HITACHI
Inspire the Next

Cosminexus V9 アプリケーションサーバ 機能解説 互換編

解説書

3020-3-Y12-60

■ 対象製品

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

■ 輸出時の注意

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

■ 商標類

Active Directory は，米国 Microsoft Corporation の，米国およびその他の国における登録商標または商標です。
AX2000 は，A10 Networks, Inc.の商品名称です。
BIG-IP，3-DNS，iControl Services Manager，FirePass　および F5 は F5 Networks, Inc. の商標または登録商標です。
BSAFE は，米国 EMC コーポレーションの米国およびその他の国における商標または登録商標です。
CORBA は，Object Management Group が提唱する分散処理環境アーキテクチャの名称です。
gzip は，米国 FSF(Free Software Foundation)が配布しているソフトウェアです。
HP-UX は，Hewlett-Packard Development Company,L.P.のオペレーティングシステムの名称です。
IBM，AIX は，世界の多くの国で登録された International Business Machines Corporation の商標です。
IIOP は，OMG 仕様による ORB(Object Request Broker)間通信のネットワークプロトコルの名称です。
Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
Microsoft は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
MyEclipse は，米国 Genuitec 社の商品名称です。
OMG, CORBA, IIOP, UML, Unified Modeling Language, MDA, Model　Driven　Architecture は, Object Management Group, Inc.の米国及びその他の国における登録商標または商標です。
Oracle と Java は，Oracle Corporation 及びその子会社，関連会社の米国及びその他の国における登録商標です。
RSA は，米国 EMC コーポレーションの米国およびその他の国における商標または登録商標です。
SOAP（Simple Object Access Protocol）は，分散ネットワーク環境において XML ベースの情報を交換するための通信プロトコルの名称です。
すべての SPARC 商標は，米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標がついた製品は，米国 Sun Microsystems, Inc. が開発したアーキテクチャに基づくものです。
SQL Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
UNIX は，The Open Group の米国ならびに他の国における登録商標です。
Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Vista は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
その他記載の会社名，製品名は，それぞれの会社の商標もしくは登録商標です。
Eclipse は，開発ツールプロバイダのオープンコミュニティである Eclipse Foundation, Inc.により構築された開発ツール統合のためのオープンプラットフォームです。
This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

■ 発行

2015 年 4 月 3020-3-Y12-60

■ 著作権

All Rights Reserved. Copyright (C) 2012, 2015, Hitachi, Ltd.

変更内容

変更内容(3020-3-Y12-60) uCosminexus Application Server 09-70，uCosminexus Application Server(64) 09-70，uCosminexus Client 09-70，uCosminexus Developer 09-70，uCosminexus Service Architect 09-70，uCosminexus Service Platform 09-70，uCosminexus Service Platform(64) 09-70

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

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

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

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

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

目次

1 アプリケーションサーバの機能 1

1.1 機能の分類 2

1.1.1 アプリケーションの実行基盤としての機能 4

1.1.2 アプリケーションの実行基盤を運用・保守するための機能 5

1.1.3 機能とマニュアルの対応 6

1.2 互換のための機能の一覧 9

1.2.1 動作モードに関する機能 9

1.2.2 基本・開発機能に関する互換機能 9

1.2.3 拡張機能に関する互換機能 9

1.3 このマニュアルに記載している機能の説明 10

1.3.1 分類の意味 10

1.3.2 分類を示す表の例 10

1.4 アプリケーションサーバ 09-70 での主な機能変更 12

1.5 推奨機能との対応 14

2 ベーシックモード 17

2.1 この章の構成 18

2.2 ベーシックモードの概要 19

2.3 使用できる機能の一覧 20

2.3.1 使用できる Java EE および J2EE の機能 20

2.3.2 使用できるアプリケーションサーバの機能 20

2.4 ベーシックモードでのリソース接続 32

2.4.1 接続できるリソースとトランザクションの管理方法 32

2.4.2 データベース接続の前提条件 32

2.4.3 データソースによるデータベースへの接続で使用できる機能 33

2.4.4 リソースへの接続テスト 38

2.4.5 JTA によるトランザクション実装時の注意事項 39

2.5 プロセスの依存関係とプロセスの再起動方法 40

2.5.1 ベーシックモードの場合のプロセスの依存関係 40

2.5.2 ベーシックモードの場合のプロセスの再起動方法 40

2.6 データソースから DB Connector への移行方法 42

2.7 パフォーマンスチューニング 47

2.7.1 チューニング項目 47

2.7.2 データベースへのアクセス方法を最適化するためのチューニングパラメタ 48

2.8 データベースとアプリケーション間の文字コード変換（ベーシックモード） 50

2.9 下位互換用のクラスローダ構成 51

2.10　ベーシックモードの設定　53
2.10.1　データソースの設定　53
2.10.2　データベースと接続するための設定（データソースを使用する場合）　55
2.11　ベーシックモードで使用するファイル　60
2.11.1　usrconf.properties（J2EE サーバ用ユーザプロパティファイル）　60
2.11.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）　62
2.11.3　データソース設定ファイル　62
2.11.4　データソース属性ファイル　63
2.11.5　コネクションプーリングとコネクションスイーパの動作　65
2.11.6　データソースへのリファレンス定義　67
2.12　ベーシックモードでデータベースに接続する場合の運用管理（互換用機能）　70
2.12.1　データソース管理　70
2.12.2　データソースの稼働情報監視　75
2.13　J2EE サーバで使用するリソース操作コマンド　80
cjclosecn（J2EE サーバのコネクションクローズまたはコネクションの状態表示）　80
2.14　ベーシックモード使用時の注意事項　83
2.14.1　サーブレットおよび JSP 実装時共通の注意事項　83
2.14.2　Stateful Session Bean 実装時の注意事項　83

3　サーブレットエンジンモード　85
3.1　この章の構成　86
3.2　使用できる機能の一覧　87
3.2.1　使用できる Java EE および J2EE の機能　87
3.2.2　使用できるアプリケーションサーバの機能　87
3.3　サーブレットエンジンモードと J2EE サーバモードの比較　95
3.3.1　システムの構築方法　95
3.3.2　Web アプリケーションのデプロイと設定　96
3.3.3　Web サーバの使用方法　97
3.3.4　データベースとの接続　98
3.3.5　Web アプリケーションに組み込むサーブレットフィルタ　100
3.3.6　Web アプリケーションのリロード　100
3.3.7　システムの起動および停止方法　103
3.3.8　使用するメモリ，スレッド数，ポート番号　103
3.4　サーブレットエンジンモードで使用できる Web コンテナの機能　105
3.4.1　Web アプリケーションの実行　107
3.4.2　Web アプリケーションの更新検知とリロード　109
3.4.3　JSP ファイルのコンパイル結果の保持　117
3.4.4　アプリケーションのイベントリスナ　119
3.4.5　リクエストおよびレスポンスのフィルタリング　119
3.4.6　Web コンテナのセキュリティ管理　120

3.4.7 EJB コンテナとの連携 121
3.4.8 データベースとの接続 121
3.4.9 Web コンテナによるスレッドの作成 122
3.4.10 同時実行スレッド数の制御 123
3.4.11 ユーザスレッドで使用できるアプリケーションサーバの機能 124
3.4.12 ドメイン名指定でのトップページの表示 125
3.4.13 静的コンテンツのキャッシュ 125
3.4.14 簡易 Web サーバ機能 126
3.5 サーブレットエンジンモードで動作するアプリケーションの実装 127
3.5.1 利用できるアプリケーションサーバの API の種類 127
3.5.2 web.xml 編集時の注意事項 127
3.5.3 そのほかの注意事項 129
3.6 サーブレットエンジンモードでのリソース接続 130
3.6.1 接続できるリソースとトランザクションの管理方法 130
3.6.2 データベース接続の前提条件 130
3.6.3 Web コンテナコネクションプール機能によるデータベース接続 131
3.7 サーブレットエンジンモードの利用方法 135
3.7.1 サーブレットエンジンモードでのシステム設計 135
3.7.2 サーブレットエンジンモードでのシステム構築 138
3.7.3 Web コンテナサーバのセットアップ 143
3.7.4 Web コンテナサーバの動作設定のカスタマイズ 144
3.7.5 トラブルシューティングの資料取得の設定 154
3.7.6 統合ユーザ管理機能の設定 168
3.7.7 Web アプリケーションの設定 170
3.7.8 Web コンテナサーバの作業ディレクトリ 173
3.8 サーブレットエンジンモードでのシステム運用 175
3.8.1 Component Container 管理者による運用に関する留意事項 175
3.8.2 システムの起動 175
3.8.3 システムの停止 177
3.8.4 システムの処理性能の解析 178
3.8.5 トラブルシューティング 179
3.8.6 資料の調査 182
3.8.7 トラブルへの対処と回復 182
3.8.8 トラブルシューティングに関連する留意事項 183
3.8.9 資料の取得方法 183
3.9 サーブレットエンジンモードの場合の移行 195
3.9.1 アプリケーションサーバの移行の概要 195
3.9.2 アプリケーションサーバの移行の手順 197
3.9.3 旧バージョンからの仕様変更の確認 198
3.9.4 移行前の確認 203

3.9.5 定義などのバックアップ 203
3.9.6 手動による定義などの修正 204
3.9.7 Web コンテナサーバの移行コマンドで自動的に追加される定義 206
3.10 Web コンテナサーバで使用するコマンド 209
3.10.1 cjdumpweb（Web コンテナサーバのスレッドダンプの取得） 210
3.10.2 cjstartweb（Web コンテナサーバの開始） 211
3.10.3 cjstopweb（Web コンテナサーバの停止） 212
3.10.4 cjwebchangedbid（データベース ID の変更） 214
3.10.5 cjwebclearpool（プール内のコネクションのクリア） 215
3.10.6 cjwebclosepool（プール内のコネクションの強制切断） 217
3.10.7 cjwebdeletepool（プール管理情報の削除） 218
3.10.8 cjwebeditpool（プール管理情報の登録と変更） 219
3.10.9 cjweblistpool（プール管理情報の一覧表示） 221
3.10.10 cjwebsetup（Web コンテナサーバのセットアップとアンセットアップ） 224
3.11 Web コンテナサーバで使用するファイル 226
3.11.1 hitachi_web.properties（Web コンテナサーバ用 Web アプリケーションプロパティファイル） 227
3.11.2 usrconf.cfg（Web コンテナサーバ用オプション定義ファイル） 229
3.11.3 usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル） 235
3.11.4 web-users.xml（ユーザ認証情報定義ファイル） 256
3.11.5 web.policy（SecurityManager 定義ファイル） 257
3.11.6 プール管理情報設定ファイル 259

4 基本・開発機能の互換機能（DABroker Library を使用したデータベース接続） 263
4.1 この章の構成 264
4.2 DABroker Library を使用したデータベース接続の概要 266
4.2.1 DABroker Library を使用してデータベースに接続する方法 266
4.2.2 接続できるデータベース 267
4.2.3 DABroker Library のメモリ所要量 267
4.3 DABroker Library の初期設定 275
4.3.1 DABroker Library のインストールおよび動作環境の設定（Windows の場合） 275
4.3.2 DABroker Library のインストールおよび動作環境の設定（UNIX の場合） 286
4.3.3 データベースの設定 291
4.4 データベースとアプリケーション間の文字コードの変換 297
4.5 受信バッファのプーリングの設定 298
4.5.1 受信バッファの管理 298
4.5.2 受信バッファプール数の指定 300
4.6 DABroker Library を使用する場合のトラブルシュート 304
4.6.1 snapshot ログとして収集できる資料 304
4.6.2 DABroker Library で取得できるログの種類と出力先 304
4.6.3 DABroker Library のスプール情報の内容 305

4.6.4 資料取得のための設定 329
4.6.5 DABroker Library で取得できるトレース 336
4.6.6 DABroker Library のトレースファイル運用時の留意事項 337
4.6.7 データベースと接続中にトラブルが発生した場合 339
4.7 DABroker Library を使用したデータベース接続の実装 341
4.7.1 DABroker Library で開発するアプリケーション 341
4.7.2 JDBC2.0 の概要 341
4.7.3 JDBC2.0 拡張機能の概要 360
4.7.4 データベース接続の設定 365
4.7.5 JDBC2.0 拡張機能を使用したデータベース接続の設定 378
4.7.6 エラー時の JDBC ドライバの処理 383
4.7.7 データ型の対応 384
4.7.8 データベース接続の実装時の注意事項 386
4.8 DB Connector を使用したデータベースとの接続 388
4.8.1 DB Connector で接続できるデータベース 388
4.8.2 DB Connector（RAR ファイル）の種類 388
4.8.3 必要なモジュール 389
4.8.4 データベースにアクセスする場合の構成 389
4.8.5 DB Connector のインポート 390
4.8.6 DB Connector のプロパティ定義 390
4.8.7 データベースと接続するときの注意事項 395
4.9 DB Connector for Reliable Messaging を使用したデータベース上のキューへの接続 398
4.9.1 DB Connector for Reliable Messaging で接続できるデータベース 398
4.9.2 DB Connector for Reliable Messaging（RAR ファイル）の種類 398
4.9.3 キューに接続する場合に必要なモジュール 399
4.9.4 データベースにアクセスする場合の構成 399
4.9.5 DB Connector for Reliable Messaging のインポートおよびプロパティ定義 399
4.9.6 データベース上のキューと接続するときの注意事項 399
4.10 DABroker Library で使用する API の一覧 400
4.11 Driver クラス 401
4.12 Connection クラス 402
4.13 Statement クラス 405
getExecuteDirectMode メソッド 408
setExecuteDirectMode メソッド 408
4.14 PreparedStatement クラス 410
getBlockUpdate メソッド 414
setBlockUpdate メソッド 414
4.15 CallableStatement クラス 416
4.16 ResultSet クラス 419
4.17 ResultSetMetaData クラス 422

4.18 DatabaseMetaData クラス 424
4.19 DataSource クラス 427
getBlockUpdate メソッド 429
getBufferPoolSize メソッド 430
getBufSize メソッド 430
getDatabaseName メソッド 431
getDBEnv メソッド 431
getDBHostName メソッド 432
getDescription メソッド 432
getEncodLang メソッド 432
getExecuteDirectMode メソッド 433
getHiRDBCursorMode メソッド 433
getJDBC_IF_TRC メソッド 434
getLONGVARBINARY_Access メソッド 435
getNetworkProtocol メソッド 435
getNotErrorOccurred メソッド 436
getOSAuthorize メソッド 436
getPassword メソッド 437
getPortNumber メソッド 437
getRowSize メソッド 438
getServerName メソッド 438
getSQLWarningIgnore メソッド 439
getSV_EVENT_TRC メソッド 439
getTRC_NO メソッド 440
getUapName メソッド 440
getUser メソッド 441
setBlockUpdate メソッド 441
setBufferPoolSize メソッド 442
setBufSize メソッド 443
setDatabaseName メソッド 444
setDBEnv メソッド 444
setDBHostName メソッド 445
setDescription メソッド 445
setEncodLang メソッド 447
setExecuteDirectMode メソッド 447
setHiRDBCursorMode メソッド 448
setJDBC_IF_TRC メソッド 450
setLONGVARBINARY_Access メソッド 451
setNetworkProtocol メソッド 451
setNotErrorOccurred メソッド 452

setOSAuthorize メソッド 453
setPassword メソッド 453
setPortNumber メソッド 454
setRowSize メソッド 454
setServerName メソッド 455
setSQLWarningIgnore メソッド 455
setSV_EVENT_TRC メソッド 456
setTRC_NO メソッド 456
setUapName メソッド 457
setUser メソッド 457
getRMID メソッド 458
getXACloseString メソッド 458
getXALocalCommitMode メソッド 459
getXAOpenString メソッド 459
getXAThreadMode メソッド 460
setRMID メソッド 461
setXACloseString メソッド 461
setXALocalCommitMode メソッド 462
setXAOpenString メソッド 462
setXAThreadMode メソッド 463
4.20 Blob インタフェース 464
4.21 DABroker Library で使用するコマンドの一覧 465
4.22 dabsetup（DABroker Library のセットアップ） 466
4.23 DABroker Library で使用するファイルの一覧 468
4.24 DABroker Library 動作環境定義ファイル 469
4.25 共通設定項目 471
DAB_LANG（LANG 環境変数） 471
DABTMP（作業ディレクトリ） 472
DABSVLOGSIZE（ログファイルサイズ［dabsv.log］） 472
DABSELECTCOLCNT（表検索カラム数） 473
DABSELECTINFCNT（更新・検索条件カラム数） 473
DABORACLE_TYPE（使用する ORACLE のバージョン） 474
DABSQL_HiRDB_DICTIONARY_DATATYPE（HiRDB のディクショナリ表のデータ属性） 474
DAB_HIRDB_DBINF_ESC（HiRDB 環境変数区切り文字） 475
DABEXSQL_TRC_LINE（拡張データベースアクセストレースを取得する） 475
トレースファイルサイズ 476
DABXA_LOG_ERROR（XA トレースを取得する） 476
4.26 拡張データベースアクセストレース設定項目 478
DABSQL_TRC_CANCEL（CANCEL） 478
DABSQL_TRC_CLOSE（CLOSE） 478

DABSQL_TRC_COMMIT（COMMIT） 478
DABSQL_TRC_CONNECT（CONNECT） 479
DABSQL_TRC_DECLARE（DECLARE） 479
DABSQL_TRC_DESCRIBE（DESCRIBE） 479
DABSQL_TRC_DISCONNECT（DISCONNECT） 480
DABSQL_TRC_EXECUTE（EXECUTE） 480
DABSQL_TRC_FETCH（FETCH） 480
DABSQL_TRC_LOGGING（LOGGING） 481
DABSQL_TRC_LOGLESS（LOGLESS） 481
DABSQL_TRC_PREPARE（PREPARE） 481
DABSQL_TRC_ROLLBACK（ROLLBACK） 482
DABSQL_TRC_OPEN（OPEN） 482
DABSQL_TRC_ATTACH（ATTACH） 482
DABSQL_TRC_ATTRGET（ATTRGET） 483
DABSQL_TRC_ATTRSET（ATTRSET） 483
DABSQL_TRC_BIND（BIND） 483
DABSQL_TRC_CHARFORM（CHARFORM） 484
DABSQL_TRC_CHARID（CHARID） 484
DABSQL_TRC_DEFINE（DEFINE） 484
DABSQL_TRC_DESALLOC（DESALLOC） 485
DABSQL_TRC_DESCANY（DESCANY） 485
DABSQL_TRC_DESFREE（DESFREE） 485
DABSQL_TRC_DETACH（DETACH） 486
DABSQL_TRC_ENVCREATE（ENVCREATE） 486
DABSQL_TRC_ERRGET（ERRGET） 486
DABSQL_TRC_GETPIECE（GETPIECE） 487
DABSQL_TRC_HDLALLOC（HDLALLOC） 487
DABSQL_TRC_HDLFREE（HDLFREE） 487
DABSQL_TRC_LOBCLOSE（LOBCLOSE） 488
DABSQL_TRC_LOBLEN（LOBLEN） 488
DABSQL_TRC_LOBOPEN（LOBOPEN） 488
DABSQL_TRC_LOBREAD（LOBREAD） 489
DABSQL_TRC_LOBWRITE（LOBWRITE） 489
DABSQL_TRC_PARAMGET（PARAMGET） 489
DABSQL_TRC_SESBEGIN（SESBEGIN） 490
DABSQL_TRC_SESEND（SESEND） 490
DABSQL_TRC_SETPIECE（SETPIECE） 490
DABSQL_TRC_GETDIAG（GETDIAG） 491
DABSQL_TRC_AUTO_COMMIT_OFF（AUTO COMMIT OFF） 491
DABSQL_TRC_AUTO_COMMIT_ON（AUTO COMMIT ON） 491

DABSQL_TRC_GET_INFO（GET INFO） 492
DABSQL_TRC_SET_INFO（SET INFO） 492
DABSQL_TRC_EXECUTE_DIRECT（EXECUTE DIRECT） 492
4.27 リモートアクセス設定項目 493
DABHIRDBA_BLOBBUFSIZE（HiRDB の BLOB 型データ受取バッファサイズ） 493
DABORADBA_LONGBUFSIZE（ORACLE の LONG 型データ受取バッファサイズ） 493
DABLANGMODE（漢字コードセット） 493
4.28 for C++設定項目 495
DABCPP_EVTTRC_SIZE（トレース取得サイズ） 495
DABCPP_EVTTRC_LEVEL（トレース取得レベル） 495
4.29 HiRDB の設定項目 496
4.30 ORACLE の設定項目 497
4.31 Connector 属性ファイルでの設定 498
4.31.1 DB Connector に設定する<config-property>タグに指定できるプロパティ 498
4.31.2 Connector 属性ファイルのテンプレートファイル 507
4.32 DABroker Library を使用する場合の注意事項 509
4.32.1 DABroker Library を使用する場合に使用できる J2EE コンポーネントおよび機能についての注意事項 509
4.32.2 トランザクションリカバリを使用する場合の注意事項 510
4.32.3 CMP フィールドとデータベースのマッピングの注意事項 510
4.32.4 複数の構築済み実行環境を切り替える場合の注意事項 511
4.32.5 UTF-8 を使用する場合の注意事項（Windows の場合） 512
4.32.6 SQL 実行時の注意事項（Windows の場合） 512
4.32.7 HiRDB クライアント経由で XDM/RD E2 を使用する場合の注意事項 512
4.32.8 Windows Server 2008 または Windows Vista 使用時の注意事項 513

5 基本・開発機能の互換機能（そのほかの機能） 515
5.1 この章の構成 516
5.2 簡易 Web サーバ機能 517
5.2.1 簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能） 517
5.2.2 簡易 Web サーバへのアクセスの制御 518
5.2.3 実行環境での設定 518
5.2.4 サーブレットおよび JSP 実装時の注意事項 519
5.3 EJB クライアントアプリケーションのログのサブディレクトリ専有モード 521
5.3.1 サブディレクトリ専有モードとは 521
5.3.2 ログ出力先のサブディレクトリ数の管理 522
5.3.3 サブディレクトリ専有モードの場合のシステムログの出力先 522
5.3.4 動作モードの指定 524
5.3.5 サブディレクトリ数の管理で使用するコマンド（cjcldellog（システムログ出力先サブディレクトリの削除）） 525
5.3.6 サブディレクトリ専有モード使用時の注意事項 528

5.4 EJB 2.1 と Servlet 2.4 でのアノテーションの利用 531
5.4.1 ロード対象のクラスとロード時に必要なクラスパス 531
5.4.2 EJB 2.1 と Servlet 2.4 でのアノテーション参照抑止機能 533
5.4.3 javax.annotation パッケージに含まれるアノテーションのサポート範囲 533
5.4.4 javax.ejb パッケージに含まれるアノテーションのサポート範囲 534
5.4.5 実行環境での設定（J2EE サーバ単位の設定） 535
5.4.6 アノテーション参照抑止機能の設定変更 536
5.4.7 アノテーションの利用時の注意事項 537

6 拡張機能の互換機能（メモリセッションフェイルオーバ機能） 539
6.1 この章の構成 540
6.2 メモリセッションフェイルオーバ機能の概要 541
6.2.1 処理の概要 541
6.2.2 適用手順 543
6.3 前提条件 545
6.3.1 前提となる構成 545
6.3.2 前提となる設定 545
6.4 グローバルセッションを利用したセッション管理（メモリセッションフェイルオーバ機能） 546
6.4.1 グローバルセッション ID 547
6.4.2 グローバルセッション情報 547
6.4.3 グローバルセッションを利用するためのシステム構成要素 555
6.4.4 グローバルセッション情報が引き継がれるセッションの条件 558
6.4.5 サーバの状態とグローバルセッション情報の引き継ぎ 560
6.4.6 グローバルセッションの制御方法 564
6.4.7 グローバルセッション情報の排他制御 565
6.4.8 グローバルセッション情報の最大数 566
6.5 グローバルセッション情報のサイズの見積もり 568
6.5.1 グローバルセッション情報のサイズ見積もり機能を有効にするための設定（見積もり時に使用する設定） 568
6.5.2 グローバルセッション情報のサイズの見積もり方法 569
6.5.3 グローバルセッション情報のサイズ見積もり機能使用時に無効となる処理 571
6.5.4 グローバルセッション情報のサイズ見積もり機能使用時の注意事項 571
6.6 HTTP Cookie によるグローバルセッションの制御 573
6.6.1 HTTP Cookie による制御の概要 573
6.6.2 HTTP Cookie による制御の前提条件 573
6.6.3 グローバルセッション ID を送信する HTTP Cookie の名称 574
6.6.4 サーブレットコンテキストへの属性追加 574
6.6.5 HTTP Cookie の有効期限 574
6.6.6 グローバルセッション破棄に伴う HTTP Cookie 情報の破棄 574
6.7 URL 書き換えによるグローバルセッションの制御 575

