意事項**

J2EE サーバのトランザクション管理の機能では，トランザクションを自動決着する機能がありますが，バッチサーバの場合，トランザクションの自動決着機能は使用できません。

### (3)　リソースアダプタのオプショナル名についての注意事項

同じオプショナル名で複数のリソースアダプタをデプロイしている場合，エラーメッセージが出力されて，リソースアダプタの開始に失敗します。

## 2.7.5　リソースアダプタの設定方法

バッチアプリケーションからデータベースに接続するには，DB Connector というリソースアダプタを使用します。ここでは，バッチサーバで使用するリソースアダプタの設定について説明します。バッチサーバの場合，リソースアダプタは，J2EE リソースアダプタとしてデプロイして使用します。

**参考**

リソースアダプタが DB Connector の場合，アプリケーションサーバが提供する Connector 属性ファイルのテンプレートファイルが使用できます。Connector 属性ファイル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。このため，編集対象の Connector 属性ファイルをサーバ管理コマンド（cjgetrarprop コマンドまたは cjgetresprop コマンド）で取得する操作が不要になります。
Connector 属性ファイルのテンプレートは，次の場所に格納されています。テンプレートファイルはコピーして使用してください。

- 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/

なお，Connector 属性ファイルのテンプレートファイル，およびテンプレートファイル使用時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 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。

**注意事項**

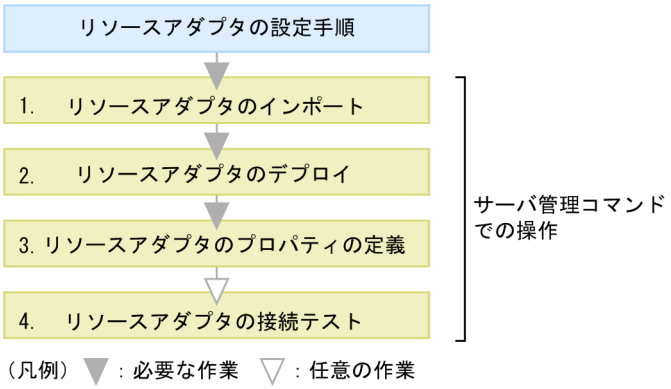
旧バージョンのアプリケーションサーバで使用していたリソースアダプタを使用する場合，リソースアダプタの移行処理が必要です。リソースの移行方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「10.8　リソースアダプタの移行」を参照してください。

## 2.7.6　リソースアダプタの設定の流れ

リソースアダプタの設定には，サーバ管理コマンドを使用します。バッチサーバの場合，リソースアダプタは，J2EE リソースアダプタとしてデプロイして使用します。

バッチサーバで使用するリソースアダプタの新規設定の流れを次の図に示します。

図 2-12　バッチサーバで使用するリソースアダプタの新規設定の流れ




図中の 1.～4.について説明します。

1. **サーバ管理コマンドを使用して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，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，「2.7.3　DB Connector（RAR ファイル）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，リソースアダプタをデプロイします。

   リソースアダプタは，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。**J2EE リソースアダプタ**とは，バッチサーバに共有スタンドアロンモジュールとして配備したリソースアダプタのことです。サーバ管理コマンドでインポートしたリソースアダプタをデプロイすると，バッチサーバ上で使用できるようになります。

3. **サーバ管理コマンドを使用して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。

バッチサーバの場合，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設定してください。ユーザ指定名前空間機能を使用した別名の設定は，リソースアダプタのプロパティで定義します。ユーザ指定名前空間機能の設定については，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(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。

リソースアダプタのプロパティ定義で設定できる内容については，「2.7.7(2)　リソースアダプタの設定」を参照してください。

4. **サーバ管理コマンドを使用して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，リソースアダプタの接続テストを実施します。リソースごと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。

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，cjimportres 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「cjimportres（リソースのインポート)」を参照してください。cjdeployrar 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「cjdeployrar（リソースアダプタのデプロイ)」を参照してください。cjgetrarprop 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「cjgetrarprop（RAR ファイルの属性の取得)」を参照してください。cjtestres 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「cjtestres（リソースの接続テスト)」を参照してください。属性については，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。

なお，次の手順については，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(コンテナ共通機能)」の「3.3.8　リソースアダプタの設定の流れ（J2EE リソースアダプタとしてデプロイして使用する場合)」を参照してください。その際，「J2EE サーバ」を「バッチサーバ」に，「J2EE アプリケーション」を「バッチアプリケーション」に置き換えてお読みください。

- リソースアダプタの設定を変更する場合の流れ
- リソースアダプタを入れ替える場合の流れ

> **参考**
>
> 次のような場合，リソースアダプタをエクスポート・インポートすることで，効率良くリソースアダプタを設定できます。
>
> - 開発環境で設定したリソースアダプタをエクスポートして，運用環境にインポートして使用する場合
> - 運用環境ですでに動いているリソースアダプタをエクスポートして，増設したバッチサーバにインポートして使用する場合
>
> エクスポートとインポートは cjexportrar と cjimportres で実行します。

なお，アプリケーションサーバのバージョンやプラットフォームが異なるホスト間では，リソースアダプタをエクスポート・インポートして使用することはできません。リソースアダプタをエクスポートするホストと，アプリケーションサーバのバージョンやプラットフォームが異なるホストでリソースアダプタを設定する場合は，リソースアダプタを新規に設定してください。

## 2.7.7　実行環境での設定

リソース接続機能を使用する場合，バッチサーバおよびリソースアダプタの設定が必要です。

ここでは，リソース接続機能を使用するための設定について説明します。

### (1)　バッチサーバの設定

バッチサーバの設定は，簡易構築定義ファイルで実施します。バッチアプリケーション実行機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリソース接続機能の定義を次の表に示します。

表 2-24　簡易構築定義ファイルでのリソース接続機能の定義

| 項目 | 指定するパラメタ | 設定内容 |
|---|---|---|
| アプリケーションサーバが管理するトランザクションの外でのコネクションシェアリングの有効化 | ejbserver.connectionpool.sharingOutsideTransactionScope.enabled | アプリケーションサーバが管理するトランザクションの外で複数回コネクションの取得を行ったときの，コネクションシェアリングの動作を指定します。 |
| DataSource オブジェクトのキャッシング | ejbserver.jndi.cache.reference | DataSource オブジェクトのキャッシングを有効にするかどうかを指定します。 |
| DB Connector のコンテナ管理でのサインオンの最適化 | ejbserver.connectionpool.applicationAuthentication.disabled | コンテナ管理のサインオンの最適化機能を有効にするかどうかを指定します。 |

注　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)　リソースアダプタの設定

リソースに接続するバッチアプリケーションの場合，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設定します。バッチアプリケーションからリソースアダプタをルックアップするには，ユーザ指定名前空間機能で設定した別名を使用します。ユーザ指定名前空間機能の設定については，「2.3.8　バッチアプリケーションの実装（リソースに接続する場合）」を参照してください。

**参考**

リソースの設定をする前に，リソースの設定に関する注意事項を理解しておいてください。また，サーバ管理コマンドを使用する場合には，必要に応じて，サーバ管理コマンドの動作設定をカスタマイズしてください。リソース設定時の注意事項や，サーバ管理コマンドを使用するための動作設定については，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「3.3 サーバ管理コマンドの動作設定のカスタマイズ」を参照してください。

リソースアダプタの設定は，Connector 属性ファイルで実施します。

Connector 属性ファイルでのリソース接続機能の定義を次の表に示します。

### 表 2-25　Connector 属性ファイルでのリソース接続機能の定義

| 分類 | 項目 | 設定内容 |
|---|---|---|
| 一般情報 | トランザクションサポートレベル | <transaction-support>タグで，トランザクションサポートレベルを設定します。トランザクション管理なし（NoTransaction），またはローカルトランザクション（LocalTransaction）を指定します。バッチサーバの場合，グローバルトランザクション（XATransaction）は指定できません。 |
| コンフィグレーションプロパティ | データベースコネクション確立までの待ち時間 | <config-property>タグの loginTimeout で，データベースコネクション確立までのバッチアプリケーションの待ち時間を指定します。 |
| | ステートメントキャンセル | <config-property>タグの CancelStatement で，トランザクションタイムアウト発生時のステートメントキャンセルを有効にするかどうかを指定します。 |
| | PreparedStatement のプールサイズ※1 | <config-property>タグの PreparedStatementPoolSize で，PreparedStatement のプールサイズを指定します。 |
| | CallableStatement のプールサイズ※1 | <config-property>タグの CallableStatementPoolSize で，CallableStatement のプールサイズを指定します。 |
| 実行時プロパティ | コネクションの最小値と最大値 | <property>タグの MinPoolSize と MaxPoolSize で，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と最大値を指定します。 |
| | コネクションの障害検知 | <property>タグの ValidationType でコネクションの障害を検知するタイミングを指定し，ValidationInterval で障害を検知する間隔を指定します。<br>なお，コネクションの障害検知にタイムアウトを設定する場合には，NetworkFailureTimeout でコネクション管理スレッドの使用を有効にします。※2 |
| | コネクションの取得リトライ | <property>タグの RetryCount でコネクション取得に失敗した場合のリトライ回数を指定し，RetryInterval でリトライ間隔を指定します。 |

| 分類 | 項目 | 設定内容 |
|---|---|---|
| | コネクションスイーパ | <property>タグの SweeperInterval でコネクションの自動破棄（コネクションスイーパ）が動作する間隔を指定し，ConnectionTimeout で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するかどうかを判定するまでの時間を指定します。 |
| | コネクション枯渇時のコネクション取得待ち | <property>タグの RequestQueueEnable でコネクション枯渇時のコネクション取得待ちを有効にするかどうかを指定し，RequestQueueTimeout で待ち時間を指定します。 |
| | 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ウォーミングアップ機能を使用する場合，<property>タグで，Warmup を指定します。 |
| | コネクション管理スレッド | コネクション管理スレッドを使用する場合，<property>タグで，NetworkFailureTimeout を指定します。<br>コネクション管理スレッドを使用する場合，コネクションの障害検知機能，およびコネクション数調節機能のタイムアウトを使用する設定になります。 |
| | コネクション数調節機能 | <property>タグの ConnectionPoolAdjustmentInterval で，コネクション数調節機能が動作する間隔を指定します。<br>なお，コネクション数調節機能にタイムアウトを設定する場合には，NetworkFailureTimeout でコネクション管理スレッドの使用を有効にします。※2 |

注　Connector 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※1　XDM/RD E2 11-01 以前のバージョンの場合，ステートメントプーリング機能を利用できないため，これらのプロパティには 0 を指定してください。

注※2　同じキーで設定します。このため，コネクションの障害検知機能でタイムアウトを使用する場合は，コネクション数調節機能でもタイムアウトを使用する設定となります。なお，タイムアウト時間は簡易構築定義ファイルの J2EE サーバで指定するキー（ejbserver.connectionpool.validation.timeout）に，任意の時間を指定してください（デフォルト値は 5 秒）。

なお，DB Connector を使用してデータベースに接続する場合に設定する，DB Connector のプロパティ定義については，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「4.1.2 設定する項目と操作の概要」を参照してください。

## 2.8　トランザクション管理

この節では，リソース接続時のトランザクション管理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 2-26　この節の構成（トランザクション管理）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | リソース接続時のトランザクション管理の概要 | 2.8.1 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.8.2 |

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。

### 2.8.1　リソース接続時のトランザクション管理の概要

リソース接続時のトランザクションを管理する方法には，アプリケーションサーバで管理する方法と，アプリケーションサーバが管理しないでユーザが直接管理する方法があります。データベースへの接続では，アプリケーションサーバのトランザクションマネジャを使用してトランザクションを管理できます。トランザクション管理については，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「3.4.1　リソース接続でのトランザクション管理の方法」を参照してください。

バッチサーバで使用できるトランザクションは，ローカルトランザクションです。グローバルトランザクションは使用できません。また，バッチサーバでは，必ずライトトランザクション機能を有効にしてください。ライトトランザクション機能とは，ローカルトランザクションに最適化された環境を提供する機能です。ローカルトランザクションおよびライトトランザクション機能については，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「3.4.2　ローカルトランザクションとグローバルトランザクション」を参照してください。

また，EJB 呼び出し時に，呼び出し先でシステム例外が発生したときの，呼び出し元，呼び出し先のトランザクションは，それぞれ次のように動作します。

**呼び出し元のトランザクション**

トランザクションはロールバックにマークされません。

**呼び出し先のトランザクション**

トランザクションはコンテナによってロールバックされます。この動作は，EJB 仕様で規定されています。

### 2.8.2　実行環境での設定（バッチサーバの設定）

トランザクション管理の機能を使用する場合，バッチサーバの設定が必要です。

バッチサーバの設定は，簡易構築定義ファイルで実施します。トランザクション管理の機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。指定するパラメタを次に示します。

- ejbserver.jta.TransactionManager.defaultTimeOut
  バッチサーバ上で開始されるトランザクションのタイムアウトのデフォルト値を指定します。

なお，トランザクションタイムアウトについては，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　フォールトトレランスのための機能」を参照してください。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。

## 2.9　GC 制御機能

バッチサーバでは，GC 制御機能を使用できます。この節では GC 制御機能の概要，処理の流れおよび設定方法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 2-27　この節の構成（GC 制御機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | GC 制御機能の概要 | 2.9.1 |
| | GC 制御の処理の流れ | 2.9.2 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.9.3 |

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。

### 2.9.1　GC 制御機能の概要

GC とは，プログラムが使用し終わったメモリ領域を自動的に回収して，ほかのプログラムが利用できるようにするための技術です。GC は JavaVM が実行します。

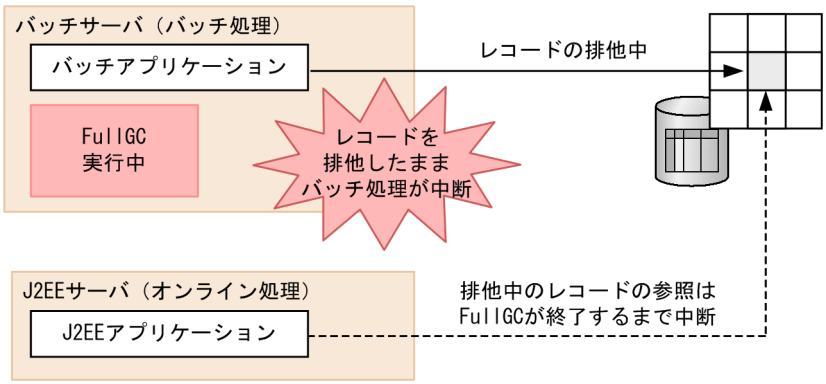
GC には処理時間が掛かります。また，GC 実行中は，JavaVM 上のすべてのプログラム処理が中断するため，GC を適切に実行できるかどうかが，システムの処理性能に大きく影響します。

バッチサーバでは，バッチアプリケーションが長時間リソースを排他するのを回避するため，**GC 制御機能**を提供しています。GC 制御機能とは，リソースが排他されていないときに明示的に FullGC を実行するための機能です。GC 制御機能の利用によって，リソースの排他中に FullGC が発生するのを回避できます。

GC 制御機能について，例を使用して説明します。

GC 制御機能を使用していない場合，バッチ処理と並行してオンライン処理を実行する環境では，次の図に示す問題があります。

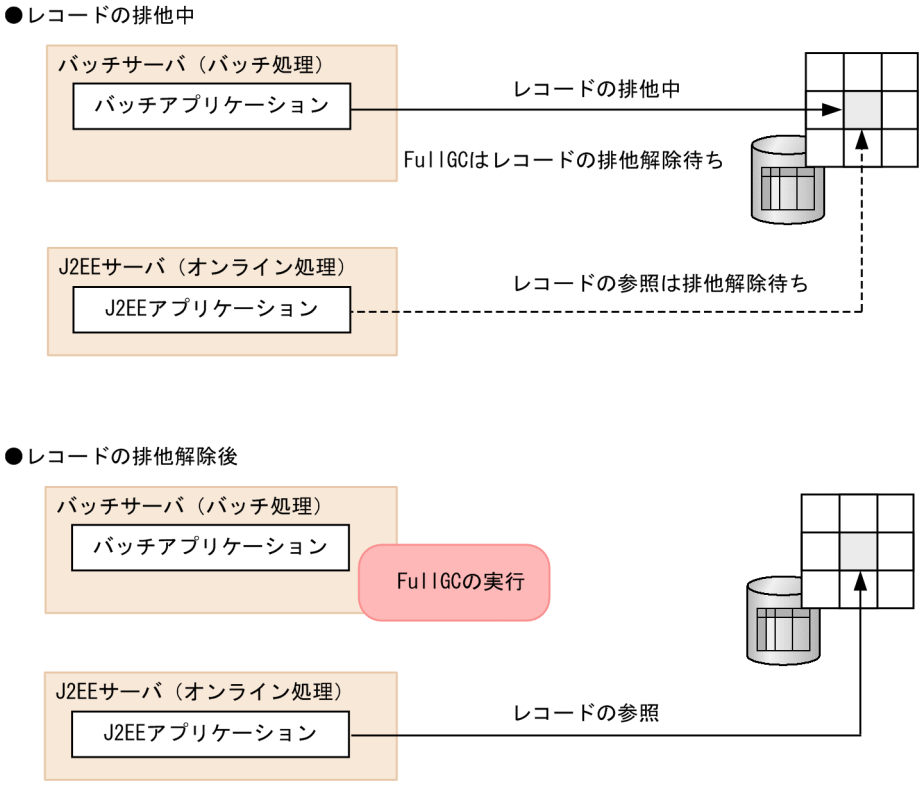
図 2-13　GC 制御機能を使用していない場合




この図では，バッチアプリケーションでのリソース排他中に，FullGC が発生しています。これによって，バッチアプリケーションはリソースを排他したまま処理が中断します。また，この間にオンライン処理から排他中のレコードが参照されると，オンライン処理もバッチサーバの FullGC が終了するまで中断します。

GC 制御機能を使用すると次の図のようになります。

図 2-14　GC 制御機能を使用している場合



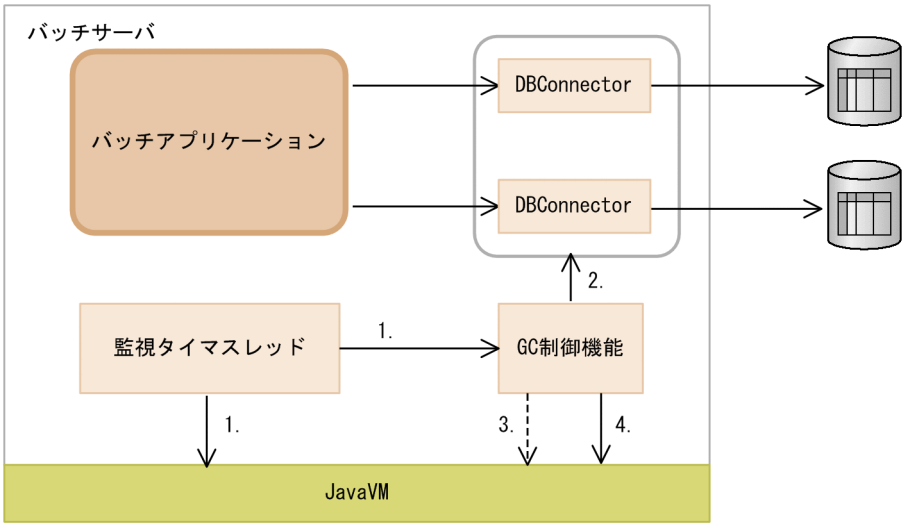

図のように，FullGC の実行要求が出たときに，バッチアプリケーションでリソースを排他していると，FullGC の実行は待ち状態になります。

レコードの排他が解除されると，バッチサーバで FullGC が実行されます。また，オンライン処理もリソースへのアクセスができるようになります。これによって，バッチアプリケーションでの長時間のリソース排他を回避できます。

## 2.9.2　GC 制御の処理の流れ

GC 制御は次の流れで処理されます。

図 2–15　GC 制御の処理の流れ




1. メモリ監視

   監視タイマスレッドは，JavaVM のメモリを監視します。(1)に示す条件を満たすと，GC 制御機能に GC 実行要求が出されます。

2. リソース排他のチェック

   GC 実行要求が出されると，GC 制御機能はリソース排他中かどうかを調査します。

3. FullGC の実行待ち

   リソース排他中の場合，FullGC の実行は待ち状態になります。

4. FullGC の実行

   リソースの排他が解除されると，FullGC が実行されます。

それぞれの処理について説明します。

### (1)　メモリ監視

監視タイマスレッドは JavaVM のメモリを監視し，次のどれかの条件を満たす場合，GC 制御機能に GC の実行要求を出します。

SerialGC が有効な場合

- Tenured 領域消費サイズ/Tenured 領域合計サイズ×100≧GC 制御のしきい値
- New 領域合計サイズ/Tenured 領域最大空きサイズ×100≧GC 制御のしきい値
- Metaspace 領域消費サイズ/Metaspace 領域最大サイズ×100≧GC 制御のしきい値

G1GC が有効な場合

- Java ヒープ領域消費サイズ/Java ヒープ領域合計サイズ×100≧GC 制御のしきい値
- Metaspace 領域消費サイズ/Metaspace 領域最大サイズ×100≧GC 制御のしきい値

## (2) リソース排他のチェック

GC 実行が要求されると，GC 制御機能は，バッチアプリケーションが使用しているコネクションを調査します。コネクションの調査では，バッチアプリケーションがリソース排他中かどうかを確認します。

次の表にリソースの排他中と見なす状態を示します。

### 表 2-28 リソースの排他中と見なす状態

| トランザクション | 状態 | | DB Connector | JDBC |
|---|---|---|---|---|
| トランザクション外 | SQL 文を実行中※1 | • java.sql.Statement#execute の実行中<br>• java.sql.Statement#executeUpdate の実行中<br>• java.sql.Statement#executeQuery の実行中<br>• java.sql.Statement#executeBatch の実行中 | ○ | × |
| | ResultSet に対する操作中 | • java.sql.ResultSet#deleteRow の実行中<br>• java.sql.ResultSet#insertRow の実行中<br>• java.sql.ResultSet#updateRow の実行中 | ○ | × |
| | オブジェクト取得などの操作中※1 | • java.sql.Statement#addBatch の実行中<br>• java.sql.Connection#prepareCall の実行中<br>• java.sql.Connection#prepareStatement の実行中 | ○ | × |
| トランザクション中 | • Connection API によるトランザクション実行中※2<br>• ローカルトランザクション（JTA）実行中※2 | | ○ | × |
| | グローバルトランザクション（JTA）実行中 | | − | − |

（凡例）○：リソース排他中として扱う　×：リソース排他がないものとして扱う
　　−：該当しない

注※1　表中の java.sql.Statement は，サブインタフェースである java.sql.PreparedStatement，java.sql.CallableStatement を含みます。

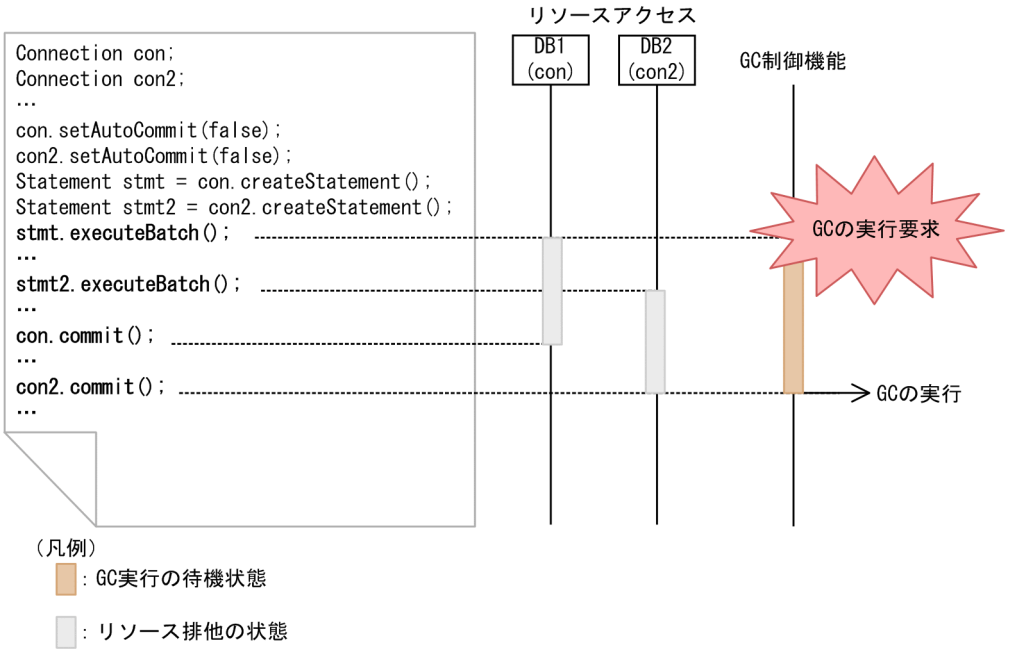
注※2　トランザクションの開始後（setAutoCommit(false)や UserTransaction.begin の実行後），SQL 文の実行または ResultSet に対する操作を 1 回以上実行していて，トランザクションの決着処理が完了していない状態を指します。

JDBC を使用したリソース操作については，リソース排他がないものとして扱われます。例えば，JDBC の SQL 文の実行と，DB Connector でのトランザクション処理が混在するプログラムを実行した場合，DB Connector のトランザクション処理だけが GC 制御の対象となります。

## (3) FullGC の実行待ち

リソース排他中と判断されると，メッセージ KDJE55024-I を出力して，FullGC の実行待ち状態になります。リソース排他が一つでもあると，FullGC は待機し続けます。FullGC 実行待ちの例を次の図に示します。

図 2-16 FullGC 実行待ちの例




この図では，一つのジョブプログラム中で二つのリソースにアクセスしています。リソース排他中に FullGC の実行が要求されると，GC 制御機能は FullGC の実行を待機状態にします。二つのリソースアクセスが終了する con2.commit()が実行されると，排他が解除されます。

## (4) GC の実行

リソース排他がなくなると，FullGC が実行されます。

## (5) 注意事項

- 同時に実行できるバッチアプリケーションは一つだけです。
- 一つのバッチアプリケーションから複数のリソースへの処理ができます。ただし，グローバルトランザクションは使用できません。
- FullGC の実行待ち状態でも，空きメモリが不足すると，JavaVM によって FullGC が実行されることがあります。これは，GC 実行時のメモリ使用量のしきい値が大きい場合や，リソースの排他区間が長い場合などに発生します。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「9.4 GC 制御で使用するしきい値を設定する」を参照して，メモリ使用量のしきい値をチューニングしてください。

### 2.9.3　実行環境での設定（バッチサーバの設定）

GC 制御機能を使用する場合，バッチサーバの設定が必要です。

バッチサーバの設定は，簡易構築定義ファイルで実施します。GC 制御機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。指定するパラメタを次に示します。

- ejbserver.batch.gc.watch.threshold
  GC を実行する条件となるメモリ使用量のしきい値を指定します。

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。

## 2.10　コンテナ拡張ライブラリ

バッチサーバでは，アプリケーション間で共通に利用したい処理がある場合に，ユーザ作成のライブラリを利用できます。ユーザ作成のライブラリを利用することで，アプリケーションの機能を拡張できます。この節では，コンテナ拡張ライブラリの概要および設定方法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 2-29　この節の構成（コンテナ拡張ライブラリ）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | コンテナ拡張ライブラリの概要 | 2.10.1 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.10.2 |

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。

### 2.10.1　コンテナ拡張ライブラリの概要

アプリケーションが共通に利用できるライブラリを**コンテナ拡張ライブラリ**といいます。このライブラリを利用することで，アプリケーション間で共通して，ユーザ作成のライブラリを呼び出せるようになります。コンテナ拡張ライブラリに設定したライブラリはシステムクラスローダでローディングされます。詳細については，「2.3.1　バッチアプリケーション実行機能の概要」を参照してください。

バッチサーバではコンテナ拡張ライブラリを利用できますが，バッチアプリケーション自体をコンテナ拡張ライブラリに設定して使用することはできません。

また，**サーバ起動・停止フック機能**を利用することで，サーバの起動，終了時にコンテナ拡張ライブラリが呼び出されるようにできます。また，コンテナ拡張ライブラリで使用する JNI 機能の初期化などを行うことができます。

コンテナ拡張ライブラリを使用するためには，ライブラリを一つの JAR ファイルにまとめ，コンテナ拡張ライブラリを使用するための設定を usrconf.cfg で定義します。また，コンテナ拡張ライブラリが JNI を利用する場合は，サーバ起動・停止フック機能を使用するための設定も必要です。

なお，コンテナ拡張ライブラリの利用の概要については，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「19.2　コンテナ拡張ライブラリの利用」を参照してください。サーバ起動・停止フック機能の実装方法については，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「19.4.2　サーバ起動・停止フック機能の実装方法」を参照してください。

**注意事項**

コンテナ拡張ライブラリには，次のアクセス権が付与されます。アクセス権は変更できません。

java.security.AllPermission

ただし，java.lang.RuntimePermission の setSecurityManager アクセス権は付与されません。

## 2.10.2　実行環境での設定（バッチサーバの設定）

コンテナ拡張ライブラリの機能を使用する場合，バッチサーバの設定が必要です。

バッチサーバの設定は，簡易構築定義ファイルで実施します。コンテナ拡張ライブラリの機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。指定するパラメタを次に示します。

- add.class.path
  コンテナ拡張ライブラリの JAR ファイルのパスを指定します。
- add.library.path
  JNI 用ライブラリの検索パスを指定します。

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。

コンテナ拡張ライブラリの機能を使用するため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(コンテナ共通機能)」の「19.3.3　コンテナ拡張ライブラリの機能を使用するための設定」を参照してください。

## 2.11　JavaVM の機能

この節では，JavaVM の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 2-30　この節の構成（JavaVM の機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | JavaVM の機能の概要 | 2.11.1 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.11.2 |

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。

### 2.11.1　JavaVM の機能の概要

アプリケーションサーバで動作するバッチサーバのプロセスは，JavaVM 上で実行されます。

JavaVM は，構成ソフトウェアである Developer's Kit for Java によって提供される，独自の JavaVM です。JavaVM の機能を次の表に示します。それぞれの機能の詳細については，参照先の説明を参照してください。

表 2-31　JavaVM の機能

| 機能 | 説明 | 参照先マニュアル※ | 参照個所 |
|---|---|---|---|
| 明示管理ヒープ機能 | FullGC 発生の要因になる Java オブジェクトを Explicit ヒープ領域に配置できます。アプリケーションで使用する Java オブジェクトによる，FullGC の発生を抑止できます。 | このマニュアル | 7 章 |
| クラス別統計機能 | 各クラスのインスタンスが持つメンバの配下にあるすべてインスタンスのサイズを，クラス別統計情報として拡張スレッドダンプに出力できます。クラス別統計情報を複数回出力すると，GC による Java オブジェクトの変化や，寿命が短い Java オブジェクトの状態などを調査できます。クラス別統計情報を出力する機能としては，インスタンス統計機能，STATIC メンバ統計機能，参照関係情報出力機能，統計前の GC 選択機能，Tenured 領域内不要オブジェクト統計機能，および Tenured 増加要因の基点オブジェクトリスト出力機能があります。 | 保守／移行編 | 9.3 |
| クラス別統計情報解析機能 | 拡張スレッドダンプに出力したクラス別統計情報を基に，クラスごとのインスタンスの合計サイズ，およびクラスごとのインスタンス数を 2 種類の CSV ファイルとして出力できます。 | | 9.10 |

| 機能 | 説明 | 参照先マニュアル※ | 参照個所 |
|---|---|---|---|
| Survivor 領域の年齢分布情報出力機能 | CopyGC 実行時に，JavaVM ログファイルに Survivor 領域の Java オブジェクトの年齢分布を出力できます。Survivor 領域の使用状況が調査でき，メモリサイズのチューニングに使用できます。 | | 9.11 |
| hndlwrap 機能 | ログオフ時の JavaVM のログオフイベントの発生を抑止できます。 | | 9.12 |

注※　参照先のマニュアル名称は，「アプリケーションサーバ 機能解説」を省略しています。

また，JavaVM では，障害発生時の要因分析やシステムの状態確認に利用できるよう，ログの出力内容が拡張されています。このログは，JavaVM ログファイルに出力され，標準の JavaVM よりも，多くのトラブルシュート情報が取得できます。さらに，このログ（拡張 verbosegc 情報）を利用して適切なチューニングを実施することで，システムの可用性向上が図れます。JavaVM ログ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 機能解説 保守／移行編」の「4.10　JavaVM ログ（JavaVM ログファイル）」を参照してください。JavaVM のチューニングについては，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.　JavaVM のメモリチューニング」を参照してください。

## 2.11.2　実行環境での設定（バッチサーバの設定）

JavaVM の機能を使用する場合，バッチサーバの設定が必要です。

バッチサーバの設定は，簡易構築定義ファイルで実施します。JavaVM の機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの JavaVM の機能の定義を次の表に示します。

### 表 2-32　簡易構築定義ファイルでの JavaVM の機能の定義

| 項目 | 指定するパラメタ | | 設定内容 |
|---|---|---|---|
| | パラメタ名 | パラメタ値 | |
| 明示管理ヒープ機能の利用 | add.jvm.arg | -XX:+HitachiUseExplicitMemory | バッチアプリケーションで明示管理ヒープ機能を実装している場合に，明示管理ヒープ機能の使用を指定します。<br>明示管理ヒープ機能使用時に設定できる JavaVM オプションについては，「7.13.1　明示管理ヒープ機能を利用するための共通の設定（JavaVM オプションの設定）」を参照してください。 |
| Suvivor 領域の年齢分布情報の出力 | add.jvm.arg | -XX:+HitachiVerboseGCPrintTenuringDistribution | CopyGC 発生時に，Suvivor 領域のオブジェクトの年齢分布情報を JavaVM ログファイルに出力することを指定します。 |

| 項目 | 指定するパラメタ | | 設定内容 |
|---|---|---|---|
| | パラメタ名 | パラメタ値 | |
| JavaVM のログの取得（JavaVM ログファイル） | add.jvm.arg | -XX:+HitachiOutOfMemoryStackTrace※ | 例外情報とスタックトレースを JavaVM ログファイルに出力することを指定します。 |
| | | -XX:+HitachiVerboseGC※ | GC が発生した場合に，拡張 verbosegc 情報を JavaVM ログファイルに出力することを指定します。 |
| | | -XX:+HitachiJavaClassLibTrace※ | クラスライブラリのスタックトレースを JavaVM ログファイルに出力することを指定します。 |

注　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。

注※　どれか一つでも指定すると，JavaVM ログファイルを出力します。なお，-XX:+HitachiOutOfMemoryStackTrace を指定すると，-XX:+HitachiOutOfMemorySize および-XX:+HitachiOutOfMemoryCause も同時に指定されます。

## 2.12　Java アプリケーションからの移行

アプリケーションサーバで提供する cjclstartap コマンドで実行していた Java アプリケーションは，バッチサーバでバッチアプリケーションとして実行できます。Java アプリケーションを，バッチサーバ上でバッチアプリケーションとして実行する場合，アプリケーションや実行環境の移行が必要となる場合があります。この節では，アプリケーションおよび実行環境の移行について，移行が必要な場合と移行方法を説明します。

この節の構成を次の表に示します。

表 2-33　この節の構成（Java アプリケーションからの移行）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 実装 | バッチアプリケーションの実装（Java アプリケーションからの移行） | 2.12.1 |
| 設定 | 実行環境での設定（バッチサーバの設定） | 2.12.2 |

注　「解説」，「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。

### 2.12.1　バッチアプリケーションの実装（Java アプリケーションからの移行）

ここでは，Java アプリケーションをバッチアプリケーションに移行する場合に，Java アプリケーションで必要な変更について説明します。

次の場合，Java アプリケーションの移行が必要になります。

- バッチアプリケーションの使用上の注意事項に該当する処理を実装している場合

  ファイルやディレクトリの操作などは，バッチアプリケーションに実装する際に注意が必要となります。

  移行方法

  バッチアプリケーションで注意が必要な処理は，「2.3.11(1)　注意が必要な処理」に記載しています。これらの内容を参照して，Java アプリケーションを修正してください。

- バッチアプリケーションで使用できない機能を実装している場合

  バッチアプリケーションでは使用できない機能が幾つかあります。例えば，標準入力からの入力や JNI の使用はできません。

  移行方法

  バッチアプリケーションで使用できない機能および機能を使用するための代替方法は，「2.3.11(2) バッチアプリケーションで実装できない機能」に記載しています。これらの内容を参照して，Java アプリケーションを修正してください。

- usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートされていないプロパティを定義している場合

  移行元の Java アプリケーションで使用していた usrconf.properties（Java アプリケーション用ユーザプロパティファイル）は引き続きバッチアプリケーションでも使用できます。

ただし，usrconf.properties（Java アプリケーション用ユーザプロパティファイル）の中に，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートされていないプロパティ※を定義して，バッチアプリケーションから値を参照している場合はアプリケーションの修正が必要です。

移行方法

usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートしていないプロパティの値を参照しないように，バッチアプリケーションを修正してください。

注※

ユーザが定義したプロパティを除きます。なお，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートされている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 リファレンス 定義編(サーバ定義)」の「3.2.6　usrconf.properties（バッチアプリケーション用ユーザプロパティファイル)」を参照してください。

## 2.12.2　実行環境での設定（バッチサーバの設定）

Java アプリケーションをバッチアプリケーションに移行する場合に，バッチサーバの設定の変更が必要な場合があります。ここでは，バッチサーバの設定で変更が必要な場合について説明します。

これまで Java アプリケーションの実行環境で使用していた次の二つのファイルは，バッチサーバの実行環境でもそのまま使用できます。

- usrconf.cfg（Java アプリケーション用オプション定義ファイル）
- usrconf.properties（Java アプリケーション用ユーザプロパティファイル）

ただし，次に示す条件に該当する場合は，ファイルの移行が必要になります。

- usrconf.cfg（Java アプリケーション用オプション定義ファイル）および usrconf.properties（Java アプリケーション用ユーザプロパティファイル）の格納場所を環境変数 CJCLUSRCONFDIR に設定している場合

  移行方法

  usrconf.cfg（バッチアプリケーション用オプション定義ファイル）および usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）の格納場所を環境変数 CJBATCHUSRCONFDIR に絶対パスで設定してください。

- usrconf.cfg（Java アプリケーション用オプション定義ファイル）の add.jvm.arg に，-cp，-classpath，-D 以外のオプションを設定している場合

  移行方法

  オプションの設定を usrconf.cfg（バッチサーバ用オプション定義ファイル）に記載してください。複数のバッチアプリケーションを一つのバッチサーバ上で順次実行する場合は，定義の設定値を調整する必要があります。次に例を示します。この例では，より大きい値を設定しているアプリケーション 2 の値をバッチサーバに設定しています。

例：アプリケーション 1 で add.jvm.arg=-Xmx512m，アプリケーション 2 で add.jvm.arg=-Xmx768m を設定していた場合

バッチサーバでは add.jvm.arg=-Xmx768m を設定してください。

- usrconf.cfg（Java アプリケーション用オプション定義ファイル）に ejb.client.log.directory を設定して，ログの出力先をデフォルトから変更している場合

  移行方法

  usrconf.cfg（バッチアプリケーション用オプション定義ファイル）に batch.log.directory を設定して，ログの出力先をデフォルトから変更してください。

- usrconf.cfg（Java アプリケーション用オプション定義ファイル）に，ejb.client.ejb.log または ejb.client.log.appid を設定して，ログの出力先をデフォルトから変更している場合

  移行方法

  ありません。ejb.client.ejb.log，および ejb.client.log.appid を使用して指定していたログの出力先は，バッチサーバの場合は指定できません。

- usrconf.cfg（Java アプリケーション用オプション定義ファイル）に ejb.client.directory.shareable=true を設定して，アプリケーションを複数同時に実行している場合

  移行方法

  一つのバッチサーバ上で，複数のバッチアプリケーションを同時に実行することはできません。このため，バッチアプリケーションの最大同時実行数のバッチサーバを用意してください。それぞれのバッチサーバ上でバッチアプリケーションが動作するように，cjexecjob コマンドに指定するサーバ名を変更してください。

- usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートされていないプロパティを定義している場合

  usrconf.properties（Java アプリケーション用ユーザプロパティファイル）の中に，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートされていないプロパティ※を定義している場合は，usrconf.properties（Java アプリケーション用ユーザプロパティファイル）の修正が必要です。

  移行方法

  usrconf.properties（Java アプリケーション用ユーザプロパティファイル）から，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートしていないプロパティの定義を削除してください。

注※

ユーザが定義したプロパティを除きます。なお，usrconf.properties（バッチアプリケーション用ユーザプロパティファイル）でサポートされている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 リファレンス 定義編(サーバ定義)」の「3.2.6　usrconf.properties（バッチアプリケーション用ユーザプロパティファイル)」を参照してください。

## 2.13　JP1/AJS との連携

バッチアプリケーションを実行するシステムは，JP1/AJS と連携して運用できます。また，JP1/AJS に加えて，BJEX または JP1/Advanced Shell を使用して運用することもできます。この節では，JP1/AJS，BJEX，および JP1/Advanced Shell と連携する場合の設定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 2-34　この節の構成（JP1/AJS との連携）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 設定 | JP1/AJS と連携するための設定 | 2.13.1 |
| | JP1/AJS，BJEX，および JP1/Advanced Shell と連携するための設定 | 2.13.2 |

注　「解説」，「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。

**参考**

JP1/AJS と連携するシステム，ならびに JP1/AJS と BJEX，および JP1/Advanced Shell と連携するシステムの概要については，「2.2.1　バッチアプリケーションを実行するシステム」および「2.2.2　バッチサーバおよびバッチアプリケーションの操作の流れ」を参照してください。

### 2.13.1　JP1/AJS と連携するための設定

ここでは，JP1/AJS と連携する場合の，JP1/AJS のジョブの定義について説明します。

なお，JP1/AJS からバッチアプリケーションを実行する際には，あらかじめバッチサーバを起動しておいてください。

#### (1)　バッチアプリケーションの開始

JP1/AJS と連携する場合，cjexecjob コマンドを JP1/AJS の UNIX ジョブまたは PC ジョブとして定義しておきます。JP1/AJS ジョブの属性を定義する画面の「スクリプトファイル名」「パラメーター」および「実行時のユーザー」の項目には，次に示す内容を設定してください。

- スクリプトファイル名

  cjexecjob コマンドを指定します。cjexecjob コマンドのパスについては，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「cjexecjob（バッチアプリケーションの実行）」を参照してください。

- パラメーター

  実行するバッチアプリケーションのクラス名と引数を指定します。

- 実行時のユーザー

バッチサーバを実行するユーザを指定します。

なお，JP1/AJS での設定の詳細については，マニュアル「JP1/Automatic Job Management System 操作ガイド」を参照してください。

### (2) バッチアプリケーションの強制停止

JP1/AJS と連携する場合，ジョブネットまたはジョブの強制終了をする際に，cjkilljob コマンドを JP1/AJS のリカバリージョブとして定義しておきます。ただし，ルートジョブネットを強制停止する場合は，リカバリージョブは実行されません。このため，バッチアプリケーションがバッチサーバ上で実行されたままになります。その場合は，cjkilljob コマンドを直接実行して，バッチアプリケーションを強制停止してください。

JP1/AJS での設定の詳細については，マニュアル「JP1/Automatic Job Management System 操作ガイド」を参照してください。

## 2.13.2 JP1/AJS，BJEX，および JP1/Advanced Shell と連携するための設定

ここでは，JP1/AJS，BJEX，および JP1/Advanced Shell と連携する場合の，JP1/AJS，BJEX，および JP1/Advanced Shell のジョブの定義について説明します。

なお，BJEX または JP1/Advanced Shell のバッチジョブアプリケーションを JP1/AJS から実行する際には，あらかじめバッチサーバを起動しておいてください。

### (1) バッチアプリケーションの開始

JP1/AJS，BJEX，および JP1/Advanced Shell と連携する場合，JP1/AJS，BJEX，および JP1/Advanced Shell にはそれぞれ次の内容を設定してください。

- BJEX と連携する場合の設定

  BJEX のバッチジョブに cjexecjob コマンドの実行を定義します。このとき，cjexecjob コマンドを，バッチジョブのジョブステップとして定義してください。

  また，BJEX のジョブ定義 XML には次の内容を定義します。

  - EXEC 要素

    cjexecjob コマンドを実行するための定義をします。
  - PGM 属性

    cjexecjob コマンドを定義します。
  - PARM 属性

    cjexecjob コマンドの引数を定義します。ただし，引数の長さの上限は，BJEX の仕様に準拠します。

BJEX での設定の詳細については，マニュアル「Batch Job Execution Server 使用の手引」を参照してください。

- JP1/Advanced Shell と連携する場合の設定

  JP1/Advanced Shell では adshjava コマンドを使用します。adshjava コマンドを JP1/Advanced Shell のジョブ定義スクリプト上で実行することで，adshjava コマンドの処理の中で cjexecjob コマンドが呼び出され，バッチアプリケーションが実行されます。adshjava コマンドには，バッチアプリケーションのクラス名に加えてバッチサーバ名称やスケジュールグループ名称を指定できるので，特定のバッチサーバ上でバッチアプリケーションを実行できます。

  adshjava コマンドの詳細については，マニュアル「JP1/Advanced Shell」を参照してください。

- JP1/AJS の設定

  BJEX または JP1/Advanced Shell のバッチジョブの実行コマンドをジョブとして定義します。

  JP1/AJS での設定の詳細については，マニュアル「JP1/Automatic Job Management System 操作ガイド」を参照してください。

## (2) バッチアプリケーションの強制停止

BJEX または JP1/Advanced Shell と連携する場合は，BJEX または JP1/Advanced Shell のバッチジョブの実行コマンドを強制停止するだけで，実行中のバッチアプリケーションも自動的に強制停止されます。そのため，リカバリージョブを定義する必要はありません。

# 3 CTM によるリクエストのスケジューリングと負荷分散

この章では，リクエストのスケジューリングと負荷分散について説明します。

業務システムには，局所的な障害発生時にシステムを止めることなく安定稼働できる信頼性と，随時変わっていく業務の処理要求に柔軟に対応できる可用性が求められます。アプリケーションサーバでは，これらの要求に対して，OLTP 技術を用いたリクエストのスケジューリングや，サーバのクラスタリングによる負荷分散などによって対応します。

なお，この章で説明する機能は，構成ソフトウェアに Component Transaction Monitor を含む製品だけで利用できる機能です。利用できる製品については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の「2.2.1　製品と構成ソフトウェアの対応」を参照してください。

## 3.1　この章の構成

この章では，CTM を使用したリクエストのスケジューリングと負荷分散について説明します。CTM を使用すると，クライアントから送信されたリクエストの実行を適切にスケジューリングしたり，複数の J2EE サーバに振り分けて処理したりできます。これによって，システムの安定稼働と処理性能向上を実現できます。

CTM を使用したリクエストのスケジューリングの概要については，「3.2　CTM を使用したリクエストのスケジューリングの概要」を参照してください。また，CTM を使用する場合のプロセス構成については，「3.3　CTM のプロセス構成」を参照してください。

CTM を使用して実行できる機能と参照先を次の表に示します。

表 3-1　CTM の機能

| 機能名 | 参照先 |
|---|---|
| リクエストの流量制御 | 3.4 |
| リクエストの優先制御 | 3.5 |
| リクエストの同時実行数の動的変更 | 3.6 |
| リクエストの閉塞制御 | 3.7 |
| リクエストの負荷分散 | 3.8 |
| リクエストのキューの滞留監視 | 3.9 |
| CTM のゲートウェイ機能を利用した TPBroker／OTM クライアントとの接続 | 3.10 |

また，CTM の稼働統計情報を収集することもできます。CTM の稼働統計情報の収集については，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「10.　CTM の稼働統計情報の収集」を参照してください。

## 3.2 CTM を使用したリクエストのスケジューリングの概要

この節では，CTM を使用したリクエストのスケジューリングの概要について説明します。

アプリケーションサーバでは，リクエストのスケジューリングに，**CTM**（Component Transaction Monitor）という構成ソフトウェアを使用します。CTM は，それぞれのリクエストを**キュー**を使用して制御します。このキューを，**スケジュールキュー**といいます。

### 3.2.1 リクエストをスケジューリングする目的

規模の大きい業務システムでは，特定の J2EE アプリケーションを実行している J2EE サーバに大量のリクエストが集中することがあります。それぞれのサーバの負荷を抑え，可用性を確保して業務を滞りなく進めるためには，リクエストの送り先を分散したり，一定時間内のリクエストの流量を制御したりすることが必要です。また，複数の J2EE サーバで処理を分散する場合に，リクエストが送信された時点で負荷が最も掛かっていない J2EE サーバに処理させることができれば，システム全体の処理性能を向上させられます。

これらを実現するのが，リクエストのスケジューリングです。これによって，それぞれの J2EE サーバが持つ性能を十分に活用しながら，安定して稼働するシステムを構築できます。また，リクエストをスケジューリングすることで，特定の J2EE サーバ，J2EE アプリケーションまたは業務処理プログラム（Enterprise Bean）にトラブルが発生した場合にも，該当する範囲だけを閉塞して縮退運転できるので，システム全体の可用性を高められます。

アプリケーションサーバでは，リクエストをスケジューリングすることで，次の 6 種類の機能を実現できます。

- **流量制御**
  J2EE サーバで一度に実行される処理数の最大値を制限することで，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実現できます。
- **優先制御**
  クライアントに優先順位を付けておくことで，優先順位の高いクライアントからのリクエストを優先的に処理します。
- **同時実行数の動的変更**
  リクエストの同時実行数を一時的に変更したい場合に，CTM デーモンを停止しないで同時実行数を変更できます。
- **閉塞制御**
  特定の J2EE アプリケーションに対するリクエストの受け付けを停止したり，リクエストを滞留したりすることで，システム全体を稼働させたままのメンテナンスを可能にします。これによって，システムの可用性が高められます。
- **負荷分散**

J2EE サーバ間で負荷が均等になるように処理を分散して割り当て，システム全体の処理性能と可用性が高められます。

- **キューの滞留監視**
  スケジュールキューで滞留しているリクエストの数を監視できます。

## 3.2.2　CTM が制御できるリクエストの種類

CTM でスケジューリングできるリクエストは，RMI-IIOP 通信を使用する，**Stateless Session Bean に対するリモートインタフェース呼び出し**だけです。

次のリクエストは，CTM でのスケジューリングができません。

**CTM でスケジューリングできないリクエスト**

- Stateful Session Bean および Entity Bean に対する呼び出し。
- ローカルインタフェースによる呼び出しおよび Message-driven Bean に対する呼び出し（RMI-IIOP 通信を使用しないため，対象になりません）。
- EJB3.0 以降の Enterprise Bean に対する呼び出し。

なお，同一の J2EE アプリケーション内の業務処理を呼び出す場合には，リクエストのスケジューリングをしたいときだけ，リモートインタフェースを使用してください。リクエストのスケジューリングをしないときは，処理性能を考慮して，ローカルインタフェースを使用して呼び出すことをお勧めします。

また，リクエストを CTM による制御の対象にするかどうかは，J2EE アプリケーション単位，または J2EE アプリケーションに含まれる業務処理プログラム単位（Bean 単位）で選択できます。例えば，リモートインタフェースを持つ業務処理プログラムを CTM による制御の対象から外したい場合は，J2EE アプリケーションのプロパティを定義して設定を変更してください。CTM によるリクエストのスケジューリングの設定については，「3.4.2　実行環境での設定」を参照してください。

## 3.2.3　リクエストを送信するクライアントアプリケーション

CTM を使用できるクライアントは，EJB クライアントである，次のクライアントです。

- EJB クライアントアプリケーション
- JSP／サーブレット
- ほかの Enterprise Bean

これらを開発する時に，特別なインタフェースを使用する必要はありません。CTM デーモンと連携するグローバル CORBA ネーミングサービス（ctmstart の-CTMINSRef オプションに指定した CORBA ネーミングサービス）に対してルックアップするように設定してください。

ただし，システム内の特定のアプリケーションサーバに障害が発生した場合にルックアップ対象の CORBA ネーミングサービスを切り替えられるように，lookup，create，invoke，remove のどの処理で例外が発生しても JNDI の lookup から処理をし直すようにコーディングしてください。

### 3.2.4　CTM を使用する場合に実行される処理

CTM を使用する場合，次のタイミングで，CTM を使用するための処理が実行されます。

- J2EE サーバの起動処理
- J2EE アプリケーションの開始処理
- J2EE アプリケーションの停止処理
- J2EE サーバの停止処理

ここでは，それぞれのタイミングで実行される処理について説明します。

#### (1)　J2EE サーバの起動処理

CTM を使用するようにカスタマイズされている J2EE アプリケーションを開始するためには，J2EE サーバの起動時に，CTM デーモンとのコネクションを確立して初期化処理を実行する必要があります。コネクションの確立と初期化は，次の操作によって実行されます。

1. CTM を使用するための設定をします。
2. CTM デーモンを起動します。
3. J2EE サーバを起動します。

J2EE サーバが起動する時に，J2EE サーバによって CTM デーモンとのコネクション確立と初期化処理が実行されます。J2EE サーバの起動前に必ず CTM デーモンを起動してください。

CTM デーモンを使用するための設定については，「3.4.2　実行環境での設定」を参照してください。CTM デーモンおよび J2EE サーバの起動については，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「4.1.24　システムを起動する（CUI 利用時）」を参照してください。Smart Composer 機能を使用してシステムを起動すると，CTM デーモン，J2EE サーバの順序で起動処理が実行されます。

なお，J2EE サーバの起動時に CTM デーモンとのコネクション確立と初期化に失敗した場合は，J2EE サーバの起動が失敗します。この場合は，失敗した要因に対処してから，J2EE サーバを再起動してください。

#### (2)　J2EE アプリケーションの開始処理

J2EE アプリケーションを開始すると，J2EE サーバから CTM デーモンに対して，指定されたキュー名称でスケジュールキューを活性化する要求が発行されます。CTM デーモンでは，キューを活性化してから，CTM デーモンの処理対象になる業務処理プログラムの create を，J2EE サーバに対して実行します。

create は，CTM デーモンから直接呼び出される業務処理プログラムごとに，同時実行スレッド数（Parallel Count）分実行されます。

業務処理プログラムに対応する EJB オブジェクトリファレンスが作成されると，それが CTM デーモンに返却されます。CTM デーモンでは，それをプールしておき，スケジュールキューにリクエストが登録された時に，それぞれのリクエストに割り当てます。これによって，リクエストが振り分けられます。

### (3) J2EE アプリケーションの停止処理

J2EE アプリケーションを停止する時には，まず，CTM デーモンから新たにリクエストが振り分けられないように，CTM デーモンが管理しているスケジュールキューを閉塞（非活性化要求）します。CTM デーモンでは，キューを非活性化してから，CTM デーモンの処理対象になる業務処理プログラムの remove を，J2EE サーバに対して実行します。remove は，CTM デーモンから直接呼び出される業務処理プログラムごとに，同時実行スレッド数（Parallel Count）分実行されます。

そのあとで，CTM を使用しない場合と同じように，J2EE アプリケーションの停止処理が実行されます。

### (4) J2EE サーバの停止処理

J2EE サーバを停止する時に，J2EE サーバと CTM デーモンとのコネクションが切断されます。

## 3.2.5 スケジュールキューの作成単位とキューの共有

キューは，J2EE アプリケーション単位，または Bean 単位で作成できます。ここでは，スケジュールキューの構成と，キューの共有について説明します。また，キューを共有する場合としない場合の利点についても説明します。

### (1) スケジュールキューの作成単位

クライアントからのリクエストは，CTM デーモンによって管理される**スケジュールキュー**によってスケジューリングされます。スケジュールキューは，J2EE アプリケーション単位または Bean 単位で作成されます。デフォルトの**キュー名称**は，J2EE アプリケーション単位の場合は J2EE アプリケーションの名称，Bean 単位の場合は Bean 名となります。

### (2) スケジュールキューの共有

J2EE アプリケーション単位または Bean 単位でスケジュールキューを持つことで，異なるインタフェースを持つ業務処理プログラム間，または J2EE アプリケーションの Bean 間でキューを共有できます。なお，**スケジュールキューで制御されるリクエスト**は，「EJB ホームリファレンスのグローバル CORBA ネーミングサービス登録名称」と「業務処理プログラムのリモートインタフェース名称」の組み合わせで管理されます。

スケジュールキューは，同じ CTM デーモンに対応づけられている，次の J2EE アプリケーション間または Bean 間で共有できます。

J2EE アプリケーション間で共有する場合

- キュー名称が同じである
- 業務処理プログラムの構成が同じである（J2EE アプリケーションに含まれる Enterprise Bean が，CTM が認識する範囲で完全に一致する）

Bean 間で共有する場合

- キュー名称が同じである
- Bean が同じである

同じ業務処理プログラム構成でもキュー名称が異なる場合や，同じキュー名称でも異なる業務処理プログラム構成を持つ J2EE アプリケーション間では，スケジュールキューは共有できません。

複数の J2EE サーバ間でスケジュールキューを共有することもできます。ただし，その場合は，ユーザ指定名前空間機能を使用して，それぞれの業務処理プログラムである Enterprise Bean に別名（Optional Name）を付与する必要があります。ユーザ指定名前空間機能については，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「2.3　JNDI 名前空間へのオブジェクトのバインドとルックアップ」を参照してください。別名の付与は，J2EE アプリケーションのプロパティとして設定します。

**参考**

- デフォルトの設定で J2EE アプリケーションをインポートすると，業務処理プログラムをルックアップするための名称が「/HITACHI_EJB/SERVERS/＜J2EE サーバ名称＞/EJB/＜J2EE アプリケーション名称＞/＜業務処理プログラムの名称＞」になります。この場合，ルックアップ名称に J2EE サーバ名称が含まれてしまうため，J2EE サーバ間でスケジュールキューを共有できません。
- 一つの J2EE サーバ内で複数の J2EE アプリケーションを同一名称でインポートして，スケジュールキューを共有することはできません。

## (3)　スケジュールキューを共有する効果

スケジュールキューを共有する効果について，J2EE アプリケーション単位での共有と，Bean 単位での共有に分けて説明します。

### (a)　J2EE アプリケーション単位の場合

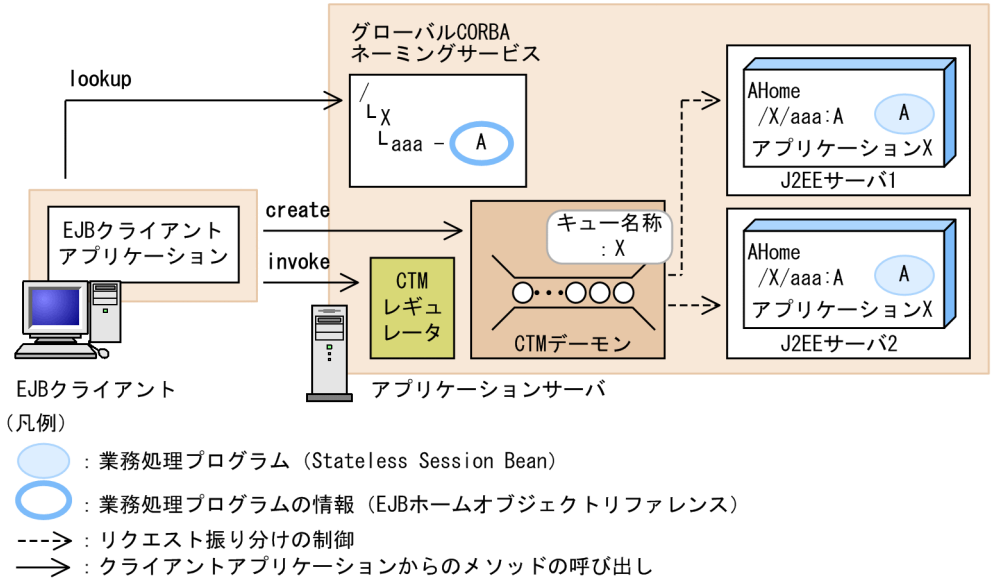
スケジュールキューを共有すると，複数の J2EE サーバ上の J2EE アプリケーションで，リクエストを分散処理できます。

スケジュールキューの共有には，次のような効果があります。

- キューを共有する J2EE アプリケーション間で同時実行スレッド数を制御できるので，特定の J2EE アプリケーションに高い負荷が掛かった時の性能劣化を防げます。これによって，システムとしての処理性能を安定させやすくなります。
- キューを共有しているどちらかの J2EE サーバで障害が発生した場合に，縮退運転に切り替え，キューに滞留したリクエストを障害が発生していない J2EE サーバの J2EE アプリケーションで処理できます。このため，業務が停止しません。

スケジュールキューを共有する例を次の図に示します。

図 3-1　スケジュールキューを共有する例（J2EE アプリケーション単位）




EJB クライアントからのルックアップは，グローバル CORBA ネーミングサービスに対して実行します。スケジュールキューを共有している場合，共有しているキュー（この場合は X）のリファレンスが取得できるので，そのキューに対して create を実行します。そこで取得した CTM レギュレータのリファレンスに対して invoke を実行すると，スケジュールキュー X によって，J2EE サーバ 1 または J2EE サーバ 2 に処理が振り分けられます。

## (b)　Bean 単位の場合

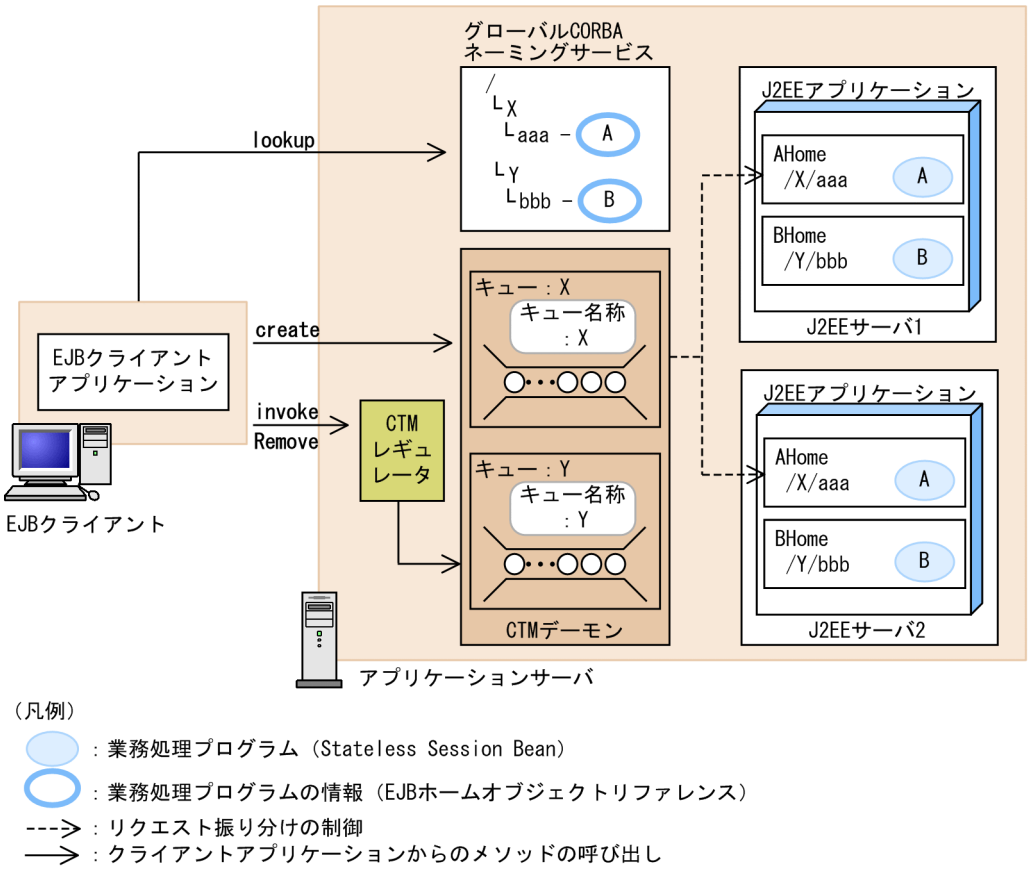
スケジュールキューを共有すると，複数の J2EE サーバ上の Bean でリクエストを分散処理できます。

スケジュールキューの共有には，次のような効果があります。

- 同一の J2EE アプリケーション内にあるほかの Bean の影響を受けることなく，J2EE アプリケーション内の Bean 種別ごとにキューを分けることができます。
- キューを共有する Bean 間で同時実行スレッド数を制御できるので，特定の Bean に高い負荷が掛かった時の性能劣化を防げます。これによって，システムとしての処理性能を安定させやすくなります。
- キューを共有しているどちらかの J2EE サーバで障害が発生した場合，縮退運転に切り替え，キューに滞留したリクエストを障害が発生していない J2EE サーバ上の Bean で処理できます。このため，業務が停止しません。

スケジュールキューを共有する例を次の図に示します。

図 3-2　スケジュールキューを共有する例（Bean 単位）




EJB クライアントからのルックアップは，グローバル CORBA ネーミングサービスに対して実行します。スケジュールキューを共有している場合，共有しているキュー（この場合は X）のリファレンスが取得できるので，そのキューに対して create を実行します。そこで取得した CTM レギュレータのリファレンスに対して invoke を実行すると，スケジュールキュー X によって，J2EE サーバ 1 または J2EE サーバ 2 上の Bean A に処理が振り分けられます。

## (4)　スケジュールキューを共有しない効果

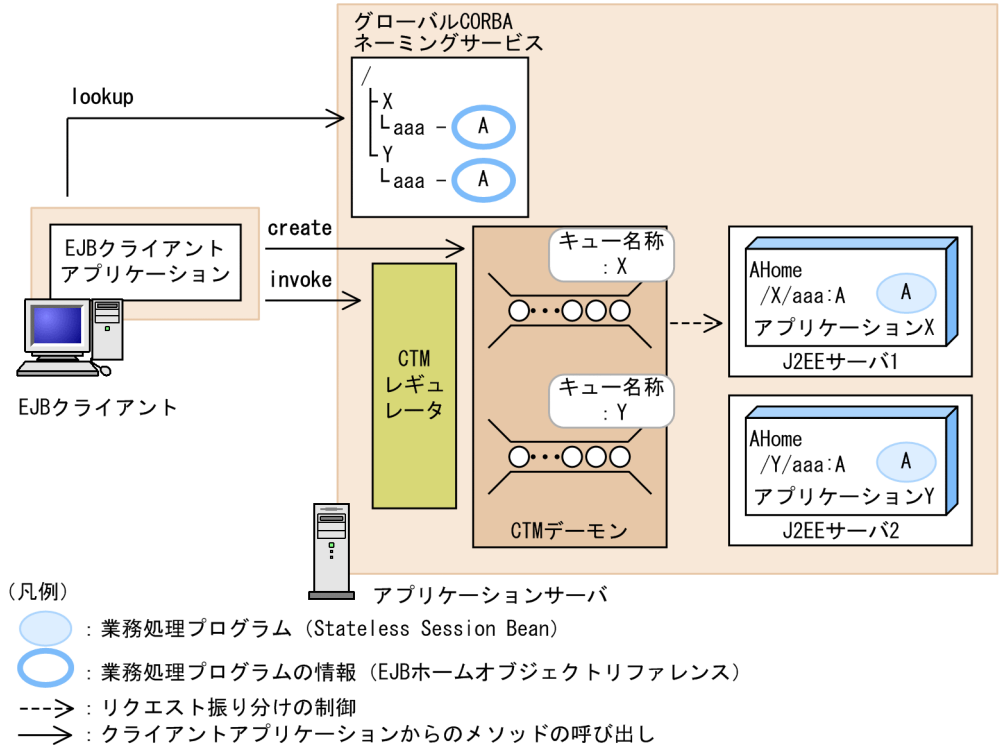
スケジュールキューを共有しない場合，異なる J2EE サーバに同じ J2EE アプリケーションがインポートされていたり，J2EE サーバ上に同じ Bean があったりしても，リクエストはそれぞれのキューで制御され，実行されます。

スケジュールキューを共有しないと，負荷分散や縮退運転はできなくなりますが，異なるスケジュールキューでリクエストが滞留してもその影響を受けなくなるため，優先的に業務処理を進められます。

スケジュールキューを共有しない場合は，それぞれの J2EE アプリケーションに含まれる業務処理プログラムに別名を指定しないで，それぞれ異なるルックアップ名称にします。

スケジュールキューを共有しない例を次の図に示します。

図 3-3　スケジュールキューを共有しない例




EJB クライアントからのルックアップは，グローバル CORBA ネーミングサービスに対して実行します。スケジュールキューを共有していない場合，指定した J2EE アプリケーションを制御するキュー（この場合は X）のリファレンスが取得できるので，そのキューに対して create を実行します。そこで取得した CTM レギュレータのリファレンスに対して invoke を実行すると，スケジュールキュー X が制御している J2EE サーバ 1 に処理が振り分けられます。

## 3.2.6　スケジュールキューの長さ

スケジュールキューの長さは，次の単位で設定できます。

- CTM デーモン単位
- J2EE アプリケーション単位
- Session Bean 単位

CTM デーモン単位の設定については，「3.3.3(2)　スケジュールキューへのリクエストの登録」を参照してください。

J2EE アプリケーション単位または Session Bean 単位の設定は，アプリケーション属性ファイルまたは Session Bean 属性ファイルの<scheduling>-<queue-length>タグで設定します。CTM によるリクエストのスケジューリングの設定については，「3.4.2(3)　サーバ管理コマンドでの設定」を参照してください。

ただし，スケジュールキューを共有する場合は，スケジュールキューはすでに作成されているため，指定したスケジュールキューの長さは無効になります。

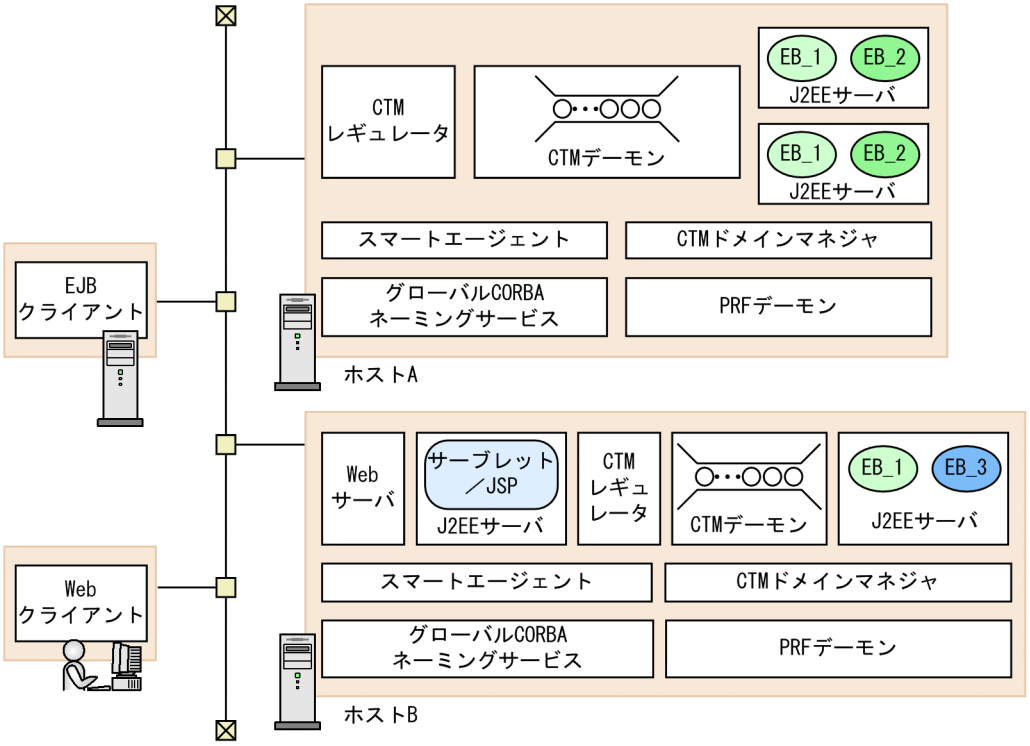
## 3.3　CTM のプロセス構成

CTM を使用してリクエストをスケジューリングする環境のプロセス構成と配置の指針について説明します。また，各プロセスの機能についても説明します。

### 3.3.1　CTM のプロセス構成と配置

CTM を使用する場合のプロセスの配置例を次の図に示します。

図 3-4　CTM を構成するプロセスの配置例




各プロセスの主な機能について，次の表で説明します。

表 3-2　CTM で使用するプロセス

| プロセス | 説明 |
|---|---|
| CTM デーモン | クライアントからのリクエストを制御するスケジュールキューを管理するプロセスです。 |
| CTM レギュレータ | CTM デーモンに集中するリクエストを，分散集約するためのプロセスです。 |
| CTM ドメインマネジャ | CTM ドメインを管理するプロセスです。CTM ドメインは，複数の CTM デーモンで構成される，情報共有と負荷分散の対象になる範囲です。 |
| グローバル CORBA ネーミングサービス | 同じ CTM ドメイン内に含まれるホスト上の業務処理プログラムの情報を共有管理しているネーミングサービスです。 |
| PRF デーモン<br>（パフォーマンストレーサ） | CTM デーモンが出力した性能解析情報をファイルに出力する，I／O プロセスです。 |

| プロセス | 説明 |
| --- | --- |
| | 詳細は，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「7.5　実行環境での設定」を参照してください。 |
| スマートエージェント | TPBroker で提供されている，動的な分散ディレクトリサービスです。CTM によってリクエストのスケジューリングをする場合，スマートエージェントが必要です。また，異なるネットワークセグメントにある CTM デーモンに情報を配布する場合などにも使用されます。<br>詳細は，マニュアル「Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド」を参照してください。 |

## 3.3.2　プロセス配置の指針

プロセスを配置する場合の指針を示します。

- CTM デーモンは，ホストに一つ配置します。
- J2EE サーバまたは CTM レギュレータを配置するホストには，CTM デーモンが必要です。
- 一つの CTM デーモンで，複数の J2EE サーバを制御できます。
- 一つの CTM デーモンに対して，複数の CTM レギュレータを配置できます。一つの CTM レギュレータに送信される同時リクエスト数が 256 を超えるような場合は，性能が劣化するおそれがあります。この場合は，CTM レギュレータの配置数を増やしてください。
- EJB クライアントが動作するクライアントホストに CTM デーモンは必要ありません。
- CTM デーモンを配置したホストに，CTM ドメインマネジャを一つ配置します。複数の CTM デーモンを同じ CTM ドメインに参加させたい場合，各ホストの CTM ドメインマネジャの名称には，同じ名称を指定します。
- 統合ネーミングスケジューラサーバにするホスト（J2EE サーバを配置しないホスト）にも，CTM デーモンを配置します。CTM レギュレータは不要です。なお，統合ネーミングスケジューラサーバについては，「(4)　独立した統合ネーミングスケジューラサーバを構築する構成（統合ネーミングスケジューラサーバモデル)」で説明します。

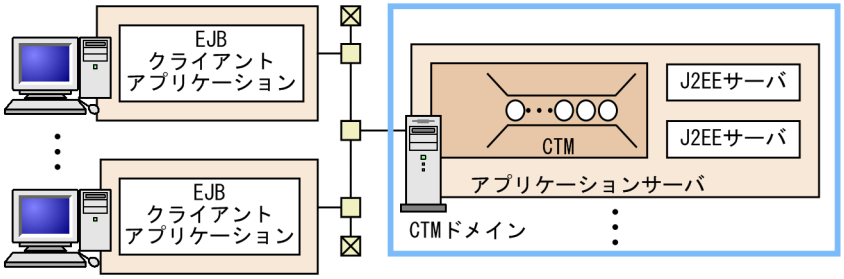
各プロセスの起動方法については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「2.　システムの起動と停止」を参照してください。

CTM で使用するプロセスの配置パターンについて説明します。

### (1)　多数の EJB クライアントから J2EE サーバを呼び出す構成

多数の EJB クライアントから，並列に配列したアプリケーションサーバ上の J2EE サーバを呼び出す構成の例を次の図に示します。

図 3-5　複数の EJB クライアントから J2EE サーバを呼び出す構成の例

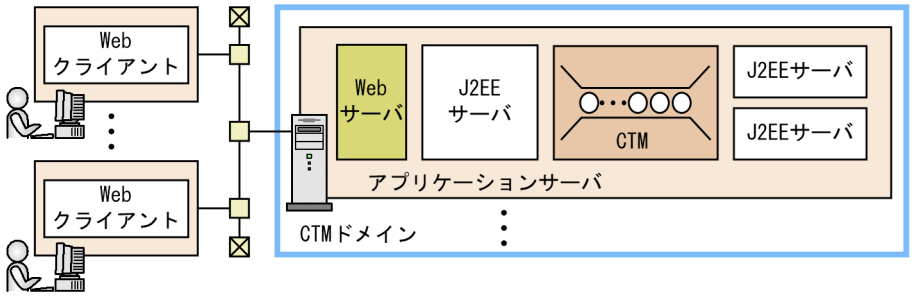



注　図中の「CTM」には，CTMデーモン，CTMレギュレータ，スマートエージェント，
グローバルCORBAネーミングサービス，CTMドメインマネジャおよびPRFデーモンを含みます。

## (2)　Web ブラウザから J2EE サーバを呼び出す構成（小規模）

Web ブラウザから，並列に配置した Web サーバ／アプリケーションサーバ上の Web コンテナを経由して，J2EE サーバを呼び出す構成の例を次の図に示します。なお，Web サーバとアプリケーションサーバは同じホスト上に配置しています。

図 3-6　Web ブラウザから J2EE サーバを呼び出す構成（小規模）の例

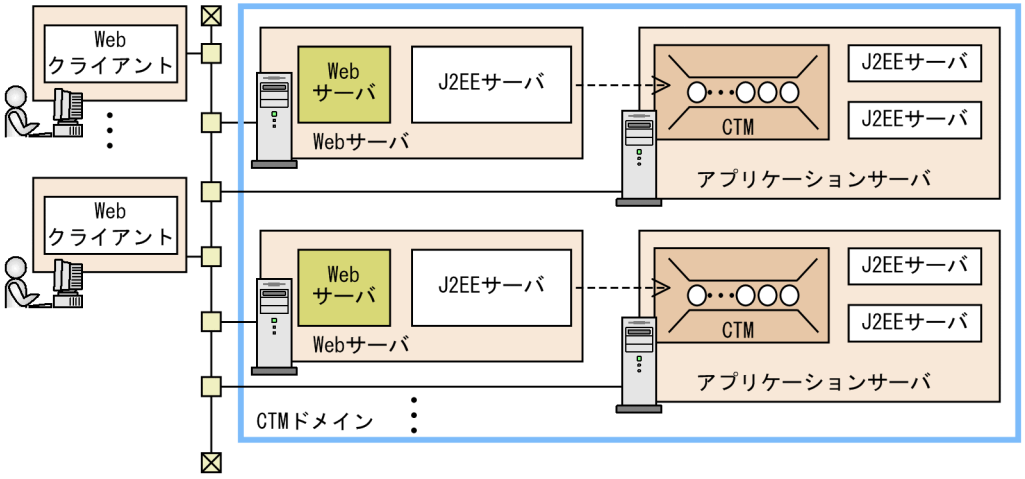



注　図中の「CTM」には，CTMデーモン，CTMレギュレータ，スマートエージェント，
グローバルCORBAネーミングサービス，CTMドメインマネジャおよびPRFデーモンを含みます。

## (3)　Web ブラウザから J2EE サーバを呼び出す構成（大規模）

Web ブラウザから Web サーバ上の Web コンテナを経由して，アプリケーションサーバ上の J2EE サーバを呼び出す構成の例を次の図に示します。Web サーバとアプリケーションサーバのホストを分けることで，それらを容易に多対多の関係で構成できます。

図 3-7　Web ブラウザから J2EE サーバを呼び出す構成（大規模）の例




注　図中の「CTM」には，CTMデーモン，CTMレギュレータ，スマートエージェント，
グローバルCORBAネーミングサービス，CTMドメインマネジャおよびPRFデーモンを含みます。

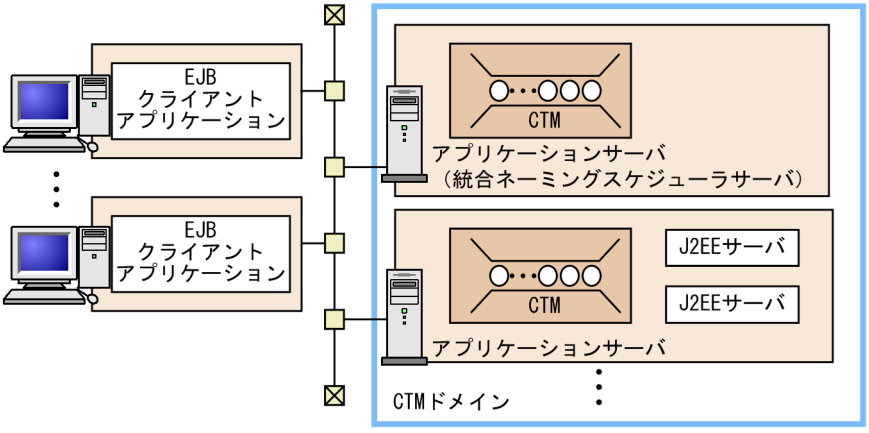
## (4)　独立した統合ネーミングスケジューラサーバを構築する構成（統合ネーミングスケジューラサーバモデル）

グローバル CORBA ネーミングサービスを独立したホストに配置してレプリカを作成することで，ネーミングサービスの可用性を向上させることができます。このホストを，統合ネーミングスケジューラサーバといいます。統合ネーミングスケジューラサーバに J2EE サーバを配置する必要はありません。

ただし，ほかのホスト上の業務処理プログラム情報を統合ネーミングスケジューラサーバのグローバル CORBA ネーミングサービスに登録するためには，CTM デーモンは統合ネーミングスケジューラサーバにも配置しておく必要があります。

独立した統合ネーミングスケジューラサーバを構築する構成を次の図に示します。

図 3-8　独立した統合ネーミングスケジューラサーバを構築する構成（統合ネーミングスケジューラサーバモデル）の例




注　図中の「CTM」には，CTMデーモン，CTMレギュレータ，スマートエージェント，
グローバルCORBAネーミングサービス，CTMドメインマネジャおよびPRFデーモンを含みます。

なお，この構成の場合，統合ネーミングスケジューラサーバの CTM デーモンには create 要求以外が送信されないため，統合ネーミングスケジューラサーバでは CTM レギュレータを起動しなくてもかまいません。

### 3.3.3　CTM デーモン

CTM デーモンは，クライアントからのリクエストを処理してリクエストのスケジューリングを実現する，スケジューラの機能を持つプロセスです。

**注意事項**

Windows のサービスから起動する場合は，開始コマンドのオプションに，「-Dvbroker.orb.isNTService=true」を指定してください。

クライアントから送信されたリクエストは，CTM レギュレータというプロセスを経由して，CTM デーモンが受信します。CTM レギュレータについては「3.3.4　CTM レギュレータ」を参照してください。

なお，CTM デーモンの機能を使用するための設定は，CTM デーモンを起動するときに ctmstart コマンドの引数として指定します。また，運用管理ポータルで構築したシステムを運用している場合は，論理 CTM であらかじめ設定しておくことができます。

CTM デーモンは，次の手順でリクエストを処理します。

1. リクエストの振り分け
2. スケジュールキューへのリクエストの登録
3. 業務処理プログラムの呼び出し
4. 処理結果の返却

それぞれの処理内容について説明します。

#### (1)　リクエストの振り分け

CTM デーモンの負荷に応じて，リクエストを受け付けた CTM デーモンがそのままリクエストを処理するか，またはほかの CTM デーモンに転送するかを決定して振り分けます。

CTM デーモンは，リクエストを受け付けると，負荷情報をほかの CTM デーモンと連絡し合ってリクエストの振り分け先を決定します。

CTM デーモンは，特定の範囲（CTM ドメイン）内の CTM デーモン間で，それぞれが処理対象としている J2EE アプリケーションに含まれる業務処理プログラムの情報を共有しています。共有した情報は，CTM デーモンと同じホストに配置されている**グローバル CORBA ネーミングサービス**に登録されています。これによって，リクエストを受け付けた CTM デーモンの処理対象に該当する業務処理プログラムがない場合でも，CTM デーモンが持つ情報を基に，適切な CTM にリクエストを振り分けられます。

グローバル CORBA ネーミングサービスについては，「3.3.6　グローバル CORBA ネーミングサービス」を参照してください。CTM ドメインについては，「3.3.5　CTM ドメインと CTM ドメインマネジャ」を参照してください。

リクエストは，**create 時の選択ポリシー**，および**スケジュールポリシー**に従って振り分けられます。

create 時の選択ポリシーおよびスケジュールポリシーは，CTM デーモンを起動するときに，次のどちらかを選択できます。

- それぞれの CTM デーモンの負荷状況に応じて，負荷が軽い CTM デーモンにリクエストを振り分けるポリシー
- リクエストを受け付けた CTM デーモンが優先的に処理をするポリシー

  ただし，この場合でも，リクエストを受け付けた CTM デーモンの負荷が高い状態の場合，または閉塞状態の場合は，ほかの CTM デーモンに処理を振り分けます。負荷の高さは，それぞれのキューでのリクエストの滞留率から計算されます。

なお，create 時の選択ポリシーおよびスケジュールポリシーのタイミングについては，「3.8　リクエストの負荷分散」を参照してください。

スケジュールポリシーは，ctmstart コマンドの引数-CTMDispatchPolicy に指定します。create 時の選択ポリシーは，ctmstart コマンドの引数-CTMCreatePolicy に指定します。

**リクエスト転送時のタイムアウトについて**

CTM デーモン間で，リクエスト転送処理が完了するまでの待ち時間にタイムアウトを設定できます。タイムアウトの設定は，ctmstart コマンドの-CTMDCSendTimeOut オプションに指定します。

## (2)　スケジュールキューへのリクエストの登録

スケジュールポリシーに従って振り分けられたリクエストは，スケジュールキューに登録されます。スケジュールキューに登録できるリクエストの数は，CTM デーモンを起動する時に，設定されています。この値を超えてリクエストを転送された場合には，クライアントにエラーが返却されます。また，指定を省略した場合は，50 が設定されます。

登録できるリクエストキューの長さは，CTM デーモンを起動するときに ctmstart コマンドの引数-CTMMaxRequestCount に指定します。また，運用管理ポータルで構築したシステムを運用する場合は，あらかじめ論理 CTM の設定で設定しておくことができます。ctmstart 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「ctmstart（CTM デーモンの開始）」を参照してください。運用管理ポータル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 運用管理ポータル操作ガイド」の「10.7.2　スケジューリングの設定」を参照してください。

## (3)　業務処理プログラムの呼び出し

スケジュールキューに登録されたリクエストは，CTM デーモンの処理対象である J2EE サーバ上の業務処理プログラムを呼び出します。なお，異常終了した J2EE サーバや，ハングアップした業務処理プログラムを呼び出すことはありません。

### (4) 処理結果の返却

処理の実行後，業務処理プログラム（Enterprise Bean）から返却された応答は，CTM デーモンを経由してクライアントに返却されます。なお，リクエストがスケジュールキューに登録されてから取り出されるまでの時間がリクエストのタイムアウト値を超えている場合は，リクエストが破棄されます。

## 3.3.4 CTM レギュレータ

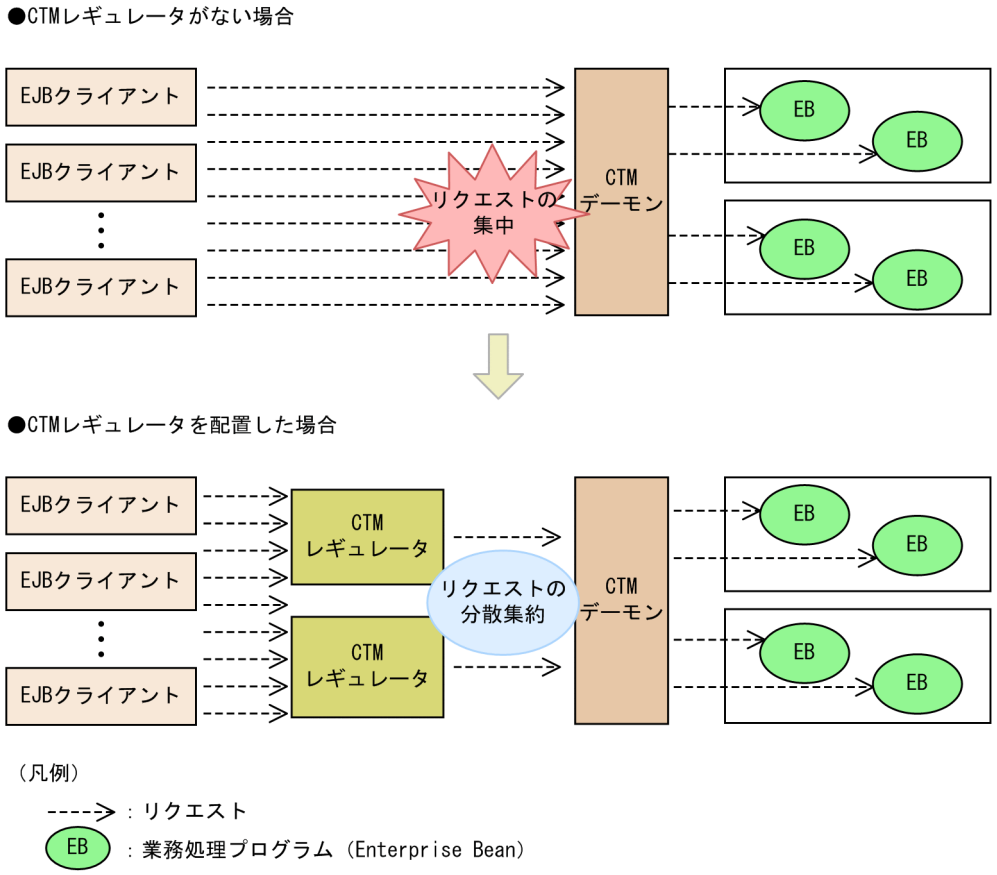
CTM レギュレータは，CTM デーモンに対するリクエストの集中による問題を，コネクションやリクエストの**レギュレート**（集約）によって解決するプロセスです。CTM デーモンのフロントに配置され，EJB クライアントからのコネクションやリクエスト（invoke または remove）の集中を，分散集約します。

例えば，大規模なシステムでクライアントの数が増え，リクエスト数が増大した場合，システムの動作状態が不安定になったり，システムで管理する資源が不足して正常な処理ができなくなったりします。これは，リクエストのスケジューリングをする CTM デーモンに対するリクエストの集中によって，コネクション数が増大し，ファイルやソケットのオープン数などのプロセス使用資源が増大するためです。

CTM レギュレータは，このようなリクエスト集中による問題を解決するための専用のプロセスです。CTM レギュレータは，クライアントからの幾つかのコネクションを集約し，CTM デーモン当たりのコネクション数の上限を管理します。これを，**コネクションのレギュレート**といいます。CTM レギュレータが大量のコネクションをレギュレートすることによって，使用資源を複数プロセスに分散し，大規模なシステムをより安定して動作させることができます。

コネクションのレギュレートの仕組みを，次の図に示します。

図 3-9　コネクションのレギュレートの仕組み




CTM レギュレータは，EJB クライアントからのリクエストを受け付けると，対応する CTM デーモンにリクエスト転送して，リクエストの応答を待ちます。応答が返ってくると，その応答を EJB クライアントに返却します。

CTM では，一つの CTM デーモンに対して，必要に応じて複数の CTM レギュレータを配置できます。一つの CTM レギュレータに対する同時リクエスト数が 256 を超えるような場合は，性能が劣化するおそれがあります。この場合は，クライアントプロセス数に関係なく，CTM レギュレータを増やしてください。なお，CTM レギュレータは，対応する CTM デーモンと同じホスト上に配置する必要があります。

また，ネーミングサービスを J2EE サーバとは別なホストに配置している統合ネーミングスケジューラサーバモデルの場合，統合ネーミングスケジューラサーバでは，create 以外のリクエストを受け付けません。このため，CTM レギュレータを起動する必要はありません。

## 3.3.5　CTM ドメインと CTM ドメインマネジャ

**CTM ドメイン**とは，複数の CTM デーモン間で，それぞれに登録された業務処理プログラムの情報やスケジュールキューの負荷情報を交換して，情報共有と負荷分散をする範囲のことです。CTM ドメイン名称で識別されます。CTM デーモンは，同じ CTM ドメイン内に存在する CTM デーモン間で，リクエストの振り分けやスケジューリングをします。CTM ドメインの範囲と，CTM ドメイン内の各 CTM デーモンの情報は，CTM ドメインマネジャによって管理されます。

**ポイント**

CTM ドメインは，Management Server が管理する運用管理ドメイン内に含まれます。

**注意事項**

CTM ドメインを新しく増やすと，ファイルシステム中に情報が増えます。使用しなくなった CTM ドメインに対する CTM ドメイン情報は，ctmdminfo コマンドを使用して適宜削除してください。

CTM ドメインマネジャは，同じ CTM ドメイン内の CTM デーモンの情報を管理するデーモンプロセスです。CTM デーモンを配置したホスト上に一つずつ配置します。

対象となる CTM ドメインマネジャが同じネットワークセグメント内にあるか，異なるネットワークセグメントにあるかによって，CTM ドメインマネジャによる情報の配布方法が異なります。

なお，CTM ドメインマネジャの機能を使用するための設定は，CTM ドメインマネジャを起動するときに ctmdmstart コマンドの引数として指定します。また，運用管理ポータルで構築したシステムを運用している場合は，論理 CTM ドメインマネジャにあらかじめ設定しておくことができます。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「ctmdmstart（CTM ドメインマネジャの開始)」を参照してください。運用管理ポータルについては，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」の「10.6.2　CTM ドメインマネジャのネットワーク設定」を参照してください。

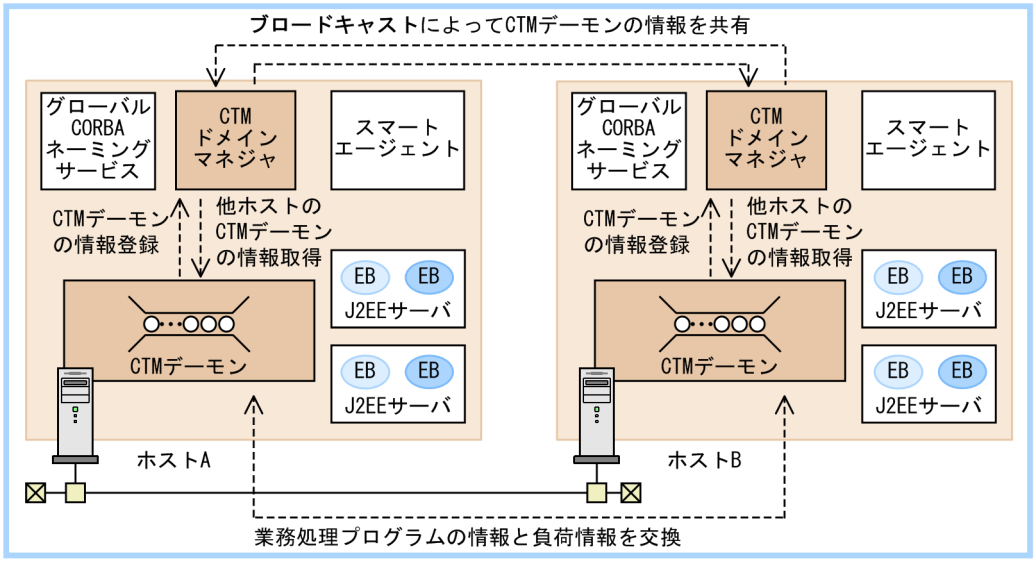
**注意事項**

- Windows のサービスから起動する場合は，開始コマンドのオプションに，「-Dvbroker.orb.isNTService=true」を指定してください。
- Windows で CTM デーモンが異常終了した場合，CTM ドメインマネジャは CTM デーモンの子プロセスを強制停止します。
- CTM ドメインマネジャが異常終了した場合，CTM ドメインマネジャの正常起動コマンド（ctmdmstart コマンド）に，-CTMForceStart オプション，または-CTMAutoForce オプションを指定してください。

## (1)　対象の CTM ドメインマネジャが同じネットワークセグメント内にある場合の情報共有

CTM ドメインマネジャは，ホスト内の CTM デーモンの情報を，ほかのホスト上の CTM ドメインマネジャにブロードキャストで配布します。対象の CTM ドメインマネジャが同じネットワークセグメント内にある場合の情報共有について，次の図に示します。

