

JEUS Webエンジンガイド

JEUS v8.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(JEUS®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りが無いことを保証しません。

Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

JEUS®は、TmaxSoft Co., Ltd.の登録商標です。

Java and Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Java、Solarisは、Oracle Corporation及びその子会社、関連会社の登録商標です。

Microsoft, Windows, and Windows NT are registered trademarks or trademarks of Microsoft Corporation.

Microsoft、Windows、Windows NTは、Microsoft Corporationの登録商標または商標です。

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

HP-UXは、Hewlett Packard Enterprise Companyの登録商標です。

AIX is a registered trademark of International Business Machines Corporation.

AIXは、International Business Machines Corporationの登録商標です。

UNIX is a registered trademark of X/Open Company, Ltd.

UNIXは、X/Open Company, Ltd.の登録商標です。

Linux is a registered trademark of Linus Torvalds.

Linuxは、Linus Torvaldsの登録商標です。

Other products and company names are trademarks or registered trademarks of their respective owners.

その他、記載されている会社名、製品名などは、各社の商号、商標または登録商標です。

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (TM, ®).

本マニュアルに記載されている会社名、システム名、製品名などには必ずしも商標表示(TM、®)を付記しておりません。

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory : \${INSTALL_PATH}/lib/licenses

この製品の一部ファイルまたはモジュールは、APACHE2.0、CDDL1.0、EDL1.0、OPEN SYMPHONY SOFTWARE1.1、TRILEAD-SSH2、Bouncy Castle、BSD、MIT、SIL OPEN FONT1.1のライセンスに準拠します。

文書情報

文書名: JEUS Webエンジンガイド

発行日: 2016年10月14日

ソフトウェアバージョン: JEUS v8.0

ガイドバージョン: v2.1.1

目次

このガイドについて	xv
第1章 Webエンジン	1
1.1. 概要	1
1.2. 構成要素	1
1.3. 主要機能	3
1.4. 管理ツール	5
1.4.1. ツールの種類	5
1.4.2. 制御およびモニタリング	6
1.5. ディレクトリー構造	10
1.6. 環境設定	12
1.6.1. XML設定ファイル	15
1.6.2. モニタリングの設定	16
1.6.3. 基本エラー・ページの設定	19
1.6.4. エラー発生の場合、スタック・トレース添付の設定	21
1.6.5. エンコーディングの設定	23
1.6.6. JSPエンジンの設定	32
1.6.7. 応答ヘッダーの設定	32
1.6.8. クッキー・ポリシーの設定	34
1.6.9. セッションの設定	36
1.6.10. ログの設定	37
1.6.11. 非同期サーブレットのタイムアウト処理の設定	49
1.6.12. Webエンジン・レベルのプロパティ設定	51
1.6.13. Web攻撃に対応する設定	53
1.6.14. 特定URLパターンのHTTP要求を制限	55
1.7. Webエンジンのチューニング	56
第2章 Webコネクションの管理	59
2.1. 概要	59
2.2. 構成要素	60
2.2.1. リスナー	60
2.2.2. コネクター	61
2.2.3. ワーカー・スレッド・プール	62
2.3. Webコネクションの設定	63
2.3.1. リスナーの共通設定	63
2.3.2. AJPリスナーの設定	69
2.3.3. HTTPリスナーの設定	76
2.3.4. TCPリスナーの設定	82
2.3.5. WebtoBコネクターの設定	85
2.3.6. Tmaxコネクターの設定	93
2.3.7. 自動スレッド・プール管理の設定(スレッド状態の通知)	99
2.4. 負荷分散のためのWebサーバーの設定	101

2.4.1.	負荷分散の構造	102
2.4.2.	Apache Webサーバー	102
2.4.3.	IIS Webサーバーおよびiplanet Webサーバーの設定	107
2.4.4.	WebtoBとの負荷分散設定	107
2.5.	TCPリスナーの使用	109
2.5.1.	カスタマイズされた通信プロトコルの定義	110
2.5.2.	Dispatcher Config Classの実装	111
2.5.3.	TCPサーブレットの実装	113
2.5.4.	カスタマイズされたプロトコルのためのTCPリスナーの設定	116
2.5.5.	TCPクライアントの実装	117
2.5.6.	TCPクライアントのコンパイルおよび実行	119
2.6.	HTTP/2の使用	120
2.6.1.	HTTP/2の設定	120
2.6.2.	サーバー・プッシュ	122
2.7.	リスナーのチューニング	123
第3章	Webコンテキスト	125
3.1.	概要	125
3.2.	基本構造	125
3.2.1.	Webコンテキスト・コンテンツ	125
3.2.2.	Webコンテキストの内部構造(WARファイルの構造)	126
3.3.	Webコンテキストのデプロイ	127
3.3.1.	jeus-web-dd.xmlの設定	127
3.3.2.	Webコンテキスト再デプロイ(グレースフル再デプロイ)	131
3.3.3.	共有ライブラリー参照の追加	131
3.3.4.	互換性のためのWebコンテキスト・レベルのオプション設定	134
3.3.5.	付加機能を使用するための設定	134
3.3.6.	Webセキュリティーの設定	135
3.3.7.	セキュリティー・ロール・マッピング	137
3.3.8.	シンボリック・リファレンスのマッピング	139
3.4.	Webコンテキストのモニタリング	140
3.5.	Webコンテキストの制御	144
3.5.1.	Webコンテキストのリロード	145
3.5.2.	Webコンテキストの非同期要求に対するスレッド・プールのモニタリング	146
3.5.3.	Webコンテキストの一時停止	148
3.5.4.	Webコンテキストのリロード	148
3.6.	Webコンテキストのチューニング	149
3.7.	非同期処理プログラミング	149
3.7.1.	サーブレット標準による注意事項	149
3.7.2.	その他の注意事項	154
第4章	JSPエンジン	155
4.1.	概要	155
4.2.	Apache Tomcat Jasper	156

4.3.	JSPエンジンの機能	157
4.3.1.	JSPのグレースフル・リローディング	158
4.3.2.	JSPのプリコンパイル	159
4.3.3.	メモリーでのJSPコンパイルおよび実行機能	159
4.4.	JSPエンジンの設定	159
4.4.1.	Webエンジン・レベルでの設定	159
4.4.2.	jeus-web-dd.xmlの設定	163
4.4.3.	JSPの下位互換性のためのWebコンテキスト・レベルのオプション設定	164
第5章	仮想ホスト	167
5.1.	概要	167
5.2.	Webエンジンと仮想ホスト	167
5.2.1.	ServletContextオブジェクトと仮想ホスト	169
5.3.	仮想ホストを利用したWebコンテキスト要求	169
5.3.1.	URLマッチングの例	170
5.3.2.	URLマッチングの手順	171
5.4.	仮想ホストの設定	171
5.4.1.	追加	171
5.4.2.	変更	174
5.4.3.	削除	177
第6章	WebSocketコンテナ	179
6.1.	概要	179
6.2.	制限事項	179
6.3.	WebSocketコンテナの機能	180
6.3.1.	WebSocket UserProperties Failover	180
6.3.2.	その他	182
6.4.	WebSocketコンテナの設定	182
第7章	JEUS WebCache	187
7.1.	概要	187
7.2.	JSPキャッシング	188
7.2.1.	基本情報	188
7.2.2.	<cache>タグ	189
7.2.3.	<jeus:cache>の使用例	192
7.2.4.	フラッシュ機能の使用	193
7.2.5.	リフレッシュ機能の使用	193
7.3.	HTTP応答キャッシング	194
7.3.1.	フィルター設定	194
第8章	リバース・プロキシ	197
8.1.	概要	197
8.1.1.	適用例	197
8.2.	使用方法	198
8.2.1.	プロキシ・サーバーの設定	198

8.2.2.	コンテキスト・パスの設定	200
8.2.3.	プロキシ・フィルターの設定	200
8.2.4.	デプロイ	202
8.3.	設定例	202
第9章	クラスの動的反映	205
9.1.	概要	205
9.2.	基本設定および動作	206
9.2.1.	サーバー設定	207
9.2.2.	アプリケーション設定	207
9.2.3.	設定による動作	207
9.2.4.	JEUSホットスワップでサポート可能なアプリケーションおよび変換	208
9.2.5.	JEUSホットスワップの制限事項	211
第10章	実習	213
索引	217

目次

[図 1.1]	Webエンジンの構成要素	1
[図 1.2]	Webエンジンのモニタリング - メニュー移動とサーバーの選択	7
[図 1.3]	Webエンジンのモニタリング - 情報照会	8
[図 1.4]	セッション・リスト - マネージャーの選択	9
[図 1.5]	セッション・リスト - マネージャーの結果	9
[図 1.6]	JEUSのディレクトリー構造でのWebエンジン	10
[図 1.7]	設定の適用 - Activate Changes	13
[図 1.8]	詳細設定	14
[図 1.9]	モニタリング - 設定メニューへ移動	17
[図 1.10]	モニタリング - 設定	18
[図 1.11]	モニタリング - 設定の適用結果	19
[図 1.12]	基本エラー・ページ - 設定	20
[図 1.13]	基本エラー・ページ - 設定の適用結果	21
[図 1.14]	エラーの発生時にスタック・トレースの添付 - 設定	22
[図 1.15]	エラーの発生時にスタック・トレースの添付 - 設定の適用結果	23
[図 1.16]	Webエンジンのエンコーディング - 設定	27
[図 1.17]	Webエンジンのエンコーディング - 設定の適用結果	28
[図 1.18]	Webエンジンの応答ヘッダー - 設定	33
[図 1.19]	Webエンジンの応答ヘッダー - 設定の適用	34
[図 1.20]	Webエンジンのクッキー・ポリシー - 設定	35
[図 1.21]	Webエンジンのクッキー・ポリシー - 設定の適用結果	36
[図 1.22]	Webエンジンのアクセス・ログ - 設定項目へ移動	38
[図 1.23]	Webエンジンのアクセス・ログ - 設定	39
[図 1.24]	Webエンジンのアクセス・ログ - 設定の適用結果	40
[図 1.25]	ユーザー・ログ - 設定	45
[図 1.26]	ユーザー・ログ - ハンドラーの追加	47
[図 1.27]	ユーザー・ログ - ハンドラーの設定	47
[図 1.28]	ユーザー・ログ - ハンドラー追加の確認	48
[図 1.29]	ユーザー・ログ - ハンドラー追加の適用結果	49
[図 1.30]	非同期サーブレットのタイムアウト処理 - Async Timeout Min Threads設定	50
[図 1.31]	非同期サーブレットのタイムアウト処理 - Async Timeout Min Threads設定の適用結果	51
[図 1.32]	Webエンジン・プロパティ - 設定	52
[図 1.33]	Webエンジン・プロパティ - 設定の適用結果	53
[図 1.34]	Web攻撃への対応 - メニュー移動	55
[図 1.35]	Web攻撃への対応 - 詳細設定	55
[図 1.36]	URLパターンを利用したHTTP要求制限機能 - 詳細設定	56
[図 2.1]	Webエンジンのリスナー	60
[図 2.2]	Webエンジンのコネクター	61
[図 2.3]	圧縮機能の設定画面 - 詳細設定	68
[図 2.4]	AJPリスナーの追加および変更 - 追加	69

[図 2.5]	AJPリスナーの追加および変更 - 基本設定	70
[図 2.6]	AJPリスナーの追加および変更 - Thread Poolの設定	70
[図 2.7]	AJPリスナーの追加および変更 - Thread State Notifyの設定	71
[図 2.8]	AJPリスナーの追加および変更 - 詳細設定	71
[図 2.9]	AJPリスナーの追加および変更 - 設定の確認	72
[図 2.10]	AJPリスナーの追加および変更 - 設定の適用結果	73
[図 2.11]	AJPリスナーの削除 - メニューの移動	74
[図 2.12]	AJPリスナーの削除 - 削除の確認	74
[図 2.13]	HTTPリスナーの追加および変更 - 基本設定	77
[図 2.14]	HTTPリスナーの追加および変更 - スレッド・プールの設定	77
[図 2.15]	HTTPリスナーの追加および変更 - 詳細設定	79
[図 2.16]	HTTPリスナーの追加および変更 - 追加の確認	80
[図 2.17]	HTTPリスナーの削除 - 削除の確認	81
[図 2.18]	TCPリスナーの追加および変更 - 基本設定	82
[図 2.19]	TCPリスナーの追加および変更 - 詳細設定	83
[図 2.20]	TCPリスナーの追加および変更 - 追加の確認	84
[図 2.21]	TCPリスナーの削除 - 削除の確認	85
[図 2.22]	WebtoBコネクターの追加および変更 - 基本設定	87
[図 2.23]	WebtoBコネクターの追加および変更 - 接続設定	88
[図 2.24]	WebtoBコネクターの追加および変更 - スレッド・プールの設定	88
[図 2.25]	WebtoBコネクターの追加および変更 - 詳細設定	89
[図 2.26]	WebtoBコネクターの追加および変更 - 追加の確認	91
[図 2.27]	WebtoBコネクターの削除 - 削除の確認	92
[図 2.28]	Tmaxコネクターの追加および変更 - 基本設定	94
[図 2.29]	Tmaxコネクターの追加および変更 - 詳細設定	95
[図 2.30]	Tmaxコネクターの追加および変更 - 追加の確認	96
[図 2.31]	Tmaxコネクターの削除 - 削除の確認	97
[図 2.32]	自動スレッド・プール管理の設定 - Thread State Notifyの設定	100
[図 2.33]	小規模のWebサイトにおける負荷分散構造	102
[図 2.34]	負荷分散サーバーの構造 - 2台のWebtoBが2台のWebエンジンにそれぞれ接続された場 合	107
[図 2.35]	WebtoBコネクター設定の状態	108
[図 3.1]	WARファイルの構造	126
[図 3.2]	Webコンテキストのモニタリング - アプリケーションの照会	141
[図 3.3]	Webコンテキストのモニタリング - Webアプリケーションの選択	142
[図 3.4]	Webコンテキストのモニタリング - Webアプリケーションの詳細情報の照会	143
[図 3.5]	Webアプリケーションのリロード	145
[図 3.6]	Webアプリケーションのthread-info	147
[図 4.1]	JSPエンジンの設定 - メニューの移動	161
[図 4.2]	JSPエンジンの設定 - 基本情報の設定	161
[図 4.3]	JSPエンジンの設定 - 設定の適用結果	163
[図 5.1]	仮想ホストの利用パターン	168
[図 5.2]	Webエンジンと仮想ホスト、Webコンテキスト間の有効な関係の例	169

[図 5.3]	仮想ホストの追加 - メニューの移動	172
[図 5.4]	仮想ホストの追加 - 基本情報の設定	173
[図 5.5]	仮想ホストの追加 - 追加の適用結果	174
[図 5.6]	仮想ホストの変更 - 基本情報の変更	176
[図 5.7]	仮想ホストの変更 - Access Logの追加設定	176
[図 5.8]	仮想ホストの変更 - Access Logの追加適用結果	177
[図 5.9]	仮想ホストの削除 - 削除の確認	178
[図 7.1]	エントリー・キャッシングに使用されるデータ構造	188

例目次

[例 1.1]	XMLヘッダー : <<domain.xml>>	16
[例 1.2]	XMLヘッダー : <<jeus-web-dd.xml>>	16
[例 1.3]	XMLヘッダー : <<web.xml >>	16
[例 1.4]	要求エンコーディングの設定例 : <<jeus-web-dd.xml>>	30
[例 1.5]	ページの文字エンコーディングが設定されたJSPの例の一部 : <<sample.jsp>>	31
[例 1.6]	応答の文字エンコーディングが設定されたJSPの例の一部 : <<sample2.jsp>>	31
[例 1.7]	Pageおよび応答の文字エンコーディングが設定されたweb.xml: <<web.xml>>	31
[例 1.8]	jeus.servlet.logger.AccessLoggerFilterインターフェースの詳細内容 : <<AccessLoggerFilter.java>>	43
[例 1.9]	フィルター・クラスの定義例 : <<SimpleAccessLoggerFilter>>	44
[例 2.1]	mod_jk設定ファイルの例 : <<workers.properties>>	103
[例 2.2]	Apacheにmod_jkを設定する例 : <<httpd.conf>>	106
[例 2.3]	WebtoBの設定例 : <<http.m>>	108
[例 2.4]	TCPDispatcherConfigの実装例 : <<SampleConfig.java>>	111
[例 2.5]	TCPサーブレットの実装例 : <<SampleServlet.java>>	113
[例 2.6]	TCPハンドラーのDD : <<web.xml>>	116
[例 2.7]	TCPクライアントの実装例 : <<Client.java>>	117
[例 2.8]	サーバー・プッシュの使用例 : <<ServerPushServlet.java>>	122
[例 3.1]	Webコンテキストの設定ファイル : <<jeus-web-dd.xml>>	127
[例 3.2]	共有ライブラリー参照の追加 : <<libraries.xml>>	131
[例 3.3]	静的リソースを処理するサーブレット・マッピング : <<web.xml>>	135
[例 3.4]	リダイレクト・ロケーション・セキュリティ設定のインターフェース: <<RedirectStrategy>> ...	136
[例 3.5]	リダイレクト・ロケーション・セキュリティ設定の実装例: <<RejectCrLfRedirectStrategy>> .	136
[例 3.6]	リダイレクト・ロケーション・セキュリティ設定 : <<jeus-web-dd.xml>>	137
[例 3.7]	セキュリティ・ロール・マッピング : <<web.xml>>	137
[例 3.8]	セキュリティ・ロール・マッピング : <<jeus-web-dd.xml>>	138
[例 3.9]	シンボリック・リファレンスのマッピング : <<web.xml>>	139
[例 3.10]	シンボリック・リファレンスのマッピング : <<jeus-web-dd.xml>>	140
[例 3.11]	<async-config>設定 : <<jeus-web-dd.xml>>	146
[例 3.12]	正しくない非同期処理プログラミングの作成例(1) : <<WrongAsyncServlet1.java>>	150
[例 3.13]	正しくない非同期処理プログラミングの作成例(2) : <<WrongAsyncServlet2.java>>	151
[例 3.14]	正しくない非同期処理プログラミングの作成例(3) : <<WrongAsyncServlet3.java>>	152
[例 4.1]	Webコンテキストの設定ファイル : <<jeus-web-dd.xml>>	163
[例 4.2]	JSPの下位互換性のためのWebコンテキストの設定 : <<jeus-web-dd.xml>>	164
[例 6.1]	Webコンテキスト設定ファイル : <<jeus-web-dd.xml>>	183
[例 7.1]	<cache>タグの設定 : <<jeuscache.tld>>	189
[例 7.2]	<jeus:cache>の使用例 : <<cache.jsp>>	192
[例 7.3]	jsp:includeを使用した<jeus:cache>の使用例 : <<main.jsp>>	192
[例 7.4]	フィルター設定 : <<web.xml>>	194
[例 8.1]	data.xmlのDTD : <<proxy-config.dtd>>	199

[例 8.2]	プロキシ・フィルターのみ使用する場合 : <<web.xml>>	201
[例 8.3]	プロキシ・フィルターとリライト・フィルターを一緒に使用する場合 : <<web.xml>>	201
[例 9.1]	Jeusホットスワップの設定 : <<startDomainAdminServer>>	207
[例 9.2]	Jeusホットスワップの設定 : <<startManagedServer>>	207
[例 9.3]	自動リロード設定 : <<jeus-web-dd.xml>>	207

このガイドについて

対象読者

本書は、JEUS[®](以下、JEUS)Webエンジンの使用方法、Webエンジンを構成する要素の概念およびモニタリング方法について説明します。Java EE Webアプリケーションをデプロイおよび管理するシステム管理者とWebアプリケーションの開発者を対象としています。

前提知識

本書を理解するためには、以下の内容についての知識が必要です。

- Java EE, サーブレット、JSP、WebSocket標準
- Webアプリケーションのパッケージングおよびデプロイ
- JEUSの基本構造

JEUSの基本的な使用方法と製品を理解するには、以下のガイドについてあらかじめ熟知することをお勧めします。

- JEUS 紹介ガイド
- JEUS インストール & スタートガイド

本書のすべてのサンプルと環境構成は、UNIXスタイルに準拠します。Microsoft Windows[™](以下、Windows)など他の環境で作業を行う場合は、次のような事項を考慮してください。たとえば、Windowsプラットフォームでは、ディレクトリー区切り子をUNIXスタイルのスラッシュ(/)からWindowsスタイルのバックスラッシュ(\)に変えて使用してください。また、環境変数もWindowsスタイル(%%)に変更して使用してください。本書で触れているJEUS_HOMEは、JEUSがインストールされているディレクトリーです。

制限事項

本書の内容は、Java標準に準拠して作成されていますが、本書で触れているJava EEやJava仕様については詳しく取り上げていません。関連内容についてはJava関連ドキュメントを参照してください。

参考

詳しい内容は、<http://www.oracle.com/technetwork/java/index.html>を参照してください。

本書の構成

本書は、計9章で構成されています。

- 「第1章 Webエンジン」

JEUS Webエンジンの主要機能および基本的な設定方法について説明します。

- 「第2章 Webコネクションの管理」

Webエンジンで提供するリスナーまたはコネクタの管理および設定方法について説明します。

- 「第3章 Webコンテキスト」

JEUS Webアプリケーションをパッケージングする方法とJEUS Webエンジンにデプロイし、モニタリングする方法について説明します。

- 「第4章 JSPエンジン」

JSPエンジンの概念、機能、設定方法について説明します。

- 「第5章 仮想ホスト」

仮想ホストの使用目的、規則、設定方法について説明します。

- 「第6章 WebSocketコンテナ」

WebSocketコンテナの概念、機能、設定方法について説明します。

- 「第7章 JEUS WebCache」

Webアプリケーションの性能を高める、JEUS Webキャッシュを適用する方法について説明します。

- 「第8章 リバース・プロキシ」

Webアプリケーションの性能を高める、リバース・プロキシの基本概念および使用方法について例を挙げて説明します。

- 「第9章 クラスの動的反映」

Webアプリケーションの開発速度を高める、サーブレットの自動リロード機能について説明します。

- 「第10章 実習」

JEUSを簡単に体験できるように例を挙げて説明します。

表記上の規則

表記	意味
<<AaBbCc123>>	プログラム・ソースコードのファイル名
<Ctrl>+C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
ハイパーリンク	メール・アカウント、Webサイト
>	メニューの実行順
+----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
参考	参照/注意事項
注	注意事項
[図 1.1]	図の名前
[表 1.1]	表の名前
AaBbCc123	Javaコード、XMLドキュメント
[<i>command argument</i>]	オプション・パラメータ
< xyz >	「<」と「>」の間の内容は実際に使用される特定の名前または値で置き換えられる
	構文の中の相互に排他的な選択項目の選択肢を示す 例) A B: AとBのいずれかを選択
...	パラメータ、値、または他の情報が繰り返される
\${ }	環境変数

システム要件

	要求事項
プラットフォーム	Solaris 9, 10, 11
	HP-UX 11.x, 11i, 11iV2
	IBM AIX 5L, 6L, AIX 7L
	MS Windows 2008, 2012, Vista, 7, 8
ハードウェア	最小2GB以上、推奨20GBのハードディスク容量
	推奨1GB以上のメモリー容量
JDK	JDK 7, JDK 8

関連文書

ガイド	説明
JEUS 紹介ガイド	JEUSサーバーについて全般的に紹介し、JEUSのアーキテクチャーを含む各構成要素について記述しています
JEUS インストール&スタートガイド	JEUSについて紹介し、JEUSのインストールおよび開始方法について記述しています
JEUS サーバガイド	JEUSシステムおよびサーバーの概要とシステムの管理方法について記述しています
JEUS Webサービスガイド	JEUSでのWebサービスについて記述しています
JEUS WebAdminガイド	JEUSのWeb管理ツールであるWebAdminを利用したJEUSの設定および制御、モニタリング、クラスタリング、リソースの設定および管理について記述しています
JEUS セッション管理ガイド	JEUSで運用するセッション・マネージャーとセッション・サーバーの設定および管理について記述しています
JEUS リファレンスガイド	JEUSを使用するために必要な詳細設定とJEUSの使用方法について記述しています
JEUS WebAdminガイド	JEUSのWeb管理ツールであるWebAdminを利用したJEUSの設定および制御、モニタリング、クラスタリング、リソースの設定および管理について記述しています

参考文献

- Java EE 7標準
- Servlet 3.1標準
- JSP 2.3標準(JSR 245 Maintenance Release)
- EL 3.0標準(JSR 245 Maintenance Release)
- JEUS_HOME/docs/reference/schema/index.htmlのXML Reference

Webエンジンの設定について詳しく説明します。

- domain.xmlのWeb/サーブレット・エンジン
- domain.xmlのWebエンジン内のエンコーディング
- domain.xmlのWebエンジン内のJSPエンジン

- domain.xmlのWebエンジン内のWeb接続(Webリスナー、Webサーバーの接続)
- domain.xmlのWebエンジン内のセッション
- domain.xmlのWebエンジン内の仮想ホスト(Virtual Host)
- JEUS_HOME/docs/reference/schema/index.htmlのjeus-web-dd.xml XML Reference
JEUSコンテキスト(Webアプリケーション)デプロイメント・ディスクリプターについて詳しく説明します。
- JEUS_HOME/docs/api/jeusapi/index.htmlのWebcontainer Tcp Listener API Reference
TCPリスナーを使用した通信プロトコルを実装するためのAPI([「第2章 Webコネクションの管理」](#)を参照)について説明します。

お問合せ先

Korea

TmaxSoft Co., Ltd.
45, Jeongjail-ro, Bundang-gu,
Seongnam-si, Gyeonggi-do, 13613
South Korea
Tel: +82-31-8018-1000
Fax: +82-31-8018-1115
Email: info@tmax.co.kr
Web (Korean): <http://www.tmaxsoft.com>
TechNet: <http://technet.tmaxsoft.com>

USA

TmaxSoft Inc.
101 North Wacker Drive, Suite 2014,
Chicago, IL 60606
U.S.A
Tel: +1-312-525-8330
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/us_en/home

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: info@tmaxsoft.co.jp
Web (Japanese): <http://www.tmaxsoft.co.jp>

China

Beijing TmaxSoft System Software Co., Ltd.
Room103, No.2 Huizhong Building, Seven Street Shangdi,
Haidian District, Beijing, 100085
P.R.China
Tel: +86-10-6298-8827
Email: info@tmaxsoft.com.cn
Web (Chinese): http://www.tmaxsoft.com/cn_en/home_cn_en

Brazil

Tmax Brasil Sistemas e Serviços Ltda.
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel
Alphaville Barueri, Sao Paulo, 06472-001
Brazil
Tel: +55-11-4191-3100
Fax: +55(11) 4191-3705 (extension#112)
Email: info.bra@tmaxsoft.com
Web (Portuguese): http://www.tmaxsoft.com/br_en/home_br_en

Russia

Tmax Rus L.L.C.
Leninsky prospekt, 113/1 (Park Place Moscow),
Office 318e, Moscow, 117198
Russia
Tel: +7(495)970-01-35
Email: info.rus@tmaxsoft.com
Web (Russian): http://www.tmaxsoft.com/ru_ru/home_ru_ru

Singapore

Tmax Singapore Pte. Ltd.
430 Lorong 6, Toa Payoh #10-02,
OrangeTee Building, 319402
Singapore
Tel: +65-6259-7223
Fax: +65-6258-7112
Email: info.sg@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/sg_en/home_sg_en

United Kingdom

TmaxSoft UK Ltd.
215 Knyvett House, Watermans Business Park,
The Causeway, Staines TW18 3BAB
United Kingdom
Tel: +44-1784-895005
Email: info.uk@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/gb_en/home_gb_en

Canada

TmaxSoft Canada, Inc.
2425 Matheson Blvd East, 8th floor,
Unit 824 Mississauga, ON, L4W 5K4
Canada
Tel: +1-905-361-2888
Email: info.canada@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/ca_en/home_ca_en

Australia

TmaxSoft Proprietary Limited
L32, 101 Miller Street, North Sydney 2060
Australia
Tel: +91-9845-330-704
Email: info.aus@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/au_en/home_au_en

India

TmaxSoft Technologies Private Limited
Sobha Alexander Plaza, 3rd Floor,
16/2 Commissariat Road, Bangalore-560025
India
Tel: +91-9845-330-704
Email: info.india@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/in_en/home_in_en

Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office
Windowist Tower. Eski Buyukdere Cad. No:26,
Maslak 34467 Istanbul
Turkey
Tel: +90-544-553-6045
Email: cslee@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/tr_en/home_tr_en

第1章 Webエンジン

本章では、JEUS Webエンジンの構成要素、主要機能、基本的な設定方法について説明します。

1.1. 概要

Java EEWebアプリケーション(以下、Webアプリケーション)は、ユーザーにWebコンテンツを動的に提供するためのサーブレット、JSP、HTMLファイルなどの静的リソースで構成されています。JEUSは、このようなWebアプリケーションを効率的に管理するWebエンジンを提供します。

本章では、JEUS Webエンジンを紹介し、運用環境で制御およびモニタリングする方法について説明します。

1.2. 構成要素

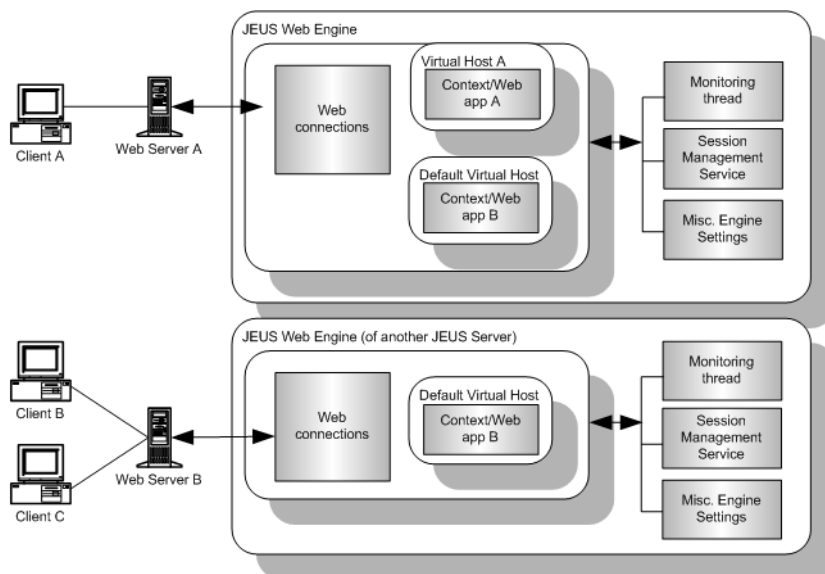
本節では、Webエンジンの構成要素およびWebエンジンに関連する外部要素について説明します。

WebエンジンはJava EE、サーブレット、JSP、EL標準に準拠するWebアプリケーションを管理、サービスするオブジェクトです。サーブレット、JSPのような動的リソースのみならず、HTMLのような静的リソースもサービスできます。

参考

Webエンジンは、Webコンテナまたはサーブレット・エンジンともいいます。

[図 1.1] Webエンジンの構成要素



主な構成要素は以下のとおりです。

- **Web Connections**

Web Connections(以下、Webコネクション)は、クライアント、Webサーバーまたはその他のサーバーとWebエンジンが接続されるための要素です。

複数のWebサーバーとWebエンジンが接続され、性能や安定性を高めた大きなクラスタリングで構成することができます。詳しい内容は、「[第2章 Webコネクションの管理](#)」を参照してください。

- **仮想ホスト**

クライアントが異なるホスト名を指定して呼び出した場合、各ホスト名にマッピングされたWebアプリケーションが呼び出されるようにする仮想のグループです。特定の仮想ホストを指定してWebアプリケーションをデプロイすることができます。詳しい内容は、「[第5章 仮想ホスト](#)」を参照してください。

- **Webコンテキスト(Webアプリケーション)**

サーブレット、JSP標準に準拠するWebアプリケーションをWebエンジンにデプロイすると、これに対するWebコンテキストを作成します。Webアプリケーションは、一般的にWAR(Web ARchive)形式でパッケージングされ、Webエンジン内の仮想ホストにデプロイされます。詳しい内容は、「[第3章 Webコンテキスト](#)」を参照してください。

- **モニタリング・スレッド**

Webエンジンの様々な構成要素をモニタリングします。詳しい内容は、「[1.3. 主要機能](#)」のモニタリングを参照してください。

- **セッション管理サービス**

分散された環境においてクライアントのセッションを管理し、複数のWebエンジンで使用可能な環境を提供します。詳しい内容は、『*JEUS セッション管理ガイド*』の「[第1章 セッション・トラッキング](#)」を参照してください。

- **その他のWebエンジンの設定**

Webエンジンに共通的に適用される設定です。基本エラー・ページ、エンコーディング、JSPエンジン、応答ヘッダーの設定などが該当されます。

参考

JEUS 6まで存在していたコンテキスト・グループは、JEUS 7からは削除されました。コンテキスト・グループのほとんどの機能は、Webエンジンが行います。

Webエンジンと関連する外部要素は以下のとおりです。

- クライアント

JEUS Webエンジンにサービスを要求します。HTTPベースのWebブラウザが一般的です。

- Webサーバー

クライアントのHTTP要求を受け、必要に応じてJEUSのWebエンジンに渡します。JEUS Webエンジンが単独でHTTPクライアントの要求を処理するより、Webサーバーとの組み合わせで使用される場合が一般的です。詳しい内容は、「[第2章 Webコネクションの管理](#)」を参照してください。

1.3. 主要機能

Webエンジンの主要機能は以下のとおりです。

- Webアプリケーションの管理

Webエンジンは、Java EE標準に準拠するWebアプリケーションのデプロイ/アンデプロイ、およびサービスの一時停止機能を提供します。以外にも、サービス無停止での再デプロイ機能を提供します。詳細内容については、「[第3章 Webコンテキスト](#)」を参照してください。

- モニタリング

モニタリングとは、Webエンジンのリソースの状態をチェックし、問題が発生したときに対応するための機能です。モニタリングには大きく3つのタイプがあります。それぞれの詳しい設定については、「[1.6.2. モニタリングの設定](#)」を参照してください。

区分	説明
スレッド・プールのモニタリング	各Webコネクションが持っているスレッド・プールをモニタリングします。詳しい内容は、「 第2章 Webコネクションの管理 」を参照してください
クラス・ローダーのモニタリング	Webアプリケーションのサブレッド・クラスの変更をモニタリングします。詳しい内容は、「 第3章 Webコンテキスト 」を参照してください
セッション・サービスのモニタリング	クライアント・セッションの有効期限をチェックし、期限切れのセッションを削除します。詳しい内容は、『 <i>JEUS セッション管理ガイド</i> 』の「第1章 セッション・トラッキング」を参照してください

- 基本エラー・ページの設定

Webエンジンは、コンテキストが準備されていないときに表示するエラー・ページを設定することができます。設定についての詳細内容は、「[1.6.3. 基本エラー・ページの設定](#)」を参照してください。

● エラー発生の場合、スタック・トレース添付の設定

Webエンジンでは、エラーが発生したとき、その時点のスタック・トレースを基本エラー・ページに添付するか否かを設定することができます。設定についての詳細内容は、「[1.6.4. エラー発生の場合、スタック・トレース添付の設定](#)」を参照してください。

● エンコーディングの設定

Webエンジンは、登録されたすべてのコンテキストによって使用できるエンコーディング設定を持っています。設定についての詳細内容は、「[1.6.5. エンコーディングの設定](#)」を参照してください。

● JSPエンジンの設定

JSPエンジンは各Webアプリケーションごとに作成されます。JSPエンジンは、クライアントがJSPリソースを要求したとき、JSPページをJavaファイルに変換した後、Javaファイルをコンパイルしてサーブレット・バイト・コードに生成する作業を行います。実行結果として、1つのJSPに対し、Javaファイルとクラス・ファイルが生成されます。

ただし、Javaファイルはjsp作成時の問題点を確認するためにデバッグ目的で使用されます。設定についての詳細内容は、「[4.4. JSPエンジンの設定](#)」を参照してください。

● 応答ヘッダーの設定

Webエンジンは、ユーザー任意のHTTP応答ヘッダー(Response Header)の名前と値をペアで設定することができます。同設定によって、Webエンジンから送信されるすべての応答には定義されたヘッダーが常に含まれます。

設定についての詳細内容は、「[1.6.7. 応答ヘッダーの設定](#)」を参照してください。

● セッション管理

Webエンジンのセッション管理に関する様々な事項を設定します。セッション・クラスタリングへの参加、セッション・オブジェクトの共有、セッション・クッキーの設定、タイムアウトなど、Webエンジンのセッションに関するすべての設定が行えます。設定についての詳細内容は、「[1.6.9. セッションの設定](#)」と『JEUS セッション管理ガイド』の「第1章 セッション・トラッキング」を参照してください。

● イベント・ロギング

Webエンジンでは、サーバー・ログとアクセス・ログが記録されます。さらに、Webアプリケーションで、サーブレット・コンテキストAPIのログ・メソッドで残すユーザー・ログがあります。

区分	説明
サーバー・ログ	Webエンジンの動作に関連するログです
アクセス・ログ	Webエンジンの要求および処理結果に関するログです

区分	説明
ユーザー・ログ	javax.servlet.ServletContext.log(String msg)またはjavax.servlet.ServletContext.log(String msg、Throwable t) APIを使用してWebアプリケーション内で生成されるメッセージを記録するログです

JEUSログの基本位置は、「[1.5. ディレクトリー構造](#)」を参照し、ログ設定についての詳細内容は、「[1.6.10. ログの設定](#)」を参照してください。

● グレースフル・シャットダウン

管理者がJEUSサーバーをシャットダウンするとき、実行中のサービスの完了を保証する機能です。実際に管理者がシャットダウンを指示したとき、以下の順で動作します。

1. 新しいサービス要求を受けません。
2. その時点で実行中の要求の完了を保証し、別のシャットダウンを行います。

実行中のサービスの完了に長時間かかる場合、無限に待つことはできないため、サーバーにシャットダウンを指示するときのタイムアウト・オプションを設定することができます。グレースフル・シャットダウンについての詳細内容は、『*JEUS サーバガイド*』の「3.1.3. Managed Serverの終了」を参照してください。

参考

JEUS 6まで提供していたWEBMain.xmlのshutdown-timeout設定は削除されました。以降のバージョンからは、コンソール・ツール(jeusadmin)でshutdown timeoutオプションを設定すると、同様な機能が使用できます。

● Web攻撃への対応

DoS(Denial of Service)攻撃のようなWeb攻撃によるJEUSサーバーの負担を減らすための設定です。

Web攻撃を防ぐため、POST要求のサイズ、GET/POST要求に含まれたパラメータ数、1つの要求に含まれたヘッダー数/サイズ、GET要求のクエリー・ストリング・サイズなどの制限が設定できます。設定についての詳細は、「[1.6.13. Web攻撃に対応する設定](#)」を参照してください。

1.4. 管理ツール

本節では、Webエンジンの制御およびモニタリング機能を提供するツールについて説明します。

1.4.1. ツールの種類

JEUS Webエンジンを運用するためのツールは以下のとおりです。

- **WebAdmin**

Webブラウザを利用してWebエンジンが管理できる運用ツールです。

JEUS WebAdmin(以下、WebAdmin)の使用方法は、『JEUS WebAdminガイド』を参照してください。

- **コンソール・ツール(jeusadmin)**

OSコンソールでWebエンジンが制御できる運用ツールです。基本的な制御機能およびモニタリング機能を提供します。コンソール・ツールの使用方法は、『JEUS サーバガイド』の「第3章 JEUSサーバーの制御とモニタリング」と『JEUS リファレンスガイド』の「4.2.8. Webエンジン関連コマンド」を参照してください。

1.4.2. 制御およびモニタリング

運用ツールを使用してWebエンジンの制御およびモニタリングが可能です。

Webエンジンの制御

Webエンジンの制御とは、Webエンジンの開始と終了を制御することを意味します。この2つの作業は、WebAdminとコンソール・ツールを使用して実行できます。

- **WebAdminの使用**

詳しい内容は、『JEUS サーバガイド』の「第3章 JEUSサーバーの制御とモニタリング」のサーバー制御についての内容を参照してください。

- **コンソール・ツールの使用**

WebエンジンはJEUSサーバーに含まれており、サーバーを単独で制御する方法はありません。詳しい内容は、『JEUS サーバガイド』の「第3章 JEUSサーバーの制御とモニタリング」を参照してください。

Webエンジンのモニタリング

WebAdminとコンソール・ツールを使用してWebエンジンをモニタリングすることができます。

- **WebAdminの使用**

WebAdminのエンジン統計部分を利用してモニタリングできます。

1. WebAdminの左側のメニューから**[Monitoring]**を選択すると、下位にモニタリング可能なモジュールが表示されます。こちらから**[Web]**メニューを選択すると、Webエンジンの統計情報を照会、または初期化することができます。

[図 1.2] Webエンジンのモニタリング - メニュー移動とサーバーの選択

The screenshot shows the 'Web' monitoring interface. At the top, there's a 'Web' logo and a 'HISTORY' dropdown. Below that, a message says 'WEBのモニタリング情報を照会します。' (Retrieve WEB monitoring information). There are tabs for 'Web Statistics' and 'List Session'. A search bar contains 'adminServer' with a 'Clear' button. The main content area displays several sections:

- Standard session information (adminServer_web)**: A table with columns 'Manager name' and 'Active session'. It shows 'webadmin_02' with 1 active session.
- Distributed session information: (adminServer_web)**: A table with columns 'Manager name', 'Active session', 'File passivated', and 'Backup server'. It shows 'app1' with 0 active sessions and 0 passivated files, and '- backup' with 0 active sessions and 0 passivated files, with 'server1' as the backup server.
- Thread information : (adminServer_web)**: A table with columns 'Thread pool name', 'Current thread count', 'Min thread count', 'Max thread count', and 'Current queue count'. It shows 'ADMIN-HTTP' with 7 current threads, 1 min, 32 max, and 0 queue, and 'http1' with 10 current threads, 10 min, 20 max, and 0 queue.
- Request information of contexts**: A table with columns 'Context name', 'Total requests', 'Successful requests', 'Successful processing time', 'Average processing time', and 'Async requests'. It shows 'app1' with 3 total requests, 3 successful, 37 ms processing time, 12.0 ms average, and 0 async requests.
- Memory information : (adminServer_web)**: A table with columns 'VM total memory' and 'VM free memory'. It shows 241696768 bytes total and 111966688 bytes free.

At the bottom, there's a section for 'server1'.

以下は、[Web]画面で照会される各統計情報についての説明です。

統計情報	説明
Standard session information	選択されたサーバーのWebエンジンで動作中のセッション・マネージャーの統計情報です。基本的にセッション数を表示します
Distributed session information	選択されたサーバーのWebエンジンで動作中のセッション・マネージャーの統計情報です。同情報はクラスター環境でのみ表示され、マネージャーが有するセッションとバックアップ・サーバーの情報を表示します。クラスター環境設定については、『JEUS セッション管理ガイド』の「2.2. 基本概念」を参照してください

統計情報	説明
Thread information	選択されたサーバーのWebエンジンに設定されているリスナーおよびコネクターのスレッド・プール内のスレッド状態の統計情報を表示します
Request information of contexts	選択されたサーバーのWebエンジンにデプロイされたWebアプリケーション別に累積された要求数、累積された要求の処理時間、平均的な要求処理時間などを表示します。 [Clear] ボタンをクリックすると、情報が初期化されます
Memory information	選択されたサーバーの総VMメモリーおよび使用可能はVMメモリーを表示します

2. **[Web]**画面からモニタリングするサーバーを選択すると、Webエンジンの統計情報が照会されます。統計情報画面の右上部の**[Clear]**ボタンをクリックすると、クリックした時点をもとにして**Request information of contexts**統計情報が初期化されます。

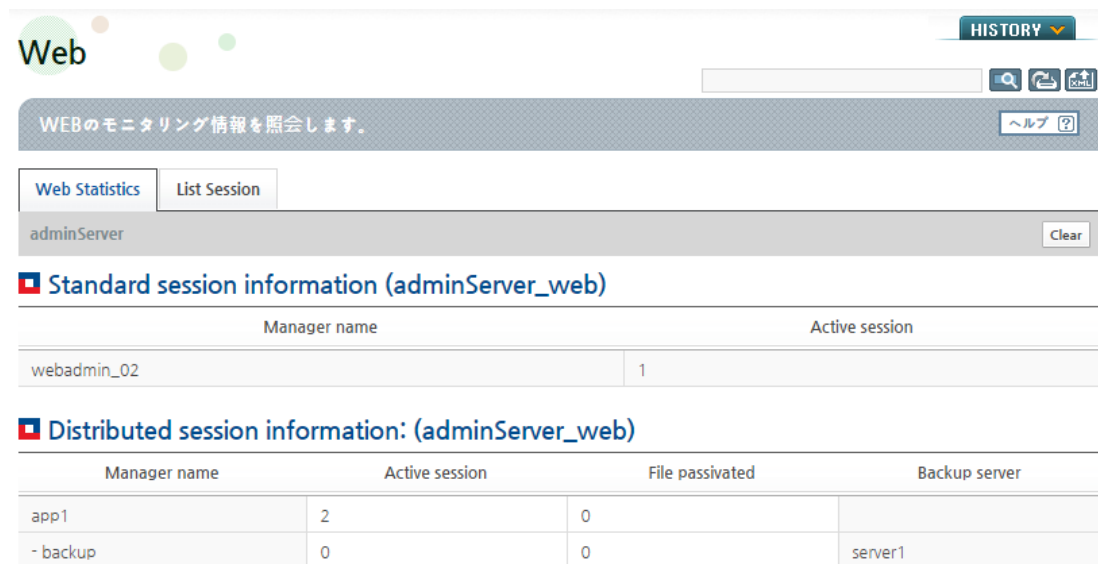
[図 1.3] Webエンジンのモニタリング - 情報照会

The screenshot displays the 'Web' monitoring interface. At the top, there's a 'HISTORY' dropdown and a search bar. A message box indicates 'The reset stats command succeeded: (adminServer)'. Below this, the 'Web Statistics' tab is active, showing 'adminServer' with a 'Clear' button. The interface is divided into several sections:

- Standard session information (adminServer_web)**: A table with columns 'Manager name' and 'Active session'. It shows 'webadmin_02' with 1 active session.
- Distributed session information: (adminServer_web)**: A table with columns 'Manager name', 'Active session', 'File passivated', and 'Backup server'. It shows 'app1' with 0 active sessions and 0 passivated files, and '- backup' with 0 active sessions and 0 passivated files, with 'server1' as the backup server.
- Thread information : (adminServer_web)**: A table with columns 'Thread pool name', 'Current thread count', 'Min thread count', 'Max thread count', and 'Current queue count'. It shows 'ADMIN-HTTP' with 7 current threads and 1 min thread count, and 'http1' with 10 current threads and 10 min thread count.
- Request information of contexts**: A table with columns 'Context name', 'Total requests', 'Successful requests', 'Successful processing time', 'Average processing time', and 'Async requests'. It shows 'app1' with 0 total requests and 0 successful requests.
- Memory information : (adminServer_web)**: A table with columns 'VM total memory' and 'VM free memory'. It shows 241696768 bytes of total memory and 66541464 bytes of free memory.

3. **session information**の「**Manager name**」を選択すると、該当のセッション・マネージャーが所有するセッション・リストが表示されます。

[図 1.4] セッション・リスト - マネージャーの選択



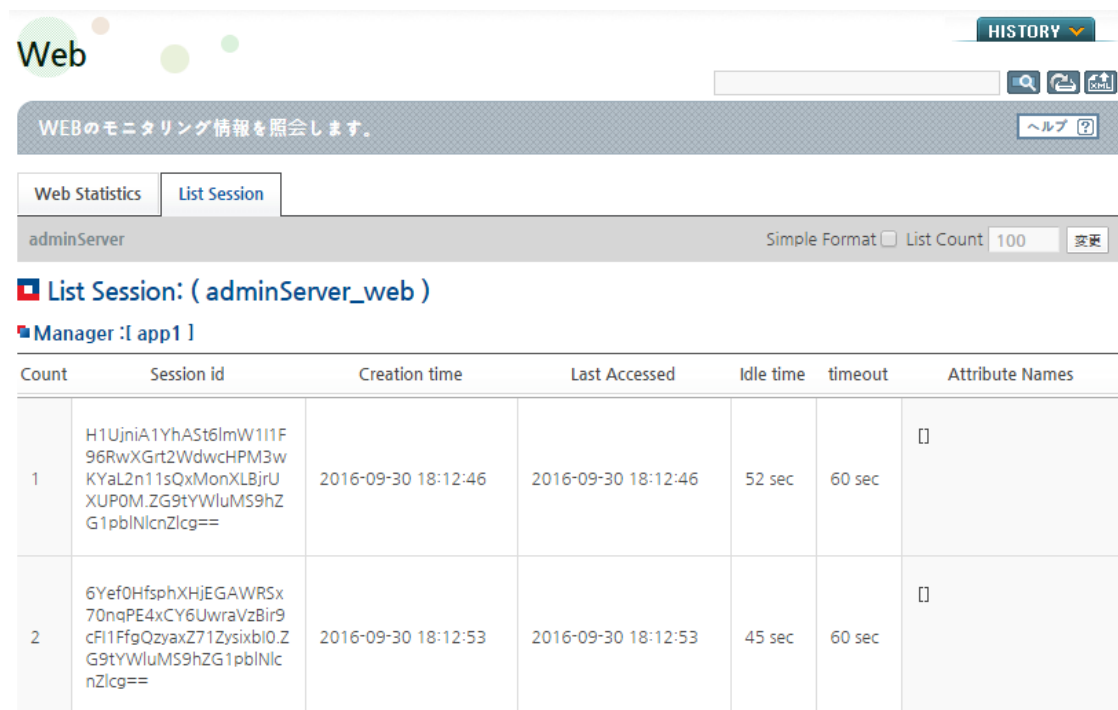
The screenshot shows the 'Web' monitoring interface. At the top, there's a 'HISTORY' dropdown and a search bar. Below that, a message says 'WEBのモニタリング情報を照会します。' with a 'ヘルプ' link. The 'List Session' tab is selected, and 'adminServer' is entered in the filter field. The 'Standard session information (adminServer_web)' table shows one entry: 'webadmin_02' with 1 active session. Below it, the 'Distributed session information: (adminServer_web)' table shows two entries: 'app1' with 2 active sessions and 'backup' with 0 active sessions, both with 0 file passivated and 'server1' as the backup server.

Manager name	Active session
webadmin_02	1

Manager name	Active session	File passivated	Backup server
app1	2	0	
- backup	0	0	server1

セッションを選択すると、該当するセッションの詳細情報が表示されます。

[図 1.5] セッション・リスト - マネージャーの結果



The screenshot shows the 'Web' monitoring interface with the 'List Session' tab selected. The filter field shows 'adminServer'. Below the filter, there are options for 'Simple Format' (unchecked) and 'List Count' (100). The 'List Session: (adminServer_web)' section shows the manager 'app1'. The table below lists two sessions with their details.

