HITACHI
Inspire the Next

Cosminexus V9 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)

解説書

3020-3-Y05-60

■ 対象製品

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

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。
なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

Active Directory は，米国 Microsoft Corporation の，米国およびその他の国における登録商標または商標です。
AX2000 は，A10 Networks, Inc.の商品名称です。
BIG-IP，3-DNS，iControl Services Manager，FirePass　および F5 は F5 Networks, Inc. の商標または登録商標です。
BSAFE は，米国 EMC コーポレーションの米国およびその他の国における商標または登録商標です。
CORBA は，Object Management Group が提唱する分散処理環境アーキテクチャの名称です。
gzip は，米国 FSF(Free Software Foundation)が配布しているソフトウェアです。
IIOP は，OMG 仕様による ORB(Object Request Broker)間通信のネットワークプロトコルの名称です。
Microsoft は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
MyEclipse は，米国 Genuitec 社の商品名称です。
OMG, CORBA, IIOP, UML, Unified Modeling Language, MDA, Model　Driven　Architecture は, Object Management Group, Inc.の米国及びその他の国における登録商標または商標です。
Oracle と Java は，Oracle Corporation 及びその子会社，関連会社の米国及びその他の国における登録商標です。
RSA は，米国 EMC コーポレーションの米国およびその他の国における商標または登録商標です。
SOAP（Simple Object Access Protocol）は，分散ネットワーク環境において XML ベースの情報を交換するための通信プロトコルの名称です。
SQL Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
UNIX は，The Open Group の米国ならびに他の国における登録商標です。
Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Vista は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
その他記載の会社名，製品名は，それぞれの会社の商標もしくは登録商標です。
Eclipse は，開発ツールプロバイダのオープンコミュニティである Eclipse Foundation, Inc.により構築された開発ツール統合のためのオープンプラットフォームです。
This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

■ マイクロソフト製品のスクリーンショットの使用について

Microsoft Corporation のガイドラインに従って画面写真を使用しています。

■ 発行

2015 年 4 月 3020-3-Y05-60

■ 著作権

変更内容

変更内容(3020-3-Y05-60) uCosminexus Application Server 09-70，uCosminexus Application Server(64) 09-70，uCosminexus Client 09-70，uCosminexus Developer 09-70，uCosminexus Service Architect 09-70，uCosminexus Service Platform 09-70，uCosminexus Service Platform(64) 09-70

追加・変更内容	変更個所
記載内容は変更なし（リンク情報だけを変更した）。	－

uCosminexus Application Server 09-60，uCosminexus Application Server(64) 09-60，uCosminexus Client 09-60，uCosminexus Developer 09-60，uCosminexus Service Architect 09-60，uCosminexus Service Platform 09-60，uCosminexus Service Platform(64) 09-60

追加・変更内容	変更個所
記載内容は変更なし（リンク情報だけを変更した）。	－

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

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

目次

1 アプリケーションサーバの機能

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

1.1　機能の分類

アプリケーションサーバは，Java EE 6 に準拠した J2EE サーバを中心としたアプリケーションの実行環境を構築したり，実行環境上で動作するアプリケーションを開発したりするための製品です。Java EE の標準仕様に準拠した機能や，アプリケーションサーバで独自に拡張された機能など，多様な機能を使用できます。目的や用途に応じた機能を選択して使用することで，信頼性の高いシステムや，処理性能に優れたシステムを構築・運用できます。

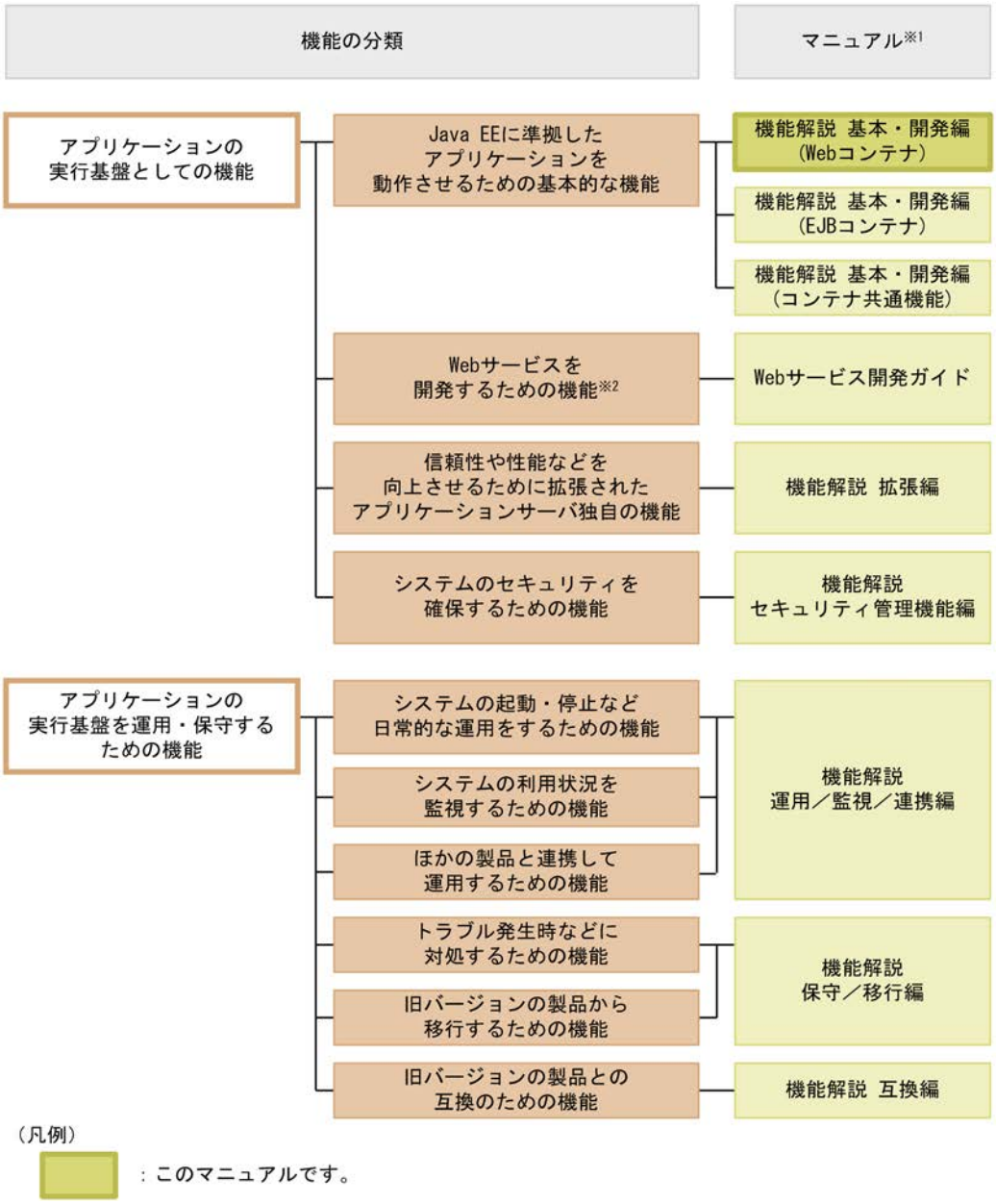
アプリケーションサーバの機能は，大きく分けて，次の二つに分類できます。

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

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

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

図 1-1　アプリケーションサーバの機能の分類と対応するマニュアル



注※1

マニュアル名称の「アプリケーションサーバ」を省略しています。

注※2

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

SOAP アプリケーションを開発・実行する場合

- アプリケーションサーバ SOAP アプリケーション開発の手引

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

- XML Security - Core ユーザーズガイド

- アプリケーションサーバ Web サービスセキュリティ構築ガイド

XML の処理について詳細を知りたい場合

- XML Processor ユーザーズガイド

ここでは，機能の分類について，マニュアルとの対応と併せて説明します。

1.1.1　アプリケーションの実行基盤としての機能

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

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

アプリケーションの実行基盤としての機能について，分類ごとに説明します。

(1)　アプリケーションを動作させるための基本的な機能（基本・開発機能）

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

アプリケーションサーバでは，Java EE 6 に準拠した J2EE サーバを提供しています。J2EE サーバでは，標準仕様に準拠した機能のほか，アプリケーションサーバ独自の機能も提供しています。

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

それぞれの分類の概要を説明します。

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

(2)　Web サービスを開発するための機能

Web サービスの実行環境および開発環境としての機能が該当します。

アプリケーションサーバでは，次のエンジンを提供しています。

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

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

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

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

(4)　システムのセキュリティを確保するための機能（セキュリティ管理機能）

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

1.1.2　アプリケーションの実行基盤を運用・保守するための機能

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

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

(1)　システムの起動・停止など日常的な運用をするための機能（運用機能）

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

(2)　システムの利用状況を監視するための機能（監視機能）

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

(3)　ほかの製品と連携して運用するための機能（連携機能）

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

(4)　トラブル発生時などに対処するための機能（保守機能）

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

(5)　旧バージョンの製品から移行するための機能（移行機能）

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

(6)　旧バージョンの製品との互換のための機能（互換機能）

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

1.1.3　機能とマニュアルの対応

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

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

表 1-1　機能の分類と機能解説のマニュアルの対応

分類	機能	マニュアル※1
基本・開発機能	Web コンテナ	基本・開発編(Web コンテナ)※2
	JSF および JSTL の利用	
	Web サーバ連携	
	インプロセス HTTP サーバ	
	サーブレットおよび JSP の実装	
	EJB コンテナ	基本・開発編(EJB コンテナ)
	EJB クライアント	
	Enterprise Bean 実装時の注意事項	
	ネーミング管理	基本・開発編(コンテナ共通機能)
	リソース接続とトランザクション管理	
	OpenTP1 からのアプリケーションサーバの呼び出し（TP1 インバウンド連携機能）	
	アプリケーションサーバでの JPA の利用	
	CJPA プロバイダ	
	CJMS プロバイダ	
	JavaMail の利用	
	アプリケーションサーバでの CDI の利用	
	アプリケーションサーバでの Bean Validation の利用	
	アプリケーションの属性管理	
	アノテーションの使用	
	J2EE アプリケーションの形式とデプロイ	
	コンテナ拡張ライブラリ	
拡張機能	バッチサーバによるアプリケーションの実行	拡張編
	CTM によるリクエストのスケジューリングと負荷分散	
	バッチアプリケーションのスケジューリング	
	J2EE サーバ間のセッション情報の引き継ぎ（セッションフェイルオーバ機能）	
	データベースセッションフェイルオーバ機能	

分類	機能	マニュアル※1
拡張機能	EADs セッションフェイルオーバ機能	拡張編
	明示管理ヒープ機能を使用した FullGC の抑止	
	アプリケーションのユーザログ出力	
	スレッドの非同期並行処理	
セキュリティ管理機能	統合ユーザ管理による認証	セキュリティ管理機能編
	アプリケーションの設定による認証	
	SSL/TLS 通信での TLSv1.2 の使用	
	API による直接接続を使用する負荷分散機の運用管理機能からの制御	
運用機能	システムの起動と停止	運用／監視／連携編
	J2EE アプリケーションの運用	
監視機能	稼働情報の監視（稼働情報収集機能）	
	リソースの枯渇監視	
	監査ログ出力機能	
	データベース監査証跡連携機能	
	運用管理コマンドによる稼働情報の出力	
	Management イベントの通知と Management アクションによる処理の自動実行	
	CTM の稼働統計情報の収集	
	コンソールログの出力	
連携機能	JP1 と連携したシステムの運用	
	システムの集中監視（JP1/IM との連携）	
	ジョブによるシステムの自動運転（JP1/AJS との連携）	
	監査ログの収集および一元管理（JP1/Audit Management - Manager との連携）	
	クラスタソフトウェアとの連携	
	1:1 系切り替えシステム（クラスタソフトウェアとの連携）	
	相互系切り替えシステム（クラスタソフトウェアとの連携）	
	N:1 リカバリシステム（クラスタソフトウェアとの連携）	
	ホスト単位管理モデルを対象にした系切り替えシステム（クラスタソフトウェアとの連携）	
保守機能	トラブルシューティング関連機能	保守／移行編
	性能解析トレースを使用した性能解析	

分類	機能	マニュアル※1
保守機能	製品の JavaVM（以降，JavaVM と略す場合があります）の機能	保守／移行編
移行機能	旧バージョンのアプリケーションサーバからの移行	
	推奨機能への移行	
互換機能	ベーシックモード	互換編
	サーブレットエンジンモード	
	基本・開発機能の互換機能	
	拡張機能の互換機能	

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

注※2　このマニュアルです。

1.2　システムの目的と機能の対応

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

この節では，Web アプリケーションを実行するためのそれぞれの機能をどのようなシステムの場合に使用するとよいかを示します。機能ごとに，次の項目への対応を示しています。

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

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

1.2.1　Web コンテナの機能

Web コンテナの機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

表 1-2　Web コンテナの機能とシステムの目的の対応

機能	システムの目的					Java EE 標準への準拠		参照先
	信頼性	性能	運用・保守	拡張性	そのほか	標準	拡張	
Web アプリケーションの実行機能	−	−	−	−	−	○	○	2.2
JSP の実行機能	−	−	−	−	−	○	○	2.3
JSP デバッグ機能	−	−	−	−	○	○	○	2.4
JSP 事前コンパイル機能とコンパイル結果の保持	−	○	−	−	−	−	○	2.5
デフォルトの文字エンコーディング設定機能	−	−	○	−	−	○	○	2.6

機能	システムの目的					Java EE 標準への準拠		参照先
	信頼性	性能	運用・保守	拡張性	そのほか	標準	拡張	
セッション管理機能	○	−	−	○	−	○	○	2.7
アプリケーションのイベントリスナ	−	−	−	−	○	○	−	2.8
リクエストおよびレスポンスのフィルタリング	−	−	○	−	−	○	○	2.9
HTTP レスポンス圧縮機能	−	○	−	−	−	−	○	2.10
EJB コンテナとの連携	−	−	−	−	−	○	○	2.11
データベースとの接続	−	−	−	○	−	○	○	2.12
Web コンテナによるスレッドの作成	−	○	−	−	−	−	○	2.13
ユーザスレッドの使用	−	−	−	−	○	−	○	2.14
同時実行スレッド数の制御	−	○	−	−	−	−	○	2.15 2.16 2.17 2.18 2.19
エラーページのカスタマイズ	−	−	−	−	○	−	○	2.20
静的コンテンツのキャッシュ	−	○	−	−	−	−	○	2.21
URI のデコード機能	−	−	○	−	−	○	−	2.22
Web アプリケーションのバージョン設定機能	−	−	○	−	−	○	○	2.23

（凡例）　○：対応する　　−：対応しない

注

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

1.2.2　JSF および JSTL の機能

JSF および JSTL の機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

表 1-3　JSF および JSTL の機能とシステムの目的の対応

機能	システムの目的					Java EE 標準への準拠		参照先
	信頼性	性能	運用・保守	拡張性	そのほか	標準	拡張	
JSF	－	－	－	－	○※	○	－	3 章
JSTL	－	－	－	－	○※	○	－	

（凡例）　○：対応する　　－：対応しない

注※

アプリケーション開発時の開発容易性を向上させる機能です。

1.2.3　Web サーバ連携の機能

Web サーバ連携の機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

表 1-4　Web サーバ連携の機能とシステムの目的の対応

機能	システムの目的					Java EE 標準への準拠		参照先
	信頼性	性能	運用・保守	拡張性	そのほか	標準	拡張	
Web サーバ（リダイレクタ）によるリクエストの振り分け	○	○	－	○	－	－	○	4.2 4.3 4.4 4.5
通信タイムアウト（Web サーバ連携）	○	○	－	－	－	－	○	4.6
IP アドレス指定（Web サーバ連携）	－	－	－	－	○	－	○	4.7
エラーページのカスタマイズ（Web サーバ連携）	－	－	－	－	○	－	○	4.8
ドメイン名指定でのトップページの表示	－	－	－	－	○	○	－	4.9
Web コンテナへのゲートウェイ情報の通知	－	－	－	－	○	－	○	4.10

（凡例）　○：対応する　　－：対応しない

注

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

1.2.4　インプロセス HTTP サーバの機能

インプロセス HTTP サーバの機能を次の表に示します。システムの目的に合った機能を選択してください。機能の詳細については，参照先を確認してください。

表 1-5　インプロセス HTTP サーバの機能とシステムの目的の対応

機能	システムの目的					Java EE 標準への準拠		参照先
	信頼性	性能	運用・保守	拡張性	そのほか	標準	拡張	
Web クライアントからの接続数の制御	−	○	−	−	−	−	○	5.3
リクエスト処理スレッド数の制御	−	○	−	−	−	−	○	5.4
Web クライアントからの同時接続数の制御によるリクエストの流量制御	−	○	−	−	−	−	○	5.5
同時実行スレッド数の制御によるリクエストの流量制御	−	○	−	−	−	−	○	5.6
リダイレクトによるリクエストの振り分け	−	−	−	−	○	−	○	5.7
Persistent Connection による Web クライアントとの通信制御	−	○	−	−	−	−	○	5.8
通信タイムアウト（インプロセス HTTP サーバ）	○	○	−	−	−	−	○	5.9
IP アドレス指定（インプロセス HTTP サーバ）	−	−	−	−	○	−	○	5.10
アクセスを許可するホストの制限によるアクセス制御	○	−	−	−	−	−	○	5.11
リクエストデータのサイズの制限によるアクセス制御	○	−	−	−	−	−	○	5.12
有効な HTTP メソッドの制限によるアクセス制御	○	−	−	−	−	−	○	5.13
HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ	−	−	−	−	○	−	○	5.14
エラーページのカスタマイズ（インプロセス HTTP サーバ）	−	−	−	−	○	−	○	5.15

機能	システムの目的					Java EE 標準への準拠		参照先
	信頼性	性能	運用・保守	拡張性	そのほか	標準	拡張	
Web コンテナへのゲートウェイ情報の通知	－	－	－	－	○	－	○	5.16
ログ・トレースの出力	－	－	－	－	○	－	○	5.17

（凡例）　○：対応する　　－：対応しない

注

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

1.3　このマニュアルに記載している機能の説明

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

1.3.1　分類の意味

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

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

1.3.2　分類を示す表の例

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

次に，機能説明の分類を示す表の例を示します。

機能説明の分類を示す表の例

表 X-1　この章の構成（○○機能）

分類	タイトル	参照先
解説	○○機能とは	X.1
実装	アプリケーションの実装	X.2
	DD および cosminexus.xml※での定義	X.3
設定	実行環境での設定	X.4
運用	○○機能を使用した運用	X.5
注意事項	○○機能使用時の注意事項	X.6

注※

cosminexus.xml については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「11.　アプリケーションの属性管理」を参照してください。

ポイント

cosminexus.xml を含まないアプリケーションのプロパティ設定

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

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

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

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

1.4　アプリケーションサーバ 09-70 での主な機能変更

この節では，アプリケーションサーバ 09-70 での主な機能の変更について，変更目的ごとに説明します。

説明内容は次のとおりです。

- アプリケーションサーバ 09-70 で変更になった主な機能と，その概要を説明しています。機能の詳細については参照先の記述を確認してください。「参照先マニュアル」および「参照個所」には，その機能についての主な記載個所を記載しています。
- 「参照先マニュアル」に示したマニュアル名の「アプリケーションサーバ」は省略しています。

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 1-6　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータルでの JSP コンパイルバージョンの追加	J2EE サーバでの JSP から生成されたサーブレットのコンパイル方法に「JDK1.7 の仕様に従ったコンパイル」と「JDK7 の仕様に従ったコンパイル」を追加する。	運用管理ポータル操作ガイド	10.9.4
		リファレンス 定義編(サーバ定義)	4.14.1
JDK8 でのメタスペース対応	JavaVM の起動で使用している Permanent 領域用のオプションを Metaspace 領域用のオプションに変更する。	システム構築・運用ガイド	付録 A.2
		運用管理ポータル操作ガイド	10.8.3, 10.9.8
		リファレンス 定義編(サーバ定義)	5.2, 5.3, 10.4
統合ユーザ管理でのユーザ認証の SHA-2 対応	統合ユーザ管理でのユーザ認証のハッシュアルゴリズムとして SHA-224, SHA-256, SHA-384, SHA-512 を追加する。	機能解説 セキュリティ管理機能編	5.3.1, 5.3.9, 5.10.6, 11.4.3, 12.4.3, 12.5.3, 13.2, 14.3

(2)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 1-7　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
V9.7 へのバージョンアップ対応	バージョンアップ時の JavaVM の起動で使用している Permanent 領域用のオプションを Metaspace 領域用のオプションに変更する手順を追加する。	機能解説 保守／移行編	10.4.2, 10.4.3, 10.4.4, 10.4.5

(3)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 1-8　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
snapshot ログの収集対象	snapshot ログの収集対象として JavaVM イベントログと Management Server のスレッドダンプを追加する。	機能解説 保守／移行編	付録 A.2

2 Web コンテナ

この章では，サーブレットと JSP を実行するためのサーバ基盤である，Web コンテナの機能について説明します。Web コンテナの機能は，サーブレットまたは JSP を使用した J2EE アプリケーションを実行する場合に使用します。

2.1　この章の構成

アプリケーションサーバでは，Web アプリケーションの実行機能を提供するコンテナとしての機能（**Web コンテナ**）を提供しています。

アプリケーションサーバで提供する Web コンテナの機能と参照先を次の表に示します。

表 2-1　Web コンテナの機能と参照先

機能	参照先
Web アプリケーションの実行機能	2.2
JSP の実行機能	2.3
JSP デバッグ機能	2.4
JSP 事前コンパイル機能とコンパイル結果の保持	2.5
デフォルトの文字エンコーディング設定機能	2.6
セッション管理機能	2.7
アプリケーションのイベントリスナ	2.8
リクエストおよびレスポンスのフィルタリング機能	2.9
HTTP レスポンス圧縮機能	2.10
EJB コンテナとの連携	2.11
データベースとの接続	2.12
Web コンテナによるスレッドの作成	2.13
ユーザスレッドの使用	2.14
同時実行スレッド数の制御	2.15
Web コンテナ単位での同時実行スレッド数の制御	2.16
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	2.17
URL グループ単位での同時実行スレッド数の制御	2.18
同時実行スレッド数の動的変更	2.19
エラーページのカスタマイズ	2.20
静的コンテンツのキャッシュ	2.21
URI のデコード機能	2.22
Web アプリケーションのバージョン設定機能	2.23
POST データ最大サイズについての注意事項	2.24

アプリケーションサーバで提供する Web コンテナの機能には，Java EE で規定された機能にアプリケーションサーバ独自の機能を拡張したものと，アプリケーションサーバ独自の機能として提供しているものがあります。アプリケーションサーバ独自の機能かどうかについては，「1.2　システムの目的と機能の対応」を参照してください。

2.2　Web アプリケーションの実行機能

Web コンテナでは，Web アプリケーションを実行できます。Web アプリケーションとは，Web コンテナ上で動作するサーバプログラムのことです。この節では，Web アプリケーションの実行機能について説明します。

この節の構成を次の表に示します。

表 2-2　この節の構成（Web アプリケーションの実行機能）

分類	タイトル	参照先
解説	Web アプリケーションのデプロイとアンデプロイ	2.2.1
注意事項	Web アプリケーションのデプロイとアンデプロイの注意事項	2.2.2

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

通常の Web サーバは，固定された HTML ファイルだけを送信します。Web コンテナが機能することによって，Web コンテナ上で Web アプリケーションを実行し，Web クライアントから受け取ったデータを処理したり，その処理の結果に応じて異なる Web ページを生成したりできるようになります。

Web アプリケーションは，主に，サーブレットおよび JSP と呼ばれる 2 種類の技術を使用して開発されたアプリケーションです。**サーブレット**は Java プログラムを使い，HTML を生成したり，Web クライアントから受け取った情報を処理したりする技術です。**JSP**（JavaServer Pages）はサーブレット技術を基盤に，HTML ページの中にタグや Java プログラムを埋め込むことで，動的に Web 画面を生成する技術です。JSP は JSP コンパイラによって，一度 Java で記述されたサーブレットプログラムに変換され，そのあと，Java コンパイラによってコンパイル，実行されます。

アプリケーションサーバの Web コンテナでは，Java Servlet 3.0 仕様，および JavaServer Pages（JSP）2.1 仕様に準拠した Web アプリケーションを実行できます。また，JSP 2.2 仕様については，新規に追加されたタグは無視されますが，JSP 2.2 仕様の web.xml を読み込むことができます。Web アプリケーション実行機能の詳細については，Java Servlet Specification v3.0，および JavaServer Pages Specification v2.2 を参照してください。

なお，Web コンテナを使用して Web アプリケーションを実行するには，Web サーバとして HTTP Server または Microsoft IIS を使用するか，インプロセス HTTP サーバを使用する必要があります。

参考

以前のバージョンのアプリケーションサーバで実行できるアプリケーションは，本バージョンでも実行できます。

2.2.1　Web アプリケーションのデプロイとアンデプロイ

WAR 形式の Web アプリケーションは，次のどちらかの方法でデプロイすることによって，実行可能な状態になります。

- Web アプリケーションを EAR 形式にパッケージ化してアーカイブ形式の J2EE アプリケーション，または展開ディレクトリ形式の J2EE アプリケーションとしてインポートします。
- WAR ファイルまたは WAR ディレクトリを指定して，Web アプリケーションを単体でインポートします。

単体でインポートした Web アプリケーションを WAR アプリケーションといいます。WAR アプリケーションについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.9　WAR アプリケーション」を参照してください。

実行できる J2EE アプリケーションの形式については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.2　実行できる J2EE アプリケーションの形式」を参照してください。

複数の Web アプリケーションのデプロイを実行した場合，Web アプリケーションごとに独立のクラスローダが作成されます。このため，異なる Web アプリケーションで同一のクラス名のクラス（Login サーブレットなど）を用いた場合でも，別々のクラスローダ上で独立に扱われます。

なお，J2EE アプリケーションとしてデプロイした Web アプリケーションをアンデプロイする場合は，J2EE アプリケーションごとアンデプロイします。WAR 単位でアンデプロイすることはできません。

2.2.2　Web アプリケーションのデプロイとアンデプロイの注意事項

ここでは，Web アプリケーションをデプロイ，またはアンデプロイする場合の留意事項について説明します。

(1)　サーブレット/JSP のデフォルトマッピングについて

クライアントがリクエストする URL に対してどのサーブレットが呼び出されるかの設定を**サーブレットマッピング**といいます。Servlet 仕様では，サーブレットマッピングを DD（WEB-INF/web.xml）内に記述するように求めています。

これに対して，Web コンテナでデフォルトで定義されているマッピングを**デフォルトマッピング**といいます。Web コンテナでは，次に示すマッピングをデフォルトで定義しています。

表 2-3　Web コンテナで定義されているサーブレット/JSP のデフォルトマッピング

URL	取り扱い
*.jsp	JSP ファイルとして取り扱われます。
*.jspx	Servlet 2.4 以降の仕様に準拠した Web アプリケーションについては，JSP ドキュメントとして取り扱われます。なお，Servlet 2.2 および Servlet 2.3 仕様に準拠した Web アプリケーションの場合は静的コンテンツとして扱われます。
/servlet/*	WEB-INF/classes 以下，または WEB-INF/lib 以下に配置された JAR ファイルに含まれているサーブレットのクラスが実行されます。実行するサーブレットは，サーブレット名から検索されます。 URL に指定された「*」の部分がサーブレット名として定義されていない場合，サーブレットクラスが検索されます。URL の「*」の部分には，サーブレットの完全修飾クラス名，または web.xml で定義したサーブレット名を指定できます。 サーブレットの完全修飾クラス名を指定した場合，指定したサーブレットが実行されます。サーブレット名を指定した場合，web.xml で定義したサーブレットが実行されます。

なお，サーブレットは web.xml にマッピング定義が必要なため，web.xml が省略された Web アプリケーションの場合，サーブレットのデフォルトマッピングは有効になりません。

サーブレットのデフォルトマッピングについては，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で，次のパラメタに有効または無効を指定します。デフォルトの設定では，無効になっています。

```
webserver.container.servlet.default_mapping.enabled
```

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　DD（WEB-INF/web.xml）内のタグが設定されていない場合のデフォルト値について

Web コンテナでは，次に示すタグが DD（WEB-INF/web.xml）に設定されていない場合，次に示すデフォルト値を使用します。

表 2-4　DD（WEB-INF/web.xml）内のタグが設定されていない場合のデフォルト値

タグ名	設定されていない場合のデフォルト値
welcome-file-list	<welcome-file-list> 　<welcome-file> 　　index.jsp 　</welcome-file> 　<welcome-file> 　　index.html 　</welcome-file> 　<welcome-file> 　　index.htm 　</welcome-file> </welcome-file-list>
session-timeout	30
mime-mapping	拡張子と MIME タイプとの対応づけ。

DD（WEB-INF/web.xml）にこれらのタグを設定している場合には，次の設定となります。

- <welcome-file-list>タグに設定された値で，デフォルト値を上書きします。
- <session-timeout>タグに設定された値で，デフォルト値を上書きします。
- <mime-mapping>タグに定義した拡張子単位の設定で，デフォルト値を拡張子単位に上書きします。

なお，DD（WEB-INF/web.xml）の mime-mapping タグにデフォルトで設定されている拡張子と MIME タイプの対応づけについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「付録 B.1　拡張子と MIME タイプの対応づけ」を参照してください。

(3)　サーブレットのインスタンス作成数

Web アプリケーション単位に同じクラスのサーブレットのインスタンスを一つ作成します。
SingleThreadModel を継承したサーブレットのインスタンスも，一般のサーブレットと同様に Web アプリケーション単位に一つ作成します。

ただし，同一のサーブレットおよび JSP に対して，デフォルトのマッピングと DD に記述したマッピングの両方でアクセスすると，インスタンスは次のように生成されます。

表 2-5　デフォルトマッピングと DD に記述したマッピングの両方でアクセスする場合のインスタンスの生成

URL	インスタンス
*.jsp	DD に記述したマッピングでアクセスした場合とは別のインスタンスが生成されます。
*.jspx	
/servlet/*	• サーブレットの完全修飾クラス名を指定した場合 DD に記述したマッピングでアクセスした場合とは別のインスタンスが生成されます。 • web.xml で定義したサーブレット名を指定した場合

URL	インスタンス
/servlet/*	DD に記述したマッピングでアクセスした場合と同じインスタンスが生成されます。

また，SingleThreadModel を継承した単一のサーブレットに対して，Web アプリケーション単位にリクエストが並列に複数到着した場合，各スレッドがオーバラップしないで 1 スレッドずつ順に実行するように制御します。

(4)　サーブレットの init メソッドおよび service メソッドの実行のタイミング

サーブレットの init メソッドおよび service メソッドの実行のタイミングは，デフォルトマッピングかサーブレットマッピングかによって異なります。

- デフォルトマッピングでサーブレットにアクセスした場合

  サーブレットの init メソッドおよび service メソッドは，該当する URL にマッピングされているフィルタの処理の延長で，実行されます。
- サーブレットマッピングでサーブレットにアクセスした場合

  init メソッドはフィルタの処理の前に，実行されます。service メソッドは，該当する URL にマッピングされたフィルタの処理の延長で，実行されます。

(5)　レスポンス送信時に使用されるサーブレットのバッファ

レスポンス送信時に使用されるサーブレットのバッファは，リクエスト処理スレッドの数だけ保持されます。javax.servlet.ServletResponse の setBufferSize メソッドを使用してバッファサイズを変更する場合は，バッファサイズ×リクエスト処理スレッド数分のメモリが確保されることを考慮した上で，メモリ使用量を見積もってください。

(6)　クエリ文字列の解析について

クエリ文字列は，URL の?マーク以降に，"名前=値"の組が一つ以上組み合わさって構成されます。

アプリケーションサーバでは，"名前=値"の部分に複数の"="がある場合，最初の"="より前の文字列が名前，後ろの文字列が値となります。例えば，URL が「http://host/examples?a=b=c」の場合は，「名前"a"の値は"b=c"」と解析されます。

2.3　JSP の実行機能

この節では，JSP の実行機能について説明します。

Web コンテナでは，Servlet 仕様が規定した JSP の文法に従って作成された JSP を，サーブレットに変換し，java のプログラムとしてコンパイルして，Java VM 内にロードして実行できます。

この節の構成を次の表に示します。

表 2-6　この節の構成（JSP の実行機能）

分類	タイトル	参照先
解説	JSP の実行機能の概要	2.3.1
	タグファイルの実行	2.3.2
	JSP EL の実行	2.3.3
	タグライブラリのライブラリ JAR への格納	2.3.4
	カスタムタグの属性名チェック	2.3.5
	<jsp:useBean>タグの id 属性重複チェック	2.3.6
	page/tag ディレクティブの import 属性暗黙インポート	2.3.7

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

2.3.1　JSP の実行機能の概要

JSP は，JSP 仕様の標準の書式である標準シンタックスと，XML 仕様の書式である XML シンタックスで記述できます。標準シンタックスで記述された JSP を JSP ページと呼び，XML シンタックスで記述された JSP を JSP ドキュメントと呼びます。また，JSP ページ，JSP ドキュメントを総称して，JSP ファイルと呼びます。

(1)　JSP の構成

JSP は要素（ディレクティブ，アクション，スクリプティング要素）とテンプレートテキストから構成されます。テンプレートテキストには，**ホワイトスペース**が含まれます。通常，テンプレートテキストに含まれるホワイトスペースはそのまま保持されますが，JSP 2.1 仕様から，JSP ページまたは標準シンタックスのタグファイルのテンプレートテキストに含まれる不要なホワイトスペースを削除する機能が追加されました。詳細は，「6.2.7(2)　不要なホワイトスペースの削除機能」を参照してください。

参考

JSP 2.0 仕様から，JSP EL が規定されました。JSP EL とは，アクションやテンプレートテキスト内に直接記述できる簡易言語です。詳細は「2.3.3　JSP EL の実行」を参照してください。

(2)　トランスレーションエラー

トランスレーションエラーとは，JSP コンパイル処理の JSP ファイルから java ファイルへの変換（**JSP トランスレーション**）時に，構文誤りなどが原因で java ファイルに変換できない場合に発生するエラーを指します。

JSP トランスレーションは次のタイミングで実行されますが，その際にトランスレーションエラーが発生するおそれがあります。

- JSP へのリクエスト受信時
- アプリケーションのリロード時
- JSP 事前コンパイル機能を使用する場合のアプリケーション開始時
- JSP 事前コンパイル機能を使用する場合のコマンド実行時

J2EE サーバ上での JSP コンパイル時にトランスレーションエラーが発生した場合，メッセージ KDJE39145-E がサーブレットログに，メッセージ KDJE39186-E がメッセージログにそれぞれ出力されます。リクエスト処理時にトランスレーションエラーとなった場合，リダイレクタがエラーステータスコード 500 を返します。

cjjspc コマンドでの JSP 事前コンパイル時にトランスレーションエラーが発生した場合，メッセージ KDJE39145-E，およびメッセージ KDJE39186-E がコンソールに出力されます。

なお，TLD ファイルの解析，タグライブラリバリデータによる JSP の検証や，TagExtraInfo クラスで指定したスクリプト変数の重複など，ほかの原因でトランスレーションエラーが発生することもあります。この場合，原因に応じたメッセージが出力されます。トランスレーションエラー発生時に出力されるメッセージを，原因ごとに次の表に示します。

表 2–7　JSP コンパイル以外の動作が原因でトランスレーションエラーが発生した場合に出力されるメッセージ

原因の分類	出力されるメッセージ
TLD ファイルの解析	KDJE39214-E, KDJE39216-E, KDJE39193-E, KDJE39055-E, KDJE39205-E, KDJE39206-E, KDJE39207-E, KDJE39208-E, KDJE39296-E, KDJE39301-E, KDJE39302-E, KDJE39303-E, KDJE39305-E, KDJE39306-E, KDJE39307-E, KDJE39308-E
タグライブラリバリデータによる JSP の検証	KDJE39104-E, KDJE39105-E, KDJE39106-E, KDJE39107-E, KDJE39108-E, KDJE39115-E, KDJE39116-E, KDJE39117-E, KDJE39134-E, KDJE39135-E
TagExtraInfo クラスで指定したスクリプト変数の重複	KDJE39131-E, KDJE39132-E, KDJE39133-E, KDJE39136-E, KDJE39282-E, KDJE39283-E, KDJE39291-E, KDJE39294-E

2.3.2　タグファイルの実行

アプリケーションサーバでは，JSP 2.0 以降で規定されている JSP の構文に従って作成されたタグファイルを実行できます。タグファイルの実行では，次の内容が実施されます。

- タグファイルを java クラスに変換する
- 変換した java クラスファイルをコンパイルする
- コンパイルしたファイルを JavaVM 内にロードして実行する

タグファイルを使用すると，従来，カスタムタグライブラリで実現していたタグの拡張を JSP の構文だけで記述できます。

2.3.3　JSP EL の実行

アプリケーションサーバでは，JSP 2.0 以降で規定されている EL の構文に従って作成された EL 式を実行できます。JSP EL を使用すると，JSP ファイルやタグファイル内で JavaBeans 属性へのアクセスなどを記述できます。

JSP 2.1 仕様で追加された EL の機能では，Java SE の Enum 型への対応，JavaBeans の属性への値の設定，メソッドを示す EL が記述できます。また，JSP 2.1 仕様では，JSP 2.0 仕様が規定した EL に加え，JSF1.1 仕様で規定されている EL が JSP 2.1 仕様の EL として記述できます。

JSP 2.2 仕様で追加された EL の機能では，引数付きのメソッドが指定できます。

JSP 2.0 仕様で規定された EL を特に指す場合，JSP 2.0 仕様の EL と呼びます。JSP 2.1 仕様の EL を特に指す場合，EL2.1 と呼びます。JSP 2.2 仕様の EL を特に指す場合，EL2.2 と呼びます。

2.3.4　タグライブラリの J2EE アプリケーションへの格納

アプリケーションサーバでは，タグライブラリを J2EE アプリケーションに格納し，JSP から使用できます。

J2EE アプリケーションに格納されたタグライブラリを JSP から使用する方法は，JSP を J2EE サーバでコンパイルするか cjjspc コマンドでコンパイルするかによって異なります。

J2EE サーバで JSP をコンパイルする場合

タグライブラリをライブラリ JAR に格納する。

cjjspc コマンドで JSP をコンパイルする場合

タグライブラリをクラスパスに指定した JAR ファイルに格納する。

それぞれの方法について説明します。

(1)　J2EE サーバで JSP をコンパイルする場合

タグライブラリをライブラリ JAR に格納することで，JSP から使用できます。

(a)　TLD ファイルの検索

タグライブラリをライブラリ JAR として J2EE アプリケーションに格納した場合，web.xml の<taglib>要素で TLD ファイルを指定できません。この場合，JSP ファイルの taglib ディレクティブの uri 属性で，TLD ファイルの<uri>要素に指定された URI を指定してください。Web アプリケーション開始時に，ライブラリ JAR に格納された TLD ファイルが Web コンテナによって検索され，TLD ファイルの<uri>要素に記述された URI とその TLD ファイル自身がマッピングされます。

なお，<uri>要素がない場合は URI とその TLD ファイル自身はマッピングされません。

ライブラリ JAR 内の TLD ファイルのマッピングは最も優先順位が低いマッピングです。このため，URI と TLD ファイルのマッピングで，ライブラリ JAR 内の TLD ファイルの URI がほかの TLD ファイルや web.xml と重複しても，07-60 以前のバージョンで動作していた J2EE アプリケーションの動作に影響はありません。ライブラリ JAR 内の TLD ファイルのマッピングの優先順位については，「6.2.6(7)　taglib ディレクティブの uri 属性で指定する URI と TLD ファイルのマッピングについて」を参照してください。

JSP ファイルでは，taglib ディレクティブの uri 属性で URI を指定することでタグライブラリを使用できます。

(b) ライブラリ JAR に格納する場合に使用できるタグライブラリの機能

タグライブラリをライブラリ JAR に格納する場合に使用できるタグライブラリの機能は，カスタムタグおよびリスナです。

タグファイルは使用できません。JSP ファイルまたはタグファイルで，ライブラリ JAR に格納されたタグファイルを使用した場合，ライブラリ JAR に格納された TLD ファイルの<tag-file>要素が無視されます。そのため，タグファイルが存在しないと見なされて，JSP トランスレーション時にトランスレーションエラーとなります。

(2) cjjspc コマンドで JSP をコンパイルする場合

JAR ファイル内のタグライブラリを cjjspc コマンドの-classpath オプションでクラスパスに指定することで，Web アプリケーションの WEB-INF/lib ディレクトリ以外に配置したタグライブラリを JSP から使用できます。

(a) TLD ファイルの検索

クラスパスに指定した JAR ファイルに含まれる TLD ファイルが自動的に検出され，TLD ファイルの<uri>要素で指定した URI とその TLD ファイル自身がマッピングされます。

クラスパスに指定した JAR ファイル内の TLD ファイルのマッピングは最も優先順位が低いマッピングです。このため，URI と TLD ファイルのマッピングで，クラスパスに指定した JAR ファイル内の TLD ファイルの URI がほかの TLD ファイルや web.xml と重複しても，07-60 以前のバージョンで動作していた J2EE アプリケーションの動作に影響はありません。クラスパスに指定した JAR ファイル内の TLD ファイルのマッピングの優先順位については，「6.2.6(7)　taglib ディレクティブの uri 属性で指定する URI と TLD ファイルのマッピングについて」を参照してください。

(b) クラスパスに指定した JAR ファイルに格納する場合に使用できるタグライブラリの機能

タグライブラリをクラスパスに指定した JAR ファイルに格納する場合に使用できるタグライブラリの機能は，カスタムタグおよびリスナです。

タグファイルは使用できません。JSP ファイルまたはタグファイルで，クラスパスに指定した JAR ファイルに格納されたタグファイルを使用した場合，JAR ファイルに格納された TLD ファイルの<tag-file>要素が無視されます。そのため，タグファイルが存在しないと見なされて，JSP トランスレーション時にトランスレーションエラーとなります。

2.3.5 カスタムタグの属性名チェック

カスタムタグの属性名チェック時に，次の属性名で大文字と小文字が一致しない場合，トランスレーションエラーが発生します。

- JSP のカスタムタグで指定した属性名
- TLD ファイルまたはタグファイルで定義した属性名

アプリケーションサーバでは，カスタムタグの属性名チェック時に，大文字と小文字の不一致によるトランスレーションエラーの発生を抑止できます。大文字と小文字の不一致によるトランスレーションエラーの発生を抑止すると，TLD ファイルまたはタグファイルに定義されている属性名から，大文字と小文字を区別しないでカスタムタグの属性定義が検索されます。

TLD ファイルおよびタグファイルの属性名を定義する個所は次のとおりです。

- TLD ファイルの<taglib><tag><attribute>要素内の<name>要素で定義した属性名

- タグファイルの attribute ディレクティブで定義した属性名

(1)　カスタムタグの属性名チェックを無効にする方法

カスタムタグの属性名チェック時に大文字と小文字の不一致によるトランスレーションエラーの発生を抑止するには，次の二つの方法があります。

- 簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内でパラメタ webserver.jsp.translation.customAction.ignoreCaseAttributeName に true を指定します。
  簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。
- cjjspc コマンドに-usebeannocheckduplicateid オプションを指定してコンパイルします。
  コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc (JSP の事前コンパイル)」を参照してください。

(2)　注意事項

大文字と小文字だけで区別されている属性がある場合に，大文字と小文字の不一致によるトランスレーションエラーの発生を抑止しないでください。抑止すると，次の問題が発生します。

- 大文字と小文字だけで区別されている複数の属性を JSP のカスタムタグで指定している場合
  それぞれの属性に対してタグハンドラの setter メソッドが実行されます。このとき，同一の属性が複数個指定されていても，トランスレーションエラーは発生しません。このため，複数個の同一の属性に対して setter メソッドが実行されてしまうおそれがあります。
- 大文字と小文字だけで区別されている属性名に対応したタグハンドラを実装している場合
  大文字と小文字の区別なく，TLD ファイルまたはタグファイルで先に記述された属性の setter メソッドが呼び出されます。このとき，トランスレーションエラーは発生しません。このため，意図した setter メソッドを呼び出すことができなくなるおそれがあります。

2.3.6　<jsp:useBean>タグの id 属性重複チェック

JSP 仕様では，<jsp:useBean>タグの id 属性値が重複していた場合，トランスレーションエラーが発生して JSP のコンパイルに失敗します。

アプリケーションサーバでは，<jsp:useBean>タグの id 属性値の重複によるトランスレーションエラーの発生を抑止して，JSP をコンパイルできます。

<jsp:useBean>タグの id 属性値の重複によるトランスレーションエラーの発生を抑止した場合，次の JSP ファイルの記述例のように<jsp:useBean>タグの id 属性値が重複していても，JSP が問題なく動作します。

JSP ファイルの記述例

```
（省略）
：
<% if（条件式）{ %>
<jsp:useBean id=”BeanTest” class=”test.TestClass1” />
<% } else { %>
<jsp:useBean id=”BeanTest” class=”test.TestClass2” />
<% } %>
```

```
:
（省略）
```

スクリプトレットを使用して，if 節と else 節にそれぞれ<jsp:useBean>タグを記述することで，片方の<jsp:useBean>タグしか実行されません。そのため，<jsp:useBean>タグの id 属性値が重複していても JSP は問題なく動作します。

<jsp:useBean>タグの id 属性値の重複によるトランスレーションエラーの発生を抑止している場合に，<jsp:useBean>タグの id 属性値が重複したときは，KDJE39544-I のメッセージが出力されます。

(1) <jsp:useBean>タグの id 属性重複チェックを有効にする方法

<jsp:useBean>タグの id 属性値の重複によるトランスレーションエラーの発生を抑止するには，次の二つの方法があります。

- 簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内でパラメタ webserver.jsp.translation.useBean.noCheckDuplicateId に true を指定します。
  簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。
- cjjspc コマンドに-usebeannocheckduplicateid オプションを指定してコンパイルします。
  コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc (JSP の事前コンパイル)」を参照してください。

(2) 注意事項

<jsp:useBean>タグで作成された JavaBeans オブジェクトは，id 属性値をキーにして，scope 属性で指定されたスコープ内で管理されます。そのため，次の条件を満たす場合に，意図しない JavaBeans オブジェクトを取得して不正な動作となるおそれがあります。

- <jsp:useBean>タグの id 属性値の重複によるトランスレーションエラーの発生を抑止している。
- <jsp:useBean>タグの id 属性値が重複している。
- id 属性値が重複している二つ以上の<jsp:useBean>タグの class 属性に，異なる JavaBeans クラスを指定している。

<jsp:useBean>タグの id 属性値の重複によるトランスレーションエラーの発生を抑止している場合に，意図したオブジェクトが取得できないおそれがある JSP ファイルの記述例について説明します。

(a) 単一のリクエストで意図しない JavaBeans オブジェクトが取得される場合の JSP ファイルの記述例（二つ以上の異なる条件式で同じ id 属性を指定した例）

```
（省略）
:
<% if (条件式1){ %>
<jsp:useBean id="BeanTest" class="test.TestClass1" scope="page"/>
<% } %>
<% if(条件式2) { %>
<jsp:useBean id="BeanTest" class="test.TestClass2" type="test.TestIF" scope="page"/>
<% } %>
:
（省略）
```

条件式 1 と条件式 2 で，id 属性値に同じ値（BeanTest）が指定されています。条件式 1 および条件式 2 の両方の条件を満たす場合，一つ目の<jsp:useBean>タグでも二つ目の<jsp:useBean>タグでも test.TestClass1 クラスのオブジェクトが取得され，test.TestClass2 クラスのオブジェクトは取得されません。

一つ目の<jsp:useBean>タグで id 属性値「BeanTest」に対して test.TestClass1 クラスのオブジェクトが登録されます。二つ目の<jsp:useBean>タグでも同じ id 属性値「BeanTest」に対しての処理であると解釈されるため，一つ目の<jsp:useBean>タグで登録済みの test.TestClass1 クラスのオブジェクトが取得されます。

(b)　複数のリクエストで意図しない JavaBeans オブジェクトが取得される場合の JSP ファイルの記述例 1（if 文と else 文で同じ id 属性値を指定した例）

```
（省略）
：
<% if（条件式）{ %>
<jsp:useBean id="BeanTest" class="test.TestClass1" scope="session"/>
<% } else { %>
<jsp:useBean id="BeanTest" class="test.TestClass2" scope="session"/>
<% } %>
：
（省略）
```

if 文と else 文で，id 属性値に同じ値（BeanTest）が指定されています。リクエストを 2 回受け付けて，1 回目のリクエストでは if 文の条件式が成立し，2 回目のリクエストでは if 文の条件式が不成立となった場合，一つ目の<jsp:useBean>タグと二つ目の<jsp:useBean>タグの両方で test.TestClass1 クラスのオブジェクトが取得され，test.TestClass2 クラスのオブジェクトは取得されません。

一つ目の<jsp:useBean>タグで id 属性値「BeanTest」に対して test.TestClass1 クラスのオブジェクトが登録されます。二つ目の<jsp:useBean>タグでも同じ id 属性値「BeanTest」に対しての処理であると解釈されるため，一つ目の<jsp:useBean>タグで登録済みの test.TestClass1 クラスのオブジェクトが取得されます。

(c)　複数のリクエストで意図しない JavaBeans オブジェクトが取得される場合の JSP ファイルの記述例 2（<jsp:getProperty>タグまたは<jsp:setProperty>タグで，二つ以上の<jsp:useBean>タグで共通して指定されている id 属性値を使用して JavaBeans オブジェクトを呼び出す例 1）

```
（省略）
：
<% if（条件式1）{ %>
<jsp:useBean id=" BeanTest"  class=" test.TestClass1"  scope=" page" />
<% } %>
<% if(条件式2) { %>
<jsp:useBean id=" BeanTest"  class=" test.TestClass2"  type="  test.TestIF"  scope=" page" />
<% } %>
：
<jsp:setProperty name=" BeanTest"  property=" *" />
：
<jsp:getProperty name=" BeanTest"  property=" value" />
：
```

```
（省略）
```

<jsp:getProperty>および<jsp:setProperty>タグで定義する処理では，<jsp:useBean>タグで作成された JavaBeans オブジェクトを呼び出します。

異なる JavaBeans クラスを作成する二つ以上の<jsp:useBean>タグで id 属性値を重複させると，最後に出現する<jsp:useBean>タグで指定されたオブジェクトが<jsp:getProperty>または<jsp:setProperty>タグでの処理に使用されます。

例では，条件式 1 と条件式 2 で，id 属性値に同じ値（BeanTest）が指定されています。そのため，条件式 1 が成立する場合でも，一つ目の<jsp:useBean>タグで登録された test.TestClass1 クラスのオブジェクトは<jsp:getProperty>および<jsp:setProperty>タグでの処理に使用されません。

(d) 複数のリクエストで意図しない JavaBeans オブジェクトが取得される場合の JSP ファイルの記述例 3（<jsp:getProperty>タグまたは<jsp:setProperty>タグで，二つ以上の<jsp:useBean>タグで共通して指定されている id 属性値を使用して JavaBeans オブジェクトを呼び出す例 2）

```
（省略）
：
<% if（条件式1）{ %>
<jsp:useBean id=”BeanTest” class=”test.TestClass1” scope=”page”/>
：
<jsp:setProperty name=”BeanTest” property=”*”/>
：
<jsp:getProperty name=”BeanTest” property=”value”/>
：
<% } %>
<% if(条件式2) { %>
<jsp:useBean id=”BeanTest” class=”test.TestClass2” type=” test.TestIF” scope=”page”/>
：
<jsp:setProperty name=”BeanTest” property=”*”/>
：
<jsp:getProperty name=”BeanTest” property=”value”/>
：
<% } %>
：
（省略）
```

<jsp:getProperty>および<jsp:setProperty>タグで定義する処理では，<jsp:useBean>タグで作成された JavaBeans オブジェクトを呼び出します。

異なる JavaBeans クラスを作成する二つ以上の<jsp:useBean>タグで id 属性値を重複させると，最後に出現する<jsp:useBean>タグで指定されたオブジェクトが<jsp:getProperty>または<jsp:setProperty>タグでの処理に使用されます。

例では，条件式 1 と条件式 2 で，id 属性値に同じ値（BeanTest）が指定されています。そのため，条件式 1 が成立する場合でも，一つ目の<jsp:setProperty>タグでの処理に使用されるのは，二つ目の<jsp:useBean>タグで登録された test.TestClass2 クラスのオブジェクトです。一つ目の<jsp:useBean>タグで登録された test.TestClass1 クラスのオブジェクトは<jsp:getProperty>および<jsp:setProperty>タグでの処理に使用されません。

2.3.7　page/tag ディレクティブの import 属性暗黙インポート

JSP 仕様では，JSP をコンパイルする際に次のクラスを暗黙にインポートします。

- java.lang.*
- javax.servlet.*
- javax.servlet.jsp.*
- javax.servlet.http.*

page/tag ディレクティブの import 属性暗黙インポート機能を使用すると，上記のクラス以外の任意のクラスを暗黙にインポートできます。

(1)　暗黙インポートするクラスの指定方法

ここでは，page/tag ディレクティブの import 属性暗黙インポート機能の指定方法について説明します。page/tag ディレクティブの import 属性暗黙インポート機能は，J2EE サーバで JSP をコンパイルする場合，または cjjspc コマンドで JSP をコンパイルする場合に使用できます。

暗黙にインポートしたいクラスは，完全修飾名のクラス名，または「パッケージ名.*」で指定します。複数のクラス名を指定する場合は，クラス名とクラス名の間をコンマ（,）で区切って指定してください。存在しないクラス名や，クラスパスに誤りがあるクラス名などを指定した場合は，JSP コンパイル時に KDJE39143-E のメッセージが出力されます。

- J2EE サーバで JSP をコンパイルする場合
  簡易構築定義ファイルの webserver.jsp.additional.import.list キーに暗黙にインポートしたいクラスを指定します。
  簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。
- cjjspc コマンドで JSP 事前コンパイルする場合
  cjjspc コマンドの-addimport オプションで暗黙にインポートしたいクラスを指定します。
  コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc (JSP の事前コンパイル)」を参照してください。

(2)　インポート文の出力順序

JSP コンパイル時に，java ファイルに出力されるインポート文の順番を次に示します。

1. JSP 仕様で規定されているクラスのインポート文
   - java.lang.*
   - javax.servlet.*
   - javax.servlet.jsp.*
   - javax.servlet.http.*
2. page/tag ディレクティブの import 属性に指定されたクラスのインポート文
3. page/tag ディレクティブの import 属性暗黙インポート機能で指定したクラスのインポート文

(3) 注意事項

(a) cjjspc コマンドで JSP 事前コンパイルした JSP ファイル

page/tag ディレクティブの import 属性暗黙インポート機能は，JSP コンパイル時に動作します。cjjspc コマンドで JSP 事前コンパイルした場合，JSP 事前コンパイルした Web アプリケーションに対して，簡易構築定義ファイルの webserver.jsp.additional.import.list キーに指定したクラスは暗黙にインポートされません。そのため，cjjspc コマンドで JSP 事前コンパイルする場合に，page/tag ディレクティブの import 属性暗黙インポート機能を使用するときは，cjjspc コマンドに-addimport オプションを指定して Web アプリケーションを JSP 事前コンパイルしてください。

(b) J2EE サーバ内に複数の Web アプリケーションが存在する場合

簡易構築定義ファイルの webserver.jsp.additional.import.list キーに指定したクラスは，cjjspc コマンドで JSP 事前コンパイルされていない J2EE サーバ内のすべての Web アプリケーションに対して有効になります。Web アプリケーションごとに異なるクラスを指定したい場合は，cjjspc コマンドに-addimport オプションを指定して Web アプリケーションを JSP 事前コンパイルしてください。

(c) JSP 事前コンパイル後の再コンパイル

次の条件をすべて満たす場合は，再コンパイル時に KDJE39143-E のメッセージが出力されます。そのため，cjjspc コマンドに-addimport オプションを指定して JSP 事前コンパイルした Web アプリケーションを使用する場合は，簡易構築定義ファイルの webserver.jsp.additional.import.list キーに cjjspc コマンドの-addimport オプションに指定したクラス名と同じクラス名を指定してください。

- JSP ファイル内で完全修飾名のクラス名でクラスを定義していない。
- cjjspc コマンドの-addimport オプションに暗黙にインポートするクラス名を指定して JSP 事前コンパイルする。
- 簡易構築定義ファイルの webserver.jsp.additional.import.list キーに，cjjspc コマンドの-addimport オプションに指定した暗黙にインポートするクラスと異なるクラス名を指定する。または，webserver.jsp.additional.import.list キーを省略するか，webserver.jsp.additional.import.list キーに空文字を指定する。
- J2EE サーバ上で JSP 事前コンパイルした Web アプリケーションが再コンパイルされる。

(d) 同名のクラスが存在するクラス名を指定した場合

page/tag ディレクティブの import 属性暗黙インポート機能で指定したクラス名と，JSP 仕様で規定されたインポート対象のパッケージ内のクラス名や page/tag ディレクティブの import 属性に指定されたクラス名が重複する場合，JSP コンパイル時にコンパイルエラーとなり，KDJE39143-E のメッセージが出力されることがあります。

JSP コンパイル時にコンパイルエラーとなるケースの具体例について説明します。なお，具体例では，次のクラス名が存在していることを前提としています。

- packageA.classA
- packageB.classA

● 異なるパッケージからのインポートクラス名が重複した場合

具体例について説明します。

次の指定内容の場合，複数のパッケージから同名のクラスをインポートしようとしているため，JSP コンパイル時にエラーとなります。

ファイルの種類	指定内容
JSP ファイル	<%@page import="packageA.classA" %>
簡易構築定義ファイル	webserver.jsp.additional.import.list=packageB.classA

● JSP ファイル内で使用するクラスのインポート元パッケージが特定できない場合

具体例について説明します。

次の指定内容の場合，JSP ファイル内で使用している「classA」が，「packageA.classA」または「packageB.classA」のどちらかを特定できないため，JSP コンパイル時にエラーとなります。

ファイルの種類	指定内容
JSP ファイル	<%@page import="packageA.*" %> <% System.out.println(classA.method1()); %>
簡易構築定義ファイル	webserver.jsp.additional.import.list=packageB.*

2.4　JSP デバッグ機能

この節では，JSP デバッグ機能について説明します。

JSP デバッグ機能を使用すると，ブレークポイントの設定などのデバッグツールの機能を JSP ファイルで実行できるようになり，変換後の java ソースでのデバッグが不要になります。なお，JSP デバッグ機能は，JSP 2.0 以降の JSP ファイルで利用できます。

この節の構成を次の表に示します。

表 2-8　この節の構成（JSP デバッグ機能）

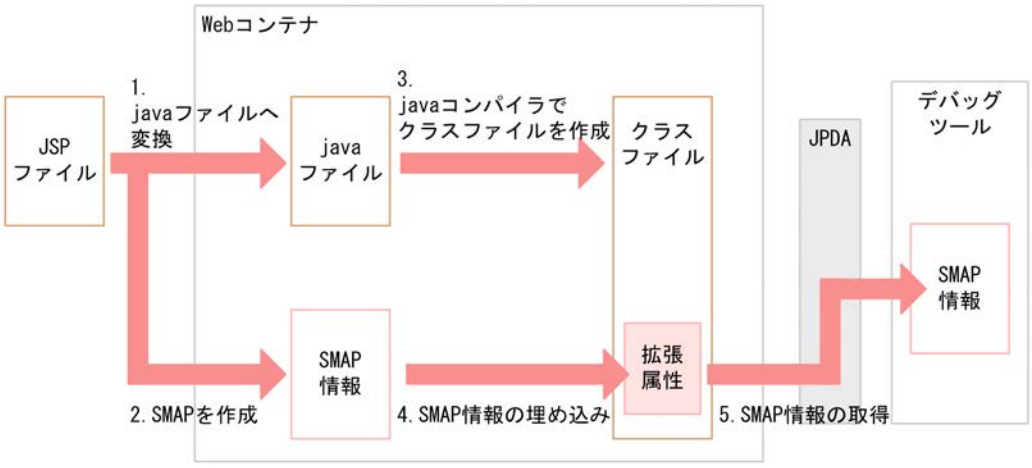
分類	タイトル	参照先
解説	JSP デバッグ機能の仕組み	2.4.1
	JSP デバッグ機能の使用手順	2.4.2
設定	実行環境での設定（J2EE サーバの設定）	2.4.3
注意事項	JSP デバッグ機能使用時の注意事項	2.4.4

注　「実装」および「運用」について，この機能固有の説明はありません。

2.4.1　JSP デバッグ機能の仕組み

JSP デバッグ機能の仕組みについて次の図に示します。

図 2-1　JSP デバッグの仕組み



図中のデータの流れについて説明します。

1. JSP ファイルからサーブレットの java ファイルへ変換します。
2. JSP ファイルの行と，JSP ファイルから変換した java ファイルの行のマッピングを記述した SMAP（Source MAP）を作成します。
3. java コンパイラで java ファイルからクラスファイルを作成します。
4. 作成したクラスファイルの拡張属性（SourceDebugExtension 属性）に SMAP 情報を埋め込みます。
5. デバッグの際，JPDA（Java Platform Debugger Architecture）を利用して，デバッグツールでクラスファイルに埋め込まれた拡張情報から埋め込まれている SMAP を取得します。

なお，SMAP 情報の埋め込みに失敗した場合は，KDJE39324-E のメッセージが出力されます。

注意事項

JSP デバッグ機能を使用する場合のクラス名

JSP デバッグ機能を使用する場合と使用しない場合とでは，コンパイル時に作成されるクラスのクラス名が異なります。JSP コンパイル結果のクラス名については，「2.5.7　JSP コンパイル結果のクラス名」を参照してください。

2.4.2　JSP デバッグ機能の使用手順

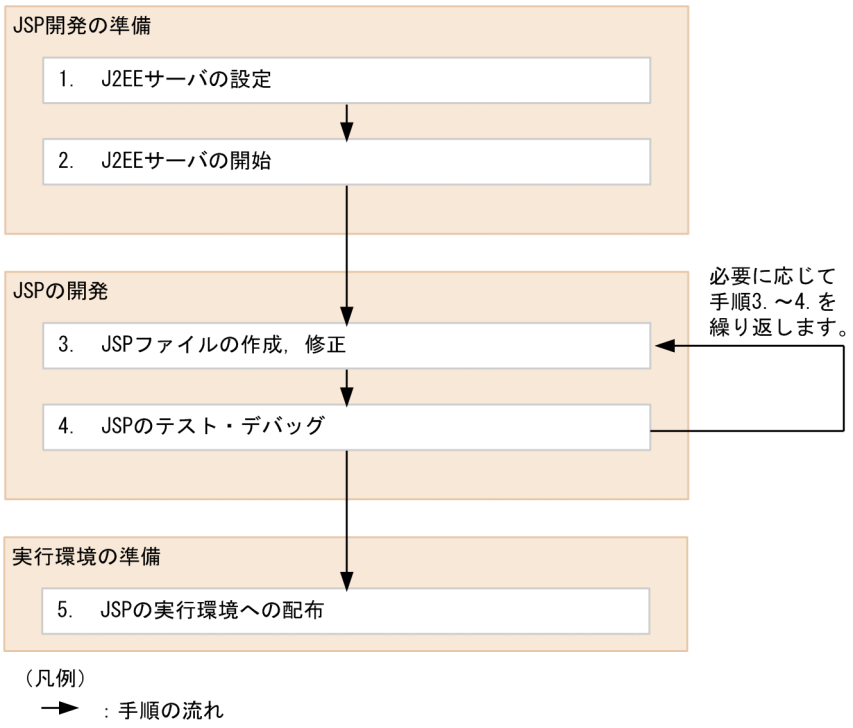
ここでは，JSP デバッグ機能の使用手順について説明します。JSP デバッグ機能は，J2EE サーバで JSP をコンパイルする場合，または cjjspc コマンドで JSP をコンパイルする場合に使用できます。

それぞれの場合の使用手順について説明します。

(1)　J2EE サーバで JSP をコンパイルする場合

J2EE サーバで JSP をコンパイルする場合の JSP デバッグ機能の使用手順の流れを次の図に示します。

図 2–2　JSP デバッグ機能の使用手順（J2EE サーバで JSP をコンパイルする場合）



図中の手順について説明します。

1. J2EE サーバの設定

   JSP デバッグ機能を有効にします。また，JSP のリロードを有効にします。JSP のリロードについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.8.9　JSP のリロード」を参照してください。

2. J2EE サーバの開始

   J2EE サーバを開始します。

JSP デバッグ機能を有効にすると，J2EE サーバの開始時には，KDJE39322-W のメッセージがメッセージログに出力されます。

3. JSP ファイルの作成，修正

JSP ファイルを作成，または修正します。

4. JSP のテスト・デバッグ

WTP などの JPDA に対応したデバッグツールを使用して，テストおよびデバッグをします。JSP を修正する場合は，手順の 3.に戻ります。

5. JSP の実行環境への配布

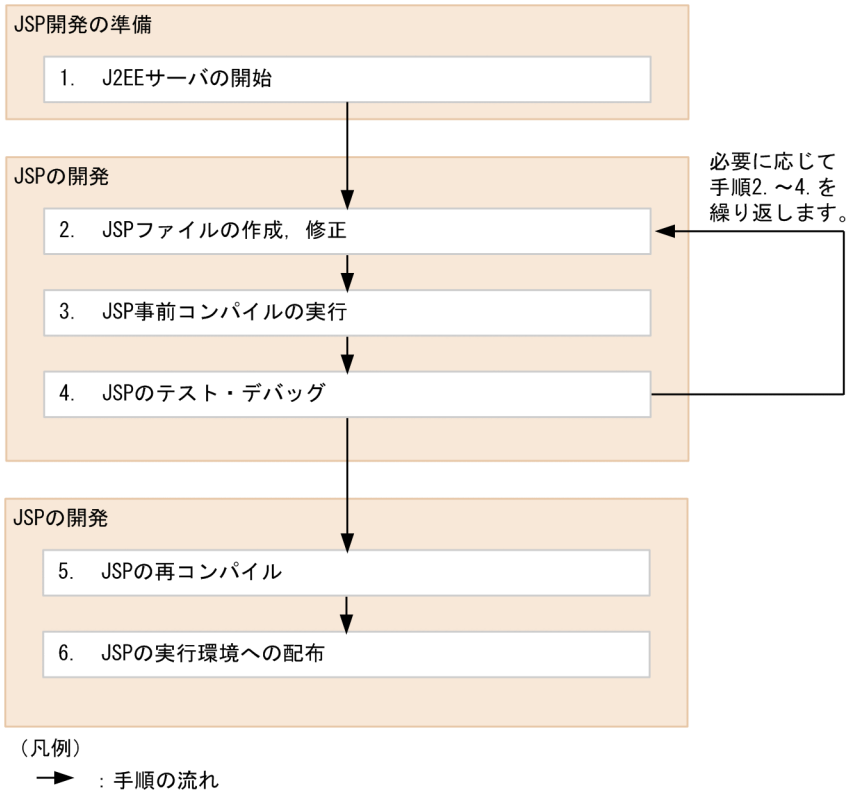
作成した JSP を含む J2EE アプリケーションを開発環境でエクスポートして，実行環境へインポートします。

(2)　cjjspc コマンドで JSP をコンパイルする場合

J2EE アプリケーション開始前に，すべての JSP をコンパイルしたい場合，cjjspc コマンドで JSP 事前コンパイルを実施します。JSP 事前コンパイルについては，「2.5　JSP 事前コンパイル機能とコンパイル結果の保持」を参照してください。

cjjspc コマンドで JSP 事前コンパイルを実施する場合の JSP デバッグ機能の使用手順の流れを次の図に示します。

図 2-3　JSP デバッグ機能の使用手順（cjjspc コマンドで JSP をコンパイルする場合）



図中の手順について説明します。

1. J2EE サーバの開始

JSP デバッグ機能を有効にします。また，JSP のリロードを有効にします。JSP のリロードについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.8.9　JSP のリロード」を参照してください。

JSP デバッグ機能を有効にすると，J2EE サーバの開始時には，KDJE39322-W のメッセージがメッセージログに出力されます。

2. JSP ファイルの作成，修正

JSP ファイルを作成，または修正します。

3. JSP 事前コンパイルの実行

cjjspc コマンドを使用して，JSP 事前コンパイルを実行します。-debugging オプションを指定してコマンドを実行し，JSP ファイルをコンパイルします。

cjjspc コマンド実行時には，KDJE53442-W のメッセージがコンソールログに出力されます。

4. JSP のテスト・デバッグ

WTP などの JPDA に対応したデバッグツールを使用して，テストおよびデバッグをします。JSP を修正する場合は，手順の 2.に戻ります。

5. JSP の再コンパイル

JSP ワークディレクトリを削除します。また，-debugging オプションを指定しないで cjjspc コマンドを使用して，JSP ファイルを再度コンパイルします。

なお，JSP ファイルを再コンパイルしない場合，JSP を実行できません。詳細は「2.4.4　JSP デバッグ機能使用時の注意事項」を参照してください。

6. JSP の実行環境への配布

作成した JSP を含む J2EE アプリケーションを開発環境でエクスポートして，実行環境へインポートします。

cjjspc コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル)」を参照してください。

2.4.3　実行環境での設定（J2EE サーバの設定）

JSP デバッグ機能を使用する場合，J2EE サーバの設定が必要です。

J2EE サーバの設定は，簡易構築定義ファイルで実施します。JSP デバッグ機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.jsp.debugging.enabled に指定します。このパラメタでは，JSP デバッグ機能を使用するかどうかを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

2.4.4　JSP デバッグ機能使用時の注意事項

ここでは，JSP デバッグ機能を使用する場合の注意事項について説明します。

(1)　JSP デバッグ機能で作成されたクラスファイルの削除

JSP デバッグ機能で作成されたクラスファイルは，JSP デバッグ機能が無効な環境では使用できません。JSP デバッグ機能が有効な環境で作成された J2EE アプリケーションを，JSP デバッグ機能が無効な環境に配布する場合，JSP デバッグ機能で作成されたクラスファイルを削除する必要があります。

クラスファイルを削除する方法を次に示します。

1. **展開ディレクトリ形式のアプリケーションについて JSP 事前コンパイル機能を使用して JSP をコンパイルした場合**

   JSP 事前コンパイル機能の JSP ワークディレクトリを削除してください。JSP ワークディレクトリについては,「2.5.5(2)　JSP コンパイル結果の出力先」を参照してください。

2. **1.以外の方法で JSP をコンパイルした場合**

   cjstopapp コマンドを使用して，J2EE アプリケーションを停止してください。

(2)　cjjspc コマンドを使用して JSP をコンパイルする場合の J2EE サーバの設定

cjjspc コマンドを使用して JSP をコンパイルする場合は，JSP が実行される J2EE サーバで JSP デバッグ機能を有効にしてください。

JSP デバッグ機能が無効な J2EE サーバのアプリケーションに対して，cjjspc コマンドに-debugging オプションを指定して JSP をコンパイルした場合，ロードするクラスファイルが異なります。そのため，JSP の HTTP リクエストに対して，Web コンテナがエラーステータスコード 404 を返します。

2.5　JSP 事前コンパイル機能とコンパイル結果の保持

この節では，JSP 事前コンパイル機能とコンパイル結果の保持について説明します。

通常，Web アプリケーション内の JSP ファイルは，JSP ファイルへの最初のリクエスト到着時に Web コンテナ上でコンパイルされます。JSP 事前コンパイル機能を使用すると，Web アプリケーションのデプロイ前にコンパイルできます。JSP 事前コンパイル機能であらかじめ JSP ファイルをコンパイルしておくと，JSP ファイルへの最初のリクエスト到着時のレスポンスタイムを短縮できます。

また，JSP コンパイル結果である，Java ソースファイルおよびクラスファイルを，J2EE サーバの再起動時に保持するかどうか設定できます。

この節の構成を次の表に示します。

表 2–9　この節の構成（JSP 事前コンパイル機能とコンパイル結果の保持）

分類	タイトル	参照先
解説	JSP 事前コンパイル機能の概要	2.5.1
	JSP 事前コンパイルの方法	2.5.2
	JSP 事前コンパイルの適用例	2.5.3
	JSP 事前コンパイルの実行時の処理	2.5.4
	JSP コンパイル結果のライフサイクルと出力先	2.5.5
	JSP 事前コンパイルを使用しない場合の JSP コンパイル結果	2.5.6
	JSP コンパイル結果のクラス名	2.5.7
設定	実行環境での設定（J2EE サーバの設定）	2.5.8

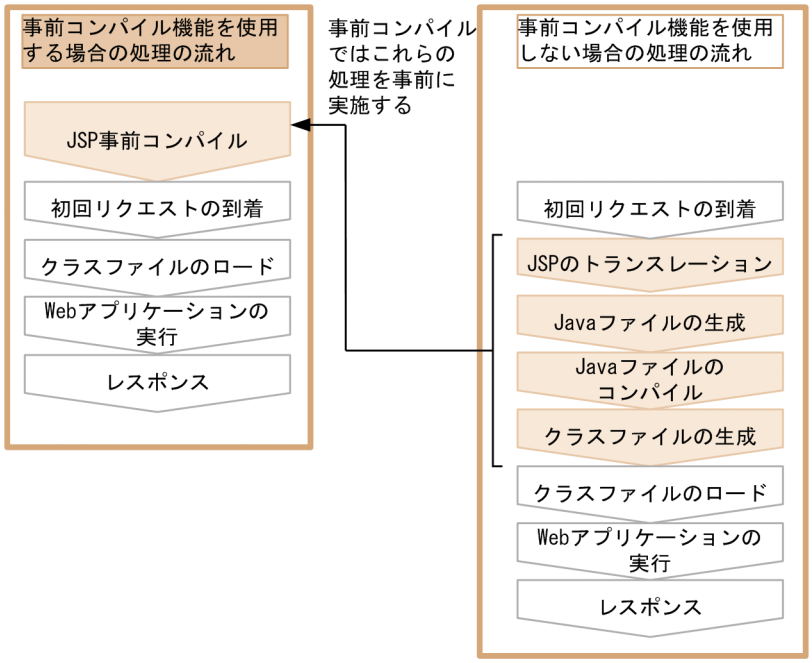
注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

2.5.1　JSP 事前コンパイル機能の概要

JSP 事前コンパイル機能とは，Web アプリケーションに含まれる JSP ファイルを，デプロイ前にコンパイルし，クラスファイルを生成する機能です。

JSP 事前コンパイル機能を使用した場合の処理の流れを次の図に示します。

図 2–4　JSP 事前コンパイル機能を使用した場合の処理の流れ



通常，Web アプリケーションに含まれる JSP ファイルは，JSP ファイルに最初のリクエストが到着したときにコンパイルされ，JSP ファイルからクラスファイルが生成されます。このため，JSP への初回リクエスト時のレスポンスタイムは遅くなります。また，web.xml で<load-on-startup>を指定していると，コンパイルのタイミングは Web アプリケーションの開始時となるため，JSP への初回リクエスト到着時のレスポンスタイムは短縮できます。ただし，Web アプリケーションの開始に時間が掛かります。

JSP 事前コンパイル機能を使用すると，あらかじめ，クラスファイルの生成までを実施しておけるので，JSP に最初にリクエストが到着したときのレスポンスタイムおよび Web アプリケーションの開始時間を短縮できます。

2.5.2　JSP 事前コンパイルの方法

JSP 事前コンパイル方法には次の二つがあります。

- cjjspc コマンドによる JSP 事前コンパイル
- cjstartapp コマンドによる J2EE アプリケーション開始時の JSP 事前コンパイル

ここでは，JSP 事前コンパイルを実行するコマンドの概要と，コンパイル方法について説明します。なお，コンパイル方法は，どの場面で JSP ファイルを事前にコンパイルするかによって異なります。JSP 事前コンパイルを適用する場面とコンパイル方法については，「2.5.3　JSP 事前コンパイルの適用例」を参照してください。

(1)　コマンドの概要

ここでは，JSP 事前コンパイルを実施するための前提条件と，JSP 事前コンパイル実行時に必要なファイル，および実行後に生成されるファイルについて説明します。

前提条件

JSP 事前コンパイルを実施するためには，コンパイルする JSP ファイルが Web アプリケーションのルートディレクトリ以下，またはそのサブディレクトリ以下に格納されていることが前提です。

JSP 事前コンパイルに必要なファイル

JSP 事前コンパイルを実施するためには，次に示すファイルが必要です。

- JSP ファイル（JSP 1.1，JSP 1.2，JSP 2.0，または JSP 2.1）※1
- JSP 2.0 仕様または JSP 2.1 仕様に準拠したタグファイル
- JSP ファイルおよびタグファイルから静的にインクルードされるファイル
- TLD ファイル※2
- web.xml※2，※3
- コンパイルに必要なクラスライブラリ

注※1

JSP ファイルとは，次に示すファイルを指します。

- 拡張子が.jsp または.jspx であるファイル（.jspx は JSP 2.0 以降の場合だけ）
- web.xml の<jsp-file>タグに指定されたファイル
- web.xml の<jsp-property-group><url-pattern>タグに合致するファイル（JSP 2.0 以降の場合だけ）

注※2

JSP 事前コンパイル実行時に DTD または XML スキーマに従っているかどうかが検証されます。

注※3

web.xml がない場合，Web アプリケーションのバージョンを 3.0 と見なしてコンパイルが実行されます。

JSP 事前コンパイル後に生成されるファイル（JSP コンパイル結果）

JSP ファイルやタグファイルから生成された，Java ソースファイルおよびクラスファイルを JSP コンパイル結果といいます。JSP 事前コンパイルを実施すると，JSP ワークディレクトリに次に示す JSP コンパイル結果が生成されます。

- JSP ファイルから生成された Java ソースファイルおよびクラスファイル
- タグファイルから生成された Java ソースファイルおよびクラスファイル

なお，JSP 事前コンパイル実行時には，Java ソースファイルを保存しておくかどうかを設定できます。

(2)　cjjspc コマンドによる JSP 事前コンパイル

cjjspc コマンドは，JSP 事前コンパイルを実施するためのコマンドです。アプリケーションの開発時などにこのコマンドを実施すると，Web アプリケーションに含まれる JSP ファイルをコンパイルできます。
cjjspc コマンドによる JSP 事前コンパイルには，次の二つの方法があります。

- JSP ファイル単位での事前コンパイル
  Web アプリケーションに含まれる JSP ファイルのうち，指定された JSP ファイルだけをコンパイルします。
- Web アプリケーション単位での事前コンパイル
  Web アプリケーションに含まれるすべての JSP ファイルをコンパイルします。

また，このコマンド実行時に次の内容を設定できます。

- コンパイル不要な JSP ファイルの指定
  コンパイル不要な JSP ファイルがある場合，あらかじめ不要なファイルを指定しておくことで，事前コンパイルの対象外にできます。指定方法には，コンパイル不要な JSP ファイル名をコマンドに一つずつ

指定する方法と，コンパイル不要な JSP ファイル名をファイルにまとめて記載したファイル（コンパイル対象外リストファイル）をコマンドに指定する方法があります。

- 実行結果リストファイルを出力するかどうかの指定

  実行結果リストファイルを出力するかどうかを指定できます。実行結果リストファイルとは，cjjspc コマンドの実行結果を出力したファイルです。コンパイルに成功した JSP ファイル，コンパイルに失敗した JSP ファイル，およびコンパイル対象外の JSP ファイルのパスを一覧で出力します。

- Java ソースファイルを保存するかどうかの指定

  JSP から生成された Java ソースファイルを保存しておくかどうかを指定できます。

- JSP コンパイル時の Java 言語仕様のバージョンの指定

  JSP トランスレーションによって生成された Java ソースファイルをコンパイルするときの Java 言語仕様のバージョンを指定できます。

- JSP ワークディレクトリ名を変更するかどうかの指定

  JSP ワークディレクトリとは，JSP コンパイル結果を格納するディレクトリのことです。JSP ワークディレクトリ名は変更できます。なお，JSP ワークディレクトリについては，「2.5.5(2)　JSP コンパイル結果の出力先」を参照してください。

- デフォルトの文字エンコーディングの指定

  JSP ファイルのデフォルトの文字エンコーディングを指定できます。なお，デフォルトの文字エンコーディングの概要については，「2.6　デフォルトの文字エンコーディング設定機能」を参照してください。また，07-00 で JSP 事前コンパイル機能を使用していた Web アプリケーションに対して，07-10 でデフォルトの文字エンコーディングを設定する場合の注意については，「2.6.8(5)　07-00 で JSP 事前コンパイルを実行した Web アプリケーションへの文字エンコーディング設定」を参照してください。

- JSP デバッグ機能を使用するかどうかの指定

  JSP デバッグ機能を使用するかどうかを指定できます。JSP デバッグ機能については「2.4　JSP デバッグ機能」を参照してください。

- 暗黙インポートするクラスの指定

  page/tag ディレクティブの import 属性暗黙インポート機能を使用して暗黙インポートするクラス名を指定できます。page/tag ディレクティブの import 属性暗黙インポート機能については「2.3.7 page/tag ディレクティブの import 属性暗黙インポート」を参照してください。

なお，これらの設定は，コマンドのオプションで指定します。JSP の事前コンパイルのコマンド（cjjspc コマンド）の使い方については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc (JSP の事前コンパイル)」を参照してください。

! 注意事項

JavaVM 起動オプションの変更

環境変数「CJ_CMD_JVM_ARGS」を設定すると，cjjspc コマンドが動作する JavaVM の起動オプションを変更できます。

デフォルトでは JavaVM の起動オプションに「-Xmx512m」（Java ヒープメモリ領域の最大値が 512MB）が設定されています。cjjspc コマンドで大規模な Web アプリケーションをコンパイルする場合，Java ヒープメモリ領域の最大値を超え，java.lang.OutOfMemoryError が発生するおそれがあります。したがって，大規模な Web アプリケーションをコンパイルする場合は，あらかじめ環境変数「CJ_CMD_JVM_ARGS」に，適切な Java ヒープメモリ領域を指定する必要があります。

参考

- cjjspc コマンドを使用した JSP 事前コンパイルで，JSP ファイルまたはタグファイルのトランスレーション時にエラーが発生すると，エラーメッセージが出力されます。エラーメッセージはコンソールに出力されます。

- コマンド実行時に標準出力または標準エラー出力にログを出力します。ログ出力の結果をファイルに残す場合は，コマンドの出力をファイルにリダイレクトしてください。
  ログ出力の結果をファイルに残す場合の指定例を示します。
  Windows の場合の指定例

```
> cjjspc -root D:¥app¥webapp1 1> .¥stdout.log 2> .¥stderr.log
```

  UNIX の場合の指定例

```
# cjjspc -root /app/webapp1 1> ./stdout.log 2> ./stderr.log
```

(3)　cjstartapp コマンドによる J2EE アプリケーション開始時の JSP 事前コンパイル

cjstartapp コマンドは，J2EE アプリケーションを開始するためのコマンドです。cjstartapp コマンドに，JSP 事前コンパイルをするオプションを指定すると，JSP の事前コンパイルを実施してから，J2EE アプリケーションが開始されます。J2EE アプリケーション開始時の JSP 事前コンパイルでは，J2EE アプリケーションに含まれるすべての JSP ファイルをコンパイルします。

このコマンドの実行時の動作はあらかじめ設定できます。設定できる内容を次に示します。

- Java ソースファイルを保存するかどうかの指定
  JSP ファイルから生成された Java ソースファイルを保存しておくかどうかを指定できます。
- JSP コンパイル時の Java 言語仕様のバージョンの指定
  JSP トランスレーションによって生成された Java ソースファイルをコンパイルするときの Java 言語仕様のバージョンを指定できます。
- JSP ワークディレクトリ名を変更するかどうかの指定
  JSP ワークディレクトリとは，JSP コンパイル結果を格納するディレクトリのことです。JSP ワークディレクトリ名は変更できます。なお，JSP ワークディレクトリについては，「2.5.5(2)　JSP コンパイル結果の出力先」を参照してください。
- 暗黙インポートするクラスの指定
  page/tag ディレクティブの import 属性暗黙インポート機能を使用して暗黙インポートするクラス名を指定できます。page/tag ディレクティブの import 属性暗黙インポート機能については「2.3.7 page/tag ディレクティブの import 属性暗黙インポート」を参照してください。

なお，これらの設定は，J2EE サーバの動作設定のカスタマイズで実施します。J2EE サーバの動作設定のカスタマイズについては，「2.5.8　実行環境での設定（J2EE サーバの設定）」を参照してください。

参考

cjstartapp コマンドを使用した JSP 事前コンパイルで，JSP ファイルまたはタグファイルのトランスレーション時にエラーが発生すると，エラーメッセージが出力されます。エラーメッセージは Web サーブレットログ，またはメッセージログに出力されます。

! 注意事項

JSP から生成される Java ソースのコンパイルについて

cjjspc コマンドを使用して生成されたクラスファイルは，J2EE サーバでの実行時に使用されます。cjjspc コマンドでは，J2EE サーバで生成されたクラスファイルと同じクラスファイルを生成します。

このため，JSP ファイルまたはタグファイルから生成された Java ソースをコンパイルする場合，-source オプションでの Java 言語仕様のバージョンの指定，または-classpath オプションでのクラスパスの指定以外はできません。Java 言語仕様のバージョンの指定方法についてはマニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル）」を参照してください。

2.5.3　JSP 事前コンパイルの適用例

JSP 事前コンパイルは次に示すときに使用できます。

- アプリケーション開発時
- システム運用時

JSP 事前コンパイルの適用場面と使用するコマンドの対応表を次に示します。

表 2-10　JSP 事前コンパイルの適用場面と使用するコマンドの対応

適用する場面			使用するコマンド	参照先
アプリケーション開発	Web アプリケーションの開発時		cjjspc コマンド	(1)
システム運用	J2EE アプリケーション開始時		cjstartapp コマンド	(2)
	J2EE アプリケーション入れ替え時	通常の入れ替え	cjstartapp コマンド	
		リロードによる入れ替え	cjjspc コマンド	
		リデプロイによる入れ替え	cjjspc コマンド	

JSP 事前コンパイルを使用する場面について次に説明します。使用するコマンドの概要については，「2.5.2　JSP 事前コンパイルの方法」を参照してください。

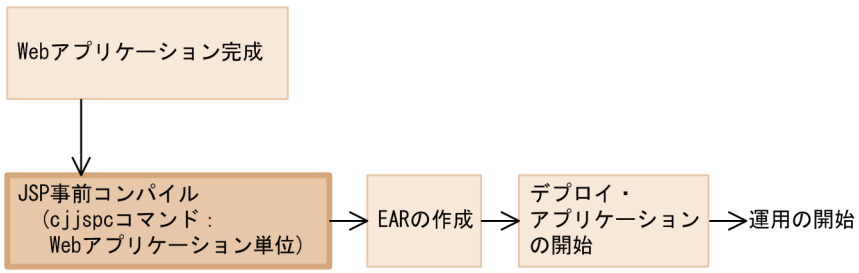
なお，cjstartapp コマンドで JSP 事前コンパイルを実施するためのオプションを指定すると，cjstartapp コマンドの実行時に JSP コンパイルが実施されます。この処理は J2EE サーバで行われるため，一時的に J2EE サーバプロセスのメモリ使用量が増加します。

(1)　アプリケーション開発での使用

Web アプリケーションの開発では，Web アプリケーション完成後に，JSP 事前コンパイルを実施します。

JSP 事前コンパイルを実施するには，cjjspc コマンドを使用します。Web アプリケーション開発での，JSP 事前コンパイルの適用例を次の図に示します。

図 2-5　Web アプリケーション開発での JSP 事前コンパイルの適用例



JSP 事前コンパイル機能を使用して，完成した Web アプリケーションに含まれるすべての JSP ファイルを，一括してコンパイルします。この結果，Web アプリケーション実行時の JSP 初回リクエストのレスポンスタイムを向上できます。

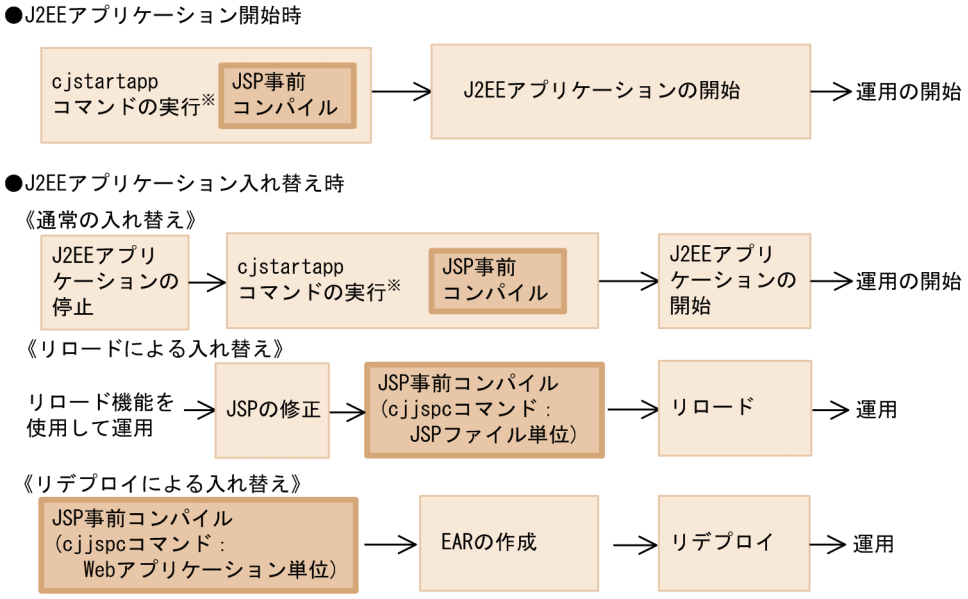
Web アプリケーション内のすべての JSP ファイルを一括でコンパイルするには，cjjspc コマンドの，Web アプリケーション単位での JSP 事前コンパイルを実施します。

なお，cjjspc コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル）」を参照してください。

(2)　システム運用での使用

システム運用時では，JSP 初回リクエストのレスポンスタイムを向上させるために，運用の開始前に，JSP 事前コンパイルを実施します。システム運用時の JSP 事前コンパイルは，J2EE アプリケーションを開始するときや，J2EE アプリケーションの入れ替えをするときに実施します。システム運用での，JSP 事前コンパイルの適用例を次の図に示します。

図 2-6　システム運用での JSP 事前コンパイルの適用



注※　JSP事前コンパイルをするためのオプションを指定すると，cjstartappコマンドの実行で，JSP事前コンパイルとJ2EEアプリケーションの開始の両方を実施します。

JSP 事前コンパイル機能を適用する場面ごとに概要を説明します。なお，システム運用時に JSP 事前コンパイルを使用するときのコンパイルの実施方法については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「5.6.3　J2EE アプリケーションの入れ替えと保守」を参照してください。

(a)　J2EE アプリケーション開始時

J2EE アプリケーション開始時に，インポート済みの J2EE アプリケーションに対して JSP 事前コンパイルを使用できます。J2EE アプリケーション開始時に実施する JSP 事前コンパイルは，cjstartapp コマンドに JSP 事前コンパイルを実施するためのオプションを指定して実行します。このコマンドを実行すると，J2EE アプリケーションの開始前に，J2EE アプリケーションに含まれる JSP ファイルを一括してコンパイルします。

(b)　J2EE アプリケーション入れ替え時

J2EE アプリケーションを入れ替える場合，入れ替えを実施する前に JSP 事前コンパイルを使用できます。なお，JSP 事前コンパイルの方法は，J2EE アプリケーション入れ替えの方法によって異なります。

- **通常の J2EE アプリケーションの入れ替えの場合**

  通常の J2EE アプリケーションの入れ替えとは，J2EE アプリケーションをいったん停止してから，新しいアプリケーションと入れ替える方法です。

  通常の J2EE アプリケーションの入れ替えの場合，入れ替え後の J2EE アプリケーションをインポートしたあとに，cjstartapp コマンドに JSP 事前コンパイルを実施するためのオプションを指定して，JSP 事前コンパイルを実行します。このコマンドを実行すると，入れ替え後の J2EE アプリケーションが開始される前に，J2EE アプリケーションに含まれる JSP ファイルが一括でコンパイルされます。

- **リロードによる J2EE アプリケーションの入れ替えの場合**

  リロードによる J2EE アプリケーションの入れ替えとは，J2EE アプリケーションを停止しないで，新しい J2EE アプリケーション（展開形式のアプリケーション）と入れ替える方法です。

  JSP コンパイル結果を含む J2EE アプリケーションをリロード機能を使用して運用している場合で，JSP ファイルの修正が発生したときに，cjjspc コマンドによって修正した JSP ファイルのコンパイルができます。

- **リデプロイによる J2EE アプリケーションの入れ替えの場合**

  リデプロイによる J2EE アプリケーションの入れ替えとは，J2EE アプリケーションを停止しないで，新しい J2EE アプリケーション（アーカイブ形式のアプリケーション）と入れ替える方法です。

  リデプロイによる J2EE アプリケーションの入れ替えの流れを次に示します。

  1. cjjspc コマンドで JSP 事前コンパイルを実施し，J2EE アプリケーションに含まれるすべての JSP ファイルを一括してコンパイルします。
  2. 1.で生成した JSP コンパイル結果を含んだ EAR ファイルを作成します。
  3. 2.の EAR ファイルをリデプロイします。

なお，J2EE アプリケーションの入れ替えの概要については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「5.6.3　J2EE アプリケーションの入れ替えと保守」を参照してください。なお，リロードについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.8　J2EE アプリケーションの更新検知とリロード」を参照してください。

2.5.4　JSP 事前コンパイルの実行時の処理

ここでは，JSP 事前コンパイル実行時に実施されるチェックや JSP 事前コンパイル機能を実行した J2EE アプリケーションの動作について説明します。

(1)　JSP 事前コンパイルで実施されるチェック

JSP 事前コンパイルの実行時には，web.xml の妥当性チェック，および JSP コンパイル結果のバージョンチェックが実施されます。

(a)　web.xml の妥当性チェック

JSP 事前コンパイル機能では，コンパイル処理を実行する前に，web.xml が DTD または XML スキーマに従っているかどうかの検証が実施されます。また，JSP 事前コンパイル時に参照する要素については，JSP 事前コンパイルに必要な範囲で，設定値が妥当であるかどうかについても検証されます。スキーマに従っていない場合は，JSP ファイルから Java ファイルを生成する JSP のトランスレーション時に，エラーが発生します。

JSP 事前コンパイル実行時に検証される web.xml の要素を次の表に示します。

表 2–11　JSP 事前コンパイル時に検証される web.xml の要素

タグ名	タグの説明	Servlet のバージョン				
		2.2	2.3	2.4	2.5	3.0
<!DOCTYPE>	DOCTYPE 宣言	○	○	×	×	×
<web-app>	ルートタグ	○	○	○	○	○
<servlet>	サーブレットについての定義	○	○	○	○	○
<jsp-file>	JSP ファイル名	○	○	○	○	○
<taglib>	タグライブラリについての定義	○	○	−	−	−
<taglib-uri>	タグライブラリの URI	○	○	−	−	−
<taglib-location>	タグライブラリ記述ファイル（TLD）の場所	○	○	−	−	−
<jsp-config>	JSP についての定義	−	−	○	○	○
<taglib>	タグライブラリについての定義	−	−	○	○	○
<taglib-uri>	タグライブラリの URI	−	−	○	○	○
<taglib-location>	タグライブラリ記述ファイル（TLD）の場所	−	−	○	○	○
<jsp-property-group>	指定した URL パターンに合致する JSP の設定	−	−	○	○	○
<url-pattern>	設置を適用する JSP の URL パターン	−	−	○	○	○
<el-ignored>	EL（式言語）を無視するかの設定	−	−	○	○	○
<scripting-invalid>	スクリプティング要素を無効にするかの設定	−	−	○	○	○
<page-encoding>	ページエンコーディング名	−	−	○	○	○
<include-prelude>	JSP のヘッダとしてインクルードするファイル	−	−	○	○	○
<include-coda>	JSP のフッタとしてインクルードするファイル	−	−	○	○	○
<is-xml>	XML 形式で記述されているかの設定	−	−	○	○	○
<deferred-syntax-allowed-as-literal>	EL が使えない部分で#{の文字列があった場合にエラーにするかの設定	−	−	−	○	○
<trim-directive-	JSP から余分な空白を出力しないようにするかの設定	−	−	−	○	○

タグ名				タグの説明	Servlet のバージョン				
					2.2	2.3	2.4	2.5	3.0
			whitespaces >	JSP から余分な空白を出力しないようにするかの設定	－	－	－	○	○

（凡例）○：検証される　　×：検証されない　　－：サポートしていない要素

(b)　JSP コンパイル結果のバージョンチェック

JSP 事前コンパイル機能使用時，J2EE サーバは web.xml で指定された Web アプリケーションのバージョンと，JSP コンパイル時の JSP のバージョンが合致するかチェックします。バージョンのチェックは次のタイミングで実施されます。

- アプリケーション開始時に，JSP 事前コンパイルを実施する指定をしていないとき（-jspc オプションを指定しないで，cjstartapp コマンドでアプリケーションを開始したとき）
- cjjspc コマンドを使用して，コンパイル対象外ファイルを指定して Web アプリケーション単位の JSP 事前コンパイルを実行したとき
- cjjspc コマンドを使用して，JSP ファイル単位に JSP 事前コンパイルを実行したとき

JSP から生成されるクラスファイルは，web.xml で指定された Web アプリケーションのバージョンに依存します。JSP 事前コンパイル実行時の Web アプリケーションとバージョンが異なる Web アプリケーションで使用することはできません。このため，Web アプリケーションのバージョンを変更した際には，すべての JSP ファイルをコンパイルする必要があります。

なお，次の場合は，Web アプリケーションに含まれるすべての JSP をコンパイルするため，JSP コンパイル結果のチェックは実施されません。

- コンパイル対象外ファイルを指定しないで Web アプリケーション単位の JSP 事前コンパイルを実行したとき
- アプリケーション開始時の JSP 事前コンパイルを実行したとき

参考

JSP コンパイル結果のバージョンチェックが実施されると，コンパイル対象の Web アプリケーションの JSP ワークディレクトリに，JSP ファイルのバージョン情報が記述されたバージョン情報ファイルが生成されます。バージョン情報ファイルは次の場所に生成されます。

<Web アプリケーションのディレクトリ>/WEB-INF/<JSP ワークディレクトリ名>/WEB-INF/<JSP ワークディレクトリ名>/jsp_compile_info

(c)　TLD ファイルのチェック

TLD ファイルは，JSP 事前コンパイル実行時に DTD または XML スキーマに従っているかどうか検証されます。Web アプリケーションのバージョンごとに TLD ファイルのチェックについて説明します。

- Web アプリケーションのバージョンが 2.4 以降の場合
  デフォルトで検証が実施されます。なお，スキーマに従っていない場合，JSP ファイルのトランスレーション時にエラーとなります。
- Web アプリケーションのバージョンが 2.3 以前の場合
  あらかじめ，検証をするかどうか設定しておく必要があります。TLD ファイルを検証する設定をしている場合に，JSP 事前コンパイル実行時に検証されます。

なお，TLD ファイルの検証については，「2.5.8　実行環境での設定（J2EE サーバの設定）」を参照してください。

(2)　JSP 事前コンパイル機能を実施したアプリケーションでの JSP ファイルの扱い

JSP 事前コンパイル機能を実行した J2EE アプリケーションの動作について説明します。

(a)　リクエスト実行時および J2EE アプリケーション開始時の動作

JSP 事前コンパイルを実施していると，リクエスト実行時には JSP コンパイルは実施されません。事前コンパイル時に作成した JSP のクラスファイルがロードされ，実行されます。

このとき，JSP ファイルからコンパイルされたクラスファイルがない場合などには，エラーになります。JSP 事前コンパイルを実施していて，ファイルがない場合の J2EE サーバの挙動を次の表に示します。

表 2-12　ファイルがない場合の J2EE サーバの挙動（JSP 事前コンパイルを実行しているとき）

存在しないファイル		J2EE サーバの挙動
JSP ファイル	JSP ファイル	JSP ファイルを参照しない
	クラスファイル	404 エラーを返す
タグファイル	タグファイル	タグファイルを参照しない
	クラスファイル	500 エラーを返す（java.lang.NoClassDefFoundError が発生する）
静的インクルードされたファイル		静的インクルードされたファイルを参照しない
TLD ファイル		TLD ファイルを参照しない

事前コンパイルを実施してない場合でファイルがないときは，J2EE サーバは次のように動作します。

表 2-13　ファイルがない場合の J2EE サーバの挙動（JSP 事前コンパイルを実行していないとき）

存在しないファイル		J2EE サーバの挙動
JSP ファイル	JSP ファイル	404 エラーを返す
	クラスファイル	JSP ファイルをコンパイルする
タグファイル	タグファイル	500 エラーを返す（コンパイルエラー）
	クラスファイル	タグファイルをコンパイルする
静的インクルードされたファイル		500 エラーを返す（コンパイルエラー）
TLD ファイル		

(b)　J2EE アプリケーション開始時の動作

web.xml で JSP ファイルに<load-on-startup>を指定した Web アプリケーションの JSP ファイルを事前コンパイルした場合，J2EE アプリケーション開始時には，JSP コンパイルは実施されません。JSP 事前コンパイル時に生成されたクラスファイルがロードされ，jspInit メソッドが実行されます。このとき，JSP のクラスファイルまたは JSP が依存するクラスファイルがない場合は，JSP ファイルのロードに失敗します。

なお，サーブレットと JSP のエラー通知の設定が有効になっている場合は，Web アプリケーション開始時に失敗します。サーブレットと JSP のエラー通知の設定については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「9.16　サーブレットと JSP のエラー通知の設定」を参照してください。

(3) JSP 事前コンパイル機能の注意

ここでは，JSP 事前コンパイル機能での注意事項を説明します。

- 「jsp_precompile」を付加したリクエストの送信

  JSP 事前コンパイルを実行したアプリケーションに，クエリ文字列「jsp_precompile」，または「jsp_precompile=true」を付加したリクエストを送信しても，JSP コンパイルは実行されません。

- JSP 事前コンパイルのコマンドの複数起動による同じ Web アプリケーションの操作

  cjjspc コマンドの複数起動によって同じ Web アプリケーションに対する JSP 事前コンパイルを実行することはできません。また，アプリケーション開始時の JSP 事前コンパイル実行時に cjjspc コマンドによって同じ Web アプリケーションに対するコンパイル処理を実行することはできません。

  なお，コマンドの排他処理のため，JSP 事前コンパイル実行中には，JSP ワークディレクトリにロックファイルが生成されます。ロックファイルは次の場所に生成されます。

  <Web アプリケーションのディレクトリ>/WEB-INF/<JSP ワークディレクトリ名>/WEB-INF/<JSP ワークディレクトリ名>/ExecutingJspPrecompilation.lock

- JSP コンパイル結果を使用するアプリケーションに移行する場合

  アーカイブ形式の J2EE アプリケーションの場合，アプリケーション開始時の JSP 事前コンパイルで生成したコンパイル結果は，アプリケーションの停止時に削除されます。

  アプリケーション開始時の JSP 事前コンパイルで生成した JSP コンパイル結果を，アプリケーションの停止後も利用する場合の手順を，J2EE アプリケーションの形式ごとに説明します。

  アーカイブ形式の J2EE アプリケーションの場合

  1. アプリケーション開始時の JSP 事前コンパイルを実行する
  2. アプリケーションをエクスポートする
  3. リデプロイ機能などを使用して，JSP コンパイル結果を含むアプリケーションに入れ替える

  展開ディレクトリ形式の J2EE アプリケーションの場合

  1. cjjspc コマンドまたはアプリケーション開始時に JSP 事前コンパイルを実行する

- JSP コンパイル結果を使用しないアプリケーションに移行する場合

  JSP 事前コンパイルで生成された JSP コンパイル結果を使用しない場合は，JSP ワークディレクトリを，ディレクトリごと削除する必要があります。

  JSP コンパイル結果を使用しない場合の手順を，J2EE アプリケーションの形式ごとに説明します。

  アーカイブ形式の J2EE アプリケーションの場合

  1. J2EE アプリケーションをエクスポートする
  2. EAR ファイルを展開する
  3. <Web アプリケーションのルートディレクトリ>/WEB-INF の下にある，JSP ワークディレクトリをディレクトリごと削除する
  4. EAR ファイルを作成する
  5. リデプロイ機能などを使用して，J2EE アプリケーションを入れ替える

  展開ディレクトリ形式の J2EE アプリケーションの場合

  1. J2EE アプリケーションを停止する

2. <Web アプリケーションのルートディレクトリ>/WEB-INF の下にある，JSP ワークディレクトリをディレクトリごと削除する
3. J2EE アプリケーションを開始する

- **JSP ファイルが依存するファイルを修正した場合**

  タグファイル，静的にインクルードされたファイル，または TLD ファイルを更新した場合は，更新したファイルを参照するすべての JSP ファイルをコンパイルする必要があります。

- **JSP 事前コンパイル機能を使用する展開ディレクトリ形式の J2EE アプリケーションを更新する場合**

  JSP 事前コンパイル機能を使用する展開ディレクトリ形式の J2EE アプリケーションを更新する場合，次の点に注意してください。

  - JSP ファイルまたはタグファイルを J2EE アプリケーションに追加した場合

    追加した JSP ファイルまたはタグファイルを参照する JSP ファイルを，すべてコンパイルしてください。なお，JSP ファイルのコンパイルには，JSP 事前コンパイル機能を使用してください。

  - 開発環境で更新した JSP コンパイル結果を実行環境に反映する場合

    開発環境の J2EE アプリケーションの JSP ワークディレクトリ下にあるクラスファイルを，実行環境の J2EE アプリケーションの JSP ワークディレクトリにコピーしてください。この場合，開発環境で JSP 事前コンパイル実行時に更新された，すべてのクラスファイルがコピー対象となります。

2.5.5　JSP コンパイル結果のライフサイクルと出力先

JSP は，Web コンテナ上でコンパイルされ，Java ソースファイルおよびクラスファイルが生成されます。Web コンテナでは，JSP のコンパイル結果である，Java ソースファイルおよびクラスファイルを，J2EE サーバの再起動時に保持するかどうか設定できます。ここでは，JSP ファイルのコンパイル結果を保持するための設定について説明します。

JSP 事前コンパイル機能を使用している場合の，JSP コンパイル結果のライフサイクルと，JSP コンパイル結果の出力先について説明します。

(1)　JSP コンパイル結果のライフサイクル

JSP 事前コンパイルを使用している場合の JSP のコンパイル結果のライフサイクルについて説明します。

コンパイル結果の生成

JSP 事前コンパイル機能を使用する場合は，次のどちらかのタイミングでコンパイル結果が生成されます。

- cjjspc コマンドを実行するとき
- cjstartapp コマンドに-jspc オプションを指定して Web アプリケーションを開始するとき

コンパイル結果の削除

アーカイブ形式の J2EE アプリケーションの場合，アプリケーション開始時の JSP 事前コンパイルを実行して生成されたコンパイル結果は，アプリケーションの停止時に削除されます。

(2)　JSP コンパイル結果の出力先

JSP 事前コンパイルを実施すると，JSP ワークディレクトリが作成され，JSP コンパイル結果は JSP ワークディレクトリに出力されます。ただし，コンパイル対象となる JSP ファイルが存在しない場合は，JSP ワークディレクトリは作成されません。出力されるファイルは次のとおりです。

1. JSP ファイルから生成された Java ソースファイル※

2. 1.の Java ソースファイルをコンパイルしたクラスファイル

3. タグファイルから生成された Java ソースファイル※

4. 3.の Java ソースファイルをコンパイルしたクラスファイル

注※　JSP 事前コンパイルを実施する場合は，これらのファイルを保存するかどうかを設定できます。

ここでは，デフォルトの出力先と，出力先のディレクトリ構成について説明します。なお，出力されるクラス名については，「2.5.7　JSP コンパイル結果のクラス名」を参照してください。

(a)　デフォルトの出力先

JSP 事前コンパイルを実行する場合, コンパイル結果は JSP ワークディレクトリに出力されます。デフォルトの JSP ワークディレクトリは次の場所になります。

Windows の場合

<Web アプリケーションの WEB-INF ディレクトリ>¥cosminexus_jsp_work

UNIX の場合

<Web アプリケーションの WEB-INF ディレクトリ>/cosminexus_jsp_work

WEB-INF ディレクトリがない場合, JSP ワークディレクトリ作成時に WEB-INF ディレクトリおよび JSP ワークディレクトリが自動的に作成されます。

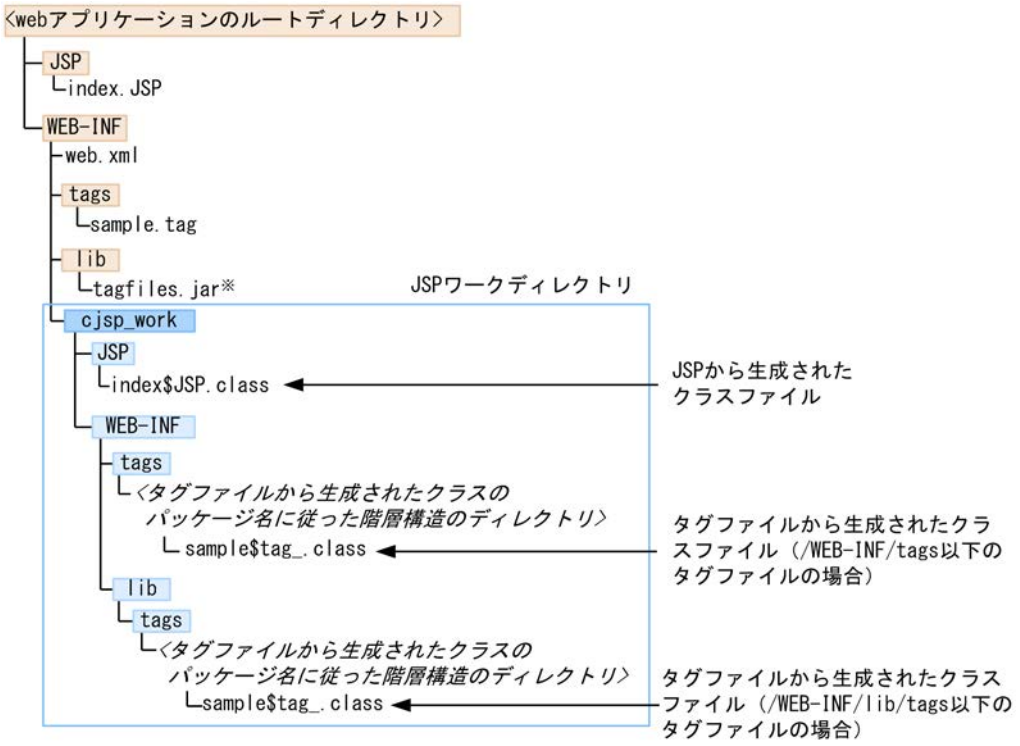
なお，JSP ワークディレクトリ名はデフォルト値が設定されていますが，必要に応じて変更できます。cjjspc コマンドで JSP 事前コンパイルを実施する場合の JSP ワークディレクトリ名の変更については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル)」を参照してください。cjstartapp コマンドで J2EE アプリケーションの開始時に JSP 事前コンパイルを実施する場合の JSP ワークディレクトリ名の変更については，「2.5.8　実行環境での設定（J2EE サーバの設定)」を参照してください。