6.7.1 URL 書き換えによる制御の概要 575
6.7.2 URL 書き換えによる制御の前提条件 575
6.7.3 グローバルセッション ID の送信に使用する URL のパスパラメタ 576
6.7.4 URL 書き換えによって制御する場合の負荷分散 577
6.7.5 リクエストラインの最大サイズの設定 579
6.7.6 URL 書き換えによって制御する場合の注意事項 581
6.8 グローバルセッションの制御方法が混在する場合のシステム構成 582
6.8.1 制御方法ごとに Web アプリケーションを分ける場合 582
6.8.2 制御方法ごとに Web アプリケーションを分けない場合 584
6.9 メモリセッションフェイルオーバ機能で実施される処理 588
6.9.1 メモリセッションフェイルオーバ機能で処理が実施されるタイミング 588
6.9.2 J2EE サーバ起動時の処理 588
6.9.3 J2EE アプリケーション開始時の処理 589
6.9.4 リクエスト実行時の処理 592
6.9.5 HttpSession オブジェクトの有効期限が切れた場合の処理 597
6.9.6 無効なグローバルセッション ID が送られた場合の処理 597
6.9.7 URL 書き換えを使用する場合の処理の違い 598
6.10 メモリセッションフェイルオーバ機能使用時の障害検知 600
6.11 アプリケーションの実装 601
6.11.1 実装時の留意事項 601
6.11.2 URL 書き換えによる制御をする場合のアプリケーションの実装 606
6.11.3 アプリケーションを実行するときに追加される属性 608
6.12 DD での定義 609
6.12.1 DD の作成 609
6.12.2 DD に定義する項目 611
6.12.3 DD での定義例 625
6.13 cosminexus.xml での定義 630
6.14 J2EE サーバの設定 631
6.14.1 J2EE サーバの設定内容 631
6.14.2 簡易構築定義ファイルでの定義 632
6.14.3 簡易構築定義ファイルのパラメタの設定要否 634
6.14.4 定義内容の詳細（webserver.sfo から始まるパラメタ） 634
6.14.5 簡易構築定義ファイルの指定例 636
6.15 SFO サーバの設定 638
6.15.1 簡易構築定義ファイルでの定義 638
6.15.2 SFO サーバの起動オプションの設定 640
6.16 Web アプリケーションの設定 642
6.17 Web アプリケーションの設定情報の削除 643

7 拡張機能の互換機能（そのほかの機能） 645
7.1 この章の構成 646
7.2 複数の構築済み実行環境の切り替え 647
7.2.1 構築済み実行環境の切り替えの概要 647
7.2.2 構築済み実行環境の切り替えの流れ 649
7.2.3 構築済み実行環境を切り替えるための準備 651
7.2.4 構築済み実行環境の切り替え手順 656
7.2.5 構築済み実行環境の切り替えで使用するコマンド（cosmienv（構築済み実行環境の退避，回復，削除，一覧出力）） 660
7.2.6 構築済み実行環境の切り替えで出力するログの取得 664
7.2.7 構築済み実行環境の切り替えで出力するログの出力形式と出力項目（UNIX の場合） 665
7.2.8 構築済み実行環境を切り替える場合の注意事項 666
7.3 J2EE アプリケーションのテスト機能 667
7.3.1 テストモードとは 667
7.3.2 テストモードで動作している J2EE アプリケーションに対する処理 668
7.3.3 テストモードで動作している J2EE アプリケーションに対してできる動作 674
7.3.4 テストモードで動作している J2EE アプリケーション情報の取得 678
7.3.5 テストモードでの J2EE アプリケーションの動作確認（新規作成時） 679
7.3.6 テストモードでの J2EE アプリケーションの動作確認（入れ替え時） 681
7.3.7 テストモードによる J2EE アプリケーションの実行手順 684
7.3.8 アプリケーションのモードの切り替えで使用するコマンド（cjchmodapp（アプリケーションのモード切り替え）） 686
7.3.9 テストモードでの J2EE アプリケーション呼び出し時の注意事項 688
7.3.10 テストモード使用時のそのほかの注意事項 690
7.4 JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java） 691
7.4.1 JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックで使用するコマンド（cjjsp2java（JSP ソースのチェック）） 691
7.4.2 JSP 実装時の注意事項 693

付録 695
付録 A 各バージョンでの主な機能変更 696
付録 A.1 09-60 での主な機能変更 696
付録 A.2 09-50 での主な機能変更 697
付録 A.3 09-00 での主な機能変更 701
付録 A.4 08-70 での主な機能変更 704
付録 A.5 08-53 での主な機能変更 706
付録 A.6 08-50 での主な機能変更 708
付録 A.7 08-00 での主な機能変更 711
付録 B 用語解説 715

索引 717

1 アプリケーションサーバの機能

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

1.1　機能の分類

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

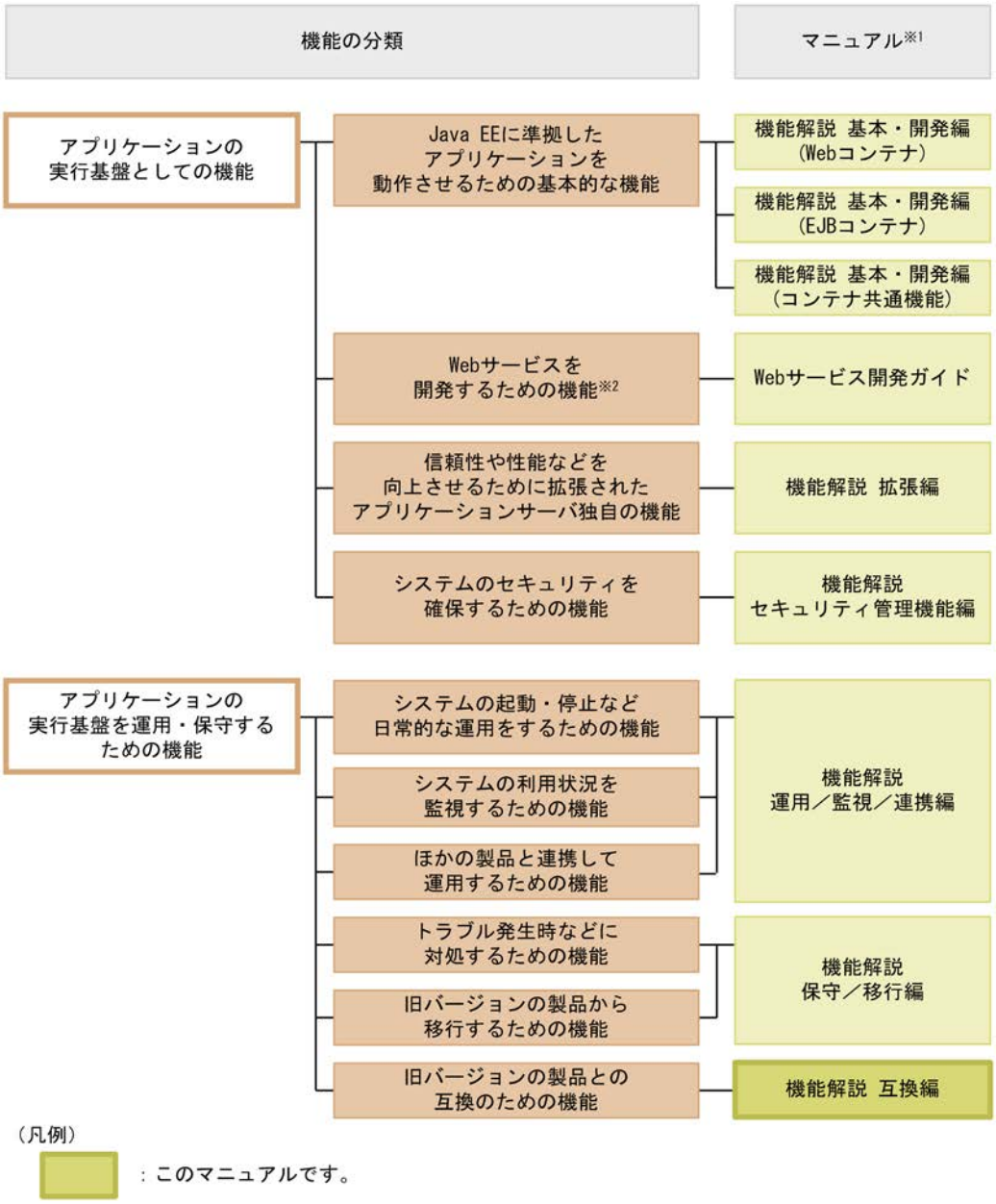
アプリケーションサーバの機能は，大きく分けて，次の二つに分類できます。

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

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

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

図 1-1　アプリケーションサーバの機能の分類と対応するマニュアル



注※1

マニュアル名称の「アプリケーションサーバ」を省略しています。

注※2

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

SOAP アプリケーションを開発・実行する場合

- アプリケーションサーバ SOAP アプリケーション開発の手引

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

- XML Security - Core ユーザーズガイド

- アプリケーションサーバ Web サービスセキュリティ構築ガイド

XML の処理について詳細を知りたい場合

- XML Processor ユーザーズガイド

ここでは，機能の分類について，マニュアルとの対応と併せて説明します。

1.1.1　アプリケーションの実行基盤としての機能

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

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

アプリケーションの実行基盤としての機能について，分類ごとに説明します。

(1)　アプリケーションを動作させるための基本的な機能（基本・開発機能）

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

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

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

それぞれの分類の概要を説明します。

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

(2)　Web サービスを開発するための機能

Web サービスの実行環境および開発環境としての機能が該当します。

アプリケーションサーバでは，次のエンジンを提供しています。

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

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

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

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

(4)　システムのセキュリティを確保するための機能（セキュリティ管理機能）

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

1.1.2　アプリケーションの実行基盤を運用・保守するための機能

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

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

(1)　システムの起動・停止など日常的な運用をするための機能（運用機能）

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

(2)　システムの利用状況を監視するための機能（監視機能）

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

(3)　ほかの製品と連携して運用するための機能（連携機能）

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

(4)　トラブル発生時などに対処するための機能（保守機能）

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

(5)　旧バージョンの製品から移行するための機能（移行機能）

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

(6)　旧バージョンの製品との互換のための機能（互換機能）

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

1.1.3　機能とマニュアルの対応

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

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

表 1-1　機能の分類と機能解説のマニュアルの対応

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

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

分類	機能	マニュアル※1
保守機能	製品の JavaVM（以降，JavaVM と略す場合があります）の機能	保守／移行編
移行機能	旧バージョンのアプリケーションサーバからの移行	
	推奨機能への移行	
互換機能	ベーシックモード	互換編※2
	サーブレットエンジンモード	
	基本・開発機能の互換機能	
	拡張機能の互換機能	

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

注※2　このマニュアルです。

1.2　互換のための機能の一覧

この節では，互換のための機能について，このマニュアルでの参照先を示します。

1.2.1　動作モードに関する機能

動作モードに関する互換機能を次の表に示します。

表 1-2　動作モードに関する互換機能

機能	参照先
ベーシックモード	2 章
サーブレットエンジンモード	3 章

1.2.2　基本・開発機能に関する互換機能

基本・開発機能に関する互換機能を次の表に示します。

表 1-3　基本・開発機能に関する互換機能

機能	参照先
DABroker Library を使用したデータベース接続	4 章
簡易 Web サーバ機能	5.2
EJB クライアントアプリケーションのログのサブディレクトリ専有モード	5.3
EJB 2.1 と Servlet 2.4 でのアノテーション	5.4

1.2.3　拡張機能に関する互換機能

拡張機能に関する互換機能を次の表に示します。

表 1-4　拡張機能に関する互換機能

機能	参照先
メモリセッションフェイルオーバ機能	6 章
複数の構築済み実行環境の切り替え	7.2
J2EE アプリケーションのテスト機能	7.3
JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java）	7.4

1.3　このマニュアルに記載している機能の説明

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

1.3.1　分類の意味

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

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

1.3.2　分類を示す表の例

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

次に，機能説明の分類を示す表の例を示します。

機能説明の分類を示す表の例

表 X-1　この章の構成（○○機能）

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

注※

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

ポイント

cosminexus.xml を含まないアプリケーションのプロパティ設定

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

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

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

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

1.4　アプリケーションサーバ 09-70 での主な機能変更

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

説明内容は次のとおりです。

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

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 1-5　標準機能・既存機能への対応を目的とした変更

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

(2)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 1-6　運用性の維持・向上を目的とした変更

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

(3)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 1-7　そのほかの目的による変更

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

1.5　推奨機能との対応

このマニュアルで説明している機能は，旧バージョンとの互換用機能です。推奨機能への移行をお勧めします。

互換機能と推奨機能の対応を次の表に示します。

表 1-8　互換機能と推奨機能の対応

互換機能	推奨機能
ベーシックモード	1.4 モードを使用することをお勧めします。
サーブレットエンジンモード	
DABroker Library を使用したデータベース接続	それぞれのデータベースが提供している JDBC ドライバを使用することをお勧めします。
簡易 Web サーバ機能	インプロセス HTTP サーバ機能を使用することをお勧めします。 インプロセス HTTP サーバ機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「5.　インプロセス HTTP サーバ」を参照してください。
EJB クライアントアプリケーションのログのサブディレクトリ専有モード	サブディレクトリ共有モードを使用することをお勧めします。 サブディレクトリ共有モードについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「3.8.2　システムログの出力先のサブディレクトリ」を参照してください。
EJB 2.1 と Servlet 2.4 でのアノテーションの利用	EJB 3.0 以降および Servlet 2.5 以降でアノテーションを利用することをお勧めします。
メモリセッションフェイルオーバ機能	データベースセッションフェイルオーバ機能または EADs セッションフェイルオーバ機能を使用することをお勧めします。 データベースセッションフェイルオーバ機能および EADs セッションフェイルオーバ機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「5.　J2EE サーバ間のセッション情報の引き継ぎ」を参照してください。
複数の構築済み実行環境の切り替え	構築済みの環境を以前の状態に戻す場合は，適用したパッチをアンインストールして対応することをお勧めします。
J2EE アプリケーションのテスト機能	J2EE アプリケーションのテストは本番環境ではなくテスト環境を用意して実行することをお勧めします。
JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java）	cjjspc コマンドを使用してコンパイルして，チェックすることをお勧めします。 cjjspc コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の

互換機能	推奨機能
JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java）	「cjjspc（JSP の事前コンパイル）」を参照してください。

2 ベーシックモード

旧バージョンとの互換用機能としてのサーバの動作モードには，ベーシックモードとサーブレットエンジンモードがあります。この章では，ベーシックモードについて説明します。

2.1　この章の構成

この章では，ベーシックモードについて説明します。

この章の構成を次の表に示します。

表 2-1　この章の構成（ベーシックモード）

分類	タイトル	参照先
解説	ベーシックモードの概要	2.2
	使用できる機能の一覧	2.3
	ベーシックモードでのリソース接続	2.4
	プロセスの依存関係とプロセスの再起動方法	2.5
	データソースから DB Connector への移行方法	2.6
	パフォーマンスチューニング	2.7
	データベースとアプリケーション間の文字コード変換	2.8
	下位互換用のクラスローダ構成	2.9
設定	ベーシックモードの設定	2.10
	ベーシックモードで使用するファイル	2.11
運用	ベーシックモードでデータベースに接続する場合の運用管理（互換用機能）	2.12
	J2EE サーバで使用するリソース操作コマンド	2.13
注意事項	ベーシックモード使用時の注意事項	2.14

注　「実装」について，この章での説明はありません。

2.2　ベーシックモードの概要

ベーシックモードは，J2EE サーバモードの一つです。単一のデータベースだけのリソースをトランザクションで使用するシステムに適用できます。旧バージョンとの互換性を重視したモードです。

なお，ベーシックモードは，1.4 モード（06-71 までのスタンダードモードまたはアドバンスドモード）に比べて，使用できる機能に制限があります。J2EE サーバモードを使用する場合は，1.4 モードを使用することを推奨します。

参考

ベーシックモードから 1.4 モードへの変更

すでにベーシックモードを使用している場合も，1.4 モードへの変更ができます。変更する場合は，次の点に留意してください。

- サーバの環境をいったん削除してから，再度サーバの環境をセットアップし直す必要があります。
- 1.4 モードを使用する場合は，トランザクション管理サーバなどが必要となるため，ベーシックモードに比べ，多量の CPU/メモリリソースが必要となります。

また，動作モードの変更によって，変更前の動作モードで使用できたプロパティが使用できなくなることがありますので注意してください。

2.3　使用できる機能の一覧

ベーシックモードで使用できる Java EE および J2EE の機能，ならびにアプリケーションサーバの機能について一覧に示します。

アプリケーションサーバで使用できる機能については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の「3.2.5(1)　アプリケーションサーバで使用できる機能」を参照してください。

2.3.1　使用できる Java EE および J2EE の機能

ベーシックモードで使用できる Java EE および J2EE の機能について，次の表に示します。なお，次の表に記載していない機能は使用できません。

表 2-2　互換用の動作モードで使用できる Java EE および J2EE の機能

Java EE および J2EE の機能	ベーシックモード
Servlet 2.2	使用できます。
Servlet 2.3	
JSP 1.2	
EJB 2.0（CMP 1.1）	
JDBC 2.0 コア JDBC 2.0 オプションパッケージ	
JTA 1.0.1（local）	
JavaMail 1.2	JavaMail で使用できるサービスプロバイダは SMTP だけです。メールの送信機能だけを使用できます（メールの受信機能は使用できません）。

2.3.2　使用できるアプリケーションサーバの機能

ベーシックモードでの，アプリケーションサーバの各機能の使用について説明します。

なお，各項目にある表に記載していない機能は使用できません。

(1)　Web コンテナの機能

Web コンテナの機能のベーシックモードでの使用について次の表に示します。Web コンテナの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.　Web コンテナ」を参照してください。

表 2-3　Web コンテナの機能のベーシックモードでの使用

Web コンテナの機能	ベーシックモードでの使用
Web アプリケーションの実行機能	◎
JSP の事前コンパイルとコンパイル結果の保持（JSP 事前コンパイルを使用しない場合の JSP コンパイル結果）	◎
デフォルトの文字エンコーディング設定機能	◎

Web コンテナの機能		ベーシックモードでの使用
セッション管理機能	Web クライアントが保持する無効なセッション ID の削除	◎
	HttpSession オブジェクト数の上限値の設定	◎
	セッション ID および Cookie へのサーバ ID の付加	◎
アプリケーションのイベントリスナ		○
リクエストおよびレスポンスのフィルタリング		○
HTTP レスポンス圧縮機能		○
EJB コンテナとの連携		◎
データベースとの接続		△
Web コンテナによるスレッドの作成		◎
ユーザスレッドの使用		△※
同時実行スレッド数の制御	Web コンテナ単位での同時実行スレッド数の制御	◎
	Web アプリケーション単位での同時実行スレッド数の制御	◎
	同時実行スレッド数の動的変更	◎
エラーページのカスタマイズ		◎
静的コンテンツのキャッシュ		◎
簡易 Web サーバ機能	簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能）	◎
	簡易 Web サーバへのアクセスの制御	◎
URI のデコード機能		◎
Web アプリケーションのバージョン設定機能		◎

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※

ユーザスレッドで使用できる機能のうち，ベーシックモードで使用できる機能だけが使用できます。

(2)　Web サーバ連携の機能

Web サーバ連携の機能のベーシックモードでの使用について次の表に示します。Web サーバ連携の機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.　Web サーバ連携」を参照してください。

表 2-4　Web サーバ連携の機能のベーシックモードでの使用

Web サーバ連携の機能	ベーシックモードでの使用
Web サーバ（リダイレクタ）によるリクエストの振り分け	◎
通信タイムアウト（Web サーバ連携）	◎
IP アドレスの指定（Web サーバ連携）	◎
エラーページのカスタマイズ（Web サーバ連携）	◎
ドメイン名指定でのトップページの表示	○
Web コンテナへのゲートウェイ情報の通知	○

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(3)　インプロセス HTTP サーバの機能

インプロセス HTTP サーバの機能のベーシックモードでの使用について次の表に示します。インプロセス HTTP サーバの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「5.　インプロセス HTTP サーバ」を参照してください。

表 2-5　インプロセス HTTP サーバの機能のベーシックモードでの使用

インプロセス HTTP サーバの機能	ベーシックモードでの使用
Web クライアントからの接続数の制御	◎
リクエスト処理スレッド数の制御	◎
Web クライアントからの同時接続数の制御によるリクエストの流量制御	◎
同時実行スレッド数の制御によるリクエストの流量制御	◎
リダイレクトによるリクエストの振り分け	◎
Persistent Connection による Web クライアントとの通信制御	◎
通信タイムアウト（インプロセス HTTP サーバ）	◎
IP アドレス指定（インプロセス HTTP サーバ）	◎
アクセスを許可するホストの制限によるアクセス制御	◎
リクエストデータのサイズの制限によるアクセス制御	◎
有効な HTTP メソッドの制限によるアクセス制御	◎
HTTP レスポンスを使用した Web クライアントへのレスポンスのカスタマイズ	◎
エラーページのカスタマイズ（インプロセス HTTP サーバ）	◎
Web コンテナへのゲートウェイ情報の通知	◎
ログ・トレースの出力	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(4)　EJB コンテナの機能

EJB コンテナの機能のベーシックモードでの使用について次の表に示します。EJB コンテナの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「2.　EJB コンテナ」を参照してください。

表 2–6　EJB コンテナの機能のベーシックモードでの使用

EJB コンテナの機能		ベーシックモードでの使用
Enterprise Bean の実行		○
EJB 仕様準拠のチェック		○
CMP フィールドとデータ型のマッピング		○
EJB コンテナの JNDI 名前空間へのリファレンス登録		○
外部リソースとの接続		○
Enterprise Bean でのトランザクション管理		○※
Entity Bean のキャッシュモデル（コミットオプション指定）		○
Stateless Session Bean，Entity Bean のプールの管理		◎
Enterprise Bean へのアクセス制御		○
EJB コンテナでのタイムアウトの設定	Stateful Session Bean のタイムアウト	◎
	Entity Bean の EJB オブジェクトのタイムアウト	◎
	RMI-IIOP 通信のタイムアウト	◎
EJB のリモートインタフェースの呼び出し		◎
EJB コンテナの通信ポートと IP アドレスの固定（TPBroker のオプション）		◎

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

注※

SessionSynchronization を実装した Stateful Session Bean へのトランザクション属性には，Required，RequiresNew および Mandatory だけが指定できます。

(5)　EJB クライアントの機能

EJB クライアントの機能のベーシックモードでの使用について次の表に示します。EJB クライアントの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「3.　EJB クライアント」を参照してください。

表 2-7　EJB クライアントの機能のベーシックモードでの使用

EJB クライアントの機能	ベーシックモードでの使用
EJB クライアントアプリケーションの開始	◎
Enterprise Bean の呼び出し	◎
EJB クライアントアプリケーションでのトランザクションの実装	◎
EJB クライアントアプリケーションでのセキュリティの実装	◎
RMI-IIOP スタブ，インタフェースの取得	◎
EJB クライアントアプリケーションのシステムログ出力	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(6)　ネーミング管理の機能

ネーミング管理の機能のベーシックモードでの使用について次の表に示します。ネーミング管理の機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.　ネーミング管理」を参照してください。

表 2-8　ネーミング管理の機能のベーシックモードでの使用

ネーミング管理の機能	ベーシックモードでの使用
JNDI 名前空間へのオブジェクトのバインドとルックアップ	◎
Enterprise Bean または J2EE リソースへの別名付与（ユーザ指定名前空間機能）	◎※
ラウンドロビンポリシーによる CORBA ネーミングサービスの検索	◎
ネーミング管理機能でのキャッシング	◎
CORBA ネーミングサービスの切り替え	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

注※

リモートホームインタフェースに対する別名だけを使用できます。

ローカルホームインタフェース，リモートビジネスインタフェースまたはローカルビジネスインタフェースに対しても別名は設定できます。ただし，ベーシックモードではこれらのインタフェースがサポートされていないため，設定した別名は使用できません。

(7)　リソース接続とトランザクション管理の機能

リソース接続とトランザクション管理の機能のベーシックモードでの使用について次の表に示します。なお，接続できるリソースやリソース接続で使用できる機能については，「2.4　ベーシックモードでのリソース接続」を参照してください。リソース接続とトランザクション管理の機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.　リソース接続とトランザクション管理」を参照してください。

表 2-9　リソース接続とトランザクション管理の機能のベーシックモードでの使用

リソース接続とトランザクション管理の機能	ベーシックモードでの使用
コネクションプーリング	◎※1
コネクションシェアリング・アソシエーション（コネクションシェアリング）	○
DataSource オブジェクトのキャッシング	◎
コネクションの取得リトライ	◎※1
コネクションプールの情報表示	◎
コネクションプールのクリア	◎
コネクションの自動クローズ（EJB コンテナによるコネクション自動クローズ）	○
コネクションの強制クローズ	○※2
コネクションスイーパ	◎
トランザクションタイムアウトとステートメントキャンセル	○※3
オブジェクトの自動クローズ	○
リソースへの接続テスト	◎※4

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

注※1　データソースを利用する場合に使用できます。詳細については，「2.4.3　データソースによるデータベースへの接続で使用できる機能」を参照してください。

注※2　コネクションの強制クローズについては，「2.4.3　データソースによるデータベースへの接続で使用できる機能」を参照してください。

注※3　トランザクションタイムアウトは使用できます。詳細については，「2.4.3　データソースによるデータベースへの接続で使用できる機能」を参照してください。

注※4　データソースおよび SMTP サーバを利用する場合に使用できます。詳細については，「2.4.4　リソースへの接続テスト」を参照してください。

(8)　OpenTP1 からのアプリケーションサーバを呼び出す機能（TP1 インバウンド連携機能）

TP1 インバウンド連携機能は，ベーシックモードでは使用できません。

(9)　アプリケーションサーバで使用する JPA の機能

アプリケーションサーバで使用する JPA の機能は，ベーシックモードでは使用できません。

(10)　CJPA プロバイダの機能

CJPA プロバイダの機能は，ベーシックモードでは使用できません。

(11)　CJMS プロバイダの機能

CJMS プロバイダの機能は，ベーシックモードでは使用できません。

(12)　セキュリティ管理の機能

セキュリティ管理の機能のベーシックモードでの使用について次の表に示します。セキュリティ管理の機能については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を参照してください。

表 2-10　セキュリティ管理の機能のベーシックモードでの使用

セキュリティ管理の機能	ベーシックモードでの使用
SSL 使用による認証情報とデータの暗号化	◎
ロールに基づく Web コンテナのユーザ認証	◎
SecurityManager による Web コンテナの実行時の保護	◎
セキュリティアイデンティティを使用した認証	△※