Count	Session id	Creation time	Last Accessed	Idle time	timeout	Attribute Names
1	H1UjniA1YhASt6lmW1I1F96RwXGrt2WdwchPM3wKYaL2n11sQxMonXLBjrUXUP0M.ZG9tYWluMS9hZG1pbINlcZlcg==	2016-09-30 18:12:46	2016-09-30 18:12:46	52 sec	60 sec	
2	6Yef0HfsphXHjEGAWRSx70nqPE4xCY6UwraVzBir9cFl1FfgQzyaxZ71ZysixblO.ZG9tYWluMS9hZG1pbINlcZlcg==	2016-09-30 18:12:53	2016-09-30 18:12:53	45 sec	60 sec	

Input list count = 100

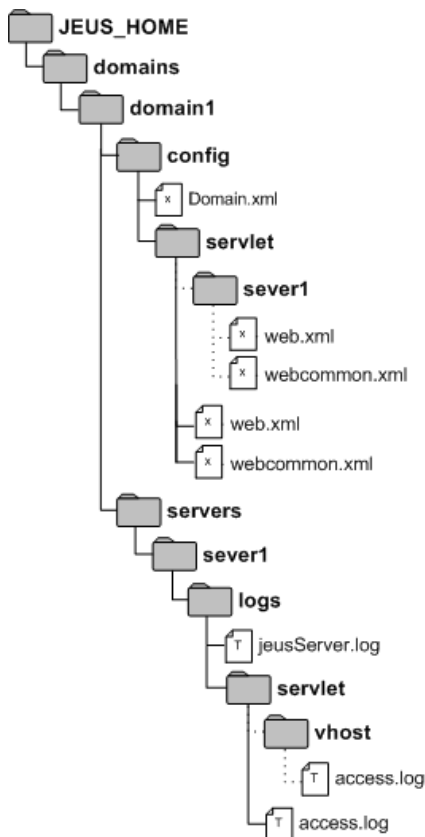
● コンソール・ツールの使用

コンソール・ツールでは、Webエンジンの基本情報が照会できる機能を提供します。詳しい内容は、『JEUS サーバガイド』と『JEUS リファレンスガイド』の「4.2.8. Webエンジン関連コマンド」を参照してください。

1.5. ディレクトリー構造

以下は、JEUSのディレクトリー構造におけるWebエンジンの構造です。

[図 1.6] JEUSのディレクトリー構造でのWebエンジン



JEUS_HOME

JEUSのインストール・ホーム・ディレクトリーです。

domains/domain1/config/

JEUS_HOMEのdomainsディレクトリーの下位には、各ドメイン名を基準にしてディレクトリーが存在します。そのうち、Webエンジンの設定に関連する内容がconfigディレクトリーの下位に位置します。domain1は例として使用したドメイン名です。

domain.xmlは統合された設定ファイルです。同ファイルにはドメインに属するすべてのサーバー設定が存在しており、サーバー別にWebエンジンが設定できます。

参考

1. domain.xmlについての詳細内容は、『JEUS サーバガイド』を参照してください。
2. JEUS 7からはWEBMain.xmlを使用しません。1つのサーバーJVMには、1つのWebエンジンのみ存在するため、エンジンごとの設定ディレクトリーは不要です。

– servlet/

ドメインに属するすべてのサーバーが各Webアプリケーションをデプロイする際、基本的に参照するXML設定ファイルが存在します。

ファイル	説明
web.xml	サーブレット3.0からはweb.xmlは不要ですが、JSPパーサーであるTomcat Jasperでは同ファイルをベースにしてWebアプリケーションのバージョンを確認するため、そのまま保持する必要があります。web.xmlファイルを持っていないWebアプリケーションでこのファイルを参照します
webcommon.xml	すべてのWebアプリケーションに適用される共通設定ファイルです。このファイルの形式はweb.xmlと同様です

参考

このディレクトリーに存在するweb.xmlとwebcommon.xmlは、ドメインに属するサーバーが起動されるときドメイン管理サーバーから取得するため、同じファイルを保持することになります。

– servlet/server1/

サーブレット設定ディレクトリーの下位に、特定のサーバー名でディレクトリーを生成し、その下位にwebcommon.xml、web.xmlを格納すると、特定のサーバーに対してのみ適用することができます。このディレクトリーの設定が上位ディレクトリー設定より優先されます。

参考

特定サーバー名のディレクトリーに存在するweb.xmlとwebcommon.xmlはJEUSが自動で管理しないため、サーバー管理者が直接管理する必要があります。

domains/domain1/servers/server1/logs

ドメインに属する各サーバーが動作するときに生成されるファイルは、serversの下位にサーバー別の名前で生成されたディレクトリーの下位に存在します。詳しい内容は、『JEUS サーバガイド』の「第8章 ロギング」を参照してください。

ログ・ファイルは、基本的にlogsディレクトリーの下位に存在します。Webエンジンで生成したログはJeusServer.logに保存されます。ただし、ログ・ローテーション機能によってファイル名が変更されることはありますが、常にJeusServerで始まります。

– servlet/

Webアプリケーション・アクセス・ログであるaccess.logを保存します。仮想ホスト別にアクセス・ログを設定した場合は、基本的にこのディレクトリーの下位に仮想ホスト名でディレクトリーを生成し、同仮想ホストにデプロイされたWebアプリケーションのアクセス・ログを保存します。詳しい内容は、「[第5章 仮想ホスト](#)」を参照してください。

同ディレクトリーには、WebアプリケーションでServletContext APIのログ・メソッドを使用して記録するユーザー・ログのuser.logまたはuser_appname.logファイルが保存されます。ユーザー・ログは、設定によってuser.logファイルに残ることも、Webアプリケーションの別途ファイルであるuser_appname.logに残る場合もあります。

1.6. 環境設定

本節では、Webエンジンに関連する環境設定ファイルとWebエンジンの設定について説明します。

Webエンジンの設定はWebAdminを使用することをお勧めします。WebAdminを使用する設定には、以下のとおり設定関連の共通的な事項についての知識が必要です。WebAdminの使用方法については、『JEUS WebAdminガイド』を参照してください。

- **[LOCK & EDIT]**ボタン

各項目を設定および修正するときは、左側の**[LOCK & EDIT]**ボタンをクリックし、設定変更モードに切り替えます。設定変更モードに切り替えないと設定および修正できません。

- **[確認]**および**[再設定]**ボタン

設定変更モードに切り替えて設定を変更した後は、**[確認]**ボタンをクリックして修正および設定内容を保存します。**[再設定]**ボタンをクリックすると、修正および設定する前の内容に戻ります。

- **[Activate Changes]**ボタン

修正および設定内容を選択されたサーバーに適用するには、必ず**[Activate Changes]**ボタンをクリックします。ただし、動的に適用されない設定の場合は、当該設定がペンディングされたとのメッセージと共に「変更内容を適用するためにはサーバーを再起動する必要があります。」とのメッセージが出力されます。そのときは、サーバーを再起動してペンディングされた設定を実際にサーバーに適用します。

[図 1.7] 設定の適用 - Activate Changes

The screenshot shows the JBoss Management Console interface. On the left, a sidebar contains a navigation menu with items like Domain, Session, Clusters, Servers, Applications, Security, Resources, Monitoring, and Console. The 'Servers' item is selected. Below the menu, there's a 'システム状態' (System Status) section showing a bar chart with 2 'Running' servers. Further down, there are buttons for 'Runtime Info', 'Activate Changes', and 'Undo All Changes'. A warning message in Japanese is displayed below these buttons. The main content area is titled 'Servers' and contains a table with columns: Name, Status, Pid, Need To Restart, Command, and actions (Delete, Duplicate). The table lists two servers: 'adminServer (*)' and 'server1', both with status 'RUNNING'. Below the table is a 'Server Templates' section with an 'Add' button and a message stating '該当する内容が存在しません。' (No matching content exists).

● Webエンジンの設定

Webエンジンの設定には、基本設定と詳細設定があります。

－ 基本設定

基本的な設定項目として、各項目についての説明が表示されます。

－ 詳細設定

基本設定以外に詳細設定のための項目で構成されており、基本的には各設定項目についての説明は表示されません。詳細設定の「すべてを開く」をクリックすると、各項目の説明が表示されます。

[図 1.8] 詳細設定

詳細設定

すべてを開く

Encoding

Request Url Encoding

Default

Forced

Request Encoding

Default

Client Override

Forced

Response Encoding

Default

Forced

Cookie Policy

Write Value On Header Policy

Apply Url Encoding Rule

Charset Encoding

Response Header

Custom Header

name=value

Monitoring

Check Thread Pool

300000ms

Check Class Reload

300000ms

Check Session

300000ms

Blocked Url Patterns

Deny Last Space Character

Deny Null Character

Encoded Pattern

%23

%2e

%2f

%5c

%00

Decoded Pattern

#

W

1.6.1. XML設定ファイル

以下は、Webエンジンと関連されているXML設定ファイルについての説明です。

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

位置	JEUS_HOME/domains/<domain-name>/config
目的	JEUSサーバー別のWebエンジンの設定
詳細情報	『JEUS サーバガイド』

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

位置	<packaged web application>/WEB-INF/
目的	JEUS WebモジュールのDeployment Descriptor(以下、DD)
詳細情報	本書

- **web.xml** (web-app_3_1.xsd)

位置	<packaged web application>/WEB-INF/
目的	Java EE標準のWebモジュールDD
詳細情報	Servlet 3.1標準

- **web.xml** (web-app_3_1.xsd)

位置	JEUS_HOME/domains/<domain-name>/config/servlet
目的	web.xmlを個別に持っていないWebモジュールのweb.xml
詳細情報	Servlet 3.1標準

- **webcommon.xml** (web-app_3_1.xsd)

位置	JEUS_HOME/domains/<domain-name>/config/servlet
目的	WebエンジンのすべてのWebモジュールに適用される共通設定ファイル
詳細情報	Servlet 3.1標準

XMLスキーマ・ファイルはJEUS_HOME/lib/schemas/jeus/ディレクトリーに存在します。

上記のファイルは、JEUSで定義されたXMLヘッダーで始まる必要があります。また、ルート要素は、JEUS XMLスキーマの名前空間で指定します。

各ファイルに使用されるヘッダーは以下のとおりです。web.xmlのXMLヘッダーはサーブレット標準に含まれています。

- domain.xmlのXMLヘッダー

[例 1.1] XMLヘッダー : <<domain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="8.0">
```

- jeus-web-dd.xmlのXMLヘッダー

[例 1.2] XMLヘッダー : <<jeus-web-dd.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

- web.xmlのXMLヘッダー

[例 1.3] XMLヘッダー : <<web.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://java.sun.com/xml/ns/javaee">
```

参考

本書で使用されるすべてのタグ順は、XMLスキーマの設定順で作成されています。実際に使用するときには順序を守ることが困難なため、JEUSではタグ順を自動でソートする機能を提供しています。したがって、XML設定ファイルを作成する際、必ず順序を守る必要はありません。タグ順については、『JEUS XMLリファレンスガイド』を参照してください。

上記ファイルの例が本書で使用されるときは、便宜上標準ヘッダーが省略されます。実際のXML設定ファイルにはこのヘッダーが含まれていることに注意してください。

1.6.2. モニタリングの設定

Webエンジンは、内部スレッド、クラスの変更事項、セッションの変更事項をモニタリングすることが可能で、これらのモニタリング周期を変更することができます。

WebAdminを使用してモニタリング周期を設定する方法は以下のとおりです。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択するとサーバーの設定画面へ移動します。設定画面で[Engine] > [Web Engine] > [Basic]メニューを選択します。

[図 1.9] モニタリング - 設定メニューへ移動

Web Engine

HISTORY

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

ヘルプ

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

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

Attach Stacktrace On Error	<input type="checkbox"/> [デフォルト: false] JEUSが返すエラーページにスタックトレースを添付するかを設定します。この機能は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。
Async Timeout Min Threads	1 [デフォルト: 1] Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。
Properties	<div>key=value</div> Webエンジンに適用される属性を設定します。
Default Error Page	<div></div> Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはリダイレクトされず、HTTP応答本文の内容として使用されます。

2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. Web Engine設定画面の詳細設定からMonitoringの各項目を設定し、[確認]ボタンをクリックします。

[図 1.10] モニタリング - 設定

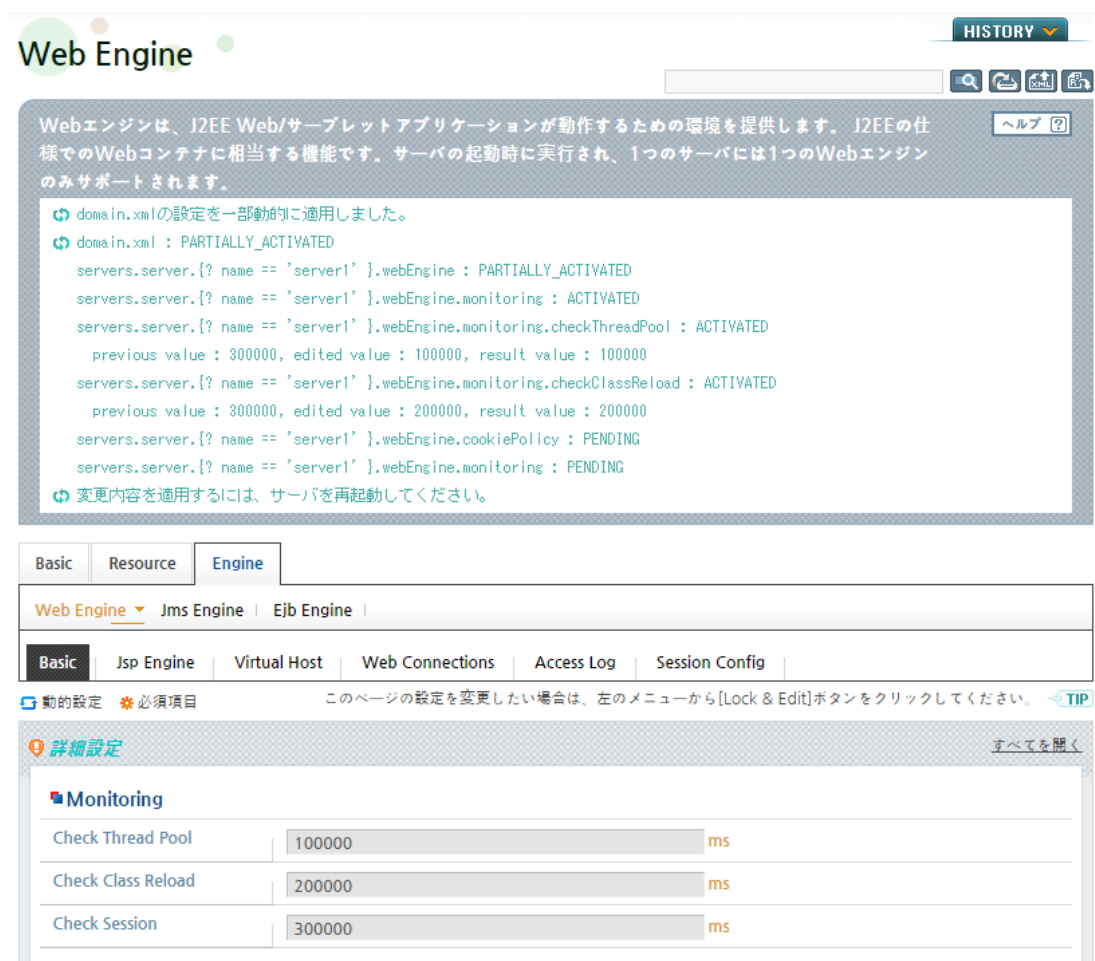


各設定項目についての説明は以下のとおりです。

項目	説明
Check Thread Pool	要求処理スレッドの状態をチェックするための時間間隔を設定します (デフォルト値: 300000(5分))
Check Class Reload	Webコンテキストを自動でリロードするため、クラスの変更有無をチェックする周期を設定します。(デフォルト値: 300000(5分)) 同設定は、jeus-web-dd.xmlに<auto-reload><enable-reload>機能を設定してから意味を持ちます。<check-on-demand>をtrueに設定したWebコンテキストの場合はチェックしません
Check Session	セッションのタイムアウト状態のチェック周期を設定します。セッションのタイムアウトは、Webエンジンのセッション設定またはweb.xmlに設定されます(デフォルト値: 300000(5分))

- 設定が終わったら、設定内容を適用するために[**Activate Changes**]ボタンをクリックします。([図 1.7]を参照)
- モニタリングの設定内容が反映されると、以下のとおり結果が出力されます。モニタリングの設定は動的に反映できないため、内容を適用するにはサーバーを再起動する必要があります。

[図 1.11] モニタリング - 設定の適用結果



1.6.3. 基本エラー・ページの設定

Web要求は、Webコンテキストが存在する場合にエラーが発生すると、当該Webコンテキストでエラーを処理します。一方、Webコンテキストが存在しない場合は、Webエンジンが内部的にエラー・ページを送る方式でエラーを処理する必要があります。この場合、エラー・ページがユーザーの希望する形式でない場合があるため、ユーザーの要求事項が処理できるエラー・ページの絶対パスが指定できる設定を提供します。

ただし、この場合にはhtmlまたはhtmファイルのみ設定可能です。こちらで設定された値は、HTTP応答内容として使用されるだけであり、フォワード概念ではありません。

WebAdminを使用して基本エラー・ページを設定する方法は以下のとおりです。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択するとサーバーの設定画面へ移動します。設定画面で[Engine] > [Web Engine] > [Basic]メニューを選択します。([図 1.9]を参照)

2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. [Web Engine]画面で基本エラー・ページと関連する「Default Error Page」項目にエラー・ページの絶対パスを設定し、[確認]ボタンをクリックします。

[図 1.12] 基本エラー・ページ - 設定

The screenshot shows the 'Web Engine' configuration interface. At the top, there's a 'Web Engine' header and a 'HISTORY' button. Below this is a descriptive text box about the Web Engine's role in J2EE. A status bar indicates '変更されました.' (Changes saved). The main configuration area has tabs for 'Basic', 'Resource', and 'Engine'. Under the 'Engine' tab, there are sub-tabs for 'Web Engine', 'Jms Engine', and 'Ejb Engine'. The 'Web Engine' sub-tab is active, showing further sub-tabs: 'Basic', 'Jsp Engine', 'Virtual Host', 'Web Connections', 'Access Log', and 'Session Config'. The 'Basic' sub-tab is selected, displaying a list of settings. The 'Attach Stacktrace On Error' setting is unchecked. The 'Async Timeout Min Threads' is set to 1. The 'Properties' section is empty. The 'Default Error Page' is set to '/jeus/user1/error.html'. A 'key=value' label is visible on the right side of the 'Properties' section. At the bottom right, there are '確認' (Confirm) and '再設定' (Reset) buttons.

Setting	Value
Attach Stacktrace On Error	<input type="checkbox"/>
Async Timeout Min Threads	1
Properties	
Default Error Page	/jeus/user1/error.html

4. 設定が終わったら、設定内容を適用するために[Activate Changes]ボタンをクリックします。([図 1.7]を参照)
5. 設定内容が反映されると、以下のとおり結果が出力されます。「Default Error Page」は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

[図 1.13] 基本エラー・ページ - 設定の適用結果

Web Engine

HISTORY

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

ヘルプ

domain.xmlの設定を変更しました。

domain.xml : PENDING

servers.server.{? name == 'server1' }.webEngine : PENDING

servers.server.{? name == 'server1' }.webEngine.defaultErrorPage : PENDING

previous value : null, edited value : /jeus/user1/error.html, result value : null

変更内容を適用するには、サーバを再起動してください。

BasicResourceEngine

Web EngineJms EngineEjb Engine

BasicJsp EngineVirtual HostWeb ConnectionsAccess LogSession Config

動的設定

必須項目

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

TIP

Attach Stacktrace On Error

☐

[デフォルト: false]

JEUSが返すエラーページにスタックトレースを添付するかどうかを設定します。この機能は開発時には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。

Async Timeout Min Threads

1

[デフォルト: 1]

Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。

Properties

key=value

Webエンジンに適用される属性を設定します。

Default Error Page

/jeus/user1/error.html

Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはダイレクトされず、HTTP応答本文の内容として使用されます。

1.6.4. エラー発生の場合、スタック・トレース添付の設定

Web要求の処理中にWebエンジンに問題が発生したときのエラーの詳細内容を、JEUSが基本的に提供するエラー・ページに表示するかについての設定です。このメッセージは開発時には有用ですが、運用するときは削除することが望ましいです。

この設定は、WebAdminを使用してWebエンジンに適用するか、アプリケーション別のjeus-web-dd.xmlに設定可能です。jeus-web-dd.xmlの設定は、「3.3.1. jeus-web-dd.xmlの設定」を参照してください。

エラーが発生した場合、WebAdminを使用してスタック・トレースの添付有無を設定する方法は以下のとおりです。

第1章 Webエンジン 21

1. WebAdminの左側のメニューから[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[**Engine**] > [**Web Engine**] > [**Basic**]メニューを選択します。([図 1.9]を参照)
2. 設定および設定変更のため、画面左側の[**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. [**Web Engine**]画面で「**Attach Stacktrace on Error**」項目をチェックして[**確認**]ボタンをクリックします。デフォルト値はfalseであり、項目をチェックすると基本エラー・ページにスタック・トレースが表示されます。ただし、「**Default Error Page**」項目が設定された場合は、設定されたエラー・ページが表示されます。
基本エラー・ページ関連の設定については、「[1.6.3. 基本エラー・ページの設定](#)」を参照してください。

[図 1.14] エラーの発生時にスタック・トレースの添付 - 設定

Web Engine

HISTORY

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

ヘルプ

変更されました。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

動的設定 必須項目

確認 再設定

Attach Stacktrace On Error	<input checked="" type="checkbox"/> <p>[デフォルト: false] JEUSが返すエラーページにスタックトレースを添付するか否かを設定します。 この機能は開発期間には有用ですが、運用環境では無効にする必要があります。 Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。</p>
Async Timeout Min Threads	<input type="text" value="1"/> <p>[デフォルト: 1] Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。 0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。</p>
Properties	<div> <input type="text"/> <input type="button" value="EX key=value"/> </div> <p>Webエンジンに適用される属性を設定します。</p>
Default Error Page	<input type="text" value="/jeus/user1/error.html"/> <p>Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。 静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。 このページは転送またはダイレクトされず、HTTP応答本文の内容として使用されます。</p>

4. 設定が終わったら、設定内容を適用するために[**Activate Changes**]ボタンをクリックします。([図 1.7]を参照)
5. 設定内容が反映されると画面に結果が出力されます。「**Attach Stacktrace on Error**」は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

[図 1.15] エラーの発生時にスタック・トレースの添付 - 設定の適用結果

Web Engine

HISTORY

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

domain.xmlの設定を変更しました。

domain.xml : PENDING

```
servers.server.[? name == 'server1' ].webEngine : PENDING
servers.server.[? name == 'server1' ].webEngine.attachStacktraceOnError : PENDING
previous value : false, edited value : true, result value : false
```

変更内容を適用するには、サーバを再起動してください。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

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

Attach Stacktrace On Error ☒

[デフォルト: false] JEUSが返すエラーページにスタックトレースを添付するかどうかを設定します。この機能は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。

Async Timeout Min Threads 1

[デフォルト: 1] Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。

Properties

key=value

Webエンジンに適用される属性を設定します。

Default Error Page /jeus/user1/error.html

Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはリダイレクトされず、HTTP応答本文の内容として使用されます。

1.6.5. エンコーディングの設定

Webエンジンは、エンジン内のすべてのコンテキストによって使用できる3つのエンコーディング設定を持っています。これらはすべてjeus-web-dd.xmlを使用してアプリケーション別に適用するか、仮想ホスト別、またはサーバー別に設定できます。羅列した順序どおりxml設定の優先順位が適用されます。管理者はWebAdminのような管理ツールを使用してdomain.xmlにエンコーディングを設定することができます。しかし、エンコーディングの設定は、Webアプリケーション別に異なる可能性が高いため、jeus-web-dd.xmlに設定することをお勧めします。

• Request Url Encoding

HTTP要求の要求行(Request Line)のうち、URIのクエリー文字列に使用されるエンコーディングです。

`ServletRequest.getParameter()`を呼び出したとき、クエリー文字列を解析して返されるStringオブジェクトに適用されます。ただし、`<request-encoding>`設定が適用された場合は、`forced`に設定した値のみ適用します。つまり、この設定は`forced`の場合のみ意味を持ちます。

– Request Url Encodingは、以下の優先順位で適用されます。

1. domain.xmlに定義された`<request-url-encoding>`の「forced」エンコーディング
2. domain.xmlに定義された`<request-url-encoding>`の「default」エンコーディング
3. ISO-8859-1

● Request Encoding

HTTP要求ヘッダーのクエリー文字列、クッキー、ボディに使用されるエンコーディングです。

– HTTP要求行のクエリー文字列は、以下の優先順位でエンコーディングが適用されます。サーブレットおよびJSPでフォワードあるいはインクルード時に記述するクエリー文字列にも適用されます。

1. domain.xmlに設定された`<request-url-encoding>`の「forced」エンコーディング
2. XMLに設定された`<request-encoding>`の「forced」エンコーディング
3. XMLに定義された`<request-encoding>`の「client-override」エンコーディング
4. XMLに設定された`<request-encoding>`内の`<url-mapping>`の「encoding」エンコーディング
5. XMLに設定された`<request-encoding>`の「default」エンコーディング
6. ISO-8859-1

– HTTP要求ヘッダーのクッキーのエンコーディングは、以下の優先順位で適用されます。

1. XMLに設定された`<cookie-policy><write-value-on-header-policy>`の「charset-encoding」値
2. XMLに設定された`<request-encoding>`の「forced」エンコーディング
3. XMLに定義された`<request-encoding>`の「client-override」エンコーディング
4. XMLに設定された`<request-encoding>`内の`<url-mapping>`の「encoding」エンコーディング
5. XMLに設定された`<request-encoding>`の「default」エンコーディング
6. ISO-8859-1

クッキーの優先順位は応答からクッキーをエクスポートするときも一緒に適用されます。クッキーのエンコーディング設定に一貫性を持たせるために、すべてのWebエンジンに1項の設定を同じく適用することをお勧めします。

– HTTPボディのエンコーディングは、以下の優先順位で適用されます。

1. XMLに定義された<request-encoding>の「forced」エンコーディング
2. アプリケーション(サーブレット/JSP)での設定(request.setCharacterEncoding())で設定したエンコーディング
3. XMLに定義した<request-encoding>の「client-override」エンコーディング
4. HTTP要求のコンテンツ・タイプの文字セットによるエンコーディング
5. XMLに設定された<request-encoding>内の<url-mapping>の「encoding」エンコーディング
6. XMLに定義された<request-encoding>の「default」エンコーディング
7. ISO-8859-1

POST要求であり、かつコンテンツ・タイプがapplication/x-www-form-urlencodedの場合、WebアプリケーションがServletRequest.getParameter()を呼び出すと、Webエンジンがボディを読み込んでパラメータを処理します。このパラメータで返されるStringオブジェクトを作成するとき、HTTPボディのエンコーディングを適用します。

ただし、ServletRequest.getParameter()でURIクエリー文字列も一緒に処理するため、<request-url-encoding><forced>設定されたエンコーディングと<request-encoding>設定されたエンコーディングが一致しないと、異なるエンコーディングが適用されたStringオブジェクトが返される可能性があります。そのため、クライアントでパラメータを送信するときは、クエリー文字列とPOSTボディのエンコーディングを一致させるのが望ましいです。

その他にも、ボディ・エンコーディングは、ServletRequest.getReader()を呼び出すときに適用されます。

旧バージョンのJEUSに依存してHTTP要求のエンコーディングを処理したアプリケーションは、上記のルールに従って変更するか、互換性オプションを適用する必要があります。

注意

JEUS 7 Fix#2から、forcedはサーブレットAPIより優先順位が高いです。JEUS 7 Fix#1以前のforced設定をそのまま維持する必要がある場合は、client-override設定に変更してください。

● Response Encoding

HTTP応答ボディに適用されるエンコーディングです。

応答エンコーディングは、ServletOutputStreamまたはPrintWriterのprintメソッドをバイト配列に変換するとき、HTTPヘッダーの「Content-Type:text/html;charset=XXX」の「XXX」値など、Webコンテナの応答にエンコーディングする文字を設定します。jeus-web-dd.xmlでも設定可能です。

– 応答エンコーディングは、以下の優先順位で適用されます。

1. <response-encoding>の「forced」エンコーディング
2. API呼び出しによるエンコーディング、JSP応答文字エンコーディング
3. <response-encoding>の「default」エンコーディング
4. ISO-8859-1

JSPページ・エンコーディングはJSPファイルのエンコーディングですが、ほとんどの場合、JSP応答エンコーディングと区別しないと予想されるため、JSPファイルのエンコーディングを<response-encoding>の<forced>に変更します。ただし、JSPファイルにBOMがある場合はUTF-8で処理します。

参考

<response-encoding><forced>をJSPページ・エンコーディングに適用する理由は、JSPのページ・タグに記述したページ・エンコーディングの値が間違っており、これが見つけにくい場合、JEUSでの設定だけで修正できるからです。たとえば、JSPファイルはEUC-KRに保存しておきながら、ページ・エンコーディングの値はUTF-8に指定した場合、<response-encoding><forced>設定をEUC-KRにすると、文字化け問題は解決されます。

JSPファイルを読み込む際の優先順位は以下のとおりです。

```
BOM > <response-encoding><forced> > JSP Page Character Encoding > JSP Response
Character Encoding > default
```

参考

JSPページの文字エンコーディングと応答文字エンコーディングについての詳細は、JSP標準を参照してください。

WebAdminの使用

WebAdminを使用してエンコーディングを設定する方法は以下のとおりです。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Basic]**メニューを選択します。([図 1.9]を参照)

2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。

3. 詳細設定のEncodingで、「Request Url Encoding」、「Request Encoding」、「Response Encoding」項目を設定し、[確認]ボタンをクリックします。

「Request Url Encoding」を除いて、jeus-web-dd.xmlでも設定可能です。

[図 1.16] Webエンジンのエンコーディング - 設定

「Default」、「Forced」、「Client Override(要求エンコーディングの場合)」のいずれかを選択して有効にした後、設定するエンコーディング名を入力します。

設定値	説明
Default	指定されたエンコーディングがない場合はデフォルト値で使します
Client Override	指定されたエンコーディングがない場合は、クライアントが送信したコンテンツ・タイプ・ヘッダーの文字セットを使します
Forced	このエンコーディングがすべての場合に強制適用されます

4. 設定が終わったら、設定内容を適用するために[Activate Changes]ボタンをクリックします。([図 1.7]を参照)

5. 設定内容が反映されると画面に結果が出力されます。「Encoding」は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

[図 1.17] Webエンジンのエンコーディング - 設定の適用結果



文字エンコーディングの設定および互換性ガイド

JEUSは、アプリケーションの開発者およびユーザーまたはサービス管理者に利便性を提供するため、XMLを使用したエンコード設定を提供してきました。しかし、これについてのポリシーを定め、実装する過程において、Request EncodingとResponse Encoding間のforced設定の意味が一致しなかったり、サーブレット標準を準拠しなかったりする問題が生じました。

JEUS 7から一貫性のあるポリシーを維持し、正すことに力を入れました。それにより、以前作成したアプリケーションの互換性問題が生じる可能性があるため、このガイドを利用してアプリケーションを修正するか互換性オプションを使用して正常な動作を維持してください。

参考

互換性オプションを使用するときは、jeus-web-dd.xmlに設定することをお勧めします。domain.xmlにサーバー別に設定すると、当該サーバーにデプロイするすべてのWebアプリケーションが影響を受けます。

● Request Url Encoding

- JEUSは、HTTP要求のAccept-Languageヘッダーを参照しません。このための互換性オプションはありません。
- <request-url-encoding>のforcedエンコーディングは、URL要求に含まれているクエリー文字列をパラメータで処理するときのみ使用します。
- サーブレットがPOST要求(コンテンツ・タイプ: application/x-www-form-urlencoded)に対して ServletRequest.getParameter()を呼び出したとき、クエリー文字列をHTTPボディに含ませて処理する

か、あるいは元の意味どおりHTTPヘッダーとして処理するかによって、<request-url-encoding>のforcedエンコーディングの使用有無を決める必要があります。

クエリー文字列をHTTPボディと区別する場合は設定し、ボディの一部として解析する場合はクエリー文字列とボディの両方とも要求エンコーディングを使用するため、要求URLエンコーディングを設定する必要があります。ただし、GET要求の場合はURL要求に含まれているクエリー文字列のみパラメータとして処理するため、結局<request-url-encoding>のforcedエンコーディング設定が必要となります。

- クエリー文字列をパラメータとして作成する時点は、サーブレットがServletRequestのパラメータAPIを初めて呼び出したときです。

● Request Encoding

- クエリー文字列、クッキー、要求ボディに適用されます。
- クエリー文字列の場合、<request-url-encoding><forced>を設定したときには影響を受けません。
- クッキーの場合、<cookie-policy><write-value-on-header-policy>にcharset-encoding設定が最優先されます。JEUS 6まで使用していたjeus.servlet.request.cookie.encodingおよびjeus.servlet.urldecode.cookieプロパティは、以降は提供しません。
- JEUS 7 Fix#2以前のバージョンで<request-encoding><forced>を使用した場合は、<request-encoding><client-override>に変更してください。
- JEUS 6では、request.setCharacterEncoding()の適用範囲をHTTPボディではなく、クエリー文字列およびクッキーまで拡大解釈する問題がありました。しかし、サーブレット標準では、該当するAPIの適用範囲をHTTPボディに限定しています。JEUS 7からはこの問題を修正していますが、JEUS 6の動作を維持する必要がある場合は、jeus.servlet.request.6CompatibleSetCharacterEncodingプロパティをfalseに指定してください。ただし、該当するアプリケーションのjeus-web-dd.xmlに設定することをお勧めします。
- **<request-encoding><forced>を設定しないことをお勧めします。** Webアプリケーションの開発時にclient-overrideまたはdefaultを設定すると、プログラムを作成する度にrequest.setCharacterEncoding()を呼び出す必要がなくなります。

Webアプリケーションで以下のような方式でnew String()するとき、エンコーディング値を与えてStringオブジェクトを直接作成する場合があります。

```
String param = request.getParameter();
String realParam = new String(param.getBytes("ISO-8859-1"), "EUC-KR");
```

上記の例でWebエンジンは、デフォルトのエンコーディングであるISO-8859-1で処理されるように設定して、paramオブジェクトがISO-8859-1で作成されるようにしています。realParam StringオブジェクトはWebアプリケーションがJVMを利用して作成したものであり、Webエンジンは関与していません。

- **JEUS 7 Fix#4からjeus-web.dd.xmlに<request-encoding><url-mapping>設定が追加されました。** jeus-web.dd.xmlに<request-encoding>を設定すると、domain.xmlの<request-encoding>設定を置き換えます。

- jeus-web-dd.xmlに<url-mapping>を設定した場合も、<forced>または<client-override>を同時に設定すると、<url-mapping>設定は無視されます。
- <url-mapping>のみ設定した場合は、<servlet-path>に指定された要求だけがマッピングされたエンコーディングが適用されます。<servlet-path>にマッピングされていないものは、サーブレット標準のデフォルト値であるISO-8859-1が適用されます。
- <url-mapping>設定と<default>を一緒に使用すると、<servlet-path>にマッピングされていないものは<default>設定で適用されます。
- domain.xmlに<request-url-encoding><forced>が設定されている場合は、POSTボディにのみ設定が適用されます。
- クライアントが送信したコンテンツ・タイプ・ヘッダーの文字セットより優先順位が低いです。

[例 1.4] 要求エンコーディングの設定例 : <<jeus-web-dd.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd>
  <encoding>
    <request-encoding>
      <url-mapping>
        <servlet-path>/test/test1</servlet-path>
        <encoding>EUC-KR</encoding>
      </url-mapping>
      <url-mapping>
        <servlet-path>/test/*</servlet-path>
        <encoding>UTF-8</encoding>
      </url-mapping>
      <url-mapping>
        <servlet-path>*.jsp</servlet-path>
        <encoding>EUC-KR</encoding>
      </url-mapping>
    </request-encoding>
  </encoding>
</jeus-web-dd>
```

● Response Encoding

- 応答ボディに適用されます。
- クッキーの場合、応答エンコーディングの影響を受けません。<cookie-policy><write-value-on-headerpolicy>のcharset-encoding設定が最優先されます。
- <response-encoding><forced>は、response.setCharacterEncoding()、setContentType()、setLocale()より優先順位が高いです。ただし、JEUS 6およびJEUS 7 Fix#1までは、forcedの優先順位についてのポリシーが明確ではなかったため、正しくない実装が存在していました。このような状況下で作成したアプリケーションをそのまま維持する必要がある場合は、jeus.servlet.response.

6CompatibleForcedEncodingプロパティをtrueに設定してください。ただし、該当するアプリケーションのjeus-web-dd.xmlに設定することをお勧めします。

- JEUS 6まではforcedを設定した場合でも、response.setCharacterEncoding()の優先順位が一番高いです。このような動作を維持したい場合は、jeus.servlet.response.6CompatibleSetCharacterEncodingプロパティをtrueに設定してください。ただし、該当するアプリケーションのjeus-web-dd.xmlに設定することをお勧めします。
- **<response-encoding><forced>を設定しないことをお勧めします。** Webアプリケーションの開発時にdefaultを設定すると、毎回エンコーディングする必要がなくなります。default設定で解決できない場合は、response APIでResponse Encodingを設定することをお勧めします。

● JSP

- JSP標準に準拠して、JSPにはページの文字エンコーディングと応答の文字エンコーディングの2タイプの文字エンコードが存在します。
- ページの文字エンコーディングはJSPファイルに対するエンコーディングです。すなわち、ファイル・システムでそのファイルを読み込むときに使用するエンコーディングです。これは基本的に応答の文字エンコーディングとは明確に区別されます。ファイルのエンコードがUTF-8であっても、そのJSPを実行して作成したHTTP応答はEUC-KRでエクスポートできるためです。ただし、ページの文字エンコーディングが分からない場合は、応答の文字エンコーディングを参照するようになっており、それで不十分な場合にはISO-8859-1でファイルを読み込みます。このようなページの文字エンコーディングを決定する優先順位についてはJSP標準に説明されています。以下の例のとおり、JSPファイルごとに明確に設定することが望ましいです。

[例 1.5] ページの文字エンコーディングが設定されたJSPの例の一部 : <<sample.jsp>>

```
<%@ page pageEncoding="UTF-8"%>
```

- 応答の文字エンコーディングは<page>タグのcontentType属性に記述された文字セットの値を意味します。この値が存在しない場合にはページの文字エンコーディングを使用します。Page Encodingも分からない場合、JSP標準で定めたデフォルト値はありません。これはWebコンテナの設定または動作によって決まります。

[例 1.6] 応答の文字エンコーディングが設定されたJSPの例の一部 : <<sample2.jsp>>

```
<%@ page contentType="text/html; charset=UTF-8"%>
```

- 上記の2つの値を両方とも設定することを推奨し、一括で適用したい場合にはweb.xmlに以下のような設定を追加します。

[例 1.7] Pageおよび応答の文字エンコーディングが設定されたweb.xml: <<web.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
```

```
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <page-encoding>UTF-8</page-encoding>
        <default-content-type>text/html; UTF-8</default-content-type>
    </jsp-property-group>
</jsp-config>
</web-app>
```

- <response-encoding>はHTTP応答についての設定であり、JSPページの文字エンコーディングとは明確に区別されますが、このような内容をすべて理解した上でJSPを作成する開発者は多くないと判断されます。したがって、<response-encoding><forced>はJSPページの文字エンコーディングと応答の文字エンコーディングに対して常に優先されます。

1.6.6. JSPエンジンの設定

WebエンジンにはJSPエンジンが含まれています。したがって、JSP関連の設定はWebエンジンまたはアプリケーション別に行います。詳しい内容は、「[第4章 JSPエンジン](#)」を参照してください。

1.6.7. 応答ヘッダーの設定

ユーザー任意のHTTP応答ヘッダーの名前と値をペアで定義することができます。

WebAdminを使用して応答に基本的に含まれるユーザー定義ヘッダーを設定する方法は以下のとおりです。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Basic]**メニューを選択します。([図 1.9](#)を参照)
2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。
3. **詳細設定のResponse Header**を設定して**[確認]**ボタンをクリックします。以下は、応答に2つのユーザー定義ヘッダーを設定した例です。1つ以上の応答ヘッダーを設定する場合には、各ヘッダーのヘッダー名とヘッダー値をイコール(=)で繋ぎ、行別に区分します。

[図 1.18] Webエンジンの応答ヘッダー - 設定

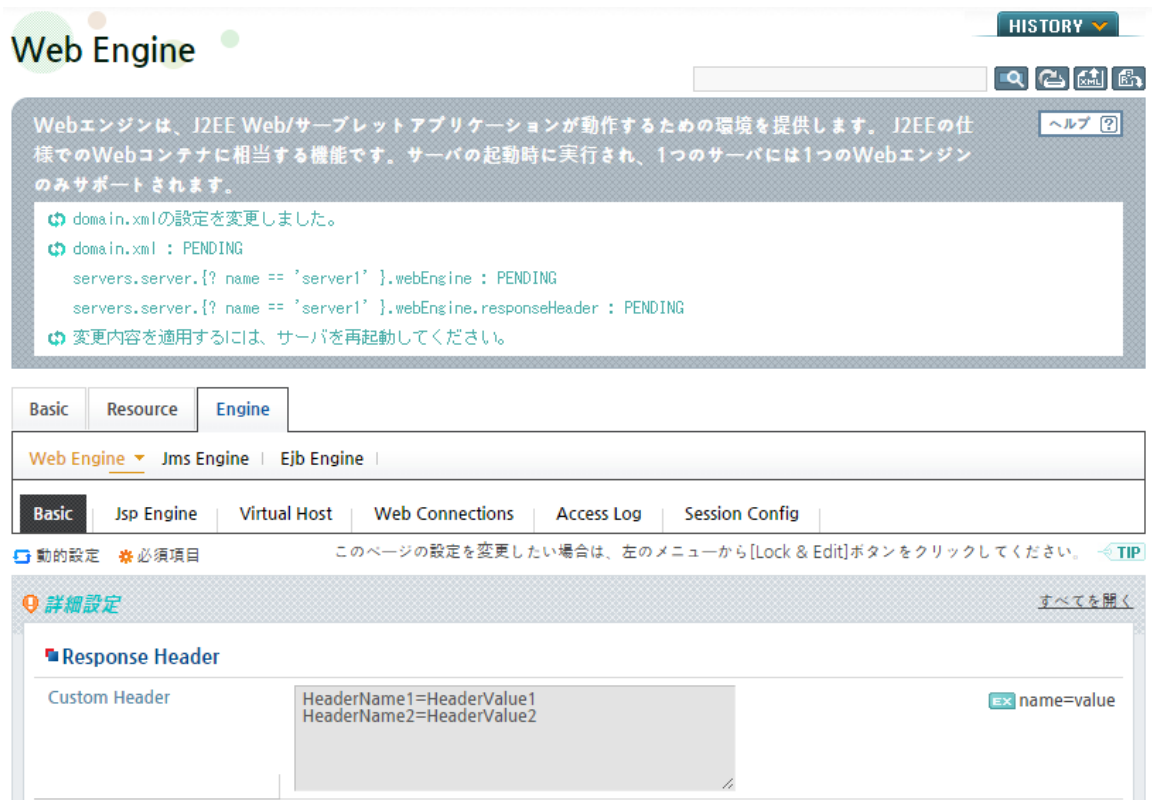
The screenshot shows the 'Web Engine' configuration interface. At the top, there's a 'Web Engine' title and a 'HISTORY' button. Below this is a descriptive text block about the Web Engine's role in J2EE. A status bar indicates '変更されました。' (Changed). The main configuration area has tabs for 'Basic', 'Resource', and 'Engine', with 'Engine' selected. Under 'Engine', there are sub-tabs: 'Web Engine', 'Jms Engine', and 'Ejb Engine'. The 'Web Engine' sub-tab is active, showing further sub-tabs: 'Basic', 'Jsp Engine', 'Virtual Host', 'Web Connections', 'Access Log', and 'Session Config'. The 'Basic' sub-tab is selected, displaying '動的設定' (Dynamic Settings) and '必須項目' (Required Items). The 'Response Header' section is expanded, showing a 'Custom Header' field with the text 'HeaderName1=HeaderValue1' and 'HeaderName2=HeaderValue2'. A 'name=value' label is visible next to the input field. Buttons for '確認' (Confirm) and '再設定' (Reset) are at the bottom right.

以下は、設定項目についての説明です。

項目	説明
Custom Header	HTTP応答メッセージに含めるためのカスタム・フィールドを定義します

4. 設定が終わったら、設定内容を適用するために[**Activate Changes**]ボタンをクリックします。([図 1.7] を参照)
5. 設定内容が反映されると画面に結果が出力されます。「**Custom Header**」は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

【図 1.19】 Webエンジンの応答ヘッダー - 設定の適用



1.6.8. クッキー・ポリシーの設定

HTTP要求ヘッダーのクッキーを読み込んだり、アプリケーションが生成した新しいクッキーをHTTP応答として送信したりするときに適用するポリシーが設定できます。クッキー・ポリシーは、Webエンジンに対してWebAdminを使用して設定可能です。また、アプリケーション別にjeus-web-dd.xmlで設定できます。

WebAdminを使用してクッキー・ポリシーを設定する方法は以下のとおりです。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。

サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Basic]**メニューを選択します。([図 1.9]を参照)

2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。
3. 詳細設定の「**Cookie Policy**」を設定し、**[確認]**ボタンをクリックします。

以下は、クッキー・ポリシーとしてUTF-8エンコーディングを使用してクッキー・エンコーディング実行を設定した例です。

[図 1.20] Webエンジンのクッキー・ポリシー - 設定

The screenshot shows the 'Web Engine' configuration page. At the top, there's a 'Web Engine' header with a 'HISTORY' dropdown. Below it, a message states: 'Webエンジンは、J2EE Web/サーブレットアプリケーションが動作するための環境を提供します。J2EEの仕様でのWebコンテナに相当する機能です。サーバの起動時に実行され、1つのサーバには1つのWebエンジンのみサポートされます。' followed by a 'ヘルプ' button. A status bar indicates '変更されました。'. The main navigation bar includes 'Basic', 'Resource', and 'Engine' tabs. Under 'Engine', there are sub-tabs: 'Web Engine', 'Jms Engine', and 'Ejb Engine'. The 'Web Engine' sub-tab is active, showing further sub-tabs: 'Basic', 'Jsp Engine', 'Virtual Host', 'Web Connections', 'Access Log', and 'Session Config'. The 'Basic' sub-tab is selected. Below the sub-tabs, there's a status bar with '動的設定' and '必須項目' icons, and '確認' and '再設定' buttons. The main content area is titled '詳細設定' and shows the 'Cookie Policy' section. Under 'Cookie Policy', there's a 'Write Value On Header Policy' section with two settings: 'Apply Url Encoding Rule' (checked) and 'Charset Encoding' (set to 'UTF-8').

項目	説明
Apply Url Encoding Rule	URLエンコーディング・ルールの適用可否を設定します
Charset Encoding	URLエンコーディング・ルールを適用するかどうかに関係なく、クッキーを応答に書き込むか解釈するときに使用する文字セット・エンコーディングです。設定しない場合は、要求エンコーディングの値に従います

4. 設定が終わったら、設定内容を適用するために[**Activate Changes**]ボタンをクリックします。([図 1.7]を参照)
5. 設定内容が反映されると画面に結果が出力されます。クッキー・ポリシーは動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

【図 1.21】 Webエンジンのクッキー・ポリシー - 設定の適用結果



1.6.9. セッションの設定

セッション・オブジェクトの共有可否、セッション・クッキーの設定、タイムアウトなど、Webエンジンのセッション管理に関するすべての事項を設定します。

セッションの設定は以下のとおり区分されます。

- Webエンジン・レベル

WebAdminを使用してセッションを設定します。詳しい内容は、『JEUS セッション管理ガイド』の「1.6.1. セッションの設定」を参照してください。

- コンテキスト・レベル

jeus-web-dd.xmlの<jeus-web-dd>下位に設定します。

Webエンジン・レベルでセッションが設定されている場合、コンテキスト・レベルでセッションを設定しなくても共通的にWebエンジンのセッション設定が適用されます。Webエンジン・レベルとコンテキスト・レベルの設定が重複された場合は、下位レベルの詳細設定に最も高い優先順位を与えます。すなわち、**コンテキスト・レベル > Webエンジン・レベル**の順で設定値が適用されます。

下位レベルで何にも設定しなかった場合は上位レベルの設定値を適用し、両レベルのどちらでも設定しなかった値はエンジン内部のデフォルト値で動作します。

参考

クラスタリングされた環境でのセッションについての詳しい内容は、『JEUS セッション管理ガイド』の「第1章 セッション・トラッキング」を参照してください。

1.6.10. ログの設定

本節では、各ログの設定方法について説明します。

- サーバー・ログの設定
- アクセス・ログの基本設定
- 仮想ホスト別のアクセス・ログの設定
- アクセス・ログのフォーマット設定
- アクセス・ログのフィルター設定
- ユーザー・ログの設定

サーバー・ログの設定

サーバー・ログ設定についての詳しい内容は、『JEUS サーバガイド』の「第8章 ロギング」を参照してください。

アクセス・ログの基本設定

Webエンジンのアクセス・ログは、WebAdminを使用して設定します。特に設定しなくてもアクセス・ログをファイルで残すように設定されています。

参考

JEUS 6 Fix#8からはアクセス・ログもログ・ローテーションを適用します。

ログ・ローテーションはJEUSが提供する機能として、JEUSの開始または運用中に設定した時間やファイル・サイズを超えた場合、以前ロギングしていたファイル名を自動的に変更し、新しく出力するログは元のログ・ファイルに継続してロギングできる機能です。

WebAdminを使用したアクセス・ログの基本設定の方法は以下のとおりです。

1. WebAdminの左側のメニューから[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[**Engine**] > [**Web Engine**] > [**Access Log**]メニューを選択します。

[図 1.22] Webエンジンのアクセス・ログ- 設定項目へ移動

Access Log HISTORY

Webエンジンレベルのアクセスログを設定します。 ヘルプ ?