図 3-10　同じネットワークセグメント内での CTM ドメインマネジャによる情報共有




CTMドメイン

（凡例）

EB EB ：業務処理プログラム

注　　CTMレギュレータおよびPRFデーモンのプロセスは省略しています。

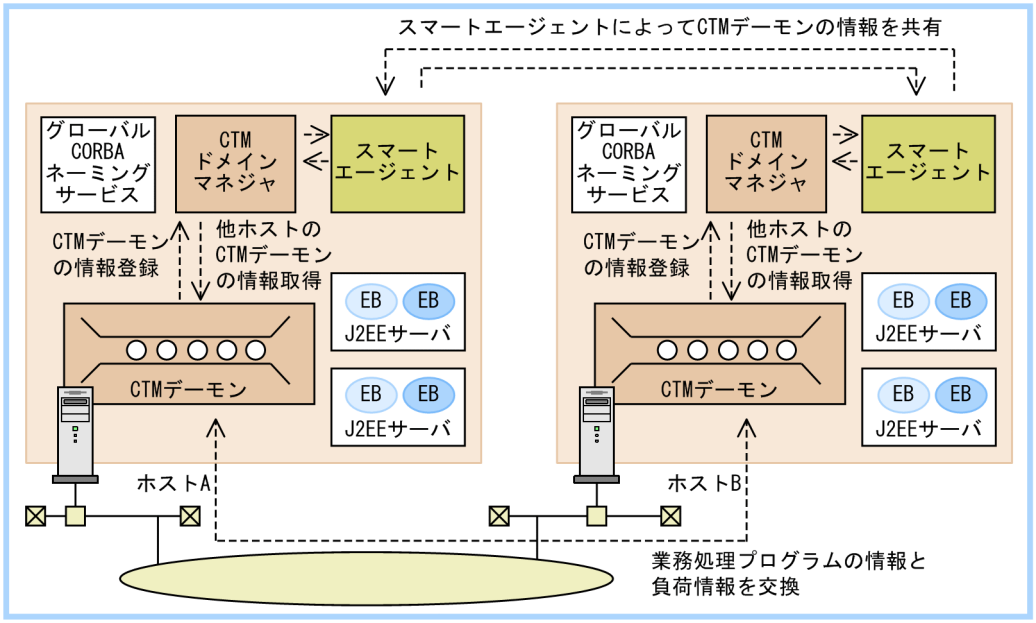
既存の CTM ドメインに新しく CTM デーモンを登録したい場合は，CTM ドメイン内のホスト上で，ほかの CTM ドメインマネジャと同じドメイン名称とポート番号を持つ CTM ドメインマネジャを開始するだけで参加できます。既存の CTM ドメインで環境の定義などを更新する必要がないので，システム環境をコピーするだけで，容易にシステムのスケールアウトができます。

## (2)　対象の CTM ドメインマネジャが異なるネットワークセグメントにある場合の情報共有

ブロードキャストされた情報はルータを越えられないため，異なるネットワークセグメントにある CTM ドメインマネジャには届きません。この場合には，スマートエージェントを使用して情報を配布する必要があります。

対象の CTM ドメインマネジャが異なるネットワークセグメントにある場合の情報共有について，次の図に示します。

図 3-11　異なるネットワークセグメントでの CTM ドメインマネジャによる情報共有




（凡例）

EB EB：業務処理プログラム

注　CTMレギュレータおよびPRFデーモンのプロセスは省略しています。

複数のネットワークセグメントで CTM ドメインを構成する場合に必要な設定は次のとおりです。

- CTM ドメインマネジャを開始するときに，情報共有の対象になる CTM ドメインマネジャのホスト名または IP アドレスを指定します。

  登録できるリクエストキューは，CTM ドメインマネジャを起動するときに ctmdmstart コマンドの引数-CTMSendHost に指定します。また，運用管理ポータルで構築したシステムを運用している場合は，論理 CTM ドメインマネジャにあらかじめ設定しておくことができます。

- スマートエージェントを，異なるネットワークセグメント上のスマートエージェントと接続するように設定します。

  スマートエージェントの設定については，マニュアル「Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド」を参照してください。

## (3)　CTM ドメインマネジャの部分再開始

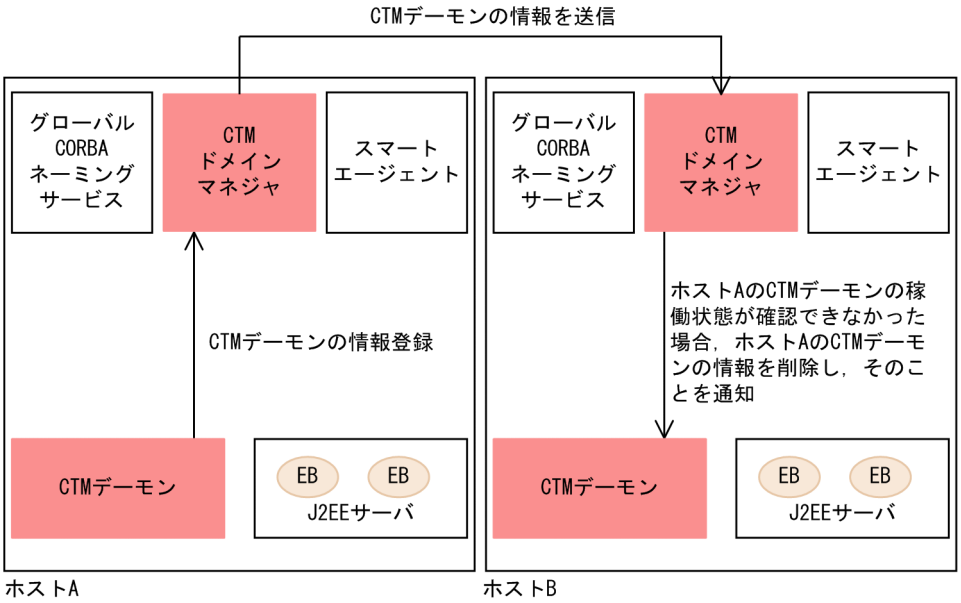
CTM ドメインマネジャが異常終了した場合，CTM ドメインマネジャだけを部分再開始できるときがあります。再開始できる障害かどうかは，CTM ドメインマネジャが再開始する時に，自動的に判断されます。部分再開始ができない場合は，システム全体が異常終了します。この場合は，システムを全面的に再開始してください。

### (4) CTM ドメインマネジャの稼働状態の確認

CTM ドメインマネジャは，ほかのホストの CTM ドメインマネジャが稼働しているかどうかを確認しています。このとき，稼働状態を確認する間隔に，任意の時間を指定できます。稼働状態の確認間隔の指定は，ctmdmstart コマンドの-CTMAliveCheckCount オプションで指定します。

なお，稼働状況の確認間隔で，CTM ノード情報が送信されなかった場合は，送信元の CTM ドメインマネジャが停止したと判断され，送信元に対する CTM の情報が削除されます。CTM ノード情報を削除した場合，その CTM デーモンへのリクエストの振り分けは実施されません。CTM ドメインマネジャの稼働状況確認について，次の図に示します。

図 3-12 CTM ドメインマネジャの稼働状況確認




ホスト B の CTM ドメインマネジャは，ホスト A の CTM ドメインマネジャからホスト A の CTM デーモンの情報を受信します。「CTM デーモンの情報の送信間隔に指定した値×生存判定監視係数」の間に CTM デーモンの情報が受信されない場合，ホスト A の CTM デーモンの情報を削除し，そのことをホスト B の CTM デーモンに通知します。これによって，ホスト B の CTM デーモンは，ホスト A の CTM デーモンにリクエストを振り分けなくなります。

## 3.3.6 グローバル CORBA ネーミングサービス

CTM を使用したリクエストのスケジューリングでは，ネーミングサービスとしてグローバル CORBA ネーミングサービスを使用します。

**グローバル CORBA ネーミングサービス**とは，同じ CTM ドメイン内に含まれるホスト上の業務処理プログラム（Stateless Session Bean）の情報を共有管理しているネーミングサービスのことです。グローバル CORBA ネーミングサービスでは，それぞれのホスト上に登録されている EJB ホームオブジェクトリファレンスの情報を，CTM ドメイン内で共有しています。これによって，リクエストを受け付けた CTM デーモンが処理対象としている J2EE サーバに目的の業務処理プログラムが登録されていない場合でも，

CTM ドメイン内のほかのホスト上に存在する CTM デーモンから，目的の業務処理プログラムが登録されている J2EE サーバを探せるようになり，適切な CTM デーモンにリクエストを振り分けられるようになります。

グローバル CORBA ネーミングサービスは，CTM デーモン一つに対して一つ配置します。CTM デーモンは，ほかの CTM デーモンと情報を交換した時に得たほかのホスト上にある業務処理プログラムの情報を，自ホスト内のグローバル CORBA ネーミングサービスに登録します。これによって，CTM ドメイン内で，グローバル CORBA ネーミングサービスの情報が共有，同期されます。このため，J2EE サーバを配置しないでグローバル CORBA ネーミングサービスだけを配置するホスト（統合ネーミングスケジューラサーバ）の場合も，ほかのホスト上で動作している J2EE サーバの情報を得るために，CTM デーモンを配置する必要があります。

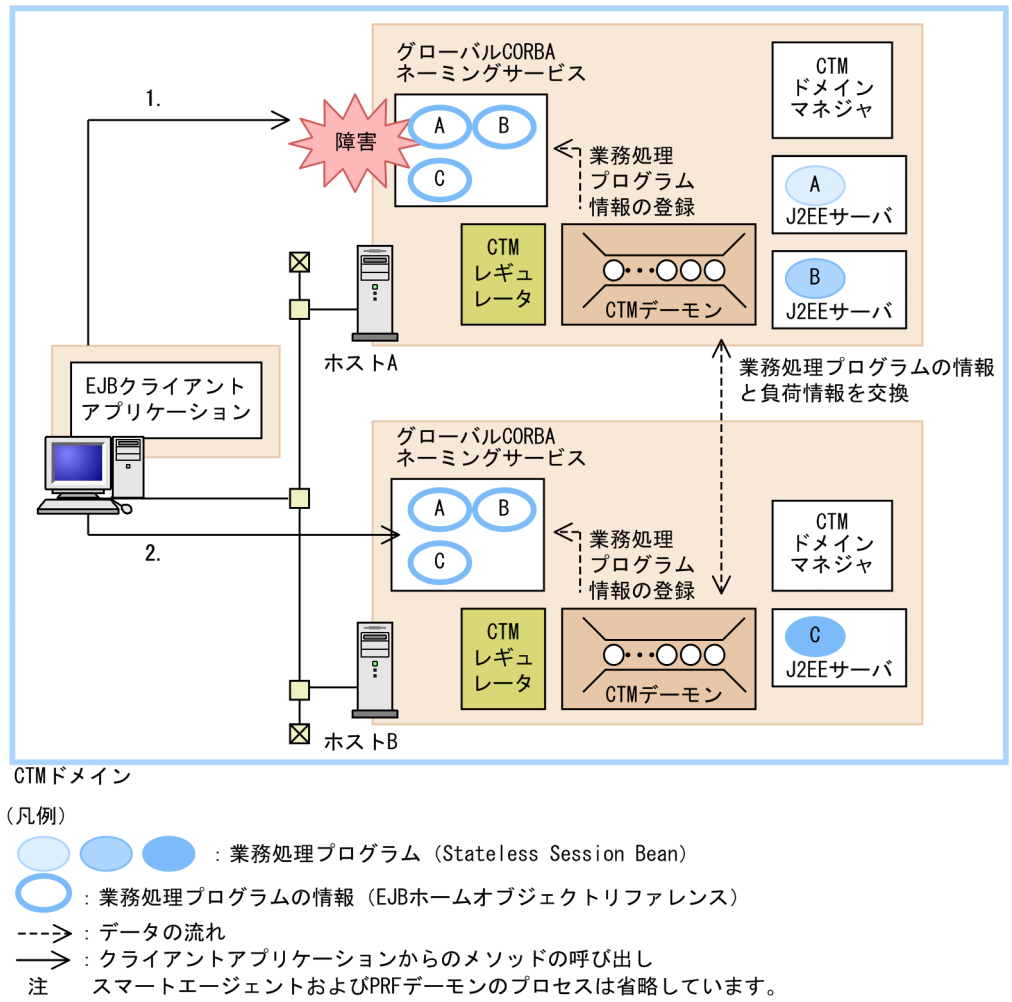
グローバル CORBA ネーミングサービスの特徴を次に示します。

- **障害の影響範囲を局所化することで，可用性を高められます。**
  CTM デーモンごとに一つずつ配置して，ドメイン内で情報を共有するので，どれかのホストのグローバル CORBA ネーミングサービスにトラブルが発生した場合には，ほかのホスト上のグローバル CORBA ネーミングサービスで対応できます。これによって，グローバル CORBA ネーミングサービスの障害の影響範囲を局所化して，システムの可用性を高められます。
- **lookup の対象にするネーミングサービスを，業務処理プログラムごとに選択する必要がありません。**
  クラスタ構成による負荷分散を実現する場合，CTM ドメイン内のすべてのグローバル CORBA ネーミングサービスに同じ業務処理プログラムの情報（EJB ホームオブジェクトリファレンスの情報）が登録されているので，実行する業務処理プログラムごとにルックアップするネーミングサービスを選択する必要がなくなります。これによって，特定のネーミングサービスに負荷を集中させることなく，適切な負荷分散が実現できます。

グローバル CORBA ネーミングサービスを使用した処理の流れの例を次の図に示します。

この例ではホスト A とホスト B の CTM デーモンが同じ CTM ドメインに登録されています。ホスト A の J2EE サーバには業務処理プログラム A と B が，ホスト B の J2EE サーバには業務処理プログラム C が登録されています。ただし，ホスト A では障害が発生しています。また，EJB クライアントアプリケーションは，システムプロパティ（java.naming.factory.initial キー）にラウンドロビン検索を実行する指定をして開始しています。

図 3-13　グローバル CORBA ネーミングサービスを使用した処理の流れの例




図について説明します。

1. EJB クライアントアプリケーションから業務処理プログラム C を実行するためには，まず，グローバル CORBA ネーミングサービスから EJB ホームオブジェクトリファレンスをルックアップする必要があります。この図では，ホスト A のグローバル CORBA ネーミングサービスを対象に lookup を実行しましたが，ホスト A では，障害が発生していたため，lookup に対して例外が発生します。
2. 特定のグローバル CORBA ネーミングサービスで障害が発生した場合に，EJB クライアントアプリケーションのシステムプロパティでラウンドロビン検索の実行が指定されていると，自動的に CTM ドメイン内のほかのグローバル CORBA ネーミングサービスに lookup の対象が切り替えられます。そこで，EJB クライアントアプリケーションから再度 lookup を実行すると，ホスト B のグローバル CORBA ネーミングサービスから業務処理プログラム C の EJB ホームオブジェクトリファレンスが取得できます。業務処理プログラム C はアプリケーションサーバ B 上にあるので，アプリケーションサーバ A の障害とは関係なく，処理を実行できます。

なお，アプリケーションサーバ A に障害が発生していなかった場合は，1.の lookup の結果，ホスト A のグローバル CORBA ネーミングサービスに登録されていたリファレンスが返却されます。そのリファレンスを使用して create を実行すると，ホスト A の CTM デーモンとホスト B の CTM デーモン間でリクエ

ストの振り分けが実行され，ホスト B にある業務処理プログラム C の EJB ホームオブジェクトリファレンスが返却されます。

### 注意事項

- グローバル CORBA ネーミングサービスが登録されているホストでトラブルが発生した場合は，そのホスト上のアプリケーションサーバ全体を再起動して，CTM デーモンからスケジュールキューのリファレンスを再度グローバル CORBA ネーミングサービスに登録し直す必要があります。
- リクエスト処理中に「CORBA::XXXX」という例外を標準出力，または標準エラー出力に表示することがあります。そのままの状態で動作し続ける場合には問題ありません。

## 3.4　リクエストの流量制御

**流量制御**では，J2EE サーバで一度に実行されるリクエスト処理数の最大値を制限することで，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実現します。

この節の構成を次の表に示します。

表 3-3　この節の構成（リクエストの流量制御）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | リクエストの流量制御の概要 | 3.4.1 |
| 設定 | 実行環境での設定 | 3.4.2 |

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。

### 3.4.1　リクエストの流量制御の概要

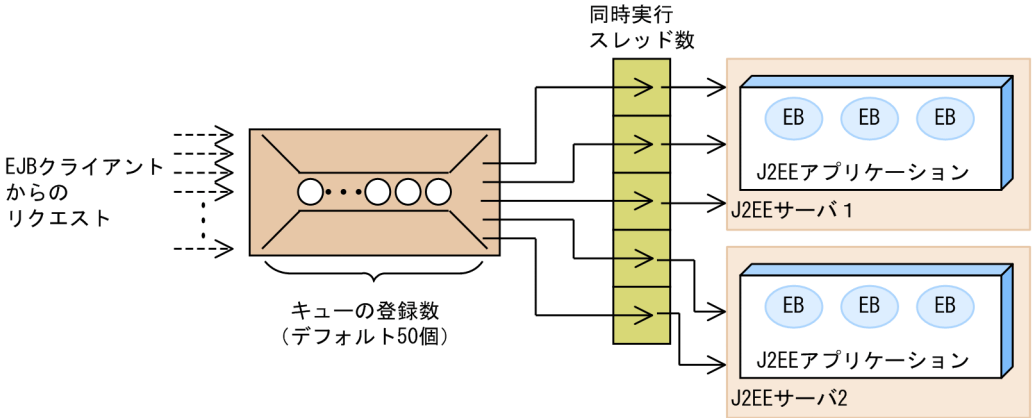
CTM を使用した流量制御について説明します。

**流量制御**は，J2EE サーバで一度に実行される処理数の最大値を設定して，リクエストの同時実行数を制限する機能です。これによって，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実現します。CPU や排他資源の競合も抑止できます。

CTM による流量制御は，CTM デーモンおよび CTM デーモンが管理しているスケジュールキューを使用して実現します。

CTM による流量制御の概要を，次の図に示します。この図は，J2EE アプリケーション単位でスケジュールキューを共有している場合の例です。

図 3-14　CTM による流量制御の概要




CTM デーモンでは，クライアントから受け付けたリクエストをスケジュールキューに登録して，スケジュールキュー単位に設定された同時実行スレッド数分ずつ実行します。クライアントからのリクエストが瞬間的に増加した場合でも，CTM デーモンによって流量が制御されるため，J2EE サーバで実行される

リクエストは同時実行スレッド数以上には増加しません。また，複数の J2EE サーバの J2EE アプリケーションで同じスケジュールキューを共有している場合は，その J2EE アプリケーション数および各 J2EE アプリケーションの同時実行スレッド数の設定で，一度に処理できる業務処理プログラムを多重化できます。リクエストは，スケジュールキューの最大リクエスト登録数分まで受け付けられます。最大リクエスト登録数は，スケジュールキュー単位で設定できます。なお，スケジュールキュー単位での設定がない場合は，CTM デーモン単位の設定がデフォルトとなります。これを超えると，エラーが返却されます。

なお，J2EE アプリケーションの同時実行数の制御は，EJB コンテナでも実行できます。EJB コンテナの同時実行数制御と CTM の流量制御の組み合わせによる効果は次のとおりです。

- ある J2EE サーバ上の EJB コンテナで同時実行数が上限に達した場合に，ほかの J2EE サーバにリクエストを転送できます。また，同時実行数が上限に達していなくても，該当する EJB コンテナの負荷が高い場合はほかの J2EE サーバにリクエストを転送できます。
- CTM のキューによって実行待ちのリクエスト数を一定に保てるので，それ以上のリクエストが EJB コンテナに送信された場合は，エラーを通知できます。
- EJB コンテナのインスタンスのプーリングと併用できます。

CTM が制御するスレッドの最大値およびキューごとのリクエストの登録数は，CTM デーモンを起動するときに，ctmstart コマンドの引数-CTMDispatchParallelCount および-CTMMaxRequestCount に指定します。また，運用管理ポータルで構築したシステムを運用している場合は，あらかじめ論理 CTM に設定しておくことができます。ctmstart 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「ctmstart（CTM デーモンの開始)」を参照してください。運用管理ポータル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 運用管理ポータル操作ガイド」の「10.7.2 スケジューリングの設定」を参照してください。

### 注意事項

- CTM による流量制御では，スケジュールキューごとに同時実行スレッド数（Parallel Count）を設定します。また，CTM に呼び出される Stateless Session Bean 単位に，プールしておくインスタンス数の最大値（Pooled Instances の Maximum）を指定できます。プールしておくインスタンス数の最大値がスケジュールキューの同時実行スレッド数よりも少ないと，CTM から呼び出された時にインスタンスが不足するおそれがあるのでご注意ください。
- CTM を使用する場合，グローバル CORBA ネーミングサービスのほかにそれぞれのホストの CORBA ネーミングサービス（ローカル CORBA ネーミングサービス）にも EJB オブジェクトリファレンスが登録されます。このため，アプリケーションの構成によっては，CTM を使用しないで直接ローカル CORBA ネーミングサービスに対して lookup を実行して Enterprise Bean を呼び出すこともできます。ただし，この場合，CTM からの同時実行スレッド数指定が保証されなくなります。このような使用方法はしないでください。

## 3.4.2　実行環境での設定

CTM の機能を使用する場合，CTM を使用する構成でシステムを構築する必要があります。CTM を使用するシステムの構成や構築手順については，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」，およびマニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。

CTM の機能を使用して，リクエストをスケジューリングするための設定は，簡易構築定義ファイルで ejbserver.ctm から始まるパラメタに，CTM デーモンの CTM 識別子，CTM キューの長さなどを指定します。

CTM によるリクエストのスケジューリングをするためには，次の設定が必要です。

- 実行環境ディレクトリの作成と環境変数の設定
- 簡易構築定義ファイルでの設定
- サーバ管理コマンドでの設定

### (1)　実行環境ディレクトリの作成と環境変数の設定

Management Server を使用しないでシステムを構築する場合，CTM を使用するためには，CTM とパフォーマンストレーサの実行環境ディレクトリを作成して，環境変数に指定する必要があります。

実行環境ディレクトリの作成および環境変数については，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 H　システムの環境変数」を参照してください。

なお，Management Server を使用してシステムを構築する場合は，CTM を使用するために必要な環境変数の設定はありません。

**注意事項**

AIX の場合は，環境変数の設定には次の点に注意してください。

- Component Transaction Monitor の実行環境では，環境変数 PSALLOC に「early」を設定してください。設定しない場合にメモリ不足が発生すると，正しい動作が保証できません。
- AIX の早期ページングスペース割り当てを指定する，環境変数 PSALLOC に「early」を指定しています。早期ページングスペース割り当てでは，ページングスペース見積もり上の考慮事項があります。詳細は，AIX のマニュアルの「システム・マネージメント・コンセプト：オペレーティング・システムおよびデバイス」を参照してください。
- Component Transaction Monitor の実行環境では，環境変数 NODISCLAIM に「true」を設定してください。環境変数 PSALLOC に「early」を設定した場合，環境変数 NODISCLAIM に「true」を設定しないと，レスポンス，スループットおよび CPU 利用率が極端に低下することがあります。

- Component Transaction Monitor で使用するユーザデータ領域と共用メモリ領域を拡張するため，環境変数 LDR_CNTRL に「MAXDATA=0x40000000」を設定してください。割り当てるメモリの値を 1 ギガバイトにしてください。
- Component Transaction Monitor の実行環境では，環境変数 EXTSHM に「ON」を設定してください。設定しない場合，共有メモリが参照できなくなることがあります。

## (2) 簡易構築定義ファイルでの設定

CTM を使用してリクエストをスケジューリングする場合には，簡易構築定義ファイルで論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，次のパラメタを設定してください。簡易構築定義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(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。

- ejbserver.ctm.ActivateTimeOut

  CTM を使用する J2EE アプリケーションの開始時に，J2EE サーバがスケジュールキューを活性化するときの待ち時間を指定します。
- ejbserver.ctm.DeactivateTimeOut

  CTM を使用する J2EE アプリケーションの停止時に，J2EE サーバがスケジュールキューを非活性化するときの待ち時間（実行中のリクエストの完了待ち時間）を指定します。
- ejbserver.ctm.QueueLength

  CTM を使用する J2EE アプリケーションの開始時に，J2EE サーバによって生成される CTM キューの長さを指定します。
- ejbserver.client.ctm.RequestPriority

  J2EE サーバから CTM に送信するリクエストの優先度を指定します。

## (3) サーバ管理コマンドでの設定

サーバ管理コマンドで設定できる内容を次に示します。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。

- J2EE アプリケーション単位での設定

  アプリケーション属性ファイルで次の設定ができます。
  - <managed-by-ctm>タグで，CTM を利用するかどうかを設定できます。
  - <scheduling>タグで，スケジューリングをするキューの名称や長さなどを設定できます。
  - <scheduling-unit>タグで，スケジュールキューの配置単位として，J2EE アプリケーション単位または Bean 単位のどちらかを選択できます。
- Stateless Session Bean 単位での設定

  SessionBean 属性ファイルで次の設定ができます。

- <enable-scheduling>タグで，J2EE アプリケーションに含まれるどの Stateless Session Bean をスケジューリングの対象にするかを設定できます。
- <stateless><pooled-instance>タグ下の<maximum>タグまたは<minimum>タグで，プールしておくインスタンス数の最大値または最小値を設定できます。なお，運用時に CTM の同時実行数を動的に変更する場合は，最大値は無制限「0」に設定する必要があります。
- <scheduling>タグで，スケジューリングをするキューの名称や長さなどを設定できます。

cjgetappprop コマンドで属性ファイルを取得し，属性ファイル編集後に，cjsetappprop コマンドで編集内容を J2EE アプリケーションに反映させてください。

## 3.5　リクエストの優先制御

ここでは，CTM によるリクエストの優先制御について説明します。

CTM を経由するリクエストには優先順位を付けることができます。EJB クライアントに優先順位を付けておくと，優先順位の高い EJB クライアントからのリクエストは，優先順位の低い EJB クライアントからのリクエストより先にキューから取り出され，処理されます。

リクエストの優先順位は，EJB クライアントとして動作する J2EE サーバ，Web コンテナサーバまたは EJB クライアントアプリケーションのプロパティとして設定します。CTM では，優先順位に小さな値が設定されている EJB クライアントから送信されてきたリクエストを，優先的に処理します。

## 3.6　リクエストの同時実行数の動的変更

リクエストの同時実行数の動的変更では，CTM を使用して流量制御をしている場合に，CTM デーモンを停止しないでスケジュールキュー単位のリクエストの同時実行数を動的に変更できます。これによって，スケジュールキューが対応するサービスの内容に応じて，同時実行数を一時的に増加させたり，減少させたりできます。

この節の構成を次の表に示します。

表 3-4　この節の構成（リクエストの同時実行数の動的変更）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | 動的変更の処理の仕組み | 3.6.1 |
| 設定 | 同時実行数に指定できる値 | 3.6.2 |
| 運用 | CTM のスケジュールキューの稼働状況の確認 | 3.6.3 |
| | CTM のスケジュールキューの同時実行数の変更 | 3.6.4 |

注　「実装」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。

CTM の同時実行数の動的変更は，ctmchpara コマンドで実行します。スケジュールキューの同時実行数の変更については，「3.6.4　CTM のスケジュールキューの同時実行数の変更」を参照してください。ctmchpara 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「ctmchpara（スケジュールキューの同時実行数の変更）」を参照してください。

**ポイント**

ctmchpara コマンドで変更したスケジュールキュー単位の同時実行数は，CTM デーモンを停止するまで有効です。個別の J2EE アプリケーションに設定した parallel count の値として保存はされません。一度 CTM デーモンを再起動して J2EE アプリケーションを再開始する場合には，個別の J2EE アプリケーションに設定した parallel count の値が有効になります。

### 3.6.1　動的変更の処理の仕組み

CTM による同時実行スレッド数の動的変更の概要を，次の図に示します。

図 3-15　CTM による同時実行数の動的変更の概要

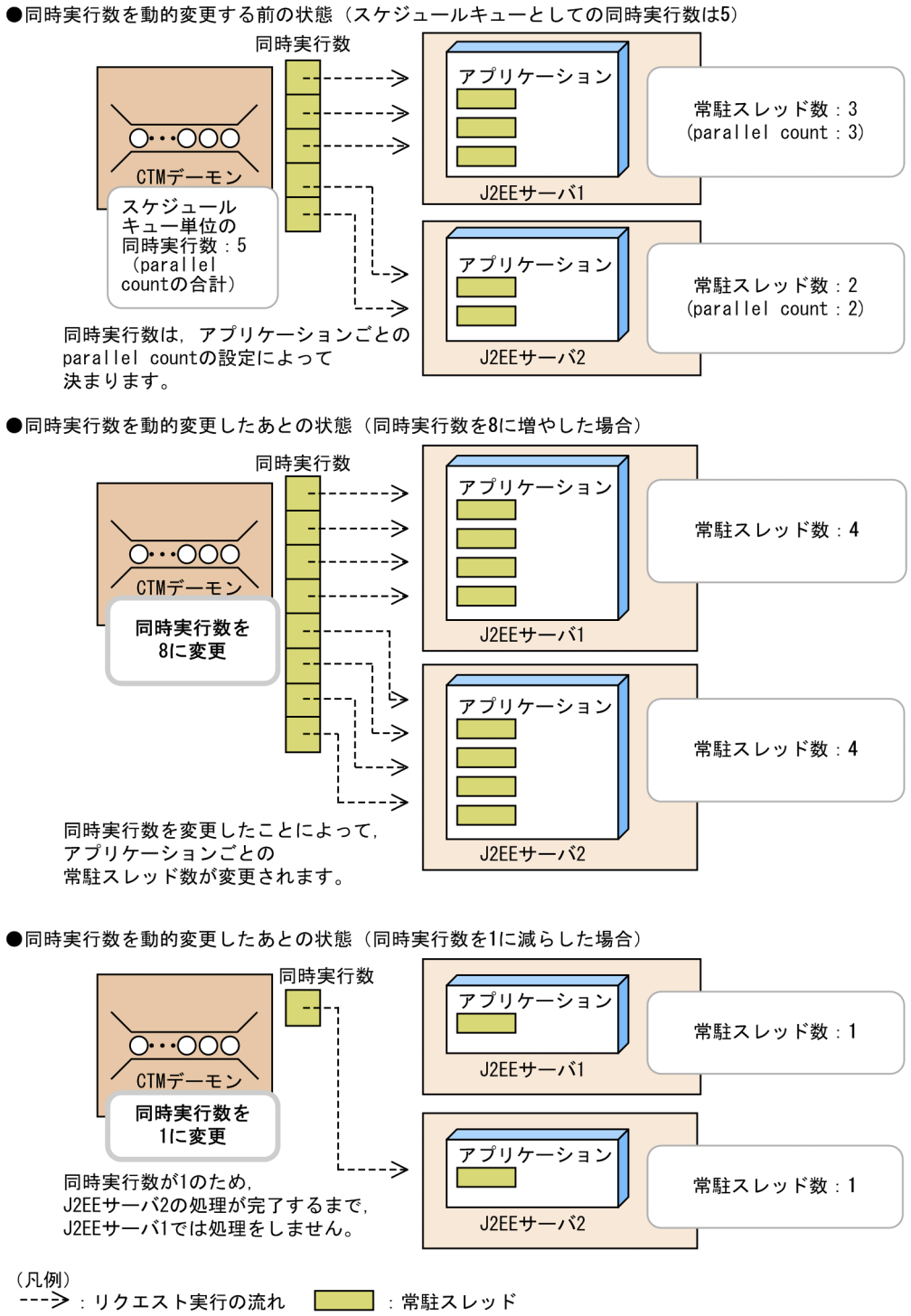



図 3-15 について，(1)～(3)で説明します。

## (1)　同時実行数を動的変更する前の状態（スケジュールキューとしての同時実行数は 5）

同時実行数を動的変更する前の，J2EE サーバ起動時の状態です。CTM デーモンのスケジュールキューは，J2EE サーバ 1 のアプリケーションと J2EE サーバ 2 の J2EE アプリケーションで共有されています。

J2EE サーバ 1 の J2EE アプリケーションでは，Stateless Session Bean の属性として，同時実行数（parallel count）が 3 に設定されています。J2EE サーバ 2 の J2EE アプリケーションでは，Stateless Session Bean の属性として，同時実行数（parallel count）が 2 に設定されています。この場合，スケジュールキュー単位の同時実行数は，それぞれの J2EE アプリケーションの同時実行数を足した 5 になります。

それぞれの J2EE サーバでは，リクエスト処理要求を受けると，必要に応じてリクエスト処理用のスレッドを生成します。最大で J2EE アプリケーションの parallel count に設定した数分のスレッドが生成されます。生成されたスレッドは，削除されないでそのまま常駐スレッドになります。

なお，parallel count は，サーバ管理コマンドで設定，変更できます。

## (2) 同時実行数を動的変更したあとの状態（同時実行数を 8 に増やした場合）

スケジュールキュー単位の同時実行数を動的に 8 に増やした場合について説明します。

スケジュールキュー単位の同時実行数を動的に増やすと，変更後の同時実行数に合わせて，それぞれの J2EE アプリケーションのリクエスト処理用の常駐スレッド数も変更されます。

なお，常駐スレッド数が変更されるときには，スケジュールキューを共有している J2EE アプリケーションで，**常駐スレッド数の平均化**が実施されます。例えば，三つの J2EE サーバ上の J2EE アプリケーションの parallel count がそれぞれ 40，30，60 で，それぞれ常駐スレッドが最大数作成されている場合に，スケジュールキュー単位の同時実行数を 120 に変更しようとすると，それぞれの J2EE サーバ上の常駐スレッド数は，「120（同時実行数）÷3（スケジュール共有数）」で平均化されて，40 になります。

図 3-15 の場合は，同時実行数 8 を二つの J2EE サーバで処理するので，常駐スレッド数がそれぞれ 4 ずつに変更されます。

## (3) 同時実行数を動的に変更したあとの状態（同時実行数を 1 に減らした場合）

スケジュールキュー単位の同時実行数を 1 に減らした場合について説明します。

同時実行数を減らす場合も，変更後の同時実行数に合わせて，それぞれの J2EE アプリケーションのリクエスト処理用の常駐スレッド数が変更されます。この場合も，常駐スレッド数は平均化されます。

ただし，スケジュールキュー単位の同時実行数を，スケジュールキューを共有している J2EE アプリケーションの数よりも小さくした場合，単純に常駐スレッド数を平均化すると，リクエストを受け付けない J2EE サーバが出てしまいます。これを防ぐため，常駐スレッドは最低で 1 個は確保されます。

図 3-15 の場合，同時実行数 1 を二つの J2EE サーバで処理するので，それぞれの J2EE サーバの常駐処理スレッドは，最低保障常駐スレッド数である 1 になります。ただし，この場合も，同時に処理されるリクエストの数は，同時実行数分だけになります。つまり，J2EE サーバ 2 での処理が完了するまで，J2EE サーバ 1 のスレッドでは処理が実行されません。

## 3.6.2　同時実行数に指定できる値

同時実行数を動的変更するときに，指定できる値について説明します。

同時実行数は，1～「スケジュールキューを共有している J2EE アプリケーションの数×127」までの整数で指定できます。127 は，一つの J2EE アプリケーションで処理できる同時実行数（parallel count）の最大数です。

ただし，CTM デーモンを起動するときに-CTMDispatchParallelCount に指定した値を超える値は指定できません。

次の値を指定した場合は，エラーが出力されて，同時実行数は変更されません。

- 0
- 「スケジュールキューを共有している J2EE アプリケーションの数×127」を超える値
- ctmstart コマンドの-CTMDispatchParallelCount の指定値を超える値

## 3.6.3　CTM のスケジュールキューの稼働状況の確認

ここでは，CTM のスケジュールキューの稼働状況を確認する方法について説明します。CTM のスケジュールキューの稼働状況は運用管理コマンド（mngsvrutil）を使用して確認できます。

CTM のスケジュールキューの稼働状況を確認するには，運用管理コマンドのサブコマンド「get」の引数に，「queueApps」を指定して実行します。コマンドを実行すると，J2EE アプリケーション開始時の同時実行数，J2EE アプリケーションに対する現在の常駐スレッド数などの情報が取得できます。

次に，実行形式と実行例を示します。

**実行形式**

```
mngsvrutil -m <Management Serverのホスト名> [:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <論理サーバ名> get queueApps
```

**実行例**

```
mngsvrutil -m mnghost -u user01 -p pw1 -t myServer get queueApps
```

mngsvrutil コマンド，そのサブコマンド，および取得できる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 リファレンス コマンド編」の「mngsvrutil（Management Server の運用管理コマンド）」を参照してください。

## 3.6.4　CTM のスケジュールキューの同時実行数の変更

J2EE アプリケーションの同時実行数をスケジュールキュー単位で動的に変更する方法について説明します。

J2EE アプリケーションの最大同時実行数を CTM スケジュールキュー単位で動的に変更する作業は次の流れで行います。

1. 現在の CTM のスケジュールキューの同時実行数を確認する
   CTM のコマンド（ctmlsque）を使用して実行します（「(1)　CTM のスケジュールキューの稼働状況の確認」参照）。
2. CTM のスケジュールキューの同時実行数を変更する
   CTM のコマンド（ctmchpara）を使用して実行します（「(2)　CTM のスケジュールキューの同時実行数の変更」参照）。
3. 変更後の CTM のスケジュールキューの同時実行数を確認する
   CTM のコマンド（ctmlsque）を使用して実行します（「(1)　CTM のスケジュールキューの稼働状況の確認」参照）。

なお，CTM のスケジュールキューの同時実行数の変更は，スケジュールキューの状態が次の場合に実行できます。

- A：スケジューリング可能状態
- H：スケジューリング閉塞中
- C：スケジューリング可能閉塞

## (1)　CTM のスケジュールキューの稼働状況の確認

CTM のスケジュールキューの稼働状況を確認するには，ctmlsque コマンドの引数に「-CTMAppInfo」を指定して実行します。このコマンドを実行すると，スケジュールキューを共有している J2EE アプリケーションの情報を確認できます。実行形式と実行例を次に示します。

**実行形式**

```
ctmlsque -CTMDomain 〈CTMドメイン名称〉 -CTMID 〈CTM識別子〉 -CTMAppInfo
```

**実行例**

```
ctmlsque -CTMDomain domain01 -CTMID CTM01 -CTMAppInfo
```

ctmlsque コマンドの詳細，および出力される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 リファレンス コマンド編」の「ctmlsque（スケジュールキュー情報の出力）」を参照してください。

## (2)　CTM のスケジュールキューの同時実行数の変更

CTM のスケジュールキューの同時実行数を変更するには，ctmchpara コマンドを実行します。実行形式と実行例を次に示します。

実行形式

```
ctmchpara -CTMDomain 〈CTMドメイン名称〉-CTMID 〈CTM識別子〉-CTMQueue 〈スケジュールキュー登録名称〉-CTMChangeCount 〈同時実行数〉
```

実行例

```
ctmchpara -CTMDomain domain01 -CTMID CTM01-CTMQueue que01 -CTMChangeCount 10
```

実行後，変更が反映されていることを確認してください。スケジュールキューの状態を確認する方法については，「(1)　CTM のスケジュールキューの稼働状況の確認」を参照してください。

ctmchpara コマンドの詳細，および出力される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 リファレンス コマンド編」の「ctmchpara（スケジュールキューの同時実行数の変更)」を参照してください。

## 3.7　リクエストの閉塞制御

**閉塞制御（サービス閉塞）**は，特定の J2EE アプリケーションに対するリクエストの受け付けを停止したり，リクエストを滞留させたりすることで，システム全体を停止させないで J2EE アプリケーションの入れ替えや再起動を可能にして，システムの可用性を高めるための機能です。

この節の構成を次の表に示します。

表 3-5　この節の構成（リクエストの閉塞制御）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | リクエストの閉塞制御の概要 | 3.7.1 |
| | オンライン状態での J2EE アプリケーションの入れ替え | 3.7.2 |
| | J2EE アプリケーションの閉塞制御 | 3.7.3 |
| | スケジュールキューの閉塞制御 | 3.7.4 |
| | J2EE サーバ異常終了時のリクエスト保持 | 3.7.5 |
| 設定 | 実行環境での設定 | 3.7.6 |

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。

### 3.7.1　リクエストの閉塞制御の概要

CTM を使用してリクエストをスケジューリングしている場合，特定のスケジュールキューの閉塞制御ができます。スケジュールキューを閉塞制御することによって，特定の J2EE アプリケーションをオンライン状態で入れ替えるサービス閉塞などができるようになります。

CTM による閉塞制御でできることは，次のとおりです。

- **オンライン状態での J2EE アプリケーションの入れ替え**
  スケジュールキューにリクエストを保持した状態で J2EE アプリケーションを入れ替えられます。
- **J2EE アプリケーションの閉塞制御**
  リクエストの完了を待ってから，J2EE アプリケーションを閉塞させます。
- **スケジュールキューの閉塞制御**
  スケジュールキューをすぐに閉塞させます。キューに登録済みのリクエストを破棄するかどうかを選択できます。
- **J2EE サーバ異常終了時のリクエスト保持**
  J2EE サーバ異常終了時にスケジュールキューのリクエストを一定時間保持します。

リクエストの閉塞制御は，運用管理コマンド（mngsvrutil）で実行します。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「mngsvrutil（Management Server の運用管理コマンド）」を参照してください。

## 3.7.2　オンライン状態での J2EE アプリケーションの入れ替え

J2EE アプリケーションを入れ替える場合に，オンライン状態で J2EE アプリケーションを入れ替えられます。

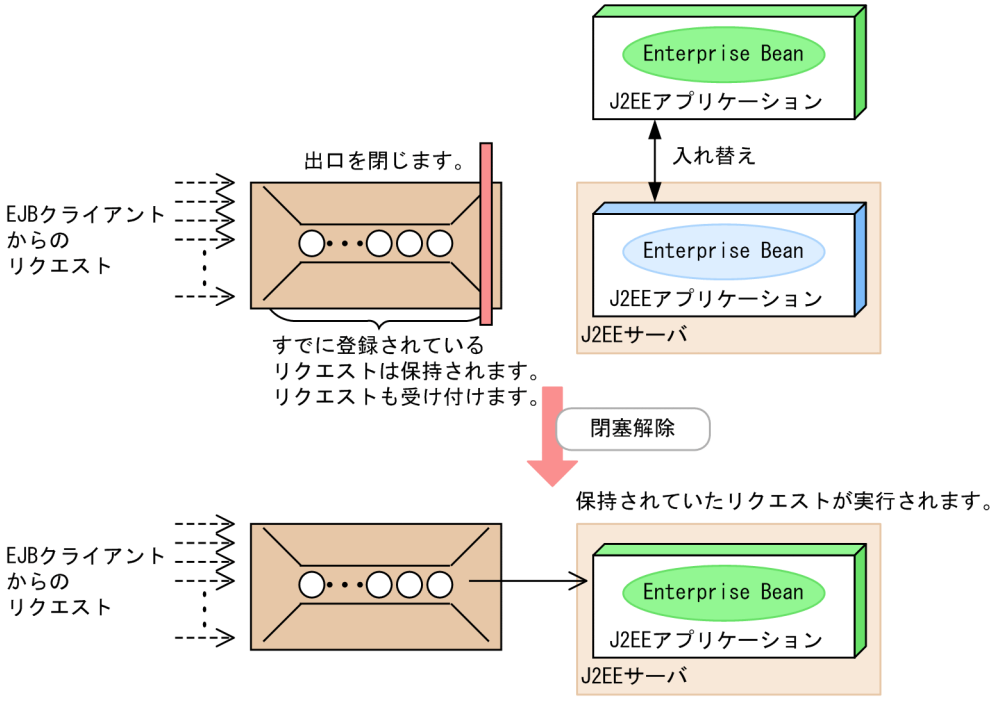
ここでは，入れ替えの概要と入れ替え手順について説明します。

### (1)　入れ替えの概要

J2EE アプリケーションを入れ替える場合には，CTM デーモンによって，スケジュールキューの出口を閉じてから，入れ替えを実行します。出口を閉じている間もクライアントからのリクエストはスケジュールキューに登録し続けることができます。このため，該当アプリケーションに対するリクエストもエラーにしないでシステムを運用し続けられます。ただし，スケジュールキューの最大リクエスト登録数を超えた場合は，クライアントにエラーが返却されます。

オンライン状態での J2EE アプリケーションの入れ替えの概要を，次の図に示します。

図 3-16　オンライン状態での J2EE アプリケーションの入れ替えの概要



## (2) 入れ替えの手順

オンライン状態で J2EE アプリケーションを入れ替える場合，J2EE アプリケーションのスケジュールキューの出口を閉じたあと，入れ替えを実行します。この操作は，運用管理コマンド（mngsvrutil）で実行できます。

J2EE アプリケーションの入れ替えは，J2EE アプリケーション単位，ホスト単位，または運用管理ドメイン単位で実行できます。

スケジュールキューの出口を閉じるには，mngsvrutil コマンドにサブコマンド「hold」を指定して実行します。mngsvrutil コマンドを実行してスケジュールキューの出口を閉塞している間も，クライアントからのリクエストはスケジュールキューに登録し続けられます。ただし，スケジュールキューの最大リクエスト登録数を超えた場合，クライアントにエラーが返却されます。

J2EE アプリケーションの入れ替えが終了したら，スケジュールキューの閉塞を解除します。スケジュールキューの閉塞解除は，mngsvrutil コマンドにサブコマンド「release」を指定して実行します。スケジュールキューの閉塞を解除すると，J2EE アプリケーションで，スケジュールキューに保持されていたリクエストの処理を再開します。

CTM を使用したオンライン状態での J2EE アプリケーションの入れ替え手順を次に示します。

1. 入れ替える J2EE アプリケーションに対応する CTM のスケジュールキューの出口を閉じます。

   J2EE アプリケーションを入れ替える場合の mngsvrutil コマンドの実行形式と実行例を次に示します。

   実行形式

   ```
   mngsvrutil -m <Management Serverのホスト名>[:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <CTMの名称> hold queue <キューの名称> out
   ```

   実行例

   ```
   mngsvrutil -m mnghost -u user01 -p pw1 -t ctm01 hold queue App1 out
   ```

2. J2EE アプリケーションを入れ替えます。

   J2EE アプリケーションを停止して，新しいアプリケーションに入れ替えます。そのあとで，J2EE アプリケーションを再開始します。

   J2EE アプリケーションの入れ替え方法については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「5.6.3　J2EE アプリケーションの入れ替えと保守」を参照してください。

3. CTM のスケジュールキューの閉塞を解除するときは，mngsvrutil コマンドにサブコマンド「release」を指定して実行します。

   この場合の mngsvrutil コマンドの実行形式と実行例を次に示します。

   実行形式

   ```
   mngsvrutil -m <Management Serverのホスト名>[:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <CTMの名称> release queue <キューの名称>
   ```

実行例

```
mngsvrutil -m mnghost -u user01 -p pw1 -t ctm01 release queue App1
```

mngsvrutil 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「mngsvrutil（Management Server の運用管理コマンド)」を参照してください。

## 3.7.3　J2EE アプリケーションの閉塞制御

J2EE アプリケーションを停止するときに，スケジュールキューに登録されたリクエストの処理が完了するのを待ってから，J2EE アプリケーションを停止できます。これによって，停止する J2EE アプリケーションがそのキューを共有する最後のアプリケーションであった場合に，すでに受け付けられたリクエストをエラーにしないで処理できます。

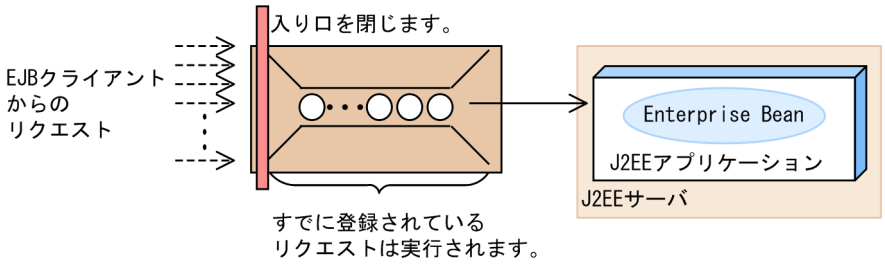
ここでは，閉塞制御の概要と閉塞の手順について説明します。

### (1)　閉塞制御の概要

キューを共有する最後の J2EE アプリケーションが停止する場合，CTM デーモンは，スケジュールキューの入り口を閉じてサービスを停止して，それ以上リクエストを受け付けないようにします。そのあと，スケジュールキューに登録されていたリクエストの処理がすべて完了するのを待ってから，J2EE アプリケーションを停止します。

J2EE アプリケーションの閉塞制御の概要を次の図に示します。

図 3-17　J2EE アプリケーションの閉塞制御の概要




CTM の閉塞処理では，次の作業が実行されます。

- 新規リクエストの受け付けが終了されます。
- スケジュールキューに格納されているリクエストのうち，すでに J2EE サーバに振り分けられて処理中のリクエストは引き続き処理されます。
- スケジュールキューに格納されているリクエストのうち，J2EE サーバへの振り分けがされていないリクエストに対しては，java.rmi.RemoteException エラーが返却されます。

### (2) 閉塞の手順

閉塞は，運用管理コマンドで実行します。

特定のホスト内の J2EE アプリケーションを一括停止する場合の運用管理コマンドの実行形式と実行例を次に示します。なお，運用管理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「mngsvrutil（Management Server の運用管理コマンド)」を参照してください。

実行形式

```
mngsvrutil -m <Management Serverのホスト名> [:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <ホスト名> -k host hold queues in:<リクエスト終了待ち時間（秒）>
```

実行例

- サービス閉塞をして，すべてのリクエスト処理の完了を待つ場合

  ```
  mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues in:0
  ```

- サービス閉塞をして，5 分間リクエストの処理を続けて，終了しないリクエストは破棄する場合

  ```
  mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues in:300
  ```

- サービス閉塞をして，リクエストはすぐに破棄する場合

  ```
  mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues in:-1
  ```

スケジュールキューの閉塞を解除するときは，mngsvrutil コマンドにサブコマンド「release」を指定して実行します。mngsvrutil コマンドの実行形式と実行例を次に示します。

実行形式

```
mngsvrutil -m <Management Serverのホスト名> [:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <ホスト名> -k host release queues
```

実行例

```
mngsvrutil -m mnghost01 -u user01 -p pw1 -t host01 -k host release queues
```

## 3.7.4 スケジュールキューの閉塞制御

スケジュールキューの閉塞制御には，次の 2 種類があります。

- 強制閉塞
- タイムアウト閉塞

ここでは，スケジュールキューの閉塞制御の概要について説明します。また，強制閉塞およびタイムアウト閉塞の実行手順についても説明します。

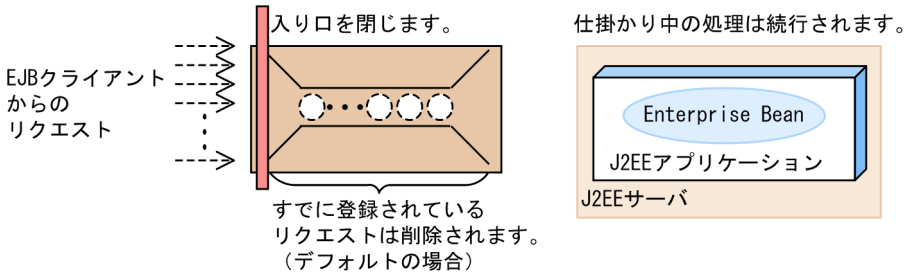
## (1) スケジュールキューの閉塞制御の概要

スケジュールキューに対して，直接閉塞を実行することもできます。これによって，複数の J2EE アプリケーションでスケジュールキューが共有されている場合に，一度に J2EE アプリケーションを停止できます。スケジュールキューに登録されている仕掛かり中のリクエストについては，破棄するか一定の時間処理を続けるかを選択できます。処理を続ける場合は，一定の時間内に処理ができなければ強制的に破棄するように，タイムアウト時間が指定できます。また，仕掛かり中のリクエストについては，処理が続行されます。

スケジュールキューの閉塞が指示されると，CTM デーモンは，スケジュールキューの入り口を閉じてサービスを停止して，それ以降のリクエストを受け付けないようにします。また，すでにスケジュールキューに登録されたリクエストは，設定に従って，破棄するか，または処理を実行してからスケジュールキューの閉塞を完了します。リクエストを破棄する場合は，キューに登録されていたリクエストの処理はすべてエラーとしてクライアントに返却されます。処理を実行してから閉塞する場合は，一定時間処理を継続して，時間内に終了しなかった処理がエラーで返却されます。

スケジュールキューの閉塞制御の概要を，次の図に示します。

図 3-18　スケジュールキューの閉塞制御の概要




CTM を使用しているバックシステムで，ホスト内の J2EE アプリケーションを一度に停止したり，キューを共有する J2EE アプリケーションを一度に停止したりする場合，J2EE アプリケーションのスケジュールキューに対して直接閉塞を実行します。そのあと，J2EE アプリケーションを停止します。

運用管理コマンドを使用して J2EE アプリケーションのスケジュールキューを直接閉塞する場合，スケジュールキューを共有する J2EE アプリケーション単位，ホスト単位，または運用管理ドメイン単位で J2EE アプリケーションを一度に停止できます。また，スケジュールキューに登録済みのリクエストを破棄するか，一定時間処理を続けるかどうかを選択します。スケジュールキューに登録済みのリクエストを破棄する場合，登録済みのリクエストはすべてエラーとしてクライアントに返却されます。一定時間処理を続ける場合，時間内に終了しなかった処理はエラーとしてクライアントに返却されます。

運用管理コマンドを使用した場合の CTM を使用したサービス閉塞の実行形式および実行例について説明します。ここでは，通常の手順で閉塞する方法と，強制的に閉塞する方法について説明します。強制閉塞は，CTM デーモンの負荷が高い場合などに，すぐにキューを閉塞したいときに実行する方法です。

## (2)　スケジュールキューの強制閉塞

スケジュールキューは，CTM デーモンとの通信をしないで閉塞することもできます。これを，**スケジュールキューの強制閉塞**といいます。強制閉塞は，CTM デーモンの負荷が高いときにすぐにキューを閉塞するための方法です。通常の閉塞方法では，キューを閉塞するときに，CTM デーモンと通信して，その処理の延長で滞留しているリクエストを破棄しています。しかし，この方法では，CTM デーモンの負荷が高い場合，通信処理に時間が掛かるため，閉塞処理にも時間が掛かってしまいます。

強制閉塞を使用すると，CTM デーモンとの通信処理をしないで，即座にキューを閉塞できます。なお，強制閉塞を使用した場合，滞留しているリクエストの破棄は，CTM デーモン間で負荷情報を監視するタイミングにあわせて実行されます。

強制閉塞をする場合は，運用管理コマンド（mngsvrutil）のサブコマンド「hold」の引数に，「queue force」を指定してください。スケジュールキューの強制閉塞をすると，滞留しているリクエストは一定時間後に破棄されます。滞留するリクエストを破棄したくない場合は，ctmholdque コマンドで-CTMRequestLeave オプションも指定してください。

なお，閉塞の解除方法については，通常の閉塞をした場合と同じです。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「mngsvrutil（Management Server の運用管理コマンド)」を参照してください。

強制閉塞を実行する場合の mngsvrutil コマンドの実行例を次に示します。

**実行形式**

```
mngsvrutil -m <Management Serverのホスト名> [:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <ホスト名> -k host hold queues force
```

**実行例**

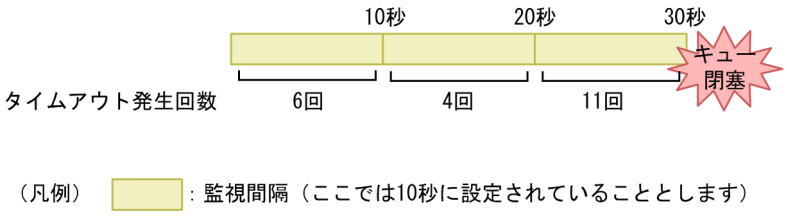
```
mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues force
```

## (3)　スケジュールキューのタイムアウト閉塞

スケジュールキューでは，EJB クライアントのタイムアウトを一定間隔で監視し，タイムアウトの発生回数が，設定した回数を超えると，スケジュールキューを閉塞します。これを，**スケジュールキューのタイムアウト閉塞**といいます。

タイムアウト発生について次の図で説明します。

図 3-19　スケジュールキューのタイムアウト閉塞の発生



図では，10 秒間隔でタイムアウト発生回数を監視しています。タイムアウト回数のカウントは，監視時間内だけとなります。次の監視時間ではタイムアウトの回数はリセットしてカウントされます。

このとき，例えば，タイムアウト発生回数として 10 回が設定されている場合，10 秒の監視時間内で 10 回以上タイムアウトが発生すると，キューが閉塞されます。なお，キューが閉塞されるタイミングは，タイムアウト回数を 10 回以上検知したあとの，次の監視時間で閉塞されます。この図の場合，監視を始めてから 30 秒後に 11 回のタイムアウトを検知したので，タイムアウトを検知した 30 秒後にキューが閉塞されます。

なお，スケジュールキューのタイムアウト閉塞は，CTM デーモン起動時にオプションを指定することで実行します。ctmstart コマンドの-CTMWatchRequest オプションで設定します。

## 3.7.5　J2EE サーバ異常終了時のリクエスト保持

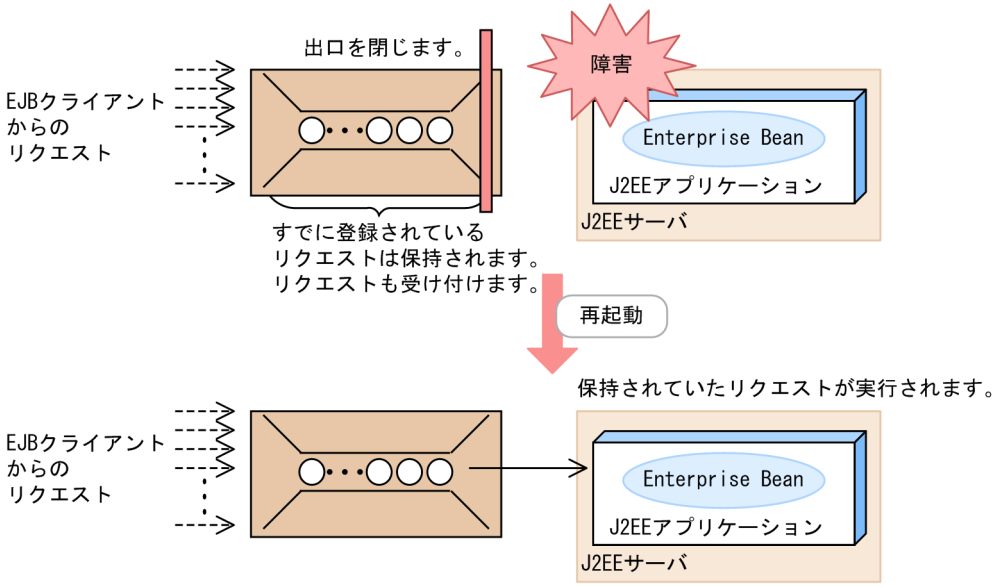
J2EE サーバ異常終了時に，スケジュールキューのリクエストを一定時間保持します。

これによって，J2EE サーバが異常終了した場合でも，すぐにユーザにエラーは返却されません。さらに，J2EE サーバが再起動するまでの間，クライアントからのリクエストは受け付け続けます。リクエストは，スケジュールキューの最大リクエスト登録数分まで受け付けられます。このため，J2EE サーバに障害が発生した場合でも，すぐに再起動すれば，クライアントに障害を気づかせないで運用を続けられます。ただし，スケジュールキューに登録できるリクエストの最大値を超えた場合は，クライアントにエラーが返却されます。

この機能は，ctmstart コマンドの-CTMQueueDeleteWait オプションで設定します。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「ctmstart（CTM デーモンの開始)」を参照してください。

J2EE サーバ異常終了時のリクエスト保持の概要を，次の図に示します。

図 3-20　J2EE サーバ異常終了時のリクエスト保持の概要



## 3.7.6　実行環境での設定

閉塞制御のうち，スケジュールキューのタイムアウト閉塞をする場合，CTM デーモンの設定が必要です。

CTM デーモンの設定は，簡易構築定義ファイルで実施します。リクエストの負荷分散の定義は，簡易構築定義ファイルの論理 CTM（component-transaction-monitor）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのスケジュールキューのタイムアウト閉塞の定義について次の表に示します。

### 表 3-6　簡易構築定義ファイルでのスケジュールキューのタイムアウト閉塞の定義

| 指定するパラメタ | 設定内容 |
|---|---|
| ctm.RequestCount | 何回タイムアウトが発生したら自動閉塞するか指定します。 |
| ctm.RequestInterval | タイムアウト発生回数を求める時間間隔を指定します。 |
| ctm.WatchRequest | J2EE サーバへのリクエストの送信でタイムアウトが発生したときにキューを閉塞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。

## 3.8　リクエストの負荷分散

**負荷分散**は，クラスタ構成などで並列に運用している J2EE サーバ間で，負荷が均等になるように処理を分散して割り当て，システム全体の可用性を高める機能です。クライアントからの create および invoke の要求をサーバ間，プロセス間およびスレッド間で負荷分散できます。

この節の構成を次の表に示します。

表 3-7　この節の構成（リクエストの負荷分散）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | 負荷分散のタイミング | 3.8.1 |
| | 負荷状況の監視 | 3.8.2 |
| 設定 | 実行環境での設定 | 3.8.3 |

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。

負荷分散は，スケジュールキューを共有している J2EE アプリケーション間で実行できるほか，複数の CTM デーモン間で負荷情報を交換することで，異なるスケジュールキューで制御されている J2EE アプリケーションに含まれる業務処理プログラムに対しても実行できます。

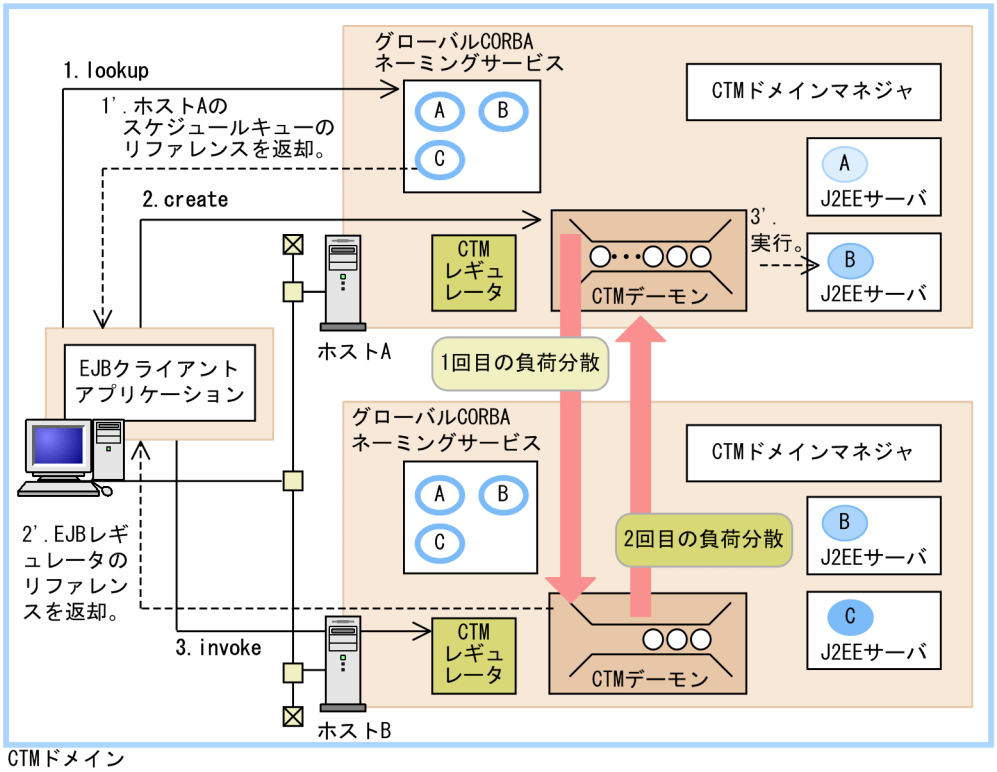
### 3.8.1　負荷分散のタイミング

CTM では，次の 2 回のタイミングで負荷分散を実行します。

- create の実行によって，EJB オブジェクトリファレンスを取得するタイミング
  このタイミングでは，負荷が軽い CTM デーモンに処理を振り分けるか，create を受け付けたホストの CTM デーモンに優先的に処理を振り分けるかが，create 時の選択ポリシーに従って決まります。
- invoke の実行によって，リモートインタフェースのビジネスメソッドを実行するタイミング
  このタイミングでは，負荷が軽い CTM デーモンに処理を振り分けるか，リクエストを受け付けたホストの CTM デーモンに優先的に処理を振り分けるかが，スケジュールポリシーに従って決まります。

クライアントから業務処理プログラムを呼び出す流れと負荷分散のタイミングを，次の図に示します。

図 3-21　EJB クライアントから業務処理プログラムを呼び出す流れと負荷分散のタイミング




注　　スマートエージェントおよびPRFデーモンのプロセスは省略しています。

図について説明します。

1. **EJB クライアントは，各ホストに配置されているグローバル CORBA ネーミングサービスに対して，lookup を実行します。**

   図の場合は，ホスト A に対して lookup を実行しています。

   グローバル CORBA ネーミングサービスには，スケジュールキューのリファレンスが登録されています。図の場合は，ホスト A から，登録されていたスケジュールキューのリファレンスが返却されます。

2. **取得したリファレンスを使用して create を実行します。**

   図の場合は，ホスト A の CTM デーモンに対して，create を実行しています。

   このタイミングで，**1 回目の負荷分散**が実行されます。

   このとき，create 時の選択ポリシーに従って負荷分散が実行されます。

   create を受け付けた CTM デーモンは，create 時の選択ポリシーに従って，次のどちらかのリファレンスを EJB クライアントに返却します。

   - create を受け付けたホストの CTM デーモンに対応する CTM レギュレータのリファレンス
   - CTM ドメイン内の負荷が軽い CTM デーモンに対応する CTM レギュレータのリファレンス

図の場合は，ホスト B の CTM レギュレータのリファレンスが返却されます。

3. **取得したリファレンスを使用して，リモートインタフェースに定義した invoke または remove を実行します。**

   図の場合は，ホスト B の CTM レギュレータに対して，invoke を実行しています。リクエストは，CTM レギュレータによって CTM デーモンに送信されます。

   このタイミングで，**2 回目の負荷分散**が実行されます。

   invoke 実行時に，スケジュールポリシーに従って負荷分散が実行されます。※

   図の場合は，リクエストを受け付けたホスト A の CTM デーモンに振り分けられました。振り分けられたリクエストはスケジュールキューに登録されます。実行時には，あらかじめプールされていた EJB オブジェクトのリファレンスと結び付けられて，J2EE サーバの業務処理プログラムを呼び出します。このとき，異常終了した J2EE サーバやハングアップしてタイムアウトした業務処理プログラムを呼び出すことはありません。

   注※

   remove 実行時にはスケジュールポリシーは適用されません。

業務処理プログラムからの応答は，リクエストを受け付けた CTM デーモンを経由して，EJB クライアントに返却されます。

## 3.8.2　負荷状況の監視

CTM では，スケジュールキューの負荷状況を監視できます。負荷状況の監視は，J2EE サーバ単位で指定した監視時間の間隔で実施されます。監視間隔の設定は，CTM デーモンを起動するときに ctmstart コマンドの引数として指定します。また，運用管理ポータルで構築したシステムを運用している場合は，論理 CTM であらかじめ設定しておくことができます。ctmstart 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「ctmstart（CTM デーモンの開始)」を参照してください。運用管理ポータル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 運用管理ポータル操作ガイド」の「10.7.2　スケジューリングの設定」を参照してください。

## 3.8.3　実行環境での設定

リクエストの負荷分散をする場合，CTM デーモンの設定が必要です。

CTM デーモンの設定は，簡易構築定義ファイルで実施します。リクエストの負荷分散の定義は，簡易構築定義ファイルの論理 CTM（component-transaction-monitor）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の負荷分散の定義について次の表に示します。

表 3-8　簡易構築定義ファイルでのリクエストの負荷分散の定義

| 項目 | 指定するパラメタ | 設定内容 |
|---|---|---|
| 負荷分散のタイミング設定 | ctm.CreatePolicy | create 要求の CTM ノード選択ポリシーを指定します。1 回目の負荷分散で使用されます。 |
| | ctm.DispatchPolicy | リクエストのスケジュールポリシーを指定します。2 回目の負荷分散で使用されます。 |
| 負荷状況の監視 | ctm.LoadCheckInterval | スケジュールキューの負荷状況を監視する時間間隔を指定します。 |

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。

## 3.9　リクエストのキューの滞留監視

この節の構成を次の表に示します。

表 3-9　この節の構成（リクエストのキューの滞留監視）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | スケジュールキューの滞留監視の概要 | 3.9.1 |
| | スケジュールキュー監視の例 | 3.9.2 |
| 設定 | 実行環境での設定 | 3.9.3 |
| 注意事項 | 注意事項 | 3.9.4 |

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。

J2EE サーバで，CTM デーモンのスケジュールキュー取り出しが遅れると，リクエストがスケジュールキューの中で滞留することがあります。これを監視する機能として，**スケジュールキュー監視機能**があります。ここでは，スケジュールキュー監視機能について説明します。

### 3.9.1　スケジュールキューの滞留監視の概要

スケジュールキューの滞留監視機能では，スケジュールキュー内に滞留しているリクエストの数を監視します。スケジュールキューにリクエストが滞留し，その数が一定の割合を超えると，メッセージを出力し，CTM デーモンは異常終了します。

スケジュールキュー監視は次のように動作します。

1. スケジュールキューの監視は，設定したキューの滞留率を超えた時点から開始します。
2. 監視状態になると，指定した監視時間間隔でスケジュールキューを監視します。
3. 監視のタイミングで，次に示すスケジュールキュー滞留監視式が成立すると，CTM デーモンが異常終了します。

   **スケジュールキュー滞留監視式**

   $(P／C_{n-1}) < (M1／100)$

   P：前回監視時点から現在までのリクエスト処理数

   $C_{n-1}$：前回の監視時点でのキュー滞留数

   M1：システム停止のしきい値（しきい値はシステムの処理率）

なお，スケジュールキュー監視は，ctmstart コマンドの-CTMWatchQueue オプションで設定します。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「ctmstart (CTM デーモンの開始)」を参照してください。

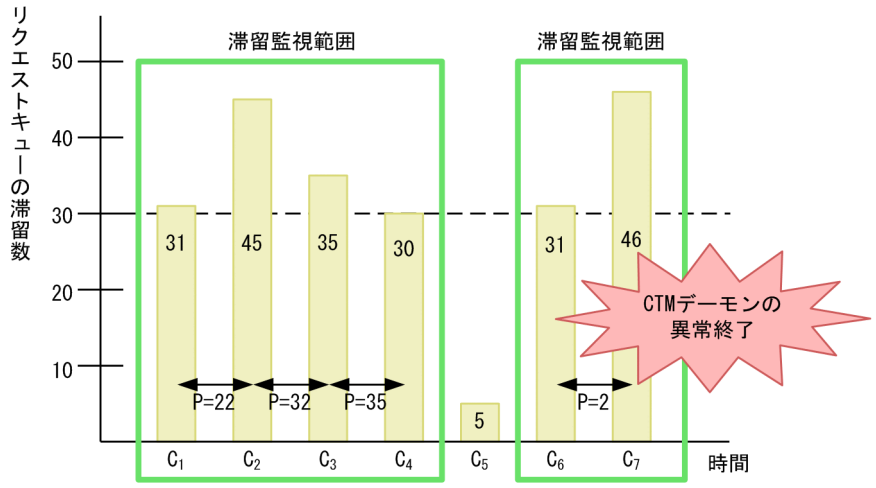
## 3.9.2　スケジュールキュー監視の例

例を使用してスケジュールキュー監視について説明します。

次の内容が設定されていることとします。

- スケジュールキュー監視を行うキューの滞留率：60%
- システム停止のしきい値：70%
- スケジュールキューの監視間隔：1 秒

図 3-22　スケジュールキュー監視




（凡例）Cn：監視時点　P：リクエスト処理数

この例の場合，システムの処理率が 70%を下回るとシステムが停止します。このため，スケジュールキュー滞留監視式「(P／$C_{n-1}$) ＜ (M1／100)」の右辺「M1／100」は，70／100 ＝ 0.7 となるため，この例でのスケジュールキュー滞留監視式は次のとおりとなります。

この例のスケジュール滞留監視式

(P／$C_{n-1}$) ＜ 0.7

左辺「(P／$C_{n-1}$)」の値が 0.7 未満になると，CTM デーモンが異常終了します。

また，この例では，スケジュールキューの滞留数の最大が 50 の場合について説明します。このため，スケジュールキューの滞留率 60%は，スケジュールキューの滞留数にすると 30 となります。滞留数が 30 を超えるとスケジュールキュー監視が開始されます。

図中の監視時点ごとにスケジュールキュー監視について説明します。

$C_1$

$C_1$ でのスケジュールキューの滞留数は 31 で，スケジュールキューの滞留率が 60%（滞留数は 30）を超えているので，スケジュールキューの滞留監視を開始します。

$C_2$

P（$C_1$ から $C_2$ までのリクエスト処理数）＝ 22 のため，スケジュールキュー滞留監視式の左辺「(P／$C_{n-1}$)」の値は次のようになります。

（P／$C_1$）＝ 22／31 ＝ 0.7

システムが停止する 0.7 と同じ値であるため，CTM デーモンは停止しません。

また，$C_2$ でのスケジュールキューの滞留数は 45 で，スケジュールキューの滞留率が 60%（滞留数は 30）を超えているので，引き続きスケジュールキューの滞留監視を実施します。

$C_3$

P（$C_2$ から $C_3$ までのリクエスト処理数）＝ 32 のため，スケジュールキュー滞留監視式の左辺「(P／$C_{n-1}$)」の値は次のようになります。

（P／$C_2$）＝ 32／45 ＝ 0.71

システムが停止する 0.7 を超えているので，CTM デーモンは停止しません。

また，$C_3$ でのスケジュールキューの滞留数は 35 で，スケジュールキューの滞留率が 60%（滞留数は 30）を超えているので，引き続きスケジュールキューの滞留監視を実施します。

$C_4$

P（$C_3$ から $C_4$ までのリクエスト処理数）＝ 35 のため，スケジュールキュー滞留監視式の左辺「(P／$C_{n-1}$)」の値は次のようになります。

（P／$C_3$）＝ 35／35 ＝ 1

システムが停止する 0.7 を超えているので，CTM デーモンは停止しません。

また，$C_4$ でのスケジュールキューの滞留数は 30 で，スケジュールキューの滞留率が 60%（滞留数は 30）と同じであるため，スケジュールキューの滞留監視は終了します。

$C_5$

P（$C_4$ から $C_5$ までのリクエスト処理数）＝ 5 のため，スケジュールキュー滞留監視式の左辺「(P／$C_{n-1}$)」の値は次のようになります。

（P／$C_4$）＝ 5／30 ＝ 0.16

システムが停止する 0.7 未満になっていますが，$C_5$ ではスケジュールキューの滞留監視をしていないため，CTM デーモンは停止しません。

$C_6$

$C_6$ でのスケジュールキューの滞留数は 31 で，スケジュールキューの滞留率が 60%（滞留数は 30）を超えているので，スケジュールキューの滞留監視を開始します。

$C_7$

P（$C_6$ から $C_7$ までのリクエスト処理数）＝ 2 のため，スケジュールキュー滞留監視式の左辺「(P／$C_{n-1}$)」の値は次のようになります。

（P／$C_6$）＝ 2／31 ＝ 0.06

システムが停止する 0.7 未満になっているので，CTM デーモンは異常停止します。

### 3.9.3　実行環境での設定

スケジュールキューの滞留を監視する場合，CTM デーモンの設定が必要です。

CTM デーモンの設定は，簡易構築定義ファイルで実施します。リクエストの負荷分散の定義は，簡易構築定義ファイルの論理 CTM（component-transaction-monitor）の<configuration>タグ内に指定します。キュー滞留監視状態へ移行する滞留率のしきい値を ctm.QueueRate パラメタに指定してください。

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。

### 3.9.4　注意事項

- 監視状態のリクエストキューに対して，閉塞コマンド（ctmholdque コマンド）を使用してリクエストキューに滞留しているリクエストを破棄した場合，破棄したリクエストは処理されたものとして判別されます。
- リクエストキューを監視している状態で，閉塞コマンド（ctmholdque コマンド）を使用した場合，監視状態は次のようになります。
  - 通常閉塞の場合（ctmholdque コマンドのオプション指定なし）
    リクエストが破棄されるため，キューの滞留数が減少します。このため，自動的に監視状態が解除されます。
  - 入り口閉塞の場合（ctmholdque コマンドに-CTMRequestLeave オプションを指定）
    滞留しているリクエストはサーバで処理されるため，監視状態のままとなります。
  - 出口閉塞の場合（ctmholdque コマンドに-CTMChangeServer オプションを指定）
    滞留しているリクエストは処理されず，処理率が 0 になるため，システムは停止します。このため，監視状態は解除されます。

## 3.10 CTM のゲートウェイ機能を利用した TPBroker／OTM クライアントとの接続

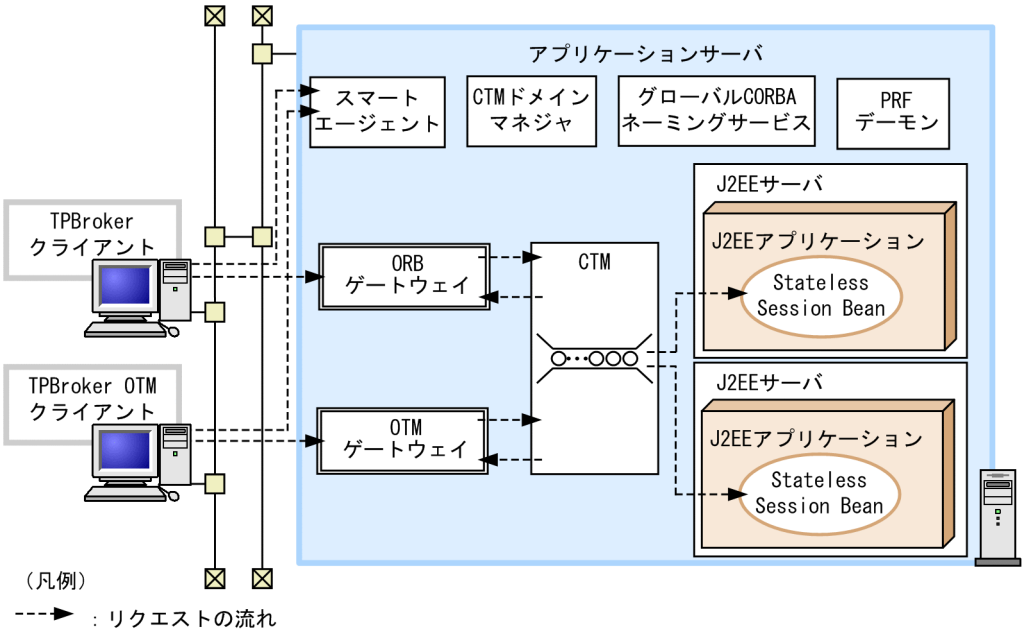
CTM では，次に示すクライアントからアプリケーションサーバ上で動作する J2EE アプリケーションを呼び出せるゲートウェイ機能を提供します。

- TPBroker クライアント
  TPBroker Version 5 以降で開発されたクライアントアプリケーションです。
- TPBroker OTM クライアント
  TPBroker Object Transaction Monitor で開発されたクライアントアプリケーションです。

また，CTM ではゲートウェイでリクエストを送受信するときの性能解析情報を出力できます。出力した性能解析情報は，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分析できます。性能解析トレース出力については，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」の性能解析トレースに関する説明を参照してください。

CTM のゲートウェイ機能を使用した，TPBroker クライアントまたは TPBroker OTM クライアントからの J2EE アプリケーションの呼び出しの概要を次の図に示します。

図 3-23 TPBroker クライアントまたは TPBroker OTM クライアントからの J2EE アプリケーションの呼び出しの概要




TPBroker クライアントは ORB ゲートウェイ経由，TPBroker OTM クライアントは OTM ゲートウェイ経由で，J2EE サーバ上の J2EE アプリケーションにリクエストを送信します。ORB ゲートウェイおよび OTM ゲートウェイは，CTM が提供するプロセス群で，ORB ゲートウェイおよび OTM ゲートウェイは，CTM デーモンを起動するときに，同時に起動されます。

TPBroker クライアントおよび TPBroker OTM クライアントから J2EE アプリケーションへのリクエスト方法と，リファレンスの解決方法を次に示します。

- TPBroker クライアントの場合

  ctmregltd コマンドの-CTMAgent オプションに 1 を指定する，または-CTMIDLConnect オプションに 1 を指定すると，CTM レギュレータで ORB ゲートウェイ機能が有効になります。-CTMAgent オプションに 1 を指定した場合，EJB のルックアップ名称をオブジェクト名称として，スマートエージェントに CORBA リファレンスを登録します。そのため，TPBroker クライアントでは，EJB のルックアップ名称を_bind()の引数に指定してリファレンスの参照を解決します。-CTMIDLConnect オプションに 1 を指定した場合，ctmgetior コマンドで IOR 文字列を取得することでリファレンスの参照を解決します。

- TPBroker OTM クライアントの場合

  ctmstart コマンドの-CTMTSCGwStart オプションに 1 以上を指定すると，OTM ゲートウェイが開始されます。TPBroker OTM クライアントでは，TSC ユーザプロキシを生成するコンストラクタの引数に，EJB のルックアップ（登録）名称を TSC アクセプタ名称として指定します。なお，TSC アクセプタ名称は省略できませんので注意してください。また，接続形態は，TSC レギュレータを選択してください。

なお，TPBroker クライアントおよび TPBroker OTM クライアントから J2EE サーバ上のアプリケーションを呼び出すためのクライアントアプリケーションの開発方法については，TPBroker または TPBroker Object Transaction Monitor のドキュメントを参照してください。

# 4 バッチアプリケーションのスケジューリング

バッチアプリケーションのスケジューリング機能を使用すると，バッチアプリケーションの実行リクエストを制御できるようになります。これによって，バッチサーバの数を変更することなく，複数のバッチアプリケーションの実行リクエストを受け付けられるようになります。この章では，バッチアプリケーションのスケジューリングの概要，スケジューリング機能を使用したバッチアプリケーションの実行，スケジューリング機能を使用するための設定などについて説明します。

なお，スケジューリング機能は，構成ソフトウェアに Component Transaction Monitor を含む製品だけで利用できる機能です。利用できる製品については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の「2.2.1　製品と構成ソフトウェアの対応」を参照してください。

## 4.1　この章の構成

**バッチアプリケーションのスケジューリング機能**は，CTM を使用して，バッチサーバで実行するバッチアプリケーションの実行リクエストを制御する機能です。以降，この機能をスケジューリング機能といいます。

この章の構成を次の表に示します。

表 4-1　この章の構成（バッチアプリケーションのスケジューリング）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | スケジューリング機能の概要 | 4.2 |
| | スケジューリング機能を使用したシステム | 4.3 |
| | スケジューリング機能使用時のバッチアプリケーション実行環境の構築と運用 | 4.4 |
| | スケジューリング機能を使用したバッチアプリケーションの実行 | 4.5 |
| | スケジューリング機能を使用する環境への移行 | 4.6 |
| 設定 | 実行環境での設定 | 4.7 |
| 注意事項 | スケジューリング機能使用時の注意事項 | 4.8 |

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。

なお，バッチサーバで提供する機能や，バッチアプリケーションの作成などについては，「2.　バッチサーバによるアプリケーションの実行」を参照してください。

## 4.2　スケジューリング機能の概要

この節では，スケジューリング機能の概要について説明します。

アプリケーションサーバでは，バッチアプリケーションのスケジューリングには，CTM を使用します。CTM は，キューを使用してそれぞれのバッチアプリケーションの実行を制御します。このキューを，**スケジュールキュー**といいます。

### 4.2.1　バッチアプリケーションをスケジューリングする利点

ここでは，スケジューリング機能を使用する利点について説明します。

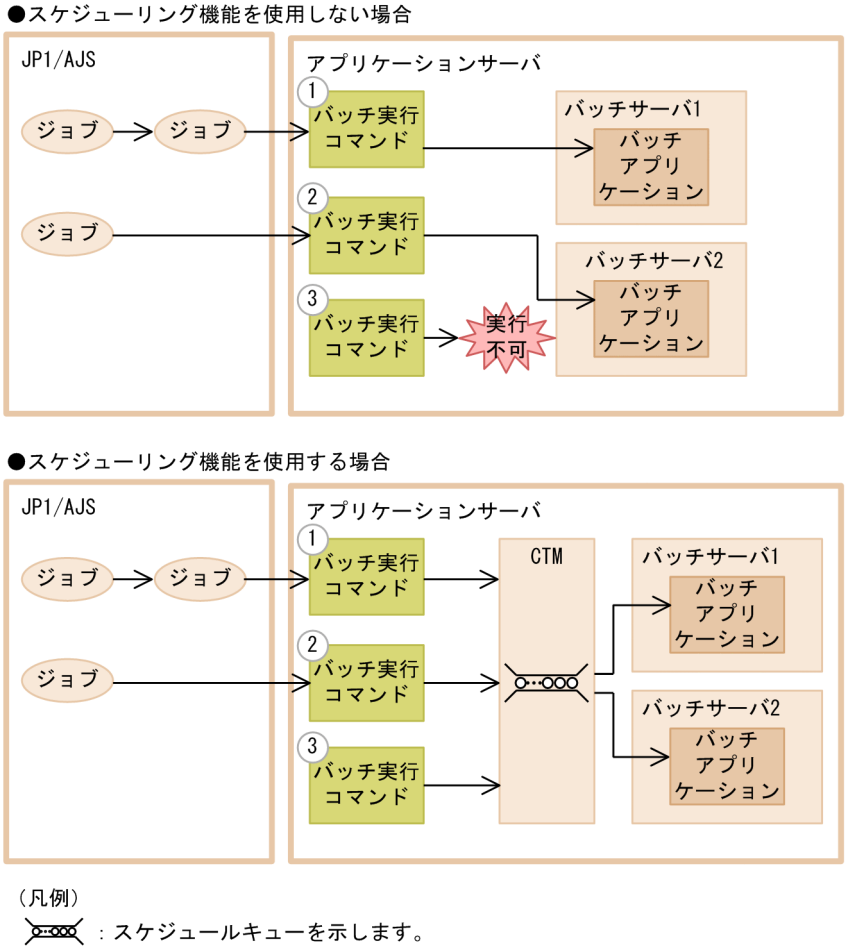
バッチサーバでは，同時に実行できるバッチアプリケーションの数は一つです。バッチアプリケーションを開始するには，アプリケーションサーバで提供しているバッチ実行コマンドを使用します。バッチサーバでは，バッチ実行コマンドによるバッチアプリケーションの実行リクエストを受けて，バッチアプリケーションを開始します。

スケジューリング機能を使用しない場合，バッチサーバの数を超えたバッチアプリケーションの実行リクエストは受け付けられません。この場合，受け付けられないリクエストはエラーとなります。また，バッチ実行コマンドにどのバッチサーバで実行するかを定義する必要があります。

スケジューリング機能を使用する場合，バッチサーバの数を超えたバッチアプリケーションの実行リクエストは，CTM によってスケジュールキューに滞留され，エラーになりません。滞留されたリクエストは，CTM によってバッチサーバに振り分けられます。このため，バッチサーバの数に関係なく，バッチ実行コマンドを実行できます。また，バッチアプリケーションの実行リクエストを実行するバッチサーバは，CTM によって振り分けられるため，バッチ実行コマンドにどのバッチサーバで実行するかを定義する必要がありません。

スケジューリング機能の使用の有無によるバッチアプリケーションの実行の流れを次の図に示します。

図 4-1　スケジューリング機能の使用の有無によるバッチアプリケーションの実行の流れ




この図は，バッチサーバが 2 台のシステムに対して，JP1/AJS のジョブまたは直接マシンからバッチ実行コマンドを同時に実行する例です。

スケジューリング機能を使用しない場合は，図中の 2 と 3 のバッチ実行コマンドを同時に実行できません。スケジューリング機能を使用する場合は，CTM によってバッチアプリケーションの実行リクエストがバッチサーバに振り分けられるため，図中の 2 と 3 のバッチ実行コマンドを同時に実行できます。

## 4.2.2　スケジューリング機能を使用するための前提

ここでは，スケジューリング機能を使用するための前提条件について説明します。

スケジューリング機能を使用する場合，CTM の使用が前提となります。CTM の詳細は，「3.　CTM によるリクエストのスケジューリングと負荷分散」を参照してください。

また，CTM を使用するためには，CTM を使用する構成でシステムを構築する必要があります。CTM を使用する構成については，「4.3　スケジューリング機能を使用したシステム」を参照してください。

### 4.2.3　スケジューリング機能を使用したバッチアプリケーションの実行処理の流れ

ここでは，バッチアプリケーションの実行処理の流れについて説明します。

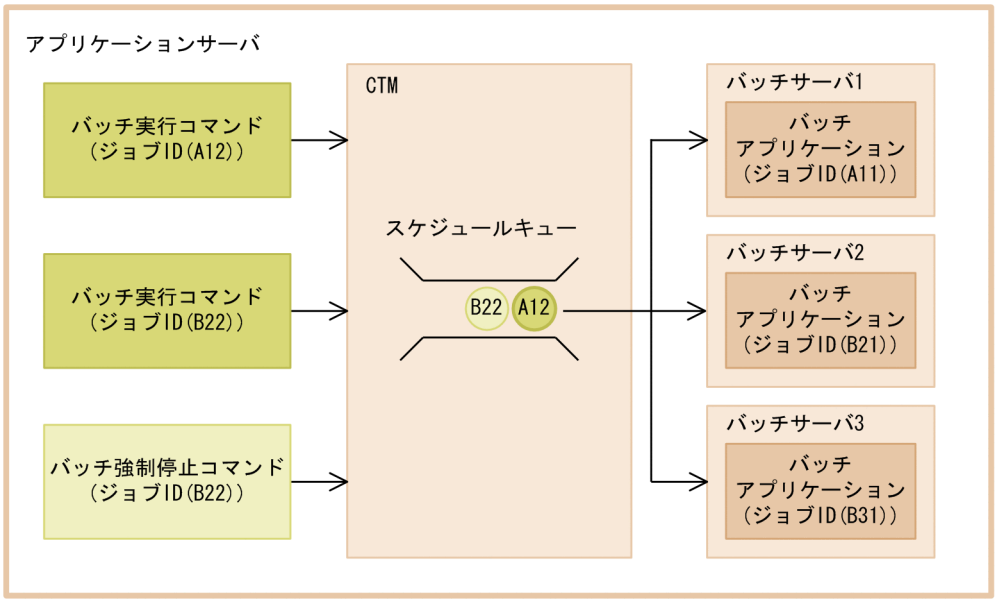
スケジューリング機能を使用する場合，各バッチサーバで実行されるバッチアプリケーションは，ジョブ ID で区別されます。**ジョブ ID** は，実行するバッチアプリケーションの実行リクエストを区別するための文字列です。コマンド実行時に任意の値を設定できます。コマンド実行時にジョブ ID を省略した場合，ジョブ ID はスケジューリング機能によって自動生成されます。このジョブ ID は，CTM によって管理されます。

また，CTM によってバッチアプリケーションが振り分けられるバッチサーバ群を，**スケジュールグループ**といいます。スケジュールキューは，スケジュールグループごとに作成されます。バッチアプリケーションの業務分類ごとに同時実行数を制御したい場合などに，スケジュールグループを指定してください。スケジュールグループは，システム内で一意になるように設定してください。マシンごとに CTM が異なる場合でも，スケジュールグループは別々に設定する必要があります。なお，スケジュールグループを指定する場合は，バッチ実行コマンドとバッチサーバで設定が必要です。設定方法については，「4.7　実行環境での設定」を参照してください。

スケジューリング機能を使用する場合も，バッチアプリケーションの実行環境は，JP1/AJS と連携できます。

スケジューリング機能を使用したバッチアプリケーション実行の流れを次の図に示します。

図 4-2　スケジューリング機能を使用したバッチアプリケーション実行の流れ




（凡例）

：スケジュールキューを示します。

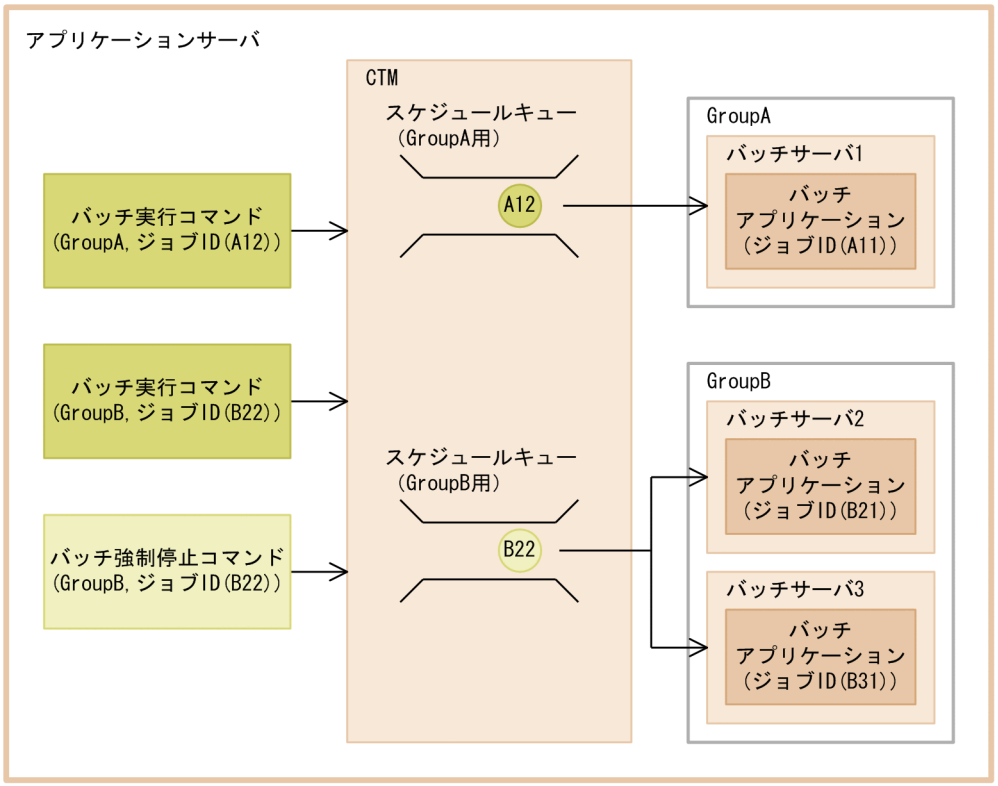
x　：バッチアプリケーションの実行リクエストを示します。xは，ジョブIDを示します。

x　：バッチ強制停止コマンドの実行によって，削除対象となったバッチアプリケーションの実行リクエストを示します。xは，ジョブIDを示します。

この図では，CTM によって，同じスケジュールグループに属するバッチサーバ 1 からバッチサーバ 3 に，バッチアプリケーションの実行リクエストを振り分けています。なお，スケジュールキューからあふれたバッチアプリケーションの実行リクエストはエラーになります。

複数のスケジュールグループを使用したバッチアプリケーション実行の流れを次の図に示します。

### 図 4-3　複数のスケジュールグループを使用したバッチアプリケーション実行の流れ




（凡例）

：スケジュールキューを示します。

x ：バッチアプリケーションの実行リクエストを示します。xは，ジョブIDを示します。

x ：バッチ強制停止コマンドの実行によって，削除対象となったバッチアプリケーションの実行リクエストを示します。xは，ジョブIDを示します。

この図は，GroupA と GroupB の二つのスケジュールグループを指定した例で，スケジュールキューは二つ作成されます。使用するスケジュールグループは，コマンドで定義します。バッチアプリケーションは，コマンドのスケジュールグループの設定に従って，スケジュールキューに振り分けられます。なお，この図の場合，各スケジュールグループのバッチサーバでバッチアプリケーションを実行中のため，CTM に受け付けられたバッチアプリケーションは，スケジュールキュー内で待機しています。