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※　サーブレット／JSP から EJB を呼び出す際に，web.xml に指定した<run-as>タグの情報は使用されません。

(13)　アノテーションの機能

アノテーションの機能は，ベーシックモードでは使用できません。

(14)　J2EE アプリケーションの形式とデプロイの機能

J2EE アプリケーションの形式とデプロイの機能のベーシックモードでの使用について次の表に示します。J2EE アプリケーションの形式とデプロイの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.　J2EE アプリケーションの形式とデプロイ」を参照してください。

表 2-11　J2EE アプリケーションの形式とデプロイの機能のベーシックモードでの使用

J2EE アプリケーションの形式とデプロイの機能	ベーシックモードでの使用
アーカイブ形式による J2EE アプリケーションのデプロイ	○
J2EE アプリケーションの入れ替え	◎△※1
J2EE アプリケーションのリデプロイ	◎
リソースアダプタを含む J2EE アプリケーションの操作	○※2

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※1　機能差については，「(21)　J2EE アプリケーションの運用機能」を参照してください。

注※2　リソースアダプタを含む J2EE アプリケーションのインポートはできますが，J2EE アプリケーションの開始はできません。また，開始状態の J2EE アプリケーションをインポートした場合，J2EE アプリケーションにリソースアダプタが含まれている場合は開始できません。

(15)　コンテナ拡張ライブラリの機能

! 注意事項

コンテナ拡張ライブラリには次のアクセス権が付与されます。アクセス権は変更できません。
java.security.AllPermission
ただし，java.lang.RuntimePermission の setSecurityManager アクセス権は付与されません。

コンテナ拡張ライブラリの機能のベーシックモードでの使用について次の表に示します。コンテナ拡張ライブラリの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.　コンテナ拡張ライブラリ」を参照してください。

表 2-12　コンテナ拡張ライブラリの機能のベーシックモードでの使用

コンテナ拡張ライブラリの機能	ベーシックモードでの使用
コンテナ拡張ライブラリ	◎
サーバ起動・停止フック機能	◎
スマートエージェント経由での CORBA オブジェクトの呼び出し	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(16)　バッチアプリケーション実行時に使用する機能

バッチアプリケーション実行時に使用する機能は，ベーシックモードでは使用できません。

(17)　CTM による Enterprise Bean のスケジューリング機能

CTM による Enterprise Bean のスケジューリング機能のベーシックモードでの使用について次の表に示します。CTM による Enterprise Bean のスケジューリング機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「3.　CTM によるリクエストのスケジューリングと負荷分散」を参照してください。

表 2-13　CTM による Enterprise Bean のスケジューリング機能のベーシックモードでの使用

CTM による Enterprise Bean のスケジューリング機能	ベーシックモードでの使用
リクエストの流量制御	◎
リクエストの優先制御	◎
リクエストの同時実行数の動的変更	◎
リクエストの閉塞制御	◎
リクエストの負荷分散	◎
リクエストのキューの滞留監視	◎
CTM のゲートウェイ機能を利用した TPBroker／OTM クライアントとの接続	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(18)　そのほかの拡張機能

そのほかの拡張機能のベーシックモードでの使用について次の表に示します。そのほかの拡張機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「1.2.3　そのほかの拡張機能」を参照してください。

表 2-14　そのほかの拡張機能のベーシックモードでの使用

そのほかの拡張機能	ベーシックモードでの使用
セッションフェイルオーバ機能（メモリセッションフェイルオーバ）	◎
明示管理ヒープ機能を使用した FullGC の抑止	◎
統合ユーザ管理	◎
ユーザログ出力	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(19)　システムの日常運用を支援する機能

システムの日常運用を支援する機能のベーシックモードでの使用について次の表に示します。システムの日常運用を支援する機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.1　システムの日常運用を支援する機能」を参照してください。

表 2-15　システムの日常運用を支援する機能のベーシックモードでの使用

システムの日常運用を支援する機能	ベーシックモードでの使用
システムの起動・停止	◎△※
運用管理コマンドによる稼働情報の出力	◎
CTM の稼働統計情報の収集	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※　プロセスの依存関係とプロセスの再起動方法が J2EE サーバモードの場合と異なります。詳細については，「2.5 プロセスの依存関係とプロセスの再起動方法」を参照してください。

(20)　システムの保守を支援する機能

システムの保守を支援する機能のベーシックモードでの使用について次の表に示します。システムの保守を支援する機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.2　システムの保守を支援する機能」を参照してください。

表 2-16　システムの保守を支援する機能のベーシックモードでの使用

システムの保守を支援する機能	ベーシックモードでの使用
コンソールログの出力	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(21)　J2EE アプリケーションの運用機能

J2EE アプリケーションの運用機能のベーシックモードでの使用について次の表に示します。J2EE アプリケーションの運用機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.3　J2EE アプリケーションの運用機能」を参照してください。

表 2-17　J2EE アプリケーションの運用機能のベーシックモードでの使用

J2EE アプリケーションの運用機能		ベーシックモードでの使用
J2EE アプリケーションのサービスの閉塞		◎
J2EE アプリケーションの停止（通常停止）		○
J2EE アプリケーションの入れ替え	Web アプリケーションのサービスの部分閉塞による入れ替え	○
	リデプロイによる J2EE アプリケーションの入れ替え	◎

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(22)　システムの監査を支援する機能

システムの監査を支援する機能は，ベーシックモードでは使用できません。

(23)　JP1 連携による運用管理機能

JP1 連携による運用管理機能のベーシックモードでの使用について次の表に示します。JP1 連携による運用管理機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.5 JP1 連携による運用管理機能」を参照してください。

表 2-18　JP1 連携による運用管理機能のベーシックモードでの使用

JP1 連携による運用管理機能	ベーシックモードでの使用
JP1 と連携したシステムの運用	○
システムの集中監視（JP1/IM との連携）	○
ジョブによるシステムの自動運転（JP1/AJS との連携）	○
監査ログの収集および一元管理（JP1/Audit Management - Manager との連携）	○

（凡例）

○：使用できる

参考

ベーシックモードで動作している J2EE サーバは Management Server の管理対象となります。このため，ベーシックモードで動作している J2EE サーバは JP1 連携による管理の対象となります。なお，JP1 連携による運用管理機能と動作モードは，直接関係ありません。

(24)　クラスタソフトウェアとの連携による系切り替え機能

クラスタソフトウェアとの連携による系切り替え機能のベーシックモードでの使用について次の表に示します。クラスタソフトウェアとの連携による系切り替え機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.6　クラスタソフトウェアとの連携による系切り替え機能」を参照してください。

表 2-19　クラスタソフトウェアとの連携による系切り替え機能のベーシックモードでの使用

クラスタソフトウェアとの連携による系切り替え機能		ベーシックモードでの使用
1:1 系切り替えシステム	アプリケーションサーバの 1:1 系切り替えシステム	○
	運用管理サーバの 1:1 系切り替えシステム	○
相互系切り替えシステム		○

（凡例）

○：使用できる

(25)　システムの保守のための機能

システムの保守のための機能のベーシックモードでの使用について次の表に示します。

表 2-20　システムの保守のための機能のベーシックモードでの使用

システムの保守のための機能	ベーシックモードでの使用	参照先マニュアル	参照先
トラブルシューティング	◎※1	アプリケーションサーバ 機能解説 保守／移行編	2 章，3 章，4 章，5 章
性能解析トレースを使用したシステムの性能解析	◎※2		7 章，8 章

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

注※1　障害検知時コマンドは使用できません。

注※2　機能レイヤのうち，JTA，JCA コンテナおよび DB Connector では性能解析情報を出力しません。

(26)　JavaVM の機能

JavaVM の機能のベーシックモードでの使用について次の表に示します。

表 2-21　JavaVM の機能のベーシックモードでの使用

JavaVM の機能	ベーシックモードでの使用	参照先マニュアル	参照先
クラス別統計機能	◎	アプリケーションサーバ 機能解説 保守／移行編	9.3
インスタンス統計機能	◎		9.4
STATIC メンバ統計機能	◎		9.5
参照関係情報出力機能	◎		9.6

JavaVM の機能	ベーシックモードでの使用	参照先マニュアル	参照先
統計前の GC コレクション選択機能	◎	アプリケーションサーバ 機能解説 保守／移行編	9.7
Survivor 領域の年齢分布情報出力機能	◎		9.11
JIT コンパイル時の C ヒープ確保量の上限値設定機能	◎		9.13
スレッド数の上限値設定機能	◎		9.14

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

2.4　ベーシックモードでのリソース接続

ここでは，ベーシックモードでのリソース接続について説明します。

2.4.1　接続できるリソースとトランザクションの管理方法

ベーシックモードの場合に接続できるリソースの種類とトランザクションの管理方法について説明します。

(1)　接続できるリソース

ベーシックモードで接続できるリソースについて次の表に示します。

表 2-22　接続できるリソース（ベーシックモード）

リソースの種類		ベーシックモード
データベース	HiRDB	○
	Oracle	○
	SQL Server	×
	XDM/RD E2	×
データベース上のキュー	HiRDB	×
	Oracle	×
OpenTP1		×
SMTP サーバ		○
JavaBeans リソース		○

（凡例）　○：接続できる　　×：接続できない

参考

リソースアダプタが Non-Managed 環境での使用に対応している場合には，接続できる場合があります。ただし，アプリケーションサーバによるコネクションプーリングやトランザクション管理は利用できません。

(2)　トランザクションの管理方法

ベーシックモードでは，次のトランザクションを管理できます。

- アプリケーションサーバが管理するトランザクションのローカルトランザクション
- アプリケーションサーバが管理しないトランザクション

2.4.2　データベース接続の前提条件

ベーシックモードでデータベースに接続する場合の前提条件を，データベースの種類ごとに示します。

ベーシックモードの場合，データベース接続には，データソースを使用します。使用できるトランザクションの種類は，ローカルトランザクションだけになります。

なお，ベーシックモードでは，次のデータベースには接続できません。

- Oracle10g 以降
- SQL Server
- XDM/RD E2

(1)　HiRDB の場合

- **使用できる JDBC ドライバ**

  使用できる JDBC ドライバは DABroker Library となります。

  DABroker Library を使用する場合，HiRDB クライアントをインストールする必要があります。

- **接続方法**

  データソースを使用して接続します。

  なお，トランザクションの管理方法は，アプリケーションサーバが管理するトランザクション，またはアプリケーションサーバが管理しないトランザクションのどちらかを選択できます。

(2)　Oracle9i の場合

- **使用できる JDBC ドライバ**

  使用できる JDBC ドライバは DABroker Library となります。Oracle JDBC Thin Driver では接続できません。

  JDBC ドライバとして DABroker Library を使用する場合，Oracle クライアントをインストールする必要があります。

- **接続方法**

  データソースを使用して接続します。

  なお，トランザクションの管理方法は，アプリケーションサーバが管理するトランザクション，またはアプリケーションサーバが管理しないトランザクションのどちらかを選択できます。

(3)　DABroker Library の文字コード設定

DABroker Library を使用してデータベースに接続する場合，文字コードを設定できます。ベーシックモードの場合，usrconf.properties の ejbserver.jdbc.propertyInfo.ENCODELANG キーに文字コードを設定します。

usrconf.properties の詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.4　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。

2.4.3　データソースによるデータベースへの接続で使用できる機能

ここでは，データソースによるデータベースへの接続機能について説明します。

なお，データソースの設定については，「2.10　ベーシックモードの設定」を参照してください。

データソースを利用すると，J2EE サーバと連携して，J2EE アプリケーションから次に示す機能を利用できます。

(1)　サインオン

データソースを利用した場合，サインオンの方式として，コンテナ管理のサインオンとコンポーネント管理のサインオンの両方を使用できます。ただし，コンポーネント管理でのサインオンの場合，トランザクションの種類によって使用が制限されます。

コンテナ管理でのサインオンの場合

コンテナ管理でのサインオンの場合には，データソースで設定したユーザ名とパスワードを使ってデータベースにアクセスします。

コンポーネント管理でのサインオンの場合

データベースへのアクセスは getConnection メソッドで渡されたユーザ名とパスワードを使ってデータベースにアクセスします。

(2)　コネクション管理

データベースの接続情報は，データソースを介して J2EE サーバで管理され，コネクションプールにプーリングされます。

(3)　トランザクション管理

ローカルトランザクションでトランザクション管理されます。

トランザクションについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.4　トランザクション管理」を参照してください。

(4)　オブジェクトの自動クローズ

データソースは，ユーザハンドル（java.sql.Connection）から生成された，Statement オブジェクト，PreparedStatement オブジェクト，CallableStatement オブジェクト，および各種ステートメントや DatabaseMetaData から生成された ResultSet オブジェクトを自動クローズできます。

(5)　パフォーマンスチューニングのための機能

ベーシックモードでデータベースに接続する場合に，パフォーマンスチューニングのために使用できる機能を，次の表に示します。

表 2-23　ベーシックモードでデータベースに接続する場合に使用できるパフォーマンスチューニングのための機能

機能	使用可否	機能差※1
コネクションプーリング	○	あり
コネクションシェアリング	○※2	なし
コネクションのアソシエーション	×	－
ステートメントプーリング	×	－
ライトトランザクション	×	－
インプロセストランザクションサービス	×	－
DataSource オブジェクトのキャッシング	○	なし
DB Connector のコンテナ管理でのサインオンの最適化	×	－
受信バッファのプーリング	×	－

（凡例）○：使用できる　×：使用できない　－：該当しない

注※1　1.4 モードとの機能差の有無です。

注※2　ベーシックモードでコネクションシェアリング機能を利用するには，コネクションプーリング機能を利用する必要があります。なお，標準 DD の<res-sharing-scope>に Unshareable を指定してもコネクションシェアリングは有効です。

以降，ベーシックモードで使用できるパフォーマンスチューニングのための機能について説明します。なお，それぞれの機能の概要および目的については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14　パフォーマンスチューニングのための機能」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.1　コネクションプーリング」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.2　コネクションプーリングで使用できる機能」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.3　コネクションシェアリング・アソシエーション」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.4　ステートメントプーリング」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.5　ライトトランザクション」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.6　インプロセストランザクションサービス」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.7　DataSource オブジェクトのキャッシング」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.8　DB Connector のコンテナ管理でのサインオンの最適化」
- このマニュアルの「4.5　受信バッファのプーリングの設定」

ここでは，ベーシックモードで特有の内容についてだけ説明します。

コネクションプーリング

J2EE サーバでは, JDBC データソースごとにプールを作成して管理します。JDBC データソースに最大値，最小値，増加，しきい値などを指定することで，プーリングの動作をカスタマイズできます。

J2EE コンポーネントが JDBC データソースから JDBC コネクションを取得したとき，最小値分の JDBC コネクションが生成されプーリングされます。また，JDBC コネクションは最大値分までプーリングされます。プール中の未使用 JDBC コネクションの数がしきい値分以下になったとき，新たに増加分の JDBC コネクションを取得してプールに補充します。プーリングされる JDBC コネクションの数は，J2EE コンポーネントからのデータベースアクセス量に応じて，最大値と最小値の間になります。

なお，JDBC コネクションプーリングは，同一 JDBC データソースに対して，引数なしの getConnection()メソッドと引数ありの getConnection(java.lang.String username,java.lang.String password)メソッド間では，別のプールになります。

また，引数ありの getConnection(java.lang.String username,java.lang.String password)メソッド間で username が異なる場合, 別のプールになります。なお, J2EE サーバ間の JDBC コネクションプーリングは，サポートしていません。

コネクションシェアリング

1.4 モードとの機能差はありません。

DataSource オブジェクトのキャッシング

1.4 モードとの機能差はありません。

(6)　フォールトトレランスのための機能

ベーシックモードでデータベースに接続する場合に，フォールトトレランスのために使用できる機能を，次の表に示します。

表 2-24　ベーシックモードでデータベースに接続する場合に使用できるフォールトトレランスのための機能

機能	使用可否	機能差※
コネクションの障害検知	×	－
コネクション枯渇時のコネクション取得待ち	×	－
コネクションの取得リトライ	○	なし
コネクションプールの情報表示	○	あり
コネクションプールのクリア	○	なし
コネクションの自動クローズ	○	あり
コネクションの強制クローズ	○	あり
コネクションスイーパ	○	あり
トランザクションタイムアウト	○	あり
トランザクションリカバリ	×	－
障害調査用 SQL の出力	×	－
クラスタコネクションプール	×	－

（凡例）○：使用できる　×：使用できない　－：該当しない

注※　1.4 モードとの機能差です。

以降，ベーシックモードで使用できるフォールトトレランスのための機能について説明します。

なお，それぞれの機能の概要および目的については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15　フォールトトレランスのための機能」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.1　コネクションの障害検知」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.2　コネクション枯渇時のコネクション取得待ち」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.3　コネクションの取得リトライ」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.4　コネクションプールの情報表示」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.5　コネクションプールのクリア」

- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.6　コネクションの自動クローズ」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.7　コネクションスイーパ」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.8　トランザクションタイムアウトとステートメントキャンセル」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.9　トランザクションリカバリ」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.10　障害調査用 SQL の出力」
- マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.17　クラスタコネクションプール機能」

ここでは，ベーシックモードで特有の内容についてだけ説明します。

コネクションの取得リトライ

1.4 モードとの機能差はありません。マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.3　コネクションの取得リトライ」を参照してください。

コネクションプールの情報表示

コネクションプール内のコネクションの情報を，コマンドを使用して表示できます。コネクションプール情報の表示には，cjclosecn コマンドを使用します。コマンドの詳細については，「cjclosecn（J2EE サーバのコネクションクローズまたはコネクションの状態表示)」を参照してください。

コネクションプールのクリア

1.4 モードとの機能差はありません。マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.5　コネクションプールのクリア」を参照してください。

コネクションの強制クローズ

アプリケーションの Connection.close メソッドの記述もれによってクローズされないコネクションは，コネクションプール内で使用中状態のまま再利用されません。これらのコネクションは，cjclosecn コマンドを使用して強制クローズできます。コマンドの詳細については「cjclosecn（J2EE サーバのコネクションクローズまたはコネクションの状態表示)」を参照してください。

なお，ユーザスレッドでは使用できません。

コネクションの自動クローズ

EJB コンテナによるコネクション自動クローズだけが有効になります。また，対象となるリソースは，データソースだけです。

コネクションスイーパ

一定間隔でコネクションプールを監視し，一定時間使用していない未使用コネクションを破棄します。ベーシックモードでのコネクションスイーパの動作については，「2.11.5　コネクションプーリングとコネクションスイーパの動作」を参照してください。

トランザクションタイムアウト

- ローカル呼び出し最適化機能を使用するかどうかによって，トランザクションタイムアウト後の JTA および JDBC インタフェースにアクセスできない期間が異なります。

  ローカル呼び出し最適化機能を使用しない場合，該当するトランザクションを含むリモートインタフェースのメソッドが終了すればアクセスできます。ローカル呼び出し最適化機能を使用する場合は，リクエスト終了するまでアクセスできません。

- トランザクションタイムアウトが発生すると，トランザクションの状態が javax.transaction.Status.STATUS_NO_TRANSACTION になるため，Required 属性，Supports 属性，または Never 属性のビジネスメソッドが呼び出されます。トランザクションタイムアウト後にビジネスメソッドが呼び出されないようにするには，Mandatory 属性を使用してください。ただし，ローカル呼び出し最適化機能を使用する場合は，トランザクションタイムアウト後に新しいトランザクションを開始できないので，タイムアウト後に Required 属性または RequiresNew 属性のビジネスメソッドは呼び出されません。
- ローカル呼び出し最適化機能を使用して，タイムアウトの発生後に Stateful Session Bean（BMT）を呼び出した場合，またはリモートインタフェースのメソッド呼び出し中（トランザクションがサスペンド中）にタイムアウトが発生した場合，J2EE アプリケーションにタイムアウトの発生を通知しません。

  J2EE アプリケーションでこの内容を考慮していない場合，意図しない DBMS のアクセスによる，データの不整合などの問題が発生するおそれがあります。この問題を回避するため，ローカル呼び出し最適化機能を使用しないか，または次の対策を行ってください。

  JTA，または JDBC インタフェースのアクセスで例外が発生した場合，およびリモートインタフェースのメソッド呼び出しで RemoteException 例外が発生した場合は，リクエストを終了させます。また，Stateful Session Bean（BMT）を使用する場合は，そのメソッド内で，JTA，JDBC インタフェースに対して，必ずアクセスします。

(7)　データソースを使用するときの注意

- java.sql.Connection を，複数のトランザクション間で再利用することはできません。java.sql.Connection をトランザクション内で利用する場合には，トランザクションごとに，javax.sql.DataSource から getConnection()メソッドを使用してコネクションを取得してください。
- java.sql.Connection のような JDBC インタフェースの API で SQLException 例外が発生すると，コネクションクローズで該当コネクションの接続が維持されていても，コネクションプールに戻さないで破棄します。その結果，余分なコネクション生成が発生して性能に影響を与えることがあります。

2.4.4　リソースへの接続テスト

データソースのプロパティ設定後，設定した内容が正しいかどうか，検証することができます。ここでは，接続テストでの検証項目と，接続テストの前提条件について説明します。

なお，接続テストの実行手順や，データソースの設定については，「2.10　ベーシックモードの設定」を参照してください。

接続テストで検証される内容は，接続するリソースごとに異なります。

ベーシックモードの場合に，リソースごとの接続テストで検証できる内容を，次の表に示します。

表 2-25　接続テストでの検証内容

リソースの種類	接続方法	接続テストでの検証内容
データベース	データソース	• Driver を使用していること • DABroker Library の設定が正しいこと • データベースとの接続を確立できること
SMTP サーバ	メールコンフィグレーション	• SMTP サーバへの接続を確立できること

2.4.5　JTA によるトランザクション実装時の注意事項

JTA を使用してトランザクションを実装したプログラムは，J2EE サーバの動作モードによって異なる動作をする場合があります。ベーシックモードの場合のプログラムの処理内容，および動作を次に示します。1.4 モードの場合については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.4.11　JTA によるトランザクション実装時の注意事項」を参照してください。

表 2-26　ベーシックモードの場合の JTA の動作

処理の内容	動作
• EJB 呼び出しで，java.rmi.RemoteException または java.rmi.RemoteException を継承した例外が発生した場合 • CORBA のシステム例外が発生した場合	クライアント側のトランザクションは，ロールバックにマークされません。
トランザクションタイムアウト発生後に，javax.transaction.UserTransaction.commit メソッドを呼び出した場合	java.lang.IllegalStateException 例外が発生します。
トランザクションタイムアウト発生後に javax.transaction.UserTransaction.rollback メソッドを呼び出した場合	java.lang.IllegalStateException 例外が発生します。
トランザクションタイムアウト後の javax.transaction.UserTransaction.getStatus メソッドの戻り値	javax.transaction.Status.STATUS_NO_TRANSACTION が返されます。
javax.ejb.SessionSynchronization を実装した EJB の beforeCompletion メソッド，または afterCompletion メソッドからの EJB 呼び出し	呼び出せません。呼び出すと，javax.transaction.SystemException 例外が発生します。

2.5　プロセスの依存関係とプロセスの再起動方法

ベーシックモードの場合，プロセスの依存関係とプロセスの再起動方法が J2EE サーバモードの場合と異なります。プロセスの依存関係とプロセスの再起動方法を次に示します。

2.5.1　ベーシックモードの場合のプロセスの依存関係

ベーシックモードの場合のプロセスの依存関係を次の表に示します。

表 2-27　ベーシックモードの場合のプロセスの依存関係

プロセスの種類	前提プロセス
パフォーマンストレーサ	－
スマートエージェント※1	－
CTM ドメインマネジャ※1	スマートエージェント
CORBA ネーミングサービス	－
CTM デーモン※1	• パフォーマンストレーサ • スマートエージェント • CTM ドメインマネジャ • ネーミングサービス
SFO サーバ	• パフォーマンストレーサ
J2EE サーバ	• パフォーマンストレーサ • スマートエージェント※2 • CTM ドメインマネジャ※2 • CORBA ネーミングサービス • CTM デーモン※2
Web サーバ	－

（凡例）

－：前提プロセスがない

注※1

CTM を使用する場合に起動するプロセスです。

注※2

CTM を使用する場合に前提となるプロセスです。

2.5.2　ベーシックモードの場合のプロセスの再起動方法

ベーシックモードの場合のシステムでプロセスが異常終了時のプロセスの再起動方法について説明します。

なお，CTM に関連するプロセスが異常終了して再起動できない場合は，次の手順で対処してください。

ベーシックモードの場合のプロセスの再起動（回復）の手順を次に示します。

表 2-28　ベーシックモードの場合の再起動の手順

異常プロセス	開始コマンド	再起動（回復）手順
データベースサーバ	−	DB サーバを再起動してください。
パフォーマンストレーサ	cprfstart	パフォーマンストレーサを再起動してください。
スマートエージェント※1	osagent	スマートエージェントを再起動してください。
CTM ドメインマネジャ※1	ctmdmstart	CTM ドメインマネジャを再起動してください。
CORBA ネーミングサービス	nameserv	次に示す手順で対処してください。なお，CTM を使用しない場合，手順 1 および手順 4 は不要です。 1. CTM デーモンの強制停止 2. J2EE サーバの強制停止 3. CORBA ネーミングサービスの再起動 4. CTM デーモンの再起動 5. J2EE サーバ再起動
CTM デーモン※1	ctmstart	次に示す手順で対処してください。 1. CORBA ネーミングサービスの強制停止 2. J2EE サーバの強制停止 3. CORBA ネーミングサービスの再起動 4. CTM デーモンの再起動 5. J2EE サーバの再起動
CTM レギュレータ	−	CTM デーモンによって自動再起動されるため，再起動は不要です。
SFO サーバ※2	cjstartsv	SFO サーバを再起動してください。
J2EE サーバ	cjstartsv	J2EE サーバを再起動してください。
Web サーバ	−	Web サーバを再起動してください。