Basic Resource **Engine**

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections **Access Log** Session Config

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

Enable ☒ [デフォルト: true] アクセスログを使用するかを設定します。仮想ホスト別にアクセスローガーを設定した場合、この値をfalseに設定すると、Webエンジンのアクセスローガーが使用されます。この値をfalseに設定した場合、Webエンジンのアクセスローガーはアクセスログを残しません。

Format default [デフォルト: default] Apacheで定義した共通ログ形式(Common Log Format)を使用します。default, common, combined, debugエイリアスを使用できます。各エイリアスの形式は以下のとおりです。common = "%h %l %u %t %r%W" %>s %b", combined = "%h %l %u %t %r%W" %>s %b %W"%{Referer}i%W" %W"%{User-agent}i%W", default = common + processing time (%D), debug = default + session ID + request thread name。JEUS 6及び以前のバージョンの形式を使用したい場合は、6deprecatedに設定します。ただし、この形式は動的変更ができません。

Exclude Ext アクセスログを残さない要求ファイルの拡張子を指定します。複数を記述する場合はコンマ(,)で区切ります。

Enable Host Name Lookup ☐ [デフォルト: false] %hフォーマットについて、IPアドレスの代わりにホスト名をロギングするかを設定します。この値をtrueに設定した場合、DNSルックアップによるオーバーヘッドが発生することがあります。

詳細設定 [すべてを開く](#)

Filter Class [com,txmax,logging,filter,MyFilter](#)

Formatter Class [jeus,util,logging,SimpleFormatter](#) [com,txmax,logging,handler,MyHandler](#)

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

Handlers

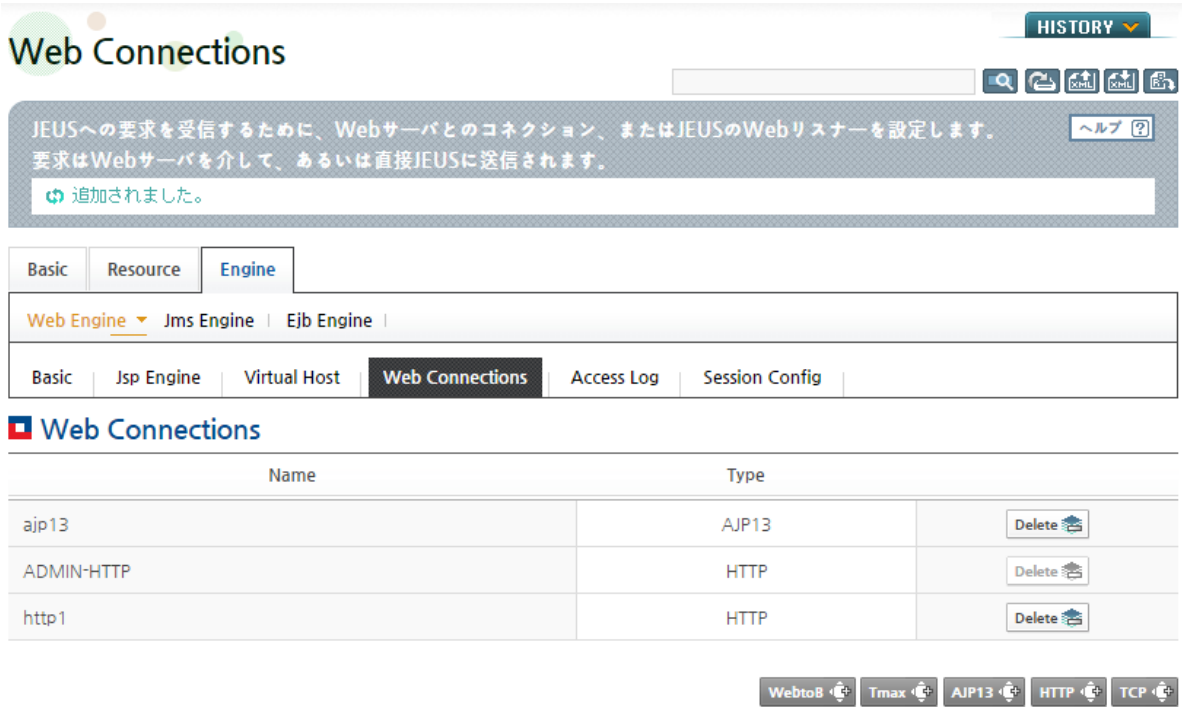
Name	Type	Level	
accessLogFileHandler	file	FINEST	Delete

2. 設定および設定変更のため、画面左側の[**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. 設定画面で必要な項目を設定します。基本設定を使用する場合は特に修正する必要はありません。主にアクセス・ログのフォーマットを変更するか、特定の拡張子のアクセスをアクセス・ログから除外する場合

に設定します。基本的に存在するファイル・ハンドラー以外に別のハンドラーを追加する場合は、設定画面の下部の「**Handlers**」項目で追加します。ハンドラーについての詳細内容は、『*JEUS サーバガイド*』の「第8章 ログイング」を参照してください。

以下は、アクセス・ログの形式を基本形式からユーザー希望の形式に変更する例です。

[図 1.23] Webエンジンのアクセス・ログ - 設定



以下は、詳細設定の各項目についての説明です。

項目	説明
Filter Class	ログーに指定するフィルター・クラスの名前を設定します
Formatter Class	ログーのハンドラーに指定するFormatterクラスの名前を設定します

4. 設定が終わったら、設定内容を適用するために[**Activate Changes**]ボタンをクリックします。([図 1.7]を参照)
5. 設定内容が反映されると画面に結果が出力されます。アクセス・ログ設定の一部項目は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

【図 1.24】 Webエンジンのアクセス・ログ - 設定の適用結果

Access Log

HISTORY

Webエンジンレベルのアクセスログを設定します。

domain.xml の設定を動的に適用しました。

domain.xml : ACTIVATED

servers.server.[? name == 'server1'].webEngine.accessLog : ACTIVATED

servers.server.[? name == 'server1'].webEngine.accessLog.excludeExt : ACTIVATED

previous value : null, edited value : jpg, txt, result value : jpg, txt

BasicResourceEngine

Web EngineJms EngineEjb Engine

BasicJsp EngineVirtual HostWeb ConnectionsAccess LogSession Config

動的設定 必須項目

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

Enable

☒

[デフォルト: true]

アクセスログを使用するかを指定します。仮想ホスト別にアクセスロガーを設定した場合、この値をfalseに設定すると、Webエンジンのアクセスロガーが使用されます。この値をfalseに設定した場合、Webエンジンのアクセスロガーはアクセスログを残しません。

Format

default

[デフォルト: default]

Apacheで定義した共通ログ形式(Common Log Format)を使用します。default、common、combined、debugエイリアスを使用できます。各エイリアスの形式は以下のとおりです。common = "%h %l %u %t %r%W" %>s %b", combined = "%h %l %u %t %r%W" %>s %b %W" %([Referer])%W" %([User-agent])%W", default = common + processing time (%D), debug = default + session ID + request thread name。JEUS 6及び以前のバージョンの形式を使用したい場合は、6deprecatedに設定します。ただし、この形式は動的変更ができません。

Exclude Ext

jpg, txt

アクセスログを残さない要求ファイルの拡張子を指定します。複数を記述する場合はコンマ(,)で区切ります。

Enable Host Name Lookup

☐

[デフォルト: false]

%hフォーマットについて、IPアドレスの代わりにホスト名をログインするかを指定します。この値をtrueに設定した場合、DNSルックアップによるオーバーヘッドが発生することがあります。

詳細設定

すべてを開く

Filter Class

com.tmax.logging.filter.MyFilter

Formatter Class

jeus.util.logging.SimpleFormattercom.tmax.logging.handler.MyHandler

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

Handlers

Name	Type	Level	
accessLogFileHandler	file	FINEST	Delete

仮想ホスト別のアクセス・ログの設定

仮想ホスト別にアクセス・ログを残すには、各仮想ホスト設定にアクセス・ログを設定します。詳しい内容は、「5.4. 仮想ホストの設定」を参照してください。

40 JEUS Webエンジンガイド

アクセス・ログのフォーマット設定

WebAdminを使用してアクセス・ログのフォーマットが設定できます。WebAdminの[**Servers**]メニューをクリックすると表示されるサーバー・リストからサーバーを選択し、[**Engine**] > [**Web Engine**] > [**Access Log**]メニューに移動して[**Access Log**]画面の「**Format**」項目を設定します。([図 1.23]を参照)

フォーマットでは基本的に「%」記号を使ってデータを表しており、任意の文字列も使用できます。JEUS 7からはCommon Log Format(以下、CLF)に従います。

参考

1. CLFについての詳細内容は、<http://httpd.apache.org/docs/2.4/logs.html>を参照してください。
 2. コンソール・ツールを使用したフォーマット設定の変更は、『*JEUS リファレンスガイド*』の「4.2.8.18. modify-web-engine-configuration」を参照してください。
-

Webエンジンは以下のようなフォーマット・エイリアスを提供します。

- default

JEUSがデフォルトで提供するフォーマットです。CLFで最も多く使用するcommonフォーマットに処理時間(%D)を追加したフォーマットです。性能上の理由により、常に「-」が表示される%Iとセッションから値を読み込む%uを削除しました。

```
%h %t \"%r\" %s %b %D
```

- common

CLFで最も多く使用するフォーマットです。以下のフォーマットをエイリアスします。

```
%h %l %u %t \"%r\" %s %b
```

- combined

HTTPヘッダーでRefererとUser-agentを記録するフォーマットです。以下のフォーマットをエイリアスします。

```
%h %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-agent}i\"
```

- debug

defaultフォーマットに%S(セッションID)と%I(要求処理スレッド名)を追加したフォーマットです。

```
%h %l %u %t \"%r\" %s %b %D %S \"%I\"
```

上記のフォーマット・エイリアスを使用して、以下のような形式でフォーマットを指定することができます。

```
default %S
```

この場合、Webエンジンでcommonを置換して以下のように登録します。

```
%h %l %u %t \"%r\" %s %b %D %S
```

フォーマット設定はサービス中にも変更可能であり、その変更内容が適用されます。したがって、現在提供しているサービスに問題がある場合は、debugフォーマットを設定してセッションID、応答処理スレッド名をアクセス・ログに残すことができます。

アクセス・ログのプログラミング方式のフィルター設定

アクセス・ログのフィルター機能は、アクセス・ログの量が多すぎる場合、特定の形式または条件を満たす内容のみログとして記録するために提供されます。

参考

1. <exclude-ext>を使用して特定の拡張子で終わる要求に対するフィルタリングをサポートしています。
 2. JEUS 7からは、jeus.servlet.utilパッケージからjeus.servlet.loggerパッケージに変更されました。
-

JEUSでは、jeus.servlet.logger.AccessLoggerFilterインターフェースおよびjeus.servlet.logger.AbstractAccessLoggerFilter抽象クラスを提供しています。

**[例 1.8] jeus.servlet.logger.AccessLoggerFilterインターフェースの詳細内容 :
<<AccessLoggerFilter.java>>**

```
package jeus.servlet.logger;

import java.util.logging.Filter;
import java.util.logging.LogRecord;

/**
 * アクセス・ロガーの内容をフィルタリングするために必要な情報を提供するインターフェース
 * {@link Filter}を継承したフィルター・インターフェースとして、付加的に提供されるインターフェースを利用して
 * {@link Filter#isLoggable(java.util.logging.LogRecord)}内で様々なフィルタリング・ポリシーが指定できるよ
 * うにガイドします。
 */
public interface AccessLoggerFilter extends Filter {

    /**
     * サーバーに接続したリモート・クライアントのアドレス情報を返します。
     *
     * @param record a LogRecord
     * @return remote clientのアドレス。正しい値が取得できない場合はnullを返します。
     */
    public String getRemoteAddr( LogRecord record );

    /**
     * 要求のメソッドを返します。 ex) GET, POST, PUT, etc...
     *
     * @param record a LogRecord
     * @return request method。正しい値が取得できない場合はnullを返します。
     */
    public String getMethod( LogRecord record );

    /**
     * 要求のuriを返します。
     *
     * @param record a LogRecord
     * @return request uri。正しい値が取得できない場合はnullを返します。
     */
    public String getRequestURI( LogRecord record );

    /**
     * 応答のステータス値を返します。 ex) 200、404
     *
     * @param record a LogRecord
     * @return status.
     */
    public int getStatus( LogRecord record );

    /**
     * 要求別の処理時間をms単位で返します。
     *
     * @param record a LogRecord
     * @return processing time。正しい値が取得できない場合は-1を返します。
     */
    public long getProcessingTimeMillis( LogRecord record );
}
```

ユーザーがアクセス・ログの特定パターンに対してフィルタリングする場合、上記のAccessLoggerFilterインターフェースおよびAbstractAccessLoggerFilter抽象クラスを使用して簡単に実装、適用することができます。

ユーザーのフィルター・クラスはjeus.servlet.logger.AbstractAccessLoggerFilterを継承し、java.util.logging.FilterのisLoggable()メソッドを実装する必要があります。isLoggable()メソッドを実装するときは、上記のjeus.servlet.logger.AccessLoggerFilterのAPIからユーザーが必要な情報を適切に選択して使用および実装します。

以下は、要求拡張子が.gifの場合アクセス・ログに記録しないフィルター・クラスを定義した例です。

[例 1.9] フィルター・クラスの定義例 : <<SimpleAccessLoggerFilter>>

```
package sample;

import jeus.servlet.logger.AbstractAccessLoggerFilter;
import java.util.logging.*;

public class SimpleAccessLoggerFilter extends AbstractAccessLoggerFilter {
    public boolean isLoggable(LogRecord record) {
        String requestURI = getRequestURI(record);
        return requestURI != null && !requestURI.endsWith(".gif");
    }
}
```

- sample.SimpleAccessLoggerFilterはユーザーが定義したクラスです。jeus.servlet.logger.AbstractAccessLoggerFilterを拡張します。
- java.util.logging.Filter#isLoggable()メソッドを実装しており、クライアントの要求URIを取得するためにjeus.servlet.util.AccessLoggerFilter#getRequestURI() APIを使用します。

クラス定義が完了したら当該クラスをコンパイルします。その後、JARファイル形式でJEUS_HOME/domains/DOMAIN_HOME/lib/applicationディレクトリーに格納し、Webエンジンのアクセス・ログ設定にフィルターを設定します。アクセス・ログの設定方法は、[「アクセス・ログの基本設定」](#)を参照してください。

ユーザー・ログの設定

何にも設定しない場合は、サーバー・ログ・ファイル(*JeusServer.log*)にログが残ります。別途のログ・ファイルが残りたい場合はユーザー・ログを設定します。ユーザー・ログの基本位置およびファイルは、[「1.5. ディレクトリー構造」](#)を参照してください。

ユーザー・ログはサーバー・ログと同様、JEUSサーバー・レベルで登録することができ、Webアプリケーションのjeus-web-dd.xmlに登録することもできます。さらに、Webアプリケーション別または特定のログ・ファイルにまとめて残すこともできます。

WebAdminを使用したユーザーログの設定方法は以下のとおりです。

1. WebAdminの左側のメニューから[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[**Basic**] > [**User Logging**]メニューを選択します。
2. 設定および設定変更のため、画面左側の[**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. webappという名前のWebアプリケーションだけのログを生成するため、「**Name**」項目を以下のとおり設定します。webappの名前なしで「jeus.systemuser」を入力すると、すべてのWebアプリケーションにユーザー・ログが適用されます。

詳細設定の各項目についての設定は、「[アクセス・ログの基本設定](#)」を参照してください。以外に「**Level**」または**詳細設定**は必要に応じて設定します。

[図 1.25] ユーザー・ログ - 設定

The screenshot displays the 'User Logging' configuration interface. At the top, there's a 'HISTORY' dropdown and a search bar. Below this, a description states: 'jeus.util.UserLoggerクラスを使って作成したログの出力方法を設定します。' (Set the output method of the log created using the jeus.util.UserLogger class). The main configuration area has three tabs: 'Basic', 'Resource', and 'Engine'. The 'Basic' tab is selected, showing a breadcrumb trail: 'Basic Info | Res Ref | Naming Server | System Thread Pool | System Logging | User Logging | Tm Config'. Below the breadcrumb, there are buttons for '動的設定' (Dynamic Settings), '必須項目' (Required Items), '確認' (Confirm), '再設定' (Reset), and '削除' (Delete). The 'Name' field is marked with a red asterisk and contains 'jeus.systemuser.webuser.app1'. A tooltip explains that this field is used to specify the log name when applying settings. The 'Level' field is a dropdown menu set to 'INFO', with a tooltip explaining its meaning. Below these fields is a '詳細設定' (Detailed Settings) section with a 'すべてを開く' (Open all) link. It contains three rows: 'Use Parent Handlers' (checked), 'Filter Class' (with an example 'com.tmax.logging.filter.MyFilter'), and 'Formatter Class' (with an example 'com.tmax.logging.handler.MyHandler'). At the bottom, there's a 'Handlers' section with a table header: 'Name', 'Type', and 'Level'. The table body is empty, and a message states: '該当する内容が存在しません。' (No content exists for this category).

4. ユーザー・ログを設定した後、**[確認]**ボタンをクリックすると、**Handlers**領域の下部にハンドラーが追加できるボタンが追加されます。**[FILE HANDLER]**、**[SMTP HANDLER]**、**[SOCKET HANDLER]**、**[USER HANDLER]**から追加するハンドラーの種類に適するボタンをクリックしてユーザー・ログのハンドラーを追加します。各ハンドラーの設定方法は、『JEUS サーバガイド』の「第8章 ログイン」を参照してください。

本例では、ファイル・ハンドラーを追加するために**[FILE HANDLER]**ボタンをクリックします。

[図 1.26] ユーザー・ログ - ハンドラーの追加

User Logging

jeus.util.UserLoggerクラスを使って作成したログの出力方法を設定します。

変更されました。

Basic Resource Engine

Basic Info | Res Ref | Naming Server | System Thread Pool | System Logging | User Logging | Tm Config

動的設定 * 必須項目

確認 再設定 削除

Name * jeus.systemuser.webuser.app1
ローガーに対して設定を適用するとき、該当するローガー名を指定します。ローガー名を確認したい場合は、ローガーページを参照してください。

Level [デフォルト: INFO] ローガーのレベルを設定します。各レベルの意味については、Java SE logging APIの「Level Class Documentation」を参照してください。

詳細設定 すべてを開く

Use Parent Handlers ☒

Filter Class EX com.tmax.logging.filter.MyFilter

Formatter Class EX com.tmax.logging.handler.MyHandler

確認 再設定 削除

Handlers

Name	Type	Level
該当する内容が存在しません。		

File Handler SMTP Handler Socket Handler User Handler

5. ユーザー・ログのハンドラーを追加するため、[File Handler]画面で各項目を設定した後、[確認]ボタンをクリックします。

[図 1.27] ユーザー・ログ - ハンドラーの設定

File Handler

HISTORY

ログメッセージをファイルに出力する場合に使用するハンドラです。

ヘルプ

BasicResourceEngine

Basic Info | Res Ref | Naming Server | System Thread Pool | System Logging | User Logging | Tm Config

動的設定 必須項目

確認 再設定

Name	<input type="text" value="fileHandler"/> EX handler1 ハンドラの名前を設定します。この名前は1つのロガー内で一意である必要があります。設定された名前は管理ツール(WebAdminのツール)などでハンドラを指すときに使用します。
File Name	<input type="text" value="/home/jeus/logs/mylog.log"/> EX /home/jeus/logs/mylog.log ハンドラが使用するファイル名を設定します。設定していない場合は、各ロガーのデフォルトファイル名が使用されます。各デフォルトファイル名については、「JEUS サーバガイド」を参照してください。
Level	<div>FINEST</div> <div>【デフォルト: FINEST】ハンドラのレベルを設定します。ロガーが送信したメッセージのレベルがハンドラに指定されているレベルに該当する場合のみ出力されます。</div>
Enable Rotation	<div><input checked="" type="checkbox"/></div> EX true 【デフォルト: true】ハンドラが使用するファイルがログファイルローテーション機能を使用するか否かを設定します。特に設定していない場合はtrueに設定され、ファイルにロギングする際にローテーション機能を使用します。
Rotation Count	<input type="text" value="10"/> EX 10 ハンドラが使用するファイルがログファイルローテーション機能を使用するとき、バックアップするファイル数を設定します。設定せずにファイルサイズでローテーションする場合、99999個まで蓄積されます。日付または時間でローテーションする場合は、ローテーションされたファイルは継続して蓄積されます。
<input checked="" type="radio"/> Valid Day	<input type="text" value="d"/> EX 1 【デフォルト: 1】ハンドラが使用するファイルをValid Dayに設定した期間のみ使用し、継続して更新されます。この設定は日数単位でファイルを更新するときに使います。この場合、ハンドラが使用するファイル名の最後にファイルが使用された日付が自動で付与されます。
<input type="radio"/> Valid Hour	<input type="text" value="h"/> EX 3 ハンドラが使用するファイルをValid Hourに設定した時間のみ使用し、継続して更新されます。この設定は時間単位でファイルを更新するときに使います。この場合、ハンドラが使用するファイル名の最後にファイルが使用された日付と時間が自動で付与されます。
<input type="radio"/> Valid Size	<input type="text" value="kbyte"/> EX 1024 ハンドラが使用するファイルがValid Sizeに設定したサイズより小さい場合のみ使用し、継続して更新されます。この設定はサイズ単位でファイルを更新するときに使います。この場合、ハンドラが使用するファイル名の最後に順次的にインデックスが付与されます。

6. ハンドラーの追加が完了したら、以下のように追加されたハンドラーが表示されます。

[図 1.28] ユーザー・ログ - ハンドラー追加の確認

User Logging HISTORY

jeus.util.UserLoggerクラスを使って作成したログの出力方法を設定します。 ヘルプ ?

追加されました。

Basic Resource Engine

Basic Info | Res Ref | Naming Server | System Thread Pool | System Logging | **User Logging** | Tm Config

動的設定 * 必須項目 確認 再設定 削除

Name * jeus.systemuser.webuser.app1
ローガーに対して設定を適用するとき、該当するローガー名を指定します。ローガー名を確認したい場合は、ローガーページを参照してください。

Level [デフォルト: INFO] ローガーのレベルを設定します。各レベルの意味については、Java SE logging APIの"Level Class Documentation"を参照してください。

詳細設定 [すべてを開く](#)

Use Parent Handlers ☒

Filter Class [Ex](#) com.tmax.logging.filter.MyFilter

Formatter Class [Ex](#) com.tmax.logging.handler.MyHandler

確認 再設定 削除

Handlers

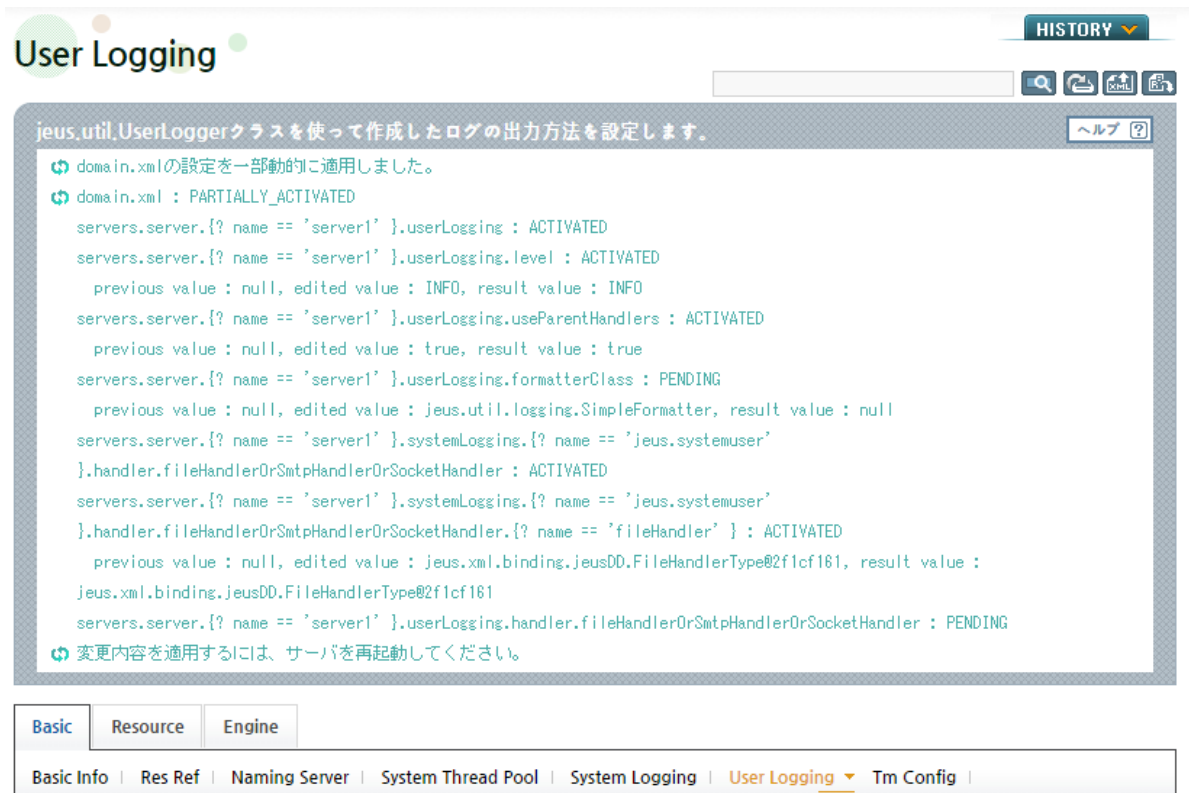
Name	Type	Level	
fileHandler	file	FINEST	Delete

[File Handler](#) [SMTP Handler](#) [Socket Handler](#) [User Handler](#)

7. 設定が終わったら、設定内容を適用するために[Activate Changes]ボタンをクリックします。([図 1.7] を参照)

8. 設定内容が反映されると画面に結果が出力されます。ユーザー・ログ設定の一部項目は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

[図 1.29] ユーザー・ログ - ハンドラー追加の適用結果



1.6.11. 非同期サーブレットのタイムアウト処理の設定

サーブレット3.0から追加された非同期サーブレットのタイムアウト処理のために必要なスレッド数を設定します。

WebAdminを使用した非同期サーブレットのタイムアウト処理の設定方法は以下のとおりです。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Basic]**メニューを選択します。([図 1.9]を参照)
2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。

3. [Web Engine]画面で「Async Timeout Min Threads」項目を設定した後、[確認]ボタンをクリックします。

[図 1.30] 非同期サーブレットのタイムアウト処理 - Async Timeout Min Threads設定

The screenshot shows the 'Web Engine' configuration interface. At the top, there's a 'Web Engine' header with a 'HISTORY' button. Below it, a message states: 'Webエンジンは、J2EE Web/サーブレットアプリケーションが動作するための環境を提供します。J2EEの仕様でのWebコンテナに相当する機能です。サーバの起動時に実行され、1つのサーバには1つのWebエンジンのみサポートされます。' followed by a 'ヘルプ (?)' button and a status bar '変更されました。'. The main configuration area has tabs for 'Basic', 'Resource', and 'Engine'. Under 'Engine', there are sub-tabs: 'Web Engine' (selected), 'Jms Engine', and 'Ejb Engine'. Below these are more specific tabs: 'Basic', 'Jsp Engine', 'Virtual Host', 'Web Connections', 'Access Log', and 'Session Config'. A status bar indicates '動的設定' and '必須項目', with '確認' and '再設定' buttons. The configuration items are listed in a table:

Attach Stacktrace On Error	<input checked="" type="checkbox"/> [デフォルト: false] JEUSが返すエラーページにスタックトレースを添付するか否かを設定します。この機能は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。
Async Timeout Min Threads	20 [デフォルト: 1] Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。
Properties	<div></div> Webエンジンに適用される属性を設定します。
Default Error Page	/jeus/user1/error.html Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはダイレクトされず、HTTP応答本文の内容として使用されます。

4. 設定が終わったら、設定内容を適用するために[Activate Changes]ボタンをクリックします。([図 1.7]を参照)

5. 設定内容が反映されると画面に結果が出力されます。非同期サーブレットのタイムアウト処理は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

[図 1.31] 非同期サーブレットのタイムアウト処理 - Async Timeout Min Threads設定の適用結果

Web Engine

HISTORY

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

domain.xmlの設定を変更しました。
 domain.xml : PENDING
 servers.server.[? name == 'server1'].webEngine : PENDING
 servers.server.[? name == 'server1'].webEngine.asyncTimeoutMinThreads : PENDING
 previous value : 1, edited value : 20, result value : 1
 変更内容を適用するには、サーバーを再起動してください。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

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

Attach Stacktrace On Error	<input checked="" type="checkbox"/> [デフォルト: false] JEUSが返すエラーページにスタックトレースを添付するか否かを設定します。この機能は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。
Async Timeout Min Threads	20 [デフォルト: 1] Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。
Properties	<div>key=value</div> Webエンジンに適用される属性を設定します。
Default Error Page	/jeus/user1/error.html Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはリダイレクトされず、HTTP応答本文の内容として使用されます。

1.6.12. Webエンジン・レベルのプロパティ設定

WebAdminを使用してJEUSで定義したプロパティを設定することができます。Webエンジン・プロパティについての詳細内容は、『JEUS リファレンスガイド』の「1.6. Webエンジンのプロパティ」を参照してください。

WebAdminを使用したプロパティの設定方法は以下のとおりです。

1. WebAdminの左側のメニューから[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[**Engine**] > [**Web Engine**] > [**Basic**]メニューを選択します。([図 1.9]を参照)
2. 設定および設定変更のため、画面左側の[**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. 以下のように「**Properties**」項目を設定します。プロパティ名と値をイコール(=)で繋ぎ、改行して名前と値をペアで入力します。入力が終わったら[**確認**]ボタンをクリックします。

[図 1.32] Webエンジン・プロパティ - 設定

Web Engine

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

変更されました。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

動的設定 必須項目

Attach Stacktrace On Error ☒

Async Timeout Min Threads 20

Properties

jeus.servlet.request.enableDns=false
jeus.servlet.loader.jspcount=10

key=value

Webエンジンに適用される属性を設定します。

Default Error Page

/jeus/user1/error.html

Webアプリケーションに別途のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはリダイレクトされず、HTTP応答本文の内容として使用されます。

参考

WebAdminを使用したプロパティ設定は、「**Properties**」項目で設定することをお勧めしますが、[**Servers**] > [**Basic**] > [**Basic Info**]項目の「**Jvm Option**」項目でも設定可能です。

4. 設定が終わったら、設定内容を適用するために[**Activate Changes**]ボタンをクリックします。([図 1.7]を参照)

5. 設定内容が反映されると画面に結果が出力されます。プロパティ設定は動的設定ではないため、内容を適用するにはサーバーを再起動する必要があります。

【図 1.33】 Webエンジン・プロパティ - 設定の適用結果

Web Engine

HISTORY

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

ヘルプ ?

- domain.xmlの設定を変更しました。
- domain.xml : PENDING
 - servers.server.{? name == 'server1' }.webEngine : PENDING
 - servers.server.{? name == 'server1' }.webEngine.properties : PENDING
- 変更内容を適用するには、サーバーを再起動してください。

Basic | Resource | **Engine**

Web Engine | Jms Engine | Ejb Engine

Basic | Jsp Engine | Virtual Host | Web Connections | Access Log | Session Config

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

Attach Stacktrace On Error	<input checked="" type="checkbox"/> [デフォルト: false] JEUSが返すエラーページにスタックトレースを添付するか否かを設定します。この機能は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。
Async Timeout Min Threads	20 [デフォルト: 1] Servlet 3.0の非同期サーブレットを使用する場合、タイムアウト処理を行うスレッドプールの最小数を指定します。0を指定した場合、タイムアウトが正常に動作しないことがあるため、1以上の値を設定してください。
Properties	<div>jeus.servlet.request.enableDns=false jeus.servlet.loader.jspcount=10</div> <div>key=value</div> <p>Webエンジンに適用される属性を設定します。</p>
Default Error Page	<div>/jeus/user1/error.html</div> <p>Webアプリケーションに別添のエラーページを設定していない場合に使用するエラーページを指定します。静的ページ(HTML、HTML)のみ設定でき、絶対パスで指定します。このページは転送またはリダイレクトされず、HTTP応答本文の内容として使用されます。</p>

1.6.13. Web攻撃に対応する設定

JEUSの要求処理に負荷をかけるため、一定サイズ以上の要求を大量に送信する悪意的な攻撃を受けた場合、正常な要求の応答が遅延されるか、処理できない問題が発生します。

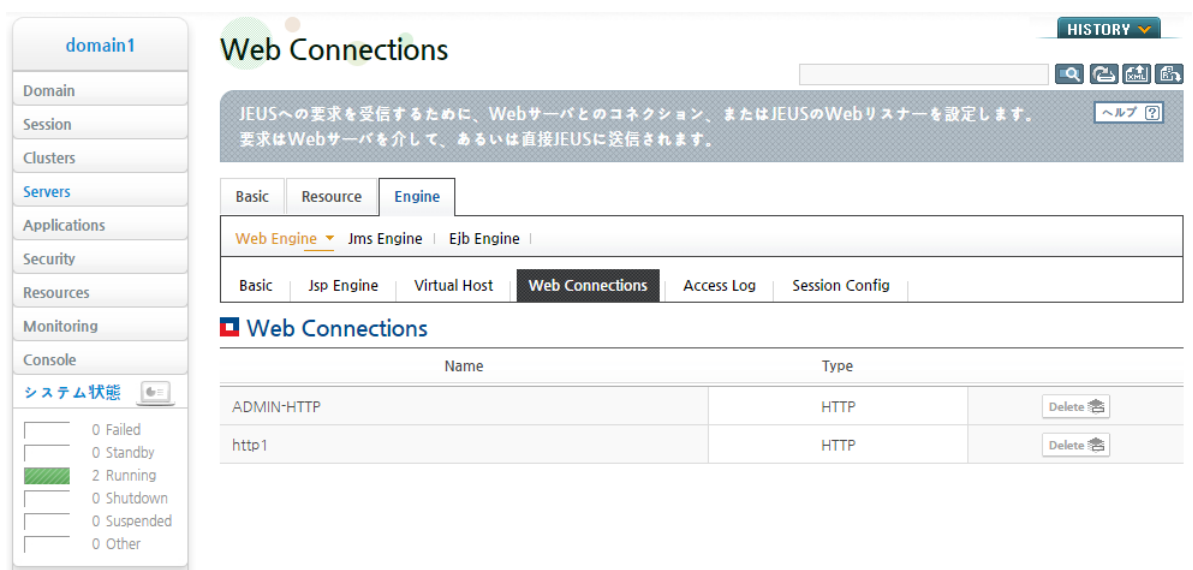
このような問題を防ぐため、JEUS管理者がJEUSサーバーを基準にして悪意的な要求に対する基準を設定し、これらの要求処理によるサーバーの負担を減らすことができます。同設定は基本的にサーバーのWebエンジン内部に設定され、Webエンジンの要求を受けるリスナーやコネクター別に設定します。すなわち、HTTP要求を受信するHTTPリスナー、WebtoBコネクター、AJP13コネクター別に設定します。リスナーおよびコネクター別の設定については、「2.3.1. リスナーの共通設定」を参照してください。

制限するWeb攻撃に対し、以下の設定を適切に組み合わせて設定するとWeb攻撃に効果的に対応できます。

WebAdminを使用したWeb攻撃に対応する設定方法は以下のとおりです。

1. WebAdminの左側のメニューから[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。
サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[**Engine**] > [**Web Engine**] > [**Web Connections**]メニューを選択すると、Webコネクション・リストの照会画面へ移動します。Webコネクション・リストからWebコネクションを選択します。

[図 1.34] Web攻撃への対応 - メニュー移動



2. 設定および設定変更のため、画面左側の[**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. 以下のとおり、**詳細設定**の**Http Listener**領域から各項目を設定します。画面に表示された項目を設定し、リスナー別にWeb攻撃に対応することができます。

各設定項目についての詳細内容は、「[2.3.1. リスナーの共通設定](#)」を参照してください。

[図 1.35] Web攻撃への対応 - 詳細設定

1.6.14. 特定URLパターンのHTTP要求を制限

特定URLパターンのHTTP要求を制限するため、HTTPクライアントが以下のような要求を送信したと仮定します。

```
GET /examples/%2e%2e%2fdb.txt HTTP/1.1
```

Webコンテナは、まず要求URIをURLデコードします。

```
/examples/../../db.txt
```

これにより、実際のHTTPサービスとは関係のないファイルへのアクセスが可能になります。主に、悪意のある目的を持つクライアントが、このような要求URIを送信します。

Webコンテナですべての悪意のある要求パターンを把握することはできないため、ユーザーがそのようなパターンを直接制限できるような設定を提供しています。HTTP要求の場合にのみ動作し、クエリー文字列を除いたURIに対して特定の文字列をマッチして無条件で「404 Not Found」として処理します。

参考

JEUS 6およびJEUS 7 Fix#1まではプロパティ・ファイルを使用しましたが、JEUS 7 Fix#2からはdomain.xmlに設定します。

WebAdminを使用した設定方法は以下のとおりです。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面にて、[Engine] > [Web Engine]メニューを選択すると、Basic設定の照会画面が表示されます。この画面で詳細設定へスクロールします。
2. 設定および設定変更のために、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. 以下のようにBlocked Url Patternsを設定します。

[図 1.36] URLパターンを利用したHTTP要求制限機能 - 詳細設定



「Encoded Pattern」はHTTPクライアントが送信したURIに対し、設定された文字列が含まれているのかをチェックします。クライアントが送信したURIはWebコンテナがURLデコードした後に使用しますが、そのとき「Decoded Pattern」が含まれているのかをチェックします。「Encoded Pattern」を1つでも設定すると、上述した基本パターンは処理しません。「Decoded Pattern」も同様です。

参考

何にも設定しない場合のWebコンテナは、Encoded Patternの場合は%00、%23、%2e、%2f、%5cを、Decoded Patternの場合は#に対して404応答を返します。パターンは大文字と小文字を区別しません。常に小文字に変換して処理します。

1.7. Webエンジンのチューニング

Webエンジンの最適化された性能のため、以下の事項を考慮します。

- <output-buffer-size>を適切なサイズに指定します。WebtoBと一緒に使用する場合は、<webtob-connector>に<send-buffer-size>と<receive-buffer-size>を適切に指定します。
- <check-included-jspfile>は、インクルードされたJSPが変更されなければ、設定をfalseに維持します。これは、インクルードされたJSPファイルに対する変更チェックが省略されるため、性能向上につながります。

- JSPファイルが変更されなければ、`jeus.servlet.jsp.reload`プロパティを`false`に設定します。それにより、JSPを呼び出すたびにファイル・システムのメタデータを照会したりしません。

第2章 Webコネクションの管理

本書では、Webエンジンで提供するリスナーとコネクタの管理および設定方法について説明します。

2.1. 概要

JEUSでは、WebリスナーとWebコネクタをWebコネクションといいます。

Webエンジンでは、WebtoBおよび別のWebサーバーとのコネクション、HTTP/TCPクライアントとの直接的なコネクション、またはTmaxとのコネクションを提供します。WebサーバーはクライアントのHTTP要求を受け、条件を満たす場合はWebエンジンに渡します。

代表的なWebサーバーとしてWebtoBとApacheを使用しており、コネクションのために以下のようなコネクタを提供しています。

- **WebtoBコネクタ**

WebtoBはコネクションを生成する際、JEUSがクライアントの役割をするためWebtoBコネクタを提供します。

- **AJPリスナー**

ApacheはJEUSがサーバーの役割をするためAJPリスナーを提供します。AJPの場合、IIS、SunOne(Iplanet)のような他ベンダーの商用Webサーバーでもサポートするので、AJPリスナーを介してWebサーバーと接続することができます。

Webエンジンはクライアントと直接コネクションを確立し、管理のためにクライアント別に以下のリスナーおよびコネクタを提供しています。

- **HTTPリスナー**

HTTPクライアントとの直接的なコネクション管理のために提供されます。**Async ServletおよびServlet 3.1 NIO、Websocketの機能を使用するためには、HTTPリスナーを使用する必要があります。**

- **TCPリスナー**

TCPクライアントとのコネクション管理のために提供されます。

- **Tmaxコネクタ**

Tmaxとの連動のためのコネクタです。WebtoBコネクタと同様、コネクションを確立するときにはJEUSがクライアントの役割をします。

リスナーはJEUSサーバーの設定を基にして動作します。したがって、セキュリティー(SSL)リスナーを使用するには、JEUSサーバーに該当する設定をし、この設定をそれぞれのWeb関連リスナーが使用するよう設定します。

参考

JEUSサーバー関連の詳細設定については、『*JEUS サーバガイド*』の「2.3.2. リスナーの設定」を参照してください。

2.2. 構成要素

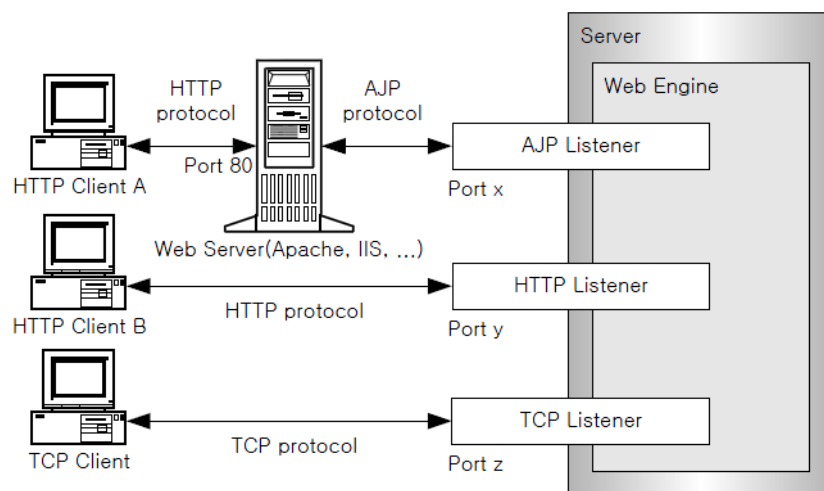
本節では、Webエンジンで提供するリスナーおよびコネクタと使用されるスレッド・プールについて説明します。

2.2.1. リスナー

リスナーは、AJPプロトコルに従うWebサーバー、HTTP/TCPクライアントがアクセスできるWebエンジンのチャンネルです。

Webエンジンのリスナー、クライアント、プロトコルのコネクションは以下のとおりです。

【図 2.1】 Webエンジンのリスナー



以下は、各リスナーについての説明です。

- AJPリスナー

WebtoB以外の別のWebサーバー(Apache、IIS、SunOne(Iplanet)など)を使用する場合にも、JEUS Webアプリケーションとの相互連動ができるようにするプロトコルです。mod_jk moduleを通じてサポートしており、AJP 1.3プロトコルを使用します。AJPリスナーはSSLをサポートします。

AJPリスナーについての詳細内容は、「[2.3.2. AJPリスナーの設定](#)」と「[2.4. 負荷分散のためのWebサーバーの設定](#)」を参照し、AJPリスナーで使用するSSLサーバー・リスナーの設定は、『JEUS サーバガイド』を参照してください。

- HTTPリスナー

HTTP要求をWebエンジンが直接受けるときに使用します。HTTPリスナーはSSLをサポートします。HTTPリスナー設定の詳細内容は、「[2.3.3. HTTPリスナーの設定](#)」を参照してください。

HTTPリスナーで使用するSSLサーバー・リスナーの設定は、『JEUS サーバガイド』を参照してください。

- TCPリスナー

HTTPプロトコルではなく、カスタム・プロトコルで動作するクライアントのリスナーです。

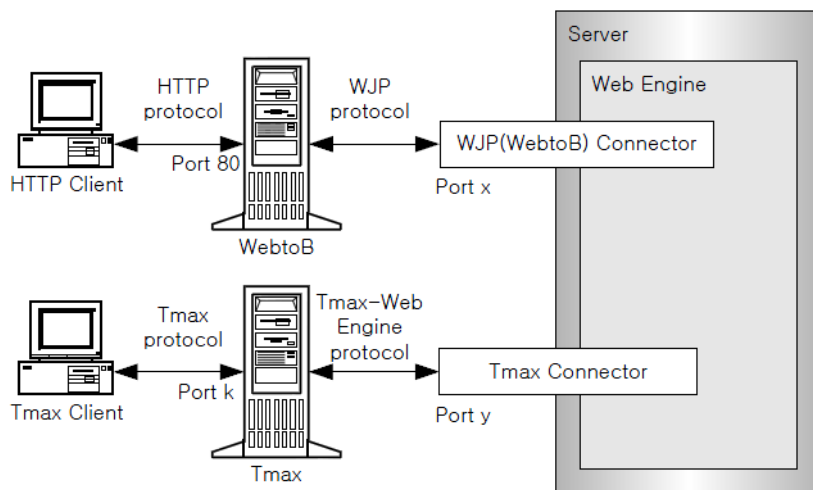
TCPリスナーについての詳細内容は、「[2.3.4. TCPリスナーの設定](#)」と「[2.5. TCPリスナーの使用](#)」を参照してください。

2.2.2. コネクター

コネクターは、WebエンジンでWebtoBおよびTmaxに接続するためのチャンネルです。

Webエンジンのコネクター、クライアント、プロトコルの接続は以下のとおりです。

[図 2.2] Webエンジンのコネクター



以下は、各コネクターについての説明です。

- WJP(WebtoB)コネクター

WebtoBはTmaxSoftで提供するWebサーバーです。JEUSに組み込まれているものを使用するか、別途製品を購入してJEUSと一緒に使用することもできます。

WJPはWebtoB-JEUSプロトコルを意味します。WebtoBコネクタは、コネクションの生成時点においてはJEUSがクライアントの役割をするため、直接WebtoBに接続する特性があります。このような特性によって、ファイアウォールの外にWebtoBサーバーが存在する場合は、特にファイアウォールを設定せずに接続することができます。これは、ファイアウォールは主に外部からの接続を抑制し、内部からの接続は可能にする属性を利用したものです。WebtoBコネクタについての詳細内容は、「[2.4. 負荷分散のためのWebサーバーの設定](#)」を参照してください。

- Tmaxコネクタ

Tmaxは、分散環境でXAトランザクションを管理するシステム・ソフトウェアです。

TmaxコネクタもWebtoBコネクタと同様、JEUSがクライアントの役割をします。Tmaxコネクタは、JEUSとTmax間の情報をやり取りするか、HTTP要求をTmaxのゲートウェイを介して受信するなど、通信チャンネルを一元化する用途で使用できます。Tmaxコネクタの詳細内容は、「[2.3.6. Tmaxコネクタの設定](#)」を参照してください。

参考

JEUSがクライアントの役割をするのは、接続してコネクションを確立するときだけです。実際に外部クライアントの要求はWebtoB、Tmaxから渡されるので、JEUS内部的にWebtoBまたはTmaxに要求を送信する場合はありません。すなわち、実際のサービス中にはJEUSがサーバーの役割をします。

2.2.3. ワーカー・スレッド・プール

リスナーまたはコネクタにはクライアントからの要求を処理するためのワーカー・スレッド・プールが存在しています。これは、ワーカー・スレッドを管理するオブジェクトです。

リスナーは要求が受信されると、その要求を処理するためのワーカー・スレッドが割り当てられます。コネクタの場合は、WebtoBまたはTmaxと1:1でコネクションを確立し、それにワーカー・スレッドが割り当てられます。ワーカー・スレッド・プールの最小値/最大値はサービスの処理性能に大きな影響を与えるため、リスナーまたはコネクタを設定する際にはワーカー・スレッド・プールに注意して設定する必要があります。

参考

JEUSは各スレッド・プールは自身の状態を直接管理し、モニタリング・スレッドは定期的にその結果のみロギングする方式を使用します。そのため、ログに残されたスレッド数の増減変化や実際のスレッド・プールの状態変化が一致しない場合があるので、注意が必要です。

アクティブ・マネージメントと状態通知

ワーカー・スレッド・プールにはアクティブ・マネージメントに関する設定が含まれています。アクティブ・マネージメントは、管理者が指定した特定の状態に至るとWebエンジンが警告メッセージをEメールで通知するか、

Webエンジンが属するサーバーの再起動を勧告する設定です。設定可能な値は、ワーカー・スレッドがブロックされたと判断される時間です。これによってブロックされたワーカー・スレッドが増加する場合、いくつか以上なら警告Eメールまたはエンジンの再起動勧告メッセージを出力するなどの特定作業の実行を設定します。

注

再起動勧告が出力されると、サーバーの要求処理が順調でない可能性が大きいため、管理者はメッセージを確認してサーバーを再起動するか否かを判断する必要があります。

2.3. Webコネクションの設定

AJPリスナー、WebtoBコネクター、Tmaxコネクターを使用する場合、各連動製品別に設定が必要です。すべてのWebコネクションはWebエンジンで管理するため、本節ではWebエンジンの設定についてのみ説明します。WebtoBと別のWebサーバーの設定については、「[2.4. 負荷分散のためのWebサーバーの設定](#)」を参照してください。

参考

1. JEUS 6のコンテキスト・グループは無くなりました。コンテキスト・グループが管理していたほとんどの事項はWebエンジンが管理します。
 2. Webリスナーの場合、サーバーが提供する統合されたサービス・リスナーを使用します。そのため、AJP、HTTP、TCPリスナーの場合、サーバーにリスナーを設定し、そのサーバー・リスナーを参照する方式で設定します。したがって、「**Server Listener Ref**」という設定が追加されました。しかし、WebtoB、Tmaxに対するコネクターはJEUSがクライアントとして直接接続するので、既存の設定方式を保持します。
-

コネクターを追加、修正、削除するときは、WebAdminとコンソール・ツールが使用できます。コネクターを設定するときにはWebAdminを使用することをお勧めします。WebAdminを使用した設定方法については本節で説明します。コンソール・ツールを使用して設定する方法は、『*JEUS リファレンスガイド*』の「4.2.8. Webエンジン関連コマンド」を参照してください。

2.3.1. リスナーの共通設定

本節では、各リスナーで共通して使用する主要設定について説明します。

● Name

- Webコネクションを識別する一意の名前です。Webエンジン内で一意である必要があり、必須設定です。

● Server Listener Ref

- リスナーが参照するサーバー・リスナーです。
- HTTP、AJPリスナーの場合、相互同じサーバー・リスナーを共有することができます。ただし、HTTPリスナーを管理目的で設定した場合は共有できません。設定しない場合、サーバーのデフォルト・リスナーを使用します。
- TCPリスナーは共有できません。
- 2つ以上の同じプロトコル・リスナーは共有できません。たとえば、AJPリスナーを2つ設定したのに、2つともこの設定がない場合はサーバー・デフォルト・リスナーを共有することになるため、設定エラーが発生します。そのため、2つのうち1つは別途のサーバー・リスナーを参照する必要があります。
- サーバー・リスナーの<keep-alive-timeout>設定により、要求しないクライアントの接続を切断することができます。(単位: ms)

参考

HTTPリスナー、TCPリスナーでpostdata-read-timeoutを適用するには、サーバー・リスナーの<keep-alive-timeout>を設定する必要があります。

● Connection Type

- 各リスナーまたはコネクタを介して送信される応答のコネクション・ヘッダーを強制的に設定します。
- 以下のいずれかを設定できます。

設定値	説明
keep-alive	応答を送信した後も接続を維持する場合に設定します
close	応答を送信した後、接続を切断する場合に設定します
none	要求ヘッダーに定義された属性に合わせて応答のコネクション・ヘッダーに従う場合に設定します。Webエンジンは、何にも設定しないと「none」に設定した場合と同様に動作します

参考

AJPリスナーの場合、「**Connection Type**」項目が設定できません。AJPリスナーは、ApacheのようなWebサーバーの設定に従うことをお勧めします。

● Thread Pool

- ワーカー・スレッド・プールに関する設定です。ただし、WebtoBコネクタの場合はこの共通設定を使用せず、WebtoBコネクタだけのスレッド・プールを設定します。

– 以下は、詳細設定の下位項目についての説明です。

項目	説明
Min	プールで管理するワーカー・スレッドの最小数です
Max	プールで管理するワーカー・スレッドの最大数です
Step	使用しない設定です
Maximum Idle Time	<p>プール内に存在していたスレッドが削除されるまでの、使用されていない時間を指定します。これによりシステム・リソースは増加します。</p> <p>各ワーカー・スレッド・プールは、要求待機キュー(Request Wait Queue)を持っています。このキューは、実際に使用可能なワーカー・スレッドより多くの要求が受信された場合に使われます。同キューは、ソケット・リスナーによって保持される低いレベルのバックログ・キューより上位レベルです。以下の2つの項目はこのキューと関連するキューです</p>
Max Wait Queue	より多くのワーカー・スレッドがプールに生成される前に、要求待機キューに存在できる要求数を指定します
Max Queue	<p>キューに待機できる要求の最大数を指定します。</p> <p>キューがいっぱいになった後、さらに多くの要求が受信されると、busyページがクライアントに返されます。</p> <p>値が-1の場合、キュー・サイズを制限しないことを意味します(ブロッキング方式のリスナーの場合)。リスナーがNIO(Non-blocking I/O)方式を使用する場合は、エンジン内部的にBoundedキューを使用するため、常に0より大きい必要があります。NIO方式において、0より大きいか、小さい値を設定した場合は、デフォルト値の4096を使用します。</p> <p>Tmaxコネクターの場合、同設定は無効です</p>
Thread State Notify	<p>ブロックされるワーカー・スレッドが発生する場合は、これを通知します。</p> <p>各スレッド・プールは障害発生に対処するアクションを定義する「Thread State Notify」項目を持っています。詳しい内容は、「2.3.7. 自動スレッド・プール管理の設定(スレッド状態の通知)」を参照してください</p>

● Output Buffer Size

- アプリケーションが使用する応答バッファのサイズです。バッファがフルになると、当該バッファの内容をWebエンジンが自動的にフラッシュします。(単位: バイト)
- AJPリスナーの場合、この値をmod_jkのworkers.propertiesファイルのmax_packet_sizeと一致させることをお勧めします。mod_jkの設定については、[「2.4. 負荷分散のためのWebサーバーの設定」](#)を参照してください。

● Postdata Read Timeout

- 要求ボディを読み込む際、待機できる最大時間を設定します。ServletInputStream.read()メソッドで適用します。(単位: ms)
- HTTPリスナーとTCPリスナーでは、**タイムアウトの適用時点が変更されました**。各リスナーに存在するI/O制御スレッドで要求ボディをすべて読み込むので、実際にサーブレットに渡す前にタイムアウトの可否が決定されます。要求ボディの読み込み中にタイムアウトが発生すると、500エラーを返します。ただし、HTTP要求がChunkedタイプの場合は、既存と同様、サーブレットがServletInputStream.read()を呼び出すときに適用されます。

● Max Post Size

- 大きすぎるPOST要求のデータが受信され、読み取り、分析するために多くの負荷がかかり、別の要求処理に障害が生じる場合のための設定です。同設定によって特定サイズ以上の要求に対する処理負担を減らすことができます。(デフォルト値: -1)
- 同設定はPOST要求の場合、要求のコンテンツ・タイプによってデータの最大サイズをバイト単位で制限します。
 - コンテンツ・タイプがx-www-form-urlencodedの場合
要求に伴うデータのバイト・サイズが設定値を超える場合、JEUSは当該要求を処理せずに「413 Request entity too large」応答を送信し、処理を完了します。
 - コンテンツ・タイプがmultipart/form-dataの場合
同設定で、アップロード・ファイルのサイズとPOST要求の全体サイズが制限できます。アップロード・ファイルが含まれたサイズを制限する場合は、Servlet 3.0のmultipart関連設定を使用することをお勧めします。
- 同設定の適用可否は、Servlet 3.0 WebアプリケーションDD設定の影響を受けます。設定によって以下のように動作します。
 - web.xmlの<multipart-config>設定による動作
web.xmlに<multipart-config>設定が存在する場合、この設定の<max-file-size>と<max-request-size>が適用されます。<multipart-config>設定が存在しない場合は、これに関する制限をWebエンジンでは提供しません。そのため、各アプリケーションで設定して使用することをお勧めします。
 - <content-length>設定による動作
<content-length>が設定されていると、この値が設定値を超過した場合に要求処理が制限されます。<content-length>が設定されていない要求の場合は、名前/値のペア(パラメータ)のバイト合計が設定値を超えるとに要求処理が制限されます。

- 負の数に設定した場合は、デフォルト値に設定した場合と同様に動作します。データ・サイズの制限はありません。
- 同設定は、HTTP要求を受けるリスナーおよびコネクターの場合のみ有効です。(TCPリスナーとTmaxコネクターの場合は無効です。)

● Max Parameter Count

- 1つの要求に過剰な名前/値のペア(パラメータ)が含まれ、分析および管理の負担が増加することを防ぐために設定します。(デフォルト値: -1)
- GETとPOST要求に含まれた「名前/値」のペア(パラメータ)の総数を設定値で制限します。
- GET要求のクエリー文字列、POST要求のデータやmultipart/formに名前/値のペアで受信されたすべてのパラメータ数が設定値を超えた場合、「413 request entity too large」応答を送信し、処理を中断します。
- 負の数に設定した場合は、デフォルト値に設定した場合と同様に動作します。パラメータ数の制限はありません。
- 同設定は、HTTP要求を受けるリスナーおよびコネクターの場合のみ有効です。(TCPリスナーとTmaxコネクターの場合は無効です。)

● Max Header Count

- 1つの要求に含まれたヘッダー数が多すぎると、要求の読み込みに負荷がかかります。この場合、当該要求を拒否し、サーバーの負荷を減らすことができます。(デフォルト値: -1)
- 設定値以上のヘッダー数が含まれた要求は処理せずに「400 Bad request」応答を送信した後、接続を切断します。すなわち、ヘッダー以降のデータは読み込まずに破棄することで、サーバーの負担を減らすことになります。
- 負の数に設定した場合は、デフォルト値に設定した場合と同様に動作します。ヘッダー数の制限はありません。
- 同設定は、HTTP要求を受けるリスナーおよびコネクターの場合のみ有効です。(TCPリスナーとTmaxコネクターの場合は無効です。)

● Max Header Size

- 1つの要求に含まれたヘッダーが大きすぎる場合、当該ヘッダーを読み取り、処理するために負荷がかかります。この場合、当該要求を拒否し、サーバーの負荷を減らすことができます。(デフォルト値: -1)
- 設定値よりバイト・サイズ(このときのヘッダーのバイトは、要求のヘッダー1つを表すヘッダー名、区切り子、ヘッダー値がすべて含まれたサイズ)が大きいヘッダーが含まれた要求は処理せずに「400 Bad request」を送信した後、接続を切断します。すなわち、ヘッダー以降のデータは読み込まずに破棄することで、サーバーの負担を減らすことになります。