また，JSP ワークディレクトリ名を変更した場合は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，パラメタ webserver.jsp.precompile.jsp_work_dir で変更した JSP ワークディレクトリ名を指定する必要があります。簡易構築定義ファイル, および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(b)　出力先のディレクトリ構成

JSP コンパイル結果は，JSP ワークディレクトリに出力されます。JSP コンパイル結果の出力先ディレクトリ構成を次の図に示します。なお，この図では JSP ワークディレクトリはデフォルトのディレクトリ名となっています。

図 2–7　JSP コンパイル結果の出力先ディレクトリ構成（JSP 事前コンパイルを実行している場合）



注※　タグファイルは，META-INF/tags/sample/sample.tagとします。

ディレクトリ構成について説明します。

- タグファイルから生成されたクラスのパッケージ名は次の形式となります。

  WEB-INF/tags 下のタグファイルの場合

  org.apache.jsp.tag.web.</WEB-INF/tags ディレクトリ下のパス>

  WEB-INF/lib 下の jar ファイルに含まれるタグファイルの場合

  org.apache.jsp.tag.meta.<jar ファイル名をエンコードした文字列>.<jar ファイル内の META-INF/tags ディレクトリ下のパス>

- JSP ファイルおよびタグファイルから生成されるクラスファイルの出力先ディレクトリのパス長は，OS の上限によって制限があります。OS の上限を超えるパス長になる場合は，JSP ワークディレクトリ名を変更してください。

! 注意事項

複数の J2EE サーバで同じ JSP ワークディレクトリを指定した場合，同じコンテキストルートのアプリケーションをデプロイしたとき，JSP ワークディレクトリの状態が不正となるおそれがあります。

2.5.6　JSP 事前コンパイルを使用しない場合の JSP コンパイル結果

JSP 事前コンパイルを使用しない場合，JSP ファイルのコンパイルは，JSP ファイルへの初回アクセス時に実施されます。ここでは，JSP 事前コンパイル機能を使用しない場合の，JSP コンパイル結果，およびコンパイル結果の出力先を変更する方法について説明します。

(1)　JSP コンパイル結果のライフサイクル

JSP 事前コンパイルを使用しない場合の JSP のコンパイル結果のライフサイクルについて説明します。

コンパイル結果の生成

JSP 事前コンパイル機能で事前に JSP ファイルのコンパイルを実施していない場合は，JSP コンパイル結果は次のどちらかのタイミングで生成されます。

- JSP に最初にアクセスするとき
- DD（web.xml）で JSP に対し<load-on-startup>が指定されている，Web アプリケーションを開始するとき

コンパイル結果の削除

JSP のコンパイル結果は，次のタイミングで削除されます。

- J2EE アプリケーションのアンデプロイ時
- J2EE サーバの起動時※
- J2EE サーバの終了時※

注※

JSP コンパイル結果を保持しない設定にしている場合，削除されます。なお，J2EE サーバ起動時については，コンパイル結果の削除は，サーバが強制終了された場合に備えて実施されます。

(2)　JSP ファイルのコンパイル結果の保持

JSP 事前コンパイルをしない場合，Web コンテナでは，JSP のコンパイル結果である，Java ソースファイルおよびクラスファイルを，J2EE サーバの再起動時に保持するかどうか設定できます。

JSP ファイルのコンパイル結果を保持するための設定は，J2EE サーバのプロパティをカスタマイズして設定します。J2EE サーバの動作設定のカスタマイズについては，「2.5.8　実行環境での設定（J2EE サーバの設定）」を参照してください。

! 注意事項

Web アプリケーションアンデプロイ時の注意

デフォルトでは，JSP のコンパイル結果を保持する設定になっています。また，JSP コンパイル結果を保持する設定をしていても，Web アプリケーションをアンデプロイすると，JSP のコンパイル結果は削除されます。このため，ユーザは，サーバの再起動時に JSP のコンパイル結果を削除する必要はありません。JSP コンパイル結果を保持する設定で Web コンテナを稼働したあと，JSP コンパイル結果が不要となった場合は，J2EE アプリケーションをアンデプロイしてください。

JSP のコンパイル結果は保持する設定にしておくことをお勧めします。

---

(3)　JSP コンパイル結果の出力先

JSP 事前コンパイルを実施していない場合，JSP コンパイル結果は JSP 用テンポラリディレクトリに出力されます。

出力されるファイルは次のとおりです。

1. JSP ファイルから生成された Java ソースファイル
2. 1.の Java ソースファイルをコンパイルしたクラスファイル
3. タグファイルから生成された Java ソースファイル
4. 3.の Java ソースファイルをコンパイルしたクラスファイル

ここでは，デフォルトの出力先と，出力先のディレクトリ構成について説明します。

なお，出力されるクラス名については，「2.5.7　JSP コンパイル結果のクラス名」を参照してください。

(a)　デフォルトの出力先

JSP 事前コンパイルを実行していない場合，JSP コンパイル結果は，JSP 用テンポラリディレクトリ下に作成される，Web アプリケーション単位のディレクトリに出力されます。デフォルトの JSP 用テンポラリディレクトリは次の場所になります。

Windows の場合

<Application Server のインストールディレクトリ>¥CC¥server¥repository¥<サーバ名称>¥web

UNIX の場合

/opt/Cosminexus/CC/server/repository/<サーバ名称>/web

なお，JSP 用テンポラリディレクトリは，デフォルト値が設定されていますが，必要に応じて変更できます。JSP 用テンポラリディレクトリの変更については，「2.5.8　実行環境での設定（J2EE サーバの設定）」を参照してください。

JSP 用テンポラリディレクトリ下には Web アプリケーション単位でディレクトリが作成され，該当する Web アプリケーション内の JSP コンパイル結果が出力されます。

なお，Web アプリケーション単位のディレクトリは，コンテキストルート名を基にしたディレクトリ名となります。コンテキストルート名にスラッシュ（/），ドル記号（$），パーセント（%），プラス記号（+）が含まれる場合は，次に示す文字に変換されます。

変換前の文字	変換後の文字
/	$2f
$	$24
%	$25
+	$2b

例：

JSP 用テンポラリディレクトリがデフォルト，コンテキストルート名が「J2EE_AP1/WEB_AP1_war」である場合，該当する Web アプリケーションの JSP コンパイル結果の出力先を次に示します。

- Windows の場合

  <Application Server のインストールディレクトリ>¥CC¥server¥repository¥<サーバ名称>¥web¥J2EE_AP1$2fWEB_AP1_war

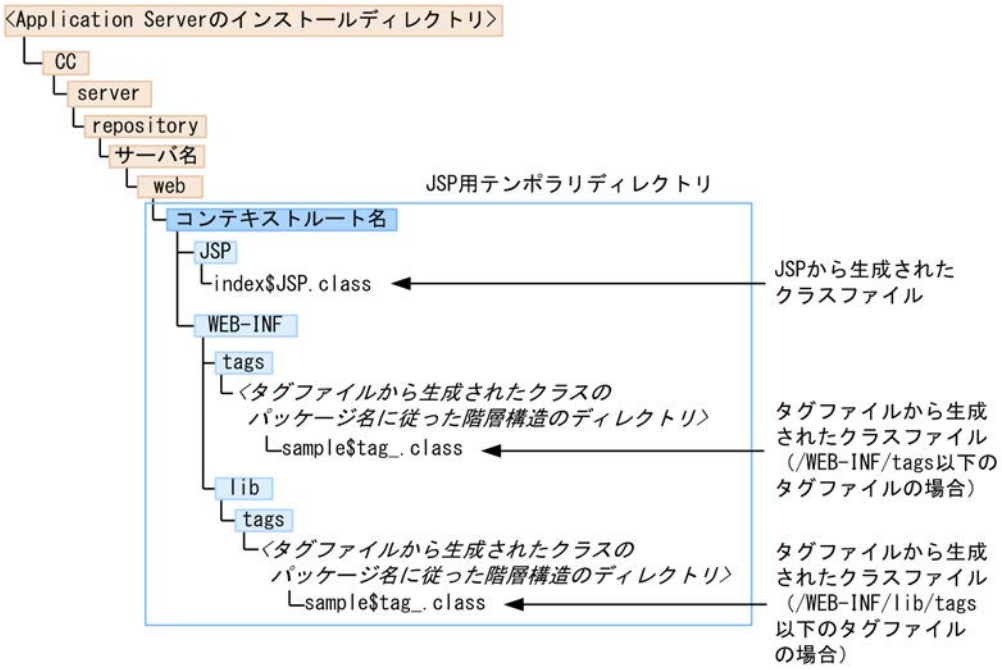
- UNIX の場合

  /opt/Cosminexus/CC/server/repository/<サーバ名称>/web/J2EE_AP1$2fWEB_AP1_war

(b)　出力先のディレクトリ構成

JSP コンパイル結果の出力先ディレクトリ構成を次の図に示します。

図 2–8　JSP コンパイル結果の出力先ディレクトリ構成（JSP 事前コンパイルを実行していない場合）



ディレクトリ構成について説明します。

- タグファイルから生成されたクラスのパッケージ名は次の形式となります。

  WEB-INF/tags 下のタグファイルの場合

  org.apache.jsp.tag.web.</WEB-INF/tags ディレクトリ下のパス>

  WEB-INF/lib 下の jar ファイルに含まれるタグファイルの場合

  org.apache.jsp.tag.meta.<jar ファイル名をエンコードした文字列>.<jar ファイル内の META-INF/tags ディレクトリ下のパス>

- JSP ファイルおよびタグファイルから生成されるクラスファイルの出力先ディレクトリのパス長は，OS の上限によって制限があります。OS の上限を超えるパス長になる場合は，JSP ワークディレクトリ名を変更してください。

2.5.7　JSP コンパイル結果のクラス名

ここでは，JSP ファイルまたはタグファイルから生成されるクラス名の形式およびクラス名の変換規則について説明します。

(1)　クラス名の形式

JSP ファイルまたはタグファイルから生成されるクラスのクラス名は，ファイルの種類，および JSP デバッグ機能が有効かどうかで形式が異なります。クラス名の形式について，次の表に示します。

表 2–14　JSP ファイルまたはタグファイルから生成されるクラスのクラス名の形式

ファイルの種別	JSP デバッグ機能が無効の場合	JSP デバッグ機能が有効の場合
JSP ファイル	<ファイル名>	<ファイル名>_jsp
タグファイル	<ファイル名>_	<ファイル名>_tag

＜ファイル名＞には「(2)　クラス名の変換規則」で示す規則が適用されます。

(2)　クラス名の変換規則

JSP ファイルまたはタグファイルから生成されるクラスのクラス名に，クラス名として使用できない文字，アンダースコア（_），またはドル記号（$）が含まれる場合，次の変換規則が順に適用されます。

1. 先頭の文字がパッケージ名，およびクラス名の先頭の文字として使用できない文字の場合は，先頭にドル記号（$）を付与する。
2. ファイル名の中のピリオド（.）はドル記号（$）に変換する。
3. クラス名として使用できない文字は，アンダースコア（_）と，使用できない文字を 4 けたの 16 進数表現の文字列に変換する。
   16 進数表現で使用される英字は小文字です。

1.の変更規則は先頭文字だけに適用されます。2., および 3.の変換規則は文字列の先頭から順に適用されます。

JSP ファイルまたはタグファイルから生成されるクラスのクラス名の変換例を次の表に示します。

表 2–15　JSP ファイルまたはタグファイルから生成されるクラスのクラス名の変換例

ファイルの種別	ファイル名	JSP デバッグ機能	クラス名
JSP ファイル	index.jsp	−	index$jsp
		○	index$jsp_jsp
	10test-10.jspx	−	$10test_002d10$jspx
		○	$10test_002d10$jspx_jsp
	test.tag	−	test$tag
		○	test$tag_jsp
タグファイル	tagfile1.tag	−	tagfile1$tag_
		○	tagfile1$tag_tag
	Tag_File$10.tagx	−	Tag_005fFile_002410$tagx_
		○	Tag_005fFile_002410$tagx_tag

（凡例）−：無効　○：有効

2.5.8　実行環境での設定（J2EE サーバの設定）

JSP 事前コンパイル機能を使用する場合，またはコンパイル結果の保持をする場合，J2EE サーバの設定が必要です。

J2EE サーバの設定は，簡易構築定義ファイルで実施します。JSP 事前コンパイル機能を使用する場合，またはコンパイル結果の保持をする場合の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの JSP 事前コンパイル機能を使用する場合，またはコンパイル結果の保持をする場合の定義について次の表に示します。

表 2-16　簡易構築定義ファイルでの JSP 事前コンパイル機能を使用する場合，またはコンパイル結果の保持をする場合の定義

項目	指定するパラメタ	設定内容
JSP 事前コンパイル	webserver.jsp.additional.import.list	JSP コンパイル時に暗黙にインポートしたいクラス名（完全修飾名のクラス名または，「パッケージ名.*」）を指定します。
	webserver.jsp.keepgenerated	JSP ファイルから生成した Java ソースファイルを保持しておくかどうかを指定します。
	webserver.jsp.compile.backcompat	Java ソースファイルをコンパイルするときの Java 言語仕様のバージョンを指定します。 なお，Web アプリケーションによって，JSP ファイルから生成された Java ソースファイルのバージョンが異なる場合は，Web アプリケーションごとにバージョンを指定して JSP 事前コンパイルを実施してください。
	webserver.jsp.precompile.jsp_work_dir	JSP 事前コンパイルを実施する場合のコンパイル結果を出力するディレクトリを指定します。
	webserver.xml.validate	Web アプリケーションに含まれるサーブレットのバージョンが 2.3 以前の場合に，JSP 事前コンパイル実行時に TLD ファイルの検証を実施するかどうかを指定します。 なお，このパラメタは，Web アプリケーションのバージョンが 2.3 以前のときに指定します。Web アプリケーションのバージョンが 2.4 以降のときはデフォルトで検証を実施するため，このパラメタへの指定はできません。
JSP ファイルのコンパイル結果の保持	webserver.work.clean	コンパイル結果を保持するかどうかを指定します。
	webserver.work.directory	コンパイル結果の出力先（JSP 用テンポラリディレクトリ）を指定します。デフォルトの出力先を変更する場合に指定します※。デフォルトの出力先については，「2.5.6(3)　JSP コンパイル結果の出力先」を参照してください。なお，この出力先は JSP 事前コンパイルを実施しない場合の出力先となります。

注※　JSP 用テンポラリディレクトリの設定を変更する場合

JSP コンパイル結果を保持する設定で Web コンテナを稼働したあと，JSP 用テンポラリディレクトリの設定を変更した場合，変更前の JSP 用テンポラリディレクトリは削除されません。このため，手動で削除する必要があります。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバリファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

2.6　デフォルトの文字エンコーディング設定機能

この節では，デフォルトの文字エンコーディング設定機能について説明します。

アプリケーションサーバでは，Servlet 仕様に準拠した文字エンコーディングおよびアプリケーションサーバ独自の文字エンコーディングが設定できます。

この節の構成を次の表に示します。

表 2-17　この節の構成（デフォルトの文字エンコーディング設定機能）

分類	タイトル	参照先
解説	デフォルトの文字エンコーディングの設定単位	2.6.1
	デフォルトの文字エンコーディングの適用個所と適用条件	2.6.2
	JSP 事前コンパイル実行時の文字エンコーディングの適用	2.6.3
	設定できる文字エンコーディング	2.6.4
実装	デフォルトの文字エンコーディングの実装（Servlet 仕様の場合）	2.6.5
	DD での定義	2.6.6
設定	実行環境での設定	2.6.7
注意事項	デフォルトの文字エンコーディングの注意事項	2.6.8

注　「運用」について，この機能固有の説明はありません。

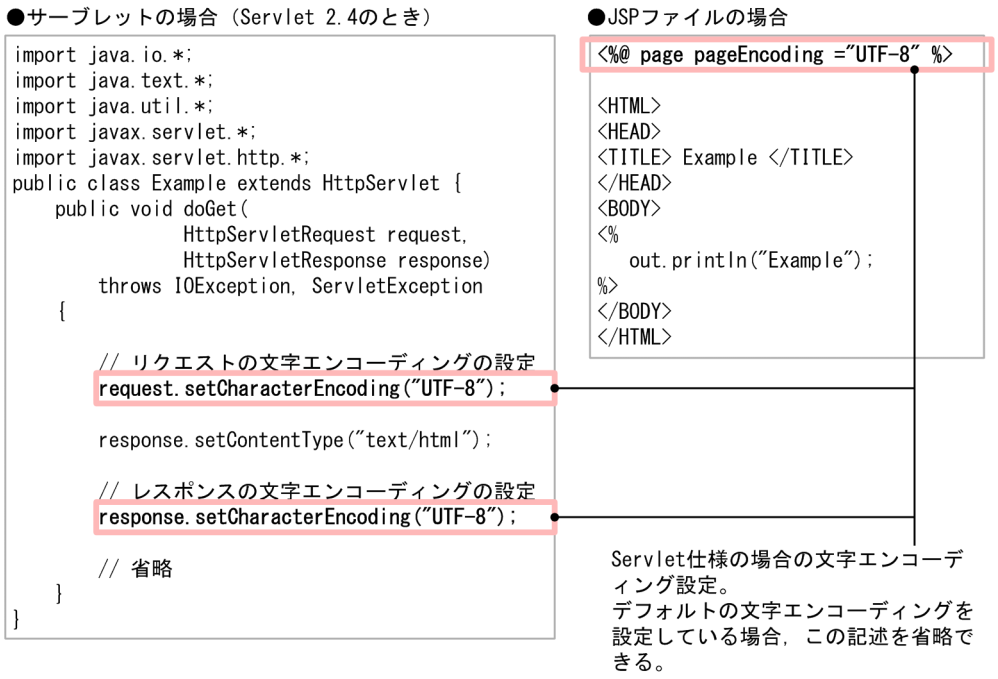
リクエストのデコードやレスポンスのエンコードで使用する文字エンコーディングは，Servlet 仕様に準拠した設定をする場合，サーブレットや JSP ファイルごとに設定を記述します。アプリケーションサーバでは，Servlet 仕様に準拠した設定のほかに，次の二つの方法でデフォルトの文字エンコーディングを設定できます。

- J2EE サーバごとに使用する文字エンコーディングを設定する
- Web アプリケーションごとに使用する文字エンコーディングを設定する

これによって，Web アプリケーション開発時に，サーブレットや JSP ファイルに記述する文字エンコーディングの設定を省略できます。また，J2EE サーバまたは Web アプリケーションごとに，容易に文字エンコーディングを統一できます。

省略できる Servlet 仕様での文字エンコーディング設定を，次の図に示します。

図 2-9　省略できる Servlet 仕様での文字エンコーディングの設定



この節では，デフォルトの文字エンコーディング設定について説明します。

2.6.1　デフォルトの文字エンコーディングの設定単位

アプリケーションサーバでは，リクエストのデコード，レスポンスのエンコード，および JSP ファイルで使用するデフォルトの文字エンコーディングを，J2EE サーバ単位，Web アプリケーション単位で設定できます。

ここでは，デフォルトの文字エンコーディングの設定について説明します。なお，デフォルトの文字エンコーディングは，JSP 事前コンパイル実行時に適用することもできます。JSP 事前コンパイル実行時のデフォルトの文字エンコーディング設定については，「2.6.3　JSP 事前コンパイル実行時の文字エンコーディングの適用」を参照してください。

(1)　J2EE サーバ単位の設定

J2EE サーバごとにデフォルトの文字エンコーディングを設定します。J2EE サーバごとにデフォルトの文字エンコーディングを設定すると，J2EE サーバにデプロイされているすべての J2EE アプリケーションのサーブレットおよび JSP ファイルに対して，設定した文字エンコーディングを適用します。これによって，J2EE サーバ単位で，文字エンコーディングを統一できます。

J2EE サーバ単位の場合，デフォルトの文字エンコーディングは，J2EE サーバの動作設定をカスタマイズする際に設定します。設定の詳細については，「2.6.7　実行環境での設定」を参照してください。

(2)　Web アプリケーション単位の設定

WAR ファイルごとに，デフォルトの文字エンコーディングを設定します。WAR ファイルごとにデフォルトの文字エンコーディングを設定すると，WAR ファイルに含まれるサーブレットおよび JSP ファイルに対して，設定した文字エンコーディングを適用します。これによって，Web アプリケーション単位で，文字エンコーディングを統一できます。

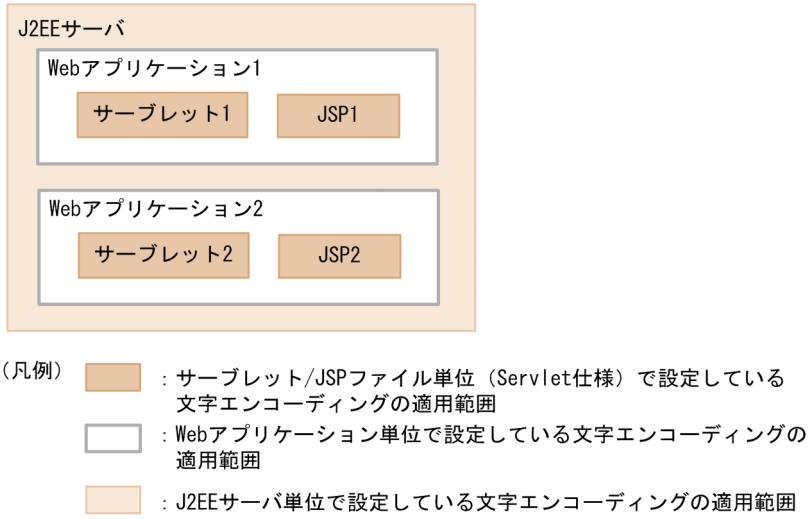
Web アプリケーション単位での設定の場合，デフォルトの文字エンコーディングは，J2EE アプリケーションのプロパティを定義する際に設定します。設定の詳細については，「2.6.7　実行環境での設定」を参照してください。

(3)　複数の範囲で文字エンコーディングを設定している場合の動作

文字エンコーディングの設定は，J2EE サーバ単位または Web アプリケーション単位での設定のほかに，Servlet 仕様での文字エンコーディングの設定もできます。Servlet 仕様での文字エンコーディング設定の場合，設定範囲は，サーブレットまたは JSP ファイル単位になります。

それぞれの設定範囲を次の図に示します。

図 2-10　文字エンコーディングの設定範囲



デフォルトの文字エンコーディングは複数範囲で設定することもできます。複数の範囲で設定されている場合，次の順序で適用されます。

1. サーブレット/JSP ファイル単位での設定（Servlet 仕様）
2. Web アプリケーション単位での設定
3. J2EE サーバ単位での設定

例えば，図 2-10 で，サーブレット 2 と J2EE サーバに文字エンコーディングが設定されているとします。この場合，J2EE サーバ内のアプリケーション全体に，J2EE サーバ単位で設定したデフォルトの文字エンコーディングが適用されますが，サーブレット 2 だけは，サーブレット内に設定した文字エンコーディングが適用されます。

次の表に，文字エンコーディング設定の組み合わせごとに，有効になる設定を示します。

表 2-18　文字エンコーディング設定の組み合わせと有効になる設定

設定の組み合わせ			有効になる設定
サーブレット/JSP ファイルでの設定（Servlet 仕様）※	Web アプリケーション単位での設定	J2EE サーバ単位での設定	
○	×	×	サーブレット/JSP ファイルでの設定

設定の組み合わせ			有効になる設定
サーブレット/JSP ファイルでの設定（Servlet 仕様）※	Web アプリケーション単位での設定	J2EE サーバ単位での設定	
○	○	×	サーブレット/JSP ファイルでの設定
○	×	○	
○	○	○	
×	○	×	Web アプリケーション単位での設定
×	○	○	
×	×	○	J2EE サーバ単位での設定
×	×	×	Servlet 仕様で規定されている文字エンコーディング

（凡例）○：設定あり　×：設定なし

注

文字エンコーディングの設定がない場合は，Servlet 仕様で規定されている文字エンコーディングが適用されます。詳細については，「2.6.5(2)　Servlet 仕様で規定されている文字エンコーディング」を参照してください。

注※

XML シンタックスの JSP ファイルまたはタグファイルでは，XML 宣言で encoding 属性を指定していない場合はデフォルトエンコーディング設定機能が有効となります。

2.6.2　デフォルトの文字エンコーディングの適用個所と適用条件

ここでは，J2EE サーバ単位，または Web アプリケーション単位で設定したデフォルトの文字エンコーディングの適用個所と，適用時の条件について説明します。

(1)　適用個所

J2EE サーバ単位または Web アプリケーション単位に設定したデフォルトの文字エンコーディングは，次の個所に適用されます。

- **リクエストのデコード**
  リクエストボディおよびクエリのデコードに使用する，デフォルトの文字エンコーディングに適用されます。
- **レスポンスのエンコード**
  レスポンスボディおよびレスポンスの Content-Type ヘッダのエンコードに使用する，デフォルトの文字エンコーディングに適用されます。
- **JSP ファイル**
  JSP ファイルのデフォルトの文字エンコーディングに適用されます。

(2)　適用条件

デフォルトの文字エンコーディングは，Servlet 仕様での文字エンコーディングが設定されていない場合に適用されます。Servlet 仕様での文字エンコーディングの設定方法については，「2.6.5　デフォルトの文字エンコーディングの実装（Servlet 仕様の場合）」を参照してください。

また，リクエストのデコードおよびレスポンスのエンコードで使用する文字エンコーディングについては，さらに次に示す適用条件があります。

(a)　リクエストでの適用条件

リクエストのデコードで使用する文字エンコーディングの場合，次に示す適用条件があります。

- クライアントから送信されたリクエストの HTTP リクエストヘッダに，charset パラメタを含む，Content-Type ヘッダが含まれていない。

さらに，リクエストボディおよびクエリには，次に示す適用条件があります。

● リクエストボディ

サーブレットの場合

リクエストの POST データを，次のどちらかの方法で読み込んでいる場合に適用されます。

- javax.servlet.ServletRequest の getReader メソッドを使用して取得した BufferedReader で読み込む。
- リクエストのパラメタとして読み込む。

リクエストのパラメタとして読み込む場合，javax.servlet.ServletRequest の getParameter メソッド，getParameterMap メソッド，getParameterName メソッド，getParameterValues メソッドを使用します。

JSP ファイルの場合

リクエストの POST データを，次のどちらかの方法で読み込んでいる場合に適用されます。

- 暗黙オブジェクト request の getReader メソッドを使用して取得した BufferedReader で読み込む。
- リクエストのパラメタとして読み込む。

リクエストのパラメタとして読み込む場合，暗黙オブジェクト request の getParameter メソッド，getParameterMap メソッド，getParameterName メソッド，getParameterValues メソッドを使用，または Expression Language で暗黙オブジェクト param，paramValues を使用します。

● クエリ

サーブレットの場合

javax.servlet.ServletRequest の getParameter メソッド，getParameterMap メソッド，getParameterName メソッド，getParameterValues メソッドを使用する方法で，クエリをリクエストのパラメタとして読み込んでいる場合に，適用されます。

JSP ファイルの場合

次のどちらかの方法で，クエリをリクエストのパラメタとして読み込んでいる場合に適用されます。

- 暗黙オブジェクト request の getParameter メソッド，getParameterMap メソッド，getParameterName メソッド，getParameterValues メソッドを使用して読み込む。
- Expression Language で暗黙オブジェクト param，paramValues を使用して読み込む。

(b)　レスポンスでの適用条件

レスポンスのエンコードで使用する文字エンコーディングの適用条件を，レスポンスボディとレスポンスの Content-Type ヘッダの文字エンコーディング名に分けて説明します。

● レスポンスボディ

サーブレットの場合

javax.servlet.ServletResponse の getWriter メソッドで取得した PrintWriter でレスポンスデータを作成している場合に適用されます。

JSP ファイルの場合

暗黙オブジェクト response の getOutputStream()メソッドによって ServletOutputStream オブジェクトを取得しないで，レスポンスを出力している場合に適用されます。※

注※

ServletOutputStream オブジェクトを取得した場合, JSP 本文のテキストや, 暗黙オブジェクト out への出力などの ServletOutputStream オブジェクトを使用しない出力は, すべて実行時にエラーとなります。このため，レスポンスとして出力することはできません。

● レスポンスの Content-Type ヘッダの文字エンコーディング名

サーブレットの場合

レスポンスのコンテンツ形式に「text/」で始まる MIME タイプを設定し，charset は設定していない場合に適用されます。

JSP ファイルの場合

次のどちらかの場合に適用されます。

- レスポンスのコンテンツ形式を設定していない。
- 「text/」で始まる MIME タイプを設定し，charset は設定していない。

静的コンテンツの場合

次の条件を満たした場合に適用されます。

- 静的コンテンツの拡張子が，デフォルトエンコーディング設定機能の対象として設定された拡張子である。
- 拡張子が「text/」で始まる MIME タイプに設定されている。
- 静的コンテンツを出力するより前に，サーブレット，JSP，フィルタなどでレスポンスに対して文字エンコーディングを設定しない。

参考

コンテンツ形式とは，コンテンツの MIME タイプを指します。コンテンツ形式には，文字エンコーディングを含めることができます。次に，サーブレットおよび JSP ファイルでのコンテンツ形式の設定例を示します。

- サーブレットの場合
  javax.servlet.ServletResponse の setContentType メソッドを使用します。
  設定例：response.setContentType("text/html");
- JSP ファイルの場合
  Page ディレクティブの contentType 属性を設定します。
  設定例：<%@ page contentType="text/plain" %>

また，デフォルトの文字エンコーディング設定が適用される MIME タイプの例と，適用されない MIME タイプの例を示します。

- 適用される MIME タイプの例：「text/plain」，「text/html」
- 適用されない MIME タイプの例：「image/gif」，「text/html;charset=UTF-8」

2.6.3　JSP 事前コンパイル実行時の文字エンコーディングの適用

JSP ファイルのデフォルトの文字エンコーディングについては，JSP 事前コンパイル実行時に適用することもできます。JSP 事前コンパイル時に適用させる場合，デフォルトの文字エンコーディング設定は，JSP 事前コンパイルの方法によって異なります。JSP 事前コンパイルの方法には，次の 2 種類があります。

- アプリケーション開発時に実施する JSP 事前コンパイル
- J2EE アプリケーション開始時に実施する JSP 事前コンパイル

JSP 事前コンパイルの種類ごとに，デフォルトの文字エンコーディング設定について説明します。JSP 事前コンパイル機能の詳細については，「2.5　JSP 事前コンパイル機能とコンパイル結果の保持」を参照してください。

なお，設定できる文字エンコーディングについては，「2.6.4　設定できる文字エンコーディング」を，設定したデフォルトの文字エンコーディングの適用個所および適用条件については，「2.6.2　デフォルトの文字エンコーディングの適用個所と適用条件」を参照してください。

(1)　アプリケーション開発時に実施する JSP 事前コンパイル（cjjspc コマンド）

cjjspc コマンドで JSP 事前コンパイルを実施する際，JSP ファイルまたはタグファイルに，デフォルトの文字エンコーディングを適用できます。cjjspc コマンドの場合，デフォルトの文字エンコーディングは，cjjspc コマンドの引数に指定します。これによって，cjjspc コマンド実行時に，実行対象となる JSP ファイルに対して，コマンドに指定したデフォルトの文字エンコーディングが適用されます。ただし，JSP ファイルやタグファイルに Servlet 仕様での文字エンコーディング指定がある場合は，設定は適用されないので注意してください。

設定の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル）」を参照してください。

! 注意事項

cjjspc コマンドの場合，リクエストのデコードおよびレスポンスのエンコードに使用するデフォルトの文字エンコーディングは，JSP ファイルのコンパイル時に適用されないため，cjjspc コマンドでは設定できません。

(2)　J2EE アプリケーション開始時に実施する JSP 事前コンパイル（cjstartapp コマンド）

cjstartapp コマンドで J2EE アプリケーション開始とあわせて JSP 事前コンパイルを実施する際，JSP ファイルまたはタグファイルに，デフォルトの文字エンコーディングを適用できます。デフォルトの文字エンコーディングの設定方法は，J2EE サーバ単位，Web アプリケーション単位の場合と同様です。J2EE サーバ単位の場合は，J2EE サーバの動作設定時に J2EE サーバごとに設定します。また，Web アプリケーション単位の場合は，Web アプリケーション開発時に WAR ファイルごとに設定します。cjstartapp コマンドを実行すると，設定したデフォルトの文字エンコーディングが適用されます。

J2EE サーバ単位の設定および Web アプリケーション単位での設定については，「2.6.1　デフォルトの文字エンコーディングの設定単位」を参照してください。また，設定の詳細については，「2.6.7　実行環境での設定」を参照してください。

2.6.4　設定できる文字エンコーディング

デフォルトの文字エンコーディングとして設定できる文字は，JavaVM がサポートしている文字エンコーディングとなります。JavaVM がサポートしている文字エンコーディングについては，JDK のドキュメントのサポートされているエンコーディングに関する説明を参照してください。

また，指定できる文字列は，java.nio API 用の正準名と java.lang API 用の正準名に記載されている文字エンコーディング，およびそれらの別名になります。

! 注意事項

J2EE アプリケーションの開発環境と運用環境の OS が異なる場合で，J2EE アプリケーションを，実行時情報を含む EAR ファイルでエクスポートおよびインポートするときは，開発環境の OS と運用環境の OS の両方でサポートしている文字エンコーディングを設定してください。開発環境で設定した文字エンコーディングが運用環境でサポートされていない場合は，アプリケーション開始時に例外が発生することがあります。

また，設定したデフォルトの文字エンコーディングは，JavaVM でサポートしている文字エンコーディングかどうか検証されます。検証のタイミングは，デフォルトの文字エンコーディングの設定方法によって異なります。文字エンコーディングの検証のタイミングについて，次の表に示します。

表 2-19　文字エンコーディングの検証のタイミング

検証のタイミング	サポートされていない文字エンコーディングが設定されていた場合の動作
J2EE サーバ開始時	警告メッセージを出力し，J2EE サーバ開始処理を続行します。なお，設定した文字エンコーディングは無視されます。
サーバ管理コマンド（cjsetappprop）実行時	エラーメッセージを出力し，サーバ管理コマンドの処理を中止します。
cjjspc コマンド実行時※	エラーメッセージを出力し，コマンドの処理を終了します。

注※

cjjspc コマンドでの JSP 事前コンパイル実行時に，デフォルトの文字エンコーディング設定をするときの検証のタイミングです。なお，JSP 事前コンパイルでのデフォルトの文字エンコーディングの適用については，「2.6.3　JSP 事前コンパイル実行時の文字エンコーディングの適用」を参照してください。

2.6.5　デフォルトの文字エンコーディングの実装（Servlet 仕様の場合）

Servlet 仕様で規定された文字エンコーディングの設定がある個所で，アプリケーションサーバで設定したデフォルトの文字エンコーディングは無効となります。

ここでは，Servlet 仕様で規定された文字エンコーディングの設定について説明します。なお，文字エンコーディングの設定は，Servlet 仕様のバージョンによって異なります。

(1)　Servlet 仕様での文字エンコーディングの設定方法

Servlet 仕様での文字エンコーディングの設定方法について，Servlet/JSP のバージョンごとに表に示します。

表 2-20　Servlet 仕様での文字エンコーディングの設定方法（Servlet 2.5，3.0/JSP 2.1）

設定内容	設定場所	Servlet 仕様での設定方法
リクエストの文字エンコーディング	サーブレット	ServletRequest.setCharacterEncoding(java.lang.String env)※1
	JSP ファイル	なし
レスポンスの文字エンコーディング	サーブレット	• ServletResponse.setCharacterEncoding(java.lang.String charset)※1 • ServletResponse.setContentType(java.lang.String type)※1 • ServletResponse.setLocale(java.util.Locale loc)※1

設定内容	設定場所	Servlet 仕様での設定方法
レスポンスの文字エンコーディング	JSP ファイル	• Page ディレクティブの contentType 属性値(charset を含む)※2 • Page ディレクティブの pageEncoding 属性※3 • web.xml の page-encoding 要素※2
JSP ファイルの文字エンコーディング	JSP ファイル	• BOM※3 • Page ディレクティブの contentType 属性値（charset を含む）※2 • Page ディレクティブまたは Tag ディレクティブの pageEncoding 属性※3 • web.xml の page-encoding 要素※2 • XML 宣言の encoding 属性※4

注※1　パッケージは javax.servlet です。
注※2　JSP ページに設定する方法です。
注※3　JSP ページまたは標準形式のタグファイルに設定する方法です。
注※4　JSP ドキュメントまたは XML 形式のタグファイルに設定する方法です。

表 2-21　Servlet 仕様での文字エンコーディングの設定方法（Servlet 2.4/JSP 2.0）

設定内容	設定場所	Servlet 仕様での設定方法
リクエストの文字エンコーディング	サーブレット	ServletRequest.setCharacterEncoding(java.lang.String env)※1
	JSP ファイル	なし
レスポンスの文字エンコーディング	サーブレット	• ServletResponse.setCharacterEncoding(java.lang.String charset)※1 • ServletResponse.setContentType(java.lang.String type)※1 • ServletResponse.setLocale(java.util.Locale loc)※1
	JSP ファイル	• Page ディレクティブの contentType 属性値（charset を含む）※2 • Page ディレクティブの pageEncoding 属性※3 • web.xml の page-encoding 要素※2
JSP ファイルの文字エンコーディング	JSP ファイル	• Page ディレクティブの contentType 属性値（charset を含む）※2 • Page ディレクティブまたは Tag ディレクティブの pageEncoding 属性※3 • web.xml の page-encoding 要素※2 • XML 宣言の encoding 属性※4

注※1　パッケージは javax.servlet です。
注※2　JSP ページに設定する方法です。
注※3　JSP ページまたは標準形式のタグファイルに設定する方法です。
注※4　JSP ドキュメントまたは XML 形式のタグファイルに設定する方法です。

表 2-22　Servlet 仕様での文字エンコーディングの設定方法（Servlet 2.3/JSP 1.2）

設定内容	設定場所	Servlet 仕様での設定方法
リクエストの文字エンコーディング	サーブレット	ServletRequest.setCharacterEncoding(java.lang.String env) ※1
	JSP ファイル	なし

設定内容	設定場所	Servlet 仕様での設定方法
レスポンスの文字エンコーディング	サーブレット	• ServletResponse.setContentType(java.lang.String type)※1 • ServletResponse.setLocale(java.util.Locale loc)※1
	JSP ファイル	Page ディレクティブの contentType 属性値（charset を含む）
JSP ファイルの文字エンコーディング	JSP ファイル	• Page ディレクティブの contentType 属性値（charset を含む）※2 • Page ディレクティブの pageEncoding 属性※2

注※1　パッケージは javax.servlet です。

注※2　JSP ページまたは JSP ドキュメントに設定する方法です。

(2)　Servlet 仕様で規定されている文字エンコーディング

Servlet 仕様での文字エンコーディング設定，およびアプリケーションサーバでのデフォルトの文字エンコーディングの設定がない場合は，Servlet 仕様で規定されている文字エンコーディングが適用されます。

文字エンコーディングを設定していない場合に適用される，Servlet 仕様で規定された文字エンコーディングを次に示します。

- リクエストの場合

  ISO-8859-1 が適用されます。なお，サーブレットおよび JSP ファイルでは，Servlet API を使用して設定されます。

- レスポンスの場合

  Servlet 仕様で規定された文字エンコーディングを，Servlet のバージョンごとに次の表に示します。

表 2-23　Servlet 仕様で規定された文字エンコーディング（レスポンス）

Servlet のバージョン	種類	適用される文字エンコーディング
Servlet 2.3	サーブレット	ISO-8859-1
	JSP ページ	
	JSP ドキュメント	
Servlet 2.4 以降	サーブレット	ISO-8859-1
	JSP ページ	
	JSP ドキュメント	UTF-8

- JSP ファイルの場合

  Servlet 仕様で規定された文字エンコーディングを，Servlet のバージョンごとに次の表に示します。

表 2-24　Servlet 仕様で規定された文字エンコーディング（JSP ファイル）

Servlet のバージョン	JSP のバージョン	種類	適用される文字エンコーディング
Servlet 2.3	JSP 1.2	JSP ページ	ISO-8859-1
		JSP ドキュメント	
Servlet 2.4 以降	JSP 2.0，2.1	JSP ページ	ISO-8859-1

Servlet のバージョン	JSP のバージョン	種類	適用される文字エンコーディング
Servlet 2.4 以降	JSP 2.0，2.1	標準形式のタグファイル	ISO-8859-1
		JSP ドキュメント	UTF-8
		XML 形式のタグファイル	

2.6.6　DD での定義

デフォルトの文字エンコーディングの Web アプリケーション単位の定義は，web.xml に指定します。

DD でのデフォルトの文字エンコーディングの定義について次の表に示します。

表 2-25　DD でのデフォルトの文字エンコーディングの定義

指定するタグ	設定内容
<http-request>－<encoding>タグ	リクエストボディおよびクエリのデコードに使用する，文字エンコーディングを設定します。
<http-response>－<encoding>タグ	レスポンスボディのエンコードに使用する，文字エンコーディングを設定します。
<jsp>－<page-encoding>タグ	JSP ファイルの文字エンコーディングを設定します。

注　Web アプリケーション内のサーブレットまたは JSP ファイルに，Servlet 仕様での文字エンコーディング設定がある場合は，Servlet 仕様での設定が有効になります。設定の優先順位については，「2.6.1(3)　複数の範囲で文字エンコーディングを設定している場合の動作」を参照してください。

2.6.7　実行環境での設定

デフォルトの文字エンコーディングを設定する場合，J2EE サーバおよび Web アプリケーションの設定が必要です。

なお，Web アプリケーションの設定は，cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

(1)　J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。デフォルトの文字エンコーディングの定義の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのデフォルトの文字エンコーディングの定義の定義について次の表に示します。

表 2-26　簡易構築定義ファイルでのデフォルトの文字エンコーディングの定義

指定するパラメタ	設定内容
webserver.http.request.encoding	リクエストボディおよびクエリのデコードに使用する文字エンコーディングを指定します。

指定するパラメタ	設定内容
webserver.http.response.encoding	レスポンスボディのエンコードに使用する文字エンコーディングを指定します。
webserver.jsp.pageEncoding	JSP ファイルのエンコーディングを指定します。
webserver.static_content.encoding.extension	デフォルトのレスポンス文字エンコーディングを適用する静的コンテンツの拡張子を指定します。

注　J2EE サーバ単位の設定とあわせて，Web アプリケーション単位での文字エンコーディング設定がある場合は，Web アプリケーション単位での設定が有効になります。設定の優先順位については，「2.6.1(3)　複数の範囲で文字エンコーディングを設定している場合の動作」を参照してください。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　Web アプリケーションの設定

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。デフォルトの文字エンコーディングの定義には，WAR 属性ファイルを使用します。

WAR 属性ファイルで指定するタグは，DD と対応しています。DD（web.xml）での定義については，「2.6.6　DD での定義」を参照してください。

2.6.8　デフォルトの文字エンコーディングの注意事項

デフォルトの文字エンコーディングの適用については，次の点に注意してください。

(1)　レスポンスへのデフォルトの文字エンコーディングの設定の可否について

次の場合は，レスポンスへのデフォルトの文字エンコーディングの設定は有効になりません。

- 静的コンテンツの拡張子が，webserver.static_content.encoding.extension パラメタ※1 に指定した拡張子以外の場合
- Servlet 仕様で規定されたエラーページの静的コンテンツに対して，レスポンスへのデフォルトの文字エンコーディングを設定しない場合※2

注※1

デフォルトの文字エンコーディングを適用させる静的コンテンツの拡張子を指定するためのパラメタです。

注※2

次のどちらかの条件のあとに出力された静的コンテンツの場合は，レスポンスへのデフォルトの文字エンコーディングの設定が有効になります。

- レスポンスに対して文字エンコーディングを設定しない場合で，サーブレット，JSP，フィルタなどで，javax.servlet.ServletResponse クラスの getWriter メソッドによって java.io.PrintWriter オブジェクトが取得されるとき。
- setAttribute メソッドが実行されても，リクエストオブジェクトがラップされる場合で，そのリクエストオブジェクトが，setAttribute メソッドを呼び出さないリクエストラッパでラップされるとき。

なお，HTTP レスポンス圧縮フィルタを使用している場合は，レスポンスへのデフォルトの文字エンコーディングの設定は有効になりません。

(2)　getCharacterEncoding メソッドに適用される文字エンコーディング

Servlet 仕様で文字エンコーディングを設定していない場合，次に示す Servlet API のメソッドには，J2EE サーバ単位または Web アプリケーション単位で設定したデフォルトの文字エンコーディングが適用されます。

- javax.servlet.ServletRequest の getCharacterEncoding メソッド（リクエストの場合）
- javax.servlet.ServletResponse の getCharacterEncoding メソッド（レスポンスの場合）

ただし，setCharacterEncoding メソッドで文字エンコーディングを変更しているときは，setCharacterEncoding メソッドで変更した文字エンコーディングが取得されます。

また，レスポンスの場合，javax.servlet.ServletResponse の reset メソッドを使用してレスポンスデータを初期化したときは，getCharacterEncoding メソッドで取得できる文字エンコーディングは，アプリケーションサーバで設定した文字エンコーディングとなります。

なお，Servlet 仕様での文字エンコーディングの設定方法については，「2.6.5　デフォルトの文字エンコーディングの実装（Servlet 仕様の場合）」を参照してください。

(3)　XML 宣言内の文字エンコーディング

JSP ドキュメントおよび XML 形式のタグファイルで，Web コンテナが XML 宣言を自動生成する場合，XML 宣言内の文字エンコーディングの宣言には，レスポンスボディのエンコードに適用されたデフォルトの文字エンコーディングが出力されます。

(4)　JSP ファイルへの文字エンコーディングの適用

JSP ファイルへの文字エンコーディング設定は，JSP ファイルのコンパイル時に適用されます。このため，すでに JSP ファイルがコンパイルされている状態で，文字エンコーディングの設定を追加または変更しても，追加または変更した文字エンコーディングは JSP ファイルに反映されません。設定を反映させるためには，再度，コンパイルを実施してください。

(5)　07-00 で JSP 事前コンパイルを実行した Web アプリケーションへの文字エンコーディング設定

07-00 で JSP 事前コンパイルを実行した Web アプリケーションに対して，レスポンスへのデフォルトの文字エンコーディングを設定する場合は，再度，JSP 事前コンパイルを実施してください。

デフォルトの文字エンコーディングの設定単位ごとに説明します。

- J2EE サーバ単位での設定の場合
  J2EE サーバ上で動作するすべての Web アプリケーションの JSP ファイルに対して，JSP 事前コンパイルを実施します。
- Web アプリケーション単位での設定の場合
  Web アプリケーションに含まれる JSP ファイルに対して，JSP 事前コンパイルを実施します。

なお，JSP 事前コンパイル実行時でのデフォルトの文字エンコーディング設定については，「2.6.3　JSP 事前コンパイル実行時の文字エンコーディングの適用」を参照してください。

(6)　デフォルトエラーページの文字エンコーディングについて

デフォルトエラーページの文字エンコーディングは UTF-8 に設定されているため，デフォルトエンコーディングは適用されません。

2.7　セッション管理機能

この節では，セッション管理機能について説明します。

セッション管理は，リクエストと Web クライアントを対応づけるための機能です。この機能を使用すると，Web クライアントから，複数の Web ページにわたって同じ情報を引き継いだ作業ができるようになります。

この節の構成を次の表に示します。

表 2–27　この節の構成（セッション管理機能）

分類	タイトル	参照先
解説	セッション情報を管理するオブジェクト	2.7.1
	セッション ID の形式	2.7.2
	セッションの管理方法	2.7.3
	Web クライアントが保持する無効なセッション ID の削除	2.7.4
	HttpSession オブジェクト数の上限値の設定	2.7.5
	セッション ID および Cookie へのサーバ ID の付加	2.7.6
実装	cosminexus.xml での定義	2.7.7
設定	実行環境での設定	2.7.8
注意事項	セッション管理の注意事項	2.7.9

注　「運用」について，この機能固有の説明はありません。

セッション管理機能は，次に示す場合にも利用されます。

- クラスタリングによる負荷分散機能を使用するときのクライアントの識別
- セキュリティ管理機能でのログイン済みクライアントの識別

セッション管理の方式として，Servlet 仕様では，Cookie を使用する方法と URL 書き換えを使用する方法が明記されています。アプリケーションサーバの Web コンテナでも，これらの方式によってセッションを管理します。ただし，Servlet 仕様では，具体的な管理方式を明記していない部分があります。このため，この節では，Servlet 仕様で明記されていないセッションの管理方式について，アプリケーションサーバでどのように管理するかを説明します。

また，アプリケーションサーバの独自の機能である，次の 3 種類のセッション管理機能についても，この節で説明します。

- Web クライアントが保持する無効なセッション ID の削除
- HttpSession オブジェクトの上限値の設定
- セッション ID および Cookie へのサーバ ID の付加

ポイント

Web クライアントが保持する無効なセッション ID の削除，HttpSession オブジェクトの上限値の設定，およびセッション ID へのサーバ ID の付加は，アプリケーションサーバのセッションフェイルオーバ機能を使

用する場合に前提になる機能です。セッションフェイルオーバ機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「5. J2EE サーバ間のセッション情報の引き継ぎ」を参照してください。

2.7.1 セッション情報を管理するオブジェクト

ここでは，セッション情報の管理に使用する HttpSession オブジェクトについて説明します。

(1) HttpSession オブジェクトの管理方法

セッション情報は，Servlet API で規定されている **HttpSession オブジェクト**によって管理される情報です。

セッション情報の管理が開始されるのは，次の時点です。

- サーブレットの場合は，HttpSession オブジェクトを参照した時点。
- JSP の場合は，ページへの参照が発生した時点（ただし，これはデフォルトの場合です）。

セッション情報の管理が開始されたあとで，同一の Web アプリケーション内のサーブレットに対して，同じブラウザプロセスからリクエストが送信されると，管理されている内容の HttpSession オブジェクトがサーブレットに渡されるようになります。

ただし，実際にサーブレットに渡される HttpSession オブジェクトのインスタンスは，リクエスト単位に異なります。つまり，同一セッションに属する一連のリクエストでは，内容が同じでインスタンスが異なる HttpSession オブジェクトが渡される場合があります。

このため，HttpSession オブジェクトに対する操作では，次の点に留意してください。

- HttpSession オブジェクトにアクセスする場合，リクエスト単位にインスタンスを取得する必要があります。
- HttpSession オブジェクトへの参照を，複数のリクエストにわたってキャッシングしないでください。
- HttpSession オブジェクトに対して，java の synchronized キーワードでロックを掛けても意味はありませんので，ロックは掛けないでください。

(2) HttpSession オブジェクトの保存期間

HttpSession オブジェクトは単一の JavaVM 内にだけ保存されています。このため，サーブレットエンジンとして動作している JavaVM プロセス（J2EE サーバ）に障害が発生した場合には，HttpSession オブジェクトが保持しているセッション情報は失われます。

また，セッション情報は，正常・異常に関係なく J2EE サーバが終了すると失われます。

J2EE サーバが終了したあとにもセッション情報を保持しておきたい場合は，アプリケーションサーバの機能であるセッションフェイルオーバ機能を使用してください。セッションフェイルオーバ機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「5. J2EE サーバ間のセッション情報の引き継ぎ」を参照してください。

2.7.2 セッション ID の形式

ここでは，セッション情報の識別に使用する**セッション ID** の形式について説明します。HttpSession オブジェクトは，セッション ID によって識別されます。

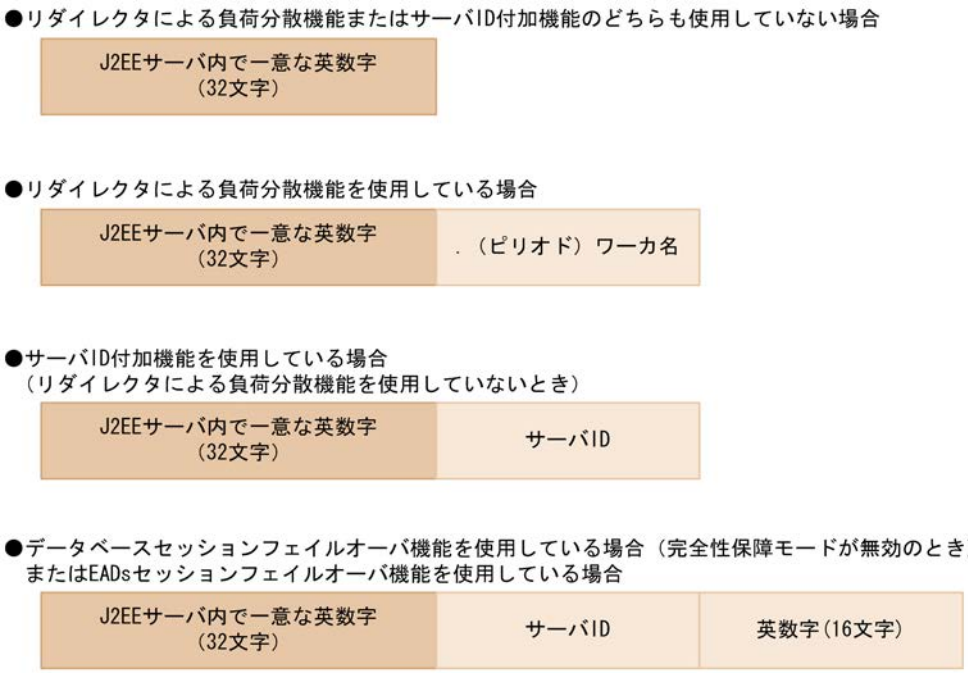
セッション ID の形式は，次の機能を使用しているかどうかで異なります。

- リダイレクタによる負荷分散機能
- セッション ID へのサーバ ID の付加機能

セッション ID は，J2EE サーバ内で一意であることが保証されています。ただし，複数の J2EE サーバにわたって一意の値にする必要がある場合には，セッション ID にワーカ名またはサーバ ID を追加して，複数の J2EE サーバにわたって一意になるように設定する必要があります。

使用している機能ごとのセッション ID の形式を，次の図に示します。

図 2-11　セッション ID の形式



それぞれの場合について説明します。

- **リダイレクタによる負荷分散機能とサーバ ID 付加機能のどちらも使用していない場合**

  セッション ID は，32 文字の英数字になります。この英数字は，一つの J2EE サーバ内で一意な値になります。

- **リダイレクタによる負荷分散機能を使用している場合**

  セッション ID は，32 文字の英数字，「.」（ピリオド），およびワーカ名で構成されます。ワーカ名には，サーバごとに一意の名称を設定しておく必要があります。

  リダイレクタによる負荷分散機能については，「4.2　Web サーバ（リダイレクタ）によるリクエストの振り分け」のロードバランサを使用する場合の説明を参照してください。

- **サーバ ID 付加機能を使用している場合（リダイレクタによる負荷分散機能を使用していないとき）**

  セッション ID は，32 文字の英数字およびサーバ ID で構成されます。サーバ ID には，サーバごとに一意の値を設定しておく必要があります。

  サーバ ID 付加機能については，「2.7.6　セッション ID および Cookie へのサーバ ID の付加」を参照してください。

  なお，リダイレクタによる負荷分散機能を使用している場合，サーバ ID は付加されません。

- **データベースセッションフェイルオーバ機能を使用している場合（完全性保障モードが無効のとき）または EADs セッションフェイルオーバ機能を使用している場合**

セッション ID は，32 文字の英数字，サーバ ID および 16 文字の英数字で構成されます。データベースセッションフェイルオーバ機能（完全性保障モードが無効のとき）および EADs セッションフェイルオーバ機能を使用する場合は，サーバ ID 付加機能によるセッション ID へのサーバ ID 付加が前提となります。

また，サーバ ID には，サーバごとに一意の値を設定しておく必要があります。

サーバ ID 付加機能については，「2.7.6　セッション ID および Cookie へのサーバ ID の付加」を参照してください。

データベースセッションフェイルオーバ機能および EADs セッションフェイルオーバ機能の前提となる設定については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「5.4.2　前提となる設定」を参照してください。

2.7.3　セッションの管理方法

ここでは，セッションの管理方法とセッション ID の管理について説明します。

(1)　セッションの管理方法

トラッキングモードで Web コンテナのセッションの管理方法を指定します。トラッキングモードには，HTTP Cookie を使用する方法と，URL 書き換えを使用する方法の 2 種類があります。トラッキングモードは，そのどちらか一方，または両方を選択できます。

- HTTP Cookie だけを使用する場合

  HTTP Cookie によるセッション管理だけが有効になります。このとき，URL 書き換えで生成される文字列にセッション ID を示す URL パスパラメタは含まれません。

- URL 書き換えだけを使用する場合

  URL 書き換えによるセッション管理だけが有効になります。このとき，レスポンスにセッション ID を示す HTTP Cookie の情報は含まれません。

- HTTP Cookie と URL 書き換えの両方を使用する場合

  HTTP Cookie によるセッション管理と URL 書き換えによるセッション管理の両方が有効になります。Web コンテナは，セッションがどの方法で管理されているかを，セッション ID が何から取得できたかによって判別します。HTTP Cookie からセッション ID を取得した場合は，HTTP Cookie によってセッションを管理していると判別します。URL のパスパラメタから取得できた場合は，URL 書き換えによってセッションを管理していると判別します。これらの判別は，リクエストごとに実行されます。

(2)　セッションの管理に HTTP Cookie を使用する場合のセッション ID の管理

セッション ID は，HTTP Cookie として管理されます。HTTP Cookie には HttpOnly 属性を付与できます。

HTTP セッションを新規に作成した場合に，HTTP レスポンスのヘッダにセッション ID を示す HTTP Cookie が付加されます。セッション ID を示す HTTP Cookie の名称は，「JSESSIONID」です。この名称は，アプリケーションサーバのバージョンが 09-00 以降の場合，変更できます。

なお，作成した HTTP セッションをコミットする前に無効化した場合，HTTP Cookie は付加されません。

(3)　セッションの管理に URL 書き換えを使用する場合のセッション ID の管理

セッション ID は，URL のパスパラメタとして管理されます。

セッション ID を示す URL のパスパラメタの名称は，「jsessionid」です。この名称は，アプリケーションサーバのバージョンが 09-00 以降の場合，変更できます。セッション ID は，Web コンテナによって URL が書き換えられるときに，URL のパスの最後に，「;jsessionid=セッション ID」の形式で付加されます。

URL のパスは，階層構造を持っている，リソースを識別するための値です。URL のパスには，クエリやフラグメントは含まれません。このため，これらの要素が URL に含まれている場合，セッション ID は，クエリまたはフラグメントの直前に付加されます。また，URL にセッション ID 以外のパスパラメタが含まれている場合，セッション ID を示すパスパラメタは，URL に含まれるパスパラメタの最後に付加されます。

ポイント

セッション ID を URL のパスパラメタに追加する場合，URL の文字数が増加します。
増加する文字数を次の表に示します。

表 2–28　URL 書き換えによって増加する URL の文字数

機能の使用状況	増加する URL の文字数（単位：文字数）
リダイレクタによる負荷分散機能またはサーバ ID 付加機能を使用していない場合	44※
リダイレクタによる負荷分散機能を使用している場合	44※ ＋ 1（ピリオドの文字数）＋ワーカ名の文字数
サーバ ID 付加機能を使用している場合（リダイレクタによる負荷分散機能を使用していないとき）	44※ ＋ サーバ ID の文字数
データベースセッションフェイルオーバ機能（完全性保障モードが無効の場合）または EADs セッションフェイルオーバ機能を使用している場合	44※ ＋ サーバ ID の文字数 ＋ 16（英数字の文字数）

注※

「;jsessionid=」の 12 文字とセッション ID の 32 文字の合計です。URL のパスパラメタの名称を変更している場合は，次の値の合計がパスパラメタのサイズとなります。

・パスパラメタの文字数

・セミコロン（;），およびイコール（=）の文字数（2 文字）

・HTTP セッションのセッション ID の文字列長（32 文字）

2.7.4　Web クライアントが保持する無効なセッション ID の削除

アプリケーションサーバでは，Web クライアントが保持する無効なセッション ID を削除します。これによって，無効なセッション ID の Web クライアントからの送信を抑止します。

HTTP セッションを無効化した場合，または無効なセッション ID を含む HTTP セッションを受信した場合，Web コンテナによって，無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie が HTTP レスポンスのヘッダに付加されます。これによって，無効なセッション ID が削除されます。

無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie とは，次のすべての条件を満たす HTTP Cookie を指します。

- セッション ID を示す HTTP Cookie であり，名称は「JSESSIONID」である（Servlet 3.0 以降は変更できます）。
- 値が「""」（空文字列）である。

- HTTP Cookie の有効期限に，経過した期限となる正の数が設定されている。

HTTP レスポンスのヘッダに，無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie が付加されるのは，次の二つの場合です。

- HTTP セッションが無効化された。
- J2EE サーバに存在しないセッション ID を受信した。

以降でそれぞれの場合について説明します。

Web サーバ連携機能を使用する場合の注意事項

レスポンスのステータスコードが 304（Not Modified）である場合に，Web サーバの仕様で Set-Cookie ヘッダが削除されるときがあります。このとき，無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie も付加されなくなるため，Web クライアントが保持する無効なセッション ID の削除ができなくなります。

(1) HTTP セッションが無効化された場合

次の条件をすべて満たす場合，無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie が HTTP レスポンスのヘッダに付加されます。

- セッション ID が HTTP Cookie を使用して通知された。
- Web アプリケーション内で HTTP レスポンスがコミットされる前に HTTP セッションが無効化された※1。
- HTTP レスポンスのコミット時に HTTP セッションが存在しない※2。

注※1

HTTP レスポンスのヘッダは，レスポンスがコミットされた時点で Web クライアントに送信されるため，コミットしたあとのレスポンスには HTTP Cookie を付加できません。そのため，Web アプリケーション内で HTTP レスポンスがコミットされたあとで HTTP セッションが無効化された場合，HTTP Cookie 情報を削除するための HTTP Cookie は付加されません。しかし，次回リクエスト受信時に，存在しないセッション ID を受信することになるため「2.7.4(2) J2EE サーバに存在しないセッション ID を受信した場合」に該当し，HTTP Cookie 情報が削除されます。

注※2

HTTP レスポンスのコミット時に新しい HTTP セッションが作成されていた場合は，新しいセッション ID を示す HTTP Cookie で Web クライアントの HTTP Cookie 情報が上書きされるため，HTTP Cookie 情報の削除は不要になります。

なお，次の条件のどちらかを満たす場合，HTTP セッションが無効化された場合でも，無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie は付加されません。

- 簡易 Web サーバでリクエストを受信した。
- サーブレットエンジンモードを使用している。

これらの機能は旧バージョンとの互換用の機能です。

(2) J2EE サーバに存在しないセッション ID を受信した場合

次の条件をすべて満たす場合，無効なセッション ID を受信したと判断され，無効なセッション ID を表す HTTP Cookie 情報を削除するための HTTP Cookie が HTTP レスポンスのヘッダに付加されます。

- セッション ID が HTTP Cookie を使用して通知された。

- 通知されたセッション ID が J2EE サーバに存在しないセッション ID である。
- HTTP レスポンスのコミット時に HTTP セッションが存在していない。

(3)　Web クライアントが保持する無効なセッション ID の削除をする場合の注意事項

複数の J2EE サーバで同じパスのリクエストを扱う構成の場合，この機能を無効にしてください。

リバースプロキシなどで Cookie の Path が書き換えられた同じパスのリクエストを扱うと，HTTP セッションが不当に削除されるおそれがあります。

2.7.5　HttpSession オブジェクト数の上限値の設定

アプリケーションサーバでは，有効となる HttpSession オブジェクト数に，上限値を設定できます。

HttpSession オブジェクトの上限値は，J2EE アプリケーションに含まれる Web アプリケーションの属性（プロパティ）として設定します。J2EE アプリケーションの設定については，「2.7.7　cosminexus.xml での定義」を参照してください。

! 注意事項

Web アプリケーションにセッションフェイルオーバ機能を適用している場合，次の設定にしているときは，HttpSession オブジェクト数の上限値の設定が必要です。

- アプリケーション開始時のネゴシエーション処理に失敗したときに，Web アプリケーションの開始処理を中止する設定（簡易構築定義ファイルの webserver.dbsfo.negotiation.high_level キー）

この場合，HttpSession オブジェクト数の上限値を設定していないと，セッション情報の引き継ぎを行う Web アプリケーションの開始時にエラーとなり，Web アプリケーションを開始できません。アプリケーション開始時のネゴシエーション処理に失敗したときに，Web アプリケーションの開始処理を続行する設定にしている場合は，HttpSession オブジェクト数の上限値の設定は任意です。

セッションフェイルオーバ機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「5.2 セッションフェイルオーバ機能の概要」を参照してください。

セッションフェイルオーバ機能の Web アプリケーション開始時のネゴシエーション処理については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「6.4.1　アプリケーション開始時の処理」を参照してください。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

参考

グローバルセッション情報のサイズ見積もり機能を有効にする場合，セッションフェイルオーバ機能の前提条件である HttpSession オブジェクト数の上限値の指定は不要です。グローバルセッション情報のサイズ見積もり機能については，マニュアル「アプリケーションサーバ 機能解説 互換編」の「6.5　グローバルセッション情報のサイズの見積もり」を参照してください。

(1)　上限値を設定する対象になる HttpSession オブジェクト

上限値を設定する対象になるのは，有効な HttpSession オブジェクトです。

有効な HttpSession オブジェクトとは，javax.servlet.http.HttpServletRequest クラスの getSession メソッドで取得した HttpSession オブジェクトのうち，次の二つの条件を満たすオブジェクトを指します。

- タイムアウトしていないオブジェクト
- invalidate メソッドが呼ばれていないオブジェクト

(2) HttpSession オブジェクト数が上限値を超えた場合の動作（例外の発生）

HttpSession オブジェクト数の上限値を設定していると，設定した上限値を超えて HttpSession オブジェクトを生成しようとした場合に javax.servlet.http.HttpServletRequest クラスの getSession メソッドで例外が発生します。設定に応じて，次のどちらかの例外が発生します。

- java.lang.IllegalStateException
- java.lang.IllegalStateException の派生クラスである com.hitachi.software.web.session.HttpSessionLimitExceededException

どちらの例外をスローするかは，J2EE アプリケーションのパラメタで設定します。J2EE アプリケーションの設定については，「2.7.8(1) J2EE サーバの設定」を参照してください。

Web コンテナでの HttpSession オブジェクト生成は，次のタイミングで実行されます。

- JSP 実行時
  page ディレクティブの session 属性に「true」を指定した場合，または session 属性を省略している場合に，JSP を実行したときを指します。
- FORM 認証を必要とする URL へのアクセス時

設定した上限値を超えて HttpSession オブジェクトの生成処理が発生すると，次のように動作します。

- JSP 実行時に上限値を超えた場合
  JSP のユーザコードが実行される前に，java.lang.IllegalStateException または com.hitachi.software.web.session.HttpSessionLimitExceededException がスローされます。
- FORM 認証を必要とする URL へのアクセス時に上限値を超えた場合
  ログイン用ページにリクエストが転送される前に，java.lang.IllegalStateException または com.hitachi.software.web.session.HttpSessionLimitExceededException がスローされます。

ポイント

com.hitachi.software.web.session.HttpSessionLimitExceededException クラスを使用する場合は，J2EE アプリケーションの開発時に<Application Server のインストールディレクトリ>/CC/lib/ejbserver.jar をクラスパスに追加して，Java プログラムをコンパイルしてください。

(3) HttpSession オブジェクト数が上限値を超えた場合の動作（メッセージの出力）

HttpSession オブジェクト数の上限値を設定していると，設定した上限値を超えて HttpSession オブジェクトを生成しようとした場合に KDJE39225-E のメッセージがログに出力されます。

KDJE39225-E は，HTTP セッションを使用するリクエストが実行されるたびに出力されるため，同じメッセージでログが埋まってしまうおそれがあります。同じメッセージが繰り返し出力されるのを抑止するために，KDJE39225-E のインターバル（出力間隔）を設定できます。必要に応じて，インターバルを設定してください。このインターバルは Web アプリケーション単位に適用されます。

メッセージの出力インターバルの設定については，「2.7.8(1) J2EE サーバの設定」を参照してください。

2.7.6 セッション ID および Cookie へのサーバ ID の付加

アプリケーションサーバでは，HttpSession のセッション ID や Cookie に，サーバ ID を付けることができます。これを，**サーバ ID 付加機能**といいます。サーバ ID は，Web コンテナごとに異なる値を設定します。

セッション ID および Cookie へのサーバ ID の付加についての設定は，J2EE サーバのプロパティをカスタマイズして設定します。J2EE サーバの動作設定のカスタマイズについては，「2.7.8　実行環境での設定」を参照してください。

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は，セッション ID へのサーバ ID の付加は必須となります。メモリセッションフェイルオーバ機能については，マニュアル「アプリケーションサーバ 機能解説 互換編」の「6.　拡張機能の互換機能（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。

! 注意事項

この機能で設定する Cookie の名前は，Servlet または JSP で設定する Cookie の名前および Web コンテナが自動で設定する Cookie の名前と重複しないようにしてください。Web コンテナが自動で設定する名前は次の名前です。

- JSESSIONID

アプリケーションサーバのバージョンが 09-00 以降の場合，Web コンテナが設定する Cookie の名前を変更できます。Web コンテナが設定する Cookie の名前を変更した場合に Cookie へサーバ ID を付与したときの注意事項については，「2.7.9　セッション管理の注意事項」を参照してください。

(1)　HttpSession のセッション ID へのサーバ ID 付加

セッション ID は，通常，同一の Web コンテナ内では一意となります。しかし，負荷分散機を使用して複数の Web コンテナで構成するシステムの場合，システム全体ではセッション ID が一意にならないおそれがあります。HttpSession のセッション ID へのサーバ ID 付加機能を使用すると，HttpSession のセッション ID に，Web コンテナごとに異なるサーバ ID が付加されます。これによって，セッション ID をシステム内で一意にできます。

参考

Web サーバとの連携時に，リダイレクタの設定でラウンドロビンでリクエストの振り分けをする場合，HttpSession のセッション ID を付加するかどうかの設定にかかわらず，セッション ID には，ワーカ名が付加されます。サーバ ID は付加されません。

---

(2)　Cookie へのサーバ ID 付加

同一のセッションのリクエストを，同一の Web コンテナに転送するには，負荷分散機の Cookie によってリクエスト転送先を指定する機能と，Cookie へのサーバ ID 付加機能を使用して実現します。

Cookie へのサーバ ID 付加機能では，Cookie に Web コンテナごとに異なるサーバ ID が付加されます。HTTP レスポンスには，サーバ ID が付加された Cookie を付けることができるので，同一のセッションのリクエストを，同一の Web コンテナに転送できます。なお，サーバ ID が付いた Cookie は，HttpSession を生成したリクエストのレスポンスに付加されます。

なお，次の場合，サーバ ID 付加機能で生成される Cookie に Secure 属性が付加されます。

- HTTP リクエストが HTTPS プロトコルで送信されている場合
- ゲートウェイ指定機能でスキームを HTTPS と見なすように設定した場合

2.7.7　cosminexus.xml での定義

アプリケーションの開発環境で必要な cosminexus.xml での定義について説明します。

HttpSession オブジェクト数の上限値の設定

cosminexus.xml の<war>タグ内の<http-session>－<http-session-max-number>タグに指定します。セッション情報の引き継ぎを行う Web アプリケーションには，HTTP セッション数の上限値に 1 以上の有効な値を設定する必要があります。

セッションパラメタのカスタマイズ

セッションパラメタは cosminexus.xml の<war>-<session-config>タグ内に指定します。

セッションパラメタの定義を次の表に示します。

表 2-29　cosminexus.xml でのセッションパラメタのカスタマイズ

指定するタグ	設定内容
<cookie-config>-<name>タグ	HTTP Cookie の名称，および URL のパスパラメタ名を指定します。
<cookie-config>-<http-only>タグ	HTTP Cookie に HttpOnly 属性を付けるかどうかを指定します。
<tracking-mode>タグ	HTTP セッションの管理方法を指定します。

指定するタグの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「2.2.6　War 属性の詳細」を参照してください。

2.7.8　実行環境での設定

セッション管理機能を使用する場合，J2EE サーバの設定が必要です。

また，cosminexus.xml を含まない J2EE アプリケーションを使用する場合，実行環境でのプロパティの設定または変更が必要です。

(1)　J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。セッション管理機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのセッション管理機能の定義について次の表に示します。

表 2-30　簡易構築定義ファイルでのセッション管理機能の定義

項目	指定するパラメタ	設定内容
セッションパラメタのカスタマイズ	webserver.session.cookie_config.name	HTTP Cookie の名称，および URL のパスパラメタ名を指定します。
	webserver.session.cookie_config.http_only	HTTP Cookie に HttpOnly 属性を付けるかどうかを指定します。
	webserver.session.tracking_mode	HTTP セッションの管理方法を指定します。
HttpSession オブジェクト数の上限値の設定	webserver.session.max.throwHttpSessionLimitExceededException	HttpSession オブジェクト数が上限値を超えた場合に com.hitachi.software.web.session.HttpSessionLimitExceededException 例外をスローするかどうかを指定します。 java.lang.IllegalStateException 例外をスローする場合は false を指定します。

項目	指定するパラメタ	設定内容
HttpSession オブジェクト数の上限値の設定	webserver.session.max.log_interval	KDJE39225-E の出力インターバルを指定します。
セッション ID および Cookie へのサーバ ID 付加	webserver.session.server_id.enabled	セッション ID にサーバ ID を付加するかどうかを指定します。
	webserver.session.server_id.value	セッション ID に付加するサーバ ID を指定します。
	webserver.container.server_id.enabled	Cookie にサーバ ID を付加するかどうかを指定します。
	webserver.container.server_id.name	サーバ ID を付加する Cookie の名称を指定します。
	webserver.container.server_id.value	Cookie に付加するサーバ ID を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　J2EE アプリケーションの設定

実行環境での J2EE アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。セッション管理機能の定義には，WAR 属性ファイルを使用します。

WAR 属性ファイルで指定するタグは，cosminexus.xml と対応しています。なお，セッションパラメタのカスタマイズの定義は，Servlet3.0 の標準仕様に準拠しています。cosminexus.xml での定義については，「2.7.7　cosminexus.xml での定義」を参照してください。

2.7.9　セッション管理の注意事項

セッション管理の注意事項について説明します。

(1)　セッションパラメタのカスタマイズと注意事項

HTTP Cookie の名称，および URL のパスパラメタ名は，次の設定で変更できます。

- 簡易構築定義ファイル（webserver.session.cookie_config.name パラメタ）
- cosminexus.xml（<war>-<session-config>-<cookie-config>-<name>タグ）
- web.xml（/web-app/session-config/cookie-config/name 要素）（Servlet 3.0 以降の場合）
- Servlet API（javax.servlet.SessionCookieConfig インタフェースの setName()メソッド）（Servlet 3.0 以降の場合）

! 注意事項

異なる方法による設定が存在する場合，「Servlet API による設定」，「cosminexus.xml の記述」，「web.xml の記述」，「簡易構築定義ファイルの記述」の順に設定が有効となります。

簡易構築定義ファイルの webserver.session.cookie_config.name パラメタが true で，指定した HTTP Cookie の名称，および URL のパスパラメタ名とサーバ ID の Cookie の付加機能で指定した Cookie の名称が重複した場合，次のように動作します。

簡易構築定義ファイルの webserver.session.cookie_config.name パラメタで指定した名称と重複した場合

デフォルトのセッション ID が設定されます。HTTP Cookie のときは「JSESSIONID」となります。URL のパスパラメタのときは「jsessionid」となります。

cosminexus.xml，web.xml，または Servlet API で指定した名称と重複した場合

Web アプリケーションの開始時に KDJE39338-E が出力され，開始に失敗します。Web アプリケーションのリロード処理の実行時は KDJE39338-E が出力され，リロード処理が続行されます。リロード処理中に KDJE39338-E が出力された場合は，Servlet API を使用して Cookie の名前を変更したファイルを修正してから，再度リロード処理を実行してください。

(2)　URL 書き換えをする場合に使用する API と注意事項

URL 書き換えは，J2EE アプリケーション内で URL 書き換えを実行する Servlet API を呼び出したときに実行されます。

- URL 書き換えを実行する Servlet API とは，次に示す javax.servlet.http.HttpServletResponse インタフェースのメソッドです。
- encodeURL(java.lang.String url)メソッド
- encodeRedirectURL(java.lang.String url)メソッド
- encodeUrl(java.lang.String url)メソッド
- encodeRedirectUrl(java.lang.String url)メソッド

これらのメソッドの詳細については，Servlet 仕様を参照してください。なお，encodeUrl(java.lang.String url)メソッド，および encodeRedirectUrl(java.lang.String url)メソッドは，Servlet 2.1 以降，非推奨の API となっています。このため，これら以外のメソッドを使用することをお勧めします。

ここでは，Servlet 仕様で明記されていない API の動作として，URL 書き換えを実行する Servlet API の戻り値についての，アプリケーションサーバでの動作について説明します。

HTTP セッションは，リクエスト処理中の Web アプリケーション内だけで有効です。このため，Servlet API の引数に指定された URL が，リクエスト処理中の Web アプリケーション内を指している URL の場合にだけ，URL 書き換えをします。引数の URL ごとに，リクエスト処理中の Web アプリケーション内を指していると判定する条件を次の表に示します。

表 2-31　引数の URL ごとに，リクエスト処理中の Web アプリケーション内を指していると判定する条件

引数の URL の種類	判定条件
相対 URL （(例）/ex/a.html）	次の条件を満たす場合にだけ，Web アプリケーション内であると判定されます。 • 引数の URL の正規化したパスが，リクエスト処理中の Web アプリケーションのコンテキストルート名を含んでいる。※1
絶対 URL （(例）http://host1/ex/）	次のすべての条件を満たす場合にだけ，Web アプリケーション内であると判定されます。 • 引数の URL のスキームが「http」，または「https」である。※2 • 引数の URL とリクエストの URL が同一のスキームの場合，ポート番号が一致する。 • 引数の URL のホスト名が，リクエストのホスト名と一致する。※1※3 • 引数の URL の正規化したパスが，リクエスト処理中の Web アプリケーションのコンテキストルート名を含んでいる。※1

注※1　名称を比較する場合に，大文字，小文字を区別します。

注※2　名称を比較する場合に，大文字，小文字を区別しません。

注※3　リクエストのホスト名は，リクエストの Host ヘッダのホスト名部分とし，ホスト名の名前解決しないで，文字列を比較します。なお，リクエストのホスト名には，javax.servlet.ServletRequest.getServerName メソッドの戻り値を使用します。なお，次の場合は，同じホストであっても異なるホストと判定します。

- 引数の URL がホスト名の指定で，リクエストの URL が IP アドレスの場合
- 引数の URL が IP アドレスの指定で，リクエストの URL がホスト名の場合

また，URL 書き換えを実行する Servlet API の引数に，URL 以外の文字列を指定した場合の戻り値について説明します。また，URL の先頭にクエリまたはフラグメントを指定した場合についても，あわせて説明します。

URL 書き換えを実行する Servlet API の引数ごとの戻り値を次の表に示します。

表 2-32　URL 書き換えを実行する Servlet API の引数ごとの戻り値

項番	条件		戻り値または発生する例外
	HTTP セッション	引数	
1	－	null	null が返されます。
2	－	URL として不正なフォーマット	java.lang.IllegalArgumentException 例外が発生します。
3	• リクエスト処理中に新規作成した HTTP セッションがある。 • URL 書き換えでセッション ID が通知されている。	空文字列	HTTP リクエストの URL のパスおよびクエリに対して，セッション ID を付加した値が返されます。※1
4		クエリから開始されている URL（先頭文字が「?」の場合）	HTTP リクエストの URL のパスに，セッション ID および引数に指定した値が付加された値が返されます。※1
5		フラグメントから開始されている URL（先頭文字が「#」の場合）	引数に指定した値が返されます。※2
6		現在の HTTP セッションのセッション ID を表すパスパラメタが含まれている URL	引数に指定した値が返されます。
7		リクエスト処理中の Web アプリケーション内と判定される URL	引数にセッション ID を付加した値が返されます。
8	上記の条件以外		引数に指定した値が返されます。

（凡例）－：該当しない

注

この表で示す項番は，条件の優先度を示します。項番の数字が小さいほど，条件の優先度は高くなります。

注※1

引数の URL にパスが含まれないため，引数の URL にパスパラメタを直接付加できません。引数が空文字列やクエリから始まる URL は，リクエストの URL のリソースを指している URL であるため，リクエストの URL にパスパラメタを付加した値を使用して，URL 書き換えをします。

注※2

フラグメントだけの URL は，現在のリソース内の特定の個所を示す URL です。Web ブラウザでは，通常この URL を，表示されているコンテンツ内の移動を示すものとして扱います。このとき，サーバにリクエストは送信しません。なお，これは，RFC3986 に従った動作です。

次に，URL 書き換えを使用してセッション ID を付加した URL の例について説明します。なお，ここで説明する例は，次の前提条件に従っています。

前提条件

- Servlet API は，HTTP セッションの生成後に実行されたものとします。
- HTTP リクエストの URL は，「http://host1/gyoumu1/app1/index.jsp?type=1」です。
- コンテキストルート名は，「/gyoumu1」です。

URL 書き換えに使用する Servlet API の引数の指定値と書き換え後の戻り値（URL）の対応の例を次の表に示します。

表 2–33　URL 書き換えに使用する Servlet API の引数の指定値と書き換え後の戻り値（URL）の対応の例

Servlet API の引数	戻り値
b.html	b.html;jsessionid=AAAAA1111122222333334444455555566svr0
../b.html	../b.html;jsessionid=AAAAA11111222223333344444555556svr0
../../b.html	../../b.html
http://host2/	http://host2
https://host1/gyoumu1/	https://host1/gyoumu1/;jsessionid=AAAAA11111222223333344444555566svr0
""(空文字列)	"/gyoumu1/app1/index.jsp;jsessionid=AAAAA11111222223333344444555566svr0?type=1"
"?mode=2"	"/gyoumu1/app1/index.jsp;jsessionid=AAAAA11111222223333344444555566svr0?mode=2"
"#aaa"	"#aaa"

(3)　URL 書き換えを使用する場合の注意事項

URL 書き換えを使用する場合の注意事項について説明します。

● 静的コンテンツからの画面遷移

静的コンテンツ（HTML ファイルなど）から遷移した場合，URL 書き換えによって管理されているセッションは維持されません。

URL 書き換えを使用してセッションを管理する場合は，常にサーブレットまたは JSP を使用して画面を遷移するように実装してください。また，サーブレットまたは JSP 内で，Servlet API によって URL を書き換えて，セッション ID を追加する処理を実装してください。

● Web アプリケーションで取得したリクエスト URL

HTTP リクエストの URL に，URL 書き換えによって追加されたセッション ID を示すパスパラメタが含まれる場合であっても，次のメソッドによって取得した URL には，セッション ID を示すパスパラメタは含まれません。

インタフェース

javax.servlet.http.HttpServletRequest インタフェース

メソッド

- getRequestURI()メソッド
- getRequestURL()メソッド

(4)　セッションの管理に使用する HTTP Cookie の Secure 属性

HTTP リクエストを HTTPS プロトコルで送信した場合に，Web コンテナが生成したセッション ID が HTTP Cookie によってクライアントに返されます。そのとき，HTTP Cookie に Secure 属性が付与されます。

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

2.8　アプリケーションのイベントリスナ

この節では，アプリケーションのイベントリスナ機能について説明します。

Web アプリケーション単位でのイベントリスナ機能があります。アプリケーションイベントリスナは，Web アプリケーションのデプロイ時にインスタンス化されます。インスタンス化されたアプリケーションイベントリスナは，サーブレットコンテキストオブジェクトまたはセッションオブジェクトのどちらか一方，または両方の状態変化イベントを受け取ります。リスナオブジェクトが受け取るイベントを次に示します。

- セッションオブジェクトの新規生成
- セッションオブジェクトのシリアライズ前※1
- セッションオブジェクトのデシリアライズ後※1
- セッションオブジェクトの抹消
- セッションオブジェクトの属性の追加，削除，変更
- サーブレットコンテキストオブジェクトの生成
- サーブレットコンテキストオブジェクトの抹消
- サーブレットコンテキストオブジェクトの属性の追加，削除，変更
- Web アプリケーションへのリクエストの到着※2
- Web アプリケーションでのリクエスト処理の完了※2
- リクエストオブジェクトの属性の追加，削除，変更※2

注※1

サーブレットエンジンのベンダ独自処理の都合で，セッションオブジェクトをシリアライズして格納，通信し，別の時点でデシリアライズして処理を再開することが，Servlet API では想定されています。

セッションオブジェクトに，これらのイベント通知が設けられている意図は，セッションオブジェクトにデータだけでなく，データベースコネクションやオブジェクトリファレンスなどの資源を持たせられるようにするためです。セッションオブジェクトに資源を持たせるような設計のアプリケーションでは，シリアライズの前にいったん資源を解放し，デシリアライズ後に再び獲得するようにしなければなりません。

セッションオブジェクトに資源を持たせるという構成は，注意深く利用しないとサーバ全体の必要資源量を大幅に増大させてしまい，資源が不足するおそれがあります。このため，イベントリスナ機能は，資源を確保できる場合に利用してください。

注※2

Servlet 2.4 以降の仕様に準拠した Web アプリケーションの場合だけ使用できます。

2.9　リクエストおよびレスポンスのフィルタリング機能

この節では，リクエストおよびレスポンスのフィルタリング機能について説明します。

この節の構成を次の表に示します。

表 2-34　この節の構成（リクエストおよびレスポンスのフィルタリング機能）

分類	タイトル	参照先
解説	アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）	2.9.1
	フィルタ連鎖の推奨例	2.9.2
実装	DD での定義	2.9.3
設定	実行環境での設定（Web アプリケーションの設定）	2.9.4

注　「運用」および「注意事項」について，この機能固有の説明はありません。

アプリケーションサーバで使用できるフィルタリングの機能には，Servlet 仕様で規定されている機能と，アプリケーションサーバで提供する機能があります。どちらの機能も，サーブレット/JSP のリクエストやレスポンスに対してフィルタリングをする機能です。

Servlet 仕様で規定されているフィルタリング機能では，サーブレット/JSP を実行する前のリクエスト，またはサーブレット/JSP を実行したあとのレスポンスをラップします。これによって，データの変更，リソースに対するトレースの取得などができます。

また，アプリケーションサーバが提供するフィルタリング機能では，セッション情報を引き継いだり，HTTP レスポンスを圧縮したりできます。アプリケーションサーバでは，これらのフィルタリング機能を使用するためにサーブレットフィルタを提供しています。アプリケーションサーバが提供するサーブレットフィルタを **built-in フィルタ**といいます。次項でアプリケーションサーバが提供する built-in フィルタについて説明します。

2.9.1　アプリケーションサーバが提供するサーブレットフィルタ (built-in フィルタ)

アプリケーションサーバでは，次に示す機能を使用するためのサーブレットフィルタ（built-in フィルタ）を提供しています。

- J2EE サーバ間のセッション情報の引き継ぎ（メモリセッションフェイルオーバ機能）
  J2EE サーバ間のセッション情報を引き継ぐために，アプリケーションサーバは built-in フィルタとしてセッションフェイルオーバ用フィルタを提供しています。
- HTTP レスポンスの圧縮
  HTTP リクエストに対する HTTP レスポンスを圧縮するために，アプリケーションサーバは built-in フィルタとして HTTP レスポンス圧縮フィルタを提供しています。

built-in フィルタの種類を次の表に示します。また，Web アプリケーションに built-in フィルタを組み込むことによって使用できる機能の参照先をあわせて示します。

表 2-35　built-in フィルタの種類

built-in フィルタの種類	機能の説明	参照先マニュアル	参照個所
セッションフェイルオーバ用フィルタ	J2EE アプリケーションで実行中のセッション情報を管理します。J2EE サーバで障害が発生したときに，管理しているセッション情報をほかの J2EE サーバに引き継ぎます。	マニュアル「アプリケーションサーバ 機能解説 互換編」	6.2，6.4
HTTP レスポンス圧縮フィルタ	サーブレット，JSP，および静的コンテンツへの HTTP リクエストに対する HTTP レスポンスを gzip 形式に圧縮します。	このマニュアル	2.10

ポイント

各フィルタを Web アプリケーションに組み込む場合は，フィルタマッピング定義で，「セッションフェイルオーバ用フィルタ」，「HTTP レスポンス圧縮フィルタ」の順に定義してください。セッションフェイルオーバ用フィルタは，すべての built-in フィルタ，ユーザのフィルタよりも前に配置する必要があります。

なお，Servlet 3.0 以降では，web.xml ではなく API でのフィルタ定義ができますが，built-in フィルタは API でのフィルタ定義はできません。

以降で，built-in フィルタの HTTP リクエスト・HTTP レスポンスへの作用，built-in フィルタの動作条件に関する制約について説明します。

(1)　HTTP リクエストおよび HTTP レスポンスへの作用

built-in フィルタは，クライアント側から送信される HTTP リクエストのリクエストヘッダ，およびリクエストボディに対して作用して，情報の削除，追加，変更などを実施する場合があります。また，サーバから送信される HTTP レスポンスのレスポンスヘッダ，およびレスポンスボディに対して同様に作用する場合もあります。built-in フィルタの HTTP リクエストおよび HTTP レスポンスへの作用を次の表に示します。

表 2-36　built-in フィルタの HTTP リクエストおよび HTTP レスポンスへの作用

built-in フィルタの種類	HTTP リクエストへの作用	HTTP レスポンスへの作用
セッションフェイルオーバ用フィルタ	－	－
HTTP レスポンス圧縮フィルタ	－	レスポンスボディが圧縮される場合，Content-Encoding ヘッダに「gzip」を指定します。また，レスポンスボディを gzip 形式で圧縮します。

（凡例）　－：該当なし

(2)　動作条件に関する制約

ユーザのフィルタと，built-in フィルタを同時に使用する場合，フィルタ連鎖の中で built-in フィルタが呼び出される順序に制約があります。

built-in フィルタごとに次に示す制約について説明します。

- 位置に関する制約
  フィルタ連鎖中の built-in フィルタの位置（呼び出される順序）に関する制約です。

- 動作条件に関する制約
  built-in フィルタが動作する上での前提条件などの動作条件に関する制約です。
- 前後に配置するほかのサーブレットフィルタに対する制約
  built-in フィルタの前後に配置するサーブレットフィルタに対する制約です。

(a)　セッションフェイルオーバ用フィルタに対する制約

セッションフェイルオーバ用フィルタに対する制約を次の表に示します。

表 2-37　セッションフェイルオーバ用フィルタに対する制約

制約の種類	説明
位置に関する制約	フィルタ連鎖の中で，最初に呼び出される必要があります。すべてのユーザのフィルタおよび built-in フィルタよりも前に配置する必要があります。
動作条件に関する制約	－
前後に配置するほかのサーブレットフィルタに対する制約	－

（凡例）　－：該当なし

(b)　HTTP レスポンス圧縮フィルタに対する制約

HTTP レスポンス圧縮フィルタに対する制約を次の表に示します。

表 2-38　HTTP レスポンス圧縮フィルタに対する制約

制約の種類	説明
位置に関する制約	－
動作条件に関する制約	－
前後に配置するほかのサーブレットフィルタに対する制約	javax.servlet.http.HttpServletResponse クラスの setHeader メソッド，addHeader メソッド，setIntHeader メソッド，または addIntHeader メソッドを使用して，Content-Length ヘッダ，または Content-Encoding ヘッダの設定を変更するサーブレットフィルタと同時に使用する場合，そのサーブレットフィルタは HTTP レスポンス圧縮フィルタよりあとに配置する必要があります。

（凡例）　－：該当なし

2.9.2　フィルタ連鎖の推奨例

フィルタ連鎖の推奨例を次に示します。次に示す順序で連鎖するように配置してください。

なお，例で説明するユーザのフィルタは，リクエストボディおよびレスポンスボディに作用する一般的なフィルタを想定しています。

- **セッションフェイルオーバ用フィルタ，HTTP レスポンス圧縮フィルタを使用する場合**
  1. セッションフェイルオーバ用フィルタ
  2. HTTP レスポンス圧縮フィルタ
- **セッションフェイルオーバ用フィルタ，HTTP レスポンス圧縮フィルタ，およびユーザのフィルタ（FilterA）を使用する場合**

1. セッションフェイルオーバ用フィルタ
2. HTTP レスポンス圧縮フィルタ
3. ユーザのフィルタ（FilterA）

- **セッションフェイルオーバ用フィルタ，およびユーザのフィルタ（FilterC）を使用する場合**

1. セッションフェイルオーバ用フィルタ
2. ユーザのフィルタ（FilterC）

- **HTTP レスポンス圧縮フィルタ，およびユーザのフィルタ（FilterD）を使用する場合**

1. HTTP レスポンス圧縮フィルタ
2. ユーザのフィルタ（FilterD）

2.9.3 DD での定義

リクエストおよびレスポンスのフィルタリング機能の定義は，web.xml に指定します。

DD でのリクエストおよびレスポンスのフィルタリング機能の定義については，それぞれ次の場所を参照してください。

- セッションフェイルオーバ用フィルタについては，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「5. J2EE サーバ間のセッション情報の引き継ぎ」を参照してください。
- HTTP レスポンス圧縮フィルタについては，「2.10 HTTP レスポンス圧縮機能」を参照してください。

2.9.4 実行環境での設定（Web アプリケーションの設定）

リクエストおよびレスポンスのフィルタリング機能を定義する場合，Web アプリケーションの設定が必要です。cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。リクエストおよびレスポンスのフィルタリングの定義には，フィルタ属性ファイルおよび WAR 属性ファイルを使用します。

フィルタ属性ファイルおよび WAR 属性ファイルで指定するタグは，DD と対応しています。DD（web.xml）での定義については，「2.9.3 DD での定義」を参照してください。

2.10　HTTP レスポンス圧縮機能

この節では，HTTP レスポンス圧縮機能について説明します。

HTTP レスポンス圧縮機能とは，サーブレット，JSP，および静的コンテンツへの HTTP リクエストに対する HTTP レスポンスを，gzip 形式に圧縮する機能です。この機能を使用して HTTP レスポンスを圧縮することで，Web コンテナと Web クライアント（ブラウザなど）との間の HTTP レスポンス通信に掛かる時間を削減できます。

この機能は，Web アプリケーションに組み込んで動作させるサーブレットフィルタとして提供されています。これを，**HTTP レスポンス圧縮フィルタ**といいます。

この節の構成を次の表に示します。

表 2-39　この節の構成（HTTP レスポンス圧縮機能）

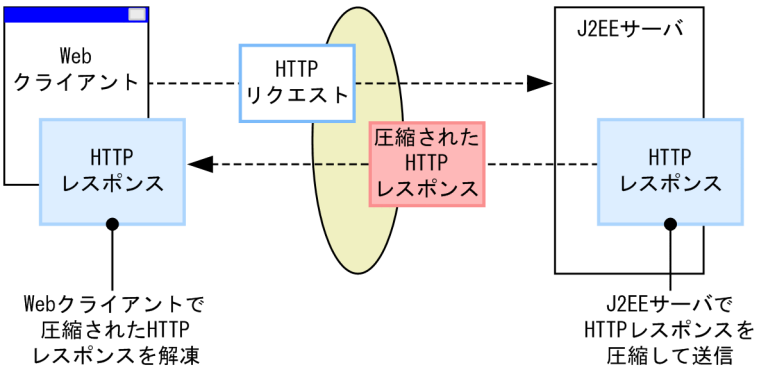
分類	タイトル	参照先
解説	HTTP レスポンス圧縮フィルタの概要	2.10.1
	HTTP レスポンス圧縮フィルタを使用するための条件	2.10.2
実装	HTTP レスポンス圧縮フィルタを使用するアプリケーションの実装	2.10.3
	DD での定義	2.10.4
	DD の定義例	2.10.5
設定	実行環境での設定（Web アプリケーションの設定）	2.10.6

注　「運用」および「注意事項」について，この機能固有の説明はありません。

2.10.1　HTTP レスポンス圧縮フィルタの概要

HTTP レスポンス圧縮機能を有効にすると，HTTP レスポンスのレスポンスボディを gzip 形式に圧縮します。HTTP レスポンス圧縮機能の概要を次の図に示します。

図 2-12　HTTP レスポンス圧縮機能の概要



HTTP レスポンス圧縮機能を有効にするためには，アプリケーションサーバが提供する HTTP レスポンス圧縮フィルタを Web アプリケーションに組み込む必要があります。HTTP レスポンス圧縮機能を適用する場合，Web アプリケーションの DD（web.xml）に HTTP レスポンス圧縮フィルタのフィルタ定義，およびフィルタマッピングの定義を追加します。J2EE サーバにデプロイ済みの Web アプリケーションに

適用する場合は，サーバ管理コマンドを使用して，HTTP レスポンス圧縮フィルタのフィルタ定義，およびフィルタマッピングの定義を追加します。

2.10.2 HTTP レスポンス圧縮フィルタを使用するための条件

ここでは，HTTP レスポンス圧縮フィルタを使用する場合の条件や注意事項について説明します。

(1) 前提条件

HTTP レスポンス圧縮機能を使用にするには，次の前提条件を満たしている必要があります。

- Web クライアントの gzip 形式の対応
  HTTP レスポンス圧縮機能を有効にした場合，gzip 形式で圧縮された HTTP レスポンスを Web クライアントで解凍する必要があります。そのため，Web クライアントが gzip 形式に対応していることが前提となります。Web クライアントが gzip 形式に対応していない場合は，HTTP レスポンス圧縮機能を有効にしていても，HTTP レスポンスは圧縮されません。
- Web クライアントの HTTP/1.1 の対応
  HTTP レスポンス圧縮機能では，Web クライアントが gzip 形式に対応しているかどうかを HTTP リクエストの Accept-Encoding ヘッダに指定された値によって判定しています。そのため，Web クライアントは Accept-Encoding ヘッダの仕様が規定されている HTTP/1.1 に対応している必要があります。

(2) 必要なメモリ量

HTTP レスポンス圧縮機能に必要なメモリ量は次の計算式で求められます。

HTTP レスポンス圧縮機能に必要なメモリ量（バイト）=HTTP レスポンス圧縮機能が有効となる HTTP リクエストの同時実行数×レスポンスの圧縮しきい値（バイト）

圧縮しきい値とは，HTTP レスポンスボディのサイズによって，HTTP レスポンスを圧縮するかどうかを判定するためのしきい値です。HTTP レスポンスボディのサイズが圧縮しきい値に定義したサイズを超える場合にだけ，HTTP レスポンスを圧縮します。なお，圧縮しきい値は，HTTP リクエストに対して設定します。

圧縮しきい値は，DD（web.xml）に定義します。圧縮しきい値を定義することで，HTTP レスポンスのサイズが小さい場合に，HTTP レスポンスの圧縮に掛かる時間が，通信に掛かる時間よりも大きくならないようにします。

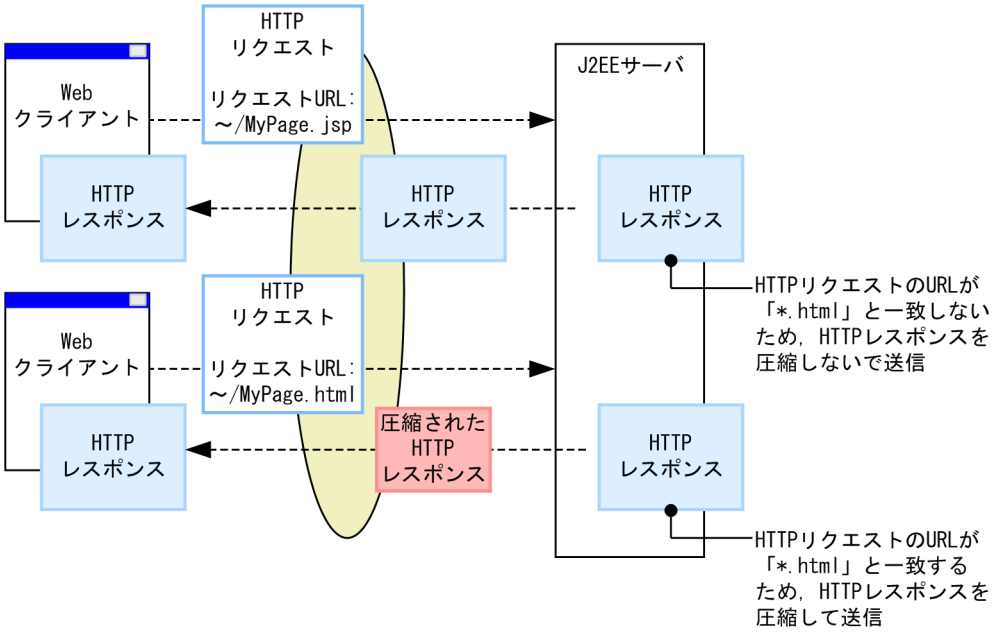
圧縮しきい値は，圧縮するリソースの種別と通信回線の速度によって適切なサイズが決まります。圧縮しきい値に定義するサイズは実測で求め，適切なサイズを定義することを推奨します。

(3) HTTP レスポンス圧縮機能を有効にする場合の条件

HTTP レスポンス圧縮機能が有効になる条件を指定できます。指定できる条件を次に示します。

- HTTP リクエストの URL パターン
  HTTP レスポンス圧縮フィルタを実装している Web アプリケーションに対するリクエストの URL が，指定した URL パターンと一致する場合，そのリクエストに対するレスポンスを圧縮します。
  HTTP レスポンスの圧縮を実行する HTTP リクエストの URL パターンとして「*.html」を指定した場合の例を次の図に示します。

図 2-13　HTTP レスポンスの圧縮を実行する HTTP リクエストの URL パターンとして「*.html」を指定した場合の例



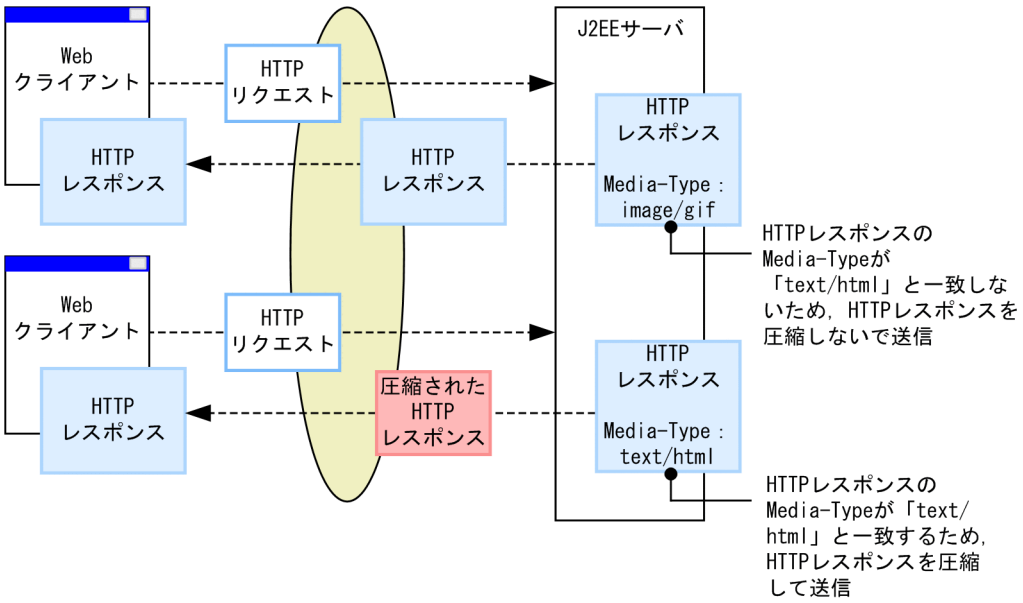
- HTTP レスポンスの Media-Type

  HTTP レスポンスの Content-Type ヘッダに含まれる Media-Type の値が，指定した値と一致する場合，HTTP レスポンスを圧縮します。

  HTTP レスポンスの Media-Type は，サーブレット/JSP の場合は J2EE アプリケーションが setContentType メソッドによって設定した値です。静的コンテンツの場合は拡張子に対応づけられた MIME タイプとなります。

  HTTP レスポンスの圧縮を実行する HTTP レスポンスの Media-Type として「text/html」を指定した場合の例を次の図に示します。

図 2-14　HTTP レスポンスの圧縮を実行する HTTP レスポンスの Media-Type として「text/html」を指定した場合の例

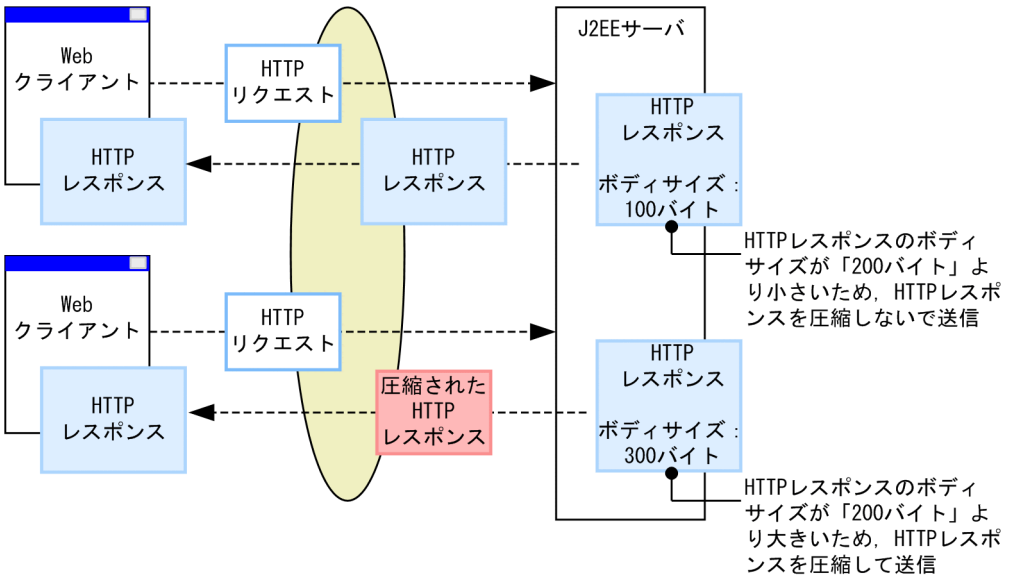


- HTTP レスポンスのボディサイズ

  HTTP レスポンスの圧縮を実行するためしきい値を設定して，ボディサイズがこのしきい値を超える場合，HTTP レスポンスを圧縮します。

  HTTP レスポンスの圧縮を実行する HTTP レスポンスのボディサイズとして「200 バイト」を指定して HTTP レスポンス圧縮機能を有効にした場合の例を次の図に示します。

  図 2-15　HTTP レスポンスの圧縮を実行する HTTP レスポンスのボディサイズとして「200 バイト」を指定した場合の例



(4)　注意事項

HTTP レスポンス圧縮フィルタの定義に関する注意事項

HTTP レスポンス圧縮フィルタを使用する場合，built-in フィルタの HTTP リクエストおよび HTTP レスポンスへの作用やフィルタ連鎖の順序制約を考慮して，Web アプリケーションに HTTP レスポンス圧縮フィルタを組み込む必要があります。built-in フィルタの詳細については，「2.9.1　アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）」を参照してください。

なお，Servlet 3.0 以降では，web.xml ではなく API でのフィルタ定義ができますが，built-in フィルタは API でのフィルタ定義はできません。

エラーページに関する注意事項

HTTP レスポンス圧縮機能を使用する Web アプリケーションでは，次の機能を使用してエラーページをカスタマイズできます。

- Web サーバの機能を使用したエラーページのカスタマイズ
- インプロセス HTTP サーバによるエラーページのカスタマイズ
- web.xml の<error-page>タグを使用したエラーページのカスタマイズ

web.xml の<error-page>タグを使用したエラーページを使用する場合は，エラーページに静的コンテンツ，またはレスポンスから javax.servlet.ServletOutputStream を取得して使用するサーブレットを指定してください。

HTTP レスポンス圧縮機能では圧縮後のデータの出力にレスポンスオブジェクトから取得した javax.servlet.ServletOutputStream を使用します。そのため，エラーページを生成するサーブレット，または JSP でレスポンスオブジェクトからは，java.io.PrintWriter を取得できません。

2.10.3　HTTP レスポンス圧縮フィルタを使用するアプリケーションの実装

ここでは，HTTP レスポンス圧縮フィルタを使用するアプリケーションを開発するときの注意事項について説明します。

(1)　ほかのフィルタと併用する場合の呼び出し順序

HTTP レスポンス圧縮フィルタは，HTTP レスポンスのヘッダに設定するほかのフィルタよりも前に呼び出される必要があります。javax.servlet.http.HttpServletResponse の setHeader メソッド，addHeader メソッド，setIntHeader メソッド，および addIntHeader メソッドを使用して，Content-Length ヘッダ，Content-Encoding ヘッダを設定するほかのフィルタを使用する場合は，HTTP レスポンス圧縮フィルタよりも後ろに配置してください。

(2)　HTTP レスポンスのバッファに関する注意事項

HTTP レスポンス圧縮機能を有効にした場合，HTTP レスポンスのバッファよりも前に圧縮しきい値に指定したサイズのバッファが設けられます。HTTP レスポンスのバッファには，出力したデータが圧縮しきい値を超えたときに書き込まれます。

圧縮後のデータサイズが HTTP レスポンスのバッファサイズを超えなければ，Web クライアントに HTTP レスポンスが書き出されることはありません。出力したデータのサイズが圧縮しきい値を超える前に，HTTP レスポンスを Web クライアントに書き出す必要がある場合は，明示的にレスポンス出力用ストリーム※の flush メソッドを呼び出す必要があります。ただし，出力したデータのサイズが圧縮しきい値を超える前に flush メソッド，または javax.servlet.ServletResponse クラスの flushBuffer メソッドを呼び出した場合は，出力したデータは圧縮されないで Web クライアントに書き出されます。

注※

レスポンス出力用ストリームとは，次に示すオブジェクトを指します。

- javax.servlet. ServletResponse クラスの getOutputStream メソッドによって取得した javax.servlet.ServletOutputStream
- javax.servlet. ServletResponse クラスの getWriter メソッドによって取得した java.io.PrintWriter
- JSP で暗黙的に利用できる javax.servlet.jsp.JspWriter

(3)　HTTP レスポンスのレスポンスヘッダに関する注意事項

HTTP レスポンス圧縮機能によって HTTP レスポンスのレスポンスボディが圧縮される場合，この HTTP レスポンスの Content-Encoding ヘッダには gzip，Vary ヘッダには Accept-Encoding が指定されます。Content-Length ヘッダには何も指定されません。