（凡例）

−：使用する製品によって開始コマンドが異なる，または該当する開始コマンドがない

注※1

CTM を使用する場合に起動するプロセスです。

注※2

メモリセッションフェイルオーバ機能を使用する場合に起動するプロセスです。

2.6　データソースから DB Connector への移行方法

以前のバージョンで，データソースを使用していた場合，J2EE サーバの移行コマンドがデータソースを DB Connector へ変換します。変換後にコマンドで取得した Connector 属性ファイルと旧バージョンでコマンドを使用して取得したデータソース属性ファイルの対応関係を表に示します。表 2-29 から表 2-31 は，変換後に cjgetrarprop コマンドで取得した Connector 属性ファイルの値を示します。表 2-29 から表 2-31 の「値」の列の括弧付きの数字は，表 2-32 または表 2-33 の括弧付きの数字に指定されていた値が設定されることを意味します。「デフォルト」はデフォルト値が設定されることを意味します。

表 2-29　変換後に cjgetrarprop コマンドを使用して取得した Connector 属性ファイル

タグ名	値
<hitachi-connector-property>	−
<description>	表 2-32 の(1)
<display-name>	表 2-32 の(2)
<icon>	−
<small-icon>	デフォルト
<large-icon>	デフォルト
<vender-name>	デフォルト
<spec-version>	デフォルト
<eis-type>	デフォルト
<version>	デフォルト
<license>	−
<license-required>	デフォルト
<resourceadapter>	−
<managedconnectionfactory-class>	デフォルト
<connectionfactory-interface>	デフォルト
<connectionfactory-impl-class>	デフォルト
<connection-interface>	デフォルト
<connection-impl-class>	デフォルト
<transaction-support>	デフォルト
<config-property>	−
<description>	デフォルト
<config-property-name>	指定されるプロパティについては表 2-30 参照
<config-property-type>	デフォルト
<config-property-value>	指定される値については表 2-30 参照

タグ名	値
<reauthentication-support>	−
<security-permission>	−
<description>	デフォルト
<security-permission-spec>	デフォルト
<runtime>	−
<property>	−
<property-name>	指定されるプロパティについては表 2-31 参照
<property-type>	デフォルト
<property-value>	指定される値については表 2-31 参照
<property-default-value>	デフォルト

（凡例）−：該当しない

表 2-30　Connector 属性ファイルの<config-property>タグに指定されるプロパティと値

<config-property-name>タグに指定されるプロパティ	<config-property-value>タグに指定される値
loginTimeout	表 2-33 の(12)
networkProtocol	表 2-33 の(13)
serverName	表 2-33 の(14)
portNumber	表 2-33 の(15)
databaseName	表 2-33 の(16)
description	表 2-33 の(17)
DBHostName	表 2-33 の(18)
DBEnv	表 2-33 の(19)
encodLang	表 2-33 の(20)
JDBC_IF_TRC	表 2-33 の(21)
SV_EVENT_TRC	表 2-33 の(22)
TRC_NO	表 2-33 の(23)
uapName	表 2-33 の(24)
bufSize	表 2-33 の(25)
rowSize	表 2-33 の(26)
OSAuthorize	表 2-33 の(27)
HiRDBCursorMode	表 2-33 の(28)
blockUpdate	表 2-33 の(29)

<config-property-name>タグに指定されるプロパティ	<config-property-value>タグに指定される値
executeDirectMode	表 2-33 の(30)
SQLWarningIgnore	表 2-33 の(31)
LONGVARBINARY_Access	表 2-33 の(32)
XAOpenString	表 2-33 の(33)
XACloseString	表 2-33 の(34)
RMID	表 2-33 の(35)
XAThreadMode	表 2-33 の(36)
XALocalCommitMode	表 2-33 の(37)
PreparedStatementPoolSize	0
CallableStatementPoolSize	0
CancelStatement	false
logLevel	デフォルト

表 2-31　Connector 属性ファイルの<property>タグに指定されるプロパティと値

<property-name>タグに指定されるプロパティ	<property-value>タグに指定される値
MinPoolSize	表 2-32 の(3)
MaxPoolSize	表 2-32 の(4)
LogEnabled	デフォルト
User	表 2-33 の(38)
Password	表 2-33 の(39)
ValidationType	デフォルト
ValidationInterval	デフォルト
RetryCount	表 2-32 の(10)
RetryInterval	表 2-32 の(11)
ConnectionTimeout	ejbserver.connectionpool.connection.timeout に値が指定されている場合はその値。指定されていない場合はデフォルト。
SweeperInterval	ejbserver.connectionpool.sweeper.interval に値が指定されている場合はその値。指定されていない場合はデフォルト。
RequestQueueEnable	false
RequestQueueTimeout	デフォルト
WatchEnabled	デフォルト
WatchInterval	デフォルト

<property-name>タグに指定されるプロパティ	<property-value>タグに指定される値
WatchThreshold	デフォルト
WatchWriteFileEnabled	デフォルト

表 2-32　旧バージョンで cjgetresprop コマンドによって取得したデータソース属性ファイル

タグ名	項目番号
<hitachi-datasource-property>	−
<description>	(1)
<display-name>	(2)
<property>	−
<name>	指定されるプロパティについては表 2-33 参照
<XADataSource>	指定される値については表 2-33 参照
<PoolConfiguration>	−
<MinimumSize>	(3)
<MaximumSize>	(4)
<Threshold>	(5)
<GrowthIncrement>	(6)
<WaitTimeout>	(7)
<ConnectionTimeout>	(8)
<SweeperInterval>	(9)
<RetryCount>	(10)
<RetryInterval>	(11)

（凡例）−：該当しない

表 2-33　データソース属性ファイルの<property>タグに指定されるプロパティ

<name>タグに指定されるプロパティ	<XADataSource>タグに指定される値
LoginTimeout	(12)
NetworkProtocol	(13)
ServerName	(14)
PortNumber	(15)
DatabaseName	(16)
Description	(17)
DBHostName	(18)
DBEnv	(19)

<name>タグに指定されるプロパティ	<XADataSource>タグに指定される値
EncodLang	(20)
JDBC_IF_TRC	(21)
SV_EVENT_TRC	(22)
TRC_NO	(23)
UapName	(24)
BufSize	(25)
RowSize	(26)
OSAuthorize	(27)
HiRDBCursorMode	(28)
BlockUpdate	(29)
ExecuteDirectMode	(30)
SQLWarningIgnore	(31)
LONGVARBINARY_Access	(32)
XAOpenString	(33)
XACloseString	(34)
RMID	(35)
XAThreadMode	(36)
XALocalCommitMode	(37)
User	(38)
Password	(39)

なお，移行コマンドでデータソースから DB Connector へ変換するときに，すでに同一表示名称（display-name）の DB Connector，またはリソースアダプタが存在した場合には，表示名称の最後に連番を付与し，表示名称を一意にします。DB Connector による接続については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6.1　DB Connector による接続の概要」を参照してください。

2.7　パフォーマンスチューニング

ここでは，ベーシックモードで使用できるパフォーマンスチューニングについて説明します。

2.7.1　チューニング項目

ベーシックモードで使用できるパフォーマンスチューニングを次の表に示します。

表 2-34　ベーシックモードで使用できるパフォーマンスチューニング

チューニング項目			使用可否	参照先※1
同時実行数の最適化	リクエスト処理スレッド数（インプロセス HTTP サーバ使用時）		○	8.3.3
	Web アプリケーションの同時実行数	URL グループ単位	○	8.3.4
		Web アプリケーション単位	○	
		Web コンテナ単位	○	
	Enterprise Bean の同時実行数	Stateless Session Bean の同時実行数	○	8.3.5
		Stateful Session Bean の同時実行数	○	
		Entity Bean の同時実行数	○	
		Message-driven Bean の同時実行数	○	
	CTM で制御する同時実行数		○	8.3.6
Enterprise Bean の呼び出し方法の最適化	ローカルインタフェースの使用		○	8.4.1
	リモートインタフェースのローカル呼び出し機能の使用		○	8.4.2
	リモートインタフェースの参照渡し機能の使用		○	8.4.3
データベースアクセス方法の最適化	コネクションプーリング		○	8.5.1
	ステートメントプーリング		×	－
タイムアウトの設定	Web フロントシステムでのタイムアウト	Web サーバ側で設定するクライアントからのリクエスト受信，およびクライアントへのデータ送信のタイムアウト	○	8.6.2
		リダイレクタ側で設定する Web コンテナへのデータ送信のタイムアウト	○	
		リダイレクタ側で設定する Web コンテナからのデータ受信のタイムアウト	○	
		Web コンテナ側で設定するリダイレクタからのデータ受信のタイムアウト	○	
		Web コンテナ側で設定するリダイレクタへのデータ受信のタイムアウト	○	
	バックシステムでのタイムアウト	EJB クライアント側で設定する Enterprise Bean のリモート呼び出し	○	8.6.3

チューニング項目			使用可否	参照先※1
タイムアウトの設定	バックシステムでのタイムアウト	（RMI-IIOP 通信）と JNDI によるネーミングサービス呼び出しのタイムアウト	○	8.6.3
		EJB クライアント側で設定する CTM から Enterprise Bean 呼び出しのタイムアウト	○	
	EJB コンテナ側で設定するデータベースのトランザクションタイムアウト（DB Connector を使用した場合）		○※2	8.6.4
	データベースのタイムアウト		○	8.6.6
	J2EE アプリケーションのメソッドタイムアウト		○	8.6.7
Web アプリケーションの動作の最適化	静的コンテンツと Web アプリケーションの配置の切り分け		○	8.7.1
	静的コンテンツのキャッシュ		○	8.7.2
	リダイレクタによるリクエストの振り分け		○	8.7.3
CTM の動作の最適化	CTM ドメインマネジャおよび CTM デーモンの稼働状態を監視する間隔の設定		○	8.8.1
	負荷状況監視間隔の設定		○	8.8.2
	CTM デーモンのタイムアウト閉塞の設定		○	8.8.3
	CTM で振り分けるリクエストの優先順位の設定		○	8.8.4
そのほかの項目のチューニング	Persistent Connection についてのチューニング		○	8.9

（凡例）○：使用できる。 ×：使用できない。 －：該当しない。

注※1

マニュアル「アプリケーションサーバ システム設計ガイド」の節または項を参照してください。

注※2

ベーシックモードの場合，CMT を使用するときのトランザクションのタイムアウトは，Enterprise Bean，インタフェースまたはメソッドの属性では設定できません。ベーシックモードの場合，CMT を使用するときのトランザクションのタイムアウトは，usrconf.properties で定義してください。

これらのチューニング項目のうち，ベーシックモードで使用できる機能や，チューニングパラメタの異なるものについて説明します。

2.7.2　データベースへのアクセス方法を最適化するためのチューニングパラメタ

ベーシックモードの場合に，コネクションプーリングで使用できる機能を次の表に示します。なお，コネクションプーリングで使用できる各機能の設計時の指針については，マニュアル「アプリケーションサーバ システム設計ガイド」の「8.5.1　コネクションプーリングを使用する」を参照してください。

表 2-35　コネクションプーリングで使用できる機能（ベーシックモードの場合）

機能	使用可否
プールするコネクションの最大値と最小値を指定する	○
プール内のコネクションを検査して無効なコネクションを破棄する	○
コネクション取得失敗時にリトライする	△
スイーパによって使わないコネクションをプールから削除する※	○
コネクションウォーミングアップによってあらかじめコネクションをプールしておく	×
コネクション枯渇時のコネクション取得要求を取得待ちキューに入れる	×
コネクションプール内の不要なコネクションを段階的に減少させる	×
コネクションプールをクラスタ化する	×
コネクションプールの未使用コネクションがしきい値以下になったとき，コネクションを追加する	○
コネクション取得待ち時間を指定する	○

（凡例）○：使用できる。 △：指定しても無視される。 ×：使用できない。

注※

ただし，スイーパを実行する処理は負荷が高いため，性能に影響が出ます。このため，スイーパを使用する場合は，自動削除の間隔を 1 時間以上の値にすることをお勧めします。

次に，ベーシックモードの場合にコネクションプーリングをチューニングするパラメタについて説明します。これらのチューニングパラメタは，データソース単位に設定します。

チューニングパラメタについて，次の表に示します。これらは，サーバ管理コマンドの CUI で設定します。CUI を使用する場合は，属性ファイルを編集してください。

表 2-36　ベーシックモードの場合のコネクションプーリングのチューニングパラメタ

設定項目	設定方法	設定個所
コネクションプールにプールするコネクションの最小値	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<MinimumSize>タグ
コネクションプールにプールするコネクションの最大値	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<MaximumSize>タグ
コネクションの生成時刻からコネクションを自動破棄するかを判定するまでの時間	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<ConnectionTimeout>タグ
コネクションの自動破棄（コネクションスイーパ）が動作する間隔	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<SweeperInterval>タグ
コネクションプールにコネクションを追加するためのしきい値	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<Threshold>タグ
コネクションプールにコネクションを追加する場合の追加数	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<GrowthIncrement>タグ
コネクション取得待ち時間	CUI（cjsetresprop）	データソース属性ファイルの<PoolConfiguration>タグ下の<WaitTimeout>タグ

2.8　データベースとアプリケーション間の文字コード変換（ベーシックモード）

アプリケーションからデータベースに文字データを格納するときには，Unicode からデータベースに設定されている文字コードに変換して格納する必要があります。詳細については「4.4 データベースとアプリケーション間の文字コードの変換」を参照してください。

2.9　下位互換用のクラスローダ構成

デフォルトのクラスローダ構成と，アプリケーションサーバ Version 5（05-01）以前のクラスローダ構成では内容が異なります。このため，下位互換用にクラスローダ構成を変更するためのオプションがあります。アプリケーションサーバ Version 5（05-01）以前のバージョンからアップグレードインストールした場合は，自動的に下位互換用のクラスローダ構成になります。

下位互換用のクラスローダ構成に変更するには，J2EE サーバ用ユーザ定義ファイル（<Application Server のインストールディレクトリ>¥CC¥server¥usrconf¥ejb¥<サーバ名称>¥usrconf.properties）に次の設定をします。

動作モードをベーシックモードに設定

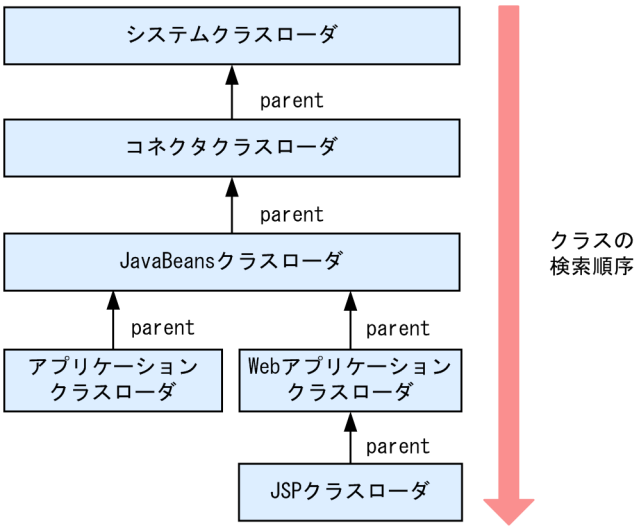
ejbserver.server.j2ee.feature=1.3basic

ローカル呼び出し最適化機能の適用範囲をなしに設定

ejbserver.rmi.localinvocation.scope=none

下位互換用のクラスローダ構成を次に示します。

図 2-1　下位互換用のクラスローダ構成



各クラスローダの内容を次に示します。

- **システムクラスローダ**
  Application Server のコンポーネントが提供するクラスをロードします。
- **コネクタクラスローダ**
  下位互換用クラスローダ構成のときには，リソースアダプタを使用することはできないため，このクラスローダでは何もロードされません。
- **JavaBeans クラスローダ**
  JavaBeans リソースのクラスをロードします。
- **アプリケーションクラスローダ**
  アプリケーション内の EJB-JAR およびライブラリ JAR に含まれるクラスをロードします。アプリケーションごとに存在します。
- **Web アプリケーションクラスローダ**

J2EE アプリケーション内の WAR に含まれるクラスをロードします。WAR ごとに存在します。
WAR から J2EE アプリケーション内の EJB 呼び出しができるように，スタブ関連のクラスもロードします。

- JSP クラスローダ
  JSP ファイルおよびタグファイルをコンパイルしたときに生成されたクラスをロードします。JSP ごとに存在します。

2.10　ベーシックモードの設定

ここでは，ベーシックモードの設定，データソースの設定について説明します。

動作モードは，usrconf.properties の ejbserver.server.j2ee.feature キーで変更できます。ベーシックモードを使用する場合には，「ejbserver.server.j2ee.feature=1.3basic」を指定します。

なお，旧バージョンでベーシックモードを使用していた場合，アプリケーションサーバをアップグレードインストールしたときには，J2EE サーバの動作モードはベーシックモードになっています。新規インストールした場合には，J2EE サーバの動作モードは 1.4 モードになっています。

2.10.1　データソースの設定

ベーシックモードを使用する場合，データソースを使用してデータベースに接続できます。データソースを利用して接続できるデータベースには，HiRDB，Oracle9i があります。なお，データソースを使用する場合，データベースとの接続には DABroker Library を使用します。また，使用できるトランザクションの種類は，ローカルトランザクションだけとなります。

! 注意事項

データソースも DB Connector と同様の使用方法で J2EE アプリケーションから利用できますが，データベース接続には 1.4 モードでの DB Connector の使用をお勧めします。DB Connector を利用したデータベース接続は，データソースを利用したデータベース接続と比較して，トランザクション管理のパフォーマンスを向上できます。

また，データベース接続にデータソースを使用できるのは，J2EE サーバの動作モードがベーシックモードの場合だけです。1.4 モードでデータベースに接続する場合には，DB Connector を使用してください。

アプリケーションサーバでは，データソースから DB Connector への移行をサポートしています。データソースから DB Connector への移行方法については，「2.6　データソースから DB Connector への移行方法」を参照してください。

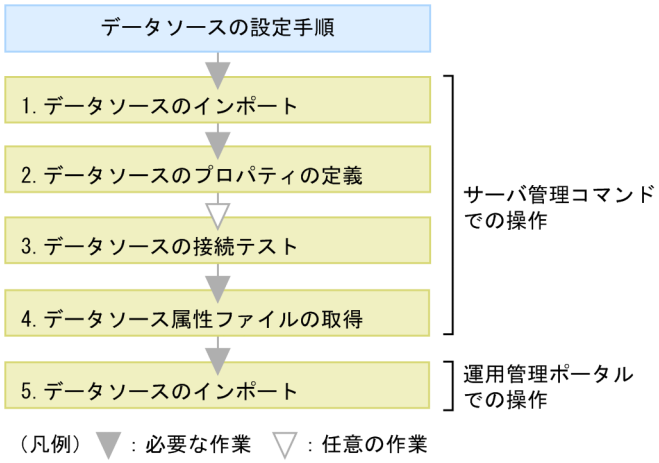
運用管理ポータルでシステムを構築する場合と，ファイル編集とコマンド実行でシステムを構築する場合で，データソースの設定方法が異なります。それぞれの設定方法について説明します。

(1)　運用管理ポータルでシステムを構築する場合

運用管理ポータルでは，開発環境（サーバ管理コマンドが動作する環境）で設定したデータソースを，運用環境（データソースを使用する環境）に移す作業をサポートしています。開発環境でサーバ管理コマンドを使用してデータソースのプロパティの設定，接続の確認などを実施したあとで，運用管理ポータルを使用して運用環境にデータソースをインポートします。

データソースの設定の流れを次の図に示します。

図 2-2　データソースの設定の流れ（運用管理ポータルでのシステム構築）



図中の 1.～5.について説明します。なお，ここでは，開発環境と運用環境は，同じホストにある別々の J2EE サーバ上にあると仮定して説明します。

1. **開発環境で，サーバ管理コマンドを使用してデータソースをインポートします。**

   cjimportres コマンドを使用して，データソースをインポートします。

2. **開発環境で，サーバ管理コマンドを使用してデータソースのプロパティを定義します。**

   cjgetresprop コマンドでデータソース属性ファイルを取得し，ファイル編集後に，cjsetresprop コマンドで編集内容を反映させます。

   なお，使用するデータベースの種類によって，プロパティの定義内容が異なります。

3. **開発環境で，サーバ管理コマンドを使用してデータソースの接続テストを実施します。**

   cjtestres コマンドを使用して，データソースの接続テストを実施します。

4. **運用環境で設定済みのデータソースから，サーバ管理コマンドを使用してデータソース属性ファイルを取得します。**

   cjgetresprop コマンドを使用して，データソース属性ファイルを取得します。

   参考

   - 同一のデータソースは，同一の J2EE サーバにインポートできません。開発環境の J2EE サーバと運用環境の J2EE サーバが同一の場合は，運用環境の J2EE サーバにデータソースをインポートする前に，開発環境の J2EE サーバにあるデータソースを削除してください。
   - 開発環境のホストと運用環境のホストが異なる場合には，開発環境のホストにあるデータソースを運用環境のホストに移動してください。

     開発環境のホストにあるデータソースから cjgetresprop コマンドでデータソース属性ファイルを取得し，データソース設定ファイルを用意します。データソース設定ファイルの格納場所を次に示します。

     Windows の場合

     　<製品のインストールディレクトリ>¥CC¥admin¥templates

     UNIX の場合

     　/opt/Cosminexus/CC/admin/templates

     取得および用意したデータソース属性ファイルとデータソース設定ファイルを，運用環境のホスト（Management Server 稼働ホスト）にコピーします。

5. **運用管理ポータルの「論理サーバのアプリケーション管理」で，運用環境の J2EE サーバにデータソースをインポートします。**

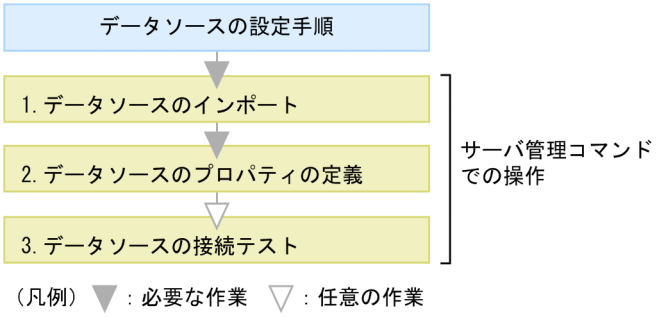
［データソースのインポート］画面で，データソースをインポートします。なお，インポート時に指定するファイルは，JDBC ドライバファイル，データソース属性ファイル，およびデータソース設定ファイルです。JDBC ドライバファイルは JDBC ドライバのクラスライブラリファイル，属性ファイルと設定ファイルはリソースのプロパティや使用するドライバのクラスなどの情報を定義したファイルです。

(2)　ファイル編集とコマンド実行でシステムを構築する場合

ファイル編集とコマンド実行でシステムを構築する場合，サーバ管理コマンドを使用してデータソースのプロパティの設定，接続の確認などを実施してください。

データソースの設定手順を次の図に示します。

図 2-3　データソースの設定手順（ファイル編集とコマンド実行でのシステム構築）



図中の 1.～3.について説明します。

1. **サーバ管理コマンドを使用してデータソースをインポートします。**
   cjimportres コマンドを使用して，データソースをインポートします。
2. **サーバ管理コマンドを使用してデータソースのプロパティを定義します。**
   cjgetresprop コマンドでデータソース属性ファイルを取得し，ファイル編集後に，cjsetresprop コマンドで編集内容を反映させます。
   なお，使用するデータベースの種類によって，プロパティの定義内容が異なります。
3. **サーバ管理コマンドを使用してデータソースの接続テストを実施します。**
   cjtestres コマンドを使用して，データソースの接続テストを実施します。

参考

次のような場合，データソースをインポートすることで，効率良くデータソースを設定できます。

- 開発環境で設定したデータソースを，運用環境にインポートして使用する場合
- 運用環境ですでに動いているデータソースを，増設した J2EE サーバにインポートして使用する場合

2.10.2　データベースと接続するための設定（データソースを使用する場合）

ここでは，ベーシックモードでのリソース接続について説明します。

データベースと接続するために必要なデータソースの設定をします。データソースの設定は，サーバ管理コマンドで実行します。

データソースを使用してデータベースと接続するには，J2EE サーバをベーシックモードで動作させる必要があります。

データソースの設定の流れを次に示します。

1. DABroker Library の環境設定をします。
2. データソースをインポートします。
3. プロパティを定義します。
4. 接続を確認します。

インポートしたデータソースは，削除およびコピーができます。

(1)　DABroker Library の環境設定

DABroker Library は，DABroker Library の環境設定ユティリティ（Windows の場合），または DABroker Library 動作環境定義ファイル（UNIX の場合）で設定します。設定方法については，「4.2 DABroker Library を使用したデータベース接続の概要」を参照してください。

(2)　データソースのインポート

データソースまたは JDBC ドライバを J2EE サーバにインポートします。

次に示すコマンドを実行してデータソースをインポートします。

実行形式

```
cjimportres ［<サーバ名称>］ [-nameserver <プロバイダ URL >］ -type datasource -resname <データソースの表示名> -c <データソース設定ファイルパス> -f < JDBC ドライバファイルパス>
```