- 負の数に設定した場合は、デフォルト値に設定した場合と同様に動作します。ヘッダー・バイトの制限はありません。
- 同設定は、HTTP要求を受けるリスナーおよびコネクターの場合のみ有効です。(TCPリスナーとTmaxコネクターの場合は無効です。)

● Max Query String Size

- GET要求のクエリー文字列が長い場合、分析および管理の負荷がかかります。この場合、クエリー文字列のサイズを制限し、負荷を減らすことができます。(デフォルト値: 8192、単位: バイト)
- 設定値以上のクエリー文字列が含まれた要求が受信される場合、その要求に対して「400 Bad Request」応答を送信した後、残りのデータは読み込まずに破棄することでサーバーの負荷を減らします。
- 負の数に設定した場合は、デフォルト値に設定した場合と同様に動作します。クエリー文字列のサイズの制限はありませんが、JEUSは内部的に要求行(要求の最初行)のサイズを64KBに制限するため、それ以上は意味がありません。
- 同設定は、HTTP要求を受けるリスナーおよびコネクターの場合のみ有効です。(TCPリスナーとTmaxコネクターの場合は無効です。)

● Compression

- HTTPリスナーまたはWebtoBコネクタを介して送信される応答を圧縮する機能を設定します。
- 要求のAccept-encodingにgzipがある場合、応答のMIMEタイプが該当の項目で設定されたのと同様な場合は、圧縮機能が動作します。
- HTTP 1.1接続のみ適用されます。圧縮された場合は、応答ヘッダーのcontent-encoding : gzipが追加されます。
- 以下は、設定画面についての説明です。

[図 2.3] 圧縮機能の設定画面 - 詳細設定

項目	説明
Max Nonchunked Compression Size	<p>圧縮する応答のサイズが大きい場合はメモリーを多く占める可能性があります。サーブレットが指定したcontent-length値がこちらに設定したサイズを超える場合は、chunked応答を返します</p> <p>(単位: KB、デフォルト値: 128KB)</p>

項目	説明
Mime Type	圧縮機能を使用するMIMEタイプを指定します。必ず1つ以上を指定してください

2.3.2. AJPリスナーの設定

WebAdminとコンソール・ツールを使用してAJPリスナーを追加、変更、削除することができます。

WebAdminの使用

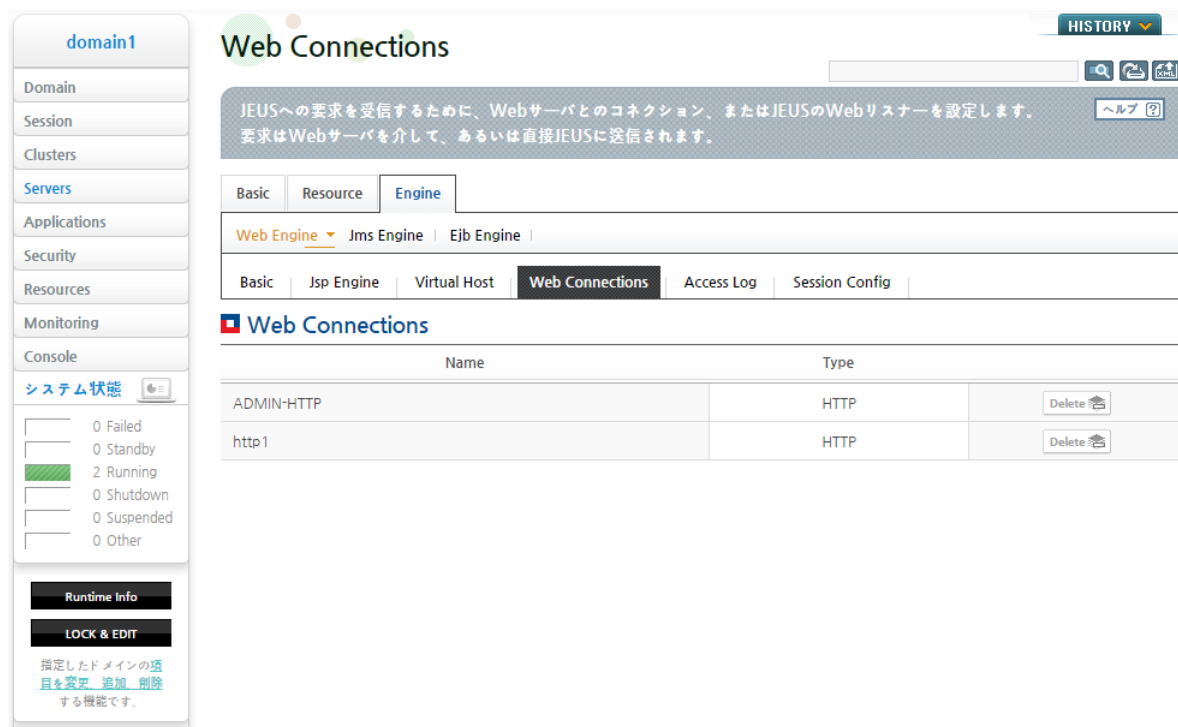
• 追加および変更

以下は、WebAdminを使用してAJPリスナーを追加および変更する方法です。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。

サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。

[図 2.4] AJPリスナーの追加および変更 - 追加



2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。

3. AJPリスナーを追加するには[AJP13]ボタンをクリックし、修正するときはWebコネクション・リストから「ajp13」タイプリスナーをクリックします。[Ajp 13 Listener]画面へ移動し、基本情報を設定します。

[図 2.5] AJPリスナーの追加および変更 - 基本設定

AJP13 Listener

Apache mod_jkモジュールを使用するAJP 1.3プロトコルをサポートします。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

動的設定 * 必須項目

確認 再設定

Name *	ajp13 リスナーまたはコネクタを識別できる一意の名前を指定します。
Output Buffer Size	byte out.write()メソッドにより出力されるJSP/Servlet出力を一時保存する出力バッファのサイズを指定します。バッファがいっぱいになると、自動でクライアントにデータを送信します。デフォルト値は、AJP13の場合は8192です。他のプロトコルの場合は、new Socket().getSendBufferSize()に従いますが、最小8192、最大16384の範囲になります。
Server Listener Ref *	ajp AJP13 Webリスナーが参照するサーバリッスナーを指定します。BASEリスナーは設定できません。

参考

AJPリスナーはAJPバージョンの1.3に準拠します。

4. 「Thread Pool」項目で、リスナーに割り当てられるスレッド・プールについて設定します。

[図 2.6] AJPリスナーの追加および変更 - Thread Poolの設定

Thread Pool

AJP13 Webリスナーから受信した要求を処理するスレッドプールを設定します。

Min *	20 プールで保持するワークスレッドの最小数を指定します。
Max *	40 プールで保持するワークスレッドの最大数を指定します。
Max Idle Time	[デフォルト: 300000] ワークスレッドをアイドル状態で維持できる最大時間を指定します。指定した時間が過ぎると、ワークスレッドはプールから削除されます。この値は、<min>の設定値を超えて生成されたスレッドに適用されます。
Max Queue	[デフォルト: -1] キューに溜めることができる最大要求数を指定します。この設定はTmax/WebtoBコネクタでは使用されません。WebtoBコネクタのキューサイズは、WebtoB設定ファイルのMaxQCount値で構成されます。

「Thread State Notify」項目で、リスナーに割り当てられたスレッド状態の自動通知に関する設定を行います。詳しい説明は、「2.3.7. 自動スレッド・プール管理の設定(スレッド状態の通知)」を参照してください。

[図 2.7] AJPリスナーの追加および変更 - Thread State Notifyの設定

▼ Thread State Notify

ワーカスレッドがブロックされる基準とブロックされたときに行うアクションを設定します。

Max Thread Active Time	<input type="text" value="0"/> ms [デフォルト: 0] スレッドがブロックされたとき見なす時間を指定します。スレッドのブロック状態の可否をチェックする周期は、<monitoring><check-thread-pool>設定に従います。基本的にブロック状態のスレッドは捨てて、新しいスレッドを作成してスレッドプールに含めます。ただし、WebtoBコネクタとTmaxコネクタに属するスレッドプールについては何のアクションも行わずに、ただブロック状態を表示します。
Notify Threshold Ratio	<input type="text" value="0.0"/> [デフォルト: 0.0] メール通知の基準を設定します。スレッドプールの最大値に対するブロックされたスレッドの割合が指定の値以上に増えた場合に、Eメールで通知します。
Restart Threshold Ratio	<input type="text" value="0.0"/> [デフォルト: 0.0] Webエンジンの再起動を勧める基準となるブロックスレッドの割合を指定します。1より小さい小数の値を指定します。ブロックされたスレッドが指定の割合に達すると、サーバの再起動を勧めるメッセージが表示されます。
Notify Subject	<input type="text"/> Notify Threshold Ratio設定によって通知されるEメールのタイトルを記述します。
Restart Subject	<input type="text"/> Restart Threshold Ratio設定によって通知されるEメールのタイトルを記述します。
Interrupt Thread	<input type="checkbox"/> [デフォルト: false] ブロックされたスレッドをインタラプトするか否かを設定します。インタラプトは、スレッドに実行している動作を取り消すようにヒントを与えるだけです。スレッド内部でインタラプト状態をチェックしていない場合は、何の影響も与えません。
Active Timeout Notification	<input type="checkbox"/> [デフォルト: false] ブロックされたスレッドが発生したことをEメールで通知するか否かを設定します。

5. 詳細設定の項目を設定します。Web攻撃への対応およびいくつかの設定が含まれています。

[図 2.8] AJPリスナーの追加および変更 - 詳細設定

🔔 詳細設定 すべてを開く

📌 Ajp13 Listener

Postdata Read Timeout	<input type="text"/>	ms
Max Post Size	<input type="text"/>	
Max Parameter Count	<input type="text"/>	
Max Header Count	<input type="text"/>	
Max Header Size	<input type="text"/>	byte
Max Querystring Size	<input type="text"/>	byte
Server Access Control	<input type="checkbox"/>	
Allowed Server	<input type="text"/>	

以下は、設定項目についての説明です。以下で説明されていない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Server Access Control	アクセス制限機能を使用するか否かを設定します
Allowed Server	許可するWebサーバーのIPアドレスを設定します。入力するアドレスが1つ以上の場合は、行で区切ります。「 Server Access Control 」項目がチェック(true)されたときに適用されます

6. 設定が終わった後、**[確認]**ボタンをクリックすると、以下のとおり「ajp13」タイプのリスナーが追加されたことが確認できます。

[図 2.9] AJPリスナーの追加および変更 - 設定の確認



7. 設定が終わったら、設定内容を適用するために**[Apply Changes]**ボタンをクリックします。以下のよう
に結果メッセージが出力されます。

[図 2.10] AJPリスナーの追加および変更 - 設定の適用結果

Web Connections

HISTORY

JEUSへの要求を受信するために、Webサーバとのコネクション、またはJEUSのWebリスナーを設定します。要求はWebサーバを介して、あるいは直接JEUSに送信されます。

domain.xmlの設定を変更しました。

domain.xml : PENDING

servers.server.{? name == 'server1' }.webEngine.webConnections : PENDING

変更内容を適用するには、サーバを再起動してください。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

Web Connections

Name	Type	
ajp13	AJP13	Delete
ADMIN-HTTP	HTTP	Delete
http1	HTTP	Delete

● 削除

以下は、WebAdminを使用してAJPリスナーを削除する方法です。

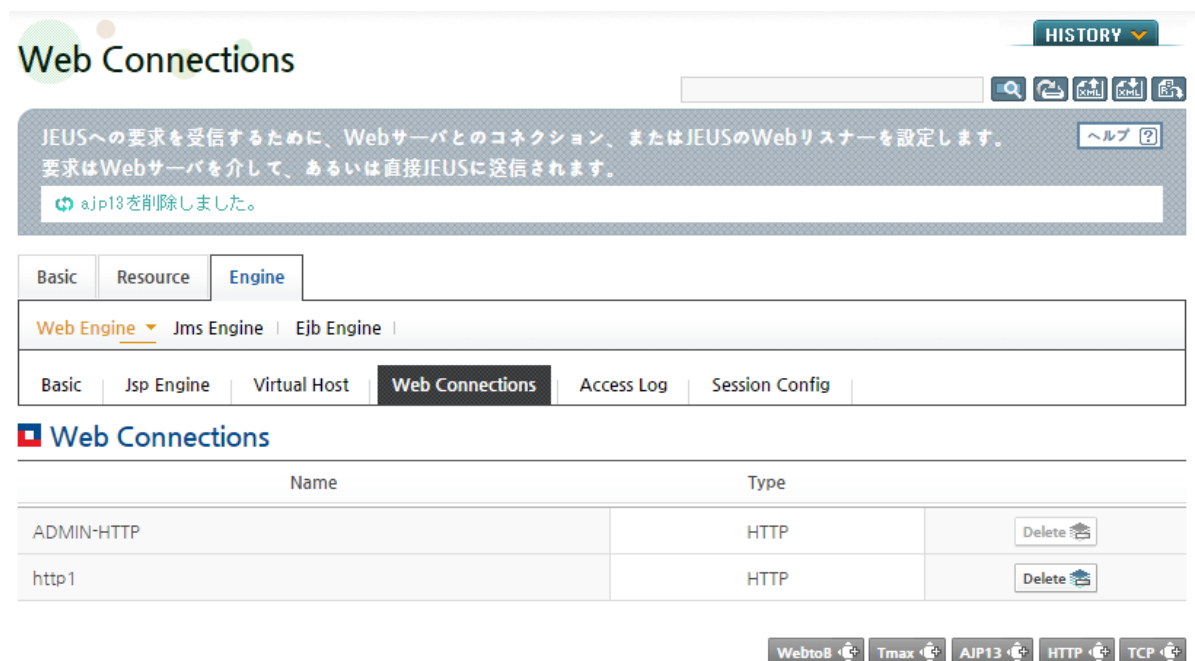
1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。
2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。

[図 2.11] AJPリスナーの削除 - メニューの移動



3. 削除するリスナーの[Delete]ボタンをクリックします。
4. 選択したリスナーが削除されると、以下のとおり結果メッセージが出力されます。

[図 2.12] AJPリスナーの削除 - 削除の確認



5. 設定内容を適用するために[Apply Changes]ボタンをクリックすると、設定適用の結果メッセージが出力されます。([図 2.10]を参照)

コンソール・ツールの使用

以下は、コンソール・ツールを使用してAJPリスナーを追加、変更、削除する方法です。

• 追加

コンソール・ツールを使用してAJPリスナーを追加するには、**add-web-listener**コマンドを実行します。詳細については、『*JEUS リファレンスガイド*』の「4.2.8.8. add-web-listener」を参照してください。

```
add-web-listener [-cluster <cluster-name> | -server <server-name>]
                  [-f, --forceLock]
                  -name <web-connection-name>
                  -tmin <minimum-thread-num>
                  [-tmax <maximum-thread-num>]
                  [-ajp | -http | -tcp]
                  -slref <server-listener-ref-name>
                  [-dcc <dispatcher-config-class>]
                  [-http2]
                  [-tauto]
                  [-tlimit <thread-num-limit>]
```

• 変更

コンソール・ツールを使用してAJPリスナーを変更するには、**remove-web-listener**コマンドを実行します。詳細については、『*JEUS リファレンスガイド*』の「4.2.8.19. modify-web-listener」を参照してください。

```
modify-web-listener [-cluster <cluster-name> | -server <server-name>]
                    [-f, --forceLock]
                    -name <web-connection-name>
                    [-tmin <minimum-thread-num>]
                    [-tmax <maximum-thread-num>]
                    [-tidle <max-idle-time>]
                    [-tauto <enable-auto-tuning>]
                    [-tlimit <thread-num-limit>]
                    [-obuf <output-buffer-size>]
                    [-http2 <enable-http2>]
```

• 削除

コンソール・ツールを使用してAJPリスナーを変更するには、**remove-web-listener**コマンドを実行します。詳細については、『*JEUS リファレンスガイド*』の「4.2.8.29. remove-web-listener」を参照してください。

```
remove-web-listener [-cluster <cluster-name> | -server <server-name> |  
                    -f, --forceLock] <web-connection-name>
```

2.3.3. HTTPリスナーの設定

WebAdminまたはコンソール・ツールを使用してHTTPリスナーを追加、変更、削除することができます。HTTPリスナーは内部管理のためにのみ使用することをお勧めします。

参考

JEUS 8はHTTP/2をサポートしています。HTTP/2の使用方法については、「[2.6. HTTP/2の使用](#)」を参照してください。

WebAdminの使用

• 追加および変更

以下は、WebAdminを使用してHTTPリスナーを追加および変更する方法です。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。([\[図 2.4\]](#)を参照)
2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。
3. HTTPリスナーを追加するときは**[HTTP]**ボタンをクリックし、変更する場合はWebコネクション・リストから「http」タイプのリスナーをクリックします。**[Http Listener]**画面に移動して基本情報を設定します。

[図 2.13] HTTPリスナーの追加および変更 - 基本設定

HTTP Listener

HISTORY

ヘルプ

HTTPリスナーは、Webエンジンが提供するWebサーバです。 HTTPリスナーは小規模の運用環境、あるいは内部管理用途でのみ使用してください。 大規模な運用環境では、WebtoBまたはApache Webサーバを使用することを推奨します。

BasicResourceEngine

Web EngineJms EngineEjb Engine

BasicJsp EngineVirtual HostWeb ConnectionsAccess LogSession Config

動的設定 必須項目

確認再設定

Name *	<div>http1-1</div> <div>リスナーまたはコネクタを識別できる一意の名前を指定します。</div>
Output Buffer Size	<div>8192byte</div> <div>out.write()メソッドにより出力されるJSP/Servlet出力を一時保存する出力バッファのサイズを指定します。 バッファがいっぱいになると、自動でクライアントにデータを送信します。 デフォルト値は、AJP13の場合は8192です。 他のプロトコルの場合は、new Socket().getSendBufferSize()に従いますが、最小8192、最大16384の範囲になります。</div>
Server Listener Ref *	<div>http-listener</div> <div>HTTPリスナーが参照するサーバのリスナーを指定します。 「ADMIN-HTTP」と名付けたリスナーは、内部管理用HTTPリスナーとして使用できます。 このリスナーは、常にbaseサーバリスナーを参照します。 また、ドメイン管理サーバのWebAdminにアクセスし、グレースフル再配布が実行された(gracefully redistributed)Webアプリケーションをテストするために使用できます。 しかし、一般Webアプリケーションにはアクセスできません。 このリスナーは、Server Access ControlおよびAllowed Server項目を設定して使用することを推奨します。</div>

4. Thread Pool領域でリスナーに割り当てられるスレッド・プールについて設定します。

[図 2.14] HTTPリスナーの追加および変更 - スレッド・プールの設定

Thread Pool

Web Listener에서 요청을 받아 처리할 Thread Pool을 설정한다.

Max Idle Time	<div>300000</div> <div>[default: 300000] Pool에서 제거되기 전에 Idle 상태로 머물러 있는 Worker Thread의 최대 시간을 설정한다. <min>을 초과해서 생성된 스레드에 적용하는 값이다.</div>
Max Queue	<div>-1</div> <div>[default: -1] Queue에 대기할 수 있는 최대 요청값을 설정한다. 이 설정은 Tmax/WebtoB connector에서는 사용되지 않는다. WebtoB connector의 Queue 사이즈는 WebtoB 설정 파일에 MaxQCount 값으로 구성된다.</div>

Thread Pool Size

Default

Min *	<div>10</div> <div>Pool에서 Worker Thread를 유지해야 하는 최소 개수를 설정한다.</div>
Max *	<div>20</div> <div>Pool에서 Worker Thread를 유지해야 하는 최대 개수를 설정한다.</div>

AutoTuning

auto-tuning의 적용 여부. 이 옵션을 켜면 min, max는 설정을 해도 적용되지 않는다. http connection에 대해서만 적용된다.

Limit	<div>0</div> <div>auto-tuning 적용시 Thread 수의 한계점.</div>
-------	--

「Thread State Notify」項目の設定はAJPリスナーと同様です。設定画面は、[\[図 2.7\]](#)を参照してください。

以下は、自動チューニング項目についての説明です。

項目	説明
Auto Tuning	<p>自動チューニングを適用するかどうかを指定します。このオプションを有効にすると、MinとMaxは設定しても適用されません。HTTPコネクションに対してのみ適用されます。</p> <p>自動チューニングは、以下のとおりスレッド数を調整します</p> <ul style="list-style-type: none">– 最初: CPUコア数の3倍– 増加: 要求により負荷が集中した場合、これを検知してスレッドをCPUのコアの数分だけ増やします– 減少: スレッド・プールのkeep alive timeの減少方式と同じです <p>[注意]</p> <p>現在の自動チューニングは、CPUを100%使用することを目標とするアルゴリズムが使われており、メモリーの使用量は考慮されていません。そのため、実際の運用サーバーに適用するときは、このような点を考慮する必要があります。テスト向けあるいは、最大性能を出すプールの最大値を測定する用途で使用することを推奨します</p>
Limit	自動チューニングを適用する際、スレッド数のリミット値です

5. 詳細設定の項目を設定します。

[図 2.15] HTTPリスナーの追加および変更 - 詳細設定

詳細設定

すべてを開く

Http Listener

Postdata Read Timeout

ms

Max Post Size

Max Parameter Count

Max Header Count

Max Header Size

byte

Max Querystring Size

byte

Connection Type

Max Keep Alive Request

Server Access Control

☐

Allowed Server

Compression

☐

Max Nonchunked Compression Size

Mime Type

HTTP/2

☐

Enable Server Push

☒

Max Concurrent Streams

Max Frame Size

SETTINGS Frame Ack Timeout

以下は、設定項目についての説明です。以下で説明していない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Max Keep Alive Request	クライアントがKeep Aliveコネクションを使用する場合、そのコネクションで許可する最大要求数を設定します
Server Access Control	アクセス制限機能を使用するかどうかを設定します

項目	説明
Allowed Server	許可するWebサーバーのIPアドレスを設定します。入力するアドレスが1つ以上の場合は行で区切ります。「 Server Access Control 」項目がチェック(true)された場合に適用されます
Enable Server Push	サーバー・プッシュを使用するかどうかを設定します
Max Concurrent Streams	同時に受信する要求数を設定します
Max Frame Size	データ・フレーム1つの最大サイズを設定します
SETTINGS Frame Ack Timeout	セッティング・フレームを送信した後、ACKが受信されるまでのタイムアウトを設定します。タイムアウトを超過すると、該当するクライアントとの接続を切断します

参考

HTTP/2についての詳細内容は、「[2.6. HTTP/2の使用](#)」を参照してください。

- 設定後、**[確認]**ボタンをクリックすると、以下のとおり「http」タイプのリスナーが追加されたことが確認できます。

[図 2.16] HTTPリスナーの追加および変更 - 追加の確認

Web Connections

JEUSへの要求を受信するために、Webサーバーとのコネクション、またはJEUSのWebリスナーを設定します。要求はWebサーバーを介して、あるいは直接JEUSに送信されます。

追加されました。

Basic Resource **Engine**

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host **Web Connections** Access Log Session Config

Name	Type	
ADMIN-HTTP	HTTP	Delete
http1	HTTP	Delete
http1-1	HTTP	Delete

Webto8 Tmax AJP13 HTTP TCP

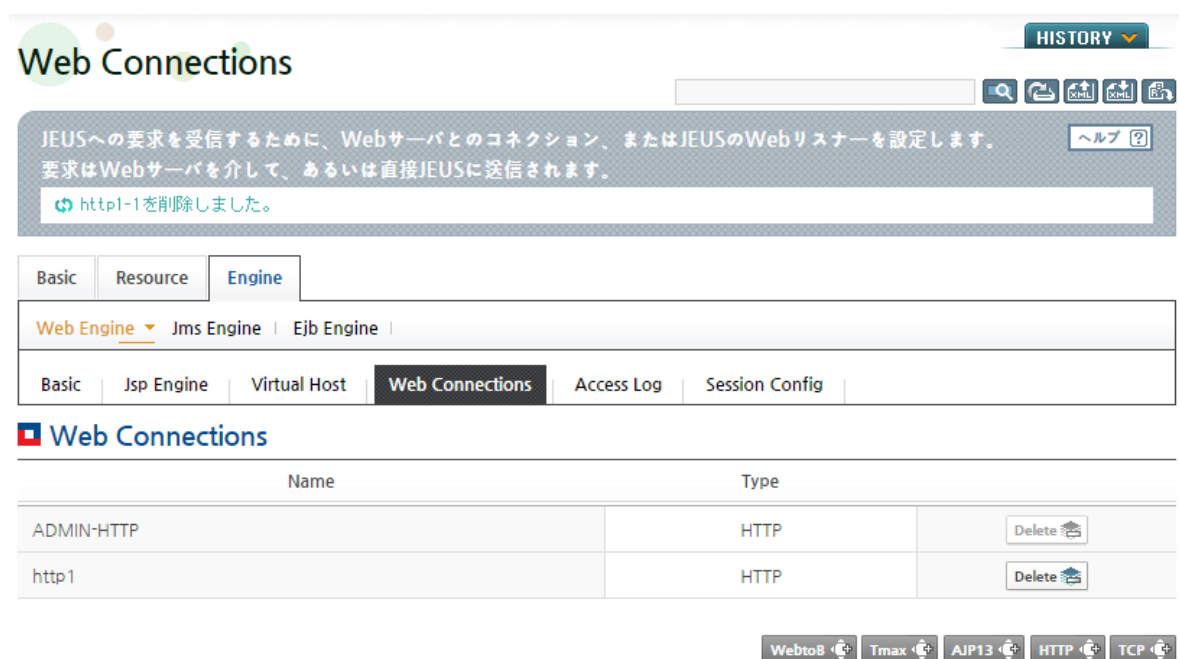
- 設定内容を適用するために**[Apply Changes]**ボタンをクリックすると、設定適用の結果メッセージが出力されます。([図 2.10]を参照)

● 削除

以下は、WebAdminを使用してHTTPリスナーを削除する方法です。

1. WebAdminの左側のメニューから[**Servers**]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[**Engine**] > [**Web Engine**] > [**Web Connections**]メニューを選択すると、Webコネクション・リストの照会画面へ移動します。([図 2.11]を参照)
2. 設定および設定変更のため、画面左側の[**LOCK & EDIT**]ボタンをクリックして設定変更モードに切り替えます。
3. 削除するリスナーの[**Delete**]ボタンをクリックします。
4. 選択したリスナーが削除されると、以下のとおり結果メッセージが出力されます。

[図 2.17] HTTPリスナーの削除 - 削除の確認



5. 設定内容を適用するために[**Apply Changes**]ボタンをクリックすると、設定適用の結果メッセージが出力されます。([図 2.10]を参照)

コンソール・ツールの使用

コンソール・ツールを使用してHTTPリスナーを追加、変更、削除する方法は、AJPリスナーの方法と同様です。ただし、HTTPリスナーを追加、変更する場合にのみHTTP/2の使用有無を設定できるオプションを提供しています。詳しい内容は、「2.3.2. AJPリスナーの設定」の「コンソール・ツールの使用」を参照してください。

2.3.4. TCPリスナーの設定

TCPリスナーはカスタム・プロトコルを使用して相互通信を可能にする特殊なリスナーです。TCPリスナーは基本的にサーバー・リスナーを別のWebリスナーと共有できないため、サーバーにTCPリスナーのための専用リスナーを追加する必要があります。WebAdminまたはコンソール・ツールを使用してTCPリスナーを追加、変更、削除することができます。

WebAdminの使用

• 追加および変更

以下は、WebAdminを使用してTCPリスナーを追加および変更する方法です。

- WebAdminの左側のメニューから**[Servers]**を選択するとサーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。[\(図 2.4\)](#)を参照)
- 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。
- TCPリスナーを追加するときは**[TCP]**ボタンをクリックし、変更する場合はWebコネクション・リストから「tcp」タイプリスナーをクリックします。**[Tcp Listener]**画面に移動して基本情報を設定します。

[図 2.18] TCPリスナーの追加および変更 - 基本設定

The screenshot displays the 'TCP Listener' configuration interface. At the top, there's a 'HISTORY' dropdown and a search bar. Below this is a descriptive text box stating that TCP listeners support custom protocols. The main configuration area has tabs for 'Basic', 'Resource', and 'Engine'. Under 'Engine', there are sub-tabs for 'Web Engine', 'Jms Engine', and 'Ejb Engine'. The 'Web Engine' sub-tab is active, showing further sub-tabs: 'Basic', 'Jsp Engine', 'Virtual Host', 'Web Connections' (selected), 'Access Log', and 'Session Config'. Below these are checkboxes for '動的設定' and '必須項目', and buttons for '確認' and '再設定'. The configuration fields are as follows:

Name *	tcp1 <small>リスナーまたはコネクタを識別できる一意の名前を指定します。</small>
Output Buffer Size	<input type="text"/> byte <small>out.write()メソッドにより出力されるJSP/Servlet出力を一時保存する出力バッファのサイズを指定します。バッファがいっぱいになると、自動でクライアントにデータを送信します。デフォルト値は、AJP13の場合は8192です。他のプロトコルの場合は、new Socket().getSendBufferSize()に従いますが、最小8192、最大16384の範囲になります。</small>
Server Listener Ref *	tcp <small>TCPリスナーが参照するサーバリスナーを指定します。BASEリスナーは指定できません。また、サーバリスナーを他のWebリスナーと共有できません。</small>
Dispatcher Config Class *	com.tmax.jeus.TcpDispatcherConfig <small>ディスパッチャの設定クラスを指定します。クラスは必ずクラスパスに設定された場所に存在し、クラス名は完全修飾クラス名である必要があります。</small>

以下は、設定項目についての説明です。以下で説明していない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Dispatcher Config Class	<p>Dispatcher Config Classの名前を指定します。Dispatcher Config Classは、TCPリスナーとクライアントの間で定義されたプロトコルを定義したクラスです。jeus.servlet.tcp.TCPDispatcherConfigインターフェースを実装した正式なクラス名を定義します。</p> <p>Dispatcher Config Classが存在しない場合はTCPリスナーが動作しません。実装されたクラスはJEUS_HOME/lib/applicationの下位に存在する必要があります。</p> <p>WebエンジンにデプロイするWebコンテキストにはjeus.servlet.tcp.TCPServletを実装したクラスが含まれる必要があります、web.xmlにマッピングします。詳しい内容は、「2.5. TCPリスナーの使用」を参照してください</p>

4. 「**Thread Pool**」項目の設定はAJPリスナーと同様です。設定画面は、「[2.3.2. AJPリスナーの設定](#)」を参照してください。

5. **詳細設定**の項目を設定します。

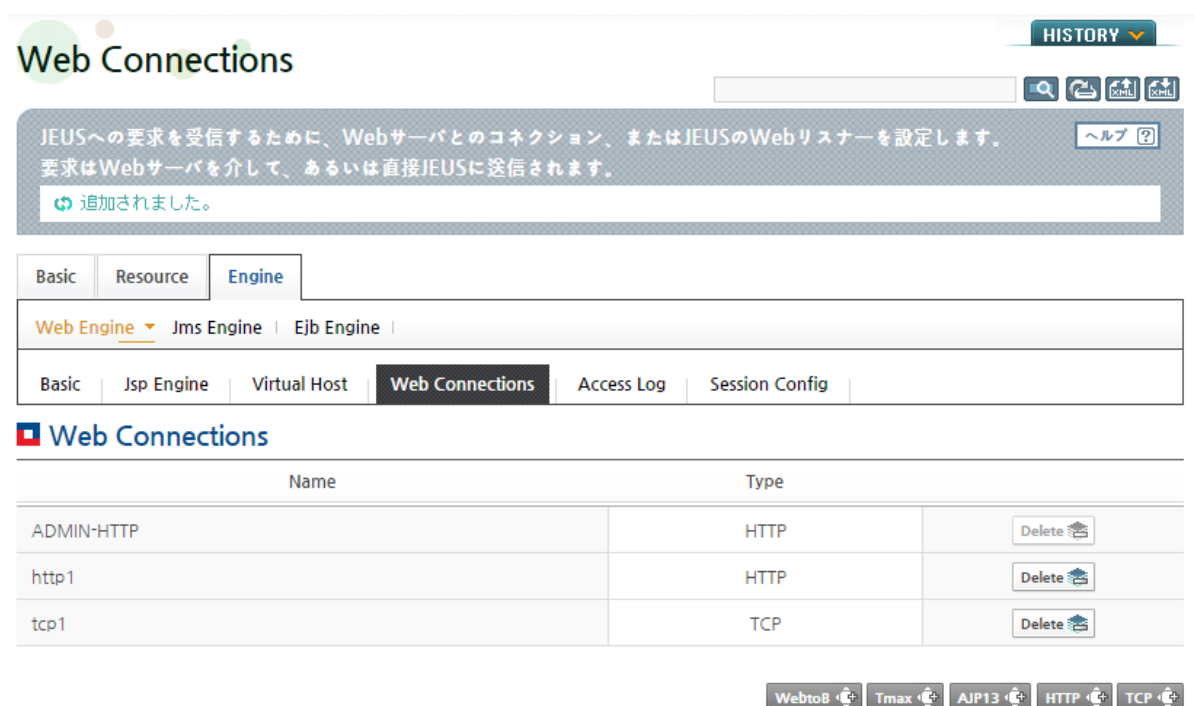
[図 2.19] TCPリスナーの追加および変更 - 詳細設定

以下は、詳細設定の項目についての説明です。以下で説明していない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Servers Adccess Control	アクセス制限機能の使用可否を設定します
Allowed Server	許可するWebサーバーのIPアドレスを設定します。入力するアドレスが1つ以上の場合は行で区分します。「 Server Access Control 」項目がチェック(true)された場合に適用されます

6. 設定後、**[確認]**ボタンをクリックすると、以下のとおり「tcp」タイプのリスナーが追加されたことが確認できます。

[図 2.20] TCPリスナーの追加および変更 - 追加の確認



7. 設定内容を適用するために**[Apply Changes]**ボタンをクリックすると、設定適用の結果メッセージが出力されます。([図 2.10]を参照)

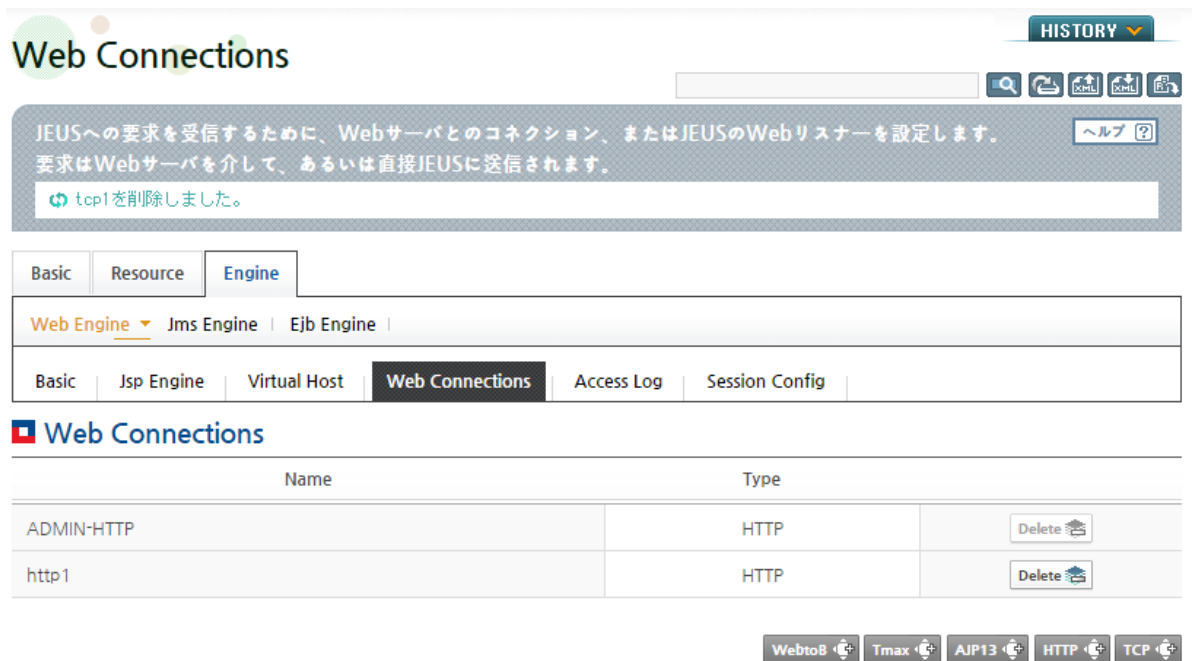
● 削除

以下は、WebAdminを使用してTCPリスナーを削除する方法です。

- WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。([図 2.11]を参照)

2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. 削除するリスナーの[Delete]ボタンをクリックします。
4. 選択したリスナーが削除されると、以下のとおり結果メッセージが出力されます。

[図 2.21] TCPリスナーの削除 - 削除の確認



5. 設定内容を適用するために[Apply Changes]ボタンをクリックすると、設定適用の結果メッセージが確認できます。([図 2.10]を参照)

コンソール・ツールの使用

コンソール・ツールを使用してTCPリスナーを追加、変更、削除する方法は、AJPリスナーの方法と同様です。詳しい内容は、「2.3.2. AJPリスナーの設定」の「コンソール・ツールの使用」を参照してください。

2.3.5. WebtoBコネクタの設定

WebtoBコネクタがコネクションを生成するとき、JEUSがクライアントの役割をするため、WebtoBアドレスとの接続ポートが必要です。WebAdminとコンソール・ツールを使用してWebtoBコネクタを追加、変更、削除することができます。

参考

WebtoBとJEUSの間の通信プロトコルであるWJPバージョンが1から2(以下、WJPv1およびWJPv2)にアップグレードされました。WJPv2の場合、WJPv1に比べてより小さいサイズの packets を使用しており、様々な付加機能を提供します。JEUSから接続するWebtoBがWJPv2をサポートしないバージョンの場合には、WJPv1を使用します。WJPバージョン情報について、WebtoBでは自動的に把握できないため、JEUSに<wjp-version>で設定する必要があります。WJPIは、WebtoB-JEUS Protocolを意味します。

WebAdminの使用

● 追加および変更

以下は、WebAdminを使用してWebtoBコネクタを追加および変更する方法です。

1. WebAdminの左側のメニューから**[Servers]**を選択するとサーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で **[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。([図 2.4]を参照)
2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。
3. WebtoBコネクタを追加するときは**[WEBTOB]**ボタンをクリックし、変更する場合はWebコネクション・リストから「webtob」タイプのコネクタをクリックします。**[WebtoB Connector]**画面に移動して基本情報を設定します。

[図 2.22] WebtoBコネクタの追加および変更 - 基本設定

HISTORY

WebtoBと連動するためのコネクタです。コネクションを作成するとき、JEUSがクライアントになります。

ヘルプ

Basic

Resource

Engine

Web Engine

Jms Engine

Ejb Engine

Basic

Jsp Engine

Virtual Host

Web Connections

Access Log

Session Config

動的設定

必須項目

確認

再設定

Name

*

リスナーまたはコネクタを識別できる一意の名前を指定します。

Output Buffer Size

byte

out.write()メソッドにより出力されるJSP/Servlet出力を一時保存する出力バッファのサイズを指定します。バッファがいっぱいになると、自動でクライアントにデータを送信します。デフォルト値は、AJP13の場合は8192です。他のプロトコルの場合は、new Socket().getSendBufferSize()に従いますが、最小8192、最大16384の範囲になります。

Wjp Version

2

[デフォルト: 2] WebtoBと接続するときに使用するWJPプロトコルのバージョンを決定します。1、2のみ設定できます。WebtoB 4.1.6以下の場合はバージョン2をサポートしないため、1に設定する必要があります。

Registration Id

*

WebtoBサーバと接続するときに使用する登録IDを設定します。WebtoBと約束されたプロトコルにより、最大15文字の制限があります。

Hth Count

1

[デフォルト: 1] HTHプロセスの数を設定します。WebtoBサーバのHTHプロセス数と同一に設定します。

Read Timeout

ms

[デフォルト: 120000] 設定時間の間WebtoBからメッセージが送信されなかった場合は、コネクションに問題があると見なし、新たにコネクションを確立します。WebtoBとJEUS間にファイアウォールがある場合は、WebtoBが送信するPingメッセージの到着の可否をこの設定によりチェックできます。WebtoBのsvrchktime値より大きく設定する必要があります。0より大きく、1秒より小さい値を設定した場合は、無条件で1秒と見なします。-1の場合は、OSの設定に従います。

以下は、設定項目についての説明です。以下で説明していない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
WJP Version	WebtoBと通信するときに使用するWJPバージョンを設定します。WJPv1、v2をサポートします
Registration Id	WebtoB設定ファイルのSERVERセクションの値と一致する必要があります。WJPv1を使用する場合は、必ず15字以内に設定します(必須入力項目)
Hth Count	WebtoB設定ファイルのNODEセクションに指定されたHTHプロセス数と一致する必要があります。「 2.4. 負荷分散のためのWebサーバーの設定 」を参照してください
Read Timeout	<p>WebtoBは設定ファイルに定義されているsvrchktime変数値の間隔でWebエンジンにpingメッセージを送信します。</p> <p>Webエンジンは設定された時間間隔でWebtoBの状態をチェックします。設定時間内にWebtoBのピングが見つからない場合は、通信接続が切断されたと見なさ</p>

項目	説明
	れ、再設定されます。そのため、設定値はsvrchktime変数値より大きい必要があります

4. 「**Network Address**」と「**Domain Socket Address**」項目を相互排他的に設定します。1つの項目を設定すると別の項目は設定できません。

「**Network Address**」にWebtoBサーバーと接続するポート、IPアドレスを設定します。「**Network Address**」の設定を使用しない場合は、「**Domain Socket Address**」を設定します。これは、UNIXドメイン・ソケットまたはWindowsでHTHプロセスとIPC通信を使用するときに設定します。すなわち、WebtoBが同じマシンに存在する場合のみ有効です。

[図 2.23] WebtoBコネクターの追加および変更 - 接続設定

 **Network Address**
WebtoBのTCP/IPアドレス情報を設定します。

Port 	9900 WebtoBに接続するポートを指定します。このポート番号は、WebtoB設定ファイルのJSVPORT値と一致する必要があります。
Ip Address	192.168.1.8 [デフォルト: localhost] WebtoBのIPアドレスを指定します。

 **Domain Socket Address**
UNIXドメインソケット情報、あるいはWindowsでHTHプロセスとのIPC通信を行うための情報を設定します。WebtoBがWebエンジンと同じマシンにある場合は、UNIXドメインソケット(パイプ)を介して相互通信します。ただし、Windowsでは一般のソケット通信を行います。


WebtoB Ipcbaseport	[デフォルト: 6666] WindowsでWebtoB HTHプロセスとIPC通信を行うためのポートを指定します。この設定値はOS環境変数のWEBTOB_IPCBASEPORTをオーバーライドします。
WebtoB Home	WebtoBのホームディレクトリを指定します。この設定値はWebtoBホームディレクトリを示すOS環境変数(JEUS_WSDIRまたはWEBTOBDIR)をオーバーライドします。



5. 「**Thread Pool**」項目にWebtoBからのHTTP要求を処理するワーカー・スレッドについて設定します。

「**Thread State Notify**」で、リスナーに割り当てられたスレッド状態の自動通知に関する設定を行います。同設定はAJPリスナーと同様なので設定画面は「[2.3.2. AJPリスナーの設定](#)」を参照し、詳しい説明については、「[2.3.7. 自動スレッド・プール管理の設定\(スレッド状態の通知\)](#)」を参照してください。

以下は、設定項目についての説明です。

[図 2.24] WebtoBコネクターの追加および変更 - スレッド・プールの設定

 **Thread Pool**
WebtoBコネクタから要求を受信して処理するスレッドプールを指定します。

Number  	10 WJPコネクション数を設定します。設定した数だけWebtoBとのコネクションを作成するので、WebtoB設定のMinProc、MaxProc値を考慮して設定する必要があります。
--	--

項目	説明
Number	<p>基本的にWebtoBとの接続数を意味します。必ずWebtoB設定の「MinProc」と「MaxProc」値をベースにして設定してください。「MinProc」と「MaxProc」についての詳細内容は、「2.4.4. WebtoBとの負荷分散設定」を参照してください。</p> <p>[参考]</p> <p>JEUS 6まではワーカー・スレッドの最小値と最大値を設定しましたが、これらの設定値が実際の動作にはあまり影響を与えないため、ワーカー・スレッドの数を設定する「Number」に統一しました</p>

6. **詳細設定**の項目を設定します。

[図 2.25] WebtoBコネクターの追加および変更 - 詳細設定

詳細設定

すべてを開く

WebtoB Connector

Postdata Read Timeout

ms

Max Post Size

Max Parameter Count

Max Header Count

Max Header Size

byte

Max Querystring Size

byte

Request Prefetch

☐

Reconnect Interval

ms

Reconnect Count For Backup

Connection Type

▼

Send Buffer Size

byte

Receive Buffer Size

byte

Secure

☐

WebtoB Backup

☐

Compression

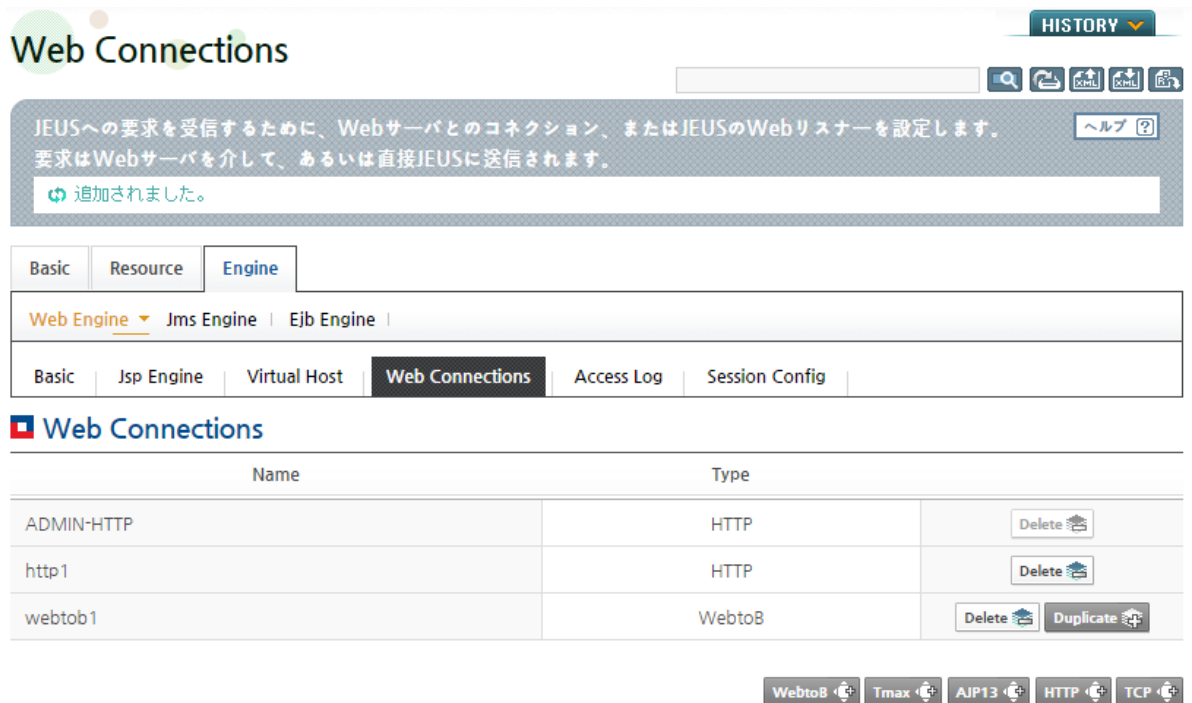
☐

以下は、設定項目についての説明です。以下で説明されていない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Request Prefetch	<p>Webエンジン側で要求を処理する間、WebtoBからの次の要求を事前に受けておくか否かについて設定します。</p> <p>この機能を有効にすると、WebエンジンはWebtoBワーカー・スレッドごとに1つのキューを割り当てます。キューはワーカー・スレッドが現在の要求を処理する間、WebtoBからの要求をバッファリングします。したがって、WebエンジンはWebtoBからの次の要求を受ける時間を短縮することができます。この機能のデメリットは、要求を処理する途中でWebエンジンに深刻な問題が生じた場合、キューに蓄積された要求を失う可能性があることです</p>
Reconnect Interval	<p>WebtoBとの接続が切断されると再接続をトライしますが、その再接続の時間間隔を設定します。1秒以上に設定します。(デフォルト値: 5000)</p> <p>接続が成功するまで設定した時間間隔で無限にトライします</p>
Reconnect Count For Backup	<p>WebtoBとの接続が切断されたとき、再接続へのトライ数です。</p> <p>設定値を超えて接続できない場合は、別のWebtoBに接続をトライします。プライマリーWebtoBで接続が切断されたらバックアップに、バックアップから切断された場合はプライマリーに接続します</p>
WebtoB Backup	<p>プライマリーWebtoBが障害状態になったと判断した場合、バックアップとして設定されたWebtoBに接続をトライします。一度バックアップの方に移されたら、バックアップWebtoBに接続されるまでトライし続けます。バックアップWebtoBからHTTP要求を受ける途中でプライマリーWebtoBが復旧すると、自動的にフェイルバックを実行します。</p> <p>こちらには、WebtoBコネクター設定の中で、WebtoBがインストールされるホストに応じて異なる部分の設定を提供します。</p> <p>スレッド・プール設定がない場合は、プライマリーWebtoB設定から継承します</p>

7. 設定後、**[確認]**ボタンをクリックすると、以下のとおり「webtob」タイプのコネクターが追加されたことが確認できます。

[図 2.26] WebtoBコネクターの追加および変更 - 追加の確認



8. 設定内容を適用するために**[Apply Changes]**ボタンをクリックすると、設定適用の結果メッセージが出力されます。([図 2.10]を参照)

● 削除

以下は、WebAdminを使用してWebtoBコネクターを削除する方法です。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Web Connections]**メニューを選択すると、Webコネクション・リストの照会画面へ移動します。([図 2.11]を参照)
2. 設定および設定変更のため、画面左側の**[LOCK & EDIT]**ボタンをクリックして設定変更モードに切り替えます。
3. 削除するコネクターの**[Delete]**ボタンをクリックします。
4. 選択したコネクターが削除されると、以下のとおり結果メッセージが出力されます。

[図 2.27] WebtoBコネクタの削除 - 削除の確認



5. 設定内容を適用するために[Apply Changes]ボタンをクリックすると、設定適用の結果メッセージが確認できます。([図 2.10]を参照)

コンソール・ツールの使用

以下は、コンソール・ツールを使用してWebtoBコネクタを追加、変更、削除する方法です。

• 追加

コンソール・ツールを使用してWebtoBコネクタを追加するには、**add-webtob-connector**コマンドを実行します。

コマンドについての詳細内容は、『JEUS リファレンスガイド』の「4.2.8.9. add-webtob-connector」を参照してください。

```
add-webtob-connector [-cluster <cluster-name> | -server <server-name>]
                    [-f,--forceLock]
                    -name <web-connection-name>
                    -num <thread-number>
                    -regid <registration-id>
                    [-ver <wjp-version>]
                    [-addr <server-address>]
                    [-port <server-port> | -dsocket]
                    [-wbhome <webtob-home> | -ipcport <ipc-base-port>]
```

```
[ -sndbuf <send-buffer-size> ]  
[ -rcvbuf <receive-buffer-size> ]
```

● 変更

コンソール・ツールを使用してWebtoBコネクターを変更するには、**modify-webtob-connector**コマンドを実行します。

コマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.8.20. modify-webtob-connector」を参照してください。

```
modify-webtob-connector [ -cluster <cluster-name> | -server <server-name> ]  
[ -f, --forceLock ]  
-name <web-connection-name>  
[ -num <thread-number> ] [ -obuf <output-buffer-size> ]  
[ -ver <wjp-version> ]  
[ -addr <server-address> ]  
[ -port <server-port> | -dsocket ]  
[ -wbhome <webtob-home> | -ipcport <ipc-base-port> ]  
[ -regid <registration-id> ]  
[ -sndbuf <send-buffer-size> ]  
[ -rcvbuf <receive-buffer-size> ]
```

● 削除

コンソール・ツールを使用してWebtoBコネクターを削除するには、**remove-webtob-connector**コマンドを実行します。

コマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.8.30. remove-webtob-connector」を参照してください。

```
remove-webtob-connector [ -cluster <cluster-name> | -server <server-name> ]  
<web-connection-name>
```

2.3.6. Tmaxコネクターの設定

TmaxコネクターもWebtoBコネクターと同様、コネクションを生成するとき、JEUSがクライアントの役割をします。そのため、Tmaxアドレスと接続ポートが必要です。WebAdminとコンソール・ツールを使用してTmaxコネクターを追加、設定、削除することができます。

WebAdminの使用

● 追加および変更

以下は、WebAdminを使用してTmaxコネクターを追加および設定する方法です。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[Engine] > [Web Engine] > [Web Connections]メニューを選択すると、Webコネクション・リストの照会画面へ移動します。([図 2.4]を参照)
2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. Tmaxコネクタを追加するときは[TMAX]ボタンをクリックし、変更する場合はWebコネクション・リストから「tmax」タイプのコネクタをクリックします。[Tmax Connector]画面に移動して基本情報を設定します。

[図 2.28] Tmaxコネクタの追加および変更 - 基本設定

Tmax Connector

HISTORY

Tmaxと連動するためのコネクタです。コネクションを作成するとき、JEUSがクライアントになります。

ヘルプ

Basic Resource **Engine**

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host **Web Connections** Access Log Session Config

動的設定 必須項目 確認 再設定

Name *	tmax1 リスナーまたはコネクタを識別できる一意の名前を指定します。
Output Buffer Size	byte out.write()メソッドにより出力されるJSP/Servlet出力を一時保存する出力バッファのサイズを指定します。バッファがいっぱいになると、自動でクライアントにデータを送信します。デフォルト値は、AJP13の場合は8192です。他のプロトコルの場合は、new Socket().getSendBufferSize()に従いますが、最小8192、最大16384の範囲になります。
Port *	9900 Tmaxサーバに接続するポート番号を指定します。
Server Group Name *	MyGroup 接続するTmaxサーバが含まれているグループを指定します。
Server Name *	Tmax1 接続するTmaxサーバの名前を指定します。
Dispatcher Config Class *	com.tmax.jeus.TmaxDispatcherConfig ディスパッチャの設定クラスを指定します。クラスは必ずクラスパスに設定された場所に存在し、クラス名は完全修飾クラス名である必要があります。
Read Timeout	ms [デフォルト: 0] Tmaxからの要求を読み込む際に待機する最大時間を設定します。
Tmax Address	 [デフォルト: localhost] TmaxサーバのIPアドレスを指定します。
Tmax Version	 接続するTmaxサーバのバージョンを指定します。4.0、3.xの2つのバージョンをサポートします。4.0バージョンは40、3.xバージョンは3Xに指定します。

以下は、設定項目についての説明です。以下で説明されていない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Port	Tmaxと接続するポートです。Tmax設定ファイルでのJEUS接続ポート値と一致する必要があります
Server Group Name	Tmaxと接続するとき、Tmaxのどのサーバーと接続するかを設定します。この設定は、接続するサーバーが属しているグループを示します
Server Name	接続するTmaxサーバー名を指定します
Dispatcher Config Class	Dispatcher Config Classを指定します。このときのクラスはクラス・パスに設定されている場所に存在する必要があり、クラス名はfully qualified class nameにします
Tmax Address	TmaxサーバーのIPアドレスを設定します
Tmax Version	接続するTmaxサーバーのバージョンを設定します

4. 「**Thread Pool**」項目の設定はAJPリスナーと同様のため、設定画面については「[2.3.2. AJPリスナーの設定](#)」を参照し、スレッド・プールの設定については、「[2.3.7. 自動スレッド・プール管理の設定\(スレッド状態の通知\)](#)」を参照してください。
5. **詳細設定**の項目を設定します。

[図 2.29] Tmaxコネクターの追加および変更 - 詳細設定

The screenshot shows the 'Tmax Connector' configuration page. The title bar includes '詳細設定' (Detailed Settings) and a link 'すべてを開く' (Expand All). The configuration fields are as follows:

- Postdata Read Timeout: [text input] ms
- Max Post Size: [text input]
- Max Parameter Count: [text input]
- Max Header Count: [text input]
- Max Header Size: [text input] byte
- Max Querystring Size: [text input] byte
- Connection Type: [dropdown menu]
- Reconnect Interval: [text input] ms
- Reconnect Count For Backup: [text input]
- Server Type: [text input]
- XA Resource Class: [text input]
- Tmax Backup Address: [text input]
- Tmax Backup Port: [text input]

以下は、設定項目についての説明です。以下で説明されていない項目については、「[2.3.1. リスナーの共通設定](#)」を参照してください。

項目	説明
Reconnect Count For Backup	Tmaxとの接続が切断されたとき、再接続をトライする回数です。常に1以上であり、バックアップ設定が存在する場合のみ使用します(デフォルト値: 12)
Reconnect Interval	Tmaxとの接続が切断されると再接続をトライしますが、その再接続の時間間隔を設定します。1秒以上を設定します。(デフォルト値: 5000) 接続が成功するまで設定した時間間隔で無限にトライします。ただし、バックアップが設定されている場合は、「 Reconnect Count For Backup 」設定値の分だけトライしてからフェイルオーバーを実行します
Server Type	接続するTmaxサーバーの種類を設定します
XA Resource Class	XAResourceクラス名を設定します
Tmax Backup Address	Tmaxに障害が発生した場合、バックアップとして動作するサーバーのアドレスです。アドレスとポートのみ設定し、以外はプライマリ・サーバーをそのまま使用します
Tmax Backup Port	Tmaxに障害が発生した場合、バックアップ・サーバーに接続するポート番号を設定します。「 Tmax Backup Address 」が設定されている場合は一緒に設定します

6. 設定後、**[確認]**ボタンをクリックすると、以下のとおり「tmax」タイプのコネクターが追加されたことが確認できます。

[図 2.30] Tmaxコネクターの追加および変更 - 追加の確認

The screenshot shows the 'Web Connections' configuration page. At the top, there's a 'Web Connections' header with a 'HISTORY' dropdown. Below it, a message box states '追加されました。' (Added). The main content area has tabs for 'Basic', 'Resource', and 'Engine'. Under 'Engine', there are sub-tabs for 'Web Engine', 'Jms Engine', and 'Ejb Engine'. The 'Web Engine' sub-tab is active, and within it, the 'Web Connections' tab is selected. The 'Web Connections' tab displays a table with the following data:

Name	Type	Action
ADMIN-HTTP	HTTP	Delete
http1	HTTP	Delete
tmax1	Tmax	Delete

At the bottom of the page, there are buttons for 'WebtoB', 'Tmax', 'AJP13', 'HTTP', and 'TCP'.