## 4.3　スケジューリング機能を使用したシステム

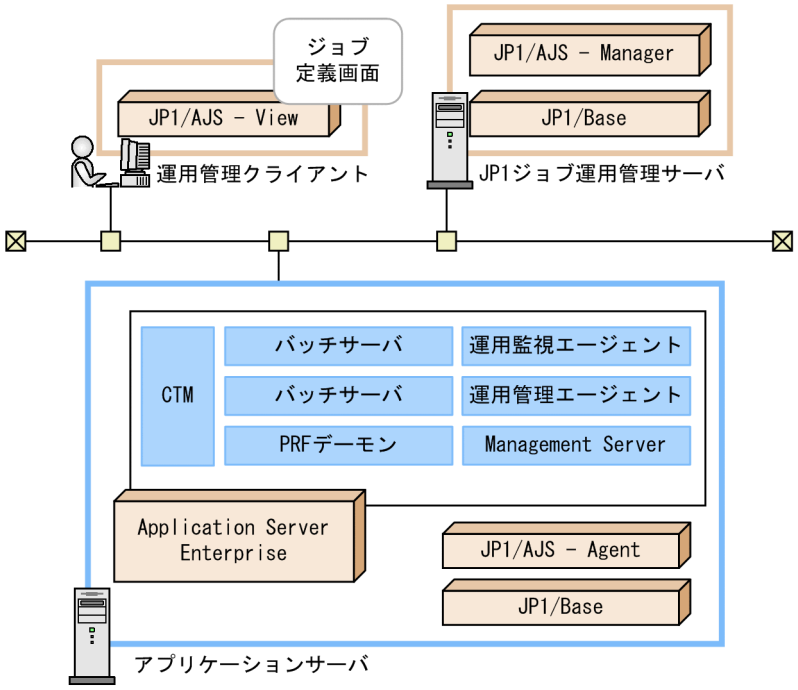
この節では，スケジューリング機能を使用したシステムの構成と，必要なプロセスについて説明します。

### 4.3.1　スケジューリング機能を使用したシステムの構成

ここでは，スケジューリング機能を使用したシステムの構成について説明します。

スケジューリング機能を使用したシステムの構成例を次の図に示します。

図 4-4　スケジューリング機能を使用したシステムの構成例



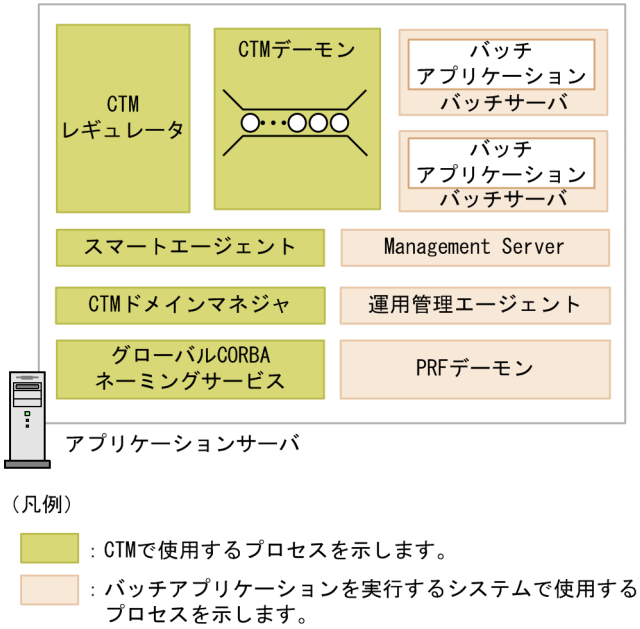

この図の場合，バッチアプリケーションを実行するシステムは，JP1/AJS と連携しています。JP1/AJS と連携しない場合は，図中の運用管理クライアント，JP1 ジョブ運用管理サーバ，ならびにアプリケーションサーバの JP1/AJS - Agent および JP1/Base は必要ありません。バッチサーバおよびバッチアプリケーションの操作の流れについては，「2.2.2(1)　JP1/AJS と連携するシステム」および「2.2.2(3)　JP1/AJS，BJEX，および JP1/Advanced Shell と連携しないシステム」を参照してください。

### 4.3.2　スケジューリング機能で必要なプロセス

ここでは，スケジューリング機能で必要なプロセスについて説明します。

スケジューリング機能を使用する場合，CTM を使用します。スケジューリング機能を使用するアプリケーションサーバのプロセス構成例を次の図に示します。

図 4-5　アプリケーションサーバのプロセス構成例（スケジューリング機能を使用する場合）




CTM で使用するプロセスの概要を次の表に示します。

表 4-2　CTM で使用するプロセスの概要（スケジューリング機能の場合）

| プロセス | 説明 |
|---|---|
| CTM デーモン | バッチアプリケーションの実行リクエストを制御するスケジュールキューを管理するプロセスです。 |
| CTM レギュレータ | CTM デーモンに集中するバッチアプリケーションの実行リクエストを，分散集約するためのプロセスです。 |
| CTM ドメインマネジャ | 同じ CTM ドメイン内の CTM デーモンの情報を管理するプロセスです。 |
| グローバル CORBA ネーミングサービス | 同じ CTM ドメイン内に含まれるホスト上のバッチアプリケーションの情報を共有管理しているネーミングサービスです。 |
| スマートエージェント | TPBroker で提供されている，動的な分散ディレクトリサービスを提供するプロセスです。異なるネットワークセグメントにある CTM デーモンに情報を配布する場合に使用されます。詳細については，マニュアル「Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド」を参照してください。 |

各プロセスを配置する指針，各プロセスについては，「3.　CTM によるリクエストのスケジューリングと負荷分散」を参照してください。

なお，PRF デーモン（パフォーマンストレーサ）は，CTM デーモンが出力した性能解析情報をファイルに出力する I／O プロセスとして，CTM でも使用されます。詳細は，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.2.1　アプリケーションサーバの性能解析トレースの概要」を参照してください。

## 4.4　スケジューリング機能使用時のバッチアプリケーション実行環境の構築と運用

この節では，バッチアプリケーションの実行環境の構築や，運用について説明します。

バッチアプリケーションの実行環境は，スケジューリング機能を使用する場合も，Smart Composer 機能と，サーバ管理コマンドを使用して構築します。バッチアプリケーションの実行環境の運用手順や，使用できる運用機能は，スケジューリング機能を使用しない場合と同じです。

バッチアプリケーションの実行環境の構築手順については，「2.2.3(1)　バッチアプリケーションの実行環境の構築」を参照してください。ただし，スケジューリング機能を使用する場合は，バッチサーバの定義や構築に加えて，CTM，スマートエージェントなどの環境設定や構築が必要になります。CTM，スマートエージェントなどの構築にも Smart Composer 機能を使用します。詳細は，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.6　バッチアプリケーションを実行するシステムの構築」を参照してください。バッチアプリケーションの実行環境でできる運用や，運用手順については，「2.2.3(2)　バッチアプリケーションの実行環境の運用」を参照してください。

また，スケジューリング機能を使用したシステムでも，JP1 やクラスタソフトウェアと連携できます。詳細は，「2.2.3(3)　ほかのプログラムとの連携」を参照してください。

## 4.5　スケジューリング機能を使用したバッチアプリケーションの実行

この節では，スケジューリング機能を使用したバッチアプリケーションの実行について説明します。バッチアプリケーション実行機能については，「2.3.1　バッチアプリケーション実行機能の概要」を参照してください。

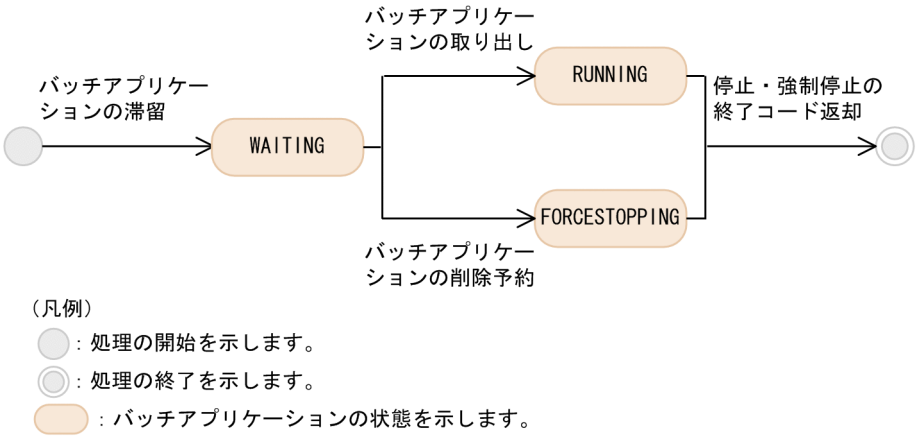
なお，バッチサーバで出力するバッチアプリケーションの実行ログは，スケジューリング機能を使用しない場合と同じです。バッチアプリケーションの実行ログについては，「2.3.5　バッチアプリケーションのログ出力」を参照してください。

### 4.5.1　スケジューリング機能を使用したバッチアプリケーションの状態遷移

ここでは，スケジューリング機能を使用したバッチアプリケーションの状態遷移について説明します。

バッチアプリケーションの状態遷移を次の図に示します。

図 4-6　バッチアプリケーションの状態遷移（スケジューリング機能を使用する場合）




バッチアプリケーションの各状態の説明を次の表に示します。

表 4-3　バッチアプリケーションの各状態の説明

| バッチアプリケーションの状態 | 説明 |
|---|---|
| WAITING | バッチサーバでほかのバッチアプリケーションが実行中のため，スケジュールキュー内で待機している状態です。 |
| RUNNING | バッチアプリケーションがバッチサーバ上にある状態です。 |
| FORCESTOPPING | バッチ強制停止コマンドによって，スケジュールキュー内のバッチアプリケーションが削除を予約された状態です。 |

バッチアプリケーションの状態は，バッチアプリケーション情報から確認できます。バッチアプリケーション情報の表示方法については，「4.5.4　バッチアプリケーション情報の一覧表示」を参照してください。

### 4.5.2　バッチアプリケーションの実行

バッチアプリケーションを開始するには，cjexecjob コマンドを使用します。cjexecjob コマンドを実行するには，次の二つの方法があります。

1. cjexecjob コマンドを直接実行する方法
   JP1/AJS を使用しない場合はこの方法で開始します。
2. cjexecjob コマンドを JP1/AJS のジョブとして定義しておき，JP1/AJS から実行する方法
   JP1/AJS を使用する場合はこの方法で開始します。

2.の方法でバッチアプリケーションを開始するときの JP1/AJS のジョブの定義については，「2.13 JP1/AJS との連携」を参照してください。なお，JP1/AJS からバッチアプリケーションを実行する際には，あらかじめバッチサーバを起動しておいてください。

バッチアプリケーションの開始処理，終了処理および実行時の注意事項については，スケジューリング機能を使用しない場合と同じです。詳細は，「2.3.2　バッチアプリケーションの実行」を参照してください。

### 4.5.3　バッチアプリケーションの強制停止

バッチアプリケーションを強制停止するには，cjkilljob コマンドを使用します。スケジューリング機能を使用する場合，cjkilljob コマンドには，ジョブ ID を指定します。

ジョブ ID で指定されたバッチアプリケーションの実行リクエストがスケジュールキュー内で待機中の場合は，削除が予約されます。削除が予約されたバッチアプリケーションの実行リクエストは，CTM によってスケジュールキューから取り出された時に削除されます。

ジョブ ID で指定されたバッチアプリケーションの実行リクエストがバッチサーバで実行中の場合は，強制停止されます。

バッチアプリケーションの強制停止方法，強制停止処理および実行時の注意事項については，スケジューリング機能を使用しない場合と同じです。詳細は，「2.3.3　バッチアプリケーションの強制停止」を参照してください。

### 4.5.4　バッチアプリケーション情報の一覧表示

実行中または待機中のバッチアプリケーションの状態や，バッチ実行コマンドの開始時刻などをバッチアプリケーション情報として一覧表示できます。ここでは，バッチアプリケーション情報の一覧表示について説明します。

## (1) バッチアプリケーション情報の一覧表示の方法

バッチアプリケーション情報の一覧を表示するには，JP1/AJS を使用しているかどうかに関係なく，cjlistjob コマンドを直接実行します。

バッチアプリケーション情報は，次の単位で取得できます。

- コマンドの引数に指定されているスケジュールグループ
- すべてのスケジュールグループ

cjlistjob コマンドの引数には，バッチアプリケーション情報を取得したいバッチサーバが属するスケジュールグループ名，または-all オプションを指定します。スケジュールグループ名は複数指定できます。-all オプションを指定すると，同一マシン内のバッチサーバが使用しているすべてのスケジュールグループのバッチアプリケーション情報を取得できます。

## (2) バッチアプリケーション情報の一覧表示処理

cjlistjob コマンドを実行すると，引数または usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.schedule.group.name キーに指定したスケジュールグループで実行中のバッチアプリケーションの情報が取得できます。バッチアプリケーション情報は，標準出力に出力されます。

取得できるバッチアプリケーション情報を次の表に示します。

### 表 4-4　取得できるバッチアプリケーション情報

| 取得できるバッチアプリケーション情報の項目 | 内容 |
|---|---|
| スケジュールグループ名 | バッチアプリケーションの実行リクエストが振り分けられるスケジュールグループの名称が出力されます。 |
| バッチアプリケーションの状態 | 「running」，「waiting」または「forceStopping」が出力されます。running，waiting および forceStopping は，それぞれバッチアプリケーションの状態が RUNNING，WAITING および FORCESTOPPING であることを示します。バッチアプリケーションの状態については，「4.5.1 スケジューリング機能を使用したバッチアプリケーションの状態遷移」を参照してください。 |
| バッチアプリケーション名 | cjexecjob コマンドに指定したバッチアプリケーションのクラス名が出力されます。 |
| 性能解析トレースのルートアプリケーション情報 | 性能解析トレースのルートアプリケーションの通信番号が出力されます。<br>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，バッチアプリケーションの状態を確認できます。 |
| バッチ実行コマンドの実行時刻 | cjexecjob コマンドを実行した時刻が次の形式で出力されます。なお，△は，半角スペースを表します。<br>yyyy/mm/dd△hh:mm:ss.ssssss<br>yyyy：西暦年，mm：月，dd：日，hh：時，mm：分，ss：秒，ssssss：マイクロ秒 |

| 取得できるバッチアプリケーション情報の項目 | 内容 |
|---|---|
| バッチアプリケーションの待機開始時刻・実行開始時刻・強制停止受付時刻 | バッチアプリケーションの状態ごとに，バッチアプリケーションの状態を開始した時刻が次の形式で出力されます。なお，△は，半角スペースを表します。<br>yyyy/mm/dd△hh:mm:ss.ssssss<br>yyyy：西暦年，mm：月，dd：日，hh：時，mm：分，ss：秒，ssssss：マイクロ秒 |
| ジョブ ID | バッチアプリケーションのジョブ ID が出力されます。 |
| バッチアプリケーションを実行しているバッチサーバ名 | バッチアプリケーションを実行しているバッチサーバ名が出力されます。なお，バッチアプリケーションの状態が「waitting」の場合は，「-」が出力されます。 |

なお，バッチアプリケーションがない場合，cjlistjob コマンドを実行しても何も出力されません。この場合，cjlistjob コマンドは正常終了します。

cjlistjob コマンドの出力形式と出力例を次に示します。なお，△は，半角スペースを表します。

### cjlistjob コマンドの出力形式

```
<スケジュールグループ名>△<バッチアプリケーションの状態>△<バッチアプリケーション名>△<性能解析トレースのルートアプリケーション情報>△<バッチ実行コマンドの実行時刻>△<バッチアプリケーションの待機開始時刻・実行開始時刻・強制停止受付時刻>△<ジョブID>△<バッチアプリケーションを実行しているバッチサーバ名>
<スケジュールグループ名>△<バッチアプリケーションの状態>△<バッチアプリケーション名>△<性能解析トレースのルートアプリケーション情報>△<バッチ実行コマンドの実行時刻>△<バッチアプリケーションの待機開始時刻・実行開始時刻・強制停止受付時刻>△<ジョブID>△<バッチアプリケーションを実行しているバッチサーバ名>
：
```

### cjlistjob コマンドの出力例

```
JOBGROUP running com.xxx.mypackage.batchApp1 0x0000000000123456 2008/04/14 17:27:35.689012 2008/04/14 17:27:37.182777 HOGE MybatchServer1
JOBGROUP running com.xxx.mypackage.batchApp2 0x00000000002345678 2008/04/14 17:45:20.123456 2008/04/14 19:21:56.271354 102 MybatchServer2
JOBGROUP running com.xxx.mypackage.batchApp3 0x000000000034567890 2008/04/14 18:15:54.397890 2008/04/14 19:00:00.123447 #5HL390_G3CV7 MybatchServer3
JOBGROUP waitting com.xxx.mypackage.batchApp4 0x000000000045678901 2008/04/14 18:30:24.125444 2008/04/14 18:30:25.006220 112345 -
```

この例では，スケジュールグループ JOBGROUP では，MybatchServer1，MybatchServer2 および MybatchServer3 でバッチアプリケーションが実行中であり，batchApp4 のバッチアプリケーションの実行リクエストが待機中であることを示しています。

### 4.5.5　バッチアプリケーションで使用するコマンドの実行について

バッチアプリケーションで使用するコマンドの種類，およびバッチサーバの状態とコマンドの実行については，次の点以外はスケジューリング機能を使用しない場合と同じです。スケジューリング機能を使用しない場合との相違点を次に示します。

- バッチサーバで cjexecjob コマンドを処理しているときも，cjexecjob コマンドを実行できます。
- バッチサーバの状態が次のどれかのときに，cjexecjob コマンド，cjkilljob コマンドまたは cjlistjob コマンドを実行すると，メッセージ KDJE55046-E が出力されます。
  - バッチサーバ起動中
  - バッチサーバ停止中
  - バッチサーバの停止完了後
- cjexecjob コマンドとバッチサーバの間，cjkilljob コマンドまたは cjlistjob コマンドと CTM の間でタイムアウトが発生するまでの時間を設定できます。タイムアウトは，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.request.timeout キーで設定します。設定方法については，「4.7.3　バッチアプリケーションで使用するコマンドの設定」を参照してください。

これらの相違点以外については，「2.3.6　バッチアプリケーションで使用するコマンドの実行について」を参照してください。

ここでは，バッチアプリケーションで使用するコマンドの処理中に異常終了した場合の対処と，コマンド実行時の注意事項について説明します。

#### (1)　コマンド処理中にバッチサーバが異常終了した場合

バッチサーバで cjexecjob コマンド，cjkilljob コマンドまたは cjlistjob コマンドを処理している場合に，バッチサーバが異常終了したときは，メッセージ KDJE55021-E が出力されます。バッチサーバの状態を確認してから再度コマンドを実行してください。

#### (2)　コマンド処理中に CTM デーモンまたは CTM レギュレータが異常終了した場合

cjexecjob コマンド，cjkilljob コマンドまたは cjlistjob コマンドを処理している場合に，CTM デーモンまたは CTM レギュレータが異常終了したときは，メッセージ KDJE55047-E が出力されます。このメッセージは，スマートエージェントからスケジュールグループ名を取得したあと，CTM デーモンまたは CTM レギュレータと通信する際にプロセスが異常終了した場合に出力されます。CTM デーモンおよび CTM レギュレータの状態を確認してから再度コマンドを実行してください。

#### (3)　コマンド実行時の注意事項

コマンド実行時の注意事項を次に示します。

- IP アドレスを複数持つマシンで，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）または環境変数に IP アドレスの指定がない場合は，ORB ゲートウェイが接続する IP アドレスが自動的に判断されます。
- スケジューリング機能を使用する場合は，CTM によって振り分けられたバッチサーバでバッチアプリケーションを実行します。そのため，cjexecjob コマンドはバッチサーバに対して直接実行できません。
- 待機中のバッチアプリケーションに対して cjkilljob コマンドを実行した場合，バッチアプリケーションは CTM によって削除予約されます。削除予約されたバッチアプリケーションはスケジュールキューから取り出された時に削除されます。この場合，削除予約されたバッチアプリケーションがスケジュールキュー内に残っているので，次の点に注意してください。
  - 削除予約されたバッチアプリケーションと重複するジョブ ID は使用できません。
  - cjexecjob コマンドの実行によって，バッチアプリケーションの実行リクエストの数がスケジュールキューに登録できる数を超えると，バッチサーバはメッセージ KDJE55060-E を出力して異常終了します。
- 待機中のバッチアプリケーションに対して cjkilljob コマンドを実行した場合，スケジュールキューから取り出されるまで，cjexecjob コマンドは終了しません。
- cjexecjob コマンド，cjkilljob コマンドまたは cjlistjob コマンド実行時に，バッチサーバがない場合，メッセージを出力してコマンドは異常終了します。出力されるメッセージは，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.ctm.enabled キーの指定によって異なります。
  - 「true」を指定している場合

    メッセージ KDJE55010-E または KDJE55046-E を出力します。
  - 「false」を指定している場合

    メッセージ KDJE55010-E を出力します。
- cjexecjob コマンド実行時，簡易構築定義ファイルおよび usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の指定によって，コマンドが異常終了することがあります。
  - 簡易構築定義ファイルで ejbserver.ctm.enabled パラメタに「true」を指定して，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.ctm.enabled キーで「false」を指定している場合

    メッセージ KDJE55067-E を出力して異常終了します。
  - 簡易構築定義ファイルで ejbserver.ctm.enabled パラメタに「false」を指定して，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.ctm.enabled キーで「true」を指定している場合

    指定したスケジュールグループがないときは，メッセージ KDJE55046-E を出力して異常終了します。
- cjlistjob コマンド実行時に，簡易構築定義ファイルで ejbserver.ctm.enabled パラメタに「false」を指定して，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）の batch.ctm.enabled キーで「true」を指定している場合，バッチサーバではコマンドを受け付けられません。この場合，バッチアプリケーション情報は出力されません。

## 4.6　スケジューリング機能を使用する環境への移行

この節では，スケジューリング機能を使用していない環境からの移行について説明します。バッチアプリケーションの実行環境を，スケジューリング機能を使用していない環境から，スケジューリング機能を使用する環境に移行する場合，使用中の環境はそのまま使用できません。

使用中の環境で，定義ファイルを編集する必要があります。環境移行時に設定を編集するファイルを次の表に示します。

### 表 4-5　環境移行時に設定を編集するファイル

| ファイル | 編集する主なキー | 設定内容 | 必須または任意 |
|---|---|---|---|
| usrconf.properties（バッチサーバ用ユーザプロパティファイル） | ejbserver.ctm.enabled | true | 必須 |
| | vbroker.agent.enableLocator | true※ | 任意 |
| | ejbserver.batch.schedule.group.name | スケジュールグループ名 | 任意 |
| | ejbserver.batch.queue.length | 作成されるスケジュールキューの長さ | 任意 |
| usrconf.cfg（バッチアプリケーション用オプション定義ファイル） | batch.ctm.enabled | true | 必須 |
| | batch.schedule.group.name | スケジュールグループ名 | 任意 |
| | batch.request.timeout | バッチ実行コマンドとバッチサーバの間，バッチ強制停止コマンドまたはバッチ一覧表示コマンドとCTM の間のタイムアウト | 任意 |
| | batch.vbroker.agent.port | スマートエージェントが使用しているポート番号 | 任意 |

（凡例）必須：必ず指定する　任意：必要に応じて設定する

注　ここでは，スケジューリング機能を使用する環境への移行時に編集する主なキーについて説明しています。usrconf.properties（バッチサーバ用ユーザプロパティファイル）のファイルおよびキーについては，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「3.2.2　usrconf.properties（バッチサーバ用ユーザプロパティファイル）」を参照してください。

usrconf.cfg（バッチアプリケーション用オプション定義ファイル）のファイルおよびキーについては，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「3.2.5　usrconf.cfg（バッチアプリケーション用オプション定義ファイル）」を参照してください。

注※　デフォルトでは false が設定されていますが，CTM との連携時には自動的に true が設定されます。

各ファイルで編集するパラメタの詳細は，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。

## 4.7　実行環境での設定

スケジューリング機能を使用する場合，次の設定が必要です。

- バッチサーバ
- CTM
- バッチアプリケーションで使用するコマンド

この節では，それぞれの設定項目について説明します。なお，スケジューリング機能を使用する場合は，バッチアプリケーション実行機能の定義も設定する必要があります。バッチアプリケーション実行機能の定義については，「2.3.10　実行環境での設定（バッチサーバの設定）」を参照してください。

### 4.7.1　バッチサーバの設定

バッチサーバの設定は，簡易構築定義ファイルで実施します。スケジューリング機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのスケジューリング機能の定義を次の表に示します。

表 4-6　簡易構築定義ファイルでのスケジューリング機能の定義

| 項目 | 指定するパラメタ | 設定内容 | 必須または任意 |
|---|---|---|---|
| スケジューリング機能を使用する設定 | ejbserver.ctm.enabled | スケジューリング機能を使用するかどうかを指定します。デフォルトでは true が設定されます。また，<tier-type>タグに ctm-tier を指定している場合，システムの構築時には自動的に true が設定されます。 | 任意 |
| スマートエージェントを使用する設定 | vbroker.agent.enableLocator | スマートエージェントを使用します。デフォルトでは false が設定されますが，CTM との連携時には自動的に true が設定されます。このため，パラメタの値を true に変更する必要はありません。 | 任意 |
| スケジュールグループ名の設定 | ejbserver.batch.schedule.group.name | CTM によって管理されるバッチサーバ群のスケジュールグループ名を指定します。デフォルトでは JOBGROUP が設定されます。<br>CTM は，スケジュールグループごとにバッチアプリケーションの実行をスケジューリングします。<br>複数のスケジュールグループを使用してスケジュールキューを分ける場合には，バッチサーバごとにスケジュールグループ名を設定してください。 | 任意 |
| スケジュールキューの長さの設定 | ejbserver.batch.queue.length | CTM で作成されるスケジュールキューの長さを指定します。デフォルトでは 50 が設定されます。 | 任意 |

（凡例）任意：必要に応じて設定する

注　ここでは，スケジューリング機能使用時に指定する主なパラメタについて説明しています。スケジューリング機能使用時には，次の ejbserver.ctm から始まるパラメタも任意で指定できます。

- ejbserver.ctm.ActivateTimeOut
- ejbserver.ctm.CTMDomain
- ejbserver.ctm.CTMID
- ejbserver.ctm.CTMMyHost
- ejbserver.ctm.DeactivateTimeOut

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)」を参照してください。

## 4.7.2　CTM の設定

CTM の設定は，簡易構築定義ファイルで実施します。スケジューリング機能の定義は，簡易構築定義ファイルの論理 CTM（componenttransaction-monitor）の<configuration>タグ内に指定します。指定するパラメタを次に示します。このパラメタは必ず指定してください。

- ctm.Agent
  スケジューリング機能を使用する場合には，CTM レギュレータの ORB ゲートウェイ機能を使用します。パラメタの値は必ず 1 を指定してください。

簡易構築定義ファイルおよび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。

## 4.7.3　バッチアプリケーションで使用するコマンドの設定

バッチアプリケーションで使用するコマンドの設定は，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）で実施します。スケジューリング機能の定義は，usrconf.cfg でコマンドのオプションを指定します。

usrconf.cfg でのスケジューリング機能の定義を次の表に示します。

### 表 4–7　usrconf.cfg でのスケジューリング機能の定義

| 項目 | 指定するキー | 設定内容 | 必須または任意 |
|---|---|---|---|
| スケジューリング機能を使用する設定 | batch.ctm.enabled | スケジューリング機能を使用するかどうかを指定します。パラメタの値は必ず true を指定してください。 | 必須 |
| スケジュールグループ名の設定 | batch.schedule.group.name | CTM によって管理されるバッチサーバ群のスケジュールグループ名を指定します。デフォルトでは JOBGROUP が設定されます。 | 任意 |

| 項目 | 指定するキー | 設定内容 | 必須または任意 |
|---|---|---|---|
| | | CTM は，スケジュールグループごとにバッチアプリケーションの実行をスケジューリングします。 | |
| CTM に接続している最大時間の設定 | batch.request.timeout | バッチ実行コマンドとバッチサーバの間，バッチ強制停止コマンドまたはバッチ一覧表示コマンドと CTM の間のタイムアウトを指定します。デフォルトでは 0（タイムアウトしない）が設定されます。 | 任意 |
| スマートエージェントが使用しているポート番号の設定 | batch.vbroker.agent.port | スマートエージェントが使用しているポート番号を指定します。デフォルトでは 14000 が設定されます。 | 任意 |

（凡例）必須：必ず指定する　任意：必要に応じて設定する

注　ここでは，スケジューリング機能使用時に指定する主なキーについて説明しています。usrconf.cfg（バッチアプリケーション用オプション定義ファイル）およびキーについては，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「3.2.5　usrconf.cfg（バッチアプリケーション用オプション定義ファイル）」を参照してください。

## 4.8 スケジューリング機能使用時の注意事項

スケジューリング機能使用時の注意事項を次に示します。

- スケジューリング機能で使用する CTM デーモンでは，J2EE サーバに対するクライアントからのリクエストを負荷分散させないでください。
- 複数の CTM デーモン間でバッチサーバに対するリクエストを負荷分散させないでください。複数の CTM デーモンに接続するバッチサーバでは，それぞれ異なるスケジュールグループ名を指定してください。

  複数の CTM デーモン間でバッチサーバに対するリクエストを負荷分散した場合，バッチアプリケーションの実行リクエストは受け付けられますが，次のような問題が発生することがあります。

  - バッチアプリケーション情報の一覧が表示できない（バッチアプリケーション情報が取得できない）。
  - バッチアプリケーションの強制停止に失敗する。
- バッチアプリケーションの実行リクエストがスケジュールキューからバッチサーバへ渡される間にバッチ強制停止コマンドを実行すると，メッセージ KDJE55016-W が出力されてバッチアプリケーションを強制停止できないことがあります。この場合，バッチ一覧表示コマンドを実行してバッチアプリケーションの状態を確認します。バッチアプリケーションの状態が「running」の場合は，再度バッチ強制停止コマンドを実行してください。
- CTM とバッチサーバの間でタイムアウトが発生すると，メッセージ KDJE55061-E が出力されます。この場合，バッチアプリケーションの実行リクエストおよび実行中のバッチアプリケーションは，CTM の管理対象外となります。この場合に，実行中のバッチアプリケーションに対して一覧表示や強制停止を実行するときは，バッチサーバ名を指定して各コマンドを実行してください。なお，バッチ一覧表示コマンドは，スケジューリング機能を使用しない設定に変更してから実行してください。スケジューリング機能を使用しない設定にする場合は，usrconf.cfg（バッチアプリケーション用オプション定義ファイル）で batch.ctm.enabled キーに「false」を指定してください。

  各コマンドに指定するバッチサーバ名は，バッチ実行コマンドのメッセージ（KDJE55066-I）で特定できます。
- バッチアプリケーションがバッチサーバで開始する前に，［Ctrl］＋［C］の入力やタイムアウトの発生などによってバッチ実行コマンドが終了されると，メッセージ KDJE55007-E がメッセージログに出力され，バッチアプリケーションの開始に失敗します。この場合には，メッセージ KDJE40062-E が標準エラー出力に出力されることがあります。

# 5 J2EE サーバ間のセッション情報の引き継ぎ

この章では，J2EE サーバ間のセッション情報を引き継ぐための機能である，セッションフェイルオーバ機能の概要，前提条件，およびメモリの見積もりについて説明します。

## 5.1　この章の構成

J2EE サーバ間のセッション情報を引き継ぐには，セッションフェイルオーバ機能を使用します。ここでは，セッションフェイルオーバ機能の概要およびセッションフェイルオーバ機能の種類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 5-1　この章の構成（セッションフェイルオーバ機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | セッションフェイルオーバ機能の概要 | 5.2 |
| | グローバルセッションを利用したセッション管理 | 5.3 |
| | 前提条件 | 5.4 |
| | データベースセッションフェイルオーバ機能 | 5.5 |
| | セッションフェイルオーバ機能使用時に設定できる機能 | 5.6 |
| | セッションフェイルオーバ機能使用時に実行される機能 | 5.7 |
| | メモリの見積もり | 5.8 |
| 注意事項 | 注意事項 | 5.9 |

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。

## 5.2　セッションフェイルオーバ機能の概要

**セッションフェイルオーバ機能**とは，J2EE サーバや Web サーバでの，ソフトウェア障害，ハードウェア障害，およびネットワーク障害の発生時に，J2EE サーバ上の HttpSession オブジェクトに登録されたオブジェクトを引き継ぐ機能です。

セッションフェイルオーバ機能を使用すると，システム内の特定の J2EE サーバに障害が発生した場合に，障害発生前のセッション情報を引き継いで，ほかの J2EE サーバで業務を続行できます。これによって，システムの可用性を高めることができます。

ここでは，セッションフェイルオーバ機能を利用する利点と，セッションフェイルオーバ機能の種類について説明します。

### 5.2.1　セッションフェイルオーバ機能を利用する利点

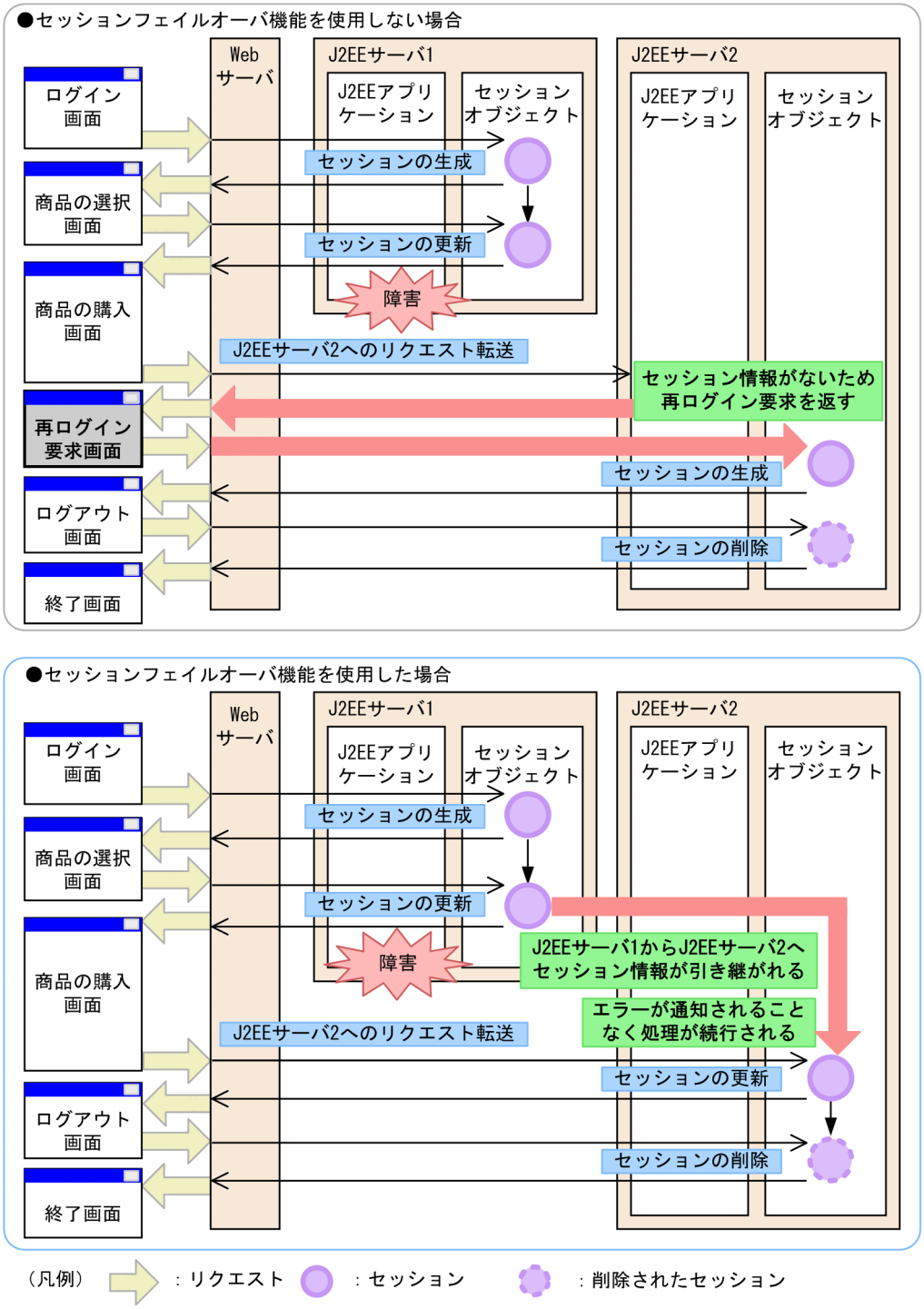
HttpSession オブジェクトは，J2EE サーバのメモリ上で保持されています。J2EE サーバで障害が発生すると，HttpSession オブジェクトは失われます。複数の J2EE サーバで構成されているシステムの場合に，一つの J2EE サーバで障害が発生すると，リクエストはほかの J2EE サーバに転送されます。しかし，HttpSession オブジェクトが失われるため，HttpSession オブジェクトに登録された情報（**セッション情報**）は引き継がれません。このため，リクエストが転送された J2EE サーバ上の J2EE アプリケーションでは，新規のセッションとして扱うことになります。例えば，ユーザ認証処理後の画面で障害が発生すると，再ログインが必要になります。

セッションフェイルオーバ機能を使用すると，セッション情報を管理し，J2EE サーバで障害が発生した場合には，管理しているセッション情報をほかの J2EE サーバに引き渡せます。このため，J2EE サーバで障害が発生し，ほかの J2EE サーバにリクエストが転送された場合でも，障害発生前の状態で業務を続行できます。

また，統合ユーザ管理を使用している場合でも，セッションフェイルオーバ機能を使用してログイン状態を別の J2EE サーバに引き継げます。

セッションフェイルオーバ機能を使用しない場合と使用した場合の処理の流れを次の図に示します。

図 5-1　セッションフェイルオーバ機能を使用しない場合と使用した場合の処理の流れ




セッションフェイルオーバ機能を使用していないときにサーバに障害が発生すると，セッション情報が失われるため，再ログインが必要になります。

セッションフェイルオーバ機能を使用すると，セッション情報がサーバ間で引き継がれるため，ブラウザでのユーザの操作では，サーバの障害発生に気づくことなく，処理を続行できます。

### 5.2.2　セッションフェイルオーバ機能の種類

セッションフェイルオーバ機能の種類にはセッション情報の格納先によって次の種類があります。

- データベースセッションフェイルオーバ機能

  セッション情報をデータベースに格納して管理します。

  データベースセッションフェイルオーバ機能の概要については，「5.5.1　データベースセッションフェイルオーバ機能の概要」を参照してください。

  適用手順，処理の流れや設定については，「6.　データベースセッションフェイルオーバ機能」を参照してください。

## 5.3　グローバルセッションを利用したセッション管理

ここでは，セッションフェイルオーバ機能で管理するグローバルセッション情報について説明します。また，グローバルセッション情報として引き継げる HTTP セッションの属性に関する条件や注意事項についても説明します。

### 5.3.1　グローバルセッション情報

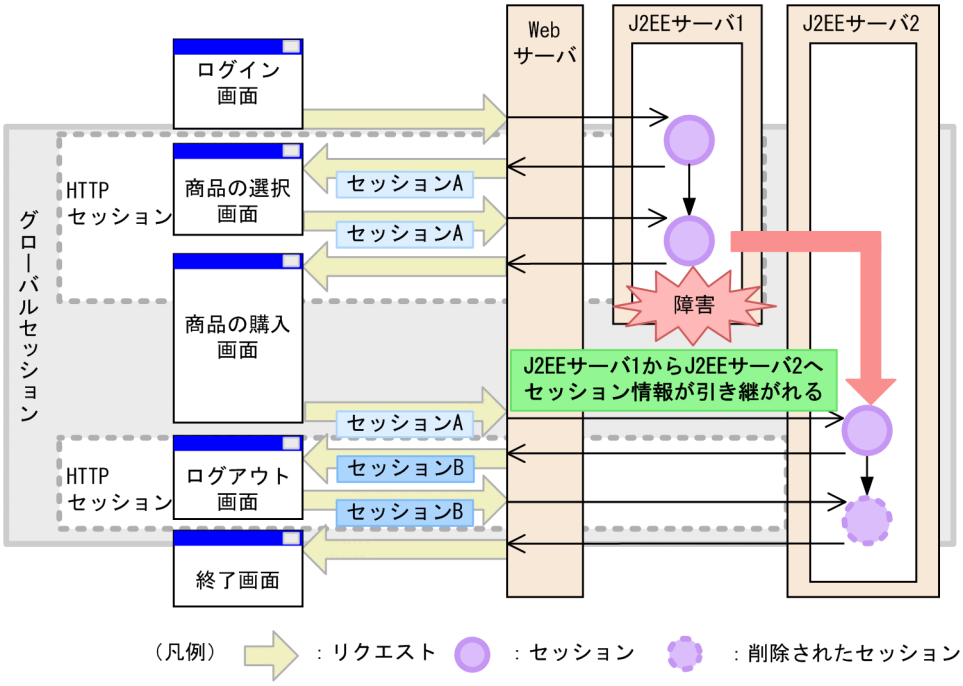
セッションフェイルオーバ機能では，J2EE サーバ上の HttpSession オブジェクトに登録されたオブジェクトの情報を，ほかの J2EE サーバに引き継ぎます。

複数の J2EE サーバ間で引き継いで使用できるセッションを，**グローバルセッション**といいます。HTTP セッションは，そのセッションを扱っている J2EE サーバに障害が発生すると消失します。一方，グローバルセッションは，J2EE サーバとは別のプロセスで管理されているため，J2EE サーバに障害が発生した場合も消失しません。このため，一つの J2EE サーバで障害が発生した場合に，別の J2EE サーバに HTTP セッションを作成して，グローバルセッションの情報を引き継げます。

グローバルセッションを使用している場合に，ほかの J2EE サーバに引き継がれる HttpSession オブジェクトの情報を，**グローバルセッション情報**といいます。

HTTP セッションとグローバルセッションの範囲について，次の図に示します。

図 5-2　HTTP セッションとグローバルセッションの範囲




セッションフェイルオーバ機能では，J2EE サーバでの障害発生時にグローバルセッション情報を引き継ぐことで，ユーザにエラーを通知することなく，障害発生前の状態で業務を続行できます。

## 5.3.2　グローバルセッション情報に含まれる情報

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，グローバルセッション情報は，データベースに作成するセッション情報格納テーブルのレコードに格納されます。グローバルセッション情報が格納される際，HTTP セッションごとに一つのレコードが割り当てられます。

グローバルセッション情報には，次の表に示す情報が含まれます。

表 5-2　グローバルセッション情報に含まれる情報

| 項番 | 冗長化の対象 | 説明 |
|---|---|---|
| 1 | セッション ID | グローバルセッション情報を管理するセッション ID です。 |
| 2 | HTTP セッションの属性情報 | HTTP セッションに登録された属性および関連づけられたすべての属性について，属性名および属性値のオブジェクトをシリアライズしたバイト配列に変換した情報です。 |
| 3 | HTTP セッションの作成時刻 | HTTP セッションが作成された時刻です。グローバルセッションの引き継ぎが発生した場合，引き継ぎ前の HTTP セッションの作成時刻をそのまま使用します。 |
| 4 | HTTP セッションの有効期限 | HTTP セッションに設定された有効期限です。 |
| 5 | 最終アクセス時刻 | HTTP セッションを使用するリクエストが最後に送信された時刻です。 |
| 6 | HTTP セッションの所有 J2EE サーバ識別子 | HTTP セッションを作成，または引き継ぎをした J2EE サーバのサーバ ID です。 |

**参考**

- データベースセッションフェイルオーバ機能で使用するデータベースのテーブルには，Web アプリケーションの設定情報を格納するアプリケーション情報テーブル，グローバルセッション情報を格納するセッション情報格納テーブル，およびセッション情報格納テーブルの未使用レコードを管理する空きレコード情報テーブルがあります。

## 5.3.3　グローバルセッション情報として引き継げる HTTP セッションの属性

ここでは，障害発生時に引き継げる HTTP セッションの属性に関する次の項目について説明します。

- 引き継げる HTTP セッションの属性の条件
- 引き継ぎ対象としてサポートされるオブジェクト
- オブジェクトの内容によるセッション情報の引き継ぎ可否
- HTTP セッションの属性引き継ぎ時のシリアライズ処理についての注意事項
- HTTP セッションの属性引き継ぎ時のデシリアライズ処理についての注意事項

- HTTP セッションの属性引き継ぎ時の注意事項

## (1) 引き継げる HTTP セッションの属性の条件

セッションフェイルオーバ機能では，グローバルセッション情報の更新処理でオブジェクトのシリアライズ，引き継ぎ処理でオブジェクトのデシリアライズ処理が発生します。そのため，HTTP セッションに登録する属性は，次の条件を満たす必要があります。

- java.io.Serializable インタフェースを実装した直列化可能クラスのオブジェクトである。

## (2) 引き継ぎ対象としてサポートされるオブジェクト

セッションフェイルオーバ機能では，次に示す直列化可能クラスのオブジェクトを引き継ぎ対象としてサポートしています。

- J2EE アプリケーションで提供されるクラスのオブジェクト。
- J2SE で提供されるクラスのオブジェクト。

ただし，引き継ぎ処理では，HTTP セッションに登録された直列化可能クラスのオブジェクトが，セッションフェイルオーバ機能でサポートされているオブジェクトかどうかはチェックされません。

## (3) オブジェクトの内容によるセッション情報の引き継ぎ可否

HTTP セッションに登録されたオブジェクトの内容によるセッション情報の引き継ぎ可否を次の表に示します。

表 5-3 HTTP セッションに登録されたオブジェクトによるセッション情報の引き継ぎ可否

| 項番 | HTTP セッションに登録されたオブジェクトの内容 | | セッション情報の引き継ぎ可否 | グローバルセッション情報の格納 |
|---|---|---|---|---|
| | java.io.Serializable インタフェースの実装の有無 | シリアライズの成功/失敗 | | |
| 1 | java.io.Serializable インタフェースの実装あり | シリアライズの成功 | 引き継ぎできます。 | シリアライズ後の情報がデータベースに格納されます。 |
| 2 | | シリアライズの失敗 | シリアライズに失敗した属性を含む HTTP セッションは，グローバルセッションの引き継ぎの対象とならないため，引き継ぎできません。 | KDJE34318-E または KDJE34411-E のメッセージが出力され，データベースにグローバルセッション情報は格納されません。<br>次回以降のリクエスト処理完了後に，HTTP セッションに |

| 項番 | HTTP セッションに登録されたオブジェクトの内容 | | セッション情報の引き継ぎ可否 | グローバルセッション情報の格納 |
|---|---|---|---|---|
| | java.io.Serializable インタフェースの実装の有無 | シリアライズの成功/失敗 | | |
| | | | | 登録されたオブジェクトがシリアライズ可能となった時点で，データベース上にグローバルセッション情報が格納されます。 |
| 3 | java.io.Serializable インタフェースの実装なし | （シリアライズできません） | シリアライズできない属性はグローバルセッションの引き継ぎの対象とならないため，引き継ぎできません。 | シリアライズできないオブジェクトが存在したときは，KDJE34317-W または KDJE34410-W のメッセージが出力され，シリアライズできない属性を除いた属性で作成したグローバルセッション情報がデータベースに格納されます。 |

## (4) HTTP セッションの属性引き継ぎ時のシリアライズ処理についての注意事項

シリアライズ処理についての注意事項を次に示します。

### (a) シリアライズ処理が性能に与える影響

引き継ぎ対象のオブジェクトだけでなく，引き継ぎ対象のオブジェクトから参照されるオブジェクトすべてを対象としてシリアライズ処理が実行されます。このため，引き継ぐ必要がない情報を含むクラスなどを HTTP セッションに登録した場合，性能が低下するおそれがあります。

### (b) java.lang.OutOfMemoryError エラーが発生する場合

シリアライズ処理では，一時的に，アプリケーションで設定した HttpSession オブジェクト数を超えてシリアライズ後のデータが作成されます。そのため，巨大なオブジェクトが HTTP セッションに登録された場合，グローバルセッション情報の作成中に java.lang.OutOfMemoryError エラーが発生することがあります。

### (c) シリアライズに失敗する場合とその対処

次のような場合は，KDJE34317-W，KDJE34318-E，KDJE34410-W，または KDJE34411-E のメッセージが出力され，シリアライズに失敗します。

- HTTP セッションに登録したオブジェクト（直列化可能クラスのオブジェクト）から参照するオブジェクトに，直列化可能クラス以外のクラスを実装したオブジェクトが含まれる場合。
- オブジェクトに writeObject(java.io.OutputStream out)メソッドが実装されており，シリアライズ時に例外が発生する場合。

失敗した場合，グローバルセッション情報の更新，および引き継ぎ処理が実行されません。処理を実行するためには，次のどちらかの対処が必要です。

- シリアライズに失敗したオブジェクトの HTTP セッションへの登録を解除する。
- シリアライズに失敗したオブジェクトを変更して，シリアライズに失敗した原因を取り除く。

## (5) HTTP セッションの属性引き継ぎ時のデシリアライズ処理についての注意事項

次のような場合は，デシリアライズに失敗します。

- デシリアライズに失敗する原因となる変更が Web アプリケーションに加えられ，シリアライズ時と Web アプリケーションが異なっている場合。
- オブジェクトに readObject(java.io.OutputStream out)メソッドが実装されており，デシリアライズ時に例外が発生する場合。

リクエストの受信時，または Web アプリケーション開始時のグローバルセッション情報の引き継ぎ処理でセッション情報のデシリアライズに失敗した場合，グローバルセッション情報およびセッション情報が削除され，KDJE34326-E，または KDJE34413-E が出力されます。セッションの引き継ぎに失敗するため，HTTP セッションがない状態でリクエストが処理されます。

## (6) HTTP セッションの属性引き継ぎ時の注意事項

完全性保証モードを使わない，かつ同じセッションに対して同時にリクエストが実行される構成である場合，セッションにはスレッドアンセーフなオブジェクトを格納してはなりません（ユーザプログラム内で synchronized 句などを利用したスレッドセーフな実装をしていても，Cosmeinxsus 側の処理との競合は防げません）。

## 5.4　前提条件

セッションフェイルオーバ機能を使用するための前提条件について説明します。

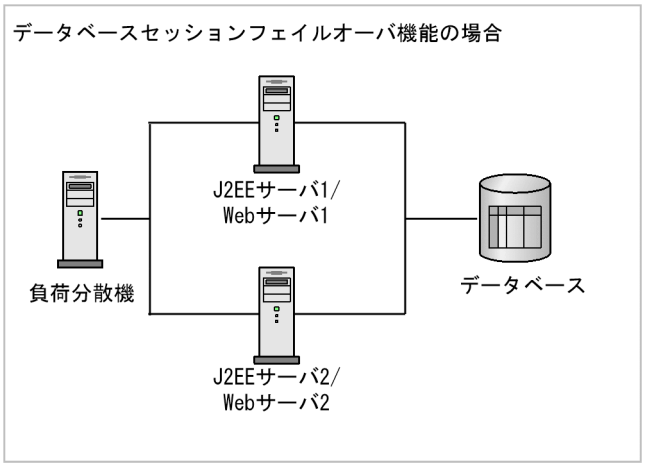
### 5.4.1　前提となる構成

セッションフェイルオーバ機能を使用する場合，負荷分散機を使用した，複数の J2EE サーバにリクエストを振り分けるシステム構成が前提となります。また，各 J2EE サーバで作成された HTTP セッションの情報を格納するためのデータベースの配置が必要です。

なお，データベースセッションフェイルオーバ機能では Oracle RAC を使用した Oracle への接続はサポートしません。

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の前提となる構成を次の図に示します。

図 5-3　セッションフェイルオーバ機能の前提となる構成




- 負荷分散機

  セッションフェイルオーバ機能を使用するには，負荷分散機の使用が前提となります。

  **参考**

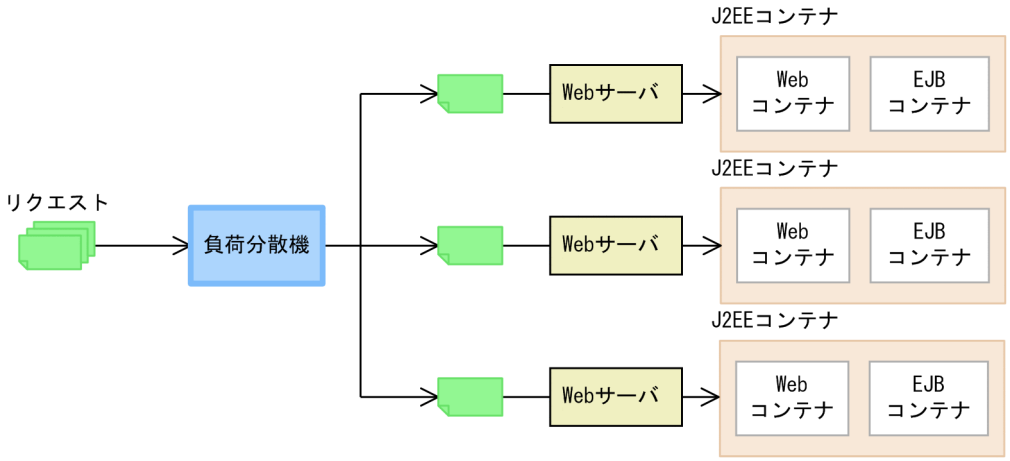
  **負荷分散機によるリクエストの振り分けとは**

  **負荷分散機**では，リクエストの振り分けが実行されます。これによって，負荷が分散されるので，システムの安定稼働と処理性能の向上が図れます。

  負荷分散機を使用した負荷分散には，Web サーバや J2EE サーバに負荷分散処理に関する負荷が掛からないという利点があります。なお，負荷分散機によるリクエストの振り分け方式は，それぞれの負荷分散機に依存します。

  負荷分散機によるリクエスト振り分けの例を次の図に示します。

図 5-4　負荷分散機によるリクエストの振り分けの例




- J2EE サーバ/Web サーバ

  セッションフェイルオーバ機能を使用する場合，一つのシステムで一つ以上の J2EE サーバおよび Web サーバを配置します。J2EE サーバ障害に備え，二つ以上配置することを推奨します。

- データベース

  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，セッション情報の格納先としてデータベースが必要です。セッション情報の格納先として使用できるデータベース，JDBC ドライバ，およびリソースアダプタの対応について次の表に示します。

表 5-4　使用できるデータベース，JDBC ドライバおよびリソースアダプタの対応

| データベース | JDBC ドライバ | リソースアダプタ※ |
|---|---|---|
| HiRDB | HiRDB Type4 JDBC Driver | DBConnector_HiRDB_Type4_CP.rar |
| Oracle | Oracle JDBC Thin Driver | DBConnector_Oracle_CP.rar |

注※　データベースセッションフェイルオーバ機能で使用するリソースアダプタは DB Connector です。データベースセッションフェイルオーバ機能で使用する DB Connector に必要な設定については，「6.6.4　DB Connector の設定」を参照してください。

なお，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ための詳細なシステム構成については，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「3.10.1　データベースを使用する構成（データベースセッションフェイルオーバ機能）」を参照してください。

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構成は，ここで示した条件を満たしている場合，構成から設計し直す必要がありません。機能の設定とパラメタチューニングを実施すればデータベースセッションフェイルオーバ機能を使用できます。

## 5.4.2　前提となる設定

セッションフェイルオーバ機能を使用するための前提となる設定について説明します。

## (1) セッションフェイルオーバ機能共通の前提となる設定

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，次の設定が必要です。

- HttpSession のサーバ ID 付加機能によるセッション ID へのサーバ ID 付加

  HTTP セッションのセッション ID にサーバ ID を付加する機能です。データベースセッションフェイルオーバ機能（完全性保障モード無効）を使用する場合は，この機能を有効にする必要があります。また，冗長化した J2EE サーバごとに，異なるサーバ ID※を設定します。

  HttpSession のサーバ ID 付加機能を無効にした場合，Web アプリケーションの開始時に KDJE34371-E または KDJE34404-E のエラーメッセージがメッセージログに出力され，Web アプリケーションの開始に失敗します。冗長化した J2EE サーバごとに異なるサーバ ID※を設定しなかった場合は，意図しない J2EE サーバにグローバルセッション情報が引き継がれ，グローバルセッション情報の完全性が失われることがあります。

  HttpSession のセッション ID へのサーバ ID の付加機能については，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(Web コンテナ)」の「2.7.6　セッション ID および Cookie へのサーバ ID の付加」を参照してください。

  注※　実行系と待機系による系切り替えシステムの場合，実行系と待機系の値は同じにしてください。

- HTTP セッションのスティッキー（Sticky）の設定

  負荷分散機を使用する環境でセッションフェイルオーバ機能を使用するには，HTTP セッションのスティッキーを設定する必要があります。

  HTTP セッションのスティッキーを設定しない場合，HTTP セッションを保持したリクエストの振り分け先が固定されません。そのため，HTTP セッションを保持したリクエストを受け付けるたびに HTTP セッションの引き継ぎ処理が実施され，性能が劣化するおそれがあります。

- ホストの時刻の設定

  セッションフェイルオーバ機能を使用するには，システム内の J2EE サーバが稼働するそれぞれのホストに同じ時刻を設定してください。

  データベースに格納するセッション情報には，HTTP セッションの作成時刻や，最終アクセス時刻などの情報が含まれます。各ホストでの設定時刻が異なっている場合，セッション情報に自ホストの設定時刻と異なる情報が含まれます。そのため，セッションの引き継ぎが発生した場合に HTTP セッションの制御に問題が発生するおそれがあります。

## (2) データベースセッションフェイルオーバ機能の前提となる設定

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，次の設定が必要です。

- Web クライアントが保持する無効なセッション ID の削除

  HTTP セッション無効化時に Web クライアントに保持されている HTTP Cookie の情報を削除し，無効化済みの HTTP セッションに対するセッション ID の送信を抑止する機能です。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には，この機能を有効にする必要があります。

  HTTP セッションのセッション ID を示す HTTP Cookie の削除を無効にしている場合，Web アプリケーション開始時に KDJE34339-E のエラーメッセージがメッセージログに出力され，Web アプリ

ケーションの開始に失敗します。HTTP セッションのセッション ID を示す HTTP Cookie の削除については，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「2.7.4 Web クライアントが保持する無効なセッション ID の削除」を参照してください。

- **HttpSession オブジェクト数の上限値の指定**

  有効となる HttpSession オブジェクト数の上限値を設定する機能です。この機能は，完全性保障モードを有効にする場合に設定します。

  アプリケーション開始時に実施されるネゴシエーション処理に失敗した場合に，Web アプリケーションの開始処理を中止する設定にしているときは，上限値に 1 以上の有効な値を設定する必要があります。HttpSession オブジェクト数の上限値を設定していない場合，アプリケーション開始時に KDJE34303-E のエラーメッセージがメッセージログに出力され，アプリケーションの開始に失敗します。

  ただし，ネゴシエーション処理に失敗した場合に，Web アプリケーションの開始処理を続行する設定にしているときは，HttpSession オブジェクト数の上限値の設定は任意です。また，上限値に-1（無制限）を設定することもできます。HTTPSession オブジェクト数の上限値に-1（無制限），またはデータベースのセッション情報格納テーブルのレコード数よりも大きな値を設定した場合，HttpSession オブジェクト数がセッション情報格納テーブルのレコード数を超過したときの動作は，次のようになります。

  完全性保障モードが無効の場合（任意）

  該当する HTTP セッションは縮退して，リクエスト処理が続行されます。

  完全性保障モードが有効の場合

  KDJE34380-E のエラーメッセージがメッセージログに出力され，該当する HTTP セッションは作成されません。

  HttpSession オブジェクト数の上限値の設定については，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「2.7.5　HttpSession オブジェクト数の上限値の設定」を参照してください。

  ネゴシエーション処理については，「6.4.1　アプリケーション開始時の処理」を参照してください。

  完全性保障モードについては，「5.5.1(4)　データベースセッションフェイルオーバ機能の運用モード」を参照してください。

  完全性保障モードが無効の場合の HTTP セッションの縮退については，「5.7.3　HTTP セッションの縮退」を参照してください。

- **デフォルトの実行待ちキュー，Web アプリケーション単位の実行待ちキュー，URL グループ単位の実行待ちキューの設定**

  Web アプリケーション単位の同時実行スレッド数の制御機能が有効の場合，デフォルトの実行待ちキュー，Web アプリケーション単位の実行待ちキュー，URL グループ単位の実行待ちキューの空きが不足したときに，クライアントに 503 エラーを返すかどうかを簡易構築定義ファイルの webserver.dbsfo.thread_control_queue.enabled パラメタに設定します。なお，デフォルトでは，クライアントに 503 エラーを返します。

  クライアントに 503 エラーを返さない設定にした場合，その実行待ちキューサイズには十分に大きな値を設定してください。

クライアントに 503 エラーを返す設定にした場合，web.xml で指定するエラーページで次に示す HTTP セッションの更新をしないでください。

- HTTP セッションの作成

  Web アプリケーションが HTTP セッションを作成した場合，javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出し元に，com.hitachi.software.web.dbsfo.SessionOperationException 例外がスローされ，HTTP セッションは作成されません。

- HTTP セッションの有効期限の変更（javax.servlet.http.HttpSession インタフェースの setMaxInactiveInterval メソッドの呼び出し）

  Web アプリケーションが HTTP セッションの有効期限を変更した場合，データベース上のグローバルセッションの有効期限は変更されません。グローバルセッションの引き継ぎが発生すると，有効期限は変更前の状態に戻ります。

- HTTP セッションの属性情報の変更

  Web アプリケーションが HTTP セッションの属性情報を変更した場合，データベース上のグローバルセッション情報は変更されません。グローバルセッションの引き継ぎが発生すると，属性情報は変更前の状態に戻ります。

- HTTP セッションの無効化（javax.servlet.http.HttpSession インタフェースの invalidate メソッドの呼び出し）

  Web アプリケーションが javax.servlet.http.HttpSession インタフェースの invalidate メソッドを呼び出した場合，com.hitachi.software.web.dbsfo.SessionOperationException 例外がスローされます。

- **ユーザ指定名前空間機能**

  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，ユーザ指定名前空間機能を利用して付与した別名での J2EE リソースのルックアップが前提となります。

  このため，J2EE サーバのプロパティに次のパラメタを指定してラウンドロビン検索機能を使用している場合，データベースセッションフェイルオーバ機能は使用できません。

  java.naming.factory.initial=com.hitachi.software.ejb.jndi.GroupContextFactory

  このパラメタを指定している場合，Web アプリケーション開始時に KDJE34305-E のエラーメッセージがメッセージログに出力され，Web アプリケーションの開始に失敗します。

  J2EE サーバで動作する J2EE アプリケーションでラウンドロビン検索が必要な場合は，InitialContextFactory の実装を委譲しているクラスを J2EE サーバのプロパティで指定しないで，各アプリケーションの InitialContext 生成時に引数で指定する必要があります。ラウンドロビン検索機能については，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「2.7　ラウンドロビンポリシーによる CORBA ネーミングサービスの検索」を参照してください。

## 5.5　データベースセッションフェイルオーバ機能

J2EE サーバ間のセッション情報の引き継ぎを実現するための機能として，データベースセッションフェイルオーバ機能があります。ここでは，データベースセッションフェイルオーバ機能の概要について説明します。

### 5.5.1　データベースセッションフェイルオーバ機能の概要

データベースセッションフェイルオーバ機能は，セッション情報をデータベースで管理することで，障害が発生した場合に J2EE サーバ間のセッション情報の引き継ぎを実現するための機能です。障害が発生した場合，データベースに格納されているセッション情報を基にセッションを再作成し，正常に業務を続行できます。

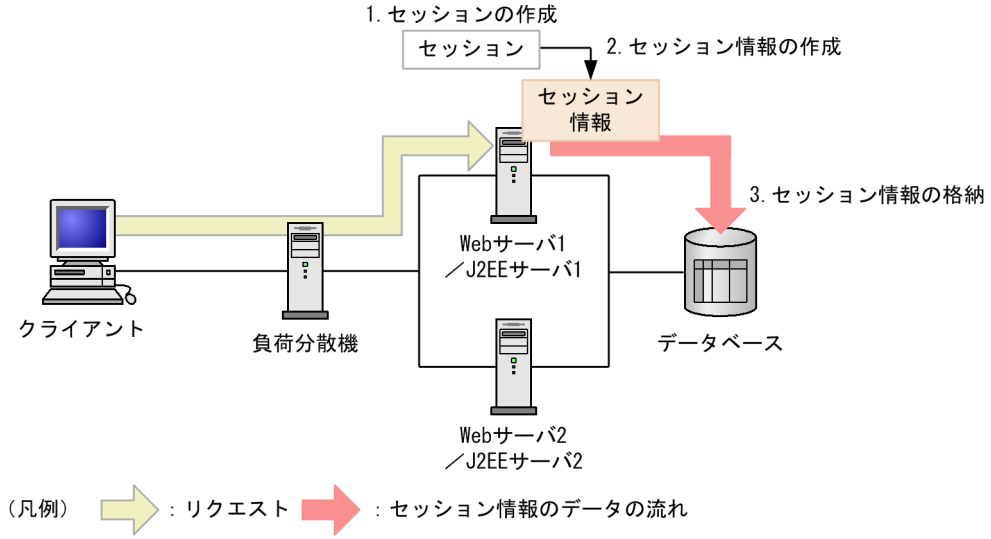
データベースセッションフェイルオーバ機能の処理の概要，運用モードについて説明します。

#### (1)　セッション情報の格納の流れ

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と，リクエストによるセッションの作成処理が発生したときに，処理の延長上でセッション情報がデータベースに格納されます。

セッション情報の格納の流れを次の図に示します。

図 5-5　セッション情報の格納の流れ（データベースセッションフェイルオーバ機能）




項番は図中の番号と対応しています。

1. Web サーバがクライアントからセッションの作成が必要なリクエストを受け取ると，J2EE サーバでセッションが作成されます。
2. 作成されたセッションについて，セッション情報が作成されます。
3. セッション情報がデータベースに格納されます。

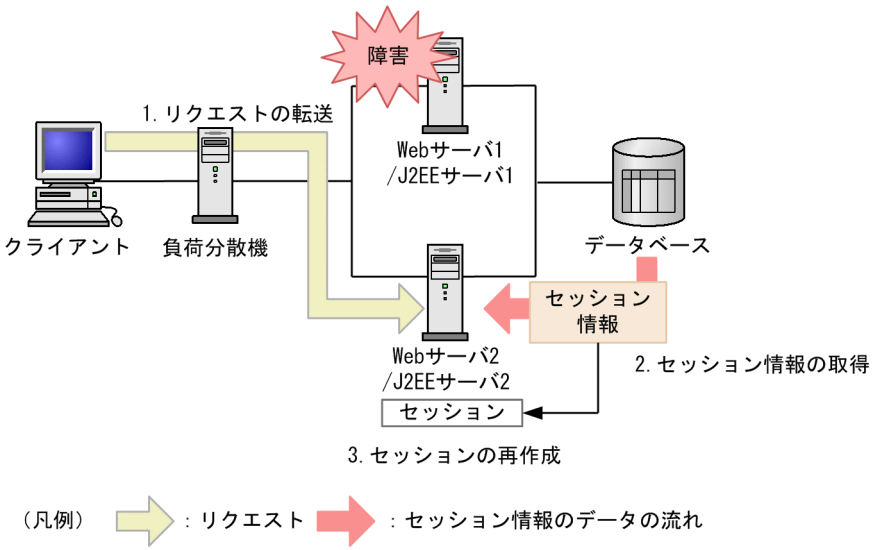
Web サーバ 1 または J2EE サーバ 1 で障害が発生すると，データベースに格納されたセッション情報が Web サーバ 2 または J2EE サーバ 2 へ引き継がれ，障害発生前の状態で業務を続行できます。

## (2) Web サーバまたは J2EE サーバで障害が発生した場合の処理の流れ

Web サーバまたは J2EE サーバで障害が発生した場合，データベースに格納されているセッション情報を基に，ほかの J2EE サーバでセッションを再作成し，正常に業務を続行できます。

Web サーバまたは J2EE サーバで障害が発生した場合の処理の流れについて，次の図に示します。

図 5-6 Web サーバまたは J2EE サーバで障害が発生した場合の処理の流れ（データベースセッションフェイルオーバ機能）




1. Web サーバ 1 で障害が発生すると，負荷分散機によって Web サーバ 2 へリクエストが転送されます。
2. 転送先の J2EE サーバ 2 でリクエストを処理するとき，リクエストに関連づけられたセッションが存在しないため，データベースからセッション情報を引き継ぎます。
3. セッションが再作成されます。

セッションが正常に引き継がれ，障害発生前の状態で業務を続行できます。

また，J2EE サーバ 1 を再起動し Web サーバ 1 が障害から回復すると，再びリクエストは Web サーバ 1 に送信されます。

## (3) データベースで障害が発生した場合の処理の流れ

データベースで障害が発生すると，J2EE サーバ上のセッション情報だけを操作して業務を続行できます。データベースが障害から回復し，それ以降のセッションの操作でデータベースにアクセスできたとき，J2EE サーバ上で操作したセッション情報でデータベースを更新します。

これによって，クライアントはデータベースに障害が発生したことを意識しないで業務を続行できます。

## (4) データベースセッションフェイルオーバ機能の運用モード

データベースに格納済みのグローバルセッション情報に対して同じセッション ID を持つリクエストが複数同時に送信された場合，デフォルトの設定では，リクエストは複数同時に処理できます。これによって，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ことによる処理性能の低下は抑えられています。

ただし，この運用は，冗長化された複数の J2EE サーバから同じセッション ID のグローバルセッション情報を同時に更新するような処理が発生しないことを前提とした運用です。複数の J2EE サーバから同じセッション ID のグローバルセッション情報が同時に更新された場合は，グローバルセッション情報の整合性が失われるおそれがあります。このようなケースを許容できないシステムでは，グローバルセッション情報の整合性を確保するためのモードを有効にする必要があります。

グローバルセッション情報の整合性を保障するモードを**完全性保障モード**といいます。このモードを有効にした場合，グローバルセッションを更新するたびにデータベースにロックが設定されます。同じセッション ID の複数のリクエストが同時に送信された場合は，リクエストがシリアルに処理されるため，グローバルセッション情報が不整合な状態になることはありません。ただし，複数同時に実行できないこと，およびグローバルセッション情報の格納のたびにロックの設定・解除処理が発生することから，リクエスト処理性能に影響を及ぼす場合があります。

このため，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は，システムの目的や特性に応じて，どちらのモードで運用するかを検討する必要があります。

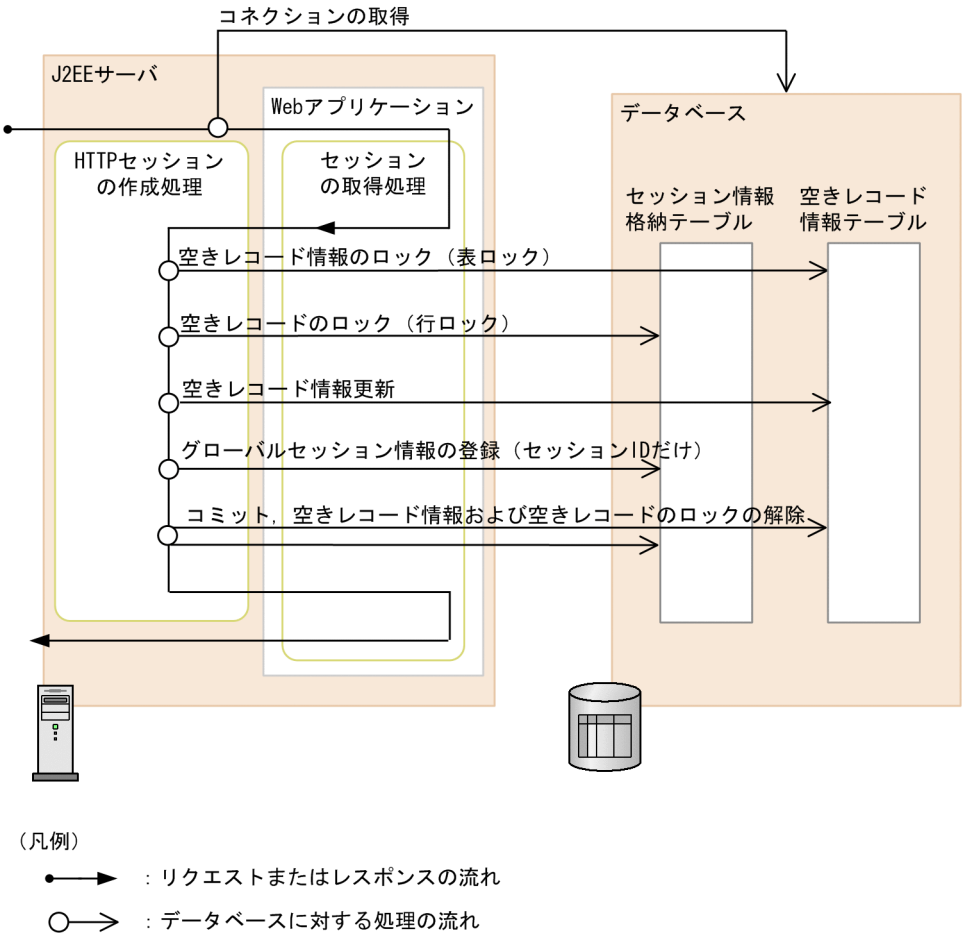
完全性保障モードの有効・無効による主な違いを次の表に示します。

### 表 5-5　完全性保障モードの有効・無効による主な違い

| 比較項目 | 完全性保障モード | |
|---|---|---|
| | 無効 | 有効 |
| 適したシステムの特性 | 性能を重視するシステムに適しています。 | 性能が低下したとしても，確実なセッション情報の引き継ぎが必要なシステムに適しています。 |
| リクエストの処理性能 | 同じセッション ID の複数のリクエストを同時に処理できるので，優れています。 | リクエストをシリアルに処理する必要があるため，性能が低下します。 |
| グローバルセッション情報の完全性 | 同じセッション ID のグローバルセッション情報を同時に更新した場合は，保障されません。 | 保障されます。 |
| データベースに障害が発生した場合の挙動 | J2EE サーバ上のセッション情報を使用して処理を継続します（データベースセッションフェイルオーバ機能の縮退運用）。 | エラーメッセージを出力して処理を停止します。 |

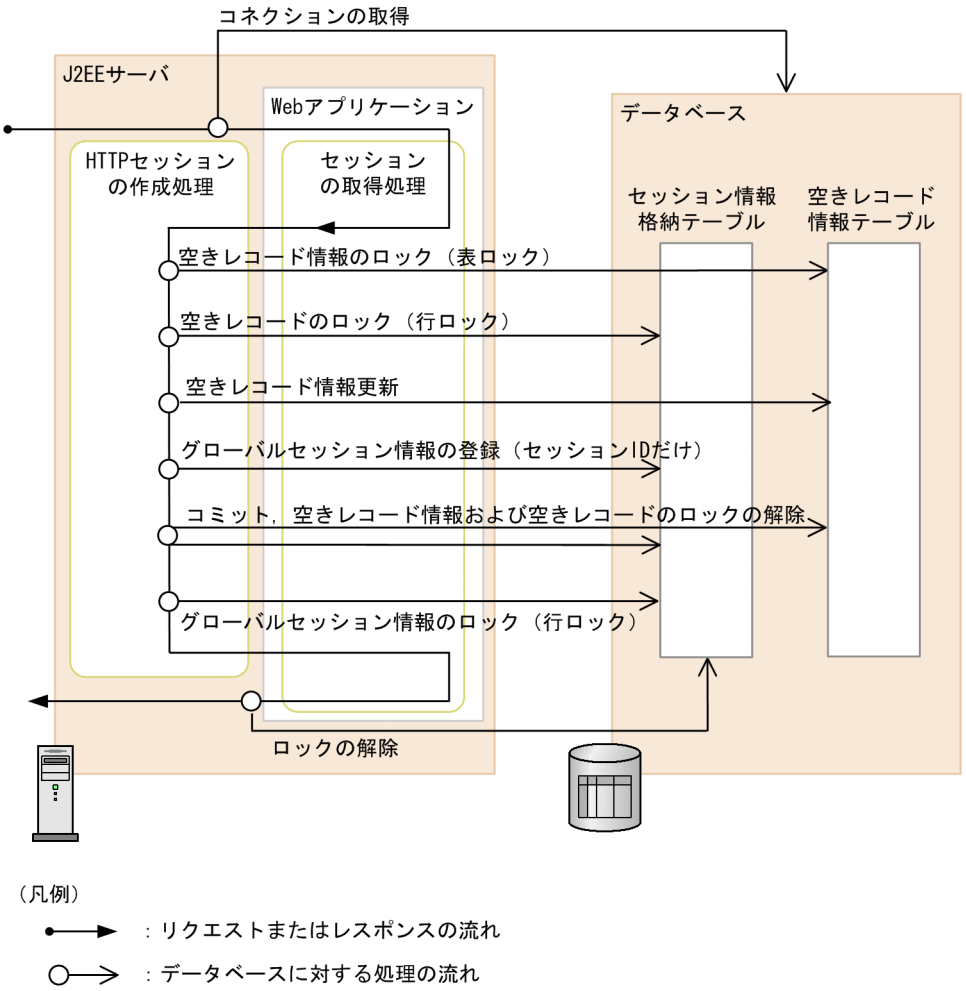
それぞれのモードでのリクエスト処理の流れを次の図に示します。

図 5-7　完全性保障モードを無効にした場合のリクエスト処理の流れ（デフォルトの設定）




完全性保障モードが無効の場合，HTTP セッション作成処理の延長でデータベース上にグローバルセッション情報を作成する際にデータベースのロックが取得・解除されますが，一度コミットしたあとのセッション取得処理ではロックが取得されません。また，以降のグローバルセッション情報の更新処理では，データベースのロックの取得処理および解除処理は実施されません。

図 5-8　完全性保障モードを有効にした場合のリクエスト処理の流れ




完全性保障モードが有効の場合，HTTP セッション作成処理の延長でデータベース上にグローバルセッション情報を作成する際に，データベースのロックが取得・解除されます。さらに，一度コミットしたあとのセッション取得処理で，再度ロックが取得されます。これによって，HTTP セッション作成後，Web アプリケーション実行中に J2EE サーバまたはデータベースで障害が発生した場合にも，データベースに不整合が発生しないことが保障されます。また，以降のグローバルセッション情報の更新処理では，更新するたびにデータベースのロックを取得し，更新したあとで解除する処理が実施されます。

グローバルセッション情報のロック時の動作については，「6.4.5(1)　ロック取得時のロック取得処理の呼び出し結果」を参照してください。

また，完全性保障モードの有効・無効の設定によって，使用できる機能が異なります。

## 5.6　セッションフェイルオーバ機能使用時に設定できる機能

ここではセッションフェイルオーバ機能を使用している場合に設定できる次の機能について説明します。この機能は必要に応じて使用できます。

- セッションフェイルオーバの抑止
- HTTP セッションの参照専用リクエストの定義※

  注※　HTTP セッションの参照専用リクエストの定義機能は，データベースセッションフェイルオーバ機能の完全性保障モード有効時には使用できません。

### 5.6.1　セッションフェイルオーバ機能の抑止

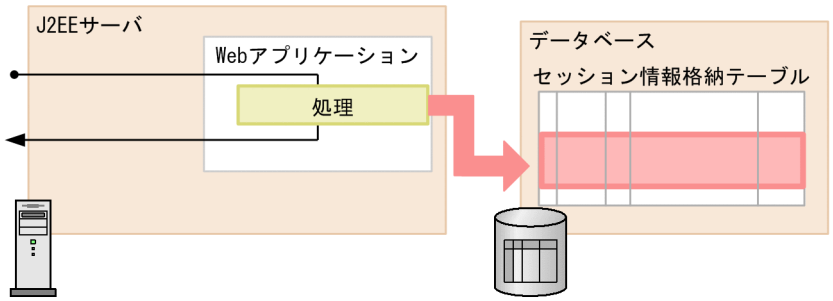
セッションフェイルオーバ機能を有効にした場合，HTTP セッションを取得済みのリクエストを受け付けたときに，データベースへのアクセスや，HTTP セッションのシリアライズなどの処理が実施されます。静的コンテンツや HTTP セッションを必要としないコンテンツに対するリクエストであっても，HTTP セッションを取得済みのリクエストと同一のセッション ID が送信された場合には，セッションフェイルオーバ機能が動作してこれらの不要な処理が発生します。

これに対して，セッションフェイルオーバ機能を抑止する URL パターンを URI または拡張子で設定すると，設定した URL パターンのリクエストに対するセッションフェイルオーバ機能の処理が抑止されるため，不要な処理が発生しなくなり，処理性能が向上します。このように，セッションフェイルオーバ機能を設定している場合に，特定の URL パターンに対してだけセッションフェイルオーバを抑止する機能を**セッションフェイルオーバ抑止機能**といいます。

セッションフェイルオーバ抑止機能の有効・無効と実施される処理の違いを，データベースセッションフェイルオーバ機能を例に次の図に示します。

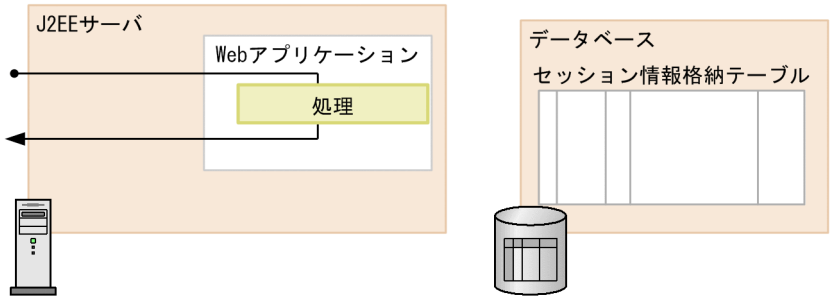
図 5-9　セッションフェイルオーバ抑止機能の有効・無効と実施される処理の違い（データベースセッションフェイルオーバ機能）

●データベースセッションフェイルオーバ機能を抑止しない場合（抑止が無効）




処理を実施し，データベースにセッション情報を格納します。

●データベースセッションフェイルオーバ機能を抑止する場合（抑止が有効）




処理の実施後，データベースにアクセスしないため処理性能が向上します。

（凡例）：リクエストまたはレスポンスの流れ
：グローバルセッション情報

性能向上以外では，次のような目的でセッションフェイルオーバ抑止機能が使用できます。

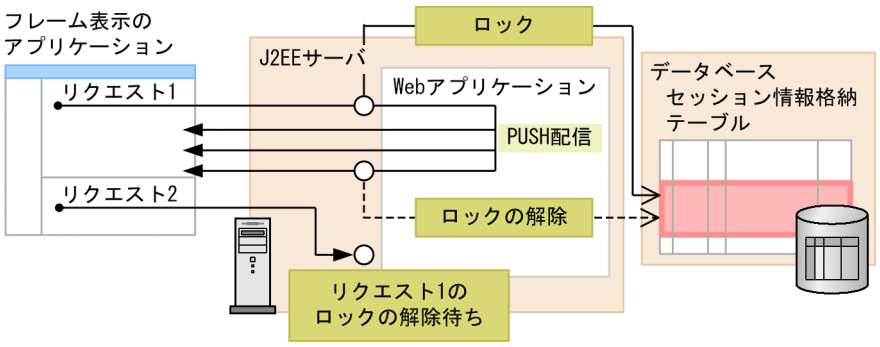
完全性保障モードが有効の場合のデータベースセッションフェイルオーバ機能では，同一セッション ID のリクエストに対して，排他的に処理を実行します。例えば，PUSH 配信をするために常駐するようなサーブレット/JSP など，長時間処理が終了しないサーブレット/JSP を HTML のフレームなどの一つから呼び出した場合，そのサーブレット/JSP の処理が終わるまで，同じフレームから送信されるすべてのリクエストは実行されません。これは，一つのフレームから送信されるリクエストはすべて同一のセッション ID を送信するリクエストになるためです。

このような状態を防ぐためには，HTTP セッションを使用していない特定のリクエストに対して，セッションフェイルオーバ機能を抑止する必要があります。

完全性保障モードを有効にしてデータベースセッションフェイルオーバ機能を使用している場合に，セッションフェイルオーバ抑止機能を有効または無効に設定したときに実施される処理の違いを次の図に示します。なお，図中のリクエスト 1 およびリクエスト 2 は，同一のセッション ID を送信するリクエストです。

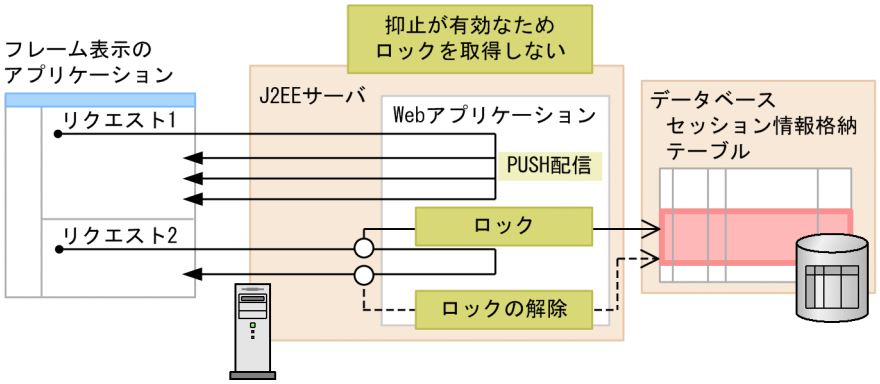
図 5-10　セッションフェイルオーバ抑止機能の有効・無効と実施される処理の違い（データベースセッションフェイルオーバ機能）

●データベースセッションフェイルオーバ機能を抑止しない場合（抑止は無効）




リクエスト1はHTTPセッションを使用しませんが，取得済みのセッションIDを送信するため，
リクエストの処理前にグローバルセッション情報のロックを取得します。
リクエスト1がPUSH配信のために処理が終了しないサーブレット/JSPの場合，
リクエスト2の処理は排他的に処理されるため，リクエスト1の処理が終了するのを
待ち続けます。

●データベースセッションフェイルオーバ機能を抑止する場合（抑止は有効）




HTTPセッションを使用しないリクエスト1のURIをデータベースセッションフェイルオーバ機能の抑止対象とすることで，あとから実行したリクエスト2は，グローバルセッション情報のロック待ちをしないでリクエストを処理できます。

（凡例）
：リクエストまたはレスポンスの流れ
：ロック（レコードの排他取得）の流れ
：ロックの解除（レコードの排他解放）の流れ
：グローバルセッション情報

セッションフェイルオーバ抑止機能の有効・無効は，J2EE サーバ単位または Web アプリケーション単位に設定できます。

- **注意事項**

  セッションフェイルオーバ抑止機能について，注意事項を示します。

  - セッションフェイルオーバ抑止機能によってセッションフェイルオーバ機能が無効となったリクエスト処理内で，javax.servlet.http.HttpServletRequest インタフェースの getSession()メソッドまたは getSession(boolean create)メソッドを呼び出すと，com.hitachi.software.web.dbsfo.SessionOperationException 例外がスローされます。このた

め，このメソッドを呼び出すリクエストに対しては，セッションフェイルオーバ抑止機能は適用できません。

com.hitachi.software.web.dbsfo.SessionOperationException 例外についての詳細は，マニュアル「アプリケーションサーバ リファレンス API 編」の「3.1　例外クラス」を参照してください。

- セッションフェイルオーバ抑止機能によってセッションフェイルオーバ機能が無効となったリクエストは，HTTP セッションを使用するリクエストではないため，HTTP セッションのアクセス時刻が更新されません。これによって，次の影響があります。

  ・javax.servlet.http.HttpSession インタフェースの getLastAccessedTime()メソッドは，前回 HTTP セッションを使用するリクエストが実行された時刻を返します。

  ・現在の時刻と HTTP セッションのアクセス時刻の差がタイムアウト時間を超えた場合，HTTP セッションはタイムアウトします。このため，HTTP セッション作成後に，セッションフェイルオーバ機能が無効になったリクエストだけを送信し続けると，HTTP セッションのタイムアウトが発生する場合があります。

- JSP では，デフォルトで暗黙的に HTTP セッションが作成されます。このため，HTTP セッションを必要としない JSP に対してセッションフェイルオーバ抑止機能を適用する場合は，明示的に page ディレクティブの session 属性を使用して HTTP セッションを作成しない設定にする必要があります。

- Web コンテナが提供するログイン認証機能として FORM 認証を使用する場合，暗黙的に HTTP セッションが作成されます。セッションフェイルオーバの抑止機能によってセッションフェイルオーバ機能が無効となったリクエストで FORM 認証を使用すると，HTTP セッションを作成できないため，com.hitachi.software.web.dbsfo.SessionOperationException 例外が発生し，認証は行われません。ただし，すでにセッションが作成されている場合は，セッションフェイルオーバ抑止機能によってセッションフェイルオーバ機能が無効となったリクエストでも例外は発生しないで，認証は行われます。

## 5.6.2　HTTP セッションの参照専用リクエストの定義

HTTP セッションの参照専用リクエストの定義機能とは，HTTP セッションを更新しないで参照だけするリクエスト（**参照専用リクエスト**）の URL パターンを設定することで，その URL パターンのリクエストに対して HTTP セッションのシリアライズや，データベースへのアクセスの処理を抑止する機能です。

この機能は，データベースセッションフェイルオーバ機能の完全性保証モードが無効の場合に使用できます。

なお，HTTP セッションを更新だけでなく参照もしないリクエストの場合は，セッションフェイルオーバ抑止機能を使用できます。参照専用リクエストとセッションフェイルオーバ抑止機能の対象リクエストの両方に該当するリクエストの場合，セッションフェイルオーバ抑止機能の対象リクエストとして処理されます。セッションフェイルオーバ抑止機能については，「5.6.1　セッションフェイルオーバ機能の抑止」を参照してください。

HTTP セッションの参照専用リクエストの定義機能は，J2EE サーバ単位または Web アプリケーション単位に設定できます。

データベースセッションフェイルオーバ機能の J2EE サーバ単位の設定については，「6.6.1　J2EE サーバの設定」を参照してください。

- **注意事項**

  HTTP セッションの参照専用リクエストの定義について，注意事項を示します。

  - 参照専用リクエストでは，HTTP セッションを無効化できません。参照専用リクエストで，HTTP セッションを無効化する javax.servlet.http.HttpSession インタフェースの invalidate()メソッドを Web アプリケーション内で呼び出した場合，com.hitachi.software.web.dbsfo.SessionOperationException 例外がスローされます。
  - 参照専用リクエストの場合でも，HTTP セッションが存在しない初回のリクエストのときは，HTTP セッションが作成・更新・削除されます。この際，データベース上のグローバルセッション情報も更新されます。

    HTTP セッションが存在する 2 回目以降のリクエストでは，Web アプリケーションが HTTP セッションを更新しても，データベース上のグローバルセッション情報は更新されません。そのため，グローバルセッションの引き継ぎが発生すると，HTTP セッションの属性情報は更新前の状態に戻ります。
  - 参照専用リクエストの処理内で，HTTP セッションの有効期限の変更（javax.servlet.http.HttpSession インタフェースの setMaxInactiveInterval()メソッドの呼び出し）をした場合，グローバルセッションの有効期限は変更されません。そのため，グローバルセッションの引き継ぎが発生すると，有効期限は変更前の状態に戻ります。
  - 参照専用リクエストの処理内で，HTTP セッションの属性情報の変更をした場合，グローバルセッション情報は変更されません。そのため，グローバルセッションの引き継ぎが発生すると，HTTP セッションの属性情報は変更前の状態に戻ります。なお，属性情報の変更とは次のことを指します。

    ・javax.servlet.http.HttpSession インタフェースの setAttribute()メソッドまたは putValue()メソッドで，HTTP セッションに新しい属性情報を登録する，または登録済みのセッション属性を置き換える。

    ・javax.servlet.http.HttpSession インタフェースの removeAttribute()メソッドまたは removeValue()メソッドで，HTTP セッションに登録済みの属性情報を削除する。

    ・HTTP セッションに登録済みの属性情報の内容を変更する。

    セッションの属性情報の内容を変更する場合の例を次に示します。

    ```
    java.util.Hashtable table = (java.util.Hashtable)session.getAttribute(”attr1”);
    table.put(”key1”, ”value1”);
    ```

    この例では，session は HttpSession オブジェクトを格納した変数です。java.util.Hashtable オブジェクトは，別のリクエストで「attr1」という名前で HttpSession オブジェクトにセッションの属性情報として登録済みとします。

## 5.7　セッションフェイルオーバ機能使用時に実行される機能

ここでは，セッションフェイルオーバ機能を使用している場合に，自動で実行される機能について説明します。ここで説明する機能は，データベースセッションフェイルオーバ機能の完全性保障モードが無効の場合に適用される機能です。データベースセッションフェイルオーバ機能の完全性保障モードが有効の場合には適用されません。

### 5.7.1　同一セッション ID の同時実行

同一セッション ID の同時実行機能とは，同じセッション ID を持つ複数のリクエストが，冗長化された複数の J2EE サーバ，または一つの J2EE サーバに送信された場合に，複数のリクエストを同時に実行する機能です。複数のリクエスト処理を同時に実行するため，グローバルセッション情報のロック，およびロックの解除は行われません。

- **注意事項**

  同一セッション ID の同時実行で複数のリクエスト処理を同時に実行する場合，Web アプリケーションが発行する Servlet API の処理順序は不定となります。

  同じ HTTP セッションに対して，属性を登録するリクエスト（javax.servlet.http.HttpSession インタフェースの setAttribute()メソッド）とセッションを無効化するリクエスト（javax.servlet.http.HttpSession インタフェースの invalidate()メソッド）を同時に送信したり，セッションを無効化するリクエストを二重に送信したりすると，Servlet API の処理順序によっては，すでに無効化された HTTP セッションに対して属性の登録や無効化をしてしまう場合があります。この場合，Servlet API は java.lang.IllegalStateException 例外をスローします。このため，Servlet API で java.lang.IllegalStateException 例外がスローされることを考慮して Web アプリケーションを実装してください。

### 5.7.2　Web アプリケーション開始時のグローバルセッション情報の引き継ぎ

Web アプリケーションや J2EE サーバを停止した場合，または J2EE サーバが障害でプロセスダウンした場合，J2EE サーバ上の HTTP セッションは破棄されます。Web アプリケーションを再開始する際に，グローバルセッション情報を引き継ぐ機能を Web アプリケーション開始時のグローバルセッション情報の引き継ぎといいます。

J2EE サーバが障害でダウンしたあと，Web アプリケーションを再開始する際の，グローバルセッション情報の引き継ぎ処理の流れを次に示します。

1. **データベースから引き継ぐグローバルセッション情報のセッション ID のリストを取得する。**

   Web アプリケーションの開始時，データベースから引き継ぐグローバルセッション情報のセッション ID のリストを取得します。

2. **グローバルセッション情報の引き継ぎ処理を実行する。**

セッション ID のリストを取得できた場合，KDJE34344-I または KDJE34429-I のメッセージをメッセージログに出力し，グローバルセッション情報の引き継ぎ処理を開始します。

グローバルセッション情報の引き継ぎ処理では，リストに含まれているセッション ID のグローバルセッション情報を一つずつデータベースから J2EE サーバ上に引き継ぎます。

グローバルセッション情報の引き継ぎができなかった場合，障害の原因に対応したメッセージが出力されます。

3. 引き継ぎ処理を終了する。

セッション ID のリストにあるすべてのグローバルセッション情報の引き継ぎが完了した時点で，KDJE34349-I または KDJE34430-I のメッセージがメッセージログに出力されます。

