HITACHI
Inspire the Next

Cosminexus アプリケーションサーバ V8

システム設計ガイド

解説書

3020-3-U03-60

■対象製品

●**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003（x64）※1，Windows Server 2003 R2（x64）※1，Windows Server 2008 x86，Windows Server 2008 x64 ※1，Windows Server 2008 R2 ※1

P-2443-7B84 uCosminexus Application Server Standard-R 08-70

P-2443-7D84 uCosminexus Application Server Standard 08-70

P-2443-7K84 uCosminexus Application Server Enterprise 08-70

P-2443-7M84 uCosminexus Web Redirector 08-70

P-2443-7S84 uCosminexus Service Platform 08-70 ※2

●**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Vista，Windows XP，Windows 7（32bit），Windows 7（x64）※1

P-2443-7E84 uCosminexus Developer Standard 08-70

P-2443-7F84 uCosminexus Developer Professional 08-70

P-2443-7T84 uCosminexus Service Architect 08-70 ※2

●**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003（x64）※1，Windows Server 2003 R2（x64）※1，Windows Server 2008 x86，Windows Server 2008 x64 ※1，Windows Server 2008 R2 ※1，Windows Vista，Windows XP，Windows 7（32bit），Windows 7（x64）※1

P-2443-7H84 uCosminexus Client 08-70

●**適用** OS：Windows Server 2003（x64），Windows Server 2003 R2（x64），Windows Server 2008 x64，Windows Server 2008 R2

P-2943-7B84 uCosminexus Application Server Standard-R 08-70

P-2943-7D84 uCosminexus Application Server Standard 08-70

P-2943-7K84 uCosminexus Application Server Enterprise 08-70

P-2943-7S84 uCosminexus Service Platform 08-70 ※2

●**適用** OS：AIX 5L V5.3，AIX V6.1，AIX V7.1

P-1M43-7D81 uCosminexus Application Server Standard 08-70 ※2

P-1M43-7K81 uCosminexus Application Server Enterprise 08-70 ※2

P-1M43-7S81 uCosminexus Service Platform 08-70 ※2

●**適用** OS：HP-UX 11i V2（IPF），HP-UX 11i V3（IPF）

P-1J43-7D81 uCosminexus Application Server Standard 08-70

P-1J43-7K81 uCosminexus Application Server Enterprise 08-70

P-1J43-7S81 uCosminexus Service Platform 08-70 ※2

●**適用** OS：Red Hat Enterprise Linux AS 4（x86），Red Hat Enterprise Linux ES 4（x86），Red Hat Enterprise Linux AS 4（AMD64 & Intel EM64T），Red Hat Enterprise Linux ES 4（AMD64 & Intel EM64T），Red Hat Enterprise Linux 5 Advanced Platform（x86），Red Hat Enterprise Linux 5（x86），Red Hat Enterprise Linux 5 Advanced Platform（AMD/Intel 64），Red Hat Enterprise Linux 5（AMD/Intel 64），Red Hat Enterprise Linux Server 6（32-bit x86），Red Hat Enterprise Linux Server 6（64-bit x86_64）

P-9S43-7B81 uCosminexus Application Server Standard-R 08-70 ※2

P-9S43-7D81 uCosminexus Application Server Standard 08-70 ※2

P-9S43-7K81 uCosminexus Application Server Enterprise 08-70 ※2

P-9S43-7M81 uCosminexus Web Redirector 08-70 ※2

P-9S43-7S81 uCosminexus Service Platform 08-70 ※2

注※１　WOW64（Windows On Windows 64）環境だけで使用できます。
注※２　この製品については，サポート時期をご確認ください。
これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。
本製品では日立トレース共通ライブラリをインストールします。

■輸出時の注意

本製品を輸出される場合には，外国為替および外国貿易法ならびに米国の輸出管理関連法規などの規制をご確認の上，必要な手続きをお取りください。
なお，ご不明な場合は，弊社担当営業にお問い合わせください。

■商標類

■マイクロソフト製品の表記について

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

製品名	表記	
Microsoft(R) Cluster Service	Microsoft Cluster Service	
Microsoft(R) Internet Information Services 6.0	Microsoft IIS 6.0	Microsoft IIS
Microsoft(R) Internet Information Services 7.0	Microsoft IIS 7.0	
Microsoft(R) Internet Information Services 7.5	Microsoft IIS 7.5	
Microsoft(R) SQL Server 2000	SQL Server 2000	SQL Server
Microsoft(R) SQL Server 2005	SQL Server 2005	
Microsoft(R) SQL Server 2008	SQL Server 2008	
Microsoft(R) SQL Server 2000 Driver for JDBC	SQL Server 2000 Driver for JDBC	SQL Server の JDBC ドライバ
Microsoft(R) SQL Server 2005 JDBC Driver	SQL Server 2005 JDBC Driver	
Microsoft(R) SQL Server JDBC Driver 2.0	SQL Server JDBC Driver	
Microsoft(R) SQL Server JDBC Driver 3.0		

製品名	表記		
Microsoft(R) Windows(R) 7 Enterprise（32bit）	Windows 7（32bit）	Windows 7	Windows
Microsoft(R) Windows(R) 7 Professional（32bit）			
Microsoft(R) Windows(R) 7 Ultimate（32bit）			
Microsoft(R) Windows(R) 7 Enterprise（x64）	Windows 7（x64）		
Microsoft(R) Windows(R) 7 Professional（x64）			
Microsoft(R) Windows(R) 7 Ultimate（x64）			
Microsoft(R) Windows Server(R) 2003, Enterprise Edition 日本語版	Windows Server 2003 Enterprise Edition	Windows Server 2003	
Microsoft(R) Windows Server(R) 2003, Standard Edition 日本語版	Windows Server 2003 Standard Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition 日本語版	Windows Server 2003 R2 Enterprise Edition	Windows Server 2003 R2	
Microsoft(R) Windows Server(R) 2003 R2, Standard Edition 日本語版	Windows Server 2003 R2 Standard Edition		
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition 日本語版	Windows Server 2003 Enterprise x64 Edition	Windows Server 2003（x64）	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition 日本語版	Windows Server 2003 Standard x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition 日本語版	Windows Server 2003 R2 Enterprise x64 Edition	Windows Server 2003 R2（x64）	
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition 日本語版	Windows Server 2003 R2 Standard x64 Edition		
Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit 日本語版	Windows Server 2008 x86		
Microsoft(R) Windows Server(R) 2008 Standard 32-bit 日本語版			
Microsoft(R) Windows Server(R) 2008 Enterprise 日本語版	Windows Server 2008 x64		
Microsoft(R) Windows Server(R) 2008 Standard 日本語版			
Microsoft(R) Windows Server(R) 2008 R2 Enterprise 日本語版	Windows Server 2008 R2		
Microsoft(R) Windows Server(R) 2008 R2 Standard 日本語版			

製品名	表記		
Microsoft(R) Windows Vista(R) Business	Windows Vista Business	Windows Vista	
Microsoft(R) Windows Vista(R) Enterprise	Windows Vista Enterprise		
Microsoft(R) Windows Vista(R) Ultimate	Windows Vista Ultimate		
Microsoft(R) Windows(R) XP Professional Operating System	Windows XP		
Windows Server(R) Failover Cluster	Windows Server Failover Cluster		

■発行

2011 年 7 月 3020-3-U03-60

■著作権

変更内容

変更内容（3020-3-U03-60）uCosminexus Application Server Enterprise 08-70，uCosminexus Application Server Standard 08-70，uCosminexus Application Server Standard-R 08-70，uCosminexus Client 08-70，uCosminexus Developer Professional 08-70，uCosminexus Developer Standard 08-70，uCosminexus Service Architect 08-70，uCosminexus Service Platform 08-70，uCosminexus Web Redirector 08-70

追加・変更内容	変更個所
J2EE サーバおよびバッチサーバのファイルディスクリプタ数の見積もりの計算式を変更した。	5.2.1，6.2.1
パラレルコピーガーベージコレクションを実行できるようにした。	5.2.1，6.2.1，7.1.1，7.1.2，7.2
運用監視エージェントが使用するリソースの見積もりの計算式を変更した。	5.2.2
パラレルコピーガーベージコレクションを実行する場合，明示管理ヒープ機能は使用できないことを追加した。	7.1.1
HTTP セッションで利用する Explicit ヒープの省メモリ化機能についての説明を追加した。	7.10.3
運用管理ポータルで Hitachi Web Server のディレクティブが設定できる機能の追加に伴い，次のチューニングパラメタに運用管理ポータルを使用した設定方法の説明を追加した。 • Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ • 静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ	付録 C.4，付録 C.5
次の製品の適用 OS に AIX，HP-UX（IPF）を追加した。 • uCosminexus Application Server Enterprise • uCosminexus Application Server Standard • uCosminexus Service Platform	－
次の製品の適用 OS に Red Hat Enterprise Linux Server 6（32-bit x86），Red Hat Enterprise Linux Server 6（64-bit x86_64）を追加した。 • uCosminexus Application Server Enterprise • uCosminexus Application Server Standard • uCosminexus Application Server Standard-R • uCosminexus Service Platform • uCosminexus Web Redirector	－

単なる誤字・脱字などはお断りなく訂正しました。

変更内容（3020-3-U03-40）uCosminexus Application Server Enterprise 08-53，uCosminexus Application Server Standard 08-53，uCosminexus Application Server Standard-R 08-53，uCosminexus Client 08-53，uCosminexus Developer Professional 08-53，uCosminexus

Developer Standard 08-53，uCosminexus Service Architect 08-53，uCosminexus Service Platform 08-53，uCosminexus Web Redirector 08-53

追加・変更内容
JavaMail で使用できるサービスプロバイダに関する説明を変更した。
アプリケーションサーバのプロセスが使用する TCP/UDP のポート番号に仮想サーバマネージャの説明を追加した。
J2EE アプリケーションを実行するシステムで使用するリソース，および仮想メモリ使用量の見積もり方法を変更した。
TP1 インバウンド連携機能を使用する場合に加算するスレッド数，およびファイルディスクリプタ数の計算式を変更した。
バッチアプリケーションを実行するシステムで使用するリソース，および仮想メモリ使用量の見積もり方法を変更した。
JavaVM で使用するメモリ空間の構成と JavaVM オプションの注意事項に， -XX:+HitachiOutOfMemoryAbort（OutOfMemory 発生時強制終了機能），および -XX:+HitachiOutOfMemoryHandling（OutOfMemory ハンドリング機能）の説明を追加した。
Explicit ヒープのメモリサイズの見積もりに関する説明を変更した。
HTTP セッションに関する次の用語について，用語解説を変更・追加した。 • HTTP セッションに格納するオブジェクト • HTTP セッション • HTTP セッション管理用オブジェクト • HTTP セッションに関するオブジェクト
HiRDB Version 9 に対応した。
SQL Server 2008 に対応した。これに伴い，使用できる JDBC ドライバに SQL Server JDBC Driver 2.0，および SQL Server JDBC Driver 3.0 を追加した。
Microsoft Internet Information Services 7.0 および Microsoft Internet Information Services 7.5 に対応した。
対象製品として uCosminexus Application Server Standard-R を追加した。
次の製品の適用 OS から AIX，HP-UX，Linux（IPF）を削除した。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform

変更内容（3020-3-U03-20）uCosminexus Application Server Enterprise 08-50，uCosminexus Application Server Standard 08-50，uCosminexus Client 08-50，uCosminexus Developer Professional 08-50，uCosminexus Developer Standard 08-50，uCosminexus Service Architect 08-50，uCosminexus Service Platform 08-50，uCosminexus Web Redirector 08-50

追加・変更内容
JP1 Version 9 に対応した。また，JP1 Version 7 は非サポートとなったため削除した。
Cosminexus JMS プロバイダの機能を追加した。
OpenTP1 からアプリケーションサーバを呼び出す機能（TP1 インバウンド連携機能）を追加した。

追加・変更内容
ホスト単位管理モデルを対象とした系切り替え構成を追加した。
明示管理ヒープ機能を使用した Explicit ヒープ領域の利用に関する説明を追加した。
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方法について説明を追加した。
JavaVM メモリ空間のサイズや割合などを指定する JavaVM オプションに -XX:+HitachiAutoExplicitMemory オプションを追加した。
コピーガーベージコレクション，およびフルガーベージコレクションが発生するタイミングに関する説明を変更した。
フルガーベージコレクション発生を抑止するためのチューニングに関する説明を変更した。
Permanent 領域の使用量の見積もり方法に，Service Platform および Service Architect を使用する場合の説明を追加した。
稼働情報の見積もりに関する説明を追加した。
Explicit ヒープの最大サイズを変更した。
コネクションプールに関する説明を変更した。
次の製品の適用 OS に Windows Server 2008 R2 を追加した。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Web Redirector • uCosminexus Service Platform • uCosminexus Client
次の製品の適用 OS から Solaris を削除した。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise
次の製品の適用 OS に Windows 7 を追加した。 • uCosminexus Developer Standard • uCosminexus Developer Professional • uCosminexus Service Architect • uCosminexus Client

はじめに

このマニュアルは，Cosminexus（コズミネクサス）のアプリケーションサーバを使用して構築するシステムの設計について説明したものです。
アプリケーションサーバでは，次に示すプログラムプロダクトを使用してシステムを構築，運用します。

- uCosminexus Application Server Enterprise
- uCosminexus Application Server Standard
- uCosminexus Application Server Standard-R
- uCosminexus Client
- uCosminexus Developer Professional
- uCosminexus Developer Standard
- uCosminexus Service Architect
- uCosminexus Service Platform
- uCosminexus Web Redirector

このマニュアルでは，これらのプログラムプロダクトの構成ソフトウェアのうち，次に示す構成ソフトウェアについて説明しています。

- Cosminexus Component Container
- Cosminexus Component Container - Client
- Cosminexus Component Container - Redirector
- Cosminexus Component Transaction Monitor
- Cosminexus Developer's Kit for Java
- Cosminexus Performance Tracer
- Cosminexus Server Plug-in
- Cosminexus TPBroker

なお，オペレーティングシステム（OS）の種類によって，機能が異なる場合があります。

■対象読者

このマニュアルは，アプリケーションサーバを使ったシステムを設計する方を対象としています。
次の内容を理解されていることを前提としています。

- Windows またはご使用の UNIX のシステム設計，構築および運用に関する知識
- Java EE に関する知識
- SQL およびリレーショナルデータベースに関する基本的な知識
- CORBA に関する基本的な知識

JP1 連携機能を使用する場合は，次の内容も理解されていることを前提とします。

- JP1 の統合管理，ジョブ管理，ネットワーク管理およびアベイラビリティ管理に関する基本的な知識

Microsoft Cluster Service 連携機能を使用する場合は，次の内容も理解されていることを前提とします。
- Microsoft Cluster Service を使用したクラスタ構成に関する基本的な知識

Windows Server Failover Cluster を使用する場合は，次の内容も理解されていることを前提とします。
- Windows Server Failover Cluster を使用したクラスタ構成に関する基本的な知識

HA モニタ連携機能を使用する場合は，次の内容も理解されていることを前提とします。
- HA モニタを使用したクラスタ構成に関する基本的な知識

■ご利用製品ごとの用語の読み替えについて

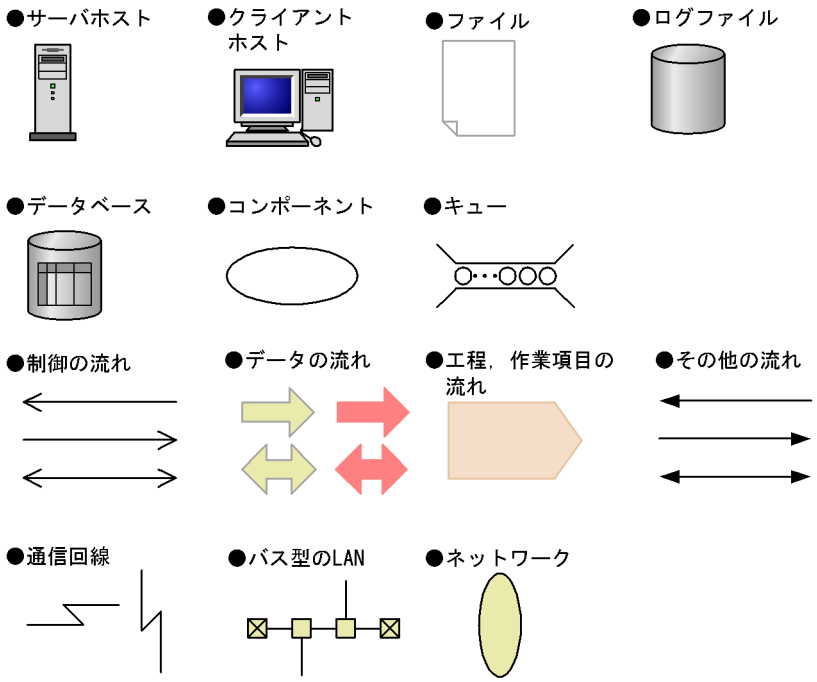
ご利用の製品によっては，マニュアルで使用している用語を，ご利用の製品名に読み替える必要があります。
次の表に従って，マニュアルで使用している用語をご利用の製品名に読み替えてください。

ご利用の製品名	マニュアルで使用している用語
uCosminexus Application Server Standard-R	Application Server
uCosminexus Developer Professional ※1	Application Server および Application Server Enterprise
uCosminexus Developer Standard ※1，※2	Application Server
uCosminexus Service Architect ※1	Application Server および Application Server Enterprise
uCosminexus Service Platform	

注※1　テスト環境で使用している場合にだけ読み替えが必要です。
注※2　uCosminexus Developer Standard と Application Server には一部機能差があります。機能差については，マニュアル「Cosminexus アプリケーションサーバ アプリケーション開発ガイド」の「付録 D　Developer Standard 使用時の注意事項」を参照してください。

■図中で使用している記号

このマニュアルの図中で使用する記号について次に示します。



■計算式で使用している記号

このマニュアルの計算式で使用する記号について次に示します。

記号	意味
↑計算式↑	計算式の答えの小数点以下を切り上げることを示します。
↓計算式↓	計算式の答えの小数点以下を切り捨てることを示します。

目次

1 アプリケーションサーバのシステム設計の目的と流れ

この章では，Cosminexus アプリケーションサーバのシステム設計の目的と，検討する必要がある項目について説明します。また，アプリケーションサーバのシステム設計の流れについても説明します。

1.1 アプリケーションサーバのシステム設計の目的

Cosminexus のアプリケーションサーバは，Java や CORBA などの業界標準に準拠したアプリケーションの実行環境である，**アプリケーションサーバ**を構築する製品です。アプリケーションサーバは，業務システムの構築基盤として位置づけられます。

業務システムには，次のような要件が求められます。

- **信頼性と可用性の高いシステムの実現**
  業務システムを止めることなく安定稼働させるために，信頼性と可用性を確保したシステムであることが必要です。
- **セキュリティの確保**
  システムを管理または運用するユーザがシステムを構築・運用していく過程や，システムが提供するサービスをエンドユーザが利用していく過程で，システムにはセキュリティ上のさまざまな脅威が想定されます。脅威からシステムを守るには，システムを物理的に安全な構成に設計したり，作業者の運用ルールを定めたりするなどの対策を実施する必要があります。
- **優れた処理性能の実現**
  Web クライアントなどの多数のクライアントからの処理要求や，EJB クライアントなどの業務のバックシステムでのミッションクリティカルな処理要求などに，迅速かつ確実に対応するレスポンスが求められます。

これらの要件を満たすシステムを構築するためには，システム構築を始める前にシステムの目的や特徴を分析したり，システムで使用するリソースを見積もったりするなど，最適なシステム構成を検討する必要があります。また，システムの運用を開始する前に，セキュリティを確保するための手順を整備したり，実際に想定される運用時の状態で動作確認およびチューニングを実施したりする必要があります。

アプリケーションサーバのシステム設計は，これらの作業を通して，アプリケーションサーバ上で動作する業務システムを，最適な状態で運用できるようにすることを目的とします。このマニュアルでは，アプリケーションサーバのシステムを設計する場合に検討，考慮する必要がある項目として，次の項目について説明します。

- システム構成の検討方法
- セキュアなシステムの検討方法
- パフォーマンスチューニングの実施方法

1.2　システム設計の流れ

アプリケーションサーバのシステム設計の流れについて説明します。

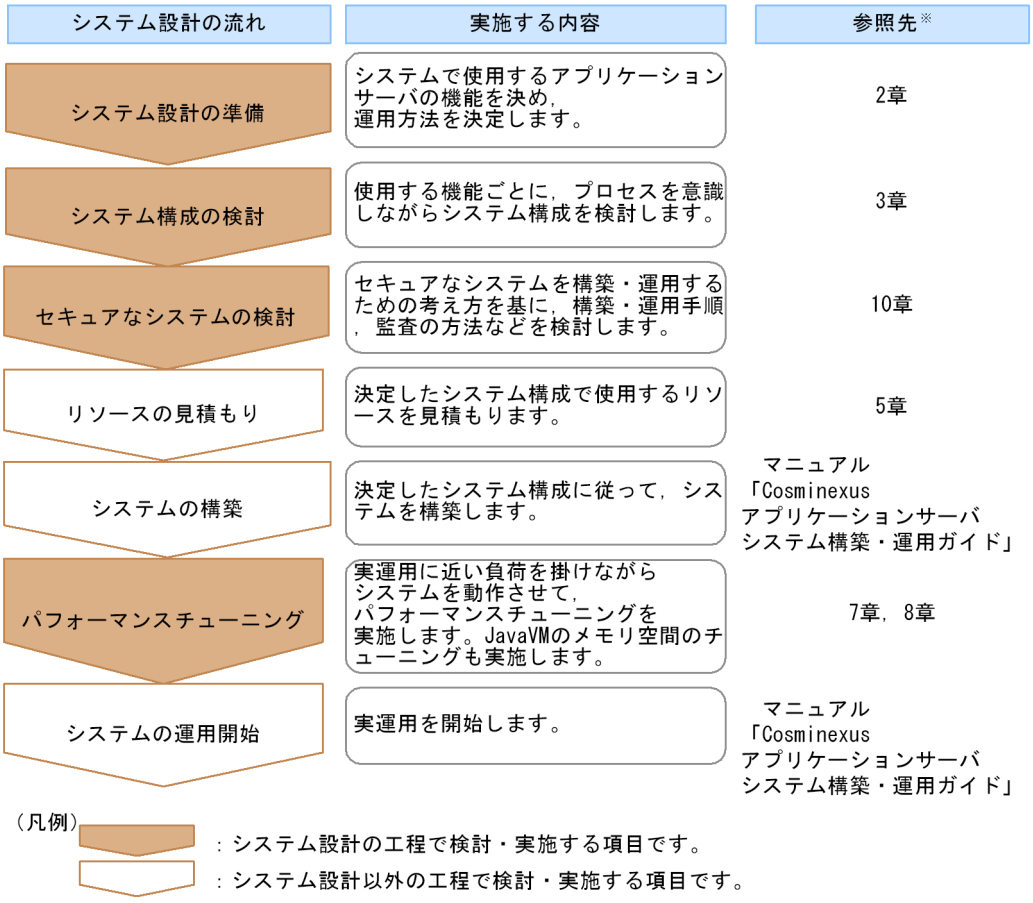
システム設計で考慮することは，そのシステムで実行するアプリケーションが，オンライン処理を実行するアプリケーション（J2EE アプリケーション）か，バッチ処理を実行するアプリケーション（バッチアプリケーション）かによって異なります。

それぞれのシステム設計の流れを示します。

1.2.1　オンライン処理を実行するアプリケーション（J2EE アプリケーション）の場合

システム設計は，次の図に示す流れで実行します。

図 1-1　システムの設計の流れ（J2EE アプリケーションを実行する場合）



注※

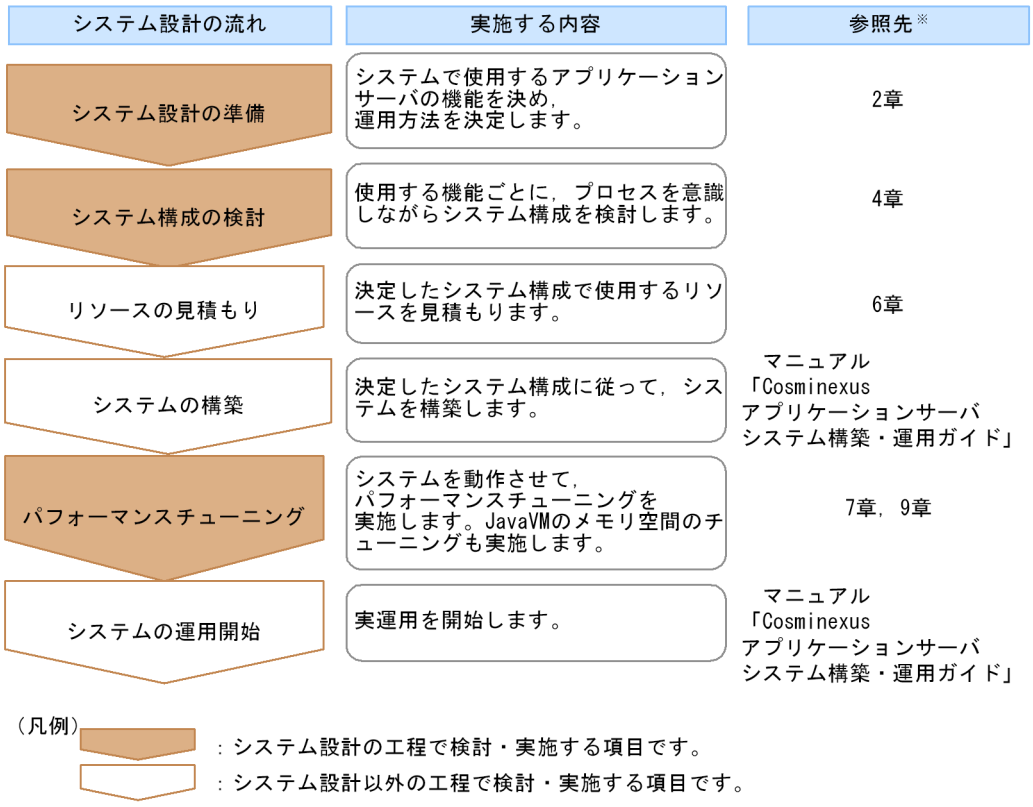
それぞれの参照先については，次の表に示した個所を参照してください。

システムの設計	参照先マニュアル	参照個所
システム設計の準備	このマニュアル	2 章
システム構成の検討		3 章
セキュアなシステムの検討		10 章
リソースの見積もり		5 章
システムの構築	Cosminexus アプリケーションサーバ システム構築・運用ガイド	8 章
パフォーマンスチューニング	このマニュアル	7 章，8 章
システムの運用開始	Cosminexus アプリケーションサーバ システム構築・運用ガイド	9 章

1.2.2　バッチ処理を実行するアプリケーション（バッチアプリケーション）の場合

システム設計は，次の図に示す流れで実行します。

図 1-2　システムの設計の流れ（バッチアプリケーションを実行する場合）



注※

それぞれの参照先については，次の表に示した個所を参照してください。

システムの設計	参照先マニュアル	参照個所
システム設計の準備	このマニュアル	2 章
システム構成の検討		4 章
リソースの見積もり		6 章
システムの構築	Cosminexus アプリケーションサーバ システム構築・運用ガイド	10 章
パフォーマンスチューニング	このマニュアル	7 章，9 章
システムの運用開始	Cosminexus アプリケーションサーバ システム構築・運用ガイド	11 章

2 システム設計の準備

この章では，システム設計の準備として，システム設計を始める前に決めておくことについて説明します。

2.1 システム設計を始める前に決めておくこと

この節では，アプリケーションサーバのシステム設計を始める前に決めておくことについて説明します。

システム設計作業を始める前に，まず，次のことを明確にしてください。これらの検討結果を踏まえて，3 章以降で説明するシステム構成の検討やパフォーマンスチューニングを実施します。

● 業務の種類を明確にする

システムで実現する業務の種類を明確にします。業務の種類によって，使用するアプリケーションの種類，構成，および必要なソフトウェアが決まります。

● システムの目的に応じたアプリケーションの構成を決める

システムの目的に従って，使用する機能を検討した上で，動作させるアプリケーションの構成を明確にします。また，必要なソフトウェアを準備します。

● 運用方法を検討する

アプリケーションサーバのシステムでは，Management Server という運用管理プロセスを使用して，複数のサーバプロセスを一括して運用します。

運用方法の検討では，アプリケーションサーバのシステムのほかに，アプリケーションサーバ以外のプログラムを含めたシステム全体を，どのように運用するかを検討します。

2.2 業務の種類を明確にする

システムでどのような業務を実現するのか，業務の種類を明確にします。

アプリケーションサーバのシステムでは，次の 2 種類の業務を実行できます。

● オンライン業務
ネットワーク経由などで送信されるクライアントからのリクエストを処理する形式の業務です。

● バッチ業務
定型的または定期的な作業をまとめて処理する形式の業務です。

業務の種類によって，実行するアプリケーションの形式や，アプリケーションを実行するサーバプロセスなどが異なります。業務の種類，アプリケーションの形式，およびアプリケーションを実行するサーバプロセスの対応を次の表に示します。

表 2-1 業務の種類，アプリケーションの形式，およびアプリケーションを実行するサーバプロセスの対応

業務の種類	アプリケーションの形式	アプリケーションを実行するサーバプロセス
オンライン業務	J2EE アプリケーション	J2EE サーバ
バッチ業務	バッチアプリケーション	バッチサーバ

ポイント

以降で実施するシステム設計の内容は，業務の種類によって異なります。業務の種類に応じて，必要なシステム設計を実施してください。業務の種類ごとに実行するシステム設計の内容と，このマニュアルでの参照先を次の表に示します。

表 2-2 業務の種類ごとに実行するシステム設計の内容とこのマニュアルでの参照先

システム設計の内容		業務の種類	
		オンライン業務	バッチ業務
システム設計の準備	アプリケーションの構成の決定	2.5	2.6
	運用方法の検討	2.7	
システム構成の検討		3 章	4 章
セキュアなシステムの検討		10 章	－
パフォーマンスチューニング		8 章	9 章
JavaVM のチューニング		7 章，7.3，7.10	

（凡例） －：該当しません。

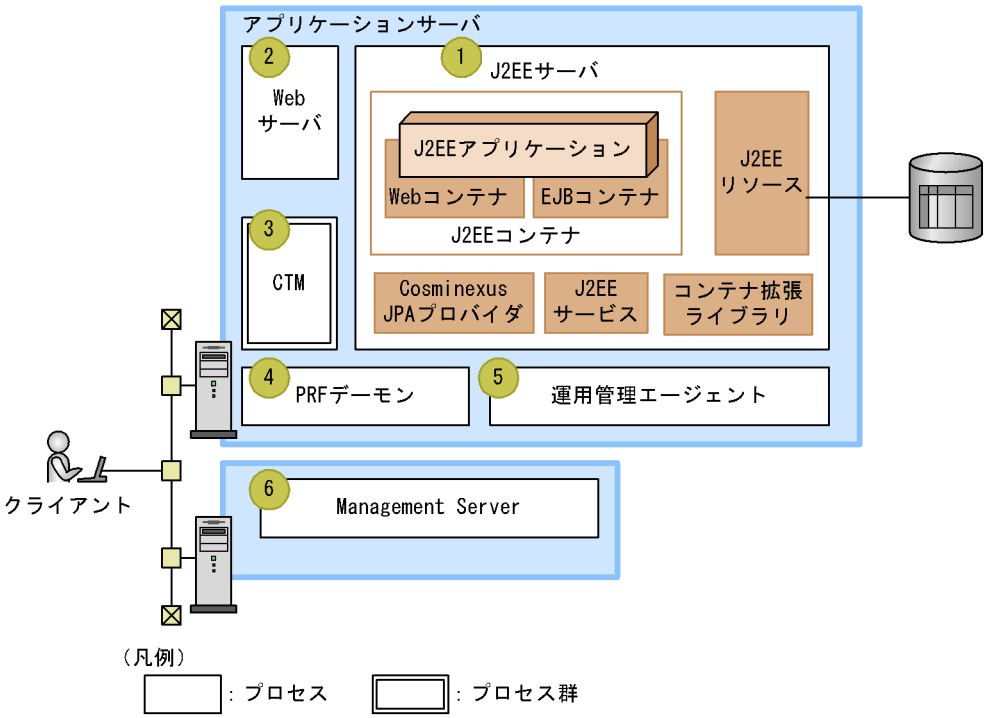
2.3　使用する機能を検討する（オンライン処理を実行する場合）

ここでは，J2EE アプリケーションを実行するアプリケーションサーバのプロセス構成と J2EE サーバの構成について説明します。

2.3.1　プロセス構成

J2EE アプリケーションを実行するアプリケーションサーバは，次の図に示すプロセスで構成されます。

図 2-1　J2EE アプリケーションを実行するアプリケーションサーバを構成するプロセス



参考

なお，システムを構築する場合は，これらのプロセスをシステムの要件に合わせて，システム内の各マシンに一つまたは複数配置します。

それぞれのプロセスについて説明します。なお，図中の番号は，(1) ～ (6) に対応します。

(1) J2EE サーバ

J2EE サーバは，J2EE アプリケーションの実行基盤となるプロセスです。J2EE サーバは，J2EE アプリケーション，J2EE コンテナ，J2EE サービス，J2EE リソースなど，複数のプログラムモジュールで構成されます。また，J2EE コンテナは，提供する機能によって，EJB コンテナと Web コンテナに分けられます。J2EE サーバを構成するプログラムモジュールについては，「2.3.2　J2EE サーバの構成」で説明します。

(2) Web サーバ

Web サーバは，Web ブラウザからのリクエスト受信，および Web ブラウザへのデータ送信に関連する処理を実行するプロセスです。J2EE サーバ上で動作する J2EE アプリケーションに Web ブラウザからアクセスするシステムの場合に，Web サーバを使用する必要があります※。なお，Web ブラウザからアクセスできるのは，J2EE アプリケーションに含まれるサーブレット，JSP，または静的コンテンツです。

アプリケーションサーバでは，Web サーバとして，Hitachi Web Server または Microsoft IIS を使用できます。Hitachi Web Server は，アプリケーションサーバの構成ソフトウェアの一つです。Hitachi Web Server の機能については，マニュアル「Hitachi Web Server」を参照してください。

注※

Web サーバを経由しないで J2EE サーバと Web ブラウザ間で直接リクエストを送受信する機能（インプロセス HTTP サーバ機能）を使用する場合，Web サーバに相当するプロセスは不要です。インプロセス HTTP サーバ機能は，Web コンテナの機能です。詳細は，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「4.　インプロセス HTTP サーバ」を参照してください。

(3) CTM

CTM は，J2EE アプリケーション内の Session Bean に対するリクエストをスケジューリングするためのプロセス群です。CTM を使用することで，クライアントからのリクエストを適切に分散，スケジューリングできます。これによって，サーバの負荷を抑え，システムの可用性を高めて業務を滞りなく進めるようにできます。

CTM としての機能は，CTM デーモン，CTM レギュレータ，CTM ドメインマネジャなどの，複数のプロセスを使用して実現します。また，ネーミングサービスとして，CORBA ネーミングサービスを使用します。

CTM の機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「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。

ポイント

CTM は，構成ソフトウェアに Cosminexus Component Transaction Monitor を含む製品だけで利用できます。利用できる製品については，マニュアル「Cosminexus アプリケーションサーバ 概説」の「2.3　構成ソフトウェア」を参照してください。

(4) PRF デーモン（パフォーマンストレーサ）

アプリケーションサーバは，リクエストを処理するときに，トレース情報をバッファに出力します。PRF デーモン（パフォーマンストレーサ）は，バッファに出力されたトレース情報をファイルに出力するための I／O プロセスです。PRF デーモンが出力するトレース情報ファイルは，システムのボトルネックを検証したり，トラブルシュートの効率向上を図ったりするために役立ちます。

PRF デーモンの機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「6.　性能解析トレースを使用したシステムの性能解析」を参照してください。

(5) 運用管理エージェント

運用管理者の代わりに，それぞれのホスト上の論理サーバを起動したり，設定ファイルを更新したりするエージェント機能を持つプロセスです。なお，論理サーバとは，Management Server の運用管理の対象になる，サーバまたはクラスタです。

(6) Management Server

運用管理ドメイン内の各ホストに配置した運用管理エージェントに指示を出して，運用管理ドメイン全体の運用管理を実行するためのプロセスです。

(7) そのほかのプロセス

(1) ～ (6) で示したプロセス以外に，機能に応じて使用するプロセスとして，次のプロセスがあります。

● SFO サーバ

メモリ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プロセスです。詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「7.　メモリセッションフェイルオーバ機能」を参照してください。

● ユーザサーバ

ユーザサーバとは，ユーザが定義する任意のサービスやプロセスです。ユーザサーバは論理サーバ（論理ユーザサーバ）として定義できます。論理ユーザサーバとして定義することで，特定のサービスやプロセスが Management Server の管理対象となります。これによって，ほかの論理サーバと同様に，Management Server で一括管理できるようになります。

● 旧バージョンとの互換用のプロセス

旧バージョンとの互換用の機能を使用する場合に使用するプロセスです。

- Web **コンテナサーバ**
  サーバの動作モードとしてサーブレットエンジンモードを使用する場合に，Web アプリケーションの実行基盤となるプロセスです。Web コンテナサーバでは，Web コンテナの機能を使用できます。なお，サーブレットエンジンモードの場合の Web コンテナの機能については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「10.4　サーブレットエンジンモードで使用できる Web コンテナの機能」を参照してください。

2.3.2　J2EE サーバの構成

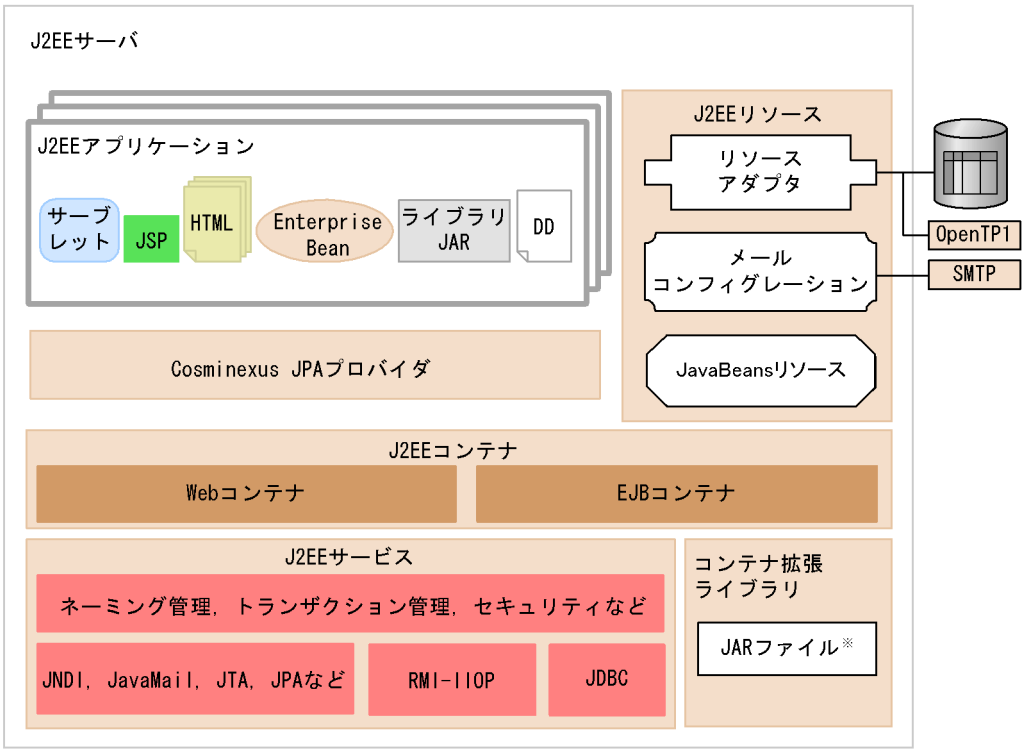
J2EE **サーバ**とは，次に示す五つのプログラムモジュールを実行する Java アプリケーションです。

- J2EE アプリケーション（サーブレット，JSP，Enterprise Bean など）
- J2EE コンテナ
- J2EE サービス
  JNDI，JavaMail，JTA，JPA，RMI-IIOP，JDBC，ネーミング管理，トランザクション管理，セキュリティなど
- J2EE リソース
- Cosminexus JPA プロバイダ
- コンテナ拡張ライブラリ

J2EE アプリケーションは，サーブレット，JSP，Enterprise Bean などによって構成されています。J2EE アプリケーションは，業務の内容に応じて，ユーザが開発します。なお，J2EE アプリケーション以外のプログラムモジュールは，アプリケーションサーバで提供されているモジュールです。

J2EE サーバの構造を次の図に示します。

図 2-2 J2EE サーバの構造



（凡例）

：Application Serverで提供されているプログラムモジュールの範囲です。

注※ JARファイルは使用する機能に応じてユーザが用意します。

以降の項で，J2EE サーバの各モジュールの概要を説明します。

2.3.3 J2EE アプリケーションと J2EE コンポーネント

J2EE アプリケーションは，一つ以上の J2EE コンポーネントで構成されています。ここでは，J2EE アプリケーションと J2EE コンポーネントについて説明します。

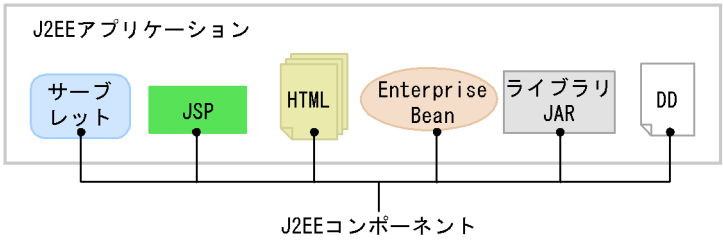
(1) J2EE アプリケーションと J2EE コンポーネントの関係

J2EE アプリケーションは，サーブレット，JSP，Enterprise Bean などのユーザアプリケーションプログラムで構成されています。J2EE アプリケーションは，J2EE コンテナ上で動作します。

J2EE アプリケーションを構成する，サーブレット，JSP，Enterprise Bean などを J2EE コンポーネントといいます。

J2EE アプリケーションと J2EE コンポーネントの関係を次の図に示します。

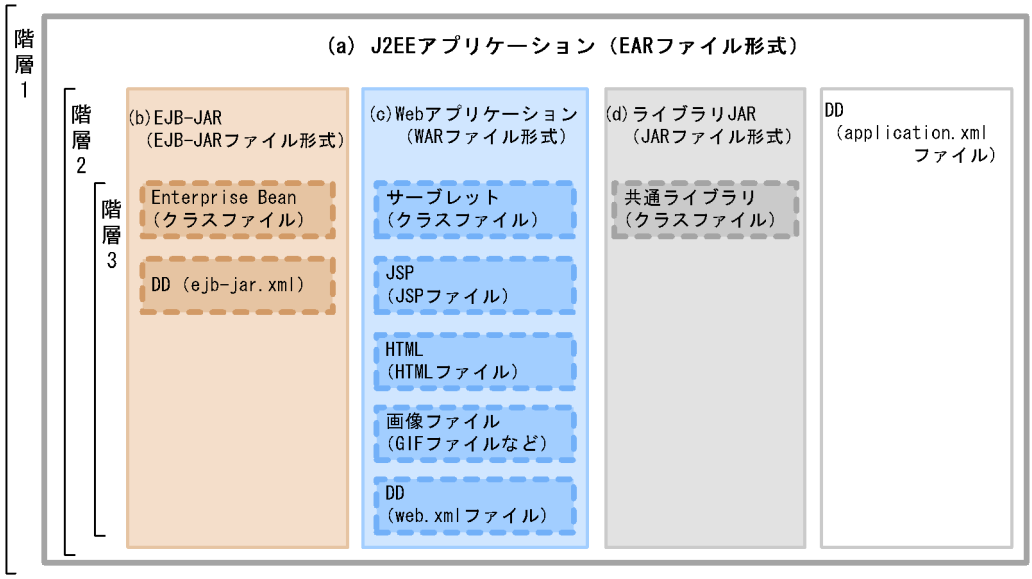
図 2-3　J2EE アプリケーションと J2EE コンポーネントとの関係



(2) J2EE アプリケーションの構造

J2EE アプリケーションは，3 層の構造になっています。J2EE アプリケーションの構造を次の図に示します。

図 2-4　J2EE アプリケーションの構造



注　図中の項番は，本文中の項番と対応しています。

J2EE アプリケーションの最小単位は，階層 3 のファイル（図中，点線で囲まれたファイル）です。階層 3 のファイルには，クラスファイルや JSP ファイルなどがあります。

そして，階層 3 のファイルをパッケージしたものが階層 2 のファイルとなります。この図の場合，階層 2 の EJB-JAR ファイルは，階層 3 に属する Enterprise Bean と DD（ejb-jar.xml）をパッケージしたものとなります。

さらに，階層 2 のそれぞれのファイルをパッケージしたものが階層 1 の J2EE アプリケーションとなります。

ここでは，階層 1 および階層 2 のパッケージファイルについて説明します。なお，図中の項番は次の説明の項番と対応しています。

参考

それぞれの階層でファイル形式，および DD の DTD が規定されています。
DD とは，アプリケーションを運用環境に配置するときの定義情報を記述したファイルを指します。EJB-JAR の場合，DD は ejb-jar.xml，Web アプリケーションの場合，DD は web.xml，J2EE アプリケーションの場合，DD は application.xml となります。
なお，Enterprise Bean でアノテーションを使用する場合，ejb-jar.xml は不要です。

(a) J2EE アプリケーション

J2EE アプリケーションは，複数の，EJB-JAR，Web アプリケーション，ライブラリ JAR と，一つの DD（application.xml）で構成されます。

J2EE サーバで実行できる J2EE アプリケーションは，アーカイブ形式の J2EE アプリケーション，および展開ディレクトリ形式の J2EE アプリケーションです。

● アーカイブ形式の J2EE アプリケーション

EJB やサーブレットなどのアプリケーションの実体を J2EE サーバの作業ディレクトリに持つ J2EE アプリケーションです。アーカイブ形式の J2EE アプリケーションを J2EE サーバ内にインポートしてクライアントから実行可能な状態にするためには，EAR 形式または ZIP 形式へのアセンブルと，デプロイが必要です。

● 展開ディレクトリ形式の J2EE アプリケーション

EJB やサーブレットなどのアプリケーションの実体を，J2EE サーバの外部にある一定のルールに従ったファイル／ディレクトリに持つ J2EE アプリケーションです。展開ディレクトリ形式の J2EE アプリケーションを J2EE サーバ内にインポートしてクライアントから実行可能な状態にするためには，デプロイが必要です。

参考

- **アセンブル**とは，単体では動作しない EJB-JAR をアプリケーションの 1 構成要素として位置づけるための組み立て作業のことです。アセンブルでは，EJB-JAR を 1 構成要素として取り込んだ J2EE アプリケーションを構築します。なお，J2EE アプリケーションの構成要素として，EJB-JAR のほかに WAR ファイル，ライブラリ JAR を含むことがあります。
  展開ディレクトリ形式の J2EE アプリケーションの場合には，EAR 形式または ZIP 形式へのアセンブルは不要です。
- **デプロイ**とは，J2EE アプリケーションをクライアントから実行可能な状態にすることです。

(b) EJB-JAR

EJB-JAR は，EJB-JAR ファイル形式でパッケージ化されています。複数の Enterprise Bean と一つの DD（ejb-jar.xml）で構成されます。なお，Enterprise Bean でアノテーションを使用している場合は，DD（ejb-jar.xml）は不要です。

(c) Web アプリケーション

Web **アプリケーション**は，WAR ファイル形式でパッケージ化されています。複数のサーブレット，JSP，HTML と一つの DD（web.xml）で構成されます。

(d) ライブラリ JAR

ライブラリ JAR は，JAR ファイル形式でパッケージ化されたものです。複数の共通ライブラリから構成されています。共通ライブラリは J2EE アプリケーション中の J2EE コンポーネントが共通で使用できるライブラリです。J2EE アプリケーションの DD（application.xml）の <module> タグ以下に定義されているファイル以外で，拡張子が小文字（.jar）の JAR ファイルがライブラリ JAR とみなされます。

(3) J2EE アプリケーションおよび J2EE コンポーネントの開発

J2EE アプリケーションでアプリケーションサーバが提供する実行基盤としての機能を使用するためには，機能に応じたアプリケーションの実装が必要です。なお，アプリケーションサーバでは，J2EE アプリケーションおよび J2EE コンポーネントを開発するための製品として，Developer を提供しています。

J2EE アプリケーションおよび J2EE コンポーネントの開発方法については，マニュアル「Cosminexus アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

2.3.4 J2EE コンテナ

J2EE **コンテナ**とは，J2EE アプリケーションを実行するためのサーバ基盤です。J2EE コンテナは，EJB コンテナと Web コンテナで構成されています。

J2EE コンポーネントは Web コンテナおよび EJB コンテナで提供する API を利用して，J2EE コンテナ上で動作します。アプリケーションサーバで提供している Web コンテナおよび EJB コンテナは，Java EE 5 に対応しています。これによって，Java EE に準拠する，本格的な基幹業務アプリケーションを迅速かつ容易に構築できます。各 API のバージョンについては，「2.3.9　サーバの動作モード」を参照してください。

(1) Web コンテナ

Web **コンテナ**は，サーブレットと JSP を実行するためのサーバ基盤です。Web クライアントからアクセスを受け取り，要求に応じたサービスを提供します。

Web コンテナでは，サーブレット，JSP の API を提供しています。

(2) EJB コンテナ

EJB コンテナは，Enterprise Bean の実行を制御し，Enterprise Bean に各種のサービスを提供するサーバ基盤です。EJB コンテナでは，EJB の API を提供しています。

2.3.5 J2EE サービス

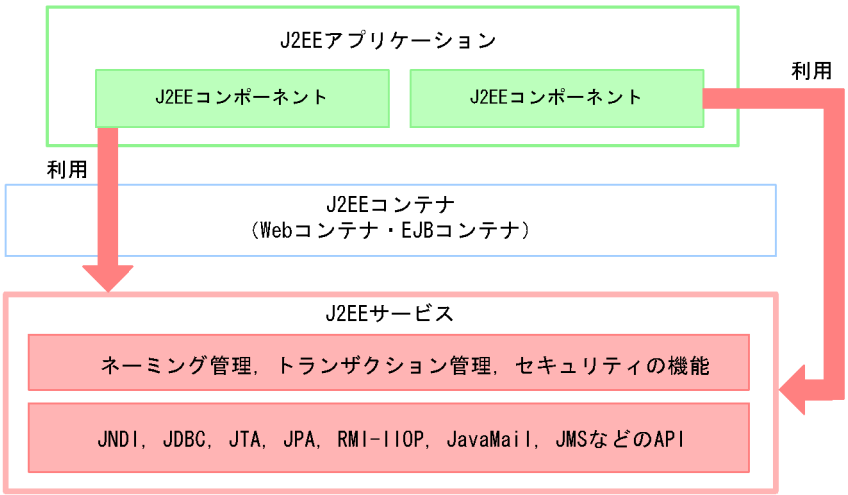
J2EE サービスでは，次に示す機能および API を提供しています。

1. トランザクション管理，セキュリティ管理，およびネーミング管理の機能
2. JNDI，JDBC，JTA，JPA，RMI-IIOP，JavaMail，JMS などの API

J2EE サービスは，J2EE コンテナの部品機能として利用され，J2EE コンポーネントである，サーブレット・JSP，および Enterprise Bean に，機能および API を提供します。J2EE サービスの API は，J2EE コンポーネントによって，直接，または J2EE コンテナ経由で利用されます。

J2EE サービスの位置づけを次の図に示します。

図 2-5 J2EE サービスの位置づけ



J2EE サービスでは，アプリケーションサーバの構成ソフトウェアの機能のほか，アプリケーションサーバ以外の製品の機能も使用します。J2EE サービスを実現する，ソフトウェア製品またはアプリケーションサーバの構成ソフトウェアを次の表に示します。

表 2-3 J2EE サービスを実現する，製品または構成ソフトウェア

分類		製品または構成ソフトウェア
サービス	ネーミング管理	Cosminexus Component Container ※ Cosminexus TPBroker ※
	トランザクション管理	
	セキュリティ	Cosminexus Component Container ※
API	JNDI	
	JTA	
	JPA	
	JavaMail	
	RMI-IIOP	Cosminexus TPBroker ※
	JDBC Standard Extension	HiRDB Type4 JDBC Driver Oracle JDBC Thin Driver SQL Server Driver for JDBC
	JDBC	
	JMS	Cosminexus RM ※ TP1/Message Queue - Access

注※ アプリケーションサーバの構成ソフトウェアです。

2.3.6 J2EE リソース

J2EE サーバはリソースとして，データベース，OpenTP1，SMTP サーバ，および JavaBeans リソースを利用できます。J2EE リソースは，これらのリソースと接続するために使用します。

アプリケーションサーバで扱う J2EE リソースには，外部リソースとの接続に利用するリソースアダプタ，およびメールコンフィグレーションがあります。また，このほかに，内部リソースとして利用できる JavaBeans リソースがあります※1。

● リソースアダプタ

接続するリソースの種類に応じて，次のリソースアダプタがあります。

- DB Connector
  データベースとの接続に利用します。
- DB Connector for Cosminexus RM **および** Cosminexus RM
  データベース上のキューとの接続に利用します。
- uCosminexus TP1 Connector
  OpenTP1 の SPP との接続に利用します。
- TP1/Message Queue - Access
  TP1/Message Queue との接続に利用します。
- **そのほかの** Connector 1.0 **または** Connector 1.5 ※2 **に準拠したリソースアダプタ**
  任意のリソースとの接続に使用します。

● メールコンフィグレーション
SMTP サーバとの接続に利用します。

● JavaBeans リソース
内部のリソースとして利用できるリソースです。

J2EE リソースの詳細については，マニュアル「Cosminexus 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. リソース接続とトランザクション管理」を参照してください。

注※ 1 このほかに，互換用のモードであるベーシックモードで使用するデータソースがあります。データソースについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「9.3 ベーシックモードでのリソース接続」を参照してください。

注※ 2 Outbound の通信モデルに対応したリソースアダプタを使用できます。詳細については，マニュアル「Cosminexus 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「3.16.8 Connector 1.5 仕様に準拠したリソースアダプタを使用する場合の設定」を参照してください。

2.3.7 Cosminexus JPA プロバイダ

アプリケーションサーバで提供している JPA 実装を Cosminexus JPA プロバイダといいます。Cosminexus JPA プロバイダを利用することによって，アプリケーションサーバで JPA アプリケーションを実行できます。

Cosminexus JPA プロバイダについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (コンテナ共通機能)」の「6. Cosminexus JPA プロバイダ」を参照してください。

2.3.8 コンテナ拡張ライブラリ

Enterprise Bean，サーブレット，JSP が利用する共通の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によって，Enterprise Bean，サーブレット，JSP からユーザ作成の共通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ついては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (コンテナ共通機能)」の「13. コンテナ拡張ライブラリ」を参照してください。

2.3.9 サーバの動作モード

サーバの動作モードには，J2EE サーバモードとサーブレットエンジンモードがあります。さらに，J2EE サーバモードには，1.4 モード，およびベーシックモードの 2 種類が

あります。このうち，ベーシックモードとサーブレットエンジンモードは，互換用の動作モードとなります。

このマニュアルのサーバの動作モードに関する記述は，すべて 1.4 モードの説明となります。なお，ベーシックモードおよびサーブレットエンジンモードについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「9. 旧バージョンとの互換用機能（ベーシックモード）」を参照してください。

1.4 モードで動作する Web コンテナは，Java EE のほかの要素と連携し，J2EE サーバの一部として動作します。この場合，Web コンテナ上で動作する Web アプリケーションからは，Java EE が定める幾つかの Java EE 関連の API を利用できます。

1.4 モードで使用できる Java EE および J2EE の機能を次に示します。

- Servlet 2.3/Servlet 2.4/Servlet 2.5
- JSP 1.2/JSP 2.0/JSP 2.1 ※1
- JSP Debugging 1.0
- EJB 2.0
  - Message-driven Bean
  - ローカルインタフェース
  - CMP 1.1
  - CMP 2.0
- EJB 2.1 ※2
- EJB 3.0（Session Bean）
- Common Annotation 1.0
- JDBC 2.0 コア /JDBC 2.0 オプションパッケージ
- JDBC 3.0 ※3
- JMS 1.0.2 ※4
- JMS 1.1 ※5
- Connector 1.0（JCA 1.0）
- Connector 1.5（JCA 1.5）
- JTA 1.0.1
  - local ※6
  - global ※7
- JTA 1.1
- JPA 1.0

● JavaMail 1.2 ※8

● JavaMail 1.3 ※8

注※1

JSP 1.1 は使用できません。

注※2

CMP 機能のうち EJB2.1 での拡張部分，<service-ref> タグを使用した Web サービス連携機能は使用できません。

注※3

接続に使用する JDBC ドライバが，JDBC 3.0 仕様で規定された機能をサポートしている必要があります。

注※4

Cosminexus RM または TP1/Message Queue - Access を使用する場合は，JMS1.0.2 であることが前提です。また，Topic を含む一部の機能に制限があります。

注※5

JMS1.1 を使用する場合，次の条件があります。

- JMS プロバイダが JMS1.1 に対応していること
- Message-driven Bean が EJB2.1 に対応していること（Message-driven Bean を使用する場合）
- 使用するリソースアダプタが Connector 1.5 に対応していること

注※6

リソースアダプタの DD（ra.xml）の transaction-support で LocalTransaction を指定し，ビジネスロジック中にリモートで JavaVM の呼び出しをしない場合に，ローカルトランザクションが利用できます。

注※7

ライトトランザクションが有効になっているときは，グローバルトランザクションを使用できません。なお，ライトトランザクションについては，マニュアル「Cosminexus 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「3.14.5　ライトトランザクション」を参照してください。

注※8

JavaMail では，送信時のサービスプロバイダとして SMTP を，受信時のサービスプロバイダとして POP3 を使用できます。

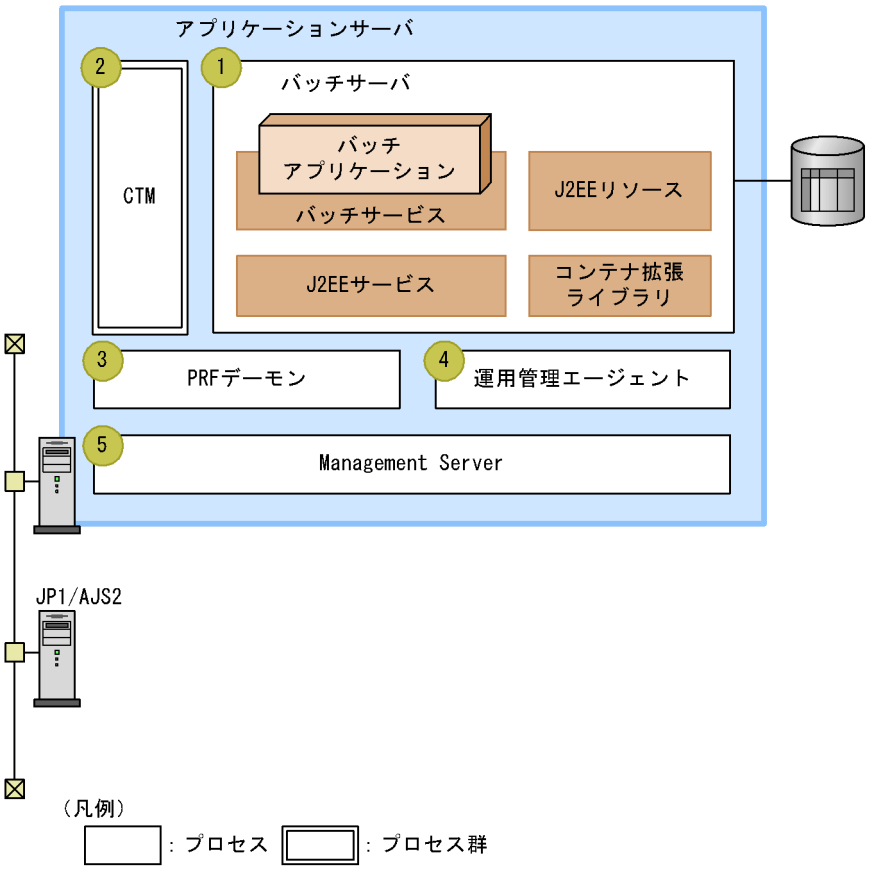
2.4　使用する機能を検討する（バッチ処理を実行する場合）

ここでは，バッチアプリケーションを実行するアプリケーションサーバのプロセス構成と，バッチサーバの構成について説明します。

2.4.1　プロセス構成

バッチアプリケーションを実行するアプリケーションサーバは，次の図に示すプロセスで構成されます。

図 2-6　バッチアプリケーションを実行するアプリケーションサーバを構成するプロセス



参考

システムを構築する場合は，これらのプロセスをシステムの要件に合わせて，システム内の各マシンに一つまたは複数配置します。

それぞれのプロセスについて説明します。なお，図中の番号は，(1) ～ (5) に対応します。

(1) バッチサーバ

バッチサーバは，バッチアプリケーションの実行基盤となるプロセスです。バッチサーバは，バッチアプリケーション，バッチサービス，J2EE サービス，J2EE リソースなど，複数のプログラムモジュールで構成されます。バッチサーバを構成するプログラムモジュールについては，「2.4.2　バッチサーバの構成」で説明します。

(2) CTM

CTM は，バッチアプリケーションの実行をスケジューリングするためのプロセス群です。CTM を使用することで，バッチアプリケーションの実行を適切に分散，スケジューリングできます。これによって，バッチサーバの数を意識することなく，複数のバッチアプリケーションを同時に実行できます。

CTM としての機能は，CTM デーモン，CTM レギュレータ，CTM ドメインマネジャなどの，複数のプロセスを使用して実現します。また，ネーミングサービスとして，CORBA ネーミングサービスを使用します。

CTM の機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「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。

ポイント

CTM は，構成ソフトウェアに Cosminexus Component Transaction Monitor を含む製品だけで利用できます。利用できる製品については，マニュアル「Cosminexus アプリケーションサーバ 概説」の「2.3　構成ソフトウェア」を参照してください。

(3) PRF デーモン（パフォーマンストレーサ）

アプリケーションサーバは，トレース情報をバッファに出力します。PRF デーモン（パフォーマンストレーサ）は，バッファに出力されたトレース情報をファイルに出力するための I／O プロセスです。PRF デーモンが出力するトレース情報ファイルは，システムのボトルネックを検証したり，トラブルシュートの効率向上を図ったりするために役立ちます。

PRF デーモンの機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「6.　性能解析トレースを使用したシステムの性能解析」を参照してください。

(4) 運用管理エージェント

運用管理者の代わりに，それぞれのホスト上の論理サーバを起動したり，設定ファイルを更新したりするエージェント機能を持つプロセスです。なお，論理サーバとは，

Management Server の運用管理の対象になる，サーバまたはクラスタです。

(5) Management Server

運用管理ドメイン内の各ホストに配置した運用管理エージェントに指示を出して，運用管理ドメイン全体の運用管理を実行するためのプロセスです。

参考

バッチ処理を実行する場合，(1) ～ (5) で示したプロセス以外に，システムの目的に応じて**ユーザサーバ**というプロセスを使用できます。ユーザサーバとは，ユーザが定義する任意のサービスやプロセスです。ユーザサーバは論理サーバ（論理ユーザサーバ）として定義できます。論理ユーザサーバとして定義することで，特定のサービスやプロセスが Management Server の管理対象となります。これによって，ほかの論理サーバと同様に，Management Server で一括管理できるようになります。

2.4.2 バッチサーバの構成

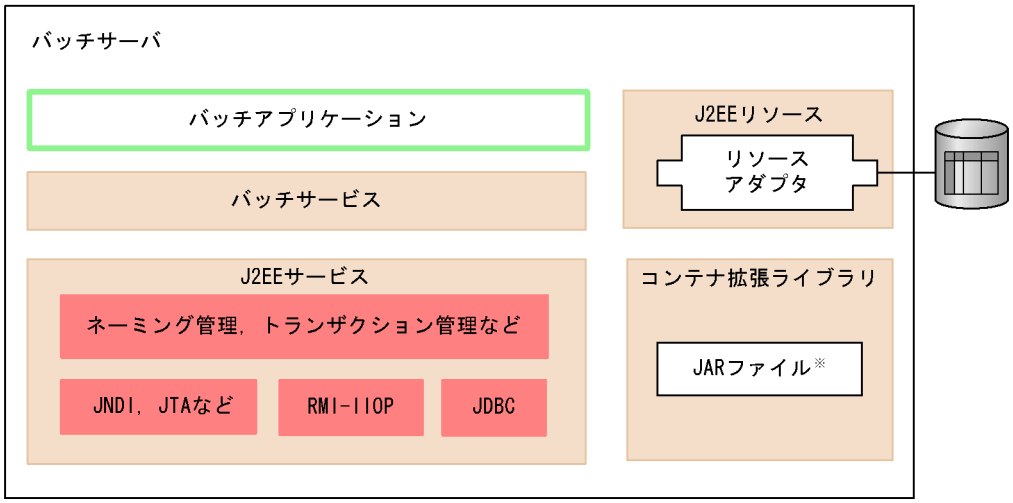
バッチサーバとは，次に示す五つのプログラムモジュールを実行する Java アプリケーションです。

- バッチアプリケーション
- バッチサービス
- J2EE サービス
  JNDI，JTA，RMI-IIOP，JDBC，ネーミング管理，トランザクション管理など
- J2EE リソース
- コンテナ拡張ライブラリ

バッチアプリケーションとは，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は，業務の内容に応じてユーザが開発します。なお，バッチアプリケーション以外のプログラムモジュールは，アプリケーションサーバで提供されているモジュールです。

バッチサーバの構造を次の図に示します。

図 2-7 バッチサーバの構造



（凡例）

：アプリケーションサーバで提供されているプログラムモジュールの範囲です。

注※ JARファイルは使用する機能に応じてユーザが用意します。

バッチサーバでは次に示す Java EE および J2EE の機能を使用できます。

- JDBC 2.0 コア /JDBC 2.0 オプションパッケージ
- JDBC 3.0 ※ 1
- Connector 1.0（DB Connector）※ 2
- JTA 1.0.1（ただし，local だけ）※ 3

注※ 1

接続に使用する JDBC ドライバが，JDBC 3.0 仕様で規定された機能をサポートしている必要があります。

注※ 2

トランザクションなし，またはローカルトランザクションの DB Connector を使用できます。

注※ 3

リソースアダプタの DD（ra.xml）の transaction-support で LocalTransaction を指定し，ビジネスロジック中にリモートで JavaVM の呼び出しをしない場合に，ローカルトランザクションが利用できます。

また，バッチサーバから次の EJB を呼び出せます。ただし，EJB の呼び出し方法はリモート呼び出しとなります。ローカル呼び出しはできません。

- EJB 2.0
- EJB 2.1
- EJB 3.0

以降の項で，バッチサーバの各モジュールの概要を説明します。

2.4.3 バッチアプリケーション

バッチアプリケーションとは，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は，一つのバッチサーバにつき一つ実行できます。

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。JP1/AJS2 と連携すると，バッチ実行コマンドを JP1 のジョブとして定義できるので，バッチアプリケーションの自動実行ができます。

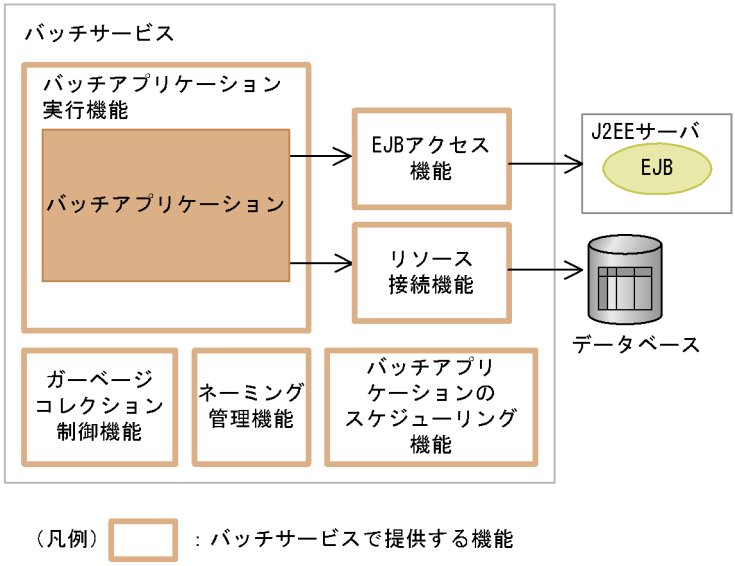
また，バッチアプリケーションのスケジューリング機能を使用すると，バッチアプリケーションの実行リクエストはスケジュールキューによって制御され，自動的にバッチサーバへ振り分けられます。複数のバッチアプリケーションを同時に開始したい場合に，バッチアプリケーションのスケジューリング機能を使用すると，バッチサーバの数や，どのバッチサーバで実行するかを意識する必要がありません。なお，バッチアプリケーションのスケジューリング機能を使用しない場合には，バッチアプリケーションごとにバッチサーバを用意してください。

バッチアプリケーションの詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「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。

2.4.4 バッチサービス

バッチアプリケーションを実行するアプリケーションサーバでは，バッチサービスを提供しています。バッチサービスとは，バッチアプリケーションを実行するための機能です。バッチサービスでは，次の図に示す機能を提供しています。

図 2-8　バッチサービスで提供している機能



- 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を開始したり，強制停止したりするための機能を提供しています。
- EJB アクセス機能
  バッチアプリケーションから J2EE サーバの EJB にアクセスするための機能を提供しています。EJB アクセス機能は J2EE サービスを使用します。
- リソース接続機能
  バッチアプリケーションからデータベースに接続するための機能を提供しています。リソース接続機能は，J2EE サービスおよび J2EE リソースを使用します。
- ガーベージコレクション制御機能
  バッチアプリケーションでリソース排他をしているときに，ガーベージコレクションの実行を制御するための機能を提供しています。
- ネーミング管理機能
  EJB またはリソースを参照するときに，名前解決をするための機能を提供しています。ネーミング管理機能は J2EE サービスを使用します。
- バッチアプリケーションのスケジューリング機能
  CTM を使用して，バッチアプリケーションの実行をスケジューリングするための機能を提供しています。

バッチサービスで提供するそれぞれの機能については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「2.3　バッチアプリケーション実行機能」を参照してください。

2.4.5　J2EE サービス

J2EE **サービス**では，次に示す機能および API を提供しています。

1. トランザクション管理，およびネーミング管理の機能
2. JNDI，JDBC，JTA，RMI-IIOP などの API

J2EE サービスは，バッチアプリケーションからデータベースに接続したり，EJB を呼び出したりするときに利用されます。

J2EE サービスでは，アプリケーションサーバの構成ソフトウェアの機能のほか，アプリケーションサーバ以外の製品の機能も使用します。J2EE サービスを実現する，ソフトウェア製品またはアプリケーションサーバの構成ソフトウェアを次の表に示します。

表 2-4　J2EE サービスを実現する，製品または構成ソフトウェア

分類		製品または構成ソフトウェア
サービス	ネーミング管理	Cosminexus Component Container ※ Cosminexus TPBroker ※
	トランザクション管理	
API	JNDI	Cosminexus Component Container ※
	JTA	
	RMI-IIOP	Cosminexus TPBroker ※
	JDBC Standard Extension	HiRDB Type4 JDBC Driver Oracle JDBC Thin Driver SQL Server Driver for JDBC
	JDBC	

注※　アプリケーションサーバの構成ソフトウェアです。

2.4.6　J2EE リソース

J2EE リソースはリソースと接続するために使用します。バッチサーバではリソースとしてデータベースを利用できます。データベースに接続するには，アプリケーションサーバで扱う J2EE リソースのうちリソースアダプタを使用します。

J2EE リソースの詳細については，マニュアル「Cosminexus 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.　リソース接続とトランザクション管理」を参照してください。

2.4.7　コンテナ拡張ライブラリ

アプリケーションが利用する共通の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によって，バッチアプリケーションからユーザ作成の共通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ついては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編（コンテナ共通機能）」の「13. コンテナ拡張ライブラリ」を参照してください。

2.5 システムの目的に応じたアプリケーションの構成を決める（オンライン処理を実行する業務の場合）

この節では，システムの目的に応じたアプリケーションの構成の検討について説明します。この節で説明するのは，J2EE アプリケーションの場合の構成です。また，必要となるソフトウェアについても説明します。

アプリケーションサーバで実現できるアプリケーションの機能については，次に示すマニュアルの機能の分類に関する説明を参照してください。

- マニュアル「Cosminexus 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「1.1　機能の分類」
- マニュアル「Cosminexus 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「1.1　機能の分類」
- マニュアル「Cosminexus 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「1.1　機能の分類」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「1.1　機能の分類」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.1　機能の分類」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「1.1　機能の分類」

2.5.1 動作させる J2EE アプリケーションの検討

システムで動作させる J2EE アプリケーションについて検討します。

システムの目的に応じたアプリケーションの構成を明確にすることで，システム構成の基本的な部分が決まります。例えば，クライアントに Web ブラウザを使用する場合には，アプリケーションはサーブレットや JSP で構成される Web アプリケーションにして，Web ブラウザからのリクエストを受け付けるようにします。また，必要に応じて，サーブレットや JSP から Enterprise Bean を呼び出すようなアプリケーションも考えられます。この場合は，システムは Web クライアントシステムにして，Web サーバの配置や，そこからアプリケーションサーバを呼び出す場合の連携方法を検討する必要があります。

また，業務システムの基幹部分を構成するシステムを構築する場合は，クライアントに EJB クライアントアプリケーションを使用する構成が考えられます。呼び出し先の Enterprise Bean の種類によっては，CTM を利用することも検討できます。

アプリケーションに含まれるコンポーネントの種類とシステム構成との関係については，

「3.3　アプリケーションの構成を検討する」で詳しく説明します。そのほか，J2EE アプリケーションを実行する場合のシステム構成の検討方法については，「3.　システム構成の検討（J2EE アプリケーション実行基盤）」を参照してください。

システムの目的に応じて使用できる機能については，次に示すマニュアルのシステムの目的と機能の対応に関する説明を参照してください。

- マニュアル「Cosminexus 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「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus 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「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus 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「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus 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　システムの目的と機能の対応」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「1.2　システムの目的と機能の対応」

また，アプリケーション開発の手順については，マニュアル「Cosminexus アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

参考

サーバの動作モードについて

サーバの動作モードは，J2EE サーバモード（1.4 モード）で動作させることをお勧めします。アプリケーションの種類によっては，サーブレットエンジンモードやベーシックモードで動作させることもできますが，これらは互換用のモードです。

2.5.2　使用するプロセスの検討と必要なソフトウェアの準備

アプリケーションサーバのシステムでは，使用するプロセスの種類とその配置によってシステム構成が決まります。

アプリケーションサーバのシステムは Web フロントシステムとバックシステムで構成されます。Web フロントシステムは，クライアントとして Web ブラウザを使用するシステムです。バックシステムは，クライアントとして EJB クライアントを使用するシステムです。システムの分類については，「3.1.1　システムの目的と構成」で詳しく説明します。

ここでは，まず，システムの分類に応じて必要なプロセスとソフトウェアについて説明します。次に，使用する機能に応じて必要なプロセス，モジュールおよびソフトウェアについて説明します。なお，これらのプロセス，モジュール，ソフトウェアをどのよう

にシステムに配置するかについては，「3. システム構成の検討（J2EE アプリケーション実行基盤）」で説明します。

（1）システムの分類に応じて必要なプロセス

システムの分類に応じて必要なプロセスを次に示します。これらは，使用する機能に関係なく共通して必要なプロセスです。アプリケーションサーバによって提供されます。

Web フロントシステムの場合に必要なプロセス

Web フロントシステムの場合に必要なプロセスは次のとおりです。

- Web サーバ※
- J2EE サーバ
- PRF デーモン

なお，アプリケーションサーバに含まれる Web サーバは，Hitachi Web Server です。クライアントには，Web ブラウザを使用します。

注※

インプロセス HTTP サーバを利用する場合は，Web サーバのプロセスは不要です。

ポイント

Web サーバ選択の指針

Web クライアントシステムでは，Web クライアントからのリクエストを，次のどちらかの Web サーバを利用して処理できます。

- **リダイレクタと連携した Web サーバ（Web サーバ連携）**
  アプリケーションサーバまたは Web Redirector が提供するリダイレクタモジュールを組み込んだ Web サーバと連携してリクエストを処理します。Web サーバが受信したリクエストは，リダイレクタモジュールを経由して，J2EE サーバに送信されます。
  Hitachi Web Server または Microsoft IIS が利用できます。
- **インプロセス HTTP サーバ**
  Web コンテナ機能の一部として提供される，J2EE サーバのプロセス内で機能する HTTP サーバでリクエストを処理します。Web クライアントからのリクエストを J2EE サーバで直接受信できます。

なお，アプリケーションサーバでは，リダイレクタと連携した Web サーバを利用することを推奨しています。また，デフォルトの設定で使用する場合は，リダイレクタと連携した Web サーバが利用されます。特に性能を重視したシステムを構築したい場合に，インプロセス HTTP サーバの利用を検討してください。
それぞれの Web サーバの特徴を次の表に示します。Web サーバを選択する場合の指針にしてください。

表 2-5 Web サーバ選択の指針

比較項目	リダイレクタと連携した Web サーバ	インプロセス HTTP サーバ
Web サーバとして利用できる機能	○ Hitachi Web Server（Apache の機能をベースにした Web サーバ）または Microsoft IIS が提供する多様な機能を利用できます。	△ サーブレット，JSP または HTML から構成される Web アプリケーションへのアクセスを目的にした最小限の機能だけが提供されています。※1
構築，運用の容易性	○ 構築時には，Web サーバの環境設定が必要です。運用時には，Web サーバの起動，停止が必要です。ただし，Smart Composer 機能のコマンドで構築・運用できるため，煩雑な操作は不要です。	○ 構築時の Web サーバの環境設定，および運用時の Web サーバの起動，停止の操作が不要です。
HTML，JPEG などの静的コンテンツに対するアクセス性能	○ 静的コンテンツを Web サーバ上に配置することによって，最適な性能を確保できます。 なお，Web コンテナ上に配置する場合は，リダイレクタ経由のアクセスになるため，アクセス処理に時間が掛かります。	○ リダイレクタを経由しないでアクセスできるため，最適な性能を確保できます。
サーブレット，JSP などの動的コンテンツに対するアクセス性能	△ リダイレクタを経由するためのアクセス処理に時間が掛かります。	○ リダイレクタを経由しないでアクセスできるため，最適な性能を確保できます。
注意事項	インターネットに接続する場合には，セキュリティ上の観点から，DMZ を確保する構成にして，リバースプロキシをフロントに配置することを推奨します。 また，リバースプロキシを配置しない場合は，リダイレクタを組み込んだ Web サーバを DMZ に配置することで，同様の効果を得ることもできます。※2	インターネットに接続する場合には，セキュリティ上の観点から DMZ を確保する構成にして，必ずリバースプロキシをフロントに配置してください。※2

（凡例）○：優れている。　△：優れていない。
注※1　インプロセス HTTP サーバで使用できる機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「4.2.2　インプロセス HTTP サーバで使用できる機能」を参照してください。
注※2　DMZ への Web サーバの配置については，「3.13　DMZ へのリバースプロキシの配置を検討する」を参照してください。

バックシステムの場合に必要なプロセス

バックシステムの場合に必要なプロセスは次のとおりです。

- J2EE サーバ
- PRF デーモン

バックシステムのクライアントには，EJB クライアントを使用します。**EJB クライアント**とは，Enterprise Bean を呼び出す，Servlet，JSP，ほかの Enterprise Bean，EJB クライアントアプリケーション，またはほかの業務システムのことです。
EJB クライアントとして EJB クライアントアプリケーションを使用するとき，Windows の場合はクライアントマシンを uCosminexus Client を使用して構築することもできます。アプリケーションサーバまたは uCosminexus Client のどちらのソフトウェアを使用した場合も，必要に応じて PRF デーモンを起動できます。

参考

CTM を使用したシステムの場合，クライアントとして TPBroker や TPBroker Object Transaction Monitor のクライアントなど，EJB クライアント以外のクライアントも使用できます。

(2) 使用する機能に応じて必要なプロセスおよびモジュール

使用する機能に応じて必要なプロセスおよびモジュールについて説明します。アプリケーションサーバによって提供されるものと，アプリケーションサーバ以外のソフトウェアによって提供されるものがあります。

使用する機能ごとに必要なプロセスのうち，アプリケーションサーバによって提供されるものを次の表に示します。これらのプロセスは，アプリケーションサーバをインストールしたマシンで起動できます。

表 2-6 機能ごとに必要なプロセスまたはモジュール（アプリケーションサーバによって提供されるもの）

機能	必要なプロセス
サーバ間連携で CTM を利用する／ CTM を利用して負荷を分散する	CTM デーモン
	CTM レギュレータ
	CTM ドメインマネジャ
	グローバル CORBA ネーミングサービス
	スマートエージェント
Management Server を利用して運用管理する	Management Server
	運用管理エージェント
メモリセッションフェイルオーバ機能を使用して可用性を向上させる	SFO サーバ
CORBA ネーミングサービスをアウトプロセスで起動する	CORBA ネーミングサービス

使用する機能ごとに必要なプロセスおよびモジュールのうち，アプリケーションサーバ以外の製品が提供するプロセスおよびモジュールを，表 2-7 および表 2-8 に示します。

表 2-7 機能ごとに必要なモジュール（アプリケーションサーバ以外によって提供されるもの）と提供するソフトウェア

機能	モジュール	提供するソフトウェア	備考
データベース（HiRDB）と接続する	HiRDB Type4 JDBC Driver	• HiRDB Server Version 9 • HiRDB Server with Additional Function Version 9 • HiRDB/Run Time Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB Developer's Suite Version 9 • HiRDB/Parallel Server Version 8 • HiRDB/Single Server Version 8 • HiRDB/Run Time Version 8 • HiRDB/Developer's Kit Version 8	JDBC ドライバとして HiRDB Type4 JDBC Driver を使用する場合に必要になります。
データベース（Oracle）と接続する	Oracle JDBC Thin Driver	• Oracle JDBC Thin Driver	JDBC ドライバとして Oracle JDBC Thin Driver を使用する場合に必要になります。
データベース（SQL Server）と接続する	SQL Server 2000 Driver for JDBC	• SQL Server 2000 Driver for JDBC	SQL Server 2000 に接続する場合に必要になります。
	SQL Server 2005 JDBC Driver	• SQL Server 2005 JDBC Driver	SQL Server 2005 に接続する場合に必要になります。
	SQL Server JDBC Driver	• SQL Server JDBC Driver	SQL Server 2008 に接続する場合に必要になります。
データベース（XDM/RD E2）と接続する	HiRDB Type4 JDBC Driver	• HiRDB/Parallel Server Version 8 • HiRDB/Single Server Version 8 • HiRDB/Run Time Version 8 • HiRDB/Developer's Kit Version 8	JDBC ドライバとして HiRDB Type4 JDBC Driver を使用する場合に必要になります。
Message Queue サーバと接続する	TP1/Message Queue - Access	• TP1/Message Queue - Access	―
OpenTP1 の SPP と接続する	uCosminexus TP1 Connector	• uCosminexus TP1 Connector	―
	TP1/Client/J	• TP1/Client/J	―

（凡例）－：該当しません。

注　モジュールは，J2EE サーバのプロセスに含まれて動作します。

表 2-8　機能ごとに必要なプロセス（アプリケーションサーバ以外によって提供されるもの）と提供するソフトウェア

機能	必要なプロセス	提供するソフトウェア
クラスタソフトウェアを使用して障害時に系を切り替える	Microsoft Cluster Service	Microsoft Cluster Service
	HA モニタ	HA モニタ

2.6 システムの目的に応じたアプリケーションの構成を決める（バッチ処理を実行する業務の場合）

この節では，システムの目的に応じたアプリケーションの構成の検討について説明します。この節で説明するのは，バッチアプリケーションの場合の構成です。また，必要となるソフトウェアについても説明します。

アプリケーションサーバで実現できるアプリケーションの機能については，次に示すマニュアルの機能の分類に関する説明を参照してください。

- マニュアル「Cosminexus 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「1.1　機能の分類」
- マニュアル「Cosminexus 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「1.1　機能の分類」
- マニュアル「Cosminexus 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「1.1　機能の分類」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「1.1　機能の分類」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.1　機能の分類」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「1.1　機能の分類」

2.6.1 動作させるバッチアプリケーションの検討

システムで動作させるバッチアプリケーションについて検討します。**バッチアプリケーション**とは，バッチ処理として実行する，定型的・定期的な処理を実装した Java アプリケーションのことです。

システム構成の基本的な部分は，バッチアプリケーションで実装した処理内容によって決まります。例えば，データベース上のデータを参照・更新するような処理を実装した場合は，トランザクションの管理方法や，リソースとの接続方法を検討する必要があります。また，ほかの J2EE サーバ上の業務処理プログラム（Enterprise Bean）を呼び出す処理を実装した場合は，サーバ間の連携方法について，検討する必要があります。

バッチアプリケーションを実行する場合のシステム構成の検討方法については，「4.　システム構成の検討（バッチアプリケーション実行基盤）」を参照してください。

システムの目的に応じて使用できる機能については，次に示すマニュアルのシステムの目的と機能の対応に関する説明を参照してください。

- マニュアル「Cosminexus 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「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus 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「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus 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「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「1.2　システムの目的と機能の対応」
- マニュアル「Cosminexus 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　システムの目的と機能の対応」
- マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「1.2　システムの目的と機能の対応」

2.6.2　使用するプロセスの検討と必要なソフトウェアの準備

アプリケーションサーバのシステムでは，使用するプロセスの種類とその配置によってシステム構成が決まります。

ここでは，まず，必要なプロセスとソフトウェアについて説明します。次に，使用する機能に応じて必要なプロセス，モジュールおよびソフトウェアについて説明します。なお，これらのプロセス，モジュール，ソフトウェアをどのようにシステムに配置するかについては，「4.　システム構成の検討（バッチアプリケーション実行基盤)」で説明します。

(1) 必要なプロセス

必要なプロセスを次に示します。これらは，使用する機能に関係なく共通して必要なプロセスです。アプリケーションサーバによって提供されます。

- バッチサーバ
- PRF デーモン

(2) 使用する機能に応じて必要なプロセスおよびモジュール

使用する機能に応じて必要なプロセスおよびモジュールについて説明します。アプリケーションサーバによって提供されるものと，アプリケーションサーバ以外のソフトウェアによって提供されるものがあります。

使用する機能ごとに必要なプロセスのうち，アプリケーションサーバによって提供されるものを次の表に示します。これらのプロセスは，アプリケーションサーバをインストールしたマシンで起動できます。

表 2-9 機能ごとに必要なプロセスまたはモジュール（アプリケーションサーバによって提供されるもの）

機能	必要なプロセス
Management Server を利用して運用管理する	Management Server
	運用管理エージェント
CTM を利用してバッチアプリケーションの実行をスケジューリングする	CTM デーモン
	CTM レギュレータ
	CTM ドメインマネジャ
	グローバル CORBA ネーミングサービス
	スマートエージェント

使用する機能ごとに必要なプロセスおよびモジュールのうち，アプリケーションサーバ以外の製品が提供するプロセスおよびモジュールを，表 2-10 および表 2-11 に示します。

表 2-10 機能ごとに必要なモジュール（アプリケーションサーバ以外によって提供されるもの）と提供するソフトウェア

機能	モジュール	提供するソフトウェア	備考
データベース（HiRDB）と接続する	HiRDB Type4 JDBC Driver	• HiRDB Server Version 9 • HiRDB Server with Additional Function Version 9 • HiRDB/Run Time Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB Developer's Suite Version 9 • HiRDB/Parallel Server Version 8 • HiRDB/Single Server Version 8 • HiRDB/Run Time Version 8 • HiRDB/Developer's Kit Version 8	JDBC ドライバとして HiRDB Type4 JDBC Driver を使用する場合に必要になります。
データベース（Oracle）と接続する	Oracle JDBC Thin Driver	• Oracle JDBC Thin Driver	JDBC ドライバとして Oracle JDBC Thin Driver を使用する場合に必要になります。
データベース（SQL Server）と接続する	SQL Server 2000 Driver for JDBC	• SQL Server 2000 Driver for JDBC	SQL Server 2000 に接続する場合に必要になります。
	SQL Server 2005 JDBC Driver	• SQL Server 2005 JDBC Driver	SQL Server 2005 に接続する場合に必要になります。

機能	モジュール	提供するソフトウェア	備考
	SQL Server JDBC Driver	• SQL Server JDBC Driver	SQL Server 2008 に接続する場合に必要になります。
データベース（XDM/RD E2）と接続する	HiRDB Type4 JDBC Driver	• HiRDB/Parallel Server Version 8 • HiRDB/Single Server Version 8 • HiRDB/Run Time Version 8 • HiRDB/Developer's Kit Version 8	JDBC ドライバとして HiRDB Type4 JDBC Driver を使用する場合に必要になります。

注　モジュールは，バッチサーバのプロセスに含まれて動作します。

表 2-11　機能ごとに必要なプロセス（アプリケーションサーバ以外によって提供されるもの）と提供するソフトウェア

機能	必要なプロセス	提供するソフトウェア
クラスタソフトウェアを使用して障害時に系を切り替える	Microsoft Cluster Service	Microsoft Cluster Service
	HA モニタ	HA モニタ

2.7 運用方法を検討する

この節では，アプリケーションサーバのシステムの運用方法を検討するときに考慮することについて説明します。

アプリケーションサーバのシステムの運用では，Management Server というプロセスを利用できます。Management Server を利用すると，アプリケーションサーバのシステムを構成する複数のプロセスを**論理サーバ**として扱い，一括して運用できるようになります。

なお，業務システムは一般的に，アプリケーションサーバだけではなくほかのプログラムで構築されるシステムと組み合わせて構築されています。運用方法の検討では，アプリケーションサーバのシステムのほかに，これらのプログラムも合わせてシステム全体をどのように運用するかを検討する必要があります。

2.7.1 JP1 と連携したシステムの運用

Management Server を利用して運用している場合，日立の統合システム運用管理用ミドルウェア JP1 のプログラムと連携して，次のようなシステムの運用を実現できます。

- JP1/IM と連携したシステム全体の集中監視
- JP1/IM - CM と連携したシステムの構成定義および構成管理（Smart Composer 機能を使用する場合）
- JP1/AJS と連携したシステム全体の自動運転
- JP1/AJS2 - SO と連携したシナリオを使用したシステム全体の自動運転（Smart Composer 機能を使用する場合）

それぞれのプログラムとの連携によって実現できる機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」を参照してください。

なお，Management Server を利用しないで運用している場合でも，JP1/AJS を利用したシステムの自動運転は実現できます。ただし，その場合，JP1/AJS のジョブとして，サーバ管理コマンドなどを一から定義する必要があります。

2.7.2 クラスタソフトウェアと連携したシステムの運用

クラスタソフトウェアと連携すると，障害発生時などに自動的に**系切り替え**ができます。

また，J2EE サーバをクラスタ構成にしている場合，N 個の J2EE サーバに対して 1 個のリカバリ専用サーバを配置することで，障害発生時にトランザクションを解決してリソースを解放できます。ただし，この機能は，バッチサーバには該当しません。

アプリケーションサーバのシステムでは，OS ごとに，次のクラスタソフトウェアを使用

できます。

● **Windows の場合**
Microsoft Cluster Service

● **AIX，HP-UX または Linux の場合**
HA モニタ

Solaris の場合，クラスタソフトウェアと連携したシステム運用はできません。

クラスタソフトウェアと連携するためには，システムを Management Server を利用して運用している必要があります。

クラスタソフトウェアと連携したシステムの運用をする場合のシステム構成については，「3.11　クラスタソフトウェアを使用した障害時の系切り替えを検討する」を参照してください。

3 システム構成の検討（J2EE アプリケーション実行基盤）

この章では，J2EE アプリケーション実行基盤を構築する場合のシステム構成について説明します。システムを設計する流れに沿って，それぞれの設計項目でのシステム構成の標準的なパターンを示します。また，それぞれのポイントで意識する必要があるコンポーネント，プロセスおよび処理の流れについて説明します。
バッチアプリケーション実行基盤のシステム構成を検討する場合は，「4. システム構成の検討（バッチアプリケーション実行基盤）」を参照してください。

3.1 システム構成を検討するときに考慮すること

この節では，アプリケーションサーバを使用するシステムの構成を検討するときに考慮する必要があることについて説明します。

システム構成を検討する場合，J2EE アプリケーションで使用する機能に応じて，その機能を実現するために必要なプロセスを意識し，それぞれを各マシンに適切に配置することが必要です。このとき，アプリケーションサーバの要件である，信頼性，可用性なども十分に考慮する必要があります。

3.1.1 システムの目的と構成

アプリケーションサーバで構築するシステムは，目的とする業務および実行する J2EE アプリケーションの要件と特徴に応じて，次の 2 種類のシステムに分類できます。

● Web フロントシステム

● バックシステム

Web フロントシステムは，Web ベースのシステムの場合に，フロントエンドである Web ブラウザから送信されるリクエストを受け付けて，そのリクエストを処理するシステムです。このシステムでは，アプリケーションサーバ上の J2EE サーバで，サーブレット，JSP および Enterprise Bean が動作します。

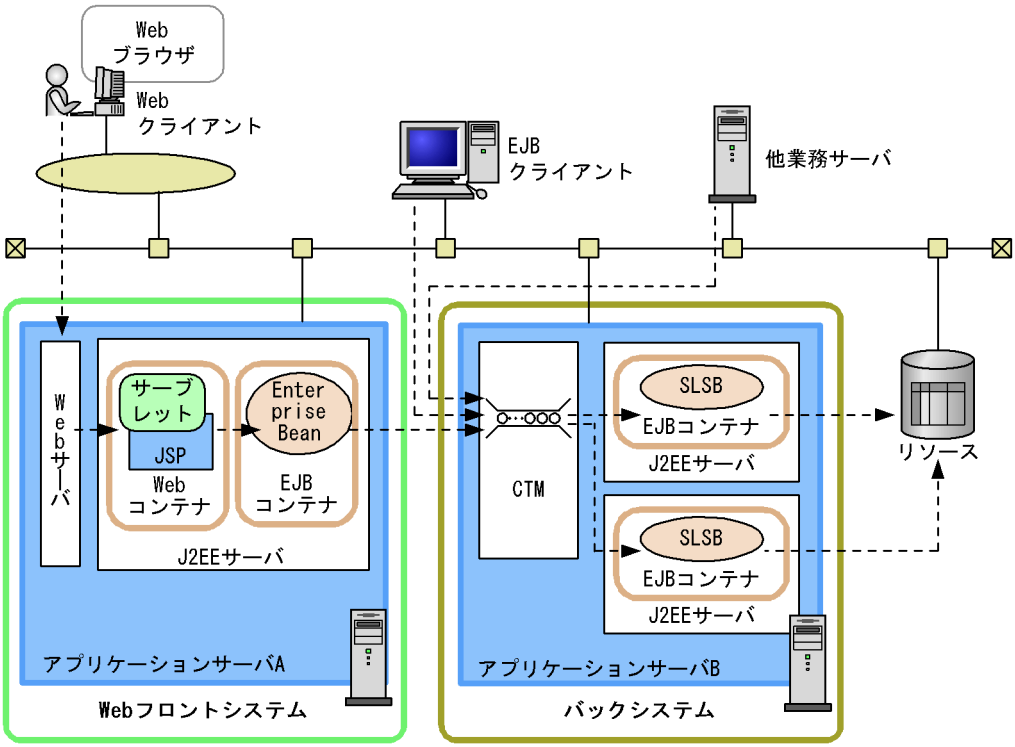
Web フロントシステムの背後で動作する，複数の業務システムに共通な業務サービスを実行するためのシステムが，**バックシステム**です。バックシステムにリクエストを送信するのは，次のようなコンポーネント，アプリケーションまたはシステムです。

- Web フロントシステム上で動作しているサーブレット，JSP または Enterprise Bean
- EJB クライアントマシンで動作している EJB クライアントアプリケーション
- ほかの業務システム

アプリケーションサーバのシステムは，目的や規模に応じて，Web フロントシステムとバックシステムを一つまたは複数組み合わせて構成されます。

Web フロントシステムとバックシステムの構成例を次の図に示します。

図 3-1　Web フロントシステムとバックシステムの構成例



（凡例）

- - ▶：リクエストの流れ

注　SLSB：Stateless Session Bean

アプリケーションサーバのシステム構成を検討する場合，まず，システムの基本構成として，システム全体をどのようなシステムの組み合わせで構成するかを検討します。次に，構成要素であるそれぞれのシステムの目的と，クライアントからアクセスされるポイントを明確にします。それから，信頼性，性能，拡張性などアプリケーションサーバに共通して求められる要件や，EIS との接続や負荷分散の実現などそのシステム独自の要件を満たすためにはどのようにソフトウェアやプロセスを配置するのがよいのかを検討し，最適なシステム構成を設計していきます。

3.1.2　システム構成の設計手順

システム構成は，次の流れで設計します。

図 3-2　システム構成を設計する流れ（J2EE アプリケーション実行基盤の場合）

注　使用する機能に応じた構成を検討する順序は任意です。

(1) アプリケーションの構成を検討する

各システム上で動作するアプリケーションで使用するコンポーネントの構成を明確にし

て，アプリケーションのどのコンポーネントをアクセスポイントにするかを決定します。

アクセスポイントとは，クライアントからリモートでアクセスされる場合に，アプリケーションのリクエスト受信窓口として動作するコンポーネントです。アクセスポイントになるコンポーネントの種類によって，実現できるシステム構成が決まります。

また，アプリケーションがデータベースなどのリソースに接続するかどうかを決めます。接続するリソースに対応して，使用するリソースアダプタが決まります。

なお，ここで検討したアクセスポイントになるコンポーネントの種類によって，それ以降のシステム構成の設計で検討が必要な項目が異なります。アクセスポイントになるコンポーネントの種類ごとに検討が必要な項目を，次の表に示します。

表 3-1　アクセスポイントになるコンポーネントの種類ごとに検討が必要な項目

設計項目	アクセスポイントになるコンポーネントの種類				参照先
	Web フロントシステム	バックシステム			
	サーブレット／JSP	Session Bean／Entity Bean	CTM を使用する場合の Stateless Session Bean	Message-driven Bean	
クライアントとサーバの構成	◎	◎	◎	―	3.4
サーバ間での連携方法	―	△	△	―	3.5
トランザクションの種類	◎	◎	◎	◎	3.6
ロードバランスクラスタによる負荷分散方式	△	△	△	―	3.7
サーバ間での非同期通信	―	―	―	◎	3.8
運用管理プロセスの配置	◎	◎	◎	◎	3.9
セッション情報を引き継ぐための構成	△	△	―	―	3.10
クラスタソフトウェアを使用した系切り替えを実現するための構成	△	△	△	△	3.11
ファイアウォールの配置	△	△	△	△	3.12
DMZ へのリバースプロキシの配置	△※	―	―	―	3.13
性能解析トレースファイルを出力するプロセスの配置	◎	◎	◎	◎	3.14
アプリケーションサーバ以外の製品との連携	△	△	△	△	3.15
任意のプロセスの運用管理	△	△	△	△	3.16

（凡例）

◎：必ず検討する項目。
△：必要に応じて検討する項目。
－：検討する必要がない項目。

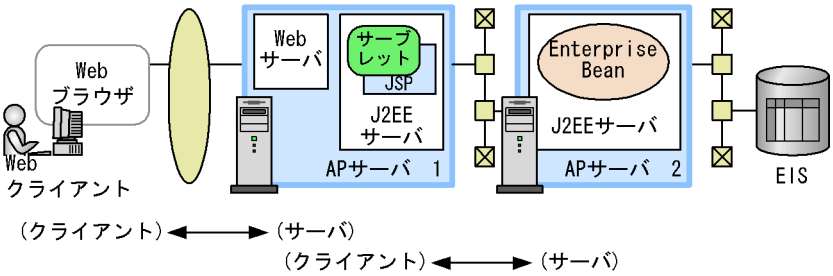
注※　インターネットに接続するシステムで Web サーバとしてインプロセス HTTP サーバを使用する場合は必ず検討してください。

(2) クライアントとサーバの構成を検討する

アクセスポイントになるコンポーネントに応じて，クライアントとサーバの対応を明確にします。システムに配置したアプリケーションサーバは，その役割に応じて，クライアントとして機能したり，サーバとして機能したりします。

クライアントとサーバの構成の考え方を次の図に示します。

図 3-3　クライアントとサーバの構成の考え方



注　APサーバ：アプリケーションサーバ

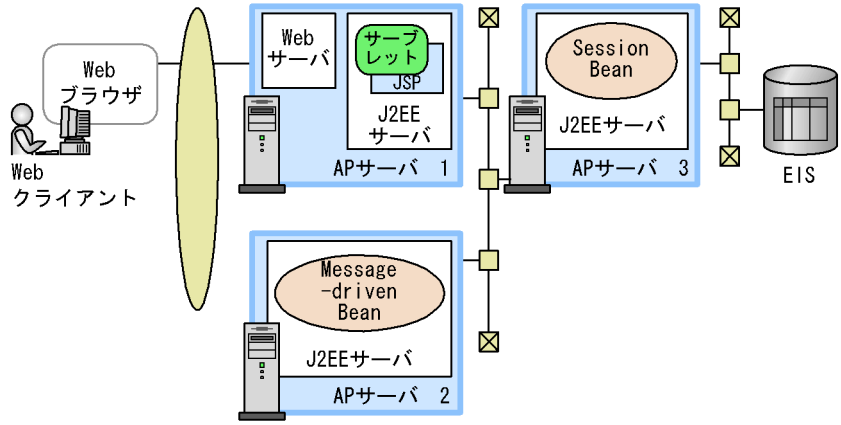
この構成の場合，AP サーバ 1 は Web クライアントに対するサーバに当たり，アクセスポイントはサーブレット／JSP になります。また，AP サーバ 1 は AP サーバ 2 に対してのクライアントになります。AP サーバ 2 は AP サーバ 1 に対するサーバに当たり，アクセスポイントは Enterprise Bean になります。

(3) サーバ間での連携方法を検討する

サーバが複数ある場合に，サーバ間で連携するかどうか，連携する場合はどのように連携するかを検討します。サーバ間連携とは，クライアントから見て垂直に並んだ複数のサーバから，ほかのサーバ上にあるアクセスポイントのコンポーネントを呼び出して処理を実行する連携方法です。

サーバ間連携の考え方を次の図に示します。

図 3-4　サーバ間連携の考え方



注　APサーバ：アプリケーションサーバ

この構成の場合，AP サーバ 1 と AP サーバ 2 は，AP サーバ 3 のクライアントになり，それぞれのサーブレット，JSP または Message-driven Bean から，AP サーバ 3 の Session Bean を呼び出します。

サーバ間連携と同時に，必要に応じて，アプリケーションを分割するなど，アプリケーションの形態も見直します。また，複数のシステム間で連携する場合も，それぞれのシステムに含まれるサーバ間の連携方法を検討する必要があります。

(4) トランザクションの種類を検討する

データベースなどのリソースを使用するシステムの場合に，トランザクション管理の対象になるリソースの数に応じて，使用するトランザクションの種類を検討します。

(5) ロードバランスクラスタによる負荷分散方式を検討する

システムの可用性を高めるために，ロードバランスクラスタ構成によって負荷を分散するかどうか検討します。負荷を分散する場合は，アクセスポイントになるコンポーネントの種類に応じてどの方式で実現するかを検討します。

(6) サーバ間の非同期通信をするための構成を検討する

Message-driven Bean を使用してアプリケーションサーバ間での非同期通信をする場合に，利用する製品に応じたシステムの構成を検討します。なお，Message-driven Bean による負荷分散についても，あわせて検討します。

(7) 運用管理プロセスの配置を検討する

運用管理プロセス（Management Server）を利用する場合に，管理対象にする範囲によって，Management Server をどのサーバマシンに配置するかを検討します。

（8） セッション情報の引き継ぎを検討する

Web フロントシステムで動作する J2EE アプリケーションまたは J2EE サーバに障害が発生した場合に，セッション情報をほかの J2EE サーバに引き継ぐための構成について検討します。

（9） クラスタソフトウェアを使用した障害時の系切り替えまたはリソース解放を実現するための構成を検討する

一つのアプリケーションサーバに障害が発生した場合およびシステムを保守する場合を考慮して，クラスタソフトウェアによって系切り替えを実行して，システムの運用を続行させるための構成を検討します。このとき，実行系と待機系がそれぞれ相互に切り替え対象になる構成（相互スタンバイ構成）も検討できます。また，トランザクションサービスを使用している場合は，障害発生時にリソースを解放するためのリカバリサーバを利用するための構成も検討します。

なお，この構成は，Solaris の場合には使用できません。

（10） ファイアウォールの配置を検討する

アプリケーションサーバおよびリソースのセキュリティを確保するための，ファイアウォールの配置について検討します。ファイアウォールは，アクセスポイントになるコンポーネントの前に配置して，アクセスポイントに対する不正なアクセスを防止します。ファイアウォールを配置するポイントは，アクセスポイントによって決まります。なお，「3.12　ファイアウォールを使用する構成を検討する」では，基本的なファイアウォールの配置についてだけを示します。侵入検知システムも含めたセキュリティ構成の詳細については，「10.10.2　ファイアウォールと侵入検知システムを配置する」を参照してください。

（11） DMZ へのリバースプロキシの配置を検討する

インターネットと接続するシステムの場合などには，アプリケーションサーバおよびリソースのセキュリティを確保するために，DMZ へのリバースプロキシの配置を検討します。

（12） 性能解析トレースファイルを出力するプロセスを配置する

性能解析に使用するプロセスである PRF デーモン（パフォーマンストレーサ）の配置について検討します。

（13） アプリケーションサーバ以外の製品との連携を検討する

システムの集中監視，構成管理，自動運転などをしたい場合には，必要に応じて JP1 などのアプリケーションサーバ以外の製品との連携を検討します。

（14） 任意のプロセスを運用管理の対象にする

ユーザが定義する任意のプロセスをユーザサーバとして Management Server による運

用管理の対象にしたい場合には，ユーザサーバの配置について検討します。

（15）そのほかの構成を検討する

(14) までで検討した以外の構成について検討します。また，必要に応じて，07-00 より前のアプリケーションサーバで構築しているシステムとの互換性を持つシステム構成も検討します。

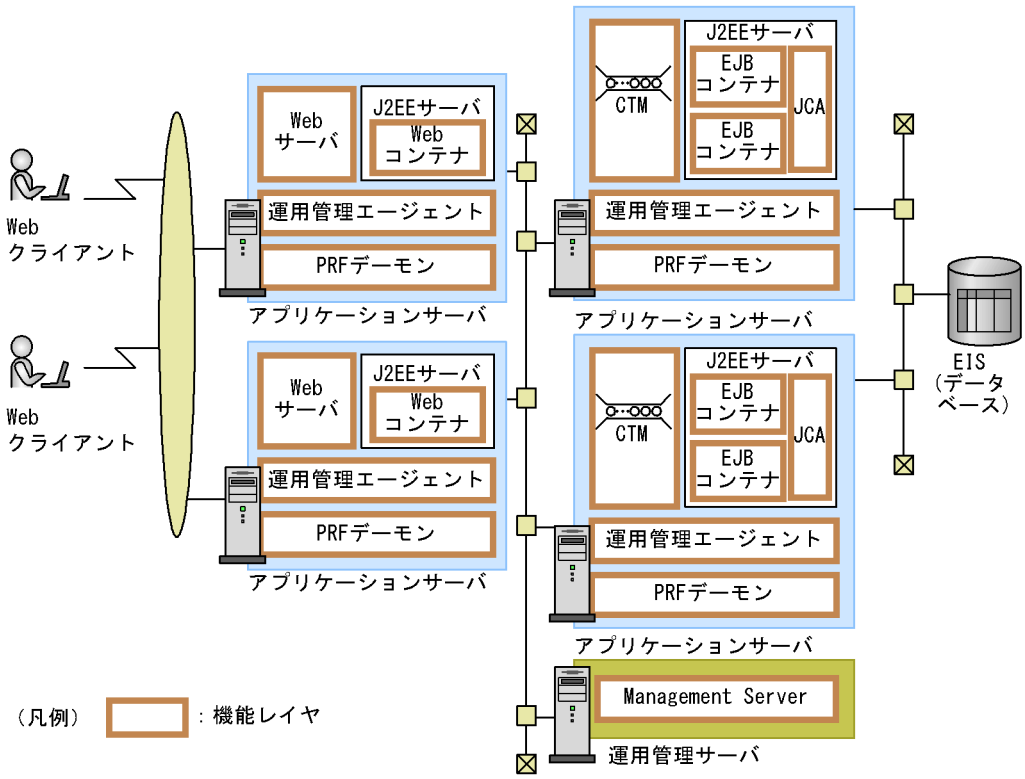
3.1.3　システム構成の考え方

ここでは，システム構成を検討するときの基本的な考え方について説明します。アプリケーションサーバで構築する実行環境は，複数のプロセスで構成されます。また，各プロセスには，提供する機能ごとに，複数のレイヤがあります。このレイヤを，**機能レイヤ**といいます。

実行環境では，各機能レイヤをそれぞれ異なるサーバに配置できます。配置は，システムの規模や目的に応じて検討する必要があります。

機能レイヤの配置例を次の図に示します。

図 3-5　機能レイヤの配置例



ここでは，次の機能レイヤを実行環境に配置する場合の構成の考え方について説明しま

す。

- Web サーバ
- Web コンテナ
- CTM
- EJB コンテナ
- JCA
- EIS
- ネーミングサービス
- 運用管理サーバ

（1） Web サーバと Web コンテナ（J2EE サーバ）の配置

Web サーバは，Web コンテナに対する HTTP リクエストのディスパッチ処理のほか，業務処理に含まれる静的コンテンツの処理を実行します。Web コンテナは，J2EE サーバの一部として動作して，サーブレットおよび JSP を実行するための基盤になる機能です。Web サーバと Web コンテナは同じホストまたは異なるホストに配置できます。なお，インプロセス HTTP サーバを使用する場合は，Web サーバは不要です。

Web サーバと Web コンテナを動作させる J2EE サーバの関係は，1 対 1 で配置することをお勧めします。なお，負荷分散をする場合は，Web サーバのフロントに負荷分散機を配置することをお勧めします。

これを踏まえた上で，次の説明を参照して，システム構成を決定してください。

- 「3.4.1　サーブレットと JSP をアクセスポイントに使用する構成（Web サーバ連携の場合）」
- 「3.4.2　サーブレットと JSP をアクセスポイントに使用する構成（インプロセス HTTP サーバを使用する場合）」
- 「3.17.1　Web サーバとアプリケーションサーバを異なるマシンに配置する構成」

（2） CTM と EJB コンテナ（J2EE サーバ）の配置

CTM は，OLTP 技術に対応して，クライアントからのリクエストのスケジューリングをする機能です。EJB コンテナは，J2EE サーバの一部として動作して，Enterprise Bean の実行および通信，トランザクション管理などのシステムレベルのサービスを提供するための基盤になる機能です。

CTM では，EJB コンテナに対して送信された IIOP リクエストのスケジュール処理と負荷分散処理を実行します。CTM は IIOP リクエストを処理するために特化された機能レイヤなので，EJB コンテナと別のホストに配置する必要はありません。IIOP リクエストの負荷分散は，CTM 間のデータ転送で実現されます。ただし，CTM による負荷分散の対象になるのは，Stateless Session Bean だけです。

CTM と EJB コンテナの関係は，1 対多にして，それぞれロードバランスクラスタ構成

にすることをお勧めします。これによって，スループットが向上します。また，拡張性と性能が高いシステムを構築できます。また，クラスタ構成にすれば障害発生時の部分縮退や部分回復ができるようになるので，信頼性と可用性も向上します。

これを踏まえた上で，次の説明を参照して，システム構成を決定してください。

- 「3.4.4　CTM を使用する場合に Stateless Session Bean をアクセスポイントに使用する構成」
- 「3.7.4　CTM を利用した負荷分散（Stateless Session Bean の場合）」

（3） JCA と EIS の配置

JCA は，既存システムやデータベースなどの EIS と接続するためのサポート機能を提供します。

EIS に対して，JCA ではコネクションプーリング処理をします。EIS を一つのサーバで構築している場合，JCA と EIS の関係は，多対 1 になります。EIS を複数のサーバで構築している場合は，JCA と EIS の関係は多対多になります。JCA では，EIS との接続で障害が発生した場合に，自動回復する機能を持っているので，信頼性と可用性が確保されています。

（4） ネーミングサービスの配置

ネーミングサービスは，名前からオブジェクトを利用できるようにするネーミング管理機能を提供します。ネーミングサービスは，Cosminexus TPBroker によって提供されるサービスです。J2EE サーバのインプロセスで起動することをお勧めします。

インプロセスで起動することによって，アプリケーションサーバ内のプロセス数が削減できます。また，個別プロセスとしての起動処理や停止処理が不要になり，運用性が向上します。

（5） 運用管理プロセスの配置

Management Server は，アプリケーションサーバ全体を運用管理するための機能を集約したサーバです。システムの運用管理，監視の対象範囲となるドメイン内に一つ配置します。

ここでは，J2EE サーバなどのほかのプロセスとは別のホストに配置した構成を使用して説明していますが，同じホストに配置した構成にすることもできます。

運用管理，監視の対象になるプロセスを配置したホストには，それぞれ運用管理エージェントを起動します。運用管理エージェントは，運用管理サーバからの指示を受けて，各ホストで操作を実行します。

運用管理者は，Smart Composer 機能または Management Server のコマンドを使用して，運用管理を実行します。これらのコマンドは，Management Server を配置したホストで実行します。

参考

開発環境やテスト環境で運用管理を実行する場合には，必要に応じて次の機能も使用できます。

- **運用管理ポータル**
  Web ブラウザから Management Server を配置したホストにアクセスして実行します。

なお，運用管理ポータルを使用する場合は，必要に応じて管理クライアントマシンを用意してください。

運用管理プロセスを配置した構成については，「3.9　運用管理プロセスの配置を検討する」を参照してください。

3.2　システム構成の説明について

3 章では，システム構成を検討するための，さまざまなシステム構成パターンを示して，それぞれの特徴について説明します。この節では，システム構成の説明をお読みになる前に確認していただきたい，各システム構成パターンに共通する留意点について説明します。また，この章のシステム構成図で使用する図の凡例についてもあわせて説明します。これらをご確認の上，次節以降をお読みください。

参考

ここで説明する凡例は，「4.　システム構成の検討（バッチアプリケーション実行基盤）」および「10.10.2　ファイアウォールと侵入検知システムを配置する」にも当てはまります。

（1） この章のシステム構成に共通する留意点

各システム構成に共通して，次のことに留意してください。

- この章では，Application Server Standard または Application Server Enterprise を使用した場合のシステム構成について説明しています。必要に応じて，この章で使用している用語を，ご利用の製品に読み替えてください。
  ご利用の製品とこの章で使用している製品の対応を次の表に示します。

表 3-2　ご利用の製品とこの章で使用している製品の対応

ご利用の製品	この章で記載している製品
uCosminexus Developer Professional ※	Application Server および Application Server Enterprise
uCosminexus Developer Standard ※	Application Server
uCosminexus Service Architect ※	Application Server および Application Server Enterprise
uCosminexus Service Platform	

注※　テスト環境で使用している場合にだけ読み替えが必要です。

- システム構成の例では，主に，アプリケーションサーバを役割ごとに 1 台ずつ配置した例を示していますが，実際の構成では，スケーラビリティと可用性を考慮して負荷分散機などを使用したクラスタ構成にすることをお勧めします。これによって，トラブルが発生した場合やサーバのメンテナンスが必要な場合に，サービスが完全に止まることを防げます。負荷分散をする構成については，「3.7　ロードバランスクラスタによる負荷分散方式を検討する」を参照してください。
- この章では，Management Server を利用して運用することにした場合の例を示しています。Management Server を利用しないで運用する場合，次のプロセスは不要ですので，省略してお読みください。
  - Management Server

- 運用管理エージェント

運用管理に必要なプロセスだけを起動するマシンを**運用管理サーバマシン**といいます。なお，Management Server を利用しないで運用する場合，運用管理サーバマシンは不要です。

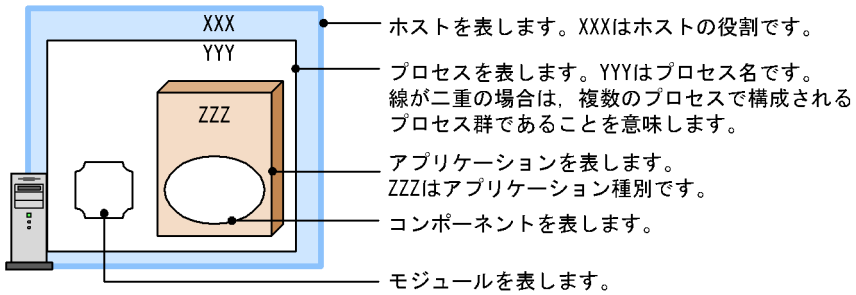
● アクセスポイントがサーブレット／ JSP のとき，この章では主に，Web サーバ連携の場合の構成例を示しています。インプロセス HTTP サーバを使用する場合は，次のプロセスは不要ですので，省略してお読みください。
- Web サーバ

● PRF デーモンは，性能解析トレースファイルを出力するためのプロセスです。アプリケーションサーバには必ず配置し，EJB クライアントには必要に応じて配置します。PRF デーモンの配置についての考え方については，「3.14　性能解析トレースファイルを出力するプロセスを配置する」を参照してください。

（2） システム構成図で使用する図の凡例

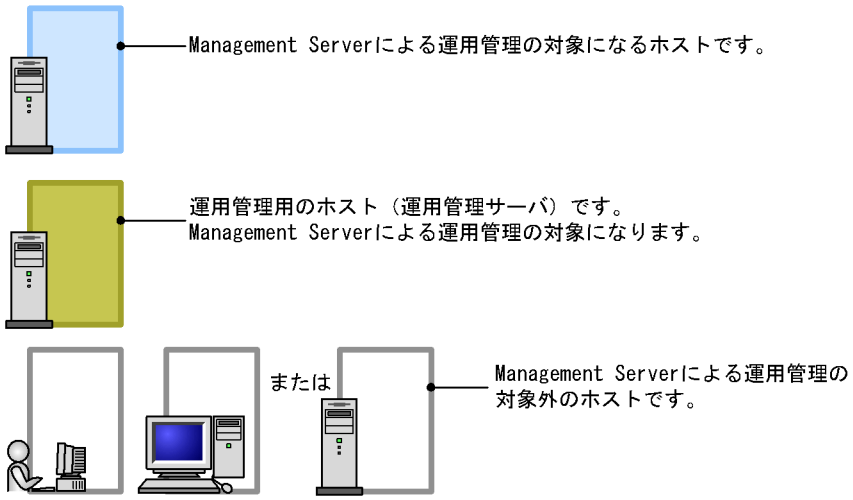
ここでは，3 章および 4 章のシステム構成図で使用する凡例を示します。

図 3-6　システム構成図で使用する図の凡例



また，ホストを表す色は，Management Server によって運用管理する場合のホストの分類を表しています。

図 3-7　システム構成図で使用する図の凡例（ホストの分類）



なお，ここで示した以外の凡例については，それぞれの図で示します。

3.3　アプリケーションの構成を検討する

この節では，アプリケーションの構成の検討方法について説明します。

アプリケーションは，構成するコンポーネントの種類によって分類できます。分類ごとに，クライアント側から見たアクセスポイントがあります。ここでは，アプリケーションを構成するコンポーネントの種類と，各アプリケーションでアクセスポイントになるコンポーネントについて説明します。また，接続するリソースの種類に応じたリソースアダプタの種類についても説明します。

3.3.1　アプリケーションの構成とアクセスポイント

ここでは，アプリケーションを構成するコンポーネントの種類と，それぞれの構成の場合のアクセスポイントについて説明します。

アプリケーションを構成するコンポーネントには，次の種類があります。

- **サーブレットと** JSP
- Session Bean **と** Entity Bean
- Message-driven Bean

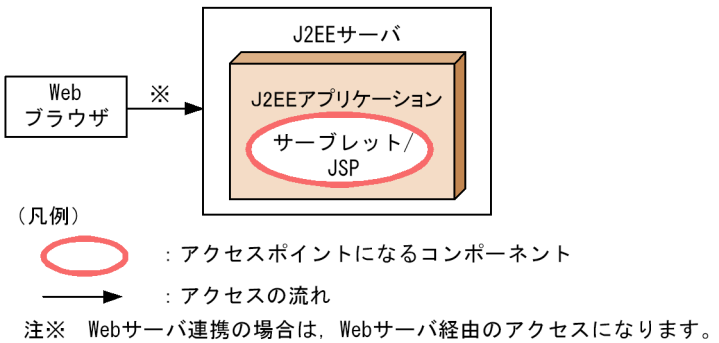
（1）サーブレットと JSP で構成されるアプリケーション

サーブレットと JSP は，クライアントマシンの Web ブラウザに表示するプレゼンテーションを，動的に生成するためのコンポーネントです。クライアントである Web ブラウザから，HTTP または HTTPS によって Web サーバ経由でアクセスされます。

サーブレットと JSP で構成されるアプリケーションの場合，クライアントから見たアクセスポイントになるコンポーネントは，フロントに配置されたサーブレットまたは JSP になります。

サーブレットと JSP で構成されるアプリケーションを次の図に示します。

図 3-8　サーブレットと JSP で構成されるアプリケーション

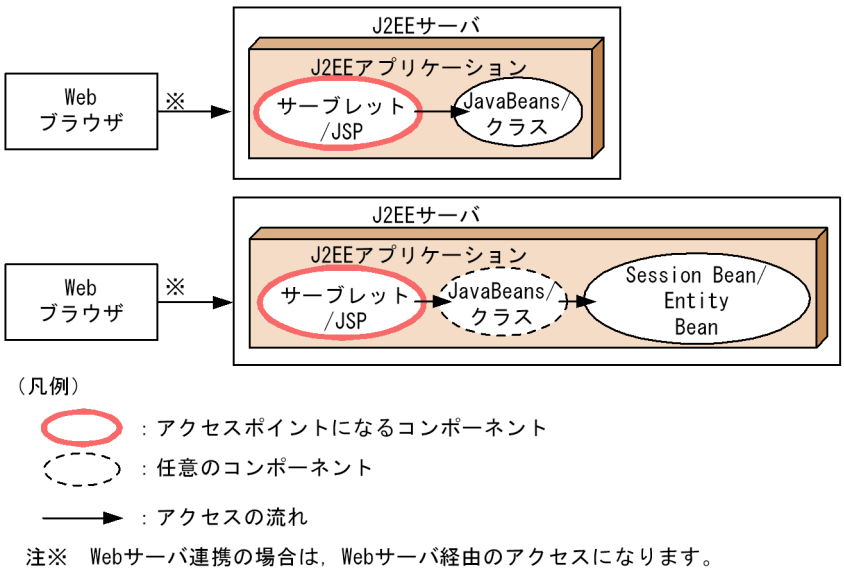


注※　Webサーバ連携の場合は，Webサーバ経由のアクセスになります。

注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

サーブレットまたは JSP からは，JavaBeans や Java クラスなど，ほかのコンポーネントを呼び出せます。また，Session Bean や Entity Bean を呼び出すこともできます。この場合も，クライアントから見たアクセスポイントとなるコンポーネントは，フロントに配置したサーブレットまたは JSP になります。

図 3-9　サーブレットと JSP からほかのコンポーネントを呼び出す場合のアクセスポイント



注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

このアプリケーションは，主に Web フロントシステムで動作します。

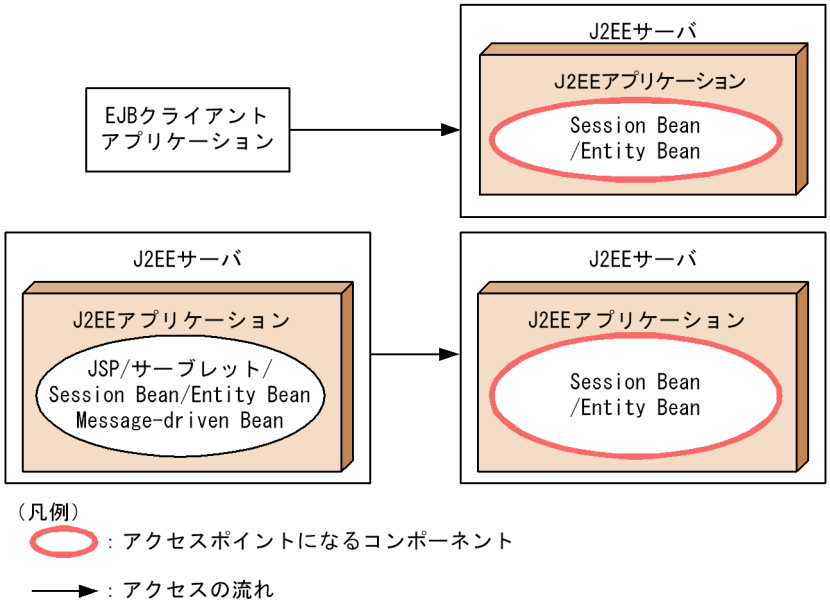
（2）Session Bean と Entity Bean で構成されるアプリケーション

Session Bean と Entity Bean は，ビジネスロジックを実装するためのコンポーネントです。EJB クライアントから，RMI-IIOP によってアクセスされます。なお，**EJB クライアント**とは，Enterprise Bean を呼び出すコンポーネントの総称です。クライアントマシンで動作する EJB クライアントアプリケーション，ほかの J2EE サーバで動作しているサーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean が該当します。

Session Bean と Entity Bean で構成されるアプリケーションの場合，アクセスポイントになるコンポーネントは，フロントに配置された Session Bean または Entity Bean になります。

Session Bean と Entity Bean で構成されるアプリケーションを次の図に示します。

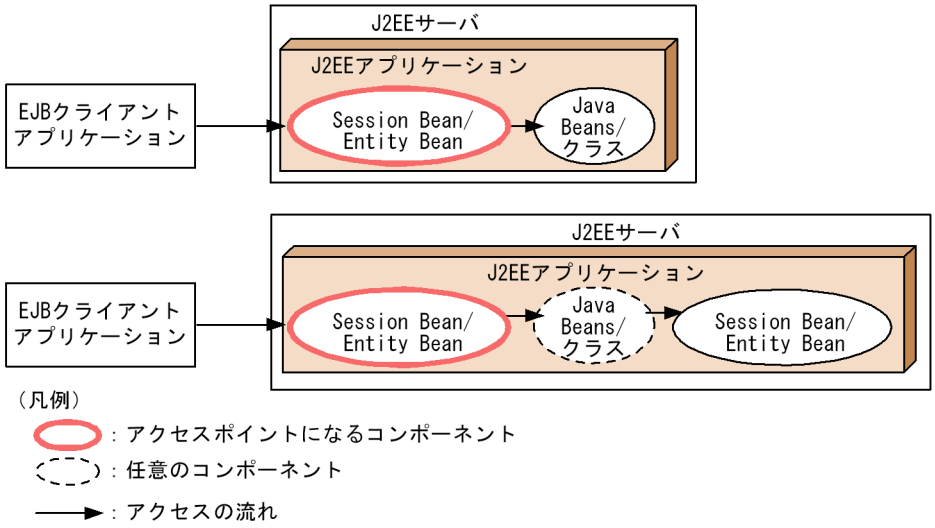
図 3-10　Session Bean と Entity Bean で構成されるアプリケーション



注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

Session Bean または Entity Bean からは，JavaBeans や Java クラスなど，ほかのコンポーネントを呼び出せます。また，ほかの Session Bean や Entity Bean を呼び出すこともできます。ただし，この場合も，クライアントから見たアクセスポイントとなるコンポーネントは，フロントに配置した Session Bean または Entity Bean になります。

図 3-11　Session Bean または Entity Bean からほかのコンポーネントを呼び出す場合のアクセスポイント



注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

このアプリケーションは，主にバックシステムで動作します。

（3） Message-driven Bean で構成されるアプリケーション

Message-driven Bean は，メッセージ駆動型のシステムでビジネスロジックを実装するためのコンポーネントです。次のどれかの方法でアクセスされます。

- Cosminexus JMS プロバイダ経由のアクセス
- TP1/Message Queue および TP1/Message Queue - Access 経由でのアクセス
- データベース（HiRDB または Oracle）および Cosminexus RM 経由でのアクセス
- TP1 インバウンド連携でのアクセス

なお，Message-driven Bean をアクセスポイントとする構成の場合，リソースアダプタとして，次のどれかが必要です。

- CJMSP リソースアダプタ※
- TP1/Message Queue - Access
- Cosminexus RM ※

  Cosminexus RM は，DB Connector for Cosminexus RM ※と組み合わせて使用します。

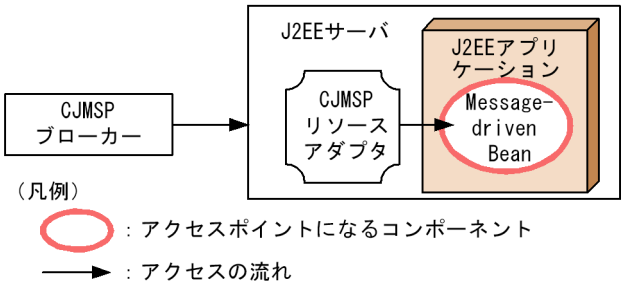
● TP1 インバウンドアダプタ※

注※　アプリケーションサーバが提供するリソースアダプタです。

Message-driven Bean で構成されるアプリケーションの場合，アクセスポイントになるコンポーネントは，Message-driven Bean になります。