7. 設定内容を適用するために[Apply Changes]ボタンをクリックすると、設定適用の結果メッセージが出力されます。([図 2.10]を参照)

● 削除

以下は、WebAdminを使用してTmaxコネクタを削除する方法です。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[Engine] > [Web Engine] > [Web Connections]メニューをを選択すると、Webコネクション・リストの照会画面へ移動します。([図 2.11]を参照)
2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. 削除するコネクタの[Delete]ボタンをクリックします。
4. 選択したコネクタが削除されると、以下のとおり結果メッセージが出力されます。

[図 2.31] Tmaxコネクタの削除 - 削除の確認



5. 設定内容を適用するために[Activate Changes]ボタンをクリックすると、設定適用の結果メッセージが確認できます。([図 2.10]を参照)

コンソール・ツールの使用

以下は、コンソール・ツールを使用してTmaxコネクタを追加、変更、削除する方法です。

• 追加

コンソール・ツールを使用してTmaxコネクタを追加するには、**add-tmax-connector**コマンドを実行します。コマンドについての詳細内容は、『JEUS リファレンスガイド』の「4.2.8.6. add-tmax-connector」を参照してください。

```
add-tmax-connector [-cluster <cluster-name> | -server <server-name>]
                    [-f,--forceLock]
                    -name <web-connection-name>
                    -tmin <minimum-thread-num>
                    [-tmax <maximum-thread-num>]
                    -addr <server-address>
                    -port <server-port>
                    -svrg <server-group-name>
                    -svr <server-name>
                    -dcc <dispatcher-config-class>
```

• 変更

コンソール・ツールを使用してTmaxコネクタを変更するには、**modify-tmax-connector**コマンドを実行します。コマンドについての詳細内容は、『JEUS リファレンスガイド』の「4.2.8.16. modify-tmax-connector」を参照してください。

```
modify-tmax-connector [-cluster <cluster-name> | -server <server-name>]
                      [-f,--forceLock]
                      -name <web-connection-name>
                      [-tmin <minimum-thread-num>]
                      [-tmax <maximum-thread-num>]
                      [-tidle <max-idle-time>]
                      [-obuf <output-buffer-size>]
                      [-addr <server-address>]
                      [-port <server-port>]
                      [-svrg <server-group-name>]
                      [-svr <server-name>]
```

• 削除

コンソール・ツールを使用してTmaxコネクタを削除するには、**remove-tmax-connector**コマンドを実行します。コマンドについての詳細説明は、『JEUS リファレンスガイド』の「4.2.8.27. remove-tmax-connector」を参照してください。

```
remove-tmax-connector [-cluster <cluster-name> | -server <server-name> |  
                        -f,--forceLock]  
                        <web-connection-name>
```

2.3.7. 自動スレッド・プール管理の設定(スレッド状態の通知)

スレッド状態の通知は、Eメール通知やサーバーの再起動勧告のために必要なワーカー・スレッドの最小値と関連するエラー条件を定義します。リスナーおよびコネクタ設定画面の**Thread State Notify**項目で設定します。

自動スレッド・プール管理が設定されると、「**Max Thread Active Time**」設定値を基準にしてワーカー・スレッドが管理されます。

要求を受けて処理を開始した時点からMax Thread Active設定時間を超過したスレッドはブロックされたスレッドとして管理されます。また、ブロックされたスレッドでない一般のスレッド数がスレッド・プールで設定した最小値より小さい場合は、新しいワーカー・スレッドを生成して最小値が保持できるようにします。

ただし、WebtoBコネクタの場合は最小値がなく、全体の接続数値のみ存在するため、別のリスナーまたはコネクタとは動作が異なります。WebtoBコネクタはブロックされたスレッドと一般スレッド数の合計が設定した接続数値と一致する必要があります。なお、ブロックされたスレッドの要求処理が終了する時点で同スレッドは削除され、新しいワーカー・スレッドにWebtoBへの接続を要求します。

スレッド・プールのスレッド状態の変更に関する通知を設定することができます。

以下は、AJPリスナーにEメール通知子が含まれている設定例です。Eメール通知子を使用して受信するEメールの設定は、『*JEUS サーバガイド*』の「8.3. ロギングの設定」のSMTPハンドラー設定を参照してください。

[図 2.32] 自動スレッド・プール管理の設定 - Thread State Notifyの設定

▼ Thread State Notify

ワーカースレッドがブロックされる基準とブロックされたときに行うアクションを設定します。

Max Thread Active Time	<input type="text" value="150000"/> ms <small>【デフォルト: 0】 スレッドがブロックされたと見なす時間を指定します。スレッドのブロック状態の可否をチェックする周期は、〈monitoring〉〈check-thread-pool〉設定に従います。基本的にブロック状態のスレッドは捨てて、新しいスレッドを作成してスレッドプールに含めます。ただし、WebtoBコネクタとTmaxコネクタに属するスレッドプールについては何のアクションも行わずに、ただブロック状態を表示します。</small>
Notify Threshold Ratio	<input type="text" value="0.8"/> <small>【デフォルト: 0.0】 メール通知の基準を設定します。スレッドプールの最大値に対するブロックされたスレッドの割合が指定の値以上に増えた場合に、Eメールで通知します。</small>
Restart Threshold Ratio	<input type="text" value="0.9"/> <small>【デフォルト: 0.0】 Webエンジンの再起動を勧める基準となるブロックスレッドの割合を指定します。1より小さい小数の値を指定します。ブロックされたスレッドが指定の割合に達すると、サーバーの再起動を勧めるメッセージが表示されます。</small>
Notify Subject	<input type="text"/> <small>Notify Threshold Ratio設定によって通知されるEメールのタイトルを記述します。</small>
Restart Subject	<input type="text"/> <small>Restart Threshold Ratio設定によって通知されるEメールのタイトルを記述します。</small>
Interrupt Thread	<input checked="" type="checkbox"/> <small>【デフォルト: false】 ブロックされたスレッドをインタラプトするか否かを設定します。インタラプトは、スレッドに実行している動作を取り消すようにヒントを与えるだけです。スレッド内部でインタラプト状態をチェックしていない場合は、何の影響も与えません。</small>
Active Timeout Notification	<input checked="" type="checkbox"/> <small>【デフォルト: false】 ブロックされたスレッドが発生したことをEメールで通知するか否かを設定します。</small>

以下は、各項目についての説明です。

項目	説明
Max Thread Active Time	<p>ブロックされる前までワーカー・スレッドが使用できる最大時間を設定します。</p> <p>この時間は、ワーカー・スレッドがクライアント要求をサービスする時点から測定されます(サブレットが実行される時点)。設定した時間を超えるか、スレッドが自由になりスレッド・プールに戻るときに満了します。(スレッドがブロックされない状況)</p> <p>設定値が0か、負の数の場合は無視されます</p>
Notify Threshold Ratio	<p>存在できるブロックされたスレッドの最大割合を設定します。ブロック・スレッド数が全スレッド数を超えるとエラーと判断し、Eメールで通知されます。</p> <p>1未満の小数点で設定し、0か負の数の場合は無視されます</p>
Restart Threshold Ratio	<p>存在できるブロックされたスレッドの最大割合を設定します。ブロック・スレッド数が全スレッド数を超えるとエラーと判断し、Eメールで通知されます。その後、サーバー・ログにサーバー再起動の勧告メッセージが記録されます。</p> <p>1未満の小数点で設定し、0か負の数の場合は無視されます</p>
Notify Subject	<p>警告Eメールを転送するときに使われます</p>

項目	説明
Restart Subject	エンジン・ログおよび通知するときに使用されるEメールに使われます
Interrupt Thread	Active Timeoutが発生するとき、すなわち、ワーカー・スレッドが「 Max Thread Active Time 」設定値以上に要求を処理している場合、当該スレッドをインタラプトするか否かを設定します。(デフォルト値: false) trueの場合、インタラプトを発生させます
Active Timeout Notification	Active Timeoutが発生したとき、Eメールを転送するか否かを設定します(デフォルト値: false) trueの場合、Eメールが転送されます

2.4. 負荷分散のためのWebサーバーの設定

本節では、JEUSとの連動事例が多いWebサーバーの設定方法およびWebエンジンをWebコネクションと連動できる方法について説明します。

WebエンジンはシステムのHTTP処理の性能を高めるため、WebサーバーとWebエンジン(サーブレット・エンジン)で構成してサービスできます。

各WebエンジンへのHTTP要求を、Webエンジンの前のWebサーバーで1次的に負荷を分散させます。以降、同じ接続やセッションの要求に対し、HTTPセッション・クラスタリング・サービスを利用して最初の要求を処理したWebエンジンに割り当てし、サービス処理の効率を高めます。

最初の要求を処理したWebエンジンに障害が発生した場合、障害発生以降に受信された要求はWebエンジンの前のWebサーバーで障害が発生していないWebエンジンに渡し、無停止サービスを可能にします。このサービスはHTTPセッション・クラスタリングを使用するとの前提条件が必要です。HTTPセッション・クラスタリングについての詳細内容は、『*JEUS セッション管理ガイド*』の「第2章 分散セッション・サーバー」を参照してください。

Webサーバーを使用せずにWebエンジンのみ使用する場合、要求を処理したWebエンジンに以降の要求が転送できる装備やソフトウェアを使用すればWebサーバーを使用するときと同様なサービスが可能になるかもしれませんが、JEUSではこれに必要なソフトウェアは提供しておらず、使用もお勧めしません。

注意

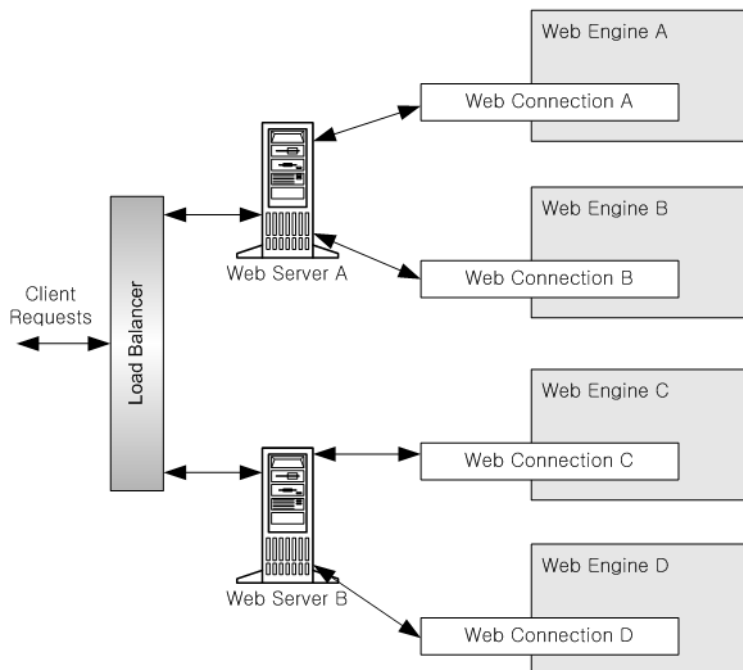
各Webサーバーの設定やmod_jk設定は、Webサーバーのバージョンによって異なる場合があります。本節の設定方法は、JEUSユーザーの理解を助けるために提供するものであり、**実際に設定するときは、必ず各Webサーバーの関連文書、国内外のコミュニティーで提供する設定事例を参考にしてください。**

2.4.1. 負荷分散の構造

サービス要求が多いWebサイトは、1台のWebサーバーとWebエンジンではサービスを提供することが困難なため、負荷分散のために複数のWebサーバーとWebエンジンが必要です。

以下は、2台のWebサーバーと4台のWebエンジンが接続されている負荷分散構造を表しています。

[図 2.33] 小規模のWebサイトにおける負荷分散構造



各Webエンジンには同じWebコンテキストがデプロイされている必要があります。このような環境において重要なのはセッションの共有可否です。アプリケーションをより簡単に管理したい場合や分散セッション・サービスが必要な場合は、Webエンジンを1つのクラスターにする必要があります。セッション・クラスターについての詳しい内容は、『JEUS セッション管理ガイド』の「第1章 セッション・トラッキング」を参照してください。

2.4.2. Apache Webサーバー

ApacheとWebエンジンの連動は、以下のような手順で実行します。

1. Apacheにmod_jkライブラリーをインストールします。
2. JEUS WebエンジンにAJP13リスナーを設定します。
3. workers.propertiesファイルを生成、編集します。
4. Apache Webサーバーのhttpd.confファイルに連動セクションを挿入します。
5. Apache Webサーバーを再起動します。

参考

Apacheの場合、2.2.4, mod_jkの場合は1.2.20を基準にして説明します。

mod_jkライブラリーのインストール

Apache Webサーバーを前端で使用するには、「mod_jk」というモジュールをApacheインストール・モジュールに追加する必要があります。「mod_jk」モジュールはサーバーとエンジン間の通信プロトコルを実装したものであり、このプロトコルはApache JServ Protocol 1.3(AJP 1.3)です。

参考

Windowsバイナリーを提供している「<http://tomcat.apache.org/download-connectors.cgi>」よりソースをダウンロードして該当するマシンでコンパイルします。

AJP13リスナーの設定

mod_jkライブラリーのインストールが完了したら、JEUS WebエンジンにAJP13リスナーを設定します。AJP13リスナーの設定方法についての詳細内容は、「[2.3.2. AJPリスナーの設定](#)」を参照してください。

workers.propertiesの設定

workers.propertiesファイルは、mod_jkのための設定ファイルです。

たとえば、JEUSにはserver1、server2という2台のサーバーがあります。ホスト名はそれぞれserver1、server2とし、各Webエンジンに設定されたAJPリスナーは9901、9902に設定したと想定します。

以下は、仮定したJEUS環境におけるworkers.propertiesの例です。

[例 2.1] mod_jk設定ファイルの例 : <<workers.properties>>

```
worker.list=jeus_load_balancer_workers
worker.jeus_load_balancer_workers.type=lb
worker.jeus_load_balancer_workers.sticky_session=true

#####
# listener specific configuration
#####
worker.jeus_load_balancer_workers.balance_workers=server1
worker.server1.reference=worker.template
worker.server1.host=192.168.0.101
worker.server1.port=9901
worker.server1.route=ZG9tYWluMS9zZXJ2MQ==

worker.jeus_load_balancer_workers.balance_workers=server2
worker.server1.reference=worker.template
```

```

worker.server1.host=192.168.0.102
worker.server1.port=9902
worker.server1.route=ZG9tYWluMS9zZXJ2Mg==

#####
# common config
#####
worker.template.type=ajp13
worker.template.socket_connect_timeout=5000
worker.template.socket_keepalive=true
worker.template.ping_mode=A
worker.template.ping_timeout=10000
worker.template.connection_pool_minsize=0
worker.template.connection_pool_timeout=600
worker.template.reply_timeout=300000
worker.template.recovery_options=3

```

workers.propertiesファイルは一部の定義を除いて、一般的に「worker.<worker_name>.<directive>=<value>」形式で定義します。

以下は、重要設定についての説明です。

- **worker.list**

ワーカーを定義する項目です。ワーカーの名前はコンマ(,)で区切られ、複数のワーカーが羅列できます。

以下は、「jeus_load_balancer_workers」という名前のワーカーを定義した例です。

```
worker.list=jeus_load_balancer_workers
```

- **worker.<worker_name>.type**

ワーカーのタイプを定義します。「ajp13」、「ajp14」、「jni」、「lb」、「status」などが選択できます。AJP13をサポートするため、JEUSでは「ajp13」、「lb」のみの設定で十分です。

以下は、「jeus_load_balancer_workers」をロード・バランサー・タイプで定義した例です。

```
worker.jeus_load_balancer_workers.type=lb
```

- **worker.<worker_name>.balance_workers**

コンマ(,)で区切り、ロード・バランスが必要なワーカーを羅列することができます。上記のworker.listに記述したワーカーが表示されてはなりません。

JEUSのAJP13リスナーと連動する場合、正しいロード・バランサーおよびスティッキー・セッションの役割をするには、ワーカーの名前をJEUSのサーブレット・エンジン名にする必要があります。その理由は、JEUS

の場合、セッションIDのルーティング情報でサーブレット・エンジン名を使用し、mod_jkではワーカーの名前を使用するためです。

- **worker.<worker_name>.sticky_session**

セッションIDをベースにしてルーティングをサポートするか否かを設定します。true(or 1)またはfalse(or 0)に設定できます。JEUSではスティッキー・セッションをデフォルトでサポートするため、常にtrueに設定します。(デフォルト値: true)

以下は、「jeus_load_balancer_workers」がスティッキー・セッションをサポートするとの例です。

```
worker.jeus_load_balancer_workers.sticky_session=true
```

- **worker.<worker_name>.host**

JEUSのAJP13リスナーが存在するホスト名またはIPアドレスを入力します。

- **worker.<worker_name>.port**

JEUSのAJP13リスナーのポート番号を設定します。

- **worker.<worker_name>.reference**

複数のロード・バランサー・ワーカーを設定するときには有用であり、指定された値に該当するワーカーの設定値が参照できる設定です。

たとえば、「worker.castor.reference=worker.template」と設定した場合、castor workerで設定した値を除いて、すべてのworker.templateの設定を継承します。

- **worker.<worker_name>.route**

設定値に該当する値の要求がスティッキーに送られた場合、該当のワーカーが処理する設定です。

JEUSのセッション・マネージャーで生成するセッションに付けるセッション・ルーティングの記述を利用して該当するワーカーをマッチさせ、ローカル・セッションの活用度を高めることができます。セッション・ルーティングについての詳細内容は、『JEUS セッション管理ガイド』の「1.2. セッション・トラッキングの構造」を参照してください。

当該値は、domain_name/server_nameのBase64アルゴリズムでエンコーディングした値です。同エンコーディング値は、JEUS_HOME/binのencryptionコマンドを使用して取得できます。

以下は、使用例です。

```
${JEUS_HOME}/bin/encryption -algorithm base64 -text domain1/server1
```

設定時には「domain1/server1」、「domain1/server2」のように「ドメイン名/サーバー名」を使用することに注意します。

注意

この設定は、mod_jkバージョンによってdeprecatedされる場合もあるため、本節で提示した内容は参考までにしてください。必ずTomcatホームページのmod_jk関連の説明に従って設定します。

httpd.confの設定

mod_jkモジュールを使用するために、httpd.confファイルに以下の事項を追加します。

[例 2.2] Apacheにmod_jkを設定する例 : <<httpd.conf>>

```
. . .
LoadModule      jk_module  "/usr/local/apache/modules/mod_jk.so"
JkWorkersFile   "/usr/local/apache/conf/workers.properties"
JkLogFile       "/usr/local/apache/logs/mod_jk.log"
JkLogLevel      info
JkMount  /examples/*  jeus_load_balancer_workers
. . .
```

注意

Apacheのバージョンによってdeprecatedされる場合もあるため、本節で提示した内容は参考までにしてください。必ずApacheのマニュアルに従って設定します。

2.4.3. IIS Webサーバーおよびiplanet Webサーバーの設定

IIS Webサーバーとiplanet WebサーバーもAJP13プロトコルをサポートします。したがって、workers.propertiesとJEUSの設定方式はすべて同様です。

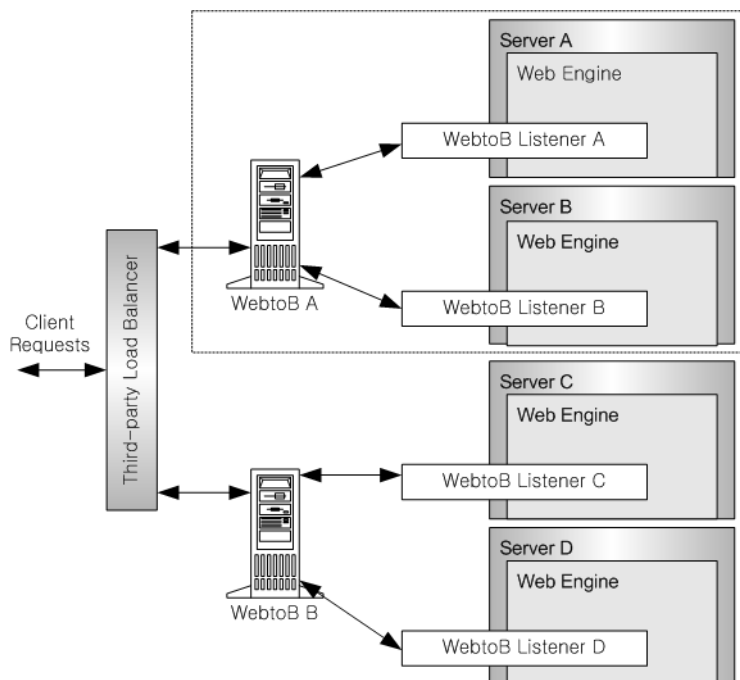
ただし、製品別にmod_jkモジュールをインストールする方法やmod_jkの使用を設定する方法が異なります。このような設定方法は、各Webサーバーが提供するマニュアルを参照してください。

2.4.4. WebtoBとの負荷分散設定

本節では、例を利用してWebtoBとJEUSの負荷分散の処理環境を構成します。

以下は、サーバー構造の例です。各エンジンがそれぞれのWebtoBサーバーを有する構造についての設定として、例では2台のWebtoBが2台のWebエンジンにそれぞれ接続されます。

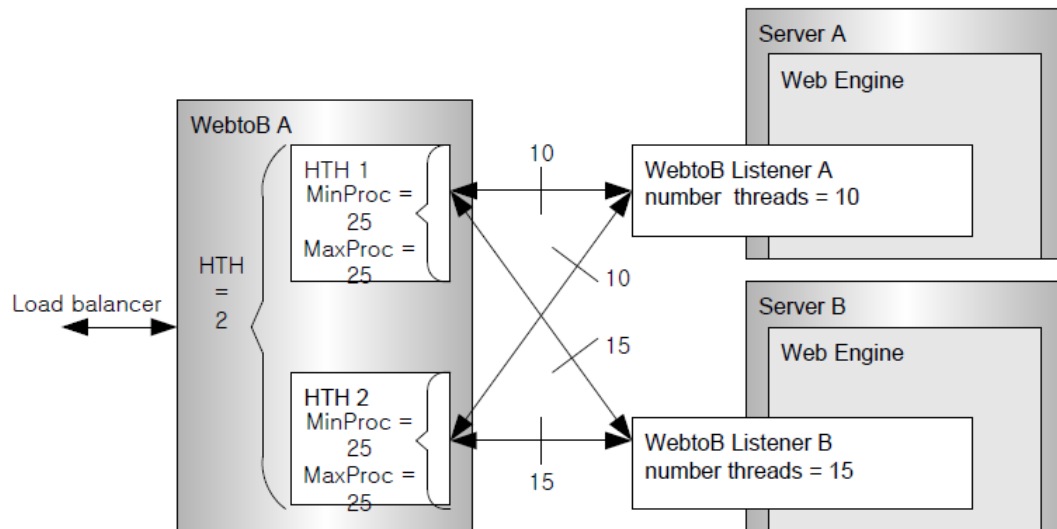
[図 2.34] 負荷分散サーバーの構造 - 2台のWebtoBが2台のWebエンジンにそれぞれ接続された場合



Server AからDには同様なWebコンテキストをデプロイします。上記のような構成では、一般的にServer AからDが1つのクラスターになり、それによって分散型セッション・サーバーを使用することになります。ユーザー・セッションが不要な場合は、クラスターを構成しなくても構いません。

以下は、WebtoB AとServer A、Server Bが接続されている状況をさらに詳しく表現した図です。それぞれのWebtoBコネクター設定の状態を示しています。

【図 2.35】 WebtoBコネクター設定の状態



設定時には、WebtoBのHTHプロセス数を考慮します。

WebtoB HTHプロセスは、いくつかの下位プロセスを持っているエンジンのように動作します。これは、WebtoBコネクターのワーカー・スレッドと1対1で接続されます。そのため、WebtoBコネクターの「**Worker Thread Pool Max**」設定値とWebtoB設定の「**MaxProc**」値を注意して設定する必要があります。

HTHプロセスの数は、WebtoB設定に存在するHTHプロセス数をそのままWebtoBコネクターの「**Hth Count**」設定に反映します。WebtoBの「**MaxProc**」値とWebtoBコネクターの「**Worker Thread Pool Max**」値は、以下の式で表現できます。

```

MinProc = Listener A number threads + Listener B number threads + ... + Listener X
number threads setting.
MaxProc = Listener A number threads + Listener B number threads + ... + Listener X
number threads setting.

```

WebtoBコネクターの「**Hth Count**」値は、WebtoB設定ファイルのNODEセクション要素のうち、HTHプロセス数と同じである必要があります。WebtoBコネクターの設定については、[2.3.5. WebtoBコネクターの設定](#)を参照してください。

以下は、WebtoB Aが上記例のように設定されるためのhttp.mファイルの設定例です。

【例 2.3】 WebtoBの設定例 : <<http.m>>

```

*NODE
foo      ...
    HTH = 2,
    JSVPORT = 9900,
    ...
*SVRGROUP
jsvg     NODENAME = foo, SVRTYPE = JSV

```

```
*SERVER
default      SVGNAME = jsvg, MinProc = 6, MaxProc = 25
*URI
uri1  Uri = "/examples/", Svrtype = JSV
uri2  Uri = "/test/", Svrtype = JSV
*EXT
jsp   MimeType = "application/jsp", SvrType = JSV
```

minとmax値の設定に注意します。minとmax値は以下のように設定されました。

```
HTH MinProc = 6 = 2 thread + 4 thread, HTH MaxProc = 25 = 10 thread + 15 thread
```

2.5. TCPリスナーの使用

TCPリスナーは通信プロトコルを直接定義してTCPクライアントとTCPサーブレット間の通信が可能です。ただし、HTTPまたはHTTPSプロトコルで満足できない事項がある場合のみ、TCPリスナーを使用することをお勧めします。

TCPリスナーを使用するには、以下の手順を実行します。

1. カスタマイズされた通信プロトコルを定義します。
2. Dispatcher Config Classを実装します。(プロトコルの設定情報)
3. TCPハンドラー・サーブレットを実装します。(プロトコルの実装)
4. カスタマイズされたプロトコルを実装するためのTCPリスナーを設定します。
5. TCPクライアントを実装します。
6. TCPクライアントをコンパイル、動作します。

以下は、本節で使用する用語についての説明です。

用語	説明
プロトコル(通信プロトコル)	TCPクライアントとTCPハンドラーとの間で(TCPリスナーがメディエーターの役割をする)送受信されるメッセージの構造および内容を定義します
メッセージ	TCPクライアントとTCPハンドラーとの間で送受信されるデータです。プロトコルで定義した構造に適する必要があります
TCPクライアント	TCPリスナーと通信し、メッセージをやり取りする外部のアプリケーション(TCPハンドラーによって処理されます。)です。メッセージをやり取りする際には標準ソケットを使用します

用語	説明
TCPハンドラー	<p>TCPリスナーを介してメッセージを受信し、実装された方法どおりメッセージを処理します。</p> <p>TCPハンドラーはjeus.servlet.tcp.TCPサーブレットの下位クラスの形式で実装され、Webエンジンに一般的なサーブレットのように登録されます。TCPハンドラーは、まるでTCPプロトコル上に存在するカスタマイズされたプロトコルの「プロバイダー」または「実装クラス」のように動作します</p>
TCP Dispatcher Configuration Class	<p>TCP Dispatcher Configuration Classは、jeus.servlet.tcp.TCPDispatcherConfigを拡張したクラスです。同クラスは、カスタマイズされたプロトコルに関する情報をTCPリスナー渡し、この非HTTPメッセージを解釈して処理します。</p> <p>このクラスはDOMAIN_HOME/lib/applicationディレクトリーの下位に格納され、Webエンジン設定下位のTCPリスナー設定の「Dispatcher Config Class」項目に設定します(「2.3.4. TCPリスナーの設定」を参照)</p>
TCPリスナー	<p>カスタマイズされたメッセージの解釈およびルーティング・インフラを提供します。TCPクライアントとTCPハンドラーとの間で非HTTPメディエーターの役割をします。</p> <p>別のHTTPリスナーと同様、Webエンジン内に存在します</p>

2.5.1. カスタマイズされた通信プロトコルの定義

TCPリスナーは、TCPクライアントとTCPハンドラーとの間で通信されるすべてのメッセージをヘッダーとボディに分けます。一般的にヘッダーは基本的であり、かつ標準情報を伝達し、サイズも決められています。一方、ボディには転送される任意のユーザー・データが含まれます。(例: HTTP応答のHTMLコード)

TCPリスナーの使用を説明するために簡単なプロトコル(メッセージ構造)を定義します。

以下は、カスタマイズされた通信プロトコル(カスタマイズされたメッセージ構造)についての説明です。

● ヘッダー

- ヘッダーは、4バイトの**magic**ナンバーで始まります。これは使用されるプロトコルを識別します。「7777」に設定します。
- 1バイトの**type**フィールドがmagicナンバーの次に使われます。「0」は要求を、「1」は応答を意味します。
- 3番目の項目は、4バイトの**body length**です。この項目は、メッセージのボディ部分のバイト数を記録します。以下の例では、その値で要求を処理するTCPServletの名前を入力します。
- 最後の項目は、32バイトの文字列で**service name**を記録します。以下の例では、この値で要求を処理するTCPサーブレット名を入力します。

- ボディ

メッセージのボディ部分は、128バイトの文字データを入れます。以下の例では、128バイトのブロックを任意の2つの64バイトのテキスト文字列で入力します。

2.5.2. Dispatcher Config Classの実装

Dispatcher Config Classは、jeus.servlet.tcp.TCPDispatcherConfig Classの下位クラスです。このクラスのabstractメソッドには、TCPリスナーがTCPハンドラーにメッセージを転送するために必要な様々な情報が実装される必要があります。同メソッドについては、JEUS_HOME/docs/api/jeusapi/index.htmlを参照してください。

JEUS 7 Fix#1からは、getBodyLengthLongメソッドが追加されました。TCPボディのサイズが2GB以上の場合は、当該メソッドを使用してバイト長を8バイトで表現します。

以下は、定義されたプロトコルに基づいてDispatcher Config Classを実装した例です。メソッドの使用方法についてはコメントを参考にしてください。

[例 2.4] TCPDispatcherConfigの実装例 : <<SampleConfig.java>>

```
package sample.config;

import javax.servlet.*;
import javax.servlet.http.*;
import jeus.servlet.tcp.*;

/*
 * This class extends the abstract class jeus.servlet.tcp.
 * TCPDispatcherConfig class. This implementation provides
 * routing and handling information to the TCP listener
 * according to our defined communications protocol.
 */
public class SampleConfig extends TCPDispatcherConfig {
    /*
     * Any init code goes into this method. This method will be
     * Called once after starting the TCP listener.
     * We leave this method empty for our simple example.
     */
    public void init() {}

    /*
     * This method returns the fixed-length of the header so
     * that the TCP listener knows when to stop
     * reading the header. The header length for
     * our example is 41 (bytes): 4 (magic) + 1
     * (type) + 4 (body length) + 32 (service name).
     */
}
```

```

*/
public int getHeaderLength() {
    return 41;
}

/*
    This method must return the length of the body.
    For our example, this length is represented as an
    "int" in the header, starting at position "5"
    (see the protocol definition above).
*/
public int getBodyLength(byte[] header) {
    return getInt(header, 5);
}

/**)
 * Returns the long-size length of request body of
 * incoming request packet.
 * If you don't need to support long, you may map to
 * {@link #getBodyLength(byte[])}.
 */
public long getBodyLengthLong(byte[] header) {
    return getBodyLength(header);
}

/*
    This method must return the context path so that the
    request can be routed by the TCP listener to the context
    that contains the TCP handler (TCPServlet implementation).
    For our example, we always use the context path "/tcpctest".
*/
public String getContextPath(byte[] header) {
    return "/tcpctest";
}

/*
    This method must return the name (path) of the TCP
    handler(TCPServlet) relative to the context path.
    For our example, we fetch this name from the 9th position
    in the header.
*/
public String getServletPath(byte[] header) {
    return "/" + getString(header, 9, 32);
}

/*
    This method returns some path info from the header.

```

```

        It is not used in our example and thus returns "null".
    */
    public String getPathInfo(byte[] header) {
        return null;
    }

    /**
     * This method returns any session ID embedded in the header.
     * It is not used in our example and thus returns "null".
    */
    public String getSessionId(byte[] header) {
        return null;
    }

    /**
     * This method determines whether the TCP listener
     * should keep the socket connection open after the TCP handler
     * has delivered its response. If it returns "false", the
     * connection will be dropped like in HTTP communications.
     * If it returns "true" the connection will be kept open
     * like in the Telnet or FTP protocols. For our example,
     * we choose to make it persistent (connection not closed
     * by the TCP listener).
    */
    public boolean isPersistentConnection() {
        return true;
    }
}

```

2.5.3. TCPサーブレットの実装

TCPサーブレットは、`jeus.servlet.tcp.TCPServlet`の下位クラスです。こちらには常に上書き(overridden)される`abstract void service(TCPServletRequest req, TCPServletResponse res)`メソッドが存在します。

サービス・メソッドは、カスタマイズされたプロトコルに準ずるメッセージが処理できるように実装します。Webコンテナはヘッダーを読み込んで`TCPServletRequest`オブジェクトに渡し、TCPサーブレットでは`TCPServletResponse`オブジェクトのアウトプット・ストリームに応答を書き込みます。

以下の例は、カスタマイズされたプロトコルに準ずるメッセージを処理するTCPサーブレットの実装コードです。

[例 2.5] TCPサーブレットの実装例 : <<SampleServlet.java>>

```

package sample.servlet;

import java.io.*;

```

```

import javax.servlet.*;
import javax.servlet.http.*;
import jeus.servlet.tcp.*;

/**
 * Sample TCPServlet implementation
 *
 * Protocol scheme:
 *
 * common header (for request and response) : total 41 byte
 *
 * magic field: length = 4 byte, value = 7777
 * type field : length = 1 byte, 0 : request, 1:response
 * body length field : length = 4, value = 128
 * service name field : length = 32
 *
 * request and response body
 *
 * message1 field : length = 64
 * message2 field : length = 64
 */

public class SampleTCPServlet extends TCPServlet {

    public void init(ServletConfig config) throws ServletException {
    }

    public void service(TCPServletRequest req, TCPServletResponse res)
        throws ServletException, IOException {
        ServletContext context = req.getServletContext();
        byte[] header = req.getHeader();
        byte[] body = new byte[req.getContentLength()];
        int read = req.getInputStream().read(body);
        if (read < body.length) {
            throw new IOException("The client sent the wrong content.");
        }

        String encoding = res.getCharacterEncoding();
        if (encoding == null)
            encoding = "euc-kr";

        DataInputStream in = new DataInputStream(new ByteArrayInputStream(header));

        int magic = in.readInt();
        context.log("[SampleTCPServlet] received magic = " + magic);
    }
}

```

```

byte type = (byte)in.read();
context.log("[SampleTCPServlet] received type = " + type);

int len = in.readInt();
context.log("[SampleTCPServlet] received body length = " + len);

byte[] svcname = new byte[32];
in.readFully(svcname);
context.log("[SampleTCPServlet] received service name = "
    + (new String(svcname)).trim());

String rcvmsg = null;
rcvmsg = (new String(body, 0, 64)).trim();
context.log("[SampleTCPServlet] received msg1 = " + rcvmsg);

try {
    rcvmsg = (new String(body, 64, 64, encoding)).trim();
} catch (Exception e) {}
context.log("[SampleTCPServlet] received msg2 = " + rcvmsg);

String msg1 = "test response";
String msg2 = "test response2";

byte[] result1 = null;
byte[] result2 = null;
if (encoding != null) {
    try {
        result1 = msg1.getBytes(encoding);
        result2 = msg2.getBytes(encoding);
    } catch (UnsupportedEncodingException uee) {
        result1 = msg1.getBytes();
        result2 = msg2.getBytes();
    }
} else {
    result1 = msg1.getBytes();
    result2 = msg2.getBytes();
}

header[4] = (byte)1; // mark as response

ServletOutputStream out = res.getOutputStream();
out.write(header);

byte[] buf1 = new byte[64];
System.arraycopy(result1, 0, buf1, 0, result1.length);
out.write(buf1);

```

```

        byte[] buf2 = new byte[64];
        System.arraycopy(result2, 0, buf2, 0, result2.length);
        out.write(buf2);

        out.flush();
    }

    public void destroy() {
    }
}

```

参考

JEUS 7 Fix#2からはTCPServletRequest.getBody()メソッドの使用をお勧めしません。このメソッドは、要求ボディが非常に大きい場合、メモリー問題を起こす可能性があります。そのため、サーブレット標準に定義されたServletInputStreamを使用することを推奨します。これは、TCPServletRequest.getInputStream()で取得できます。

2.5.4. カスタマイズされたプロトコルのためのTCPリスナーの設定

実装したコード(SampleConfig.javaとSampleTCPServlet.java)をベースにしてTCPリスナーを設定します。設定手順は以下のとおりです。

1. WebエンジンにTCPリスナー情報を設定します。詳細内容については、「[2.3.4. TCPリスナーの設定](#)」を参照してください。ただし、TCPリスナーで参照するサーバー・リスナーのポートは「5555」、「Dispatcher Config Class」は「sample.config.SampleConfig」を入力します。
2. JEUSサーバーを開始し、上記で作成したTCPサーブレットが含まれたWebアプリケーションをデプロイします。WebアプリケーションのDDの例は以下のとおりです。

[例 2.6] TCPハンドラーのDD : <<web.xml>>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <display-name>test</display-name>
    <distributable/>
    <servlet>
        <servlet-name>SampleServlet</servlet-name>
        <servlet-class>sample.servlet.SampleServlet</servlet-class>
        <load-on-startup>0</load-on-startup>
        <async-supported>true</async-supported>
    </servlet>

```

```

<servlet-mapping>
    <servlet-name>SampleServlet</servlet-name>
    <url-pattern>/sample</url-pattern>
</servlet-mapping>
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
</web-app>

```

2.5.5. TCPクライアントの実装

TCPクライアントはTCPリスナーとソケットの接続を確立し、これを利用してメッセージを転送します。このメッセージは、[「2.5.1. カスタマイズされた通信プロトコルの定義」](#)で定義したカスタマイズされた通信プロトコルに準ずるバイト・ストリームです。

設定されたTCPリスナーはメッセージを受け、SampleConfigクラスに定義されたディスパッチ情報をベースにしてSampleTCPServletのservice()メソッドを呼び出します。SampleTCPServletはクライアントから渡されたデータをベースにして応答を作成して転送します。この応答をクライアントが受け、System.outに出力します。

以下は、TCPクライアントの実装例です。

[例 2.7] TCPクライアントの実装例 : <<Client.java>>

```

package sample.client;

import java.io.*;
import java.net.*;

public class Client {
    private String address;
    private int port;

    private int magic = 7777;
    private byte type = 0;
    private int bodyLength = 128;
    private byte[] serviceName="sample".getBytes();

    public Client(String host, int port) {
        this.address = host;
        this.port = port;
    }

    public void test()
        throws IOException, UnsupportedEncodingException {

```

```

Socket socket = new Socket(address, port);
DataOutputStream out = new DataOutputStream(
    new BufferedOutputStream(socket.getOutputStream()));
DataInputStream in = new DataInputStream(
    new BufferedInputStream(socket.getInputStream()));

out.writeInt(7777);
out.write(type);
out.writeInt(bodyLength);
byte[] buf = new byte[32];
System.arraycopy(serviceName, 0, buf, 0, serviceName.length);
out.write(buf);
byte[] msg1 = "test request".getBytes();
byte[] msg2 = "test request2".getBytes();
buf = new byte[64];
System.arraycopy(msg1, 0, buf, 0, msg1.length);
out.write(buf);
buf = new byte[64];
System.arraycopy(msg2, 0, buf, 0, msg2.length);
out.write(buf);

out.flush();

// rx msg
int magic = in.readInt();
System.out.println("[Client] received magic = " + magic);

byte type = (byte)in.read();
System.out.println("[Client] received type = " + type);

int len = in.readInt();
System.out.println("[Client] received body length = " + len);

byte[] svcname = new byte[32];
in.readFully(svcname);
System.out.println("[Client] received service name = " +
    (new String(svcname)).trim());

byte[] body = new byte[128];
in.readFully(body);
String rcvmsg = null;
rcvmsg = (new String(body, 0, 64)).trim();
System.out.println("[Client] received msg1 = " + rcvmsg);
rcvmsg = (new String(body, 64, 64, "euc-kr")).trim();
System.out.println("[Client] received msg2 = " + rcvmsg);

out.close();

```



```

        in.close();
        socket.close();
    }

    public static void main(String[] argv) throws Exception {
        Client client = new Client("localhost", 5555);
        client.test();
    }
}

```

上記のクライアント・コードでプロトコルに必要な多様なヘッダーのフィールド設定を注意深く確認します。

「magic」番号は「7777」、「type」は「0」(要求)、「body length」は「128バイト」(固定長)、「service name」は「sample」に設定されています。(SampleServletの名前はweb.xmlに設定されています。) 次に、2つのメッセージを作成してヘッダー情報を送信した後、TCPリスナーにそれらを転送します。最後に「hostname」を「localhost」に、ポート番号は「5555」に設定しました。

2.5.6. TCPクライアントのコンパイルおよび実行

TCPリスナーが設定され、「localhost」に「5555」ポートを使用して運用されていると仮定します。

このような環境において、TCPクライアントをコンパイルおよび実装する手順は以下のとおりです。

1. Client.javaをコンパイルします。

```
javac -classpath ${JEUS_HOME}/lib/system/jeus.jar -d . Client.java
```

2. Client.classを以下のとおり実行します。

```
java -classpath ${JEUS_HOME}/lib/system/jeus.jar:.sample.client.Client
```

3. JEUS管理者のコンソール・ウィンドウには、以下ようにTCPハンドラーの結果値が表示される必要があります。(SampleServletクラス)

```

[SampleServlet] received magic = 7777
[SampleServlet] received type = 0
[SampleServlet] received body length = 128
[SampleServlet] received service name = sample
[SampleServlet] received msg1 = test request
[SampleServlet] received msg2 = test request2

```

4. 以下の内容がクライアントの実行画面に表示されます。

```

[Client] received magic = 7777
[Client] received type = 1
[Client] received body length = 128
[Client] received service name = sample

```

```
[Client] received msg1 = test response
[Client] received msg2 = test response2
```

2.6. HTTP/2の使用

HTTP/2は、HTTP/1.1を改良したHTTP標準の新しいバージョンです。

以下は、HTTP/2の特徴です。

● ヘッダー圧縮

HTTP/2からはヘッダー圧縮機能を提供します。また、一度送信したヘッダーが再度送信されないよう、サーバーとクライアントがヘッダー・テーブルにインデックスを作成しておきます。Settings Header Table Sizeより、インデックス付けするヘッダーのサイズを設定できます。

● 多重リクエストと多重レスポンス

HTTP/2では、リクエストとレスポンスのペアを1つのストリームと呼び、1つのストリームは複数のフレーム(ヘッダー・フレーム、データ・フレームなど)で構成されます。実際に接続によって送信される単位はフレームです。HTTP/1.1までは1つのリクエストとレスポンスが完了してから、次のリクエストを送信しましたが、HTTP/2からは複数のストリームを作成して同時にリクエストを送信することができます。

Settings Max Concurrent Streamsより、同時に使用するストリームの数を設定します。実際のデータ(ボディ)が送信されるフレームであるデータ・フレームのサイズは、Settings Max Frame Sizeで設定できます。

● サーバー・プッシュ

サーバー・プッシュについては、[「2.6.2. サーバー・プッシュ」](#)を参照してください。

参考

本書では、JEUSでHTTP/2を使用する内容についてのみ説明しています。HTTP/2についての詳細内容は、[『RFC文書』](#)を参照してください。

2.6.1. HTTP/2の設定

HTTP/2は、h2cとh2の2つの識別子を定義しています。各々の意味は以下のとおりです。

識別子	説明
h2c	HTTP/2を示します。「http」でサービスされます
h2	TLS(Transport Layer Security)を使用して動作するHTTP/2を示します。「https」でサービスされます

参考

ほとんどのブラウザは、**h2**のみサポートしています。

HTTP/2を**h2c**で使用するためには、「[2.3.3. HTTPリスナーの設定](#)」を参考にしてHTTP/2の使用項目をチェックします。

HTTP/2を**h2**で使用するためには追加操作が必要です。**h2**は、TLS(Transport Layer Security)を使用して動作しますが、TLSハンドシェイク中にALPN(Application Layer Protocol Negotiation)が必要となります。ALPNは、JDK 9に含まれる予定なので、JDK 7をベースにして開発されたJEUS 8では基本的にALPNを使用することができません。**h2**はTLSを使用して動作するので、セキュリティー・リスナーで設定したポートを使用する必要があります。

以下は、ALPNを使用するために必要な操作です。

1. 以下のアドレスより、使用するJDKバージョンに合ったalpn-bootライブラリーを確認した後、『[MVN Repository](#)』からダウンロードします。

```
http://www.eclipse.org/jetty/documentation/current/alpn-chapter.html#alpn-versions
```

2. ダウンロードしたalpn-bootライブラリーをbootclasspathに追加する<jvm-option>をdomain.xmlに追加します。

```
<jvm-config>
  <jvm-option>-Xbootclasspath/p:<path_to_alpn_boot_jar> ...</jvm-option>
</jvm-config>
```

注意

1. 上記で使用したalpn-bootライブラリーはOpenJDK、OracleJDKのみサポートします。したがって、現在のIBM JDKではHTTP/2を使用できません。

2. HTTP/2仕様では、TLS通信に必要な暗号スイート(Cipher Suites)のうち、推奨していない暗号スイートを明示しています。<http://httpwg.org/specs/rfc7540.html#BadCipherSuites>仕様では、推奨していない暗号スイートを使用した場合、接続が切断される可能性があるということを明示しており、ほとんどのブラウザはこれに従って接続を切断しています。

JDK 7で提供する暗号スイートはすべて推奨していないため、より強力な暗号スイートを使用するJDK 8バージョンの使用が不可欠です。

HTTP/2仕様では、TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256を必須対応として明示しているので、この暗号スイートをリスナーSSLで設定することをお勧めします。

3. JEUSサーバーと関連する詳細設定については、『[JEUS サーバガイド](#)』の「[2.3.2. リスナーの設定](#)」を参照してください。
-

2.6.2. サーバー・プッシュ

サーバー・プッシュは、指定されたリソースに対してクライアントからの要求がなくてもサーバーが応答を返す機能です。要求送信に必要な時間を短縮して、より迅速に応答を受信することができます。サーバー・プッシュを使用するためには、Webアプリケーションの開発時にサーバー・プッシュを適用するリソースを指定する必要があります。本節では、同機能の使用方法について説明します。

参考

サーバー・プッシュのためのサーブレットAPIは、サーブレット4.0に含まれる予定です。JEUSは、開発時点まで確定されたサーブレット4.0 APIに基づいてサーバー・プッシュを実装しています。

サーバー・プッシュの基本的なフローは、PushBuilderオブジェクトを取得した後、リソースのパスを指定してpush()メソッドを呼び出します。

次は、サーバー・プッシュの基本的な使用例です。例以外にも、プッシュされるリソースのクエリ文字列、クッキー、ヘッダーなどを変更することができます。

[例 2.8] サーバー・プッシュの使用例 : <<ServerPushServlet.java>>

```
package sample.serverpush

...
import jeus.servlet.engine.HttpServletRequestImpl;
import jeus.servlet.engine.PushBuilder;