なお，次の表に示す条件の場合，グローバルセッション情報は引き継がれません。

表 5-6　グローバルセッション情報が引き継がれない場合の条件と動作

| 項番 | 条件 | 動作 |
|---|---|---|
| 1 | ネットワーク障害などの理由でデータベースから引き継ぐグローバルセッション情報のセッション ID のリストを取得できなかった場合。 | KDJE34345-W または KDJE34431-W※のメッセージがメッセージログに出力され，グローバルセッション情報の引き継ぎ処理が終了します。 |
| 2 | 引き継ぐグローバルセッション情報が，すでに J2EE サーバ上に存在する場合（リクエストの受信で J2EE サーバ上にすでに引き継がれている）。 | KDJE34347-I または KDJE34432-I のメッセージがメッセージログに出力され，グローバルセッション情報の引き継ぎ処理がスキップされます。 |
| 3 | 引き継ぐグローバルセッション情報が，データベースセッションフェイルオーバ機能で，すでに冗長化された別の J2EE サーバに引き継がれている場合。 | KDJE34348-I のメッセージがメッセージログに出力され，グローバルセッション情報の引き継ぎ処理はスキップされます。 |
| 4 | ネットワーク障害などの理由でデータベースからグローバルセッション情報を取得できなかった場合。 | KDJE34346-W のメッセージがメッセージログに出力され，グローバルセッション情報の引き継ぎ処理がスキップされます。 |
| 5 | J2EE サーバ上の HTTP セッション数が，HttpSession オブジェクト数の上限値の指定機能で設定した上限値に達したために引き継げなかった場合。 | KDJE34370-W のメッセージがメッセージログに出力され，グローバルセッション情報の引き継ぎ処理がスキップされます。 |
| 6 | グローバルセッション情報のデシリアライズに失敗した場合。 | KDJE34328-E または KDJE34436-E のメッセージがメッセージログに出力され，グローバルセッション情報を引き継がないで，データベースから削除されます。 |
| 7 | データベースセッションフェイルオーバ機能を使用していて，HttpSession のサーバ ID 付加機能で設定したサーバ ID を変更した場合。 | KDJE34348-I のメッセージがメッセージログに出力され，グローバルセッション情報の引き継ぎ処理がスキップされます。 |

## 5.7.3　HTTP セッションの縮退

**HTTP セッションの縮退**とは，データベースで次の表に示す障害が発生した場合に，処理を中断しないで，J2EE サーバ上の HTTP セッションを使用してリクエスト処理を続行する機能です。

表 5-7　HTTP セッションの縮退が機能する障害の内容

| 障害の発生個所 | 使用する機能 | 障害の内容 |
|---|---|---|
| データベース | データベースセッションフェイルオーバ機能 | • グローバルセッション情報の作成時にデータベースに空きレコードが存在しない<br>• グローバルセッション情報の操作時にデータベース障害が発生 |

発生した障害によって HTTP セッションが縮退するときの動作を次の表に示します。

- データベースセッションフェイルオーバ機能

表 5-8　HTTP セッションの縮退時の動作（データベースセッションフェイルオーバ機能の場合）

| 発生した障害 | 縮退の動作 | 縮退時に出力されるメッセージ※ | 縮退が解除されるタイミング | 縮退した HTTP セッションの引き継ぎ |
|---|---|---|---|---|
| グローバルセッション情報の作成時にデータベースに空きレコードがない。 | J2EE サーバ上の HTTP セッションだけを作成する。 | KDJE34367-W | 以降の HTTP セッションの操作時にデータベースに空きレコードがあり，グローバルセッション情報を作成できた場合。 | データベースにグローバルセッション情報が存在しないため，引き継がれない。 |
| グローバルセッション情報の操作時にデータベース障害が発生した。 | J2EE サーバ上の HTTP セッションだけを操作する。 | KDJE34368-W | 以降の HTTP セッションの操作時にデータベースアクセスに成功した場合。 | • データベースにグローバルセッション情報が存在しない場合：引き継がれない。<br>• データベースにグローバルセッション情報が存在する場合：古いグローバルセッション情報が引き継がれるときがある。 |

注※　発生した障害ごとに，初めて縮退したときに出力されるメッセージです。それ以降は，縮退した HTTP セッションがすべてなくなってから，再び縮退するまで出力されません。なお，縮退した HTTP セッションがすべてなくなった場合は，メッセージ KDJE34369-I が出力されます。

## 5.8　メモリの見積もり

セッションフェイルオーバ機能を使用する場合，環境構築の準備として次のメモリサイズの見積もりをします。

- データベースセッションフェイルオーバ機能
  - シリアライズ処理で使用するメモリ
  - HTTP セッションの属性情報のサイズ
  - データベースのテーブル容量

ここでは，見積もり方法について説明します。

### 5.8.1　シリアライズ処理で使用するメモリの見積もり

セッションフェイルオーバ機能では，リクエスト処理完了時に，HTTP セッションの属性情報のシリアライズのために一時的にメモリが確保されます。このメモリ確保によって必要となるメモリ領域のサイズについて，JavaVM のチューニング時に考慮する必要があります。

チューニングでは，複数スレッドでメモリを確保する処理が重なった場合のメモリの増加量（**最大増加量**）を見積もります。リクエストの処理時に使用するメモリの最大増加量について，Web アプリケーション単位，および J2EE サーバ単位に求める式をそれぞれ次に示します。

```
Webアプリケーション単位の使用メモリ最大増加量（バイト）＝
最大同時実行スレッド数※1 × HTTPセッションの属性情報の最大サイズ※2

J2EEサーバ単位の使用メモリ最大増加量（バイト）＝
Webアプリケーション単位の使用メモリ最大増加量の合計 ＝
    Webアプリケーション1の使用メモリ最大増加量
  ＋Webアプリケーション2の使用メモリ最大増加量
      :
  ＋Webアプリケーション n の使用メモリ最大増加量
```

注※1

Web アプリケーション単位の同時実行スレッド数を設定している場合は Web アプリケーション単位の最大同時実行スレッド数の値を指します。Web アプリケーション単位の同時実行スレッド数を設定していない場合は Web コンテナ単位の最大同時実行スレッド数の値を指します。

注※2

HTTP セッションの属性情報のサイズ見積もり機能で見積もった値を指します。

上記の式で求めた値を基に，JavaVM のチューニングを実施してください。

## 5.8.2　HTTP セッションの属性情報のサイズの見積もり

セッションフェイルオーバ機能が使用するデータベースのディスク容量を確保する際，HTTP セッションの属性情報の最大サイズが必要になります。

HTTP セッションの属性情報のサイズを Web アプリケーションの内容から計算して求めることは困難です。そのため，アプリケーションサーバでは HTTP セッションの属性情報のサイズ見積もり機能を提供しています。HTTP セッションの属性情報のサイズ見積もり機能を使用すると，実際にアプリケーションを実行し，HTTP セッションに登録した属性のシリアライズ後のサイズ情報をメッセージとして出力できます。

ここでは，HTTP セッションの属性情報のサイズ見積もり機能および HTTP セッションの属性情報のサイズを求める計算式について説明します。

また，FullGC の抑止をする場合のメモリの確保についても説明します。

### (1)　HTTP セッションの属性情報のサイズの見積もり機能

HTTP セッションの属性情報のサイズ見積もり機能を使用すると，出力されたサイズ情報を参考にして，HTTP セッションの属性情報の最大サイズに適切な値を見積もることができます。

なお，この機能は見積もり時に使用する機能です。データベースへのグローバルセッションの格納は実施されないため，データベースへの接続は発生しません。

**注意事項**

HTTP セッションの属性情報のサイズの見積もり機能は，運用環境で使用しないでください。この機能を使用した場合，データベースセッションフェイルオーバ機能は無効となり，グローバルセッション情報がデータベースに冗長化されません。

#### (a)　HTTP セッションの属性情報のサイズの見積もり機能を有効にするための設定

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，次のパラメタに「on」を指定してください。

- webserver.dbsfo.check_size.mode パラメタ

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，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。

HTTP セッションの属性情報のサイズの見積もり機能を有効にすると，データベースセッションフェイルオーバ機能に関するほかの設定はすべて無効になります。

なお，HTTP セッションの属性情報のサイズの見積もり機能を使用する場合，「5.4.2　前提となる設定」で示している HttpSession オブジェクト数の上限値の指定機能および HTTP セッションのセッション ID を示す HTTP Cookie の削除機能を有効にしなくても動作します。

### (b) HTTP セッションの属性情報のサイズを通知するメッセージ

HTTP セッションの属性情報のサイズ見積もり機能が有効の場合，Web アプリケーションのリクエスト処理完了時に HTTP セッションの属性情報のサイズを通知する次のメッセージが Error レベルで出力されます。

表 5-9 HTTP セッションの属性情報のサイズを通知するメッセージ

| メッセージ ID | 内容 | サイズの情報に含まれる内容 |
|---|---|---|
| KDJE34330-I または KDJE34416-I | リクエストごとに作成した HTTP セッションの属性情報のサイズ | HTTP セッションに登録されている属性をシリアライズした結果の合計サイズ（KDJE34331-I または KDJE34417-I で出力するサイズの合計）※1 |
| KDJE34331-I または KDJE34417-I | シリアライズが完了した属性 1 個分のサイズ | • 属性名をシリアライズした結果（バイト配列）のサイズ<br>• 属性値をシリアライズした結果（バイト配列）のサイズ<br>• java.io.ObjectOutputStream クラスが書き込むマジックナンバー※2<br>• java.io.ObjectOutputStream クラスが書き込むバージョン情報のデータ分のサイズ※2 |

注※1 HTTP セッションに属性が登録されていない場合，java.io.ObjectOutputStream クラスが書き込むマジックナンバーおよび java.io.ObjectOutputStream クラスが書き込むバージョン情報のデータ分のサイズです。

注※2 最初にシリアライズされた属性のサイズにだけ含まれます。

登録されている属性が「Attribute1」，および「Attribute2」である HTTP セッションから，HTTP セッションの属性情報を作成した場合のメッセージの出力例を次に示します（データベースセッションフェイルオーバ機能の場合）。

```
KDJE34331-I  An attribute was serialized. (J2EE application = App01, context root = /test, r
equest URL = http://host01/test/TestServlet, attribute name = Attribute1, class name = app.M
yObject1, size(bytes) = 36, HTTP session ID = 01234567aaaabbbbccccddddeeeeffff)
KDJE34331-I  An attribute was serialized. (J2EE application = App01, context root = /test, r
equest URL = http://host01/test/TestServlet, attribute name = Attribute2, class name = app.M
yObject2, size(bytes) = 25, HTTP session ID = 01234567aaaabbbbccccddddeeeeffff)
KDJE34330-I  The attribute information was created. (J2EE application = App01, context root
= /test, request URL = http://host01/test/TestServlet, size(byte) = 61, HTTP session ID = 01
234567aaaabbbbccccddddeeeeffff)
```

## (2) HTTP セッションの属性情報のサイズを求める計算式

HTTP セッションの属性情報の最大サイズは次の式で求めることができます。

ここでは，一つの HTTP セッションに対し，一つの java.io.ObjectOutputStream のオブジェクトを使用してシリアライズしているものとします。

```
HTTPセッションの属性情報の最大サイズ（バイト）＝
        HTTPセッションに登録されたすべての属性の属性名をシリアライズしたバイト配列のバイト数
の合計
```

```
        ＋HTTPセッションに登録されたすべての属性の属性値をシリアライズしたバイト配列のバイト数
の合計
```

また，HTTP セッションに n 個のオブジェクトを属性として登録して，登録した属性をそれぞれ属性 1～属性 n とする場合，HTTP セッションの属性情報の最大サイズは次の式で求めることができます。

```
HTTPセッションの属性情報の最大サイズ（バイト）＝
          属性1の属性名をシリアライズしたバイト配列のバイト数
        ＋属性1の属性値をシリアライズしたバイト配列のバイト数
        ＋属性2の属性名をシリアライズしたバイト配列のバイト数
        ＋属性2の属性値をシリアライズしたバイト配列のバイト数
              ：（中略）
        ＋属性ｎの属性名をシリアライズしたバイト配列のバイト数
        ＋属性ｎの属性値をシリアライズしたバイト配列のバイト数
```

属性名をシリアライズしたバイト配列のバイト数，および属性値をシリアライズしたバイト配列のバイト数は次の式で求めることができます。

**属性名をシリアライズしたバイト配列のバイト数**

```
属性名をシリアライズしたバイト配列のバイト数＝
  属性名の文字数×3×1.2
```

**属性値をシリアライズしたバイト配列のバイト数**

```
属性値をシリアライズしたバイト配列のバイト数＝
  属性値のオブジェクトが持つすべてのフィールドの値のバイト数の合計×1.2
```

なお，フィールドの値のバイト数は次の式で求めることができます。

- String オブジェクトの場合：文字数×3
- その他のオブジェクトの場合：そのオブジェクトが持つすべてのフィールドの値のバイト数の合計
- プリミティブ型の場合：それぞれのプリミティブ型を格納するために必要なバイト数

**注意事項**

HTTP セッションの属性情報のサイズを求める計算式で算出できる値は概算です。最終的に HTTP セッションの属性情報の最大値を求める際には，HTTP セッションの属性情報のサイズの見積もり機能を使用してください。

## (3) FullGC の抑止をする場合のメモリの確保

HTTP セッションの属性情報のサイズはシリアライズ後のサイズであるため，HTTP セッションに登録した属性オブジェクトのメモリ上でのサイズとは異なります。そのため，FullGC の抑止で必要となる外部ヒープ領域のメモリサイズの見積もりは別途実施して，適切な値を設定する必要があります。

FullGC の抑止についての詳細は「7.　明示管理ヒープ機能を使用した FullGC の抑止」を参照してください。

## 5.8.3　データベースのディスク容量の見積もり

データベースセッションフェイルオーバ機能では，3 種類のテーブル（アプリケーション情報テーブル，セッション情報テーブル，空きレコード情報テーブル）を作成します。確保するディスク容量のサイズは，テーブルとインデクスの情報を基に各データベースのマニュアルを参照して見積もります。なお，これらの情報は Component Container のバージョンアップや修正パッチで変更になる場合があります。

### (1)　テーブルの情報

テーブルごとの列の要素，および行数について説明します。

- アプリケーション情報テーブル

列の要素を次の表に示します。

#### 表 5-10　アプリケーション情報テーブルの列の要素

| 項番 | 列名 | HiRDB 型 | ORACLE 型 | インデクスの有無 |
|---|---|---|---|---|
| 1 | APP_INFO_KEY | CHAR(128) PRIMARY KEY | VARCHAR2(128) PRIMARY KEY | なし |
| 2 | APP_INFO_VALUE | CHAR(512) | VARCHAR2(512) | なし |

行数は次のとおりです。

13 ＋参照専用リクエストの定義数

- セッション情報格納テーブル
  列の要素を次の表に示します。

#### 表 5-11　セッション情報格納テーブルの列の要素

| 項番 | 列名 | HiRDB 型 | ORACLE 型 | インデクスの有無 |
|---|---|---|---|---|
| 1 | RECORD_NO | INTEGER PRIMARY KEY | NUMBER(10,0) PRIMARY KEY | なし |
| 2 | SESSIONID | CHAR(112) | VARCHAR2(112) | あり |
| 3 | CREATION_TIME | DECIMAL(23,0) | NUMBER(23,0) | なし |
| 4 | MAX_INACTIVE_INTERVAL | INTEGER | NUMBER(10,0) | なし |
| 5 | THIS_ACCESSED_TIME | DECIMAL(23,0) | NUMBER(23,0) | なし |

| 項番 | 列名 | HiRDB 型 | ORACLE 型 | インデクスの有無 |
|---|---|---|---|---|
| 6 | ATTRIBUTES_DATA | BINARY(HTTP セッションの属性情報の最大サイズ)※1 | BLOB※2 | なし |
| 7 | STATUS | CHAR(16) | VARCHAR2(16) | なし |
| 8 | OWNER_SERVER | CHAR(512) | VARCHAR2(512) | なし |
| 9 | NEXT_FREE_RECORD_NO | INTEGER | NUMBER(10,0) | なし |

注※1　HTTP セッションの属性情報のサイズの見積もりについては，「5.8.2　HTTP セッションの属性情報のサイズの見積もり」を参照してください。

注※2　BLOB の列に格納する値の最大サイズは，HTTP セッションの属性情報の最大サイズです。HTTP セッションの属性情報のサイズの見積もりについては，「5.8.2　HTTP セッションの属性情報のサイズの見積もり」を参照してください。

行数は次のとおりです。

- ネゴシエーション処理に失敗した場合に Web アプリケーションの開始処理を続行する設定のとき
  **データベースに格納するグローバルセッション情報の数**
- ネゴシエーション処理に失敗した場合に Web アプリケーションの開始処理を中止する設定のとき
  **HttpSession オブジェクト数の最大値**
- 空きレコード情報テーブル

列の要素を次の表に示します。

### 表 5-12　空きレコード情報テーブルの列の要素

| 項番 | 列名 | HiRDB 型 | ORACLE 型 | インデクスの有無 |
|---|---|---|---|---|
| 1 | BLOCK_NO | INTEGER PRIMARY KEY | NUMBER(10,0) PRIMARY KEY | なし |
| 2 | FREE_RECORD_NO | INTEGER | NUMBER(10,0) | なし |

行数は 10 固定です。

## (2)　インデクスの情報

セッション情報格納テーブルのインデクスを次の表に示します。

| 項番 | インデクス名 | UNIQUE 属性 | カラム名 |
|---|---|---|---|
| 1 | <アプリケーション識別子>_SESSIONS_IDX | なし | SESSIONID |

**参考**

HiRDB を使用する場合，次の条件を満たすことで性能が向上することがあります。

- データベースセッションフェイルオーバ機能で使用するテーブルおよびインデクスを RD エリアに配置している※。
- RD エリアに配置したテーブルおよびインデクスにそれぞれグローバルバッファを設定している。

RD エリア，およびグローバルバッファの設計については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

注※

RD エリアにテーブルおよびインデクスを配置する場合，SQL ファイルの編集が必要です。

## 5.9 注意事項

ここでは，セッションフェイルオーバ機能使用時の注意事項，およびアプリケーション実装時の注意事項について説明します。

### 5.9.1 JSP で暗黙的に作成される HTTP セッション

セッションの引き継ぎを必要としない処理では明示的に HttpSession オブジェクトを作成しないように設定してください。

セッションフェイルオーバ機能を有効にしたアプリケーションでは，属性が登録されない HTTP セッションの作成時にも，グローバルセッション情報が作成され，さらにグローバルセッション情報の更新処理も発生します。

JSP 仕様では，デフォルトで HttpSession オブジェクトが作成されます。そのため，不要な処理によって，メモリの使用量が増加したり，データベースとの通信による負荷が発生したりするおそれがあります。

HttpSession オブジェクトの作成に関する設定には，page ディレクティブの session 属性を使用します。

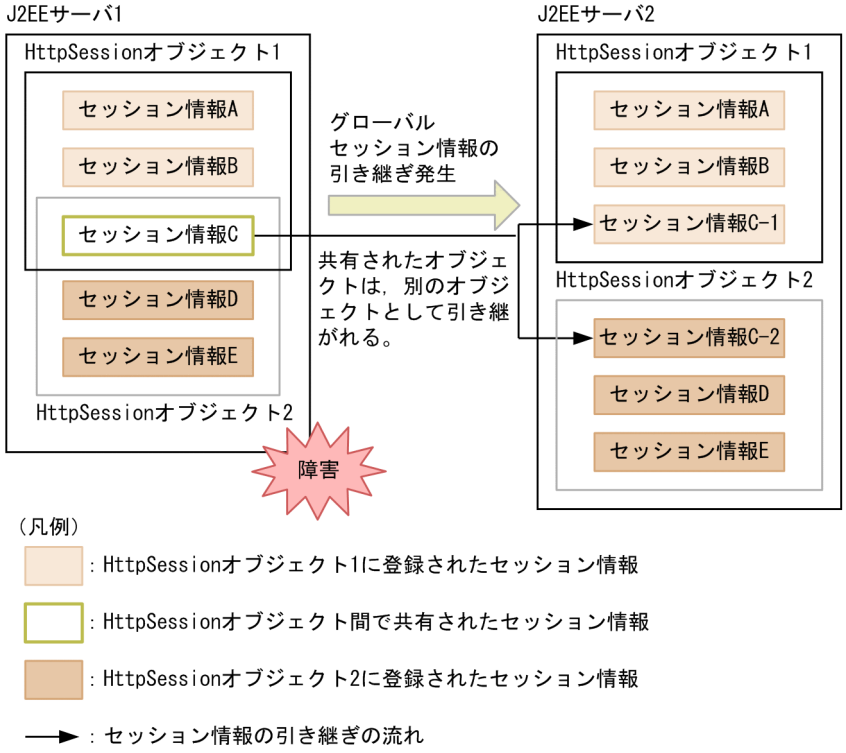
### 5.9.2 異なる HTTP セッションに同一のオブジェクトが登録されている場合を考慮した処理

グローバルセッション情報は HTTP セッション単位に作成されます。

異なる HttpSession オブジェクトで同一のオブジェクトをセッション情報として共有している場合に，グローバルセッション情報の引き継ぎが発生したときは，オブジェクトが共有されません。別々のオブジェクトとして作成されます。

異なる HttpSession オブジェクトに同一のオブジェクトが登録されている場合の引き継ぎの例を次の図に示します。

図 5-11　異なる HttpSession オブジェクトに同一のオブジェクトが登録されている場合の引き継ぎの例




この図では，J2EE サーバ 1 上の HttpSession オブジェクト 1 および HttpSession オブジェクト 2 で，同一のオブジェクトのセッション情報 C を共有しています。J2EE サーバ 1 で障害が発生して，J2EE サーバ 2 へセッション情報が引き継がれた場合，J2EE サーバ 2 上の HttpSession オブジェクト 1 および HttpSession オブジェクト 2 には，共有されていたセッション情報 C が，それぞれ別のセッション情報 C-1 とセッション情報 C-2 として作成されます。セッション情報 C-1 およびセッション情報 C-2 は，インスタンスは異なりますが，内容は同じです。

## 5.9.3　セッション情報の引き継ぎが発生した場合の認証情報の扱い

アプリケーションサーバではログイン認証機能として，Form 認証，Basic 認証，および HttpServletRequest のメソッド authenticate/login/logout があります。セッションフェイルオーバ機能を使用したアプリケーションでこれらのログイン認証機能を使用した場合，次の動作となります。

Form 認証を使用した場合

J2EE サーバの障害でセッションの引き継ぎが発生したとき，セッションの引き継ぎに成功した場合でも，再度 Form 認証による認証が必要です。

Basic 認証を使用した場合

J2EE サーバの障害でセッションの引き継ぎが発生したかどうかに関係なく，再度 Basic 認証を行う必要はなく，続けてアクセスできます。

HttpServletRequest のメソッド authenticate/login/logout を使用した場合

J2EE サーバの障害でセッションの引き継ぎが発生したとき，セッションの引き継ぎに成功した場合でも，再度メソッドによる認証が必要です。

Basic 認証および Form 認証については，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「6.2　DD の設定による Web コンテナのユーザ認証」を参照してください。

## 5.9.4　サーブレット API への影響

セッションフェイルオーバ機能を使用した場合のサーブレット API への影響として，次の項目について説明します。

- 引き継ぎ後の HttpSession オブジェクトに関連するサーブレット API の動作
- サーブレット API の呼び出しによるデータベースとの通信の発生

### (1)　引き継ぎ後の HttpSession オブジェクトに関連するサーブレット API の動作

引き継ぎ後の HttpSession オブジェクトに関連するサーブレット API の注意点について次の表に示します。

表 5-13　HttpSession オブジェクトに関連するサーブレット API の注意点

| 項番 | API 名 | 注意点 |
|---|---|---|
| 1 | getCreationTime() | 引き継ぎによって，HttpSession オブジェクトが作成された場合，引き継ぎ前の HttpSession オブジェクトの情報が引き継がれます。 |
| 2 | getLastAccessedTime() | |
| 3 | getId() | 引き継ぎによって，HttpSession オブジェクトが作成された場合，引き継ぎ前の HttpSession オブジェクトと同一の ID が取得できます。 |
| 4 | isNew() | 引き継ぎによって，HttpSession オブジェクトが作成されても，戻り値「true」は返されません。 |

この表に示していないサーブレット API については，セッションフェイルオーバ機能を使用した場合の影響はありません。

### (2)　サーブレット API の呼び出しによるデータベースとの通信の発生

次の表に示すサーブレット API を実装した場合，API 呼び出しの延長でデータベースとの通信が発生します。そのため，性能への影響があります。

表 5-14　データベースとの通信

| 項番 | クラス | メソッド |
|---|---|---|
| 1 | javax.servlet.http.HttpServletRequest | getSession()※1 |

| 項番 | クラス | メソッド |
|---|---|---|
| 2 | javax.servlet.http.HttpServletRequest | getSession(boolean create)※1 |
| 3 | javax.servlet.http.HttpSession | invalidate()※2 |

注※1

新規に HttpSession オブジェクトを作成した場合にだけ性能に影響があります。

注※2

有効な HttpSession オブジェクトの invalidate()メソッドを呼び出した場合にだけ性能に影響があります。

# 6 データベースセッションフェイルオーバ機能

この章では，データベースセッションフェイルオーバ機能について説明します。

## 6.1　この章の構成

ここでは，データベースセッションフェイルオーバ機能について説明します。

この機能を使用すると，アプリケーションで実行中のセッション情報がデータベースに格納されます。格納されたセッション情報は，Web サーバや J2EE サーバで障害が発生したときに，ほかの J2EE サーバに引き渡されます。これによって，障害発生時にリクエストがほかの J2EE サーバに転送された場合でも，障害発生前の状態で業務を続行できるようになります。

なお，セッションフェイルオーバ機能の種類や機能差異，前提条件，メモリの見積もり，注意事項については，「5.　J2EE サーバ間のセッション情報の引き継ぎ」を参照してください。

この章の構成を次の表に示します。

### 表 6-1　この章の構成（データベースセッションフェイルオーバ機能）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | 適用手順 | 6.2.1 |
| | 性能を重視したモードの選択（完全性保障モードの無効化） | 6.3 |
| | データベースセッションフェイルオーバ機能で実施される処理 | 6.4 |
| 実装 | cosminexus.xml での定義 | 6.5 |
| 設定 | J2EE サーバの設定 | 6.6.1 |
| | Web アプリケーションの設定 | 6.6.2 |
| | データベースの設定 | 6.6.3 |
| | DB Connector の設定 | 6.6.4 |
| | データベースセッションフェイルオーバ機能に関する設定の変更 | 6.7 |
| | データベースのテーブルの削除 | 6.8 |
| 注意事項 | データベースセッションフェイルオーバ機能使用時の注意事項 | 6.9 |

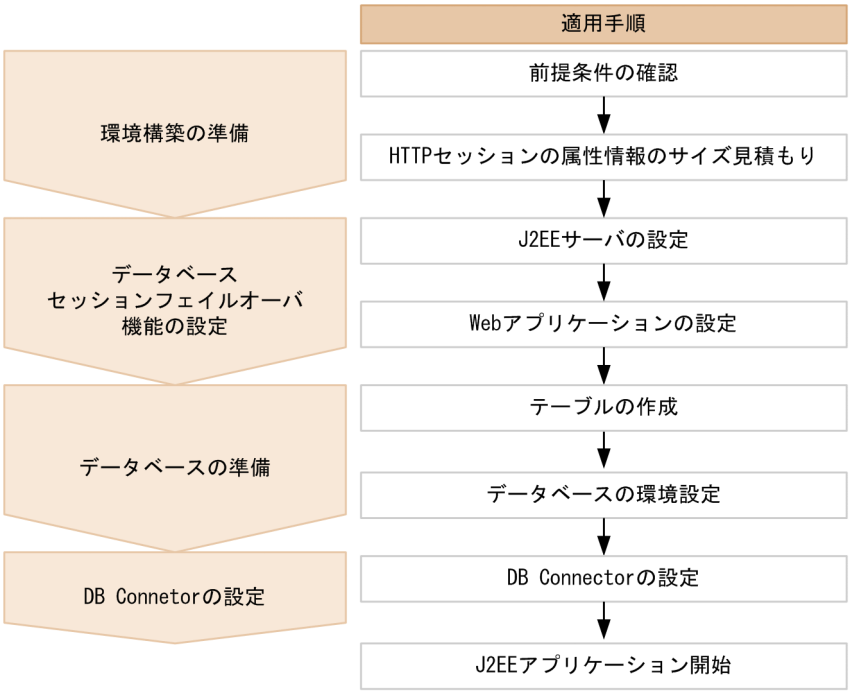
注　「運用」について，この機能固有の説明はありません。

# 6.2　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ための準備

## 6.2.1　適用手順

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必要な，環境構築の準備と各種設定について説明します。データベースセッションフェイルオーバ機能の適用手順を次の図に示します。

図 6-1　適用手順（データベースセッションフェイルオーバ機能）




ここで示す適用手順に従って環境構築の準備や各種設定を実施したあと，J2EE アプリケーションを開始します。

### (1)　環境構築の準備

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環境構築の準備として実施する項目について，実施内容および参照先を次の表に示します。

表 6-2　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環境構築の準備として実施する項目の実施内容および参照先

| 実施順序 | 実施項目 | 実施内容 | 参照先 |
|---|---|---|---|
| 1 | 前提条件の確認 | 前提となる構成および設定を確認します。 | 5.4 |
| 2 | HTTP セッションの属性情報のサイズ見積もり | HTTP セッションの属性情報のサイズを見積もります。見積もった値はデータベースの環境設定で必要になります。 | 5.8.2 |

## (2)　データベースセッションフェイルオーバ機能の設定

データベースセッションフェイルオーバ機能の設定について，設定内容および参照先を次の表に示します。

表 6-3　データベースセッションフェイルオーバ機能の設定内容および参照先

| 設定順序 | 設定項目 | 設定内容 | 参照先 |
|---|---|---|---|
| 1 | J2EE サーバの設定 | 次の設定をします。<br>• データベースセッションフェイルオーバ機能の設定（J2EE サーバ単位）<br>• DB Connector の別名の設定<br>• 完全性保障モードの設定<br>• シリアライズ処理で使用するメモリ量の設定<br>• セッションフェイルオーバ抑止機能の設定（拡張子，または URI 単位）<br>• 参照専用リクエストの設定<br>• HttpSession のサーバ ID 付加機能の設定<br>• 同時実行スレッド数制御機能を使用する場合の実行待ちキュー不足時の設定<br>• ネゴシエーション失敗時の Web アプリケーション開始処理の設定<br>• データベースセッションフェイルオーバ機能の抑止対象リクエスト内での，getSession メソッド実行時の例外の設定 | 6.6.1 |
| 2 | Web アプリケーションの設定※ | 次の設定をします。<br>• データベースセッションフェイルオーバ機能の設定（Web アプリケーション単位）<br>• HttpSession オブジェクト数の上限値設定<br>• アプリケーション識別子の設定<br>• HTTP セッションの属性情報の最大サイズ<br>• 拡張子によるセッションフェイルオーバ機能の抑止の設定 | 6.6.2 |

注※　Web アプリケーションの設定は開発環境で実施します。サーバ管理コマンドを使用して実行環境で Web アプリケーションの設定をしたい場合は「6.6.2　Web アプリケーションの設定」を参照してください。

## (3)　データベースの準備

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データベースの準備として実施する項目について，実施内容および参照先を次の表に示します。

表 6-4　データベースの準備として実施する項目の実施内容および参照先

| 実施順序 | 実施項目 | 実施内容 | 参照先 |
|---|---|---|---|
| 1 | テーブルの作成 | • データベースのディスク容量の確保<br>• アプリケーション情報テーブルの作成<br>• セッション情報格納テーブルの作成<br>• 空きレコード情報テーブルの作成 | 5.8.3,<br>6.6.3(2),<br>6.6.3(3),<br>6.6.3(4) |
| 2 | データベースの環境設定 | 次の設定をします。<br>• データベースのタイムアウトの設定 | 6.6.3(5) |

## (4)　DB Connector の設定

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必要な DB Connector の設定について，設定内容および参照先を次の表に示します。

表 6-5　DB Connector の設定内容および参照先

| 設定順序 | 設定項目 | 設定内容 | 参照先 |
|---|---|---|---|
| 1 | DB Connector の設定 | 次の設定をします。<br>• トランザクションサポートのレベルの設定<br>• DB Connector の環境設定<br>• DB Connector の別名の設定 | 6.6.4 |

## (5)　注意事項

データベースとして HiRDB を使用する場合，自動コミットを有効にしてください。自動コミットを無効にした場合，データベースセッションフェイルオーバ機能の SQL 実行のタイミングでエラーが発生します。

## 6.3　性能を重視したモードの選択（完全性保障モードの無効化）

ここでは，性能を重視したモードを選択した場合の動作，使用できる機能（グローバルセッション情報の削除），および注意事項について説明します。

性能を重視したモードを選択するには，完全性保障モードの設定を無効にします（デフォルト）。完全性保障モードを無効にすると，同一セッション ID を持つリクエスト処理を同時に実行できるようになります（**同一セッション ID の同時実行**）。

### 6.3.1　完全性保障モード無効時の動作

完全性保障モードが無効の場合の動作については，「5.7　セッションフェイルオーバ機能使用時に実行される機能」を参照してください。

### 6.3.2　グローバルセッション情報の削除

グローバルセッション情報の有効期限監視は，J2EE サーバ上の HTTP セッションを監視することで実施されます。有効期限の監視下では，有効期限が切れた HTTP セッションについて，データベース上のグローバルセッション情報が削除されます。しかし，J2EE サーバに障害が発生して停止した場合，その J2EE サーバで使用されていたグローバルセッション情報は，ほかの J2EE サーバに引き継がれるか，その J2EE サーバが再起動されるまで有効期限の監視が行われません。有効期限の監視が行われない状態が長く続くと，有効期限が過ぎても削除されないグローバルセッション情報が，セッション情報格納テーブルのレコードを使用し続けることになります。

このため，データベース上に残ったグローバルセッション情報を適宜削除する必要があります。

ここでは，グローバルセッション情報をコマンドによって削除する方法について説明します。

- **グローバルセッション情報の削除方法**

  グローバルセッション情報を削除するには，cjclearsession コマンドを使用します。J2EE サーバまたは Web アプリケーションが停止してから，HTTP セッションの有効期限以上の時間が経過したあとに，J2EE サーバまたは Web アプリケーションが再起動する前にコマンドを実行します。

  Web アプリケーション内で，Sevlet API を使用して HTTP セッションごとに有効期限を設定している場合は，最も長い有効期限に合わせてコマンドを実行してください。

  グローバルセッション情報を削除する手順を次に示します。

  1. 環境変数 CLASSPATH に，使用する JDBC ドライバのパスを設定する。

     cjclearsession コマンドを初めて使用する場合，環境変数 CLASSPATH に使用する JDBC ドライバのパスを指定します。

  2. cjclearsession コマンドを実行して，グローバルセッション情報を削除する。

コマンドにアプリケーション識別子，サーバ ID，使用する JDBC ドライバの情報，およびデータベースアクセスに必要な情報を指定して実行します。アプリケーション識別子で指定した Web アプリケーションの，サーバ ID で指定した J2EE サーバが所有するグローバルセッション情報がすべて削除されます。

3. 必要に応じて，J2EE サーバまたは Web アプリケーションを再起動する。

なお，cjclearsession コマンドに-count オプションを指定して実行すると，J2EE サーバが所有するグローバルセッション情報数を表示できます。

データベースへの接続試行のタイムアウト，データベースのグローバルセッション情報の取得または削除する SQL の実行タイムアウトは 8 秒です。

コマンド実行中にデータベースアクセスでエラーが発生した場合，エラーが発生した時点でコマンドの実行を中止します。

cjclearsession 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「cjclearsession（グローバルセッション情報の削除（データベースセッションフェイルオーバ機能））」を参照してください。

- **注意事項**

  グローバルセッション情報の削除についての注意事項を示します。

  - 削除する HTTP セッションを所有する J2EE サーバが稼働中の場合の削除

    J2EE サーバが稼働中の場合，リクエスト処理が行われてグローバルセッション情報が新たに作成されることがあります。このため，削除対象とする HTTP セッションを所有する J2EE サーバが稼働中の場合は，グローバルセッションの有効期限が切れる前に削除される可能性があります。グローバルセッション情報を削除する場合は，削除対象とする HTTP セッションを所有する J2EE サーバを停止してからコマンドを実行してください。

  - 有効期限前の削除

    グローバルセッションの有効期限が切れる前に cjclearsession コマンドを実行してグローバルセッション情報の削除をした場合，次の動作になります。

| 項番 | 完全性保障モード | J2EE サーバ上の HTTP セッションの有無 | 動作 |
|---|---|---|---|
| 1 | 無効 | なし | グローバルセッションの引き継ぎができません。 |
| 2 | | あり | 削除されたグローバルセッション情報は，以降，データベース上に保存されない状態となり，J2EE サーバ上の HTTP セッションだけで Web アプリケーションが動作します。 |

  - 完全性保障モードが有効の場合の削除

    完全性保障モードが有効の場合，動作は保障されません。

  - Oracle JDBC Thin Driver を使用する場合

    cjclearsession コマンドは，JDBC ドライバの setQueryTimeout メソッドを使用して SQL 実行時のタイムアウトを実現しています。Oracle JDBC Thin Driver を使用して Oracle に接続する場合の注意事項は，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「3.6.6　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。

### 6.3.3　注意事項

ここでは，完全性保障モード無効時の注意事項について説明します。

- **完全性保障モードの切り替え**

  完全性保障モードを無効から有効に切り替える場合は，次の手順に従って，データベースのセッション情報格納テーブルとアプリケーション情報テーブルを初期化してください。

  1. 冗長化した J2EE サーバをすべて停止する。
  2. HTTP セッションを破棄する。

     HTTP セッションの破棄の手順については，「6.7.3　グローバルセッション情報の削除（HTTP セッションの破棄）」を参照してください。
  3. データベースに保存した設定情報を初期化する。

     データベースに保存した設定情報を初期化する手順については，「6.7.2　データベーステーブルの初期化」を参照してください。

- **J2EE サーバ停止時のグローバルセッション情報の有効期限監視**

  Web アプリケーションまたは J2EE サーバを停止したときや，J2EE サーバに障害が発生してプロセスダウンしたとき，グローバルセッション情報の有効期限は監視されません。Web アプリケーションの開始，またはリクエストの受信でグローバルセッション情報が J2EE サーバに引き継がれた時点から，有効期限の監視が開始されます。

  なお，完全性保障モードが有効の場合，J2EE サーバが停止したときは別の J2EE サーバによって有効期限が監視されます。有効期限監視の処理の詳細については，「6.4.3　グローバルセッション情報の有効期限が切れた場合の処理」を参照してください。

- **グローバルセッション情報の数が上限に達した場合の動作**

  グローバルセッション情報作成時に，データベース上のグローバルセッション情報の数が上限に達していた場合，HTTP セッションを縮退します。HTTP セッションの縮退については，「5.7.3　HTTP セッションの縮退」を参照してください。

## 6.4　データベースセッションフェイルオーバ機能で実施される処理

データベースセッションフェイルオーバ機能では，次に示す時点でそれぞれ処理が実施されます。

- アプリケーション開始時
  アプリケーションのネゴシエーション処理が実施されます。
- リクエスト実行時
  グローバルセッション情報の格納，更新，削除が実施されます。

この節では，データベースセッションフェイルオーバ機能で実施される処理について説明します。

また，グローバルセッション情報の有効期限が切れた場合の処理，データベースセッションフェイルオーバ機能で発生するイベントに関連して動作するリスナ，完全性保障モードの場合にだけ実施されるグローバルセッション情報のロック処理，およびグローバルセッション情報操作中の障害発生時の動作についても説明します。

### 6.4.1　アプリケーション開始時の処理

ここでは，アプリケーション開始時に実施されるアプリケーションのネゴシエーション処理，およびアプリケーションのネゴシエーション処理で使用するアプリケーション識別子について説明します。

#### (1)　アプリケーションのネゴシエーション処理

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 Web アプリケーションでは，アプリケーション開始時にネゴシエーション処理が実行されます。

ネゴシエーション処理では，次の内容が確認されます。

- Web アプリケーションが一致していること
- 各 Web アプリケーションの設定が一致していること
- J2EE サーバの設定が一致していること
- データベースの設定が正しいこと

ネゴシエーション処理の結果によって Web アプリケーションが開始されるかどうかが決まります。

ネゴシエーション処理の結果と Web アプリケーションの状態の関係を次の表に示します。

表 6-6　ネゴシエーション処理の結果と Web アプリケーションの状態の関係

| ネゴシエーション処理の結果 | Web アプリケーションの状態 | ネゴシエーションの失敗の要因 | 出力されるメッセージ |
|---|---|---|---|
| 成功（確認内容に問題なし） | 開始される | － | KDJE34306-I |

| ネゴシエーション処理の結果 | Web アプリケーションの状態 | ネゴシエーションの失敗の要因 | 出力されるメッセージ |
|---|---|---|---|
| 失敗（確認内容に問題あり） | 開始されない | Web アプリケーションが一致していない。 | KDJE34340-E |
| | 開始されない※ | Web アプリケーションの設定が一致していない。 | KDJE34307-E |
| | 開始される※ | | KDJE34358-I |
| | 開始されない | J2EE サーバの設定が一致していない。 | KDJE34307-E |
| | 開始されない | 必要なテーブルがデータベースに存在していない。 | KDJE34308-W |
| | 開始されない | 存在するテーブルの内容がデータベースセッションフェイルオーバ機能用のテーブルの内容ではない。 | KDJE34309-E |
| | 開始されない | 存在するテーブルがほかのアプリケーションで使用されている。 | KDJE34340-E |

（凡例）－：該当なし

注※　次の確認項目について，開始する Web アプリケーションと，そのほかの J2EE サーバ上の同一 Web アプリケーションとで異なる値が設定されていた場合，Web アプリケーションの開始処理を続行するか中止するかを，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグに webserver.dbsfo.negotiation.high_level パラメタを指定することで選択できます。

- HttpSession オブジェクト数の上限値
- DD（web.xml）に定義された HTTP セッションの有効期間

これ以外の確認項目について一致していない場合は，Web アプリケーションは開始されません。

なお，Web アプリケーションの開始処理を中止する設定にした場合，HttpSession オブジェクト数の上限値に 1 以上の有効な値を必ず設定してください。

ネゴシエーションの処理中にデータベースアクセスでエラーが発生した場合，KDJE34312-W のメッセージがメッセージログに出力されます。

## (a)　ネゴシエーションで確認される内容（データベースセッションフェイルオーバ機能）

ネゴシエーションで確認される内容の詳細について説明します。

- Web アプリケーションが一致していること

  確認項目がすべて一致することで，Web アプリケーションが一致していると判断されます。確認項目を次の表に示します。

### 表 6–7　Web アプリケーションの一致の確認のために使用される項目

| 項番 | 確認項目 |
|---|---|
| 1 | アプリケーション識別子※ |
| 2 | J2EE アプリケーション名 |

| 項番 | 確認項目 |
|---|---|
| 3 | Web アプリケーション名（コンテキストルート名） |

注※　アプリケーション識別子については，「6.4.1(2)　アプリケーション識別子」を参照してください。

- 各 Web アプリケーションの設定が一致していること

  次の表に示す確認項目について，冗長化した各 Web アプリケーションの設定が一致しているか確認されます。

表 6-8　各 Web アプリケーションの設定の一致を確認するための項目

| 項番 | 確認項目 |
|---|---|
| 1 | HttpSession オブジェクト数の上限値 |
| 2 | グローバルセッション情報に含めることができる HTTP セッションの属性情報の最大サイズ |
| 3 | DD（web.xml）に定義された HTTP セッションの有効期間 |
| 4 | データベースセッションフェイルオーバ機能を抑止する拡張子 |

- J2EE サーバの設定が一致していること

  次の表に示す確認項目について，冗長化した各 J2EE サーバの設定が一致しているか確認されます。

表 6-9　各 J2EE サーバの設定の一致を確認するための項目

| 項番 | 確認項目 |
|---|---|
| 1 | 完全性保障モードの設定 |
| 2 | 参照専用リクエストの設定 |
| 3 | 同時実行スレッド数制御機能を使用する場合の実行待ちキュー不足時の設定 |
| 4 | データベースセッションフェイルオーバ機能の抑止対象リクエスト内での，getSession メソッド実行時の例外の設定 |

- データベースの設定が正しいこと

  次の表に示す条件を満たしているかが確認されます。

表 6-10　データベースの設定が正しいことを確認するための条件

| 項番 | 条件 |
|---|---|
| 1 | 必要なテーブルがデータベースに存在すること。 |
| 2 | 存在するテーブルの内容がデータベースセッションフェイルオーバ機能用のテーブルの内容であること。 |
| 3 | 存在するテーブルがほかのアプリケーションで使用中でないこと。 |

### (b)　ネゴシエーションで確認される Web アプリケーションの設定内容

最初にネゴシエーション処理に成功した Web アプリケーションの設定内容は，データベースのアプリケーション情報テーブルに保存されます。保存された設定内容はネゴシエーションの確認で使用される，正しい設定情報として扱われます。

このため，Web アプリケーションの設定内容を変更する場合は，すでにデータベースに保存されている，変更対象の Web アプリケーションに関連する設定情報を削除する必要があります。設定の変更手順については，「6.7　データベースセッションフェイルオーバ機能に関する設定の変更」を参照してください。

## (2)　アプリケーション識別子

**アプリケーション識別子**とは，データベースセッションフェイルオーバ機能使用時にクラスタリングされた Web アプリケーションを認識するための名称です。デフォルトの設定では，システムによって自動的に生成されます。

アプリケーション識別子は，ネゴシエーションで Web アプリケーションが一致しているかどうかの確認に使用されます。そのため，次の条件を満たしている必要があります。

- 冗長化した J2EE サーバで動作する同一の Web アプリケーションで一致している。
- システム内で一意の値である。

システムによって自動的に生成されるアプリケーション識別子が条件を満たさない場合，条件を満たす値を定義する必要があります。定義方法については「6.5　cosminexus.xml での定義」を参照してください。

アプリケーション識別子の自動生成規則，および自動生成されるアプリケーション識別子の例について説明します。

**注意事項**

異なる Web アプリケーションに同じアプリケーション識別子が設定されている場合，二つ目の Web アプリケーション開始時にネゴシエーションに失敗して，Web アプリケーションが開始されません。

### (a)　アプリケーション識別子の自動生成規則

デフォルトの設定では，アプリケーション識別子にはコンテキストルート名を基にした文字列が自動的に設定されます。アプリケーション識別子が自動的に生成された場合，Web アプリケーション開始時に，適用した値が KDJE34302-I のメッセージでメッセージログに出力されます。

コンテキストルート名を基にしたアプリケーション識別子の自動生成には，次に示す規則が適用されます。

- 先頭のスラッシュ（/）は削除する。
- 先頭のスラッシュ（/）を除き，文字列長が 16 文字を超える場合，16 文字までの文字列を使用する。
- アプリケーション識別子に使用できない文字をコンテキストルート名で使用している場合，アンダースコア（_）に置換する。
  アプリケーション識別子に使用できる文字は英数字（A～Z，a～z，0～9），およびアンダースコア（_）だけです。また，設定した値は，大文字小文字が区別されます。
- ルートコンテキストの場合，空文字列ではなく，「ROOT」とする。

自動生成規則を適用した結果，アプリケーション識別子がシステム内で一意でなくなる場合があります。この場合，同じアプリケーション識別子が設定されている二つ目の Web アプリケーション開始時にネゴシエーションに失敗して，Web アプリケーションが開始されません。そのため，Web アプリケーションに対してシステム内で一意になるアプリケーション識別子を設定する必要があります。

### (b) 自動生成されるアプリケーション識別子の例

コンテキストルート名から自動生成されるデフォルトのアプリケーション識別子の例を次の表に示します。

表 6-11 自動生成されるデフォルトのアプリケーション識別子の例

| 項番 | コンテキストルート名 | アプリケーション識別子 | 作成時に適用されるルール |
|---|---|---|---|
| 1 | /examples | examples | 先頭の"/"を削除する |
| 2 | /App01/test1 | App01_test1 | • 先頭の"/"を削除する<br>• 途中の"/"を"_"に置換する |
| 3 | /WebApplication_001 | WebApplication_0 | • 先頭の"/"を削除する<br>• 17 文字以降を削除する |
| 4 | /examples/WebApplication | examples_WebAppl | • 先頭の"/"を削除する<br>• 途中の"/"を"_"に置換する<br>• 17 文字以降を削除する |
| 5 | / | ROOT | ルートコンテキストのため"ROOT"とする |

## 6.4.2 リクエスト実行時の処理

ここでは，リクエスト実行時のグローバルセッションの作成，更新，削除およびグローバルセッションの引き継ぎについて説明します。

Web アプリケーション内で処理が実行されると，処理の延長でグローバルセッション情報に対しての処理が発生します。Web アプリケーション内で実行される処理の例と，例に対応したリクエスト実行時のグローバルセッション情報に対して実行される処理，および参照先を次の表に示します。

表 6-12 Web アプリケーション内での処理の例とグローバルセッション情報に対して実行される処理の対応

| 項番 | Web アプリケーション内で実行される処理の例 | グローバルセッション情報に対して実行される処理 | 参照先 |
|---|---|---|---|
| 1 | ログイン | グローバルセッション情報の作成 | (1) |
| 2 | 業務の実行（ページ遷移/更新） | グローバルセッション情報の更新 | (2) |
| 3 | ログアウト | グローバルセッション情報の削除 | (3) |

| 項番 | Web アプリケーション内で実行される処理の例 | グローバルセッション情報に対して実行される処理 | 参照先 |
|---|---|---|---|
| 4 | タイムアウトによるログアウト | 有効期限切れによるグローバルセッション情報の削除 | 6.4.3 |
| 5 | 別の J2EE サーバにグローバルセッションを引き継いでの業務の実行<br>(J2EE サーバでの障害発生時) | グローバルセッション情報を使用したセッション情報の引き継ぎ | (4) |

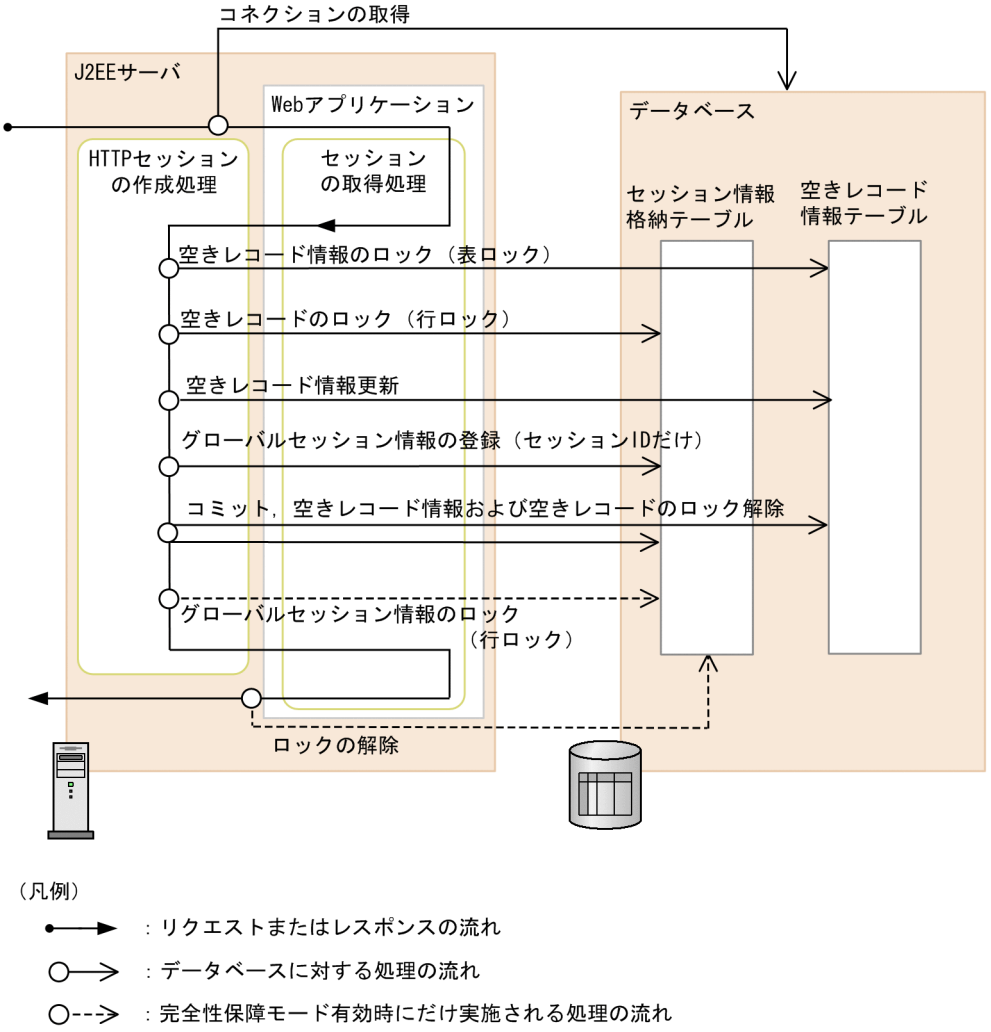
グローバルセッション情報操作中に障害が発生した場合の処理結果については，「6.4.6　グローバルセッション情報操作中の障害発生時の動作」を参照してください。

## (1)　グローバルセッション情報の作成

J2EE サーバ上で新規に HTTP セッションが作成されると，データベース上にグローバルセッション情報が作成されます。

グローバルセッション情報の作成で実行される処理の流れを次の図に示します。

図 6-2　グローバルセッション情報の作成（データベースセッションフェイルオーバ機能）



1. HTTP セッションが必要なリクエストを受け付けると，新規に HTTP セッションが作成されます。HTTP セッションの作成のタイミングは，Web アプリケーション内で，javax.servlet.http.HttpServletRequest インタフェースの getSession()メソッド，または getSession(true)メソッドを使用して HttpSession オブジェクトを新規に取得したときです。

   次のような場合も HttpSession オブジェクトが作成されるため，新規に HTTP セッションが作成されます。

   - Form 認証を使用した場合
   - JSP で page ディレクティブの session 属性に true を指定した場合
   - JSP で page ディレクティブの session 属性の指定を省略した場合

2. HTTP セッション作成処理の延長でデータベース上にグローバルセッション情報が作成されます。作成されたグローバルセッション情報は，セッション情報格納テーブルに格納されます。

   グローバルセッション情報は，作成と同時にロックされます。

3. グローバルセッション情報の作成に伴って，空きレコード情報が更新されます。

4. 作成されたグローバルセッション情報は一度コミットされます。

   完全性保障モードが有効の場合，改めてロックが取得されます。これは，HTTP セッション作成後に，Web アプリケーション実行中の J2EE サーバ，またはデータベースの障害発生によって，セッション情報格納テーブルと空きレコード情報テーブルの間で不整合を発生させないためです。

5. Web アプリケーションでの処理が終了すると，HTTP セッションが更新されます。

6. HTTP セッションの更新処理の延長で，グローバルセッション情報が更新されます。完全性保障モードが有効の場合，更新が完了すると，ロックが解除されます。

### 注意事項

グローバルセッション情報の数が上限に達していた場合の動作

グローバルセッション情報作成時に，データベース上のグローバルセッション情報の数が上限に達していた場合，HTTP セッションを縮退します。HTTP セッションの縮退については，「5.7.3　HTTP セッションの縮退」を参照してください。

完全性保障モードが有効の場合にデータベース上のグローバルセッション情報の数が上限に達していたときは，java.lang.IllegalStateException がスローされ，HTTP セッションの取得に失敗します。

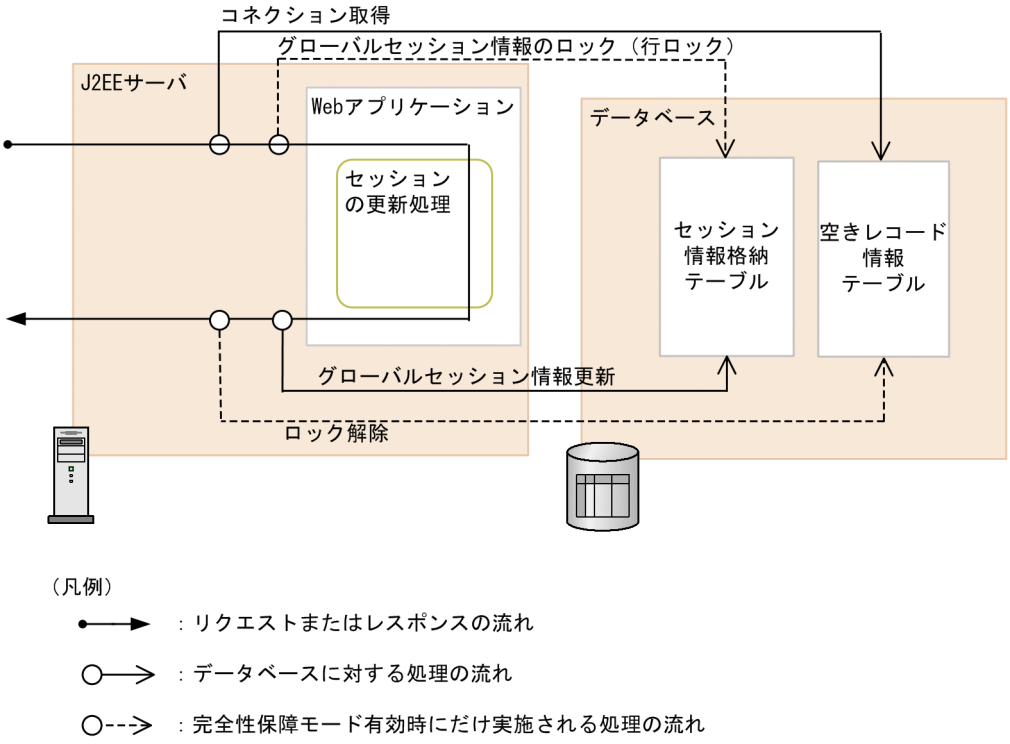
また，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグで webserver.session.max.throwHttpSessionLimitExceededException パラメタに true が設定された場合，java.lang.IllegalStateException の代わりに com.hitachi.software.web.session.HttpSessionLimitExceededException がスローされます。HttpSessionLimitExceededException については，マニュアル「アプリケーションサーバ リファレンス API 編」の「3.1　例外クラス」を参照してください。

## (2)　グローバルセッション情報の更新

Web アプリケーション実行中にセッションが更新されると，J2EE サーバで HTTP セッションが更新されます。それに伴って，データベース上のグローバルセッション情報も更新されます。

グローバルセッション情報の更新で実行される処理の流れを次の図に示します。

図 6-3　グローバルセッション情報の更新（データベースセッションフェイルオーバ機能）




1. HTTP セッションが存在するリクエストを受信します。
   完全性保障モードが有効の場合，Web アプリケーション実行前にデータベース上のグローバルセッション情報がロックされます。
2. Web アプリケーションでのセッションの更新に伴って，HTTP セッションが更新されます。
3. HTTP セッションの更新によって，データベース上のグローバルセッション情報が最新の情報に更新されます。
   完全性保障モードが有効の場合，ロックが解除されます。

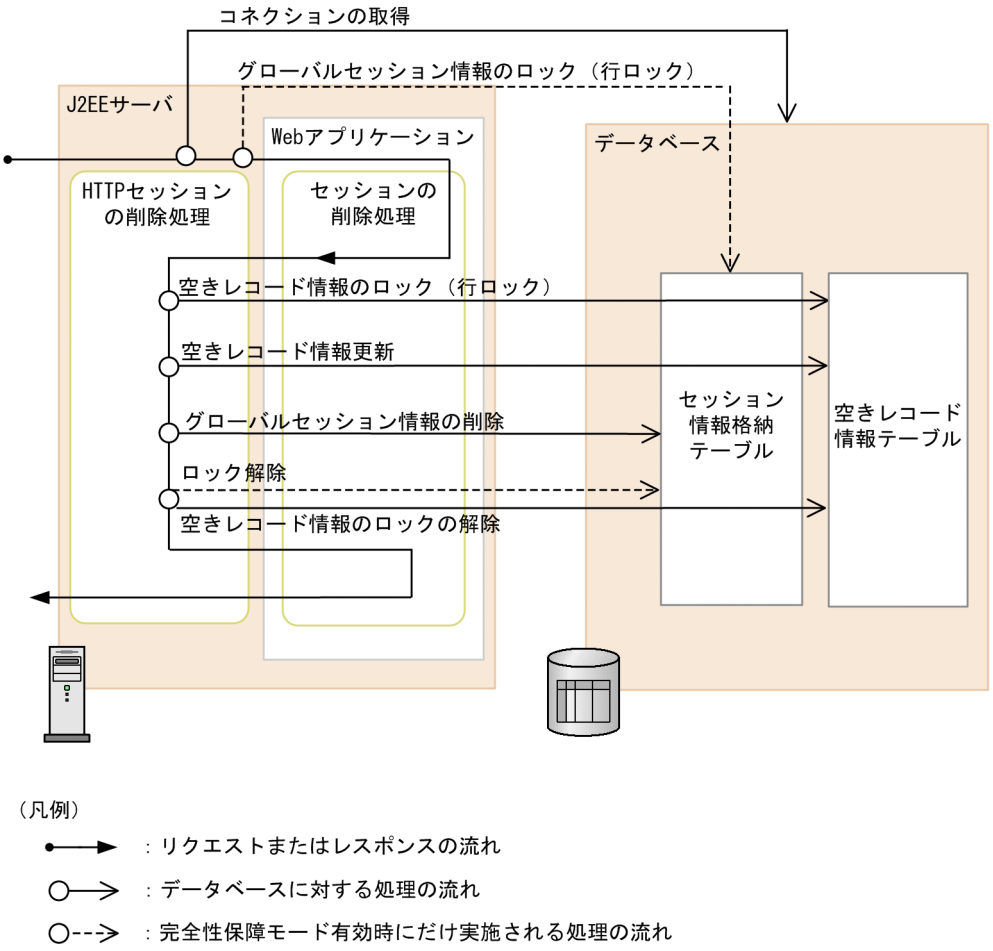
グローバルセッション情報のロック時の動作については，「6.4.5(1)　ロック取得時のロック取得処理の呼び出し結果」を参照してください。

## (3)　グローバルセッション情報の削除

Web アプリケーション内でのセッションの削除処理で，javax.servlet.http.HttpSession インタフェースの invalidate()メソッドを実装して明示的に HTTP セッションを削除すると，その処理の延長でデータベース上のグローバルセッション情報が削除されます。

グローバルセッション情報の削除で実行される処理の流れを次の図に示します。

図 6-4　グローバルセッション情報の削除（データベースセッションフェイルオーバ機能）



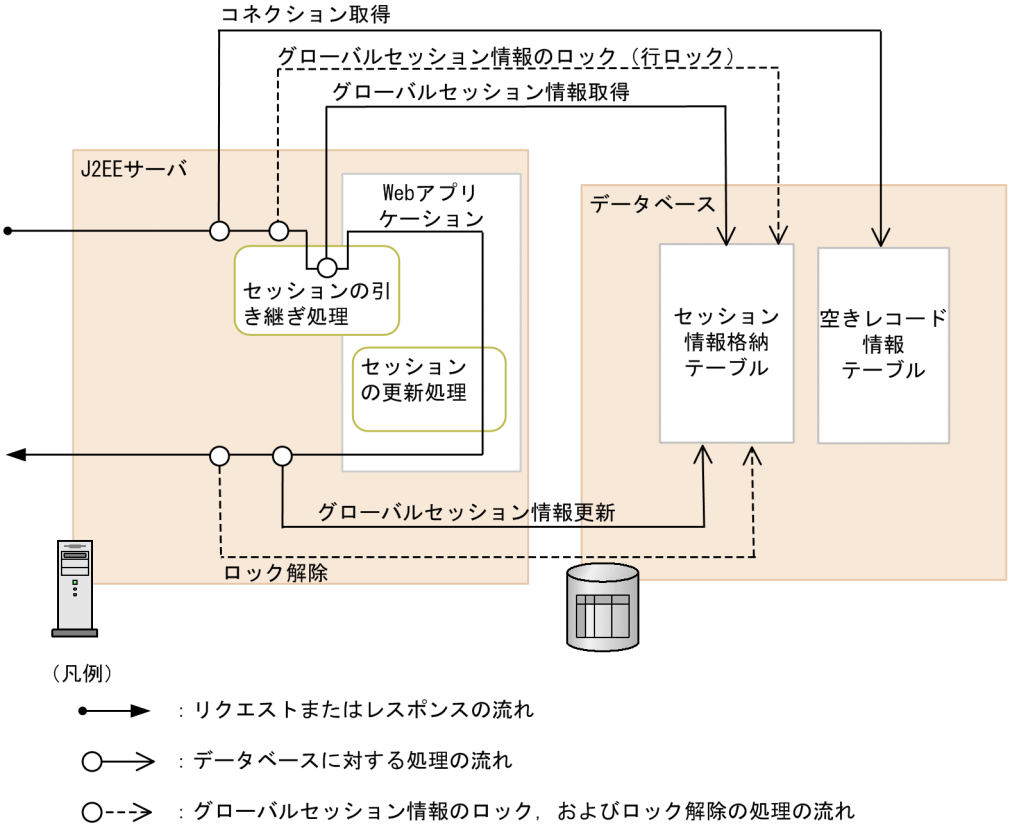

1. HTTP セッションの削除が必要なリクエストを受信します。
   完全性保障モードが有効の場合，Web アプリケーション実行前にデータベース上のグローバルセッション情報がロックされます。
2. Web アプリケーションでのセッションの削除に伴って，HTTP セッションが削除されます。
3. HTTP セッションの削除によって，データベース上のグローバルセッション情報および空きレコード情報が削除されます。
   完全性保障モードが有効の場合，ロックが解除されます。

## (4)　グローバルセッション情報を使用したセッション情報の引き継ぎ

受信したリクエストに関連づけられた HTTP セッションが J2EE サーバ上に存在しない場合，データベース上のグローバルセッション情報を使用して HTTP セッションが再作成されます。これによってセッションが引き継げます。

グローバルセッション情報を使用したセッション情報の引き継ぎで実行される処理の流れを次の図に示します。

図 6-5　グローバルセッション情報を使用したセッション情報の引き継ぎ（データベースセッションフェイルオーバ機能）




1. 受信したリクエストに関連づけられた HTTP セッションが J2EE サーバ上に存在しない場合，データベース上のグローバルセッション情報を呼び出して，J2EE サーバ上に HTTP セッションが再作成されます。

   再作成された HTTP セッションによってセッションが引き継がれ，Web アプリケーションでセッションの更新処理が実行されます。セッションの更新処理の延長上で，HTTP セッションが更新されます。

2. HTTP セッションの更新に伴って，グローバルセッション情報が更新されます。

なお，グローバルセッション情報の引き継ぎができた場合，KDJE34321-I のメッセージがメッセージログに出力されます。クライアントから受信したセッション ID に対応するグローバルセッション情報がデータベース上に存在しないためにグローバルセッション情報の引き継ぎができなかった場合，KDJE34325-W のメッセージがメッセージログに出力されます。

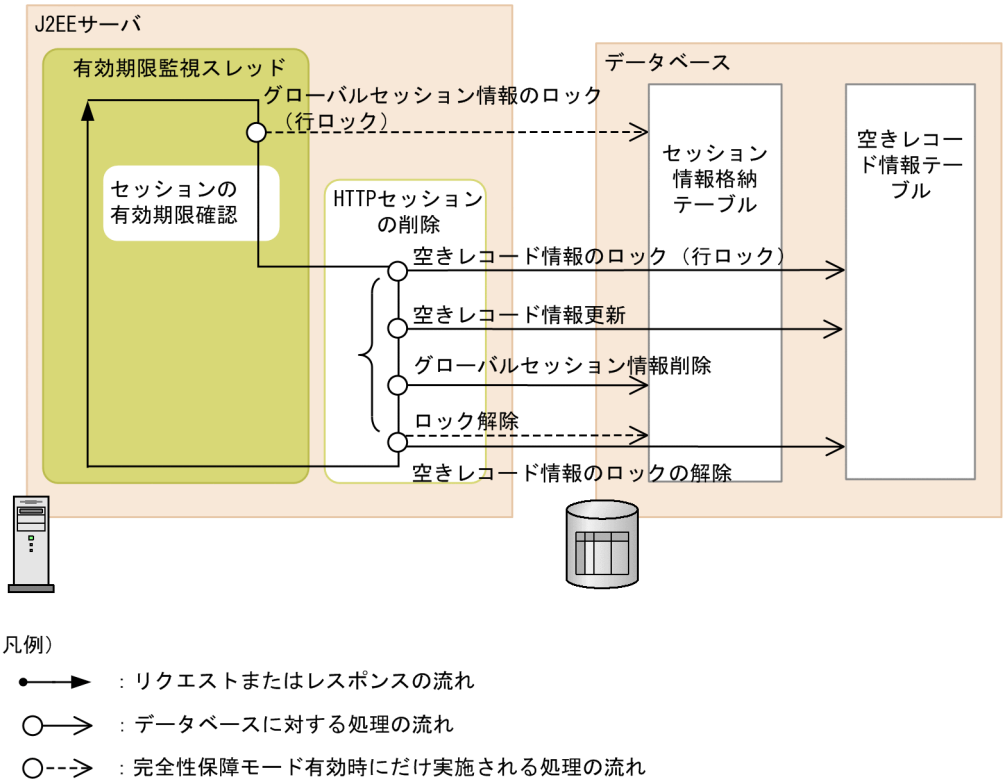
## 6.4.3　グローバルセッション情報の有効期限が切れた場合の処理

HTTP セッションにはそれぞれ有効期限が設定されています。最終アクセス時刻の情報を基にした有効期限確認の結果，有効期限を超過している HTTP セッションは削除されます。HTTP セッションが有効期限の超過によって削除されると，その処理の延長で，対応するグローバルセッション情報も削除されます。

HTTP セッションの有効期限は，Web コンテナに存在する有効期限監視スレッドによって定期的に監視されています。有効期限監視スレッドは Web アプリケーションごとに存在しています。

有効期限切れによるグローバルセッション情報の削除で実行される処理の流れを次の図に示します。

図 6-6　グローバルセッション情報の有効期限が切れた場合の処理（データベースセッションフェイルオーバ機能）




1. 完全性保障モードが有効の場合，有効期限監視スレッドによって有効期限切れであると判断されたセッションについて，対応するグローバルセッション情報がロックされます。
2. セッションの削除処理の延長で，HTTP セッションが削除されます。また，HTTP セッションの削除によって，データベース上のグローバルセッション情報および空きレコード情報が削除されます。

   完全性保障モードが有効の場合，ロックが解除されます。

## 6.4.4　データベースセッションフェイルオーバ機能で発生するイベントに関連して動作するリスナ

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，グローバルセッションの引き継ぎが発生したタイミングで javax.servlet.http.HttpSessionActivationListener インタフェースの sessionDidActivate()メソッドが呼び出されます。また，このとき javax.servlet.http.HttpSessionListener インタフェースの sessionCreated()メソッドは呼び出されません。

HTTP セッションを使用する処理では，Java EE で規定されたイベントに対応して HTTP セッションに関連するリスナが動作します。HTTP セッションに関連するリスナとは，次のインタフェースを実装したクラスです。

- javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSessionActivationListener
- javax.servlet.http.HttpSessionAttributeListener
- javax.servlet.http.HttpSessionBindingListener

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，HTTP セッションに関連するリスナはデータベースセッションフェイルオーバ機能のイベントを契機として動作します。

Java EE で規定されたイベントとデータベースセッションフェイルオーバ機能で発生するイベントの対応，およびイベントを契機として動作するリスナについて次の表に示します。

### 表 6-13　データベースセッションフェイルオーバ機能で発生するイベントと動作するリスナ

| 項番 | Java EE で規定されたイベント | 対応するイベント（データベースセッションフェイルオーバ機能使用時） | 動作するリスナ |
|---|---|---|---|
| 1 | HTTP セッション作成 | HTTP セッション作成 | javax.servlet.http.HttpSessionListener インタフェースの sessionCreated()メソッド |
| 2 | HTTP セッション無効化 | • HTTP セッション無効化<br>• Web アプリケーション停止 | • javax.servlet.http.HttpSessionListener インタフェースの sessionDestroyed()メソッド<br>• javax.servlet.http.HttpSessionAttributeListener インタフェースの attributeRemoved()メソッド※<br>• javax.servlet.http.HttpSessionBindingListener インタフェースの valueUnbound()メソッド※ |
| 3 | HTTP セッションの属性追加 | HTTP セッションの属性追加 | • javax.servlet.http.HttpSessionAttributeListener インタフェースの attributeAdded()メソッド<br>• javax.servlet.http.HttpSessionBindingListener インタフェースの valueBound()メソッド |
| 4 | HTTP セッションの属性変更 | HTTP セッションの属性変更 | javax.servlet.http.HttpSessionAttributeListener インタフェースの attributeReplaced()メソッド |
| 5 | HTTP セッションの属性削除 | • HTTP セッションの属性削除<br>• Web アプリケーション停止 | • javax.servlet.http.HttpSessionAttributeListener インタフェースの attributeRemoved()メソッド<br>• javax.servlet.http.HttpSessionBindingListener インタフェースの valueUnbound()メソッド |
| 6 | セッションの活性化 | グローバルセッションの引き継ぎ | javax.servlet.http.HttpSessionActivationListener インタフェースの sessionDidActivate()メソッド |
| 7 | セッションの非活性化 | （対応するイベントなし） | （動作するリスナなし） |

注※　イベント発生時に属性が追加されていた場合です。

このほかのリスナの動作については，データベースセッションフェイルオーバ機能を使用しない場合と同じです。

## 6.4.5　グローバルセッション情報のロック（完全性保障モードが有効の場合）

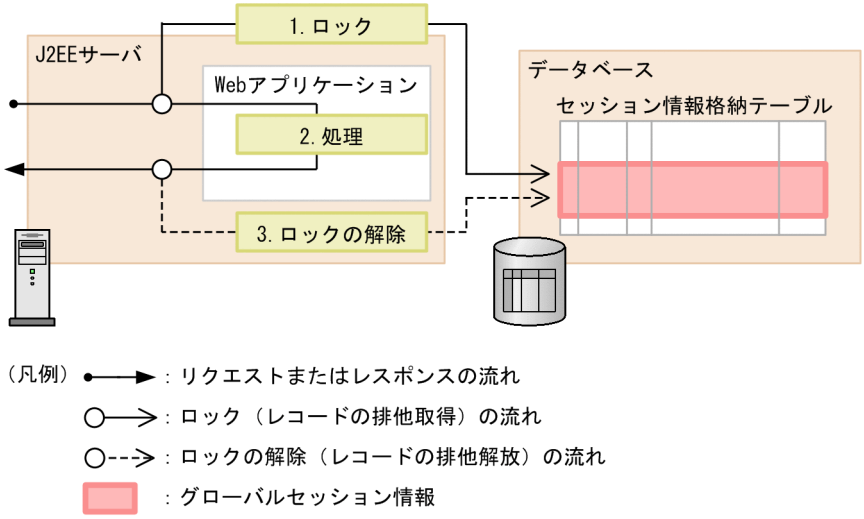
ここでは，完全性保障モードが有効の場合にだけ実行されるグローバルセッション情報のロックについて説明します。完全性保障モードが無効の場合は実施されません。

J2EE サーバを冗長化したシステムでは，異なる J2EE サーバに対して，同じセッション ID を持つリクエストが同時に送信される場合があります。例えば，フレーム構造のページや，複数の画像（image タグ）を含むページなどへのアクセスが発生した場合，ブラウザの機能によってマルチスレッドでサーバに対してリクエストが送信されます。

異なる J2EE サーバで同じグローバルセッションの情報が更新されると，グローバルセッション情報の完全性が失われてしまいます。そのため，データベースセッションフェイルオーバ機能では，ほかのサーバで使用できないように，更新中のグローバルセッション情報が格納されているレコードの排他を取得し，制御します。この排他の取得処理をグローバルセッション情報の**ロック**と呼びます。また，排他を解除する処理を**ロックの解除**と呼びます。

レコードの排他によるグローバルセッション情報のロックについて，次の図に示します。

図 6-7　レコードの排他によるグローバルセッション情報のロック




レコードの排他によるグローバルセッション情報のロックで実行される処理について説明します。項番は図中の番号に対応しています。

1. クライアントからリクエストを受け取ると，データベース上のグローバルセッション情報がロックされます。
2. ロック後に，Web アプリケーションの処理が実行されます。
3. Web アプリケーションの処理が完了すると，グローバルセッション情報のロックが解除されます。

このように，Web アプリケーションの動作中，データベース上のグローバルセッション情報はロックされているため，システム内で同じセッション ID を持つリクエストが同時に処理されないことが保証されます。

J2EE サーバにリクエストが送信された場合，Web アプリケーション内で HTTP セッションを使用するかどうかに関係なく，グローバルセッション情報がロックされます。ただし，次のリクエストに対しては，グローバルセッション情報はロックされません。

- HTTP セッションが作成されていないリクエスト
- データベースセッションフェイルオーバ機能を抑止された拡張子または URI を含む URL を持つリクエスト

  デフォルトの設定では，「txt」，「htm」，「html」，「jpg」，「gif」，および「js」が抑止対象の拡張子です。デフォルトの設定で，抑止対象となる URI はありません。データベースセッションフェイルオーバ機能の抑止については，「5.6.1　セッションフェイルオーバ機能の抑止」を参照してください。

データベースセッションフェイルオーバ機能では，異なる J2EE サーバではなく，一つの J2EE サーバに対して送信されるスレッド間でもグローバルセッションのロックは有効です。セッション ID が同一である複数のリクエストが一つの J2EE サーバに送信された場合，受信したリクエストから順に一つずつ処理されます。あとで受信したリクエストは，先に受信したリクエストの処理が終わるのを待ってから，処理を開始します。

**注意事項**

フレームなどを使用した，HTTP セッションを使用する複数の動的ページを組み合わせたコンテンツの更新などによって，Web クライアントからセッション ID が同一である複数のリクエストが送信される場合があります。この場合，受信したリクエストから順に一つずつ処理されます。その結果，データベースセッションフェイルオーバ機能を使用しない場合に比べ，処理性能が低下するおそれがあります。

## (1)　ロック取得時のロック取得処理の呼び出し結果

データベース上のグローバルセッション情報の状態によってロック取得処理の呼び出し結果が異なります。グローバルセッション情報の状態とロック取得処理の呼び出し結果の関係を次の表に示します。

### 表 6-14　グローバルセッション情報の状態とロック取得処理の呼び出し結果の関係

| 項番 | データベース上のグローバルセッション情報の状態 | ロック取得処理の呼び出し結果 | 出力されるメッセージ |
|---|---|---|---|
| 1 | 存在し，ロック中ではない（正常時）。 | データベース上のグローバルセッション情報がロックされます（正常終了）。 | 出力されない |
| 2 | 存在しない。 | タイムアウトで無効化したセッション，または無効なセッション ID のセッションを対象にしていると判断されます。そのため，J2EE サーバ内の HTTP セッションが削除されます。 | KDJE34315-W |

| 項番 | データベース上のグローバルセッション情報の状態 | ロック取得処理の呼び出し結果 | 出力されるメッセージ |
|---|---|---|---|
| | | その結果，Web アプリケーションは，HTTP セッションが存在しない状態で実行されます。 | |
| 3 | 存在するが，ほかの J2EE サーバで更新され，J2EE サーバ内の HTTP セッションの情報よりも新しい。 | ほかの J2EE サーバで使用されたグローバルセッション情報であると判断されます。そのため，データベース上のグローバルセッション情報の内容が引き継がれます（セッションの引き継ぎ※1 発生）。 | KDJE34322-I※2 |
| 4 | 存在し，使用されているためロック中である。 | ロック待ち※3 が発生します。HTTP セッション使用中のリクエストの処理が終了したあとでロックが取得され，Web アプリケーションが開始されます。 | 出力されない |

注※1

セッションの引き継ぎについては「6.4.2(4)　グローバルセッション情報を使用したセッション情報の引き継ぎ」を参照してください。

注※2

Warning レベルで出力されます。

注※3

ロック待ちについては「6.4.5(2)　ロック待ち」を参照してください。

## (2)　ロック待ち

ロック対象となっているグローバルセッション中の HTTP セッションを使用するリクエストを受信すると，ロックの取得を待つ必要があります。ロックの取得を待っている状態を，グローバルセッション情報の**ロック待ち**といいます。また，ロック待ちが原因で発生するタイムアウトのことを，**ロックタイムアウト**といいます。

ロック待ち発生後のグローバルセッション情報の状態と，ロック取得処理の呼び出し結果の関係を次の表に示します。

### 表 6-15　ロック待ち発生後のグローバルセッション情報の状態とロック取得処理の呼び出し結果の関係

| 項番 | ロック待ち発生後のグローバルセッション情報の状態 | ロック待ち発生後のロック取得処理の呼び出し結果 | 出力されるメッセージ |
|---|---|---|---|
| 1 | 先にセッションを使用していたリクエスト処理が終了し，ロックが解除された。 | データベースのグローバルセッション情報がロックされます（正常終了）。 | 出力されない |
| 2 | タイムアウト時間を経過したが，ロックは解除されなかった（ロックタイムアウトの発生）※1。 | Web アプリケーション内でセッション取得処理が実行されると，com.hitachi.software.web.dbsfo.DatabaseAccessException※2 がスローされます。 | KDJE34312-W |
| 3 | ロック待ちの間にデータベースで障害が発生し，ロックは解除されなかった（ロックタイムアウトの発生）。 | Web アプリケーション内でセッション取得処理が実行されると， | KDJE34312-W |

| 項番 | ロック待ち発生後のグローバルセッション情報の状態 | ロック待ち発生後のロック取得処理の呼び出し結果 | 出力されるメッセージ |
|---|---|---|---|
| | | com.hitachi.software.web.dbsfo.DatabaseAccessException※2 がスローされます。 | |

注※1

ロックをするための SQL をデータベースに送信し，通信路の障害などによってタイムアウトが発生した場合もこの状態に含まれます。

注※2

DatabaseAccessException クラスは，java.lang.IllegalStateException クラスを継承したクラスです。
DatabaseAccessException クラスの詳細については，マニュアル「アプリケーションサーバ リファレンス API 編」の「3.1 例外クラス」を参照してください。

### (3) J2EE サーバでの障害発生時のグローバルセッション情報のロック

Web アプリケーション実行中の J2EE サーバで，OS のハングアップやネットワークの障害などが発生すると，データベース上でロック中のグローバルセッション情報が一時的にロックされたままの状態となる場合があります。

ロックされたままの状態から回復するには，次のどちらかの対処が必要です。

- データベースに，クライアントからの接続を監視する設定，および接続の切断を検知して回復する設定をする。

  この設定をすると，データベースの機能によって一定時間後に J2EE サーバからの接続の切断が検知され，自動的にロックが解除されます。さらに，切断が検知されたタイミングで，ロック取得前の状態に戻ります。HiRDB を使用する場合，UAP 処理時間監視機能を設定してください。UAP 処理時間監視機能については，マニュアル「HiRDB UAP 開発ガイド」を参照してください。
- データベースの定期的なメンテナンスをする。

データベースの設定，運用については使用するデータベースのマニュアルを参照してください。

## 6.4.6 グローバルセッション情報操作中の障害発生時の動作

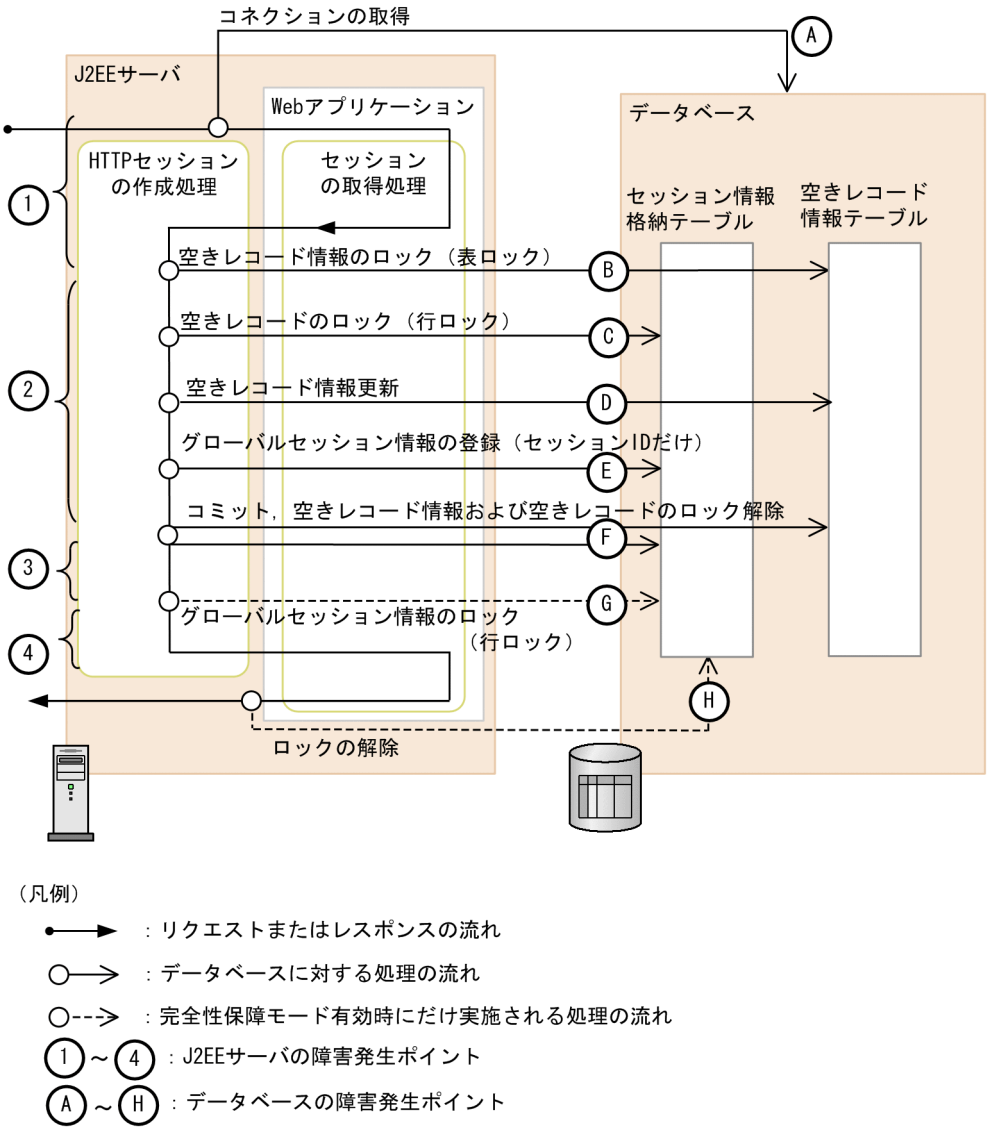
グローバルセッション情報の操作中の障害発生時の動作について説明します。ここでは，グローバルセッション情報の操作ごとに，障害が発生したポイント，セッションの状態，ほかのリクエストへの影響，および出力されるメッセージについて説明します。

### (1) グローバルセッション情報の作成時に障害が発生した場合の動作

グローバルセッション情報の作成時に，J2EE サーバ障害またはデータベース障害が発生した場合の動作について説明します。

グローバルセッション情報の作成処理の流れと障害発生のポイントを次の図に示します。

図 6-8　グローバルセッション情報の作成処理の流れと障害発生のポイント




以降の説明では，図中の数字（J2EE サーバの障害発生ポイント）またはアルファベット（データベースの障害発生ポイント）と，表中の障害発生ポイントの数字またはアルファベットが対応しています。

## (a)　J2EE サーバ障害発生時の動作（プロセスダウン）

グローバルセッション情報の作成時に J2EE サーバ障害が発生し，プロセスダウンした場合の動作を次の表に示します。

表 6-16　J2EE サーバ障害発生時の動作（プロセスダウン）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 |
|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | |
| 1 | 作成されない | 作成されない | なし |

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 |
|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | |
| 2 | 作成されない | 作成されない（ロールバック）※1 | データベースがクライアント接続を検知するまでの間，すべての HTTP セッションの新規作成はできない |
| 3 | 作成されない | 作成される※2 | なし |
| 4 | プロセスダウンによって消滅 | 作成される※2 | なし |

注※1　SQLException が発生し，リクエスト受信前の状態にロールバックします。

注※2　この状態のグローバルセッション情報の引き継ぎはできません。有効期限が切れると，有効期限監視によって削除されます。

## (b)　データベース障害発生時の動作（SQLException が発生した場合）

グローバルセッション情報の作成時にデータベース障害が発生し，SQLException が発生した場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

### 表 6-17　データベース障害で SQLException が発生した場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 縮退して作成される※1 | 作成されない | なし | 正常終了 | KDJE34368-W |
| B～F | 縮退して作成される※1 | 作成されない（ロールバック）※2 | なし | 正常終了 | KDJE34368-W |
| G | － | － | － | － | － |
| H | － | － | － | － | － |

（凡例）－：該当なし

注※1　縮退した HTTP セッションは，次回リクエスト受信時のグローバルセッション情報の更新処理でデータベースに更新されます。

注※2　SQLException が発生し，リクエスト受信前の状態にロールバックします。

### 表 6-18　データベース障害で SQLException が発生した場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 作成されない | 作成されない | なし | HTTP セッションの取得時に例外発生※1 | KDJE34314-W |

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| B～F | 作成されない | 作成されない（ロールバック）※2 | なし | HTTP セッションの取得時に例外発生※1 | KDJE34312-W |
| G | 作成されない | 作成される※3 | なし | HTTP セッションの取得時に例外発生※1 | KDJE34312-W |
| H | 作成されない（削除される） | 作成される※3 | なし | － | KDJE34312-W |

（凡例）－：該当なし

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2　SQLException が発生し，リクエスト受信前の状態にロールバックします。

注※3　この状態のグローバルセッション情報の引き継ぎはできません。有効期限が切れると，有効期限監視によって削除されます。

### (c)　データベース障害発生時の動作（データベースが無応答またはスローダウンしている場合）

グローバルセッション情報の作成時にデータベース障害が発生し，データベースが無応答またはスローダウンした場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

#### 表 6-19　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 縮退して作成される※1 | 作成されない | なし | 正常終了 | KDJE34368-W |
| B～F | 縮退して作成される※1 | 作成されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，すべての HTTP セッションの新規作成はできない | 正常終了 | KDJE34368-W |
| G | － | － | － | － | － |
| H | － | － | － | － | － |

（凡例）－：該当なし

注※1　縮退した HTTP セッションは，次回リクエスト受信時のグローバルセッション情報の更新処理でデータベースに更新されます。

注※2　データベースのロック解放待ちのタイムアウトが発生し，リクエスト受信前の状態にロールバックします。

表 6-20　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 作成されない | 作成されない | なし | HTTP セッションの取得時に例外発生※1 | KDJE34314-W |
| B～F | 作成されない | 作成されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，すべての HTTP セッションの新規作成はできない | HTTP セッションの取得時に例外発生※1 | KDJE34312-W |
| G | 作成されない | 作成される※3 | なし | HTTP セッションの取得時に例外発生※1 | KDJE34312-W |
| H | 作成されない（削除される） | 作成される※3 | なし | － | KDJE34312-W |

（凡例）－：該当なし

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2　データベースのロック解放待ちのタイムアウトが発生し，リクエスト受信前の状態にロールバックします。

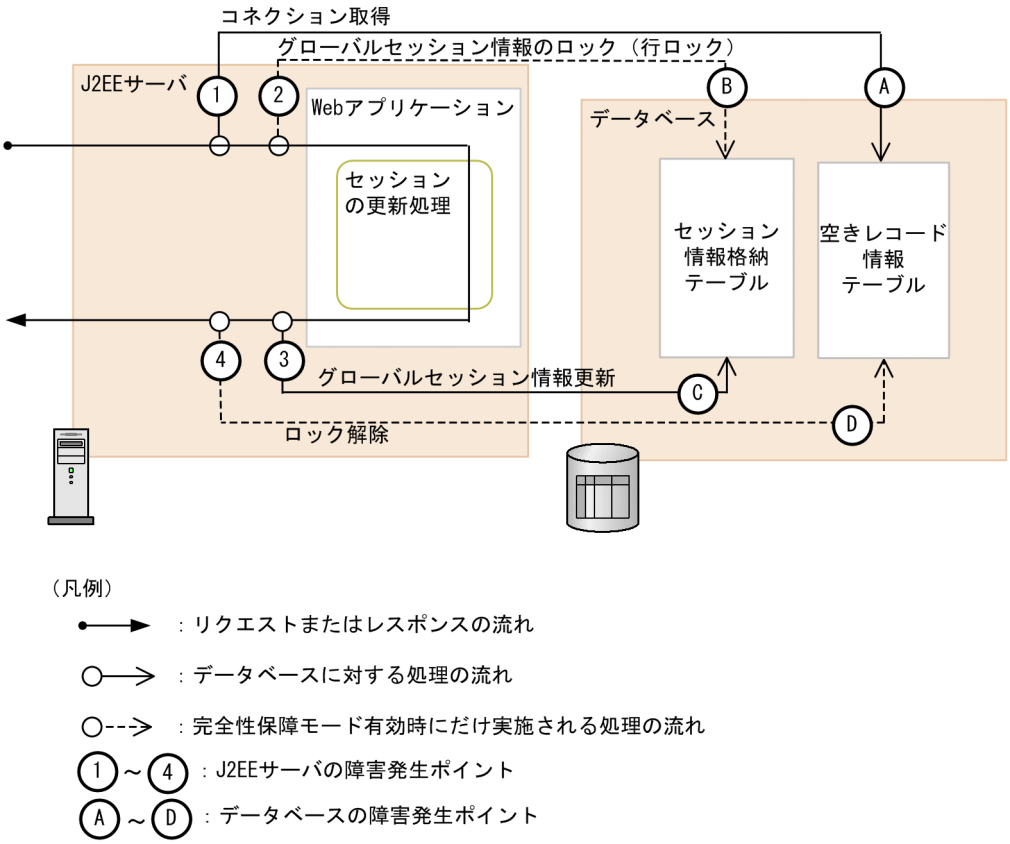
注※3　この状態のグローバルセッション情報の引き継ぎはできません。有効期限が切れると，有効期限監視によって削除されます。

## (2)　グローバルセッション情報更新時に障害が発生した場合の動作

グローバルセッション情報の更新時に，J2EE サーバ障害またはデータベース障害が発生した場合の動作について説明します。

グローバルセッション情報の更新処理の流れと障害発生のポイントを次の図に示します。

図 6-9　グローバルセッション情報の更新処理の流れと障害発生のポイント




## (a)　J2EE サーバ障害発生時の動作（プロセスダウン）

グローバルセッション情報の更新時に J2EE サーバ障害が発生し，プロセスダウンした場合の動作を次の表に示します。

表 6-21　J2EE サーバ障害発生時の動作（プロセスダウン）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 |
|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | |
| 1 | プロセスダウンによって消滅 | 更新されない | なし |
| 2 | プロセスダウンによって消滅 | 更新されない（ロールバック）※ | データベースがクライアント接続を検知するまでの間，該当する HTTP セッションの操作はできない |
| 3 | プロセスダウンによって消滅 | 更新されない（ロールバック）※ | データベースがクライアント接続を検知するまでの間，該当する HTTP セッションの操作はできない |
| 4 | プロセスダウンによって消滅 | 更新されない（ロールバック）※ | データベースがクライアント接続を検知するまでの間，該当する HTTP セッションの操作はできない |

注※　SQLException が発生し，リクエスト受信前の状態にロールバックします。

### (b) データベース障害発生時の動作（SQLException が発生した場合）

グローバルセッション情報の更新時にデータベース障害が発生し，SQLException が発生した場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

表 6-22 データベース障害で SQLException が発生した場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 縮退して更新される※ | 更新されない | なし | 正常終了 | KDJE34368-W |
| B | － | － | － | － | － |
| C | 縮退して更新される※ | 更新されない | なし | － | KDJE34368-W |
| D | － | － | － | － | － |

（凡例）－：該当なし

注※ 縮退した HTTP セッションは，次回リクエスト受信時のグローバルセッション情報の更新処理でデータベースに更新されます。

表 6-23 データベース障害で SQLException が発生した場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 更新されない | 更新されない | なし | HTTP セッションの取得時に例外発生※1 | KDJE34314-W |
| B | 更新されない（削除される） | 更新されない（ロールバック）※2 | なし | HTTP セッションの取得時に例外発生※1 | KDJE34312-W |
| C | 更新されない（削除される） | 更新されない（ロールバック）※2 | なし | － | KDJE34312-W |
| D | 更新されない（削除される） | 更新されない（ロールバック）※2 | なし | － | KDJE34312-W |

（凡例）－：該当なし

注※1 Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2 SQLException が発生し，リクエスト受信前の状態にロールバックします。

### (c) データベース障害発生時の動作（データベースが無応答またはスローダウンしている場合）

グローバルセッション情報の更新時にデータベース障害が発生し，データベースが無応答またはスローダウンした場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