したがって，javax.servlet.ServletResponse クラスの setContentLength メソッドを使用する場合と javax.servlet.http.HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※を使用する場合は，次の点に注意してください。

- 次のどちらかの API を使用して Content-Length ヘッダ，Content-Encoding ヘッダを設定するフィルタを追加している場合は，レスポンス圧縮用フィルタよりもあとに実行されるように DD（web.xml）で定義してください。
  - ServletResponse クラスの setContentLength メソッド
  - HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※

- HTTP レスポンスのレスポンスボディが圧縮される場合，ServletResponse クラスの setContentLength メソッド，および HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※を使用しても，HTTP レスポンスの Content-Length ヘッダは付加されません。Content-Length ヘッダが付加されていない HTTP レスポンスは Web コンテナによって chunk 形式でクライアントに送信されます。
- HTTP レスポンスのレスポンスボディが圧縮される場合，HttpServletResponse クラスのレスポンスヘッダを追加・変更する API※を使用しても Content-Encoding ヘッダには値が設定されません。レスポンスボディが圧縮される場合は，Web コンテナによって Content-Encoding ヘッダに gzip が指定されます。

注※

レスポンスヘッダを追加・変更する API とは，javax.servlet.http.HttpServletResponse クラスの次のメソッドを指します。

- setHeader メソッド
- addHeader メソッド
- setIntHeader メソッド
- addIntHeader メソッド

(4) HTTP レスポンスのデータ出力に関する注意事項

javax.servlet.ServletResponse クラスの getOutputStream メソッド，または getWriter メソッドによって ServletOutputStream または PrintWriter を取得し，HTTP レスポンスを出力する場合は，次の点に注意してください。

- ServletOutputStream または PrintWriter を使用して，HTTP レスポンスのバッファにデータを書き出している状態で，ServletResponse の setContentType メソッドを呼び出した場合，圧縮するように指定された Media-Type を持つ HTTP レスポンスであっても圧縮されません。

  ただし，圧縮する Media-Type にアスタリスク（*）が指定されているときは圧縮されます。
- Media-Type を指定して JSP の出力を圧縮するためには，page ディレクティブの contentType 属性を指定するか，または JSP のバッファを超える前に ServletResponse クラスの setContentType メソッドを呼び出す必要があります。

(5) アプリケーションで HTTP レスポンスを圧縮する場合の注意事項

アプリケーションで圧縮した HTTP レスポンスには，HTTP レスポンス圧縮機能が有効にならないように設定してください。アプリケーションで圧縮した HTTP レスポンスに対して，HTTP レスポンス圧縮機能を有効にした場合の動作は保証できません。

2.10.4 DD での定義

ここでは，HTTP レスポンス圧縮機能を使用する場合に必要な DD の定義について説明します。

HTTP レスポンス圧縮機能を有効にするためには，Web アプリケーションの DD に，フィルタ定義およびフィルタマッピング定義を追加する必要があります。HTTP レスポンス圧縮機能は，フィルタマッピング定義によってマッピングされた URL パターンを持つリソースに対してリクエストがあった場合にだけ有効となります。

HTTP レスポンス圧縮機能の定義は，web.xml に指定します。

DD での HTTP レスポンス圧縮機能の定義について次の表に示します。

表 2-40　DD での HTTP レスポンス圧縮機能の定義

設定項目	指定するタグ	設定内容
フィルタ定義	<filter>タグ内の<filter-name>タグ	追加するフィルタ名称を指定します。設定値は，固定値です。 **設定値（固定値）** com.hitachi.software.was.web.ResponseCompressionFilter
	<filter>タグ内の<filter-class>タグ	追加するフィルタのクラス名を指定します。設定値は，固定値です。 **設定値（固定値）** com.hitachi.software.was.web.ResponseCompressionFilter
URL パターンと HTTP レスポンス圧縮規則のマッピング	<filter-class><init-param>タグ内の<param-name>タグ，<param-value>タグ	URL パターンと HTTP レスポンス圧縮機能のマッピングを指定します。 詳細については，「2.10.4(1)　URL パターンと HTTP レスポンス圧縮規則のマッピング（url-mapping）」を参照してください。
HTTP レスポンス圧縮規則		HTTP レスポンスの圧縮規則として，圧縮規則名，Media-Type と圧縮しきい値を指定します。 詳細については，「2.10.4(2)　HTTP レスポンス圧縮規則」を参照してください。
フィルタマッピング定義	<filter-mapping>タグ内の<filter-name>タグ	フィルタ名称を指定します。設定値は，固定値です。 **設定値（固定値）** com.hitachi.software.was.web.ResponseCompressionFilter
	<filter-mapping>タグ内の<url-pattern>タグ	URL パターンやサーブレットクラスと，HTTP レスポンス圧縮フィルタのマッピングを指定します。HTTP レスポンス圧縮機能は，フィルタマッピング定義によって，マッピングされた URL パターンを持つリソースに対してリクエストがあった場合だけ有効となります。 設定値は，任意です。

次に示す DD の定義例を基に，DD の定義内容を説明します。

```
      :
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /*=rule1;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       *:1000;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

<filter>タグで囲まれた部分がフィルタ定義，<filter-mapping>タグで囲まれた部分がフィルタマッピング定義となります。フィルタ定義の<filter-name>タグおよび<filter-class>タグに指定する値は，次のパッケージ名で固定です。

com.hitachi.software.was.web.ResponseCompressionFilter

次に，DD の<init-param>タグ内に定義する内容を示します。

(1) URL パターンと HTTP レスポンス圧縮規則のマッピング（url-mapping）

HTTP レスポンス圧縮機能が有効となる URL パターンと，指定した URL パターンに適用する HTTP レスポンス圧縮規則名のマッピングを指定します。

パラメタ指定時の規則および URL パターンのマッピング規則を示します。

(a) パラメタ指定時の規則

- URL パターンは，大文字と小文字が区別されます。
- HTTP レスポンス圧縮規則名は，大文字と小文字が区別されます。
- URL パターンと HTTP レスポンス圧縮規則名は，半角のイコール（=）で区切ります。
- 複数指定する場合は，半角のセミコロン（;）で区切ります。
- 複数の URL パターンに該当する場合は，先に指定したものが使用されます。
- パラメタ値の改行，タブ，スペースは無視されます。
- パラメタ値の末尾にある半角のセミコロン（;）は無視されます。

(b) URL パターンのマッピング規則

url-mapping パラメタに定義する URL パターンには，パス一致，拡張子一致，および完全一致のマッピング規則が適用されます。それぞれのマッピング規則を次に示します。

- **パス一致**

  URL パターンとして「/」で始まり「/*」で終わる文字列を指定した場合は，リクエスト URL のコンテキストルートからの相対パスが URL パターンの「*」を除く文字列で始まっているときに，一致したとみなされます。また，URL パターンとして「/*」を指定した場合は，すべてのリクエスト URL が一致したとみなされます。

  （例）

  URL パターンが「/jsp/*」では，リクエスト URL のコンテキストルートからの相対パスが「/jsp/index.jsp」の場合に一致したとみなされます。

- **拡張子一致**

  URL パターンとして「*.」で始まる文字列を指定した場合は，リクエスト URL の拡張子と URL パターンの「*.」に続く文字列と同じであるときに，一致したとみなされます。

  （例）

  URL パターンが「*.jsp」では，リクエスト URL のコンテキストルートからの相対パスが「/jsp/index.jsp」の場合に一致したとみなされます。

- **完全一致**

  URL パターンとして「/」で始まる上記以外の文字列を指定した場合は，リクエスト URL のコンテキストルートからの相対パスがこの URL パターンと完全に同じであるときに，一致したとみなされます。

  （例）

  URL パターンが「/jsp/index.jsp」では，リクエスト URL のコンテキストルートからの相対パスが「/ jsp/index.jsp」の場合に一致したとみなされます。

(2)　HTTP レスポンス圧縮規則

圧縮する HTTP レスポンスの Media-Type と圧縮しきい値を指定します。

Media-Type

HTTP レスポンス圧縮機能では，条件として指定した Media-Type が HTTP レスポンスに含まれる場合に，HTTP レスポンスを圧縮します。

圧縮しきい値

圧縮しきい値は，100 から 2,147,483,647 までの整数値で指定します。デフォルトの設定では，すべての Media-Type に対して圧縮しきい値「100」が適用されます。

圧縮しきい値とは，HTTP レスポンスボディのサイズによって，HTTP レスポンスを圧縮するかどうかを判定するためのしきい値です。HTTP レスポンス圧縮機能では，HTTP レスポンスボディのサイズが圧縮しきい値に定義したサイズを超える場合に，HTTP レスポンスを圧縮します。

圧縮しきい値を定義することで，HTTP レスポンスのサイズが小さい場合に，HTTP レスポンスの圧縮に掛かる時間が，通信に掛かる時間よりも大きくならないようにします。

圧縮しきい値は，圧縮するリソースの種別と通信回線の速度によって適切なサイズが決まるので，圧縮しきい値に定義するサイズは実測で求め，適切なサイズを定義することをお勧めします。

パラメタ指定時の規則を示します。

- Media-Type にアスタリスク（*）を指定した場合は，すべての Media-Type を表します。ただし，Media-Type ごとの指定がある場合は，Media-Type ごとの指定が優先されます。
- Media-Type は，大文字と小文字が区別されません。
- Media-Type と圧縮しきい値は，半角のコロン（:）で区切ります。
- 複数指定する場合は半角のセミコロン（;）で区切ります。
- 同じ Media-Type を複数指定した場合はあとに指定したものが使用されます。
- パラメタ値の改行，タブ，スペースは無視されます。
- パラメタ値の末尾にある半角のセミコロン（;）は無視されます。

(3)　注意事項

HTTP レスポンス圧縮フィルタの設定時の注意事項を次に示します。

- レスポンス圧縮用フィルタの初期化時に，フィルタの初期化パラメタの妥当性をチェックします。ここで初期化パラメタに定義された値に問題がある場合は，フィルタの初期化処理でエラーとなり，Web アプリケーションは開始しません。
- リクエスト URL が url-mapping パラメタに指定した URL パターンに一致しても，レスポンス圧縮用フィルタがマッピングされている URL パターンに一致しない場合は，url-mapping パラメタに指定したレスポンス圧縮規則が適用されることはありません。
- レスポンス圧縮フィルタに複数の URL パターンをマッピングする場合は，リクエスト URL が同時に複数の URL パターンに一致しないようにフィルタマッピングの定義を記述する必要があります。複数の URL パターンに対して異なるレスポンス圧縮機能を設定する場合は，url-mapping パラメタに複数の URL パターンを指定してください。
- メモリセッションフェイルオーバ機能を使用している場合は，セッションフェイルオーバ用フィルタのフィルタマッピングの定義よりも下にレスポンス圧縮フィルタのフィルタマッピングの定義を記述してください。

- Web アプリケーションのバージョンが Servlet 2.4 以降の場合，<filter-mapping>の<dispatcher>タグは指定しないでください。また，<dispatcher>タグで「FORWARD」，「INCLUDE」，「ERROR」を指定した場合，Web アプリケーションの開始時にエラーが発生し，アプリケーションが開始できないので，注意してください。

2.10.5 DD の定義例

ここでは，次に示すそれぞれの場合を例に，HTTP レスポンス圧縮機能を使用する DD の定義例を示します。

- HTTP レスポンスのボディサイズで圧縮条件を指定する場合
- URL パターンで圧縮条件を指定する場合
- HTTP レスポンスの Media-Type で圧縮条件を指定する場合
- 圧縮条件を組み合わせて指定する場合

(1) HTTP レスポンスのボディサイズで圧縮条件を指定する場合

HTTP レスポンスのボディサイズで圧縮条件を指定する場合の定義例を示します。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /*=rule1;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       *:1000;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例では，URL パターンが/*のアクセスに対する HTTP レスポンスで，ボディサイズが 1,000 バイトを超えるものを圧縮します。

(2) URL パターンで圧縮条件を指定する場合

URL パターンで圧縮条件を指定する場合の定義例を示します。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /app/dir/*=rule1;
       *.html=rule1;
```

```
      </param-value>
    </init-param>
    <init-param>
      <param-name>rule1</param-name>
      <param-value>
        *:100;
      </param-value>
    </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/app/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例では，URL パターンが/app/*のアクセスに対する HTTP レスポンスで次の条件を満たすものを圧縮します。

- HTTP リクエストの URL パターンが/app/dir/*で，ボディサイズが 100 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが*.html で，ボディサイズが 100 バイトを超える HTTP レスポンス

(3)　HTTP レスポンスの Media-Type で圧縮条件を指定する場合

HTTP レスポンスの Media-Type で圧縮条件を指定する場合の定義例を示します。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /*=rule1;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       text/html:500;
       application/pdf:1000;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例は URL パターンが/*のアクセスに対する HTTP レスポンスで次の条件を満たすものを圧縮します。

- Media-Type が text/html でボディサイズが 500 バイトを超える HTTP レスポンス
- Media-Type が application/pdf でボディサイズが 1,000 バイトを超える HTTP レスポンス

(4)　圧縮条件を組み合わせて指定する場合

次に示す例のように，圧縮条件を組み合わせて定義することもできます。

```
<web-app>
      :
<!-- The filter for Response Compression Filter -->
<filter>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
   <init-param>
     <param-name>url-mapping</param-name>
     <param-value>
       /app/dir1/*=rule1;
       /app/dir2/*=rule2;
       *.html=rule3;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule1</param-name>
     <param-value>
       *:500;
       application/pdf:1000;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule2</param-name>
     <param-value>
       application/pdf:2000;
     </param-value>
   </init-param>
   <init-param>
     <param-name>rule3</param-name>
     <param-value>
       *:100;
     </param-value>
   </init-param>
</filter>
      :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
   <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
   <url-pattern>/app/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

定義例では，URL パターンが/app/*のアクセスに対する HTTP レスポンスで次の条件を満たすものを圧縮します。

- HTTP リクエストの URL パターンが/app/dir1/*で Media-Type が application/pdf 以外，ボディサイズが 500 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが/app/dir1/*で Media-Type が application/pdf，ボディサイズが 1,000 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが/app/dir2/*で Media-Type が application/pdf，ボディサイズが 2,000 バイトを超える HTTP レスポンス
- HTTP リクエストの URL パターンが*.html でボディサイズが 100 バイトを超える HTTP レスポンス

2.10.6 実行環境での設定（Web アプリケーションの設定）

フィルタリングによる HTTP レスポンスの圧縮を使用する場合，Web アプリケーションの設定が必要です。

なお, Web アプリケーションの設定は, cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。フィルタリングによる HTTP レスポンスの圧縮の定義には，フィルタ属性ファイルおよび WAR 属性ファイルを使用します。

フィルタ属性ファイルおよび WAR 属性ファイルで指定するタグは，DD と対応しています。DD（web.xml）での定義については，「2.10.4　DD での定義」および「2.10.5　DD の定義例」を参照してください。

2.11　EJB コンテナとの連携

Web コンテナは EJB コンテナと連携して，J2EE サーバとして動作します。ここでは，Enterprise Bean の呼び出しについて説明します。なお，Enterprise Bean を呼び出すには，ビジネスインタフェースを使用することもできます。

この節の構成を次の表に示します。

表 2-41　この節の構成（EJB コンテナとの連携）

分類	タイトル	参照先
解説	Enterprise Bean の呼び出し方法	2.11.1
実装	EJB コンテナとの連携のための実装	2.11.2
設定	実行環境での設定（J2EE サーバの設定）	2.11.3

注　「運用」および「注意事項」について，この機能固有の説明はありません。

2.11.1　Enterprise Bean の呼び出し方法

Enterprise Bean の呼び出しは，ルックアップを使用する方法と DI を使用する方法があります。ルックアップを使用する場合，呼び出し方法は CORBA ネーミングサービスの切り替え機能を利用するかどうかで異なります。

CORBA ネーミングサービスの切り替え機能を利用する場合，または同じ EAR に含まれる Enterprise Bean を呼び出す場合

呼び出し対象の Enterprise Bean が同一の EAR に含まれている場合，次の例に示すように Enterprise Bean を呼び出します。

呼び出し対象の Enterprise Bean が同一の EAR に含まれていない場合でも，CORBA ネーミングサービスの切り替え機能を使用して，次に示すように Enterprise Bean を呼び出すことができます。

CORBA ネーミングサービスの切り替え機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.10　CORBA ネーミングサービスの切り替え」を参照してください。

例：

```
Context ctx = new InitialContext();
Object o = ctx.lookup("java:comp/env/ejb/cart");
CartHome h = (CartHome)PortableRemoteObject.narrow(o, CartHome.class);
Cart c = h.create();
c.call();
```

CORBA ネーミングサービスの切り替え機能を利用しないで別 EAR に含まれる Enterprise Bean を呼び出す場合

CORBA ネーミングサービスの切り替え機能を利用しない場合で，呼び出し対象の Enterprise Bean が別の EAR に含まれているとき，次の例に示すように Enterprise Bean を呼び出します。例を次に示します。

例：

```
Context ctx = new InitialContext();
Object o =
 ctx.lookup("HITACHI_EJB/SERVERS/MyServer/EJB/APName/Cart");
CartHome h = (CartHome)PortableRemoteObject.narrow(o, CartHome.class);
Cart c = h.create();
c.call();
```

2.11.2　EJB コンテナとの連携のための実装

Enterprise Bean の呼び出しを利用する場合, 該当 Web アプリケーションをデプロイする前に, J2EE サーバから RMI-IIOP のスタブ（stubs.jar），および RMI-IIOP インタフェースを取得し，Web アプリケーションの WEB-INF/lib ディレクトリに格納します。

また，複数の RMI-IIOP のスタブ（stubs.jar）を格納する場合，名前の重複を避けるために WEB-INF/lib ディレクトリに格納する前に適当な名前に変更してください。

なお，RMI-IIOP のスタブについては，J2EE サーバのダイナミッククラスローディングを利用できます。ダイナミッククラスローディングの詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「3.7.3　ダイナミッククラスローディング」を参照してください。

WAR から EJB を呼び出す場合は，WAR の DD に ejb-ref を定義する必要があります。ただし，アノテーションを使用して参照を定義している場合は，web.xml に参照を定義する必要はありません。また，WAR から同一アプリケーション内の EJB を呼び出す場合は，WAR に，スタブおよびリモートインタフェースを含める必要はありません。

なお，WAR から別アプリケーションで動作する EJB を呼び出す場合，または別 J2EE サーバで動作する EJB を呼び出す場合，リモートインタフェースとスタブが必要になります。WAR にリモートインタフェースを含めて，実行環境で J2EE アプリケーションの開始時にスタブが自動生成されるように設定してください。実行環境での設定については，「2.11.3　実行環境での設定（J2EE サーバの設定）」を参照してください。

2.11.3　実行環境での設定（J2EE サーバの設定）

J2EE サーバの設定は，簡易構築定義ファイルで実施します。EJB コンテナとの連携の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の ejbserver.deploy.stub.generation.scope に指定します。このパラメタでは，別アプリケーションの EJB をリモートインタフェースで呼び出すためのスタブを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

2.12 データベースとの接続

Web コンテナを利用している場合，Java EE サービスのリソースアダプタを使用して，データベースにアクセスできます。なお，リソースアダプタを通じてデータベースにアクセスする場合は，DB Connector を使用します。データベースへのアクセス時には，次に示す機能を利用できます。

- JDBC コネクションプーリング
- JDBC コネクションシェアリング
- 分散トランザクション

データベースとの接続の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6　データベースへの接続」を参照してください。データベース接続するためのリソースアダプタの設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6.9　実行環境での設定（リソースアダプタでの設定)」を参照してください。

2.13　Web コンテナによるスレッドの作成

この節では，Web コンテナによるスレッドの作成について説明します。

この節の構成を次の表に示します。

表 2-42　この節の構成（Web コンテナによるスレッドの作成）

分類	タイトル	参照先
解説	作成するスレッドの種類と数	2.13.1
	作成するスレッドの総数	2.13.2

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

Web コンテナでは，Web サーバ連携のためのスレッド，Web アプリケーションのためのスレッドなどを作成します。これらのスレッドに対して，システムのリソースが十分であるか検討してください。

なお，アプリケーションサーバでは，サーブレットおよび JSP からスレッド（ユーザスレッド）を作成して使用することもできます。ユーザスレッドで使用できるアプリケーションサーバの機能の詳細については，「2.14　ユーザスレッドの使用」を参照してください。

ここでは，Web コンテナが作成するスレッドについて説明します。

2.13.1　作成するスレッドの種類と数

Web コンテナが作成するスレッドを次の表に示します。

表 2-43　Web コンテナが作成するスレッド

スレッドの分類	参照先
Web コンテナの簡易 Web サーバ用のスレッド	(1)
Web サーバ連携機能を使用するためのスレッド※1	(2)
インプロセス HTTP サーバを使用するためのスレッド※2	(3)
Web アプリケーションのためのスレッド	(4)
管理用コンテキストのためのスレッド	(5)
レスポンス送信時のタイムアウトを監視するためのスレッド	(6)

注※1

Web サーバ連携機能を使用する場合に作成されるスレッドです。

注※2

インプロセス HTTP サーバを使用する場合に作成されるスレッドです。

(1)　Web コンテナの簡易 Web サーバ用のスレッド（必須）

簡易 Web サーバ用のスレッドとして，Web コンテナが作成するスレッドとスレッド数を次に示します。

作成するスレッド

TCP のコネクション接続要求を受信するスレッド, および受信したリクエストの処理用スレッドを作成します。

スレッドの数

TCP のコネクション接続要求を受信するスレッドは，1 スレッドです。簡易 Web サーバが受信したリクエストの処理用スレッドは，最小 5 スレッド，最大 100 スレッドです。

リクエスト処理用スレッドは，起動時に 5 スレッド作成し，同時実行スレッド数が起動済みスレッド数を超えた場合，100 スレッドまで作成されます。

なお，Web コンテナの簡易 Web サーバは，必ず使用します。

(2) Web サーバ連携機能を使用するためのスレッド

Web サーバとの連携に使用するスレッドとして，Web コンテナが作成するスレッドとスレッド数を次に示します。

作成するスレッド

リダイレクタからのコネクション接続要求を受信するスレッド，およびリダイレクタから受信したリクエストの処理用スレッドを作成します。

スレッドの数

リダイレクタからのコネクション接続要求を受信するスレッドは，1 スレッドです。リダイレクタから受信したリクエストの処理用スレッドは，Web サーバと Web コンテナとのコネクション数分スレッドを作成します。

リダイレクタから受信したリクエスト処理用スレッドは，起動時に，usrconf.properties の webserver.connector.ajp13.max_threads キーに指定した数分作成します（webserver.connector.ajp13.max_threads キーのデフォルトは 10）。

(3) インプロセス HTTP サーバを使用するためのスレッド

インプロセス HTTP サーバ用のスレッドとして，Web コンテナが作成するスレッドとスレッド数を次に示します。

作成するスレッド

リクエスト処理スレッド，およびリクエスト処理スレッド数の監視用スレッドを作成します。

スレッドの数

Web クライアントまたはプロキシサーバからの接続数と同じ数のリクエスト処理スレッドを作成します。リクエスト処理スレッドは，J2EE サーバ起動時に，usrconf.properties の webserver.connector.inprocess_http.init_threads キーに指定した数分作成します（webserver.connector.inprocess_http.init_threads キーのデフォルトは 10）。同時実行スレッド数が J2EE サーバ起動時に作成したスレッド数を超えた場合，Web クライアントからの接続数の最大値を上限として，リクエスト処理スレッドを作成します（webserver.connector.inprocess_http.max_connections キーのデフォルトは 100）。

また，リクエスト処理スレッド数の監視用スレッドを 1 スレッド作成します。

(4) Web アプリケーションのためのスレッド

Web アプリケーション単位に，セッションの有効期限監視スレッドを作成します。最小 1 スレッド，最大 3 スレッドです。

(5) 管理用コンテキストのためのスレッド

管理用コンテキストを二つ生成するため，2 スレッド作成します。

(6)　レスポンス送信時のタイムアウトを監視するためのスレッド

レスポンス送信時のタイムアウトを有効にすると，送信タイムアウトを監視するためのスレッドを 1 スレッド作成します。

2.13.2　作成するスレッドの総数

Web コンテナがデフォルトの設定でプロセス起動時に作成するスレッドの総数を，Web サーバと連携する場合と，インプロセス HTTP サーバを使用する場合に分けて示します。ただし，この数値には Web コンテナ以外のスレッド，および JavaVM が作成するスレッドは含まれていません。

(1)　Web サーバと連携する場合

Web サーバと連携する場合の作成するスレッドの総数について説明します。また，Web サーバ連携に使用するスレッド数についても説明します。

(a)　作成するスレッドの総数

作成するスレッドの総数は，レスポンス送信時のタイムアウトを設定しているかどうかによって異なります。

レスポンス送信時のタイムアウトを設定しているとき

スレッド総数=A ＋ B ＋ C ＋ D ＋ E

レスポンス送信時のタイムアウトを設定していないとき

スレッド総数=A ＋ B ＋ C ＋ D

（凡例）

A：Web コンテナの簡易 Web サーバ用のスレッド数

B：Web サーバとの連携に使用するスレッド数

C：Web アプリケーション単位のスレッド数

D：管理用コンテキストのスレッド数

E：レスポンス送信時のタイムアウトを監視するためのスレッド数

このため，Web サーバ連携の場合のプロセス起動時のスレッド総数は次のようになります。

レスポンス送信時のタイムアウトを設定しているときのスレッド総数

=6 ＋ 11 ＋（1×Web アプリケーション数）＋ 2 ＋ 1

=20 ＋ Web アプリケーション数

レスポンス送信時のタイムアウトを設定していないときのスレッド総数

=6 ＋ 11 ＋（1×Web アプリケーション数）＋ 2

=19 ＋ Web アプリケーション数

プロセス起動後，Web サーバとのコネクション数，簡易 Web サーバの同時実行数によってスレッド数が増加します。

(b)　Web サーバとの連携に使用するスレッド数

Web サーバ連携機能を使用する場合のリクエスト処理スレッドは，Web コンテナ起動時に usrconf.properties の webserver.connector.ajp13.max_threads キーに指定した数分作成し，以降はリダイレクタからのコネクション数分作成します。そのため，リクエスト処理スレッド数の最大数は Web サーバの最大接続数に依存します。

なお，リダイレクターWeb コンテナ間のコネクションがタイムアウトによって切断された場合は，Web サーバの最大接続数以上にリクエスト処理スレッドが作成されます。リクエスト処理用スレッド数の最大値を算出する式を次に示します。

- HTTP Server を使用している場合

  リクエスト処理スレッド数の最大値＝ A ＋ B

  （凡例）

  A：HTTP Server の ThreadsPerChild ディレクティブの設定値

  B：リダイレクターWeb コンテナ間のコネクションがタイムアウトによって切断されたときの，実行中のリクエスト数（最大値は webserver.connector.ajp13.max_threads の設定値）

- Microsoft IIS を使用している場合

  リクエスト処理スレッド数の最大値＝ A×B ＋ C

  （凡例）

  A：Microsoft IIS のスレッド数

  B：Microsoft IIS のプロセス数

  C：リダイレクターWeb コンテナ間のコネクションがタイムアウトによって切断されたときの，実行中のリクエスト数（最大値は webserver.connector.ajp13.max_threads の設定値）

(2)　インプロセス HTTP サーバを使用する場合

作成するスレッドの総数は，レスポンス送信時のタイムアウトを設定しているかどうかによって異なります。

レスポンス送信時のタイムアウトを設定しているとき

スレッド総数=A ＋ B ＋ C ＋ D ＋ E

レスポンス送信時のタイムアウトを設定していないとき

スレッド総数=A ＋ B ＋ C ＋ D

（凡例）

A：Web コンテナの簡易 Web サーバを使用する場合のスレッド数

B：インプロセス HTTP サーバを使用する場合のスレッド数

C：Web アプリケーション単位のスレッド数

D：管理用コンテキストのスレッド数

E：レスポンス送信時のタイムアウトを監視するためのスレッド数

このため，インプロセス HTTP サーバを使用する場合のプロセス起動時のスレッド総数は次のようになります。

レスポンス送信時のタイムアウトを設定しているときのスレッド総数

=6 ＋ 11 ＋（1×Web アプリケーション数）＋ 2 ＋ 1

=20 ＋ Web アプリケーション数

レスポンス送信時のタイムアウトを設定していないときのスレッド総数

=6 ＋ 11 ＋（1×Web アプリケーション数）＋ 2

=19 ＋ Web アプリケーション数

J2EE サーバ起動後，Web クライアントとのコネクション数，簡易 Web サーバの同時実行数によってスレッド数が増加します。

2.14　ユーザスレッドの使用

アプリケーションサーバでは，サーブレットおよびJSP からスレッドを作成して使用できます。ユーザがプログラムの中で明示して作成するスレッドのことを，**ユーザスレッド**といいます。この節では，ユーザスレッドの使用について説明します。

この節の構成を次の表に示します。

表 2-44　この節の構成（ユーザスレッドの使用）

分類	タイトル	参照先
解説	ユーザスレッドでの機能の使用可否	2.14.1
設定	ユーザスレッド生成のための権限の設定	2.14.2

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

2.14.1　ユーザスレッドでの機能の使用可否

ここでは, ユーザスレッドで使用できるアプリケーションサーバの機能について説明します。ユーザスレッドの使用方法については,「6.2.1(15)　ユーザスレッドの使用方法」を参照してください。

アプリケーションサーバで提供される各機能について, ユーザスレッドで使用できるかどうかを次の表に示します。

表 2-45　ユーザスレッドでの機能の使用可否

機能名	使用可否	参照先
サーブレット API の利用	△	(1)
Enterprise Bean の呼び出し	×	−
ネーミングサービス	○	(2)
リソース接続	△	(3)
トランザクションサービス	△	
統合ユーザ管理	×	−
ログ運用および障害検知	○	(4)
J2EE アプリケーション運用	△	(5)
コンテナ拡張ライブラリの利用	○	(6)

（凡例）○：使用できる　△：一部の機能が使用できない　×：使用できない　−：該当なし

ユーザスレッドで使用できる機能をさらに詳細機能に分類し，各機能についてサーブレット/JSP，およびユーザスレッドで使用できるかどうかを示します。また, ユーザスレッドで使用する場合の注意事項についても示します。

(1)　サーブレット API

サーブレット API をユーザスレッドで使用する場合，リクエストオブジェクトおよびレスポンスオブジェクトは使用できません。リクエスト処理スレッドでだけ使用してください。詳細は，サーブレット仕様書の Thread Safety の項を参照してください。

(2)　ネーミングサービス

ネーミングサービスとして提供される機能が，サーブレット/JSP およびユーザスレッドで使用できるかどうかを次の表に示します。

表 2-46　ネーミングサービスの機能の使用可否（ユーザスレッド）

分類／機能名			サーブレット/JSP	ユーザスレッド
JNDI のルックアップ	リソースアダプタ	DB Connector	○	○
		DB Connector for Reliable Messaging と Reliable Messaging	○	○
		TP1/Message Queue-Access	○	○
		TP1 Connector	○	○
	Java Mail		○	○
	JavaBeans リソース		○	○
	ユーザトランザクション		○	○

（凡例）○：使用できる

ネーミングサービスを使用する場合，ユーザスレッドの実行中にアプリケーションを停止しないでください。

(3)　リソース接続およびトランザクションサービス

リソース接続およびトランザクションサービスとして提供される機能が，サーブレット/JSP およびユーザスレッドで使用できるかどうかを次の表に示します。

表 2-47　リソース接続およびトランザクション管理の機能の使用可否（ユーザスレッド）

分類／機能名		サーブレット/JSP	ユーザスレッド
コネクションプーリング	DB Connector によるコネクションプーリング	○	○
	DB Connector for Reliable Messaging と Reliable Messaging によるコネクションプーリング	○	○
	TP1 Connector とのコネクションプーリング	○	○
	TP1/Message Queue - Access とのコネクションプーリング	○	○
	SMTP サーバとのコネクションプーリング	×	×

分類／機能名		サーブレット/JSP	ユーザスレッド
コネクションプーリング	コネクションプールのウォーミングアップ	○	○
	コネクション数調節	○	○
コネクションシェアリング		○	○
コネクションアソシエーション		○	○
ステートメントプーリング（DB Connector）		○	○
ライトトランザクション		○	○
インプロセストランザクションサービス		○	○
DataSource オブジェクトのキャッシング		○	○
DB Connector のコンテナ管理でのサインオンの最適化		○	○
受信バッファのプーリング		○	○
コネクションの障害検知	DB Connector による障害検知	○	○
	DB Connector for Reliable Messaging と Reliable Messaging による障害検知	○	○
	TP1 Connector とのコネクション障害検知	×	×
	TP1/Message Queue - Access とのコネクション障害検知	×	×
	SMTP サーバとのコネクション障害検知	×	×
コネクション枯渇時のコネクション取得待ち		○	○
コネクションの取得リトライ	DB Connector によるコネクション取得リトライ	○	○
	DB Connector for Reliable Messaging と Reliable Messaging によるコネクション取得リトライ	○	○
	TP1 Connector とのコネクション取得リトライ	○	○
	TP1/Message Queue - Access とのコネクション取得リトライ	○	○
	SMTP サーバとのコネクション取得リトライ	×	×
コネクションプールクリア		○	○
コネクションのクローズ・解放	コネクション自動クローズ（Web コンテナ）	○	×
コネクションスイーパ		○	○
トランザクションタイムアウト		○	○
トランザクションリカバリ		○	○
トランザクションの自動決着※		○	×

分類／機能名	サーブレット/JSP	ユーザスレッド
障害調査用 SQL の出力	○	○
クラスタコネクションプール	○	○

（凡例）○：使用できる　×：使用できない

注※

サーブレットのメソッドからリターンするとき，未決着のトランザクションをロールバックする機能です。

ユーザスレッドでリソース接続およびトランザクションサービスを使用する場合の注意事項を示します。

- SingleThreadModel インタフェースを実装したサーブレットで生成したスレッド上で，トランザクションの開始と決着，およびコネクションの取得と解放はできません。
- スレッド生成時にトランザクションを引き継ぐことはできません。
- トランザクションは，同じスレッド上で開始または決着してください。
- スレッド間でコネクションを渡すことはできません。コネクションを使用すると，動作不正となります。
- ユーザスレッドでコネクションを取得した場合，コネクションを取得したスレッド上で確実にコネクションをクローズしてください。

(4)　ログ運用および障害検知

ログ運用および障害検知として提供される機能が，サーブレット/JSP およびユーザスレッドで使用できるかどうかを次の表に示します。

表 2-48　ログ運用および障害検知の機能の使用可否（ユーザスレッド）

分類／機能名	サーブレット/JSP	ユーザスレッド
Management イベントによる処理の自動実行	○	○
JP1 イベントによるシステムの監視	○	○
ユーザログ出力	○	○
性能解析トレース	○	○
CTM の稼働情報の監視	○	○

（凡例）○：使用できる

(5)　J2EE アプリケーション運用

J2EE アプリケーション運用機能として提供される機能が，サーブレット/JSP およびユーザスレッドで使用できるかどうかを次の表に示します。

表 2-49　J2EE アプリケーション運用機能の使用可否（ユーザスレッド）

分類／機能名	サーブレット/JSP	ユーザスレッド
Web コンテナでの同時実行スレッド数の制御	○	×
スケジュールキュー単位の同時実行数の動的変更	○	○

分類／機能名		サーブレット/JSP	ユーザスレッド
J2EE アプリケーションの実行時間監視	メソッドタイムアウト機能	○	×
	メソッドキャンセル機能	○	×
J2EE アプリケーションの停止	通常停止	○	○※1
	強制停止	○	○※1
J2EE アプリケーションの入れ替え	リデプロイ機能による入れ替え	○	○※2
	リロード機能による入れ替え	○	○

（凡例）○：使用できる　×：使用できない

注※1

コンテナでは，ユーザスレッドを停止しません。

注※2

開始状態の J2EE アプリケーションを入れ替えた場合，コンテナではユーザスレッドを停止しません。

(6)　コンテナ拡張ライブラリ

コンテナ拡張ライブラリは，サーブレット/JSP およびユーザスレッドのどちらでも使用できます。

2.14.2　ユーザスレッド生成のための権限の設定

ユーザがプログラムの中で明示して生成するスレッド（ユーザスレッド）を生成するためには，対象となるサーブレットや JSP にスレッドの生成権限を与える必要があります。ここでは，ユーザスレッドを生成するための権限の設定について説明します。

ユーザスレッドを生成するには，server.policy に次の記述があるかどうかを確認してください。この定義によって，ユーザスレッドを生成するための権限が与えられます。

```
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
```

バージョン 07-00 以降に構築したサーバには構築時に設定されています。

なお，server.policy は，Smart Composer 機能のコマンドでシステムを構築したあとに設定してください。server.policy の記述例を次に示します。

```
:
//
// Grant permissions to JSP/Servlet
//
grant codeBase "file:${ejbserver.http.root}/web/${ejbserver.serverName}/-" {
permission java.lang.RuntimePermission "loadLibrary.*";
permission java.lang.RuntimePermission "queuePrintJob";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
permission java.net.SocketPermission "*", "connect";
permission java.io.FilePermission "<<ALL FILES>>", "read, write";
permission java.util.PropertyPermission "*", "read";
permission javax.security.auth.AuthPermission "getSubject";
```

```
permission javax.security.auth.AuthPermission "createLoginContext.*";
};
 :
```

2.15　同時実行スレッド数の制御の概要

Web コンテナでは，マルチスレッドでサーブレットのリクエストを処理します。このとき，同時に実行できるスレッド数に上限を設定できます。これによって，スラッシングなどによるパフォーマンスの低下を防止できます。また，適切なスレッド数を設定することで，アクセス状況に従ったパフォーマンスのチューニングができます。

この節では，同時に実行するスレッド数を制御するための設定について説明します。

この節の構成を次の表に示します。

表 2-50　この節の構成（同時実行スレッド数の制御）

分類	タイトル	参照先
解説	スレッド数を制御する単位	2.15.1
	同時実行スレッド数制御のパラメタ	2.15.2
	静的コンテンツやリクエストのエラー処理に使用されるスレッド数	2.15.3

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

2.15.1　スレッド数を制御する単位

同時に実行するスレッド数を制御するには，Web コンテナ単位で制御する方法，Web アプリケーション単位で制御する方法，および URL グループ単位で制御する方法の 3 種類あります。

- Web コンテナ単位での同時実行スレッド数制御

  Web コンテナ上の Web アプリケーション全体で，同時にリクエストを処理するスレッド数を設定します。詳細については「2.16　Web コンテナ単位での同時実行スレッド数の制御」を参照してください。

- Web アプリケーション単位での同時実行スレッド数制御

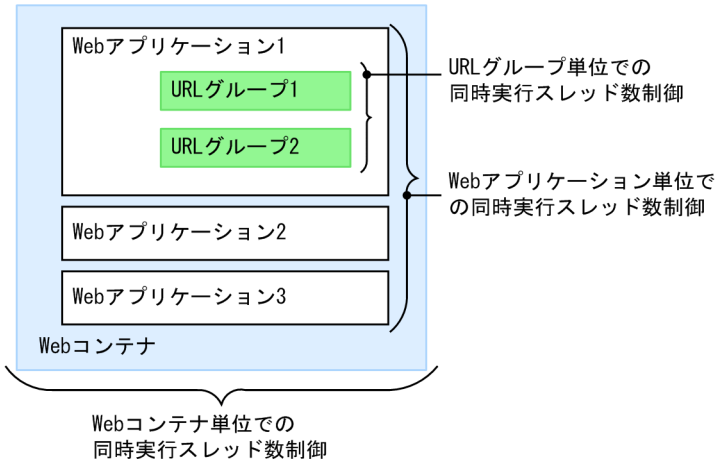
  Web コンテナ上の Web アプリケーションごとに，同時にリクエストを処理するスレッド数を設定します。Web コンテナ単位での同時実行スレッド数制御より細かい単位でスレッド数を制御できます。詳細については，「2.17　Web アプリケーション単位での同時実行スレッド数の制御」を参照してください。

- URL グループ単位での同時実行スレッド数制御

  Web アプリケーション内のサーブレットや JavaBeans などの業務ロジックに対応する URL ごとに，同時にリクエストを処理するスレッド数を設定します。特定の URL へのリクエストを処理する業務ロジックを URL グループといいます。URL グループ単位で，同時実行スレッド数を制御するので，Web アプリケーション単位の制御より細かい単位でスレッド数を制御できます。詳細については，「2.18　URL グループ単位での同時実行スレッド数の制御」を参照してください。

それぞれの制御単位の関係について次の図に示します。

図 2-16　同時実行スレッド数制御の単位の関係



図に示すように，同時実行スレッド数制御のいちばん大きな単位は，Web コンテナ単位となります。Web コンテナ内の Web アプリケーションごとにスレッド数を制御する場合は Web アプリケーション単位で設定します。さらに，Web アプリケーション内の URL グループごとにスレッド数を制御する場合は URL グループ単位で設定します。スレッド数制御の最小単位は URL グループ単位となります。

スレッド数の制御には包含関係があるので，Web アプリケーション単位でスレッド数を制御する場合は Web コンテナ単位での設定も必要になります。また，URL グループ単位でスレッド数を制御する場合は，Web コンテナ単位および Web アプリケーション単位での設定も必要になります。

2.15.2　同時実行スレッド数制御のパラメタ

同時実行スレッド数制御をする場合，最大同時実行スレッド数，占有スレッド数，実行待ちキューサイズなどのパラメタでスレッド数を制御します。

ここでは，スレッド数を制御するための主なパラメタについて説明します。

(1)　最大同時実行スレッド数

最大同時実行スレッド数とは，利用できる全体のスレッド数のうち，同時実行スレッド数制御の対象となるリクエストを最大で同時に実行できるスレッド数です。

最大同時実行スレッド数は，Web コンテナ単位，Web アプリケーション単位，および URL グループ単位で設定します。

(2)　占有スレッド数

占有スレッド数とは，利用できる全体のスレッドのうち，同時実行スレッド数制御の対象となるリクエストを確実に実行できるスレッド数です。Web アプリケーション単位，および URL グループ単位で設定することで，Web アプリケーションごと，または URL グループごとにスレッド数を最低限確保できます。

(3)　実行待ちキューサイズ

同時実行スレッド数制御の対象となるリクエストが，同時実行スレッド数の上限に達した場合に，リクエストが入るキューサイズを指定できます。キューサイズには，キューに格納するリクエストの個数を指定します。

実行待ちキューにリクエストが格納される条件を次に示します。

- 同時実行スレッド数＜最大同時実行スレッド数で，かつ同時実行スレッド数≧占有スレッド数の場合に，使用できる共有スレッド数がないとき
- 同時実行スレッド数≧最大同時実行スレッド数の場合

なお，実行待ちキューに空きがない場合は，リクエストは処理されないで，クライアントにはエラーが返ります。

実行待ちキューサイズは，Web アプリケーション単位，および URL グループ単位で設定できます。

(4)　共有スレッド数

共有スレッド数とは，利用できるスレッドのうち，占有されないスレッド数です。共有スレッド数には，Web コンテナの共有スレッド数と，Web アプリケーションの共有スレッド数があります。

- Web コンテナの共有スレッド数
  Web コンテナの共有スレッド数とは，Web コンテナ上にデプロイされているすべての Web アプリケーションで共有するスレッド数です。
- Web アプリケーションの共有スレッド数
  Web アプリケーションの共有スレッド数とは，Web アプリケーションに含まれるすべての処理で共有するスレッド数です。

なお，共有スレッド数は，最大同時実行スレッド数と占有スレッド数から導き出します。

共有スレッド数の算出のしかたについては，「(5)　共有スレッド数の算出のしかた」を参照してください。

(5)　共有スレッド数の算出のしかた

ここでは，Web コンテナの共有スレッド数と，Web アプリケーションの共有スレッド数の算出のしかたについて説明します。なお，Web アプリケーション単位での同時実行スレッド数制御の設定をしている場合の，Web アプリケーションの共有スレッド数は，URL グループ単位の同時実行スレッド数制御を設定しているかどうかで異なります。

また，URL グループには共有スレッド数はありません。Web アプリケーション内に URL グループ単位の同時実行スレッド数制御を設定している場合は，Web アプリケーション単位の共有スレッド数を使用します。

- **Web コンテナの共有スレッド数**
  Web コンテナ上に，占有スレッド数を設定した Web アプリケーションがある場合の共有スレッド数は，次のとおりです。

  Web コンテナの共有スレッドの総数＝

  　Web コンテナの最大同時実行スレッド数－Web アプリケーション単位の占有スレッド数の合計※

  注※　Web コンテナにデプロイされているすべての Web アプリケーションに設定されている，占有スレッド数の合計となります。

  各 Web アプリケーションに設定した占有スレッド数は，Web アプリケーションで最低限確保するためのスレッド数です。このスレッド数は，ほかの Web アプリケーションのリクエスト処理には使用されません。
- **Web アプリケーションの共有スレッド数（URL グループ単位の同時実行スレッド数制御を設定している場合）**

Web アプリケーションの共有スレッド数＝

Web アプリケーション単位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の合計※

注※　Web アプリケーション内に設定されているすべての URL グループの占有スレッド数の合計です。

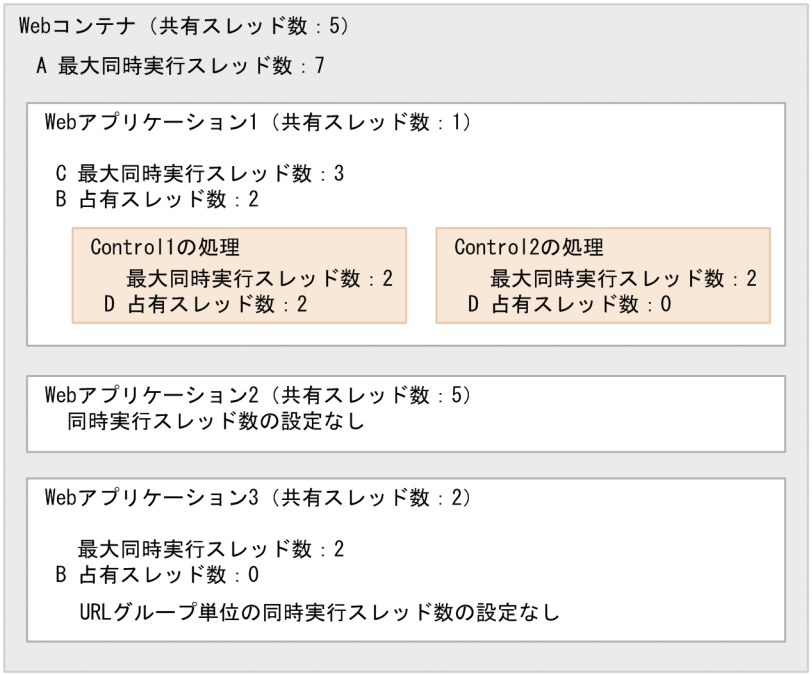
- **Web アプリケーションの共有スレッド数（URL グループ単位の同時実行スレッド数制御を設定していない場合）**

Web アプリケーションの共有スレッド数＝

Web アプリケーション単位の最大同時実行スレッド数

同時実行スレッド数が，次の図に示すとおり設定されている場合の，Web コンテナおよび Web アプリケーションの共有スレッド数の算出例を説明します。

図 2-17　共有スレッド数の算出例



- **Web コンテナの共有スレッド数＝ A－（B の合計）**

この図の場合，Web コンテナの共有スレッド数は 7－（2 ＋ 0）で，5 になります。

- **Web アプリケーション 1 の共有スレッド数＝ C－（D の合計）**

Web アプリケーション 1 には，URL グループ単位の占有スレッド数が指定されています。この図では，共有スレッド数は 3－（2 ＋ 0）で，1 になります。

- **Web アプリケーション 2 の共有スレッド数＝ Web コンテナの共有スレッド数**

Web アプリケーション 2 には，同時実行スレッド数制御の設定はありません。このため，Web アプリケーション 2 の共有スレッド数は，Web コンテナの共有スレッド数が適用されます。この図では，共有スレッド数は 5 になります。

- **Web アプリケーション 3 の共有スレッド数＝最大同時実行スレッド数**

Web アプリケーション 3 には，URL グループ単位の占有スレッド数が指定されていません。このため，Web アプリケーション 3 の共有スレッド数は Web アプリケーション 3 の最大同時実行スレッド数が適用されます。この図では，共有スレッド数は 2 になります。

2.15.3　静的コンテンツやリクエストのエラー処理に使用されるスレッド数

Web サーバの最大同時接続数は，Web コンテナでのリクエスト処理のほかに，Web サーバ上に配置された静的コンテンツの処理，および Web コンテナのリクエストを送信し，同時実行スレッド数と実行待ちキューを超えたリクエストのエラー処理に使用されます。静的コンテンツの処理および実行待ちキューを超えたリクエストのエラー処理に使用されるスレッド数は，次のとおりです。

静的コンテンツやリクエストのエラー処理に使用されるスレッド数＝

Web サーバの最大同時接続数－（Web コンテナ単位の最大同時実行スレッド数＋ Web アプリケーション単位，URL グループ単位およびデフォルトの実行待ちキューサイズの合計）

なお，デフォルトの設定では静的コンテンツに使用されるスレッド数は割り当てられていません。静的コンテンツの処理に使用するスレッド数を確保する場合は，上記の式を満たすように Web アプリケーション単位，URL グループ単位およびデフォルトの実行待ちキューサイズに適切な値を設定してください。

2.16　Web コンテナ単位での同時実行スレッド数の制御

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

この節の構成を次の表に示します。

表 2-51　この節の構成（Web コンテナ単位での同時実行スレッド数の制御）

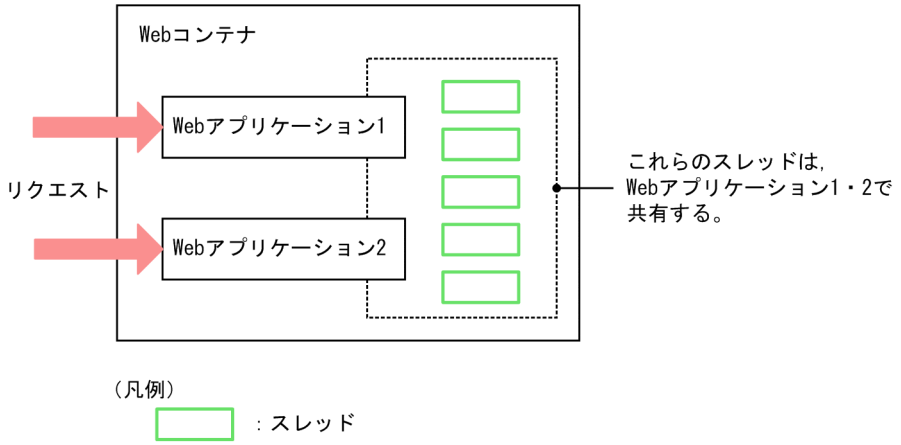
分類	タイトル	参照先
解説	同時実行スレッド数の制御の仕組み（Web コンテナ単位）	2.16.1
設定	実行環境での設定（J2EE サーバの設定）	2.16.2

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

2.16.1　同時実行スレッド数の制御の仕組み（Web コンテナ単位）

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

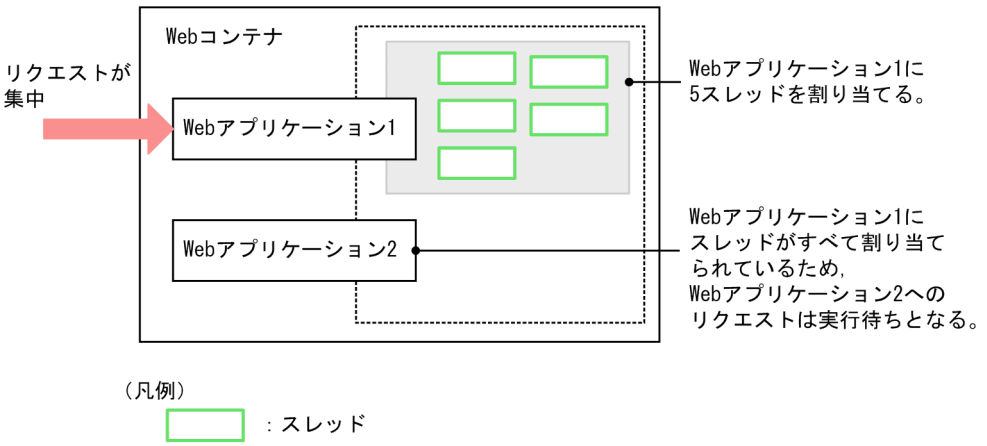
図 2-18　Web コンテナ単位での同時実行スレッド数制御



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

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

図 2-19　アクセスが集中したときのスレッドの扱い（Web コンテナ単位の場合）



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

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

2.16.2　実行環境での設定（J2EE サーバの設定）

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

- webserver.connector.ajp13.max_threads
  Web コンテナ全体の最大同時実行スレッド数を指定します。このパラメタは Web サーバ連携の場合に指定します。
- webserver.connector.inprocess_http.max_execute_threads
  Web コンテナ全体の最大同時実行スレッド数を指定します。このパラメタはインプロセス HTTP サーバを使用する場合に指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

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

この節では，Web アプリケーション単位での同時実行スレッド数の制御について説明します。

Web アプリケーション単位でスレッド数を制御するときには，Web コンテナ単位でのスレッド数制御も同時に設定する必要があります。「2.16　Web コンテナ単位での同時実行スレッド数の制御」もあわせて参照してください。

この節の構成を次の表に示します。

表 2-52　この節の構成（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）

分類	タイトル	参照先
解説	同時実行スレッド数の制御の仕組み（Web アプリケーション単位）	2.17.1
	同時実行スレッド数の制御に必要なパラメタ（Web アプリケーション単位）	2.17.2
	同時実行スレッド数設定の指針（Web アプリケーション単位）	2.17.3
実装	cosminexus.xml での定義	2.17.4
設定	実行環境での設定	2.17.5
	同時実行スレッド数および実行待ちキューサイズの設定例（Web アプリケーション単位）	2.17.6
注意事項	Web アプリケーション単位での同時実行スレッド数制御についての注意事項	2.17.7

注　「運用」について，この機能固有の説明はありません。

2.17.1　同時実行スレッド数の制御の仕組み（Web アプリケーション単位）

Web コンテナ単位より細かい粒度で同時実行スレッド数を制御したい場合，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ます。

Web アプリケーション単位で同時実行スレッド数を設定することで，Web アプリケーションごとの同時実行スレッド数に上限が設けられます。これによって，特定の 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するのを防ぎ，また，ほかの業務も滞りなく実行できるようになります。

なお，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の指針については，マニュアル「アプリケーションサーバ システム設計ガイド」の「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。また，Web コンテナ単位での設定パラメタについては，「2.16.2　実行環境での設定（J2EE サーバの設定）」を参照してください。

2.17.2　同時実行スレッド数の制御に必要なパラメタ（Web アプリケーション単位）

Web アプリケーション単位でスレッド数を制御する場合に必要なパラメタについての概要を次に示します。

- Web コンテナ単位の同時実行スレッド数制御の設定
  Web コンテナ単位の最大同時実行スレッド数を設定します。なお，ここで設定した最大同時実行スレッド数は，Web コンテナ上にデプロイされているすべての Web アプリケーションで共有します。
- Web アプリケーション単位の同時実行スレッド数制御の設定
  同時実行スレッド数を制御したい Web アプリケーションに対して，次に示すパラメタを設定します。
  - 最大同時実行スレッド数
    Web アプリケーションで最大で幾つのスレッドを同時に実行できるかを設定します。
  - 占有スレッド数
    Web アプリケーションの占有スレッド数を設定します。
  - Web アプリケーション単位の実行待ちキューサイズ
    Web アプリケーションの実行待ちキューサイズを設定します。
  - デフォルトの実行待ちキューサイズ
    Web アプリケーション単位の同時実行スレッド数制御を設定していない Web アプリケーションのための，実行待ちキューサイズを設定します。

Web アプリケーション単位の同時実行スレッド数制御の設定に必要なパラメタの詳細について説明します。

(1)　Web アプリケーションの最大同時実行スレッド数

Web アプリケーション単位の最大同時実行スレッド数は，値を設定している場合は，その数が適用されます。ここでは，同時実行スレッド数を設定していない Web アプリケーションの最大同時実行スレッド数，および占有スレッド数を設定していない Web アプリケーションでの最大同時実行スレッド数の考え方について説明します。

(a)　同時実行スレッド数制御を設定していない Web アプリケーションの場合

最大同時実行スレッド数を設定していない Web アプリケーションの，最大同時実行スレッド数は次のとおりです。

最大同時実行スレッド数＝
　Web コンテナの最大同時実行スレッド数－Web アプリケーション単位の占有スレッド数の合計※
　注※
　　Web コンテナにデプロイされているすべての Web アプリケーションに設定されている，占有スレッド数の合計となります。

(b)　占有スレッド数を設定していない Web アプリケーションの場合

Web アプリケーション単位の同時実行スレッド数制御の設定では，占有スレッド数の設定は任意となります。占有スレッド数を設定していない Web アプリケーションの最大同時実行スレッド数は，次のどちらかの値のうち，小さい方の値が適用されます。

最大同時実行スレッド数＝
　Web アプリケーションに設定した最大同時実行スレッド数
　　または
　Web コンテナの最大同時実行スレッド数－Web アプリケーション単位の占有スレッド数の合計※

注※

Web コンテナにデプロイされているすべての Web アプリケーション設定されている，占有スレッド数の合計となります。

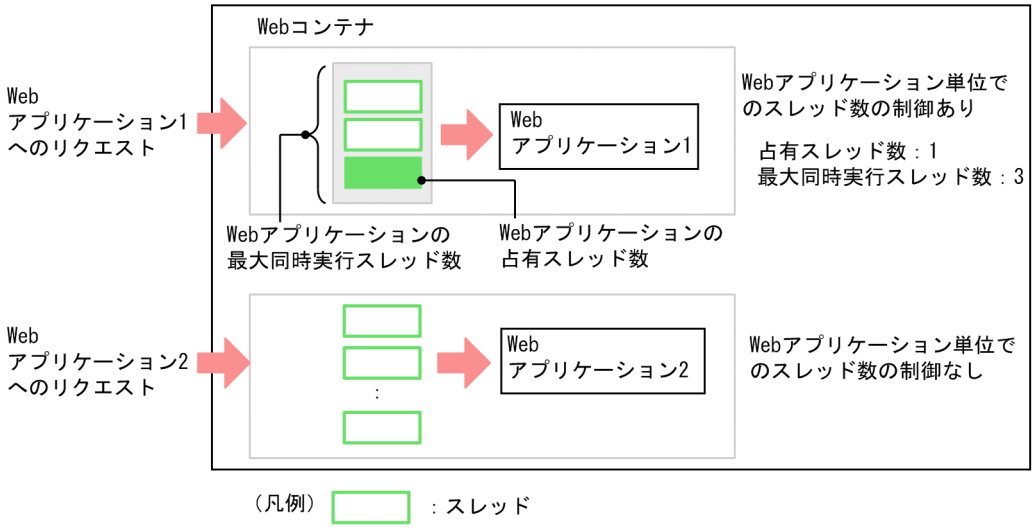
(2)　Web アプリケーションの占有スレッド数

Web コンテナ単位での同時実行スレッド数だけを設定している場合，Web コンテナ内のほかの Web アプリケーションへのアクセスが集中すると，スレッドはアクセスが集中しているアプリケーションに使用されます。占有スレッド数を設定することで，Web アプリケーション内での実行に必要なスレッド数を最低限確保できるので，Web コンテナ内のほかの Web アプリケーションへのアクセスが集中した場合でも，リクエストが待ち状態になることなく，実行できます。

(a)　占有スレッド数の設定の有無による Web アプリケーションの動作

Web アプリケーションの占有スレッド数について次の図に示します。この図を使用して，占有スレッド数を設定した Web アプリケーションと占有スレッド数を設定していない Web アプリケーションの動作を説明します。

図 2-20　Web アプリケーションの占有スレッド数



図の内容について説明します。Web コンテナでは，Web アプリケーション 1 と Web アプリケーション 2 が動作しています。Web アプリケーション 2 では，Web アプリケーションでの同時実行スレッド数制御を設定していませんが，Web アプリケーション 1 では，Web アプリケーションでの同時実行スレッド制御を設定しています。Web アプリケーション 1 の最大同時実行スレッド数は 3 で，占有スレッド数は 1 に設定されています。

例えば，Web アプリケーション 2 にアクセスが集中した場合，Web アプリケーション 1 に占有スレッド数を設定していないと，すべてのスレッドが Web アプリケーション 2 に使用されます。図のように Web アプリケーション 1 に占有スレッド数を設定することで，Web アプリケーション 2 へのアクセスが集中したときでも，Web アプリケーション 1 では 1 スレッドは最低限確保できます。これによって，占有スレッド数を設定している Web アプリケーション 1 の処理を確実に実行できます。

このように，占有スレッド数を設定しておくと，ほかの業務へのアクセスが集中した場合でも確実に実行できます。このため，管理用アプリケーションなどの重要度の高い Web アプリケーションに対して占有スレッド数を設定しておくことをお勧めします。

なお，占有スレッド数に指定した数のスレッドは，ほかの Web アプリケーションのリクエスト処理には使用されません。また，Web アプリケーション単位の同時実行スレッド制御の設定では，占有スレッド数の設定は任意となります。

(b)　占有スレッド数と最大同時接続数

Web サーバの最大同時接続数（Web サーバ連携機能を使用する場合）または Web クライアントからの最大同時接続数（インプロセス HTTP サーバを使用する場合）が少ないときに，占有スレッド数を設定していない Web アプリケーションへのリクエストによって最大同時接続数が占有されると，占有スレッド数を設定した Web アプリケーションにアクセスしても，Web サーバ上またはインプロセス HTTP サーバ上で実行待ちになったり，エラーになったりします。

占有スレッド数を設定した Web アプリケーションを，ほかの Web アプリケーションへのアクセス流量に依存しないで確実に実行させるためには，最大同時接続数に適切な値を設定する必要があります。最大同時接続数の設定方法を次に示します。

- **Web サーバの最大同時接続数の設定方法（Web サーバ連携機能を使用する場合）**

  Web サーバ連携機能を使用する場合，Web サーバの最大同時接続数を，次に示す値よりも大きい値にする必要があります。

  Web サーバの最大同時接続数＞ Web アプリケーション単位およびデフォルトの実行待ちキューサイズの総和＋ Web コンテナ単位の最大同時実行スレッド数

  占有スレッド数を設定する場合は，上記の式を満たすように適切な値を設定してください。

  なお，Web サーバの最大同時接続数は，次の個所に設定されています。

  HTTP Server を使用している場合

  httpsd.conf の ThreadsPerChild ディレクティブ（Windows のとき）または httpsd.conf の MaxClients ディレクティブ（UNIX のとき）

  Microsoft IIS を使用している場合

  ［インターネット サービス マネージャ］で設定したクライアントとの接続数

  HTTP Server を使用している場合の設定の詳細については，マニュアル「HTTP Server」を参照してください。Microsoft IIS を使用している場合の設定の詳細については，Microsoft IIS のヘルプを参照してください。

- **Web クライアントからの最大同時接続数の設定方法（インプロセス HTTP サーバを使用する場合）**

  インプロセス HTTP サーバを使用する場合，Web クライアントからの最大同時接続数を次に示す値よりも大きい値にする必要があります。

  Web クライアントからの最大同時接続数＞ Web アプリケーション単位およびデフォルトの実行待ちキューサイズの総和＋ Web コンテナ単位の最大同時実行スレッド数

  なお，Web クライアントからの同時接続数とは，Web クライアントからの最大接続数から，接続を拒否するリクエスト数を引いた値です。詳細については，「5.5　Web クライアントからの同時接続数の制御によるリクエストの流量制御」を参照してください。

(3)　Web アプリケーションの実行待ちキューサイズ

Web アプリケーションの実行待ちキューサイズを設定します。

Web アプリケーションに最大同時実行スレッド数を設定している場合，実行スレッド数が最大数に達してしまうと，リクエストはキューにためられます。このときの，実行待ちのキューのサイズを，Web アプリケーション単位で指定できます。

Web アプリケーション単位の実行待ちキューサイズの設定は，Web アプリケーション単位での同時実行スレッド数を設定しているかどうかで異なります。

- Web アプリケーション単位で同時実行スレッド数を設定している場合
  Web アプリケーションごとに実行待ちキューサイズを設定できます。
- Web アプリケーション単位で同時実行スレッド数を設定していない場合
  Web アプリケーションで共通となる実行待ちキューサイズが使用されます。共通の実行待ちキューサイズのことを，**デフォルトの実行待ちキューサイズ**といいます。

Web アプリケーション単位およびデフォルトの実行待ちキューでの動作

実行待ちキューの設定が，Web アプリケーションの実行待ちキューサイズとデフォルトの実行待ちキューサイズを含め，複数ある場合，共有スレッド数を使用して実行されるリクエストの順序は，キューのリクエストのうち，先に到着したリクエストから処理されます。

Web サーバの最大同時接続数と Web アプリケーション単位およびデフォルトの実行待ちキュー（Web サーバ連携機能を使用する場合）

Web サーバから Web コンテナに転送されるリクエストの多重度は，Web サーバの最大同時接続数が上限となります。このため，占有スレッド数を設定する場合には，Web アプリケーション単位およびデフォルトの実行待ちキューサイズは Web サーバの最大同時接続数より少ない数にしてください。

Web クライアントからの最大同時接続数と Web アプリケーション単位およびデフォルトの実行待ちキュー（インプロセス HTTP サーバを使用する場合）

Web クライアントからの接続数の多重度の上限は次に示す値となります。

Web クライアントからの最大同時接続数－接続を拒否するリクエスト数

このため，占有スレッド数を設定する場合には，Web アプリケーション単位およびデフォルトの実行待ちキューサイズは Web クライアントからの最大同時接続数より少ない値にする必要があります。

なお，Web クライアントからの最大同時接続数が「Web アプリケーション単位およびデフォルトの実行待ちキューサイズの総和＋ Web コンテナ単位の最大同時実行スレッド数」よりも大きい場合，インプロセス HTTP サーバによる Web クライアントからの同時実行数の制御は不要です。また，Web クライアントからの最大同時接続数が「Web アプリケーション単位およびデフォルトの実行待ちキューサイズの総和＋ Web コンテナ単位の最大同時実行スレッド数」よりも小さい場合，インプロセス HTTP サーバによる Web クライアントからの同時実行数の制御を実行することによって，クライアントに対して接続待ちを発生させないで即座にエラーを返すことができます。

ポイント

同時実行スレッド数が最大数に達したときのリクエストの動作

Web アプリケーション単位の同時実行スレッド数が最大数に達した場合で，キューに空きがあるときは，リクエストはキューにたまります。そして，処理中のリクエスト処理が完了したあとに，順次キューからリクエストが取り出され，実行されます。なお，キューに空きがないときは，リクエストはエラーとなり，クライアントには，HTTP 503 エラーが返ります。

2.17.3　同時実行スレッド数設定の指針（Web アプリケーション単位）

Web アプリケーション単位の同時実行スレッド数制御をする場合の設定の指針について説明します。

- 設定する値は，次の順に決定してください。
  1. Web コンテナ単位の最大同時実行スレッド数を決定する。
  2. Web アプリケーション単位の最大同時実行スレッド数を決定する。
  3. Web アプリケーション単位の占有スレッド数を決定する。

4. Web アプリケーション単位およびデフォルトの実行待ちキューサイズを決定する。

- Web アプリケーション単位の同時実行スレッド数の設定は，各 Web アプリケーションの運用，および J2EE サーバが動作するホストの処理能力を考慮して設定してください。設定値が妥当であるかどうかは，Web アプリケーションを実行し，評価する必要があります。

  なお，実行中の Web アプリケーションの稼働情報は，Management Server を利用して確認できます。Web アプリケーションの稼働状況の確認については，「2.19.2(1)　Web アプリケーションの稼働状況の確認」を参照してください。
- 複数の Web アプリケーションに占有スレッド数を設定する場合，各 Web アプリケーションの占有スレッド数の合計は，Web コンテナの最大同時実行スレッドの値以下にする必要があります。設定値については，Web コンテナ単位の同時実行スレッド数の制御とあわせて検討してください。
- Web コンテナ単位の最大同時実行スレッド数が少ない場合，Web アプリケーション単位の最大同時実行スレッド数は，設定した数よりも少なくなるおそれがありますので，注意してください。
- 占有スレッド数は，Web アプリケーションの最大同時実行スレッド数以下にしてください。

2.17.4　cosminexus.xml での定義

アプリケーションの開発環境で必要な cosminexus.xml での定義について説明します。

Web アプリケーション単位でのスレッド数の制御を使用するための定義は，cosminexus.xml の<war>タグ内に指定します。

cosminexus.xml での Web アプリケーション単位でのスレッド数の制御の定義について次の表に示します。

表 2–53　cosminexus.xml での Web アプリケーション単位でのスレッド数の制御の定義

項目	指定するタグ	設定内容
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>タグ内の<thread-control-max-threads>タグ	Web アプリケーションで，最大で幾つのスレッドを同時に実行できるかを指定できます。
Web アプリケーション単位の占有スレッド数	<thread-control>タグ内の<thread-control-exclusive-threads>タグ	Web アプリケーションで，最低限確保するスレッド数（**占有スレッド数**）を指定できます。 なお，占有スレッド数を設定しない場合は，0 を指定します。
Web アプリケーション単位の実行待ちキュー	<thread-control>タグ内の<thread-control-queue-size>タグ	実行スレッド数が最大数に達した場合，リクエストはキューに溜められます。このときの，実行待ちキューサイズを Web アプリケーション単位で指定します。

注※　Web アプリケーション単位の同時実行スレッド数の設定は，運用管理コマンド（mngsvrutil）を利用して，動的に変更することもできます。これによって，稼働中の Web アプリケーションのサービスを停止することなく，同時実行スレッド数の設定を変更できます。Web アプリケーションの最大同時実行スレッド数の動的変更については，「2.19 同時実行スレッド数の動的変更」を参照してください。

指定するタグの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「2.2.6　War 属性の詳細」を参照してください。

2.17.5　実行環境での設定

ここでは，Web アプリケーション単位でスレッド数を制御する場合の設定について説明します。

Web アプリケーション単位でスレッド数を制御する場合，J2EE サーバおよび J2EE アプリケーションの設定が必要です。

なお，J2EE アプリケーションの設定は，cosminexus.xml を含まない J2EE アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

(1)　J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web アプリケーション単位でのスレッド数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Web アプリケーション単位でのスレッド数の制御の定義について次の表に示します。

表 2-54　簡易構築定義ファイルでの Web アプリケーション単位でのスレッド数の制御の定義

項目	指定するパラメタ	設定内容
同時実行スレッド数の制御の有無	webserver.container.thread_control.enabled	Web アプリケーション単位で同時実行スレッド数を制御するかどうかを指定します。「true」を指定すると，Web アプリケーション単位での同時実行スレッド数制御が有効になります。なお，「false」を指定すると Web アプリケーション単位での同時実行スレッド数制御が無効となり，Web コンテナ単位で同時実行スレッド数が制御されます。
デフォルトの実行待ちキューサイズ	webserver.container.thread_control.queue_size	実行スレッド数が最大数に達した場合，リクエストはキューに溜められます。このときの，Web アプリケーションで共通となる実行待ちキューサイズ（デフォルトの実行待ちキューサイズ）を指定します。 設定した値は，Web アプリケーション単位での同時実行数を指定していない場合に適用されます。

簡易構築定義ファイル，指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　J2EE アプリケーションの設定

実行環境での J2EE アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。Web アプリケーション単位でのスレッド数の制御の定義には，WAR 属性ファイルを使用します。

WAR 属性ファイルで指定するタグは，cosminexus.xml と対応しています。cosminexus.xml での定義については，「2.17.4　cosminexus.xml での定義」を参照してください。

2.17.6　同時実行スレッド数および実行待ちキューサイズの設定例（Web アプリケーション単位）

同時実行スレッド数および Web アプリケーション単位の実行待ちキューサイズの設定例について説明します。ここでは，Web サーバ連携機能を使用する場合の例について説明します。

(1)　説明で使用する Web アプリケーションの設定例

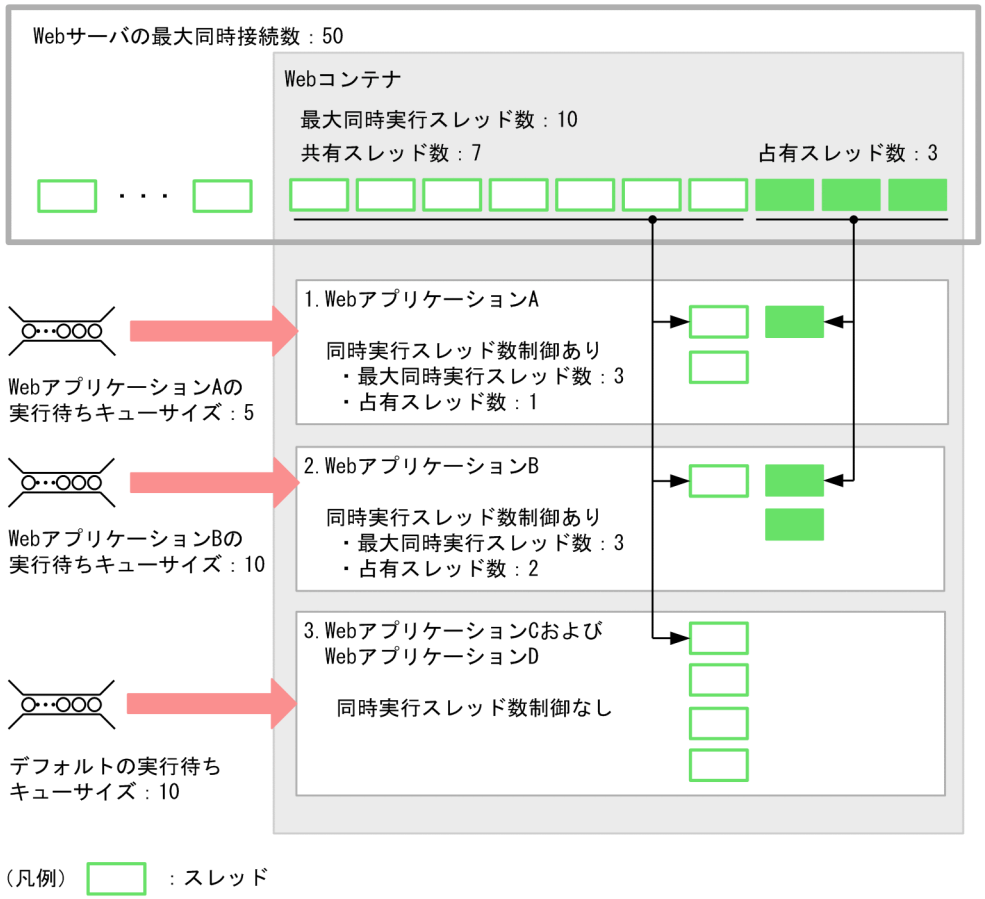
この例では，Web コンテナに四つの Web アプリケーションがデプロイされていて，そのうちの二つの Web アプリケーションに Web アプリケーション単位での同時実行スレッド数制御がされているものとします。設定内容を次に示します。

- Web サーバの最大同時接続数：50
- Web コンテナ単位の最大同時実行スレッド数：10
- デフォルトの実行待ちキューサイズ：10
- Web アプリケーション単位の同時実行スレッド数制御の設定
  Web アプリケーション A および B に，それぞれ次の設定がされているものとします。
  なお，Web アプリケーション C および D には，Web アプリケーション単位の同時実行スレッド数制御の設定はありません。

Web アプリケーション名	最大同時実行スレッド数	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
Web アプリケーション A	3	1	5
Web アプリケーション B	3	2	10

Web コンテナ上に，Web アプリケーションごとの同時実行スレッド数制御の設定がある Web アプリケーションと，設定のない Web アプリケーションが混在している場合の例を，次の図に示します。

図 2-21　Web アプリケーションの設定例



(2)　各 Web アプリケーションで使用できるスレッド数

図 2-21 の設定の場合に使用できる最大同時実行スレッド数，占有スレッド数，および実行待ちキューサイズについて，Web アプリケーションごとに説明します。なお，説明の項番は，図中の項番と対応しています。

1. Web アプリケーション A

- 最大同時実行スレッド数および占有スレッド数

  Web アプリケーション A では，最大同時実行スレッド数および占有スレッド数が設定されているので，それぞれ設定された値までスレッド数を使用できます。

  Web アプリケーション A の同時に実行できるスレッド数は最大で 3 スレッドです。そのうち，1 スレッドは，Web アプリケーション A で最低限確保できる占有スレッド数となります。

- Web アプリケーション単位の実行待ちキューサイズ

  Web アプリケーション A では Web アプリケーション単位の実行待ちキューサイズを指定しています。Web アプリケーション A で同時に 3 スレッド使用している場合，Web アプリケーション A へのリクエストは，最大で 5 個，キューにたまります。

2. Web アプリケーション B

- 最大同時実行スレッド数および占有スレッド数

  Web アプリケーション B では，最大同時実行スレッド数および占有スレッド数が設定されているので，それぞれ設定された値までスレッド数を使用できます。

Web アプリケーション B の同時に実行できるスレッド数は最大で 3 スレッドです。そのうち, 2 スレッドは，Web アプリケーション B で最低限確保できる占有スレッド数となります。

- Web アプリケーション単位の実行待ちキューサイズ

  Web アプリケーション B では Web アプリケーション単位の実行待ちキューサイズを指定しています。Web アプリケーション B で同時に 3 スレッド使用している場合，Web アプリケーション B へのリクエストは，最大で 10 個，キューにたまります。

3. Web アプリケーション C および Web アプリケーション D

Web アプリケーション C および Web アプリケーション D では，Web アプリケーション単位の同時実行スレッド制御を設定していません。

このため，次のように動作します。

- 最大同時実行スレッド数

  Web アプリケーション C および Web アプリケーション D では, Web コンテナの共有スレッド数を使用します。この場合，Web コンテナの共有スレッド数は 7 であるため，Web アプリケーション C および Web アプリケーション D で，合わせて最大で 7 スレッド使用できます。

  なお，スレッド数は，Web アプリケーション C および Web アプリケーション D で共有することになります。

  また，同時実行スレッド数制御を設定していないため，占有スレッド数はありません。Web アプリケーション A および Web アプリケーション B へのアクセスが集中し，使用できるスレッドがなくなると, Web アプリケーション C および Web アプリケーション D の処理は実行待ちになります。

- デフォルトの実行待ちキューサイズ

  Web アプリケーション C または Web アプリケーション D で実行待ちが発生した場合，これらの Web アプリケーションへのリクエストは，キューにたまります。Web アプリケーション C および Web アプリケーション D では Web アプリケーション単位の実行待ちキューサイズを指定していないため，キューはデフォルトの実行待ちキューにためられます。最大で 10 個のリクエストがキューにたまります。

参考

Web コンテナでは，スレッドは静的コンテンツやリクエストのエラー処理にも使用されます。これらの目的で使用されるスレッド数は，次の式から求められます。

Web サーバの処理スレッド数－（Web コンテナ単位の最大同時実行スレッド数＋実行待ちキューサイズの総和※）

注※　実行待ちキューサイズの総和とは，この図の場合，Web コンテナ，Web アプリケーション A，Web アプリケーション B の実行待ちキューサイズを足した値となります。

このため，この図の例の場合，50－（10 ＋（10 ＋ 5 ＋ 10））となり，静的コンテンツやリクエストのエラー処理に使用されるスレッド数は，15 スレッドとなります。

2.17.7　Web アプリケーション単位での同時実行スレッド数制御についての注意事項

Web アプリケーション単位での同時実行スレッド数を制御する場合，次のことに注意してください。

- スレッド数が，Web コンテナ単位の最大同時実行スレッド数を上回る場合

  次の条件すべてを満たすときは，Web アプリケーションで最低限確保されているスレッド数だけリクエストが実施されるため，スレッド数が，一時的に Web コンテナ単位の最大同時実行スレッド数を上回るおそれがあります。

  - アクセスが集中したときなど，Web コンテナ単位の同時実行スレッド数に設定した数すべてを使用しているとき

- 占有スレッド数を設定した Web アプリケーションをデプロイしているとき
- 実行中のリクエスト処理スレッドが完了する前に，デプロイした Web アプリケーションにリクエストがあったとき

- 制御の対象となるアクセス

  サーブレット，JSP のほかに，静的コンテンツへのアクセスも，同時実行スレッド数の制御の対象となります。

- エラーページのカスタマイズ機能使用時の注意

  web.xml に<error-page>タグを使用した，エラーページのカスタマイズ機能を使用している場合，エラー発生時に転送されたリクエスト処理は，同時実行スレッド数制御の対象外となります。同時実行スレッド数の制御による制限はありません。

- スレッドについて

  占有スレッド数は，Web アプリケーション専用のスレッドとして生成されるものではありません。ここで設定するスレッド数は，Web アプリケーションを実行するに当たり，最低限確保されるスレッド数です。

  なお，リクエスト処理用のスレッドは，Web コンテナ全体で再利用されます。

- Web アプリケーション単位の最大同時実行スレッド数について

  Web アプリケーション単位の最大同時実行スレッド数が，Web コンテナ単位の最大同時実行スレッド数より大きな値が設定されていても，Web アプリケーションを J2EE サーバにデプロイできます。ただし，デプロイされた Web アプリケーションの最大同時実行スレッド数は，Web コンテナ単位の最大同時実行スレッド数になるので，注意してください。

- Web アプリケーション単位の共有スレッド数が 0 以下になる場合について

  次のような Web アプリケーションをデプロイすることで，Web アプリケーション単位の共有スレッド数が 0 以下になることがあります。Web アプリケーション単位の共有スレッド数が 0 以下になると，Web アプリケーションのデプロイに失敗するので注意してください。次のような場合で共有スレッド数が 0 以下になることが考えられます。

  - すでに URL グループ単位の同時実行スレッド数制御を設定している Web アプリケーションがデプロイされていて，さらに占有スレッド数を設定した Web アプリケーションをデプロイしようとしている場合
  - デプロイする Web アプリケーションで，占有スレッド数，および URL グループ単位の同時実行スレッド数制御を設定していて，設定段階ですでに Web アプリケーション単位の共有スレッド数が 0 以下となっている場合

2.18　URL グループ単位での同時実行スレッド数の制御

この節では，URL グループ単位での同時実行スレッド数の制御について説明します。

なお，URL グループ単位でスレッド数を制御するときには，Web コンテナ単位，および Web アプリケーション単位でのスレッド数制御も同時に設定する必要があります。「2.16　Web コンテナ単位での同時実行スレッド数の制御」および「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」もあわせて参照してください。

この節の構成を次の表に示します。

表 2-55　この節の構成（URL グループ単位での同時実行スレッド数の制御）

分類	タイトル	参照先
解説	同時実行スレッド数の制御の仕組み（URL グループ単位）	2.18.1
	URL パターンのマッピング処理	2.18.2
	同時実行スレッド数の制御に必要なパラメタ（URL グループ単位）	2.18.3
	同時実行スレッド数設定の指針（URL グループ単位）	2.18.4
実装	cosminexus.xml での定義	2.18.5
設定	実行環境での設定（Web アプリケーションの設定）	2.18.6
	同時実行スレッド数および実行待ちキューサイズの設定例（URL グループ単位）	2.18.7

注　「運用」および「注意事項」について，この機能固有の説明はありません。

2.18.1　同時実行スレッド数の制御の仕組み（URL グループ単位）

Web アプリケーション単位より細かい粒度で同時実行スレッド数制御をしたい場合，URL グループ単位で同時実行スレッド数を制御します。

一つの Web アプリケーション内に処理に時間が掛かる業務ロジック（サーブレットや JavaBeans）がある場合，Web アプリケーション内の同時実行スレッド数のほとんどが使用されてしまい，ほかの業務ロジックの処理が待ち状態になるおそれがあります。このような場合に URL グループ単位の同時実行スレッド数制御を設定することで，Web アプリケーション内の業務ロジックがほかの処理に影響を与えることなく実行できます。

なお，URL グループ単位の同時実行スレッド数の設定の指針については，マニュアル「アプリケーションサーバ システム設計ガイド」の「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。

2.18.2　URL パターンのマッピング処理

同時実行スレッド数制御で制御できる URL，および URL パターンに指定した URL とリクエスト URL とのマッピングの順序について説明します。また，welcome ファイルに対して URL パターンを設定している場合についても説明します。

(1)　同時実行スレッド数制御で制御できる URL

URL グループ単位での同時実行スレッド数制御で制御できる URL とは，RFC 2616 で定義されているリクエスト URI を指します。URL グループ単位での同時実行スレッド数制御で制御できる URL は，リクエスト URI のスキーム，ホスト，ポート，およびクエリ文字列を含みません。

URL グループ単位での同時実行スレッド数制御で制御できる URL の例を次に示します。

受信した HTTP リクエストのリクエスト URI：

http://localhost/webapp/index.html?id=0001

URL グループ単位の同時実行スレッド数制御で使用する部分：

/webapp/index.html

(2)　マッピングの順序

リクエスト URL は，次の 1.～3.の順序でマッピングされます。なお，マッピングの順序は，Servlet 仕様のサーブレットマッピングの適用順序と同じです。

1. 完全一致
   リクエスト URL と URL パターンが完全に一致する場合，一致した URL パターンが適用されます。
2. プリフィックス一致
   リクエスト URL とプリフィックスが一致し，さらに，リクエスト URL とできるだけ長く文字列が一致する URL パターンが適用されます。
3. 拡張子一致
   リクエスト URL と拡張子が一致する場合，一致した URL パターンが適用されます。

なお，上記の 1.～3.に一致しない場合は，URL グループ単位の同時実行スレッド数制御の対象にはなりません。このようなリクエスト URL は，Web アプリケーション単位の同時実行スレッド数制御の対象になります。

次に示す URL パターンを使用して，URL のマッピングの例を説明します。

表 2-56　URL パターンの例

URL パターン	URL パターンに対応する URL グループ
/foo/bar	Control1
/foo/*	Control2
/foo/bar/*	Control3
*.do	Control4

マッピング例 1：リクエスト URL が「/foo/bar」の場合

Control1 の URL パターンと完全一致するため，Control1 に振り分けられます。

マッピング例 2：リクエスト URL が「/foo/bb」の場合

完全一致する URL パターンがないため，プリフィックスが一致する Control2 に振り分けられます。

マッピング例 3：リクエスト URL が「/foo/aa.do」の場合

この場合，Control2 と Control4 で次の個所が一致します。

- プリフィックス「/foo」が Control2 と一致します。

- 拡張子「.do」が Control4 と一致します。

マッピング順序では，拡張子一致よりプリフィックス一致が優先されるため，Control2 に振り分けられます。

マッピング例 4：リクエスト URL が「/foo/bar/」の場合

この場合，Control2 と Control3 でそれぞれプリフィックスが一致します。

- プリフィックス「/foo」が Control2 と一致します。
- プリフィックス「/foo/bar」が Control3 と一致します。

より長い文字列で一致する Control3 に振り分けられます。

マッピング例 5：リクエスト URL が「/foo/bar/action.do」の場合

この場合，Control2，Control3，Control4 で次の個所が一致します。

- プリフィックス「/foo」が Control2 と一致します。
- プリフィックス「/foo/bar」が Control3 と一致します。
- 拡張子「.do」が Control4 と一致します。

マッピング順序では拡張子一致よりプリフィックス一致が優先され，さらにより長い文字列が一致する URL パターンが優先されるので，Control3 に振り分けられます。

マッピング例 6：リクエスト URL が「/context/fo」の場合

該当する URL パターンがないため，Web アプリケーション単位での同時実行スレッド制御として扱われます。

マッピング例 7：リクエスト URL が「/action.do」の場合

Control4 と拡張子が一致するため，Control4 に振り分けられます。

マッピング例 8：リクエスト URL が「/boo/action.do」の場合

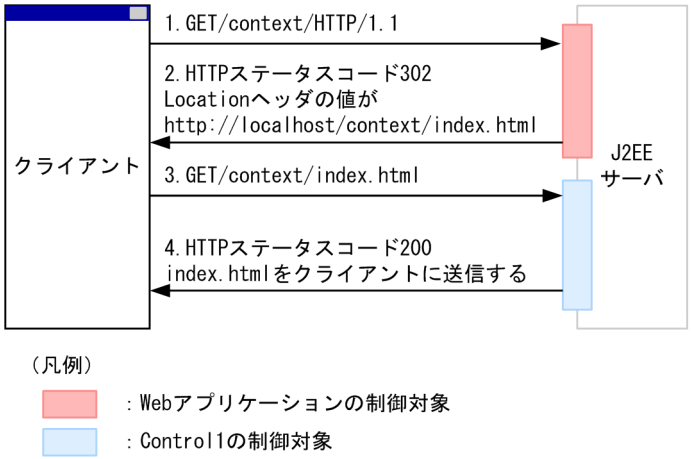
Control4 と拡張子が一致するため，Control4 に振り分けられます。

(3)　welcome ファイルに URL パターンを設定しているときの処理の流れ

welcome ファイルに URL パターンを設定しているときの処理の流れについて，次の図で説明します。この図の例では，Web アプリケーションのコンテキストルートは context とし，welcome ファイルは web.xml で/index.html が設定されているものとします。また，URL パターンと URL グループ単位での同時実行スレッド数制御の定義名は次のとおりとします。

- URL パターン：/index.html
- URL グループ単位での同時実行スレッド数制御の定義名：Control1

図 2-22　welcome ファイルに URL パターンを設定する場合の例



図の流れについて次に説明します。

1. クライアントは http://localhost/context/にリクエストを送信する。
2. J2EE サーバでは，web.xml に設定された welcome ファイルに対して再度クライアントにアクセスしてもらうため，HTTP ステータスコード 302 を返す。また，Location ヘッダに http://localhost/context/index.html を含める。
3. HTTP レスポンスを受信したクライアントは，Location ヘッダの値（http://localhost/context/index.html）にリクエストを送信する。
4. 該当する Web アプリケーションのリクエストが/index.html なので，Control1 の URL グループ単位の同時実行スレッド数制御の制御対象となる。リクエスト処理終了後，クライアントに対してレスポンスとして index.html を送信する。

2.18.3　同時実行スレッド数の制御に必要なパラメタ（URL グループ単位）

ここでは，URL グループ単位でスレッド数を制御する場合の設定について説明します。URL グループ単位でスレッド数を制御する場合は，次の設定が必要になります。

1. **Web コンテナ単位の同時実行スレッド数制御の設定**

   Web コンテナ単位の最大同時実行スレッド数などを設定します。なお，ここで設定した最大同時実行スレッド数は，Web コンテナ上にデプロイされているすべての Web アプリケーションで共有します。

   Web コンテナ単位での同時実行スレッド数制御の設定については，「2.16　Web コンテナ単位での同時実行スレッド数の制御」を参照してください

2. **Web アプリケーション単位の同時実行スレッド数制御の設定**

   Web アプリケーション単位の同時実行スレッド数や，占有スレッド数などを設定します。なお，URL グループ単位での同時実行スレッド数を設定する場合，Web アプリケーション単位での占有スレッド数の設定は必須になります。

   Web アプリケーション単位での同時実行スレッド数制御の設定については，「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。

3. **URL グループ単位の同時実行スレッド数制御の設定**

URL グループ単位の同時実行スレッド数制御は，Web アプリケーション単位での同時実行スレッド数制御を設定している Web アプリケーション内の URL グループに対して，次に示すパラメタを設定します。

- URL グループ単位の同時実行スレッド数制御の定義名
  同時実行スレッド数制御の単位となる URL グループの名称を設定します。
- 最大同時実行スレッド数
  URL グループで最大で幾つのスレッドを同時に実行できるかを設定します。
- 占有スレッド数
  URL グループの占有スレッド数を設定します。
- URL グループ単位の実行待ちキューサイズ
  URL グループの実行待ちキューサイズを設定します。
- URL パターン
  スレッド数制御の対象となるリクエスト URL に振り分けるための，URL パターンを設定します。

以降で，URL グループ単位での同時実行スレッド数制御の設定項目の詳細を説明します。

(1)　URL グループ単位の最大同時実行スレッド数

URL グループ単位で使用するスレッド数は，URL グループが属する Web アプリケーションのスレッド数です。このため，URL グループ単位でスレッドを一つ使用している場合，その URL グループを含む Web アプリケーションでもスレッドが一つ実行されていることになります。

なお，URL グループ単位の最大同時実行スレッド数を設定していないリクエスト URL については，Web アプリケーション単位の共有スレッド数が使用されます。Web アプリケーション単位の共有スレッド数は，次のようになります。

Web アプリケーション単位の共有スレッド数＝

Web アプリケーション単位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の合計

注※　ここでの最大同時実行スレッド数の値には，次の 1.と 2.のうち，小さい方の値が適用されます。

1. Web コンテナ単位の共有スレッド数
2. Web アプリケーション単位の最大同時実行スレッド数に設定した値

このとき，Web アプリケーション単位の共有スレッド数は 1 以上である必要があります。共有スレッド数が 0 以下になる場合は，エラーが発生します。詳細は，「2.17.7　Web アプリケーション単位での同時実行スレッド数制御についての注意事項」を参照してください。

(2)　URL グループ単位の占有スレッド数

URL グループ単位の占有スレッド数は，特定の業務ロジックを Web アプリケーション内のほかの業務ロジックの影響を受けないで実行させるために設定します。

URL グループ単位の占有スレッド数は，Web アプリケーションで設定されている占有スレッド数の範囲で指定します。このため，URL グループが含まれる Web アプリケーションで占有スレッド数を設定していない場合は，URL グループ単位でも占有スレッド数を設定できないので注意してください。

また，URL グループ単位の同時実行スレッド数制御を設定しているリクエスト URL に対するリクエスト数が，URL グループ単位の占有スレッド数を超える場合で，かつ URL グループ単位の最大同時実行スレッド数に満たない場合は，Web アプリケーション単位の共有スレッド数を使用してリクエストを処理しま

す。このため，Web アプリケーション単位の共有スレッド数が少ないと，URL グループ単位の最大同時実行スレッド数は，設定値よりも少なくなることがあるので注意してください。

(3)　URL グループ単位の実行待ちキュー

URL グループ単位の実行待ちキューは，URL グループ単位の同時実行スレッド数が上限に達したときにリクエストが入るキューです。URL グループ単位の実行待ちキューは，URL グループ単位の同時実行スレッド数の制御を設定した場合に，URL グループごとに作成されます。この実行待ちキューのサイズを設定します。

リクエストは，URL グループ単位の同時実行スレッド数が上限に達していて，URL グループ単位の実行待ちキューに空きがある場合，その実行待ちキューに入ります。URL グループ単位の実行待ちキューの中のリクエストは，処理中のリクエストが完了したあとに，その実行待ちキューから順次取り出され，実行されます。URL グループ単位の同時実行スレッド数が上限に達していて，URL グループ単位の実行待ちキューサイズに空きがない場合はエラーとなり，クライアントに HTTP ステータスコード 503 が返ります。

なお，URL グループ単位の実行待ちキューに入るリクエストは，デフォルトの実行待ちキューや Web アプリケーションの実行待ちキューに入ることはありません。また，URL グループ単位の同時実行スレッド数の制御で設定したリクエスト URL に該当しないリクエストについては，Web アプリケーションの実行待ちキューが使用されます。

(4)　URL パターンの設定

リクエスト URL を振り分けるための URL パターンを設定します。URL パターンには，Servlet 仕様のサーブレットマッピングの URL パターンが指定できます。指定できる URL パターンを次に示します。

- "/"で始まる文字列
  例：/index.jsp
- "/"で始まり，"/*"で終わる文字列
  例：/test/*
- "*."で始まる文字列
  例：*.do

なお，リクエスト URL の URL パターンとのマッピング順序については，「2.18.2　URL パターンのマッピング処理」を参照してください。

URL グループ単位で同時実行スレッド数を制御する場合，Web アプリケーション単位での同時実行スレッド数制御，および Web コンテナでのスレッド数制御も同時に設定する必要があります。URL グループ単位での同時実行スレッド数制御で設定するパラメタを次の表に示します。

表 2-57　URL グループ単位でスレッド数を制御する場合の設定パラメタ

設定するパラメタ	設定単位		
	Web コンテナ単位	Web アプリケーション単位	URL グループ単位
同時実行スレッド数の制御をするかどうか	−	○	−
最大同時実行スレッド数	○	○	○
占有スレッド数	−	○	○
実行待ちキューサイズ	−	○	○

設定するパラメタ	設定単位		
	Web コンテナ単位	Web アプリケーション単位	URL グループ単位
デフォルト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	−	−	○
制御対象となる URL パターン	−	−	○

（凡例）　○：設定する　−：該当しない

次に，URL グループ単位での同時実行スレッド数制御の設定について説明します。URL グループ単位での同時実行スレッド数の制御は，サーバ管理コマンドで設定します。なお，Web コンテナ単位での設定パラメタについては，「2.16.2　実行環境での設定（J2EE サーバの設定）」を，Web アプリケーション単位での設定パラメタについては，「2.17.5　実行環境での設定」を参照してください。

サーバ管理コマンドでは，次の内容が設定できます。

- **URL グループ単位の同時実行スレッド数制御の定義名**

  同時実行スレッド数制御をする URL のグループ名を指定します。グループ名は Web アプリケーション内で一意な名称を指定する必要があります。

  使用できる文字は次のとおりです。

  - 半角英数字（A～Z，a～z，0～9）
  - 半角ハイフン（-）
  - 半角アンダーバー（_）

  また，文字列の長さは 1～64 文字となります。

- **URL グループ単位の最大同時実行スレッド数**

  URL グループで，最大で幾つのスレッドを同時に実行できるかを指定します。設定範囲は次のとおりです。

  最大同時実行スレッド数の設定範囲

  1≦最大同時実行スレッド数（URL グループ単位）≦最大同時実行スレッド数（Web アプリケーション単位）

- **URL グループ単位の占有スレッド数**

  URL グループで，最低限確保するスレッド数（**占有スレッド数**）を指定します。設定範囲は次のとおりです。

  占有スレッド数の設定範囲

  0≦占有スレッド数（URL グループ単位）≦最大同時実行スレッド数（URL グループ単位）

  さらに，URL グループ単位の占有スレッド数は，Web アプリケーション単位の占有スレッド数以下である必要があります。

  また，Web アプリケーション内の URL グループ単位に設定している占有スレッド数の総和は，次の条件を満たしている必要があります。なお，条件は，Web アプリケーション単位の最大同時実行スレッド数と Web アプリケーション単位の占有スレッド数の値によって異なります。

  Web アプリケーション単位の設定が，最大同時実行スレッド数≠占有スレッド数の場合

  占有スレッド数（Web アプリケーション単位）≧占有スレッド数の総和（URL グループ単位）

Web アプリケーション単位の設定が，最大同時実行スレッド数＝占有スレッド数の場合

占有スレッド数（Web アプリケーション単位）＞占有スレッド数の総和（URL グループ単位）

なお，占有スレッド数を設定しない場合は，0 を指定します。

- **URL グループ単位の実行待ちキューサイズ**

  実行スレッド数が最大数に達した場合，リクエストはキューに溜められます。このときの，実行待ちキューサイズを URL グループ単位で指定します。設定範囲は次のとおりです。

  URL グループ単位の実行待ちキューサイズの設定範囲

  0≦実行待ちキューサイズ（URL グループ単位）≦2,147,483,647

  なお，0 を設定すると，URL グループ単位の実行待ちキューを使用しない設定になります。このとき，同時実行スレッド数が上限に達すると，リクエストはエラーとなります。

- **制御対象となる URL パターン**

  同時実行スレッド数の制御対象となる URL を指定します。URL パターンは Web アプリケーション内で一意な名称を指定する必要があります。また，Web アプリケーションのコンテキスト以下を指定してください。

サーバ管理コマンドを使用する場合は，WAR 属性ファイルの<thread-control>タグ下の<urlgroup-thread-control>タグで，同時実行スレッド数を設定します。

- <urlgroup-thread-control-name>タグに，同時実行スレッド数制御の定義名を指定します。
- <urlgroup-thread-control-max-threads>タグに，URL グループ単位での最大同時実行スレッド数を指定します。
- <urlgroup-thread-control-exclusive-threads>タグに，占有スレッド数を指定します。
- <urlgroup-thread-control-queue-size>タグに，URL グループ単位の実行待ちキューサイズを指定します。
- <urlgroup-thread-control-mapping>タグに，制御対象となる URL パターンを<url-pattern>タグで囲んで指定します。

サーバ管理コマンドの cjgetappprop コマンドで属性ファイルを取得し，属性ファイル編集後に，cjsetappprop コマンドで編集内容を反映させてください。サーバ管理コマンドについては，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「3.　サーバ管理コマンドの基本操作」を参照してください。

2.18.4　同時実行スレッド数設定の指針（URL グループ単位）

URL グループ単位の同時実行スレッド数制御をする場合の設定の指針について説明します。

- URL グループ単位の同時実行スレッド数制御の設定をする場合, 事前に次の内容を設定しておいてください。
  - Web コンテナ単位の最大同時実行スレッド数
  - Web アプリケーション単位の最大同時実行スレッド数，占有スレッド数，実行待ちキューサイズ
- 設定する値は，次の順に決定してください。
  1. URL グループ単位の最大同時実行スレッド数を設定する。
  2. URL グループ単位の占有スレッド数を設定する。
  3. URL グループ単位の実行待ちキューサイズを設定する。
  4. URL パターンを設定する。

- URL グループ単位の最大同時実行スレッド数は，J2EE アプリケーション内にある業務ロジックに対して，適切な値を設定してください。

  例えば，特定の業務ロジックの処理が重い場合，その業務ロジックの同時実行スレッド数に上限を設けることで，特定の業務ロジックにアクセスが集中した場合でも，特定の業務ロジックが Web アプリケーション全体の処理能力を占有するのを防ぐことができます。
- URL グループ単位の占有スレッド数は，特定の業務ロジックを Web アプリケーション内のほかの業務ロジックの影響を受けないで実行させるために設定します。
- URL グループ単位の実行待ちキューサイズは，特定の業務ロジックへの流量の上限を設けるために設定します。URL グループ単位の実行待ちキューは，デフォルトの実行待ちキュー，Web アプリケーション単位の実行待ちキューと設定指針は同じです。

2.18.5　cosminexus.xml での定義

アプリケーションの開発環境で必要な cosminexus.xml での定義について説明します。

URL グループ単位での同時実行スレッド数の制御をするための定義は，cosminexus.xml の<war>タグ内に指定します。

cosminexus.xml での URL グループ単位での同時実行スレッド数の制御の定義について次の表に示します。

表 2-58　cosminexus.xml での URL グループ単位での同時実行スレッド数の制御の定義

指定するタグ	設定内容
<urlgroup-thread-control-name>タグ	同時実行スレッド数制御の定義名を指定します。
<urlgroup-thread-control-max-threads>タグ	URL グループ単位での最大同時実行スレッド数を指定します。
<urlgroup-thread-control-exclusive-threads>タグ	占有スレッド数を指定します。
<urlgroup-thread-control-queue-size>タグ	URL グループ単位の実行待ちキューサイズを指定します。
<urlgroup-thread-control-mapping>タグ	制御対象となる URL パターンを<url-pattern>タグで囲んで指定します。

指定するタグの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「2.2.6　War 属性の詳細」を参照してください。

2.18.6　実行環境での設定（Web アプリケーションの設定）

URL グループ単位での同時実行スレッド数の制御をする場合，Web アプリケーションの設定が必要です。

なお，Web アプリケーションの設定は，cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。URL グループ単位での同時実行スレッド数の制御の定義には，WAR 属性ファイルを使用します。

WAR 属性ファイルで指定するタグは，cosminexus.xml と対応しています。cosminexus.xml での定義については，「2.18.5　cosminexus.xml での定義」を参照してください。

2.18.7　同時実行スレッド数および実行待ちキューサイズの設定例（URL グループ単位）

同時実行スレッド数および URL グループ単位の実行待ちキューサイズの設定例について説明します。ここでは，Web サーバ連携機能を使用する場合の例について説明します。

(1)　説明で使用する Web アプリケーションの設定例

この例では，Web コンテナに二つの Web アプリケーションがデプロイされていて，そのうちの一つの Web アプリケーションに Web アプリケーション単位での同時実行スレッド数制御および URL グループ単位の同時実行スレッド数制御がされているものとします。設定内容を次に示します。

- Web サーバの最大同時接続数：40
- Web コンテナ単位の最大同時実行スレッド数：8
- デフォルトの実行待ちキューサイズ：5
- Web アプリケーション単位の同時実行スレッド数制御の設定

  Web アプリケーション A に次の内容が設定されているものとします。なお，Web アプリケーション B では，Web アプリケーション単位の同時実行スレッド数制御の設定はありません。

Web アプリケーション名	最大同時実行スレッド数	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
Web アプリケーション A	7	3	5

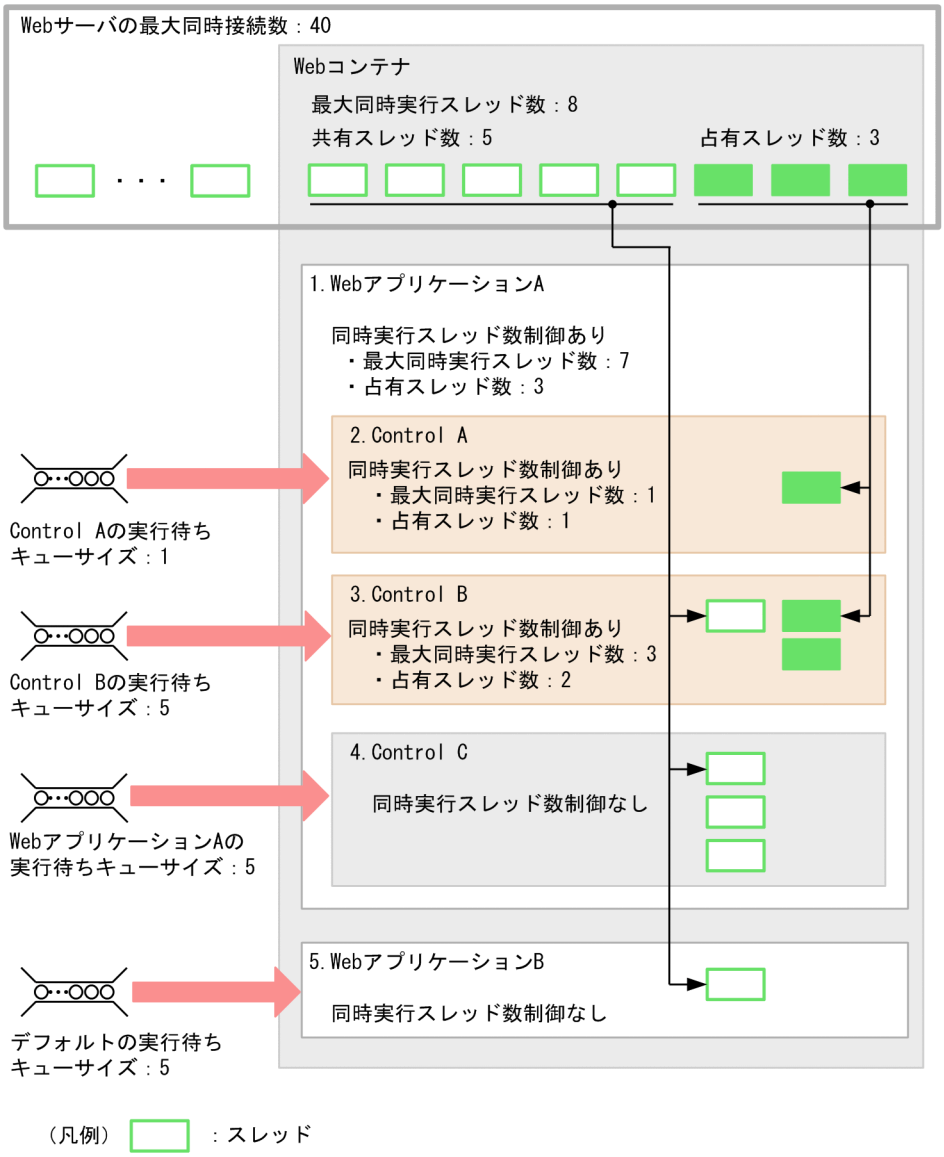
- URL グループ単位の同時実行スレッド数制御の設定

  Web アプリケーション A では，次に示す URL グループの設定がされているものとします。なお，Control C には，URL グループ単位の同時実行スレッド数制御の設定はありません。

URL グループ名	URL パターン	最大同時実行スレッド数	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
Control A	/health_check.jsp	1	1	1
Control B	/create_pdf	3	2	5

URL グループ単位でスレッド数を制御する場合の例を，次の図に示します。

図 2-23　URL グループ単位の設定例



(2)　各 Web アプリケーションで使用できるスレッド数

図 2-23 の設定の場合に使用できる最大同時実行スレッド数，占有スレッド数，および実行待ちキューサイズについて，Web アプリケーションまたは URL グループごとに説明します。なお，説明の項番は，図中の項番と対応しています。

1. Web アプリケーション A

Web アプリケーション A では同時実行スレッド数制御を設定しています。また，Web アプリケーション A 内の業務ロジック（Control A および Control B）には URL グループ単位の同時実行スレッド数制御を設定しています。

ここでは，Web アプリケーション A のスレッド数について説明します。

- 最大同時実行スレッド数および占有スレッド数

  最大同時実行スレッド数および占有スレッド数が設定されているので，それぞれ設定された値までスレッド数を使用できます。

Web アプリケーション A の同時に実行できるスレッド数は最大で 7 スレッドです。そのうち，3 スレッドは，Web アプリケーション A で最低限確保できる占有スレッド数となります。

- 共有スレッド数

  Web アプリケーション A 全体で使用できる共有スレッド数は, Web アプリケーション A の最大同時実行スレッド数－占有スレッド数の合計となります。この場合 7−3 となるので, 共有スレッド数は 4 となります。

- Web アプリケーション単位の実行待ちキューサイズ

  Web アプリケーション A では Web アプリケーション単位の実行待ちキューサイズを指定しています。Web アプリケーション A 全体で同時に 7 スレッド使用している場合，リクエストは最大で 5 個，Web アプリケーション単位の実行待ちキューにたまります。なお，このキューは，Web アプリケーション内の業務ロジックのうち，URL グループ単位の同時実行スレッド数制御を設定していない，Control C の業務ロジックへのリクエストに使用されます。

2. Control A（/health_check.jsp へのリクエスト）

- 最大同時実行スレッド数および占有スレッド数

  Control A では，最大同時実行スレッド数および占有スレッド数が設定されているので，それぞれ設定された値までスレッド数を使用できます。

  Control A では，同時に実行できるスレッド数は最大で 1 スレッドです。この 1 スレッドは Control A で最低限確保できる占有スレッドでもあります。なお，Control A の占有スレッド数は Web アプリケーション A の占有スレッドのうちの一つとなります。