@WebServlet("/ServerPush")
public class ServerPushServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
        ServletException, IOException {

        // JEUS内の実装であるHttpServletRequestImplでキャストした後
        // PushBuilderオブジェクトを取得します。
        HttpServletRequestImpl reqImpl = (HttpServletRequestImpl)req;
        PushBuilder builder = reqImpl.getPushBuilder();
        String pushResourcePath = "resources/a.txt";

        // プッシュするリソースのパスを入力してからpush()関数を呼び出します。
        builder.path(pushResourcePath);
        builder.push();

        resp.getWriter().write("main page\n");
    }
}
```

参考

サーバー・プッシュを使用しないと設定した場合は、上記の例は無効です。

2.7. リスナーのチューニング

最適の性能を発揮するため、リスナーの設定時にいくつかの事項を考慮する必要があります。

- システム・リソースを多く使用するか、待機時間を長くして出力バッファのサイズを増加させます。
- 一般的にワーカー・スレッド・プールのmin、max、step値を大きく設定すると、Webエンジンに多くのクライアントがアクセスするときにプールが良い性能を出します。システム・メモリーを少なく使用するには、これらの値を低く設定します。
- 「**Server Access Control**」オプションを無効にすると、性能改善が期待できます。
- WebtoBコネクターには前述した規則に従ってWebエンジンのWebtoBコネクターのワーカー・スレッド数とhttp.m値を同じく設定すると、最良の性能を発揮します。

第3章 Webコンテキスト

本章では、JEUS Webアプリケーションをパッケージングする方法と、これをJEUS Webエンジンにデプロイし、モニタリングする方法について説明します。

3.1. 概要

WebエンジンにWebアプリケーションをデプロイすると、Webコンテキストが生成されます。すなわち、WebアプリケーションとWebコンテキストは同様な意味で使われます。

3.2. 基本構造

本節では、Webコンテキストに含まれる内容と内部構造について説明します。

3.2.1. Webコンテキスト・コンテンツ

Webアプリケーションは、クライアントの要求時にWebベースのサービスを実行するための静的コンテンツと動的コンテンツで構成されています。

- 静的コンテンツ

すでに完成されているコンテンツとして、Webエンジンではいかなる処理もせずに返すすべてのタイプのデータを意味します。静的コンテンツの種類は以下のとおりです。

- HTMLページ
- テキスト・ファイル
- 画像ファイル
- 動画ファイル
- その他

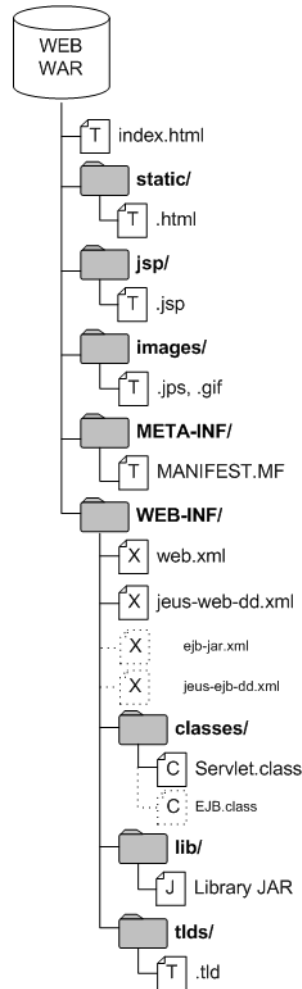
- 動的コンテンツ

サーブレット、JSPのようにユーザーの要求時に応答を生成するコンテンツです。

3.2.2. Webコンテキストの内部構造(WARファイルの構造)

WebコンテキストをWebエンジンにデプロイする前に、.war拡張子を持つJARファイル形式の圧縮ファイルにパッケージングします。これをWAR(Web ARchive)ファイルといいます。WARファイルの構造は以下のとおりです。

[図 3.1] WARファイルの構造



META-INF/

このディレクトリーは選択事項です。同ディレクトリーを使用する場合には、JARフォーマットに定義されたディスクリプター・ファイルのMANIFEST.MFファイルが含まれます。

WEB-INF/

このディレクトリーは必須事項です。サーブレット、フィルター、リスナー・クラスおよびライブラリーが含まれています。同ディレクトリーの下位には、以下のコンポーネントが存在します。

ファイル	説明
web.xml	Java EE標準のWebアプリケーションDDファイルです。Webアプリケーションのメタ情報が含まれています。

ファイル	説明
	Servlet 3.0からはweb.xmlが必須ファイルではなくなりました。その代わり、アノテーション、レジストレーションAPIを利用してサーブレット、フィルター、リスナーなどが追加できます。詳細内容は、サーブレット標準を参照してください
jeus-web-dd.xml	JEUSのWebアプリケーションDDです。詳細説明は、 「3.3.1. jeus-web-dd.xmlの設定」 を参照してください
ejb-jar.xml jeus-ejb-dd.xml	Java EE 6に含まれたEJB 3.1の定義により、WARファイルにEJBを含めることができます。サーブレットと同様、アノテーションで定義できます。詳しい説明は、EJB標準または『JEUS EJBガイド』を参照してください
classes/	サーブレット・クラスとユーティリティ・クラスが下位ディレクトリーに含まれています。 標準Javaパッケージ構造で整理されています
lib/	Webアプリケーションに必要なJavaライブラリーが含まれます。 ライブラリーは.jarファイルにパッケージされ、このパスに格納されます。同ファイルの内容は、自動的に全サーブレットのクラス・パスに追加されます
tlds/	JSPページのための Custom Tag Library Descriptorが含まれます

その他

JSP、HTML、画像ファイルのようなコンテンツが含まれている任意の名前のディレクトリーです。

3.3. Webコンテキストのデプロイ

本節では、Webコンテキストのデプロイについて説明します。デプロイとは、JEUS WebエンジンでWebアプリケーションがサービスできるように準備し、Webコンテキストを生成する作業です。

Webコンテキストは論理的に仮想ホストの下位に存在します。仮想ホストについての詳細内容は、[「第5章 仮想ホスト」](#)を参照してください。

3.3.1. jeus-web-dd.xmlの設定

必要な場合に限り、WEB-INF/ディレクトリーの下位にweb.xmlとjeus-web-dd.xmlを生成します。

以下は、jeus-web-dd.xmlファイルの例です。一部の項目は省略されています。省略された項目については、『JEUS XMLリファレンスガイド』の「13. jeus-web-dd.xmlの設定」を参照してください。

[例 3.1] Webコンテキストの設定ファイル：<<jeus-web-dd.xml>>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="8.0">
  <context-path>/examples</context-path>
```

```

<enable-jsp>true</enable-jsp>
<auto-reload>
  <enable-reload>true</enable-reload>
  <check-on-demand>true</check-on-demand>
</auto-reload>
<added-classpath>
  <class-path>/home/user1/libs/lib.jar</class-path>
</added-classpath>
<session-config>
  . . .
</session-config>
<async-config>
  <async-timeout-millis>600000</async-timeout-millis>
  <background-thread-pool>
    <min>0</min>
    <max>10</max>
  </background-thread-pool>
  <dispatch-thread-pool>
    <min>0</min>
    <max>10</max>
  </dispatch-thread-pool>
</async-config>
<properties>
  <property>
    <key>jeus.servlet.jsp.modern</key>
    <value>true</value>
  </property>
</properties>
</jeus-web-dd>

```

以下は、設定タグについての説明です。

タグ	説明
<context-path>	Webコンテキストにアクセスするためのルート・パスです。必ず「/」で始まり、仮想ホスト内で一意である必要があります。 「http://www.foo.com/examples/index.html」のようなURLにおいて、<context path>は「/examples」になります
<enable-jsp>	falseに設定した場合はJSP機能を使用しません。基本的にJSPをサポートします
<user-log>	Webコンテキストが残すログです。ユーザー・ログについての詳細内容は、 1.6.10 節「ユーザー・ログの設定」 を参照してください
<auto-reload>	ディスクのサーブレット・クラスが変更されたとき、自動でローディングするか否かを決めるオプションです。詳細内容は、「 第9章 クラスの動的反映 」を参照してください

タグ	説明
<added-classpath>	<p>サーブレットを実行するときに参照するライブラリーのリストです</p> <p>ディレクトリー設定も可能です</p>
<allow-indexing>	<p>クライアントの要求時に、実際のディレクトリー・リストが出力されるURLパスのリストです。クライアントが要求したファイルがディレクトリーに存在しない場合、ディレクトリーのリストが出力されます。</p> <p>ディレクトリー・リストからファイルを1つ選択すると、ファイルの内容が出力されます</p>
<deny-download>	<p>いかなる場合であっても直接ダウンロードやアクセスできないリソースを規定するためのフィルターです。このフィルターはファイル名の拡張子でファイル名とパスを指定することができます。</p> <p>フィルターのファイル名とディレクトリーは、context document baseディレクトリーに相対的なパスで設定されます</p>
<aliasing>	<p>カスタムURLパスに対するディレクトリー・マッピングです。</p> <p>別の位置のディレクトリーをURL要求パスにマッピングする必要がある場合にエイリアシング機能を使用します</p>
<file-caching>	<p>応答速度を高めるため、ランタイム・メモリーにどのような静的コンテンツをキャッシュするかを決めます。</p> <p>この項目の下位設定には、キャッシュ・メモリーの最大サイズ(単位: MB)、要求がないときにキャッシュ内に存在できる静的コンテンツの最大サイズ、キャッシングが必要なコンテンツのパスなどがあります</p>
<jsp-engine>	<p>Webコンテキストに含まれたJSPページについての設定です。詳細内容は、「4.4.2. jeus-web-dd.xmlの設定」を参照してください</p>
<session-config>	<p>Webコンテキスト・レベルのセッションを設定します。</p> <p>この設定はWebエンジン設定より優先順位が高いです。セッション設定についての詳細内容は、「1.6.9. セッションの設定」を参照してください</p>
<webinf-first>	<p>クラス・ローディングの優先順位に関するオプションです。trueに設定すると、当該アプリケーションに限ってWEB-INF下位のクラスを優先的にローディングします。</p> <p>JEUS 7 Fix#4からは、trueに設定した際、クラス・ローディング関連の競合が発生した場合は、以下のように特定のパッケージあるいはクラスのみ除外できる機能が追加されました</p> <pre><webinf-first> <enabled>true</enabled></pre>

タグ	説明
	<pre><excluded-package>package.name.</excluded-package> ... </webinf-first></pre>
<attach-stacktrace-on-error>	サーバーに問題が生じる場合、エラーの詳細内容をブラウザに表示するか否かについての設定です。このメッセージは開発時には有用ですが、運用時には削除するのが望ましいです
<keep-alive-error-response-codes>	<p>エラー応答コードのうち、Connection: Closeの代わりにKeep-Alive応答を希望する場合に設定します。Webアプリケーションが直接エラー応答を送信するときに適用され、エンジン内部的にコネクションの切断が必要な場合には適用されません。</p> <p>たとえば、サーブレットがresponse.sendError(500)を実行した場合は適用されますが、サーブレットで例外が発生し、エンジンが500エラーを送信する場合には適用されません。複数の応答コードを設定するには、「404、503」のようにコンマ(,)で区切ります</p>
<encoding>	<p>要求(Request)/応答(Response)に関するエンコーディング設定です。Webエンジン設定(domain.xml)と同様ですが、同時に設定される場合は、jeus-web-dd.xmlの<request-encoding>設定と<response-encoding>設定が優先的に適用されます。詳しい内容は、「1.6.5. エンコーディングの設定」を参照してください</p> <ul style="list-style-type: none"> 要求エンコーディング: URLクエリ文字列、application/x-www-form-urlencoded形式のボディ、getReader() <p>JEUS 7 Fix#4から<request-encoding><url-mapping>設定が追加されました。これは、特定のサーブレット・パスにマッピングされているエンコーディングを適用する設定です。ただし、<forced>または<client-override>が設定されている場合は無視されます</p> <ul style="list-style-type: none"> 応答エンコーディング: 応答HTTPボディに適用されるエンコーディング
<cookie-policy>	HTTP要求ヘッダーに存在するクッキーを読み込んだり、アプリケーション要求によって新しいクッキーをHTTP応答ヘッダーに適用するポリシーが設定できます。詳しい内容は、「 1.6.8. クッキー・ポリシーの設定 」を参照してください
<async-config>	サーブレット3.0から追加された非同期プロセッシングに関する設定です。下位にはAsync Timeout、Background Thread Pool、Dispatch Thread Poolについての設定が可能です。非同期プロセッシングについての詳細内容は、サーブレット標準を参考にしてください
<web-security>	sendRedirectを行う場合のアドレス値の検証に関するポリシー設定など、Webアプリケーションに限られたセキュリティー・ポリシーの設定グループです

タグ	説明
<websocket>	WebアプリケーションのWebSocketコンテナについての設定です。詳細内容は、 「6.4. WebSocketコンテナの設定」 を参照してください
<properties>	Webアプリケーションに適用されるプロパティを設定します。こちらに設定されたプロパティは別のWebアプリケーションには適用されず、設定されたWebアプリケーションにのみ適用されます

参考

<async-config><async-timeout-millis>のデフォルト値は30秒です。バックグラウンド・スレッドで実行する作業が長くなり、タイムアウト・エラーが発生する場合は、jeus-web-dd.xmlにこの値を設定するか、`javax.servlet.AsyncContext#setTimeout()`を使用してプログラムで調整します。

3.3.2. Webコンテキスト再デプロイ(グレースフル再デプロイ)

本章では、Webコンテキストの再デプロイについての説明は省略します。詳しい内容は、『*JEUS アプリケーション&デプロイメントガイド*』の「2.2. グレースフル再デプロイ」を参照してください。

3.3.3. 共有ライブラリー参照の追加

共有ライブラリー(Shared Library)は、アプリケーション別に必要なライブラリーをWEB-INF/libではなく、JEUS_HOME/lib/sharedの下位に追加し、複数のアプリケーションで共有して使用することができます。JEUSの再起動は不要であり、libraries.xmlファイルを修正した後、再デプロイすると変更されたライブラリーが適用されます。

共有ライブラリーを追加するには、以下のとおりJEUS_HOME/lib/shared/libraries.xmlファイルで<library>タグを追加し、共有するJARファイルを指定します。バージョンは、仕様バージョンと実装バージョンをそれぞれ指定できます。

[例 3.2] 共有ライブラリー参照の追加 : <<libraries.xml>>

```
<libraries xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <!-- DO NOT MODIFY JSF1.2 and JSTL1.2 libraries!!! -->
  <library>
    <library-name>jsf</library-name>
    <specification-version>1.2</specification-version>
    <implementation-version>1.2_06</implementation-version>
    <files dir=".">
      <include name="jsf-injection-provider.jar"/>
    </files>
    <files dir="jsf_ri_1.2"/>
  </library>
```

```

<library>
  <library-name>jsf</library-name>
  <specification-version>2.0</specification-version>
  <implementation-version>2.0.0</implementation-version>
  <files dir=".">
    <include name="jsf-injection-provider.jar"/>
    <include name="jsf-weld-integration.jar"/>
  </files>
  <files dir="jsf-ri_2.0"/>
</library>
<library>
  <library-name>jstl</library-name>
  <specification-version>1.2</specification-version>
  <implementation-version>1.2</implementation-version>
  <files dir="jstl_1.2"/>
</library>

<!-- Add user libraries from here -->
<library>
  <library-name>myLibrary</library-name>
  <specification-version>2.0</specification-version>
  <implementation-version>2.1</implementation-version>
  <files dir=".">
    <include name="commons-logging.jar"/>
    <include name="commons-util.jar"/>
  </files>
  <files dir="myLib-2.1"/>
</library>
</libraries>

```

設定した共有ライブラリーを参照するには、jeus-web-dd.xmlに<library-ref>を設定します。バージョンが明示されていない場合は、libraries.xmlに登録された同じ名前のライブラリーの中から、ルート・バージョンを参照します。

```

<library-ref>
  <library-name>myLibrary</library-name>
  <specification-version>
    <value>2.0</value>
  </specification-version>
</library-ref>

```

JSF、JSTLライブラリーの設定

[例 3.2]で説明したlibraries.xmlファイルには、JSFとJSTLが共有ライブラリーとして追加されています。Java EE標準にはJSFとJSTLが含まれているため、JEUSではデフォルトで提供しています。

JEUSシステム・ライブラリーとして提供することも可能ですが、1つのJSF、JSTLの実装のみ使用できる制約が発生します。JSF実装は、2.0のみならず、1.2バージョンもあり、Sun RI以外にApacheのMyFacesも存在するため、JEUSでは様々なバージョンの実装が使用できるように共有ライブラリーの形式でJSF、JSTLを提供します。

実装の位置と種類によって以下のように区分されます。

- WEB-INF/libの下位に含まれたJSFまたはJSTL実装
- 共有ライブラリーとして含ませた実装(上記の実装より優先順位が低い)

[変更]

以下は、JEUS 7 Fix#2から変更された事項についての説明です。

このバージョンからは、JEUS_HOME/lib/shared下位のOracle JSF RIを自動で追加しません。そのため、明示的にjeus-web-dd.xmlから<library-ref>を参照するように設定する必要があります。

以下のように宣言すると、JEUS_HOME/lib/sharedにデフォルトで含まれたライブラリーを使用できます。

```
<library-ref>
  <library-name>jsf</library-name>
</library-ref>
```

また、JEUS内で判断してOracle JSF RI 2.xに含まれたWebリスナーを自動的に追加しません。明示的にweb.xmlで<listener>に設定します。

```
<listener>
  <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
</listener>
```

同じJSFやJSTLライブラリーを複数のアプリケーションで参照して使用するためには、JEUS_HOME/lib/shared/libraries.xmlに<library>を直接登録し、jeus-web-dd.xmlから<library-ref>で参照するように設定します。

JSFの場合、マネージドBeanから@EJBや@Resourceのようなアノテーションを使用してWASが提供するリソースにアクセスすることが可能です。WASはアノテーションとして明示されたリソースをインジェクションで必要があります。

JEUSでは、JSF RIにJEUSインジェクションをサポートするため、共有ライブラリーとしてjsf-injection-provider.jarを提供しています。共有ライブラリーは以下のディレクトリーに存在します。

```
JEUS_HOME/lib/shared
```

これは、web.xmlにjavax.faces.webapp.FacesServletが含まれた場合、自動的に含まれます。

3.3.4. 互換性のためのWebコンテキスト・レベルのオプション設定

旧バージョンのJEUSまたは他WASとの互換性に問題が生じた場合、Webコンテキスト・レベルでオプションを設定して解決することができます。

旧バージョンのJEUSとの互換性

サーブレット2.4標準の以前は、制限事項についての記述が不十分な場合が多くありました。これは、Servlet 2.4、2.5を経て補完され、JEUSにおいても自然と制限事項が生じるようになりました。また、JEUSも様々な問題点を経験し、修正しながら、標準に準拠しない動作のサポートを中止しているため、JEUS 4で作成したWebアプリケーションがJEUS 6で正常に動作しない問題が生じる可能性があります。

JEUSには、サーブレットおよびJSP標準に準拠すると同時に既存アプリケーションの動作互換性を保証する責任もあります。しかし、この2つは相反する面があり、標準に準拠するためにjavax.servlet標準APIや内部動作を修正すると、既存の誤った動作をベースにして作成したアプリケーションが存在する場合は問題が発生します。このような状況においては、基本的に標準への準拠のほうが優先順位が高く、当該アプリケーションを別のWebコンテナでは実行できなくなるため、該当するアプリケーションを修正する必要があります。さらに、アプリケーションをアップグレードする際にサーブレット標準に準拠するフレームワークとの衝突問題が発生する可能性があります。

このようなアプリケーションの問題点を初期に把握できず、サービス開始日の直前に発見した場合は、アプリケーションを修正する代わりにJEUSの動作修正が要求されます。この要求をサポートするため、JEUSでは下位互換性のためのプロパティを提供します。

JSPエンジンに関する互換性問題は、[「4.4.3. JSPの下位互換性のためのWebコンテキスト・レベルのオプション設定」](#)を参照し、下位互換性のためにJEUSが提供するプロパティについては、『*JEUS リファレンスガイド*』の「1.6.3.3. 互換性提供プロパティ」を参照してください。

他WASとの互換性

サーブレット2.4標準の以前は、サーブレットとJSPを直接使用してWebアプリケーションを開発しましたが、現在はWebフレームワークを使用する傾向であり、Java EE 5からは標準WebフレームワークとしてJSFとJSTLが追加されました。このようなWebフレームワークは、主にサーブレット、JSP仕様の参照実装であるTomcatを使って開発、テストするため、Webフレームワークが非標準的な機能をサポートする場合、様々な互換性関連の問題が発生する可能性があります。

互換性に問題が生じた場合、必要に応じてjeus-web-dd.xmlに設定することができます。jeus-web-dd.xmlの設定についての詳細内容は、『*JEUS リファレンスガイド*』の「1.6. Webエンジンのプロパティ」を参照してください。

3.3.5. 付加機能を使用するための設定

JEUSが提供する以下の付加機能を使用するには別途の設定が必要です。

- 静的リソースを処理するサーブレット・マッピング

サーブレット標準ではサーブレットまたはJSP以外の場合、WASで提供するデフォルト・サーブレットで処理するようになっています。これをリソース・サーブレットといいます。

以下のようにweb.xmlの<servlet-mapping>設定にjeus.servlet.servlets.ResourceServletでマッピングします。

[例 3.3] 静的リソースを処理するサーブレット・マッピング : <<web.xml>>

```
<servlet-mapping>
  <servlet-name>jeus.servlet.servlets.ResourceServlet</servlet-name>
  <url-pattern>/static/*</url-pattern>
</servlet-mapping>
```

- **Spring 3.0以上でmvc:default-servlet-handlerを登録する方法**

JEUS 8では、Spring 3.0以降でmvc:default-servlet-handlerを登録する必要はありません。Tomcatと同様、「default」という名前でJEUSのdefault servletを提供します。

- **Servlet 3.0 Multipart機能を使用する場合、ProgressListenerを登録する方法**

サーブレット3.0からはサーブレットごとにmultipart-config設定を登録して、POST要求の*multipart/form-data*処理をWebコンテナに委任することができます。このとき、処理進行状況についてのイベントを受信したい場合は、ProgressListenerを登録します。

1. jeus-servlet.jarに含まれた**jeus.servlet.engine.multipart.ProgressListener interface**を実装します。インターフェースについての内容は、サーブレットAPI文書を参照してください。
2. 上記のインターフェース実装クラスのインスタンスをHttpServletRequest attributeに追加します。名前はインターフェースと同様、**jeus.servlet.engine.multipart.ProgressListener**です。このときの注意点は、getPart()、getParts()またはgetParameter()を呼び出す前である必要があります。

3.3.6. Webセキュリティの設定

jeus-web-dd.xmlを利用してアプリケーション別に適用するWebセキュリティ設定について説明します。

リダイレクト・ロケーション・セキュリティ設定

アプリケーションはjavax.servlet.http.HttpServletResponse.sendRedirect(String location)標準APIを介して「302 Found」応答が送信できます。このとき、基本的にロケーションとして渡す文字列に対し、いかなる確認もせずにそのままURLに変換してロケーション・ヘッダーとして設定します。そのため、ユーザーが悪意的にCRLFインジェクションのような攻撃をする可能性があります。これを防ぐため、アプリケーションはjeus.servlet.security.RedirectStrategyインターフェースを実装してjeus-web-dd.xmlに設定することができます。

以下は、jeus.servlet.security.RedirectStrategyインターフェースの設定例です。

[例 3.4] リダイレクト・ロケーション・セキュリティ設定のインターフェース: <<RedirectStrategy>>

```
package jeus.servlet.security;

import javax.servlet.http.HttpServletRequest;

public interface RedirectStrategy {
    /**
     * Makes the redirect URL.
     *
     * @param location the target URL to redirect to, for example "/login"
     */
    String makeRedirectURL(HttpServletRequest request, String location)
        throws IllegalArgumentException;
}
```

設定したインターフェースを以下のように実装できます。

[例 3.5] リダイレクト・ロケーション・セキュリティ設定の実装例: <<RejectCrLfRedirectStrategy>>

```
package jeus.servlet.security;

import javax.servlet.http.HttpServletRequest;

public class RejectCrLfRedirectStrategy implements RedirectStrategy {
    private Pattern CR_OR_LF = Pattern.compile("\\r|\\n");

    @Override
    String String makeRedirectURL(HttpServletRequest request, String location)
        throws IllegalArgumentException {
        if (CR_OR_LF.matcher(location).find()) {
            throw new IllegalArgumentException("invalid characters (CR/LF) in
redirect location");
        }
        return makeAbsolute(location);
    }

    private String makeAbsolute(String location) {
        // make code for make absolute path
    }
}
```

jeus-web-dd.xmlの設定方法は以下のとおりです。

[例 3.6] リダイレクト・ロケーション・セキュリティ設定 : <<jeus-web-dd.xml>>

```
...
<web-security>
  <redirect-strategy-ref>
    jeus.servlet.security.RejectCrLfRedirectStrategy
  </redirect-strategy-ref>
</web-security>
..
```

<redirect-strategy-ref>は、jeus.servlet.security.RedirectStrategy インターフェースを実装したクラス名です。同クラスは、Webアプリケーションのクラス・パスに含まれます。

上記のjeus.servlet.security.RejectCrLfRedirectStrategyは、デフォルトで提供するRedirectStrategyであり、CR、LFまたはCRLFが存在するロケーションがsendRedirect APIに渡される場合、500エラーが発生します。

以外にもCR、LFまたはCRLF文字列を空文字列に置き換えるjeus.servlet.security.RemoveCrLfRedirectStrategyを提供します。

3.3.7. セキュリティー・ロール・マッピング

本節では、web.xmlに定義されたセキュリティ・ロールを実際のシステム・ユーザーとユーザー・グループに設定する方法について説明します。同マッピングは、jeus-web-dd.xmlに記述されます。

以下の内容をweb.xmlに定義したと仮定します。

[例 3.7] セキュリティー・ロール・マッピング : <<web.xml>>

```
<web-app>
  <security-role>
    <role-name>manager</role-name>
  </security-role>
  <security-role>
    <role-name>developer</role-name>
  </security-role>
  <servlet>
    . . .
  </servlet>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>
        MyResource
      </web-resource-name>
      <url-pattern>/jsp/*</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
  </security-constraint>
</web-app>
```

```

        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>manager</role-name>
    </auth-constraint>
</security-constraint>
</web-app>

```

上記例では、2つのセキュリティ・ロールが宣言されています(太字で記述された部分)。太字で記述された3つ目の部分は、マネージャー・ロールが<security-constraint>タグにどのように使用されるかを示しています。

このマッピングをjeus-web-dd.xmlに記述すると以下のとおりです。

[例 3.8] セキュリティ・ロール・マッピング : <<jeus-web-dd.xml>>

```

<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="7.0">
    . . .
    <role-mapping>
        <role-permission>
            <principal>Peter</principal>
            <role>manager</role>
        </role-permission>
        <role-permission>
            <principal>Linda</principal>
            <role>developer</role>
        </role-permission>
    </role-mapping>
    . . .
</jeus-web-dd>

```

Webアプリケーションが実際のシステムにデプロイされるとき、「manager」と「developer」ロールをシステムの特定のユーザーにバインディングする必要があります。このマッピングは<context><role-mapping>に定義されています。<role-mapping>タグは<role-permission>タグを含みます。同タグを使用して<principal-to-role>マッピングを定義します。

以下の下位タグと一緒にバインディングされます。

タグ	説明
<role>(1つ、必須)	web.xmlに定義された<role-name>値です。上記例では<role-name>が「manager」です
<principal>(0以上)	<p><role-name>との連携が必要な、JEUSが管理するユーザー名です。</p> <p>詳しい内容は、『JEUS セキュリティガイド』の「3.3. Webモジュールのセキュリティ設定」を参照してください</p>

例では、2つのロール「manager」と「developer」がそれぞれ「Peter」と「Linda」にマッピングされています。

3.3.8. シンボリック・リファレンスのマッピング

ロールが実際のユーザーにマッピングされるのと同様、EJBリファレンス、リソース・リファレンス、管理されるオブジェクトのリファレンスを実際のシステム・リソースにマッピングする必要があります。

以下は、シンボリック・リファレンスのマッピング例です。

[例 3.9] シンボリック・リファレンスのマッピング : <<web.xml>>

```
<web-app>
    . . .
    <ejb-ref>
        <ejb-ref-name>ejb/account</ejb-ref-name>
        <ejb-ref-type>Entity</ejb-ref-type>
        <home>com.mycompany.AccountHome</home>
        <remote>com.mycompany.Account</remote>
    </ejb-ref>
    <resource-ref>
        <res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
        <res-sharing-scope>Shareable</res-sharing-scope>
    </resource-ref>
    <resource-env-ref>
        <resource-env-ref-name>
            jms/StockQueue
        </resource-env-ref-name>
        <resource-env-ref-type>
            javax.jms.Queue
        </resource-env-ref-type>
    </resource-env-ref>
    . . .
</web-app>
```

このWebアプリケーションを登録するため、web.xmlの<ejb-ref>、<resource-ref>、<resource-env-ref>下位のすべてのシンボリック・リファレンス名が、実際に目的のリソースのJNDI名とマッピングされる必要があります。同マッピングはjeus-web-dd.xmlに<ejb-ref>、<res-ref>、<res-env-ref>を追加すると完成されます。

このタグには、以下の下位タグが含まれています。

- <jndi-info>

タグ	説明
<ref-name>	リファレンス名としてweb.xmlとサーブレット・コードに定義されている名前と同じです

タグ	説明
<export-name>	JNDIエクスポート名は、<ref-name>が指し示す実際リソースのエクスポート名です

以下は、JNDI名が上記の3つのリファレンスとマッピングされたjeus-web-dd.xmlの例です。

[例 3.10] シンボリック・リファレンスのマッピング : <<jeus-web-dd.xml>>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="7.0">
  . . .
  <ejb-ref>
    <jndi-info>
      <ref-name>ejb/account</ref-name>
      <export-name>AccountEJB</export-name>
    </jndi-info>
  </ejb-ref>
  <res-ref>
    <jndi-info>
      <ref-name>jdbc/EmployeeAppDB</ref-name>
      <export-name>EmployeeDB</export-name>
    </jndi-info>
  </res-ref>
  <res-env-ref>
    <jndi-info>
      <ref-name>jms/StockQueue</ref-name>
      <export-name>StockQueue</export-name>
    </jndi-info>
  </res-env-ref>
  . . .
</jeus-web-dd>
```

3.4. Webコンテキストのモニタリング

モニタリングとは、Webコンテキストの情報を確認する作業です。WebエンジンにデプロイされているWebコンテキストのリストおよび現在の状態などをWebAdminまたはコンソール・ツールを使用して照会することができます。

WebAdminの使用

WebAdminを使用してデプロイされたWebコンテキストを照会することができます。照会方法は以下のとおりです。

1. WebAdminの**[Applications]**メニューを選択すると、ドメイン内のアプリケーション・リストが照会されます。リストから「**Application Type**」が「**WAR**」であるアプリケーションのIDをクリックします。(WebアプリケーションのタイプはWARです。)

[図 3.2] Webコンテキストのモニタリング - アプリケーションの照会

The screenshot shows the 'Deployed Application' interface. At the top, there's a 'Deployed Application' header with a 'HISTORY' dropdown. Below it, a message states: 'ドメインでサービスするアプリケーションの照会およびデプロイコマンドをサポートします。Lock & Editモードでは一部のコマンドが使用できません。' (Supports querying and deployment commands for applications served by the domain. Some commands are not available in Lock & Edit mode). There are tabs for 'Applications' (selected) and 'Deployed Library'. A sub-header reads 'Application information for the domain [domain 1]'. Below this is a table with columns: Application ID, Application Type, State, and a set of action buttons. The table contains one entry: 'app1' (WAR, RUNNING). Action buttons for 'app1' include 'Redeploy', 'Add Target', and 'Remove Target'. At the bottom right, there are global action buttons: 'Start', 'Stop', 'Deploy', 'Undeploy', and 'Uninstall'.

Application ID	Application Type	State	Actions
app1	WAR	RUNNING	Redeploy, Add Target, Remove Target

Global Actions: Start, Stop, Deploy, Undeploy, Uninstall

2. モニタリングするWebアプリケーションのIDをクリックすると、以下のとおり当該アプリケーションの情報が照会されます。WebアプリケーションのID、パス、タイプなどの基本情報およびターゲット関連の情報、オプション関連の情報などが照会されます。画面についての詳しい説明は、『*JEUS アプリケーション&デプロイメントガイド*』の「4.3.10.1. WebAdminの使用」を参照してください。

【図 3.3】 Webコンテキストのモニタリング - Webアプリケーションの選択

Deployed Application

HISTORY

ドメイン内でサービスされるアプリケーションのデプロイに関する詳細設定を定義します。サーバが起動するときにデプロイされるアプリケーションの設定です。

ヘルプ

Deployed Application

ApplicationsDeployed Library

Information

Deployed Application

ID	app1
Path	C:\TmaxSoft\JEUS8.0\domains\domain1\applications\app1\app1.war
Type	WAR
Application Time	Fri Sep 30 17:05:18 KST 2016

Targets

Cluster	cluster1
Running Servers	adminServer [RUNNING] server1 [RUNNING]

Options

Security Domain Name	SYSTEM_DOMAIN
----------------------	---------------

- 画面の「**Running Servers**」に照会されたサーバーから目的のサーバーをクリックすると、選択したサーバーの詳細情報画面が新しいタブで表示されます。

[図 3.4] Webコンテキストのモニタリング - Webアプリケーションの詳細情報の照会

Deployed Application

HISTORY

ドメイン内でサービスされるアプリケーションのデプロイに関する詳細設定を定義します。サーバが起動するときにデプロイされるアプリケーションの設定です。

ヘルプ ?

Applications

Deployed Library

Information | server1

General information about the web module [app1].

Module Name	Unique Module Name	Context Path	Target Session Cluster	Command
app1	app1	/app1		<div>Reload</div> <div>Thread Info</div>

Servlets

Name	Class	State	Count	Attribute	RegType	URLPatterns
該当する内容が存在しません。						

Filters

Name	Class	Attribute	RegType	URLPatterns	Servlets
該当する内容が存在しません。					

Listeners

Name	Type	RegType
該当する内容が存在しません。		

Jeus web deployment descriptor

Jeus-web-dd.xml
該当する内容が存在しません。

EJBs

Bean Name	Type	Local Export Name	Remote Export Name
該当する内容が存在しません。			

以下は、画面の各項目についての説明です。

領域	説明
General information of web module	選択されたWebアプリケーションのモジュール名、コンテキスト名、コンテキスト・パスの情報が照会されます
Servlets	選択されたWebアプリケーションに登録されたサーブレットに関する情報が照会されます。

領域	説明
	各サーブレットに対し、web.xmlやアノテーションまたはプログラムAPIを利用して登録された形式およびマッピングされたURLパターン、クラス名、サーブレットの状態、非同期/同期などの情報が照会されます
Filters	<p>選択されたWebアプリケーションに登録されたフィルターの情報が照会されます。</p> <p>各フィルターに対し、web.xmlやアノテーションまたはプログラムAPIを利用して登録された形式およびマッピングされたURLパターン、クラス名などが照会されます</p>
Listeners	選択されたWebアプリケーションに登録されたリスナーの登録方式およびリスナーの種類が照会されます
Jeus web deployment descriptor	選択されたWebアプリケーションのjeus-web-dd.xmlファイルの内容を表示します
EJBs	選択されたWebアプリケーションに登録されたEJBのBean Name、Type、Local Export Name、Remote Export Nameの情報が照会されます

コンソール・ツールの使用

コンソール・ツールを使用してWebコンテキストをモニタリングできます。

デプロイされたWebコンテキストを照会するには、**application-info**コマンドを実行します。Webコンテキストのすべての情報が照会されます。application-infoコマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.6.3. application-info」を参照してください。

```
application-info -type WAR
```

特定のWebコンテキストの情報を照会するには、-id <application-id> オプションを設定してコマンドを実行します。該当するIDのWebコンテキスト情報が照会されます。

```
application-info -id <application-id>
```

3.5. Webコンテキストの制御

Webコンテキストの制御とは、Webコンテキストの状態を変更してサービスを変更することを意味します。WebAdminまたはコンソール・ツールを使用してWebコンテキストをリロード、一時停止、再開などの制御が行えます。

3.5.1. Webコンテキストのリロード

デプロイされているWebコンテキストが変更されたり、何らかの理由で再び初期化する必要があったりする場合、Webコンテキストを削除して再デプロイするのではなく、変更事項を適用してWebコンテキストをリロードします。

WebAdminとコンソール・ツールを使用してWebコンテキストをリロードすることができます。

WebAdminの使用

WebAdminを使用したWebコンテキストのリロード方法は以下のとおりです。

[Applications]メニューを選択すると表示されるアプリケーション・リストから[reload]ボタンをクリックすると、画面の上部にリロードの結果が表示されます。

[図 3.5] Webアプリケーションのリロード



コンソール・ツールの使用

コンソール・ツールを使用してWebコンテキストをリロードできます。

以下のとおり`reload-web-context`コマンドを実行します。`reload-web-context`コマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.8.22. reload-web-context」を参照してください。

```
reload-web-context -ctx <context-name>
seccessfully reloaded
```

3.5.2. Webコンテキストの非同期要求に対するスレッド・プールのモニタリング

Webコンテキストが非同期要求(Asynchronous Request)を処理する際、JEUSが提供するスレッド・プールを使用するには、jeus-web-dd.xmlの<async-config>に関連情報を設定する必要があります。<async-config>を設定した後、JEUSが提供する非同期スレッド・プールの情報をモニタリングすることができます。

以下は、jeus-web-dd.xmlの設定例です。

[例 3.11] <async-config>設定 : <<jeus-web-dd.xml>>

```
...
<async-config>
  <dispatch-thread-pool>
    <min>5</min>
    <max>20</max>
  </dispatch-thread-pool>
  <background-thread-pool>
    <min>5</min>
    <max>20</max>
  </background-thread-pool>
  <async-timeout-millis>120000</async-timeout-millis>
</async-config>
...
```

以下は、設定タグについての説明です。

タグ	説明
<dispatch-thread-pool>	AsyncContext#dispatchの呼び出し時に使用するスレッド・プールの最小スレッド値および最大スレッド値を設定します
<background-thread-pool>	AsyncContext#startの呼び出し時に使用するスレッド・プールの最小スレッド値および最大スレッド値を設定します
<async-timeout-millis>	非同期作業の実行時にWebコンテナがタイムアウトを処理する基準となる時間を設定します。アプリケーションがAsyncContext#setTimeoutを呼び出さない場合、AsyncContext#getTimeoutのときに設定値を返します (デフォルト値: 30000、単位: ms)

設定された情報による非同期スレッド・プールの情報は、WebAdminを使用して確認できます。

WebAdminの使用

WebAdminを使用したWebコンテキストのスレッド情報の照会方法は以下のとおりです。

1. **[Applications]**メニューを選択すると照会されるアプリケーション・リストからWebアプリケーションのIDをクリックします。「**Running Servers**」リストから照会するサーバーをクリックします。
2. 当該サーバー名のタブが追加されます。Webコンテキストの**[thread-info]**ボタンをクリックすると、thread-info情報画面が新しいタブで表示されます。

[図 3.6] Webアプリケーションのthread-info

Deployed Application

HISTORY

ドメイン内でサービスされるアプリケーションのデプロイに関する詳細設定を定義します。サーバが起動するときにデプロイされるアプリケーションの設定です。

ヘルプ ?

ApplicationsDeployed Library

Informationserver1Thread Info

Thread information for the server [server1]

Asynchronous processing background threads for the web context[lapp1]

tid	name	state	elapsed	uri
該当する内容が存在しません。				

elapsed: Elapsed time (ms)

Asynchronous processing background thread statistics for the web context[lapp1]

	total	active	idle	blocked	reconn
The number of threads.	0	0	0	0	0

total = active + idle, reconn: reconnecting

Asynchronous processing dispatch threads for the web context[lapp1]

tid	name	state	elapsed	uri
該当する内容が存在しません。				

elapsed: Elapsed time (ms)

Asynchronous processing dispatch thread statistics for the web context[lapp1]

	total	active	idle	blocked	reconn
The number of threads.	0	0	0	0	0

total = active + idle, reconn: reconnecting

3.5.3. Webコンテキストの一時停止

Webエンジン内にデプロイされ、サービスされているWebコンテキストのサービスを、管理者が一時停止することができます。その後、Webコンテキストをリロードすることもできます。Webコンテキストのリロードについての詳細内容は、「[3.5.4. Webコンテキストのリロード](#)」を参照してください。

- WebAdminの使用

WebAdminを使用したWebコンテキストの一時停止方法は、『*JEUS アプリケーション&デプロイメントガイド*』の「4.3.7. アプリケーションの停止」を参照してください。

- コンソール・ツールの使用

コンソール・ツールを使用してWebコンテキストを一時停止することができます。

以下のとおり**suspend-web-component**コマンドを実行します。suspend-web-componentコマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.8.35. suspend-web-component」を参照してください。

```
suspend-web-component -ctx <context> -svl <servlet>
```

3.5.4. Webコンテキストのリロード

一時停止されているWebコンテキストを再開することができます。WebAdminとコンソール・ツールを使用してWebコンテキストを再開することができます。

- WebAdminの使用

WebAdminを使用して一時停止されたWebコンテキストを再開する方法は、『*JEUS アプリケーション&デプロイメントガイド*』の「4.3.6. アプリケーションの開始」を参照してください。

- コンソール・ツールの使用

コンソール・ツールを使用して一時停止されたWebコンテキストを再開することができます。

以下のとおり**resume-web-component**コマンドを実行します。resume-web-componentコマンドについての詳細内容は、『*JEUS リファレンスガイド*』の「4.2.8.31. resume-web-component」を参照してください。

```
resume-web-component -ctx <context> -svl <servlet>
```

3.6. Webコンテキストのチューニング

最適の性能を出すため、Webコンテキストの設定(jeus-web-dd.xml)をチューニングするときは、以下の事項を考慮します。

- ユーザー・ログはバッファ・サイズを増加させ、「stdout」の使用を控えます。
- JSPエンジンを使用しない場合はOFFにします。
- <enable-reload>と<check-on-demand>オプションは常にOFFにします。同オプションは、クラス変更が頻繁に起こる開発環境でのみ使用します。
- なるべくファイル・キャッシング機能を使用します。最大限多くの静的コンテンツ・ディレクトリーをファイル・キャッシング・タグに設定します。キャッシュの<max-idle-time>と<max-cache-memory>の値を大きく設定します。(無制限に設定しても構いません。)

3.7. 非同期処理プログラミング

本節では、サーブレット3.0から追加された非同期処理(Asynchronous Processing)について、アプリケーションの開発者が熟知する必要のある内容を説明します。

3.7.1. サーブレット標準による注意事項

アプリケーションの開発者がスレッドを直接扱う非同期処理プログラミングの特性上、以下のミスが発生する可能性があります。

- スレッドの過剰な作成
- 異なるスレッド同士が共有してはならないオブジェクトを共有
- 非同期処理のスレッドでjavax.servlet.RequestDispatcherを使用

この中には、テスト時には発見されず、実際のサービス中に過負荷になったときに発生する問題があります。これらはJEUSでは保証できない問題であり、アプリケーション開発者の責任となりますので注意が必要です。

参考

本節で説明する内容は、サーブレット標準をベースにして作成したものです。(JEUSにのみ該当する内容ではありません。)

スレッドの過剰な作成

スレッドを過剰に作成するとJVMの性能が低下するか、OutOfMemoryErrorが発生します。以下は、スレッドを過剰に生成した、誤った非同期処理プログラミングの作成例です。

[例 3.12] 正しくない非同期処理プログラミングの作成例(1) : <<WrongAsyncServlet1.java>>

```
import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(urlPatterns = "/WrongAsyncServlet1", asyncSupported = true)
public class WrongAsyncServlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        AsyncContext asyncContext = req.startAsync();
        Thread thread = new Thread(new TestRunnable(asyncContext));
        thread.start();
    }

    private class TestRunnable implements Runnable {
        private final AsyncContext asyncContext;

        public TestRunnable(AsyncContext asyncContext) {
            this.asyncContext = asyncContext;
        }

        @Override
        public void run() {
        }
    }
}
```

サービスの状況によっては、非同期処理が必要な要求が同時に数千件になる場合があります。その都度新しいスレッドを生成すると、数千のスレッドが同時に存在することになります。これにより、JVMの性能が深刻に低下するか、「OutOfMemoryError: unable to create new native thread」のようなエラーが発生する場合があります。したがって、なるべく`javax.servlet.AsyncContext#start(Runnable)`メソッドを使用するか、直接管理するスレッド・プールの使用をお勧めします。

```
AsyncContext asyncContext = req.startAsync();
asyncContext.start(new TestRunnable(asyncContext));
```


異なるスレッド同士が共有してはならないオブジェクトを共有

異なるスレッド同士が共有してはならないマルチ・スレッドに安全ではないオブジェクトを同時に使用すると、性能低下、デッドロック、データの整合性を侵害する恐れがあります。

以下は、異なるスレッド同士が共有してはならないオブジェクトを共有した、誤った非同期処理プログラミングの作成例です。

[例 3.13] 正しくない非同期処理プログラミングの作成例(2) : <<WrongAsyncServlet2.java>>

```
import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(urlPatterns = "/WrongAsyncServlet2", asyncSupported = true)
public class WrongAsyncServlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        AsyncContext asyncContext = req.startAsync();
        asyncContext.start(new TestRunnable(asyncContext));

        byte[] buffer = new byte[1024];
        ServletInputStream inputStream = req.getInputStream();
        inputStream.read(buffer);

        ServletOutputStream outputStream = resp.getOutputStream();
        outputStream.write(buffer);
    }

    private class TestRunnable implements Runnable {
        private final AsyncContext asyncContext;

        public TestRunnable(AsyncContext asyncContext) {
            this.asyncContext = asyncContext;
        }

        @Override
        public void run() {
            byte[] buffer = new byte[1024];
```

```

        try {
            ServletInputStream inputStream =
                asyncContext.getRequest().getInputStream();
            inputStream.read(buffer);
        } catch (IOException e) {
            //log error
            return;
        }

        ServletOutputStream outputStream;
        try {
            outputStream = asyncContext.getResponse().getOutputStream();
            outputStream.write(buffer);
        } catch (IOException e) {
            //log error
            return;
        }
    }
}
}

```

上記の例は、Requestオブジェクトから取得できるServletInputStreamまたはReaderオブジェクト、Responseオブジェクトから取得できるServletOutputStreamまたはWriterオブジェクトを、異なるスレッドで同時に使用する場合です。JEUSはサーブレット標準に準拠しており、このような動作への安定性は保証していません。**フィルターまたはサーブレットで非同期処理を開始し、別のスレッドに動作を渡した場合、その後からはフィルターおよびサーブレット・コードではなるべく非同期処理を実行するスレッド側にすべての動作を委ねる必要があります。**これは、マルチ・スレッド・プログラミングの誤った作成による性能低下、デッドロック、データ整合性の侵害を避ける方法でもあります。

startAsync()を使用して直接呼出ししていないフィルターおよびサーブレットで非同期処理が開始されたのかを確認するには、`javax.servlet.ServletRequest#isAsyncStarted()`をチェックします。

非同期処理するスレッドでjavax.servlet.RequestDispatcherを使用

非同期処理するスレッドでjavax.servlet.RequestDispatcherを使用すると、プログラムが正常に動作しないか、例外が発生する場合があります。

以下は、スレッドでjavax.servlet.RequestDispatcherを使用した、誤った非同期処理プログラミングの作成例です。

[例 3.14] 正しくない非同期処理プログラミングの作成例(3) : <<WrongAsyncServlet3.java>>

```

import javax.servlet.AsyncContext;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(urlPatterns = "/WrongAsyncServlet3", asyncSupported = true)
public class WrongAsyncServlet3 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        AsyncContext asyncContext = req.startAsync();
        asyncContext.start(new TestRunnable(asyncContext));
    }

    private class TestRunnable implements Runnable {
        private final AsyncContext asyncContext;

        public TestRunnable(AsyncContext asyncContext) {
            this.asyncContext = asyncContext;
        }

        @Override
        public void run() {
            // ..... do something

            RequestDispatcher requestDispatcher =
                asyncContext.getRequest().getRequestDispatcher("/views/report.jsp");

            try {
                requestDispatcher.forward(asyncContext.getRequest(),
                    asyncContext.getResponse());
            } catch (ServletException e) {
                // log error
            } catch (IOException e) {
                // log error
            }
        }
    }
}

```

非同期処理を実行するスレッドで、別のサーブレットまたはJSPにディスパッチする方法は、`javax.servlet.AsyncContext`で提供しています。

```

asyncContext.dispatch("/views/report.jsp");

```

3.7.2. その他の注意事項

非同期処理プログラミングに対し、標準で説明している内容以外にも注意事項が存在します。

- **AsyncContext.start(Runnable)の呼び出し時に、スレッドでjava.util.concurrent.RejectedExecutionExceptionが発生する場合**

AsyncContext.startは新しいタスクを表します。このタスクを実行するにはスレッドが必要ですが、一般的にスレッドは制限されたリソースなのでスレッド・プールを使用します。新しいタスクをスレッド・プールに割り当てたとき、当該タスクが実行できるスレッドが1つも存在しない場合は、キューに蓄積されることになります。

通常、実行時間が長いタスクの場合、スレッドの処理が遅延されるとキューに蓄積されます。しかし、キューもサイズ(デフォルト値(最大値): 4096)が制限されているため、結局新しいタスクが割り当てできない状態になります。このような状況に至ると、**java.util.concurrent.RejectedExecutionException**が発生します。そのため、アプリケーション・プログラマーはこのような状況を考慮して非同期処理のプログラミングを行う必要があります。

- **WebtoB、Apacheなど、Webサーバーと通信する方式のリスナーで非同期処理を行う場合**

WebtoB、AJP13リスナーは制限された数のTCPコネクションを使用します。WebサーバーからWASに送信するHTTP要求が一度に殺到する場合、WASの処理容量を超え、サービス不能状態に陥る可能性があります。このような方式は、同期処理をターゲットにして設計されたものです。WebサーバーからHTTP要求が送信されたら、WASはその要求が渡されたTCPコネクションに対する所有権をサーブレットに渡します。該当するサーブレットはできるだけ迅速にHTTP応答を送信し、TCPコネクションを返すことが前提です。

しかし、非同期処理はサーブレットが迅速にHTTP応答を返すことを前提することができません。WebサーバーとのTCPコネクションをすべて非同期処理に使用する場合、一般的なHTTP要求を一時的に処理できなくなります。

このような問題を解決するため、以下のような方法を考慮します。

- Webサーバーからの非同期要求に対するURLマッピングを利用してリバース・プロキシで処理します。リバース・プロキシのターゲットはHTTPリスナーです。

参考

WebtoBは4.1.6以上を使用することを推奨します。

-
- WebサーバーとのTCPコネクション数を十分に指定して、すべてのコネクションの非同期処理を防ぎます。

第4章 JSPエンジン

本章では、JSPエンジンの概念と機能および設定方法について説明します。

4.1. 概要

JSPエンジンはWebエンジンに含まれており、Webコンテキスト別に1つずつ存在します。本章では、JSP標準についての説明は省略します。JSPの作成などについての詳細情報は、JSP標準を参照してください。

JSPエンジンは、WebクライアントがJSPページを要求したとき、該当するページを見つけてサーブレットに変換する手順で、JavaファイルとSMAPファイルを生成した後、Javaファイルをコンパイルしてサービスに使用するクラス・ファイルを生成します。

JSPプリコンパイル機能を使用しなかった場合、JSPコンパイルは最初の要求時点で実行されます。そのため、最初要求の場合、OSファイル・システムへのアクセスが頻繁に発生するので、応答時間が遅延される可能性があります。

NAS(Network Attached Storage)を使用する環境において、タグとJSPインクルードの関係によってコンパイルが必要なファイルが多く存在する場合、NASに多くの要求が集中され、応答時間が遅延される現象が発生します。また、NASドライバーの動作によってjava.io.IOExceptionが発生せず、サイズが0のJSPファイルを読み込む場合もあります。Jasperでは、JVM File I/O APIを介してファイルを読み込むので、このような場合はJSPファイルに対するパッシングが失敗します。

Webコンテキスト内でのJSP

JSPファイルは、Webコンテキストのルート下に存在します。ディレクトリーを別途作成してパッケージングするか、サーブレット3.0から定義したMETA-INF/resources/ディレクトリーの下位にパッケージングすることができます。

WEB-INF/lib/下位の*.jarライブラリー・ファイルの中にMETA-INF/resources/ディレクトリーが存在する場合もJSPファイルが検索できます。ただし、WEB-INF/ディレクトリーの下位に存在するJSPファイルはサービスされません。セキュリティ上、WebクライアントがWEB-INF/下位のファイルにアクセスすることはできません。

Webコンテキストの内部構造についての詳細内容は、[「3.2.2. Webコンテキストの内部構造\(WARファイルの構造\)」](#)を参照してください。

参考

META-INF/resources/についての詳細内容は、<http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html>のリソース関連のAPI説明を参照してください。

[警告]

内部テストの結果、Oracle JDK 6で提供するjavacライブラリーがthread-safeしません。そのため、jeus.servlet.jsp.compile-java-source-concurrentlyプロパティを提供しており、デフォルト値はfalseです。要求スレッドが.javaファイルをコンパイルするときは、JVMスコープにロックを設定してから実行します。

JDKで提供するjavacライブラリーがthread-safeした場合は、以下のとおりjeus-web-dd.xmlにtrueに設定して使用します。ただし、単スレッドではなくマルチ・スレッドの状況で、JSPコンパイルが正常に動作されるのかをテストする必要があります。