表 6-24　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 縮退して更新される※1 | 更新されない | なし | 正常終了 | KDJE34368-W |
| B | － | － | － | － | － |
| C | 縮退して更新される※1 | 更新されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | － | KDJE34368-W |
| D | － | － | － | － | － |

（凡例）－：該当なし

注※1　縮退した HTTP セッションは，次回リクエスト受信時のグローバルセッション情報の更新処理でデータベースに更新されます。

注※2　データベースのロック解放待ちのタイムアウトが発生し，リクエスト受信前の状態にロールバックします。

表 6-25　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 更新されない | 更新されない | なし | HTTP セッションの取得時に例外発生※1 | KDJE34314-W |
| B | 更新されない（削除される） | 更新されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | HTTP セッションの取得時に例外発生※1 | KDJE34312-W |
| C | 更新されない（削除される） | 更新されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | － | KDJE34312-W |

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| D | 更新されない（削除される） | 更新されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | － | KDJE34312-W |

（凡例）－：該当なし

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

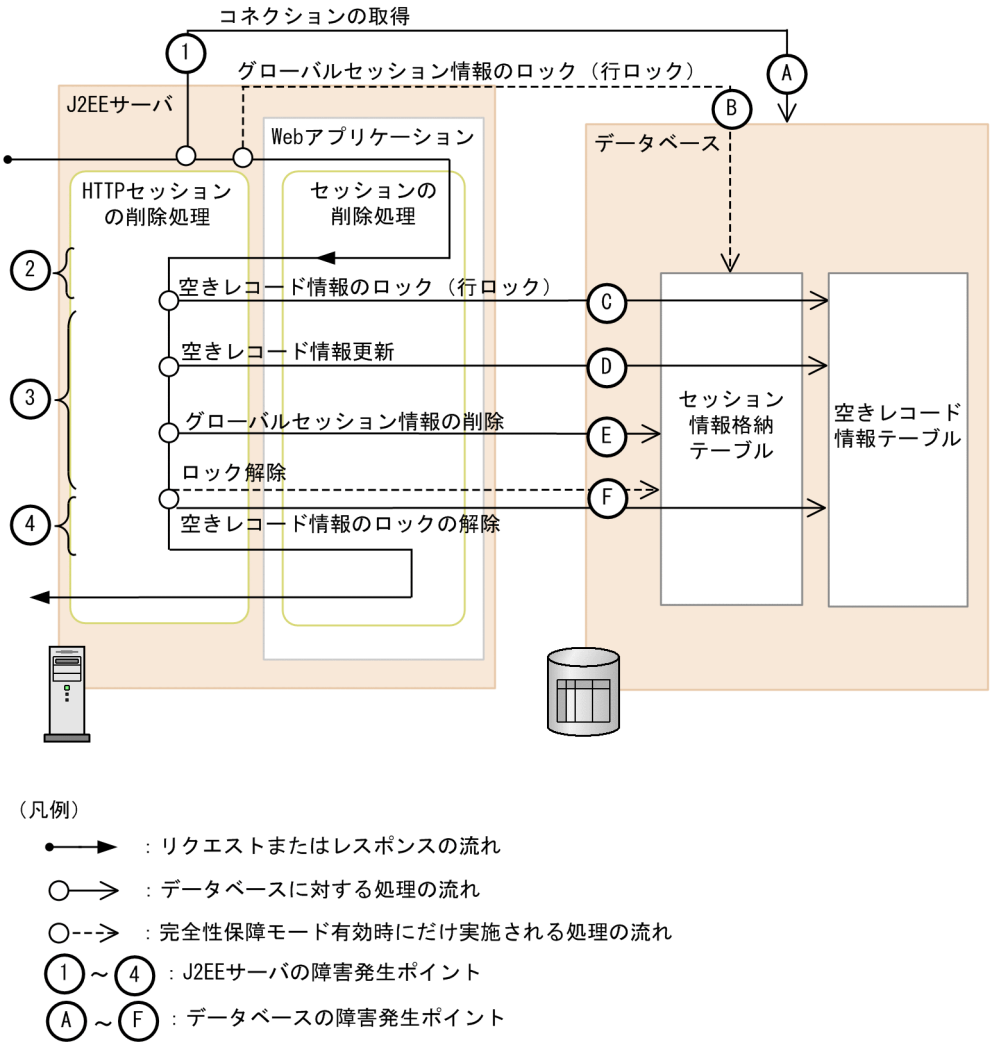
注※2　データベースのロック解放待ちのタイムアウトが発生し，リクエスト受信前の状態にロールバックします。

## (3)　グローバルセッション情報の削除時に障害が発生した場合の動作

グローバルセッション情報の削除時に，J2EE サーバ障害またはデータベース障害が発生した場合の動作について説明します。

グローバルセッション情報の削除処理の流れと障害発生のポイントを次の図に示します。

図 6-10　グローバルセッション情報の削除処理の流れと障害発生のポイント




## (a)　J2EE サーバ障害発生時の動作（プロセスダウン）

グローバルセッション情報の削除時に J2EE サーバ障害が発生し，プロセスダウンした場合の動作を次の表に示します。

表 6-26　J2EE サーバ障害発生時の動作（プロセスダウン）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 |
|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | |
| 1 | プロセスダウンによって消滅 | 削除されない | なし |
| 2 | プロセスダウンによって消滅 | 削除されない（ロールバック）※ | データベースがクライアント接続を検知するまでの間，該当する HTTP セッションの操作はできない |
| 3 | プロセスダウンによって消滅 | 削除されない（ロールバック）※ | データベースがクライアント接続を検知するまでの間，該当する HTTP セッションの操作はできない |

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 |
|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | |
| 4 | プロセスダウンによって消滅 | 削除されている | なし |

注※　SQLException が発生し，リクエスト受信前の状態にロールバックします。

## (b)　データベース障害発生時の動作（SQLException が発生した場合）

グローバルセッション情報の削除時にデータベース障害が発生し，SQLException が発生した場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

### 表 6-27　データベース障害で SQLException が発生した場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 削除される | 削除されない | なし | HTTP セッションの取得時に例外が発生※1 | KDJE34377-E※2 |
| B | － | － | － | － | － |
| C～F | 削除される | 削除されない（ロールバック）※3 | なし | HTTP セッションの無効化時に例外が発生※4 | KDJE34377-E※2 |

（凡例）－：該当なし

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2　初めての障害発生時にだけメッセージが出力されます。それ以降は Web アプリケーションを再開始するまで同じ障害ではメッセージは出力されません。

注※3　SQLException が発生し，リクエスト受信前の状態にロールバックします。

注※4　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの invalidate メソッドの呼び出しで，JSP の場合は暗黙オブジェクト session の invalidate メソッドの呼び出しで，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

### 表 6-28　データベース障害で SQLException が発生した場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 削除されない | 削除されない | なし | HTTP セッションの取得時に例外が発生※1 | KDJE34314-W |

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| B | 削除される | 削除されない（ロールバック）※2 | なし | HTTP セッションの取得時に例外が発生※1 | KDJE34312-W |
| C～F | 削除される | 削除されない（ロールバック）※2 | なし | HTTP セッションの無効化時に例外が発生※3 | KDJE34312-W |

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2　SQLException が発生し，リクエスト受信前の状態にロールバックします。

注※3　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの invalidate メソッドの呼び出しで，JSP の場合は暗黙オブジェクト session の invalidate メソッドの呼び出しで，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

## (c)　データベース障害発生時の動作（データベースが無応答またはスローダウンしている場合）

グローバルセッション情報の削除時にデータベース障害が発生し，データベースが無応答またはスローダウンした場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

### 表 6-29　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 削除される | 削除されない | なし | HTTP セッションの取得時に例外が発生※1 | KDJE34377-E※2 |
| B | － | － | － | － | － |
| C～F | 削除される | 削除されない（ロールバック）※3 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | HTTP セッションの無効化時に例外が発生※4 | KDJE34377-E※2 |

（凡例）－：該当なし

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2　初めての障害発生時にだけメッセージが出力されます。それ以降は Web アプリケーションを再開始するまで同じ障害ではメッセージは出力されません。

注※3　データベースのロック解放待ちのタイムアウトが発生し，リクエスト受信前の状態にロールバックします。

注※4　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの invalidate メソッドの呼び出しで，JSP の場合は暗黙オブジェクト session の invalidate メソッドの呼び出しで，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

表 6-30　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 削除されない | 削除されない | なし | HTTP セッションの取得時に例外が発生※1 | KDJE34314-W |
| B | 削除される | 削除されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | HTTP セッションの取得時に例外が発生※1 | KDJE34312-W |
| C～F | 削除される | 削除されない（ロールバック）※2 | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | HTTP セッションの無効化時に例外が発生※3 | KDJE34312-W |

注※1　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

注※2　データベースのロック解放待ちのタイムアウトが発生し，リクエスト受信前の状態にロールバックします。

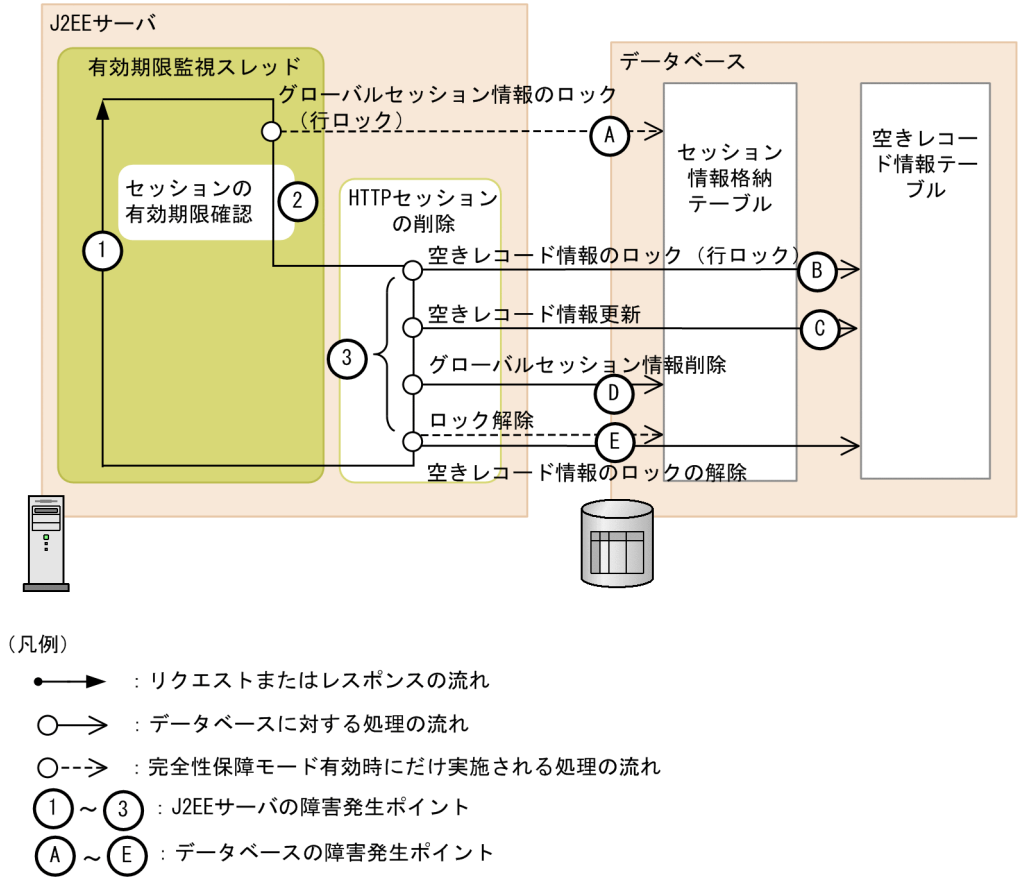
注※3　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの invalidate メソッドの呼び出しで，JSP の場合は暗黙オブジェクト session の invalidate メソッドの呼び出しで，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

## (4)　有効期限切れによるグローバルセッション情報削除時に障害が発生した場合の動作

有効期限切れによるグローバルセッション情報削除時に，J2EE サーバ障害またはデータベース障害が発生した場合の動作について説明します。

有効期限切れによるグローバルセッション情報削除処理の流れと障害発生のポイントを次の図に示します。

図 6-11　有効期限切れによるグローバルセッション情報削除処理の流れと障害発生のポイント




## (a)　J2EE サーバ障害発生時の動作（プロセスダウン）

有効期限切れによるグローバルセッション情報の削除時に J2EE サーバ障害が発生し，プロセスダウンした場合の動作を次の表に示します。

表 6-31　J2EE サーバ障害発生時の動作（プロセスダウン）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 |
|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | |
| 1 | プロセスダウンによって消滅 | 削除されない | なし |
| 2，3 | プロセスダウンによって消滅 | 削除されない（ロールバック） | データベースがクライアントの切断を検知するまでの間，該当する HTTP セッションの操作はできない |

## (b)　データベース障害発生時の動作（SQLException が発生した場合）

有効期限切れによるグローバルセッション情報の削除時にデータベース障害が発生し，SQLException が発生した場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

表 6-32　データベース障害で SQLException が発生した場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | － | － | － | － | － |
| B～E | 削除される | 削除されない（ロールバック） | なし | － | KDJE34377-E※ |

（凡例）－：該当なし

注※　初めての障害発生時にだけメッセージが出力されます。それ以降は Web アプリケーションを再開始するまで同じ障害ではメッセージは出力されません。

表 6-33　データベース障害で SQLException が発生した場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 削除される | 削除されない（ロールバック） | なし | － | KDJE34336-W |
| B～E | 削除される | 削除されない（ロールバック） | なし | － | KDJE34312-W |

（凡例）－：該当なし

### (c)　データベース障害発生時の動作（データベースが無応答またはスローダウンしている場合）

有効期限切れによるグローバルセッション情報の削除時にデータベース障害が発生し，データベースが無応答またはスローダウンした場合の動作を次の表に示します。なお，完全性保障モード無効時と有効時とでは動作が異なります。

表 6-34　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード無効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | － | － | － | － | － |
| B～E | 削除される | 削除されない（ロールバック） | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | － | KDJE34377-E※ |

（凡例）－：該当なし

注※　初めての障害発生時にだけメッセージが出力されます。それ以降は Web アプリケーションを再開始するまで同じ障害ではメッセージは出力されません。

表 6-35　データベース障害で無応答またはスローダウンしている場合の動作（完全性保障モード有効時）

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| A | 削除される | 削除されない（ロールバック） | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | － | KDJE34336-W |
| B～E | 削除される | 削除されない（ロールバック） | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | － | KDJE34312-W |

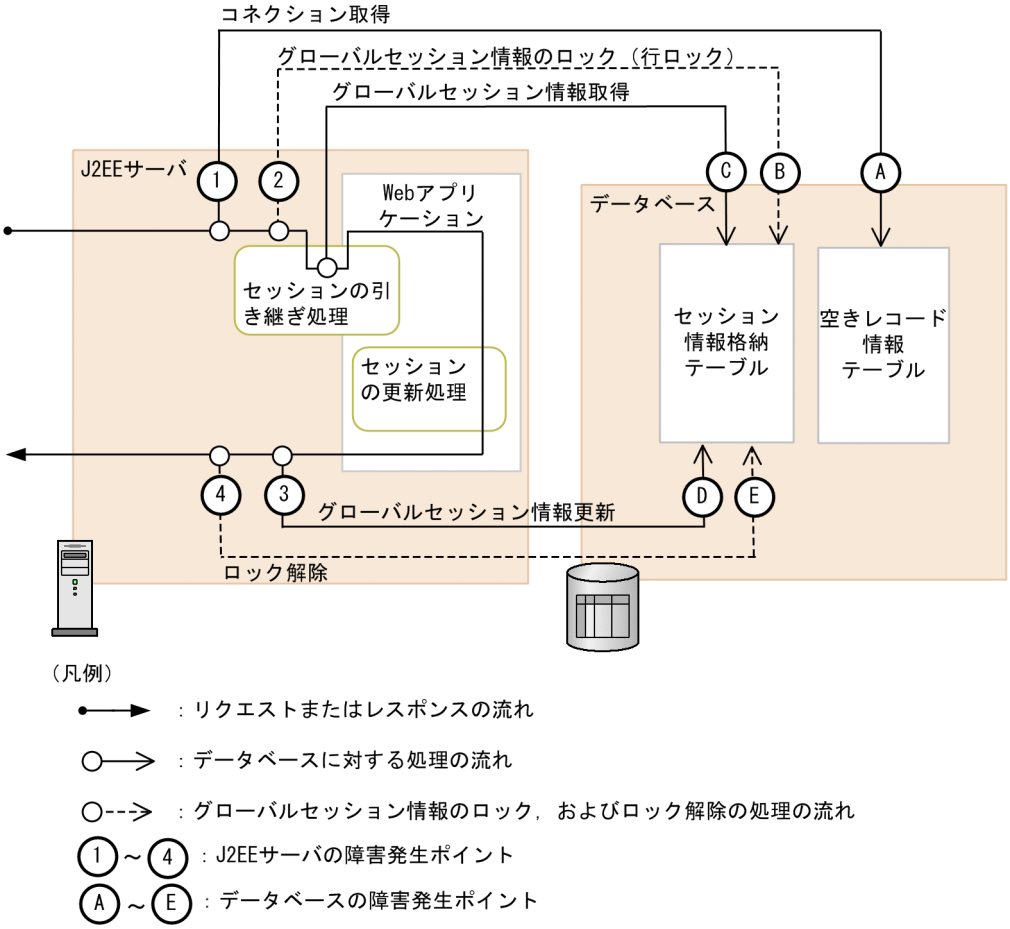
（凡例）－：該当なし

## (5)　グローバルセッション情報を使用したグローバルセッション引き継ぎ時に障害が発生した場合の動作

グローバルセッション情報を使用したグローバルセッション引き継ぎ時に，J2EE サーバ障害またはデータベース障害が発生した場合の動作について説明します。

グローバルセッション情報を使用したグローバルセッション引き継ぎ処理の流れと障害発生のポイントを次の図に示します。

図 6-12　グローバルセッション情報を使用したグローバルセッション引き継ぎ処理の流れと障害発生のポイント




## (a)　J2EE サーバ障害発生時の動作（プロセスダウン）

グローバルセッション情報を使用したグローバルセッション引き継ぎ時に J2EE サーバ障害が発生し，プロセスダウンした場合の動作は，グローバルセッション情報の更新時に J2EE サーバ障害が発生した場合と同じ動作になります。

グローバルセッション情報の更新時に J2EE サーバ障害が発生した場合の動作については，「6.4.6(2)　グローバルセッション情報更新時に障害が発生した場合の動作」の J2EE サーバ障害発生時の動作を参照してください。

## (b)　データベース障害発生時の動作（SQLException が発生した場合）

グローバルセッション情報を使用したグローバルセッション引き継ぎ時の，図中 C の処理中にデータベース障害が発生し，SQLException が発生した場合の動作を次の表に示します。図中 A，B，D，E の処理中に障害が発生した場合の動作は，グローバルセッション情報の更新時にデータベースで SQLException が発生した場合と同じ動作になります。

グローバルセッション情報の更新時にデータベース障害で SQLException が発生した場合の動作については，「6.4.6(2)　グローバルセッション情報更新時に障害が発生した場合の動作」のデータベース障害発生時の動作（SQLException が発生した場合）を参照してください。

表 6-36　データベース障害で SQLException が発生した場合の動作

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| C | 引き継がれない | 引き継がれない | なし | HTTP セッションの取得時に例外が発生※ | KDJE34314-W |

注※　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

### (c)　データベース障害発生時の動作（データベースが無応答またはスローダウンしている場合）

グローバルセッション情報を使用したグローバルセッション引き継ぎ時の，図中 C の処理中にデータベース障害が発生し，データベースが無応答，またはスローダウンした場合の動作を次の表に示します。図中 A，B，D，E の処理中に障害が発生した場合の動作は，グローバルセッション情報の更新時にデータベースが無応答またはスローダウンした発生した場合と同じ動作になります。

グローバルセッション情報の更新時にデータベース障害で無応答またはスローダウンした場合の動作については，「6.4.6(2)　グローバルセッション情報更新時に障害が発生した場合の動作」のデータベース障害発生時の動作（データベースが無応答またはスローダウンしている場合）を参照してください。

表 6-37　データベース障害で無応答またはスローダウンしている場合の動作

| 障害発生ポイント | セッションの状態 | | ほかのリクエストへの影響 | Web アプリケーションの動作 | メッセージ |
|---|---|---|---|---|---|
| | J2EE サーバの HTTP セッション | グローバルセッション情報 | | | |
| C | 引き継がれない | 引き継がれない | ロック解放待ちでタイムアウトするまでの間，該当する HTTP セッションの操作はできない | HTTP セッションの取得時に例外が発生※ | KDJE34314-W |

注※　Servlet の場合は javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドの呼び出しで，JSP の場合はユーザコードが実行される前に，com.hitachi.software.web.dbsfo.DatabaseAccessException が発生します。

## 6.5　cosminexus.xml での定義

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ための定義は，cosminexus.xml の<war>タグ内に指定します。

cosminexus.xml でのデータベースセッションフェイルオーバ機能の定義について次の表に示します。

### 表 6-38　cosminexus.xml でのデータベースセッションフェイルオーバ機能の定義

| 項目 | 指定するタグ | 設定内容 |
|---|---|---|
| データベースセッションフェイルオーバ機能の設定 | <http-session>-<dbsfo>-<enabled> | データベースセッションフェイルオーバ機能を有効にするかどうかを Web アプリケーション単位で設定します。 |
| HttpSession オブジェクト数の上限値 | <http-session>-<http-session-max-number> | HttpSession オブジェクト数の上限値を設定します。 |
| アプリケーション識別子 | <http-session>-<dbsfo>-<application-id> | アプリケーション識別子を設定します。 |
| HTTP セッションの属性情報の最大サイズ | <http-session>-<dbsfo>-<attribute-data-size-max> | グローバルセッション情報に含められる HTTP セッションの属性情報の最大サイズを設定します。 |
| 拡張子によるデータベースセッションフェイルオーバ機能の抑止 | <http-session>-<dbsfo>-<exclude-extensions> | データベースセッションフェイルオーバ機能を Web アプリケーション単位で有効にする場合に，データベースセッションフェイルオーバ機能を抑止する拡張子を設定します。 |

指定するタグの詳細は，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(アプリケーション／リソース定義)」の「2.2.6　War 属性の詳細」を参照してください。

## 6.6　実行環境での設定

### 6.6.1　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。データベースセッションフェイルオーバ機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのデータベースセッションフェイルオーバ機能の定義について次の表に示します。

表 6-39　簡易構築定義ファイルでのデータベースセッションフェイルオーバ機能の定義

| 項目 | 指定するパラメタ | 設定内容 | 参照先 |
|---|---|---|---|
| データベースセッションフェイルオーバ機能の設定 | webserver.dbsfo.enabled | J2EE サーバ単位で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かどうかを設定します。 | － |
| DB Connector の別名の設定 | webserver.dbsfo.connector.name | DB Connector で設定する，DB Connector の別名を指定します。<br>DB Connector の別名の設定については「6.6.4(2)　DB Connector の別名の設定」を参照してください。 | － |
| グローバルセッション情報に含めることができる HTTP セッションの属性情報の最大サイズの設定 | webserver.dbsfo.attribute_data_size.max | グローバルセッション情報に含めることができる HTTP セッションの属性情報の最大サイズを設定します。 | － |
| HTTP セッションの属性情報のサイズ見積もり機能の設定 | webserver.dbsfo.check_size.mode | HTTP セッションの属性情報のサイズ見積もり機能を使用するかどうかを設定します。 | － |
| 完全性保障モードの設定 | webserver.dbsfo.integrity_mode.enabled※1 | データベースセッションフェイルオーバ機能の完全性保障モードを有効にするかどうかを設定します。 | － |
| シリアライズ処理で使用するメモリ量の設定 | － | シリアライズ処理で使用するメモリを考慮して，JavaVM のチューニングをします※2。 | － |
| 拡張子によるデータベースセッションフェイルオーバ機能の抑止の設定 | webserver.dbsfo.exclude.extensions | J2EE サーバ単位で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，データベースセッションフェイルオーバ機能を抑止する拡張子を設定します。 | － |
| URI 単位のデータベースセッションフェイルオーバ機能の抑止の設定 | webserver.dbsfo.exclude.uris | J2EE サーバ単位で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，データベースセッションフェイルオーバ機能を抑止する URI を設定します。 | (1) |
| 参照専用リクエストの設定 | webserver.dbsfo.session_read_only.uris | 参照専用リクエストとする URI を設定します。 | (2) |

| 項目 | 指定するパラメタ | 設定内容 | 参照先 |
|---|---|---|---|
| HttpSession のサーバ ID 付加機能の設定 | • webserver.session.server_id.enabled<br>• webserver.session.server_id.value | HttpSession のサーバ ID 付加機能を設定します。また，サーバ ID には冗長化した J2EE サーバごとに異なる値を設定します※3。 | − |
| 同時実行スレッド数制御機能を使用する場合の実行待ちキュー不足時の設定 | webserver.dbsfo.thread_control_queue.enabled | Web アプリケーション単位の同時実行スレッド数制御機能が有効の場合に，実行待ちキューの空きが不足したときの動作を設定します。 | − |
| ネゴシエーション失敗時の Web アプリケーション開始処理の設定 | webserver.dbsfo.negotiation.high_level | ネゴシエーションが失敗した場合に Web アプリケーションの開始処理を続行するか中止するかを設定します。 | − |
| データベースセッションフェイルオーバ機能の抑止対象リクエスト内で HTTP セッションを使用した場合にスローされる例外の設定 | webserver.dbsfo.exception_type_backcompat | データベースセッションフェイルオーバ機能の抑止対象リクエスト内で HTTP セッションを使用した場合に発生する例外を設定します。 | − |

（凡例）−：該当なし。

注※1　完全性保障モードの設定で webserver.dbsfo.integrity_mode.enabled パラメタに false を指定した場合，HttpSession のサーバ ID 付加機能の設定で webserver.session.server_id.enabled パラメタ，および webserver.session.server_id.value パラメタを指定する必要があります。

注※2　シリアライズ処理で使用するメモリ量の見積もり方法については，「5.8.1　シリアライズ処理で使用するメモリの見積もり」を参照してください。JavaVM のチューニングについては，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.　JavaVM のメモリチューニング」を参照してください。

注※3　実行系と待機系による系切り替えシステムの場合，実行系と待機系の値は同じにしてください。

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，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。

## (1)　データベースセッションフェイルオーバ機能の抑止の設定

ここでは，URI 単位のデータベースセッションフェイルオーバ機能の抑止をする場合の，URI の指定方法について説明します。

### (a)　URI の指定方法

コンテキストパスを含む「/」(スラッシュ)で始まる URI を指定します。パスパラメタ，クエリ，およびフラグメントは含みません。なお，設定値の URI 中に「;」（セミコロン）を使用することはできません。

また，複数の URI を指定する場合は，「;」（セミコロン）で区切って指定してください。

### (b)　完全一致指定とプリフィックス一致指定

次のどちらかで指定できます。

### 完全一致指定

指定したURIがリクエストURIに完全一致した場合だけ，データベースセッションフェイルオーバ機能の抑止の対象になります。

#### 指定例（簡易構築定義ファイルの場合）

```
:
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
    <param>
      <param-name>webserver.dbsfo.exclude.uris</param-name>
      <param-value>/examples/test/TestServlet;/examples/test2/TestServlet2</param-value>
    </param>
</configuration>
:
```

この例の場合，次のリクエストURIがデータベースセッションフェイルオーバ機能の抑止対象になります。

- http://host/examples/test/TestServlet
- http://host/examples/test/TestServlet?name=value
- http://host/examples/test/TestServlet;gsessionid=XXXXXXXXXX

### プリフィックス一致指定

リクエストURIとプリフィックスが一致する場合，データベースセッションフェイルオーバ機能の抑止の対象になります。

#### 指定例（簡易構築定義ファイルの場合）

```
:
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
    <param>
      <param-name>webserver.dbsfo.exclude.uris</param-name>
      <param-value>/examples/*</param-value>
    </param>
</configuration>
:
```

この例の場合，次のリクエストURIがデータベースセッションフェイルオーバ機能の抑止対象になります。

- http://host/examples/test/TestServlet
- http://host/examples/dbsfo/DbsfoServlet?name=value

なお，プリフィックス一致指定の場合，URIの指定は「/*」で終了する必要があります。例えば，次のようなURIを指定した場合は，プリフィックス一致指定ではなく，完全一致指定と扱われます。

- /examples/test*

### (c) URI の正規化

データベースセッションフェイルオーバ機能の抑止対象とする URI は，正規化して指定する必要があります。正規化していない URI を指定した場合，KDJE34341-W のメッセージが出力されて，該当する URI は抑止の対象外になります。

正規化した URI の例を次に示します。

- /examples/test/servlet/TestServlet

正規化していない URI の例を次に示します。これらの URI は抑止の対象外になります。

- /examples/test/jsp/../servlet/TestServlet
- /examples/test/./servlet/TestServlet

### (d) URL エンコードとの対応

URL エンコードをした URI を抑止対象として指定した場合は，指定した URI と一致する，URL エンコードされた URL のリクエストがデータベースセッションフェイルオーバ機能の抑止対象になります。同様に，URL エンコードをしない URI を指定した場合は，URL エンコードされていない URL のリクエストがデータベースセッションフェイルオーバ機能の抑止対象になります。

ただし，URI のデコード機能を使用する場合，対象の URL は，デコードが実施されたあとで URI によるデータベースフェイルオーバ機能の抑止の対象かどうかが判定されます。このため，URL エンコードされた URL が抑止対象として指定した URI と一致する場合に，URI 単位のデータベースセッションフェイルオーバ機能の抑止対象になります。

URI のデコード機能の有効・無効によって抑止の対象となる URL について，次の表に示します。

#### 表 6-40 URI のデコード機能の有効・無効によって抑止の対象となる URL

| プロパティ<br>設定値 | リクエスト URL | | | |
|---|---|---|---|---|
| | URI のデコード機能 有効 | | URI のデコード機能 無効 | |
| | エンコードあり | エンコードなし | エンコードあり | エンコードなし |
| エンコードあり | 抑止しない | 抑止しない | 抑止する | 抑止しない |
| エンコードなし | 抑止する | 抑止する | 抑止しない | 抑止する |

(凡例)

抑止する：データベースセッションフェイルオーバ機能を抑止する（データベースセッションフェイルオーバ機能が無効になる)。

抑止しない：データベースセッションフェイルオーバ機能を抑止しない（データベースセッションフェイルオーバ機能が有効になる)。

エンコードあり：URL エンコードされた文字列を含む URI。

(例：/examples/%61/Servlet)

エンコードなし：URL エンコードされた文字列を含まない URI。

(例：/examples/a/Servlet)

### (e) URI 指定時の注意事項

URI によるデータベースセッションフェイルオーバ機能の抑止で設定する URI は，ネゴシエーション時に確認される項目ではありません。このため，それぞれの J2EE サーバで設定する URI が同じことを確認してください。

## (2) 参照専用リクエストの設定

ここでは，参照専用リクエストを設定する場合の，URI の指定方法について説明します。

### (a) URI の指定方法

コンテキストパスを含む「/」(スラッシュ)で始まる URI を指定します。パスパラメタ，クエリ，およびフラグメントは含みません。指定できる文字数は 512 文字までです。なお，設定値の URI 中に「;」（セミコロン）を使用することはできません。

また，複数の URI を指定する場合は，「;」（セミコロン）で区切って指定してください。

### (b) 完全一致指定とプリフィックス一致指定

次のどちらかで指定できます。

#### 完全一致指定

指定した URI がリクエスト URI に完全一致した場合だけ，参照専用リクエストになります。

**指定例（簡易構築定義ファイルの場合）**

```
 :
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
    <param>
      <param-name>webserver.dbsfo.session_read_only.uris</param-name>
      <param-value>/examples/test/TestServlet;/examples/test2/TestServlet2</param-value>
    </param>
</configuration>
 :
```

この例の場合，次のリクエスト URI が参照専用リクエストになります。

- http://host/examples/test/TestServlet
- http://host/examples/test/TestServlet?name=value
- http://host/examples/test/TestServlet;gsessionid=XXXXXXXXXX

#### プリフィックス一致指定

リクエスト URI とプリフィックスが一致する場合，参照専用リクエストになります。

**指定例（簡易構築定義ファイルの場合）**

```
 :
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
    <param>
```

```
      <param-name>webserver.dbsfo.session_read_only.uris</param-name>
      <param-value>/examples/*</param-value>
    </param>
</configuration>
 :
```

この例の場合，次のリクエスト URI が参照専用リクエストになります。

- http://host/examples/test/TestServlet
- http://host/examples/dbsfo/DbsfoServlet?name=value

なお，プリフィックス一致指定の場合，URI の指定は「/*」で終了する必要があります。例えば，次のような URI を指定した場合は，プリフィックス一致指定ではなく，完全一致指定と扱われます。

- /examples/test*

### (c) URI の正規化

参照専用リクエストとする URI は，正規化して指定する必要があります。正規化していない URI を指定した場合，KDJE34357-W のメッセージが出力されて，該当する URI は参照専用リクエストになりません。

正規化した URI の例を次に示します。

- /examples/test/servlet/TestServlet

正規化していない URI の例を次に示します。これらの URI は参照専用リクエストになりません。

- /examples/test/jsp/../servlet/TestServlet
- /examples/test/./servlet/TestServlet

### (d) URL エンコードとの対応

URL エンコードをした URI を参照専用リクエストとして指定した場合は，指定した URI と一致する，URL エンコードされた URL のリクエストが参照専用リクエストになります。同様に，URL エンコードをしない URI を指定した場合は，URL エンコードされていない URL のリクエストが参照専用リクエストになります。

ただし，URI のデコード機能を使用する場合，対象の URL は，デコードが実施されたあとで URI による参照専用リクエストかどうかが判定されます。このため，URL エンコードされた URL が参照専用リクエストとして指定した URI と一致する場合に，URI 単位の参照専用リクエストになります。

URI のデコード機能の有効・無効によって参照専用リクエストになる URL について，次の表に示します。

表 6-41　URI のデコード機能の有効・無効によって参照専用リクエストになる URL

| プロパティ<br>設定値 | リクエスト URL | | | |
|---|---|---|---|---|
| | URI のデコード機能 有効 | | URI のデコード機能 無効 | |
| | エンコードあり | エンコードなし | エンコードあり | エンコードなし |
| エンコードあり | 参照専用リクエストにならない | 参照専用リクエストにならない | 参照専用リクエストになる | 参照専用リクエストにならない |
| エンコードなし | 参照専用リクエストになる | 参照専用リクエストになる | 参照専用リクエストにならない | 参照専用リクエストになる |

（凡例）

参照専用リクエストになる：リクエスト URL が参照専用リクエストになる。

参照専用リクエストにならない：リクエスト URL が参照専用リクエストにならない。

エンコードあり：URL エンコードされた文字列を含む URI。

（例：/examples/%61/Servlet）

エンコードなし：URL エンコードされた文字列を含まない URI。

（例：/examples/a/Servlet）

## 6.6.2　Web アプリケーションの設定

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。データベースセッションフェイルオーバ機能の定義には，WAR 属性ファイルを使用します。

WAR 属性ファイルで指定するタグは，cosminexus.xml と対応しています。cosminexus.xml での定義については，「6.5　cosminexus.xml での定義」を参照してください。

## 6.6.3　データベースの設定

この節では，データベースセッションフェイルオーバ機能を使用する場合に必要となる，テーブルの作成，および環境設定について説明します。

**注意事項**

テーブルを作成する際，テンプレートファイルについてここで説明しない変更をした場合，データベースセッションフェイルオーバ機能の動作は保障されません。

### (1)　データベース接続に必要な権限

データベースのテーブルを操作するには権限が必要です。また，条件を満たす必要があります。データベースごとのテーブルの操作に必要な権限および条件について説明します。ここでは，データベースに接続するユーザのことを**データベース接続ユーザ**といいます。

- HiRDB の場合

  ここでは，データベース接続ユーザが，データベースセッションフェイルオーバ機能で使用するテーブルに関するすべての操作を実施することを想定しています。データベース接続ユーザは，次の権限および条件を満たす必要があります。

  - スキーマを所有すること。
  - CONNECT 権限を持つこと。
  - データベース接続ユーザのスキーマに，データベースセッションフェイルオーバ機能が使用するテーブル，インデックスおよびストアドプロシジャを作成すること。

  データベースのテーブルの作成の詳細については，「6.6.3(2)　データベースのテーブルの作成」を参照してください。データベースのテーブルの削除の詳細については，「6.8　データベースのテーブルの削除」を参照してください。

- Oracle の場合

  ここでは，データベースセッションフェイルオーバ機能が使用するデータベースのテーブルの作成または削除の操作はデータベース管理者が実施し，そのほかの通常の運用で必要となるデータベースの操作は，データベースセッションフェイルオーバ機能のデータベース接続ユーザが実施することを想定しています。データベース接続ユーザは，次の権限および条件を満たす必要があります。

  - CREATE SESSION システム権限を持つこと。
  - データベース接続ユーザのスキーマに，データベースセッションフェイルオーバ機能が使用するテーブル，インデックスおよびストアドプロシジャを作成すること。

  データベースのテーブルの作成の詳細については，「6.6.3(2)　データベースのテーブルの作成」を参照してください。データベースのテーブルの削除の詳細については，「6.8　データベースのテーブルの削除」を参照してください。

## (2)　データベースのテーブルの作成

データベースセッションフェイルオーバ機能では，データベース上に 3 種類のテーブルを作成する必要があります。作成するテーブルと，作成手順の参照先について次の表に示します。

### 表 6-42　作成するテーブルと，作成手順の参照先

| テーブル名 | データベース上の物理名称 | 作成手順の参照先 |
|---|---|---|
| アプリケーション情報テーブル | SFO_<APPLICATION_ID>_APP_INFO | 6.6.3(3) |
| セッション情報格納テーブル | SFO_<APPLICATION_ID>_SESSIONS | 6.6.3(4) |
| 空きレコード情報テーブル | SFO_<APPLICATION_ID>_REC_INFO | |

データベースセッションフェイルオーバ機能で使用するデータベースのテーブル作成用のテンプレートファイルは次の場所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¥sfo¥sql¥
```

UNIX の場合：

/opt/Cosminexus/CC/sfo/sql/

テーブル作成用のテンプレートファイルは使用するデータベースごとに 2 種類ずつあります。使用するデータベース，ファイル，および作成するテーブルの種類の対応を次の表に示します。

表 6-43　テーブル作成用テンプレートファイルと作成するテーブル

| 使用するデータベース | テンプレートファイル | 作成するテーブルの種類 | | |
|---|---|---|---|---|
| | | アプリケーション情報テーブル | セッション情報格納テーブル | 空きレコード情報テーブル |
| HiRDB | hirdb_create_apptbl.sql | ○ | − | |
| | hirdb_create_sessiontbl.sql | − | ○ | |
| Oracle | oracle_create_apptbl.sql | ○ | − | |
| | oracle_create_sessiontbl.sql | − | ○ | |

（凡例）○：作成できる　−：作成できない

以降で，使用するデータベースごとにテンプレートファイルの詳細について示します。

また，DB Connector に設定するユーザには，テーブルの作成者を登録してください。

## (3)　アプリケーション情報テーブルの作成

アプリケーション情報テーブルは，Web アプリケーションに設定したデータベースセッションフェイルオーバ機能に関する設定を格納するテーブルです。

アプリケーション情報テーブルの作成手順を次に示します。

1. テンプレートファイルを任意の場所にコピーします。

   テーブル作成用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレートファイルの格納場所を，使用するデータベースごとに次に示します。

   - HiRDB を使用する場合のテンプレートファイルの格納場所

     Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥hirdb_create_apptbl.sql

     UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_create_apptbl.sql

   - Oracle を使用する場合のテンプレートファイルの格納場所

     Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥oracle_create_apptbl.sql

     UNIX の場合：/opt/Cosminexus/CC/sfo/sql/oracle_create_apptbl.sql

2. テンプレートファイルを編集します。

Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，テーブル作成用 SQL ファイルを作成します。

テンプレートファイル内の変更個所と変更内容を次の表に示します。

表 6-44　テンプレートファイル内の変更個所と変更内容

| 変更個所 | | 変更対象 | 変更内容 |
|---|---|---|---|
| HiRDB | Oracle | | |
| • 1 行目<br>• 5 行目 | • 1 行目<br>• 5 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |
| なし | • 1 行目<br>• 5 行目 | <SCHEMA_NAME> | データベース接続ユーザのスキーマ名に変更してください。 |
| 6 行目 | 6 行目 | <HTTP_SESSION_NO> | データベースに格納するグローバルセッション情報の数に変更してください。 |

3. 作成したテーブル作成用 SQL ファイルを実行します。

SQL ファイルの実行には，HiRDB を使用する場合は SQL Executer，Oracle を使用する場合は SQL*Plus などを使用してください。

## (4)　セッション情報格納テーブルおよび空きレコード情報テーブルの作成

セッション情報格納テーブルは，グローバルセッション情報を格納するテーブルです。空きレコード情報テーブルは，セッション情報格納テーブルの未使用レコードを管理するテーブルです。セッション情報格納テーブルおよび空きレコード情報テーブルは，一つのテーブル作成用の SQL ファイルを実行することで同時に作成されます。

セッション情報格納テーブルおよび空きレコード情報テーブルの作成手順を次に示します。

1. テンプレートファイルを任意の場所にコピーします。

テーブル作成用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレートファイルの格納場所を，使用するデータベースごとに次に示します。

- HiRDB を使用する場合のテンプレートファイルの格納場所

  Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥hirdb_create_sessiontbl.sql

  UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_create_sessiontbl.sql

- Oracle を使用する場合のテンプレートファイルの格納場所

  Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥oracle_create_sessiontbl.sql

  UNIX の場合：/opt/Cosminexus/CC/sfo/sql/oracle_create_sessiontbl.sql

2. テンプレートファイルを編集します。

Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，テーブル作成用 SQL ファイルを作成します。

テンプレートファイル内の，使用するデータベースごとの変更個所と変更内容について次の表に示します。

表 6-45　テンプレートファイル内の変更個所と変更内容

| 変更個所 | | 変更対象 | 変更内容 |
|---|---|---|---|
| HiRDB | Oracle | | |
| • 1 行目<br>• 13 行目<br>• 18 行目<br>• 19 行目<br>• 23 行目<br>• 48 行目<br>• 50 行目<br>• 57 行目<br>• 60 行目<br>• 74 行目 | • 1 行目<br>• 13 行目<br>• 18 行目<br>• 19 行目<br>• 23 行目<br>• 49 行目<br>• 51 行目<br>• 58 行目<br>• 61 行目<br>• 74 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |
| なし | • 1 行目<br>• 13 行目<br>• 18 行目<br>• 19 行目<br>• 23 行目<br>• 49 行目<br>• 51 行目<br>• 58 行目<br>• 60 行目<br>• 74 行目 | <SCHEMA_NAME> | データベース接続ユーザのスキーマ名に変更してください。 |
| 7 行目 | なし | <ATTRIBUTE_DATA_SIZE_MAX> | HTTP セッションの属性情報の最大サイズ（単位：バイト）に変更してください。 |
| 74 行目 | 74 行目 | <HTTP_SESSION_NO> | データベースに格納するグローバルセッション情報の数に変更してください。 |

3. 作成したテーブル作成用 SQL ファイルを実行します。

SQL ファイルの実行には，HiRDB を使用する場合は SQL Executer，Oracle を使用する場合は SQL*Plus などを使用してください。

## (5)　データベースの環境設定

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，データベースにはタイムアウト（HiRDB の場合 UAP 処理時間監視機能）の設定をしてください。

データベースセッションフェイルオーバ機能が有効の場合，機能の処理の中で操作対象となるデータベースのテーブルのレコードが排他制御されます。そのため，J2EE サーバのホストでの障害発生時などに，操作対象となっていたレコードが排他されたままになる場合があります。このとき，HTTP セッションの新規作成や，J2EE サーバとデータベースとの接続に失敗するおそれがあります。

タイムアウトの設定をしておくと，このような状況を検知してタイムアウト時にトランザクションがロールバックされ，レコードが排他制御される前の状態に戻るため，システムへの影響はなくなります。

なお，誤動作を防ぐために，データベースのタイムアウトの値には DB Connector に設定するタイムアウトの値よりも大きな値を設定してください。データベースの設定内容，手順については，HiRDB を使用する場合はマニュアル「HiRDB UAP 開発ガイド」を，Oracle を使用する場合は Oracle のマニュアルを参照してください。

レコードが排他制御される処理，処理中に操作対象となるテーブル，処理中に J2EE サーバで障害が発生した場合のシステムへの影響，および出力されるメッセージを次の表に示します。

### 表 6-46　レコードが排他制御される処理と処理中に J2EE サーバで障害が発生した場合のシステムへの影響

| 項番 | レコードが排他制御される処理 | 操作対象のテーブル | 障害が発生した場合のシステムへの影響 | 出力されるメッセージ |
|---|---|---|---|---|
| 1 | Web アプリケーション開始時のネゴシエーション処理 | アプリケーション情報テーブル | アプリケーションのネゴシエーションに失敗するため，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 Web アプリケーションの開始に失敗します。 | 出力されない |
| 2 | グローバルセッション情報の作成処理 | 空きレコード情報テーブル | システムで作成できる HTTP セッションの数が全体の 90%になります。このあと，HTTP セッションの作成，または削除処理に失敗する場合があります。 | • 完全性保障モード無効時：KDJE34368-W<br>• 完全性保障モード有効時：KDJE34312-W |
| 3 | グローバルセッション情報の削除処理 | 空きレコード情報テーブル | システムで作成できる HTTP セッションの数が全体の 90%になります。このあと，HTTP セッションの作成，または削除処理に失敗する場合があります。 | • 完全性保障モード無効時：KDJE34377-E<br>• 完全性保障モード有効時：KDJE34312-W |
| 4 | グローバルセッション情報の更新処理 | セッション情報格納テーブル | システムで作成できる HTTP セッションの数が 1 個分減少します。このあと，減少した HTTP セッションを操作するリクエストを受信すると，HTTP セッションの取得に失敗します。 | • 完全性保障モード無効時：KDJE34368-W<br>• 完全性保障モード有効時：KDJE34312-W |
| 5 | グローバルセッション情報の有効期限監視処理 | アプリケーション情報テーブル | 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が監視されなくなります。 | • 完全性保障モード無効時：排他処理を行わない |

| 項番 | レコードが排他制御される処理 | 操作対象のテーブル | 障害が発生した場合のシステムへの影響 | 出力されるメッセージ |
|---|---|---|---|---|
| | | | | • 完全性保障モード有効時：KDJE34336-W |

## 6.6.4　DB Connector の設定

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，アプリケーションで使用するものとは別に，DB Connector を新規に作成します。DB Connector は，J2EE サーバごとに一つ必要です。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アプリケーションは，すべて同一の DB Connector を使用します。

DB Connector のインポートから開始までの手順については，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　データベースと接続するための設定」を参照してください。

この節では，データベースセッションフェイルオーバ機能で使用する DB Connector に必要な次の設定について説明します。

- トランザクションサポートのレベルの設定
- DB Connector の別名の設定
- DB Connector の環境設定

### (1)　トランザクションサポートのレベルの設定

データベースセッションフェイルオーバ機能ではトランザクションサポートのレベルを設定する必要があります。Connector 属性ファイルの<hitachi-connector-property>-<resourceadapter>-<outbound-resourceadapter>タグ以下の<transaction-support>タグに`NoTransaction` を指定します。

### (2)　DB Connector の別名の設定

データベースセッションフェイルオーバ機能では DB Connector に別名を設定する必要があります。デフォルトでは"COSMINEXUS_SFO_DBCONNECTOR"が DB Connector に別名として設定されます。

設定する名称をデフォルトの名称から変更する場合，Connector 属性ファイルの<hitachi-connector-property>-<resourceadapter>-<outbound-resourceadapter>-<connection-definition>-<connector-runtime>-<resource-external-property>タグ以下の<optional-name>タグに任意の名称を指定します。DB Connector の別名の設定については，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(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。

また，J2EE サーバに定義する DB Connector の別名も同じ値に変更する必要があります。J2EE サーバの DB Connector の別名の設定については「6.6.1　J2EE サーバの設定」を参照してください。

## (3)　DB Connector の環境設定

データベースセッションフェイルオーバ機能は 24 時間連続稼働を実現するための機能です。連続稼働を実現するには，データベースに障害が発生した場合もシステムに影響を与えないために，コネクションの障害検知などの設定が必要です。障害回復に必要な時間を考慮して，値を設定してください。

コネクションの障害検知については，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「3.15.1　コネクションの障害検知」を参照してください。

DB Connector に設定するプロパティおよび設定方法の詳細については，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「4.2.2　DB Connector のプロパティ定義」を参照してください。

DB Connector に設定が必要なプロパティについて説明します。

### (a)　<config-property>タグで設定するプロパティ

<config-property>タグで設定するプロパティについて使用するデータベースごとに示します。なお，ここで示していないプロパティについては，データベースセッションフェイルオーバ機能に関する設定は不要です。

**注意事項**

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，ステートメントプーリングを設定する必要があります。ステートメントプーリングは，J2EE サーバのメモリ使用量に大きく影響を与えます。そのため，コネクションプーリングの設定も考慮して，PreparedStatementPoolSize プロパティの指定値を決定してください。

ステートメントプーリングについては，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「3.14.4　ステートメントプーリング」を参照してください。コネクションプーリングについては，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「3.14.1　コネクションプーリング」を参照してください。
PreparedStatementPoolSize プロパティに指定できる値については，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「4.1.10　DB Connector に設定する<config-property>タグに指定できるプロパティ」を参照してください。なお，ステートメント一つ当たりの使用するメモリサイズについては，JDBC 関連ドキュメントを参照してください。

**●HiRDB を使用する場合に設定するプロパティ**

HiRDB を使用する場合に設定するプロパティについて次の表に示します。

表 6-47　HiRDB を使用する場合に設定するプロパティ

| <config-property-name>タグに指定する値 | <config-property-type>タグに指定する値 | <config-property-value>タグに指定する内容または値 | 必須／任意 |
|---|---|---|---|
| description | java.lang.String | データベースへの接続に必要な接続付加情報を指定します。 | 必須 |
| DBHostName | java.lang.String | 接続する HiRDB のホスト名を指定します。 | 必須 |
| loginTimeout | java.lang.Integer | getConnection メソッドで Connection オブジェクトを取得する際の，HiRDB サーバとの物理接続確立の最大待ち時間（秒）を指定します。 | 任意 |
| LONGVARBINARY_Access | java.lang.String | 「LOCATOR」を指定します。 | 必須 |
| PreparedStatementPoolSize | java.lang.Integer | 30×<J2EE サーバ内の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 Web アプリケーション数>で求められる数値を指定します。 | 必須 |
| CancelStatement | java.lang.Boolean | 「true」を指定します。 | 必須 |
| logLevel | java.lang.String | DB Connector が出力するログトレースのレベルについて，任意のレベルを指定します。 | 任意 |

（凡例）必須：必ず指定する　任意：必要に応じて設定する

### 注意事項

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，クライアント環境定義で次の値を設定してください。

| 環境変数名 | 値 |
|---|---|
| PDISLLVL | 2 |
| PDFORUPDATEEXLOCK | NO |
| PDDBLOG | ALL |

●Oracle を使用する場合に設定するプロパティ

Oracle を使用する場合に設定するプロパティについて次の表に示します。

表 6-48　Oracle を使用する場合に設定するプロパティ

| <config-property-name>タグに指定する値 | <config-property-type>タグに指定する値 | <config-property-value>タグに指定する内容または値 | 必須／任意 |
|---|---|---|---|
| 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。 | 必須※ |
| 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。 | 必須※ |
| 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。 | 必須※ |
| 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。 | 必須※ |
| loginTimeout | java.lang.Integer | データベースへの接続試行のタイムアウト（単位：ミリ秒）を指定します。 | 任意 |
| PreparedStatementPoolSize | java.lang.Integer | 30×<J2EE サーバ内の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 Web アプリケーション数>で求められる数値を指定します。 | 必須 |
| logLevel | java.lang.String | DB Connector が出力するログトレースのレベルについて，任意のレベルを指定します。 | 任意 |

（凡例）　必須：必ず指定する　任意：必要に応じて設定する

注※　databaseName，serverName および portNumber のすべての値を指定するか，url の値を指定してください。

## (b)　<property>タグに指定するプロパティ

<property>タグで設定するプロパティについて次の表に示します。なお，ここで示していないプロパティについては，データベースセッションフェイルオーバ機能に関する設定は不要です。

表 6-49　<property>タグに指定するプロパティ

| <property-name>タグに指定する値 | <property-type>タグに指定する値 | <property-default-value>タグの値 | <property-value>タグに指定する内容または値 | 必須／任意 |
|---|---|---|---|---|
| MaxPoolSize | int | 10 | コネクションプールの最大値※1を指定します。 | 必須 |
| MinPoolSize | int | 10 | コネクションプールの最小値※1を指定します。 | 必須 |
| LogEnabled | boolean | true | 「true」を指定します。 | 必須 |

| <property-name>タグに指定する値 | <property-type>タグに指定する値 | <property-default-value>タグの値 | <property-value>タグに指定する内容または値 | 必須／任意 |
|---|---|---|---|---|
| User※2 | String | － | ユーザ名を指定します。 | 必須 |
| Password | String | － | パスワードを指定します。 | 必須 |
| ValidationType | int | 1 | 「1」を指定します。 | 必須 |
| RetryCount | int | 0 | コネクション取得リトライ回数を指定します。<br>データベースの設定やネットワーク環境に合わせて，障害発生時にデータベースの回復が有効になるような値を指定してください。 | 任意 |
| RetryInterval | int | 10 | コネクション取得リトライ間隔を指定します。<br>データベースの設定やネットワーク環境に合わせて，障害発生時にデータベースの回復が有効になるような値を指定してください。 | 任意 |
| RequestQueueEnable | boolean | true | 「true」を指定します。 | 必須※3 |
| RequestQueueTimeout | int | 30 | コネクション枯渇時のコネクション取得待ち行列のとどまることのできる最大値を指定します。※4 | 必須※3 |
| WatchEnabled | boolean | true | コネクションプール監視を有効にするかどうかを指定します。 | 任意 |
| WatchInterval | int | 30 | コネクションプール監視間隔を指定します。 | 任意 |
| WatchThreshold | int | 80 | コネクションプール使用状態を監視するしきい値を指定します。 | 任意 |
| WatchWriteFileEnabled | boolean | true | 「true」を指定します。 | 任意 |

（凡例） 必須：必ず指定する　任意：必要に応じて設定する　－：なし

注※1

コネクションプールの値は，次の式で算出します。なお，コネクションプールの最大値と最小値は同じ値にしてください。

**Web アプリケーション単位，または URL 単位の同時実行スレッド数を設定した場合**

J2EE サーバ内の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 Web アプリケーションの同時実行数の和＋ 2

J2EE サーバ単位の同時実行スレッド数を設定した場合

J2EE サーバの同時実行スレッド数＋ 2

コネクションプールの最大値を超えて J2EE サーバがリクエストを受信すると，そのリクエストはコネクション枯渇時のコネクション取得待ち行列で待ち状態となります。

注※2

DB Connector に設定するユーザには，テーブルの作成者を登録してください。

注※3

Web アプリケーション単位の同時実行スレッド数の制御機能が無効の場合，設定は不要です。

注※4

次に示す範囲で値を指定してください。

Web サーバ連携を使用している場合

1 ＜ RequestQueueTimeout ＜リバースプロキシで設定する Web コンテナからのデータ受信のタイムアウト

リバースプロキシで設定する Web コンテナからのデータ受信のタイムアウトとは，ProxyPass ディレクティブの timeout キーまたは Timeout ディレクティブに指定した値です。

Web サーバを経由しないで J2EE サーバに直接リクエストを送受信する場合

1 ＜ RequestQueueTimeout

## 6.7　データベースセッションフェイルオーバ機能に関する設定の変更

この節では，データベースセッションフェイルオーバ機能に関する設定の変更について説明します。データベースセッションフェイルオーバ機能では，データベースのテーブルにアプリケーション情報やグローバルセッション情報などの設定情報を格納します。Web アプリケーション開始時のネゴシエーション処理で設定に誤りがないことを確認するため，一度開始した Web アプリケーションの設定を変更する場合は，データベースに保存した Web アプリケーションの設定情報の初期化が必要となります。ネゴシエーション処理については，「6.4.1　アプリケーション開始時の処理」を参照してください。

データベースセッションフェイルオーバ機能に関する設定の変更の流れを次の図に示します。

図 6-13　データベースセッションフェイルオーバ機能に関する設定の変更の流れ


注※1　Webアプリケーションの設定を変更する場合に必要です。

注※2　J2EEサーバの設定を変更する場合に必要です。

注※3　WebアプリケーションのHTTPセッションの属性情報の最大サイズを変更した場合に必要です。

データベースセッションフェイルオーバ機能に関する設定を変更する場合の注意事項を次に示します。

- **設定変更時に停止する範囲に関する注意事項**

  次の設定を変更する場合は，冗長化したほかの J2EE サーバまたはアプリケーションをすべて停止してください。

  - 完全性保障モードの設定
  - データベース上のグローバルセッション情報数の設定

- **HTTP セッションの属性情報の最大サイズの変更に関する注意事項**

  HTTP セッションの属性情報の最大サイズを小さく変更した場合，変更前に作成したグローバルセッション情報を引き継いだ時に，変更後の最大サイズを超えることがあります。最大サイズを超えた場合，属性情報のシリアライズ時に KDJE34320-E のメッセージが出力され，グローバルセッション情報はデータベースに保存されません。このため，HTTP セッションの属性情報の最大サイズを小さく変更する場合は，HTTP セッションを破棄してください。HTTP セッションの破棄については，「6.7.3 グローバルセッション情報の削除（HTTP セッションの破棄）」を参照してください。

この節では，J2EE サーバおよびアプリケーションの設定変更，およびデータベーステーブルの初期化について説明します。

> **参考**
>
> アプリケーションの停止および開始には，サーバ管理コマンドまたは運用管理ポータルを使用します。アプリケーションの開始については，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「cjstartapp（J2EE アプリケーションの開始）」を参照してください。アプリケーションの停止については，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「cjstopapp（J2EE アプリケーションの停止）」を参照してください。運用管理ポータルの操作については，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」の「12.3　J2EE アプリケーション管理」を参照してください。

## 6.7.1　J2EE サーバおよびアプリケーションの設定変更

ここでは，J2EE サーバの設定，および Web アプリケーションの設定変更の手順について説明します。設定を変更した場合，データベースに保存した情報の初期化が必要です。データベースに保存した情報の初期化については，「6.7.2　データベーステーブルの初期化」を参照してください。

### (1)　アプリケーションの停止および設定変更

アプリケーションの設定を変更するには，J2EE アプリケーションを停止して，Web アプリケーションの設定を変更します。

一つの J2EE サーバの Web アプリケーションの設定変更終了後に，冗長化した別の J2EE サーバの Web アプリケーションの設定を変更します。冗長化した J2EE サーバに対して，一つずつ同じ Web アプリケーションの設定変更をしていくことで，システム全体を停止することなく全体の設定を変更できます。

設定項目については，「6.5　cosminexus.xml での定義」，Web アプリケーションの設定変更の詳細については「6.6.2　Web アプリケーションの設定」を参照してください。

### (2)　J2EE サーバの停止および設定変更

J2EE サーバの設定を変更するには，次の手順を実施してください。

1. J2EE アプリケーションを停止します。
   J2EE サーバ内のすべての J2EE アプリケーションを停止します。

2. J2EE サーバを停止します。
   J2EE サーバを停止します。

3. 簡易構築定義ファイルで J2EE サーバの設定を変更します。
   簡易構築定義ファイルで設定を変更します。J2EE サーバでの設定項目については「6.6.1　J2EE サーバの設定」を参照してください。

4. 冗長化した別の J2EE サーバの設定を変更します。
   冗長化した別の J2EE サーバに対して順番に手順 1.～3.を実施し，冗長化した J2EE サーバすべてに同じ設定変更をします。

## 6.7.2　データベーステーブルの初期化

Web アプリケーションで使用する情報，または Web アプリケーションに関する情報を変更した場合，データベース上に保存された Web アプリケーションの設定情報を初期化する必要があります。ここでは，データベース上に保存された設定情報の初期化の手順について説明します。また，データベース上のグローバルセッション情報数の変更，およびデータベース上の HTTP セッションの属性情報の最大サイズの変更の手順について説明します。

### (1)　データベースに保存された設定情報の初期化

一度開始した Web アプリケーションの設定を変更した場合は，データベースに保存された設定情報を初期化する必要があります。

データベースに保存された設定情報を初期化する手順について説明します。

1. テンプレートファイルを任意の場所にコピーします。
   データベースに保存された設定情報の初期化用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレートファイルの格納場所を，使用するデータベースごとに次に示します。

   - HiRDB を使用する場合のテンプレートファイルの格納場所
     Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥hirdb_reset_apptbl.sql
     UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_reset_apptbl.sql
   - Oracle を使用する場合のテンプレートファイルの格納場所
     Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥oracle_reset_apptbl.sql
     UNIX の場合：/opt/Cosminexus/CC/sfo/sql/oracle_reset_apptbl.sql

2. テンプレートファイルを編集します。

Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，データベースに保存された設定情報の初期化用 SQL ファイルを作成します。テンプレートファイル内の変更個所と変更内容について次の表に示します。

表 6-50　テンプレートファイル内の変更個所と変更内容

| 変更個所 | 変更対象 | 変更内容 |
|---|---|---|
| 1 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |

3. 作成したデータベースに保存された設定情報の初期化用 SQL ファイルを実行します。

SQL ファイルの実行には，HiRDB を使用する場合は SQL Executer，Oracle を使用する場合は SQL*Plus などを使用してください。

## (2)　データベース上のグローバルセッション情報数の変更

データベース上のグローバルセッション情報数は，HTTPSession オブジェクト数の上限値に合わせて変更します。ただし，アプリケーション開始時に実施されるネゴシエーション処理に失敗した場合に，Web アプリケーションの開始処理を続行する設定にしているときは，データベース上のグローバルセッション情報数と HTTPSession オブジェクト数の上限値の設定は異なっていてもかまいません。

データベース上のグローバルセッション情報数を変更する手順について説明します。なお，データベース上のグローバルセッション情報数を変更すると，データベース上のセッション情報はすべて削除されます。

1. J2EE アプリケーション，および J2EE サーバを停止します。

J2EE サーバ内のすべての J2EE アプリケーション，および冗長化したすべての J2EE サーバを停止します。

2. テンプレートファイルを任意の場所にコピーします。

データベース上のグローバルセッション情報数の変更用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレートファイルの格納場所を，使用するデータベースごとに次に示します。

- HiRDB を使用する場合のテンプレートファイルの格納場所

  Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥hirdb_change_session_num.sql

  UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_change_session_num.sql

- Oracle を使用する場合のテンプレートファイルの格納場所

  Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥oracle_change_session_num.sql

  UNIX の場合：/opt/Cosminexus/CC/sfo/sql/oracle_change_session_num.sql

3. テンプレートファイルを編集します。

Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，データベース上のグローバルセッション情報数の変更用 SQL ファイルを作成します。テンプレートファイル内の変更個所と変更内容について次の表に示します。

表 6-51　テンプレートファイル内の変更個所と変更内容

| 変更個所 | | 変更対象 | 変更内容 |
|---|---|---|---|
| HiRDB | Oracle | | |
| • 1 行目<br>• 2 行目<br>• 3 行目<br>• 6 行目<br>• 7 行目 | • 1 行目<br>• 2 行目<br>• 3 行目<br>• 6 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |
| • 4 行目<br>• 7 行目 | • 4 行目<br>• 6 行目 | <HTTP_SESSION_NO> | データベース上のグローバルセッション情報数を変更してください。 |

4. 作成したデータベース上のグローバルセッション情報数の変更用 SQL ファイルを実行します。

SQL ファイルの実行には，HiRDB を使用する場合は SQL Executer，Oracle を使用する場合は SQL*Plus などを使用してください。

## (3)　データベース上の HTTP セッションの属性情報の最大サイズの変更（HiRDB の場合だけ）

セッション情報格納テーブルを作成したあとに，Web アプリケーションに設定したグローバルセッション情報に含めることができる HTTP セッションの属性情報の最大サイズを変更した場合，データベース上の HTTP セッションの属性情報の最大サイズを変更する必要があります。なお，データベース上の HTTP セッションの属性情報の最大サイズは，Web アプリケーションに設定したグローバルセッション情報に含めることができる HTTP セッションの属性情報の最大サイズよりも大きい値にしてください。Web アプリケーションに設定したグローバルセッション情報に含めることができる HTTP セッションの属性情報の最大サイズの設定については，「6.5　cosminexus.xml での定義」を参照してください。

データベース上の HTTP セッションの属性情報の最大サイズを変更する手順について説明します。この手順は，HiRDB を使用する場合だけ必要です。

1. J2EE アプリケーション，および J2EE サーバを停止します。

J2EE サーバ内のすべての J2EE アプリケーション，および冗長化したすべての J2EE サーバを停止します。

2. テンプレートファイルを任意の場所にコピーします。

データベース上の HTTP セッションの属性情報の最大サイズの変更用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレート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¥sfo¥sql¥hirdb_change_attributes_size.sql

- UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_change_attributes_size.sql

3. テンプレートファイルを編集します。

Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，データベース上の HTTP セッションの属性情報の最大サイズの変更用 SQL ファイルを作成します。テンプレートファイル内の変更個所と変更内容について次の表に示します。

表 6-52　テンプレートファイル内の変更個所と変更内容

| 変更個所 | 変更対象 | 変更内容 |
|---|---|---|
| 1 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |
| 2 行目 | <ATTRIBUTE_DATA_SIZE_MAX> | HTTP セッションの属性情報を格納するカラムのサイズ（単位：バイト）を変更してください。 |

4. 作成したデータベース上のグローバルセッション情報数の変更用 SQL ファイルを実行します。

SQL ファイルの実行には，SQL Executer を使用してください。

## 6.7.3　グローバルセッション情報の削除（HTTP セッションの破棄）

システムの運用中，アプリケーションのバージョンアップをする場合などに，システム内に存在する HTTP セッションの破棄が必要となる場合があります。

データベースセッションフェイルオーバ機能では，データベースにグローバルセッション情報を格納するため，J2EE アプリケーションや J2EE サーバの停止では HTTP セッションが破棄できません。グローバルセッション情報をデータベースから削除することで，HTTP セッションを破棄します。

グローバルセッション情報を削除するには，次の手順を実施してください。

1. J2EE アプリケーションを停止します。

J2EE サーバ内のすべての J2EE アプリケーションを停止します。

2. データベース上のグローバルセッション情報を削除します。

データベース上のグローバルセッション情報数を変更する手順でグローバルセッション情報を削除します。この時，グローバルセッション情報数を変更しないで，変更手順だけを実施します。変更手順については，「6.7.2(2)　データベース上のグローバルセッション情報数の変更」を参照してください。

3. J2EE アプリケーションを開始します。

## 6.8　データベースのテーブルの削除

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アプリケーションの設定を変更する際に，データベースのテーブルの削除が必要な場合があります。ここでは，データベースのテーブルの削除について説明します。

削除するテーブルと，削除手順の参照先について次の表に示します。

### 表 6-53　削除するテーブルと，削除手順の参照先

| テーブル名 | データベース上の物理名称 | 削除手順の参照先 |
|---|---|---|
| アプリケーション情報テーブル | SFO_<APPLICATION_ID>_APP_INFO | 6.8.1 |
| セッション情報格納テーブル | SFO_<APPLICATION_ID>_SESSIONS | 6.8.2 |
| 空きレコード情報テーブル | SFO_<APPLICATION_ID>_REC_INFO | |

データベースセッションフェイルオーバ機能で使用するデータベースのテーブル削除用のテンプレートファイルは次の場所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¥sfo¥sql¥
```

UNIX の場合：

```
/opt/Cosminexus/CC/sfo/sql/
```

テーブル削除用のテンプレートファイルは使用するデータベースごとに 2 種類ずつあります。使用するデータベース，ファイル，および削除するテーブルの種類の対応を次の表に示します。

### 表 6-54　テーブル削除用テンプレートファイルと削除するテーブル

| 使用するデータベース | テンプレートファイル | 削除するテーブルの種類 | | |
|---|---|---|---|---|
| | | アプリケーション情報テーブル | セッション情報格納テーブル | 空きレコード情報テーブル |
| HiRDB | hirdb_delete_apptbl.sql | ○ | − | − |
| | hirdb_delete_sessiontbl.sql | − | ○ | ○ |
| Oracle | oracle_delete_apptbl.sql | ○ | − | − |
| | oracle_delete_sessiontbl.sql | − | ○ | ○ |

（凡例）○：削除できる　−：削除できない

以降で，使用するデータベースごとにテンプレートファイルの詳細について示します。

### 6.8.1　アプリケーション情報テーブルの削除

アプリケーション情報テーブルは，Web アプリケーションに設定したデータベースセッションフェイルオーバ機能に関する設定を格納するテーブルです。

アプリケーション情報テーブルの削除手順を次に示します。

1. テンプレートファイルを任意の場所にコピーします。

   テーブル削除用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレートファイルの格納場所を，使用するデータベースごとに次に示します。

   - HiRDB を使用する場合のテンプレートファイルの格納場所

     Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥hirdb_delete_apptbl.sql

     UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_delete_apptbl.sql

   - Oracle を使用する場合のテンプレートファイルの格納場所

     Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥oracle_delete_apptbl.sql

     UNIX の場合：/opt/Cosminexus/CC/sfo/sql/oracle_delete_apptbl.sql

2. テンプレートファイルを編集します。

   Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，テーブル削除用 SQL ファイルを作成します。テンプレートファイル内の変更個所と変更内容について次の表に示します。

   表 6-55　テンプレートファイル内の変更個所と変更内容

| 変更個所 | | 変更対象 | 変更内容 |
|---|---|---|---|
| HiRDB | Oracle | | |
| 1 行目 | 1 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |
| なし | 1 行目 | <SCHEMA_NAME> | データベース接続ユーザのスキーマ名に変更してください。 |

3. 作成したテーブル削除用 SQL ファイルを実行します。

   SQL ファイルの実行には，HiRDB を使用する場合は SQL Executer，Oracle を使用する場合は SQL*Plus などを使用してください。

### 6.8.2　セッション情報格納テーブルおよび空きレコード情報テーブルの削除

セッション情報格納テーブルは，グローバルセッション情報を格納するテーブルです。空きレコード情報テーブルは，セッション情報格納テーブルの未使用レコードを管理するテーブルです。

セッション情報格納テーブルおよび空きレコード情報テーブルの削除手順について説明します。

1. テンプレートファイルを任意の場所にコピーします。

テーブル削除用の SQL ファイルとして，テンプレートファイルが用意されています。テンプレートファイルの格納場所を，使用するデータベースごとに次に示します。

- HiRDB を使用する場合のテンプレートファイルの格納場所

  Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥hirdb_delete_sessiontbl.sql

  UNIX の場合：/opt/Cosminexus/CC/sfo/sql/hirdb_delete_sessiontbl.sql

- Oracle を使用する場合のテンプレートファイルの格納場所

  Windows の場合：<Application Server のインストールディレクトリ>¥CC¥sfo¥sql¥oracle_delete_sessiontbl.sql

  UNIX の場合：/opt/Cosminexus/CC/sfo/sql/oracle_delete_sessiontbl.sql

2. テンプレートファイルを編集します。

Web アプリケーションの設定情報に合わせて，テンプレートファイルを編集して，テーブル削除用 SQL ファイルを作成します。テンプレートファイル内の変更個所と変更内容について次の表に示します。

表 6-56　テンプレートファイル内の変更個所と変更内容

| 変更個所 | | 変更対象 | 変更内容 |
|---|---|---|---|
| HiRDB | Oracle | | |
| • 1 行目<br>• 2 行目<br>• 3 行目<br>• 4 行目 | • 1 行目<br>• 2 行目<br>• 3 行目<br>• 4 行目 | <APPLICATION_ID> | 使用するアプリケーションのアプリケーション識別子に変更してください。 |
| なし | • 1 行目<br>• 2 行目<br>• 3 行目<br>• 4 行目 | <SCHEMA_NAME> | データベース接続ユーザのスキーマ名に変更してください。 |

3. 作成したテーブル削除用 SQL ファイルを実行します。

SQL ファイルの実行には，HiRDB を使用する場合は SQL Executer，Oracle を使用する場合は SQL*Plus などを使用してください。

## 6.9　データベースセッションフェイルオーバ機能使用時の注意事項

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際の注意事項について説明します。

- データベースセッションフェイルオーバ機能で使用するテーブルの内容を操作した場合，システムの情報を正しく保つことができないため，正常な運用を続けることができなくなります。

  データベース上のデータベースセッションフェイルオーバ機能で使用するテーブルの変更を伴う操作として実行できるのは，次の操作だけです。なお，次の操作を実施する場合も，参照先の手順に従って操作してください。

  - データベースセッションフェイルオーバ機能の設定変更（6.7 参照）
  - テーブルの初期化（6.7.2 参照）
  - HTTP セッションの破棄（6.7.3 参照）

  これらの操作以外については，実行しないでください。
- HTTP セッションの属性情報のサイズが最大値を超えた場合，HTTP セッションの情報はデータベースに冗長化されません。

# 7 明示管理ヒープ機能を使用した FullGC の抑止

アプリケーションサーバでは，Java アプリケーション実行時に，Java オブジェクトの配置先として Java ヒープとは別のメモリ空間を使用できます。この機能を明示管理ヒープ機能といいます。明示管理ヒープ機能を効果的に使用することで，FullGC の発生を抑止できます。

この章では，明示管理ヒープ機能を使用した FullGC の抑止について説明します。

なお，G1GC を使用した場合に，明示管理ヒープ機能を使用できません。G1GC 使用時に-XX:+HitachiUseExplicitMemory を指定した場合，次のメッセージを標準出力に出力して JavaVM を終了します。

Using -XX:+UseG1GC and -XX:+HitachiUseExplicitMemory at the same time is not supported.

また，アプリケーションサーバでは，明示管理ヒープ機能がデフォルトで有効になります。そのため，G1GC を使用する場合は，-XX:-HitachiUseExplicitMemory を指定してください。

## 7.1　この章の構成

明示管理ヒープ機能は，Java オブジェクトの配置先として Java ヒープ外の領域（Explicit ヒープ）を使用することで，FullGC の発生を抑止する機能です。

この章の構成を次の表に示します。

表 7-1　この章の構成（明示管理ヒープ機能を使用した FullGC の抑止）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | 明示管理ヒープ機能の概要 | 7.2 |
| | 明示管理ヒープ機能で使用するメモリ空間の概要 | 7.3 |
| | J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクト | 7.4 |
| | アプリケーションで任意に Explicit ヒープに配置できるオブジェクト | 7.5 |
| | Explicit メモリブロックのライフサイクルと実行される処理 | 7.6 |
| | 自動解放機能が有効な場合の Explicit メモリブロックの解放 | 7.7 |
| | 自動解放機能が無効な場合の Explicit メモリブロックの解放 | 7.8 |
| | javagc コマンドによる Explicit メモリブロックの解放 | 7.9 |
| | Explicit メモリブロックの自動解放処理に掛かる時間の短縮 | 7.10 |
| | HTTP セッションで利用する Explicit ヒープのメモリ使用量の削減 | 7.11 |
| 実装 | 明示管理ヒープ機能 API を使った Java プログラムの実装 | 7.12 |
| 設定 | 実行環境での設定 | 7.13 |
| 注意事項 | 明示管理ヒープ機能使用時の注意事項 | 7.14 |

注 「運用」について，この機能固有の説明はありません。

なお，GC の仕組みについては，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. JavaVM のメモリチューニング」を参照してください。

## 7.2　明示管理ヒープ機能の概要

この節では，明示管理ヒープ機能の概要について説明します。

### 7.2.1　明示管理ヒープ機能を利用する目的

**明示管理ヒープ機能**は，FullGC の発生を抑止する機能です。この機能を使用することで，システムが停止する回数を低減し，安定したスループットを実現します。

システムで扱う論理アドレス空間の増加やシステム規模の拡大などによって，アプリケーションサーバで扱う Java ヒープのサイズは増加しています。Java ヒープのサイズの増加に伴って問題になるのは，GC の実行に掛かる時間の増加です。GC が実行されている間，システムは停止します。特に，FullGC の実行時間は，使用済みの Java ヒープのサイズに応じて増加します。使用できる Java ヒープのサイズの増加に従って，FullGC に掛かる時間も増えるおそれがあります。

**参考**

GC のアルゴリズムとシステム停止時間の関係

JavaVM では，CopyGC のアルゴリズムとして Copy アルゴリズム，FullGC のアルゴリズムとして Mark Sweep Compact アルゴリズムを採用しています。これらのアルゴリズムは，Stop The World 型のアルゴリズムです。Stop The World 型では，GC の実行に掛かる時間は，その JavaVM を利用したシステムが停止する時間と等しくなります。

### 7.2.2　明示管理ヒープ機能の利用による FullGC の抑止の仕組み

明示管理ヒープ機能では，Java オブジェクトの配置先として，**Explicit ヒープ**という独自の領域を使用します。Explicit ヒープは，Java ヒープ外にある，GC の対象にならない領域です。明示管理ヒープ機能を使用していない場合に Java ヒープに配置していた Java オブジェクトを Explicit ヒープに配置することで，FullGC が発生することを抑止できます。

ここでは，明示管理ヒープ機能を利用して FullGC を抑止する仕組みについて説明します。また，明示管理ヒープ機能の位置づけについてもあわせて説明します。

#### (1)　FullGC 発生を抑止する仕組み

Java アプリケーション実行中に Eden 領域に空き領域がなくなると，GC が発生します。このとき，次の式が成り立つ場合は，JavaVM によって FullGC が実行されます。

```
New領域で使用されているメモリのサイズ＞Tenured領域の空き領域のサイズ
```

注

Eden 領域に空き領域がなくなっているため，New 領域で使用されているメモリのサイズは New 領域の最大サイズにほぼ等しくなります。

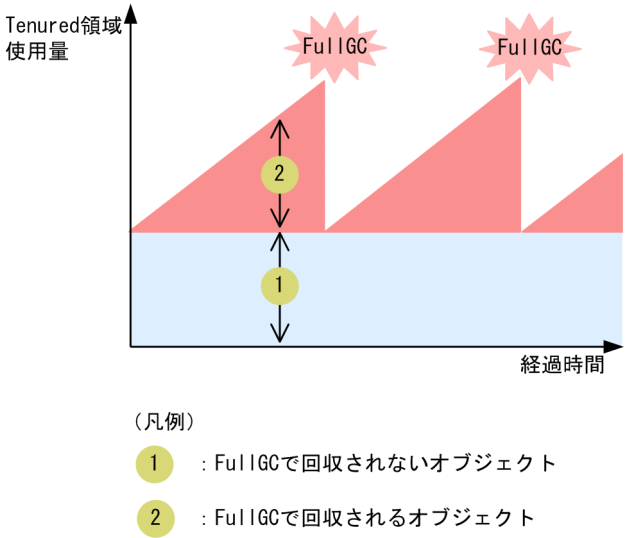
式が示すとおり，Tenured 領域の空き領域のサイズが小さくなると FullGC が発生します。Tenured 領域の空き領域は，CopyGC が発生したときに Survivor 領域から移動（昇格）する Java オブジェクトによって使用されます。つまり，昇格する Java オブジェクトを削減できれば，FullGC の発生を抑えられます。なお，何回かの CopyGC の実行で削除されないで，昇格の対象になるオブジェクトを**長寿命オブジェクト**といいます。

長寿命オブジェクトには大きく分けて 2 種類あります。一つは，FullGC で回収されないオブジェクトです。例えばアプリケーションの実行中は常に生存し続けるような，本来 Tenured 領域に格納される必要があるオブジェクトが当てはまります。このようなオブジェクトは増加し続けることはないため，FullGC が発生する本質的な原因ではありません。このような長寿命オブジェクトの影響を排除したい場合は，Tenured 領域のサイズを増加することで対処できます。

二つ目は，FullGC で回収されるオブジェクトです。FullGC で回収される長寿命オブジェクトとは，Tenured 領域に昇格する程度に長寿命であるが，一定期間で不要となるオブジェクトを指します。このような長寿命オブジェクトは FullGC の発生までは増加し続けるため，FullGC 発生の原因となります。

FullGC で回収されるオブジェクト，および FullGC で回収されないオブジェクトについて，次の図で説明します。

図 7-1　FullGC で回収されるオブジェクト，および FullGC で回収されないオブジェクト

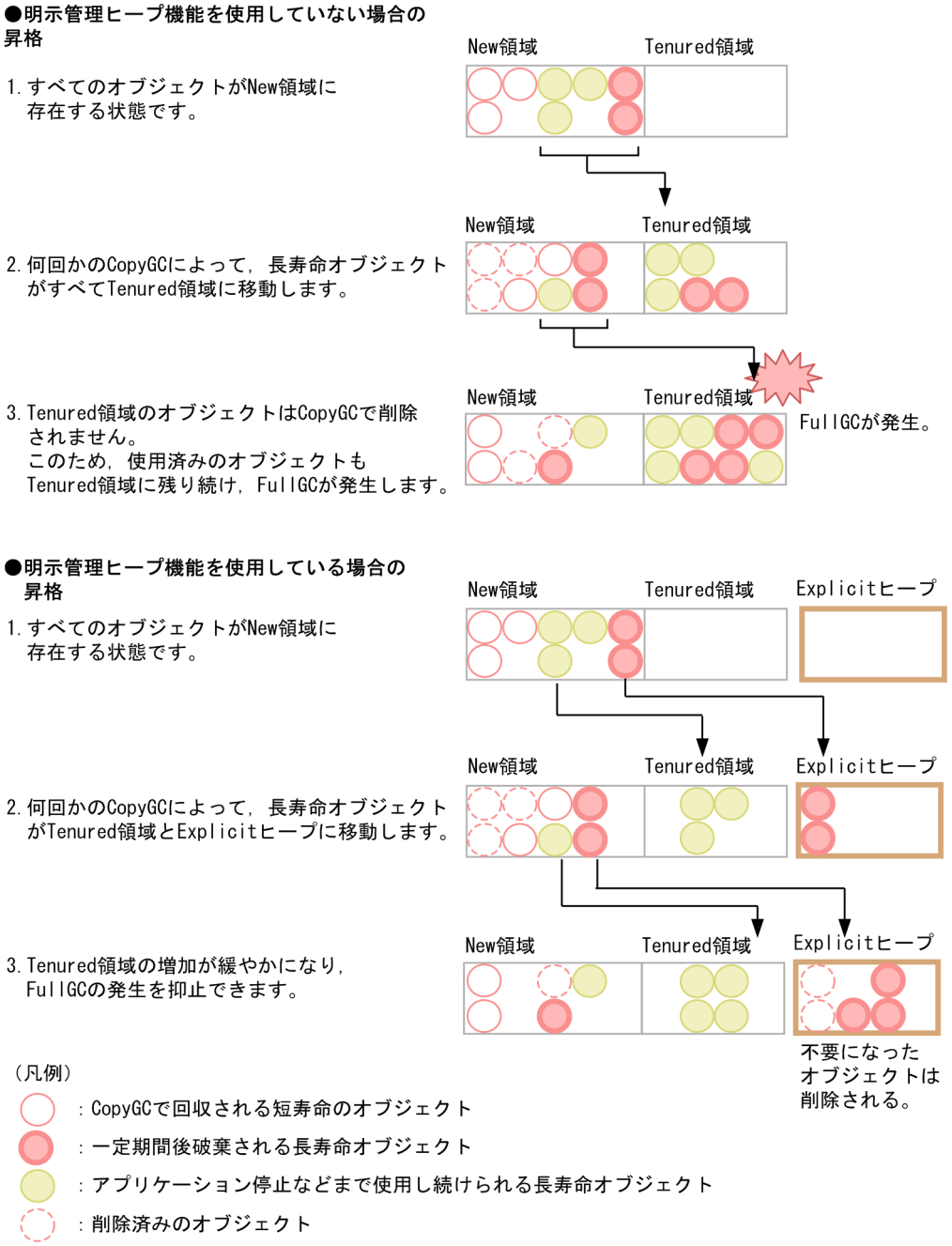



一定期間で不要となるオブジェクトの増加を防ぐ対策として，Tenured 領域のサイズを増加しただけでは，効果がありません。Tenured 領域のサイズを 2 倍にしても，FullGC の発生間隔が 2 倍になるだけで，期待するほどの効果は得られません。

つまり，FullGC の発生を抑止するためには，一定期間で不要となるオブジェクトの Tenured 領域への昇格を減らすことがポイントとなります。

アプリケーションサーバでは，一部の Java オブジェクトについて，CopyGC 発生時の昇格先が Explicit ヒープになるように設定されています。明示管理ヒープ機能を使用していない場合の昇格と明示管理ヒープ機能を使用している場合の昇格の違いを次の図に示します。

図 7-2　明示管理ヒープ機能を使用していない場合の昇格と明示管理ヒープ機能を使用している場合の昇格の違い




図の 1.のタイミングでは，どちらの場合も同じ状態です。2.のオブジェクトが昇格するタイミングで，明示管理ヒープ機能を使用していない場合の昇格では，すべての長寿命オブジェクトが Tenured 領域に移動します。一方，明示管理ヒープ機能を使用している場合，長寿命オブジェクトのうち，一定期間後に破棄されることがわかっているオブジェクトについては Explicit ヒープに移動します。これによって，Tenured 領域に移動するのは，破棄される予定がない長寿命オブジェクトに限定され，Tenured 領域の使用済みサ

イズの増加が緩やかになります。なお，3.で示すとおり，明示管理ヒープ機能を使用している場合の Explicit ヒープのオブジェクトは，不要になったタイミングで削除されます。

対象となる Java オブジェクトについては，「7.4　J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクト」を参照してください。また，GC のアルゴリズムについては，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.　JavaVM のメモリチューニング」を参照してください。

なお，ユーザが開発するアプリケーションで明示管理ヒープ機能を使用する場合は，一定期間後に破棄される長寿命オブジェクトを，直接 Explicit ヒープに生成します。これによって，Tenured 領域のメモリサイズ増加を防ぎます。Explicit ヒープに生成できる Java オブジェクトについては，「7.5　アプリケーションで任意に Explicit ヒープに配置できるオブジェクト」を参照してください。

## (2)　明示管理ヒープ機能の位置づけ

明示管理ヒープ機能は，JavaVM の機能です。明示管理ヒープ機能を利用する方法には，次の 2 種類があります。

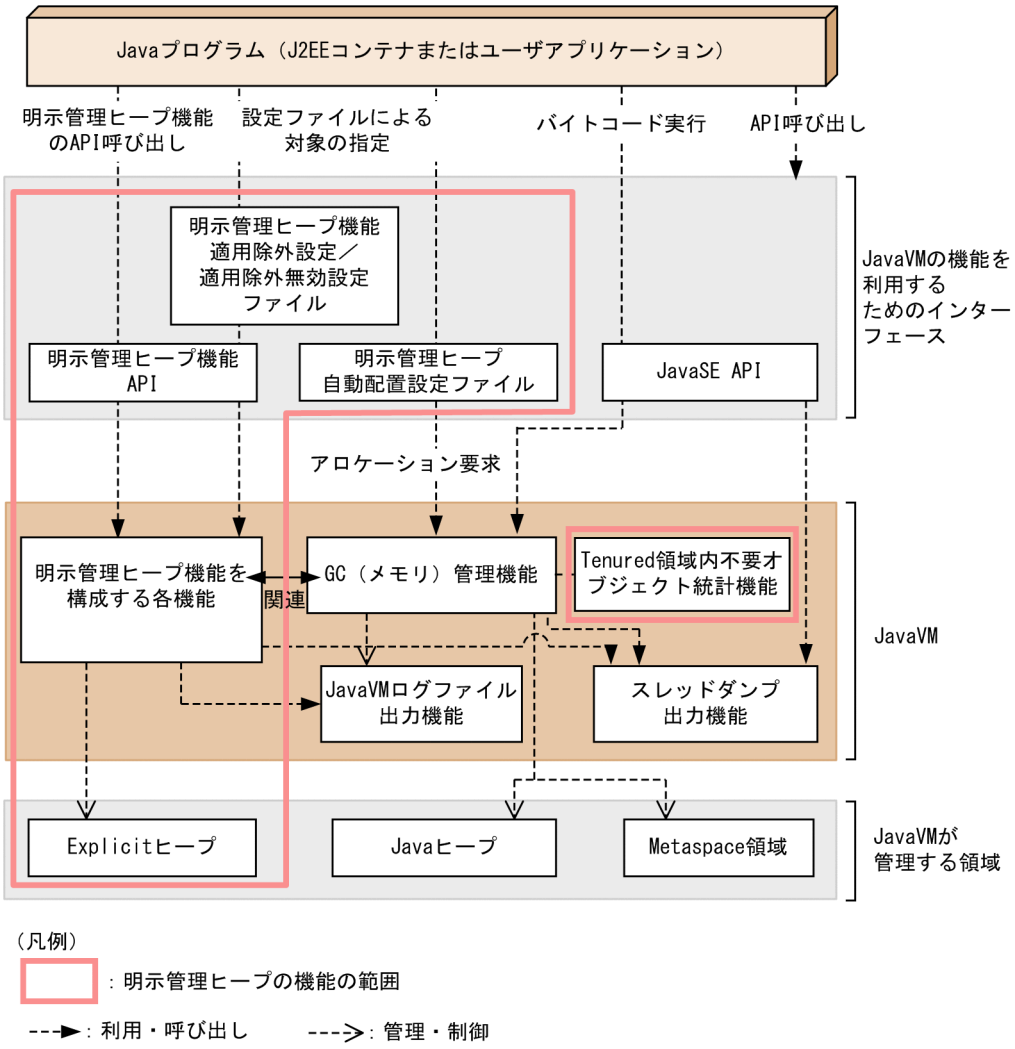
- 明示管理ヒープ機能の設定ファイルを使う方法

  明示管理ヒープ機能の設定ファイルには，次のものがあります。これらを使用して，明示管理ヒープ機能を利用する対象オブジェクトを設定できます。

  - 明示管理ヒープ機能適用除外設定/適用除外無効設定ファイル
  - 自動配置設定ファイル
- 明示管理ヒープ機能 API を使う方法

明示管理ヒープ機能の位置づけを次の図に示します。なお，図中の JavaVM ログファイル出力機能は，JavaVM ログファイル出力機能のことです。

図 7-3　明示管理ヒープ機能の位置づけ




明示管理ヒープ機能の範囲である，明示管理ヒープ機能 API，自動配置設定ファイル，明示管理ヒープ機能適用除外設定／適用除外無効設定ファイル，明示管理ヒープ機能を構成する各機能，Tenured 領域内不要オブジェクト統計機能，および Explicit ヒープについて説明します。

### 明示管理ヒープ機能 API

Java プログラムから明示管理ヒープ機能を使用する場合，明示管理ヒープ機能 API を使用します。この API では，Explicit ヒープに関連する操作が実行できます。また，Explicit ヒープの使用状況を稼働情報として収集することもできます。

### 自動配置設定ファイル

Java プログラムを変更しないで明示管理ヒープ機能を使用する場合，自動配置設定ファイルを使用します。このファイルに明示管理ヒープに配置させたいオブジェクトを指定します。

### 明示管理ヒープ機能適用除外設定／適用除外無効設定ファイル

自動配置機能で明示管理ヒープに配置したオブジェクトから参照されているオブジェクトは，GC 発生時に，参照関係に基づいて明示管理ヒープへ自動で移動します。この参照関係に基づく移動の対象となるオブジェクトを，クラス単位に明示管理ヒープ機能の適用対象から除外する場合，明示管理ヒープ機能適用除外設定ファイルと，明示管理ヒープ機能適用除外無効設定ファイルを使用します。

明示管理ヒープ機能の適用対象から除外する場合は，明示管理ヒープ機能適用除外設定ファイルを使用します。このファイルに明示管理ヒープへ移動しないオブジェクトのクラスを指定します。

また，明示管理ヒープ機能適用除外設定ファイルで同一パッケージ内のすべてのクラスを明示管理ヒープ機能の適用対象から除外している場合などに，一部のクラスを明示管理ヒープ機能の適用対象とするときは，明示管理ヒープ機能適用除外無効設定ファイルを使用します。このファイルに明示管理ヒープ機能適用除外の設定を無効にしたいクラスを指定します。

**明示管理ヒープ機能を構成する各機能**

明示管理ヒープ機能を構成する各機能は，JavaVM に含まれます。API で呼び出されます。次の処理を実行できます。

- Explicit ヒープおよびヒープ内のメモリブロックの管理および制御
- GC と連携したアロケーション処理の変更による Explicit ヒープへのオブジェクトの配置
  なお，アロケーション処理は new キーワードの延長で実行されます。
- Explicit ヒープメモリブロックへのオブジェクトの移動制御
- JavaVM ログファイルとスレッドダンプへの Explicit ヒープのイベントログおよび状態の出力

**Tenured 領域内不要オブジェクト統計機能**

Tenured 領域内でメモリ増加の原因となっている不要オブジェクトを調査します。Tenured 領域内不要オブジェクト統計機能については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「9.8　Tenured 領域内不要オブジェクト統計機能」を参照してください。

**Explicit ヒープ**

GC の対象外にする Java オブジェクトの配置先になる領域です。明示管理ヒープ機能が管理します。Explicit ヒープは，複数のメモリブロック（Explicit メモリブロック）で構成されます。

### (3)　明示管理ヒープ機能を使用する場合に必要なメモリサイズ

明示管理ヒープ機能で管理する Explicit ヒープは，Java ヒープ外の領域です。Explicit ヒープを使用する場合，使用しない場合に比べて，メモリの使用量が増加します。

明示管理ヒープ機能を使用する場合は，必要なメモリサイズとして Explicit ヒープの最大サイズを見積もり，適切に設定する必要があります。明示管理ヒープ機能を利用する流れ，Explicit ヒープに格納するオブジェクト（Tenured 領域のメモリサイズ増加の要因になるオブジェクト），および Explicit ヒープのサイズの見積もりについては，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 Explicit ヒープのチューニング」を参照してください。

## 7.2.3　明示管理ヒープ機能を利用する場合の前提条件

ここでは，明示管理ヒープ機能を利用する場合の前提条件について説明します。明示管理ヒープ機能は，サーバまたはコマンドごとに，使用可否が異なります。

明示管理ヒープ機能のサポート有無を次の表に示します。デフォルトの設定については，「7.13.1　明示管理ヒープ機能を利用するための共通の設定（JavaVM オプションの設定）」を参照してください。

### 表 7-2　明示管理ヒープ機能のサポート有無

| サーバまたはコマンドの種類 | サポート有無 |
|---|---|
| J2EE サーバ | ○ |
| バッチサーバ | ○ |
| cjclstartap コマンド | ○ |

（凡例）

○：サポートされています。

## 7.3　明示管理ヒープ機能で使用するメモリ空間の概要

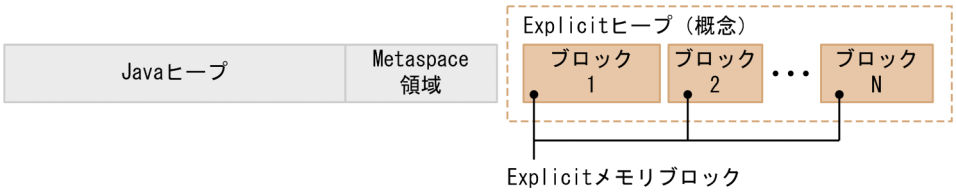
この節では，明示管理ヒープ機能で使用するメモリ空間である Explicit ヒープの構造について説明します。なお，JavaVM で使用するメモリ空間の構成については，マニュアル「アプリケーションサーバ システム設計ガイド」の「7.2.6　SerialGC 使用時の JavaVM で使用するメモリ空間の構成と JavaVM オプション」もあわせて参照してください。

Explicit ヒープは，GC の対象にならないメモリ空間です。複数のメモリブロックで構成されます。Explicit ヒープを構成するメモリブロックを**Explicit メモリブロック**といいます。Explicit ヒープは，Explicit メモリブロック全体を表す概念です。