- URL グループ単位の実行待ちキューサイズ

  Control A では URL グループ単位の実行待ちキューサイズを指定しています。Control A で 1 スレッド使用している場合，リクエストは最大で 1 個，URL グループ単位の実行待ちキューにたまります。

3. Control B（/create_pdf へのリクエスト）

- 最大同時実行スレッド数および占有スレッド数

  Control B では，最大同時実行スレッド数および占有スレッド数が設定されているので，それぞれ設定された値までスレッド数を使用できます。

  Control B では，同時に実行できるスレッド数は最大で 3 スレッドです。そのうち 2 スレッドは，Control B で最低限確保できる占有スレッドです。なお, Control B の占有スレッド数は Web アプリケーション A の占有スレッドのうちの二つとなります。

- URL グループ単位の実行待ちキューサイズ

  Control B では URL グループ単位の実行待ちキューサイズを指定しています。Control B で 3 スレッド使用している場合，リクエストは最大で 5 個，URL グループ単位の実行待ちキューにたまります。

4. Control C の処理

Web アプリケーション A 内の，Control C へのリクエストに対しては，同時実行スレッド数制御を設定していません。

このため，次のように動作します。

- 最大同時実行スレッド数

  Web アプリケーション A の共有スレッド数が最大同時実行スレッド数になります。Web アプリケーション A の共有スレッド数は 4 スレッドであるため，Control C の処理の最大同時実行スレッド数は 4 になります。

また，同時実行スレッド数制御を設定してないため，占有スレッド数はありません。このため，Control A または B へのアクセスが集中し，Web アプリケーション A 内で使用できるスレッドがなくなると，Control C の業務ロジックの処理は実行待ちになります。

- Web アプリケーション単位の実行待ちキューサイズ

  Control C の業務ロジックの処理で実行待ちが発生した場合，これらの処理へのリクエストは，キューにたまります。Control C の業務ロジックでは URL グループ単位の実行待ちキューサイズを指定していないため，キューは Web アプリケーション A の実行待ちキューにためられます。リクエストは，最大で 5 個，キューにたまります。

5. Web アプリケーション B

Web アプリケーション B では，Web アプリケーション単位の同時実行スレッド制御を設定していません。

このため，次のように動作します。

- 最大同時実行スレッド数

  Web コンテナの共有スレッド数が最大同時実行スレッド数になります。Web コンテナの共有スレッド数は，次の式で求められます。

  Web コンテナの最大同時実行スレッド数－Web アプリケーション A の占有スレッド数

  この例の場合，8－3 となるため，Web アプリケーション B の最大同時実行スレッド数は 5 になります。

- デフォルトの実行待ちキューサイズ

  Web アプリケーション B で実行待ちが発生した場合，Web アプリケーション B へのリクエストは，キューにたまります。Web アプリケーション B では Web アプリケーション単位の実行待ちキューサイズを指定していないため，キューはデフォルトの実行待ちキューにためられます。リクエストは，最大で 5 個，キューにたまります。

参考

Web コンテナでは，スレッドは静的コンテンツやリクエストのエラー処理にも使用されます。これらの目的で使用されるスレッド数は，次の式から求められます。

Web サーバの処理スレッド数－（Web コンテナ単位の最大同時実行スレッド数＋実行待ちキューサイズの総和※）

注※　実行待ちキューサイズの総和とは，この図の場合，Web コンテナ，Web アプリケーション A，Control A，および Control B の実行待ちキューサイズを足した値となります。

このため，この図の例の場合，40－（8 ＋（5 ＋ 5 ＋ 1 ＋ 5））となり，静的コンテンツやリクエストのエラー処理に使用されるスレッド数は，16 スレッドとなります。

2.19　同時実行スレッド数の動的変更

この節では，同時実行スレッド数の動的変更について説明します。

この節の構成を次の表に示します。

表 2–59　この節の構成（同時実行スレッド数の動的変更）

分類	タイトル	参照先
解説	同時実行スレッド数の動的変更の概要	2.19.1
	同時実行スレッド数の動的変更の流れ	2.19.2
	同時実行スレッド数を動的に変更したときの Web アプリケーションの動作	2.19.3
注意事項	同時実行スレッド数の動的変更についての注意事項	2.19.4

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

2.19.1　同時実行スレッド数の動的変更の概要

ここでは，同時実行スレッド数の動的変更の概要について説明します。

(1)　同時実行スレッド数の動的変更の用途

アプリケーションサーバを使用して構築したシステムでは，Web アプリケーション単位の最大同時実行スレッド数，占有スレッド数，および実行待ちキューサイズを，サービスを停止しないで動的に変更できます。Web アプリケーション単位の最大同時実行スレッド数，占有スレッド数，および実行待ちキューサイズは，運用管理コマンド（mngsvrutil）を使用して変更します。

Web アプリケーション単位の最大同時実行スレッド数を変更すると，次のような局面に対応できます。

- **Web アプリケーション単位の稼働状態でのパフォーマンスチューニング**
  クライアントにサービスを提供しながらパフォーマンスをチューニングできます。
- **アクセス状況に応じた一時的な Web アプリケーション単位の最大同時実行スレッド数の変更**
  アクセス状況に応じて，特定の Web アプリケーションの最大同時実行スレッド数を一時的に増加または減少させたい場合に，対処できます。
- **時間帯に応じた計画的な Web アプリケーション単位の最大同時実行スレッド数の変更**
  時間帯に応じて，Web アプリケーションの最大同時実行スレッド数を計画的に増加または減少させたい場合に，対処できます。

なお，ここで設定した項目は，サービスを停止すると無効になり，設定内容は J2EE サーバに保存されません。また，この方法で動的に変更できるのは，Web アプリケーションについての情報だけです。Web コンテナについての設定を変更する場合は，J2EE サーバを再起動しないと有効になりません。

Web コンテナ単位の最大同時実行スレッド数を変更したい場合，また，Web アプリケーションの最大同時実行スレッド数を一時的ではなく恒常的に変更したい場合は，システムの構築時と同じ手順で設定してください。

! 注意事項

Web アプリケーションの最大同時実行スレッド数を変更した場合，URL グループ単位の最大同時実行スレッド数や占有スレッド数との関係によっては，URL グループ単位の同時実行スレッド数制御の動作へ影響が出ること

があります。詳細については「2.19.1(4)　URL グループ単位の同時実行スレッド数制御への影響」を参照してください。

(2)　設定変更の例

ここでは，特定の Web アプリケーションに対して，スループットを向上し，エラーとなるリクエストを減らしたい場合の，設定変更例を紹介します。この例では，Web アプリケーションの，最大同時実行スレッド数，占有スレッド数，および実行待ちキューサイズを増やしています。なお，同時実行スレッド数の動的変更では，Web コンテナ単位の最大同時実行スレッド数および URL グループ単位の最大同時実行スレッド数は変更できません。

表 2-60　同時実行スレッド数の動的変更の例

パラメタ		変更前の設定値	変更後の設定値
Web コンテナ単位の最大同時実行スレッド数		10	－
Web アプリケーションの設定	最大同時実行スレッド数	7	8
	占有スレッド数	4	5
	実行待ちキューサイズ	8	10

（凡例）　－：設定を変更できない

(3)　設定変更後の動作

同時実行スレッド数の変更はすぐに反映されます。変更した直後に注意が必要な動作を次に示します。

最大同時実行スレッド数を変更した場合

- 最大同時実行スレッド数を増加させたとき

  Web アプリケーション単位の実行待ち状態のリクエストで，実行できる状態になったリクエストはすぐに実行されます。

- 最大同時実行スレッド数に指定しているスレッド数をすべて使用している状態で，最大同時実行スレッド数を減少させたとき

  一時的に変更後の最大同時実行スレッド数を上回るリクエストが同時に実行されます。

占有スレッド数を変更した場合

- Web コンテナ単位の最大同時実行スレッド数に設定した数のスレッドをすべて使用している状態で，占有スレッド数を増加させたとき

  占有スレッド数を増加させた Web アプリケーションに実行待ちのリクエストがあるときは，占有スレッド数分のリクエストはすぐ実行されます。ただし，このとき，一時的に Web コンテナ単位の最大同時実行スレッド数を超えたリクエストが同時に実行されます。

- 占有スレッド数を減少させたとき

  占有スレッド数を減少させると，すべての Web アプリケーション単位で共有するスレッド数が増加します。このときに，実行待ち状態で，すべての Web アプリケーション単位で共有するスレッド数の増加によって実行できる状態になったリクエストがあれば，すぐに実行されます。

Web アプリケーション単位の実行待ちキューサイズを変更した場合

- Web アプリケーション単位の実行待ちキューで，キューの上限までリクエストの待ちがある状態で，実行待ちキューサイズを減少したとき

  実行待ちキューサイズを超えたリクエストは HTTP 503 エラーが返ります。

(4) URL グループ単位の同時実行スレッド数制御への影響

URL グループ単位の同時実行スレッド数制御を設定している Web アプリケーションの同時実行スレッド数を動的に変更した場合，URL グループ単位の同時実行スレッド数の設定に影響が出ることがあります。影響が出る変更を次に示します。

Web アプリケーション単位の最大同時実行スレッド数を減らした場合

Web アプリケーション単位の最大同時実行スレッド数の減少によって，次の条件を満たしたとき，URL グループ単位の最大同時実行スレッド数は，一時的に Web アプリケーション単位の同時実行スレッド数になります。

URL グループ単位の最大同時実行スレッド数が変更される条件

URL グループ単位の最大同時実行スレッド数＞ Web アプリケーション単位の最大同時実行スレッド数

ただし，URL グループ単位の最大同時実行スレッド数の設定値が変更になるわけではありません。動的変更時に設定したスレッド数まで Web アプリケーション単位の最大同時実行スレッド数が減少し，URL グループ単位の最大同時実行スレッド数が Web アプリケーション単位の最大同時実行スレッド数を下回ると，URL グループ単位の最大同時実行スレッド数は設定した値で動作します。

なお，この変更によって，実行中のリクエスト処理を継続させるために，一時的に変更後の Web アプリケーション単位の最大同時実行スレッド数を上回るリクエストが同時に実行されることがあります。

Web アプリケーション単位の占有スレッド数を減らした場合

Web アプリケーション単位の占有スレッド数の減少によって，Web アプリケーション内のすべての URL グループ単位に設定されている占有スレッド数が使用できなくなります。使用できなくなる条件を次に示します。なお，条件は，Web アプリケーション単位の最大同時実行スレッド数と占有スレッド数の関係によって異なります。

Web アプリケーション単位の最大同時実行スレッド数＝ Web アプリケーション単位の占有スレッド数の場合

次に示す式を満たしたとき，URL グループ単位に設定されている占有スレッド数は使用できなくなります。

Web アプリケーション単位の占有スレッド数≦URL グループ単位の占有スレッド数の総和

Web アプリケーション単位の最大同時実行スレッド数≠Web アプリケーション単位の占有スレッド数の場合

次に示す式を満たしたとき，URL グループ単位に設定されている占有スレッド数は使用できなくなります。

Web アプリケーション単位の占有スレッド数＜ URL グループ単位の占有スレッド数の総和

2.19.2 同時実行スレッド数の動的変更の流れ

Web アプリケーションの最大同時実行スレッド数を動的に変更するための準備と手順を次に示します。

準備

Web アプリケーションの最大同時実行スレッド数の動的変更は，J2EE サーバおよび Web アプリケーションを含む J2EE アプリケーションが起動，開始されている状態で実行します。

J2EE サーバの起動ついては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.24　システムを起動する（CUI 利用時）」を参照してください。J2EE アプリケーションの開始を含むシステムの起動方法については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.29　業務アプリケーションを設定して開始する（CUI 利用時）」を参照してください。

手順

次の手順で実行します。

1. **Web アプリケーションの稼働状況を監視して，最大同時実行スレッド数を変更する必要があるかどうかを確認する（(1)参照）**
   運用管理コマンドを使用して実行します。
2. **必要と判断した場合，Web アプリケーションの最大同時実行スレッド数を変更する（(2)参照）**
   運用管理コマンドを使用して実行します。
3. **Web アプリケーションの稼働状況を確認して，改善されたことを確認する（(1)参照）**
   運用管理コマンドを使用して実行します。

(1)　Web アプリケーションの稼働状況の確認

稼働中の Web アプリケーションの稼働状況を確認します。Web アプリケーションの稼働状況は，運用管理コマンド（mngsvrutil）で確認できます。確認した結果，例えば，次のような場合には，最大同時実行スレッド数を変更することを検討してください。

- 稼働スレッド数に比べて実行待ちスレッド数が極端に多い状況を想定していない場合に，稼働スレッド数に比べて実行待ちスレッド数が極端に多いとき
- Web アプリケーション単位の実行待ちリクエスト数の現在値が，Web アプリケーション単位の実行待ちリクエスト数の最大値に近づいている状況を想定していない場合に，Web アプリケーション単位の実行待ちリクエスト数の現在値が，Web アプリケーション単位の実行待ちリクエスト数の最大値に近づいているとき
- Web アプリケーション単位の実行待ちキューからリクエストがあふれる状況を想定していない場合に，その実行待ちキューからリクエストがあふれているとき

  なお，監視した結果，一時的にではなく恒常的に最大同時実行スレッド数を変更する必要があると判断した場合は，動的変更をするのではなく，Web アプリケーションを停止して，最大同時実行スレッド数の設定をし直してください。Web アプリケーションを停止した状態での Web コンテナでの最大同時実行スレッド数の制御の設定については，「2.16　Web コンテナ単位での同時実行スレッド数の制御」，「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」，および「2.18　URL グループ単位での同時実行スレッド数の制御」を参照してください。

Web アプリケーションの稼働状況を確認するには，mngsvrutil コマンドに，サブコマンド「get」を指定して実行します。

実行形式および実行例を次に示します。なお，mngsvrutil コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「mngsvrutil（Management Server の運用管理コマンド）」を参照してください。

実行形式

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

実行例

```
mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host get webApps
```

コマンドの実行結果は，標準出力またはファイルに出力されます。

Web アプリケーションの稼働情報のうち，Web アプリケーションの最大同時実行スレッド数を変更する場合に参考になる情報は，次に示すヘッダ情報の項目で確認できます。なお，N 秒とは，運用監視で設定しているサンプリング時間です。

表 2-61　Web アプリケーションの最大同時実行スレッド数を変更する場合に参考になる情報

ヘッダ情報	内容
contextRoot	Web アプリケーションのコンテキストルート。
exclusiveThreadCountUpperBound	Web アプリケーションの占有スレッド数。
activeThreadCountUpperBound	Web アプリケーションの最大同時実行スレッド数。
waitingRequestCountUpperBound	Web アプリケーションの実行待ちキューサイズ。
currentThreadCountUpperBound	Web アプリケーションの同時実行可能スレッド数の上限値。
activeThreadCount	稼働スレッド数の現在値。
activeThreadCountPeak	稼働スレッド数の N 秒ピーク。
activeThreadCountAverage	稼働スレッド数の N 秒平均値。
activeThreadCountHightWaterMark	稼働スレッド数の最大値。
activeThreadCountLowWaterMark	稼働スレッド数の最小値。
waitingRequestCount	Web アプリケーション単位の実行待ちリクエスト数の現在値。
waitingRequestCountPeak	Web アプリケーション単位の実行待ちリクエスト数の N 秒ピーク。
waitingRequestCountAverage	Web アプリケーション単位の実行待ちリクエスト数の N 秒平均値。
waitingRequestCountHighWaterMark	Web アプリケーション単位の実行待ちリクエスト数の最大値。
waitingRequestCountLowWaterMark	Web アプリケーション単位の実行待ちリクエスト数の最小値。
overflowRequestCount	Web アプリケーション単位の実行待ちキューからあふれたリクエスト数。

(2)　Web アプリケーションの最大同時実行スレッド数の設定変更

稼働状況を確認した Web アプリケーションの次の項目を，必要に応じて変更します。

- 最大同時実行スレッド数
- 占有スレッド数
- 実行待ちキューサイズ

これらの項目は，運用管理コマンド（mngsvrutil）を使用して変更できます。ここで設定した値は，Web アプリケーションを停止するまで有効です。

! 注意事項

Web アプリケーションの最大同時実行スレッド数の動的変更実行中（mngsvrutil コマンドにサブコマンド「change」を指定して実行している間）は，J2EE アプリケーションのデプロイおよびアンデプロイは実行しないでください。

Web アプリケーションの最大同時実行スレッド数を動的に変更するには，mngsvrutil コマンドに，サブコマンド「change」を指定して実行します。

実行形式を次に示します。なお，mngsvrutil コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「mngsvrutil（Management Server の運用管理コマンド）」を参照してください。

(a)　実行形式

```
mngsvrutil -m <Management Serverのホスト名> [:<ポート番号>] -u <管理ユーザID> -p <管理パスワード> -t <ホスト名> -k host change webAppThreadCtrl <Webアプリケーションのコンテキストルート> <最大同時実行スレッド数>,<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>
```

次に，実行例を示します。この例では，次の表に示すように設定を変更します。なお，Web アプリケーションの名称は，「WebAP1」とします。

表 2-62　Web アプリケーション（WebAP1）の最大同時実行スレッド数の動的変更の設定例

設定対象	設定項目	変更前の設定値	変更後の設定値
Web コンテナ	最大同時実行スレッド数	10	10（変更できません）
Web アプリケーション（WebAP1）	最大同時実行スレッド数	7	8
	占有スレッド数	4	5
	実行待ちキューサイズ	8	10

(b)　実行例

```
mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host change webAppThreadCtrl "WebAP1" 8,5,10
```

設定内容は，コマンド実行後，すぐに反映されます。

2.19.3　同時実行スレッド数を動的に変更したときの Web アプリケーションの動作

ここでは，Web アプリケーションの最大同時実行スレッド数を動的に変更したときの Web アプリケーションの動作について説明します。

(1)　最大同時実行スレッドを変更したことによる動作

最大同時実行スレッド数を変更した場合，Web アプリケーションは次のように動作します。

最大同時実行スレッド数を増やした場合

Web アプリケーション単位の実行待ち状態だったリクエストのうち，実行可能になったリクエストが直ちに実行されます。

例えば，設定変更によって最大同時実行スレッド数を 7 から 8 に変更した場合に，Web アプリケーション単位の実行待ちキューにリクエストがあったときには，その実行待ちキューのリクエストの一つがすぐに実行されることになります。

最大同時実行スレッド数を減らした場合

実行可能な最大同時実行スレッド数が減少されます。

ただし，最大同時実行スレッド数のスレッドをすべて使用している状態で設定を変更しようとした場合，実行中のスレッド数は減らないため，一時的に最大同時実行スレッド数を超える数のスレッドが実行されます。

例えば，設定変更によって最大同時実行スレッド数を 8 から 7 に変更した場合に，その時点で使用されているスレッド数が 8 個あったときは，一時的に設定変更後の最大同時実行スレッド数を超える 8 個のスレッドが実行されます。

稼働中のスレッドの一つが終了したときにスレッド数が減らされ，以降は設定値どおり，最大で 7 個のスレッドが同時実行されるようになります。

(2) 占有スレッド数を変更したことによる動作

占有スレッド数を変更した場合，Web アプリケーションは次のように動作します。

占有スレッド数を増やした場合

Web アプリケーション単位の実行待ち状態だったリクエストのうち，該当する Web アプリケーションの占有スレッド数が増えたことで実行可能になったリクエストが，直ちに実行されます。

また，アクセスのピーク時など，Web コンテナ単位の最大同時実行スレッド数に設定されているスレッド数をすべて使用している状態で特定の Web アプリケーションの占有スレッド数を増やした場合，その Web アプリケーションの実行待ちキューにリクエストがあるときには，実行待ちキューのリクエストがすぐに実行されます。これによって，一時的に，Web コンテナ単位の最大同時実行スレッド数を超える数のスレッドが実行されることがあります。

例えば，Web コンテナ単位の最大同時実行スレッド数が 10 で，Web アプリケーションである WebAP1 と WebAP2 でそれぞれ 7 個と 3 個のスレッドを使用している状態で WebAP1 の占有スレッド数を 8 に変更した場合，一時的に Web コンテナ単位で 11 個のスレッドが実行されます。

占有スレッド数を減らした場合

特定の Web アプリケーションの占有スレッド数を減らすことで，Web コンテナ単位で共有されるスレッド数が増加します。Web アプリケーション単位，URL グループ単位およびデフォルトの実行待ちキューのリクエストのうち，共有スレッド数が増えることで実行可能になったスレッドがある場合は，直ちに実行されます。

(3) Web アプリケーション単位の実行待ちキューサイズを変更したことによる動作

Web アプリケーション単位の実行待ちキューサイズを変更した場合，Web アプリケーションは次のように動作します。

Web アプリケーション単位の実行待ちキューサイズを増やした場合

Web アプリケーション単位の実行待ちキューサイズが直ちに増やされます。

Web アプリケーション単位の実行待ちキューサイズを減らした場合

Web アプリケーション単位の実行待ちキューで待っているリクエストの数よりも少ない値に Web アプリケーション単位の実行待ちキューサイズを変更した場合，その実行待ちキューサイズを超えるリクエストは HTTP 503 エラーとして返却されます。

2.19.4 同時実行スレッド数の動的変更についての注意事項

- 動的に変更できるのは，Web アプリケーション単位の同時実行スレッド数の設定です。Web コンテナ単位の同時実行スレッド数および URL グループ単位の同時実行スレッド数は，動的に変更できません。
- 動的に変更された同時実行スレッド数の情報は J2EE サーバには保存されません。このため，サービスを停止すると変更した値は無効になるので注意してください。
- Web アプリケーション単位の同時実行スレッド数の動的変更によって，Web アプリケーション単位の共有スレッド数が 0 以下になると，Web アプリケーション内のすべての URL グループ単位で設定している占有スレッド数が使用できなくなります。

2.20　エラーページのカスタマイズ

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

エラーページをカスタマイズするには，Servlet 仕様で規定されている web.xml の<error-page>タグを使用する方法と，Web サーバの機能を使用する方法があります。また，インプロセス HTTP サーバによるエラーページのカスタマイズを利用することもできます。

Web サーバの機能を使用する場合のエラーページのカスタマイズについては，「4.8　エラーページのカスタマイズ（Web サーバ連携）」を参照してください。インプロセス HTTP サーバによるエラーページのカスタマイズについては，「5.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）」を参照してください。

2.21　静的コンテンツのキャッシュ

一度アクセスした静的コンテンツは，メモリ上にキャッシュできます。一度アクセスした静的コンテンツをメモリ上にキャッシュし，2 回目以降のアクセスではキャッシュからブラウザにレスポンスを返すことで，静的コンテンツのレスポンスタイムを短縮できます。

Web コンテナでは，Web アプリケーション単位でキャッシュに使用するメモリサイズの上限と，キャッシュの対象とする静的コンテンツのファイルサイズの上限を設定して，静的コンテンツのキャッシュを制御できます。

この節では，静的コンテンツのキャッシュについて説明します。

この節の構成を次の表に示します。

表 2-63　この節の構成（静的コンテンツのキャッシュ）

分類	タイトル	参照先
解説	静的コンテンツのキャッシュの制御	2.21.1
実装	DD での定義（Web アプリケーション単位での設定）	2.21.2
設定	実行環境での設定	2.21.3

注　「運用」および「注意事項」について，この機能固有の説明はありません。

2.21.1　静的コンテンツのキャッシュの制御

Web コンテナでは，Web アプリケーション単位でキャッシュに使用するメモリサイズの上限と，キャッシュの対象とする静的コンテンツのファイルサイズの上限を設定して，静的コンテンツのキャッシュを制御できます。

静的コンテンツのキャッシュの制御には次の 2 種類の方法があります。

- **Web コンテナ単位の静的コンテンツのキャッシュの制御**

  静的コンテンツのキャッシュを Web コンテナ単位で制御する方法です。Web コンテナ単位に，Web アプリケーション単位でキャッシュするメモリサイズの上限値と，キャッシュを許可する静的コンテンツのファイルサイズの上限値を設定します。設定した Web アプリケーション単位のキャッシュに使用するメモリサイズの上限値，および静的コンテンツのファイルサイズの上限値は，Web コンテナにデプロイされているすべての Web アプリケーションに適用されます。

- **Web アプリケーション単位の静的コンテンツのキャッシュの制御**

  静的コンテンツのキャッシュを Web アプリケーション単位で制御する方法です。Web アプリケーション単位に，キャッシュするメモリサイズの上限値，およびキャッシュを許可するファイルサイズの上限値を設定します。

  Web アプリケーション単位での制御と，Web コンテナ単位での制御の両方を設定した場合，Web アプリケーション単位での制御の設定が優先されます。

なお，Web アプリケーション単位のキャッシュするメモリサイズが上限値を超えた場合，または静的コンテンツのファイルサイズが上限値を超えた場合は，メモリ上にはキャッシュしないで，毎回ファイルシステムからブラウザにレスポンスを返します。

静的コンテンツのキャッシュの設定は，設定する範囲ごとに，次の個所に設定します。

- Web コンテナ単位の静的コンテンツのキャッシュの制御
  J2EE サーバのプロパティとして指定します。
- Web アプリケーション単位の静的コンテンツのキャッシュの制御
  Web アプリケーションの属性（プロパティ）として設定します。

! 注意事項

J2EE アプリケーションのリロード機能が有効な場合は，静的コンテンツのキャッシュ機能は無効になるので注意してください。

2.21.2　DD での定義（Web アプリケーション単位での設定）

Web アプリケーション単位に，静的コンテンツのキャッシュ機能を使用するかどうか，キャッシュを許可する静的コンテンツのメモリサイズ，およびファイルサイズの上限値を設定します。

アプリケーションの開発環境で必要な DD での定義について説明します。

静的コンテンツのキャッシュの Web アプリケーション単位の定義は，web.xml の<web-app><context-param>タグ内の<param-name>タグに指定します。

DD での静的コンテンツのキャッシュの定義について次の表に示します。

表 2-64　DD での静的コンテンツのキャッシュの定義

項目	<param-name>タグに指定するパラメタ	<param-value>タグの設定内容
静的コンテンツキャッシュ機能の有効／無効	com.hitachi.software.web.static_content.cache.enabled	静的コンテンツキャッシュ機能の有効／無効を設定します。
Web アプリケーション単位のメモリサイズ	com.hitachi.software.web.static_content.cache.size	静的コンテンツキャッシュ機能を有効にした場合，メモリにキャッシュできるサイズを byte 単位で指定します。 • Web アプリケーション単位で，キャッシュの合計サイズが指定した値を超えた場合は，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て，キャッシュの合計サイズが設定した値以下になるまでキャッシュの削除を繰り返します。 • メモリサイズが設定されていない Web アプリケーションでは，そのプロパティに指定した値が用いられます。しかし，メモリサイズが設定されている Web アプリケーションでは，そのプロパティに指定した値は用いられません。 • 0～2147483647 までの整数値で指定します。0 を指定した場合は，Web アプリケーション単位でメモリにキャッシュできるサイズに制限を設けません。 • このプロパティに無効な値が設定された場合，およびキャッシュを許可するファイルサイズで指定した値よりも小さい値の場合，デフォルト値が使用されます。

項目	<param-name>タグに指定するパラメタ	<param-value>タグの設定内容
Web アプリケーション単位のメモリサイズ	com.hitachi.software.web.static_content.cache.size	• このプロパティに空文字列，または空白文字が設定された場合は，デフォルト値が使用されます。
キャッシュを許可するファイルサイズ	com.hitachi.software.web.static_content.cache.filesize.threshold	静的コンテンツキャッシュ機能を有効にした場合，キャッシュできるファイルサイズを byte 単位で指定します。 • 指定した値を超えるサイズのファイルはキャッシュされません。 • ファイルサイズが設定されていない Web アプリケーションでは，そのプロパティに指定した値が用いられます。しかし，ファイルサイズが設定されている Web アプリケーションでは，そのプロパティに指定した値は用いられません。 • 0～2147483647 までの整数値で指定します。0 を指定した場合は，キャッシュできるファイルのサイズに制限を設けません。 • プロパティに無効な値が設定された場合，および Web アプリケーション単位のメモリサイズで指定した値より大きい場合は，デフォルト値が使用されます。 • このプロパティに空文字列，または空白文字が設定された場合は，デフォルト値が使用されます。

! 注意事項

次に示すパラメタは，静的コンテンツキャッシュ機能で使用するため，DD の<context-param>タグ内で任意に使用できません。

- com.hitachi.software.web.static_content.cache.enabled
- com.hitachi.software.web.static_content.cache.size
- com.hitachi.software.web.static_content.cache.filesize.threshold

DD の定義例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN' 'http://
java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
   <context-param>
      <param-name>
         com.hitachi.software.web.static_content.cache.enabled
      </param-name>-
      <param-value>true</param-value>
   </context-param>

   <context-param>
      <param-name>
         com.hitachi.software.web.static_content.cache.size
      </param-name>
      <param-value>5242880</param-value>
   </context-param>

   <context-param>
      <param-name>
         com.hitachi.software.web.static_content.cache.filesize.threshold
```

```
        </param-name>
        <param-value>102400</param-value>
    </context-param>
</web-app>
```

定義例では，次の内容が定義されています。

- 静的コンテンツキャッシュ機能を有効にします。
- Web アプリケーション単位のメモリサイズを 5MB にします。
- キャッシュを許可するファイルサイズの上限値を 100KB にします。

なお，DD で無効な値や空文字が設定された場合は，Web コンテナ単位での設定内容（簡易構築定義ファイルの設定内容）が使用されます。

2.21.3　実行環境での設定

静的コンテンツのキャッシュを Web コンテナ単位に定義する場合，J2EE サーバの設定が必要です。

静的コンテンツのキャッシュを Web アプリケーション単位に定義する場合，Web アプリケーションの設定が必要です。cosminexus.xml を含まない Web アプリケーションのプロパティを設定または変更する場合にだけ参照してください。

ポイント

Web コンテナ単位での設定，Web アプリケーション単位での設定をどちらも設定した場合には，Web アプリケーション単位での設定が優先されます。

(1)　J2EE サーバの設定（Web コンテナ単位での設定）

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web コンテナ単位での静的コンテンツのキャッシュの定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの静的コンテンツのキャッシュの定義について次の表に示します。

表 2-65　簡易構築定義ファイルでの静的コンテンツのキャッシュの定義

項目	指定するパラメタ	設定内容
静的コンテンツキャッシュ機能の有効／無効	webserver.static_content.cache.enabled	静的コンテンツのキャッシュの有効，無効，または強制無効を指定します。
Web アプリケーション単位のメモリサイズ	webserver.static_content.cache.size	静的コンテンツのキャッシュを許可する，Web アプリケーション単位のメモリサイズを設定します。
キャッシュを許可するファイルサイズ	webserver.static_content.cache.filesize.threshold	静的コンテンツのキャッシュを許可する，Web アプリケーション単位のファイルサイズの上限値を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバリファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　Web アプリケーションの設定（Web アプリケーション単位での設定）

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。静的コンテンツのキャッシュの定義には，WAR 属性ファイルを使用します。

WAR 属性ファイルで指定するタグは，DD と対応しています。DD での定義については，「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。

2.22　URI のデコード機能

URI のデコード機能は，Web サーバ連携，およびインプロセス HTTP サーバで利用できます。

この節では，URI のデコード機能について説明します。

この節の構成を次の表に示します。

表 2-66　この節の構成（URI のデコード機能）

分類	タイトル	参照先
解説	URI のデコード機能の概要	2.22.1
設定	実行環境での設定（J2EE サーバの設定）	2.22.2
注意事項	URI のデコード機能を使用する場合の注意事項	2.22.3

注　「実装」および「運用」について，この機能固有の説明はありません。

2.22.1　URI のデコード機能の概要

URI のデコード機能を使用すると，アプリケーションサーバでは，リクエスト URI のサーブレットパスおよび追加のパス情報に含まれる URL エンコードされた文字列がデコードされます。ただし，コンテキストパスはデコードされません。

デコードされた URI を使用しない Web アプリケーションを実行するときは，URI のデコード機能を使用しないように設定するか，Web アプリケーション側で対応する必要があります。

URI のデコード機能の使用時に影響がある Servlet API，デコードされた文字列を使用する機能，デコード時に使用される文字コード，および文字列のデコードと正規化の実行順序を次に記述します。

(1)　URI のデコード機能の使用時に影響がある Servlet API

URI のデコード機能を使用する場合，javax.servlet.http.HttpServletRequest インタフェースの次のメソッドでは，デコードされた URI が戻り値となります。

- getPathInfo メソッド
- getPathTranslated メソッド
- getServletPath メソッド

ただし，getRequestURI メソッドおよび getRequestURL メソッドでは，デコードされていない URL が戻り値となります。

(2)　デコードされた文字列を使用する機能

URI のデコード機能を使用する場合，次に示すマッチング処理で，デコードされた文字列が使用されます。

- サーブレットおよび JSP の URL パターンとのマッチング
- デフォルトマッピングとのマッチング
- 静的コンテンツとのマッチング
- フィルタの URL パターンとのマッチング

- web.xml の<error-page>タグ，または JSP の page ディレクティブの errPage 属性で指定するエラーページとのマッチング
- アクセスを制限する URL パターンとのマッチング
- ログイン認証の URL 判定
- リクエストのフォワードおよびインクルード
- HTTP レスポンス圧縮フィルタの URL パターンとのマッチング
- URL グループ単位の同時実行スレッド数制御の URL パターンとのマッチング

ただし，コンテキストパスはデコードされないで，元の文字列のまま扱われるため，コンテキストルートと一致しない場合は「404 Not Found」が戻り値となります。

また，アプリケーションサーバの次の機能では，デコード後の文字列でのマッチングは行われません。

- インプロセス HTTP サーバのエラーページのカスタマイズ機能
- インプロセス HTTP サーバのリダイレクトによるリクエストの振り分け機能

(3)　デコード時に使用される文字コード

URI のデコード機能を使用する場合，デコード時に使用される文字コードは UTF-8 です。

(4)　文字列のデコードと正規化の実行順序

クライアントから送信されたリクエスト URI で，マッチングに利用される URL は，デコードされたあとで正規化されます。

2.22.2　実行環境での設定（J2EE サーバの設定）

URI のデコード機能を使用する場合，J2EE サーバの設定が必要です。

J2EE サーバの設定は，簡易構築定義ファイルで実施します。URI のデコード機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.http.request.uri_decode.enabled に指定します。このパラメタでは，URI のデコード機能を使用するかどうかを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

2.22.3　URI のデコード機能を使用する場合の注意事項

ここでは，URI のデコード機能を使用する場合の注意事項について説明します。

(1)　文字列のデコードと正規化の実行順序

サーブレットパスと URL パターンのマッチングでは，デコードされたあとで正規化された URI が使用されます。

コンテキストパスとコンテキストルートのマッチングでは，デコードされないで正規化された URI が使用されます。

(2)　リクエストの属性

フォワード時またはインクルード時にリクエストに追加される属性にも，デコードされた値が格納されるものがあります。フォワード時またはインクルード時にリクエストに設定される各属性について，格納される値がデコードされるかどうかを次の表に示します。

処理	属性	格納される値のデコードの実行
フォワード	javax.servlet.forward.request_uri	×
	javax.servlet.forward.context_path	×
	javax.servlet.forward.servlet_path	○
	javax.servlet.forward.path_info	○
	javax.servlet.forward.query_string	×
インクルード	javax.servlet.include.request_uri	×
	javax.servlet.include.context_path	×
	javax.servlet.include.servlet_path	○
	javax.servlet.include.path_info	○
	javax.servlet.include.query_string	×

（凡例）

○：デコードされる

×：デコードされない

各属性および各属性に格納される値については，サーブレット仕様書を参照してください。

(3)　HTTP セッションの引き継ぎ

コンテキストパスはデコードされないで，元の文字列のまま扱われるため，HTTP セッションは引き継がれます。

2.23　Web アプリケーションのバージョン設定機能

この節では，Web アプリケーションのバージョン設定機能について説明します。

この節の構成を次の表に示します。

表 2-67　この節の構成（Web アプリケーションのバージョン設定機能）

分類	タイトル	参照先
解説	Web アプリケーションのバージョン設定機能の概要	2.23.1
	JSP ファイルおよびタグファイルのコンパイルと実行	2.23.2
設定	実行環境での設定	2.23.3
注意事項	Web アプリケーションのバージョン指定機能を使用する場合の注意事項	2.23.4

注　「実装」および「運用」について，この機能固有の説明はありません。

2.23.1　Web アプリケーションのバージョン設定機能の概要

アプリケーションサーバでは，Web アプリケーションの実行時に，web.xml に定義されている Web アプリケーションのバージョンに準拠して，Servlet や JSP が実行されます。

Web アプリケーションのバージョン設定機能は，Web アプリケーション実行時のバージョンを指定するための機能です。この機能を使用すると，web.xml で定義されている Web アプリケーションのバージョンを変更しないで，設定したバージョンに準拠して Web アプリケーションを実行できます。

Web アプリケーションのバージョン設定機能を使用する場合と使用しない場合では，次の違いがあります。

Web アプリケーションのバージョン設定機能を使用する場合

旧バージョンで作成した Web アプリケーションの実行時に，新バージョンに準拠して Servlet や JSP を実行させるために，web.xml に定義されている Web アプリケーションのバージョンを変更する必要はありません。

実行したい Web アプリケーションのバージョンを設定すれば，web.xml を変更したときと同様に，JSP コンパイルでの文法チェックおよびサーブレット API の動作が変更されます。ただし，web.xml に定義が必要な機能は使用できません。

Web アプリケーションのバージョン設定機能を使用しない場合

旧バージョンで作成した Web アプリケーションの実行時に，新バージョンに準拠して Servlet や JSP を実行させるためには，web.xml に定義されている Web アプリケーションのバージョンを変更する必要があります。

web.xml で定義されている Web アプリケーションのバージョンごとに，Web アプリケーションのバージョン設定機能で設定したバージョンによる動作の違いを次に示します。

表 2-68　Web アプリケーションのバージョン設定機能で設定したバージョンによる動作の違い

web.xml で定義されているバージョン	Web アプリケーションのバージョン設定機能で設定したバージョン		
	指定なし	2.4	2.5
2.2	2.3 として動作する※1	2.4 として動作する	2.5 として動作する
2.3	2.3 として動作する		
2.4	2.4 として動作する		
2.5，3.0	2.5 として動作する	2.5※2 として動作する	

注※1　web.xml で定義されているバージョンが 2.2 の場合，Web アプリケーションのバージョン設定機能で何も設定しないときは，2.3 の Web アプリケーションとして動作します。

注※2　web.xml で定義されているバージョンが 2.5 の場合，Web アプリケーションのバージョン設定機能で 2.4 を設定しても，2.5 の Web アプリケーションとして動作します。

! 注意事項

Web アプリケーションのバージョン設定機能は，旧バージョンのアプリケーションの移行などを支援するための機能です。新規にアプリケーションを開発する場合に使用することは推奨しません。

新規にアプリケーションを開発する場合は，正しいバージョンの web.xml を指定してください。

2.23.2　JSP ファイルおよびタグファイルのコンパイルと実行

Web アプリケーションのバージョン設定機能を使用する際で，JSP ファイルおよびタグファイルのコンパイル時と実行時で JSP 仕様のバージョンが異なる場合の実行時の動作について説明します。ここでは，JSP 事前コンパイル機能を使用している場合と，使用していない場合に分けて説明します。

(1)　JSP 事前コンパイル機能を使用している場合

JSP 事前コンパイル機能では，JSP ファイルのコンパイル時に，JSP ファイルから生成されたクラスファイルだけでなく，バージョン情報ファイルも作成されます。バージョン情報ファイルは，JSP 事前コンパイル機能実行時に出力されるファイルで，JSP ファイルのバージョンが記載されています。

このため，JSP のバージョンが異なる状態には，次の二つの場合があります。

- バージョン情報ファイルと Web アプリケーション実行時の Web アプリケーションバージョンに対応する JSP 仕様のバージョンが異なる場合
- JSP 事前コンパイル機能によって JSP ファイルから生成されたクラスファイルのバージョンと，それに対応する JSP 仕様のバージョンが異なる場合

それぞれの場合について，Web アプリケーションの動作を，次の表に記述します。

表 2-69　バージョン情報ファイルと JSP 仕様のバージョンが異なる場合

ファイル変更のタイミング		Web アプリケーションの動作
Web アプリケーション開始前		ケース 1
Web アプリケーション開始後	リロード有効時	ケース 2

ファイル変更のタイミング		Web アプリケーションの動作
Web アプリケーション開始後	リロード無効時	ケース 2

ケース 1

Web アプリケーション開始前に JSP 事前コンパイルコマンドが実行されている場合，Web アプリケーション開始時に KDJE39522-E が出力されて，Web アプリケーションの開始に失敗します。

ケース 2

J2EE サーバの再起動または Web アプリケーションの再開始後，ケース 1 と同じ動作になります。

表 2-70　クラスファイルと JSP 仕様のバージョンが異なる場合

ファイル変更のタイミング		Web アプリケーションの動作
Web アプリケーション開始前		ケース 3
Web アプリケーション開始後	リロード有効時	ケース 4
	リロード無効時	ケース 5

ケース 3

- web.xml で<load-on-startup>が指定されている JSP ファイルの場合

  WAR 属性ファイルまたは cosminexus.xml の<start-notify-error>タグに true が指定されているとき，またはタグの指定が省略されているときは，Web アプリケーション開始時に KDJE39298-E が出力されて，Web アプリケーションの開始に失敗します。

  cosminexus.xml の<start-notify-error>タグに false が指定されているときは，Web アプリケーション開始時に KDJE39298-E が出力されますが，Web アプリケーションの開始は成功します。ただし，リクエスト時に KDJE39298-E が出力されて，エラーコード 500（Internal Server Error）が返されます。

- web.xml で<load-on-startup>が指定されていない JSP ファイルの場合

  初回リクエスト時に KDJE39298-E が出力されて，エラーコード 500（Internal Server Error）が返されます。

ケース 4

- 実行済みの JSP ファイルの場合

  J2EE サーバの再起動または Web アプリケーションの再開始後，ケース 3 と同じ動作になります。

- 未実行の JSP ファイルの場合

  ケース 3 の「web.xml で<load-on-startup>が指定されていない JSP ファイルの場合」と同じ動作になります。

ケース 5

- 実行済みの JSP ファイルの場合

  リロード実行時に KDJE39317-E が出力されて，JSP のリロードが失敗します（該当するリクエストからエラーコード 500（Internal Server Error）が返されます）。

- 未実行の JSPP ファイルの場合

  ケース 4 の「未実行の JSP ファイルの場合」と同じ動作になります。

注意事項

- クラスファイルだけが異なるケース（ケース 3，ケース 4，ケース 5）は，異なるバージョンを指定して JSP 事前コンパイルコマンドでコンパイルしたクラスファイルで上書きした場合を想定しています。そのため，Web アプリケーション開始時の JSP 事前コンパイル機能は該当しません。
- Web アプリケーション開始時の JSP 事前コンパイル機能は，Web アプリケーション開始後にバージョン情報ファイルを更新できない（ケース 2）ため，該当しません。
- ケース 1 で Web アプリケーション開始時の JSP 事前コンパイル機能を使用して Web アプリケーションを開始した場合は，バージョン情報ファイル自体を再作成するため，エラーにはなりません。また，Web アプリケーション開始前に，JSP 事前コンパイルコマンドを実行する場合は，コマンド内でバージョンのチェックを実施しているため，コマンド実行時にバージョンの不一致を検知してエラーとなることがあります。
- JSP 事前コンパイルで個別の JSP ファイルを指定して JSP コンパイルを実行する場合は，既存のバージョン情報ファイルで指定された JSP 仕様のバージョンと，コンパイル時の JSP 仕様のバージョンが比較されます。その際，バージョンが異なるときは KDJE39522-E が出力されてエラーとなります。個別の JSP ファイルを指定しない場合は，バージョン情報ファイルが再作成されるため，エラーにはなりません。

(2)　JSP 事前コンパイル機能を使用していない場合

コンパイル時の Web アプリケーションバージョンと実行時の Web アプリケーションバージョンが異なるクラスファイルの場合，KDJE39334-I が出力されて，JSP ファイルとタグファイルが再コンパイルされます。

コンパイル時の Web アプリケーションバージョンと実行時の Web アプリケーションが異なる場合の再コンパイルの動作を次に示します。

コンパイル時のバージョン			実行時のバージョン（JSP 仕様のバージョン）			
web.xml のバージョン	ファイル種別	TLD のバージョン	2.2	2.3	2.4	2.5
2.2	JSP ファイル	－	－	－	2.0	2.1
2.3		－	－	×	2.0	2.1
2.4	JSP ファイル	－	1.2※1	1.2	×	2.1
	タグファイル	2.0	－	－	×	×※2
2.5，3.0	JSP ファイル	－	1.2※1	1.2	2.0	×
	タグファイル	2.0	－	－	×	×※2
		2.1	－	－	2.0	×

（凡例）

－：該当しない

×：再コンパイルされない

1.2：JSP1.2 仕様で再コンパイルされる

2.0：JSP2.0 仕様で再コンパイルされる

2.1：JSP2.1 仕様で再コンパイルされる

注※1

web.xml のバージョンが 2.2 の場合は Web アプリケーションバージョン 2.3 として動作するため，JSP1.2 仕様で再コンパイルされます。

注※2

JSP2.1 仕様では，タグファイルは TLD ファイルに定義される JSP バージョンによって準拠する JSP 仕様が決定されます。Web アプリケーションバージョンが 2.5 でも，タグファイルは JSP2.0 仕様で動作できるため，再コンパイルされません。

2.23.3　実行環境での設定

Web アプリケーションのバージョン設定機能を使用する場合，J2EE サーバおよび JSP 事前コンパイルのコマンド設定が必要です。

(1)　J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web アプリケーションのバージョン設定機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.application.lower_version に指定します。このパラメタでは，Web アプリケーションのバージョン設定機能を使用するかどうかを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　JSP 事前コンパイルのコマンドでの設定

JSP 事前コンパイルの設定は，cjjspc コマンドの-lowerversion オプションで実施します。cjjspc コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjjspc（JSP の事前コンパイル）」を参照してください。

2.23.4　Web アプリケーションのバージョン指定機能を使用する場合の注意事項

ここでは，Web アプリケーションのバージョン設定機能を使用する場合の注意事項について説明します。

(1)　アノテーションの使用

Web アプリケーションのバージョン設定機能でアノテーションを使用できる Web アプリケーションのバージョンを設定しても，アノテーション情報は読み込まれません。

(2)　J2EE アプリケーションのエクスポート

Web アプリケーションのバージョン設定機能を有効にして，Web アプリケーションをエクスポートした場合，Web アプリケーションのバージョン設定機能で設定したバージョンに変更されてエクスポートはされません。インポートしたときと同じバージョンの Web アプリケーションがエクスポートされます。

2.24　Web コンテナに関する注意事項

- Web コンテナで取り扱える POST データの最大サイズは 2GB です。また，Web コンテナの設定や，Web コンテナのフロントにある Web サーバやリダイレクタの設定によって，取り扱える POST データの最大サイズは制限されます。
- Web コンテナが返す Eta（Entity Tag）は，ファイルサイズおよび最終更新日時から構成され，ファイルに割り当てられた一意な ID（inode）は含まれません。

3 JSF および JSTL の利用

この章では，JSF および JSTL の利用について説明します。

3.1　この章の構成

この章では，JSF および JSTL の利用について説明します。

JSF（Java Server Faces）は，JavaEE の標準仕様で提供している Web アプリケーションフレームワークです。

JSTL（Java Server Pages Standard Tag Library）は，Web アプリケーションで頻繁に使用されるタグをまとめたタグライブラリです。

この章の構成を次の表に示します。

表 3–1　この章の構成（JSF および JSTL の利用）

分類	タイトル	参照先
解説	JSF および JSTL の概要	3.2
	JSF および JSTL の機能	3.3
実装	クラスパスの設定（開発環境の設定）	3.4
	DD での定義	3.5
	JSF アプリケーションの開発の流れ	3.6
	デバッグ用ログ（開発調査ログ）の利用	3.7
設定	実行環境の設定	3.8
運用	障害対応用の情報の出力および確認	3.9
注意事項	JSF および JSTL 使用時の注意事項	3.10

3.2　JSF および JSTL の概要

この節では，JSF と JSTL の概要について説明します。なお，JSF からは，Bean Validation の機能を利用できます。

3.2.1　JSF の概要

(1)　JSF とは

JSF は，JaveEE で標準仕様として策定された Web アプリケーションのユーザインタフェースを開発するためのフレームワークです。Web アプリケーションの開発効率の向上を目的としています。JSF で開発した Web アプリケーションを **JSF アプリケーション**といいます。

JSF では，MVC モデルに準じ，Model 層（ビジネスロジック）と View 層（入出力画面）の開発を明確に区別しています。これらの層を区別することによって，JSF アプリケーション開発に携わる担当者の作業を分担しやすくなります。各担当者は自分の作業にだけ専念できるため，開発効率の向上が図れます。

(2)　Bean Validation との連携

アプリケーションサーバでは，JSF アプリケーションでの入力値の検証処理を簡略化するために，Bean Validation の機能を利用できます。

Bean Validation の利用に関しては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「10.　アプリケーションサーバでの Bean Validation の利用」を参照してください。

3.2.2　JSTL の概要

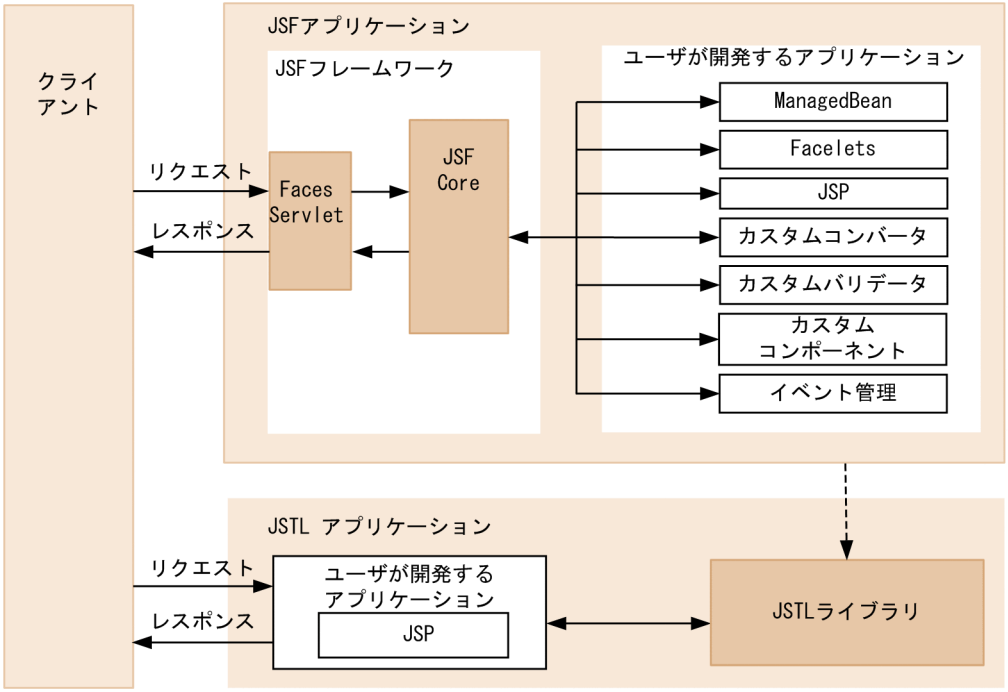
JSTL は，Web アプリケーションで実行するデータベースアクセスやループ処理などの共通処理を表現するタグをまとめたライブラリです。JSTL のタグを使用することによって，これまでコーディングする必要があった処理を記述する時間を短縮することができ，開発効率の向上が図れます。

3.3　JSF および JSTL の機能

この節では，JSF および JSTL の機能について説明します。

JSF および JSTL を使った Web アプリケーションでの処理の流れを次の図に示します。

図 3-1　JSF および JSTL を使った Web アプリケーションでの処理の流れ



- クライアントから JSF アプリケーションを使用した Web ページにアクセスすると，リクエストが JSF フレームワーク内のサーブレット FacesServlet に渡ります。FacesServlet を起点として JSF アプリケーションの処理が始まり，JSF フレームワークの JSFCore からユーザが定義したクラスやページの処理が実行されます。リクエストを処理した結果はレスポンスとして FacesServlet を介してクライアントへ返されます。
- クライアントから JSTL を使用した Web ページにアクセスすると，JSTL ライブラリが Web ページで使用しているタグに対応した処理を実行します。処理した結果はレスポンスとしてクライアントへ返されます。

3.3.1　JSF の機能

ここでは，JSF の機能について説明します。

(1)　JSF の基本機能

JSF を利用した Web アプリケーションの開発では，アクセスページビューの定義，ページの再利用，クライアントからの入力値の型の変換やバリデーション，アプリケーション内でのイベントの制御など，いろいろな機能を利用できます。

JSF の機能の多くは，ManagedBean と EL 式を利用して実現します。

ManagedBean とは，JSP および Facelets ページで使用するデータとメソッドを定義する JavaBean のことです。ManagedBean の詳細については，JSF の仕様を参照してください。

EL 式とは，規定の形式で記述することで，ManagedBean に定義したプロパティやメソッドを JSF タグの属性と関連づけることができる機能です。EL 式は JSP の仕様の一部です。詳細は，JSP の仕様を参照してください。

(2)　アプリケーションサーバでの JSF の動作

アプリケーションサーバでの JSF の動作を次に示します。

- ValueChangeListener，ActionListener，AjaxBehaviorListener，ComponentSystemEventListener として動作するメソッドを EL 式で指定した場合に，同じメソッド名で引数あり，引数なしが存在すると，引数ありのメソッドだけが呼ばれます。
- <f:facet>タグの中に複数のコンポーネントがある場合，JSP ページでは，最初に記述されているコンポーネントが処理されますが，Facelets ページでは，記述されているすべてのコンポーネントが処理されます。
- <f:subview>タグが<f:view>タグの外に記述された場合，JSP ページでは，<f:view>の値が<f:subview>の値に上書きされます。Facelets ページでは，<f:view>の値は変わらず，<f:view>および<f:subview>が一緒に表示されます。
- <f:ajax>タグを利用する場合，<h:head>タグを記述しなければいけません。記述しないと，<f:ajax>が動作しません。
- イベントの処理に valueChangeListener 属性または<f:valueChangeListener>タグを利用する場合に，<h:inputText>タグの value 属性に ValueExpression を，scope 属性に RequestScoped を設定した場合，入力値を変更しなくても，valueChangeListener のメソッドが実行されます。
- ユーザアプリケーションが"csfcfc"という Cookie を登録し，Facelets に Flash オブジェクトを利用して繰り返しリクエストをした場合，Flash オブジェクトのデータの読み込みが失敗して，例外処理（NullPointerException）が発生します。
- <ui:repeat>タグの繰り返し回数は，size 属性に設定された値より 1 回多くなります。
- ブラウザの仕様によって，一部のタグの属性が無効になるおそれがあります。ブラウザの仕様をご確認ください。
- <composite:attribute>タグ，<composite:facet>タグ，<composite:insertFacet>タグ，<composite:renderFacet>タグの requierd 属性を有効にする場合は，コンテキストパラメタ javax.faces.PROJECT_STAGE の値を Development に設定してください。
- コンテキストパラメタ javax.faces.FACELETS_REFRESH_PERIOD の指定値は，Facelets ページへの最新のリクエストを受けてから Facelets ファイルの更新を確認するまでの時間になります。そのため，短時間のうちに多くのリクエストが送られる環境では，最新のリクエストに合わせて更新の確認までの時間が延長されてしまい，Facelets ファイルの更新が反映されない状態が続くおそれがあります。
- アプリケーションサーバでは，Bean Validation の機能は常に有効になります。Bean Validation の機能を無効にする方法は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「10.5.1　JSF から Bean Validation の利用手順」を参照してください。
- <f:attribute>タグで値を指定する対象として，文字列（java.lang.String 型）を指定値とする属性を指定してください。
- JSF アプリケーションのクライアント画面からの POST データに含まれるリクエストのパラメタ数は，使用する JSF タグの種類や使用方法によって異なります。
  webserver.connector.limit.max_parameter_count キーでリクエストパラメタ数の上限値を設定する場合は，本番環境と同等の環境で実際の JSF アプリケーションを使用して検証することをお勧めします。

- アプリケーションの JAR ファイル内の META-INF/resources のリソースを更新した場合，リロード時に KDJE39556-W のメッセージが出力されます。この状態で JSF からリソースにアクセスした場合，更新したリソースが取得できます。

3.3.2　JSTL の機能

JSTL では，ページにタグを定義することで，アプリケーションで共通に使用する処理を実行できます。

JSTL には，値の設定，条件分岐，データベースへのアクセス，国際化，XML 解析などに関するタグが含まれています。

3.3.3　アプリケーションサーバ独自の機能

JSF または JSTL で利用できるアプリケーションサーバ独自の機能を次の表に示します。

表 3–2　アプリケーションサーバ独自の機能

項番	機能	説明
1	ログの出力	実行時の情報をログファイルに記録する機能です。
2	性能解析トレースの出力	特定の機能(メソッド)の開始と終了をファイルにトレースする機能です。 この情報を使って，システム性能およびボトルネックを分析できます。性能解析トレースの出力の詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.　性能解析トレースを使用した性能解析」を参照してください。

3.3.4　アプリケーションサーバのほかの機能との関連

ここでは，JSF および JSTL とアプリケーションサーバのほかの機能との関連について説明します。

JSF または JSTL とあわせて利用する場合に，留意する必要がある機能を次の表に示します。

表 3–3　JSF または JSTL と合わせて利用する場合に留意する必要がある機能

項番	機能	機能についての参照先
1	明示管理ヒープ機能	マニュアル「機能解説 拡張編」の「8.　明示管理ヒープ機能を使用した FullGC の抑止」
2	セッションフェイルオーバ機能	マニュアル「機能解説 拡張編」の「5.2　セッションフェイルオーバ機能の概要」
3	リデプロイ機能とリロード機能	マニュアル「機能解説 基本・開発編(コンテナ共通機能)」の「13.　J2EE アプリケーションの形式とデプロイ」
4	JSP 事前コンパイル機能	「2.5　JSP 事前コンパイル機能とコンパイル結果の保持」
5	J2EE リソースへの別名付与（ユーザ指定名前空間機能）	マニュアル「機能解説 基本・開発編(コンテナ共通機能)」の「2.　ネーミング管理」

注　マニュアル名の「アプリケーションサーバ」は省略しています。

以降，JSF および JSTL と各機能との関連について説明します。

(1)　明示管理ヒープ機能

JSF では，ユーザが作成した Facelets ファイルや JSP ファイルを基に生成された次の情報およびオブジェクトを，HTTP セッションに登録します。

- HTML ページの画面の情報（ビューの状態）
- SessionScope を定義した ManagedBean クラスのオブジェクト

これらの情報およびオブジェクトは，ほかの Web アプリケーションと同様に，明示管理ヒープ機能を使用して管理できます。

ただし，情報およびオブジェクトが HTTP セッションに登録されるかどうかには条件があります。また，HTTP セッションに登録されるすべての情報が明示管理ヒープを使用して管理されるわけではありません。HTTP セッションに情報またはオブジェクトが登録される条件と，それらの情報またはオブジェクトが明示管理ヒープ機能を使用して管理されるかどうかを次の表に示します。

表 3-4　HTTP セッションに情報またはオブジェクトが登録される条件

情報またはオブジェクト	HTTP セッションに登録される条件	明示管理ヒープ機能を使用して管理されるかどうか
UIComponent のビュー情報（テキストフィールド，ラジオボタン，サブミットボタンなどのユーザとの入出力インターフェースを構成するビューの情報）	JSF 標準コンテキストパラメタの javax.faces.STATE_SAVING_METHOD の値が「server」（デフォルト値）の場合	使用しない
ManagedBean オブジェクト	SessionScope アノテーションが指定された場合，または faces-config.xml の<managed-bean-scope>に「session」を指定した場合	使用する
ページで使用する文字コード	HTTP セッションが生成されていた場合	使用する
SessionMap に登録したオブジェクト	ユーザアプリケーションで使用した場合	使用する

また，明示管理ヒープ機能を使用する場合，JSF アプリケーションが明示管理ヒープ領域で使用するメモリサイズの概算は，次の式を使用して算出してください。

JSF アプリケーションが明示管理ヒープ領域で使用するメモリサイズ

1セッション当たりで明示管理ヒープ領域を使用するサイズ
＝(A※＋1) × 0.4キロバイト
＋ManagedBeanオブジェクトサイズ（HTTPセッションに登録される場合）
＋SeesionMapに登録した場合のオブジェクトサイズ

注※　A は JSF の論理ページの最大値を設定するプロパティ（com.sun.faces.numberOfLogicalViews）に指定した値です。デフォルトは 15 です。

JSF 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注意事項

JSF では，HTTP セッションの破棄（javax.servlet.http.HttpServletRequest インタフェースの invalidate メソッド呼び出し）を実行しません。このため，ユーザアプリケーション内で HTTP セッションの破棄（javax.servlet.http.HttpServletRequest インタフェースの invalidate メソッド呼び出し）を実行するか，または適切なセッションタイムアウトを設定してください。

(2)　セッションフェイルオーバ機能

JSF が HTTP セッションに登録するオブジェクトを利用して，ほかの Web アプリケーションと同様にセッションフェイルオーバ機能を使用できます。JSF 固有の設定も不要です。

JSF アプリケーションが HTTP セッションに登録するオブジェクトサイズ

セッションフェイルオーバ機能を使用する場合，JSF アプリケーションが HTTP セッションに登録するオブジェクトサイズの概算は，次の表に示す値を使用して見積もってください。

表 3-5　JSF アプリケーションが HTTP セッションに登録するオブジェクトサイズ

ページ	メモリを使用する処理	使用メモリ
Facelets ページ	必須なオブジェクトと各タグ部分	1.3 キロバイト
	一つの Form タグのページ生成部分	0.2 キロバイト※1
	EL 式を使用した場合※2	0.8 キロバイト※3
JSP ページ	必須なオブジェクトと各タグ部分	2.2 キロバイト
	一つの Form タグのページ生成部分	2.0 キロバイト※1
	EL 式を使用した場合※2	0.8 キロバイト※3

注※1　ManagedBean の作り方やタグの ID 設定などによって，多少の増減があります。

注※2　リソース数ごとに見積もってください。

注※3　リソースの作り方によって，多少の増減があります。

なお，この表で示した値は概算値です。実際に HTTP セッションで利用するメモリサイズについては，アプリケーションを実行して得られた情報を基に見積もってください。

JSF アプリケーションでセッションフェイルオーバ機能を使用する場合の注意事項

セッションフェイルオーバ機能では，HTTP セッションの属性引き継ぎ時のシリアライズ処理でシリアライズできない情報がある場合，シリアライズに失敗して，その情報は保持できません。これは，JSF アプリケーションでも同様です。次の場合，シリアライズ処理に失敗して，セッションフェイルオーバ機能は使用できません。

- SessionScope アノテーションが指定されているか，faces-config.xml の<managed-bean-scope>に「session」を指定していて，ManagedBean にシリアライズできない情報が含まれていた場合
- SessionMap にシリアライズできない情報を登録した場合

(3)　リデプロイ機能とリロード機能

リデプロイ機能について，JSF アプリケーションとして留意する点はありません。ほかの Web アプリケーションと同様に機能を使用できます。

リロード機能についても，コマンドによるリロードを実行する場合は，JSF アプリケーションとして留意する点はありません。ほかの Web アプリケーションと同様に機能を使用できます。

ただし，更新検知によるリロードについては，更新検知の対象となるファイルに次の制限があります。

JSP ファイルの更新検知

ほかの Web アプリケーションの更新検知と差異はありません。

Facelets ファイルの更新検知

更新検知の対象外です。

次に，Facelets に含まれるファイルが更新された場合に，どの範囲でリロード機能が実行されるかを示します。

表 3-6　Facelets に含まれるファイルが更新された場合のリロード機能の適用範囲

更新検知の対象ファイル	リロード機能の適用範囲		
	app	web	jsp
Facelets ファイル	×	×	×
タグファイル	○	○	○
ManagedBean コンパイル結果	○	○	○
静的コンテンツ	×	×	×

（凡例）○：適用される　×：適用されない

クラスローダによってロードされるファイルであるサーブレットや JSP ファイルは，監視対象のファイルなので，更新されたときに J2EE サーバが更新を検知してリロードが実行されます。しかし，Facelets ファイルはクラスローダによってロードされるファイルではありません。このため，Facelets ファイルを更新しても，J2EE サーバで更新が検知されないため，リロードも実行されません。

Facelets ファイルの更新を自動的に検知して反映したい場合は，標準コンテキストパラメタに javax.faces.FACELETS_REFRESH_PERIOD を指定してください。このパラメタに定義した時間間隔で Facelets ファイルの更新状態が監視され，更新が検知されると，その内容が反映されます。ただし，この機能は展開ディレクトリ形式のアプリケーションの場合だけ有効です。

なお，Facelets ファイルのページ出力を実行したあとで Facelets ファイルを更新した場合は，次にページにアクセスした時に JSF によって Facelets ファイルの更新が検知され，KDJE59227-I メッセージが出力されます。

(4)　JSP 事前コンパイル機能

JSF アプリケーション内の JSP ファイルは，JSP 事前コンパイル機能を使用してコンパイルできます。

ただし，cjjspc コマンドを使用して JSF アプリケーション内の JSP ファイルをコンパイルする場合，-classpath オプションに使用するタグライブラリとクラスライブラリを明示的に指定する必要があります。

-classpath オプションの指定例を次に示します。

JSF アプリケーション内の JSP ファイルをコンパイルする場合の cjjspc コマンドの-classpath オプションの指定例（Windows の場合）

```
-classpath <Application Serverのインストールディレクトリ>/CC/lib/cjsf.jar ; <Application Serverのインストールディレクトリ>/CC/lib/cjstl.jar
```

(5)　J2EE リソースへの別名付与（ユーザ指定名前空間機能）

JSTL では，J2EE リソースの別名は使用できません。

JSTL の<sql:setDataSource>タグの datasource 属性には，java:comp/env からの相対パスを指定してください。

3.4　クラスパスの設定（開発環境の設定）

この節では，JSF および JSTL の利用のために必要なクラスパスの設定について説明します。

3.4.1　ファイルの格納先

アプリケーションサーバのインストール時にインストールされる JSF および JSTL ライブラリの格納先を次の表に説明します。

表 3-7　JSF および JSTL ライブラリの格納先

種類	ファイル概要	ファイル名	クラスパス
JSF	インタフェースおよび実装をまとめたライブラリ	cjsf.jar	<Application Server のインストールディレクトリ>¥CC¥lib¥cjsf.jar
JSTL	インタフェースおよび実装をまとめたライブラリ	cjstl.jar	<Application Server のインストールディレクトリ>¥CC¥lib¥cjstl.jar

注　UNIX の場合は，「<Application Server のインストールディレクトリ>」を「/opt/Cosminexus」に，「¥」を「/」に読み替えてください。

3.4.2　クラスパスの設定

JSF および JSTL を使用したアプリケーションをアプリケーションサーバで動かすためには，J2EE サーバの usrconf.cfg（Java アプリケーション用オプション定義ファイル）の add.class.path キーにクラスパスを設定する必要があります。

例を示します。この例は，Windows の場合の例です。

```
add.class.path=<Application Serverのインストールディレクトリ>¥CC¥lib¥cjsf.jar
add.class.path=<Application Serverのインストールディレクトリ>¥CC¥lib¥cjstl.jar
```

! 注意事項

異なるバージョンのライブラリを同時にクラスパスに指定すると，アプリケーションが予期しない動作をするおそれがあるため，異なるバージョンのライブラリを同時にクラスパスに指定しないてください。アプリケーションのバージョンに合わせたライブラリをクラスパスに指定してください。

なお，Developer の環境で JSF アプリケーションを開発する場合の設定については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「付録 L.1　JSF を利用する場合」を参照してください。

3.5　DD での定義

JSF アプリケーションを利用するためには，DD（web.xml）にコンテキストパラメタの設定が必要です。

この節では，DD での JSF のコンテキストパラメタの設定について説明します。

3.5.1　標準のコンテキストパラメタ

JSF の標準のコンテキストパラメタを次の表に示します。

表 3-8　JSF の標準のコンテキストパラメタ

パラメタ名	データ型	指定可能値	不正な値を指定した場合の扱われ方	省略値	説明
javax.faces.CONFIG_FILES	String	コンマ（,）またはセミコロン（;）で区切られたカレントコンテキストルート下に関連する JSF 設定ファイルのリスト	不正なコンフィグファイルが無視されます。	/WEB-INF/faces-config.xml	アプリケーションで使用する JSF 設定ファイルのパスを指定します。
javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE	Boolean	true または false	false	false	convertDateTime タグで設定されるタイムゾーンに GMT を使用するかどうかを指定します。
javax.faces.DECORATORS	String	セミコロン（;）で区切られた，javax.faces.view.facelets.TagDecoratorlist 型でコンストラクタ引数のないクラス名のリスト	指定したクラスが無視されます。	""（空文字）	ユーザ定義の Decorator クラスを指定します。
javax.faces.DEFAULT_SUFFIX	String	スペースで区切られたページの拡張子のリスト	.xhtml .view .xml .jsp	.xhtml .view .xml .jsp	JSF ページとして扱うファイルの接尾辞を指定します。
javax.faces.DISABLE_FACELET_JSF_VIEWHANDLER	Boolean	true または false	false	false	Facelets ビューハンドラをアプリケーションで使用するかどうかを指定します。
javax.faces.FACELETS_BUFFER_SIZE	int	1～2147483647	1024	1024	レスポンスページをクライアントへ返す際に使用するストリームのバッファサイズを指定します。
javax.faces.FACELETS_LIBRARIES	String	セミコロン（;）で区切られたアプリケーションルートにある Facelets タグライブラリパスのリスト	指定したタグのライブラリファイル	""（空文字）	ユーザ定義の Facelets で使用するタグライブラリファイルのパスを指定します。

パラメタ名	データ型	指定可能値	不正な値を指定した場合の扱われ方	省略値	説明
javax.faces.FACELETS_LIBRARIES	String	セミコロン（;）で区切られたアプリケーションルートにある Facelets タグライブラリパスのリスト	が無視されます。	""（空文字）	ユーザ定義の Facelets で使用するタグライブラリファイルのパスを指定します。
javax.faces.FACELETS_REFRESH_PERIOD	int	-2147483648～2147483647	2	2	Facelets ページがリクエストされた際に，JSF が Facelets ファイルの変更を確認する間隔をミリ秒で設定します。※1
javax.faces.FACELETS_RESOURCE_RESOLVER	String	javax.faces.view.facelets を継承する有効な java クラス名（ResourceResolver クラスでは，引数のないコンストラクタまたは一つ ResourceResolver 型の引数を持つコンストラクタを定義します）	指定されたクラスが無視されます。	""（空文字）	ユーザ定義の ResourceResolver クラスを指定します。
javax.faces.FACELETS_SKIP_COMMENTS	Boolean	true または false	false	false	Facelets ファイルに記述したコメントをレスポンスページに出力するかどうかを指定します。
javax.faces.FACELETS_VIEW_MAPPINGS	String	セミコロン（;）で区切られた，"*"で開始または終了する文字列のリストが有効な値と見なされます。	指定された文字列が無視されます	""（空文字）	Facelets ファイルを認識するファイル名のパターンを指定します。
javax.faces.FULL_STATE_SAVING_VIEW_IDS	String	コンマ（,）で区切られたビュー ID を示す文字列のリスト	指定された文字列が無視されます。	""（空文字）	状態をすべて保存したいビューの ID を指定します。このパラメタで指定したビューでは，状態を部分的に保存するメソッドが使用できなくなります。※2
javax.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL	Boolean	true または false	false	false	サブミットされた値が空文字であった場合に，JSF 内部で null に変換するかどうかを指定します。
javax.faces.LIFECYCLE_ID	String	Java ID 名	DEFAULT	JSF アプリケーションが起動するときに，警告メッセージは出力され	ユーザ定義のライフサイクル ID を指定します。

パラメタ名	データ型	指定可能値	不正な値を指定した場合の扱われ方	省略値	説明
javax.faces.LIFECYCLE_ID	String	Java ID 名	DEFAULT	ませんが，FacesServlet が起動するときに，IllegalArgumentException 例外処理が発生します。	ユーザ定義のライフサイクル ID を指定します。
javax.faces.PARTIAL_STATE_SAVING	Boolean	true または false	true	true	アプリケーションでビューの状態を部分的に保存するメソッドを使用できるかどうかを指定します。
javax.faces.PROJECT_STAGE	String	Production，Development，UnitTest または SystemTest	Production	Production	ソフトウェア開発の工程にあった設定値を指定します。
javax.faces.SEPARATOR_CHAR	Character	web.xml の解析に識別ができる任意の文字列	文字列の先頭文字	:	レスポンスページに出力されるタグの Id 属性を区切る文字を指定します。
javax.faces.STATE_SAVING_METHOD	String	client または server	server	server	ビューの状態を保存する方法を指定します。
javax.faces.VALIDATE_EMPTY_FIELDS	String	auto，true または false	false	auto	サブミットされた値が空文字，または null であった場合に，その値を検証するかどうかを指定します。
javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR	Boolean	true または false	false	false	アプリケーションで Bean Validation を使用できなくするかどうかを指定します。※3

注※1　アーカイブ形式のファイルの修正はできないため，アーカイブ形式のアプリケーションの場合，このパラメタは利用できません。展開ディレクトリ形式のアプリケーションの場合に利用できます。

注※2　指定する ID 文字列が，"/"で始まらない場合，Warning メッセージは出ませんが，すべての状態の保存ではなく，部分的な状態の保存となります。このため，文字列は"/"で始めることを推奨します。

注※3　設定値ごとの動作については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「10.5.1　JSF から Bean Validation の利用手順」を参照してください。

3.5.2　アプリケーションサーバ独自のコンテキストパラメタ

アプリケーションサーバ独自のコンテキストパラメタを次の表に示します。

表 3-9　アプリケーションサーバ独自のコンテキストパラメタ

パラメタ名	データ型	指定可能値	不正な値の対処	省略値	説明
com.sun.faces.clientStateTimeout	Long	-9223372036854775808 ～ 9223372036854775807	タイムアウトは動作しない	タイムアウトは動作しない	ビューの状態に対するタイムアウト値を指定します。 ビューを表示してから次のビューを表示するまでの時間がタイムアウト値を超えるとビューは表示されません。 タイムアウト値を超えると，javax.faces.application.ViewExpiredException がスローされます。 このパラメタは javax.faces.STATE_SAVING_METHOD の値が"client"の場合に有効になります。 このパラメタを設定しない場合，タイムアウトは発生しません。 このパラメタの値に 1 を加えた値がタイムアウト値として有効になります。 例えば，0 が設定された場合は，1 分でタイムアウトが発生します。 負の値を設定すると，サブミット後のビューは必ず表示されなくなります。
com.sun.faces.disableUnicodeEscaping※1	String	auto，true または false	false	auto	クライアントへ返すレスポンスページの非 ASCII 文字の出力方法を決定します。 JSF の非 ASCII 文字の出力方法には，そのまま文字で出力するか，または文字参照で出力するかの 2 とおりがあります。 この機能はレスポンスページのエンコーディングの設定に影響を受けます。 また，この機能の影響する範囲は JSP と Facelets で異なります。 JSP では JSF タグ（HTML タグまたはコアタグ）に指定した文字が対象となります。 Facelets ではページに記載したすべての文字が対象となります。
com.sun.faces.numberOfLogicalViews	int	0～2147483647	15	15	論理ビュー※2 の数を指定します。 このパラメタは javax.faces.STATE_SAVING_METHOD の値が"server"の場合に有効になります。

パラメタ名	データ型	指定可能値	不正な値の対処	省略値	説明
com.sun.faces.numberOfLogicalViews	int	0～2147483647	15	15	このパラメタに設定した値を超えるビューにアクセスすると，参照されていない時間が最も長い論理ビューが，現在アクセスした論理ビューに置換されます。 置換されたビューをサブミットしても結果のページは表示されません。 このパラメタに 0 を指定すると，サブミット後のビューは必ず表示されなくなります。 負の値を指定した場合は，JSP と Facelets で挙動が異なります。 JSP の場合は，例外が発生しますが，ビューは表示され，サブミットもできます。ただし，ビューの状態は保存されません。 Facelets の場合は，例外が発生します。ビューは表示されません。
com.sun.faces.numberOfViewsInSession	int	0～2147483647	15	15	論理ビュー※2 に登録できるビューの状態の数を指定します。 このパラメタは javax.faces.STATE_SAVING_METHOD の値が"server"の場合に有効になります。 同じビューをサブミットした回数がこのパラメタに設定した値を超えると，参照されていない時間が最も長いビューの状態が，現在アクセスしたビューの状態に置換されます。 置換された状態を持っていたビューをサブミットしても結果のページは表示されません。 このパラメタに 0 を指定すると，サブミット後のビューは必ず表示されなくなります。 負の値を指定した場合は，JSP と Facelets で挙動が異なります。 JSP の場合は，例外が発生しますが，ビューは表示され，サブミットもできます。ただし，ビューの状態は保存されません。 Facelets の場合は，例外が発生します。ビューは表示されません。

注※1

com.sum.faces.disableUnicode.Escaping の設定値およびレスポンスページの出力との対応を次の表に示します。

表 3-10　com.sum.faces.disableUnicode.Escaping の設定値とレスポンスページの出力との対応

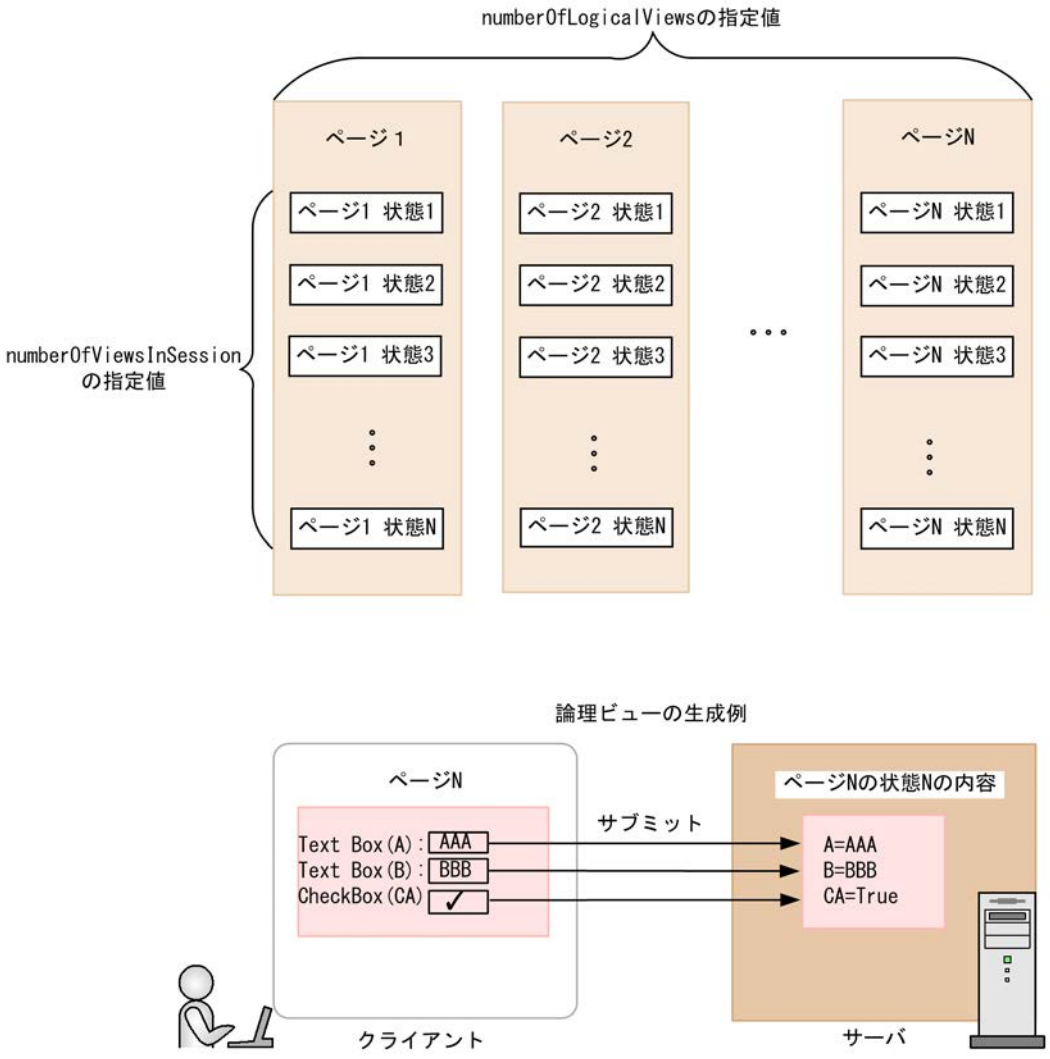
com.sun.faces.disableUnicodeEscaping の設定値	レスポンスページのエンコーディングが UTF 型，ISO-8859-1 の場合	レスポンスページのエンコーディングが UTF 型，ISO-8859-1 以外の場合
true	文字	文字
false	文字参照	文字参照
auto	文字	文字参照

注※2

論理ビューとは，JSF が保存しているレスポンスページのことです。クライアントが今までにアクセスしたビューを識別するために使用します。クライアントがビューにアクセスした際に論理ビューは作成されます。また，論理ビューはビューの状態を保持します。ビューの状態は，同じビューをサブミットするごとに一つずつ増加していきます。

numberOfLogicalViews および numberOfViewsInSession の指定値とビューの関係を次の図に示します。

図 3-2　numberOfLogicalViews および numberOfViewsInSession の指定値とビューの関係



numberOfLogicalViews に指定した数の論理ビューごとに，numberOfViewsInSession に指定した数の状態が保持されます。

クライアントからサーバに値をサブミットすると，新しい状態のビューがサーバに保存されます。なお，状態とは，論理ビューで記述した UI コンポーネントの情報のことです。

JSF では，ユーザが作成した Facelets ファイルや JSP ファイルを基にして生成した HTML ページの画面の情報（ビューの状態）と SessionScope を定義した ManagedBean クラスのオブジェクトを HTTP セッションに登録します。

3.5.3　Servlet の設定

Servlet の設定は，web.xml に定義します。web.xml の定義は，Servlet のバージョンごとに異なります。

(1)　Servlet2.5 の場合

JSF アプリケーションが動作するためには，web.xml に次のタグの定義が必要です。

- <servlet>

  <servlet>タグを使用して，FacesServlet クラスをサーブレットとして登録してください。

  web.xml には，次の例のように設定してください。

```
<servlet>
<servlet-name>FacesServlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>
```

- <servlet-mapping>

  servlet-mapping の要素を web.xm に定義する必要があります。

  web.xml には，次の例のように設定してください。

```
<servlet-mapping>
<servlet-name>FacesServlet</servlet-name>
<url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

(2)　Servlet3.0 の場合

Servlet3.0 の場合は，FacesServlet クラスの登録や URL マッピングの定義はデフォルトで設定されます。このため，web.xml の設定は必要ありません。web.xml の作成は任意です。

次に，web.xml での FacesServlet クラスの登録および URL マッピングの定義の有無と，動作の関係を示します。

条件 1：

条件

次のどちらかを満たす場合

- web.xml は作成しているが，FacesServlet クラスの登録と URL マッピングの定義はしていない
- web.xml を作成していない

動作

FacesServlet は自動的に初期化され，次のデフォルト URL にマッピングされます。

- /faces/*
- *.jsf

- *.faces

ユーザは，デフォルト URL を利用して FacesServlet にアクセスします。

条件 2：

条件

web.xml を作成し，FacesServlet クラスの登録，および URL のマッピングの定義を web.xml で設定している場合

動作

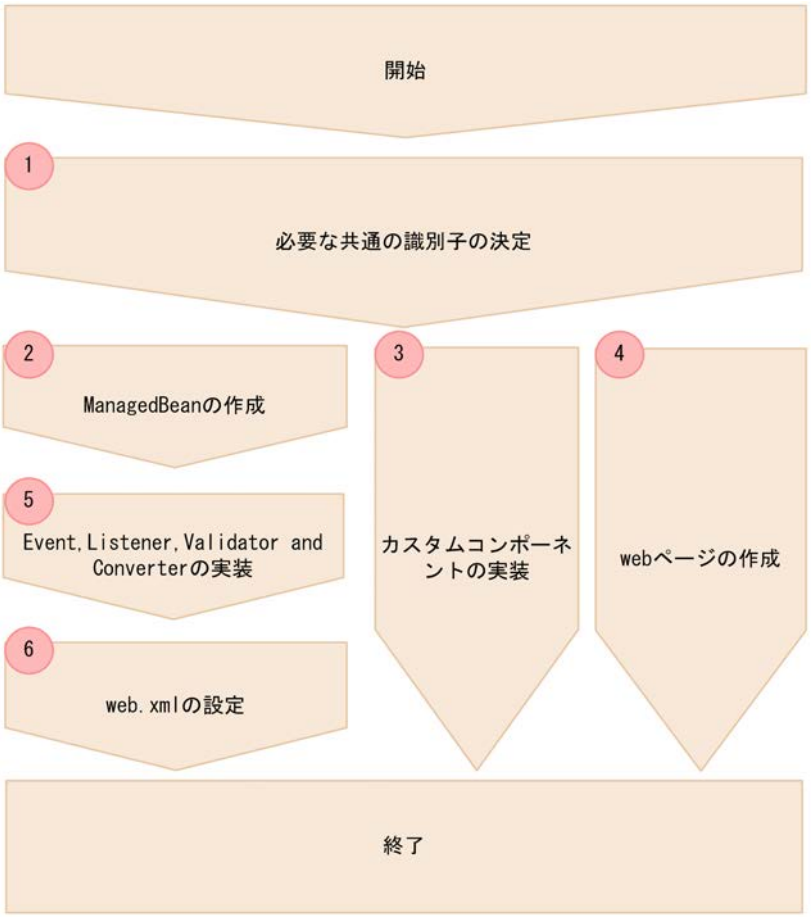
ユーザは，web.xml に設定した内容に従って FacesServlet にアクセスします。この場合，例 1 で示したデフォルトの設定は使用されません。

3.6　JSF アプリケーションの開発の流れ

この節では，JSF アプリケーションの開発の流れについて説明します。

JSF アプリケーションの開発では，次の図に示す作業を実施します。図中の番号に沿って，開発を進めてください。図中の 1.では，2.～6.を作成するために必要な共通の識別子を決定します。

図 3-3　JSF アプリケーションの開発の流れ



この手順は，JSF アプリケーションの作成までの開発手順です。実際に JSF アプリケーションを動作させるためには，上記で作成したファイルを基に EAR ファイルを作成し，アプリケーションサーバに JSF アプリケーションをデプロイする必要があります。

3.7　デバッグ用ログ（開発調査ログ）の利用

JSF アプリケーションの開発では，デバッグ用のログとして，**開発調査ログ**を利用できます。

開発している JSF アプリケーションで，障害や不具合などがあった場合，原因の調査および対象機能の詳細な動作を把握するために，開発調査ログを利用できます。

JSF アプリケーションでは，開発に関するメッセージが開発調査ログに出力されます。JSF アプリケーションのデバッグをする場合に確認が必要なメッセージは，メッセージに出力されたクラス名から判断できます。メッセージを出力するコンポーネントと出力されるクラス名について，次の表に示します。

表 3-11　JSF アプリケーションの開発調査ログに出力されるクラス名

コンポーネント名	出力されるクラス名	出力内容
JSF	javax.faces.～ com.sun.faces.～	• 障害に関するメッセージ • 設定ミスを知らせるメッセージ • 例外のスタックトレース • 機能の動作状況に関するメッセージ • 内部状態に関するメッセージ

なお，開発調査ログはデフォルトの設定では出力されません。必要に応じて設定を変更してください。

開発調査ログを出力するための設定，および開発調査ログの出力先，出力形式，ログレベルなどについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「24.4　開発調査ログに出力されるメッセージ」を参照してください。

また，JSF アプリケーションで利用している Bean Validation の開発調査ログの詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「10.6　デバッグ用ログ（開発調査ログ）の利用」を参照してください。

3.8　実行環境の設定

JSF および JSTL を使用したアプリケーションの実行環境の設定は，開発環境の設定と同じです。ライブラリを J2EE サーバのクラスパスに設定する必要があります。

設定するクラスパスについては，「3.4　クラスパスの設定（開発環境の設定）」を参照してください。

3.9　障害対応用の情報の出力および確認

JSF および JSTL を使用したアプリケーションで障害が発生した場合は，ログを参照して対処してください。詳細は，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.　トラブルシューティングで必要な資料の出力先と出力方法」および「5.　トラブルの分析」を参照してください。

3.10　JSF および JSTL 使用時の注意事項

この節では，JSF および JSTL 使用時の注意事項について説明します。

- コンテキストパラメタに指定した値は，次の規則に従って扱われます。
  - boolean 型のプロパティでは，小文字，大文字が区別されます。
  - すべてのコンテキストパラメタには，省略値があります。不正な値や指定できる範囲外の値を設定した場合は，省略値が利用され，誤ったコンテキストパラメタがメッセージに出力されます。
- JSF および JSTL のクラスパスの設定時，ライブラリのバージョンで注意しなければならない事項を次に示します。
  - 動作させたいバージョンのライブラリと異なるバージョンのライブラリを同時にクラスパスに指定しないでください。アプリケーションが予期しない動作をするおそれがあります。
  - アプリケーションのバージョンに合わせたライブラリをクラスパスに指定してください。
- アプリケーション作成時の注意事項を次に示します。
  - Component Container 09-50-02 以降の JSF を使用する場合は，リクエストされる可能性があるすべての xhtml ファイルに XML 宣言，DOCTYPE 宣言を記述してください。レスポンスには，リクエストされた xhtml ファイルの XML 宣言，DOCTYPE 宣言が出力されます。リクエストされた xhtml ファイルに XML 宣言，DOCTYPE 宣言が書かれていない場合は，レスポンスにどの XML 宣言，DOCTYPE 宣言が出力されるか保証されません。
- アプリケーション実行時の注意事項を次に示します。
  - アプリケーションの JAR ファイル内の META-INF/resources のリソースを更新した場合，リロード時に KDJE39556-W のメッセージが出力されます。しかし，この状態で JSF からリソースにアクセスした場合，更新後のリソースが取得できます。
  - JSF アプリケーションのページでは，script または style ブロック内ではエスケープ処理が行われません。

4 Web サーバ連携

この章では，Web サーバ連携に関する機能について説明します。

4.1　この章の構成

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

Web サーバ連携に関する機能と参照先を次の表に示します。

表 4-1　Web サーバ連携に関する機能と参照先

機能	参照先
Web サーバ（リダイレクタ）によるリクエストの振り分け	4.2
URL パターンでのリクエストの振り分け	4.3
ラウンドロビン方式によるリクエストの振り分け	4.4
POST データサイズでのリクエストの振り分け	4.5
通信タイムアウト（Web サーバ連携）	4.6
IP アドレス指定（Web サーバ連携）	4.7
エラーページのカスタマイズ（Web サーバ連携）※	4.8
ドメイン名指定でのトップページの表示	4.9
Web コンテナへのゲートウェイ情報の通知	4.10

注※

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

また，Web サーバと連携した場合，HTTP Server の SSL による認証やデータ暗号化，および Microsoft IIS の SSL による認証やデータ暗号化の機能も使用できます。HTTP Server の SSL による認証やデータ暗号化については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「7.2.5 HTTP Server の SSL の設定」を参照してください。Microsoft IIS の SSL による認証やデータ暗号化については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「7.2.6　Microsoft IIS の設定（Web リダイレクタ環境の場合）」を参照してください。なお，この機能が使用できるのは Web リダイレクタ環境でだけです。

なお，アプリケーションサーバで提供する Web サーバ連携の機能には，Java EE で規定された機能にアプリケーションサーバ独自の機能を拡張したものと，アプリケーションサーバ独自の機能として提供しているものがあります。アプリケーションサーバ独自の機能かどうかについては，「1.2　システムの目的と機能の対応」を参照してください。

Web サーバと連携する場合に必要な環境設定

Web サーバと連携する場合，次に示す環境設定が必要です。

- Smart Composer を使用する場合

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

- Smart Composer を使用しない場合

  次のどちらかの Web サーバの環境を設定してください。

- HTTP Server（付録 B）
- Microsoft IIS（付録 C）

4.2　Web サーバ（リダイレクタ）によるリクエストの振り分け

この節では，リダイレクタを使用したリクエストの振り分けについて説明します。

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

この節の構成を次の表に示します。

表 4-2　この節の構成（Web サーバ（リダイレクタ）によるリクエストの振り分け）

分類	タイトル	参照先
解説	リダイレクタを使用したリクエスト振り分けの仕組み	4.2.1
	リクエストの振り分け方法を設定するユーザ定義ファイル（Smart Composer 機能を使用する場合）	4.2.2
	リクエストの振り分け方法を設定するユーザ定義ファイル（Smart Composer 機能を使用しない場合）	4.2.3
注意事項	Web サーバ連携時の注意事項	4.2.4

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

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

表 4-3　使用する Web サーバの種類と定義

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

（凡例）－：該当しない

インプロセス HTTP サーバを使用する場合のリダイレクトによるリクエストの振り分けについては，「5.7　リダイレクトによるリクエストの振り分け」を参照してください。

4.2.1　リダイレクタを使用したリクエスト振り分けの仕組み

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

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

注※

ワーカプロセスは，実際には，J2EE サーバとなります。

(1)　リクエストの転送パターン

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

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

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

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

図 4-1　一つの Web サーバから一つのワーカプロセスへの転送

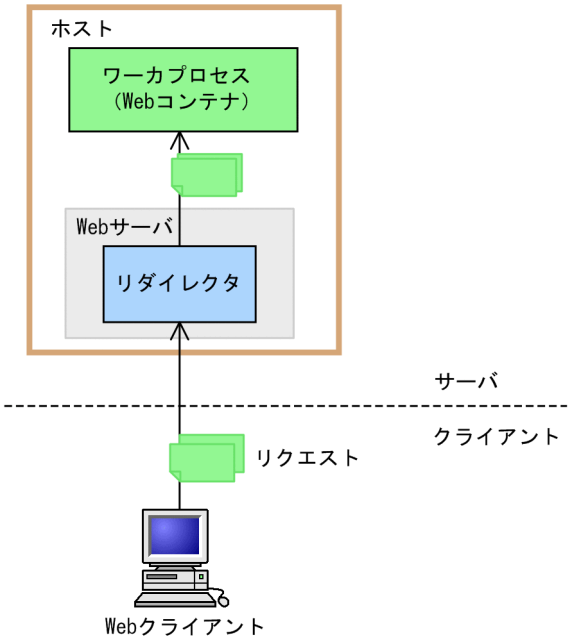
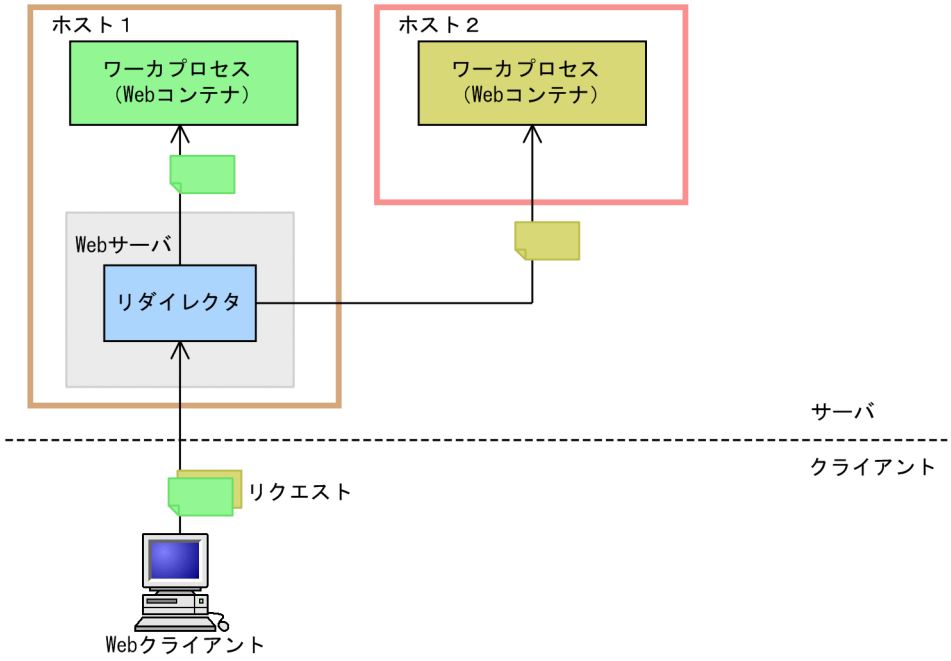


図 4-2　一つの Web サーバから複数のワーカプロセスへの転送



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

(2)　リクエストの振り分け方法

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

- **URL パターンによって振り分ける方法**
  一つの Web コンテナに特定の処理をさせたい場合，および複数の Web コンテナに処理を振り分けたい場合に使用します。
  ワーカプロセスが一つの場合と複数の場合に使用できます。
- **ロードバランサを使用してラウンドロビン方式で振り分ける方法**
  複数の Web コンテナに処理を振り分けたい場合に使用します。
- **POST データサイズによって振り分ける方法**
  複数の Web コンテナに処理を振り分けたい場合に使用します。この振り分け方法は，Web サーバに HTTP Server を使用している場合にだけ設定できます。
  なお，次の機能を使用している場合，この振り分け方法は適用できません。
  - セッションフェイルオーバ機能
  - ラウンドロビン方式によるリクエストの振り分け

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

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

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

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

ワーカプロセスの定義方法の詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル)」を参照してください。

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

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

- 簡易構築定義ファイル

  ワーカの定義，およびワーカごとのパラメタ，URL パターンとワーカのマッピングを設定します。URL パターンでのリクエスト振り分けを設定する場合に使用します。ラウンドロビン方式によるリクエスト振り分け，および POST データサイズでのリクエスト振り分けの場合，設定内容が物理ティアが free-tier に出力されます。

- workers.properties（ワーカ定義ファイル）

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

- mod_jk.conf（リダイレクタ動作定義ファイル）

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

簡易構築定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。ワーカ定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル)」を参照してください。リダイレクタ動作定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「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。

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

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

設定するファイルは，使用する Web サーバによって異なります。共通のユーザ定義ファイルと，Web サーバごとのユーザ定義ファイルを分けて説明します。なお，ユーザ定義ファイルのうち，httpsd.conf の詳細

については，マニュアル「HTTP Server」を参照してください。ほかのユーザ定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

(1)　共通のユーザ定義ファイル

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

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

(2)　HTTP Server を使用する場合のユーザ定義ファイル

HTTP Server を使用する場合のユーザ定義ファイルを次に示します。

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

(3)　Microsoft IIS を使用する場合のユーザ定義ファイル

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

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

(4)　注意事項

ユーザ定義ファイルに関する注意事項を次に示します。

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

4.2.4　Web サーバ連携時の注意事項

Web サーバと連携するときの注意事項について説明します。

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

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

(2)　HTTP Server を使用するときの注意事項

アプリケーションサーバで構築したシステムでは，HTTP Server の仮想ホスト機能を使用できません。同一ホスト上で，複数の HTTP Server を起動したい場合は，Management Server を使用してください。

(3)　Microsoft IIS を使用するときの注意事項

Microsoft IIS を使用するときの注意事項を説明します。

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

  このため，Microsoft IIS 用リダイレクタ以降に実行される ISAPI フィルタでは，Microsoft IIS が最初に受信したリクエスト URL を取得できません。Microsoft IIS が受信したリクエスト URL を ISAPI フィルタで取得したい場合は，ISAPI フィルタの優先順位を Microsoft IIS 用リダイレクタよりも高く設定する必要があります。なお，優先順位を調整するために，Microsoft IIS 用リダイレクタの優先順位を「中」または「低」に変更する必要がある場合は，Microsoft IIS 用リダイレクタ動作定義ファイルの filter_priority キーで指定します。filter_priority キーについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「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。
- Microsoft IIS と連携する場合，次の HTTP リクエストヘッダは Web クライアントで指定していても，Web アプリケーションでは受信できません。
  - tomcaturl
  - tomcatquery
  - tomcatworker
  - tomcattranslate

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

4.3　URL パターンでのリクエストの振り分け

この節では，URL パターンでのリクエストの振り分けについて説明します。

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

この節の構成を次の表に示します。

表 4-4　この節の構成（URL パターンでのリクエストの振り分け）

分類	タイトル	参照先
解説	URL パターンでのリクエストの振り分けの概要	4.3.1
	URL パターンの種類と適用されるパターンの優先順位	4.3.2
設定	実行環境での設定（Smart Composer 機能を使用する場合）	4.3.3
	実行環境での設定（Smart Composer 機能を使用しない場合）	4.3.4

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

4.3.1　URL パターンでのリクエストの振り分けの概要

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

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

リダイレクタによるリクエストの振り分けの例を，次の図に示します。

図 4-3　リダイレクタによるリクエストの振り分け（特定のリクエストを Web コンテナに転送する場合）

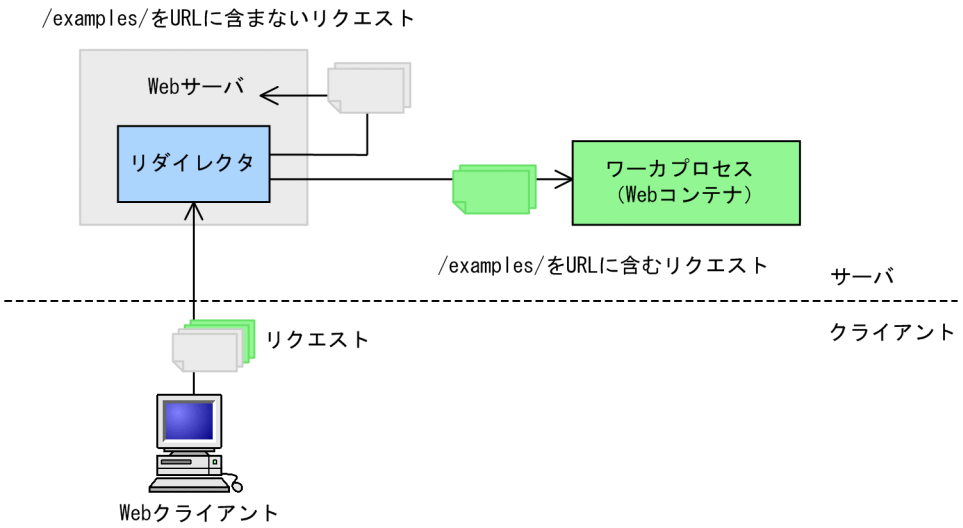
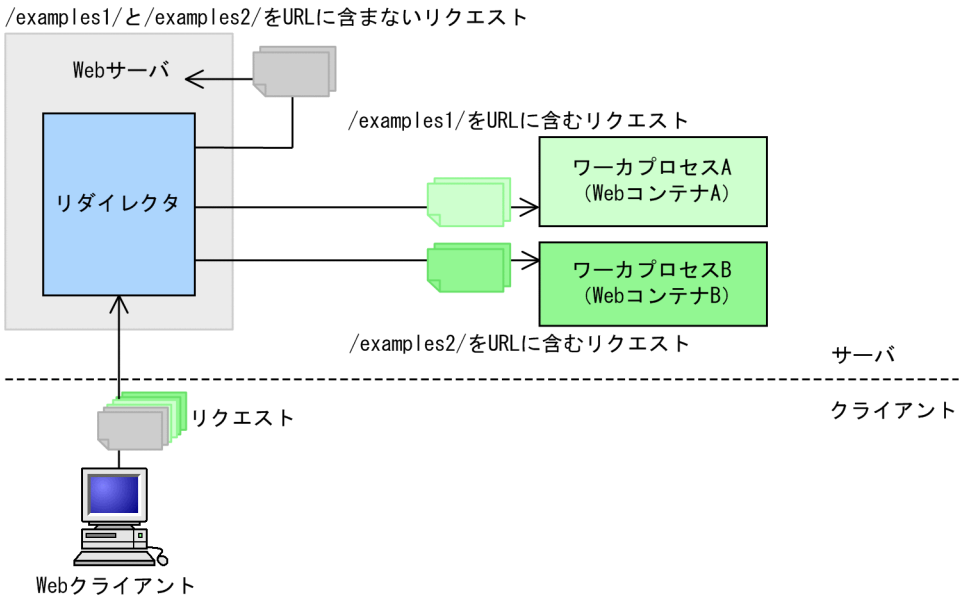


図 4-4　リダイレクタによるリクエストの振り分け（リクエストを複数の Web コンテナに振り分けて転送する場合）



4.3.2　URL パターンの種類と適用されるパターンの優先順位

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

(1)　URL パターンの種類

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

- **完全パス指定**

  完全に一致するパターンです。

  URL の形式：

  /<パス>

  ルートだけ指定する場合は，「/」。

  /<パス>に指定できる文字：

  次の文字を使用した，1 文字以上の文字列を指定します。

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

  例：

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

- **パス指定**

  パスが一致するパターンです。

  URL の形式：

  /<パス>/*

  すべてのリクエストを対象にする場合の形式は，「/*」。

/<パス>に指定できる文字：

次の文字を使用した，1 文字以上の文字列を指定します。

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

例：

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

- **拡張子指定**

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

URL の形式：

/<パス>/*.<拡張子>

すべてのパスを対象にする場合の形式は，「/*.<拡張子>」。

<パス>，および<拡張子>に指定できる文字：

次の文字を使用した，1 文字以上の文字列を指定します。

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

例：

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

- **サフィックス指定**

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

URL の形式：

/<パス>/*<サフィックス>

すべてのパスを対象にする場合の形式は，「/*<サフィックス>」。

<パス>，および<サフィックス>に指定できる文字：

次の文字を使用した，1 文字以上の文字列を指定します。

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

例：

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

! 注意事項

URL パターンの指定に関する注意事項を次に示します。

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

  「完全パス指定」，および「パス指定」の場合，先に書いた URL パターンのマッピングが有効となります。「拡張子指定」，または「サフィックス指定」の場合は，あとに書いたものが有効となります。

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

---

(2)　適用される URL パターンの優先順位

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

1.「完全パス指定」に一致しない場合

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

次のように二つのマッピングが定義されている場合を例に，説明します。

マッピング定義：

/examples/* worker1

/examples/jsp/* worker2

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

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

「パス指定」より，「拡張子指定」または「サフィックス指定」が優先されます。

次のように二つのマッピングが定義されている場合を例に，説明します。

マッピング定義：

/examples/jsp/* worker1

/examples/jsp/*.jsp worker2

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

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

あとに指定した URL パターンが優先されます。

次のように二つのマッピングが定義されている場合を例に，説明します。

マッピング定義：

/examples/*.jsp worker1

/examples/*jsp worker2

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

! 注意事項

適用されるパターンの優先順位の判定に関する注意事項を次に示します。

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

  例：

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

- リクエスト URL にパスパラメタ（URL の「;」以降の文字列）がある場合，その部分は URL パターンとの比較には使用されません。
  例：
  リクエスト URL が"/examples/jsp/index.jsp;jsessionid=0000"の場合，比較に使用される URL は"/examples/jsp/index.jsp"となります。
- リクエスト URL はパスの正規化をしてから，URL パターンとの一致が判定されます。
  例：
  リクエスト URL が"/examples/../examples/./jsp//index.jsp"の場合，比較に使用される URL は"/examples/jsp/index.jsp"となります。
- URL パターンの正規化はしません。そのため，「./」，「../」などを含む URL パターンを定義しても，リクエストと一致することはありません。
- Windows の場合，「拡張子指定」の URL パターンの拡張子について，大文字小文字は区別されないで判定されます。

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

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

(1)　設定の手順

URL パターンによるリクエストの振り分けは，次の手順で設定します。

1. **簡易構築定義ファイルで，ワーカ，および URL パターンとワーカのマッピングを定義します。**
   ワーカ名のリスト，ワーカのタイプ（「ajp13」を設定），ポート番号，ホスト名などを設定します。
   また，すでにマッピングが定義されている場合は，定義を削除，または置き換えてください。
2. **Web サーバの環境を設定して，Web サーバを再起動します。**
   Web サーバの設定については，「付録 B　HTTP Server の設定に関する注意事項」を参照してください。

(2)　簡易構築定義ファイルでの設定

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

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

表 4–5　簡易構築定義ファイルでの URL パターンによるリクエストの振り分けの定義

分類	パラメタ	説明
ワーカの定義	worker.list	一つまたは複数のワーカ名のリストを指定します。
	worker.<ワーカ名>.host	ワーカのホスト名，または IP アドレスを指定します。
	worker.<ワーカ名>.port	ワーカのポート番号を指定します。
	worker.<ワーカ名>.type	ワーカのタイプ（ajp13，lb または post_size_lb）を指定します。

分類	パラメタ	説明
ワーカの定義	worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカのコネクション数を指定します。 Windows の場合にだけ指定できます。
	worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。
	worker.<ワーカ名>.delegate_error_code	エラーページの作成を Web サーバに委任する場合に利用するエラーステータスコードを指定します。
URL パターンとワーカのマッピングの定義	JkMount	URL パターンと，worker.list で指定されているワーカのどれかとの組み合わせを指定します。

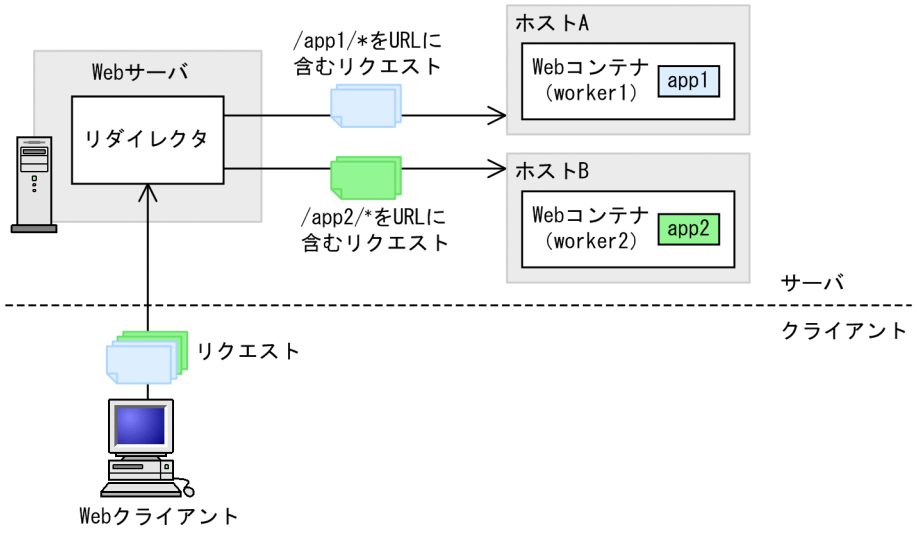
注　<ワーカ名>には，worker.list パラメタで指定したワーカの名称を定義します。