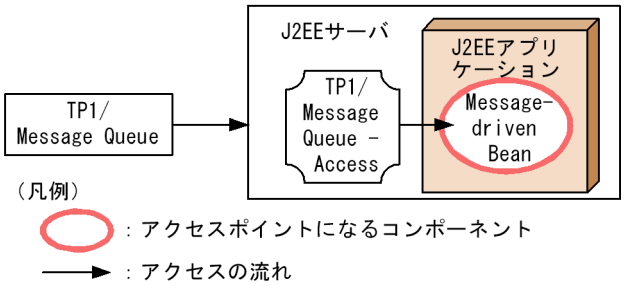
Message-driven Bean で構成されるアプリケーションを次の図に示します。

図 3-12　Message-driven Bean で構成されるアプリケーション（Cosminexus JMS プロバイダ経由の場合）



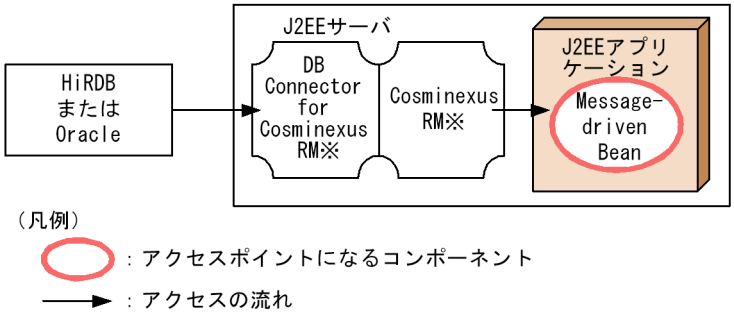
注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

図 3-13　Message-driven Bean で構成されるアプリケーション（TP1/Message Queue - Access 経由の場合）



注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

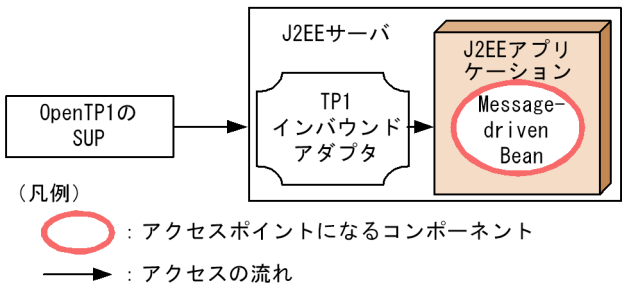
図 3-14　Message-driven Bean で構成されるアプリケーション（Cosminexus RM 経由の場合）



注※　アプリケーションサーバで提供されるリソースアダプタです。

注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

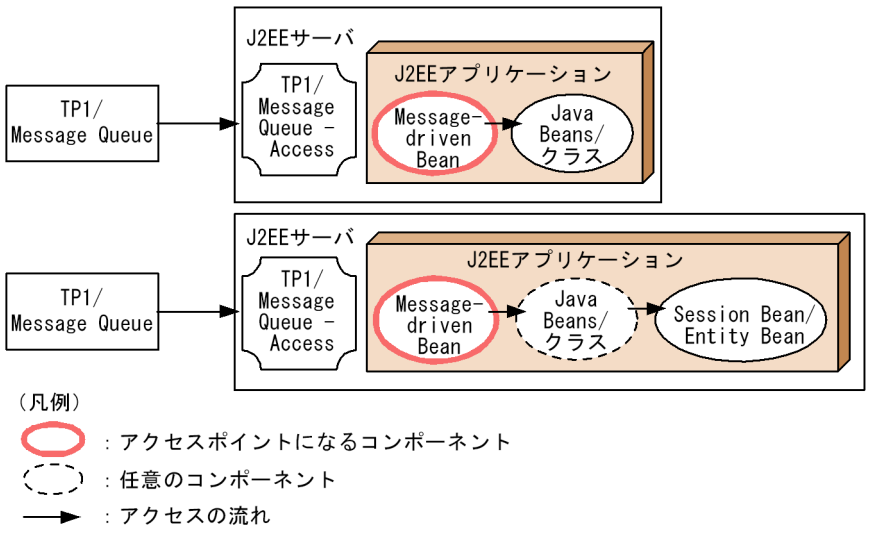
図 3-15　Message-driven Bean で構成されるアプリケーション（TP1 インバウンド連携の場合）



注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

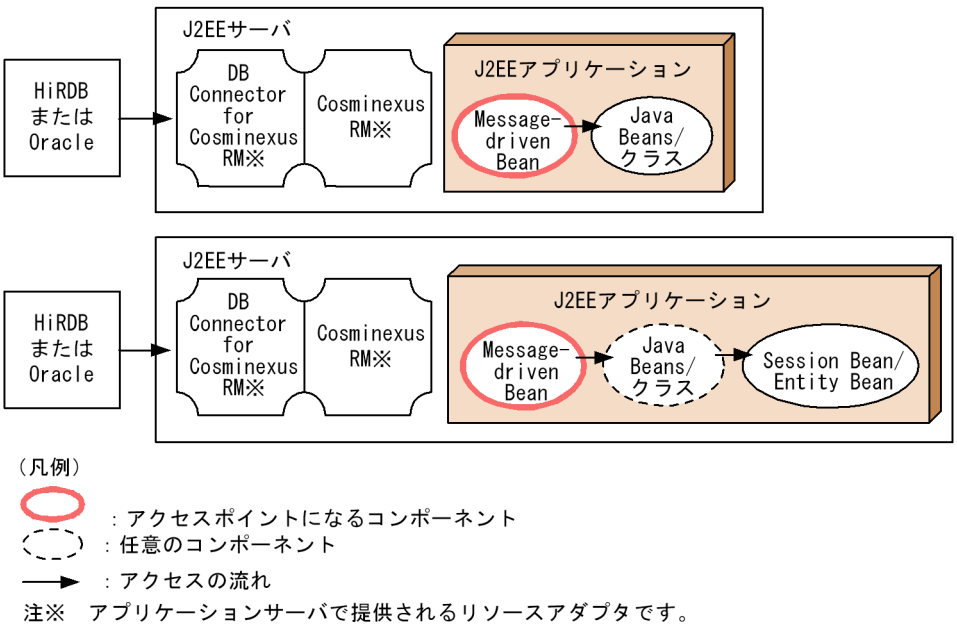
Message-driven Bean からは，JavaBeans や Java クラスなど，ほかのコンポーネントを呼び出せます。また，Session Bean や Entity Bean を呼び出すこともできます。ただし，この場合も，クライアントから見たアクセスポイントとなるコンポーネントは，Message-driven Bean になります。例を示します。

図 3-16　Message-driven Bean からほかのコンポーネントを呼び出す場合のアクセスポイント（TP1/Message Queue - Access 経由の場合）



注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

図 3-17　Message-driven Bean からほかのコンポーネントを呼び出す場合のアクセスポイント（Cosminexus RM 経由の場合）



注※　アプリケーションサーバで提供されるリソースアダプタです。

注　これ以外の凡例については，「3.2　システム構成の説明について」を参照してくだ

さい。

このアプリケーションは，主にバックシステムで動作します。

3.3.2 リソースの種類とリソースアダプタ

アプリケーションサーバのシステムでは，次のリソースと接続できます。

- データベース（HiRDB，Oracle，SQL Server または XDM/RD E2）
- OpenTP1 の TP1/Message Queue
- OpenTP1 の SPP
- OpenTP1 の SUP
- Cosminexus JMS プロバイダの CJMSP ブローカー
- メールコンフィグレーション
- JavaBeans リソース

OpenTP1 の SUP とは，Inbound で接続します。なお，メールコンフィグレーションまたは JavaBeans リソースを使用する場合，リソースアダプタは不要です。

ここでは，アプリケーションが接続するリソースの種類ごとに，使用するリソースアダプタについて説明します。

ポイント

アプリケーションサーバでは，Connector 1.0 または Connector 1.5 仕様に準拠したリソースアダプタを利用できます。
ここで説明する以外のリソースアダプタについては，ご使用のリソースアダプタの説明をご確認ください。

（1）データベースと接続するためのリソースアダプタ（JDBC インタフェースを使用する場合）

JDBC インタフェースを使用してデータベースと接続する場合，リソースアダプタとして DB Connector を使用します。DB Connector を使用すると，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean から，JDBC インタフェースを使用してデータベースにアクセスできます。

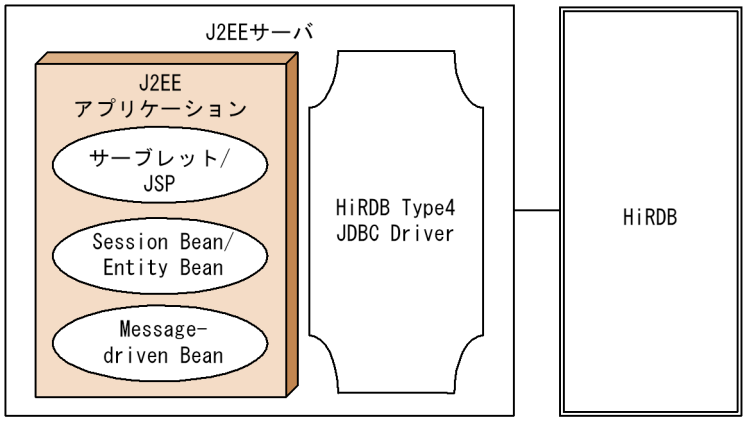
なお，アプリケーションサーバがリソースアダプタを使用してアクセスできるデータベースは，HiRDB，Oracle，SQL Server または XDM/RD E2 です。リソースアダプタを使用してデータベースにアクセスする場合の構成を，データベースの種類ごとに図に示します。

● HiRDB にアクセスする場合の構成

HiRDB に接続する場合，J2EE サーバと同じマシンに，HiRDB Type4 JDBC Driver が必要となります。

HiRDB にアクセスする場合の構成を次の図に示します。

図 3-18　リソースアダプタを使用して HiRDB にアクセスする場合の構成



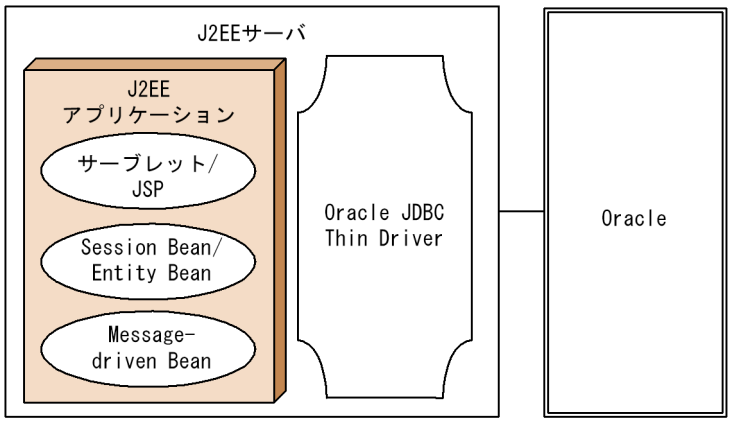
注　凡例については，「3.2　システム構成の説明について」を参照してください。

● **Oracle にアクセスする場合の構成**

Oracle に接続する場合，J2EE サーバと同じマシンに，Oracle JDBC Thin Driver が必要となります。

Oracle にアクセスする場合の構成を次の図に示します。

図 3-19　リソースアダプタを使用して Oracle にアクセスする場合の構成



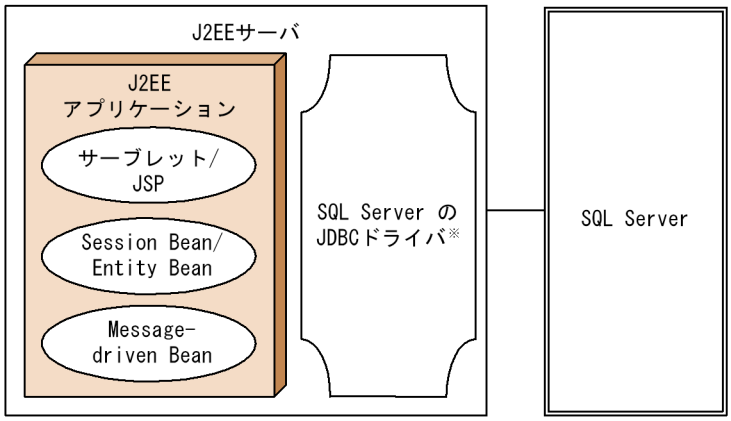
注　凡例については，「3.2　システム構成の説明について」を参照してください。

● **SQL Server にアクセスする場合の構成**

SQL Server に接続する場合，J2EE サーバと同じマシンに，SQL Server の JDBC ドライバが必要となります。

SQL Server にアクセスする場合の構成を次の図に示します。

図 3-20 リソースアダプタを使用して SQL Server にアクセスする場合の構成



注※ SQL Server 2000に接続する場合は，SQL Server 2000 Driver for JDBCになります。
SQL Server 2005に接続する場合は，SQL Server 2005 JDBC Driverになります。

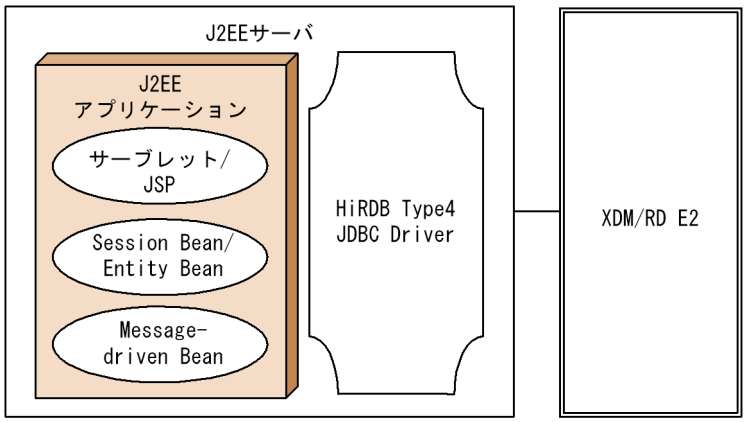
注 凡例については，「3.2 システム構成の説明について」を参照してください。

● **XDM/RD E2 にアクセスする場合の構成**

XDM/RD E2 に接続する場合，J2EE サーバと同じマシンに，HiRDB Type4 JDBC Driver が必要となります。

XDM/RD E2 にアクセスする場合の構成を次の図に示します。

図 3-21 リソースアダプタを使用して XDM/RD E2 にアクセスする場合の構成



注 凡例については，「3.2 システム構成の説明について」を参照してください。

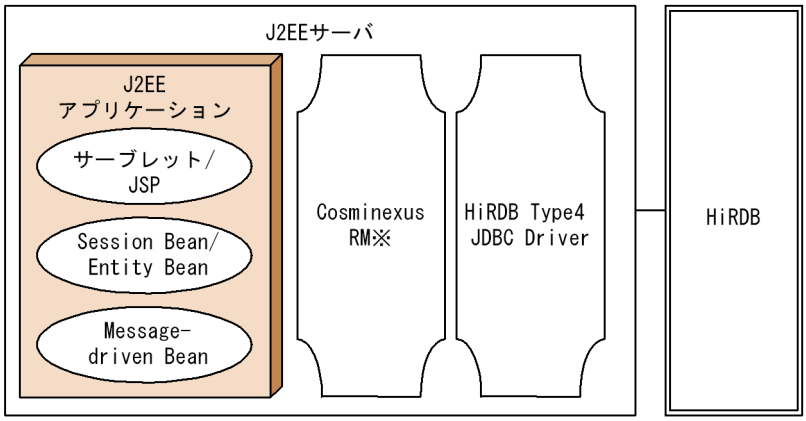
(2) データベースと接続するためのリソースアダプタ（JMS インタフェースを使用する場合）

JMS インタフェースを使用してデータベースと接続する場合，リソースアダプタとして

DB Connector for Cosminexus RM を使用します。DB Connector for Cosminexus RM は，Cosminexus RM と連携するためのリソースアダプタです。Cosminexus RM を使用すると，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean から，JMS インタフェースを使用してデータベース上に実現したキューにアクセスできます。この場合，J2EE サーバ上では，Cosminexus RM のライブラリが動作して，キューのデータはデータベースに保存されます。なお，Cosminexus RM はアプリケーションサーバの構成ソフトウェアです。

Cosminexus RM を使用してデータベースにアクセスする場合の構成を，HiRDB にアクセスする場合と，Oracle にアクセスする場合に分けて，図に示します。

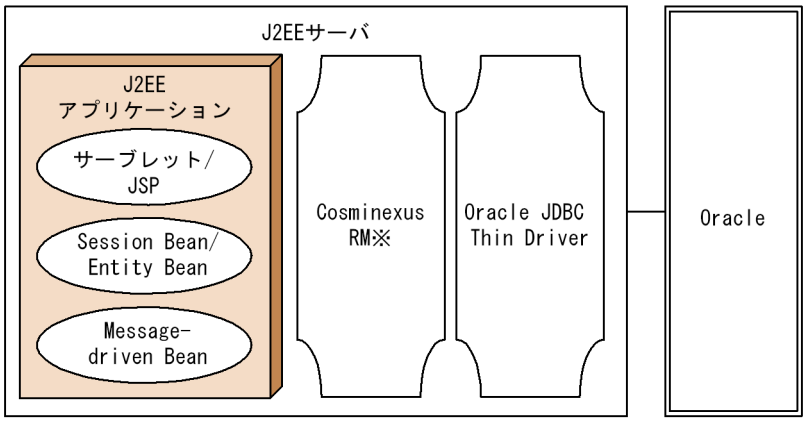
図 3-22　Cosminexus RM を使用して HiRDB にアクセスする場合の構成



注※　アプリケーションサーバで提供されるリソースアダプタです。

注　凡例については，「3.2　システム構成の説明について」を参照してください。

図 3-23　Cosminexus RM を使用して Oracle にアクセスする場合の構成



注※　アプリケーションサーバで提供されるリソースアダプタです。

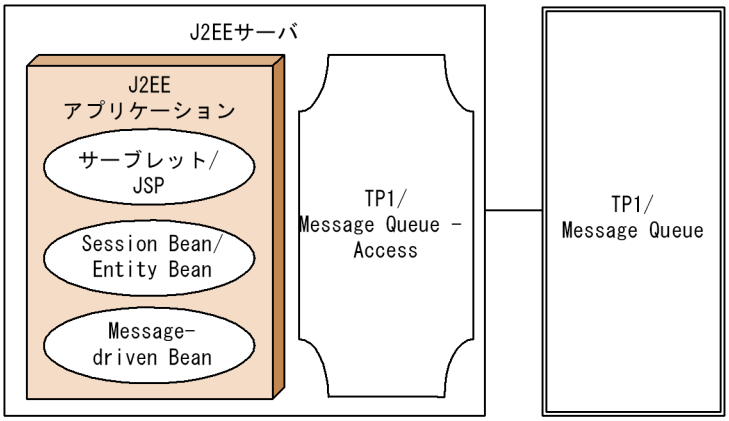
注　凡例については，「3.2　システム構成の説明について」を参照してください。

（3） OpenTP1 の TP1/Message Queue に接続するためのリソースアダプタ

OpenTP1 のメッセージキューイング機能である TP1/Message Queue と接続する場合，リソースアダプタとして TP1/Message Queue - Access を使用します。TP1/Message Queue - Access を使用すると，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean から，JMS インタフェースを使用して TP1/Message Queue にアクセスできます。この場合，J2EE サーバと同じマシンに，TP1/Message Queue - Access が必要となります。

リソースアダプタを使用して TP1/Message Queue にアクセスする場合の構成を，次の図に示します。

図 3-24　リソースアダプタを使用して TP1/Message Queue にアクセスする場合の構成



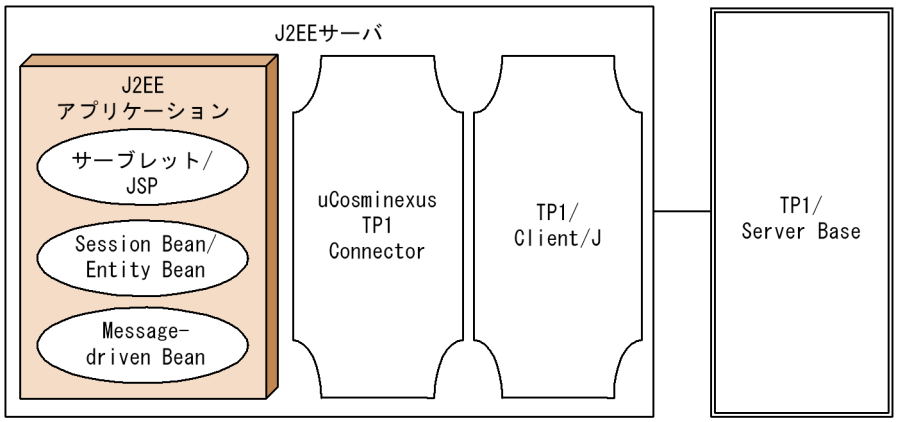
注　凡例については，「3.2　システム構成の説明について」を参照してください。

（4） OpenTP1 の SPP と接続するためのリソースアダプタ

OpenTP1 の SPP（サービス提供プログラム：Service Providing Program）と接続する場合，リソースアダプタとして uCosminexus TP1 Connector および TP1/Client/J を使用します。uCosminexus TP1 Connector および TP1/Client/J を使用すると，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean から，OpenTP1 の SPP にアクセスできます。このとき，J2EE サーバと同じマシンに，uCosminexus TP1 Connector および TP1/Client/J が必要となります。

リソースアダプタを使用して OpenTP1 の SPP にアクセスする場合の構成を，次の図に示します。

図 3-25　リソースアダプタを使用して OpenTP1 の SPP にアクセスする場合の構成



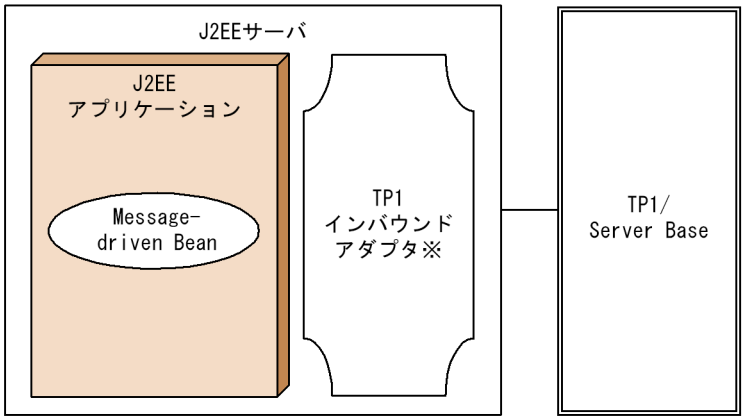
注　凡例については，「3.2　システム構成の説明について」を参照してください。

（5） OpenTP1 の SUP と接続するためのリソースアダプタ

TP1 インバウンド連携機能を利用して OpenTP1 の SUP（サービス利用プログラム：Service Using Program）と接続する場合，リソースアダプタとして **TP1 インバウンドアダプタ**を使用します。TP1 インバウンド連携機能を利用すると，OpenTP1 の SUP から，J2EE サーバ上で動作する Message-driven Bean に Inbound でアクセスできます。このとき，J2EE サーバと同じマシンに，TP1 インバウンドアダプタが必要となります。

リソースアダプタを使用して OpenTP1 の SUP から Inbound でアクセスする場合の構成を，次の図に示します。

図 3-26　リソースアダプタを使用して OpenTP1 の SUP から Inbound でアクセスする場合の構成



注※　アプリケーションサーバで提供されるリソースアダプタです。

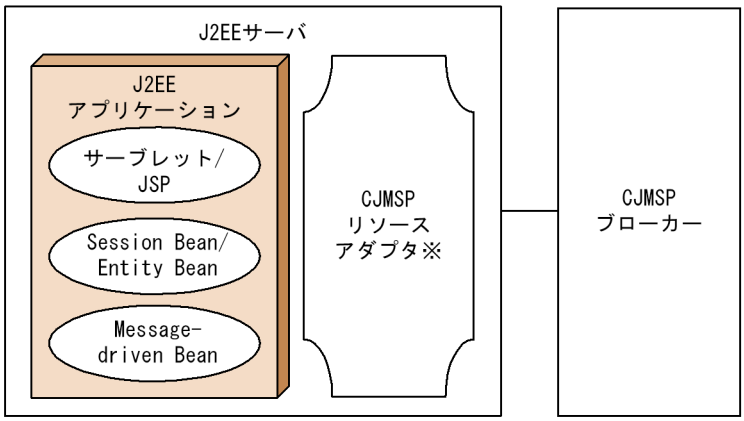
注　凡例については，「3.2　システム構成の説明について」を参照してください。

（6）Cosminexus JMS プロバイダの CJMSP ブローカーと接続するためのリソースアダプタ

Cosminexus JMS プロバイダの CJMSP ブローカーと接続する場合，リソースアダプタとして CJMSP **リソースアダプタ**を使用します。CJMSP リソースアダプタを使用すると，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean から，JMS インタフェースを使用して CJMSP ブローカーにアクセスできます。このとき，J2EE サーバと同じマシンに，CJMSP リソースアダプタが必要となります。

リソースアダプタを使用して CJMSP ブローカーにアクセスする場合の構成を，次の図に示します。

図 3-27　リソースアダプタを使用して CJMSP ブローカーにアクセスする場合の構成



注※　アプリケーションサーバで提供されるリソースアダプタです。

注　凡例については,「3.2　システム構成の説明について」を参照してください。

3.4　クライアントとサーバの構成を検討する

この節では，クライアントとサーバの構成の種類と，それぞれの場合に各マシンに配置するプロセスについて説明します。また，それぞれの構成の特徴についても説明します。

クライアントとサーバの構成を検討するためには，処理を呼び出す側であるクライアントと呼び出される側であるサーバの対応を決めて，サーバ側で動作するアプリケーションのアクセスポイントを明確にする必要があります。

ここでは，次のコンポーネントをアクセスポイントとするクライアントとサーバの構成について説明します。なお，サーブレットと JSP をアクセスポイントにする構成については，Web サーバ連携の場合とインプロセス HTTP サーバを使用する場合に分けて説明します。

● サーブレットと JSP

● Session Bean と Entity Bean

● CTM を使用する場合の Stateless Session Bean

以降，アプリケーションサーバを配置したマシンを**アプリケーションサーバマシン**，クライアントとして使用するマシンを**クライアントマシン**といいます。

3.4.1　サーブレットと JSP をアクセスポイントに使用する構成（Web サーバ連携の場合）

サーブレットと JSP がアクセスポイントになる，Web ベースのシステム構成について説明します。この構成を **Web クライアント構成**といいます。Web フロントシステムで使用できる構成です。ここでは，Web サーバ連携の場合の構成について説明します。

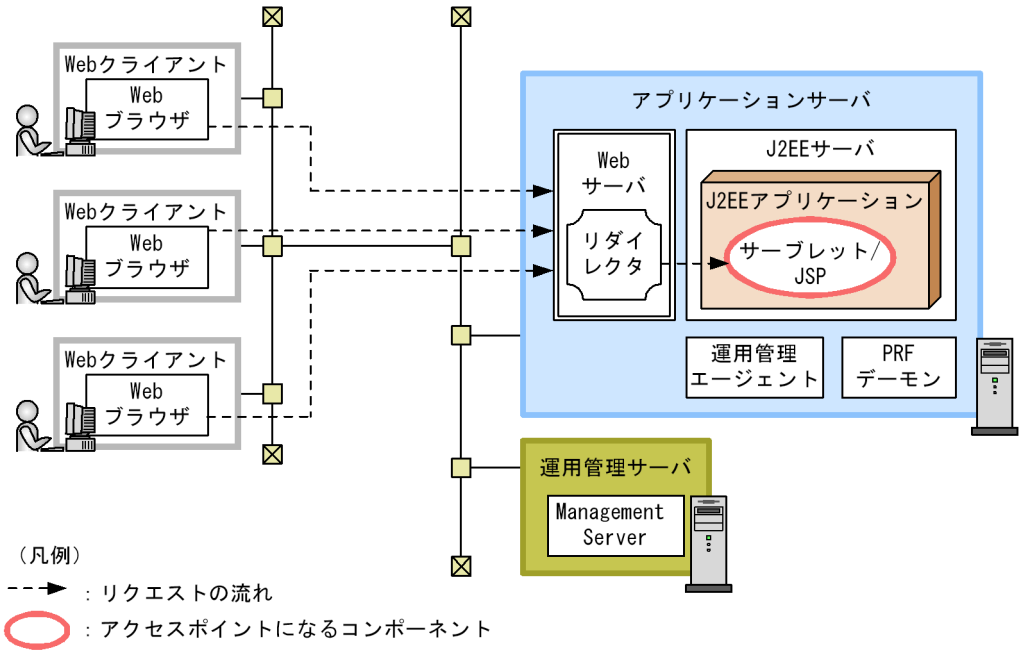
また，この構成でインターネットに接続する場合，セキュリティの観点から，アプリケーションサーバと同じマシンに配置する Web サーバとは別に，DMZ にリバースプロキシの機能を持つ Web サーバを配置することをお勧めします。詳細は，「3.13　DMZ へのリバースプロキシの配置を検討する」を参照してください。

（1）システム構成の特徴

Web フロントシステムで，Web ブラウザから送信されたリクエストをアプリケーションサーバで処理する場合に適用されるシステム構成です。

最も基本的な Web クライアント構成は，クライアントマシンと 1 台のアプリケーションサーバマシンによって構築できます。1 台のアプリケーションサーバマシンで構築する Web クライアント構成の例を次の図に示します。

図 3-28　1 台のアプリケーションサーバマシンで構築する Web クライアント構成の例（Web サーバ連携の場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

クライアントで使用するソフトウェアは Web ブラウザだけです。

リクエストの流れ

アクセスポイントであるサーブレットと JSP は，J2EE サーバ上で動作します。これらのコンポーネントは，Web ブラウザから Web サーバ経由でアクセスされます。

(2) それぞれのマシンに必要なプロセスとソフトウェア

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。なお，リソースと接続するために必要なプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

必要なプロセスは次のとおりです。

- Web サーバ
- J2EE サーバ

● 運用管理エージェント

● PRF デーモン

なお，Application Server Standard または Application Server Enterprise には，Web サーバである Hitachi Web Server が含まれています。Windows の場合，Web サーバに Microsoft IIS を使用することもできます。この場合は，ソフトウェアとして Microsoft IIS が必要です。

(b) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

(c) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.4.2 サーブレットと JSP をアクセスポイントに使用する構成（インプロセス HTTP サーバを使用する場合）

サーブレットと JSP がアクセスポイントになる，Web ベースのシステム構成について説明します。ここでは，インプロセス HTTP サーバを使用する場合の構成について説明します。

なお，この構成でインターネットに接続する場合，セキュリティの観点から，DMZ にリバースプロキシの機能を持つ Web サーバを配置してください。詳細は，「3.13　DMZ へのリバースプロキシの配置を検討する」を参照してください。

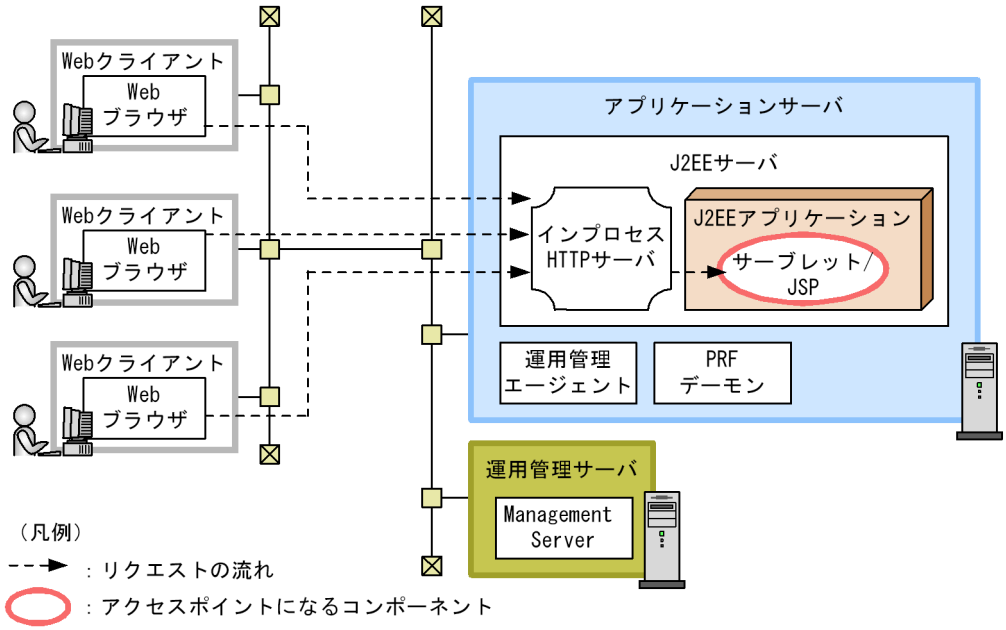
(1) システム構成の特徴

Web フロントシステムで，Web ブラウザから送信されたリクエストをアプリケーションサーバで処理する場合に適用されるシステム構成です。

最も基本的な Web クライアント構成は，クライアントマシンと 1 台のアプリケーションサーバマシンによって構築できます。インプロセス HTTP サーバを使用する場合，J2EE サーバのフロントに Web サーバを起動する必要がありません。J2EE サーバ上の HTTP サーバの機能を使用します。

インプロセス HTTP サーバを使用する場合の Web クライアント構成の例を次の図に示します。

図 3-29　1 台のアプリケーションサーバマシンで構築する Web クライアント構成の例（インプロセス HTTP サーバを使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

クライアントで使用するソフトウェアは Web ブラウザだけです。

Web ブラウザから Web サーバを経由しないで直接 J2EE サーバにアクセスできるため，性能上のメリットがあります。また，Web サーバを起動する必要がないため，運用が簡易になります。

なお，インプロセス HTTP サーバを使用する場合，次の点に留意してください。

- インプロセス HTTP サーバでは，Web サーバとして動作するための最小限の機能だけがサポートされています。このため，Web サーバのさまざまな機能を必要とする場合は，この構成は適していません。リダイレクタモジュールを組み込んだ Web サーバと連携する構成を選択してください。インプロセス HTTP サーバで使用できる機能については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「4.2.2　インプロセス HTTP サーバで使用できる機能」を参照してください。
- インプロセス HTTP サーバでは，HTTPS はサポートされていません。このため，SSL を使用する場合には，プロキシサーバをフロントに配置してプロキシモジュールを組み込んだ Web サーバの SSL 機能を使用するか，または，SSL アクセラレータをフロントに配置してください。
- インターネットに接続するシステムでは，セキュリティ上の観点から，必ずリバースプロキシをフロントに配置してください。リバースプロキシを配置する構

成については，「3.13　DMZ へのリバースプロキシの配置を検討する」を参照してください。

リクエストの流れ

アクセスポイントであるサーブレットと JSP は，J2EE サーバ上で動作します。これらのコンポーネントは，Web ブラウザから直接アクセスされます。

(2) それぞれのマシンに必要なプロセスとソフトウェア

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。なお，リソースと接続するために必要なプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

必要なプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

(c) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.4.3 Session Bean と Entity Bean をアクセスポイントに使用する構成

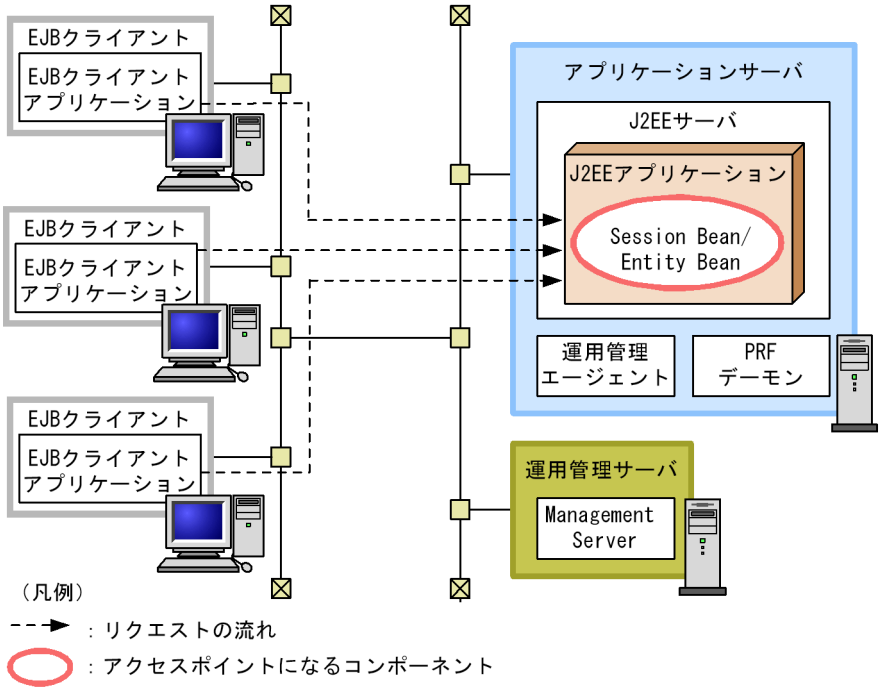
Session Bean と Entity Bean がアクセスポイントになり，クライアントで EJB クライアントアプリケーションを動作させるシステム構成について説明します。この構成を，**EJB クライアント構成**といいます。

(1) システム構成の特徴

最も基本的な EJB クライアント構成は，クライアントマシンと 1 台のアプリケーションサーバマシンによって構築できます。

1 台のアプリケーションサーバマシンで構築する EJB クライアント構成の例を次の図に示します。

図 3-30　1 台のアプリケーションサーバマシンで構築する EJB クライアント構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

Windows の場合は，uCosminexus Client を使用してクライアントマシンの環境を構築できます。

リクエストの流れ

アクセスポイントである Session Bean と Entity Bean は，J2EE サーバ上で動作します。

EJB クライアントアプリケーションからのリクエストは，RMI-IIOP によってアクセスポイントに送られ，Session Bean と Entity Bean を呼び出します。このとき，EJB クライアントアプリケーションは，J2EE サーバ内で動作しているネーミングサービスから名前を検索（ルックアップ）して，Session Bean と Entity Bean にアクセスします。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。なお，リソースに接続するためのプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

(c) EJB クライアントマシン

EJB クライアントマシンには，Application Server Standard，Application Server Enterprise または uCosminexus Client（Windows の場合）をインストールする必要があります。

起動するプロセスは，EJB クライアントアプリケーションのプロセスです。

3.4.4 CTM を使用する場合に Stateless Session Bean をアクセスポイントに使用する構成

CTM を使用する場合の，Stateless Session Bean をアクセスポイントとするシステム構成について説明します。

参考

ここでは，クライアントとして EJB クライアントアプリケーションを使用する EJB クライアント構成の例を示します。このほか，CTM ゲートウェイ機能を使用すると，EJB クライアント以外のクライアントアプリケーションから J2EE サーバ上の EJB アプリケーションを直接呼び出す構成も実現できます。これらの構成については，「3.15.3　CTM ゲートウェイ機能を利用して EJB クライアント以外から Stateless Session Bean を呼び出す構成」を参照してください。

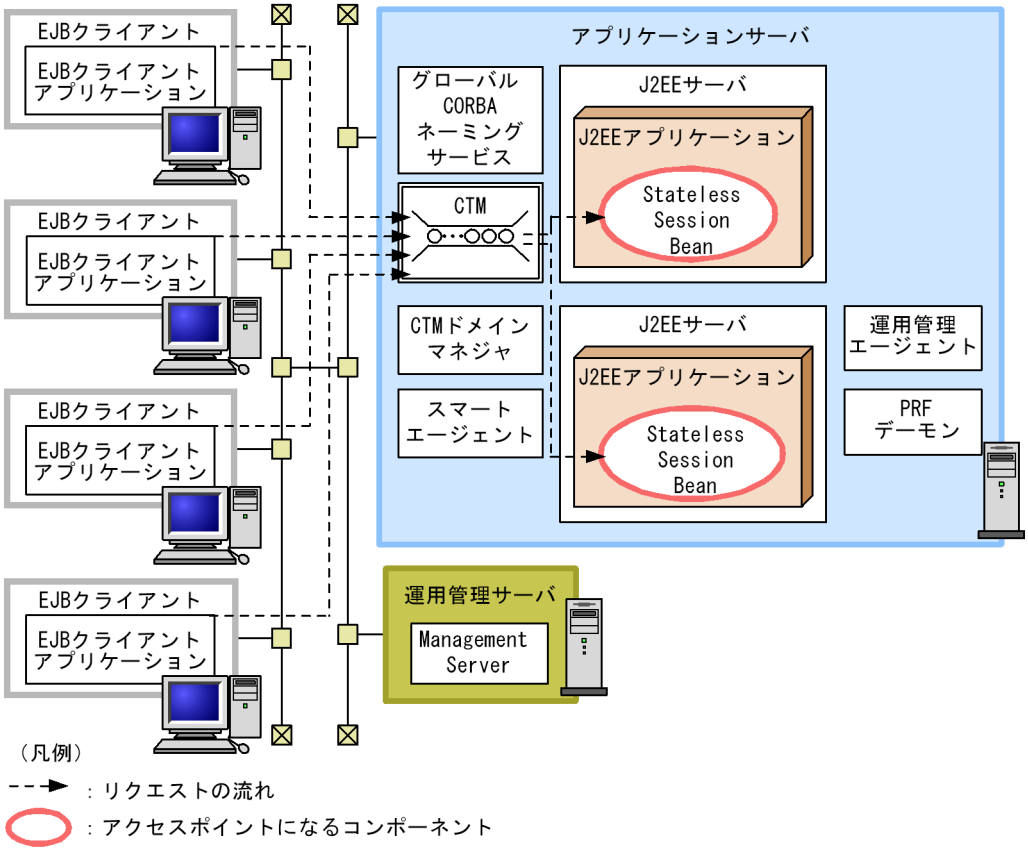
(1) システム構成の特徴

EJB クライアント構成の一つです。アクセスポイントである Stateless Session Bean に

対して，CTM によってスケジューリングされたリクエストが送信されます。

CTM を使用する場合の EJB クライアント構成の例を次の図に示します。

図 3-31　CTM を使用する場合の EJB クライアント構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- CTM によってリクエストをスケジューリングすることで，サービス閉塞を実行したり同時実行数を制御したりするシステム運用ができるようになります。
- J2EE サーバを二つ以上起動することで，一つの J2EE サーバにトラブルが発生した場合に，縮退運転をしてシステムの稼働を続けられます。
- Windows の場合は，uCosminexus Client を使用してクライアントマシンの環境を構築できます。

リクエストの流れ

アクセスポイントである Stateless Session Bean は，J2EE サーバ上で動作します。EJB クライアントアプリケーションからのリクエストは，CTM 経由で送られ，Stateless Session Bean を呼び出します。このとき，EJB クライアントアプリケー

ションは，グローバル CORBA ネーミングサービスから名前をルックアップして，Stateless Session Bean にアクセスします。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。なお，リソースに接続するためのプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン
- グローバル CORBA ネーミングサービス
- CTM のプロセス群（CTM デーモンおよび CTM レギュレータ）
- CTM ドメインマネジャ
- スマートエージェント

(b) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

(c) EJB クライアントマシン

EJB クライアントマシンには，Application Server Standard，Application Server Enterprise または uCosminexus Client（Windows の場合）をインストールする必要があります。

起動するプロセスは EJB クライアントアプリケーションのプロセスです。

3.5　サーバ間での連携を検討する

この節では，複数のアプリケーションサーバ上の J2EE サーバで動作するアプリケーションを連携させる構成の種類と，それぞれの場合に各マシンに配置するプロセスについて説明します。また，それぞれの構成の特徴についても説明します。

なお，複数のアプリケーションサーバで構成するシステムの場合，呼び出し元になるアプリケーションが動作しているアプリケーションサーバを**クライアント側のアプリケーションサーバ**，呼び出されるアプリケーションが動作しているアプリケーションサーバを**サーバ側のアプリケーションサーバ**といいます。

ここでは，次に示す 2 種類のサーバ間連携の構成について説明します。

● Session Bean と Entity Bean を呼び出すサーバ間連携

● CTM 経由で Stateless Session Bean を呼び出すサーバ間連携

3.5.1　Session Bean と Entity Bean を呼び出すサーバ間連携

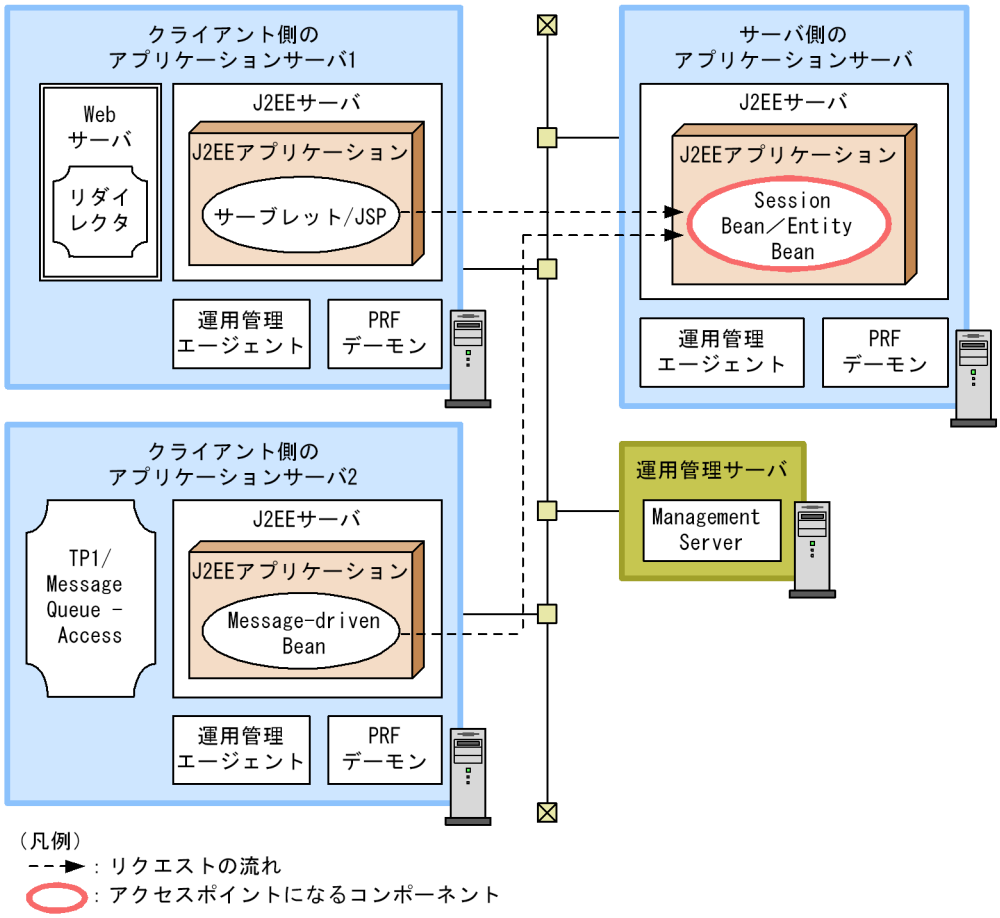
ほかのアプリケーションサーバ上の J2EE サーバから Session Bean または Entity Bean を呼び出す場合の構成について説明します。

(1) システム構成の特徴

クライアント側のアプリケーションサーバから，サーバ側のアプリケーションサーバを呼び出す構成です。クライアント側のアプリケーションサーバでは，サーブレット，JSP，Entity Bean，Session Bean または Message-driven Bean で構成されるアプリケーションが動作します。サーバ側のアクセスポイントになるコンポーネントは，Session Bean または Entity Bean です。クライアント側のアプリケーションからの呼び出しは，RMI-IIOP で実行されます。

Session Bean と Entity Bean を呼び出すサーバ間連携の構成の例を，次の図に示します。

図 3-32　Session Bean と Entity Bean を呼び出すサーバ間連携の構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーション間連携で使用できる構成です。複数のクライアントアプリケーションから，サーバアプリケーションを呼び出す場合などに適用できます。
- システム間連携の場合にも使用できます。すでに構築されたクライアント側システムとサーバ側システムをこのシステム構成にすることで，アプリケーションをシステム間で呼び出せます。

リクエストの流れ

サーバ側のアクセスポイントである Session Bean または Entity Bean へのリクエストは，クライアント側の J2EE サーバから送られます。

このとき，クライアント側では，サーバ側のアプリケーションサーバの J2EE サーバ内でインプロセスで起動されている CORBA ネーミングサービスから名前をルックアップして，Session Bean または Entity Bean にアクセスします。

（2）それぞれのマシンに必要なソフトウェアと起動するプロセス

Session Bean または Entity Bean を呼び出すサーバ間連携の場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。なお，リソースと接続するために必要なプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

（a）クライアント側のアプリケーションサーバマシン（サーブレット／ JSP の実行環境）

サーブレットおよび JSP を動作させるクライアント側のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Web サーバ
- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

（b）クライアント側のアプリケーションサーバマシン（Message-driven Bean の実行環境）

Message-driven Bean を動作させるクライアント側のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。また，リソースアダプタとして TP1/Message Queue - Access を使用する場合は，TP1/Message Queue - Access をインストールする必要があります。なお，Cosminexus RM を使用する場合は，アプリケーションサーバに含まれているため，個別にインストールする必要はありません。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

（c）サーバ側のアプリケーションサーバマシン

サーバ側のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

（d）運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

3.5.2 CTM 経由で Stateless Session Bean を呼び出すサーバ間連携

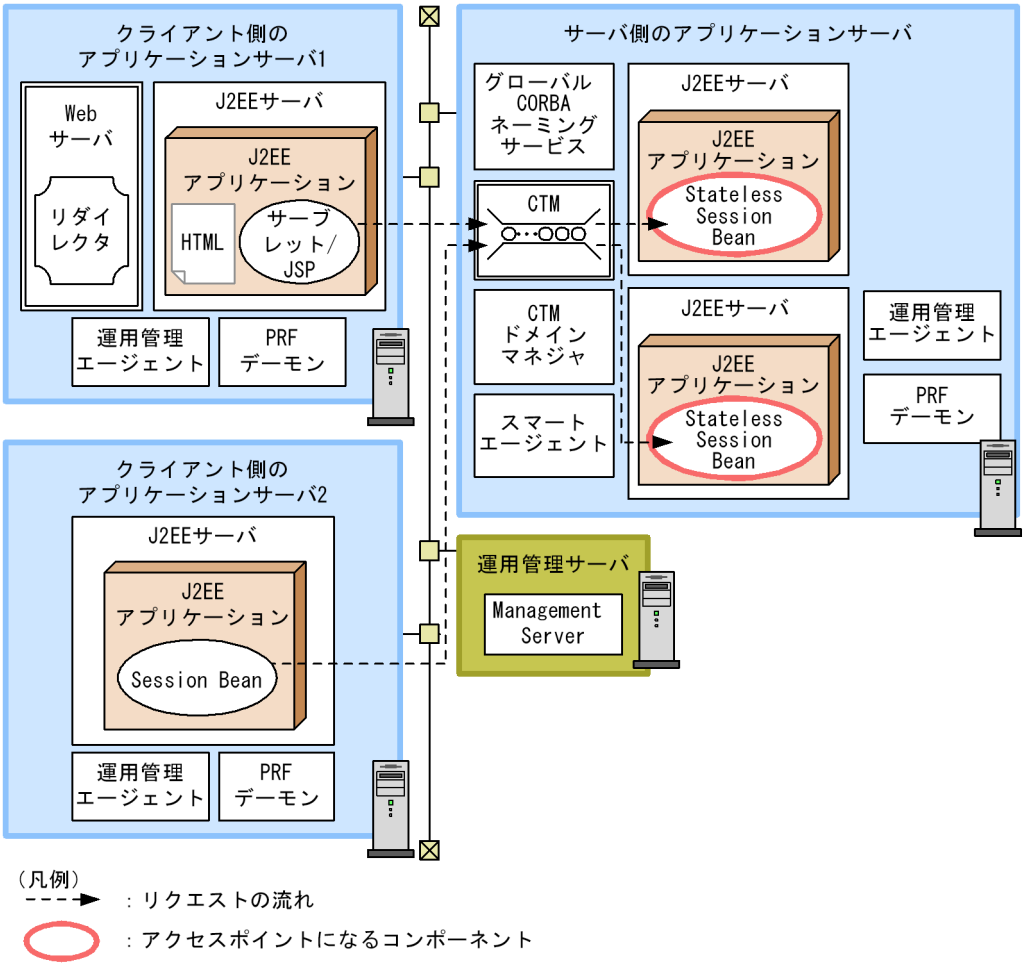
CTM を使用する場合に，ほかのアプリケーションサーバ上の J2EE サーバから CTM 経由で Stateless Session Bean を呼び出すときの構成について説明します。

（1）システム構成の特徴

クライアント側のアプリケーションサーバから，CTM 経由でサーバ側のアプリケーションサーバを呼び出す構成です。クライアント側のアプリケーションサーバでは，サーブレット，JSP，Entity Bean，Session Bean または Message-driven Bean で構成されるアプリケーションが動作します。サーバ側のアクセスポイントになるコンポーネントは，Stateless Session Bean です。クライアント側のアプリケーションからの呼び出しは，RMI-IIOP で実行されます。

CTM 経由で Stateless Session Bean を呼び出すサーバ間連携の構成の例を次の図に示します。

図 3-33　Stateless Session Bean を呼び出すサーバ間連携の構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーション間連携で使用できる構成です。複数のクライアントアプリケーションから，サーバアプリケーションを呼び出す場合などに適用できます。
- システム間連携の場合にも使用できます。すでに構築されたクライアント側システムとサーバ側システムをこのシステム構成にすることで，アプリケーションをシステム間で呼び出せます。

リクエストの流れ

サーバ側のアクセスポイントである Stateless Session Bean へのリクエストは，クライアント側の J2EE サーバから CTM 経由で送られます。

このとき，クライアント側では，サーバ側のアプリケーションサーバのグローバル CORBA ネーミングサービスから名前をルックアップして，CTM 経由で Stateless

Session Bean にアクセスします。
CTM を経由したリクエストは，CTM によって J2EE サーバ上で動作する Stateless Session Bean に適切に振り分けられます。

（2） それぞれのマシンに必要なソフトウェアと起動するプロセス

CTM 経由で Stateless Session Bean を呼び出すサーバ間連携の場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。なお，リソースと接続するために必要なプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

（a） クライアント側のアプリケーションサーバマシン（サーブレット／ JSP の実行環境）

サーブレットおよび JSP を動作させるクライアント側のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Web サーバ
- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

（b） クライアント側のアプリケーションサーバマシン（Session Bean の実行環境）

Session Bean を動作させるクライアント側のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

（c） サーバ側のアプリケーションサーバマシン

サーバ側のアプリケーションサーバマシンには，Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

● グローバル CORBA ネーミングサービス

● CTM のプロセス群（CTM デーモンおよび CTM レギュレータ）

● CTM ドメインマネジャ

● スマートエージェント

（d）運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

3.6 トランザクションの種類を検討する

この節では，トランザクションの種類ごとのシステム構成について説明します。

トランザクションに参加するリソースが一つの場合は，ローカルトランザクションを使用します。トランザクションに参加するリソースが複数の場合は，グローバルトランザクションを使用します。また，サーバ間で連携する構成の場合に，それぞれの J2EE サーバが異なるリソースに接続するときには，トランザクションコンテキストのプロパゲーションも使用できます。

なお，接続するリソースとリソースアダプタの対応については，「3.3.2　リソースの種類とリソースアダプタ」を参照してください。

参考

この節では，アプリケーションサーバによって開始されるトランザクションの種類ごとの構成について説明しますが，EJB クライアントアプリケーションでもトランザクションを開始できます。なお，EJB クライアントアプリケーションでトランザクションを開始する場合は，EJB クライアントマシンに Application Server Standard または Application Server Enterprise が必要です。

3.6.1 ローカルトランザクションを使用する場合の構成

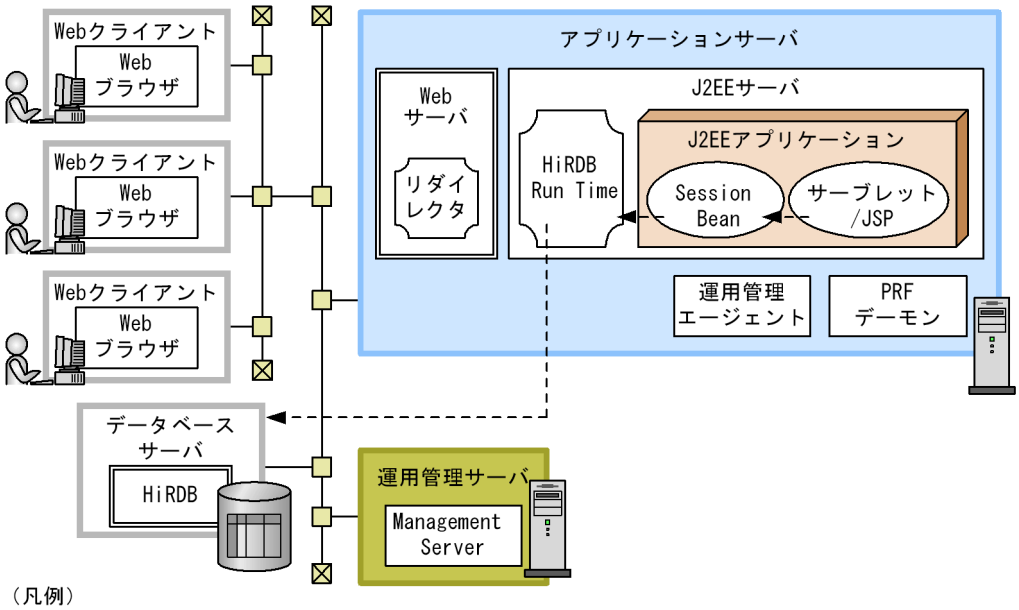
トランザクションに参加するリソースが一つの場合の構成について説明します。

（1）システム構成の特徴

サーブレット，JSP および Session Bean で構成される一つの J2EE アプリケーションから，リソースアダプタ経由で一つのリソースマネジャにアクセスする場合の構成です。アプリケーションからリソースマネジャへのアクセスで使用するトランザクションは，リソースマネジャ側で管理されます。トランザクションの種類は XA インタフェースを使用しない，ローカルトランザクションになります。

ローカルトランザクションを使用する場合の構成の例を，次の図に示します。

図 3-34　ローカルトランザクションを使用する場合の構成の例



（凡例）

--► ：アプリケーションからリソースアダプタ経由でリソースマネジャにアクセスする流れ

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

アプリケーションから一つのリソースマネジャにアクセスします。2 フェーズコミットは必要ありません。

アプリケーションからリソースアダプタ経由でリソースマネジャにアクセスする流れ

Web ブラウザから Web サーバ経由でアクセスされたサーブレットおよび JSP は，同じアプリケーション内の Session Bean をローカルで呼び出します。Session Bean は，リソースアダプタを経由してリソースマネジャ（図 3-37 の場合は HiRDB）にアクセスします。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

ローカルトランザクションを使用する場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

また，リソースマネジャと接続するために，次のソフトウェアが必要です。

接続するリソースマネジャ	必要なソフトウェア
HiRDB	HiRDB Run Time または HiRDB Type4 JDBC Driver
Oracle	Oracle Client または Oracle JDBC Thin Driver
SQL Server	SQL Server の JDBC ドライバ
XDM/RD E2	HiRDB Run Time または HiRDB Type4 JDBC Driver
TP1/Message Queue	TP1/Message Queue - Access
OpenTP1 の SPP	uCosminexus TP1 Connector TP1/Client/J

起動するプロセスは次のとおりです。

- Web サーバ
- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) リソースマネジャが動作するマシン

リソースマネジャが動作するマシンには，次に示すソフトウェアのどれかをインストールしてください。

- HiRDB（HiRDB と接続する場合）
- Oracle（Oracle と接続する場合）
- SQL Server（SQL Server と接続する場合）
- XDM/RD E2（XDM/RD E2 と接続する場合）
- TP1/Message Queue（TP1/Message Queue と接続する場合）
- TP1/Server Base（OpenTP1 の SPP と接続する場合）

また，それぞれのリソースマネジャで必要なプロセスを起動してください。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

(d) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.6.2　グローバルトランザクションを使用する場合の構成

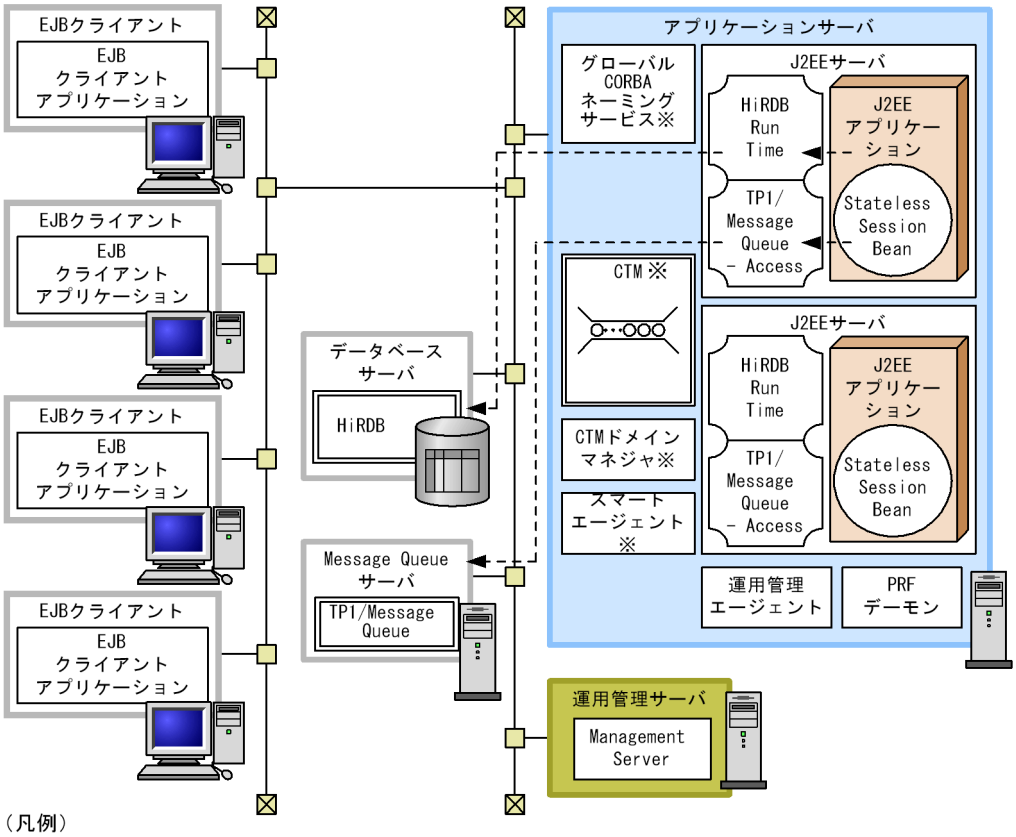
トランザクションに参加するリソースが複数の場合の構成について説明します。

（1）システム構成の特徴

Session Bean を構成要素とする一つの J2EE アプリケーションから，リソースアダプタ経由で複数のリソースマネジャにアクセスする構成です。アプリケーションからリソースマネジャへのアクセスで使用するトランザクションは，J2EE サーバ側で管理します。トランザクションの種類は，XA インタフェースを使用するグローバルトランザクションになります。

グローバルトランザクションを使用する場合の構成の例を，次の図に示します。この例は，CTM を使用するシステムの場合の例です。

図 3-35　グローバルトランザクションを使用する場合の構成の例



（凡例）

--►：アプリケーションからリソースアダプタ経由でリソースマネジャにアクセスする流れ

注※　CTMを使用する場合に必要となります。

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションから複数のリソースマネジャにアクセスします。2 フェーズコミットが必要になります。
- 一つのリソースマネジャにアクセスするアプリケーションと，複数のリソースマネジャにアクセスするアプリケーションを混在させるシステムの場合も，この構成にする必要があります。

アプリケーションからリソースアダプタ経由でリソースマネジャにアクセスする流れ

EJB クライアントアプリケーションからアクセスされた Stateless Session Bean は，リソースアダプタを経由してリソースマネジャにアクセスします。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

グローバルトランザクションを使用する場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

また，リソースマネジャと接続するために，次のソフトウェアをインストールする必要があります。

接続するリソースマネジャ	必要なソフトウェア
HiRDB	HiRDB Run Time または HiRDB Type4 JDBC Driver
Oracle	Oracle Client または Oracle JDBC Thin Driver
TP1/Message Queue	TP1/Message Queue - Access
OpenTP1 の SPP	uCosminexus TP1 Connector TP1/Client/J

注　SQL Server および XDM/RD E2 はグローバルトランザクションでは使用できません。

起動するプロセスは次のとおりです。

● J2EE サーバ

● 運用管理エージェント

● PRF デーモン

また，図 3-38 のように CTM を使用する構成の場合は，Application Server Enterprise をインストールして，上記のプロセスのほかに，グローバル CORBA ネーミングサービス，CTM のプロセス群，CTM ドメインマネジャおよびスマートエージェントも起動する必要があります。詳細は，「3.5.2　CTM 経由で Stateless Session Bean を呼び出すサーバ間連携」を参照してください。

(b) リソースマネジャが動作するマシン

リソースマネジャが動作するマシンには，次に示すソフトウェアのどれかをインストールしてください。なお，SQL Server および XDM/RD E2 はグローバルトランザクションでは使用できません。

- HiRDB（HiRDB と接続する場合）
- Oracle（Oracle と接続する場合）
- TP1/Message Queue（TP1/Message Queue と接続する場合）
- TP1/Server Base（OpenTP1 の SPP と接続する場合）

また，それぞれのリソースマネジャで必要なプロセスを起動してください。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

(d) EJB クライアントマシン

EJB クライアントマシンには，Application Server Standard，Application Server Enterprise または uCosminexus Client（Windows の場合）をインストールする必要があります。

起動するプロセスは，EJB クライアントアプリケーションのプロセスです。

3.6.3 トランザクションコンテキストのプロパゲーションを使用する場合の構成

サーバ間で連携する構成の場合に，それぞれの J2EE サーバが異なるリソースに接続する構成について説明します。この構成では，トランザクションコンテキストのプロパゲーションを使用します。なお，この構成の場合，CTM は使用できません。

(1) システム構成の特徴

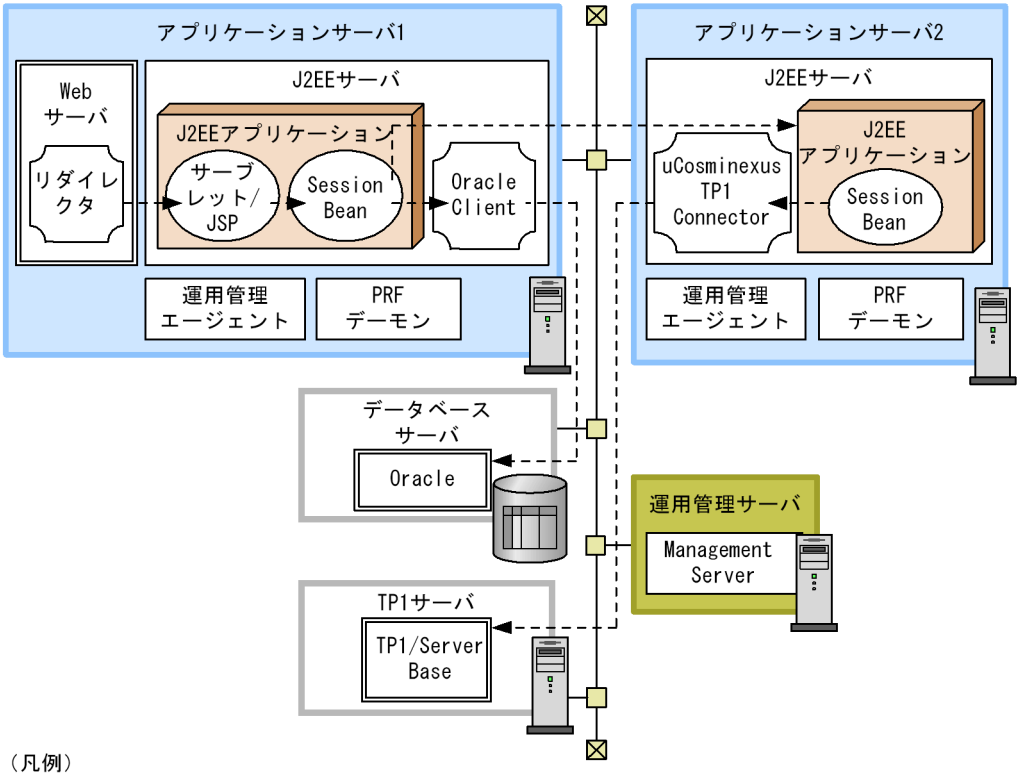
複数の J2EE アプリケーションでサーバ間連携をする構成です。ここでは，サーブレット，JSP，および Session Bean で構成される J2EE アプリケーションと，Session Bean だけで構成される J2EE アプリケーションでサーバ間連携をする例について説明します。サーブレット，JSP および Session Bean で構成されるアプリケーションは，クライアント側のアプリケーションサーバで動作し，Session Bean で構成される J2EE アプリケーションは，サーバ側のアプリケーションサーバで動作します。

それぞれの J2EE アプリケーションは，リソースアダプタ経由で異なるリソースマネジャにアクセスします。この場合，リソースマネジャへのアクセスで使用するトランザクションは，J2EE サーバ側で管理します。このときのトランザクションの種類は，XA インタフェースを使用するグローバルトランザクションになります。

なお，CTM 経由で Stateless Session Bean を呼び出す構成では，トランザクションコンテキストのプロパゲーションは使用できません。

トランザクションコンテキストのプロパゲーションを使用する場合の構成の例を次の図に示します。この例では，J2EE アプリケーションは，それぞれ，データベースと OpenTP1 の SPP にアクセスします。

図 3-36　トランザクションコンテキストのプロパゲーションを使用する場合の構成の例



（凡例）

－－►：アプリケーションからリソースアダプタ経由でリソースマネジャにアクセスする流れ

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- サーバ間連携をするアプリケーションから複数のリソースマネジャにアクセスする場合に適用できます。
- サーバ間連携をする場合に，一つのリソースマネジャにアクセスするアプリケーションと，複数のリソースマネジャにアクセスするアプリケーションを混在させ

るシステムの場合は，この構成にする必要があります。

アプリケーションからリソースアダプタ経由でリソースマネジャにアクセスする流れ

Web ブラウザから Web サーバ経由でアクセスされたサーブレットおよび JSP は，同じアプリケーション内の Session Bean をローカルで呼び出します。Session Bean は，トランザクションを開始して，データベースにアクセスしてから，サーバ側のアプリケーションサーバ上の J2EE サーバで動作している Session Bean を呼び出します。呼び出されたサーバ側のアプリケーションサーバ上の Session Bean は，OpenTP1 の SPP にアクセスします。クライアント側のアプリケーションサーバ上の Session Bean は，サーバ側の Session Bean から制御が戻ると，トランザクションをコミットします。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

トランザクションコンテキストのプロパゲーションを使用するときに必要なソフトウェアと起動するプロセスは，「3.6.2　グローバルトランザクションを使用する場合の構成」と同様です。ただし，コンテキストのトランザクションプロパゲーションを使用する場合，CTM は使用できません。

3.7　ロードバランスクラスタによる負荷分散方式を検討する

この節では，ロードバランスクラスタによって負荷を分散するための構成について説明します。

負荷分散は次の方法で実現します。アクセスポイントになるコンポーネントの種類によって，実現できるものが異なります。

● 負荷分散機を利用する（サーブレット／ JSP の場合）

● ネーミングサービスを利用する（Session Bean ／ Entity Bean の場合）

● CTM を利用する（Stateless Session Bean の場合）

負荷分散機を利用する構成については，Web サーバ連携の場合とインプロセス HTTP サーバを利用する場合に分けて説明します。なお，アクセスポイントが Message-driven Bean の場合の負荷分散については，「3.8.4　Message-driven Bean のインスタンスプールを利用した負荷分散（TP1/Message Queue を使用する場合）」を参照してください。

3.7.1　Web サーバ連携時の負荷分散機を利用した負荷分散（サーブレット／ JSP の場合）

負荷分散機（**レイヤ 5 スイッチ**）によって負荷を分散する構成について説明します。この方法は，サーブレットまたは JSP がアクセスポイントの場合に使用できます。

ここでは，Web サーバと連携する場合について説明します。

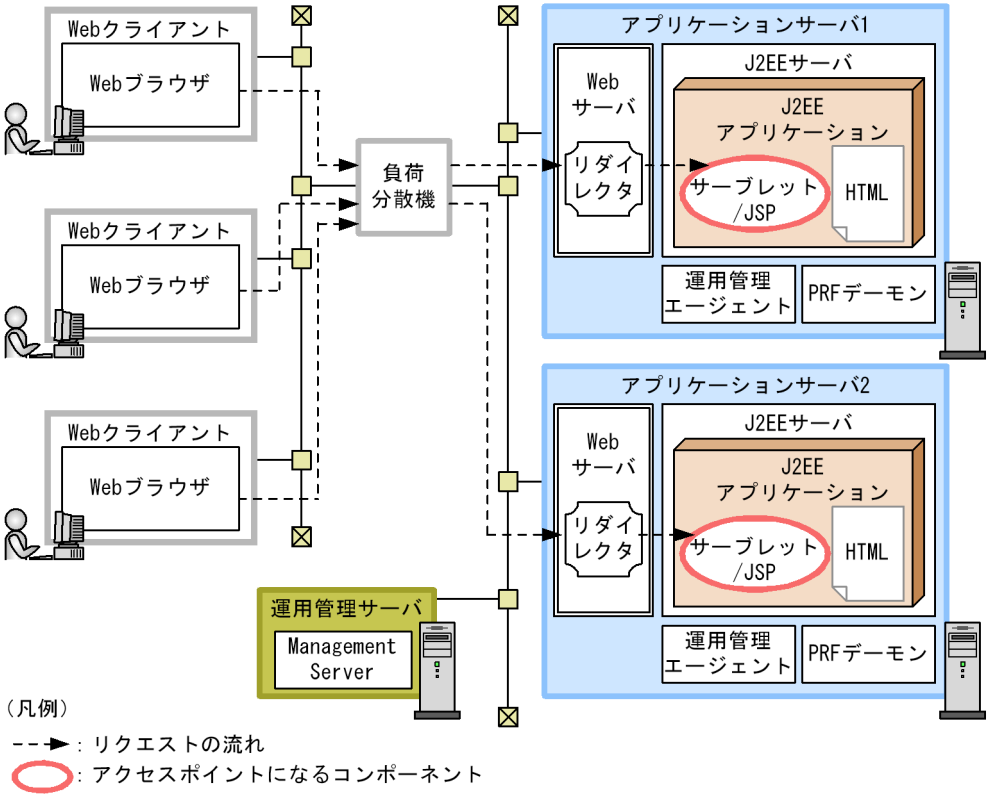
（1）システム構成の特徴

J2EE サーバ上で動作するアプリケーションのアクセスポイントが，サーブレットまたは JSP の場合に使用できる構成です。

負荷分散は，負荷分散機に対象となるアプリケーションサーバを登録することで実現できます。アプリケーションサーバを複数登録することで，クライアントからのアクセスによる負荷を分散できます。

負荷分散機を利用した負荷分散の構成の例を次の図に示します。

図 3-37　負荷分散機を利用した負荷分散の構成の例（Web サーバと連携する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- サーブレットと JSP のスケーラビリティと可用性を確保できます。
- 特定のアプリケーションサーバでトラブルが発生した場合，またはメンテナンスが必要な場合に，負荷分散機から該当するアプリケーションサーバへのアクセスを停止することで，システムの縮退運転ができます。

リクエストの流れ

アクセスポイントである J2EE サーバ上のサーブレットと JSP へのリクエストは，Web ブラウザから，負荷分散機および Web サーバ経由で送られます。その際，負荷分散機では，Web ブラウザからのアクセスを，それぞれのアプリケーションサーバ上で動作している Web サーバに対して振り分けます。なお，負荷分散機では，HTTP セッションのスティッキー（Sticky）やアフィニティ（Afinity）の関連づけも制御します。

参考

- HTTPS を使用する場合に，負荷分散機でのリクエストの振り分けにリクエストの中身（ヘッダなど）を使用するときは，負荷分散機のフロントに SSL アクセラレータを配置する必要があります。
- Smart Composer 機能を使用する場合は，負荷分散機を冗長化構成で配置することもできます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

負荷分散機を使用する場合にそれぞれのマシンに必要なソフトウェアと起動するプロセスは，サーブレットと JSP をアクセスポイントにする構成と同じです。「3.4.1　サーブレットと JSP をアクセスポイントに使用する構成（Web サーバ連携の場合）」を参照してください。

3.7.2 インプロセス HTTP サーバ使用時の負荷分散機を利用した負荷分散（サーブレット／ JSP の場合）

負荷分散機（**レイヤ 5 スイッチ**）によって負荷を分散する構成について説明します。この方法は，サーブレットまたは JSP がアクセスポイントの場合に使用できます。

ここでは，インプロセス HTTP サーバ機能を使用する場合について説明します。

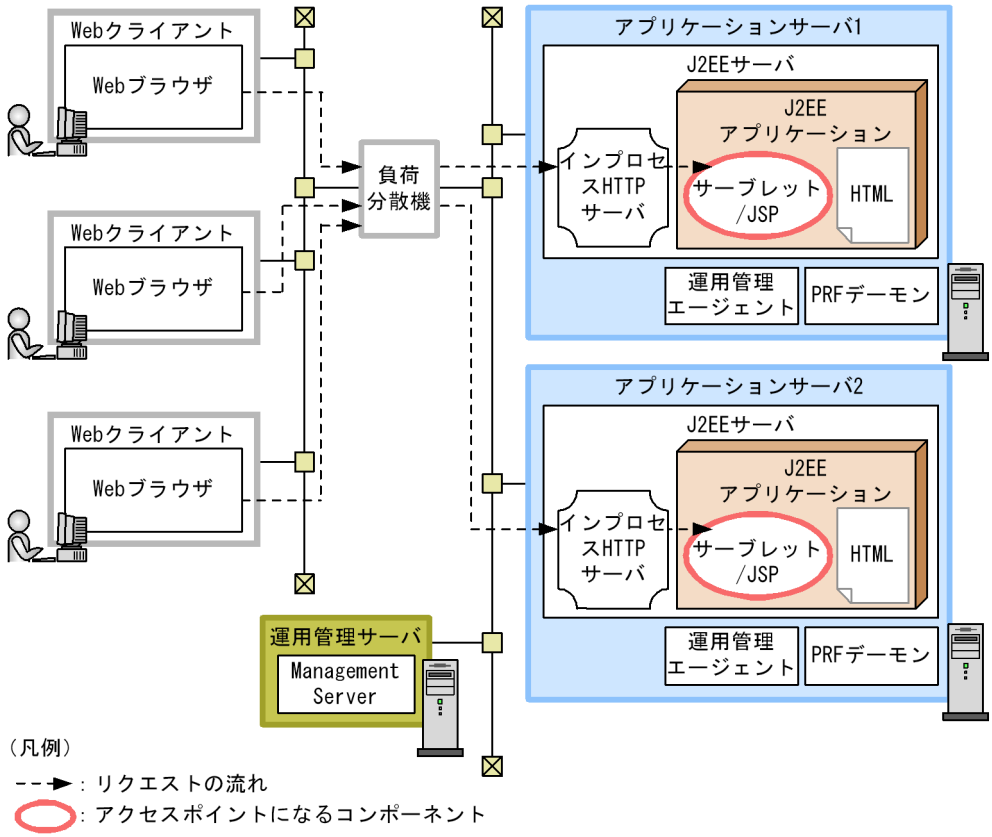
(1) システム構成の特徴

J2EE サーバ上で動作するアプリケーションのアクセスポイントが，サーブレットまたは JSP の場合に使用できる構成です。

負荷分散は，負荷分散機に対象になるアプリケーションサーバを登録することで実現できます。アプリケーションサーバを複数登録することで，クライアントからのアクセスによる負荷を分散できます。

負荷分散機を利用した負荷分散の構成の例を次の図に示します。

図 3-38　負荷分散機を利用した負荷分散の構成の例（インプロセス HTTP サーバを使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- サーブレットと JSP のスケーラビリティと可用性を確保できます。
- 特定のアプリケーションサーバでトラブルが発生した場合，またはメンテナンスが必要な場合に，負荷分散機から該当するアプリケーションサーバへのアクセスを停止することで，システムの縮退運転ができます。
- Web ブラウザから Web サーバを経由しないで直接 J2EE サーバにアクセスできるため，性能上のメリットがあります。また，Web サーバを起動する必要がないため，運用が簡易になります。ただし，使用できる機能や構成について，留意する必要があります。また，インターネットに接続する場合は，必ずリバースプロキシを組み込んだ Web サーバをフロントに配置してください。詳細は，「3.4.2 サーブレットと JSP をアクセスポイントに使用する構成（インプロセス HTTP サーバを使用する場合）」を参照してください。

リクエストの流れ

アクセスポイントである J2EE サーバ上のサーブレットと JSP へのリクエストは，

Web ブラウザから，負荷分散機経由で送られます。その際，負荷分散機では，Web ブラウザからのアクセスを，それぞれのアプリケーションサーバ上で動作している J2EE サーバに対して振り分けます。なお，負荷分散機では，HTTP セッションのスティッキー（Sticky）やアフィニティ（Afinity）の関連づけも制御します。

参考

HTTPS を使用する場合は，負荷分散機のフロントに SSL アクセラレータを配置する必要があります。

（2） それぞれのマシンに必要なソフトウェアと起動するプロセス

負荷分散機を使用する場合にそれぞれのマシンに必要なソフトウェアと起動するプロセスは，サーブレットと JSP をアクセスポイントにする構成と同じです。「3.4.2　サーブレットと JSP をアクセスポイントに使用する構成（インプロセス HTTP サーバを使用する場合）」を参照してください。

3.7.3　CORBA ネーミングサービスを利用した負荷分散（Session Bean と Entity Bean の場合）

アクセスポイントになるコンポーネントが Session Bean または Entity Bean の場合に，J2EE サーバ内（インプロセス）の CORBA ネーミングサービスのラウンドロビン検索機能を使用して負荷を分散する構成について説明します。

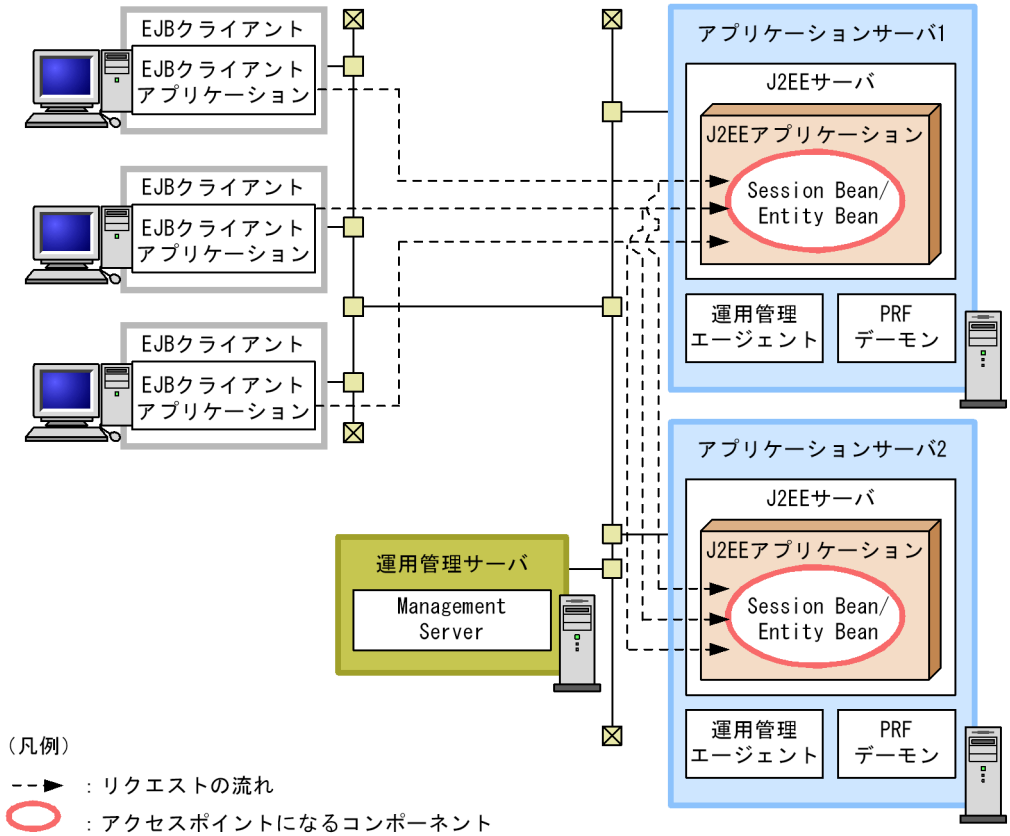
（1） システム構成の特徴

J2EE サーバ上で動作するアプリケーションのアクセスポイントが，Session Bean または Entity Bean の場合に使用できる構成です。クライアントは，EJB クライアントアプリケーションです。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に登録した論理ネーミングサービスから，オブジェクトリファレンスをラウンドロビン方式でルックアップすることで，リクエストの振り分け先を分散します。ただし，それぞれの J2EE サーバでは同じ名前（同じ別名）で Session Bean および Entity Bean を開始しておく必要があります。

なお，EJB クライアントアプリケーションは，ラウンドロビン方式で J2EE サーバ内のネーミングサービスをルックアップしたあと再度ネーミングサービスをルックアップするまでの間は，同じ J2EE サーバにアクセスします。

Session Bean と Entity Bean を対象にした負荷分散の構成の例を次の図に示します。

図 3-39　Session Bean と Entity Bean を対象にした負荷分散の構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- Session Bean と Entity Bean のスケーラビリティと可用性を確保できます。
- アプリケーションサーバを複数用意して，EJB クライアントからラウンドロビン方式で CORBA ネーミングサービスを選択してアクセスすることで，負荷を分散できます。
- 特定のアプリケーションサーバでトラブルが発生した場合，またはメンテナンスが必要な場合，EJB クライアントアプリケーションでは，J2EE サーバの終了を検出したあと，該当するアプリケーションサーバにアクセスしません。このため，システムの縮退運転ができます。

リクエストの流れ

アクセスポイントである J2EE サーバ上の Session Bean と Entity Bean へのリクエストは，EJB クライアントアプリケーションから送られます。その際，EJB クライアントアプリケーションでは，J2EE サーバ内のネーミングサービスをラウンドロビン方式で選択して，オブジェクトリファレンスをルックアップします。

（2） それぞれのマシンに必要なソフトウェアと起動するプロセス

CORBA ネーミングサービスを使用して負荷を分散する場合にそれぞれのマシンに必要なソフトウェアと起動するプロセスは，Session Bean と Entity Bean をアクセスポイントにする構成と同じです。「3.4.3　Session Bean と Entity Bean をアクセスポイントに使用する構成」を参照してください。

3.7.4　CTM を利用した負荷分散（Stateless Session Bean の場合）

アクセスポイントになるコンポーネントが Stateless Session Bean の場合に，CTM によって負荷を分散する構成です。

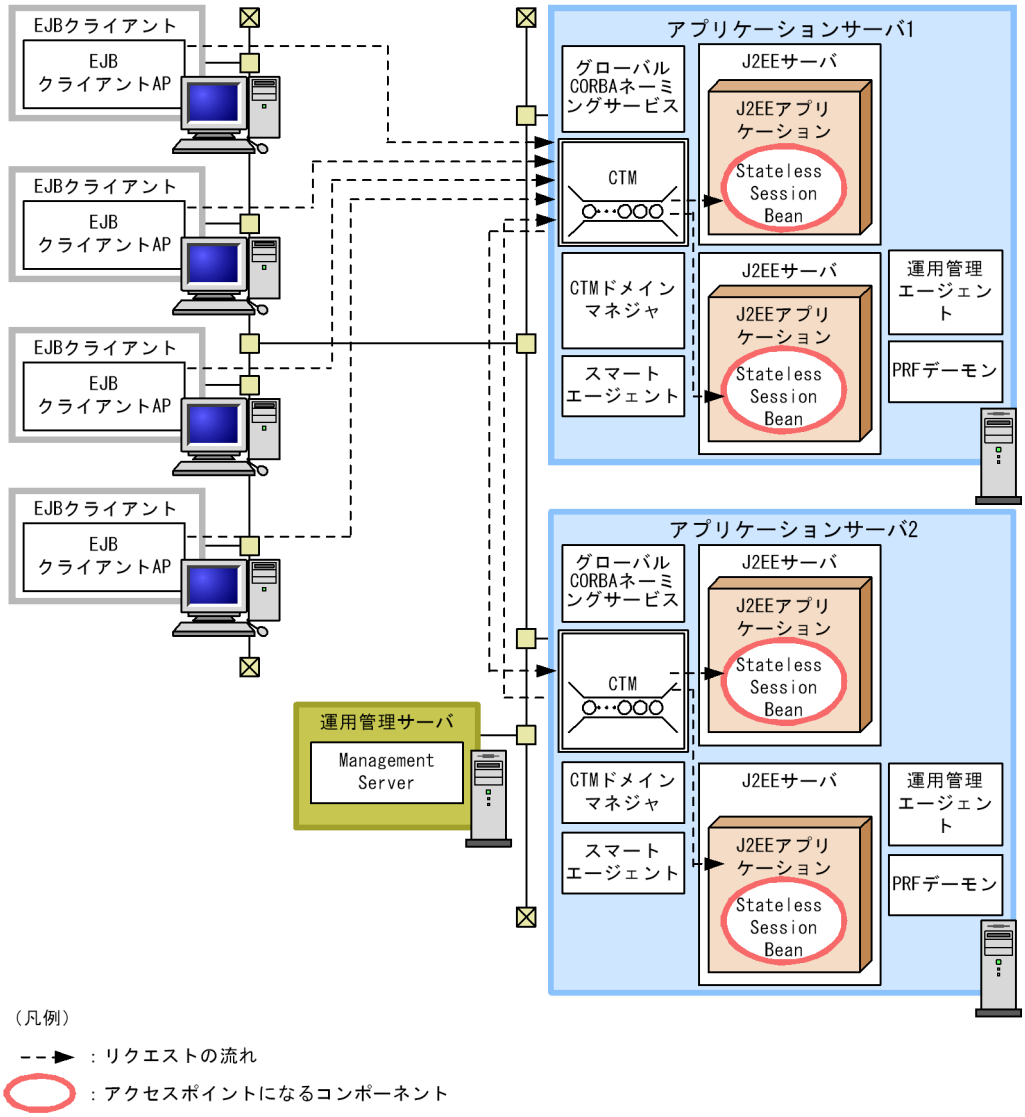
（1） システム構成の特徴

J2EE サーバ上で動作するアプリケーションのアクセスポイントが Stateless Session Bean の場合に，CTM を使用して実現する構成です。ここでは，クライアントが EJB クライアントアプリケーションの場合について説明します。

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に登録したグローバル CORBA ネーミングサービスからオブジェクトリファレンスをラウンドロビン方式でルックアップすることで，リクエストの振り分け先を分散します。ただし，それぞれの J2EE サーバでは同じ名前（同じ別名）で Stateless Session Bean を開始しておく必要があります。

Stateless Session Bean を対象にした CTM による負荷分散の構成の例を次の図に示します。

図 3-40　Stateless Session Bean を対象にした CTM による負荷分散の構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- Stateless Session Bean の負荷の均衡を実現して，高い可用性を確保できます。
- EJB クライアントからのラウンドロビン方式によるグローバル CORBA ネーミングサービスのルックアップと，CTM によるリクエストの振り分けによって，J2EE サーバ間の負荷均衡が実現できます。
- Stateless Session Bean のスケールアウトが容易に実現できるので，アプリケーションサーバの稼働率を向上できます。
- 特定の J2EE サーバにトラブルが発生した場合，CTM によってほかの J2EE サー

バにリクエストをスケジューリングすることで，システムの縮退運転ができます。また，EJB クライアントアプリケーションからは，トラブルを検知した J2EE サーバにはアクセスされません。

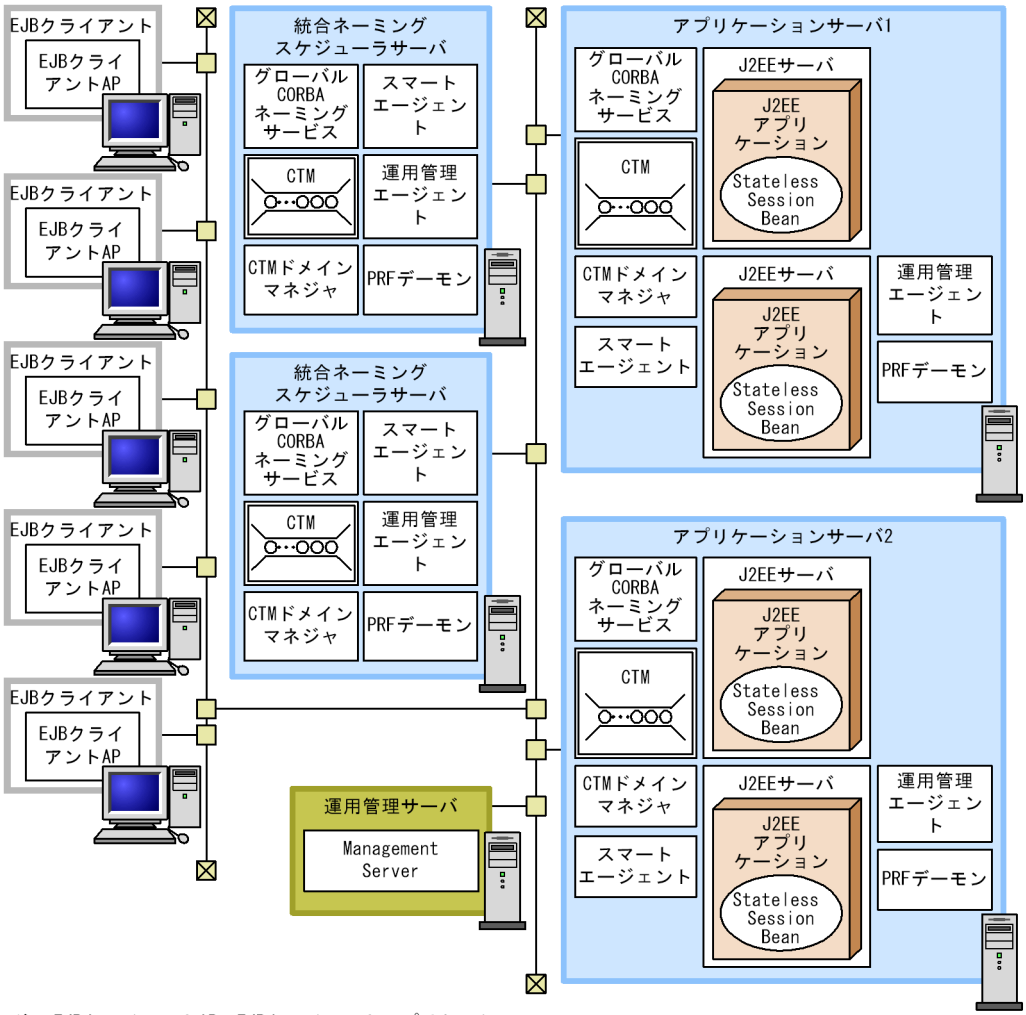
リクエストの流れ

アクセスポイントである J2EE サーバ上の Stateless Session Bean へのリクエストは，EJB クライアントから CTM 経由で送られます。その際，EJB クライアントは，グローバル CORBA ネーミングサービスから Stateless Session Bean の EJBHome オブジェクトリファレンスの名前をルックアップします。そのあと，リクエストは CTM によって，適切なアプリケーションサーバ上の J2EE サーバに振り分けられます。

CTM を使用して負荷を分散する場合，グローバル CORBA ネーミングサービスを独立したマシンに配置することもできます。このとき，グローバル CORBA ネーミングサービスを配置したマシンを，**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を配置したシステムの構成を次の図に示します。

図 3-41　Stateless Session Bean を対象にした CTM による負荷分散の構成の例（統合ネーミングスケジューラサーバを配置した場合）



注　EJBクライアントAP：EJBクライアントアプリケーション

凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- Stateless Session Bean の負荷の均衡を実現して，高い可用性を確保できます。
- 統合ネーミングスケジューラサーバのレプリカを作成して複数配置することで，ネーミングサービスの可用性を確保できます。
- EJB クライアントからのラウンドロビン方式によるグローバル CORBA ネーミングサービスのルックアップと，CTM によるリクエストの振り分けによって，J2EE サーバ間の負荷均衡が実現できます。
- Stateless Session Bean のスケールアウトが容易に実現できるので，アプリケー

ションサーバの稼働率を向上できます。スケールアウト時に，EJB クライアントで定義したグローバル CORBA ネーミングサービスのリストを変更する必要はありません。

- 特定の J2EE サーバにトラブルが発生した場合，CTM によってほかの J2EE サーバにリクエストをスケジューリングすることで，システムの縮退運転ができます。EJB クライアントアプリケーションからは，トラブルを検知した J2EE サーバにはアクセスされません。
- EJB クライアントは，統合ネーミングスケジューラサーバ上のグローバル CORBA ネーミングサービスから Stateless Session Bean の EJBHome オブジェクトリファレンスの名前をルックアップします。そのあと，アプリケーションサーバの CTM デーモンに処理が振り分けられます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

CTM を使用して負荷を分散する場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● J2EE サーバ

● 運用管理エージェント

● PRF デーモン

● グローバル CORBA ネーミングサービス

● CTM のプロセス群（CTM デーモンおよび CTM レギュレータ）

● CTM ドメインマネジャ

● スマートエージェント

(b) 統合ネーミングスケジューラサーバマシン

統合ネーミングスケジューラサーバを配置するシステム構成の場合，統合ネーミングスケジューラサーバマシンには，Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。J2EE サーバを起動する必要はありません。

● グローバル CORBA ネーミングサービス

● CTM のプロセス群（CTM デーモン）

● CTM ドメインマネジャ

● スマートエージェント

● 運用管理エージェント

● PRF デーモン

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

(d) EJB クライアントマシン

EJB クライアントマシンには，Application Server Standard，Application Server Enterprise，または uCosminexus Client（Windows の場合）をインストールする必要があります。

起動するプロセスは EJB クライアントアプリケーションのプロセスです。

3.8 サーバ間で非同期通信をする場合の構成を検討する

この節では，サーバ間で Message-driven Bean を使用して非同期通信をする場合のシステム構成について説明します。アクセスポイントは Message-driven Bean になります。

Message-driven Bean を使って非同期通信をするシステム構成としては，Cosminexus JMS プロバイダを使用するシステム構成，TP1/Message Queue を使用するシステム構成，および Cosminexus RM を使用するシステム構成があります。JMS プロバイダと Cosminexus RM は，アプリケーションサーバの構成ソフトウェアです。なお，Cosminexus RM を使用したシステムでは，Message-driven Bean のインスタンスプールを使用した負荷分散はできません。

3.8.1 Message-driven Bean をアクセスポイントに使用する場合の構成（Cosminexus JMS プロバイダを使用する場合）

ここでは，Cosminexus JMS プロバイダを使用して J2EE サーバ上の Message-driven Bean を呼び出す構成について説明します。

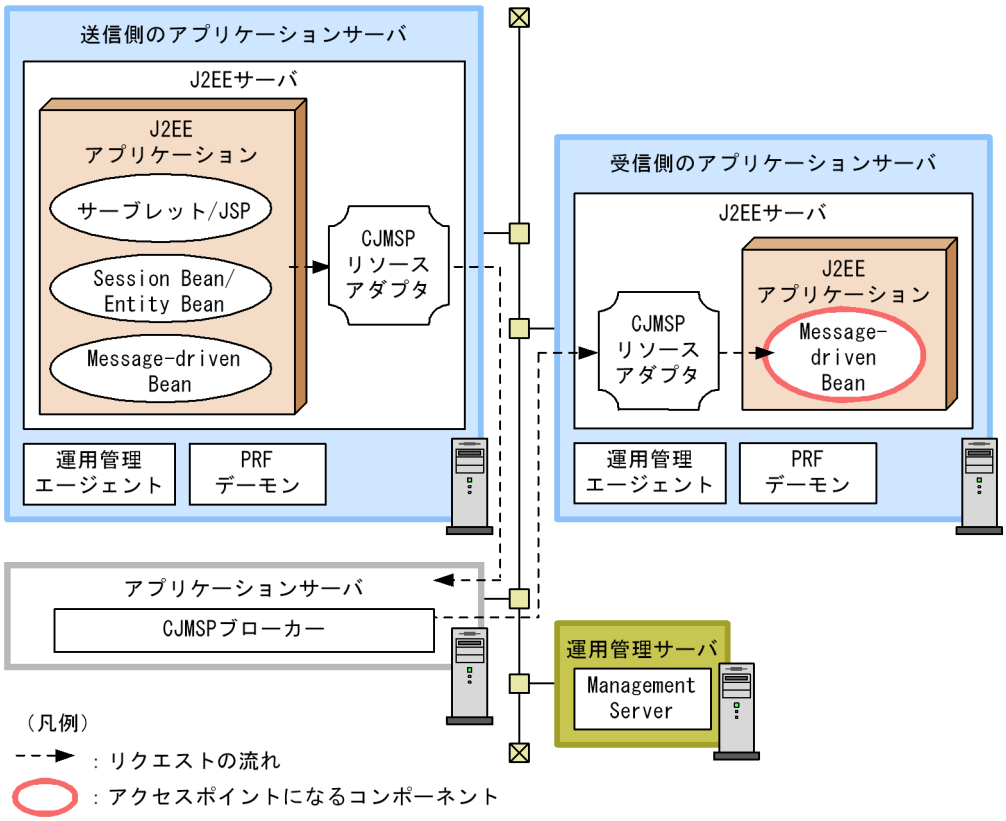
メッセージの送信側のアプリケーションサーバから，CJMSP ブローカーを経由して受信側のアプリケーションサーバを呼び出す構成です。送信側のアプリケーションサーバでは，サーブレット，JSP，Entity Bean，Session Bean または Message-driven Bean で構成されるアプリケーションが動作します。受信側のアクセスポイントになるコンポーネントは，Message-driven Bean です。

（1） システム構成の特徴

最も基本的なメッセージ駆動型のシステムの一つです。

Cosminexus JMS プロバイダを使用する場合のメッセージ駆動型のシステム構成の例を次の図に示します。

図 3-42　メッセージ駆動型のシステム構成の例（Cosminexus JMS プロバイダを使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

送信側のアプリケーションサーバでは，メッセージを送信するために JMS インタフェースを使用する J2EE クライアントアプリケーションと CJMSP リソースアダプタを使用します。

リクエストの流れ

アクセスポイントである Message-driven Bean は，受信側のアプリケーションサーバの J2EE サーバ上で動作します。リソースアダプタである CJMSP リソースアダプタのライブラリは，送信側のアプリケーションサーバの J2EE サーバ，および受信側のアプリケーションサーバの J2EE サーバ上で動作します。

送信側のアプリケーションサーバの J2EE アプリケーションからのリクエスト（メッセージ）は，CJMSP ブローカー経由で送られ，受信側のアプリケーションサーバ上の Message-driven Bean を呼び出します。

なお，送信側のアプリケーションサーバ，受信側のアプリケーションサーバおよび CJMSP ブローカーは，同じマシンに配置することもできます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン（送信側のアプリケーションサーバマシン）

アプリケーションサーバマシン（サーバ側のアプリケーションサーバマシン）には，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) アプリケーションサーバマシン（CJMSP ブローカーを配置するマシン）

CJMSP ブローカーを配置するマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動する必要があるプロセスは，CJMSP ブローカーです。なお，CJMSP ブローカーは，Management Server による運用管理の対象になりません。

(c) アプリケーションサーバマシン（受信側のアプリケーションサーバマシン）

クライアントマシン（クライアント側のアプリケーションサーバマシン）には，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(d) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

3.8.2　Message-driven Bean をアクセスポイントに使用する場合の構成（TP1/Message Queue を使用する場合）

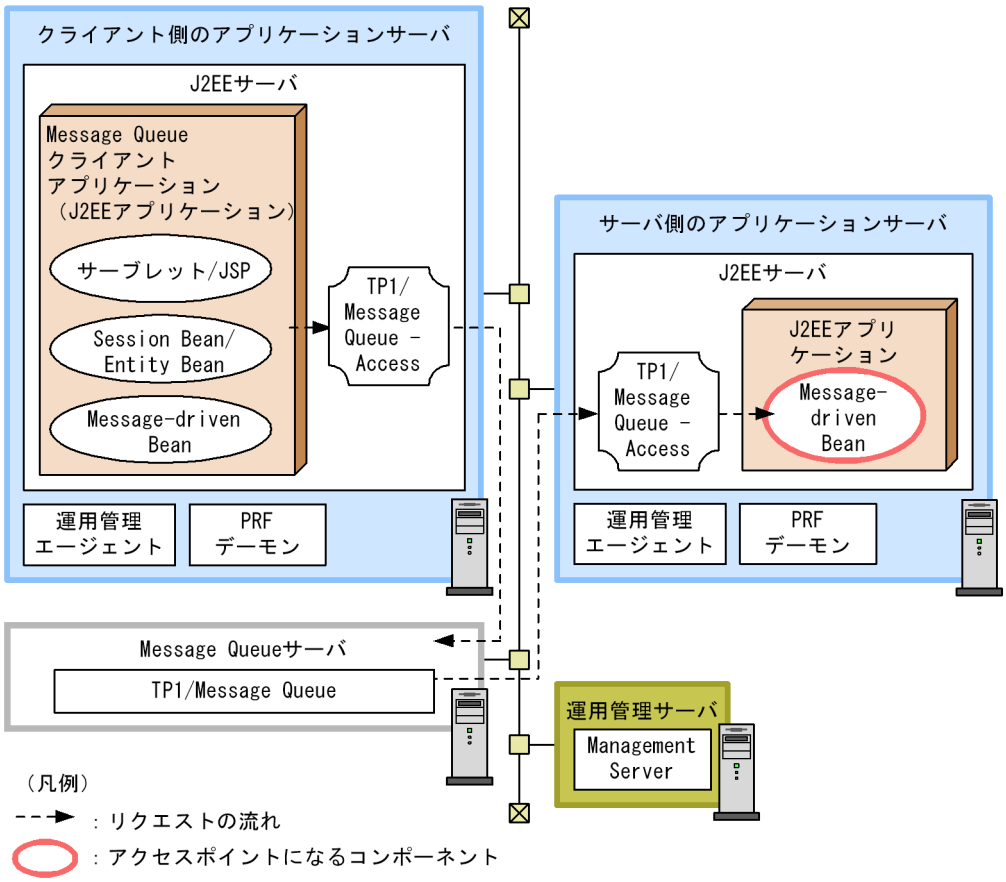
ここでは，クライアントとして，TP1/Message Queue が動作している Message Queue サーバを使用する場合の構成について説明します。

（1） システム構成の特徴

最も基本的なメッセージ駆動型のシステムの一つです。

TP1/Message Queue を使用する場合のメッセージ駆動型のシステム構成の例を次の図に示します。

図 3-43　メッセージ駆動型のシステム構成の例（TP1/Message Queue を使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

クライアントに，TP1/Message Queue にメッセージを送信する Message Queue ク

ライアントアプリケーションと TP1/Message Queue を使用します。なお，Message Queue クライアントアプリケーションとは，JMS インタフェースを使用する J2EE アプリケーションです。

リクエストの流れ

アクセスポイントである Message-driven Bean は，サーバ側のアプリケーションサーバの J2EE サーバ上で動作します。リソースアダプタである TP1/Message Queue - Access のライブラリは，サーバ側のアプリケーションサーバの J2EE サーバ，およびクライアント側のアプリケーションサーバの J2EE サーバ上で動作します。

クライアント側のアプリケーションサーバの J2EE アプリケーションからのリクエスト（メッセージ）は，TP1/Message Queue 経由で送られ，サーバ側のアプリケーションサーバ上の Message-driven Bean を呼び出します。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。なお，リソースに接続するためのプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) アプリケーションサーバマシン（サーバ側のアプリケーションサーバマシン）

アプリケーションサーバマシン（サーバ側のアプリケーションサーバマシン）には，Application Server Standard または Application Server Enterprise，および TP1/Message Queue - Access をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) Message Queue サーバマシン

Message Queue サーバマシンには，TP1/Message Queue をインストールする必要があります。

起動する必要があるプロセスは，TP1/Message Queue のプロセスです。

(c) アプリケーションサーバマシン（クライアント側のアプリケーションサーバマシン）

クライアントマシン（クライアント側のアプリケーションサーバマシン）には，Application Server Standard または Application Server Enterprise，および TP1/Message Queue - Access をインストールする必要があります。

起動するプロセスは次のとおりです。

● J2EE サーバ

● 運用管理エージェント

● PRF デーモン

（d）運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

3.8.3　Message-driven Bean をアクセスポイントに使用する場合の構成（Cosminexus RM を使用する場合）

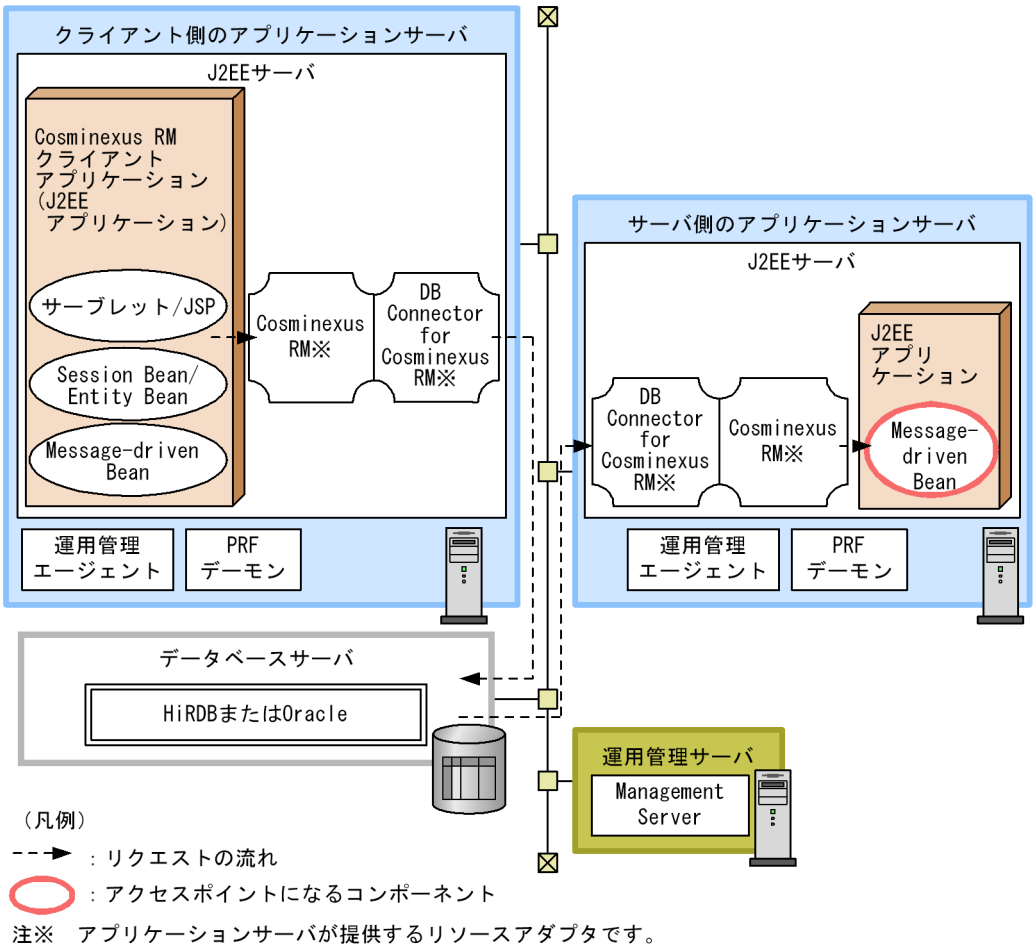
ここでは，クライアントとして，Cosminexus RM と連携しているデータベース（HiRDB または Oracle）を使用する場合の構成について説明します。

（1）システム構成の特徴

最も基本的なメッセージ駆動型のシステムの一つです。

Cosminexus RM を使用する場合のメッセージ駆動型のシステム構成の例を次の図に示します。

図 3-44　メッセージ駆動型のシステム構成の例（Cosminexus RM を使用する場合）



注※　アプリケーションサーバが提供するリソースアダプタです。

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

クライアントに，Cosminexus RM にメッセージを送信する Cosminexus RM のクライアントアプリケーションとデータベースサーバを使用します。なお，Cosminexus RM クライアントアプリケーションとは，JMS インタフェースを使用する J2EE アプリケーションです。

リクエストの流れ

アクセスポイントである Message-driven Bean は，サーバ側のアプリケーションサーバの J2EE サーバ上で動作します。リソースアダプタである Cosminexus RM のライブラリは，サーバ側のアプリケーションサーバの J2EE サーバ上，およびクライアント側のアプリケーションサーバの J2EE サーバ上で動作します。

クライアント側のアプリケーションサーバの J2EE アプリケーションからのリクエ

スト（メッセージ）はデータベース上に実現されたキュー経由で送られ，サーバ側のアプリケーションサーバ上の Message-driven Bean を呼び出します。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。なお，リソースに接続するためのプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) サーバ側のアプリケーションサーバマシン

サーバ側のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) データベースサーバマシン

データベースサーバマシンには，HiRDB または Oracle をインストールする必要があります。

起動する必要があるプロセスは，HiRDB または Oracle のプロセスです。

(c) クライアントマシン（クライアント側のアプリケーションサーバマシン）

クライアントマシン（クライアント側のアプリケーションサーバマシン）には，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(d) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

3.8.4 Message-driven Bean のインスタンスプールを利用した負荷分散（TP1/Message Queue を使用する場合）

アクセスポイントになるコンポーネントが Message-driven Bean の場合に，J2EE サーバごとの Message-driven Bean のインスタンスのプール数に応じて負荷を分散する構成について説明します。

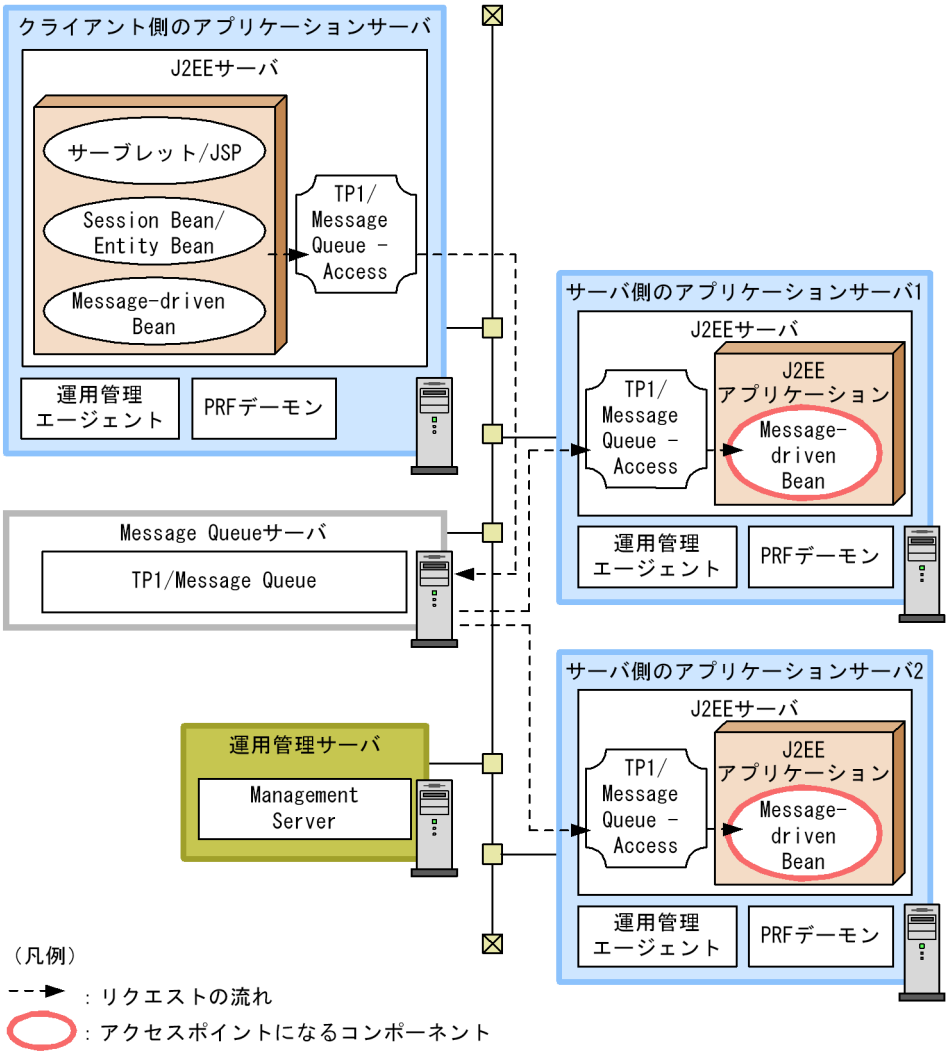
なお，この構成によって負荷を分散できるのは，TP1/Message Queue を使用して Message-driven Bean にアクセスする場合だけです。Cosminexus RM を使用して Message-driven Bean にアクセスする場合は，Message-driven Bean のインスタンスプールを利用した負荷分散はできません。

（1）システム構成の特徴

J2EE サーバ上で動作するアプリケーションのアクセスポイントが Message-driven Bean である，メッセージ駆動型のシステムで使用できる構成です。Message Queue クライアントからのメッセージが Message Queue サーバ上の TP1/Message Queue のキューに到着すると，J2EE サーバ上でプールされている Message-driven Bean のインスタンス数に応じて，Message-driven Bean が呼び出されます。

Message-driven Bean を対象にした負荷分散の構成の例を次の図に示します。

図 3-45　Message-driven Bean を対象にした負荷分散の構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- Message-driven Bean の負荷分散を実現して，高い可用性を確保できます。
- アプリケーションサーバを複数用意することで，アプリケーションサーバの負荷を分散できます。
- サーバ側のアプリケーションサーバで障害が発生した場合やアプリケーションサーバのメンテナンスが必要な場合，TP1/Message Queue から該当するアプリケーションサーバ上の J2EE サーバにはアクセスしません。このため，縮退運転ができます。

リクエストの流れ

アクセスポイントであるサーバ側のアプリケーションサーバの J2EE サーバ上の Message-driven Bean へのリクエスト（メッセージ）は，クライアント側のアプリケーションサーバの J2EE サーバ上で動作する J2EE アプリケーションから，TP1/Message Queue 経由で送られます。TP1/Message Queue にメッセージが到着すると，サーバ側のアプリケーションサーバの J2EE サーバ上の Message-driven Bean が呼び出されます。その際，Message-driven Bean のインスタンスプールのプール数に応じて，リクエストが振り分けられます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

Message-driven Bean のインスタンスプールを使用して負荷を分散する場合に必要なソフトウェアと起動するプロセスは，Message-driven Bean をアクセスポイントにする構成と同じです。「3.8.2　Message-driven Bean をアクセスポイントに使用する場合の構成（TP1/Message Queue を使用する場合）」を参照してください。

3.9　運用管理プロセスの配置を検討する

この節では，運用管理に使用するプロセスである，Management Server の配置について説明します。Management Server は，複数のホスト上に構築されているアプリケーションサーバのシステム全体を一括管理，および一括運用するプロセスです。Management Server を使用することで，各ホスト上のサーバの環境設定をまとめて実施したり，サーバを一括起動したりできます。また，システム全体の状態を把握できます。

ここでは，次の構成を説明します。

● 運用管理サーバに Management Server を配置する構成（運用管理サーバモデル）

● マシン単位に Management Server を配置する構成（ホスト単位管理モデル）

● コマンドを使用して運用する構成

なお，運用管理サーバモデルで Management Server を配置したマシンを，**運用管理サーバ**といいます。

参考

- Management Server の機能を使用して，J2EE サーバ，SFO サーバ，またはバッチサーバの稼働情報を監視・収集する場合，運用監視エージェントというエージェントプログラムを使用します。監視対象にするホスト上で，監視対象にする J2EE サーバ，SFO サーバ，またはバッチサーバに一つずつ含まれます。また，クラスタ構成の場合は，クラスタに含まれる J2EE サーバに一つずつ含まれます。
- 業務用のサーバを配置した LAN と管理用のサーバを配置した LAN に分けている場合，運用管理サーバを管理用のサーバを配置した LAN に置くこともできます。LAN を分け，一つのマシンを複数のネットワークセグメントに接続する場合，環境設定に注意が必要です。詳細については，マニュアル「Cosminexus 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「付録 E　一つのマシンを複数のネットワークセグメントに接続する場合の環境設定での注意」を参照してください。

3.9.1　運用管理サーバに Management Server を配置する構成

Management Server を利用して運用する場合に，ドメイン内に一つ運用管理サーバを配置する構成について説明します。

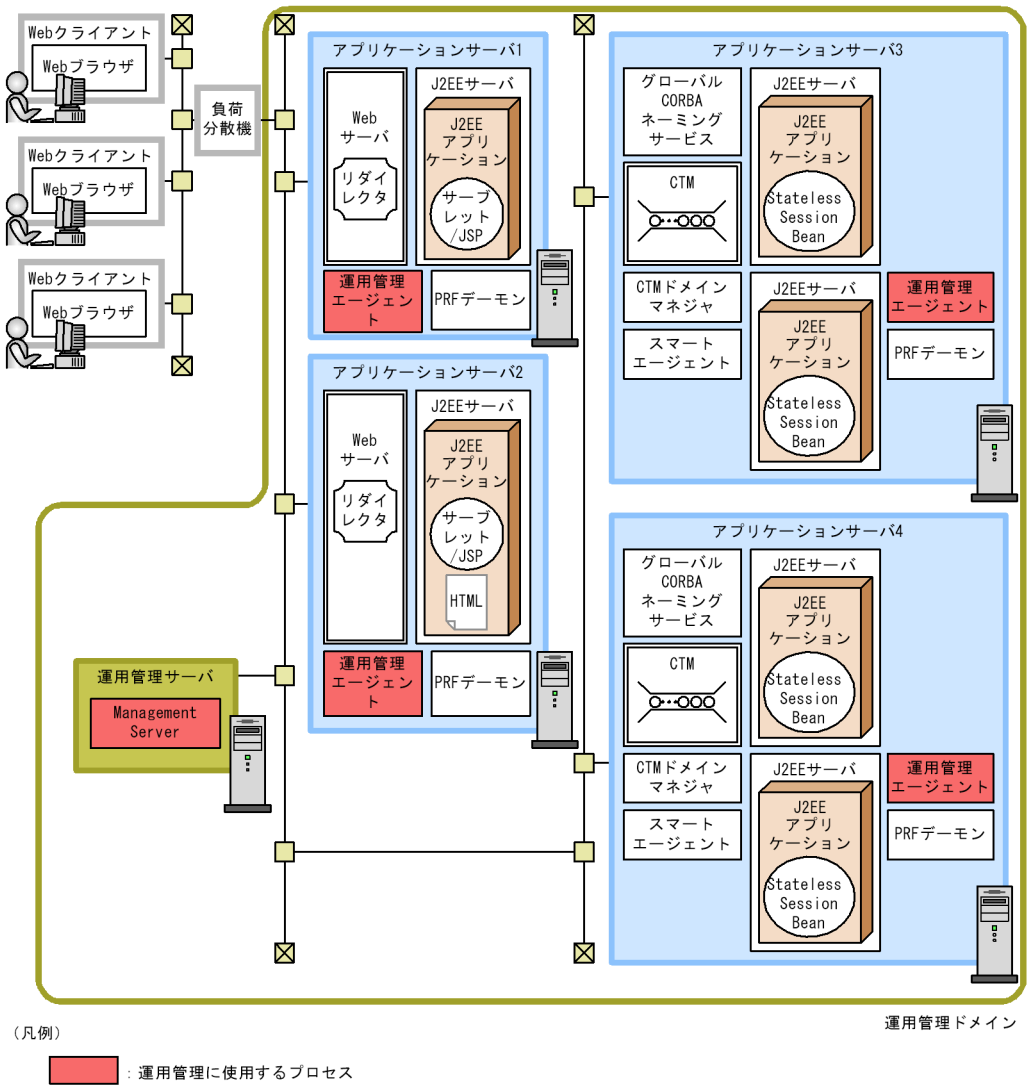
（1）システム構成の特徴

複数のアプリケーションサーバで構成されるシステムで，ドメイン内に運用管理サーバマシンを配置する構成です。Management Server であらかじめ定義したドメイン（**運用管理ドメイン**）内の運用管理および監視が実現できます。運用管理サーバマシンの Management Server は，それぞれのアプリケーションサーバに配置した運用管理エー

ジェントと協調して，運用管理操作を実行します。

運用管理サーバに Management Server を配置する構成の例を次の図に示します。なお，この例では，接続するリソースマネジャについては省略しています。

図 3-46　運用管理サーバに Management Server を配置する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- システムの一括運用に適しています。
- 独立した運用管理サーバ用のマシンを用意することをお勧めします。ただし，システム内の一つのアプリケーションサーバ内に Management Server を配置するこ

ともできます。

(2) それぞれのマシンで起動するプロセス

運用管理サーバに Management Server を配置する構成の場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

ポイント

運用管理サーバマシンを用意できないときは，ドメイン内のどれかのアプリケーションサーバが運用管理サーバを兼ねるシステム構成にもできます。
この場合は，運用管理サーバにするアプリケーションサーバマシンに，Management Server のプロセスを配置してください。

(b) アプリケーションサーバマシン

運用管理サーバで運用管理するために，それぞれのアプリケーションサーバマシンで起動するプロセスは次のとおりです。

● 運用管理エージェント

これ以外にアプリケーションサーバマシンに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

3.9.2 マシン単位に Management Server を配置する構成

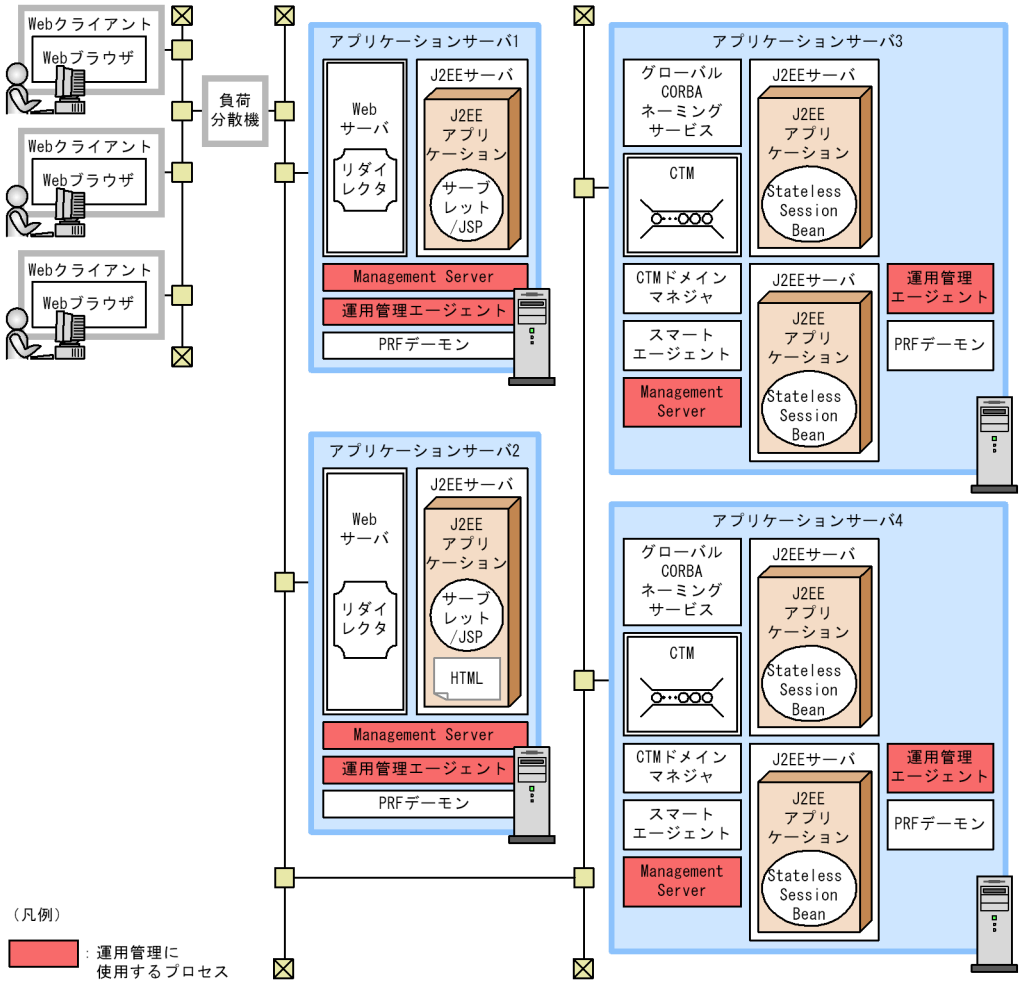
Management Server を利用して運用する場合に，それぞれのアプリケーションサーバマシンに Management Server を配置する構成について説明します。

(1) システム構成の特徴

複数のアプリケーションサーバで構成されるシステムで，Management Server をそれぞれのアプリケーションサーバマシンに配置する構成です。アプリケーションサーバ単位の運用管理および監視が実行できます。Management Server は，運用管理エージェントと協調して，運用管理操作を実行します。

マシン単位に Management Server を配置する構成の例を次の図に示します。なお，この例では，接続するリソースマネジャについては省略しています。

図 3-47　マシン単位に Management Server を配置する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションサーバ単位の運用に適しています。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

マシン単位に Management Server を配置する構成の場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン

マシン単位の運用管理をするためにそれぞれのアプリケーションサーバマシンで起動するプロセスは次のとおりです。

● Management Server

● 運用管理エージェント

これ以外にアプリケーションサーバマシンに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

3.9.3 コマンドで運用する場合の構成

Management Server を利用しないで運用する場合の構成について説明します。

Management Server を利用しない場合は，定義ファイルの編集や，サーバ管理コマンドの実行などによって，アプリケーションサーバを運用します。このとき，それぞれのアプリケーションサーバに，運用管理エージェントおよび Management Server は不要です。アプリケーションサーバ以外の運用支援ソフトウェアなどを使用してシステムを管理する場合に，この構成を検討してください。