初期化や解放などの操作は，Explicit メモリブロック単位で実行します。

Explicit ヒープの概念を次の図に示します。

### 図 7-4　Explicit ヒープの概念




Explicit ヒープの最大サイズは，JavaVM 起動オプションの-XX:HitachiExplicitHeapMaxSize オプションで設定します。-XX:HitachiExplicitHeapMaxSize オプションの詳細については，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「-XX:HitachiExplicitHeapMaxSize（Explicit メモリブロックの最大サイズ指定オプション)」を参照してください。生成（初期化）できる Explicit メモリブロックの数は，最大 1,048,575 個です。この数を超えて生成することはできません。

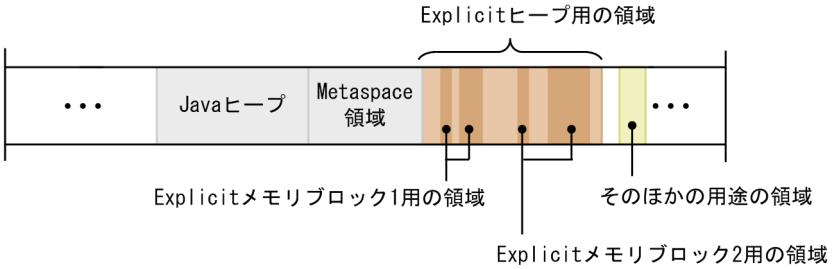
#### Explicit ヒープ領域の確保のタイミングについて

-XX:HitachiExplicitHeapMaxSize オプションで指定した Explicit ヒープの最大サイズの実メモリ領域が，JavaVM を起動したタイミングで確保されます。領域は，Java ヒープおよび Metaspace 領域からの連続領域として確保されます。

Java オブジェクトを Explicit メモリブロックに配置するために必要なメモリが不足している場合，JavaVM 起動時に確保した Explicit ヒープの領域から，Explicit メモリブロック向けのメモリ領域を確保します。このため，Explicit メモリブロック内のメモリ領域は複数の領域に分かれていることがあります。

仮想メモリ空間の利用イメージを次の図に示します。

図 7-5　仮想メモリ空間の利用イメージ




Explicit ヒープ用の領域は連続領域になりますが，一つの Explicit メモリブロックで使用する領域が，非連続になります。

## 7.4　J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクト

この節では，J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクトについて説明します。

J2EE サーバでは，FullGC の発生を抑止するために，次のオブジェクトを Explicit ヒープに配置します。

- HTTP セッションに関するオブジェクト

Explicit メモリブロック領域の確保や Explicit メモリブロックの解放予約は，Web コンテナが実行します。ここでは，オブジェクトに対して Web コンテナによって実行される処理について説明します。

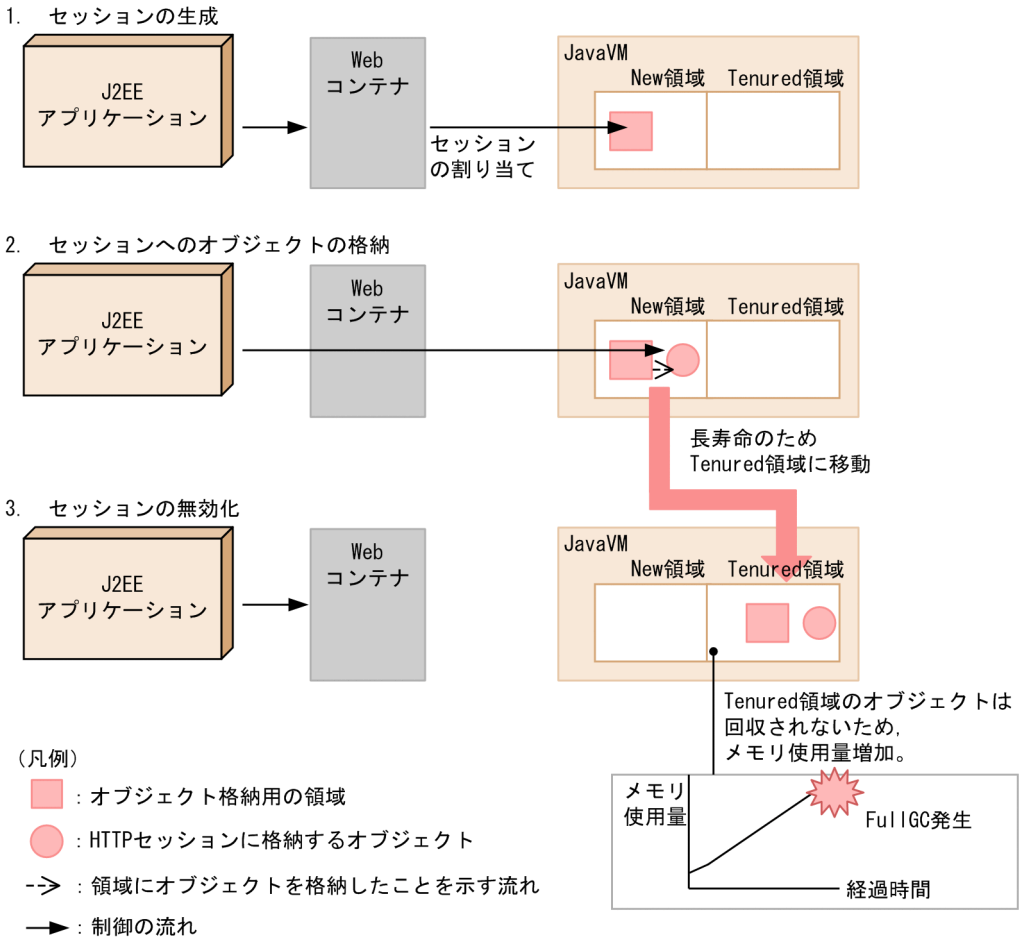
### 7.4.1　HTTP セッションに関するオブジェクト

HTTP セッションに格納するオブジェクトは，セッションが有効な間保持されるオブジェクトです。生存期間は，セッションの生成から破棄までの間です。

このオブジェクトは，明示管理ヒープ機能を使用していない場合，何回かの CopyGC が実行される間使用され続けることが多いオブジェクトです。このため，長寿命オブジェクトとして Tenured 領域に昇格しやすくなります。Tenured 領域に昇格したオブジェクトは CopyGC では回収されないため，このオブジェクトはセッションが破棄されたあとも回収されません。このため，Tenured 領域のメモリ使用量が増加していき，FullGC が発生します。

明示管理ヒープ機能を利用していない場合の例を次の図に示します。

図 7-6　明示管理ヒープ機能を利用していない場合の例

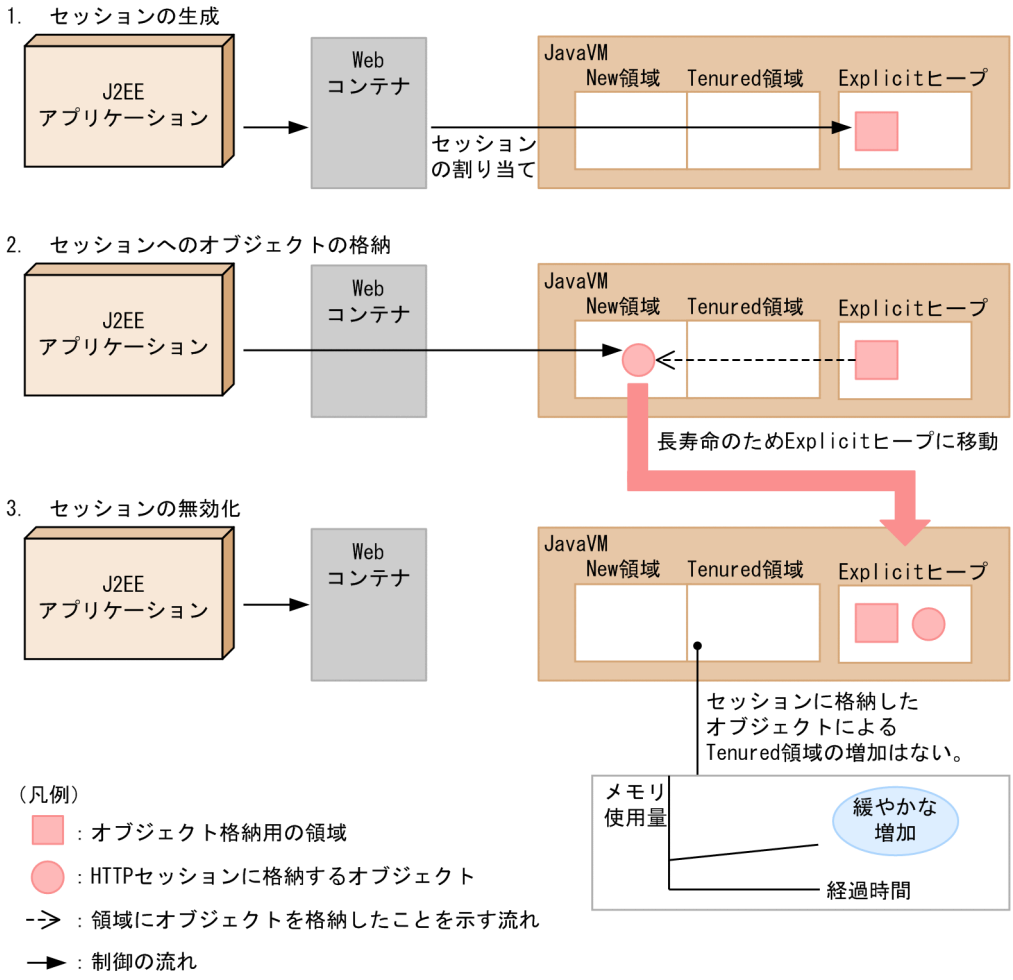



1.でセッションを生成すると，New 領域にオブジェクトを格納するための領域が確保されます。2.でセッションにオブジェクトが格納されます。何回かの CopyGC のあと，1.および 2.のオブジェクトは Tenured 領域に移動します。3.でセッションが無効化された場合も，Tenured 領域のオブジェクトは回収されないため，メモリ使用量が増加していきます。

これに対して，HTTP セッションに関するオブジェクトの昇格先を Tenured 領域から Explicit ヒープに変更することで，FullGC 発生を抑止できます。

明示管理ヒープ機能を利用した例を次の図に示します。

図 7-7　明示管理ヒープ機能を利用した例




明示管理ヒープ機能を使用すると，HTTP セッションに関するオブジェクトによる Tenured 領域の増加がありません。このため，FullGC 発生を抑止できます。なお，Explicit ヒープに配置した Java オブジェクトは，セッションが破棄されたあと，J2EE サーバによって明示的に解放されます。

HTTP セッションを配置する Explicit メモリブロック領域の確保および解放のタイミングについて説明します。

### HTTP セッションを作成したとき

新しく HTTP セッションを作成すると，Explicit ヒープ領域に Explicit メモリブロックが作成されます。Explicit メモリブロックは，1 セッションに一つ割り当てられます。また，HTTP セッションが，Explicit メモリブロック内に確保されます。

ただし，セッション作成直後のオブジェクトの配置先は Java ヒープです。何回かの CopyGC のあと，該当する Java オブジェクトが昇格するタイミングで，オブジェクトが Explicit ヒープに移動します。

### HTTP セッションにオブジェクトが格納されたとき（setAttribute メソッドを実行したとき）

javax.servlet.http.HttpSession.setAttribute メソッドで HTTP セッションに格納したオブジェクトは，セッションごとに割り当てられた Explicit メモリブロックに配置されます。

ただし，setAttribute メソッド実行時のオブジェクトの配置先は Java ヒープです。何回かの CopyGC のあと，該当する Java オブジェクトが昇格するタイミングで，オブジェクトが Explicit ヒープに移動

します。このとき，移動するオブジェクトから参照されているオブジェクトは，すべて Explicit ヒープに移動します。ただし，Explicit メモリブロックへのオブジェクト移動制御オプションの指定によって，Explicit ヒープに移動しなくなることがあります。

**HTTP セッションからオブジェクトが削除されたとき（removeAttribute メソッドを実行したとき）**

Explicit ヒープは，GC の対象外です。このため，javax.servlet.http.HttpSession.removeAttribute メソッドを実行して HTTP セッションからオブジェクトを削除しても，オブジェクトが使用していた領域は解放されません。

なお，setAttribute メソッドで属性を変更した場合も，変更前の属性は GC の対象にならないため，オブジェクトが使用していた領域は解放されません。

メモリの解放は，Explicit メモリブロック単位で実行されます。setAttribute メソッドを繰り返し頻繁に実行するような Web アプリケーションの場合，removeAttribute メソッドを実行しても Explicit メモリブロック内の領域を不要に消費してしまうおそれがありますので，注意してください。

**HTTP セッションが破棄されるとき**

HTTP セッション作成時に作成された Explicit メモリブロックは，HTTP セッションが破棄されるときに，Web コンテナによって解放が予約されます。

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。

なお，Explicit メモリブロックを解放したあとで，セッションに格納したオブジェクトへの参照が残っていた場合については，次の説明を参照してください。

- 「7.7　自動解放機能が有効な場合の Explicit メモリブロックの解放」
- 「7.8　自動解放機能が無効な場合の Explicit メモリブロックの解放」

Web アプリケーションで実行される操作または動作と JavaVM の動作の対応を次の表に示します。

### 表 7-3　Web アプリケーションで実行される操作（API）と JavaVM の動作の対応

| Web アプリケーションで実行される操作（API）または動作 | Web コンテナの動作 | JavaVM の動作 |
|---|---|---|
| • javax.servlet.http.HttpServletRequest.getSession()<br>• javax.servlet.http.HttpServletRequest.getSession(boolean) | セッションの生成 | Explicit メモリブロックの確保 |
| • javax.servlet.http.HttpSession.setAttribute(String, Object) | セッションへのオブジェクトの格納 | Explicit メモリブロックへのオブジェクトの配置 |
| • セッションのタイムアウト<br>• javax.servlet.http.HttpSession.invalidate() | セッションの破棄 | Explicit メモリブロックの解放 |

HTTP セッションとは別に，HTTP セッション管理用オブジェクトのために，Web アプリケーション数＋ 2 個※の Explicit メモリブロックが Web コンテナの内部で使用されます。

注※　Web コンテナで，内部的に 2 個の管理用オブジェクトを持つため，その個数を加算します。

なお，HTTP セッションで使用する Explicit ヒープのメモリサイズは，HTTP セッションで利用する Explicit ヒープの省メモリ化機能を使用することで削減できます。詳細は，「7.11　HTTP セッションで利用する Explicit ヒープのメモリ使用量の削減」を参照してください。また，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」を参照してアプリケーションを実装すると，HTTP セッションに明示管理ヒープ機能を効率良く適用できます。

## 7.5　アプリケーションで任意に Explicit ヒープに配置できるオブジェクト

この節では，Explicit ヒープに任意に配置できるオブジェクトについて説明します。

J2EE サーバで設定されている以外の Java オブジェクトを Explicit ヒープに配置したい場合は，自動配置設定ファイルを使用して配置したいオブジェクトを指定します。自動配置設定ファイルの詳細については「7.13.2　自動配置設定ファイルを使った明示管理ヒープ機能の使用」を参照してください。

また，明示管理ヒープの自動解放機能については，「7.7　自動解放機能が有効な場合の Explicit メモリブロックの解放」を参照してください。

明示管理ヒープ機能 API を使って実装する場合は「7.12　明示管理ヒープ機能 API を使った Java プログラムの実装」を参照してください。

**ポイント**

Java ヒープおよび Explicit ヒープをチューニングしても Tenured 領域のメモリ使用量が増加し，かつ FullGC が発生する要因になる Java オブジェクトがある場合には，Explicit ヒープへオブジェクトを配置することを検討してください。

### 7.5.1　Explicit ヒープに配置できるオブジェクトの条件

ここでは，Explicit ヒープに配置できるオブジェクトの前提と，配置すると効果があるオブジェクトについて説明します。

#### (1)　配置できるオブジェクトの前提

Explicit ヒープ（Explicit メモリブロック）に配置するオブジェクトは，次の前提を満たす必要があります。

- **Tenured 領域のメモリサイズ増加の要因になる長寿命なオブジェクトであること**

  Explicit メモリブロックに対するオブジェクトの配置および解放には，一定のオーバーヘッドが掛かります。このため，Explicit メモリブロックへのオブジェクトの配置と解放は，できるだけ少なくなるようにしてください。

  明示管理ヒープ機能を使用していない場合に CopyGC で回収の対象となっていた短寿命のオブジェクトを配置した場合，FullGC 抑止の目的に一致しない上，オーバーヘッドが掛かってしまいます。Explicit メモリブロックには，CopyGC で回収の対象にならない，長寿命のオブジェクトを配置するようにしてください。

  Tenured 領域のメモリサイズ増加の要因になる長寿命のオブジェクトを特定する方法については，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「9.8　Tenured 領域内不要オブジェクト統計機能」を参照してください。

- **生存期間が既知であること（明示管理ヒープ機能 API を使用する場合だけ）**

明示管理ヒープ機能 API を使用して明示管理ヒープを使用する場合，Explicit メモリブロックは GC の対象にならないため，使用済みのオブジェクトは自動的に回収されません。

Explicit メモリブロックに配置したオブジェクトは，アプリケーションで明示的に解放する必要がありますが，生存期間がわからないオブジェクトの場合，明示的に解放できません。このため，生存期間が既知であるオブジェクトを配置するようにしてください。

### (2)　配置すると効果があるオブジェクト

明示管理ヒープ機能は，長寿命オブジェクトのうち，一定期間後に破棄されて FullGC で回収されるオブジェクトが，Tenured 領域に昇格することを防ぐ機能です。このため，アプリケーションの停止まで使用されるオブジェクトなど，FullGC でも回収されないオブジェクトに対しては，適用する必要がありません。

Explicit ヒープに配置すると効果があるオブジェクトとは，次のようなオブジェクトです。

- 一定のライフサイクルで生成と破棄が実行される。
- ライフサイクルの期間が CopyGC によってオブジェクトが昇格する期間よりも長いために，破棄実施後に使用済みのオブジェクトが正しく回収されない。

このようなオブジェクトを Explicit ヒープに配置することで，Tenured 領域に不要なオブジェクトが残存することを防ぎ，FullGC を抑止できます。

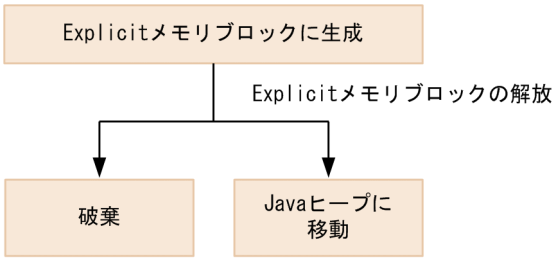
## 7.5.2　オブジェクトのライフサイクルと状態遷移

Explicit メモリブロックに配置するオブジェクトのライフサイクルと状態遷移について説明します。

Explicit メモリブロックに配置したオブジェクトは，生存期間に基づいて，明示管理ヒープ機能 API を使用して明示的に生成，解放する必要があります。オブジェクトの生存期間および寿命は，アプリケーションの処理によって異なります。

Explicit メモリブロックに配置するオブジェクトのライフサイクルを次の図に示します。

図 7-8　Explicit メモリブロックに配置するオブジェクトのライフサイクル




オブジェクトは，直接 Explicit メモリブロックに生成されます。その後，明示管理ヒープ機能 API によって Explicit メモリブロックの解放が実行されると，状態によって，破棄されるかまたは Java ヒープに移動されます。解放処理実行時の動作については，「7.8.2　自動解放機能が無効な場合の Explicit メモリブロックの解放処理」を参照してください。

## 7.6 Explicit メモリブロックのライフサイクルと実行される処理

この節では，Explicit メモリブロックのライフサイクルと，それぞれの段階で実行される処理について説明します。

### 7.6.1 Explicit メモリブロックのライフサイクルと状態

ここでは，Explicit メモリブロックのライフサイクルと状態について説明します。

明示管理ヒープ機能では，Explicit メモリブロックを解放する方法として，次の 2 種類の解放処理があります。

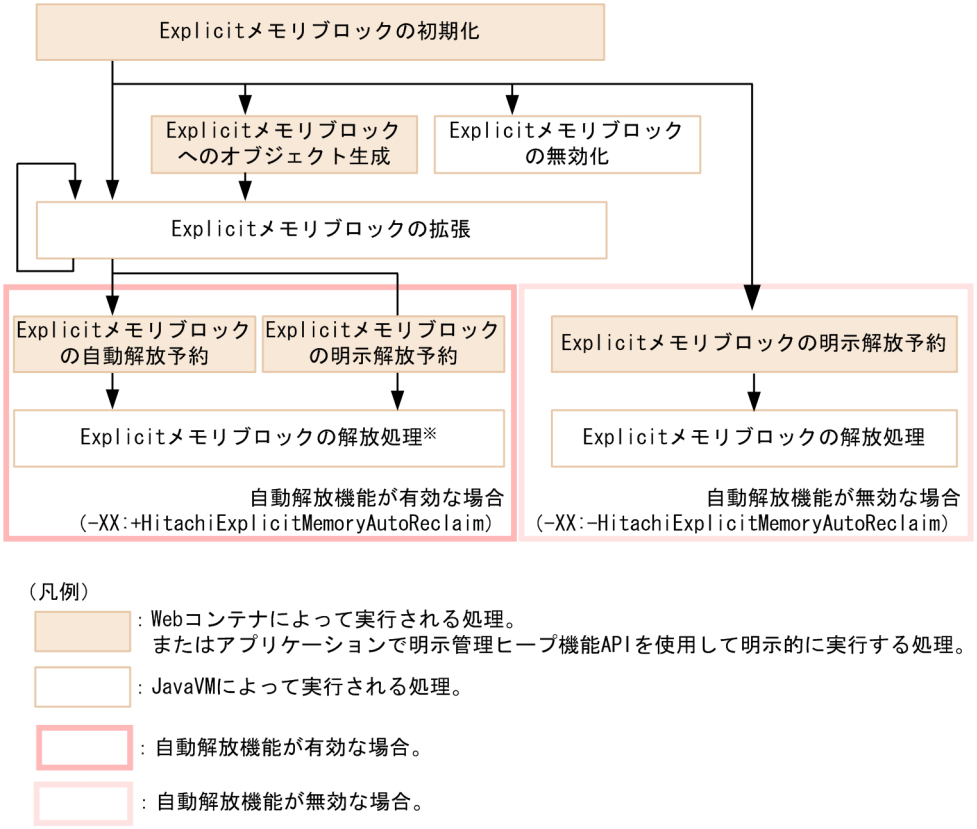
- Explicit メモリブロックの自動解放処理（自動解放機能が有効な場合に実行される解放処理）
- Explicit メモリブロックの明示解放処理（自動解放機能が無効な場合に実行される解放処理）

Explicit メモリブロックの解放処理が異なることで，指定方法や処理が異なります。8.7 以降で，解放処理ごとに詳しく説明します。

#### (1) Explicit メモリブロックのライフサイクル

Explicit メモリブロックのライフサイクルを次の図に示します。

図 7-9　Explicit メモリブロックのライフサイクル




注※　javagcコマンドによるExplicitメモリブロックの解放処理も含まれます。

図に示したライフサイクルの各段階について説明します。

### Explicit メモリブロックの初期化と Explicit メモリブロックの無効化

Explicit メモリブロックの初期化が実行され，Explicit メモリブロックが生成されます。

- HTTP セッションに関するオブジェクトを格納する Explicit メモリブロックは，Web コンテナによって初期化されます。
- アプリケーションで任意のオブジェクトを Explicit ヒープに配置する場合は，明示管理ヒープ機能 API を使用して，明示的に Explicit メモリブロックを初期化します。

なお，初期化実行時の状態によっては，Explicit メモリブロックが無効化されることがあります。

Explicit メモリブロックの初期化で実行される処理，および Explicit メモリブロックが無効化される条件については，「7.6.2　Explicit メモリブロックの初期化」で説明します。

### Explicit メモリブロックへのオブジェクトの生成

アプリケーションで任意のオブジェクトを Explicit メモリブロックに格納する場合，明示管理ヒープ機能 API を使用して，オブジェクトを Explicit メモリブロックに生成，配置します。

Explicit メモリブロックへのオブジェクトの生成については，「7.6.3　Explicit メモリブロックへのオブジェクトの直接生成」で説明します。

### Explicit メモリブロックの拡張

利用中にオブジェクトを配置する領域が不足した場合，JavaVM によって Explicit メモリブロックの拡張が実行されます。

Explicit メモリブロックの拡張については，「7.6.4　Explicit メモリブロックの拡張」で説明します。

### Explicit メモリブロックの解放予約と解放処理

Explicit メモリブロックの解放予約と解放処理は，明示管理ヒープ機能の自動解放機能の有効（-XX:+HitachiExplicitMemoryAutoReclaim）・無効（-XX:-HitachiExplicitMemoryAutoReclaim）で挙動が異なります。

#### 自動解放機能が有効な場合（-XX:+HitachiExplicitMemoryAutoReclaim）

- **Explicit メモリブロックの明示解放予約**

  明示管理ヒープ機能 API によって指定された，Explicit メモリブロックに対して，解放が予約されます。自動解放機能が有効な場合の Explicit メモリブロックの明示解放予約については，「7.7.1　自動解放機能が有効な場合の Explicit メモリブロックの明示解放予約」を参照してください。

- **Explicit メモリブロックの自動解放予約**

  明示管理ヒープの自動解放機能によって処理される，Explicit メモリブロックに対して解放が予約されます。自動解放機能が有効な場合の Explicit メモリブロックの自動解放予約については，「7.7.2　自動解放機能が有効な場合の Explicit メモリブロックの自動解放予約」を参照してください。

- **Explicit メモリブロックの解放処理**

  明示解放予約，または自動解放予約によって予約された Explicit メモリブロックに配置されているオブジェクトを解放します。自動解放機能が有効な場合の Explicit メモリブロックの解放処理については，「7.7.3　自動解放機能が有効な場合の Explicit メモリブロックの解放処理」を参照してください。

なお，javagc コマンドを使用すると，解放されていない Explicit メモリブロックに対して，任意のタイミングで解放処理を実行できます。javagc コマンドによる Explicit メモリブロックの解放処理については，「7.9　javagc コマンドによる Explicit メモリブロックの解放」を参照してください。

#### 自動解放機能が無効な場合（-XX:-HitachiExplicitMemoryAutoReclaim）

- **Explicit メモリブロックの明示解放予約**

  Explicit メモリブロックに配置したオブジェクトが不要になった場合は，Explicit メモリブロック単位で解放を予約します。

1. HTTP セッションに関するオブジェクトを格納した Explicit メモリブロックは，Web コンテナによって解放が予約されます。
2. アプリケーションで任意のオブジェクトを Explicit メモリブロックに格納した場合は，明示管理ヒープ機能 API を使用して，明示的に Explicit メモリブロックの解放を予約します。

自動解放機能が無効な場合の Explicit メモリブロックの明示解放予約については，「7.8.1　自動解放機能が無効な場合の Explicit メモリブロックの明示解放予約」を参照してください。なお，明示解放予約を実行した段階では，まだ Explicit メモリブロックは破棄されていません。

- Explicit メモリブロックの解放処理

  解放が予約された Explicit メモリブロックは，CopyGC または FullGC が発生したタイミングで，JavaVM によって解放されます。同時に Explicit メモリブロックに配置したオブジェクトも破棄されます。ただし，一部のオブジェクトは破棄されないで，Java ヒープに移動します。

  自動解放機能が無効な場合の Explicit メモリブロックの解放処理については，「7.8.2　自動解放機能が無効な場合の Explicit メモリブロックの解放処理」を参照してください。

### (2)　Explicit メモリブロックの状態

Explicit メモリブロックは，ライフサイクルの各段階で，有効，解放済み，解放予約済みなどの状態に遷移します。

さらに，有効な状態の Explicit メモリブロックは，次の表に示すサブ状態を保持します。

表 7-4　Explicit メモリブロックのサブ状態

| サブ状態 | Explicit メモリブロックの状態 |
|---|---|
| Enable | 初期状態です。Explicit メモリブロックのすべての機能が使用できます。 |
| Disable | 該当する Explicit メモリブロックに Java オブジェクトを移動できない状態です。Explicit メモリブロックを拡張したときに，この状態になることがあります。 |

## 7.6.2　Explicit メモリブロックの初期化

ここでは，Explicit メモリブロックの初期化の実行と，初期化時に実行される処理について説明します。

### (1)　実行契機

アプリケーションで任意のオブジェクトを Explicit ヒープに配置する場合，次に示す明示管理ヒープ機能 API を呼び出すことで Explicit メモリブロックが初期化されます。

- BasicExplicitMemory.BasicExplicitMemory()
- BasicExplicitMemory.BasicExplicitMemory(String name)

また，これらの API のほかに，自動配置設定ファイルで指定したオブジェクトが生成されたタイミングでも Explicit メモリブロックが初期化されます。なお，J2EE サーバがオブジェクトを配置する Explicit メモリブロックでは，Web コンテナによって初期化と最初のオブジェクトの配置が実行されます。実行契機については，「7.4　J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクト」を参照してください。

### (2)　実行される内容

Explicit メモリブロックが初期化されます。ただし，この段階では Explicit メモリブロック用のメモリ領域の確保は実行されません。

また，次の場合は，初期化が実行されません。

- **最大数の制限を超えて Explicit メモリブロックを初期化しようとした場合**
  現存する Explicit メモリブロックの数が 1,048,575 個の場合が該当します。
- **明示管理ヒープ機能がオフになっている場合**
  -XX:-HitachiUseExplicitMemory オプションが指定されている場合が該当します。

この場合，コンストラクタの実行は成功しますが，無効な Explicit メモリブロックとして扱われます。初期化した Explicit メモリブロック（ExplicitMemory インスタンス）に対する処理は，すべて無効になります。

## 7.6.3 Explicit メモリブロックへのオブジェクトの直接生成

ここでは，Explicit メモリブロックにオブジェクトを直接生成する方法と実行される処理について説明します。

アプリケーションで Explicit メモリブロックにオブジェクトを生成するためには，「(1)実行契機」で示す API を使用します。API を実行すると，引数で指定したクラスのオブジェクトが Explicit メモリブロックに生成されます。ただし，そのオブジェクトのコンストラクタなどによる初期化処理中に生成されたオブジェクトについては，Java ヒープに生成されます。

### (1) 実行契機

アプリケーションで明示管理ヒープ機能を使用する場合，Explicit メモリブロックへのオブジェクトの直接生成は，次のどれかの明示管理ヒープ機能 API を呼び出すことで実行できます。

- ExplicitMemory.newInstance(Class type)
- ExplicitMemory.newInstance(Class type, Object... args)
- ExplicitMemory.newInstance(java.lang.reflect.Constructor cons, Object... args)
- ExplicitMemory.newArray(Class type, int number)
- ExplicitMemory.newArray(Class type, int[] dimensions)

また，これらの API のほかに，明示管理ヒープ自動配置設定ファイルで指定したオブジェクトが生成され，Explicit メモリブロックの初期化が実行されたタイミングでも，Explicit メモリブロックにオブジェクトが直接生成されます。

J2EE サーバが配置するオブジェクトについては，意識する必要はありません。

### (2) 実行される内容

API ごとに実行される内容を説明します。

表 7-5　API ごとに実行される内容

| API | 実施される内容 |
|---|---|
| ExplicitMemory.newInstance(Class type) | 引数 type で指定したクラスをインスタンス化して，レシーバが示す Explicit メモリブロックに配置します。 |
| ExplicitMemory.newInstance(Class type, Object... args) | |
| ExplicitMemory.newInstance(java.lang.reflect.Constructor cons, Object... args) | java.lang.reflect.Constructor が示すクラスをインスタンス化して，レシーバが示す Explicit メモリブロックに配置します。 |
| ExplicitMemory.newArray(Class type, int number) | 引数 type で指定したクラスの配列を，引数 number で指定した長さでインスタンス化して，レシーバが示す Explicit メモリブロックに配置します。 |
| ExplicitMemory.newArray(Class type, int[] dimensions) | 引数 type で指定したクラスの配列を，引数 dimensions で指定した次元数でインスタンス化して，レシーバが示す Explicit メモリブロックに配置します。 |

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」の「10.3 ExplicitMemory クラス」を参照してください。

ただし，配置先の Explicit メモリブロックに必要な領域を確保できない場合は，生成したオブジェクトが Explicit メモリブロックに配置されません。生成したオブジェクトは Java ヒープに配置されます。

領域を確保できない要因および実行される処理については，「7.6.4　Explicit メモリブロックの拡張」の拡張処理を実行できない場合の説明を参照してください。

## 7.6.4　Explicit メモリブロックの拡張

ここでは，Explicit メモリブロックの拡張処理について説明します。拡張処理が実行されると，Explicit メモリブロック内の空き領域が増加します。

### (1)　実行契機

拡張は，JavaVM によって次の契機で実行されます。

- Explicit メモリブロックに最初のオブジェクトが配置される時
- オブジェクトを配置するための Explicit メモリブロックの空き領域が足りない場合

明示管理ヒープ機能 API を使用したアプリケーションから Explicit メモリブロックにオブジェクトを配置しようとした場合に，オブジェクトのサイズが配置対象の Explicit メモリブロックの空き領域を超えるとき，拡張処理が実行されます。

Explicit メモリブロックの初期化後，初めて Explicit メモリブロックにオブジェクトを配置するときには，必ず拡張処理が実行されます。

なお，J2EE サーバがオブジェクトを配置する Explicit メモリブロックでは，Web コンテナによって初期化と最初のオブジェクトの配置が実行されます。実行契機については，「7.4　J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクト」を参照してください。

### (2)　実行される内容

JavaVM によって，OS からメモリ領域が確保され，該当の Explicit メモリブロックが拡張されます。メモリ領域の確保には，OS のメモリ確保 API が使用されます。

ただし，次の場合，拡張処理が実行されません。

- **Explicit ヒープの最大値の制限を超えて拡張しようとした場合**

  すべての Explicit メモリブロックの合計サイズに，拡張しようとしたサイズを加えた値が，-XX:HitachiExplicitHeapMaxSize オプションに指定した値を超えた場合です。

  該当する Explicit メモリブロックのサブ状態が「Disable」に変化して，この Explicit メモリブロックへのオブジェクトの配置が中止されます。

  「Disable」に変化した Explicit メモリブロックには，以降オブジェクトは配置されません。

  -XX:HitachiExplicitHeapMaxSize オプションについては，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)」を参照してください。

- **サブ状態が「Disable」になっている Explicit メモリブロックを拡張しようとした場合**

  サブ状態が「Disable」になっている Explicit メモリブロックを拡張しようとした場合は，Explicit メモリブロックへのオブジェクトの配置が中止されます。

Explicit メモリブロックの拡張が失敗した場合のサブ状態の変化について，要因ごとに次の表に示します。

表 7-6　Explicit メモリブロックの拡張が失敗した場合のサブ状態の変化

| 拡張が失敗した要因 | サブ状態の変化 |
|---|---|
| OS からのメモリ領域の確保に失敗した | 「Enable」→「Disable」 |
| Explicit ヒープの最大値の制限を超えて拡張しようとした | 「Enable」→「Disable」 |
| サブ状態が「Disable」になっている Explicit メモリブロックを拡張しようとした | 変化しません。 |

## 7.6.5　参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動

Java ヒープから Explicit メモリブロックへのオブジェクトの移動では，Explicit メモリブロック内にあるオブジェクトから参照されている Java ヒープ内のオブジェクトが，自動で Explicit メモリブロックへ移動します。このため，移動するオブジェクトと関係を持つオブジェクトに対して，Java ヒープから Explicit メモリブロックへ移動する設定を作り込む必要はありません。ただし，-

XX:+ExplicitMemoryUseExcludeClass オプションを指定した場合，明示管理ヒープ機能適用除外設定ファイルに記述されたクラスのオブジェクトは，Explicit メモリブロックへ移動しません。

なお，参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動は，自動配置機能で作成した Explicit メモリブロックが対象となります。明示管理ヒープ API で作成した Explicit メモリブロックは対象外です。

**参考**

FullGC の発生時，大量のオブジェクトが Explicit ヒープに移動したあとに次の現象が発生する場合は，参照関係に基づく移動の対象となるオブジェクトを Explicit メモリブロックへ移動しないようにする運用を検討してください。

- Explicit メモリブロックの自動解放処理に時間が掛かる
- Tenured 領域の使用量が少ない

Explicit メモリブロックへオブジェクトを移動しないためには，次の機能を使用します。

1. Explicit メモリブロックへのオブジェクト移動制御機能
2. 明示管理ヒープ機能適用除外クラス指定機能

1.の機能は，FullGC 発生時にオブジェクトを Explicit ヒープへ移動しない機能であり，Explicit メモリブロックの自動解放処理に掛かる時間を短縮できます。また，2.の機能は，CopyGC 発生時に設定ファイルに指定したクラスのオブジェクトを Explicit ヒープへ移動しない機能であり，指定するクラスによっては Explicit ヒープへ移動するオブジェクトの量を少なくできます。なお，2.の機能を使用すると，1.の機能も有効になります。2.の機能は，1.の機能を利用しても，Explicit ヒープへのオブジェクトの移動が多く，Explicit メモリブロックの自動解放処理に時間が掛かるような場合に利用します。

## (1) 実行契機

CopyGC，および FullGC が発生したタイミングで実行されます。

## (2) 実行される内容

CopyGC または FullGC の処理が終了したあと，JavaVM によって解放予約されていない Explicit メモリブロックがあるかどうか調査されます。調査の基点となるオブジェクトから参照関係を調べ，参照先がなくなるまで調査を続けます。参照関係を調査する際，Java ヒープ外の領域は調査対象外です。また，Explicit メモリブロックから参照されているオブジェクトは移動対象のオブジェクトとなります。

- CopyGC の場合
  CopyGC が実行された場合には，これらに加えて次の規則に従って動作します。
  - 参照されている Explicit メモリブロック内のオブジェクトが昇格するタイミングで移動します。

- Metaspace 領域，Explicit ヒープ，および Tenured 領域への参照についても調査対象としません。
- Explicit メモリブロックが解放予約されている場合でも，移動対象として扱います。

  Explicit メモリブロックの領域が確保できず，オブジェクトが Java ヒープに移動するときに，移動先の Java ヒープに空き領域がなくなり，移動できなくなった場合が該当します。この場合は，FullGC が実行され，Java ヒープに空き領域が確保されます。FullGC 実行後，Java ヒープへのオブジェクトの移動が実行されます。

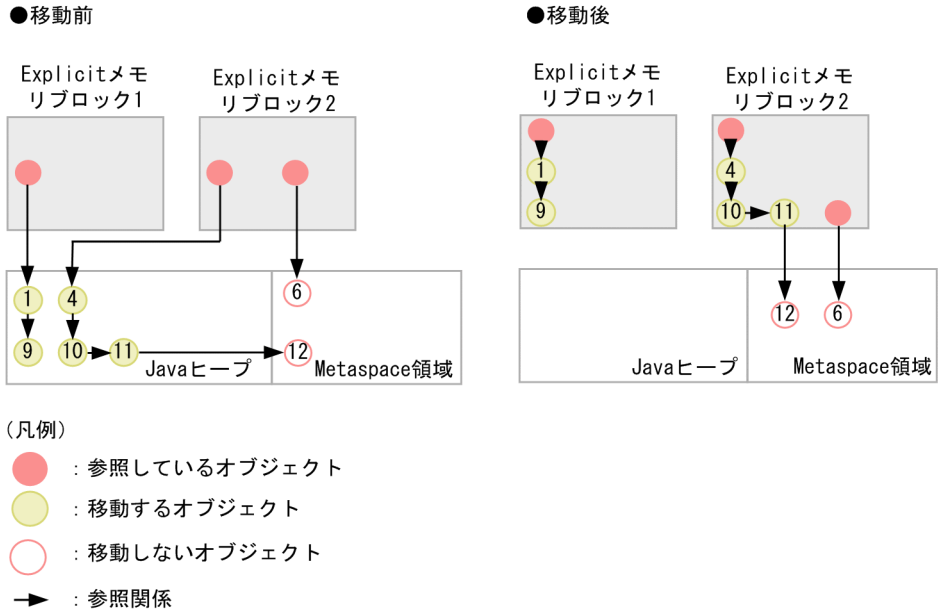
- FullGC の場合

  FullGC が実行された場合には，これらに加えて次の規則に従って動作します。

  - -XX:ExplicitMemoryFullGCPolicy オプションに 1 を指定した場合，参照関係に基づく移動の対象となるオブジェクトは，Explicit メモリブロックへ移動しません。なお，New 領域にあるオブジェクトは Tenured 領域へ移動します。

これらの規則に従ったオブジェクトの移動の流れを図 8-11 および図 8-12 で例を使って説明します。なお，ここで説明するオブジェクトの移動の流れは，-XX:ExplicitMemoryFullGCPolicy オプションに 0 を指定していることを前提としています。

### 図 7-10　参照関係に基づいて移動するオブジェクト（例 1）

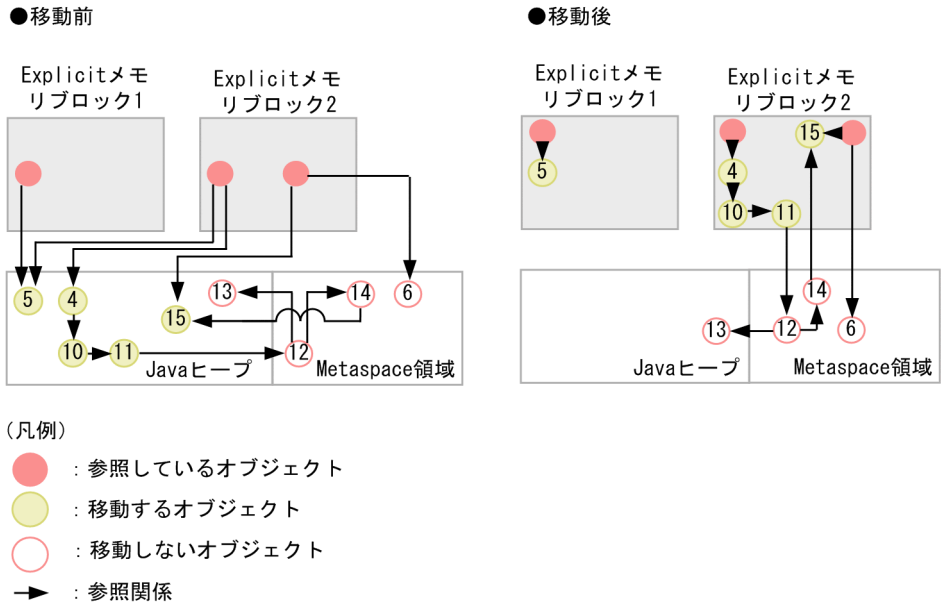



図中のオブジェクトは次の順番に動作します。

1. オブジェクト 1 は，Explicit メモリブロック 1 内のオブジェクトから参照されています。そのため，オブジェクト 1 は Explicit メモリブロック 1 へ移動します。
2. オブジェクト 9 も，オブジェクト 1 から参照されているため Explicit メモリブロック 1 に移動します。
3. 1.および 2.の処理と同様に，オブジェクト 4，オブジェクト 10，およびオブジェクト 11 は Explicit メモリブロック 2 へ移動します。
4. オブジェクト 6 は，Explicit メモリブロック 2 内のオブジェクトから参照されています。しかし，Java ヒープ内のオブジェクトではないためこのままとなります。

5. 4.と同様に，オブジェクト 12 についてもこのままとなります。

図 7-11　参照関係に基づいて移動するオブジェクト（例 2）




図中のオブジェクトは次の順番に動作します。

1. オブジェクト 13 は，Java ヒープ内にあり，また Explicit メモリブロック 2 内のオブジェクトから到達できます。しかし，オブジェクト 12 の時点で調査が打ち切られているため，移動しません。
2. オブジェクト 15 は，オブジェクト 13 と同様に Metaspace 領域からの参照があります。しかし，この参照に加え，Explicit メモリブロック 2 内のオブジェクトから Metaspace 領域やほかの Explicit メモリブロックを介さずに到達できます。そのため，Explicit メモリブロック 2 に移動します。
3. オブジェクト 5 は Explicit メモリブロック 1，および Explicit メモリブロック 2 の両方から参照されていますが，Explicit メモリブロック 1 に移動します。

   なお，オブジェクト 5 は Explicit メモリブロック 1，および Explicit メモリブロック 2 の両方から参照されています。このような場合，Explicit メモリブロック 1 または 2 のどちらかに移動しますが，どちらの Explicit メモリブロックに移動するかは未定義です。

また，次の条件に該当する場合は，例で説明した動作と異なります。

- Explicit メモリブロックの空き領域を確保できない場合

  オブジェクトを Explicit メモリブロックに配置する際，配置先の Explicit メモリブロックに配置対象となるオブジェクトを配置する空き領域がない場合が該当します。この場合，Explicit メモリブロックにオブジェクトを配置できません。配置できなかったオブジェクトは，Java ヒープ領域へ配置されます。なお，API の利用方法が誤っている場合，API レベルの例外が発生することがあります。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「10.7　例外クラス」を参照してください。

## 7.6.6　ライフサイクルの各段階で出力されるイベントログ

Explicit メモリブロックのライフサイクルの各段階では，イベントログが出力されます。イベントログは，出力契機になるイベントが発生したときに出力されます。

ライフサイクルの各段階と，イベントログの出力契機の対応を次の表に示します。ログ出力レベルの設定によって，ログを出力する契機になるイベントは異なります。

表 7-7　ライフサイクルの各段階と出力されるイベントログの対応

| ライフサイクルの段階 | イベントログの出力契機 | ログ出力レベル |
|---|---|---|
| Explicit メモリブロックの初期化 | Explicit メモリブロックの初期化 | verbose |
| | Explicit メモリブロックの初期化（詳細情報出力） | debug |
| | Explicit メモリブロックの初期化失敗 | verbose |
| Explicit メモリブロックの拡張 | Explicit メモリブロックのサブ状態の Disable 化 | verbose |
| Explicit メモリブロックの明示解放予約 | ファイナライザによる Explicit メモリブロックの解放予約 | verbose |
| Explicit メモリブロックの自動解放自動予約 | | verbose |
| Explicit メモリブロックの自動解放明示予約 | | verbose |
| Explicit メモリブロックの明示解放処理 | Explicit メモリブロックの明示解放処理 | normal |
| | Explicit メモリブロックの明示解放処理（詳細情報出力） | verbose |
| | Explicit メモリブロックの明示解放処理時の Java ヒープあふれ | normal |
| | Explicit メモリブロックの明示解放による Java ヒープへのオブジェクトの移動 | debug |
| Explicit メモリブロックの自動解放処理 | Explicit メモリブロックの自動解放処理 | normal |
| | Explicit メモリブロックの自動解放処理時の Java ヒープあふれ | normal |
| Explicit メモリブロックへのオブジェクト直接生成 | Explicit メモリブロックへのオブジェクト生成 | verbose |

このほか，ライフサイクルの段階と対応しないイベントとして，GC 発生時にイベントログが出力されます。

明示管理ヒープ機能のイベントログ取得の設定については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.17　JavaVM の資料取得の設定」を参照してください。明示管理ヒープ機能のイベントログの内容については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.19 明示管理ヒープ機能のイベントログ」を参照してください。

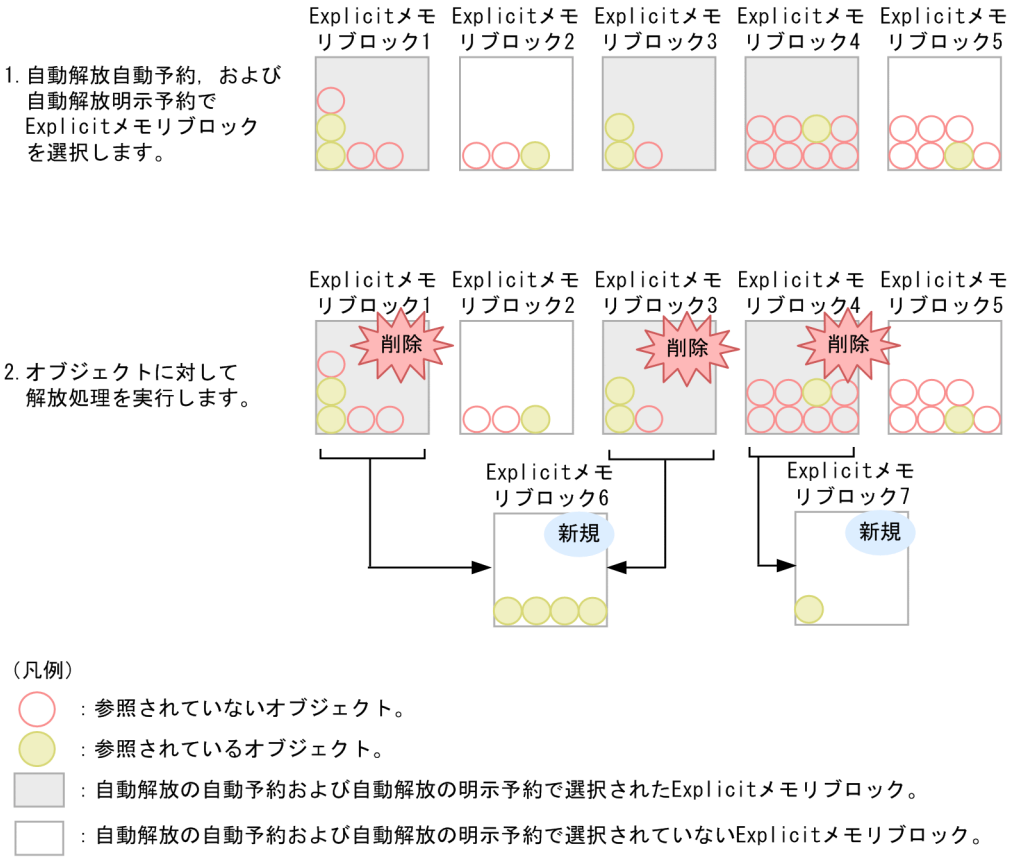
## 7.7　自動解放機能が有効な場合の Explicit メモリブロックの解放

ここでは，明示管理ヒープの自動解放について説明します。

Explicit メモリブロックの自動解放は，解放予約と解放処理の 2 段階に分けて実行されます。複数の Explicit メモリブロックの解放をそれぞれ予約しておき，解放処理を一斉に実行することで，効率良く処理できます。

また，自動解放予約には自動解放の明示予約，および自動解放の自動予約の 2 種類があります。Explicit メモリブロックの自動解放について，図で説明します。

図 7-12　Explicit メモリブロックの自動解放




以降では，自動解放機能が有効な場合に実行される処理について説明します。

### 7.7.1　自動解放機能が有効な場合の Explicit メモリブロックの明示解放予約

自動解放機能が有効な場合の Explicit メモリブロックの明示解放予約では，API によって明示的に Explicit メモリブロックに対して解放を予約します。

#### (1)　実行契機

自動解放機能が有効な場合の Explicit メモリブロックの明示解放予約では，次のどれかの明示管理ヒープ機能 API を呼び出すことで実行できます。

- ExplicitMemory.reclaim(ExplicitMemory... areas)
- ExplicitMemory.reclaim(ExplicitMemory area)
- ExplicitMemory.reclaim(ExplicitMemory area0,ExplicitMemory area1)
- ExplicitMemory.reclaim(Iterable<ExplicitMemory> areas)
- BasicExplicitMemory.finalize()

### (2) 実行される内容

(1)で示した API が呼び出されると，API の引数で指定された Explicit メモリブロック領域が解放予約された状態になります。

なお，API の利用方法が誤っている場合，API レベルの例外が発生することがあります。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「10.7　例外クラス」を参照してください。

## 7.7.2 自動解放機能が有効な場合の Explicit メモリブロックの自動解放予約

自動解放機能が有効な場合の Explicit メモリブロックの自動解放予約では，JavaVM が自動的に Explicit メモリブロックに対して解放を予約します。自動配置機能で配置した Explicit メモリブロックが対象となります。

### (1) 実行契機

GC が発生したタイミングで JavaVM によって実行されます。

### (2) 実行される内容

GC が発生したタイミングで，JavaVM が自動で Explicit メモリブロック領域を予約します。

## 7.7.3 自動解放機能が有効な場合の Explicit メモリブロックの解放処理

自動解放機能が有効な場合の Explicit メモリブロックの解放処理は，事前に自動解放予約，および明示解放予約で予約されている Explicit メモリブロックに対して実行されます。不要になった Explicit メモリブロックは，解放処理によってメモリから削除されます。

なお，外部（解放対象外の Explicit メモリブロック）から参照されているオブジェクトがある場合，オブジェクトは新規の Explicit メモリブロックに移動されます。

### (1) 実行契機

自動解放予約で解放予約が実行されるのと同じ GC で，JavaVM によって実行されます。

## (2) 実行される内容

実行される内容は，解放対象外の Explicit メモリブロックから参照されているオブジェクトの挙動以外は，自動解放機能が無効な場合の Explicit メモリブロックの解放処理の処理内容と同じです。Explicit メモリブロックの解放処理で実行される内容については，「7.8.2　自動解放機能が無効な場合の Explicit メモリブロックの解放処理」を参照してください。

また，次の条件に該当する場合は動作が異なります。

- **Explicit メモリブロックの空き領域を確保できない場合**

  オブジェクトを Explicit メモリブロックに配置する際，配置先の Explicit メモリブロックに配置対象となるオブジェクトを配置する空き領域がない場合が該当します。この場合，Explicit メモリブロックにオブジェクトを配置できません。配置できなかったオブジェクトは，Java ヒープ領域へ配置されます。

- **Java ヒープへのオブジェクト移動時に Java ヒープがあふれた場合**

  Explicit メモリブロックの領域を確保できず，オブジェクトが Java ヒープに移動する場合に，移動先の Java ヒープに空き領域がなくなり，移動できなくなったときが該当します。この場合は，FullGC が実行され，Java ヒープに空き領域が確保されます。FullGC 実行後，Java ヒープへのオブジェクトの移動が実行されます。

  FullGC を実行しても Java オブジェクトを移動するために必要な空き領域が確保できない場合は，ログファイルが出力され，オブジェクトが再度 Explicit メモリブロックに配置されます。出力されるログ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 機能解説 保守／移行編」の「4.19　明示管理ヒープ機能のイベントログ」を参照してください。

- **FullGC で十分な空き領域を作成できない場合**

  Java ヒープに空き領域がなくなり，FullGC を実行しても Java オブジェクトを移動するために必要な空き領域が確保できない場合が該当します。この場合は，C ヒープ不足の場合と同様に，JavaVM がアボートします。ただし，C ヒープ不足時には「request *nnn* bytes」として *nnn* に必要なメモリサイズが出力されますが，JavaVM のアボート時には，*nnn* として常に「0」が出力されます。

## 7.8　自動解放機能が無効な場合の Explicit メモリブロックの解放

ここでは，明示管理ヒープの明示解放について説明します。

Explicit メモリブロックの明示解放は，明示管理ヒープの自動解放と同様に，解放予約と解放処理の 2 段階に分けて実行されます。複数の Explicit メモリブロックの解放をそれぞれ予約しておき，解放処理を一斉に実行します。

以降では，明示解放機能で実行される処理について説明します。

### 7.8.1　自動解放機能が無効な場合の Explicit メモリブロックの明示解放予約

Explicit メモリブロックの解放は，解放予約と実際の解放処理の 2 段階に分けて実行されます。複数の Explicit メモリブロックの解放をそれぞれ予約しておき，解放処理を一斉に実行することで，効率良く処理できます。

#### (1)　実行契機

アプリケーションで任意のオブジェクトを Explicit ヒープに配置した場合，Explicit メモリブロックの解放予約は，次のどれかの明示管理ヒープ機能 API を呼び出すことで実行できます。

- ExplicitMemory.reclaim(ExplicitMemory... areas)
- ExplicitMemory.reclaim(ExplicitMemory area)
- ExplicitMemory.reclaim(ExplicitMemory area0,ExplicitMemory area1)
- ExplicitMemory.reclaim(Iterable<ExplicitMemory> areas)
- BasicExplicitMemory.finalize()

J2EE サーバが配置したオブジェクトについては，Web コンテナによって解放が予約されます。実行契機については，「7.4　J2EE サーバ利用時に Explicit ヒープに配置されるオブジェクト」を参照してください。

#### (2)　実行される内容

(1)で示した API が呼び出されると，API の引数で指定された Explicit メモリブロック領域が解放予約された状態になります。

なお，API の利用方法が誤っている場合，API レベルの例外が発生することがあります。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「10.7　例外クラス」を参照してください。

## 7.8.2　自動解放機能が無効な場合の Explicit メモリブロックの解放処理

解放処理は，事前に解放予約されている Explicit メモリブロックに対して実行されます。不要になった Explicit メモリブロックは，解放処理によってメモリから実際に削除されます。

### (1)　実行契機

解放処理は，次の契機で JavaVM によって実行されます。

- CopyGC 発生時
- FullGC 発生時

### (2)　実行される内容

CopyGC または FullGC の処理が終了したあと，JavaVM によって解放予約された Explicit メモリブロックがあるかどうかが調査されます。該当する Explicit メモリブロックが一つ以上ある場合は，その Explicit メモリブロックが解放されます。解放は，OS のメモリ解放 API で実行されます。その際，解放された Explicit メモリブロック内のオブジェクトは，破棄されます。

ただし，解放する Explicit メモリブロック内のオブジェクトのうち，どこからも参照されていない finalize メソッドを実装しているクラスのオブジェクトは，破棄されません。このオブジェクトは，ファイナライズキューに登録され，Java ヒープに移動されます。

また，解放対象の Explicit メモリブロックのオブジェクトが次の条件に該当する場合は動作が異なります。

#### (a)　外部（解放対象の Explicit メモリブロック以外）から参照されている場合

解放対象の Explicit メモリブロックのオブジェクトが，次の領域のオブジェクトから参照されている場合が該当します。

- Java ヒープ
- Metaspace 領域
- 解放対象でない Explicit メモリブロック

それぞれの場合に実行される内容を次に示します。

**Java ヒープまたは Metaspace 領域から参照されている場合**

Java ヒープまたは Metaspace 領域内のオブジェクトから参照されている場合，そのオブジェクトは破棄されません。

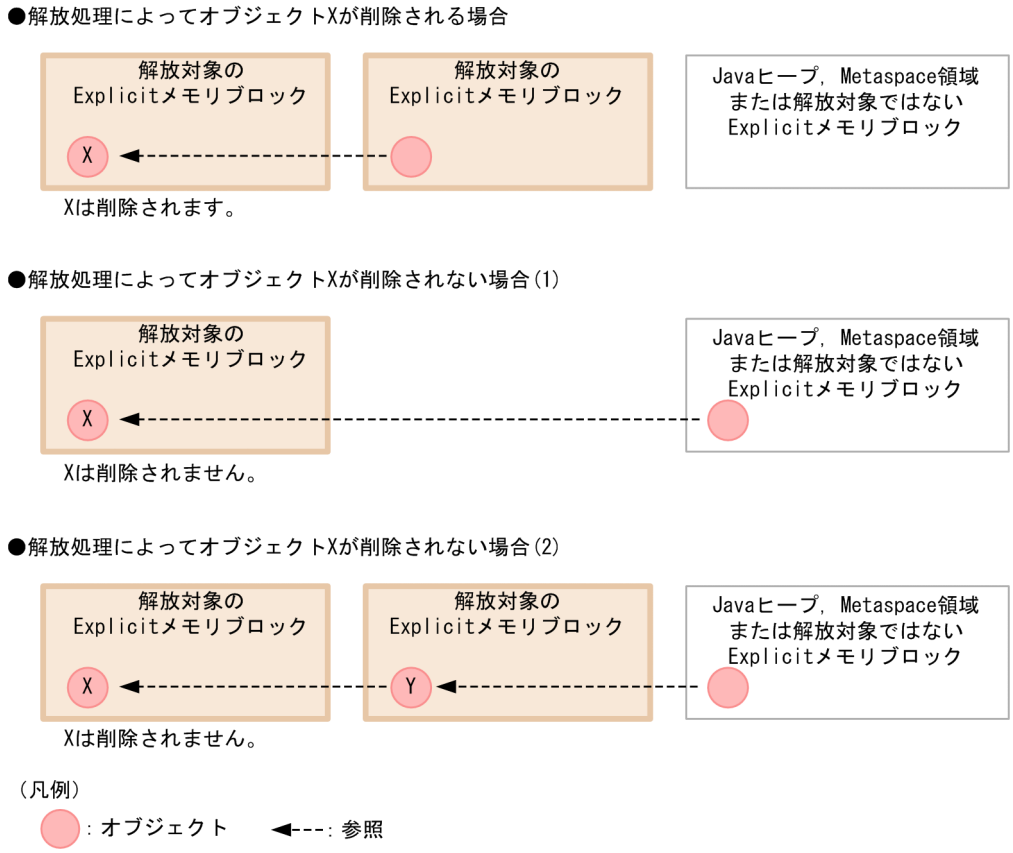
該当するオブジェクトは，Java ヒープの Tenured 領域に優先的に移動されます。ただし，Tenured 領域に空き容量がない場合，または Tenured 領域があふれた場合は，New 領域に移動されます。また，すでに使用されていない Tenured 領域のオブジェクトから参照されている場合も，移動の対象になります。

### 解放対象ではない Explicit メモリブロックから参照されている場合

解放対象ではない Explicit メモリブロック内のオブジェクトから参照されている場合，そのオブジェクトは破棄されません。また，参照元のオブジェクトが解放対象の Explicit メモリブロック内のオブジェクトである場合も，そのオブジェクトが破棄されないで Java ヒープに移動するオブジェクトである場合は，参照されているオブジェクトも削除されません。

Explicit メモリブロック解放時に外部から参照されている場合の動作を次の図に示します。

### 図 7-13　Explicit メモリブロック解放時に外部から参照されている場合の動作




解放対象の Explicit メモリブロックに含まれるオブジェクトである，オブジェクト X を例にして説明します。

解放対象の Explicit メモリブロックのオブジェクトから参照されている場合，オブジェクト X は削除されます。

Java ヒープ，Metaspace 領域または解放対象でない Explicit メモリブロックのオブジェクトから参照されている場合，オブジェクト X は削除されません。

また，オブジェクト X が解放対象の Explicit メモリブロックのオブジェクト（オブジェクト Y）から参照されている場合も，参照元であるオブジェクト Y が Java ヒープ，Metaspace 領域または解放対象でない Explicit メモリブロックのオブジェクトから参照されているときには，オブジェクト X は削除されません。この場合，オブジェクト X は，オブジェクト Y とともに Java ヒープに移動します。

### (b) Javaヒープへのオブジェクト移動時にJavaヒープがあふれた場合

外部から参照されているオブジェクトがJavaヒープに移動するときに，移動先のJavaヒープに空き領域がなくなり，移動できなくなった場合が該当します。

この場合は，FullGCが実行され，Javaヒープに空き領域が確保されます。FullGC実行後，Javaヒープへのオブジェクトの移動が実行されます。

### (c) FullGCで十分な空き領域を作成できない場合

Javaヒープに空き領域がなくなり，FullGCを実行してもJavaオブジェクトを移動するために必要な空き領域が確保できない場合が該当します。この場合は，Cヒープ不足の場合と同様に，JavaVMがアボートします。ただし，Cヒープ不足時には「request *nnn* bytes」として*nnn*に必要なメモリサイズが出力されますが，JavaVMのアボート時には，*nnn*として常に「0」が出力されます。

## 7.9　javagc コマンドによる Explicit メモリブロックの解放

ここでは，javagc コマンドによる明示管理ヒープの解放について説明します。

javagc コマンドによる Explicit メモリブロックの解放は，任意のタイミングで実行できます。これによって，自動解放機能が有効な場合の解放処理で解放されなかった Explicit メモリブロックを明示的に解放できます。

### 7.9.1　実行契機

-ehgc オプションを指定して javagc コマンドを実行したタイミングで，解放処理が実行されます。

**注意事項**

javagc コマンドによる Explicit メモリブロックの解放処理では，FullGC が実行されます。このため，動作中のアプリケーションに対する処理には適していません。アプリケーションのアンデプロイ時や夜間など，アプリケーションが動作していない時間帯に実行することをお勧めします。

### 7.9.2　実行される内容

javagc コマンドを実行すると，JavaVM によって FullGC が実行され，拡張 verbosegc 情報に GC の要因として「EMJavaGC Command」が出力されます。そのあと，次に示す Explicit メモリブロックが解放されます。

- 明示管理ヒープ機能の自動解放機能が有効な場合に，明示解放予約によって予約された Explicit メモリブロック
- 明示管理ヒープ自動配置設定ファイル，または JavaVM で生成された Explicit メモリブロック
- 前回の解放処理で解放されなかった Explicit メモリブロック

GC の要因については，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「-XX:[+|-]HitachiVerboseGCPrintCause（GC 要因内容出力オプション）」を参照してください。

また，次の場合は，解放処理が実行されません。

- **最大数の制限を超えて Explicit メモリブロックを解放しようとした場合**
  現存する Explicit メモリブロックの数が 1,048,575 個の場合が該当します。
- **明示管理ヒープ機能がオフになっている場合**
  -XX:-HitachiUseExplicitMemory オプションが指定されている場合が該当します。

この場合，コンストラクタの実行は成功しますが，無効な Explicit メモリブロック（ExplicitMemory インスタンス）として扱われます。

## 7.10　Explicit メモリブロックの自動解放処理に掛かる時間の短縮

ここでは，明示管理ヒープ機能の自動配置機能使用時に，Explicit メモリブロックの自動解放処理に掛かる時間を短縮する機能について説明します。自動解放処理時間の短縮には，**Explicit メモリブロックへのオブジェクト移動制御機能**と，この機能に加えて**明示管理ヒープ機能適用除外クラス指定機能**を使用します。Explicit メモリブロックへのオブジェクト移動制御機能は，明示管理ヒープ機能適用除外クラス指定機能の前提となる機能です。

これらの機能を使用すると，GC 発生時に，自動配置機能で配置したオブジェクトから参照されているオブジェクトが，Java ヒープから Explicit メモリブロックへ参照関係に基づいた移動をしなくなり，Explicit ヒープ領域の使用量が少なくなります。これによって，Explicit メモリブロックの自動解放処理に掛かる時間を短縮できます。オブジェクトの参照関係に基づいた移動については，「7.6.5　参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動」を参照してください。

### 7.10.1　適用効果があるかどうかの確認

Explicit メモリブロックへのオブジェクト移動制御機能は，FullGC 発生時にオブジェクトが Explicit ヒープへ移動しないようにする機能です。この機能を適用して効果があるかどうかは，スレッドダンプに含まれる Explicit メモリブロック情報と，明示管理ヒープ機能のイベントログを確認することで判定できます。Tenured 領域の使用量が少なく，次の条件を満たす Explicit メモリブロックがある場合は適用効果がありますので，機能の利用を検討してください。

- Explicit メモリブロック情報の<EM_NAME>が「NULL」である（一度自動解放処理を実施した Explicit メモリブロックである）。
- Explicit メモリブロック情報の<EH_TOTAL>の値が，ほかの Explicit メモリブロックに比べると，極端に大きい Explicit メモリブロックがある。
- FullGC 発生時に出力される明示管理ヒープ機能のイベントログで，<EH_USED_AF>が<EH_USED_BF>に比べて大幅に増加している。

明示管理ヒープ機能適用除外クラス指定機能は，オブジェクト移動制御機能を利用しても Explicit メモリブロックの自動解放処理に時間が掛かるような場合に，要因となるオブジェクトを指定して Explicit ヒープへ移動しないようにする機能です。明示管理ヒープ機能適用除外クラス指定機能を適用すると，設定ファイルに指定したクラスのオブジェクトが適用除外対象になります。この機能を適用して効果があるかどうかは，スレッドダンプに含まれる Explicit メモリブロック情報を確認することで判定できます。Tenured 領域の使用量が少なく，Explicit メモリブロック内に次の条件を満たすクラスがある場合は適用効果がありますので，機能の利用を検討してください。

- Explicit メモリブロック情報の<EM_NAME>が「NULL」である（一度自動解放処理を実施した Explicit メモリブロックである）。
- Explicit メモリブロック情報の<EH_TOTAL>の値が，ほかの Explicit メモリブロックに比べると，極端に大きい Explicit メモリブロックがある。

- Explicit メモリブロック情報にある，オブジェクト統計情報の<ISIZE>の値が大きく，オブジェクト解放率情報の<FRATIO>の値が低いオブジェクトがあり，そのクラスは Java SE が提供しているクラス以外である。

スレッドダンプに含まれる Explicit メモリブロック情報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 保守／移行編」の「5.5　JavaVM のスレッドダンプ」を参照してください。また，明示管理ヒープ機能のイベントログについては，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「5.11　明示管理ヒープ機能のイベントログ」を参照してください。

## 7.10.2　自動解放処理に掛かる時間を短縮する仕組み

明示管理ヒープ機能では，長寿命オブジェクトのうち，一定期間で不要となるオブジェクトを Explicit ヒープに配置して解放処理で回収することで，FullGC の発生を抑止しています。また，Explicit ヒープに配置したオブジェクトと参照関係にあるオブジェクトは，生存期間が一致しやすいことから，Java ヒープから Explicit メモリブロックへ参照関係に基づいて移動し，Explicit ヒープでまとめて管理する仕組みになっています。

しかし，運用によっては，Explicit メモリブロックへ移動したオブジェクトが，Explicit メモリブロックのサイズを巨大化させ，それが原因で自動解放処理が長時間化することがあります。Explicit メモリブロックの自動解放処理中はシステムが停止するため，自動解放処理の長時間化がシステム上問題となることがあります。巨大なサイズの Explicit メモリブロックのことを巨大ブロックといいます。参照関係に基づいた移動によって，寿命の異なるオブジェクトが一つのブロックに移動し，この繰り返しによってブロックは巨大化します。参照関係が複雑で，アプリケーション開発者が参照関係を把握できないような場合に，巨大ブロックが発生しやすくなります。

### 注意事項

Java オブジェクトには，次の表に示す種類があります。Java オブジェクトの種類によってその寿命は異なり，Explicit ヒープに配置するのが適切なものと適切でないものがあります。

表 7-8　Java オブジェクトの種類

| 項番 | 分類 | オブジェクトの種別 | 解放されるタイミング | 配置に適切なメモリ領域 |
|---|---|---|---|---|
| 1 | 短寿命オブジェクト | リクエスト処理やレスポンス処理など一時的に利用するオブジェクト | CopyGC 発生時 | Java ヒープ（New 領域）※1 |
| 2 | 長寿命オブジェクト | 一定期間で不要となるオブジェクト | 自動解放処理時 | Explicit ヒープ |
| 3 | | アプリケーションの動作に必要でアプリケーションの停止まで使用されるオブジェクト | アプリケーション停止時 | Java ヒープ（Tenured 領域）※2 |

注※1　Explicit ヒープに配置すると，Explicit メモリブロックの生成とその自動解放処理が多発してオーバーヘッドが掛かってしまうため，適切ではありません。

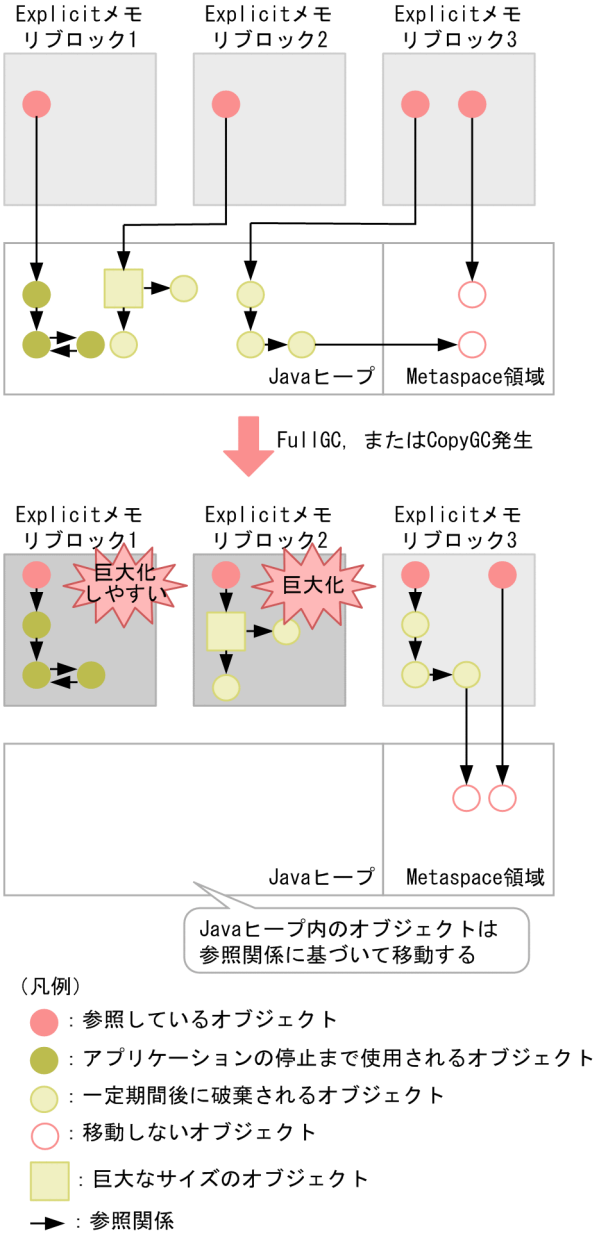
注※2　Explicit ヒープに配置すると，巨大ブロック生成の要因となり，自動解放処理が長時間化してしまうため，適切ではありません。

機能を利用しない場合と利用する場合の比較によって，自動解放処理に掛かる時間を短縮する仕組みを説明します。

## (1) Explicit メモリブロックへのオブジェクト移動制御機能と，明示管理ヒープ機能適用除外クラス指定機能を利用していない場合

Explicit メモリブロックへのオブジェクト移動制御機能と，明示管理ヒープ機能適用除外クラス指定機能を利用しない場合の仕組みについて説明します。オブジェクトを参照関係に基づいて移動する Explicit メモリブロックが三つある場合の例を次の図に示します。

図 7-14　Explicit メモリブロックへのオブジェクト移動制御機能と，明示管理ヒープ機能適用除外クラス指定機能を利用していない場合

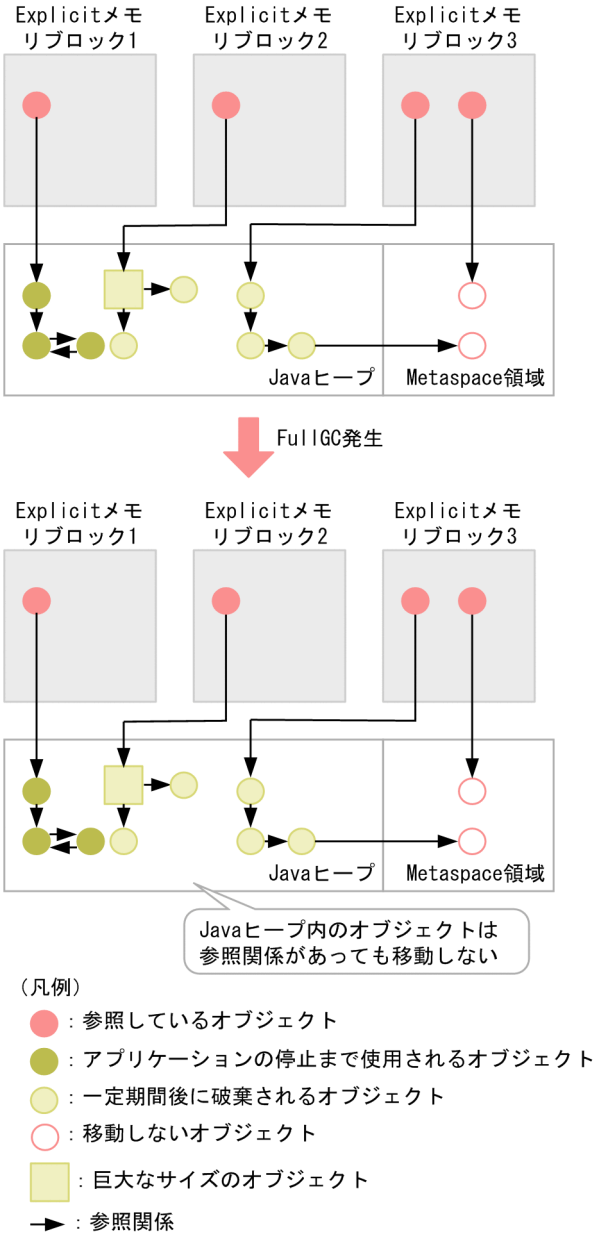



Explicit メモリブロックへのオブジェクト移動制御機能と，明示管理ヒープ機能適用除外クラス指定機能を利用していない場合は，GC が発生したときに，オブジェクトは参照関係に基づいて Java ヒープから Explicit メモリブロックへ移動します。Explicit メモリブロック 1 の場合，移動したオブジェクトがアプリケーションの停止まで使用されるオブジェクトであり，自動解放処理で回収されないため，Explicit メモリブロックに残り続けます。この図に示す段階では，Explicit メモリブロック 1 のオブジェクトの総サイズは巨大でないため，問題はありません。しかし，オブジェクトの参照関係によっては，GC が発生するたびに，参照先のオブジェクトが参照関係に基づいて Explicit メモリブロックへ移動します。この参照関係に基づいた移動がアプリケーション停止時まで続くことで，Explicit メモリブロック 1 は巨大ブロックとなるおそれがあります。Explicit メモリブロック 2 の場合，移動したオブジェクトが巨大サイズであるため，巨大ブロックとなります。このように Explicit ヒープに適切でないオブジェクトが多量に配置されると，Explicit メモリブロックが巨大ブロックとなり，自動解放処理が長時間化します。

## (2) Explicit メモリブロックへのオブジェクト移動制御機能を利用している場合

Explicit メモリブロックへのオブジェクト移動制御機能を利用している場合の仕組みについて説明します。オブジェクトを参照関係に基づいて移動する Explicit メモリブロックが三つある場合の例を次の図に示します。

図 7-15　Explicit メモリブロックへのオブジェクト移動制御機能を利用している場合



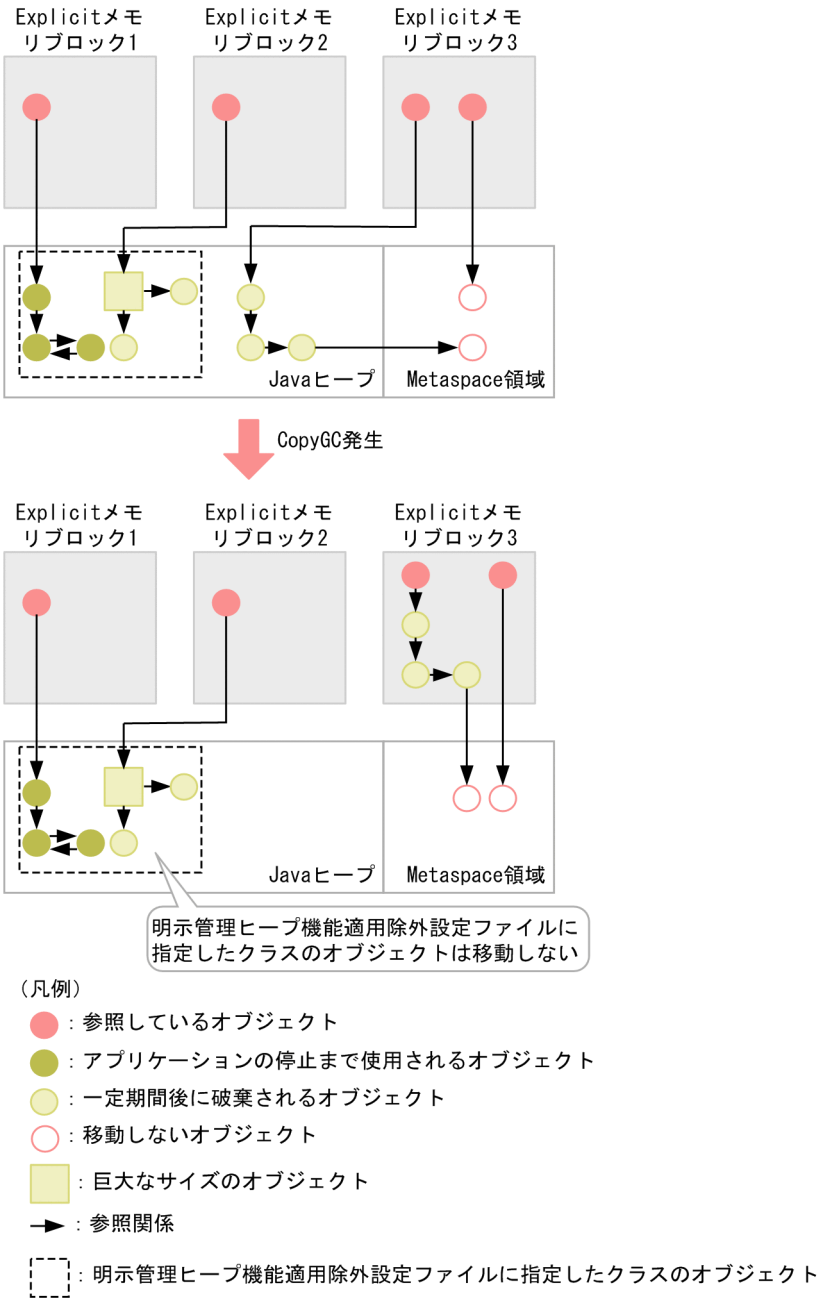

Explicit メモリブロックへのオブジェクト移動制御機能を利用している場合，FullGC が発生しても，オブジェクトは Java ヒープから Explicit メモリブロックへ参照関係に基づいた移動をしません。この機能を利用すると，Java ヒープに配置するオブジェクトが増加するため，Java ヒープ領域のメモリサイズの再設定が必要となる場合もあります。

なお，この機能を利用しても，巨大ブロックが生成される場合は，明示管理ヒープ機能適用除外クラス指定機能を利用します。

### (3) Explicit メモリブロックへのオブジェクト移動制御機能に加えて明示管理ヒープ機能適用除外クラス指定機能を利用している場合

Explicit メモリブロックへのオブジェクト移動制御機能に加えて明示管理ヒープ機能適用除外クラス指定機能を利用している場合の仕組みについて説明します。オブジェクトを参照関係に基づいて移動する Explicit メモリブロックが三つある場合の例を次の図に示します。ここでは，Explicit メモリブロック 1 と Explicit メモリブロック 2 に移動するオブジェクトのクラスが，明示管理ヒープ機能適用外設定ファイルに設定されているとします。

図 7-16　Explicit メモリブロックへのオブジェクト移動制御機能に加えて明示管理ヒープ機能適用除外クラス指定機能を利用している場合




明示管理ヒープ機能適用除外クラス指定機能を利用している場合，明示管理ヒープ機能適用除外設定ファイルに指定したクラスのオブジェクトは，明示管理ヒープ機能の対象から除外され，CopyGC が発生しても，Java ヒープから Explicit メモリブロックへ移動しません。このオブジェクトは，昇格するタイミングで Tenured 領域へ移動します。この機能を利用すると，Java ヒープに配置するオブジェクトが増加するため，Java ヒープ領域のメモリサイズの再設定が必要となる場合もあります。

明示管理ヒープ機能適用除外クラス指定機能で利用する，明示管理ヒープ機能適用除外設定ファイルには，次の 2 種類があります。

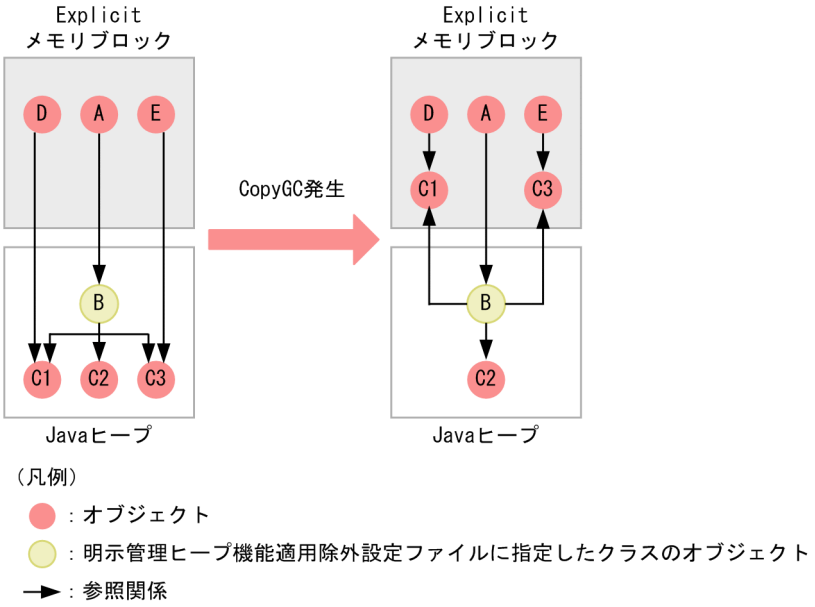
- システムで提供している明示管理ヒープ機能適用除外設定ファイル（sysexmemexcludeclass1100.cfg）

- -XX:ExplicitMemoryExcludeClassListFile オプションにファイルパスを指定した，明示管理ヒープ機能適用除外設定ファイル（exmemexcludeclass.cfg，または任意のファイル名）

明示管理ヒープ機能適用除外クラス指定機能を利用する設定を（-XX:+ExplicitMemoryUseExcludeClass オプションを指定）すると，sysexmemexcludeclass1100.cfg に設定されているクラスは，自動的に明示管理ヒープ機能適用除外対象となり，そのクラスのオブジェクトは Explicit ヒープへ移動しません。さらに適用除外対象のオブジェクトを指定したい場合は，exmemexcludeclass.cfg，または任意のファイル名の明示管理ヒープ機能適用除外設定ファイルに，適用除外対象のオブジェクトのクラスを指定します。また，sysexmemexcludeclass1100.cfg に設定されているクラスのオブジェクトに対して明示管理ヒープ機能を適用したい場合は，明示管理ヒープ機能適用除外無効設定ファイル（exmemnotexcludeclass.cfg）にそのクラスを指定します。このため，設定ファイルに指定するクラスによっては，Explicit ヒープへ移動するオブジェクトを少なくできます。明示管理ヒープ機能適用除外クラスは，スレッドダンプに出力される Explicit ヒープ情報のオブジェクト解放率を参考に指定します。オブジェクト解放率については，「7.10.3　Explicit メモリブロックのオブジェクト解放率情報の利用」を参照してください。

なお，参照経路が明示管理ヒープ機能適用除外対象のオブジェクトを経由するオブジェクトのうち，明示管理ヒープ機能適用除外対象のオブジェクト以外の経路から参照されないオブジェクトも明示管理ヒープ機能適用除外対象となります。明示管理ヒープ機能適用除外対象のオブジェクトからの参照経路が複数ある場合の例を次の図に示します。

図 7-17　明示管理ヒープ機能適用除外対象のオブジェクトからの参照経路が複数ある場合の例




この図の場合，オブジェクト B は明示管理ヒープ機能適用除外対象のオブジェクトです。オブジェクト B を経由する参照経路には，次のものがあります。

- A→B→C1
- A→B→C2
- A→B→C3

このうち，オブジェクト C1 には D→C1，オブジェクト C3 には E→C3 の参照経路があるため，これらのオブジェクトは参照関係に基づいて Explicit メモリブロックへ移動します。一方，オブジェクト C2 はオブジェクト B を経由する以外の参照経路がないため，明示管理ヒープ機能適用除外対象となって Explicit メモリブロックへ移動しません。

## 7.10.3　Explicit メモリブロックのオブジェクト解放率情報の利用

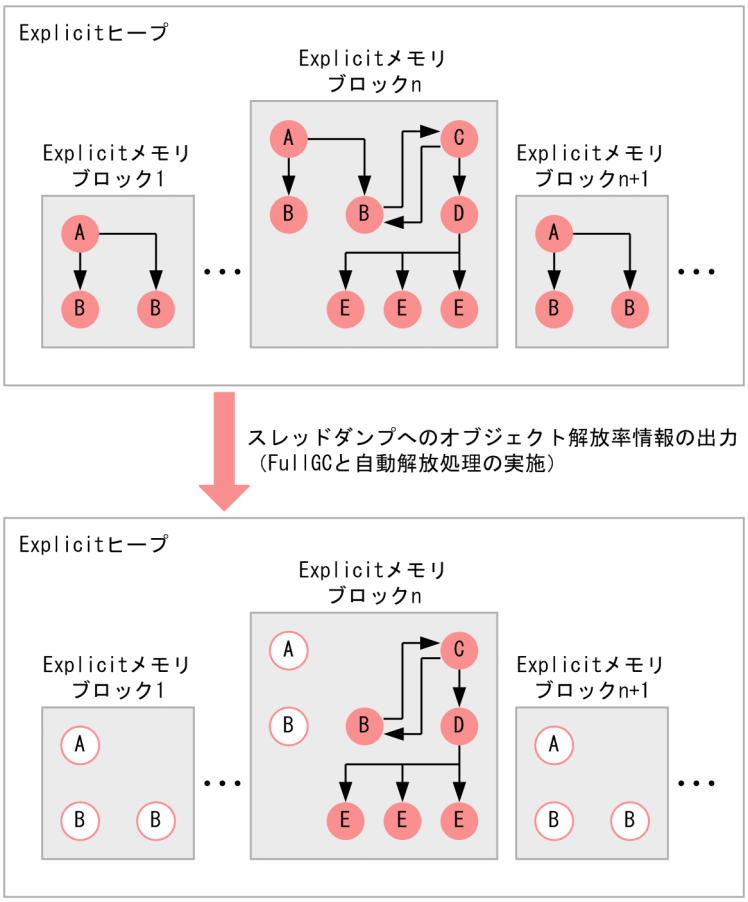
Explicit メモリブロックの自動解放処理が長時間化する要因となる巨大ブロックは，アプリケーションの停止まで使用される長寿命のオブジェクトがユーザプログラムやフレームワークの利用によって生成され，配置されることで生成されます。明示管理ヒープ機能を効果的に使用するためには，この巨大ブロックを生成する要因となるオブジェクトを特定して，Explicit ヒープに配置しないようにすることが必要になります。

### (1)　Explicit メモリブロックのオブジェクト解放率情報の出力

巨大ブロックを生成する要因となるオブジェクトは，オブジェクト解放率情報を利用すると特定できます。オブジェクト解放率情報は，Explicit メモリブロックの自動解放処理で解放されたオブジェクトの割合です。巨大ブロック上で解放率が低いオブジェクトは，巨大ブロックを生成する要因となるオブジェクトであることがわかります。オブジェクト解放率情報は，eheapprof コマンドに-freeratio オプションを指定して実行すると，拡張スレッドダンプの Explicit ヒープ情報に出力できます。この情報を参考にして，そのオブジェクトのクラスを明示管理ヒープ機能適用除外設定ファイルに指定します。明示管理ヒープ機能適用除外設定ファイルへの指定方法については，「7.13.3　設定ファイルを使った明示管理ヒープ機能の適用対象の制御」を参照してください。

オブジェクト解放率情報の出力例を次の図に示します。

図 7-18　オブジェクト解放率情報の出力例




Explicitメモリブロックのオブジェクト解放率情報

| [Block:1] | [Block:n] | [Block:n+1] |
|---|---|---|
| クラスA:100% | クラスA:100% | クラスA:100% |
| クラスB:100% | クラスB: 50% | クラスB:100% |
| | クラスC: 0% | |
| | クラスD: 0% | |
| | クラスE: 0% | |

（凡例）

- x ：クラスxのオブジェクト
- x ：解放されたクラスxのオブジェクト
- ：参照関係

この図に示すように，JavaVM は，オブジェクト解放率情報を求めるために，FullGC と，Explicit メモリブロックの自動解放処理を実行します。これらの処理によって，アプリケーションの実行が数秒間止まるおそれがあるため，オブジェクト解放率情報は，システム開発時や業務停止時間中に出力することをお勧めします。

また，ここで出力されるオブジェクト解放率情報は，情報出力時に発生させた自動解放処理 1 回の結果を基に算出したものです。このため，オブジェクト解放率情報の精度を上げるためには，複数回取得することをお勧めします。

## (2) 実行される内容

eheapprof コマンドに-freeratio オプションを指定して実行すると，JavaVM によって次の処理が実行されます。

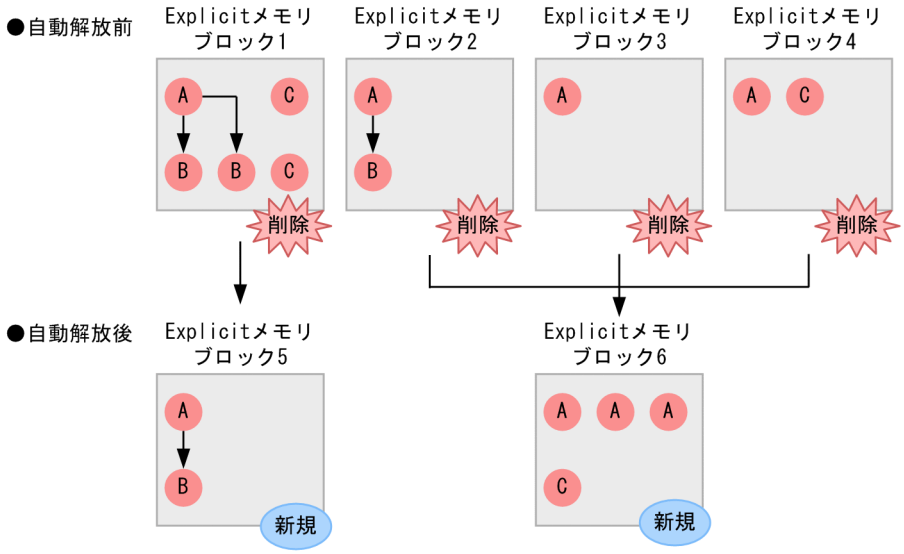
1. FullGC の発生
2. 自動配置機能で生成した Explicit メモリブロック，および eheapprof コマンド実行前に明示解放予約された Explicit メモリブロックに対する自動解放予約

   自動解放予約の対象とならなかった Explicit メモリブロックのオブジェクト解放率には，「-」が出力されます。
3. 2.で自動解放予約された Explicit メモリブロックに対する自動解放処理

   JavaVM は，自動解放処理の前後で Explicit メモリブロック単位にクラスごとのオブジェクト数情報を取得して保持します。
4. 3.で取得した情報から算出したオブジェクト解放率情報の拡張スレッドダンプへの出力

オブジェクト解放率情報の算出例を次の図に示します。

## 図 7-19　オブジェクト解放率情報の算出例




●Explicitメモリブロック5のオブジェクト解放率情報の算出結果

| クラス名 | 自動解放前のオブジェクト数 | 自動解放後のオブジェクト数 | オブジェクト解放率(%) |
|---|---|---|---|
| A | 1 | 1 | 0 |
| B | 2 | 1 | 50 |
| C | 2 | 0 | 100 |

●Explicitメモリブロック6のオブジェクト解放率情報の算出結果

| クラス名 | 自動解放前のオブジェクト数 | 自動解放後のオブジェクト数 | オブジェクト解放率(%) |
|---|---|---|---|
| A | 3 | 3 | 0 |
| B | 1 | 0 | 100 |
| C | 1 | 1 | 0 |

（凡例）

x：クラスxのオブジェクト

→：参照関係

外部（解放対象外の Explicit メモリブロック）から参照されているオブジェクトがある場合は，自動解放処理時に新規の Explicit メモリブロックへ移動します。Explicit メモリブロック 6 のように，自動解放前が複数の Explicit メモリブロックのときは，複数の Explicit メモリブロックのオブジェクト数の合計値と，新規の Explicit メモリブロックのオブジェクト数からオブジェクト解放率を算出します。

なお，eheapprof コマンドの指定方法，および Explicit ヒープ情報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 リファレンス コマンド編」の「eheapprof（Explicit ヒープ詳細情報付き拡張スレッドダンプの出力）」を参照してください。

## 7.10.4　自動解放処理に掛かる時間を短縮する場合の注意事項

ここでは，自動解放処理に掛かる時間を短縮する場合に使用する，Explicit メモリブロックへのオブジェクト移動制御機能，および明示管理ヒープ機能適用除外クラス指定機能使用時の注意事項について説明します。