実行例

```
cjimportres MyServer -type datasource -resname JdbcDbpsv -c JdbcDbpsvConf.xml -f
JdbcDbpsv.jar
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

注意事項

- JDBC のデータソースとしてインポートできる DABroker Library の JAR ファイルは，JdbcDbpsv.jar だけです。JdbcDbpsvEX.jar はサポートされていません。また，JdbcDbpsv.jar は，「< Application Server のインストールディレクトリ>¥DABJ」ディレクトリ（Windows の場合）または「/opt/Cosminexus/DABJ」ディレクトリ（UNIX の場合）に配置されている必要があります。
- データソースの表示名には，すでにあるデータソースの名前やメールサーバ構成情報の名前を指定しないでください。
- インポート時に指定した JAR ファイル名は，作業ディレクトリ中のディレクトリ名として用いられます。作業ディレクトリのパス長がプラットフォームの上限に達しないように JAR ファイル名を指定してください。作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1(2)　作業ディレクトリのパス長の見積もり式」を参照してください。
- データソースを削除するまで，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の ejbserver.http.port キーの値は変更しないでください。変更した場合は，データソースを再作成してください。

(3)　データソースのプロパティ定義

データソースのプロパティを定義します。プロパティの設定手順については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「3.5　属性ファイルによるプロパティの設定」を参照してください。

(a)　編集する属性ファイル

データソース属性ファイル

(b)　編集する属性ファイルの取得と属性の設定

● 属性ファイルの取得

次に示すコマンドを実行して，データソースのデータソース属性ファイルを取得します。

実行形式

```
cjgetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type datasource -resname <データソースの表示名> -c <属性ファイルパス>
```

実行例

```
cjgetresprop MyServer -type datasource -resname JdbcDbpsv -c JdbcDbpsvProp.xml
```

● 属性の設定

次に示すコマンドを実行して，データソース属性ファイルの値を反映します。

実行形式

```
cjsetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type datasource -resname <データソースの表示名> -c <属性ファイルパス>
```

実行例

```
cjsetresprop MyServer -type datasource -resname JdbcDbpsv -c JdbcDbpsvProp.xml
```

(c)　編集する属性設定項目

データソースのプロパティ設定項目を次に示します。

- データソースの一般情報
- 実行時プロパティ
- コネクションプーリング情報

● データソースの一般情報

データソースの一般情報の設定項目を次に示します。

項目	対応するタグ
データソースの説明	<description>
データソース名	<display-name>

● 実行時プロパティ

データソースの実行時プロパティ（<property>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<name>
XADataSource インタフェースのプロパティ値	<XADataSource>

注　データソースについて設定する項目は，インポートしたデータベースの種類によって異なります。データベースに固有の項目については，そのデータベースのマニュアルを参照してください。

定義するプロパティの数だけ，<property>タグの設定を繰り返してください。

プロパティの設定項目については，「2.11.4　データソース属性ファイル」を参照してください。

● コネクションプーリング情報

プーリングとはデータソースとデータベースのコネクションをあらかじめ確立しておき，接続時のオーバーヘッドを減らす手法です。データソースがデータベースコネクションプーリングをサポートする場合，データベースコネクションプールの管理方法を指定できます。コネクションプーリング情報は，プールにあらかじめどのくらいのコネクションを用意しておくのか，また，プールにある不要なコネクションを削除するタイミングなどを指定します。

コネクション取得時にエラーとなった場合に，アプリケーションにエラーを返さないで，繰り返し取得を試みることができます。プーリングしてあるすべてのコネクションが使用中で一時的に取得失敗になる場合など，本機能によってアプリケーションにエラーを返すことなくコネクションを再取得できます。

データソースのコネクションプーリング情報，およびコネクション再取得情報（<PoolConfiguration>タグ）の設定項目を次に示します。

項目	説明	対応するタグ
コネクションプールにプールするコネクションの最小値	データベースコネクションプールに割り当てるデータベースコネクションの最小数を指定します。この最小値に達すると，データベースコネクションはそれ以上削除されません。 定常状態の同時コネクション数を指定してください。	<MinimumSize>
コネクションプールにプールするコネクションの最大値	データベースコネクションプールで使用できるデータベースコネクションの最大数を指定します。この最大値に達すると，それ以上のデータベースコネクションは追加されません。-1 を指定すると，無制限にコネクションを使用できます。この場合，データベースで許容される最大コネクション数によって最大数は制限されます。 同時コネクション数の最大値を指定してください。	<MaximumSize>
コネクションプールにプールするコネクションのしきい値	データベースコネクションプールにデータベースコネクションを追加するしきい値を指定します。つまり，プールにある未使用コネクションの数がここで指定した値以下になった場合，データベースコネクションをプールに追加します。	<Threshold>
コネクションプールにプールするコネクションの増分値	しきい値に達した場合にデータベースコネクションプールに追加するデータベースコネクションの数を指定します。	<GrowthIncrement>

項目	説明	対応するタグ
コネクション取得待ち時間（秒）	データベースコネクションプールからデータベースコネクションを取得しようとするときに，コネクション取得のリトライをするまでのアプリケーションの待ち時間を秒単位で指定します。指定した時間が経過してもリトライしない場合，エラーが返されます。コネクションが使用できない場合に待たないようにするときは 0 を指定し，コネクションを取得できるまで待つようにするときは-1 を指定します。	<WaitTimeout>
コネクション接続タイムアウト値（秒）	データベースコネクションの保持時間を秒単位で指定します。指定した時間が経過すると，次のスイーパサイクルでデータベースコネクションを削除します。コネクションタイムアウトの経過後，最小数よりも多く使用できるコネクションがプール内にある場合，そのコネクションは削除されます。プール内に最小数のコネクションだけがある場合，そのコネクションはリオープンされます。	<ConnectionTimeout>
スイーパ起動時間	スイーパサイクルの間隔を秒単位で指定します。スイーパサイクルは，すべてのデータベースコネクションをチェックし，<ConnectionTimeout>で指定した時間よりも古いコネクションを削除します。	<SweeperInterval>
コネクション取得のリトライ回数	コネクションの取得に失敗して，再度取得を行う場合のリトライ回数を指定します。	<RetryCount>
コネクション取得をリトライするまでの時間（秒）	コネクションの取得に失敗して，再度取得を行う場合のリトライ間隔を指定します。	<RetryInterval>

プーリングを使用しない場合，すべてのフィールドに 0 を指定します。この場合，データソースは，非プーリングデータソースのように動作します。つまり，データベースコネクションを保持しません。また，データベースコネクションが必要な場合，要求側は，データソースがインタフェースを取得するまで待たなければなりません。したがって，性能にかなりの影響を与えることになります。

(4)　データソースの接続テスト

データソースに設定した内容が正しいかどうか，接続テストによって検証します。

次に示すコマンドを実行してデータソースの接続を確認します。

実行形式

cjtestres ［<サーバ名称>］ ［-nameserver <プロバイダ URL >］ -type datasource -resname <データソースの表示名> ［-resname <データソースの表示名>］

実行例

```
cjtestres -type datasource -resname Myds1
```

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト)」を参照してください。

2.11　ベーシックモードで使用するファイル

ここではベーシックモードで使用するファイルについて説明します。

2.11.1　usrconf.properties（J2EE サーバ用ユーザプロパティファイル）

J2EE サーバの動作を設定するファイルです。

ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.4　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。ここでは，ベーシックモード独自の指定内容について説明します。

(1)　ejbserver.jdbc から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

キー名称	内容	省略値	VR
ejbserver.jdbc.propertyInfo.BUF_SIZE	該当キーに値が指定されている場合，J2EE サーバが JDBC コネクションを取得するとき，BUF_SIZE キーに該当値を設定したプロパティを作成して，構成ソフトウェアである DABroker Library の java.sql.Driver.connect の引数に渡します。 なお，このプロパティは，ベーシックモードでだけ指定できます。	なし	－
ejbserver.jdbc.propertyInfo.ENCODELANG	該当キーに値が指定されている場合，J2EE サーバが JDBC コネクションを取得するとき，ENCODELANG キーに該当値を設定したプロパティを作成して，構成ソフトウェアである DABroker Library の java.sql.Driver.connect の引数に渡します。 なお，このプロパティは，ベーシックモードでだけ指定できます。	なし	－

（凡例）

－：08-00 より前のバージョンを示します。

(2)　ejbserver.jta から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

キー名称	内容	省略値	VR
ejbserver.jta.TransactionManager.defaultTimeOut	J2EE サーバ上で開始されるトランザクションのトランザクションタイムアウトのデフォルト値（単位：秒）を次の範囲で指定します。 ベーシックモードの場合： 1～2000000	180	－
ejbserver.jta.TransactionManager.enlist.compatibleMode.enabled	J2EE サーバモードがベーシックモードの場合のトランザクション処理に関する仕様を指定します。	true	－

キー名称	内容	省略値	VR
ejbserver.jta.TransactionManager.enlist.compatibleMode.enabled	トランザクション中に，複数回データベースコネクションを取得した場合（コネクションシェアリング機能が有効になる条件を満たしているとき）： 該当キーの値による動作の違いはありません。トランザクション中に 2 回目以降に取得したコネクションも，トランザクションに参加して動作します。 トランザクション中に，複数回 DB コネクションを取得した場合（コネクションシェアリング機能が有効になる条件を満たしていないとき）： 該当キーの値による動作の違いは次のとおりです。 • **true を指定したとき：** トランザクション中に 2 回目以降に取得したコネクションは，トランザクションに参加しないで動作します。 • **false を指定したとき：** トランザクション中に 2 回目にコネクションを取得した場合に，java.sql.SQLException が発生します。 複数回データベースコネクションを取得したあとで，トランザクションを開始した場合： 該当キーの値による動作の違いは次のとおりです。 • **true を指定したとき：** トランザクション開始前に取得したコネクションは，トランザクションに参加しないで動作します。 • **false を指定したとき：** トランザクション開始時に，javax.transaction.SystemException が発生します。	true	－

（凡例）

－：08-00 より前のバージョンを示します。

(3)　ejbserver.server から始まるキー

指定できるキーについて次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

キー名称	内容	省略値	VR
ejbserver.server.j2ee.feature	Component Container の動作モードを指定します。 • 1.3basic ベーシックモード（単一 DB だけのリソースをトランザクションで使用するシステムに適用） なお，旧バージョンでベーシックモードを使用していた場合，アプリケーションサーバを更新インストールしたときには，J2EE サーバの動作モードはベーシックモードになります。 新規インストールした場合には，J2EE サーバの動作モードは 1.4 モードになります。 ベーシックモードと 1.4 モードの違いについては，「2.2　ベーシックモードの概要」を参照してください。	1.4	－

（凡例）

－：08-00 より前のバージョンを示します。

2.11.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）

EJB クライアントアプリケーションなどの Java アプリケーションの動作を設定するファイルです。

ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「14.3　usrconf.properties（Java アプリケーション用ユーザプロパティファイル)」を参照してください。ここでは，ベーシックモード独自の指定内容について説明します。

(a)　ejbserver.jta から始まるキー

Java アプリケーション用 usrconf.properties ファイルで指定できる ejbserver.jta から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.jta.TransactionManager.defaultTimeOut	あり	Java アプリケーションで開始されるトランザクションのトランザクションタイムアウトのデフォルト値（単位：秒）を次の範囲で指定します。 ベーシックモードの場合： 1～2000000	－

（凡例）

あり：

Java アプリケーション用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

－：

Java アプリケーション用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。デフォルト値の詳細については，「2.11.1(2)　ejbserver.jta から始まるキー」を参照してください。

2.11.3　データソース設定ファイル

データソース設定ファイルは，次に示す DOCTYPE 宣言を持ちます。

```
<!DOCTYPE hitachi-datasource-configuration PUBLIC '-//Hitachi, Ltd.//DTD DataSource Configuration 6.0//EN' 'file:///<Application Server のインストールディレクトリ>/CC/admin/dtds/hitachi-datasource-configuration_6_0.dtd'>
```

! 注意事項

サンプルファイルで提供している属性ファイルなどをサーバ管理コマンド以外の方法で使用する場合，DOCTYPE 宣言内の URL の指定を URI に書き直す必要があります。

例

変更前

```
<!DOCTYPE hitachi-datasource-configuration PUBLIC '-//Hitachi, Ltd.//DTD DataSource Configuration 6.0//EN' 'file:///C:/Program%20Files/Hitachi/Cosminexus/CC/admin/dtds/hitachi-datasource-configuration_6_0.dtd'>
```

変更後

<!DOCTYPE hitachi-datasource-configuration PUBLIC '-//Hitachi, Ltd.//DTD DataSource Configuration 6.0//EN' 'file:///<Application Server のインストールディレクトリ>/CC/admin/dtds/hitachi-datasource-configuration_6_0.dtd'>

データソース設定ファイルは，次に示すタグを持つ xml ファイルです。

タグ名	出現パターン	説明
<hitachi-datasource-configuration>	1 回	ルートタグ。
└ <XADataSource>	0 または 1 回	トランザクション利用の場合に使用するドライバのクラス名。 指定できる値を次に示します。 • javax.sql.XADataSource インタフェースを実装しているクラス。 • java.sql.Driver インタフェースを実装しているクラス。

cjimportres コマンドでデータソースをインポートするときに使用するデータソース設定ファイルは，あらかじめエディタなどを使って作成したものを使用するか，次のディレクトリにあるサンプルファイルを使用します。サンプルファイルでは，デフォルトの設定内容が記述されています。

ディレクトリ	ファイル	接続データベース種別	説明
• Windows の場合 <Application Server のインストールディレクトリ>¥CC¥admin¥templates • UNIX の場合 /opt/Cosminexus/CC/admin/templates	JdbcDbpsv_cf.xml	Oracle または HiRDB	ベーシックモード用（JdbcDbpsv.jar）

2.11.4　データソース属性ファイル

データソース属性ファイルは，次に示す DOCTYPE 宣言を持ちます。

<!DOCTYPE hitachi-datasource-property PUBLIC '-//Hitachi, Ltd.//DTD DataSource Property 6.0//EN' 'file:///<Application Server のインストールディレクトリ>/CC/admin/dtds/hitachi-datasource-property_6_0.dtd'>

データソース属性ファイルは，次に示すタグを持つ xml ファイルです。

タグ名	出現パターン	説明
<hitachi-datasource-property>	1 回	ルートタグ。
├ <description>	0 または 1 回	データソースの説明。
├ <display-name>	1 回	データソース表示名。 1≦文字列長≦240 変更する場合，英数字，およびアンダースコア「_」を使用できます。

タグ名	出現パターン	説明
<display-name>	1 回	データソース表示名。 1≦文字列長≦240 変更する場合，英数字，およびアンダースコア「_」を使用できます。
<property>	0 回以上	各種プロパティ。※1 <property>の指定が複数あり，<name>の設定値が同じ場合，一つにマージされます。その場合，ワーニングメッセージ（KDJE37600-W）を出力し，後ろに定義された<XADataSource>の値が有効になります。
<name>	1 回	プロパティ名。
<XADataSource>	0 または 1 回	XADataSource インタフェースのプロパティ値。 値は変更できます。 タグの追加，削除は禁止です。
<PoolConfiguration>	0 または 1 回	コネクションプールの設定。※1，※2
<MinimumSize>	1 回	最小サイズ。 0≦MinimumSize≦MaximumSize
<MaximumSize>	1 回	最大サイズ。 -1（無制限）または 0≦MaximumSize
<Threshold>	1 回	しきい値。 0 または 1≦Threshold≦MaximumSize 0 を指定した場合，使用できるコネクションがない場合にだけプールにコネクションを追加します。
<GrowthIncrement>	1 回	増分値。 0 または 1≦GrowthIncrement≦2147483647 0 を指定した場合，プールは増加しません。
<WaitTimeout>	1 回	コネクション取得待ち時間（単位：秒）。 -1 または 0≦WaitTimeout≦2147483647 -1 を指定した場合，取得できるまで待ち続けます。
<ConnectionTimeout>	1 回	接続タイムアウト（単位：秒）。 0 または 1≦ConnectionTimeout≦2147483647 0 を指定した場合，アクティブではないコネクションがあってもプールから削除されません。

タグ名	出現パターン	説明
<ConnectionTimeout>	1 回	接続タイムアウト（単位：秒）。 0 または 1≦ConnectionTimeout≦2147483647 0 を指定した場合，アクティブではないコネクションがあってもプールから削除されません。
<SweeperInterval>	1 回	スイーパ起動間隔（単位：秒）。 0 または 1≦SweeperInterval≦2147483647 0 を指定した場合，アクティブではないコネクションがあってもプールから削除されません。
<RetryCount>	1 回	コネクション取得リトライ回数。 0～2147483647
<RetryInterval>	1 回	コネクション取得リトライまでの待ち時間（単位：秒）。 1～2147483647

注※1

cjgetresprop コマンドで出力されたものに対し，タグ名の変更，タグの追加・削除，値の変更をしてはいけません。

注※2

MinimumSize タグを 0 に設定した場合，コネクションプールは機能しません。

2.11.5　コネクションプーリングとコネクションスイーパの動作

データソースのプロパティで設定するコネクションプーリングとコネクションスイーパの動作，コネクションを共有する場合の留意点について説明します。

コネクションプーリングについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.1　コネクションプーリング」を参照してください。コネクションスイーパについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.7　コネクションスイーパ」を参照してください。

(1)　コネクションプーリングの動作

- サーバ上での最初のアクセス時（具体的には最初の javax.sql.DataSource#getConnection メソッド発行時），コネクションプールにプールするコネクションの最小値（<MinimumSize>）の値の JDBC コネクションを確立（接続）しプールします。データベースサーバの制限によって<MinimumSize>の値のコネクションが確立できない場合，<MinimumSize>の値以下の状態になります。
- ユーザアプリケーションプログラムが JDBC コネクションを取得（javax.sql.DataSource#getConnection メソッド）すると，プール内の未使用状態のコネクションが一つ選択され，ユーザアプリケーションプログラムに渡されます。選択された該当コネクションは，プール内で使用中状態となります。
- ユーザアプリケーションプログラムが JDBC コネクションを解放 (java.sql.Connection#close メソッド）すると，解放した JDBC コネクションに異常がなく，再利用できる場合，該当する JDBC コネクションはプール内で未使用状態に戻ります。解放したコネクションが再利用できない場合，該当する JDBC コネクションは破棄されます。
- 未使用状態のコネクションが，コネクションプールにプールするコネクションのしきい値（<Threshold>）の値以下になると，プール内のコネクションを，コネクションプールにプールするコ

ネクションの増分値（<GrowthIncrement>）の値ずつ増加させます。DB サーバの制限によって<GrowthIncrement>の値のコネクションが確立できない場合は，増分は<GrowthIncrement>の値以下になります。

- ユーザアプリケーションプログラムが JDBC コネクションを取得しようとしたときにプールに未使用コネクションがない場合，コネクション取得待ち時間（<WaitTimeout>）の値の間，待たされます。このあと，まだプールに未使用コネクションがない場合，ユーザアプリケーションプログラムに例外が通知されます。

(2)　コネクションスイーパの動作

- 前回のコネクションスイーパの動作が終了してから，スイーパ起動時間（<SweeperInterval>）の値の経過後，コネクションスイーパが動作します。
- コネクションスイーパは，プール内の未使用状態コネクションを次の観点で監視します。
  - 該当コネクションの生成時点からの経過時間が，コネクション接続タイムアウト値（<ConnectionTimeout>）の値未満の場合は何もしません。
  - 該当コネクションの生成時点からの経過時間が，<ConnectionTimeout>の値以上の場合はコネクションを破棄します。
  - プールに残ったコネクション数が，<MinimumSize>の値以下の場合，コネクションスイーパは<MinimumSize>の値までコネクションを確立します。

(3)　コネクションを共有する場合の留意点

コネクションシェア機能は，同一 JavaVM 内という条件で，同一トランザクション（ここでのトランザクションとは javax.transaction.UserTransaction によるトランザクションまたは EJB での CMT によるトランザクションだけ）中での同一データソース（javax.sql.DataSource）へのコネクション取得要求（getConnection メソッド）に対して，アプリケーションが同一の物理コネクションを共有できるようにするものです。これによって，アプリケーションがトランザクション中で同じデータソースから複数回コネクションを取得して SQL を発行した場合，トランザクションコミット時にそれぞれの SQL の処理結果をデータベースにまとめて反映できます。この機能を利用するには，コネクションプールの設定でコネクション数<MinimumSize>を 1 以上に設定する必要があります。なお，コネクションシェア機能は，同一データソース（javax.sql.DataSource）に対し，パラメタなし getConnection メソッドと，パラメタあり getConnection（java.lang.String username，java.lang.String password）メソッド間では，同一トランザクション中でもコネクションを共有できません。また，パラメタありの getConnection（java.lang.String username，java.lang.String password）メソッド間で username が異なる場合，同一トランザクション中でもコネクションを共有できません。

(4)　注意事項

- データソースの表示名（<display-name>）には，すでにあるデータソースの名前やメールサーバ構成情報の名前を指定しないでください。
- コネクションプーリングの設定を変更した場合，設定内容を有効にするために J2EE サーバを再起動してください。
- コネクションプーリング機能を利用する場合には，データベースサーバ側からコネクションを強制的に切断する機能（タイムアウトなど）を利用しないでください。

2.11.6　データソースへのリファレンス定義

J2EE アプリケーションを構成する，Enterprise Bean および Web アプリケーション（サーブレットおよび JSP）がデータソースを呼び出す場合，データソースのリファレンスを解決するためのプロパティを設定します。

(1)　Enterprise Bean のリファレンス定義

J2EE アプリケーションを構成する Enterprise Bean が，データソースを参照している場合，リファレンスを解決するためのプロパティを設定します。

(a)　編集する属性ファイル

J2EE アプリケーションを構成する Enterprise Bean の種類に対応する属性ファイルを編集します。

- Session Bean 属性ファイル
- Entity Bean 属性ファイル
- Message-driven Bean 属性ファイル

(b)　編集する属性ファイルの取得と属性の設定

● 属性ファイルの取得

次に示すコマンドを実行して Enterprise Bean の属性ファイルを取得します。

実行形式

```
cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -
resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Beanの属性ファイルパス>
```

実行例

```
cjgetappprop MyServer -name adder -type ejb -resname addr/cadder_eb -c C:¥home
¥adder_ejb.xml
```

● 属性の設定

次に示すコマンドを実行して，Enterprise Bean の属性ファイルの値を反映します。

実行形式

```
cjsetappprop［<サーバ名称>］［-nameserver <プロバイダ URL >］ -name < J2EE アプリケーション名> -type ejb
-resname < EJB-JAR 表示名>/< Enterprise Bean 表示名> -c < Enterprise Bean の属性ファイルパス>
```

実行例

```
cjsetappprop MyServer -name adder -type ejb -resname addr/adder_eb -c C:¥home¥adder_ejb.xml
```

(c)　編集する属性設定項目

データソースのリファレンス設定項目（<resource-ref>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
リソース参照名	○	<res-ref-name>
リソースタイプ	○	<res-type>

項目	必須	対応するタグ名
リソース認証方式	○	<res-auth>
リソース共有	△	<res-sharing-scope>
リンク先のデータソース名	△	<linked-to>

（凡例）　○：必須　△：任意

プロパティの設定項目については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.6.1　MessageDrivenBean 属性ファイルの指定内容」

(2)　サーブレットと JSP のリファレンス定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）がデータソースを呼び出す場合，データソースのリファレンスを解決するためのプロパティを設定します。

(a)　編集する属性ファイル

WAR 属性ファイル

(b)　編集する属性ファイルの取得と属性の設定

● 属性ファイルの取得

次に示すコマンドを実行して WAR 属性ファイルを取得します。

実行形式

```
cjgetappprop ［<サーバ名称>］ ［-nameserver ＜プロバイダURL>］ -name ＜J2EEアプリケーション名> -type war -resname ＜WAR表示名> -c ＜WAR属性ファイルパス>
```

実行例

```
cjgetappprop MyServer -name adder -type war -resname adder -c C:¥home¥adder_war.xml
```

● 属性の設定

次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

実行形式

```
cjsetappprop［<サーバ名称>］［-nameserver ＜プロバイダ URL ＞］-name ＜ J2EE アプリケーション名> -type war -resname ＜ WAR 表示名> -c ＜ WAR 属性ファイルパス>
```

実行例