3.10 セッション情報の引き継ぎを検討する

この節では，セッション情報の引き継ぎを実現するためのシステム構成について説明します。この構成は，J2EE サーバが負荷分散機によって冗長化されていることが前提です。

セッション情報を引き継ぐ構成には，次の二とおりがあります。

- 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使用する場合）
- SFO サーバを使用する構成（メモリセッションフェイルオーバ機能を使用する場合）
  SFO サーバを使用する構成の場合，SFO サーバはシステムに一つまたは複数配置できます。SFO サーバにトラブルが発生した場合の影響範囲を小さくするために，一つの J2EE アプリケーションに対して一つの SFO サーバを配置することをお勧めします。ここでは次の 2 種類の構成を説明します。
  - SFO サーバがシステムに複数存在する構成
  - SFO サーバがシステムに一つだけ存在する構成

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）

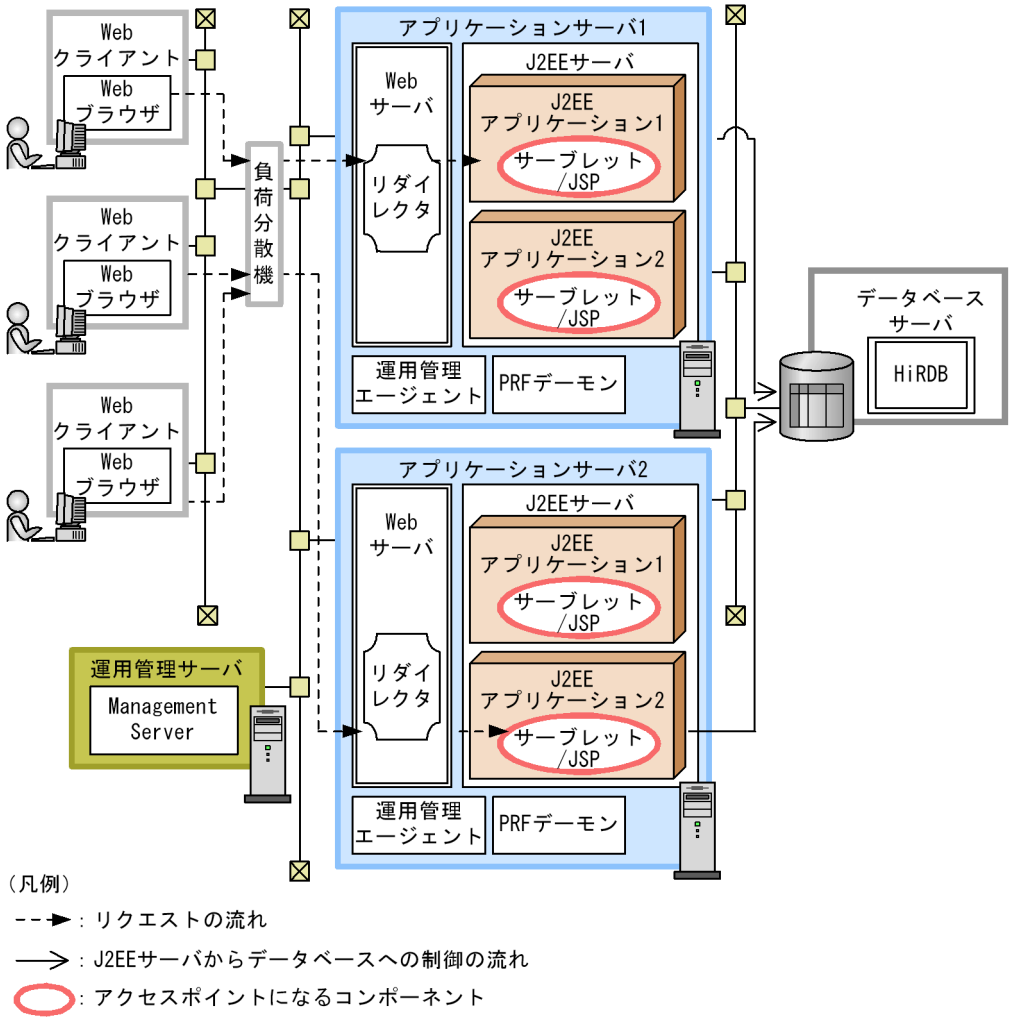
データベースを使用してセッション情報を引き継ぐ構成について説明します。

（1） システム構成の特徴

セッション情報を格納するためのデータベースを用意します。業務情報を格納するために使用しているデータベースがある場合は，同じデータベースを使用できます。

データベースを使用してセッション情報を引き継ぐ場合の構成の例を次の図に示します。

図 3-48　データベースを使用してセッション情報を引き継ぐ構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 特定の J2EE サーバでトラブルが発生した場合に，セッション情報をほかの J2EE サーバに引き継げます。
- データベースサーバにトラブルが発生した場合も，データベースに格納したセッション情報は残ります。データベースサーバ再起動後に利用できます。

リクエストの流れ

データベースでは，J2EE サーバ上のグローバルセッション情報が冗長化されて管理されています。

J2EE サーバがリクエストを受け付けてセッションが確立された時点で，データベースに格納されたグローバルセッション情報にロックが掛けられます。J2EE ア

プリケーションの処理が完了すると，J2EE サーバ上のグローバルセッション情報の内容に合わせてデータベースのグローバルセッション情報が更新されます。そのあとで，データベースのロックが解除されます。
J2EE サーバにトラブルが発生した場合は，データベースに格納したグローバルセッション情報が別な J2EE サーバから取得され，セッション情報が引き継がれます。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン

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，データベースに接続するためのソフトウェアをインストールする必要があります。必要なソフトウェアを次の表に示します。

使用するデータベース	必要なソフトウェア
HiRDB	HiRDB Type4 JDBC Driver
Oracle	Oracle JDBC Thin Driver

これ以外に必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

(b) データベースサーバマシン

データベースが動作するマシンには，次に示すソフトウェアのどちらかをインストールしてください。

- HiRDB（HiRDB と接続する場合）
- Oracle（Oracle と接続する場合）

また，それぞれのデータベースで必要なプロセスを起動してください。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

(d) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.10.2 SFO サーバがシステムに複数存在する構成（メモリセッションフェイルオーバ機能）

SFO サーバがシステムに複数存在する構成について説明します。

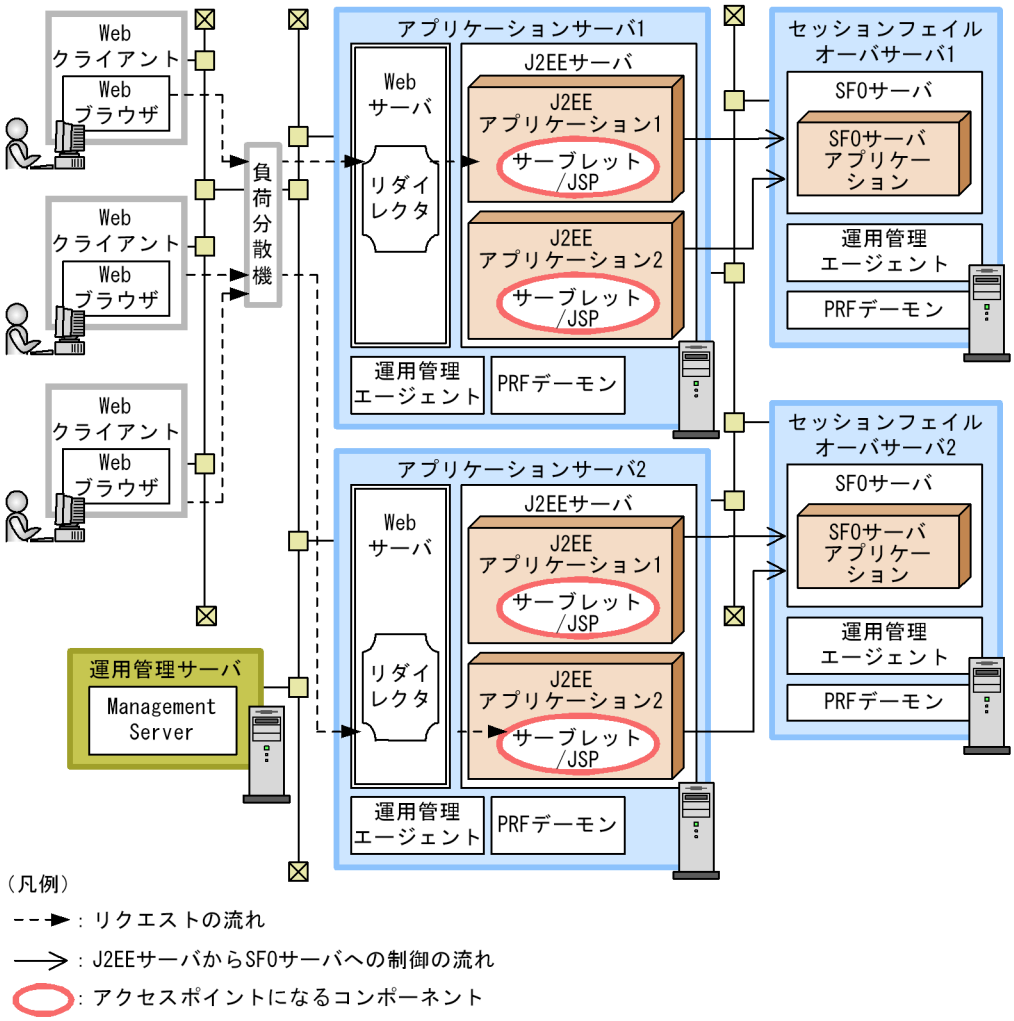
（1）システム構成の特徴

J2EE サーバ上で動作する J2EE アプリケーションごとに，SFO サーバを用意します。J2EE アプリケーションと SFO サーバを 1:1 にすることをお勧めします。これによって，SFO サーバにトラブルが発生した場合の影響を最小限に抑えられます。これに対して，複数の J2EE アプリケーションに対して一つの SFO サーバを配置した場合，SFO サーバにトラブルが発生したときには，その SFO サーバに対応するすべての J2EE アプリケーションのセッション情報が引き継げなくなります。

SFO サーバ上では，SFO サーバアプリケーションが動作しています。

システムに複数の SFO サーバが存在する場合の構成の例を次の図に示します。この例では，SFO サーバは J2EE サーバと異なるマシン上に配置されています。この構成の場合，SFO サーバを配置したマシンを，**セッションフェイルオーバサーバ**といいます。

図 3-49　SFO サーバがシステムに複数存在する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 特定の J2EE サーバでトラブルが発生した場合に，セッション情報をほかの J2EE サーバに引き継げます。
- SFO サーバにトラブルが発生した場合も，対応する J2EE アプリケーションだけに影響範囲を抑えられます。

リクエストの流れ

SFO サーバでは，J2EE サーバ上のグローバルセッション情報が冗長化されて管理されています。

J2EE サーバがリクエストを受け付けてセッションが確立された時点で，J2EE サーバ上のセッションフェイルオーバ用フィルタによって SFO サーバ上のグローバル

セッション情報にロックが掛けられます。J2EE アプリケーションの処理が完了すると，J2EE サーバ上のグローバルセッション情報の内容に合わせて SFO サーバ上のグローバルセッション情報が更新されます。そのあとで，セッションフェイルオーバ用フィルタによって SFO サーバ上のグローバルセッション情報のロックが解除されます。
J2EE サーバにトラブルが発生した場合は，別な J2EE サーバから SFO サーバ上のグローバルセッション情報が取得され，セッション情報が引き継がれます。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン

メモリ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ソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

(b) セッションフェイルオーバサーバマシン

セッションフェイルオーバサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- SFO サーバ
- 運用管理エージェント
- PRF デーモン

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

(d) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.10.3 SFO サーバがシステムに一つだけ存在する構成（メモリセッションフェイルオーバ機能）

SFO サーバがシステムに一つだけ存在する構成について説明します。

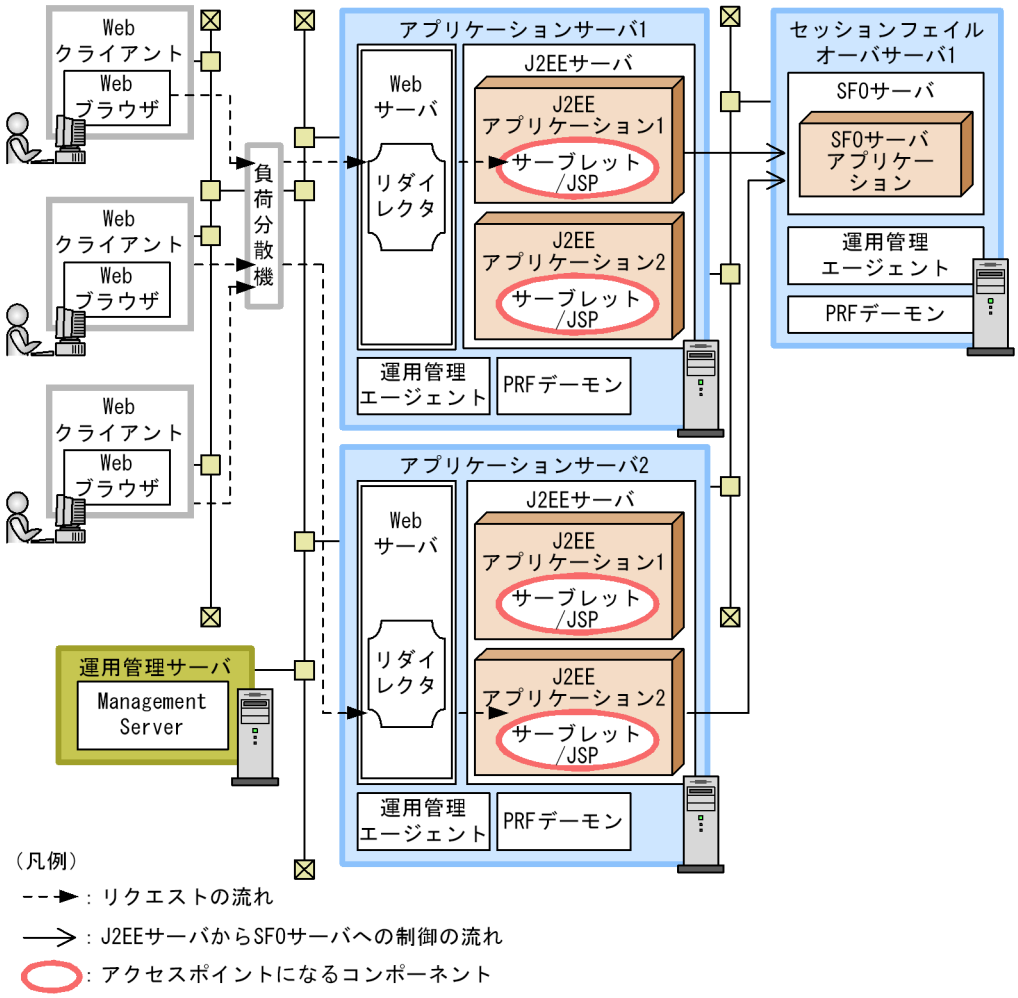
（1）システム構成の特徴

J2EE サーバ上で動作する複数の J2EE アプリケーションに対して，一つの SFO サーバを用意します。

SFO サーバ上では，SFO サーバアプリケーションが動作しています。

システムに SFO サーバが一つだけ存在する場合の構成の例を次の図に示します。なお，この例では，SFO サーバは J2EE サーバと異なるマシン上に配置されています。

図 3-50　SFO サーバがシステムに一つだけ存在する構成の例（セッションフェイルオーバサーバを配置した場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 特定の J2EE サーバでトラブルが発生した場合に，セッション情報をほかの

J2EE サーバに引き継げます。

- SFO サーバにトラブルが発生した場合は，システム上のすべての J2EE アプリケーションでセッション情報が引き継げなくなります。

リクエストの流れ

SFO サーバでは，J2EE サーバ上のグローバルセッション情報が冗長化されて管理されています。

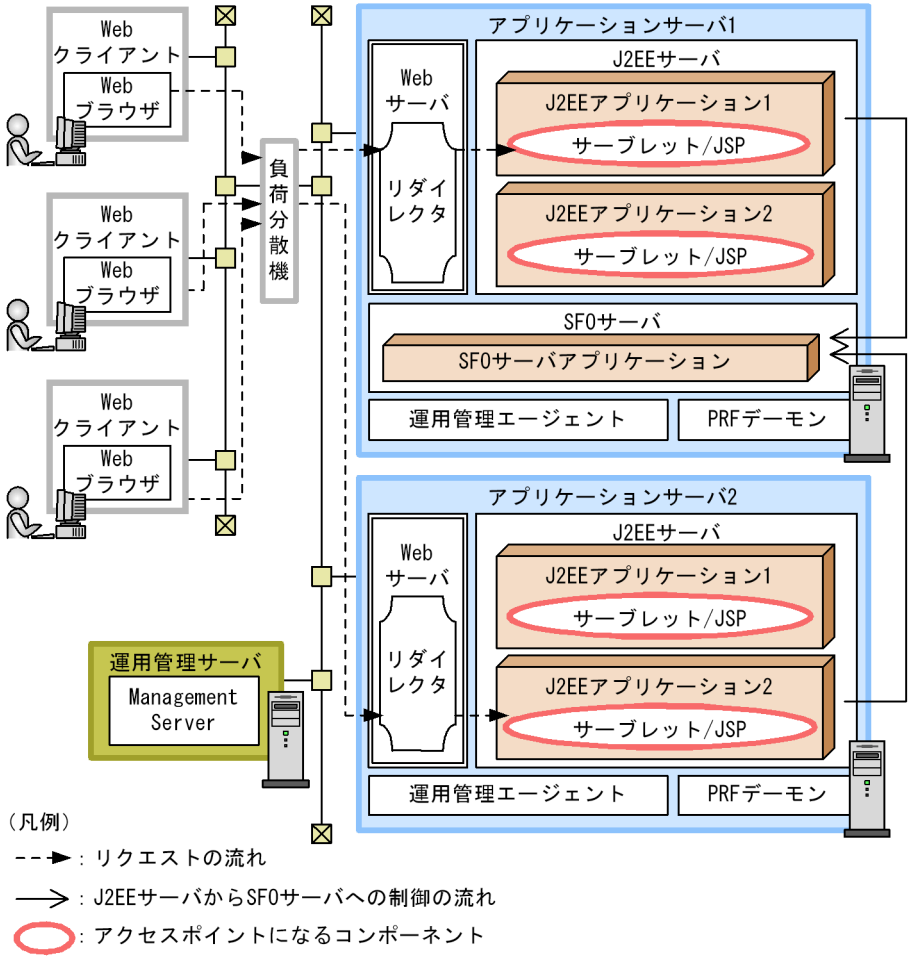
J2EE サーバがリクエストを受け付けてセッションが確立された時点で，J2EE サーバ上のセッションフェイルオーバ用フィルタによって SFO サーバ上のグローバルセッション情報にロックが掛けられます。J2EE アプリケーションの処理が完了すると，J2EE サーバ上のグローバルセッション情報の内容に合わせて SFO サーバ上のグローバルセッション情報が更新されます。そのあとで，セッションフェイルオーバ用フィルタによって SFO サーバ上のグローバルセッション情報のロックが解除されます。

J2EE サーバにトラブルが発生した場合は，別な J2EE サーバから SFO サーバ上のグローバルセッション情報が取得され，セッション情報が引き継がれます。

なお，SFO サーバは，J2EE サーバと同じマシンにも配置できます。

SFO サーバを J2EE サーバと同じマシンに配置する構成の例を次の図に示します。

図 3-51　SFO サーバを J2EE サーバと同じマシンに配置する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴およびリクエストの流れについては，別なマシンとしてセッションフェイルオーバサーバを配置する場合と同様です。

注意事項

この構成の場合，J2EE サーバのプロセス障害には対応できますが，アプリケーションサーバ 1 でハードウェア障害が発生した場合には対応できません。アプリケーションサーバで発生するハードウェア障害に対応するためには，SFO サーバを別のマシンに配置してください。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン

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ソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

なお，SFO サーバを J2EE サーバと同じマシンに配置する場合は，次のプロセスを起動してください。

● SFO サーバ

(b) セッションフェイルオーバサーバマシン

セッションフェイルオーバサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● SFO サーバ

● 運用管理エージェント

● PRF デーモン

なお，SFO サーバを J2EE サーバと同じマシンに配置する場合は，このマシンは不要です。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

(d) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.11 クラスタソフトウェアを使用した障害時の系切り替えを検討する

この節では，クラスタソフトウェアを使用した障害時の系切り替えを実現するためのシステム構成について説明します。

アプリケーションサーバのシステムでは，OS ごとに，次のクラスタソフトウェアを使用できます。

● **Windows の場合**
Microsoft Cluster Service

● **AIX，HP-UX または Linux の場合**
HA モニタ

● **Solaris の場合**
クラスタソフトウェアと連携したシステム運用はできません。

アプリケーションサーバで構築するシステムでは，クラスタソフトウェアを使用して，次に示す構成を実現できます。なお，これらの構成は，Management Server を利用して運用することが前提になります。

● **アプリケーションサーバの実行系と待機系を 1:1 にする構成**
実行系のアプリケーションサーバマシン 1 台に対して待機系のアプリケーションサーバマシンを 1 台用意しておく構成です。実行系のアプリケーションサーバマシンにトラブルが発生してマシンが終了した場合，または運用管理エージェントが終了した場合に，クラスタソフトウェアによって待機系のアプリケーションサーバを起動して処理を切り替えます。トランザクションサービスを使用しない場合と使用する場合で，共有ディスク装置の要否が異なります。なお，Microsoft Cluster Service を使用する場合は，トランザクションサービスを使用しない場合も，共有ディスク装置が必要です。

● **運用管理サーバの実行系と待機系を 1:1 にする構成**
実行系の運用管理サーバマシン 1 台に対して待機系の運用管理サーバマシンを 1 台用意しておく構成です。実行系の運用管理サーバマシンにトラブルが発生してマシンが終了した場合，または Management Server のプロセスが終了した場合に，クラスタソフトウェアによって待機系の運用管理サーバを起動して処理を切り替えます。

● **アプリケーションサーバの実行系と待機系を相互スタンバイにする構成**
アプリケーションサーバの実行系と待機系を 1：1 にする構成の一つです。それぞれのアプリケーションサーバに同じ種類の J2EE サーバを配置して，異なる J2EE サーバを起動しておくことで，それぞれのアプリケーションサーバが実行系として動作しながらお互いの待機系として機能します。どちらか一方の系に障害が発生すると，もう一方の系に切り替えられます。これによって，少ない台数のアプリケーションサーバマシンで，むだの少ない運用が可能になります。

この構成では，それぞれのアプリケーションサーバマシンで Management Server を起動して，それぞれに異なる運用管理ドメインを管理します。

● N 台の実行系に対して 1 台の待機系を用意する構成（リカバリ専用サーバを使用する構成）

この構成は，J2EE サーバを冗長化した構成でグローバルトランザクションを使用する場合に，特定の J2EE サーバにトラブルが発生したときにトランザクションを決着するためのシステム構成です。

N 台の J2EE サーバを冗長化するためには，負荷分散機などを使用します。

● ホスト単位管理モデルの実行系と予備系を N：1 にする構成

アプリケーションサーバマシン（ホスト）の実行系複数（1 ～ N 台）と待機系 1 台を配置し，それぞれに Management Server および運用管理エージェントを配置するシステムです。実行系のアプリケーションサーバマシンに障害が発生したときに待機系のアプリケーションサーバマシンに系を切り替えて，業務を続行できます。

3.11.1　アプリケーションサーバの実行系と待機系を 1:1 にする構成（トランザクションサービスを使用しない場合）

アプリケーションサーバの実行系と待機系を 1:1 にする構成の場合に，トランザクションサービスを使用しないときの構成について説明します。

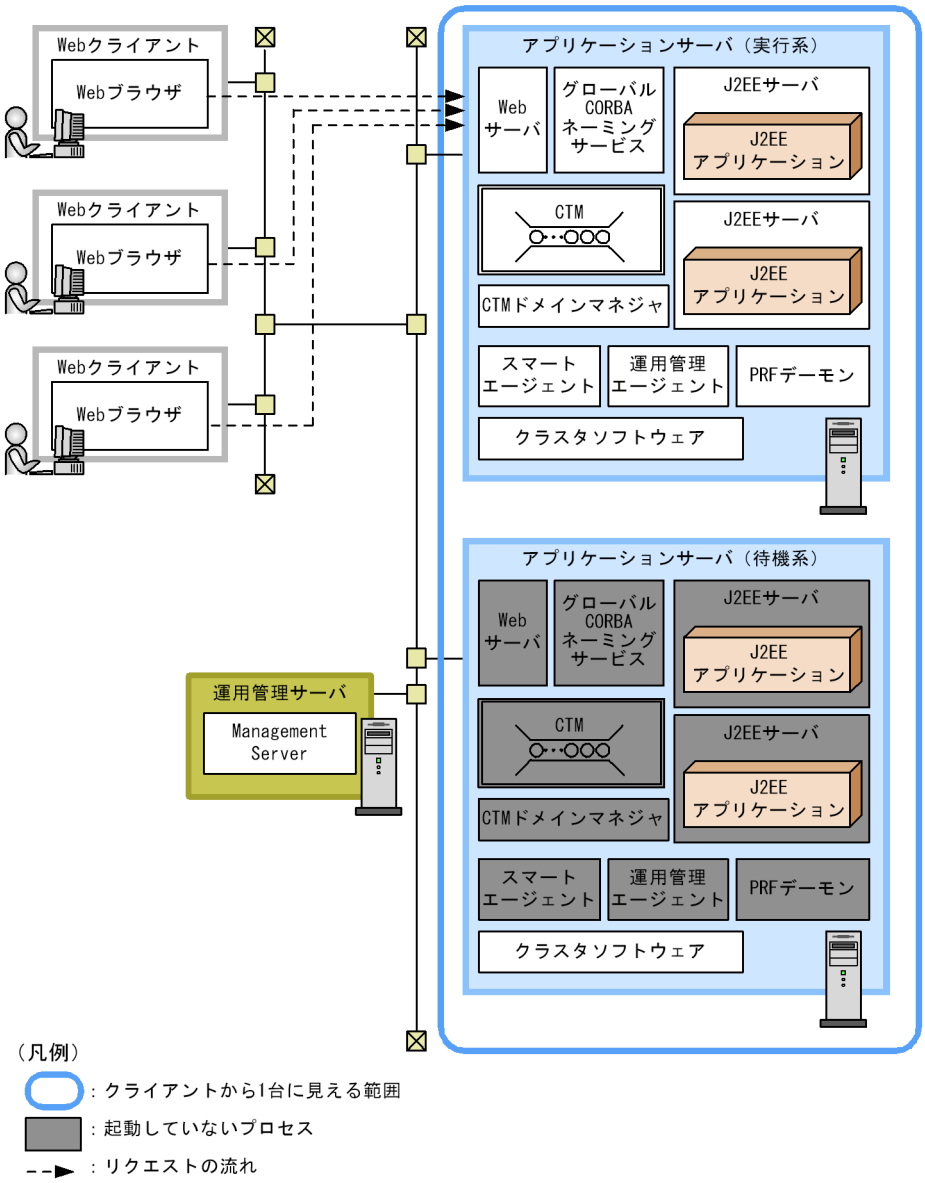
（1） システム構成の特徴

ローカルトランザクションを使用する場合など，トランザクションサービスを使用しない場合のシステム構成です。

なお，ここでは，アプリケーションサーバを系切り替えする場合の構成を示します。運用管理サーバを系切り替えする場合の構成については，「3.11.3　運用管理サーバの実行系と待機系を 1:1 にする構成」を参照してください。

トランザクションサービスを使用しない場合の構成の例を次の図に示します。

図 3-52　クラスタソフトウェアを使用して実行系と待機系を 1:1 にする場合のシステム構成の例（トランザクションサービスを使用しない場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 実行系と待機系は 1：1 で構成します。
- HA モニタを使用する場合は，共有ディスク装置は不要です。Microsoft Cluster Service を使用する場合は，共有ディスク装置が必要です。図 3-55 は，HA モニタの場合の構成です。

- データベースがクラスタ構成になっている場合でも，アプリケーションサーバでは仮想アドレス（論理アドレス）だけを認識していれば，接続できます。
- Management Server をアプリケーションサーバと同じマシンに配置する場合については，「3.11.3　運用管理サーバの実行系と待機系を 1:1 にする構成」を参照してください。

リクエストの流れ

クライアントからは，実行系，待機系のアプリケーションサーバは，合わせて 1 台として認識されます。リクエストは，すべて実行系のアプリケーションサーバに送信されます。

系切り替えの流れ

実行系のアプリケーションサーバマシンに障害が発生すると，系切り替えが実行されます。系切り替えは，OS，または運用管理エージェントがダウンした場合に実行されます。なお，このほか，明示的に系切り替えを実行するコマンドを実行した場合や，これ以外にクラスタソフトウェアが系切り替えの対象にしている事象が発生した場合も，系切り替えが実行されます。

系切り替えが実行されると，待機系のアプリケーションサーバの運用管理エージェント，およびそれまで停止していたプロセスが起動されます。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン（実行系）

クラスタソフトウェアを使用する場合に必ず起動するプロセスは次のとおりです。

● 運用管理エージェント

これ以外にアプリケーションサーバに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

(b) アプリケーションサーバマシン（待機系）

待機系のアプリケーションサーバマシンにインストールするソフトウェアの構成は，実行系と一致させてください。ただし，系切り替えが発生するまで，プロセスは起動しません。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

（d）クライアントマシン

クライアントマシンには，次のソフトウェアをインストールしてください。

Web クライアント構成の場合
Web ブラウザ

EJB クライアント構成の場合
uCosminexus Client（Windows の場合），Application Server Standard または Application Server Enterprise

3.11.2 アプリケーションサーバの実行系と待機系を 1:1 にする構成（トランザクションサービスを使用する場合）

アプリケーションサーバの実行系と待機系を 1:1 にする構成の場合に，トランザクションサービスを使用するときの構成について説明します。

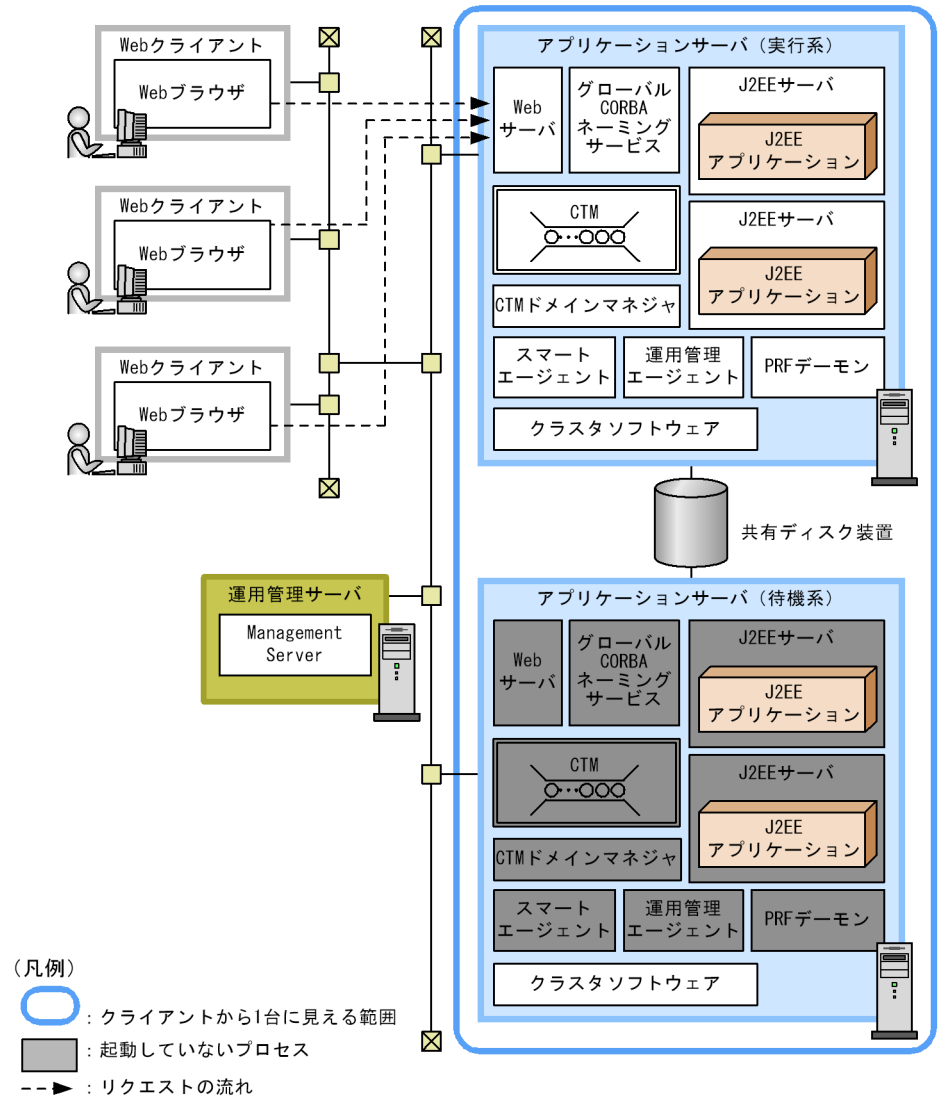
（1）システム構成の特徴

グローバルトランザクションを使用する場合など，トランザクションサービスを使用する場合のシステム構成です。この場合，共有ディスク装置が必要です。

なお，ここでは，アプリケーションサーバを系切り替えする場合の構成を示します。運用管理サーバを系切り替えする場合の構成については，「3.11.3　運用管理サーバの実行系と待機系を 1：1 にする構成」を参照してください。

トランザクションサービスを使用する場合の構成の例を次の図に示します。

図 3-53　クラスタソフトウェアを使用してアプリケーションサーバの実行系と待機系を 1:1 にする場合のシステム構成の例（トランザクションサービスを使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 実行系と待機系は１：１で構成します。
- 共有ディスク装置が必要です。
- データベースがクラスタ構成になっている場合でも，アプリケーションサーバでは仮想アドレス（論理アドレス）だけを認識していれば，接続できます。
- Management Server をアプリケーションサーバと同じマシンに配置する場合につ

いては，「3.11.3　運用管理サーバの実行系と待機系を 1:1 にする構成」を参照してください。

リクエストの流れ

クライアントからは，実行系，待機系のアプリケーションサーバは，合わせて 1 台として認識されます。リクエストは，すべて実行系のアプリケーションサーバに送信されます。

系切り替えの流れ

実行系のアプリケーションサーバマシンに障害が発生すると，系切り替えが実行されます。系切り替えは，OS，または運用管理エージェントがダウンした場合に実行されます。なお，このほか，明示的に系切り替えを実行するコマンドを実行した場合や，これ以外にクラスタソフトウェアが系切り替えの対象にしている事象が発生した場合も，系切り替えが実行されます。

系切り替えが実行されると，待機系のアプリケーションサーバの運用管理エージェント，およびそれまで停止していたプロセスが起動されます。なお，トランザクション情報については，共有ディスクに格納されていた情報が引き継がれます。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスは，トランザクションサービスを利用しない場合と同じです。

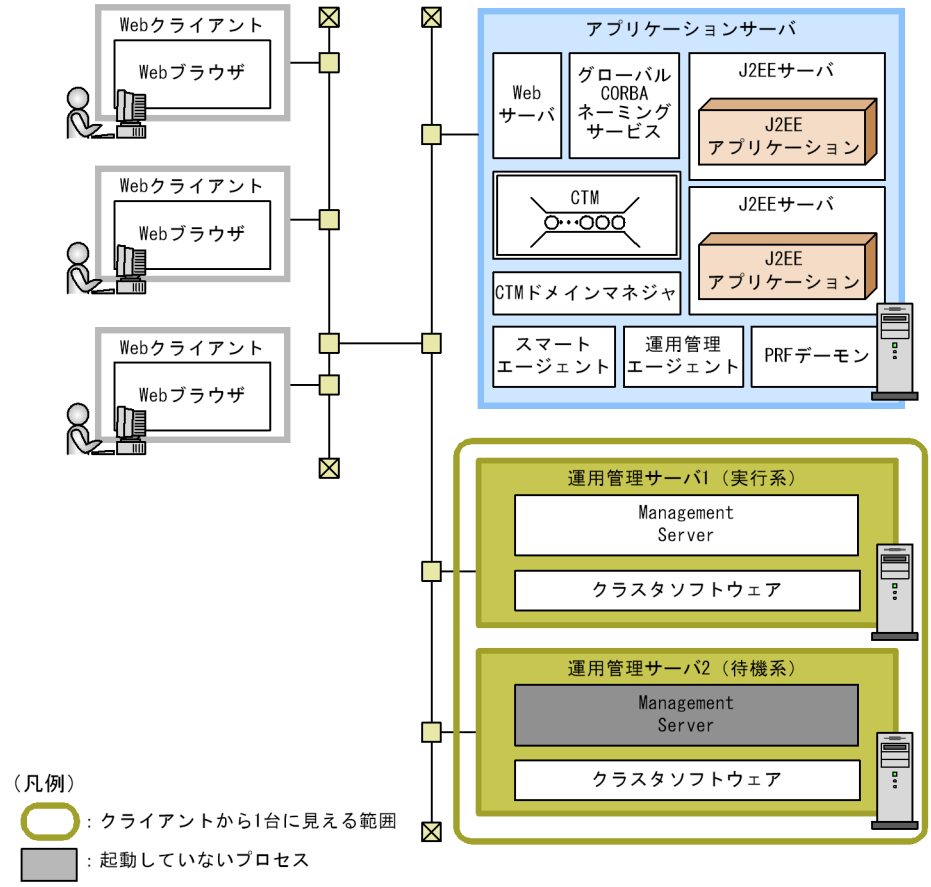
3.11.3　運用管理サーバの実行系と待機系を 1:1 にする構成

運用管理サーバを系切り替えの対象にする場合の構成について説明します。運用管理サーバの実行系と待機系を 1：1 にします。

(1) システム構成の特徴

運用管理サーバの実行系と待機系を 1：1 にする場合の構成の例を次に示します。

図 3-54　クラスタソフトウェアを使用して運用管理サーバの実行系と待機系を 1:1 にする場合のシステム構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 実行系と待機系は 1：1 で構成します。それぞれのマシンに Management Server を配置します。
- Microsoft Cluster Service を使用する場合は，共有ディスク装置が必要です。

系切り替えの流れ

実行系の運用管理サーバマシンに障害が発生すると，系切り替えが実行されます。系切り替えは，OS，または Management Server がダウンした場合に実行されます。なお，このほか，明示的に系切り替えを実行するコマンドを実行した場合や，これ以外にクラスタソフトウェアが系切り替えの対象にしている事象が発生した場合も，系切り替えが実行されます。

系切り替えが実行されると，待機系の運用管理サーバの Management Server のプロセスが起動されます。

（2） それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

（a） アプリケーションサーバマシン

アプリケーションサーバに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

（b） 運用管理サーバマシン（実行系）

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

（c） 運用管理サーバマシン（待機系）

待機系のアプリケーションサーバマシンにインストールするソフトウェアの構成は，実行系と一致させてください。ただし，系切り替えが発生するまで，プロセスは起動しません。

（d） クライアントマシン

クライアントマシンには，次のソフトウェアをインストールしてください。

Web クライアント構成の場合
　Web ブラウザ

EJB クライアント構成の場合
　uCosminexus Client（Windows の場合），Application Server Standard または Application Server Enterprise

3.11.4 アプリケーションサーバの実行系と待機系を相互スタンバイにする構成

アプリケーションサーバの実行系と待機系を 1：1 にする構成で，それぞれのアプリケーションサーバを実行系として稼働させながら，お互いの待機系にする構成について説明します。この構成を，**相互スタンバイ構成**といいます。また，この構成で構築されたシステムを，**相互系切り替えシステム**といいます。

（1） システム構成の特徴

この構成は，実行系のアプリケーションサーバマシン 1 台に対して待機系のアプリケーションサーバマシンを 1 台用意しておく構成の一つです。ただし，待機系のアプリケーションサーバを停止させておくのではなく，実行系とは異なる J2EE サーバを動作させ

ておきます。これを，**相互スタンバイ**といいます。

それぞれのアプリケーションサーバに同じ種類の J2EE サーバを配置して，異なる J2EE サーバを起動しておくことで，それぞれのアプリケーションサーバが現用系として動作しながらお互いの予備系として機能します。どちらかのマシンに障害が発生した場合は，系が切り替えられ，もう片方のアプリケーションサーバマシンで両方の J2EE サーバの処理が実行されるようになります。

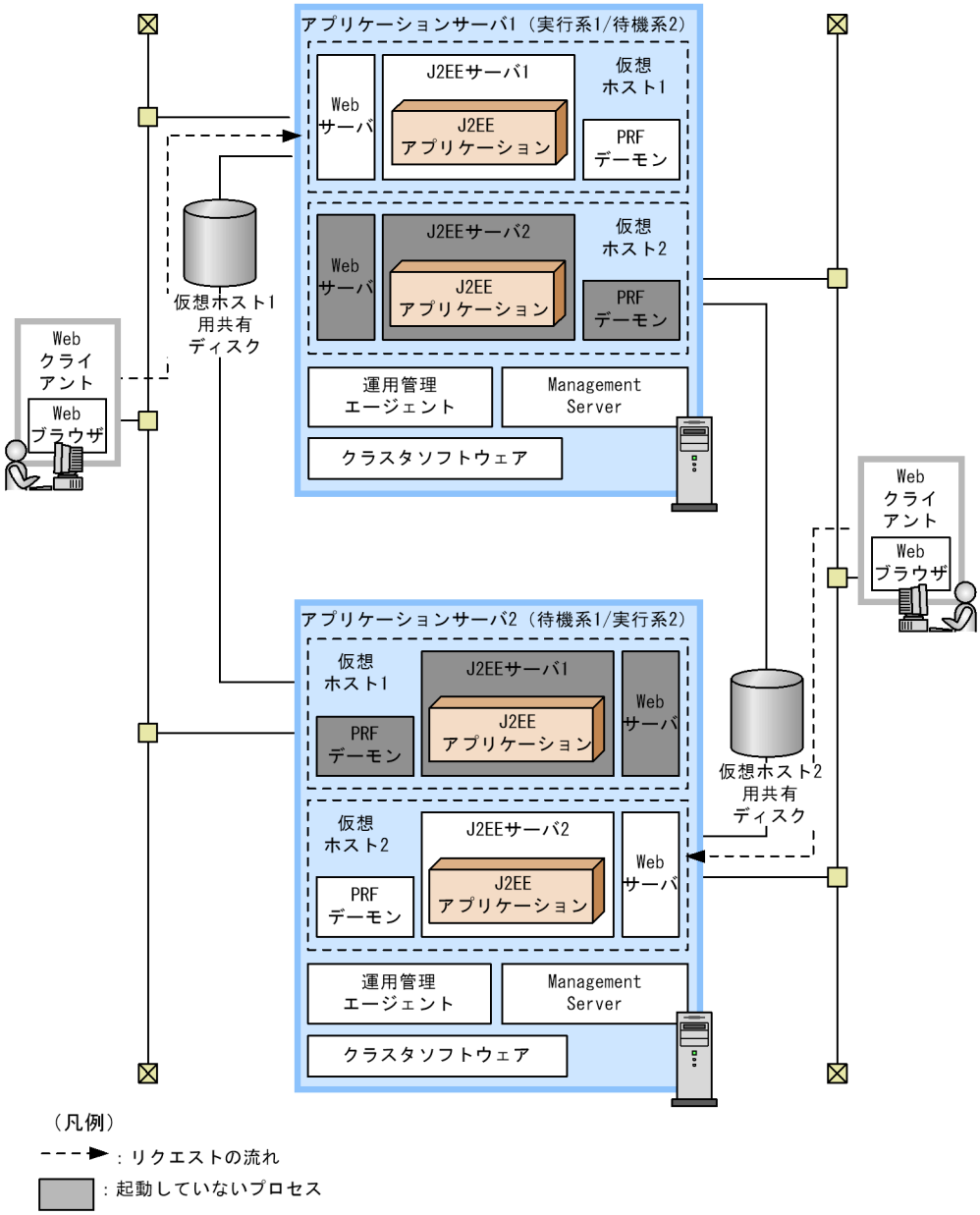
この構成は，次のことが前提になります。

- CORBA ネーミングサービスはインプロセスで起動されていること
- グローバルトランザクションを使用する場合，トランザクションのステータスファイルは共有ディスクに格納されていること
- Management Server を利用して運用していること

Management Server は，アプリケーションサーバと同じマシンに，一つずつ配置して，両方のホストで起動します。それぞれの Management Server は，別の運用管理ドメインを管理します。それぞれの運用管理ドメインの範囲は，Management Server が配置されているアプリケーションサーバマシン内です。

相互系切り替えシステムの構成の例を次の図に示します。この例は，トランザクションサービスを使用する例です。共有ディスクは，トランザクションサービスを使用する場合にだけ必要です。

図 3-55　相互系切り替えシステムの構成の例（トランザクションサービスを使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 実行系と待機系は 1：1 で構成します。それぞれのマシンに Management Server を配置します。
- 一つの運用管理ドメイン（一つのマシン）内に，二つの仮想ホストを定義します。

それぞれの仮想ホストを，実行系のアプリケーションサーバのホスト，待機系のアプリケーションサーバのホストとして，システムを構築します。

- 運用管理ドメイン内のそれぞれの仮想ホストは，一つの運用管理エージェントで起動・停止を制御されます。ただし，それぞれの仮想ホストには実際に運用に使用される IP アドレスとは異なる IP アドレスが割り当てられているため，見かけ上異なるホスト上で動作しているように定義されています。
- 各アプリケーションサーバの運用に使用する IP アドレスは，クラスタソフトウェアによって動的に割り当てられる IP アドレスを使用します。Management Server から運用管理エージェントに対するリクエストの送信には，系切り替えによって他系へ移動しない IP アドレスを使用します。
- グローバルトランザクションを使用する場合は，共有ディスク装置が必要です。共有ディスクは，仮想ホストごとに必要です。
- 図 3-58 の場合，仮想ホスト単位に LAN を分けていますが，必須ではありません。

系切り替えの流れ

どちらかのアプリケーションサーバマシンに障害が発生すると，系切り替えが実行されます。系切り替えは，Management Server，運用管理エージェント，またはクラスタソフトウェアなどに障害が発生した場合に実行されます。なお，このほか，明示的に系切り替えを実行するコマンドを実行した場合や，これ以外にクラスタソフトウェアが系切り替えの対象にしている事象が発生した場合も，系切り替えが実行されます。

系切り替えが実行されると，待機系だった側のアプリケーションサーバマシンの運用管理エージェントによって，それまで停止していた論理サーバのプロセスが起動されます。

例えば，図 3-58 の場合に，アプリケーションサーバ 1 に障害が発生した場合は，クラスタソフトウェアによってアプリケーションサーバ 2 への系切り替えが実行され，アプリケーションサーバ 2 の運用管理エージェントによって，アプリケーションサーバ 2 上の J2EE サーバ 1 およびそれ以外の論理サーバのプロセスが起動されます。

（2） それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

（a） アプリケーションサーバマシン（実行系 1 ／待機系 2）

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

相互系切り替えシステムのアプリケーションサーバマシンで必ず起動するプロセスは次のとおりです。

● 運用管理エージェント

● Management Server

これ以外にアプリケーションサーバに必要なソフトウェアおよびプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

なお，系切り替えの対象になるプロセスは，仮想ホスト 1 のプロセスだけを起動してください。

(b) アプリケーションサーバマシン（待機系 1 ／実行系 2）

待機系のアプリケーションサーバマシンにインストールするソフトウェアおよびプロセスの構成は，実行系 1 ／待機系 2 と一致させてください。

なお，系切り替えの対象になるプロセスは，仮想ホスト 2 のプロセスだけを起動してください。

(c) クライアントマシン

クライアントマシンには，次のソフトウェアをインストールしてください。

Web クライアント構成の場合

Web ブラウザ

EJB クライアント構成の場合

uCosminexus Client（Windows の場合），Application Server Standard または Application Server Enterprise

3.11.5 リカバリ専用サーバを使用する場合の構成（N:1 リカバリシステム）

N 台の実行系に対して 1 台の待機系（**リカバリ専用サーバ**）を用意する構成について説明します。この構成を，**N:1 リカバリシステム**といいます。

(1) システム構成の特徴

この構成は，J2EE サーバを冗長化した構成でグローバルトランザクションを使用する場合に，特定の J2EE サーバにトラブルが発生したときにトランザクションを解決してリソースを解放するためのシステム構成です。N 台の J2EE サーバを冗長化するためには，負荷分散機などを使用します。

次のことが前提になります。

- グローバルトランザクションを使用するシステムで利用されること
- CORBA ネーミングサービスはインプロセスで起動されていること
- トランザクションのステータスファイルは共有ディスクに格納されていること
- Management Server を利用して運用していること

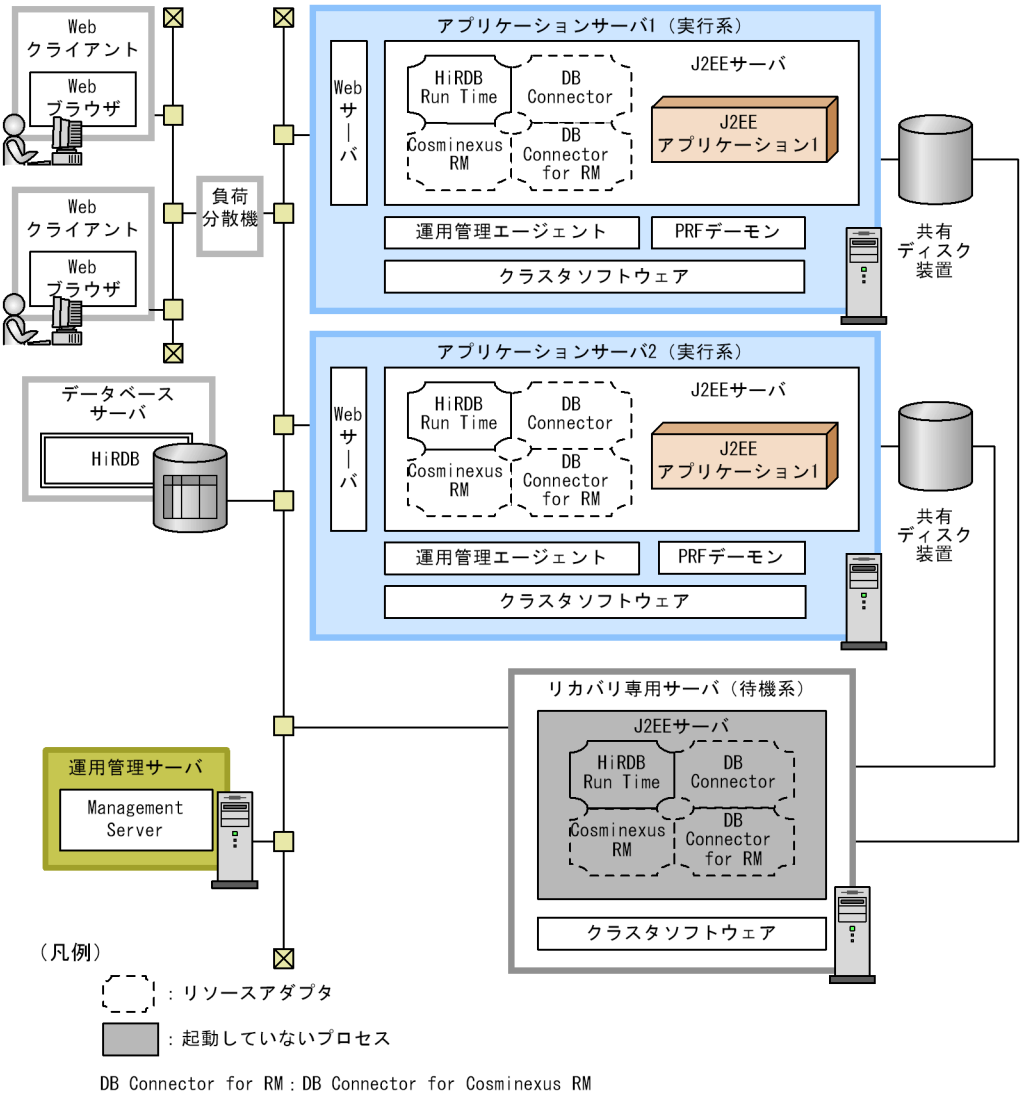
なお，N:1 リカバリシステムで系切り替えの対象になるのは，アプリケーションサーバです。

また，N:1 リカバリシステムを構成する場合，実行系で使用するリソースアダプタが，待機系にも必要です。ここでは，リソースアダプタの配置とシステム構成の関係について，次の 3 種類の構成を例として説明します。

- N 台の実行系マシンで，J2EE アプリケーションとリソースアダプタの構成がすべて同じ場合
- N 台の実行系マシンで，J2EE アプリケーションの構成は異なり，リソースアダプタはすべて同一の場合
- N 台の実行系マシンで，J2EE アプリケーションとリソースアダプタの構成がすべて異なる場合

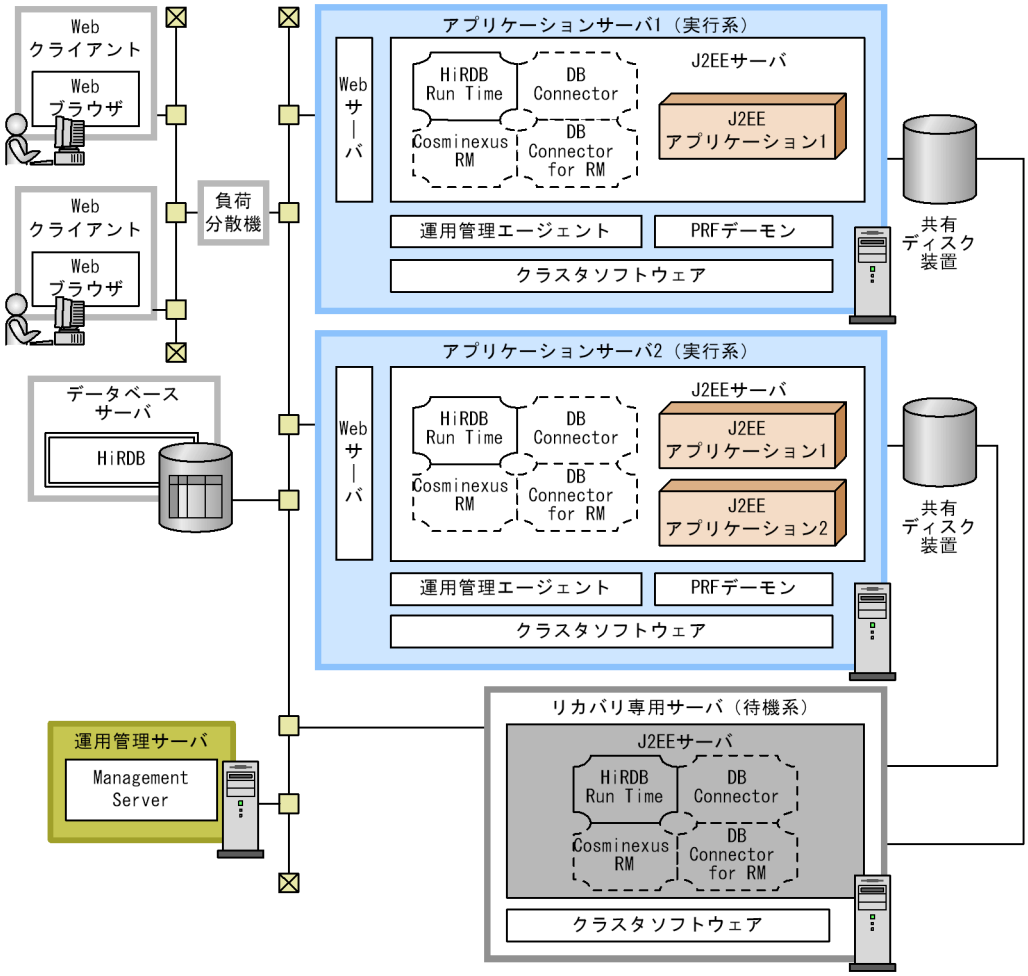
次にそれぞれの構成の例を示します。

図 3-56　N:1 リカバリシステムのシステム構成の例（N 台の実行系マシンで，J2EE アプリケーションとリソースアダプタの構成がすべて同じ場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

図 3-57　N:1 リカバリシステムのシステム構成の例（N 台の実行系マシンで，J2EE アプリケーションの構成は異なり，リソースアダプタはすべて同一の場合）



（凡例）

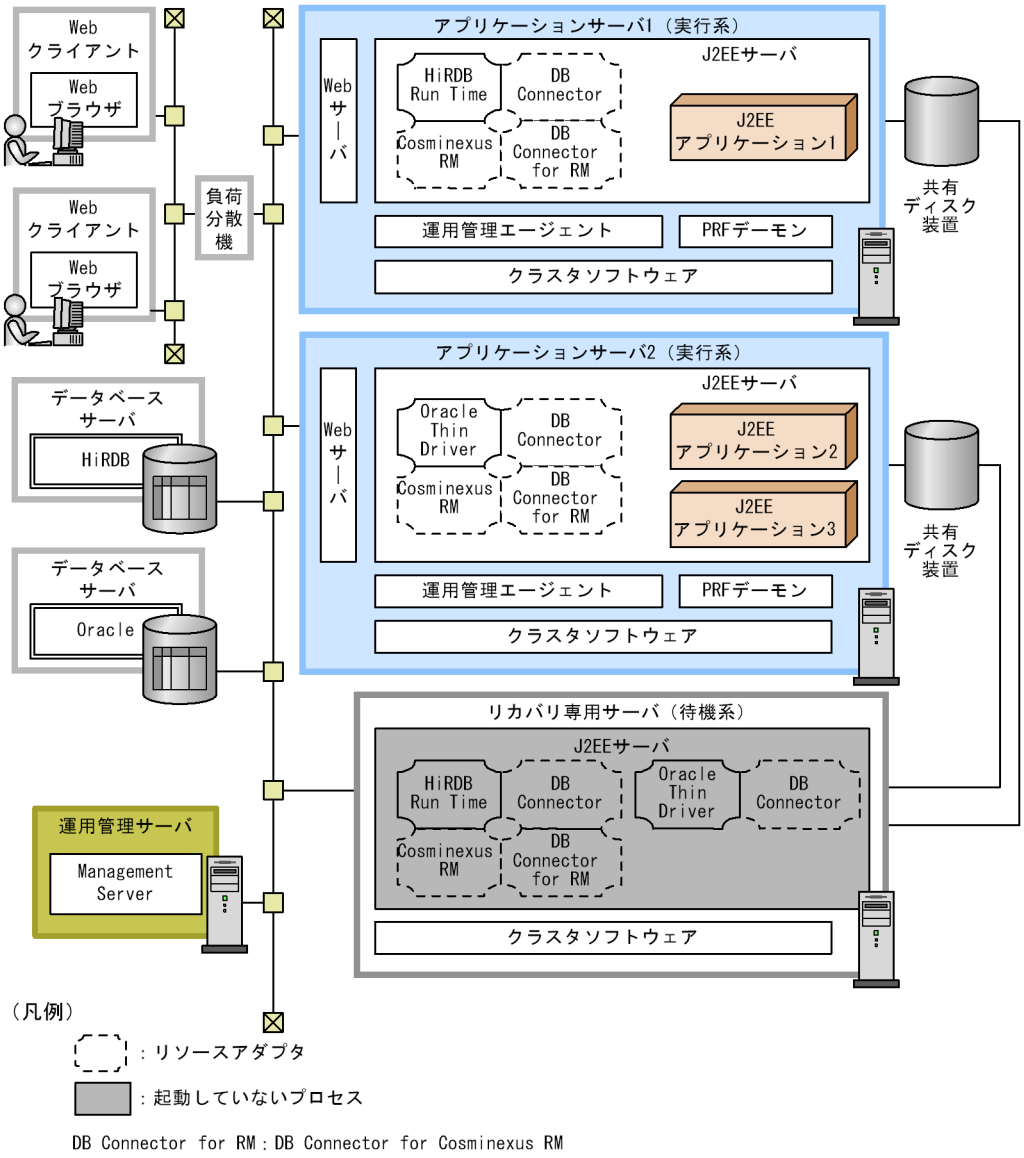
：リソースアダプタ

：起動していないプロセス

DB Connector for RM：DB Connector for Cosminexus RM

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

図 3-58　N:1 リカバリシステムのシステム構成の例（N 台の実行系マシンで，J2EE アプリケーションとリソースアダプタの構成がすべて異なる場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- 実行系と待機系は N：1 で構成します。
- 共有ディスク装置が必要です。共有ディスク装置には実行系の個数（N 個）分のボリュームグループが必要です。
- CORBA ネーミングサービスはインプロセスで起動します。

- 待機系には，実行系と同じリソースアダプタが必要です。実行系の J2EE サーバごとに異なるリソースアダプタがインポートされている場合，待機系の J2EE サーバにはそれらすべてをインポートする必要があります。
  なお，待機系の J2EE サーバに J2EE アプリケーションは不要です。
- データベースがクラスタ構成になっている場合でも，アプリケーションサーバでは仮想アドレス（論理アドレス）だけを認識していれば，接続できます。

系切り替えの流れ

実行系のアプリケーションサーバで起動されている J2EE サーバのうち，どれかの J2EE サーバにトラブルが発生すると，クラスタソフトウェアによってリカバリ専用サーバの J2EE サーバが起動され，トラブルが発生した J2EE サーバが使用していたトランザクションが決着されます。そのあとで，トラブルが発生した J2EE サーバマシンのクラスタソフトウェアと，それに対応する待機系のクラスタサービスが停止されます。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン（実行系）

実行系のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

クラスタソフトウェアを使用する場合に必ず起動するプロセスは次のとおりです。

● 運用管理エージェント

これ以外にアプリケーションサーバに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

(b) アプリケーションサーバマシン（待機系）

待機系のアプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

また，実行系のアプリケーションサーバマシン上の J2EE サーバにインポートされているすべてのリソースアダプタをインポートしてください。

なお，待機系のアプリケーションサーバマシンは，コールドスタンバイの状態になります。

系切り替えが発生するまで，プロセスは起動しません。系切り替えが発生すると，J2EE サーバのプロセスが，リカバリモードで起動されます。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server

Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

（d）クライアントマシン

クライアントマシンには，次のソフトウェアをインストールしてください。

Web クライアント構成の場合

Web ブラウザ

EJB クライアント構成の場合

uCosminexus Client（Windows の場合），Application Server Standard または Application Server Enterprise

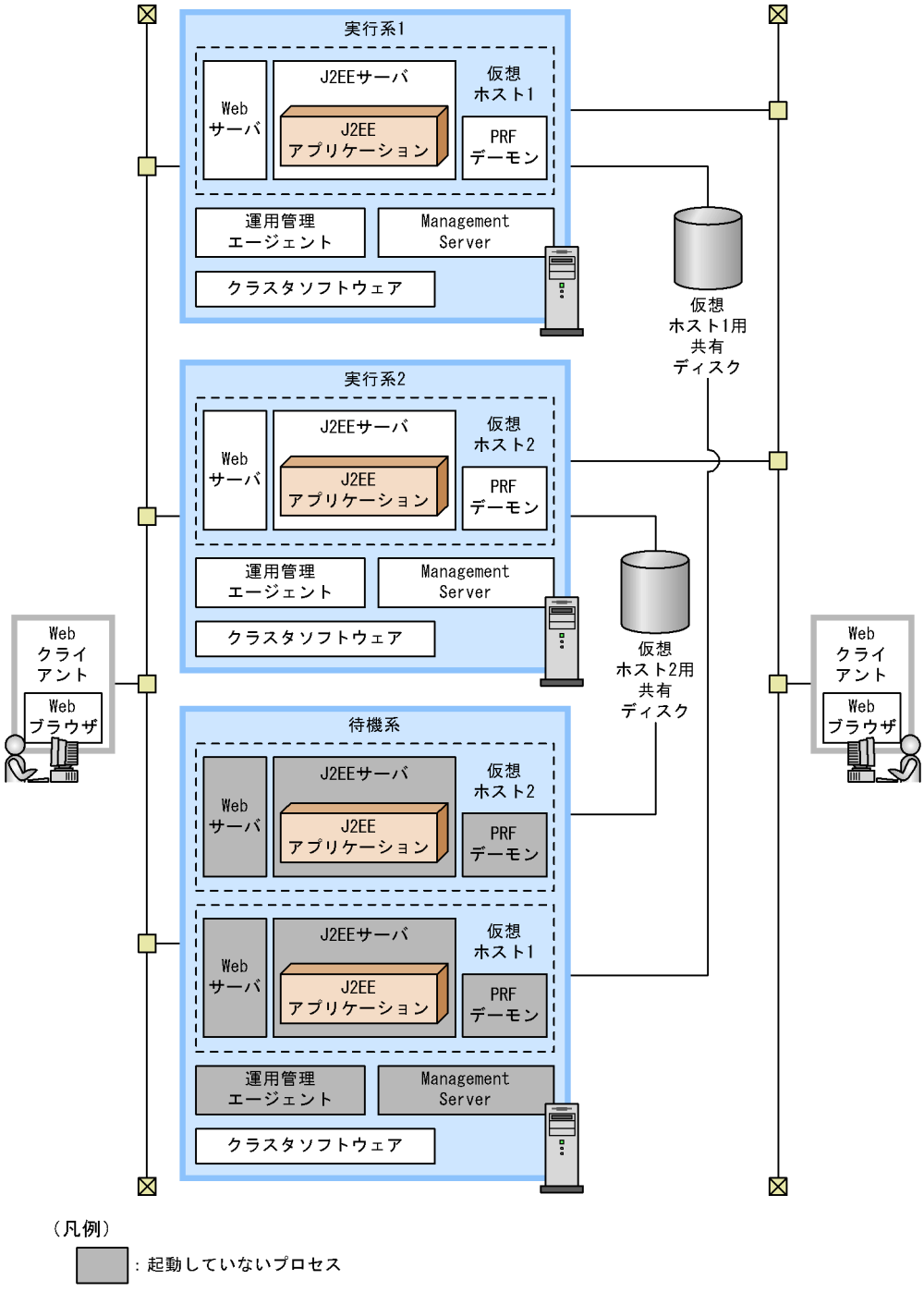
3.11.6 ホスト単位管理モデルの実行系と予備系を N：1 にする構成

ホスト単位管理モデルを系切り替えの対象にする場合の構成について説明します。

（1）システム構成の特徴

ホスト単位管理モデルを N：1 で構成します。システム構成の例を次に示します。

図 3-59　クラスタソフトウェアを使用して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のシステム構成例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションサーバの実行系 N 台と予備系 1 台を配置し，それぞれに Management Server と運用管理エージェントを配置している構成です。
- 論理サーバは別々のホスト内に配置されているが，外部からは同じ論理サーバとして動作するのに対し，Management Server は別々のホスト内で動作し，それぞれ運用管理ドメインを持ちます。一つの運用管理ドメイン内に一つの仮想ホストを定義し，それぞれを現用系アプリケーションサーバのホスト，予備系アプリケーションサーバのホストとして構築します。
- 各アプリケーションサーバの運用に使用する IP アドレスは，クラスタソフトウェアによって割り当てられる仮想 IP アドレスを使用し，Management Server，運用管理エージェントの IP アドレスには実 IP アドレスを使用します。
- グローバルトランザクションを使用する場合は，共有ディスク装置が必要です。共有ディスクは，仮想ホストごとに必要です。

参考

仮想ホストとは，運用管理エージェントによってアプリケーションサーバの起動，停止を制御しますが，運用に使用する IP アドレスと同じ IP アドレスを割り当て，見かけ上同じホストであるかのように定義したものです。

系切り替えの流れ

実行系のアプリケーションサーバマシンに障害が発生すると，系切り替えが実行されます。系切り替えは，Management Server，運用管理エージェント，またはクラスタソフトウェアなどに障害が発生した場合に実行されます。
系切り替えが実行されると，待機系のアプリケーションサーバマシンのクラスタソフトウェアによって運用管理エージェントと Management Server が起動されます。起動された運用管理エージェントによって，それまで停止していた論理サーバのプロセスが起動されます。

(2) それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン（実行系／待機系）

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

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のアプリケーションサーバマシンで必ず起動するプロセスは次のとおりです。

● 運用管理エージェント

● Management Server

これ以外にアプリケーションサーバに必要なソフトウェアおよびプロセスは，使用する

機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

(b) クライアントマシン

クライアントマシンには，次のソフトウェアをインストールしてください。

Web クライアント構成の場合

Web ブラウザ

EJB クライアント構成の場合

uCosminexus Client（Windows の場合），Application Server Standard または Application Server Enterprise

3.12 ファイアウォールを使用する構成を検討する

この節では，ファイアウォールを使用して，セキュリティを確保するための構成について説明します。

なお，ここでは，アクセスポイントになるコンポーネントの種類に応じたファイアウォールの配置位置について説明します。このほかのセキュリティの考え方については，「10. セキュアなシステムの検討」を参照してください。

3.12.1 サーブレットと JSP に対するファイアウォールの配置

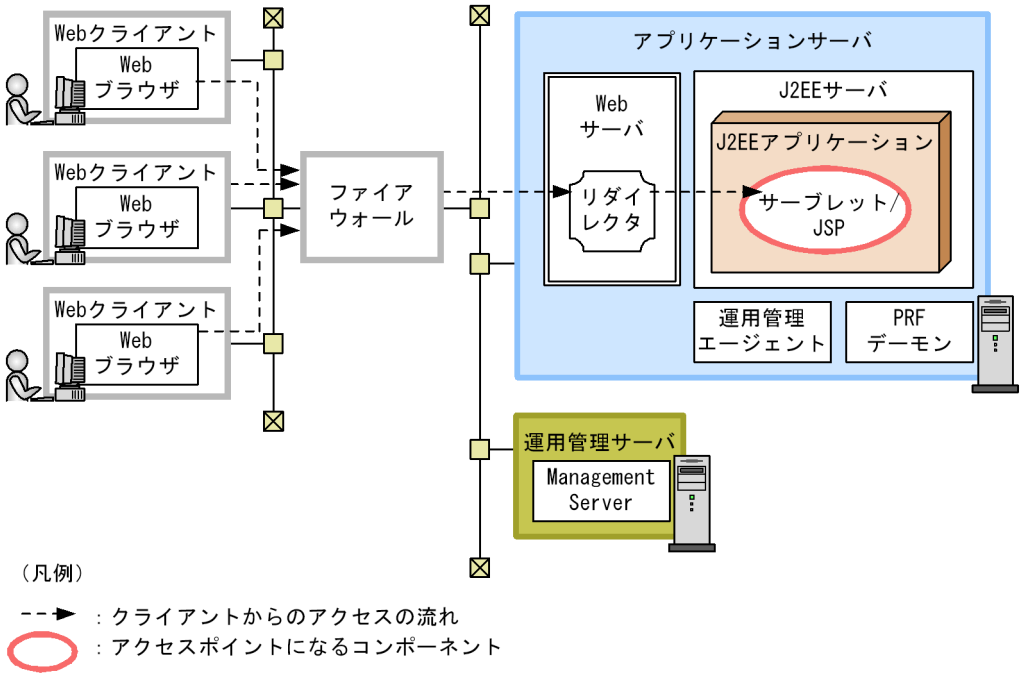
サーブレットと JSP に対するアクセスをファイアウォール経由にする構成について説明します。

（1）システム構成の特徴

Web クライアント側から見たサーブレットと JSP の手前に，ファイアウォールを配置します。

サーブレットと JSP に対するアクセスをファイアウォール経由にする構成の例を次の図に示します。なお，この図は，Web サーバ連携の場合の例です。

図 3-60　サーブレットと JSP に対するアクセスをファイアウォール経由にする構成



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

サーブレットと JSP に対するアクセスをファイアウォール経由にすることで，システムへの第三者の不正侵入，アプリケーションで扱うデータの漏洩，および第三者による不正な運用を防ぎます。

クライアントからのアクセスの流れ

サーブレットと JSP に対するクライアントからのアクセスは，すべてファイアウォール経由で実行されます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

ファイアウォールを使用する場合にアプリケーションサーバマシンおよびクライアントマシンに必要なソフトウェアと起動するプロセスは，サーブレットと JSP をアクセスポイントにする構成と同じです。「3.4.1　サーブレットと JSP をアクセスポイントに使用する構成（Web サーバ連携の場合）」または「3.4.2　サーブレットと JSP をアクセスポイントに使用する構成（インプロセス HTTP サーバを使用する場合）」を参照してください。

3.12.2　Session Bean と Entity Bean に対するファイアウォールの配置

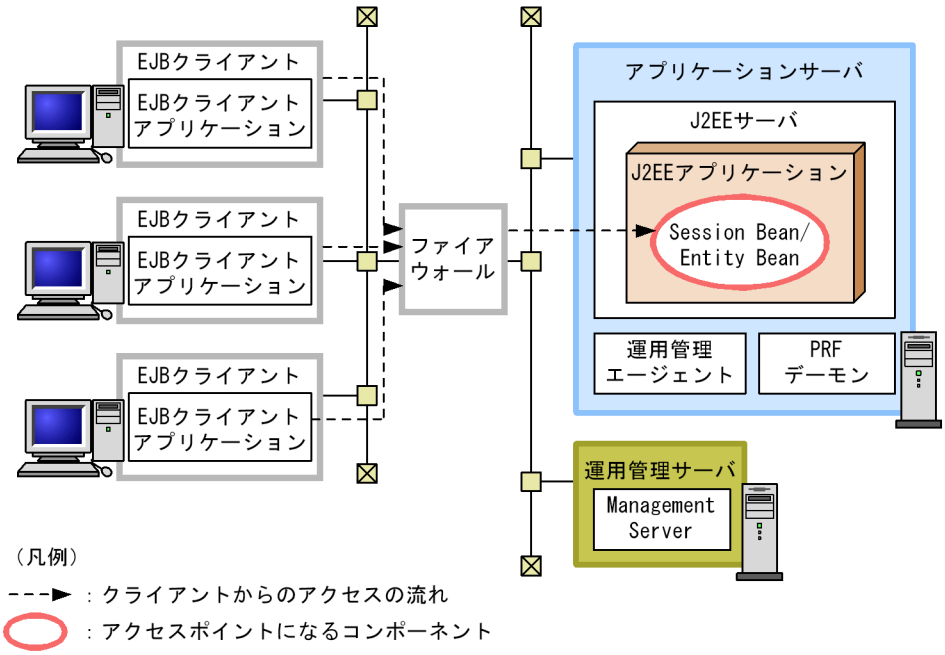
Session Bean と Entity Bean に対するアクセスをファイアウォール経由にする構成です。

（1）システム構成の特徴

EJB クライアント側から見た Session Bean と Entity Bean の手前に，ファイアウォールを配置します。

Session Bean と Entity Bean に対するアクセスをファイアウォール経由にする構成の例を次の図に示します。

図 3-61　Session Bean と Entity Bean に対するアクセスをファイアウォール経由にする構成



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

Session Bean と Entity Bean に対するアクセスをファイアウォール経由にすることで，システムへの第三者の不正侵入，アプリケーションで扱うデータの漏洩，および第三者による不正な運用を防ぎます。

クライアントからのアクセスの流れ

Session Bean と Entity Bean に対する EJB クライアントからのアクセスは，すべ

てファイアウォール経由で実行されます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

ファイアウォールを使用する場合にアプリケーションサーバマシンおよびクライアントマシンに必要なソフトウェアと起動するプロセスは，Session Bean と Entity Bean をアクセスポイントにする構成と同じです。「3.4.3　Session Bean と Entity Bean をアクセスポイントに使用する構成」を参照してください。

3.12.3　リソースマネジャに対するファイアウォールの配置

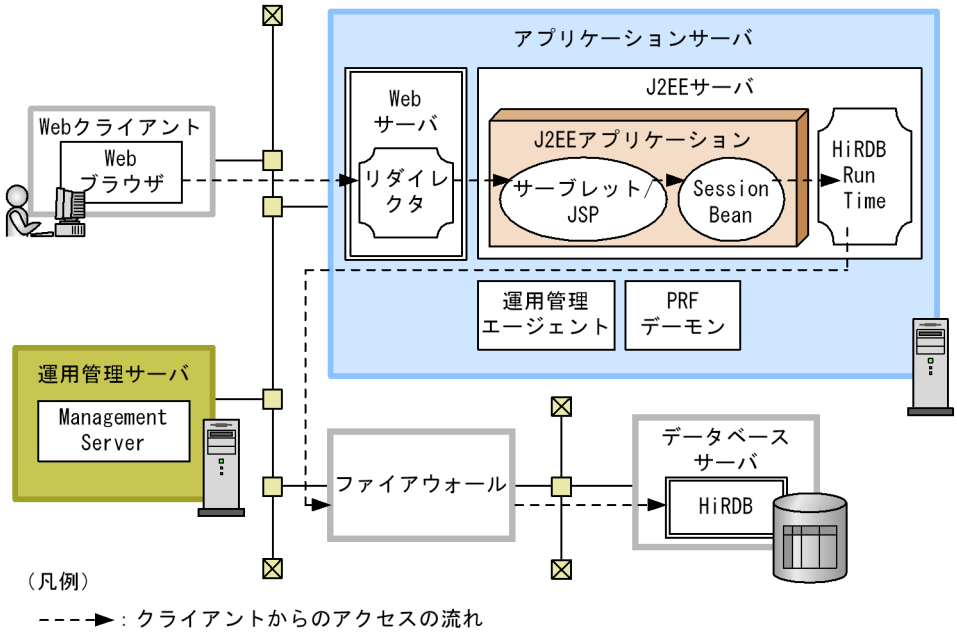
リソースマネジャに対するアクセスをファイアウォール経由にする構成です。

(1) システム構成の特徴

アプリケーション側から見たリソースマネジャの手前に，ファイアウォールを配置します。

リソースマネジャに対するアクセスをファイアウォール経由にする構成の例を次の図に示します。

図 3-62　リソースマネジャに対するアクセスをファイアウォール経由にする構成



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

リソースマネジャに対するアクセスをファイアウォール経由にすることで，システムへの第三者の不正侵入，リソースマネジャで管理するデータの漏洩，および第三

者による不正な運用を防ぎます。

クライアントからのアクセスの流れ

クライアントマシン上の Web ブラウザからのリクエストは，Web サーバ経由でサーブレットと JSP に送信されます。サーブレットと JSP は，ローカルで Session Bean を呼び出します。Session Bean からデータベースへのアクセスが，ファイアウォール経由になります。

（2） それぞれのマシンに必要なソフトウェアと起動するプロセス

トランザクションの使用方法に応じたソフトウェアとプロセスを起動してください。詳細は，「3.6　トランザクションの種類を検討する」を参照してください。

3.13　DMZ へのリバースプロキシの配置を検討する

この節では，DMZ にリバースプロキシを配置してセキュリティを確保するための構成について説明します。

インターネットに接続するシステムの場合，ここで説明する構成を基に，リバースプロキシを配置する構成を検討してください。

なお，ここでは，使用する Web サーバの種類に応じた，リバースプロキシの配置について説明します。このほかのセキュリティの考え方については，「10.　セキュアなシステムの検討」を参照してください。

3.13.1　Web サーバ連携時のリバースプロキシの配置

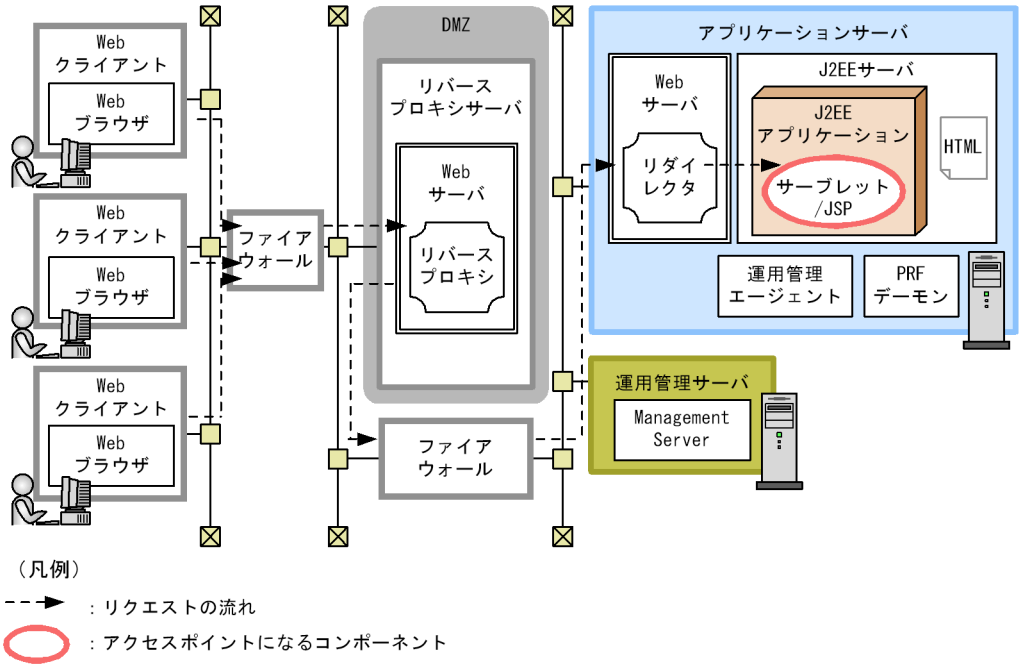
Web サーバとの連携時に，リバースプロキシを配置する構成について説明します。

（1）システム構成の特徴

Web ブラウザとアプリケーションサーバの間に DMZ を確保して，リバースプロキシサーバを配置する構成です。

Web サーバ連携時に DMZ にリバースプロキシを配置する構成の例を次の図に示します。

図 3-63　Web サーバ連携時に DMZ にリバースプロキシを配置する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションサーバへのアクセスはリバースプロキシサーバからだけになり，Web ブラウザからアプリケーションサーバに直接アクセスされることを抑止できます。
- 通常，リバースプロキシ上には HTML などの静的コンテンツは配置しません。

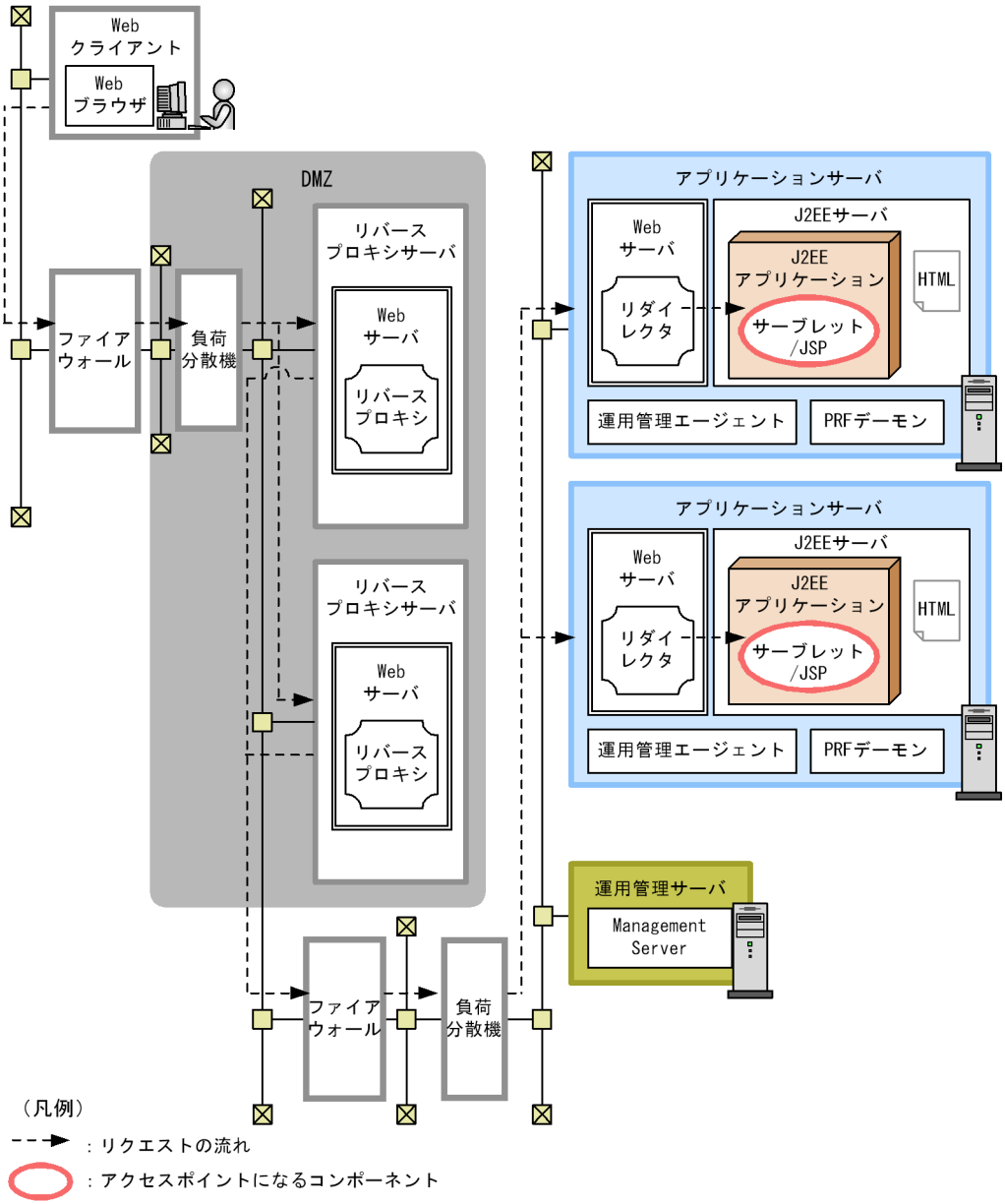
リクエストの流れ

サーブレットと JSP に対するクライアントからのアクセスは，リバースプロキシモジュールを組み込んだ Web サーバ，およびリダイレクタモジュールを組み込んだ Web サーバ経由で実行されます。

また，ロードバランスクラスタを使用して負荷分散をする場合は，リバースプロキシサーバおよびアプリケーションサーバそれぞれに対して負荷分散機（レイヤ 5 スイッチ）を使用して負荷分散を実行できます。

ロードバランスクラスタ構成の場合に，DMZ にリバースプロキシを配置する構成の例を次の図に示します。

図 3-64　Web サーバ連携時に DMZ にリバースプロキシを配置する構成の例（ロードバランスクラスタ構成の場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションサーバへのアクセスはリバースプロキシサーバからだけになり，Web ブラウザからアプリケーションサーバに直接アクセスされることを抑止できます。

- 通常，リバースプロキシ上には HTML などの静的コンテンツは配置しません。
- リバースプロキシサーバおよびアプリケーションサーバへの負荷を分散してスケーラビリティと可用性を確保できます。

リクエストの流れ

サーブレットと JSP に対するクライアントからのアクセスは，一つ目の負荷分散機，リバースプロキシモジュールを組み込んだ Web サーバ，二つ目の負荷分散機およびリダイレクタモジュールを組み込んだ Web サーバ経由で実行されます。
このとき，一つ目の負荷分散機は，Web ブラウザからのアクセスを，複数のリバースプロキシサーバに対して振り分けて負荷を分散します。二つ目の負荷分散機は，リバースプロキシサーバからのアクセスを，複数のアプリケーションサーバに対して振り分けて負荷を分散します。二つ目の負荷分散機では，HTTP セッションのスティッキー（Sticky）やアフィニティ（Affinity）の関連づけも制御します。

なお，HTTPS を使用する場合は，一つ目の負荷分散機のフロントに SSL アクセラレータを配置する必要があります。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) リバースプロキシサーバマシン

リバースプロキシサーバマシンには，Hitachi Web Server をインストールします。

必ず起動するプロセスは次のとおりです。

● Web サーバ

この Web サーバには，リバースプロキシモジュールが組み込まれている必要があります。

(b) アプリケーションサーバマシン，運用管理サーバマシンおよびクライアントマシン

アプリケーションサーバマシン，運用管理サーバマシンおよびクライアントマシンに必要なソフトウェアと起動するプロセスは，サーブレットと JSP をアクセスポイントにする構成と同じです。「3.4.1　サーブレットと JSP をアクセスポイントに使用する構成（Web サーバ連携の場合）」を参照してください。

3.13.2 インプロセス HTTP サーバ使用時のリバースプロキシの配置

インプロセス HTTP サーバを使用する場合にリバースプロキシを配置する構成について説明します。

インターネットに接続するシステムでインプロセス HTTP サーバを使用する場合は，ここで示す構成を基に，必ず DMZ を確保して，リバースプロキシを配置する構成にしてく

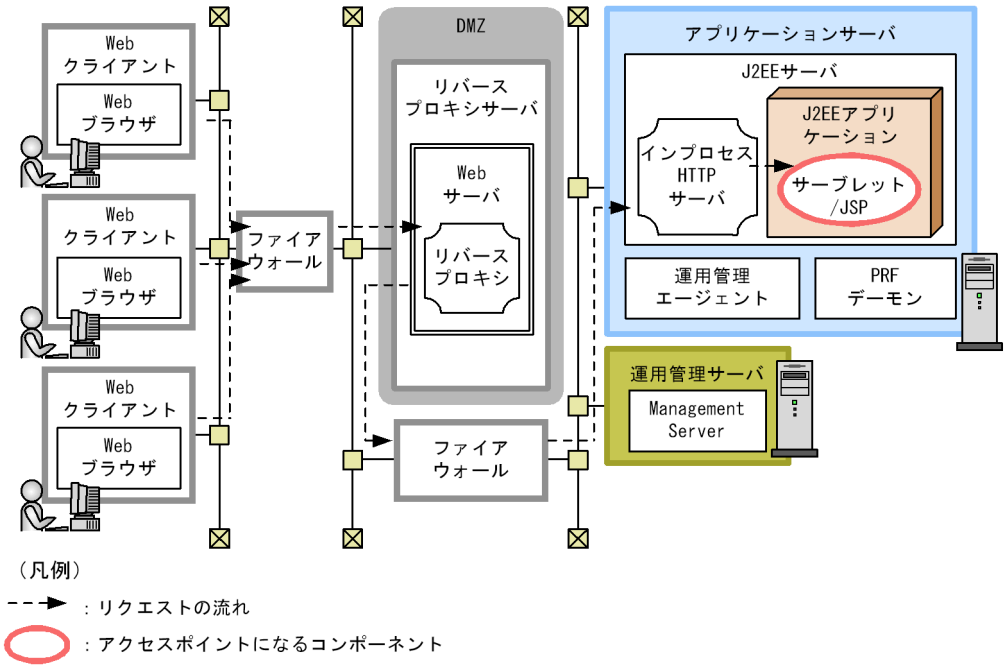
ださい。

（1） システム構成の特徴

Web ブラウザとアプリケーションサーバの間に DMZ を確保して，リバースプロキシサーバを配置する構成です。

インプロセス HTTP サーバを使用する場合に DMZ にリバースプロキシを配置する構成の例を次の図に示します。

図 3-65　インプロセス HTTP サーバを使用する場合に DMZ にリバースプロキシを配置する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションサーバへのアクセスはリバースプロキシサーバからだけになり，Web ブラウザからアプリケーションサーバに直接アクセスされることを抑止できます。
- 通常，リバースプロキシ上には HTML などの静的コンテンツは配置しません。

リクエストの流れ

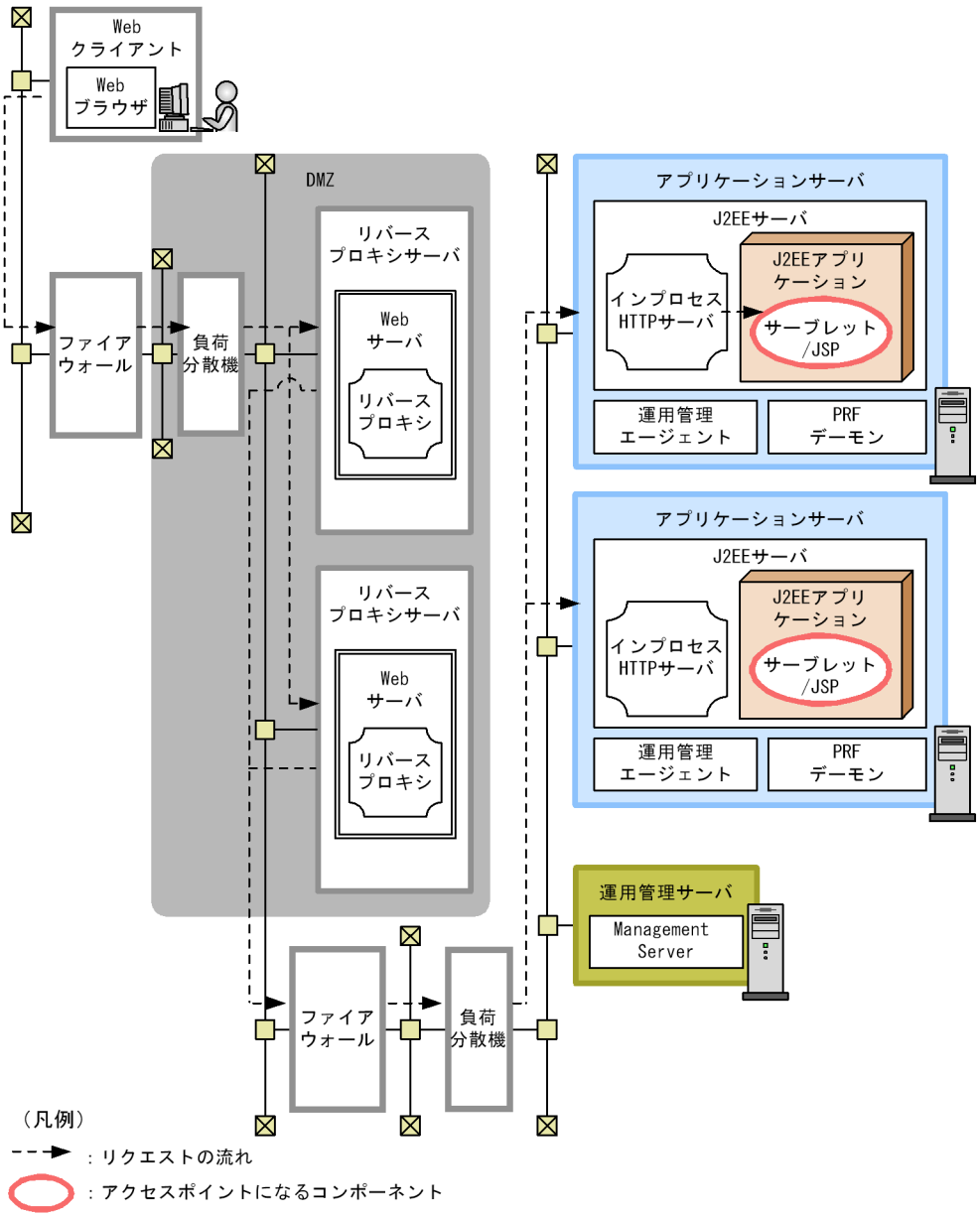
サーブレットと JSP に対するクライアントからのアクセスは，リバースプロキシモジュールを組み込んだ Web サーバ経由で実行されます。

また，ロードバランスクラスタを使用して負荷分散をする場合は，リバースプロキシ

サーバおよびアプリケーションサーバそれぞれに対して負荷分散機（レイヤ 5 スイッチ）を使用して負荷分散を実行できます。

ロードバランスクラスタ構成の場合に，DMZ にリバースプロキシを配置する構成の例を次の図に示します。

図 3-66　インプロセス HTTP サーバを使用する場合に DMZ にリバースプロキシを配置する構成の例（ロードバランスクラスタ構成の場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- アプリケーションサーバへのアクセスはリバースプロキシサーバだけからになり，Web ブラウザからアプリケーションサーバに直接アクセスされることを抑止できます。
- 通常，リバースプロキシ上には HTML などの静的コンテンツは配置しません。
- リバースプロキシサーバおよびアプリケーションサーバへの負荷を分散してスケーラビリティと可用性を確保できます。

リクエストの流れ

サーブレットと JSP に対するクライアントからのアクセスは，一つ目の負荷分散機，リバースプロキシモジュールを組み込んだ Web サーバ，および二つ目の負荷分散機経由で実行されます。

このとき，一つ目の負荷分散機は，Web ブラウザからのアクセスを，複数のリバースプロキシサーバに対して振り分けて負荷を分散します。二つ目の負荷分散機は，リバースプロキシサーバからのアクセスを，複数のアプリケーションサーバに対して振り分けて負荷を分散します。二つ目の負荷分散機では，HTTP セッションのスティッキー（Sticky）やアフィニティ（Afinity）の関連づけも制御します。

なお，HTTPS を使用する場合は，一つ目の負荷分散機のフロントに SSL アクセラレータを配置する必要があります。

（2）それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) リバースプロキシサーバマシン

リバースプロキシサーバマシンには，Hitachi Web Server をインストールします。

必ず起動するプロセスは次のとおりです。

● Web サーバ

この Web サーバには，リバースプロキシモジュールが組み込まれている必要があります。

(b) アプリケーションサーバマシン，運用管理サーバマシンおよびクライアントマシン

アプリケーションサーバマシン，運用管理サーバマシンおよびクライアントマシンに必要なソフトウェアと起動するプロセスは，サーブレットと JSP をアクセスポイントにする構成と同じです。「3.4.2　サーブレットと JSP をアクセスポイントに使用する構成（インプロセス HTTP サーバを使用する場合）」を参照してください。

3.14　性能解析トレースファイルを出力するプロセスを配置する

この節では，性能解析トレースファイルを出力するプログラムの配置について説明します。このプロセスは，アプリケーションサーバのシステムには必ず配置するプロセスです。アプリケーションサーバマシンには必ず配置してください。EJB クライアントマシンへの配置は任意です。

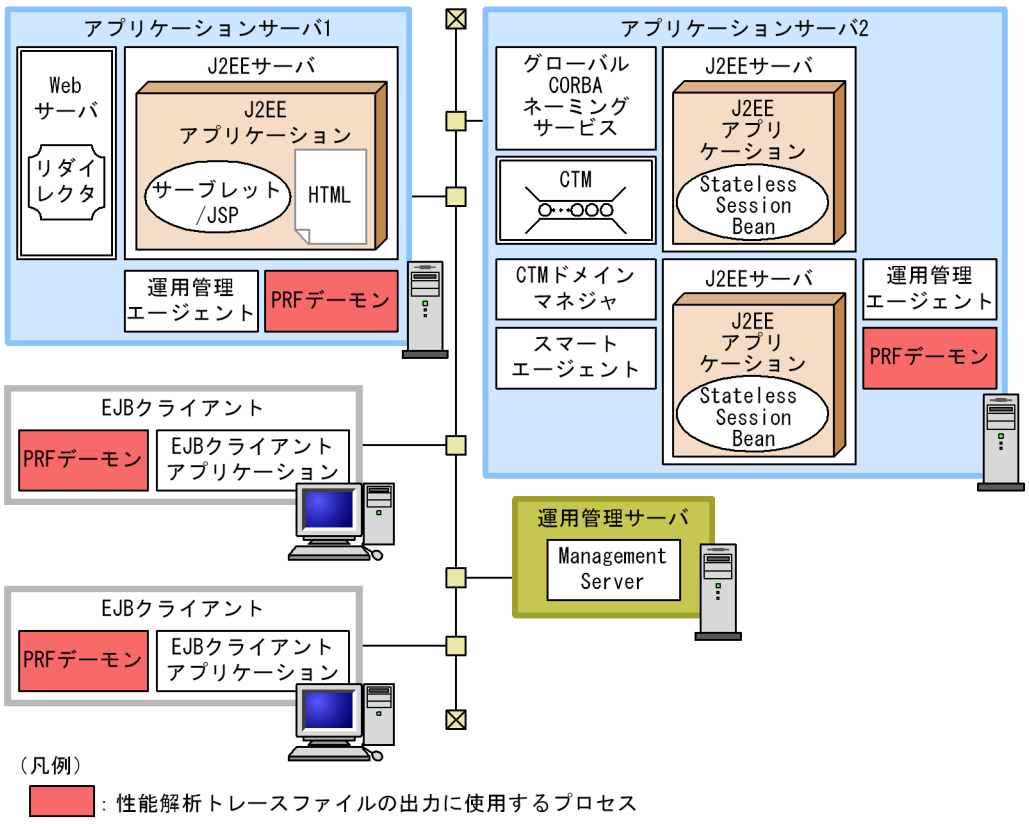
性能解析トレースファイルは，PRF デーモン（パフォーマンストレーサ）によって出力されます。

(1) システム構成の特徴

PRF デーモンは，アプリケーションサーバおよび EJB クライアントアプリケーションを実行するマシンに配置します。

PRF デーモンを配置する例を次の図に示します。

図 3-67　PRF デーモンを配置する例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- システム性能を解析するためのトレースファイルを出力できます。
- トレースファイルは，トラブルが発生した場合のエラー個所の特定に利用できます。

性能解析トレースを出力する仕組み

クライアントとサーバ間でリクエストを処理するときに，アプリケーションサーバまたは EJB クライアントアプリケーションの各機能レイヤは，性能解析情報をバッファに出力します。PRF デーモンは，アプリケーションサーバまたは EJB クライアントアプリケーションの各機能レイヤがバッファに出力したトレース情報を，各マシン内のファイルに出力します。ファイル出力されたトレース情報は，Management Server を利用して収集できます。

（2）それぞれのマシンに必要なソフトウェアと起動するプロセス

性能解析トレースファイルを出力するために起動するプロセスは次のとおりです。

● PRF デーモン

これ以外にアプリケーションサーバに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェアとプロセスを配置してください。

3.15 アプリケーションサーバ以外の製品との連携を検討する

この節では，Cosminexus のアプリケーションサーバ以外の製品と連携する場合のシステム構成について説明します。

ここでは，次の構成について説明します。

- JP1 を使用して運用する場合の構成
  運用管理プログラムとして JP1 を使用する場合の構成です。
- TP1 インバウンド連携機能を使用して OpenTP1 の SUP から Message-driven Bean を呼び出す場合の構成
  TP1 インバウンド連携機能を使用して OpenTP1 の SUP からアプリケーションサーバ上の Message-driven Bean を呼び出す場合の構成です。
- CTM ゲートウェイ機能を利用して EJB クライアント以外から Stateless Session Bean を呼び出す構成
  クライアントとして TPBroker クライアントまたは TPBroker OTM クライアントを使用する場合の構成です。

3.15.1 JP1 を使用して運用する場合の構成

JP1 を使用して運用する場合の運用管理プログラムの配置について説明します。

JP1/IM と連携する場合，またはカスタムジョブの定義やシナリオなどアプリケーションサーバが提供する機能を利用して JP1/AJS と連携する場合には，Management Server を利用して運用していることが前提になります。「3.9.1　運用管理サーバに Management Server を配置する構成」または「3.9.2　マシン単位に Management Server を配置する構成」を参照して Management Server を利用するシステムの構成を決定した上で，JP1 のプログラムを配置してください。JP1 との連携で必要なプログラムについては，マニュアル「Cosminexus 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。

なお，JP1/AJS または JP1/AJS2 - SO を利用する場合に，アプリケーションサーバの機能を使用しないでジョブまたはシナリオの定義をするときには，Management Server を利用しなくても連携できます。この場合は，「3.9.3　コマンドで運用する場合の構成」に示すシステムの構成を検討した上で，JP1 のマニュアルを参照して，システム構成を決定してください。

3.15.2　TP1 インバウンド連携機能を使用して OpenTP1 の SUP から Message-driven Bean を呼び出す場合の構成

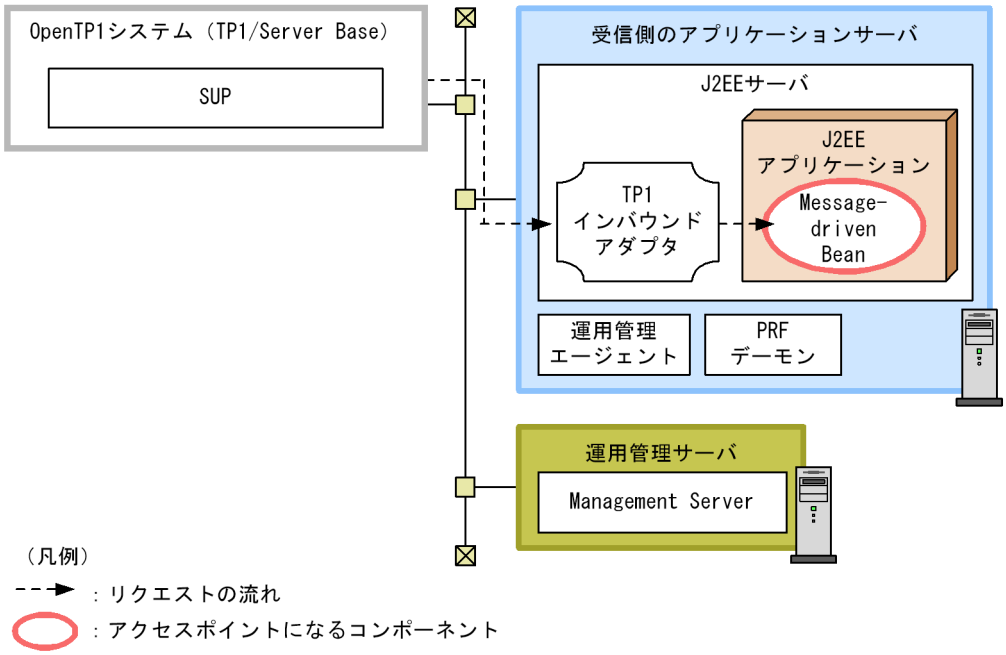
TP1 インバウンド連携機能を使用して，OpenTP1 システムの SUP から J2EE サーバ上の Message-driven Bean を呼び出す構成について説明します。

（1）システム構成の特徴

メッセージ駆動型のシステムの一つです。

TP1 インバウンド連携機能を使用する場合のメッセージ駆動型のシステム構成の例を次の図に示します。

図 3-68　メッセージ駆動型のシステム構成の例（TP1 インバウンド連携機能を使用する場合）



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

TP1/Server Base 上で動作する SUP から送信されたリクエストを J2EE サーバ上で動作する TP1 インバウンドアダプタで受け付けて，処理を実行します。SUP からは，SPP を呼び出すときと同様の手順※で，J2EE サーバ上の Message-driven Bean にアクセスできます。

注※

一部の手順は異なります。詳細は，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (コンテナ共通機能)」の「4. OpenTP1 からのアプリケーションサーバの呼び出し（TP1 インバウンド連携機能）」を参照してください。

リクエストの流れ

アクセスポイントである Message-driven Bean，およびリソースアダプタである TP1 インバウンドアダプタのライブラリは，アプリケーションサーバの J2EE サーバ上で動作します。
OpenTP1 システムの TP1/Server Base 上で動作する SUP からからのリクエストをアプリケーションサーバ上の TP1 インバウンドアダプタで受け付け，Message-driven Bean を呼び出します。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシン（サーバ側のアプリケーションサーバマシン）には，Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- J2EE サーバ
- 運用管理エージェント
- PRF デーモン

(b) OpenTP1 システム（TP1/Server Base）

OpenTP1 システムには，TP1/Server Base をインストールする必要があります。

起動する必要があるプロセスは，SUP の実行に必要なプロセスです。

(c) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

- Management Server

3.15.3 CTM ゲートウェイ機能を利用して EJB クライアント以外から Stateless Session Bean を呼び出す構成

CTM を使用するシステムの場合，EJB クライアントのほか，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**

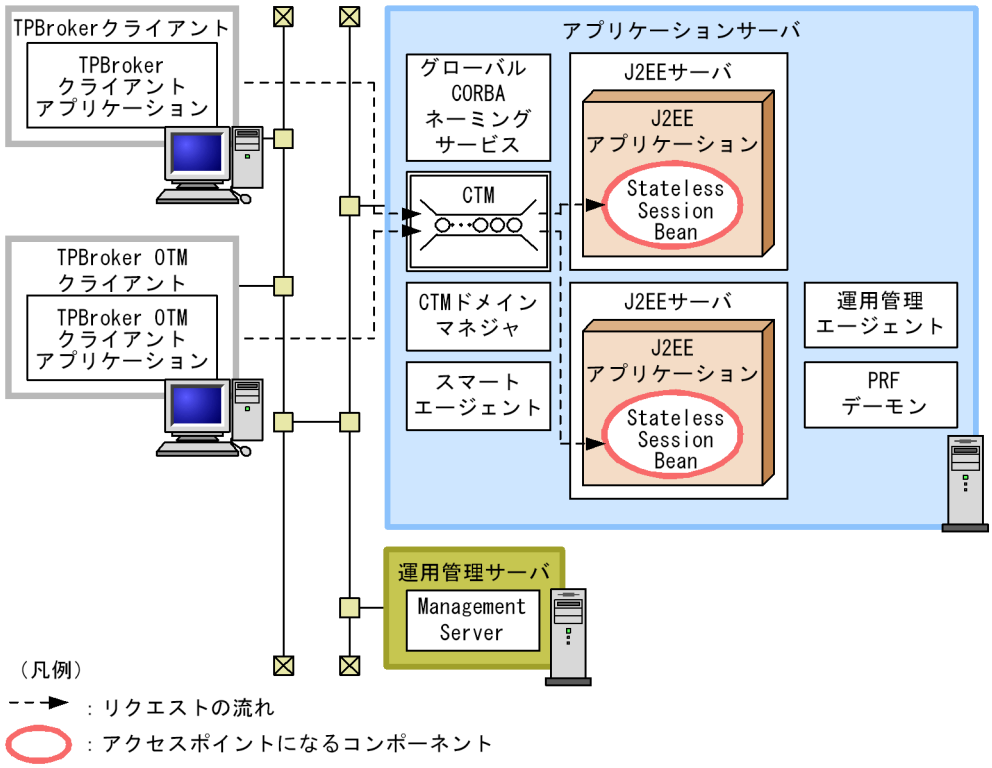
TPBroker Object Transaction Monitor で開発されたクライアントアプリケーションです。

これらのクライアントから 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，CTM の CORBA ／ OTM ゲートウェイ機能についてのドキュメントを参照してください。

この構成では，TPBroker クライアントまたは TPBroker OTM クライアントから J2EE アプリケーションを呼び出すためのゲートウェイとしての機能を，CTM が提供します。

TPBroker クライアントまたは TPBroker OTM クライアントから CTM 経由で J2EE アプリケーションを呼び出すシステム構成の例を次の図に示します。

図 3-69　TPBroker クライアントまたは TPBroker OTM クライアントから CTM 経由で J2EE アプリケーションを呼び出す構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

TPBroker クライアントまたは TPBroker OTM クライアントからのリクエストは，CTM が提供するプロセス群を経由して，J2EE サーバ上の J2EE アプリケーションに渡されます。

なお，CTM のプロセス群のうち，リクエストを受け付けるプロセスは，TPBroker クライアントの場合と TPBroker OTM クライアントの場合で異なります。TPBroker クライアントの場合，CTM レギュレータプロセスによってリクエストが受け付けられます。TPBroker OTM クライアントの場合，OTM ゲートウェイプロセスによってリクエストが受け付けられます。なお，CTM レギュレータプロセス，および OTM ゲートウェイプロセスは，CTM デーモンを起動するときに，同時に起動されるプロセスです。

3.16　任意のプロセスを運用管理の対象にする

この節では，ユーザが定義する任意のプロセスを，Management Server による運用管理の対象とするためのシステム構成について説明します。

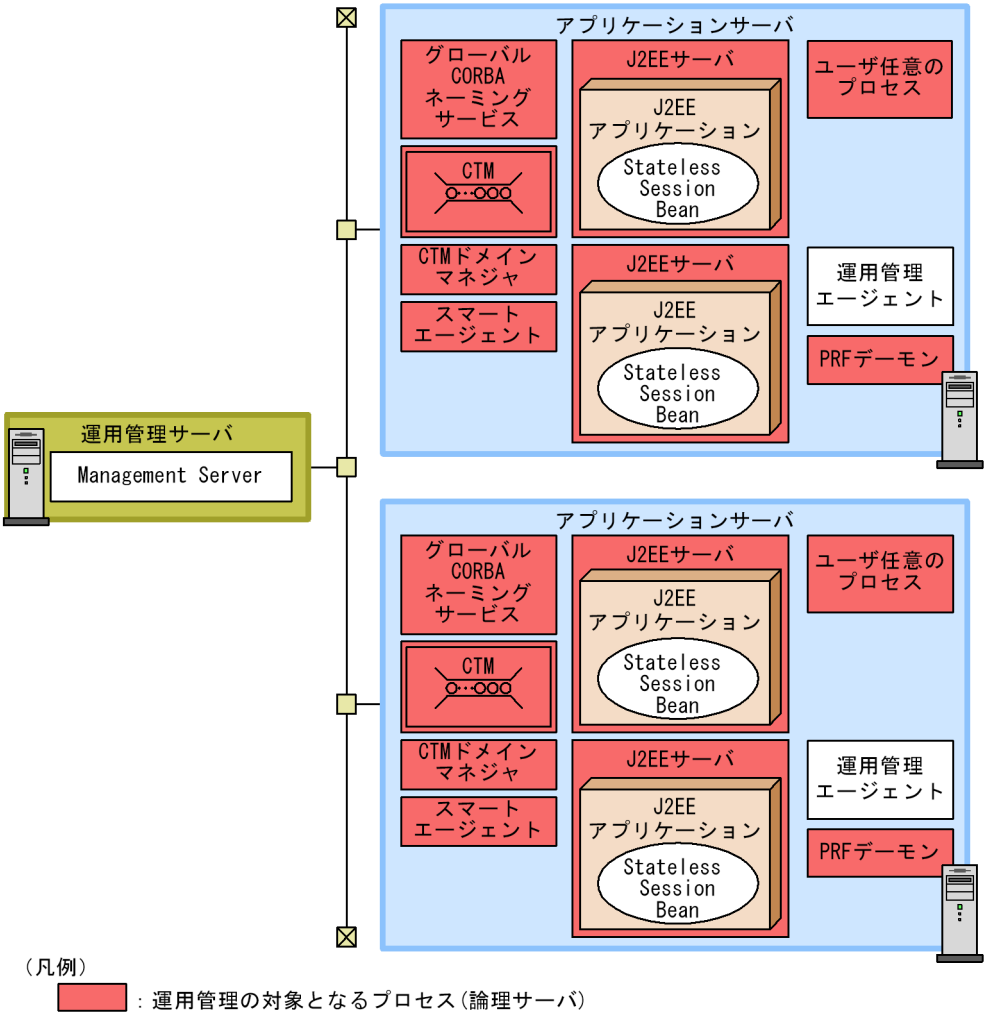
Management Server による運用管理の対象とする任意のプロセスを，論理サーバとして定義したユーザサーバとして配置します。論理サーバとして定義したユーザサーバを**論理ユーザサーバ**といいます。論理ユーザサーバとして定義しておくと，Smart Composer 機能のコマンドを使用して，任意のプロセスを開始したり，停止したり，任意のプロセスのステータスを監視したりできるようになります。

（1）システム構成の特徴

このシステム構成は，Management Server で運用管理するシステムで，論理サーバとして定義したユーザサーバを配置する構成です。運用管理に使用する Management Server は，運用管理サーバに配置することも，マシン単位に配置することもできます。Management Server の配置については，「3.9　運用管理プロセスの配置を検討する」を参照してください。

ユーザサーバを配置する構成の例を次の図に示します。この構成例では，アプリケーションサーバマシンでユーザ任意のプロセスを実行します。このユーザ任意のプロセスを Management Server の運用管理の対象とするために，論理ユーザサーバとして定義して，配置しています。

図 3-70　ユーザサーバを配置する構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- Management Server を利用して任意のプロセスを運用管理できます。

(2) それぞれのマシンで起動するプロセス

任意のプロセスをユーザサーバとして配置する構成の場合に，それぞれのマシンに必要なソフトウェアと起動するプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンに必要なソフトウェアと起動するプロセスは，使用する機能に応じたシステム構成ごとに異なります。使用する機能に応じて必要なソフトウェ

アとプロセスを配置してください。

(b) 運用管理サーバマシン

運用管理サーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

起動するプロセスは次のとおりです。

● Management Server

3.17 そのほかの構成を検討する

この節では，前の節までに説明した以外の構成について説明します。

ポイント

この節で説明する構成は，旧バージョンとの互換性を保つための構成など，07-00 以降のシステムでは推奨されていない互換用の構成です。07-00 以降のアプリケーションサーバで新規にシステムを構成する場合は，前の節までに説明した構成を使用することをお勧めします。
互換用の構成と，同等の機能を実現できる 07-00 以降で推奨されている構成の対応について，次の表に示します。

表 3-3 互換用の構成と推奨構成の対応

互換用の構成	07-00 以降で推奨されている構成	推奨されている構成についての参照先
Web サーバとアプリケーションサーバを異なるマシンに配置する構成	リダイレクタモジュールを組み込んだ Web サーバと J2EE サーバは同じマシンに配置することが推奨されています。	• 3.4.1 サーブレットと JSP をアクセスポイントに使用する構成（Web サーバ連携の場合） • 3.13.1 Web サーバ連携時のリバースプロキシの配置
リダイレクタを利用して負荷分散する構成	サーブレット／ JSP に対する負荷分散には負荷分散機を使用することが推奨されています。	• 3.7.1 Web サーバ連携時の負荷分散機を利用した負荷分散（サーブレット／ JSP の場合）
CORBA ネーミングサービスをアウトプロセスで起動する構成	CORBA ネーミングサービスはインプロセスで起動することが推奨されています。	—※

（凡例）—：該当しない。
注※ 3.4 ～ 3.16 で説明したすべての構成で使用できます。

このほかの互換用の構成については，「付録 A ベーシックモードの利用（互換機能）」および「付録 B サーブレットエンジンモードの利用（互換機能）」を参照してください。

3.17.1 Web サーバとアプリケーションサーバを異なるマシンに配置する構成

サーブレットと JSP がアクセスポイントになる，Web ベースのシステム構成について説明します。ここでは，リダイレクタモジュールを組み込んだ Web サーバとアプリケーションサーバを異なるマシンに配置する場合について説明します。

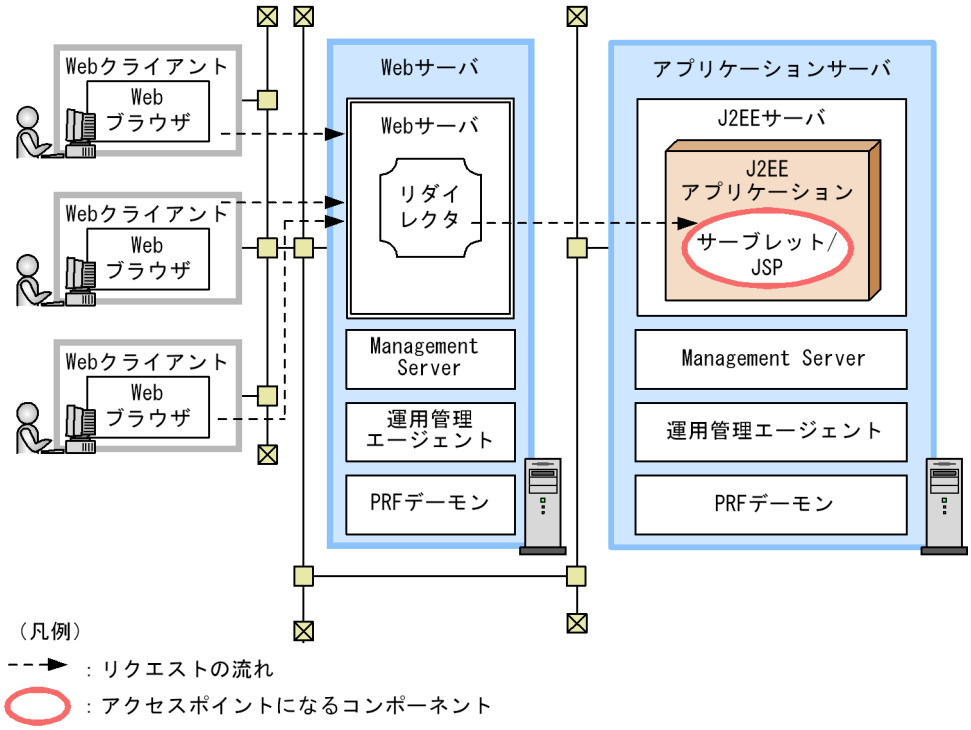
なお，複数の Web リダイレクタ環境を構築する場合は，Management Server が提供する機能を使用して構築してください。

（1） システム構成の特徴

Web フロントシステムで，Web ブラウザから送信されたリクエストをアプリケーションサーバで処理する場合に適用されるシステム構成です。

Web クライアント構成では，Web サーバとアプリケーションサーバを異なるマシンに分けて配置できます。Web サーバとアプリケーションサーバを異なるマシンに配置した Web クライアント構成の例を次の図に示します。

図 3-71　Web サーバとアプリケーションサーバを異なるマシンに配置した Web クライアント構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

HTML ファイルなどの静的なコンテンツを配置するマシンと，サーブレットや JSP などの Web アプリケーションを実行するマシンを分けることで，パフォーマンスの向上が図れます。静的コンテンツと Web アプリケーションを分けて配置する場合の配置方法については，「8.7.1　静的コンテンツと Web アプリケーションの配置を切り分ける」を参照してください。

リクエストの流れ

アクセスポイントであるサーブレットと JSP は，J2EE サーバ上で動作します。Web ブラウザからのリクエストは，Web サーバ経由でアクセスポイントに送られ，サーブレットと JSP を呼び出します。

(2) それぞれのマシンに必要なプロセスとソフトウェア

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。なお，リソースと接続するために必要なプロセスについては，「3.6　トランザクションの種類を検討する」を参照してください。

(a) Web サーバマシン

Web サーバマシンには，次のソフトウェアをインストールする必要があります。

- Application Server Standard
- Application Server Enterprise
- Web Redirector

必要なプロセスは次のとおりです。

- Web サーバ
- Management Server
- 運用管理エージェント
- PRF デーモン

Application Server Standard または Application Server Enterprise には，Web サーバである Hitachi Web Server が含まれています。Windows の場合，Web サーバに Microsoft IIS を使用することもできます。この場合は，ソフトウェアとして Microsoft IIS が必要です。

Web Redirector には，Web サーバは含まれていません。Hitachi Web Server または Microsoft IIS が必要です。

(b) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。

必要なプロセスは次のとおりです。

- J2EE サーバ
- Management Server
- 運用管理エージェント

● PRF デーモン

(c) Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

3.17.2 リダイレクタを利用して負荷分散する構成

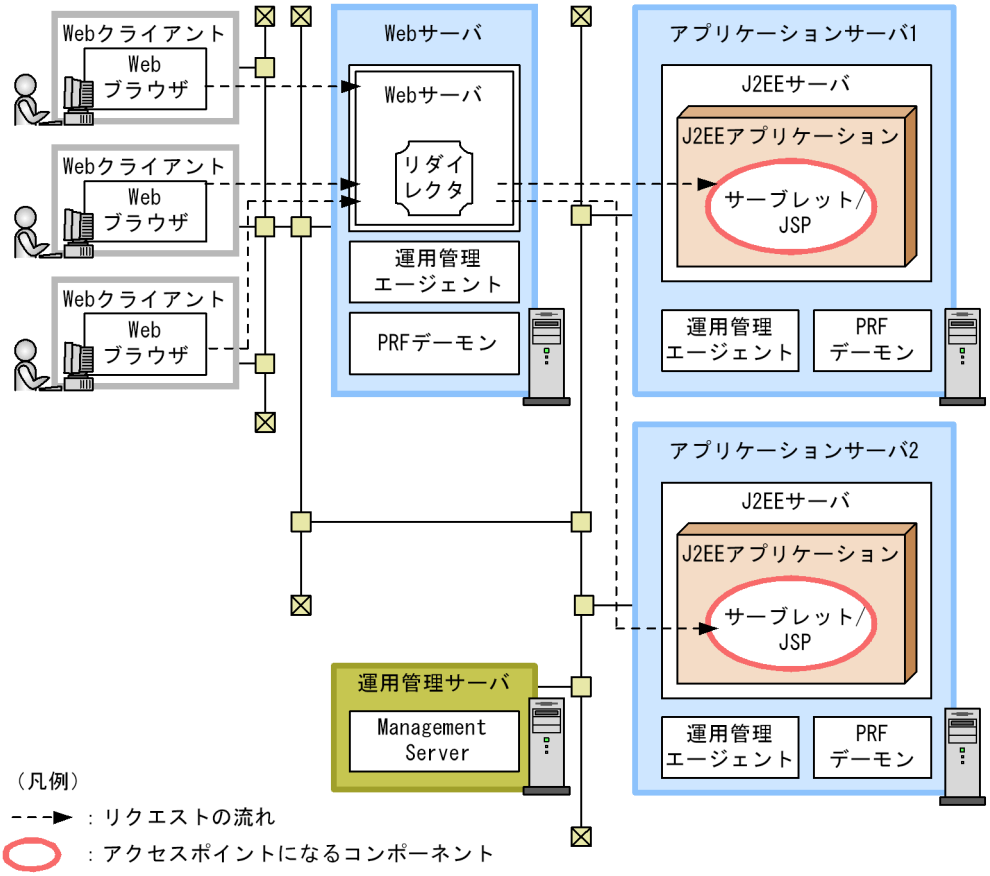
アクセスポイントになるコンポーネントがサーブレットまたはJSP の場合に，Web サーバに登録したリダイレクタによって負荷を分散する構成について説明します。

(1) システム構成の特徴

J2EE サーバ上で動作するアプリケーションのアクセスポイントが，サーブレットまたはJSP の場合に使用できる構成です。Web サーバに登録したリダイレクタで負荷を分散します。リダイレクタの設定ファイルに対象となるアプリケーションサーバを登録することで実現できます。

リダイレクタを利用した負荷分散の構成の例を次の図に示します。

図 3-72　リダイレクタを利用した負荷分散の構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- サーブレットと JSP のスケーラビリティと可用性を確保できます。アプリケーションサーバを複数用意して，リダイレクタによってクライアントからのアクセスを分散することで，負荷を分散できます。
- リダイレクタを登録した Web サーバは，アプリケーションサーバとは異なるマシン上に配置する必要があります。
- 特定のアプリケーションサーバでトラブルが発生した場合，またはメンテナンスが必要な場合に，リダイレクタから該当するアプリケーションサーバへのアクセスを停止することで，システムの縮退運転ができます。

ポイント

負荷分散機を使用する方が，Web サーバを実行するマシンへの負荷を掛けないで，性能の高い負荷分散を実現できます。

リクエストの流れ

アクセスポイントである J2EE サーバ上のサーブレットと JSP へのリクエストは，Web ブラウザから Web サーバ経由で送られます。その際，Web サーバに登録されているリダイレクタは，ロードバランスクラスタの構成要素である J2EE サーバにラウンドロビン方式でリクエストを振り分けます。

(2) それぞれのマシンに必要なソフトウェアと起動するプロセス

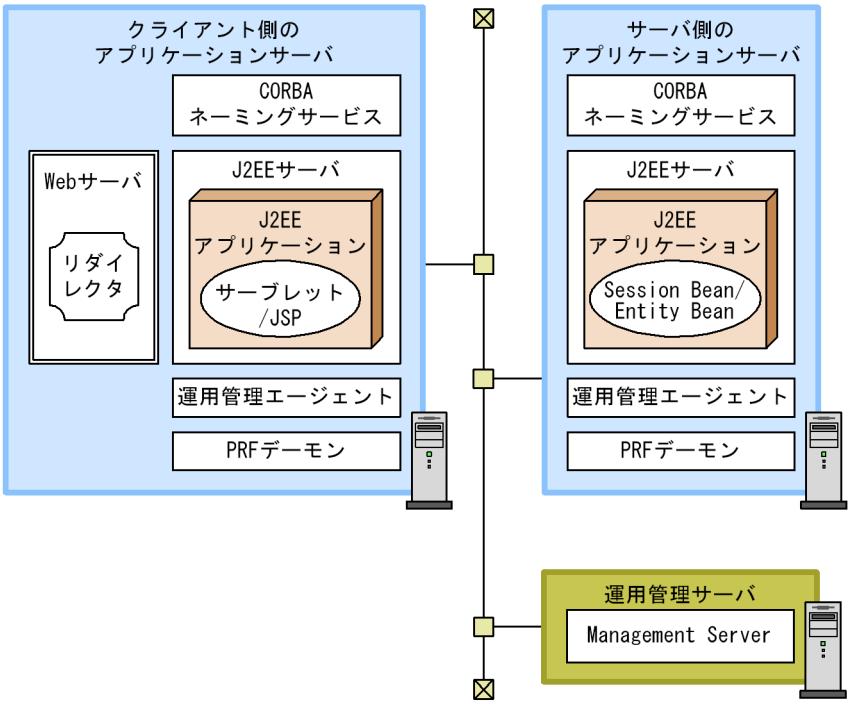
リダイレクタを使用した負荷を分散する場合に必要なソフトウェアと起動するプロセスは，サーブレットと JSP をアクセスポイントにする構成のうち，Web サーバとアプリケーションサーバを分離して配置する構成と同じです。「3.17.1　Web サーバとアプリケーションサーバを異なるマシンに配置する構成」を参照してください。