```
<properties>
  <property>
    <key>jeus.servlet.jsp.compile-java-source-concurrently</key>
    <value>true</value>
  </property>
</properties>
```

4.2. Apache Tomcat Jasper

JEUSは、以前からTomcatのJSPパーサーであるJasperを導入して使用しており、パッケージ名を変更してjeus.jarに含めて提供していました。

JEUS 8からはTomcat 8ベースのJasperを使用しており、org.apache.jasperパッケージの名前をそのまま維持し、別途のjasper.jarライブラリーにパッケージングして提供します。このライブラリーは以下のパスに存在します。

```
$JEUS_HOME/lib/system/jasper.jar
```

以下は、Jasperの使用時の注意事項です。

- jasper.jarはJEUSに合わせて一部修正したものであり、Tomcatのもので上書きしてはなりません。上書きした場合はサーバー起動に失敗します。
- WebアプリケーションでWEB-INF/lib内のTomcatのjasper.jarを使用する場合、jeus-web-dd.xmlの<webinf-first>オプションをtrueに設定します。そうしないと、JEUSで使用するjasper.jarのクラスを使用することになるため、期待する動作と異なる場合があります。

Tomcat JasperとJEUS Jasperの相違点

上述のように、JEUSはTomcatが提供するJasperを一部修正して提供しているため、使用時には以下の事項を考慮する必要があります。

- JEUSは、In-memory JSP Compilation機能のようにTomcat Jasperで提供していない機能を提供します。
- JEUSでは、Tag Handler Pool、PageContext Poolを使用しません。

Javaソースのコンパイル時に64KBメソッドのサイズ制限の問題

JSPの作成時に1つの.jspファイル内に内容が多すぎる場合、これを.javaファイルに作成すると内部メソッドのサイズが64KBを超えることがあります。この場合、当該.javaファイルはJava言語の標準規約に反することになるため、Javaコンパイラでコンパイルされません。しかし、.jspの内容のほとんどがSQL文やHTMLタグのような文字列の場合は、64KB制限にかからない場合があります。アプリケーションが有するweb.xmlに以下のとおり設定します。

```
<servlet>
  <servlet-name>jeus.servlet.servlets.JspServlet</servlet-name>
  <servlet-class>jeus.servlet.servlets.JspServlet</servlet-class>
  <init-param>
    <param-name>genStringAsCharArray</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>jeus.servlet.servlets.JspServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
```

文字列ではなく、実際にJavaコードの量が多い場合はコンパイルに失敗します。そのときは、<jsp:include>を使用して応答内容を分離して実装します。以下は、<jsp:include>を使用した例です。

```
<body>
  Template Page<br/>
  <jsp:include page="module_one.jsp" />
  <jsp:include page="module_two.jsp" />
</body>
```

4.3. JSPエンジンの機能

JSPエンジンは、グレースフル・リローディング、プリコンパイル、メモリーでのJSPコンパイルおよび実行機能などを提供します。

4.3.1. JSPのグレースフル・リローディング

JSPファイルの使用目的は、ビジネス・ロジックよりユーザー・ビューに近い場合、実際のサービス運用中にも修正を希望する場合があります。そのため、JSPを多く使用するWebアプリケーションの場合、WAR形式よりはディレクトリー形式でデプロイします。

- ディレクトリー形式でデプロイした場合

ディレクトリー形式でデプロイした状態で、当該ディレクトリーの下位のJSPファイルを修正すると、JSPエンジンで以前コンパイルされたクラス・ファイルの最終修正時刻と元のJSPファイルの修正時刻を比べてコンパイルを行います。

そのとき、<check-included-jspfile>設定がtrueの場合は、要求したJSPファイルが変更されていなくても、当該JSPファイルがインクルードしたJSPファイルやTAGファイル(.tld)をチェックし、JSPファイルを再コンパイルします。

JSPエンジンはグレースフル・リローディングに対応しているため、上記のようにJSPファイルの修正により再コンパイルが発生した場合でも、以前コンパイルしたもので継続してサービスすることができます。

- WAR形式でデプロイした場合

WAR形式でデプロイする場合、WARファイルの全体を再パッケージングして再デプロイする必要があるため、サービス運用の面においてリスクが大きいです。しかし、変更するJSPファイルが多い場合は、JEUSで提供するグレースフル再デプロイを考慮することができます。グレースフル再デプロイについての詳細内容は、「[3.3.2. Webコンテキスト再デプロイ\(グレースフル再デプロイ\)](#)」を参照してください。

注意

JSPファイル内でSystem.LoadLibrary()を使ってネイティブ・ライブラリーを使用する場合、JSPリローディングに失敗することがあります。

JSPリローディングのためには、JVMの構造上、JSP別にクラス・ローダーを生成し、かつ、リローディング時にクラス・ローダーを取り替える必要があります。ネイティブ・ライブラリーの参照はJVMがクラス・ローダーで管理し、クラス・ローダー・インスタンスがGC(Garbage Collection)されるとき、当該参照を整理します。そのため、JSPのクラス・ローダーを取り替える時点に合わせてJVM GCが発生しないと、JSPリローディングに失敗することになります。JEUSではJVM GCをコントロールすることができず、JVM内部の構造によって発生する問題なので、これを解決する方法はありません。

したがって、ネイティブ・ライブラリーのロード作業は、起動時に一度だけ行うように実装することが望ましいです。なお、ネイティブ・ライブラリーを変更する場合、JVM内部でのライブラリー・リロードに対して、JEUSで保証することはできません。

4.3.2. JSPのプリコンパイル

JSPのプリコンパイルは、WebコンテキストのJSPページを事前にコンパイルできる機能です。これは、appcompilerスクリプトやコンソール・ツールの**precompile-jsp(jspc)**コマンドを使用して実行できます。

appcompilerは、オフライン状態においてもJSPプリコンパイルが可能であり、precompile-jspはJEUSにデプロイされたモジュールに対してプリコンパイルが実行できます。同機能を使用すると、JSPが最初に要求されたときの性能を高めることができます。

appcompilerまたはコンソール・ツールのprecompile-jspコマンドの詳細内容は、『*JEUS リファレンスガイド*』の「4.3. appcompiler」と『*JEUS リファレンスガイド*』の「4.2.8.21. precompile-jsp」を参照してください。

4.3.3. メモリーでのJSPコンパイルおよび実行機能

複数のWebエンジンがNASを共有し、1つのソースで同様なサービスを実行する場合、JSPのコンパイル時に必要なOSファイル・システムへのアクセス作業によってサービスの遅延およびエラーが発生する可能性があります。このような問題を解決するため、JSPのプリコンパイル、JSPのグレースフル・リローディング機能を提供していますが、より根本的に解決するためにメモリーでのJSPコンパイルおよび実行機能を提供します。

JSPエンジンは要求処理スレッドにJSPコンパイルの結果である.javaファイルおよび.classファイルを保存せず、すべてメモリーに保存して使用します。したがって、.jspファイルのメタデータや.jspファイルのコンテンツを読み込む作業以外にファイル・システムにはアクセスしません。このようにファイル・システムへのアクセスを最小限にし、JSPコンパイル・タイムに発生し得るサービスの遅延を最小化しました。

しかし、.javaファイルおよび.classファイルは別の用途のためにも必要です。このファイルは、要求処理スレッドではなく、JSPエンジン別に1つずつ持っているバックグラウンド・スレッドを利用してファイル・システムに使用します。順次的に処理するため、ファイルI/Oがファイル・システムに一気に集中する可能性は低いですが、.smapファイルは使用しません。.smapファイルが必要な場合は、既存のJSPコンパイル方式を使用する必要があります。

JSPエンジンは同機能を基本的に使用します。既存の方式を希望する場合は、jeus-web-dd.xmlに設定することができます。詳しい設定事項は、「[4.4.2. jeus-web-dd.xmlの設定](#)」を参照してください。

4.4. JSPエンジンの設定

JSPエンジンはWebAdminを使用するか、各Webアプリケーションのjeus-web-dd.xmlに設定できます。

4.4.1. Webエンジン・レベルでの設定

WebエンジンのJSPエンジンは、WebAdminを使用して設定します。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択すると、サーバーの設定画面へ移動します。設定画面で[Engine] > [Web Engine] > [Jsp Engine]メニューを選択します。

[図 4.1] JSPエンジンの設定 - メニューの移動

domain1

Domain

Session

Clusters

Servers

Applications

Security

Resources

Monitoring

Console

システム状態

0 Failed

0 Standby

2 Running

0 Shutdown

0 Suspended

0 Other

Runtime Info

LOCK & EDIT

指定したドメインの項目を変更、追加、削除する機能です。

運用マニュアル

Domain

Server もっと見る

JSP Engine

HISTORY

WebアプリケーションのJSPをコンパイルおよびサービスするために使用するJSPエンジンを設定します。

ヘルプ

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

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

Jsp Work Dir

avaで作成したJSPのファイルとそのソースファイルをコンパイルしたクラスファイルが保存される場所を設定します。

Java Compiler

java6

[デフォルト: java6] JSPのJavaソースをサーブレットクラスにコンパイルするためのJavaコンパイラを指定します。基本的にJVMに含まれているJava Compiler APIを使用します。ただし、Javaコンパイル作業は瞬間メモリ消費量が多いため、サービス環境ではメモリ問題を引き起こすことがあります。この場合は、javacを選択して別途のプロセスでコンパイルするか、Ap pCompilerを用いて予めコンパイルして使用することをお勧めします。

Compile Output Dir

JSPファイルによって作成されたクラスファイルをJsp Work Dir以外のディレクトリに格納できる設定です。設定していない場合は、クラスファイルがJsp Work Dirディレクトリに格納されます。通常、この設定は使用されません。

Compile Option

サーブレットコンパイラとして使用されるオプションです。一般的には使用されません。

Compile Encoding

JSPファイルから作成するJavaファイルのエンコーディングは、JSPファイルのエンコーディングと同一に設定します。 jeus.servlet.jsp.modern=falseの場合以外には参照しません。

Check Included Jspfile

☒

[デフォルト: true] インクルードされたJSPファイル、タグファイルの変更有無をチェックし、変更された場合は該当するJSPファイルを再コンパイルする機能です。

Keep Generated

☒

[デフォルト: true] JSPページから作成されたJavaソースファイルをコンパイルして、サーブレットクラスファイルを作成した後、Javaソースファイルを保存するか否かを決めます。このファイルはデバッグ目的で有効です。

Graceful Jsp Reloading

☐

[デフォルト: false] この設定を行う場合、指定された周期ごとにJSPファイルの変更を検知してJSPページインスタンスを新たに作成します。

Graceful Jsp Reloading Period

☐

[デフォルト: 30000] JSPのグレースフルリロードの動作周期を設定します。

Use In Memory Compilation

☒

[デフォルト: true] サービス中のjspファイルを再コンパイルする場合、.javaおよび.classファイルをメモリー上に作成してコンパイルする機能です。ただし、.classファイルの場合は、今後再起動時に再コンパイルされないように、バックグラウンドスレッドを利用してファイルシステム上に書き込みます。また、<keep-generated>がtrueの場合には、.javaファイルをバックグラウンドスレッドを利用してファイルシステム上に書き込みます。

2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。

3. [JSP Engine]画面で基本情報を設定した後、[確認]ボタンをクリックします。

[図 4.2] JSPエンジンの設定 - 基本情報の設定

domain1

Domain

Session

Clusters

Servers

Applications

Security

Resources

Monitoring

Console

システム状態

0 Failed

0 Standby

2 Running

0 Shutdown

0 Suspended

0 Other

Runtime Info

Activate Changes

Undo All Changes

変更された設定を保存、またはキャンセルする機能です。

運用マニュアル

Domain

Server [もっと見る](#)

JSP Engine

HISTORY

WebアプリケーションのJSPをコンパイルおよびサービスするために使用するJSPエンジンを設定します。

変更されました。

ヘルプ

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

動的設定 必須項目

確認 再設定

Jsp Work Dir	<input type="text"/> <p>avaで作成したJSPのファイルとそのソースファイルをコンパイルしたクラスファイルが保存される場所を設定します。</p>
Java Compiler	<input type="text" value="java6"/> <p>[デフォルト: java6] JSPのJavaソースをサーブレットクラスにコンパイルするためのJavaコンパイラを指定します。基本的にJVMに含まれているJava Compiler APIを使用します。ただし、Javaコンパイル作業は瞬間メモリ消費量が多いため、サービス環境ではメモリ問題を引き起こすことがあります。この場合は、javacを選択して別途のプロセスでコンパイルするか、Ap pCompilerを用いて予めコンパイルして使用することをお勧めします。</p>
Compile Output Dir	<input type="text"/> <p>JSPファイルによって作成されたクラスファイルをJsp Work Dir以外のディレクトリに格納できる設定です。設定していない場合は、クラスファイルがJsp Work Dirディレクトリに格納されます。通常、この設定は使用されません。</p>
Compile Option	<input type="text"/> <p>サーブレットコンパイラとして使用されるオプションです。一般的には使用されません。</p>
Compile Encoding	<input type="text"/> <p>JSPファイルから作成するJavaファイルのエンコーディングは、JSPファイルのエンコーディングと同一に設定します。jeus.servlet.jsp.modern=falseの場合以外には参照しません。</p>
Check Included Jspfile	<input type="checkbox"/> <p>[デフォルト: true] インクルードされたJSPファイル、タグファイルの変更有無をチェックし、変更された場合は該当するJSPファイルを再コンパイルする機能です。</p>
Keep Generated	<input checked="" type="checkbox"/> <p>[デフォルト: true] JSPページから作成されたJavaソースファイルをコンパイルして、サーブレットクラスファイルを作成した後、Javaソースファイルを保存するか否かを決めます。このファイルはデバッグ目的で有効です。</p>
Graceful Jsp Reloading	<input type="checkbox"/> <p>[デフォルト: false] この設定を行う場合、指定された周期ごとにJSPファイルの変更を検知してJSPページインスタンスを新たに作成します。</p>
Graceful Jsp Reloading Period	<input type="text"/> <p>[デフォルト: 30000] JSPのグレースフルリロードの動作周期を設定します。</p>
Use In Memory Compilation	<input checked="" type="checkbox"/> <p>[デフォルト: true] サービス中のjspファイルを再コンパイルする場合、.javaおよび.classファイルをメモリー上に作成してコンパイルする機能です。ただし、.classファイルの場合は、今後再起動時に再コンパイルされないように、バックグラウンドスレッドを利用してファイルシステム上に書き込みます。また、<keep-generated>がtrueの場合には、.javaファイルをバックグラウンドスレッドを利用してファイルシステム上に書き込みます。</p>

確認 再設定

以下は、設定項目についての説明です。

項目	説明
Jsp Work Dir	<p>JSPで生成されたJavaソース・ファイルが保存されるルート・ディレクトリーを指定します。指定したとしてもルート・ディレクトリーに直接ファイルを作成するではありません。JSPエンジンが属しているドメイン、サーバー名、Webアプリケーションの名前でディレクトリーを生成した後、その下位にファイルを作成します。すなわち、異なるWebエンジン間でクラス・ファイルを共有しません。</p> <p>「Compile Output Dir」項目を設定しない場合、クラス・ファイルも同じ場所に作成されます。基本的に以下の場所に作成されます</p>

項目	説明
	<p>– EARアプリケーション</p> <pre>INTERNALGENERATED_HOME/ear1/web1/___jsp_work</pre> <p>– スタンドアローンWebモジュール</p> <pre>INTERNALGENERATED_HOME/web1/___jsp_work</pre>
Java Compiler	<p>Javaコンパイラーの実行コマンドです。JSPの生成されたJavaソースをサーブレット・クラスにコンパイルするためのコンパイラーを指定します。</p> <p>基本設定が最も効率的なので、使用しないことをお勧めします</p>
Compile Output Dir	<p>JSPパーサーが生成したJavaファイルをコンパイルしたクラス・ファイルの位置です。同クラス・ファイルを実際のサービスに使用します。</p> <p>設定しない場合は、Javaファイルと同じ場所に生成されます</p>
Compile Option	Javaコンパイラーの実行オプションです
Compile Encoding	<p>JSPパーサーが生成したJavaファイルのエンコーディングを指定します。Javaコンパイラーのencodingオプションを使用します。</p> <p>JEUS 8からはこの値を参照せず、JSPページ・エンコーディングで決定された値を使用します</p>
Check Included Jspfile	<p>JSPエンジンは、基本的に要求したJSPページの変更有無だけではなく、<code><%@ include file="xxx.jsp" %></code>ディレクティブでインクルードされているすべてのJSPファイルおよびタグ・ファイルに対する変更有無を確認してコンパイルします。falseに設定すると、要求したJSPページの変更有無のみ確認してコンパイルします</p>
Keep Generated	<p>JSPパーサーが生成したJavaファイルおよびSMAPファイルを保持するオプションです。</p> <p>デバッグ時に有用であり、falseに設定するとファイルを生成した後削除するので、性能のために別途の設定はしないことをお勧めします</p>
Graceful Jsp Reloading	JSPファイルが変更された場合、指定された周期ごとにこれを検知してJSPページ・インスタンスを新しく作成します
Graceful Jsp Reloading Period	Graceful Jsp Reloadingが動作する周期をms単位で設定します
Use In Memory Compilation	サービス中のJSPファイルを再コンパイルする場合、.javaおよび.classファイルをメモリー上に作成してコンパイルする機能です。詳しい内容

項目	説明
	は、「 4.3.3. メモリーでのJSPコンパイルおよび実行機能 」を参照してください

4. 設定が終わったら設定内容を適用するために[**Activate Changes**]ボタンをクリックすると、以下の結果メッセージが確認できます。

[図 4.3] JSPエンジンの設定 - 設定の適用結果



4.4.2. jeus-web-dd.xmlの設定

JSPエンジンの設定は、jeus-web-dd.xmlでも可能です。

[例 4.1] Webコンテキストの設定ファイル：<<jeus-web-dd.xml>>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="8.0">
  <enable-jsp>true</enable-jsp>
  <jsp-engine>
    <jsp-work-dir>/home/user1/myjspwork/</jsp-work-dir>
    <java-compiler>java6</java-compiler>
    <compile-option>-g:none -verbose</compile-option>
    <compile-encoding>UTF-8</compile-encoding>
    <check-included-jspfile>true</check-included-jspfile>
    <keep-generated>true</keep-generated>
    <use-in-memory-compilation>true</use-in-memory-compilation>
  </jsp-engine>
</jeus-web-dd>
```

以下は、設定タグについての説明です。

タグ	説明
<enable-jsp>	falseに設定するとJSP機能を使用しません。JEUSデフォルト・サーブレットによってjspファイル内容でサービスします
<jsp-engine>	<p>Webコンテキストに含まれたJSPページに関する設定です。この要素は、WebAdminのJSPエンジン設定と同様です。jeus-web-dd.xmlファイルにJSPエンジンが設定されている場合、こちらで設定された内容がdomain.xmlの設定より優先します。</p> <p>WebAdminを使用したJSPエンジン設定についての詳細内容は、「4.4.1. Webエンジン・レベルでの設定」を参照してください</p>

4.4.3. JSPの下位互換性のためのWebコンテキスト・レベルのオプション設定

JEUS 4および5では、ユーザーの利便性やサーブレット2.3以前に開発されたアプリケーションのために標準ではない機能も提供し、JSPの文法チェックも最新仕様に比べ厳しくありませんでした。一方、JEUS 6からはより厳格な文法チェックやJSP 2.1機能を本格的にサポートするため、JasperベースのJSPパーサーに代替しましたが、既存のJSPに対する下位互換性のために既存のJSPパーサーもサポートします。

したがって、JEUS 4および5では問題なかったWebモジュールであってもデプロイ時に様々なエラーが発生する可能性があります。JSP 2.1などの最新仕様を使用ときは、JSPコンパイル時に発生するエラー・メッセージを確認した後、修正してからアップグレードする必要があります。

最新仕様が不要で、既存開発されたモジュールをそのまま使用する場合は、以下のとおりjeus-web-dd.xmlにJEUS 4および5に互換されるJSPパーサーが該当のWebコンテキストにのみ適用されるように設定できます。

[例 4.2] JSPの下位互換性のためのWebコンテキストの設定 : <<jeus-web-dd.xml>>

```
<properties>
  <property>
    <key>jeus.servlet.jsp.modern</key>
    <value>false</value>
  </property>
</properties>
```

jeus-web-dd.xmlに設定したオプションは、該当するWebコンテキストにのみ適用されます。Webエンジンや仮想ホスト単位でもオプションを設定することができます。Webエンジン・レベルでオプションを適用するには、以下のようなVMオプションを設定します。VMオプションの設定は、『JEUS サーバガイド』の「2.2. サーバーの追加」を参照してください。

```
-Djeus.servlet.jsp.modern=false
```

VMオプションは、仮想ホスト、Webコンテキスト・レベルでの置換えが可能であり、jeus-web-dd.xmlに設定したオプションが最終的に適用されます。jeus-web-dd.xmlにオプション設定がない場合は、上位の基本設定が適用されます。

参考

このオプションの使用は推奨していません。やむを得ず下位互換性が必要な場合のみ適用し、新規開発は新しいアプリケーションで標準に準拠して開発することをお勧めします。**この互換性オプションは、次期バージョンあるいは次のFixからは削除される可能性があります。**以外のオプションについての詳細は、『JEUS リファレンスガイド』の「1.6. Webエンジンのプロパティ」を参照してください。

第5章 仮想ホスト

本章では、仮想ホストの使用目的、規則および設定方法について説明します。

5.1. 概要

仮想ホストはインターネット・ドメイン名を基準にし、同じURLで異なるWebアプリケーションにマッピングできるようにします。すなわち、2つ以上のドメイン名(例:「www1.foo.com」and「www2.foo.com」)を1つのWebエンジンに設定し、異なるWebコンテキストをサービスすることができます。

参考

Webエンジンの観点で、WebコンテキストはWebアプリケーションと同様な意味です。

5.2. Webエンジンと仮想ホスト

本節では、仮想ホストの使用目的、規則およびServletContextオブジェクトと仮想ホストの関係について説明します。

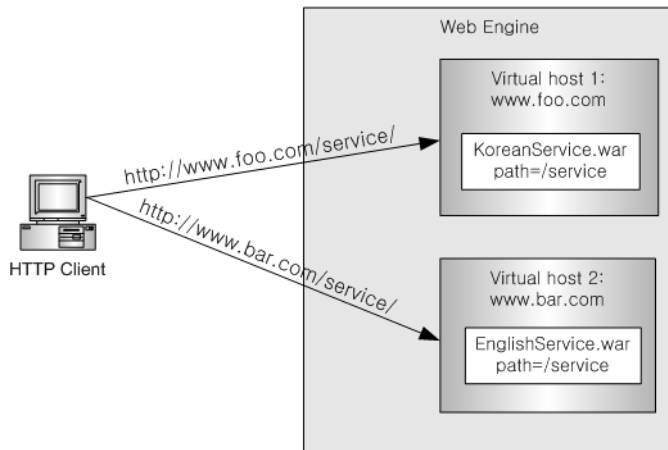
使用目的

仮想ホストにマッピングされたドメイン名を基準にし、同じURLで異なるWebアプリケーションにマッピングできます。したがって、サービス・プロバイダーは1つのWebエンジンで2つ以上のWebサイトをサービス利用者に提供することができます。これは、HTTP 1.1のホスト・ヘッダーを使用して仮想ホストを提供する機能と同様です。

仮想ホストはWebエンジンに設定できる一種のWebコンテキスト・グループです。仮想ホストがWebエンジンの構成要素としてどのように存在するのかは、[\[図 1.1\]](#)を参照してください。

以下は、仮想ホストの使用目的による利用パターンを示しています。

[図 5.1] 仮想ホストの利用パターン



上記の図では、異なる2つのアドレスで同じコンテキスト・パス(/service)にアクセスしています。実際には、1つのサーバーですが、HTTPクライアント側では「www.foo.com」と「www.bar.com」という2台のサーバーが存在するかのように認識されます。

サービス・プロバイダーは「/service」という同一のアドレス・パターンで、異なるサービスを提供することができます。上記の図では、「www.foo.com」は韓国語サービスを、「www.bar.com」は英語サービスを提供しています。

規則

以下は、仮想ホストを構成するときに適用される規則です。

- 仮想ホストには仮想ホスト名を指定します。

仮想ホスト名は設定ファイル内で仮想ホストを参照するために内部的に使用される名前として、Webエンジン内で一意である必要があります。

- 1つの仮想ホストは1つ以上のドメイン名やIPアドレスをマッピングすることができます。

JEUSではこれをホスト名といいます。異なる仮想ホストに同じホスト名をマッピングすることはできません。

- 同じ名前のWebコンテキストは異なる仮想ホストにデプロイできません。

注意

サーブレット標準では、異なる仮想ホストで同じWebコンテキストは共有できないと定義されています。

-
- 同じパスを持つ異なるWebコンテキストを異なる仮想ホストにデプロイすることができます。ただし、1つの仮想ホスト内では、同じパスを持つ2つ以上のWebコンテキストは存在できません。

Webコンテキストの名前は、Java EE標準で定義したアプリケーションまたはモジュール名を意味します。パスはWebアプリケーション内で定義するコンテキスト・ルートまたはコンテキスト・パスを意味します。Webコンテキスト名はJEUSのデプロイ・レベルで管理し、Webエンジンはコンテキスト・パスを管理します。

JEUS Webエンジンにはデフォルト仮想ホストという暗黙的な仮想ホストが存在します。Webコンテキストをデプロイする際、明示的に仮想ホストに指定しないと、デフォルト仮想ホストにデプロイされます。デフォルト仮想ホストの名前は、「DEFAULT_HOME」です。予約語であるため、別の仮想ホスト名で指定することはできません。

5.2.1. ServletContextオブジェクトと仮想ホスト

サーブレットAPIには、`javax.servlet.ServletContext.getContext(String contextPath)`というメソッドがあります。「contextPath」によって与えられたServletContextオブジェクトを返します。同メソッドは、サーブレット・コンテキストが属している仮想ホストに存在するServletContextオブジェクトを返します。

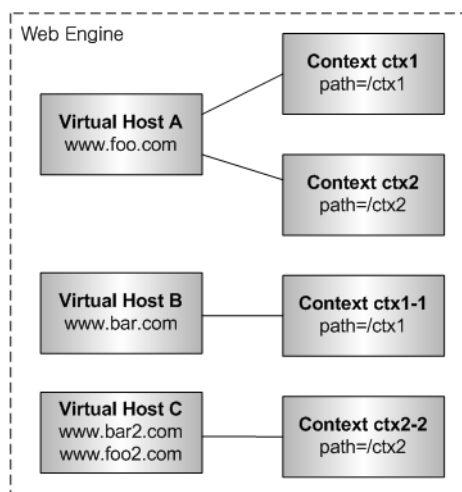
仮想ホスト内に存在しない場合は、デフォルト仮想ホストで探します。

5.3. 仮想ホストを利用したWebコンテキスト要求

本節では、URLと仮想ホスト内に存在するWebコンテキストをマッチングする方法について説明します。

以下は、Webエンジンと仮想ホスト、Webコンテキスト間の有効な関係の例です。

[図 5.2] Webエンジンと仮想ホスト、Webコンテキスト間の有効な関係の例



5.3.1. URLマッチングの例

[図 5.2]にて、それぞれのURLがマッチングされる仮想ホストとWebコンテキストは以下のとおりです。

- `http://www.foo.com/ctx1/test.jsp`

マッチングされる仮想ホスト	A
マッチングされるWebコンテキスト名	ctx1

- `http://www.foo.com/ctx2/test.jsp`

マッチングされる仮想ホスト	A
マッチングされるWebコンテキスト名	ctx2

- `http://www.bar.com/ctx1/`

マッチングされる仮想ホスト	B
マッチングされるWebコンテキスト名	ctx1-1

- `http://www.bar2.com/ctx1/test.jsp`

マッチングされる仮想ホスト	C
マッチングされるWebコンテキスト名	なし(404 Not Found)

- `http://www.foo2.com/ctx2/`

マッチングされる仮想ホスト	C
マッチングされるWebコンテキスト名	ctx2-2

注意

Webコンテキスト名とコンテキスト・パスは異なる概念です。一般的に同じ値を使用しますが、上記のように仮想ホストを使用してサービスを区別する場合は変わってきます。

5.3.2. URLマッチングの手順

URLがマッチングされる手順は以下のとおりです。

1. ホスト・ヘッダーのドメイン名とポート文字列を登録されたすべての仮想ホストにマッチングします。マッチングされた仮想ホストが存在する場合、その中からWebコンテキストを探します。

参考

仮想ホストに設定したホスト名に「IP:Port」形式でポート情報もマッピングすることができます。ポートが存在する場合は、ホスト・ヘッダーの値全体(ポートを含む)をマッチングする操作を行います。

2. Webコンテキストが見つからなかった場合は、デフォルト仮想ホストから探します。
3. デフォルト仮想ホストで目的のWebコンテキストが見つからなかった場合は、「404 Not Found」エラーが発生します。

5.4. 仮想ホストの設定

WebAdminとコンソール・ツールを使用して仮想ホストを追加、変更、削除することができます。

参考

本節の設定例で使用する名前は、便宜上「A」、「B」、「C」にしています。実際の環境では意味のある名前を使用してください。

5.4.1. 追加

WebAdminまたはコンソール・ツールを使用して仮想ホストを追加することができます。

WebAdminの使用

WebAdminを使用して仮想ホストを追加する方法は以下のとおりです。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択するとサーバーの設定画面へ移動します。設定画面で**[Engine] > [Web Engine] > [Virtual Host]**メニューを選択します。

[図 5.3] 仮想ホストの追加 - メニューの移動



2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. 仮想ホストを追加するために[Add]ボタンをクリックして、以下の項目を設定してから[確認]ボタンをクリックします。

以下は、「virtual_host1」という名前の仮想ホストを設定した画面です。

[図 5.4] 仮想ホストの追加 - 基本情報の設定

Virtual Host

HISTORY

同一URLでドメイン名をベースに複数のWebアプリケーションをサービスするための仮想ホストを設定します。

BasicResourceEngine

Web EngineJms EngineEjb Engine

BasicJsp EngineVirtual HostWeb ConnectionsAccess LogSession Config

BasicAccess Log

動的設定 必須項目

Virtual Host Name

virtual_host1

仮想ホストの内部識別名を指定します。DEFAULT_HOSTは予約語なので使用できません。

Host Name

www.foo.com

DNS名(例:「www.foo.com」)またはIPアドレスを指定します。必要に応じて、TCPポートを記述することができます。(例: www.foo.com:8088) ここに記述された文字列はWebエンジン内で一意である必要があります。

Attach Stacktrace On Error

☐

JEUSが返すエラーページにスタックトレースを添付するか否かを設定します。この設定は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。

詳細設定

すべてを開く

Properties

key=value

Encoding

Request Encoding

Response Encoding

Cookie Policy

Write Value On Header Policy

Apply Url Encoding Rule

Charset Encoding

確認再設定

項目	説明
Virtual Host Name	仮想ホストを参照するために内部的に使用する名前です。 「DEFAULT_HOST」はデフォルト仮想ホストの名前なので使用してはなりません
Host Name	ドメイン名またはIPアドレスおよびポートを含む文字列です

項目	説明
Properties	仮想ホストごとにプロパティを適用することができます。JEUSで定義したプロパティも適用可能です。詳しい内容は、『JEUS リファレンスガイド』の「1.6. Webエンジンのプロパティ」を参照してください
Encoding	仮想ホストごとにエンコーディング・オプションを適用することができます。詳しい内容は、「1.6.5. エンコーディングの設定」を参照してください
Cookie policy	仮想ホストごとにクッキー・ポリシーを適用することができます。詳しい内容は、「1.6.8. クッキー・ポリシーの設定」を参照してください

4. 設定が終わったら設定内容を適用するために[**Activate Changes**]ボタンをクリックします。
5. 仮想ホストを追加した結果が画面に表示されます。

[図 5.5] 仮想ホストの追加 - 追加の適用結果



コンソール・ツールの使用

コンソール・ツールを使用して仮想ホストを追加するときは、**add-virtual-host**コマンドを実行します。コマンドについての詳細内容は、『JEUS リファレンスガイド』の「4.2.8.7. add-virtual-host」を参照してください。

5.4.2. 変更

WebAdminまたはコンソール・ツールを使用して仮想ホストを変更することができます。

WebAdminの使用

WebAdminを使用して仮想ホストを変更する方法は以下のとおりです。

1. WebAdminの左側のメニューから[Servers]を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択するとサーバーの設定画面へ移動します。設定画面で[Engine] > [Web Engine] > [Virtual Host]メニューを選択します。仮想ホストのリストから変更する仮想ホストの名前をクリックします。([図 5.5]を参照)
2. 設定および設定変更のため、画面左側の[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。
3. [Virtual Host]画面で設定内容を変更することができます。

[図 5.6] 仮想ホストの変更 - 基本情報の変更

Virtual Host

HISTORY

同一URLでドメイン名をベースに複数のWebアプリケーションをサービスするための仮想ホストを設定します。

ヘルプ

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

Basic Access Log

動的設定 必須項目 確認 再設定

Virtual Host Name virtual_host1
仮想ホストの内部識別名を指定します。DEFAULT_HOSTは予約語なので使用できません。

Host Name www.foo.com
DNS名(例: "www.foo.com")またはIPアドレスを指定します。必要に応じて、TCPポートを記述することができます。(例: www.foo.com:8088) ここに記述された文字列はWebエンジン内で一意である必要があります。

Attach Stacktrace On Error
JEUが返すエラーページにスタックトレースを添付するか否かを設定します。この設定は開発期間には有用ですが、運用環境では無効にする必要があります。Default Error Page項目を設定して使用する場合は、このオプションは意味がありません。

詳細設定 すべてを開く

Properties

key=value

4. [Access Log]項目で仮想ホスト別にアクセス・ログを設定することができます。アクセス・ログについての詳しい内容は、1.6.10節「アクセス・ログの基本設定」を参照してください。各項目を設定した後、[確認]ボタンをクリックします。

[図 5.7] 仮想ホストの変更 - Access Logの追加設定

Access Log

HISTORY

仮想ホストレベルのアクセスログを設定します。仮想ホストのアクセスログは、基本的にWebエンジンレベルのアクセスログにも記録されます。

追加されました。

BasicResourceEngine

Web Engine

Jms Engine

Ejb Engine

Basic

Jsp Engine

Virtual Host

Web Connections

Access Log

Session Config

Basic

Access Log

動的設定

必須項目

確認

再設定

Enable

☒

[デフォルト: true] アクセスログを使用するか否かを設定します。仮想ホスト別にアクセスロガーを設定した場合、この値をfalseに設定すると、Webエンジンのアクセスロガーが使用されます。この値をfalseに設定した場合、Webエンジンのアクセスロガーはアクセスログを残しません。

Format

[デフォルト: default] Apacheで定義した共通ログ形式(Common Log Format)を使用します。default、common、combined、debugエイリアスを使用できます。各エイリアスの形式は以下のとおりです。common = "%h %l %u %t %r%W" %>s %b %W" % [Referer]i%W" % [User-agent]i%W" %", combined = "%h %l %u %t %r%W" %>s %b %W" % [Referer]i%W" % [User-agent]i%W" %", default = common + processing time (%D), debug = default + session ID + request thread name. . JEUS 6及び以前のバージョンの形式を使用したい場合は、6deprecatedに設定します。ただし、この形式は動的変更ができません。

Exclude Ext

アクセスログを残さない要求ファイルの拡張子を指定します。複数を記述する場合はコンマ(,)で区切ります。

Enable Host Name Lookup

☐

[デフォルト: false] %hフォーマットについて、IPアドレスの代わりにホスト名をロギングするか否かを設定します。この値をtrueに設定した場合、DNSルックアップによるオーバーヘッドが発生することがあります。

詳細設定

すべてを開く

Filter Class

com.tmax.logging.filter.MyFilter

Formatter Class

com.tmax.logging.handler.MyHandler

確認

再設定

Handlers

Name	Type	Level	
acccssLogFileHandler	file	FINEST	Delete

File Handler

SMTP Handler

Socket Handler

User Handler

5. 設定が終わったら設定内容を適用するために[Activate Changes]ボタンをクリックします。
6. 以下のとおり、変更内容の適用結果が画面に出力されます。変更内容を適用するにはサーバーを再起動します。

[図 5.8] 仮想ホストの変更 - Access Logの追加適用結果

Access Log

HISTORY

仮想ホストレベルのアクセスログを設定します。仮想ホストのアクセスログは、基本的にWebエンジンレベルのアクセスログにも記録されます。

ヘルプ

domain.xmlの設定を変更しました。

domain.xml : PENDING

servers.server.{? name == 'server1' }.webEngine.virtualHost.{? virtualHostName == 'virtual_host1' } : PENDING

変更内容を適用するには、サーバを再起動してください。

Basic Resource Engine

Web Engine Jms Engine Ejb Engine

Basic Jsp Engine Virtual Host Web Connections Access Log Session Config

Basic Access Log

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

Enable ☒

Format default

Exclude Ext

[デフォルト:true] アクセスログを使用するか否かを設定します。仮想ホスト別にアクセスロガーを設定した場合、この値をfalseに設定すると、Webエンジンのアクセスロガーが使用されます。この値をfalseに設定した場合、Webエンジンのアクセスロガーはアクセスログを残しません。

[デフォルト:default] Apacheで定義した共通ログ形式(Common Log Format)を使用します。default、common、combined、debugエイリアスを使用できます。各エイリアスの形式は以下のとおりです。common = "%h %l %u %t %r" %>s %b", combined = "%h %l %u %t %r" %>s %b %w "%{Referer}i" %w "%{User-agent}i" %w", default = common + processing time (%D), debug = default + session ID + request thread name. . JEUS 6及び以前のバージョンの形式を使用したい場合は、6deprecatedに設定します。ただし、この形式は動的変更ができません。

アクセスログを残さない要求ファイルの拡張子を指定します。複数を記述する場合はコンマ(,)で区切ります。

コンソール・ツールの使用

コンソール・ツールを使用して仮想ホストを変更するときは、**modify-virtual-host**コマンドを実行します。コマンドについての詳細内容は、『JEUS リファレンスガイド』の「4.2.8.17. modify-virtual-host」を参照してください。

5.4.3. 削除

WebAdminまたはコンソール・ツールを使用して不要な仮想ホストを削除することができます。

WebAdminの使用

WebAdminを使用して仮想ホストを削除する方法は以下のとおりです。

1. WebAdminの左側のメニューから**[Servers]**を選択すると、サーバー・リストの照会画面が表示されます。サーバー・リストから実行するサーバーを選択するとサーバーの設定画面へ移動します。設定画面で

[Engine] > [Web Engine] > [Virtual Host]メニューをクリックします。仮想ホストのリストから削除する仮想ホストの[Delete]ボタンをクリックします。([図 5.5]を参照)

2. 仮想ホストが正常に削除されると以下のとおり結果メッセージが出力され、リストから仮想ホストが削除されたことが確認できます。

[図 5.9] 仮想ホストの削除 - 削除の確認



3. 設定が終わったら設定内容を適用するために[Activate Changes]ボタンをクリックします。

参考

仮想ホストに接続されているWebアプリケーションが存在する場合(すなわち、当該仮想ホストを対象にしてデプロイされたアプリケーションが存在する場合)、当該仮想ホストは削除されません。そのときは、接続されている対象をWebアプリケーションから削除してから仮想ホストを削除することができます。Webアプリケーションの対象を削除する説明については、『JEUS アプリケーション&デプロイメントガイド』の「4.3.9. サービス中のアプリケーションからサービス中のサーバーを削除」を参照してください。

コンソール・ツールの使用

コンソール・ツールを使用して仮想ホストを削除するときは、**remove-virtual-host**コマンドを実行します。コマンドについての詳細内容は、『JEUS リファレンスガイド』の「4.2.8.28. remove-virtual-host」を参照してください。

第6章 WebSocketコンテナ

本章では、WebSocketコンテナの概念、機能、設定方法について説明します。

6.1. 概要

WebSocketコンテナは、Webエンジン内に含まれている形式であり、Webコンテキストごとに1つずつ存在します。JEUSで提供するWebSocketコンテナは、JSR 356、Java API for WebSocketに準拠します。したがって、サーバーのWebSocketサービスは、その標準で定義したAPIに従って作成します。ただし、本章ではWebSocket RFC6455およびJava API for WebSocket標準についての説明は省略します。

参考

1. HTML5 WebSocket APIはクライアント・ライブラリーであるため、WebtoBおよびJEUSとは関係ありません。
 2. Java API for WebSocketはJava EE 7に含まれています。したがって、Java EE 7ベースで開発する場合は、基本的なJava EE 7ライブラリーを参照して開発してください。
-

WebSocketコンテナの基本的な役割は、Webコンテキストに含まれたWebSocket Server Endpointオブジェクトをデプロイし、クライアントからWebSocketのハンドシェイク要求が送信されたとき、これにマッピングされるServer Endpointオブジェクトを接続することです。なお、クライアントとWebSocketの接続が確立したらWebSocketセッションが生成されますが、このSessionオブジェクトのライフサイクルも管理します。

サービス開発者の役割は、WebSocket Server Endpointクラス(`javax.websocket.Endpoint`)およびConfigurationクラス(`javax.websocket.server.ServerApplicationConfig`)を作成して、Webアプリケーションにパッケージングすることです。また、アノテーションを使用したPOJOスタイルで開発することもできます。

6.2. 制限事項

JEUS 8でWebSocketコンテナを使用するときは、以下のような制限事項が存在します。

- HTTPリスナーでのみ使用可能です。そのため、WebtoB 4.1.8以降バージョンのリバース・プロキシと組み合わせて使用することをお勧めします。WebtoBリバース・プロキシにおけるWebSocketサポートについての内容は、WebtoBマニュアルを参照してください。
- `jeus-web-dd.xml`の`<websocket>`項目に詳細設定が可能ですが、当該設定がなくてもWebSocketを使用することができます。

- Java API for WebSocket標準で提供するクライアントAPIはlib/system/jeus-websocket-client.jarです。他ベンダーのクライアント・モジュールを使用する場合は、上記のjarファイルのMETA-INF/service/javax.websocket.ContainerProvider内に他ベンダーのプロバイダ・クラスのフルネームを指定すると、サービスローダーを介して使用することができます。

6.3. WebSocketコンテナの機能

本節では、WebSocketコンテナの付加機能について説明します。

6.3.1. WebSocket UserProperties Failover

WebSocketコンテナは、WebSocketセッションごとにメモリー・データを格納できる領域を提供します。javax.websocket.Session.getUserProperties() APIを呼び出して取得したMapオブジェクト(以下、UserProperties)がその領域を示します。

WebSocketアプリケーション(WebSocketのサーバー・エンドポイントが含まれたWebコンテキスト)を異なるサーバーにデプロイした状況では、一方のサーバーにWebSocketを接続してUserPropertiesにデータを格納して使用しましたが、そのサーバーが異常終了した場合は、すべてのデータが消えてしまいます。このような状況を防ぐため、もう一方のサーバーにデータをバックアップしておけば、一方のサーバーが異常終了しても他のサーバーにWebSocketを接続してUserPropertiesを復元することができます。WebSocketサーバーを使用するユーザーとしては、問題なくサービスを使用できるということです。これを、**WebSocket UserProperties Failover**または**Distributed WebSocket UserProperties**といいます。

注

既存のユーザーは保存しておいたUserProperties内のデータを取得することができますが、フェイルオーバーされたWebSocketのエンドポイントは既存のサーバーとは異なるサーバーに存在するエンドポイントであり、WebSocketセッションは新規作成されます。したがって、同機能は限られたケースにのみ使用する必要があります。

この機能を使用するためには、以下の条件を満たす必要があります。

- WebSocket UserProperties FailoverはHTTPセッション・サービスをベースにして動作します。また、WebSocketセッション・クラスターに対応する必要があります。セッション・クラスターについての詳細内容は、『JEUS セッション管理ガイド』の「2.8. セッション・クラスター・モード」を参照してください。
- WebSocketアプリケーションのjeus-web-dd.xml設定ファイルに以下のとおり記述します。

```
<websocket>
  <distributed-userProperties>
    <enabled>true</enabled>
```

```
</distributed-userProperties>
</websocket>
```

すなわち、基本的にはWebSocketセッションとHTTPセッションを連携しません。

- UserPropertiesにPUTするデータは直列化可能(Serializable)である必要があります。
- WebSocket接続はHTTP要求で始まりますが、そのときWebSocketのハンドシェイク要求のクッキー・ヘッダーにJSESSIONIDが存在しなければなりません。JSESSIONIDが指すHTTPセッションはServer Endpointが含まれたWebコンテキストに作成されている必要があります。

たとえば、HTML 5 WebSocket APIを呼び出すように作成されたJSPを呼び出すと、HTTPセッションが生成され、応答クッキー・ヘッダーにJSESSIONIDが渡されます。WebブラウザでHTML 5 WebSocket JavaScriptを実行しながら、JESUSにWebSocketのハンドシェイク要求を送信しますが、その要求ヘッダーにクッキー・ヘッダーを含める必要があります。**これをサポートしないWebSocketクライアントを使用すると、WebSocket UserProperties Failover機能を使用することはできません。**

- FireFox 30.0は、HTML 5 WebSocket APIを使用するとき、WebSocketのハンドシェイク要求にクッキー・ヘッダーを付けます。

```
GET /service/chat HTTP/1.1
Connection: keep-alive, Upgrade
Cookie:
JSESSIONID=FheDy8e0bOTPO7KNqdeJ7Eps8j51CaQqcRWHvpYo9mdVw1BCSwlwrFiyrclsolkr.
amVlczcvc2VydmVyMg==
Sec-WebSocket-Key: Csv/FCQolgleZfqMtPd8+g==
Sec-WebSocket-Version: 13
Upgrade: websocket
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Gecko/20100101
Firefox/30.0 FirePHP/0.7.4
```

参考

WebtoBリバース・プロキシとこの機能を一緒に使用する場合は、リバース・プロキシ・セクションでセッション・ルーティングを設定します。詳しい内容は、WebtoBマニュアルを参照してください。

以下は、この機能を使用するときの注意事項です。

- HTTPセッションのタイムアウトとWebSocketセッションのタイムアウトは互いに独立しています。特に、WebSocketセッションの場合、HTTPセッションに依存して動作するため、WebSocketコンテナ・レベルでHTTPセッションの属性を変更することはできません。状況によっては、UserPropertiesに直列化可能なデータをPUTする時点でHTTPセッションが先にタイムアウト処理され、フェイルオーバーされない場合があります。このような状況を考慮してHTTPセッションのタイムアウトを適切に調整する必要があります。

参考

UserPropertiesに直列化可能なデータをPUTする時点で、HTTPセッションがタイムアウトされた場合には、WARNINGレベルのログを残します。

- 異なるWebコンテキスト同士において、WebSocketセッションのUserPropertiesは共有されません。

6.3.2. その他

javax.websocket.Session APIにより取得できない情報を提供

Session.getUserProperties()の戻り値であるMap<String, Object>から、以下の情報を取得することができます。

項目	説明
jeus.websocket.remoteAddr(String)	HTTP要求ヘッダーにForwardedまたはX-Forwarded-Forが存在する場合は、当該ヘッダーの値を使用します。 存在しない場合は、HttpServletRequest.getRemoteAddr()を使用します
jeus.websocket.remoteHost(String)	HTTP要求ヘッダーにForwardedまたはX-Forwarded-Forが存在する場合は、当該ヘッダーの値を使用します。 存在しない場合は、HttpServletRequest.getRemoteHost()を使用します
jeus.websocket.remotePort(String)	HTTP要求ヘッダーにForwardedまたはX-Forwarded-Forが存在する場合は、当該ヘッダーに含まれているポートを使用します。ヘッダーは存在するがポート情報がない場合、このプロパティは提供されません。 ヘッダーが存在しない場合は、HttpServletRequest.getRemotePort()を使用します。0より大きい場合のみ有効です

6.4. WebSocketコンテナの設定

WebSocketコンテナの情報は、各Webアプリケーションのjeus-web-dd.xmlに設定します。

[例 6.1] Webコンテキスト設定ファイル : <<jeus-web-dd.xml>>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="8.0">
  <websocket>
    <max-incoming-binary-message-buffer-size>8192</max-incoming-binary-message-buffer-size>

    <max-incoming-text-message-buffer-size>8192</max-incoming-text-message-buffer-size>
    <max-session-idle-timeout-in-millis>1800000</max-session-idle-timeout-in-millis>
    <monitoring-period-in-millis>300000</monitoring-period-in-millis>
    <blocking-send-timeout-in-millis>10000</blocking-send-timeout-in-millis>
    <async-send-timeout-in-millis>30000</async-send-timeout-in-millis>
    <websocket-executor>
      <min>10</min>
      <max>30</max>
      <keep-alive-time>60000</keep-alive-time>
      <queue-size>4096</queue-size>
    </websocket-executor>
    <distributed-userProperties>
      <enabled>false</enabled>
      <use-write-through-policy>false</use-write-through-policy>
    </distributed-userProperties>
    <init-param>
      <name>name</name>
      <value>value</value>
    </init-param>
    <batching-buffer-size>655536</batching-buffer-size>
    <websocket-timeout-min-threads>1</websocket-timeout-min-threads>
  </websocket>
</jeus-web-dd>
```

以下は、設定タグについての説明です。

タグ	説明
<max-incoming-binary-message-buffer-size>	クライアントから送信されるバイナリー・メッセージをバッファリングするときに使用するバッファの最大値です。 設定値より大きいメッセージが受信された場合は1009エラーを出力し、WebSocketセッションを閉じます
<max-incoming-text-message-buffer-size>	クライアントから送信されるテキスト・メッセージをバッファリングするときに使用するバッファの最大値です。 設定値より大きいメッセージが受信された場合は1009エラーを出力し、WebSocketセッションを閉じます
<max-session-idle-timeout-in-millis>	アイドル状態のWebSocketセッションを閉じる時点を決める値です。 設定値が0より大きく1000より小さい場合には、無条件1000として扱います(デフォルト値: 30分(1800000ms))

タグ	説明
<monitoring-period-in-millis>	<p>WebSocketセッションのタイムアウトをチェックするための周期を設定します。</p> <p>設定値が1000より小さい場合には、無条件1000として扱います(デフォルト値: 5分(300000ms))</p>
<blocking-send-timeout-in-millis>	<p>同時送信(Synchronous Send)を使用する場合の待機時間を設定します。</p> <p>javax.websocket.RemoteEndpoint.Basicを使用するときに適用されます(デフォルト値: 10秒)</p>
<async-send-timeout-in-millis>	<p>非同期送信(Asynchronous Send)を使用する場合、サーバーから送信できていないメッセージに対するタイムアウトです。</p> <p>javax.websocket.WebSocketContainer.getDefaultAsyncSendTimeout()でリターンされます</p>
<websocket-executor>	<p>WebSocketコンテナの内部で使用するスレッド・プールに関連する設定です。主に非同期送信を処理するために使用されます。</p> <p>以下の下位項目を設定します</p> <ul style="list-style-type: none"> – <keep-alive-time> <p>Minを超えるスレッドに対し、設定時間の間に使われていない場合は自動的にスレッド・プールから削除されます。0の場合は削除しません(デフォルト値: 1分(60000ms))</p> <ul style="list-style-type: none"> – <queue-size> <p>スレッド・プールが処理するタスクを保存するキューのサイズを指定します(デフォルト値: 4096)</p>
<distributed-userProperties>	<p>javax.websocket.Session.getUserProperties()に定義されている内容に従って提供するWebSocketセッションのフェイルオーバー関連の設定です。</p> <p>以下の下位項目を設定します</p> <ul style="list-style-type: none"> – <enabled> <p>WebSocketセッションのフェイルオーバーを使用するかどうかを設定します。HTTPセッションと連携する必要があるため、基本的には使用しません</p> <ul style="list-style-type: none"> – <use-write-through-policy>

タグ	説明
	<p>WebSocketセッションのUserPropertiesにput/removeを実行するとき、バックアップ・サーバーへの同期化が完了するまで待機するかどうかを指定します。基本的には待機せずにバックグラウンドで同期化するようにします</p>
<init-param>	<p>WebSocketコンテナで使用する追加設定です。</p> <p><name>項目にパラメータ名と<value>に設定する値を指定します</p>

第7章 JEUS WebCache

本章では、Webアプリケーションの性能向上のためにJEUS WebCacheを適用する方法について説明します。

7.1. 概要

同時要求者の増加により応答速度が低下するなど、Webアプリケーションの性能低下問題が発生した場合、ハードウェアの面およびソフトウェアの面で解決することができます。

ハードウェアの面では、サーバーを増設し、継続して受信される要求に対するロード・バランシングを行い、応答速度を高める方法があります。しかしこの方法は、サーバーの追加によるコストアップやクラスタリングなどで、サーバー運用および管理の煩わしさが発生する可能性があります。

ソフトウェアの面では、サーバーを増設することなく、Webアプリケーションで多く使われるデータをキャッシングする方法があります。この方法は、次の要求からはデータを再作成せずにキャッシングされたデータを使用するため、Webアプリケーションの応答時間を短縮し、性能を高めることができます。

本章では、JEUSシステムにおいてWebアプリケーションの性能向上のためにJEUS WebCacheをどのように適用するかについて説明します。提供されるキャッシング方法は以下のとおりです。

- JSPキャッシング

タグ・ライブラリー(Tag Library)を使用してJSP内の一部をキャッシングします。全ページのうち一部変更が発生するJSPページを要求する場合に適しています。

- HTTP応答キャッシング

HTTP応答全体をキャッシングします。静的コンテンツに対する要求に適しています。

JEUS WebCacheにキャッシングされるエントリーは、SoftReferenceで実装されています。そのため、深刻なメモリーの増加によるOutOfMemoryエラーを事前に防ぐことができます。

キャッシング機能を効果的に使用するため、頻繁に要求されるか、実行ロジックが複雑である場合、またデータベースからデータを取得する時間が長く、応答時間が長くなる可能性のあるWebページをキャッシングの対象として選択することをお勧めします。

7.2. JSPキャッシング

JSPキャッシングは、JSPのタグ・ライブラリーを使用してJSPページ内の一部をJEUS WebCacheに保存し、Webアプリケーションの性能を高める方法です。

7.2.1. 基本情報

JEUS WebCacheでは、ユーザー定義(custom)タグとして<jeus:cache>を使用します。

<jeus:cache>のJSPページ内でキャッシングしようとするコンテンツが存在する部分を入力すると、最初の要求でタグ内にボディ・コンテンツ(Body Contents)が生成されます。同コンテンツはブラウザーに転送されキャッシングされます。次の要求からはメモリーにキャッシングされたコンテンツがブラウザーに送られます。

<jeus:cache>タグを使用する基本的な形式は以下のとおりです。