```
cjsetappprop MyServer -name adder -type war -resname adder -c C:¥home¥adder_war.xml
```

(c)　編集する属性設定項目

Enterprise Bean が，データソースを参照している場合のプロパティ設定項目と同じです。「(1) Enterprise Bean のリファレンス定義」の「(c)　編集する属性設定項目」を参照してください。

2.12　ベーシックモードでデータベースに接続する場合の運用管理（互換用機能）

ここでは，互換用の動作モードであるベーシックモード（1.3basic モード）で，データベースに接続する場合の運用，管理について説明します。

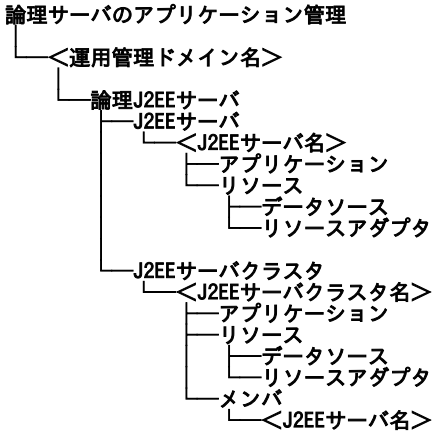
ベーシックモードでデータベースに接続する場合は，**データソース**を使用します。Management Server では，サーバ管理コマンドで設定したデータソース（JAR ファイル）の，運用環境へのインポート，削除および稼働状況の監視が行えます。

2.12.1　データソース管理

ここでは，運用管理ドメイン内の論理 J2EE サーバへのデータソースのインポート，およびインポートしたデータソースの削除などの操作に関する画面の機能概要，表示手順，操作手順，画面詳細について説明します。

- ［データソースのインポート］画面
  開発環境で作成したデータソースを論理 J2EE サーバで実行する場合に操作します。
- ［データソースの削除］画面
  運用で必要と判断される場合に操作します。
- ［データソースのログの表示］画面
  運用で必要と判断される場合に操作します。

これらの画面は，「論理サーバのアプリケーション管理」のデータソースノードで操作できます。データソースノードは，「論理サーバのアプリケーション管理」の動作モードとして V7 互換モードを選択した場合に表示されます。V8 モードの場合は表示されません。「論理サーバのアプリケーション管理」のツリーペインの構成を次に示します。



データソースノードがどのツリー下にあるかによって，操作対象となるデータソースが異なります。

- J2EE サーバのツリー下のデータソースノード
  J2EE サーバで使用するデータソースです。選択した J2EE サーバのデータソースを対象にした次の操作ができます。
  - データソースのインポート（(1)参照）
  - データソースの削除（(2)参照）

- ログの表示（(3)参照）

- J2EE サーバクラスタのツリー下のデータソースノード

  J2EE サーバクラスタを構成する J2EE サーバで使用するデータソースです。選択した J2EE サーバクラスタのデータソースを対象にした次の操作ができます。

  - データソースのインポート（(1)参照）
  - データソースの削除（(2)参照）
  - ログの表示（(3)参照）

  J2EE サーバクラスタの場合，クラスタを構成しているすべての J2EE サーバに同様の操作を実行します。J2EE サーバクラスタに J2EE サーバが登録されていない場合にはエラー画面が表示されます。

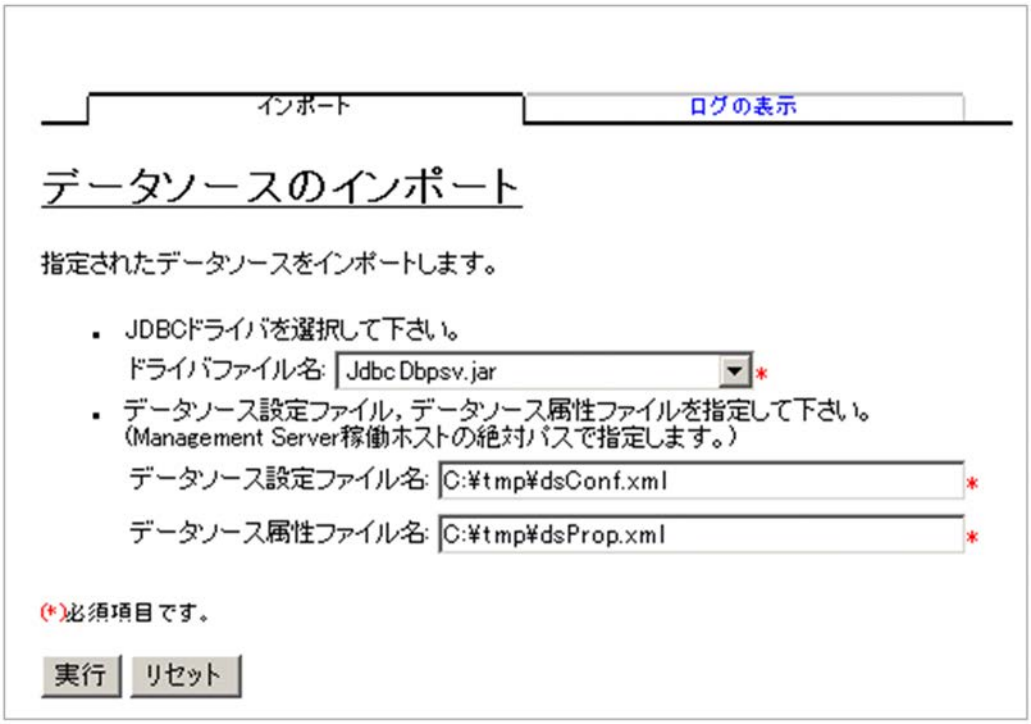
操作の排他制御について

データソースに対して，同時に複数の操作を実行できません。このため，ほかの操作が実行中であることを示すメッセージが表示される場合があります。この場合，ほかの操作が終了したことを確認してから，再度操作してください。

(1)　データソースのインポート

［データソースのインポート］画面を次の図に示します。

図 2-4　［データソースのインポート］画面



(a)　機能概要

論理 J2EE サーバにデータソースをインポートします。

インポートするデータソースがすでにある場合，インポート操作の実行結果は「失敗」と表示され，上書きはされません。データソース名（データソースの表示名）に使用できる文字は，半角英数字（0～9，A～Z，a～z）およびアンダースコア(_)です。

(b)　表示手順

画面の表示手順を次に示します。

1. 運用管理ポータルで［論理サーバのアプリケーション管理］アンカーをクリックします。
2. ツリーペインで次のどちらかの操作をします。

J2EE サーバの場合

［論理 J2EE サーバ］－［J2EE サーバ］－［＜ J2EE サーバ名＞］－［リソース］－［データソース］をクリックします。

J2EE サーバクラスタの場合

［論理 J2EE サーバ］－［J2EE サーバクラスタ］－［＜ J2EE サーバクラスタ名＞］－［リソース］－［データソース］をクリックします。

3. ［インポート］タブをクリックします。

(c)　操作手順

画面での操作手順を次に示します。

1. ［ドライバファイル名］，［データソース設定ファイル名］，［データソース属性ファイル名］を指定します。
2. ［実行］ボタンをクリックします。

指定内容に誤りがある場合，エラーが表示されます。［戻る］アンカーをクリックして原因を取り除き，再度［実行］ボタンをクリックします。

指定内容が正しい場合，結果確認画面の［ステータス］に「実行中」または「実行待ち」と表示されます。処理が終了すると，［ステータス］が「成功」または「失敗」に更新されます。

3. 次のどちらかの方法で，画面の表示内容を更新します。

- ［最新の情報に更新］アンカーをクリックする
- ［更新時間間隔］から表示内容を自動更新する間隔を選択して，［適用］ボタンをクリックする

失敗した場合は，［ログの表示］タブをクリックしてエラーの内容を確認します。原因を取り除き，［インポート］タブをクリックして再度実行します。

(d)　画面詳細

画面に表示される項目およびボタンについて説明します。

ドライバファイル名（必須）

JDBC ドライバのクラスライブラリファイルを指定します。必ず「選択して下さい」以外を指定してください。

- 選択して下さい（デフォルト）
- JdbcDbpsv.jar
- JdbcDbpsv.jar+JdbcDbpsvEX.jar

データソース設定ファイル名（必須）

Management Server 稼働ホストに配置したデータソース設定ファイルを絶対パスで指定します。デフォルトでは，何も表示されません。

データソース属性ファイル名（必須）

Management Server 稼働ホストに配置したデータソース属性ファイルを絶対パスで指定します。デフォルトでは，何も表示されません。

［実行］ボタン

指定値をチェックし，データソースをインポートします。

［リセット］ボタン

指定した内容をリセットします。

(2)　データソースの削除

［データソースの削除］画面を次の図に示します。

図 2-5　［データソースの削除］画面

(a)　機能概要

論理 J2EE サーバからデータソースを削除します。

指定したデータソースがない場合，削除操作の実行結果は「失敗」と表示されます。また，削除しようとしたデータソースが稼働中の J2EE アプリケーションから使用されている場合，データソースの削除に失敗します。

(b)　表示手順

画面の表示手順を次に示します。

1. 運用管理ポータルで［論理サーバのアプリケーション管理］アンカーをクリックします。
2. ツリーペインで次のどちらかの操作をします。

J2EE サーバの場合

［論理 J2EE サーバ］－［J2EE サーバ］－［< J2EE サーバ名>］－［リソース］－［データソース］をクリックします。

J2EE サーバクラスタの場合

［論理 J2EE サーバ］－［J2EE サーバクラスタ］－［< J2EE サーバクラスタ名>］－［リソース］－［データソース］をクリックします。

3. ［削除］タブをクリックします。

(c)　操作手順

画面での操作手順を次に示します。

1. 削除するデータソースの［削除］アンカーをクリックします。

データソースの削除確認画面に，データソースの詳細が表示されます。

図 2-6　削除確認画面（データソースの削除）

2. 内容を確認して，［はい］ボタンをクリックします。

結果確認画面が表示されます。画面の［ステータス］に「実行中」または「実行待ち」と表示され，処理が終了すると［ステータス］が「成功」または「失敗」に更新されます。

3. 次のどちらかの方法で，画面の表示内容を更新します。

- ［最新の情報に更新］アンカーをクリックする
- ［更新時間間隔］から表示内容を自動更新する間隔を選択して，［適用］ボタンをクリックする

失敗した場合は，［ログの表示］タブをクリックしてエラーの内容を確認します。原因を取り除き，［削除］タブをクリックして再度実行します。

(d)　画面詳細

画面に表示される項目およびボタンについて説明します。

●　［データソースの削除］画面

操作

［削除］アンカー

論理 J2EE サーバから削除するデータソースを選択するための削除確認画面が表示されます。

リソース名

該当論理 J2EE サーバにインポートされているデータソースの表示名が表示されます。

インポート時刻

論理 J2EE サーバへのインポートが完了した時刻（Management Server 稼働ホストの時刻）が表示されます。

●　削除確認画面

リソース名

削除するデータソース名が表示されます。

インポート時刻

削除するデータソースを論理 J2EE サーバにインポートした時刻（Management Server 稼働ホストの時刻）が表示されます。

［はい］ボタン

データソースを論理 J2EE サーバから削除します。結果確認画面が表示されます。

［いいえ］ボタン

［データソースの削除］画面に戻ります。

(3)　ログの表示（論理サーバのアプリケーション管理）

(a)　機能概要

運用管理ドメイン内のすべての論理 J2EE サーバで出力された，J2EE アプリケーション，データソース，およびリソースアダプタにかかわる実行結果を，ログとして一覧で参照できます。

ログは［論理サーバのアプリケーション管理］内の［ログの表示］タブから表示できますが，どこから表示しても内容は同一です。

(b)　表示手順

画面の表示手順を次に示します。

1. 運用管理ポータルで［論理サーバのアプリケーション管理］アンカーをクリックします。
2. ツリーペインで次のどちらかの操作をします。

J2EE サーバの場合

［論理 J2EE サーバ］－［J2EE サーバ］－［＜J2EE サーバ名＞］－［リソース］－［データソース］をクリックします。

J2EE サーバクラスタの場合

［論理 J2EE サーバ］－［J2EE サーバクラスタ］－［＜J2EE サーバクラスタ名＞］－［リソース］－［データソース］をクリックします。

3. ［ログの表示］タブをクリックします。

(c)　操作手順

画面での操作はありません。

(d)　画面詳細

画面には，データソースへのインポートや削除の処理結果が表示されます。これらのログから，処理実行時に発生したエラーの原因を知ることができます。

［ログの表示］画面に出力されるログについては，マニュアル「アプリケーションサーバ 運用管理ポータル 操作ガイド」の「7.4.2　ログの出力形式」を参照してください。

2.12.2　データソースの稼働情報監視

ここでは，J2EE リソースのデータソース（SimpleJTA または FullJTA）の，稼働情報の監視方法および監視情報について説明します。稼働情報監視は，運用で必要と判断される場合に操作します。

データソースの稼働情報監視は，［論理サーバの運用監視］画面のデータソースノードで操作できます。［論理サーバの運用監視］画面のサーバビューでのデータソースノードの位置づけを次に示します。

```
└─<運用管理ドメイン名>
    │
    :
    │
    ├─論理 J2EE サーバ
    │   ├─J2EE サーバ
    │   │   └─< J2EE サーバ名>
    │   │       │
    │   │       :
    │   │       │
    │   │       ├─リソース
    │   │       │   ├─<データソース名>
    │   │       │   └─<リソースアダプタ名>
    │   │       :
    │   │
    │   └─J2EE サーバクラスタ
    │       └─< J2EE サーバクラスタ名>
    │           └─< J2EE サーバ名>
    │               └─以下，上記の< J2EE サーバ名>の構成と同じです。
    :
```

参考

ここでは，データソースノードについて説明しています。データソースノード以外の「論理サーバの運用監視」のサーバビューのノードについては，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「13.1.2 「論理サーバの運用監視」のサーバビューの構成」を参照してください。

サーバビューに表示されるデータソースノードでは，選択した J2EE サーバの J2EE リソースのデータソース（SimpleJTA または FullJTA）を対象にした稼働情報の監視ができます。それぞれのデータソースで監視できる情報には次の種類があります。

- データソース（基本情報）
- データソース（稼働情報）

データソース（SimpleJTA または FullJTA）で監視できる項目を一覧で示します。

表 2-37　データソース（SimpleJTA または FullJTA）で監視できる項目

監視の対象		監視できる項目
リソース	データソース（SimpleJTA）	**基本情報** • リソース名 • リソースタイプ • 説明 • 認証タイプ • ログインタイムアウト • ユーザ ID • コネクションプール **稼働情報**

監視の対象		監視できる項目
リソース	データソース（SimpleJTA）	• リソース名 • プール現在値（総数） • 使用中のコネクション数 • 未使用のコネクション数
	データソース（FullJTA）	**基本情報** • リソース名 • リソースタイプ • 説明 • ログインタイムアウト • ユーザ ID • コネクションプール **稼働情報** • リソース名 • プール現在値 • 使用中のコネクション数 • 未使用のコネクション数 • getConnection()メソッドの実行時間 • getXAConnection()メソッドの実行時間 • getConnection()メソッドの失敗回数 • Connection で FATAL エラーが発生した回数

(1)　データソースの稼働情報監視（基本情報）

(a)　機能概要

J2EE リソースのデータソース（SimpleJTA または FullJTA）の稼働に関する初期設定値を表示できます。

(b)　表示手順

画面の表示手順を次に示します。

1. 運用管理ポータルで，［論理サーバの運用監視］アンカーをクリックします。
2. ［サーバビュー］タブ－［論理 J2EE サーバ］－［J2EE サーバ］－［< J2EE サーバ名>］－［リソース］－［<データソース名>］をクリックします。
3. ［基本情報］タブをクリックします。

(c)　操作手順

画面での操作はありません。

(d)　画面詳細

画面に表示される項目について説明します。

表示される項目は，リソースタイプが SimpleJTA か FullJTA かで異なります。

データソースが SimpleJTA の場合

DataSource

- リソース名
- リソースタイプ（DataSource と表示される）
- 説明
- 認証タイプ
- ログインタイムアウト（単位：秒）
- ユーザ ID
- コネクションプール
  最大値
  最小値

データソースが FullJTA の場合

DataSource

- リソース名
- リソースタイプ（DataSourceFullJTA と表示される）
- 説明
- ログインタイムアウト（単位：秒）
- ユーザ ID
- コネクションプール
  最大値
  最小値

(2)　データソースの稼働情報監視（稼働情報）

(a)　機能概要

J2EE リソースのデータソース（SimpleJTA または FullJTA）の稼働中の統計情報を監視できます。

(b)　表示手順

画面の表示手順を次に示します。

1. 運用管理ポータルで，［論理サーバの運用監視］アンカーをクリックします。
2. ［サーバビュー］タブ－［論理 J2EE サーバ］－［J2EE サーバ］－［< J2EE サーバ名>］－［リソース］－［<データソース名>］をクリックします。
3. ［稼働情報］タブをクリックします。

(c)　操作手順

画面での操作はありません。

(d)　画面詳細

画面に表示される項目について説明します。

表示される項目は，リソースタイプが SimpleJTA か FullJTA かで異なります。

リソースタイプが SimpleJTA の場合

DataSource

- リソース名
- プール現在値（総数）
- 使用中のコネクション数
- 未使用のコネクション数

リソースタイプが FullJTA の場合

DataSource

- リソース名
- プール現在値（総数）
- 使用中のコネクション数
- 未使用のコネクション数
- getConnection()メソッドの実行時間
  N 秒平均値（単位：ミリ秒）
  最大値（単位：ミリ秒）
  最小値（単位：ミリ秒）
- getXAConnection()メソッドの実行時間
  N 秒平均値（単位：ミリ秒）
  最大値（単位：ミリ秒）
  最小値（単位：ミリ秒）
- getConnection()メソッドの失敗回数
- Connection で FATAL エラーが発生した回数

注

「N 秒」とは［サンプリング時間］で設定した時間を表します。

［最新の情報に更新］アンカー

画面を最新の情報に更新します。

画面は時間の経過とともに変わります。常に最新の状態でご参照ください。

2.13　J2EE サーバで使用するリソース操作コマンド

ここでは，J2EE サーバで使用するリソース操作コマンドについて説明します。

J2EE サーバで使用するリソース操作コマンドの一覧を，次の表に示します。

表 2-38　J2EE サーバで使用するリソース操作コマンドの一覧

コマンド名称	分類	概要	コマンドの系統
cjclosecn	J2EE サーバのコネクションクローズまたはコネクションの状態表示	データソースのコネクションをクローズします。またはコネクションの状態を表示します。	更新※

（凡例）

更新：更新系コマンド。J2EE サーバの内容を更新，または構成情報を取得するコマンドです。

注※

リスト表示の場合は，参照になります。

cjclosecn（J2EE サーバのコネクションクローズまたはコネクションの状態表示）

形式

JDBC コネクション一覧を表示する場合

指定されたデータソースを対象とする場合

```
cjclosecn [<サーバ名称>] [-nameserver <プロバイダURL>]
          -l [-verbose] -resname <データソース表示名>
          [-resname <データソース表示名> ...]
```

プールに登録されたすべてのデータソースを対象とする場合

```
cjclosecn [<サーバ名称>] [-nameserver <プロバイダURL>]
          -l [-verbose] -resall
```

タイムアウト時間を指定してコネクションをクローズする場合

指定されたデータソースを対象とする場合

```
cjclosecn [<サーバ名称>] [-nameserver <プロバイダURL>]
          -t <seconds> -resname <データソース表示名>
          [-resname <データソース表示名> ...]
```

プールに登録されたすべてのデータソースを対象とする場合

```
cjclosecn [<サーバ名称>] [-nameserver <プロバイダURL>]
          -t <seconds> -resall
```

コネクション ID を指定してコネクションをクローズする場合

```
cjclosecn [<サーバ名称>] [-nameserver <プロバイダURL>]
          -resname <データソースの表示名>
          -cid <コネクションID> [-cid <コネクションID> ...]
```

機能

指定されたデータソースまたはプールに登録されたすべてのデータソースを対象として，使用中のコネクションの中で，取得時からの経過時間が指定時間を超過しているものを強制的にクローズします。このコマンドでは，次の機能を使用できます。

JDBC コネクション一覧を表示する

指定されたデータソースまたはプールに登録されたすべてのデータソースを対象として，JDBC コネクション一覧を表示します。

-verbose を指定しない場合

使用中の JDBC コネクション一覧を表示します。データソースを指定する場合，一度に複数のデータソースを指定できます。

-verbose を指定した場合

使用中および未使用のすべての JDBC コネクション一覧を表示します。

タイムアウト時間を指定してコネクションをクローズする

指定されたデータソースまたはプールに登録されたすべてのデータソースを対象として，使用中のコネクションの中で，取得時からの経過時間が指定時間を超過しているものを強制的にクローズします。データソースを指定する場合，一度に複数のデータソースを指定できます。

コネクション ID を指定してコネクションをクローズする

指定されたデータソースを対象として，JDBC コネクションの中で，コネクション ID を指定されたコネクションを強制的にクローズします。

引数

<サーバ名称>

接続先 J2EE サーバ名称を指定します。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

-nameserver <プロバイダ URL>

CORBA ネーミングサービスへのアクセスプロトコル，CORBA ネーミングサービスが稼働しているホスト名，およびそれが使用しているポート番号を，次に示す形式で指定します。

<プロトコル名称>::<ホスト名称>:<ポート番号>

指定内容の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.1 J2EE サーバで使用するコマンドの概要」を参照してください。

-l

使用中のコネクション情報の一覧を表示します。

-t <seconds>

コネクションを解放する際のタイムアウト時間を指定します。指定時間を超過したコネクションが解放対象となります。指定は秒単位で，指定できる値は 0≦指定値≦2147483647 の範囲になります。

-verbose

使用中のコネクションおよび使用中でないコネクションを表示対象とします。

-resname <データソース表示名>

クローズ対象となるデータソース名称を指定します。一度に複数のデータソースを指定できます。

-resall

すべてのデータソースの使用中コネクションが対象となります。

-resname と同時に指定できません。

-cid <コネクション ID>

-l オプションで表示した一覧情報の中から，-resname で指定したリソースが取得したコネクションを ID 番号で指定します。コネクション ID は，16 進数で指定します。ID の指定は 8 けたまで指定できます。

入力例

JDBC コネクション一覧を表示する場合（指定されたデータソースを対象とする場合）

```
cjclosecn MyServer -l -resname JdbcDbpsv
```

タイムアウト時間を指定してコネクションをクローズする場合（指定されたデータソースを対象とする場合）

```
cjclosecn MyServer -t 200 -resname JdbcDbpsv -resname JdbcDbpsv2
```

コネクション ID を指定してコネクションをクローズする場合