3.17.3　CORBA ネーミングサービスをアウトプロセスで起動する構成

アプリケーションサーバでは，CORBA ネーミングサービスを J2EE サーバのインプロセスで起動できますが，アウトプロセスでも起動できます。

CORBA ネーミングサービスをアウトプロセスで起動する構成の例を次の図に示します。

図 3-73　CORBA ネーミングサービスをアウトプロセスで起動する構成の例



凡例については，「3.2　システム構成の説明について」を参照してください。

ただし，07-00 以降で新規にシステムを構築する場合は，CORBA ネーミングサービスはインプロセスで起動することをお勧めします。

3.18 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

ここでは，アプリケーションサーバのプロセスが使用する TCP/UDP のポート番号について説明します。

デフォルト値が「(浮動)」のポートは，ポート番号を明示的に固定しない場合にアプリケーションサーバによって自動的に番号が付けられるポートです。

アプリケーションサーバが使用する TCP/UDP のポート番号の説明を次の表に示します。なお，ご使用の OS によっては，ネットワーク単位ではなくホスト単位でファイアウォールが設定されているものがあります。これらのファイアウォールでは，localhost（127.0.0.1）以外との通信は，同一ホスト内でもファイアウォールのフィルタリングの対象になる場合があります。この場合は，ホスト内でしか通信しないポートであっても，フィルタで通信を許可する設定にしてください。

表 3-4 アプリケーションサーバが使用する TCP/UDP のポート番号

項番	プロセス	説明	デフォルト値
(1)	J2EE サーバまたは SFO サーバ	EJB コンテナのリクエスト受付ポート。	（浮動）
(2)		管理用通信ポート。※	8080
(3)		Web サーバ（リダイレクタ）からのリクエスト受付ポート。※	8007
(4)		トランザクションサービス使用時のトランザクションリカバリ処理通信ポート。 トランザクションサービス使用時に必要です。	20302
(5)		インプロセスで起動するネーミングサービスのリクエスト受付ポート。	900
(6)		インプロセス HTTP サーバのリクエスト受付ポート。 インプロセス HTTP サーバを使用するときに必要です。	80
(7)		RMI レジストリのリクエスト受付ポート。	23152
(8)		共有キューを使用して複数システム間でのアプリケーション連携をする場合のイベント受信用ポート。	20351
(9)		稼働情報取得時のリクエスト受付ポート。	（浮動）
(10)		OpenTP1 からの RPC 要求を待ち受けるポート。	23700
(11)		OpenTP1 からの同期点要求を待ち受けるポート。	23900
(12)	運用監視エージェント	運用監視エージェントの通信用ポート。	（浮動）
(13)	スマートエージェント	スマートエージェントの通信用ポート環境変数。 UDP による双方向通信に必要です。	14000

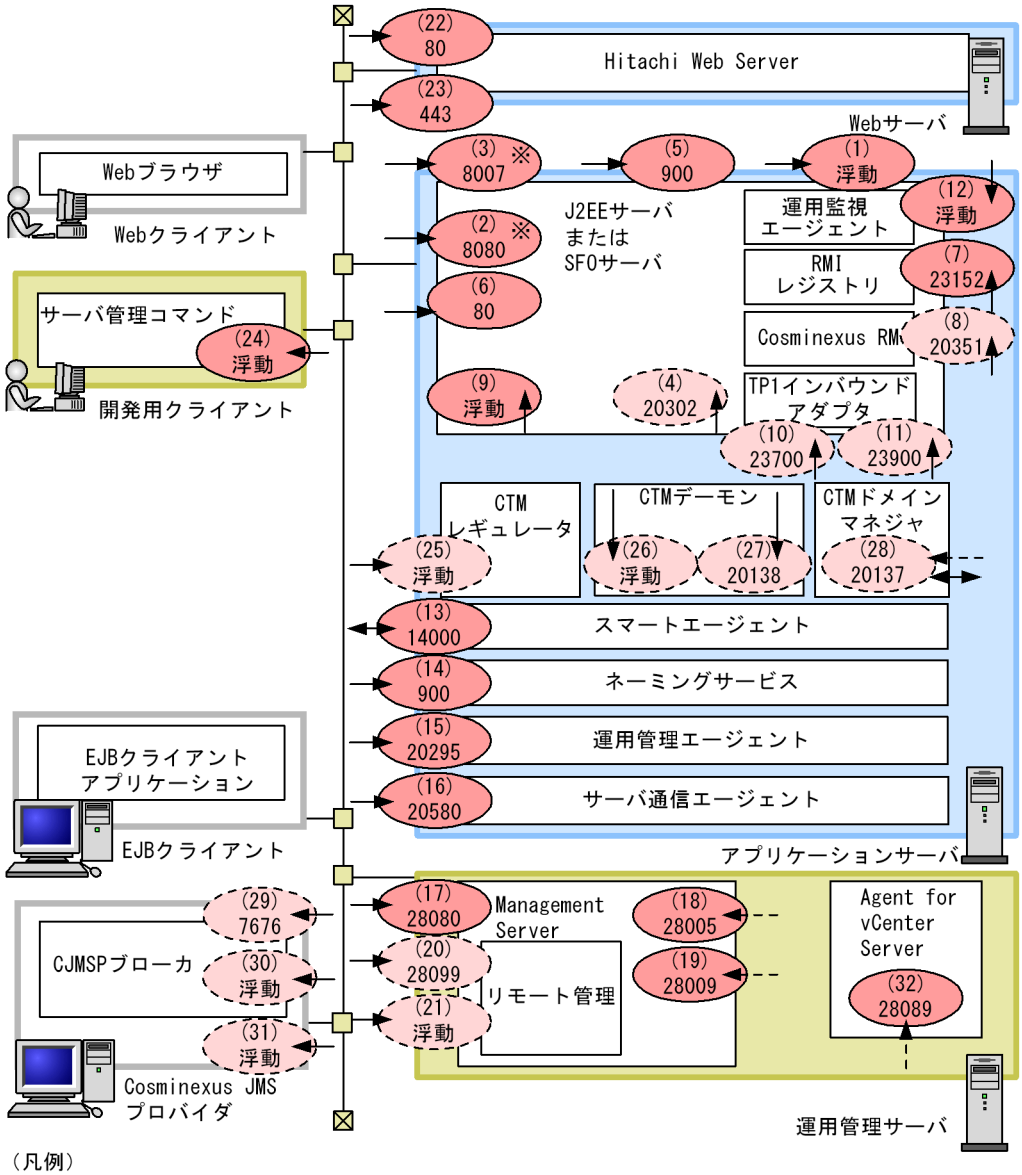
項番	プロセス	説明	デフォルト値
(14)	ネーミングサービス	ネーミングサービスのリクエスト受付ポート引数（Cosminexus TPBroker が利用）。	900
(15)	運用管理エージェント	運用管理エージェントが Management Server との通信に使用するポート。	20295
(16)	サーバ通信エージェント	サーバ通信エージェントが仮想サーバマネージャとの通信に使用するポート。	20580
(17)	Management Server	Management Server の http ポート。	28080
(18)		Management Server の終了要求ポート。 ホスト内通信に必要です。	28005
(19)		Management Server の内部通信ポート。 ホスト内通信に必要です。	28009
(20)		Manager リモート管理機能への接続ポート。	28099
(21)		Manager リモート管理機能へのクライアント接続ポート。	（浮動）
(22)	Hitachi Web Server	Hitachi Web Server の http ポート。	80
(23)		Hitachi Web Server の https ポート。	443
(24)	サーバ管理コマンド	サーバ管理コマンドが J2EE サーバと通信するポート。	（浮動）
(25)	CTM レギュレータ	CTM レギュレータが EJB クライアントからのリクエストを受け付けるポートの基底値。基底値＋プロセス数だけ使用します。 CTM 使用時に必要です。	（浮動）
(26)	CTM デーモン	CTM デーモンが EJB クライアントからのリクエストを受け付けるポート。 CTM 使用時に必要です。	（浮動）
(27)		CTM デーモンがほかのデーモンや J2EE サーバなどと通信するポート。 CTM 使用時に必要です。	20138
(28)	CTM ドメインマネジャ	CTM ドメインマネジャがほかの CTM ドメインマネジャと通信するポート。 CTM 使用時に，TCP および UDP 通信（ブロードキャスト）をするために必要です。	20137
(29)	CJMSP ブローカー	Cosminexus JMS プロバイダのブローカがリソースアダプタやコマンドからのリクエストを受け付けるためのポート。	7676
(30)		Cosminexus JMS プロバイダのブローカがリソースアダプタとコネクションを確立するためのポート。	（浮動）
(31)		Cosminexus JMS プロバイダのブローカがコマンドとコネクションを確立するためのポート。	（浮動）

項番	プロセス	説明	デフォルト値
(32)	Management Server	08-50 モードの仮想サーバマネージャ（Management Server）が vCenter Server の接続処理を行うための，内部で起動するプロセス（Agent for vCenter Server）のポート。	28089

注※　SFO サーバでは使用しません。

アプリケーションサーバのプロセスが使用する TCP/UDP のポート番号について，次の図に示します。(x) は表の項番と対応しています。

図 3-74　アプリケーションサーバが使用する TCP/UDP のポート番号



（凡例）

(X) YYYYY：ファイアウォールを使用する場合は必ず設定するポートです。
YYYYYはデフォルトのポート番号です。(X)は表の項番と対応しています。

(X) YYYYY：ファイアウォールを使用する場合に，使用する機能に応じて設定するポートです。
YYYYYはデフォルトのポート番号です。(X)は表の項番と対応しています。

――――►：そのポートを通じて，ホスト外にサービスを提供することを示します。

- - - -►：そのポートを通じて，ホスト内で通信することを示します。

◄――►：そのポートを通じて，ホスト外と双方向に通信することを示します。

注※　SFOサーバでは使用しません。

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

ポート番号の指定個所を次の表に示します。表の項番は図中の項番と対応しています。

表 3-5　アプリケーションサーバが使用する TCP/UDP のポート番号の指定個所

項番	定義ファイル	設定対象	パラメタ名※1
(1)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-server）	vbroker.se.iiop_tp.scm.iiop_tp.listener.port※2
(2)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	ejbserver.http.port
(3)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.port
(4)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-tier）	ejbserver.distributedtx.recovery.port
(5)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-tier）	inprocess.ns.port
(6)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.port
(7)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-server）	ejbserver.rmi.naming.port
(8)	Connector 属性ファイル	Cosminexus RM	<config-property> タグに指定する RMSHPort※3
(9)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-server）	ejbserver.rmi.remote.listener.port
(10)	Connector 属性ファイル	TP1 インバウンドアダプタ	<config-property> タグに指定する scd_port※4
(11)	Connector 属性ファイル	TP1 インバウンドアダプタ	<config-property> タグに指定する trn_port※4
(12)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-server）	mngagent.connector.port
(13)	簡易構築定義ファイル	論理スマートエージェント（smart-agent）	smartagent.port

項番	定義ファイル	設定対象	パラメタ名※1
(14)	簡易構築定義ファイル	論理 J2EE サーバ（j2ee-server） 論理 SFO サーバ（sfo-server）	ejbserver.naming.port
(15)	adminagent.properties	運用管理エージェント	adminagent.adapter.port キー
(16)	sinaviagent.properties※5	サーバ通信エージェント	sinaviagent.port キー
(17)	mserver.properties	Management Server	webserver.connector.http.port キー
(18)	mserver.properties	Management Server	webserver.shutdown.port キー
(19)	mserver.properties	Management Server	webserver.connector.ajp13.port キー
(20)	mserver.properties	Management Server	com.cosminexus.mngsvr.management.port キー
(21)	mserver.properties	Management Server	com.cosminexus.mngsvr.management.listen.port キー
(22)	簡易構築定義ファイル	論理 Web サーバ（web-server）	Listen
(23)	簡易構築定義ファイル	論理 Web サーバ（web-server）	Listen
(24)	usrconf.properties（サーバ管理コマンド用システムプロパティファイル）	サーバ管理コマンド	vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー
(25)	簡易構築定義ファイル	論理 CTM（component-transaction-monitor）	ctm.RegOption
(26)	簡易構築定義ファイル	論理 CTM（component-transaction-monitor）	ctm.EjbPort
(27)	簡易構築定義ファイル	論理 CTM（component-transaction-monitor）	ctm.port
(28)	簡易構築定義ファイル	論理 CTM ドメインマネジャ（ctm-domain-manager）	cdm.port
(29)	config.properties	CJMSP ブローカー	imq.portmapper.port キー
(30)	config.properties	CJMSP ブローカー	imq.jms.tcp.port キー
(31)	config.properties	CJMSP ブローカー	imq.admin.tcp.port キー
(32)	vmx.properties	08-50 モードの仮想サーバマネージャ（Management Server）	vmx.vcenterserver.agent.port キー

（凡例）－：該当しない。
注※ 1　設定ファイルが簡易構築定義ファイルの場合は，<configuration> タグ内の <param-name> の指定値を指します。
注※ 2　Developer Standard 以外の場合のパラメタ名です。Developer Standard を使用する場合のパラメタ名は，ejbserver.rmi.remote.listener.port になります。
注※ 3　RMSHPort は，リソースアダプタ Cosminexus RM のプロパティ定義で指定するコンフィグレーションプロパティです。RMSHPort については，マニュアル「Cosminexus Reliable Messaging」の「6.　コンフィグレーションプロパティ」を参照してください。
注※ 4　scd_port および trn_port は，リソースアダプタ TP1 インバウンドアダプタのプロパティ定義で指定するコンフィグレーションプロパティです。scd_port および trn_port については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (コンテナ共通機能)」の「4.11.2　リソースアダプタの設定」を参照してください。
注※ 5　サーバ通信エージェントの詳細については，サーバ通信エージェントのドキュメントを参照してください。

参考

運用管理ポータルまたはファイル編集によってアプリケーションサーバを構築している場合の TCP/UDP のポートの設定個所を次に示します。

表 3-6　運用管理ポータルまたはファイル編集によってアプリケーションサーバを構築している場合の TCP/UDP のポートの設定個所

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(1)	• J2EE サーバの場合 ［EJB コンテナの設定］画面の「オプション」の「通信ポート番号」 • SFO サーバの場合 ［システムプロパティの設定］画面の「システムプロパティの定義」に追加	usrconf.properties の vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー
(2)	［J2EE サーバの基本設定］画面の「コンテナの設定」の「簡易 Web サーバのポート番号」	usrconf.properties の ejbserver.http.port キー
(3)	［Web コンテナの設定］画面の「Web サーバとの接続」の「ポート番号」	usrconf.properties の webserver.connector.ajp13.port キー
(4)	［トランザクションの設定］画面の「トランザクションに関する設定」の「JTA リカバリの固定ポート番号」	usrconf.properties の ejbserver.distributedtx.recovery.port キー
(5)	• J2EE サーバの場合 ［ネーミングの設定］画面の「インプロセス選択時の設定」の「ポート番号」 • SFO サーバの場合 ［SFO サーバの基本設定］画面の「ネーミングサービスの設定」	usrconf.properties の ejbserver.naming.port キー

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(6)	［Web コンテナの設定］画面の「インプロセス HTTP サーバ機能の使用」の「ポート番号」	usrconf.properties の webserver.connector.inprocess_http.port キー
(7)	• J2EE サーバの場合 ［通信の設定］画面の「RMI レジストリの設定」の「ポート番号」 • SFO サーバの場合 ［通信の設定］画面の「RMI レジストリの設定」の「ポート番号」	usrconf.properties の ejbserver.rmi.naming.port キー
(8)	Connector 属性ファイルの <config-property> タグに指定する RMSHPort ※1	Connector 属性ファイルの <config-property> タグに指定する RMSHPort ※1
(9)	• J2EE サーバの場合 ［通信の設定］画面の「RMI レジストリの設定」の「通信ポート番号」 • SFO サーバの場合 ［通信の設定］画面の「RMI レジストリの設定」の「通信ポート番号」	usrconf.properties の ejbserver.rmi.remote.listener.port キー
(10)	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの <config-property> タグに指定する scd_port	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの <config-property> タグに指定する scd_port
(11)	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの <config-property> タグに指定する trn_port	TP1 インバウンドアダプタの Connector 属性ファイルのリソースアダプタの <config-property> タグに指定する trn_port
(12)	• J2EE サーバの場合 ［J2EE コンテナの設定］画面の「運用監視エージェントの設定」の「ポート番号」 • SFO サーバの場合 ［コンテナの設定］画面の「運用監視エージェントの設定」の「ポート番号」	mngagent.< 実サーバ名 >.properties の mngagent.connector.port キー
(13)	［スマートエージェントの設定］画面の「スマートエージェントに関する設定」の「監視ポート番号」	環境変数 OSAGENT_PORT
(14)	• CORBA ネーミングサービスをインプロセスで起動する場合 ［J2EE サーバの基本設定］画面の「利用するネーミングサービスの設定」の「インプロセス用のポート番号」 • CORBA ネーミングサービスをアウトプロセスで起動する場合 ［ホスト内のサーバの設定］画面の「ネーミングサービスの設定」の「ネーミングサービスのポート番号」	• CORBA ネーミングサービスをインプロセスまたはアウトプロセスで自動起動する場合 usrconf.properties の ejbserver.naming.port キー • CORBA ネーミングサービスを手動起動する場合 nameserv コマンドのコマンド引数に「-Dvbroker.se.iiop_tp.scm.iiop_tp.listener.port=< ポート番号 >」を指定。

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(15)	adminagent.properties の adminagent.adapter.port キー	adminagent.properties の adminagent.adapter.port キー
(16)	sinaviagent.properties の sinaviagent.port キー※2	sinaviagent.properties の sinaviagent.port キー
(17)	［ネットワークの設定］画面の「Management Server 接続 HTTP ポート番号」	mserver.properties の webserver.connector.http.port キー
(18)	［ネットワークの設定］画面の「Management Server 終了要求受信ポート番号」	mserver.properties の webserver.shutdown.port キー
(19)	［ネットワークの設定］画面の「Management Server 内部通信用ポート番号」	mserver.properties の webserver.connector.ajp13.port キー
(20)	mserver.properties の com.cosminexus.mngsvr.management.port キー	mserver.properties の com.cosminexus.mngsvr.management.port キー
(21)	mserver.properties の com.cosminexus.mngsvr.management.listen.port キー	mserver.properties の com.cosminexus.mngsvr.management.listen.port キー
(22)	［ホスト内のサーバの設定］画面の「J2EE サーバの設定」の「ポート番号」の「http」	httpsd.conf の Listen ディレクティブまたは Port ディレクティブ
(23)	［ホスト内のサーバの設定］画面の「J2EE サーバの設定」の「ポート番号」の「https」	httpsd.conf の Listen ディレクティブまたは Port ディレクティブ
(24)	usrconf.properties（サーバ管理コマンド用システムプロパティファイル）の vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー	usrconf.properties（サーバ管理コマンド用システムプロパティファイル）の vbroker.se.iiop_tp.scm.iiop_tp.listener.port キー
(25)	［レギュレータの設定］画面の「CTM レギュレータの設定」の「設定ファイル」	ctmregltd コマンドまたは ctmstart コマンドの引数 -CTMEjbPort
(26)	［スケジューリングの設定］画面の「詳細設定」の「EJB リクエスト受信ポート番号」	ctmstart コマンドの引数 -CTMEjbPort
(27)	［CTM の基本設定］画面の「基本設定」の「ポート番号」	ctmstart コマンドの引数 -CTMPort
(28)	［CTM ドメインマネジャの基本設定］画面の「ポート番号」	ctmdmstart コマンドの引数 -CTMPort
(29)	config.properties の imq.portmapper.port キー	config.properties の imq.portmapper.port キー
(30)	config.properties の imq.jms.tcp.port キー	config.properties の imq.jms.tcp.port キー

項番	運用管理ポータルで構築している場合の設定個所	ファイル編集で構築している場合の設定個所
(31)	config.properties の imq.admin.tcp.port キー	config.properties の imq.admin.tcp.port キー
(32)	vmx.properties の vmx.vcenterserver.agent.port キー	vmx.properties の vmx.vcenterserver.agent.port キー

注※ 1　RMSHPort は，リソースアダプタ Cosminexus RM のプロパティ定義で指定するコンフィグレーションプロパティです。RMSHPort については，マニュアル「Cosminexus Reliable Messaging」の「6. コンフィグレーションプロパティ」を参照してください。
注※ 2　サーバ通信エージェントの詳細については，サーバ通信エージェントのドキュメントを参照してください。

4 システム構成の検討（バッチアプリケーション実行基盤）

この章では，バッチアプリケーション実行基盤を構築する場合のシステム構成の検討について説明します。システムを設計する流れに沿って，標準的なシステム構成のパターンを示します。また，意識する必要があるコンポーネント，プロセスおよび処理の流れについて説明します。
J2EE アプリケーション実行基盤のシステム構成を検討する場合は，「3. システム構成の検討（J2EE アプリケーション実行基盤）」を参照してください。

4.1　システム構成を検討するときに考慮すること

この節では，バッチアプリケーション実行基盤としてアプリケーションサーバを使用する場合に，システム構成の検討で考慮することについて説明します。

4.1.1　システムの目的と構成

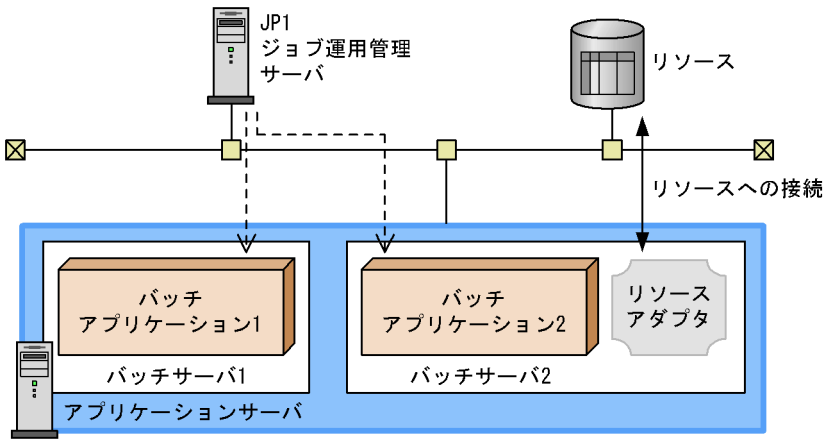
バッチアプリケーション実行基盤は，Java アプリケーションとして実装されたバッチアプリケーションを実行するための環境です。バッチアプリケーションは，JSP，Servlet および Enterprise Bean などの J2EE アプリケーションを利用しない Java アプリケーションとして実装します。なお，バッチアプリケーションから，J2EE サーバ上の Enterprise Bean を呼び出すことはできます。

バッチアプリケーションは，バッチサーバ上で動作します。バッチアプリケーションがリソースと接続する場合に使用するリソースアダプタも，バッチサーバ上で動作します。

バッチアプリケーションは，バッチ実行コマンドによって実行されます。バッチ実行コマンドは，ユーザが直接実行するほか，JP1/AJS2 のジョブとして自動実行することができます。

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。この図では，アプリケーションサーバ上で，バッチアプリケーションごとに二つのバッチサーバが動作しています。また，バッチアプリケーションは，JP1/AJS2 を使用して実行しています。

図 4-1　バッチアプリケーションを実行するシステムの構成例



（凡例）

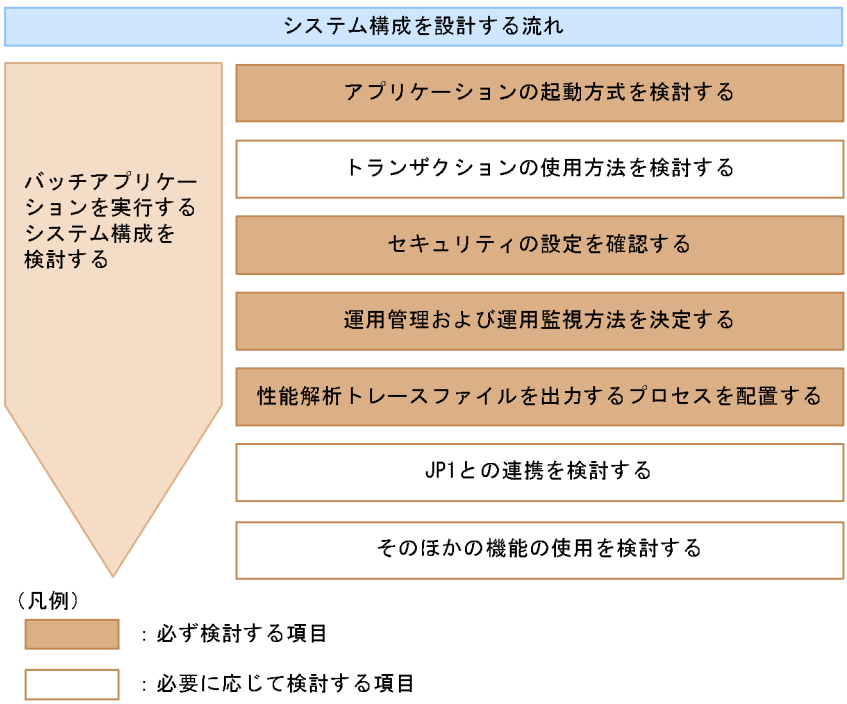
--≫：バッチジョブの実行

このほか，バッチアプリケーションから J2EE サーバ上の Enterprise Bean に接続する場合は，オンライン処理を実行するバックシステムに接続することもできます。バックシステムについては，「3.1.1　システムの目的と構成」を参照してください。

4.1.2　システム構成の設計手順

システム構成は，次の流れで設計します。

図 4-2　システム構成を設計する流れ（バッチアプリケーション実行基盤の場合）



注　使用する機能に応じた構成を検討する順序は任意です。

（1）アプリケーションの起動方式を検討する

バッチアプリケーションの起動方式を検討します。次の 2 種類から選択します。

● バッチサーバ上で起動する

バッチサーバ上で Java アプリケーションを起動する方式です。常駐型の JavaVM プロセスであるバッチサーバを使用することで，JavaVM の起動に掛かるオーバーヘッドを削減できます。また，CTM を利用すると，バッチサーバ上で動作するバッチアプリケーションの実行をスケジューリングできます。

また，バッチサーバ上で起動する場合，リソースアダプタとして DB Connector が使用できます。

● cjclstartap コマンドを使用して個別に起動する

java コマンドと同じように Java アプリケーションを起動する方式です。この方式の場合，バッチアプリケーションを実行するたびに，JavaVM の起動が必要です。

起動方式としてバッチサーバ上で起動することを選択した場合は，次の検討項目に進みます。なお，cjclstartap コマンドを使用することを選択した場合は，以降の検討項目は該当しません。

(2) トランザクションの使用方法を検討する

リソースと接続する場合は，トランザクションの使用方法を検討します。次の 2 種類から選択します。

● DB Connector を使用する

アプリケーションサーバが提供するリソースアダプタである DB Connector を使用する方法です。DB Connector の機能として，コネクションプーリングやステートメントプーリングが使用できます。また，フルガーベージコレクションの発生を制御する機能も使用できます。なお，バッチサーバで管理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。

● JDBC ドライバを直接使用する

JDBC ドライバが提供する API を使用して，トランザクション管理に必要な処理を実装する方法です。

DB Connector を使用する場合のリソースアダプタとリソースの構成については，「3.3.2 リソースの種類とリソースアダプタ」を参照してください。ただし，バッチサーバの場合，接続できるリソースはデータベースだけです。

(3) セキュリティの設定を確認する

バッチサーバは，SecurityManager によるセキュリティ保護を無効にして起動します。

(4) 性能解析トレースファイルを出力するプロセスを配置する

性能解析トレースファイルを出力するためのプロセスである PRF デーモン（パフォーマンストレーサ）の配置を確認します。PRF デーモンは，バッチサーバごとに配置します。

(5) 運用管理および運用監視方法を決定する

運用管理および運用監視を実行するためのプロセスである Management Server の配置を確認します。バッチアプリケーション実行基盤では，バッチサーバと同じマシンに配置します（ホスト単位管理モデルを使用します）。

(6) JP1 との連携を検討する

バッチアプリケーションの実行を JP1/AJS2 のジョブとして自動投入する場合は，JP1 との連携方法を検討します。

（7） そのほかの機能の使用を検討する

使用する機能に応じて，次の構成を検討してください。

● サーバ間連携をする構成

バッチアプリケーションから J2EE サーバ上の Enterprise Bean を呼び出す場合に検討します。「3.5　サーバ間での連携を検討する」を参照してください。

● クラスタソフトウェアを使用した障害時の系切り替えをする構成

バッチサーバを系切り替えの対象にする場合に検討します。
次の説明を参照してください。

- 「3.11.1　アプリケーションサーバの実行系と待機系を 1:1 にする構成（トランザクションサービスを使用しない場合）」
- 「3.11.4　アプリケーションサーバの実行系と待機系を相互スタンバイにする構成」

参照先の記述のうち，「J2EE サーバ」は「バッチサーバ」に読み替えてください。

4.1.3　バッチアプリケーションを実行するシステムで使用する TCP/UDP のポートについての注意事項

バッチアプリケーションを実行するシステムで使用する TCP/UDP のポートについて，プロセスごとに説明します。バッチアプリケーションを実行するシステムで使用する TCP/UDP のポートを次の表に示します。

表 4-1　バッチアプリケーションを実行するシステムのプロセスが使用する TCP/UDP のポート

項番※1	プロセス	説明
(1)	バッチサーバ	EJB コンテナのリクエスト受付ポート。
(2)		管理用通信ポート。
(3)		Web サーバ（リダイレクタ）からのリクエスト受付ポート。
(5)		インプロセスで起動するネーミングサービスのリクエスト受付ポート。
(7)		RMI レジストリのリクエスト受付ポート。
(9)		稼働情報取得時のリクエスト受付ポート。
(13)	スマートエージェント※2	スマートエージェントの通信用ポート環境変数。 UDP による双方向通信に必要です。
(25)	CTM レギュレータ※2	CTM レギュレータが EJB クライアントからのリクエストを受け付けるポートの基底値。基底値＋プロセス数だけ使用します。 CTM 使用時に必要です。

項番※1	プロセス	説明
(26)	CTM デーモン※2	CTM デーモンが EJB クライアントからのリクエストを受け付けるポート。 CTM 使用時に必要です。
(27)		CTM デーモンがほかのデーモンや J2EE サーバなどと通信するポート。 CTM 使用時に必要です。
(28)	CTM ドメインマネジャ※2	CTM ドメインマネジャがほかの CTM ドメインマネジャと通信するポート。 CTM 使用時に，TCP および UDP 通信（ブロードキャスト）をするために必要です。

注　スマートエージェントを使用する指定をした場合，スマートエージェントと通信するために，この表で示した以外の複数のポートが使用されます。スマートエージェントの詳細については，マニュアル「Borland(R) Enterprise Server VisiBroker(R) プログラマーズリファレンス」を参照してください。
注※1 「3.18　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」の「表 3-4　アプリケーションサーバが使用する TCP/UDP のポート番号」の項番と対応しています。
注※2　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，必要なプロセスです。

次の場合は，使用するポートが重複しないように設定してください。

- 一つのマシンで J2EE サーバとバッチサーバを同時に使用する場合
- 一つのマシンで複数のバッチサーバを同時に使用する場合

各プロセスが使用する TCP/UDP のポートの詳細については，「3.18　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。

4.2 バッチサーバを使用する場合のシステム構成

この節では，バッチサーバを使用する場合のシステム構成について説明します。バッチサーバを使用するシステムは，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かどうかによって，構成が異なります。

4.2.1 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ないシステムのシステム構成

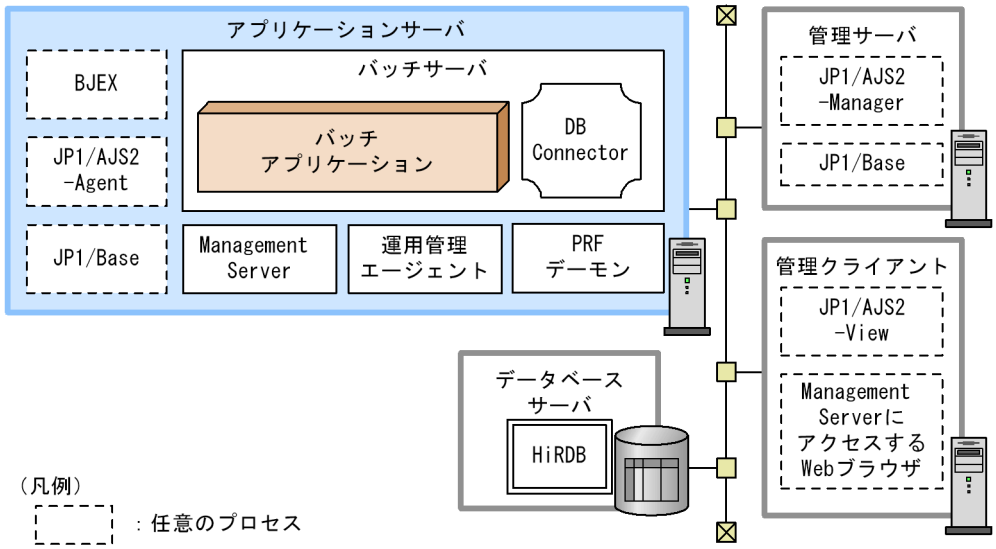
バッチサーバを使用する場合に，バッチアプリケーションのスケジューリング機能を使用しないときのシステムの構成について説明します。

（1）システム構成の特徴

バッチサーバを使用する構成の一つで，CTM を使用しないシステム構成です。この場合，アプリケーションサーバにはバッチサーバを配置します。バッチサーバは，Smart Composer 機能の Web システム（j2ee-tier）として構築，運用します。バッチアプリケーションの実行には，バッチ実行コマンドを使用します。バッチ実行コマンドは，JP1/AJS2 にジョブとして定義して実行できます。

バッチサーバを配置するシステム構成の例を次の図に示します。この例では，バッチアプリケーションは JP1/AJS2 から実行します。また，バッチアプリケーションからリソースアダプタを使用して HiRDB にアクセスします。

図 4-3　バッチサーバを配置するシステム構成の例



注　BJEX：uCosminexus Batch Job Execution Server

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

リソースとの接続に DB Connector が使用できます。また，uCosminexus Batch Job Execution Server との連携もできます。

処理の流れ

バッチアプリケーションの実行は，JP1/AJS2 のジョブとして定義しておきます。JP1/AJS2 のジョブとして実行されたバッチアプリケーションは，バッチサーバ上の DB Connector を経由して HiRDB にアクセスします。

(2) それぞれのマシンに必要なプロセスとソフトウェア

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server をインストールする必要があります。なお，開発環境の場合は，Developer をインストールする必要があります。

さらに，JP1/AJS2 によってバッチアプリケーションを実行する場合や，uCosminexus Batch Job Execution Server と連携したジョブの制御をする場合は，次の製品もインストールする必要があります。

JP1/AJS2 と連携する場合に必要な製品

- JP1/Base
- JP1/AJS2 - Agent

- JP1/AJS2 - Manager
- JP1/AJS2 - View

uCosminexus Batch Job Execution Server と連携する場合に必要な製品

- uCosminexus Batch Job Execution Server
- JP1/AJS2 と連携する場合に必要な製品

起動するプロセスは次のとおりです。

● バッチサーバ

● PRF デーモン

● Management Server

● 運用管理エージェント

● JP1/Base と JP1/AJS2 のプロセス（JP1/AJS2 と連携する場合）

● uCosminexus Batch Job Execution Server のプロセス（uCosminexus Batch Job Execution Server と連携する場合）

また，データベースと接続する場合は，使用するデータベースと接続するためのソフトウェアが必要です。データベースに接続するために必要な製品については，「3.6.1　ローカルトランザクションを使用する場合の構成」を参照してください。ただし，バッチサーバで使用できるリソースは，次に示すデータベースだけです。

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

(b) 管理サーバマシンおよび管理クライアントマシン

管理サーバマシンおよび管理クライアントマシンは，JP1/AJS2 と連携する場合に必要です。インストールする製品および起動するプロセスについては，マニュアル「JP1/Automatic Job Management System 2 設計・運用ガイド」を参照してください。

(c) データベースサーバマシン

データベースサーバマシンに必要な製品については，「3.6.1　ローカルトランザクションを使用する場合の構成」のリソースマネジャが動作するマシンの説明を参照してください。ただし，バッチサーバで使用できるリソースは，次に示すデータベースだけです。

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

4.2.2　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システムのシステム構成

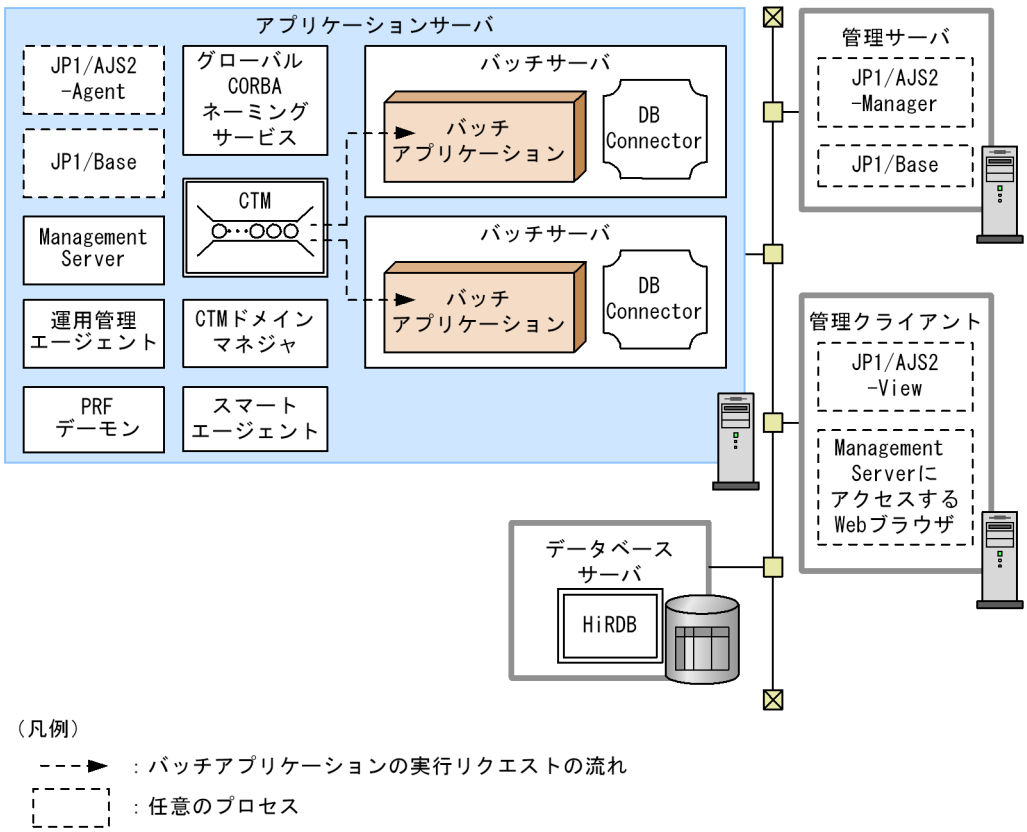
バッチサーバを使用する場合に，バッチアプリケーションのスケジューリング機能を使用するときのシステムの構成について説明します。

（1）システム構成の特徴

バッチサーバを使用する構成の一つで，CTM を使用するシステム構成です。この場合，アプリケーションサーバにはバッチサーバと CTM を配置します。バッチサーバと CTM は，Smart Composer 機能の Web システム（ctm-tier）として構築，運用します。バッチアプリケーションの実行には，バッチ実行コマンドを使用します。JP1/AJS2 のジョブ，または直接コマンドから実行されたバッチアプリケーションの実行は，CTM によってスケジューリングされ，バッチサーバに振り分けられます。

CTM を使用するシステム構成の例を次の図に示します。この例では，バッチサーバを 2 台配置し，CTM を使用してバッチアプリケーションの実行リクエストをスケジューリングします。バッチ実行コマンドは JP1/AJS2 のジョブから実行します。

図 4-4　CTM を使用するシステム構成の例



これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

CTM によってバッチアプリケーションの実行リクエストをスケジューリングすることで，複数のバッチ実行コマンドを同時に実行できます。また，バッチ実行コマンドでバッチサーバを指定する必要がありません。このため，バッチ実行コマンドを JP1/AJS のジョブで定義している場合でも，ジョブの定義を変更することなく，バッチサーバの同時実行数を変更できます。

処理の流れ

JP1/AJS2 のジョブ，または直接バッチ実行コマンドから実行されたバッチアプリケーションは，CTM のスケジュールキューにバッチアプリケーションの実行リクエストとして登録されます。スケジュールキュー内のバッチアプリケーションの実行リクエストは，CTM によって適切なバッチサーバに振り分けられます。なお，振り分け先のバッチサーバがない場合，バッチアプリケーションの実行リクエストはスケジュールキュー内に滞留（待機）します。

(2) それぞれのマシンに必要なプロセスとソフトウェア

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

(a) アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Enterprise をインストールする必要があります。なお，開発環境の場合は，Developer Professional をインストールする必要があります。

さらに，JP1/AJS2 によってバッチアプリケーションを実行する場合は，次の製品もインストールする必要があります。

- JP1/Base
- JP1/AJS2 - Agent
- JP1/AJS2 - Manager
- JP1/AJS2 - View

注意事項

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は，uCosminexus Batch Job Execution Server と連携できません。

起動するプロセスは次のとおりです。

● バッチサーバ

● PRF デーモン

● グローバル CORBA ネーミングサービス

● CTM のプロセス群（CTM デーモンおよび CTM レギュレータ）

● CTM ドメインマネジャ

● スマートエージェント

● Management Server

● 運用管理エージェント

● JP1/Base と JP1/AJS2 のプロセス（JP1/AJS2 と連携する場合）

また，データベースと接続する場合は，使用するデータベースと接続するためのソフトウェアが必要です。データベースに接続するために必要な製品については，「3.6.1　ローカルトランザクションを使用する場合の構成」を参照してください。ただし，バッチサーバで使用できるリソースは，次に示すデータベースだけです。

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

(b) 管理サーバマシンおよび管理クライアントマシン

管理サーバマシンおよび管理クライアントマシンは，JP1/AJS2 と連携する場合に必要です。インストールする製品および起動するプロセスについては，マニュアル「JP1/Automatic Job Management System 2 設計・運用ガイド」を参照してください。

(c) データベースサーバマシン

データベースサーバマシンに必要な製品については，「3.6.1　ローカルトランザクションを使用する場合の構成」のリソースマネジャが動作するマシンの説明を参照してください。ただし，バッチサーバで使用できるリソースは，次に示すデータベースだけです。

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

5 使用するリソースの見積もり（J2EE アプリケーション実行基盤）

この章では，J2EE アプリケーションを実行するシステムで使用するリソース，および仮想メモリ使用量の見積もり方法について説明します。システムを動作させるために必要なディスクおよびメモリの容量を算出するときの参考にしてください。バッチアプリケーション実行基盤の使用リソースおよび仮想メモリ所要量の見積もりについては，「6. 使用するリソースの見積もり（バッチアプリケーション実行基盤）」を参照してください。

5.1 システム構成ごとに使用するリソース

システムが動作するためには，OS やデータベースなどのホスティング環境の設定が必要な場合があります。システムが必要とするリソースは，システムの構成ごとに異なるため，ここではシステム構成ごとに使用するリソースと，リソースの所要量の見積もりについて説明します。システム構成ごとに使用するリソースと，リソースの見積もりの参照先を次の表に示します。

表 5-1 システム構成ごとに使用するリソースと見積もりの参照先

システム構成ごとに使用するリソース	参照先
Web サーバと J2EE サーバを同じマシンに配置する場合の使用リソース	5.1.1
Web サーバと J2EE サーバを別のマシンに配置する場合の使用リソース	5.1.2
インプロセス HTTP サーバ機能を使用する場合の使用リソース	5.1.3
データベースの使用リソース	5.1.4
運用管理サーバの使用リソース	5.1.5
メモリセッションフェイルオーバ機能を使用する場合の使用リソース※	5.1.6
CTM を使用する場合の使用リソース	5.1.7

注※
メモリセッションフェイルオーバ機能を使用する場合，J2EE サーバが使用するメモリ量が増加します。

また，プロセスごとに使用するリソースの見積もりについては，「5.2 プロセスごとに使用するリソース」を参照してください。

仮想メモリ所要量については，「5.3 仮想メモリの使用量の見積もり」を参照してください。ディスク占有量については，アプリケーションサーバのリリースノートを参照してください。

ポイント

Windows システムの場合は，この節で説明する内容のうち，「5.1.4 データベースの使用リソース」だけを参照してください。
システムで利用できるプロセス数，共用メモリ，ファイルディスクリプタ数，および Windows システムやプロセスで利用できるスレッド数に，特に制限はありません。

5.1.1 Web サーバと J2EE サーバを同じマシンに配置する場合の使用リソース

Web サーバと J2EE サーバを同じマシンに配置する場合の使用リソースの見積もりについて，OS ごとに説明します。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

（1） AIX の場合

AIX の場合の，使用リソースの見積もりについて，次の表に示します。

表 5-2　使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	－	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	－
プロセス数	－	リクエスト最大同時処理数※2 × 2 ＋ 5	－
スレッド数	－	リクエスト最大同時処理数※2 × 2 ＋ 41 ＋ J2EE サーバのスレッド数※3	－
ファイルディスクリプタ数	nofiles	J2EE サーバのファイルディスクリプタ数※3 ＋ 76 ＋リクエスト最大同時処理数※2 × 4	/etc/security/limits

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

注※ 3

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（2） HP-UX の場合

HP-UX の場合の，使用リソースの見積もりについて，次の表に示します。

表 5-3 使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	kctune shmmax= 1073741824
プロセス数	nproc	リクエスト最大同時処理数※2 × 2 ＋ 5	kctune nproc=4200
スレッド数	nkthread	リクエスト最大同時処理数※2 × 2 ＋ 41 ＋ J2EE サーバのスレッド数※3	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	J2EE サーバのファイルディスクリプタ数※3 ＋ 76 ＋リクエスト最大同時処理数※2 × 4	kctune nfile= 65536

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15 論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

注※ 3

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1 J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

(3) Linux の場合

Linux の場合の，使用リソースの見積もりについて，次の表に示します。

表 5-4 使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	SHMMAX	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	/proc/sys/kernel/shmmax
プロセス数	threads-max※3	リクエスト最大同時処理数※2 × 2 ＋ 5	/proc/sys/kernel/threads-max
スレッド数	threads-max※3	リクエスト最大同時処理数※2 × 2 ＋ 41 ＋ J2EE サーバのスレッド数※4	－

システムリソース	パラメタ	所要量	オプション設定ファイル例
ファイルディスクリプタ数	fs.file-max	J2EE サーバのファイルディスクリプタ数※4 ＋ 76 ＋リクエスト最大同時処理数※2 × 4	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

注※ 3

threads-max パラメタには，プロセス数とスレッド数の合計を指定してください。

注※ 4

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（4） Solaris の場合

Solaris の場合の，使用リソースの見積もりについて，次の表に示します。

表 5-5　使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shminfo_shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	/etc/system
プロセス数	max_nprocs	リクエスト最大同時処理数※2 × 2 ＋ 5	/etc/system
スレッド数	－	リクエスト最大同時処理数※2 × 2 ＋ 41 ＋ J2EE サーバのスレッド数※3	－
ファイルディスクリプタ数	rlim_fd_max	J2EE サーバのファイルディスクリプタ数※3 ＋ 76 ＋リクエスト最大同時処理数※2 × 4	/etc/system

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

注※ 3

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

5.1.2 Web サーバと J2EE サーバを別のマシンに配置する場合の使用リソース

Web サーバと J2EE サーバを別のマシンに配置する場合の使用リソースの見積もりについて，OS ごとに説明します。Web サーバと J2EE サーバを別マシンに配置する場合は，Web サーバマシンとアプリケーションサーバマシンのそれぞれで使用するリソースを見積もります。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

（1） AIX の場合

AIX の場合の，Web サーバマシンおよびアプリケーションサーバマシンの使用リソースの見積もりについて説明します。

（a） Web サーバマシンの使用リソースの見積もり

Web サーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-6　Web サーバマシンの使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	－	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	－
プロセス数	－	リクエスト最大同時処理数※2 × 2 ＋ 4	－
スレッド数	－	リクエスト最大同時処理数※2 × 2 ＋ 35	－
ファイルディスクリプタ数	nofiles	リクエスト最大同時処理数※2 × 4 ＋ 75	/etc/security/limits

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

（b）アプリケーションサーバマシンの使用リソースの見積もり

アプリケーションサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-7　アプリケーションサーバマシンの使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	－	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	－
プロセス数	－	4	－
スレッド数	－	J2EE サーバのスレッド数※2 ＋ 34	－
ファイルディスクリプタ数	nofiles	J2EE サーバのファイルディスクリプタ数※2 ＋ 43	/etc/security/limits

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（2）HP-UX の場合

HP-UX の場合の，Web サーバマシンおよびアプリケーションサーバマシンの使用リソースの見積もりについて説明します。

（a）Web サーバマシンの使用リソースの見積もり

Web サーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-8 Web サーバマシンの使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	kctune shmmax= 1073741824
プロセス数	nproc	リクエスト最大同時処理数※2 × 2 ＋ 4	kctune nproc=4200
スレッド数	nkthread	リクエスト最大同時処理数※2 × 2 ＋ 35	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	リクエスト最大同時処理数※2 × 4 ＋ 75	kctune nfile= 65536

注※1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※2

Web サーバで同時に処理できるリクエストの最大数を指します。

(b) アプリケーションサーバマシンの使用リソースの見積もり

アプリケーションサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-9 アプリケーションサーバマシンの使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	kctune shmmax= 1073741824
プロセス数	nproc	4	kctune nproc=4200
スレッド数	nkthread	J2EE サーバのスレッド数※2 ＋ 34	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	J2EE サーバのファイルディスクリプタ数※2 ＋ 43	kctune nfile= 65536

注※1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※2

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（3） Linux の場合

Linux の場合の，Web サーバマシンおよびアプリケーションサーバマシンの使用リソースの見積もりについて説明します。

（a） Web サーバマシンの使用リソースの見積もり

Web サーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-10　Web サーバマシンの使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	SHMMAX	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	/proc/sys/kernel/shmmax
プロセス数	threads-max※3	リクエスト最大同時処理数※2 × 2 ＋ 4	/proc/sys/kernel/threads-max
スレッド数	threads-max※3	リクエスト最大同時処理数※2 × 2 ＋ 35	－
ファイルディスクリプタ数	fs.file-max	リクエスト最大同時処理数※2 × 4 ＋ 75	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

注※ 3

threads-max パラメタには，プロセス数とスレッド数の合計を指定してください。

（b） アプリケーションサーバマシンの使用リソースの見積もり

アプリケーションサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-11 アプリケーションサーバマシンの使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	SHMMAX	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	/proc/sys/kernel/shmmax
プロセス数	threads-max※2	4	/proc/sys/kernel/threads-max
スレッド数	threads-max※2	J2EE サーバのスレッド数※3 ＋ 34	－
ファイルディスクリプタ数	fs.file-max	J2EE サーバのファイルディスクリプタ数※3 ＋ 43	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15 論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

threads-max パラメタには，プロセス数とスレッド数の合計を指定してください。

注※ 3

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1 J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（4） Solaris の場合

Solaris の場合の，Web サーバマシンおよびアプリケーションサーバマシンの使用リソースの見積もりについて説明します。

（a） Web サーバマシンの使用リソースの見積もり

Web サーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-12 Web サーバマシンの使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shminfo_shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋リクエスト最大同時処理数※2 × 14 × 1,024	/etc/system
プロセス数	max_nprocs	リクエスト最大同時処理数※2 × 2 ＋ 4	/etc/system
スレッド数	－	リクエスト最大同時処理数※2 × 2 ＋ 35	－

システムリソース	パラメタ	所要量	オプション設定ファイル例
ファイルディスクリプタ数	rlim_fd_max	リクエスト最大同時処理数※2 × 4 ＋ 75	/etc/system

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

Web サーバで同時に処理できるリクエストの最大数を指します。

(b) アプリケーションサーバマシンの使用リソースの見積もり

アプリケーションサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-13　アプリケーションサーバマシンの使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shminfo_shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	/etc/system
プロセス数	max_nprocs	4	/etc/system
スレッド数	－	J2EE サーバのスレッド数※2 ＋ 34	－
ファイルディスクリプタ数	rlim_fd_max	J2EE サーバのファイルディスクリプタ数※2 ＋ 43	/etc/system

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

5.1.3　インプロセス HTTP サーバ機能を使用する場合の使用リソース

インプロセス HTTP サーバ機能を使用する場合の使用リソースの見積もりについて，OS

ごとに説明します。インプロセス HTTP サーバ機能を使用する場合，アプリケーションサーバマシンの使用リソースを見積もります。

（1） AIX の場合

AIX の場合の，アプリケーションサーバマシンの使用リソースの見積もりについては，「5.1.2(1)(b)　アプリケーションサーバマシンの使用リソースの見積もり」を参照してください。

（2） HP-UX の場合

HP-UX の場合の，アプリケーションサーバマシンの使用リソースの見積もりについては，「5.1.2(2)(b)　アプリケーションサーバマシンの使用リソースの見積もり」を参照してください。

（3） Linux の場合

Linux の場合の，アプリケーションサーバマシンの使用リソースの見積もりについては，「5.1.2(3)(b)　アプリケーションサーバマシンの使用リソースの見積もり」を参照してください。

（4） Solaris の場合

Solaris の場合の，アプリケーションサーバマシンの使用リソースの見積もりについては，「5.1.2(4)(b)　アプリケーションサーバマシンの使用リソースの見積もり」を参照してください。

5.1.4　データベースの使用リソース

DBMS の使用リソースの見積もりについて説明します。

仮想メモリ所要量については，「5.3　仮想メモリの使用量の見積もり」を参照してください。また，ディスク占有量については，Application Server または Developer のリリースノートを参照してください。

DBMS の使用リソースの見積もりについて，次の表に示します。

表 5-14　DBMS の使用リソースの見積もり

DBMS	使用リソース	所要量
HiRDB	最大同時接続数(pd_max_users)	$\Sigma_{i=1}^{n\ ※1}$（リソースアダプタiのコネクションプールの最大値※2 ×2※3＋1※4）＋α※5
Oracle	最大同時接続数(PROCESSES)	$\Sigma_{i=1}^{n\ ※1}$（リソースアダプタiのコネクションプールの最大値※2 ＋1※4）＋α※5

注※ 1

n は，システム内の J2EE サーバにデプロイするリソースアダプタの総和です。

注※ 2

Connector 属性ファイルの MaxPoolSize パラメタの値を指定します。

注※ 3

次に示す条件に当てはまる場合に，× 2 を行ってください。

(a) HiRDB のバージョンが 07-01 以前の場合に次の操作をするとき

1. トランザクションサポートレベルに XATransaction を使用する。
2. トランザクション外でコネクションを使用してデータベースにアクセスする。

(b) HiRDB のバージョンが 07-02 以降の場合に次の操作をするとき

1. トランザクションサポートレベルに XATransaction を使用する。
2. アプリケーションサーバが管理するトランザクション内でコネクション※を使ってデータベースにアクセスする。
3. 2. のトランザクションが決着する前に，トランザクション外でコネクション※を使ってデータベースにアクセスする。

注※　このコネクションは 1. の DB Connector から取得したコネクションで，かつ同一コネクションです。

注※ 4

トランザクションサポートレベルに XATransaction を指定しているリソースアダプタの場合に，＋ 1 を行ってください。

注※ 5

＋ α は，一時的にコネクションプールの最大値をオーバーするおそれのあるコネクションを指します。詳細を次に示します。

- コネクションの障害検知機能を使用する場合
  コネクションの障害検知機能を使用する場合，コネクションプールから取り除いた未使用のコネクションは，コネクションプール内のコネクション数としてカウントされません。そのため，コネクションプール内のコネクションとコネクションプールから取り除いた未使用のコネクションの総和が，コネクションプールのコネクション数の最大値を一時的に超えることがあります。
- cjclearpool コマンドを使用する場合
  通常モードの場合，コネクションプールから取り除いた使用中のコネクションは，コネクション数としてカウントされません。そのため，コネクションプール内のコネクションとコネクションプールから取り除いた使用中のコネクションの総和がコネクションプールの最大値を超えることがあります。

5.1.5 運用管理サーバの使用リソース

運用管理サーバの使用リソースの見積もりについて，OS ごとに説明します。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

（1） AIX の場合

運用管理サーバの使用リソースの見積もりについて，次の表に示します。

表 5-15 運用管理サーバの使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
プロセス数	－	5	－
スレッド数	－	56	－
ファイルディスクリプタ数	nofiles	43 ＋ J2EE サーバ数	/etc/security/limits

（凡例）－：該当しません。

（2） HP-UX の場合

運用管理サーバの使用リソースの見積もりについて，次の表に示します。

表 5-16 運用管理サーバの使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
プロセス数	nproc	5	kctune nproc=4200
スレッド数	nkthread	56	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	43 ＋ J2EE サーバ数	kctune nfile= 65536

（3） Linux の場合

運用管理サーバの使用リソースの見積もりについて，次の表に示します。

表 5-17 運用管理サーバの使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
プロセス数	threads-max ※	5	/proc/sys/kernel/threads-max

システムリソース	パラメタ	所要量	オプション設定ファイル例
スレッド数	threads-max ※	56	―
ファイルディスクリプタ数	fs.files-max	43 ＋ J2EE サーバ数	/proc/sys/fs/file-max

（凡例）―：該当しません。

注※

threads-max パラメタには，プロセス数とスレッド数の合計を指定してください。

（4） Solaris の場合

運用管理サーバの使用リソースの見積もりについて，次の表に示します。

表 5-18　運用管理サーバの使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
プロセス数	maxuprc	5	/etc/system
スレッド数	―	56	―
ファイルディスクリプタ数	rlim_fd_max	43 ＋ J2EE サーバ数	/etc/system

（凡例）―：該当しません。

5.1.6　メモリセッションフェイルオーバ機能を使用する場合の使用リソース

メモリセッションフェイルオーバ機能を使用する場合の，セッションフェイルオーバサーバマシンの使用リソースの見積もりについて，OS ごとに説明します。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

なお，メモリセッションフェイルオーバ機能を使用する場合，J2EE サーバが使用するメモリ量が増加します。J2EE サーバ単位のメモリ増加量を求める式を次に示します。

```
J2EE サーバ単位のメモリ増加量（単位：メガバイト）
  ＝ Web アプリケーション単位のメモリ増加量の合計＋ 1.5
```

Web アプリケーション単位のメモリ増加量の合計は，それぞれの Web アプリケーションで使用するメモリ増加量を合計したものです。Web アプリケーションで使用するメモリ

増加量は，次の式で求められます。

Web アプリケーションで使用するメモリ増加量（単位：メガバイト）
＝最大同時実行スレッド数※1 ×グローバルセッション情報の最大サイズ※2 × 1.5

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かどうかによって値が異なります。最大同時実行スレッド数に指定する値を次の表に示します。

表 5-19 最大同時実行スレッド数に指定する値

Web アプリケーション単位の同時実行スレッド数の設定	最大同時実行スレッド数に指定する値
設定あり	Web アプリケーション単位の最大同時実行スレッド数
設定なし	Web コンテナ単位の最大同時実行スレッド数

なお，Web アプリケーション単位の同時実行スレッド数については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「2.17 Web アプリケーション単位での同時実行スレッド数の制御」を参照してください。

注※ 2

DD（web.xml）で設定したメモリセッションフェイルオーバ機能用の設定パラメタ（GSInfosLengthMax）の値です。GSInfosLengthMax パラメタについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「7. メモリセッションフェイルオーバ機能」を参照してください。

(1) AIX の場合

セッションフェイルオーバサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-20 セッションフェイルオーバサーバマシンの使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	－	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	－
プロセス数	－	4	－
スレッド数	－	SFO サーバのスレッド数※2 ＋ 34	－
ファイルディスクリプタ数	nofiles	SFO サーバのファイルディスクリプタ数※2 ＋ 43	/etc/security/limits

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

SFO サーバのスレッド数とファイルディスクリプタ数は，J2EE サーバと同じです。J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（2） HP-UX の場合

セッションフェイルオーバサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-21　セッションフェイルオーバサーバマシンの使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	kctune shmmax= 1073741824
プロセス数	nproc	4	kctune nproc=4200
スレッド数	nkthread	SFO サーバのスレッド数※2 ＋ 34	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	SFO サーバのファイルディスクリプタ数※2 ＋ 43	kctune nfile= 65536

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

SFO サーバのスレッド数とファイルディスクリプタ数は，J2EE サーバと同じです。J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（3） Linux の場合

セッションフェイルオーバサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-22 セッションフェイルオーバサーバマシンの使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	SHMMAX	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	/proc/sys/kernel/shmmax
プロセス数	threads-max※2	4	/proc/sys/kernel/threads-max
スレッド数	threads-max※2	SFO サーバのスレッド数※3 ＋ 34	－
ファイルディスクリプタ数	fs.file-max	SFO サーバのファイルディスクリプタ数※3 ＋ 43	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

threads-max パラメタには，プロセス数とスレッド数の合計を指定してください。

注※ 3

SFO サーバのスレッド数とファイルディスクリプタ数は，J2EE サーバと同じです。J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

（4） Solaris の場合

セッションフェイルオーバサーバマシンの使用リソースの見積もりについて，次の表に示します。

表 5-23 セッションフェイルオーバサーバマシンの使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shminfo_shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496	/etc/system
プロセス数	max_nprocs	4	/etc/system
スレッド数	－	SFO サーバのスレッド数※2 ＋ 34	－
ファイルディスクリプタ数	rlim_fd_max	SFO サーバのファイルディスクリプタ数※2 ＋ 43	/etc/system

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

SFO サーバのスレッド数とファイルディスクリプタ数は，J2EE サーバと同じです。J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

5.1.7　CTM を使用する場合の使用リソース

CTM を使用する場合の，使用リソースの見積もりについて，OS ごとに説明します。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

（1）AIX の場合

CTM を使用する場合の，使用リソースの見積もりについて次の表に示します。

表 5-24　CTM 使用時の使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	－	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	－
プロセス数	－	7 ＋ J2EE サーバ数※3	－
スレッド数	－	72 ＋ (J2EE サーバのスレッド数※4 ＋ 7) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	－
ファイルディスクリプタ数	nofiles	88 ＋ (J2EE サーバのファイルディスクリプタ数※4 ＋ 6) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	/etc/security/limits

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「5.1.7(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「5.1.7(1)(b)　CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

(a) 共用メモリ用ファイルサイズの計算式

共用メモリ用ファイルサイズを算出するには，CTM ドメインマネジャの共用メモリおよび CTM デーモンの共用メモリを算出する必要があります。それぞれの計算式について次に示します。

なお，計算式中の可変値には，次の値を使用してください。「ctm.」で始まるパラメタについては，マニュアル「Cosminexus 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「4.18　論理 CTM で指定できるパラメタ」を参照してください。

計算式に使用する値

-CTMMaxCTM：64
-CTMQueueCount：ctm.QueueCount
-CTMClientConnectCount：256
-CTMServerConnectCount：ctm.ServerConnectCount
-CTMEntryCount：-CTMClientConnectCount ＋ -CTMServerConnectCount
-CTMServerCacheSize：ctm.ServerCacheSize
-CTMQueueRegistCount：ctm.QueueRegistCount
-CTMDispatchParallelCount：ctm.DispatchParallelCount

● CTM ドメインマネジャの共用メモリ用ファイルサイズの計算式

CTM ドメインマネジャの共用メモリ用ファイルサイズの計算式を次に示します。

共用メモリ用ファイルサイズ（単位：バイト）＝
1,018,320 ＋（2,362 × -CTMMaxCTM 指定値）

● CTM デーモンの共用メモリ用ファイルサイズの計算式

CTM デーモンの場合は，CTM デーモン単位で固定長の共用メモリ用ファイルと可変長の共用メモリ用ファイルを確保する必要があります。それぞれの計算式を次に示します。

固定長の共用メモリ用ファイルサイズ（単位：バイト）=
551,840 ＋ （1,208 × -CTMQueueCount 指定値）

可変長の共用メモリ用ファイルサイズ（単位：バイト）=
1,027,008
＋（928 × -CTMClientConnectCount 指定値）
＋（256 × -CTMServerConnectCount 指定値）
＋（512 × -CTMEntryCount 指定値）
＋（1,024 × -CTMServerCacheSize 指定値）
＋（512 × -CTMQueueCount 指定値）
＋（544 × -CTMQueueCount 指定値 × -CTMQueueRegistCount 指定値）
＋（512 × -CTMDispatchParallelCount 指定値）

(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式

スレッド数およびファイルディスクリプタ数を算出するには，CTM デーモンで必要とするスレッド数とファイルディスクリプタ数を算出する必要があります。それぞれの計算式について次に示します。

● CTM デーモンで必要とするスレッド数の計算式

最大値 =
（A × 4 ＋ B × 3 ＋ C × 2 ＋ D × E ＋ F ＋ G ＋ 32）/ 0.8
（凡例）
- A：-CTMMaxCTM 値（ctmd が属する ctmdmd で指定された値）
- B：-CTMClientConnectCount 値
- C：-CTMServerConnectCount 値
- D：-CTMQueueCount 値
- E：-CTMQueueRegistCount 値
- F：-CTMDispatchParallelCount 値
- G：Create を発行する EJB クライアントの総数

● CTM デーモンで必要とするファイルディスクリプタ数の計算式

最大値 =
（A × 2 ＋ B × 4 ＋ C × 2 ＋ D × E ＋ F × EJB のインタフェース数＋ G ＋ 100）/ 0.8
（凡例）
- A：-CTMMaxCTM 値（ctmd が属する ctmdmd で指定された値）
- B：-CTMClientConnectCount 値

- C：-CTMServerConnectCount 値
- D：-CTMQueueCount 値
- E：-CTMQueueRegistCount 値
- F：-CTMDispatchParallelCount 値
- G：Create を発行する EJB クライアントの総数

（2） HP-UX の場合

CTM を使用する場合の，使用リソースの見積もりについて次の表に示します。

表 5-25　CTM 使用時の使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	kctune shmmax= 1073741824
プロセス数	nproc	7 ＋ J2EE サーバ数※3	kctune nproc=4200
スレッド数	nkthread	72 ＋ (J2EE サーバのスレッド数※4 ＋ 7) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	88 ＋ (J2EE サーバのファイルディスクリプタ数※4 ＋ 6) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	kctune nfile= 65536

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「5.1.7(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「5.1.7(1)(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

（3） Linux の場合

CTM を使用する場合の，使用リソースの見積もりについて，次の表に示します。

表 5-26　CTM 使用時の使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	SHMMAX	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	/proc/sys/kernel/shmmax
プロセス数	threads-max	7 ＋ J2EE サーバ数※3	/proc/sys/kernel/threads-max
スレッド数	threads-max	72 ＋ (J2EE サーバのスレッド数※4 ＋ 7) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	－
ファイルディスクリプタ数	fs.file-max	88 ＋ (J2EE サーバのファイルディスクリプタ数※4 ＋ 6) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「5.1.7(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「5.1.7(1)(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

（4） Solaris の場合

CTM を使用する場合の，使用リソースの見積もりについて次の表に示します。

表 5-27　CTM 使用時の使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shminfo_shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	/etc/system
プロセス数	max_nprocs	7 ＋ J2EE サーバ数※3	/etc/system
スレッド数	－	72 ＋ (J2EE サーバのスレッド数※4 ＋ 7) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	－
ファイルディスクリプタ数	rlim_fd_max	88 ＋ (J2EE サーバのファイルディスクリプタ数※4 ＋ 6) × J2EE サーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	/etc/system

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「5.1.7(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

J2EE サーバのスレッド数とファイルディスクリプタ数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「5.1.7(1)(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

5.2　プロセスごとに使用するリソース

この節では，アプリケーションサーバの各プロセスで使用するリソースの所要量の見積もりについて説明します。

5.2.1　J2EE サーバが使用するリソースの見積もり

ここでは，J2EE サーバプロセスのスレッド数とファイルディスクリプタ数の見積もりについて説明します。

参考

SFO サーバのスレッド数とファイルディスクリプタ数は，J2EE サーバと同じです。J2EE サーバの計算式を使用して算出してください。

（1）スレッド数

スレッド数の計算式を次に示します。(a) と (b) の合計が，J2EE サーバが使用するスレッド数です。

（a）基本のスレッド数

次の計算式で算出してください。

● **CORBA ネーミングサービスをインプロセスで起動し，インプロセス HTTP サーバを使用する場合**
最大スレッド数 =（65 ＋ A ＋ B ＋ 3 × C ＋ D ＋ 5 × E ＋ 2 × F ＋ G ＋ H ＋ I ＋ K）/ 0.8

● **CORBA ネーミングサービスをインプロセスで起動し，Hitachi Web Server を使用する場合**
最大スレッド数 =（63 ＋ A ＋ B ＋ 3 × C ＋ D ＋ 5 × E ＋ 2 × F ＋ G ＋ H ＋ J ＋ K）/ 0.8

（凡例）

- A：デプロイ済みの Entity Bean，Stateless Session Bean，Stateful Session Bean の数の合計
- B：Message-driven Bean を使用する場合，Message-driven Bean の最大インスタンス数
- C：EJB サーバ側の最大同時実行数（接続する最大 EJB クライアント数）
- D：CORBA ネーミングサービスのスレッド数（ただし，CORBA ネーミングサービスをインプロセスで起動（usrconf.properties の

ejbserver.naming.startupMode キーに inprocess を指定）した場合だけ加算する）
- E：データベースコネクションの数
- F：リソースアダプタ数
- G：簡易 Web サーバへの同時接続クライアント数（ただし，簡易 Web サーバへの同時接続クライアント数が 5 以下の場合は 5，100 以上の場合は 100 を指定する）
- H：デプロイ済みの Web アプリケーションの数
- I：インプロセス HTTP サーバのスレッド数（インプロセス HTTP サーバ機能を利用する場合だけ加算する。このスレッド数は，usrconf.properties で制御できる）
- J：Web サーバの同時接続クライアント数（ただし，Web サーバの同時接続クライアント数が Web コンテナの最大同時実行スレッド数（usrconf.properties の webserver.connector.ajp13.max_threads キーに指定した値）以下の場合は，Web コンテナの最大同時実行スレッド数を指定する）
- K：次の式で算出した TP1 インバウンドアダプタのスレッド数（TP1 インバウンド連携機能を使用する場合だけ加算する。このスレッド数は Connector 属性ファイルで制御できる）
  4 ＋ TP1 インバウンドアダプタのプロパティ rpc_max_thread_count に指定したスレッド数
  ＋ TP1 インバウンドアダプタのプロパティ trn_max_thread_count に指定したスレッド数
  ＋ TP1 インバウンドアダプタと連携するデプロイ済みの Message-driven Bean（サービス）の数の合計
  ＋ TP1 インバウンドアダプタのプロパティ MaxTPoolSize に指定した数

（b）JavaVM のオプション指定に応じて使用するスレッド数

JavaVM のオプション指定に応じて，次の計算式で算出してください。A は，-XX:+UseParNewGC オプションを指定している場合だけ加算します。B は，-XX:+HitachiUseExplicitMemory オプションを指定した場合だけ加算します。

最大スレッド数 = A + B

（凡例）
- A：パラレルコピーガーベージコレクションで使用するスレッド数（-XX:ParallelGCThreads オプションに指定した値。このオプションの指定を省略した場合は，論理 CPU 数を基にした -XX:ParallelGCThreads オプションのデフォルト値。なお，J2EE サーバ起動時の論理 CPU 数によって決定されるため，起動後に論理 CPU の数を変更してもスレッド数は変化しない）
- B：明示管理ヒープ機能で使用するスレッド数（論理 CPU 数。ただし，論理プロセッサ数が 8 以上の場合は 8。なお，J2EE サーバ起動時の論理 CPU 数によって決定されるため，起動後に論理 CPU の数を変更してもスレッド数は変化しない）

JavaVM のオプションについては，マニュアル「Cosminexus 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）」の次の個所を参照してください。

- 19.5　Cosminexus で指定できる Java HotSpot VM のオプション
- -XX:[+|-]HitachiUseExplicitMemory（明示管理ヒープ機能オプション）

（2） ファイルディスクリプタ数

ファイルディスクリプタ数の計算式を次に示します。

最大ファイルディスクリプタ数 =（137 ＋ A ＋ B × 3 ＋ C ＋ D ＋ E × 2 ＋ F ＋ G ＋ H ＋ I ＋ J × 2 ＋ K）/ 0.8

（凡例）

- A：データベースコネクションの数
- B：EJB クライアントのプロセス数
- C：Web サーバの同時接続クライアント数（ただし，Web サーバの同時接続クライアント数が Web コンテナの最大同時実行スレッド数（usrconf.properties の webserver.connector.ajp13.max_threads キーに指定した値）より大きい場合は，Web サーバの同時接続クライアント数＋ 1 を指定する）
- D：Web コンテナの最大同時実行スレッド数
- E：簡易 Web サーバへの同時接続クライアント数
- F：インプロセス HTTP サーバ機能を有効にしている場合は 4，無効にしている場合は 0
- G：インプロセス HTTP サーバへの同時接続クライアント数
- H：次の式で算出した TP1 インバウンドアダプタが使用するファイルディスクリプタ数（TP1 インバウンド連携機能を使用する場合だけ加算する。なお，先頭で加算している固定値は，TP1 インバウンド連携機能の内部のファイルディスクリプタ数）

  JDK 6 かつ，Linux の場合

  12 ＋ TP1 インバウンドアダプタのプロパティ max_connections に指定した値
  ＋ TP1 インバウンドアダプタのプロパティ trn_max_connections に指定した値
  ＋各 MDB（サービス）の Message-driven Bean 属性ファイルの
  <pooled-instance><maximum> に指定した値の総和× 3
  ＋ TP1 インバウンドアダプタのプロパティ rpc_max_thread_count に指定したスレッド数× 3
  ＋ TP1 インバウンドアダプタのプロパティ trn_max_thread_count に指定したスレッド数× 3

  上記以外の場合

  8 ＋ TP1 インバウンドアダプタのプロパティ max_connections に指定した値
  ＋ TP1 インバウンドアダプタのプロパティ trn_max_connections に指定した値
  ＋各 MDB（サービス）の Message-driven Bean 属性ファイルの

<pooled-instance><maximum> に指定した値の総和× 2
＋ TP1 インバウンドアダプタのプロパティ rpc_max_thread_count に指定したスレッド数× 2
＋ TP1 インバウンドアダプタのプロパティ trn_max_thread_count に指定したスレッド数× 2

- I：J2EE アプリケーションに含む JAR ファイルの数
- J：リソースアダプタ数
- K：usrconf.cfg の add.class.path キーに指定した JAR ファイルの数

（3） CORBA ネーミングサービス（インプロセス起動時）のスレッド数の見積もり

CORBA ネーミングサービスを J2EE サーバ起動時にインプロセスで起動させる場合に，J2EE サーバ上で生成される CORBA ネーミングサービスのスレッド数の見積もりについて説明します。

インプロセスで起動する場合の CORBA ネーミングサービスのスレッド数は，次のように見積もります。

合計スレッド数 = 初期化時に生成されるスレッド数＋ワーカスレッド数

（a） 初期化時に生成されるスレッド数

初期化時に生成されるスレッド数は，usrconf.properties の vbroker.agent.enableLocator キーの値が true の場合は 6，false の場合は 4 です。なお，vbroker.agent.enableLocator キーは，CTM 連携機能を有効（ejbserver.ctm.enabled キーに true を指定）にした場合，自動的に true が設定されます。

（b） ワーカスレッド数

ワーカスレッド数は，「同時受け付けリクエスト数＋ 1」と「クライアントと CORBA ネーミングサービス間のコネクション数」の合計になります。

ワーカスレッド数に関連するキーを次に示します。

- vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMax
- vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMin
- vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMaxIdle

これらのキーは，usrconf.properties の ejbserver.naming.exec.args キーの値として指定します。これらのキーの詳細については，マニュアル「Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド」，およびマニュアル「Borland(R) Enterprise Server VisiBroker(R) プログラマーズリファレンス」を参照してください。

vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMax キーで最大値を指定している場合のワーカスレッド数は，「このキーで指定した最大値」と「クライアントと CORBA ネー

ミングサービス間のコネクション数」の合計になります。

ただし，vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMin キーでワーカスレッド数の最小値を指定している場合，ワーカスレッド合計数が最小値に満たないときは，最小値がワーカスレッド数となります。

最大値を指定していない場合は，多重度の増加に伴いワーカスレッド数も増加していきます。ただし，ワーカスレッドは，アイドルになってから vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMaxIdle キー（デフォルト値は 300 秒）で指定した時間が経過したあとに消滅（30 秒の誤差があります）しますので，負荷が下がるとスレッド数も減少します。

ワーカスレッド数の最大値を指定している場合に，スレッド数とワーカスレッドの最大値が同じになったときは，これ以降のリクエスト受け付けはエラー扱いにはしないで，次のように処理を継続します。

- 受信済みのリクエストについては処理を継続します。
- 新規のリクエストはソケットから read() されないで，TCP の受信バッファ，クライアント側の送信バッファで滞留します。TCP のバッファがいっぱいの場合は，クライアント側で送信待ちとなります。

処理中だったワーカスレッドが空きになった（応答を返した）時点で，次のリクエストの受信処理が行われます。

5.2.2　運用管理エージェントが使用するリソースの見積もり

運用管理エージェントが使用するリソースの見積もりについて OS ごとに説明します。

（1）Windows の場合

Windows を使用する場合のスレッド数の計算式を次に示します。

● スレッド数の計算式

使用スレッド数 = 30 + 7 × 論理サーバの数※

注※　論理 CTM は論理サーバの数を 2 で計算してください。

（凡例）

- 30：運用管理エージェント本体が使用するスレッド数
- 7：論理サーバ一つ当たりの運用管理エージェントが使用するスレッド数

論理サーバのステータスが稼働になったあとのスレッド数の計算式を次に示します。

● 平時のスレッド数の計算式

使用スレッド数 = 30 + 5 × 論理サーバの数※

注※　論理 CTM は論理サーバの数を 2 で計算してください。

（凡例）

- 30：運用管理エージェント本体が使用するスレッド数

- 5：論理サーバ一つ当たりの運用管理エージェントが使用するスレッド数

（2） UNIX の場合

UNIX を使用する場合のスレッド数とファイルディスクリプタ数の計算式について説明します。

（a） スレッド数

スレッド数の計算式を次に示します。

● スレッド数の計算式

使用スレッド数 = 30 + 5 × **論理サーバの数**※

注※　論理 CTM は論理サーバの数を 2 で計算してください。

（凡例）

- 30：運用管理エージェント本体が使用するスレッド数
- 5：論理サーバ一つ当たりの運用管理エージェントが使用するスレッド数

論理サーバのステータスが稼働になったあとのスレッド数の計算式を次に示します。

● 平時のスレッド数の計算式

使用スレッド数 = 30 + 5 × **論理サーバの数**※

注※　論理 CTM は論理サーバの数を 2 で計算してください。

（凡例）

- 30：運用管理エージェント本体が使用するスレッド数
- 5：論理サーバ一つ当たりの運用管理エージェントが使用するスレッド数

（b） ファイルディスクリプタ数

ファイルディスクリプタ数の計算式を次に示します。

● ファイルディスクリプタ数の計算式

使用ファイルディスクリプタ数 = 20 + **論理サーバを構成するプロセスの数** × 6

（凡例）

- 20：運用管理エージェント本体が使用するファイルディスクリプタ数
- 6：論理サーバを構成する 1 プロセス当たりで運用管理エージェントが使用するファイルディスクリプタ数

論理サーバのステータスが稼働になったあとのファイルディスクリプタ数の計算式を次に示します。

● 平時のファイルディスクリプタ数の計算式

使用ファイルディスクリプタ数 = 20 + **論理サーバを構成するプロセスの数** × 3

（凡例）

- 20：運用管理エージェント本体が使用するファイルディスクリプタ数
- 3：論理サーバを構成する 1 プロセス当たりで運用管理エージェントが使用するファ

イルディスクリプタ数

5.2.3 パフォーマンストレーサが使用するリソースの見積もり

パフォーマンストレーサが使用するリソースの見積もりについて，OS ごとに説明します。

（1） Windows の場合

Windows を使用する場合の，パフォーマンストレーサが使用するリソースの見積もりについて説明します。

（a） 共用メモリ用ファイルサイズ

パフォーマンストレーサが使用する共用メモリ用ファイルサイズ（単位：バイト）は，PRF デーモンごとに算出します。計算式を次に示します。

● PRF デーモンごとの共用メモリの計算式

共用メモリ用ファイルサイズ = -PrfTraceBufferSize **指定値**× 1,024 + 18,496

（b） %PRFSPOOL% のディスク占有量

%PRFSPOOL% のディスク占有量の計算式を次に示します。

● %PRFSPOOL% のディスク占有量の計算式

ディスク占有量 = 2.0MB
＋ {(-PrfTraceBufferSize **指定値**＋ 20KB）× 4
＋ -PrfTraceFileSize **指定値**× -PrfTraceCount **指定値**× 2} × n
＋ 224KB × m
＋ 1,120KB ×（64 ＋ p）

（凡例）

- n：PRF デーモンの数
- m：起動中のユーザアプリケーションのプロセス数と正常終了しなかったユーザアプリケーションのプロセス数
  パフォーマンストレーサでは，ユーザアプリケーションごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されますが，プロセスが正常終了しない場合はファイルが残ります。ファイルの削除処理は PRF デーモン起動時，および PRF デーモン起動後 24 時間ごとに実施しますが，256 ファイルは削除されないで残ります。このため，最大ファイル数は「256 ＋起動中プロセス数」になります。
- p：起動中のシステム系プロセス数
  パフォーマンストレーサでは，プロセスごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されます。PRF デーモン起動時，および PRF デーモン起動後 24 時間ごとにファイル削除処理を実施し

ますが，64 ファイルは削除されないで残ります。このため，最大ファイル数は「64＋起動中プロセス数」になります。

上記のディスク容量は目安です。このため，十分な余裕を持って，%PRFSPOOL% を作成してください。

（2） AIX の場合

AIX を使用する場合，パフォーマンストレーサを使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，パフォーマンストレーサのプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

（a） 共用メモリ用ファイルサイズ

パフォーマンストレーサが使用する共用メモリ用ファイルサイズ（単位：バイト）は，PRF デーモンごとに算出します。計算式を次に示します。

● PRF デーモンごとの共用メモリの計算式

共用メモリ用ファイルサイズ = -PrfTraceBufferSize **指定値**× 1,024 + 18,496

共用メモリ用ファイルサイズは，環境変数 EXTSHM で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。

（b） ファイルディスクリプタ数

ファイルディスクリプタ数は，/etc/security/limits ファイルの「nofiles」で設定します。PRF デーモン起動時に使用するファイルディスクリプタの数は，32 以上に設定してください。

（c） $PRFSPOOL のディスク占有量

$PRFSPOOL のディスク占有量の計算式を次に示します。

● $PRFSPOOL のディスク占有量の計算式

ディスク占有量 = 2.0MB
＋｛(-PrfTraceBufferSize **指定値**＋ 20KB）× 4
＋ -PrfTraceFileSize **指定値**× -PrfTraceCount **指定値**× 2｝× n
＋ 224KB × m
＋ 1,120KB ×（64 ＋ p）

（凡例）

- n：PRF デーモンの数
- m：起動中のユーザアプリケーションのプロセス数と正常終了しなかったユーザアプリケーションのプロセス数
  パフォーマンストレーサでは，ユーザアプリケーションごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されま

すが，プロセスが正常終了しない場合はファイルが残ります。ファイルの削除処理は PRF デーモン起動時，および PRF デーモン起動後 24 時間ごとに実施しますが，256 ファイルは削除されないで残ります。このため，最大ファイル数は「256 ＋起動中プロセス数」になります。

- p：起動中のシステム系プロセス数
  パフォーマンストレーサでは，プロセスごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されます。PRF デーモン起動時，および PRF デーモン起動後 24 時間ごとにファイル削除処理を実施しますが，64 ファイルは削除されないで残ります。このため，最大ファイル数は「64 ＋起動中プロセス数」になります。

上記のディスク容量は目安です。このため，十分な余裕を持って，$PRFSPOOL を作成してください。

（3） HP-UX の場合

HP-UX を使用する場合，パフォーマンストレーサを使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，パフォーマンストレーサのプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

（a） 共用メモリ用ファイルサイズ

パフォーマンストレーサが使用する共用メモリ用ファイルサイズ（単位：バイト）は，PRF デーモンごとに算出します。計算式を次に示します。

● PRF デーモンごとの共用メモリの計算式
共用メモリ用ファイルサイズ = -PrfTraceBufferSize **指定値**× 1,024 + 18,496

共用メモリ用ファイルサイズは，Kernel Configuration の「shmmax」で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。

（b） ファイルディスクリプタ数

ファイルディスクリプタ数は，Kernel Configuration の「maxfiles」で設定します。PRF デーモン起動時に使用するファイルディスクリプタの数は，32 以上に設定してください。

（c） $PRFSPOOL のディスク占有量

$PRFSPOOL のディスク占有量の計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

（4） Linux の場合

Linux を使用する場合，パフォーマンストレーサを使用するために，次の値を考慮して

カーネルパラメタを設定する必要があります。正しく設定できていない場合には，パフォーマンストレーサのプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

（a） 共用メモリ用ファイルサイズ

パフォーマンストレーサが使用する共用メモリ用ファイルサイズ（単位：バイト）は，PRF デーモンごとに算出します。計算式を次に示します。

● PRF デーモンごとの共用メモリの計算式

共用メモリ用ファイルサイズ = -PrfTraceBufferSize **指定値**× 1,024 + 18,496

共用メモリ用ファイルサイズは，/etc/sysctl.conf ファイルの「kernel.shmmax」で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。

（b） ファイルディスクリプタ数

ファイルディスクリプタ数は，/etc/security/limits.conf ファイルの「nofiles」で設定します。PRF デーモン起動時に使用するファイルディスクリプタの数は，32 以上に設定してください。

（c） $PRFSPOOL のディスク占有量

$PRFSPOOL のディスク占有量の計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

（5） Solaris の場合

Solaris を使用する場合，パフォーマンストレーサを使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，パフォーマンストレーサのプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

（a） 共用メモリ用ファイルサイズ

パフォーマンストレーサが使用する共用メモリ用ファイルサイズ（単位：バイト）は，PRF デーモンごとに算出します。計算式を次に示します。

● PRF デーモンごとの共用メモリの計算式

共用メモリ用ファイルサイズ = -PrfTraceBufferSize **指定値**× 1,024 + 18,496

共用メモリ用ファイルサイズは，/etc/system ファイルの「shmsys:shminfo_shmmax」で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。

(b) ファイルディスクリプタ数

ファイルディスクリプタ数は，/etc/system ファイルの「rlim_fd_max」で設定します。PRF デーモン起動時に使用するファイルディスクリプタの数は，32 以上に設定してください。

(c) $PRFSPOOL のディスク占有量

$PRFSPOOL のディスク占有量の計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

5.2.4 CTM が使用するリソースの見積もり

CTM が使用するリソースの見積もりについて，OS ごとに説明します。なお，バッチアプリケーションの実行環境では CTM は使用できません。

(1) Windows の場合

Windows を使用する場合に CTM が使用するリソースの見積もりについて説明します。

(a) 共用メモリ用ファイルサイズ

CTM で使用する共用メモリ用ファイルサイズ（単位：バイト）の計算式について説明します。CTM デーモンの場合は，複数の共用メモリを確保する必要があります。CTM デーモン単位で固定長の共用メモリファイルと，可変長の共用メモリファイルがあります。それぞれのファイルサイズの計算式を次に示します。

● CTM ドメインマネジャの共用メモリ用ファイルサイズの計算式
共用メモリ用ファイルサイズ = 1,018,320 ＋（2,362 × -CTMMaxCTM **指定値**）

● CTM デーモンの共用メモリ用ファイルサイズの計算式
CTM デーモンの場合は，CTM デーモン単位で固定長の共用メモリ用ファイルと可変長の共用メモリ用ファイルを確保する必要があります。

- **固定長の共用メモリ用ファイルサイズ** =
  551,840 ＋ （1,208 × -CTMQueueCount **指定値**）
- **可変長の共用メモリ用ファイルサイズ** =
  1,027,008
  ＋（928 × -CTMClientConnectCount **指定値**）
  ＋（256 × -CTMServerConnectCount **指定値**）
  ＋（512 × -CTMEntryCount **指定値**）
  ＋（1,024 × -CTMServerCacheSize **指定値**）
  ＋（512 × -CTMQueueCount **指定値**）
  ＋（544 × -CTMQueueCount **指定値** × -CTMQueueRegistCount **指定値**）
  ＋（512 × -CTMDispatchParallelCount **指定値**）

（b） 稼働統計情報ファイルサイズ

稼働統計情報ファイルサイズ（単位：バイト）の計算式を次に示します。

● オンライン開始から終了までに必要な稼働統計情報ファイルサイズの計算式

ファイルサイズ = A ＋ B

（凡例）

- A：（オンライン開始から終了までに実行されるリクエストの数）×（1 リクエストで出力される情報量）
- B：（オンライン開始から終了までの時間（分）／ ctmstsstart -CTMInterval に指定した取得間隔）×（1 回に出力される CTM ノード単位，キュー単位統計情報量）

1 リクエストで出力される情報量と，1 回に出力される CTM ノード単位，キュー単位統計情報量の計算式を次に示します。

● 1 リクエストで出力される情報量の計算式

情報量 = （↑（80 ＋ A ＋ B ＋ C ＋ D ＋ 63）/ 64 ↑ × 64）× 3

（凡例）

- A：リクエストを実行するアプリケーションを管理している CTM ドメイン名のドメイン長
- B：リクエストを実行するアプリケーションを管理している CTM デーモンの CTM 識別子の長さ
- C：キュー名称の長さ
- D：オペレーション名称の長さ

● 1 回に出力される CTM ノード単位，キュー単位統計情報量の計算式

統計情報量 = ↑（2,144 ＋ 344 × **キュー数**＋ 63）/ 64 ↑ × 64

（c） %CTMSPOOL% のディスク占有量

%CTMSPOOL% のディスク占有量の計算式を次に示します。

● %CTMSPOOL% のディスク占有量の計算式

ディスク占有量 = 7.0MB
＋（18.5MB ＋ 1.0MB × -CTMLogFileSize **指定値**× -CTMLogFileCount **指定値**）× n
＋（1KB ＋ 0.5KB × k）×（m ＋ l）
＋ 1KB × m × j
＋ 224KB × p
＋ 1,120KB ×（64 ＋ q）
＋ CTM **ドメインマネジャの共有メモリ用ファイルサイズ**× 5
＋ CTM **デーモンの共有メモリ用ファイルサイズ**× 5 × n
＋ CTM **ドメインマネジャの** core **サイズ**
＋ CTM **デーモンの** core **サイズ**× n
＋ CTM **レギュレータの** core **サイズ**× 3 × m

＋ OTM ゲートウェイの core サイズ× 3 × l
＋（-CTMStatsFileSize 指定値× -CTMStatsFileCount 指定値）× n

（凡例）

- j：EJB の総数
- k：CTM レギュレータおよび OTM ゲートウェイに接続できるクライアント数（-CTMClientConnectCount オプション指定値）
- l：OTM ゲートウェイの総数
- m：CTM レギュレータの総数
- n：CTM デーモンの数
- p：起動中のユーザアプリケーションのプロセス数と正常終了しなかったユーザアプリケーションのプロセス数
  CTM では，ユーザアプリケーションごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されますが，プロセスが正常終了しない場合はファイルが残ります。ファイルの削除処理は CTM ドメインマネジャ起動時，および CTM ドメインマネジャ起動後 24 時間ごとに実施しますが，256 ファイルは削除されないで残ります。このため，最大ファイル数は「256 ＋起動中プロセス数」になります。
- q：起動中のシステム系プロセス数
  CTM では，プロセスごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されます。CTM ドメインマネジャ起動時，および CTM ドメインマネジャ起動後 24 時間ごとにファイル削除処理を実施しますが，64 ファイルは削除されないで残ります。このため，最大ファイル数は「64 ＋起動中プロセス数」になります。

上記のディスク容量は目安です。このため，十分な余裕を持って，%CTMSPOOL% を作成してください。

（2） AIX の場合

AIX を使用する場合，CTM を使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，CTM のプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

（a） 共用メモリ用ファイルサイズ

共用メモリ用ファイルサイズは，環境変数 EXTSHM で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。なお，計算式については，Windows の場合と同じです。Windows の計算式を参照してください。

（b） 稼働統計情報ファイルサイズ

稼働統計情報ファイルサイズの計算式については，Windows の場合と同じです。

Windows の計算式を参照してください。

（c） ファイルディスクリプタ数

CTM デーモンでは，起動時のオプションを基に，次の計算式で示すように，プロセスで使用できるファイルディスクリプタの数を増加させます。OS の設定値が，計算式で設定した CTM デーモンで必要とするファイルディスクリプタの最大値に満たない場合には，プロセス起動でエラー終了します。計算式を基に /etc/security/limits.conf ファイルの「nofiles」を変更してください。

● **CTM デーモンで必要とするファイルディスクリプタ数の最大値の計算式**

最大値 =（A × 2 ＋ B × 4 ＋ C × 2 ＋ D × E ＋ F × EJB **数**＋ G ＋ 100）/ 0.8

（凡例）

- A：-CTMMaxCTM 値（ctmd が属する ctmdmd で指定された値）
- B：-CTMClientConnectCount 値
- C：-CTMServerConnectCount 値
- D：-CTMQueueCount 値
- E：-CTMQueueRegistCount 値
- F：-CTMDispatchParallelCount 値
- G：Create を発行する EJB クライアントの総数

（d） $CTMSPOOL のディスク占有量

$CTMSPOOL のディスク占有量の計算式を次に示します。

● **$CTMSPOOL のディスク占有量の計算式**

ディスク占有量 = 7.0MB

＋（18.5MB ＋ 1.0MB × -CTMLogFileSize **指定値**× -CTMLogFileCount **指定値**）× n

＋（1KB ＋ 0.5KB × k）×（m ＋ l）

＋ 1KB × m × j

＋ 224KB × p

＋ 1,120KB ×（64 ＋ q）

＋ CTM **ドメインマネジャの共有メモリ用ファイルサイズ**× 5

＋ CTM **デーモンの共有メモリ用ファイルサイズ**× 5 × n

＋ CTM **ドメインマネジャの** core **サイズ**

＋ CTM **デーモンの** core **サイズ**× n

＋ CTM **レギュレータの** core **サイズ**× 3 × m

＋ OTM **ゲートウェイの** core **サイズ**× 3 × l

＋（-CTMStatsFileSize **指定値**× -CTMStatsFileCount **指定値**）× n

（凡例）

- j：EJB の総数
- k：CTM レギュレータおよび OTM ゲートウェイに接続できるクライアント数

（-CTMClientConnectCount オプション指定値）

- l：OTM ゲートウェイの総数
- m：CTM レギュレータの総数
- n：CTM デーモンの数
- p：起動中のユーザアプリケーションのプロセス数と正常終了しなかったユーザアプリケーションのプロセス数
  CTM では，ユーザアプリケーションごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されますが，プロセスが正常終了しない場合はファイルが残ります。ファイルの削除処理は CTM ドメインマネジャ起動時，および CTM ドメインマネジャ起動後 24 時間ごとに実施しますが，256 ファイルは削除されないで残ります。このため，最大ファイル数は「256 ＋起動中プロセス数」になります。
- q：起動中のシステム系プロセス数
  CTM では，プロセスごとに，保守情報として内部トレースをファイルに出力します。このファイルはプロセス起動時に作成されます。CTM ドメインマネジャ起動時，および CTM ドメインマネジャ起動後 24 時間ごとにファイル削除処理を実施しますが，64 ファイルは削除されないで残ります。このため，最大ファイル数は「64 ＋起動中プロセス数」になります。

上記のディスク容量は目安です。このため，十分な余裕を持って，$CTMSPOOL を作成してください。

（3） HP-UX の場合

HP-UX を使用する場合，CTM を使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，CTM のプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

(a) 共用メモリ用ファイルサイズ

共用メモリ用ファイルサイズは，Kernel Configuration の「shmmax」で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。なお，計算式については，Windows の場合と同じです。Windows の計算式を参照してください。

(b) 稼働統計情報ファイルサイズ

稼働統計情報ファイルサイズの計算式については，Windows の場合と同じです。
Windows の計算式を参照してください。

(c) ファイルディスクリプタ数

ファイルディスクリプタ数は，Kernel Configuration の「maxfiles」で設定します。計

算式を基に「maxfiles」を変更してください。なお，計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

(d) スレッド数

CTM デーモンでは，J2EE サーバの起動や J2EE アプリケーションの開始，および EJB クライアントからの要求に対してスレッドが作成されます。計算式を基に，プロセスで使用するスレッドの最大値を Kernel Configuration の「max_thread_proc」で変更してください。

● CTM デーモンで必要とするスレッド数の最大値の計算式

最大値 =（A × 4 ＋ B × 3 ＋ C × 2 ＋ D × E ＋ F ＋ G ＋ 32）/ 0.8

（凡例）

- A：-CTMMaxCTM 値（ctmd が属する ctmdmd で指定された値）
- B：-CTMClientConnectCount 値
- C：-CTMServerConnectCount 値
- D：-CTMQueueCount 値
- E：-CTMQueueRegistCount 値
- F：-CTMDispatchParallelCount 値
- G：Create を発行する EJB クライアントの総数

(e) $CTMSPOOL のディスク占有量

$CTMSPOOL のディスク占有量の計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

(4) Linux の場合

Linux を使用する場合，CTM を使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，CTM のプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

(a) 共用メモリ用ファイルサイズ

共用メモリ用ファイルサイズは，/etc/sysctl.conf ファイルの「kernel.shmmax」で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。なお，計算式については，Windows の場合と同じです。Windows の計算式を参照してください。

(b) 稼働統計情報ファイルサイズ

稼働統計情報ファイルサイズの計算式については，Windows の場合と同じです。
Windows の計算式を参照してください。

(c) ファイルディスクリプタ数

ファイルディスクリプタ数は，/etc/security/limits.conf ファイルの「nofiles」で設定します。計算式を基に「nofiles」を変更してください。なお，計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

(d) $CTMSPOOL のディスク占有量

$CTMSPOOL のディスク占有量の計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

(5) Solaris の場合

Solaris を使用する場合，CTM を使用するために，次の値を考慮してカーネルパラメタを設定する必要があります。正しく設定できていない場合には，CTM のプロセスが起動できなかったり，動作中にリソース不足で異常終了になったりするおそれがあります。なお，カーネルパラメタの設定については，使用している OS のマニュアルを参照してください。

(a) 共用メモリ用ファイルサイズ

共用メモリ用ファイルサイズは，/etc/system ファイルの「shmsys:shminfo_shmmax」で設定します。計算式で算出した値以上に共用メモリが割り当てられるように設定してください。なお，計算式については，Windows の場合と同じです。Windows の計算式を参照してください。

(b) 稼働統計情報ファイルサイズ

稼働統計情報ファイルサイズの計算式については，Windows の場合と同じです。
Windows の計算式を参照してください。

(c) ファイルディスクリプタ数

ファイルディスクリプタ数は，計算式を基に /etc/system ファイルの「rlim_fd_max」で設定します。計算式を基に「rlim_fd_max」を変更してください。なお，計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

(d) $CTMSPOOL のディスク占有量

$CTMSPOOL のディスク占有量の計算式については，AIX の場合と同じです。AIX の計算式を参照してください。

5.3　仮想メモリの使用量の見積もり

ここでは，仮想メモリの使用量の見積もり方法について説明します。なお，仮想メモリの使用量の計算式に使用している JavaVM 起動時のオプションの詳細については，「7.1.2　JavaVM で使用するメモリ空間の構成と JavaVM オプション」を参照してください。

また，ここで説明する内容のほか，明示管理ヒープ機能で使用するメモリサイズの見積もりが必要です。明示管理ヒープ機能で使用するメモリサイズの見積もりについては，「7.10　Explicit ヒープのチューニング」を参照してください。

（1）仮想メモリの使用量の計算式

仮想メモリの使用量（単位：メガバイト）の計算式を次に示します。

J2EE サーバの仮想メモリの使用量 = A ＋ B ＋ C ＋（D ＋ 10）× E ＋ F

（凡例）

- A：Java ヒープサイズ
  初期値は，JavaVM 起動時のオプション -Xms に指定した値です。この値は，J2EE サーバ起動中に，最大で JavaVM 起動時のオプション -Xmx に指定した値まで拡張されます。
- B：Permanent 領域サイズ
  初期値は，JavaVM 起動時のオプション -XX:PermSize に指定した値です。この値は，J2EE サーバ起動中に，最大で JavaVM 起動時のオプション -XX:MaxPermSize に指定した値まで拡張されます。
  監査ログを使用する場合
  監査ログを使用する場合は，値に 1 メガバイトを加えてください。
- C：ネイティブプログラム使用領域サイズ
  OS ごとに使用する値が異なります。ネイティブプログラム使用領域サイズの OS ごとの値について，次の表に示します。

表 5-28　ネイティブプログラム使用領域サイズの値一覧

OS 種別	使用領域の値 （単位：メガバイト）
Windows	300
AIX	400
HP-UX	400
Linux（x86）	400
Linux（AMD64 & Intel EM64T）	600

監査ログを使用する場合

監査ログを使用する場合は，OS ごとのネイティブプログラム使用領域サイズの値に，監査ログで使用する値を加える必要があります。監査ログで使用する値を算出するには，auditlog.raslog.message.filesize キー，および auditlog.raslog.exception.filesize キーを使用します。これらのキーは，auditlog.properties（監査ログ定義ファイル）で指定するキーです。これらのキーの詳細については，マニュアル「Cosminexus 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「13.2　監査ログ定義ファイル」を参照してください。
監査ログで使用する値（単位：メガバイト）の計算式を次に示します。

監査ログで使用する値 =

8 ＋（auditlog.raslog.message.filesize **の指定値** / $1{,}024^2$）※＋
（auditlog.raslog.exception.filesize **の指定値** / $1{,}024^2$）※

注※　指定値の単位はバイトです。仮想メモリの使用量の値の単位はメガバイトであるため，$1{,}024^2$ で割る必要があります。

- D：J2EE サーバのスレッド数
  J2EE サーバが使用するスレッド数です。J2EE サーバが使用するスレッド数については，「5.2.1　J2EE サーバが使用するリソースの見積もり」を参照してください。
- E：スタック領域サイズ
  JavaVM 起動時のオプション -Xss に指定した値です。
- F：Explicit ヒープサイズ
  明示管理ヒープ機能で使用される Explicit ヒープのサイズです。この値は，J2EE サーバ起動中に最大で -XX:HitachiExplicitHeapMaxSize に指定した値まで拡張されます。

（2）仮想メモリの使用量を計算する場合の注意事項

● メモリセッションフェイルオーバ機能を使用する構成の場合，SFO サーバの JavaVM の Old 領域，New 領域のサイズを J2EE サーバの仮想メモリの使用量に加える必要があります。詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「7.15.2　SFO サーバの起動オプションの設定」を参照してください。

● ネイティブプログラム使用領域サイズの値は変動します。値が変動するのは次のような場合です。
  - ORB のトレースサイズを変更した場合
  - JDBC ドライバの使用領域サイズの値を変更した場合
  - 使用する製品のネイティブライブラリ使用領域サイズの値を変更した場合

  このような場合，製品ごとのメモリ使用量をネイティブプログラム使用領域サイズの値に加えてください。製品ごとのメモリ使用量は，使用する製品のドキュメントに従って算出してください。

● J2EE サーバの標準構成で使用する仮想メモリの使用量は，使用するソフトウェア製

品の影響で，見積もり値よりも大きくなる場合があります。その場合の増加分については，仮想メモリの使用量の値に加えてください。増加分のメモリ使用量は，使用するソフトウェア製品のドキュメントに従い，製品ごとのメモリ使用量を算出してください。

6 使用するリソースの見積もり（バッチアプリケーション実行基盤）

この章では，バッチアプリケーションを実行するシステムで使用するリソース，および仮想メモリ使用量の見積もり方法について説明します。システムを動作させるために必要なディスクおよびメモリの容量を算出するときの参考にしてください。J2EE アプリケーション実行基盤の使用リソースおよび仮想メモリ所要量の見積もりについては，「5. 使用するリソースの見積もり（J2EE アプリケーション実行基盤）」を参照してください。

6.1 システム構成ごとに使用するリソース

システムが動作するためには，OS やデータベースなどのホスティング環境の設定が必要な場合があります。システムが必要とするリソースは，システムの構成ごとに異なるため，ここではシステム構成ごとに使用するリソースと，リソースの所要量の見積もりについて説明します。システム構成ごとに使用するリソースと，リソースの見積もりの参照先を次の表に示します。

表 6-1 システム構成ごとに使用するリソースと見積もりの参照先

システム構成ごとに使用するリソース	参照先
バッチサーバを配置する場合の使用リソース	6.1.1
データベースの使用リソース	6.1.2
CTM を使用する場合の使用リソース	6.1.3

6.1.1 バッチサーバを配置する場合の使用リソース

バッチサーバを配置する場合の使用リソースの見積もりについて，OS ごとに説明します。バッチサーバを配置する場合，バッチサーバを配置するマシンで使用するリソースを見積もります。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

また，プロセスごとに使用するリソースの見積もりについては，「6.2 プロセスごとに使用するリソース」を参照してください。

なお，仮想メモリ所要量については，「6.3 仮想メモリの使用量の見積もり」を参照してください。また，ディスク占有量については，アプリケーションサーバのリリースノートを参照してください。

（1） AIX の場合

AIX の場合の，アプリケーションサーバマシンの使用リソースの見積もりについて次の表に示します。

表 6-2　アプリケーションサーバマシンの使用リソースの見積もり（AIX の場合）

システムリソース		パラメタ	所要量	オプション設定ファイル例
サービスユニット※1	共用メモリ	－	PrfTraceBufferSize※2 × 1,024 ＋ 18,496	－
	プロセス数	－	4	－
	スレッド数	－	バッチサーバのスレッド数※3 ＋ 34	－
	ファイルディスクリプタ数	nofiles	バッチサーバのファイルディスクリプタ数※3 ＋ 43	/etc/security/limits
Management Server	プロセス数	－	5	－
	スレッド数	－	56	－
	ファイルディスクリプタ数	nofiles	43 ＋バッチサーバ数	/etc/security/limits

（凡例）－：該当しません。

注※ 1

サービスユニットとは，次のまとまりを指します。

バッチサーバ + パフォーマンストレーサ

注※ 2

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 3

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1　バッチサーバが使用するリソースの見積もり」を参照して算出してください。

(2) HP-UX の場合

HP-UX の場合の，アプリケーションサーバマシンの使用リソースの見積もりについて次の表に示します。

表 6-3 アプリケーションサーバマシンの使用リソースの見積もり（HP-UX の場合）

システムリソース		パラメタ	所要量	オプション設定ファイル例
サービスユニット※1	共用メモリ	shmmax	PrfTraceBufferSize※2 × 1,024 ＋ 18,496	kctune shmmax= 1073741824
	プロセス数	nproc	4	kctune nproc=4200
	スレッド数	nkthread	バッチサーバのスレッド数※3 ＋ 34	kctune nkthread= 8416
	ファイルディスクリプタ数	nfile	バッチサーバのファイルディスクリプタ数※3 ＋ 43	kctune nfile= 65536
Management Server	プロセス数	nproc	5	kctune nproc=4200
	スレッド数	nkthread	56	kctune nkthread= 8416
	ファイルディスクリプタ数	nfile	43 ＋バッチサーバ数	kctune nfile= 65536

注※ 1

サービスユニットとは，次のまとまりを指します。

バッチサーバ + パフォーマンストレーサ

注※ 2

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15 論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 3

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1 バッチサーバが使用するリソースの見積もり」を参照して算出してください。

(3) Linux の場合

Linux の場合の，アプリケーションサーバマシンの使用リソースの見積もりについて次の表に示します。

表 6-4　アプリケーションサーバマシンの使用リソースの見積もり（Linux の場合）

システムリソース		パラメタ	所要量	オプション設定ファイル例
サービスユニット※1	共用メモリ	SHMMAX	PrfTraceBufferSize※2 × 1,024 ＋ 18,496	/proc/sys/kernel/shmmax
	プロセス数	threads-max※3	4	/proc/sys/kernel/threads-max
	スレッド数	threads-max※3	バッチサーバのスレッド数※4 ＋ 34	－
	ファイルディスクリプタ数	fs.file-max	バッチサーバのファイルディスクリプタ数※4 ＋ 43	/proc/sys/fs/file-max
Management Server	プロセス数	threads-max※3	5	/proc/sys/kernel/threads-max
	スレッド数	threads-max※3	56	－
	ファイルディスクリプタ数	fs.files-max	43 ＋バッチサーバ数	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

サービスユニットとは，次のまとまりを指します。

バッチサーバ + パフォーマンストレーサ

注※ 2

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 3

threads-max パラメタには，プロセス数とスレッド数の合計を指定してください。

注※ 4

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1　バッチサーバが使用するリソースの見積もり」を参照して算出してください。

（4） Solaris の場合

Solaris の場合の，アプリケーションサーバマシンの使用リソースの見積もりについて次の表に示します。

表 6-5 アプリケーションサーバマシンの使用リソースの見積もり（Solaris の場合）

システムリソース		パラメタ	所要量	オプション設定ファイル例
サービスユニット※1	共用メモリ	shminfo_shmmax	PrfTraceBufferSize※2 × 1,024 ＋ 18,496	/etc/system
	プロセス数	max_nprocs	4	/etc/system
	スレッド数	－	バッチサーバのスレッド数※3 ＋ 34	－
	ファイルディスクリプタ数	rlim_fd_max	バッチサーバのファイルディスクリプタ数※3 ＋ 43	/etc/system
Management Server	プロセス数	maxuprc	5	/etc/system
	スレッド数	－	56	－
	ファイルディスクリプタ数	rlim_fd_max	43 ＋バッチサーバ数	/etc/system

（凡例）－：該当しません。

注※ 1

サービスユニットとは，次のまとまりを指します。

バッチサーバ + パフォーマンストレーサ

注※ 2

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15 論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 3

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1 バッチサーバが使用するリソースの見積もり」を参照して算出してください。

6.1.2 データベースの使用リソース

DBMS の使用リソースの見積もりについて説明します。

仮想メモリ所要量については，「6.3 仮想メモリの使用量の見積もり」を参照してください。また，ディスク占有量については，Application Server または Developer のリリースノートを参照してください。

DBMS の使用リソースの見積もりについて，次の表に示します。

表 6-6 DBMS の使用リソースの見積もり

DBMS	使用リソース	所要量
HiRDB	最大同時接続数(pd_max_users)	$\Sigma_{i=1}^{n}$※1（リソースアダプタiのコネクションプールの最大値※2 ×2※3＋1※4）＋α※5
Oracle	最大同時接続数(PROCESSES)	$\Sigma_{i=1}^{n}$※1（リソースアダプタiのコネクションプールの最大値※2 ＋1※4）＋α※5

注※ 1

n は，システム内のバッチサーバにデプロイするリソースアダプタの総和です。

注※ 2

Connector 属性ファイルの MaxPoolSize パラメタの値を指定します。

注※ 3

次に示す条件に当てはまる場合に，× 2 を行ってください。

(a) HiRDB のバージョンが 07-01 以前の場合に次の操作をするとき

1. トランザクションサポートレベルに XATransaction を使用する。
2. トランザクション外でコネクションを使用してデータベースにアクセスする。

(b) HiRDB のバージョンが 07-02 以降の場合に次の操作をするとき

1. トランザクションサポートレベルに XATransaction を使用する。
2. アプリケーションサーバが管理するトランザクション内でコネクション※を使ってデータベースにアクセスする。
3. 2. のトランザクションが決着する前に，トランザクション外でコネクション※を使ってデータベースにアクセスする。

注※ このコネクションは 1. の DB Connector から取得したコネクションで，かつ同一コネクションです。

注※ 4

トランザクションサポートレベルに XATransaction を指定しているリソースアダプタの場合に，＋ 1 を行ってください。

注※ 5

＋ α は，一時的にコネクションプールの最大値をオーバーするおそれのあるコネクションを指します。詳細を次に示します。

- コネクションの障害検知機能を使用する場合
  コネクションの障害検知機能を使用する場合，コネクションプールから取り除いた未使用のコネクションは，コネクションプール内のコネクション数としてカウントされません。そのため，コネクションプール内のコネクションとコネクションプールから取り除いた未使用のコネクションの総和が，コネクションプールの

コネクション数の最大値を一時的に超えることがあります。

- cjclearpool コマンドを使用する場合
  通常モードの場合，コネクションプールから取り除いた使用中のコネクションは，コネクション数としてカウントされません。そのため，コネクションプール内のコネクションとコネクションプールから取り除いた使用中のコネクションの総和がコネクションプールの最大値を超えることがあります。

6.1.3 CTM を使用する場合の使用リソース

CTM（バッチアプリケーションのスケジューリング機能）を使用する場合の，使用リソースの見積もりについて，OS ごとに説明します。

なお，使用リソースの見積もりの各表にある，「オプション設定ファイル例」については，使用している OS のバージョン，およびカーネルのバージョンごとに異なります。使用している OS のマニュアルを参照して，表中の見積もり式を基に見積もった値を設定してください。使用している OS で該当するカーネルパラメタが設定できない場合には，設定は不要です。

（1） AIX の場合

CTM を使用する場合の，使用リソースの見積もりについて次の表に示します。

表 6-7 CTM 使用時の使用リソースの見積もり（AIX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	－	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	－
プロセス数	－	7 ＋バッチサーバ数※3	－
スレッド数	－	72 ＋（バッチサーバのスレッド数※4 ＋ 7）×バッチサーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	－
ファイルディスクリプタ数	nofiles	88 ＋（バッチサーバのファイルディスクリプタ数※4 ＋ 6）×バッチサーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	/etc/security/limits

（凡例）－：該当しません。

注※1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15 論理パフォーマンストレーサで指定で

きるパラメタ」を参照してください。

注※ 2

値については，「6.1.3(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1　バッチサーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「6.1.3(1)(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

(a) 共用メモリ用ファイルサイズの計算式

共用メモリ用ファイルサイズを算出するには，CTM ドメインマネジャの共用メモリおよび CTM デーモンの共用メモリを算出する必要があります。それぞれの計算式について次に示します。

なお，計算式中の可変値には，次の値を使用してください。「ctm.」で始まるパラメタについては，マニュアル「Cosminexus 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「4.6　簡易構築定義ファイル」を参照してください。

計算式に使用する値

-CTMMaxCTM：64
-CTMQueueCount：ctm.QueueCount
-CTMClientConnectCount：256
-CTMServerConnectCount：ctm.ServerConnectCount
-CTMEntryCount：-CTMClientConnectCount ＋ -CTMServerConnectCount
-CTMServerCacheSize：ctm.ServerCacheSize
-CTMQueueRegistCount：ctm.QueueRegistCount
-CTMDispatchParallelCount：ctm.DispatchParallelCount

● CTM ドメインマネジャの共用メモリ用ファイルサイズの計算式

CTM ドメインマネジャの共用メモリ用ファイルサイズの計算式を次に示します。

共用メモリ用ファイルサイズ（単位：バイト）=
　1,018,320 ＋（2,362 × -CTMMaxCTM 指定値）

● CTM デーモンの共用メモリ用ファイルサイズの計算式

CTM デーモンの場合は，CTM デーモン単位で固定長の共用メモリ用ファイルと可変長の共用メモリ用ファイルを確保する必要があります。それぞれの計算式を次に示します。

固定長の共用メモリ用ファイルサイズ（単位：バイト）=
551,840 ＋（1,208 × -CTMQueueCount 指定値）

可変長の共用メモリ用ファイルサイズ（単位：バイト）=
1,027,008
＋（928 × -CTMClientConnectCount 指定値）
＋（256 × -CTMServerConnectCount 指定値）
＋（512 × -CTMEntryCount 指定値）
＋（1,024 × -CTMServerCacheSize 指定値）
＋（512 × -CTMQueueCount 指定値）
＋（544 × -CTMQueueCount 指定値 × -CTMQueueRegistCount 指定値）
＋（512 × -CTMDispatchParallelCount 指定値）

(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式

スレッド数およびファイルディスクリプタ数を算出するには，CTM デーモンで必要とするスレッド数とファイルディスクリプタ数を算出する必要があります。それぞれの計算式について次に示します。

● CTM デーモンで必要とするスレッド数の計算式

最大値 =
（A × 4 ＋ B × 3 ＋ C × 2 ＋ D × E ＋ F ＋ G ＋ 32）/ 0.8
（凡例）
- A：-CTMMaxCTM 値（ctmd が属する ctmdmd で指定された値）
- B：-CTMClientConnectCount 値
- C：-CTMServerConnectCount 値
- D：-CTMQueueCount 値
- E：-CTMQueueRegistCount 値
- F：-CTMDispatchParallelCount 値
- G：Create を発行する EJB クライアントの総数

● CTM デーモンで必要とするファイルディスクリプタ数の計算式

最大値 =
（A × 2 ＋ B × 4 ＋ C × 2 ＋ D × E ＋ F × EJB のインタフェース数＋ G ＋ 100）/ 0.8
（凡例）
- A：-CTMMaxCTM 値（ctmd が属する ctmdmd で指定された値）
- B：-CTMClientConnectCount 値
- C：-CTMServerConnectCount 値
- D：-CTMQueueCount 値
- E：-CTMQueueRegistCount 値
- F：-CTMDispatchParallelCount 値

- G：Create を発行する EJB クライアントの総数

(2) HP-UX の場合

CTM を使用する場合の，使用リソースの見積もりについて次の表に示します。

表 6-8　CTM 使用時の使用リソースの見積もり（HP-UX の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	kctune shmmax= 1073741824
プロセス数	nproc	7 ＋バッチサーバ数※3	kctune nproc=4200
スレッド数	nkthread	72 ＋ (バッチサーバのスレッド数※4 ＋ 7) ×バッチサーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	kctune nkthread= 8416
ファイルディスクリプタ数	nfile	88 ＋ (バッチサーバのファイルディスクリプタ数※4 ＋ 6) ×バッチサーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	kctune nfile= 65536

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「6.1.3(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1　バッチサーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「6.1.3(1)(b)　CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

(3) Linux の場合

CTM を使用する場合の，使用リソースの見積もりについて，次の表に示します。

表 6-9 CTM 使用時の使用リソースの見積もり（Linux の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	SHMMAX	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	/proc/sys/kernel/shmmax
プロセス数	threads-max	7 ＋バッチサーバ数※3	/proc/sys/kernel/threads-max
スレッド数	threads-max	72 ＋（バッチサーバのスレッド数※4 ＋ 7）×バッチサーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	－
ファイルディスクリプタ数	fs.file-max	88 ＋（バッチサーバのファイルディスクリプタ数※4 ＋ 6）×バッチサーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	/proc/sys/fs/file-max

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15 論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「6.1.3(1)(a) 共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1 バッチサーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「6.1.3(1)(b) CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

（4） Solaris の場合

CTM を使用する場合の，使用リソースの見積もりについて次の表に示します。

表 6-10　CTM 使用時の使用リソースの見積もり（Solaris の場合）

システムリソース	パラメタ	所要量	オプション設定ファイル例
共用メモリ	shminfo_shmmax	PrfTraceBufferSize※1 × 1,024 ＋ 18,496 ＋ CTM ドメインマネジャの共用メモリ※2 ＋ CTM デーモンの共用メモリ※2	/etc/system
プロセス数	max_nprocs	7 ＋バッチサーバ数※3	/etc/system
スレッド数	－	72 ＋（バッチサーバのスレッド数※4 ＋ 7）×バッチサーバ数※3 ＋ CTM デーモンで必要とするスレッド数※5	－
ファイルディスクリプタ数	rlim_fd_max	88 ＋（バッチサーバのファイルディスクリプタ数※4 ＋ 6）×バッチサーバ数※3 ＋ CTM デーモンで必要とするファイルディスクリプタ数※5	/etc/system

（凡例）－：該当しません。

注※ 1

パフォーマンストレーサのバッファメモリサイズを 512 キロバイト～ 102,400 キロバイトの範囲で指定します。PrfTraceBufferSize については，マニュアル「Cosminexus 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「4.15　論理パフォーマンストレーサで指定できるパラメタ」を参照してください。

注※ 2

値については，「6.1.3(1)(a)　共用メモリ用ファイルサイズの計算式」を参照して算出してください。

注※ 3

簡易構築定義ファイルの <j2ee-server-count> タグの指定値を指します。

注※ 4

バッチサーバのスレッド数とファイルディスクリプタ数については，「6.2.1　バッチサーバが使用するリソースの見積もり」を参照して算出してください。

注※ 5

CTM デーモンで必要とするスレッド数とファイルディスクリプタ数については，「6.1.3(1)(b)　CTM デーモンで必要とするスレッド数とファイルディスクリプタ数の計算式」を参照して算出してください。

6.2 プロセスごとに使用するリソース

この節では，アプリケーションサーバの各プロセスで使用するリソースの所要量の見積もりについて説明します。

6.2.1 バッチサーバが使用するリソースの見積もり

ここでは，バッチサーバのスレッド数とファイルディスクリプタ数の見積もり方法について説明します。アプリケーションサーバを動作させるために必要なディスクおよびメモリの容量を算出するときの参考にしてください。

（1） スレッド数

スレッド数の計算式を次に示します。(a) と (b) の合計が，バッチサーバが使用するスレッド数です。

（a） 基本のスレッド数

最大スレッド数 = （63 ＋ A ＋ 5 × B ＋ 2 × C ＋ D）/ 0.8

（凡例）

- A：CORBA ネーミングサービスのスレッド数（ただし，CORBA ネーミングサービスをインプロセスで起動（usrconf.properties の ejbserver.naming.startupMode キーに inprocess を指定）した場合だけ加算する）
  CORBA ネーミングサービスのスレッド数の見積もりについては，「5.2.1(3) CORBA ネーミングサービス（インプロセス起動時）のスレッド数の見積もり」を参照してください。
- B：データベースコネクションの数
- C：リソースアダプタ数
- D：簡易 Web サーバへの同時接続クライアント数（ただし，簡易 Web サーバへの同時接続クライアント数が 5 以下の場合は 5，100 以上の場合は 100 を指定する）

（b） JavaVM のオプション指定に応じて使用するスレッド数

JavaVM のオプション指定に応じて，次の計算式で算出してください。A は，-XX:+UseParNewGC オプションを指定している場合だけ加算します。B は，-XX:+HitachiUseExplicitMemory オプションを指定した場合だけ加算します。

最大スレッド数 = A + B

（凡例）

- A：パラレルコピーガーベージコレクションで使用するスレッド数（-XX:ParallelGCThreads オプションに指定した値。このオプションの指定を省略した場合は，論理 CPU 数を基にした -XX:ParallelGCThreads オプションのデフォルト値。なお，J2EE サーバ起動時の論理 CPU 数によって決定されるため，

起動後に論理 CPU の数を変更してもスレッド数は変化しない）
- B：明示管理ヒープ機能で使用するスレッド数（論理 CPU 数。ただし，論理プロセッサ数が 8 以上の場合は 8。なお，J2EE サーバ起動時の論理 CPU 数によって決定されるため，起動後に論理 CPU の数を変更してもスレッド数は変化しない）

JavaVM のオプションについては，マニュアル「Cosminexus 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)」の次の個所を参照してください。

- 19.5　Cosminexus で指定できる Java HotSpot VM のオプション
- -XX:[+|-]HitachiUseExplicitMemory（明示管理ヒープ機能オプション）

(2) ファイルディスクリプタ数

ファイルディスクリプタ数の計算式を次に示します。

最大ファイルディスクリプタ数 = （137 ＋ A ＋ B × 2 ＋ C × 2 ＋ D）/ 0.8

（凡例）
- A：データベースコネクションの数
- B：簡易 Web サーバへの同時接続クライアント数
- C：リソースアダプタ数
- D：usrconf.cfg の add.class.path キーに指定した JAR ファイルの数

6.2.2　運用管理エージェントが使用するリソースの見積もり

運用管理エージェントが使用するリソースの見積もりについては，J2EE アプリケーション実行基盤の場合と同じため，「5.2.2　運用管理エージェントが使用するリソースの見積もり」を参照してください。

6.2.3　パフォーマンストレーサが使用するリソースの見積もり

パフォーマンストレーサが使用するリソースの見積もりについては，J2EE アプリケーション実行基盤の場合と同じため，「5.2.3　パフォーマンストレーサが使用するリソースの見積もり」を参照してください。

6.2.4　CTM が使用するリソースの見積もり

CTM が使用するリソースの見積もりについては，J2EE アプリケーション実行基盤の場合と同じため，「5.2.4　CTM が使用するリソースの見積もり」を参照してください。

6.3 仮想メモリの使用量の見積もり

ここでは，仮想メモリの使用量の見積もり方法について説明します。なお，仮想メモリの使用量の計算式に使用している JavaVM 起動時のオプションの詳細については，「7.1.2 JavaVM で使用するメモリ空間の構成と JavaVM オプション」を参照してください。

また，明示管理ヒープ機能を使用する場合は，ここで説明する内容のほか，明示管理ヒープ機能で使用するメモリサイズの見積もりが必要です。明示管理ヒープ機能で使用するメモリサイズの見積もりについては，「7.10 Explicit ヒープのチューニング」を参照してください。

（1） 仮想メモリの使用量の計算式

仮想メモリの使用量（単位：メガバイト）の計算式を次に示します。

バッチサーバの仮想メモリの使用量 = A ＋ B ＋ C ＋（D ＋ 10）× E ＋ F

（凡例）

- A：Java ヒープサイズ
  初期値は，JavaVM 起動時のオプション -Xms に指定した値です。この値は，J2EE サーバ起動中に，最大で JavaVM 起動時のオプション -Xmx に指定した値まで拡張されます。
- B：Permanent 領域サイズ
  初期値は，JavaVM 起動時のオプション -XX:PermSize に指定した値です。この値は，J2EE サーバ起動中に，最大で JavaVM 起動時のオプション -XX:MaxPermSize に指定した値まで拡張されます。
  監査ログを使用する場合
  監査ログを使用する場合は，値に 1 メガバイトを加えてください。
- C：ネイティブプログラム使用領域サイズ
  OS ごとに使用する値が異なります。ネイティブプログラム使用領域サイズの OS ごとの値について，次の表に示します。

表 6-11 ネイティブプログラム使用領域サイズの値一覧

OS 種別	使用領域の値（単位：メガバイト）
Windows	300
AIX	400
HP-UX	400
Linux（x86）	400
Linux（AMD64 & Intel EM64T）	600

監査ログを使用する場合

監査ログを使用する場合は，OS ごとのネイティブプログラム使用領域サイズの値に，監査ログで使用する値を加える必要があります。監査ログで使用する値を算出するには，auditlog.raslog.message.filesize キー，および auditlog.raslog.exception.filesize キーを使用します。これらのキーは，auditlog.properties（監査ログ定義ファイル）で指定するキーです。これらのキーの詳細については，マニュアル「Cosminexus 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「13.2　監査ログ定義ファイル」を参照してください。

監査ログで使用する値（単位：メガバイト）の計算式を次に示します。

監査ログで使用する値 =

8 ＋（auditlog.raslog.message.filesize **の指定値** / $1{,}024^2$）※＋

（auditlog.raslog.exception.filesize **の指定値** / $1{,}024^2$）※

注※　指定値の単位はバイトです。仮想メモリの使用量の値の単位はメガバイトであるため，$1{,}024^2$ で割る必要があります。

- D：バッチサーバのスレッド数
  バッチサーバが使用するスレッド数です。バッチサーバが使用するスレッド数については，「6.2.1　バッチサーバが使用するリソースの見積もり」を参照してください。
- E：スタック領域サイズ
  JavaVM 起動時のオプション -Xss に指定した値です。
- F：Explicit ヒープサイズ
  明示管理ヒープ機能で使用される Explicit ヒープのサイズです。この値は，J2EE サーバ起動中に最大で -XX:HitachiExplicitHeapMaxSize に指定した値まで拡張されます。なお，デフォルトでは Explicit ヒープは無効になります。

（2）仮想メモリの使用量を計算する場合の注意事項

● ネイティブプログラム使用領域サイズの値は変動します。値が変動するのは次のような場合です。
- ORB のトレースサイズを変更した場合
- JDBC ドライバの使用領域サイズの値を変更した場合
- 使用する製品のネイティブライブラリ使用領域サイズの値を変更した場合

このような場合，製品ごとのメモリ使用量をネイティブプログラム使用領域サイズの値に加えてください。製品ごとのメモリ使用量は，使用する製品のドキュメントに従って算出してください。

● バッチサーバの標準構成で使用する仮想メモリの使用量は，使用するソフトウェア製品の影響で，見積もり値よりも大きくなる場合があります。その場合の増加分については，仮想メモリの使用量の値に加えてください。増加分のメモリ使用量は，使用するソフトウェア製品のドキュメントに従い，製品ごとのメモリ使用量を算出してください。

7 JavaVM のメモリチューニング

システムの処理性能を高めるには，基盤となる JavaVM 自体のチューニングを適切に実施する必要があります。日立の JavaVM では，2 種類のメモリ空間を管理しています。
この章では，ガーベージコレクションと日立の JavaVM でのメモリ管理，および Java ヒープと Explicit ヒープのチューニングについて説明します。

討

7.1 ガーベージコレクションと JavaVM のメモリ管理の概要

JavaVM のチューニングの目的は，システムの処理性能の向上です。特に，ガーベージコレクションの仕組みを踏まえ，適切なメモリ管理ができるようにチューニングすることで，システムの処理性能が向上します。

ここでは，JavaVM のチューニングをする前提として知っておく必要がある，次の項目について説明します。

- ガーベージコレクションの仕組み
- JavaVM で使用するメモリ空間の構成と JavaVM オプション
- ガーベージコレクションの発生とメモリ空間の関係

7.1.1 ガーベージコレクションの仕組み

ここでは，ガーベージコレクションの仕組みについて説明します。

(1) ガーベージコレクションとは

ガーベージコレクションは，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。

ガーベージコレクションの実行中は，プログラムの処理が停止します。このため，ガーベージコレクション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。

プログラムの中で new によって作成された Java オブジェクトは，JavaVM が管理するメモリ領域に格納されます。Java オブジェクトが作成されてから不要になるまでの期間を，Java **オブジェクトの寿命**といいます。