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(3)　設定例

URL パターンでのリクエスト振り分けの例を次の図に示します。

図 4-5　URL パターンでのリクエスト振り分けの例



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

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

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

簡易構築定義ファイルの例

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

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

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

(1)　設定の手順

URL パターンによるリクエストの振り分けは，次の手順で設定します。

1. workers.properties でワーカを定義します。

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

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

   workers.properties（ワーカ定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル）」を参照してください。

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

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

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

   uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.4　uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル)」を参照してください。

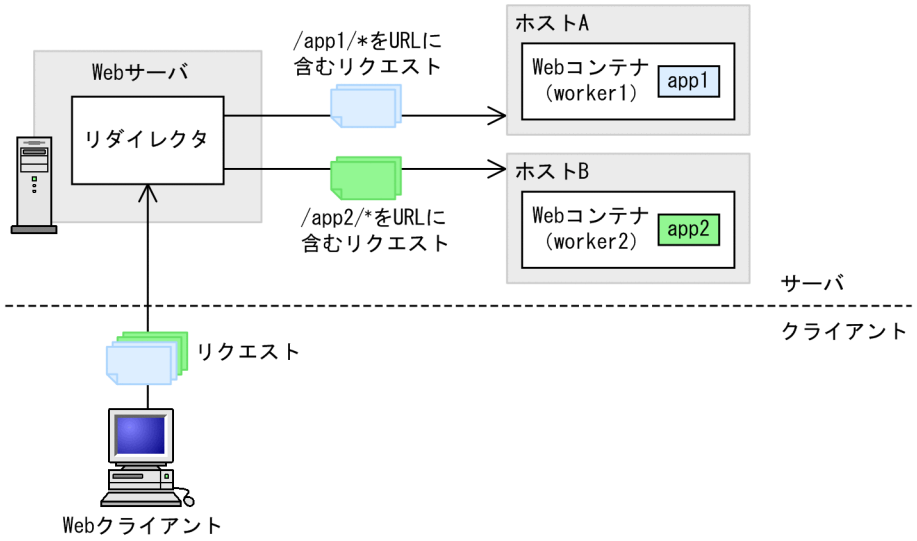
3. Web サーバの環境を設定して，Web サーバを再起動します。

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

(2)　設定例

URL パターンでのリクエスト振り分けの例を次の図に示します。

図 4-6　URL パターンでのリクエスト振り分けの例



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

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

workers.properties の例（Windows の場合）

```
worker.list=worker1,worker2

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

```
worker.worker1.cachesize=64

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

workers.properties の例（UNIX の場合）

```
worker.list=worker1,worker2

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

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

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

mod_jk.conf の例（HTTP Server の場合）

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

uriworkermap.properties の例（Microsoft IIS の場合）

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

4.4　ラウンドロビン方式によるリクエストの振り分け

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

この節の構成を次の表に示します。

表 4-6　この節の構成（ラウンドロビン方式によるリクエストの振り分け）

分類	タイトル	参照先
解説	ラウンドロビン方式によるリクエストの振り分けの概要	4.4.1
	ラウンドロビン方式によるリクエストの振り分けの例	4.4.2
	ラウンドロビン方式によるリクエストの振り分けの定義	4.4.3
設定	実行環境での設定（Smart Composer 機能を使用する場合）	4.4.4
	実行環境での設定（Smart Composer 機能を使用しない場合）	4.4.5
注意事項	ラウンドロビン方式によるリクエストの振り分けに関する注意事項	4.4.6

注 「実装」および「運用」について，この機能固有の説明はありません。

Web コンテナがクラスタ構成で配置されている場合，リダイレクタを使用して，ラウンドロビン方式でそれぞれの Web コンテナにリクエストを振り分けられます。リダイレクタは，各リクエストに付けられたセッション ID を参照することで，同一の Web クライアントから来たリクエストが常に同一の Web コンテナへ転送されるように，リクエストを振り分けます。

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

4.4.1　ラウンドロビン方式によるリクエストの振り分けの概要

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

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

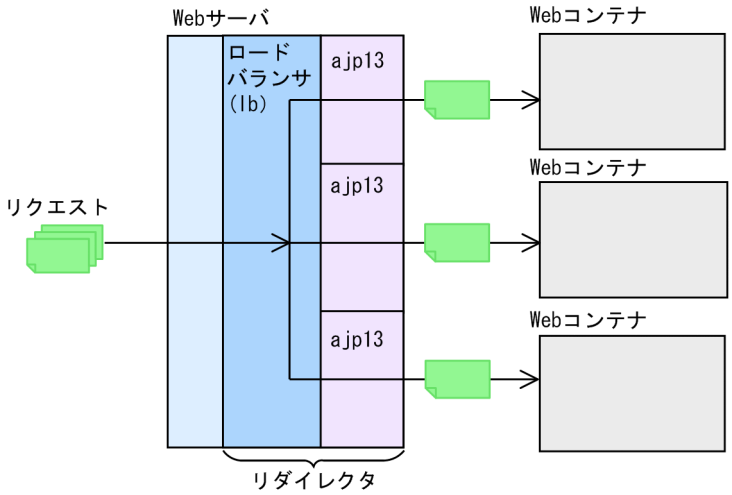
参考

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

4.4.2　ラウンドロビン方式によるリクエストの振り分けの例

ロードバランサを使用したリクエストの振り分けの例を次の図に示します。

図 4-7　ロードバランサを使用したリクエストの振り分けの例



4.4.3　ラウンドロビン方式によるリクエストの振り分けの定義

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

```
#worker.list=loadbalancer1

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

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

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

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

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

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

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

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

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

振り分け先となる各ワーカに負荷分散値を設定して，リクエスト振り分けの比率を定義すると，ホストごとの負荷の割合を調整できます。リダイレクタは，この比率で HTTP リクエスト単位にラウンドロビンでリ

クエストを振り分けるので，比率が高いワーカほど転送されるリクエストの割合が多くなります。ただし，同じセッションに属する HTTP リクエストは前回と同じワーカに振り分けられます。

(1)　設定の手順

ラウンドロビン方式によるリクエスト振り分けは，次の手順で設定します。

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

   ロードバランサの定義

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

   各ワーカの定義

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

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

   workers.properties（ワーカ定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル)」を参照してください。

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

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

   mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

3. Web サーバの環境を設定して，Web サーバを再起動します。

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

(a)　注意事項

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

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

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

(b)　ディレクティブの指定例

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

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

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

表 4-7　workers.properties および mod_jk.conf で定義するキー（ラウンドロビン方式によるリクエストの振り分けの場合）

ファイルの種類	キー名	説明
workers.properties	worker.list	一つまたは複数のワーカ名のリストを指定します。
	worker.<ワーカ名>.host	ワーカのホスト名，または IP アドレスを指定します。
	worker.<ワーカ名>.port	ワーカのポート番号を指定します。
	worker.<ワーカ名>.type	ワーカのタイプを指定します。ロードバランサには「lb」を，負荷分散の対象となるワーカには「ajp13」を指定します。
	worker.<ワーカ名>.balanced_workers	負荷分散の対象となるワーカのリストを指定します。
	worker.<ワーカ名>.lbfactor	負荷分散値を指定します。
	worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカのコネクション数を指定します。 Windows の場合にだけ指定できます。
	worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。
	worker.<ワーカ名>.delegate_error_code	エラーページの作成を Web サーバに委任する場合に利用するエラーステータスコードを指定します。
mod_jk.conf	JkMount	URL パターンと，worker.list で指定されているワーカのどれかとの組み合わせを指定します。

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

ワーカの種類ごとに指定できるパラメタを次の表に示します。

表 4-8　ワーカの種類ごとに指定できるキー

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

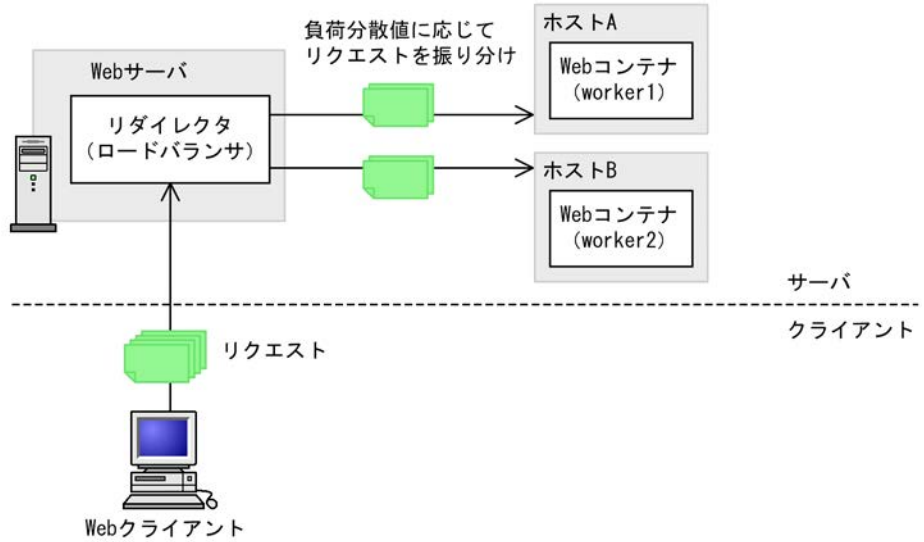
キー名	ワーカの種類（worker.<ワーカ名>.type キーの指定値）	
	ロードバランサ（「lb」を指定）	ワーカ（「ajp13」を指定）
worker.<ワーカ名>.balanced_workers	○	×
worker.<ワーカ名>.lbfactor	×	△
worker.<ワーカ名>.cachesize	×	△
worker.<ワーカ名>.receive_timeout	×	△
worker.<ワーカ名>.delegate_error_code	×	△

（凡例）○：指定できる　×：指定できない　△：任意に指定できる

(3)　設定例

ラウンドロビン方式によるリクエスト振り分けの例を次の図に示します。

図 4-8　ラウンドロビン方式によるリクエスト振り分けの例



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

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

workers.properties の例（Windows の場合）

```
worker.list=loadbalancer1

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

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

worker.worker2.port=8007
```

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

workers.properties の例（UNIX の場合）

```
worker.list=loadbalancer1

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

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

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

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

mod_jk.conf の例

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

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

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

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

(1)　設定の手順

ラウンドロビン方式によるリクエスト振り分けは，次の手順で設定します。

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

   ロードバランサの定義

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

   各ワーカの定義

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

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

   workers.properties（ワーカ定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル）」を参照してください。

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

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

mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.4　uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル)」を参照してください。

3. Web サーバの環境を設定して，Web サーバを再起動します。

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

(a)　注意事項

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

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

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

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

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

(b)　ディレクティブの指定例

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

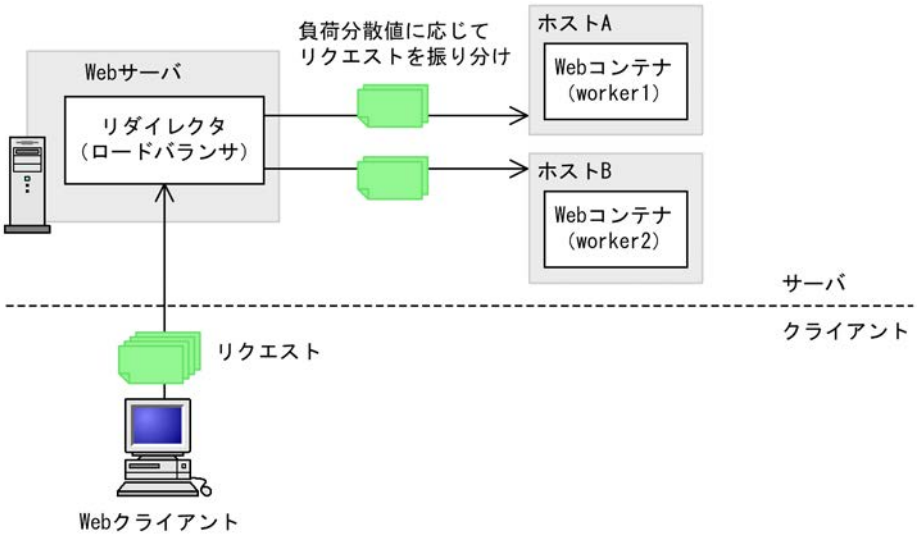
(2)　workers.properties および mod_jk.conf での設定

workers.properties および mod_jk.conf での設定については，Smart Composer を使用する場合と同様です。「4.4.4(2)　workers.properties および mod_jk.conf での設定」を参照してください。

(3)　設定例

ラウンドロビン方式によるリクエスト振り分けの例を次の図に示します。

図 4-9　ラウンドロビン方式によるリクエスト振り分けの例



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

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

workers.properties の例（Windows の場合）

```
worker.list=loadbalancer1

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

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

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

workers.properties の例（UNIX の場合）

```
worker.list=loadbalancer1

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

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

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

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

mod_jk.conf の例（HTTP Server の場合）

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

uriworkermap.properties の例（Microsoft IIS の場合）

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

4.4.6　ラウンドロビン方式によるリクエストの振り分けに関する注意事項

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

- **J2EE アプリケーション停止中の Web コンテナへのリクエストの送信**
  ラウンドロビン方式でリクエストを振り分ける場合，J2EE アプリケーションが停止中であっても Web コンテナへリクエストが送信されます。このため，J2EE アプリケーションの変更は，すべての Web コンテナをシステムから隔離した状態で実施する必要があります。
- **負荷分散機によるヘルスチェックの無効**
  負荷分散機と，ラウンドロビン方式によるリクエストの振り分けを併用した場合，J2EE サーバに障害が発生したときも，リダイレクタで正常な Web コンテナへ転送されます。このため，負荷分散機で J2EE サーバの障害が検知できなくなり，Web コンテナの監視が行えません。

4.5　POST データサイズでのリクエストの振り分け

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

この節の構成を次の表に示します。

表 4–9　この節の構成（POST データサイズでのリクエストの振り分け）

分類	タイトル	参照先
解説	POST データサイズでのリクエストの振り分けの概要	4.5.1
	POST データサイズによるリクエストの振り分けの例	4.5.2
	リクエストの振り分け条件	4.5.3
	POST データサイズによるリクエストの振り分けの定義	4.5.4
設定	実行環境での設定（Smart Composer 機能を使用する場合）	4.5.5
	実行環境での設定（Smart Composer 機能を使用しない場合）	4.5.6

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

4.5.1　POST データサイズでのリクエストの振り分けの概要

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

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

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

! 注意事項

HTTP Cookie または URL 書き換えによる制御でセッションが管理されている場合でも，POST データサイズによる振り分けを設定しているときは，POST データサイズによって，リクエストの振り分け先が決定されます。このため，同一クライアントからのリクエストの場合でも，HttpSession のセッション ID は引き継がれません。

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

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

4.5.2　POST データサイズによるリクエストの振り分けの例

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

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

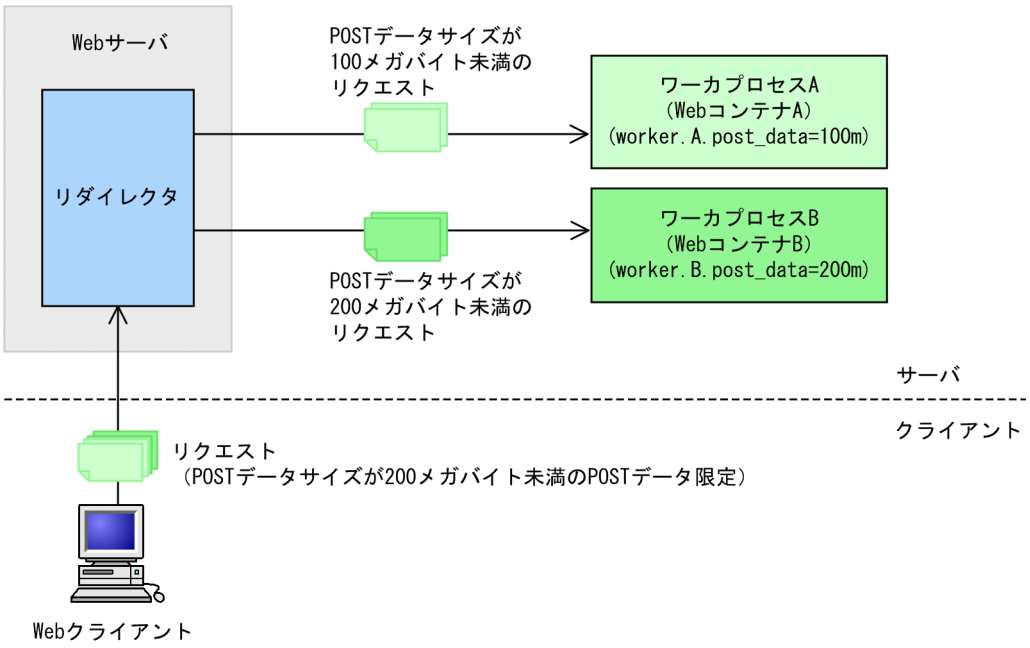
  ここでは，次の条件を満たすリクエストが，POST リクエスト振り分けワーカに転送されるものとして説明します。

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

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

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

  図 4-10　POST データサイズによるリクエストの振り分けの例（リクエストが限定できる場合）



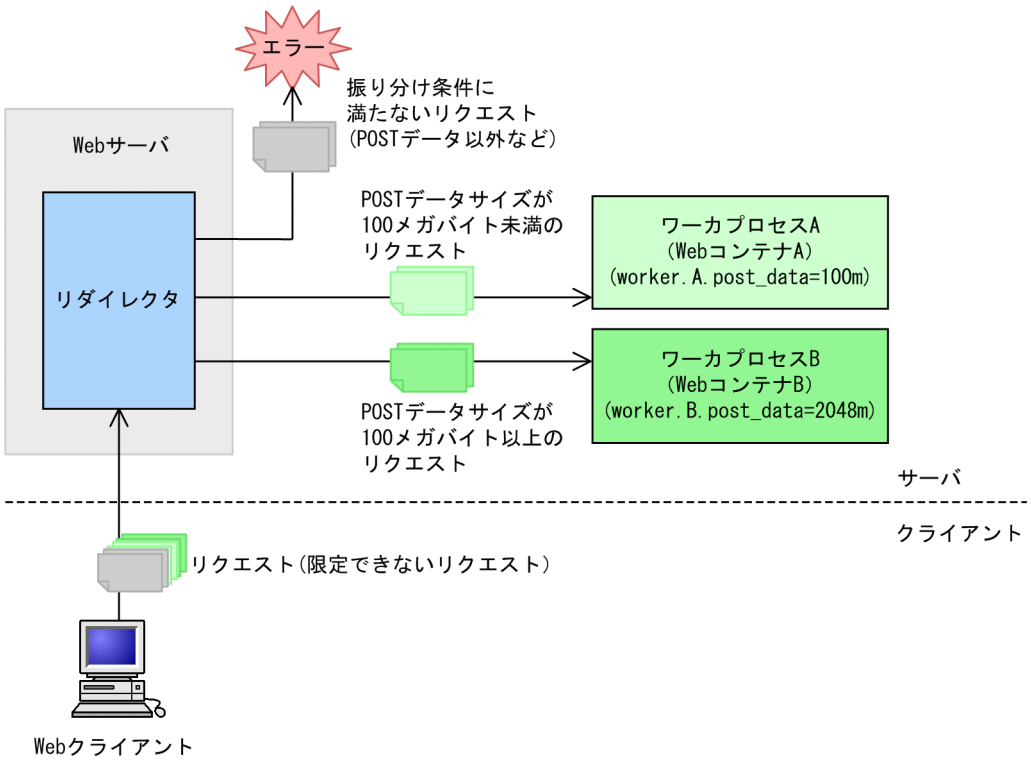
  この図では，POST リクエスト転送先ワーカを二つ用意し，POST データサイズが 100 メガバイト以上 200 メガバイト未満のリクエストがワーカプロセス B に転送されるように，それぞれに POST データサイズの上限値を設定しています。リクエストの POST データサイズが上限値未満の場合に，それぞれの POST リクエスト転送先ワーカに振り分けられます。リクエストの POST データサイズが，複数の POST リクエスト転送先ワーカに当てはまる場合，POST データサイズの上限値が最も小さい POST リクエスト転送先ワーカに，リクエストは振り分けられます。例えば，POST データサ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

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

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

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

図 4-11　POST データサイズによるリクエストの振り分けの例（リクエストが限定できない場合）



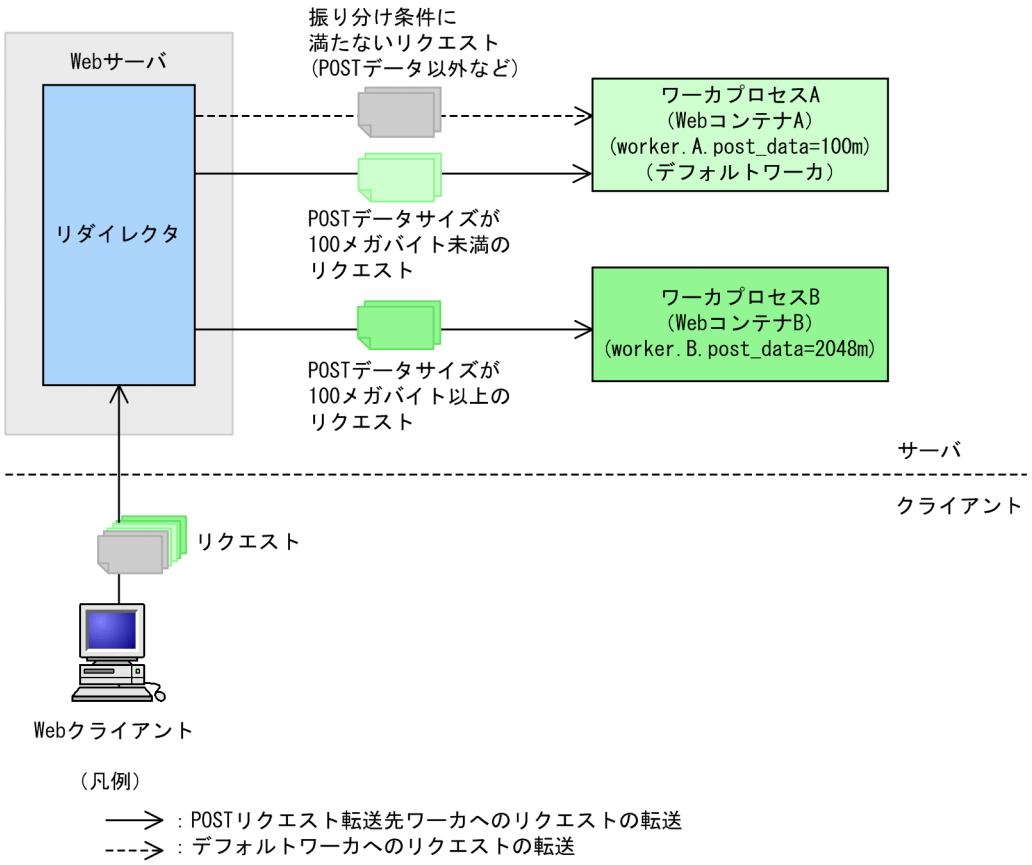
この図では，POST リクエスト転送先ワーカを二つ用意し，それぞれに POST データサイズの上限値を設定しています。長大な POST データのリクエストすべてをワーカプロセス B で処理するように，ワーカプロセス B の POST データサイズの上限値には最大値を設定しています。ワーカプロセス A の上限値以上（100 メガバイト以上）の POST データのリクエストは，すべてワーカプロセス B に転送されます。なお，この場合，POST データ以外のリクエストや，POST データサイズが参照できないリクエストなどが転送されると，リクエスト振り分けワーカで振り分けられないため，リダイレクタによ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

POST データサイズでリクエストを振り分ける場合，リクエストの振り分け条件に満たないリクエストが POST リクエスト振り分けワーカに転送されると，リダイレクタによってエラーステータスコード 400 が返されてエラーとなります。リクエストの振り分け条件については，「4.5.3　リクエストの振り分け条件」を参照してください。

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

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

図 4-12　POST データサイズによるリクエストの振り分けの例（デフォルトワーカを設定した場合）



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

4.5.3　リクエストの振り分け条件

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

POST リクエスト転送先ワーカに振り分けられるリクエストの条件

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

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

4.5.4　POST データサイズによるリクエストの振り分けの定義

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

```
#worker.list=postsizelb1
#worker.postsizelb1.type=post_size_lb
```

```
#worker.postsizelb1.post_size_workers=worker1,worker2
#worker.postsizelb1.default_worker=worker1
```

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

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

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

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

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

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

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

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

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

(1)　設定の手順

POST データサイズでのリクエスト振り分けは，次の手順で設定します。

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

   POST リクエスト振り分けワーカの定義

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

   各 POST リクエスト転送先ワーカの定義

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

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

workers.properties（ワーカ定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル）」を参照してください。

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

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

mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）」を参照してください。

3. Web サーバの環境を設定して，Web サーバを再起動します。

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

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

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

表 4-10　workers.properties および mod_jk.conf で定義するキー（POST データサイズでのリクエストの振り分けの場合）

ファイルの種類	キー名	説明
workers.properties	worker.list	一つまたは複数のワーカ名のリストを指定します。
	worker.<ワーカ名>.host	ワーカのホスト名，または IP アドレスを指定します。
	worker.<ワーカ名>.port	ワーカのポート番号を指定します。
	worker.<ワーカ名>.type	ワーカのタイプを指定します。POST リクエスト振り分けワーカには「post_size_lb」を，振り分け対象となる POST リクエスト転送先ワーカには「ajp13」を指定します。
	worker.<ワーカ名>.post_size_workers	POST データサイズによる振り分けの対象となるワーカのリストを指定します。
	worker.<ワーカ名>.post_data	リクエストの POST データサイズの上限値（バイト単位）を指定します。
	worker.<ワーカ名>.default_worker	リクエストの振り分け先に該当するワーカが，クラスタ構成内の POST リクエスト転送先ワーカにない場合に，リクエストを転送するワーカ（デフォルトワーカ）を指定します。
	worker.<ワーカ名>.cachesize	リダイレクタで再利用するワーカのコネクション数を指定します。 Windows の場合にだけ指定できます。
	worker.<ワーカ名>.receive_timeout	通信タイムアウト値を指定します。
	worker.<ワーカ名>.delegate_error_code	エラーページの作成を Web サーバに委任する場合に利用するエラーステータスコードを指定します。
mod_jk.conf	JkMount	URL パターンと，worker.list で指定されているワーカのどれかとの組み合わせを指定します。

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

ワーカの種類ごとに指定できるキーを次の表に示します。

表 4-11　ワーカの種類ごとに指定できるキー

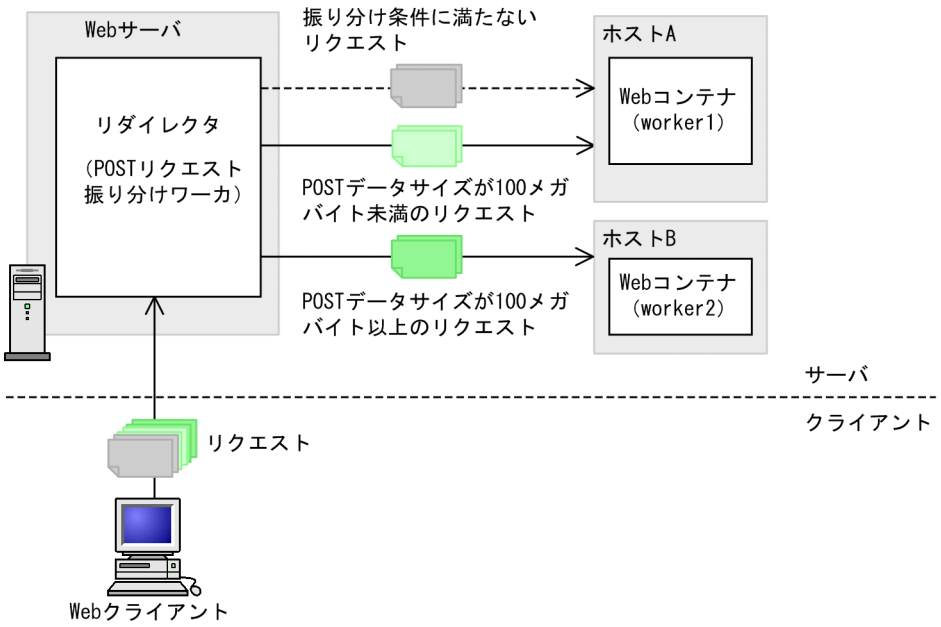
キー名	ワーカの種類（worker.<ワーカ名>.type キーの指定値）	
	POST リクエスト振り分けワーカ（「post_size_lb」を指定）	POST リクエスト転送先ワーカ（「ajp13」を指定）
worker.<ワーカ名>.host	×	○
worker.<ワーカ名>.port	×	○
worker.<ワーカ名>.type	○	○
worker.<ワーカ名>.post_size_workers	○	×
worker.<ワーカ名>.post_data	×	○
worker.<ワーカ名>.default_worker	△	×
worker.<ワーカ名>.cachesize	×	△
worker.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

（凡例）○：指定できる。 ×：指定できない。 △：任意に指定できる。

(3)　設定例

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

図 4-13　POST データサイズでのリクエスト振り分けの例



この例では，「/examples」以下のリクエストのうち，POST データサイズが 100 メガバイト未満のリクエストをホスト A，POST データサイズが 100 メガバイト以上 200 メガバイト未満のリクエストをホスト B に振り分けます。リクエストの振り分け条件に満たないリクエストは，デフォルトワーカに設定したホスト A に振り分けます。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。リクエストの振り分け条件については，「4.5.3　リクエストの振り分け条件」を参照してください。

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

workers.properties の例（Windows の場合）

```
worker.list=postsizelb1

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

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

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

workers.properties の例（UNIX の場合）

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1,worker2
```

```
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

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

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

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

mod_jk.conf の例

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

参考

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

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

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

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

！ 注意事項

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

(1)　設定の手順

POST データサイズでのリクエスト振り分けは，次の手順で設定します。

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

   POST リクエスト振り分けワーカの定義

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

   必要に応じて，デフォルトワーカも設定します。

   各 POST リクエスト転送先ワーカの定義

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

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

workers.properties（ワーカ定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル)」を参照してください。

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

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

mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

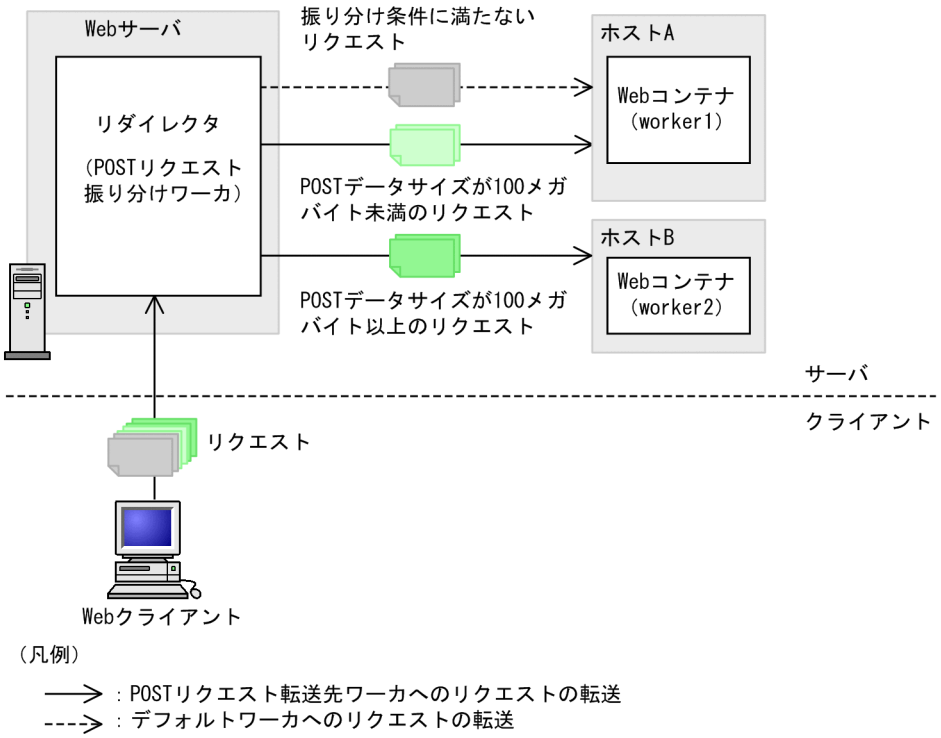
3. Web サーバの環境を設定して，Web サーバを再起動します。

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

(2)　設定例

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

図 4-14　POST データサイズでのリクエスト振り分けの例



この例では，「/examples」以下のリクエストのうち，POST データサイズが 100 メガバイト未満のリクエストをホスト A，POST データサイズが 100 メガバイト以上 200 メガバイト未満のリクエストをホスト B に振り分けます。リクエストの振り分け条件に満たないリクエストは，デフォルトワーカに設定したホスト A に振り分けます。ホスト A のワーカ名は「worker1」，ホスト B のワーカ名は「worker2」です。リクエストの振り分け条件については，「4.5.3　リクエストの振り分け条件」を参照してください。

workers.properties の例を次に示します。ここでは，POST リクエスト振り分けワーカ，POST リクエスト転送先ワーカおよびデフォルトワーカを定義します。POST データサイズの上限値は，「worker1」に

「100m」,「worker2」に「2048m」（POST データサイズの上限値の最大値）を指定します。デフォルトワーカには,「worker1」を指定します。

workers.properties の例（Windows の場合）

```
worker.list=postsizelb1

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

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

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

workers.properties の例（UNIX の場合）

```
worker.list=postsizelb1

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

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

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

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

mod_jk.conf の例

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

参考

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

4.6　通信タイムアウト（Web サーバ連携）

この節では，Web サーバ連携での通信タイムアウトについて説明します。

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

この節の構成を次の表に示します。

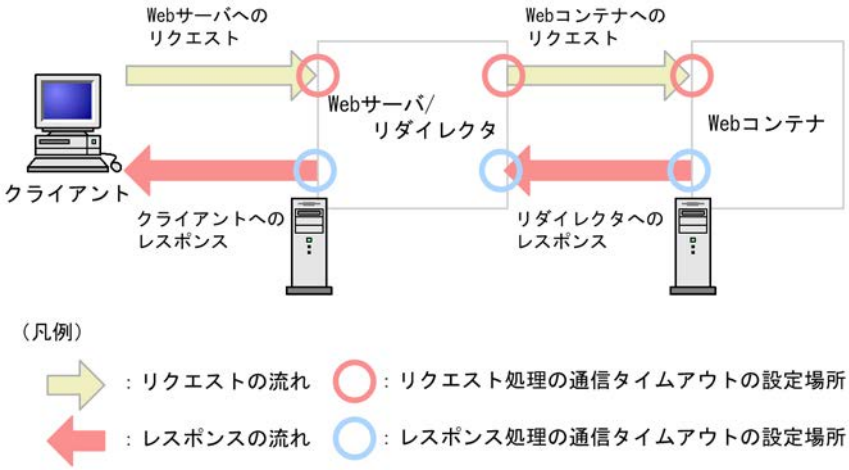
表 4-12　この節の構成（通信タイムアウト（Web サーバ連携））

分類	タイトル	参照先
解説	リクエスト送受信時の通信タイムアウト	4.6.1
	レスポンス送受信時の通信タイムアウト	4.6.2
設定	通信タイムアウトの設定	4.6.3
	リクエスト送受信時の通信タイムアウトの設定（Smart Composer 機能を使用する場合）	4.6.4
	リクエスト送受信時の通信タイムアウトの設定（Smart Composer 機能を使用しない場合）	4.6.5
	レスポンス送受信時の通信タイムアウトの設定（Smart Composer 機能を使用する場合）	4.6.6
	レスポンス送受信時の通信タイムアウトの設定（Smart Composer 機能を使用しない場合）	4.6.7

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

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

図 4-15　タイムアウトを設定できる通信

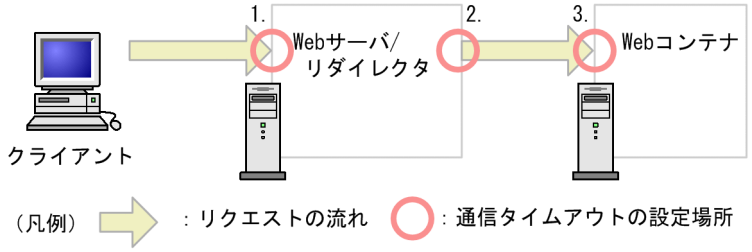


通信タイムアウトの設定について，リクエストの受信とレスポンスの送信に分けて説明します。

4.6.1　リクエスト送受信時の通信タイムアウト

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

図 4-16　リクエスト送受信時の通信タイムアウトの設定場所



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

1. Web サーバでのリクエスト受信時（クライアント－Web サーバ）

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

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

2. リダイレクタでのリクエスト送信時（リダイレクター－Web コンテナ）

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

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

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

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

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

(1)　Web サーバでのリクエスト受信時

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

検知できる障害

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

(2)　リダイレクタでのリクエスト送信時

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

(a)　設定できる通信タイムアウトと検知できる障害

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

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

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

検知できる障害

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

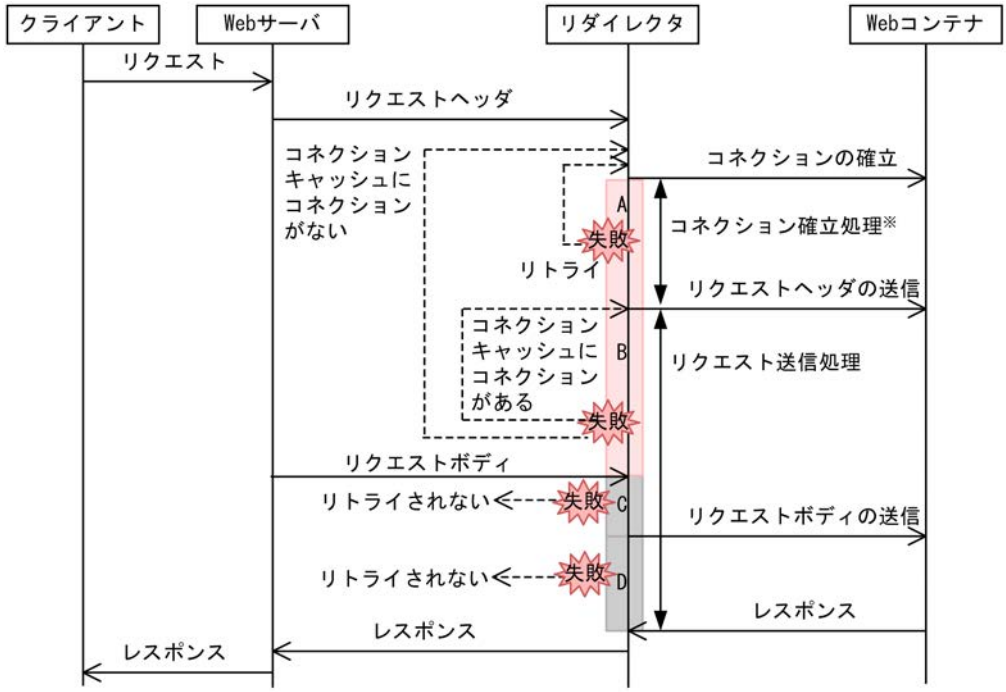
(b)　リクエスト送信処理のリトライ

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

- ネットワークの一時的な障害が発生した場合
- コネクション確立時に，Web コンテナにリクエストが集中している状態で，確立要求が一時的にリスンキューからあふれた場合
- Web コンテナの起動完了を待っている場合

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

図 4-17　Web コンテナへのリクエスト送信時のリトライ処理の流れ



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

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

ポイント

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

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

● 図中 A の部分で処理に失敗した場合のリトライ

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

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

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

● 図中 B の部分で処理に失敗した場合のリトライ

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

1. 設定したタイムアウトの時間が経過すると，リクエスト送信でタイムアウトが発生したことを示すメッセージが出力されます。
2. リクエストヘッダの送信処理に使用されたコネクションをクローズします。
3. 指定した回数だけリクエストヘッダの送信をリトライします。
   なお，このとき，コネクションキャッシュにコネクションがあるかどうかで動作が異なります。
   - コネクションキャッシュにコネクションがある場合
     コネクションキャッシュ中のコネクションを使用して，リクエストヘッダの送信処理からリトライします。
   - コネクションキャッシュにコネクションがない場合
     再度，コネクションの確立を実施してから，リクエストヘッダの送信処理をリトライします。

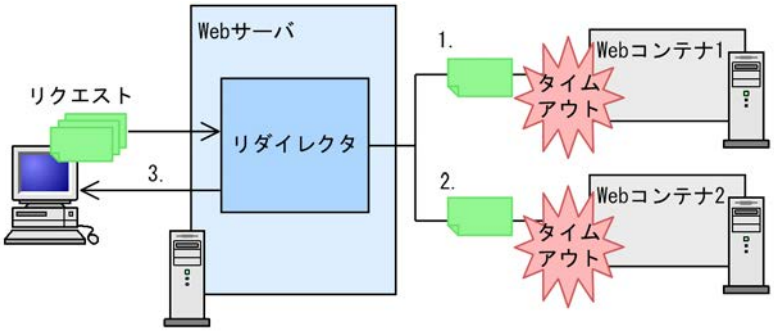
なお，リトライ回数分実施してもリクエストヘッダの送信ができなかった場合，リクエスト送信に失敗したことを示すメッセージが出力され，クライアントにステータスコード 500 が返されます。

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

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

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

図 4-18　ロードバランサを使用してリクエストの振り分けをしている場合のリトライ動作



図のリトライ動作について説明します。

1. Web コンテナ 1 へのリクエスト送信
   リクエストは Web コンテナ 1 に送信されます。Web コンテナ 1 へのコネクションの確立またはリクエストヘッダの送信処理に失敗すると，リトライをします。リトライは 3 回まで実施されます。
2. Web コンテナ 2 へのリクエスト送信

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

3. クライアントへのエラー通知

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

参考

リクエストの転送は，設定されている Web コンテナの数だけ実施されます。

(3)　Web コンテナでのリクエスト受信時

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

検知できる障害

- Web サーバが稼働するホストがダウンした
- Web サーバ－Web コンテナ間のネットワーク障害が発生した
- クライアント－Web サーバ間での処理が終わる前に，Web コンテナ側でタイムアウトが発生した
  これは，Web コンテナが要求したデータサイズ分の読み込みを，クライアント－Web サーバ間で行っているときに，クライアント－Web サーバ間の通信速度が十分ではないなどの理由によって，データの読み込みが終了する前に，Web コンテナで設定した通信タイムアウトが発生したことを意味します。

通信タイムアウトが発生する条件と発生後の動作について次の表に示します。

表 4–13　通信タイムアウトが発生する条件と発生後の動作

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

通信タイムアウトが発生する条件	発生後の動作
• リクエストにボディデータが存在する。 • 読み込み処理に入ったあとで，リダイレクタが稼働するホスト，またはリダイレクターWeb コンテナ間のネットワークに障害が発生した。	• メッセージログに KDJE39188-E のメッセージを出力する。

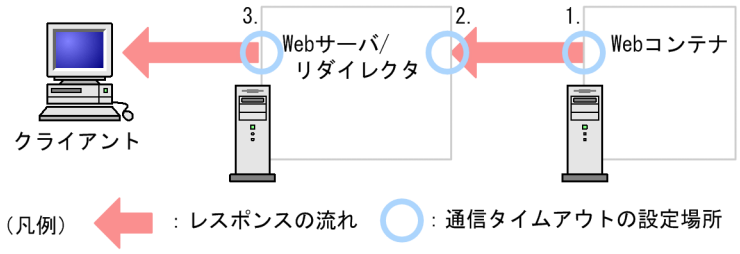
注※

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

4.6.2　レスポンス送受信時の通信タイムアウト

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

図 4-19　レスポンス送信時の通信タイムアウトの設定場所（Web サーバ連携機能を使用する場合）



Web サーバ連携機能を使用する場合，図中の〇印の場所に通信タイムアウトを設定します。通信タイムアウトを設定する場所について説明します。なお，図中の項番は次の説明の項番と対応しています。

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

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

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

2. リダイレクタでのレスポンス受信時（Web コンテナーリダイレクタ）

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

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

3. Web サーバでのレスポンス送信時（Web サーバークライアント）

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

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

(1)　Web コンテナでのレスポンス送信時

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

検知できる障害

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

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

表 4-14　通信タイムアウトが発生する条件と発生後の動作

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

（凡例）－：該当しない

(2)　リダイレクタでのレスポンス受信時

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

検知できる障害

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

Web アプリケーションで発生する障害には次のようなものがあります。

Web アプリケーションで発生する障害

- サーブレット/JSP 処理内の無限ループによって，レスポンスが返されない

- サーブレット/JSP の延長で Enterprise Bean，データベースなどが呼び出され，それによる応答待ちをしている
- Web アプリケーションでデッドロックが発生している
- アクセスピーク時にサーバの処理が追いつかないで滞っている

リダイレクタでの通信タイムアウト後の動作

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

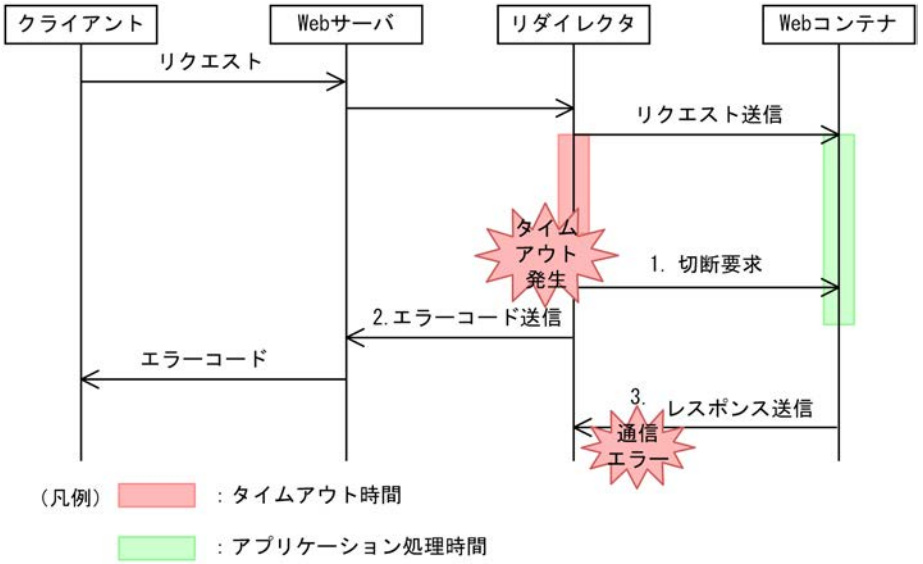
ポイント

アプリケーション処理中にタイムアウトが発生したときの動作

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

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

図 4-20　アプリケーション処理中にタイムアウトが発生したときの動作



図について説明します。

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

(3)　Web サーバでのレスポンス送信時

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

検知できる障害

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

4.6.3　通信タイムアウトの設定

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

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

表 4-15　通信タイムアウト（Web サーバ連携）の設定方法の参照先

設定するタイミング	Smart Composer の使用	
	使用する	使用しない
リクエスト送受信時	4.6.4	4.6.5
レスポンスの送受信時	4.6.6	4.6.7

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

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

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

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1)　Web サーバでのリクエスト受信時の設定

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

(a)　設定方法

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

- **リクエスト受信処理の待ち時間**

  Timeout ディレクティブに，クライアントからのリクエスト受信処理の待ち時間（秒）を設定します。Timeout ディレクティブのデフォルト値は 300 秒です。なお，ここに設定する値は，クライアントへのデータ送信処理に対する通信タイムアウトと共用です。クライアントへのデータ送信処理に対する

通信タイムアウトについては，「4.6.6(3)　Web サーバでのレスポンス送信時の設定」を参照してください。

(b)　設定時の注意

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

(2)　リダイレクタでのリクエスト送信時の設定

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

(a)　設定方法

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

- **コネクション確立の通信タイムアウト**
  論理 Web サーバ（web-server）の<configuration>タグ内に，JkConnectTimeout パラメタで，Web コンテナへのコネクション確立処理の待ち時間（秒）を設定します。JkConnectTimeout パラメタのデフォルト値は 30 秒です。
- **リクエスト送信処理のタイムアウト**
  論理 Web サーバ（web-server）の<configuration>タグ内に，JkSendTimeout パラメタで，Web コンテナへのリクエスト送信処理の待ち時間（秒）を設定します。JkSendTimeout パラメタのデフォルト値は 100 秒です。
- **コネクション確立およびリクエスト送信のリトライ回数**
  論理 Web サーバ（web-server）の<configuration>タグ内に，JkRequestRetryCount パラメタで，Web コンテナへのコネクション確立およびリクエスト送信のリトライ回数を設定します。JkRequestRetryCount パラメタのデフォルト値は 3 回です。

(b)　設定時の注意

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

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

(3)　Web コンテナでのリクエスト受信時の設定

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

(a)　設定方法

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

- リクエスト受信処理のタイムアウト

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

(b)　設定時の注意

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

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

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

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

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

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

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1)　Web サーバでのリクエスト受信時の設定

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

(a)　設定方法

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

- httpsd.conf（HTTP Server の場合）

  Timeout ディレクティブに，クライアントからのリクエスト受信処理の待ち時間（秒）を設定します。Timeout ディレクティブのデフォルト値は 300 秒です。なお，ここに設定する値は，クライアントへのデータ送信処理に対する通信タイムアウトと共用です。クライアントへのデータ送信処理に対する通信タイムアウトについては，「4.6.6(3)　Web サーバでのレスポンス送信時の設定」を参照してください。
- isapi_redirect.conf（Microsoft IIS の場合）

receive_client_timeout キーに，クライアントからのリクエスト受信処理の待ち時間（秒）を設定します。

(b) 設定時の注意

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

(2) リダイレクタでのリクエスト送信時の設定

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

(a) 設定方法

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

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

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

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

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

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

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

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

(b) 設定時の注意

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

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

(3)　Web コンテナでのリクエスト受信時の設定

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

(a)　設定方法

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

- usrconf.properties

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

(b)　設定時の注意

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

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

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

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

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

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

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1)　Web コンテナでのレスポンス送信時の設定

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

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

(2) リダイレクタでのレスポンス受信時の設定

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

(a) 設定方法

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

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

(b) 設定時の注意

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

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

- 通信タイムアウトには，Web アプリケーションの処理に必要な時間より長い時間を設定してください。
  設定したタイムアウトの時間が，Web アプリケーションの処理時間より短い場合，Web アプリケーションは正常に処理されていても，リダイレクタ側ではタイムアウトが発生したと判断して，クライアントにエラーが返されます。
- Web コンテナのピーク時の，同時実行スレッド数の制御による待ち時間を考慮してください。
  同時実行スレッド数の制御によって，Web コンテナのピーク時には，処理待ちリクエストの発生が考えられます。このため，同時実行スレッド数の制御を設定している場合，リクエストの処理時間が延長されることも考慮して，通信タイムアウトを設定しておく必要があります。同時実行スレッド数の制御の設定については，「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 コンテナでのリダイレクタへのデータの送信時に障害が発生した場合，TCP の再送タイマでのタイムアウトによって障害が検知されます。
  なお，UNIX の場合，タイムアウトまでの時間は，OS によって異なります。

(3) Web サーバでのレスポンス送信時の設定

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

(a) 設定方法

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

- データ送信処理の通信タイムアウト
  Timeout ディレクティブに通信タイムアウトを設定してください。なお，Timeout ディレクティブで指定する通信タイムアウトは，リクエスト受信処理に対する通信タイムアウトおよびレスポンス送信処

理に対する通信タイムアウトの両方の時間として指定します。このため，リクエスト受信処理の通信タイムアウトとレスポンス送信処理の通信タイムアウトで異なる時間を設定することはできません。

クライアントからのリクエスト受信処理に対する通信タイムアウトについては，「4.6.4(1)　Web サーバでのリクエスト受信時の設定」を参照してください。

(b)　設定時の注意

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

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

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

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

それぞれの場合での通信タイムアウトの設定方法について説明します。

(1)　Web コンテナでのレスポンス送信時の設定

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

- usrconf.properties

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

(2)　リダイレクタでのレスポンス受信時の設定

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

(a)　設定方法

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

- workers.properties

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

(b)　設定時の注意

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

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

- 通信タイムアウトには，Web アプリケーションの処理に必要な時間より長い時間を設定してください。
  設定したタイムアウトの時間が，Web アプリケーションの処理時間より短い場合，Web アプリケーションは正常に処理されていても，リダイレクタ側ではタイムアウトが発生したと判断して，クライアントにエラーが返されます。
- Web コンテナのピーク時の，同時実行スレッド数の制御による待ち時間を考慮してください。
  同時実行スレッド数の制御によって，Web コンテナのピーク時には，処理待ちリクエストの発生が考えられます。このため，サーバ管理コマンドを使用して同時実行スレッド数の制御を設定している場合，リクエストの処理時間が延長されることも考慮して，通信タイムアウトを設定しておく必要があります。サーバ管理コマンドでの同時実行スレッド数の制御の設定については，「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 コンテナでのリダイレクタへのデータの送信時に障害が発生した場合，TCP の再送タイマでのタイムアウトによって障害が検知されます。
  なお，UNIX の場合，タイムアウトまでの時間は，OS によって異なります。

(3)　Web サーバでのレスポンス送信時の設定

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

(a)　設定方法

次のファイルに，クライアントからのレスポンス待ち時間を設定してください。

- httpsd.conf（HTTP Server の場合）
  Timeout ディレクティブに通信タイムアウトを設定してください。なお，Timeout ディレクティブで指定する通信タイムアウトは，リクエスト受信処理に対する通信タイムアウトおよびレスポンス送信処理に対する通信タイムアウトの両方の時間として指定します。このため，リクエスト受信処理の通信タイムアウトとレスポンス送信処理の通信タイムアウトで異なる時間を設定することはできません。
  クライアントからのリクエスト受信処理に対する通信タイムアウトについては，「4.6.4(1)　Web サーバでのリクエスト受信時の設定」を参照してください。
- MinFileBytesPerSec プロパティ（Microsoft IIS の場合）
  MinFileBytesPerSec プロパティに通信タイムアウトを設定してください。MinFileBytesPerSec プロパティには，Web サーバからクライアントへ送信するレスポンスデータのスループット（バイト/秒）を指定します。レスポンスデータのスループットが MinFileBytesPerSec に設定した値を下回った場合に，通信タイムアウトが発生します。なお，通信タイムアウトのデフォルトは 240 バイト/秒です。
  設定の詳細については，Microsoft IIS のマニュアルを参照してください。

(b)　設定時の注意

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

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

4.7　IP アドレス指定（Web サーバ連携）

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

この節の構成を次の表に示します。

表 4-16　この節の構成（IP アドレス指定（Web サーバ連携））

分類	タイトル	参照先
解説	バインド先アドレス設定機能	4.7.1
設定	実行環境での設定（J2EE サーバの設定）	4.7.2
注意事項	Web サーバ連携での IP アドレス指定をする場合の注意事項	4.7.3

注　「実装」および「運用」について，この機能固有の説明はありません。

4.7.1　バインド先アドレス設定機能

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

バインドする IP アドレスは，J2EE サーバのプロパティをカスタマイズして設定します。J2EE サーバの動作設定のカスタマイズについては，「4.7.2　実行環境での設定（J2EE サーバの設定）」を参照してください。

4.7.2　実行環境での設定（J2EE サーバの設定）

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

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

webserver.connector.ajp13.bind_host
　Web サーバ連携機能を使用する場合の IP アドレスまたはホスト名を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

4.7.3　Web サーバ連携での IP アドレス指定をする場合の注意事項

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

- ホスト名または IP アドレスを設定した場合は，指定された IP アドレスへの接続要求しか受け付けません。IP アドレスを設定する代わりに，ワイルドカードアドレスを指定することで，そのホスト上の任意の IP アドレスへの接続を受け付けることができます。デフォルトでは，ワイルドカードアドレスを使用する設定になっています。なお，ワイルドカードアドレスを使用する場合は，次のことに注意してください。

- 指定されたホスト名が，hosts ファイルまたは DNS などで解決できない場合は，ワイルドカードアドレスを使用してサーバを起動します。
- 指定されたホスト名，または IP アドレスがリモートホストの場合，ワイルドカードアドレスを使用してサーバを起動します。

4.8　エラーページのカスタマイズ（Web サーバ連携）

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

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

この節の構成を次の表に示します。

表 4-17　この節の構成（エラーページのカスタマイズ（Web サーバ連携））

分類	タイトル	参照先
解説	エラーページのカスタマイズの概要	4.8.1
	エラーページのカスタマイズの仕組み	4.8.2
設定	実行環境での設定（Smart Composer 機能を使用する場合）	4.8.3
	実行環境での設定（Smart Composer 機能を使用しない場合）	4.8.4
注意事項	エラーページのカスタマイズの注意事項	4.8.5

注　「実装」および「運用」について，この機能固有の説明はありません。

! 注意事項

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

4.8.1　エラーページのカスタマイズの概要

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

表 4-18　エラー発生場所とエラーページのカスタマイズ方法の対応

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

（凡例）　○：カスタマイズできる　×：カスタマイズできない

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

4.8.2　エラーページのカスタマイズの仕組み

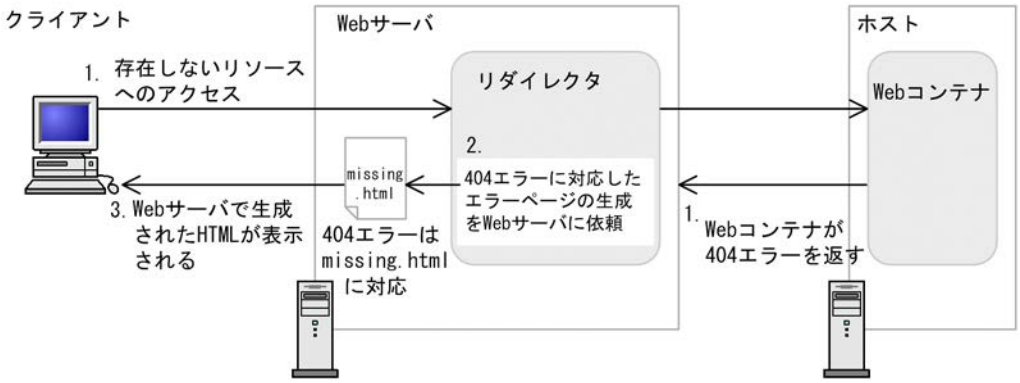
エラーページのカスタマイズの処理の仕組みを，Web コンテナでエラーが発生した場合，およびリダイレクタでエラーが発生した場合のそれぞれについて説明します。

Web コンテナでエラーが発生した場合

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

エラーページのカスタマイズの処理の流れを次の図に示します。

図 4-21　ユーザ作成のエラーページ表示の流れ（Web サーバの機能を使用する場合）



図中の 1.～3.について説明します。

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

リダイレクタでエラーが発生した場合

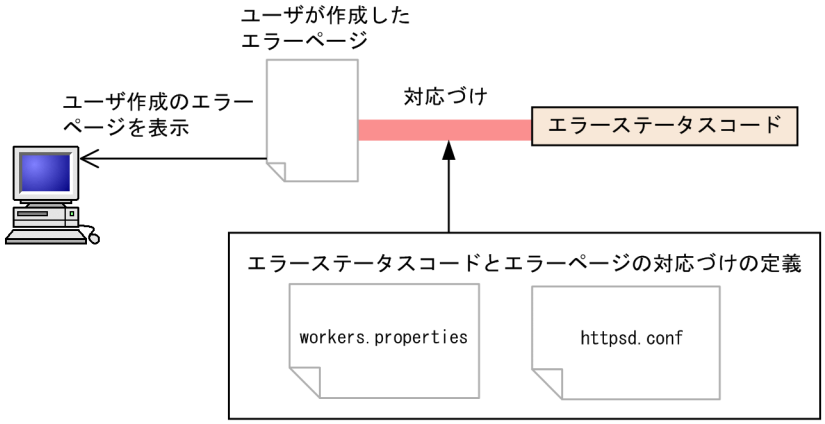
リダイレクタでエラーが発生すると，リダイレクタは設定情報を基にして，発生したエラーに対応するエラーページの生成を Web サーバに依頼します。Web サーバでは，設定情報※を基にエラーステータスコードに対応したユーザ作成のページをクライアントに送信します。これによって，クライアントでは，ユーザが作成したページが表示されます。

注※

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

対応づけの概要を次の図に示します。

図 4-22　エラーステータスコードとエラーページの対応づけ（Web サーバの機能を使用する場合）



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

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

ここでは，エラーページのカスタマイズの設定について説明します。

(1)　設定方法

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

- **簡易構築定義ファイル**

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

  簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

- **httpsd.conf**

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

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

(a)　エラーステータスコード設定時の注意事項

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

- エラーステータスコードは，ワーカ単位で指定します。
- エラーステータスコードを指定できるワーカのタイプは，「ajp13」だけです。ワーカのタイプが「lb」（ラウンドロビンに基づく負荷分散をするときの設定），および「post_size_lb」（POST データサイズに基づく振り分けをするときの設定）の場合，指定した内容は無視されます。

- 指定できるエラーステータスコードは，次の表のとおりです。これ以外のエラーステータスコードにエラーページを対応づけることはできません。

表 4-19　エラーページと対応づけできるエラーステータスコード

エラーステータスコード	説明句
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
422	Unprocessible Entity
423	Locked
424	Failed Dependency
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported
507	Insufficient Storage
510	Not Extended

(b)　ErrorDocument ディレクティブ指定時の注意事項

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

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

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

  (例)

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

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

(2)　設定例

エラーページのカスタマイズの例を次に示します。

簡易構築定義ファイルの例

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

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

httpsd.conf の例

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

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

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

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

ここでは，エラーページのカスタマイズの設定について説明します。

(1)　設定方法

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

- workers.properties

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

  workers.properties（ワーカ定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.5　workers.properties（ワーカ定義ファイル）」を参照してください。

- httpsd.conf

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

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

workers.properties 設定時の注意事項

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

表 4-20　エラーページと対応づけできるエラーステータスコード

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

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

ErrorDocument ディレクティブ指定時の注意事項

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

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

  ErrorDocument 404 http://hostA/error404.jsp

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

(2)　設定例

エラーページのカスタマイズの例を次に示します。

workers.properties の例

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

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

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

httpsd.conf の例

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

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

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

4.8.5　エラーページのカスタマイズの注意事項

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

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

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

4.9　ドメイン名指定でのトップページの表示

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

この節では，ドメイン名指定でのトップページの表示について説明します。

この節の構成を次の表に示します。

表 4-21　この節の構成（ドメイン名指定でのトップページの表示）

分類	タイトル	参照先
解説	ドメイン名指定でのトップページの表示について	4.9.1
設定	実行環境での設定（Smart Composer 機能を使用する場合）	4.9.2
	実行環境での設定（Smart Composer 機能を使用しない場合）	4.9.3

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

4.9.1　ドメイン名指定でのトップページの表示について

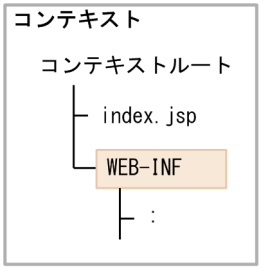
ドメイン名だけでトップページを表示させるには，welcome ファイルを**ルートコンテキスト**に配置する必要があります。ルートコンテキストとは，コンテキストルート※が空文字（コンテキストルートに名称が指定されていない）のコンテキストです。

注※

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

コンテキストとコンテキストルートについて，次の図に示します。

図 4-23　コンテキストとコンテキストルート



ドメイン名だけでトップページを表示させるには，次の設定が必要になります。

- **リダイレクタの設定**

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

- **アプリケーションの設定**

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

● 注意事項

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

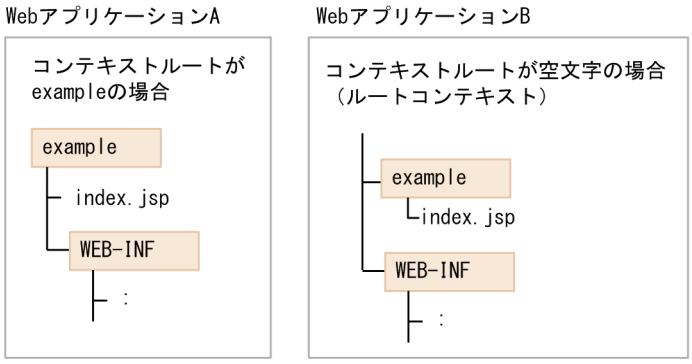
- コンテキストルートとルートコンテキストが同じ階層を持つ場合にアクセスされる階層

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

  例

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

  図 4-24　コンテキストルートとルートコンテキストが同じ階層を持つ場合にアクセスされる階層を持つ例



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

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

- Web アプリケーション内の構成

  URL の先頭に「ejb」や「web」を使用することはできません。

  使用できない例

  http://<ホスト名>:<ポート番号>/ejb/

  http://<ホスト名>:<ポート番号>/web/

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

- メッセージ本文での表示方法

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

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

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

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

(1)　設定方法

ドメイン名指定でトップページを表示するための設定方法を次に示します。

1. ルートコンテキストを設定します。

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

   J2EE アプリケーションのコンテキストルート定義については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「9.11.1　J2EE アプリケーションのコンテキストルート定義」を参照してください。

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

   ルートコンテキストへのリクエストの振り分けは，簡易構築定義ファイルに指定します。

   簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　設定例

ルートコンテキストへのリクエストの振り分けの設定例について説明します。

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

簡易構築定義ファイルの例

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

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

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

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

(1)　設定方法

ドメイン名指定でトップページを表示するための設定方法を次に示します。

1. ルートコンテキストを設定します。

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

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

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

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

mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.4　uriworkermap.properties（Microsoft IIS 用マッピング定義ファイル)」を参照してください。

(2)　設定例

ルートコンテキストへのリクエストの振り分けの設定例について説明します。

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

mod_jk.conf の例（HTTP Server の場合）

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

uriworkermap.properties の例（Microsoft IIS の場合）

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

4.10　Web コンテナへのゲートウェイ情報の通知

この節では，Web コンテナへのゲートウェイ情報の通知について説明します。

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

この節の構成を次の表に示します。

表 4-22　この節の構成（Web コンテナへのゲートウェイ情報の通知）

分類	タイトル	参照先
解説	ゲートウェイ指定機能	4.10.1
設定	実行環境での設定（Smart Composer 機能を使用する場合）	4.10.2
	実行環境での設定（Smart Composer 機能を使用しない場合）	4.10.3
注意事項	Web コンテナにゲートウェイ情報を通知する場合の注意事項	4.10.4

注　「実装」および「運用」について，この機能固有の説明はありません。

4.10.1　ゲートウェイ指定機能

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

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

ゲートウェイ指定機能は次のような場合に使用できます。

- **クライアントと，Web サーバとの間に SSL アクセラレータを配置する場合**

  クライアントから SSL アクセラレータへのアクセスが HTTPS の場合でも，SSL アクセラレータから Web サーバへのアクセスは HTTP となるため，Web コンテナは HTTP によるアクセスであると認識します。このため，welcome ファイルや Form 認証画面へのリダイレクト先 URL のスキームは HTTP となります。

  この場合，ゲートウェイ指定機能を使用して，スキームを常に https と見なすように指定することで，正しくリダイレクトできるようになります。

- **Host ヘッダのないリクエストに対して，リクエストを受けた Web サーバ以外へリダイレクトする必要がある場合**

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

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

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

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

ここは，ゲートウェイ指定機能を使用するための設定について説明します。

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

(1)　設定方法

ゲートウェイ指定機能を使用するための設定方法を次に示します。

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

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

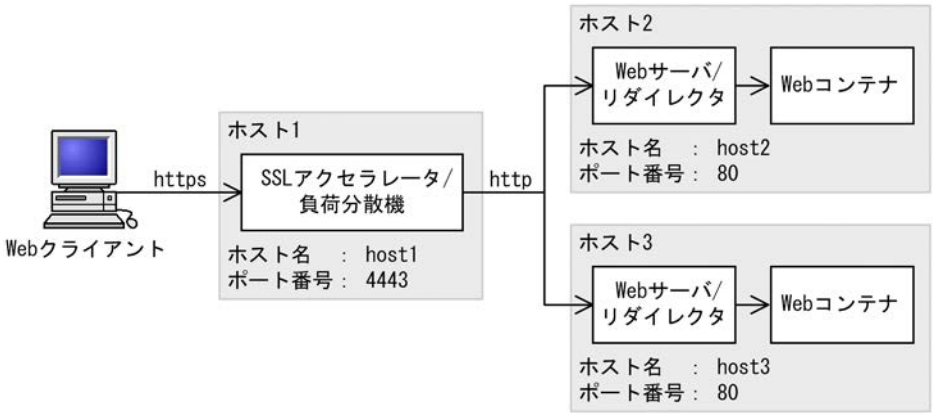
- ホスト名：JkGatewayHost
- ポート番号：JkGatewayPort
- リダイレクト先 URL のスキーム：JkGatewayHttpsScheme

簡易構築定義ファイルおよびパラメタについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　設定例

ゲートウェイ指定機能の設定例を次に示します。

図 4–25　ゲートウェイ指定機能の設定例



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

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

簡易構築定義ファイルの例

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

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

ここでは，ゲートウェイ指定機能を使用するための設定について説明します。

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

(1)　設定方法

ゲートウェイ指定機能を使用するための設定方法を次に示します。

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

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

mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「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。

- mod_jk.conf の場合
  ホスト名：JkGatewayHost キー

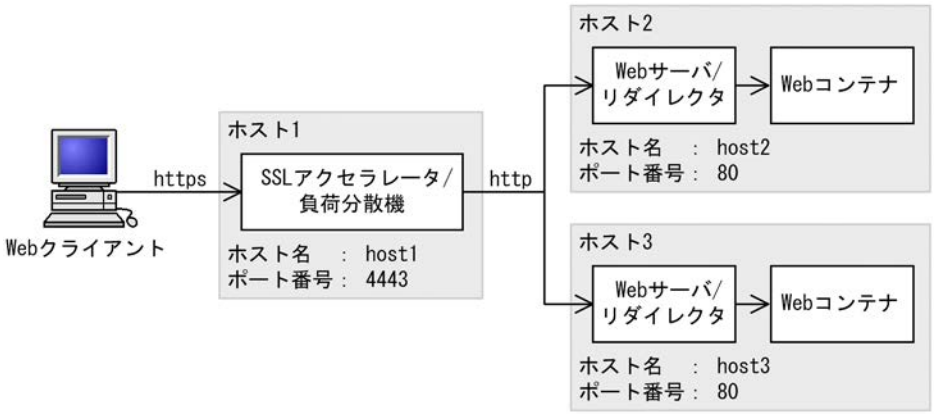
ポート番号：JkGatewayPort キー
リダイレクト先 URL のスキーム：JkGatewayHttpsScheme キー

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

(2)　設定例

ゲートウェイ指定機能の設定例を次に示します。

図 4-26　ゲートウェイ指定機能の設定例



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

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

mod_jk.conf の例（HTTP Server の場合）

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

isapi_redirect.conf の例（Microsoft IIS の場合）

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

4.10.4　Web コンテナにゲートウェイ情報を通知する場合の注意事項

ゲートウェイ指定機能を使用する上での注意事項を次に示します。

- リダイレクト先 URL のホスト名，およびポート番号の指定について

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

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

- サーブレット API の動作について

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

  なお，動作が変わるサーブレット API については，「6.2.2(10)　ゲートウェイ指定機能を使用する場合の注意」を参照してください。

- web.xml の<transport-guarantee>タグについて

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

- Cookie の Secure 属性について

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

- ゲートウェイを通さない Web サーバへの通信

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

5 インプロセス HTTP サーバ

この章では，インプロセス HTTP サーバ機能の設定について説明します。

5.1　この章の構成

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

インプロセス HTTP サーバの機能と参照先を次の表に示します。

表 5-1　インプロセス HTTP サーバの機能と参照先

機能	参照先
インプロセス HTTP サーバの概要	5.2
Web クライアントからの接続数の制御	5.3
リクエスト処理スレッド数の制御	5.4
Web クライアントからの同時接続数の制御によるリクエストの流量制御	5.5
同時実行スレッド数の制御によるリクエストの流量制御	5.6
リダイレクトによるリクエストの振り分け	5.7
Persistent Connection による Web クライアントとの通信制御	5.8
通信タイムアウト（インプロセス HTTP サーバ）	5.9
IP アドレス指定（インプロセス HTTP サーバ）	5.10
アクセスを許可するホストの制限によるアクセス制御	5.11
リクエストデータのサイズの制限によるアクセス制御	5.12
有効な HTTP メソッドの制限によるアクセス制御	5.13
HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ	5.14
エラーページのカスタマイズ（インプロセス HTTP サーバ）	5.15
Web コンテナへのゲートウェイ情報の通知	5.16
ログ・トレースの出力	5.17

なお，アプリケーションサーバで提供するインプロセス HTTP サーバの機能には，Java EE で規定された機能にアプリケーションサーバ独自の機能を拡張したものと，アプリケーションサーバ独自の機能として提供しているものがあります。アプリケーションサーバ独自の機能かどうかについては，「1.2　システムの目的と機能の対応」を参照してください。

5.2　インプロセス HTTP サーバの概要

この節では，インプロセス HTTP サーバの概要について説明します。

この節の構成を次の表に示します。

表 5-2　この節の構成（インプロセス HTTP サーバの概要）

分類	タイトル	参照先
解説	インプロセス HTTP サーバの使用	5.2.1
	インプロセス HTTP サーバで使用できる機能	5.2.2
設定	実行環境での設定（J2EE サーバの設定）	5.2.3

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

5.2.1　インプロセス HTTP サーバの使用

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

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

ただし，HTTP Server および Microsoft IIS と比べて提供機能に差異があるため，機能差異を確認した上で，インプロセス HTTP サーバを使用するかどうか検討してください。また，インプロセス HTTP サーバと Web サーバ連携機能を同時に使用することはできません。システム設計時に，あらかじめどちらの機能を使用するか選択しておく必要があります。選択の指針については，マニュアル「アプリケーションサーバ システム設計ガイド」を参照してください。

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

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

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

- Web クライアントとインプロセス HTTP サーバとの TCP コネクションの接続中に，cjstopsv コマンドを実行して J2EE サーバを停止した場合，Web クライアントがインプロセス HTTP サーバとの TCP コネクションを切断するか，usrconf.properties（J2EE サーバ用ユーザプロパティファイル）の

webserver.connector.inprocess_http.persistent_connection.timeout キーで指定するタイムアウトが発生するまで，J2EE サーバが停止しません。Web クライアントからの TCP コネクション切断や，タイムアウト時間に関係なく J2EE サーバを停止したい場合は，cjstopsv コマンドに「-f」オプションを指定して J2EE サーバを強制停止してください。

インプロセス HTTP サーバの設定は，J2EE サーバのプロパティをカスタマイズして設定します。J2EE サーバの動作設定のカスタマイズについては，「5.2.3　実行環境での設定（J2EE サーバの設定）」を参照してください。

ポイント

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

5.2.2　インプロセス HTTP サーバで使用できる機能

インプロセス HTTP サーバで使用できる機能と参照先を次の表に示します。

表 5-3　インプロセス HTTP サーバで使用できる機能と参照先

機能名		参照先
Web クライアントからの接続数の制御		5.3
Web クライアントからのリクエスト処理スレッド数の制御		5.4
リクエストの流量制御	Web クライアントからの同時接続数の制御	5.5
	同時実行スレッド数の制御※	5.6
リダイレクトによるリクエストの振り分け		5.7
Web クライアントとの通信制御	Persistent Connection による通信制御	5.8
	インプロセス HTTP サーバで設定できる通信タイムアウト	5.9
	インプロセス HTTP サーバで利用する IP アドレス指定※	5.10
Web クライアントからのアクセス制御	アクセスを許可するホストの制限	5.11
	リクエストデータのサイズ制限	5.12
	有効な HTTP メソッドの制限	5.13
Web クライアントへのレスポンスのカスタマイズ	HTTP レスポンスヘッダのカスタマイズ	5.14
	エラーページのカスタマイズ	5.15
Web コンテナへのゲートウェイ情報の通知※		5.16
ログ・トレースの出力		5.17

注※

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

5.2.3　実行環境での設定（J2EE サーバの設定）

ここでは，インプロセス HTTP サーバの設定手順について説明します。

J2EE サーバのプロセス内で提供される Web サーバ機能を使用して，HTTP リクエストを受信するためには，インプロセス HTTP サーバ機能を使用する構成でシステムを構築する必要があります。インプロセス HTTP サーバ機能を使用するシステムの構成や構築手順については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。

設定手順を次に示します。

1. インプロセス HTTP サーバ機能の使用を有効にします。

   インプロセス HTTP サーバの仕様の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内の webserver.connector.inprocess_http.enabled パラメタで「true」を指定します。デフォルトでは「false」が指定されています。簡易構築定義ファイル，および指定するパラメタの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

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

   サーバを稼働するホストの性能やクライアントからのアクセス状況に合わせてリクエスト処理スレッド数を調節することで，インプロセス HTTP サーバのパフォーマンスを向上できます。設定については，「5.3　Web クライアントからの接続数の制御」および「5.4　リクエスト処理スレッド数の制御」を参照してください。

   設定時の留意点を次に示します。

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

3. Web クライアントからのアクセス制御の設定をします。

   クライアントからの接続や送信されるリクエストに対するセキュリティを強化することで，外部からの不正アクセスやサーバへの攻撃を防ぐことができます。設定については，「5.11　アクセスを許可するホストの制限によるアクセス制御」，「5.12　リクエストデータのサイズの制限によるアクセス制御」，および「5.13　有効な HTTP メソッドの制限によるアクセス制御」を参照してください。

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

   インプロセス HTTP サーバで使用できる機能については，「5.2.2　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。

5.3　Web クライアントからの接続数の制御

Web クライアントからの接続数とリクエスト処理スレッド数を制御して，リクエスト処理スレッド数を最適化することによって，J2EE サーバの負荷を一定に抑え，安定した高いスループットを維持できます。リクエスト処理スレッド数の制御については，「5.4　リクエスト処理スレッド数の制御」を参照してください。

この節では，Web クライアントからの接続数の制御について説明します。

この節の構成を次の表に示します。

表 5–4　この節の構成（Web クライアントからの接続数の制御）

分類	タイトル	参照先
解説	Web クライアントからの接続数の制御の概要	5.3.1
設定	実行環境での設定（J2EE サーバの設定）	5.3.2

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

インプロセス HTTP サーバでは，一度に接続できる Web クライアントの数を設定することで，インプロセス HTTP サーバで作成するリクエスト処理スレッド数を制御できます。また，処理を実行していないリクエスト処理スレッドを予備スレッドとして一定数プールしておくことで，リクエスト処理スレッドの追加・削除に掛かる処理を最小限に抑えられます。

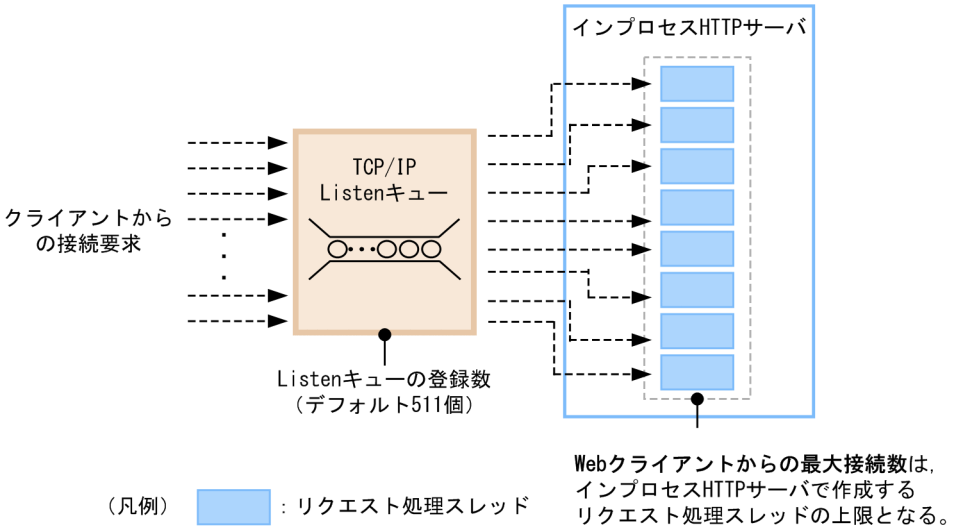
5.3.1　Web クライアントからの接続数の制御の概要

インプロセス HTTP サーバでは，一度に接続する Web クライアントやプロキシサーバの数の最大値を設定して，リクエスト処理スレッド数を制御します。インプロセス HTTP サーバでは Web クライアントからの接続数分のリクエスト処理スレッドを作成するため，Web クライアントからの接続数の最大値は，インプロセス HTTP サーバで作成するリクエスト処理スレッド数の上限となります。

なお，クライアントからの接続要求は，TCP/IP の Listen キューに登録されて，リクエスト処理スレッドに渡されます。接続数の上限を超えたクライアントからの接続要求は，Listen キューに蓄えられます。Listen キューに蓄えられたクライアントからの接続要求が指定した最大値を超えた場合，クライアントはサーバへの接続に失敗します。

Web クライアントからの接続数の制御の概要を次の図に示します。

図 5-1　Web クライアントからの接続数の制御の概要



5.3.2　実行環境での設定（J2EE サーバの設定）

Web クライアントからの接続数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Web クライアントからの接続数の制御の定義について次の表に示します。

表 5-5　簡易構築定義ファイルでの Web クライアントからの接続数の制御の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.max_connections	Web クライアントやプロキシサーバとの接続数の最大値を指定します。インプロセス HTTP サーバでは，Web クライアントからの接続数分のリクエスト処理スレッドを作成するため，ここで指定した値がリクエスト処理スレッド数の上限になります。
webserver.connector.inprocess_http.backlog	Web クライアントからの接続数の上限を超えた HTTP リクエストは，Listen キューに蓄えられます。ここでは，Listen キューの登録数の最大値を指定します。

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

5.4　リクエスト処理スレッド数の制御

Web クライアントからの接続数とリクエスト処理スレッド数を制御して，リクエスト処理スレッド数を最適化することによって，J2EE サーバの負荷を一定に抑え，安定した高いスループットを維持できます。Web クライアントからの接続数の制御については，「5.3　Web クライアントからの接続数の制御」を参照してください。

この節では，リクエスト処理スレッド数の制御について説明します。この節の構成を次の表に示します。

表 5-6　この節の構成（リクエスト処理スレッド数の制御）

分類	タイトル	参照先
解説	リクエスト処理スレッド数の制御の概要	5.4.1
設定	実行環境での設定（J2EE サーバの設定）	5.4.2

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

5.4.1　リクエスト処理スレッド数の制御の概要

インプロセス HTTP サーバでは, 起動時にリクエスト処理スレッドを作成したあと, リクエスト処理スレッドの状態，およびスレッド数を定期的に監視します。インプロセス HTTP サーバにリクエストが集中している場合は，リクエスト処理スレッドを追加して十分な予備スレッドをプールしておき，リクエストが少ない場合は余分にプールしている予備スレッドを削除します。

リクエスト処理スレッド数の制御は次のように実行します。

1. J2EE サーバ起動時に，指定した数分のリクエスト処理スレッドを作成します。
2. J2EE サーバの稼働中はリクエスト処理スレッド数を監視します。
3. 監視のタイミングで，予備スレッド数が指定した最小値より少ない場合は，リクエスト処理スレッドを追加して予備スレッドとしてプールします。また，予備スレッド数が指定した最大値より多い場合は，余分な予備スレッドを削除します。

   なお，J2EE サーバ起動時に作成したスレッド数を維持することもできます。起動時に作成したスレッド数を維持する場合，リクエスト処理スレッドと予備スレッドの総数が起動時に作成したスレッド数以下の場合，予備スレッド数が最大値を超えていても予備スレッドは削除されません。例えば，起動時に作成したスレッド数が 8 で，予備スレッド数の最大値が 5 の場合，リクエスト処理スレッドが 2 で，予備スレッド数が 6 のときは，予備スレッドは削除されません。

次に，リクエスト処理スレッド数の遷移について，例を使用して説明します。

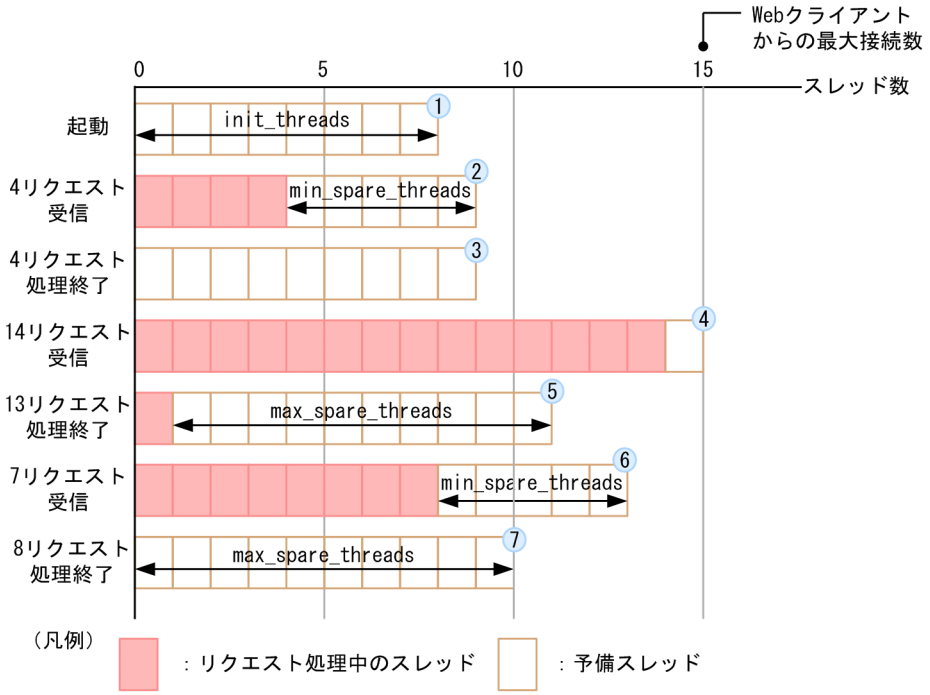
- **遷移例 1**

  次の内容が設定されていることとします。

  - Web クライアントからの最大接続数：15
  - Listen キューの登録数：100
  - J2EE サーバ起動時に作成するリクエスト処理スレッド数：8
  - 予備スレッドの最小値：5
  - 予備スレッドの最大値：10
  - J2EE サーバ起動時に作成したスレッド数の維持：無効

リクエスト処理スレッド数の遷移例を次の図に示します。

図 5-2　リクエスト処理スレッド数の遷移例



図中の項番 1～7 について説明します。

1. J2EE サーバ起動時に，指定した数分（8 スレッド）のリクエスト処理スレッドを作成します。
2. 4 リクエストを受信した時点で，予備スレッドが 4 スレッドとなり，最小値より少ないため 1 スレッド追加します。
3. 4 リクエストの処理が終了した時点で，予備スレッドが 9 スレッドとなり，最大値より少なく，最小値より多いため現状を維持します。
4. 14 リクエストを受信した時点で，予備スレッド数が最小値より少なくなっていますが，Web クライアントからの最大接続数に達しているため，予備スレッドを 1 スレッドだけ追加します。
5. 13 リクエストの処理が終了した時点で，予備スレッド数が最大値を超えているため，4 スレッド削除します。
6. 7 リクエストを受信した時点で，予備スレッド数が最小値より少ないため，2 スレッド追加します。
7. 8 リクエストの処理が終了した時点で，予備スレッド数が最大値を超えているため，3 スレッド削除します。

- **遷移例 2**

予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にすることによって，一度作成したリクエスト処理スレッドを削除しないで使い続けることができます。

予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にした場合のリクエスト処理スレッド数の遷移例について説明します。

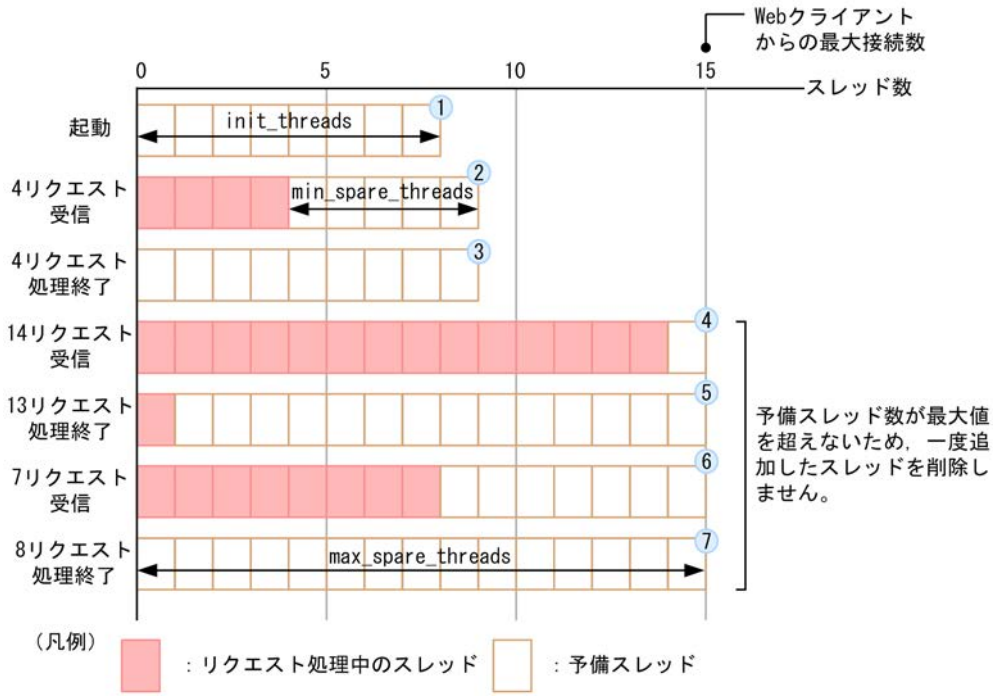
なお，ここでは，次の内容が設定されていることとします。

- Web クライアントからの接続数の最大値：15
- Listen キューの登録数の最大値：100
- J2EE サーバ起動時に作成するリクエスト処理スレッド数：8

- 予備スレッドの最小値：5
- 予備スレッドの最大値：15
- J2EE サーバ起動時に作成したスレッド数の維持：無効

予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にした場合のリクエスト処理スレッド数の遷移例を次の図に示します。

図 5-3　予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にした場合のリクエスト処理スレッド数の遷移例



図中の項番 1～7 について説明します。

1. J2EE サーバ起動時に，指定した数分（8 スレッド）のリクエスト処理スレッドを作成します。
2. 4 リクエストを受信した時点で，予備スレッドが 4 スレッドとなり，最小値より少ないため 1 スレッド追加します。
3. 4 リクエストの処理が終了した時点で，予備スレッドが 9 スレッドとなり，最大値より少なく，最小値より多いため現状を維持します。
4. 14 リクエストを受信した時点で，予備スレッド数が最小値より少なくなっていますが，リクエスト処理スレッドの総数が Web クライアントからの接続数の最大値を超えないように，予備スレッドを 1 スレッドだけ追加します。
5. 13 リクエストの処理が終了した時点で，予備スレッドが 14 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。
6. 7 リクエストを受信した時点で，予備スレッドが 7 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。
7. 8 リクエストの処理が終了した時点で，予備スレッド数が 15 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。

- **設定例 3**

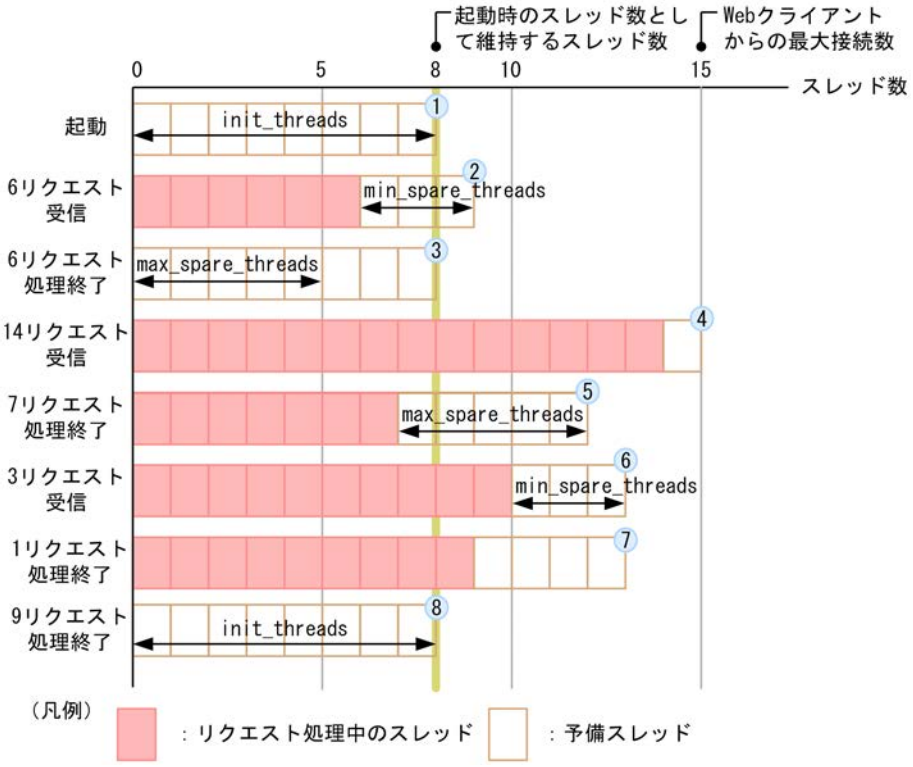
サーバ起動時に作成されたスレッド数を維持する場合のリクエスト処理スレッド数の遷移例について説明します。

なお，ここでは，次の内容が設定されていることとします。

- Web クライアントからの接続数の最大値：15
- Listen キューの登録数の最大値：100
- J2EE サーバ起動時に作成するリクエスト処理スレッド数：8
- 予備スレッドの最小値：3
- 予備スレッドの最大値：5
- J2EE サーバ起動時に作成したスレッド数の維持：有効

サーバ起動時に作成されたスレッド数を維持する場合のリクエスト処理スレッド数の遷移例を次の図に示します。

図 5-4　J2EE サーバ起動時に作成したスレッドを維持する場合のリクエスト処理スレッド数の遷移例



図中の項番 1～8 について説明します。

1. J2EE サーバ起動時に，指定した数分（8 スレッド）のリクエスト処理スレッドを作成します。
2. 6 リクエストを受信した時点で，予備スレッドが 2 スレッドとなり，最小値より少ないため 1 スレッド追加します。
3. 6 リクエストの処理が終了した時点で，予備スレッドが 9 スレッドとなり，最大値を超えていますが，サーバ起動時に作成したスレッド数を維持するため，1 スレッドだけ削除します。
4. 14 リクエストを受信した時点で，予備スレッド数が最小値より少なくなっていますが，リクエスト処理スレッドの総数が Web クライアントからの接続数の最大値を超えないように，予備スレッドを 1 スレッドだけ追加します。
5. 7 リクエストの処理が終了した時点で，予備スレッドが 8 スレッドとなり，最大値を超えたため，3 スレッド削除します。

6. 3 リクエストを受信した時点で，予備スレッドが 2 スレッドとなり，最小値より少ないため，1 スレッド追加します。

7. 1 リクエストの処理が終了した時点で，予備スレッド数が 4 スレッドとなりましたが，最大値より少なく，最小値より多いため現状を維持します。

8. 9 リクエストの処理が終了した時点で，予備スレッド数が 13 スレッドとなり，最大値より多くなっていますが，J2EE サーバ起動時のスレッド数を維持するため，5 スレッドだけ削除します。

5.4.2　実行環境での設定（J2EE サーバの設定）

リクエスト処理スレッド数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリクエスト処理スレッド数の制御の定義について次の表に示します。

表 5-7　簡易構築定義ファイルでのリクエスト処理スレッド数の制御の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.init_threads	J2EE サーバ起動時に作成するリクエスト処理スレッド数を指定します。
webserver.connector.inprocess_http.min_spare_threads	予備スレッドの最小値を指定します。予備スレッド数が指定した最小値より少ない場合は，リクエスト処理スレッドが追加されて予備スレッドとしてプールされます。
webserver.connector.inprocess_http.max_spare_threads	予備スレッドの最大値を指定します。予備スレッド数が指定した最大値より多い場合は，余分な予備スレッドが削除されます。予備スレッドの最大値を Web クライアントからの接続数の最大値と同じ値にすることによって，一度作成したリクエスト処理スレッドを削除しないで使い続けることができます。
webserver.connector.inprocess_http.keep_start_threads	J2EE サーバ起動時に作成したリクエスト処理スレッドを維持するかどうかを指定します。

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

5.5　Web クライアントからの同時接続数の制御によるリクエストの流量制御

この節では，Web クライアントからの同時接続数の制御によるリクエストの流量制御について説明します。

この節の構成を次の表に示します。

表 5-8　この節の構成（Web クライアントからの同時接続数の制御によるリクエストの流量制御）

分類	タイトル	参照先
解説	Web クライアントからの同時接続数の制御	5.5.1
設定	実行環境での設定（J2EE サーバの設定）	5.5.2

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

5.5.1　Web クライアントからの同時接続数の制御

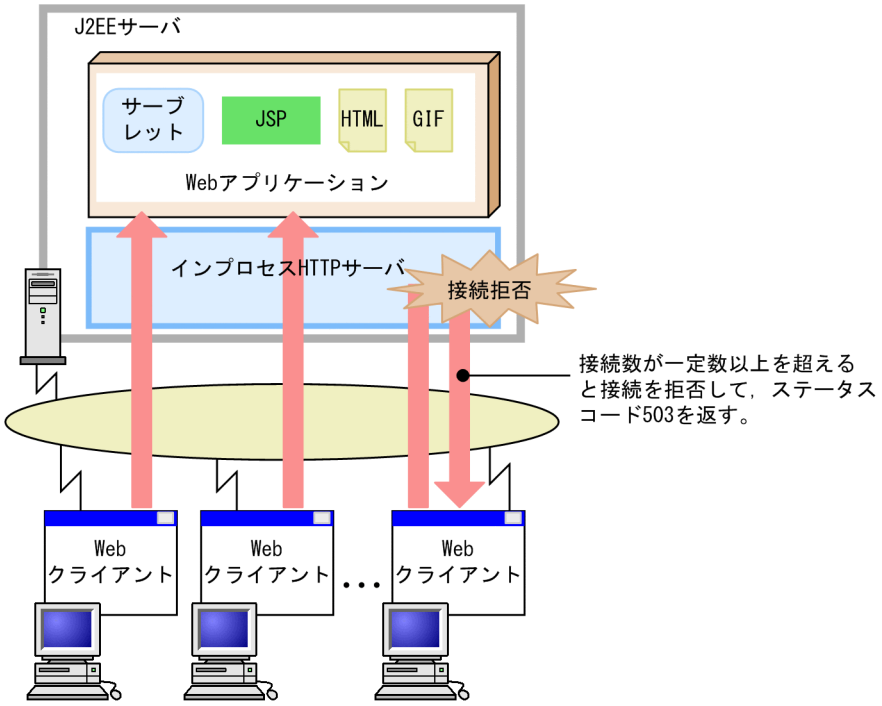
インプロセス HTTP サーバでは，Web クライアントからの最大接続数の設定とあわせて，接続を拒否するリクエスト数を設定することによって，Web クライアントからの同時接続数を制御します。

Web クライアントからの接続数の増加や，J2EE アプリケーションなどの影響で J2EE サーバの負荷が高くなった場合に，Web クライアントからのリクエストの受け付けを拒否して，即座にエラーを返すことで，Web クライアントは即座にレスポンスを受信できます。これによって，J2EE サーバの負荷を一定に抑え，リクエストに対するレスポンスタイムを維持できます。

Web クライアントからの最大接続数から接続を拒否するリクエスト数を引いた値が，Web クライアントからの接続を承諾する数となります。

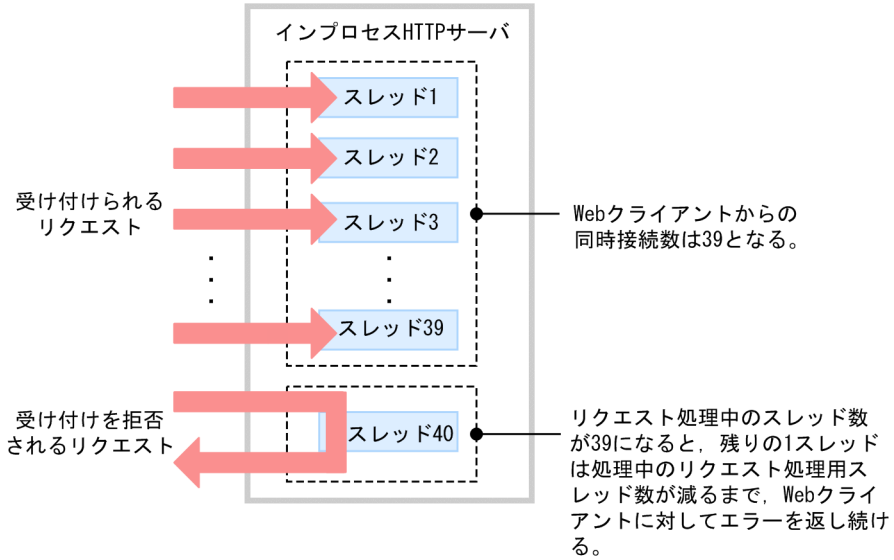
Web クライアントからの同時接続数の制御の概要を，次の図に示します。

図 5-5　Web クライアントからの同時接続数の制御の概要



例えば，Web クライアントからの最大接続数を 40 として，接続を拒否するリクエスト数を 1 とする場合，インプロセス HTTP サーバに接続し同時にリクエストを処理できる Web クライアントの数は 39 となります。リクエスト処理中のスレッド数が 39 になると，残りの 1 スレッドは処理中のリクエスト処理スレッド数が減るまで受信したリクエストに対してエラーを返し続けます。Web クライアントからの同時接続数の制御の例を次の図に示します。

図 5-6　Web クライアントからの同時接続数の制御の例



Web クライアントからの同時接続数の制御によって，接続を拒否したリクエストに対しては，ステータスコード 503 のエラーを Web クライアントに返します。このときクライアントに返すエラーページをカスタマイズすると，レスポンスメッセージのカスタマイズや，ほかのサーバへのリダイレクトができます。

Web クライアントへのレスポンスのカスタマイズ（インプロセス HTTP サーバの場合）の設定については，「5.14　HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ」，および「5.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）」を参照してください。

! 注意事項

フレームを使用したページや，画像を挿入したページを表示する際，Web クライアントから複数のリクエストを受信する場合があります。その場合，Web クライアントからの同時接続数の制御によって表示されるページが部分的にエラーになることがあります。

5.5.2　実行環境での設定（J2EE サーバの設定）

Web クライアントからの同時接続数の制御をする場合，J2EE サーバの設定が必要です。

Web クライアントからの同時接続数の制御の設定方法および設定例について説明します。

(1)　設定方法

Web クライアントからの同時接続数の制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.rejection_threads

接続を拒否するリクエスト数を指定します。

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

! 注意事項

フレームを使用したページや，画像を挿入したページを表示する際，Web クライアントから複数のリクエストを受信する場合があります。その場合，Web クライアントからの同時接続数の制御によって表示されるページが部分的にエラーになることがあります。

(2)　設定例

Web クライアントからの同時接続数の制御の設定例について説明します。

次に，リクエスト処理スレッド数の上限が 40，接続を拒否するリクエスト処理スレッド数が 1 の場合の設定例を示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.max_connections</param-name>
  <param-value>40</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
  <param-value>1</param-value>
</param>
 :
```

この設定例では，接続後に同時にリクエストを処理できる Web クライアントの数は 39 となります。リクエスト処理中のスレッド数が 39 に達すると，残りの 1 スレッドは Web クライアントに対してエラーを返し続けます。

Web クライアントからの同時接続数の制御によって，接続を拒否したリクエストに対しては，ステータスコード 503（Service Unavailable）のエラーを Web クライアントに返します。このときクライアントに返すエラーページをカスタマイズすると，レスポンスメッセージのカスタマイズや，ほかのサーバへのリダ

イレクトができます。それぞれの場合の設定例を次に示します。なお，エラーページのカスタマイズについては，「5.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）」を参照してください。

- レスポンスメッセージをカスタマイズする場合

  Web クライアントからの接続を拒否した場合に，レスポンスボディとして特定のファイルを Web クライアントに返すための設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
  <param-value>3</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
  <param-value>REJECTION_1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.status</param-name>
  <param-value>503</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.file</param-name>
  <param-value>C:/data/busy.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.file.content_type=text/
html; charset</param-name>
  <param-value>ISO-8859-1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.request_url</param-name>
  <param-value>/*</param-value>
</param>
 :
```

  この設定例では，エラーステータスコードに「503」，対応するエラーページのファイルに「C:/data/busy.html」，レスポンスの Content-Type ヘッダに「text/html; charset=ISO-8859-1（Media-Type は text/html で，ISO-8859-1 文字セットを使用）」，URL パターンに「/*」を指定しています。このため，エラーステータスコード「503」が発生した場合には，リクエストの URI に関係なく，「C:/data/busy.html」のファイルの内容をレスポンスとして返します。

- ほかのサーバへリダイレクトする場合

  接続を拒否したリクエストをほかのサーバへリダイレクトする場合の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
  <param-value>3</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
  <param-value>REJECTION_1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.status</param-name>
  <param-value>503</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.redirect_url</param-name>
  <param-value>http://host1/busy.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.request_url</param-name>
  <param-value>/*</param-value>
</param>
<param>
 :
```

  この設定例では，エラーステータスコードに「503」，対応するエラーページのファイルに「http://host1/busy.html」，URL パターンに「/*」を指定しています。このため，エラーステータスコード「503」が発生した場合には，リクエストの URI に関係なく，すべてのリクエストが「http://host1/busy.html」の URL にリダイレクトされます。

5.6　同時実行スレッド数の制御によるリクエストの流量制御

この節では，同時実行スレッド数の制御によるリクエストの流量制御について説明します。

この節の構成を次の表に示します。

表 5-9　この節の構成（同時実行スレッド数の制御によるリクエストの流量制御）

分類	タイトル	参照先
解説	同時実行スレッド数の制御によるリクエストの流量制御の概要	5.6.1
設定	実行環境での設定（J2EE サーバの設定）	5.6.2

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

参考

インプロセス HTTP サーバを使用しない場合（Web サーバ連携機能を使用する場合）との機能差異はありません。機能の解説については，「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。

5.6.1　同時実行スレッド数の制御によるリクエストの流量制御の概要

Web コンテナでは，マルチスレッドでサーブレットのリクエストを処理します。このとき，同時に実行できるスレッド数に上限を設定することで，スラッシングなどによるパフォーマンスの低下を防止できます。また，適切なスレッド数を設定することで，アクセス状況に従ったパフォーマンスのチューニングができます。

5.6.2　実行環境での設定（J2EE サーバの設定）

Web コンテナでの同時実行スレッド数の制御をする場合，J2EE サーバの設定が必要です。

ここでは，Web コンテナでの同時実行スレッド数の制御の設定について説明します。

同時に実行するスレッド数を制御するには，Web コンテナ単位で制御する方法，Web アプリケーション単位で制御する方法，および URL グループ単位で制御する方法の 3 種類あります。

(1)　Web コンテナ単位での制御

Web コンテナ単位で同時実行スレッド数を制御する場合，Web コンテナ全体で同時に実行できるスレッドの最大数を設定します。ここで設定したスレッド数は，Web コンテナにデプロイされているすべての Web アプリケーションで共用するスレッド数となります。

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

webserver.connector.inprocess_http.max_execute_threads

Web コンテナ全体で同時に実行できるスレッドの最大数を指定します。

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

(2)　Web アプリケーション単位での制御

Web アプリケーション単位で同時実行スレッド数を制御する場合，Web コンテナでのスレッド数制御も同時に設定する必要があります。

Web アプリケーション単位での同時実行スレッド数の制御は，簡易構築定義ファイルおよびサーバ管理コマンドで設定します。設定方法は，Web サーバ連携機能を使用する場合と同じです。Web サーバ連携機能を使用する場合の，Web アプリケーション単位での同時実行スレッド数の設定方法については，「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。

(3)　URL グループ単位での制御

URL グループ単位で同時実行スレッド数を制御する場合，Web アプリケーション単位での同時実行スレッド数制御，および Web コンテナでのスレッド数制御も同時に設定する必要があります。

URL グループ単位での同時実行スレッド数の制御は，サーバ管理コマンドで設定します。設定方法は，Web サーバ連携機能を使用する場合と同じです。Web サーバ連携機能を使用する場合の，URL グループ単位での同時実行スレッド数の設定方法については，「2.18　URL グループ単位での同時実行スレッド数の制御」を参照してください。

5.7　リダイレクトによるリクエストの振り分け

この節では，リダイレクトによるリクエストの振り分けについて説明します。

インプロセス HTTP サーバでは，HTTP リクエストに含まれる URL パターンによってリクエストを振り分けます。また，振り分けたリクエストに対するレスポンスをカスタマイズしてクライアントに返すこともできます。

URL パターンによるリクエストの振り分けと，レスポンスのカスタマイズの処理について説明します。また，リダイレクトによるリクエストの振り分けの設定の概要についても説明します。

この節の構成を次の表に示します。

表 5-10　この節の構成（リダイレクトによるリクエストの振り分け）

分類	タイトル	参照先
解説	URL パターンによるリクエストの振り分け	5.7.1
	レスポンスのカスタマイズ	5.7.2
設定	実行環境での設定（J2EE サーバの設定）	5.7.3
注意事項	リダイレクトによるリクエストの振り分けに関する注意事項	5.7.4

注　「実装」および「運用」について，この機能固有の説明はありません。

5.7.1　URL パターンによるリクエストの振り分け

インプロセス HTTP サーバでは，インプロセス HTTP サーバあての HTTP リクエストのうち，特定の URL へのリクエストを，指定した Web コンテナに振り分けて処理させることができます。これによって，システム構成の変更などの理由で Web アプリケーションを別な J2EE サーバに移動する場合に，変更前の URL へのリクエストを，変更後の URL に転送できます。

また，インプロセス HTTP サーバでのリダイレクトによる振り分けでは，特定の Web アプリケーション，および Web アプリケーション内の特定のサーブレット/JSP に対するリクエストを，一時的にほかの Web サーバにリダイレクトする場合にも使用できます。インプロセス HTTP サーバでは，リクエストされたサーブレット/JSP などのリソースが実際にあるかどうかに関係なく，リダイレクトを実行します。リダイレクトは，サーブレット/JSP よりも優先されます。そのため，サーブレット/JSP へのリクエストがリダイレクトされる URL と一致した場合，サーブレット/JSP は実行されません。

5.7.2　レスポンスのカスタマイズ

特定の URL へのリクエストに対して，特定のファイルをレスポンスとして返すようにカスタマイズすることもできます。リダイレクトする URL へのリクエストに対するレスポンスのステータスコードが 300～307 の場合，レスポンスボディを自動生成してクライアントにレスポンスを返します。また，指定したファイルをレスポンスボディとして使用することもできます。ファイルを指定する場合，レスポンスの Content-Type ヘッダもあわせて指定します。

レスポンスボディが自動生成されるステータスコードの詳細，およびリダイレクトによるリクエストの振り分けの設定（インプロセス HTTP サーバの場合）については，「5.7.3　実行環境での設定（J2EE サーバの設定）」を参照してください。

5.7.3　実行環境での設定（J2EE サーバの設定）

ここでは，リダイレクトによるリクエストの振り分けの設定について説明します。

(1)　概要

インプロセス HTTP サーバでは，HTTP リクエストに含まれる URL パターンによってリクエストを振り分けることができます。また，振り分けたリクエストに対するレスポンスをカスタマイズして特定のファイルをクライアントに返すこともできます。リダイレクトする URL へのリクエストに対するレスポンスのステータスコードが 300 番台の場合，レスポンスボディを自動生成してクライアントにレスポンスを返します。また，指定したファイルをレスポンスボディとして使用することもできます。ファイルを指定する場合，レスポンスの Content-Type ヘッダもあわせて指定します。

自動生成されるレスポンスボディを次に示します。