- Explicit メモリブロックへのオブジェクト移動制御機能は，自動配置機能で作成した Explicit メモリブロックにあるオブジェクトから参照されている Java ヒープのオブジェクトを，FullGC 発生時に Explicit ヒープへ移動しないようにします。この機能を有効にすると，これまで FullGC が発生していたシステムでは，FullGC の発生回数が増加することがあります。FullGC の発生回数がシステム上問題となる場合は，JavaVM で使用する領域のメモリサイズを再度チューニングしてください。メモリチューニングについては，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.　JavaVM のメモリチューニング」を参照してください。
- 明示管理ヒープ機能適用除外クラス指定機能は，自動配置機能で作成した Explicit メモリブロックにあるオブジェクトから参照されている Java ヒープのオブジェクトのうち，設定ファイルに指定されているクラスのオブジェクトを Explicit ヒープに移動しないようにします。この機能を有効にすると，Explicit ヒープへ移動していたオブジェクトが Tenured 領域に移動するため，Tenured 領域の使用量が増加します。このため，FullGC の発生回数が増加するおそれがあります。FullGC の発生回数がシステム上問題になる場合は，JavaVM で使用する領域のメモリサイズを再度チューニングしてください。メモリチューニングについては，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.　JavaVM のメモリチューニング」を参照してください。
- 明示管理ヒープ機能適用除外クラス指定機能は，J2EE サーバまたはバッチサーバ起動時に設定ファイルを読み込みます。そのため，多量のクラスを設定ファイルに指定すると，J2EE サーバまたはバッチサーバの起動時間が長くなるおそれがあります。J2EE サーバまたはバッチサーバの起動時間と，自動解放処理時間などを比較して，この機能の利用を検討してください。
- 明示管理ヒープ機能適用除外クラス指定機能は，クラスロード時に明示管理ヒープ機能適用除外対象かどうかを判定します。そのため，多量のクラスを設定ファイルに指定すると，クラスロード時間が増加するおそれがあります。
- 明示管理ヒープ機能適用除外クラス指定機能は，設定ファイルで指定したクラスのオブジェクトを明示管理ヒープ機能の対象から除外します。そのため，Explicit ヒープに配置すると効果があるオブジェクトのクラスを設定ファイルに指定してしまうと，FullGC 抑止の効果が得られないおそれがあります。
- 明示管理ヒープ機能適用除外クラス指定機能は，「7.6.5　参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動」を実行しないことで，明示管理ヒープ機能適用対象外のオブジェクトとします。このため，アプリケーションでオブジェクトの直接生成を指定した場合，そのオブジェクトのクラスを明示管理ヒープ機能適用除外クラスに指定しても，Explicit メモリブロックに生成されます。アプリケーションでのオブジェクトの直接生成については，「7.6.3　Explicit メモリブロックへのオブジェクトの直接生成」を参照してください。

  なお，HTTP セッションに格納されたオブジェクトを明示管理ヒープ機能適用除外クラスに指定した場合は，明示管理ヒープ機能適用除外対象となり，Explicit メモリブロックへ移動しません。HTTP セッションに格納されたオブジェクトについては，「7.4.1　HTTP セッションに関するオブジェクト」を参照してください。

- アプリケーションのデバック時などに GC を抑止している間は，FullGC も自動解放処理も実行できません。GC を抑止している間に，-freeratio オプションを指定して eheapprof コマンドを実行してもオブジェクト解放率情報が取得できないため，スレッドダンプの Explicit ヒープ詳細情報には，Explicit メモリブロック内のオブジェクト統計情報だけが出力され，オブジェクト解放率情報は出力されません。

## 7.11　HTTP セッションで利用する Explicit ヒープのメモリ使用量の削減

ここでは，HTTP セッションで利用する Explicit ヒープのメモリ使用量を削減する機能について説明します。メモリ使用量の削減には，**HTTP セッションで利用する Explicit ヒープの省メモリ化機能**を使用します。

この機能を使用すると，アプリケーションサーバ内での HTTP セッションと Explicit メモリブロックの関係が多対 1 になります。複数の HTTP セッションで一つの Explicit メモリブロックを共有できるため，Explicit メモリブロックの利用率が向上します。これによって，HTTP セッションが確保する Explicit ヒープのメモリ使用量を削減できます。

### 7.11.1　適用効果があるかの確認

HTTP セッションで利用する Explicit ヒープの省メモリ化機能を適用して効果があるかどうかは，スレッドダンプに含まれる Explicit メモリブロック情報を確認することで判定できます。次の条件を満たす Explicit メモリブロックが多数ある場合は，適用効果がありますので，この機能の利用を検討してください。

- Explicit メモリブロック情報の<EM_TYPE>が「R」である。
- <EM_USED>が，次に示すサイズよりも小さい Explicit メモリブロックが多数ある。

  明示管理ヒープ機能の自動配置機能を利用している場合

  　8KB

  明示管理ヒープ機能の自動配置機能を利用していない場合

  　32KB

スレッドダンプでの Explicit メモリブロック情報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 保守／移行編」の「5.5　JavaVM のスレッドダンプ」を参照してください。

なお，稼働情報による Explicit ヒープ領域のメモリサイズの見積もりについては，この機能を利用するかどうかによる違いはありません。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「7.11.4　稼働情報による見積もり」を参照してください。

ただし，稼働情報ファイルの出力内容については，一部違いがあります。違いについては，「7.11.3　HTTP セッションで利用する Explicit ヒープの省メモリ化機能利用時の注意事項」を参照してください。

### 7.11.2　メモリ使用量を削減する仕組み

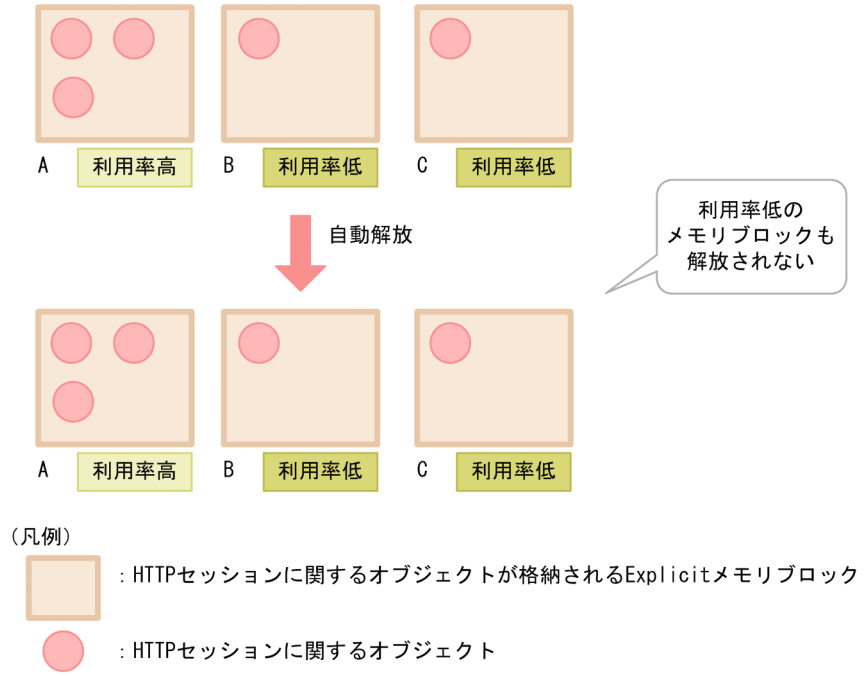
HTTP セッションが格納される Explicit ヒープ領域の Explicit メモリブロックは，アプリケーションが HTTP セッションを作成するたびに作成されます。HTTP セッションに関するオブジェクトは，作成された Explicit メモリブロックに配置されます。

この機能を利用しない場合と利用する場合の比較によって，メモリ使用量を削減する仕組みを説明します。

## (1) HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用しない場合

HTTP セッションに関するオブジェクトが格納される Explicit メモリブロックが三つある場合の例を次の図に示します。このうち，B と C の Explicit メモリブロックは，利用率が低く，長時間利用されていない HTTP セッションに関連づけられた Explicit メモリブロックです。

図 7-20　HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用していない場合



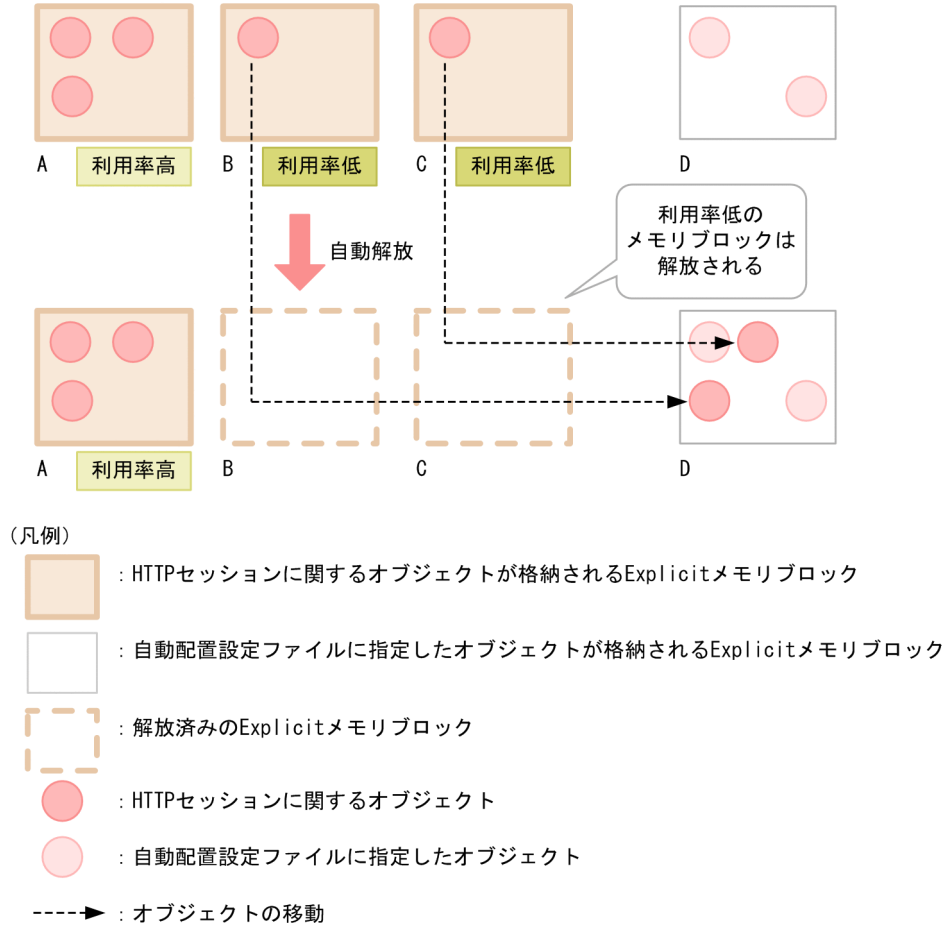

HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用していない場合，自動解放が発生したときも，Explicit メモリブロック A～C は解放されません。この場合，HTTP セッションに関するオブジェクトが格納される Explicit メモリブロックの使用済みサイズは，HTTP セッションに関するオブジェクトの総バイト数に一致します。また，Explicit メモリブロックの個数は有効なセッション数に一致します。

ただし，小さいサイズのオブジェクトを格納する場合も，Explicit メモリブロックは一定サイズ以上の大きさで確保されるため，Explicit ヒープ領域のメモリは，実際に必要な Explicit メモリのサイズ以上に消費されます。

## (2) HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用する場合

HTTP セッションに関するオブジェクトが格納される Explicit メモリブロックが三つある場合の例を次の図に示します。このうち，B と C の Explicit メモリブロックは，利用率が低く，長時間利用されていない HTTP セッションに関連づいた Explicit メモリブロックです。また，自動配置設定ファイルに指定したオブジェクトが格納される Explicit メモリブロックとして，D があります。

図 7-21　HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用している場合




HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用している場合，自動解放が発生したタイミングで，HTTP セッションで利用する Explicit ヒープの省メモリ化機能が実行されます。利用率が低い Explicit メモリブロックである B および C が解放され，この Explicit メモリブロックに格納されていた HTTP セッションに関するオブジェクトは D に移動します。

このように，利用率の低い Explicit メモリブロックに格納されていたオブジェクトをほかの領域に移動して集約して，利用率の低い Explicit メモリブロックを解放することで，Explicit メモリブロックの利用率が向上します。

## 7.11.3　HTTP セッションで利用する Explicit ヒープの省メモリ化機能利用時の注意事項

ここでは，HTTP セッションで利用する Explicit ヒープの省メモリ化機能利用時の注意事項について説明します。

HTTP セッションで利用する Explicit ヒープの省メモリ化機能を利用しているかどうかによって，稼働情報ファイルの出力内容が一部異なります。

**参考**

ここで説明する違いによって，解放される Explicit メモリブロックに格納される HTTP セッションに関するオブジェクトのメモリサイズを計上する領域が，HTTP セッションで利用する領域からアプリケーションで利用する領域に変更になります。ただし，システム全体で使用するメモリサイズには変更ありません。このため，この機能を利用するかどうかは，稼働情報を使用した Explicit ヒープ領域のメモリサイズの見積もりには影響ありません。

出力内容が異なる項目について説明します。

## (1) HTTP セッションで取得した Explicit メモリブロックの個数

次の項目に出力される Explicit メモリブロックの個数が異なります。

- HTTPSessionEMemoryBlockCount.HighWaterMark
- HTTPSessionEMemoryBlockCount.LowWaterMark
- HTTPSessionEMemoryBlockCount.Current

**この機能を利用していない場合**

システムで有効な HTTP セッション数が出力されます。

**この機能を利用している場合**

内部動作を反映した値が出力されるため，システムで有効な HTTP セッション数とは異なる値が出力されます。

## (2) アプリケーションで利用する Explicit ヒープ領域のサイズ

次の項目に出力される Explicit ヒープ領域のサイズが異なります。

- ApplicationEHeapSize.HighWaterMark
- ApplicationEHeapSize.LowWaterMark

**この機能を利用していない場合**

自動配置機能で利用される Explicit メモリのサイズが出力されます。

**この機能を利用している場合**

「この機能が自動解放対象とした Explicit メモリのサイズ＋自動配置機能で利用する Explicit メモリのサイズ」の合計サイズとなります。

## 7.12　明示管理ヒープ機能 API を使った Java プログラムの実装

この節では，アプリケーションで明示管理ヒープ機能を使用する場合の実装について説明します。明示管理ヒープ機能は，明示管理ヒープ機能 API を使用して実装します。

明示管理ヒープ機能は，JP.co.Hitachi.soft.jvm.MemoryArea パッケージ内のクラスで使用できます。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。なお，明示管理ヒープ機能 API は，すべてスレッドセーフです。

明示管理ヒープ機能 API では，次の 2 種類の処理を実装します。

- オブジェクトを Explicit ヒープに配置するための実装
- 明示管理ヒープ機能の稼働情報を取得するための実装

### 7.12.1　オブジェクトを Explicit ヒープに配置するための実装

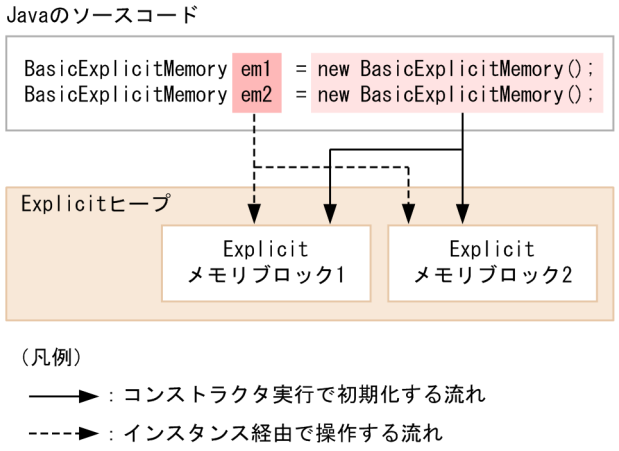
ここでは，明示管理ヒープ機能 API のクラスの概要と，API の基本的な利用方法について説明します。

#### (1)　ExplicitMemory インスタンスと Explicit メモリブロックの関係

Explicit ヒープ内の Explicit メモリブロックは，明示管理ヒープ機能 API で扱う ExplicitMemory クラスのインスタンスと 1：1 で対応します。

ExplicitMemory インスタンスと Explicit メモリブロックの関係を次の図に示します。

図 7-22　ExplicitMemory インスタンスと Explicit メモリブロックの関係




Explicit メモリブロックは，ExplicitMemory クラスのコンストラクタを実行すると初期化されます。以降は，初期化されたインスタンスが，Explicit メモリブロックを操作するためのインタフェースになります。図の場合は，インスタンス em1 が Explicit メモリブロック 1 に，インスタンス em2 が Explicit メモリブロック 2 に対応しています。

## (2) 明示管理ヒープ機能 API のクラス構成

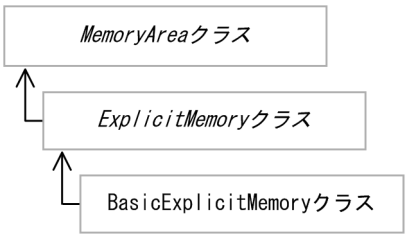
明示管理ヒープ機能 API のクラスを次の表に示します。

### 表 7-9 明示管理ヒープ機能 API のクラス

| クラス | 説明 |
|---|---|
| MemoryArea クラス | Java ヒープまたは Explicit メモリブロックを表現する抽象クラスです。 |
| ExplicitMemory クラス | Explicit メモリブロックを表現する抽象クラスです。このクラスの機能は，派生クラスである BasicExplicitMemory クラスを介して利用します。 |
| BasicExplicitMemory クラス | アプリケーションで扱う Explicit メモリブロックを表現するクラスです。アプリケーションでは，このクラスの API を使用して明示管理ヒープ機能を実装します。 |

クラスの階層を次の図に示します。

### 図 7-23 明示管理ヒープ機能 API のクラス階層




注 *斜体*は，抽象クラスを示します。

## (3) 明示管理ヒープ機能 API の利用方法

基本的な操作とメソッドの対応は次のとおりです。

- Explicit メモリブロックへのオブジェクトの直接生成
  newArray メソッドまたは newInstance メソッドを利用します。
- Explicit メモリブロックの解放
  reclaim メソッドを使用します。

次に，BasicExplicitMemory クラスの利用例を示します。この例では，二つの Explicit メモリブロックを作成します。

### BasicExplicitMemory クラスの利用例

| 行 | Java プログラム |
|---|---|
| 01 | BasicExplicitMemory em1 = new BasicExplicitMemory(); |
| 02 | BasicExplicitMemory em2 = new BasicExplicitMemory(); |
| 03 | Object o1 = em1.newInstance(Object.class); |
| 04 | Object o2 = em2.newInstance(Object.class); |
| 05 | ExplicitMemory.reclaim(em1); |

| 行 | Java プログラム |
|---|---|
| 06<br>07<br>08 | |

01 行目と 02 行目では，BasicExplicitMemory インスタンスを生成しています。この例では，em1 が Explicit メモリブロック 1，em1 が Explicit メモリブロック 2 に対応するものとします。

04 行目および 06 行目の newInstance メソッドによって，Explicit メモリブロックにオブジェクトを直接生成します。04 行目では，em1 インスタンスから newInstance メソッドを呼び出すことで，Object クラスのオブジェクトを Explicit メモリブロック 1 に配置します。06 行目では，em2 インスタンスから newInstance メソッドを呼び出すことで，Object クラスのオブジェクトを Explicit メモリブロック 2 に配置します。

Explicit メモリブロックが不要になったら，Explicit メモリブロックを破棄します。08 行目の ExplicitMemory.reclaim(em1)メソッドの実行は，Explicit メモリブロック 1 を解放するための処理です。これによって，Explicit メモリブロック 1 が解放され，同時に 04 行目で生成したオブジェクト o1 も破棄されます。なお，Explicit メモリブロックの解放は，オブジェクト単位ではなく，該当の Explicit メモリブロックに対応する領域全体が対象になります。

この例の場合，Explicit メモリブロック 1 の生存期間は，01 行目から 08 行目になります。

## 7.12.2　明示管理ヒープ機能の稼働情報を取得するための実装

ここでは，アプリケーションで明示管理ヒープ機能の稼働情報を取得するための実装について説明します。稼働情報を取得することで，デバッグや障害解析を実行できます。

アプリケーションで明示管理ヒープ機能を実装している場合，稼働情報として次の情報を取得できます。

- Explicit ヒープの利用状況
- ExplicitMemory インスタンスが表す Explicit メモリブロックサイズ
- Explicit メモリブロックの情報

また，稼働情報の取得に関連した処理として，次の処理も実行できます。

- Explicit メモリブロックの名前の設定と取得
- Explicit メモリブロックの処理可能状態の判定
- Explicit メモリブロックの解放予約状態の判定

ここでは，明示管理ヒープ機能 API を使用した，それぞれの処理の実装について説明します。

## (1) Explicit ヒープの利用状況の取得

Explicit ヒープの情報を取得方法について説明します。Explicit ヒープは，Explicit メモリブロック全体のことです。なお，各 Explicit メモリブロックの情報を取得する方法については，「7.12.2(3) Explicit メモリブロックの情報の取得」を参照してください。

### 使用する API

getMemoryUsage()

この API は，java.lang.management.MemoryUsage クラスのインスタンスを作成して，そのインスタンスを返却します。

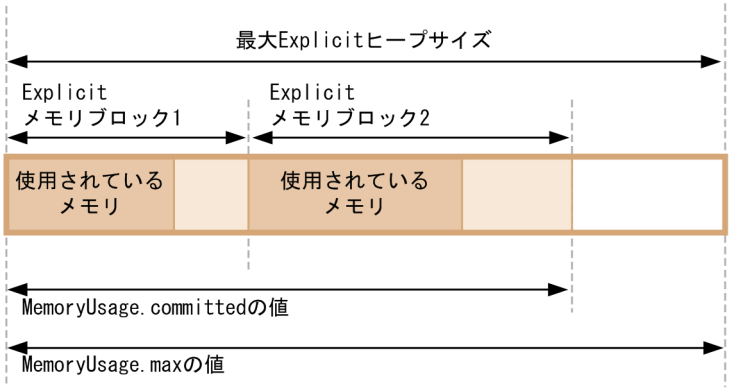
返却されたインスタンスの各フィールドには，インスタンス作成時の情報として，次の表に示す情報が設定されています。

### 表 7-10 各フィールドの情報（MemoryUsage クラスのインスタンス）

| フィールド | 設定内容 |
|---|---|
| init | 0 |
| used | Explicit ヒープの使用されているメモリのサイズ（単位：バイト） |
| committed | Explicit ヒープの確保済みサイズ（単位：バイト） |
| max | -XX:HitachiExplicitHeapMaxSize で指定した最大 Explicit ヒープサイズの値（単位：バイト）<br>ただし，明示管理ヒープ機能がオフの場合（-XX:-HitachiUseExplicitMemory が設定されている場合）は，0 になります。 |

各フィールドが示す値を次の図に示します。

### 図 7-24 各フィールドが示す値（MemoryUsage クラスのインスタンス）




注 MemoryUsage.usedの値は，「使用されているメモリ」の合計値になります。

## (2) ExplicitMemory インスタンスが表す Explicit メモリブロックサイズ

Explicit メモリブロックの利用状況として，ExplicitMemory インスタンスが表す Explicit メモリブロックサイズを取得します。これによって明示管理ヒープ機能でのメモリの使用状況をチェックできます。

使用する API

- freeMemory()
- usedMemory()
- totalMemory()

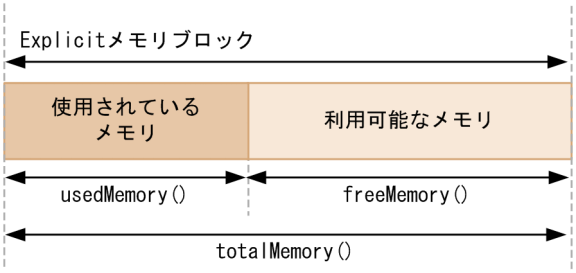
それぞれの API で取得できる Explicit メモリブロックの利用状況を次の表に示します。なお，サイズは，long 型の値として取得できます。

表 7-11　各 API で取得できる Explicit メモリブロックの利用状況

| API | 取得できる Explicit メモリブロックの利用状況 |
|---|---|
| freeMemory() | メモリの利用可能なサイズ（単位：バイト） |
| usedMemory() | 使用されているメモリのサイズ（単位：バイト） |
| totalMemory() | 確保済み総サイズ（単位：バイト） |

各 API で取得できる値を次の図に示します。

図 7-25　各 API で取得できる値




## (3)　Explicit メモリブロックの情報の取得

Explicit ヒープに実体がある Explicit メモリブロックの個数を取得します。状態が解放済みまたは無効になっている Explicit メモリブロックは対象になりません。有効な Explicit メモリブロックの個数を取得すると，各 Explicit メモリブロックで使用されているメモリの平均サイズなどが算出できるようになります。

使用する API

countExplicitMemories()

この API は，Explicit ヒープにあるメモリブロックの数を数えて，int 型の値として返却します。数える対象になるのは，次の条件を満たしている Explicit メモリブロックです。

- 有効な Explicit メモリブロック
  サブ状態が「Enable」または「Disable」のどちらの場合も対象になります。
- 解放予約済みの Explicit メモリブロック

## (4) Explicit メモリブロックの名称の設定と取得

Explicit メモリブロックに対応するインスタンスに名称を設定したり，設定されている名称を取得したりできます。Explicit メモリブロックのインスタンスは，アプリケーションで扱いやすいように，名称を持っています。任意の名称を設定することで，インスタンスを利用しやすくなります。

設定した値は，明示管理ヒープ機能のイベントログにも出力されます。

### 使用する API

- setName()
  名前を設定します。
- getName()
  名前を取得します。

なお，ユーザのアプリケーションで名前を設定しない場合，次に示すデフォルトの名前が設定されています。

```
BasicExplicitMemory-<ID>
```

<ID>は，JavaVM で管理している値です。

**注意事項**

Explicit メモリブロックの名称として，「CCC#」で始まる名前は付けないでください。「CCC#」で始まる名称は，J2EE サーバが使用します。

J2EE サーバで使用する Explicit メモリブロックの名称は次のとおりです。

- CCC#HttpSession
  HTTP セッションを配置する Explicit メモリブロックです。
- CCC#HttpSessionManager
  HTTP セッション管理用オブジェクトを配置する Explicit メモリブロックです。

## (5) Explicit メモリブロックの処理可能状態の判定

Explicit メモリブロックは，メモリ確保に失敗した場合などに処理不能な状態になることがあります。Explicit メモリブロックが処理可能な状態かどうかを判定できます。

### 使用する API

isActive()

API を呼び出したときの Explicit メモリブロック（ExplicitMemory インスタンス）の状態と，API の戻り値の対応を次の表に示します。

表 7-12　isActive()を呼び出したときの Explicit メモリブロックの状態と API の戻り値の対応

| Explicit メモリブロックの状態 | サブ状態 | 戻り値 |
|---|---|---|
| 解放済み | − | false |
| 無効 | − | false |
| 解放予約済み | − | false |
| 有効 | Enable | true |
| | Disable | false |

（凡例）

−：該当しません。

## (6)　Explicit メモリブロックの解放予約状態の判定

Explicit メモリブロックが解放予約状態や解放済み状態になったあとでも，その Explicit メモリブロックに対応する ExplicitMemory インスタンスは参照できます。API を使用することで，アプリケーションから，Explicit メモリインスタンスの状態を確認できます。

### 使用する API

isReclaimed()

API を呼び出したときの Explicit メモリブロック（ExplicitMemory インスタンス）の状態と，API の戻り値の対応を次の表に示します。

表 7-13　isReclaimed()を呼び出したときの Explicit メモリブロックの状態と API の戻り値の対応

| Explicit メモリブロックの状態 | サブ状態 | 戻り値 |
|---|---|---|
| 解放済み | − | true |
| 無効 | − | true |
| 解放予約済み | − | true |
| 有効 | Enable | false |
| | Disable | false |

（凡例）

−：該当しません。

## 7.13　実行環境での設定

この節では，明示管理ヒープ機能を利用するための実行環境での設定について説明します。

**注意事項**

J2EE サーバでは，デフォルトで HTTP セッションに関するオブジェクトを Explicit ヒープ領域に配置する設定になっています。

運用を開始する前に，必要な Explicit ヒープサイズを見積もり，JavaVM オプション（-XX:HitachiExplicitHeapMaxSize オプション）を適切な値にチューニングしてください。Explicit ヒープのチューニングについては，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「7.3.2　チューニング手順」を参照してください。

### 7.13.1　明示管理ヒープ機能を利用するための共通の設定（JavaVM オプションの設定）

ここでは，明示管理ヒープ機能を利用するための共通の設定について説明します。

明示管理ヒープ機能を利用する場合，次の設定が必要です。

- J2EE サーバ
- バッチサーバ
- Java アプリケーション
- 自動配置設定ファイルの設定

また，明示管理ヒープ機能の適用対象を制御する場合，次の設定が必要です。

- 明示管理ヒープ機能適用除外設定ファイルの設定

#### (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。

明示管理ヒープ機能を使用するための共通の設定は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の JavaVM 起動パラメタ（add.jvm.arg）で JavaVM のオプションを指定します。

明示管理ヒープ機能の JavaVM オプションの定義について次の表に示します。

## 表 7-14　明示管理ヒープ機能の JavaVM オプションの定義

| JavaVM オプション | 設定内容 | デフォルト値 |
|---|---|---|
| -XX:[+\|-]HitachiUseExplicitMemory | 明示管理ヒープ機能を使用するかどうかを設定します。 | 新規インストールの場合<br>実行環境によって異なります※1。<br>バージョンアップの場合<br>-XX:-HitachiUseExplicitMemory |
| -XX:[+\|-]HitachiAutoExplicitMemory | 明示管理ヒープ機能の自動配置機能を使用するかどうかを設定します。 | -XX:-HitachiAutoExplicitMemory |
| -XX:HitachiAutoExplicitMemoryFile:<文字列> | 明示管理ヒープ機能の自動配置機能を使用する場合に利用する，自動配置設定ファイルのパスを指定します。 | 空文字 |
| -XX:[+\|-]HitachiExplicitMemoryAutoReclaim | 明示管理ヒープ機能の自動解放機能を使用するかどうかを指定します。 | -XX:+HitachiExplicitMemoryAutoReclaim |
| -XX:[+\|-]HitachiExplicitMemoryCompatibleToV8 | Explicit メモリブロックを確保する方法を，08-00 と同様にするかどうか指定します。08-50 以降の新機能を利用しないで，08-00 で動作するアプリケーションをそのまま 08-50 で動作させる場合，このオプションを有効にします。 | -XX:-HitachiExplicitMemoryCompatibleToV8 |
| -XX:HitachiExplicitHeapMaxSize※2 | Explicit ヒープ領域の最大サイズを設定します。（単位：バイト） | -XX:HitachiExplicitHeapMaxSize=64m |
| -XX:HitachiExplicitMemoryLogLevel:<文字列> | <文字列>に明示管理ヒープ機能で出力するイベントログのログレベルを設定します。<br>次のどれかを指定します。<br>• none<br>• normal<br>• verbose<br>• debug | -XX:HitachiExplicitMemoryLogLevel:normal |
| -XX:HitachiExplicitMemoryJavaLog:<文字列> | <文字列>に明示管理ヒープ機能で出力するイベントログの出力先パス名を指定します。 | Windows の場合<br>-XX:HitachiExplicitMemoryJavaLog:<Application Server のインストールディレクトリ>¥CC¥server¥public¥ejb¥<サーバ名>¥logs<br>UNIX の場合<br>-XX:HitachiExplicitMemoryJavaLog:/opt/ |

| JavaVM オプション | 設定内容 | デフォルト値 |
|---|---|---|
| | | Cosminexus/CC/server/public/ejb/<サーバ名>/logs |
| -XX:HitachiExplicitMemoryJavaLogFileSize=<整数値> | <整数値>にイベントログのファイルサイズを指定します。（単位：バイト） | -XX:HitachiExplicitMemoryJavaLogFileSize =4m |
| -XX:ExplicitMemoryFullGCPolicy=<数値> | FullGC 発生時に，参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動を制御するかどうかを指定します。 | -XX:ExplicitMemoryFullGCPolicy=0 |
| -XX:[+\|-]ExplicitMemoryUseExcludeClass | 明示管理ヒープ機能適用除外クラス指定機能を有効にするかどうかを指定します。 | 推奨モードの場合<br>-XX:+ExplicitMemoryUseExcludeClass |
| -XX:ExplicitMemoryExcludeClassListFile:<文字列> | 明示管理ヒープ機能適用除外クラス指定機能を使用する場合に利用する，明示管理ヒープ機能適用除外設定ファイルのパスを指定します。 | 空文字 |
| -XX:ExplicitMemoryNotExcludeClassListFile:<文字列> | 明示管理ヒープ機能適用除外クラス指定機能を使用する場合に利用する，明示管理ヒープ機能適用除外無効設定ファイルのパスを指定します。 | 空文字 |

注※1

新規インストールの場合，実行環境によってデフォルト値は次のように異なります。

J2EE サーバの場合

-XX:+HitachiUseExplicitMemory

バッチサーバおよび Java アプリケーションの場合

-XX:-HitachiUseExplicitMemory

注※2

見積もりについては，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　Explicit ヒープのチューニング」を参照してください。

簡易構築定義ファイルでの定義例を次に示します。

```
<param-name>add.jvm.arg</param-name>
 <param-value>-Xms512m</param-value>
 <param-value>-Xmx512m</param-value>
 <param-value>-XX:+HitachiUseExplicitMemory</param-value>
 <param-value>-XX:HitachiExplicitHeapMaxSize=64m</param-value>
```

**参考**

J2EE サーバの設定は，運用管理ポータルの［起動パラメタの設定］画面（論理 J2EE サーバの定義）でも設定できます。運用管理ポータルで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 運用管理ポータル操作ガイド」の「10.8.23　起動パラメタの設定（J2EE サーバ)」を参照してください。

指定する JavaVM オプション，および簡易構築定義ファイルで指定するタグの詳細は，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。

## (2)　バッチサーバの設定

バッチサーバの設定は，簡易構築定義ファイルで実施します。

明示管理ヒープ機能を使用するための共通の設定は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の JavaVM 起動パラメタ（add.jvm.arg）で JavaVM のオプションを指定します。

指定する JavaVM のオプションは，「(1)　J2EE サーバの設定」を参照してください。

## (3)　Java アプリケーションの設定

cjclstartap コマンドで動作させる Java アプリケーションの設定は，Java アプリケーション用オプション定義ファイル（usrconf.cfg）で実施します。

明示管理ヒープ機能を使用するための共通の設定は，Java アプリケーション用オプション定義ファイル（usrconf.cfg）の JavaVM 起動パラメタ（add.jvm.arg）で JavaVM のオプションを指定します。

指定する JavaVM のオプションは，「(1)　J2EE サーバの設定」を参照してください。

Java アプリケーション用オプション定義ファイル（usrconf.cfg）での定義例を次に示します。

```
add.jvm.arg=-Xms512m
add.jvm.arg=-Xmx512m

add.jvm.arg=-XX:+HitachiUseExplicitMemory
add.jvm.arg=-XX:HitachiExplicitHeapMaxSize=64m
```

Java アプリケーション用オプション定義ファイル（usrconf.cfg）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「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル)」を参照してください。

## (4)　自動配置設定ファイルの設定

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利用する場合，-XX:+HitachiAutoExplicitMemory オプションを指定し，Explicit ヒープに配置したいオブジェクトの設定が必要です。

Explicit ヒープに配置したいオブジェクトの設定は，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 AutoExplicitMemoryText パラメタで指定します。

定義例を次に示します。

```
   :
<param>
<param-name>AutoExplicitMemoryText</param-name>
<param-value>
<![CDATA[
com.sample.*, java.util.ArrayList
com.sample.Main.main(java.lang.String[]), java.util.LinkedList
]]>
</param-value>
</param>
   :
```

自動配置設定ファイルの作成方法については，「7.13.2　自動配置設定ファイルを使った明示管理ヒープ機能の使用」を参照してください。

自動配置設定ファイルは，運用管理ポータルの［起動パラメタの設定］画面（論理 J2EE サーバの定義）またはユーザ任意のファイル（-XX:HitachiAutoExplicitMemoryFile プロパティで指定したファイル）でも設定できます。

## (5)　明示管理ヒープ機能適用除外設定ファイルの設定

明示管理ヒープ機能適用除外設定ファイルを使用し，参照関係に基づく移動の対象となるオブジェクトに対して明示管理ヒープ機能の適用を制御する場合は，次のオプションの指定と，Explicit ヒープに移動しないクラスの設定が必要です。

- -XX:ExplicitMemoryFullGCPolicy=0
- -XX:+ExplicitMemoryUseExcludeClass

Explicit ヒープに移動しないクラスの設定は，明示管理ヒープ機能適用除外設定ファイルに記述します。

記述例を次に示します。

```
com.sample.ClassA
com.sample.ClassB
```

また，明示管理ヒープ機能適用除外設定ファイルに記述しているクラスのうち，一部のクラスを Explicit ヒープに移動したい場合は，明示管理ヒープ機能適用除外無効設定ファイルに，明示管理ヒープ機能適用除外の設定を無効にしたいクラスを記述します。

明示管理ヒープ機能適用除外設定ファイル，および明示管理ヒープ機能適用除外無効設定ファイルの作成方法については，「7.13.3　設定ファイルを使った明示管理ヒープ機能の適用対象の制御」を参照してください。

## 7.13.2　自動配置設定ファイルを使った明示管理ヒープ機能の使用

明示管理ヒープ機能の自動配置機能は，自動配置設定ファイルを使って設定します。自動配置設定ファイルを使用することで，Java プログラムを変更することなく明示管理ヒープ機能を使用できます。

自動配置設定ファイルには，Explicit ヒープに配置したいオブジェクト，およびオブジェクトを生成する場所を指定します。なお，このファイルに指定したオブジェクト（Explicit メモリブロックに生成されたオブジェクト）から参照されているオブジェクトは，Java ヒープから Explicit メモリブロックへ移動します。オブジェクトの移動については「7.6.5　参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動」を参照してください。

-XX:+HitachiAutoExplicitMemory オプションを指定して，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利用する場合の，自動配置設定ファイルの記述形式および記述する際の注意事項について説明します。

自動配置設定ファイルの内容は次のどれかに記述できます。

- 簡易構築定義ファイル
- 運用管理ポータルの［起動パラメタの設定］画面（論理 J2EE サーバの定義）
- ユーザ任意のファイル（jvm.userprf.File プロパティで指定したファイル）

### (1)　自動配置設定ファイルの記述形式

自動配置設定ファイルの記述形式を次に示します。

```
〈生成点〉※，〈指定したオブジェクトの完全修飾クラス名〉 # コメント
 :
〈生成点〉※，〈指定したオブジェクトの完全修飾クラス名〉
```

注※

生成点の指定例を次に示します。

| 生成点の指定例 | 意味 |
|---|---|
| * | すべてのパッケージのすべてのクラスに含まれる，すべてのメソッドでの，ユーザ指定オブジェクトの生成を生成点として指定します。 |
| com.sample.* | com.sample で始まるすべてのパッケージのクラスに含まれるメソッドでの，ユーザ指定オブジェクトの生成を生成点として指定します。<br>そのため，下位のパッケージ（com.sample.abc，または com.sample.abc.test）が存在する場合は，これらのパッケージも対象となります。 |
| com.sample.Main | com.sample.Main クラスに含まれるすべてのメソッド（コンストラクタ，および静的初期化子を含む）でのユーザ指定オブジェクトの生成を生成点として指定します。 |

| 生成点の指定例 | 意味 |
|---|---|
| com.sample.Main.main(java.lang.String[]) | com.sample.Main クラスで定義された main(java.lang.String[])メソッドでのユーザ指定オブジェクトの生成を生成点として指定します。 |

**ポイント**

- 構文要素を区切る空白文字は，半角スペース文字（「0x20」）またはタブ文字(「¥t」もしくは「0x09」)となります。
- 行末は改行文字（「¥n」もしくは「0x0A」）または復帰文字（「¥r」もしくは「0x0D」）が 1 文字以上続いたものとなります。
- コメントは「#」で始まり，「#」から行末までの間の文字すべてをコメントとします。
- 生成点での文字「*」は，同一またはサブパッケージに存在するすべてのクラスを表します。サブパッケージのクラスも対象とする点で，Java 言語の import 宣言の「*」と生成点の「*」は意味が異なります。

## (2) 自動配置設定ファイルの記述例

自動設定ファイルの記述例を次に示します。

```
# comment
com.sample.*, java.util.ArrayList # comment
com.sample.Main.main(java.lang.String[]), java.util.LinkedList
```

記述例の内容について説明します。

1. 1 行目はすべてコメントとなります。
2. com.sample.* で始まるすべてのパッケージに含まれる，クラス，およびメソッドで生成される java.util.ArrayList オブジェクトを，Explicit メモリブロックに配置するように指定します。「#」から行末まではコメントとします。
3. com.sample.Main.main(java.lang.String[])メソッドで生成される java.util. LinkedList オブジェクトを Explicit メモリブロックに配置するように指定します。

**参考**

JavaVM 内のクラス（例：java, javax で始まるパッケージのクラス）をユーザ指定オブジェクトの生成点として指定したエントリを記述できます。しかし，指定したエントリが存在しないものとして扱われることがあります。存在しないものとして扱われた場合は，明示管理ヒープログへエラーメッセージは出力されません。

## (3) 自動配置設定ファイルの注意事項

自動配置設定ファイルを指定する場合の注意事項を次に示します。

- 自動配置機能を使用することで，クラスローディング時間が増加し，その結果 JavaVM の起動時間やアプリケーションサーバでのアプリケーションのデプロイ時間が増加する場合があります。
- 自動配置機能を使用することで，CopyGC の処理に時間が掛かる場合があります。
- 自動配置機能の対象となるオブジェクトは，new で生成しているオブジェクトだけです。JNI やリフレクションで生成しているオブジェクトは対象になりません。
- クラス名，およびメソッドの引数は，java.lang パッケージのクラスも含め，すべて完全修飾クラス名で記述してください。

  （例）

  誤：String

  正：java.lang.String
- ジェネリックス（総称）を用いたクラス名は記述できません。パラメタ化されていないクラス名（raw 型）を記述してください。

  （例）

  誤：java.util.HashMap<java.lang.String, java.lang.Object>

  正：java.util.HashMap
- ネストしたクラスは，「.」ではなく「$」で区切った名前を記述してください。

  （例）

  誤：java.util.AbstractMap.SimpleEntry

  正：java.util.AbstractMap$SimpleEntry
- コンストラクタは，クラス名と同じメソッド名，または<init>と記述してください。MyMain クラスのコンストラクタの場合は次のように記述してください。

  （例）

  MyMain.MyMain()または MyMain.<init>()
- クラス名と同じ名前のメソッドが存在する場合，コンストラクタを指定しているのか，メソッドを指定しているのか判別できません。そのため，コンストラクタ，およびメソッドの両方を指定したものとして扱われます。

  （例）

  MyMain.MyMain(int) # MyMain クラスの int 引数を持つコンストラクタと# MyMain(int)メソッドの両方を生成点とする
- 静的初期化子は，<clinit>と記述してください。MyMain クラスの静的初期化子の場合，次のように記述します。

  （例）

  MyMain.<clinit>()
- フィールド宣言時の代入によるオブジェクトの生成を生成点に指定する場合，生成点にデフォルトコンストラクタを記述します。
- ユーザ指定オブジェクトの完全修飾クラス名に配列を指定することはできません。

（例）

java.lang.String[]

- 存在しないクラス名，メソッド名，およびバイトコードを持たないメソッド(native メソッドおよび abstract メソッド)を含む行が存在する場合，その行は存在しないものとして扱います。
- ユーザ指定オブジェクトのクラス名に J2SE の内部クラスを指定した場合，明示管理ヒープ機能が適切なクラス名に読み替えることがあります。例えば，java.util.HashMap$Entry を java.util.HashMap に読み替えます。
- 生成点として Java 言語仕様の限界に近い巨大なクラスやメソッドを指定した場合，自動配置に失敗する場合があります。この場合，明示管理ヒープ機能のイベントログの明示管理ヒープ自動配置エラーの<MESSAGE>として，"Invalid class file format"と出力されます。このような場合は，クラスやメソッドを小さくすることを検討してください。

## 7.13.3　設定ファイルを使った明示管理ヒープ機能の適用対象の制御

自動配置機能で作成した Explicit メモリブロックにあるオブジェクトから参照されているオブジェクトは，GC 発生時に Java ヒープから Explicit ヒープへ参照関係に基づいて移動します。明示管理ヒープ機能適用除外クラス指定機能は，設定ファイルを使って，この参照関係に基づく移動の対象となるオブジェクトを明示管理ヒープ機能の適用対象から除外し，Explicit ヒープへ移動させないようにします。この機能を使用すると，アプリケーションの停止まで使用されるオブジェクトなど，FullGC でも回収されないオブジェクトを，明示管理ヒープ機能の適用対象から除外できます。オブジェクトの参照関係に基づいた移動については「7.6.5　参照関係に基づくオブジェクトの Java ヒープから Explicit メモリブロックへの移動」を参照してください。

### (1)　設定ファイルの種類

明示管理ヒープ機能適用除外クラス指定機能で使用するファイルには，次の 2 種類があります。

- **明示管理ヒープ機能適用除外設定ファイル**

  Explicit ヒープへ移動させたくないオブジェクトのクラスを指定します。このファイルに指定したクラスのオブジェクトは，GC が発生しても，Explicit ヒープへ移動しません。昇格するタイミングで Tenured 領域へ移動します。

  明示管理ヒープ機能適用除外設定ファイルには，システムで提供しているファイルがあります。明示管理ヒープ機能適用除外クラス指定機能を有効にすると，システムで提供している明示管理ヒープ機能適用除外設定ファイルが使用されます。システムで提供している明示管理ヒープ機能適用除外設定ファイルのファイルパスを次に示します。

  Windows の場合

  <JDK インストールディレクトリ>¥lib¥explicitmemory¥sysexmemexcludeclass1100.cfg

  UNIX の場合

  /opt/Cosminexus/jdk/lib/explicitmemory/sysexmemexcludeclass1100.cfg

明示管理ヒープ機能の適用対象から除外するクラスを追加したい場合は，次のファイルパスにあるファイルを更新するか，または新たなファイルを作成してください。

Windows の場合

<JDK インストールディレクトリ>¥usrconf¥exmemexcludeclass.cfg

UNIX の場合

/opt/Cosminexus/jdk/usrconf/exmemexcludeclass.cfg

なお，新たに明示管理ヒープ機能適用除外設定ファイルを作成した場合は，-XX:ExplicitMemoryExcludeClassListFile オプションにファイルパスを指定してください。

- **明示管理ヒープ機能適用除外無効設定ファイル**

  明示管理ヒープ機能適用除外設定ファイルに指定したクラスのうち，適用除外設定を無効にしたいクラスを指定します。このファイルに指定したクラスのオブジェクトは，GC が発生すると，Explicit ヒープへ移動します。

  明示管理ヒープ機能の適用対象から除外されているクラスを無効にしたい場合は，次のファイルパスにあるファイルを更新するか，または新たなファイルを作成してください。システムで提供している明示管理ヒープ機能適用除外設定ファイルに指定されているクラスも設定を無効にできます。

  Windows の場合

  <JDK インストールディレクトリ>¥usrconf¥exmemnotexcludeclass.cfg

  UNIX の場合

  /opt/Cosminexus/jdk/usrconf/exmemnotexcludeclass.cfg

  なお，新たに明示管理ヒープ機能適用除外無効設定ファイルを作成した場合は，-XX:ExplicitMemoryNotExcludeClassListFile オプションにファイルパスを指定してください。

## (2) 設定ファイルの指定と明示管理ヒープ機能の適用範囲

明示管理ヒープ機能適用除外無効設定ファイルの指定は，明示管理ヒープ機能適用除外設定ファイルの指定よりも優先されます。

パッケージ「com.sample」を例に，設定ファイルの指定と明示管理ヒープ機能の適用範囲について説明します。パッケージ「com.sample」には，ClassA と ClassB の二つのクラスがあります。各設定ファイルを次のように指定します。

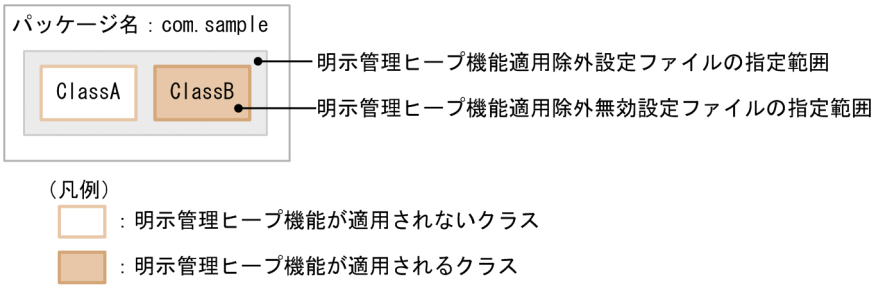
- **明示管理ヒープ機能適用除外設定ファイルの指定例**

```
com.sample.*
```

- **明示管理ヒープ機能適用除外無効設定ファイルの指定例**

```
com.sample.ClassB
```

明示管理ヒープ機能適用除外設定ファイルの指定には，ClassA と ClassB の両方が含まれています。しかし，明示管理ヒープ機能適用除外無効設定ファイルの指定が優先されるため，次の図のように，明示管理ヒープ機能の適用が除外されるのは ClassA だけとなり，ClassB には明示管理ヒープ機能が適用されます。




## (3) 設定ファイルの記述形式

設定ファイルの記述形式を次に示します。

- 配列型以外の場合

```
<指定したクラスの完全修飾クラス名>※#コメント
：
<指定したクラスの完全修飾クラス名>※
```

注※

クラス名は，「*」を使用すると省略できます。

- 配列型の場合

```
<配列の次元数分の"["">※L<指定したクラスの完全修飾クラス名>;
```

注※

多次元配列のときは，「[」を次元数分続けて指定します。3 次元配列の場合は「[[[」となります。

（例）aaa.bbb.Myclass クラスの 1 次元配列の場合

[Laaa.bbb.Myclass;

### ポイント

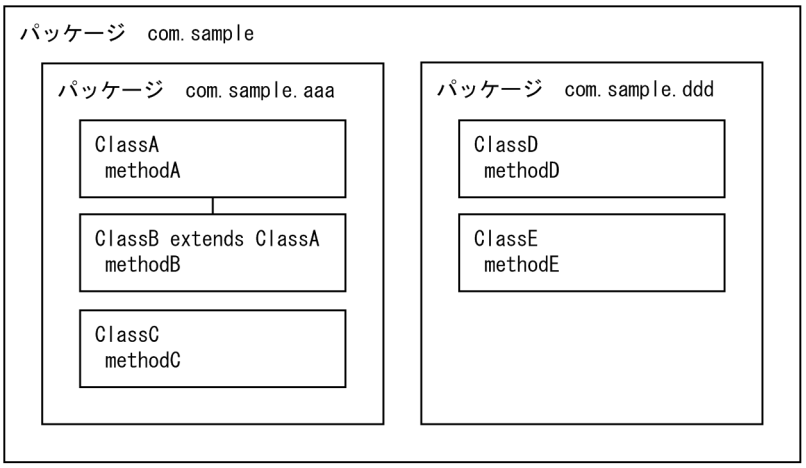
- クラス名は 1 行に一つずつ記述します。
- 1 行に記述できる文字数は 1,024 文字までです。この文字数は空文字やコメントを含みます。1 行に 1,025 文字以上記述すると，パースに失敗してワーニングメッセージを出力し，その行を無視して読み込み処理を続けます。
- クラス名は，「<パッケージ名>.*」と指定すると省略できます。Java 言語の import 宣言の「*」とは異なり，サブパッケージのクラスも対象となります。
- 行末は，改行文字（「¥n」もしくは「0x0A」）または復帰文字（「¥r」もしくは「0x0D」）が 1 文字以上続いたものとなります。
- 空白文字は，半角スペース文字（「0x20」）またはタブ文字（「¥t」もしくは「0x09」）となります。なお，設定ファイルに空白文字を記述した場合は無視されます。
- コメントは，「#」で始まり，「#」から行末までの間の文字すべてをコメントとします。

## (4) 設定ファイルの記述例

明示管理ヒープ機能適用除外設定ファイル，および明示管理ヒープ機能適用除外無効設定ファイルの記述例を次に示します。

なお，ここで説明する記述例は，パッケージ名が「com.sample」で，次の図に示すクラス構造とします。

図 7-26　クラス構造の例




- 完全修飾クラス名で指定する場合

  完全修飾クラス名で指定する場合の明示管理ヒープ機能適用除外設定ファイルの記述例を次に示します。

```
com.sample.aaa.ClassA
com.sample.aaa.ClassC
com.sample.ddd.ClassD
```

  この例では，ClassA クラス，ClassC クラス，および ClassD クラスのオブジェクトが Tenured 領域へ移動します。

- クラス名を省略して指定する場合

  クラス名を省略して指定する場合の明示管理ヒープ機能適用除外設定ファイル，および明示管理ヒープ機能適用除外無効設定ファイルの記述例を次に示します。

  - 明示管理ヒープ機能適用除外設定ファイルの記述例

```
com.sample.*
```

  - 明示管理ヒープ機能適用除外無効設定ファイルの記述例

```
com.sample.aaa.ClassB
com.sample.ddd.ClassE
```

  この例では，明示管理ヒープ機能適用除外設定ファイルの記述から，同一パッケージ内のクラスだけでなく，サブパッケージに存在するクラスも含めすべてのクラスが Tenured 領域への移動対象となります。しかし，明示管理ヒープ機能適用設定ファイルの記述から，ClassB クラスと ClassE クラスのオブジェクトが Explicit メモリブロックへの移動対象となります。このため，ClassA クラス，ClassC クラス，および ClassD クラスのオブジェクトが Tenured 領域へ移動します。

**ポイント**

完全修飾クラス名で指定するか，またはクラス名を省略して指定するかは，設定ファイルの記述量が少ない方で指定することをお勧めします。記述例はどちらも同じ制御となります。この場合は，クラス名を省略して指定する方が望ましい記述です。

## 7.13.4　J2EE サーバで利用するための設定

ここでは，J2EE サーバで明示管理ヒープ機能を利用するための設定について説明します。J2EE サーバでは，次のオブジェクトを Explicit ヒープに配置する対象にするかどうかを，J2EE サーバのプロパティとして設定します。

- HTTP セッションに関するオブジェクト

デフォルトでは，Explicit ヒープに配置するように設定されています。ただし，「7.13.1　明示管理ヒープ機能を利用するための共通の設定（JavaVM オプションの設定）」で説明した JavaVM オプションで，明示管理ヒープ機能を使用しない設定に変更した場合は，J2EE サーバのプロパティの設定は無効になります。

### (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。明示管理ヒープ機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの明示管理ヒープ機能の定義について次の表に示します。

表 7-15　簡易構築定義ファイルでの明示管理ヒープ機能の定義

| 指定するパラメタ | 設定内容 |
|---|---|
| ejbserver.server.eheap.httpsession.enabled | HTTP セッションに関するオブジェクトを Explicit ヒープに配置するかどうかを指定します。 |

指定するパラメタの詳細は，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)」の「4.11　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。

次に，JavaVM オプションと各プロパティの関係について説明します。

#### JavaVM オプションと ejbserver.server.eheap.httpsession.enabled プロパティの関係

前提となる JavaVM オプションと ejbserver.server.eheap.httpsession.enabled プロパティの指定値によって，HTTP セッションに関するオブジェクトの配置先が異なります。HTTP セッションに関するオブジェクトの配置先を次の表に示します。

表 7-16　JavaVM オプションと ejbserver.server.eheap.httpsession.enabled プロパティの値による HTTP セッションに関するオブジェクトの配置先

| JavaVM オプション | ejbserver.server.eheap.httpsession.enabled プロパティの値 | 配置先 |
|---|---|---|
| -XX:+HitachiUseExplicitMemory | true | Explicit ヒープ領域 |
| | false | Java ヒープ領域 |
| | そのほか（不正な文字列，指定なしなど） | Explicit ヒープ領域 |
| -XX:-HitachiUseExplicitMemory | true | Java ヒープ領域 |
| | false | |
| | そのほか（不正な文字列，指定なしなど） | |
| 指定なし | true | Java ヒープ領域 |
| | false | |
| | そのほか（不正な文字列，指定なしなど） | |

## (2)　簡易構築定義ファイルの定義例

簡易構築定義ファイルでの定義例を次に示します。

- 簡易構築定義ファイルでの定義例

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
  <param>
    <param-name>ejbserver.server.eheap.httpsession.enabled</param-name>
    <param-value>true</param-value>
  </param>
     :
 </configuration>
```

## 7.14　明示管理ヒープ機能使用時の注意事項

この節では，明示管理ヒープ機能使用時の注意事項について説明します。

### 7.14.1　Java ヒープの初期サイズと最大サイズの設定

Java ヒープの初期サイズ（-Xms）と最大サイズ（-Xmx）には，必ず同じ値を指定してください。同じ設定でない場合，CopyGC の回数が増加するおそれがあります。

なお，この設定は，明示管理ヒープ機能を使用しない場合にも推奨の設定です。

**補足：**

Java ヒープの初期サイズと最大サイズが異なる場合，各領域のサイズは，次のタイミングで変更されます。

- CopyGC 終了時

  New 領域のサイズが動的に変更されます。

- FullGC 終了時

  Tenured 領域と New 領域のサイズが動的に変更されます。

New 領域のサイズは，主に Tenured 領域のサイズと-XX:NewRatio オプションに指定した値によって決まります。

明示管理ヒープ機能によって FullGC の発生が抑止されると，Tenured 領域のサイズが変更されるタイミングがなくなります。これに伴って，New 領域のサイズも，ほぼ一定になります。

このため，初期サイズよりも大きい最大サイズを定義していても，New 領域が拡張されるタイミングがなくなり，初期サイズで指定したままのサイズになります。初期サイズで指定した New 領域が小さい場合，明示管理ヒープ機能を使用しない場合に比べて，CopyGC が多く発生するおそれがあります。

### 7.14.2　HTTP セッションに関するオブジェクトで Explicit ヒープを利用する際の注意

- HTTP セッション生成以降，setAttribute メソッドで設定したすべてのセッション属性（オブジェクト）は，HTTP セッションを破棄するまで解放されないで，Explicit ヒープに残存します。その時点で HTTP セッションに設定されているかどうかは関係ありません。このため，HTTP セッションを破棄しないで setAttribute メソッドを繰り返し実行した場合，Explicit ヒープあふれが発生して，FullGC 抑止の効果が得られないおそれがあります。Explicit ヒープあふれが発生しているかどうかを確認するには，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「7.14.3　Explicit ヒープあふれが発生した場合の確認と対処」を参照してください。
- 自動解放機能を利用しない場合（-XX:-HitachiExplicitMemoryAutoReclaim の場合）で，HTTP セッションを削除する時に，そのセッションに格納したオブジェクトへの外部からの参照が残っているオブ

ジェクトは，Explicit ヒープから Java ヒープの Tenured 領域に移動します。この場合，Tenured 領域の使用済みサイズが増加することになり，FullGC の発生を抑止できません。

Explicit ヒープから Java ヒープへのオブジェクトの移動を防ぐためには，HTTP セッションを破棄する前に，セッションに格納したオブジェクトへの参照を削除する必要があります。

次の API を使用して取得したオブジェクトへの参照が残っている場合も同様です。

- getAttribute(String)
- getAttributeNames()

なお，自動解放機能を利用する場合（-XX:+HitachiExplicitMemoryAutoReclaim の場合）は，それらのオブジェクトが Java ヒープの Tenured 領域に移動することはありません。

- 次の場合，オブジェクトは Explicit ヒープではなく Java ヒープに配置されます。
  - Explicit メモリブロックの数が最大値になっており，新たに Explicit メモリブロックを作成できない場合（同時に存在する Explicit メモリブロックが 1,048,575 個になっている場合）に，新しいセッションを生成した場合
  - Explicit ヒープ領域の最大サイズを超えた場合
  - Explicit メモリブロックを確保できなかった場合

  これらに該当する場合，オブジェクトは Java ヒープに作成されるため，FullGC の発生抑止ができないおそれがあります。
- JSP では，デフォルトで暗黙的に HttpSession オブジェクトが作成されます。不要な HttpSession オブジェクトの生成による Explicit ヒープあふれを防ぐため，セッションを必要としない JSP では，明示的に HttpSession オブジェクトを作成しない設定にしてください。設定には，page ディレクティブの session 属性を使用します。
- FullGC 抑止の効果を検証するテストを実行する場合，セッションを破棄しないままで連続してセッションを生成し続けるような条件では実行しないでください。Explicit メモリブロックが解放されないため，Explicit ヒープがあふれる可能性が高くなります。

  また，Explicit メモリブロックは，セッションが破棄されたときに解放予約され，その後，GC が発生したときに解放されます。このため，セッションを破棄していても，セッションの破棄と生成の繰り返し回数が 1 回の GC 間隔に対して多過ぎる場合には，解放予約された Explicit メモリブロックが残存したままの状態で別の Explicit メモリブロックが作成されてしまいます。その結果，Explicit メモリブロックの個数が増加し，Explicit ヒープがあふれるおそれがあります。

  FullGC 抑止の効果を検証するには，セッションを適切に管理する条件でテストを実行してください。
- セッションに格納したオブジェクトは，生成直後は Java ヒープに配置されます。何度か CopyGC が実行されたあと，通常は Tenured 領域に昇格するタイミングで Explicit ヒープに移動します。このため，短時間で削除される場合や，セッションがすぐに破棄される場合は，オブジェクトは Explicit ヒープには配置されません。

### 7.14.3　スレッドダンプへ出力する Explicit メモリブロック名称の文字数の上限

JavaVM のスレッドダンプに出力する Explicit ヒープ詳細情報には，Explicit メモリブロックの名称が出力されます。Explicit メモリブロックの名称の文字数の上限は 2,000 文字です。
JP.co.Hitachi.soft.jvm.MemoryArea クラスの setName メソッドで，2,000 文字を超える Explicit メモリブロックの名称を設定した場合，2,000 文字を超えた部分の名称は，スレッドダンプに出力されません。

# 8 アプリケーションのユーザログ出力

この章では，J2EE アプリケーション，バッチアプリケーション，および EJB クライアントアプリケーションのログ出力の概要と出力方法について説明します。

## 8.1　この章の構成

J2EE アプリケーション，バッチアプリケーション，および EJB クライアントアプリケーションが出力するログを，ユーザログといいます。この章では，アプリケーションのユーザログ出力について説明します。

トラブル発生時には，出力されたユーザログを収集・分析して，トラブルの発生要因を調査します。ユーザログの取得には，snapshot ログとして一括して取得する方法と，個別に取得する方法があります。ユーザログを含む snapshot ログの収集については，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「2.3.3　snapshot ログの収集」を参照してください。

この章の構成を次の表に示します。

### 表 8-1　この章の構成（アプリケーションのユーザログ出力）

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | ユーザログ出力の概要 | 8.2 |
| | ログのフォーマット | 8.3 |
| 実装 | ユーザログ出力で使用するメソッド | 8.4 |
| | ユーザログを出力するための実装 | 8.5 |
| 設定 | ロガーとハンドラの作成と設定 | 8.6 |
| | ユーザ独自のフィルタ／フォーマッタ／ハンドラの使用方法 | 8.7 |
| | J2EE アプリケーションのユーザログ出力の設定 | 8.8 |
| | バッチアプリケーションのユーザログ出力の設定 | 8.9 |
| | EJB クライアントアプリケーションのユーザログ出力の設定（cjclstartap コマンドを使用する場合） | 8.10 |
| | EJB クライアントアプリケーションのユーザログ出力の実装と設定（vbj コマンドを使用する場合） | 8.11 |
| 注意事項 | ユーザログ機能を使用する場合の注意事項 | 8.12 |

注　「運用」について，この機能固有の説明はありません。

ユーザログ出力の参照先はアプリケーションの種類によって異なります。参照先について次の表に示します。

### 表 8-2　ユーザログ出力に関する参照先

| 参照先 | アプリケーションの種類 | | |
|---|---|---|---|
| | J2EE アプリケーション | バッチアプリケーション | EJB クライアントアプリケーション |
| 8.2　ユーザログ出力の概要 | ○ | ○ | ○ |
| 8.3　ログのフォーマット | ○ | ○ | ○ |

| 参照先 | アプリケーションの種類 | | |
|---|---|---|---|
| | J2EE アプリケーション | バッチアプリケーション | EJB クライアントアプリケーション |
| 8.4　ユーザログ出力で使用するメソッド | ○ | ○ | ○ |
| 8.5　ユーザログを出力するための実装 | ○ | ○ | × |
| 8.6　ロガーとハンドラの作成と設定 | ○ | ○ | × |
| 8.7　ユーザ独自のフィルタ／フォーマッタ／ハンドラの使用方法 | ○ | ○ | × |
| 8.8　J2EE アプリケーションのユーザログ出力の設定 | ○ | × | × |
| 8.9　バッチアプリケーションのユーザログ出力の設定 | × | ○ | × |
| 8.10　EJB クライアントアプリケーションのユーザログ出力の設定（cjclstartap コマンドを使用する場合） | × | × | ○ |
| 8.11　EJB クライアントアプリケーションのユーザログ出力の実装と設定（vbj コマンドを使用する場合） | × | × | ○ |
| 8.12　ユーザログ機能を使用する場合の注意事項 | ○ | ○ | ○ |

（凡例）○：参照する　　×：参照しない

## 8.2　ユーザログ出力の概要

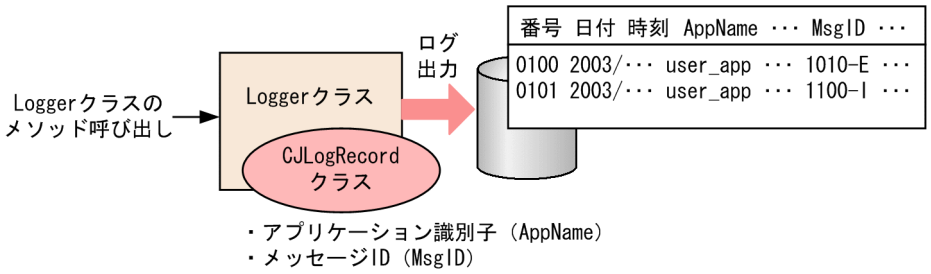
この節では，ユーザログ出力の概要について説明します。

### 8.2.1　ユーザログ出力の概要

J2EE アプリケーション，バッチアプリケーション，および EJB クライアントアプリケーションが出力するログを，**ユーザログ**といいます。アプリケーションサーバでは，ユーザログを，トレース共通ライブラリ形式で出力できます（**ユーザログ機能**）。これによって，システムのログとアプリケーションのログを同じ形式で扱えるようになり，システム全体のログ運用の信頼性を高められます。

次に，ユーザログ機能を使用したログ出力の流れを示します。

図 8-1　ユーザログ機能の処理の流れ




ユーザログ出力の実装には J2SE の標準のログ出力機能（Java ロギング API）を使用します。この機能を使用する場合は，ユーザログ出力を Java ロギング API で実装してください。

**参考**

リソースアダプタからユーザログを出力することはできません。なお，リソースアダプタから呼び出される Message-driven Bean からは，ユーザログを出力できます。

### 8.2.2　ユーザログ出力の仕組み

ユーザログを出力する J2EE アプリケーション，バッチアプリケーション，および EJB クライアントアプリケーションの実装には，J2SE の Java ロギング API を使用できます。Java ロギング API は，メモリ，コンソール，ファイルなどのさまざまな出力ができる汎用性の高い API です。ただし，ロジックが単純なため，ミッション・クリティカルなシステムに適用する場合は，信頼性と耐久性を備えたログ出力用クラスをアプリケーション開発者が実装する必要があります。

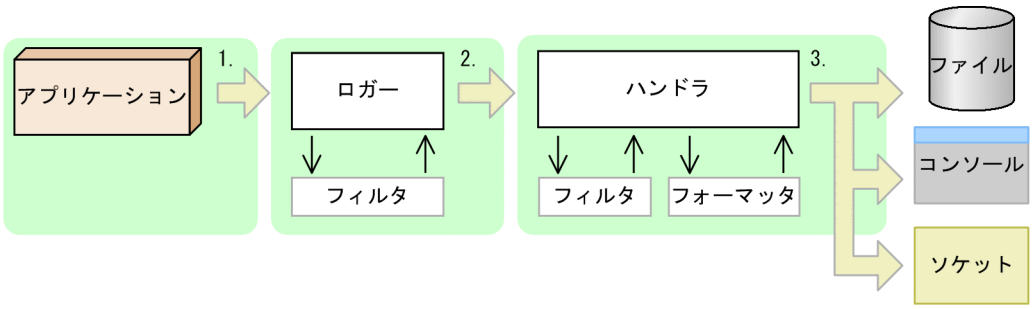
ユーザログ機能を使用すると，アプリケーション開発者によってログ出力用クラスを実装しなくても，信頼性が高いユーザログを出力できます。

Java ロギング API を使用して開発した J2EE アプリケーション，バッチアプリケーション，および EJB クライアントアプリケーションから出力されたログは，**トレース共通ライブラリ**を使用して，ほかのアプリケーションサーバシステムの構成ソフトウェアが出力する形式（**トレース共通ライブラリ形式**）で出力できます。このライブラリを使用することで，ユーザログをほかのシステムのログと同じ形式で扱うことができ，高い信頼性を持つ統一的なログ運用ができます。

ユーザログ出力は，J2SE の Java ロギングの仕組みに従って出力します。Java ロギングでは，**ロガー**と**ハンドラ**という 2 種類の要素を使用します。なお，ロガーおよびハンドラは，それぞれ，Logger クラスおよび Handler クラスのオブジェクトです。

Java ロギングの仕組みを次の図に示します。

図 8-2　Java ロギングの仕組み




図について説明します。

1. **アプリケーションから，ロガーを使用して，ユーザログを出力します。**
   ユーザログは，アプリケーションの処理の中で，Logger クラスのメソッドを使用して出力されます。

2. **ロガーは，アプリケーションから出力されたログにレベルやメッセージ文字列などの付加情報を追加して LogRecord にしたものを，ハンドラに渡します。**
   なお，このとき，ロガーに接続されたフィルタ（Filter クラスのオブジェクト）を使用して，ログレベルとして指定する制御以上のきめ細やかな制御をすることもできます。

3. **ハンドラは，受け取った LogRecord を基に，ログをファイル，コンソールまたはソケットに出力します。**
   出力先や出力形式は，あらかじめハンドラのプロパティとして設定しておきます。ハンドラでは，ハンドラに接続されたフィルタを使用してきめ細やかな制御ができます。また，フォーマッタ（Formatter クラスのオブジェクト）を使用して任意の形式にフォーマットしたメッセージを出力できます。

アプリケーションサーバでは，トレース共通ライブラリ形式でログをファイルに出力するためのファイルハンドラを提供しています。提供しているファイルハンドラについて，アプリケーションの種類ごとに説明します。

- **J2EE アプリケーションまたはバッチアプリケーションの場合**
  ファイルハンドラとして，CJMessageFileHandler を提供しています。CJMessageFileHandler のログの出力先ファイル，ログレベル，ログ面数，使用するフィルタおよびフォーマッタなどは，システム

構築時に設定できます。J2EE アプリケーションまたはバッチアプリケーションのユーザログ出力の設定については，「8.8　J2EE アプリケーションのユーザログ出力の設定」および「8.9　バッチアプリケーションのユーザログ出力の設定」を参照してください。

また，ユーザログに，ログを出力したアプリケーションの名称やメッセージの内容と対応したメッセージ ID を出力したい場合は，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実装する必要があります。この場合は，Application Server が提供する拡張 LogRecord 作成用のクラス（CJLogRecord クラス）を使用して実装してください。CJLogRecord クラスの使用方法については，「8.4　ユーザログ出力で使用するメソッド」を参照してください。また，CJLogRecord クラスの API 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」の「7.　ユーザログ機能で使用する API」を参照してください。

**注意事項**

ハンドラやロガーの設定を J2EE アプリケーション内に直接実装する場合は，実行するアプリケーションに LoggingPermission("control")のセキュリティ権限が必要になります。LoggingPermission("control")のセキュリティ権限の設定方法については，「8.8.2　セキュリティポリシーの設定」を参照してください。

- EJB クライアントアプリケーションの場合

ファイルハンドラとして，CJMPMessageFileHandler を提供しています。なお，EJB クライアントアプリケーションの開始に使用するコマンドによって，EJB クライアントアプリケーションのユーザログの設定方法が異なります。EJB クライアントアプリケーションのユーザログ出力の設定については，「8.10　EJB クライアントアプリケーションのユーザログ出力の設定（cjclstartap コマンドを使用する場合）」または「8.11　EJB クライアントアプリケーションのユーザログ出力の実装と設定（vbj コマンドを使用する場合）」を参照してください。

## 8.3 ログのフォーマット

ユーザログ機能を使用した場合，次に示すフォーマットでログが出力されます。

| 番号　日付　時刻 | AppName | pid　tid | MsgID | メッセージテキスト | CRLF |
|---|---|---|---|---|---|

フォーマットの項目の出力内容を次に示します。

### 表 8-3　ログフォーマット

| 項目 | 出力内容 |
|---|---|
| 番号 | トレースコードの通番（4 けた）が出力されます。0000 から始まり，9999 まで行くと，0000 に戻ります。 |
| 日付 | 出力時の日付（yyyy/mm/dd 形式）が出力されます。 |
| 時刻 | 出力時の時刻（hh:mm:ss.nnn 形式）が出力されます。 |
| AppName | アプリケーション識別名が出力されます。アプリケーション識別名は，16 バイト以内で指定します。長さの制限を超えた場合は，切り捨てられます。 |
| pid | プロセス識別子（16 進表示）が出力されます。OS の管理する値とは異なります。<br>CJMessageFileHandler を使用して出力したログの場合，JavaVM が Runtime のインスタンスに付けたハッシュ値が出力されます。<br>CJMPMessageFileHandler を使用して出力したログの場合，JavaVM がトレース共通ライブラリをロードした時刻（ミリ秒単位の時間）の下位 32 ビットが出力されます。 |
| tid | スレッド識別子（16 進表示）が出力されます。JavaVM が Thread のインスタンスに付けたハッシュ値です。OS の管理する値とは異なります。 |
| MsgID | メッセージ ID が出力されます。メッセージ ID は，21 バイト以内で指定します。長さの制限を超えた場合は，切り捨てられます。 |
| メッセージテキスト | メッセージの本体です。CR（0x0D），LF（0x0A），NULL（0x00），EOF（0x1A）などの制御文字を含まない任意の文字列です。長さは 0～4,095 文字で指定します。長さの制限を超えた場合は，切り捨てられます。なお，制御文字を含んでいた場合の出力内容は保障されません。 |
| CRLF | レコード終端記号（0x0D，0x0A）が出力されます。 |

## 8.4　ユーザログ出力で使用するメソッド

ユーザログ出力で使用する Logger クラスのメソッドと，CJLogRecord クラスが属するパッケージを示します。CJLogRecord クラスのメソッドの一覧，および機能と文法については，マニュアル「アプリケーションサーバ リファレンス API 編」の「7.　ユーザログ機能で使用する API」を参照してください。

### 8.4.1　ユーザログ出力で使用する Logger クラスのメソッド

CJLogRecord メソッドを使用して，AppName と MsgID の受け渡しをする場合，次に示す log メソッドを使用します。

```
void log(LogRecord record)
```

### 8.4.2　CJLogRecord クラスが属するパッケージ

CJLogRecord クラスをソースプログラム上で使用するには，次に示すパッケージをインポートする必要があります。

```
com.hitachi.software.ejb.application.userlog
```

このパッケージの格納先を，次に示します。

```
<Application Serverのインストールディレクトリ>¥CC¥client¥lib¥HiEJBClientStatic.jar
```

ユーザログ機能を使用する場合のプログラムの実装例については，「8.5　ユーザログを出力するための実装」を参照してください。

## 8.5　ユーザログを出力するための実装

J2EE アプリケーションまたはバッチアプリケーションでのログの出力は，Java ロギング API を使用してコーディングします。ユーザログに J2EE アプリケーションまたはバッチアプリケーションの名称やメッセージ ID を出力したい場合には，アプリケーションサーバが提供している，CJLogRecord クラスを使用します。

CJLogRecord クラスは，Java ロギング API の LogRecord クラスを継承したクラスです。メッセージ ID とアプリケーション名を設定した CJLogRecord オブジェクトを作成できます。このクラスで作成したオブジェクトを Logger クラスの log メソッドの引数に指定することで，ユーザログに任意のメッセージ ID とアプリケーション名が出力できるようになります。

アプリケーション名「UserApp」，メッセージ ID「USER10000-E」のユーザログを出力する例

```
try{
   //エラー出力する処理の実行
}
catch(Error ex){
   logger.log(CJLogRecord.create(Level.SEVERE, ”Catch an exception”, ”UserApp”, ”USER1000
0-E”));
}
```

API 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。

## 8.6　ロガーとハンドラの作成と設定

Java ロギング API を使用してユーザログ出力をするためには，ロガーとハンドラを作成して，必要な設定をします。ログ出力に必要なアプリケーション識別名（AppName）やメッセージ ID（MsgID）などのパラメタは，アプリケーションサーバが提供する CJLogRecord クラスの create メソッドの引数に指定します。また，独自のクラスを作成することで，ログのフィルタリングや出力内容のフォーマットをカスタマイズすることもできます。

なお，ユーザログ出力をするには，ログの出力先や構成面数などのプロパティを，実行環境に設定する必要があります。実行環境でのユーザログの設定については，「8.8　J2EE アプリケーションのユーザログ出力の設定」または「8.9　バッチアプリケーションのユーザログ出力の設定」を参照してください。

J2EE アプリケーションまたはバッチアプリケーションのユーザログを出力する場合のロガーとハンドラの作成および設定の概要について説明します。

なお，EJB クライアントアプリケーションのユーザログ出力については，「8.10　EJB クライアントアプリケーションのユーザログ出力の設定（cjclstartap コマンドを使用する場合）」または「8.11　EJB クライアントアプリケーションのユーザログ出力の実装と設定（vbj コマンドを使用する場合）」を参照してください。

### 8.6.1　ロガーの作成と設定

ロガーは，ロガー名称を指定して作成します。作成時には，システム構築時に設定した内容が使用されます。

各アプリケーション内では，ロガー名称を指定して作成されたロガーを取得し，取得したロガーを使用してログを出力します。Logger クラスのメソッドでは，ロガーの作成およびログ出力の指定ができます。指定したログは，LogRecord に変換され，ハンドラに渡されて，ログファイルまたはコンソールに出力されます。

このほか，ロガーでログを取捨選択するためのフィルタ，ログのレベル，ロガーで使用するハンドラについても，必要に応じて設定できます。

### 8.6.2　ハンドラの作成と設定

ハンドラは，システム構築時に設定した内容に従って作成，設定されます。

CJMessageFileHandler を使用する場合は，ハンドラ名称を変えることで，複数のファイルハンドラを作成できます。

CJMessageFileHandler で作成したファイルハンドラには，次の項目が設定できます。

- ユーザログの出力先ファイル，面数，サイズなどのログファイルの設定

- ログ取得レベル
- 使用するフィルタ，フォーマッタ

なお，一つのハンドラが出力するログのアプリケーション名およびメッセージ ID が同じでかまわない場合には，CJMessageFileHandler のプロパティとして設定できます。メッセージごとに出力するログのアプリケーション名とメッセージ ID を変更したい場合は，アプリケーション内のログ出力処理ごとに，アプリケーション名およびメッセージ ID を出力するように CJLogRecord クラスを使用して実装してください。

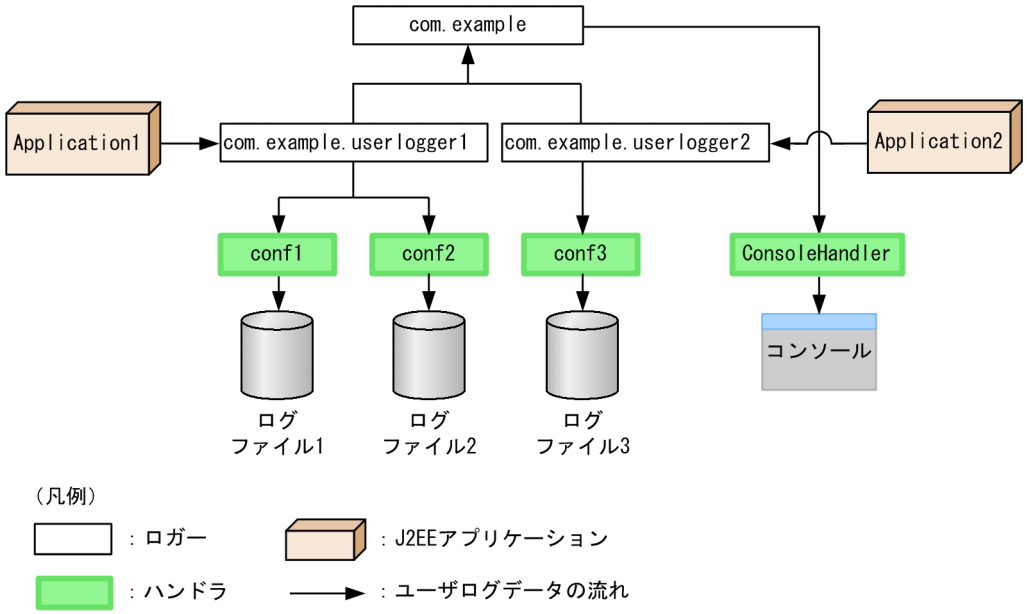
### 8.6.3 ロガーおよびハンドラを作成・設定する場合の指針

ロガーおよびハンドラを作成，設定する場合の指針を次に示します。

- 一つのロガーに対して複数のファイルハンドラを接続できますが，複数のロガーから出力先が同じファイルハンドラに接続して利用することはできません。
- アプリケーションごとにログの出力先を変えたい場合は，アプリケーションごとのロガーを作成します。
- ロガーは，階層関係を持たせることができます。階層関係を持たせた場合，下位のロガーが取得したログメッセージは，上位のロガーに伝播します。必要に応じて，ロガーの伝播を止めてください。特に，ロガーの最上位にはルートロガーがデフォルトで存在し，J2SE デフォルトの設定の場合，ルートロガーには ConsoleHandler が接続されています。上位ロガーへの伝播を止めていない場合，ルートロガーの ConsoleHandler からコンソールにすべてのメッセージが出力されます。
- ハンドラはインスタンスごとにメッセージを出力するため，一つの出力メッセージが複数のハンドラに送信された場合，一つの出力メッセージが複数回出力されます。例えば，2 か所の ConsoleHandler のメッセージは，コンソールに 2 回出力されます。
- 一つのアプリケーションで複数のログファイルを利用する場合は，出力先ごとにハンドラを作成してください。

ロガーとハンドラの作成例を次の図に示します。

図 8-3　ロガーとハンドラの作成例




この例では，J2EE アプリケーション 1 と 2 に対して，com.example.userlogger1 と com.example.userlogger2 という 2 種類のロガーを作成しています。com.example.userlogger1 からは，ログの出力レベルおよび出力内容に応じて 2 種類のログファイルを出力するために，conf1 と conf2 という 2 種類の CJMessageFileHandler ハンドラを作成しています。このような構成にすると，ログファイル 1 には SEVERE レベル以上の重要なユーザログを，ログファイル 2 には INFO レベル以上のすべてのユーザログを出力するという運用ができます。一方，com.example.userlogger2 からは 1 種類のログファイルだけを出力します。この場合は，J2EE アプリケーションから指定されたログのうち，com.example.userlogger2 のロガーとハンドラ conf3 に指定したレベル以下のユーザログは，すべてログファイル 3 に出力されます。なお，コンソールにログを出力したい場合は，J2SE の標準のハンドラである ConsoleHandler を使用してください。

それぞれのログファイルのサイズおよび面数は，アプリケーションが出力するユーザログの量や指定した出力レベルに応じて，適切に設定してください。

## 8.7　ユーザ独自のフィルタ／フォーマッタ／ハンドラの使用方法

ここでは，ユーザが作成した独自の Filter クラス，Formatter クラスまたは Handler クラスをユーザログ機能で利用するための使用方法について説明します。なお，ここでは，ユーザが作成したクラスを**ユーザ作成クラス**といいます。

ユーザ作成クラスを作成することで，ログのフィルタリングや出力内容をフォーマットできます。ユーザ作成クラスとして，Filter クラス，Formatter クラスまたは Handler クラスを作成し，ライブラリ JAR またはコンテナ拡張ライブラリに含めて使用します。

ユーザ作成クラスをユーザログ機能で使用する方法には，次の 2 種類の方法があります。

- ライブラリ JAR を利用する
  J2EE アプリケーションの場合に使用できる方法です。バッチアプリケーションの場合は使用できません。
- コンテナ拡張ライブラリを利用する
  J2EE アプリケーションまたはバッチアプリケーションで使用できる方法です。

それぞれの方法について説明します。

### 8.7.1　ライブラリ JAR を利用する方法

ユーザ作成クラスである Filter クラス，Formatter クラスまたは Handler クラスを，アプリケーション上で作成してロガーに追加して利用する方法です。この場合は，次の処理が実行されます。

- まず，アプリケーションの中で Handler クラスをインスタンス化します。
- 次に，Filter クラスや Formatter クラスをインスタンス化したものを Handler クラスのインスタンスに接続します。
- 最後に，接続した Handler クラスのインスタンスを，ロガーに追加します。

この場合のユーザ作成クラスは，J2SE の java.util.logging の仕様に従って作成してください。作成したクラスは，通常のユーザクラスと同じように，WAR，EJB-JAR またはインポート用ライブラリ JAR に含めて利用できます。

次に，ライブラリ JAR に含めて利用する場合のユーザ作成クラスの作成手順を示します。

1. セキュリティポリシーファイル（server.policy）にセキュリティポリシーを設定します。
   セキュリティポリシーの設定については，「8.8.2　セキュリティポリシーの設定」を参照してください。
2. 独自の Handler クラス，Filter クラス，および Formatter クラスを含めたインポート用のライブラリ JAR を作成します。
3. サーバ管理コマンドを使用して，作成したライブラリ JAR のクラスをインポートするように指定します。

4. アプリケーションのソースプログラム上で，独自クラスのインスタンスを生成します。

5. Logger クラス，Handler クラスへ接続する処理を実装します。

   J2SE1.4 仕様のログマネジャ（LogManager）を利用して実装する場合は，次の点に注意してください。

   - プロパティ（java.util.logging.class や java.util.logging.file など）を使用して，ログマネジャをカスタマイズすることはできません。カスタマイズした場合，ユーザログの構築に失敗するおそれがあります。
   - ソースプログラム上でログマネジャの readConfiguration(InputStream ins)メソッドなどを呼び出すことはできません。readConfiguration(InputStream ins)メソッドを呼び出して，Logger クラスの構成を初期化した場合，ユーザログ機能によって構築されたログ体系が失われます。

なお，コーディング上の注意事項については，「8.12　ユーザログ機能を使用する場合の注意事項」を参照してください。

## 8.7.2　コンテナ拡張ライブラリを利用する方法

ユーザ作成クラスである Filter クラス，Formatter クラスまたは Handler クラスのクラス名称をユーザログ機能のプロパティキーに指定しておき，J2EE サーバ起動時にユーザ作成クラスを含むログ構成を構築して利用する方法です。J2EE 標準の方法とは異なります。

ユーザ作成クラスを含めた JAR ファイルをコンテナ拡張ライブラリとして指定して，作成したライブラリへのクラスパスを指定します。これによって，J2EE サーバ起動時に，プロパティキーで指定している CJMessageFileHandler クラスとフォーマッタ，フィルタの作成，接続などが実行されて，ログ構成を構築できます。

手順を示します。

1. ユーザ作成クラスの Formatter クラス，Filter クラスおよび Handler クラスを含めた JAR ファイル（コンテナ拡張ライブラリ JAR）を作成します。

   ここでは，myloglib.jar とします。

2. myloglib.jar を任意の場所に配置します。

   ここでは，次の場所に配置することを前提に説明しています。

   - Windows の場合

     c:¥mylib

   - UNIX の場合

     /usr/mylib

3. 配置したライブラリへのクラスパスを指定します。

   例えば，J2EE サーバの場合は，usrconf.cfg（オプション定義ファイル）内に，次のように指定します。

   - Windows の場合

add.class.path=C:¥mylib¥myloglib.jar

- UNIX の場合
  add.class.path=/usr/mylib/myloglib.jar

4. usrconf.properties（ユーザプロパティファイル）のユーザログ機能用のプロパティキーに，パッケージ名称を含むフルクラス名を指定します。

## 8.8　J2EE アプリケーションのユーザログ出力の設定

この節では，J2EE アプリケーションが出力するログを，トレース共通ライブラリ形式で出力するための設定方法について説明します。また，アプリケーションのユーザログ出力例についても説明します。なお，J2EE アプリケーションのログを出力しない場合は，この設定は不要です。

トレース共通ライブラリ形式でログを出力するためには，次の設定が必要です。

- J2EE サーバの設定
- セキュリティポリシーの設定

### 8.8.1　J2EE サーバの設定

簡易構築定義ファイルを編集して，ハンドラからのログの出力先，ログレベル，ログ面数，使用するフィルタ，フォーマッタなどを指定してください。

#### (1)　設定内容

簡易構築定義ファイルで論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，ejbserver.application から始まるパラメタで，J2EE アプリケーションのユーザログを出力するための設定をします。ejbserver.application から始まるパラメタを次に示します。なお，<ハンドラ名称>には，キーの値を区別するためのハンドラ名称を指定してください。また，<ロガー名称>には，Logger のインスタンスを取得するときに指定するロガー名称を指定してください。

- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.appname
  ハンドラごとに，ログファイルに出力するメッセージの J2EE アプリケーション名（AppName フィールドの値）のデフォルト値を指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.count
  ハンドラごとに，ログファイルの面数を指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.encoding
  ハンドラごとに，ログファイルに出力する文字列のエンコーディングを指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.filter
  ハンドラごとに，使用するフィルタ名を指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.formatter
  ハンドラごとに，使用するフォーマッタ名を指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.level
  ハンドラごとに，ログ取得レベルの上限を指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.limit
  ハンドラごとに，ログファイルのサイズを指定します。

- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.msgid
  ハンドラごとに，ログファイルに出力するメッセージのメッセージ ID（MsgID フィールドの値）のデフォルト値を指定します。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.path
  ハンドラごとに，ログファイルの出力先とプリフィックスを指定します。出力されるログファイル名は，「<プリフィックス><1～16 の番号>.log」になります。このキーは必ず指定してください。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.separator
  ハンドラごとに，ログファイルに出力するメッセージを 1 文で出力するための，要素区切り文字のデフォルト値を指定します。
- ejbserver.application.userlog.loggers
  使用するロガー名称を宣言します。このキーは必ず指定してください。
- ejbserver.application.userlog.Logger.<ロガー名称>.handlers
  ロガーごとに，使用するハンドラ名称を指定します。このキーは必ず指定してください。
- ejbserver.application.userlog.Logger.<ロガー名称>.level
  ロガーごとに，ロガーのログ取得レベルを指定します。
- ejbserver.application.userlog.Logger.<ロガー名称>.useParentHandlers
  ロガーごとに，ロガーを通過するレベルのログレコードを，親のロガーが使用しているハンドラに伝播させるかどうかを指定します。
- ejbserver.application.userlog.Logger.<ロガー名称>.filter
  ロガーごとに，ロガーでメッセージの取捨選択に使用するフィルタを指定します。

J2EE アプリケーションのユーザログを出力するためには，少なくとも，次の三つのパラメタを指定する必要があります。

- ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.path
- ejbserver.application.userlog.loggers
- ejbserver.application.userlog.Logger.<ロガー名称>.handlers

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。

## (2)　注意事項

- ロガーには複数のハンドラを接続できます。ただし，複数のロガーに同一の Path 設定を持つファイルハンドラ（CJMessageFileHandler）は接続できません。ファイルハンドラはロガーへの接続の指定（ejbserver.application.userlog.Logger.<ロガー名称>.handlers の値）を参照してインスタンス化します。この際，ログの出力先とプリフィックス（ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.path の値）の同じハンドラがインスタンス化されている場合は，ログファイルのオープンに失敗します。

- ハンドラやロガーの設定および構築については，簡易構築定義ファイルで指定できますが，ハンドラの作成やロガーの構成変更を J2EE アプリケーション内に直接実装する場合は，実行するアプリケーションに LoggingPermission("control")のセキュリティ権限が必要になります。
  LoggingPermission("control")のセキュリティ権限の設定方法については，「8.8.2　セキュリティポリシーの設定」を参照してください。

## 8.8.2　セキュリティポリシーの設定

ここでは，セキュリティポリシーの設定について説明します。

アプリケーションのソースプログラム上で，J2SE1.4 仕様の Logger クラスの構成を変更したり，FileHandler クラスを作成したりして，J2SE 標準のロギング機能を直接実装する場合，セキュリティポリシーを設定する必要があります。セキュリティポリシーは，server.policy（J2EE サーバ用セキュリティポリシーファイル）または web.policy（SecurityManager 定義ファイル）に定義します。

なお，server.policy にセキュリティポリシーを定義する場合は，Smart Composer 機能のコマンドでシステムを構築したあとに設定してください。

簡易構築定義ファイルのパラメタを基に構築されたロガーに対して出力指定をする場合，セキュリティポリシーを設定する必要はありません。セキュリティポリシーの設定が必要なのは，次のような場合です。

- ユーザのアプリケーションのソースコード上で J2SE 標準のファイルハンドラを作成する場合
- Logger クラスの addHandler メソッドを呼び出してロガーの構成を変更する場合

この場合には，Java ロギング API 操作用のセキュリティポリシーが必要になります。必要に応じて次のセキュリティパーミッションを指定してください。

server.policy の設定内容を次に示します。

### (1)　フィルタやフォーマッタをリフレクションで作成する場合

Filter クラスや Formatter クラスなどをリフレクションで作成する場合は，次に示す行を追加します。

```
permission java.lang.reflect.ReflectPermission ”suppressAccessChecks”;
```

それぞれの Handler クラスは，ログマネジャ（LogManager）からプロパティを取得して，実行時に Reflection 機能を使って Formatter クラスまたは Filter クラスを生成します。このため，Reflection に関する権限が必要です。

### (2)　ログマネジャ（LogManager）のプロパティを設定する場合

ログマネジャのプロパティを設定する場合は，次に示す行を追加します。

```
permission java.util.PropertyPermission ”*”, ”read, write”;
```

ログマネジャがログ出力用のプロパティの値を読み込んだり，書き込んだりするための権限（Property の set**）が必要となります。

## (3) J2SE 標準のファイルハンドラを使用する場合（File 出力を行うクラス（FileHandler，CJMessageFileHandler）を使用する場合）

File 出力を行うクラス（FileHandler，CJMessageFileHandler）を使用する場合は，次に示す行を追加します。

```
permission java.io.FilePermission "<<ALL FILES>>", "read, write";
```

ログを実際にファイルに出力するための権限が必要です。ログのファイルへの出力には，読み取り権限だけではなく，書き込み権限も必要です。

## (4) Java ロギング API の Logger.addHandler メソッドなどを使用してログ体系を変更する場合

J2SE1.4 仕様のロギング API を使用する場合は，次に示す行を追加します。

```
permission java.util.logging.LoggingPermission "control";
```

Java ロギング API を使用するためのセキュリティパーミッションの指定が必要です。この値を指定しないと，ロギング API が使用できません。

## (5) 設定例

J2EE アプリケーションのサーブレットから，Java ロギング API の Logger.addHandler メソッドなどを使用してログ体系を変更する場合の server.policy（J2EE サーバ用セキュリティポリシーファイル）の設定例を次に示します。

**設定例**

```
//
// 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.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.*";

   //For J2SE Logging Source
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
   permission java.util.PropertyPermission "*", "read, write";
   permission java.util.logging.LoggingPermission "control";
```