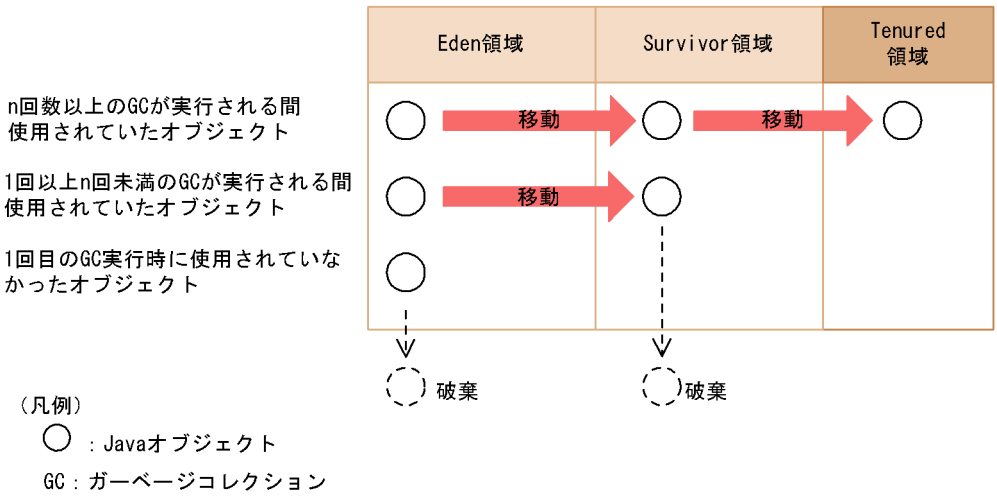
Java オブジェクトには，寿命の短いオブジェクトと寿命の長いオブジェクトがあります。サーバサイドで動作する Java アプリケーションの場合，リクエストやレスポンス，トランザクション管理などで，多くの Java オブジェクトが作成されます。これらの Java オブジェクトは，その処理が終わると不要になる，寿命が短いオブジェクトです。一方，アプリケーションの動作中使用され続ける Java オブジェクトは，寿命が長いオブジェクトです。

効果的なガーベージコレクションを実行するためには，寿命の短いオブジェクトに対してガーベージコレクションを実行して，効率良くメモリ領域を回収することが必要です。また，繰り返し使用される寿命の長いオブジェクトに対する不要なガーベージコレクションを抑止することが，システムの処理性能の低下防止につながります。これを実現するのが，**世代別ガーベージコレクション**です。

世代別ガーベージコレクションでは，Java オブジェクトを，寿命が短いオブジェクトが格納される New 領域と，寿命が長いオブジェクトが格納される Tenured 領域に分けて管理します。New 領域はさらに，new によって作成されたばかりのオブジェクトが格納される Eden 領域と，1 回以上のガーベージコレクションの対象になり，回収されなかったオブジェクトが格納される Survivor 領域に分けられます。New 領域内で一定回数以上のガーベージコレクションの対象になった Java オブジェクトは，長期間必要な Java オブジェクトと判断され，Tenured 領域に移動します。

世代別ガーベージコレクションで管理するメモリ空間と Java オブジェクトの概要を次の図に示します。

図 7-1　世代別ガーベージコレクションで管理するメモリ空間と Java オブジェクトの概要



世代別ガーベージコレクションで実行されるガーベージコレクションには，次の 2 種類があります。

● コピーガーベージコレクション

Eden 領域と Survivor 領域を対象にしたガーベージコレクションです。Java オブジェクトの作成によって，Eden 領域を使い切ると発生します。

Java HotSpot VM のオプションの指定によって，シリアルコピーガーベージコレクションとパラレルコピーガーベージコレクションを選択できます。

パラレルコピーガーベージコレクションは，シリアルコピーガーベージコレクションの処理を並列に実行することで，高速に実行できます。ただし，パラレルコピーガーベージコレクションを選択した場合，明示管理ヒープ機能は使用できません。

デフォルトでは，シリアルコピーガーベージコレクションが選択されています。なお，この章で説明している内容については，どちらのコピーガーベージコレクションにも当てはまります。

● フルガーベージコレクション

Tenured 領域も含む，JavaVM 固有領域全体を対象にしたガーベージコレクションです。Tenured 領域を一定サイズまで使うと発生します。

一般的に，コピーガーベージコレクションの方が，フルガーベージコレクションよりも短い時間で処理できます。

次に，ガーベージコレクションの処理について，ある Java オブジェクト（オブジェクト A）を例にして説明します。

Eden 領域で実行される処理

オブジェクト A の作成後，1 回目のコピーガーベージコレクションが実行された時点で使用されていない場合，オブジェクト A は破棄されます。
1 回目のコピーガーベージコレクションが実行された時点で使用されていた場合，オブジェクト A は Eden 領域から Survivor 領域に移動します。

Survivor 領域で実行される処理

Survivor 領域に移動したオブジェクト A は，その後何回かコピーガーベージコレクションが実行されると，Survivor 領域から Tenured 領域に移動します。移動する回数のしきい値は，JavaVM オプションや Java ヒープの利用状況によって異なります。図 7-1 では，しきい値を n 回としています。
Survivor 領域への移動後，n 回目未満のコピーガーベージコレクションが実行された時点でオブジェクト A が使用されていなかった場合，オブジェクト A はそのコピーガーベージコレクションで破棄されます。

Tenured 領域で実行される処理

オブジェクト A が Tenured 領域に移動した場合，その後のコピーガーベージコレクションでオブジェクト A が破棄されることはありません。コピーガーベージコレクションは，Eden 領域と Survivor 領域だけを対象としているためです。

このようにしてオブジェクトが移動することで，Tenured 領域の使用サイズは増加します。Tenured 領域を一定サイズまで使うと，フルガーベージコレクションが発生します。

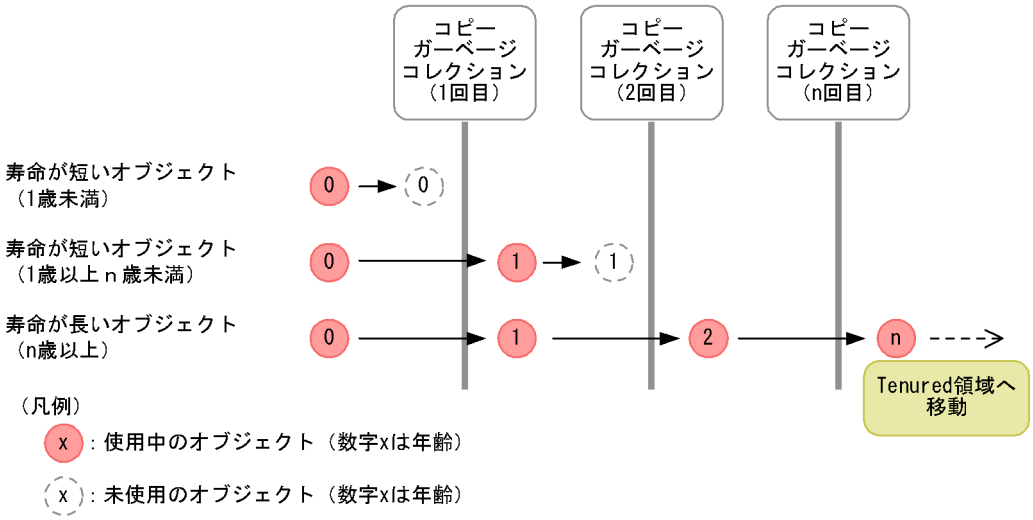
JavaVM のチューニングでは，JavaVM オプションでそれぞれのメモリ空間のサイズや割合を適切に設定することで，不要なオブジェクトが Tenured 領域に移動することを抑止します。これによって，フルガーベージコレクションが頻発することを防ぎます。

(2) オブジェクトの寿命と年齢の関係

オブジェクトがコピーガーベージコレクションの対象になった回数を**オブジェクトの年齢**といいます。

オブジェクトの寿命と年齢の関係を次の図に示します。

図 7-2　オブジェクトの寿命と年齢の関係



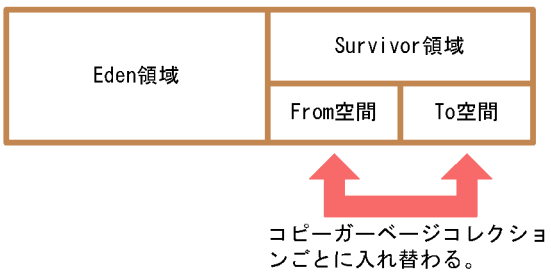
アプリケーションが開始して初期化処理が完了したあとで，何度かのコピーガーベージコレクションが実行されると，長期間必要になる寿命の長いオブジェクトは Tenured 領域に移動します。このため，アプリケーションの開始後しばらくすると，Java ヒープの状態は安定し，作成される Java オブジェクトとしては，寿命が短いオブジェクトが多くなります。特に，New 領域のチューニングが適切にできている場合，Java ヒープが安定したあとの大半のオブジェクトは，1 回目のコピーガーベージコレクションで回収される，寿命が短いオブジェクトになります。

(3) コピーガーベージコレクションの仕組み

JavaVM では，コピーガーベージコレクションの対象になる New 領域のメモリ空間を，Eden 領域，Survivor 領域に分けて管理します。さらに，Survivor 領域は，From 空間と To 空間に分けられます。From 空間と To 空間は，同じメモリサイズです。

New 領域の構成を次の図に示します。

図 7-3　New 領域の構成



Eden 領域は，new によって作成されたオブジェクトが最初に格納される領域です。プロ

グラムで new が実行されると，Eden 領域のメモリが確保されます。

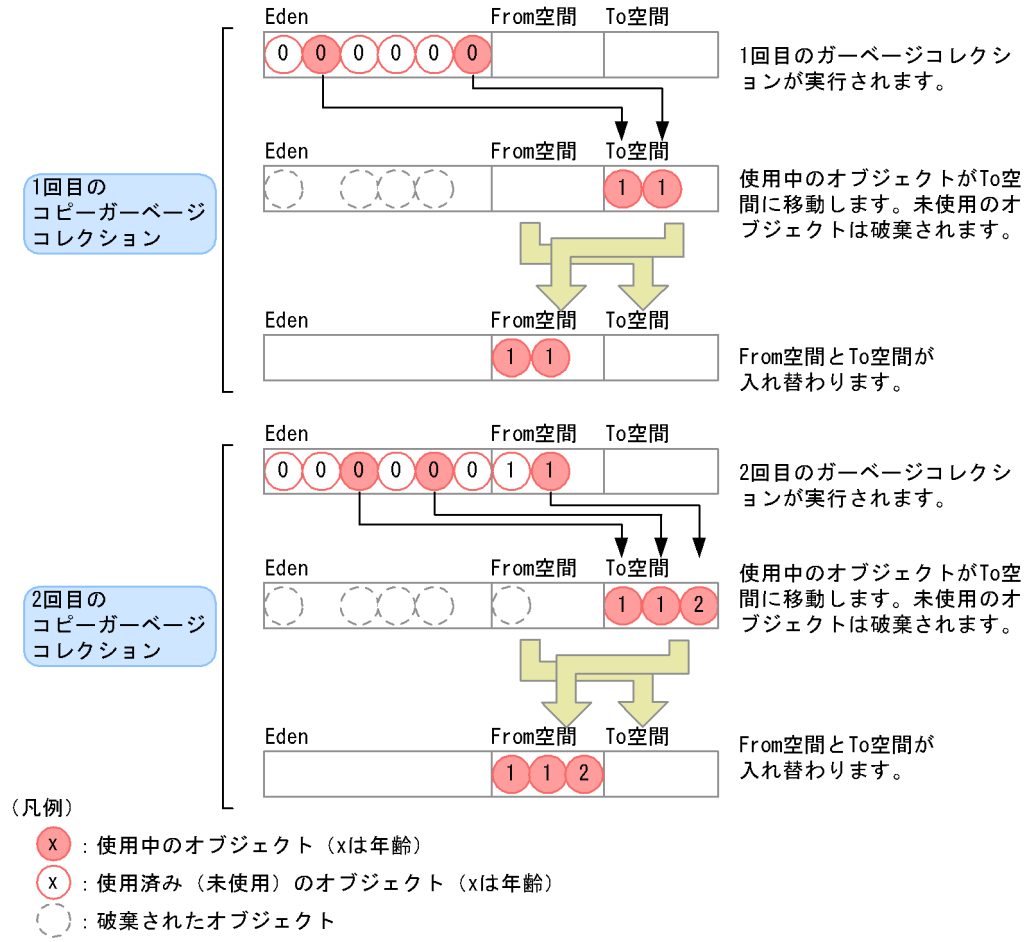
Eden 領域がいっぱいになると，コピーガーベージコレクションが実行されます。コピーガーベージコレクションでは，次の処理が実行されます。

1. Eden 領域および Survivor 領域の From 空間にある Java オブジェクトのうち，使用中の Java オブジェクトが，Survivor 領域の To 空間にコピーされます。使用されていない Java オブジェクトは破棄されます。
2. Survivor 領域の To 空間と From 空間が入れ替わります。

この結果，Eden 領域と To 空間は空になり，使用中のオブジェクトは From 空間に存在することになります。

コピーガーベージコレクション実行時に発生するオブジェクトの移動を次の図に示します。

図 7-4　コピーガーベージコレクション実行時に発生するオブジェクトの移動

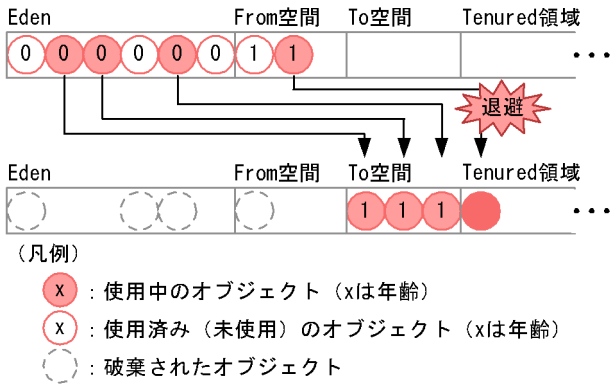


このようにして，使用中のオブジェクトは，コピーガーベージコレクションが発生するたびに，From 空間と To 空間を行ったり来たりします。ただし，寿命の長いオブジェクトを行き来させ続けると，コピー処理の負荷などが問題になります。これを防ぐために，From 空間と To 空間で Java オブジェクトを入れ替える回数にしきい値を設定して，年齢がしきい値に達した Java オブジェクトは Tenured 領域に移動させるようにします。

（4） オブジェクトの退避

年齢がしきい値に達していない Java オブジェクトが Tenured 領域に移動することを，退避といいます。退避は，コピーガーベージコレクション実行時に Eden 領域および From 空間で使用中のオブジェクトが多くなり，移動先である To 空間のメモリサイズが不足する場合に発生します。この場合，To 空間に移動できなかったオブジェクトが，Tenured 領域に移動します。

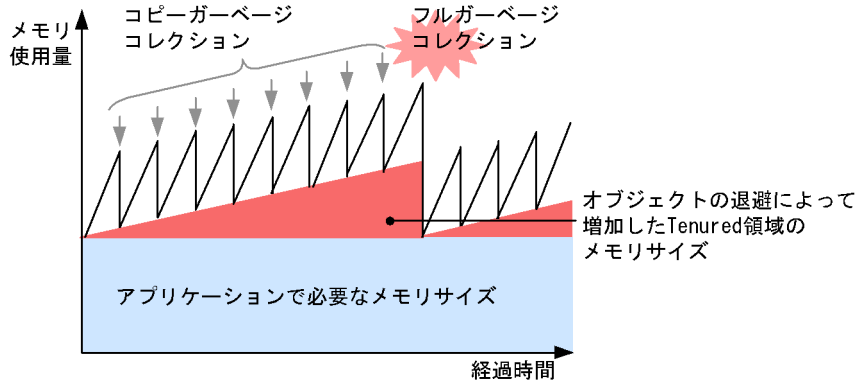
図 7-5　オブジェクトの退避



オブジェクトの退避が発生した場合，Tenured 領域に本来格納されないはずの寿命の短いオブジェクトが格納されます。これが繰り返されると，コピーガーベージコレクションで回収されるはずのオブジェクトがメモリ空間に残っていくため，Java ヒープのメモリ使用量が増加していき，最終的にはフルガーベージコレクションが発生します。

オブジェクトの退避が発生した場合のメモリ使用量の変化について，次の図に示します。

図 7-6　オブジェクトの退避が発生した場合のメモリ使用量の変化



フルガーベージコレクションでは，システムが数秒から数十秒停止することがあります。

したがって，メモリ空間の構成とメモリサイズを検討するときには，オブジェクトの退避が発生しないように，Eden 領域と Survivor 領域のバランスを検討する必要があります。

（5）ガーベージコレクション対象外の領域（明示管理ヒープ機能を使用した Explicit ヒープ領域の利用）

日立の JavaVM では，Eden 領域，Survivor 領域，および Tenured 領域以外の領域として Explicit ヒープという領域を利用することができます。Explicit ヒープ領域はガーベージコレクション対象外の領域です。

Explicit ヒープ領域に格納するオブジェクトは，自動配置設定ファイル，および明示管理ヒープ機能 API を使って指定します。指定されたオブジェクトは，Survivor 領域から Tenured 領域へ移動する際に Explicit ヒープ領域へ移動します。コピーガーベージコレクションで回収の対象にならない長寿命オブジェクトを指定することで，Tenured 領域のメモリ使用量を少なくし，フルガーベージコレクションの発生を抑止します。また，明示管理ヒープ機能の自動配置設定ファイルや，明示管理ヒープ機能 API を使って，指定したオブジェクトを Explicit ヒープ領域に生成することもできます。

明示管理ヒープ機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.　明示管理ヒープ機能を使用したフルガーベージコレクションの抑止」を参照してください。

注意事項

-XX:+UseParNewGC オプションを指定している場合，明示管理ヒープ機能は使用できません。

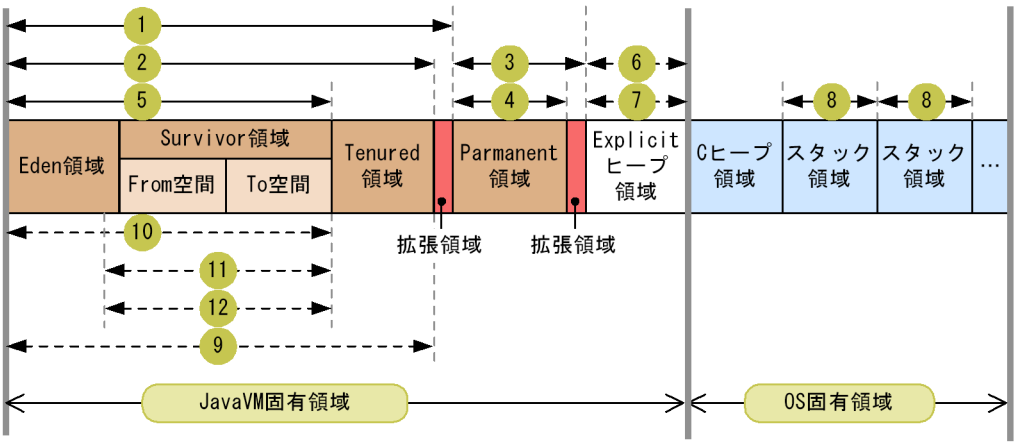
7.1.2　JavaVM で使用するメモリ空間の構成と JavaVM オプション

ここでは，JavaVM で使用するメモリ空間の構成と，JavaVM オプションについて説明します。

JavaVM では，JavaVM **固有領域**と OS **固有領域**という，2 種類のメモリ空間を使用します。

日立の JavaVM で使用するメモリ空間の構成を次の図に示します。なお，図中の番号は，表 7-1 の項番と対応しています。

図 7-7　JavaVM で使用するメモリ空間の構成



（凡例）◀——▶：サイズを指定するオプションの対象になる範囲
◀- -▶：割合または回数を指定するオプションの対象になる範囲

それぞれの領域について説明します。なお，Eden 領域，Survivor 領域，および Tenured 領域を合わせた領域を，Java **ヒープ**といいます。

Eden 領域

new によって作成された Java オブジェクトが最初に格納される領域です。

Survivor 領域

New 領域に格納されていた Java オブジェクトのうち，コピーガーベージコレクション実行時に破棄されなかった Java オブジェクトが格納される領域です。Survivor 領域は，From 空間と To 空間で構成されます。From 空間と To 空間のサイズは同じです。

Tenured 領域

長期間必要であると判断された Java オブジェクトが格納される領域です。Survivor 領域で指定回数を超えてコピーガーベージコレクションの実行対象となり，破棄さ

れなかった Java オブジェクトが，この領域に移動されます。

Permanent 領域

ロードされた class などの情報が格納される領域です。

Explicit ヒープ領域

フルガーベージコレクションの対象外になる Java オブジェクトが格納される領域です。Explicit ヒープ領域は日立の JavaVM 独自のメモリ空間で，明示管理ヒープ機能を利用する場合だけ確保されます。

C ヒープ領域

JavaVM 自身が使用する領域です。JNI で呼び出されたネイティブライブラリでも使用されます。

スタック領域

Java スレッドのスタック領域です。

それぞれの領域のサイズや割合などを指定する JavaVM オプションを次の表に示します。なお，表の項番は，図 7-7 と対応しています。

表 7-1 JavaVM メモリ空間のサイズや割合などを指定する JavaVM オプション

項番	オプション名	オプションの意味
1	-Xmx<size>	Java ヒープの最大サイズを設定します。
2	-Xms<size>	Java ヒープの初期サイズを設定します。
3	-XX:MaxPermSize=<size>	Permanent 領域の最大サイズを設定します。
4	-XX:PermSize=<size>	Permanent 領域の初期サイズを設定します。
5	-Xmn<size>	New 領域の初期値および最大値を設定します。
6	-XX:[+|-]HitachiAutoExplicitMemory	自動配置機能を使用する場合に，JavaVM を起動したタイミングで Explicit メモリブロックで使用するメモリを確保します。※
7	-XX:HitachiExplicitHeapMaxSize=<size>	Explicit ヒープ領域サイズの最大サイズを設定します。※
8	-Xss<size>	１スタック領域の最大サイズを設定します。
9	-XX:NewRatio=<value>	New 領域に対する Tenured 領域の割合を設定します。<value> が 2 の場合は，New 領域と Tenured 領域の割合が，1:2 になります。
10	-XX:SurvivorRatio=<value>	Survivor 領域の From 空間と To 空間に対する Eden 領域の割合を設定します。<value> に 8 を設定した場合は，Eden 領域，From 空間，To 空間の割合が，8:1:1 になります。

項番	オプション名	オプションの意味
11	-XX:TargetSurvivorRatio=<value>	ガーベージコレクション実行後の Survivor 領域内で Java オブジェクトが占める割合の目標値を設定します。
12	-XX:MaxTenuringThreshold=<value>	コピーガーベージコレクション実行時に，From 空間と To 空間で Java オブジェクトを入れ替える回数の最大値を設定します。 設定した回数を超えて入れ替え対象になった Java オブジェクトは，Tenured 領域に移動されます。

注　<size> の単位はバイトです。

注※　明示管理ヒープ機能を使用するための前提オプションが有効になっている必要があります。詳細は「7.10　Explicit ヒープのチューニング」を参照してください。

参考

JavaVM オプションは，次の個所に設定します。

表 7-2　JavaVM オプションを設定する個所

対象	設定方法	設定個所
J2EE サーバ	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** add.jvm.arg
バッチサーバ	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** add.jvm.arg
SFO サーバ	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 SFO サーバ（sfo-server） **パラメタ名** add.jvm.arg
EJB クライアントアプリケーション	ファイル編集	**定義ファイル** usrconf.cfg ※ **パラメタ名** add.jvm.arg キー
Web コンテナサーバ	ファイル編集	**定義ファイル** usrconf.cfg **パラメタ名** add.jvm.arg キー

注※　cjclstartap コマンドを使用して開始する場合に有効になるファイルです。

ポイント

それぞれのオプションのデフォルト値は，OS によって異なります。オプションのデフォルト値については，マニュアル「Cosminexus 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「19.4　Cosminexus で指定できる Java HotSpot VM のオプションのデフォルト値」を参照してください。

注意事項

Java ヒープ領域，Permanent 領域，C ヒープ領域のどれかが不足すると OutOfMemory が発生し，メモリ不足が解消されないかぎり正常に稼働できない状態が長く続くことになります。
これに対して，OutOfMemory 発生時のシステムへの影響を小さくするために，次のオプションを使用できます。

- -XX:+HitachiOutOfMemoryAbort（OutOfMemory **発生時強制終了機能**）
- -XX:+HitachiOutOfMemoryHandling（OutOfMemory **ハンドリング機能**）

OutOfMemory **発生時強制終了機能**は，Java ヒープ不足や Perm ヒープ不足などが原因で OutOfMemory が発生した場合に，J2EE サーバを強制終了するための機能です。Java ヒープ領域，Permanent 領域，C ヒープ領域のメモリ不足によって OutOfMemory が発生したときに，J2EE サーバを強制終了して，J2EE サーバを自動再起動します。これによって，J2EE サーバを正常に稼働できる状態に早期に回復できます。
OutOfMemory **ハンドリング機能**は，OutOfMemory 発生時強制終了機能を前提とする機能です。OutOfMemory 発生時強制終了機能を使用している場合でも，特定の条件に合致するときに限って J2EE サーバの実行を継続したい場合に使用します。
リクエスト処理でオブジェクトを大量に確保しようとした場合，巨大なオブジェクトを確保しようとした場合などに Java ヒープ不足が原因の OutOfMemory が発生したときに，J2EE サーバの実行を継続するようにしたいときには，OutOfMemory ハンドリング機能を使用してください。
オプションの詳細については，マニュアル「Cosminexus 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:[+|-]HitachiOutOfMemoryAbort（強制終了オプション）」および「-XX:[+|-]HitachiOutOfMemoryHandling（OutOfMemory ハンドリングオプション）」を参照してください。

7.1.3　ガーベージコレクションの発生とメモリ空間の関係

ガーベージコレクションは，メモリ空間の使用状況に応じて，次のタイミングで発生します。

(1) コピーガーベージコレクションが発生するタイミング

コピーガーベージコレクションは，次のタイミングで発生します。

1. Eden 領域へのアロケーションで空き領域が不足した場合
2. jheapprof コマンドに -copygc オプションを指定して実行した場合

(2) フルガーベージコレクションが発生するタイミング

フルガーベージコレクションは，次のタイミングで発生します。

1. New 領域（Eden 領域と Survivor 領域の合計）で使用しているメモリサイズが Tenured 領域の最大値に対する未使用メモリサイズを上回っている状態の時に，Eden 領域へのアロケーションで空き領域が不足した場合
2. New 領域と Tenured 領域のそれぞれの未使用メモリサイズを上回るメモリサイズ（Java オブジェクトのサイズ）のアロケーション要求があった場合
3. コピーガーベージコレクションの実施の結果，次のどちらかの状態になった場合
   - 確保済み Tenured 領域の未使用メモリサイズが 10,000 バイトを下回った場合
   - コピーガーベージコレクション実施時の Tenured 領域へのオブジェクトの移動によって，確保済み Tenured 領域の拡張が発生した場合
4. java.lang.System.gc() メソッドが実行された場合
5. Permanent 領域にアロケーションしたいメモリサイズが確保済み Permanent 領域の未使用メモリサイズを上回る場合
6. javagc コマンドを実行した場合
7. jheapprof コマンドを実行した場合

JavaVM のチューニングでは，主に 1. と 2. の発生を抑えることを検討します。

参考

フルガーベージコレクションが発生した場合の要因は，拡張 verbosegc 情報を使用して確認できます。フルガーベージコレクション発生時に要因を確認する方法については，「7.9 拡張 verbosegc 情報を使用したフルガーベージコレクションの要因の分析方法」を参照してください。

7.2 フルガーベージコレクション発生を抑止するためのチューニングの概要

この節では，フルガーベージコレクションの発生を抑止するための Java ヒープ，および Explicit ヒープのチューニングの考え方，およびチューニング手順について説明します。

Java ヒープ，および Explicit ヒープのチューニング方法の詳細は 7.3 ～ 7.12 で説明します。

参考

コピーガーベージコレクションの違いによって，チューニング方法が変わることはありません。

7.2.1 チューニングの考え方

一般的に，コピーガーベージコレクションの方が，フルガーベージコレクションよりも短い時間で処理できます。

New 領域へのコピーガーベージコレクションの実施によって適切にメモリを回収して，Java ヒープ全体を対象とするフルガーベージコレクションの発生をできるだけ抑止することが，システムの停止回数の削減や，処理性能向上につながります。これを実現するためには，JavaVM オプションでそれぞれのメモリ空間のサイズや割合を適切に設定することが必要です。

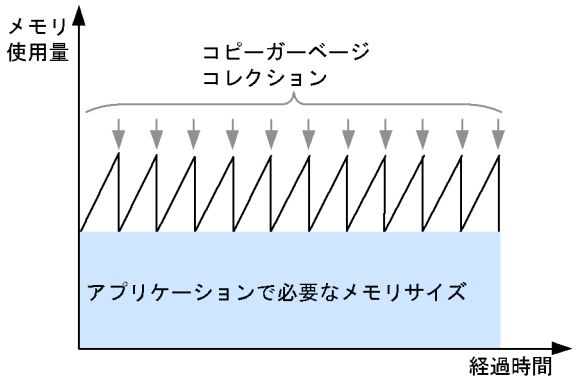
また，Tenured 領域に配置されるオブジェクトの一部を，明示管理ヒープ機能で管理する Explicit ヒープに配置することも，フルガーベージコレクションの発生抑止に効果的です。Explicit ヒープは，自動配置設定ファイルや明示管理ヒープ API を使って，アプリケーションから利用できます。また，Explicit ヒープは J2EE サーバからも利用されます。

これらを踏まえ，JavaVM のチューニングは，次の 2 点を目的として実施します。

- フルガーベージコレクションをできるだけ発生させないこと
- フルガーベージコレクションの頻発を抑止した上で，不要なコピーガーベージコレクションを発生させないこと

理想的なメモリ使用量と経過時間の関係を次の図に示します。

図 7-8 理想的なメモリ使用量と経過時間の関係



この図の場合は，寿命の短いオブジェクトはすべてコピーガーベージコレクションによって回収できていて，オブジェクトの昇格や退避が発生しません。このため，コピーガーベージコレクション実行後のメモリサイズが一定です。これによって，フルガーベージコレクションが発生しない，安定した状態での運用を実現できます。

JavaVM のチューニングでは，この状態を理想として，JavaVM の使用するメモリ空間の各領域で使用するメモリサイズを見積もってチューニングします。

ポイント

チューニングの目安
図 7-8 では，フルガーベージコレクションを一度も発生させない理想的な例を示しましたが，現実的にはフルガーベージコレクションが 1 回発生する間にコピーガーベージコレクションが 10 ～ 20 回程度発生することを目安にして，チューニングを実施してください。

7.2.2 チューニング手順

フルガーベージコレクションの発生を抑止するための Java ヒープ，および Explicit ヒープのチューニング手順について説明します。

なお，手順 (1) は必ず実施します。手順 (2) ～ (4) は明示管理ヒープ機能を利用して Explicit ヒープを使う場合に，手順 (1) のあとに続けて実施します。手順 (5) 以降については，それぞれの手順の説明を確認の上，必要に応じて実施してください。

Java ヒープ，および Explicit ヒープのチューニング手順を次の図に示します。

図 7-9　Java ヒープおよび Explicit ヒープのチューニング手順

(1) Java ヒープのチューニング

Java ヒープのチューニングによってフルガーベージコレクションを抑止する方法を検討します。Java ヒープのチューニングについては，「7.3　Java ヒープのチューニング」を参照してください。

なお，Explicit ヒープ領域を利用しない場合は，Java ヒープのチューニングをした段階で J2EE サーバのテストを実施してください。Java ヒープのメモリを適切に見積もってもフルガーベージコレクションが頻発する場合は，Survivor 領域があふれているなど，チューニングに問題がないか確認してください。再度 Java ヒープのチューニングを見直しても問題が発生する場合は，明示管理ヒープ機能を使用した Explicit ヒープの利用を検討してください。Explicit ヒープを利用する場合は，手順 (2) に進んでください。

(2) Explicit ヒープのチューニング

明示管理ヒープ機能を使用して Explicit ヒープ領域を利用する場合に，Explicit ヒープ領域のメモリを見積もります。Explicit ヒープのチューニングについては，「7.10 Explicit ヒープのチューニング」を参照してください。

J2EE サーバの場合，明示管理ヒープ機能はデフォルトで使用する設定になっています。また，Tenured 領域のメモリサイズ増加の要因となるオブジェクトが Explicit ヒープに配置されるように設定されています。このため，J2EE サーバが配置するオブジェクトに

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

(3) 稼働情報による見積もり結果の確認

明示管理ヒープ機能を使用する場合に，手順 (1) および手順 (2) で JavaVM のメモリを適切に見積もったあと，J2EE サーバのテストを実施します。テストで得た稼働情報を収集して Explicit ヒープの使用状況を確認します。稼働情報を基にした Explicit ヒープの見積もりについては「7.10.5　稼働情報による見積もり」を参照してください。

参考

J2EE サーバのテストを実施し，フルガーベージコレクションの発生回数が削減できない場合は，次の内容を確認してください。

- Survivor 領域があふれているなど，Java ヒープのチューニングに問題がないか
  手順 (1) で見積もった値が適切かを見直します。
- Explicit ヒープがあふれていないか
  手順 (2) で見積もった値が適切かを見直します。
- Web アプリケーションの構成が適切か
  Web アプリケーションの構成（アプリケーション内の API の使用方法など）によっては，HTTP セッションに関するオブジェクトに対して，明示管理ヒープ機能の効果が出ない場合があります。詳細は，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.12　明示管理ヒープ機能使用時の注意事項」を参照してください。

これらを確認しても問題が解決しない場合に，手順 (4) に進んでください。

(4) Tenured 領域のメモリサイズが増加するかどうかの確認

アプリケーションを開始して，Tenured 領域のメモリサイズを調査します。調査には，手順 (3) で取得した稼働情報，または拡張 verbosegc 情報で取得した情報を使用します。拡張 verbosegc 情報の取得については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「5.7.2　拡張 verbosegc 情報の取得」を参照してください。

(5) Explicit ヒープへのオブジェクトの生成

手順 (4) で Tenured 領域のメモリサイズの増加が確認された場合，必要に応じて任意のオブジェクトを Explicit ヒープに生成するように自動配置設定ファイルを設定します。自動配置設定ファイルを使用するかどうかの検討については，「7.12　明示管理ヒープの自動配置機能を使用した Explicit ヒープの利用の検討」を参照してください。

Explicit ヒープへオブジェクトを生成する場合，Java ヒープへオブジェクトを生成する場合に比べて実行時のオーバーヘッドが大きくなります。そのため，Explicit ヒープへオブジェクトを生成する場合は，オブジェクトの生成個所を絞り込むことで，実行時の

オーバーヘッドを小さくできます。自動配置設定ファイルについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.11.2　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。

注意事項

明示管理ヒープ機能 API を使用して実装する場合は，Explicit ヒープに格納するオブジェクトのライフサイクルが既知である必要があります。具体的には，オブジェクトの生成，およびオブジェクトが不要になるタイミングが，Java プログラム上で明確になっている必要があります。明示管理ヒープ機能 API を使用した実装については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.10　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。

(6) Explicit ヒープ全体のメモリサイズの見直し

手順 (5) で修正したアプリケーションを動作させて，J2EE サーバとアプリケーションが使用する Explicit ヒープ全体のメモリサイズを見直します。見直し方法については，「7.11　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メモリサイズの見積もり」を参照してください。

ポイント

明示管理ヒープ機能によってフルガーベージコレクションの発生抑止の効果を得るためには，Explicit ヒープからオブジェクトがあふれないようにする必要があります。次の点を確認してください。

- Web アプリケーション内でセッションの破棄（invalidate メソッド呼び出し）および適切なセッションタイムアウトを設定していること。
- 適切なメモリサイズの Explicit ヒープ領域を Java ヒープ領域とは別に確保できること。

Explicit ヒープあふれが発生した場合の確認と対処については，「7.11.4(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。
以降では，Java ヒープのチューニング，および Explicit ヒープのチューニングについて説明します。なお，これ以外の注意事項については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.　明示管理ヒープ機能を使用したフルガーベージコレクションの抑止」を参照してください。

7.3　Java ヒープのチューニング

ここでは，Java ヒープのチューニングについて説明します。

7.3.1　Java ヒープのメモリサイズの見積もり

JavaVM のチューニングでは，JavaVM 固有領域の各領域のメモリサイズを適切に見積もる必要があります。

見積もりのポイントになるメモリサイズは次のとおりです。

● Java ヒープ全体のメモリサイズ

● Tenured 領域のメモリサイズ

● Survivor 領域のメモリサイズ

● Eden 領域のメモリサイズ

このほか，Permanent 領域も，必要に応じて見積もります。

コピーガーベージコレクション後の生存オブジェクトのサイズが Survivor 領域のサイズよりも大きい場合，Survivor 領域があふれ，1 度のコピーガーベージコレクションの実行で Tenured 領域に昇格するオブジェクトが発生します。また，Survivor 領域のサイズが小さい場合に，Survivor 領域の使用率が上がってくると，本来短寿命（コピーガーベージコレクション間隔以下，またはコピーガーベージコレクション間隔の 1 ～ 2 倍程度の寿命）の Java オブジェクトが，数回のコピーガーベージコレクションの実行で Tenured 領域に昇格してしまいます。

ポイント

次に示す問題が確認できた場合は，Survivor 領域の不足が原因でフルガーベージコレクションが発生しています。なお，拡張 verbosegc 情報とは，J2EE サーバ起動時にオプションを設定しておくと，ガーベージコレクション発生時に日立 JavaVM ログファイルに出力される情報です。

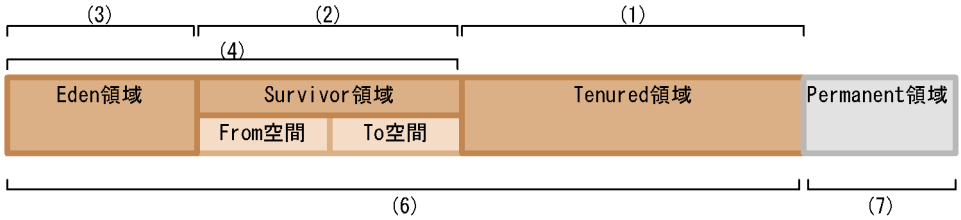
- Tenured 領域の増加要因が短寿命オブジェクトであると特定された場合
- 拡張 verbosegc 情報で，コピーガーベージコレクション時の Survivor 領域あふれが確認できた場合
- -XX:+HitachiVerboseGCPrintTenuringDistribution 指定時に出力される拡張 verbosegc 情報で，オブジェクトの昇格年齢が常に 1 であることが観測できた場合

これらの問題を回避するためには，-XX:SurvivorRatio オプションの設定値を小さくして，Eden 領域と Survivor 領域の割合を最適化する必要があります。

このことを考慮し，見積もりでは，まず，Tenured 領域のメモリサイズと New 領域のメモリサイズを算出して，それらを基に Java ヒープ全体のメモリサイズを算出します。

メモリサイズを見積もる順序を次の図に示します。図中の番号の順番で見積もりを実施します。

図 7-10　メモリサイズを見積もる順序



見積もり手順を示します。なお，番号は図中の番号と対応しています。

1. Tenured **領域で使用するメモリサイズを見積もります。**
   見積もり方法については，「7.4　Java ヒープ内の Tenured 領域のメモリサイズの見積もり」を参照してください。
2. Survivor **領域で使用するメモリサイズを見積もります。**
   見積もり方法については，「7.5.1　Java ヒープ内の Survivor 領域のメモリサイズの見積もり」を参照してください。
3. Eden **領域で使用するメモリサイズを見積もります。**
   見積もり方法については，「7.5.2　Java ヒープ内の Eden 領域のメモリサイズの見積もり」を参照してください。
4. 2. **と** 3. **の合計として，**New **領域全体のメモリサイズを算出します。**
5. **一定期間存在するオブジェクトの扱いを検討して，必要なメモリサイズを** Tenured **領域または** New **領域のメモリサイズに追加します。**
   検討方法については，「7.6　Java ヒープ内に一定期間存在するオブジェクトの扱いの検討」を参照してください。
6. 1.**，**4. **および** 5. **の合計として，**Java **ヒープ全体のメモリサイズを算出します。**
7. **必要に応じて** Permanent **領域のメモリサイズを見積もります。**
   見積もり方法については，「7.8　Java ヒープ内の Permanent 領域のメモリサイズの見積もり」を参照してください。

7.3.2　Java ヒープのメモリサイズの設定方法

見積もったメモリサイズは，「7.1.2　JavaVM で使用するメモリ空間の構成と JavaVM オプション」で説明したオプションで指定します。それぞれの領域のメモリサイズの設定方法を次に示します。

Java ヒープ全体のメモリサイズ

-Xmx<size> オプションで最大サイズを指定して，-Xms<size> オプションで初期サ

イズを指定します。

Tenured 領域のメモリサイズ

-XX:NewRatio=<value> オプションで，Java ヒープ全体に対する，Tenured 領域と New 領域の分割比を指定します。例えば，「-XX:NewRatio=5」とした場合には，-Xmx<size> オプションで指定したメモリサイズが，次のように分割されます。

```
New領域のメモリサイズ：Tenured領域のメモリサイズ = 1：5
```

Survivor 領域のメモリサイズと Eden 領域のメモリサイズ

-XX:SurvivorRatio=<value> オプションで，New 領域全体に対する，Survivor 領域と Eden 領域の分割比を指定します。なお，分割比は，Survivor 領域の From 空間および To 空間に対して Eden 領域を何倍確保するかの数値で指定します。例えば，「-XX:SurvivorRatio=2」とした場合には，-XX:NewRatio=<value> オプションで決まった New 領域のメモリサイズが，次のように分割されます。

```
Eden領域のメモリサイズ：From空間のメモリサイズ：To空間のメモリサイズ =
2：1：1
```

Permanent 領域のメモリサイズ

-XX:MaxPermSize=<size> オプションで最大サイズを指定して，
-XX:PermSize=<size> オプションで初期サイズを指定します。

7.3.3 Java ヒープのメモリサイズの使用状況の確認方法

それぞれのメモリサイズは，アプリケーションを実際に動作させて，メモリ使用量を測定しながらチューニングしていきます。アプリケーションサーバでは，usrconf.cfg ファイルに -XX:+HitachiVerboseGC **オプション**を指定して J2EE サーバを起動することで，ガーベージコレクション実行時の各領域の詳細なメモリサイズを**拡張** verbosegc **情報**として出力できます。この出力内容を基にチューニングを実施してください。

拡張 verbosegc 情報として出力できる主な内容を次に示します。

- ガーベージコレクションの発生日時
- ガーベージコレクション種別
- ガーベージコレクション情報※1
- ガーベージコレクション経過時間
- Eden 情報※1
- Survivor 情報※1
- Tenured 情報※1
- Perm 情報※1
- ガーベージコレクション要因内容※2

注※ 1

ガーベージコレクション前後の使用領域長および領域サイズが出力されます。

注※ 2

-XX:+HitachiVerboseGCPrintCause オプションが指定されている場合に出力されます。

拡張 verbosegc 情報の出力例とフルガーベージコレクションの要因分析方法については，「7.9　拡張 verbosegc 情報を使用したフルガーベージコレクションの要因の分析方法」を参照してください。また，オプションについては，マニュアル「Cosminexus 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:[+|-]HitachiVerboseGC（拡張 verbosegc 情報出力オプション）」を参照してください。

なお，アプリケーションサーバでは，javagc コマンドを使用して，任意のタイミングでガーベージコレクションを発生させることもできます。この場合，-v オプションを指定することで，拡張 verbosegc 情報と同じ内容を出力できます。javagc コマンドの詳細については，マニュアル「Cosminexus アプリケーションサーバ リファレンス コマンド編」の「javagc（ガーベージコレクションの強制発生）」を参照してください。

7.4　Java ヒープ内の Tenured 領域のメモリサイズの見積もり

この節では，Tenured 領域のメモリサイズの見積もりについて説明します。

Tenured 領域のメモリサイズは，次のように見積もります。

```
Tenured 領域のメモリサイズ
= アプリケーションで必要なメモリサイズ +New 領域のメモリサイズ
```

ここでは，アプリケーションで必要なメモリサイズの算出方法について説明します。

また，見積もったメモリサイズに New 領域のメモリサイズを追加する理由についても説明します。

7.4.1　アプリケーションで必要なメモリサイズの算出

Tenured 領域のメモリサイズは，アプリケーションが最低限必要とするメモリサイズから見積もります。必要なメモリサイズが確保できない場合，OutOfMemoryError が発生して JavaVM が停止します。

アプリケーションが必要とするメモリサイズは，フルガーベージコレクション実行時の拡張 verbosegc 情報で，フルガーベージコレクション実行後に使用しているメモリサイズを確認することで判断できます。これは，フルガーベージコレクション実行後に Java ヒープ全体から不要なオブジェクトをすべて削除した状態のメモリサイズが，アプリケーションが必要とするメモリサイズに近いと考えられるためです。

フルガーベージコレクション実行時の拡張 verbosegc 情報の出力例を次に示します。

```
…
[VGC]<Wed May 11 23:12:05 2005>[Full GC 31780K->30780K(32704K),
0.2070500secs][DefNew::Eden:
3440K->1602K(3456K)][DefNew::Survivor:58K->0K(64K)][Tenured:
28282K->29178K(29184K)][Perm:1269K->1269K(4096K)][cause:ObjAllocFail][User:
0.0156250 secs][Sys: 0.0312500 secs]
…
```

「Full GC」に続いて出力されている情報のうち，ガーベージコレクションの実行後の情報「->30780K」を確認します。ここでは，フルガーベージコレクション実行後に，30,780 キロバイトのメモリサイズを必要としていることがわかります。

何回分かのフルガーベージコレクションの拡張 verbosegc 情報を集め，ガーベージコレクション実行後の領域長がいちばん大きい情報を，アプリケーションが必要とするメモ

リサイズであると判断してください。

7.4.2 Java ヒープ内の New 領域のメモリサイズを追加する理由

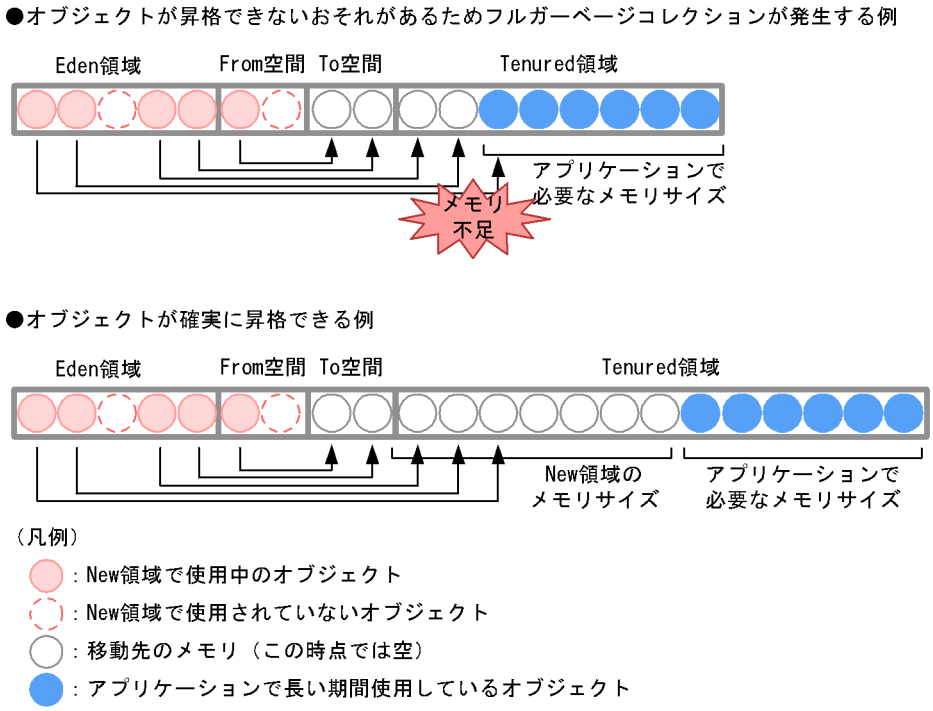
Tenured 領域のメモリサイズには，アプリケーションが最低限必要とするメモリサイズに，New 領域分のメモリサイズを追加することをお勧めします。これは，Tenured 領域の未使用メモリサイズが New 領域の使用メモリサイズを下回ることによって，フルガーベージコレクションが頻発するのを防ぐためです。

通常，Eden 領域がいっぱいになると，コピーガーベージコレクションが発生します。このとき，Eden 領域と Survivor 領域の From 空間に存在する使用中の Java オブジェクトが，Survivor 領域の To 空間に移動しようとします。このとき，Tenured 領域の未使用領域が Eden 領域と Survivor 領域で使用中のメモリサイズよりも小さいと，New 領域のすべての Java オブジェクトが昇格した場合に，Java オブジェクトを Tenured 領域に移動できなくなります。そこで JavaVM は，フルガーベージコレクションを発生させ，Tenured 領域の未使用メモリサイズを増やそうとします。

これを防ぐために，Tenured 領域には，アプリケーションが必要とするメモリサイズに加えて，New 領域分のメモリサイズを追加してください。

考え方を次の図に示します。

図 7-11　New 領域のメモリサイズを追加する理由の考え方



● オブジェクトが昇格できないおそれがあるためフルガーベージコレクションが発生する例

Tenured 領域のメモリの空き領域（アプリケーションで必要なメモリ領域以外の領域）が New 領域のメモリサイズよりも小さいため，Eden 領域および From 空間からの移動オブジェクトが多い場合，オブジェクトの昇格に対応できないおそれがあります。この場合，フルガーベージコレクションが発生します。

● オブジェクトが確実に昇格できる例

Tenured 領域のメモリの空き領域（アプリケーションで必要なメモリ領域以外の領域）が New 領域と同じサイズ分確保してあるため，Eden 領域および From 空間からの移動オブジェクトが多い場合も，オブジェクトの昇格に対応できます。

なお，New 領域のメモリサイズの見積もりについては，「7.5　Java ヒープ内の New 領域のメモリサイズの見積もり」で説明します。

ポイント

拡張 verbosegc 情報などで確認したときに，コピーガーベージコレクションが発生しないでフルガーベージコレクションが頻発している場合，New 領域からの退避オブジェクトに対して Tenured 領域のメモリサイズが小さいことが考えられます。New 領域のサイズを増やした場合などにこの状態になることがあります。必要に応じて，Tenured 領域のメモリサイズを見直してください。また，New 領域内の Eden 領域と Survivor 領域の関係もあわせて見直してください。

7.5 Java ヒープ内の New 領域のメモリサイズの見積もり

この節では，New 領域のメモリサイズの見積もりについて説明します。

New 領域のメモリサイズは，次のように見積もります。

New 領域のメモリサイズ
=Survivor 領域のメモリサイズ +Eden 領域のメモリサイズ

ここでは，Survivor 領域および Eden 領域のメモリサイズを見積もる方法について説明します。

7.5.1 Java ヒープ内の Survivor 領域のメモリサイズの見積もり

Survivor 領域のメモリサイズは，実際にアプリケーションを動作させて，Survivor 領域の使用状況を確認しながらチューニングしていきます。

チューニングの流れを次に示します。

1. **アプリケーションでのリクエスト／レスポンス処理に使用するメモリサイズを見積もり，それを** Survivor **領域のメモリサイズに指定して，アプリケーションを実行します。**
   このとき，チューニングで使用する情報を出力するために，-XX:+HitachiVerboseGCPrintTenuringDistribution **オプション**を指定して J2EE サーバを起動します。
2. Survivor **領域に割り当てられているメモリサイズと，アプリケーション実行時に実際に使用されているメモリ使用量から，メモリ使用率を確認します。**
   メモリ使用率が 100% に近い場合，コピーガーベージコレクション実行時に New 領域および Survivor 領域の From 空間の使用中オブジェクトが To 空間に入り切らなくなり，オブジェクトの退避が発生します。この場合は，Survivor 領域を増やすことを検討してください。
3. Survivor **領域のオブジェクトの年齢分布を確認します。**
   Survivor 領域のメモリサイズを増やしたり，昇格するためのしきい値を上げたりすることで，オブジェクトが昇格するのが遅くなります。寿命の長いオブジェクトを Survivor 領域に格納し続けるのは，性能を低下させる要因になります。
   逆に，Survivor 領域のメモリサイズを減らしたり，昇格するためのしきい値を下げたりすることで，オブジェクトが昇格するのが早くなります。ただし，寿命の短いオブジェクトが昇格するのは，フルガーベージコレクションの発生頻度を増やす要因にな

ります。
Survivor 領域のメモリサイズと昇格のしきい値は，この二つのバランスを取るように検討してください。

それぞれのチューニング作業について説明します。

(1) リクエスト／レスポンス処理に使用するメモリサイズの見積もり

Survivor 領域は，寿命の短いオブジェクトを格納する領域です。サーバサイドで動作するアプリケーションの場合，リクエストやレスポンスを処理するために使われている，寿命の短いオブジェクトを格納する領域と考えることができます。このため，Survivor 領域のメモリサイズの見積もりでは，ある時点で存在する寿命が短いオブジェクトの最大サイズ，つまり，ある時点でのリクエストやレスポンスの処理に使用するメモリの最大サイズを考えます。例えば，ステートレスなサーブレットで構成されたアプリケーションの場合，Survivor 領域のメモリサイズを，「一つのリクエスト処理で使用する最大メモリサイズ×リクエストの同時実行数」と考えることができます。

(2) メモリ使用率の確認

「(1) リクエスト／レスポンス処理に使用するメモリサイズの見積もり」で見積もった値を Survivor 領域のメモリサイズとして設定して，アプリケーションを実行します。実行時に使用されるメモリ使用量から，Survivor 領域に割り当てたメモリサイズに対するメモリ使用率を確認します。

参考

Survivor 領域のメモリサイズは，直接は指定できません。
Survivor 領域のメモリサイズを指定する場合は，まず，-Xmx オプションで Java ヒープの最大サイズを指定して，-XX:NewRatio=<value> によって Java ヒープのメモリサイズを New 領域と Tenured 領域で分ける割合を指定した上で，-XX:SurvivorRatio=<value> オプションによって，Eden 領域との割合を指定する必要があります。

メモリ使用率は，拡張 verbosegc 情報で確認できます。

コピーガーベージコレクション実行時の拡張 verbosegc 情報の出力例を次に示します。

```
…
[VGC]<Wed May 11 23:12:05 2005>[GC 27340K->27340K(32704K), 0.0432900
secs][DefNew::Eden: 3440K->0K(3456K)][DefNew::Survivor:
58K->64K(64K)][Tenured: 23841K->27282K(29184K)][Perm:
1269K->1269K(4096K)][cause:ObjAllocFail][User: 0.0156250 secs][Sys: 0.0312500
secs]
…
```

「DefNew::Survivor: 58K->64K(64K)」は，「ガーベージコレクション実行前のメモリサイズ -> ガーベージコレクション実行後のメモリサイズ（割り当てられているメモリサイ

ズ)」を意味します。この場合，64 キロバイトの Survivor 領域中 64 キロバイトがすでに使用されていて，使用率は 100% になります。これは，コピーガーベージコレクションで，To 空間のメモリサイズが不足し，Java オブジェクトの退避が行われたことを示しています。退避では，Tenured 領域に本来格納されないはずの寿命の短いオブジェクトが格納され，フルガーベージコレクションの発生頻度を増やす要因になります。このような場合は，Survivor 領域のメモリサイズを増やすことを検討してください。新しい Survivor 領域のメモリサイズは次のように見積もります。

新しい Survivor 領域のメモリサイズ
= 現在の Survivor 領域のメモリサイズ + 退避された Java オブジェクトの合計サイズ

「退避された Java オブジェクトの合計サイズ」は，ガーベージコレクション実行後の Tenured 領域メモリの増加サイズで近似できます。この例では，「Tenured: 23841K->27282K(29184K)」が，Tenured 領域メモリの増加サイズを示していて，27,282 キロバイト -23,841 キロバイト =3,441 キロバイトとなります。

また，Survivor 領域のメモリサイズを増加させると，Java オブジェクトの昇格のしきい値が上がり，昇格しにくくなります。詳細については，「(3)　オブジェクトの年齢分布の確認と見積もり」を参照してください。その結果，コピーガーベージコレクションで回収されない Java オブジェクトが増えることがあるため，
-XX:TargetSurvivorRatio=<value> を次のように見積もり，設定してください。

新しい -XX:TargetSurvivorRatio=<value>
= 現在の -XX:TargetSurvivorRatio=<value> ×（現在の Survivor 領域のメモリサイズ / 新しい Survivor 領域のメモリサイズ）

(3) オブジェクトの年齢分布の確認と見積もり

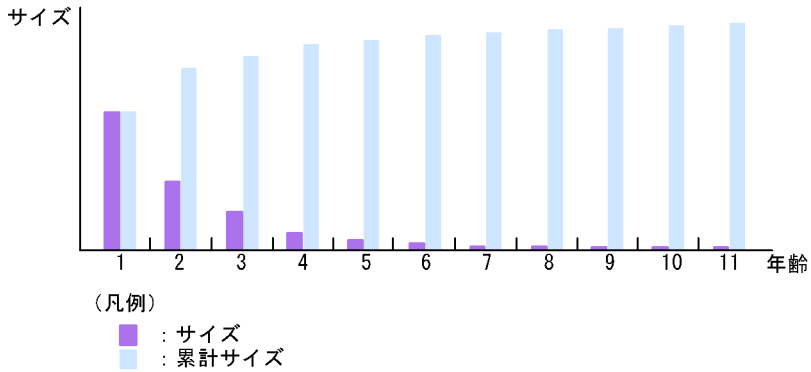
Survivor 領域のオブジェクトの年齢分布を確認し，寿命の長いオブジェクトが存在し続けていないか，または寿命の短いオブジェクトが昇格していないかを確認します。オブジェクトの年齢分布は，-XX:+HitachiVerboseGCPrintTenuringDistribution オプションの出力結果で確認できます。

J2EE サーバ起動時に usrconf.cfg に
-XX:+HitachiVerboseGCPrintTenuringDistribution オプションを指定すると，Survivor 領域の使用状況がコピーガーベージコレクション発生のタイミングで日立 JavaVM ログファイルに出力されます。出力例を次に示します。

```
[PTD]<Wed May 28 11:45:23 2008>[Desired survivor:5467547 bytes][New
threshold:3][MaxTenuringThreshold:31][age1:1357527][age2:1539661]
```

「New threshold:」に続けて出力されているのが，次のコピーガーベージコレクションで昇格する Java オブジェクトの最低の年齢です。例の場合は，次回のコピーガーベージコレクションで，年齢が 3 歳以上の Java オブジェクトが昇格します。「age< 数値 >:」に続けて出力されているのが，Survivor 領域で 1 歳からその年齢までの Java オブジェクトが使用しているメモリサイズの累計です。例の場合は，1 歳の Java オブジェクトのメモリサイズが 1,357,527 バイト，1 歳と 2 歳の Java オブジェクトのメモリサイズの累計が 1,539,661 バイトであることを示しています。また，累計から逆算することで，2 歳の Java オブジェクトのメモリサイズが 182,134(1,539,661-1,357,527) バイトであることがわかります。一般的に，Survivor 領域のオブジェクトの年齢分布は次に示すグラフのようになります。

図 7-12　Survivor 領域のオブジェクトの年齢分布



グラフの「サイズ」は，ある「年齢」の Java オブジェクトの合計のサイズです。また，「累計サイズ」は，ある「年齢」までの Java オブジェクトの合計のサイズです。

このグラフでは，Survivor 領域の Java オブジェクトが占めるサイズが，年齢が上がるに連れて減少しています。また，1 歳年齢が上がったときに減少するサイズは年齢が若いほど大きくなります。このことから，次のことがわかります。

- 若い年齢の Java オブジェクトほど，Survivor 領域中に占めるサイズが大きい
- 若い年齢の Java オブジェクトほど，ガーベージコレクションで回収されやすい

この例では，6 歳以上の Java オブジェクトはほとんどコピーガーベージコレクションで回収されていません。そのため，Java オブジェクトの昇格のしきい値が 7 歳以上の場合，回収される可能性が低い Java オブジェクトに対してコピーガーベージコレクションを行うことになり，性能を低下させる要因になります。逆に，Java オブジェクトの昇格のしきい値が 2 歳以下の場合，コピーガーベージコレクションで回収される可能性の高

いオブジェクトが昇格することになり，フルガーベージコレクションの発生が増す要因になります。この例では，昇格のしきい値を 5 ～ 6 歳程度とするのが，バランスの取れたしきい値となります。

グラフの傾きはシステムによって異なるため，Survivor 領域のオブジェクトの年齢分布を確認し，システムごとの最適な昇格年齢を決定することが重要です。

参考

Java オブジェクトの昇格のしきい値は，コピーガーベージコレクションごとに動的に変更され，-XX:MaxTenuringThreshold=<value> オプションと，Survivor 領域のメモリサイズおよび -XX:TargetSurvivorRatio=<value> オプションに設定した値を基に決まります。-XX:MaxTenuringThreshold=<value> は，昇格のしきい値の最大の年齢です。Java オブジェクトの年齢がこの値を超えると，必ず昇格します。-XX:TargetSurvivorRatio=<value> は，Survivor 領域のメモリ使用率の目標値です。JavaVM は，Survivor 領域のメモリ使用率が，できるだけこの値に近くなるように，昇格のしきい値を決定します。具体的には，コピーガーベージコレクションが終了した時の 1 歳から n 歳までの Java オブジェクトの累計サイズが，目標の使用率となる n を探し，次のコピーガーベージコレクションの昇格のしきい値を n にします。-XX:TargetSurvivorRatio=<value> のデフォルト値は 50% です。
Survivor 領域のメモリ使用率が大きいほど Survivor 領域を有効に利用できますが，Survivor 領域の空きに余裕がなくなるため，オブジェクトの退避が発生しやすくなります。

7.5.2 Java ヒープ内の Eden 領域のメモリサイズの見積もり

Eden 領域のメモリサイズは，コピーガーベージコレクションを発生させる間隔に影響します。Eden 領域のメモリサイズを大きく取ると，コピーガーベージコレクションの発生間隔が長くなります。なお，コピーガーベージコレクションに掛かる時間は，使用中のオブジェクトの個数に影響され，Eden 領域のメモリサイズにはあまり影響されません。このため，コピーガーベージコレクションが頻発しないように，Eden 領域のメモリサイズを十分に確保することが，性能向上には効果的です。

7.6 Java ヒープ内に一定期間存在するオブジェクトの扱いの検討

これまでの説明は，オブジェクトの寿命に応じて，それぞれの領域に次のように格納することを前提としてきました。

- アプリケーションの動作に必要な寿命の長いオブジェクトは，Tenured 領域に格納する。
- リクエスト処理やレスポンス処理などの寿命の短いオブジェクトは，New 領域に格納する。

しかし，寿命が中間的な一定期間使用されるオブジェクトがあります。このようなオブジェクトは，寿命は長くありませんが，何回かのコピーガーベージコレクションの対象になります。

メモリサイズの見積もりでは，これらのオブジェクトを，New 領域，Tenured 領域のどちらかに格納することを前提として，見積もりをする必要があります。

ここでは，それぞれの特徴を示します。アプリケーションの種類や目的に応じて，どちらかのメモリサイズを増加させるように見積もりをしてください。

7.6.1 Java ヒープ内の New 領域に格納する方法

一定期間存在するオブジェクトを New 領域で管理する方法です。New 領域のメモリサイズに，これらの一定期間存在するオブジェクト分のメモリサイズを追加して見積もります。

New 領域サイズを大きくしてオブジェクトの Tenured 領域への移動を抑止することによって，フルガーベージコレクションの発生も抑止できます。ただし，コピーガーベージコレクション実行時に New 領域内にある使用中オブジェクトの数が増えるため，New 領域内でのコピー処理に時間が掛かり，1 回当たりのコピーガーベージコレクション実行時間は長くなります。コピーガーベージコレクションの実行時間がフルガーベージコレクション実行時間よりも長くなるような場合は，メモリサイズの再見積もりが必要です。また，メモリ空間のサイズ設定によっては，本来コピーガーベージコレクションで回収されるはずの寿命の短いオブジェクトが使用する領域が不足して，Tenured 領域への退避が発生するおそれがあります。この場合は，最終的にはフルガーベージコレクションが発生してしまいます。

なお，一定期間存在するオブジェクトを New 領域で管理できているかどうかは，拡張 verbosegc 情報で確認できます。実際にアプリケーションを動作させて，出力された拡張 verbosegc 情報で，コピーガーベージコレクション発生後の Tenured 領域のメモリサイズが大幅に増えていないことを確認してください。

New 領域での管理に失敗している場合，システムの処理性能が大幅に低下していること

があります。また，New 領域で管理できるオブジェクトの最大の年齢には限界があります（限界はプラットフォームやバージョンによって異なります。詳細は，マニュアル「Cosminexus 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「19.4 Cosminexus で指定できる Java HotSpot VM のオプションのデフォルト値」の -XX:MaxTenuringThreshold=<value> を参照してください）。New 領域で管理できていないことがわかった場合は，一定期間存在するオブジェクトは，Tenured 領域に格納して管理することを検討してください。Tenured 領域で管理する方法については，「7.6.2 Java ヒープ内の Tenured 領域に格納する方法」を参照してください。

7.6.2　Java ヒープ内の Tenured 領域に格納する方法

New 領域で管理するオブジェクトの最大の年齢は，
-XX:MaxTenuringThreshold=<value> オプションで指定できます。例えば，
「-XX:MaxTenuringThreshold=2」を指定しておけば，3 回目のコピーガーベージコレクションの対象になったオブジェクトは，すべて Tenured 領域に移動します。

この方法を使用すると，コピーガーベージコレクションの対象になるオブジェクトが少なくなり，実行時間が短縮できます。ただし，多くのオブジェクトが Tenured 領域に移動するため，Tenured 領域がいっぱいになった段階でフルガーベージコレクションが定期的に発生します。システムを安定して動作させるためには，システムに掛かる負荷が低いときなどに，フルガーベージコレクションを強制的に発生させてください。フルガーベージコレクションを強制的に発生させるには，次の方法があります。

- プログラム内で System.gc() メソッドを呼び出す
- javagc コマンドを実行する
  javagc コマンドの詳細については，マニュアル「Cosminexus アプリケーションサーバ リファレンス コマンド編」の「javagc（ガーベージコレクションの強制発生）」を参照してください。

7.6.3　Explicit ヒープに格納する方法

チューニングではありませんが，Java プログラムを変更することで，一定期間存在するオブジェクトを Explicit ヒープに格納して管理する方法があります。Explicit ヒープはガーベージコレクションの対象とならない領域です。Explicit ヒープを利用することで，オブジェクトが Tenured 領域に移動することを防ぎ，フルガーベージコレクションの発生を抑えることができます。また，自動配置設定ファイルを利用することで，Explicit ヒープに直接オブジェクトを配置することもできます。詳細は，「7.10　Explicit ヒープのチューニング」を参照してください。アプリケーションサーバでは，HTTP セッションに関連するオブジェクトを Explicit ヒープで管理しています。

7.7 Java ヒープの最大サイズ／初期サイズの決定

Tenured 領域および New 領域の見積もりができたら，それらを基に Java ヒープの最大サイズと初期サイズを決定します。

Java ヒープの最大サイズは，次のように決定します。

Java ヒープの最大サイズ
=Tenured 領域のメモリサイズ +New 領域のメモリサイズ

Java ヒープのメモリサイズを設定する場合，まず，-Xmx オプションに，拡張領域のサイズも含む，Java ヒープ領域の最大サイズを指定します。次に，-Xms オプションに，Java ヒープ領域の初期サイズを指定します。なお，-Xms オプションに指定するサイズは，-Xmx オプションに指定したサイズよりも小さくする必要があります。

JavaVM は，起動時に，-Xms オプションで指定された分のメモリ領域を Java ヒープ領域として確保します。その後，アプリケーション実行中に -Xms オプションで指定した以上のメモリ領域が必要になった場合に，-Xmx オプションで指定したサイズになるまで，ヒープ領域を追加して割り当てていきます。逆に，アプリケーションの中で不要なメモリ領域が発生した場合は，-Xms オプションで指定したサイズまで，Java ヒープ領域として確保している領域を減らしていきます。

なお，システムの安定稼働のためには，-Xmx オプションと -Xms オプションには同じ値を指定することをお勧めします。

7.8 Java ヒープ内の Permanent 領域のメモリサイズの見積もり

この節では，Permanent 領域のメモリサイズの見積もりについて説明します。
Permanent 領域は，ロードされた class などが格納される領域です。

Permanent 領域のメモリサイズは，「7.1.2 JavaVM で使用するメモリ空間の構成と JavaVM オプション」で示したとおり，-Xmx オプションで指定したメモリサイズ（Java ヒープ）とは別に確保されます。

デフォルト値については，マニュアル「Cosminexus 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「19.4 Cosminexus で指定できる Java HotSpot VM のオプションのデフォルト値」を参照してください。

次に，Permanent 領域の使用量の見積もり方法を示します。

● Permanent 領域の使用量の見積もり方法

Permanent 領域でのメモリ使用量は，大体，J2EE サーバにロードされるクラスファイルの合計サイズになります。アプリケーションサーバの場合，次に示すクラスファイルのサイズの総和から見積もることができます。

1. WEB-INF/classes 以下のすべてのクラスファイル
2. WEB-INF/lib 以下の JAR ファイルに含まれる，すべてのクラスファイル
3. JSP コンパイル結果として生成された，すべてのクラスファイル
4. EJB-JAR に含まれるすべてのクラスファイル
5. コンテナ拡張ライブラリ，ライブラリ JAR，参照ライブラリを利用している場合に追加する JAR ファイルに含まれる，すべてのクラスファイル
6. アプリケーションサーバ提供のクラスファイル（システムクラスファイル）
   システムクラスファイルの総和は約 80MB（Service Platform および Service Architect を使用する場合には約 90MB）になります。

Permanent 領域のメモリサイズは，-XX:MaxPermSize=<size> オプション，および -XX:PermSize=<size> オプションで指定します。これらのオプションのデフォルト値については，マニュアル「Cosminexus 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「19.4 Cosminexus で指定できる Java HotSpot VM のオプションのデフォルト値」を参照してください。

なお，アプリケーションのインポート時に一時的に Permanent 領域の使用量が増加する場合があります。

7.9　拡張 verbosegc 情報を使用したフルガーベージコレクションの要因の分析方法

この節では，拡張 verbosegc 情報を使用したフルガーベージコレクションの要因の分析方法について説明します。拡張 verbosegc 情報は，日立固有の JavaVM オプションである -XX:+HitachiVerboseGC オプションを指定することによって出力できる JavaVM のログ情報です。チューニングに役立つ情報のほか，障害要因の分析にも役立つ情報が出力されます。

チューニング実行時に拡張 verbosegc 情報を参照することで，次の情報を確認できます。

- **ガーベージコレクション実行前と実行後の各領域の使用メモリサイズ**
- **ガーベージコレクションが発生した要因**

また，-XX:+HitachiVerboseGC とほかの日立固有の JavaVM オプションを組み合わせることによって，さらに詳細な情報が出力できます。-XX:+HitachiVerboseGC オプション，およびそのほかの日立固有の JavaVM 拡張オプションの詳細については，マニュアル「Cosminexus 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「19.2　日立固有の JavaVM 拡張オプションの詳細」を参照してください。

7.9.1　拡張 verbosegc 情報の出力形式の概要

拡張 verbosegc 情報は，コピーガーベージコレクションが発生した場合と，フルガーベージコレクションが発生した場合に出力されます。

コピーガーベージコレクションが発生すると，ガーベージコレクションの種類として「GC」と出力されます。また，フルガーベージコレクションが発生すると，ガーベージコレクションの種類として「Full GC」と出力されます。種類に続いて，それぞれの領域の「＜ガーベージコレクション前の領域長＞ -> ＜ガーベージコレクション後の領域長＞(＜確保済み領域サイズ＞)」が出力されます。

以降に，フルガーベージコレクションが発生した場合の拡張 verbosegc 情報の出力例を示します。拡張 verbosegc 情報には，ほかにもガーベージコレクションの発生要因や GC スレッドの CPU 時間も出力されます。

拡張 verbosegc 情報の出力形式の詳細，およびそれぞれのオプションの詳細については，マニュアル「Cosminexus 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「19.　JavaVM 起動オプション」を参照してください。

7.9.2 フルガーベージコレクション発生時の拡張 verbosegc 情報の出力例

ここでは，フルガーベージコレクションの発生時の拡張 verbosegc 情報の出力例を示します。

（1） New 領域（Eden 領域と Survivor 領域の合計）で使用しているメモリサイズが Tenured 領域の最大値に対する未使用メモリサイズを上回った場合

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Wed May 11 23:12:05 2005>[GC 27340K->27340K(32704K), 0.0432900
secs][DefNew::Eden: 3440K->0K(3456K)][DefNew::Survivor:
58K->58K(64K)][Tenured: 23841K->27282K(29184K)][Perm:
1269K->1269K(4096K)][cause:ObjAllocFail][User: 0.0156250 secs][Sys: 0.0312500
secs]
[VGC]<Wed May 11 23:12:05 2005>[Full GC 30780K->30780K(32704K), 0.2070500
secs][DefNew::Eden: 3440K->1602K(3456K)][DefNew::Survivor:
58K->0K(64K)][Tenured: 27282K->29178K(29184K)][Perm:
1269K->1269K(4096K)][cause:ObjAllocFail][User: 0.0156250 secs][Sys: 0.0312500
secs]
…
```

この出力例からは，次のことがわかります。

- New 領域で使用しているメモリサイズ（3440K+58K=3498K）が，Tenured 領域の最大値に対する未使用メモリサイズ（29184K-27282K=1902K）を上回りました。
- フルガーベージコレクションの要因は，オブジェクトの割り付け失敗です。

（2） アロケーションしたいメモリサイズ（new で作成する Java オブジェクトのサイズ）が Tenured 領域の未使用メモリサイズを上回る場合

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Wed May 11 23:53:18 2005>[GC 28499K->28490K(32704K), 0.0540590
secs][DefNew::Eden: 808K->0K(3456K)][DefNew::Survivor: 64K->62K(64K)][Tenured:
27626K->28428K(29184K)][Perm: 1269K->1269K(4096K)][cause:ObjAllocFail][User:
0.0156250 secs][Sys: 0.0312500 secs]
[VGC]<Wed May 11 23:53:18 2005>[Full GC 28490K->8959K(32704K), 0.1510380
secs][DefNew::Eden: 0K->0K(3456K)][DefNew::Survivor: 62K->0K(64K)][Tenured:
28428K->8959K(29184K)][Perm: 1269K->1269K(4096K)][cause:ObjAllocFail][User:
0.0156250 secs][Sys: 0.0312500 secs]
…
```

この出力例からは，次のことがわかります。

- Tenured 領域の未使用メモリサイズ（29184K-28428K=756K）を上回るメモリサイズの Java オブジェクトを，new で作成しようとしました。
- フルガーベージコレクションの要因は，オブジェクトの割り付け失敗です。

（3） コピーガーベージコレクション実施の結果，確保済み Tenured 領域の未使用メモリサイズが 10,000 バイトを下回った場合

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Fri May 25 15:21:33 2007>[GC 15436K->15416K(19840K), 0.0111825
secs][DefNew::Eden: 4413K->0K(4416K)][DefNew::Survivor:
512K->509K(512K)][Tenured: 10511K->14906K(14912K)][Perm:
1976K->1976K(8192K)][cause:ObjAllocFail][User: 0.0000000 secs][Sys: 0.0000000
secs]
[VGC]<Fri May 25 15:21:33 2007>[Full GC 15416K->8622K(19840K), 0.0284614
secs][DefNew::Eden: 0K->0K(4416K)][DefNew::Survivor: 509K->0K(512K)][Tenured:
14906K->8622K(14912K)][Perm: 1976K->1976K(8192K)][cause:ObjAllocFail][User:
0.0312500 secs][Sys: 0.0000000 secs]
…
```

この出力例からは，次のことがわかります。

- 1 行目のコピーガーベージコレクションで New 領域から Tenured 領域にオブジェクトが移動したことによって，Tenured 領域の使用済みメモリサイズが 10,511 キロバイトから 14,906 キロバイトに増加しました。これによって，確保済み Tenured 領域の未使用メモリサイズが 14,912 キロバイト -14,906 キロバイト =6 キロバイトとなり，10,000 バイト（約 10 キロバイト）を下回りました。
- 1 行目のコピーガーベージコレクションの原因は，オブジェクトの割り付け失敗です。1 行目のコピーガーベージコレクションと 2 行目のフルガーベージコレクションは，Java プログラムに制御が戻る前に連続して発生します。

（4） コピーガーベージコレクション実施時の Tenured 領域へのオブジェクトの移動によって，確保済み Tenured 領域の拡張が発生した場合

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Fri May 25 15:42:00 2007>[GC 12745K->10151K(15872K), 0.0048346
secs][DefNew::Eden: 4416K->0K(4416K)][DefNew::Survivor:
137K->512K(512K)][Tenured: 8192K->9639K(10944K)][Perm:
1976K->1976K(8192K)][cause:ObjAllocFail][User: 0.0156250 secs][Sys: 0.0000000
secs]
[VGC]<Fri May 25 15:42:00 2007>[GC 14563K->14536K(19072K), 0.0104957
secs][DefNew::Eden: 4412K->0K(4416K)][DefNew::Survivor:
512K->510K(512K)][Tenured: 9639K->14026K(14144K)][Perm:
1976K->1976K(8192K)][cause:ObjAllocFail][User: 0.0156250 secs][Sys: 0.0000000
secs]
[VGC]<Fri May 25 15:42:00 2007>[Full GC 14536K->8610K(19072K), 0.0287254
secs][DefNew::Eden: 0K->0K(4416K)][DefNew::Survivor: 510K->0K(512K)][Tenured:
14026K->8610K(14144K)][Perm: 1976K->1976K(8192K)][cause:ObjAllocFail][User:
0.0312500 secs][Sys: 0.0000000 secs]
…
```

この出力例からは，次のことがわかります。

- 2 行目のコピーガーベージコレクションで New 領域から Tenured 領域へのオブジェクトが移動したことによって，Tenured 領域が最低でも 14,026 キロバイト以上必要になりました。このため，確保済み Tenured 領域サイズが 10,944 キロバイトから 14,144 キロバイトに拡張されました。
- 2 行目のコピーガーベージコレクションの原因は，オブジェクトの割り付け失敗です。2 行目のコピーガーベージコレクションと 3 行目のフルガーベージコレクションは，Java プログラムに制御が戻る前に連続して発生します。

(5) アプリケーション内で java.lang.System.gc() メソッドが実行された場合

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Mon Apr 18 20:36:29 2005>[Full GC 330K->150K(3520K), 0.0387690
secs][DefNew::Eden: 330K->0K(2048K)][DefNew::Survivor: 0K->0K(64K)][Tenured:
0K->150K(1408K)][Perm: 1266K->1266K(4096K)][cause:System.gc][User: 0.0156250
secs][Sys: 0.0312500 secs]
…
```

この出力例からは，次のことがわかります。

- フルガーベージコレクションの要因は，J2EE アプリケーション内またはバッチアプリケーション内での java.lang.System.gc() メソッド呼び出しです。

(6) Permanent 領域にアロケーションしたいメモリサイズが確保済み Permanent 領域の未使用メモリサイズを上回る場合

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Mon Apr 18 20:36:29 2005>[Full GC 57051K->25121K(129792K), 0.5531230
secs][DefNew::Eden: 40943K->0K(41088K)][DefNew::Survivor:
1280K->0K(1280K)][Tenured: 14827K->25121K(87424K)][Perm:
20479K->20479K(20480K)][cause:PermAllocFail][User: 0.0156250 secs][Sys:
0.0312500 secs]
…
```

この出力例からは，次のことがわかります。

- Permanent 領域にアロケーションしようとしたメモリサイズが，確保済み Permanent 領域の未使用メモリサイズ（20480K-20479K=1K）を上回りました。
- フルガーベージコレクションの要因は，Permanent ヒープの割り付け失敗です。

（7）javagc コマンドを実行した場合の出力例

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Mon Apr 18 21:46:50 2005>[Full GC 369K->189K(3520K), 0.0403010
secs][DefNew::Eden: 369K->0K(2048K)][DefNew::Survivor: 0K->0K(64K)][Tenured:
0K->189K(1408K)][Perm: 1266K->1266K(4096K)][cause:JavaGC Command][User:
0.0156250 secs][Sys: 0.0312500 secs]
…
```

この出力例からは，次のことがわかります。

- フルガーベージコレクションの要因は，javagc コマンド実行です。

（8）jheapprof コマンドを実行した場合の出力例

拡張 verbosegc 情報の出力例を次に示します。**太字**の部分がフルガーベージコレクション発生の要因を示す個所です。

```
…
[VGC]<Mon Apr 18 21:46:50 2005>[Full GC 369K->189K(3520K), 0.0403010
secs][DefNew::Eden: 369K->0K(2048K)][DefNew::Survivor: 0K->0K(64K)][Tenured:
0K->189K(1408K)][Perm: 1266K->1266K(4096K)][cause:JHeapProf Command][User:
0.0156250 secs][Sys: 0.0312500 secs]
…
```

この出力例からは，次のことがわかります。

- フルガーベージコレクションの要因は，jheapprof コマンド実行です。

7.10 Explicit ヒープのチューニング

ここでは，Explicit ヒープのチューニングについて説明します。

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

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

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

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

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

リダイレクタとの通信用オブジェクトが利用する Explicit ヒープのメモリサイズは，定義ファイルでの設定値から算出できます。見積もり方法については，「7.10.2 リダイレクタとの通信用オブジェクトで利用するメモリサイズ」を参照してください。

HTTP セッションに関するオブジェクトが利用する Explicit ヒープのメモリサイズは，実際にアプリケーションを動作させて情報を取得した上で見積もります。見積もり方法については，「7.10.3 HTTP セッションに関するオブジェクトで利用するメモリサイズ」を参照してください。

7.10.2 リダイレクタとの通信用オブジェクトで利用するメモリサイズ

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

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

注※

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

表 7-3 明示管理ヒープの自動配置機能の使用の有無による 1 コネクションで使用するメモリサイズ

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

（凡例）

○：明示管理ヒープの自動配置機能を使用する。

×：明示管理ヒープの自動配置機能を使用しない。

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

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

表 7-4 最大接続数の設定個所

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

7.10.3 HTTP セッションに関するオブジェクトで利用するメモリサイズ

ここでは，HTTP セッションに関するオブジェクトで利用するメモリサイズの見積もりについて説明します。

注意事項

なお，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場合，この手順での見積もりはできません。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場合は，「7.10.5 稼働情報による見積もり」で示す手順でメモリサイズを見積もってください。

HTTP セッションに関するオブジェクトで利用する Explicit ヒープのメモリサイズは，次の式で見積もります。

```
HTTPセッションに関するオブジェクトで利用するExplicitヒープのメモリサイズ
= HTTPセッションで利用するExplicitヒープのメモリサイズ
 + Webコンテナ内部で利用するExplicitヒープ領域のメモリサイズ
```

HTTP セッションで利用する Explicit ヒープのメモリサイズは，次の式で見積もります。

```
HTTPセッションで利用するExplicitヒープのメモリサイズ
= 1セッションで使用するメモリサイズ※1×システムで必要なセッション数
```

HTTP セッションのために，Web コンテナ内部で利用する Explicit ヒープ領域のメモリサイズは，次の式で見積もります。

```
Webコンテナ内部で利用するExplicitヒープ領域のメモリサイズ
= HTTPセッション管理用オブジェクトのサイズ※2×（Webアプリケーションの数※3 + 2）
```

注※ 1

1 セッションで使用するメモリサイズは，Explicit メモリブロック 1 個のサイズに相当します。Explicit メモリブロック 1 個のサイズは，実際にアプリケーションを動作させて，Explicit ヒープの使用状況を確認しながら見積もります。なお，Explicit メモリブロックの最小サイズは，明示管理ヒープの自動配置機能の使用の有無によって異なります。明示管理ヒープの自動配置機能の使用の有無による Explicit メモリブロックの最小サイズを次に示します。

表 7-5 明示管理ヒープの自動配置機能の使用の有無による Explicit メモリブロックの最小サイズ

項番	明示管理ヒープの自動配置機能使用の有無	Explicit メモリブロックの最小サイズ
1	○	16 キロバイト
2	×	64 キロバイト

（凡例）

○：明示管理ヒープの自動配置機能を使用する。

×：明示管理ヒープの自動配置機能を使用しない。

なお，Explicit メモリブロックは 64 キロバイト単位で拡張されます。そのため，実際は「Explicit メモリブロック 1 個のサイズ≧ 1 セッションで使用するメモリサイズ」となります。また，明示管理ヒープの自動配置機能を使用する場合は，さらに 16 キロバイトを加算して見積もってください。

注※ 2

HTTP セッション管理用オブジェクトのサイズは表 7-5 に示す Explicit メモリブロックの最小サイズです。

注※ 3

「Web アプリケーションの数」は，開始している Web アプリケーションの個数を表します。

（1） 見積もり手順

見積もり手順を示します。

1. **リダイレクタとの通信用オブジェクトを** Explicit **ヒープに配置しないように** J2EE **サーバの設定を変更します。**

   J2EE サーバの設定は，簡易構築定義ファイルの論理 J2EE サーバの J2EE サーバ用ユーザプロパティを設定するパラメタに設定してください。設定値を次に示します。

   param-name **指定値**

   ```
   ejbserver.server.eheap.ajp13.enabled
   ```

   param-value **指定値**

   ```
   false
   ```

2. HTTP **セッションを作成するアプリケーションを開始して，セッション破棄（ログアウトなど）の直前まで処理を実行します。**

3. javagc **コマンドを実行して，フルガーベージコレクションを発生させます。**

4. **明示管理ヒープ機能のイベントログに出力されたフルガーベージコレクション実行後の情報を確認します。**

   明示管理ヒープ機能のイベントログは，JavaVM の起動オプションである -XX:HitachiExplicitMemoryJavaLog オプションで指定したファイルまたはディレクトリに出力されます。デフォルトの出力先は，次のとおりです。

   Windows **の場合**

   <Cosminexus インストールディレクトリ >¥CC¥server¥public¥ejb¥<J2EE サーバ名 >¥logs¥ehjavalog[n].log

   UNIX **の場合**

   /opt/Cosminexus/CC/server/public/ejb/<J2EE サーバ名 >/logs/ehjavalog[n].log

ここでは，一連の業務で必要な Explicit ヒープの利用サイズと Explicit メモリブロックの個数を確認します。
メモリサイズ算出時には，イベントログの出力項目のうち，次の項目を確認します。

- Explicit ヒープの確保済みサイズ
- Explicit メモリブロックの個数

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「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「5.11　明示管理ヒープ機能のイベントログの内容」を参照してください。これらの項目は，ガーベージコレクション発生時に出力される，Explicit ヒープ利用状況に含まれます。

5. **手順 4. で確認した内容を基に，Explicit メモリブロック 1 個のサイズを算出します。**
次の式で算出できます。

```
Explicitメモリブロック1個のサイズ
=Explicitヒープの確保済みサイズ／Explicitメモリブロックの個数
```

Explicit メモリブロック 1 個のサイズは，1 セッションで使用するメモリサイズに相当します。

6. **手順 5. で算出した値に，業務で必要になるセッション数を掛け，HTTP セッションで使用する Explicit ヒープの合計メモリサイズを算出します。**

(2) 見積もり例

ここでは，明示管理ヒープ機能のイベントログの出力例を基に，見積もり例を示します。
明示管理ヒープ機能のイベントログの出力例を次に示します。

明示管理ヒープ機能のイベントログの出力例

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12672K->12800K(12800K/65536K)][E/F/D:
200/0/0][cause:GC][CF: 0]
```

この例では，Explicit ヒープの確保済みサイズは 12,800 キロバイト，Explicit メモリブロックの個数は 200 個と出力されています。この値を「(1) 見積もり手順」の手順 5. で示した式に当てはめると，次のようになります。

Explicit メモリブロックサイズの見積もり例

```
Explicitメモリブロック1個のサイズ
=Explicitヒープの確保済みサイズ（12,800キロバイト）/Explicitメモリブロックの個数（200）
=64キロバイト
```

これに業務で想定するセッション数を掛けた値が，HTTP セッションで使用する Explicit ヒープの合計メモリサイズになります。

7.10.4 明示管理ヒープ機能利用によるメモリサイズの見積もりへの影響

アプリケーションサーバの機能の中には，その機能を使用するかどうかで Explicit ヒープ領域のメモリサイズに影響が出るものがあります。機能利用の有無による見積もりへの影響について表に示します。

表 7-6 機能利用の有無による見積もりへの影響

項番	機能	機能利用の有無による Explicit ヒープ領域の違い	見積もりへの影響
1	明示管理ヒープの自動解放機能（-XX:+HitachiExplicitMemoryAutoReclaim オプション）	有効な場合 Explicit ヒープ領域の中で「Java ヒープの Survivor 領域のサイズ× 2」の領域を JavaVM が使用します。 無効な場合 JavaVM は Explicit ヒープ領域を使用しません。	機能が有効な場合の最終的な Explicit ヒープ領域の見積もりサイズは，稼働情報から算出した見積もりサイズに「Java ヒープの Survivor 領域のサイズ× 2」を加算した値となります。
2	明示管理ヒープの自動配置機能（-XX:+HitachiAutoExplicitMemory オプション）	有効な場合 Explicit メモリブロックの最小サイズが 16 キロバイトになります。 無効な場合 Explicit メモリブロックの最小サイズが 64 キロバイトになります。 また，見積もりには影響しませんが，機能の有効・無効によって Explicit ヒープ領域の確保方法が変わります。 有効な場合 プロセス起動時に -XX:HitachiExplicitHeapMaxSize オプションで指定したサイズのメモリを確保します。 無効な場合 Explicit メモリブロック取得時に，必要なだけのメモリサイズを確保します。	機能の有効・無効によって，稼働情報に出力される Explicit ヒープサイズに関する情報が異なります。

7.10.5 稼働情報による見積もり

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

稼働情報の出力内容，出力するための設定，および稼働情報ファイルの出力先について

は，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「3.3　稼働情報ファイルの出力機能」を参照してください。

(1) 稼働情報を使用した見積もりの考え方

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

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

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

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

表 7-7　Explicit ヒープ領域を使用するものの具体例

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

(2) 見積もりに使用する稼働情報取得時の注意

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

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

- 各定義ファイルに設定するプロパティ，およびオプションに指定する値
- サーバに登録されている Web アプリケーションの数
- リダイレクタとのコネクションの数
- 業務アプリケーションが処理するデータのサイズ
- 一定時間内に処理するデータの数

また，見積もりのために稼働情報を取得する場合，Explicit ヒープ領域を使い切った状

態にならないよう Explicit ヒープ領域サイズの最大値を設定するオプションを指定してください。

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

(3) 見積もり方法

稼働情報を基にした見積もり方法を次に示します。

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

HTTP セッションで利用するメモリサイズは，「7.10.3　HTTP セッションに関するオブジェクトで利用するメモリサイズ」で示した HTTP セッションで利用する Explicit ヒープのメモリサイズの式で求めます。このとき，式に含まれる「1 セッションで使用するメモリサイズ」を稼働情報で確認できます。

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

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

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

(b) コンテナで利用する Explicit ヒープ領域のメモリサイズ

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

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

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

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

（4） 稼働情報の確認手順

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

見積もり式

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

それぞれの値の確認方法について説明します。

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

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

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

図 7-13　HTTP セッションで利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例

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

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

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

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

(b) コンテナで利用する Explicit ヒープ領域のメモリサイズ

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

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

図 7-14　コンテナで利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例

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

1

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

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

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

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

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

図 7-15 アプリケーションおよび JavaVM で利用する Explicit ヒープ領域のメモリサイズの稼働情報の出力例

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

1

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

(d) 必要な Explicit ヒープ領域のメモリサイズ

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

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

明示管理ヒープの自動配置設定ファイルを使用する場合，「Java ヒープの Survivor 領域のサイズ× 2」を足した値が，最終的な Explicit ヒープ領域の見積もりサイズとなります。

7.11　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メモリサイズの見積もり

ユーザが作成したアプリケーションに Tenured 領域のメモリサイズ増加の要因になっているオブジェクトがある場合，該当するオブジェクトを Explicit ヒープに配置できます。ここでは，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メモリサイズの見積もりについて説明します。

ポイント

この節の説明は，J2EE サーバを含めた，JavaVM 上で動作するすべての Java アプリケーションに該当します。ただし，J2EE サーバで使用する Explicit ヒープだけを使用する場合，必ず読む必要はありません。必要に応じて参照してください。

7.11.1　アプリケーションで明示管理ヒープ機能を使用するかどうかの検討

Java ヒープのチューニングおよび「7.10.1　Explicit ヒープのメモリサイズの見積もり（J2EE サーバが使用するメモリサイズの見積もり）」の手順を実施してもフルガーベージコレクションが頻発する場合は，アプリケーションでの明示管理ヒープ機能の使用を検討します。

まず，フルガーベージコレクションの発生要因となっているオブジェクトを調査します。特定のオブジェクトを Explicit ヒープに配置することでフルガーベージコレクション発生を抑止できる場合は，明示管理ヒープ機能を適用します。

ただし，Explicit ヒープに配置するオブジェクトは，ライフサイクルが既知であることが必要です。オブジェクトの生成のタイミングおよびオブジェクトが不要になるタイミングが Java プログラム上で明確な場合に，適用を検討してください。

明示管理ヒープ機能を適用するためには，自動配置設定ファイル，または明示管理ヒープ機能 API を使用します。明示管理ヒープ機能を使用したフルガーベージコレクションの抑止については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.　明示管理ヒープ機能を使用したフルガーベージコレクションの抑止」を参照してください。

7.11.2　見積もりの考え方

アプリケーションが利用する Explicit ヒープのメモリサイズは，運用開始前に見積もります。実際にアプリケーションを動作させた上で，明示管理ヒープ機能のイベントログを確認して見積もります。見積もり方法については，「7.11.3　アプリケーションが使用するメモリサイズ」を参照してください。

アプリケーション開発中および運用開始後に Explicit ヒープの使用状況などを調査する場合は，日立 JavaVM ログファイルや Java API を使用します。開発中や運用開始後の調査の手順については，「7.11.4　アプリケーション開発・運用時の確認・調査」を参照してください。

参考

メモリサイズの見積もりに利用できる情報は，稼働情報にも出力されます。稼働情報を利用したメモリサイズのチューニングについては「7.10.5　稼働情報による見積もり」を参照してください。
なお，ここでは稼働情報に出力される項目と，スレッドダンプに出力される項目の対応についても説明します。

7.11.3　アプリケーションが使用するメモリサイズ

運用を開始する前に，アプリケーションが使用する Explicit ヒープのメモリサイズを見積もり，-XX:HitachiExplicitHeapMaxSize オプションに設定します。

メモリサイズは，実際に明示管理ヒープ機能を実装したアプリケーションを動作させてテストを実行し，出力されたログを確認して見積もります。見積もった値を，本番で使用する実行環境の -XX:HitachiExplicitHeapMaxSize オプションに設定します。

ここでは，テストを実行する環境に応じた 2 種類の見積もり方法について説明します。

どちらの方法でテストを実行する場合も，次のことが前提になります。

見積もりの前提

- Explicit ヒープの最大サイズを十分に大きく設定してテストを実行してください。
- -XX:HitachiExplicitMemoryLogLevel オプションには「none」以外を設定してください。

見積もりで使用するイベントログの出力項目については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「5.11　明示管理ヒープ機能のイベントログの内容」を参照してください。

(1) 本番環境と同等の環境でテストを実行できる場合

本番環境と同等の環境でテストを実行できる場合，イベントログに出力された Explicit ヒープの確保済みメモリサイズの最大値を Explicit ヒープのメモリサイズとします。

確認手順を示します。

1. **テスト用の環境でアプリケーションを一とおり実行します。**
   アプリケーション実行中にガーベージコレクションが発生した時に，明示管理ヒープ機能のイベントログが出力されます。
2. **出力されたイベントログのすべてのレコード（行）のうち，Explicit ヒープの確保済**

みメモリサイズ（<EH_TOTAL>）が最大の値を確認します。

この値を，Explicit メモリヒープのメモリサイズとしてください。

（2） 本番環境よりも小さなスケールの環境でテストを実行する場合

本番環境よりも小さなスケールの環境でテストを実行する場合，本番環境で必要なメモリサイズは次の式で見積もります。

```
本番環境でのExplicitヒープサイズ
= （Explicitヒープの最大サイズ)/(Explicitメモリブロックの数） × 本番環境でのExplicitメモリブロック数
+ Survivor領域サイズ × 2※
```

注※

自動配置機能が有効な場合に「Survivor 領域サイズ × 2」を加算します。

確認手順を示します。

1. **テスト用の環境でアプリケーションを一とおり実行します。**
   アプリケーション実行中にガーベージコレクションが発生した時に，明示管理ヒープ機能のイベントログが出力されます。
2. **出力されたイベントログのすべてのレコード（行）のうち，**Explicit **ヒープの確保済みメモリサイズ（**<EH_TOTAL>**）が最大の値を確認します。また，同じレコードに出力されている有効な** Explicit **メモリブロックの数（**<AC_NUM> **と** <DA_NUM> **の合計）を確認します。**
3. 2. **で確認した** <EH_TOTAL> **の値を（**<AC_NUM>+<DA_NUM>**）で割ります。**
   Explicit メモリブロック一つ当たりのおおよそのサイズを算出できます。
4. 3. **で算出した値に，本番環境で予測される最大の** Explicit **メモリブロック数を掛けます。**

注

この方法は，開発環境と本番環境での Explicit メモリブロックの一つ当たりのサイズが同じであり，数だけが異なる場合に適用できます。
また，Explicit ヒープを複数の用途に利用している場合，用途ごと（見積もり対象の用途以外の Explicit メモリブロックを利用しない状態）で確認する必要があります。

ポイント

スレッドダンプに出力される項目，および稼働情報に出力される項目の対応を次の表に示します。

表 7-8 スレッドダンプおよび稼働情報の出力項目の対応

スレッドダンプの出力項目	稼働情報の出力項目	出力内容
<EH_TOTAL>	EHeapSize	Explicit ヒープの確保済みメモリサイズ
<AC_NUM> + <DA_NUM>	EMemoryBlockCount	同じレコードに出力されている有効な Explicit メモリブロックの数

7.11.4 アプリケーション開発・運用時の確認・調査

次のような場合には，ログに出力された内容や API で取得した内容を基に，メモリサイズをチューニングします。

- アプリケーション開発中にテストを実行する場合
- 運用を開始したあとでチューニングを実行する場合
- 運用を開始したあとで問題が発生したときに対処する場合

これらの場合に必要な確認・調査方法について説明します。

(1) Explicit ヒープのある時点での利用状況（スナップショット）の調査

Explicit ヒープのある時点での利用状況（スナップショット）を調査する方法には，スレッドダンプを確認する方法と，Java API で情報を取得する方法があります。

● スレッドダンプを確認する方法

cjdumpsv コマンドを実行すると，任意のタイミングでスレッドダンプを出力できます。スレッドダンプには，Explicit ヒープおよび各 Explicit メモリブロックの利用状況が出力されます。
出力例を次に示します。

```
Explicit Heap Status
--------------------
 max 65536K, total 21376K, used 20480K, garbage 1234K (31.2% used/max, 95.8%
used/total, 6.0% garbage/used), 1 spaces exist
 Explicit Memories(0x12345678)
  "EJBMgrData" eid=1(0x02f25610)/R, total 21376K, used 20480K, garbage 1234K
(95.8% used/total, 6.0% garbage/used, 0 blocks) Enable
```

太字の部分が，Explicit ヒープおよび個別の Explicit メモリブロックの利用状況です。この例の場合，Explicit ヒープの最大サイズは 65,536 キロバイト，確保済み Explicit ヒープサイズは 21,376 キロバイトです。また，「EJBMgrData」という名称の Explicit メモリブロックのメモリ確保済みサイズは 21,376 キロバイト，利用済みサイズは 20,480 キロバイトであることがわかります。

● Java API で情報を取得する方法

日立の JavaVM の Java API を使用して，Explicit ヒープおよび Explicit メモリブロックの利用状況を取得できます。次に示す API を使用してアプリケーションを実装することで，任意の処理のタイミングで情報を取得できます。

Explicit ヒープの利用状況を取得するメソッド

JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.getMemoryUsage()

Explicit メモリブロック利用状況を取得するメソッド

JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.freeMemory()
JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.totalMemory()
JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.usedMemory()

(2) 利用状況の推移の調査

Explicit ヒープの利用状況の推移を調査する方法には，明示管理ヒープ機能のイベントログを確認する方法と，Java API で情報を取得する方法があります。Java API を使用する方法については，「(1) Explicit ヒープのある時点での利用状況（スナップショット）の調査」で示した API を使用します。

ここでは，明示管理ヒープ機能のイベントログを使用した調査方法について説明します。

(a) Explicit ヒープの利用状況の推移

JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「normal」を指定します。これによって，次のタイミングで Explicit ヒープの利用状況が出力されます。

- Explicit メモリブロックの解放時
- ガーベージコレクション発生時（定期的）

また，JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「verbose」を指定すると，「normal」で出力されるタイミングに加えて，次のタイミングで Explicit ヒープの利用状況が出力されるようになります。

- ExplicitMemory.newInstance メソッドなどによる Explicit メモリブロックへのオブジェクト生成時

出力例を次に示します。

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12672K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
```

太字で示した「EH:」で始まる部分が，Explicit ヒープの利用状況を示します。ログに出力された内容のうち，Explicit ヒープの利用状況を示す行には，必ず**太字**に該当する情報が含まれます。この値をグラフなどに記してプロットすることによって，利用状況の推移を確認できます。

なお，利用状況を示すログの先頭は，[ENS] または [EVS] で開始されます。この文字列でイベントログをフィルタリングすると，利用状況を確認しやすくなります。

(b) Explicit メモリブロックごとの利用状況の推移

JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「verbose」を指定します。これによって，次のタイミングでサイズ変化のあった Explicit メモリブロックの利用状況が出力されます。

- Explicit メモリブロックへのオブジェクト移動時
- Explicit メモリブロックへのオブジェクト生成時

出力例を示します。

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 11422K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
[EVS]["REM2" eid=2/R: 0K->88K(128K)]["REM3" eid=3/R: 30K->230K(256K)]["REM6"
eid=6/R: 30K->200K(256K)]¥
["Session1" eid=8/R: 30K->250K(256K)]["Session2" eid=10/R: 30K->250K(256K)]
[EVS]["Session3" eid=12/R: 30K->510K(512K)]
```

太字で示した部分が，「"REM2"」という名称の一つの Explicit メモリブロックについての利用状況を示します。

また，-XX:HitachiExplicitMemoryLogLevel オプションに「verbose」を指定した場合，Explicit メモリブロックの解放時に，解放した Explicit メモリブロックの情報が出力されます。出力例を示します。

```
[ENS]<Tue Jul 24 01:23:51 2007>[EH: 12800K->11776K(11776K/65536K), 0.1129602
secs][E/F/D: 523/0/0]¥
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 12K->0K(17400K)][Tenured:
103400K->103400K(556800K)][cause:Reclaim]
[EVS]["REM2" eid=2/R: 320K]["BEM3" eid=5/B: 320K]["BEM1" eid=7/B: 384K]
```

太字で示した部分が，解放された「"REM2"」という名称の Explicit メモリブロックの情報を示しています。「320K」は，解放されたメモリのサイズ（確保済みだった Explicit メモリブロックのサイズ）です。

これらの値をグラフなどに記してプロットすることによって，Explicit メモリブロックごとの利用状況の推移を確認できます。なお，個々の Explicit メモリブロックのサイズは，解放するまで単調増加します。

(3) Explicit ヒープあふれが発生した場合の確認と対処

Explicit ヒープあふれが発生した場合の確認と対処について説明します。

Explicit ヒープあふれとは，次の状態を示します。

- Explicit ヒープを最大サイズまで使い切った状態
- Explicit メモリブロック拡張時に OS からのメモリ確保に失敗する状態

Explicit ヒープあふれが発生すると，発生後に領域を拡張しようとした Explicit メモリブロックのサブ状態が，「Enable」から「Disable」に変わります。「Disable」になった Explicit メモリブロックには，オブジェクトを配置できません。

Explicit ヒープあふれが発生しているかどうかは，明示管理ヒープ機能のイベントログまたはスレッドダンプの内容から調査できます。また，Java API で取得した情報で確認することもできます。

Explicit ヒープあふれが発生した場合は，次の対処を実施してください。

Explicit ヒープあふれが発生した場合の対処

- Explicit ヒープの最大サイズを増やす。
  -XX:HitachiExplicitHeapMaxSize オプションの指定を変更します。
- Explicit ヒープの最大サイズに達していない状態であふれた場合は，OS からのメモリ確保可能サイズを増やす。
  アプリケーションサーバが利用できるメモリサイズを増やしてください。
- Explicit ヒープを大量に消費している原因を取り除く。

ここでは，Explicit ヒープあふれが発生しているかどうかの確認方法について説明します。

(a) 明示管理ヒープ機能のイベントログの調査

明示管理ヒープ機能のイベントログで調査をするためには，あらかじめ JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「normal」を指定しておく必要があります。これによって，ガーベージコレクションが発生するごとに，明示管理ヒープ機能のイベントログに Explicit メモリブロックの利用状況が出力されるようになります。

出力例を示します。

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12672K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
```

太字で示した部分が，Explicit メモリブロックの数を示しています。「E」および「D」は，Explicit メモリブロックのサブ状態である「Enable」および「Disable」を表します。「Disable」の Explicit メモリブロックがある場合は，Explicit ヒープあふれが発生しています。この例の場合は，「Enable」の Explicit メモリブロックが 200 個あり，「Disable」の Explicit メモリブロックはないことがわかります。なお，「Disable」の Explicit メモリブロックがある場合は，Explicit ヒープ最大サイズとの関係を確認してください。Explicit ヒープ最大サイズまでに余裕があるときには，OS からのメモリ確保に失敗していることが考えられます。

また，JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「verbose」を指定した場合，Explicit メモリブロックのサブ状態が「Disable」になった要因も出力されます。

出力例を示します。

```
[EVO]<Tue Jul 24 01:23:51 2007>[alloc failed(Disable)][EH: 32760K(0K)/32768K/
65536K][E/F/D: 321/0/1][cause:GC]¥
["BasicExplicitMemory-3" eid=3/B: 128K(0K)/128K][Thread: 0x00035a60]
[EVO][Thread: 0x00035a60] at ExplicitMemory.newInstance0(Native Method)
[EVO][Thread: 0x00035a60] at BasicExplicitMemory.newInstance(Unknown Source)
[EVO][Thread: 0x00035a60] at AllocTest.test(AllocTest.java:64)
[EVO][Thread: 0x00035a60] at java.lang.Thread.run(Thread.java:2312)
```

この例は，Explicit ヒープあふれが発生した場合の例です。

太字で示した部分のうち，[alloc failed(Disable)] が，Explicit メモリブロックのサブ状態が「Disable」になった要因を示します。["BasicExplicitMemory-3" eid=3/B: 128K(0K)/128K] は，「Disable」になった Explicit メモリブロックの情報を示します。また，[EVO][Thread: 0x00035a60] で始まる行は，イベントが発生した時のスタックトレースを表しています。ただし，ガーベージコレクションによるオブジェクトの移動で Explicit ヒープあふれが発生した場合，スタックトレースは出力されません。

(b) スレッドダンプで出力されたログファイルからの調査

cjdumpsv コマンドなどを使用してスレッドダンプを出力することによって，各 Explicit メモリブロックのサブ状態を出力できます。

出力例を次に示します。

```
Explicit Heap Status
--------------------
 max 65536K, total 21888K, used 20992K, garbage 1288K (32.0% used/max, 95.9%
used/total, 6.1% garbage/used), 2 spaces exist

 Explicit Memories(0x12345678)

  "EJBMgrData" eid=1(0x02f25610)/R, total 21376K, used 20480K, garbage 1234K
(95.8% used/total, 6.0% garbage/used, 0 blocks) Enable

  "ExplicitMemory-4" eid=4(0x02f45800)/B, total 512K, used 512K, garbage 54K
(100.0% used/total, 10.5% garbage/used, 0 blocks) Disable
```

太字で示した部分が，それぞれの Explicit メモリブロックのサブ状態を表しています。

(c) Java の API からの調査

次に示すメソッドを使用して，Explicit メモリブロックのサブ状態を調査できます。

- JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.isActive()

- JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.isReclaimed()

これらのメソッド両方の戻り値が false の場合，その Explicit メモリブロックのサブ状態は Disable と判断できます。

（4） Explicit メモリブロックの初期化が失敗した場合の確認と対処

Explicit メモリブロックの初期化が失敗した場合の確認と対処について説明します。

Explicit メモリブロックの数が最大になると，それ以上 Explicit メモリブロックを初期化できなくなります。

この場合は，Explicit メモリブロックの数を減らしてください。

ここでは，Explicit メモリブロックの初期化が失敗しているかどうかの確認方法について説明します。

（a） 明示管理ヒープ機能イベントログからの調査

明示管理ヒープ機能イベントログで調査をするためには，あらかじめ JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「normal」を指定しておく必要があります。これによって，ガーベージコレクションが発生するごとに，Explicit メモリブロックの初期化に失敗した回数が明示管理ヒープ機能イベントログに出力されるようになります

出力例を示します。

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12672K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
```

太字で示した部分が，前回の出力から今回の出力までの間に Explicit メモリブロックの初期化に失敗した回数です。この例の場合は，「0」です。初期化失敗が発生していない，問題のない状態です。

また，JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「verbose」を指定した場合，Explicit メモリブロックの初期化失敗イベントについての情報も出力されます。

出力例を示します。

```
[EVO]<Tue Jul 24 01:23:51 2007>[Creation failed][EH: 32760K(0K)/32768K/
65536K][E/F/D: 65535/0/0][Thread: 0x00035a60]
[EVO][Thread: 0x00035a60] at ExplicitMemory.registerExplicitMemory(Native
Method)
[EVO][Thread: 0x00035a60] at BasicExplicitMemory.<init>(Unknown Source)
[EVO][Thread: 0x00035a60] at AllocTest.test(AllocTest.java:64)
[EVO][Thread: 0x00035a60] at java.lang.Thread.run(Thread.java:2312)
```

太字で示した部分で，Explicit メモリブロック初期化に失敗したことが確認できます。また，[EVO][Thread: 0x00035a60] で始まる行は，イベントが発生した時のスタックトレースを示しています。

さらに，JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「debug」を指定した場合，初期化に失敗したイベント以外の Explicit メモリブロック初期化イベントの詳細情報が出力されます。Explicit メモリブロック数が一定以上になると，初期化は失敗します。このため，初期化に失敗する前の初期化イベントの情報が，調査に役立つことがあります。

出力例を次に示します。

```
[EVO]<Tue Jul 24 01:23:51 2007>[Created]["BasicExplicitMemory-2"
eid=2(0x1234568)/B][Thread: 0x00035a60]
[EDO][Thread: 0x00035a60] at ExplicitMemory.registerExplicitMemory(Native
Method)
[EDO][Thread: 0x00035a60] at BasicExplicitMemory.<init>(Unknown Source)
[EDO][Thread: 0x00035a60] at AllocTest.test(AllocTest.java:64)
[EVO][Thread: 0x00035a60] at java.lang.Thread.run(Thread.java:2312)
```

太字で示した部分で，Explicit メモリブロック初期化イベントであることが確認できます。また，[EDO][Thread: 0x00035a60] で始まる行は，イベントが発生した時のスタックトレースを示しています。

(b) スレッドダンプで出力されたログファイルからの調査

スレッドダンプで出力された情報からは，Explicit メモリブロック初期化失敗の直接要因は確認できませんが，Explicit メモリブロックの個数は調べられます。

出力例を次に示します。

```
Explicit Heap Status
--------------------
 max 65536K, total 21888K, used 20992K, garbage 1288K (32.0% used/max, 95.9%
used/total, 6.1% garbage/used), 2 spaces exist

 Explicit Memories(0x12345678)

  "EJBMgrData" eid=1(0x02f25610)/R, total 21376K, used 20480K, garbage 1234K
(95.8% used/total, 6.0% garbage/used, 0 blocks) Enable

  "ExplicitMemory-4" eid=4(0x02f45800)/B, total 512K, used 512K, garbage 54K
(100.0% used/total, 10.5% garbage/used, 0 blocks) Disable
```

太字で示した部分が Explicit メモリブロックの個数を示しています。

(c) Java の API からの調査

次に示すメソッドを使用して，Explicit メモリブロックの個数を調査できます。

- JP.co.Hitachi.soft.jvm.ExplicitMemory.countExplicitMemories()

ただし，この API では，Explicit メモリブロック初期化失敗の直接の要因は確認できません。

(5) Explicit メモリブロック解放処理時に Java ヒープへのオブジェクト移動が発生した場合の確認と対処

Explicit メモリブロックの解放処理時に，解放対象の Explicit ヒープ内のオブジェクトに対する参照があると，参照されているオブジェクトおよびそのオブジェクトから直接または間接的に参照されているオブジェクトが Java ヒープに移動します。オブジェクトは，Tenured 領域に優先的に移動されます。このため，移動が多いと Tenured 領域の利用済みサイズが増加して，フルガーベージコレクション発生の要因となってしまいます。

Java ヒープへの移動が発生したかどうかは，日立 JavaVM ログファイルの拡張 verbosegc 情報または明示管理ヒープイベントログで調査できます。

(a) 拡張 verbosegc 情報を使用した確認

明示管理ヒープ機能を利用しない場合，Tenured 領域の利用済みサイズの増加は，コピーガーベージコレクションだけで発生します。このため，N 回目のコピーガーベージコレクション終了後の Tenured 領域利用済みサイズは，N+1 回目のコピーガーベージコレクション開始前の Tenured 領域利用済みサイズと一致します。

これに対して，Explicit ヒープから Java ヒープへのオブジェクトの移動が発生した場合は，Explicit ヒープ解放時に Tenured 領域の利用済みサイズが増加します。この差分から，Explicit メモリブロック解放処理時にオブジェクトの移動が発生したことが確認できます。

N 回目のコピーガーベージコレクション終了後の Explicit メモリブロック解放処理時に Java ヒープに移動したオブジェクトのサイズは，次の式で算出できます。

```
ExplicitヒープからJavaヒープに移動したオブジェクトのサイズ
=N+1回目のCopy GC前のTenured領域利用済みサイズ
 -N回目のCopy GC後のTenured領域利用済みサイズ
```

(b) 明示管理ヒープのイベントログを使用した確認

JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「none」以外を指定した場合，Explicit メモリブロックの解放処理についてのログが出力されます。このログでは，Explicit メモリブロック解放処理時の Tenured 領域利用済みサイズの増加を直接確認できます。

出力例を次に示します。

```
[ENS]<Tue Jul 24 01:23:51 2007>[EH: 12800K->11776K(11776K/65536K), 0.1129602
secs][E/F/D: 523/0/0]¥
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured:
103400K->103464K(556800K)][cause:Reclaim]
```

太字で示した部分のうち，[cause:Reclaim] は，Explicit メモリブロック解放処理時に出力された情報であることを示します。また，[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured: 103400K->103464K(556800K)] の部分は，Explicit メモリブロック解放処理時の Java ヒープの変化を示しています。この例の場合は，Tenured 領域のメモリサイズが 103,400 キロバイトから 103,464 キロバイトに，64 キロバイト分増えています。このことから，Explicit メモリブロックの解放処理で，64 キロバイトのオブジェクトが Java ヒープに移動していることがわかります。

また，JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「verbose」を指定した場合，解放された Explicit メモリブロックについての情報も出力されます。これによって，どの Explicit メモリブロックの解放で Tenured 領域利用済みサイズが増加したかを確認できます。

出力例を次に示します。

```
[ENS]<Tue Jul 24 01:23:51 2007>[EH: 12800K->11776K(11776K/65536K), 0.1129602
secs][E/F/D: 523/0/0]¥
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured:
103400K->103464K(556800K)][cause:Reclaim]
[EVS]["REM2" eid=2/R: 320K]["BEM3" eid=5/B: 320K]["BEM1" eid=7/B: 384K]
```

太字で示した部分が，解放された Explicit メモリブロックを示しています。出力内容から，Java ヒープに移動した 64 キロバイトのオブジェクトが，「REM2」「BEM3」「BEM1」のどれかの Explicit メモリブロックから移動したことがわかります。

さらに，JavaVM の -XX:HitachiExplicitMemoryLogLevel オプションに「debug」を指定した場合，解放処理をしたときに解放対象 Explicit ヒープ内のオブジェクトを参照していたオブジェクトが確認できます。

出力例を次に示します。

```
[EDO][eid=3: Reference to ClassZ(0x1234680), total 64K]
[EDO]    ClassU(0x1233468)(Tenured)
```

[eid=3: Reference to ClassZ(0x1234680), total 64K] の部分から，次のことがわかります。

- Java ヒープへ移動したオブジェクトは "ClassZ" のインスタンスである。

- "ClassZ" のインスタンスから参照されていることによって Java ヒープに移動したオブジェクトの合計サイズは 64 キロバイトである。

また，「ClassU(0x1233468)(Tenured)」の部分から，"ClassZ" のインスタンスを参照しているオブジェクトが "ClassU" のインスタンスであることがわかります。

これらの情報を基に，Explicit メモリブロック解放処理時にその Explicit メモリブロック内のオブジェクトへの参照をなくすように，Java プログラムを修正してください。

7.12 明示管理ヒープの自動配置機能を使用した Explicit ヒープの利用の検討

ここでは，明示管理ヒープの自動配置機能を使用した Explicit ヒープの利用の検討について説明します。

明示管理ヒープ機能を使用して Explicit ヒープ領域を利用する際，自動配置機能を使用することで，明示管理ヒープ機能を容易に使用することができます。また，次に示すような場合は，自動配置機能を使用することをお勧めします。

(1) アプリケーション内に Tenured 領域の増加原因のオブジェクトがある場合

アプリケーション内に Tenured 領域の増加原因のオブジェクトがある場合，自動配置設定ファイルを使用してオブジェクトを Explicit ヒープに配置することをお勧めします。オブジェクトの配置を検討した方がよい Java プログラムの例を次に示します。

```
01:package abcd.efg;
02:import java.util.HashMap;
03:// KVStorageのインスタンスは，長期間生存し続ける
04:class KVStorage {
05:  HashMap _map = new HashMap();
06:
07:  public void store(MyKey k,MyData d) {
08:    // ...前処理...
09:    _map.put(k,d);
10:    // ...後処理...
11:  }
12:
13:  public MyData load(MyKey k) {
14:    // ...前処理...
15:    MyData d = map.get(k);
16:    // ...後処理...
17:    return d;
18:  }
19:}
```

この設定例の場合，KVStorage がインスタンスフィールドに保持している HashMap クラスが Tenured 領域のメモリサイズ増加の要因となる長寿命オブジェクトとなります。このオブジェクトの生成先を Explicit ヒープへ変更する場合，次の例のように自動配置設定ファイルを指定します。

```
# 生成個所（メソッド名やクラス名），生成するクラス名 の対で記載。
abcd.efg.KVStorage.<init>, java.util.HashMap
```

この例のように自動配置設定ファイルを指定することで，Java プログラムの例の 5 行目の HashMap インスタンス（_map）の生成先が Java ヒープから Explicit ヒープに変更されます。また，store メソッドで _map に格納した MyKey のインスタンス，および MyData のインスタンスも，順次 Explicit ヒープに移動されます。これらのインスタン

スは，不要となった時点で JavaVM によって自動的に解放されます。

(2) 特定のフレームワークを使用している場合

アプリケーションサーバと Hibernate の両方を動作させた場合，Tenured 領域の増加の要因となるオブジェクトがあります。アプリケーションサーバでは，これらオブジェクト向けの明示管理ヒープ自動配置設定を内部に保持しています。この設定は，Explicit メモリブロックの自動配置機能を有効にする（-XX:+HitachiAutoExplicitMemory オプションを指定）だけで有効となります。

Hibernate へのクエリ結果を長期間保持するようなアプリケーションの場合，次に示す内容を自動配置設定ファイルに指定することで，Tenured 領域の増加を抑止できることがあります。

```
org.hibernate.impl.AbstractSessionImpl.createQuery(java.lang.String),
org.hibernate.impl.QueryImpl

org.hibernate.engine.query.HQLQueryPlan.performList(org.hibernate.engine.Query
Parameters, org.hibernate.engine.SessionImplementor), java.util.ArrayList

org.hibernate.engine.query.HQLQueryPlan.performList(org.hibernate.engine.Query
Parameters, org.hibernate.engine.SessionImplementor),
org.hibernate.util.IdentitySet
```

なお，アプリケーションサーバと Hibernate との接続確認には，Hibernate Core 3.2.6 GA を使用しています。また，Explicit メモリブロックの自動配置機能によって，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おそれがあります。

(3) Tenured 領域利用済みサイズの増加原因が不明な場合

Survivor 領域のチューニングを実施しても，Tenured 領域利用済みサイズが増加し，それによるフルガーベージコレクションの発生間隔がシステムの要件を満たせない場合，Tenured 領域利用済みサイズの増加原因となるオブジェクトを Explicit ヒープへ生成します。Explicit ヒープへオブジェクトを生成するには，明示管理ヒープの自動配置設定ファイルを設定します。

Tenured 領域利用済みサイズの増加原因となるオブジェクトを調査する方法，および自動配置設定ファイルの設定方法について説明します。

(a) Tenured 領域利用済みサイズ増加の調査

実行中のアプリケーションに対して，jheapprof コマンドに -garbage オプションを指定し，Tenured 領域内不要オブジェクト統計機能を実行します。

なお，08-70 よりも前のバージョンを使用している場合，Explicit メモリブロックの自動配置機能（-XX:+HitachiAutoExplicitMemory）と同時に -garbage オプションを利用するときには，あらかじめ -XX:-HitachiExplicitMemoryPartialTenuredAreaCollection オプションを指定した状態で，アプリケーションサーバを起動してください。

Tenured 領域内不要オブジェクト統計機能の出力例を次に示します。これによって，Tenured 領域利用済みサイズ増加の原因となっているオブジェクト（Tenured 増加要因の基点オブジェクト）のクラス名のリストがスレッドダンプログファイルに出力されます。

```
Garbage Profile Root Object Information
-----------------------------------
*, java.util.HashMap # 35234568
*, java.util.WeakHashMap # 4321000
```

この出力例では，Tenured 領域利用済みサイズの増加原因として，java.util.HashMap のオブジェクトが 35,234,568 バイト，また java.util.WeakHashMap のオブジェクトが 4,321,000 バイトであることがわかります。Tenured 領域内不要オブジェクト統計機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「8.8　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。

(b) 自動配置設定ファイルへの記載

Tenured 領域内不要オブジェクト統計機能の出力例のリスト部分（後半 2 行）を，自動配置設定ファイルへ入力します。自動配置設定ファイルの設定例を示します。

```
*, java.util.HashMap # 35234568
*, java.util.WeakHashMap # 4321000
```

この場合，プログラム中のすべての java.util.HashMap オブジェクト，および java.util.WeakHashMap オブジェクトは Explicit ヒープに生成されます。

また，これらのオブジェクトに格納したオブジェクトも順次 Explicit ヒープに移動します。しかし，Explicit ヒープへのオブジェクトの生成は，Java ヒープへのオブジェクトの生成よりも実行時にオーバーヘッドが掛かります。このため，オブジェクトの生成個所を絞り込むことで，実行時のオーバーヘッドを削減できます。

自動配置設定ファイルでは，「*」は「JavaVM 上で動作するすべてのクラス」を意味します。この設定例の場合，すべてのクラスでの java.util.HashMap および java.util.WeakHashMap のオブジェクトの生成先が Explicit ヒープになります。これによって，実際には Tenured 利用済みサイズ増加の原因ではないオブジェクトの生成先も Explicit ヒープとなり，オーバーヘッドが増加するおそれがあります。

「*」を指定したことによって，アプリケーションのスループットが要件を満たせなくなった場合は，Tenured 領域利用済みサイズ増加の原因となっているオブジェクトの生成個所を絞り込むことを検討してください。

システム運用者とアプリケーション開発者が異なる場合は，アプリケーション開発者への調査の依頼が必要です。アプリケーションの詳細な調査が困難な場合でも，自動配置設定ファイルでは生成個所を「すべてのクラス」，「特定のパッケージ」，「特定のクラ

ス」，および「特定のメソッド」の 4 段階の粒度でオブジェクトの生成個所を指定できます。そのため，調査可能な範囲で絞り込みを実施して，自動配置設定ファイルを指定することによって，スループットが向上する場合もあります。

例えば，生成個所が「com.abc.defg」パッケージ下の場合，自動配置設定ファイルの設定例を次のように変更することによって，「すべてのパッケージのすべてのクラス」から，「com.abc.defg パッケージおよびサブパッケージのすべてのクラス」まで絞り込みができます。

```
com.abc.defg.*, java.util.HashMap # 35234568
com.abc.defg.*, java.util.WeakHashMap # 4321000
```

自動配置設定ファイルの指定方法の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「8.11.2　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。

(c) Tenured 領域内不要オブジェクト統計機能によるアプリケーションの調査

自動配置設定ファイルの内容を基に，アプリケーションを調査する場合に，Tenured 領域内不要オブジェクト統計機能を利用します。

jheapprof コマンドに -garbage オプションを指定し，Tenured 領域内不要オブジェクト統計機能を実行することで，Tenured 増加要因の基点オブジェクトリスト，および Tenured 領域内不要オブジェクトの統計情報を拡張スレッドダンプに出力します。拡張スレッドダンプの出力例を次に示します。

```
Garbage Profile
---------------
_________________Size__Instances__Class_________________
             35234568      10648 java.util.HashMap
              5678900      10668 [Ljava.util.HashMap$Entry;
              4456788       7436 java.util.HashMap$Entry
              4321000        200 java.util.WeakHashMap
              1234568        190 [Ljava.util.WeakHashMap$Entry
1456788        9524 java.lang.String
1256788        6424 com.abc.defg.MyData;
 :
```

Tenured 領域内不要オブジェクト統計機能では，Tenured 増加要因の基点オブジェクトリストには出力されない，java.util.HashMap や java.util.WeakHashMap に格納されているオブジェクトも出力されます。また，各オブジェクトのインスタンス数も出力されます。

さらに，このログを複数回取得して，クラス別統計情報解析機能（jheapprofanalyzer コマンド）の入力ファイルとすることで，各オブジェクトサイズ，およびインスタンス数の時間の変化を調査することができます。

これらの情報を基に，アプリケーションを調査してオブジェクト生成個所の絞り込みをします。Tenured 領域内不要オブジェクト統計機能の詳細については，マニュアル

「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「8.8 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。クラス別統計情報解析機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「8.10　クラス別統計情報解析機能」を参照してください。

8 パフォーマンスチューニング（J2EE アプリケーション実行基盤）

この章では，J2EE アプリケーションを実行するシステムのパフォーマンスをチューニングする方法について説明します。パフォーマンスチューニングによって動作環境を最適化することで，システムの性能を最大限に生かせるようになります。バッチアプリケーション実行基盤のパフォーマンスチューニングについて検討する場合は，「9. パフォーマンスチューニング（バッチアプリケーション実行基盤）」を参照してください。

8.1 パフォーマンスチューニングで考慮すること

この節では，J2EE アプリケーション実行基盤のパフォーマンスチューニングで考慮することについて説明します。

8.1.1 パフォーマンスチューニングの観点

J2EE アプリケーション実行基盤のパフォーマンスチューニングは，次の観点で実施します。

- **同時実行数の最適化**
- Enterprise Bean **の呼び出し方法の最適化**
- **データベースアクセス方法の最適化**
- **タイムアウトの設定**
- Web **アプリケーションの動作の最適化**
- CTM **の動作の最適化**
- **そのほかの項目のチューニング**

それぞれのポイントについて説明します。

（1） 同時実行数の最適化

同時実行数の最適化は，処理を多重化して CPU の処理能力を最大限に引き出して，システムのスループットを向上させることを目的とします。しかし，次のような場合，多重化しただけではスループットが向上しません。場合によっては，スループットが低下するおそれがあります。

- 入出力処理，排他処理などのボトルネックがある場合
- 最大スループットに到達している場合
- CPU の利用率が飽和した状態で多重度以上の負荷を掛けた場合
- 実行待ちキューのサイズが不適切な場合
- 階層的な最大実行数の設定が不適切な場合

パフォーマンスチューニングでは，これらを考慮しながら適切なチューニングを実施して，同時実行数の最適化を図ります。

（2） Enterprise Bean の呼び出し方法の最適化

Enterprise Bean の呼び出し方法の最適化は，同じ J2EE アプリケーションや同じ J2EE サーバ内のコンポーネントを呼び出すときに，ローカルインタフェースやリモートイン

タフェースのローカル呼び出し機能を利用することで，不要なネットワークアクセスを削減することを目的とします。

次の機能を利用することで，RMI-IIOP 通信によって発生する不要なネットワークアクセスを削減できます。

- ローカルインタフェースの利用
- リモートインタフェースのローカル呼び出し機能の利用

また，引数や戻り値の渡し方を参照渡しにすることで，さらに処理性能を向上できる場合があります。パフォーマンスチューニングでは，アプリケーションやシステムの特徴によってこれらの機能を有効に活用して，処理性能の向上を図ります。

(3) データベースアクセス方法の最適化

データベースアクセス方法の最適化は，処理に時間が掛かるコネクションやステートメントを事前に生成しておくことで，データベースアクセス時のオーバーヘッドを削減することを目的とします。