```
<HTML><HEAD>
<TITLE>ステータスコードおよび説明句</TITLE>
</HEAD><BODY>
<H1>ステータスコードおよび説明句</H1>
</BODY></HTML>
```

レスポンスボディが自動生成されるステータスコードおよび説明句を次に示します。

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 305 Use Proxy
- 307 Temporary Redirect

リクエスト処理時にレスポンスボディとして使用するファイルの読み込みに失敗した場合，ステータスコードに 300 番台が指定されていれば，レスポンスボディを自動生成してクライアントに返します。ステータスコードに 200 が指定されていれば，ステータス 500 エラーをクライアントに返します。

(2)　設定方法

リダイレクトによるリクエストの振り分けの定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリダイレクトによるリクエストの振り分けの定義について次の表に示します。

表 5-11　簡易構築定義ファイルでのリダイレクトによるリクエストの振り分けの定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.redirect.list	リダイレクト定義名を指定します。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.request_url	リダイレクトするリクエストの URL を「/」（スラッシュ）で始まる絶対パスで指定します。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.redirect_url	リクエストのリダイレクト先の URL を指定します。なお，ステータスコードに 200 を指定した場合は，URL を指定できません。

指定するパラメタ	設定内容
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.status	リダイレクトを実行する場合のレスポンスのステータスコードを指定します。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.file	特定のファイルをレスポンスとしてクライアントに返す場合にレスポンスボディとして使用するファイルを指定します。なお，ステータスコードに 200 を指定した場合は，使用するファイルを指定する必要があります。
webserver.connector.inprocess_http.redirect.＜リダイレクト定義名＞.file.content_type	特定のファイルをレスポンスとしてクライアントに返す場合にレスポンスボディとして使用するファイルの Content-Type ヘッダを指定します。

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

! 注意事項

- HTTP レスポンスが HTML ファイルの場合，その HTML からほかのファイル（画像ファイルなど）を参照していると，正しくブラウザに表示されないことがあります。
- リダイレクト URL の指定値がリクエスト URL の指定値と一致する場合，クライアントによってはリダイレクトを実行し続けることがあります。
- URL の書き換えによるセッション管理をしている場合，リクエスト URL と同じ Web アプリケーション内へのリダイレクトを実行してもセッションを引き継ぐことはできません。

(3)　設定例

リダイレクトによるリクエストの振り分けの設定例について説明します。

- **設定例 1**

  リダイレクトによるリクエストの振り分けの設定例を次に示します。

```
:
<param>
  <param-name>webserver.connector.inprocess_http.redirect.list</param-name>
  <param-value>REDIRECT_1,REDIRECT_2</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.request_url</param-name>
  <param-value>/index.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.redirect_url</param-name>
  <param-value>http://host1/new_dir/index.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.status</param-name>
  <param-value>302</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.request_url</param-name>
  <param-value>/old_dir/*</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.redirect_url</param-name>
  <param-value>http://host1/new_dir/</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.status</param-name>
  <param-value>301</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.file</param-name>
  <param-value>C:/data/301.html</param-value>
</param>
<param>
```

```
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.file.content_type</param-name>
  <param-value>text/html; charset=ISO-8859-1</param-value>
</param>
 :
```

この設定例では，リダイレクト定義名として，「REDIRECT_1」と「REDIRECT_2」を使用しています。「REDIRECT_1」では，「/index.html」へのリクエストを「http://host1/new_dir/index.html」にステータスコード「302」でリダイレクトします。「REDIRECT_2」では，「/old_dir/」以下へのリクエストを「http://host1/new_dir/」以下にステータス「301」でリダイレクトします。また，レスポンスボディとして「C:/data/301.html」を使用し，Content-Type ヘッダとして「text/html; charset=ISO-8859-1」を使用します。

- 設定例 2

  リクエスト URL としてワイルドカードを使用し，リダイレクト URL に「/」で終わる値を指定した場合，レスポンスの Location ヘッダに設定される値は，「リダイレクト URL に指定した値」＋「実際のリクエスト URL のワイルドカード以降のパス」になります。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.redirect.list</param-name>
  <param-value>REDIRECT_3</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_3.request_url</param-name>
  <param-value>/dir1/*</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_3.redirect_url</param-name>
  <param-value>http://host/dir2/</param-value>
</param>
 :
```

この設定例の場合，実際のリクエスト URL が「/dir1/subdir1/index.html」のときには，Location ヘッダには「http://host/dir2/subdir1/index.html」が設定されます。なお，リクエスト URL の指定でワイルドカードを使用した場合でも，リダイレクト URL が「/」で終わらないときは，Location ヘッダの値は，リダイレクト URL と同じ値になります。

リダイレクトが行われる際，実際のリクエスト URL にクエリ文字列が付加されていた場合，Location ヘッダに設定される値は，リダイレクト URL の指定値にクエリ文字列を付加した値となります。また，リダイレクト URL にはクエリ文字列の付いた値を指定できます。この場合，実際のリクエスト URL にもクエリ文字列が付いているときは，Location ヘッダに設定される値は，リダイレクト URL の指定値の後ろにリクエストのクエリ文字列が付加された値となります。

5.7.4　リダイレクトによるリクエストの振り分けに関する注意事項

リダイレクトによるリクエストの振り分けに関する注意事項を次に示します。

- HTTP レスポンスが HTML ファイルの場合，その HTML からほかのファイル（画像ファイルなど）を参照していると，正しくブラウザに表示されない場合があります。
- リダイレクト URL の指定値とリクエスト URL の指定値が一致すると，クライアントによってはリダイレクトを実行し続けることがあるため注意してください。
- URL の書き換えによるセッション管理をしている場合，Web アプリケーション内へのリダイレクトを実行してもセッションを引き継ぐことはできません。

5.8　Persistent Connection による Web クライアントとの通信制御

この節では，Persistent Connection による Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 5-12　この節の構成（Persistent Connection による Web クライアントとの通信制御）

分類	タイトル	参照先
解説	Persistent Connection による通信制御	5.8.1
設定	実行環境での設定（J2EE サーバの設定）	5.8.2

注　「実装」および「運用」について，この機能固有の説明はありません。

5.8.1　Persistent Connection による通信制御

Persistent Connection は，Web クライアントとインプロセス HTTP サーバ間で確立した TCP コネクションを持続して，複数の HTTP リクエスト間で使用し続けるための機能です。Persistent Connection を使用することによって，Web クライアントと Web サーバ間でのコネクション接続に掛かる時間を短縮し，処理時間の短縮と通信トラフィックの軽減を図れます。

インプロセス HTTP サーバでは，次に示す項目を設定することで，Persistent Connection による通信制御を実現します。

- Persistent Connection 数の上限値

  Persistent Connection 数の上限値を設定することで，一つの TCP コネクションで連続してリクエストを処理できる Web クライアント数を制御します。TCP コネクション数が指定した上限値を超えた場合，リクエスト処理終了後に切断します。これによって，新規リクエストを処理するスレッドが確保でき，リクエスト処理スレッドを特定のクライアントに占有されることを防げます。

- Persistent Connection のリクエスト処理回数の上限値

  Persistent Connection のリクエスト処理回数の最大値を設定することで，同じ Web クライアントから連続してリクエスト要求があった場合の処理を制御します。

  Persistent Connection のリクエスト処理回数が指定した上限値を超えた場合，リクエスト処理終了後にコネクションを切断します。これによってリクエスト処理スレッドを特定のクライアントに占有し続けられることを防げます。

- Persistent Connection のタイムアウト

  Persistent Connection のリクエスト待ち時間にタイムアウトを設定することで，Persistent Connection のリクエスト待ち時間を制御します。指定したタイムアウト時間を超えてリクエスト処理要求がない場合は，TCP コネクションを切断します。これによって，使用されていない状態で TCP コネクションが占有され続けることを防げます。また，Persistent Connection のリクエスト待ち時間に 0 を指定してタイムアウトをしない設定にしている場合でも，リクエスト処理回数の上限値を超えたリクエスト要求があるとコネクションが切断されます。

なお，コネクションを切断された Web クライアントは，接続のリトライを実行して，再度リクエストを送信します。

5.8.2　実行環境での設定（J2EE サーバの設定）

Persistent Connection による通信制御を使用する場合，J2EE サーバの設定が必要です。

ここでは，Persistent Connection による通信制御の設定および設定例について説明します。

(1)　J2EE サーバの設定

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Persistent Connection による通信制御の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Persistent Connection による通信制御の定義について次の表に示します。

表 5-13　簡易構築定義ファイルでの Persistent Connection による通信制御の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.persistent_connection.max_connections	一つの TCP コネクションで連続してリクエストを処理できる Web クライアント数を制御するため，Persistent Connection 数の上限値を指定します。
webserver.connector.inprocess_http.persistent_connection.max_requests	同じ Web クライアントから連続してリクエスト要求があった場合の処理を制御するため，Persistent Connection のリクエスト処理回数の上限値を指定します。
webserver.connector.inprocess_http.persistent_connection.timeout	Persistent Connection のリクエスト待ち時間を制御するため，Persistent Connection のタイムアウトを指定します。

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

(2)　設定例

Persistent Connection による通信制御の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.persistent_connection.max_connections</param-name>
  <param-value>5</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.persistent_connection.max_requests</param-name>
  <param-value>100</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.persistent_connection.timeout</param-name>
  <param-value>15</param-value>
</param>
 :
```

この設定例では，TCP コネクション数が「5」を超えた場合，またはリクエスト処理回数が「100」を超えた場合には，リクエスト処理終了後に TCP コネクションを切断します。また，タイムアウト時間「15」秒を過ぎてもリクエスト処理要求がない場合は，TCP コネクションを切断します。

5.9　通信タイムアウト（インプロセス HTTP サーバ）

この節では，インプロセス HTTP サーバでの通信タイムアウトによる Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 5-14　この節の構成（通信タイムアウト（インプロセス HTTP サーバ））

分類	タイトル	参照先
解説	通信タイムアウトの概要	5.9.1
設定	実行環境での設定（J2EE サーバの設定）	5.9.2

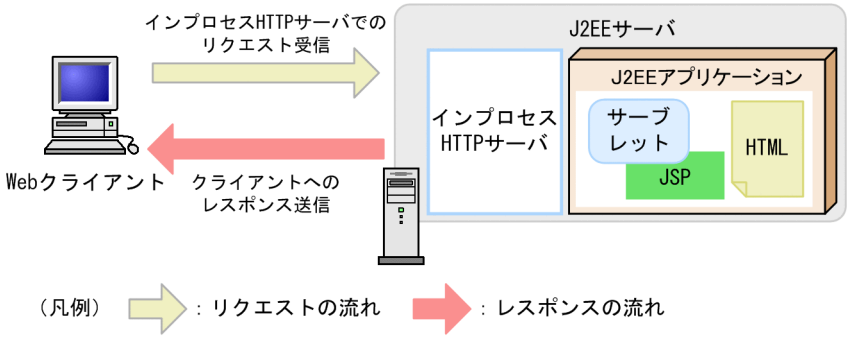
注　「実装」,「運用」および「注意事項」について，この機能固有の説明はありません。

5.9.1　通信タイムアウトの概要

インプロセス HTTP サーバを使用する場合，Web クライアントとインプロセス HTTP サーバ間での，リクエスト受信およびレスポンス送信に，通信タイムアウトを設定できます。ネットワークの障害やアプリケーションの障害などが発生し応答待ち状態になった場合，通信タイムアウトを設定していると，タイムアウトの発生によって障害の発生を検知できます。

インプロセス HTTP サーバを使用する場合，次に示す図中の二つの矢印が示す通信に対してタイムアウトを設定します。

図 5-7　タイムアウトを設定できる通信（インプロセス HTTP サーバを使用する場合）



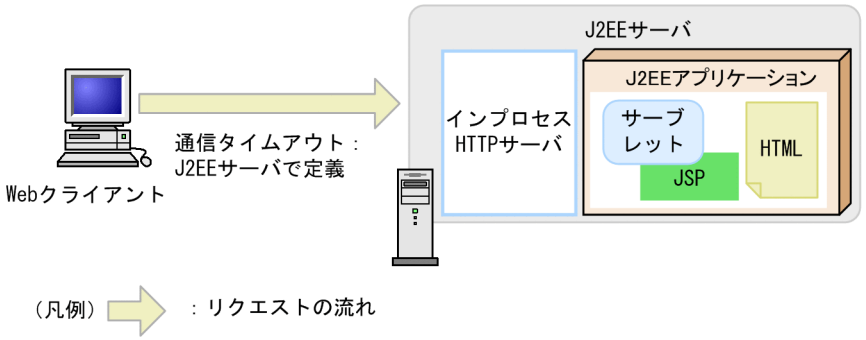
図に示すように，通信タイムアウトはリクエスト受信とレスポンス送信に対して設定します。通信タイムアウトの設定について，リクエストの受信とレスポンスの送信に分けて説明します。

なお，Web クライアントからのリクエスト受信，および Web クライアントへのレスポンス送信でタイムアウトが発生した場合，Web クライアントまたはネットワークで障害が発生したと見なされて，Web クライアントとの接続が切断されるため，レスポンスは返されません。

(1)　リクエスト受信時の通信タイムアウト

リクエスト受信時の通信タイムアウトの設定場所を次の図に示します。

図 5-8　リクエスト受信時の通信タイムアウトの設定場所（インプロセス HTTP サーバを使用する場合）



インプロセス HTTP サーバを使用する場合，Web クライアントとインプロセス HTTP サーバ間の通信に，タイムアウトを設定します。

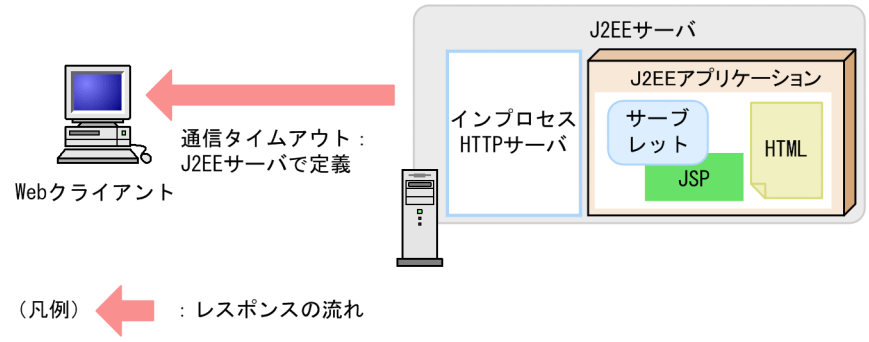
Web クライアントとインプロセス HTTP サーバ間の通信にタイムアウトを設定することによって，クライアント側で次に示す障害が発生したことを検知できます。

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

(2)　レスポンス送信時の通信タイムアウト

レスポンス送信時の通信タイムアウトの設定場所を次の図に示します。

図 5-9　レスポンス送信時の通信タイムアウトの設定場所（インプロセス HTTP サーバを使用する場合）



インプロセス HTTP サーバを使用する場合，インプロセス HTTP サーバと Web クライアント間の通信に，タイムアウトを設定します。

インプロセス HTTP サーバと Web クライアント間の通信にタイムアウトを設定することによって，次に示す障害を検知できます。

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

5.9.2　実行環境での設定（J2EE サーバの設定）

インプロセス HTTP サーバの通信タイムアウトの設定をする場合，J2EE サーバの設定が必要です。

ここでは，インプロセス HTTP サーバの通信タイムアウトの設定および設定例について説明します。

通信タイムアウトは，リクエストの受信時，またはレスポンスの送信時に設定します。リクエスト受信時およびレスポンス送信時の通信タイムアウトの設定についてそれぞれ説明します。

(1)　リクエスト受信時の通信タイムアウトの設定

リクエスト受信時の通信タイムアウトは，クライアント－インプロセス HTTP サーバ間に設定します。

インプロセス HTTP サーバでのリクエスト受信時の通信タイムアウトは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.receive_timeout
　クライアントからのリクエスト受信処理の待ち時間を指定します。

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

(2)　レスポンス送信時の通信タイムアウトの設定

レスポンス送信時の通信タイムアウトは，インプロセス HTTP サーバ－クライアント間に設定します。

インプロセス HTTP サーバでのレスポンス送信時の通信タイムアウトは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.send_timeout
　クライアントへのレスポンス送信処理の待ち時間を指定します。

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

(3)　設定例

インプロセス HTTP サーバの通信タイムアウトの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.receive_timeout</param-name>
  <param-value>300</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.send_timeout</param-name>
  <param-value>600</param-value>
</param>
 :
```

この設定例では，リクエスト受信のタイムアウトに「300」秒，リクエスト送信のタイムアウトに「600」秒を設定しています。

5.10　IP アドレス指定（インプロセス HTTP サーバ）

この節では，インプロセス HTTP サーバでの IP アドレス指定による Web クライアントとの通信制御について説明します。

この節の構成を次の表に示します。

表 5-15　この節の構成（IP アドレス指定（インプロセス HTTP サーバ））

分類	タイトル	参照先
解説	バインド先アドレス設定機能	5.10.1
設定	実行環境での設定（J2EE サーバの設定）	5.10.2
注意事項	インプロセス HTTP サーバでの IP アドレス指定をする場合の注意事項	5.10.3

注　「実装」および「運用」について，この機能固有の説明はありません。

参考

インプロセス HTTP サーバを使用しない場合（Web サーバ連携機能を使用する場合）との機能差異はありません。機能の解説については，「4.7　IP アドレス指定（Web サーバ連携）」を参照してください。

5.10.1　バインド先アドレス設定機能

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

5.10.2　実行環境での設定（J2EE サーバの設定）

インプロセス HTTP サーバの IP アドレスの設定をする場合，J2EE サーバの設定が必要です。

ここでは，インプロセス HTTP サーバの IP アドレスの設定について説明します。

インプロセス HTTP サーバの IP アドレスは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.bind_host

インプロセス HTTP サーバで利用するホスト名または IP アドレスを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

5.10.3　インプロセス HTTP サーバでの IP アドレス指定をする場合の注意事項

インプロセス HTTP サーバでの IP アドレス指定をする場合の注意事項を次に示します。

- ホスト名または IP アドレスを設定した場合は，指定された IP アドレスへの接続要求しか受け付けません。IP アドレスを設定する代わりに，ワイルドカードアドレスを指定することで，そのホスト上の任意の IP アドレスへの接続を受け付けることができます。デフォルトでは，ワイルドカードアドレスを使

用する設定になっています。なお，ワイルドカードアドレスを使用する場合は，次のことに注意してください。

- 指定されたホスト名が，hosts ファイルまたは DNS などで解決できない場合は，ワイルドカードアドレスを使用してサーバを起動します。
- 指定されたホスト名，または IP アドレスがリモートホストの場合，ワイルドカードアドレスを使用してサーバを起動します。

5.11　アクセスを許可するホストの制限によるアクセス制御

J2EE サーバへの不正アクセスを防ぐため，アクセスできるホストを制限できます。デフォルトでは，全ホストからのアクセスが可能な状態になっています。アクセスを許可するホストのホスト名や IP アドレスを設定しておくことで，特定のホストからのアクセスだけを許可して，不正アクセスを防止できます。

この節では，アクセスを許可するホストの制限によるアクセス制御について説明します。

表 5-16　この節の構成（アクセスを許可するホストの制限によるアクセス制御）

分類	タイトル	参照先
解説	アクセスを許可するホストの制限	5.11.1
設定	実行環境での設定（J2EE サーバの設定）	5.11.2

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

5.11.1　アクセスを許可するホストの制限

ホストを制限するには，アクセスを許可するホストの，ホスト名，または IP アドレスを設定します。このとき，ホスト名や IP アドレスの代わりにアスタリスク（*）を指定すると，全ホストからのアクセスを許可する設定になります。アクセスを許可するホストをホスト名で指定した場合，J2EE サーバ起動時にホスト名が解決されます。なお，ローカルホストは明記しなくても常にアクセスが許可されます。通常，外部ネットワークからアクセスされるシステムでは，プロキシサーバの IP アドレスを指定します。

アクセスを許可するホストをホスト名で指定した場合の注意事項を次に示します。

注意事項

- hosts ファイル，または DNS などでホスト名を解決できるようにする必要があります。解決できない場合はデフォルトの設定でサーバが起動されます。
- ホスト名の解決を J2EE サーバの起動時にするため，サーバの起動に時間が掛かることがあります。また，起動後に変更された IP アドレスは反映されないことがあります。

5.11.2　実行環境での設定（J2EE サーバの設定）

アクセスを許可するホストの制限の設定をする場合，J2EE サーバの設定が必要です。

ここでは，アクセスを許可するホストの制限の設定方法と設定例について説明します。

(1)　設定方法

アクセスを許可するホストの制限は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.permitted.hosts

アクセスを許可するホストのホスト名や IP アドレスを指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

! 注意事項

アクセスを許可するホストをホスト名で指定した場合の注意事項を次に示します。

- hosts，または DNS などでホスト名を解決できるようにする必要があります。解決できない場合はデフォルトの設定（全ホストからのアクセスが可能な状態）で J2EE サーバが起動されます。
- J2EE サーバの起動時にホスト名を解決するため，J2EE サーバの起動に時間が掛かることがあります。また，起動後に変更された IP アドレスは反映されないことがあります。

(2)　設定例

アクセスを許可するホストの制限の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.permitted.hosts</param-name>
  <param-value>host1,host2</param-value>
</param>
 :
```

この設定例では，「host1」と「host2」からのアクセスだけを許可し，ほかのホストからのアクセスを許可しません。

5.12　リクエストデータのサイズの制限によるアクセス制御

インプロセス HTTP サーバでは，一定のサイズ以下のリクエストデータだけを受け付けることによって，不正なリクエストデータの受け付けを拒否し，サーバへの負荷を抑え，安定した稼働を維持できます。

この節では，リクエストデータのサイズの制限によるアクセス制御について説明します。

この節の構成を次の表に示します。

表 5-17　この節の構成（リクエストデータのサイズの制限によるアクセス制御）

分類	タイトル	参照先
解説	リクエストデータのサイズの制限	5.12.1
設定	実行環境での設定（J2EE サーバの設定）	5.12.2

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

5.12.1　リクエストデータのサイズの制限

インプロセス HTTP サーバでは，一定のサイズ以下のリクエストデータだけを受け付けることによって，不正なリクエストデータの受け付けを拒否し，サーバへの負荷を抑え，安定した稼働を維持できます。

次に示す項目を設定することで，リクエストデータのサイズ制限によるアクセス制御を実現します。

- リクエストラインの長さの制限

  リクエストラインの長さの上限値を設定することで，アクセスを制御します。リクエストラインの長さには，HTTP メソッド，URI（クエリ文字列を含む），HTTP のバージョン，およびリクエストラインの終わりを示す改行文字（2 バイト）が含まれます。

  受信したリクエストラインの長さが上限値を超えた場合，Web クライアントに対して，ステータスコード 414 のエラーを返します。

- HTTP ヘッダ数の制限

  HTTP リクエストに含まれる HTTP ヘッダ数の上限値を設定することで，アクセスを制御します。

  受信した HTTP リクエストに含まれる HTTP ヘッダ数が上限値を超えた場合，Web クライアントに対してステータスコード 400 のエラーを返します。

- リクエストヘッダサイズの制限

  HTTP リクエストのリクエストヘッダのサイズに上限値を設定することによって，アクセスを制御します。

  受信した HTTP リクエストの HTTP ヘッダのサイズが上限値を超えた場合，Web クライアントに対してステータスコード 400 のエラーを返します。

- リクエストボディサイズの制限

  HTTP リクエストのボディサイズに上限値を設定することによって，アクセスを制御します。インプロセス HTTP サーバでは，リクエストヘッダに含まれる Content-Length ヘッダの値で判断します。

  HTTP リクエストのボディサイズが上限値を超えた場合，Web クライアントに対してステータスコード 413 のエラーを返します。

  なお，チャンク形式でリクエストボディが送信された場合，サーブレット内部では指定された上限値までのデータを読み込みます。上限値を超えるとサーブレットで例外（IOException）が発生しますが，

サーブレットの処理は続行されます。リクエストを送信したクライアントには，指定された上限値までのデータを読み込んだ結果に基づいて，アプリケーションが作成したレスポンスを返します。

ポイント

SSL アクセラレータや負荷分散機などのゲートウェイ機器や，プロキシサーバを配置していて，ゲートウェイ機器やプロキシサーバにリクエストデータサイズの制御機能がある場合は，その制御機能の設定値以下の値を設定する必要があります。

5.12.2　実行環境での設定（J2EE サーバの設定）

リクエストデータのサイズ制限の設定をする場合，J2EE サーバの設定が必要です。

ここでは，リクエストデータのサイズ制限の設定方法と設定例について説明します。

(1)　設定方法

J2EE サーバの設定は，簡易構築定義ファイルで実施します。リクエストデータのサイズ制限の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのリクエストデータのサイズ制限の定義について次の表に示します。

表 5-18　簡易構築定義ファイルでのリクエストデータのサイズ制限の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.limit.max_request_line	リクエストラインの上限値を指定します。
webserver.connector.inprocess_http.limit.max_headers	HTTP リクエストに含まれる HTTP ヘッダ数の上限値を指定します。
webserver.connector.inprocess_http.limit.max_request_header	HTTP リクエストのリクエストヘッダのサイズの上限値を指定します。
webserver.connector.inprocess_http.limit.max_request_body	HTTP リクエストのボディサイズの上限値を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

ポイント

SSL アクセラレータや負荷分散機などのゲートウェイ機器や，プロキシサーバを配置していて，ゲートウェイ機器やプロキシサーバにリクエストデータサイズの制御機能がある場合は，その制御機能の設定値以下の値を設定する必要があります。

(2)　設定例

リクエストデータのサイズ制限の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_request_line</param-name>
  <param-value>1024</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_headers</param-name>
  <param-value>100</param-value>
</param>
```

```
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_request_header</param-name>
  <param-value>8192</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.limit.max_request_body</param-name>
  <param-value>16384</param-value>
</param>
 :
```

5.13　有効な HTTP メソッドの制限によるアクセス制御

この節では，有効な HTTP メソッドの制限によるアクセス制御について説明します。

この節の構成を次の表に示します。

表 5-19　この節の構成（有効な HTTP メソッドの制限によるアクセス制御）

分類	タイトル	参照先
解説	有効な HTTP メソッドの制限	5.13.1
設定	実行環境での設定（J2EE サーバの設定）	5.13.2

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

5.13.1　有効な HTTP メソッドの制限

インプロセス HTTP サーバでは，HTTP リクエストで有効な HTTP メソッドを制限することによって，無効な HTTP メソッドを含むリクエストの受け付けを拒否します。これによって，サーバ上のリソースへの不正アクセスを防止できます。デフォルトでは，DELETE メソッド，HEAD メソッド，GET メソッド，OPTIONS メソッド，POST メソッド，および PUT メソッドの使用を許可しています。

HTTP メソッドを制限するには，有効な HTTP メソッドのメソッド名を設定します。有効な HTTP メソッドとして設定する値は，RFC2616 に規定されている値を使用する必要があります。ただし，メソッド名の文字列にアスタリスク（*）は使用できません。メソッド名の代わりにアスタリスク（*）を指定すると，すべてのメソッドの使用を許可する設定になります。

無効な HTTP メソッドを含むリクエストを受信した場合，Web クライアントに対してステータスコード 405 のエラーを返します。

なお，静的コンテンツに対して OPTIONS メソッドを含むリクエストを送信した場合，レスポンスに含まれる Allow ヘッダには静的コンテンツに対してデフォルトで有効なメソッド（GET メソッド，POST メソッド，TRACE メソッド，および OPTIONS メソッド）から，インプロセス HTTP サーバで無効なメソッドを除いたメソッドが返されます。サーブレット/JSP の場合, Web アプリケーションの実装に依存します。

5.13.2　実行環境での設定（J2EE サーバの設定）

有効な HTTP メソッドの制限の設定をする場合，J2EE サーバの設定が必要です。

ここでは，有効な HTTP メソッドの制限の設定方法と設定例について説明します。

(1)　設定方法

有効な HTTP メソッドの制限は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の <configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.enabled_methods

有効な HTTP メソッドのメソッド名を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

ポイント

静的コンテンツに対して OPTIONS メソッドを含むリクエストを送信した場合，静的コンテンツに対してデフォルトで有効なメソッド（GET メソッド，POST メソッド，TRACE メソッド，および OPTIONS メソッド）から，インプロセス HTTP サーバで無効なメソッドを除いたリクエストが返されます。サーブレット/JSP の場合，Web アプリケーションの実装に依存します。

(2)　設定例

有効な HTTP メソッドの制限の設定例を次に示します。なお，この設定例はデフォルトの設定です。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.enabled_methods</param-name>
  <param-value>GET,HEAD,POST,PUT,DELETE,OPTIONS</param-value>
</param>
 :
```

この設定例では，GET メソッド，HEAD メソッド，POST メソッド，PUT メソッド，DELETE メソッド，および OPTIONS メソッドに対してアクセスを許可し，TRACE メソッドに対してアクセスを拒否します。

5.14　HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ

この節では，HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズについて説明します。

この節の構成を次の表に示します。

表 5-20　この節の構成（HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ）

分類	タイトル	参照先
解説	HTTP レスポンスヘッダのカスタマイズ	5.14.1
設定	実行環境での設定（J2EE サーバの設定）	5.14.2

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

5.14.1　HTTP レスポンスヘッダのカスタマイズ

ここでは，HTTP レスポンスヘッダのカスタマイズについて説明します。

インプロセス HTTP サーバでは，HTTP レスポンスの Server ヘッダに自動設定する情報をカスタマイズできます。デフォルトでは，「CosminexusComponentContainer」が自動設定されます。

Server ヘッダに自動設定する値は，RFC2616 に規定されている値を使用する必要があります。サーブレット/JSP で Server ヘッダを使用する設定にしている場合はその設定が優先されます。

5.14.2　実行環境での設定（J2EE サーバの設定）

HTTP レスポンスヘッダのカスタマイズの設定をする場合，J2EE サーバの設定が必要です。

ここでは，HTTP レスポンスヘッダのカスタマイズの設定方法と設定例について説明します。

(1)　設定方法

HTTP レスポンスヘッダのカスタマイズは，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.connector.inprocess_http.response.header.server

HTTP レスポンスの Server ヘッダに自動設定する文字列を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバリファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　設定例

HTTP レスポンスヘッダのカスタマイズの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.response.header.server</param-name>
  <param-value>GyoumuServer/1.0</param-value>
</param>
 :
```

この設定例では，Server ヘッダの値として「GyoumuServer/1.0」を指定しています。

5.15　エラーページのカスタマイズ（インプロセス HTTP サーバ）

クライアントから，存在しないリソースへのアクセスなどがあると，インプロセス HTTP サーバはエラーステータスコードとエラーページを返し，クライアントにはインプロセス HTTP サーバが生成したエラーページが表示されます。インプロセス HTTP サーバでは，このエラーページの代わりに，ユーザが作成したページをクライアントに表示させることができます。これを，**エラーページのカスタマイズ**と呼びます。

この節では，エラーページを使用した Web クライアントへのレスポンスのカスタマイズについて説明します。

この節の構成を次の表に示します。

表 5-21　この節の構成（エラーページのカスタマイズ（インプロセス HTTP サーバ））

分類	タイトル	参照先
解説	カスタマイズできるエラーページ	5.15.1
実装	エラーページのカスタマイズを実行する場合に必要な実装	5.15.2
設定	実行環境での設定（J2EE サーバの設定）	5.15.3
注意事項	エラーページのカスタマイズに関する注意事項	5.15.4

注　「運用」について，この機能固有の説明はありません。

5.15.1　カスタマイズできるエラーページ

存在しないリソースへのアクセスなどのエラーが発生したときに，エラーステータスコードが表示されたエラーページの代わりにユーザが作成したエラーページをクライアントに表示させることができます。

インプロセス HTTP サーバによるエラーページのカスタマイズを利用することで，特定のステータスコードに対応するエラーページのカスタマイズや，リクエスト URL に対応するエラーページのカスタマイズを Web コンテナ単位にまとめて制御できます。また，次に示すような Web アプリケーションによるエラーページのカスタマイズが実行できない場合でも，エラーページをカスタマイズできます。

- リクエストに対応するコンテキストがない場合（ステータスコード 404）
- 停止処理中のコンテキストでリクエスト処理を実行しようとした場合（ステータスコード 503）
- インプロセス HTTP サーバがエラーステータスコードを返す場合

インプロセス HTTP サーバが返すエラーステータスコードについては，「付録 A.3　インプロセス HTTP サーバが返すエラーステータスコード」を参照してください。

インプロセス HTTP サーバでは，次に示すエラーページのカスタマイズができます。

- **ステータスコードに対応するエラーページのカスタマイズ**
  ステータスコード 400 番台，500 番台に対応するエラーページをカスタマイズできます。
  ステータスコードに対応するエラーページをカスタマイズすることによって，ステータスコードに対応するファイルの送信と，ステータスコードに対応するリダイレクトを実行できます。
  - ステータスコードに対応するファイルの送信

カスタマイズするステータスコードに対して，レスポンスボディとして特定のファイルをクライアントに返すことができます。この場合，レスポンスの Content-Type ヘッダの値を指定します。

なお，リクエスト処理時にファイルの読み込みに失敗した場合，デフォルトのエラーページが使用されます。

- ステータスコードに対応するリダイレクト

  カスタマイズするステータスコードに対して，特定の URL にリダイレクトできます。リダイレクトする場合，レスポンスのステータスコードに 302 を設定して，Location ヘッダにリダイレクト URL を指定します。

  ステータスコードに対応するファイルを送信する場合，リダイレクトは実行できません。

- リクエスト URL に対応するエラーページのカスタマイズ

  リクエスト URL を指定して，特定の URL のエラーページをカスタマイズできます。リクエスト URL を指定した場合，指定した URL と一致するリクエスト処理でエラーが発生した場合だけ，カスタマイズしたエラーページをクライアントに返します。

5.15.2　エラーページのカスタマイズを実行する場合に必要な実装

インプロセス HTTP サーバによるエラーページのカスタマイズを実行する場合，javax.servlet.http.HttpServletResponse インタフェースの sendError メソッドを使用してレスポンスのステータスコードを設定する必要があります。なお，setStatus メソッドを使用した場合（JSP で setStatus メソッドを使用した場合など），インプロセス HTTP サーバによるカスタマイズが実行されないことがあります。ただし，sendError メソッドを使用しても，Web アプリケーションが次に示すどちらかの条件に該当する場合，インプロセス HTTP サーバによるエラーページのカスタマイズは実行されません。

- sendError メソッドの実行時に例外が発生した場合
- Web アプリケーションでエラーページのカスタマイズの設定をしていて，エラー発生時にその設定によるエラーページの実行が正常終了した場合※

注※

エラーページの実行が正常終了した場合とは，次の条件を満たす場合のことです。

- エラーページで catch されない例外が発生していない。
- ステータスコードが 400～599 以外で終了している。

5.15.3　実行環境での設定（J2EE サーバの設定）

エラーページのカスタマイズを実行する場合，J2EE サーバの設定が必要です。

ここでは，エラーページのカスタマイズの設定方法と設定例について説明します。

(1)　設定方法

J2EE サーバの設定は，簡易構築定義ファイルで実施します。エラーページのカスタマイズの定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのエラーページのカスタマイズの定義について次の表に示します。

表 5-22　簡易構築定義ファイルでのエラーページのカスタマイズの定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.error_custom.list	エラーページのカスタマイズ定義名を指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.status	エラーステータスコードに対応づけるエラーページのカスタマイズをする場合に，エラーページをカスタマイズするエラーステータスコードを指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.file	エラーステータスコードに対応するファイルを送信する場合に，レスポンスボディとしてクライアントに返すファイルを指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.file.content_type	webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.file パラメタに指定したファイルをレスポンスボディとしてクライアントに送信する際の Content-Type ヘッダの値を指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.redirect_url	エラーステータスコードに対応するリダイレクトをする場合に，リダイレクト先の URL を指定します。
webserver.connector.inprocess_http.error_custom.＜エラーページカスタマイズ定義名＞.request_url	リクエスト URL に対応づけるエラーページのカスタマイズをする場合に，エラーページのカスタマイズを適用するリクエスト URL を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　設定例

エラーページのカスタマイズの設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
  <param-value>ERR_CUSTOM_1,ERR_CUSTOM_2</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.status</param-name>
  <param-value>404</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.file</param-name>
  <param-value>C:/data/404.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.file.content_type</param-name>
  <param-value>text/html; charset=ISO-8859-1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.status</param-name>
  <param-value>503</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.redirect_url</param-name>
  <param-value>http://host1/503.html</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.request_url</param-name>
  <param-value>/dir1/*</param-value>
</param>
 :
```

この設定例では，エラーページのカスタマイズ定義名として，「ERR_CUSTOM_1」と「ERR_CUSTOM_2」を使用しています。「ERR_CUSTOM_1」では，レスポンスのステータスコードが「404」の場合には，「C:/data/404.html」をクライアントに返します。Content-Type ヘッダの値は，「text/html; charset=ISO-8859-1」を使用します。また，「ERR_CUSTOM_2」では，リクエストが「/dir1/」から始まる URL で，レスポンスのステータスコードが「503」の場合に，「http://host1/503.html」にリダイレクトします。

5.15.4　エラーページのカスタマイズに関する注意事項

インプロセス HTTP サーバによるエラーページのカスタマイズに関する注意事項を次に示します。

- HTTP レスポンスが HTML ファイルの場合，その HTML からほかのファイル（画像ファイルなど）を参照していると，エラーページが正しく表示されない場合があります。
- ブラウザの設定によっては，ステータスコードがエラーを示し，レスポンスボディのサイズが小さい場合，レスポンスボディがブラウザ固定のメッセージに置き換えられて表示されることがあるため注意してください。特にエラーページをカスタマイズしなかった場合に表示されるデフォルトのエラーページは，レスポンスボディのサイズが小さいため注意してください。
- リダイレクト URL の指定値とリクエスト URL の指定値が一致して，リダイレクト後に同じエラーステータスが発生すると，クライアントによってはリダイレクトを実行し続けることがあるため注意してください。
- ステータスコードに対応するリダイレクトを実施する場合，エラー発生時にリクエスト URL に付加されていたクエリ文字列はリダイレクト URL に付加されません。また，URL の書き換えによるセッション管理をしている場合，エラーの発生したページと同じ Web アプリケーション内へのリダイレクトを実施してもセッションを引き継ぐことはできません。

5.16　Web コンテナへのゲートウェイ情報の通知

この節では，Web コンテナへのゲートウェイ情報の通知について説明します。

この節の構成を次の表に示します。

表 5-23　この節の構成（Web コンテナへのゲートウェイ情報の通知）

分類	タイトル	参照先
解説	ゲートウェイ指定機能	5.16.1
設定	実行環境での設定（J2EE サーバの設定）	5.16.2
注意事項	Web コンテナにゲートウェイ情報を通知する場合の注意事項	5.16.3

注　「実装」および「運用」について，この機能固有の説明はありません。

参考

インプロセス HTTP サーバを使用しない場合（Web サーバ連携機能を使用する場合）との機能差異はありません。機能の解説については，「4.10　Web コンテナへのゲートウェイ情報の通知」を参照してください。

5.16.1　ゲートウェイ指定機能

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

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

ゲートウェイ指定機能は次のような場合に使用できます。

- **クライアントとインプロセス HTTP サーバとの間に SSL アクセラレータを配置する場合**

  クライアントから SSL アクセラレータへのアクセスが HTTPS の場合でも，SSL アクセラレータから Web サーバへのアクセスは HTTP となるため，Web コンテナは HTTP によるアクセスであると認識します。このため，welcome ファイルや Form 認証画面へのリダイレクト先 URL のスキームは HTTP となります。

  この場合，ゲートウェイ指定機能を使用して，スキームを常に https と見なすように指定することで，正しくリダイレクトできるようになります。

- **Host ヘッダのないリクエストに対して，リクエストを受けたインプロセス HTTP サーバ以外へリダイレクトする必要がある場合**

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

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

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

5.16.2　実行環境での設定（J2EE サーバの設定）

Web コンテナにゲートウェイ情報を通知するための設定をする場合，J2EE サーバの設定が必要です。

ここでは，Web コンテナにゲートウェイ情報を通知するための設定方法と設定例について説明します。

(1)　設定方法

J2EE サーバの設定は，簡易構築定義ファイルで実施します。Web コンテナにゲートウェイ情報を通知するための設定の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでの Web コンテナにゲートウェイ情報を通知するための設定の定義について次の表に示します。

表 5-24　簡易構築定義ファイルでの Web コンテナにゲートウェイ情報を通知するための設定の定義

指定するパラメタ	設定内容
webserver.connector.inprocess_http.gateway.https_scheme	リダイレクト先 URL のスキームを指定します。
webserver.connector.inprocess_http.gateway.host	ゲートウェイのホスト名を指定します。
webserver.connector.inprocess_http.gateway.port	ゲートウェイのポート番号を指定します。

簡易構築定義ファイル，および指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

(2)　設定例

ゲートウェイ指定機能の設定例を次に示します。

```
 :
<param>
  <param-name>webserver.connector.inprocess_http.gateway.host</param-name>
  <param-value>host1</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.gateway.port</param-name>
  <param-value>4443</param-value>
</param>
<param>
  <param-name>webserver.connector.inprocess_http.gateway.https_scheme</param-name>
  <param-value>true</param-value>
</param>
 :
```

この設定例では，ゲートウェイ指定機能を使用して，スキームを常に HTTPS と見なすように設定しています。

5.16.3　Web コンテナにゲートウェイ情報を通知する場合の注意事項

ゲートウェイ指定機能を使用する上での注意事項を次に示します。

- リダイレクト先 URL のホスト名，およびポート番号の指定について

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

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

- サーブレット API の動作について

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

  なお，動作が変わるサーブレット API については，「6.2.2(10)　ゲートウェイ指定機能を使用する場合の注意」を参照してください。

- web.xml の<transport-guarantee>タグについて

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

- Cookie の Secure 属性について

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

5.17　ログ・トレースの出力

この節では，インプロセス HTTP サーバが出力するログ・トレースについて説明します。

この節の構成を次の表に示します。

表 5-25　この節の構成（ログ・トレースの出力）

分類	タイトル	参照先
解説	インプロセス HTTP サーバが出力するログ・トレース	5.17.1
設定	インプロセス HTTP サーバのアクセスログのカスタマイズ	5.17.2

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

5.17.1　インプロセス HTTP サーバが出力するログ・トレース

インプロセス HTTP サーバでは，アプリケーション開発のサポート，運用時の性能解析，および障害発生時のトラブルシュートのために，次の表に示すログおよびトレースを出力します。

表 5-26　インプロセス HTTP サーバが出力するログ・トレース

ログ・トレースの種類	説明
アクセスログ	Web クライアントからのリクエスト，およびレスポンスの処理結果を出力します。Web クライアントとの通信の解析に使用します。 アクセスログを解析することによって，Web クライアントからリクエストされたファイルや，インプロセス HTTP サーバの性能情報，およびセッショントラッキング情報などを分析できます。
性能解析トレース	インプロセス HTTP サーバでリクエストを送受信するときの性能解析情報や，障害発生時のトラブルシュートのための情報を出力します。 性能解析トレースは，CSV 形式などに変換して，ほかの J2EE サーバの各機能が出力する性能解析情報とあわせて，システム全体のボトルネックの解析などに使用できます。性能解析トレースの詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.6　性能解析トレース」を参照してください。
スレッドトレース	保守用のトレースです。
通信トレース	保守用のトレースです。

5.17.2　インプロセス HTTP サーバのアクセスログのカスタマイズ

インプロセス HTTP サーバでは，アプリケーション開発のサポート，運用時の性能解析，および障害発生時のトラブルシュートのために，アクセスログ，性能解析トレース，スレッドトレース，および通信トレースを出力します。これらのファイルは，ファイル面数やファイルサイズなどを変更できます。また，アクセスログでは，ログの出力形式をカスタマイズできます。

ここでは，インプロセス HTTP サーバのアクセスログの出力形式のカスタマイズについて説明します。インプロセス HTTP サーバのアクセスログおよびトレースファイルのファイル面数やファイルサイズなどの変更については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.11　インプロセス HTTP サーバのログ取得の設定」を参照してください。

(1)　カスタマイズの手順

アクセスログの出力形式のカスタマイズの手順を次に示します。

1. アクセスログのフォーマット名を定義します。

   フォーマットを新規作成する場合には，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.logger.access_log.format_list パラメタで，新規に作成するフォーマット名を追加します。

   設定例

   ```
   :
   <param>
     <param-name>webserver.logger.access_log.format_list</param-name>
     <param-value>formatA</param-value>
   </param>
   :
   ```

   フォーマットを新規作成する場合には，デフォルトで提供されているアクセスログのフォーマットを参考にしてください。フォーマットについては，「(2)　アクセスログのフォーマット」を参照してください。

2. アクセスログの出力形式を定義します。

   簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.logger.access_log.<フォーマット名>パラメタで，<フォーマット名>で指定したフォーマットでのアクセスログの出力形式を定義します。出力形式の定義では，アクセスログのフォーマットの引数を指定します。

   設定例

   ```
   :
   <param>
     <param-name>webserver.logger.access_log.formatA</param-name>
     <param-value>%h %u %t &quot;%r&quot; %&gt;s HostHeader=&quot;%{host}i&quot;</param-value>
   </param>
   :
   ```

   指定できるフォーマットの引数については，「(3)　アクセスログのフォーマットの引数」を参照してください。

3. アクセスログの出力に使用するフォーマットを指定します。

   簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.logger.access_log.inprocess_http.usage_format パラメタで，アクセスログの出力に使用するフォーマット名を指定します。ここで指定したフォーマットで，アクセスログが出力されるようになります。

   設定例

   ```
   :
   <param>
     <param-name>webserver.logger.access_log.inprocess_http.usage_format</param-name>
     <param-value>formatA</param-value>
   </param>
   :
   ```

(2)　アクセスログのフォーマット

アプリケーションサーバでは，インプロセス HTTP サーバのアクセスログのフォーマットとして，common（デフォルトフォーマット）と combined（拡張フォーマット）という 2 種類のフォーマットを提供しています。フォーマットを新規作成する場合には，これらのフォーマットを参考にしてください。

(a)　出力形式

アクセスログの出力形式について説明します。なお，△は，半角スペースを表します。また，ここでは，表記の都合上，複数行にわたっていますが，実際には 1 行で出力されます。

デフォルトフォーマットの出力形式を次に示します。

```
Webクライアントのホスト名またはIPアドレス△リモートログ名△認証ユーザ名
△Webクライアントからのリクエストの処理を開始した時刻△リクエストライン
△最終ステータスコード△HTTPヘッダを除く送信バイト数
```

拡張フォーマットの出力形式を次に示します。

```
Webクライアントのホスト名またはIPアドレス△リモートログ名△認証ユーザ名
△Webクライアントからのリクエストの処理を開始した時刻△リクエストライン
△最終ステータスコード△HTTPヘッダを除く送信バイト数
△"Refererヘッダの内容"△"User-Agentヘッダの内容"
```

下線部分が，デフォルトフォーマットと拡張フォーマットの差異です。拡張フォーマットでは，デフォルトフォーマットの出力内容に加えて，「Referer ヘッダの内容」と「User-Agent ヘッダの内容」が出力されます。

(b)　出力例

デフォルトフォーマットでのアクセスログの出力例を次に示します。

```
10.20.30.40 - user [20/Dec/2004:15:45:01 +0900] "GET /index.html HTTP/1.1" 200 8358
10.20.30.40 - user [20/Dec/2004:15:45:01 +0900] "GET /left.html HTTP/1.1" 200 2358
10.20.30.40 - user [20/Dec/2004:15:45:01 +0900] "GET /right.html HTTP/1.1" 200 4358
```

拡張フォーマットでのアクセスログの出力例を次に示します。

```
10.20.30.40 - - [18/Jan/2005:13:06:10 +0900] "GET / HTTP/1.0" 200 38 "-" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1)"
10.20.30.40 - - [18/Jan/2005:13:06:25 +0900] "GET /demo/ HTTP/1.0" 500 684 "-" "Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1)"
```

(3)　アクセスログのフォーマットの引数

フォーマットの出力形式を定義するときに指定する，アクセスログのフォーマットの引数を次の表に示します。

表 5-27　アクセスログのフォーマットの引数の一覧

フォーマットの引数	出力内容	出力例
%%	%記号	%
%a	Web クライアントの IP アドレス	10.20.30.40
%A	J2EE サーバの IP アドレス	10.20.30.100
%b	HTTP ヘッダを除く送信バイト数 （0 バイトの場合は「-」となる）	2048
%B	HTTP ヘッダを除く送信バイト数 （0 バイトの場合は「0」となる）	1024
%h	Web クライアントのホスト名または IP アドレス	10.20.30.40

フォーマットの引数	出力内容	出力例
%h	（ホスト名を取得できない場合は IP アドレスとなる）	10.20.30.40
%H	リクエストプロトコル	HTTP/1.1
%l	リモートログ名※1 （常に「-」になる）	-
%m	リクエストメソッド	GET
%p	Web クライアントからのリクエストを受け付けたポート番号	80
%q	クエリ文字列 （「?」から始まる。クエリ文字列がない場合は空文字となる）	?id=100&page=15
%r	リクエストライン	GET /index.html HTTP/1.1
%>s	最終ステータスコード （内部リダイレクトされた値は出力しない）	200
%S※2	ユーザのセッション ID （セッション ID がない場合は「-」となる）	00455AFE4DA4E7B7789F247B8FE5D605
%t	Web クライアントからのリクエストの処理を開始した時刻 （単位：秒，出力形式：dd/MMM/YYYY:HH:mm:ss Z）	[18/Jan/2005:13:06:10 +0900]
%T	Web クライアントのリクエストの処理に掛かった時間 （単位：秒）	2
%d	Web クライアントからのリクエストの処理を開始した時刻 （単位：ミリ秒，出力形式：dd/MMM/YYYY:HH:mm:ss.nnn Z（nnn はミリ秒））	[18/Jan/2005:13:06:10.152 +0900]
%D	Web クライアントのリクエストの処理に掛かった時間 （単位：ミリ秒）	2000
%u	Basic 認証ユーザ名，Form 認証ユーザ名 （認証ユーザ名がない場合は「-」となる）	user
%U	リクエストファイルパス	/index.html
%v	J2EE サーバのローカルホスト名	server
%{foo}i※3	リクエストヘッダ foo の内容 （foo ヘッダが存在しない場合は「-」となる）	%{Host}i の場合 www.example.com:8888
%{foo}c	Web クライアントが送信した Cookie 情報で Cookie の名前が foo の内容 （Cookie の名前に foo がない場合は「-」となる）	%{JSESSIONID}c の場合 00455AFE4DA4E7B7789F247B8FE5D605
%{foo}o※3	レスポンスヘッダ foo の内容 （foo ヘッダが存在しない場合は「-」となる）	%{Server}o の場合 CosminexusComponentContainer

注※1

リモートログ名は，RFC 1413 で規定されている Identification プロトコルで取得できる Web クライアント側のユーザ名です。

注※2

%S で表示されるセッション ID は，Cookie 名「JSESSIONID」の値です。メモリセッションフェイルオーバ機能でのグローバルセッション ID とは異なります。グローバルセッション ID を出力させたい場合には，%{GSESSIONID}c を指定してください。GIDCookieName を変更した場合は，変更した GIDCookieName の値を指定してください。

注※3

一度の HTTP リクエストまたは HTTP レスポンスで同じヘッダ名を複数回送信する場合があります。この場合，最初に読み込んだヘッダの内容を出力します。

! 注意事項

フォーマットの引数の指定に誤りがある場合には，デフォルトフォーマットが使用されます。デフォルトフォーマットが使用される例を次に示します。

- フォーマットの引数の一覧にない文字列（例：%G など）を指定した場合
- リクエストヘッダの内容，レスポンスヘッダの内容，Cookie 名で 0 文字（%{}i など）を指定した場合

参考

デフォルトフォーマットと拡張フォーマットをフォーマットの引数で記述すると，次のような形式になります。

- デフォルトフォーマット
  %h %l %u %t "%r" %>s %b
- 拡張フォーマット
  %h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-Agent}i"

6 サーブレットおよび JSP の実装

この章では，サーブレットおよびJSP を実装するときの注意事項について説明します。

6.1　Servlet 仕様および JSP 仕様で追加，変更された機能のサポート範囲

ここでは，Servlet 仕様および JSP 仕様で追加，変更された機能の概要と，アプリケーションサーバでのサポート範囲を示します。

各バージョンの Servlet 仕様および JSP 仕様で追加，変更された機能をアプリケーションサーバで使用する場合の動作や注意事項は，6.2.3 以降を参照してください。

Servlet 3.0 で追加，変更された機能の概要とサポート範囲を次の表に示します。

表 6-1　Servlet 3.0 の機能の概要とサポート範囲

項番	機能名	機能概要	サポート
1	web.xml (Servlet 3.0) と新規アノテーション	Servlet 3.0 に対応した web.xml を使用できます。	○
		アノテーションでサーブレットを定義できます（web.xml を省略できます）。	○
		web-fragment.xml を使用できます。	×※1
2	動的サーブレット定義	API でサーブレット，フィルタ，またはリスナを定義できます。	○
3	ファイルアップロード	content-type が multipart/form-data のリクエストを処理できます。	○
4	静的リソースの配置	JAR ファイルの META-INF/resources に静的リソースや JSP を配置できます。	○
5	セキュリティエンハンス	• アノテーションでセキュリティ設定ができます。 • 認証用の API が使えます。	○
6	非同期サーブレット	リクエストを受け付けたスレッドとは別のスレッドで，リクエスト処理やレスポンス生成ができます。	×※2
7	Servlet 仕様のその他の変更	Cookie に HttpOnly 属性を付けられます。	○
		HTTP セッションのセッション ID を示す HTTP Cookie の名前を変更できます。	○
		HTTP ダイジェスト認証が使えます。	×※3
		ServletRequest の属性として SSL Session ID が取得できます。	×※4
8	JSP 仕様の変更	デフォルトのコンテンツタイプやバッファサイズなどを web.xml で指定できます。	×※5
9	EL パラメタ付きメソッド	パラメタ付きメソッドを呼び出せます。	○※6
10	API のエンハンス	新規に追加または変更となった API が使用できます。	△※7

（凡例）
○：サポートする　×：サポートしない　△：一部サポートする

注※1

web-fragment.xml が Web アプリケーションに含まれていた場合，無視されます。

注※2

非同期サーブレットに関する DD およびアノテーションは無視されます。また，非同期サーブレットに関する API が呼び出された場合は，例外が発生します。

注※3

DD でダイジェスト認証が設定された場合，動作は保証されません。

注※4

ServletRequest クラスで，「javax.servlet.request.ssl_session_id」を引数に指定して getAttribute メソッドを呼び出すと，常に null が返ります。

注※5

JSP 2.2 に対応した web.xml は読み込めますが，JSP 2.2 で追加されたタグについては無視されます。

Web アプリケーションのバージョンが 3.0 の場合でも JSP が準拠する JSP のバージョンは 2.1 となります。

注※6

JSP 2.1 仕様または 2.0 仕様の EL の場合，JSP 2.2 の EL 式は使用できません。JSP 2.2 の EL の API を使用した場合，チェックされないため動作は保証されません。

注※7

サポートする機能の API は使用できますが，サポートしない機能の API は使用できません。Servlet 3.0 の API については，「6.2.3(9)　API について」を参照してください。

6.2　サーブレットおよび JSP の実装時の注意事項

この節では，サーブレットおよび JSP の実装時の注意事項について説明します。

この節の構成を次の表に示します。

表 6-2　この節の構成（サーブレットおよび JSP の実装時の注意事項）

タイトル	参照先
サーブレットおよび JSP 実装時共通の注意事項	6.2.1
サーブレット実装時の注意事項	6.2.2
Servlet 3.0 仕様で追加，変更された仕様についての注意事項	6.2.3
Servlet 2.5 仕様で追加，変更された仕様についての注意事項	6.2.4
Servlet 2.4 仕様で追加，変更された仕様についての注意事項	6.2.5
JSP 実装時の注意事項	6.2.6
JSP 2.1 仕様で追加，変更された仕様についての注意事項	6.2.7
JSP 2.0 仕様で追加，変更された仕様についての注意事項	6.2.8
JSP 1.2 仕様の JSP 実装時の注意事項	6.2.9
EL2.2 仕様で追加，変更された仕様についての注意事項	6.2.10
既存の Web アプリケーションを Servlet 3.0 仕様にバージョンアップする場合の留意点	6.2.11
既存の Web アプリケーションを Servlet 2.5 仕様にバージョンアップする場合の留意点	6.2.12
前バージョンから 09-50 へ移行する場合の Web アプリケーションに関する注意事項	6.2.13
既存の Web アプリケーションを Servlet 2.4 仕様にバージョンアップする場合の留意点	6.2.14
サーブレットでのアノテーションの使用	6.2.15
JavaVM のメソッドサイズ制限についての注意事項	6.2.16

6.2.1　サーブレットおよび JSP 実装時共通の注意事項

アプリケーションサーバ上で動作するアプリケーションのプログラムとして，サーブレットおよび JSP を実装するときの共通の注意事項を示します。

(1)　Web アプリケーションの動作の前提となる J2EE アプリケーションのバージョン

Web アプリケーションの動作の前提となる J2EE アプリケーションが準拠する J2EE 仕様のバージョンについて，Web アプリケーションのバージョンごとに次の表に示します。

表 6-3　J2EE アプリケーションが準拠する J2EE 仕様のバージョン

J2EE アプリケーションが準拠する J2EE 仕様のバージョン	Web アプリケーションに対応する Servlet 仕様				
	3.0	2.5	2.4	2.3	2.2
Java EE 6	○	○	○	○	○

J2EE アプリケーションが準拠する J2EE 仕様のバージョン	Web アプリケーションに対応する Servlet 仕様				
	3.0	2.5	2.4	2.3	2.2
Java EE 5	×	○	○	○	○
J2EE1.4	×	×	○	○	○
J2EE1.3	×	×	△	○	○
J2EE1.2	×	×	△	△	○

（凡例）

○：使用できる

△：使用できる（J2EE アプリケーションのインポート時に J2EE 仕様のバージョンが 1.4 に更新されるため）

×：使用できない

(2)　Web アプリケーションのサポート範囲

Web アプリケーションのバージョンは，web.xml に記述する Servlet 仕様のバージョン情報で識別されます。上位のバージョンの Web アプリケーションは下位のバージョンの機能を使用できます。下位のバージョンの Web アプリケーションは上位のバージョンの機能を使用できません。

Web アプリケーションのバージョンごとに，使用できる機能範囲を次の表に示します。

表 6-4　Web アプリケーションのサポート範囲

Web アプリケーションのバージョン	Servlet					JSP					タグライブラリ※1			
	3.0	2.5	2.4	2.3	2.2	2.2	2.1	2.0	1.2	1.1	2.1	2.0	1.2	1.1
3.0	○	○	○	○	○	×※2	○	○	○	○	○	○	○	○
2.5	×	○	○	○	○	×	○	○	○	○	○	○	○	○
2.4	×	×	○	○	○	×	×	○	○	○	×	○	○	○
2.3	×	×	×	○	○	×	×	×	○	○	×	×	○	○
2.2	×	×	×	×	○	×	×	×	×	○	×	×	×	○

（凡例）○：使用できる　×：使用できない

注※1

タグライブラリのバージョンとは，タブライブラリ・ディスクリプタ（TLD ファイル）のバージョンを表します。

注※2

JSP 2.2 に対応した web.xml は読み込めますが，JSP 2.2 で追加されたタグについては無視されます。

なお，下位のバージョンの Web アプリケーションから上位のバージョンの機能を使用した場合，エラーが発生することがあります。発生するエラーについてバージョンごとに次に示します。

表 6–5　Servlet 2.2，2.3，2.4 または 2.5 仕様に対応する Web アプリケーションから Servlet 3.0 の機能を使用した場合のエラー

仕様	使用する機能	エラー時の処理
Servlet 3.0	新規 API の呼び出し	Servlet 3.0 仕様で追加された API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。
	新規アノテーションの使用	アノテーションを使用してエラーとなった場合の処理については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「12.　アノテーションの使用」を参照してください。
JSP 2.2	EL の新規 API	EL2.2 の API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。

Servlet 2.2 仕様，Servlet 2.3 仕様，Servlet 2.4 仕様および Servlet 2.5 仕様から，Servlet 3.0 仕様に Web アプリケーションのバージョンアップする場合の作業，および注意事項については，「6.2.11　既存の Web アプリケーションを Servlet 3.0 仕様にバージョンアップする場合の留意点」を参照してください。

表 6–6　Servlet 2.2，2.3，または 2.4 仕様に対応する Web アプリケーションから Servlet 2.5 の機能を使用した場合のエラー

仕様	使用する機能	エラー時の処理
Servlet 2.5	新規 API の呼び出し	Servlet 2.5 仕様で追加された API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。
	新規アノテーションの使用	アノテーションを使用してエラーとなった場合の処理については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「12.　アノテーションの使用」を参照してください。
JSP 2.1	新規ディレクティブの属性※1	サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	TLD 2.1	Web アプリケーション開始時に次に示す TLD ファイルが存在した場合，メッセージログに KDJE39293-W のメッセージが出力され，処理されません。 • web.xml の<taglib>要素内の<tablib-location>要素に指定された TLD ファイル • /WEB-INF/lib ディレクトリ以下の Jar ファイル内の/META-INF ディレクトリ以下に配置された TLD ファイル Web アプリケーション開始時にこれら以外の TLD ファイルが存在した場合は，JSP コンパイル時にメッセージログに KDJE39293-W のメッセージが出力され，処理されません。アプリケーションへの初回アクセス時などに JSP コンパイルが発生した場合は，サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。

仕様	使用する機能	エラー時の処理
JSP 2.1	EL の追加機能	• JSP 2.1 で追加された Enum 型については，専用の処理は実施されないで，一般のクラスと同様に処理されます。 ただし，JSP 2.1 仕様で非推奨となった JSP 2.0 仕様の EL の API を使用した場合，Web アプリケーションのバージョンに関係なく，JSP 2.0 仕様の EL の機能範囲で処理されます。 • #{}の書式の EL は文字列として表示されます。 • "¥#"はエスケープシーケンスとして扱われません。「"¥#"」という文字列として表示されます。

注※1

JSP ページで<jsp:directive.XXX />形式で XXX に JSP 仕様で未定義のディレクティブを指定した場合，または<jsp:XXX>形式で XXX に JSP 仕様で未定義のスタンダードアクションを指定した場合，定義内容はそのまま出力されます。

注※2

KDJE39145-E には JSP のコンパイルエラーの詳細，KDJE39186-E にはトランスレーションエラーの発生通知が出力されます。

Servlet 2.2 仕様，Servlet 2.3 仕様および Servlet 2.4 仕様から，Servlet 2.5 仕様に Web アプリケーションのバージョンアップする場合の作業，および注意事項については，「6.2.12　既存の Web アプリケーションを Servlet 2.5 仕様にバージョンアップする場合の留意点」を参照してください。

表 6–7　Servlet 2.2 または 2.3 仕様に対応する Web アプリケーションから Servlet 2.4 仕様の機能を使用した場合のエラー

仕様	使用する機能	エラー時の処理
Servlet 2.4	新規 API の呼び出し	Servlet 2.4 仕様で追加された API を使用したかどうかはチェックされません。呼び出した場合の動作は保証されないため呼び出さないよう注意してください。
	新規リスナ登録	Web アプリケーションの開始時に KDJE39297-W のメッセージがメッセージログに出力され，そのリスナ定義は無視されます。
JSP 2.0	新規ディレクティブ新規スタンダードアクション※1	サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	タグファイル	TLD を使用しない場合 taglib ディレクティブで新規属性となる tagdir 属性が不正として，サーブレットログに KDJE39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。 TLD を使用する場合 TLD 2.0 を使用したエラーになります。
	TLD 2.0	次に示す TLD ファイルは，Web アプリケーション開始時にチェックされます。該当する場合，メッセージログに KDJE39293-W のメッセージが出力され，無視されます。

仕様	使用する機能	エラー時の処理
JSP 2.0	TLD 2.0	• web.xml の<taglib><tablib-location>に指定された TLD ファイル • /WEB-INF/lib 下の Jar ファイル内の/META-INF 以下に配置された TLD ファイル これら以外の TLD ファイルは，JSP コンパイル時にチェックされます。初回アクセス時など，JSP ファイルのコンパイル時は，サーブレットログに KDJE 39145-E のメッセージ，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。
	シンプル・タグ・ハンドラ	サーブレットログに KDJE39145-E のメッセージを，メッセージログに KDJE39186-E のメッセージがそれぞれ出力され※2，トランスレーションエラーとなります。

注※1

JSP ページで<jsp:directive.XXX />形式で XXX に JSP 仕様で未定義のディレクティブを指定した場合，または<jsp:XXX>形式で XXX に JSP 仕様で未定義のスタンダードアクションを指定した場合，定義内容はそのまま出力されます。

注※2

KDJE39145-E には JSP のコンパイルエラーの詳細，KDJE39186-E にはトランスレーションエラーの発生通知が出力されます。

Servlet 2.2 仕様および Servlet 2.3 仕様から，Servlet 2.4 仕様に Web アプリケーションのバージョンアップする場合の作業，および注意事項については，「6.2.14　既存の Web アプリケーションを Servlet 2.4 仕様にバージョンアップする場合の留意点」を参照してください。

なお，Servlet 2.2 仕様に対応する Web アプリケーションから Servlet 2.3 の機能を使用しても，アプリケーションのインポート時に Servlet 2.3 仕様に準拠した Web アプリケーションに書き換えられるため，正常に処理され，エラーは通知されません。

(3)　トランザクションと JDBC コネクション利用時の注意

サーブレット，JSP でトランザクションを利用する場合，該当するサービスメソッドで JDBC コネクションを取得し，該当するサービスメソッドが終了する前に解放してください。トランザクションが開始しているサーブレットおよび JSP では，次に示す JDBC コネクションの使用はサポートされません。

- サーブレット，JSP のサービスメソッドが生成したスレッド上の JDBC コネクションを使用する。
- サーブレット，JSP のサービスメソッドから呼び出した別のサーブレット，JSP のサービスメソッドで JDBC コネクションを使用する。
- サーブレット，JSP のサービスメソッドの init メソッドで取得した JDBC コネクションを使用する。
- インスタンス変数に格納された JDBC コネクションを使用する。※

注※

SingleThreadModel のサーブレットおよび JSP を使用した場合は，インスタンス変数に JDBC コネクションを格納できます。

(4)　パッケージ名の指定に関する注意

不正なパッケージ名が指定されたクラスをサーブレットおよび JSP で使用した場合，ブラウザからアクセスしたときにステータスコード 500 のエラーになります。例えば，作成したクラスファイルを正しく配置

して，ブラウザからアクセスしても，パッケージ名の宣言に不正があった場合は，該当クラスが見つかりません。この場合，ステータスコード 500 のエラーが返されます。

(5)　Cookie 利用時の注意

- 日本語などの 2 バイトコードを含む Cookie は使用しないでください。使用した場合，サーブレットおよび JSP で利用している HTTP セッションが失われる場合があります。
- Cookie でセッション管理をする場合，ホスト名による URL でアクセスしたサーブレットまたは JSP で生成されたセッションは，ホスト名の代わりに IP アドレスを指定した URL でアクセスしたサーブレットまたは JSP に引き継がれません（逆も同様です）。

(6)　特別な意味を持つ入力値の表示に関する注意

フォームなどで「<」や「>」などの特別な意味を持つ文字の入力値をそのまま表示した場合，悪意のあるユーザが<SCRIPT>，<OBJECT>，<APPLET>，<EMBED>のスクリプトなどを実行できるタグを使用して，重大なセキュリティ上の問題を引き起こすおそれがあります。アプリケーション開発者は，ユーザから入力されたデータに対して必ず検査をする処理を追加して，特別な意味を持つ文字を排除する必要があります。

(7)　コミット後のエラーページの表示に関する注意

サーブレットまたは JSP でレスポンスがコミットされたあとは，例外などのエラーが発生したとしても，次に示すエラーページはブラウザに表示されません。

- web.xml で指定したエラーページ
- JSP の page ディレクティブの errorPage 属性で指定したエラーページ
- Web コンテナサーバが出力するデフォルトのエラーページ

レスポンスのコミットは，ユーザが ServletResponse クラスの flushBuffer メソッドなどを明示的に呼び出してコミットする場合以外にも，レスポンスのバッファが満杯になって自動的に Web コンテナがコミットすることがあります。

サーブレットまたは JSP でコミットされているかどうかを調べるには，ServletResponse クラスの isCommitted メソッドを使用します。また，バッファサイズの変更は，サーブレットの場合は ServletResponse クラスの setBufferSize メソッドで，JSP の場合は page ディレクティブの buffer 属性の指定で実施できます。

(8)　PrintWriter，JSPWriter クラス利用時の性能向上について

PrintWriter クラスおよび JSPWriter クラスの print メソッドと println メソッドの呼び出し回数を少なくすることで，アクセス回数を減らし，性能を向上できます。例えば，StringBuffer クラスを使用し，最後に println メソッドを呼び出すようにして，print および println メソッドの呼び出し回数を削減します。

(9)　javax.servlet.error.XXXXX によるエラー情報参照時の注意

Servlet 2.3 仕様で定義されている javax.servlet.error.XXXXX 属性は，web.xml の<error-page>タグに指定されたサーブレットまたは JSP 内でそのエラーページを実行する要因となったエラー情報を参照するためのものです。web.xml の<error-page>タグに指定されたサーブレットまたは JSP 以外からは，これらの属性を参照しないでください。

(10)　ファイルアクセス時の注意

ファイルにアクセスする場合は，必ず絶対パスを指定してください。相対パスを指定すると，J2EE サーバは Web コンテナサーバの実行ディレクトリからの相対パスによって目的のパスを検索しようとします。ServletContext クラスの getRealPath メソッドで相対パスを指定すると，WAR ファイルを展開したディレクトリでの相対パスが取得されます。

また，ファイルにアクセスする場合は，必ずファイルをクローズしてください。WAR ファイル展開ディレクトリでファイルにアクセスしてクローズしないと，J2EE サーバで正常にアンデプロイできなくなります。WAR ファイルの展開ディレクトリ下のパスを指定していない場合でもファイルをクローズしていないと，J2EE サーバの起動中にファイルを削除できないなどの現象が発生します。

(11)　例外発生時のエラーページの設定について

JSP，サーブレットへのアクセスで例外が発生した場合，Web コンテナのデフォルトの処理では例外のステータスコードをブラウザに返します。このデフォルトの処理を変える場合は JSP の errorPage の指定や web.xml でエラーページを設定してください。

(12)　クラスローダの取得に関する注意

J2EE アプリケーション内のコードから Component Container のクラスローダを取得して，次に示すメソッドを使用する場合に，java.net.JarURLConnection クラスが使用されます。

- getResource(String).openConnection().getInputStream()
- getResource(String).openStream()

これらのメソッドが呼び出される過程で java.net.JarURLConnection クラスの openConnection メソッドが呼び出され，該当する URL に指定された JAR ファイルがオープンされます。この JAR ファイルは close メソッドを明示的に呼ばないかぎり，オープンされたままになり削除できません。これらのメソッドは J2EE アプリケーション内で使用しないでください。また，JAR ファイルに対する操作が必要で java.net.JarURLConnection クラスの openConnection メソッドを使用する場合には，java.net.JarURLConnection の getJarFile メソッドが返す JarFile インスタンスの close メソッドを必ず呼び出すようにしてください。

(13)　URLConnection クラス使用時の注意

java.net.URLConnection クラスは setUseCaches(boolean)メソッドを使用して，指定された URL に対してコネクションを取得するときにキャッシュの情報を利用するかどうかを指定できます。
URLConnection クラスに対して setUseCaches(false)メソッドを指定した場合に，コネクションごとに対象のオブジェクトが生成されます。J2EE アプリケーション内のコードから使用する場合には，J2EE サーバの JavaVM がメモリ不足となるおそれがあります。

(14)　ネイティブライブラリのロードに関する注意

System.loadLibrary メソッドを使用して，サーブレットおよび JSP からネイティブライブラリをロードしないでください。サーブレットおよび JSP でネイティブライブラリをロードすると，JNI 仕様の制約によって，java.lang.UnsatisfiedLinkError が発生することがあります。ネイティブライブラリのロードが必要な場合は，System.loadLibrary メソッドを呼び出すコンテナ拡張ライブラリを作成し，サーブレットおよび JSP からコンテナ拡張ライブラリを参照するように実装してください。コンテナ拡張ライブラリの作成については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.　コンテナ拡張ライブラリ」を参照してください。

(15)　ユーザスレッドの使用方法

アプリケーションを構成するサーブレットおよび JSP からスレッドを生成して，使用できます。ユーザがプログラムの中で明示して生成するスレッドのことを，**ユーザスレッド**といいます。

ユーザスレッドは，生成後の動作のしかた（ライフサイクル）によって，次の二つに分けられます。

- サービスメソッドや init メソッドの範囲内で動作させる。
- サービスメソッドや init メソッドのバックグラウンドで動作させる。

ユーザスレッドの使用条件

- ユーザスレッドは，Enterprise Bean では使用できません（EJB 仕様で，Enterprise Bean からのスレッドの生成が禁止されているため）。
- ユーザスレッドが使用できるアプリケーションサーバのサーバモードを次に示します。

表 6-8　ユーザスレッドの前提条件（サーバモード）

サーバモード		使用可否
J2EE サーバモード	1.4 モード	○
	ベーシックモード	△
サーブレットエンジンモード		△

（凡例）

○：使用できる

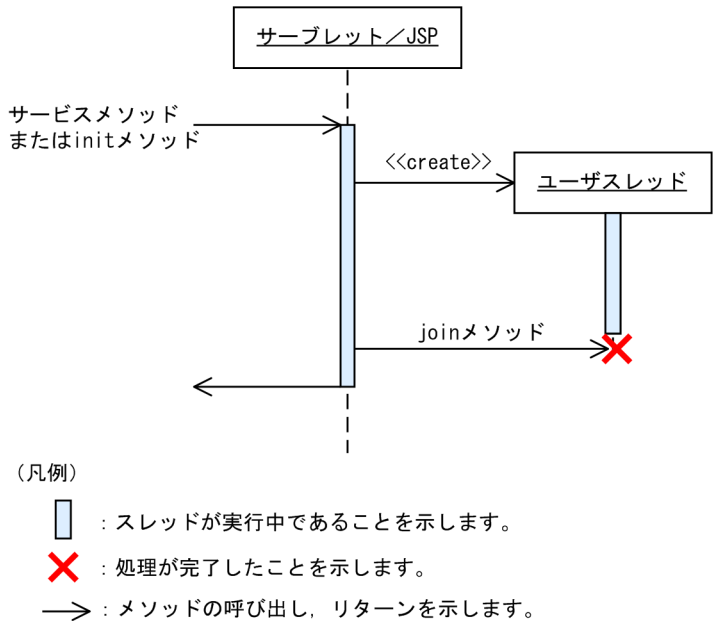
△：サーバモード（ベーシックモードまたはサーブレットエンジンモード）でサポートされている範囲で機能を使用できます。ただし，コネクションプールは使用できません。なお，これらの機能は互換用の機能です。

ユーザスレッドを使用する場合のライフサイクルについて説明します。

(a)　サービスメソッドや init メソッドの範囲内で動作させる場合

サービスメソッドや init メソッドでユーザスレッドの処理を完了させるモデルです。このモデルの処理の流れを次の図に示します。

図 6-1　サービスメソッドや init メソッドの範囲内で動作させる場合の処理

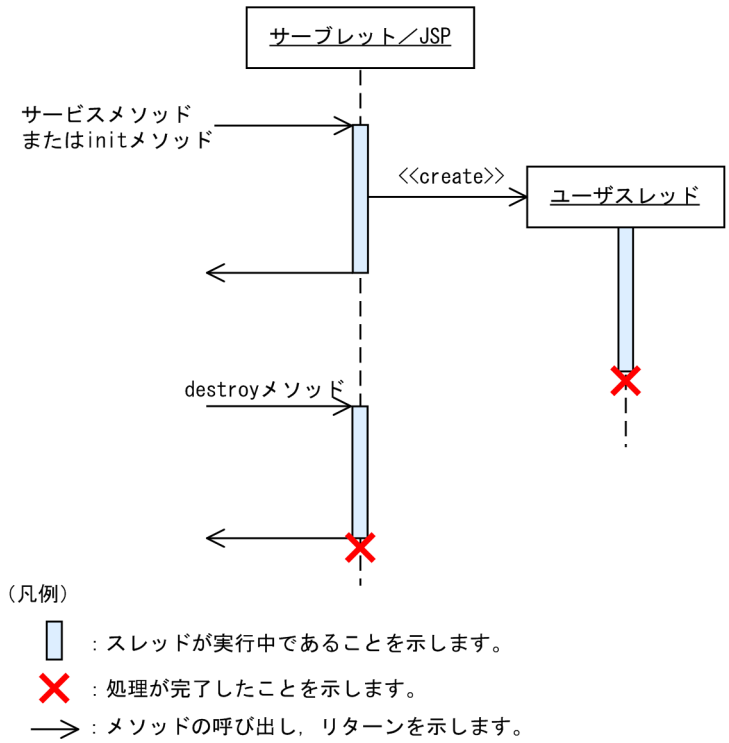


サービスメソッドや init メソッドの呼び出しの範囲内で，ユーザスレッドを生成します。サービスメソッドや init メソッドでは，join メソッドによってユーザスレッドの処理が完了するのを待ってから，リターンします。

(b)　サービスメソッドや init メソッドのバックグラウンドで動作させる場合

サービスメソッドや init メソッドでユーザスレッドを生成し，その後ユーザスレッドをバックグラウンドで動作させるモデルです。このモデルの処理の流れを次の図に示します。

図 6-2　サービスメソッドや init メソッドのバックグラウンドで動作させる場合の処理



ユーザスレッドを生成したサービスメソッドや init メソッドは，ユーザスレッドを生成したあと，処理の完了を待たないでリターンします。ただし，アプリケーションを停止したあとは，ユーザスレッドから J2EE サービスを利用できなくなります。したがって，アプリケーションの停止によって javax.servlet.ServletContextListener の contextDestroyed メソッドか，JSP または Servlet の destroy メソッドでユーザスレッドを停止すれば問題ありません。

(16)　セッション情報の永続化について

Web コンテナではセッション情報の永続化はサポートされません。Web コンテナではセッション情報は，正常，異常に関係なく Web コンテナが終了すると失われます。セッション情報を保持したい場合は，セッションフェイルオーバ機能を使用してください。

また，web.xml で<distributable>タグを指定した場合，および Serializable でないオブジェクトをセッション情報として登録した場合も IllegalArgumentException は発生しません。

(17)　init メソッドおよび destroy メソッドをオーバーライドしていない場合に出力されるメッセージ

init メソッドおよび destroy メソッドをオーバーライドしていないサーブレットを初期化または終了すると，次の形式のログがサーブレットログに出力されます。

- メッセージ ID：KDJE39037-I
- メッセージ本文：path="aa....aa" :bb....bb: init※

  aa....aa

  「/」から始まるコンテキストパスを表します。

bb....bb

web.xml の<servlet-name>タグで指定したサーブレット名を表します。デフォルトマッピングのサーブレットの場合は，「org.apache.catalina.INVOKER.<クラス名>」となります。

注※

init メソッドの場合は「init」，destroy メソッドの場合は「destroy」となります。出力されるメッセージは，それぞれ javax.servlet.GenericServlet クラスの init メソッドおよび destroy メソッドで出力されるログです。したがって，init メソッドまたは destroy メソッドをオーバーライドしたサーブレットではこれらのメッセージは出力されません。

また，JSP の場合は，page ディレクティブの extends 属性で指定する JSP の基底クラスで init メソッドおよび destroy メソッドをオーバーライドしなかった場合，同様のメッセージが出力されます。その場合，サーブレット名は"com.hitachi.software.web.servlet-name.jsp"となります。JSP で page ディレクティブの extends 属性を指定しなかった場合は，init メソッドのログだけが出力され，destroy メソッドのログは出力されません。

ただし，サーブレットの場合も JSP の場合も，init メソッドおよび destroy メソッドをオーバーライドしてスーパークラスの init メソッドおよび destroy メソッドを呼ぶときは，このメッセージを出力します。

(18)　javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性について

javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性について，Servlet で例外をスローした場合と JSP ファイルで例外をスローした場合の二つに分けて説明します。

(a)　Servlet で例外をスローした場合

Servlet でスローした例外クラスが java.lang.Error，またはその派生クラスの場合

javax.servlet.ServletException クラスの例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。Servlet でスローした例外は，javax.servlet.ServletException クラスの getRootCause メソッドで取得できます。

Servlet でスローした例外クラスが java.lang.Error，またはその派生クラス以外のクラスの場合

Servlet でスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

(b)　JSP ファイルで例外をスローした場合

●　エラーページが JSP ファイルの場合

web.xml の<error-page>タグでエラーページを指定した場合

web.xml の<error-page>タグでエラーページを指定した場合について，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

JSP ファイルでスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

JSP 1.2

JSP ファイルでスローした例外クラスが次のクラスのどれかであれば，JSP ファイルでスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

- java.io.IOException，またはその派生クラス

- java.lang.RuntimeException，またはその派生クラス
- javax.servlet.ServletException，またはその派生クラス

JSP ファイルでスローした例外クラスがこれら以外の場合，javax.servlet.ServletException クラスの例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。JSP ファイルでスローした例外は，javax.servlet.ServletException クラスの getRootCause メソッドで取得できます。

page ディレクティブの errorPage 属性でエラーページを指定した場合

page ディレクティブの errorPage 属性でエラーページを指定した場合について，エラーページで page ディレクティブの isErrorPage 属性に true を指定した場合と false を指定した場合に分けて示します。

エラーページで page ディレクティブの isErrorPage 属性に true を指定した場合

JSP ファイルでスローした例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に設定されます。

エラーページで page ディレクティブの isErrorPage 属性に false を指定した場合

javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に値は設定されません。

● エラーページが Servlet の場合

web.xml の<error-page>タグでエラーページを指定した場合

エラーページが JSP ファイルの場合の，web.xml の<error-page>タグでエラーページを指定した場合と同様です。

page ディレクティブの errorPage 属性でエラーページを指定した場合

javax.servlet.ServletRequest オブジェクトの javax.servlet.error.exception 属性に値は設定されません。

(19)　バイナリデータを含む Web アプリケーションの操作について

バイナリデータを含む Web アプリケーションでは，次のことに注意してください。

- クライアントから送信されたバイナリデータへのリクエストを実行する場合
  バイナリデータへのリクエストで適用されるフィルタ内で，レスポンスオブジェクトからの PrintWriter を取得しないでください。
- クライアントから送信されたリクエストを処理するサーブレットまたは JSP がディスパッチする場合
  次の場所では，レスポンスオブジェクトからの PrintWriter を取得しないでください。
  - バイナリデータへのリクエストで適用されるフィルタ内
  - バイナリデータにディスパッチするサーブレットまたは JSP 内

参考

バイナリデータとは，拡張子にマッピングされた MIME タイプが"text/"から始まっていない静的コンテンツ，またはマッピングが存在しない静的コンテンツです。

(20)　レスポンスの文字エンコーディングに関する注意

JSP またはサーブレットのレスポンスボディの文字エンコーディングが，UTF-16（16 ビット UCS 変換形式）の場合，ブラウザによって正しく表示できない場合があります。その場合は，JSP またはサーブレットの文字エンコーディングに，UTF-16BE（16 ビット UCS 変換形式のビッグエンディアンバイト順），または UTF-16LE（16 ビット UCS 変換形式のリトルエンディアンバイト順）を使用してください。

(21)　javax.servlet.ServletRequest インタフェースの getServerName メソッドおよび getServerPort メソッドの戻り値について

getServerName メソッド，および getServerPort メソッドの戻り値について説明します。

Servlet 2.4 仕様以降では，Host ヘッダの有無によって，getServerName メソッド，および getServerPort メソッドの戻り値が異なります。Servlet 2.4 仕様以降での getServerName メソッド，および getServerPort メソッドの戻り値を次の表に示します。

表 6-9　getServerName メソッド，および getServerPort メソッドの戻り値（Servlet 2.4 仕様以降の場合）

Host ヘッダの有無	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
あり	Host ヘッダの「:」より前の部分	Host ヘッダの「:」よりあとの部分
なし	解決されたサーバ名または IP アドレス	クライアントとの接続を受け付けたサーバのポート番号

アプリケーションサーバでは，getServerName メソッド，および getServerPort メソッドの戻り値は，HTTP リクエストと，使用する機能の組み合わせによって得られます。なお，HTTP 1.1 のリクエストに Host ヘッダが含まれない場合，HTTP 1.1 仕様に従って，400 エラーとなります。また，HTTP 1.1 仕様では，リクエストラインのリクエスト URI が絶対 URI の場合，ホストにはリクエスト URI のホストを使用して，Host ヘッダの内容は無視するように定義されています。なお，Servlet 仕様では明記されていませんが，HTTP 仕様に従って，リクエストラインの URI に含まれるホスト名を優先するようになっています。

HTTP リクエストと，使用する機能の組み合わせによって得られる，getServerName メソッド，および getServerPort メソッドの戻り値を次の表に示します。なお，ゲートウェイ指定機能を使用している場合の，getServerName メソッド，および getServerPort メソッドの戻り値については，表 6-9 を参照してください。

表 6-10　getServerName メソッド，および getServerPort メソッドの戻り値（アプリケーションサーバの場合）

HTTP リクエスト		使用する機能	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類			
あり	絶対 URI	Web サーバ連携	リクエストラインのホスト名	リクエストラインのポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	Host ヘッダのホスト名	Host ヘッダのポート番号
		インプロセス HTTP サーバ	Host ヘッダのホスト名	Host ヘッダのポート番号
なし	絶対 URI	Web サーバ連携	リクエストラインのホスト名	リクエストラインのポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号

HTTP リクエスト		使用する機能	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類			
なし	相対 URI	Web サーバ連携	Web サーバのホスト名または IP アドレス※2	Web サーバのポート番号
		インプロセス HTTP サーバ	J2EE サーバのホスト名または IP アドレス※1	インプロセス HTTP サーバのポート番号

注※1　java.net.InetAddress.getLocalHost メソッド，または getHostName メソッドの戻り値となります。

注※2　HTTP Server を使用する場合，ServerName ディレクティブに指定した値となります。ServerName ディレクティブについては，マニュアル「HTTP Server」を参照してください。

ゲートウェイ指定機能使用時の getServerName メソッド，および getServerPort メソッドの戻り値を次の表に示します。

表 6–11　ゲートウェイ指定機能使用時の getServerName メソッド，および getServerPort メソッドの戻り値（アプリケーションサーバの場合）

HTTP リクエスト		使用する機能	getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類			
あり	絶対 URI	Web サーバ連携	リクエストラインのホスト名	リクエストラインのポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	Host ヘッダのホスト名	Host ヘッダのポート番号
		インプロセス HTTP サーバ	Host ヘッダのホスト名	Host ヘッダのポート番号
なし	絶対 URI	Web サーバ連携	ゲートウェイ指定機能で指定したホスト名	ゲートウェイ指定機能で指定したポート番号
		インプロセス HTTP サーバ	リクエストラインのホスト名	リクエストラインのポート番号
	相対 URI	Web サーバ連携	ゲートウェイ指定機能で指定したホスト名	ゲートウェイ指定機能で指定したポート番号
		インプロセス HTTP サーバ	ゲートウェイ指定機能で指定したホスト名	ゲートウェイ指定機能で指定したポート番号

(22)　javax.servlet.ServletException クラスのコンストラクタで指定した根本原因の例外の取得について

アプリケーションサーバでは，コンストラクタ ServletException(String, Throwable)または ServletException(Throwable)で指定した根本原因の例外を getCause メソッドで取得できます。なお，

getRootCause メソッドでも取得できます。ただし，07-60 以前のバージョンでは getCause メソッドには null を返します。

javax.servlet.ServletException クラスのコンストラクタで指定した根本原因の例外の取得について，互換用のパラメタおよび注意事項について説明します。

- 互換用のパラメタ

  07-60 以前と同じ動作にする場合，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で互換用のパラメタ webserver.servlet_api.exception.getCause.backcompat に true を指定します。パラメタの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

  互換用のパラメタ webserver.servlet_api.exception.getCause.backcompat の指定値による getCause メソッドおよび getRootCause メソッドの戻り値の違いを次の表に示します。

  表 6-12　webserver.servlet_api.exception.getCause.backcompat の指定値による getCause メソッドおよび getRootCause メソッドの戻り値の違い

メソッド	webserver.servlet_api.exception.getCause.backcompat の指定値	
	true	false
getCause()	×	○
getRootCause()	○	○

（凡例）○：根本原因の例外を返す　×： null を返す

なお，この互換用のパラメタの指定内容は，javax.servlet.jsp.JspException クラスの getCause メソッドおよび getRootCause メソッドの動作にも適用されます。

- 注意事項

  根本原因の例外を getCause メソッドの実装によって取得できる場合，コンストラクタ ServletException(String, Throwable)または ServletException(Throwable)で生成した ServletException オブジェクトに対して initCause(Throwable)を呼び出すことはできません。initCause(Throwable)を呼び出した場合，java.lang.IllegalStateException 例外がスローされます。

(23)　javax.servlet.ServletOutputStream オブジェクトに対する flush メソッドの実行についての注意

アプリケーションサーバでは，javax.servlet.ServletResponse オブジェクトから取得する javax.servlet.ServletOutputStream オブジェクトに対して，close メソッドを実行したあとで flush メソッドを実行しても，java.io.IOException 例外をスローしません。

(24)　リクエスト URI の正規化

アプリケーションサーバでは，リクエスト URI に含まれる文字列は，正規化されたあと，次に示すマッチング処理で使用されます。

- コンテキストパスとコンテキストルートのマッチング
- サーブレットおよび JSP の URL パターンとのマッチング
- デフォルトマッピングとのマッチング
- 静的コンテンツとのマッチング
- フィルタの URL パターンとのマッチング

- web.xml の<error-page>タグ，または JSP の page ディレクティブの errPage 属性で指定するエラーページとのマッチング
- アクセスを制限する URL パターンとのマッチング
- ログイン認証の URL 判定
- リクエストのフォワードおよびインクルード
- HTTP レスポンス圧縮フィルタの URL パターンとのマッチング
- URL グループ単位の同時実行スレッド数制御の URL パターンとのマッチング
- インプロセス HTTP サーバのエラーページのカスタマイズ
- インプロセス HTTP サーバのリダイレクトによるリクエストの振り分け

(25)　javax.servlet.http.HttpServletRequest インタフェースの getRequestURI メソッドおよび getRequestURL メソッドの戻り値について

javax.servlet.http.HttpServletRequest インタフェースの getRequestURI メソッドおよび getRequestURL メソッドでは，正規化された URL が戻り値となります。

(26)　welcome ファイルに URL マッピングされた Servlet または JSP の指定

リクエスト URL が URL マッピングされた Servlet または JSP と一致しないで，welcome ファイルに転送される必要がある場合，Web コンテナでは次のように転送先の welcome ファイルが選択されます。

まず，指定された welcome ファイル名から静的コンテンツや JSP ファイルの候補が優先して選択されます。該当するものがない場合，URL マッピングされた Servlet または JSP の候補が選択されます。

welcome ファイルに関する注意事項について説明します。

● welcome ファイル転送方式による制約

welcome ファイルの転送は，HTTP リダイレクト（HTTP ステータスコード 302 でブラウザがリダイレクトする）によって実現しています。この転送方式には制約があるため，URL 設計の際に次のことに注意してください。

- POST リクエストを受け付けた際，ブラウザから送信されたリクエストボディの情報を転送先の welcome ファイルに引き継げません。POST された情報がフォーム入力形式（Content-Type が application/x-www-form-urlencoded）の場合だけ，Web コンテナが生成する welcome ファイル転送先 URL のクエリ文字列に情報を付与する形で引き継げます。ただし，この場合も，リクエストボディの情報が多い場合に転送先 URL が長くなり過ぎる，ブラウザのアドレスバーにクエリ文字列として情報がそのまま見える，などについて考慮が必要です。
- 転送先の welcome ファイルのサーブレットが doGet メソッドを実装していない場合，ブラウザに「400 Bad Request」（HTTP/1.1 以外の場合）または「405 Method Not Allowed」（HTTP/1.1 の場合）が表示されます。
- Web アプリケーションから javax.servlet.RequestDispatcher インタフェースの include メソッドを呼び出した際，インクルードする対象の URL として welcome ファイルが存在するディレクトリを指定していても，転送先の welcome ファイルのコンテンツは挿入されません。

● JSP 事前コンパイル済み環境での welcome ファイルの追加

JSP 事前コンパイル済みの Web アプリケーションに，welcome ファイルに指定した JSP ファイルを追加する場合，JSP ファイルの追加後に JSP 事前コンパイルを再度実行する必要があります。JSP 事前コンパイルを再度実行しなかった場合，正しく welcome ファイル転送処理が実行されません。

● サーブレットクラスが参照できないサーブレットの welcome ファイルの指定

サーブレットクラスが参照できないサーブレットを welcome ファイルに指定しないでください。サーブレットクラスが参照できないサーブレットを指定した場合，正しく welcome ファイル転送処理が実行されません。

● ディレクトリが存在しないパスへの welcome ファイル要求

Web アプリケーション内のリソースとして存在しないディレクトリのパスに対するリクエストの場合，リクエスト URL の末尾が「/」であっても welcome ファイル転送処理は実行されません。

(27)　サーブレット，フィルタ，リスナの開始・終了順序

Web アプリケーションを開始すると，リクエストの受付を開始する前に次の順序で初期化処理をすることが Servlet 2.4 仕様で明確化されました。アプリケーションサーバでは，Servlet2.3 以前の Web アプリケーションでも同じ順序で初期化処理をします。Web アプリケーション開始時のサーブレット，フィルタ，およびリスナは次の順序で開始されます。

1. リスナの開始（インスタンスの生成※1，@PostConstruct アノテーションのメソッドおよび ServletContextListener の contextInitialized メソッドの呼び出し※2)
2. フィルタの開始（インスタンスの生成※1，@PostConstruct アノテーションのメソッドおよび init メソッドの呼び出し）
3. load-on-startup タグで指定された Servlet/JSP の開始（インスタンスの生成※1，@PostConstruct アノテーションのメソッドおよび init メソッドの呼び出し）

注※1　Servlet 3.0 以降では，API 呼び出しによって動的にサーブレット，フィルタ，リスナを追加できますが，インスタンスを指定する API 呼び出しによって定義を追加したサーブレット，フィルタ，リスナについては，インスタンス生成済みのため，Web コンテナではインスタンスを生成しません。

注※2　リスナの contextInitialized()メソッドの呼び出しで例外が発生しても，KDJE39103-E のメッセージを出力して Web アプリケーションの開始処理を継続します。

なお，web.xml の<load-on-startup>要素によって Web アプリケーション開始時の初期化処理実行を指定しなかったサーブレットについては，初回リクエスト実行時にサーブレットのインスタンスの生成および init()メソッドを呼び出します。

このとき，サーブレットのインスタンスの生成および init()メソッドはフィルタより前に呼び出します。

Web アプリケーション終了時のサーブレット，フィルタ，およびリスナは次の順序で終了します。

1. 開始済みの Servlet/JSP の終了(destroy メソッド，@PreDestroy アノテーションのメソッド呼び出し)
2. フィルタの終了(destroy メソッド，@PreDestroy アノテーションのメソッド呼び出し)
3. リスナの終了(@PreDestroy アノテーションのメソッド呼び出し)

(28)　Web アプリケーション内の静的コンテンツへのアクセス

Web アプリケーション内の静的コンテンツへのアクセス時に使用できるメソッドは，GET，HEAD，POST，TRACE，OPTIONS のどれかです。

POST メソッドを使用した場合，GET メソッド使用時と同様，静的コンテンツの内容を応答します。

(29)　文字エンコーディングに関する注意

同じ Web アプリケーション内では，web.xml で指定したエラーページと，HTTP レスポンスに文字エンコーディングを使用するサーブレットおよび JSP に，同じ文字エンコーディングを使用してください。

(30)　クエリ文字列にイコール（"="）以降だけ指定した場合の戻り値について

リクエストのクエリ文字列にイコール（"="）以降しか指定していない場合（例えば，http://localhost/application/getparam.jsp?=param のような場合），使用している Web サーバ種別によって，javax.servlet.ServletRequest インタフェースのリクエストパラメタを取得する Servlet API の戻り値が異なります。

Web サーバ種別ごとの Servlet API の戻り値を次に示します。

- Web サーバ連携機能または簡易 Web サーバを使用している場合
  - getParameter メソッド
    空文字（""）を指定するとイコール（"="）以降に指定されたパラメタを返します。
  - getParameterMap メソッド
    空文字（""）をキーとするパラメタを含む java.util.Map オブジェクトを返します。
  - getParameterNames メソッド
    空文字（""）を含む java.util.Enumeration オブジェクトを返します。
  - getParameterValues メソッド
    空文字（""）を指定するとイコール（"="）以降に指定されたパラメタを返します。
- インプロセス HTTP サーバを使用している場合
  - getParameter メソッド
    空文字（""）を指定しても null を返します。
  - getParameterMap メソッド
    空の java.util.Map オブジェクトを返します。
  - getParameterNames メソッド
    空の java.util.Enumeration オブジェクトを返します。
  - getParameterValues メソッド
    空文字（""）を指定しても null を返します。

(31)　javax.servlet.http.HttpServletResponse インタフェースの containsHeader メソッドについて

以下のレスポンスヘッダは Web コンテナにより自動的にレスポンスにセットされる場合があります。このようなレスポンスヘッダは，javax.servlet.http.HttpServletResponse インタフェースの containsHeader メソッドで，レスポンスにセットされているかどうかを確認できません。

- Web サーバ連携の場合
  - Content-Length
  - Content-Type
  - Set-Cookie
- インプロセス HTTP サーバの場合
  - Connection

- Content-Language
- Content-Length
- Content-Type
- Date
- Server
- Set-Cookie
- Transfer-Encoding

(32)　アプリケーションサーバのライブラリに関する注意

アプリケーションサーバのライブラリを J2EE アプリケーションに含めると，ライブラリのバージョン不整合などが原因で，アプリケーションのインポートや開始，実行で不正な動作になることがあります。そのため，製品が使用方法として明示している場合を除いて，アプリケーションサーバのライブラリは J2EE アプリケーションに含めないようにしてください。

6.2.2　サーブレット実装時の注意事項

サーブレットを実装するときの注意事項を示します。

(1)　入出力ストリーム利用時の注意

- ServletOutputStream クラスの print(char c)メソッドの引数には，0x00～0xFF の範囲で指定してください。範囲外の値を指定すると java.io.CharConversionException が発生します。
- ServletInputStream クラスでは，mark メソッドおよび reset メソッドをサポートしていません。また，markSupported メソッドは常に false を返します。
- ServletInputStream クラスからデータを読み出したあとに転送（forward）した場合，転送先の ServletInputStream クラスから読み出されるデータは，転送する前に読み出された位置からのものとなります。また，転送する前に ServletInputStream クラスからすべてのデータを読み出した場合，転送先のリクエストパラメタは空となります。

(2)　ロケール設定時の注意

ServletResponse クラスの setLocale メソッドに Locale.JAPANESE を指定した場合，Content-Type ヘッダの charset は Shift_JIS（シフト JIS）になります。

(3)　URI 取得時の注意

HttpServletRequest クラスの getRequestURI メソッドでは，最適化された URI が返されます。例えば，「xxx//yyy/zzz」は「xxx/yyy/zzz」に，「xxx/yyy/../zzz」は「xxx/zzz」のように変換されます。

(4)　POST データの読み込み失敗時の動作について

Web サーバで POST データの読み込みに失敗した場合，Web コンテナで動作するサーブレットでは，ServletRequest クラスの次に示すメソッドの呼び出し時に IllegalStateException 例外が発生します。

- getParameter メソッド
- getParameterNames メソッド
- getParameterValues メソッド
- getParameterMap メソッド

また，Content-Type が multipart/form-data のフォームデータを受信した場合は，上記のメソッドまたは HttpServletRequest クラスの次に示すメソッドの呼び出し時に，KDJE39336-E のメッセージの出力を伴う IllegalStateException 例外が発生することがあります。この場合，受信したフォームデータのサイズが適切かどうかを確認し，適切なサイズのときは webserver.connector.limit.max_post_form_data の設定値を見直してください。

- getPart メソッド
- getParts メソッド

(5)　属性の変更に対するイベント通知時の注意

ServletContextAttributeListener インタフェース，HttpSessionAttributeListener インタフェース，および ServletRequestAttributeListener では，Web コンテナが内部で使用している属性の追加，削除，更新があった場合にもイベントが通知される場合があります。通知されたイベントの属性名を参照して，Web アプリケーションで使用している属性名以外の場合には無視するようにしてください。

(6)　ServletContext インタフェース利用時の注意

- ServletContext インタフェースの getResourcePaths メソッドによって得られる java.util.Set は参照用です。この java.util.Set に対し，要素の追加，削除の操作をしないでください。add，addAll，clear，remove，および removeAll メソッドを使用すると IllegalStateException 例外が発生する場合があります。
- ServletContext インタフェースの getContext メソッドの引数には，存在するコンテキストルート名を使用した URL を指定してください。存在しないコンテキストルート名を使用した URL を指定した場合の動作は保証されません。
- ServletContext インタフェースの getResource メソッドおよび getResourceAsStream メソッドには，該当 Web アプリケーションに含まれるリソースを指定してください。該当 Web アプリケーション外のリソースを指定した場合の動作は保証されません。

(7)　Web アプリケーションに含まれるディレクトリにアクセスするときの注意

Web アプリケーションに含まれるディレクトリにアクセスするときは，クエリ文字列および POST データがリダイレクト先リソースで取得できないことがあるため，クエリ文字列および POST データは付けないようにしてください。

(8)　ServletRequest インタフェース利用時の注意

ServletRequest インタフェースの getRemoteHost メソッドは，リクエストを送信したクライアントのホスト名を返しますが，Web サーバがホスト名を解決できないか，解決しないように設定されている場合は IP アドレスを返します。

既定の設定では Web サーバの設定がされていないため，IP アドレスを返します。ホスト名を取得する場合は，Web サーバの設定を変更する必要があります。ただし，設定を変更すると，ホスト名の解決のため，レスポンスが遅くなる場合があります。Web サーバの設定の変更方法については，マニュアル「HTTP Server」を参照してください。

(9)　プロセス内で複数回実行してはならない処理を実装する場合の注意

1 プロセス内で複数回実行してはならない処理をサーブレット中に記述する場合，サーブレットの実行とその処理が同期することがないようにしてください。特に，OTM との通信を開始するための初期化処理の中には，インスタンスを削除しても終了しない常駐スレッドを生成するものがあります。例えば，TPBroker の初期化関数である ORB.init メソッドは，呼び出されるたびに GC 実行のための監視用常駐スレッドを生

成し，このスレッドはプロセス終了まで消えません。そのため，1 プロセス内で ORB.init メソッドを必要以上に実行すると，不要な GC 処理が増え，システム全体の性能が著しく低下するなどの悪影響があります。

このような事態を避けるため，プロセス中で 1 回だけ実行させたい処理をサーブレットに記述する場合には，その処理がプロセス中で実行済みかどうかをあらかじめ判定する必要があります。具体的には，ある処理が実行済みかどうかの状態を格納する条件フラグとして static 変数を任意のクラス中で用意します。static 変数の値が「未実行」を意味するものである場合にだけ処理を実行し，値を「実行済み」を意味するものに変更することで，その処理の実行回数をプロセス中で 1 回だけに限定できます。ただし，次の 2 点に注意してください。

- 任意のクラスで static 変数を使用する場合は，usrconf.properties または hitachi_web.properties に次の設定をしないでください。

  ・webserver.context.reloadable=true

  ・webserver.jsp.recompilable=true

  設定した場合，そのクラスのインスタンスが自動的に破棄されて，再生成されることがあるため，それに伴って static 変数の値も初期化されてしまうことがあります。usrconf.properties と hitachi_web.properties の設定については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。
- あるスレッドが条件フラグである static 変数の値を参照してから値を変更するまでの間に，ほかのスレッドが同様の処理を実行することがないよう，この処理を実行するメソッドに synchronized キーワードを指定してください。この方法を用いて TPBroker の初期化関数 ORB.init メソッドが 1 回だけ呼び出されるようにするためのコーディング例を次に示します。