```
cjclosecn MyServer -resname JdbcDbpsv -cid 657ed
```

戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

3：

タイムアウトが発生しました。

9：

管理者特権がないため，コマンドが実行できません（Windows Server 2012，Windows Server 2008，Windows 8，Windows 7 または Windows Vista の場合）

注意事項

- このコマンドの機能は，ベーシックモードで実行する場合だけ有効となります。1.4 モードで実行すると，サーバエラーとなります。
- cjclearpool コマンドでプールから削除したコネクションは，このコマンドの対象にはなりません。
- コマンド実行時，未決着の JDBC トランザクションはタイムアウトになります。また，未決着の JDBC トランザクションはロールバックします。
- コマンドの引数にサーバ名称を指定する場合，コマンド名の直後にする必要があります。そのほかの引数は，サーバ名称より後ろ（サーバ名称を省略した場合はコマンド名の後ろ）であれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：-resname <データソース表示名>）やオプション名と値を非対応にすること（例：-cid <データソース表示名> -resname <コネクション ID>）はできません。
- コマンドの引数にサーバ名称を指定する場合，cjsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。
- 一度に複数のデータソースを指定する場合，存在しないデータソースがあったときでも，すべてのデータソースに対して，JDBC コネクション一覧の表示またはコネクションのクローズを試みます。
- コネクション ID を指定する場合，未使用のコネクションを指定したときはエラーとなります。

2.14　ベーシックモード使用時の注意事項

この節では，ベーシックモードを使用するときの共通の注意事項を示します。

2.14.1　サーブレットおよび JSP 実装時共通の注意事項

アプリケーションサーバ上で動作するアプリケーションのプログラムとして，サーブレットおよび JSP を実装するときの共通の注意事項を示します。

(1)　ユーザスレッドの使用方法

アプリケーションを構成するサーブレットおよび JSP からスレッドを生成して，使用できます。ユーザがプログラムの中で明示して生成するスレッドのことを，**ユーザスレッド**といいます。

ユーザスレッドは，生成後の動作のしかた（ライフサイクル）によって，次の二つに分けられます。

- サービスメソッドや init メソッドの範囲内で動作させる。
- サービスメソッドや init メソッドのバックグラウンドで動作させる。

ユーザスレッドの使用条件

- ユーザスレッドは，Enterprise Bean では使用できません（EJB 仕様で，Enterprise Bean からのスレッドの生成が禁止されているため）。
- ユーザスレッドが使用できるアプリケーションサーバのサーバモードを次に示します。

表 2–39　ユーザスレッドの前提条件（サーバモード）

サーバモード		使用可否
J2EE サーバモード	1.4 モード	○
	ベーシックモード	△
サーブレットエンジンモード		△

（凡例）

○：使用できる

△：サーバモード（ベーシックモードまたはサーブレットエンジンモード）でサポートされている範囲で機能を使用できます。ただし，コネクションプールは使用できません。

2.14.2　Stateful Session Bean 実装時の注意事項

ベーシックモードで，Stateful Session Bean を実装するときの注意事項を示します。

(1)　SessionSynchronization のメソッド呼び出しについての注意

クライアントでトランザクション開始後に SessionSynchronization の Stateful Session Bean（トランザクション属性は Required または Mandatory）を呼び出して，afterBegin メソッドまたは setRollbackOnly メソッドを呼び出すと，クライアントでのコミット時に beforeCompletion メソッドが呼び出されません。1.4 モードのときは，beforeCompletion メソッドが呼び出されます。

(2)　beforeCompletion メソッドからの Enterprise Bean の呼び出しについて

Stateful Session Bean の beforeCompletion メソッドから，ほかの Enterprise Bean は呼び出せません。ほかの Enterprise Bean を呼び出した場合，java.rmi.RemoteException 例外が発生します。

(3)　afterCompletion メソッドからの Enterprise Bean の呼び出しについて

Stateful Session Bean の afterCompletion メソッドからほかの Enterprise Bean を呼び出した場合，java.rmi.RemoteException 例外が発生します。

EJB の仕様上，Stateful Session Bean の afterCompletion メソッドからほかの Enterprise Bean を呼び出せないので，呼び出さないようにしてください。

3 サーブレットエンジンモード

旧バージョンとの互換用機能としてのサーバの動作モードには，ベーシックモードとサーブレットエンジンモードがあります。この章では, サーブレットエンジンモードについて説明します。

3.1　この章の構成

1.4 モードまたはベーシックモードでは J2EE コンテナ（EJB コンテナと Web コンテナ）を利用します。一方，サーブレットエンジンモードは，Web アプリケーションだけを動作させる（Enterprise Bean を使用しない）場合など，Web コンテナ部分だけを使用する構成のときに利用できる動作モードです。サーブレットエンジンモードでは，サーブレットエンジンだけが単独で動作します。サーブレットエンジンとは，サーブレット実行機能，JSP 実行機能を持つサーバのことです。

このモードでは，Web アプリケーションからは Java Servlet Specification v2.3，JavaServer Pages Specification v1.2 に示す API および DABroker Library を通じ，Sun JDBC API 仕様に従った方法によるデータベースアクセスが利用できます。

なお，サーブレットエンジンモードで実行したときのサーバを **Web コンテナサーバ**と呼びます。

この章の構成を次の表に示します。

表 3-1　この章の構成（サーブレットエンジンモード）

分類	タイトル	参照先
解説	使用できる機能の一覧	3.2
	サーブレットエンジンモードと J2EE サーバモードの比較	3.3
	サーブレットエンジンモードで使用できる Web コンテナの機能	3.4
実装	サーブレットエンジンモードで動作するアプリケーションの実装	3.5
設定	サーブレットエンジンモードでのリソース接続	3.6
	サーブレットエンジンモードの利用方法	3.7
運用	サーブレットエンジンモードでのシステム運用	3.8
	サーブレットエンジンモードの場合の移行	3.9
	Web コンテナサーバで使用するコマンド	3.10
	Web コンテナサーバで使用するファイル	3.11

注　「注意事項」について，この章での説明はありません。

3.2　使用できる機能の一覧

サーブレットエンジンモードで使用できる Java EE および J2EE の機能, ならびにアプリケーションサーバの機能について一覧に示します。

アプリケーションサーバで使用できる機能については, マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の「3.2.5(1)　アプリケーションサーバで使用できる機能」を参照してください。

3.2.1　使用できる Java EE および J2EE の機能

サーブレットエンジンモードで使用できる Java EE および J2EE の機能について，次の表に示します。なお，次の表に記載していない機能は使用できません。

表 3-2　サーブレットエンジンモードで使用できる Java EE および J2EE の機能

Java EE および J2EE の機能	サーブレットエンジンモード
Servlet 2.2	使用できます。
Servlet 2.3	
JSP 1.2	
JDBC 2.0 コア JDBC 2.0 オプションパッケージ	

3.2.2　使用できるアプリケーションサーバの機能

サーブレットエンジンモードでの，アプリケーションサーバの各機能の使用について説明します。

なお，各項目にある表に記載していない機能は使用できません。

(1)　Web コンテナの機能

Web コンテナの機能のサーブレットエンジンモードでの使用について次の表に示します。Web コンテナの機能については,「3.4　サーブレットエンジンモードで使用できる Web コンテナの機能」を参照してください。

表 3-3　Web コンテナの機能のサーブレットエンジンモードでの使用

Web コンテナの機能	サーブレットエンジンモードでの使用
Web アプリケーションの実行機能	△
JSP の事前コンパイルとコンパイル結果の保持 (JSP 事前コンパイルを使用しない場合の JSP コンパイル結果)	◎△
セッション管理機能（セッション ID および Cookie へのサーバ ID の付加）	◎
アプリケーションのイベントリスナ	○
リクエストおよびレスポンスのフィルタリング	○
HTTP レスポンス圧縮機能	○

Web コンテナの機能		サーブレットエンジンモードでの使用
EJB コンテナとの連携		◎△
データベースとの接続		◎△
Web コンテナによるスレッドの作成		◎△
ユーザスレッドの使用		△※
同時実行スレッド数の制御（Web コンテナ単位での同時実行スレッド数の制御）		◎
エラーページのカスタマイズ		◎
静的コンテンツのキャッシュ		○
簡易 Web サーバ機能	簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能）	○
	簡易 Web サーバへのアクセスの制御	◎△
URI のデコード機能		◎

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※

ユーザスレッドで使用できる機能のうち，サーブレットエンジンモードで使用できる機能だけが使用できます。

(2)　Web サーバ連携の機能

Web サーバ連携の機能のサーブレットエンジンモードでの使用について次の表に示します。Web サーバ連携の機能については，「3.4　サーブレットエンジンモードで使用できる Web コンテナの機能」を参照してください。

表 3-4　Web サーバ連携の機能のサーブレットエンジンモードでの使用

Web サーバ連携の機能	サーブレットエンジンモードでの使用
Web サーバ（リダイレクタ）によるリクエストの振り分け	◎
通信タイムアウト（Web サーバ連携）	◎
IP アドレスの指定（Web サーバ連携）	○
エラーページのカスタマイズ（Web サーバ連携）	◎
ドメイン名指定でのトップページの表示	○
Web コンテナへのゲートウェイ情報の通知	○

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(3)　インプロセス HTTP サーバの機能

インプロセス HTTP サーバの機能は，サーブレットエンジンモードでは使用できません。

(4)　EJB コンテナの機能

EJB コンテナの機能は，サーブレットエンジンモードでは使用できません。

(5)　EJB クライアントの機能

EJB クライアントの機能は，サーブレットエンジンモードでは使用できません。

(6)　ネーミング管理の機能

ネーミング管理の機能のサーブレットエンジンモードでの使用について次の表に示します。ネーミング管理の機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.　ネーミング管理」を参照してください。

表 3-5　ネーミング管理の機能のサーブレットエンジンモードでの使用

ネーミング管理の機能	サーブレットエンジンモードでの使用
ラウンドロビンポリシーによる CORBA ネーミングサービスの検索	◎
ネーミング管理機能でのキャッシング	◎
CORBA ネーミングサービスの切り替え	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(7)　リソース接続とトランザクション管理の機能

リソース接続とトランザクション管理の機能のサーブレットエンジンモードでの使用について次の表に示します。リソース接続とトランザクション管理の機能については,「3.6　サーブレットエンジンモードでのリソース接続」を参照してください。

表 3-6　リソース接続とトランザクション管理の機能のサーブレットエンジンモードでの使用

リソース接続とトランザクション管理の機能	サーブレットエンジンモードでの使用
コネクションプーリング	◎※1
コネクションの取得リトライ	◎※1
コネクションプールの情報表示	◎※2
コネクションプールのクリア	◎※2
コネクションの強制クローズ	◎※3

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

注※1　Web コンテナコネクションプーリング機能を利用する場合に使用できます。詳細については,「3.6.3　Web コンテナコネクションプール機能によるデータベース接続」を参照してください。

注※2　J2EE サーバモードの場合と実行するコマンドが異なります。

注※3　J2EE サーバモードの場合と実行するコマンドが異なります。コネクションの強制クローズについては,「3.6.3 Web コンテナコネクションプール機能によるデータベース接続」を参照してください。

(8)　OpenTP1 からのアプリケーションサーバを呼び出す機能（TP1 インバウンド連携機能）

TP1 インバウンド連携機能は，サーブレットエンジンモードでは使用できません。

(9)　アプリケーションサーバで使用する JPA の機能

アプリケーションサーバで使用する JPA の機能は，サーブレットエンジンモードでは使用できません。

(10)　CJPA プロバイダの機能

CJPA プロバイダの機能は，サーブレットエンジンモードでは使用できません。

(11)　CJMS プロバイダの機能

CJMS プロバイダの機能は，サーブレットエンジンモードでは使用できません。

(12)　セキュリティ管理の機能

セキュリティ管理の機能のサーブレットエンジンモードでの使用について次の表に示します。セキュリティ管理の機能については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を参照してください。

表 3-7　セキュリティ管理の機能のサーブレットエンジンモードでの使用

セキュリティ管理の機能	サーブレットエンジンモードでの使用
SSL 使用による認証情報とデータの暗号化	◎
ロールに基づく Web コンテナのユーザ認証	◎△
SecurityManager による Web コンテナの実行時の保護	◎△

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

(13)　アノテーションの機能

アノテーションの機能は，サーブレットエンジンモードでは使用できません。

(14)　J2EE アプリケーションの形式とデプロイの機能

J2EE アプリケーションの形式とデプロイの機能のサーブレットエンジンモードでの使用について次の表に示します。J2EE アプリケーションの形式とデプロイの機能については,マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.　J2EE アプリケーションの形式とデプロイ」を参照してください。

表 3-8　J2EE アプリケーションの形式とデプロイの機能のサーブレットエンジンモードでの使用

J2EE アプリケーションの形式とデプロイの機能	サーブレットエンジンモードでの使用
J2EE アプリケーションの入れ替え	◎△※1
Web アプリケーションの更新検知とリロード	◎△※2
リソースアダプタを含む J2EE アプリケーションの操作	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※1　機能差については，「(21)　J2EE アプリケーションの運用機能」を参照してください。

注※2　機能差については，「3.4.2　Web アプリケーションの更新検知とリロード」を参照してください。

(15)　コンテナ拡張ライブラリの機能

! 注意事項

- コンテナ拡張ライブラリのアクセス権について

  コンテナ拡張ライブラリにはアクセス権が付与されません。

  cjstartweb コマンドに-security オプションを指定し，Web コンテナの実行時保護機能を有効にする場合は，SecurityManager 定義ファイル（web.policy）に適切なアクセス権を設定してください。

- ハンドラ関数の設定について（Windows の場合）

  Windows でコンテナ拡張ライブラリを使用してプロセスのハンドラ関数を設定する場合，CTRL+BREAK 信号を処理するハンドラ関数では，TRUE を返したり，DLL の終了処理をしたり，ExitProcess 関数などを呼び出してプロセスを終了したりしないでください。

  Web コンテナサーバ，Component Container を利用するほかのプログラムなどが動作しなくなるおそれがあります。

コンテナ拡張ライブラリの機能のサーブレットエンジンモードでの使用について次の表に示します。コンテナ拡張ライブラリの機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.　コンテナ拡張ライブラリ」を参照してください。

表 3-9　コンテナ拡張ライブラリの機能のサーブレットエンジンモードでの使用

コンテナ拡張ライブラリの機能	サーブレットエンジンモードでの使用
コンテナ拡張ライブラリ	◎
サーバ起動・停止フック機能	◎
スマートエージェント経由での CORBA オブジェクトの呼び出し	◎

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

(16)　バッチアプリケーション実行時に使用する機能

バッチアプリケーション実行時に使用する機能は，サーブレットエンジンモードでは使用できません。

(17)　CTM による Enterprise Bean のスケジューリング機能

CTM による Enterprise Bean のスケジューリング機能は，サーブレットエンジンモードでは使用できません。

(18)　そのほかの拡張機能

そのほかの拡張機能のサーブレットエンジンモードでの使用について次の表に示します。そのほかの拡張機能については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「1.2.3　そのほかの拡張機能」を参照してください。

表 3-10　そのほかの拡張機能のサーブレットエンジンモードでの使用

そのほかの拡張機能	サーブレットエンジンモードでの使用
統合ユーザ管理	◎△※1
ユーザログ出力	◎※2

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※1　リポジトリ管理およびリソース監視は使用できません。

注※2　Web コンテナサーバの usrconf.properties で，接尾辞が「.level」で終わるキーのうち，値の範囲に SEVERE，WARNING，INFO，CONFIG，FINE，および FINEST 以外を持つキーが設定された場合は，次の現象が発生します。

1.　サーバ起動時，java.util.logging.LogManager クラスがキーを読み込むときに接尾辞が「.level」で終わるキーの値をチェックします。値が範囲以外の場合は，java.util.logging.LogManager クラスが標準エラー出力へエラーメッセージを出力します。

（例）sample.level=Error と指定されていた場合

「Bad level value for property : sample.level」と出力されます。

2.　ユーザログ機能の接尾辞が「.level」で終わるキーでは値が適切でない場合，1.と同様にエラーメッセージを出力します。

ただし，どちらの場合もメッセージが表示されるだけで，動作上は影響ありません。

(19)　システムの日常運用を支援する機能

システムの日常運用を支援する機能のサーブレットエンジンモードでの使用について次の表に示します。システムの日常運用を支援する機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.1　システムの日常運用を支援する機能」を参照してください。

表 3-11　システムの日常運用を支援する機能のサーブレットエンジンモードでの使用

システムの日常運用を支援する機能	サーブレットエンジンモードでの使用
システムの起動・停止	◎△※

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※　システムの起動・停止については，「3.3.7　システムの起動および停止方法」を参照してください。

(20)　システムの保守を支援する機能

システムの保守を支援する機能は，サーブレットエンジンモードで使用できません。

(21)　J2EE アプリケーションの運用機能

J2EE アプリケーションの運用機能のサーブレットエンジンモードでの使用について次の表に示します。J2EE アプリケーションの運用機能については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「1.2.3　J2EE アプリケーションの運用機能」を参照してください。

表 3-12　J2EE アプリケーションの運用機能のサーブレットエンジンモードでの使用

J2EE アプリケーションの運用機能		サーブレットエンジンモードでの使用
J2EE アプリケーションのサービスの閉塞		◎
J2EE アプリケーションの停止（通常停止）		○
J2EE アプリケーションの入れ替え	Web アプリケーションのサービスの部分閉塞による入れ替え	○※1
	リロードによる J2EE アプリケーションの入れ替え	◎△※2

（凡例）

○：使用できる

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

△：使用できる。ただし，1.4 モードと機能差がある

注※1　Web サーバに HTTP Server を使用している場合だけ使用できます。

注※2　機能差については，「3.4.2　Web アプリケーションの更新検知とリロード」を参照してください。

(22)　システムの監査を支援する機能

システムの監査を支援する機能は，サーブレットエンジンモードでは使用できません。

(23)　JP1 連携による運用管理機能

JP1 連携による運用管理機能は，サーブレットエンジンモードでは使用できません。

参考

サーブレットエンジンモードで動作しているサーバ（Web コンテナサーバ）は Management Server の管理対象にはなりません。JP1 連携は，Management Server を利用した運用をしていることが前提となる機能です。このため，サーブレットエンジンモードで動作しているサーバ（Web コンテナサーバ）は JP1 連携による管理の対象にはなりません。なお，JP1 連携による運用管理機能と動作モードは，直接関係ありません。

(24)　クラスタソフトウェアとの連携による系切り替え機能

クラスタソフトウェアとの連携による系切り替え機能は，サーブレットエンジンモードでは使用できません。

(25)　システムの保守のための機能

システムの保守のための機能のサーブレットエンジンモードでの使用について次の表に示します。

表 3-13　システムの保守のための機能のサーブレットエンジンモードでの使用

システムの保守のための機能	サーブレットエンジンモードでの使用	参照先マニュアル	参照先
トラブルシューティング	◎※	アプリケーションサーバ 機能解説 保守／移行編	2 章，3 章，4 章，5 章
性能解析トレースを使用したシステムの性能解析	◎		7 章，8 章

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

注※　障害検知時コマンドは使用できません。

(26)　JavaVM の機能

JavaVM の機能のサーブレットエンジンモードでの使用について次の表に示します。

表 3-14　JavaVM の機能のサーブレットエンジンモードでの使用

JavaVM の機能	サーブレットエンジンモードでの使用	参照先マニュアル	参照先
クラス別統計機能	◎	アプリケーションサーバ 機能解説 保守／移行編	9.3
インスタンス統計機能	◎		9.4
STATIC メンバ統計機能	◎		9.5
参照関係情報出力機能	◎		9.6
統計前の GC 選択機能	◎		9.7
Survivor 領域の年齢分布情報出力機能	◎		9.11
JIT コンパイル時の C ヒープ確保量の上限値設定機能	◎		9.13
スレッド数の上限値設定機能	◎		9.14

（凡例）

◎：J2EE の機能に加えてアプリケーションサーバで拡張した機能を使用できる，またはアプリケーションサーバ独自の機能を使用できる

3.3　サーブレットエンジンモードと J2EE サーバモードの比較

サーブレットエンジンモードは互換用のサーバの動作モードです。旧バージョンでサーブレットエンジンモードを使用していた場合には，推奨の動作モードである J2EE サーバモードの 1.4 モード（以降，J2EE サーバモードと呼びます）への移行をお勧めします。ここでは，移行時の留意事項として，サーブレットエンジンモードと J2EE サーバモードを比較し，機能の差異およびシステムの構築方法の差異について説明します。

サーブレットエンジンモードから J2EE サーバモードへ移行する場合には，次の差異に留意する必要があります。

- システムの構築方法
- Web アプリケーションのデプロイと設定
- Web サーバの使用方法
- データベースとの接続
- Web アプリケーションに組み込むサーブレットフィルタ
- Web アプリケーションのリロード
- システムの起動および停止
- 使用するメモリ，スレッド数，ポート番号

3.3.1　システムの構築方法

サーブレットエンジンモードでは，Web コンテナサーバ用の usrconf.properties や usrconf.cfg を編集し，cjwebsetup などのコマンドを実行してシステムを構築しますが，J2EE サーバモードでは次に示すシステム構築方法があります。

- **セットアップウィザードを使用して構築する**

  セットアップウィザードを使用してシステムを構築します。システム構築の概要やシステム構築の詳細については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。

- **Smart Composer 機能を使用して構築する**

  Smart Composer 機能で提供するファイルおよびコマンドを使用して，システムを構築します。システム構築の概要やシステム構築の詳細については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。

- **運用管理ポータルを使用して構築する**

  運用管理ポータルに設定内容を入力して，システムを構築します。システム構築の概要や運用管理ポータルでのシステム構築方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」を参照してください。

どの方法でシステムを構築するかは，どのようなシステムを構築してどのように運用するかによって決定します。システム構築方法の検討については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。

参考

これらのほかに，ファイル編集とコマンド実行を使用してシステムを構築する方法もあります。この場合，各サーバ用の usrconf.properties や usrconf.cfg を編集し，cjsetup などのコマンドを実行してシステムを構築し

ます。ファイル編集とコマンド実行でのシステム構築方法については，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。

3.3.2　Web アプリケーションのデプロイと設定

サーブレットエンジンモードと J2EE サーバモードでは，Web アプリケーションのデプロイおよび属性の設定方法が異なります。

(1)　Web アプリケーションのデプロイ

サーブレットエンジンモードと J2EE サーバモードでは，Web アプリケーションのデプロイ方法が異なります。

- サーブレットエンジンモードの場合
  次のどちらかの方法で，Web アプリケーションを Web コンテナサーバにデプロイします。
  - WAR ファイルのコピー
  - ディレクトリのコピー

  また，WAR ファイル，またはディレクトリを削除することで，アンデプロイできます。
- J2EE サーバモードの場合
  Web アプリケーションを J2EE サーバにデプロイするためには，Web アプリケーションをアーカイブ形式，または展開ディレクトリ形式の J2EE アプリケーションにしてからデプロイします。
  - Web アプリケーションからアーカイブ形式の J2EE アプリケーションを作成し，J2EE サーバにインポートおよびデプロイします。
  - Web アプリケーションから展開ディレクトリ形式の J2EE アプリケーションを作成し，J2EE サーバにインポートおよびデプロイします。

  また，アーカイブ形式，または展開ディレクトリ形式の J2EE アプリケーションをアンデプロイします。

サーブレットエンジンモードと J2EE サーバモードの Web アプリケーションのデプロイおよびアンデプロイ方法の違いを次に示します。

項目	サーブレットエンジンモードの場合		J2EE サーバモードの場合	
	WAR ファイルのコピーの場合	ディレクトリのコピーの場合	アーカイブ形式の場合	展開ディレクトリ形式の場合
アセンブル	WAR 形式にアセンブルする	アセンブルは不要	EAR 形式または ZIP 形式にアセンブルする	アセンブルは不要
デプロイ／アンデプロイ単位	WAR ファイル単位	ディレクトリ単位	EAR ファイルまたは ZIP ファイル単位	ディレクトリ単位
入れ替えできる最小単位	WAR ファイル単位	JSP ファイルや Servlet クラス単位	EAR ファイルまたは ZIP ファイル単位	JSP ファイルや Servlet クラス単位
デプロイ／アンデプロイできるタイミング	Web コンテナサーバの起動中／停止中		J2EE サーバの起動中	
デプロイ／アンデプロイ結果の反映	Web コンテナサーバの起動／再起動後		デプロイ操作完了後	

サーブレットエンジンモードから J2EE サーバモードへ移行する場合には，サーブレットエンジンモードでの Web アプリケーションのデプロイ方法と操作性が近い，展開ディレクトリ形式によるデプロイ方法をお勧めします。

アーカイブ形式の J2EE アプリケーションについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.3　アーカイブ形式の J2EE アプリケーション」を参照してください。展開ディレクトリ形式の J2EE アプリケーションについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.4　展開ディレクトリ形式の J2EE アプリケーション」を参照してください。

(2)　Web アプリケーションの属性の設定

サーブレットエンジンモードの場合は，アプリケーション開発時に Web アプリケーションの DD（web.xml）を直接編集するか，または hitachi_web.properties のキーを設定して，Web アプリケーションの属性を設定します。なお，Web アプリケーションの DD（web.xml）と hitachi_web.properties は，Web アプリケーションの WEB-INF ディレクトリ下に置きます。

J2EE サーバモードの場合は，これらに加えて，次の手順で Web アプリケーションの属性を設定できます。

1. システム構築時に，サーバ管理コマンドで J2EE アプリケーションを J2EE サーバにインポートする。
2. 属性ファイルを編集して属性を設定する。

サーバ管理コマンドでは，J2EE アプリケーションの実行時間の監視や同時実行スレッド数の制御など，J2EE サーバの拡張機能の設定ができます。J2EE サーバモードの場合は，サーバ管理コマンドで設定する方法をお勧めします。サーバ管理コマンドの操作方法，および設定内容については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「3.　サーバ管理コマンドの基本操作」を参照してください。

なお，hitachi_web.properties の設定内容は，サーブレットエンジンモードと J2EE サーバモードで異なります。

- **サーブレットエンジンモードと J2EE サーバモードで同じ場合**

  JSP ファイルのコンパイル結果の保持（webserver.jsp.keepgenerated）と TLD ファイルの妥当性チェック（webserver.xml.validate）は，hitachi_web.properties または usrconf.properties で設定します。

- **サーブレットエンジンモードと J2EE サーバモードで異なる場合**

  サーブレットエンジンモードの場合，Web アプリケーションのリロードの設定は，hitachi_web.properties または usrconf.properties で設定します。

  J2EE サーバモードの場合，Web アプリケーションのリロードの設定は，usrconf.properties で設定します。J2EE サーバ用の hitachi_web.properties では，Web アプリケーションのリロードの設定はできません。

3.3.3　Web サーバの使用方法

サーブレットエンジンモードと J2EE サーバモードでは，Web サーバの使用方法が異なります。

サーブレットエンジンモードの場合	J2EE サーバモードの場合
Web サーバと連携して使用する	Web サーバと連携して使用する
Web コンテナサーバ内の簡易 Web サーバを使用する	J2EE サーバ内のインプロセス HTTP サーバを使用する

Web サーバと連携して使用する方法の場合，サーブレットエンジンモードから J2EE サーバモードに移行するときは，設定内容には差異はありません。ただし，システムの構築方法によって設定方法が異なります。

- Smart Composer 機能を使用してシステムを構築する場合，Smart Composer 機能で提供するファイルで Web サーバの設定をします。
- 運用管理ポータルを使用してシステムを構築する場合，運用管理ポータル上で Web サーバの設定をします。また，論理 Web サーバとして使用できるのは，HTTP Server です。
- ファイル編集とコマンド実行によってシステムを構築する場合，設定方法はサーブレットエンジンモードと同じです。設定の変更などは必要ありません。

簡易 Web サーバを使用する方法の場合, J2EE サーバモードのインプロセス HTTP サーバに移行するときは，設定内容および設定方法が異なります。

- Smart Composer 機能を使用してシステムを構築する場合，インプロセス HTTP サーバは使用できません。
- 運用管理ポータルを使用してシステムを構築する場合，運用管理ポータル上でインプロセス HTTP サーバの設定をします。
- ファイル編集とコマンド実行によってシステムを構築する場合，J2EE サーバ用の usrconf.properties にインプロセス HTTP サーバの設定をします。なお, 簡易 Web サーバを使用する方法で Web コンテナサーバ用の usrconf.properties の webserver.connector.http.で始まるキーに設定していた内容は, インプロセス HTTP サーバでは J2EE サーバ用の usrconf.properties の webserver.connector.inprocess_http.で始まるキーに設定します。

なお，インプロセス HTTP サーバでは，サーブレットエンジンモードの簡易 Web サーバよりも機能が多く, 同時実行スレッド数の制御などの機能も使用できます。インプロセス HTTP サーバについては, マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「5.　インプロセス HTTP サーバ」を参照してください。

3.3.4　データベースとの接続

Web アプリケーションから JDBC を使用してデータベースに接続をする場合, サーブレットエンジンモードと J2EE サーバモードでは Web アプリケーションの実装方法および設定方法が異なります。

(1)　データベースとの接続に使用できるクラスとインタフェース

サーブレットエンジンモードの Web アプリケーションの場合，使用できるのは java.sql パッケージで定義されたクラス，インタフェースだけです。java.sql.DriverManager クラスの getConnection メソッドを使ってデータベースコネクションを取得し，コネクションプール機能が使用できます。

J2EE サーバモードの Web アプリケーションの場合，java.sql パッケージと javax.sql パッケージで定義されたクラス，インタフェースが使用できます。このうち，javax.sql.DataSource クラスの getConnection メソッドを使ってデータベースコネクションを取得する方法をお勧めします。この場合, コネクションプール機能が使用でき, さらにローカルトランザクションやグローバルトランザクション配下でデータベースコネクションを使用できるようになります。

javax.sql.DataSource クラスを使用してデータベースコネクションを取得する例を次に示します。

```
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;
```

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, java.io.IOException {

 . . .
  InitialContext ic = new InitialContext();              // JNDIの初期コンテキストの生成
  DataSource ds = (DataSource)ic.lookup( “java:comp/env/MyDB” );    // DataSourceをlookupにより取得
  Connection con = ds.getConnection();                   // DataSourceからコネクションを取得・・・
}
```

(2)　ルックアップの方法

J2EE サーバモードでは javax.sql.DataSource クラスを JNDI のルックアップで取得します。ルックアップの実装および設定方法を次に示します。なお，ここでは，サーバ管理コマンドを使用した設定方法について説明します。