```
};
```

server.policy（J2EE サーバ用セキュリティポリシーファイル）の定義方法については，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「2.2.4　server.policy（J2EE サーバ用セキュリティポリシーファイル)」を参照してください。

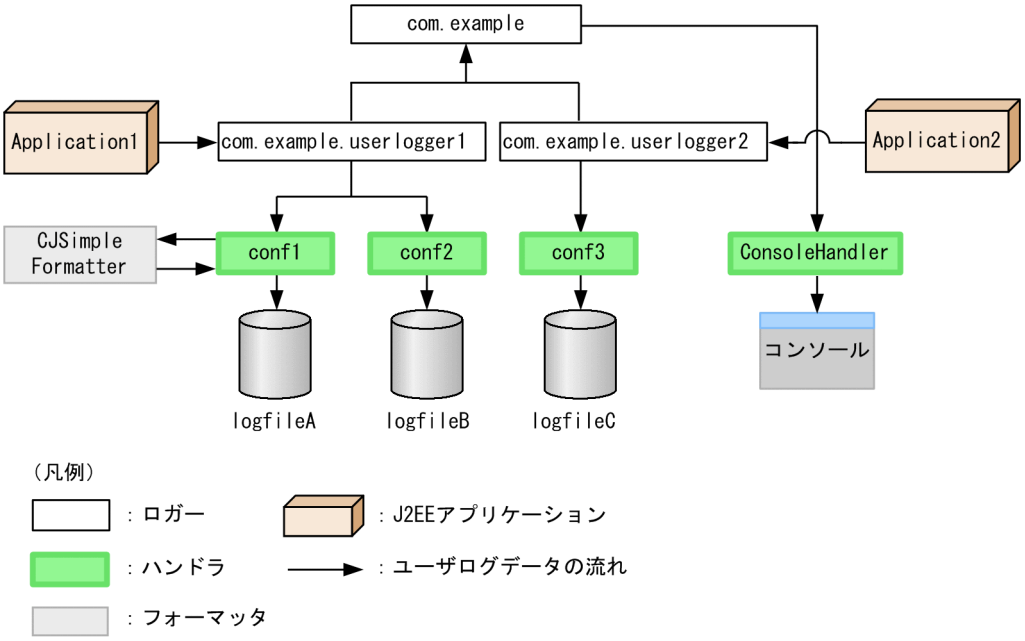
## 8.8.3　アプリケーションのユーザログ出力例

ここでは，具体的な例を示して，J2EE アプリケーションのユーザログを出力するための設定について説明します。

### (1)　ユーザログ出力で使用する例

次に示す例を使用して，J2EE アプリケーションのユーザログ出力の設定について説明します。使用する例の概要を次の図に示します。

図 8-4　J2EE アプリケーションのユーザログ出力例




A 社では，ロガーの機能を使用して，業務履歴として J2EE アプリケーションの動作履歴をログファイルに出力します。A 社の J2EE アプリケーションのうち，動作履歴を出力したい J2EE アプリケーションは「Application1」と「Application2」の 2 種類です。J2EE アプリケーションごとに，別々のファイルに異なるメッセージレベルのログを出力します。また，J2EE アプリケーション名のディレクトリを作成して，それぞれのログファイルを格納します。

#### (a)　「Application1」の特徴

「Application1」のロガー名称は「com.example.userlogger1」とします。

「Application1」は，複雑で規模の大きいJ2EEアプリケーションです。「SEVERE」レベルの重大なエラーが起こったときに，原因の切り分けを素早く行うために，Javaの Exceptionのトレース情報を含むメッセージを「logfileA」に残しておきます。また，動作のトレースログとして「INFO」レベル以下のメッセージを「logfileB」に出力します。「com.example.userlogger1」からは，ログの出力レベルおよび出力内容に応じて2種類のログファイルを出力するために，「conf1」と「conf2」という2種類のCJMessageFileHandlerハンドラを作成しています。

「logfileA」の詳細

- トレース情報を取得するため，「logfileA」への出力フォーマッタとして，「CJSimpleFormatter」を使用します。
- 「logfileA」は，「SEVERE」レベルのメッセージだけが出力されるため，それほど多くのファイル容量を必要としません。しかし，トレース情報を出力するので，1メッセージ当たりのレコードは大きく（最大約4,096バイト），10,000レコードを蓄積できるようにするためには約40メガバイトの容量が必要となります。このため，サイズは10メガバイト，面数は4とします。
- 「Application1」が出力したメッセージであることを判別するために，J2EEアプリケーション名を「my_app1」とします。

「logfileB」の詳細

- 「logfileB」は，「INFO」レベル以下のすべてのメッセージが出力されるため，多くのファイル容量を必要とします。1日当たりのメッセージ量と保存期間から算出したログディスク容量は約256メガバイトです。また，ファイルの最大面数は16であるため，サイズは16メガバイト，面数は16とします。
- 「Application1」が出力したメッセージであることを判別するために，J2EEアプリケーション名を「my_app1」とします。

## (b) 「Application2」の特徴

「Application2」のロガー名称は「com.example.userlogger2」とします。

「Application2」は，ログメッセージの作り込み品質が高く，規模が小さいJ2EEアプリケーションです。必要最小限のメッセージだけをログに出力するため，「WARNING」レベル以上のメッセージを「logfileC」に残します。「com.example.userlogger2」からは1種類のログファイルを出力するため，「conf3」というCJMessageFileHandlerハンドラを作成しています。

「logfileC」の詳細

- 「WARNING」レベルのメッセージだけが出力されます。また，1メッセージ当たりの最大長が約200バイトであるので，10,000レコードを蓄積するためには約2メガバイトの容量が必要となります。このため，サイズは1メガバイト，面数は2とします。
- 「Application2」が出力したメッセージであることを判別するために，J2EEアプリケーション名を「my_app2」とします。

### (c) デバッグ用の設定

開発中のデバッグ用の設定もしておきます。「com.example.userlogger1」と「com.example.userlogger2」へ送信されてきたすべてのメッセージの内容を表示するために，ロガー名称「com.example」のロガーへ，java.util.logging の「ConsoleHandler」を接続しておきます。このロガーでは，子のロガーから伝播されるすべてのメッセージの内容を表示したいので，ロガーおよびハンドラのログ取得レベルは「ALL」とします。

## (2) ユーザログ出力の設定例

「(1) ユーザログ出力で使用する例」で示した例でユーザログを出力する場合の設定例を次に示します。

### (a) 簡易構築定義ファイルの設定例

簡易構築定義ファイルの設定例（物理ティアの定義の場合）を次に示します。

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
<!-- ロガーに渡されたログレコードを，親ロガーが使用しているハンドラに -->
<!-- 伝播させないようにします（ルートロガーがデフォルトで存在するため）。-->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.useParentHandlers</param-na
me>
    <param-value>false</param-value>
  </param>
<!-- 「logfileA」のJ2EEアプリケーション名，出力先，サイズ，面数，ログ取得レベル， -->
<!-- 使用するフォーマッタ名を指定します。 -->
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.appname</param-name>
    <param-value>my_app1</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.path</param-name>
    <param-value>application1/logfileA</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.limit</param-name>
    <param-value>10485760</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.count</param-name>
    <param-value>4</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.level</param-name>
    <param-value>SEVERE</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.formatter</param-name>
    <param-value>com.hitachi.software.ejb.application.userlog.CJSimpleFormatter</param-value
>
  </param>

<!-- 「logfileB」のJ2EEアプリケーション名，出力先，サイズ，面数，ログ取得レベルを指定しま
```

```
す。 -->
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.appname</param-name>
    <param-value>my_app1</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.path</param-name>
    <param-value>application1/logfileB</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.limit</param-name>
    <param-value>16777216</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.count</param-name>
    <param-value>16</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.level</param-name>
    <param-value>INFO</param-value>
  </param>

<!-- 「com.example.userlogger1」の使用するハンドラ名称「conf1」「conf2」の設定を使用して，--
>
<!-- ファイルハンドラ（CJMessageFileHandler）を初期化して接続します。-->
<!-- ここで，ロガーとハンドラが作成されます。-->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger1.handlers</param
-name>
    <param-value>com.hitachi.software.ejb.application.userlog.CJMessageFileHandler;conf1,com
.hitachi.software.ejb.application.userlog.CJMessageFileHandler;conf2</param-value>
  </param>

<!-- 「com.example.userlogger1」のログ取得レベルを指定します。-->
<!-- 「conf1」「conf2」のレベルの高い方に合わせて，「INFO」とします。-->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger1.level</param-na
me>
    <param-value>INFO</param-value>
  </param>

<!-- 「logfileC」の出力先，ログ取得レベルを指定します。-->
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf3.appname</param-name>
    <param-value>my_app2</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf3.path</param-name>
    <param-value>application2/logfileC</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf3.level</param-name>
    <param-value>WARNING</param-value>
  </param>

<!-- 「com.example.userlogger2」の使用するハンドラ名称「conf3」の設定を使用して，-->
<!-- ファイルハンドラ（CJMessageFileHandler）を初期化して接続します。-->
<!-- ここで，ロガーとハンドラが作成されます。-->
```

```
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger2.handlers</param
-name>
    <param-value>com.hitachi.software.ejb.application.userlog.CJMessageFileHandler;conf3</pa
ram-value>
  </param>

<!-- 「com.example.userlogger2」のログ取得レベルを指定します。-->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger2.level</param-na
me>
    <param-value>WARNING</param-value>
  </param>

<!-- デバッグ用の設定をします**********************************************-->
<!-- 「ConsoleHandler」のログ取得レベルを指定します。-->
  <param>
    <param-name>java.util.logging.ConsoleHandler.level</param-name>
    <param-value>INFO</param-value>
  </param>

<!-- 「com.example」のロガーで使用するハンドラ名称「ConsoleHandler」を指定して，-->
<!-- ハンドラに接続します。ここで，ロガーとハンドラが作成されます。-->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.handlers</param-name>
    <param-value>java.util.logging.ConsoleHandler</param-value>
  </param>

<!-- 「com.example」のロガーのログ取得レベルを指定します。-->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.level</param-name>
    <param-value>ALL</param-value>
  </param>

<!-- デバッグが不要になった場合は，親のロガーへの伝播の設定を解除します。-->
<!--
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger1.useParentHandle
rs</param-name>
    <param-value>false</param-value>
  </param>
-->
<!--
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger2.useParentHandle
rs</param-name>
    <param-value>false</param-value>
  </param>
-->
<!-- デバッグが不要になった場合は，「com.example」の作成を解除します。-->
<!--
  <param>
    <param-name>ejbserver.application.userlog.loggers</param-name>
    <param-value>com.example.userlogger1, com.example.userlogger2</param-value>
  </param>
-->
<!-- **********************************************************************-->
```

```
<!-- ロガーの使用を宣言します。-->
  <param>
    <param-name>ejbserver.application.userlog.loggers</param-name>
    <param-value>com.example,com.example.userlogger1,com.example.userlogger2</param-value>
  </param>
</configuration>
```

## (b) 「Application1」の設定例

「Application1」のソースコード例を次に示します。

```
import java.util.logging.*;
import com.hitachi.software.ejb.application.userlog.*;

public class application1{

    static Logger logger = Logger.getLogger("com.example.userlogger1");

    public static void exec(){

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application1 start.","AP1_10000-I"));

        try{

            throw new Exception("Exception1!");

        }
        catch(Exception ex){

            logger.log(
                       CJLogRecord.create(Level.SEVERE,
                       "Catch an exception!", ex, "AP1_10100-E"));

        }

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application1 end.","AP1_10001-I"));

    }

}
```

application1/logfileA1.log の出力例を次に示します。

```
     yyyy/mm/dd hh:mm:ss.sss            pid      tid      message-id  message(LANG=ja)
0047 2003/12/06 19:51:32.265  my_app1   00EB7859 012A54F9 AP1_10100-E 2003/12/06
19:51:32|application1|exec|致命的|Catch an exception!|java.lang.Exception:
Exception1!|application1.exec(application1.java.18)|application1.main(application1.java.64)
```

application1/logfileB1.log の出力例を次に示します。

```
     yyyy/mm/dd hh:mm:ss.sss           pid      tid      message-id   message(LANG=ja)
0046 2003/12/06 19:51:32.250  my_app1  00EB7859 012A54F9 AP1_10000-I  application1 start.
0048 2003/12/06 19:51:32.265  my_app1  00EB7859 012A54F9 AP1_10100-E  Catch an exception!
0049 2003/12/06 19:51:32.265  my_app1  00EB7859 012A54F9 AP1_10001-I  application1 end.
```

コンソール画面の出力例を次に示します。

```
情報: application1 start.
2003/12/06 19:51:32 application1 exec
致命的: Catch an exception!
java.lang.Exception: Exception1!
        at application1.exec(application1.java:18)
        at application1.main(application1.java:64)
2003/12/06 19:51:32 application1 exec
情報: application1 end.
```

## (c) 「Application2」の設定例

「Application2」のソースコード例を次に示します。

```
import java.util.logging.*;
import com.hitachi.software.ejb.application.userlog.*;

public class application2{

    static Logger logger = Logger.getLogger("com.example.userlogger2");

    public static void exec(){

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application2 start.","AP2_20000-I"));

        try{

            throw new Exception("Exception2!");

        }
        catch(Exception ex){

            logger.log(
                        CJLogRecord.create(Level.SEVERE,
                       "Catch an exception!", ex, "AP2_20100-E"));

        }

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application2 end.","AP2_20001-I"));

    }

}
```

application2/logfileC1.log の出力例を次に示します。

```
     yyyy/mm/dd hh:mm:ss.sss           pid      tid      message-id   message(LANG=ja)
0048 2003/12/06 19:51:32.265  my_app2  00EB7859 012A54F9 AP2_20100-E  Catch an exception!
```

### (d) 「Application3」の設定例

さらに,「Application3」という J2EE アプリケーションのログを「Application1」と同じログファイルに出力する場合の例を説明します。この場合,「Application3」は「Application1」と同じプロセス内（スレッドは異なっていてもよい）で同じロガー名称を使用してロガーを取得する必要があります。

「Application3」のソースコード例を次に示します。

```
import java.util.logging.*;
import com.hitachi.software.ejb.application.userlog.*;

public class application1{

    static Logger logger = Logger.getLogger(”com.example.userlogger1”);

    public static void exec(){

        logger.log(,
                   CJLogRecord.create(Level.INFO,
                   ”application3 start.”,”my_app3”,”AP3_30000-I”));

        try{

            throw new Exception(”Exception3!”);

        }
        catch(Exception ex){

            logger.log(,
                       CJLogRecord.create(Level.SEVERE,
                       ”Catch an exception!”, ex, ”my_app3”,”AP3_30100-E”));

        }

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   ”application3 end.”,”my_app3”,”AP3_30001-I”));

    }

}
```

application1/logfileB1.log の出力例を次に示します。

```
     yyyy/mm/dd hh:mm:ss.sss           pid      tid      message-id   message(LANG=ja)
0046 2003/12/06 19:51:32.250  my_app1  00EB7859 012A54F9 AP1_10000-I  application1 start.
0093 2003/12/06 19:51:32.265  my_app3  00EB7859 010CB027 AP3_30000-I  application3 start.
0095 2003/12/06 19:51:32.265  my_app1  00EB7859 012A54F9 AP1_10100-E  Catch an exception!
```

## 8.9　バッチアプリケーションのユーザログ出力の設定

バッチアプリケーションのユーザログ出力の設定は，J2EE アプリケーションのユーザログと同じです。設定方法については，「8.8　J2EE アプリケーションのユーザログ出力の設定」を参照してください。

ただし，バッチアプリケーションの場合，セキュリティポリシーの設定は不要です。

## 8.10　EJB クライアントアプリケーションのユーザログ出力の設定（cjclstartap コマンドを使用する場合）

EJB クライアントアプリケーションのユーザログ出力の設定について説明します。

EJB クライアントアプリケーションの開始に使用するコマンドによって，EJB クライアントアプリケーションのユーザログの設定方法が異なります。ここでは，cjclstartap コマンドを使用して開始する場合の設定について説明します。

cjclstartap コマンドを使用する場合は，EJB クライアントアプリケーションのプロパティファイル（usrconf.properties）でユーザログの設定をします。ejbserver.application から始まるキーで，ユーザログの出力先ファイル，ログレベル，ログ面数，使用するフィルタおよびフォーマッタなどを指定してください。指定できるキーについては，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「3.2.2　usrconf.properties（バッチサーバ用ユーザプロパティファイル）」を参照してください。

また，EJB クライアントアプリケーションのオプション定義ファイル（usrconf.cfg）で，クラスパスに JAR ファイルを設定します。クラスパスへの JAR ファイルの設定については，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(EJB コンテナ)」の「3.7.4　EJB クライアントアプリケーションのクラスパスへの JAR ファイルの設定」を参照してください。

# 8.11 EJB クライアントアプリケーションのユーザログ出力の実装と設定（vbj コマンドを使用する場合）

ここでは，EJB クライアントアプリケーションを，vbj コマンドを使用して開始する場合の設定について説明します。vbj コマンドを使用して開始する場合，ユーザログ機能を利用するための実装が必要です。

ここでは，EJB クライアントアプリケーションでユーザログ機能を利用するための準備と，ユーザログが出力されるまでの処理の流れについて説明します。

## 8.11.1 vbj コマンドを使用する場合の処理の概要

EJB クライアントアプリケーションのユーザログのファイルハンドラとして，CJMPMessageFileHandler が提供されています。vbj コマンドを使用する場合は，CJMPMessageFileHandler のログの出力先ファイル，ログレベル，ログ面数，使用するフィルタおよびフォーマッタなどは，EJB クライアントアプリケーションのユーザログ用の設定ファイルで設定します。

ユーザログ機能を実装する際，EJB クライアントアプリケーションのユーザログ用の設定ファイルに，CJMPMessageFileHandler のログの出力先ファイル，ログレベル，ログ面数，使用するフィルタとフォーマッタなどを設定します。ユーザアプリケーションプログラムでは，ユーザログ用の設定ファイルを読み込むように記述する必要があります。

EJB クライアントアプリケーションの開始コマンド実行時に，設定ファイルがユーザアプリケーションプログラムから読み込まれて，EJB クライアントアプリケーションのシステムプロパティに設定されます。

## 8.11.2 利用の準備

EJB クライアントアプリケーションでユーザログ機能を利用する場合には，次の準備が必要です。

なお，EJB クライアントアプリケーションでユーザログ機能を利用する前提として，Application Server 側での設定が必要になります。

- **ユーザログ機能用の設定ファイルの準備**

  システムプロパティを設定するための，ユーザログ機能用の設定ファイルを準備します。設定ファイルは，J2SE のプロパティファイル形式で記述します。ファイル名称および格納ディレクトリは任意です。

  設定ファイルには，J2EE サーバ用の usrconf.properties に指定できるキーのうち，「ejbserver.application.userlog」で始まるキーを指定できます。指定できるキーについては，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「2.2.3　usrconf.properties (J2EE サーバ用ユーザプロパティファイル)」を参照してください。

- **システムプロパティの設定処理の実装**

設定ファイルを読み込んでシステムプロパティに設定するための処理を，EJB クライアントアプリケーションのソースコードに追加する必要があります。この処理は，EJB クライアント機能の初期化が実行される処理よりも前に実行されるようにする必要があります。

- JAR ファイルのクラスパスの追加

  EJB クライアントアプリケーションを開始するときのクラスパスに，使用するハンドラに対応する JAR ファイルのクラスパスを追加してください。クラスパスの指定については，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(EJB コンテナ)」の「3.7.4　EJB クライアントアプリケーションのクラスパスへの JAR ファイルの設定」を参照してください。

> **参考**
>
> EJB クライアントアプリケーションでユーザログ機能を利用する場合，セキュリティポリシーを設定する必要はありません。

## 8.11.3　ユーザログ出力処理の流れ

EJB クライアントアプリケーションでのユーザログの出力は，次の流れで行われます。

1. システムプロパティの設定

   設定ファイルを利用してシステムプロパティが設定されます。

2. EJB クライアントの初期化

   EJB クライアント機能を初期化するメソッドが呼び出されて，ログ体系が構築されます。

3. Java ロギング API の実行

   Java ロギング API の実行によって，ユーザログが出力されます。

流れに沿って，それぞれの処理内容について説明します。

### (1)　システムプロパティの設定

EJB クライアントアプリケーションのユーザログ機能用のシステムプロパティは，設定ファイルを利用して設定されます。

システムプロパティで設定できるプロパティは，J2EE サーバ用の usrconf.properties に指定できるプロパティのうち，「ejbserver.application.userlog」で始まるキーです。設定例を次に示します。

```
# user-log handler function
ejbserver.application.userlog.CJLogHandler.conf1.appname=my_app1
ejbserver.application.userlog.CJLogHandler.conf1.path=application1/logfileA
ejbserver.application.userlog.CJLogHandler.conf1.limit=10485760
ejbserver.application.userlog.CJLogHandler.conf1.count=2
ejbserver.application.userlog.CJLogHandler.conf1.level=SEVERE

# user-log logger function
ejbserver.application.userlog.Logger.com.example.userlogger1.handlers=com.hitachi.software.e
```

```
jb.application.userlog.CJMPMessageFileHandler;conf1
ejbserver.application.userlog.Logger.com.example.userlogger1.useParentHandlers=true
ejbserver.application.userlog.Logger.com.example.userlogger1.level=INFO
ejbserver.application.userlog.loggers=com.example.userlogger1
```

EJB クライアントアプリケーションでは，ユーザログを出力するためのハンドラとして，CJMPMessageFileHandler または CJMessageFileHandler を指定できます。使用するハンドラは，ejbserver.application.userlog.Logger.<ロガー名>.handlers キーに指定します。例の場合は，userlogger1 というロガーに，CJMPMessageFileHandler クラスを指定しています。

CJMPMessageFileHandler は，複数のプロセスから同時に同じファイルにログ出力できる機能を持つハンドラです。このため，EJB クライアントアプリケーションの複数のプロセスが出力するユーザログをまとめて出力できます。このハンドラは，EJB クライアントアプリケーションの場合だけ使用できるハンドラです。

なお，複数のプロセスから同時に同じファイルにログ出力しない場合は，J2EE アプリケーションのユーザログを出力する場合と同様に，CJMessageFileHandler も使用できます。CJMessageFileHandler を使用すると，CJMPMessageFileHandler を使用する場合に比べて，ログ出力性能が高くなります。

**注意事項**

CJMPMessageFileHandler クラスを使用すると，トレース共通ライブラリがログ管理に使用するファイルを「<ユーザログ出力ディレクトリ>/mmap/<ログファイル名のプレフィックス>.mm」に作成します。このユーザログ出力ディレクトリを使用している間は，このファイルを変更または削除しないでください。

## (2) EJB クライアント機能の初期化

EJB クライアント機能を初期化するメソッドが呼び出されて，ログ体系が構築されます。EJB クライアント機能は，次のどれかのタイミングで初期化されます。

- JNDI の初期コンテキスト生成（new InitialContext メソッド）
- セキュリティ機能 API でのログイン（LoginInfoManager クラスの login メソッド）
- 通信タイムアウト機能 API での通信タイムアウト設定用オブジェクトの取得（RequestTimeoutConfigFactory クラスの getRequestTimeoutConfig メソッド）

**注意事項**

初期化処理が失敗した場合，EJB クライアントアプリケーションのユーザログを出力する機能は使用できません。ただし，ユーザアプリケーションのソースコード上で，J2SE 標準の Handler クラスや Logger クラス，またはユーザが独自に作成した Handler クラスや Logger クラスの設定および構築をして，ログを出力することはできます。

### (3) Java ロギング API の実行

アプリケーション内の処理で，Java ロギング API が実行されて，ユーザログが出力されます。
CJMPMessageFileHandler を使用するときは，次の点に注意してください。

#### CJMPMessageFileHandler 使用時の注意

CJMPMessageFileHandler を使用する場合，メモリマップトファイルを使用しているため，実際にファイルへ反映されるまで遅延が発生することがあります。プロセスが終了するときにはファイルに反映されますが，長時間動作し続ける場合やファイルへの反映が遅延すると問題がある場合などは，flush を実行することをお勧めします。

flush を実行する方法には，次の二つの方法があります。

- Logger.getHandlers メソッドが返すすべての Handler に対して，flush メソッドを呼び出す。
- ejbserver.application.userlog.CJLogHandler.<ハンドラ名>.autoFlush.enabled プロパティを指定する。

ejbserver.application.userlog.CJLogHandler.<ハンドラ名>.autoFlush.enabled プロパティを指定する場合，flush は自動的に実行されます。このため，flush メソッドは使用しないでください。

## 8.11.4 EJB クライアントアプリケーションでのユーザログ出力の拡張

ユーザが作成した独自のクラス（Formatter，Filter，Handler）を EJB クライアントアプリケーションのユーザログ機能で利用するには，ユーザが作成した独自のクラスを EJB クライアントアプリケーションの JavaVM を開始するときのクラスパスに指定します。

## 8.11.5 ユーザ独自のフィルタ／フォーマッタ／ハンドラの使用方法

ユーザが作成した独自の Filter クラス，Formatter クラス，または Handler クラスを，EJB クライアントアプリケーションのユーザログ機能で使用する場合は，EJB クライアントアプリケーションの JavaVM を開始するときのクラスパスに，ユーザ作成のクラスを指定してください。

## 8.12　ユーザログ機能を使用する場合の注意事項

ここでは，ユーザログ機能を使用する場合の注意事項について説明します。

### 8.12.1　LogManager のカスタマイズについて

J2SE 標準の LogManager は，java.util.logging.config.class などのプロパティを使用してカスタマイズできます。ただし，Application Server が提供するユーザログ機能を使用する場合，カスタマイズはしないでください。ユーザログ機能で使用するプロパティを使用したログ体系の構築では，J2EE サーバの起動時に，ユーザログ機能が LogManager を使用してプロパティからログ構成を取得します。このため，LogManager をユーザがカスタマイズすると，ログ構成の構築に失敗するおそれがあります。

また，アプリケーションのソースコード上で LogManager の readConfiguration(InputStream ins)メソッドなどを実行して，ロガーの構成を初期化した場合も，ユーザログ機能が構築したログ構成が失われます。このため，このメソッドは実行しないでください。

ただし，カスタマイズした LogManager が，すでに構築されているログの構成（LogManager の内容）を完全に引き継いで，さらに独自の処理を追加した構造になっている場合は，カスタマイズ後もユーザログ機能を使用できます。

### 8.12.2　ユーザが作成したフィルタ・フォーマッタを使用する場合の注意

ログメッセージの出力時に，ハンドラに接続している，ユーザ作成のフィルタが例外をスローした場合，ハンドラの isLoggable メソッド※1 は，true※2 を返します。

ハンドラに接続している，ユーザ作成のフォーマッタが例外をスローした場合，ハンドラはフォーマッタで整形したメッセージを出力できません。ユーザが指定したメッセージは，フォーマッタで整形しないで出力されます。

ユーザ作成のフィルタ，フォーマッタがスローした例外の内容については，cjexception.log を参照してください。

注※1

isLoggable メソッドは，ログメッセージの取捨選択を判定するメソッドです。

注※2

true は，メッセージを出力対象とすることを意味します。

### 8.12.3　ロガーとハンドラとの接続

ロガーには複数のハンドラを接続できますが，複数のロガーに同一の設定を持つハンドラ（CJMessageFileHandler または CJMPMessageFileHandler）を接続することはできません。

### 8.12.4　EJB クライアントアプリケーションのログの出力モードの設定

EJB クライアントアプリケーションのログの出力方法には，2 種類のモードがあります。プロセスごとにログ出力先のサブディレクトリを作成する動作モードのことをサブディレクトリ専有モード，複数のプロセスでログ出力先のサブディレクトリを共有する動作モードのことをサブディレクトリ共有モードといいます。

サブディレクトリ専有モードは 06-50 よりも前のバージョンとの互換用に使用するモードであるため，EJB クライアントアプリケーションを新たに作成する場合は，サブディレクトリ共有モードの使用をお勧めします。

EJB クライアントアプリケーションのユーザログ機能を使用する場合，または cjclstartap コマンドで EJB クライアントアプリケーションを実行する場合は，サブディレクトリ共有モードを使用してください。

EJB アプリケーションのログの出力方法については，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「3.8　EJB クライアントアプリケーションのシステムログ出力」を参照してください。また，EJB アプリケーションのログを出力するサブディレクトリについては，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「7.6.1　アプリケーションのユーザログの取得」を参照してください。

### 8.12.5　usrconf.properties の接尾辞が「.level」で終わるキーについて

J2EE サーバの usrconf.properties で，接尾辞が「.level」で終わるキーのうち，値の範囲に SEVERE，WARNING，INFO，CONFIG，FINE，および FINEST 以外を持つキーが設定された場合は，次の現象が発生します。

1. サーバ起動時，java.util.logging.LogManager クラスがキーを読み込むときに接尾辞が「.level」で終わるキーの値をチェックします。値が範囲以外の場合は，java.util.logging.LogManager クラスが標準エラー出力へエラーメッセージを出力します。

   （例）sample.level=Error と指定されていた場合

   「Bad level value for property : sample.level」と出力されます。

2. ユーザログ機能の接尾辞が「.level」で終わるキーでは値が適切でない場合，1.と同様にエラーメッセージを出力します。

ただし，どちらの場合もメッセージが表示されるだけで，動作上は影響ありません。

# 9 CORBA/OTM ゲートウェイ機能

この章では，CORBA/OTM ゲートウェイ機能について説明します。

## 9.1　この章の構成

この章では，TPBroker OTM クライアント（OTM クライアント），および ORB クライアントから EJB アプリケーションを直接呼び出しできる CORBA/OTM ゲートウェイ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。

| 分類 | タイトル | 参照先 |
|---|---|---|
| 解説 | CORBA/OTM ゲートウェイ機能の概要 | 9.2 |
| 実装 | OTM アプリケーションの EJB 呼び出しに関する実装手順 | 9.3 |
| | ORB クライアントからの EJB 呼び出しに関する実装手順 | 9.4 |
| 解説 | CORBA システム例外発生時のトラブルシュート | 9.5 |

## 9.2 CORBA/OTM ゲートウェイ機能の概要

Cosminexus Component Transaction Monitor では，OTM クライアント，および ORB クライアントから EJB アプリケーションを直接呼び出しできるゲートウェイ機能を提供します。ORB クライアントは，次の三つのクライアントの総称です。

- TPBroker クライアント
- TPBroker 以外の CORBA2.1 準拠の CORBA クライアント
- TMS-4V/SP/Object Access の CORBA クライアント

### 9.2.1 クライアント

EJB アプリケーションを直接呼び出すことのできるクライアントの種類を次に示します。

表 9-1 ゲートウェイ機能でサポートするクライアント

| クライアント | Java | C++ | COBOL |
|---|---|---|---|
| OTM | ○※1 | ○ | × |
| TPBrokerV3 | ○ | ○ | − |
| TPBrokerV5 | ○ | ○ | × |
| CORBA2.1 準拠の CORBA クライアント | ○※2 | ○※2 | ○※2 |
| TMS-4V/SP/Object Access | − | − | ○ |

（凡例）

○：接続可能

×：接続不可

−：未サポート

注※1

OTMV3 と Cosminexus に含まれる TPBroker，または TPBrokerV5 を使用した Java アプリケーションから EJB アプリケーションを呼び出せません。

注※2

CORBA2.1 の仕様に準拠したクライアントであれば仕様上接続できますが，Cosminexus Component Transaction Monitor では接続を保証していません。ご使用になる場合は事前に接続検証をしてください。

### 9.2.2 データ型

ゲートウェイ機能では，IDL に記述できるデータ型を送受信できます。

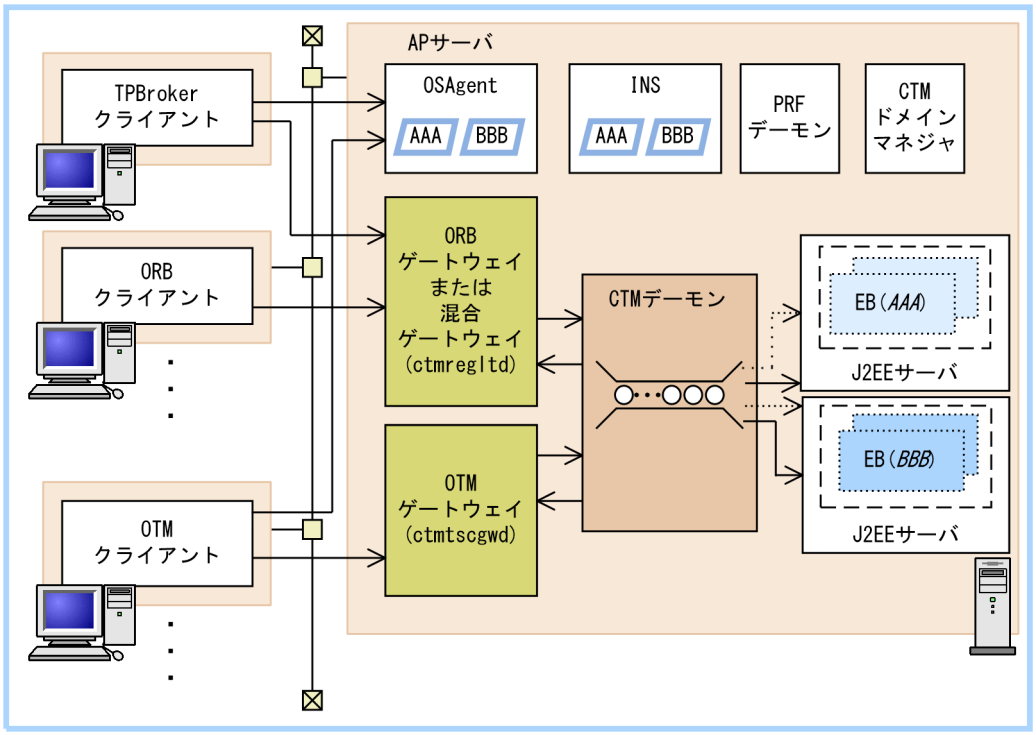
EJB のリモートインタフェースに記述できる引数は in 属性だけで，引数やリターン値に使える構造型は，Java の配列と Serializable インタフェースを実装したクラスだけです。また，Java が提供するオブジェクトを通信データとして扱うことはできません。

OTM からリクエストする場合，string と char の扱いに注意してください。また，CTM が提供する IDL 生成コマンドは，OTM で使用できる IDL を出力します。string と char の扱いについては「9.3.2　OTM で文字や文字列を扱う場合」を参照してください。IDL の詳細についてはマニュアル「TPBroker Object Transaction Monitor プログラマーズガイド」を参照してください。

## 9.2.3　プロセス構成

ORB クライアントは ORB ゲートウェイ経由または混合ゲートウェイ経由で，OTM クライアントは OTM ゲートウェイ経由でリクエストします。また，TPBroker クライアントおよび OTM クライアントは OSAgent を使用したリクエストも可能です。プロセス構成の概要を次の図に示します。

図 9-1　プロセス構成概要




## 9.2.4　ゲートウェイの開始方法

各種ゲートウェイに接続可能なクライアント，および開始コマンドとオプションの組み合わせを次の表に示します。

表 9-2　各種ゲートウェイに接続可能なクライアントおよび開始コマンドとオプションの組み合わせ

| ゲートウェイ種別 | | クライアント | コマンド | オプション | | |
|---|---|---|---|---|---|---|
| | | | | -CTMAgent 1 | -CTMIDLConnect | -CTMTSCGwStart |
| CTM レギュレータ | EJB レギュレータ | EJB | ctmstart | × | × | × |
| | 混合ゲートウェイ | EJB，ORB | ctmstart | △※ | △※ | × |
| ORB ゲートウェイ | | ORB | ctmstartgw | △ | △ | × |
| OTM ゲートウェイ | | OTM | ctmstart | × | × | ○ |

（凡例）

○：指定可

△：両方もしくは，どちらか一方を指定

×：指定不可

注※

-CTMRegOption に設定するファイルに指定してください。

EJB クライアントと ORB クライアントからのリクエストを同一のゲートウェイで受信したい場合は，CTM レギュレータ（混合ゲートウェイ）を開始してください。EJB クライアントと ORB クライアントからのリクエストをそれぞれ別々のゲートウェイで受信したい場合は，CTM レギュレータ（EJB レギュレータ）と ORB ゲートウェイを開始してください。

それぞれのコマンドとオプションの詳細については，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」を参照してください。

## 9.2.5　リファレンス解決と Lookup 名称の指定方法

ここではリファレンス解決と Lookup 名称の指定方法について説明します。

### (1)　クライアントが EJB アプリケーションを呼び出すためのリファレンス解決

クライアント種別ごとに，EJB アプリケーションを呼び出すためのリファレンス解決を次の表に示します。

表 9-3　EJB アプリケーションのリファレンス解決

| クライアント | リファレンス解決 | | |
|---|---|---|---|
| | CORBA Naming Service (INS) | OSAgent | IOR 文字列 |
| EJB | ○ | × | × |

| クライアント | | リファレンス解決 | | |
|---|---|---|---|---|
| | | CORBA Naming Service (INS) | OSAgent | IOR 文字列 |
| OTM | | × | ○ | × |
| ORB | TPBroker | × | ○ | ○ |
| | TPBroker 以外 | × | × | ○ |

（凡例）

○：サポート

×：未サポート

## (2) Lookup 名称の指定方法

OTM クライアントの場合

OTM クライアントでは，TSC ユーザプロキシを生成するコンストラクタの引数に，EJB の Lookup（登録）名称を TSC アクセプタ名称として指定します。なお，TSC アクセプタ名称は省略できません。また，接続形態は，TSC レギュレータ経由接続を選択してください。

**EJB アプリケーションの Lookup 名称が「AAA/a1」の例**

＜ C++＞

```
Converter_TSCprxy(TSCClient_ptr _tsc_client, ”AAA/a1”);
```

＜ Java ＞

```
Converter_TSCprxy(TSCClient _tsc_client, new String(”AAA/a1”));
```

なお，アプリケーションの作成手順は，「9.3.1 OTM アプリケーションの作成手順」を参照ください。

TPBroker クライアントの場合

ctmstartgw コマンド，または，ctmregltd コマンドに「-CTMAgent 1」を指定すると，EJB の Lookup 名称をオブジェクト名称として，OSAgent に CORBA リファレンスを登録します。

そのため，TPBroker クライアントでは，EJB の Lookup 名称を_bind()の引数に指定して参照を解決します。

**converter サンプルのデフォルト Lookup 名称の例**

＜ C++＞

```
Converter::_bind(”HITACHI_EJB/SERVERS/MyServer/EJB/converter/MyConverter”);
```

＜ Java ＞

```
ConverterHelper::_bind(org.omg.CORBA.ORB orb,
      new String(”HITACHI_EJB/SERVERS/MyServer/EJB/converter/MyConverter”));
```

なお，アプリケーションの作成手順は，「9.4 ORB クライアントからの EJB 呼び出しに関する実装手順」を参照ください。

## 9.3　OTM アプリケーションの EJB 呼び出しに関する実装手順

アプリケーションの作成手順について，Cosminexus Component Container のサンプルプログラム「converter」を例に説明します。

### 9.3.1　OTM アプリケーションの作成手順

1. EJB アプリケーション（サーバ AP）を作成する。

EJB アプリケーション（サーバ AP）の作成には，次の Java プログラムが必要です。

- Converter.java：EJBObject 継承クラス
- ConverterEJB.java：SessionBean 実装クラス
- ConverterHome.java：EJBHome 継承クラス

サンプルで提供する compileBean.bat を使用して，Enterprise Bean をコンパイルすると，次のファイルが生成されます。

- Converter.class
- ConverterEJB.class
- ConverterHome.class
- converter.jar

詳細については，マニュアル「Cosminexus アプリケーション開発ガイド」を参照してください。

2. IDL ファイルを生成する。

手順 1 で生成された class ファイルから，CTM が提供する IDL 生成コマンドを使用して IDL ファイルを生成します。

% ctmjava2idl -g -o Converter.idl Converter.class

- Converter.id
- TSCjava.idl

この操作は Cosminexus 環境で実行してください。また，TPBrokerV5 の java2idl が必要になるため，PATH 環境変数には Cosminexus TPBroker の bin ディレクトリを設定してください。

3. スタブを生成する。

手順 2 で生成された IDL から，CTM が提供するスタブ生成コマンドでスタブを生成します。

この操作はクライアント開発環境で実行してください。また，クライアント AP を実行する OTM とその前提の TPBroker の環境を設定してください。

＜ C++アプリケーションの場合＞

% ctmidl2cpp -gen_included_files Converter.idl

これによってクライアント AP の作成に必要な次のファイルが生成されます。

・Converter_c.hh
・Converter_c.cc
・Converter_s.hh
・Converter_s.cc
・Convtrter_TSC_c.hh
・Converter_TSC_c.cc

＜Java アプリケーションの場合＞

% ctmidl2j -gen_included_files Converter.idl

これによってクライアント AP の作成に必要な次のファイルが生成されます。

・Converter.java
・ConverterHelper.java
・ConverterOperations.java
・ConveterPOA.java（なし※）
・_Converter_Stub.java（_st_Converter.java※）
・_Converter_Tie.java（_tie_Converter.java※）
・Converter_TSCprxy.java
・その他構造型データのクラス

注※　TPBrokerV3 を前提とする場合に出力されるファイルです。

4. コンパイルとリンケージ

OTM アプリケーション開発手順に従い，手順 3 で生成したスタブファイルを使用して，コンパイルとリンケージを実行してください。

## 9.3.2　OTM で文字や文字列を扱う場合

OTM で文字列を扱う場合のデータ型を次の表に示します。

表 9-4　OTM で扱うデータ型

| 項番 | リモートインタフェースでの定義 | クライアントで使用するデータ型 |
|---|---|---|
| 1 | string | ::TSC::TSCWStringValue |
| 2 | char | ::TSC::TSCWChar |

## (1)　::TSC::TSCWStringValue

::TSC::TSCWStringValue は IDL で次のように定義しています。

```
module TSC {
  typedef sequence<octet> TSCWString;
```

```
  struct TSCWStringValue {
    TSCWString value;
  };
};
```

実際に wchar_t*の文字列として使用するために，次の機能を提供します。

### (a) C++インタフェース

- インクルードファイル

  #include “Converter_TSC_c.hh”（スタブヘッダ）

- 提供関数

  void TSCsetWString( ::TSC::TSCWString&, const ::CORBA::WChar* )

  【用途】

  ワイド文字列を::TSC::TSCWString に設定します。

  【IO】

  ::TSC::TSCWString& :out ：設定する::TSC::TSCWString 型の変数

  const ::CORBA::WChar*:in ：設定したいワイド文字列

  ::CORBA::WChar* TSCgetWString( ::TSC::TSCWString )

  【用途】

  ::TSC::TSCWString に設定されているワイド文字列を取り出します。

  【IO】

  ::TSC::TSCWString :in ：ワイド文字列を持つオブジェクト

  【戻り値】

  ::TSC::TSCWString から取り出したワイド文字列

  【注意事項】

  OTM V3 で使用する場合，戻り値は delete[]を使用して解放してください。

- サンプルコード

```
CORBA::WChar* wstr_data = new CORBA::WChar[5];
wstr_data[0] = L’さ’;
wstr_data[1] = L’ん’;
wstr_data[2] = L’ぷ’;
wstr_data[3] = L’る’;
wstr_data[4] = 0;
::TSC::TSCWStringValue tsc_wstr_value_data;

// ワイド文字列の設定
TSCsetWString( tsc_wstr_value_data.value, wstr_data );
// 使い終わった領域の解放
delete[] wstr_data;

// ワイド文字列の取り出し
wstr_data = TSCgetWString( tsc_wstr_data.value );
```

```
// 取り出した領域の解放
delete[] wstr_data;
```

### (b) Java インタフェース

Java インタフェースは提供されません。

- サンプルコード

```
String wstr_data = new String(”さんぷる”);
System.out.println(wstr_data);
TSC.TSCWStringValue tsc_wstr_value_data = new TSC.TSCWStringValue();

// ワイド文字列の設定
tsc_wstr_value_data.value = wstr_data.getBytes(”UTF-16”);

// ワイド文字列の取り出し
wstr_data = new String( tsc_wstr_value_data.value );
```

## (2) ::TSC::TSCWChar

::TSC::TSCWChar は IDL で次のように定義しています。

```
module TSC {
  typedef octet TSCWChar[3];
};
```

実際に wchar_t の文字として使用するために，次の機能を提供します。

### (a) C++インタフェース

- インクルードファイル

  #include “Converter_TSC_c.hh”（スタブヘッダ）

- 提供関数

  void TSCsetWChar( ::TSC::TSCWChar , ::CORBA::WChar )

  【用途】

  ワイド文字を::TSC::TSCWChar に設定します。

  【IO】

  ::TSC::TSCWChar :out: 設定する領域

  const ::CORBA::WChar :in : 設定したいワイド文字

  【戻り】

  なし

  ::CORBA::WChar TSCgetWChar( ::TSC::TSCWChar )

  【用途】

  ::TSC::TSCWChar に設定されているワイド文字を取り出します。

【IO】

const ::TSC::TSCWChar :in :ワイド文字を持つオブジェクト

【戻り】

::TSC::TSCWChar から取り出したワイド文字

- サンプルコード

```
::TSC::TSCWChar tsc_wchar_data;
// ワイド文字の設定
TSCsetWChar( tsc_wchar_data, L'あ' );

// ワイド文字の取り出し
CORBA::WChar wchar_data = TSCgetWChar( tsc_wchar_data );
```

### (b) Java インタフェース

Java インタフェースは提供されません。

- サンプルコード

```
TSC.TSCWCharHolder tsc_wchar_data = new TSC.TSCWCharHolder();
tsc_wchar_data.value = new byte[3];
char wch_data = 'あ';

// ワイド文字の設定
tsc_wchar_data.value[0] = (byte)2;
tsc_wchar_data.value[1] = (byte)((wch_data >> 8) & 0xff);
tsc_wchar_data.value[2] = (byte)( wch_data & 0xff);

// ワイド文字の取り出し
wch_data = (char)(((char)tsc_wchar_data.value[1] & 0xff) << 8 |
((char)tsc_wchar_data.value[2] & 0xff));
```

## 9.3.3 OTM クライアントで例外を参照する方法

OTM クライアントは，EJB からスローされる例外クラスをそのままキャッチできないので，TSC ユーザプロキシ（スタブ）で変換してからキャッチしています。

以降では TSC ユーザプロキシで変換した例外について説明します。

### (1) OTM クライアントで受け取る例外一覧

EJB で発生した例外と OTM クライアントで受け取る例外の対応一覧を，次の表に示します。

表 9-5 OTM クライアントで受け取る例外一覧

| 項番 | EJB で発生した例外 | OTM クライアントで受け取る例外 |
|---|---|---|
| 1 | Java ユーザ定義例外 | ユーザ定義例外（IDL 定義の exception を参照）※ |

| 項番 | EJB で発生した例外 | OTM クライアントで受け取る例外 |
|---|---|---|
| | （java.lang.Exception を継承） | |
| 2 | Java ユーザ定義例外<br>（java.rmi.RemoteException を継承） | TSC::java::rmi::RemoteEx 例外 |
| 3 | Java システム定義例外<br>（java.lang.Exception を継承した例外） | TSCUnknown 例外（該当 Java システム定義例外） |
| 4 | Java ランタイム例外<br>（java.lang.RuntimeException を継承した例外） | TSC::java::rmi::RemoteEx 例外 |
| 5 | Java システム定義エラー<br>（java.lang.Error を継承したエラー） | TSC::java::rmi::RemoteEx 例外 |
| 6 | java.rmi.RemoteException | TSC::java::rmi::RemoteEx 例外 |
| 7 | CORBA システム例外<br>（org.omg.CORBA.SystemException を継承した例外） | TSCUnknown 例外（該当 TSC システム例外） |
| 8 | CORBA ユーザ例外<br>（org.omg.CORBA.UserException を継承した例外） | ユーザ定義例外（IDL 定義の exception を参照） |

注※

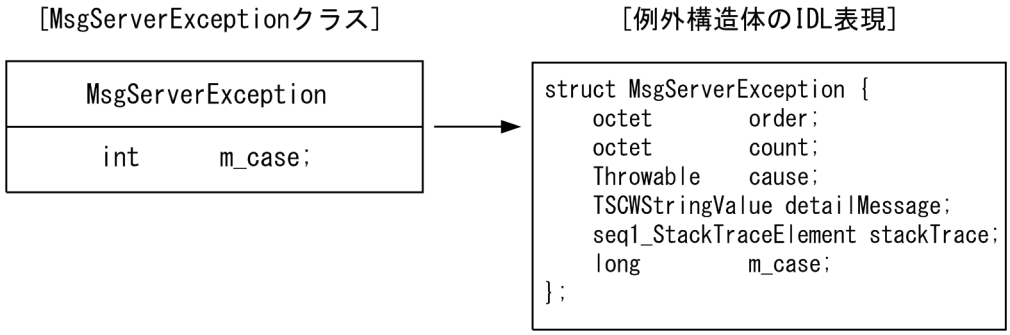
例外名称が～Exception の場合～Ex になります。そのほかの場合，末尾に Ex が付加されます。例外のメンバ value の値が例外構造体。

## (2)　例外構造体について

### (a)　例外構造体の概要

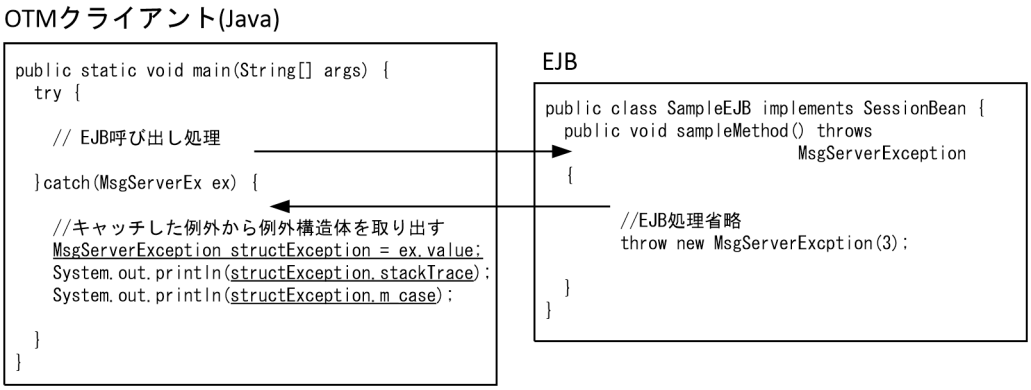
OTM クライアントでは再帰的な構造のデータを解釈できない，valuetype が使用できないという制限があるため，EJB からの例外は次の図のように構造体をメンバに持つ例外に変換されます。この構造体を例外構造体と呼びます。

図 9-2　例外クラスと例外構造体




例外は例外構造体に変換されますが，OTM クライアントでは，例外構造体をメンバとして持った例外をキャッチします。ユーザが MsgServerException を EJB からスローすると，OTM クライアントは MsgServerEx 例外としてキャッチします。MsgServerEx 例外のメンバである value は，例外構造体 MsgServerException になります。

図 9-3　例外クラスからの例外構造体の取得




## (b)　例外構造体のメンバ

例外構造体に含まれるメンバ名と内容を次の表に示します。

表 9-6　例外構造体のメンバの説明

| 項番 | 構造体のメンバ | 内容 |
|---|---|---|
| 1 | order | OTM アプリケーションの開発者は意識する必要はありません。<br>OTM クライアントが例外として受け取るために必要な情報です。 |
| 2 | count | |
| 3 | cause | OTM アプリケーションの開発者は意識する必要はありません。<br>OTM クライアントの場合は，EJB から cause に値を設定して例外をスローしないでください。 |
| 4 | detailMessage | スーパクラスに指定した詳細メッセージです。 |
| 5 | stackTrace | EJB 側の例外が生成されるまでのスタックトレースです。 |
| 6 | メンバ変数 | Java ユーザ定義例外で設定したメンバ変数です。<br>RemoteException の場合は，メンバ変数はありません。 |

## (c)　スタックトレースの各情報

例外構造体のメンバである stackTrace は，StackTraceElement 構造体の配列であり，
StackTraceElement 構造体には，スタックトレースの 1 スタック分の情報が入っています。

StackTraceElement 構造体の各メンバに設定される情報を，次の表に示します。

表 9-7　StackTraceElement 構造体のメンバ

| 項番 | フィールド名 | 内容 | 例 |
|---|---|---|---|
| 1 | LineNumber | ソースの行番号 | 29 |
| 2 | declaringClass | クラス名 | jp.co.Hitachi.soft.test.ejb.testMsgSyncServiceDelEJB |
| 3 | FileName | ファイル名 | testMsgSyncServiceDelEJB.java |

| 項番 | フィールド名 | 内容 | 例 |
|---|---|---|---|
| 4 | MethodName | メソッド名 | InvokeBinary |

## (3) OTM クライアントで例外情報を参照する例

OTM クライアントで，EJB からの例外を参照する方法について，サンプルコードを用いて説明します。

Java ユーザ定義例外を次に示します。

```
Package jp.co.Hitachi.soft.test;
public class MsgServerException extends Exception
{
    public MsgServerException()
    {
      m_case=1;
    }
    private int m_case;
}
```

### (a) C++で例外情報を参照する例

C++で例外情報を参照する例を次に示します。

```
#define ERR_FORMAT ”EC=%d,DC=%d,PC=%d,CS=%d,MC1=%d,MC2=%d,MC3=%d,MC4=%d¥n”
// 指定された文字列の長さを返す関数
int my_wstringlen(CORBA::WChar* arg){
    int i;
    for(i=0;arg[i] != 0;i++);
    return i;
}
// 指定された文字列を表示する関数
void my_print_wstring(CORBA::WChar* arg){
    for(int i=0;i < my_wstringlen(arg); i++){
      printf(”%c”, arg[i]);
    }
}

int main(int argc, char** argv){
  try {
      // EJB呼び出し処理省略
  }catch(jp::co::Hitachi::soft::test::MsgServerEx &e){
   // EJBがJavaユーザ定義例外 MsgServerExceptionをスロー
    jp::co::Hitachi::soft::test::CSCMsgServerException& ex_val = e.value;

    // 詳細メッセージの出力
    printf(”detailMessage:”);
    ::TSC::TSCWStringValue& detailMessage = ex_val.detailMessage;
    CORBA::WChar* w_detail_msg = new CORBA::WChar[detailMessage.value.length()];
    w_detail_msg = TSCgetWString(detailMessage.value );
    my_print_wstring(w_detail_msg);
    printf(”¥n”);

    // メンバ変数 m_case の出力
```

```
    printf("m_case:%d", ex_val.m_case);
    // スタックトレースの出力
    ::TSC::java::lang::seq1_StackTraceElement& stackTrace = ex_val.stackTrace;
    CORBA::ULong len = stackTrace.value.length();
    :TSC::java::lang::StackTraceElement stackTraceElement;
    for ( int i = 0 ; i < len ; i ++ ) {
        stackTraceElement = stackTrace.value [ i ] ;
        ::TSC::TSCWStringValue& className = stackTraceElement.declaringClass;
        ::TSC::TSCWStringValue& methodName= stackTraceElement.methodName;
        ::TSC::TSCWStringValue& fileName = stackTraceElement.fileName;
        CORBA::Long lineNumber= stackTraceElement.lineNumber;

        CORBA::WChar* w_class_name = new CORBA::WChar[className.value.length()];
        w_class_name = TSCgetWString( className.value );

      CORBA::WChar* w_method_name = new CORBA::WChar[methodName.value.length()];
      w_method_name = TSCgetWString( methodName.value );

      CORBA::WChar* w_file_name = new CORBA::WChar[fileName.value.length()];
      w_file_name = TSCgetWString( fileName.value );

      printf("at ");
      my_print_wstring(w_class_name);
      printf(".");
      my_print_wstring(w_method_name);
      printf("(");
      my_print_wstring(w_file_name);
      printf(":%d)¥n",lineNumber);
    }
}catch(TSC::java::rmi::RemoteEx &e){
    // EJBからJavaランタイム例外がスロー
    // EJBからJavaシステム定義エラーがスロー
    // EJBからjava.rmi.RemoteExceptionがスロー

    TSC::java::rmi::RemoteException& ex_val = e.value;

    // 詳細メッセージの出力
    printf("detailMessage:");
    ::TSC::TSCWStringValue& detailMessage = ex_val.detailMessage;
    CORBA::WChar* w_detail_msg = new CORBA::WChar[detailMessage.value.length()];
    w_detail_msg = TSCgetWString(detailMessage.value );
    my_print_wstring(w_detail_msg);
    printf("¥n");

    // スタックトレースの出力
    ::TSC::java::lang::seq1_StackTraceElement& stackTrace = ex_val.stackTrace;
    CORBA::ULong len = stackTrace.value.length();
    :TSC::java::lang::StackTraceElement stackTraceElement;
    for ( int i = 0 ; i < len ; i ++ ) {
        stackTraceElement = stackTrace.value [ i ] ;
        ::TSC::TSCWStringValue& className = stackTraceElement.declaringClass;
        ::TSC::TSCWStringValue& methodName= stackTraceElement.methodName;
        ::TSC::TSCWStringValue& fileName = stackTraceElement.fileName;
        CORBA::Long lineNumber= stackTraceElement.lineNumber;

        CORBA::WChar* w_class_name = new CORBA::WChar[className.value.length()];
        w_class_name = TSCgetWString( className.value );
      CORBA::WChar* w_method_name = new CORBA::WChar[methodName.value.length()];
```

```
      w_method_name = TSCgetWString( methodName.value );

      CORBA::WChar* w_file_name = new CORBA::WChar[fileName.value.length()];
      w_file_name = TSCgetWString( fileName.value );

      printf("at ");
      my_print_wstring(w_class_name);
      printf(".");
      my_print_wstring(w_method_name);
      printf("(");
      my_print_wstring(w_file_name);
      printf(":%d)¥n",lineNumber);
    }
} catch(TSCSystemException& se) {
    // EJBからJavaシステム定義例外がスロー
    // EJBからCORBAシステム例外がスロー
    // EJBの呼び出しに失敗
    printf(ERR_FORMAT,
            se.getErrorCode(), se.getDetailCode(),
            se.getPlaceCode(), se.getCompletionStatus(),
            se.getMaintenanceCode1(), se.getMaintenanceCode2(),
            se.getMaintenanceCode3(), se.getMaintenanceCode4());
} catch(UserExcept& se) {
    // EJBからCORBAユーザ例外がスロー
    printf("UserExcept¥n");
}
```

## (b) Java で例外情報を参照する例

Java で例外情報を参照する例を次に示します。

```
// byte[]をchar[]に変換し，Stringにする関数
private static String myString(byte[] barray) {
    char[] carry = new char[barray.length/2];
    for ( int i=0;i<carry.length;i++) {
        carry[i] = (char)((( barray[i*2]& 0xff) << 8 ) | barray[(i*2)+1]& 0xff) ;
    }
    return new String(carry);
}

public static void main(String[] args) {
  try {
      // EJB呼び出し処理省略
  }catch(jp.co.Hitachi.soft.test.MsgServerEx ex) {

     // EJBがJavaユーザ定義例外 MsgServerExceptionをスロー
    jp.co.Hitachi.soft.csc.msg.message.reception.MsgServerException testException = ex.value
;

    // 詳細メッセージの出力
    System.out.println ("detailMessage:"+ myString(testException.detailMessage);

    // メンバ変数 m_case の出力
    System.out.println ("m_case:%d"+ testException .m_case);

    // スタックトレースの出力
```

```
    TSC.java.lang.StackTraceElement[] stackElements= testException.stackTrace.value;
    java.lang.StringBuffer stb = new java.lang.StringBuffer();

    for (int i=0;i<stackElements.length;i++) {
        stb = stb.append("at ");
        stb = stb.append(myString(stackElements[i].declaringClass.value));
        stb = stb.append(".");
        stb = stb.append(myString(stackElements[i].methodName.value));
        stb = stb.append("(");
        stb = stb.append(myString(stackElements[i].fileName.value));
        stb = stb.append(":");
        stb = stb.append(stackElements[i].lineNumber);
        stb = stb.append(")");
        stb = stb.append("¥n");
    }
        System.out.println(stb.toString());
}catch(TSC.java.rmi.RemoteEx ex) {
    // EJBからJavaランタイム例外がスロー
    // EJBからJavaシステム定義エラーがスロー
    // EJBからjava.rmi.RemoteExceptionがスロー
    TSC.java.rmi.RemoteException testException = ex.value;

    // 詳細メッセージの出力
    System.out.println ("detailMessage:"+ myString(testException.detailMessage);
    // スタックトレースの出力
    TSC.java.lang.StackTraceElement[] stackElements= testException.stackTrace.value;
    java.lang.StringBuffer stb = new java.lang.StringBuffer();

    for (int i=0;i<stackElements.length;i++) {
        stb = stb.append("at ");
        stb = stb.append(myString(stackElements[i].declaringClass.value));
        stb = stb.append(".");
        stb = stb.append(myString(stackElements[i].methodName.value));
        stb = stb.append("(");
        stb = stb.append(myString(stackElements[i].fileName.value));
        stb = stb.append(":");
        stb = stb.append(stackElements[i].lineNumber);
        stb = stb.append(")");
        stb = stb.append("¥n");
    }
        System.out.println(stb.toString());
} catch(TSCSystemException tsc_se) {
    // EJBからJavaシステム定義例外がスロー
    // EJBからCORBAシステム例外がスロー
    // EJBの呼び出しに失敗
    System.out.println(tsc_se);
} catch(UserExcept tsc_se) {
    // EJBからCORBAユーザ例外がスロー
    System.out.println("catch" + tsc_se.value);
}
```

## 9.4　ORB クライアントからの EJB 呼び出しに関する実装手順

この節では ORB クライアントからの EJB 呼び出しに関する実装手順について説明します。

### 9.4.1　TPBrokerV5 アプリケーションの作成手順

ここでは TPBrokerV5 を使用した場合のアプリケーションの作成手順について説明します。

TPBrokerV5 を除く ORB クライアントのアプリケーションの作成手順については「9.4.2　TPBrokerV5 を除く ORB クライアントアプリケーションの作成手順」を参照してください。

1. EJB アプリケーション（サーバ AP）を作成する。

   次の Java プログラムが必要です。

   - Converter.java ：EJBObject 継承クラス
   - ConverterEJB.java ：SessionBean 実装クラス
   - ConverterHome.java ：EJBHome 継承クラス

   サンプルで提供する compileBean.bat を使用して，Enterprise Bean をコンパイルすると，次のファイルが生成されます。

   - Converter.class
   - ConverterEJB.class
   - ConverterHome.class
   - converter.jar

   詳細については，マニュアル「Cosminexus アプリケーション開発ガイド」を参照してください。

2. IDL ファイルを生成する。

   C++アプリケーションを作成する場合，手順 1 で生成された class ファイルから，TPBroker が提供する次のコマンドで IDL ファイルを生成します。

   % java2idl Converter.class > Converter.idl

3. スタブを生成する。

   手順 2 で生成された IDL または，class ファイルからスタブを生成します。

   この操作はクライアント開発環境で実行してください。

   ＜ C++アプリケーションの場合＞

   % idl2cpp -namespace Converter.idl

   これによってクライアント AP の作成に必要な次のファイルが生成されます。

   ・Converter_c.cc

   ・Converter_c.hh

・Converter_s.cc

・Converter_s.hh

＜Java アプリケーションの場合＞

% java2iiop Converter.class

これによってクライアント AP の作成に必要な次のファイルが生成されます。

・Converter.java

・ConverterHelper.java

・ConverterHolder.java

・ConverterOperations.java

・ConveterPOA.java

・_Converter_Stub.java

・_Converter_Tie.java

・その他構造型データのクラス

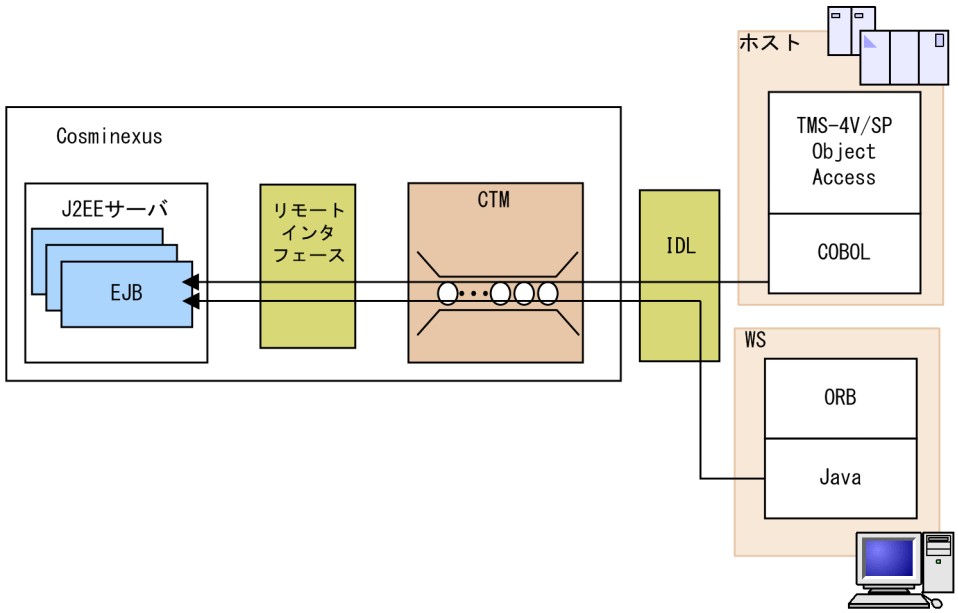
4. コンパイルとリンケージ

TPBroker アプリケーション開発手順に従い，手順 3 で生成したスタブファイルを使用して，コンパイルとリンケージを実行してください。

## 9.4.2　TPBrokerV5 を除く ORB クライアントアプリケーションの作成手順

TPBrokerV5 を除く ORB クライアント（CORBA2.1 準拠の CORBA クライアントまたは TMS-4V/SP Object Access，TPBrokerV3）から EJB アプリケーション（Cosminexus）の EJB を CORBA 通信によって呼び出す方式を次の図に示します。

図 9-4　ORB クライアントからの EJB 呼び出し



この方式では IDL として次のデータ型/データ定義だけ使用できます。

long，unsigned long，string，sequence<octet>，struct

## (1) インタフェース設計手順

アプリケーションサーバ側の通信インタフェースは EJB のリモートインタフェースとして，ORB クライアント側の通信インタフェースは CORBA の IDL としてそれぞれ記述する必要があります。

リモートインタフェースおよび IDL の設計手順の概略は次のようになります。

1. インタフェースのベースとなる IDL 定義を作成
2. IDL に合わせて EJB のリモートインタフェースを作成
3. ctmjava2idl コマンドの出力によって，IDL の構造体メンバの並びを補正
4. string,sequence<octet>,struct の IDL 定義を変更

各手順の詳細について説明します。

### (a) インタフェースのベースとなる IDL 定義の作成

インタフェースのベースとなる IDL 定義は，次の注意事項を考慮して作成してください。

- out/inout 引数は使用できません（戻りデータとしてはリターン値を使用します。戻りデータが複数ある場合は struct にまとめてください）。
- ユーザ例外は使用しないでください。
- struct のメンバは long,unsigned long,string,sequence<octet>だけを使用してください。
- #pragma は使用しないでください。

### (b) リモートインタフェースの作成

ベースとなる IDL 定義の作成後，それを基に EJB のリモートインタフェースを作成します。

このとき，対応するメンバ名は，IDL とリモートインタフェースで同じにしてください。

また，IDL の module は，Java のパッケージに対応します。

サポートするデータ型のマッピングを次の表に示します。

表 9-8 IDL からリモートインタフェースへのデータ型のマッピング

| データ型 | IDL | リモートインタフェース |
|---|---|---|
| 整数 | long | int |
| | unsigned long※1 | |
| 文字列 | string※3 | byte[]※2 |

| データ型 | IDL | リモートインタフェース |
|---|---|---|
| バイナリ | sequence<octet>※3 | byte[] |
| 構造体 | struct | class（Serializable を実装） |

注※1

Java には unsigned に対応する型が存在しません。そのため IDL の unsigned long は long と同じ型として扱われます。

注※2

Java の String は IDL の Wstring に対応するため，リモートインタフェースでは byte[]を使用してください。

注※3

IDL 上の string/sequence<octet>に対して最大長を指定できます。ただし Java 側ではすべて可変長データとして扱われます。

IDL の記述例

```
struct AAA {
  long longData;
  string<4> strData;
  unsigned long ulongData;
  sequence<octet> octseqData;
};
```

リモートインタフェースの記述例

```
public class AAA
  implements java.io.Serializable {
  public int longData;
  public byte[] strData;
  public int ulongData;
  public byte[] octSeqData;
};
```

### (c) ctmjava2idl コマンドの出力によって，IDL の構造体メンバの並びを補正

EJB に対する CORBA 通信では，IDL の struct 内のメンバの並びを Java の Serializable の仕様に沿った形で並べ替える必要があります。ここで，IDL 上での適正な struct メンバの並びを確認するために，ctmjava2idl コマンドを使用します。

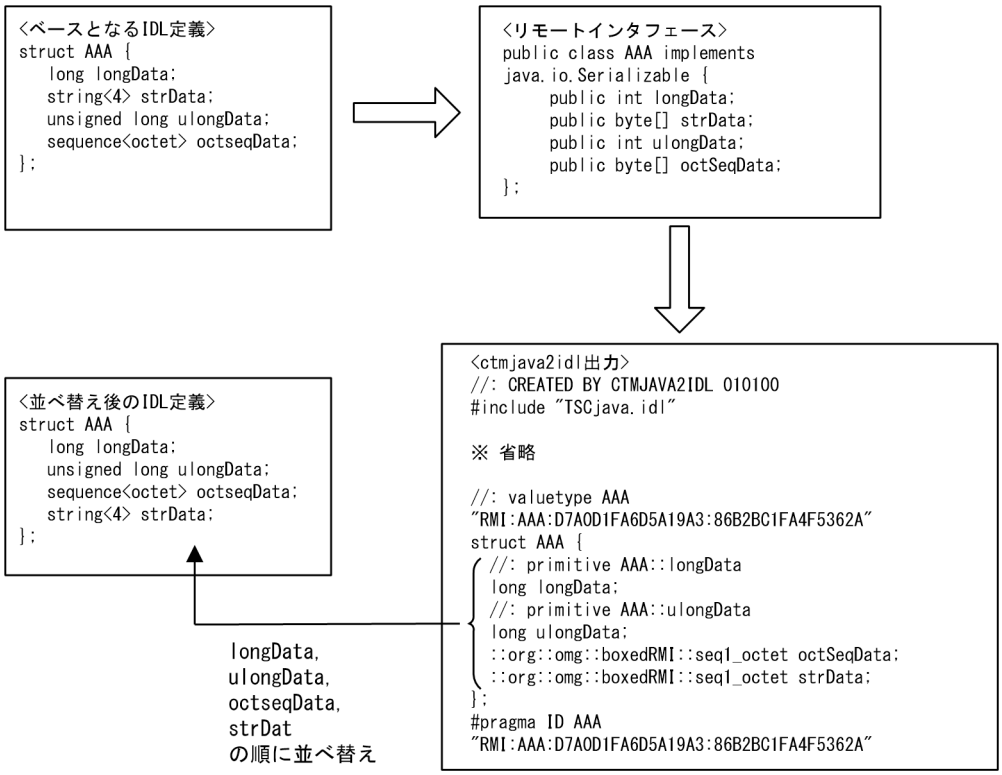
ctmjava2idl コマンドは，EJB のリモートインタフェースを IDL 定義へ変換するコマンドです。

<例：表 9-8 の例で示した構造体 AAA を使用した場合の手順>

1. AAA に対応する Java ソースプログラム（AAA.java）をコンパイルする。
   % javac AAA.java
2. 生成されたクラスファイル（AAA.class）を ctmjava2idl コマンドに適用する。
   % ctmjava2idl AAA.class
3. 標準出力に出力された IDL 定義を基に，ベースとなる IDL を修正する。

ctmjava2idl コマンドによる IDL の構造体メンバの並びの変更例を，次の図に示します。

図 9-5　ctmjava2idl コマンドによる IDL の構造体メンバの並びの変更の例




**参考**

struct メンバの並べ替えルール

リモートインタフェース上でのデータ型とメンバ名によって順番が決定されます。

1. int のメンバは byte[]のメンバより先に並びます。
2. int 型のメンバ同士では，メンバ名によって文字列順に並びます。
3. byte[]型のメンバ同士では，メンバ名によって文字列順に並びます。
4. int 型のメンバ同士，および，byte[]のメンバ同士は ctmjava2idl コマンドで出力した IDL 定義の struct に定義されている順番に並べます。

## (d)　string，sequence<octet>，struct の IDL 定義を変更

ORB クライアントからの EJB 呼び出しでは string，sequence<octet>，struct のデータを特定の形式の構造体に変更して扱う必要があります（long/unsigned long はそのまま）。

### ORB クライアント側送信データの形式

ORB クライアント側から EJB に対し送信するデータに対しては，次のような形式に変更します。

（例：「図 9-6　EJB 向けの IDL への変換例」の invoke メソッドの in 引数 StringValue,OctSeqValue,AAARequest）

```
struct 任意の構造体名 {
  long VALUE_TAG;
```

```
  string REPOSITORY_ID;
  データ本体;
}
```

ただし struct 内の string,sequence<octet>メンバに対しては，上記構造体を使用する代わりに，次の構造体を各データの直前に配置するようにしてください。

（例：「図 9-6　EJB 向けの IDL への変換例」の AAARequest のメンバ tag1 および tag2）

```
struct ValueTag {
  long VALUE_TAG;
  string REPOSITORY_ID;
}
```

ORB クライアント側のプログラム実装では，VALUE_TAG/REPOSITORY_ID に値を設定する必要があります。

### ORB クライアント側受信データの形式

戻り値としての struct そのものに対しては，送信データと同様の形式に変更します。

（例：「図 9-6　EJB 向けの IDL への変換例」の AAAReply）

```
struct 任意の構造体名 {
  long VALUE_TAG;
  string REPOSITORY_ID;
  データ本体;
}
```

struct 内のメンバの内，最初に出現した string,sequence<octet>に対して次の構造体をデータの直前に配置してください。

（例：「図 9-6　EJB 向けの IDL への変換例」の AAAReply のメンバ tag1）

```
struct ValueTag {
  long VALUE_TAG;
  string REPOSITORY_ID;
}
```

二つ目以降に出現した string,sequence<octet>メンバに対しては，次の構造体を各データの直前に配置してください。

（例：「図 9-6　EJB 向けの IDL への変換例」の AAAReply のメンバ tag2 。手前に sequence<octet>のメンバがあるため，ValueTag でなく ReplyTag になる。）

```
struct ReplyTag {
  long VALUE_TAG;
  long INDIRECTION_TAG;
  long INDIRECTION;
}
```

受信データとしては，VALUE_TAG/REPOSITORY_ID/ INDIRECTION_TAG/INDIRECTION の値は無視してください。

### IDL の変更例

IDL の変更例を次の図に示します（対応するリモートインタフェースも示します）。

図 9-6　EJB 向けの IDL への変換例

＜変更前＞

```
typedef sequence<octet>
octetSequence;

struct AAA {
  long longData;
  unsigned long ulongData;
  octetSequence octseqData;
  string<4> strData;
};
interface XXX {
  AAA invoke(
    in long data1,
    in string string1,
    in string string2,
    in octetSequence octetSeq1,
    in AAA struct1
    );
};
```

＜参考：リモートインタフェース＞

```
import javax.ejb.EJBObject;
import
java.rmi.RemoteException;

public class AAA implements
       java.io.Serializable {
  public int longData;
  public byte[] strData;
  public int ulongData;
  public byte[] octSeqData;
};
public interface XXX extends
EJBObject {
  public AAA invoke(
       int data1,
       byte[] string1,
       byte[] string2,
       byte[] octetSeq1,
       AAA struct1
       ) throws
RemoteException;
};
```

＜変更後＞

```
typedef sequence<octet> octetSequence;
struct ValueTag {
  long              VALUE_TAG;
  string            REPOSITORY_ID;
};
struct ReplyTag {
  long              VALUE_TAG;
  long              INDIRERCTION_TAG;
  long              INDIRECTION;
};
struct StringValue {
  long              VALUE_TAG;
  string            REPOSITORY_ID;
  string            data;
};
struct OctSeqValue {
  long              VALUE_TAG;
  string            REPOSITORY_ID;
  octetSequence     data;
};
struct AAARequest { //送信用
  long                          VALUE_TAG;
  string                        REPOSITORY_ID;
  long                          longData;
  unsigned long                 ulongData;
  ValueTag                      tag1;
  octetSequence                 octseqData;
  ValueTag                      tag2;
  String<4>                     strData;
};
struct AAAReply { //受信用
  long                          VALUE_TAG;
  string                        REPOSITORY_ID;
  long                          longData;
  unsigned long                 ulongData;
  ValueTag                      tag1;
  octetSequence                 octseqData;
  ReplyTag                      tag2;
  String<4>                     strData;
};
interface XXX {
  AAAReply invoke(
    in long data1,
    in StringValue              string1,
    in StringValue              string2,
    in OctSeqValue              octetSeq1,
    in AAARequest               struct1
    );
};
```

struct内の sequence<octet>, stringメンバ用

struct内の sequence<octet> ,stringメンバ

struct内の sequence<octet> ,stringメンバ

各送信データに対応した

受信データ & 手前にstring or sequence<octet>有り

## (2)　通信処理に関するプログラム実装

プログラム実装として追加すべき処理について説明します。

### (a)　ORB クライアント送信データでの VALUE_TAG，REPOSITORY_ID 設定

string/sequence<octet>/struct に関しては，ORB クライアント側から EJB に対し送信する場合には，送信前にデータ本体以外に，VALUE_TAG,REPOSITORY_ID にも値を設定してください。

VALUE_TAG

常に固定値（0x7fffff02）を設定してください。

REPOSITORY_ID

string/sequence<octet>の場合

固定値（“RMI:[B:0000000000000000”）を設定してください。

struct の場合

リポジトリ ID は struct の名称・構造により異なってくるため，ctmjava2idl コマンドの出力結果からリポジトリ ID を取得してください。ctmjava2idl コマンド出力結果の「#pragma ID クラス名」の行，“RMI:～”部分がリポジトリ ID です。次の図の場合，"RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A"が AAA のリポジトリ ID になります。

図 9-7　リポジトリ ID の例

```
<ctmjava2idl出力>
//: CREATED BY CTMJAVA2IDL 010100
#include "TSCjava.idl"

※ 省略

//: valuetype AAA
"RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A"
struct AAA {
  //: primitive AAA::longData
  long longData;
  //: primitive AAA::ulongData
  long ulongData;
  ::org::omg::boxedRMI::seq1_octet octSeqData;
  ::org::omg::boxedRMI::seq1_octet strData;
};
#pragma ID AAA
"RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A"
```

また，COBOL のマップファイルで REPOSITORY_ID 部の長さを指定する場合，次の値を指定してください。

string/sequence<octet>の場合

23 文字以上

（固定値（“RMI:[B:0000000000000000”）の文字数以上）

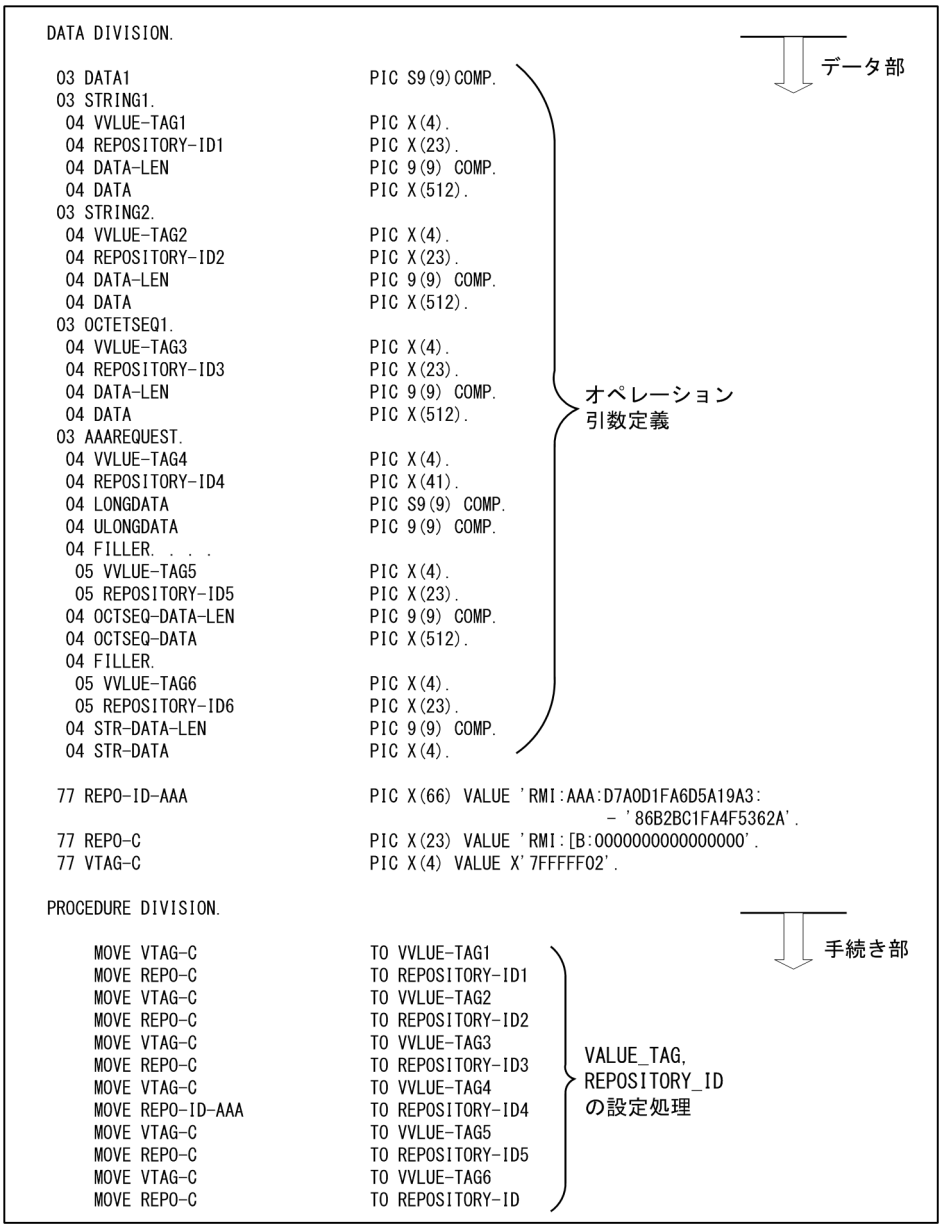
struct の場合

ctmjava2idl コマンドの出力結果から取得したリポジトリ ID の文字数以上

ORB クライアント側受信データに関しては，データ本体以外の情報（VALUE_TAG など）は無視してください。

「図 9-6　EJB 向けの IDL への変換例」の XXX::invoke()に対応した VALUE_TAG，REPOSITORY_ID の設定例を次に示します。

## 図 9-8　VALUE_TAG，REPOSITORY_ID の設定例（COBOL）



## 図 9-9　VALUE_TAG，REPOSITORY_ID の設定例（Java）

```
/* EJBの参照の取得*/
org.omg.CORBA.Object object = orb.string_to_object("IOR:・・・" );
XXX exRecive = XXXHelper.narrow(object);

/* ValueTagのセット */
final int vtg = 0x7FFFFF02;
final String rid="RMI:[B:0000000000000000";
ValueTag wVl = new ValueTag(vtg,rid);

/* 送信データの宣言*/
final int wldata=100;
final int wuldata=200;

/* 送信データ (string) の作成 */
StringValue wsvdata = new StringValue();
wsvdata.VALUE_TAG = 0x7FFFFF02;
wsvdata.REPOSITORY_ID="RMI:[B:0000000000000000";
wsvdata.data= new String("data");

/* 送信データ (octet sequence) の作成 */
byte[] wbdata=new byte[3];
wbdata[0]=0x01;
wbdata[1]=0x02;
wbdata[2]=0x03;
OctSeqValue wovdata = new OctSeqValue();
wovdata.VALUE_TAG = 0x7FFFFF02;
wovdata.REPOSITORY_ID="RMI:[B:0000000000000000";
wovdata.data=wbdata;

/* 送信データ
requestData.tag2=wVl;
requestData.strData= new String("test");

/* EJBの呼び出し */
AAAReply replyData = exRecive.invoke( wldata,
                                      wsvdata,
                                      wsvdata,
                                      wovdata,
                                      requestData);
```

図 9-10　VALUE_TAG，REPOSITORY_ID の設定例（C++）

```
/* EJBの参照の取得*/
XXX_var exrcv = XXX::_bind("HITACHI_EJB/…");

ValueTag wvl;
wvl.VALUE_TAG=0x7FFFFF02;
wvl.REPOSITORY_ID="RMI:[B:0000000000000000";

/* 送信データの宣言*/
int wldata=100;
int wuldata=200;

/* 送信データ (string) の作成 */
StringValue wsvdata ;
wsvdata.VALUE_TAG = 0x7FFFFF02;
wsvdata.REPOSITORY_ID="RMI:[B:0000000000000000";
wsvdata.data= "data";

/* 送信データ (octet sequence) の作成 */
OctSeqValue wovdata;
wovdata.VALUE_TAG = 0x7FFFFF02;
wovdata.REPOSITORY_ID="RMI:[B:0000000000000000";

char *str="sendData";
unsigned char* buf = new unsigned char[strlen(str)+1];
strcpy((char *)buf,str);
const octetSequence wk(strlen(str)+1,strlen(str)+1,buf,1);
wovdata.data=wk;

/* 送信データ (AAARequest) の作成 */
AAARequest requestData ;
requestData.VALUE_TAG = 0x7FFFFF02;
requestData.REPOSITORY_ID= "RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A";
requestData.longData=wldata;
requestData.ulongData=wuldata;
requestData.tag1=wvl;
requestData.octseqData=wk;
requestData.tag2=wvl;
requestData.strData="test";

/* EJBの呼び出し */
AAAReply_var replyData = exrcv->invoke( wldata,
                                        wsvdata,
                                        wsvdata,
                                        wovdata,
                                        requestData);
```

## (b)　IDL の string に対する EJB 上での取り扱い

EJB 側では文字列を byte[]として送受信するため，必要に応じて Java の String との相互変換をしてください。

受信した byte[]をそのまま Java の String に変換した場合，終端に’¥0’が付加された状態になるため，文字列長を取得すると，文字数が 1 文字分多い値が得られます。必要であれば終端の’¥0’を削除してください。

byte[]に文字列を設定する場合は，文字列の終端としてプログラムで明示的に’¥0’を付加してください。’¥0’で終了していないデータを IDL の string として返した場合，受信側でエラーになるおそれがあります。

IDL で string の最大長を指定している場合，終端文字’¥0’はその最大長に含まれないため，byte[]のサイズとしては最大長+1 を確保してください。

### (c) EJB での NULL

Java からの戻り値の class/byte[]に，null オブジェクトを使用しないでください。

## (3) 文字列化オブジェクトリファレンスの生成

ORB クライアントから ORB ゲートウェイに接続する IOR 文字列ファイルの取得手順について説明します。

1. ctmstartgw コマンドに-CTMIDLConnect オプションを指定して ORB ゲートウェイを開始します。
2. ORB クライアントから呼び出す J2EE アプリケーションをデプロイしてから開始します。
3. ctmgetior コマンドを実行し，IOR 文字列ファイルを取得します。

なお，IOR 文字列が記述されたファイルの再生成については，次の点に注意してください。

EJB のリモートインタフェースと CORBA ネーミングサービスの登録名称，サーバの IP アドレス，または，ORB ゲートウェイの受信ポート番号を変更しない場合

再生成する必要はありません。

ctmstartgw コマンドに「-CTMIDLConnect 0」を指定して ORB ゲートウェイを開始すると，ctmgetior コマンドによって IOR 文字列は取得できませんが，アプリケーションや ORB ゲートウェイの開始時間を短縮できます。

EJB のリモートインタフェース，CORBA ネーミングサービスの登録名称，サーバの IP アドレス，または，ORB ゲートウェイの受信ポート番号を変更する場合

文字列化オブジェクトリファレンスの生成手順 1 からやり直す必要があります。

## 9.5　CORBA システム例外発生時のトラブルシュート

CORBA システム例外発生時のトラブルシュートについて説明します。

### 9.5.1　CORBA システム例外のマイナーコード

J2EE サーバ，CTM デーモンおよび ORB ゲートウェイで発生した例外は，CORBA システム例外に変換して ORB クライアントに通知されます。このとき，マイナーコードには 1213473792～1213474047（0x48542400～0x485424FF）の値が設定されます。

CORBA システム例外にマイナーコードが設定された場合，次の表を参照し，エラー要因を調査して適切に対処してください。

表 9-9　CTM が設定するマイナーコード

| マイナーコード | エラー要因 | 対処 |
|---|---|---|
| 1213473792 | CTM でエラーが発生しました。 | トラブルシュート情報を取得し，障害情報と併せて保守員に連絡してください。 |
| 1213473793～<br>1213474042 | CTM でエラーが発生しました。 | 詳細は「9.5.2　CTM でエラーが発生した時のトラブルシュート」を参照してください。 |
| 1213474046 | CTM でエラーが発生しました。 | トラブルシュート情報を取得し，障害情報と併せて保守員に連絡してください。 |
| 1213474047 | J2EE サーバから RemoteException が throw されました。 | J2EE サーバが RemoteException を throw した要因を調査してください。 |

### 9.5.2　CTM でエラーが発生した時のトラブルシュート

CORBA システム例外のマイナーコードに 1213473793～1213474042 が設定された場合，CTMSPOOL ディレクトリに出力される例外ログファイルから CTM で発生したエラー要因を調査できます。

#### (1)　例外ログファイルフォーマット

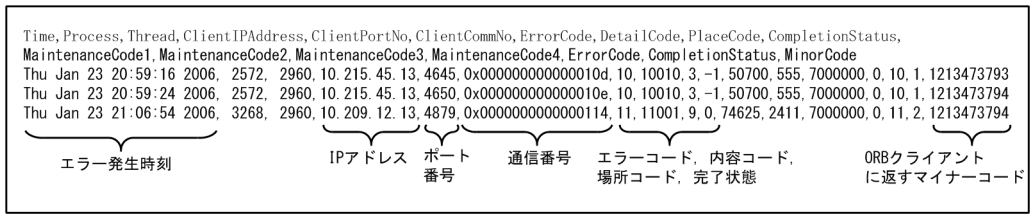
例外ログは 16 個のカンマ”,”区切りでエラー発生時に取得した情報を出力します。

| けた | 取得情報 | 説明 |
|---|---|---|
| 1 | Time | エラー発生時刻 |
| 2 | Process | プロセス ID |
| 3 | Thread | スレッド ID |
| 4 | ClientIPAddress | ORB クライアントの IP アドレス |

| けた | 取得情報 | 説明 |
|---|---|---|
| 5 | ClientPortNo | ORB クライアントのポート番号 |
| 6 | ClientCommNo | 通信番号 |
| 7 | ErrorCode | エラーコード |
| 8 | DetailCode | 内容コード |
| 9 | PlaceCode | 場所コード |
| 10 | CompletionStatus | 完了状態 |
| 11 | MaintenanceCode1 | 保守情報 |
| 12 | MaintenanceCode2 | 保守情報 |
| 13 | MaintenanceCode3 | 保守情報 |
| 14 | MaintenanceCode2 | 保守情報 |
| 15 | ErrorCode | ORB クライアントに返す完了状態 |
| 16 | CompletionStatus | ORB クライアントに返す完了状態 |
| 17 | MinorCode | ORB クライアントに返すマイナーコード |

例外ログの出力例を次の図に示します。

図 9-11　例外ログの出力例




## (2)　トラブルシュートの手順

1. 例外ログファイルに，ORB クライアントが受け取ったマイナーコードが 17 けた目に出力されているか確認してください。

   例外ログファイルは次の場所に格納されています。

   Windows の場合：

   <CTMSPOOL ディレクトリ>¥ejb¥<接続先の CTMID 名>¥expt¥

   Unix の場合：

   <CTMSPOOL ディレクトリ>/ejb/<接続先の CTMID 名>/expt/

2. マイナーコードが出力されていた場合は，該当する行のエラー発生時刻（1 けた目）と IP アドレス（4 けた目）とポート番号（5 けた目）が，ORB クライアントの IP アドレスとポート番号と一致するかどうか確認してください。

3. 該当する行のエラーコード（7 けた目），内容コード（8 けた目），場所コード （9 けた目），完了状態（10 けた目）を基にマニュアル「Cosminexus メッセージ 3 KFCT／KFDB／KFDJ 編」の付録を参照して,適切な対処をしてください。

4. 該当する行の通信番号（6 けた目）から PRF トレースでリクエストがどこまで実行されているかを確認して問題発生個所を絞り込んでください。

# 付録

# 付録 A　各バージョンでの主な機能変更

ここでは，11-20 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。11-20 での主な機能変更については，「1.4　アプリケーションサーバ 11-20 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 11-20 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 A.1　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 A-1　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 A-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 |
|---|---|---|---|
| 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 A–3　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 A.2　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 A–4　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 A–5　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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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 A-6 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 H |

## (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 A-7 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 A.3　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 A–8　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 A.4　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 A–9　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 機能におけるラムダ式の使用 | 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 A–10　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 G |
| 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 G |
| 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 G |

# 付録 A.5　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 A–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 |
|---|---|---|---|
| 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 A–12　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 A–13　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 A.6　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 A–14　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 | 9.18 |

### (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 A–15　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 | 9.16 |

### (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 A–16　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 A.7　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 A–17　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 A–18　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。<br>• Web サーバと J2EE サーバを別のホストに配置するパターン<br>• 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<br>• CTM を使用するパターン | 仮想化システム構築・運用ガイド | 1.1.2 |

## (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 A-19　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 やユーザアプリケーションでも 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 A-20　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 |
| | | 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。<br>• Explicit メモリブロックへのオブジェクト移動制御機能<br>• 明示管理ヒープ機能適用除外クラス指定機能<br>• 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 |
| | | このマニュアル | 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 |

| 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 A-21 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。 | このマニュアル | 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 | cjimport war(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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| | | | (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。 | このマニュアル | 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 A-22 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 (Component 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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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。<br>• 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。<br>• 使用できるイベント ID の範囲を拡張しました。<br>• 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。<br>• 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 A.8　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 A–23　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 |
| 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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| | | | 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 A-24 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, 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, |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| | | | 1.3.2,<br>1.4.2,<br>1.5.1,<br>1.6，2.3,<br>11 章,<br>12 章,<br>13 章,<br>17 章,<br>24 章,<br>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 A-25 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 A-26 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 |
| 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。<br>• ［設定情報の配布］画面<br>• Web サーバ，J2EE サーバおよび SFO サーバの起動画面 | 運用管理ポータル操作ガイド | 10.10.1,<br>11.9.2,<br>11.10.2,<br>11.10.4,<br>11.10.6,<br>11.11.2,<br>11.12.2, |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| | • 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 | | 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 A–27　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。<br>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 |
| | | 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 A.9　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 A-28　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。<br>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。<br>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。<br>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 |
| | | 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 A-29　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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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 A-30 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。 | このマニュアル | 6 章 |
| 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。 | このマニュアル | 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 A-31 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， |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| | | | 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 A–32　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。<br>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 A.10　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した項目を次の表に示します。

### 表 A–33　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。<br>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した項目を次の表に示します。

表 A–34　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した項目を次の表に示します。

表 A–35　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。<br>• OutOfMemoryError 発生時の動作を変更できるようになりました。<br>• 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。<br>• スレッド数の上限値を設定できるようになりました。<br>• 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した項目を次の表に示します。

表 A–36　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した項目を次の表に示します。

表 A–37　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。 | − | − |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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。<br>• HiRDB Server Version 9<br>• HiRDB/Developer's Kit Version 9<br>• HiRDB/Run Time Version 9<br>• SQL Server 2008<br>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 A.11　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した項目を次の表に示します。

### 表 A–38　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 |
| | | 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した項目を次の表に示します。

表 A-39　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した項目を次の表に示します。

表 A-40　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 |
| | | このマニュアル | 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。 | このマニュアル | 5.6.1 |
| 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した項目を次の表に示します。

表 A-41　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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| | | | 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した項目を次の表に示します。

### 表 A–42　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 |
| 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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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。 | このマニュアル | － |

# 付録 A.12　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した項目を次の表に示します。

### 表 A–43　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。<br>• 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。<br>• 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,<br>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した項目を次の表に示します。

### 表 A-44　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 |
| 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した項目を次の表に示します。

### 表 A-45　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。 | このマニュアル | 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，FullGC 発生を抑止できるようになりました。 | このマニュアル | 7 章 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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した項目を次の表に示します。

### 表 A-46 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した項目を次の表に示します。

### 表 A-47 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。 | このマニュアル | 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 |

| 項目 | 変更の概要 | 参照先マニュアル | 参照個所 |
|---|---|---|---|
| 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 B　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

## F



## G



## H



## J



## L



## P



## R



## S



## T



## U

## さ



## し



## す

## せ



## そ



## ち



## て



## と



## ね

## は

## ふ



## へ



## ま



## め



## ゆ



## ら



## り

## れ



## ろ