```
static org.omg.CORBA.ORB _orb=null;
public static synchronized org.omg.CORBA.ORB getORB(String[] args, Properties props){
  if(_orb==null){
    _orb=org.omg.CORBA.ORB.init(args,props);
  }
return _orb;
}
```

(10)　ゲートウェイ指定機能を使用する場合の注意

Web コンテナにゲートウェイ情報を通知し，welcome ファイルや FORM 認証画面に正しくリダイレクトする**ゲートウェイ指定機能**を使用できます。ゲートウェイ指定機能については，「4.10　Web コンテナへのゲートウェイ情報の通知」を参照してください。

ゲートウェイ指定機能を使用する場合，一部のサーブレット API の動作が変わります。ゲートウェイ指定機能使用時のサーブレット API の注意事項を，使用するメソッドごとに示します。

- javax.servlet.http.HttpServletResponse クラスの sendRedirect メソッド

  引数に相対 URL を指定し，Host ヘッダのないリクエストの場合，リダイレクト先 URL のホスト名とポート番号は，ゲートウェイ指定機能で指定した値となります。引数に相対 URL を指定し，ゲートウェイ指定機能でスキームを https と見なすように設定した場合，リダイレクト先 URL のスキームは常に「https」となります。
- javax.servlet.ServletRequest インタフェースの getRequestURL メソッド

  ゲートウェイ指定機能でスキームを https と見なすように設定した場合，戻り値は常に「https://」で始まる URL となります。
- javax.servlet.ServletRequest インタフェースの getServerName メソッド

  ゲートウェイ指定機能でリダイレクト先 URL のホスト名を指定し，リクエストに Host ヘッダがない場合，戻り値は指定した値となります。

- javax.servlet.ServletRequest インタフェースの getServerPort メソッド
  ゲートウェイ指定機能でリダイレクト先 URL のポート番号を指定し，リクエストに Host ヘッダがない場合，戻り値は指定した値となります。ゲートウェイ指定機能でリダイレクト先 URL のホスト名を指定し，ポート番号を省略した場合，戻り値はリクエストのスキームが http であれば「80」，https であれば「443」となります。
- javax.servlet.ServletRequest インタフェースの getScheme メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合，戻り値は常に「https」となります。
- javax.servlet.ServletRequest インタフェースの isSecure メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合，戻り値は常に「true」となります。
- javax.servlet.ServletRequest インタフェースの getAttribute メソッド
  ゲートウェイ指定機能でスキームを https と見なすように設定した場合でも，次の属性は取得できません。
  - javax.servlet.request.cipher_suite（Web サーバに Microsoft IIS を使用している場合は，ゲートウェイ指定機能の使用に関係なく取得できません）
  - javax.servlet.request.key_size
  - javax.servlet.request.X509Certificate

(11)　ゲートウェイ使用時の注意

SSL アクセラレータや負荷分散器などのゲートウェイを使用した場合，次のサーブレット API の戻り値はクライアントの IP アドレスやホスト名ではなく，ゲートウェイの IP アドレスやホスト名になります。

- javax.servlet.ServletRequest クラスの getRemoteAddr メソッド
- javax.servlet.ServletRequest クラスの getRemoteHost メソッド

(12)　ServletContext オブジェクトに登録する製品独自の属性

Web コンテナは，Web アプリケーションの制御に必要な情報を javax.servlet.ServletContext オブジェクトの属性に登録しています。Web アプリケーションで ServletContext インタフェースの getAttributeNames メソッドによって取得する属性名には，Web コンテナによって登録された属性の名称も含まれます。

Web アプリケーション内で ServletContext オブジェクトに属性を登録する時，次の文字列で開始するキー名称を使用しないでください。

- org.apache.catalina
- com.hitachi.software.web
- jspx

また，ServletContext には Java EE の仕様で定められた属性も追加されるため，これらと同じキー名称の属性を登録しないでください。

(13)　ServletRequest クラスのプロキシ取得用メソッドを使用する場合の注意

次に示す javax.servlet.ServletRequest クラスのメソッドは，リクエストを送信したクライアント，または最後に通ったプロキシの情報を取得するためのメソッドですが，リバースプロキシを使用した環境では，取得する情報がリバースプロキシの情報となります。

- getRemoteAddr メソッド
- getRemoteHost メソッド
- getRemotePort メソッド

(14)　javax.servlet.ServletResponse インタフェースの reset メソッド実行時の注意

javax.servlet.ServletResponse インタフェースの getWriter メソッドを実行したあとに，reset メソッドを実行した場合，HTTP レスポンスの Content-Type で指定する文字エンコーディングは，次に示す API のどれか（すべて javax.servlet.ServletResponse インタフェース）を使用して，再度同じ文字エンコーディングを指定してください。

- setContentType メソッド
- setLocale メソッド
- setCharacterEncoding メソッド※

注※

Servlet 2.4 仕様で追加されたメソッドです。

Servlet 2.4 仕様以降では，これらの API を使用して文字エンコーディングを設定する場合は，getWriter メソッドを実行する前に呼び出す必要があります。ただし，getWriter メソッドを実行したあとに reset メソッドを実行した場合にかぎり，再度 getWriter メソッドを呼び出すまではこれらの API で文字エンコーディングを設定できます。

(15)　setMaxInactiveInterval メソッドの引数に 0 を指定した場合の動作

javax.servlet.http.HttpSession インタフェースの setMaxInactiveInterval メソッドの引数に 0 を指定した場合，セッションがタイムアウトになることはありません。

(16)　java.io.BufferedReader の mark 操作について

インプロセス HTTP サーバ機能を使用している場合に，javax.servlet.ServletRequest の getReader メソッドで得られる java.io.BufferedReader は，mark 操作をサポートしていません。markSupported メソッドでは false が返されます。

(17)　setVersion メソッドの引数に 1 を指定した場合の動作

javax.servlet.http.Cookie の setVersion メソッドの引数に 1 を指定した場合，Web サーバ連携機能使用時には，レスポンスに Set-Cookie2 ヘッダが付加されますが，インプロセス HTTP サーバ機能使用時には Set-Cookie ヘッダが付加されます。

(18)　getRequestDispatcher メソッドでのパスの指定について

javax.servlet.ServletRequest の getRequestDispatcher メソッドの引数に「/」ではじまらない相対パスを指定した場合，このサーブレットのサーブレットマッピングに指定した URL パターンからの相対パスになります。URL パターンが「/」で終わっている場合は，親のディレクトリからの相対パスになります。

例えば，サーブレットマッピングを"/a/b/"に指定したサーブレットから，"hello.html"を指定して getRequestDispatcher メソッドを実行すると，"/a/hello.html"が得られます。

(19)　setBufferSize メソッドを使用してバッファサイズを変更する場合の注意

レスポンス送信時に使用されるサーブレットのバッファは，リクエスト処理スレッドごとに保持されます。javax.servlet.ServletResponse の setBufferSize メソッドを実行してバッファサイズを変更した場合，変更したバッファサイズは，同一 J2EE サーバ上のほかの Web アプリケーションを含め，該当するスレッドが処理するすべてのリクエストに適用されます。javax.servlet.ServletResponse の setBufferSize メソッドを使用してバッファサイズを変更する場合は，（バッファサイズ）×（リクエスト処理スレッド数）分のメモリが確保されることを考慮した上で，メモリ使用量を見積もってください。

なお，一度確保されたバッファは，処理スレッドごとに Web アプリケーションから setBufferSize メソッドで更新されるまで有効となります。

(20)　HTTP レスポンスの Content-Type ヘッダについての注意

サーブレットでは，javax.servlet.ServletResponse クラスの setContentType メソッドで明示的に Content-Type を設定していない場合，Content-Type ヘッダを作成しません。そのため，HTTP レスポンスの文字エンコーディングを Content-Type ヘッダの”charset=”フィールドから確認することはできません。

(21)　javax.servlet.http.HttpSession クラスの getId メソッドについての注意事項

Servlet 2.4 仕様以前に準拠した Web アプリケーションで，無効化された javax.servlet.http.HttpSession オブジェクトの getId メソッドを呼び出した場合の動作が Servlet 仕様とアプリケーションサーバとで異なります。それぞれの動作を次に示します。

Servlet 仕様

　java.lang.IllegalStateException 例外をスローする。

アプリケーションサーバ

　null を返す。

(22)　javax.servlet.ServletRequest クラスおよび javax.servlet.http.HttpServletRequest クラスのメソッドについての注意事項

次の表に示すメソッドで取得した情報をレスポンスに出力する場合は，サニタイズする必要があります。

表 6–13　取得した情報をサニタイズする必要があるメソッド

クラス名	メソッド名
javax.servlet.ServletRequest	getCharacterEncoding()
	getContentType()
	getInputStream()
	getParameter(java.lang.String name)
	getParameterMap()
	getParameterNames()
	getParameterValues(java.lang.String name)
	getProtocol()
	getReader()

クラス名	メソッド名
javax.servlet.ServletRequest	getServerName()
javax.servlet.http.HttpServletRequest	getCookies()
	getHeader(java.lang.String name)
	getHeaderNames()
	getHeaders(java.lang.String name)
	getMethod()
	getPathInfo()
	getPathTranslated()
	getQueryString()
	getRequestedSessionId()
	getRequestURI()
	getRequestURL()
	getServletPath()

(23)　javax.servlet.ServletRequest クラスの getLocale(getLocales)メソッドについての注意事項

javax.servlet.ServletRequest クラスの getLocale メソッド，または getLocales メソッドで取得できる java.util.Locale オブジェクトは，HTTP リクエストの Accept-Language ヘッダの値から作成します。

Web コンテナでは，Accept-Language ヘッダの値のロケール（ISO 言語コード，ISO 国コード，またはバリアント）が英字以外を含むかどうかをチェックします。ロケールが英字以外を含む場合は，不正なロケールと判断して，不正なロケールごとに KDJE39546-W のメッセージをメッセージログに出力して無視します。

不正なロケールを含む Accept-Language ヘッダを受信した場合は，getLocale メソッド，または getLocales メソッドは正しいロケールの java.util.Locale オブジェクトだけを返します。Accept-Language ヘッダに指定されたロケールがすべて不正な場合は，Accept-Language ヘッダがないと見なしてサーバのデフォルトロケールを返します。

(24)　javax.servlet.http.HttpServletRequest インタフェースの getServerName メソッドの戻り値

javax.servlet.http.HttpServletRequest インタフェースの getServerName メソッドの戻り値は，HTTP Server やリバースプロキシなどで Host ヘッダを書き換えた場合，HTTP クライアントが設定した Host ヘッダの値と異なることがあります。

(25)　Servlet 2.4 以前の include 先のサーブレットでのレスポンスヘッダの設定

Servlet 2.4 以前では，include 先のサーブレットでのレスポンスヘッダの設定はすべて無視される仕様です。ただし，アプリケーションサーバでは，Servlet 2.4 を使用した場合でも，getSession でのレスポンスヘッダの設定は有効になります。

(26)　MIME マッピングがない静的コンテンツのコンテンツ形式

MIME マッピングがない静的コンテンツの場合，Content-Type は付与されません。

(27)　HTTP セッションのアクセス時刻について

セッション ID の付加されたリクエストが Web コンテナに送信されたとき，HTTP セッションのアクセス時刻は現在時刻で更新されます。ただし，すでにセッションがタイムアウトしたり，無効化したりしている場合には該当しません。

HTTP セッションのアクセス時刻は次で使用されます。

- javax.servlet.http.HttpSession インタフェースの getLastAccessedTime()メソッドの戻り値
  javax.servlet.http.HttpSession インタフェースの getLastAccessedTime()メソッドは，前回更新時の HTTP セッションのアクセス時刻を返します。
- HTTP セッションのタイムアウト
  現在時刻と HTTP セッションのアクセス時刻の差がタイムアウト時間を超えたとき，HTTP セッションはタイムアウトします。なお，HTTP セッションのタイムアウト監視は 30 秒間隔で実行されるため，正確なタイムアウト時間にタイムアウトが発生するわけではありません。

(28)　Content-Length ヘッダの値取得時の注意事項

HTTP リクエストに Content-Length ヘッダが含まれていない場合，javax.servlet.ServletRequest の getContentLength()メソッドの戻り値，および javax.servlet.http.HttpServletRequest の getIntHeader()メソッドで引数に"Content-Length"を指定した場合の戻り値が Servlet 仕様とアプリケーションサーバとで異なります。それぞれの戻り値を次に示します。

Servlet 仕様
　-1 を返す。

アプリケーションサーバ
　0 を返す。

6.2.3　Servlet 3.0 仕様で追加，変更された仕様についての注意事項

Servlet 3.0 仕様で追加，変更された仕様を，アプリケーションサーバ上で使用するときの注意事項を示します。Servlet 3.0 仕様および Servlet 2.5 仕様については，それぞれの仕様書（Servlet 3.0 仕様書，Servlet 2.5 仕様書）を参照してください。

(1)　Servlet 3.0 とアプリケーションサーバの機能を組み合わせた場合の仕様

Servlet 3.0 で追加された機能と，アプリケーションサーバの機能を組み合わせた場合の仕様について説明します。

(a)　Web プリケーションのバージョン設定機能

Web アプリケーションのバージョン設定機能で，バージョンとして 3.0 は設定できません。

(b)　META-INF/resources 以下の JSP 事前コンパイル

Servlet 3.0 仕様では，JSP ファイルを JAR ファイルの中の META-INF/resources の階層に含めることができるようになりましたが，META-INF/resources 内に JSP を含む Web アプリケーションの場合，JSP

の事前コンパイルは実行できません。JSP 事前コンパイルを実行した場合，META-INF/resources 内の JSP は事前コンパイルされないで，リクエスト実行時にエラーとなります。

(c)　Servlet 3.0 を使用したアプリケーションのリロード

Servlet 3.0 仕様の機能を使用したアプリケーションをリロードする場合，次の制約があります。

META-INF/resources に静的コンテンツ（JSP も含む）を持つ JAR ファイルを含むアプリケーションの場合，JAR ファイルを更新してリロードしても，JAR ファイルの META-INF/resources 内の静的コンテンツは更新前のものにしかアクセスできません。META-INF/resources に静的コンテンツを持つ JAR ファイルを含むアプリケーションをリロードした場合，KDJE39556-W が出力されます。

! 注意事項

09-00-02 より前のバージョンの場合，次のアプリケーションをリロードしたときの動作は保証されません。

- ServletContainerInitializer インタフェースの実装クラスを含むアプリケーションの場合。
  リロードした場合，KDJE39557-W が出力されます。
- 動的サーブレット定義で追加したサーブレット，フィルタまたはリスナを含むアプリケーションの場合。
  リロードした場合，KDJE39558-W が出力されます。

(d)　API で定義したサーブレットでの run-as 機能

API で定義したサーブレットで run-as 機能を使用できません。次の設定をした場合，無視されます。

- サーブレットのクラスで@RunAs アノテーションを使用する。
- ServletRegistration.Dynamic インタフェースの setRunAsRole メソッドを呼び出す。

(e)　API で定義したサーブレットまたはフィルタの実行時間の監視

API で定義したサーブレットまたはフィルタでは，J2EE アプリケーションの実行時間の監視機能を使用できません。cosminexus.xml に<method-observation-timeout>タグが設定されていても無視されます。

(f)　Web アプリケーションの呼び出し時の HTTP セッションの継続

Web アプリケーションの呼び出し時の HTTP セッションの継続については，Servlet 2.5 仕様の場合と同じ動作となります。詳細は，「6.2.4(6)　Web アプリケーションの呼び出し時の HTTP セッションの継続について」を参照してください。

(g)　セッションのクッキー名を変更した場合のリダイレクタでのリクエストの振り分け

Web コンテナがクラスタ構成で配置される場合に，リダイレクタを使用してリクエストを振り分けることができます。リダイレクタは，各リクエストに付加されたセッション ID を参照して，同一の Web クライアントから来たリクエストが同一の Web コンテナに転送されるように，リクエストを振り分けます。ただし，セッションのクッキー名をデフォルトの「JSESSIONID」からほかの名称に変更した場合，同一の Web クライアントから来たリクエストを同一の Web コンテナに転送する動作は保証されません。

(h)　アノテーション参照抑止機能

Servlet3.0 で追加されたアノテーションのうち，@HandlesTypes アノテーションはアノテーション参照抑止機能の対象になりません。アノテーション参照抑止機能が有効な場合でも処理されます。

(2)　Servlet 3.0 と CDI を組み合わせた場合について

Servlet 3.0 と CDI を組み合わせて使用する場合，Servlet 3.0 に対応した web.xml を使用してください。これ以外のバージョンの web.xml と CDI を組み合わせて使用できません。

(3)　API で定義したフィルタまたはリスナと，@PostConstruct アノテーションまたは @PreDestroy アノテーションを組み合わせた場合について

API で定義したフィルタまたはリスナでは，@PostConstruct アノテーションまたは@PreDestroy アノテーションを使用できません。これらのアノテーションを使用した場合，無視されます。

(4)　インクルード時に対象のコンテンツがない場合について

Servlet 3.0 仕様では，インクルードの対象がデフォルトサーブレット（静的コンテンツを提供するためにコンテナが用意しているサーブレット）で，対象のコンテンツがない場合，FileNotFoundException 例外をスローすることになっています。また，その例外がユーザプログラムで処理されなかった場合，レスポンスはコミットされないで，ステータスコード 500 を返すことなっています。

アプリケーションサーバでは，この仕様はサポートされません。08-70 以前のバージョンと同様にステータスコード 404 が返されます。

(5)　エスケープシーケンスの使用について

Servlet 3.0 で新規に追加された API やプロパティの入力文字に，エスケープシーケンス（¥b，¥t,，¥n，¥f，¥r，¥"，¥'，¥¥）を使用しないでください。使用した場合，ログが途中で改行されるなど，表示が不正になることがあります。

(6)　動的サーブレット定義について

動的サーブレット定義について，アプリケーションサーバで使用した場合に Servlet 3.0 仕様と異なる点について説明します。

(a)　サーブレット，フィルタ，またはリスナの定義

- API および web.xml の両方でフィルタを追加した場合に，API で定義されたほかのフィルタマッピングの isMatchAfter メソッドが false に設定されているときは，リクエスト処理の中で API の定義が先に呼ばれます。
- API および web.xml の両方でリスナを定義した場合，リスナの生成のときはアプリケーションの開始時に web.xml で定義されたリスナが先に読み込まれます。リスナを削除するときは，API の定義が先に読み込まれます。
- URL パターンに空文字（""）を指定して javax.servlet.ServletRegistration インタフェースの addMapping(String... urlPatterns)メソッドを呼び出した場合，サーブレットはコンテキスト名だけでアクセスされます。コンテキスト名も空文字（""）の場合は，サーブレットにアクセスできません。
- すでに web.xml で定義された名称，または@WebServlet アノテーションでサーブレットクラスが生成された名称と同じサーブレット名を指定して，addServlet()メソッドを呼び出さないでください。
- ユーザアプリケーションの中で，同じサーブレットに対して setMultipartConfig()メソッドを呼び出し，@MultipartConfig アノテーションを指定した場合，setMultipartConfig()メソッドで指定した値が使用されます。

  同様に，ユーザアプリケーションの中で，同じサーブレットに対して setServletSecurity()メソッドを呼び出し，@ServletSecurity アノテーションを指定した場合，setServletSecurity()メソッドで指定した値が使用されます。

- getRunAsRole()メソッドおよび setRunAsRole(String roleName)メソッドを使用して，ロールの取得および設定はできません。また，ワーニングメッセージまたはエラーメッセージも出力されません。
- すでに登録されている名称でサーブレットまたはフィルタを登録した場合，アプリケーションの開始時に KDJE39552-W が出力され，処理は続行されます。
- URL パターンに改行コードが含まれていた場合，アプリケーションの開始時に KDJE39555-W が出力され，処理は続行されます。

(b)　ServletContainerInitializer インタフェースを使用した定義

アプリケーションサーバでの ServletContainerInitializer インタフェースを使用した定義について，Servlet 3.0 仕様と次の点が異なります。

- ServletContainerInitializer インタフェースを実装したクラスを含む JAR ファイルは次の場所に配置できます。

  1. アプリケーションに含まれる WAR ファイル内の WEB-INF/lib ディレクトリ
  2. WEB-INF/lib ディレクトリ以外の場所

  WEB-INF/lib ディレクトリ以外の場所に，ServletContainerInitializer インタフェースを実装したクラスを含む JAR ファイルを配置する場合，JAR ファイルの絶対パスを usrconf.properties の次のプロパティに指定します。

  ```
  webserver.ServletContainerInitializer_jar.include.path
  ```

  設定例

  ```
  webserver.ServletContainerInitializer_jar.include.path=C:/Program Files/xxx/foo/lib/
  bar.jar
  ```

- @HandlesTypes アノテーションのクラスの検索範囲は，ServletContainerInitializer インタフェースの実装クラスを含む WAR の中です。WAR の WEB-INF/classes ディレクトリにあるクラス，および WEB-INF/lib ディレクトリにある JAR ファイルが対象となります。
- ServletContainerInitializer インタフェースを実装したクラスを含む JAR ファイルが読み込めないか処理できない場合，アプリケーションの開始時に KDJE39548-E が出力され，アプリケーションの開始に失敗します。
- @HandlesTypes アノテーションで指定したクラスを継承(extends)や実装(implements)したクラス，または@HandlesTypes アノテーションで指定したクラスのアノテーションを付けているクラスを検索する際に，クラスのロードに失敗した場合，KDJE39549-W が出力され，アプリケーションの開始処理は続行されます。
- webserver.ServletContainerInitializer_jar.include.path に指定した，ServletContainerInitializer インタフェースを実装したクラスを含む JAR ファイルが見つからない場合，または JAR ファイルもしくはファイルが読み込めない場合，アプリケーションの開始時に KDJE39553-W が出力され，処理は続行されます。このメッセージは webserver.ServletContainerInitializer_jar.include.path に指定された JAR ファイルごとに出力されます。
- webserver.ServletContainerInitializer_jar.include.path に指定した，ServletContainerInitializer インタフェースを実装したクラスを含む JAR ファイルがクラスパスの中に見つからない場合，アプリケーションの開始時に KDJE39554-W が出力され，処理は続行されます。このメッセージは webserver.ServletContainerInitializer_jar.include.path に指定された JAR ファイルごとに出力されます。

また，javax.servlet.ServletContainerInitializer ファイルに定義する，ServletContainerInitializer インタフェースの実装クラスの情報は，次のように読み込まれます。

- 1 行目に記述されたクラスだけが読み込まれ，2 行目以降に記述されたクラスは無視されます。

(7)　ファイルアップロードについて

ファイルアップロードについて，アプリケーションサーバで使用した場合に Servlet 3.0 仕様と異なる点について説明します。

- web.xml および@MultipartConfig アノテーションの両方で multipart config 要素を設定した場合，web.xml の定義が優先されます。
- リクエストタイプが multipart/form-data 以外のリクエストでファイルアップロードを実施した場合，javax.servlet.ServletException 例外がスローされます。ファイルアップロードを実施するリクエストのタイプと実行結果を次の表に示します。

表 6-14　ファイルアップロードを実施するリクエストのタイプと実行結果

web.xml または@MultipartConfig アノテーションを指定したリクエストのタイプ	実行結果
mime-multipart 以外	javax.servlet.ServletException 例外がスローされます。
multipart/form-data 以外の mime-multipart	javax.servlet.ServletException 例外がスローされます。
multipart/form-data	ファイルアップロードが実行されます。例外は発生しません。

- MultipartConfigElement がプログラムで定義された場合，web.xml で定義された場合と同じ動作になります。ただし，空文字（""）が location に指定された場合，ファイルが Web アプリケーションの作業ディレクトリに格納されます。
- アップロードされたファイルが，web.xml の<file-size-threshold>タグまたは@MultipartConfig アノテーションの fileSizeThreshold 属性で指定した値よりも大きいサイズの場合，リクエスト内のオブジェクトの一部分に対応する一時ファイルが生成されます。一時ファイルの名称は，「upload」から始まる，一意に識別される名称で，拡張子は「.tmp」です。

  例：upload__5f8d5c62_1316c5ef08b__8000_00000012.tmp

  一時ファイルはリクエストの処理の最後に削除されます。ただし，一時ファイルが開かれたままの場合，J2EE サーバが正常に停止されなかった場合，システムダウンした場合など，一時ファイルが削除されないケースがあります。このような場合は，location で指定した場所から一時ファイルを手動で削除する必要があります。

  ユーザが手動で作成したファイルと同じ名称の一時ファイルが生成された場合は，ユーザが作成したファイルが上書きされます。
- Part オブジェクトはリクエスト内で利用できます。Part オブジェクトはリクエストが完了した時点で削除されます。あとで利用するためにセッションの中で Part オブジェクトを保持しても，動作は保証されません。

(8)　静的リソースの配置について

静的リソースの配置について，アプリケーションサーバで使用した場合に Servlet 3.0 仕様と異なる点について説明します。

- 静的リソースが含まれる JAR ファイルが読み込めないか処理できない場合，アプリケーションの開始時に KDJE39548-E が出力され，アプリケーションの開始に失敗します。

- 指定した JAR ファイルに形式不正があった場合，アプリケーションの開始時に KDJE39550-E が出力され，処理は続行されます。
- WEB-INF/lib に不正な JAR ファイル，または読み込めないファイルが含まれていた場合，アプリケーションの開始時に KDJE39551-W が出力され，その JAR ファイルを無視して処理が続行されます。

(9)　API について

API について，アプリケーションサーバで使用した場合に Servlet 3.0 仕様と異なる点について説明します。

- アプリケーションサーバでは，Servlet 3.0 仕様で追加された API のうち，次の表に記載されていない API を使用できます。API の詳細については，Servlet 3.0 の仕様書を参照してください。
  次の表に記載された非サポートの API を使用した場合，java.lang.UnsupportedOperationException 例外がスローされます。

表 6–15　非サポートの API（Servlet 3.0）

項番	パッケージ	クラス	インタフェース/クラス	メソッド	機能
1	javax.servlet	AsyncContext	インタフェース	すべてのメソッド	非同期サーブレット
2		AsyncListener	インタフェース	すべてのメソッド	非同期サーブレット
3		ServletContext	インタフェース	getJspConfigDescriptor	JSP
4		ServletRequest	インタフェース	startAsync	非同期サーブレット
5				isAsyncStarted	非同期サーブレット
6				isAsyncSupported	非同期サーブレット
7				getAsyncContext	非同期サーブレット
8				getDispatcherType	非同期サーブレット
9		AsyncEvent	クラス	すべてのメソッド	非同期サーブレット
10		ServletRequestWrapper	クラス	startAsync	非同期サーブレット
11				isAsyncStarted	非同期サーブレット
12				isAsyncSupported	非同期サーブレット
13				getAsyncContext	非同期サーブレット
14				getDispatcherType	非同期サーブレット
15		Registration	インタフェース	setAsyncSupported	非同期サーブレット
16	javax.servlet.descriptor	JspConfigDescriptor	インタフェース	すべてのメソッド	JSP
17		JspPropertyGroupDescriptor	インタフェース	すべてのメソッド	JSP

項番	パッケージ	クラス	インタフェース/クラス	メソッド	機能
18	javax.servlet.descriptor	TaglibDescriptor	インタフェース	すべてのメソッド	JSP

- Servlet 2.5 仕様に対応する Web アプリケーションで，Servlet 3.0 で追加された API は使用できません。使用した場合，チェックされないため動作は保証されません。
- SessionTrackingMode では，SSL は使用できません。アプリケーションサーバでは COOKIE と URL だけ使用できます。

(10)　web.xml の省略について

Servlet 3.0 では，次のアプリケーションの場合，web.xml を省略できます。

- 静的コンテンツおよび JSP だけを含む Web アプリケーション（サーブレット，フィルタ，およびリスナを含まない）
- サーブレット，フィルタ，リスナをアノテーションで定義した Web アプリケーション

(11)　セッション ID を示す HTTP Cookie 名の変更について

セッション ID を示す HTTP Cookie 名を変更する場合，Cookie 名に「csfcfc」を指定できません。

また，使用できる文字に次の条件があります。条件に違反した場合，KDJE39559-W が出力されます。セッションの不正な動作の原因になることがあるため，条件に合った文字を使用してください。

- ASCII 文字を使用してください。ただし，次の文字は使用できません。
  「#」，「(」，「)」，「<」，「>」，「@」，「,」，「;」，「:」，「¥」，「/」，「"」，「[」，「]」，「?」，「=」，「{」，「}」
- 空白，および制御文字は使用できません。
- 文字列の先頭に「$」は使用できません。

(12)　Web アプリケーションで使用できる HTTP Cookie 名

Web アプリケーションで HTTP Cookie を使用する場合，次の名称を指定できません。なお，大文字と小文字は区別されません。

- HTTP セッションのセッション ID で使用する HTTP Cookie の名称。デフォルトは「JSESSIONID」。
- グローバルセッションを表す HTTP Cookie の名称。デフォルトは「GSESSIONID」。
- J2EE サーバのサーバ ID 付加機能によってサーバ ID を付加した HTTP Cookie の名称と同じ名称。
- J2EE サーバ内部で使用する HTTP Cookie の名称「csfcfc」。

(13)　<servlet-class>要素に指定するクラスについて

web.xml の<servlet-class>要素を省略する場合，@WebServlet アノテーションを指定した javax.servlet.http.HttpServlet のサブクラスを Web アプリケーションに含め，name 属性に<servlet-class>要素を省略したサーブレットのサーブレット名を設定する必要があります。該当する HttpServlet クラスがない場合は，KDJE39339-E が出力され，サーブレットクラスのロードに失敗します。

(14)　<filter-class>要素に指定するクラスについて

web.xml の<filter-class>要素を省略する場合，@WebFilter アノテーションを指定した javax.servlet.Filter の実装クラスを Web アプリケーションに含め，filterName 属性に<filter-class>要素を省略したフィルタのフィルタ名を設定する必要があります。該当する Filter クラスがない場合は，KDJE39340-E が出力され，アプリケーションの開始に失敗します。

(15)　セッション ID を示す HTTP Cookie の Path 属性の変更について

セッション ID を示す HTTP Cookie の Path 属性を変更する場合，別の Web アプリケーションとセッション ID を示す HTTP Cookie 名や Path 属性が重複すると，HTTP Cookie のセッション ID を示す値が上書きまたは削除されるなどの理由で，HTTP セッションが正しく引き継がれなくなることがあります。

(16)　セッション ID を示す HTTP Cookie の Path 属性と Domain 属性に使用できる文字について

セッション ID を示す HTTP Cookie の Path 属性と Domain 属性に使用できる文字には次の条件があります。条件に違反した場合，KDJE39559-W が出力されます。セッションの不正な動作の原因になることがあるため，条件に合った文字を使用してください。

- Path 属性には，ASCII 文字だけが使用できます。ただし，制御文字およびセミコロン（;）は使用できません。
- Domain 属性には，英数字，ハイフン（-）およびピリオド（.）だけが使用できます。

6.2.4　Servlet 2.5 仕様で追加，変更された仕様についての注意事項

Servlet 2.5 仕様で追加，変更された仕様を，アプリケーションサーバ上で使用するときの注意事項を示します。Servlet 2.5 仕様および Servlet 2.4 仕様については，それぞれの仕様書（Servlet 2.5 仕様書，Servlet 2.4 仕様書）を参照してください。

(1)　web.xml の XML スキーマ変更について

アプリケーションサーバでは，Servlet 2.5 仕様に従って web.xml の XML スキーマが変更されています。

(2)　web.xml の省略に関する注意事項

Servlet 2.5 仕様では，web.xml の省略について記述が追加されています。

アプリケーションサーバでは，Web アプリケーションが JSP および静的コンテンツしか含まない場合，web.xml を省略できます。web.xml が省略された Web アプリケーションのバージョンは 2.5 と見なされます。また，web.xml を省略した場合，WEB-INF ディレクトリを省略できます。

ここでは web.xml 省略時の注意事項について説明します。web.xml の省略については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「11.4.5　web.xml を省略した Web アプリケーションに対する操作」を参照してください。

(a)　Web アプリケーションにサーブレットおよびフィルタが含まれている場合の web.xml の省略

サーブレット，およびフィルタは web.xml にマッピング定義が必要なため，web.xml を省略すると動作しません。

(b)　Web アプリケーションにリスナが含まれている場合の web.xml の省略

次に示すリスナは web.xml に定義が必要なため，web.xml を省略すると動作しません。

- javax.servlet.ServletContextListener
- javax.servlet.ServletContextAttributeListener
- javax.servlet.ServletRequestListener
- javax.servlet.ServletRequestAttributeListener
- javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSessionAttributeListener

ただし，次に示すリスナは web.xml に定義しないため，web.xml を省略しても動作します。

- javax.servlet.http.HttpSessionBindingListener
- javax.servlet.http.HttpSessionActivationListener

また，タグライブラリでリスナが定義されている場合，web.xml を省略してもリスナは実行されます。

(c)　実行環境での Web アプリケーションのプロパティの設定

web.xml を省略した場合，サーバ管理コマンドおよび属性ファイルを使用したプロパティの設定はできません。

(d)　マッピングの設定

web.xml を省略した場合でも，拡張子が jsp および jspx のファイルのマッピング，デフォルトの welcome ファイル，セッションタイムアウト，およびデフォルトの mime-mapping の設定は有効です。サーブレットのデフォルトマッピングの有効または無効の設定に関係なく，サーブレットのデフォルトマッピングは使用できません。

(e)　フィルタの使用

web.xml を省略した Web アプリケーションではフィルタを使用できません。そのため，HTTP レスポンス圧縮機能やメモリセッションフェイルオーバ機能など，built-in フィルタが必要な機能は使用できません。

(3)　<load-on-startup>要素への空文字列の指定

Servlet 2.5 仕様で定義された web.xml のスキーマでは，<load-on-startup>要素に指定できる値が変更されました。

アプリケーションサーバでは，<load-on-startup>要素に指定できる値は web.xml に記述する Servlet 仕様のバージョン情報に対応したサーブレットに従って制御されます。サーブレットのバージョンごとに<load-on-startup>要素に指定できる値について次の表に示します。

表 6-16　<load-on-startup>要素に指定できる値

サーブレットのバージョン	指定できる値
Servlet 2.5	整数，または空文字列。空文字列が指定された場合，<load-on-startup>要素が指定されなかった場合と同様に，サーブレットのロードを行わない。
Servlet 2.4	整数。07-60 と同様の指定値。

なお，指定できない値を指定した場合，J2EE アプリケーションのインポートに失敗します。

(4)　<security-constraint>要素での HTTP メソッドの指定範囲について

Servlet 2.5 仕様で定義された web.xml のスキーマでは，<security-constraint><web-resource-collection>要素内の<http-method>要素に指定できる内容について変更されました。指定できる内容について次の表に示します。

表 6-17　<security-constraint><web-resource-collection>要素内の<http-method>要素に指定できる内容

サーブレットのバージョン	指定できる内容
Servlet 2.5	半角英数字（0-9，A-Z，a-z）および特殊文字（! # $ % & ' * + - . ^ _ ` | ~）を 1 回以上。
Servlet 2.4	GET，POST，PUT，DELETE，HEAD，OPTIONS，TRACE のどれか。

<security-constraint><web-resource-collection>要素内の<http-method>要素に指定するのは，リクエストの HTTP メソッドです。

アプリケーションサーバでサポートするリクエストの HTTP メソッドは，web.xml に記述する Servlet 仕様のバージョン情報に対応したサーブレットでサポートする内容と同じです。ただし，Servlet 2.5 仕様に準拠した Web アプリケーションでサポートするリクエストの HTTP メソッドは，使用する Web サーバによって異なります。アプリケーションサーバでサポートするリクエストの HTTP メソッドについて，Web サーバごとに次に示します。

表 6-18　アプリケーションサーバでサポートするリクエストの HTTP メソッド

サーブレットのバージョン	使用する Web サーバ	指定できる内容
Servlet 2.5	HTTP Server および Microsoft IIS	HTTP1.1 で使用可能なメソッドの一部※。
	インプロセス HTTP サーバ	HTTP1.1 で使用可能なすべてのメソッド。
Servlet 2.4	HTTP Server，Microsoft IIS，インプロセス HTTP サーバ	GET，POST，PUT，DELETE，HEAD，OPTIONS，TRACE のどれか。

注※　指定できるメソッドの詳細については「付録 A.2　リダイレクタが返すエラーステータスコード」を参照してください。

指定できない値を指定した場合，J2EE アプリケーションのインポートに失敗します。

(5)　アノテーションの使用

アプリケーションサーバでは，Servlet 2.5 仕様で規定されたアノテーションをサポートします。アノテーションの使用については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「12.　アノテーションの使用」を参照してください。

(6)　Web アプリケーションの呼び出し時の HTTP セッションの継続について

Servlet 2.5 仕様では，クロスコンテキストで呼び出されたサーブレットおよび JSP 内でセッションを作成した場合，次のどちらの場合もセッションが継続するように記述が追加されています。

- 関連づけられたコンテキストに直接リクエストが来た場合のクロスコンテキストの呼び出し

- ディスパッチ（javax.servlet.RequestDispatcher インタフェースの forward メソッドおよび include メソッド呼び出し）によるクロスコンテキストの呼び出し

しかし，アプリケーションサーバでは，Web アプリケーションの呼び出し時の HTTP セッションの継続はできません。

(7)　フィルタマッピングでのすべてのサーブレットの指定について

Servlet 2.5 仕様では，web.xml の<filter-mapping>要素内の<servlet-name>要素での*（半角アスタリスク）の指定について記述が追加されています。

web.xml の<filter-mapping>要素内の<servlet-name>要素で*を指定することで，すべてのサーブレットへのリクエストをフィルタ呼び出しの対象にできます。

web.xml での定義例を次に示します。

```
<filter-mapping>
    <filter-name>All Filter</filter-name>
    <servlet-name>*</servlet-name>
</filter-mapping>
```

この例では，すべてのサーブレットへのリクエストに対してフィルタ名「All Filter」のフィルタが呼び出されます。

! 注意事項

Servlet 2.4 仕様に準拠した Web アプリケーションではフィルタマッピングで指定する*（半角アスタリスク）に特別な意味はありませんでした。しかし，Servlet 2.5 仕様に準拠した Web アプリケーションでは*はすべてのサーブレットを意味します。サーブレット名が*のサーブレットだけを指定する方法はないため注意してください。

アプリケーションサーバでは，web.xml の<filter-mapping><servlet-name>要素で*（半角アスタリスク）を指定した場合，該当する web.xml を含む Web アプリケーションへのすべてのリクエストがフィルタ呼び出しの対象となります。

(8)　javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッドの呼び出しの無効について

Servlet 2.5 仕様では，javax.servlet.ServletRequest インタフェースの getReader メソッドが呼び出されたあと，setCharacterEncoding メソッドの呼び出しが無効になることについて明記されました。

アプリケーションサーバでは，サーブレットのバージョンに関係なく，Servlet 2.5 仕様に従います。getReader メソッド呼び出し後の setCharacterEncoding メソッドの呼び出しは無効となり，次の内容は変更されません。

- getReader メソッドで取得した BufferedReader オブジェクトが使用する文字エンコーディング
- getCharacterEncoding メソッドの戻り値

07-60 以前は，getReader メソッド呼び出し後の setCharacterEncoding メソッドの呼び出しによって getCharacterEncoding メソッドの戻り値が変更されていましたが，08-00 では，サーブレットのバージョンに関係なく，変更されません。

(9)　javax.servlet.http.HttpServletRequest インタフェースの getRequestURL メソッドの戻り値となる URL の変更について

Servlet 2.5 仕様では，forward 先で javax.servlet.http.HttpServletRequest インタフェースの getRequestURL メソッドが呼び出された場合に戻り値となる URL について，クライアントに指定された URL のパスではなく，javax.servlet.RequestDispatcher オブジェクトを取得するために使用したパスが反映されることが明記されました。

アプリケーションサーバでは，web.xml に記述する Servlet 仕様のバージョン情報に対応したサーブレットに従って，戻り値となる URL を決定します。

Servlet 2.5 仕様では，javax.servlet.RequestDispatcher オブジェクトを取得するために呼び出した javax.servlet.ServletRequest インタフェースの getRequestDispatcher メソッドの引数に指定したパスが戻り値となります。Servlet 2.4 仕様では，リクエストの URL のパスが戻り値となります。

(10)　javax.servlet.http.HttpSession インタフェースの getId メソッドが呼び出された場合の動作について

Servlet 2.5 仕様では，無効化された javax.servlet.http.HttpSession オブジェクトの getId メソッドが呼び出された場合に java.lang.IllegalStateException 例外がスローされる仕様が削除されました。

アプリケーションサーバでは，無効化された javax.servlet.http.HttpSession オブジェクトの getId メソッドが呼び出された場合の戻り値は，web.xml に記述する Servlet 仕様のバージョン情報に対応したサーブレットに従って制御されます。

Servlet 2.5 仕様の場合はセッション ID，Servlet 2.4 仕様の場合は 07-60 と同様に null が戻り値となります。

(11)　サーブレットのバージョンが異なる Web アプリケーション間でのクロスコンテキストの呼び出しについて

サーブレットのバージョンが異なる Web アプリケーション間でクロスコンテキストを呼び出した場合，呼び出し元と呼び出し先のサーブレットのバージョンによって，呼び出し先の動作が異なるメソッドがあります。サーブレットのバージョンが異なる Web アプリケーション間でのクロスコンテキストの呼び出しをした場合のクロスコンテキストの呼び出し先の動作について，メソッドごとに表に示します。

表 6-19　サーブレットのバージョンが異なる Web アプリケーション間でのクロスコンテキストの呼び出し先の動作（javax.servlet.http.HttpServletRequest インタフェースの getRequestURL メソッド）

Web アプリケーションのサーブレットのバージョン		クロスコンテキストの呼び出し先の動作	参照先
呼び出し元	呼び出し先		
Servlet 2.5	Servlet 2.4 以前	Servlet 2.5 仕様に従う。	6.2.4(9)
Servlet 2.4 以前	Servlet 2.5	Servlet 2.4 仕様に従う。	

表 6-20　サーブレットのバージョンが異なる Web アプリケーション間でのクロスコンテキストの呼び出し先の動作（無効化された javax.servlet.http.HttpSession オブジェクトの getId メソッド）

Web アプリケーションのサーブレットのバージョン		クロスコンテキストの呼び出し先の動作	参照先
呼び出し元	呼び出し先		
Servlet 2.5	Servlet 2.4 以前	Servlet 2.5 仕様に従い，セッション ID を返す。	6.2.4(10)
Servlet 2.4 以前	Servlet 2.5	Servlet 2.4 仕様に従い，null を返す。	

(12)　include 先のサーブレットでのレスポンスヘッダの設定について

Servlet 2.5 仕様では，include 先のサーブレットでのレスポンスヘッダの設定について，getSession の場合だけ設定が有効となります。ただし，アプリケーションサーバでは，Servlet 2.4 を使用した場合でも，getSession でのレスポンスヘッダの設定は有効になります。

6.2.5　Servlet 2.4 仕様で追加，変更された仕様についての注意事項

Servlet 2.4 仕様で追加，変更された仕様を，アプリケーションサーバ上で使用するときの注意事項を示します。Servlet 2.4 仕様および Servlet 2.3 仕様については，それぞれの仕様書（Servlet 2.4 仕様書，Servlet 2.3 仕様書）を参照してください。

(1)　X-Powered-By ヘッダの利用

Servlet 2.4 仕様で追加された X-Powered-By ヘッダはレスポンスに追加されません。

(2)　java.servlet.RequestDispatcher クラスの forward メソッド利用時の注意

javax.servlet.ServletRequest クラス，および javax.servlet.ServletContext クラスの getRequestDispatcher メソッドで取得した javax.servlet.RequestDispatcher クラスの forward メソッドを実行すると，リクエストオブジェクトには次のキーの属性が追加されます。ただし，javax.servlet.ServletContext クラスの getNamedDispatcher メソッドで取得した RequestDispatcher オブジェクトの forward メソッドでは追加されません。

- javax.servlet.forward.request_uri
- javax.servlet.forward.context_path
- javax.servlet.forward.servlet_path
- javax.servlet.forward.path_info※1
- javax.servlet.forward.query_string※2

注※1

Web コンテナが受信した HTTP リクエストが追加のパス情報を含まない場合，この属性は追加されません。

注※2

Web コンテナが受信した HTTP リクエストのリクエスト URI がクエリ文字列を含まない場合，この属性は追加されません。

これらの属性は，Web コンテナによって追加されます。javax.servlet.ServletRequestAttributeListener に属性追加のイベントは通知されません。追加される属性の値については，Servlet 2.4 仕様書を参照してください。

(3)　javax.servlet.SingleThreadModel インタフェースの非推奨化について

javax.servlet.SingleThreadModel インタフェースは，Servlet 2.4 仕様から非推奨となっています。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，javax.servlet.SingleThreadModel インタフェースを使用できます。ただし，Servlet 2.4 仕様を参照し，非推奨となった理由に注意して使用してください。

(4)　javax.servlet.ServletResponse クラスの setLocale メソッドについて

javax.servlet.ServletResponse クラスの setLocale メソッドによって，HTTP レスポンスの Content-Type ヘッダに文字エンコーディングが設定されます。Servlet 2.4 仕様では，設定される文字エンコーディングが有効となる条件が変更されています。

アプリケーションサーバで有効となる条件を，Servlet 2.4 以降と Servlet 2.3 に分けて示します。

Servlet 2.4 以降

次のすべての条件を満たす必要があります。

- HTTP レスポンスがコミットされる前であること。
- getWriter メソッドの呼び出し前であること。
- setCharacterEncoding メソッドの呼び出し前であること。
- setContentType メソッドによって文字エンコーディングが設定されていないこと。

条件に反する場合，setLocal メソッドは，ServletResponse クラスにロケールを設定するだけで，レスポンスの文字エンコーディングは設定しません。

Servlet 2.3

次の条件を満たす必要があります。

- HTTP レスポンスがコミットされる前であること※。

注※

- レスポンスがコミットされる前の場合，getWriter メソッドの呼び出し前後に関係なく，文字エンコーディングが設定されます。
- レスポンスがコミットされる前の場合，setContentType メソッドで文字エンコーディングが設定されているかどうかに関係なく，文字エンコーディングが設定されます。

(5)　javax.servlet.UnavailableException について

javax.servlet.UnavailableException 例外は永久的に利用できないことを示します。
javax.servlet.UnavailableException 例外を throw したサーブレット，JSP にアクセスした場合の HTTP レスポンスコードの仕様が，Servlet 2.4 仕様で追記されています。

アプリケーションサーバでこの例外を throw したサーブレット，JSP にアクセスした場合の HTTP レスポンスコードを，Servlet 2.4 以降と Servlet 2.3 に分けて示します。

Servlet 2.4 以降

404 エラーとなります。

Servlet 2.3

503 エラーとなります。

(6)　javax.servlet.http.HttpSessionListener インタフェースの sessionDestroyed メソッドについて

javax.servlet.http.HttpSessionListener インタフェースの sessionDestroyed メソッドを呼び出すタイミングが，Servlet 2.4 仕様で変更されています。

アプリケーションサーバでこのメソッドを呼び出す場合のタイミングを, Servlet 2.4 以降と Servlet 2.3 に分けて示します。

Servlet 2.4 以降

セッションが破棄される前に実行されます。

Servlet 2.3

セッションが破棄されたあとに実行されます。

なお, セッションタイムアウトが無効のとき, 次の順序でセッションに関するリスナが通知されます。Web アプリケーションのバージョンごとに，順序を示します。

Servlet 2.4 以降の場合

1. javax.servlet.http.HttpSessionListener インタフェースの sessionDestroyed()メソッド
2. javax.servlet.http.HttpSessionBindingListener インタフェースの valueUnbound()メソッド
3. javax.servlet.http.HttpSessionAttributeListener インタフェースの attributeRemoved()メソッド

Servlet 2.3 の場合

1. javax.servlet.http.HttpSessionBindingListener インタフェースの valueUnbound()メソッド
2. javax.servlet.http.HttpSessionAttributeListener インタフェースの attributeRemoved()メソッド
3. javax.servlet.http.HttpSessionListener インタフェースの sessionDestroyed()メソッド

(7)　javax.servlet.http.HttpServletResponse クラスの sendRedirect メソッドについて

javax.servlet.http.HttpServletResponse クラスの sendRedirect メソッドを使用する条件が, Servlet 2.4 仕様で変更されています。

アプリケーションサーバでこのメソッドを正常に実行するには，次の条件をすべて満たす必要があります。

- 実行するタイミングが，レスポンスがコミットされる前であること。
- 引数に適切な URL を指定すること。

この条件を満たさない場合のエラー制御を，Servlet 2.4 以降と Servlet 2.3 に分けて示します。

Servlet 2.4 以降

条件を満たさない場合は，java.lang.IllegalStateException 例外が throw されます。

Servlet 2.3

- レスポンスがすでにコミットされている場合
  java.lang.IllegalStateException 例外が throw されます。

- 引数に不正な URL が指定された場合
  レスポンスコードが 404 になります。

(8)　HTTP ステータスコード 302 のステータスメッセージについて

Servlet 2.4 仕様では，HTTP ステータスコードの 302 を示す定数として，javax.servlet.http.HttpServletResponse クラスに「SC_FOUND」が追加されています。また，下位互換性のため，Servlet 2.3 仕様で定義されていた「SC_MOVED_TEMPORARILY」はそのまま使用できます。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，「SC_FOUND」および「SC_MOVED_TEMPORARILY」を使用できます。

なお，ステータスメッセージの「Found」が Web コンテナで使用されるのは，次の場合です。

- Web アプリケーションで 302 を返す場合
- デフォルトエラーページが出力された場合（HTML 本文中）

(9)　サーブレットのサービスメソッド実行中の HttpSession のタイムアウト

Servlet 2.4 仕様では，サービスメソッド実行中の javax.servlet.http.HttpSession クラスでのタイムアウトについて仕様が追記されています。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，Web アプリケーションでのリクエスト処理を実行している間は，HttpSession はタイムアウトされません。

また，Web アプリケーション単位または URL グループ単位の同時実行スレッド数制御によって，リクエストが実行待ち状態の場合も，HttpSession はタイムアウトされません。ただし，Web コンテナ単位での同時実行スレッド数制御による実行待ち状態の場合は, HttpSession はタイムアウトされるので注意してください。

(10)　リスナで例外が発生した場合の制御について

Servlet 2.4 仕様では，リスナで例外が発生した場合についての記述が追加されています。

アプリケーションサーバを使用している場合で，リスナで例外が発生した場合の制御を，Servlet 2.4 以降と Servlet 2.3 に分けて示します。

Servlet 2.4 以降
: 該当するイベントを処理するリスナが複数あっても，例外を発生したリスナ以降のリスナは，実行されません。

Servlet 2.3
: 発生した例外は Web コンテナによって catch されます。複数のリスナが登録されている場合，例外が catch されたあとに，正常時と同様に登録されているリスナが順次実行されます。

(11)　共通で使用する外部のライブラリ（Extension）について

Web アプリケーションで外部のライブラリを使用する場合に記載する MANIFEST ファイルの扱いについて，Servlet 2.4 仕様では記述が変更されています。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，MANIFEST ファイルの存在，および MANIFEST ファイルの内容は確認されません。

(12)　サーブレットのバージョンが異なる Web アプリケーション間のクロスコンテキストの使用について

サーブレットのバージョンが異なる Web アプリケーション間で，クロスコンテキストを使用したリクエストを forward したあとの動作，および include したあとの動作を，次の表に示します。

表 6-21　forward 後および include 後の動作

項番	Servlet 2.4 仕様の追加機能／Web アプリケーションのバージョンで違いのある機能	リクエストの forward 先／include 先の動作	
		2.4 から 2.3 に forward／include※1	2.3 から 2.4 に forward／include※2
1	forward 時または include 時のフィルタ適用	forward 後／include 後のサーブレット，または JSP から，さらに forward または include する場合，Servlet 2.4 仕様が適用され，フィルタは使用できます。	forward 後／include 後のサーブレット，または JSP からさらに forward または include する場合，Servlet 2.3 仕様が適用され，フィルタは使用できません。
2	javax.servlet.ServletRequestAttributeListener の呼び出し	Servlet 2.4 仕様が適用され，リクエストへの属性追加時にリスナが使用できます。	Servlet 2.3 仕様が適用され，リクエストへの属性追加時にリスナは使用できません。
3	JSP のコンパイル	Servlet 2.3 に対応したアプリケーションとして JSP コンパイルを実行します。	Servlet 2.4 に対応したアプリケーションとして JSP コンパイルを実行します。
4	javax.servlet.ServletResponse クラスの setLocale メソッド	Servlet 2.4 仕様が適用され，次に示す条件をすべて満たす場合，文字エンコーディングの設定が有効となります。 • レスポンスがコミットされる前であること。 • getWriter メソッドの呼び出し前であること。 • setCharacterEncoding メソッドの呼び出し前であること。 • serContentType メソッドで文字エンコーディングが設定されていないこと。	Servlet 2.3 仕様が適用され，レスポンスがコミットされる前である場合，文字エンコーディングの設定が有効となります。
5	永久的に利用できないことを示す javax.servlet.UnavailableException 例外を throw したサーブレット，JSP へのディスパッチ	Servlet 2.3 仕様が適用され，ステータス 503 を設定したレスポンスが返されます。	Servlet 2.4 仕様が適用され，ステータス 404 を設定したレスポンスが返されます。
6	javax.servlet.http.HttpSessionListener インタフェースの sessionDestroyed メソッド	Servlet 2.4 仕様が適用され，HTTP セッションが破棄される前に実行されます。	Servlet 2.3 仕様が適用され，HTTP セッションが破棄されたあとに実行されます。
7	javax.servlet.http.HttpServletResponse クラスの	Servlet 2.4 仕様の仕様が適用され，java.lang.IllegalStateException 例外が throw されます。	Servlet 2.3 の仕様が適用され，ステータス 404 がレスポンスに設定されます。

項番	Servlet 2.4 仕様の追加機能／Web アプリケーションのバージョンで違いのある機能	リクエストの forward 先／include 先の動作	
		2.4 から 2.3 に forward／include※1	2.3 から 2.4 に forward／include※2
7	sendRedirect メソッドに不正な URL を指定	Servlet 2.4 仕様の仕様が適用され，java.lang.IllegalStateException 例外が throw されます。	Servlet 2.3 の仕様が適用され，ステータス 404 がレスポンスに設定されます。
8	使用するリスナ定義	次に示すリスナの場合，forward 先または include 先のアプリケーションで定義されたリスナが動作します。 • javax.servlet.ServletContextAttributeListener 次に示すリスナの場合，forward または include を呼び出したアプリケーションで定義されたリスナが動作します。 • javax.servlet.ServletRequestAttributeListener • javax.servlet.http.HttpSessionListener • javax.servlet.http.HttpSessionAttributeListener	次に示すリスナの場合，forward 先または include 先のアプリケーションで定義されたリスナが動作します。 • javax.servlet.ServletContextAttributeListener 次に示すリスナの場合，forward または include を呼び出したアプリケーションで定義されたリスナが動作します。 • javax.servlet.http.HttpSessionListener • javax.servlet.http.HttpSessionAttributeListener
9	該当するイベントを処理するリスナが web.xml で複数定義されていた場合に，forward もしくは include 先のアプリケーション内でリスナが例外を発生したときの動作	次に示すリスナの場合，Servlet 2.4 仕様が適用され，例外を発生したリスナ以降のリスナは実行されません。 • javax.servlet.ServletRequestAttributeListener • javax.servlet.http.HttpSessionListener • javax.servlet.http.HttpSessionAttributeListener 次に示すリスナの場合，Servlet 2.3 仕様が適用され，発生した例外をキャッチし，その後正常時と同様に登録されている次のリスナへ処理が移ります。 • javax.servlet.ServletContextAttributeListener	次に示すリスナの場合，Servlet 2.4 仕様が適用され，例外を発生したリスナ以降のリスナは実行されません。 • javax.servlet.ServletContextAttributeListener 次に示すリスナの場合，Servlet 2.3 仕様が適用され，発生した例外をキャッチし，その後正常時と同様に登録されている次のリスナへ処理が移ります。 • javax.servlet.http.HttpSessionListener • javax.servlet.http.HttpSessionAttributeListener
10	web.xml で指定されたエラーページを表示したレスポンスのステータスコード	Servlet 2.4 仕様が適用され，エラー発生時のステータスコードのレスポンスが返されます。	Servlet 2.3 仕様が適用され，ステータス 200 のレスポンスが返されます。

注※1

Servlet 2.4 に対応したアプリケーションから Servlet 2.3 に対応したアプリケーションに forward または include した場合を表します。

注※2

Servlet 2.3 に対応したアプリケーションから Servlet 2.4 に対応したアプリケーションに forward または include した場合を表します。

(13)　Servlet 2.4 の Web アプリケーションから EJB 3.0 の Session Bean を呼び出す場合

Servlet 2.4 の Web アプリケーションから EJB 3.0 の Session Bean を呼び出す場合は，

web.xml に<ejb-ref>タグ，<ejb-local-ref>タグを指定しないで，サーブレットクラスに@EJB アノテーション，@EJBs アノテーションを指定してください。

J2EE サーバがサーブレットに対して Enterprise Bean の DI を実行します。

6.2.6　JSP 実装時の注意事項

JSP を実装するときの注意事項を示します。

(1)　include ディレクティブ利用時の注意

- JSP ファイルの include ディレクティブでファイルをインクルードする場合，インクルード元となる JSP ファイルでエンコーディングを指定してください。
- JSP で HTML などの静的ファイルをインクルードする場合は，include アクションを使用しないで，include ディレクティブを使用してください。include アクションは JSP やサーブレットなど動的なページをインクルードする場合に使用してください。

(2)　タグライブラリ利用時の注意

- タグライブラリを作成する場合には，タグライブラリを記述したクラスファイルの先頭に package 文で必ずパッケージ名を付けるようにしてください。パッケージ名が付いていない場合，そのタグライブラリは正常に動作しません。
- タグライブラリ・ディスクリプタ（TLD ファイル）で，<variable>タグ内の<scope>タグ要素には，NESTED，AT_BEGIN，AT_END のどれかを指定してください。これら以外の値を指定した場合には，デフォルトの NESTED が仮定されて実行されます。
- タグライブラリのタグハンドラを実装する場合には，doStartTag メソッド，doAfterBody メソッド，および doEndTag メソッドが仕様で規定されている戻り値を返すようにしてください。仕様で規定されている戻り値以外の値を返した場合，デフォルトの戻り値を仮定して動作します。デフォルトの戻り値とは，javax.servlet.jsp.tagext.TagSupport または javax.servlet.jsp.tagext.BodyTagSupport クラスの各メソッドがオーバーライドしない場合に返す戻り値です。例えば，BodyTag インタフェースを実装したクラス（または BodyTagSupport クラスを継承したクラス）で，doStartTag メソッドが EVAL_PAGE を戻り値として返した場合，デフォルトの戻り値である EVAL_BODY_BUFFERED が返されたと仮定して動作します。
- Web アプリケーションのバージョンが 2.3 の場合で，かつタグライブラリのタグハンドラで属性の指定を持つカスタムタグを実装した場合，JSP のカスタムタグで同じ属性を複数指定すると，該当するタグハンドラの setter メソッドが指定された回数呼び出されます。通常指定された値を上書きするような setter メソッドを実装している場合には，後ろに記述された属性の値が有効になります。
- タグライブラリ・ディスクリプタ（TLD ファイル）では，タグの要素として空文字列を指定しないでください。空文字列を指定するとは，開始タグと終了タグ間に何も記述しない（例えば，<param-name></param-name>）または空要素タグを記述する（例えば，<param-name />）ことを言います。タグの要素として空文字列を指定した場合の動作は保証されません。
- タグライブラリ・ディスクリプタ（TLD）ファイルで xsi:schemaLocation 属性を指定する場合，絶対パスを指定してください。

- TLD ファイルでタグまたはタグライブラリの拡張要素を使用する際に，xsi:schemaLocation 属性に相対パスを指定した場合の動作の保証はされません。

(3)　<%=%>タグ記述時の注意

JSP で<%= %>タグを使用する場合は，その中に「;（セミコロン）」が入らないようにしてください。セミコロンが入っている場合，JSP コンパイルでエラーとなります。

(4)　URL 指定とマッピング定義によるアクセスについて

直接 JSP ファイルのパスを URL 指定してアクセスする場合とマッピング定義された URL でアクセスする場合には，それぞれ別々のインスタンスが生成されます。したがって，jspInit メソッドがそれぞれのインスタンスで実行されることに注意してください。なお，<load-on-startup>タグで起動時にロードされるように指定した場合，起動時にロードされるインスタンスは，マッピング定義された URL でアクセスするものと同じになります。

(5)　<jsp:plugin>タグ利用時の注意

JSP ページで，plugin アクションまたは JSP ドキュメントでの<jsp:plugin>タグの code 属性は必ず指定してください。省略した場合はコンパイルエラーとなります。

(6)　JSP ドキュメント内の version 属性について

JSP ドキュメントでは，<jsp:root>タグの属性として，使用するバージョン情報を記述できます。しかし，JSP で使用できる機能範囲は，該当する JSP を含む Web アプリケーションの web.xml のバージョン情報に従います。

例えば，<jsp:root version="1.2">と記述している JSP ドキュメントであっても，JSP 2.0 仕様で追加された JSP EL を使用できます。

(7)　taglib ディレクティブの uri 属性で指定する URI と TLD ファイルのマッピングについて

taglib ディレクティブの uri 属性で指定する URI は，JSP 仕様によって次に示すどれかの方法でマッピングします。同じ URI を異なる TLD ファイルにマッピングすることはできません。URI が重複した場合は，次に示す番号順を優先順位とし，優先順位の高いマッピングが有効になります。

1. JSTL および JSF の URI の暗黙マッピング（Servlet 2.5 仕様以降の Web アプリケーション）
2. web.xml の<taglib>要素の<taglib-uri>要素で指定した URI と，<taglib-location>要素で指定した TLD ファイルのマッピング
3. Web アプリケーション内の TLD ファイルの<uri>要素で指定した URI と，その TLD ファイル自身のマッピング
4. ライブラリ JAR（cjjspc コマンドの場合は-classpath オプションで指定した JAR ファイル）の/META-INF ディレクトリ以下に格納された TLD ファイルの<uri>要素で指定した URI と，その TLD ファイル自身のマッピング

次に，2.～4.までの URI と TLD ファイルのマッピングする場合のディレクトリ構成とファイルの記述例を示します。

(a)　web.xml の<taglib>要素の<taglib-uri>要素で指定した URI と，<taglib-location>要素で指定した TLD ファイルのマッピング（2.の場合）

2.の場合のディレクトリ構成とファイルの記述例について説明します。

2.の場合，TLD ファイルは Web アプリケーション内に格納されています。2.の場合のディレクトリ構成の例を次に示します。

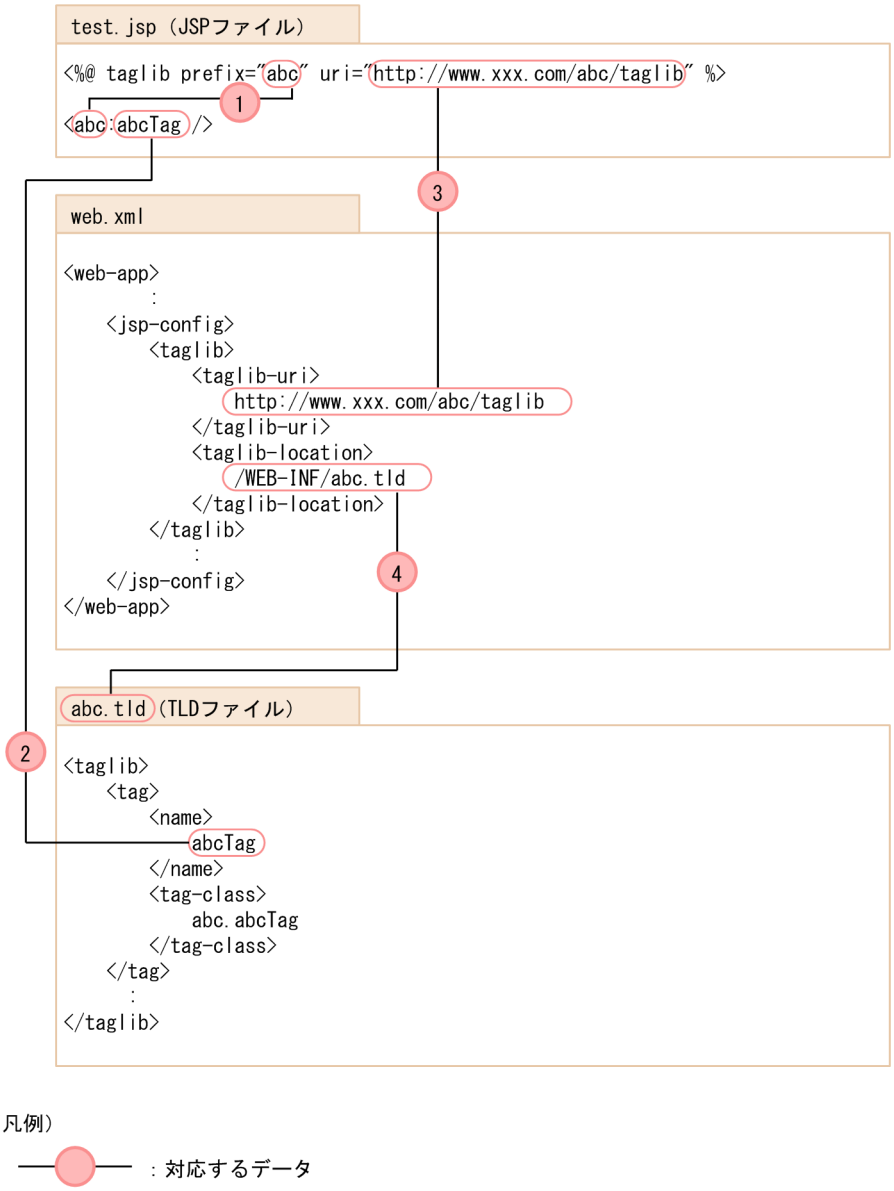
図 6-3　ディレクトリ構成（2.の場合）

アプリケーションのディレクトリ構成

```
application¥
  └context1¥
      ├jsp¥
      │ └test.jsp
      └WEB-INF¥
        ├web.xml
        ├classes¥
        │ └abc
        │   └abcTag.class
        └abc.tld
```

このディレクトリ構成の場合の taglib ディレクティブの uri 属性で指定する URI と TLD ファイルのマッピングを次の図に示します。

図 6-4　URI と TLD ファイルのマッピング（2.の場合）



図中のデータの対応について説明します。

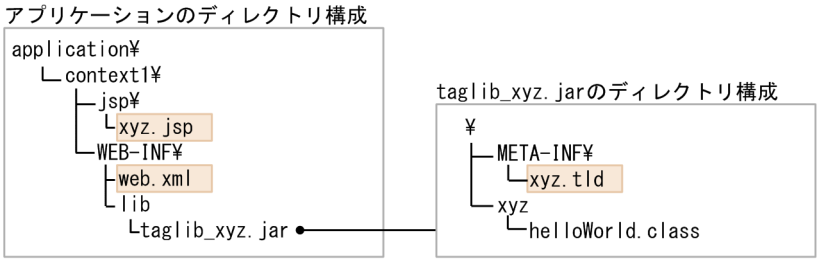
1. タグのプリフィックスに taglib ディレクティブで指定したプリフィックスを指定します。
2. TLD ファイルの<name>要素で指定した値を，JSP ファイルのプリフィックスと対応づけます。
3. web.xml の<taglib-uri>要素で，JSP ファイルの taglib ディレクティブで指定した uri を指定します。
4. web.xml の<taglib-location>要素で，マッピングする TLD ファイル名を指定します。

(b)　TLD ファイルの<uri>要素で指定した URI と，その TLD ファイル自身をマッピングする場合

3.または 4 の場合のディレクトリ構成とファイルの記述例について説明します。

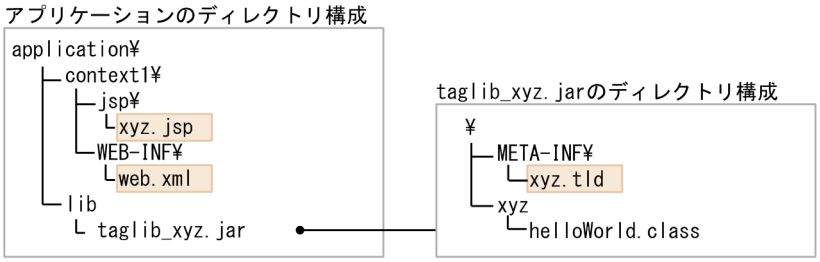
3.の場合，TLD ファイルは Web アプリケーション内に格納されています。3.の場合のディレクトリ構成の例を次に示します。

図 6-5　ディレクトリ構成（3.の場合）



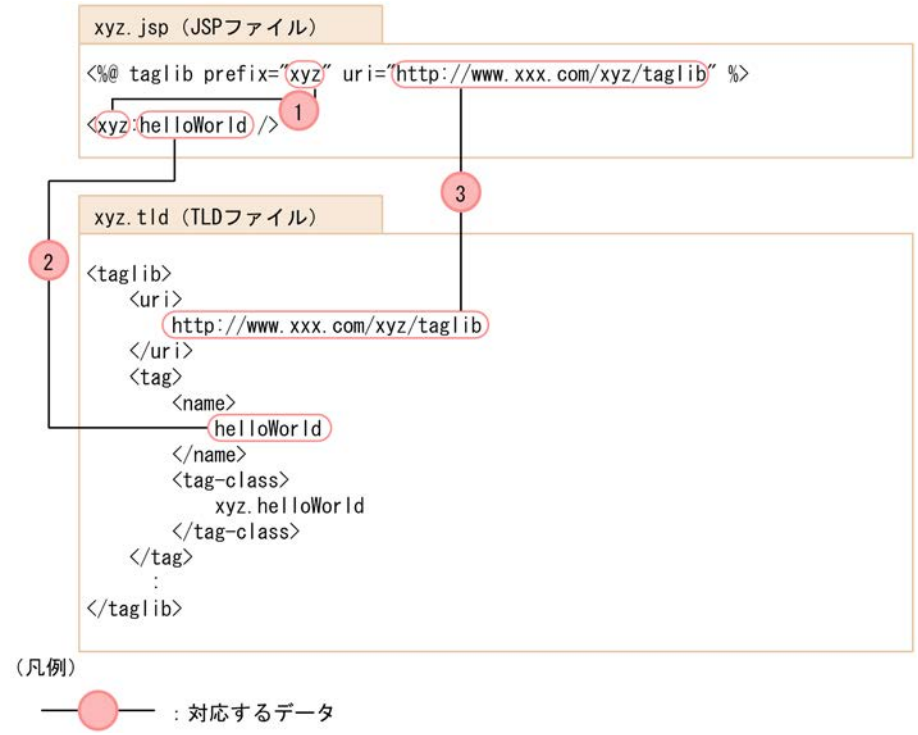
4.の場合，TLD ファイルはライブラリ JAR の/META-INF ディレクトリ以下に格納されています。4.の場合のディレクトリ構成の例を次に示します。

図 6-6　ディレクトリ構成（4.の場合）



3.または 4.の場合のディレクトリ構成での taglib ディレクティブの uri 属性で指定する URI と TLD ファイルのマッピングを次の図に示します。

図 6-7　TLD ファイルの<uri>要素で指定した URI と，その TLD ファイル自身のマッピング（3.または 4.の場合）



図中のデータの対応について説明します。

1. タグのプリフィックスに taglib ディレクティブで指定したプリフィックスを指定します。
2. TLD ファイルの<name>要素で指定した値を，JSP ファイルのプリフィックスと対応づけます。
3. TLD ファイルの<url>要素で，JSP ファイルの taglib ディレクティブで指定した uri を指定します。

URI の重複が検出された場合，次のメッセージが Web アプリケーション単位で出力され，該当するマッピングは無視されます。

表 6-22　URI 重複時に出力されるメッセージと出力条件

メッセージ ID	出力条件
KDJE39314-W	TLD ファイルで記述した<uri>要素の内容が，web.xml の<taglib-uri>要素の内容，またはほかの TLD ファイルで記述した<uri>要素の内容と重複したときに出力されます。
KDJE39315-W	TLD ファイルで記述した<uri>要素の内容が，ほかの TLD ファイルで記述した<uri>要素の内容と重複したときに出力されます。
KDJE39316-W	web.xml に指定した<taglib>要素の内容と重複する<taglib-uri>要素を持つほかの<taglib>要素が指定されている場合に出力されます。
KDJE39325-W	web.xml の<taglib-uri>要素の内容，またはほかの TLD ファイルで記述した<uri>要素の内容が，Java EE 仕様のタグライブラリ（JSTL，JSF）の URI と重複したときに出力されます。
KDJE39326-W	ライブラリ JAR（cjjspc コマンドの場合は-classpath オプションで指定した JAR ファイル）に格納された TLD ファイルの<uri>要素の内容が，web.xml の<taglib-uri>要素の内容，またはほかの TLD ファイルで記述した<uri>要素の内容と重複したときに出力されます。

(8)　JSP で動的なページをインクルードする場合の注意

JSP で，JSP やサーブレットなどの動的なページをインクルードする場合は，javax.servlet.RequestDispatcher の include メソッドを使用しないで，include アクションを使用してください。

(9)　TLD ファイルでの DOCTYPE 宣言への内部サブセットの記述について

TLD ファイル（タグライブラリ・ディスクリプタ）で，DOCTYPE 宣言に内部サブセットを記述しないでください。

タグの要素拡張またはタグライブラリの要素拡張で xsi:schemaLocation 属性に指定する URI は絶対 URI だけ指定できます。タグの要素拡張またはタグライブラリの要素拡張以外の目的で，Java EE 仕様で定められた DTD／XML スキーマ以外を参照しないでください。

(10)　JSP ドキュメントおよび XML 形式のタグファイルでの外部サブセット URI の指定について

DOCTYPE 宣言を指定する場合，外部サブセット URI には絶対 URI だけ指定できます。また，XML1.0 仕様で定義された外部エンティティ参照をする場合，外部エンティティ宣言に指定する URI には絶対 URI だけ指定できます。相対 URI を指定した外部サブセットおよび外部エンティティは参照できません。

(11)　javax.servlet.ServletRequest オブジェクトの javax.servlet.jsp.jspException 属性の値について

JSP ファイルで例外をスローした場合，page ディレクティブの errorPage 属性でエラーページを指定しているとき，例外が javax.servlet.ServletRequest オブジェクトの javax.servlet.jsp.jspException 属性に設定されると JSP 仕様に記述されていますが，page ディレクティブの errorPage 属性でエラーページを指定しない場合も設定されます。

(12)　TLD ファイルのバージョンに関する注意事項

07-00 以降のバージョンでは，JSP トランスレーション時に TLD ファイルのバージョン（TLD ファイルが対応する JSP のバージョン）をチェックします。そのため，TLD ファイルのバージョンが，Web アプリケーションのバージョンに対応している JSP および TLD ファイルのバージョンより上位の JSP の場合，JSP トランスレーションでエラーになります。また，JSP 1.2，および JSP 2.0 仕様以降では，TLD ファイルにスキーマ言語を定義する必要があります。

TLD ファイルのバージョンは，TLD ファイルに記述されたスキーマ言語から判定します。ただし，スキーマ言語が定義されていない場合は，Web アプリケーションのバージョンから判定します。

TLD ファイルにスキーマ言語を定義していない場合の TLD ファイルのバージョンを次の表に示します。

表 6-23　スキーマ言語未定義時の TLD ファイルのバージョン

Web アプリケーションのバージョン	TLD ファイルのバージョン
2.2	1.1
2.3	1.2
2.4	2.0
2.5	2.1

(13)　JSP でサポートしている文字エンコーディングについて

ここでは，JSP ファイル，およびタグファイルで使用できる文字エンコーディングについて説明します。

(a)　JSP ページの場合

JSP ページで使用できる文字エンコーディングと，文字エンコーディングの指定方法について次に示します。

- JSP ページで使用できる文字エンコーディング

  JavaVM がサポートしている文字エンコーディングが使用できます。JavaVM がサポートしている文字エンコーディングについては，JDK のドキュメントを参照してください。

  ただし，UTF-16 などの英数字が複数バイトで表現される文字エンコーディングについては，次の二つの条件を満たしている必要があります。

  - Servlet 2.4/JSP 2.0 仕様以降に準拠した Web アプリケーションである。
  - JSP ファイルの先頭に BOM を付加している。

- 文字エンコーディングの指定方法

  JSP ページに使用する文字エンコーディングは，次に示す方法から一つ以上の方法を選択して指定できます。

- JSP ページの先頭に BOM を付加する（Servlet 2.5/JSP 2.1 仕様以降に準拠した Web アプリケーションの場合）。
- web.xml の<jsp-property-group>要素内の<page-encoding>要素に指定する（Servlet 2.4/JSP 2.0 仕様以降に準拠した Web アプリケーションの場合）。
- page ディレクティブの pageEncoding 属性に指定する（Servlet 2.3/JSP 1.2 仕様以降に準拠した Web アプリケーションの場合）。
- page ディレクティブの contentType 属性に指定する。

指定できる文字列は，java.nio API 用の正準名と java.lang API 用の正準名に記載されている文字エンコーディング，およびそれらの別名になります。

なお，指定する文字エンコーディングと，JSP ページで使用する文字エンコーディングは，必ず一致するようにしてください。

(b)　JSP ドキュメントの場合（Servlet 2.4 仕様以降に準拠した Web アプリケーションの場合）

Servlet 2.4 仕様以降に準拠した Web アプリケーション内の JSP ドキュメントで使用できる文字エンコーディングと，使用する文字エンコーディングの指定方法について次に示します。

- JSP ドキュメントで使用できる文字エンコーディング

  XML Processor がサポートしている文字エンコーディング※が使用できます。

  ただし，拡張子が jspx でない JSP ドキュメントについては，web.xml の<jsp-property-group>要素内の<is-xml>要素が指定されていない場合，JSP ドキュメントの文字エンコーディングに ISO-10646-UCS-4 は使用できません。

  また，UTF-16 などの英数字が複数バイトで表現される文字エンコーディングについては，次の二つの条件を満たしている必要があります。

  - JSP ドキュメントの先頭に BOM を付加している
  - ISO-10646-UCS-2 を使用している場合は，ビッグエンディアンの ISO-10646-UCS-2 を使用している

- 文字エンコーディングの指定方法

  JSP ドキュメントに使用する文字エンコーディングは，XML 宣言の encoding 属性に指定します。指定できる文字列は，XML Processor がサポートしている文字エンコーディング※です。

  ただし，拡張子が jspx でない JSP ドキュメントについては，web.xml の<jsp-property-group>要素内の<is-xml>要素が指定されていない場合，次に示す方法のどれか一つ以上の方法で使用する文字エンコーディングを指定します。

  - XML 宣言の encoding 属性に指定する。
  - web.xml の<jsp-property-group>要素内の<page-encoding>要素に指定する。
  - page ディレクティブの pageEncoding 属性に指定する。

  指定できる文字列は，java.nio API 用の正準名と java.lang API 用の正準名に記載されている文字エンコーディング，およびそれらの別名になります。

  なお，指定する文字エンコーディングと，JSP ドキュメントで使用する文字エンコーディングは，必ず一致するようにしてください。

注※　XML Processor がサポートしている文字エンコーディングについては，マニュアル「XML Processor ユーザーズガイド」の「1.3.2　処理できる文字コード」を参照してください。

(c)　JSP ドキュメントの場合（Servlet 2.3 仕様に準拠した Web アプリケーションの場合）

Servlet 2.3 仕様に準拠した Web アプリケーション内の JSP ドキュメントで使用できる文字エンコーディングと，使用する文字エンコーディングの指定方法について次に示します。

- JSP ドキュメントで使用できる文字エンコーディング

  XML Processor がサポートしている文字エンコーディングが使用できます。XML Processor がサポートしている文字エンコーディングについては，マニュアル「XML Processor ユーザーズガイド」の「1.3.2　処理できる文字コード」を参照してください。

  ただし，UTF-16 などの英数字が複数バイトで表現される文字エンコーディングは使用できません。

- 文字エンコーディングの指定方法

  JSP ドキュメントに使用する文字エンコーディングは，次に示す方法の両方またはどちらかを選択して指定できます。

  - XML 宣言の encoding 属性に指定する。
  - page ディレクティブの pageEncoding 属性に指定する。

  指定できる文字列は，java.nio API 用の正準名と java.lang API 用の正準名に記載されている文字エンコーディング，およびそれらの別名になります。

  なお，指定する文字エンコーディングと，JSP ドキュメントで使用する文字エンコーディングは，必ず一致するようにしてください。

(d)　標準シンタックスのタグファイルの場合

標準シンタックスのタグファイルで使用できる文字エンコーディングと，使用する文字エンコーディングの指定方法について次に示します。

- タグファイルで使用できる文字エンコーディング

  JavaVM がサポートしている文字エンコーディングが使用できます。JavaVM がサポートしている文字エンコーディングについては，JDK のドキュメントを参照してください。

  ただし，UTF-16 などの英数字が複数バイトで表現される文字エンコーディングについては，タグファイルの先頭に BOM を付加する必要があります。

- 文字エンコーディングの指定方法

  タグファイルに使用する文字エンコーディングは，次に示す方法の両方またはどちらかを選択して指定できます。

  - タグファイルの先頭に BOM を付加する（Servlet 2.5/JSP 2.1 仕様以降に準拠した Web アプリケーションの場合）。
  - tag ディレクティブの pageEncoding 属性に指定する（Servlet 2.4/JSP 2.0 仕様以降に準拠した Web アプリケーションの場合）。

  指定できる文字列は，java.nio API 用の正準名と java.lang API 用の正準名に記載されている文字エンコーディング，およびそれらの別名になります。

  なお，指定する文字エンコーディングとタグファイルで使用する文字エンコーディングは，必ず一致するようにしてください。

(e)　XML シンタックスのタグファイルの場合

XML シンタックスのタグファイルで使用できる文字エンコーディングと，使用する文字エンコーディングの指定方法を示します。

- タグファイルで使用できる文字エンコーディング

XML Processor がサポートしている文字エンコーディング※が使用できます。

- 文字エンコーディングの指定方法
  タグファイルに使用する文字エンコーディングは，XML1.0 仕様に従って指定してください。指定できる文字列は，XML Processor がサポートしている文字エンコーディング※です。
  なお，指定する文字エンコーディングとタグファイルで使用する文字エンコーディングは，必ず一致するようにしてください。

注※　XML Processor がサポートしている文字エンコーディングについては，マニュアル「XML Processor ユーザーズガイド」の「1.3.2　処理できる文字コード」を参照してください。

(f)　デフォルトの文字エンコーディング

JSP ファイル，またはタグファイルに文字エンコーディングを明示的に指定しない場合，デフォルトの文字エンコーディングで JSP を処理します。

なお，この場合も，デフォルトの文字エンコーディングと，タグファイルで使用する文字エンコーディングは，必ず一致するようにしてください。デフォルトの文字エンコーディングについて，Servlet および JSP の仕様ごとに次の表に示します。

表 6–24　デフォルトの文字エンコーディング

仕様	JSP ページ	JSP ドキュメント	標準シンタックスのタグファイル	XML シンタックスのタグファイル
Servlet 2.2/ JSP1.1	ISO-8859-1	ISO-8859-1	−	−
Servlet 2.3/ JSP 1.2	ISO-8859-1	ISO-8859-1	−	−
Servlet 2.4 以降/ JSP 2.0，JSP 2.1	ISO-8859-1	UTF-8	ISO-8859-1	UTF-8

（凡例）−：該当しない

デフォルトの文字エンコーディングは, デフォルトの文字エンコーディング設定機能を使用しても設定できます。詳細については,「2.6　デフォルトの文字エンコーディング設定機能」を参照してください。

(14)　setProperty アクション使用時の注意事項

JSP ページでの setProperty アクション, または JSP ドキュメントでの<jsp:setProperty>タグの name 属性に, 不正な値を指定しないでください。<jsp:setProperty>タグの name 属性に不正な値を指定した場合の動作は保証されません。

(15)　JSP のタグの属性への空文字列の指定について

JSP ページのディレクティブやアクションのタグ，および JSP ドキュメントの「jsp:」で始まるタグの属性に空文字列を指定しないでください。

(16)　JSP のテンプレートテキストの前後にスクリプトレットを記述する場合の注意

JSP のテンプレートテキストの前後に if 文などの制御文を含むスクリプトレットを記載する場合は, 明示的に"{}"（中括弧）で囲む必要があります。

JSP コンパイルの結果，1 行の JSP のテンプレートテキストが Java ファイルでも 1 行のステートメントとなるとは限りません。複数行のステートメントとして出力される可能性があります。そのため, テンプレー

トテキストの前後のスクリプトレットで if 文などの制御文を明示的に"{}"（中括弧）で囲んでいない場合は，意図しない動作となるおそれがあります。

(17)　スクリプトレット使用時の注意

JSP でスクリプトレットを使用して java コードを記述する場合に，スクリプトレットに return 文または throw 文を記述するときは，if 文などのブロック内に記述するようにしてください。

(18)　EL が無効な場合のトランスレーションエラーについて

JSP 内に"${aaa"のように EL として不正な文字列を含む場合，EL を無効に設定しても，EL 不正のトランスレーションエラーが発生します。

JSP 仕様のバージョンごとに，トランスレーションエラーが発生する EL 開始文字を次の表に示します。

表 6-25　トランスレーションエラーが発生する EL 開始文字

JSP 仕様のバージョン	EL 開始文字
JSP2.1	'$'，'#'
JSP2.0	'$'
JSP1.2	なし

(19)　TLD ファイルの schemaLocation の記載について

TLD ファイルに記述する schemaLocation は，TLD のバージョンに応じて，次の表に示すどちらかの値を指定する必要があります。

TLD のバージョン	schemaLocation に指定する値
2.0	"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
2.1	"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"

6.2.7　JSP 2.1 仕様で追加，変更された仕様についての注意事項

JSP 2.1 仕様で追加，変更された仕様を，アプリケーションサーバ上で利用するときの注意事項を示します。JSP 2.1 仕様および JSP 2.0 仕様については，それぞれの仕様書（JSP 2.1 仕様書，JSP 2.0 仕様書）を参照してください。

(1)　EL2.1

JSP 2.1 仕様の EL は，JSP 2.0 仕様の EL と，JSF1.1 仕様の EL が統合された EL です。

EL2.1 の仕様として EL の文法や API が定めてあり，JSP 2.1 仕様や JSF1.2 仕様で定義される暗黙オブジェクトなどの変数は，EL の API を通して使用できます。

次に示す JSP 2.1 仕様の EL に関する機能の追加および仕様変更について，以降で説明します。

- #{}の書式の EL
- TLD への要素追加

- 下位互換性のオプション追加

(a)　#{}の書式の EL

JSP 2.0 仕様である${}の書式の EL に加え，JSF1.1 仕様の EL である#{}の書式の EL を JSP 2.1 仕様の機能として記述できます。

#{}の書式の EL の評価のタイミングについて

- #{}の書式の EL は JSP 出力時に評価されません。
- Web コンテナによって，EL の評価結果の値ではなく EL のオブジェクトである javax.el.ValueExpression または javax.el.MethodExpression がタグハンドラに渡されます。渡されたオブジェクトのメソッドが，タグハンドラの実装によって任意のタイミングで評価されます。

(b)　TLD への要素追加

JSP 2.1 仕様では，#{}の書式を期待するタグの属性であるかどうかを示すため，TLD ファイルの<attribute>要素内に次に示す要素が追加されました。

- <deferred-value>要素
- <deferred-value>要素内の<type>要素
- <deferred-method>要素
- <deferred-method>要素内の<method-signature>要素

また，タグファイルの attribute ディレクティブにもここに示した TLD の要素に対応する属性が追加されました。

(c)　下位互換性のオプション追加

JSP 2.1 仕様では，EL2.1 仕様で#{}の書式が追加されたことに伴い，次の個所に#{}の書式の EL を記述するとトランスレーションエラーが発生します。

- JSP のファイルまたはタグファイルのテンプレートテキスト
- 遅延評価でないカスタムタグの属性値

アプリケーションサーバでは，次に示す要素または属性の値に true と指定した場合，テンプレートテキストまたは遅延評価でないカスタムタグの属性値に#{}の書式の EL があってもトランスレーションエラーは発生しません。#{}はそのまま文字列として出力されます。

- web.xml の<web-app><jsp-config><jsp-property-group>要素内の<deferred-syntax-allowed-as-literal>要素
- page，tag ディレクティブの deferredSyntaxAllowedAsLiteral 属性

web.xml での設定と page，tag ディレクティブでの設定を組み合わせた場合，#{}の EL に対してトランスレーションエラーが発生するかどうかを次の表に示します。

表 6-26　web.xml での設定と page，tag ディレクティブでの設定を組み合わせた場合のトランスレーションエラーの発生有無

web.xml の<deferred-syntax-allowed-as-literal>要素	page，tag ディレクティブの deferredSyntaxAllowedAsLiteral 属性		
	true	false	指定なし
true	○	×	○

web.xml の<deferred-syntax-allowed-as-literal>要素	page，tag ディレクティブの deferredSyntaxAllowedAsLiteral 属性		
	true	false	指定なし
false	○	×	×
指定なし	○	×	×

（凡例）

○：トランスレーションエラーは発生しないで#{}の EL がそのまま出力される

×：トランスレーションエラーが発生する

注　web.xml での設定より page，tag ディレクティブでの設定が優先されます。

(2)　不要なホワイトスペースの削除機能

JSP 2.1 仕様では，JSP のテンプレートテキストに含まれる不要なホワイトスペースを削除する機能が追加されました。アプリケーションサーバでは，JSP 2.1 仕様に従ってこの機能をサポートします。

次に示す要素または属性の値に true と指定した場合，JSP の出力時にホワイトスペースだけのテンプレートテキストは削除されます。

- web.xml の<web-app><jsp-config><jsp-property-group>要素内の<trim-directive-whitespaces>要素
- page，tag ディレクティブの trimDirectiveWhitespaces 属性

ただし，ホワイトスペースではないテンプレートテキストと連続しているホワイトスペースは削除されません。

表 6–27　web.xml での設定と page，tag ディレクティブでの設定を組み合わせた場合のホワイトスペースの削除機能の有効／無効

web.xml の<trim-directive-whitespaces>要素	page，tag ディレクティブの trimDirectiveWhitespaces 属性		
	true	false	指定なし
true	○	×	○
false	○	×	×
指定なし	○	×	×

（凡例）

○：ホワイトスペースの削除機能が有効になる

×：ホワイトスペースの削除機能が無効になる

注　web.xml での設定より page，tag ディレクティブでの設定が優先されます。

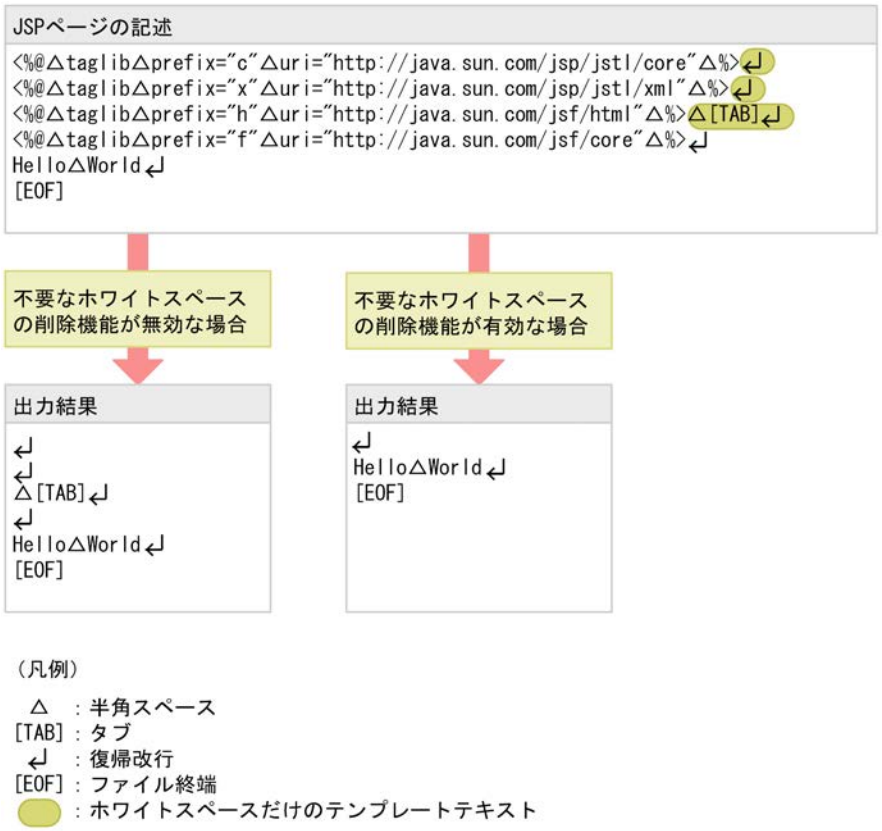
なお，JSP ドキュメントの page ディレクティブ，または XML シンタックスの tag ディレクティブに trimDirectiveWhitespaces 属性を設定した場合，設定値によって次のように処理されます。

- 設定値が true または false の場合，trimDirectiveWhitespaces 属性の設定は無視され，正常に処理されます。
- 設定値が true または false 以外の不正値を指定した場合，トランスレーションエラーとなります。

(a)　ホワイトスペースの削除例

次の図に，JSP 要素の間に，ホワイトスペースだけのテンプレートテキストが存在する例を示します。

図 6-8　ホワイトスペースだけのテンプレートテキストが存在する例



（凡例）

△　：半角スペース
[TAB]：タブ
↲　：復帰改行
[EOF]：ファイル終端
　　：ホワイトスペースだけのテンプレートテキスト

1～4 行目の taglib ディレクティブは出力しないため無視し，1 行目の復帰改行，2 行目の復帰改行，および 3 行目の半角スペースから復帰改行がホワイトスペースだけのテンプレートテキストとなります。

JSP ページの 4 行目の復帰改行は，ホワイトスペースではないテンプレートテキストと連続しているホワイトスペースであるため削除されません。そのため，ホワイトスペースの削除機能が有効な場合，4 行目の復帰改行から[EOF]までが出力されます。

(3)　タグハンドラに一意の識別子を設定する機能

JSP 2.1 仕様では，トランスレーション単位（JSP と include ディレクティブでインクルードされるファイル）ごとに一意である識別子をタグハンドラに設定する機能が追加されました。この機能は，JspIdConsumer インタフェースをタグハンドラに実装することで使用できます。JspIdConsumer インタフェースのパッケージ名は javax.servlet.jsp.tagext です。

Web コンテナは，JSP の実行時にこのインタフェースを実装したタグハンドラに対して，インタフェースの setJspId メソッドを使用して識別子を設定します。識別子は JSP コンパイル時に決定されるため，リクエスト処理ごとに識別子が変更されるおそれはありません。

一意の識別子として，id<N>という文字列を使用します。<N>は整数値を表します。<N>は 0～2,147,483,647 の範囲で割り当てられます。

(4)　JSP の API の追加

JSP 2.1 仕様では次に示すクラスおよびメソッドが追加されました。

- JSP 2.1 仕様で追加されたクラス

パッケージ名	クラス名
javax.servlet.jsp	JspApplicationContext

- JSP 2.1 仕様でコンストラクタが追加されたクラス

パッケージ名	クラス名
javax.servlet.jsp.tagext	TagAttributeInfo

- JSP 2.1 仕様で追加されたメソッド

パッケージ名	クラス名	メソッド名
javax.servlet.jsp	JspFactory	getJspApplicationContext
javax.servlet.jsp.tagext	TagAttributeInfo	getDescription
		isDeferredValue
		isDeferredMethod
		getExpectedTypeName
		getMethodSignature
	TagLibraryInfo	getTagLibraryInfos

(5)　アノテーションの使用

アプリケーションサーバでは，JSP 2.1 仕様で規定されたアノテーションをサポートします。アノテーションの使用については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「12.　アノテーションの使用」を参照してください。

(6)　タグファイルのバージョンの決定方法

JSP 2.1 仕様でタグファイルに記述できる要素が追加されたため，タグファイルのバージョンの明確化が必要になりました。そのため，アプリケーションサーバでは，JSP 2.1 仕様に従ってタグファイルのバージョンを決定します。

ただし，タグファイルのバージョンはタグファイルのファイル単位で一致する必要があります。異なるバージョンの TLD ファイルを使用して同一のタグファイルを実行した場合の動作は保証されません。

なお，TLD ファイルや implicit.tld など，バージョン決定要素となるもののバージョンが異なっていても，実際の使用方法で決定するバージョンがファイル単位で一致すれば問題なく動作します。

例えば，次のように TLD ファイルと implicit.tld のバージョンが異なっていても，常に TLD を使ってタグファイルを使用する場合，または常に JSP からディレクトリを指定して使用する場合は問題なく動作します。

表 6-28　バージョン決定要素のバージョンが異なっている例

バージョン決定要素	バージョン
タグファイルを参照する TLD ファイル	2.1
タグファイルを配置しているディレクトリの implicit.tld	2.0

(7)　タグの属性に指定できる要素について

JSP 2.1 仕様では，タグの属性の設定で rtexprvalue が false の場合，deferredValue または deferredMethod が true のときでも，タグの属性にスクリプティング要素の式を指定できません。

アプリケーションサーバでは，タグの属性の設定で rtexprvalue が false の場合でも，deferredValue または deferredMethod が true のときは，タグの属性にスクリプティング要素の式を指定できます。アプリケーションサーバでのタグの属性の設定と指定できる要素の対応について，次の表に示します。

表 6-29　タグの属性の設定と指定できる要素の対応

タグの属性の設定		指定できる要素			
rtexprvalue	deferredValue または deferredMethod	文字列	式(<%= %>)	EL(${})	EL(#{})
false	false	○	×	×	×
true	false	○	○	○	×
false	true	○	○	×	○
true	true	○	○	○	○

（凡例）○：指定できる　×：指定できない

(8)　エラーページの参照

JSP ページのエラー発生時の遷移先として page ディレクティブの errorPage 属性に自分自身を指定している場合の動作を次に示します。

表 6-30　JSP ページのエラー発生時の遷移先として page ディレクティブの errorPage 属性に自分自身を指定している場合の動作

Servlet 仕様/JSP 仕様のバージョン	内容
Servlet 2.5/JSP 2.1	JSP 2.1 仕様と同様にトランスレーションエラーとなります※。
Servlet 2.4/JSP 2.0	JSP にアクセスし例外が発生した場合に，例外に対処しないで，例外が際限なく発生すると，スタックオーバーフローエラーが発生するおそれがあります。

注※

エラー発生時の遷移先をファイル名だけで指定した場合，およびコンテキストルートからの相対パスを指定した場合以外はトランスレーションエラーとなりません。

(a)　トランスレーションエラーとなるエラー発生時の遷移先の指定例

次のようなファイル構成の場合に，トランスレーションエラーとなる，JSP ページのエラー発生時の遷移先の指定例について説明します。

```
├─jsp
│   └example.jsp
└─WEB-INF
    └web.xml
```

example.jsp の JSP ページのエラー発生時の遷移先に，次のどちらかの方法で自分自身（example.jsp）を指定すると，トランスレーションエラーとなります。

- ファイル名だけで自分自身を指定する
  指定例：<%@ page isErrorPage="true" errorPage="example.jsp" %>
- コンテキストルートからの相対パスで自分自身を指定する
  指定例：<%@ page isErrorPage="true" errorPage="/jsp/example.jsp" %>

(b)　異なるページ間で相互にエラー発生時の遷移先に指定していた場合に発生するエラー

次のような JSP ページがあるとします。

- JSP ページ A：エラー発生時の遷移先を JSP ページ B に設定している。
- JSP ページ B：エラー発生時の遷移先を JSP ページ A に設定している。

JSP ページ A でエラーが発生し，JSP ページ B でもエラーが発生した場合，例外が発生して JSP ページ A に遷移します。ここに示した構造で例外が際限なく発生した場合, スタックオーバーフローエラーとなります。

(9)　拡張子以外同じ名前のタグファイルの扱い

タグファイルには.tag と.tagx の 2 種類の拡張子があります。example.tag と example.tagx など, 拡張子だけが異なる同じ名前のタグファイルは配置しないでください。

JSP2.1 仕様では，拡張子だけが異なる同じ名前のタグファイルが配置されている場合，タグライブラリが無効化されます。

アプリケーションサーバでは，拡張子だけが異なる同じ名前のタグファイルが配置されていても，タグライブラリは無効化されません。ただし, タグファイルはファイルシステムから検索されたファイルが使用されます。拡張子だけが異なる同じ名前のタグファイルの場合は，先に検索されたファイルが使用され，ファイルの検索順序については保証されないため，注意してください。

(10)　API の仕様変更

アプリケーションサーバでは，JSP 2.1 仕様での API の仕様変更に従って API の仕様を変更します。javax.servlet.jsp.JspException の動作変更の詳細については「6.2.13(2) javax.servlet.ServletException，および javax.servlet.jsp.JspException で生成したオブジェクトに対する initCause(Throwable)の呼び出しについて」を参照してください。

なお，Web アプリケーションのクラスで総称を使用するようになった API について，JSP 2.0 以前に準拠しているタイプセーフではないメソッドを使用している場合，メソッドの動作に変更はなく，08-00 以降でもそのままクラスを使用できます。なお, コンパイル時に javac コマンドで警告メッセージ（unchecked warning）が発生するようになりますが, 動作に影響する警告メッセージではありません。警告メッセージが発生した原因の個所にアノテーション@SuppressWarnings("unchecked")を適用することで警告メッセージが発生しなくなります。

(11)　JSP ページおよびタグファイルの文字エンコーディングの設定方法

アプリケーションサーバでは，JSP 2.1 仕様に従って，JSP ページ，および標準シンタックスのタグファイルの文字エンコーディングの設定方法に BOM の付加での設定を追加します。

JSP ページについては，Web アプリケーションが準拠する JSP 仕様に従って制御されます。タグファイルの場合は，タグファイルのバージョンに対応する JSP 仕様に従って制御されます。JSP ページおよびタグファイルの文字エンコーディングの判別方法について，次の表に示します。

表 6-31　JSP ページおよびタグファイルの文字エンコーディングの判別方法

JSP 仕様のバージョン	内容
JSP 2.1	JSP 2.1 仕様に従って文字エンコーディングが設定されます。ただし，UTF-32 の BOM はサポートされません。
JSP 2.0	BOM の付加での文字エンコーディングの設定はできません。 文字エンコーディングの指定が必要な場合は，web.xml の<jsp-property-group>要素内の<page-encoding>要素または page，tag ディレクティブでの pageEncoding 属性で指定できます。

JSP ファイルおよびタグファイルの文字エンコーディングの詳細は「6.2.6(13)　JSP でサポートしている文字エンコーディングについて」を参照してください。

Servlet 2.5 仕様以降に準拠した Web アプリケーションの JSP ページおよびタグファイルでは，BOM の付加で文字エンコーディングを設定するかどうかを次のどちらかの方法で指定できます。

- **簡易構築定義ファイルを使用した指定方法**
  論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で，パラメタ webserver.jsp.jsp_page.bom.enabled の値に BOM の付加での文字エンコーディングを有効にする場合は true を，無効にする場合は false を指定します。
- **cjjspc コマンドを使用した指定方法**
  cjjspc コマンドに BOM の付加での文字エンコーディングを有効にする-jsppagedisablebom オプションを指定して JSP 事前コンパイルを実施します。

なお，Servlet 2.5 仕様以降に準拠した Web アプリケーションの JSP ページおよびタグファイルで，BOM の付加での文字エンコーディングを設定しない場合，Servlet 2.4 仕様に従って JSP ページの文字エンコーディングが設定されます。

(12)　TLD ファイルと URI のマッピングについて

TLD ファイルと URI のマッピングについては，Web アプリケーションが準拠する JSP 仕様に従って制御されます。JSP 仕様のバージョンごとの TLD ファイルと URI のマッピング方法を次の表に示します。

表 6-32　TLD ファイルと URI のマッピング方法

Version	マッピング方法
Servlet 2.5（JSP 2.1）	JSP 2.1 仕様に従って，JSTL および JSF の URI を自動的にマッピングします。マッピングする URI は以下のとおりです。 • http://java.sun.com/jsp/jstl/core • http://java.sun.com/jsp/jstl/xml • http://java.sun.com/jsp/jstl/fmt • http://java.sun.com/jsp/jstl/sql • http://java.sun.com/jsp/jstl/functions • http://java.sun.com/jstl/core • http://java.sun.com/jstl/xml • http://java.sun.com/jstl/fmt • http://java.sun.com/jstl/sql • http://java.sun.com/jstl/core_rt

Version	マッピング方法
Servlet 2.5（JSP 2.1）	• http://java.sun.com/jstl/xml_rt • http://java.sun.com/jstl/fmt_rt • http://java.sun.com/jstl/sql_rt • http://java.sun.com/jsf/core • http://java.sun.com/jsf/html
Servlet 2.4（JSP 2.0）	自動的にマッピングしません。JSTL，JSF を使用する場合，通常の TLD ファイルと同様に，JSTL，JSF 仕様の TLD ファイルを配置してください。

Servlet 2.5 仕様以降に準拠した Web アプリケーションでは，自動的なマッピングが最優先で処理されるため，TLD ファイルと URI のマッピングの上書きができません。

そのため，アプリケーションサーバでは，TLD ファイルと URI の自動的なマッピングをするかどうかを次のどちらかの方法で指定できます。

- **簡易構築定義ファイルを使用した指定方法**
  論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で，パラメタ webserver.jsp.tld.mapping.java_ee_tag_library.enabled の値に，自動的なマッピングを有効にする場合は true を，無効にする場合は false を指定します。
- **cjjspc コマンドを使用した指定方法**
  cjjspc コマンドに自動的なマッピングを無効にする-nojavaeetaglib オプションを指定して JSP 事前コンパイルを実施します。

Web アプリケーションごとに複数のバージョンのタグライブラリを使用したい場合は，これらの方法で，自動的なマッピングを無効に設定してください。自動的なマッピングが無効になると，それぞれの Web アプリケーションにライブラリを配置することで複数のバージョンのライブラリを混在させることができます。

(13)　EL のエスケープシーケンス

JSP 2.1 仕様では，EL の開始を示す文字に"#{"が追加されました。

JSP ページおよび JSP ドキュメントについては，Web アプリケーションが準拠する JSP 仕様に従って制御されます。タグファイルの場合は，タグファイルのバージョンに対応する JSP 仕様に従って制御されます。また，EL の設定を有効にしているかどうかで制御が異なります。

"#"を文字列として表すエスケープシーケンスへの制御について，サーブレットおよび JSP 仕様のバージョンごとに次の表に示します。

表 6-33　#を文字列として表すエスケープシーケンスへの制御

サーブレット仕様/JSP 仕様のバージョン	仕様			
	EL の設定を有効にしている場合		EL の設定を無効にしている場合	
	¥$の出力	¥#の出力	¥$の出力	¥#の出力
Servlet 2.5/JSP 2.1	$	#	$	#
Servlet 2.4/JSP 2.0	$	¥#	$	¥#

Servlet 2.5 仕様に準拠した Web アプリケーションの場合，"¥#"は"¥$"と同様に条件に関係なくエスケープシーケンスによって#と出力されます。そのため，"¥#"と出力したい場合は，"¥¥#"と記述する必要があります。

JSP 2.0 仕様の"¥$"のエスケープシーケンスの仕様については「6.2.8(15)　EL（Expression Language）のエスケープシーケンスについて」を参照してください。

(14)　EL での Enum 型に対する処理の変更

JSP 2.1 仕様の EL から，Java SE 5 仕様で規定された Enum 型のオブジェクトに対応した処理が追加されました。

JSP ページおよび JSP ドキュメントについては，Web アプリケーションが準拠する JSP 仕様に従って制御されます。タグファイルの場合は，タグファイルのバージョンに対応する JSP 仕様に従って制御されます。

アプリケーションサーバでのサーブレットおよび JSP 仕様のバージョンごとの EL での Enum 型に対する処理について，次の表に示します。

表 6-34　EL での Enum 型に対する処理

サーブレット仕様/JSP 仕様のバージョン	内容
Servlet 2.5/JSP 2.1	JSP 2.1 仕様に従って処理されます。
Servlet 2.4/JSP 2.0	JSP 2.0 仕様に従って，Enum 型であってもほかのオブジェクトと同様に処理されます。

ただし，JSP 2.1 仕様で非推奨となった，JSP 2.0 仕様で規定されている EL の API を使用した場合，Web アプリケーションのバージョンに関係なく JSP 2.0 仕様の EL の機能範囲で処理されます。

(15)　EL での<，>，lt，gt 演算子の処理の変更

JSP ページおよび JSP ドキュメントについては，Web アプリケーションが準拠する JSP 仕様に従って制御されます。タグファイルの場合は，タグファイルのバージョンに対応する JSP 仕様に従って制御されます。

アプリケーションサーバでの EL での<，>，lt，gt 演算子の処理について，サーブレットおよび JSP 仕様のバージョンごとに次の表に示します。

表 6-35　EL での<，>，lt，gt 演算子の処理

サーブレット仕様/JSP 仕様のバージョン	内容
Servlet 2.5/JSP 2.1	JSP 2.1 仕様に従って，<，>，lt，gt 演算子が処理されます。
Servlet 2.4/JSP 2.0	JSP 2.0 仕様に従って，<，>，lt，gt 演算子が処理されます。

ただし，JSP 2.1 仕様で非推奨となった，JSP 2.0 仕様で規定されている EL の API を使用した場合，Web アプリケーションのバージョンに関係なく JSP 2.0 仕様の EL の機能範囲で処理されます。

(16)　EL の API の変更

アプリケーションサーバでの JSP 仕様の EL の API について説明します。

JSP 2.1 仕様で追加となった API に対応します。JSP 2.1 仕様で追加された EL の機能を使用する場合，javax.el パッケージの API を使用してください。

JSP 2.0 仕様で規定された EL の API を使用した場合，JSP 2.0 仕様で規定された EL の仕様に従って EL が評価されます。

6.2.8　JSP 2.0 仕様で追加，変更された仕様についての注意事項

JSP 2.0 仕様で追加，変更された仕様を，アプリケーションサーバ上で利用するときの注意事項を示します。JSP 2.0 仕様および JSP 1.2 仕様については，それぞれの仕様書（JSP 2.0 仕様書，JSP 1.2 仕様書）を参照してください。

(1)　JSP ドキュメントのデフォルト拡張子

JSP 2.0 仕様では，JSP ドキュメントの標準的な拡張子を「jspx」としています。アプリケーションサーバで利用する Web コンテナでは，「jspx」を拡張子としたファイルは，デフォルトマッピングによって web.xml に URL マッピング定義しなくても JSP ドキュメントとして扱われます。

(2)　タグファイルの Java ソースファイルとクラスファイルの出力先

タグファイルは，JSP ファイルと同様に，JSP のコンパイルによって Java ソースファイルとクラスファイルが生成されます。Java ソースファイルおよびクラスファイルは，JSP コンパイル結果の出力先ディレクトリに出力されます。

JSP コンパイル結果の出力先ディレクトリは変更できます。なお，生成される Java ソースファイル，およびクラスファイルのパスが OS の上限を超える場合は，出力先ディレクトリを変更する必要があります。

JSP コンパイル結果の出力先ディレクトリについては，JSP 事前コンパイル機能を使用している場合，「2.5.5(2)　JSP コンパイル結果の出力先」，JSP 事前コンパイル機能を使用していない場合，「2.5.6(3)　JSP コンパイル結果の出力先」を参照してください。

(3)　JSP EL 式の評価 API の複数指定

JSP 2.0 仕様では，EL 式の構文解析と評価をする API として次の API が提供されます。

- javax.servlet.jsp.el.ExpressionEvaluator クラスの evaluate メソッド
- javax.servlet.jsp.el.Expression クラスの evaluate メソッド

JSP 2.0 仕様では，これらの API から複数の EL 式を指定することはできませんが，アプリケーションサーバでは，複数の EL 式を指定できます。

(4)　XML シンタックスで記述された JSP ファイルとタグファイル

- **文字エンコードについて**

  Web アプリケーションのバージョン 2.4 で，JSP ドキュメントの文字エンコードを指定する場合は，XML 宣言で文字エンコードを指定してください。

  JSP 1.2 仕様の場合は，ファイルの文字エンコードは，page ディレクティブの pageEncoding 属性，または contentType 属性の charset 値で指定していましたが，JSP 2.0 仕様からは XML 宣言で文字エンコードを指定するように変更されています。

- **標準アクションの接頭辞ついて**

  JSP の標準アクションは，XML 名前空間の「http://java.sun.com/JSP/Page」で識別されます。したがって，XML 名前空間の接頭辞で標準アクションを指定する必要があります。接頭辞を jsp とした場合の記述例を示します。

```
<?xml version="1.0" ?>
<jsp:root
    xmlns:jsp=http://java.sun.com/JSP/Page
    version="2.0">
    <jsp:directive.page import="java.util.*"/>
    <jsp:useBean id="name" class="test.Bean"/>