パフォーマンスチューニングでは，次に示す機能を有効に活用することで，データベースアクセス処理を最適化し，スループットを向上させます。

- コネクションプーリング
- ステートメントプーリング（PreparedStatement および CallableStatement のプーリング）

(4) タイムアウトの設定

タイムアウトの設定は，システムのトラブル発生を検知して，リクエストの応答が返らなくなることを防ぎ，適宜リソースを解放することを目的とします。

設定できるタイムアウトには，次の種類があります。

- Web フロントシステムのタイムアウト
- バックシステムのタイムアウト
- トランザクションのタイムアウト
- データベースのタイムアウト

(5) Web アプリケーションの動作の最適化

Web アプリケーションの動作の最適化は，コンテンツの配置方法の検討やキャッシュの利用によって不要なネットワークアクセスを削減して処理速度を速めたり，負荷分散によってシステムのスループットの向上を図ったりすることを目的とします。

なお，Web サーバとして，リダイレクタモジュールを組み込んだ Web サーバと連携する場合と，インプロセス HTTP サーバを使用している場合で，チューニングできる項目が異なります。

Web サーバと連携する場合には，次の処理ができます。

● 静的コンテンツと Web アプリケーションでの処理の振り分け

● 静的コンテンツのキャッシュ

● セッション情報に応じたリクエストの振り分け

インプロセス HTTP サーバを使用する場合は，次の処理ができます。

● 静的コンテンツと Web アプリケーションの配置の切り分け

● 静的コンテンツのキャッシュ

（6） CTM の動作の最適化

CTM の動作の最適化は，CTM で使用するプロセス間の通信間隔を最適化して通信負荷を軽減したり，トラブル発生時に迅速に検知して対処したりすることで，システムとしての性能を向上させることを目的とします。また，CTM によってリクエストの処理に優先順位を付けることで，重要なリクエストをすばやく処理するようにもチューニングできます。

（7） そのほかの項目のチューニング

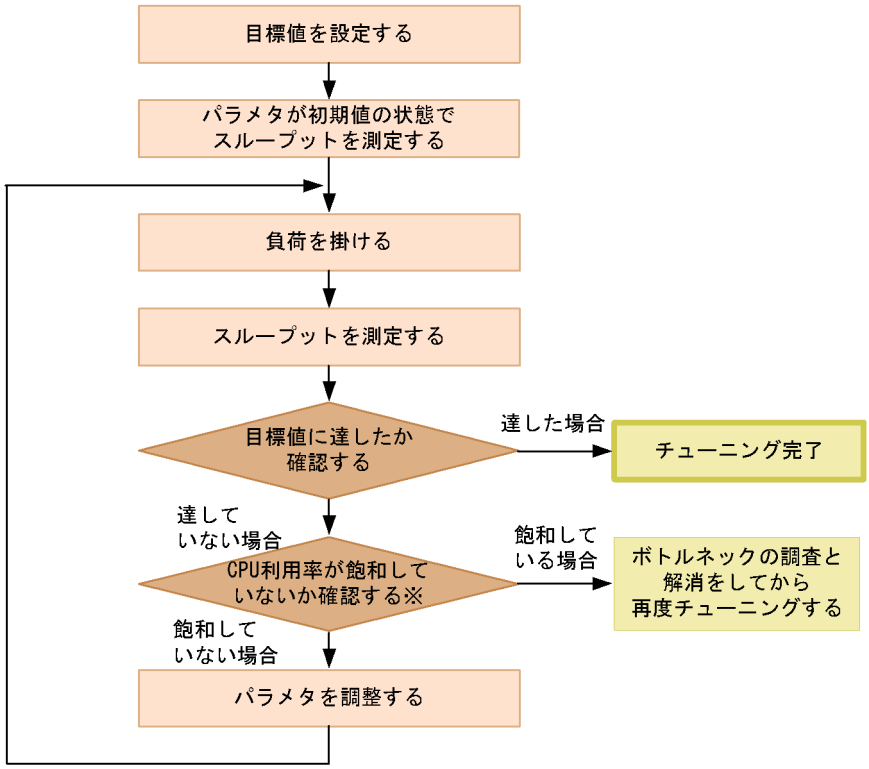
アプリケーションサーバでは，(1) ～ (6) で説明した項目以外にも，チューニングできる項目があります。必要に応じてチューニングを実施してください。

8.1.2 チューニング手順

パフォーマンスチューニングは，システムのパフォーマンスを生かす最適な設定を見つける作業です。構築した環境で，実際に処理を実行したり，模擬的な負荷を掛けたりしながら，パラメタの調整やボトルネックの調査，解消によってパフォーマンスを向上させていきます。

パフォーマンスチューニングの手順の例として，ここでは，同時実行数のチューニング手順を示します。

図 8-1　パフォーマンスチューニングの手順（同時実行数をチューニングする場合）



注※　パラメタを変更してもスループットが向上しない場合，飽和しています。

チューニング作業では，まず，目標値を決定します。ここでは，CPU 利用率などが該当します。

次に，各パラメタに初期値を設定した状態でのスループットを測定し，そのあとで模擬的な負荷を掛けながら各パラメタを調整して，目標値に近い最適な値を見つけていきます。模擬的な負荷は，専用のツールを使用して発生させます。

チューニングの際，CPU の利用率の測定には，OS に付属している監視ツールなどが利用できます。スループットの測定は，負荷発生ツールなどによって測定できます。また，稼働スレッド数など，アプリケーションサーバの稼働情報については，稼働情報収集機能などで確認できます。確認方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「3.　稼働情報の監視（稼働情報収集機能）」を参照してください。

スループットが目標値に達したところで，パフォーマンスチューニングは完了です。なお，CPU 利用率が 100% からかなり低い状態で飽和した場合は，システム上に入出力処理や排他処理などのボトルネックがあるおそれがあります。ボトルネックを調査し，対策してから，再度パフォーマンスチューニングを実行してください。アプリケーションサーバのシステムのボトルネックの調査には，性能解析トレースを利用できます。性能

解析トレースの機能詳細，および性能解析トレースを利用して取得したトレースファイルの利用方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「6. 性能解析トレースを使用したシステムの性能解析」を参照してください。

8.1.3 アプリケーションの種類ごとにチューニングできる項目

チューニング項目は，アプリケーションの種類によって異なります。アプリケーションに含まれるコンポーネントごとのチューニング項目について，次に示します。

表 8-1 サーブレットと JSP で構成されるアプリケーション（Web アプリケーション）のチューニング項目

チューニング項目	利用できる機能	参照先
リクエスト処理スレッド数の最適化（インプロセス HTTP サーバを使用する場合）	リクエスト処理スレッド数の制御（インプロセス HTTP サーバを使用する場合）※	8.3.3
同時実行数の最適化	Web アプリケーションでの同時実行スレッド数制御（Web コンテナ単位，Web アプリケーション単位，または URL グループ単位）	8.3.4
データベースアクセス方法の最適化	コネクションプーリング	8.5.1
	ステートメントプーリング	8.5.2
タイムアウトの設定	Web フロントシステムでのタイムアウトの設定	8.6.2
	J2EE アプリケーションのメソッド実行時間に対するタイムアウトの設定	8.6.6
Web アプリケーションの動作の最適化	静的コンテンツと Web アプリケーションの配置の切り分け	8.7.1
	静的コンテンツのキャッシュ	8.7.2
	リダイレクタによるリクエストの振り分け（Web サーバ連携の場合）	8.7.3
そのほかの項目のチューニング	Persistent Connection の制御（インプロセス HTTP サーバを使用する場合）	8.9

注※ Web サーバと連携する場合は，Web サーバの機能を使用してチューニングしてください。

表 8-2 Enterprise Bean で構成されるアプリケーションのチューニング項目

チューニング項目	利用できる機能	参照先
同時実行数の最適化	Stateless Session Bean のインスタンスプーリング	8.3.5
	Stateful Session Bean のセッション制御	

チューニング項目	利用できる機能	参照先
	Message-driven Bean のインスタンスプーリング	
	CTM による同時実行数制御※（CTM を使用している場合）	8.3.6
Enterprise Bean の呼び出し方法の最適化	ローカルインタフェースの使用	8.4.1
	リモートインタフェースのローカル呼び出し最適化	8.4.2
	リモートインタフェースの参照渡し	8.4.3
データベースアクセス方法の最適化	コネクションプーリング	8.5.1
	ステートメントプーリング	8.5.2
タイムアウトの設定	バックシステムでのタイムアウトの設定	8.6.3
	トランザクションタイムアウトの設定	8.6.4
	データベースでのタイムアウトの設定	8.6.5
	J2EE アプリケーションのメソッド実行時間に対するタイムアウトの設定	8.6.6

注※　Stateless Session Bean だけが対象です。

また，CTM を使用したシステムの場合に設定できる，CTM の動作のチューニング項目を次の表に示します。CTM は，アプリケーションが Stateless Session Bean で構成されている場合に使用できます。

表 8-3　CTM の動作についてのチューニング項目

チューニング項目	利用できる機能	参照先
CTM の動作の最適化	CTM ドメインマネジャおよび CTM デーモンの稼働状態を監視する間隔のチューニング	8.8.1
	負荷状況監視間隔のチューニング	8.8.2
	CTM デーモンのタイムアウト閉塞の設定	8.8.3
	CTM で振り分けるリクエストの優先順位の設定	8.8.4

8.2 チューニングの方法

この節では，チューニングの方法について説明します。チューニングの方法は，設定対象の種類によって異なります。

（1）J2EE サーバおよび Web サーバ（リダイレクタを含む）のチューニング

J2EE サーバおよび Web サーバ（リダイレクタを含む）のチューニングには，Smart Composer 機能の簡易構築定義ファイルを使用します。簡易構築定義ファイルでは，<configuration> タグ下の <logical-server-type> に設定対象とする論理サーバの種類（J2EE サーバまたは Web サーバ）を指定して，<param> タグ下でパラメタ名とその値を設定します。簡易構築定義ファイルの詳細については，マニュアル「Cosminexus 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「4.6　簡易構築定義ファイル」を参照してください。

参考

Web Redirector を使用する場合など，Smart Composer 機能を使用できない場合，Web サーバ（リダイレクタを含む）のチューニングはファイルを編集して定義します。
Smart Composer 機能を使用できない場合に，Web サーバ（リダイレクタを含む）のチューニングに使用するファイルについて，次の表に示します。

表 8-4　Smart Composer 機能を使用できない場合に Web サーバ（リダイレクタを含む）のチューニングに使用するファイル

対象	チューニング方法
Web サーバ	httpsd.conf の編集
Web サーバ（リダイレクタ）	mod_jk.conf（Hitachi Web Server の場合）の編集
	isapi_redirect.conf（Microsoft IIS の場合）の編集
	workers.properties（ワーカの設定の場合）の編集

mod_jk.conf の詳細については，マニュアル「Cosminexus 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「9.3　mod_jk.conf（Hitachi Web Server 用リダイレクタ動作定義ファイル）」を参照してください。isapi_redirect.conf の詳細については，マニュアル「Cosminexus 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「9.2 isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）」を参照してください。workers.properties の詳細については，マニュアル「Cosminexus 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「9.5　workers.properties（ワーカ定義ファイル）」を参照してください。httpsd.conf の詳細については，マニュアル「Hitachi Web Server」を参照してください。

(2) アプリケーションまたはリソースのチューニング

アプリケーションおよびリソースのチューニングをする場合は，サーバ管理コマンドを使用します。

サーバ管理コマンドを使用する場合は，属性ファイルを編集します。属性ファイルの詳細については，マニュアル「Cosminexus 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。

(3) CTM の動作のチューニング

CTM のチューニングには，Smart Composer 機能の簡易構築定義ファイルを使用します。簡易構築定義ファイルでは，<configuration> タグ下の <logical-server-type> に設定対象とする論理サーバの種類（CTM ドメインマネジャまたは CTM）を指定して，<param> タグ下でパラメタ名とその値を設定します。

簡易構築定義ファイルの詳細については，マニュアル「Cosminexus 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「4.6　簡易構築定義ファイル」を参照してください。

(4) それ以外の項目のチューニング

このほか，チューニングで使用するパラメタには，API で設定する項目や，データベースで設定する項目などがあります。これらの設定については，それぞれの節のチューニングパラメタの説明を参照してください。

8.3 同時実行数を最適化する

この節では，アプリケーションのリクエストの同時実行数を最適化するための考え方とチューニング方法について説明します。

8.3.1 同時実行数制御および実行待ちキュー制御の考え方

アプリケーションサーバのシステムでアプリケーションのスループットを向上させるためには，複数のリクエストを多重処理することが有効です。複数のスレッドで多重にリクエストを処理した方が，単一スレッドで一度に 1 リクエストずつ処理するのに比べて，多くの場合，スループットを向上させられます。

ただし，入出力処理，排他処理などにボトルネックがあったり，すでに最大スループットに到達していたりする場合は，多重化によるスループットの向上はできません。このため，同時実行数のチューニングは，次の点を確認しながら進めます。

入出力処理，排他処理などのボトルネックの排除

スレッドを多重化しても CPU 利用率が少ないままでスループットが向上しない場合は，アプリケーションのデータベースアクセスなどの入出力処理や排他処理などにボトルネックがあるおそれがあります。この場合は，ボトルネックになっている処理を特定して，排除してからチューニングする必要があります。例えば，データベースアクセス方法をチューニングしたり，排他処理方法を変更したりして，入出力処理，排他処理などのボトルネックを取り除きます。

最大スループットの確認

リクエストの多重度を大きくしてスレッド数を増やしていくと，スレッド数の増加に伴って CPU の空き時間が減少し，さらに増やしていくと，ほとんど CPU の空き時間がない状態になります。この状態になると，スレッドを増やしてもスループットは上がりません。

これは，CPU がボトルネックになっている状態であり，マシン単体の性能として，限界に達している状態です。つまり，この時点のスループットが，マシン単体でのアプリケーションの最大スループットになります。

これ以上のスループットを確保したい場合は，CPU やマシン数を増加させるなど，ハードウェアの増強が必要になります。

同時実行数制御によるスループットの維持

CPU の利用率が飽和した状態で，さらに多重度を大きくしてスレッドを増やした場合，実行可能な状態で CPU が割り当てられないスレッドが増えます。この状態では，スレッド間のロックが競合したり，スレッドのコンテキストスイッチが多発したりするので，スループットが低下するおそれがあります。

また，スレッド数を増やすと，アプリケーションサーバ内ではスレッド数に応じてメモリ使用量が増えます。つまり，スループット低下とメモリ使用量の増加を防ぐためには，スレッド数の増加を実行可能なスレッド数までに制限する必要があります。

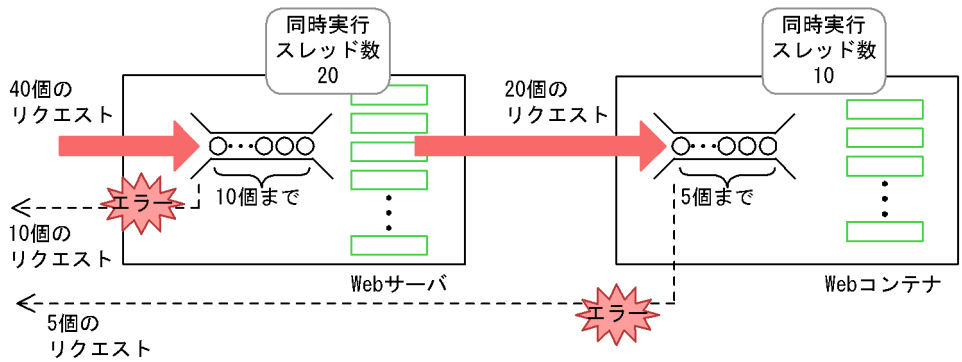
同時実行数制御機能を適切に使用することで，最大同時実行数をチューニングして，リクエストの多重度を大きくした場合でも，最大同時実行数分以上のリクエストの実行を待たせることができます。この結果，一時的に過負荷状態が発生したり，負荷のピークの状態になったりしても，高いスループットを維持できます。

実行待ちキューサイズの調整

アプリケーションサーバに到着したリクエストが同時実行数の上限を超えたとき，リクエストをキューに登録することで，実行中のほかのリクエストの処理が終わるまでそのリクエストを待たせることができます。しかし，ほかのリクエストの処理が終わるまでリクエストを待たせておくためのキューのサイズ（**実行待ちキューサイズ**）に上限を設定している場合，実行待ちキューが上限に達したあとで到着したリクエストについては，実行待ちキューには登録されないでクライアントにエラーとして返却されます。このため，実行待ちキューに上限を設定する場合は，待たせるキューに必要な数を確保しておく必要があります。

実行待ちキューへのリクエストの登録とエラーの返却の考え方を，次の図に示します。

図 8-2　実行待ちキューへのリクエストの登録とエラーの返却



・Webサーバでは，40個のリクエストに対して，20個のスレッドを同時実行して，10個のリクエストを実行待ちキューに登録できます。
実行待ちキューの上限以上の10個のリクエストについては，エラーが返却されます。

・Webコンテナでは，20個のリクエストに対して，10個のスレッドを同時実行して，5個のリクエストを実行待ちキューに登録できます。
実行待ちキューの上限以上の5個のリクエストについては，エラーが返却されます。

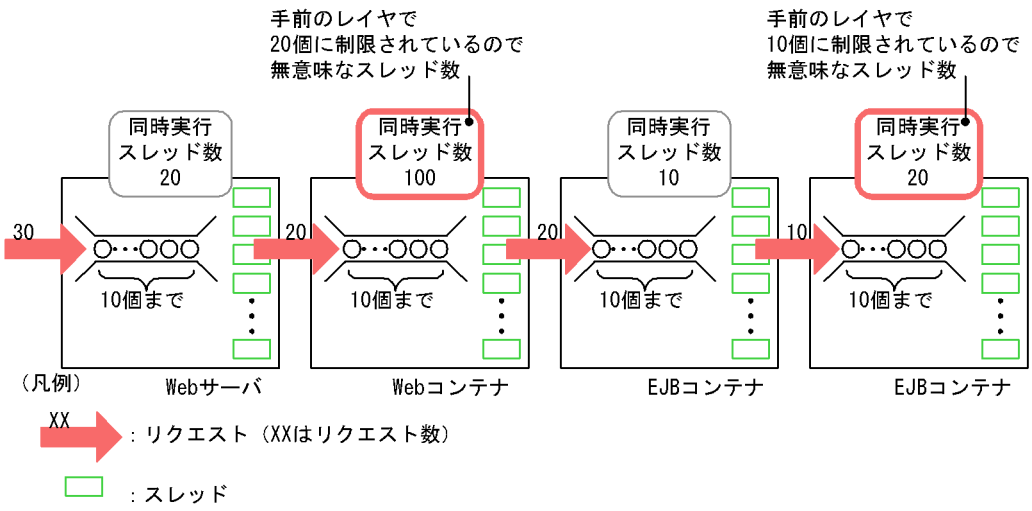
ポイント

リクエストを実行待ちキューで待たせるのは，一時的な過負荷状態の場合や負荷のピークの場合にエラーが発生するのを防ぐためです。エラーが返却されるのを避けるために実行待ちキューサイズをむやみに増やすのは，本質的な解決にはなりません。同時実行数を増やしたり，必要に応じて CPU やマシン数を増設したりするなどの対処をしてください。
また，クライアント側でタイムアウトを設定している場合は，リクエストが実行待ちキューに登録されてから実際に実行されるまでの時間が掛かり過ぎると，リクエストの実行前にタイムアウトが発生してエラーになるおそれがあります。
このため，実行待ちキューサイズは，最適なサイズに設定してください。

階層的なアプリケーションでの最大同時実行数のバランス

あるレイヤだけをチューニングして同時実行数を増やしても，システム全体の性能は向上しません。これは，システム全体としての性能は，性能が低いレイヤに制約されてしまうためです。
特定のレイヤだけを最適化したために無意味な設定を含んでしまった例を，次の図に示します。

図 8-3　特定のレイヤだけを最適化したために無意味な設定を含んでしまった例



また，同時実行数を増やすと，実際にはリクエストを処理していないアイドル状態の場合にも，メモリなどのリソースをむだに消費するおそれがあります。

このため，同時実行数を制御する場合には，システム全体を見渡して，適切な同時実行数をそれぞれのレイヤに設定するようにしてください。

8.3.2　最大同時実行数と実行待ちキューを求める手順

最大同時実行数および実行待ちキューサイズは，次の手順に従ってチューニングできます。

1. **負荷発生ツールなどを利用して，リクエストの多重度を増やします。**
   このとき，サーバ側の CPU 利用率が 80% ～ 90% に達した場合は，手順 2. に進みます。80% ～ 90% に達しない，低い状態でスループットが向上しなくなった場合は，入出力処理や排他処理などにボトルネックがあることが考えられます。この場合は，ボトルネックになっている処理を特定して，性能を改善します。
2. **サーバ側の CPU 利用率が 80% ～ 90% に達した多重度を，最大同時実行数としてチューニングパラメタに設定します。**
   この状態でのスループットが，単体マシンの最大スループットになります。これ以上のスループットを求めたい場合は，ハードウェアの増強が必要です。
3. **負荷発生ツールなどでさらに高い負荷を掛けて，最大スループットが維持できるかどうかを確認します。**
   維持できない場合は，最大スループット以上の負荷が掛からないように，チューニングパラメタを調整します。
4. **実際のシステムでの一時的な過負荷状態，および負荷のピークの状態でのリクエスト数を見積もり，実行待ちキューサイズを決定します。**
5. **階層的な構造を持つアプリケーションでは，各レイヤでの同時実行数および実行待ちキューサイズのバランスが取れるよう，調整します。**

8.3.3 Web サーバでのリクエスト処理スレッド数を制御する

Web フロントシステムの場合，Web ブラウザなどのクライアントからのリクエストは，Web サーバが作成するリクエスト処理スレッドによって処理されます。リクエスト処理スレッド数を適切に制御することで，処理性能の向上が図れます。

ここでは，Web サーバでのリクエスト処理スレッド数を制御する目的と，チューニングの指針について説明します。

なお，ここでは，インプロセス HTTP サーバを使用している場合のチューニングの方法について説明します。

参考

Web サーバ連携時に Hitachi Web Server を使用している場合は，Hitachi Web Server の設定で同様のチューニングができます。詳細は，マニュアル「Hitachi Web Server」を参照してください。
また，Smart Composer 機能を使用してシステムを構築する場合，Web サーバでのリクエスト処理スレッド数の設定は，抽象パラメタの利用によって設定できます。抽象パラメタとは，互いに関連のあるパラメタを一つにまとめたパラメタです。抽象パラメタを利用することで，Web サーバのリクエスト処理スレッド数の設定を，同時実行スレッド数などの関連するパラメタと一緒に定義できます。抽象パラメタについては，マニュアル「Cosminexus アプリケーションサーバ リファレンス コマンド編」の「付録 I　Smart Composer 機能で使用できる抽象パラメタ」を参照してください。

(1) リクエスト処理スレッド数を制御する目的

リクエスト処理スレッド数を J2EE サーバが動作しているホストの性能やクライアントからのアクセス状況に合わせてチューニングすることで，性能向上を図れます。

リクエスト処理スレッドの生成は，負荷が高い処理です。リクエスト処理スレッドをあらかじめ生成してプールしておくことで，Web ブラウザなどのクライアントからのリクエスト処理要求時の負荷を軽くして，処理性能を高めることができます。

インプロセス HTTP サーバを使用する場合，J2EE サーバ起動時にリクエスト処理スレッドをまとめて生成してプールしておき，Web ブラウザなどのクライアントからリクエスト処理要求があった場合にそれを利用するようにできます。これによって，リクエスト処理要求時の処理性能の向上を図れます。なお，プールしているスレッドの数を監視しておくことで，プールしているスレッド数が少なくなった場合はさらに追加生成して，プールに確保しておくこともできます。

ただし，使用しないスレッドを大量にプールしておくと，むだなリソースを消費します。このため，システムの処理内容に応じて，プールするリクエスト処理スレッド数を適切に制御し，場合によっては不要なスレッドを削除することが必要です。

リクエスト処理スレッドの制御では，これらを考慮して，パラメタに適切な値を設定してください。

(2) 設定の指針

リクエスト処理スレッド数の制御では，次のパラメタを使用してチューニングできます。

- J2EE サーバ起動時に生成するリクエスト処理スレッドの数
- Web クライアントとの接続数（リクエスト処理スレッド数）の上限数
- 接続数の上限を超えた場合の TCP/IP の Listen キュー（バックログ）の最大値
- 予備スレッド数の最大数および最小数
- J2EE サーバ起動時に生成したリクエスト処理スレッド数を維持するかどうかの選択

これらのパラメタを設定するときには，次の点に留意してください。

- 提供するサービスの内容によっては，J2EE サーバ起動直後から大量のリクエストを処理する必要があります。この場合は，J2EE サーバ起動時に生成するリクエスト処理スレッドの数に，大きな値を指定してください。
- 予備スレッド数の最大数を大きくしておくと，クライアントからのアクセスが急に増加した場合にも，処理性能を下げることなく迅速に対応できます。ただし，多くの予備スレッドをプールしておくと，多くのリソースが消費されます。このため，急な増加が予想されるアクセス数を見積もって，適切な数の予備スレッドがプールされるように，注意して設定してください。
- 一定数のリクエスト処理スレッドは確保した状態で，それを超えるリクエスト処理ス

レッドの増減を最大数と最小数を指定して制御したい場合は，J2EE サーバ起動時に生成したリクエスト処理スレッド数を維持する設定にしてください。これによって，システムとして最低確保しておきたい数のリクエスト処理スレッドを確保した状態で，クライアントからのアクセスピーク時のリクエスト処理スレッド数の増減に対応できます。未使用のリクエスト処理スレッド数が予備スレッド数の最大値を超えている場合も，J2EE サーバ起動時に生成した数のリクエスト処理スレッドは維持されます。

- 一度作成したスレッドを削除しないでプールし続けたい場合は，予備スレッド数の最大数を，Web クライアントとの最大接続数と同じ値にしてください。

このほか，Web アプリケーションの同時実行スレッド数との関係についても留意してください。Web アプリケーションの同時実行スレッド数については，「8.3.4　Web アプリケーションの同時実行数を制御する」を参照してください。

8.3.4　Web アプリケーションの同時実行数を制御する

Web アプリケーションの同時実行数制御では，Web フロントシステムの場合に，Web サーバが Web ブラウザなどのクライアントから受け付けたリクエストの処理を，同時に幾つのスレッドで実行するかを制御します。

同時実行スレッド数は，URL グループ単位，Web アプリケーション単位，または Web コンテナ単位で制御できます。同時実行スレッド数は，Web サーバ連携の場合，インプロセス HTTP サーバを使用する場合，どちらの場合も制御できます。

（1） 同時実行スレッド数制御の違い

Web コンテナ単位，Web アプリケーション単位，および URL グループ単位の同時実行スレッド数制御の違いは次のとおりです。

Web コンテナ単位

Web コンテナ全体で同時にリクエストを処理するスレッド数を設定できます。

Web アプリケーション単位

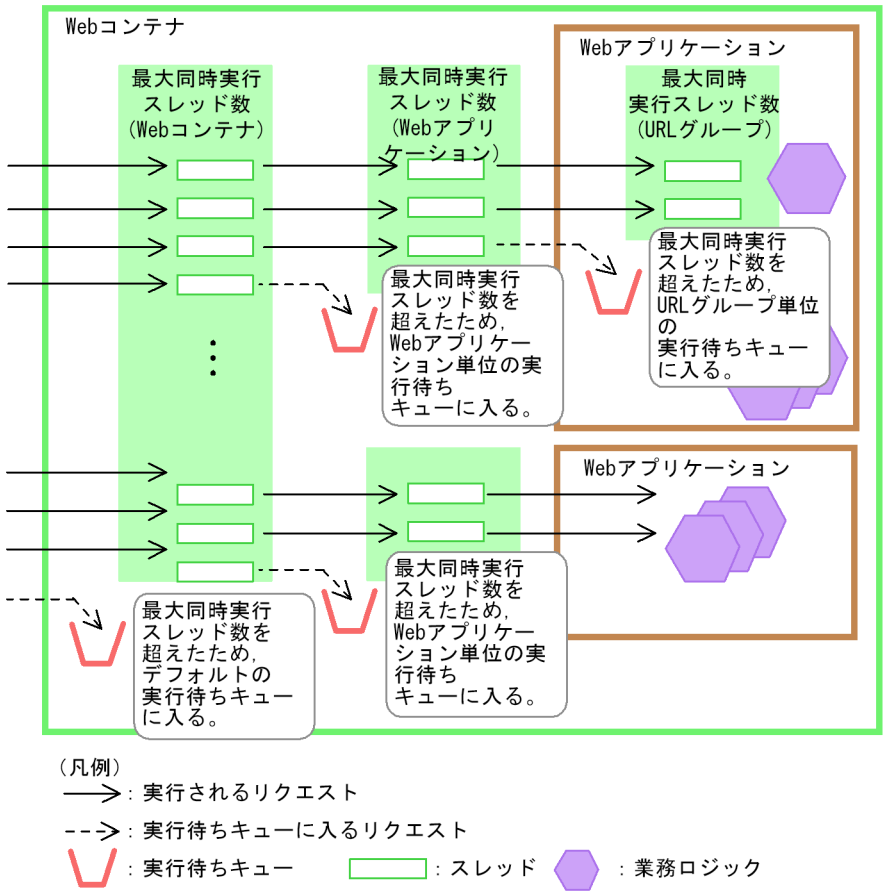
Web コンテナ上で動作する Web アプリケーションごとに，同時にリクエストを処理するスレッド数を設定できます。なお，Web アプリケーション単位の同時実行スレッド数制御は簡易 Web サーバに対しても有効になります。

URL グループ単位

リクエストを Web アプリケーション内の特定の業務処理（業務ロジック）に対応する URL に振り分ける場合，振り分け先 URL の処理ごとに，同時にリクエストを処理するスレッド数を設定できます。

Web コンテナ単位，Web アプリケーション単位，および URL グループ単位の同時実行スレッド数の関係を次の図に示します。

図 8-4　Web コンテナ単位，Web アプリケーション単位，および URL グループ単位の同時実行スレッド数の関係



Web アプリケーションに対するリクエストの実行は，Web コンテナ単位，Web アプリケーション単位，および URL グループ単位に設定した同時実行スレッド数に制限されます。Web コンテナ単位，Web アプリケーション単位および URL グループ単位に設定した同時実行スレッド数を超えるリクエストは，それぞれの実行待ちキューに入ります。

（2） 選択の指針

同時実行スレッド数制御の単位を選択するときの指針について説明します。

なお，同時実行スレッド数を制御する機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「2.15　同時実行スレッド数の制御の概要」を参照してください。

● Web アプリケーション単位の選択の指針

Web アプリケーション単位の同時実行数を制御することで，J2EE サーバが TCP 接続要

求だけではなく，Web アプリケーションの実行待ちキューを管理できるようになります。このため，J2EE サーバ上で実行する Web アプリケーションが一つだけの場合でも，Web アプリケーション単位の同時実行スレッド数を設定することをお勧めします。

Web アプリケーション単位での同時実行スレッド数の設定は，Web コンテナ単位で設定する場合に比べて，次のような利点があります。

- Web アプリケーションごとの同時実行スレッド数に上限を設けることで，特定の業務に対応する Web アプリケーションへのリクエストが増大した場合に，その Web アプリケーションが Web コンテナ全体の処理能力を占有しないようにできます。これによって，ほかの業務も滞りなく実行できます。
- CPU や I／O 処理に掛かる負荷が異なる複数の Web アプリケーションが Web コンテナ上にある場合，それぞれの条件に適した同時実行スレッド数が設定できます。
- Web アプリケーションごとにリクエストの実行待ちキューサイズが設定できるので，Web アプリケーションの特徴に応じた実行待ちキュー管理ができます。また，Web アプリケーション単位の実行待ちキュー以上のリクエストが送信された場合には，クライアントに HTTP レスポンスコードで通知できます。

なお，Web アプリケーション単位の同時実行スレッド数は，稼働中の J2EE サーバでも動的に変更できます。稼働中の J2EE サーバで実行する Web アプリケーションの同時実行スレッド数の動的変更の手順については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「2.19.2　同時実行スレッド数の動的変更の流れ」を参照してください。

● URL グループ単位の選択の指針

Web アプリケーション単位で同時実行スレッド数を制御している場合に，さらに業務ロジック単位で同時実行スレッド数の制御をしたいときには，URL グループ単位で同時実行スレッド数を制御します。

Web アプリケーションが次のような業務ロジックを含む場合，URL グループ単位の設定を検討してください。

- ほかの処理に影響を受けないで優先して実行したい業務ロジック
- ほかの処理に比べて処理時間が掛かる，または CPU や I／O の負荷が大きい業務ロジック

URL グループ単位での同時実行スレッド数の設定は，Web アプリケーション単位だけの設定に比べて，次のような利点があります。

- 重要度が高い業務ロジック（URL グループ）には，確実に実行するためのスレッド数を割り当てられます。これによって，ほかの業務ロジックに対するリクエスト数が増大した場合も，Web アプリケーション全体の同時実行スレッド数をその業務ロジックに占有されないで，重要度が高い業務ロジックを実行できます。

- 処理時間が掛かる業務ロジック（URL グループ）の同時実行数に上限を設けることで，特定の業務ロジックによって Web アプリケーション全体の同時実行数が占有されないように制御できます。
- Web アプリケーション内に CPU や I／O の負荷が異なる複数の業務ロジック（URL グループ）がある場合は，業務ロジックに応じた同時実行数が設定できます。
- Web アプリケーション内の業務ロジック（URL グループ）ごとにリクエストの待ち行列長（実行待ちキューのキューサイズ）が設定できるので，業務ロジックの特徴に応じた実行待ちキューを管理できます。また，この URL グループ単位の実行待ちキューがあふれた場合，クライアントに HTTP レスポンスコード 503（Service Temporarily Unavailable）を通知できます。

8.3.5 Enterprise Bean の同時実行数を制御する

ここでは，Enterprise Bean の同時実行数を制御する方法について，Enterprise Bean の種類ごとに説明します。なお，インスタンスプーリングおよびセッション制御による Enterprise Bean の同時実行数制御は，EJB コンテナの機能を利用して実現します。EJB コンテナの機能の詳細は，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (EJB コンテナ)」の「2. EJB コンテナ」を参照してください。

（1） Enterprise Bean の同時実行数制御で使用できる機能の種類

Enterprise Bean の同時実行数を制御する場合，EJB コンテナの機能である，次の 2 種類の機能を利用できます。

インスタンスプーリング

Enterprise Bean のインスタンスを事前に作成しておくことで，クライアントからリクエストが送信されたときにすぐ処理を実行できるようにする機能です。プールするインスタンス数の上限値を設定することで，上限値以上のリクエストの実行を待たせることができます。これによって，同時実行数が制御できます。

参考

運用時に CTM の同時実行数を動的に変更する場合は，上限値は無制限に設定してください。

セッション制御

セッション内で，同時に生成できるセッション（インスタンス）数を制限する機能です。

なお，Enterprise Bean の種類ごとに使用できる機能が異なります。

Enterprise Bean の種類ごとに使用できる同時実行数制御の機能を次の表に示します。

表 8-5　Enterprise Bean の種類ごとに使用できる同時実行数制御の機能

Enterprise Bean の種類	使用できる制御機能
Stateless Session Bean ※	インスタンスプーリング
Stateful Session Bean	セッション制御
Entity Bean	インスタンスプーリングとセッション制御
Message-driven Bean	インスタンスプーリング

注※　Stateless Session Bean の同時実行数を制御する場合は，CTM を利用することをお勧めします。CTM を利用して同時実行数を制御する方法については，「8.3.6　CTM を使用して同時実行数を制御する」で説明します。

なお，Enterprise Bean の同時実行数制御のうち，実行待ちリクエストをキューの概念で管理できるのは，Message-driven Bean だけです。Message-driven Bean では，JMS のキューを使用して実行待ちリクエストを管理します。これ以外の Enterprise Bean では，同時実行数以上のリクエストが送信された場合，設定に応じて次のどれかの処理が実行されます。

- インスタンスに空きが出るまで待ち続ける
- すぐに例外としてクライアントに返却する
- インスタンス取得用に設定したタイムアウト時間が経過したら，例外としてクライアントに返却する（Stateless Session Bean の method-ready プールまたは Entity Bean の pool プールの場合）

（2） Stateless Session Bean の同時実行数制御

Stateless Session Bean では，インスタンスプーリングを利用できます。通常の同時アクセス数をインスタンスプーリングの最小値として設定して，想定している最大同時アクセス数以上の値を，インスタンスプーリングの上限値に指定します。これによって，通常のアクセス時にはインスタンスを生成する時間を省略できるので，処理性能が向上します。さらに，アクセス数が増大した場合でも，想定している最大同時アクセス数までは処理でき，それ以上のリクエストの実行は待たせるように制御できます。

なお，デフォルトの設定の場合，上限値以上のリクエストがエラーで返却されることはありません。プールされているインスタンスに空きができるまで待ち続けます。エラーで返却したい場合は，必要に応じてインスタンス取得待ちのタイムアウトを設定してください。

（3） Stateful Session Bean の同時実行数制御

Stateful Session Bean では，クライアントごとにセッションごとの状態があるため，厳密な同時実行数制御はできません。ただし，セッション単位での流量制御（セッション制御）ができます。

想定している最大同時セッション数以上の値を，セッション制御の上限値として指定します。アクセス数が増大して，想定している最大同時セッション数以上のアクセスがあった場合には，セッションを確立しないで，クライアントに例外（java.rmi.RemoteException）を通知できます。

ポイント

リクエスト単位で同時実行制御が必要な場合は，サーブレット，JSP または Stateless Session Bean を経由して Stateful Session Bean を呼び出すことで，Web アプリケーションの同時実行数制御またはインスタンスプーリングを利用した同時実行数制御ができるようになります。

（4） Entity Bean の同時実行数制御

Entity Bean では，クライアントごとに管理する状態を持つセッションの上限値の設定とインスタンスプーリングによって，同時実行数の制御ができます。なお，セッション数が上限に達した場合，インスタンスプーリングに空きがあっても，新しいセッションでのリクエストは実行できません。

セッション数の上限を超えた場合は，セッションの作成に失敗した時点ですぐにクライアントに例外（java.rmi.RemoteException）が通知されます。また，デフォルトの設定の場合，インスタンスプール数の上限値以上のリクエストは，エラーで返却されることはなく，プールされているインスタンスに空きができるまで待ち続けます。エラーで返却したい場合は，必要に応じてインスタンス取得待ちのタイムアウトを設定してください。

なお，Entity Bean に対するリクエストの実行では，データベースへのアクセスが発生します。このため，同時実行できるリクエスト数は，データベースにアクセスするためのコネクション数にも制限されます。

（5） Message-driven Bean の同時実行数制御

Message-driven Bean では，インスタンスプーリングを利用できます。想定している最大メッセージ数以上の値を，インスタンスプーリングの上限値として指定します。これによって，メッセージ到着時にインスタンスを生成する時間を省略できるので，処理性能が向上します。さらに，メッセージ数が増大した場合でも，インスタンスプーリングの上限値以上のメッセージは，実行を待たせるように制御できます。

なお，Message-driven Bean では，JMS のキューによって，到着メッセージを実行待ちキューで管理できます。

参考

TP1 インバウンド連携機能を使用して OpenTP1 の SUP から Message-driven Bean を呼び出す場合や，Cosminexus JMS プロバイダを使用して Message-driven Bean を呼び出す場合，ここで説明した内容以外に，使用するコンポーネントを考慮した同時実行数を検討する必要があります。詳細は，マニュアル「Cosminexus 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「1.2.3　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）の機能」，または「7.　Cosminexus JMS プロバイダ」を参照してください。

8.3.6　CTM を使用して同時実行数を制御する

CTM を使用している場合，Stateless Session Bean の同時実行数を制御できます。

CTM は，J2EE サーバとは独立したプロセス群です。EJB クライアントと J2EE サーバ間の Stateless Session Bean の呼び出しを中継し，Stateless Session Bean を呼び出す際に，同時実行数制御をします。なお，CTM による同時実行数制御の単位は J2EE アプリケーションです。

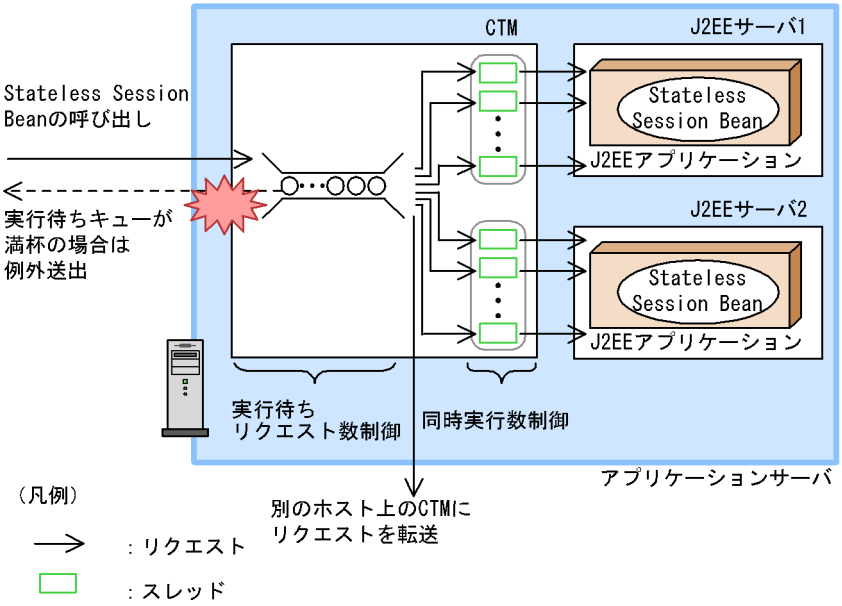
（1） CTM を使用した Stateless Session Bean の同時実行数制御

CTM による同時実行数制御（流量制御）によって，次に示すチューニングができます。

- CPU や I／O 処理に掛かる負荷が異なる，複数の J2EE アプリケーションが J2EE サーバ上にある場合に，それぞれの条件に適した同時実行数が設定できます。
- CTM で実行待ちキュー（スケジュールキュー）を管理するので，実行待ちリクエスト数を一定数以下に保ち，それ以上のリクエストが送信された場合にはクライアントに例外を通知できます。
- 特定の J2EE サーバの負荷が高い場合に，ほかの J2EE サーバにリクエストを振り分けられます。

CTM による Stateless Session Bean の同時実行数制御の例を次の図に示します。

図 8-5　CTM による Stateless Session Bean の同時実行数制御の例



ポイント

CTM は同一ホスト上の J2EE サーバでの Stateless Session Bean の呼び出しを制御して，そのホスト上での同時実行スレッド数を制御できます。アプリケーションサーバマシンのマシンスペックにも左右されますが，1 台のマシン当たり CTM デーモンを 1 プロセス起動して，J2EE サーバを 2 ～ 4 プロセス起動する構成を推奨します。

なお，CTM による同時実行スレッド数は，稼働中の CTM デーモンでも動的に変更できます。

CTM の同時実行数を制御する機能の詳細，および稼働中の CTM デーモンで実行する CTM の同時実行スレッド数動的変更の手順については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 拡張編」の「3.4　リクエストの流量制御」を参照してください。

(2) EJB コンテナのインスタンスプーリングとの使い分けの指針

CTM を使用して同時実行数を制御することをお勧めします。なお，CTM による同時実行数制御は，EJB コンテナでのインスタンスプーリングを利用した同時実行数制御と併用できます。

EJB コンテナの機能を使用した同時実行数制御に加えて，CTM によって同時実行数を制御するメリットは，次のとおりです。

● ある 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

ストを振り分けられます。

- 同時実行数が上限に達していなくても，特定の J2EE サーバの負荷が高い場合，ほかの J2EE サーバにリクエストを振り分けられます。
- CTM で実行待ちキュー（スケジュールキュー）の管理をして，実行待ちリクエスト数を一定に保ち，それ以上のリクエストを受け付けた場合はクライアントにエラーを通知できます。

ポイント

CTM による同時実行数制御と EJB コンテナでのインスタンスプーリングを併用する場合，Stateless Session Bean のインスタンスプーリング数は CTM の同時実行数以上に設定する必要があります。
また，運用時に CTM の同時実行数を動的に変更する場合は，Stateless Session Bean のインスタンスプーリング数の上限は無制限にする必要があります。なお，デフォルトでは無制限に設定されています。デフォルトから変更しないでください。

8.3.7　同時実行数を最適化するためのチューニングパラメタ

ここでは，同時実行数の最適化で使用するチューニングパラメタの設定方法についてまとめて示します。

（1）リクエスト処理スレッド数（インプロセス HTTP サーバ使用時）

インプロセス HTTP サーバを使用している場合の，リクエスト処理スレッド数のチューニングパラメタの設定方法について説明します。

次の表に示す項目を Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-6　リクエスト処理スレッド数のチューニングパラメタ（インプロセス HTTP サーバ使用時）

設定項目	設定対象	設定個所（パラメタ名）
J2EE サーバ起動時に生成するリクエスト処理スレッド数	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.init_threads
Web クライアントとの接続数の上限（リクエスト処理スレッド数の上限）	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.max_connections
Web クライアントとの接続数の上限を超えた場合に使用される TCP/IP の Listen キュー（バックログ）の最大値	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.backlog

設定項目	設定対象	設定個所（パラメタ名）
予備スレッド数の最大数	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.max_spare_threads
予備スレッド数の最小数	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.min_spare_threads

なお，Web サーバ連携で Hitachi Web Server を使用している場合のチューニングパラメタについては，マニュアル「Hitachi Web Server」を参照してください。

（2） Web アプリケーションの同時実行数

URL グループ単位，Web アプリケーション単位，または Web コンテナ単位に設定します。

（a） URL グループ単位の同時実行数

URL グループ単位の同時実行数のチューニングパラメタの設定方法について説明します。設定項目ごとに設定方法と設定個所が異なります。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-7 URL グループ単位の同時実行数のチューニングパラメタ（Smart Composer 機能で設定する項目）

設定項目	設定対象	設定個所（パラメタ名）
Web コンテナ単位での最大同時実行スレッド数（Web サーバ連携時）	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.max_threads
Web コンテナ単位での最大同時実行スレッド数（インプロセス HTTP サーバ使用時）	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.max_execute_threads
Web アプリケーション単位で同時実行数を制御するかどうか	論理 J2EE サーバ（j2ee-server）	webserver.container.thread_control.enabled
デフォルトの実行待ちキューサイズ	論理 J2EE サーバ（j2ee-server）	webserver.container.thread_control.queue_size

次の表に示す項目は，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは，WAR 属性ファイルに定義します。

表 8-8 URL グループ単位の同時実行数のチューニングパラメタ（サーバ管理コマンド（cjsetappprop）で設定する項目）

設定項目	設定個所（パラメタ名）
Web アプリケーション単位での最大同時実行スレッド数	<thread-control> タグ下の <thread-control-max-threads>
Web アプリケーションの占有スレッド数	<thread-control> タグ下の <thread-control-exclusive-threads>
Web アプリケーション単位の実行待ちキューサイズ	<thread-control> タグ下の <thread-control-queue-size>
URL グループ単位の同時実行スレッド数制御の定義名	<thread-control><urlgroup-thread-control> タグ下の <urlgroup-thread-control-name>
URL グループ単位での最大同時実行スレッド数	<thread-control><urlgroup-thread-control> タグ下の <urlgroup-thread-control-max-threads>
URL グループ単位の占有スレッド数	<thread-control><urlgroup-thread-control> タグ下の <urlgroup-thread-control-exclusive-threads>
URL グループ単位の実行待ちキューサイズ	<thread-control><urlgroup-thread-control> タグ下の <urlgroup-thread-control-queue-size>
URL グループ単位の制御対象となる URL パターン	<thread-control><urlgroup-thread-control> タグ下の <urlgroup-thread-control-mapping>

（b）Web アプリケーション単位の同時実行数

Web アプリケーション単位の同時実行数をチューニングするパラメタの設定方法について説明します。設定項目ごとに設定方法と設定個所が異なります。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-9 Web アプリケーション単位の同時実行数のチューニングパラメタ（Smart Composer 機能で設定する項目）

設定項目	設定対象	設定個所（パラメタ名）
Web コンテナ単位での最大同時実行スレッド数（Web サーバ連携時）	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.max_threads
Web コンテナ単位での最大同時実行スレッド数（インプロセス HTTP サーバ使用時）	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.max_execute_threads
Web アプリケーション単位で同時実行数を制御するかどうか	論理 J2EE サーバ（j2ee-server）	webserver.container.thread_control.enabled
デフォルトの実行待ちキューサイズ	論理 J2EE サーバ（j2ee-server）	webserver.container.thread_control.queue_size

次の表に示す項目は，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは，WAR 属性ファイルに定義します。

表 8-10 Web アプリケーション単位の同時実行数のチューニングパラメタ（サーバ管理コマンド（cjsetappprop）で設定する項目）

設定項目	設定個所（パラメタ名）
Web アプリケーション単位での最大同時実行スレッド数	<thread-control-max-threads>
Web アプリケーションの占有スレッド数	<thread-control-exclusive-threads>
Web アプリケーション単位の実行待ちキューサイズ	<thread-control-queue-size>

（c） Web コンテナ単位の同時実行数

Web コンテナ単位の同時実行数をチューニングするパラメタの設定方法について説明します。

次の表に示す項目を Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-11 Web コンテナ単位の同時実行数のチューニングパラメタ

設定項目	設定対象	設定個所（パラメタ名）
Web コンテナ単位での最大同時実行スレッド数（Web サーバ連携時）	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.max_threads
Web コンテナ単位での最大同時実行スレッド数（インプロセス HTTP サーバ使用時）	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.max_execute_threads

参考

このほか，Web コンテナでは，リダイレクタからの TCP 接続要求の最大待ち行列数も指定できます（論理 J2EE サーバ（j2ee-server）の webserver.connector.ajp13.backlog）。ただし，これはソケットの Listen キューの大きさを指定するキーであり，リクエストの実行待ちキューと直接の関係はありません。

（3） Enterprise Bean の同時実行数

Enterprise Bean の同時実行数は，Enterprise Bean 単位に設定します。Enterprise Bean の種類ごとに説明します。

(a) Stateless Session Bean の同時実行数

Stateless Session Bean の同時実行数をチューニングするパラメタの設定方法について説明します。

次の表に示す項目を，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは，Session Bean 属性ファイルに定義します。

表 8-12 Stateless Session Bean の同時実行数のチューニングパラメタ

設定項目	設定個所（パラメタ名）
プールで管理するインスタンスの最大値	<stateless><pooled-instance> タグ下の <maximum> ※
プールで管理するインスタンスの最小値	<stateless><pooled-instance> タグ下の <minimum>

注※ 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。

(b) Stateful Session Bean の同時実行数

Stateful Session Bean の同時実行数をチューニングするパラメタの設定方法について説明します。

次の表に示す項目を，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは，Session Bean 属性ファイルに定義します。

表 8-13 Stateful Session Bean の同時実行数のチューニングパラメタ

設定項目	設定個所（パラメタ名）
クライアントから作成可能なセッションの上限	<stateful> タグ下の <maximum-active-sessions>
使われていないインスタンスが削除されるまでの時間（分）	<stateful> タグ下の <removal-timeout>

(c) Entity Bean の同時実行数

Entity Bean の同時実行数をチューニングするパラメタの設定方法について説明します。

次の表に示す項目を，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは，Entity Bean 属性ファイルに定義します。

表 8-14 Entity Bean の同時実行数のチューニングパラメタ

設定項目	設定個所（パラメタ名）
クライアントから作成可能な Entity Bean の最大値	<maximum-instances>
使われていない Entity Bean の EJBObject が削除されるまでの時間（分）	<entity-timeout>

（d） Message-driven Bean の同時実行数

Message-driven Bean の同時実行数をチューニングするパラメタの設定方法について説明します。

次の表に示す項目を，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは，Message-driven Bean 属性ファイルに定義します。

表 8-15　Message-driven Bean の同時実行数のチューニングパラメタ

設定項目	設定個所（パラメタ名）
プールで管理するインスタンスの数の最大値	<pooled-instance><maximum>
プールで管理するインスタンスの数の最小値	<pooled-instance><minimum>

（4） CTM で制御する同時実行数

CTM で制御する同時実行数をチューニングするパラメタの設定方法について説明します。CTM デーモン，アプリケーション，および Stateless Session Bean に設定する項目があります。

● CTM デーモンに設定する項目

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-16　CTM で制御する同時実行数のチューニングパラメタ（Smart Composer 機能で設定する項目）

設定項目	設定対象	設定個所（パラメタ名）
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	論理 CTM（component-transaction-monitor）	ctm.DispatchParallelCount

● アプリケーションまたは Stateless Session Bean に設定する項目

次の表に示す項目は，サーバ管理コマンドで設定します。パラメタは，アプリケーション属性ファイルまたは Session Bean 属性ファイルに定義します。

表 8-17　CTM で制御する同時実行数のチューニングパラメタ（サーバ管理コマンドで設定する項目）

設定項目	定義ファイル	設定対象	設定個所（パラメタ名）
アプリケーションを CTM による同時実行数制御の対象にするかどうか	アプリケーション属性ファイル	アプリケーション	<managed-by-ctm>

設定項目	定義ファイル	設定対象	設定個所（パラメタ名）
アプリケーションの同時実行スレッド数	アプリケーション属性ファイル	アプリケーション	<scheduling> タグ下の <parallel-count>
Stateless Session Bean を CTM による同時実行数制御の対象にするかどうか	Session Bean 属性ファイル	Stateless Session Bean	<enable-scheduling>
Bean 単位のキューについての設定※	Session Bean 属性ファイル	Stateless Session Bean	<scheduling> タグ下の <parallel-count>

注※　スケジュールキューを Bean 単位に配置する場合に必要です。スケジュールキューの配置方法は，アプリケーション属性ファイルの <scheduling-unit> タブに設定します。

8.4 Enterprise Bean の呼び出し方法を最適化する

この節では，Enterprise Bean の呼び出し方を最適化する方法について説明します。

通常，Enterprise Bean は，リモートインタフェースを使用して RMI-IIOP 経由で呼び出されます。しかし，この方法の場合，同じ J2EE アプリケーションや同じ J2EE サーバ内で動作している Enterprise Bean から呼び出すときにも，リモート接続と同じオーバーヘッドが掛かってしまいます。

これに対して，Enterprise Bean の呼び出し方を最適化してスループットの向上を図るために，次の呼び出し方法が使用できます。

- ローカルインタフェースを使用した呼び出し
- リモートインタフェースのローカル呼び出し
- リモートインタフェースの参照渡し

これらの呼び出し方法には，それぞれ，次の表に示す特徴があります。この表では，標準仕様に準拠しているか（標準仕様），性能は向上するか（性能），位置透過性はあるか（位置透過），保守性に優れているか（保守性）の四つの特徴で比較しています。アプリケーションやシステムの特徴に応じて使い分けてください。

表 8-18　Enterprise Bean の呼び出し方法の特徴

呼び出し方法の種類	標準仕様	性能	位置透過	保守性
ローカルインタフェース	○	○	×	○
リモートインタフェースのローカル呼び出し	○	△	○	○
リモートインタフェースの参照渡し	×	○	△	×

（凡例）
○：対応している。／優れている。
△：対応しているが，一部制限がある。／やや悪い。
×：対応していない。／悪い。

なお，ローカルインタフェースの詳細については，EJB の仕様を参照してください。

リモートインタフェースのローカル呼び出しおよびリモートインタフェースの参照渡しの詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (EJB コンテナ)」の「2.13　EJB のリモートインタフェースの呼び出し」を参照してください。

8.4.1 ローカルインタフェースを使用する

アプリケーション開発時に，J2EE 標準仕様に準拠したローカルインタフェースを使用する方法です。

ローカルインタフェースは，Enterprise Bean の呼び出しを通常の同一スレッドのメソッド呼び出しとして実行する機能です。J2EE の標準機能なので，ポータビリティがあります。ただし，クライアント側，サーバ側の両方でローカル呼び出し専用のローカルインタフェースを使用したプログラムを作成する必要があるため，位置透過性はなくなります。

なお，ローカルインタフェースは，参照渡しによる呼び出しになります。また，同一 J2EE サーバ内であっても，異なる J2EE アプリケーション間での呼び出しには適用できません。

8.4.2 リモートインタフェースのローカル呼び出し最適化機能を使用する

同一アプリケーション内または同一 J2EE サーバ内での Enterprise Bean のリモートインタフェースによる呼び出しを，同一スレッドの延長によって実行する機能（**ローカル呼び出し最適化機能**）を使用する方法です。クライアント側，サーバ側の両方でリモートインタフェースを使用したプログラムを作成するので，位置透過性が保てます。

なお，デフォルトの状態では，同一アプリケーション内のローカル呼び出し最適化機能を使用する設定になっています。

8.4.3 リモートインタフェースの参照渡し機能を使用する

ローカル呼び出し最適化機能を使用する場合に，引数や戻り値で使用するオブジェクト型などのサイズが大きいデータの受け渡しを参照渡しにすることで，処理の高速化を図る方法です。

次のような条件の場合に，高速化が期待できます。

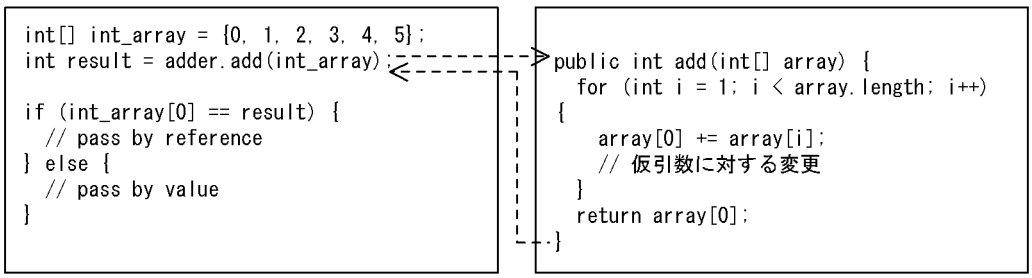
表 8-19 リモートインタフェースの参照渡しで高速化が期待できる条件

項目	処理内容
ビジネスメソッドの処理	仮引数経由で取得したデータに対して，直接変更をしないで参照だけする処理
ビジネスメソッドの引数と戻り値	配列や Collection などのクラスで大量のデータを受け渡す処理
呼び出す Enterprise Bean の位置	同一の J2EE アプリケーション内，または同一の J2EE サーバ内

ただし，リモートインタフェースでのデータの受け渡しは本来値渡しです。このため，値渡しを前提に作成されたプログラムは正しく動作しなくなるおそれがあるので注意し

てください。正しく動作しなくなるプログラムの例を次に示します。

図 8-6 正しく動作しなくなるプログラムの例



また，次の場合，この機能を使用しても効果がありません。

- プリミティブ型（単純型）の場合（常に値渡しになるため）
- 受け渡すデータのサイズが小さい場合

この機能は，J2EE サーバ単位または Enterprise Bean 単位で設定できます。

8.4.4 Enterprise Bean の呼び出し方法を最適化するためのチューニングパラメタ

ここでは，Enterprise Bean の呼び出し方法の最適化で使用するチューニングパラメタの設定方法についてまとめて示します。

(1) ローカルインタフェースの使用

アプリケーション作成時に，J2EE で定義されているローカルインタフェースを使用してください。

(2) リモートインタフェースのローカル呼び出し機能の使用

リモートインタフェースのローカル呼び出し機能のチューニングパラメタの設定方法について説明します。

次の表に示す項目を Smart Composer 機能で設定します。パラメタは簡易構築定義ファイルに定義します。

表 8-20 リモートインタフェースのローカル呼び出し機能のチューニングパラメタ

設定項目	設定対象	設定個所（パラメタ名）
ローカル呼び出し最適化機能の適用範囲	論理 J2EE サーバ（j2ee-server）	ejbserver.rmi.localinvocation.scope

(3) リモートインタフェースの参照渡し機能の使用

リモートインタフェースの参照渡し機能のチューニングパラメタの設定方法について説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは簡易構築定義ファイルに定義します。

表 8-21 リモートインタフェースの参照渡し機能のチューニングパラメタ（Smart Composer 機能で設定する項目）

設定項目	設定対象	パラメタ名
リモートインタフェースの参照渡し機能の使用（J2EE サーバ単位）	論理 J2EE サーバ（j2ee-server）	ejbserver.rmi.passbyreference

次の表に示す項目は，サーバ管理コマンド（cjsetappprop）で設定します。パラメタは Session Bean 属性ファイルまたは Entity Bean 属性ファイルに定義します。

表 8-22 リモートインタフェースの参照渡し機能のチューニングパラメタ（サーバ管理コマンド（cjsetappprop）で設定する項目）

設定項目	パラメタ名
リモートインタフェースの参照渡し機能の使用（Enterprise Bean 単位）	<pass-by-reference>

8.5　データベースへのアクセス方法を最適化する

データベースへのアクセス方法の最適化について説明します。

J2EE サーバでは，データベースアクセス方法のチューニングに次の 2 種類の方法を使用できます。

- コネクションプーリング
- ステートメントプーリング

8.5.1　コネクションプーリングを使用する

ここでは，J2EE サーバでコネクションプーリングを使用する利点と，コネクションプーリングに関連して使用できる機能について説明します。

また，Cosminexus JPA プロバイダでは，DB Connector のコネクションを取得します。Cosminexus JPA プロバイダが使用するコネクション数の見積もりについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (コンテナ共通機能)」の「6.2.4　DB Connector のコネクション数の見積もり」を参照してください。

(1) コネクションプーリングを使用する利点

データベースなどの EIS とのコネクションの確立は，負荷が高くなる処理です。コネクションプーリングを使用することで，負荷を軽くできます。**コネクションプーリング**は，J2EE サーバによって一度取得，生成したコネクションをプールしておき，それを再利用することで処理性能の向上を図る機能です。データベースなどにアクセスするたびにコネクションを再取得する場合に比べて，性能劣化を防ぐ効果があります。

なお，EIS との接続方法によって，使用できる機能が異なります。詳細は，マニュアル「Cosminexus 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。

なお，デフォルトの設定では，コネクションプーリングは有効になっています。

(2) 使用できる機能

コネクションプーリングで使用できる機能と設定時の指針について示します。

コネクションプーリングでは，次の機能を使用できます。

- プールするコネクションの最大値と最小値を指定する
- プール内のコネクションを検査して無効なコネクションを破棄する※ 1
- コネクション取得失敗時にリトライする※ 2

● スイーパによって使わないコネクションをプールから削除する※2

● コネクションウォーミングアップによってあらかじめコネクションをプールしておく

● コネクション枯渇時のコネクション取得要求を取得待ちキューに入れる

● コネクションプール内の不要なコネクションを段階的に減少させる

● コネクションプールをクラスタ化する※3

注※1　この機能を使用する場合，コネクションの検査にタイムアウトを設定することもできます。

注※2　これらの機能は，デフォルトの設定では無効になっています。必要に応じて使用してください。

注※3　この機能はデータベースが Oracle10g，または Oracle11g の場合に使用できます。

(a) プールするコネクションの最小値と最大値を指定する

コネクションプーリングを設定する場合は，プールするコネクションの最小値と最大値を設定します。無制限にした場合はコネクションが無制限に確立されます。

最小値および最大値は，定常状態で発生するデータベースなどの EIS への同時アクセス数，トランザクション数，業務の同時実行数などを参考にして決定してください。

(b) プール内のコネクションを検査して無効なコネクションを破棄する

プール内のコネクションに障害が発生していないかどうかをコネクション取得時または定期的にチェックして，障害が発生したコネクションをプールから削除します（**コネクション障害検知**）。コネクション取得時に障害が発生しているコネクションを取得してしまうことを防ぎ，接続に失敗する可能性を低くできます。この機能は，DB Connector を使用してデータベースにアクセスするためのコネクションに対して有効です。なお，SQL Server を使用する場合，DB Connector の selectMethod プロパティに direct を指定したときには，この機能は使用できません。

コネクション障害検知を実行するタイミングの使い分けの指針は次のとおりです。業務の種類に応じて使い分けてください。

表 8-23　コネクション障害検知を実行するタイミングの使い分けの指針

業務の種類	タイミング
• EIS との接続の失敗が許容されない業務 • コネクション取得頻度が低い業務	コネクション取得時に障害検知を実行する設定をお勧めします。コネクション取得時に掛かる処理時間は，コネクション障害検知を実行しない場合に比べて多く掛かりますが，障害が発生しているコネクションを取得する可能性が低くなります。

業務の種類	タイミング
• コネクション取得頻度が高い業務 • 障害が発生したコネクションを取得してもある程度許容されるような業務	定期的に障害検知を実行する設定をお勧めします。チェック間隔をある程度長くすることで，コネクション障害検知処理のために性能が劣化することを防げます。ただし，障害が発生しているコネクションを取得してしまうおそれがあります。

また，コネクション障害検知を実行する場合，コネクション障害検知の実行にタイムアウトを設定できます。サーバ障害やネットワーク障害が発生してリソースからの応答が返らない場合，コネクション障害検知の実行に対しても応答が返らなくなることがあります。タイムアウトを設定しておくことで，リソースからの応答が返らない場合も，コネクション障害検知処理を終了して，処理を継続できます。なお，この場合は，コネクションに障害が発生していると判断されます。

ポイント

- コネクション障害検知にタイムアウトを設定した場合，システム内で，コネクションプールのコネクション数に応じたコネクション管理スレッドが生成されます。このため，コネクション障害検知にタイムアウトを設定すると，設定しない場合に比べて多くのメモリを消費するので，注意が必要です。
  コネクション管理スレッドは，コネクションプールのコネクション数の最大値の 2 倍の数が作成されます。必要なメモリ使用量を適切に見積もってください。
- コネクション管理スレッドは，コネクション数調節機能のタイムアウトと共通で使用されます。このため，コネクション障害検知にタイムアウトを設定した場合，コネクション数調節機能のタイムアウトも有効になります。
- コネクション障害検知のタイムアウトを有効にしてコネクション障害検知を実施する場合，コネクションプールから取り除いた未使用コネクションは，コネクションプール内のコネクション数としてカウントされません。そのため，コネクションプール内のコネクションとコネクションプールから取り除いた未使用コネクションの総数が，コネクションプールのコネクション数の最大値を一時的に超える場合があります。

(c) コネクション取得失敗時にリトライする

コネクションの取得に失敗したときに，ユーザプログラムでリトライする必要がなくなります。業務処理のレスポンスが下がっても，障害発生による業務停止を防ぎたい場合に設定してください。

リトライ回数とリトライ間隔を設定する場合の指針について次に示します。

● リトライ回数を増やしたり，リトライ間隔を大きくしたりすると，コネクション取得処理が発生した場合に，待ち時間が発生するおそれがあります。

● リトライ間隔×リトライ回数の時間が長過ぎると，RMI-IIOP 通信のタイムアウトなどが発生しますが，タイムアウト後もリトライを続けます。このため，リトライ間隔×リトライ回数の時間は，タイムアウト値以下の値になるように設定してください。

● 同一リソースのコネクションを使用する業務の所要時間にも左右されますが，10 秒以

上を指定することを推奨します。

(d) スイーパによって使わないコネクションをプールから自動削除する

一定時間使用しなかったコネクションをスイーパによってプールから自動削除します。コネクションを未使用のままプーリングすると問題が発生する場合に設定します。リソースアダプタ単位に設定できます。

(e) コネクションウォーミングアップによってあらかじめコネクションをプールしておく

J2EE サーバの起動時またはリソースアダプタのスタート時にコネクションを最小値まであらかじめ取得してプールしておく機能です。コネクションプールの使用を開始した直後の，アプリケーションからのコネクション要求のレスポンスを向上できます。ただし，J2EE サーバ起動またはリソースアダプタのスタート時の処理時間が掛かります。

(f) コネクション枯渇時のコネクション取得要求を待ち状態にする

プールしているコネクションをすべて使い切っている状態でコネクション取得要求があった場合に，コネクション取得要求を待ち状態にする機能です。

使用中のコネクションが解放されるか，コネクションが破棄されてコネクション最大数よりも少なくなった場合に，すぐにコネクション取得要求を再開できます。また，待ち時間も設定できるので，一定時間を経過してコネクションを取得できなかった場合はエラーにできます。

(g) コネクションプール内の不要なコネクションを段階的に減少させる

コネクションプール内に不要なコネクションがある場合に，コネクション数を段階的に減らす機能です（**コネクション数調節機能**）。

一定間隔でコネクションプールの数をチェックします。チェック直前までの間の最大同時実行コネクション数を基準として，プール内にそれ以上の数のコネクションがある場合は，多い分のコネクションを削除します。これによって，プール内のコネクションは実際の稼働実績に適した数になるので，コネクション生成コストの削減や，リソース資源の節約ができます。

また，コネクションの削除処理にタイムアウトを設定できます。タイムアウトには，コネクション管理スレッドを使用します。なお，コネクション管理スレッドは，コネクション障害検知のタイムアウトでも使用されます。

ポイント

コネクション管理スレッドは，コネクション障害検知のタイムアウトと共通で使用されます。このため，コネクション数調節機能にタイムアウトを設定した場合，コネクション障害検知のタイムアウトも有効になります。
また，コネクション数調節機能を使用すると，コネクションプールのコネクション数を調整するために取り除いた未使用コネクションは，コネクションプール内のコネクション数としてカウントされません。そのため，コネクションプール内のコネクションとコネクションプールから取り除いた未使用コネクションの総数が，コネクションプールのコネクション数の最大値を一時的に超える場合があります。

（h）コネクションプールをクラスタ化する

データベースをクラスタ構成にしている場合，コネクションプールもクラスタ構成にして，クラスタ構成になっているデータベースノードごとにコネクションプールを保持できます（**コネクションプールのクラスタ化機能**）。データベースノードとコネクションプールは 1:1 で対応しています。データベースノードに障害が発生するとそれに対応するコネクションプールも一時停止し，使用できなくなります。これによって，障害が発生したコネクションを使用するおそれがなくなるため，システムの可用性が向上します。

8.5.2 ステートメントプーリングを使用する

ここでは，ステートメントプーリングを使用する利点と，設定の指針について説明します。

この機能は，DB Connector を利用している場合に使用できる機能です。また，使用できるかどうかは，EIS との接続方法によって異なります。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は使用できません。詳細は，マニュアル「Cosminexus 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。

（1）ステートメントプーリングを使用する利点

データベースにアクセスするときに必要な SQL 文やストアドプロシジャなどのステートメントの生成は，負荷が高くなる処理です。ステートメントプーリングを使用することで，負荷を軽くできます。**ステートメントプーリング**は，一度生成したステートメントである PreparedStatement と CallableStatement をプーリングしておき，それを再利用することで処理性能の向上を図る機能です。データベースなどにアクセスするたびにステートメントを再生成する場合に比べて，処理性能が向上します。

なお，PreparedStatement および CallableStatement とは，それぞれ，JDBC の API である java.sql.PreparedStatement と java.sql.CallableStatement のインスタンスです。

（2）設定の指針

ステートメントプーリングを利用するためには，アプリケーション開発時に次の点に留

意する必要があります。

- 同じシグネチャを持つ java.sql.Connection#prepareStatement メソッドを使用します。
- java.sql.Connection#prepareStatement メソッドでは同じ引数値を指定します。

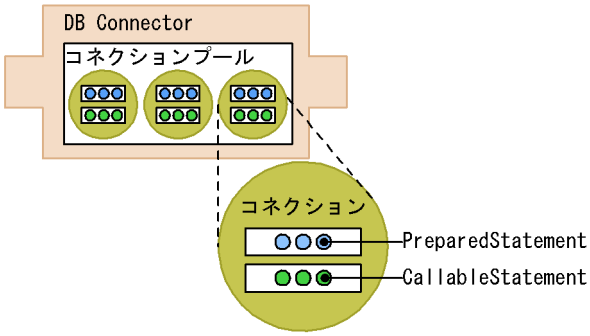
これらを満たさないアプリケーションでは，ステートメントプーリングは有効に動作しません。

また，ステートメントプーリングを使用するときには，コネクションプーリングとの関連を理解した上で使用してください。アプリケーション構成および環境設定時に留意する点を次に示します。

- 物理コネクションのインスタンスごとに PreparedStatement および CallableStatement がプールされます。このため，コネクション取得時にプールされているコネクションが複数ある場合には，ステートメントがまだプールされていないコネクションが割り当てられることがあります。
- それぞれのステートメントのインスタンスをプールするときには，使用している JDBC ドライバのコネクションごとの上限値を考慮してプールサイズを設定する必要があります。

コネクションプールとステートメントプールの関係を次の図に示します。

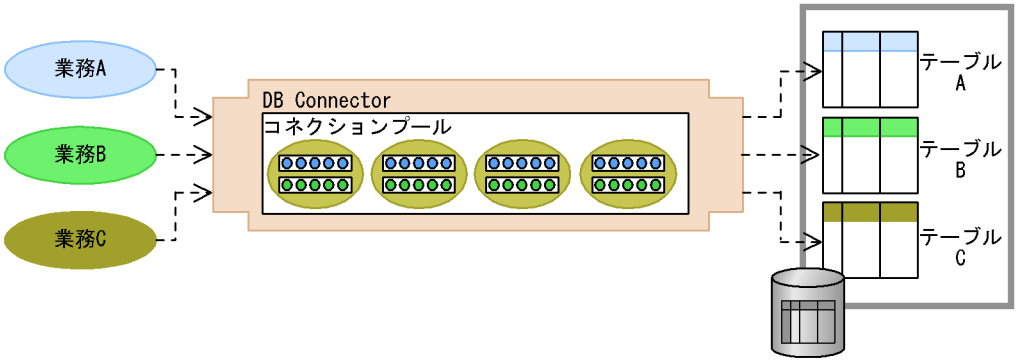
図 8-7　コネクションプールとステートメントプールの関係



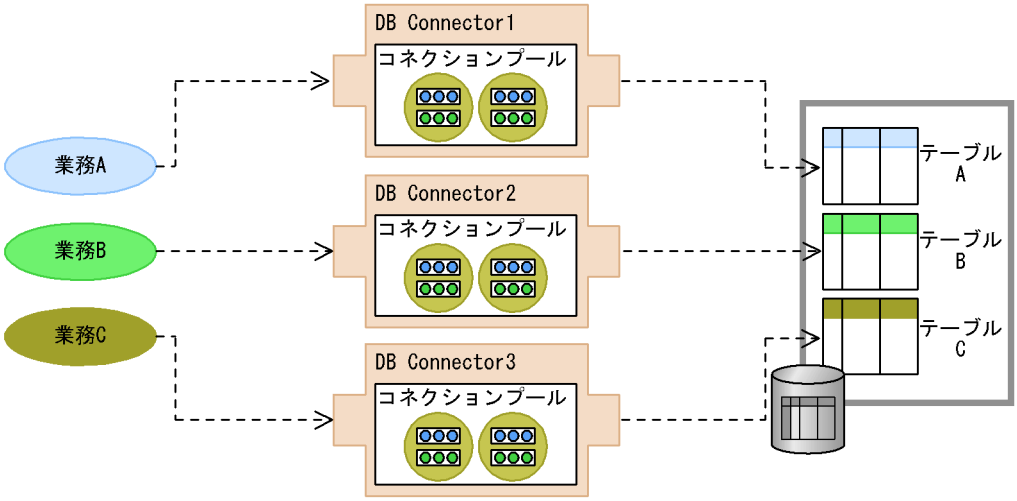
ステートメントプーリングのプールサイズを決めるときには，業務処理の内容に応じて，コネクションプーリングとの関係を考慮した上で決めてください。例えば，異なる業務処理で同じデータベースを使用する場合であっても，異なるテーブルにアクセスする場合や異なる SQL 文を使用する場合は，一つの DB Connector でコネクションプールとステートメントプールのサイズを大きくするよりも，業務処理ごとに別々の DB Connector を用意して，その業務で必要な分だけの小さなコネクションプールやステートメントプールを用意する方が，効率が良い場合があります。

図 8-8　業務処理に応じたコネクションプールとステートメントプールの利用

●複数の業務で一つの大きなプールを使用する方法



●業務ごとに一つの小さなプールを占有して使用する方法



ただし，DB Connector を複数用意してプールサイズを小さくすると，業務ごとにプールされているコネクション数の余裕がなくなり，アクセスのピーク時にコネクション不足が発生するおそれがあります。このため，業務の同時実行数などを詳細に見積もって，コネクション数不足ができるだけ発生しないようにチューニングしてください。また，プールのサイズは，業務処理ごとに，どのくらいのステートメントを利用するかによって調整してください。

一つの DB Connector を利用する場合のステートメントプールサイズは，次の値を集計して決定します。

PreparedStatement のプールサイズを設定する指針とサイズの算出方法

コネクションごとの JDBC ドライバのリソース制限を上限として，同一の DB Connector を利用する PreparedStatement の利用数を基に算出してください。なお，JDBC ドライバのリソース制限については，使用している JDBC ドライバの制限値に従ってください。

利用数は，次の数を集計して求めます。

- サーブレットおよび JSP から，異なる引数を指定して java.sql.Connection#prepareStatement メソッドを呼び出している数
- Session Bean および Entity Bean（BMP）から，異なる引数を指定して java.sql.Connection#prepareStatement メソッドを呼び出している数
- Entity Bean（CMP）に対して，J2EE サーバが内部的に PreparedStatement で使用する SQL の数
- Message-driven Bean から，異なる引数を指定して java.sql.Connection#prepareStatement メソッドを呼び出している数

CallableStatement のプールサイズを設定する指針とサイズの算出方法

コネクションごとの JDBC ドライバのリソース制限を上限として，同一の DB Connector を利用する CallableStatement の利用数を基に算出してください。なお，JDBC ドライバのリソース制限については，使用している JDBC ドライバの制限値に従ってください。

利用数は，次の数を集計して求めます。

- サーブレットおよび JSP から，異なる引数を指定して java.sql.Connection#prepareCall メソッドを呼び出している数
- Session Bean および Entity Bean（BMP）から，異なる引数を指定して java.sql.Connection#prepareCall メソッドを呼び出している数
- Message-driven Bean から，異なる引数を指定して java.sql.Connection#prepareCall メソッドを呼び出している数

8.5.3 データベースへのアクセス方法を最適化するためのチューニングパラメタ

ここでは，データベースへのアクセス方法の最適化で使用するチューニングパラメタの設定方法についてまとめて示します。

(1) コネクションプーリング

コネクションプーリングのチューニングパラメタの設定方法について説明します。

次の表に示す項目は，リソースアダプタ単位に，サーバ管理コマンド（cjsetresprop/cjsetrarprop）で設定します。パラメタは，Connector 属性ファイルに定義します。

表 8-24 コネクションプーリングのチューニングパラメタ

設定項目	設定個所（パラメタ名）※1
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	<property> タグに指定する MinPoolSize
コネクションプールにプールするコネクションの最大値	<property> タグに指定する MaxPoolSize

設定項目	設定個所（パラメタ名）※1
プール内のコネクションに障害が発生しているかどうかをチェックする方法の選択	<property> タグに指定する ValidationType
プール内のコネクションに障害が発生しているかどうかのチェックを定期的に実行する場合の間隔	<property> タグに指定する ValidationInterval
コネクション枯渇時にコネクション取得要求をキューで管理するかどうかの選択	<property> タグに指定する RequestQueueEnable
コネクション枯渇時のコネクション取得要求をキューで管理する場合の待ち時間	<property> タグに指定する RequestQueueTimeout
コネクションの取得に失敗した場合のリトライ回数※2	<property> タグに指定する RetryCount
コネクションの取得に失敗した場合のリトライ間隔※2	<property> タグに指定する RetryInterval
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を判定するまでの時間	<property> タグに指定する ConnectionTimeout
コネクションの自動破棄（コネクションスイーパ）が動作する間隔	<property> タグに指定する SweeperInterval
コネクションのウォーミングアップを使用するかどうかの選択	<property> タグに指定する Warmup
コネクション障害検知にタイムアウト時間を設定するかどうかの選択※3	<property> タグに指定する NetworkFailureTimeout
コネクション数調節機能での削除処理にタイムアウト時間を設定するかどうかの選択※3	
コネクション数調節機能が動作する間隔	<property> タグに指定する ConnectionPoolAdjustmentInterval

注※1　コネクションプールのクラスタ化機能を使用している場合は，メンバリソースアダプタで設定します。

注※2　コネクションプールのクラスタ化機能を使用している場合は，設定できません。

注※3　コネクション障害検知のタイムアウトとコネクション数調節機能のタイムアウトでは，共通のコネクション管理メソッドを使用します。このため，どちらかのタイムアウトを設定すると，コネクション障害検知とコネクション数調節機能の両方のタイムアウトが有効になります。コネクション障害検知，およびコネクション数調節機能のタイムアウト時間を変更したい場合は，簡易構築定義ファイルで設定する J2EE サーバ用のパラメタ（ejbserver.connectionpool.validation.timeout）の設定を変更してください。詳細は，マニュアル「Cosminexus 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「4.14.1　J2EE サーバ用ユーザプロパティを設定するパラメタ」を参照してください。

ポイント

HiRDB でコネクションプーリング機能を使用する場合，次の設定が必要です。

- HiRDB クライアント環境変数の PDSWATCHTIME に「0」を指定してください。このパラメタについては，マニュアル「HiRDB UAP 開発ガイド」を参照してください。

XDM/RD E2 でコネクションプーリング機能を使用する場合，次の設定が必要です。

- DB コネクションサーバのコントロール空間起動制御文／サーバ空間起動制御文の SVINTERVAL パラメタに「0」を指定してください。このパラメタについては，マニュアル「VOS3　Database Connection Server」を参照してください。

(2) ステートメントプーリング

ステートメントプーリングのチューニングパラメタの設定方法について説明します。

次の表に示す項目は，サーバ管理コマンド（cjsetresprop/cjsetrarprop）で設定します。パラメタは，Connector 属性ファイルに定義します。

表 8-25　ステートメントプーリングのチューニングパラメタ

設定項目	パラメタ名※
物理コネクションごとにプールする PreparedStatement の数	<config-property> タグに指定する PreparedStatementPoolSize
物理コネクションごとにプールする CallableStatement の数	<config-property> タグに指定する CallableStatementPoolSize

注※　コネクションプールのクラスタ化機能を使用している場合は，メンバリソースアダプタで設定します。

8.6 タイムアウトを設定する

アプリケーションサーバのシステムでは，トラブル発生時にリクエストの応答が戻ってこない状態になることを防ぐために，幾つかのポイントにタイムアウトを設定できます。

この節では，システム全体でタイムアウトが設定できるポイントと，設定する場合の指針について説明します。

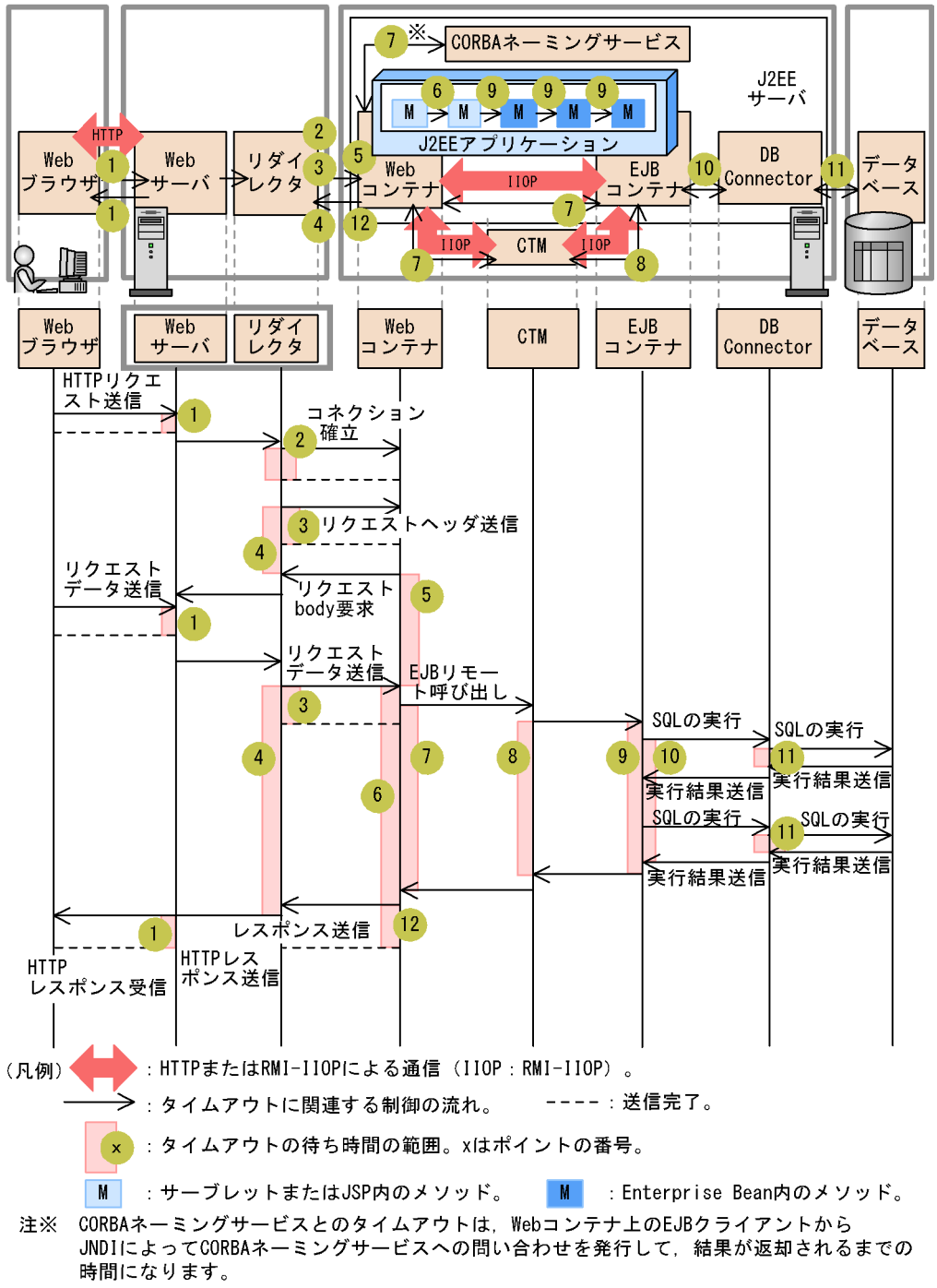
参考

TP1 インバウンド連携機能を使用して OpenTP1 からアプリケーションサーバを呼び出す場合，この節で説明する内容のほか，OpenTP1 側の設定を考慮したタイムアウトの設定が必要です。詳細は，マニュアル「Cosminexus 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「1.2.3　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）の機能」を参照してください。

8.6.1 タイムアウトが設定できるポイント

J2EE アプリケーションを実行するシステムでは，次の図に示すポイントにタイムアウトが設定できます。なお，次の図は，クライアントが Web ブラウザの場合です。また，Web サーバ連携をする場合とインプロセス HTTP サーバを使用する場合で，ポイントが異なります。

図 8-9　タイムアウトが設定できるポイント（Web サーバ連携の場合）



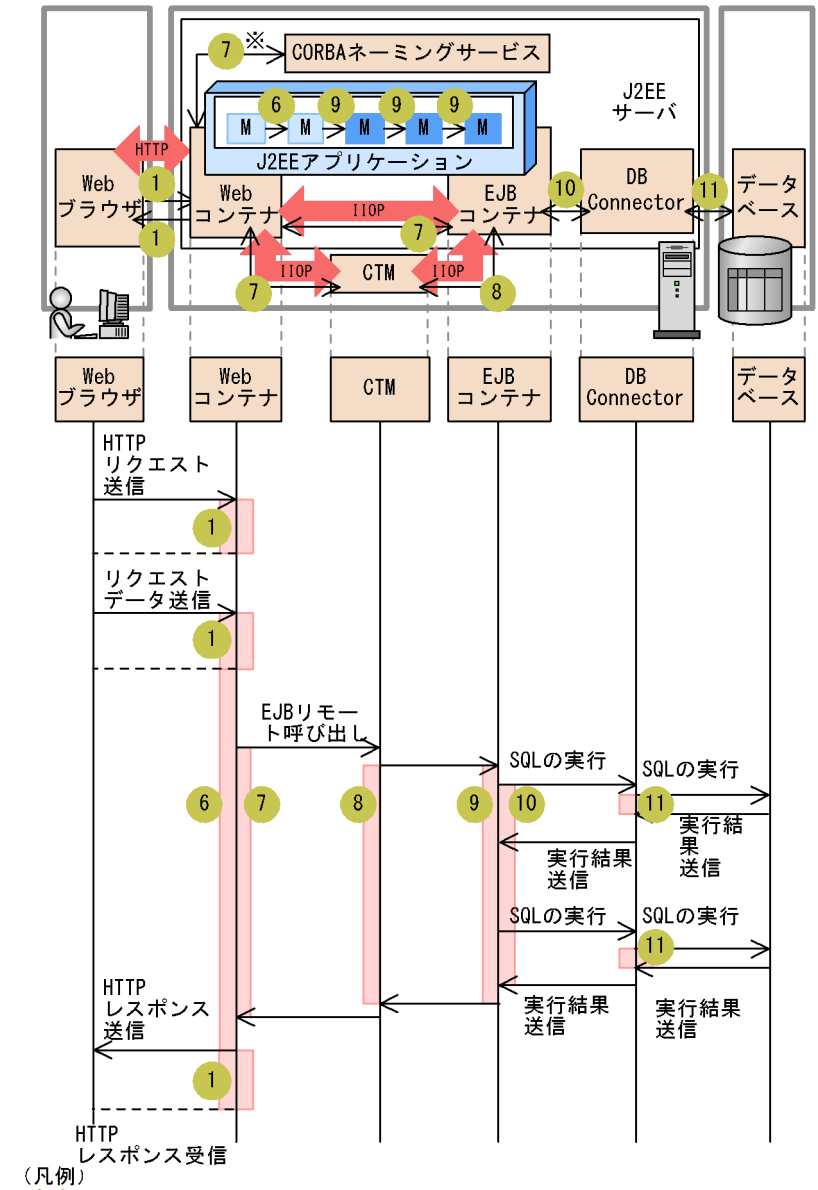
注※　CORBAネーミングサービスとのタイムアウトは，Webコンテナ上のEJBクライアントからJNDIによってCORBAネーミングサービスへの問い合わせを発行して，結果が返却されるまでの時間になります。

なお，クライアントが EJB クライアントの場合は，Web コンテナを EJB クライアントに置き換えてください。EJB クライアントからデータベースまでの範囲のタイムアウト

が設定できます。

また，インプロセス HTTP サーバを使用する場合は，リダイレクタは該当しません。このため，タイムアウトを設定するポイントとして，ポイント 2 ～ 5 と 12 は該当しません。インプロセス HTTP サーバを使用する場合にタイムアウトが設定できるポイントについて，次の図に示します。

図 8-10　タイムアウトが設定できるポイント（インプロセス HTTP サーバの場合）



（凡例）

：HTTPまたはRMI-IIOPによる通信（IIOP：RMI-IIOP）。

：タイムアウトに関連する制御の流れ。　----　：送信完了。

x　：タイムアウトの待ち時間の範囲。xはポイントの番号。

M　：サーブレットまたはJSP内のメソッド。　M　：Enterprise Bean内のメソッド。

注※　CORBAネーミングサービスとのタイムアウトは，Webコンテナ上のEJBクライアントからJNDIによってCORBAネーミングサービスへの問い合わせを発行して，結果が返却されるまでの時間になります。

それぞれのポイントに設定するタイムアウトは，次の表に示すような用途で使い分けられます。

表 8-26 各ポイントに設定するタイムアウトの目的とデフォルトのタイムアウト設定

ポイント	タイムアウトの種類	主な用途
1	サーバ側で設定するクライアントからのリクエスト受信およびクライアントへのデータ送信のタイムアウト	Web サーバ連携の場合 通信路の障害，または Web サーバの障害の検知 インプロセス HTTP サーバの場合 通信路の障害，または不正なクライアントからのアクセスの検知
2	リダイレクタ側で設定する Web コンテナへのリクエスト送信処理のうち，コネクション確立のタイムアウト	通信路の障害または Web コンテナの障害検知
3	リダイレクタ側で設定する Web コンテナへのリクエスト送信処理のうち，リクエストヘッダおよびリクエストボディ送信のタイムアウト	通信路の障害または Web コンテナの障害検知
4	リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
5	Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト	通信路の障害または Web サーバの障害検知
6	Web アプリケーションで設定するメソッドの実行時間のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど）
7	EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI ネーミングサービス呼び出しのタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
8※	EJB クライアント側で設定する CTM からの Enterprise Bean 呼び出しのタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
9	EJB で設定するメソッドの実行時間のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど）
10	EJB コンテナ側で設定するデータベースのトランザクションタイムアウト	データベースサーバの障害（サーバダウンまたはデッドロックなど）の検知，またはリソースの長時間占有防止
11	データベースのタイムアウト	データベースサーバの障害（サーバダウンまたはデッドロックなど）の検知，またはリソースの長時間占有防止
12	Web コンテナ側で設定するリダイレクタへのレスポンス送信のタイムアウト	通信路の障害またはリダイレクタの障害検知

注※ CTM を使用している場合にだけ存在するポイントです。CTM を利用しない構成の場合，ポイント 7 の範囲は Web コンテナから EJB コンテナに EJB リモート呼び出しを実行してから，EJB コンテナから Web コンテナに実行結果が送信されるまでの間になります。

これらのタイムアウトの基本的な設定指針は次のとおりです。

- タイムアウト値の設定は，呼び出し元（Web クライアントまたは EJB クライアント）に近いほど大きな値を設定するのが原則です。このため，次の関係で設定することを推奨します。
  - ポイント 1< ポイント 5
  - ポイント 4> ポイント 6> ポイント 7
  - ポイント 7= ポイント 8> ポイント 9> ポイント 10
  - ポイント 10 ＜ポイント 11
  - ポイント 4> ポイント 6
  - ポイント 1 ＜ポイント 12
- 4，7，10，11 のポイントのタイムアウト値を設定する場合は，呼び出し処理に通常どの程度の時間が掛かっているかを見極めた上で，呼び出す処理（業務）ごとに算出して設定してください。

なお，1 ～ 12 のポイントは，システムでの位置づけによって，次の三つに分けられます。

- Web フロントシステムで意識する必要があるポイント（1 ～ 6，および 12）
  詳細は，「8.6.2　Web フロントシステムでのタイムアウトを設定する」を参照してください。
- バックシステムで意識する必要があるポイント（7 ～ 9）
  詳細は，「8.6.3　バックシステムでのタイムアウトを設定する」を参照してください。
- データベース接続時に意識する必要があるポイント（10 と 11）
  このポイントは，さらにトランザクションでのタイムアウトとデータベースでのタイムアウトに分けて意識する必要があります。
  詳細は，「8.6.4　トランザクションタイムアウトを設定する」および「8.6.5　データベースでのタイムアウトを設定する」を参照してください。

それぞれのポイントでの設定については，「8.6.7　タイムアウトを設定するチューニングパラメタ」を参照してください。

参考

それぞれのポイントのデフォルト値は次のとおりです。

ポイント	デフォルト値
1	300 秒
2	30 秒
3	100 秒
4	3,600 秒
5	600 秒
6	設定されていません。タイムアウトしません。
7	設定されていません。レスポンスを待ち続けます。
8	ポイント 7 と同じ値が Enterprise Bean 呼び出し時に自動的に引き継がれて設定されます。
9	設定されていません。タイムアウトしません。
10	180 秒
11	データベースの種類とタイムアウトの設定個所ごとに異なります。 HiRDB の場合 ロック解放待ちタイムアウト：180 秒 レスポンスタイムアウト：0 秒（HiRDB クライアントは HiRDB サーバからの応答があるまで待ち続けます） リクエスト間隔タイムアウト：600 秒 Oracle の場合（グローバルトランザクションを使用するとき） ロック解放待ちタイムアウト：60 秒 SQL Server の場合 メモリ取得待ちタイムアウト：-1（-1 を指定した場合の動作は，SQL Server のドキュメントを参照してください） ロック解放待ちタイムアウト：-1（ロックが解放されるまで待ち続けます） XDM/RD E2 の場合 ロック解放待ちタイムアウト：なし（タイムアウト時間を監視しません） SQL 実行 CPU 時間タイムアウト：10 秒 SQL 実行経過時間タイムアウト：0 秒（タイムアウト時間を監視しません） トランザクション経過時間タイムアウト：600 秒 レスポンスタイムアウト：0 秒（HiRDB クライアントは XDM/RD E2 サーバからの応答があるまで待ち続けます）
12	600 秒

8.6.2 Web フロントシステムでのタイムアウトを設定する

ここでは，Web フロントシステムでのタイムアウトの設定について説明します。

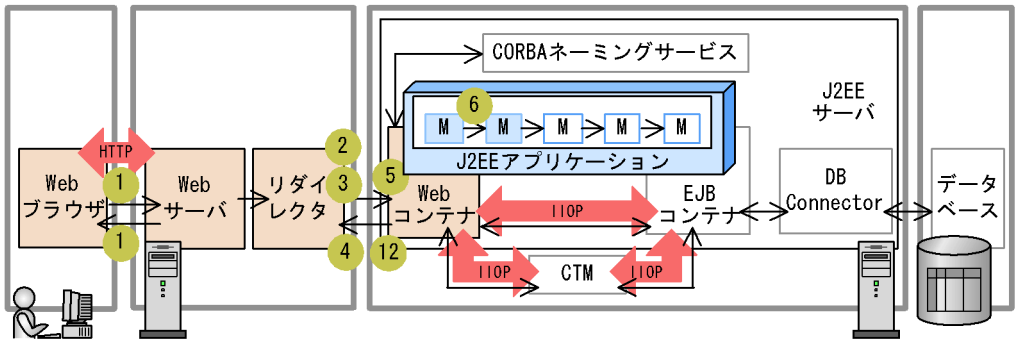
Web フロントシステムのタイムアウトを設定する場合は，システム全体のタイムアウトのうち，次の図に示す，1 ～ 6 および 12 のポイントについて意識する必要があります。この番号は，図 8-9 または図 8-10 と対応しています。

ポイント

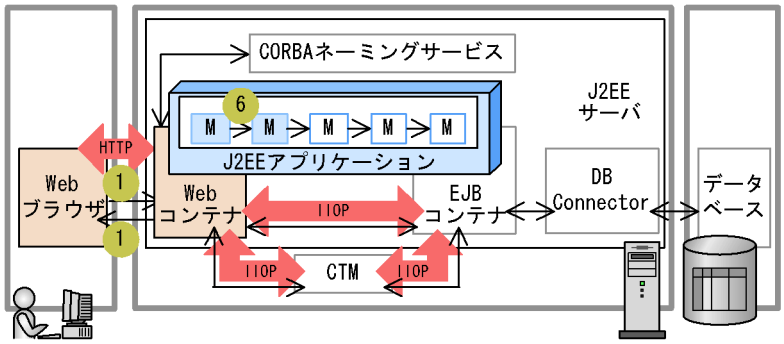
インプロセス HTTP サーバを使用する場合，設定できるのは 1 と 6 のポイントです。2 ～ 5，および 12 のポイントは該当しません。

図 8-11　Web フロントシステムの場合に意識するタイムアウトのポイント（1 ～ 6，および 12 のポイント）

●Webサーバ連携をする場合



●インプロセスHTTPサーバを使用する場合



（凡例）

：Webフロントシステムで意識するタイムアウトを設定する対象。

：HTTPまたはRMI-IIOPによる通信（IIOP：RMI-IIOP）。

：タイムアウトに関連する制御の流れ。

M　：サーブレットまたはJSP内のメソッド。

● **Web サーバでのクライアントからのリクエスト受信，およびクライアントへのデータ送信待ち時間（1 のポイント）**

Web ブラウザからの要求が滞った場合に，タイムアウトによってリダイレクタのリソースを解放します。また，Web ブラウザへの応答が滞った場合（Web ブラウザが受信しない場合）に，タイムアウトによってリダイレクタおよび J2EE サーバ内の Web

コンテナのリソースを解放します。
Web サーバ連携の場合，これらの待ち時間には，同じ値が設定されます。
インプロセス HTTP サーバを使用する場合，クライアントからのリクエスト受信待ち時間と，クライアントへのデータ送信待ち時間には，それぞれ異なる値を指定できます。

● **Web サーバに登録したリダイレクタでの Web コンテナへのリクエスト送信待ち時間（2 および 3 のポイント）**

リダイレクタから Web コンテナへのリクエスト送信時に，Web コンテナ自体のトラブル，またはリダイレクタと Web コンテナ間の通信路でのトラブルによって制御が戻らなくなった場合に，タイムアウトによってリダイレクタのリソースを解放します。また，同時に Web ブラウザにエラーを通知します。Web サーバと連携する場合にだけ設定できるポイントです。
ポイント 2 は Web コンテナとのコネクション確立の待ち時間，ポイント 3 は Web コンテナへのリクエスト送信処理の待ち時間です。

● **Web サーバに登録したリダイレクタでの Web コンテナからのデータ受信待ち時間（4 のポイント）**

J2EE アプリケーションで何かのトラブルが発生して制御が戻らなくなった場合に，タイムアウトによってリダイレクタのリソースを解放します。また，同時に Web ブラウザにエラーを通知します。Web サーバと連携する場合にだけ設定できるポイントです。

ポイント

設定の単位はワーカです。このため，業務によって処理に掛かる時間が異なる場合は，業務に対応する Web アプリケーション単位でワーカを定義してタイムアウトを設定することをお勧めします。

● **Web コンテナでのリダイレクタからのデータ受信待ち時間（5 のポイント）**

ブラウザからの要求が滞ったときに，J2EE サーバ（Web コンテナ）のリソースを解放します。Web サーバと連携する場合にだけ設定できるポイントです。

● **Web コンテナ上でのリクエスト処理待ち時間（6 のポイント）**

J2EE アプリケーションの実行時間監視機能を利用します。
「8.6.6　J2EE アプリケーションのメソッドタイムアウトを設定する」を参照してください。

● **Web コンテナからリダイレクタへのレスポンス送信待ち時間（12 のポイント）**

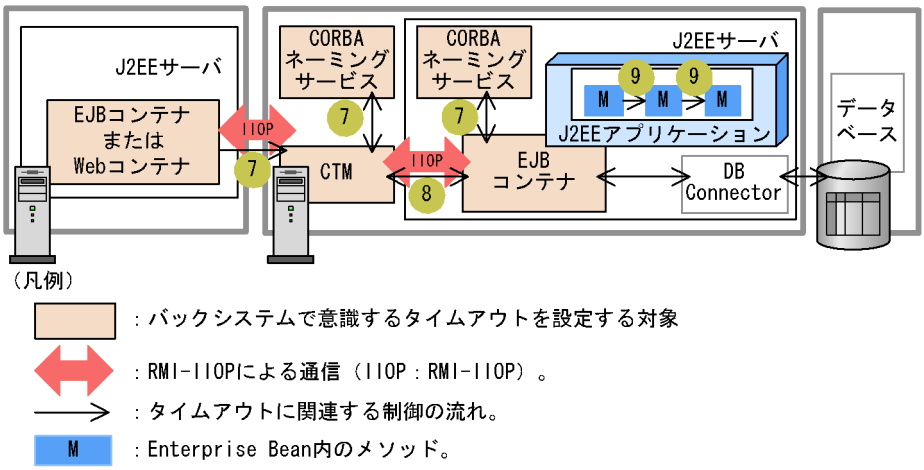
Web コンテナからリダイレクタへのレスポンス送信時に，リダイレクタ自体のトラブル，またはリダイレクタと Web コンテナ間の通信路でのトラブルによって制御が戻らなくなった場合に，タイムアウトによって Web コンテナのリソースを解放します。また，同時に Web ブラウザにエラーを通知します。Web サーバと連携する場合にだけ設定できるポイントです。

8.6.3　バックシステムでのタイムアウトを設定する

ここでは，バックシステムでのタイムアウトの設定について説明します。なお，バックシステムでのタイムアウトのうち，データベースなどの EIS とのトランザクションに関連するタイムアウトについては，「8.6.4　トランザクションタイムアウトを設定する」で説明します。ここでは EJB クライアントと EJB コンテナに関連するタイムアウトについて説明します。

バックシステムのタイムアウトを設定する場合は，システム全体のタイムアウトのうち，次の図に示す，7 ～ 9 のポイントについて意識する必要があります。なお，この番号は，図 8-9 または図 8-10 と対応しています。

図 8-12　バックシステムの場合に意識するタイムアウトのポイント



- **Enterprise Bean をリモート呼び出し（RMI-IIOP 通信）する場合と CORBA ネーミングサービスを呼び出す場合のクライアント側待ち時間（7 のポイント）**

  CORBA ネーミングサービスまたは J2EE アプリケーションへのアクセスで何かのトラブルが発生して制御が戻らなくなった場合に，タイムアウトによって EJB クライアントにエラーを通知します。

- **CTM または EJB クライアントから Enterprise Bean を呼び出す場合のクライアント側待ち時間（8 のポイント）**

  J2EE アプリケーションで，無限ループやデッドロックなど何らかのトラブルが発生した場合に，CTM のリソースを解放します。また，EJB クライアントにエラーを通知します。

ポイント

EJB クライアントから Enterprise Bean を呼び出す場合，タイムアウトは usrconf.properties またはアプリケーションサーバが提供する API（com.hitachi.software.ejb.ejbclient.RequestTimeoutConfig クラスのメソッド）に指定できます。
usrconf.properties の定義は，プロセス全体に影響します。API に指定したタイムアウトは，ビジネスメソッドを呼び出すスレッドまたはオブジェクトの範囲に影響します。また，API の指定は，usrconf.properties の定義よりも優先されます。
このため，プロセス全体に設定したい標準的な値を usrconf.properties に定義して，呼び出す業務によって細かく設定したい値は適宜 API を使用して設定することをお勧めします。

Enterprise Bean の呼び出しでタイムアウトが発生した場合，EJB クライアント側には，javax.rmi.RemoteException（org.omg.CORBA.TIMEOUT など）例外が通知されます。これによって，クライアント側のリクエストはキャンセルされます。ただし，この時点でサーバ側の Enterprise Bean の処理がすでに開始されていた場合は，そのまま処理が継続されます。このため，タイムアウトが発生した場合でも，サーバ側の処理は正常に終了することがあります。

● **EJB コンテナ上でのメソッド処理待ち時間（9 のポイント）**

J2EE アプリケーションの実行時間監視機能を利用します。
「8.6.6　J2EE アプリケーションのメソッドタイムアウトを設定する」を参照してください。

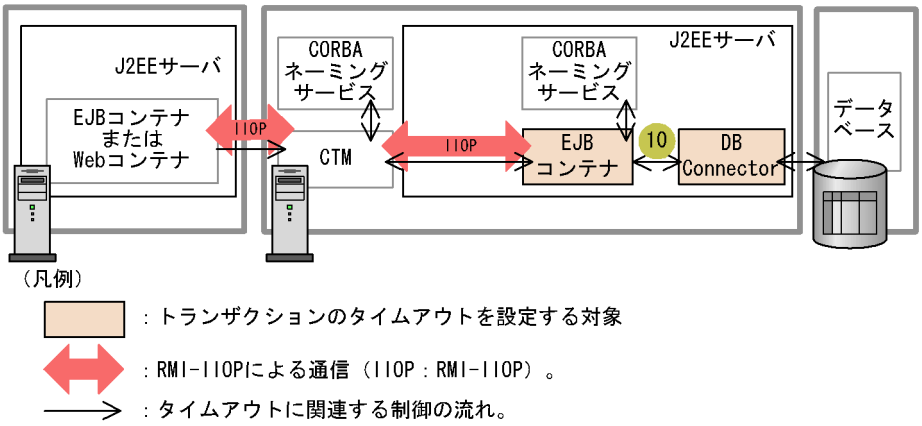
また，9 のポイントで，Stateless Session Bean の method-ready プールおよび Entity Bean の pool プールのインスタンスが時間内に取得できない場合も，タイムアウトが設定できます。この場合は，クライアントに java.rmi.RemoteException（リモートクライアントの場合）または javax.ejb.EJBException（ローカルクライアントの場合）が送出されます。

8.6.4　トランザクションタイムアウトを設定する

ここでは，トランザクションタイムアウトの設定について説明します。トランザクションタイムアウトは，データベースシステムなど EIS とのトランザクションに設定します。DB Connector を使用してデータベースにアクセスするときのトランザクションタイムアウトについて説明します。

トランザクションタイムアウトを設定する場合は，システム全体のタイムアウトのうち，**EJB コンテナとデータベースのトランザクション（図の 10 のポイント）**について意識する必要があります。なお，この番号は，図 8-9 または図 8-10 と対応しています。

図 8-13　EIS とのトランザクションで意識するタイムアウトのポイント



トランザクションタイムアウトが発生すると，アプリケーションサーバによって，次の処理が実行されます。

- 実行中のトランザクションはロールバックされます。
- トランザクションに参加しているコネクションはクローズされ，コネクションプールから削除されます。

ポイント

トランザクションのタイムアウトは，CMT の場合と BMT の場合で設定方法が異なります。

- **CMT の場合**
  CMT の場合，トランザクションのタイムアウトは，usrconf.properties で定義するか，または Enterprise Bean，インタフェース，もしくはメソッドの属性として設定できます。Enterprise Bean，インタフェースまたはメソッドの属性は，サーバ管理コマンドで設定します。
  usrconf.properties の定義は，プロセス全体に影響します。Enterprise Bean，インタフェースまたはメソッドの属性に設定したタイムアウトは，該当する Enterprise Bean，インタフェースまたはメソッドが使用するトランザクションの範囲だけに影響します。また，この指定は，usrconf.properties の定義よりも優先されます。
  このため，プロセス全体に設定したい標準的な値を usrconf.properties に定義して，呼び出す業務によって細かく設定したい値は Enterprise Bean，インタフェース，またはメソッドの属性として設定することをお勧めします。
- **BMT の場合**
  BMT の場合，トランザクションのタイムアウトは，usrconf.properties または JTA の API（javax.transaction.UserTransaction#setTransactionTimeout メソッド）に指定できます。
  usrconf.properties の定義は，プロセス全体に影響します。API に指定したタイムアウトは，API を発行したトランザクションの範囲だけに影響します。また，API の指定は，usrconf.properties の定義よりも優先されます。
  このため，プロセス全体に設定したい標準的な値を usrconf.properties に定義して，呼び出す業務によって細かく設定したい値は適宜 API を使用して設定することをお勧めします。

トランザクションタイムアウトが発生した場合，ユーザアプリケーションに例外は通知されません。ただし，メッセージ KDJE31002-W がログファイルと J2EE サーバのコンソールに出力されます。また，トランザクションタイムアウトが発生したあとで，ユーザアプリケーションから該当するトランザクションを使用して JTA インタフェースまたは JDBC インタフェースを使用しようとすると，例外が通知されます。

8.6.5 データベースでのタイムアウトを設定する

ここでは，次に示すデータベースでのタイムアウトの設定について説明します。

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

なお，Oracle の場合は，グローバルトランザクションとローカルトランザクションのどちらを使用しているかによって，設定できる項目が異なります。

（1） HiRDB のタイムアウト

HiRDB では，次の 3 種類のタイムアウトを設定できます。

● ロック解放待ちタイムアウト

デッドロックやリソースの長時間占有を防止するために設定するタイムアウトです。HiRDB サーバのシステム共通定義の pd_lck_wait_timeout パラメタに設定します。ここで設定するタイムアウト時間は，排他待ち時間を監視する最大時間です。排他待ち時間とは，排他要求が待ち状態になってから解除されるまでの時間です。

このタイムアウトが発生した場合にアプリケーションサーバおよび HiRDB によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す HiRDB のメッセージ KFPA11770-I が出力されます。または，デッドロックが発生したことを示す HiRDB のメッセージ KFPA11911-E が出力されます。
- 実行中のトランザクションはロールバックされます。
- ユーザアプリケーションのビジネスメソッド終了後に，コネクションはクローズされ，コネクションプールから削除されます。

● レスポンスタイムアウト

データベースシステムのサーバ側の障害を検知するためのタイムアウトです。HiRDB のクライアント環境変数の PDCWAITTIME に設定します。ここで設定するタイムアウト時間は，HiRDB クライアントから HiRDB サーバに要求をしてから，応答が戻ってくるまでの HiRDB クライアントの最大待ち時間です。長時間 SQL の時間を監視する場合などに指定します。

このタイムアウトが発生した場合にアプリケーションサーバおよび HiRDB によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す HiRDB のメッセージ KFPA11732-E が出力されます。
- 実行中のトランザクションはロールバックされます。
- コネクションはクローズされて，コネクションプールから削除されます。

● リクエスト間隔タイムアウト

データベースシステムのクライアント側の障害を検知するためのタイムアウトです。HiRDB のクライアント環境変数の PDSWAITTIME に設定します。ここで設定するタイムアウトは，HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が送信されるまでの HiRDB サーバの最大待ち時間です。時間監視は，トランザクションの処理中（SQL 実行開始からコミットまたはロールバックまでの間）が対象になります。HiRDB クライアントからの要求が HiRDB サーバに到着した段階でリセットされます。

このタイムアウトが発生した場合にアプリケーションサーバおよび HiRDB によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す HiRDB のメッセージ KFPA11723-E が出力されます。
- 実行中のトランザクションはロールバックされます。
- コネクションはクローズされて，コネクションプールから削除されます。

（2） Oracle のタイムアウト（ローカルトランザクションの場合）

Oracle でローカルトランザクションを使用している場合は，次のタイムアウトを設定できます。

● クエリータイムアウト

クエリータイムアウトは，JDBC ドライバとして Oracle JDBC Thin Driver を使用する場合だけ設定できるタイムアウトです。java.sql.Statement クラスの setQueryTimeout メソッドを利用してタイムアウトを設定します。ただし，このタイムアウトが有効になるのは，MTS 接続の場合だけです。専用接続では無効です。詳細はオラクルのサポートサービスにお問い合わせください。

なお，デッドロックが発生した場合は，Oracle のメッセージ ORA-00060 が出力されます。また，アプリケーションサーバによってユーザアプリケーションのビジネスメソッド終了後にコネクションがクローズされ，コネクションプールから削除されます。

（3） Oracle のタイムアウト（グローバルトランザクションの場合）

Oracle でグローバルトランザクションを使用している場合は，次のタイムアウトを設定できます。

● クエリータイムアウト

クエリータイムアウトについては，「(2) Oracle のタイムアウト（ローカルトランザクションの場合）」のクエリータイムアウトの説明を参照してください。

● ロック解放待ちタイムアウト

デッドロックやリソースの長時間占有を防止するために設定するタイムアウトです。Oracle のサーバ定義の DISTRIBUTED_LOCK_TIMEOUT パラメタに設定します。このタイムアウトが発生した場合にアプリケーションサーバおよび Oracle によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す Oracle のメッセージ ORA-02049 が出力されます。または，デッドロックが発生したことを示す Oracle のメッセージ ORA-00060 が出力されます。
- ユーザアプリケーションのビジネスメソッド終了後に，コネクションはクローズされ，コネクションプールから削除されます。

なお，実行中のトランザクションはロールバックされません。

（4） SQL Server のタイムアウト

SQL Server では，次の 2 種類のタイムアウトを設定できます。

● メモリ取得待ちタイムアウト

SQL 実行時のメモリ取得待ち時間を監視するために設定するタイムアウトです。SQL Server の環境設定オプションの query wait パラメタに設定します。ここで設定するタイムアウト時間は，SQL の実行に必要なメモリが得られなかったときのメモリ取得待ち時間です。

このタイムアウトが発生した場合にアプリケーションサーバおよび SQL Server によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す SQL Server のメッセージ 8645 が出力されます。
- 実行中のトランザクションはロールバックされます。
- ユーザアプリケーションプログラムのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

● ロック解放待ちタイムアウト

デッドロックやリソースの長時間占有を防止するために設定するタイムアウトです。SQL Server の SET LOCK_TIMEOUT ステートメントを実行することで設定します。ここで設定するタイムアウト時間は，ロックが解除されるまでの待ち時間です。

このタイムアウトが発生した場合にアプリケーションサーバおよび SQL Server によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す SQL Server のメッセージ 1222 が出力されます。
- ユーザアプリケーションプログラムのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

また，SQL Server でデッドロックが発生した場合にアプリケーションサーバおよび SQL Server によって実行される動作は次のとおりです。

- ユーザアプリケーションに例外（java.sql.SQLException）が通知されます。
- デッドロックが発生したことを示す SQL Server のメッセージ 1205 が出力されます。
- 実行中のトランザクションはロールバックされます。
- ユーザアプリケーションプログラムのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

（5） XDM/RD E2 のタイムアウト

XDM/RD E2 では，次の 5 種類のタイムアウトを設定できます。

● ロック解放待ちタイムアウト

デッドロックやリソースの長時間占有を防止するために設定するタイムアウトです。

XDM/BASE のシステムオプション定義の TIMER パラメタに設定します。ここで設定するタイムアウト時間は，排他待ち時間を監視する最大時間です。排他待ち時間とは，排他要求が待ち状態になってから解除されるまでの時間です。
このタイムアウトが発生した場合に，アプリケーションサーバおよび XDM/RD E2 によって実行される動作は次のとおりです。

- J2EE アプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトまたはデッドロックが発生したことを示す XDM/RD E2 のメッセージ JXZ1911I が出力されます。
- 実行中のトランザクションはロールバックされます。
- J2EE アプリケーションのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

また，XDM/RD E2 でデッドロックが発生したときの動作は，ロック解放待ちタイムアウトが発生したときと同じになります。

● SQL 実行 CPU 時間タイムアウト

SQL 実行時の CPU 処理時間を監視するために設定するタイムアウトです。DB コネクションサーバのコントロール空間起動制御文またはサーバ空間起動制御文の SQLCTIME パラメタに設定します。ここで設定するタイムアウト時間は，一つの SQL を実行したときの CPU 処理時間を監視する最大時間です。長時間 SQL の時間を監視する場合などに指定します。
このタイムアウトが発生した場合にアプリケーションサーバおよび XDM/RD E2 によって実行される動作は次のとおりです。

- J2EE アプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示すメッセージが出力されます。メッセージは，DB コネクションサーバのコントロール空間起動制御文に指定した VPARTOPTION パラメタの指定値によって変わります。指定を省略するか ERROR NORMAL を指定した場合には，HiRDB クライアントのメッセージ KFPA11723-E が出力されます。それ以外の値を指定した場合には，XDM/RD E2 のメッセージ JXZ1874I が出力されます。
- 実行中のトランザクションはロールバックされます。
- VPARTOPTION パラメタの指定を省略するか ERROR NORMAL を指定した場合には，コネクションはクローズされ，コネクションプールから削除されます。それ以外の値を指定した場合には，タイムアウト発生後の初回のデータベースアクセス時，または J2EE アプリケーションのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

● SQL 実行経過時間タイムアウト

SQL 実行時の経過時間を監視するために設定するタイムアウトです。DB コネクションサーバのコントロール空間起動制御文またはサーバ空間起動制御文の SQLETIME パラメタに設定します。ここで設定するタイムアウト時間は，一つの SQL を実行したときの経過時間を監視する最大時間です。長時間 SQL の時間を監視する場合などに指定します。

このタイムアウトが発生した場合にアプリケーションサーバおよび XDM/RD E2 によって実行される動作は次のとおりです。

- J2EE アプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示すメッセージが出力されます。メッセージは，DB コネクションサーバのコントロール空間起動制御文に指定した VPARTOPTION パラメタの指定値によって変わります。指定を省略するか，ERROR NORMAL または ERROR SQLCTIME を指定した場合には，HiRDB クライアントのメッセージ KFPA11723-E が出力されます。それ以外の値を指定した場合には，XDM/RD E2 のメッセージ JXZ1874I が出力されます。
- 実行中のトランザクションはロールバックされます。
- VPARTOPTION パラメタの指定を省略するか，ERROR NORMAL または ERROR SQLCTIME を指定した場合には，コネクションはクローズされ，コネクションプールから削除されます。それ以外の値を指定した場合には，タイムアウト発生後の初回のデータベースアクセス時，または J2EE アプリケーションのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

● トランザクション経過時間タイムアウト

トランザクションの開始時点からの経過時間を監視するために設定するタイムアウトです。DB コネクションサーバのコントロール空間起動制御文またはサーバ空間起動制御文の SVETIME パラメタに設定します。ここで設定するタイムアウト時間は，トランザクションの経過時間を監視する最大時間です。

このタイムアウトが発生した場合にアプリケーションサーバおよび XDM/RD E2 によって実行される動作は次のとおりです。なお，タイムアウトが発生したときに SQL を実行していた場合には，その時点で実行されます。SQL を実行していなかった場合には，タイムアウト発生後の SQL 実行時に実行されます。

- J2EE アプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示すメッセージが出力されます。メッセージは，DB コネクションサーバのコントロール空間起動制御文に指定した VPARTOPTION パラメタの指定値によって変わります。指定を省略するか，ERROR NORMAL または ERROR SQLCTIME を指定した場合には，HiRDB クライアントのメッセージ KFPA11723-E が出力されます。それ以外の値を指定した場合には，XDM/RD E2 のメッセージ JXZ1874I が出力されます。
- 実行中のトランザクションはロールバックされます。
- VPARTOPTION パラメタの指定を省略するか，ERROR NORMAL または ERROR SQLCTIME を指定した場合には，コネクションはクローズされ，コネクションプールから削除されます。それ以外の値を指定した場合には，タイムアウト発生後の初回のデータベースアクセス時，または J2EE アプリケーションのビジネスメソッドの終了後に，コネクションはクローズされ，コネクションプールから削除されます。

● レスポンスタイムアウト

データベースシステムのサーバ側の障害を検知するためのタイムアウトです。
HiRDB のクライアント環境変数の PDCWAITTIME に設定します。ここで設定するタイムアウト時間は，HiRDB クライアントから XDM/RD E2 サーバに要求をしてから，応答が戻ってくるまでの HiRDB クライアントの最大待ち時間です。長時間 SQL の時間を監視する場合などに指定します。
このタイムアウトが発生した場合にアプリケーションサーバおよび HiRDB によって実行される動作は次のとおりです。

- J2EE アプリケーションに例外（java.sql.SQLException）が通知されます。
- タイムアウトが発生したことを示す HiRDB クライアントのメッセージ KFPA11732-E が出力されます。
- 実行中のトランザクションはロールバックされます。
- コネクションはクローズされて，コネクションプールから削除されます。

(6) データベースへのアクセスでタイムアウトまたはデッドロックが発生した場合のユーザアプリケーションの処理

ユーザアプリケーションでデータベースのタイムアウトまたはデッドロックによる例外が発生した場合には，実行中のトランザクションをロールバックして，ビジネスメソッドの処理を中止してください。また，必要に応じてこの項で説明したタイムアウトパラメタを見直してください。

8.6.6 J2EE アプリケーションのメソッドタイムアウトを設定する

ここでは，J2EE アプリケーションのメソッドタイムアウトの設定について説明します。メソッドタイムアウトを設定することによって，ポイント 6 とポイント 9 で，Web コンテナまたは EJB コンテナ上の業務処理での無限ループが発生した場合などに，タイムアウトによって検知できるようになります。また，タイムアウトを検知したメソッドを強制的にキャンセル（**メソッドキャンセル**）することもできます。メソッドキャンセル機能の詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「5.3.2　J2EE アプリケーション実行時間の監視とは」を参照してください。

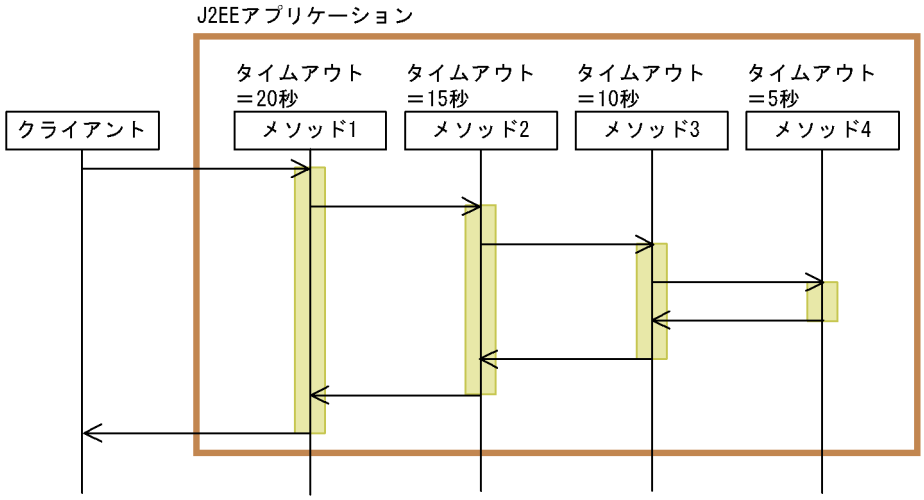
メソッドタイムアウトを設定する場合の考え方について説明します。

J2EE アプリケーション内でメソッドの呼び出しが入れ子になっている場合，タイムアウト値として，呼び出し元の方に大きな値を設定するように，メソッドの呼び出し順序を考慮して設定してください。

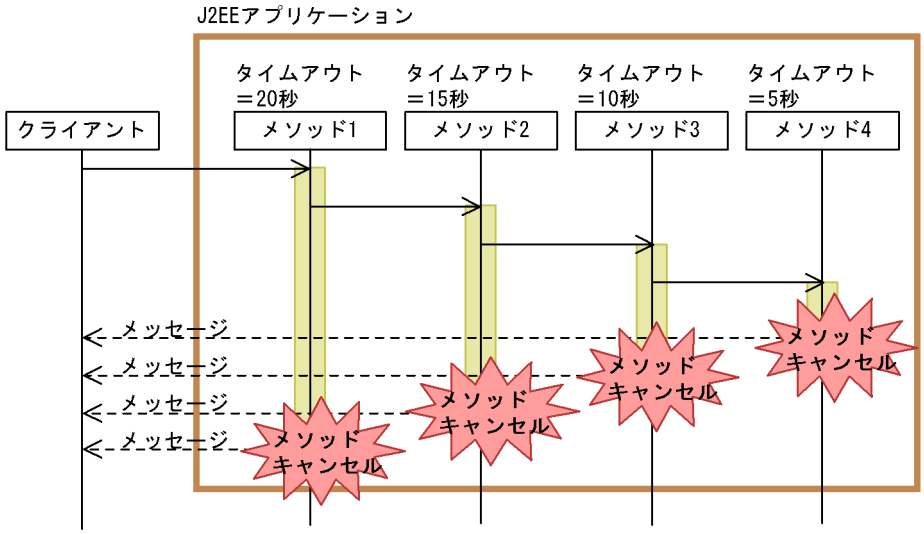
設定例を次に示します。

図 8-14　メソッドのタイムアウトの設定例

●通常の処理の流れ



●タイムアウトが発生した場合



（凡例）

：タイムアウトに関連する制御の流れ。

：メッセージの流れ。

この例では，呼び出し元に近いメソッドに大きな値を設定しています。これによって，メソッドのどこかでタイムアウトが発生した場合，クライアントから遠いメソッドから順番にタイムアウトが発生します。タイムアウトはメッセージで通知されます。設定によっては，このタイミングでメソッドキャンセルを自動実行できます。

リモート呼び出しを実行するメソッドに対してメソッドタイムアウトおよびメソッドキャンセルを設定した場合は，呼び出し順序に注意してください。メソッドキャンセル機能では，リモート呼び出し中のメソッドは保護区で実行中と判断されます。呼び出し元に近いメソッドで先にタイムアウトが発生した場合，メソッドはリモート呼び出し中であるため，メソッドキャンセルができません。例のように呼び出し元に近い方に大きい値を設定しておけば，タイムアウトは呼び出し元から遠い順に発生するため，タイムアウトが発生したメソッドがリモート呼び出しをしていることはありません。このため，確実にメソッドキャンセルを実行できます。

ローカル呼び出しを実行するメソッドに対してメソッドタイムアウトおよびメソッドキャンセルをする設定をした場合も，呼び出し元から遠い順にキャンセルされるようにすることで，タイムアウトが発生したメソッドとキャンセルが実行されたメソッドを一致させることができます。

タイムアウト値を設定できるメソッドについては，マニュアル「Cosminexus 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。

ローカル呼び出しのメソッドを入れ子で呼び出している場合の注意

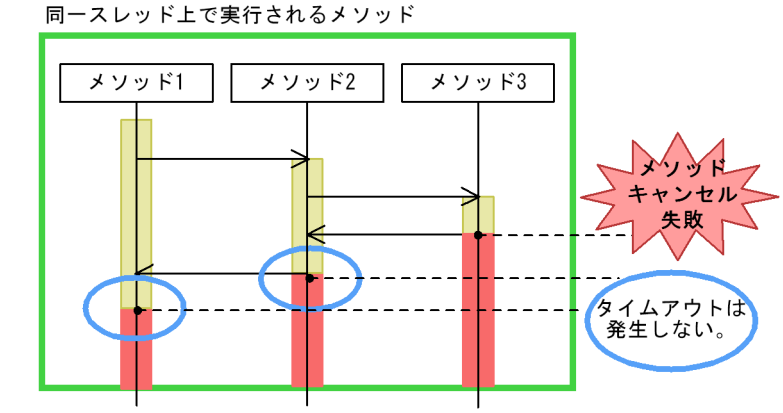
簡易構築定義ファイルの <param-name> タグに指定する ejbserver.rmi.localinvocation.scope に app または all を指定している場合，一つのメソッドから入れ子で呼び出されるローカル呼び出しのメソッドは，すべて同一スレッド上で実行されます。このとき，次の点に注意してください。

- 入れ子で呼び出されるメソッドのタイムアウトやメソッドキャンセルが失敗すると，そのメソッドが終了するまで同一スレッドのほかのメソッドではタイムアウトが発生しません。
- タイムアウトが発生したメソッドから入れ子で呼び出されているメソッドに対してメソッドキャンセルを実行した場合，コンテナによってキャンセルされるのは，メソッドキャンセルを実行した対象のメソッドだけです。タイムアウトが発生した呼び出し元のメソッドは，キャンセルの対象にはなりません。このため，キャンセルを実行したあとも，通常入れ子のメソッドの呼び出す場合と同様に，順次呼び出されたメソッドが終了していきます。なお，それらの順次呼び出されるメソッドもタイムアウト監視の対象になります。