```
<%@ taglib uri="http://www.tmaxsoft.com/jeuscache" prefix="jeus" %>

<jeus:cache name="..." key="..." scope="..." timeout="..."
    size="..." async="..." df="...">
    . . . Body content to be cached. . .
</jeus:cache>
```

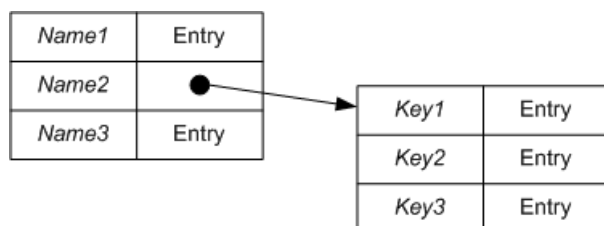
上記で記述されていないflush属性はクローズ・タグ(/>)を使用します。詳しい説明は、「[7.2.2. <cache>タグ](#)」を参照してください。

JSPキャッシングで使用するアルゴリズムはLRUです。そのため、JEUS WebCacheの最大許可数を超過すると、LRUアルゴリズムによって以前キャッシングされたエントリーが削除されます。

なお、TLD(Tag Library Descriptor)ファイルはjeus.jarファイルに含まれデプロイされます。jeuscache.tldファイルに関するURI情報をJSPエンジンに渡すために、<jeus:cache>タグ内の「taglib uri」を必ず「http://www.tmaxsoft.com/jeuscache」に明示します。

以下は、「name」属性、「name」属性+「key」属性を使ってエントリーをキャッシングするときに使用するデータ構造です。

【図 7.1】 エントリー・キャッシングに使用されるデータ構造



上記の図で、Name1とName3は「**key**」属性なしで「**name**」属性だけでタグを使用するときにエントリーがキャッシングされる方式です。「**key**」属性、「**name**」属性を両方とも使用しない場合にも同方式が使われます。しかし、「**name**」属性と「**key**」属性が両方とも使用されるタグでは、Name2のような方式でエントリーがキャッシングされます。

7.2.2. <cache>タグ

<jeus:cache>ユーザー・タグを定義したファイルはjeuscache.tldです。同ファイルはjeus-servlet.jar内に含まれており、パスは以下のとおりです。

```
jeus/servlet/cache/resource/jeuscache.tld
```

以下は、jeuscache.tldファイルの<cache>タグの設定例です。

[例 7.1] <cache>タグの設定 : <<jeuscache.tld>>

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library 1.2//EN" "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>jeuscache</short-name>
  <uri>http://www.tmaxsoft.com/jeuscache</uri>
  <display-name>JEUSCache Tag Library</display-name>
  <tag>
    <name>cache</name>
    <tag-class>jeus.servlet.cache.web.tag.CacheTag</tag-class>
    <body-content>JSP</body-content>
    <description>JEUS WebCache</description>
    <attribute>
      <name>flush</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>timeout</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>scope</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

```

</attribute>
<attribute>
  <name>name</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>size</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>key</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>async</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>df</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
</tag>
</taglib>

```

以下は、各タグを設定するときの属性についての説明です。

タグ	説明
flush	<p>キャッシュ内のエントリーを削除するために使用します。特定のエントリーを削除するには「name」属性または「name」属性 + 「key」属性を指定する必要があります。</p> <p>「name」属性に「key」属性を使用して多数のエントリーをキャッシングした場合、「name」属性だけを使用して「flush」属性を実行するとき、「key」属性を識別子とするすべてのエントリーが削除されることに注意します。なお、同属性は別の属性とは違ってボディー・コンテンツが存在しないクローズ・タグ(</>)を使用します</p>
timeout	<p>コンテンツがキャッシングされる期間を設定します。Simple Date Format形式で明示されます。(デフォルト値: 1時間)</p> <p>数字と時間を表す英数字の組み合わせで明示します。期間を表す有効な文字は「s(seconds)」、「m(minutes)」、「h(hours)」、「d(days)」、「w(weeks)」です。たとえば、</p>

タグ	説明
	<p>10sは10秒、10dは10日、4wは4週を示し、数字だけを使って設定すると秒(second)単位で認識します。</p> <p>0に設定すると、毎回ボディ・コンテンツを生成するので更新(refresh)効果が得られます。-1が設定された場合は、強制的にフラッシュされない限りコンテンツはExpireされません</p>
scope	キャッシュされるエントリーの運用範囲を表します。有効な値は「application」と「session」です。(デフォルト値: application)
name	<p>複数のページからキャッシングされたデータを共有するために使用されます。該当するスコープ内でユニークである必要があります。</p> <p>name属性を指定しなかった場合は、HTTP URI情報などを使用して内部的に生成し、管理されます。jeus:cache内のボディ・コンテンツの共有が不要な場合は、同属性を設定しない方が望ましいです</p>
size	<p>キャッシングできるオブジェクトの最大数を設定します。</p> <p>設定値を超えた場合は、LRUアルゴリズムによりキャッシュに保存されているオブジェクトが削除されます。同値は、Webアプリケーションの状況に合わせて適切に設定される必要があります。(デフォルト値: Integer.MAX_VALUE)</p>
key	<p>キャッシングされるエントリーを識別する追加的な値です。同属性は「name」属性と一緒に使用される必要があります。したがって、エントリーを識別するときは、「name」+「key」値がユニークな識別子として使われます。</p> <p>「key」属性は、以下のとおりスコープを指定する羅列形式で設定可能です。</p> <pre><jeus:cache name=". . . " key="[parameter page session request application].keyname" . . . ></pre> <p>上記の例で、「keyname」の値は要求時に設定する必要があります。「parameter」、「page」、「request」の場合はキャッシュ・ページのような要求ページで「keyname」をkey値として設定し、「application」、「session」の場合は別のページで設定できます</p> <pre>http://sample.com:8088/Sample/index.jsp?keyname=test</pre>
async	<p>1つのスレッドがエントリーをアップデート中の場合(未完了状態)、同じエントリーを要求する別のスレッドに対し、ブロッキングするか否かを設定します</p> <ul style="list-style-type: none"> – true : スレッドはブロッキングされず、以前のエントリー、すなわちアップデートされていないエントリーを取得します(デフォルト値) – false : エントリーがアップデートされるまで待機した(ブロッキング)後、最新のエントリーを取得します

タグ	説明
df	<p>オーバーフローが発生した場合、スペースを確保するためにエントリーを削除します。そのとき、設定された「size」属性に対する割合で削除されるvictim数を「df」属性として設定することができます。</p> <p>有効な値の範囲は、「0.0 <= factor <= 1.0」間の浮動小数点数です。(デフォルト値: 0.25)</p> <p>たとえば、factor 1.0はメモリーにキャッシュされているすべてのエントリーを削除するとの意味であり、factor 0.0はオーバーフローが発生する場合、追加要求に対してこれ以上キャッシュしないとの意味です</p>

7.2.3. <jeus:cache>の使用例

本節では、例を使用して<jeus:cache>タグ属性の使用方法について説明します。

以下は、cache.jspページを要求する場合、現在の日付とキャッシングされた日付を比較する簡単な例です。

[例 7.2] <jeus:cache>の使用例 : <<cache.jsp>>

```
<%@ taglib uri="http://www.tmaxsoft.com/jeuscache" prefix="jeus" %>
<HTML>
<BODY>
    Current time: <%= new Date() %><br>
    <jeus:cache timeout="60s">
        Cached time: <%= new Date() %>
    </jeus:cache>
</BODY>
</HTML>
```

<jeus:cache>タグの最初の要求では現在の日付を画面に出力し、JEUS WebCacheに保存されます。次の呼び出しからは、キャッシングされた日付が出力されます。60秒が過ぎてからは、更新された日付が出力されます。

以下は、<jsp:include>を使用して別のページをインクルードするとき、<jeus:cache>タグを使用する例です。

[例 7.3] jsp:inclueを使用した<jeus:cache>の使用例 : <<main.jsp>>

```
<HTML>
<BODY>
    <jsp:include page="cache.jsp"/>
</BODY>
</HTML>
```

main.jspにインクルードされたcache.jsp内の<jeus:cache>タグの内容は、cache.jspだけを使用する1番目の例と同様な実行結果を出力します。

7.2.4. フラッシュ機能の使用

フラッシュ機能は、キャッシングされたエントリーを強制的に削除する機能です。対象となるエントリーを指定する「name」属性または「name」属性 + 「key」属性を必ず明示します。

たとえば、以下のとおり「name」属性 + 「key」属性を使用してstock contentをキャッシングしたと仮定します。

```
<jeus:cache name="stock" key="parameter.company" scope="application">
    . . . stock content . . .
</jeus:cache>
```

キャッシングされたparameter.companyに該当するstock contentを削除するために、以下のとおり設定します。

```
<jeus:cache name="stock" key="parameter.company" scope="application" flush="true"/>
```

上記のようにフラッシュ機能が正常に実行されたら、次回からは<jeus:cache>タグ内のstock contentを要求すると、更新されたstock contentが表示されます。上記のフラッシュ機能を、以下のとおり「key」属性なしで「name」属性だけを使用する場合は、parameter.company key属性として保存されたすべてのエントリーが削除されます。

```
<jeus:cache name="stock" scope="application" flush="true"/>
```

「key」属性を使用せずに「name」属性だけでエントリーをキャッシングした場合は、「name」属性だけを使用します。

7.2.5. リフレッシュ機能の使用

すべての<jeus:cache>タグを呼び出すたびにボディー・コンテンツを生成する、リフレッシュ機能を提供します。

同機能は<jeus:cache>タグの属性を使用しません。その代わりに、「_jeuscache_refresh」をキーに、trueを値にして希望するスコープに設定することができます。

アプリケーションとセッション・スコープのすべてのエントリーをリフレッシュするためには、以下のとおり作成します。

```
<% application.setAttribute("_jeuscache_refresh", "true"); %>
<% session.setAttribute("_jeuscache_refresh", "true"); %>
```

同機能は<jeus:cache>タグにアクセスしたとき、各スコープの「_jeuscache_refresh」値を確認し、trueの場合はそのボディー・コンテンツを更新します。なお、リフレッシュ機能を使用しない場合は、「_jeuscache_refresh」をfalseに指定します。

「timeout」、「flush」属性を使用すると、1つまたは一部の更新されたエントリーが確認できますが、リフレッシュ機能を使用した場合は、設定したスコープ内のすべてのボディ・コンテンツが毎回更新されます。

7.3. HTTP応答キャッシング

JEUS WebCacheはサーブレット・フィルターを使ってHTTP応答全体をキャッシングする方法として、HTTP応答キャッシング機能を提供します。

ページの内容が動的に変更するWebページには適切ではありません。静的コンテンツに対するHTTP要求に適した方法です。画像ファイル、PDFなどのバイナリー・コンテンツを要求するHTTP要求がこれに該当されます。ただし、一定時間の間Webページの内容が変更されないか、変更内容が適用されなくても構わないWebページにはこの方法が使用できます。

```
http://www.sample.com/filter/respcacheTest.jsp?key=value
```

上記のようにkey、valueを含め、全HTTP URIがエントリー・キーとして適用されます。key、value値が変更される場合は、それぞれのURIに該当するHTTP応答がキャッシングされます。

注

HTTP応答の状態が200 OK(HttpServletResponse.SC_OK)の場合にのみ、当該HTTP応答がキャッシングされるということです。HTTP応答をJEUS WebCacheに保存するときに使われるエントリー・キーとしてHTTP URIが使用されます。

本節では、HTTP応答キャッシングをWebアプリケーションに適用する方法について説明します。

7.3.1. フィルター設定

WebアプリケーションでHTTP応答キャッシングを使用するには、以下のとおりweb.xmlにフィルターを登録します。

以下は、<url-pattern>が「/filter/」である全HTTP要求に対し、HTTP応答を10分間キャッシングする例です。

参考

<filter-class>として必ず`jeus.servlet.cache.web.filter.CacheFilter`クラスを使用します。

[例 7.4] フィルター設定 : <<web.xml>>

```
<web-app>
. . .
<filter>
  <filter-name>CacheFilter</filter-name>
  <filter-class>jeus.servlet.cache.web.filter.CacheFilter</filter-class>
```

```

<init-param>
  <param-name>timeout</param-name>
  <param-value>600</param-value>
</init-param>
<init-param>
  <param-name>lastModified</param-name>
  <param-value>on</param-value>
</init-param>
<init-param>
  <param-name>expires</param-name>
  <param-value>off</param-value>
</init-param>
</filter>

<filter-mapping>
  <filter-name>CacheFilter</filter-name>
  <url-pattern>/filter/*</url-pattern>
</filter-mapping>
. . .
</web-app>

```

以下は、フィルター・クラスに渡す初期化パラメータについての説明です。

パラメータ	説明
timeout	<p>HTTP応答をキャッシングする時間を設定します。(デフォルト値: 3600、単位: 秒(second))</p> <p>JSPキャッシングで使用する「timeout」属性および設定方法は同様です。ただし、HTTP応答キャッシングではフラッシュ機能が提供されないため、「timeout」を「-1」に設定するとWebアプリケーションがアンデプロイされない限りexpireされないのので注意が必要です。</p>
lastModified	<p>HTTP応答としてLast-Modifiedヘッダーを転送するか否かを決めます。Webエンジンの負荷を減らす目的で使われます。</p> <p>ブラウザーは自身がキャッシュしているコンテンツが、最終要求の後に変更されたか否かについてWebエンジンにリクエストすることができます。リクエストを受けたWebエンジンは、HTTP要求でIf-Modified-Sinceヘッダー情報と現在エントリーの最終変更時間を比較し、エントリーに変更事項がないとの304状態コード(HttpServletResponse.SC_NOT_MODIFIED)をブラウザーに送信します。</p> <p>有効な値は以下のとおりです</p> <ul style="list-style-type: none"> on : 最終変更時間をフィルター・チェーン(Filter Chain)の実行中に決定し、304状態コードを送信します off : HTTP応答として304状態コードを送信しません

パラメータ	説明
	<ul style="list-style-type: none"> initial : 最終変更時間を現在時間にし、304状態コードを送信します(デフォルト値)
expires	<p>HTTP応答としてExpiresヘッダー情報を転送するか否かを決めます。ブラウザでキャッシュ機能を使用している場合、キャッシュされたコンテンツは期限切れまで有効であり、以降継続的なHTTP要求に対しては、ブラウザ・キャッシュに保存されているコンテンツを使用することになります。</p> <p>JEUS WebCacheのエントリーがアップデートされた場合、ブラウザ・キャッシュに保存されているコンテンツと新しいエントリーが一致しない問題が発生します。このときは、WebエンジンでExpiresヘッダー情報を設定し、ブラウザ・キャッシュに保存されているコンテンツの使用を無効にして、JEUS WebCacheのアップデートされたエントリーを使用する必要があります。</p> <p>有効な値は以下のとおりです</p> <ul style="list-style-type: none"> on : フィルター・チェーンで値が設定されている場合、Expiresヘッダー情報を転送します off : Expiresヘッダー情報を転送しません time : HTTP応答のlast-modified値に上述したtimeパラメータ値が加えられExpiresヘッダー情報として設定され、クライアントに転送されます

参考

上記のパラメータ以外に追加で**scope**、**size**、**async**、**df**などが設定できますが、これらは「[7.2. JSPキャッシング](#)」での説明と同じ意味なので、該当する部分を参照してください。

第8章 リバース・プロキシ

本章では、Webアプリケーションの性能向上のためのリバース・プロキシの基本概念および使用方法について例を使って説明します。

8.1. 概要

リバース・プロキシは要求を処理するサーバーの手前に置かれ、サーバーへの要求を受け取ってそれを背後のサーバーに受け渡し、またその結果を受け、要求側に転送する役割をするサーバーです。リバース・プロキシはセキュリティ(実際のサーバーを外部に隠す場合)および負荷分散(複数のサーバーで要求を処理する場合)などのために用いられます。

8.1.1. 適用例

example.comという企業が、インターネットを介してアクセス可能なパブリックIPアドレスとDNSエントリーを有する、www.example.comというWebサイトを持っていると想定します。

同企業は、ファイアウォール内にパブリックIPアドレスと登録されていないDNSエントリーを有する様々なアプリケーション・サーバーを持っています。このようなネットワーク内のアプリケーション・サーバーとして「internal1.example.com」と「internal2.example.com」が存在するとします。これらのサーバーは、パブリックDNSエントリーを持っていないため、社外からは「internal1.example.com」にアクセスできず、「no such host」エラーが発生することになります。

このアプリケーション・サーバーへのWebアクセスが必要な場合、インターネットを介して直接エクスポートすることはできないため、該当するWebサーバーに統合するために内部で以下のとおりマッピングします。

- http://www.example.com/app1/any-path-here : http://internal1.example.com/any-path-hereにマッピングします。
- http://www.example.com/app2/other-path-here : http://internal2.example.com/other-path-hereにマッピングします。

リバース・プロキシを使用するために作成する主要ファイルは以下のとおりです。

ファイル名	説明
web.xml	リバース・プロキシ機能をするフィルターを設定します。サーバー内容が存在するファイルを指定します
jeus-web-dd.xml	<jeus-web-dd><context-path>設定にサービスを提供するパスを指定します

ファイル名	説明
config/data.xml	リバース・プロキシをするサーバーを指定します
config/sample.xml	data.xmlファイルを作成するためのサンプル・ファイルです

ファイルはWEB-INFディレクトリーの下位に存在します。

8.2. 使用方法

リバース・プロキシを使用するには、リバース・プロキシ機能を有効にするアプリケーションをデプロイする必要があります。本章の例では、アプリケーション名をReverseProxyと想定します。

以下は、リバース・プロキシ機能を使用するための設定手順です。

1. プロキシ・サーバーの設定
2. コンテキスト・パスの設定
3. プロキシ・フィルターの設定
4. デプロイ

8.2.1. プロキシ・サーバーの設定

リバース・プロキシを使用するために、まず、ReverseProxy/WEB-INF/config/の下位にプロキシされるサーバーの内容が含まれているdata.xmlファイルを作成します。このファイルで設定する内容は、プロキシを行う実際のサーバー内容です。プロキシするサーバーのアドレスおよび同サーバーでサービスするルールを設定することができます。実際にユーザーがプロキシ・サーバーを設定するときは、「[8.3. 設定例](#)」を参考にして項目を変更します。

以下は、サーバーの種類です。

- **<server>**

- 特定の要求に対し、1つのサーバーをプロキシするときに設定します。
- <server>は、以下のような下位設定項目を持ちます。

項目	説明
<domain-name>	<p>実際サーバーのホスト名です。</p> <p>「host name:port」または「ip address:port」などの形式で設定できます</p> <pre>www.remote.com:8088</pre>

項目	説明
<path>	<p>実際サーバーの特定ディレクトリーの下位項目をプロキシしたい場合に設定します。</p> <p><path>/content</path>と設定すると、www.remote.com:8088/contentがプロキシされます</p>
<rewrite>	絶対アドレスのリライト有無をtrueまたはfalseに設定する必要があります
<rule>	<p>サーバーに適用される要求を定義することができます。このルールに該当する要求に対し、サーバーの内容をプロキシします。</p> <p>JEUS以下の2つのルールをサポートします</p> <ul style="list-style-type: none"> – <directory-rule> : 特定のディレクトリー要求に適用されるルールです。www.proxy.comがプロキシ・サーバーであり、www.proxy.com/remote/という要求に対して特定のサーバーをプロキシしたい場合に使用します – <accept-everthing-rule> : <directory-rule>で定義されたルール以外のすべての要求に適用されるルールです。各ルールは上から順に適用されるため、このルールは最後の項目に入れる必要があります
<cluster-server>	同じ要求に対して複数サーバーがプロキシする場合に設定します。同タグの下位に<server>というタグを持っており、この設定は<server>と類似しています

data.xmlはReverseProxy/WEB-INF/config/proxy-config.dtdに以下のような内容の文法定義を持っている必要があります。このファイルの目的は、data.xmlファイルの文法を検証することです。

[例 8.1] data.xmlのDTD : <<proxy-config.dtd>>

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT config (single-server*, cluster-server*)>

<!-- single-server: プロキシに受信される要求 -->
<!ELEMENT
    single-server
    (domain-name, rewriting?, path?, (directory-rule|accept-everything-rule))>

<!-- cluster-server: プロキシに受信される要求を複数サーバーにラウンドロビンで転送する場合に設定 -->
<!ELEMENT cluster-server (server*, (directory-rule|accept-everything-rule))>

<!-- server: クラスターされた1つのサーバー -->
<!ELEMENT server (domain-name, rewriting?, path?)>

<!-- directory rule: プロキシ・サーバーに対する要求を区別するためのルールです。
ディレクトリー名で区分します。パスを「/dir」に設定すると、HOSTNAME/dirで要求に適するサーバーに渡されます。-->
```

```
-->
<!ELEMENT directory-rule (path)>

<!-- path: ディレクトリーの設定、「/」で始めます。 -->
<!ELEMENT path (#PCDATA)>

<!-- accept-everthing-rule: すべての要求に対し、該当するサーバーに要求を転送するためのルールです。
最下位に設定される必要があります。 -->
<!ELEMENT accept-everything-rule EMPTY>

<!-- domain-name: サーバーのアドレス HOSTNAME:PORT
ex) www.server1.com, www.server2.com:9999 -->
<!ELEMENT domain-name (#PCDATA)>

<!-- rewriting: プロキシされた文書内容のリンクの中で絶対アドレスをリライトするか否かを決めます。 -->
<!ELEMENT rewriting (#PCDATA)>
```

8.2.2. コンテキスト・パスの設定

WEB-INF/jeus-web-dd.xmlファイルの<jeus-web-dd><context-path>を変更すると、プロキシ・サービスを提供する<context-path>を設定することができます。「/」で設定すると該当するサーバー下のすべての要求に適用されます。設定の詳細内容は、[「3.3.1. jeus-web-dd.xmlの設定」](#)を参照してください。

8.2.3. プロキシ・フィルターの設定

JEUSは、以下の2つのフィルター・クラスを通じてリバース・プロキシ機能を提供しています。

- プロキシ・フィルター

data.xmlに定義された要求を受け、該当するサーバーの結果を渡す役割をします。

- リライト・フィルター

サーバーから受けた結果のリンクを自身のアドレスにリライトする役割をします。

たとえば、www.proxy.com/remote/index.htmlという要求に対してwww.server1.com/index.htmlを要求側に送信する場合、実際ページの内容にhref="www.server1.com/links.html"のような絶対アドレス、またはhref="/contents.html"のような「/」で始まるアドレスが存在するときは、プロキシ・サーバーのアドレスに合わせて「www.proxy.com/remote/links.html」と「/remote/contents.html」に変更される必要があります。

このようにhtml、javascript、cssなどの文書内のアドレスを変更するときに使用します。

JEUSでは、プロキシ・フィルターのみ使用するか、プロキシ・フィルターとリライト・フィルターを一緒に使用することができます。状況に合わせて、WEB-INFディレクトリーのweb.xmlファイルを以下のとおり設定します。

- プロキシ・フィルタのみ使用する場合

[例 8.2] プロキシ・フィルタのみ使用する場合 : <<web.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_3_1.xsd"
  version="3.1">

  <display-name>j2ep</display-name>
  <description>
    A J2EE application implementing a reverse proxy.
  </description>

  <filter>
    <filter-name>Proxy</filter-name>
    <filter-class>jeus.servlet.reverseproxy.ProxyFilter</filter-class>
    <init-param>
      <param-name>dataUrl</param-name>
      <param-value>/WEB-INF/config/data.xml</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>Proxy</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

- プロキシ・フィルタとリライト・フィルタを一緒に使用する場合

[例 8.3] プロキシ・フィルタとリライト・フィルタを一緒に使用する場合 : <<web.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_3_1.xsd"
  version="3.1">

  <display-name>j2ep</display-name>
  <description>
    A J2EE application implementing a reverse proxy.
  </description>
  <filter>
    <filter-name>Rewriter</filter-name>
```

```

        <filter-class>jeus.servlet.reverseproxy.RewriteFilter</filter-class>
        <init-param>
            <param-name>dataUrl</param-name>
            <param-value>/WEB-INF/config/data.xml</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>Rewriter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>Proxy</filter-name>
        <filter-class>jeus.servlet.reverseproxy.ProxyFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>Proxy</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>

```

8.2.4. デプロイ

上記の設定が終わったらアプリケーションをJEUSにデプロイします。デプロイについての詳細内容は、『*JEUS アプリケーション&デプロイメントガイド*』の「第4章 アプリケーションの作成およびデプロイ」を参照してください。

8.3. 設定例

以下は、ケース別のReverse Proxy/WEB-INF/config/data.xmlの設定例です。

ケース1

プロキシ・サーバーのホスト名がwww.proxy.comで、すべての要求に対して「www.server1.com/content」をプロキシする「www.proxy.com/index.html」を要求すると「www.server1.com/content/index.html」の内容を出力する設定は以下のとおりです。

```

<config>
    <single-server>
        <domain-name>www.server1.com</domain-name>
    </single-server>
</config>

```

```
<path>/content</path>
<rewriting>true</rewriting>
  <accept-everything-rule/>
</single-server>
</config>
```

ケース2

www.proxy.com/remoteへの要求は、www.server1.comをプロキシし、www.proxy.com/internaへの要求はwww.server2.com:8080をプロキシします。以外の要求に対してwww.server3.comをプロキシする設定は以下のとおりです。

```
<config>
  <single-server>
    <domain-name>www.server1.com</domain-name>
    <rewriting>true</rewriting>
    <directory-rule>
      <path>/remote</path>
    </directory-rule>
  </single-server>

  <single-server>
    <domain-name>www.server2.com:8080</domain-name>
    <rewriting>true</rewriting>
    <directory-rule>
      <path>/internal</path>
    </directory-rule>
  </single-server>

  <single-server>
    <domain-name>www.server3.com</domain-name>
    <rewriting>true</rewriting>
    <accept-everything-rule/>
  </single-server>
</config>
```


第9章 クラスの動的反映

本章では、Webアプリケーションの開発速度を高めるためのサーブレットの自動リロード機能について説明します。

9.1. 概要

以下は、一般的なJava EEの開発ライフサイクルです。

1. 編集
2. ビルド
3. デプロイ
4. テスト

開発者がJava EEアプリケーション、特にWebアプリケーションを開発するときにサーブレットなどのクラスを変更する場合があります。このような開発プロセスを迅速に実行するために様々な工夫がされており、WebLogic 10.3ではFastSwapを使用して再デプロイ・プロセスを短縮するための機能を提供しています。

Java EE 5では、運用中にクラス・ローダーやインスタンスを終了せずにクラスを再定義する機能が紹介されましたが、宣言されたフィールドやメソッドの変更ができないとの制約がありました。

JEUS 6まではクラスの変更後、変更されたクラスを適用するために、アプリケーションを再デプロイするか、自動リロード機能を利用して定期的なチェックまたは要求ごとにクラス・ローダーを生成して既存のクラス・ローダーと交替する必要がありました。

ただし、再デプロイの場合、アプリケーションのサイズが大きいと相当な時間がかかります。また、自動リロード機能もクラス・ローダーを再作成するので再デプロイ時と同様な負荷がかかります。

JEUS 7からは、既存のクラス・ローダーのリローディングが必要な動的反映(自動リロード)に加え、JDKのインストルメンテーション・パッケージを利用してクラス・ローダーをリローディングせずにJavaクラスの再定義が可能な、向上されたクラス動的反映(Auto Reload)機能であるJEUSホットスワップ機能を提供します。ただし現在は、Webアプリケーションのクラスのみに限定してサポートします。

注意

本章で説明するJEUSの自動リロード(JEUSホットスワップ機能の有効/無効)機能は、運用状況によっては予期せぬ負荷が発生する可能性があるため、開発段階でのみ使用することをお勧めします。

9.2. 基本設定および動作

本節では、JEUSの自動リロード機能の詳細設定および動作方式について説明します。

9.2.1. サーバー設定

JEUSホットスワップ機能を使用するには、JEUSのサーバーを開始する前にシステム・オプションを設定します。(JEUSホットスワップ機能がない自動リロードは、サーバーの開始時にシステム・オプション無しで動作します。)

システム・オプションはjeus.server.useHotSwapAgentであり、設定しない場合のデフォルトはfalseなので機能が動作しません。

以下は、JEUSの開始スクリプトの設定例です。

- startDomainAdminServerスクリプトの設定例

[例 9.1] Jeusホットスワップの設定 : <<startDomainAdminServer>>

```
...
"${JAVA_HOME}/bin/java" $VM_OPTION $SESSION_MEM
-Xbootclasspath/p:"${JEUS_HOME}/lib/system/extension.jar"
...
-Djeus.server.useHotSwapAgent=true
...
jeus.launcher.Launcher ${BOOT_PARAMETER}
...
```

- startManagedServerスクリプトの設定例

[例 9.2] Jeusホットスワップの設定 : <<startManagedServer>>

```
...
"${JAVA_HOME}/bin/java" $VM_OPTION $SESSION_MEM
-Xbootclasspath/p:"${JEUS_HOME}/lib/system/extension.jar"
...
-Djeus.server.useHotSwapAgent=true
...
jeus.launcher.ManagedServerLauncher ${BOOT_PARAMETER}
...
```


自動リロード機能を設定して開発を行い、開発が完了したら設定したオプションを削除して、運用時にはこの機能を使用しないようにします。

9.2.2. アプリケーション設定

JEUS自動リロード機能を使用するには、JEUSのjeus-web-dd.xmlに<auto-reload>を設定し、JEUSホットスワップ機能を使用するには、<auto-reload>下位の<use-jvm-hotswap>を設定します。

これらの機能はディレクトリー形式(Exploded Directory)のWebアプリケーションのクラス・ファイルの変更のみ可能です。開発中に変更されたクラスを動的に反映するためなので、すでにクラスがオープンされているディレクトリーに限定されます。すなわち、Webアプリケーション・ディレクトリーのWEB-INF/classesディレクトリー下位に存在するクラスの変更のみサポートします。

jeus-web-dd.xmlに<auto-reload>を以下のとおり設定し、自動リロードのモニタリング周期はWebAdminを使用して設定します。WebAdminを使用した自動リロードのモニタリング周期設定の詳細内容については、[「1.6.2. モニタリングの設定」](#)を参照してください。

[例 9.3] 自動リロード設定 : <<jeus-web-dd.xml>>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="7.0">
    . . .
    <auto-reload>
        <enable-reload>true</enable-reload>
        <use-jvm-hotswap>true</use-jvm-hotswap>
        <check-ok-demand>true</check-ok-demand>
    </auto-reload>
    . . .
</jeus-web-dd>
```

各タグについての詳細説明は、[「9.2.3. 設定による動作」](#)を参照してください。

9.2.3. 設定による動作

JEUSの自動リロードは、最初にアプリケーションがデプロイされた後、クラスの変更が発生した時点を基準にして動作します。

各設定によって以下のとおり動作します。

タグ	説明
<enable-reload>	自動リロード機能を使用するかどうかを設定します。<enable-reload>がtrueに設定されている必要があります
<use-jvm-hotswap>	– true : クラス・ローダーを取り替えずに、変更されたクラスを再定義(redefinition)するか、クラスを再変換(retransformation)して変更内容を適用します。変更されたクラスの再定義または再変換時に、適用できないクラス

タグ	説明
	<p>が存在する場合は、JEUSホットスワップを中断して既存の自動リロードを実行し、変更内容を反映します。JEUSホットスワップができないクラスの変更内容は、「9.2.5. JEUSホットスワップの制限事項」を参照してください</p> <p>– false : JEUSホットスワップを使用せず、アプリケーションのクラス・ローダーを取り替えて変更したクラスを適用します</p>
<check-on-demand>	<p>– true : 要求が受信されたとき、直ちに該当するアプリケーションのクラス・ローダーを基準にしてクラスの最終変更時間をチェックし、変更されたファイルを検索します。ファイルをチェックした後、変更されたクラスが存在する場合は、<use-jvm-hotswap>の設定有無に応じて自動リロードが実行されます</p> <p>– false : Webエンジンが「Check Class Reload」設定に従って、一定周期ごとに(基本的に300秒)該当するアプリケーション・ローダーを基準にしてクラスの最終変更時間をチェックして変更されたファイルを検索します。「Check Class Reload」設定についての詳細内容は、「1.6.2. モニタリングの設定」を参照してください。</p> <p>ファイルをチェックした後、変更されたクラスが存在する場合は、<use-jvm-hotswap>の設定有無に応じて自動リロードが実行されます</p>

9.2.4. JEUSホットスワップでサポート可能なアプリケーションおよび変換

JEUSホットスワップは、Explodedディレクトリー内のPOJO(Plain Old Java Object)、Webアプリケーション・クラスをサポートします。

JEUSホットスワップは、以下のタイプ変更をサポートします。

- 静的クラス構築子の追加/削除
- 一般的なクラス構築子の追加/削除
- 静的メソッド・ボディの変更
- 一般的なメソッド・ボディの変更

以下は、JEUSホットスワップでサポートするクラス変換リストです。

- **Javaクラス・インスタンス(non-astract)**

Javaの変換タイプ	対応可否	Notes
メソッドの追加/削除	非対応	

Javaの変換タイプ	対応可否	Notes
フィールドの追加/削除(1)	非対応	
メソッド・ボディの変更(2)	対応	
構築子の追加/削除(3)	対応	
フィールドの修飾子(field modifier) (4)	非対応	

- Staticクラス

Javaの変換タイプ	対応可否	Notes
メソッドの追加/削除	非対応	
メソッド・ボディの変更	対応	

- Abstract Javaクラス

Javaの変換タイプ	対応可否	Notes
abstractメソッドの追加/削除	非対応	
Javaクラス・インスタンスの1-4変化	対応	

- final Javaクラス

Javaの変換タイプ	対応可否	Notes
Javaクラス・インスタンスの1-4変化	対応	

- final Javaメソッド

Javaの変換タイプ	対応可否	Notes
Javaクラス・インスタンスの1-4変化	対応	

- final Javaフィールド

Javaの変換タイプ	対応可否	Notes
Javaクラス・インスタンスの1-4変化	対応	

- 列挙型(enum)

Javaの変換タイプ	対応可否	Notes
定数の追加/削除	非対応	
メソッドの追加/削除	非対応	

- 匿名内部クラス

Javaの変換タイプ	対応可否	Notes
フィールドの追加/削除	有効でない	Java言語で提供されません
メソッドの追加/削除	非対応	

- 静的内部クラス

Javaの変換タイプ	対応可否	Notes
Javaクラス・インスタンスの1-4変化	対応	

- 一般的な内部クラス

Javaの変換タイプ	対応可否	Notes
Javaクラス・インスタンスの1-4変化	対応	

- Javaインターフェース

Javaの変換タイプ	対応可否	Notes
メソッドの追加	非対応	

- Javaリフレクション

Javaの変換タイプ	対応可否	Notes
存在するフィールド/メソッドのアクセス	対応	
新しいメソッドのアクセス	非対応	新しいメソッドはリフレクションを使用すると見えません
新しいフィールドのアクセス	非対応	新しいフィールドはリフレクションを使用すると見えません

- Annotations on Classes

Javaの変換タイプ	対応可否	Notes
メソッド、属性、アノテーションの追加/削除	非対応	

- アノテーションのタイプ

Javaの変換タイプ	対応可否	Notes
メソッド、属性、アノテーションの追加/削除	非対応	

- Exceptionクラス

Javaの変換タイプ	対応可否	Notes
Javaクラス・インスタンスの1-4変化	対応	

- EJBインターフェース

Javaの変換タイプ	対応可否	Notes
メソッドの追加/削除	非対応	リフレクションと関連するEJBインターフェースの変化はサポートされません

- EJB 3.0 Session/MDB、実装クラス

Javaの変換タイプ	対応可否	Notes
メソッド、フィールドの追加/削除	非対応	EJBによって参照されるクラスの中から、サポートされるクラスに限り変更がサポートされます

- EJB 2.X Entity Bean

Javaの変換タイプ	対応可否	Notes
メソッド、フィールドの追加/削除	非対応	EJBによって参照されるクラスの中から、サポートされるクラスに限り変更がサポートされます

- EJB Interceptors

Javaの変換タイプ	対応可否	Notes
メソッド、フィールドの追加/削除	非対応	EJBによって参照されるクラスの中から、サポートされるクラスに限り変更がサポートされます

9.2.5. JEUSホットスワップの制限事項

JEUSホットスワップでサポートされる変換とサポートできない変換があります。サポートできない変換が存在する場合はJDKで`UnsupportedOperationException`が発生し、JEUSはこれを受け「Retransforming all modified classes in the servlet context [AAA] failed.」のようなログを残します。この場合、アプリケーションの動的な再定義は反映されませんが、JEUSの自動リローディング動作は継続されます。

サポートできない変換は以下のとおりです。

- Javaリフレクションの結果は、新しく変更されたフィールドとメソッドを含めません。

したがって変更されたクラスのAPIのリフレクションを使用すると、予期しない動作が発生する場合があります。

- すでに存在するクラスの階層変更はサポートされません。

たとえば、クラスのインターフェースを実装したリストの変更やクラスのスーパークラス(superclass)を変更する場合はサポートされません。

- Javaアノテーションの追加および削除はサポートされません。上述したリフレクションがその理由です。

- EJBインターフェースのメソッドの追加/削除はサポートされません。EJB編集は変更の反映(Reflect)が必要であるためです。

- 列挙型(Enums)の定数を追加/削除する場合はサポートされません。

- finalizeメソッドの追加/削除の場合はサポートされません。

第10章 実習

以下により、JEUSを簡単に体験することができます。以下の例では、JEUSのサーバー名を「server1」としています。ユーザーはこのサーバー名をJEUSがインストールされたマシンの名前に変更する必要があります。

1. システムにJEUSの正常なインストールとパスおよびシステム変数が適切に設定されたのかを確認します。(特に、システム・パスに「JEUS_HOME/bin/」ディレクトリーが含まれているのかを確認します。)

参考

1. 本章での「JEUS_HOME」は、JEUSの実際のインストール・ルート・ディレクトリーを意味します。

(例: 「/home/user/jeus8」)

2. JEUSのインストールについては、『「JEUS インストール&スタートガイド」』を参照してください。

-
2. MS(Managed Server)内のWebエンジン(サーブレット・エンジン)に少なくとも1つのWebコネクションが設定されている必要があります。Webコネクション設定については、「[第2章 Webコネクションの管理](#)」を参照してください。

「[2.3.3. HTTPリスナーの設定](#)」でのようにポートが「8088」に設定された同じHTTPリスナーを追加して簡単に実行することができます。

3. 設定が完了したらコマンド画面を実行して以下のとおり入力すると、JEUSランチャーが実行されます。「-u」と「-p」は、JEUSのインストール時に設定した管理者名とパスワードです。

以下の例では、管理者名とパスワードを「jeus」に指定しています。

```
startDomainAdminServer -domain domain1 -server server1 -u jeus -p jeus
```

4. 別のコマンド画面を開いて、「**jeusadmin -u jeus -p jeus**」を実行します。(-u、-pを入力していない場合は、再び「login」を入力します。)

```
$ jeusadmin
JEUS8 Administration Tool
To view help, use the 'help' command.
offline>login
```

5. 接続するサーバーのアドレス、管理者名とパスワードを入力するとコンソール・ツールが開始されます。入力する情報は、JEUSのインストール時に設定した情報と同じである必要があります。

```

offline>login
Enter the server address: localhost
User name: jeus
Password:
Attempting to connect to 127.0.0.1:9736.
The connection has been established to Domain Administration Server adminServer
in the domain domain1.
[DAS]domain1.adminServer>

```

6. 以下のようにコマンドを実行すると、Webエンジンをモニタリングおよび制御できるコマンド・リストが出力されます。

```

[DAS]domain1.adminServer>help -g Web
[ Web]_____
add-response-header      Add an HTTP response custom header.
add-tmax-connector       Add Tmax Connector.
add-virtual-host         A virtual host was dynamically added to
                        domain.xml, but this was only applied to
                        the XML file, not to a server.
add-web-cookie-policy    Add the cookie policy configuration.
add-web-encoding         Add web engine charset encoding.
add-web-listener         Add HTTP listener, AJP13 listener, or TCP
                        listener. If the type is not specified,
                        anHTTP Listener will be added by default.
add-web-properties       Add web engine properties.
add-webtob-connector     Add the WebtoB Connector.
clear-web-statistics     Resets the web engine statistics.
list-session             Show session list sorted by idle time.
modify-jsp-engine        Modify JSP engine configurations.
modify-response-header   Modify the HTTP response custom header.
modify-session-configuration Modifies the session configuration.
modify-tmax-connector    Modify the thread pool number of the
                        tmax-connector.
modify-virtual-host      Modify the access log format of a virtual
                        host dynamically. The access log must be
                        enabled.
modify-web-cookie-policy Modify the cookie policy configuration.
modify-web-encoding      Modify the web engine charset
                        encoding configurations.
modify-web-engine-configuration Modify some parts of the web engine
                        configuration dynamically.
modify-web-listener      Modify the thread pool number of the web
                        listener (http-listener, tcp-listener, or
                        ajp13-listener).
modify-web-properties    Modify web engine properties.
modify-webtob-connector  Modify the thread pool number of the

```


	webtob-connector.
reload-web-context	Forcibly reloads the servlet context.
remove-response-header	Remove the HTTP response custom header.
remove-session	Remove session.
remove-tmax-connector	Remove the tmax-connector.
remove-virtual-host	A virtual host was dynamically removed from domain.xml, but this was only applied to the XML file, not to a server.
remove-web-cookie-policy	Remove the HTTP cookie policy configuration.
remove-web-encoding	Remove the web engine charset encoding configurations.
remove-web-listener	Remove a web listener (http-listener, tcp-listener, or ajp13-listener).
remove-web-properties	Remove web engine properties.
remove-webtob-connector	Remove webtob-connector.
resume-web-component	Temporarily resumes the given component (servlet element of context or given web-connection name).
show-request-processing-flow	Shows the request processing flow of mapped URL patterns and the specified hostname.
show-session-configuration	Shows the session configuration.
show-web-engine-configuration	Show web engine configurations, including the monitoring period and access-logs.
show-web-statistics	Shows the web engine statistics.
suspend-web-component	Temporarily suspends the servlet.
precompile-jsp	Precompile JSP files for a deployed web module. Connect to Domain Administration Server or a server.
To show detailed information for a command, use 'help [COMMAND_NAME]'. ex) help connect	

7. Webエンジンの制御コマンドおよびモニタリング・コマンドを実行します。Webエンジン・コマンドについては、『*JEUS リファレンスガイド*』の「4.2.8. Webエンジン関連コマンド」を参照してください。

8. WebAdminを使用しても同様な作業が実行できます。WebAdminを使用した実行方法については、『*JEUS WebAdminガイド*』の「2.2. ログイン」を参照してください。

上記の実習で問題が発生した場合は、JEUSの環境設定を確認し、正しく再設定してください。問題の原因を詳しく調べるためには、JEUS管理者のコンソール・ウィンドウ・ログに記録されている情報を参照する方法もあります。

参考

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

索引

シンボル

[Activate Changes], 12

[LOCK & EDIT], 12

A

Active Timeout Notification, 101

AJPリスナー, 59, 60

Allowed Server, 72, 80, 84

Apache Tomcat Jasper, 156

appcompiler, 159

C

cache.jsp, 192

Check Class Reload, 18

Check Session, 18

Check Thread Pool, 18

Common Log Format, 41

Compression, 68

Configurationクラス, 179

Connection Type, 64

D

data.xml

<cluster-server>, 199

<server>, 198

<server><domain-name>, 198

<server><path>, 199

<server><rewrite>, 199

<server><rule>, 199

<server><rule><accept-everthing-rule>, 199

<server><rule><directory-rule>, 199

Default virtual host, 169

Dispatcher Config Class, 83, 95

Distributed session information, 7

Distributed WebSocket UserProperties, 180

E

EJBs, 144

Enable Server Push, 80

F

Filters, 144

G

General information of web module, 143

H

Hth Count, 87

HTTPリスナー, 59, 61

HTTP応答キャッシング, 187, 194

HTTP応答キャッシングのフィルター設定

<filter-class>, 194

jeus.servlet.cache.web.filter.CacheFilter, 194

I

Interrupt Thread, 101

J

Java EE Webアプリケーション, 1

javax.servlet.ServletContext.getContext(String
contextPath), 169

JEUS HotSwap

Enum, 209

Jeus web deployment descriptor, 144

jeus-web-dd.xml

<added-classpath>, 129

<aliasing>, 129

<allow-indexing>, 129

<async-config>, 130

<attach-stacktrace-on-error>, 130

<auto-reload>, 128

<auto-reload><enable-reload>, 207, 208

<context-path>, 128

<cookie-policy>, 130

<deny-download>, 129

<ejb-ref>, 139

<enable-jsp>, 128

- <encoding>, 130
- <file-caching>, 129
- <jsp-engine>, 129
- <keep-alive-error-response-codes>, 130
- <library-ref>, 132
- <principal>, 138
- <properties>, 131
- <redirect-strategy-ref>, 137
- <res-env-ref>, 139
- <res-ref>, 139
- <role-mapping>, 138
- <role-permission>, 138
- <role>, 138
- <session-config>, 129
- <user-log>, 128
- <web-security>, 130
- <webinf-first>, 129
- <websocket>, 131

jeus.servlet.jsp.compile-java-source-concurrently, 156

jeus.servlet.security.RedirectStrategyインターフェース, 135

jeus.servlet.security.RemoveCrLfRedirectStrategy, 137

jeusadmin, 6

JEUSホットスワップ, 208

- Abstract Javaクラス, 209
- Annotations on Classes, 210
- EJB 2.X Entity Bean, 211
- EJB 3.0 Session/MDB, 実装クラス, 211
- EJB Interceptors, 211
- EJBインターフェース, 211
- Exceptionクラス, 211
- final Javaクラス, 209
- final Javaフィールド, 209
- final Javaメソッド, 209
- Javaインターフェース, 210
- Javaクラス・インスタンス(non-abstract), 208
- Javaリフレクション, 210
- Staticクラス, 209
- 匿名内部クラス, 210
- 一般内部クラス, 210
- アノテーションのタイプ, 210
- 静的内部クラス, 210

jsf-injection-provider.jar, 133

JSF、JSTLライブラリーの設定, 132

JSP , 155

JSPPエンジンの設定

- Check Included Jspfile, 162
- Compile Encoding, 162
- Compile Option, 162
- Compile Output Dir, 162
- Graceful Jsp Reloading, 162
- Graceful Jsp Reloading Period, 162
- Java Compiler, 162
- Keep Generated, 162
- Use In Memory Compilation, 162

JSPのプリコンパイル, 159

JSPエンジンの設定

- <enable-jsp>, 164
- <jsp-engine>, 164
- Jsp Work Dir, 161

JSPキャッシング, 187

L

Limit, 78

Listeners, 144

M

main.jsp, 193

Max Concurrent Streams, 80

Max Header Count, 67

Max Header Size, 67

Max Keep Alive Request, 79

Max Parameter Count, 67

Max Query String Size, 68

Max Thread Active Time, 100

Memory information, 8

N

Name, 63

NAS(Network Attached Storage), 155

Notify Subject, 100

Notify Threshold Ratio, 100

Number, 89

O

Output Buffer Size, 65

P

Port, 95

Postdata Read Timeout, 66

precompile-jsp, jspc, 159

R

Read Timeout, 87

Reconnect Count For Backup, 96

Reconnect Interval, 90, 96

Registration Id, 87

Request Encoding, 24, 29

Request information, 8

Request Prefetch, 90

Request Url Encoding, 23, 28

ResourceServlet, 135

Response encoding, 25, 30

Restart Subject, 101

Restart Threshold Ratio, 100

S

sample.SimpleAccessLoggerFilter, 44

Server Access Control, 72, 79

Server Group Name, 95

Server Listener Ref, 64

Server Name, 95

Server Type, 96

Servers Adccess Control, 84

Servlets, 143

SETTINGS Frame Ack Timeout, 80

Settings Max Frame Size, 80

Standard session information, 7

T

TCP Dispatcher Configuration Class, 110

TCPクライアント, 109

TCPハンドラー, 110

TCPリスナー, 59, 61, 110

Thread information, 8

Thread Pool, 64, 66

Auto Tuning, 78

Max, 65

Max Queue, 65

Max Wait Queue, 65

Maximum Idle Time, 65

Min, 65

Step, 65

Thread State Notify, 65

Tmax Address, 95

Tmax Backup Address, 96

Tmax Backup Port, 96

Tmax Version, 95

Tmaxコネクター, 59, 62

W

WAR(Web ARchive)ファイル, 126

Web Connections, 2

web.xml

<ejb-ref>, 139

<resource-env-ref>, 139

<resource-ref>, 139

expires, 196

lastModified, 195

timeout, 195

WebAdmin, 6

WebSocket Server Endpointクラス, 179

WebSocket UserProperties Failover, 180

WebSocketコンテナ, 179

WebtoBコネクター, 59, 61

Webエンジン, 1

Webエンジンのアクセス・ログ, 37

Webエンジンのモニタリング, 6

Webエンジンの制御, 6

Webエンジンの設定, 13

Webコンテキスト, 2

Webコンテキストのモニタリング

application-infoコマンド, 144

Webコンテキストのリロード

reload-web-contextコマンド, 145

Webコンテキストの一時停止

suspend-web-componentコマンド, 148

Webコンテキストの再開

 resume-web-componentコマンド, 148

Webコンテキスト内でのJSP, 155

Webサーバー, 3

WJP Version, 87

Worker Thread Pool, 62

workers.properties

 worker.<worker_name>.balance_workers, 104

 worker.<worker_name>.host, 105

 worker.<worker_name>.port, 105

 worker.<worker_name>.reference, 105

 worker.<worker_name>.route, 105

 worker.<worker_name>.sticky_session, 105

 worker.<worker_name>.type, 104

 worker.list, 104

X

Xaresource Class, 96

XML設定ファイル

 domain.xml, 15

 jeus-web-dd.xml, 15

 web.xml, 15

 webcommon.xml, 15

あ

アクセス・ログ, 4

アクセス・ログのフォーマット設定, 41

か

仮想ホスト, 2, 40

仮想ホストの設定

 Cookie policy, 174

 Encoding, 174

 Host Name, 173

 Properties, 174

 Virtual Host Name, 173

仮想ホスト名, 168

カスタマイズされた通信プロトコルのヘッダー, 110

カスタマイズされた通信プロトコルのボディ, 111

クライアント, 3

クラス・ローダーのモニタリング, 3

コンソール・ツール, 6

さ

静的コンテンツ, 125

その他のWebエンジンの設定, 2

サーバー・ログ, 4, 37

スレッド・プールのモニタリング, 3

セッション・サービスのモニタリング, 3

セッション管理サービス, 2

た

動的コンテンツ, 125

は

フォーマット・エイリアス

 combined, 41

 common, 41

 debug, 42

 default, 41

フラッシュ機能, 193

プロキシ・フィルター, 200

プロトコル, 109

ホスト名, 168

ま

メッセージ, 109

モニタリング・スレッド, 2

や

ユーザー・ログ, 5

ら

リスナー, 60

リスナーのチューニング, 123

リバース・プロキシ, 197

リフレッシュ機能, 193

リライト・フィルター, 200

ログ・ローテーション, 37