</jsp:root>
```

- <jsp:root>要素の扱いについて

  JSP 2.0 仕様から<jsp:root>要素をルート要素に指定していなくても，XML シンタックスの JSP ファイル，またはタグファイルとして扱われます。

  JSP 1.2 仕様では，JSP ドキュメントの条件として<jsp:root>がルート要素に指定されている必要がありましたが，JSP 2.0 仕様からは，<jsp:root>が指定されていなくても，web.xml に定義した<jsp-config><jsp-property-group><is-xml>の値が true，または拡張子が jspx，tagx であれば，XML シンタックス形式の JSP と扱うように変更されています。

(5)　page ディレクティブの isThreadSafe 属性の非推奨化について

page ディレクティブの isThreadSafe 属性は，javax.servlet.SingleThreadModel インタフェースが非推奨となったことによって，JSP 2.0 仕様では非推奨とされています。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，page ディレクティブの isThreadSafe 属性を使用できます。ただし，Servlet 2.4 仕様で，javax.servlet.SingleThreadModel インタフェースが非推奨となった理由に注意して使用してください。

(6)　JSP ドキュメントの HTTP レスポンスの ContentType のデフォルト値について

JSP 2.0 仕様では，JSP ドキュメントを使用した場合に，デフォルトの ContentType の値が「text/xml」になると追記されています。

アプリケーションサーバでは，JSP 2.0 以降の場合は「text/xml」，JSP 1.2 の場合は「text/html」をデフォルトとして動作します。

(7)　タグライブラリ・ディスクリプタ（TLD ファイル）の配置について

JSP 2.0 仕様では，タグライブラリ・ディスクリプタの配置場所についての規定が追加されています。

アプリケーションサーバでは，配置するディレクトリによって，JSP のコンパイル時に KDJE39289-W のメッセージが出力されることがあります。ただし，エラーとはならないで Web アプリケーションが実行されます。

メッセージが出力される条件を次に示します。

配置するディレクトリ

- /WEB-INF ディレクトリ以下以外
- /WEB-INF/classes ディレクトリ以下
- /WEB-INF/lib ディレクトリ以下

メッセージが出力されるタイミング

- 該当するタグライブラリ・ディスクリプタを，web.xml の<taglib><taglib-location>タグに指定して，Web アプリケーション開始するとき
- 該当するタグライブラリ・ディスクリプタを，タグライブラリの宣言で直接指定して使用する JSP をコンパイルするとき

(8)　javax.servlet.jsp.tagext.PageData クラスの getInputStream メソッドで取得できる XML ビュー情報について

javax.servlet.jsp.tagext.PageData オブジェクトの getInputStream メソッドで取得できる XML ビュー情報の仕様が，JSP 2.0 仕様で変更されています。getInputStream メソッドは，javax.servlet.jsp.tagext.TagLibraryValidator クラスの validate メソッドの第三引数に指定して使用されます。

アプリケーションサーバでの変更点を，JSP 2.0 以降と JSP 1.2 に分けて示します。

(a)　jsp:id 属性

JSP 2.0 以降

jsp:id 属性を付加します。

JSP 1.2

jsp:id 属性を付加しません。

(b)　XML ビューの文字エンコード

JSP 2.0 以降

XML ビューの文字エンコードを常に UTF-8 とし，文字コードを UTF-8 として XML 宣言を出力します。

JSP 1.2

XML ビューの文字エンコードを常に UTF-8 とし，文字コードを UTF-8 として XML 宣言を出力しません。

(c)　page ディレクティブの pageEncoding 属性

JSP 2.0 以降

pageEncoding 属性の値を UTF-8 に設定します。pageEncoding 属性がない場合は，pageEncoding 属性を追加します。

JSP 1.2

pageEncoding 属性の値は変更しません。

(d)　page ディレクティブの contentType 属性

JSP 2.0 以降

contentType 属性の値に ServletResponse クラスの setContentType メソッドで設定された値を設定します。contentType 属性がない場合は，contentType 属性を追加します。

JSP 1.2

contentType 属性の値は変更しません。

(9)　include ディレクティブでインクルードされるファイルのデフォルトの文字コードについて

JSP 2.0 仕様では，page ディレクティブの pageEncoding 属性は，pageEncoding 属性を記述したファイルにだけ適用されることが追記されています。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，include ディレクティブでファイルをインクルードするときに，インクルード先のファイルに文字コードの指定がないと，インクルード元の文字コードがインクルード先のファイルに適用されます。

(10)　JSP ドキュメント内の矛盾する文字コードについて

JSP ドキュメントでの XML 宣言に指定する文字コードと，JSP ドキュメントでの page ディレクティブの pageEncoding 属性に指定する文字コードが異なる場合の仕様が JSP 2.0 仕様では追記されています。JSP 1.2 仕様には記述がありません。

アプリケーションサーバで文字コードが異なる場合の制御を，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

トランスレーションエラーとなります。

JSP 1.2

page ディレクティブの pageEncoding 属性が使用されます。

(11)　JSP ドキュメントでの HTTP レスポンスの文字コードのデフォルト値について

JSP ドキュメントで page ディレクティブの contentType 属性がない場合や，属性に CHARSET 値がない場合に使用される HTTP レスポンスのデフォルトの文字コードが JSP 2.0 仕様では追記されています。

アプリケーションサーバでのデフォルト値を，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

UTF-8 が使用されます。

JSP 1.2

ISO-8859-1 が使用されます。

(12)　page ディレクティブの pageEncoding 属性の複数回指定について

page ディレクティブの pageEncoding 属性の複数回指定について，JSP 2.0 仕様では仕様が変更されています。

JSP 2.0 仕様では，トランスレーション単位（JSP と include ディレクティブでインクルードされるファイル）で pageEncoding 属性の複数回指定ができるようになっています。また，同じ JSP ファイル内で pageEncoding 属性の複数回指定をするとコンパイルエラーとなることが追記されています。

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，トランスレーション単位で pageEncoding 属性の複数回指定ができます。このとき，ファイル単位に指定した値が該当するファイルに適用されます。また，同じ JSP ファイル内での pageEncoding 属性の複数回指定については，JSP 2.0 以降と JSP 1.2 で仕様が異なります。アプリケーションサーバでの仕様を，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

一つのファイルに 1 回だけ指定できます。複数回指定した場合は，コンパイルエラーとなります。

JSP 1.2

一つのファイルに複数回指定できます。最初に記述した値が適用されます。

(13)　JSP ドキュメントのタグライブラリの宣言で taglib マップに登録されていない uri を記述した場合について

JSP ドキュメントで namespace を使ってタグライブラリを宣言し，指定した uri が taglib マップ（uri とタグライブラリ・ディスクリプタのマッピング）で見つからない場合の動作について，JSP 2.0 仕様では追記されています。

アプリケーションサーバでの動作を，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

指定した uri が taglib マップに登録されていない場合，uri の namespace で定義されたアクションは，解析しないで扱われます（テキストとして出力されます）。

JSP 1.2

- uri が絶対 URI の場合
  トランスレーションエラーとなります。
- uri が絶対 URI 以外の場合
  Web アプリケーション内のパスとして TLD ファイル（タグライブラリ・ディスクリプタ）を検索して使用されます。TLD ファイルがない場合は，トランスレーションエラーとなります。

(14)　JSP ドキュメントの文字コードについて

JSP ドキュメントでのファイルの文字コードの決定方法について，JSP 2.0 仕様で仕様が変更されています。

アプリケーションサーバでの文字コードの決定方法を，JSP 2.0 以降と JSP 1.2 に分けて示します。

JSP 2.0 以降

XML 1.0 の仕様に従い，XML 宣言に従います。XML 宣言がない場合はデフォルト値の UTF-8 となります。

JSP 1.2

page ディレクティブの pageEncoding 属性に従います。pageEncoding 属性がない場合は contentType 属性の charset=で指定した文字コードに従います。どちらもない場合はデフォルト値の ISO-8859-1 になります。

(15)　EL（Expression Language）のエスケープシーケンスについて

JSP 2.0 仕様である EL の開始を示す"${"に含まれる"$"を文字列として表すエスケープシーケンスについて，JSP 仕様と，アプリケーションサーバで利用する Web コンテナの仕様を次に示します。

アプリケーションサーバで利用する Web コンテナでは，"¥$"はエスケープシーケンスによって，"$"と出力されます。"¥$"と出力したい場合は，"¥¥$"と記述します。

"¥$"と記述した場合の動作を，JSP 2.0 と JSP 1.2 に分けて示します。

JSP 2.0

JSP 2.0 仕様では，EL の設定を無効に設定している場合，"$"は EL の開始文字とする必要がなく，"¥"は制御コードとしては扱われません。JSP 2.0 で動作する場合は，EL の設定を有効にしているかどうかによって，"¥$"の出力結果が異なります。JSP 2.0 で動作する場合の"¥$"の出力結果を次の表に示します。

表 6-36　JSP 2.0 で動作する場合の"¥$"の出力結果

EL の設定の有効/無効	仕様	出力結果
有効	JSP 2.0 仕様	"$"
	アプリケーションサーバで利用する Web コンテナ	"$"
無効	JSP 2.0 仕様	"¥$"

EL の設定の有効/無効	仕様	出力結果
無効	アプリケーションサーバで利用する Web コンテナ	"$"

EL の設定を無効にする場合は，次に示すどれかの方法で設定します。

- page ディレクティブの isELIgnored 属性に true を指定する。
- tag ディレクティブの isELIgnored 属性に true を指定する。
- web.xml の<el-ignored>タグに true を指定する。

JSP 1.2

JSP 1.2 仕様では"$"は予約語ではありません。"¥"は制御コードとしては扱われないため，"¥$"と出力されます。

アプリケーションサーバで利用する Web コンテナでは，JSP1.2 で動作する場合でも"¥"は制御コードとして扱われるため，"$"と出力されます。ただし，JSP ドキュメント形式の属性値で使用する場合は，"¥$"と出力されます。

JSP 1.2 で動作する場合の"¥$"の出力結果を次の表に示します。

表 6–37　JSP 1.2 で動作する場合の"¥$"の出力結果

仕様	出力結果
JSP 1.2 仕様	"¥$"
アプリケーションサーバで利用する Web コンテナ	"$"

(16)　EL の評価結果の型について

カスタムタグの属性に指定した EL の評価結果の型について，JSP 仕様と，アプリケーションサーバの仕様を次に示します。

JSP 2.0 仕様

EL の評価結果は，カスタムタグの属性の期待する型に変換されます。

アプリケーションサーバ

EL の評価結果は，カスタムタグの属性に対応するセッターメソッドの引数の型に変換されます。TLD の属性の定義に記述された type 要素は型の変換に使用しません。

EL の記載個所がタグファイルの場合，attribute ディレクティブの type 属性に指定した型に変換されます。

EL の評価結果の型が JSP 仕様とアプリケーションサーバで異なる場合の例について次に示します。

例

- カスタムタグの属性名：attr
- カスタムタグのセッターのシグニチャ：void setAttr(java.lang.String hoge)
- TLD での attr 属性の type 要素の値：java.lang.Integer

この例の場合，EL の評価結果の型は次に示す型になります。

JSP 2.0 仕様

java.lang.Integer に変換されます。

アプリケーションサーバ

java.lang.String に変換されます。

6.2.9　JSP 1.2 仕様の JSP 実装時の注意事項

JSP 1.2 仕様で JSP を実装する場合の注意事項について説明します。JSP 2.0 以降の JSP を実装する場合は，該当しません。

(1)　JSP ドキュメント利用時の注意事項

JSP ドキュメントを利用するときの注意事項を次に示します。

- JSP ドキュメントでは，<jsp:root>タグの xmlns:jsp 属性に JSP 1.2 仕様で規定された正しい値を指定してください。JSP 1.2 仕様で規定された値以外を指定した場合，無視されます。
- JSP ドキュメントの<jsp:root>タグの version 属性には，必ず「1.2」を指定してください。
- JSP ドキュメントでは，タグの属性に JSP 1.2 仕様で規定された正しい属性だけを指定してください。タグに不正な属性を指定した場合，無視されます。
- JSP ドキュメントでは，JSP 1.2 仕様で必須と規定されているタグの属性は必ず指定してください。必須と規定されている属性を省略した場合の動作は保証されません。
- JSP ドキュメントで，子タグとして使用できないタグを子タグとして記述しないでください。子タグとして使用できないタグを子タグとして記述した場合の動作は保証されません。
- 文字エンコーディングが UTF-8 で BOM が付加された JSP ドキュメントは使用できません。JSP ドキュメントの文字エンコーディングに UTF-8 を使用する場合，BOM は付加しないでください。JSP ドキュメントの文字エンコーディングが UTF-8 であり，BOM が付加されている場合，JSP コンパイル時にエラーとなります。

(2)　page ディレクティブの pageEncoding 属性利用時の注意事項

JSP ページの page ディレクティブまたは JSP ドキュメントの<jsp:directive.page>タグの pageEncoding 属性は正しい値を一度だけ記述してください。

ただし，JSP ページの page ディレクティブまたは JSP ドキュメントの<jsp:directive.page>タグの pageEncoding 属性を複数記述した場合でも，エラーにはなりません。

(3)　標準アクション利用時の注意事項

JSP 1.2 仕様で規定されたタグ以外の「jsp:」で始まるタグは記述できません。

(4)　カスタムタグ利用時の注意事項

タグ内に同一の属性を複数定義しないでください。JSP ページでは，タグ内に同一の属性を複数定義した場合の動作は保証されません。

(5)　タグライブラリ利用時の注意事項

プロパティに webserver.xml.validate=false を設定した場合，タグライブラリ・ディスクリプタ（TLD ファイル）の検証は実施されません。仕様（XML のスキーマ定義）に従っていない TLD ファイルを使用した場合，エラーとならないで動作することがありますが，動作は保証されません。

TLD ファイルは仕様（XML のスキーマ定義）に従って正しく記述してください。

(6)　plugin アクション利用時の注意事項

JSP ページでの plugin アクションまたは JSP ドキュメントでの<jsp:plugin>タグの type 属性に，JSP 1.2 仕様で規定された値以外は指定しないでください。JSP 1.2 仕様で規定された値以外を指定すると，出力される type 属性の値が空文字列となります。

(7)　params アクション利用時の注意事項

必須タグとして<jsp:param>タグ要素を記述するように規定されているため，params アクションを記述した場合，または<jsp:params>タグを記述した場合は，必ず<jsp:param>タグ要素を記述してください。

ただし，JSP ドキュメントの JSP ページで params アクションを記述した場合，または JSP ドキュメントの<jsp:params>タグを記述した場合に，<jsp:param>タグ要素を記述していないときでも，エラーにはなりません。

6.2.10　EL2.2 仕様で追加，変更された仕様についての注意事項

EL2.2 仕様で追加，変更された仕様を，アプリケーションサーバ上で使用するときの注意事項を示します。EL2.2 仕様については，EL2.2 仕様書を参照してください。

- アプリケーションサーバでは EL2.2 をサポートしますが，EL2.1 の処理系と EL2.2 の処理系の両方を保持します。使用する処理系は簡易構築定義ファイルの次のパラメタで切り替えられます。このパラメタは簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に指定します。

  ```
  webserver.jsp.el2_2.enabled
  ```

  webserver.jsp.el2_2.enabled パラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.14.1　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。
- 一つの Bean 内に同じ数のパラメタを持つメソッドが複数定義された場合，最初に定義されたメソッドに型変換されます。そのため，メソッドを呼び出すときの引数の型が，最初に定義されたメソッドに合う場合は正しく呼び出されますが，引数の型が合わない場合は ELException が発生します。

6.2.11　既存の Web アプリケーションを Servlet 3.0 仕様にバージョンアップする場合の留意点

Servlet 2.5 仕様に対応した Web アプリケーションを Servlet 3.0 仕様にバージョンアップする場合に必要な作業，および注意事項を示します。なお，Servlet 3.0 仕様の詳細については，Servlet 3.0 仕様書を参照してください。

(1)　Servlet 3.0 仕様，および JSP 2.2 仕様で追加された仕様，および変更された仕様について

Servlet 3.0 仕様で追加，変更された仕様についての注意事項については，「6.2.3　Servlet 3.0 仕様で追加，変更された仕様についての注意事項」を参照してください。

JSP 2.2 仕様で追加された仕様はサポートされません。ただし，JSP 2.2 の web.xml のスキーマには対応するため，web.xml に JSP 2.2 で追加されたタグを追加してもエラーとなりません。追加されたタグは無視されます。

(2)　web.xml の移行について

web.xml は，Servlet 3.0 仕様に従った定義に修正します。詳細は Java Servlet Specification v3.0 を参照してください。また，仕様変更については「6.2.3　Servlet 3.0 仕様で追加，変更された仕様についての注意事項」を参照してください。

6.2.12　既存の Web アプリケーションを Servlet 2.5 仕様にバージョンアップする場合の留意点

Servlet 2.4 仕様に対応した Web アプリケーションを Servlet 2.5 仕様にバージョンアップする場合に必要な作業，および注意事項を示します。なお，Servlet 2.5 仕様の詳細については，Servlet 2.5 仕様書を参照してください。

(1)　Servlet 2.5 仕様，および JSP 2.1 仕様で追加された仕様，および変更された仕様について

Servlet 2.5 仕様および JSP 2.1 仕様で追加，変更された仕様についての注意事項については，それぞれ「6.2.4　Servlet 2.5 仕様で追加，変更された仕様についての注意事項」または「6.2.7　JSP 2.1 仕様で追加，変更された仕様についての注意事項」を参照してください。

(2)　web.xml の移行について

web.xml は，Servlet 2.5 仕様に従った定義に修正します。詳細は Java Servlet Specification v2.5 を参照してください。また，仕様変更については「6.2.4　Servlet 2.5 仕様で追加，変更された仕様についての注意事項」を参照してください。

(3)　JSP から生成するクラスのサイズについて

Servlet 2.5 仕様に準拠した Web アプリケーションに含まれる JSP から生成したクラスは，JSP 2.1 仕様の機能が適用されるようになります。そのため，JSP から生成した Java ソースファイル，クラスファイルの内容が変わります。

この場合，Java ソースファイルの行数，JSP から生成したクラスに含まれるメソッドのサイズ，または Metaspace 領域の使用量が増加することがあるため，注意してください。

6.2.13　前バージョンから 09-50 へ移行する場合の Web アプリケーションに関する注意事項

前バージョンから 09-50 へ移行する場合の，Web アプリケーションに関する注意事項を示します。

(1)　Web コンテナがサポートするバージョン情報取得 API

次の表に示す Servlet API では，Web コンテナがサポートする Servlet 仕様のバージョン情報を取得できます。

表 6–38　Web コンテナがサポートするバージョン情報取得 API

クラス名	メソッド名
javax.servlet.ServletContext	getMajorVersion
	getMinorVersion

クラス名	メソッド名
javax.servlet.jsp.JspEngineInfo	getSpecificationVersion

Web アプリケーションのバージョンが Servlet 2.5/JSP 2.0 以前であっても，Web コンテナがサポートする Servlet 仕様のバージョンは，Servlet 3.0/JSP 2.1 となります。そのため，これらの Servlet API は Servlet 仕様のバージョンとして Servlet 3.0/JSP 2.1 の情報を返します。なお，サーバの動作モードにも関係なく情報を取得します。

(2)　javax.servlet.ServletException，および javax.servlet.jsp.JspException で生成したオブジェクトに対する initCause(Throwable)の呼び出しについて

次に示すオブジェクトに対して，initCause(Throwable)を呼び出すことができません。

- javax.servlet.ServletException のコンストラクタ ServletException(String, Throwable)および ServletException(Throwable)で生成した ServletException オブジェクト
- javax.servlet.jsp.JspException のコンストラクタ JspException(String, Throwable)および JspException(Throwable)で生成した JspException オブジェクト

initCause(Throwable)を呼び出したい場合は，簡易構築定義ファイルに互換用のプロパティを設定します。設定の方法については，「6.2.1(22)　javax.servlet.ServletException クラスのコンストラクタで指定した根本原因の例外の取得について」を参照してください。

(3)　javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッドを呼び出した場合の動作

javax.servlet.ServletRequest インタフェースの getReader メソッドを呼び出したあとで javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッドを呼び出した場合，getCharacterEncoding メソッドの戻り値が変更されません。詳細については，「6.2.4(8) javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッドの呼び出しの無効について」を参照してください。

(4)　javax.servlet.GenericServlet クラスが保持する ServletConfig オブジェクトが null の場合の動作

javax.servlet.GenericServlet クラスがインスタンス変数として保持する ServletConfig が null の場合，次に示すメソッドによってスローされる例外がバージョンごとに異なります。

- getInitParameter(String)
- getInitParameterNames()
- getServletContext()
- getServletName()

これらのメソッドによってスローされる例外をバージョンごとに次に示します。

07-60 以前

java.lang.NullPointerException

08-00 以降

java.lang.IllegalStateException

例外のメッセージとして「ServletConfig has not been initialized.」が出力されます。

これらの例外は，次の二つの条件を両方満たす場合にスローされます。

- javax.servlet.GenericServlet を継承したサーブレットが init(ServletConfig)をオーバーライドしている場合
- オーバーライドした init(ServletConfig)の中で super.init(ServletConfig)を呼び出していない，または，引数に null を指定して super.init(ServletConfig)を呼び出している場合

(5)　javax.servlet.GenericServlet クラスの init メソッドと destroy メソッドで出力するログ

javax.servlet.GenericServlet クラスの init メソッドと destroy メソッドをオーバーライドしていないサーブレットを初期化または終了すると，サーブレットログにログが出力されます。出力されるログについては「6.2.1(17)　init メソッドおよび destroy メソッドをオーバーライドしていない場合に出力されるメッセージ」を参照してください。

ただし，javax.servlet.GenericServlet クラスがインスタンス変数で持つ ServletConfig が null の場合の動作が 07-60 以前と 08-00 以降で異なります。バージョンごとの動作の差異を次に示します。

07-60 以前

ログ出力に失敗し，java.lang.NullPointerException 例外をスローする。

08-00 以降

例外をスローしない。また，ログも出力しない。

(6)　ライブラリ JAR 内の TLD ファイルの検索

08-00 以降では，ライブラリ JAR に含まれる TLD ファイルを検索し，ライブラリ JAR 内のタグライブラリが使用できます。ライブラリ JAR に TLD ファイルが含まれる場合，以前のバージョンでは使用できなかった TLD ファイルが使用できるため，次に示す動作変更が発生します。

- ライブラリ JAR 内の TLD ファイルの<uri>要素に記述された URI が，web.xml やほかの TLD ファイルに記述された URI と重複する場合，警告メッセージが出力されます。ライブラリ JAR 内の TLD ファイルに記述された URI と TLD ファイルのマッピングは無効となります。
- JSP の taglib ディレクティブの uri 属性で，ライブラリ JAR 内の TLD ファイルに記述された URI を指定していた場合，JSP トランスレーションができるようになります。なお，07-60 以前では JSP トランスレーション時にトランスレーションエラーとなります。

詳細は，「2.3.4　タグライブラリの J2EE アプリケーションへの格納」を参照してください。

(7)　cjjspc コマンドでのクラスパスに指定した JAR ファイル内の TLD ファイルの検索

08-00 以降では，cjjspc コマンドでの JSP コンパイル時に，-classpath オプションでクラスパスに指定した JAR ファイル内の TLD ファイルを検索し，JAR ファイル内のタグライブラリが使用できます。クラスパスに指定した JAR ファイルに TLD ファイルが含まれる場合，07-60 以前では使用できなかった TLD ファイルが使用できるため，次の現象が発生します。

- クラスパスに指定した JAR ファイル内の TLD ファイルの<uri>要素に記述された URI が，web.xml やほかの TLD ファイルに記述された URI と重複する場合，警告メッセージが出力されます。クラスパスに指定した JAR ファイル内の TLD ファイルに記述された URI と TLD ファイルのマッピングは無効となります。

- JSP の taglib ディレクティブの uri 属性で，クラスパスに指定した JAR ファイル内の TLD ファイルに記述された URI を指定していた場合，JSP トランスレーションができるようになります。なお，07-60 以前では JSP トランスレーション時にトランスレーションエラーとなります。

詳細は，「2.3.4　タグライブラリの J2EE アプリケーションへの格納」を参照してください。

(8)　HTTP セッションのセッション ID を示す HTTP Cookie の削除

Web アプリケーション内で HTTP セッションを無効化した場合に，Web クライアントの保持する HTTP Cookie の情報についての動作が 07-60 以前と 08-00 以降で異なります。バージョンごとの動作の差異を次に示します。

08-00 以降

Web クライアントの保持する HTTP Cookie の情報を削除します。また，無効化済みの HTTP セッションに対するセッション ID 送信を抑止します。

07-60 以前

Web クライアントの保持する HTTP Cookie 情報を削除しません。そのため，HTTP セッション無効化後も無効なセッション ID の送信が続く場合があります。

HTTP Cookie の削除については「2.7.4　Web クライアントが保持する無効なセッション ID の削除」を参照してください。

07-60 以前のバージョンから 08-00 以降へアップグレードインストールをした場合，セットアップ済みの J2EE サーバの設定に，次のパラメタの設定が追加されます。

```
webserver.session.delete_cookie.backcompat=true
```

この設定の追加によって，HTTP Cookie 情報を削除しなくなり，07-60 以前と同様の動作になります。パラメタについての詳細はマニュアル「アプリケーションサーバ リファレンス 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。

6.2.14　既存の Web アプリケーションを Servlet 2.4 仕様にバージョンアップする場合の留意点

Servlet 2.2 仕様または Servlet 2.3 仕様に対応した Web アプリケーションを Servlet 2.4 仕様にバージョンアップする場合に必要な作業，および注意事項を示します。なお，Servlet 2.4 仕様の詳細については，Servlet 2.4 仕様書を参照してください。

(1)　web.xml の移行

Servlet 2.2 仕様または Servlet 2.3 仕様に従って作成した web.xml は，Servlet 2.4 仕様に従った定義に修正します。Servlet 2.4 仕様で変更された点については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.5　Servlet 2.4 以降で追加，変更された仕様についての注意事項（web.xml)」を参照してください。

(2)　Servlet 2.4 仕様に対応したコードへの修正

Servlet 2.4 仕様では，Servlet 2.2 仕様および Servlet 2.3 仕様から仕様が追加，変更されています。追加，変更された点を確認し，Servlet 2.4 仕様に対応したコードに修正してください。Servlet 2.4 仕様で追加，変更された点については，「6.2.5　Servlet 2.4 仕様で追加，変更された仕様についての注意事項」を参照してください。

また，JSP 2.0 仕様でも JSP 1.2 仕様から仕様が追加，変更されています。追加，変更された点を確認し，JSP 2.0 仕様に対応したコードに修正してください。JSP 2.0 仕様で追加，変更された点については，「6.2.8　JSP 2.0 仕様で追加，変更された仕様についての注意事項」を参照してください。

(3)　サーブレットエンジンモードから J2EE サーバモードへの移行

Servlet 2.4 仕様に対応した Web アプリケーションは，サーブレットエンジンモードでは実行できません。したがって，サーブレットエンジンモードを使用していた場合は，J2EE サーバモードに移行する必要があります。

サーブレットエンジンモードから J2EE サーバモードへの移行方法については，マニュアル「アプリケーションサーバ 機能解説 互換編」の「3.　サーブレットエンジンモード」を参照してください。

(4)　JSP のシンタックスチェックについて

Servlet 2.4 仕様に対応した Web アプリケーションに含まれる JSP ファイルは，JSP 2.0 仕様に準拠します。JSP 2.0 仕様では, JSP 1.2 仕様よりも厳密にシンタックスチェックされます。したがって, Servlet 2.3 仕様の Web アプリケーションでは通知されなかったエラーが通知されることがあります。

Servlet 2.2 仕様または Servlet 2.3 仕様に対応した Web アプリケーションを Servlet 2.4 仕様にバージョンアップする場合は，cjjspc コマンドで JSP をコンパイルして，エラーが発生しないことを確認してください。コンパイルエラーが通知された場合は，エラー通知内容に従って修正してください。

cjjspc コマンドについては，「2.5.2　JSP 事前コンパイルの方法」を参照してください。

6.2.15　サーブレットでのアノテーションの使用

アプリケーションサーバでは, サーブレットでアノテーションを使用できます。アプリケーションサーバが対応しているアノテーションについては，マニュアル「アプリケーションサーバ リファレンス API 編」の「2.　アプリケーションサーバが対応しているアノテーションおよび Dependency Injection」を参照してください。

6.2.16　JavaVM のメソッドサイズ制限についての注意事項

JavaVM では，64KB を超えるメソッドが存在すると，クラスファイル生成時にエラーとなるか，クラスのロード時に java.lang.LinkageError 例外が発生します。そのため，1 メソッドのバイトコードは 64KB 以内のサイズにする必要があります。

また，64KB 以内であっても，非常に大きいサイズのメソッドが存在する場合は，次のような弊害が発生するおそれがあります。

- GC 処理の実行に非常に時間が掛かる。
- JIT コンパイルに非常に時間が掛かる。
- JIT コンパイルに非常に多くのメモリを消費する。

Web アプリケーションでは，自動生成される java ソースコードによって，1 メソッドのバイトコードが 64KB を超える場合があります。java ソースコードの自動生成と，メソッドサイズが大きい場合の見直し方法について説明します。

(1)　java ソースコードの自動生成

java ソースコードの自動生成について説明します。

- JSP 仕様での自動生成

  JSP 仕様では，JSP ファイル，またはタグファイルに記述した内容から，_jspService メソッドまたは doTag メソッド内に java ソースコードが自動生成されます。

- アプリケーションサーバでのカスタムタグ使用時の自動生成

  アプリケーションサーバでは，カスタムタグを使用している場合，カスタムタグの処理やボディに記述した内容がメソッド化され，java ソースコードが自動生成されます。

  なお，カスタムタグをメソッド化できるのは，カスタムタグの処理またはボディに含まれるすべてのカスタムタグが次の条件を満たす場合です。

  - 属性やボディがスクリプトレスである。
  - スクリプト変数を定義していない。

(2)　メソッドサイズが大きい場合の見直し方法

自動生成された java ソースコードのメソッドの行数が，コメントおよび空行を含めて 1000 行を超える場合，KDJE39231-W および KDJE39333-W のメッセージが出力されます。

メッセージが出力された場合は，JSP ファイル，タグファイル，またはカスタムタグのボディの内容を見直してください。

見直す個所ごとに，見直す方法を次に示します。

- **JSP ファイルの内容のサイズが大きい場合**

  動的インクルード（インクルードアクション）で JSP ファイルを分割する。

- **タグファイルの内容のサイズが大きい場合**

  次のどちらかを実施します。

  - JSP ファイルから使用するタグファイルを複数のタグファイルに分割する。
  - タグファイルから別のタグファイルを呼び出すように分割する。

- **カスタムタグのボディのサイズが大きい場合**

  動的インクルード（インクルードアクション）でカスタムタグを分割する。

6.3　JSP 移行時の注意事項

この節では，JSP 移行時の注意事項について説明します。

07-00 より前のバージョンと 07-00 以降のバージョンでは，JSP のコンパイルの動作が異なります。アプリケーションサーバでは，JSP トランスレーション時に，JSP 仕様に従って JSP の内容がチェックされるため，JSP トランスレーション時にエラーになって移行できないことがあります。その場合，**JSP トランスレーション下位互換機能**を使用すると，07-00 より前のバージョンと 07-00 以降のバージョンで JSP のコンパイルの動作を同じように設定できて，エラーが発生しないようにできます。

JSP トランスレーション下位互換機能の定義は，簡易構築定義ファイルで指定します。指定内容については，以降の各項を参照してください。また，JSP トランスレーション下位互換機能の定義は，cjjspc コマンドを実行するときのオプションにも指定できます。cjjspc コマンドを実行するときにオプションに指定する方法については，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。

この節の構成を次の表に示します。

表 6-39　この節の構成（JSP 移行時の注意事項）

タイトル	参照先
カスタムタグのスクリプト変数の定義に関する注意事項	6.3.1
<jsp:useBean>タグの class 属性に関する注意事項	6.3.2
タグの属性値の Expression チェックに関する注意事項	6.3.3
タグの属性値に指定する Expression に関する注意事項	6.3.4
taglib ディレクティブの prefix 属性に関する注意事項	6.3.5

6.3.1　カスタムタグのスクリプト変数の定義に関する注意事項

ここでは，JSP のカスタムタグで，スクリプト変数を定義する場合の注意事項，JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異，および JSP トランスレーション下位互換機能の定義について説明します。

(1)　注意事項

● スコープ指定時の注意

複数のカスタムタグで，スクリプト変数名とスクリプト変数のスコープが重複した場合，アプリケーションサーバのバージョンによって，JSP のコンパイル結果が異なります。スクリプト変数のスコープは，次のどちらかで指定します。

- javax.servlet.jsp.tagext.TagExtraInfo クラスのサブクラス
- TLD ファイルの variable 要素内の scope 要素

JSP のカスタムタグで同じ名称のスクリプト変数を指定した場合に，アプリケーションサーバのバージョンごとの動作の違いを次に示します。

- 07-00 より前のバージョンの場合

2 回目以降のカスタムタグに対応する JSP から生成された Java コードでも，スクリプト変数の変数宣言をします。

- 07-00 以降のバージョンの場合

  2 回目以降のカスタムタグに対応する JSP から生成された Java コードでは，スクリプト変数の変数宣言をしません。

スコープが「AT_BEGIN」で，属性 id に指定された変数名（var）のスクリプト変数を定義するカスタムタグ<my:foo>を例に説明します。

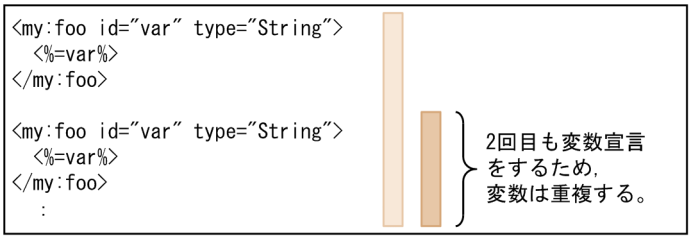
定義例

```
<my:foo id="var" type="String">
  <%=var%>
</my:foo>

<my:foo id="var" type="String">
  <%=var%>
</my:foo>
```

- 07-00 より前のバージョンの場合

  2 回目のカスタムタグに対応する JSP から生成された Java コードでも，スクリプト変数 var で変数宣言をします。この場合，各カスタムタグでのスコープ「AT_BEGIN」の範囲は次のようになります。



（凡例）

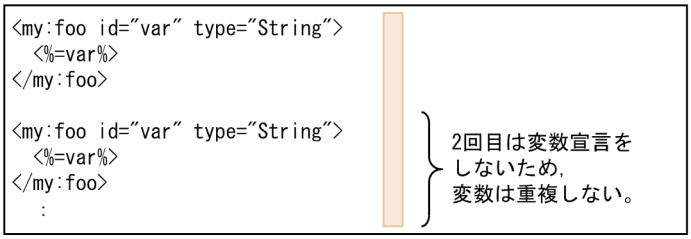
：1回目のカスタムタグで定義するスクリプト変数の有効範囲

：2回目のカスタムタグで定義するスクリプト変数の有効範囲

  このため，JSP から生成された Java ソースのコンパイル時にコンパイルエラーとなります。

- 07-00 以降のバージョンの場合

  2 回目のカスタムタグに対応する JSP から生成された Java コードでは，スクリプト変数 var で変数宣言をしません。この場合，各カスタムタグでのスコープ「AT_BEGIN」の範囲は次のようになります。



（凡例）

：1回目のカスタムタグで定義するスクリプト変数の有効範囲

  このため，同一スコープ内で同一の名称のスクリプト変数を定義しても，JSP から生成された Java ソースのコンパイルは正常に実行されます。

● スクリプトレット記述時の注意

JSP から Java ソースを生成する際，スクリプトレットで記述された Java コードは解析しません。このため，カスタムタグの前後，またはボディのスクリプトレットで，スクリプト変数のスコープが変更される処理を定義していると，アプリケーションサーバのバージョンによっては，JSP コンパイルの結果がエラーとなることがあります。

- スコープが「AT_BEGIN」の場合にコンパイルエラーとなる例

  この例では，カスタムタグ<my:foo>で，スコープに「AT_BEGIN」，属性 id に指定された変数名（var）のスクリプト変数を定義しています。

```
<% if(flag) { %>

  <my:foo id="var" type="String">
    <%=var%>
  </my:foo>

<% } else { %>

  <my:foo id="var" type="String">
    <%=var%>
  </my:foo>

<% } %>
```

  この定義例での，アプリケーションサーバのバージョンごとの動作を次に示します。

  - 07-00 より前のバージョンの場合

    2 回目のカスタムタグに対応する JSP から生成された Java コードでも，スクリプト変数 var で変数宣言をします。このため，2 回目のスクリプト変数 var の参照でもエラーとなりません。JSP コンパイルは正常に実行されます。

  - 07-00 以降のバージョンの場合

    2 回目のカスタムタグでは，スクリプト変数 var が宣言済みと解析します。この場合，カスタムタグに対応する JSP から生成された Java コードでは，変数 var で変数宣言をしません。このため，2 回目のスクリプト変数 var の参照でエラーとなります。

(2)　JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異

07-00 以降のバージョンで，JSP トランスレーション下位互換機能を使用するときと使用しないときの，コンパイルの動作の差異を次に示します。

複数のカスタムタグで，スクリプト変数名とスクリプト変数のスコープが重複した場合

- JSP トランスレーション下位互換機能を使用する

  2 回目以降のカスタムタグに対応する JSP から生成された Java コードでも，スクリプト変数の変数宣言をする。

- JSP トランスレーション下位互換機能を使用しない

  2 回目以降のカスタムタグに対応する JSP から生成された Java コードでは，スクリプト変数の変数宣言をしない。

(3)　JSP トランスレーション下位互換機能の定義

JSP トランスレーション下位互換機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.jsp.translation.backcompat.customAction.declareVariable

JSP ファイルから生成された Java コードで 2 回目のカスタムタグに対応するスクリプト変数の変数宣言を出力するかどうかを指定します。

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

6.3.2　<jsp:useBean>タグの class 属性に関する注意事項

ここでは，<jsp:useBean>タグの class 属性に関する注意事項，JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異，および JSP トランスレーション下位互換機能の定義について説明します。

(1)　注意事項

<jsp:useBean>タグでは，class 属性はオブジェクトの実装クラス名であることが JSP の仕様で定義されています。07-00 以降では，この仕様に準拠して，JSP トランスレーション時に，class 属性に指定されたクラスについて次のチェックを実施しています。

- クラスの修飾子が public である。
- クラスの修飾子が interface ではない。
- クラスの修飾子が abstract ではない。
- メソッドの修飾子が public で引数がないコンストラクタが存在する。

このため，class 属性にこれらのチェック項目に該当しないクラスを指定したとき，07-00 より前と 07-00 以降では JSP コンパイルの結果が異なります。

次のように include 元で実装クラスを指定して, include 先でインタフェースを指定した JSP を例にして説明します。

- test1.jsp（インクルード元）

```
<jsp:useBean id="bean" scope="request" class="test.TestBean"/>
<jsp:include page="test1_included.jsp"/>
```

- test1_included.jsp（インクルード先）

```
<jsp:useBean id="bean" scope="request" class="test.TestBeanIF"/>
```

注

class 属性に指定した test.TestBean は JSP 仕様に準拠した実装クラス，test.TestBeanIF は test.TestBean のインタフェースになります。

この定義例での，アプリケーションサーバのバージョンごとの動作を次に示します。

- 07-00 より前のバージョンの場合

  JSP トランスレーション時にチェックを実施しないため，JSP ファイルから生成されたサーブレットが実行されます。

  JSP ファイルから生成されたサーブレットでは，id 属性に指定された"bean"で既存のスクリプト変数を検索します。上記ではインクルード元（test1.jsp）ですでに同一スクリプト変数が定義済みのため，class 属性で指定されたクラス（test.TestBeanIF インタフェース）からのインスタンス化はしないで，既存のオブジェクト（test.TestBean クラスのインスタンス）を使用します。そのため，正常に実行されます。

- 07-00 以降のバージョンの場合

  test1_included.jsp は，JSP トランスレーション時に class 属性に指定されたクラスのチェックを実施するため，JSP トランスレーションでエラーとなります。

(2)　JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異

07-00 以降のバージョンで，JSP トランスレーション下位互換機能を使用するときと使用しないときの，コンパイルの動作の差異を次に示します。

インスタンス化できないクラス名を class 属性に指定した場合

- **JSP トランスレーション下位互換機能を使用する**

  2 回目以降の<jsp:useBean>タグで指定した id 属性値がエラーにならないで，Bean が取得できる。

- **JSP トランスレーション下位互換機能を使用しない**

  JSP のトランスレーション時にエラーになる。

(3)　JSP トランスレーション下位互換機能の定義

JSP トランスレーション下位互換機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.jsp.translation.backcompat.useBean.noCheckClass

JSP トランスレーション時に<jsp:useBean>タグのクラス属性値のチェック処理を実行するかどうかを指定します。

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

6.3.3　タグの属性値の Expression チェックに関する注意事項

ここでは，タグの属性値の Expression チェックに関する注意事項，JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異，および JSP トランスレーション下位互換機能の定義について説明します。

(1)　注意事項

JSP 仕様では，Expression を指定できるタグの属性が制限されています。07-00 以降のバージョンでは，JSP トランスレーション下位互換機能を使用しないで Expression が指定できる属性以外で Expression を指定した場合，JSP トランスレーション時にエラーになります。しかし，07-00 より前のバージョンでは，JSP トランスレーション時に Expression が指定できるかどうかをチェックしないため，Expression を表す「<%=」や「%>」は文字列として認識されてエラーになりません。そのため，Expression が指定できる属性以外で Expression を指定した場合, 07-00 より前のバージョンと 07-00 以降のバージョンでは JSP コンパイルの結果が異なります。

Expression が指定できる属性以外で Expression を指定している JSP を使用する場合は，必ず JSP トランスレーション下位互換機能を設定してください。

(2)　JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異

07-00 以降のバージョンで，JSP トランスレーション下位互換機能を使用するときと使用しないときの，コンパイルの動作の差異を次に示します。

Expression の指定が許されていないタグの属性値に，Expression を指定した場合

- JSP トランスレーション下位互換機能を使用する

  Expression の指定が許されていないタグの属性値に指定した Expression は，文字列として扱われる。
- JSP トランスレーション下位互換機能を使用しない

  JSP のトランスレーション時にエラーになる。

(3)　JSP トランスレーション下位互換機能の定義

JSP トランスレーション下位互換機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.jsp.translation.backcompat.tag.noCheckRtexprvalue

Expression が指定できないタグの属性値に Expression が指定されているかどうか検証するかどうかを指定します。

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

6.3.4　タグの属性値に指定する Expression に関する注意事項

ここでは，タグの属性値に指定する Expression に関する注意事項，JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異，および JSP トランスレーション下位互換機能の定義について説明します。

(1)　注意事項

タグの属性値に Expression を指定する場合，「"<%= scriptlet_expr %>"」または，「'<%= scriptlet_expr %>'」と指定します。

タグの属性値が，「"<%=」（または「'<%=」）で開始していて「%>"」（または「%>'」）で終了していない場合，07-00 より前のバージョンでは，「"」（または「'」）で囲まれた値を文字列として扱います。例えば，「%>」と「"」の間に，任意の文字列がある場合は，「"」で囲まれた値を文字列として扱います。しかし，07-00 以降のバージョンでは，「%>"」（または「%>'」）を属性値の末尾として扱うため，JSP トランスレーション時にエラーになります。

タグの属性値が，「"<%=」（または「'<%=」）で開始していて「%>"」（または「%>'」）で終了していない場合は，必ず JSP トランスレーション下位互換機能を設定してください。

(2)　JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異

07-00 以降のバージョンで，JSP トランスレーション下位互換機能を使用するときと使用しないときの，コンパイルの動作の差異を次に示します。

タグの属性値が「"<%=」（または「'<%=」）で開始していて，「%>"」（または「%>'」）で終了していない場合

- JSP トランスレーション下位互換機能を使用する
  「"」（または，「'」）で囲まれた属性値を，文字列として処理する。
- JSP トランスレーション下位互換機能を使用しない
  「"」（または，「'」）で囲まれた属性値を，Expression として処理する。

(3)　JSP トランスレーション下位互換機能の定義

JSP トランスレーション下位互換機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.jsp.translation.backcompat.tag.rtexprvalueTerminate
タグの属性値が，「"<%=」または「'<%=」で開始しており，「%>"」（「'<%」で開始した場合は「%>'」）で終了していない属性値の「"」（または「'」）で囲まれた値を文字列として扱うかどうかを指定します。

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

6.3.5　taglib ディレクティブの prefix 属性に関する注意事項

ここでは，taglib ディレクティブの prefix 属性に関する注意事項，JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異，および JSP トランスレーション下位互換機能の定義について説明します。

(1)　注意事項

JSP 仕様では，taglib ディレクティブの前に taglib ディレクティブで指定した prefix を使用したカスタムタグを指定できません。07-00 以降のバージョンでは，JSP 仕様に従って，taglib ディレクティブの前に taglib ディレクティブで指定した prefix を使用したカスタムタグを記述しているかをチェックします。taglib ディレクティブの前に taglib ディレクティブで指定した prefix を使用したカスタムタグを記述している場合，トランスレーション時にエラーになります。しかし，07-00 より前のバージョンでは，このチェックをしないため，記述されたカスタムタグは，文字列として扱われます。

そのため，taglib ディレクティブの前に taglib ディレクティブで指定した prefix を使用してカスタムタグを記述している場合は, 07-00 より前と 07-00 以降のバージョンでは JSP コンパイルの結果が異なります。

taglib ディレクティブの前に taglib ディレクティブで指定した prefix を使用したカスタムタグを記述している場合は，必ず JSP トランスレーション下位互換機能を設定してください。

(2)　JSP トランスレーション下位互換機能の使用有無による JSP のコンパイルの動作の差異

07-00 以降のバージョンで，JSP トランスレーション下位互換機能を使用するときと使用しないときの，コンパイルの動作の差異を次に示します。

taglib ディレクティブの前に，taglib ディレクティブで指定した prefix を使用したカスタムタグを記述している場合

- JSP トランスレーション下位互換機能を使用する
  カスタムタグではなく文字列として扱う。

- JSP トランスレーション下位互換機能を使用しない
  JSP のトランスレーション時にエラーになる。

(3)　JSP トランスレーション下位互換機能の定義

JSP トランスレーション下位互換機能の定義は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に次のパラメタを指定します。

webserver.jsp.translation.backcompat.taglib.noCheckPrefix
: taglib ディレクティブの前に，taglib ディレクティブで指定した prefix を使用したカスタムタグを記述しているかチェックするかどうかを指定します。

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

付録

付録 A　エラーステータスコード

ここでは，Web コンテナ，リダイレクタ，およびインプロセス HTTP サーバが返すエラーステータスコードについて示します。

使用する Web サーバによって，エラーが発生する個所が異なります。エラーが発生する個所に応じたエラーステータスコードを参照してください。使用する Web サーバとエラーが発生する個所の対応について次の表に示します。

表 A-1　使用する Web サーバとエラーが発生する個所の対応

使用する Web サーバ	エラーが発生する個所		
	Web コンテナ	リダイレクタ	インプロセス HTTP サーバ
HTTP Server または Microsoft IIS	○	○	－
インプロセス HTTP サーバ	○	－	○

（凡例）○：エラーが発生する。　－：エラーは発生しない。

付録 A.1　Web コンテナが返すエラーステータスコード

クライアントから，存在しないリソースや例外が発生したサーブレットなどにアクセスがあると，Web コンテナはエラーステータスコードを返します。Web コンテナが返すエラーステータスコードと，エラーステータスコードを返す条件を次の表に示します。

表 A-2　Web コンテナが返すエラーステータスコードと条件

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	次のどれかに該当する場合，エラーステータスコード 400 が返ります。 • FORM 認証で使用するログインページとして指定されたリソースに対して，クライアントから直接リクエストを送信し，その結果表示されたログインページからユーザ認証に成功した場合 • 次の三つの条件をすべて満たしているアクセスの場合 1. HTTP のバージョンが"HTTP/1.0"のとき 2. アクセス対象となるサーブレットが javax.servlet.http.HttpServlet を継承しているとき 3. アクセス時の HTTP メソッドが該当するサーブレットでオーバーライドされていないとき • Content-Length ヘッダの値が 2147483647 より大きい，または 0 より小さいリクエストヘッダでアクセスされた場合 • Content-Length ヘッダの値が数値以外のリクエストヘッダでアクセスされた場合 • Content-Length ヘッダを複数含むリクエストヘッダでアクセスされた場合 • リクエスト URI を正規化できなかった場合
401　Unauthorized	BASIC 認証を必要とするリソースに対して，次のようなアクセスがあった場合，エラーステータスコード 401 が返ります。 • 不正なユーザ名，またはパスワードでアクセスされた場合

エラーステータスコード	エラーステータスコードを返す条件
401　Unauthorized	• 認証情報を含まないでアクセスされた場合（Authorization ヘッダがないアクセス）。
403　Forbidden	次のどれかに該当する場合，エラーステータスコード 403 が返ります。 • BASIC 認証，または FORM 認証を必要とするリソースに対して，認可できないユーザ名でアクセスされた場合 • web.xml で，auth-constraint 要素に role-name 要素を指定しないで，すべてのアクセスを許可しないとするリソースにアクセスされた場合※1 • 静的コンテンツに対して，PUT または DELETE メソッドでアクセスされた場合 • web.xml の<transport-guarantee>要素で，INTEGRAL または CONFIDENTIAL が設定されているリソースに http でアクセスされた場合※2
404　Not Found	次のどちらかのアクセスがあった場合，エラーステータスコード 404 が返ります。 • 存在しないリソースにアクセスされた場合 • javax.servlet.UnavailableException が発生しているサーブレット，または JSP ファイルにアクセスされた場合※3
405　Method Not Allowed	次の三つの条件をすべて満たしているアクセスの場合，エラーステータスコード 405 が返ります。 • HTTP のバージョンが"HTTP/1.1"の場合 • アクセス対象となるサーブレットが javax.servlet.http.HttpServlet を継承している場合 • アクセス時の HTTP メソッドが該当するサーブレットでオーバーライドされていない場合
412　Precondition Failed	If-Match ヘッダ，または If-Unmodified-Since ヘッダで指定した条件に一致しない静的コンテンツへのアクセスの場合，エラーステータスコード 412 が返ります。
413　Request Entity Too Large	リクエストボディのサイズが上限値を超えた場合，エラーステータスコード 413 が返ります。
416　Requested Range Not Satisfiable	次のどれかに当てはまる不正な Range ヘッダの値を使用した静的コンテンツへのアクセスの場合，エラーステータスコード 416 が返ります。 • Range ヘッダの値が"byte"から始まっていない • 範囲定義に数字や"-"を使用していない • 指定範囲が妥当ではない
500　Internal Server Error	次のどれかに該当する場合，エラーステータスコード 500 が返ります。 • 例外が発生するサーブレットまたは JSP ファイルにアクセスされた場合※4 • コンパイルに失敗した JSP ファイルにアクセスされた場合 • 削除された静的コンテンツにアクセスされた場合※5 • 静的コンテンツへのアクセスで I/O エラーが発生した場合 • web.xml の定義が不正な状態で，<auth-constraint>要素で保護されたリソースにアクセスされた場合※6
501　Not Implemented	静的コンテンツまたは javax.servlet.http.HttpServlet を継承したサーブレットに対して，GET，HEAD，POST，PUT，DELETE，OPTIONS，TRACE メソッド以外の HTTP メソッドでアクセスされた場合，エラーステータスコード 501 が返ります。

エラーステータスコード	エラーステータスコードを返す条件
503　Service Unavailable	次のどれかに該当する場合，エラーステータスコード 503 が返ります。 • リクエストの実行待ちキューに空きがない場合※7 • javax.servlet.UnavailableException が発生しているサーブレットまたは JSP ファイルにアクセスされた場合※8 • 終了処理中の Web コンテナに対してアクセスされた場合 • 予期しないエラーまたは例外によって，異常な状態になった Web アプリケーションにアクセスされた場合 • Web コンテナサーバで，Web コンテナサーバだけが起動して，Web アプリケーションが起動していない場合に，起動しなかった Web アプリケーションにアクセスされたとき

注※1　Web アプリケーションのバージョンが 2.4 以降の場合に該当します。

注※2　usrconf.properties の webserver.connector.redirect_https.port キーに転送先の https のポート番号を設定していない場合が該当します。

注※3　Web アプリケーションのバージョンが 2.4 以降の場合で，永久的に利用できないことを示す javax.servlet.UnavailableException が発生し，サーブレットおよび JSP ファイルで例外を catch していないときが該当します。

注※4　次のような場合が該当します。

- Web アプリケーションのバージョンが 2.4 以降の場合
  サーブレットまたは JSP で例外を catch していないとき
- Web アプリケーションのバージョンが 2.3 の場合
  web.xml の<error-page>タグまたは JSP ファイルの page ディレクティブでエラーページの指定がなく，サーブレットまたは JSP ファイルで例外を catch していないとき

注※5　Web アプリケーションのリロード機能，JSP ファイルの再コンパイル機能，または J2EE アプリケーションのリロード機能を使用しない場合が該当します。

注※6　web.xml で<auth-constraint>要素に<role-name>要素が定義され，<login-config>要素が定義されていない場合に該当します。この状態でアプリケーションを開始すると，KDJE39150-W の警告メッセージがコンソール画面，およびメッセージログに出力されます。

注※7　Web アプリケーション単位の同時実行スレッド数制御，または URL グループ単位の同時実行スレッド数制御を設定している場合が該当します。

注※8　次のような場合が該当します。

- Web アプリケーションのバージョンが 2.4 以降の場合
  一時的に利用できないことを示す，javax.servlet.UnavailableException が発生し，サーブレットまたは JSP で例外を catch していないとき
- Web アプリケーションのバージョンが 2.3 の場合
  web.xml の<error-page>タグまたは JSP ファイルの page ディレクティブでエラーページの指定がなく，サーブレットまたは JSP ファイルで例外を catch していないとき

付録 A.2　リダイレクタが返すエラーステータスコード

Web コンテナとのデータ送受信時にタイムアウトが発生したり，定義ファイルに記載誤りがあったりすると，リダイレクタはエラーステータスコードを返します。リダイレクタが返すエラーステータスコードと，エラーステータスコードを返す条件を，Web サーバの種類ごとに表に示します。

表 A-3　リダイレクタが返すエラーステータスコードと発生条件（HTTP Server の場合）

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	次のどれかに該当する場合，エラーステータスコード 400 が返ります。 • リクエストの Host ヘッダのポート番号が不正な場合 • リクエストのメソッドが POST ではない場合※1 • リクエストに Content-Length ヘッダがない（POST リクエストである場合，ボディがチャンク形式である）場合※1 • リクエストの Content-Length ヘッダの値が，POST リクエスト転送先ワーカに設定された POST データサイズの上限を超えた値である場合※1
500　Internal Server Error	次のどれかに該当する場合，エラーステータスコード 500 が返ります。 • mod_jk.conf の内容に記述誤りがある場合※2 • workers.properties の読み込みに失敗した場合，または内容に記述誤りがある場合※1 • リクエストヘッダが 16KB を超えている場合※3 • Web コンテナとのコネクション確立に失敗した場合 • Web コンテナとのコネクション確立でタイムアウトが発生した場合 • Web コンテナへのデータ送信時にエラーが発生した場合 • Web コンテナへのデータ送信時にタイムアウトが発生した場合 • Web コンテナからのデータ受信時にエラーが発生した場合 • Web コンテナからのデータ受信時にタイムアウトが発生した場合 • クライアントからの POST データの読み込みでタイムアウトが発生した場合 • リクエストにサポートしない HTTP メソッド※4 が指定されている場合

注※1　POST データサイズによる振り分けで，デフォルトワーカを指定していない場合が該当します。

注※2　Windows の場合だけ該当します。UNIX の場合は起動に失敗します。

注※3　HTTP Server のリクエストヘッダの制限に関する設定で合計 16KB 以上のリクエストヘッダを受信可能にしている場合，該当するおそれがあります。

注※4　HTTP メソッドのサポート可否については，表 A-5 を参照してください。

表 A-4　リダイレクタが返すエラーステータスコードと発生条件（Microsoft IIS の場合）

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	次のどちらかの場合，エラーステータスコード 400 が返ります。 • リクエスト URL に"%"を含み，さらに"%"のあとに続く 2 文字が 16 進数を表さない文字（A～F，a～f，または 0～9 以外の文字）の場合 • リクエストの Host ヘッダのポート番号が不正の場合
403　Forbidden	次のどちらかの場合，エラーステータスコード 403 が返ります。 • リクエスト URL が"hitachi_ccfj"から始まる場合※1 • リクエスト URL に"%2F"が含まれる場合※1
500　Internal Server Error	次のどれかに該当する場合，エラーステータスコード 500 が返ります。 • isapi_redirect.conf の内容に記述誤りがある場合 • workers.properties の読み込みに失敗した場合，または内容に記述誤りがある場合

エラーステータスコード	エラーステータスコードを返す条件
500　Internal Server Error	• リクエストヘッダが 16KB を超えている場合※2 • Web コンテナとのコネクション確立に失敗した場合 • Web コンテナとのコネクション確立でタイムアウトが発生した場合 • Web コンテナへのデータ送信時にエラーが発生した場合 • Web コンテナへのデータ送信時にタイムアウトが発生した場合 • Web コンテナからのデータ受信時にエラーが発生した場合 • Web コンテナからのデータ受信時にタイムアウトが発生した場合 • クライアントからの POST データの読み込みでタイムアウトが発生した場合 • リクエストにサポートしない HTTP メソッド※3 が指定されている場合

注※1　大文字・小文字の区別はありません。

注※2　Microsoft IIS のリクエストヘッダの制限に関する設定で合計 16KB 以上のリクエストヘッダを受信可能にしている場合，該当するおそれがあります。

注※3　HTTP メソッドのサポート可否については，表 A-5 を参照してください。

リダイレクタでのリクエストの HTTP メソッドのサポート可否

リダイレクタでのリクエストの HTTP メソッドのサポート可否について，次の表に示します。

表 A-5　リダイレクタでのリクエストの HTTP メソッドのサポート可否

HTTP メソッド	サポート可否
OPTIONS	○
GET	○
HEAD	○
POST	○
PUT	○
DELETE	○
TRACE	○
CONNECT	×※
PROPFIND	○
PROPPATCH	○
MKCOL	○
COPY	○
MOVE	○
LOCK	○
UNLOCK	○
ACL	○
REPORT	○

HTTP メソッド	サポート可否
VERSION-CONTROL	○
CHECKIN	○
CHECKOUT	○
UNCHECKOUT	○
SEARCH	○
上記以外の HTTP1.1 で使用できるメソッド	×※

（凡例）○：サポートする　×：サポートしない

注※

サポートしない HTTP メソッドが指定されているリクエストには，リダイレクタがステータス 500 エラーを返します。また，メッセージ KDJE41001-E が出力されます。

付録 A.3　インプロセス HTTP サーバが返すエラーステータスコード

クライアントからのリクエストのサイズが上限値を超えたり，値が不正だったりした場合，インプロセス HTTP サーバはエラーステータスコードを返します。インプロセス HTTP サーバが返すエラーステータスコードとエラーステータスコードを返す条件を次の表に示します。

表 A-6　インプロセス HTTP サーバが返すエラーステータスコードと条件

エラーステータスコード	エラーステータスコードを返す条件
400　Bad Request	リクエストの状態が次のどれかの場合，エラーステータスコード 400 が返ります。 • リクエストの HTTP のバージョンが 1.1 で，Host ヘッダがない場合 • リクエストの Host ヘッダのポート番号が不正な場合 • リクエストヘッダのサイズが上限値を超えた場合 • リクエストヘッダの数が上限値を超えた場合 • リクエスト URI が不正な場合 • リクエスト URI のデコードに失敗した場合 • リクエスト URI を正規化できなかった場合 • リクエストの Content-Length ヘッダの値が 2147483647 より大きい，または 0 より小さい場合 • リクエストの Content-Length ヘッダの値が数値以外の場合 • リクエストの Content-Length ヘッダが複数指定されている場合 • リクエストラインの HTTP のバージョンがサポートされていない場合
403　Forbidden	web.xml の<transport-guarantee>要素で，INTEGRAL または CONFIDENTIAL が設定されているリソースに http でアクセスされた場合，エラーステータスコード 403 が返ります。
405　Method Not Allowed	許可しない HTTP メソッドからアクセスがあった場合，エラーステータスコード 405 が返ります。
413　Request entity too large	リクエストボディのサイズが上限値を超えた場合，エラーステータスコード 413 が返ります。

エラーステータスコード	エラーステータスコードを返す条件
414　Request-URI too large	リクエストラインの長さが上限値を超えた場合，エラーステータスコード 414 が返ります。
500　Internal Server Error	リダイレクト機能によってステータスコード 200 でファイルを返す際にファイルの読み込みに失敗した場合，エラーステータスコード 500 が返ります。
501　Not implemented	リクエストの transfer-encoding ヘッダの値が，サポートされていない場合に，エラーステータスコード 501 が返ります。
503　Service Unavailable	流量制御の上限を超えてリクエスト処理を実行しようとした場合，エラーステータスコード 503 が返ります。

付録 B　HTTP Server の設定に関する注意事項

ここでは，HTTP Server の設定に関する注意事項について説明します。

付録 B.1　HTTP Server の再起動時の注意事項

HTTP Server の再起動に失敗した原因が，簡易構築定義ファイル（Smart Composer 機能を使用しない場合はリダイレクタ定義ファイル（mod_jk.conf），またはワーカ定義ファイル（workers.properties））にある場合，次のどちらかにメッセージが出力されます。

- HTTP Server の起動コマンド実行画面またはイベントログ
  - HTTP Server をコマンドプロンプトから起動，停止，または再起動するときは，起動コマンド実行画面に出力されます。
  - HTTP Server をサービスから起動，停止，または再起動するときは，イベントログに出力されます。Windows の場合だけが該当します。
- HTTP Server のエラーログファイル

  エラーログファイルは，デフォルトでは次の場所に出力されます。
  - Windows の場合

    <HTTP Server のインストールディレクトリ>¥logs¥error.log
  - UNIX の場合

    /opt/hitachi/httpsd/logs/error.log

それぞれに出力されるメッセージを表に示します。

表 B-1　HTTP Server の起動コマンド実行画面またはイベントログに出力されるメッセージ

メッセージ	原因・対策方法
JkWorkersFile file_name invalid※	JkWorkersFile キーに指定されたファイル名が無効です。JkWorkersFile キーに指定している値を修正したあと，Web サーバを再起動してください。
Can't find the workers file specified※	指定されたワーカ定義ファイルが見つかりません。JkWorkersFile キーに指定されているファイルの有無を確認したあと，Web サーバを再起動してください。
Content should start with /※	URL パターンの先頭文字が「/」（スラッシュ）以外です。JkMount キーに指定されている URL パターンの先頭文字を「/」（スラッシュ）に修正したあと，Web サーバを再起動してください。
JkOptions: Illegal option 'a...a'	JkOption キーに指定された値（a...a）は不正です。 JkOption キーに指定している値を修正したあと，Web サーバを再起動してください。
a...a takes b...b arguments,	a...a に示すキーに指定できる値の数は b...b です。 a...a に示すキーに指定している値の数を修正したあと，Web サーバを再起動してください。
a...a must be On or Off	a...a に示すキーに指定できる値は On または Off です。 a...a に示すキーに指定している値を On または Off にしたあと，Web サーバを再起動してください。
JkModulePriority: Invalid value specified.a...a	JkModulePriority キーに設定された値 a...a が不正です。JkModulePriority キーに正しい値を設定して，Web サーバを再起動してください。

注※　UNIX の場合だけ出力されるメッセージです。

表 B-2　HTTP Server のエラーログファイル

メッセージ	説明・対策方法
[時刻] [emerg] Memory error	メモリが不足しています。 システムが利用できるメモリ使用量を確保したあと，Web サーバを再起動してください。
[時刻] [emerg] Error while opening the workers※	次のどちらかの問題が発生しています。 • ワーカ定義ファイルのオープンに失敗しました。 JkWorkersFile キーに指定されているファイルのアクセス権に読み取り権限があることを確認したあと，Web サーバを再起動してください。 • ワーカ定義ファイルの内容が不正です。 ワーカ定義ファイルの内容を正しく修正したあと，Web サーバを再起動してください。
[時刻] [emerg] Error while checking the mod_jk.conf※	リダイレクタ定義ファイルに不適切な記述がありました。リダイレクタに出力されているメッセージを確認したあと，Web サーバを再起動してください。

注※　UNIX の場合だけ出力されるメッセージです。

付録 B.2　リダイレクタのログに関する注意事項

- ログ出力先ディレクトリに HTTP Server の実行アカウントの書き込み権限がない場合，リクエストは正常に処理されますが，ログは出力されません。
- Windows の場合，デフォルトの状態ではリダイレクタのログ出力先ディレクトリ（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥logs）がありません。このため，HTTP Server は，起動時に一つ上のディレクトリ（redirector ディレクトリ）のアクセス権を引き継いで，logs ディレクトリを生成してログを出力しようとします。このとき，HTTP Server の実行アカウントの書き込み権限がないとログは出力されません。この場合は，logs ディレクトリを生成してアクセス権を設定するか，または一つ上のディレクトリ（redirector ディレクトリ）にアクセス権を設定してください。

  また，リダイレクタのログ出力先ディレクトリを変更している場合に，指定しているパスが途中までしかないときは，存在する最下層のディレクトリに対してアクセス権を設定するか，または指定したパスに対応するディレクトリをすべて作成して，最下層のディレクトリにアクセス権を設定してください。

付録 B.3　HTTP Server のアップグレード時の注意事項

HTTP Server をアップグレードした場合，HTTP Server 用のリダイレクタの変更内容を反映するためには，HTTP Server を再起動する必要があります。HTTP Server の再起動の方法については，マニュアル「HTTP Server」を参照してください。

付録 C　Microsoft IIS の設定

ここでは，Microsoft IIS を使用して Web サーバと連携する場合の設定方法について説明します。

付録 C.1　Microsoft IIS 7.0，Microsoft IIS 7.5，Microsoft IIS 8.0，または Microsoft IIS 8.5 の設定

ここでは，Microsoft IIS を使用して Web サーバと連携する場合の Microsoft IIS 7.0，Microsoft IIS 7.5，Microsoft IIS 8.0，または Microsoft IIS 8.5 の設定方法について説明します。

J2EE サーバと Microsoft IIS との連携は，次に示す手順で設定します。

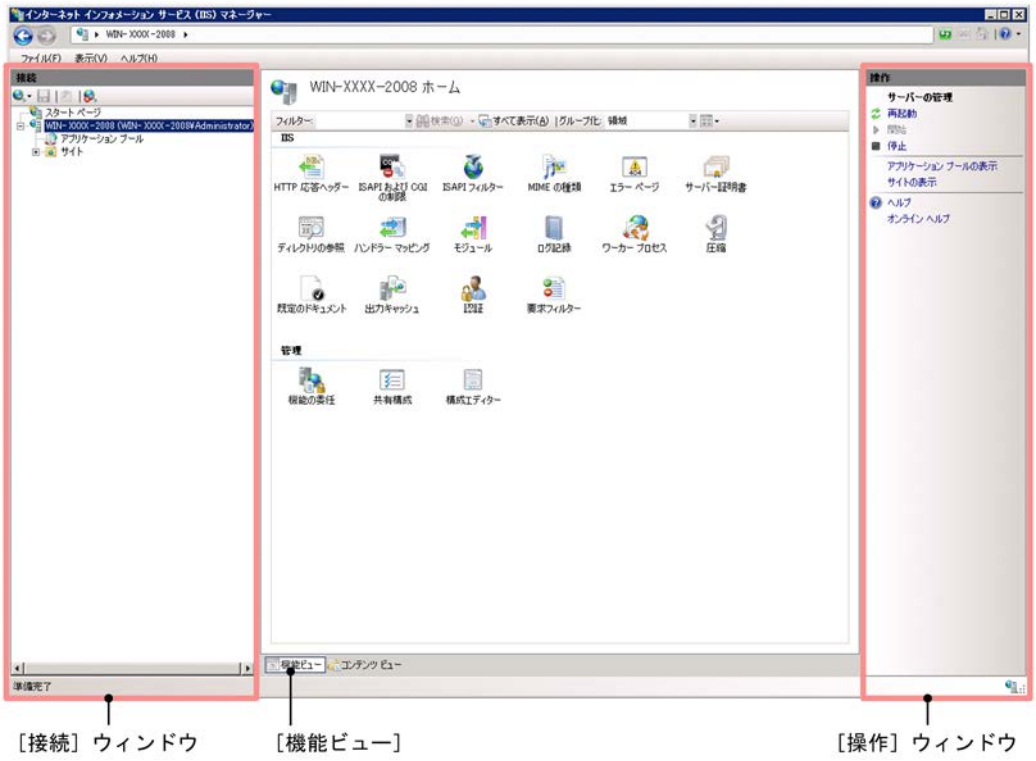
1. Microsoft IIS および役割サービスのインストール
2. ISAPI および CGI の制限の追加
3. ISAPI フィルタの追加
4. ハンドラマッピングの設定
5. 仮想ディレクトリの追加
6. [サーバー] ノードでのアプリケーションプールの設定
7. [サイト] ノードでのアプリケーションプールの設定
8. リダイレクタログ出力先ディレクトリへのアクセス権の設定
9. Web サイトの開始

参考

- Microsoft IIS 7.0 と Microsoft IIS 7.5 以降では，画面表記が一部異なります。以降の手順では，Microsoft IIS 7.5 以降の場合の表記で説明します。Microsoft IIS 7.0 を使用している場合は，次の表に示すとおりに読み替えてください。

Microsoft IIS 7.5 以降の画面表記	Microsoft IIS 7.0 の画面表記
サーバーマネージャー	サーバーマネージャ
インターネット インフォメーション サービス（IIS）マネージャー	インターネット インフォメーション サービス（IIS）マネージャ
ISAPI フィルター	ISAPI フィルタ
フィルター名	フィルタ名
ハンドラー マッピング	ハンドラ マッピング
ワーカー プロセス	ワーカ プロセス

- Microsoft IIS の設定で使用する，[インターネット インフォメーション サービス（IIS）マネージャー] の画面構成を次に示します。なお，以降で示す手順に従って，[接続] ウィンドウ，[機能ビュー]，[操作] ウィンドウなどで設定を実施してください。



(1)　Microsoft IIS および役割サービスのインストール

Microsoft IIS を使用する場合, Microsoft IIS および用途に応じた役割サービスをインストールする必要があります。IIS および役割サービスをインストールする手順を次に示します。

Microsoft IIS がインストール済みの場合，不足している役割サービスがあれば追加インストールしてください（必要な役割サービスについては後述）。

＜ Microsoft IIS 7.0 または Microsoft IIS 7.5 を使用する場合＞

1. [スタート］メニューの［すべてのプログラム］－［管理ツール］から，［サーバーマネージャー］を起動します。
2. [サーバーマネージャー］の［役割］を選択し，［役割の追加］をクリックします。
3. ウィザードが起動するので［次へ］をクリックします。
4. [サーバーの役割］では，［Web サーバー（IIS)］を選択し，［次へ］をクリックします。
   ［Web サーバー（IIS）に必要な機能を追加しますか？］のポップアップが出た場合，［必要な機能を追加］をクリックします。
5. [Web サーバー（IIS)］では［次へ］をクリックします。
6. [役割サービス］では，最低限次の役割サービスを選択し，［次へ］をクリックします。
   - 静的なコンテンツ
   - 既定のドキュメント
   - ディレクトリの参照
   - HTTP エラー
   - ISAPI 拡張

- ISAPI フィルター
- IIS 管理コンソール

指定した役割サービス以外も必要に応じて追加してください。特に［HTTP ログ］や［トレース］は障害発生時のトラブルシュートに有益です。

7. [インストール] をクリックします。
8. [閉じる] をクリックします。
   インストールが完了します。

＜ Microsoft IIS 8.0 または Microsoft IIS 8.5 を使用する場合＞

1. [サーバーマネージャー] を起動します。
   （自動起動していない場合，スタート画面から起動できます）
2. [サーバーマネージャー] の［役割と機能の追加］をクリックします。
3. ウィザードが起動するので［次へ］をクリックします。
4. [役割ベースまたは機能ベースのインストール] を選択し，［次へ］をクリックします。
5. 対象のサーバを選択し，［次へ］をクリックします。
6. [サーバーの役割] では，［Web サーバー（IIS)］を選択し，［次へ］をクリックします。
   ［Web サーバー（IIS）に必要な機能を追加しますか？］のポップアップが出た場合，［管理ツールを含める（存在する場合)］にチェックを入れて［機能の追加］をクリックします。
7. [機能の選択] では［次へ］をクリックします。
8. [Web サーバーの役割（IIS)］では［次へ］をクリックし，［役割サービスの選択］では最低限次の役割サービスを選択し，［次へ］をクリックします。
   - HTTP エラー
   - ディレクトリの参照
   - 既定のドキュメント
   - 静的なコンテンツ
   - ISAPI 拡張
   - ISAPI フィルター
   - IIS 管理コンソール

   指定した役割サービス以外も必要に応じて追加してください。特に［HTTP ログ］や［トレース］は障害発生時のトラブルシュートに有益です。
9. [インストール] をクリックします。
10. [閉じる] をクリックします。
    インストールが完了します。

(2)　ISAPI および CGI の制限の追加

ISAPI および CGI の制限を追加する手順を次に示します。

1. [インターネット インフォメーション サービス（IIS）マネージャー] を起動します。
   Microsoft IIS 7.0 または Microsoft IIS 7.5 の場合は［スタート］メニューの［すべてのプログラム］－［管理ツール］から，Microsoft IIS 8.0 または Microsoft IIS 8.5 の場合は［サーバマネージャー］の［ツール］から起動できます。

2. [機能ビュー] のサーバの［ホーム］ページで，[ISAPI および CGI 制限] をダブルクリックします。
3. [ISAPI および CGI 制限] ページの［操作］ウィンドウで［追加］をクリックします。
4. [ISAPI または CGI の制限の追加] ダイアログで次の操作をします。
   - ［ISAPI または CGI パス］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）を指定します。
   - ［説明］に「ISAPI」と入力します。
   - ［拡張パスの実行を許可する］をチェックします。
5. [OK] ボタンをクリックします。
   ［ISAPI または CGI の制限の追加］ダイアログが閉じて，設定が反映されます。

(3)　ISAPI フィルタの追加

ISAPI フィルタを追加する手順を次に示します。

1. [機能ビュー] のサイトの［ホーム］ページで［ISAPI フィルター］をダブルクリックします。
2. [ISAPI フィルター] ページの［操作］ウィンドウで［追加］をクリックします。
3. [ISAPI フィルターの追加] ダイアログで次の操作をします。
   - ［フィルター名］に「hitachi_ccfj」と入力します。
   - ［実行可能ファイル］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）を指定します。
4. [OK] ボタンをクリックします。
   ［ISAPI フィルターの追加］ダイアログが閉じて，設定が反映されます。

(4)　ハンドラマッピングの設定

ハンドラマッピングを設定する手順を次に示します。

1. [機能ビュー] のサイトの［ホーム］ページで［ハンドラー マッピング］をダブルクリックします。
2. [ハンドラー マッピング] ページで「ISAPI-dll」を選択し，［操作］ウィンドウで［編集...］をクリックします。
3. [モジュール マップの編集] ダイアログで次の操作をします。
   - ［要求パス］に「*.dll」と入力します。
   - ［実行可能ファイル］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）を指定します。

   すでにハンドラマッピングが設定されている場合，［スクリプトマップの編集］ダイアログが起動しますが，操作内容は同じです。
4. [OK] ボタンをクリックします。
5. モジュールマップの編集を有効にするかどうかを確認するメッセージダイアログで，ISAPI 拡張機能を有効にするために，［はい］ボタンをクリックします。
6. [ハンドラー マッピング] ページで「ISAPI-dll」を選択し，［操作］ウィンドウで［機能のアクセス許可の編集...］をクリックします。
7. [機能のアクセス許可の編集] ダイアログで「読み取り」，「スクリプト」および「実行」のすべてのアクセス許可をチェックします。
8. [OK] ボタンをクリックします。

［機能のアクセス許可の編集］ダイアログが閉じて，設定が反映されます。

(5)　仮想ディレクトリの追加

「hitachi_ccfj」という名称の仮想ディレクトリを追加します。仮想ディレクトリを追加する手順を次に示します。

1. **［接続］ウィンドウで［サイト］ノードを展開し，仮想ディレクトリを追加するサイトをクリックします。**
2. **［操作］ウィンドウで，［仮想ディレクトリの表示］をクリックします。**
   ［仮想ディレクトリ］ページが表示されます。
3. **［仮想ディレクトリ］ページの［操作］ウィンドウで，［仮想ディレクトリの追加...］をクリックします。**
4. **［仮想ディレクトリの追加］ダイアログで次の操作をします。**
   - ［エイリアス］に「hitachi_ccfj」と入力します。
   - ［物理パス］にリダイレクタの DLL（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.dll）が格納されているディレクトリを指定します。
5. **［OK］ボタンをクリックします。**
   ［仮想ディレクトリの追加］ダイアログが閉じて，設定が反映されます。

(6)　［サーバー］ノードでのアプリケーションプールの設定

［サーバー］ノードでのアプリケーションプールの設定手順を次に示します。

1. **［接続］ウィンドウでサーバノードを展開し，［アプリケーション プール］をクリックします。**
2. **［アプリケーション プール］ページで，使用するアプリケーションプールを選択し，［操作］ウィンドウの［詳細設定］をクリックします。**
3. **［詳細設定］ダイアログで次の操作をします。**
   - ［32 ビットアプリケーションの有効化］に「True」を指定します（Windows x86 のリダイレクタを Windows x64 の OS で動かす場合だけに必要な指定です）。
   - ［ワーカー プロセスの最大数］※に，Microsoft IIS 内でリクエストを処理するプロセスの最大数を指定します。

   注※

   このマニュアルでは，Web コンテナの実行プロセスのことも「ワーカプロセス」と表記していますが，ここで設定する「ワーカー プロセス」とは Microsoft IIS 内でリクエストを処理するプロセスを指します。
4. **［OK］ボタンをクリックします。**
   ［詳細設定］ダイアログが閉じて，設定が反映されます。

(7)　［サイト］ノードでのアプリケーションプールの設定

サイトノードでのアプリケーションプールの設定手順を次に示します。

1. **［接続］ウィンドウで［サイト］ノードを展開し，アプリケーションプールを指定するサイトをクリックします。**
2. **［操作］ウィンドウで，［詳細設定］をクリックします。**
3. **［詳細設定］ダイアログで，［アプリケーションプール］を入力します。**

［アプリケーションプール］に，「(6) ［サーバー］ノードでのアプリケーションプールの設定」で設定したアプリケーションプールの名称を指定します。

4. **［OK］ボタンをクリックします。**
［詳細設定］ダイアログが閉じて，設定が反映されます。

(8) リダイレクタログ出力先ディレクトリへのアクセス権の設定

リダイレクタのログ出力先ディレクトリには，Microsoft IIS のアプリケーションプールの実行アカウントに対する書き込み権限を付加しておく必要があります。エクスプローラから，リダイレクタのログ出力ディレクトリにアクセス権を設定してください。

アプリケーションプールの実行アカウントは，アプリケーションプールの［詳細設定］ダイアログの［ID］で指定します。デフォルトの ID を指定している場合は，「IIS_IUSRS」グループに対して書き込み権限を付加してください。

デフォルトの ID は次のとおりです。

- Microsoft IIS 7.0：「NetworkService」
- Microsoft IIS 7.5：「ApplicationPoolIdentity」
- Microsoft IIS 8.0：「ApplicationPoolIdentity」
- Microsoft IIS 8.5：「ApplicationPoolIdentity」

なお，新規インストール時には，デフォルトの状態ではリダイレクタのログ出力先ディレクトリ（<Application Server のインストールディレクトリ>¥CC¥web¥redirector¥logs）がありません。このため，logs ディレクトリを作成してアクセス権を設定するか，または一つ上のディレクトリ（redirector ディレクトリ）にアクセス権を設定してください。

また，リダイレクタのログ出力先ディレクトリを変更している場合に，指定しているパスが途中までしかないときは，存在する最下層のディレクトリに対してアクセス権を設定するか，または指定したパスに対応するディレクトリをすべて作成して，最下層のディレクトリにアクセス権を設定してください。

(9) Web サイトの開始

Microsoft IIS の Web サイトの開始の手順を次に示します。

1. **［接続］ウィンドウで［サイト］ノードを展開し，開始するサイトをクリックします。**
2. **［操作］ウィンドウで，［開始］をクリックします。開始済みの場合は，［再起動］をクリックします。**
Web サイトが開始，または再起動します。

(10) 注意事項

ここでは，Microsoft IIS の設定時の注意事項について説明します。

(a) 構成設定の複数環境へのレプリケーションについての注意事項

Microsoft IIS では，構成設定を web.config ファイルに保存できます。また，保存した web.config ファイルを基に，xcopy を使用して複数環境に構成環境をレプリケーションできます。

ただし，リダイレクタを使用する構成環境の場合，xcopy での複数環境へのレプリケーションでリダイレクタの設定はできません。xcopy で複数環境を同一の構成設定にする場合も，リダイレクタの設定は環境ごとに手動で同期してください。

(b)　エラーページのカスタマイズについての注意事項

Microsoft IIS でエラーページにカスタムエラーページを返す設定にしている場合，web.xml の<error-page>タグによるエラーページのカスタマイズは無効となることがあります。web.xml の<error-page>タグによるエラーページのカスタマイズを有効にしたい場合は，Microsoft IIS のエラーページのカスタムエラーページの設定を無効にしてください。

(c)　ほかのフィルタと共存させる場合の注意事項

Microsoft IIS が受信したリクエストが，Web コンテナに転送するリクエストの場合，Microsoft IIS 用リダイレクタは ISAPI フィルタ内で使用するリクエスト URL 情報を変更します。そのため，Microsoft IIS 用リダイレクタよりあとに実行される ISAPI フィルタでは，Microsoft IIS が受信したリクエスト URL を取得できません。Microsoft IIS が受信したリクエスト URL をフィルタで取得する場合は，Microsoft IIS 用リダイレクタよりも順番が先になるように設定してください。順番を調整するために，Microsoft IIS 用リダイレクタの優先順位を「中」または「低」に変更する必要がある場合は，isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）の filter_priority キーで指定します。isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）の filter_priority キーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「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。

付録 D　各バージョンでの主な機能変更

ここでは，09-70 よりも前のアプリケーションサーバの各バージョンでの主な機能の変更について，変更目的ごとに説明します。09-70 での主な機能変更については，「1.4　アプリケーションサーバ 09-70 での主な機能変更」を参照してください。

説明内容は次のとおりです。

- アプリケーションサーバの各バージョンで変更になった主な機能と，その概要を説明しています。機能の詳細については，「参照先マニュアル」の「参照個所」の記述を確認してください。「参照先マニュアル」および「参照個所」には，その機能についての 09-70 のマニュアルでの主な記載個所を記載しています。
- 「参照先マニュアル」に示したマニュアル名の「アプリケーションサーバ」は省略しています。

付録 D.1　09-60 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 D-1　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
G1GC への対応	G1GC を選択できるようになりました。	システム設計ガイド	7.15
		リファレンス 定義編(サーバ定義)	16.5
圧縮オブジェクトポインタ機能への対応	圧縮オブジェクトポインタ機能を使用できるようになりました。	機能解説 保守／移行編	9.18

(2)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D-2　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ファイナライズ滞留解消機能の追加	ファイナライズ処理の滞留を解消でき，OS 資源の解放遅れなどの発生を抑止できるようになりました。	機能解説 保守／移行編	9.16

(3)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D-3　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
ログファイルの非同期出力機能の追加	ログのファイル出力を非同期でできるようになりました。	リファレンス定義編(サーバ定義)	16.2

付録 D.2　09-50 での主な機能変更

(1)　開発生産性の向上

開発生産性の向上を目的として変更した項目を次の表に示します。

表 D–4　開発生産性の向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Eclipse セットアップの簡略化	GUI を利用して Eclipse 環境をセットアップできるようになりました。	アプリケーション開発ガイド	1.1.5, 2.4
ユーザ拡張性能解析トレースを使ったデバッグ支援	ユーザ拡張性能解析トレース設定ファイルを開発環境で作成できるようになりました。	アプリケーション開発ガイド	1.1.3, 6.5

(2)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 D–5　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境でのシステム構成パターンの拡充	仮想化環境で使用できるティアの種類（http-tier, j2ee-tier および ctm-tier）が増えました。これによって，次のシステム構成パターンが構築できるようになりました。 • Web サーバと J2EE サーバを別のホストに配置するパターン • フロントエンド（サーブレット，JSP）とバックエンド（EJB）を分けて配置するパターン • CTM を使用するパターン	仮想化システム構築・運用ガイド	1.1.2

(3)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 D–6　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JDBC 4.0 仕様への対応	DB Connector で JDBC 4.0 仕様の HiRDB Type4 JDBC Driver，および SQL Server の JDBC ドライバに対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	3.6.3
Portable Global JNDI 名での命名規則の緩和	Portable Global JNDI 名に使用できる文字を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	2.4.3
Servlet 3.0 仕様への対応	Servlet 3.0 の HTTP Cookie の名称，および URL のパスパラメタ名の変更が，Servlet 2.5 以前のバージョンでも使用できるようになりました。	このマニュアル	2.7
Bean Validation と連携できるアプリケーションの適用拡大	CDI やユーザアプリケーションでも Bean Validation を使って検証できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	10 章

項目	変更の概要	参照先マニュアル	参照個所
JavaMail への対応	JavaMail 1.4 に準拠した API を使用したメール送受信機能を利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	8 章
javacore コマンドが使用できる OS の適用拡大	javacore コマンドを使って，Windows のスレッドダンプを取得できるようになりました。	リファレンス コマンド編	javacore (スレッドダンプの取得／Windows の場合)

(4)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D–7　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
コードキャッシュ領域の枯渇回避	システムで使用しているコードキャッシュ領域のサイズを確認して，領域が枯渇する前にしきい値を変更して領域枯渇するのを回避できるようになりました。	システム設計ガイド	7.2.6
		機能解説 保守／移行編	5.7.2, 5.7.3
		リファレンス 定義編 (サーバ定義)	16.1, 16.2, 16.4
明示管理ヒープ機能の効率的な適用への対応	自動解放処理時間を短縮し，明示管理ヒープ機能を効率的に適用するための機能として，Explicit ヒープに移動するオブジェクトを制御できる機能を追加しました。 • Explicit メモリブロックへのオブジェクト移動制御機能 • 明示管理ヒープ機能適用除外クラス指定機能 • Explicit ヒープ情報へのオブジェクト解放率情報の出力	システム設計ガイド	7.14.6
		機能解説 拡張編	8.2.2, 8.6.5, 8.10, 8.13.1, 8.13.3
		機能解説 保守／移行編	5.5
クラス別統計情報の出力範囲拡大	クラス別統計情報を含んだ拡張スレッドダンプに，static フィールドを基点とした参照関係を出力できるようになりました。	機能解説 保守／移行編	9.6

(5)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 D–8　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
EADs セッションフェイルオーバ機能のサポート	EADs と連携してセッションフェイルオーバ機能を実現する EADs セッションフェイルオーバ機能をサポートしました。	機能解説 拡張編	5 章, 7 章

項目	変更の概要	参照先マニュアル	参照個所
WAR による運用	WAR ファイルだけで構成された WAR アプリケーションを J2EE サーバにデプロイできるようになりました。	このマニュアル	2.2.1
		機能解説 基本・開発編(コンテナ共通機能)	13.9
		リファレンス コマンド編	cjimportwar(WAR アプリケーションのインポート)
運用管理機能の同期実行による起動と停止	運用管理機能（Management Server および運用管理エージェント）の起動および停止を，同期実行するオプションを追加しました。	機能解説 運用／監視／連携編	2.6.1, 2.6.2, 2.6.3, 2.6.4
		リファレンス コマンド編	adminagentctl(運用管理エージェントの起動と停止), mngautorun（自動起動および自動再起動の設定／設定解除), mngsvrctl(Management Server の起動／停止／セットアップ)
明示管理ヒープ機能での Explicit メモリブロックの強制解放	javagc コマンドで，Explicit メモリブロックの解放処理を任意のタイミングで実行できるようになりました。	機能解説 拡張編	8.6.1, 8.9
		リファレンス コマンド編	javagc(GC の強制発生)

(6)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D-9　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
定義情報の取得	snapshotlog（snapshot ログの収集）コマンドで定義ファイルだけを収集できるようになりました。	機能解説 保守／移行編	2.3
		リファレンス コマンド編	snapshotlog（snapshot ログの収集）
cjenvsetup コマンドのログ出力	Component Container 管理者のセットアップ（cjenvsetup コマンド）の実行情報がメッセージログに出力されるようになりました。	システム構築・運用ガイド	4.1.4
		機能解説 保守／移行編	4.20
		リファレンス コマンド編	cjenvsetup（Component Container 管理者のセットアップ）
BIG-IP v11 のサポート	使用できる負荷分散機の種類に BIG-IP v11 が追加になりました。	システム構築・運用ガイド	4.7.2
		仮想化システム構築・運用ガイド	2.1
明示管理ヒープ機能のイベントログへの CPU 時間の出力	Explicit メモリブロック解放処理に掛かった CPU 時間が，明示管理ヒープ機能のイベントログに出力されるようになりました。	機能解説 保守／移行編	5.11.3
ユーザ拡張性能解析トレースの機能拡張	ユーザ拡張性能解析トレースで，次の機能を追加しました。 • トレース対象の指定方法を通常のメソッド単位の指定方法に加えて，パッケージ単位またはクラス単位で指定できるようになりました。 • 使用できるイベント ID の範囲を拡張しました。 • ユーザ拡張性能解析トレース設定ファイルに指定できる行数の制限を緩和しました。 • ユーザ拡張性能解析トレース設定ファイルでトレース取得レベルを指定できるようになりました。	機能解説 保守／移行編	7.5.2, 7.5.3, 8.28.1
Session Bean の非同期呼び出し使用時の情報解析向上	PRF トレースのルートアプリケーション情報を使用して，呼び出し元と呼び出し先のリクエストを突き合わせることができるようになりました。	機能解説 基本・開発編 (EJB コンテナ)	2.17.3

付録 D.3　09-00 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 D–10　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での構築・運用の操作対象単位の変更	仮想化環境の構築・運用時の操作対象単位が仮想サーバから仮想サーバグループへ変更になりました。仮想サーバグループの情報を定義したファイルを使用して，複数の仮想サーバを管理ユニットへ一括で登録できるようになりました。	仮想化システム構築・運用ガイド	1.1.2
セットアップウィザードによる構築環境の制限解除	セットアップウィザードを使用して構築できる環境の制限が解除されました。ほかの機能で構築した環境があってもアンセットアップされて，セットアップウィザードで構築できるようになりました。	システム構築・運用ガイド	2.2.7
構築環境の削除手順の簡略化	Management Server を使用して構築したシステム環境を削除する機能（mngunsetup コマンド）の追加によって，削除手順を簡略化しました。	システム構築・運用ガイド	4.1.37
		運用管理ポータル操作ガイド	3.6，5.4
		リファレンス コマンド編	mngunsetup（Management Server の構築環境の削除）

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 D–11　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 3.0 への対応	Servlet 3.0 に対応しました。	このマニュアル	6 章
EJB 3.1 への対応	EJB 3.1 に対応しました。	機能解説 基本・開発編 (EJB コンテナ)	2 章
JSF 2.1 への対応	JSF 2.1 に対応しました。	このマニュアル	3 章
JSTL 1.2 への対応	JSTL 1.2 に対応しました。	このマニュアル	3 章
CDI 1.0 への対応	CDI 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	9 章
Portable Global JNDI 名の利用	Portable Global JNDI 名を利用したオブジェクトのルックアップができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	2.4

項目	変更の概要	参照先マニュアル	参照個所
JAX-WS 2.2 への対応	JAX-WS 2.2 に対応しました。	Web サービス開発ガイド	1.1, 16.1.5, 16.1.7, 16.2.1, 16.2.6, 16.2.10, 16.2.12, 16.2.13, 16.2.14, 16.2.16, 16.2.17, 16.2.18, 16.2.20, 16.2.22, 19.1, 19.2.3, 37.2, 37.6.1, 37.6.2, 37.6.3
JAX-RS 1.1 への対応	JAX-RS 1.1 に対応しました。	Web サービス開発ガイド	1.1, 1.2.2, 1.3.2, 1.4.2, 1.5.1, 1.6, 2.3, 11 章, 12 章，13 章，17 章，24 章，39 章

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D-12　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
SSL/TLS 通信での TLSv1.2 の使用	RSA BSAFE SSL-J を使用して，TLSv1.2 を含むセキュリティ・プロトコルで SSL/TLS 通信ができるようになりました。	－	－

（凡例）－：09-70 で削除された機能です。

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 D-13　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web コンテナ全体の実行待ちキューの総和の監視	Web コンテナ全体の実行待ちキューの総和を稼働情報に出力して監視できるようになりました。	機能解説 運用／監視／連携編	3 章
アプリケーションの性能解析トレース（ユーザ拡張トレース）の出力	ユーザが開発したアプリケーションの処理性能を解析するための性能解析トレースを，アプリケーションの変更をしないで出力できるようになりました。	機能解説 保守／移行編	7 章
仮想化環境でのユーザスクリプトを使用した運用	任意のタイミングでユーザ作成のスクリプト（ユーザスクリプト）を仮想サーバ上で実行できるようになりました。	仮想化システム構築・運用ガイド	7.8
運用管理ポータル改善	運用管理ポータルの次の画面で，手順を示すメッセージを画面に表示するように変更しました。 • ［設定情報の配布］画面 • Web サーバ，J2EE サーバおよび SFO サーバの起動画面 • Web サーバクラスタと J2EE サーバクラスタの一括起動，一括再起動および起動画面	運用管理ポータル操作ガイド	10.11.1, 11.9.2, 11.10.2, 11.11.2, 11.11.4, 11.11.6, 11.12.2, 11.13.2, 11.13.4, 11.13.6
運用管理機能の再起動機能の追加	運用管理機能（Management Server および運用管理エージェント）で自動再起動が設定できるようになり，運用管理機能で障害が発生した場合でも運用が継続できるようになりました。また，自動起動の設定方法も変更になりました。	機能解説 運用／監視／連携編	2.4.1, 2.4.2, 2.6.3, 2.6.4
		リファレンス コマンド編	mngautorun（自動起動および自動再起動の設定／設定解除）

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D-14　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
ログ出力時のファイル切り替え単位の変更	ログ出力時に，日付ごとに出力先のファイルを切り替えられるようになりました。	機能解説 保守／移行編	3.2.1
Web サーバの名称の変更	アプリケーションサーバに含まれる Web サーバの名称を HTTP Server に変更しました。	HTTP Server	－
BIG-IP の API（SOAP アーキテクチャ）を使用した直接接続への対応	BIG-IP（負荷分散機）で API（SOAP アーキテクチャ）を使用した直接接続に対応しました。 また，API を使用した直接接続を使用する場合に，負荷分散機の接続環境を設定する方法が変更になりました。	システム構築・運用ガイド	4.7.3，付録 K
		仮想化システム構築・運用ガイド	2.1，付録 C

項目	変更の概要	参照先マニュアル	参照個所
BIG-IP の API（SOAP アーキテクチャ）を使用した直接接続への対応	BIG-IP（負荷分散機）で API（SOAP アーキテクチャ）を使用した直接接続に対応しました。 また，API を使用した直接接続を使用する場合に，負荷分散機の接続環境を設定する方法が変更になりました。	機能解説 セキュリティ管理機能編	8.2，8.4，8.5，8.6，18.2，18.3，18.4

（凡例）－：マニュアル全体を参照する

付録 D.4　08-70 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 D-15　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータル改善	運用管理ポータルの画面で，リソースアダプタの属性を定義するプロパティ（Connector 属性ファイルの設定内容）の設定，および接続テストができるようになりました。また，運用管理ポータルの画面で，J2EE アプリケーション（ear ファイルおよび zip ファイル）を Management Server にアップロードできるようになりました。	ファーストステップガイド	3.5
		運用管理ポータル操作ガイド	－
page/tag ディレクティブの import 属性暗黙インポート機能の追加	page/tag ディレクティブの import 属性暗黙インポート機能を使用できるようになりました。	このマニュアル	2.3.7
仮想化環境での JP1 製品に対する環境設定の自動化対応	仮想サーバへのアプリケーションサーバ構築時に，仮想サーバに対する JP1 製品の環境設定を，フックスクリプトで自動的に設定できるようになりました。	仮想化システム構築・運用ガイド	7.7.2
統合ユーザ管理機能の改善	ユーザ情報リポジトリでデータベースを使用する場合に，データベース製品の JDBC ドライバを使用して，データベースに接続できるようになりました。 DABroker Library の JDBC ドライバによるデータベース接続はサポート外になりました。 簡易構築定義ファイルおよび運用管理ポータルの画面で，統合ユーザ管理機能に関する設定ができるようになりました。 また，Active Directory の場合，DN で日本語などの 2 バイト文字に対応しました。	機能解説 セキュリティ管理機能編	5 章，14.3
		運用管理ポータル操作ガイド	3.5，10.9.1
HTTP Server 設定項目の拡充	簡易構築定義ファイルおよび運用管理ポータルの画面で，HTTP Server の動作環境を定義するディレクティブ（httpsd.conf の設定内容）を直接設定できるようになりました。	システム構築・運用ガイド	4.1.21
		運用管理ポータル操作ガイド	10.10.1
		リファレンス 定義編(サーバ定義)	4.13

（凡例）－：マニュアル全体を参照する

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 D–16　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ejb-jar.xml の指定項目の追加	ejb-jar.xml に，クラスレベルインターセプタおよびメソッドレベルインターセプタを指定できるようになりました。	機能解説 基本・開発編 (EJB コンテナ)	2.15
パラレルコピーガーベージコレクションへの対応	パラレルコピーガーベージコレクションを選択できるようになりました。	リファレンス 定義編 (サーバ定義)	16.5
Connector 1.5 仕様に準拠した Inbound リソースアダプタのグローバルトランザクションへの対応	Connector 1.5 仕様に準拠したリソースアダプタで Transacted Delivery を使用できるようにしました。これによって，Message-driven Bean を呼び出す EIS がグローバルトランザクションに参加できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3.16.3
TP1 インバウンドアダプタの MHP への対応	TP1 インバウンドアダプタを使用してアプリケーションサーバを呼び出す OpenTP1 のクライアントとして，MHP を使用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	4 章
cjrarupdate コマンドの FTP インバウンドアダプタへの対応	cjrarupdate コマンドでバージョンアップできるリソースアダプタに FTP インバウンドアダプタを追加しました。	リファレンス コマンド編	2.2

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D–17　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
データベースセッションフェイルオーバ機能の改善	性能を重視するシステムで，グローバルセッション情報を格納したデータベースのロックを取得しないモードを選択できるようになりました。また，データベースを更新しない，参照専用のリクエストを定義できるようになりました。	機能解説 拡張編	6 章
OutOfMemory ハンドリング機能の対象となる処理の拡大	OutOfMemory ハンドリング機能の対象となる処理を追加しました。	機能解説 保守／移行編	2.5.7
		リファレンス 定義編 (サーバ定義)	16.2
HTTP セッションで利用する Explicit ヒープの省メモリ化機能の追加	HTTP セッションで利用する Explicit ヒープのメモリ使用量を抑止する機能を追加しました。	機能解説 拡張編	8.11

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 D-18　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での JP1 製品を使用したユーザ認証への対応（クラウド運用対応）	JP1 連携時に，JP1 製品の認証サーバを利用して，仮想サーバマネージャを使用するユーザを管理・認証できるようになりました。	仮想化システム構築・運用ガイド	1.2.2，3 章，4 章，5 章，6 章，7.9

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D-19　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
負荷分散機への API（REST アーキテクチャ）を使用した直接接続の対応	負荷分散機への接続方法として，API（REST アーキテクチャ）を使用した直接接続に対応しました。 また，使用できる負荷分散機の種類に ACOS（AX2500）が追加になりました。	システム構築・運用ガイド	4.7.2，4.7.3
		仮想化システム構築・運用ガイド	2.1
		リファレンス 定義編(サーバ定義)	4.5
snapshot ログ収集時のタイムアウトへの対応と収集対象の改善	snapshot ログの収集が指定した時間で終了（タイムアウト）できるようになりました。一次送付資料として収集される内容が変更になりました。	機能解説 保守／移行編	付録 A

付録 D.5　08-53 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 D-20　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
さまざまなハイパーバイザに対応した仮想化環境の構築	さまざまなハイパーバイザを使用して実現する仮想サーバ上に，アプリケーションサーバを構築できるようになりました。 また，複数のハイパーバイザが混在する環境にも対応しました。	仮想化システム構築・運用ガイド	2 章，3 章，5 章

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 D-21　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
トランザクション連携に対応した OpenTP1 からの呼び出し	OpenTP1 からアプリケーションサーバ上で動作する Message-driven Bean を呼び出すときに，トランザクション連携ができるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	4 章

項目	変更の概要	参照先マニュアル	参照個所
JavaMail	POP3 に準拠したメールサーバと連携して，JavaMail 1.3 に準拠した API を使用したメール受信機能を利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	8 章

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D–22　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JavaVM のトラブルシュート機能強化	JavaVM のトラブルシュート機能として，次の機能が使用できるようになりました。 • OutOfMemoryError 発生時の動作を変更できるようになりました。 • JIT コンパイル時に，C ヒープ確保量の上限値を設定できるようになりました。 • スレッド数の上限値を設定できるようになりました。 • 拡張 verbosegc 情報の出力項目を拡張しました。	機能解説 保守／移行編	4 章，5 章，9 章

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 D–23　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JP1/ITRM への対応	IT リソースを一元管理する製品である JP1/ITRM に対応しました。	仮想化システム構築・運用ガイド	1.3，2.1

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D–24　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
Microsoft IIS 7.0 および Microsoft IIS 7.5 への対応	Web サーバとして Microsoft IIS 7.0 および Microsoft IIS 7.5 に対応しました。	－	－
HiRDB Version 9 および SQL Server 2008 への対応	データベースとして次の製品に対応しました。 • HiRDB Server Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 また，SQL Server 2008 に対応する JDBC ドライバとして，SQL Server JDBC Driver に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	3 章

（凡例）－：該当なし。

付録 D.6　08-50 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 D-25　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web サービスプロバイダ側での web.xml の指定必須タグの変更	Web サービスプロバイダ側での web.xml で，listener タグ，servlet タグおよび servlet-mapping タグの指定を必須から任意に変更しました。	リファレンス 定義編(サーバ定義)	2.4
論理サーバのネットワークリソース使用	J2EE アプリケーションからほかのホスト上にあるネットワークリソースやネットワークドライブにアクセスするための機能を追加しました。	機能解説 運用／監視／連携編	1.2.3，5.2，5.7
サンプルプログラムの実行手順の簡略化	一部のサンプルプログラムを EAR 形式で提供することによって，サンプルプログラムの実行手順を簡略化しました。	ファーストステップガイド	3.5
		システム構築・運用ガイド	付録 M
運用管理ポータルの画面の動作の改善	画面の更新間隔を「更新しない」から「3 秒」に変更しました。	運用管理ポータル操作ガイド	7.4.1
セットアップウィザードの完了画面の改善	セットアップウィザード完了時の画面に，セットアップで使用した簡易構築定義ファイルおよび Connector 属性ファイルが表示されるようになりました。	システム構築・運用ガイド	2.2.6
仮想化環境の構築	ハイパーバイザを使用して実現する仮想サーバ上に，アプリケーションサーバを構築する手順を追加しました。※	仮想化システム構築・運用ガイド	3 章，5 章

注※

08-50 モードで構築する場合は，マニュアル「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」の「付録 D　08-50 モードの仮想サーバマネージャを利用する場合の設定」を参照してください。

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 D-26　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
OpenTP1 からの呼び出しへの対応	OpenTP1 からアプリケーションサーバ上で動作する Message-driven Bean を呼び出せるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	4 章
JMS への対応	JMS 1.1 仕様に対応した CJMS プロバイダ機能を使用できるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	7 章

項目	変更の概要	参照先マニュアル	参照個所
Java SE 6 への対応	Java SE 6 の機能が使用できるようになりました。	機能解説 保守／移行編	5.5, 5.8.1
ジェネリクスの使用への対応	EJB でジェネリクスを使用できるようになりました。	機能解説 基本・開発編 (EJB コンテナ)	4.2.19

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D–27　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
明示管理ヒープ機能の使用性向上	自動配置設定ファイルを使用して，明示管理ヒープ機能を容易に使用できるようになりました。	システム設計ガイド	7.2, 7.7.3, 7.11.5, 7.12.1
		機能解説 拡張編	8 章
データベースセッションフェイルオーバ機能の URI 単位での抑止	データベースセッションフェイルオーバ機能を使用する場合に，機能の対象外にするリクエストを URI 単位で指定できるようになりました。	機能解説 拡張編	5.6.1
仮想化環境での障害監視	仮想化システムで，仮想サーバを監視し，障害の発生を検知できるようになりました。	仮想化システム構築・運用ガイド	付録 D

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 D–28　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
管理ユーザアカウントの省略	運用管理ポータル，Management Server のコマンド，Smart Composer 機能のコマンドで，ユーザのログイン ID およびパスワードの入力を省略できるようになりました。	システム構築・運用ガイド	4.1.15
		運用管理ポータル操作ガイド	2.2，7.1.1，7.1.2，7.1.3，8.1，8.2.1，付録 F.2
		リファレンス コマンド編	1.4，mngsvrctl（Management Server の起動／停止／セットアップ），mngsvrutil（Management Server の運用管理コマンド），8.3，cmx_admin_passwd（Management Server の管理ユーザアカウントの設定）
仮想化環境の運用	仮想化システムで，複数の仮想サーバを対象にした一括起動・一括停止，スケールイン・スケールアウトなどを実行する手順を追加しました。※	仮想化システム構築・運用ガイド	4 章，6 章

注※

08-50 モードで運用する場合は，マニュアル「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」の「付録 D　08-50 モードの仮想サーバマネージャを利用する場合の設定」を参照してください。

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D-29　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
Tenured 領域内不要オブジェクト統計機能	Tenured 領域内で不要となったオブジェクトだけを特定できるようになりました。	機能解説 保守／移行編	9.8
Tenured 増加要因の基点オブジェクトリスト出力機能	Tenured 領域内不要オブジェクト統計機能を使って特定した，不要オブジェクトの基点となるオブジェクトの情報を出力できるようになりました。		9.9
クラス別統計情報解析機能	クラス別統計情報を CSV 形式で出力できるようになりました。		9.10
論理サーバの自動再起動回数オーバー検知によるクラスタ系切り替え	Management Server を系切り替えの監視対象としているクラスタ構成の場合，論理サーバが異常停止状態(自動再起動回数をオーバーした状態，または自動再起動回数の設定が 0 のときに障害を検知した状態)になったタイミングでの系切り替えができるようになりました。	機能解説 運用／監視／連携編	18.4.3, 18.5.3, 20.2.2, 20.3.3, 20.3.4
ホスト単位管理モデルを対象とした系切り替えシステム	クラスタソフトウェアと連携したシステム運用で，ホスト単位管理モデルを対象にした系切り替えができるようになりました。		20 章
ACOS（AX2000，BS320）のサポート	使用できる負荷分散機の種類に ACOS（AX2000，BS320）が追加になりました。	システム構築・運用ガイド	4.7.2, 4.7.3, 4.7.5, 4.7.6，付録 K，付録 K.2
		リファレンス 定義編(サーバ定義)	4.5, 4.6.2, 4.6.4, 4.6.5, 4.6.6, 4.10.1
CMT でトランザクション管理をする場合に Stateful Session Bean（SessionSynchronization）に指定できるトランザクション属性の追加	CMT でトランザクション管理をする場合に，Stateful Session Bean（SessionSynchronization）にトランザクション属性として Supports, NotSupported および Never を指定できるようになりました。	機能解説 基本・開発編(EJB コンテナ)	2.7.3
OutOfMemoryError 発生時の運用管理エージェントの強制終了	JavaVM で OutOfMemoryError が発生したときに，運用管理エージェントが強制終了するようになりました。	機能解説 保守／移行編	2.5.8

項目	変更の概要	参照先マニュアル	参照個所
スレッドの非同期並行処理	TimerManager および WorkManager を使用して，非同期タイマ処理および非同期スレッド処理を実現できるようになりました。	機能解説 拡張編	10 章

付録 D.7　08-00 での主な機能変更

(1)　開発生産性の向上

開発生産性の向上を目的として変更した項目を次の表に示します。

表 D–30　開発生産性の向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ほかのアプリケーションサーバ製品からの移行容易化	ほかのアプリケーションサーバ製品からの移行を円滑に実施するため，次の機能を使用できるようになりました。 • HTTP セッションの上限が例外で判定できるようになりました。 • JavaBeans の ID が重複している場合や，カスタムタグの属性名と TLD の定義で大文字・小文字が異なる場合に，トランスレーションエラーが発生することを抑止できるようになりました。	このマニュアル	2.3，2.7.5
cosminexus.xml の提供	アプリケーションサーバ独自の属性を cosminexus.xml に記載することによって，J2EE アプリケーションを J2EE サーバにインポート後，プロパティの設定をしないで開始できるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	11.3

(2)　標準機能への対応

標準機能への対応を目的として変更した項目を次の表に示します。

表 D–31　標準機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 2.5 への対応	Servlet 2.5 に対応しました。	このマニュアル	2.2，2.5.4，2.6，6 章
JSP 2.1 への対応	JSP 2.1 に対応しました。	このマニュアル	2.3.1，2.3.3，2.5，2.6，6 章
JSP デバッグ	MyEclipse を使用した開発環境で JSP デバッグができるようになりました。※	このマニュアル	2.4
タグライブラリのライブラリ JAR への格納と TLD のマッピング	タグライブラリをライブラリ JAR に格納した場合に，Web アプリケーション開始時に Web コンテナによってライブラリ JAR 内の TLD ファイルを検索し，自動的にマッピングできるようになりました。	このマニュアル	2.3.4

項目	変更の概要	参照先マニュアル	参照個所
application.xml の省略	J2EE アプリケーションで application.xml が省略できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	11.4
アノテーションと DD の併用	アノテーションと DD を併用できるようになり，アノテーションで指定した内容を DD で更新できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	12.5
アノテーションの Java EE 5 標準準拠（デフォルトインターセプタ）	デフォルトインターセプタをライブラリ JAR に格納できるようになりました。また，デフォルトインターセプタから DI できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	11.4
@Resource の参照解決	@Resource でリソースの参照解決ができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	12.4
JPA への対応	JPA 仕様に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	5 章, 6 章

注※　09-00 以降では，WTP を使用した開発環境で JSP デバッグ機能を使用できます。

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 D–32　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
セッション情報の永続化	HTTP セッションのセッション情報をデータベースに保存して引き継げるようになりました。	機能解説 拡張編	5 章, 6 章
FullGC の抑止	FullGC の要因となるオブジェクトを Java ヒープ外に配置することで, FullGC 発生を抑止できるようになりました。	機能解説 拡張編	8 章
クライアント性能モニタ	クライアント処理に掛かった時間を調査・分析できるようになりました。	－	－

（凡例）－：09-00 で削除された機能です。

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 D–33　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータルでのアプリケーション操作性向上	アプリケーションおよびリソースの操作について，サーバ管理コマンドと運用管理ポータルの相互運用ができるようになりました。	運用管理ポータル操作ガイド	1.1.3

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 D-34　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
無効な HTTP Cookie の削除	無効な HTTP Cookie を削除できるようになりました。	このマニュアル	2.7.4
ネーミングサービスの障害検知	ネーミングサービスの障害が発生した場合に，EJB クライアントが，より早くエラーを検知できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	2.9
コネクション障害検知タイムアウト	コネクション障害検知タイムアウトのタイムアウト時間を指定できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3.15.1
Oracle11g への対応	データベースとして Oracle11g が使用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3 章
バッチ処理のスケジューリング	バッチアプリケーションの実行を CTM によってスケジューリングできるようになりました。	機能解説 拡張編	4 章
バッチ処理のログ	バッチ実行コマンドのログファイルのサイズ，面数，ログの排他処理失敗時のリトライ回数とリトライ間隔を指定できるようになりました。	リファレンス 定義編 (サーバ定義)	3.6
snapshot ログ	snapshot ログの収集内容が変更されました。	機能解説 保守／移行編	付録 A.1，付録 A.2
メソッドキャンセルの保護区公開	メソッドキャンセルの対象外となる保護区リストの内容を公開しました。	機能解説 運用／監視／連携編	付録 C
統計前のガーベージコレクション選択機能	クラス別統計情報を出力する前に，ガーベージコレクションを実行するかどうかを選択できるようになりました。	機能解説 保守／移行編	9.7
Survivor 領域の年齢分布情報出力機能	Survivor 領域の Java オブジェクトの年齢分布情報を JavaVM ログファイルに出力できるようになりました。	機能解説 保守／移行編	9.11
ファイナライズ滞留解消機能	JavaVM のファイナライズ処理の状態を監視して，処理の滞留を解消できるようになりました。	－	－
サーバ管理コマンドの最大ヒープサイズの変更	サーバ管理コマンドが使用する最大ヒープサイズが変更されました。	リファレンス 定義編 (サーバ定義)	5.2，5.3
推奨しない表示名を指定された場合の対応	J2EE アプリケーションで推奨しない表示名を指定された場合にメッセージが出力されるようになりました。	メッセージ(構築／運用／開発用)	KDJE42374-W

（凡例）－：09-00 で削除された機能です。

付録 E　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

記号

数字

B

C

E

H

I

J

M

P

S

T

U

W

X

あ

い

え

か

き

く

け

こ

さ

し

す

せ

そ

た

つ

て

と

に

ね

は

ふ

ほ

ま

も

ゆ

ら

り

れ

ろ

わ