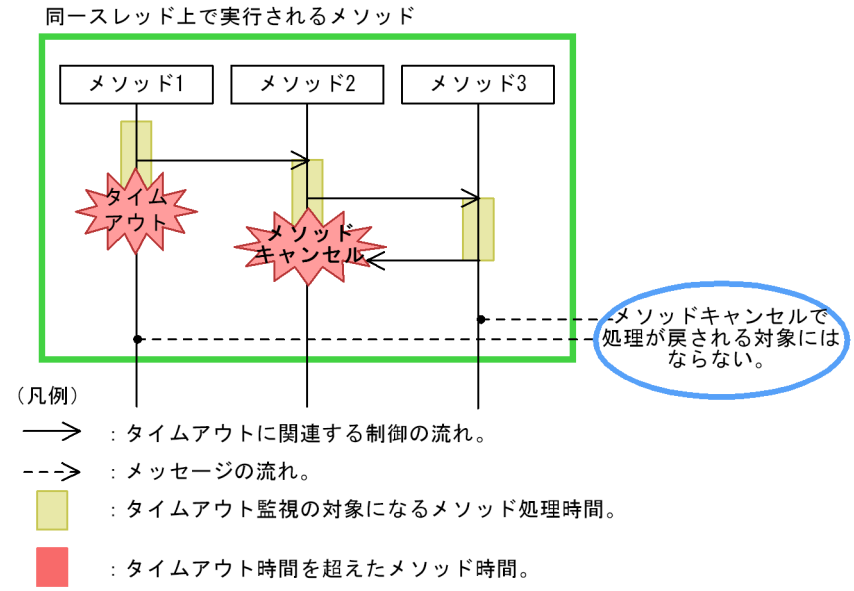
同一スレッド上でローカル呼び出しのメソッドを入れ子で呼び出している場合の注意を次の図に示します。

図 8-15　同一スレッド上でローカル呼び出しのメソッドを入れ子で呼び出している場合の注意

●入れ子で呼び出されるメソッドのタイムアウトやメソッドキャンセルが失敗した場合



●タイムアウトが発生したメソッドから入れ子で呼び出されているメソッドに対してキャンセルが実行された場合



（凡例）

：タイムアウトに関連する制御の流れ。

：メッセージの流れ。

：タイムアウト監視の対象になるメソッド処理時間。

：タイムアウト時間を超えたメソッド時間。

8.6.7　タイムアウトを設定するチューニングパラメタ

ここでは，タイムアウトの設定で使用するチューニングパラメタの設定方法についてまとめて示します。

（1） Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト

図 8-9 または図 8-10 のポイント 1 のタイムアウトを設定するチューニングパラメタです。使用する Web サーバによって設定個所が異なります。

Web サーバ連携の場合は，Web サーバ単位に設定します。ファイル編集によって設定します。

表 8-27　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ（Web サーバ連携の場合）

設定項目	設定個所
クライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト	httpsd.conf の Timeout ディレクティブ

注　Web Redirector を使用する場合などで，Web サーバとして Microsoft IIS を使用しているときには，isapi_redirect.conf の receive_client_timeout キーを編集してください。

インプロセス HTTP サーバの場合は，J2EE サーバ単位に設定します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-28　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ（インプロセス HTTP サーバの場合）

設定項目	設定対象	設定個所（パラメタ名）
クライアントからのリクエスト受信のタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.receive_timeout
クライアントへのデータ送信のタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.inprocess_http.send_timeout

（2） リダイレクタ側で設定する Web コンテナへのデータ送信のタイムアウト

図 8-9 のポイント 2，およびポイント 3 のタイムアウトを設定するチューニングパラメタです。リダイレクタ側で設定するタイムアウトのチューニングパラメタについて説明します。なお，これらのチューニングパラメタは，Web サーバ連携の場合だけ指定できます。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-29　リダイレクタ側で設定するタイムアウトのチューニングパラメタ

ポイント	設定項目	設定対象	設定個所（パラメタ名）※
2	リクエスト送信時の Web コンテナに対するコネクション確立のタイムアウト	論理 Web サーバ（web-server）	JkConnectTimeout
3	リクエスト送信のタイムアウト	論理 Web サーバ（web-server）	JkSendTimeout

注※　Web Redirector を使用する場合などで，Web サーバとして Microsoft IIS を使用しているときには，isapi_redirect.conf の connect_timeout キーを編集してください。

（3）リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト

図 8-9 のポイント 4 のタイムアウトを設定するチューニングパラメタです。

リダイレクタのワーカ定義単位で設定します。リダイレクタ側で設定するタイムアウトのチューニングパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-30　リダイレクタ側で設定するタイムアウトのチューニングパラメタ

設定項目	設定対象	設定個所（パラメタ名）
レスポンスデータ待ちの通信タイムアウト	論理 Web サーバ（web-server）	worker.< ワーカ名 >.receive_timeout

Web サーバ連携の場合だけ指定できます。

（4）Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト

図 8-9 のポイント 5 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-31　Web コンテナ側で設定するタイムアウトのチューニングパラメタ

設定項目	設定対象	設定個所（パラメタ名）
リダイレクタからの応答待ちのタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.receive_timeout

Web サーバ連携の場合だけ指定できます。

（5） Web コンテナ側で設定するリダイレクタへのデータ受信のタイムアウト

図 8-9 のポイント 12 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-32　Web コンテナ側で設定するタイムアウトのチューニングパラメタ

設定項目	設定対象	パラメタ名
レスポンス送信処理のタイムアウト	論理 J2EE サーバ（j2ee-server）	webserver.connector.ajp13.send_timeout

Web サーバ連携の場合だけ指定できます。

（6） EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト

図 8-9 または図 8-10 のポイント 7 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位，EJB クライアントアプリケーション単位または API による呼び出し単位に設定します。

EJB クライアント側で設定するタイムアウトのチューニングパラメタ（RMI-IIOP 通信によるリモート呼び出し）を次の表に示します。

表 8-33 EJB クライアント側で設定するタイムアウトのチューニングパラメタ（RMI-IIOP 通信によるリモート呼び出し）

単位	設定方法	設定項目	設定個所
J2EE サーバ単位	Smart Composer 機能	クライアントとサーバ間の通信タイムアウト	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** ejbserver.rmi.request.timeout
EJB クライアントアプリケーション単位	ファイル編集または開始時に指定するシステムプロパティの指定		**定義ファイル（ファイル編集の場合）** usrconf.properties **パラメタ名** ejbserver.rmi.request.timeout キー
API 単位	API		**オブジェクト単位で設定する場合** RequestTimeoutConfig#setRequestTimeout(java.rmi.Remote obj, int sec) メソッド※ **スレッド単位で設定する場合** RequestTimeoutConfig#setRequestTimeout(int sec) メソッド※

注※ パッケージ名は com.hitachi.software.ejb.ejbclient です。

EJB クライアント側で設定するタイムアウトのチューニングパラメタ（ネーミングサービス呼び出し）を次の表に示します。

表 8-34 EJB クライアント側で設定するタイムアウトのチューニングパラメタ（ネーミングサービス呼び出し）

単位	設定方法	設定項目	設定個所
J2EE サーバ単位	Smart Composer 機能	ネーミングサービスとの通信タイムアウト時間	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** ejbserver.jndi.request.timeout
EJB クライアントアプリケーション単位	ファイル編集または開始時に指定するシステムプロパティの指定		**定義ファイル（ファイル編集の場合）** usrconf.properties **パラメタ名** ejbserver.jndi.request.timeout キー

(7) EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト

図 8-9 または図 8-10 のポイント 8 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位，EJB クライアントアプリケーション単位または API による呼び出し単位に設定します。

なお，このタイムアウトの設定値には，「(6) EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト」で指定した設定値と同じ値が引き継がれます。

(8) EJB コンテナ側で設定するデータベースのトランザクションタイムアウト（DB Connector を使用した場合）

図 8-9 または図 8-10 のポイント 10 のタイムアウトを設定するチューニングパラメタです。

J2EE サーバ単位，Enterprise Bean，インタフェース，メソッド単位（CMT の場合），または API による呼び出し単位（BMT の場合）に設定します。

トランザクションタイムアウトのチューニングパラメタを次の表に示します。

表 8-35　トランザクションタイムアウトのチューニングパラメタ

単位	設定方法	設定項目	設定個所
J2EE サーバ単位	Smart Composer 機能	トランザクションのトランザクションタイムアウト時間のデフォルト	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** ejbserver.jta.TransactionManager.defaultTimeOut
Enterprise Bean，インタフェース，メソッド単位（CMT の場合）	サーバ管理コマンドの cjsetappprop コマンド	トランザクションタイムアウト時間	**定義ファイル** Session Bean 属性ファイル，Entity Bean 属性ファイル，または Message-driven Bean 属性ファイル **パラメタ名** <ejb-transaction-timeout>
API 単位（BMT の場合）	API		UserTransaction#setTransactionTimeout メソッド※

注※　パッケージ名は javax.transaction です。

(9) データベースのタイムアウト

図 8-9 または図 8-10 のポイント 11 のタイムアウトを設定するチューニングパラメタで

す。

データベースのタイムアウトは，使用するデータベースの種類，およびサーバの動作モードによって異なります。なお，ここでは，DB Connector を使用して HiRDB，Oracle，SQL Server または XDM/RD E2 にアクセスする場合のタイムアウトの設定方法について説明します。

参考

Oracle を使用している場合，チューニングパラメタによってタイムアウトが設定できるのは，グローバルトランザクションを使用しているときだけです。ローカルトランザクションを使用している場合，パラメタによってタイムアウトを設定することはできません。ただし，メソッドで設定するクエリータイムアウトは，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のどちらでも設定できます。

(a) HiRDB のタイムアウトの設定

HiRDB サーバのシステム共通定義または HiRDB のクライアント環境変数に設定します。詳細は，マニュアル「HiRDB システム定義」またはマニュアル「HiRDB UAP 開発ガイド」を参照してください。

HiRDB のタイムアウトを設定するチューニングパラメタを次の表に示します。

表 8-36 HiRDB のタイムアウトを設定するチューニングパラメタ

タイムアウトの種類	設定個所	設定方法（パラメタ名）	設定内容
ロック解放待ちタイムアウト	HiRDB サーバのシステム共通定義	pd_lck_wait_timeout パラメタ	設定値は任意です。
レスポンスタイムアウト	HiRDB のクライアント環境変数	PDCWAITTIME	トランザクションタイムアウトの値よりも大きな値を指定します。
リクエスト間隔タイムアウト	HiRDB のクライアント環境変数	PDSWAITTIME	トランザクションタイムアウトの値よりも大きな値を指定します。

(b) Oracle のタイムアウトの設定（グローバルトランザクションを使用している場合）

Oracle のサーバ定義の DISTRIBUTED_LOCK_TIMEOUT パラメタに設定します。

なお，このほか，XAOpenString の SesTm パラメタの設定がタイムアウトに影響します。このパラメタは，チューニングできません。

(c) SQL Server のタイムアウトの設定

SQL Server の環境設定オプションのパラメタ，またはステートメントの実行によって設定します。

SQL Server のタイムアウトを設定するチューニングパラメタを次の表に示します。

表 8-37 SQL Server のタイムアウトを設定するチューニングパラメタ

タイムアウトの種類	設定個所	設定方法（パラメタ名／ステートメント名）	設定内容
メモリ取得待ちタイムアウト	SQL Server の環境設定オプション（SQL Server 2000 の場合）またはサーバー構成オプション（SQL Server 2005 または SQL Server 2008 の場合）	query wait パラメタ	設定値は任意です。
ロック解放待ちタイムアウト	－	SET LOCK_TIMEOUT ステートメント	設定値は任意です。

（凡例）
－：該当しない。

（d）XDM/RD E2 のタイムアウトの設定

XDM/BASE のシステムオプション定義，HiRDB のクライアント環境変数，または DB コネクションサーバのコントロール空間起動制御文／サーバ空間起動制御文に設定します。

XDM/RD E2 のタイムアウトを設定するチューニングパラメタを次の表に示します。

表 8-38 XDM/RD E2 のタイムアウトを設定するチューニングパラメタ

タイムアウトの種類	設定個所	設定方法（パラメタ名）	設定内容
ロック解放待ちタイムアウト	XDM/BASE のシステムオプション定義	TIMER	設定値は任意です。※1
SQL 実行 CPU 時間タイムアウト	DB コネクションサーバのコントロール空間起動制御文／サーバ空間起動制御文	SQLCTIME	トランザクションタイムアウトの値よりも大きな値を指定します。※2
SQL 実行経過時間タイムアウト	DB コネクションサーバのコントロール空間起動制御文／サーバ空間起動制御文	SQLETIME	トランザクションタイムアウトの値よりも大きな値を指定します。※2
トランザクション経過時間タイムアウト	DB コネクションサーバのコントロール空間起動制御文／サーバ空間起動制御文	SVETIME	トランザクションタイムアウトの値よりも大きな値を指定します。※2
レスポンスタイムアウト	HiRDB クライアント環境変数	PDCWAITTIME	トランザクションタイムアウトの値よりも大きな値を指定します。※3

注※1

詳細については，マニュアル「VOS3　データマネジメントシステム XDM E2 系 システム定義（XDM/BASE・SD・TM2）」を参照してください。

注※ 2

詳細については，マニュアル「VOS3　Database Connection Server」を参照してください。

注※ 3

詳細については，マニュアル「HiRDB XDM/RD E2 接続機能」を参照してください。

（10）J2EE アプリケーションのメソッドタイムアウト

図 8-9 または図 8-10 のポイント 6 とポイント 9 のタイムアウトを設定するチューニングパラメタです。

Web アプリケーション内または Enterprise Bean 内のメソッド単位にタイムアウトを設定する場合は，アプリケーションの属性として設定します。また，タイムアウトが発生した場合の動作についても，アプリケーションの属性として設定します。これらの項目は，サーバ管理コマンド（cjsetappprop）で設定します。

メソッドの実行時間のタイムアウトを設定するチューニングパラメタを次の表に示します。なお，ポイントごとに設定個所が異なります。

表 8-39　メソッド実行時間のタイムアウトを設定するチューニングパラメタ

対象になるポイント	タイムアウトの種類およびタイムアウト時の動作	設定個所
6	フィルタ，サーブレットまたは JSP のリクエスト処理メソッド	**定義ファイル** サーブレット属性ファイル **パラメタ名** <method-observation-timeout> ※
9	Enterprise Bean のリクエスト処理メソッド	**定義ファイル** Session Bean 属性ファイル，Entity Bean 属性ファイルまたは Message-driven Bean 属性ファイル **パラメタ名** <ejb-method-observation-timeout>
6 および 9	タイムアウト発生時のアプリケーション単位の動作	**定義ファイル** アプリケーション属性ファイル **パラメタ名** <method-observation-recovery-mode>

注※　メソッドキャンセルを実行する場合に「thread」を指定します。

8.7　Web アプリケーションの動作を最適化する

この節では，Web アプリケーションのパフォーマンスチューニングの方法について説明します。Web フロントシステムの場合に検討してください。

ここでは，次の 3 種類のチューニング方法について説明します。

- 静的コンテンツと Web アプリケーションの配置を切り分ける
- 静的コンテンツをキャッシュする
- リダイレクタを使用してリクエストを振り分ける（Web サーバ連携の場合）

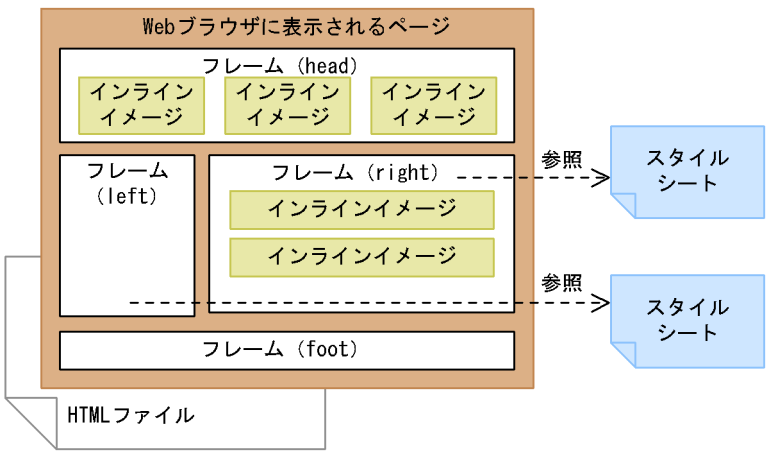
8.7.1　静的コンテンツと Web アプリケーションの配置を切り分ける

HTML ファイルや画像ファイルなど，クライアントからの要求に対する応答に使用するファイルのうち，リクエスト内容に影響されないで常に同じ内容になるコンテンツを，**静的コンテンツ**といいます。一方，サーブレットや JSP のように，クライアントからの要求に応じて動的に生成されるコンテンツを**動的コンテンツ**といいます。

ここでは，静的コンテンツを動的コンテンツである Web アプリケーションと切り分けて配置することで，パフォーマンスの向上を図る方法について説明します。設定方法は，Web サーバ連携の場合とインプロセス HTTP サーバを使用する場合とで異なります。

なお，それぞれの設定例では，Web ブラウザに，次の図に示すようなフレームやインラインイメージで構成される HTML ページを表示する場合の例を基に説明します。

図 8-16　静的コンテンツと動的コンテンツで構成される HTML ページの例



この構成の場合，次のファイルは動的に生成されない静的コンテンツです。

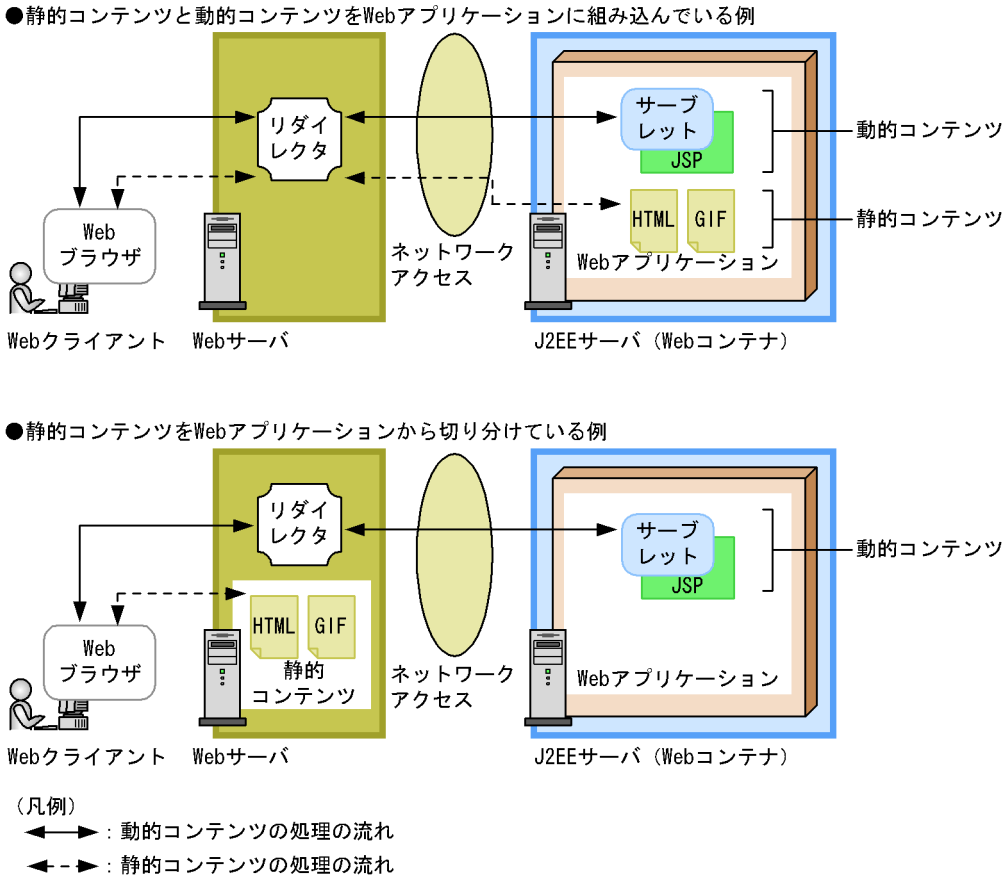
- スタイルシート（CSS ファイルなど）
- インラインイメージ（画像ファイル）
- フレームなどを定義した HTML ファイル

（1） Web サーバ連携の場合

静的コンテンツを Web アプリケーションに組み込んで扱うと，Web コンテナで処理する必要がない静的コンテンツをやり取りする場合でもリダイレクタと Web コンテナ間で常にアクセスが発生します。特に画像ファイルなどの場合は，データサイズが大きいため，処理時間が掛かります。

静的コンテンツは Web アプリケーションと分離して，Web サーバ上に配置することをお勧めします。これによって，ネットワークアクセスの回数およびやり取りするデータのサイズを減らし，パフォーマンスの向上が図れます。静的コンテンツと Web アプリケーションの処理のイメージを次の図に示します。

図 8-17　静的コンテンツと Web アプリケーションの処理のイメージ



Web アプリケーションと分離した静的コンテンツの配置方法について説明します。なお，

ここでは，図 8-16 で示した構成の HTML ページを例として説明します。

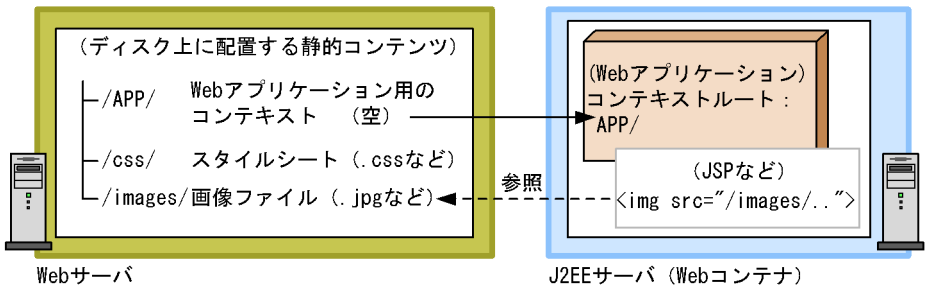
図 8-16 の場合，次のような静的コンテンツを Web サーバに配置することで，パフォーマンスの向上が図れます。

Web サーバに配置するコンテンツ

- スタイルシート（CSS ファイルなど）
- インラインイメージ（画像ファイル）
- フレームなどを定義した HTML ファイル

配置の例を次の図に示します。

図 8-18　静的コンテンツを Web サーバに配置する例（Web サーバ連携の場合）



/APP/がWebアプリケーションのコンテキストルートにマッピングされています。

このマッピングをするためには，マッピング定義を次のように記述します。詳細は，マニュアル「Cosminexus 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「9.　Web サーバ連携で使用するファイル」を参照してください。

Web サーバとして Hitachi Web Server を使用している場合（mod_jk.conf）

```
#
JkMount /APP/* worker1
# JkMount /* worker1
# を使用するとWebサーバ上の/images/などは参照できません。
```

Web サーバとして Microsoft IIS を使用している場合（uriworkermap.properties）

```
#
/APP/*=worker1
# /*=worker1
# を使用するとWebサーバ上の/images/などは参照できません。
```

（2） インプロセス HTTP サーバを使用する場合

インプロセス HTTP サーバを使用する場合も，静的コンテンツのように常に同じ内容をクライアントに渡すときには，静的コンテンツをインプロセス HTTP サーバおよび Web コンテナとは別の Web サーバで処理した方が，パフォーマンスが向上することがあります。特に，画像ファイルなどファイルサイズが大きいデータを大量に扱う場合は，Web アプリケーションに組み込むのではなく，静的コンテンツの処理専用の Web サーバを配

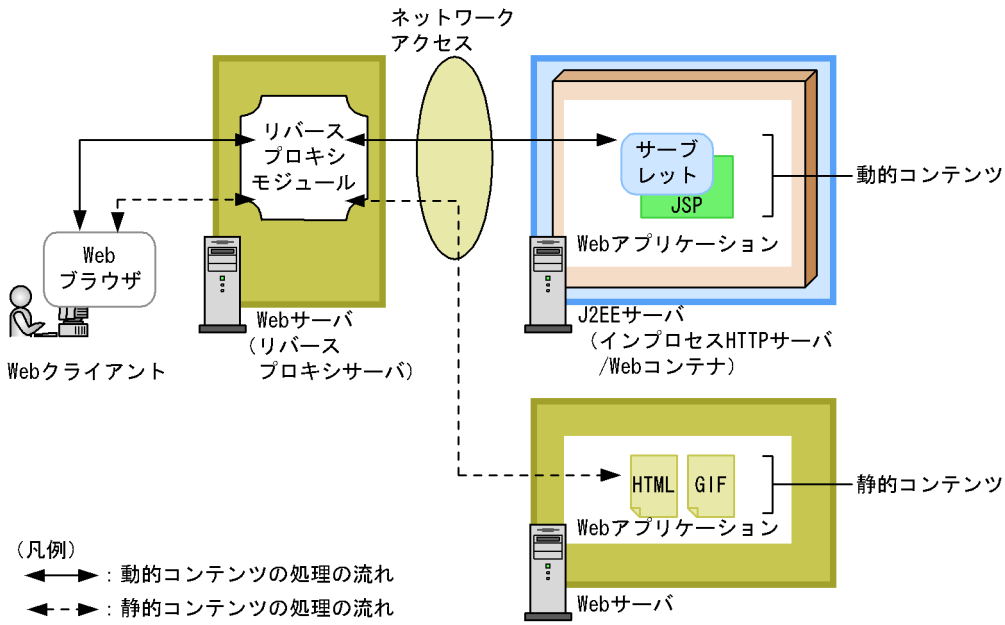
置して扱うことをお勧めします。

静的コンテンツを Web アプリケーションから切り出して扱うためには，フロントにリバースプロキシサーバや負荷分散機を配置する構成が考えられます。この場合，リバースプロキシサーバや負荷分散機によって，処理を次のように振り分けます。

- 動的コンテンツへのアクセスはインプロセス HTTP サーバおよび Web コンテナが動作する J2EE サーバにディスパッチします。
- 静的コンテンツへのアクセスは Web サーバにディスパッチします。

振り分け処理のイメージを次の図に示します。次の図は，リバースプロキシサーバによって処理を振り分けている例です。

図 8-19　リバースプロキシサーバによる動的コンテンツと静的コンテンツの振り分けイメージ



Web アプリケーションと分離した静的コンテンツの配置方法について説明します。なお，ここでは，図 8-16 で示した構成の HTML ページを例として説明します。

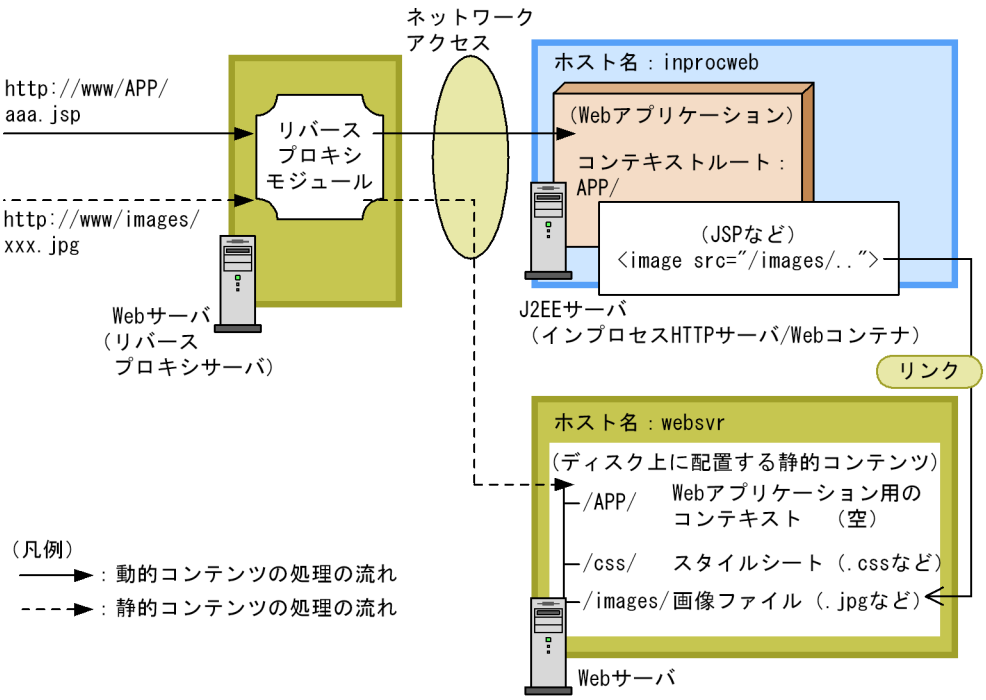
図 8-16 の場合，次のような静的コンテンツを Web サーバに配置することで，パフォーマンスの向上が図れます。

Web サーバに配置するコンテンツ

- スタイルシート（CSS ファイルなど）
- インラインイメージ（画像ファイル）
- フレームなどを定義した HTML ファイル

配置の例を次の図に示します。

図 8-20　静的コンテンツを Web サーバに配置する例（インプロセス HTTP サーバの場合）



この振り分けをするためには，Hitachi Web Server のコンフィグファイル（httpsd.conf）の ProxyPass ディレクティブに，次のようにリバースプロキシの定義を記述します。

詳細は，マニュアル「Hitachi Web Server」を参照してください。

リバースプロキシの定義

```
ProxyPass /APP/     http://inprocweb/APP/
ProxyPass /images/ http://websvr/images/
ProxyPass /css/      http://websvr/css/
```

8.7.2　静的コンテンツをキャッシュする

Web コンテナでは，静的コンテンツをメモリに保持（キャッシュ）できます。一度アクセスした静的コンテンツの内容をメモリに保持しておくことによって，2 回目以降のアクセス時のファイルシステムへのアクセス回数を減らし，応答速度の向上を図れます。

ここでは，静的コンテンツをキャッシュする場合に必要なチューニングの方法について説明します。

チューニングできるのは，次の項目です。

● 静的コンテンツをキャッシュするかどうかの選択

● 静的コンテンツのキャッシュに使用するメモリサイズの上限値

● キャッシュする静的コンテンツのファイルサイズの上限値

それぞれについて説明します。なお，これらの項目は，Web コンテナ単位および Web アプリケーション単位で設定できます。Web アプリケーション単位の設定は，Web コンテナ単位の設定よりも優先されます。このため，J2EE サーバ全体としてデフォルトの値を指定したい場合は Web コンテナ単位の値に設定して，細かな設定をしたい場合は適宜 Web アプリケーション単位に設定してください。

ポイント

メモリサイズの見積もりとの関係

静的コンテンツのキャッシュでは，メモリを使用して応答速度の向上を図ります。このため，この機能を使用する場合は，サーバで使用できるメモリ量に応じてチューニングする必要があります。

静的コンテンツのキャッシュで使用するメモリサイズは，Web アプリケーション単位に設定します。Web アプリケーション単位に設定したメモリサイズの合計値が，J2EE サーバ全体で静的コンテンツのキャッシュのために使用するメモリサイズの最大値になります。このため，この機能を使用する場合は，J2EE サーバで必要なメモリサイズを見積もる際に，Web アプリケーション単位に設定したメモリサイズの合計値を加算してください。

（1） 静的コンテンツをキャッシュするかどうかの選択

静的コンテンツのキャッシュは，デフォルトの設定では使用されない機能です。このため，使用する場合はパラメタを変更する必要があります。

なお，静的コンテンツのキャッシュの使用は，Web コンテナ単位および Web アプリケーション単位で設定でき，Web アプリケーション単位の設定は，Web コンテナ単位の設定よりも優先されます。ただし，Web コンテナ単位の設定で強制的な無効を選択している場合は，Web アプリケーション単位の設定も無効になります。強制的な無効は，次のような場合に使用できます。

● 静的コンテンツのキャッシュを有効にしたときと無効にしたときのメモリ使用量の差を検証したい場合

● Web アプリケーション単位の設定を保持した状態で静的コンテンツのキャッシュを一時的に無効にしたい場合

（2） 静的コンテンツのキャッシュに使用するメモリサイズの上限値

静的コンテンツをキャッシュするために，Web アプリケーション単位で使用するメモリサイズを設定できます。なお，それぞれの Web アプリケーションで設定した値よりもキャッシュの合計サイズが大きくなる場合は，アクセスされていない時間が最も長いキャッシュから削除されます。削除は，キャッシュの合計サイズが設定値以下になるまで繰り返されます。

メモリサイズの設定の指針を次に示します。

メモリサイズの設定の指針

- Web アプリケーション内に含まれる静的コンテンツのサイズの合計値以下の値を設定します。
- Web アプリケーションの内容によって最適な値は異なります。このため，値を設定してから静的コンテンツに対するリクエストの応答速度を測定して，高い効果が出る最適なキャッシュのサイズを見つける必要があります。

（3） キャッシュする静的コンテンツのファイルサイズの上限値

静的コンテンツのキャッシュの対象とするコンテンツのファイルサイズの上限を設定できます。上限を設定した場合，上限を超えるファイルサイズのコンテンツについてはキャッシュされないで，使用するたびに毎回ファイルシステムから取得されます。

ファイルサイズの設定の指針を次に示します。

ファイルサイズの設定の指針

- Web アプリケーション内に含まれる静的コンテンツ中の，ファイルサイズが最大であるコンテンツのファイルサイズ以下の値を設定します。
- 大きなサイズの静的コンテンツがキャッシュされることによって，アクセス頻度の高いほかの静的コンテンツのキャッシュが破棄されないように留意して，値を設定します。

（4） 静的コンテンツの稼働状況の確認

静的コンテンツの稼働状況は，Web アプリケーションを停止したときに出力されるメッセージ KDJE39234-I の出力内容で確認できます。キャッシュされている静的コンテンツの合計サイズやコンテンツの個数などが出力されるので，必要に応じて各パラメタをチューニングしてください。

8.7.3 リダイレクタによってリクエストを振り分ける

リダイレクタでは，複数の Web コンテナに処理を振り分けるためのマッピングを定義できます。これによって，負荷分散を図り，個々の Web コンテナの負荷を減らすことで，パフォーマンスの向上が図れます。なお，この機能は，Web サーバ連携の場合にだけ使用できます。インプロセス HTTP サーバを使用する場合は使用できません。

なお，リダイレクタによってリクエストを振り分ける場合，リダイレクタによってセッション情報に応じて振り分けられるため，Web アプリケーションのセッションに応じたリクエストの振り分けを考慮する必要はありません。これは，ある Web コンテナ上の Web アプリケーションでセッションが生成されると，そのあとはリダイレクタによってクライアントからのリクエストに含まれるセッション情報が調べられ，クライアントからのリクエストに対応するセッションが生成されている Web コンテナに振り分けられるためです。

Web サーバ（リダイレクタ）によるリクエストの振り分けの詳細については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 基本・開発編 (Web コンテナ)」の「3.2　Web サーバ（リダイレクタ）によるリクエストの振り分け」を参照してください。

8.7.4　Web アプリケーションの動作を最適化するためのチューニングパラメタ

ここでは，Web アプリケーションの動作を最適化するために使用するチューニングパラメタの設定方法についてまとめて示します。

（1）静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ

静的コンテンツと Web アプリケーションの配置の切り分けは，Web サーバの動作を定義するファイルのパラメタとして指定します。設定個所，ファイルおよびパラメタは，使用する Web サーバの種類によって異なります。

Web サーバ連携で Hitachi Web Server を使用している場合は，リダイレクタモジュールを使用して切り分けます。インプロセス HTTP サーバを使用している場合は，リバースプロキシサーバ（Hitachi Web Server）に配置しているリバースプロキシモジュールを使用して切り分けます。

設定方法および設定個所を次に示します。

表 8-40　静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ

使用する Web サーバ	設定方法	設定個所
Hitachi Web Server（リダイレクタモジュールを使用した切り分け※ 1）	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 Web サーバ（web-server） **パラメタ名** JkMount
インプロセス HTTP サーバ（リバースプロキシモジュールを使用した切り分け）	ファイル編集	**定義ファイル** httpsd.conf **設定対象** リバースプロキシサーバ **パラメタ名** ProxyPass ディレクティブ※ 2

注※ 1　Web Redirector を使用する場合などに，Web サーバとして Microsoft IIS を使用しているときは，uriworkermap.properties で設定します。

注※ 2　httpsd.conf の詳細については，マニュアル「Hitachi Web Server」を参照してください。

（2）静的コンテンツをキャッシュするためのチューニングパラメタ

静的コンテンツをキャッシュするためのチューニングパラメタについて説明します。これらのチューニングパラメタは，Web コンテナ単位または Web アプリケーション単位に設定します。

Web コンテナ単位に設定するチューニングパラメタの設定方法について，次の表に示します。これらの項目は，Smart Composer 機能で設定します。

表 8-41　静的コンテンツをキャッシュするためのチューニングパラメタ（Web コンテナ単位で設定する項目）

設定項目	設定個所
静的コンテンツのキャッシュを使用するかどうかの選択	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.static_content.cache.enabled
Web アプリケーション単位のメモリサイズの上限値の設定	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.static_content.cache.size
キャッシュする静的コンテンツのファイルサイズの上限値の設定	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.static_content.cache.filesize.threshold

Web アプリケーション単位に設定するチューニングパラメタについて示します。Web アプリケーション単位に設定する項目は，web.xml を直接編集するか，サーバ管理コマンドを使用して設定します。デプロイ前の Web アプリケーションに設定する場合は，web.xml を編集してください。デプロイ後の Web アプリケーションに設定する場合は，サーバ管理コマンド（cjsetappprop）を使用してください。

設定内容を次の表に示します。

表 8-42　静的コンテンツをキャッシュするためのチューニングパラメタ（Web アプリケーション単位で設定する項目）

設定項目	設定内容※
静的コンテンツのキャッシュを使用するかどうかの選択	<param-name> タグ com.hitachi.software.web.static_content.cache.enabled <param-value> タグ （設定値）

設定項目	設定内容※
Web アプリケーション単位のメモリサイズの上限値の設定	<param-name> タグ com.hitachi.software.web.static_content.cache.size <param-value> タグ （設定値）
キャッシュする静的コンテンツのファイルサイズの上限値の設定	<param-name> タグ com.hitachi.software.web.static_content.cache.filesize.threshold <param-value> タグ （設定値）

注

（設定値）に設定できる値の詳細については，マニュアル「Cosminexus 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「2.21.2　DD での定義（Web アプリケーション単位での設定)」を参照してください。

注※

web.xml を直接編集する場合，<web-app> タグ内に <context-param> タグを追加して，<context-param> タグ内に <param-name> タグおよび <param-value> タグを追加します。サーバ管理コマンドを使用する場合，WAR 属性ファイルの <hitachi-war-property> タグ内に <context-param> タグを追加して，<context-param> タグ内に <param-name> タグおよび <param-value> タグを追加します。

(3) リダイレクタによってリクエストを振り分けるためのチューニングパラメタ

リダイレクタによってリクエストを振り分けるためのチューニングパラメタは，Web サーバの動作を定義するファイルのパラメタとして指定します。

なお，この定義は，Web サーバ連携の場合だけできます。インプロセス HTTP サーバを使用している場合は定義できません。

設定方法および設定個所を次に示します。

表 8-43　リダイレクタによってリクエストを振り分けるためのチューニングパラメタ

設定項目	設定方法	設定個所
URL パターンのマッピング定義※	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** JkMount

注※　Web Redirector を使用する場合などに，Web サーバとして Microsoft IIS を使用しているときは，uriworkermap.properties で設定します。

8.8 CTM の動作を最適化する

CTM を使用したシステムのパフォーマンスチューニングの方法について説明します。CTM を使用したバックシステムの場合に検討してください。

ここでは，次の 4 種類のチューニング方法について説明します。

- CTM ドメインマネジャおよび CTM デーモンの稼働状態の監視間隔をチューニングする
- 負荷状況監視間隔をチューニングする
- CTM デーモンのタイムアウト閉塞を設定する
- CTM で振り分けるリクエストの優先順位を設定する

8.8.1 CTM ドメインマネジャおよび CTM デーモンの稼働状態の監視間隔をチューニングする

システムに存在する複数の CTM ドメインマネジャ間，および CTM ドメイン内の複数の CTM デーモン間では，お互いの稼働状態を監視するために，定期的に通信処理が実行されています。

ここでは，それぞれの通信処理間隔をチューニングする場合の考え方について説明します。

（1）CTM ドメインマネジャ間で稼働状態を監視する間隔のチューニング

CTM ドメインマネジャ間では，お互いのホストにある CTM デーモンの情報を定期的に交換しています。この情報を基に，自ホストで受け付けたリクエストを適宜ほかのホストの CTM デーモンに振り分けています。

情報を交換するタイミングでは，お互いが稼働状態であるかどうかも同時に確認します。相手のホストの CTM ドメインマネジャが停止している場合は，そのホストにはリクエストを振り分けないようにします。CTM ドメインマネジャでは，情報を交換する間隔に一定の係数を掛けた時間以上応答がない場合に，相手の CTM ドメインマネジャが停止していると判断します。

情報を交換する間隔の設定個所は，相手の 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か，異なるネットワークセグメントにあるかによって異なります。稼働状態を判断するときに使用する係数のデフォルトは 2 です。例えば，同じネットワークセグメント内にある CTM ドメインマネジャの稼働情報を確認する場合，Management Server を使用して構築したシステムでは，CTM ドメイン構成情報の送信間隔に 2 を掛けた時間以上応答がない場合は，停止していると判断します。CTM ドメイン構成情報の送信間隔が 60 秒の場合は，120 秒間応答がない場合に，停止していると判断します。

この係数を変更することによって，通信処理の最適化が図れます。

係数に指定する値は，通信によって発生する処理の負荷を考慮して，適切な値を検討してください。基準になる送信間隔も必要に応じて検討してください。係数を小さくすると，間隔が短くなり，相手の CTM ドメインマネジャが停止したことがすぐに検知できます。これによって，CTM デーモンが停止しているホストにリクエストを送信してしまうことを防げます。ただし，間隔が短過ぎると，通信処理が多く発生し，通信負荷が掛かります。

(2) CTM デーモン間で稼働状態を監視する間隔のチューニング

CTM デーモン間では，CTM ドメインマネジャ間でやり取りした情報を基に，相互にリクエストの振り分け処理をしています。

リクエストの振り分け先の CTM デーモンから一定時間応答がない場合，振り分け元の CTM デーモンでは，相手の CTM デーモンが停止していると判断して，以降のリクエストを振り分けないようにします。

デフォルトの設定では，CTM デーモンは 180 秒間応答を待ちます。180 秒以上応答がない場合，相手の CTM デーモンが停止していると判断します。この値を変更することで，不要な処理待ち時間を短縮できます。

値は，リクエストとして送信するデータの大きさなど考慮して，適切な値を設定してください。間隔を短くすることで，相手のホストのトラブルを迅速に検知できるため，トラブルの影響が少ない状態でシステムから切り離すことができるようになります。ただし，間隔が短過ぎると，大きなサイズのデータを転送している間にタイムアウトが発生してしまうおそれがあります。

8.8.2 負荷状況監視間隔をチューニングする

CTM ドメイン内の複数の CTM デーモン間では，それぞれのスケジュールキューの負荷情報を監視しています。監視結果は，CTM デーモン間のリクエスト振り分け時に利用されます。

負荷状況監視間隔は，チューニングできます。なお，デフォルトの状態では 10 秒です。

負荷状況監視間隔を短くすることで，その時点の状況に応じた振り分けができるようになります。ただし，短くし過ぎると，通信が多発して，負荷が高くなります。

なお，負荷状況監視間隔に 0 を設定した場合は，J2EE サーバ起動時の負荷状況を一度だけ収集して，その値を使用し続けます。

8.8.3 CTM デーモンのタイムアウト閉塞を設定する

CTM デーモンに対応する J2EE サーバにトラブルが発生した場合，CTM デーモンが送信したリクエストでタイムアウトが発生します。そのままの状態で運用を続けると，

CTM デーモンはトラブルが発生した J2EE サーバに対してリクエストを送信し続けるため，そのつどリクエストでタイムアウトが発生してしまいます。

これに対処するために，CTM デーモンには，タイムアウト閉塞を設定できます。タイムアウト閉塞とは，一定時間内に規定回数以上のリクエストのタイムアウトが発生した場合に，CTM デーモンのスケジュールキューを閉塞する機能です。これによって，トラブルが発生した CTM デーモンでそれ以上リクエストを受け付けなくなり，ほかの CTM デーモンが受け付けるようになります。リクエストは正常に稼働している J2EE サーバに振り分けられるようになります。

8.8.4 CTM で振り分けるリクエストの優先順位を設定する

CTM で制御するリクエストには，優先順位が付けられます。すぐに実行する必要があるリクエストに高い優先順位を設定しておくことで，スケジュールキューの中で滞留することなく，迅速な処理ができるようになります。

リクエストの優先順位は，CTM にリクエストを送信する，EJB クライアントアプリケーション，J2EE サーバまたは Web コンテナサーバに対して設定できます。優先順位を高く設定した EJB クライアントアプリケーション，J2EE サーバまたは Web コンテナサーバから送信されたリクエストは，スケジュールキューに格納されているほかのクライアントからのリクエストよりも優先されて処理されます。

8.8.5 CTM の動作を最適化するチューニングパラメタ

ここでは，CTM の動作の最適化で使用するチューニングパラメタの設定方法についてまとめて示します。

（1）CTM ドメインマネジャおよび CTM デーモンの稼働状態の監視間隔を設定するチューニングパラメタ

CTM ドメインマネジャの稼働状態の監視間隔をチューニングするパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

なお，監視間隔は，送信間隔×係数の値になります。

表 8-44 CTM ドメインマネジャの稼働状態の監視間隔をチューニングするパラメタ

対象	設定項目	設定対象	設定個所（パラメタ名）
同じネットワークセグメント内にある CTM ドメインマネジャ	送信間隔	論理 CTM ドメインマネジャ（ctm-domain-manager）	cdm.SendInterval
	係数	論理 CTM ドメインマネジャ（ctm-domain-manager）	cdm.AliveCheckCount
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	送信間隔	論理 CTM ドメインマネジャ（ctm-domain-manager）	cdm.SendHostInterval
	係数	論理 CTM ドメインマネジャ（ctm-domain-manager）	cdm.AliveCheckCount

CTM デーモンの稼働状態の監視間隔をチューニングするパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-45 CTM デーモンの稼働状態の監視間隔をチューニングするパラメタ

設定項目	設定対象	設定個所（パラメタ名）
CTM デーモン間転送時のタイムアウト	論理 CTM	ctm.DCSendTimeOut

(2) 負荷状況監視間隔を設定するチューニングパラメタ

負荷状況監視間隔をチューニングするパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-46 負荷情報監視間隔をチューニングするパラメタ

設定項目	設定対象	設定個所（パラメタ名）
CTM デーモン間転送時のタイムアウト	論理 CTM	ctm.LoadCheckInterval

(3) CTM デーモンのタイムアウト閉塞を設定するチューニングパラメタ

タイムアウト閉塞は，タイムアウト発生回数と監視間隔を設定しておくことによって，実行されます。

CTM デーモンのタイムアウト閉塞をチューニングするパラメタについて説明します。

次の表に示す項目は，Smart Composer 機能で設定します。パラメタは，簡易構築定義ファイルに定義します。

表 8-47　CTM デーモンのタイムアウト閉塞をチューニングするパラメタ

設定項目	設定対象	設定個所（パラメタ名）
タイムアウト発生回数	論理 CTM	ctm.RequestCount
監視時間間隔	論理 CTM	ctm.RequestInterval

（4）CTM で振り分けるリクエストの優先順位を設定するチューニングパラメタ

CTM で振り分けるリクエストの優先順位の設定は，EJB クライアントアプリケーションの場合と，J2EE サーバの場合で異なります。また，J2EE サーバの場合，システムの構築方法によって設定個所が異なります。CTM で振り分けるリクエストの優先順位を設定するチューニングパラメタを次の表に示します。

表 8-48　CTM で振り分けるリクエストの優先順位を設定するチューニングパラメタ

設定単位	設定方法	設定個所
EJB クライアントアプリケーション	ファイル編集または EJB クライアントアプリケーション開始時に指定するシステムプロパティの指定	**定義ファイル（ファイル編集の場合）** usrconf.properties **パラメタ名** ejbserver.client.ctm.RequestPriority キー
J2EE サーバ	Smart Composer 機能	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** ejbserver.client.ctm.RequestPriority

8.9 そのほかの項目のチューニング

ここでは，前の節までに説明した項目以外のチューニング項目について説明します。

ここで説明するのは，次の項目です。

- Persistent Connection **についてのチューニング**

この項目は，Web フロントシステムの場合で，インプロセス HTTP サーバを使用するときにチューニングを検討してください。

HTTP/1.1 では，Web クライアントと Web サーバ間で確立した TCP コネクションを持続して，複数の HTTP リクエスト間で使用し続けるための Persistent Connection が定義されています。Persistent Connection を使用することによって，Web クライアントと Web サーバ間でコネクション接続に掛かる時間を短縮し，通信トラフィックを軽減できます。

ただし，Persistent Connection を使用すると，特定の Web クライアントがリクエスト処理スレッドを占有することになるため，サーバ全体の処理性能が低下することがあります。このため，Persistent Connection を有効に活用し，かつサーバ処理性能を維持できるようにチューニングする必要があります。

インプロセス HTTP サーバを使用する場合，Persistent Connection について，次の項目がチューニングできます。

● Persistent Connection **数の上限値**
この上限値を超える TCP コネクションについては，リクエスト処理終了後に切断されます。これによって，新規接続を処理するスレッドが確保でき，リクエスト処理スレッドを特定のクライアントに占有されることを防げます。

● Persistent Connection **のリクエスト処理回数の上限値**
同じ Web クライアントから連続してリクエスト要求があった場合も，この上限値を超えると，リクエスト処理終了後に一度 TCP コネクションが切断されます。
これによってリクエスト処理スレッドを特定のクライアントに占有され続けることを防げます。

● Persistent Connection **のタイムアウト**
Persistent Connection のリクエスト待ち時間にタイムアウトを設定できます。指定したタイムアウト時間を超えてリクエスト処理要求がない場合は，TCP コネクションが切断されます。これによって，使用されていない状態で TCP コネクションが占有され続けることを防げます。

これらの項目は，Smart Composer 機能で使用する簡易構築定義ファイルのパラメタとして指定します。Persistent Connection について設定するチューニングパラメタについて次の表に示します。

表 8-49 Persistent Connection について設定するチューニングパラメタ

設定項目	設定個所
Persistent Connection 数の上限値	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.connector.inprocess_http.persistent_connection.max_connections
リクエスト処理回数の上限値	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.connector.inprocess_http.persistent_connection.max_requests
タイムアウト	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** webserver.connector.inprocess_http.persistent_connection.timeout

なお，各パラメタの詳細については，マニュアル「Cosminexus 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「4.14　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。

9 パフォーマンスチューニング（バッチアプリケーション実行基盤）

この章では，バッチアプリケーションを実行するシステムのパフォーマンスをチューニングする方法について説明します。パフォーマンスチューニングによって動作環境を最適化することで，システムの性能を最大限に生かせるようになります。J2EE アプリケーション実行基盤のパフォーマンスチューニングについて検討する場合は，「8. パフォーマンスチューニング（J2EE アプリケーション実行基盤）」を参照してください。

9.1 パフォーマンスチューニングで考慮すること

この節では，バッチアプリケーション実行基盤のパフォーマンスチューニングで考慮することについて説明します。

9.1.1 パフォーマンスチューニングの観点

バッチアプリケーション実行基盤のパフォーマンスチューニングは，次の観点で実施します。

- **データベースアクセス方法の最適化**
- **タイムアウトの設定**
- **フルガーベージコレクションを発生させるメモリ使用量のしきい値の設定**

それぞれのポイントについて説明します。

(1) データベースアクセス方法の最適化

データベースアクセス方法の最適化は，生成に時間が掛かるコネクションやステートメントをプールしておくことで，データベースアクセス時のオーバーヘッドを削減することを目的とします。

パフォーマンスチューニングでは，次に示す機能を有効に活用することで，データベースアクセス処理を最適化し，スループットを向上させます。

- コネクションプーリング
- ステートメントプーリング（PreparedStatement および CallableStatement のプーリング）

データベースアクセス方法は，データベースとの接続に DB Connector を使用している場合に最適化できます。

(2) タイムアウトの設定

タイムアウトの設定は，システムのトラブル発生を検知して，リクエストの応答が返らなくなることを防ぎ，適宜リソースを解放することを目的とします。

設定できるタイムアウトには，次の種類があります。

- Enterprise Bean を呼び出す時のタイムアウト
- トランザクションのタイムアウト
- データベースのタイムアウト

(3) フルガーベージコレクションを発生させるメモリ使用量のしきい値の設定

フルガーベージコレクションの制御で使用するしきい値は，オンライン処理とバッチ処理で同じリソースを使用する場合に設定します。バッチサーバのフルガーベージコレクションによってオンライン処理を中断させないことを目的とします。

フルガーベージコレクションの実行を制御することによって，リソースを排他していないタイミングで適切にフルガーベージコレクションを実行できます。

9.1.2 チューニング手順

パフォーマンスチューニングは，システムのパフォーマンスを生かす最適な設定を見つける作業です。構築した環境で，実際に処理を実行しながら，パラメタの調整やボトルネックの調査，解消によってパフォーマンスを向上させていきます。

チューニング作業では，まず，目標値を決定します。次に，各パラメタに初期値を設定した状態でスループットを測定します。各パラメタを調整しながら，目標値に近い最適な値を見つけていきます。

チューニングの際，CPU の利用率の測定には，OS に付属している監視ツールなどが利用できます。JavaVM に関するアプリケーションサーバの稼働情報については，稼働情報収集機能などで確認できます。確認方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「3. 稼働情報の監視（稼働情報収集機能）」を参照してください。

なお，CPU 利用率が 100% からかなり低い状態で飽和した場合は，システム上に入出力処理や排他処理などのボトルネックがあるおそれがあります。ボトルネックを調査し，対策してから，再度パフォーマンスチューニングを実行してください。アプリケーションサーバでは，システムのボトルネックの調査に，性能解析トレースを利用できます。性能解析トレースの機能詳細，および性能解析トレースを利用して取得したトレースファイルの利用方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「6. 性能解析トレースを使用したシステムの性能解析」を参照してください。

9.1.3 チューニング項目

バッチアプリケーション実行基盤でのチューニング項目について，次の表に示します。

表 9-1 バッチアプリケーション実行基盤でのチューニング項目

チューニング項目	利用できる機能	参照先
データベースアクセス方法の最適化	コネクションプーリング※1	8.5.1※2
	ステートメントプーリング※1	8.5.2※2

チューニング項目	利用できる機能	参照先
タイムアウトの設定	Enterprise Bean 呼び出しでのタイムアウトの設定	8.6.3 ※2，※3
	トランザクションタイムアウトの設定	9.3.2
	データベースでのタイムアウトの設定	8.6.5 ※2
フルガーベージコレクションの制御※1	しきい値の設定	9.4

注※ 1

DB Connector を使用している場合に利用できる機能です。

注※ 2

J2EE アプリケーション実行基盤の説明を参照してください。その際，説明中の「J2EE サーバ」を「バッチサーバ」に読み替えてください。また，「J2EE アプリケーション」を「バッチアプリケーション」に読み替えてください。

注※ 3

Enterprise Bean 呼び出しでは，J2EE アプリケーション実行基盤のバックシステムと同じ項目にタイムアウトを設定できます。

9.2　チューニングの方法

この節では，チューニングの方法について説明します。チューニングの方法は，設定対象の種類によって異なります。

9.2.1　バッチサーバのチューニング

バッチサーバのチューニングには，Smart Composer 機能の簡易構築定義ファイルを使用します。簡易構築定義ファイルでは，<configuration> タグ下の <logical-server-type> に設定対象とする論理サーバの種類（J2EE サーバ※）を指定して，<param> タグ下でパラメタ名とその値を設定します。簡易構築定義ファイルの詳細については，マニュアル「Cosminexus 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「4.6　簡易構築定義ファイル」を参照してください。

注※

Smart Composer 機能では，バッチサーバを J2EE サーバとして扱います。

9.2.2　リソースのチューニング

リソースのチューニングをする場合は，サーバ管理コマンドを使用します。

サーバ管理コマンドを使用する場合は，Connector 属性ファイルを編集します。Connector 属性ファイルの詳細については，マニュアル「Cosminexus 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　Connector 属性ファイル」を参照してください。

9.3 タイムアウトを設定する

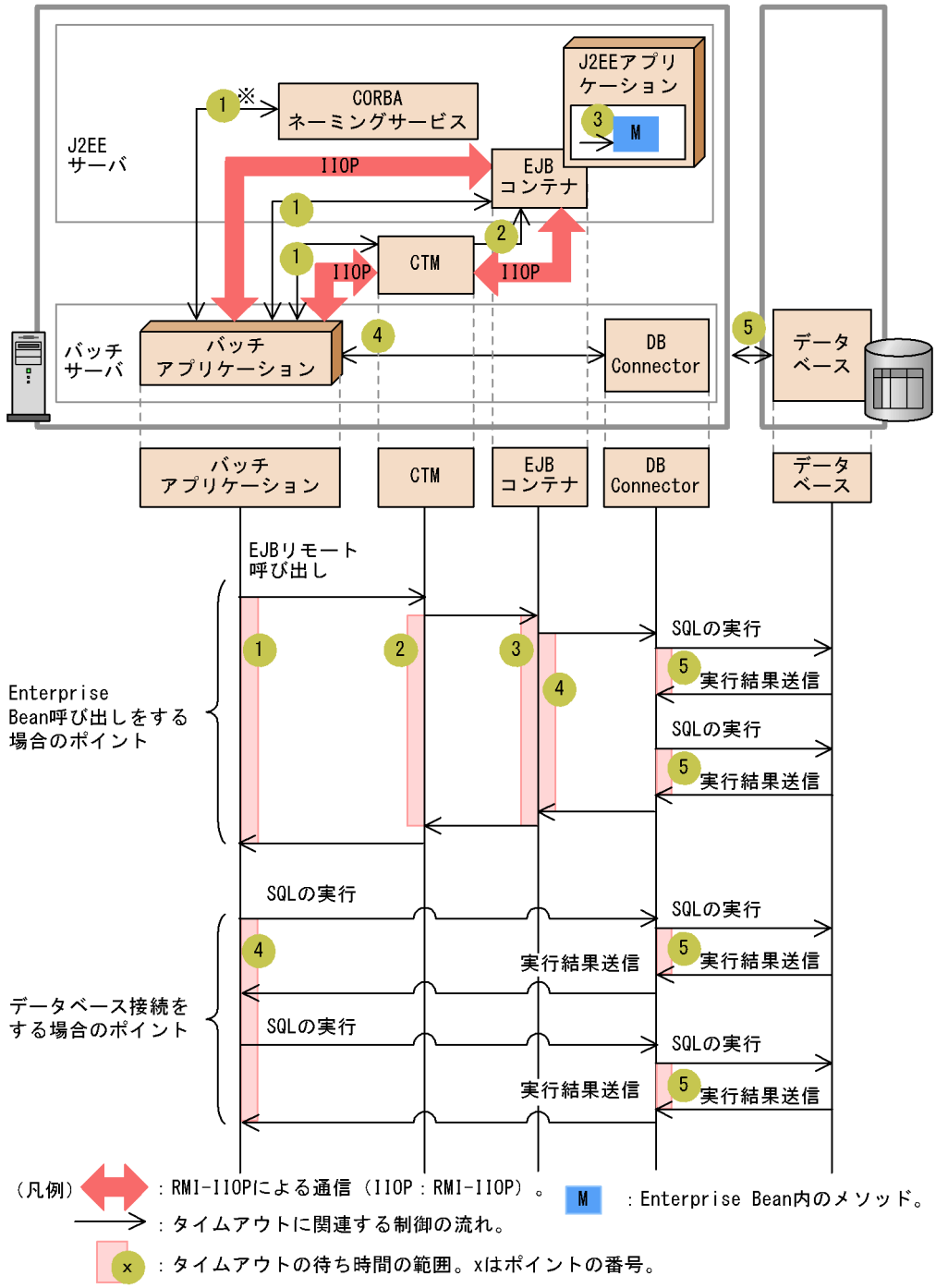
アプリケーションサーバのシステムでは，トラブル発生時にリクエストの応答が戻ってこない状態になることを防ぐために，幾つかのポイントにタイムアウトを設定できます。

この節では，システム全体でタイムアウトが設定できるポイントと，設定する場合の指針について説明します。

9.3.1 タイムアウトが設定できるポイント

バッチアプリケーションを実行するシステムでは，次の図に示すポイントにタイムアウトが設定できます。

図 9-1　タイムアウトが設定できるポイント



注※　CORBAネーミングサービスとのタイムアウトは，バッチアプリケーションから
JNDIによってCORBAネーミングサービスへの問い合わせを発行して，結果が返却されるまでの
時間になります。

それぞれのポイントに設定するタイムアウトは，次の表に示すような用途で使い分けられます。

表 9-2 各ポイントに設定するタイムアウトの目的とデフォルトのタイムアウト設定

ポイント	タイムアウトの種類	主な用途
1	バッチサーバ側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI ネーミングサービス呼び出しのタイムアウト	バッチサーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
2※	バッチサーバ側で設定する CTM からの Enterprise Bean 呼び出しのタイムアウト	バッチサーバの業務処理の障害（無限ループ，デッドロックなど），または通信路の障害の検知
3	Enterprise Bean 呼び出しアクセスした EJB で設定するメソッドの実行時間のタイムアウト	J2EE サーバの業務処理の障害（無限ループ，デッドロックなど）
4	バッチサーバ側で設定するデータベースのトランザクションタイムアウト	データベースサーバの障害（サーバダウンまたはデッドロックなど）の検知，またはリソースの長時間占有防止
5	データベースのタイムアウト	データベースサーバの障害（サーバダウンまたはデッドロックなど）の検知，またはリソースの長時間占有防止

注※ CTM を使用している場合にだけ存在するポイントです。CTM を利用しない構成の場合，ポイント 2 の範囲はバッチサーバから EJB コンテナに EJB リモート呼び出しを実行してから，EJB コンテナからバッチサーバに実行結果が送信されるまでの間になります。

これらのタイムアウトの基本的な設定指針は次のとおりです。

- タイムアウト値の設定は，呼び出し元（バッチサーバ）に近いほど大きな値を設定するのが原則です。このため，次の関係で設定することを推奨します。
  - ポイント 1= ポイント 2> ポイント 3> ポイント 4> ポイント 5
- 1，4，5 のポイントのタイムアウト値を設定する場合は，呼び出し処理に通常どの程度の時間が掛かっているかを見極めた上で，呼び出す処理（業務）ごとに算出して設定してください。

なお，1 ～ 5 のポイントは，システムでの位置づけによって，次の二つに分けられます。

- Enterprise Bean 呼び出しで意識する必要があるポイント（1 ～ 3）
  このポイントでタイムアウトを設定する項目は，J2EE アプリケーション実行基盤のバックシステムで設定できる項目と同じです。詳細は，「8.6.3 バックシステムでのタイムアウトを設定する」を参照してください。
- データベース接続時に意識する必要があるポイント（4 と 5）
  このポイントは，さらにトランザクションでのタイムアウトとデータベースでのタイムアウトに分けて意識する必要があります。

トランザクションでのタイムアウトの詳細は，「9.3.2　トランザクションタイムアウトを設定する」を参照してください。
データベースでのタイムアウトを設定する項目は，J2EE アプリケーション実行基盤のバックシステムで設定できる項目と同じです。詳細は，「8.6.5　データベースでのタイムアウトを設定する」を参照してください。

それぞれのポイントでの設定については，バッチアプリケーション実行基盤の「9.3.3 タイムアウトを設定するチューニングパラメタ」，および J2EE アプリケーション実行基盤の「8.6.7　タイムアウトを設定するチューニングパラメタ」を参照してください。

参考

それぞれのポイントのデフォルト値は次のとおりです。

ポイント	デフォルト値
1	設定されていません。レスポンスを待ち続けます。
2	ポイント 1 と同じ値が Enterprise Bean 呼び出し時に自動的に引き継がれて設定されます。
3	設定されていません。タイムアウトしません。
4	180 秒
5	データベースの種類とタイムアウトの設定個所ごとに異なります。※ HiRDB の場合 ロック解放待ちタイムアウト：180 秒 レスポンスタイムアウト：0 秒（HiRDB クライアントは HiRDB サーバからの応答があるまで待ち続けます） リクエスト間隔タイムアウト：600 秒 SQL Server の場合 メモリ取得待ちタイムアウト：-1（-1 を指定した場合の動作は，SQL Server のドキュメントを参照してください） ロック解放待ちタイムアウト：-1（ロックが解放されるまで待ち続けます） XDM/RD E2 の場合 ロック解放待ちタイムアウト：なし（タイムアウト時間を監視しません） SQL 実行 CPU 時間タイムアウト：10 秒 SQL 実行経過時間タイムアウト：0 秒（タイムアウト時間を監視しません） トランザクション経過時間タイムアウト：600 秒 レスポンスタイムアウト：0 秒（HiRDB クライアントは XDM/RD E2 サーバからの応答があるまで待ち続けます）

注※
Oracle の場合，ロック解放待ちタイムアウトのデフォルトはありません。

9.3.2　トランザクションタイムアウトを設定する

ここでは，トランザクションタイムアウトの設定について説明します。トランザクションタイムアウトは，データベースシステムなど EIS とのトランザクションに設定します。

DB Connector を使用してデータベースにアクセスするときのトランザクションタイムアウトについて説明します。

トランザクションタイムアウトを設定する場合は，システム全体のタイムアウトのうち，バッチサーバとデータベースのトランザクションについて意識する必要があります。

トランザクションタイムアウトが発生すると，アプリケーションサーバによって，次の処理が実行されます。

- 実行中のトランザクションはロールバックされます。
- トランザクションに参加しているコネクションはクローズされ，コネクションプールから削除されます。

ポイント

トランザクションの管理方法は BMT になります。トランザクションのタイムアウトは，usrconf.properties または JTA の API（javax.transaction.UserTransaction#setTransactionTimeout メソッド）に指定できます。usrconf.properties の定義は，プロセス全体に影響します。API に指定したタイムアウトは，API を発行したトランザクションの範囲だけに影響します。また，API の指定は，usrconf.properties の定義よりも優先されます。
このため，プロセス全体に設定したい標準的な値を usrconf.properties に定義して，呼び出す業務によって細かく設定したい値は適宜 API を使用して設定することをお勧めします。

トランザクションタイムアウトが発生した場合，バッチアプリケーションに例外は通知されません。ただし，メッセージ KDJE31002-W がログファイルとバッチサーバのコンソールに出力されます。また，トランザクションタイムアウトが発生したあとで，バッチアプリケーションから該当するトランザクションを使用して JTA インタフェースまたは JDBC インタフェースを使用しようとすると，例外が通知されます。

9.3.3 タイムアウトを設定するチューニングパラメタ

ここでは，タイムアウトの設定で使用するチューニングパラメタの設定方法についてまとめて示します。

（1） バッチサーバ側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト

図 9-1 のポイント 1 のタイムアウトを設定するチューニングパラメタです。

設定方法は，J2EE アプリケーション実行基盤と同じです。「8.6.7　タイムアウトを設定するチューニングパラメタ」の「(6)EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト」を参照してください。

なお，「8.6.7　タイムアウトを設定するチューニングパラメタ」の説明中の「ポイント7」が，図 9-1 の「ポイント 1」に対応します。

（2） EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト

図 9-1 のポイント 2 のタイムアウトを設定するチューニングパラメタです。

設定方法は，J2EE アプリケーション実行基盤と同じです。「8.6.7　タイムアウトを設定するチューニングパラメタ」の「(7)EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト」を参照してください。

なお，「8.6.7　タイムアウトを設定するチューニングパラメタ」の説明中の「ポイント8」が，図 9-1 の「ポイント 2」に対応します。

（3） Enterprise Bean 呼び出しアクセスした EJB で設定するメソッドの実行時間のタイムアウト

図 9-1 のポイント 3 のタイムアウトを設定するチューニングパラメタです。

設定方法は，J2EE アプリケーション実行基盤と同じです。「8.6.7　タイムアウトを設定するチューニングパラメタ」の「(10)　J2EE アプリケーションのメソッドタイムアウト」を参照してください。

なお，「8.6.7　タイムアウトを設定するチューニングパラメタ」の説明中の「ポイント9」が，図 9-1 の「ポイント 3」に対応します。

（4） バッチサーバ側で設定するデータベースのトランザクションタイムアウト（DB Connector を使用した場合）

図 9-1 のポイント 4 のタイムアウトを設定するチューニングパラメタです。

バッチサーバ単位，Enterprise Bean，インタフェース，API による呼び出し単位（BMT の場合）に設定します。

トランザクションタイムアウトのチューニングパラメタを次の表に示します。

表 9-3　トランザクションタイムアウトのチューニングパラメタ

単位	設定方法	設定項目	設定個所
バッチサーバ単位	Smart Composer 機能	トランザクションのトランザクションタイムアウト時間のデフォルト	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** ejbserver.jta.TransactionManager.defaultTimeOut

単位	設定方法	設定項目	設定個所
API 単位（BMT）	API	トランザクションタイムアウト時間	UserTransaction#setTransactionTimeout メソッド※

注※　パッケージ名は javax.transaction です。

(5) データベースのタイムアウト

図 9-1 のポイント 5 のタイムアウトを設定するチューニングパラメタです。

設定方法は，J2EE アプリケーション実行基盤と同じです。「8.6.7　タイムアウトを設定するチューニングパラメタ」の「(9) データベースのタイムアウト」を参照してください。

なお，「8.6.7　タイムアウトを設定するチューニングパラメタ」の説明中の「ポイント 11」が，図 9-1 の「ポイント 5」に対応します。

9.4 ガーベージコレクション制御で使用するしきい値を設定する

バッチサーバのフルガーベージコレクションを実行するタイミングは，メモリ使用量にしきい値を設定することで制御できます。しきい値は，バッチ処理とオンライン処理で同じリソースにアクセスする場合に設定することをお勧めします。適切なしきい値を設定しておくことで，オンライン処理とバッチ処理の両方のスループットを確保できます。

ポイント

フルガーベージコレクションの詳細については，「7.1.3 ガーベージコレクションの発生とメモリ空間の関係」を参照してください。

9.4.1 しきい値を設定する目的

オンライン処理とバッチ処理で同じリソースにアクセスする場合，オンライン処理のスループットに影響を与えないように考慮する必要があります。

バッチアプリケーション実行時に空きメモリが少なくなると，JavaVM によってバッチサーバのフルガーベージコレクションが実行されます。この場合，バッチサーバ上で動作するすべてのプログラムの処理が中断されます。バッチアプリケーションがリソースを排他していた場合，そのリソースはバッチサーバのフルガーベージコレクション実行中も排他された状態になります。オンライン処理の中に排他中のリソースを使用する処理があった場合は，そのオンライン処理も中断されてしまいます。

これを防ぐために，メモリ使用量のしきい値を設定して，メモリ不足が起こる前に明示的にフルガーベージコレクションを発生させるようにします。明示的なフルガーベージコレクションは，リソースを排他していないタイミングで発生するように制御できます。JavaVM によってフルガーベージコレクションが実行される前に空きメモリを増やしておくことで，リソース排他中にフルガーベージコレクションが実行されることを防ぎます。

しきい値を設定した場合，次に示す状態になるとフルガーベージコレクションが実行されます。ただし，そのときにバッチアプリケーションがリソースを排他していた場合は，排他が解除されるまで待ってから実行されます。

- Tenured 領域消費サイズの Tenured 領域合計サイズに対する割合≧しきい値
- New 領域合計サイズの Tenured 領域最大空きサイズに対する割合≧しきい値
- Permanent 領域消費サイズの Permanent 領域合計サイズに対する割合≧しきい値

9.4.2 しきい値設定の考え方

しきい値は，次の式で算出した値を目安に算出できます。

しきい値≦ 100 －（100 ×リソース排他解除を待つ間に必要な空きメモリのサイズ）／最大メモリサイズ

しきい値設定の際には，次の点を考慮してください。

● フルガーベージコレクションの発生頻度

● リソース排他解除を待つ間に必要な空きメモリ

ここでは，フルガーベージコレクションの発生頻度としきい値の関係と，リソース排他解除を待つ間に必要な空きメモリのサイズの見積もり方法について説明します。

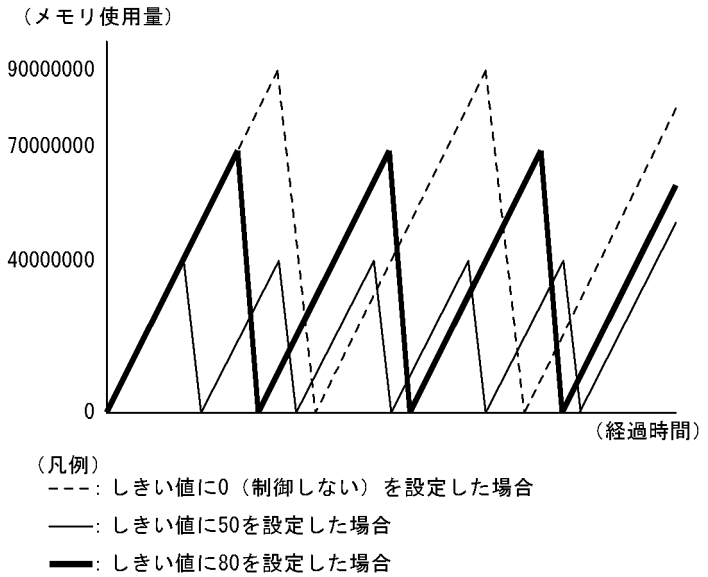
（1） フルガーベージコレクションの発生頻度としきい値の関係

フルガーベージコレクションはプログラムの実行速度に比べて時間の掛かる処理です。このため，JavaVM のチューニングでは，フルガーベージコレクションをできるだけ発生させないようにします。JavaVM のチューニングの考え方については，「7.2.1　チューニングの考え方」を参照してください。

しきい値を設定して明示的に実行するフルガーベージコレクションも，頻度が少なくなるよう，チューニングする必要があります。

設定したしきい値ごとのメモリ使用量の例を，次の図に示します。

図 9-2　設定したしきい値ごとのメモリ使用量の例



JavaVM でのメモリ使用量は，時間の経過とともに増加していき，フルガーベージコレクションが発生すると減少します。

しきい値に 0 を設定した場合は，JavaVM によって自動的に実行されるまでフルガーベージコレクションは実行されません。しきい値に小さな値を設定した場合は，大きな値を指定した場合に比べて，フルガーベージコレクションが実行される頻度が高くなります。図の場合は，しきい値として 80 を設定した場合の方が，50 を設定した場合よりもフルガーベージコレクションの実行頻度を低く抑えることができています。

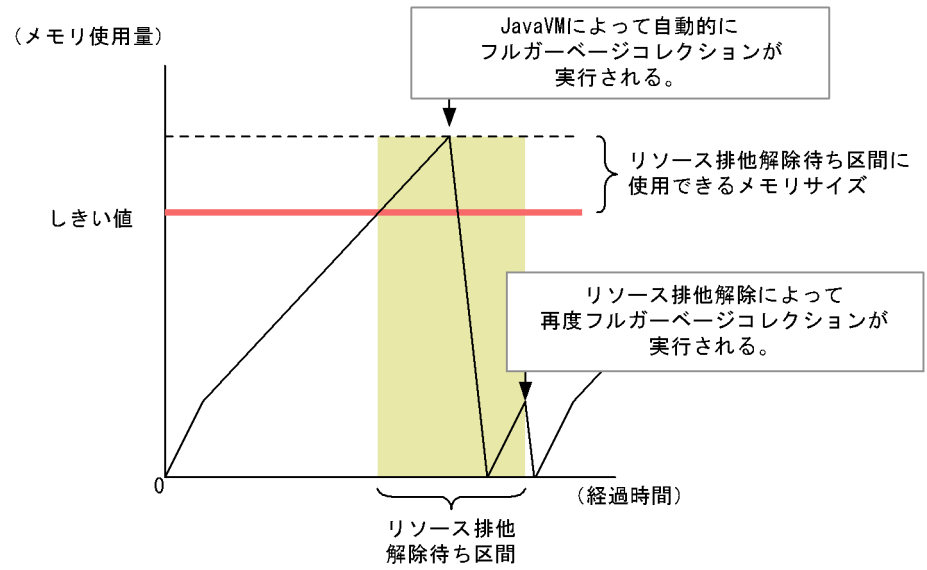
ただし，フルガーベージコレクションの実行頻度を少なくした場合，1 回の実行に掛かる時間は，頻繁に実行する場合に比べて長くなります。

(2) リソース排他解除を待つ間に必要な空きメモリの見積もり

メモリ使用量がしきい値以上になった場合，リソース排他が解除されるまでフルガーベージコレクションは実行されません。しかし，リソース排他の解除を待っている間に JavaVM が必要とするメモリが不足した場合は，リソース排他解除を待たないで JavaVM によってフルガーベージコレクションが実行されてしまいます。

リソース排他解除を待つ間に空きメモリが不足した場合の例を次の図に示します。

図 9-3　リソース排他解除を待つ間に空きメモリが不足した場合の例



ポイント

JavaVM にフルガーベージコレクションを実行させないために必要な空きメモリについては，「7.1.3　ガーベージコレクションの発生とメモリ空間の関係」を参照してください。

フルガーベージコレクションが発生するメモリ使用量を 100 とする場合，リソース排他解除待ち区間に使用できるメモリサイズは，「100 －しきい値（%）」です。

例えば，しきい値として「95」のように高い値を指定した場合は，リソース排他解除を待っている間に使用できる空きメモリが 5% と少ないため，リソース排他が解除される前に JavaVM によって自動的にフルガーベージコレクションが実施されてしまうおそれがあります。

このため，しきい値を見積もる場合には，リソース排他待ち区間にメモリが不足しないよう，十分に余裕を持った値を設定するようにしてください。

注意事項

リソース排他解除待ち区間で空きメモリが不足して JavaVM によるフルガーベージコレクションが実行された場合，リソースの排他が解除されたタイミングで，フルガーベージコレクション制御によってもう一度フルガーベージコレクションが実行されます。

9.4.3 ガーベージコレクション制御で使用するしきい値を設定するためのチューニングパラメタ

ここでは，バッチサーバのフルガーベージコレクションを実行するしきい値を設定するために使用するチューニングパラメタの設定方法について示します。

表 9-4 バッチサーバのフルガーベージコレクションを実行するしきい値を設定するチューニングパラメタ

設定項目	設定個所
しきい値	**定義ファイル** 簡易構築定義ファイル **設定対象** 論理 J2EE サーバ（j2ee-server） **パラメタ名** ejbserver.batch.gc.watch.threshold

10 セキュアなシステムの検討

業務システムを安全に稼働させ，扱うデータを確実に守るためには，システム設計の段階でセキュリティについて十分に検討しておく必要があります。この章では，セキュアなシステムを構築・運用するための考え方，正しい構築・運用手順，および監査の方法について説明します。
また，外部ネットワークを使用するシステムの場合に想定されるセキュリティ上の脅威を明確にし，それぞれに対して，ハードウェアおよびソフトウェアを使用して対策する方法についても説明します。
なお，この章は，J2EE アプリケーションを実行するシステムの場合に参照してください。バッチアプリケーションを実行するシステムの場合は該当しません。

10.1 セキュアなシステムの検討の概要

システムを管理または運用するユーザがシステムを構築・運用していく過程や，システムが提供するサービスをエンドユーザが利用していく過程で，システムにはセキュリティ上のさまざまな脅威が想定されます。脅威からシステムを守るには，システムを物理的に安全な構成に設計したり，作業者の運用ルールを定めたりするなど，対策を実施する必要があります。

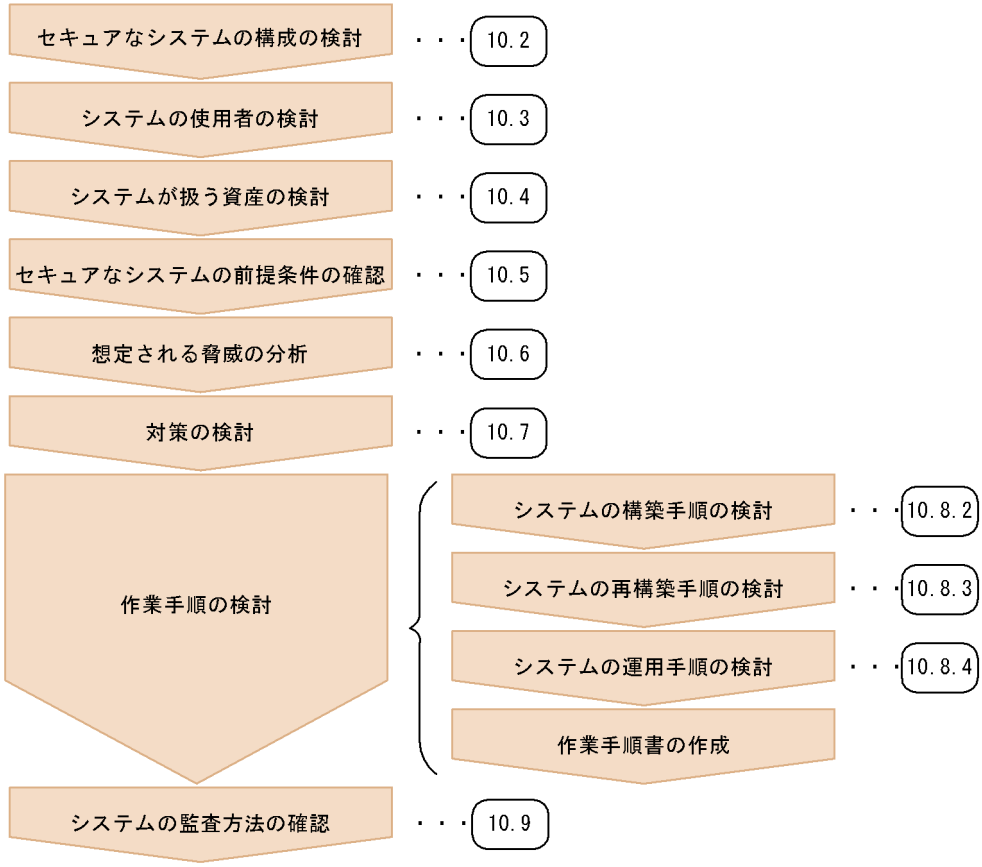
また，近年，組織の健全な運営の保証や，複雑化・多様化する IT システムの安全な構築・運用の観点から，組織の内部統制の重要性が高まっています。その中では，システムがセキュアに保たれていることを監査者に証明できることが求められます。そのためには，システムに対して，いつ，だれが，どんな操作を実施したのかをログに記録して，システムに対して正当な権限を持つ作業者が正当な業務を実施したことを監査できる仕組みが必要です。

このようなセキュアなシステムを実現するためには，想定される脅威をシステムの設計時に明確にして，それに応じた対策を実現できるシステムを検討していく必要があります。

この章では，システムに対して想定される脅威を明確にした上で，セキュアなシステムを構築・運用するための考え方や手順など，システム設計時に検討が必要なことについて説明します。

セキュアなシステムの検討は，次の流れで実施します。

図 10-1　セキュアなシステムの検討の流れ



図中の参照先を示します。

セキュアなシステムの検討の流れ		参照先マニュアル	参照個所
セキュアなシステムの構成の検討		このマニュアル	10.2
システムの使用者の検討			10.3
システムが扱う資産の検討			10.4
セキュアなシステムの前提条件の確認			10.5
想定される脅威の分析			10.6
対策の検討			10.7
作業手順の検討	システムの構築手順の検討		10.8.2

セキュアなシステムの検討の流れ		参照先マニュアル	参照個所
	システムの再構築手順の検討		10.8.3
	システムの運用手順の検討		10.8.4
	作業手順書の作成		―
システムの監査方法の確認			10.9

（凡例）
　―：該当なし。

なお，この図では，企業内部で使用されるシステムのセキュリティを確保するための作業の流れを示しています。外部からの脅威に対する対策については，「10.10　外部ネットワークを使用するシステムでのセキュリティの検討」を参照してください。

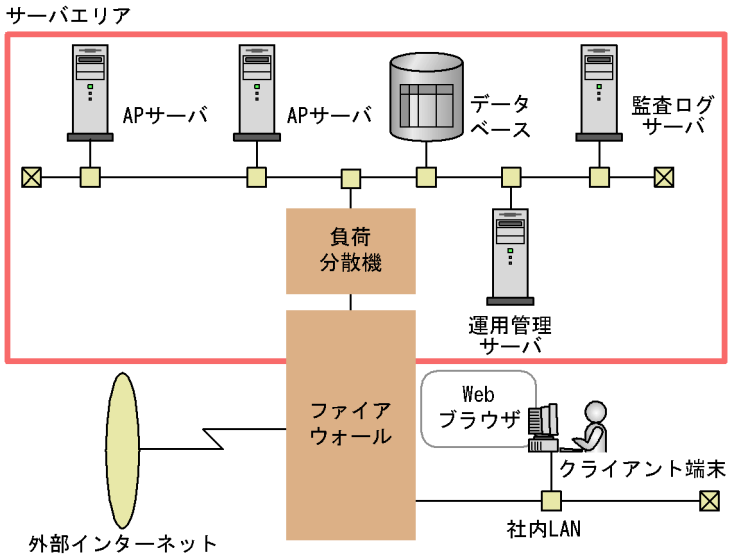
10.2　セキュアなシステムの構成の検討

ここでは，セキュアなシステムの構成について説明します。このマニュアルではセキュアなシステムとして，次のようなシステムを想定しています。

- 大企業で常時稼働して，社内で利用されるシステムです。
- システムを構成する要素はすべて社内 LAN 上に配置されます。
- システムが提供するサービスは，社内のクライアント端末から Web ブラウザを使用して利用されます。
- クライアント端末からサービスを利用するには，ログイン処理が必要です。システムに登録されていないユーザはサービスを利用できません。

このシステムの構成を次の図に示します。

図 10-2　セキュアなシステムの構成



注　APサーバ：アプリケーションサーバ

このシステムの構成要素を次に示します。なお，説明中にある「システム管理者」，「システム運用者」，「監査者」，および「エンドユーザ」の定義については，「10.3　システムの使用者の検討」を参照してください。

サーバエリア

ハードウェアを管理するための，物理的に隔離されたスペースです。サーバエリア内のハードウェアは，システム管理者が管理します。サーバエリアに入室できるのは，システム管理者，システム運用者，および監査者だけです。

アプリケーションサーバ

Web サーバ，サービスを提供する J2EE アプリケーション，および J2EE アプリケーションが稼働するために必要なサーバプログラムが稼働しているマシンです。サーバエリアに複数台設置されて，負荷分散機によって負荷分散されます。

データベース
ユーザの情報やサービスが処理した情報が格納されているマシンです。サーバエリアに設置します。

監査ログサーバ
監査ログを収集して，監査を実施するためのサーバです。監査者だけが使用します。サーバエリアに設置します。

運用管理サーバ
アプリケーションサーバを管理する，運用管理プログラムが稼働するマシンです。システム構築時にシステム管理者が，システム運用時にシステム運用者が使用します。サーバエリアに設置します。

負荷分散機
アプリケーションサーバを複数台設置する場合に，負荷分散をする装置です。サーバエリアに設置します。

ファイアウォール
サーバエリア，社内 LAN，および外部インターネットの間に配置します。

クライアント端末
システムが提供するサービスを利用するための端末です。エンドユーザは，クライアント端末上の Web ブラウザを使用して，社内 LAN 経由でアプリケーションサーバにアクセスします。

10.3 システムの使用者の検討

セキュアなシステムを検討するには，まず，システムの使用者を定義します。システムの使用者にはどのようなユーザがいるかを洗い出して，それぞれの使用者の目的や作業範囲を明確に定義します。これが，システムの監査を実施する観点の一つである，正しいユーザがそれぞれの操作を実行したかどうかを検証する根拠になります。

なお，それぞれの使用者の作業手順は，作業手順書を作成して定義する必要があります。作業手順書には，「システム構築手順書」，「システム運用手順書」，「エンドユーザ操作手順書」，「入退室手順書」などが考えられます。作業手順書の検討については，「10.8 作業手順の検討」を参照してください。

このシステムでは，次に示す使用者を定義します。

システム管理者

システム管理者は，「システム構築手順書」に従って，システムの構築，および管理を実施します。具体的には，主に次の作業を実施します。

- サーバエリア内のハードウェア，ソフトウェア，ネットワークの設置・設定
- ソフトウェアのアップデート
- システムの起動と停止

システム管理者は，組織内の情報システム部門から選ばれたユーザが担当します。

システム運用者

システム運用者は，「システム運用手順書」に従って，サーバエリア内でエンドユーザの登録・削除などのシステムの運用作業を実施します。システム運用者は，社内の情報システム部門から委託されたユーザが担当します。

エンドユーザ

エンドユーザは，「エンドユーザ操作手順書」に従って，システムが提供するサービスを利用します。サービスの利用には，社内 LAN に接続されたクライアント端末上の Web ブラウザを使用します。

監査者

監査者は「入退室手順書」に従ってサーバエリアに入室して，監査ログを取得します。取得した監査ログを調査して，信頼できるシステム管理者が「システム構築手順書」に従った正しい方法でシステムを構築しているかどうかを確認します。また，正しい方法で構築されたシステムが，「システム運用手順書」および「エンドユーザ操作手順書」に従った正しい方法で運用，および利用されているかどうかを監査します。監査者は，社内の内部監査を実施するコンプライアンス部門から選ばれたユーザが担当します。

10.4 システムが扱う資産の検討

セキュアなシステムを検討するには，システムが扱う資産（データ）のうち，保護する必要がある資産にはどのようなものがあるのかを明確に判断する必要があります。

このシステムでは，システムが扱う資産（データ）のうち，次の資産を保護する必要があると判断しています。

- **システム管理者のユーザ情報**
- **エンドユーザのユーザ情報**
- **システム構築時の設定ファイル**
- J2EE **アプリケーション**
- **サービスを利用する中でエンドユーザが送信して，**J2EE **アプリケーションが処理した情報**
- **監査ログ**

保護する必要があると判断した資産については，使用できる権限を制御するなどの対策が必要です。対策の詳細については，「10.7　対策の検討」を参照してください。

10.5 セキュアなシステムの前提条件の確認

ここでは，セキュアなシステムの前提条件について説明します。

セキュアなシステムでは，ハードウェアの設置方法や作業者に関する前提条件の確認が必要になります。前提条件を把握した上で，アプリケーションサーバが提供する機能やOSの機能などを使用して，想定される脅威に対する対策を実施します。

ここでは，次の二つの前提条件を想定します。

- 物理的な前提条件
- 運用上の前提条件

10.5.1 物理的な前提条件

セキュアなシステムを構築するための，物理的な前提条件を次に示します。

- システムが稼働するハードウェア，ファイアウォール，それぞれのサーバ，および内部ネットワークが，外部から物理的に隔離されたサーバエリアに設置されていること。
- 施錠管理などによって，認証されたユーザ以外がサーバエリアに入室できないこと。
- システムが稼働するために必要のないハードウェア，およびソフトウェアは，サーバエリア内には持ち込まれないこと。

10.5.2 運用上の前提条件

セキュアなシステムを構築するための運用上の前提条件を次に示します。

運用上の前提条件には，作業手順書，システム管理，システム運用，およびシステム監査に関する前提条件があります。それぞれの前提条件を次に示します。

作業手順書に関する前提条件

システムの構築手順は「システム構築手順書」に，システムの運用手順は「システム運用手順書」に，システムの操作手順は「エンドユーザ操作手順書」に記載されていること。

システム管理に関する前提条件

システムが稼働するために必要なサーバエリア内のハードウェア，ソフトウェア，およびJ2EEアプリケーションは，「システム構築手順書」に従ってシステム管理者が構築・設定すること。また，システム管理者は，信頼できる者が担当すること。

システム運用に関する前提条件

システムが稼働するために必要な，サーバエリア内のハードウェア，ソフトウェア，およびJ2EEアプリケーションは，システム運用者が運用すること。

システム監査に関する前提条件

システム監査を実施する監査者は，信頼できる者が担当すること。

10.6 想定される脅威の分析

「10.3 システムの使用者の検討」と「10.4 システムが扱う資産の検討」で検討した内容，および「10.5 セキュアなシステムの前提条件の確認」で確認した内容から，システムにはどんな脅威が想定されるかを分析します。

このシステムでは次の脅威が想定されます。

- **不正なユーザによるサービスの利用**
  システムに登録されていないエンドユーザがサービスを利用するおそれがあります。
- **手順書に従わないユーザによるサービスの利用**
  システムに登録されているユーザ ID およびパスワードを入手しているエンドユーザが，「エンドユーザ操作手順書」に従わないで，システムの脆弱性を利用してサービスを利用するおそれがあります。
  また，システムに登録しているユーザが，権限のないサービスを利用するおそれがあります。
- **不正なシステム管理者によるシステム構築**
  システム管理者ではないユーザが，「入退室手順書」に従わないで不正にサーバエリアに入室してシステムを構築するおそれがあります。
- **不正なシステム運用者によるシステム運用**
  システム運用者ではないユーザが，「入退室手順書」に従わないで不正にサーバエリアに入室してシステムを運用するおそれがあります。
- **手順書に従わないシステム運用者によるシステム運用**
  システム運用者の Management Server の管理ユーザアカウントを使用して，「システム運用手順書」に従わない方法でシステムを運用するおそれがあります。

「10.7 対策の検討」で説明する対策を実施すると，これらの脅威からシステムを守れます。

「入退室手順書」，「システム構築手順書」，「システム運用手順書」，および「エンドユーザ操作手順書」の詳細については，「10.8 作業手順の検討」を参照してください。

10.7　対策の検討

この節では，実施する対策の内容と，対策を実施したシステムの動作について説明します。

実施する対策には，次の二つがあります。

- 前提条件に対して実施する対策
  「10.5　セキュアなシステムの前提条件の確認」で確認した前提条件に対する対策です。
- 想定した脅威に対して実施する対策
  「10.6　想定される脅威の分析」で想定した脅威に対する対策です。

それぞれについて次に示します。

10.7.1　前提条件に対して実施する対策

ここでは，「10.5　セキュアなシステムの前提条件の確認」で確認した前提条件に対して実施する，対策の内容について説明します。

「10.5　セキュアなシステムの前提条件の確認」で確認した前提条件と，実施する対策を次の表に示します。それぞれの前提条件の詳細については，「10.5　セキュアなシステムの前提条件の確認」を参照してください。

表 10-1　前提条件と実施する対策

前提条件	対策
物理的な前提条件	• 物理的な対策
運用上の前提条件	• システム管理者に対する対策 • システム運用者に対する対策 • システム監査者に対する対策

それぞれの対策の概要を次に示します。

(1) 物理的な前提条件に対する対策

物理的な前提条件に対する対策を次に示します。

● 物理的な対策

- システム管理者は，システムが稼働するハードウェア，ファイアウォール，それぞれのサーバ，および内部ネットワークを，外部から物理的に隔離されたサーバエリアに設置します。
- システム管理者は，システムが稼働するために必要のないハードウェア，およびソフトウェアを，サーバエリア内に持ち込まないようにします。
- システム管理者，システム運用者，および監査者は，「入退室手順書」に従ってサーバエリアに入退室するようにします。

「入退室手順書」については，「10.8　作業手順の検討」を参照してください。

(2) 運用上の前提条件に対する対策

運用上の前提条件に対する対策を次に示します。

● システム管理者に対する対策

- システム管理者には，システム全体に責任を持っていて，悪意のある行為はしない，信頼できるユーザを選定すること。
- システム管理者は，システムの構築および管理に関するトレーニングなどを実施して，システムの構築，および管理方法を熟知していること。また，システムで利用するそれぞれの機器の構築，および管理方法について熟知していること。
- システム管理者は，セキュリティ面での注意点を考慮しながらシステムを構築および管理すること。
- システム管理者は，自分の OS のパスワード，システム管理者の Management Server の管理パスワード，システム運用者の OS のパスワード，およびシステム運用者の Management Server の管理パスワードとして，推測されにくい十分に強度のあるパスワードを設定すること。

「システム構築手順書」については，「10.8　作業手順の検討」を参照してください。

● システム運用者に対する対策

- システム運用者は，システムの運用に関するトレーニングなどを実施して，システムの運用方法を熟知していること。
- システム運用者は，セキュリティ面での注意点を考慮しながらシステムを運用すること。
- システム運用者は，エンドユーザのパスワードとして，推測されにくい十分に強度のあるパスワードを設定すること。

「システム運用手順書」については，「10.8　作業手順の検討」を参照してください。

● システム監査者に対する対策

- 監査者には，システム全体に責任を持っていて，悪意のある行為はしない，信頼できるユーザを選定すること。
- 監査者には，システム管理者以外のユーザを選定すること。
- 監査者は，システムの構築手順が正当であるかどうかを確認すること。また，運用手順が正当であるかどうかを監査すること。

10.7.2　想定した脅威に対して実施する対策

ここでは，「10.6　想定される脅威の分析」で想定した脅威に対して実施する，対策の内容について説明します。

このシステムで想定される脅威と，それぞれの脅威に対する対策を，対策を実施する対象者ごとに次の表に示します。それぞれの脅威の詳細については，「10.6　想定される脅威の分析」を参照してください。

表 10-2　想定される脅威と実施する対策

対策の対象者	脅威	対策
システム管理者	不正なシステム管理者によるシステム構築	• OS のユーザ識別・認証
システム運用者	不正なシステム運用者によるシステム運用	• OS のユーザ識別・認証 • システム運用者のユーザ識別・認証
	手順書に従わないシステム運用者によるシステム運用	• システムの監査ログ出力 • J2EE アプリケーションの監査ログ出力
エンドユーザ	不正なユーザによるサービスの利用	• J2EE アプリケーションの監査ログ出力 • J2EE アプリケーションのユーザ識別・認証
	手順書に従わないユーザによるサービスの利用	• J2EE アプリケーションの監査ログ出力 • J2EE アプリケーションのアクセス制御

それぞれの対策の概要を次に示します。

システム管理者を対象にした対策

- **OS のユーザ識別・認証**

  システム管理者だけがシステムを管理するように，システムが動作する OS のユーザ識別・認証を設定して，コマンドの実行権限を制御します。

システム運用者を対象にした対策

- **OS のユーザ識別・認証**

  システム運用者がシステムを運用するように，システムが動作する OS のユーザ識別・認証を設定して，コマンドの実行権限を制御します。
- **システム運用者のユーザ識別・認証**

  システム運用者がシステムを運用するように，システムでユーザ識別・認証をします。
- **システムの監査ログ出力**

  手順書に従った方法でシステムを運用したかどうかを監査するために，システムで監査ログを出力するようにします。
- **J2EE アプリケーションの監査ログ出力**

  手順書に従った方法でエンドユーザを管理したかどうかを監査するために，アプリケーションサーバが提供する監査ログ出力用の API を使用して J2EE アプリケーションを実装して，J2EE アプリケーションの監査ログを出力するようにします。監査ログ出力用の API を使用した J2EE アプリケーションの実装方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.8　アプリケーションの監査ログを出力するための実装」を参照してください。

エンドユーザを対象にした対策

- **J2EE アプリケーションの監査ログ出力**

  正当なエンドユーザが手順書に従った方法でサービスを利用したかどうかを監査

するために，アプリケーションサーバが提供する監査ログ出力用の API を使用して J2EE アプリケーションを実装して，J2EE アプリケーションの監査ログを出力するようにします。監査ログ出力用の API を使用した J2EE アプリケーションの実装方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.8　アプリケーションの監査ログを出力するための実装」を参照してください。

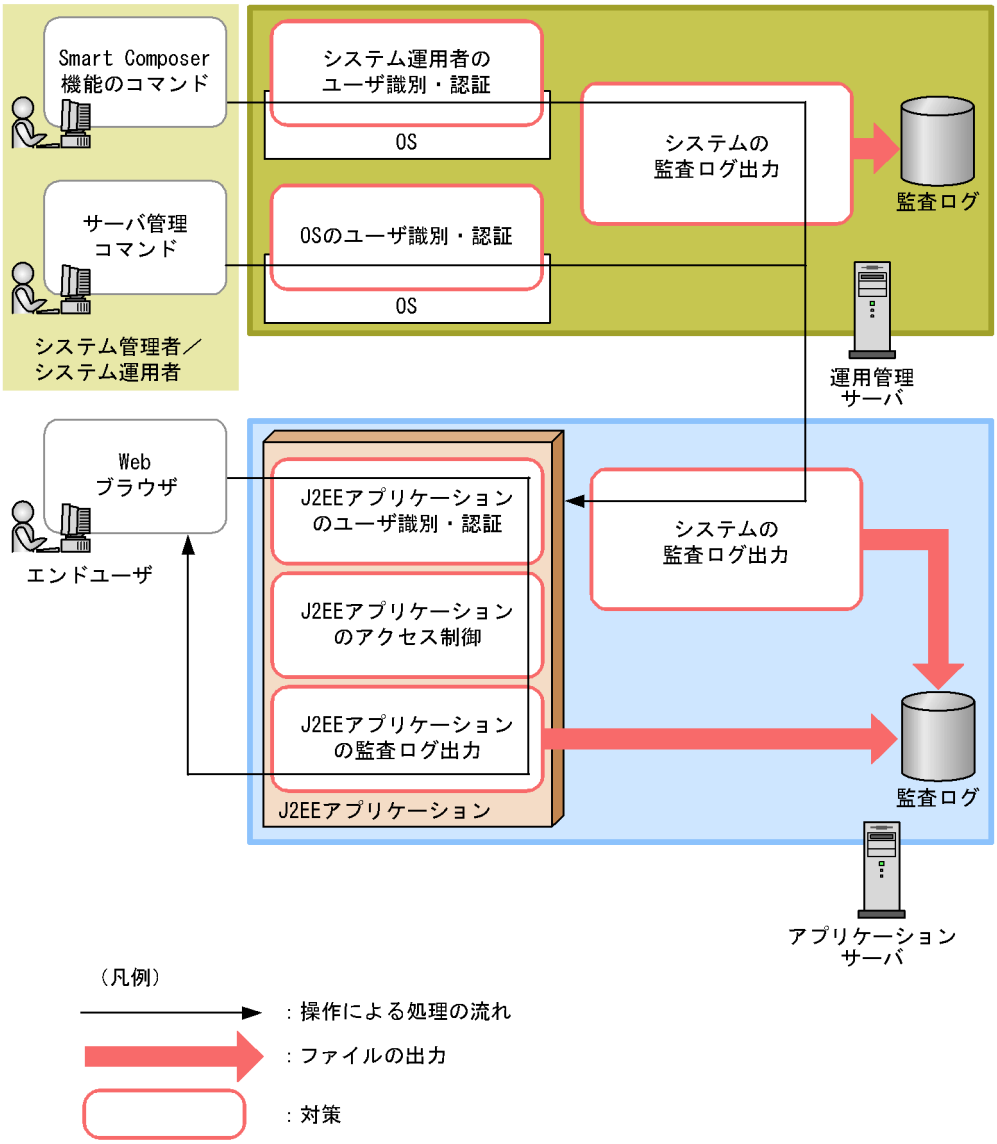
- J2EE **アプリケーションのユーザ識別・認証**
  正当なエンドユーザだけがサービスを利用するように，J2EE アプリケーションにユーザ識別・認証機能を実装します。
- J2EE **アプリケーションのアクセス制御**
  アクセス権限を持っているエンドユーザだけが保護するべきデータにアクセスできるように，J2EE アプリケーションにアクセス制御機能を実装します。

10.7.3　対策を実施したセキュアなシステムの動作

ここでは，対策を実施したセキュアなシステムの動作について説明します。

対策を実施したシステムの動作の概要を次の図に示します。なお，図中に示した対策は，「10.7.2　想定した脅威に対して実施する対策」で説明した対策と対応しています。

図 10-3　システム管理者の操作とシステムの動作



この図で示した，対策を実施したシステムの動作の概要を，システムの使用者ごとに説明します。

(1) システム管理者・システム運用者の操作とシステムの動作

システム管理者・システム運用者の操作とシステムの動作の概要について説明します。

システム管理者の操作

- Smart Composer 機能のコマンドを使用してアプリケーションサーバを構築します。ただし，J2EE アプリケーションの設定，リソースの設定などの作業は，サー

バ管理コマンドを使用して実施します。
- ユーザ識別・認証機能とアクセス制御機能，および監査ログを出力する機能を実装したアプリケーションを，デプロイ・開始します。

システム運用者の操作

Smart Composer 機能のコマンドを使用してシステムを運用します。ただし，障害発生時のログの収集は，snapshotlog コマンドを使用して実施します。

システムの動作

コマンドの実行によるそれぞれの操作に対して監査ログを出力します。

ポイント

一部のコマンドは監査ログを出力しないため，使用するコマンドが監査ログを出力するコマンドかどうかを確認して作業を実施してください。監査ログを出力するコマンドについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.6　監査ログを出力するコマンド・操作一覧」を参照してください。

(2) エンドユーザの操作とシステムの動作

システムが提供するサービス利用時のエンドユーザの操作とシステム（J2EE アプリケーション）の動作の概要を次に示します。

エンドユーザの操作

クライアント端末の Web ブラウザから HTTP リクエストをアプリケーションサーバへ送信します。

システム（J2EE アプリケーション）の動作

- HTTP リクエストに含まれているユーザ情報を識別して，ユーザ認証をします。
- アクセス制御機能を使用して，認証されたユーザにアクセス権があるかどうかをチェックします。
- アクセス制御機能で許可されたリクエストが J2EE アプリケーションのサービスを実行します。
- 処理の実行時に，監査ログを出力します。

10.8　作業手順の検討

この節では，セキュアなシステムを構築・運用するために，システムごとに検討する必要がある作業手順について説明します。

セキュアなシステムを構築・運用するには，それぞれの作業者が実施する作業の手順を明確にする必要があります。システムの監査では，監査ログと作業手順書を照合して，それぞれに矛盾がないかを調査します。**作業手順書**とは，システム構築，システム運用，エンドユーザによるサービスの利用などの作業について，正しい手順，方法を明文化した文書です。作業手順書は，それぞれのシステムに応じて作成する必要があります。作業者や操作内容の記録である監査ログと作業手順書を照合することで，正しい作業者が正しい手段・手順で作業を実施したかどうかが明確になります。これによって，システムの安全性を保てます。

作業手順書の作成では，それぞれの作業手順書に記載する作業とその手順・方法を検討する必要があります。監査の観点で，正しいユーザが正しい手順・方法で実施するように規定する必要がある作業を洗い出してください。また，それぞれの作業の手順では，必ず監査ログを出力するコマンドを使用して作業を実行するように規定してください。

10.8.1　作成する作業手順書の概要

ここでは，作成する必要がある作業手順書の概要について説明します。

作成する作業手順書は，システムによって異なります。このシステムでは，次の作業手順書を作成します。

- **入退室手順書**
  施錠管理がされているサーバエリアに入退室する手順が規定された文書です。
- **システム構築手順書**
  システムを構築する手順が規定された文書です。記載する手順は，「10.8.2　システムの構築手順の検討」および「10.8.3　システムの再構築手順の検討」で説明している手順を基に作成します。
  なお，「10.8.2　システムの構築手順の検討」，および「10.8.3　システムの再構築手順の検討」で説明している手順では，システム管理者は Smart Composer 機能のコマンドおよびサーバ管理コマンドを使用して操作を実行しています。また，すべての操作に，監査ログが出力されるコマンドを使用しています。
- **システム運用手順書**
  システムを運用する手順が規定された文書です。記載する手順は，「10.8.4　システムの運用手順の検討」で説明している手順を基に作成します。
  なお，「10.8.4　システムの運用手順の検討」で説明している手順では，システム運用者は Smart Composer 機能のコマンド，および snapshotlog コマンドを使用して操作を実行しています。
- **エンドユーザ操作手順書**
  システムが提供するサービスを利用する手順が規定された文書です。

10.8.2 システムの構築手順の検討

ここでは，「システム構築手順書」に記載するシステムの構築手順の例について説明します。「システム構築手順書」を作成するときは，ここで説明している手順を参考にしてください。

セキュアなシステムの構築には，Smart Composer 機能のコマンド，およびサーバ管理コマンドを使用します。また，すべての操作に，実行時に監査ログが出力されるコマンドを使用します。ここで説明している以外の操作を作業手順書に記載する場合も，監査ログが出力されるコマンドを使用する方法を記載してください。監査ログが出力されるコマンドについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.6　監査ログを出力するコマンド・操作一覧」を参照してください。

なお，この項で説明する手順は，すべてシステム管理者が実行します。

(1) ハードウェアの設置

システム管理者が，ハードウェアを設置します。ハードウェアを設置する手順を次に示します。

1. **「入退室手順書」に従って，外部から物理的に隔離されたサーバエリアに入室します。**
2. **システムが稼働するハードウェア，およびファイアウォールをサーバエリア内に設置します。**

なお，「システム構築手順書」では，ハードウェア，およびファイアウォールを設置する手順の詳細を規定してください。

(2) OS のインストール

システム管理者が，システムで使用する OS のインストールを実施します。OS をインストールする手順を次に示します。

1. **OS のインストール，および IP アドレス・ホスト名の設定などの，ネットワークの接続に必要な設定をします。**
2. **システムに必要なセキュリティパッチを適用します。**
3. **システムに必要なソフトウェアをインストールして，環境変数を設定します。**
4. **システム管理者用の OS のアカウントを作成して，管理者権限を付与します。**
5. **監査者用の OS のアカウントを作成して，安全な手段で監査者へ通知します。**

(3) システムの管理の開始

システム管理者が，「(2) OS のインストール」で設定したシステム管理者用の OS のアカウントを使用して OS にログインします。

(4) 監査ログを出力するための設定

システム管理者が，運用管理サーバおよびアプリケーションサーバの端末で監査ログを出力するための設定をします。監査ログを出力するための設定をする手順を次に示します。

1. **システム構成を基に，ログファイルのサイズを決定します。**
2. **システム管理者，およびシステム運用者に，監査ログファイルに対する読み取り権限と書き込み権限を設定します。また，監査者に監査ログファイルに対する読み取り権限を設定します。**
3. **手順** 1. **および手順** 2. **で決定，および設定した内容を監査ログ定義ファイル（**auditlog.properties**）に反映します。**
4. **監査ログ定義ファイルに指定した監査ログ出力ディレクトリを作成します。**
5. **システム管理者，およびシステム運用者に，手順** 4. **で作成した監査ログ出力ディレクトリに対する読み取り権限と書き込み権限を設定します。また，監査者に，手順** 4. **で作成した監査ログ出力ディレクトリに対する読み取り権限を設定します。**
6. **セットアップコマンド（**auditsetup **コマンド）を実行します。**

(5) 負荷分散機およびデータベースの設定

システム管理者が，負荷分散機およびデータベースをサーバエリア内に設置して，負荷分散機およびデータベースを設定します。

なお，「システム構築手順書」では，負荷分散機およびデータベースの設定手順の詳細を規定してください。

(6) 運用管理サーバの設定

システム管理者が，運用管理サーバの初期設定をします。運用管理サーバを設定する手順を次に示します。

1. mngsvrctl **コマンドで，引数「**setup**」を指定して，**Management Server **のセットアップ，および** Management Server **の管理ユーザアカウントの設定をします。**
2. mngautorun **コマンドで，引数「**server**」を指定して，**Management Server **の自動起動の設定をします。**

(7) Web システムの構成定義

システム管理者が，Web システムの構成を定義します。Web システムの構成を定義する手順を次に示します。

1. mngsvrctl **コマンドで，引数「**start**」を指定して，**Management Server **を起動します。**
2. **簡易構築定義ファイルを編集して保存します。**

3. adminagentctl コマンドで引数「start」を指定して，それぞれのアプリケーションサーバで運用管理エージェントを起動します。
4. 運用管理サーバで cmx_build_system コマンドを使用して，Web システムを構築します。

(8) Web システムの準備

システム管理者が，運用管理サーバの管理者端末で Smart Composer 機能のコマンドを使用して，Web システムの準備をします。Web システムを準備する手順を次に示します。

1. cmx_start_target コマンドを使用して，Web システムを準備状態にします。
2. cmx_list_status コマンドを使用して，Web システム内のサービスユニットが準備状態であることを確認します。

(9) リソースアダプタの設定

システム管理者が，運用管理サーバの管理者端末でサーバ管理コマンドを使用して，データベースと連携するアプリケーションに必要なリソースアダプタを設定します。リソースアダプタを設定する手順を次に示します。

1. 次のディレクトリから，使用するリソースアダプタの Connector 属性ファイルのテンプレートをコピーします。

   Windows の場合

   <Cosminexus インストールディレクトリ >¥CC¥admin¥templates¥

   UNIX の場合

   /opt/Cosminexus/CC/admin/templates/

2. 手順 1. でコピーしてきた Connector 属性ファイルのテンプレートを編集します。
3. 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。
4. cjsetresprop コマンドを使用して，Connector 属性ファイルの編集内容をリソースアダプタに反映します。
5. 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。
6. 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。

(10) J2EE アプリケーションの確認

システム管理者が，「10.7.2　想定した脅威に対して実施する対策」で説明した対策が J2EE アプリケーションに施されているかを確認します。ここでは，「10.7.2　想定した脅威に対して実施する対策」で説明した対策のうち，次の対策について確認します。

- J2EE アプリケーションの監査ログ出力
- J2EE アプリケーションのユーザ識別・認証

- J2EE アプリケーションのアクセス制御

具体的には，J2EE アプリケーションが次の仕様を満たしているかどうかを確認してください。

- システム運用者によってエンドユーザのユーザ ID・パスワードを登録，および削除するための機能がある。
- ユーザ ID・パスワードの識別・認証機能がある。
- 提供するサービスに対するアクセス制御機能がある。
- ユーザがサービスを利用するときに，監査ログを出力する機能がある。

（11）J2EE アプリケーションの設定

システム管理者が，運用管理サーバの管理者端末でサーバ管理コマンドを使用して，J2EE アプリケーションを設定します。J2EE アプリケーションを設定する手順を次に示します。

1. cjimportapp **コマンドを使用して，**J2EE **アプリケーションをインポートします。**
2. cjgetappprop **コマンドを使用して，アプリケーション統合属性ファイルを取得します。**
3. **手順** 2. **で取得したアプリケーション統合属性ファイルを編集します。**
4. cjsetappprop **コマンドを使用して，アプリケーション統合属性ファイルの編集内容を** J2EE **アプリケーションに反映します。**

注意事項

ここでは，実行時情報を持たない J2EE アプリケーションを設定する手順について説明しています。実行時情報を持つ J2EE アプリケーションを設定する場合は，手順 1. で J2EE アプリケーションをインポートしたあとに，cjstopapp コマンドを使用して J2EE アプリケーションを停止してから，手順 2. に進んでください。

（12）Web システムの開始

システム管理者が，運用管理サーバの管理者端末で Smart Composer 機能のコマンドとサーバ管理コマンドを使用して，Web システムを開始します。Web システムを開始する手順を次に示します。

1. cjstartrar **コマンドを使用して，リソースアダプタを開始します。**
2. cjstartapp **コマンドを使用して，**J2EE **アプリケーションを開始します。**
3. cmx_start_target **コマンドを使用して，**Web **システム内のサービスユニットを稼働状態にします。**

(13) 不要な機能の無効化

不正なユーザによって不要な機能を使用されないように，機能を無効化します。具体的には，システム管理者が，コマンドの実行権限を変更したり，コマンドの実行に必要なファイルを削除したりします。無効化する必要がある機能を，Windows の場合と UNIX の場合に分けて，次の表に示します。

表 10-3 無効化する必要がある機能（Windows の場合）

機能名	対象ディレクトリ	対象ファイル	対処
Hitachi Web Server の GUI サーバ管理機能	<Cosminexus のインストールディレクトリ >¥httpsd	adm-httpsd.exe	システム管理者以外の実行権限を解除します。
Hitachi Web Server のパスワードファイル編集コマンド	<Cosminexus のインストールディレクトリ >¥httpsd¥bin	htpasswd.exe	システム管理者以外の実行権限を解除します。
CTM のスケジュールキューの同時実行数の変更	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmchpara.exe	システム管理者以外の実行権限を解除します。
CTM の CTM ドメインの情報表示と削除	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmdminfo.exe	システム管理者以外の実行権限を解除します。
CTM のスケジュールキューの閉塞	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmholdque.exe	システム管理者以外の実行権限を解除します。
CTM の実行形式ファイルおよびライブラリのバージョン情報の出力	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmjver.exe	システム管理者以外の実行権限を解除します。
CTM のメッセージの編集と出力	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmlogcat.exe	システム管理者以外の実行権限を解除します。
CTM のスケジュールキュー情報の出力	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmlsque.exe	システム管理者以外の実行権限を解除します。
CTM のスケジュールキューの閉塞解除	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmrlesque.exe	システム管理者以外の実行権限を解除します。
CTM の稼働統計情報の編集と出力	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmstsed.exe	システム管理者以外の実行権限を解除します。
CTM のバッファ内容の強制ファイル出力	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmstsflush.exe	システム管理者以外の実行権限を解除します。
CTM の実行形式ファイルおよびライブラリのバージョン情報の出力	<Cosminexus のインストールディレクトリ >¥CTM¥bin	ctmver.exe	システム管理者以外の実行権限を解除します。

機能名	対象ディレクトリ	対象ファイル	対処
PRF の性能解析トレース情報の編集出力	<Cosminexus のインストールディレクトリ >¥PRF¥bin	cprfed.exe	システム管理者以外の実行権限を解除します。
PRF のバッファ内容の強制ファイル出力	<Cosminexus のインストールディレクトリ >¥PRF¥bin	cprfflush.exe	システム管理者以外の実行権限を解除します。
PRF トレース取得レベルの表示と変更	<Cosminexus のインストールディレクトリ >¥PRF¥bin	cprflevel.exe	システム管理者以外の実行権限を解除します。
Management Server で使用するコマンド	<Cosminexus のインストールディレクトリ >¥manager¥bin	mngsvrutil.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	mstrexport.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	mstrimport.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	ssoexport.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	ssogenkey.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	ssoimport.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	uachpw.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	mngsvr_adapter_setup.exe	このコマンドを実行しないようにします。
	<Cosminexus のインストールディレクトリ >¥manager¥bin	Adapter_HITACHI_COSMINEXUS_MANAGER.exe	システム管理者以外の実行権限を解除します。
	<Cosminexus のインストールディレクトリ >¥manager¥externals¥jp1¥mngsvrmonitor	mngsvr_monitor_setup.exe	このコマンドを実行しないようにします。
運用管理ポータル	<Cosminexus のインストールディレクトリ >¥manager¥containers¥m¥webapps¥mngsvr	index.jsp	ファイルを削除します。

機能名	対象ディレクトリ	対象ファイル	対処
	<Cosminexus のインストールディレクトリ>¥manager¥containers¥m¥webapps¥mngsvr	login.jsp	ファイルを削除します。
Server Plug-in	<Cosminexus のインストールディレクトリ>¥manager¥config	mserver.properties	Server Plug-in の設定をしないようにします。

表 10-4　無効化する必要がある機能（UNIX の場合）

機能名	対象ディレクトリ	対象ファイル	対処
Hitachi Web Server の GUI サーバ管理機能	/opt/hitachi/httpsd/sbin	adminctl	システム管理者以外の実行権限を解除します。
	/opt/hitachi/httpsd/sbin	adm-httpsd	システム管理者以外の実行権限を解除します。
Hitachi Web Server のパスワードファイル編集コマンド	/opt/hitachi/httpsd/bin	htpasswd	システム管理者以外の実行権限を解除します。
CTM のスケジュールキューの同時実行数の変更	/opt/Cosminexus/CTM/bin	ctmchpara	システム管理者以外の実行権限を解除します。
CTM の CTM ドメイン情報の表示と削除	/opt/Cosminexus/CTM/bin	ctmdminfo	システム管理者以外の実行権限を解除します。
CTM のスケジュールキューの閉塞	/opt/Cosminexus/CTM/bin	ctmholdque	システム管理者以外の実行権限を解除します。
CTM の実行形式ファイルおよびライブラリのバージョン情報の出力	/opt/Cosminexus/CTM/bin	ctmjver	システム管理者以外の実行権限を解除します。
CTM のメッセージの編集と出力	/opt/Cosminexus/CTM/bin	ctmlogcat	システム管理者以外の実行権限を解除します。
CTM のスケジュールキュー情報の出力	/opt/Cosminexus/CTM/bin	ctmlsque	システム管理者以外の実行権限を解除します。
CTM のスケジュールキューの閉塞解除	/opt/Cosminexus/CTM/bin	ctmrlesque	システム管理者以外の実行権限を解除します。
CTM の稼働統計情報の編集と出力	/opt/Cosminexus/CTM/bin	ctmstsed	システム管理者以外の実行権限を解除します。
CTM のバッファ内容の強制ファイル出力	/opt/Cosminexus/CTM/bin	ctmstsflush	システム管理者以外の実行権限を解除します。

機能名	対象ディレクトリ	対象ファイル	対処
CTM の実行形式ファイルおよびライブラリのバージョン情報の出力	/opt/Cosminexus/CTM/bin	ctmver	システム管理者以外の実行権限を解除します。
PRF の性能解析トレース情報の編集出力	/opt/Cosminexus/PRF/bin	cprfed	システム管理者以外の実行権限を解除します。
PRF のバッファ内容の強制ファイル出力	/opt/Cosminexus/PRF/bin	cprfflush	システム管理者以外の実行権限を解除します。
PRF トレース取得レベルの表示と変更	/opt/Cosminexus/PRF/bin	cprflevel	システム管理者以外の実行権限を解除します。
Management Server で使用するコマンド	/opt/Cosminexus/manager/bin	mngsvrutil	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	mstrexport	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	mstrimport	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	ssoexport	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	ssogenkey	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	ssoimport	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	uachpw	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	mngsvr_adapter_setup	システム管理者以外の実行権限を解除します。
	/opt/Cosminexus/manager/bin	Adapter_HITACHI_COSMINEXUS_MANAGER	システム管理者以外の実行権限を解除します。
運用管理ポータル	/opt/Cosminexus/manager/containers/m/webapps/mngsvr	index.jsp	ファイルを削除します。
	/opt/Cosminexus/manager/containers/m/webapps/mngsvr	login.jsp	ファイルを削除します。
Server Plug-in	/opt/Cosminexus/manager/config	mserver.properties	Server Plug-in の設定をしないようにします。

(14) システム運用者の登録

システム管理者が，運用管理サーバの管理者端末で OS の機能，および Smart

Composer 機能のコマンドを使用して，システム運用者のユーザ ID とパスワードを設定します。また，設定したユーザ ID とパスワードをシステム運用者に通知します。システム運用者を登録する手順を次に示します。

1. **OS の機能を使用して，システム運用者の** OS **のユーザ** ID **とパスワードを設定します。**
2. **OS の機能を使用して，システム運用者に管理者権限を与えないよう設定します。**
3. cmx_admin_passwd **コマンドを使用して，システム管理者の** Management Server **の管理ユーザ** ID **と管理パスワードを，システム運用者の** Management Server **の管理ユーザ** ID **と管理パスワードに変更します。**
4. **手順** 1. **および手順** 3. **で設定したユーザ** ID **とパスワードを安全な手段でシステム運用者へ通知します。**

10.8.3 システムの再構築手順の検討

ここでは，「システム構築手順書」に記載するシステムの再構築手順の例について説明します。「システム構築手順書」を作成するときは，ここで説明している手順を参考にしてください。

セキュアなシステムの再構築には，Smart Composer 機能のコマンド，およびサーバ管理コマンドを使用します。また，すべての操作に，実行時に監査ログが出力されるコマンドを使用します。ここで説明している以外の操作を作業手順書に記載する場合も，監査ログが出力されるコマンドを使用する方法を記載してください。監査ログが出力されるコマンドについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.6　監査ログを出力するコマンド・操作一覧」を参照してください。

なお，この項で説明する手順は，すべてシステム管理者が実行します。

(1) J2EE アプリケーションの入れ替え

システム管理者が，メンテナンスが必要な場合などに 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入れ替える手順を次に示します。

1. cmx_stop_target **コマンドを使用して，**Web **システム内のサービスユニットを準備状態にします。**
2. cjstopapp **コマンドを使用して，入れ替え前の** J2EE **アプリケーションを停止します。**
3. cjdeleteapp **コマンドを使用して，入れ替え前の** J2EE **アプリケーションを削除します。**
4. cjimportapp **コマンドを使用して，入れ替え後の** J2EE **アプリケーションをインポートします。**

5. cjgetappprop **コマンドを使用して，入れ替え後の** J2EE **アプリケーションのアプリケーション統合属性ファイルを取得します。**
6. **手順** 5. **で取得したアプリケーション統合属性ファイルを編集して，**J2EE **アプリケーションに必要な情報を設定します。また，必要に応じて** J2EE **アプリケーションをカスタマイズします。**
7. cjsetappprop **コマンドを使用して，手順** 6. **で編集したアプリケーション統合属性ファイルを，入れ替え後の** J2EE **アプリケーションに反映します。**
8. cjstartapp **コマンドを使用して，入れ替え後の** J2EE **アプリケーションを開始します。**
9. cmx_start_target **コマンドを使用して，**Web **システム内のサービスユニットを稼働状態にします。**

なお，J2EE アプリケーションの入れ替えには，この手順のほかに，cjreplaceapp コマンドを使用したリデプロイ機能，または cjreloadapp コマンドを使用したリロード機能を使用することもできます。

（2）システムのチューニング

システム管理者が，必要に応じてシステムをチューニングします。システムをチューニングする手順を次に示します。

1. **簡易構築定義ファイルを編集します。**
2. cmx_stop_target **コマンドを使用して，**Web **システム内のサービスユニットを停止します。**
3. cmx_build_system **コマンドを使用して，**Web **システムの設定を変更します。**
4. cmx_start_target **コマンドを使用して，**Web **システム内のサービスユニットを起動します。**

（3）システム構成の変更（サービスユニットの追加）

システム管理者が，必要に応じてサービスユニットを追加してシステム構成を変更します。サービスユニットを追加してシステム構成を変更する手順を次に示します。

1. **構成変更定義ファイルを作成・編集します。**
2. cmx_change_model **コマンドを使用して，**Management Server **の** Web **システムの情報モデルを変更します。**
3. cmx_build_system **コマンドを使用して，変更した** Web **システムの情報モデルを適用します。**
4. cmx_start_target **コマンドを使用して，追加する** Web **システム内のサービスユニットを準備状態にします。**
5. cjstartrar **コマンドを使用して，リソースアダプタを開始します。**

6. cjstartapp **コマンドを使用して，**J2EE **アプリケーションを開始します。**
7. cmx_start_target **コマンドを使用して，追加する** Web **システム内のサービスユニットを稼働状態にします。**

（4） システム構成の変更（サービスユニットの削除）

システム管理者が，必要に応じてサービスユニットを削除してシステム構成を変更します。サービスユニットを削除してシステム構成を変更する手順を次に示します。

1. cmx_stop_target **コマンドを使用して，削除する** Web **システム内のサービスユニットを停止します。**
2. cmx_delete_system **コマンドを使用して，手順** 1. **で指定した** Web **システム内のサービスユニットを削除します。**

10.8.4 システムの運用手順の検討

ここでは，「システム運用手順書」に記載するシステムの運用手順の例について説明します。「システム運用手順書」を作成するときは，ここで説明している手順を参考にしてください。

セキュアなシステムの運用には，Smart Composer 機能のコマンド，および snapshotlog コマンドを使用します。ここで説明している以外の操作を作業手順書に記載する場合も，監査ログが出力されるコマンドを使用する方法を記載してください。監査ログが出力されるコマンドについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.6　監査ログを出力するコマンド・操作一覧」を参照してください。

なお，この項で説明する手順のうち，Web システムの起動の作業の一部，および Web システムのメンテナンス以外は，システム運用者が実行します。Web システムのメンテナンスは，システム運用者に依頼されたシステム管理者が実行します。

（1） Web システムの起動

システム運用者が，運用管理サーバの管理者端末で Web システムを起動します。ただし，リソースアダプタおよび J2EE アプリケーションの起動は，システム管理者が，サーバ管理コマンドを使用して実行します。Web システムを起動する手順を次に示します。

1. **システム運用者が，**cmx_start_target **コマンドを使用して，**Web **システム内のサービスユニットを準備状態にします。**
2. **システム運用者が，システム管理者にサービスの起動を依頼します。**
3. **システム管理者が，**cjstartrar **コマンドを使用して，リソースアダプタを開始します。**
4. **システム管理者が，**cjstartapp **コマンドを使用して，**J2EE **アプリケーションを開始し**

ます。

5. **システム運用者が，**cmx_start_target **コマンドを使用して，**Web **システム内のサービスユニットを稼働状態にします。**

（2） Web システムの停止

システム運用者が，運用管理サーバの管理者端末で Smart Composer 機能のコマンドを使用して，Web システムを停止します。Web システムを停止する手順を次に示します。

1. cmx_stop_target **コマンドを使用して，**Web **システム内のサービスユニットを停止状態にします。**

（3） エンドユーザの管理

システム運用者が，「システム運用手順書」に従って，エンドユーザのアクセス権やユーザ ID などを管理します。次の作業を実施します。

- エンドユーザの登録，削除
- エンドユーザの権限の変更
- エンドユーザのパスワード変更

なお，「システム運用手順書」では，これらの手順の詳細を規定してください。

（4） エンドユーザへの通知

システム運用者が，「(3) エンドユーザの管理」で登録したエンドユーザのユーザ ID とパスワードをエンドユーザに通知します。ユーザ ID とパスワードをエンドユーザに通知する手順を次に示します。

1. **「システム運用手順書」に従って登録したエンドユーザのユーザ** ID **とパスワードを使用して，サービスを利用できることを確認します。**
2. **サービス利用時に，監査ログが出力されることを確認します。**
3. **手順** 1. **および手順** 2. **の確認が終わったら，システム運用者がエンドユーザにユーザ** ID **とパスワードを安全な手段で通知します。**

（5） Web システムのメンテナンス

必要に応じて Web システムをメンテナンスします。Web システムのメンテナンスは，システム運用者に依頼されたシステム管理者が実施します。Web システムをメンテナンスする手順を次に示します。

1. **システム運用者が，**cmx_stop_target **コマンドを使用して，メンテナンスするサービスユニットを閉塞，または停止します。**
2. **アプリケーションサーバの修正パッチを適用する場合，システム運用者がアプリケーションサーバ関連のプログラムを停止します。修正パッチを適用しない場合は，手順**

3. に進みます。

3. システム管理者にシステムのメンテナンスを依頼します。
4. システム管理者が，システム管理者用のユーザ ID で，OS にログインします。
5. システム管理者が OS のサービスパック，セキュリティパッチ，アプリケーションサーバの修正パッチなどを適用します。この作業をシステム管理者が実施しているとき，システム運用者はその場に立ち会う必要があります。
6. アプリケーションサーバの修正パッチを適用した場合，システム運用者がアプリケーションサーバ関連のプログラムを再開します。
7. システム運用者が，cmx_start_target コマンドを使用して，サービスユニットを再開します。

(6) システムの障害対応

障害発生時に，システム運用者が障害に対応します。システムの障害に対応する手順を次に示します。

1. snapshotlog コマンドを使用して，アプリケーションサーバのログを収集します。
2. 必要に応じて，snapshotlog コマンドで収集できないログを個別に収集します。
3. 収集したログを保守員に送付して，調査を依頼します。
4. 調査の結果に基づいてシステムをメンテナンスします。

10.9　システムの監査方法の確認

この節では，システムの監査方法について説明します。

システムの監査では，監査者がそれぞれの作業手順書と監査ログに出力されている操作の記録を照合して，正しい作業者が正しい手順で作業を実行しているかどうかを調査します。

10.9.1　監査ログの入手

監査ログを入手する手順について次に示します。

1. **監査者が，「入退室手順書」に従って外部から物理的に隔離されたサーバエリアに入室します。**
2. **監査者が，監査ログサーバにログインして，各種サーバが稼働しているマシンから監査ログを入手します。**

10.9.2　監査ログの調査

監査ログの調査では，次の点を確認します。

1. **信頼できるシステム管理者が正しい方法でシステムを構築しているか**
   監査ログに出力された時刻，操作者，事象内容，および結果と，「システム構築手順書」の内容に相違がないことを確認します。
2. **正しい方法で構築されたシステムが，正しく運用，および利用されているか**
   監査ログに出力された時刻，操作者，事象内容，および結果と，「システム運用手順書」および「エンドユーザ操作手順書」に相違がないことを確認します。

監査ログの調査方法，および監査ログに出力されるメッセージの詳細については，次のマニュアルを参照してください。

● 監査ログの調査方法

マニュアル「Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編」の「6.3　監査ログとは」を参照してください。

● 監査ログに出力されるメッセージの詳細

マニュアル「Cosminexus アプリケーションサーバ メッセージ 4」を参照してください。

10.10 外部ネットワークを使用するシステムでのセキュリティの検討

この節では，外部ネットワークを使用するシステムで想定されるセキュリティ上の脅威とその対策について説明します。

10.10.1 外部ネットワークを使用するシステムで想定されるセキュリティ上の脅威

ここでは，外部ネットワークを使用するシステムで想定されるセキュリティ上の脅威について説明します。

(1) 想定するセキュリティ上の脅威

ネットワークを使用するシステムでは，セキュリティ対策が十分でないと，アプリケーションを不正に実行されたり，通信内容やバックエンドのデータベースで管理しているデータが漏洩または改ざんされたりするおそれがあります。これらを防ぐためには，セキュリティ上の脅威を認識して，それらに十分に対策しておく必要があります。

ここでは，次のようなセキュリティ上の脅威を想定します。

- システム外からシステムへの第三者の不正侵入
- アプリケーションの扱うデータの第三者への漏洩
- アプリケーションの通信内容の第三者への漏洩
- アプリケーションの通信内容の第三者による改ざん
- システム利用者の許可された権限の範囲を超えた操作や情報の入手

なお，ここでは，システム外部からの脅威を想定した対策について検討します。システム内部の脅威については，ここでは対象にしません。

(2) 考えられる対策

想定したセキュリティ上の脅威に対しては，それぞれ，次の表に示すような対策が考えられます。それぞれの具体的な対策方法については，参照先の説明を参照してください。

表 10-5 セキュリティ上の脅威に対して考えられる対策

脅威	対策	参照先
システム外からシステムへの第三者の不正侵入	ファイアウォールと侵入検知システムを配置する	10.10.2
アプリケーションの扱うデータの第三者への漏洩		
アプリケーションの通信内容の第三者への漏洩	通信内容を暗号化する	10.10.3 ※

脅威	対策	参照先
アプリケーションの通信内容の第三者による改ざん		
システム利用者の許可された権限の範囲を超えた操作や情報の入手	アプリケーションでユーザを認証する	10.10.4

注※　通信内容の暗号化には，HTTPS を使用します。参照先では，HTTPS を使用する場合に，SSL アクセラレータを使用して暗号通信を処理する方法について説明します。

10.10.2　ファイアウォールと侵入検知システムを配置する

ここでは，ファイアウォールと侵入検知システムを適切に配置，設定することでシステムのセキュリティを向上させる方法について説明します。

(1) ファイアウォールと侵入検知システムを配置する目的

ファイアウォールは，外部のネットワークと内部のネットワーク間のアクセスを制御します。あらかじめアクセスを許可するクライアントや通信を特定し，決められたルールに従って通信を許可したり破棄したりすることで，外部ネットワークからの不正なアクセスを防止します。このため，ファイアウォールを使用する場合は，通信を許可するポートや IP アドレスを明確にして，設定しておく必要があります。

侵入検知システム（IDS）は，通信回線を監視して，通信パターンによって不正なアクセスを判断します。

ファイアウォールと侵入検知システムを適切な個所に配置，設定することで，次のようなセキュリティ上の脅威からシステムを守れます。

- システム外からシステムへの第三者の不正侵入
- アプリケーションの扱うデータの第三者への漏洩

ここでは，次の表に示すシステム構成ごとのファイアウォールと侵入検知システムの配置個所，および設定するときに留意することについて説明します。

表 10-6　ファイアウォールと侵入検知システムの配置を検討するためのシステム構成の分類

システム構成	説明	参照先
基本的な Web クライアント構成	1 台のアプリケーションサーバで構成されるシステム構成です。クライアントは Web ブラウザです。	10.10.2(2)
基本的な EJB クライアント構成	1 台のアプリケーションサーバで構成されるシステム構成です。クライアントは EJB クライアントアプリケーションです。	10.10.2(3)

システム構成	説明	参照先
サーバの層ごとにファイアウォールで区切る構成（アプリケーション集中型）	複数のアプリケーションサーバで構成されるシステム構成で，サーバの層ごとにファイアウォールで区切る構成です。アプリケーションはすべて同じ層のアプリケーションサーバ上で動作します。	10.10.2(4)
サーバの層ごとにファイアウォールで区切る構成（アプリケーション分散型）	複数のアプリケーションサーバで構成されるシステム構成で，サーバの層ごとにファイアウォールで区切る構成です。アプリケーションは異なる複数の層のアプリケーションサーバ上で動作します。	10.10.2(5)

なお，インターネットと接続するシステムの場合は，外部ネットワークから直接内部ネットワークのアプリケーションサーバにアクセスされないように，DMZ を確保して，リバースプロキシを使用した構成を検討することをお勧めします。

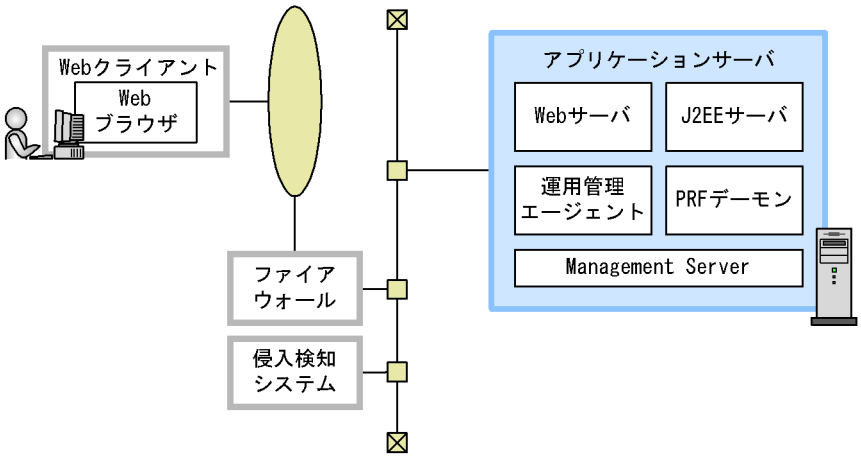
(2) 基本的な Web クライアント構成の場合

ここでは，1 台のアプリケーションサーバで構成される基本的な Web クライアント構成の場合のファイアウォールと侵入検知システムの配置個所について説明します。

ファイアウォールは，ネットワークから見てアプリケーションサーバの手前に配置します。この構成の場合，ネットワーク上の Web クライアントは，ファイアウォール経由でだけアプリケーションサーバにアクセスできます。

基本的な Web クライアント構成の場合のファイアウォールと侵入検知システムの配置例を次の図に示します。

図 10-4 基本的な Web クライアント構成でのファイアウォールと侵入検知システムの配置例



(a) アプリケーションサーバの設定

アプリケーションサーバでは，次の項目を設定します。

● J2EE サーバの管理用通信ポートのアドレス指定

J2EE サーバの管理用通信ポートにアクセスできるアドレスを指定します。

● Management Server および運用管理エージェントのアドレス指定

Management Server および運用管理エージェントにアクセスできるアドレスを指定します。

(b) ファイアウォールの設定

外部ネットワークとアプリケーションサーバ内の Web サーバ（Hitachi Web Server）のアクセスについて制御するために，次の項目を設定します。

● 外部ネットワークからの Web サーバに対してのアクセス許可

ファイアウォールの外部のネットワークとアプリケーションサーバ間の通信では，公開ポートである HTTP/80 や HTTPS/443 などだけを許可するようにします。ただし，システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

● アクセスを許可する Web クライアントの IP アドレスの限定（任意）

ファイアウォールの機能を使用してアクセスを許可する Web クライアントの IP アドレスを限定することで，そのシステムの利用者を特定できます。この場合は，ファイアウォールに通信を許可する IP アドレスを指定してください。

● Management Server および運用管理エージェントの通信ポートの指定

Management Server および運用管理エージェントの通信ポートについては，ファイアウォールの外部からはアクセスできないように通信を遮断してください。これらの通信ポートにアクセスできると，運用管理者以外の外部のユーザからアプリケーションサーバに対して不正な運用操作が実行されるおそれがあります。

(c) 侵入検知システムの設定

外部ネットワークとアプリケーションサーバ内の Web サーバ（Hitachi Web Server）の公開ポートの通信内容を監視するために，次の項目を設定します。

● 通信内容の監視

通信内容に既知の攻撃パターンや，攻撃と疑われるようなパターンが含まれている場合は運用管理者などに警告を通知するように設定します。侵入検知システムとファイアウォールの連携機能を利用して，疑わしい通信は自動的に遮断するように設定することもできます。

● SSL コネクションの確立部分への攻撃の監視

HTTPS によるアクセスは，通信内容が暗号化されているため，基本的に通信内容を監視できません。この場合は，HTTPS に関する既知の攻撃パターンとして，SSL コネクションの確立部分への攻撃などを監視対象としてください。

● 公開ポート以外のポートへの通信の監視

アプリケーションサーバの公開ポート以外のポートへの通信が外部ネットワークから送信されている場合は，設定ミスなどによってファイアウォールが突破されているお

それがあります。この場合，警告を通知するように設定することをお勧めします。

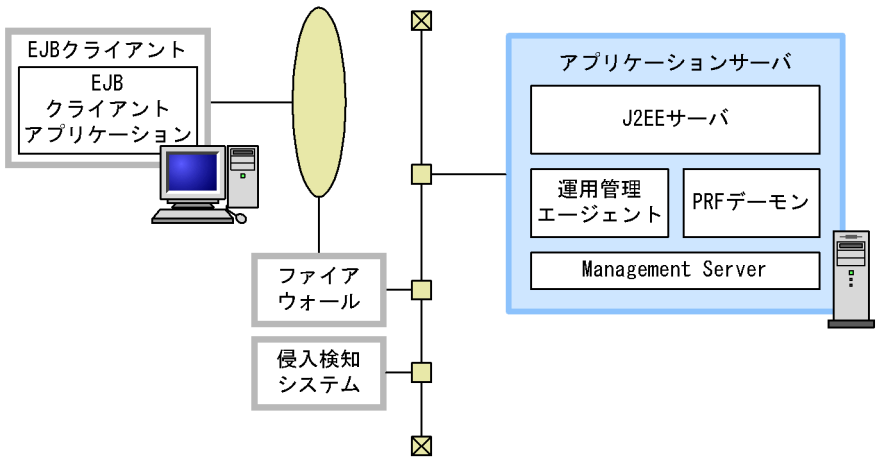
(3) 基本的な EJB クライアント構成の場合

ここでは，1 台のアプリケーションサーバで構成される基本的な EJB クライアント構成の場合のファイアウォールと侵入検知システムの配置個所について説明します。

ファイアウォールは，ネットワークから見てアプリケーションサーバの手前に配置します。この構成の場合，ネットワーク上の EJB クライアントは，ファイアウォール経由でだけアプリケーションサーバにアクセスできます。

基本的な EJB クライアント構成の場合のファイアウォールと侵入検知システムの配置例を次の図に示します。

図 10-5 基本的な EJB クライアント構成でのファイアウォールと侵入検知システムの配置



(a) アプリケーションサーバの設定

アプリケーションサーバでは，次の項目を設定します。

- **J2EE サーバの管理用通信ポートのアドレス指定**
  J2EE サーバの管理用通信ポートにアクセスできるアドレスを指定します。
- **Management Server および運用管理エージェントのアドレス指定**
  Management Server および運用管理エージェントにアクセスできるアドレスを指定します。
- **EJB クライアントからアクセスされるポート番号の固定**
  EJB クライアントからアプリケーションサーバを利用するために，EJB クライアントから次のポート番号に通信できるように設定してください。
  - CORBA ネーミングサービス
    ポート番号は，通常固定されています（デフォルトのポート番号：900）。

- EJB コンテナ
  ポート番号が固定されていないため，EJB コンテナが利用するポート番号を明示的に指定（固定）する必要があります。指定個所については，「3.18　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。

● Management Server **および運用管理エージェントの通信ポートの指定**

Management Server および運用管理エージェントの通信ポートの指定については，ファイアウォールの外部からはアクセスできないように，公開ポートを使用しないことを推奨します。これらの通信ポートにアクセスできると，運用管理者以外の外部のユーザからアプリケーションサーバに対して不正な運用操作が実行されるおそれがあります。

(b) ファイアウォールの設定

外部ネットワークとアプリケーションサーバ間のアクセスについて制御するために，次の項目を設定します。

● **外部ネットワークからアプリケーションサーバに対してのアクセス許可**

ファイアウォールの外部のネットワークとアプリケーションサーバ間の通信では，公開ポートである CORBA ネーミングサービスおよび EJB コンテナの固定ポートなどだけを許可するようにします。ただし，システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

● **アクセスを許可するクライアントの** IP **アドレスの限定（任意）**

ファイアウォールの機能を使用してアクセスを許可するクライアントの IP アドレスを限定することで，そのシステムの利用者を特定できます。この場合は，ファイアウォールに通信を許可する IP アドレスを指定してください。

● Management Server **および運用管理エージェントの通信ポートの指定**

Management Server および運用管理エージェントの通信ポートについては，ファイアウォールの外部からはアクセスできないように通信を遮断してください。これらの通信ポートにアクセスできると，運用管理者以外の外部のユーザからアプリケーションサーバに対して不正な運用操作が実行されるおそれがあります。

(c) 侵入検知システムの設定

外部ネットワークとアプリケーションサーバの公開ポートの通信内容を監視するために，次の項目を設定します。

● **通信内容の監視**

通信内容に既知の攻撃パターンや，攻撃と疑われるようなパターンが含まれている場合は運用管理者などに警告を通知するように設定します。侵入検知システムとファイアウォールの連携機能を利用して，疑わしい通信は自動的に遮断するように設定することもできます。

● SSL **コネクションの確立部分への攻撃の監視**

HTTPS によるアクセスは，通信内容が暗号化されているため，基本的に通信内容を

監視できません。この場合は，HTTPS に関する既知の攻撃パターンとして，SSL コネクションの確立部分への攻撃などを監視対象としてください。

● 公開ポート以外のポートへの通信の監視

アプリケーションサーバの公開ポート以外のポートへの通信が外部ネットワークから送信されている場合は，設定ミスなどによってファイアウォールが突破されているおそれがあります。この場合，警告を通知するように設定することをお勧めします。

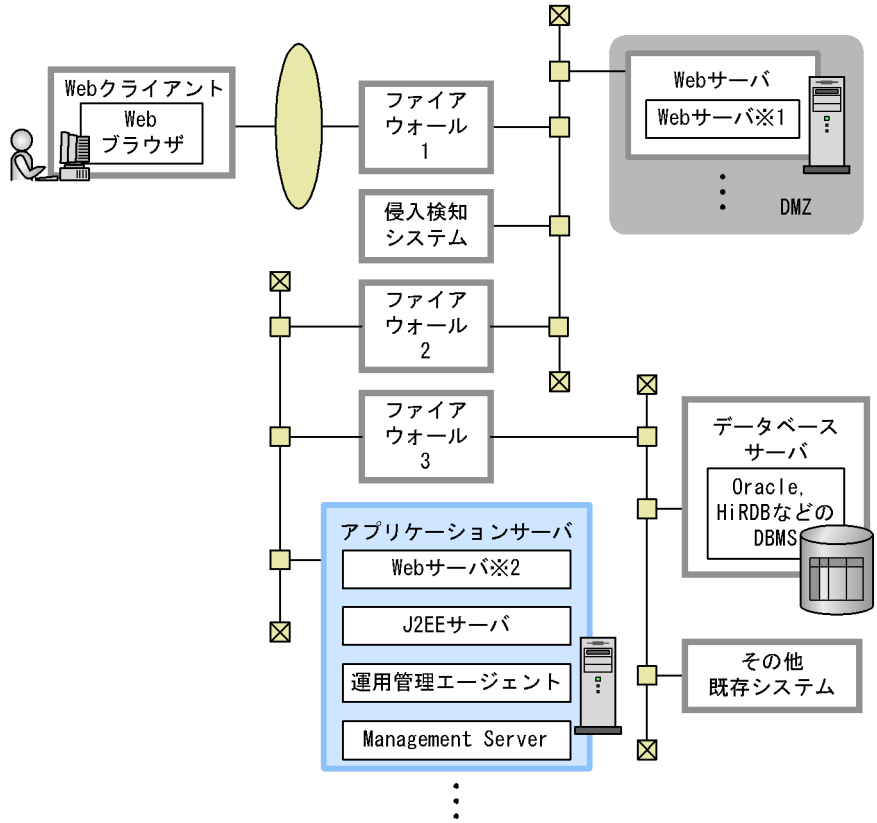
（4） サーバの層ごとにファイアウォールで区切る構成（アプリケーション集中型）

システムの規模によっては，一つのシステムが複数のアプリケーションサーバおよびそのほかのサーバで構成されることがあります。このような構成では，それぞれの層でセキュリティを確保する必要があります。

ここでは，Web サーバ，アプリケーションサーバおよびデータベースサーバを多層化した場合に，アプリケーションはすべて同じ層のアプリケーションサーバで動作させる構成について説明します。この構成を，**アプリケーション集中型の構成**といいます。

アプリケーション集中型の構成の場合のファイアウォールと侵入検知システムの配置例を次の図に示します。この構成では，サーバの層ごとに 1 台ずつ，合計 3 台のファイアウォールを配置しています。また，DMZ には，リバースプロキシモジュールを組み込んだ Web サーバ（リバースプロキシサーバ）を配置しています。

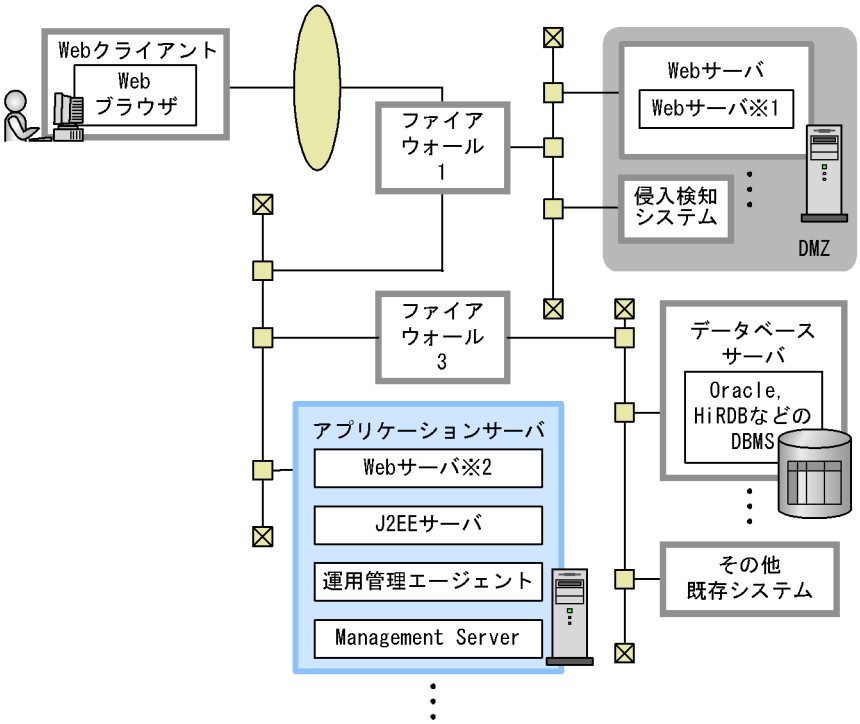
図 10-6　アプリケーション集中型のファイアウォールと侵入検知システムの配置



注※1　リバースプロキシモジュールを組み込んだWebサーバです。DMZに配置します。
注※2　リダイレクタモジュールを組み込んだWebサーバです。
　　アプリケーションサーバと同じマシンに配置します。

また，コストなどの要因からファイアウォール数を削減したい場合は，次の図に示すような構成にできます。この例では，ファイアウォール 1 とファイアウォール 2 で実行するアクセス制御をファイアウォール 1 に集約して，ファイアウォール 2 を削減します。

図 10-7　ファイアウォールを削減した構成



注※1　リバースプロキシモジュールを組み込んだWebサーバです。DMZに配置します。
注※2　リダイレクタモジュールを組み込んだWebサーバです。
　　アプリケーションサーバと同じマシンに配置します。

この構成の場合は，ファイアウォール 2 で設定する内容を，ファイアウォール 1 にまとめて設定してください。

(a) アプリケーションサーバの設定

アプリケーションサーバでは，次の項目を設定します。

- **J2EE サーバの管理用通信ポートのアドレス指定**
  J2EE サーバの管理用通信ポートにアクセスできるアドレスを指定します。
- **Management Server および運用管理エージェントのアドレス指定**
  Management Server および運用管理エージェントにアクセスできるアドレスを指定します。

(b) ファイアウォールの設定

この構成では，次に示す三つのファイアウォールを使用しています。

- ファイアウォール 1
  外部のネットワークと DMZ の Web サーバ（リバースプロキシサーバ）間のアクセスを制御します。

● ファイアウォール 2
DMZ の Web サーバ（リバースプロキシサーバ）と内部のネットワークにあるアプリケーションサーバ間のアクセスを制御します。

● ファイアウォール 3
アプリケーションサーバとデータベースサーバ間のアクセスを制御します。

それぞれのファイアウォールに設定する内容について説明します。

● ファイアウォール 1 の設定

外部のネットワークと DMZ の Web サーバ（リバースプロキシサーバ）間のアクセスを制御するためのファイアウォールです。次の項目を設定します。

● 外部ネットワークから Web サーバ（リバースプロキシサーバ）に対してのアクセス許可
ファイアウォール 1 よりも外部のネットワークからアプリケーションサーバ内の Web サーバに対する通信は，公開ポートへの通信だけを許可します。HTTP/80 や HTTPS/443 などが該当します。ただし，システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

● アクセスを許可する Web クライアントの IP アドレスの限定（任意）
ファイアウォールの機能を使用してアクセスを許可する Web クライアントの IP アドレスを限定することで，そのシステムの利用者を特定できます。この場合は，ファイアウォール 1 に通信を許可する IP アドレスを指定してください。

● ファイアウォール 2 の設定

Web サーバとアプリケーションサーバ間のアクセスを制御するためのファイアウォールです。次の項目を設定します。

● DMZ の Web サーバ（リバースプロキシサーバ）から内部ネットワークのアプリケーションサーバ内の Web サーバに対してのアクセス許可
ファイアウォール 2 よりも外部のネットワーク（DMZ）からアプリケーションサーバ内の Web サーバに対する通信は，公開ポートへの通信だけを許可します。HTTP/80 や HTTPS/443 などが該当します。ただし，システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

● アクセスを許可する Web クライアントの IP アドレスの限定（任意）
ファイアウォールの機能を使用してアクセスを許可する Web クライアントの IP アドレスを限定することで，そのシステムの利用者を特定できます。この場合は，リバースプロキシサーバの IP アドレスを指定してください。

このほかの通信の設定は，システム構成によって，必要に応じて許可してください。
DNS などの通信の許可が必要な場合があります。

参考

リダイレクタモジュールを DMZ の Web サーバに組み込んだ場合など，Web サーバと J2EE サーバが動作するアプリケーションサーバの間にファイアウォールを配置した場合には，次の設定が必要です。

- **Web サーバからアプリケーションサーバに対してのアクセス許可**
  J2EE サーバの Web サーバ通信用ポート（リダイレクタからのリクエスト受付ポート。デフォルトのポート番号：8007）への通信の許可を設定します。

● ファイアウォール 3 の設定

アプリケーションサーバとデータベース間のアクセスを制御するためのファイアウォールです。このファイアウォールが，システムで最も重要な情報を守るための最終防衛ラインになります。

次の項目を設定します。

● アプリケーションサーバからデータベースサーバに対してのアクセス許可
アプリケーションサーバからデータベースサーバに対する通信は，データベースサーバの通信用ポートだけを許可するように設定します。データベースサーバの通信用ポートは，使用するデータベースの設定に従ってください。データベースサーバからアプリケーションサーバへのコネクション確立が必要な場合があるので注意してください。

このほかの通信の設定は，システム構成によって，必要に応じて許可してください。DNS などの通信の許可が必要な場合があります。

(c) 侵入検知システムの設定

外部ネットワークとアプリケーションサーバ内の Web サーバの公開ポートの通信内容を監視するために，次の項目を設定します。

● 通信内容の監視
通信内容に既知の攻撃パターンや，攻撃と疑われるようなパターンが含まれている場合は運用管理者などに警告を通知するように設定します。侵入検知システムとファイアウォールの連携機能を利用して，疑わしい通信は自動的に遮断するように設定することもできます。

● SSL コネクションの確立部分への攻撃の監視
HTTPS によるアクセスは，通信内容が暗号化されているため，基本的に通信内容を監視できません。この場合は，HTTPS に関する既知の攻撃パターンとして，SSL コネクションの確立部分への攻撃などを監視対象としてください。

● 公開ポート以外のポートへの通信の監視
アプリケーションサーバの公開ポート以外のポートへの通信が外部ネットワークから送信されている場合は，設定ミスなどによってファイアウォールが突破されているおそれがあります。この場合，警告を通知するように設定することをお勧めします。

（5） サーバの層ごとにファイアウォールで区切る構成（アプリケーション分散型）

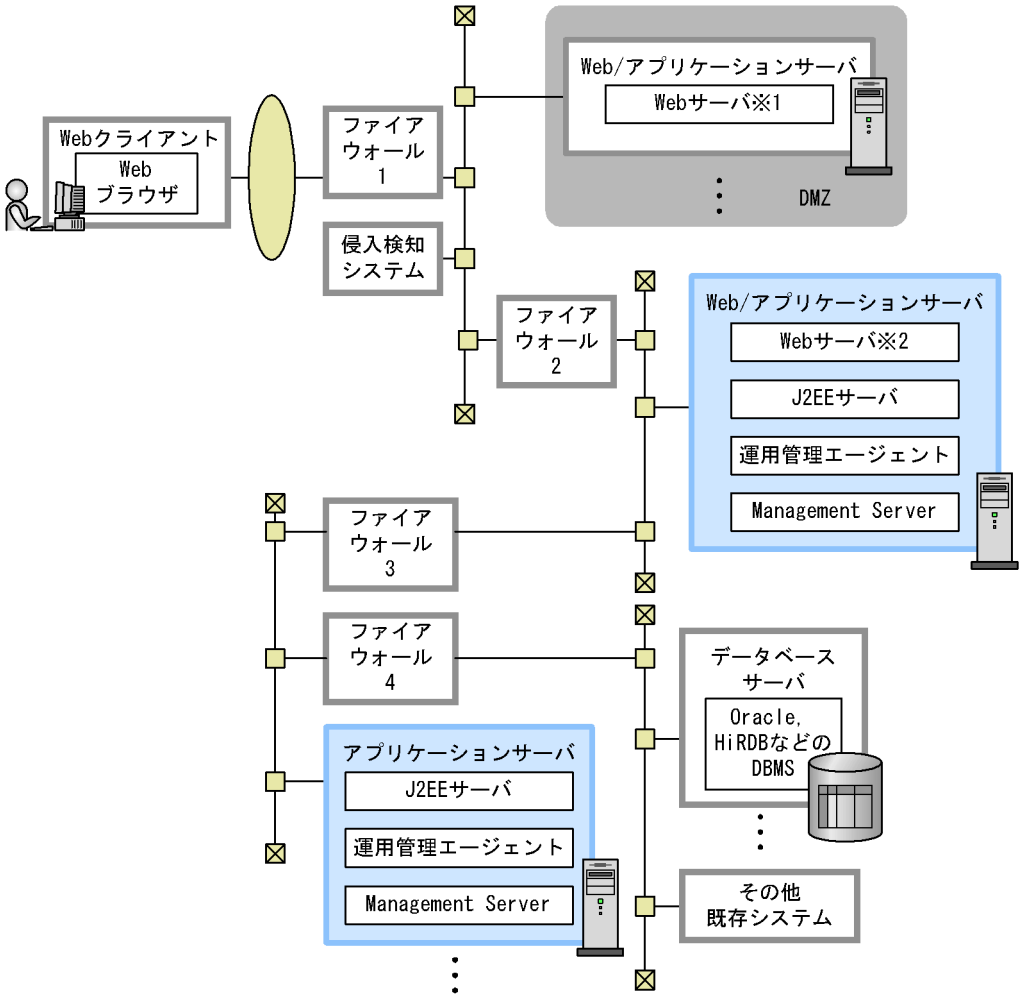
ここでは，「10.10.2(4) サーバの層ごとにファイアウォールで区切る構成（アプリケーション集中型）」と同様に，Web サーバ，アプリケーションサーバおよびデータベースサーバを多層化した場合に，アプリケーションを複数の層のアプリケーションサーバで動作させるときの構成について説明します。この構成を，**アプリケーション分散型の構成**といいます。

アプリケーション分散型の構成の場合のファイアウォールと侵入検知システムの配置例を次の図に示します。この例では，Web サーバに該当するマシンがアプリケーションサーバを兼ねているため，Web アプリケーションが Web サーバと同じ層で動作します。また，Enterprise Bean は，Web サーバとは別のマシン上に構築されたアプリケーションサーバ上で動作します。

なお，運用管理は，各ホストに配置した Management Server によって実行します。このため，管理ホストは層ごとに配置しています。

この構成では，DMZ のフロントに 1 台，およびサーバの層ごとに 1 台ずつ，合計 4 台のファイアウォールを配置しています。また，DMZ には，リバースプロキシモジュールを組み込んだ Web サーバ（リバースプロキシサーバ）を配置しています。

図 10-8　アプリケーション分散型のファイアウォールと侵入検知システムの配置



注※1　リバースプロキシモジュールを組み込んだWebサーバです。DMZに配置します。
注※2　リダイレクタモジュールを組み込んだWebサーバです。
　　アプリケーションサーバと同じマシンに配置します。

(a) Web／アプリケーションサーバでの設定

Web サーバを兼ねたアプリケーションサーバマシン（Web／アプリケーションサーバ）では，次の項目を設定します。なお，このアプリケーションサーバマシンでは，Web アプリケーションが動作します。

● J2EE サーバの管理用通信ポートのアドレス指定

J2EE サーバの管理用通信ポートにアクセスできるアドレスを指定します。

● Management Server および運用管理エージェントのアドレス指定

Management Server および運用管理エージェントにアクセスできるアドレスを指定

します。

(b) アプリケーションサーバの設定

Enterprise Bean が動作するアプリケーションサーバでは，次の項目を設定します。

- **J2EE サーバの管理用通信ポートのアドレス指定**
  J2EE サーバの管理用通信ポートにアクセスできるアドレスを指定します。
- **Management Server および運用管理エージェントのアドレス指定**
  Management Server および運用管理エージェントにアクセスできるアドレスを指定します。
- **EJB コンテナが利用するポート番号の固定**
  EJB コンテナが利用するポート番号を明示的に指定（固定）する必要があります。指定個所については，「3.18　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。

(c) ファイアウォールの設定

この構成では，次の 4 台のファイアウォールを使用しています。

- ファイアウォール 1
  外部のネットワークと DMZ の Web サーバ（リバースプロキシサーバ）間のアクセスを制御します。
- ファイアウォール 2
  DMZ の Web サーバ（リバースプロキシサーバ）と内部のネットワーク上の Web ／アプリケーションサーバ間のアクセスを制御します。
- ファイアウォール 3
  Web ／アプリケーションサーバとアプリケーションサーバ間のアクセスを制御します。
- ファイアウォール 4
  アプリケーションサーバとデータベースサーバ間のアクセスを制御します。

それぞれのファイアウォールに設定する内容について説明します。

- **ファイアウォール 1 の設定**

外部のネットワークと DMZ の Web サーバ（リバースプロキシサーバ）間のアクセスを制御するためのファイアウォールです。次の項目を設定します。

- **外部ネットワークから Web サーバ（リバースプロキシサーバ）に対してのアクセス許可**
  ファイアウォール 1 よりも外部のネットワークからアプリケーションサーバ内の Web サーバに対する通信は，公開ポートへの通信だけを許可します。HTTP/80 や HTTPS/443 などが該当します。ただし，システム構成によっては，DNS など，そのほかの通

信を必要に応じて許可してください。

● **アクセスを許可する** Web **クライアントの** IP **アドレスの限定（任意）**
ファイアウォールの機能を使用してアクセスを許可する Web クライアントの IP アドレスを限定することで，そのシステムの利用者を特定できます。この場合は，ファイアウォール 1 に通信を許可する IP アドレスを指定してください。

● **ファイアウォール** 2 **の設定**

外部のネットワークと内部のネットワーク上の Web ／アプリケーションサーバ間のアクセスを制御するためのファイアウォールです。次の項目を設定します。

● DMZ **の** Web **サーバ（リバースプロキシサーバ）からアプリケーションサーバ内の** Web **サーバに対してのアクセス許可**
ファイアウォール 1 よりも外部のネットワークから，アプリケーションサーバ内の Web サーバに対する通信は，公開ポートへの通信だけを許可します。HTTP/80 や HTTPS/443 などが該当します。ただし，システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

● **アクセスを許可する** Web **クライアントの** IP **アドレスの限定（任意）**
ファイアウォールの機能を使用してアクセスを許可する Web クライアントの IP アドレスを限定することで，そのシステムの利用者を特定できます。この場合は，リバースプロキシサーバの IP アドレスを指定してください。

● **ファイアウォール** 3 **の設定**

Web ／アプリケーションサーバとアプリケーションサーバ間のアクセスを制御するためのファイアウォールです。次の項目を設定します。

● Web **／アプリケーションサーバからアプリケーションサーバに対してのアクセス許可**
Web ／アプリケーションサーバからアプリケーションサーバ上の J2EE サーバを利用するために，次のポート番号に通信できるように設定してください。
- CORBA ネーミングサービス
  ポート番号は，通常固定されています（デフォルトのポート番号：900）。
- EJB コンテナ
  ポート番号が固定されていないため，EJB コンテナが利用するポート番号を明示的に指定（固定）する必要があります。
  指定個所については，「3.18　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。

● **トランザクション関連の双方向アクセスの許可（グローバルトランザクションでトランザクションコンテキストプロパゲーションを使用する場合）**
Web ／アプリケーションサーバとアプリケーションサーバの間で，グローバルトランザクションでトランザクションコンテキストプロパゲーションを使用する場合は，両方のアプリケーションサーバの次に示すポートが双方向で通信できるように設定します。

- J2EE サーバのトランザクションリカバリ用通信ポート（デフォルトのポート番号：20302）
- スマートエージェントの通信ポート（デフォルトのポート番号：14000）

● そのほかの設定（任意）

システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

● ファイアウォール 4 の設定

アプリケーションサーバとデータベース間のアクセスを制御するためのファイアウォールです。このファイアウォールが，システムで最も重要な情報を守るための最終防衛ラインになります。

次の項目を設定します。

● アプリケーションサーバからデータベースサーバに対してのアクセス許可

アプリケーションサーバからデータベースサーバに対する通信は，データベースサーバの通信用ポートだけを許可するように設定します。データベースサーバの通信用ポートは，使用するデータベースの設定に従ってください。データベースサーバからアプリケーションサーバへのコネクション確立が必要な場合があるので注意してください。また，システム構成によっては，DNS など，そのほかの通信を必要に応じて許可してください。

(d) 侵入検知システムの設定

外部ネットワークとアプリケーションサーバ内にある Web サーバの公開ポートの通信内容を監視するために，次の項目を設定します。

● 通信内容の監視

通信内容に既知の攻撃パターンや，攻撃と疑われるようなパターンが含まれている場合は運用管理者などに警告を通知するように設定します。侵入検知システムとファイアウォールの連携機能を利用して，自動的に疑わしい通信は遮断するように設定することもできます。

● SSL コネクションの確立部分への攻撃の監視

HTTPS によるアクセスは，通信内容が暗号化されているため，基本的に通信内容を監視できません。この場合は，HTTPS に関する既知の攻撃パターンとして，SSL コネクションの確立部分への攻撃などを監視対象としてください。

● 公開ポート以外のポートへの通信の監視

Web ／アプリケーションサーバの公開ポート以外のポートへの通信が外部ネットワークから送信されている場合は，設定ミスなどによってファイアウォールが突破されているおそれがあります。この場合，警告を通知するように設定することをお勧めします。

10.10.3　SSL アクセラレータを使用して暗号通信を処理する

ここでは，SSL アクセラレータを使用して暗号通信を処理する方法について説明します。

(1) SSL アクセラレータを使用する目的

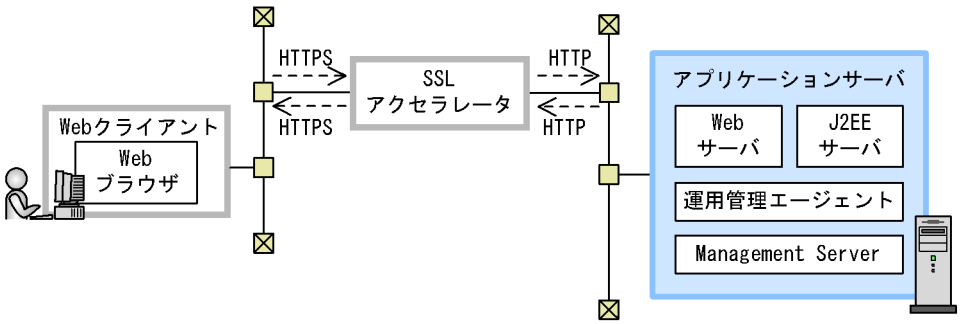
セキュリティ上の脅威のうち，アプリケーションの通信内容の第三者への漏洩または改ざんを防ぐ方法の一つに，通信内容の暗号化があります。暗号化手段としては，通信に HTTPS を使用する方法があります。ただし，HTTPS の基盤となる TLS ／ SSL による通信は大変負荷が高い処理です。

SSL アクセラレータは，このような場合に，Web サーバやアプリケーションサーバに負荷を掛けないで，HTTPS によって暗号化された通信内容の処理を実現するための専用ハードウェアです。SSL アクセラレータを適切に配置することで，Web サーバやアプリケーションサーバに負荷を掛けることなく，暗号化された内容の通信を高速化できます。

(2) SSL アクセラレータの配置

SSL アクセラレータを適用した構成の例を次の図に示します。

図 10-9　SSL アクセラレータを適用した構成



Web クライアントから HTTPS によって送信された通信内容は，SSL アクセラレータによって復号化され，HTTP によって Web サーバまたはアプリケーションサーバに送信されます。また，Web サーバまたはアプリケーションサーバから HTTP によって送信された通信内容は，SSL アクセラレータによって暗号化されて，Web クライアントに送信されます。

SSL アクセラレータを配置する場合は，次の点に留意してください。

- SSL アクセラレータは，ファイアウォールとの併用もできます。併用する場合は，SSL アクセラレータを Web サーバまたはアプリケーションサーバと一体として扱ってください。
- Web サーバ連携をする場合は，SSL アクセラレータ使用時に，リダイレクタの追加設定が必要な場合があります。リダイレクタを設定すると，例えば，Web クライアント

からのリクエストが HTTP/1.0 の場合にリクエストのフォワード先指定などができます。Management Server を利用してシステムを構築する場合は，運用管理ポータルの「論理サーバの環境設定」で設定できます。論理 Web サーバの［リダイレクタの設定］画面の「ゲートウェイ指定機能の設定」で，「SSL アクセラレータの使用」に，「する」を選択してください。
Management Server を利用しないでシステムを構築する場合は，次のファイルに設定します。

Hitachi Web Server の場合

mod_jk.conf ファイルの JkGatewayHttpsScheme キー

Microsoft IIS の場合

isapi_redirect.conf ファイルの gateway_https_scheme キー

Management Server を利用してシステムを構築する場合は，マニュアル「Cosminexus 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「10.10.2 リダイレクタの設定」を参照してください。
Management Server を利用しないでシステムを構築する場合は，マニュアル「Cosminexus 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「9. Web サーバ連携で使用するファイル」を参照してください。

- インプロセス HTTP サーバを使用する場合は，SSL アクセラレータ使用時に，Web コンテナの追加設定が必要な場合があります。この設定によって，例えば，Web クライアントからのリクエストが HTTP/1.0 の場合にリクエストのフォワード先指定などができます。Management Server を利用してシステムを構築する場合は，運用管理ポータルの「論理サーバの環境設定」で設定できます。論理 J2EE サーバの［その他の設定］画面で，「ゲートウェイ指定機能の設定」の「SSL アクセラレータの使用」に，「する」を選択してください。
  Management Server を利用しないでシステムを構築する場合は，usrconf.properties ファイルの webserver.connector.inprocess_http.gateway.https_scheme キーに設定します。
  Management Server を利用してシステムを構築する場合は，マニュアル「Cosminexus 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「10.9.16 レスポンスヘッダ・ゲートウェイ指定機能の設定（インプロセス HTTP サーバ）」を参照してください。
  Management Server を利用しないでシステムを構築する場合は，マニュアル「Cosminexus 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「2.4 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。

10.10.4 アプリケーションでユーザを認証する

ここでは，Web クライアント構成の場合に，セキュリティ確保のためにアプリケーションで利用できる認証機能について説明します。

(1) アプリケーションでユーザを認証する目的

アプリケーションを実行するユーザを認証することで，セキュリティ上の脅威のうち，システム利用者によって，許可した権限の範囲を超えて操作されたり情報を入手されたりすることを防げます。

アプリケーションサーバでは，次の 3 種類のプロトコルを使用してユーザ認証によるセキュリティ確保ができます。

- HTTPS（クライアント認証）
- HTTP（Basic 認証）
- HTTP（Form 認証）

これらのプロトコルを目的に応じて使い分けて，セキュリティを確保してください。

(2) アプリケーションによるユーザ認証方法の比較

通信プロトコルごとの，認証実施個所および認証に使用されるエンジンについて，次の表に示します。

表 10-7 通信プロトコルごとの認証実施個所および認証に使用されるエンジン

使用するプロトコル	認証実施個所	認証に使用されるエンジン
HTTPS(クライアント認証)	Hitachi Web Server または Microsoft IIS	SSL
	SSL アクセラレータ	SSL
HTTP(Basic 認証)	Hitachi Web Server	HWS パスワードファイル
		LDAP リポジトリ
	J2EE サーバ（Web コンテナ）	パスワードファイル
HTTP(Form 認証)	J2EE サーバ（Web コンテナ）	パスワードファイル
	J2EE サーバ（統合ユーザ管理）	統合ユーザ管理パスワードファイル
		データベース
		LDAP リポジトリ

それぞれのプロトコルおよび認証に使用されるエンジンには特徴があります。それらの特徴を考慮して，システムの目的に合った認証方法を選択してください。

(a) プロトコルの特徴

アプリケーションサーバのシステムの認証処理に使用できるプロトコルの特徴について，次の表に示します。

表 10-8 プロトコルの特徴

使用するプロトコル	認証インタフェースの自由度	クライアントでの管理の容易さ	ネットワークの安全性
HTTPS（クライアント認証）	Web ブラウザが提供する機能に制約されます。	クライアント証明書が必要です。	暗号化されているため，盗聴されても認証情報は安全です。
HTTP（Basic 認証）	Web ブラウザが提供する機能に制約されます。	一般的なユーザ名称／パスワード形式を使用した認証ができます。	パスワードが平文またはそれと同等の形式で流出します。 このため，通常は，HTTPS（サーバ認証だけ）の暗号機能と併用して使用します。
HTTP（Form 認証）	アプリケーションごとにデザインできます。	一般的なユーザ名称／パスワード形式を使用した認証ができます。	パスワードが平文またはそれと同等の形式で流出します。 このため，通常は，HTTPS（サーバ認証だけ）の暗号機能と併用して使用します。

(b) 認証に使用されるエンジンの特徴

認証に使用されるエンジンの特徴について，次の表に示します。

表 10-9 認証に使用されるエンジンの特徴

エンジンの種類	はん用性	保守性	システム構成への影響	性能影響への影響
パスワードファイル	使用する機能ごとに形式が異なります。	サーバまたはホストごとにユーザ情報が散在します。	特に認証用のプロセスは必要ありません。	認証時にほかのプロセスやホストとの通信が発生しないため，高速です。
データベース	形式によっては，既存のユーザ情報データベースを利用できます。	ユーザ情報の一括管理ができます。	ユーザ情報を格納するデータベースサーバが必要です。	認証時にデータベースアクセスに必要な時間が掛かります。
LDAP リポジトリ	形式によっては，既存のユーザ情報リポジトリを利用できます。	ユーザ情報の一括管理，および分散したユーザ情報の集中管理ができます。	ユーザ情報を格納する LDAP 対応のディレクトリサーバが必要です。	認証時に LDAP ディレクトリサーバへのアクセスに必要な時間が掛かります。

付録

付録 A　ベーシックモードの利用（互換機能）

互換用のサーバの動作モードとして，ベーシックモードがあります。ベーシックモードは，J2EE サーバモードの一つです。単一のデータベースだけのリソースをトランザクションで使用するシステムに適用できます。ここでは，ベーシックモードの利用について説明します。

なお，ベーシックモードは，1.4 モードに比べて，使用できる機能に制限があります。旧バージョンで J2EE サーバモードにベーシックモードを使用していた場合には，1.4 モードに移行することをお勧めします。

付録 A.1　パフォーマンスチューニング

ここでは，ベーシックモードで使用できるパフォーマンスチューニングについて説明します。

ベーシックモードで使用できるパフォーマンスチューニングを次の表に示します。

表 A-1　ベーシックモードで使用できるパフォーマンスチューニング

チューニング項目			使用可否	参照先
同時実行数の最適化	リクエスト処理スレッド数（インプロセス HTTP サーバ使用時）		○	8.3.3
	Web アプリケーションの同時実行数	URL グループ単位	○	8.3.4
		Web アプリケーション単位	○	
		Web コンテナ単位	○	
	Enterprise Bean の同時実行数	Stateless Session Bean の同時実行数	○	8.3.5
		Stateful Session Bean の同時実行数	○	
		Entity Bean の同時実行数	○	
		Message-driven Bean の同時実行数	○	
	CTM で制御する同時実行数		○	8.3.6
Enterprise Bean の呼び出し方法の最適化	ローカルインタフェースの使用		○	8.4.1
	リモートインタフェースのローカル呼び出し機能の使用		○	8.4.2
	リモートインタフェースの参照渡し機能の使用		○	8.4.3
データベースアクセス方法の最適化	コネクションプーリング		○	付録 A.1(1)

チューニング項目			使用可否	参照先
	ステートメントプーリング		×	－
タイムアウトの設定	Web フロントシステムでのタイムアウト	Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト	○	8.6.2
		リダイレクタ側で設定する Web コンテナへのデータ送信のタイムアウト	○	
		リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト	○	
		Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト	○	
		Web コンテナ側で設定するリダイレクタへのデータ受信のタイムアウト	○	
	バックシステムでのタイムアウト	EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト	○	8.6.3
		EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト	○	
	EJB コンテナ側で設定するデータベースのトランザクションタイムアウト（DB Connector を使用した場合）		○※	8.6.4
	データベースのタイムアウト		○	8.6.5
	J2EE アプリケーションのメソッドタイムアウト		○	8.6.6
Web アプリケーションの動作の最適化	静的コンテンツと Web アプリケーションの配置の切り分け		○	8.7.1
	静的コンテンツのキャッシュ		○	8.7.2
	リダイレクタによるリクエストの振り分け		○	8.7.3
CTM の動作の最適化	CTM ドメインマネジャおよび CTM デーモンの稼働状態を監視する間隔の設定		○	8.8.1
	負荷状況監視間隔の設定		○	8.8.2
	CTM デーモンのタイムアウト閉塞の設定		○	8.8.3
	CTM で振り分けるリクエストの優先順位の設定		○	8.8.4
そのほかの項目のチューニング	Persistent Connection についてのチューニング		○	8.9

（凡例）○：使用できる。　×：使用できない。　－：該当しない。

注※

ベーシックモードの場合，CMT を使用するときのトランザクションのタイムアウトは，Enterprise Bean，インタフェースまたはメソッドの属性では設定できません。ベーシックモードの場合，CMT を使用するときのトランザクションのタイムアウトは，usrconf.properties で定義してください。

これらのチューニング項目のうち，ベーシックモードで使用できる機能や，チューニングパラメタの異なるものについて説明します。

(1) データベースへのアクセス方法を最適化するためのチューニングパラメタ

ベーシックモードの場合に，コネクションプーリングで使用できる機能を次の表に示します。なお，コネクションプーリングで使用できる各機能の設計時の指針については，「8.5.1　コネクションプーリングを使用する」を参照してください。

表 A-2　コネクションプーリングで使用できる機能（ベーシックモードの場合）

機能	使用可否
プールするコネクションの最大値と最小値を指定する	○
プール内のコネクションを検査して無効なコネクションを破棄する	○
コネクション取得失敗時にリトライする	△
スイーパによって使わないコネクションをプールから削除する※	○
コネクションウォーミングアップによってあらかじめコネクションをプールしておく	×
コネクション枯渇時のコネクション取得要求を取得待ちキューに入れる	×
コネクションプール内の不要なコネクションを段階的に減少させる	×
コネクションプールをクラスタ化する	×
コネクションプールの未使用コネクションがしきい値以下になったとき，コネクションを追加する	○
コネクション取得待ち時間を指定する	○

（凡例）○：使用できる。　△：指定しても無視される。　×：使用できない。

注※

ただし，スイーパを実行する処理は負荷が高いため，性能に影響が出ます。このため，スイーパを使用する場合は，自動削除の間隔を 1 時間以上の値にすることをお勧めします。

次に，ベーシックモードの場合にコネクションプーリングをチューニングするパラメタについて説明します。これらのチューニングパラメタは，データソース単位に設定します。

チューニングパラメタについて，次の表に示します。これらは，サーバ管理コマンドの CUI で設定します。CUI を使用する場合は，属性ファイルを編集してください。

表 A-3　ベーシックモードの場合のコネクションプーリングのチューニングパラメタ

設定項目	設定方法	設定個所
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	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <MinimumSize> タグ
コネクションプールにプールするコネクションの最大値	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <MaximumSize> タグ
コネクションの生成時刻からコネクションを自動破棄するかを判定するまでの時間	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <ConnectionTimeout> タグ
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	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <SweeperInterval> タグ
コネクションプールにコネクションを追加するためのしきい値	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <Threshold> タグ
コネクションプールにコネクションを追加する場合の追加数	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <GrowthIncrement> タグ
コネクション取得待ち時間	CUI（cjsetresprop）	データソース属性ファイルの <PoolConfiguration> タグ下の <WaitTimeout> タグ

注

cjsetresprop コマンドについては，マニュアル「Cosminexus アプリケーションサーバ リファレンス コマンド編」の「cjsetresprop（リソースの属性設定）」を参照してください。データソース属性ファイルについては，マニュアル「Cosminexus 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「4.4　データソース属性ファイル」を参照してください。

付録 B　サーブレットエンジンモードの利用（互換機能）

互換用のアプリケーションサーバの動作モードとして，サーブレットエンジンモードがあります。サーブレットエンジンモードは，サーブレットと JSP で構成される Web アプリケーションだけをアプリケーションサーバで動作させたい場合に利用できます。ここでは，サーブレットエンジンモードの利用について説明します。

なお，旧バージョンでサーブレットエンジンモードを使用していた場合には，アプリケーションサーバの運用管理機能などが使用できる J2EE サーバモードに移行することをお勧めします。

付録 B.1　システム構成の設計

ここでは，サーブレットエンジンモードで使用できるシステム構成について説明します。

（1）システム設計の準備

システム設計作業を始める前に，次のことを明確にする必要があります。

- アプリケーションおよびプロセスの構成
- 運用方法

（a）アプリケーションおよびプロセスの構成

サーブレットエンジンモードでは，サーブレットと JSP だけで構成されるアプリケーション（Web アプリケーション）が使用できます。

また，サーブレットエンジンモードでアプリケーションサーバを動作させるシステムでは，次に示すプロセスが必要となります。

- Web サーバ※
- Web コンテナサーバ
- PRF デーモン

注※
　アプリケーションサーバに含まれる Web サーバは，Hitachi Web Server です。

これらのプロセスは，アプリケーションサーバによって提供され，アプリケーションサーバをインストールしたマシンで起動できます。

サーブレットエンジンモードの場合，Web クライアントからのリクエストを，リダレクタと連携した Web サーバ（Web サーバ連携）を利用して処理できます。利用できる Web サーバは，Hitachi Web Server または Microsoft IIS です。

(b) 運用方法

サーブレットエンジンモードでは，Management Server を利用した運用，およびクラスタソフトウェアと連携したシステムの運用はできません。サーブレットエンジンモードで運用する場合，システムの構築には，アプリケーションサーバが提供するコマンドと定義ファイルを使用します。

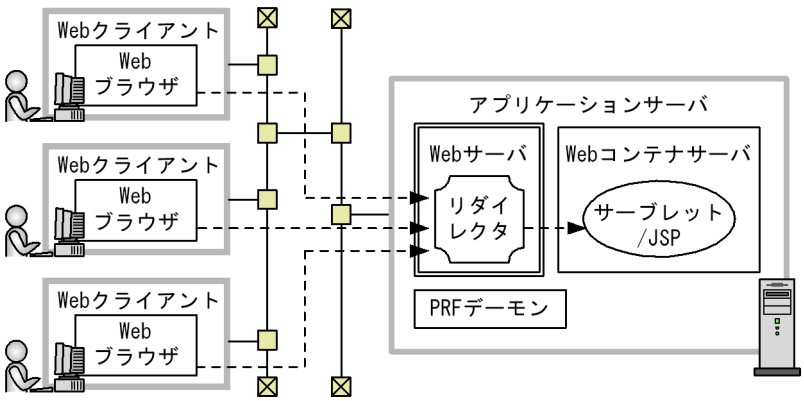
(2) サーブレットエンジンモードのシステムの構成

サーブレットエンジンモードを使用して，アプリケーションサーバのシステムを設計する場合，Web コンテナサーバを使用した構成になります。Web コンテナサーバは，サーブレットエンジンモードで動作しているアプリケーションサーバです。

システム構成の特徴

Web コンテナサーバの構成の例を次の図に示します。

図 B-1　Web コンテナサーバの構成の例



（凡例）

----▶　：リクエストの流れ

これ以外の凡例については，「3.2　システム構成の説明について」を参照してください。

特徴

- サーブレットと JSP だけが動作します。
- サーブレットと JSP を対象にした，負荷分散機またはリダイレクタを使用した負荷分散構成，および Web サーバとアプリケーションサーバを分離した構成が実現できます。
- Management Server を利用した運用はできません。

リクエストの流れ

サーブレットと JSP に対するクライアントからのリクエストは，Web ブラウザから Web サーバを経由して送られます。

それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

アプリケーションサーバマシン

アプリケーションサーバマシンには，Application Server Standard または Application Server Enterprise をインストールする必要があります。
起動するプロセスは次のとおりです。

- Web サーバ
- Web コンテナサーバ
- PRF デーモン

Application Server Standard または Application Server Enterprise には，Web サーバである Hitachi Web Server が含まれています。Windows の場合，Web サーバに Microsoft IIS を使用することもできます。この場合は，ソフトウェアとして Microsoft IIS が必要です。

Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

付録 B.2　パフォーマンスチューニング

ここでは，サーブレットエンジンモードで使用できるアプリケーションのチューニング項目について説明します。サーブレットエンジンモードでは，アプリケーションがサーブレットと JSP だけで構成される Web アプリケーションが使用できます。サーブレットエンジンモードで使用できるアプリケーションのチューニング項目を次の表に示します。

表 B-1　サーブレットエンジンモードで使用できるアプリケーションのチューニング項目

チューニング項目	利用できる機能	参照先
同時実行数の最適化	Web アプリケーションでの同時実行スレッド数制御 （Web コンテナ単位）	8.3.4
データベースアクセス方法の最適化	Web コンテナコネクションプーリング	付録 B.2(1)
タイムアウトの設定	Web フロントシステムでのタイムアウトの設定	8.6.2
Web アプリケーションの動作の最適化	静的コンテンツと Web アプリケーションの配置の切り分け	8.7.1
	静的コンテンツのキャッシュ	8.7.2
	リダイレクタによるリクエストの振り分け（Web サーバ連携の場合）	8.7.3

これらのチューニング項目のうち，サーブレットエンジンモードで動作する場合に異なる機能について説明します。

（1） Web コンテナコネクションプーリングを使用する

サーブレットエンジンモード上で動作する Web アプリケーションからデータベースへ接続する場合，Web コンテナコネクションプールを使用します。このとき，Web コンテナコネクションプーリング機能を利用してデータベースへの接続をプールできます。これによって，コネクション生成時のパフォーマンスを向上できます。

Web コンテナコネクションプーリングを使用する場合，最初のコネクション取得要求があったときに，プール数の最小値までコネクションが生成され，プールされます。クライアントからコネクション取得要求があると，プールから未使用のコネクションが選択され，クライアントに渡されます。コネクションの解放要求があると，使用中になっているコネクションの状態が未使用に設定され，コネクションがプールに返されます。なお，コネクションの状態が，再利用できない状態に設定されている場合は，データベースとの接続が切断され，コネクションが削除されます。この場合，コネクションはプールに返されません。

Web コンテナコネクションプーリングを使用する場合は，チューニングパラメタを設定する必要があります。Web コンテナコネクションプーリングのチューニングパラメタは，プール管理情報設定ファイルに設定します。設定した内容は，cjwebeditpool コマンドで登録します。プール管理情報設定ファイルについては，マニュアル「Cosminexus 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「4.5　プール管理情報設定ファイル」を，cjwebeditpool コマンドについては，マニュアル「Cosminexus アプリケーションサーバ リファレンス コマンド編」の「cjwebeditpool（プール管理情報の登録と変更）」を参照してください。

Web コンテナコネクションプーリングのチューニングパラメタを次の表に示します。

表 B-2　Web コンテナコネクションプーリングのチューニングパラメタ

設定項目	設定個所
プールするコネクションの最小値	<MinimumSize> タグ
プールするコネクションの最大値	<MaximumSize> タグ
コネクションを取得できなかった場合にリトライするまでのアプリケーションの待ち時間	<RetryInterval> タグ
コネクションを取得できなかった場合のリトライ回数	<RetryCount> タグ

付録 C　推奨手順以外の方法でパフォーマンスチューニングをする場合のチューニングパラメタ

ここでは，「8.　パフォーマンスチューニング（J2EE アプリケーション実行基盤）」および「9.　パフォーマンスチューニング（バッチアプリケーション実行基盤）」で説明した項目について，推奨手順以外の方法でチューニングを実施するためのチューニングパラメタについて説明します。推奨以外の方法とは，運用管理ポータルを使用して設定する方法，およびファイル編集によって設定する方法です。

チューニングの考え方については，「8.　パフォーマンスチューニング（J2EE アプリケーション実行基盤）」および「9.　パフォーマンスチューニング（バッチアプリケーション実行基盤）」を参照してください。

付録 C.1　同時実行数を最適化するためのチューニングパラメタ（推奨手順以外の方法）

ここでは，同時実行数の最適化で使用するチューニングパラメタの設定方法と設定個所についてまとめて示します。

(1) インプロセス HTTP サーバ使用時のリクエスト処理スレッド数

インプロセス HTTP サーバを使用している場合の，リクエスト処理スレッド数のチューニングパラメタの設定方法および設定個所について，次の表に示します。

なお，Web サーバ連携で Hitachi Web Server を使用している場合のチューニングパラメタについては，マニュアル「Hitachi Web Server」を参照してください。

表 C-1　インプロセス HTTP サーバ使用時のリクエスト処理スレッド数のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
J2EE サーバ起動時に生成するリクエスト処理スレッド数	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「初期スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.init_threads キー
Web クライアントとの接続数の上限（リクエスト処理スレッド数の上限）	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「初期スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.max_connections キー

設定項目	設定方法	設定個所
Web クライアントとの接続数の上限を超えた場合に使用される TCP/IP の Listen キュー（バックログ）の最大値	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Web クライアントとの接続設定」の「TCP リスンキューの長さ」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.backlog キー
予備スレッド数の最大数	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「予備スレッド」の「最大数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.max_spare_threads キー
予備スレッド数の最小数	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「予備スレッド」の「最小数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.min_spare_threads キー

(2) Web アプリケーションの同時実行数

URL グループ単位，Web アプリケーション単位，または Web コンテナ単位に設定します。

(a) URL グループ単位の同時実行数

URL グループ単位の同時実行数のチューニングパラメタの設定方法および設定個所について，次の表に示します。

表 C-2　URL グループ単位の同時実行数のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
Web コンテナ単位での最大同時実行スレッド数（Web サーバ連携時）	運用管理ポータル	［Web コンテナの設定］画面の「Web サーバとの接続」の「最大スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.ajp13.max_threads キー
Web コンテナ単位での最大同時実行スレッド数（インプロセス HTTP サーバ使用時）	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「同時実行スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.max_execute_threads キー
Web アプリケーション単位で同時実行数を制御するかどうか	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「同時実行スレッド数制御機能の使用」
	ファイル編集	usrconf.properties の webserver.container.thread_control.enabled キー

設定項目	設定方法	設定個所
デフォルトの実行待ちキューサイズ	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「実行待ちキューサイズ」
	ファイル編集	usrconf.properties の webserver.container.thread_control.queue_size キー

次の設定項目については，推奨手順の場合と違いがありません。

- Web アプリケーション単位での最大同時実行スレッド数
- Web アプリケーションの占有スレッド数
- Web アプリケーション単位の実行待ちキューサイズ
- URL グループ単位の同時実行スレッド数制御の定義名
- URL グループ単位での最大同時実行スレッド数
- URL グループ単位の占有スレッド数
- URL グループ単位の実行待ちキューサイズ
- URL グループ単位の制御対象となる URL パターン

（b）Web アプリケーション単位の同時実行数

Web アプリケーション単位の同時実行数をチューニングするパラメタの設定方法および設定個所について，次の表に示します。

表 C-3　Web アプリケーション単位の同時実行数のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
Web コンテナ単位での最大同時実行スレッド数（Web サーバ連携時）	運用管理ポータル	［Web コンテナの設定］画面の「Web サーバとの接続」の「最大スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.ajp13.max_threads キー
Web コンテナ単位での最大同時実行スレッド数（インプロセス HTTP サーバ使用時）	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「同時実行スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.max_execute_threads キー
Web アプリケーション単位で同時実行数を制御するかどうか	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「同時実行スレッド数制御機能の使用」
	ファイル編集	usrconf.properties の webserver.container.thread_control.enabled キー
デフォルトの実行待ちキューサイズ	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「実行待ちキューサイズ」
	ファイル編集	usrconf.properties の webserver.container.thread_control.queue_size キー

次の設定項目については，推奨手順と違いがありません。

- Web アプリケーション単位での最大同時実行スレッド数
- Web アプリケーションの占有スレッド数
- Web アプリケーション単位の実行待ちキューサイズ

(c) Web コンテナ単位の同時実行数

Web コンテナ単位の同時実行数をチューニングするパラメタの設定方法および設定個所について，次の表に示します。

表 C-4　Web コンテナ単位の同時実行数のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
Web コンテナ単位での最大同時実行スレッド数（Web サーバ連携時）	運用管理ポータル	［Web コンテナの設定］画面の「Web サーバとの接続」の「最大スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.ajp13.max_threads キー
Web コンテナ単位での最大同時実行スレッド数（インプロセス HTTP サーバ使用時）	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「スレッド制御設定」の「同時実行スレッド数」
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.max_execute_threads キー

参考

このほか，Web コンテナでは，リダイレクタからの TCP 接続要求の最大待ち行列数も指定できます（usrconf.properties の webserver.connector.ajp13.backlog キー）。ただし，これはソケットの Listen キューの大きさを指定するキーであり，リクエストの実行待ちキューと直接の関係はありません。

(3) Enterprise Bean の同時実行数

Enterprise Bean の同時実行数は，Enterprise Bean 単位に設定します。

Enterprise Bean の同時実行数のチューニングパラメタについては，推奨手順の場合と違いがありません。

(4) CTM で制御する同時実行数

CTM で制御する同時実行数のチューニングパラメタの設定方法および設定個所について，次の表に示します。CTM デーモン，アプリケーション，および Stateless Session Bean に設定する項目があります。

CTM で制御する同時実行数のチューニングパラメタの設定方法および設定個所について，次の表に示します。

表 C-5　CTM で制御する同時実行数のチューニングパラメタ（推奨手順以外の方法）

設定対象	設定項目	設定方法	設定個所
CTM デーモン	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	運用管理ポータル	［スケジューリングの設定］画面の「CTM キューの設定」の「スレッド最大値」
		ファイル編集	ctmstart コマンドの引数「-CTMDispatchParallelCount」

なお，アプリケーションまたは Stateless Session Bean に対する設定項目については，推奨手順の場合と違いがありません。

付録 C.2　Enterprise Bean の呼び出し方法を最適化するためのチューニングパラメタ（推奨手順以外の方法）

ここでは，Enterprise Bean の呼び出し方法の最適化で使用するチューニングパラメタの設定個所についてまとめて示します。

(1) ローカルインタフェースの使用

アプリケーション作成時に，J2EE で定義されているローカルインタフェースを使用してください。

推奨手順の場合と違いはありません。

(2) リモートインタフェースのローカル呼び出し機能の使用

リモートインタフェースのローカル呼び出し機能をチューニングするパラメタの設定方法および設定個所について，次の表に示します。

表 C-6　リモートインタフェースのローカル呼び出し機能のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
ローカル呼び出し最適化機能の適用範囲	運用管理ポータル	［EJB コンテナの設定］画面の「オプション」の「J2EE アプリケーションの呼び出し方式」
	ファイル編集	usrconf.properties の ejbserver.rmi.localinvocation.scope キー

(3) リモートインタフェースの参照渡し機能の使用

リモートインタフェースの参照渡し機能をチューニングするパラメタの設定方法および設定個所について，次の表に示します。

表 C-7　リモートインタフェースの参照渡し機能のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
リモートインタフェースの参照渡し機能の使用（J2EE サーバ単位）	運用管理ポータル	［EJB コンテナの設定］画面の「オプション」の「リモートインタフェースの参照渡し」
	ファイル編集	usrconf.properties の ejbserver.rmi.passbyreference キー

なお，リモートインタフェースの参照渡し機能の使用（Enterprise Bean 単位）については，推奨手順の場合と違いがありません。

付録 C.3　データベースへのアクセス方法を最適化するためのチューニングパラメタ（推奨手順以外の方法）

データベースへのアクセス方法の最適化で使用するチューニングパラメタについては，推奨手順の場合と違いがありません。

付録 C.4　タイムアウトを設定するチューニングパラメタ（推奨手順以外の方法）

ここでは，タイムアウトの設定で使用するチューニングパラメタの設定個所についてまとめて示します。

(1) Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト

Web サーバ連携の場合は，Web サーバ単位に設定します。インプロセス HTTP サーバの場合は，J2EE サーバ単位に設定します。

表 C-8　Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウトのチューニングパラメタ（推奨手順以外の方法）

使用する Web サーバ	設定方法	設定個所
Web サーバ連携	運用管理ポータル（Hitachi Web Server の場合）	［Web サーバの設定］画面の「項目ごとに設定します。」の「追加ディレクティブ」の Timeout ディレクティブ
		［Web サーバの設定］画面の「設定ファイルの内容を直接設定します。」の「設定ファイルの内容」の Timeout ディレクティブ

使用する Web サーバ	設定方法	設定個所
	ファイル編集※	• Hitachi Web Server の場合 httpsd.conf の Timeout ディレクティブ • Microsoft IIS の場合 isapi_redirect.conf の receive_client_timeout キー
インプロセス HTTP サーバ	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Web クライアントとの接続設定」の「通信タイムアウト」の「リクエスト受信」
		［通信・スレッド制御に関する設定］画面の「Web クライアントとの接続設定」の［通信タイムアウト］の［リクエスト送信］
	ファイル編集	usrconf.properties の webserver.connector.inprocess_http.receive_timeout キー
		usrconf.properties の webserver.connector.inprocess_http.send_timeout キー

注※　Hitachi Web Server の定義ファイルである httpd.conf を編集して設定します。

(2) リダイレクタ側で設定する Web コンテナへのデータ送信のタイムアウト

リダイレクタ側で設定するタイムアウトのチューニングパラメタについて，次の表に示します。なお，これらのチューニングパラメタは，Web サーバ連携の場合だけ指定できます。

表 C-9　リダイレクタ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

Web サーバの種類	設定方法	設定個所
Hitachi Web Server	運用管理ポータル	［リダイレクタの設定］画面の「オプション」の「リクエスト送信コネクション確立タイムアウト時間」
	ファイル編集	mod_jk.conf の JkConnectTimeout パラメタ
Microsoft IIS	ファイル編集	isapi_redirect.conf の connect_timeout キー
Hitachi Web Server	運用管理ポータル	［リダイレクタの設定］画面の「オプション」の「リクエスト送信タイムアウト時間」
	ファイル編集	mod_jk.conf の JkSendTimeout パラメタ
Microsoft IIS	ファイル編集	isapi_redirect.conf の send_timeout キー

(3) リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト

リダイレクタのワーカ定義単位で設定します。リダイレクタ側で設定するタイムアウトのチューニングパラメタの設定方法と設定個所を次の表に示します。

表 C-10　リダイレクタ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

設定方法	設定個所
運用管理ポータル	［ワーカの設定］画面の「リダイレクタで再利用するワーカとのコネクション数の定義」の「通信タイムアウト」
ファイル編集	workers.properties の worker.< ワーカ名 >.receive_timeout キー

Web サーバ連携の場合だけ指定できます。

(4) Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタを次の表に示します。

表 C-11　Web コンテナ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

設定方法	設定個所
運用管理ポータル	［Web コンテナの設定］画面の「Web サーバとの接続」の「タイムアウト時間」
ファイル編集	usrconf.properties の webserver.connector.ajp13.receive_timeout キー

Web サーバ連携の場合だけ指定できます。

(5) Web コンテナ側で設定するリダイレクタへのデータ受信のタイムアウト

J2EE サーバ単位で設定します。Web コンテナ側で設定するタイムアウトのチューニングパラメタを次の表に示します。

表 C-12　Web コンテナ側で設定するタイムアウトのチューニングパラメタ（推奨手順以外の方法）

設定方法	設定個所
運用管理ポータル	［Web コンテナの設定］画面の「Web サーバとの接続」の「レスポンス送信タイムアウト時間」
ファイル編集	usrconf.properties の webserver.connector.ajp13.send_timeout キー

Web サーバ連携の場合だけ指定できます。

（6）EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト

J2EE サーバ単位，EJB クライアントアプリケーション単位または API による呼び出し単位に設定します。

EJB クライアント側で設定するタイムアウトのチューニングパラメタ（RMI-IIOP 通信によるリモート呼び出し）を次の表に示します。

表 C-13　EJB クライアント側で設定するタイムアウトのチューニングパラメタ（RMI-IIOP 通信によるリモート呼び出し）（推奨手順以外の方法）

単位	設定方法	設定個所
J2EE サーバ単位	運用管理ポータル	［EJB コンテナの設定］画面の「サーバとの接続」の「タイムアウト時間」
	ファイル編集	usrconf.properties の ejbserver.rmi.request.timeout キー

EJB クライアントアプリケーション単位の設定，および API 単位の設定については，推奨手順の場合と違いがありません。

EJB クライアント側で設定するタイムアウトのチューニングパラメタ（ネーミングサービス呼び出し）を次の表に示します。

表 C-14　EJB クライアント側で設定するタイムアウトのチューニングパラメタ（ネーミングサービス呼び出し）（推奨手順以外の方法）

単位	設定方法	設定個所
J2EE サーバ単位	運用管理ポータル	［ネーミングの設定］画面の「利用するネーミングサービスの設定」の「タイムアウト時間」
	ファイル編集	usrconf.properties の ejbserver.jndi.request.timeout キー

EJB クライアントアプリケーション単位の設定については，推奨手順の場合と違いがありません。

（7）EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト

J2EE サーバ単位，EJB クライアントアプリケーション単位または API による呼び出し単位に設定します。

なお，このタイムアウトの設定値には，「(6) EJB クライアント側で設定する Enterprise Bean のリモート呼び出し（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出

しのタイムアウト」で指定した設定値と同じ値が引き継がれます。

(8) EJB コンテナ側で設定するデータベースのトランザクションタイムアウト（DB Connector を使用した場合）

J2EE サーバ単位，Enterprise Bean，インタフェース，メソッド単位（CMT の場合），または API による呼び出し単位（BMT の場合）に設定します。

トランザクションタイムアウトのチューニングパラメタを次の表に示します。

表 C-15　トランザクションタイムアウトのチューニングパラメタ（推奨手順以外の方法）

単位	設定方法	設定個所
J2EE サーバ単位	運用管理ポータル	［トランザクションの設定］画面の「トランザクションに関する設定 」の「タイムアウト時間」
	ファイル編集	usrconf.properties の ejbserver.jta.TransactionManager.defaultTimeOut キー

Enterprise Bean，インタフェース，メソッド単位（CMT の場合）および API 単位（BMT の場合）のチューニングパラメタについては，推奨手順の場合と違いがありません。

(9) データベースのタイムアウト

データベースのタイムアウトを設定するチューニングパラメタについては，推奨手順の場合と違いがありません。

付録 C.5　Web アプリケーションの動作を最適化するためのチューニングパラメタ（推奨手順以外の場合）

ここでは，Web アプリケーションの動作を最適化するために使用するチューニングパラメタの設定個所についてまとめて示します。

(1) 静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ

静的コンテンツと Web アプリケーションの配置の切り分けは，Web サーバの動作を定義するファイルのパラメタとして指定します。設定個所，ファイルおよびパラメタは，使用する Web サーバの種類によって異なります。

Web サーバの種類ごとの設定方法および設定個所を次に示します。

表 C-16　静的コンテンツと Web アプリケーションの配置を切り分けるためのチューニングパラメタ（推奨手順以外の場合）

使用する Web サーバ	Web サーバの種類	設定方法	設定個所
Web サーバ連携（リダイレクタモジュールを使用した切り分け）	Hitachi Web Server	運用管理ポータル	［論理 Web サーバの定義］画面の「マッピング定義」
		ファイル編集	mod_jk.conf のマッピング定義（JkMount パラメタ）
	Microsoft IIS	ファイル編集	uriworkermap.properties
インプロセス HTTP サーバ（リバースプロキシモジュールを使用した切り分け）	Hitachi Web Server	運用管理ポータル	［Web サーバの設定］画面の「項目ごとに設定します。」の「追加ディレクティブ」の ProxyPass ディレクティブ
			［Web サーバの設定］画面の「設定ファイルの内容を直接設定します。」の「設定ファイルの内容」の ProxyPass ディレクティブ
		ファイル編集	httpsd.conf の ProxyPass ディレクティブ※

注※　httpsd.conf の詳細については，マニュアル「Hitachi Web Server」を参照してください。

(2) 静的コンテンツをキャッシュするためのチューニングパラメタ

静的コンテンツをキャッシュするためのチューニングパラメタについて説明します。これらのチューニングパラメタは，Web コンテナ単位または Web アプリケーション単位に設定します。

Web コンテナ単位に設定するチューニングパラメタの設定方法および設定個所について，次の表に示します。

表 C-17　静的コンテンツをキャッシュするためのチューニングパラメタ（Web コンテナ単位で設定する項目）（推奨手順以外の場合）

設定項目	設定方法	設定個所
静的コンテンツのキャッシュを使用するかどうかの選択	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「静的コンテンツキャッシュ機能」
	ファイル編集	usrconf.properties の webserver.static_content.cache.enabled キー

設定項目	設定方法	設定個所
Web アプリケーション単位のメモリサイズの上限値の設定	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「キャッシュサイズ」
	ファイル編集	usrconf.properties の webserver.static_content.cache.size キー
キャッシュする静的コンテンツのファイルサイズの上限値の設定	運用管理ポータル	［Web コンテナの設定］画面の「Web コンテナの設定」の「ファイルサイズ」
	ファイル編集	usrconf.properties の webserver.static_content.cache.filesize.threshold キー

Web アプリケーション単位に設定するチューニングパラメタについては，推奨手順の場合と違いがありません。

(3) リダイレクタによってリクエストを振り分けるためのチューニングパラメタ

リダイレクタによってリクエストを振り分けるためのチューニングパラメタは，Web サーバの動作を定義するファイルのパラメタとして指定します。設定個所，ファイルおよびパラメタは，使用する Web サーバごとに異なります。

なお，この定義は，Web サーバ連携の場合だけできます。インプロセス HTTP サーバを使用している場合は定義できません。

Web サーバごとの設定方法および設定個所を次に示します。

表 C-18　リダイレクタによってリクエストを振り分けるためのチューニングパラメタ（推奨手順以外の場合）

Web サーバの種類	設定方法	設定個所
Hitachi Web Server	運用管理ポータル	［論理 Web サーバの定義］画面の「マッピング定義」
	ファイル編集	mod_jk.conf のマッピング定義（JkMount パラメタ）
Microsoft IIS	ファイル編集	uriworkermap.properties

付録 C.6　CTM の動作を最適化するチューニングパラメタ（推奨手順以外の方法）

ここでは，CTM の動作の最適化で使用するチューニングパラメタの設定個所についてまとめて示します。

(1) CTM ドメインマネジャおよび CTM デーモンの稼働状態の監視間隔を設定するチューニングパラメタ

CTM ドメインマネジャの稼働状態の監視間隔をチューニングするパラメタを次に示します。

監視間隔は，送信間隔×係数の値になります。

表 C-19　CTM ドメインマネジャの稼働状態の監視間隔をチューニングするパラメタ（推奨手順以外の方法）

対象	設定項目	設定方法	設定個所
同じネットワークセグメント内にある CTM ドメインマネジャ	送信間隔	運用管理ポータル	論理 CTM ドメインマネジャの［ネットワーク設定］画面の「CTM ドメイン構成情報の送信間隔」
		コマンド実行	ctmdmstart コマンドの引数「-CTMSendInterval」
	係数	運用管理ポータル	論理 CTM ドメインマネジャの［ネットワーク設定］画面の「ドメインマネジャ生存判定間隔係数」
		コマンド実行	ctmdmstart コマンドの引数「-CTMAliveCheckCount」
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	送信間隔	運用管理ポータル	論理 CTM ドメインマネジャの［ネットワーク設定］画面の「指定ホストへの構成情報の送信間隔」
		コマンド実行	ctmdmstart コマンドの引数「-CTMSendHostInterval」
	係数	運用管理ポータル	論理 CTM ドメインマネジャの［ネットワーク設定］画面の「ドメインマネジャ生存判定間隔係数」
		コマンド実行	ctmdmstart コマンドの引数「-CTMAliveCheckCount」

CTM デーモンの稼働状態の監視間隔をチューニングするパラメタを次に示します。

表 C-20　CTM デーモンの稼働状態の監視間隔をチューニングするパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
CTM デーモン間転送時のタイムアウト	運用管理ポータル	論理 CTM の［CTM 間通信の設定］画面の「リクエスト転送時のタイムアウト時間」
	コマンド実行	ctmstart コマンドの引数「-CTMDCSendTimeOut」

(2) 負荷状況監視間隔を設定するチューニングパラメタ

負荷状況監視間隔のチューニングパラメタを次に示します。

表 C-21　負荷情報監視間隔のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
CTM デーモン間転送時のタイムアウト	運用管理ポータル	論理 CTM の［スケジューリングの設定］画面の「負荷情報監視間隔」
	コマンド実行	ctmstart コマンドの引数「-CTMLoadCheckInterval」

(3) CTM デーモンのタイムアウト閉塞を設定するチューニングパラメタ

タイムアウト閉塞は，タイムアウト発生回数と監視間隔を設定しておくことによって，実行されます。

CTM デーモンのタイムアウト閉塞のチューニングパラメタを次に示します。

表 C-22　CTM デーモンのタイムアウト閉塞のチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
タイムアウト発生回数	運用管理ポータル	論理 CTM の［スケジューリングの設定］画面の「タイムアウト閉塞」の「自動閉塞するタイムアウト発生回数」
	コマンド実行	ctmstart コマンドの引数「-CTMWatchRequest」（一つ目のオプション引数）
監視時間間隔	運用管理ポータル	論理 CTM の［スケジューリングの設定］画面の「タイムアウト閉塞」の「監視時間間隔」
	コマンド実行	ctmstart コマンドの引数「-CTMWatchRequest」（二つ目のオプション引数）

(4) CTM で振り分けるリクエストの優先順位を設定するチューニングパラメタ

CTM で振り分けるリクエストの優先順位の設定は，EJB クライアントアプリケーションの場合と，J2EE サーバの場合で異なります。また，J2EE サーバの場合，システムの構築方法によって設定個所が異なります。CTM で振り分けるリクエストの優先順位を設定するチューニングパラメタを次に示します。

表 C-23　CTM で振り分けるリクエストの優先順位を設定するチューニングパラメタ（推奨手順以外の方法）

設定単位	設定方法	設定個所
J2EE サーバ	運用管理ポータル	論理 J2EE サーバの［EJB コンテナの設定］画面の「CTM の設定」の「リクエストの優先順位」
	ファイル編集	usrconf.properties の ejbserver.client.ctm.RequestPriority キー

EJB クライアントアプリケーション単位のチューニングパラメタについては，推奨手順の場合と違いがありません。

付録 C.7　Persistent Connection についてのチューニングパラメタ（推奨手順以外の方法）

Persistent Connection についてのチューニングパラメタについて説明します。

この項目は，Web フロントシステムの場合で，インプロセス HTTP サーバを使用するときにチューニングを検討してください。

表 C-24　Persistent Connection について設定するチューニングパラメタ（推奨手順以外の方法）

設定項目	設定方法	設定個所
Persistent Connection 数の上限値	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Persistent コネクション」の「上限値」
	usrconf.properties	webserver.connector.inprocess_http.persistent_connection.max_connections キー
リクエスト処理回数の上限値	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Persistent コネクション」の「リクエスト処理回数の上限値」
	usrconf.properties	webserver.connector.inprocess_http.persistent_connection.max_requests キー
タイムアウト	運用管理ポータル	［通信・スレッド制御に関する設定］画面の「Persistent コネクション」の「タイムアウト」
	usrconf.properties	webserver.connector.inprocess_http.persistent_connection.timeout キー

付録 C.8　バッチサーバのフルガーベージコレクションを発生させるしきい値を設定するためのチューニングパラメタ（推奨手順以外の方法）

バッチサーバのフルガーベージコレクションを実行するしきい値を設定するために使用

するチューニングパラメタについて説明します。

この項目は，バッチサーバで，フルガーベージコレクションの実行を制御するときにチューニングを検討してください。

表 C-25　バッチサーバのフルガーベージコレクションを実行するしきい値を設定するチューニングパラメタ

設定項目	設定方法	設定個所
しきい値	運用管理ポータル	［コンテナの設定］画面の「拡張パラメタ」
	usrconf.properties	ejbserver.batch.gc.watch.threshold キー

付録 D　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 D.1　関連マニュアル

アプリケーションサーバのマニュアルについて次に示します。

- Cosminexus **アプリケーションサーバ** V8 **概説**（3020-3-U01）
  アプリケーションサーバの概要について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **ファーストステップガイド**（3020-3-U02）
  Application Server または Developer を使用して，サンプルプログラムを動かすためのシステムを構築する手順について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **システム構築・運用ガイド**（3020-3-U04）
  セットアップウィザードおよび Smart Composer 機能を使用したシステムの構築・運用の手順について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 基本・開発編** (Web **コンテナ**)（3020-3-U05）
  アプリケーションサーバで提供する Web コンテナの機能，および Web コンテナに関連する機能（Web サーバ，サーブレット／ JSP など）について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 基本・開発編** (EJB **コンテナ**)（3020-3-U06）
  アプリケーションサーバで提供する EJB コンテナの機能，および EJB コンテナに関連する機能（EJB，EJB クライアントなど）について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 基本・開発編** (**コンテナ共通機能**)（3020-3-U07）
  Web コンテナおよび EJB コンテナで共通して利用する機能について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 拡張編**（3020-3-U08）
  アプリケーションサーバで提供する拡張機能（セッションフェイルオーバ機能，バッチサーバ，CTM など）について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 運用／監視／連携編**（3020-3-U09）
  アプリケーションサーバで提供する運用・監視機能，およびほかのプログラムとの連携について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 保守／移行／互換編**（3020-3-U10）
  アプリケーションサーバで構築したシステムの保守に関する機能，移行情報，および互換用機能について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **アプリケーション設定操作ガイド**（3020-3-U12）
  アプリケーションサーバで動作するアプリケーションの操作方法について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **運用管理ポータル操作ガイド**（3020-3-U13）

運用管理ポータルの使用方法について説明しています。

- Cosminexus **アプリケーションサーバ** V8 **リファレンス コマンド編**（3020-3-U14）
  アプリケーションサーバを構築・運用するときに使用するコマンドについて説明しています。
- Cosminexus **アプリケーションサーバ** V8 **リファレンス 定義編 (サーバ定義)**（3020-3-U15）
  アプリケーションサーバを構築・運用するとき，またはアプリケーションを開発するときに使用するファイルのうち，J2EE サーバや Management Server などのサーバの定義に使用するファイルの形式について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **リファレンス 定義編 (アプリケーション／リソース定義)**（3020-3-U16）
  アプリケーションサーバを構築・運用するとき，またはアプリケーションを開発するときに使用するファイルのうち，アプリケーションやリソースの属性設定に使用するファイルの形式について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **仮想化システム構築・運用ガイド**（3020-3-U18）
  アプリケーションサーバを仮想化したサーバ上に構築する場合の設計，構築，運用の手順について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **アプリケーション開発ガイド**（3020-3-U25）
  アプリケーションサーバで動作させるアプリケーションの開発方法について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **リファレンス** API **編**（3020-3-U26）
  アプリケーションを開発するときに使用する API の形式について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **メッセージ** 1 KDAL-KDCG **および** Hitachi Web Server **編**（3020-3-U41）
  アプリケーションサーバで出力される KDAL から KDCG までのメッセージ，および Hitachi Web Server のメッセージについて説明しています。
- Cosminexus **アプリケーションサーバ** V8 **メッセージ** 2 KDJE-KDJW **編**（3020-3-U42）
  アプリケーションサーバで出力される KDJE から KDJW までのメッセージについて説明しています。
- Cosminexus **アプリケーションサーバ** V8 **メッセージ** 3 KECX-KEDT ／ KEOS02000-29999 ／ KEUC-KFRM **編**（3020-3-U43）
  アプリケーションサーバで出力される KECX から KEDT までのメッセージ，KEOS02000 から KEOS29999 までのメッセージ，および KEUC から KFRM までのメッセージについて説明しています。
- Cosminexus **アプリケーションサーバ** V8 **メッセージ** 4 **監査ログ編**（3020-3-U44）
  アプリケーションサーバで出力される監査ログメッセージについて説明しています。

また，このマニュアルと関連するこのほかのマニュアルを次に示します。必要に応じてお読みください。

- Hitachi Web Server（3020-3-U17）

- Cosminexus アプリケーションサーバ V8 Cosminexus Reliable Messaging（3020-3-U21）
- VisiBroker Version 5 Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド（3020-3-U28）
- VisiBroker Version 5 Borland(R) Enterprise Server VisiBroker(R) プログラマーズリファレンス（3020-3-U29）
- スケーラブルデータベースサーバ HiRDB Version 8 解説（UNIX(R) 用）（3000-6-351）
- スケーラブルデータベースサーバ HiRDB Version 8 システム定義（UNIX(R) 用）（3000-6-353）
- HiRDB Version 9 解説（UNIX(R) 用）（3000-6-451）
- HiRDB Version 9 システム定義（UNIX(R) 用）（3000-6-453）
- スケーラブルデータベースサーバ HiRDB Version 8 解説（Windows(R) 用）（3020-6-351）
- スケーラブルデータベースサーバ HiRDB Version 8 システム定義（Windows(R) 用）（3020-6-353）
- HiRDB Version 9 解説（Windows(R) 用）（3020-6-451）
- HiRDB Version 9 システム定義（Windows(R) 用）（3020-6-453）
- スケーラブルデータベースサーバ HiRDB Version 8 UAP 開発ガイド（3020-6-356）
- スケーラブルデータベースサーバ HiRDB Version 8 XDM/RD E2 接続機能（3020-6-365）
- HiRDB Version 9 UAP 開発ガイド（3020-6-456）
- HiRDB Version 9 XDM/RD E2 接続機能（3020-6-465）
- JP1 Version 8 JP1/Automatic Job Management System 2 設計・運用ガイド（3020-3-K22）
- JP1 Version 9 JP1/Automatic Job Management System 3 設計ガイド（システム構築編）（3020-3-S03）
- VOS3　データマネジメントシステム XDM E2 系 システム定義（XDM/BASE・SD・TM2）（6190-6-625）
- VOS3　Database Connection Server（6190-6-648）

なお，このマニュアルでは，次のマニュアルについて，対象 OS およびバージョン番号を省略して表記しています。マニュアルの正式名称とこのマニュアルでの表記を次の表に示します。

正式名称	このマニュアルでの表記
Cosminexus アプリケーションサーバ V8 概説	Cosminexus アプリケーションサーバ 概説
Cosminexus アプリケーションサーバ V8 ファーストステップガイド	Cosminexus アプリケーションサーバ ファーストステップガイド

正式名称	このマニュアルでの表記
Cosminexus アプリケーションサーバ V8 システム構築・運用ガイド	Cosminexus アプリケーションサーバ システム構築・運用ガイド
Cosminexus アプリケーションサーバ V8 機能解説 基本・開発編 (Web コンテナ)	Cosminexus 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)
Cosminexus アプリケーションサーバ V8 機能解説 基本・開発編 (EJB コンテナ)	Cosminexus 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)
Cosminexus アプリケーションサーバ V8 機能解説 基本・開発編 (コンテナ共通機能)	Cosminexus 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)
Cosminexus アプリケーションサーバ V8 機能解説 拡張編	Cosminexus アプリケーションサーバ 機能解説 拡張編
Cosminexus アプリケーションサーバ V8 機能解説 運用／監視／連携編	Cosminexus アプリケーションサーバ 機能解説 運用／監視／連携編
Cosminexus アプリケーションサーバ V8 機能解説 保守／移行／互換編	Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編
Cosminexus アプリケーションサーバ V8 アプリケーション設定操作ガイド	Cosminexus 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
Cosminexus アプリケーションサーバ V8 運用管理ポータル操作ガイド	Cosminexus アプリケーションサーバ 運用管理ポータル操作ガイド
Cosminexus アプリケーションサーバ V8 リファレンス コマンド編	Cosminexus アプリケーションサーバ リファレンス コマンド編
Cosminexus アプリケーションサーバ V8 リファレンス 定義編 (サーバ定義)	Cosminexus 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)
Cosminexus アプリケーションサーバ V8 リファレンス 定義編 (アプリケーション／リソース定義)	Cosminexus 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)
Cosminexus アプリケーションサーバ V8 仮想化システム構築・運用ガイド	Cosminexus アプリケーションサーバ 仮想化システム構築・運用ガイド
Cosminexus アプリケーションサーバ V8 アプリケーション開発ガイド	Cosminexus アプリケーションサーバ アプリケーション開発ガイド

正式名称	このマニュアルでの表記
Cosminexus アプリケーションサーバ V8 リファレンス API 編	Cosminexus アプリケーションサーバ リファレンス API 編
Cosminexus アプリケーションサーバ V8 メッセージ 1 KDAL-KDCG および Hitachi Web Server 編	Cosminexus アプリケーションサーバ メッセージ 1
Cosminexus アプリケーションサーバ V8 メッセージ 2 KDJE-KDJW 編	Cosminexus アプリケーションサーバ メッセージ 2
Cosminexus アプリケーションサーバ V8 メッセージ 3 KECX-KEDT ／ KEOS02000-29999 ／ KEUC-KFRM 編	Cosminexus アプリケーションサーバ メッセージ 3
Cosminexus アプリケーションサーバ V8 メッセージ 4 監査ログ編	Cosminexus アプリケーションサーバ メッセージ 4
Cosminexus アプリケーションサーバ V8 Cosminexus Reliable Messaging	Cosminexus Reliable Messaging
VisiBroker Version 5 Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド	Borland(R) Enterprise Server VisiBroker(R) デベロッパーズガイド
VisiBroker Version 5 Borland(R) Enterprise Server VisiBroker(R) プログラマーズリファレンス	Borland(R) Enterprise Server VisiBroker(R) プログラマーズリファレンス
JP1 Version 8 JP1/Automatic Job Management System 2 設計・運用ガイド	JP1/Automatic Job Management System 2 設計・運用ガイド
JP1 Version 9 JP1/Automatic Job Management System 3 設計ガイド	JP1/Automatic Job Management System 設計ガイド
HiRDB Version 9 解説（UNIX(R) 用）	HiRDB 解説
HiRDB Version 9 解説（Windows(R) 用）	
スケーラブルデータベースサーバ HiRDB Version 8 解説（UNIX(R) 用）	
スケーラブルデータベースサーバ HiRDB Version 8 解説（Windows(R) 用）	
HiRDB Version 9 システム定義（UNIX(R) 用）	HiRDB システム定義
HiRDB Version 9 システム定義（Windows(R) 用）	
スケーラブルデータベースサーバ HiRDB Version 8 システム定義（UNIX(R) 用）	
スケーラブルデータベースサーバ HiRDB Version 8 システム定義（Windows(R) 用）	
HiRDB Version 9 UAP 開発ガイド	HiRDB UAP 開発ガイド
スケーラブルデータベースサーバ HiRDB Version 8 UAP 開発ガイド	
HiRDB Version 9 XDM/RD E2 接続機能	HiRDB XDM/RD E2 接続機能

正式名称	このマニュアルでの表記
スケーラブルデータベースサーバ HiRDB Version 8 XDM/RD E2 接続機能	

付録 D.2　このマニュアルでの表記

このマニュアルで使用している表記と，対応する製品名を次に示します。

表記		製品名
Application Server	Application Server Enterprise	uCosminexus Application Server Enterprise
	Application Server Standard	uCosminexus Application Server Standard
Developer	Developer Professional	uCosminexus Developer Professional
	Developer Standard	uCosminexus Developer Standard
Eclipse		Eclipse 3.6.1
HiRDB または HiRDB サーバ	HiRDB Server	HiRDB Server Version 9
	HiRDB/Parallel Server	HiRDB/Parallel Server Version 8
	HiRDB/Single Server	HiRDB/Single Server Version 8
HiRDB Run Time または HiRDB クライアント		HiRDB/Developer's Kit Version 8
		HiRDB/Developer's Kit Version 9
		HiRDB/Run Time Version 8
		HiRDB/Run Time Version 9
IPF		Itanium(R) Processor Family
JP1/AJS	JP1/AJS - Agent	JP1/Automatic Job Management System 2 - Agent ※
		JP1/Automatic Job Management System 3 - Agent
	JP1/AJS - Manager	JP1/Automatic Job Management System 2 - Manager ※
		JP1/Automatic Job Management System 3 - Manager
	JP1/AJS - View	JP1/Automatic Job Management System 2 - View ※
		JP1/Automatic Job Management System 3 - View
JP1/AJS2 - SO	JP1/AJS2 - SO Manager	JP1/Automatic Job Management System 2 - Scenario Operation Manager
	JP1/AJS2 - SO View	JP1/Automatic Job Management System 2 - Scenario Operation View

表記			製品名
JP1/IM	JP1/IM - CM		JP1/Integrated Management - Central Information Master
	JP1/IM - Manager		JP1/Integrated Management - Manager
	JP1/IM - View		JP1/Integrated Management - View
Oracle	Oracle10g		Oracle 10*g*
			Oracle 10*g* R2
			Oracle Database 10*g*
	Oracle11g		Oracle Database 11*g*
			Oracle Database 11*g* R2
	Oracle9i		Oracle9*i*
			Oracle9*i* R2
UNIX	AIX		AIX 5L V5.3
			AIX V6.1
			AIX V7.1
	HP-UX または HP-UX（IPF）		HP-UX 11i V2（IPF）
			HP-UX 11i V3（IPF）
	Linux	Linux（IPF）	Red Hat Enterprise Linux(R) AS 4（IPF）
			Red Hat Enterprise Linux(R) 5（Intel Itanium）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（Intel Itanium）
		Linux（x86/ AMD64 & Intel EM64T）	Red Hat Enterprise Linux(R) AS 4（x86）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（x86）
			Red Hat Enterprise Linux(R) ES 4（x86）
			Red Hat Enterprise Linux(R) 5（x86）
			Red Hat Enterprise Linux(R) AS 4(AMD64 & Intel EM64T)
			Red Hat Enterprise Linux(R) 5 Advanced Platform（AMD/Intel 64）
			Red Hat Enterprise Linux(R) ES 4(AMD64 & Intel EM64T)
			Red Hat Enterprise Linux(R) 5（AMD/Intel 64）
			Red Hat Enterprise Linux(R) Server 6(32-bit x86)

表記			製品名
			Red Hat Enterprise Linux(R) Server 6（64-bit x86_64）
	Solaris		Solaris 10（SPARC）
			Solaris 10（x64）
			Solaris 9（SPARC）
Web Redirector			uCosminexus Web Redirector
XDM/RD E2			VOS3 XDM/RD E2

注※　総称して，JP1/AJS2 と表記することがあります。

なお，Application Server および Developer を総称して，アプリケーションサーバと表記します。

また，Linux に関しては，バージョンごとに次のように表記することがあります。

表記	OS 名
Red Hat Enterprise Linux 4	Red Hat Enterprise Linux(R) AS 4（AMD64 & Intel EM64T）
	Red Hat Enterprise Linux(R) ES 4（AMD64 & Intel EM64T）
	Red Hat Enterprise Linux(R) AS 4（IPF）
	Red Hat Enterprise Linux(R) AS 4（x86）
	Red Hat Enterprise Linux(R) ES 4（x86）
Red Hat Enterprise Linux 5	Red Hat Enterprise Linux(R) 5 Advanced Platform（AMD/Intel 64）
	Red Hat Enterprise Linux(R) 5 Advanced Platform（Intel Itanium）
	Red Hat Enterprise Linux(R) 5 Advanced Platform（x86）
	Red Hat Enterprise Linux(R) 5（AMD/Intel 64）
	Red Hat Enterprise Linux(R) 5（Intel Itanium）
	Red Hat Enterprise Linux(R) 5（x86）
Red Hat Enterprise Linux Server 6	Red Hat Enterprise Linux(R) Server 6（32-bit x86）
	Red Hat Enterprise Linux(R) Server 6（64-bit x86_64）

このマニュアルで使用している表記と，対応するアプリケーションサーバの機能名を次に示します。

表記	アプリケーションサーバの機能名
CJMSP ブローカー	Cosminexus JMS プロバイダのブローカー機能
CJMSP リソースアダプタ	Cosminexus JMS プロバイダのリソースアダプタ
Cosminexus Developer's Kit for Java	Cosminexus Developer's Kit for Java™
Cosminexus RM	Cosminexus Reliable Messaging
CTM	Cosminexus Component Transaction Monitor
DB Connector for Cosminexus RM	DB Connector for Cosminexus Reliable Messaging
Management Server	Cosminexus Management Server
PRF	Cosminexus Performance Tracer
TPBroker	Cosminexus TPBroker
Server Plug-in	Cosminexus Server Plug-in
Smart Composer	Cosminexus Smart Composer

このマニュアルで使用している表記と，対応する Java 関連用語を次に示します。

表記	Java 関連用語
Connector 1.0	J2EE™ Connector Architecture 1.0
Connector 1.5	J2EE™ Connector Architecture 1.5
DI	Dependency Injection
EAR	Enterprise ARchive
EJB または Enterprise JavaBeans	Enterprise JavaBeans™
J2EE または Java 2 Platform, Enterprise Edition	J2EE™
	Java™ 2 Platform, Enterprise Edition
JAR	Java™ Archive
Java	Java™
Java 2 Runtime Environment, Standard Edition	Java™ 2 Runtime Environment, Standard Edition
JavaBeans	JavaBeans™
Java EE または Java Platform, Enterprise Edition	Java™ Platform, Enterprise Edition
JavaMail	JavaMail™
Java SE	Java™ Platform, Standard Edition
JavaVM	Java™ Virtual Machine
JCA	J2EE™ Connector Architecture

表記	Java 関連用語
JDBC	JDBC™
	Java™ Database Connectivity
JDK	JDK™
	Java™ Development Kit
JMS	Java™ Message Service
JNDI	Java™ Naming and Directory Interface™
JNI	Java™ Native Interface
JSP	JSP™
	JavaServer Pages™
JTA	Java™ Transaction API
Servlet またはサーブレット	Java™ Servlet
WAR	Web ARchive

付録 D.3　英略語

このマニュアルで使用している英略語を次に示します。

英略語	英字での表記
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BMP	Bean-Managed Persistence
BMT	Bean-Managed Transaction
CMP	Container-Managed Persistence
CMR	Container-Managed Relationship
CMT	Container-Managed Transaction
CORBA	Common Object Request Broker Architecture
CSV	Comma Separated Value
CUI	Character User Interface
DB	Database
DBMS	Database Management System
DD	Deployment Descriptor
DIT	Directory Information Tree
DMZ	Demilitarized Zone
DN	Distinguished Name

英略語	英字での表記
DNS	Domain Name System
DTD	Document Type Definition
EAR	Enterprise ARchive
EIS	Enterprise Information System
EL	Expression Language
EUC	Extended UNIX Code
GUI	Graphical User Interface
HA	High Availability
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Security
IDE	Integrated Development Environment
IIOP	Internet Inter-Orb Protocol
JIS	Japanese Industrial Standards
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
MIB	Management Information Base
OID	Object Identifier
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating System
OTM	Object Transaction Monitor
OTS	Object Transaction Service
PIM	Platform Independent Model
POA	Portable Object Adapter
POP3	Post Office Protocol - Version 3
PSM	Platform Specific Model
RAC	Real Application Clusters
RDB	Relational Database
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SFO	Session Fail Over
SHA	Secure Hash Algorithm

英略語	英字での表記
SOA	Service Oriented Architecture
SPI	Service Provider Interface
SPP	Service Providing Program
SSL	Secure Sockets Layer
SUP	Service Using Program
TCS	Transaction Context Server
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	UCS Transformation Format
VM	Virtual Machine
WSDL	Web Service Description Language
XML	Extensible Markup Language

付録 D.4　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

付録 E　用語解説

アプリケーションサーバで使用する用語について説明します。

（数字）

1.4 モード

サーバの動作モードです。J2EE 1.4 以降の機能を使用できます。データベースを含む複数のリソースのトランザクション管理ができます。

1:1 系切り替えシステム

実行系のアプリケーションサーバと待機系のアプリケーションサーバが 1:1 で対応しているシステムです。実行系のアプリケーションサーバに障害が発生すると，待機系のアプリケーションサーバで処理を引き継ぎます。

（英字）

Application Server

アプリケーションサーバの実行環境を構築する基盤製品です。Application Server Standard と，Application Server Enterprise の総称です。

CJMSP ブローカー

Cosminexus JMS プロバイダで使用するプロセスです。メッセージの送信先を管理します。

CJMSP リソースアダプタ

Cosminexus JMS プロバイダで使用するリソースアダプタです。
J2EE サーバから CJMSP ブローカーに接続するために使用します。Connector 1.5 仕様に準拠しています。

CORBA ネーミングサービス

CORBA の仕様に準拠した，リモートオブジェクトの格納場所を管理するためのネーミングサービスです。アプリケーションサーバの構成ソフトウェアである Cosminexus TPBroker によって提供される機能です。

Cosminexus Component Container

サーバ・サイドの業務処理プログラム（ビジネスロジック）をコンポーネントとして実行するための構成ソフトウェアです。また，アプリケーションサーバの運用管理をするための Management Server，統合ユーザ管理，snapshot ログ収集などの機能も提供しています。

Cosminexus Component Transaction Monitor

クライアントからのリクエストのスケジューリングを実現するための構成ソフトウェアです。

Cosminexus JMS プロバイダ

アプリケーションサーバで提供している JMS プロバイダです。JMS1.1 仕様に基づいた機能を提供しています。CJMSP ブローカーと CJMSP リソースアダプタによって構成されています。

Cosminexus Performance Tracer

リクエストが処理されるときに，決められたポイントごとに各機能が出力する性能解析情報をファイルに出力するための構成ソフトウェアです。

Cosminexus Reliable Messaging

アプリケーションサーバで構築したシステム上の J2EE アプリケーションが，メッセージを使用して非同期に通信するための構成ソフトウェアです。JMS インタフェースでのメッセージ通信機能を J2EE アプリケーションに提供します。

Cosminexus アプリケーションサーバ

アプリケーションサーバを中核とした，性能および信頼性の高いアプリケーションを実行および開発するためのシステム構築基盤製品です。

CTM

Cosminexus Component Transaction Monitor のことです。リクエストのスケジューリングをするための構成ソフトウェアです。

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。リクエストのスケジューリングを実現する，スケジューラとして機能します。
Management Server を利用している場合，CTM デーモンは論理サーバとして扱えます。

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。ホスト内の CTM デーモンの情報を，CTM ドメイン内のほかの CTM ドメインマネジャに配布します。
Management Server を利用している場合，CTM ドメインマネジャは論理サーバとして扱えます。

CTM レギュレータ

EJB クライアントから 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。TPBroker クライアント（CORBA クライアント）から J2EE サーバ内で動作している 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も提供します。

DB Connector

データベースに接続するためのリソースアダプタです。

DD

アプリケーションを運用環境に配置するときの定義情報を記述したものです。

EIS

データベースやトランザクションサーバなど，企業内に構築されているバックエンドシステムです。

EJB クライアント

J2EE サーバ上で開始されている Enterprise Bean を呼び出すクライアントプログラムです。次の 3

種類があります。
- EJB クライアントアプリケーション
- サーブレットまたは JSP などの Web アプリケーション
- ほかの Enterprise Bean

EJB クライアントアプリケーション
Enterprise Bean を呼び出す Java アプリケーションです。

EJB コンテナ
Enterprise Bean を制御する実行環境です。また，通信，トランザクション管理などのシステムレベルのサービスも提供します。Enterprise Bean の実体は，EJB コンテナの中で実行されます。

Explicit ヒープ
明示管理ヒープ機能で使用するメモリ空間です。Java ヒープ外にあるため，フルガーベージコレクションの対象になりません。

HA モニタ
クラスタソフトウェアの一つです。
システムを監視し，障害発生時などにシステムの切り替えを実行するプログラムです。HA モニタと連携することで，アプリケーションサーバで構築したシステムの信頼性や稼働率を向上できます。
AIX，HP-UX または Linux を使用しているときに連携できます。
HA モニタと連携することで，1：1 系切り替えシステム，相互系切り替えシステム，N：1 リカバリシステム，および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が実現できます。

HTTP セッション
javax.servlet.http.HttpServletRequest クラスの getSession メソッドで取得される HttpSession オブジェクトです。「HTTP セッションに格納するオブジェクト」に加え，そのほかに HTTP セッションがメンバとして参照するオブジェクトを含む，HTTP セッション全体を指します。

HTTP セッション管理用オブジェクト
Web コンテナの内部で使用され，HTTP セッションを管理するオブジェクトです。

HTTP セッションに格納するオブジェクト
javax.servlet.http.HttpSession クラスの setAttribute メソッドで HTTP セッションへ格納されるユーザオブジェクトです。
HTTP セッションの属性ともいいます。

HTTP セッションに関するオブジェクト
「HTTP セッション管理用オブジェクト」と「HTTP セッション」とをまとめて表現した用語です。

J2EE アプリケーション
JSP，サーブレット，EnterpriseBean などで構成されるアプリケーションです。アプリケーションサーバで扱う J2EE アプリケーションの形式には，EAR ファイル形式でパッケージ化されたアーカイブ形式のアプリケーションと，アーカイブ化しない展開ディレクトリ形式のアプリケーションがあります。EAR ファイル形式でパッケージ化されたアプリケーションの場合，複数の EJB-JAR ファイル，複数の WAR ファイル，および一つの DD から構成されます。

J2EE コンテナ

J2EE アプリケーションを実行するためのサーバ基盤です。
J2EE コンポーネントへ各種 API を提供する，Web コンテナ，EJB コンテナから構成されます。

J2EE コンポーネント

サーブレット，JSP，Enterprise Bean などのユーザアプリケーションプログラムのことです。

J2EE サーバ

J2EE コンテナを生成，実行する環境です。

J2EE サーバモード

J2EE サーバの動作モードの一つです。このモードの場合，J2EE コンテナ上で動作するアプリケーションから，J2EE 関連の API を利用できます。なお，J2EE サーバモードには，1.4 モードと，互換用の動作モードであるベーシックモードがあります。

J2EE サービス

J2EE コンテナの部品機能として利用されます。J2EE コンポーネントに JNDI，JDBC，JTA，RMI-IIOP および JavaMail の API を提供します。

JP1

日立の統合システム運用管理ソフトウェアです。

JP1/AJS

日々の業務の中から，定型的・定期的なものを自動化して，システム運用に掛かるコストを削減し，少ない人員で確実な運用を実現するためのプログラム群です。JP1/AJS - Manager，JP1/AJS - Agent，JP1/AJS - View の製品で構成されます。

JP1/AJS2 - SO

一つ一つの運用手順を蓄積し，ライブラリとして統合する運用管理の基盤機能を提供するプログラムです。立案したシナリオに合う運用手順をライブラリから選択し，運用環境に合わせてカスタマイズすることで，業務運用の設計を容易に，かつ高品質で実現することができます。

JP1/IM

企業情報システム全体を統合管理する基盤になるプログラム群です。JP1/IM - Manager，JP1/IM - View，および JP1/IM - CM の三つの製品で構成されます。

JP1/IM - CM

業務システムを構成する各種のリソース（サーバ，ストレージなど）やシステム構成に関する情報を集約して管理するためのプログラムです。管理対象のリソースを任意にグルーピングできるため，システムの監視（性能，障害）や運用（設定配布，構成変更）の際に，グループ単位で管理したり操作したりできます。

Management Server

運用管理ドメインを構成するサーバプログラムです。運用管理ドメイン単位に一つ配置します。
Management Server は運用管理ドメイン内の各ホストに配置した運用管理エージェントに指示を出して，運用管理ドメイン全体の運用管理を実行します。

Microsoft Cluster Service

クラスタソフトウェアの一つです。
システムを監視し，障害発生時などにシステムの切り替えを実行するプログラムです。Microsoft Cluster Service と連携することで，アプリケーションサーバで構築したシステムの信頼性や稼働率を向上できます。Windows Server 2003 で使用できます。
Microsoft Cluster Service と連携したシステムでは，1：1 系切り替えシステム，相互系切り替えシステム，N：1 リカバリシステム，および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が実現できます。

N:1 リカバリシステム

クラスタ構成の実行系のアプリケーションサーバに対して，1 台のリカバリ専用サーバを待機系として配置したシステムです。リカバリ専用サーバでは，障害が発生した実行系のアプリケーションサーバのトランザクションの決着をします。

Outbound

Connector 1.5 仕様で定められた J2EE サーバとリソースアダプタ間の通信モデルの一つです。
J2EE サーバからリソースアダプタにアクセスする通信モデルです。

SFO サーバ

システム内の J2EE サーバの J2EE アプリケーション内で生成されたグローバルセッション情報を管理する J2EE サーバです。メモリセッションフェイルオーバ機能を使用するときにシステム構成に含めます。

SFO サーバアプリケーション

SFO サーバ上で稼働する J2EE アプリケーションを，SFO サーバアプリケーションといいます。
SFO サーバアプリケーションは EJB で実装されています。なお，SFO サーバアプリケーションはアプリケーションサーバで提供しています。

Smart Composer 機能

一般的な 3 階層モデルのアプリケーションサーバのシステムを，簡単に構築および運用できるよう支援する機能です。システム全体に対して，システムの設定，および J2EE アプリケーションやリソースアダプタのデプロイを一括で実行できるので，簡単，迅速にシステム構築ができます。
構築したシステムを一括で起動したり，一括でシステムの設定を変更したりするなど，運用機能についてもサポートしています。

TP1 インバウンドアダプタ

TP1 インバウンド連携機能で使用するリソースアダプタです。Connector 1.5 仕様に準拠しています。

TP1 インバウンド連携機能

TP1 インバウンドアダプタを使用して，OpenTP1 システムの SUP からアプリケーションサーバ上の Message-driven Bean を Inbound で呼び出す機能です。

uCosminexus Client

EJB クライアント環境を構築するための製品です。

usrconf.properties（ユーザプロパティファイル）

J2EE サーバ，バッチサーバ，Web コンテナサーバ，またはサーバ管理コマンドで使用する JavaVM のシステムプロパティを定義するファイルです。

Web アプリケーション

Web ブラウザを備えたクライアントを対象に作成されたアプリケーションです。具体的には，サーブレットプログラム，JSP ファイル，HTML/XML ドキュメントなどの集合体です。

Web コンテナ

Web アプリケーションを制御する実行環境です。また，セキュリティ，トランザクションなどの各種サービスも提供します。Web アプリケーションは，Web コンテナ上で動作します。
Java Servlet2.5 仕様，および JavaServer Pages Specification v2.1 仕様に準拠した Web アプリケーションを実行できます。

Web サーバ

Web ブラウザからのリクエスト受信および Web ブラウザへのデータ送信に関連する処理を実行するプログラムです。アプリケーションサーバでは，Hitachi Web Server，Microsoft IIS，またはインプロセス HTTP サーバを使用できます。インプロセス HTTP サーバは，J2EE サーバプロセス内で動作する Web サーバです。
なお，Management Server を利用する場合，Hitachi Web Server は論理サーバとして扱えます。

Web サーバ連携

アプリケーションサーバで使用する Web サーバとして，Hitachi Web Server または Microsoft IIS を使用する方法です。
Hitachi Web Server または Microsoft IIS に Cosminexus Component Container が提供するリダイレクタモジュールを組み込んで使用します。

Windows Server Failover Cluster

クラスタソフトウェアの一つです。
システムを監視し，障害発生時などにシステムの切り替えを実行するプログラムです。Windows Server Failover Cluster と連携することで，アプリケーションサーバで構築したシステムの信頼性や稼働率を向上できます。Windows Server 2008 で使用できます。

（ア行）

アウトプロセス

プロセスの起動のしかたです。アウトプロセスで起動させると，J2EE サーバのプロセス外で実行します。アウトプロセスでネーミングサービスを使用する場合，CORBA ネーミングサービスはユーザが起動する必要があります。

アプリケーションサーバ

情報システムの中間に位置し，ユーザの要求（プレゼンテーション層）とデータベースなどの業務システム（データ層）の処理を橋渡しするためのアプリケーション層を構築するためのミドルウェアです。

インプロセス

プロセスの起動のしかたです。インプロセスで起動させると，J2EE サーバのプロセス内で実行するように最適化されるので，パフォーマンスの高いシステムが実現できます。ネーミングサービス，トランザクションサービスおよび HTTP サーバ機能をインプロセスで起動できます。このとき，CORBA ネーミングサービス，インプロセス OTS およびインプロセス HTTP サーバは，J2EE サーバ起動時に自動で起動されます。

インプロセス HTTP サーバ

J2EE サーバのインプロセスで動作する Web サーバ機能です。Web コンテナの機能の一部として提供されます。

運用監視エージェント

ホスト上で動作する論理サーバの稼働状況を監視して，稼働情報を収集するエージェントプログラムです。

運用管理エージェント

運用管理者の代わりに，それぞれのホスト上の論理サーバを起動したり，設定ファイルを更新したりするエージェントプログラムです。

運用管理サーバ

Smart Composer 機能または運用管理ポータルでシステムを構築している場合に，Management Server を配置したホストです。運用管理ドメイン内の各ホストに配置されている運用管理エージェントに対して，管理操作を指示します。

運用管理ドメイン

Management Server が管理する論理サーバの集合です。同じ運用管理ポリシーが適用されます。

運用管理ポータル

Management Server を操作するための GUI です。Web ブラウザで表示します。

（カ行）

カスタムジョブ

JP1/AJS 以外のプログラムと JP1/AJS が連携するジョブを定義する場合に，目的のジョブを容易に作成するために使用できるジョブのテンプレートです。Windows の場合，アプリケーションサーバでは，「論理サーバ制御用カスタムジョブ」と「アプリケーション制御用カスタムジョブ」の 2 種類を提供しています。

機能レイヤ

性能解析トレース収集をする場合に，PRF トレースを出力する機能の層です。EJB クライアント，リダイレクタ，Web コンテナ，CTM，EJB コンテナ，JNDI，JTA，JCA/JDBC などがあります。

クラスタ

ある共通の機能を提供するサーバの集合です。
Management Server では，J2EE サーバクラスタまたは Web サーバクラスタを論理サーバとして扱えます。

クラスタソフトウェア

システムの信頼性向上，稼働率向上を目的として，システムの切り替えを実現するプログラムです。実行中のサーバシステムに障害が発生した場合，事前に待機しているサーバシステムに，直ちに自動で切り替えることができます。そのため，オペレータが特に意識することなく，システムの信頼性や稼働率を高められます。

アプリケーションサーバでは，クラスタソフトウェアを使用して，1:1 系切り替えシステム，N:1 リカバリシステム，相互系切り替えシステム，および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を実現できます。

AIX，HP-UX または Linux を使用している場合は，クラスタソフトウェアとして HA モニタが使用できます。Windows の場合は，クラスタソフトウェアとして Microsoft Cluster Service および Windows Server Failover Cluster が使用できます。

グローバル CORBA ネーミングサービス

CTM によってリクエストをスケジューリングする場合に，CTM ドメイン内に含まれる複数の J2EE サーバに登録されている業務処理プログラム（Stateless Session Bean）の情報を共有管理するネーミングサービスです。

系

系とは，業務処理に必要なハードウェアのほか，実行するプログラムや通信機器を含めたシステム全体の総称です。系の種類には，実行系と待機系があります。

系切り替え

システムに障害が発生した場合やシステムの保守のために，システムを切り替えることをいいます。系切り替えが行われると，待機系は業務処理を引き継いで実行系となります。一方，実行系は，さらに障害が発生した場合に備えて待機系となります。以降，系切り替えのたびに，実行と待機を交代して，どちらかの系で常に業務処理を実行するようにします。

現用系

最初に実行系として動作させる系を現用系といいます。系切り替えによって，実行系が待機系になった場合でも，呼び方は変わりません。

コネクションプーリング

サーブレット，JSP，Enterprise Bean などの J2EE コンポーネントや，バッチアプリケーションからリソースへのアクセス量に応じて，リソースコネクションをメモリ上にプーリングする機能です。リソースコネクションには，JDBC コネクション，リソースアダプタのコネクションなどがあります。コネクションをプーリングしておくことで，アプリケーションからのリソース接続要求を高速に処理できます。

（サ行）

サーバ管理コマンド

サーバで管理しているアプリケーションおよびリソースの設定をするためのコマンド群です。

サーブレットエンジンモード

サーバの動作モードの一つです。Web コンテナ部分だけを使用して，サーブレットエンジンを単独で動作させます。Enterprise Bean は動作しません。

サーブレットエンジンとは，サーブレット実行機能および JSP 実行機能を持つサーバのことです。
なお，サーブレットエンジンモードは互換用の動作モードとなります。

実行系
業務処理を実行させる系（実行中のサーバ）を指します。

シナリオ
JP1/AJS2 - SO で扱う，シナリオジョブに実行順序を付けてネットワーク化したオブジェクトです。
シナリオジョブとは，シナリオ中に定義されたコマンド，シェルスクリプト，Windows 実行ファイルなどを定義したオブジェクトです。
シナリオは JP1/AJS に登録して実行できます。

ジョブ
JP1/AJS で扱うシステム運用の各作業のことです。

スケールアウト
システム全体の処理性能を向上させることを目的として，サーバの台数を増やすことをいいます。

スケールイン
システムの規模を縮小する場合などに，サーバの台数を減らすことをいいます。

スマートエージェント
1.4 モードでグローバルトランザクションを使用する場合，または CTM を使用する場合に，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。
Management Server を利用する場合，スマートエージェントは論理サーバとして扱えます。CTM はスマートエージェントによって管理されます。
なお，グローバルトランザクションの場合でも，インプロセストランザクションサービスを利用するときは，スマートエージェントは不要です。

静的コンテンツ
HTML ファイルや画像ファイルなど，クライアントからの要求に対する応答に使用するファイルのうち，リクエスト内容に影響されないで常に同じ内容になるコンテンツのことです。

性能解析トレース
アプリケーションサーバで構築したシステムの性能を解析するためのトレース情報です。

性能解析トレースファイル
アプリケーションサーバで構築したシステムの性能を解析するためのトレース情報を CSV 形式で編集出力したテキストファイルです。

セッションフェイルオーバ機能
J2EE アプリケーション実行中の HttpSession オブジェクトに登録されたオブジェクトをセッション情報として管理し，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 サーバに引き渡す機能です。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。
セッションフェイルオーバ機能には，メモリセッションフェイルオーバ機能とデータベースセッションフェイルオーバ機能の 2 種類があります。メモリセッションフェイルオーバ機能の場合，セッション情報を SFO サーバで管理します。データベースセッションフェイルオーバ機能の場合，

セッション情報をデータベースで管理します。

セッションフェイルオーバサーバ

メモリセッションフェイルオーバ機能を使用する場合に，J2EE サーバを配置しないで，SFO サーバを配置して使用するホストです。

相互系切り替えシステム

クラスタソフトウェアを使用して実現できるシステムの一つです。アプリケーションサーバの実行系と待機系を 1：1 にする構成で，それぞれのアプリケーションサーバを実行系として稼働させながら，お互いの待機系にするシステムです。

相互スタンバイ

クラスタソフトウェアを使用した運用をする場合に，アプリケーションサーバの実行系と待機系を 1:1 にする構成で，それぞれのアプリケーションサーバを実行系として稼働させながら，お互いの待機系にすることです。

（タ行）

統合ネーミングスケジューラサーバ

CTM によるリクエストのスケジューリングをする場合に，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を配置して使用するホストです。統合ネーミングスケジューラサーバのレプリカを作成することで，可用性を向上できます。なお，統合ネーミングスケジューラサーバにも CTM デーモンは必要です。

動作モード

サーバの動作モードです。J2EE サーバモードとサーブレットエンジンモードがあります。J2EE サーバモードには，さらに，1.4 モード，ベーシックモードがあります。
ただし，ベーシックモードおよびサーブレットエンジンモードは旧バージョンとの互換用の動作モードとなります。

動的コンテンツ

サーブレットや JSP のように，クライアントからの要求に応じて動的に生成されるコンテンツのことです。

トランザクションサービス

グローバルトランザクションを使用する場合に，トランザクションを管理するサービスです。
TPBroker OTS によって提供されるサービス全体を表します。トランザクションサービスは，J2EE サーバのインプロセスで起動されます。

（ナ行）

ネーミングサービス

オブジェクトに名前を付けて格納場所を管理しておくことで，格納先を知らなくても名前からそのオブジェクトを利用できるようにするサービスです。アプリケーションサーバでは CORBA ネーミングサービスを利用します。

Management Server では，ネーミングサービスを論理サーバとして扱えます。

（ハ行）

バッチアプリケーション

バッチ処理を実装した Java アプリケーションです。バッチ実行コマンドを使用して，バッチサーバ上で実行します。バッチ実行コマンドを JP1/AJS のジョブとして定義しておくと，JP1/AJS からバッチアプリケーションを実行できます。

バッチサーバ

バッチアプリケーションを実行するためのサーバです。バッチアプリケーションの実行機能のほかに，バッチアプリケーションからデータベースに接続したり，EJB にアクセスしたりする機能も提供しています。

パフォーマンストレーサ

PRF デーモンのことです。
Management Server では，パフォーマンストレーサを論理サーバとして扱えます。

負荷分散機

Web ブラウザなどからのリクエストを一元的に受け付けて，同等の機能を持つ複数のサーバに転送して各サーバの負荷を分散させるための装置です。

ベーシックモード

旧バージョンとの互換性を確保するためのサーバの動作モードの一つです。
単一リソースとのトランザクション管理ができます。

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を使用して実現できるシステムの一つです。複数の実行系アプリケーションサーバと 1 台の待機系アプリケーションサーバを配置して，それぞれに Management Server および運用管理エージェントを配置したシステムです。

（マ行）

明示管理ヒープ機能

Java オブジェクトの配置先として，Explicit ヒープを使用する機能です。Explicit ヒープは Java ヒープ外にある，フルガーベージコレクションの対象にならない領域です。明示管理ヒープ機能を使用することによって，フルガーベージコレクションの発生を抑止できます。

（ヤ行）

ユーザプロパティファイル

usrconf.properties のことです。J2EE サーバ，Web コンテナサーバまたはサーバ管理コマンドを実行する JavaVM のシステムプロパティを指定します。なお，J2EE サーバ，Web コンテナサーバおよびサーバ管理コマンドで使用するユーザプロパティファイルは，それぞれ格納先および指定でき

るキーが異なります。

予備系

最初に待機系として起動する系を予備系といいます。系切り替えによって，待機系が実行系になった場合でも，呼び方は変わりません。

（ラ行）

ラウンドロビン検索

複数の CORBA ネーミングサービス上にある同一名称の EJB ホームオブジェクトを，ラウンドロビンポリシーに従ってルックアップする検索機能のことです。

ラウンドロビンポリシー

複数の CORBA ネーミングサービス上にある同一名称の EJB ホームオブジェクトをルックアップする場合に適用されるポリシーです。
EJB ホームオブジェクトリファレンスをラウンドロビン検索によって取得できるので，EJB クライアントでは意識しないで J2EE サーバに送信するリクエストの負荷分散を実現できます。

リカバリ専用サーバ

クラスタソフトウェアを使用して N:1 リカバリシステムを構築している場合に待機系として機能するアプリケーションサーバです。
障害が発生した実行系のアプリケーションサーバに未決着のトランザクションがあった場合に，系が切り替えられ，該当するトランザクションを解決してリソースを解放します。

リソースアダプタ

J2EE Connector Architecture によって，J2EE サーバまたはバッチサーバと，EIS を接続するための接続機能です。
アプリケーションサーバで構築したシステムでは，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 および DB Connector for Cosminexus RM を提供しています。また，OpenTP1 の SPP と接続するためのリソースアダプタである uCosminexus TP1 Connector，TP1/Message Queue と接続するためのリソースアダプタである TP1/Message Queue - Access，データベース上に実現したキューに接続するためのリソースアダプタである Cosminexus RM も使用できます。

リソースマネジャ

リソースを管理する機能です。DBMS などが該当します。

リダイレクタ

Web サーバに登録し Web コンテナへの接続を可能にするプラグインコンポーネント（ライブラリ）です。Web サーバに登録したリダイレクタによって，リクエストを URL パターンまたはラウンドロビン方式で複数の Web コンテナに振り分けて処理できます。

リバースプロキシサーバ

インターネットなどの外部ネットワークとアプリケーションサーバが配置されている内部ネットワークの間の DMZ に配置され，外部ネットワークからのリクエストを内部ネットワークに中継するための機能を持つ Web サーバです。

アプリケーションサーバでは，Hitachi Web Server をリバースプロキシサーバとして使用できます。リバースプロキシサーバには，リバースプロキシモジュールが組み込まれて動作します。

ローカルトランザクション

接続先のリソースマネジャによって管理されるトランザクションです。単一のリソースだけがトランザクションに参加できます。

論理サーバ

Management Server の運用管理の対象になる，サーバまたはクラスタです。サーバには，Web サーバ，J2EE サーバなどがあります。クラスタとは，ある共通の機能を提供するサーバの集合です。

論理ネーミングサービス

JNDI がラウンドロビン検索の対象にする複数の CORBA ネーミングサービスのことです。

論理ユーザサーバ

任意のサービスやプロセスを Management Server の管理対象として定義した論理サーバです。Management Server で起動・停止したり，ステータス監視したりできるようになります。

索引

記号

数字

A

C

D

E

H

I

J

M

N

O

P

S

T

U

W

あ

い

う

お

か

き

く

け

こ

さ

し

す

せ

そ

ち

て

と

ね

は

ふ

へ

ほ

め

ゆ

よ

ら

り

れ

ろ