- ルックアップ名称の定義

  Web アプリケーションの開発時に，Web アプリケーションの lookup メソッドの引数に「java:comp/env/<<res-ref-name>タグ指定値>」と指定して実装します。

- リソースの参照の定義

  次のどちらかの方法でリソースの参照を定義します。

  - Web アプリケーションの開発時に定義します。Web アプリケーションの DD（web.xml）を編集し，<resource-ref>タグでリソースの参照を定義します。
  - システム構築時に定義します。サーバ管理コマンドで J2EE アプリケーションを J2EE サーバにインポートし，Web アプリケーションの WAR 属性ファイルの<resource-ref>タグで，リソースの参照を定義します。

- DB Connector への関連づけ

  システム構築時に定義します。サーバ管理コマンドで J2EE アプリケーションを J2EE サーバにインポートし，Web アプリケーションの WAR 属性ファイルの<resource-ref>タグ下の<linked-to>タグで，関連づける DB Connector の表示名を定義します。

なお，DB Connector では，ルックアップされるための設定は必要ありません。

(3)　コネクションプーリングの方法

J2EE サーバモードでは，Web アプリケーションは DB Connector というリソースアダプタを経由してデータベースに接続します。DB Connector は，Web アプリケーションに対して JDBC のインタフェースを提供します。また，DB Connector を使用する場合，Web アプリケーションで開始したトランザクションの配下でデータベースに接続できます。

サーブレットエンジンモードで使用する java.sql.DriverManager では，getConnection メソッドの引数に与えるデータベース URL によって接続先データベースを指定しますが，J2EE サーバモードで使用する DB Connector は，データベースとの接続方法やトランザクションの形態によって実装が分かれています。このため，用途に合わせて DB Connector を選択し，接続先データベースやコネクションプーリングなどを DB Connector のプロパティ（サーバ管理コマンドを使用する場合は，Connector 属性ファイルに設定）に定義します。

サーブレットエンジンモードと同様に J2EE サーバモードでもトランザクション管理なしでデータベースを使用する場合に，DABroker Library を使用するときは，次の DB Connector を選択します。

- DBConnector_DABJ_CP.rar

  DABroker Library 用の DB Connector です。ローカルトランザクションを使用する場合，またはトランザクション管理なしで使用する場合（トランザクションのサポートレベルに LocalTransaction または NoTransaction を指定する場合）に選択します。DABroker Library の

ConnectionPoolDataSource を使用してデータベースに接続します。接続可能なデータベースは，HiRDB と Oracle です。

なお，DBConnector_DABJ_CP.rar 以外の DB Connector については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6.1　DB Connector による接続の概要」を参照してください。

3.3.5　Web アプリケーションに組み込むサーブレットフィルタ

サーブレットエンジンモードと J2EE サーバモードでは，使用できるサーブレットフィルタ（built-in フィルタ）の種類と設定方法が異なります。

(1)　built-in フィルタの種類

サーブレットエンジンモードでは，HTTP レスポンス圧縮フィルタを使用できますが，J2EE サーバモードの場合は，これらに加えて，次の built-in フィルタも使用できます。

- セッションフェイルオーバ用フィルタ

  メモリセッションフェイルオーバ機能で使用するフィルタです。J2EE アプリケーションで実行中のセッション情報を管理します。

built-in フィルタについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.9.1　アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）」を参照してください。

(2)　built-in フィルタの設定

サーブレットエンジンモードの場合は，アプリケーション開発時に Web アプリケーションの DD（web.xml）を直接編集し，built-in フィルタを Web アプリケーションに組み込みます。J2EE サーバモードの場合は，これに加えて，次の手順で built-in フィルタを Web アプリケーションに組み込むことができます。

1. システム構築時に，サーバ管理コマンドで J2EE アプリケーションを J2EE サーバにインポートする。
2. 属性ファイルを編集してフィルタを組み込む。

J2EE サーバモードの場合は，サーバ管理コマンドを使用して設定する方法をお勧めします。

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

3.3.6　Web アプリケーションのリロード

サーブレットエンジンモードと J2EE サーバモードでは，Web アプリケーションのリロード機能が異なります。サーブレットエンジンモードと J2EE サーバモードでのリロード機能の違いを次に示します。

項目	サーブレットエンジンモードの場合	J2EE サーバモードの場合
リロード方法	更新検知によるリロード	更新検知によるリロード，またはコマンド（cjreloadapp）によるリロード
リロードの対象範囲	更新があったファイル※1	更新があったファイルとそのファイルを参照するファイル

項目	サーブレットエンジンモードの場合	J2EE サーバモードの場合
セッション情報の引き継ぎ	セッション情報を引き継ぐかどうか変更できる	常にセッション情報を引き継ぐ
リロードの設定方法	Web コンテナサーバの usrconf.properties，または hitachi_web.properties	J2EE サーバの usrconf.properties※2

注※1

参照先のファイルが更新された場合，参照元のファイルは更新されません。

注※2

Smart Composer 機能でシステムを構築する場合には，Smart Composer 機能で提供するファイルにリロード機能の設定をします。運用管理ポータルでシステムを構築する場合には，運用管理ポータル上でリロード機能の設定をします。

リロード時のリクエストの処理や遅延実行など，このほかの Web アプリケーションのリロード機能には，違いはありません。

サーブレットエンジンモードのリロード機能については，「3.4.2　Web アプリケーションの更新検知とリロード」を参照してください。J2EE サーバモードのリロード機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「13.8　J2EE アプリケーションの更新検知とリロード」を参照してください。

サーブレットエンジンモードと J2EE サーバモードの usrconf.properties で指定する，リロード機能のキーとそのデフォルト値の対応を次に示します。

設定項目	サーブレットエンジンモードでの usrconf.properties のデフォルト値※1	J2EE サーバモードでの usrconf.properties のデフォルト値
Web アプリケーションの更新検知	webserver.context.reloadable=false	ejbserver.deploy.context.reload_scope=none
Web アプリケーションの更新検知インターバル	webserver.context.check_interval=1	webserver.context.check_interval デフォルト値はなし (ejbserver.deploy.context.check_interval キーに設定した値（デフォルト値は 0）を使用する)。
Web アプリケーションの構成ファイル更新用インターバル	webserver.context.update.interval=0	webserver.context.update.interval デフォルト値はなし (ejbserver.deploy.context.update.interval キーに設定した値（デフォルト値は 0）を使用する)。
Web アプリケーションの遅延実行	webserver.context.reload_delay_timeout=0	webserver.context.reload_delay_timeout=0
JSP の更新検知インターバル	webserver.jsp.check_interval=1	webserver.jsp.check_interval デフォルト値はなし (ejbserver.deploy.context.check_interval キーに設定した値（デフォルト値は 0）を使用する)。
JSP の構成ファイル更新用インターバル	webserver.jsp.update.interval=0	webserver.jsp.update.interval

設定項目	サーブレットエンジンモードでの usrconf.properties のデフォルト値※1	J2EE サーバモードでの usrconf.properties のデフォルト値
JSP の構成ファイル更新用インターバル	webserver.jsp.update.interval=0	デフォルト値はなし（ejbserver.deploy.context.update.interval キーに設定した値（デフォルト値は 0）を使用する）。
セッション情報の引き継ぎ	webserver.session.persistence.enabled=false	－※2
JSP の再コンパイルの有無	webserver.jsp.recompilable=false	－※2

（凡例）－：該当しない

注※1

hitachi_web.properties でもデフォルト値は同じです。ただし，webserver.context.check_interval と webserver.jsp.check_interval は，hitachi_web.properties では設定できません。

注※2

J2EE サーバモードの usrconf.properties には，対応するキーはありません。常にセッション情報が引き継がれます。JSP がリロード対象となっていて対象ファイルが更新された場合には，再コンパイルが実行されます。

サーブレットエンジンモードでの，Web アプリケーションの更新検知の有無（webserver.context.reloadable），および JSP の再コンパイルの有無（webserver.jsp.recompilable）の組み合わせを次に示します。

リロード機能の動作	webserver.context.reloadable キーの値	webserver.jsp.recompilable キーの値
リロード機能を使用しない	false（デフォルト値）	false（デフォルト値）
サーブレットと JSP をリロードする	true	true
サーブレットだけをリロードする	true	false（デフォルト値）
JSP だけをリロードする	false（デフォルト値）	true

J2EE サーバモードでの，J2EE アプリケーションの適用の有無と適用範囲（ejbserver.deploy.context.reload_scope），Web アプリケーションの更新検知インターバル（webserver.context.check_interval），および JSP の更新検知インターバル（webserver.jsp.check_interval）の組み合わせを次に示します。

リロード機能の動作	ejbserver.deploy.context.reload_scope キーの値	webserver.context.check_interval キーの値	webserver.jsp.check_interval キーの値
リロード機能を使用しない	none（デフォルト値）	0	0
サーブレットと JSP をリロードする	web	1 以上	1 以上
サーブレットだけをリロードする	web	1 以上	0
JSP だけをリロードする	web	0	1 以上

3.3.7　システムの起動および停止方法

サーブレットエンジンモードでシステムを起動および停止する場合には，cjstartweb コマンド，cjstopweb コマンド，cprfstart コマンドなどを使用します。

J2EE サーバモードの場合には，システムの構築方法によって，システムの起動および停止方法が異なります。

- **Smart Composer 機能を使用して構築する場合**

  cmx_start_target，cmx_stop_target などの Smart Composer 機能で提供するコマンドを使用して，Web システム（サービスユニット，J2EE アプリケーション，リソースアダプタ）を一括起動，一括停止します。

- **運用管理ポータルを使用して構築する場合**

  運用管理ポータルの「論理サーバの起動／停止」と「論理サーバのアプリケーション管理」で，論理サーバ，J2EE アプリケーション，リソースアダプタを一括起動，一括停止します。

- **ファイル編集とコマンド実行によってシステムを構築する場合**

  cjstartsv コマンド，cjstopsv コマンドなどのコマンドを使用して，J2EE サーバなどの各サーバを一つずつ起動，停止します。サーバ管理コマンドを使用して，J2EE アプリケーション，リソースアダプタを起動，停止します。

! 注意事項

サーブレットエンジンモードと J2EE サーバモードで，サーバ起動時の実行時保護機能のデフォルト値が異なるので注意してください。

- サーブレットエンジンモードで，cjstartweb コマンドを使用して Web コンテナサーバを起動する場合，デフォルトでは実行時保護機能は無効（有効にする場合には-security を指定）となっています。
- J2EE サーバモードで，cjstartsv コマンドや cjstartrecover コマンドを使用して J2EE サーバを起動する場合，デフォルトでは実行時保護機能は有効（無効にする場合には-nosecurity を指定）となっています。

3.3.8　使用するメモリ，スレッド数，ポート番号

サーブレットエンジンモードと J2EE サーバモードを比較する場合，J2EE サーバモードの方が，機能が多く，メモリやスレッド，ポートなどのリソースを多く消費します。このため，サーブレットエンジンモードから J2EE サーバモードへ移行する場合には，注意してください。

各プロセスで使用するポートを次に示します。

- **Web コンテナサーバと J2EE サーバのポートの対応**

デフォルトのポート	説明	Web コンテナサーバプロセス	J2EE サーバプロセス ※1
80（TCP）	インプロセス HTTP サーバの http ポート	−	○
8080（TCP）	簡易 Web サーバの http ポート	○	−
	管理用通信ポート	−	○
8007（TCP）	Web サーバ（リダイレクタ）からのリクエスト受付ポート	○	○
8005（TCP）	cjstopweb コマンドからのプロセス終了要求の受付ポート	○	−

デフォルトのポート	説明	Web コンテナサーバプロセス	J2EE サーバプロセス※1
900（TCP）	ネーミングサービスのリクエスト受付ポート	－	○
浮動（TCP）※2	EJB コンテナのリクエスト受付ポート	－	○
浮動（TCP）※2	サーバ管理コマンドの CUI との通信ポート。	－	○
23152（TCP）	RMI レジストリのリクエスト受付ポート	－	○
浮動（TCP）※2	運用監視エージェントの通信ポート	－	○※3

（凡例）○：使用する　－：使用しない

注※1

ここでは J2EE サーバプロセスのポートだけを説明しています。J2EE サーバモードの場合，CTM を使用する場合などで使用するポートが増えます。

注※2

デフォルトの設定です。固定に変更できます。

注※3

Management Server と連携する場合だけ使用するポートです。

- HTTP Server のポート

デフォルトのポート	説明
80（TCP）	HTTP Server の http ポート
443（TCP）	HTTP Server の https ポート

- Management Server のポート

デフォルトのポート	説明
28080（TCP）	Management Server の http ポート
28005（TCP）	Management Server の終了要求ポート（ホスト内通信に必要）
28009（TCP）	Management Server の内部通信ポート（ホスト内通信に必要）

- 運用管理エージェントのポート

デフォルトのポート	説明
20295（TCP）	Management Server との通信に使用するポート

3.4　サーブレットエンジンモードで使用できる Web コンテナの機能

Web コンテナは，J2EE サーバモードおよびサーブレットエンジンモードで使用されますが，動作モードによって使用できる機能および機能の内容が異なります。この章では，サーブレットエンジンモードで使用できる Web コンテナの機能について説明します。

Web コンテナでは，Web アプリケーションの実行機能を提供するコンテナの機能と，Web サーバと連携するためのリダイレクタの機能を提供しています。

サーブレットエンジンモードで使用できる Web コンテナおよび Web サーバ連携関連の機能を次の表に示します。

表 3-15　サーブレットエンジンモードで使用できる Web コンテナおよび Web サーバ連携関連の機能

Web コンテナおよび Web サーバ連携関連の機能		参照先マニュアル	参照個所
Web アプリケーションの実行機能		このマニュアル	3.4.1
JSP の事前コンパイルとコンパイル結果の保持	JSP 事前コンパイルを使用しない場合の JSP コンパイル結果	このマニュアル	3.4.3
セッション管理機能	セッション ID および Cookie へのサーバ ID の付加※1	基本・開発編(Web コンテナ)	2.7
アプリケーションのイベントリスナ※2		このマニュアル	3.4.4
		基本・開発編(Web コンテナ)	2.8
リクエストおよびレスポンスのフィルタリング※2		このマニュアル	3.4.5
		基本・開発編(Web コンテナ)	2.9
HTTP レスポンス圧縮機能		基本・開発編(Web コンテナ)	2.10
EJB コンテナとの連携※2		このマニュアル	3.4.7
		基本・開発編(Web コンテナ)	2.11
データベースとの接続		このマニュアル	3.4.8
Web コンテナによるスレッドの作成		このマニュアル	3.4.9
ユーザスレッドで使用できるアプリケーションサーバの機能※1		このマニュアル	3.4.11
同時実行スレッド数の制御※2	Web コンテナ単位での同時実行スレッド数の制御	このマニュアル	3.4.10
		基本・開発編(Web コンテナ)	2.16
エラーページのカスタマイズ※1※3		基本・開発編(Web コンテナ)	2.20
静的コンテンツのキャッシュ※2		このマニュアル	3.4.13

Web コンテナおよび Web サーバ連携関連の機能		参照先マニュアル	参照個所
静的コンテンツのキャッシュ※2		基本・開発編(Web コンテナ)	2.21
簡易 Web サーバ機能	簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能）※1	このマニュアル	5.2.1
	簡易 Web サーバへのアクセスの制御		3.4.14
Web サーバ（リダイレクタ）によるリクエストの振り分け※1		基本・開発編(Web コンテナ)	4.2
通信タイムアウト（Web サーバ連携）※1			4.6
IP アドレスの指定（Web サーバ連携）※1			4.7
エラーページのカスタマイズ（Web サーバ連携）※1※3			4.8
ドメイン名指定でのトップページの表示※2		このマニュアル	3.4.12
		基本・開発編(Web コンテナ)	4.9
Web コンテナへのゲートウェイ情報の通知※1		基本・開発編(Web コンテナ)	4.10
SSL 使用による認証情報とデータの暗号化※2		セキュリティ管理機能編	7.2
ロールに基づく Web コンテナのユーザ認証※2		このマニュアル	3.4.6
SecurityManager による Web コンテナの実行時の保護※2			
Web アプリケーションの更新検知とリロード			3.4.2

（凡例）

基本・開発編(Web コンテナ)：マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」を示す

セキュリティ管理機能編：マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を示す

注※1

使用できる機能および使用方法は，J2EE サーバモードとの差異はありません。サーブレットエンジンモードの場合，J2EE サーバを Web コンテナサーバに置き換えてください。

注※2

これらの機能の概要については，動作モードの違いによる差異はありませんが，サーブレットエンジンモードの場合，使用方法や使用できる機能に差異があります。サーブレットエンジンモードの場合の差異については，次項以降を，機能の概要については，参照先に示したマニュアルを参照してください。

注※3

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

アプリケーションサーバで提供する Web コンテナ Web コンテナおよび Web サーバ連携関連の機能には，J2EE で規定された機能にアプリケーションサーバ独自の機能を拡張したものと，アプリケーションサーバ独自の機能として提供しているものがあります。アプリケーションサーバ独自の機能かどうかについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「1.2　システムの目的と機能の対応」を参照してください。

以降の節で，サーブレットエンジンモードで使用できる Web コンテナおよび Web サーバ連携関連の機能の詳細について説明します。

参考

アプリケーション開発時に，アプリケーションサーバが提供する cjjsp2java コマンドを使用することで，JSP ファイルから Java ソースファイルを生成し，JSP に記述ミスがないかを確認できます。

3.4.1　Web アプリケーションの実行

Web コンテナでは，Web アプリケーションを実行できます。ただし，サーブレットエンジンモードの場合，Java Servlet2.4 仕様，および JavaServer Pages（JSP）2.0 仕様に準拠した Web アプリケーションは実行できません。また，Web コンテナを使用して Web アプリケーションを実行するには，必ず Web サーバとして HTTP Server または Microsoft IIS を使用する必要があります。

(1)　Web アプリケーションのデプロイとアンデプロイ

ここでは，サーブレットエンジンモードの場合の Web アプリケーションのデプロイおよびアンデプロイについて説明します。

(a)　Web アプリケーションのデプロイ

サーブレットエンジンモードの場合，WAR 形式の Web アプリケーションをデプロイするには，次に示す二つの方法があります。

- WAR ファイルのコピーによるデプロイ
- ディレクトリのコピーによるデプロイ

Web アプリケーションのデプロイの手順については，「3.7.7　Web アプリケーションの設定」を参照してください。

複数の Web アプリケーションのデプロイ

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

(b)　Web アプリケーションのアンデプロイ

サーブレットエンジンモードの場合，次に示す二つの方法があります。

- WAR ファイルの削除によるアンデプロイ
- ディレクトリの削除によるアンデプロイ

Web アプリケーションのアンデプロイの手順については，「3.7.7　Web アプリケーションの設定」を参照してください。

(2)　Web アプリケーションを実行する場合の留意事項

ここでは，Web アプリケーションを実行する場合の留意事項について説明します。

(a)　サーブレット／JSP のデフォルトマッピングについて

Java Servlet Specification v2.3 では，クライアントがリクエストする URL に対してどのサーブレットが呼び出されるかの設定（サーブレットマッピング）を DD（WEB-INF/web.xml）内に記述するように求めていますが，Web コンテナでは，これ以外にも次に示すマッピングをデフォルトで定義しています。

URL	取り扱い
*.jsp	JSP ファイルとして取り扱われます。
/servlet/*	WEB-INF/classes 以下にあるサーブレットクラス，または WEB-INF/lib 以下にある JAR ファイルに含まれるサーブレットクラスが呼び出されます。

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

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

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

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

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

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

(c)　サーブレットのインスタンス作成数

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

ただし，同一のサーブレットおよび JSP に対して，デフォルトのマッピングと DD に記述したマッピングの両方でアクセスすると，それぞれに別のインスタンスが生成されます。

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

(d)　クエリ文字列の解析について

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

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

3.4.2　Web アプリケーションの更新検知とリロード

サーブレットエンジンモードでは，Web コンテナ上のアプリケーションの更新を検知し，更新したアプリケーションをリロードする機能を備えています。この機能によって，Web コンテナサーバを再起動することなく，デプロイ済みのサーブレットや JSP を動的に入れ替えることができます。この機能は，Web アプリケーション開発時のテストを支援する機能として利用します。

リロード機能には，次の二つがあります。

- Web アプリケーションのリロード機能

  Web コンテナサーバ起動中に Web アプリケーションを更新すると，Web コンテナは Web アプリケーションが更新されたことを検知し，更新後の Web アプリケーションをリロードします。
- JSP ファイルの再コンパイル機能

  Web コンテナサーバ起動中に JSP を更新すると，Web コンテナは JSP が更新されたことを検知し，更新後の JSP を再コンパイルします。

これらの機能によって，Web コンテナサーバにロード済み※のサーブレットまたは JSP を，Web コンテナサーバ起動中に更新後のサーブレットまたは JSP に置き換えができます。

注※

Web コンテナサーバの起動後にアクセスされたサーブレット／JSP，または web.xml の<load-on-startup>タグに指定されているサーブレット／JSP を指します。

- **更新検知の対象となるファイル**

  Web アプリケーションのリロード機能および JSP ファイルの再コンパイル機能の対象は，webapps 下にデプロイされた Web アプリケーションのディレクトリ下のファイルです。

  webapps は，次の場所にあります。

  - Windows の場合

    <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥webapps
  - UNIX の場合

    /opt/Cosminexus/CC/web/containers/<サーバ名称>/webapps

  なお，WAR ファイルのコピーでのデプロイの場合，Web コンテナサーバ起動後に展開されたディレクトリ内のファイルが対象です。

Web アプリケーションのリロードと，JSP ファイルの再コンパイルについて次に示します。

(1)　Web アプリケーションのリロード

Web アプリケーションのリロードの条件について説明します。

(a)　Web アプリケーションのリロードの条件

Web コンテナサーバ起動後，更新検知対象のファイルが次に示す条件でリロードされます。

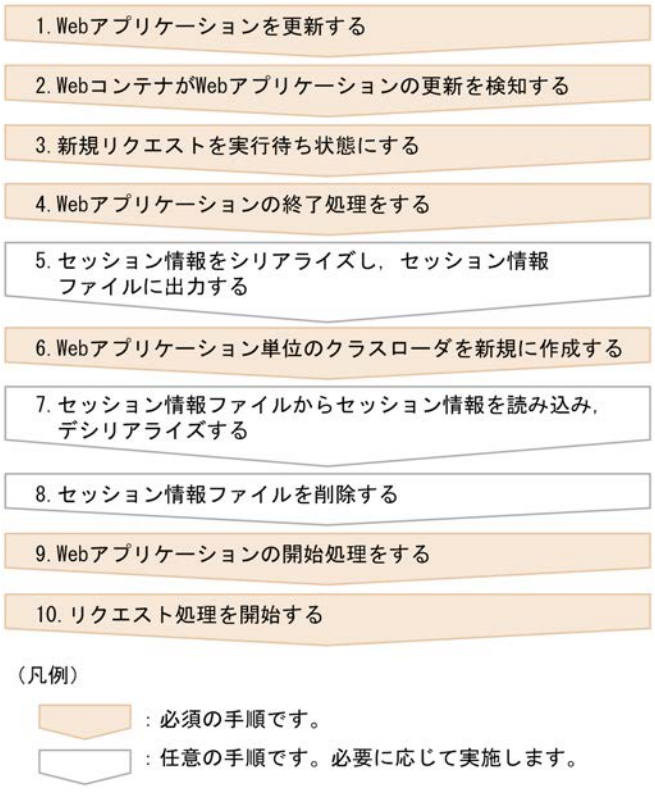
表 3-16　リロードの条件

更新検知対象のファイル種別	リロードの条件
Windows の場合 WEB-INF¥classes ディレクトリ下のクラスファイルおよびリソースファイル（java.util.ResourceBundle によってロードされたファイルなど） UNIX の場合 WEB-INF/classes ディレクトリ下のクラスファイルおよびリソースファイル（java.util.ResourceBundle によってロードされたファイルなど）	• ロード時の更新日時と異なる場合 • ロードされたファイルが削除された場合
Windows の場合 WEB-INF¥lib ディレクトリ下の JAR ファイル UNIX の場合 WEB-INF/lib ディレクトリ下の JAR ファイル	• ロード時の更新日時と異なる場合 • JAR ファイルが追加された場合 • JAR ファイルが削除された場合

(b)　リロード処理の流れ

Web アプリケーションの更新を検知して，リロードするまでの流れを次の図に示します。

図 3-1　Web アプリケーションの更新検知からリロードまでの動作



1. Web アプリケーションを更新する
2. Web コンテナが Web アプリケーションの更新を検知する

Web コンテナは，設定した間隔で Web アプリケーションが更新されているかどうかチェックします。この間隔は，更新検知のインターバルで設定します。詳細については，「(2)(a)　更新検知のインターバル」を参照してください。

また，更新を検知してから，手順 3.の処理を実施するまでのインターバルを設定できます。インターバルの設定については，「(2)(b)　リソース更新用インターバル」を参照してください。

3. 新規リクエストを実行待ち状態にする

なお，処理中のリクエストはそのまま実行します。

デフォルトでは，新規リクエストを実行待ちにしますが，処理中のリクエストの処理が終わってから，新規リクエストを実行待ちにすることもできます。この場合，リロード遅延実行の設定が必要になります。リロード遅延実行については，「(2)(c)　リロード遅延実行（任意)」を参照してください。リロード遅延実行の設定は任意となります。

4. Web アプリケーションの終了処理をする

終了処理では次の処理が実行されます。

- リロード前のクラスローダにローディングされたサーブレットのインスタンスが破棄されます。サーブレットが destroy メソッドを実装している場合，destroy メソッドが実行されます。また，JSP ファイルから生成されたサーブレットのインスタンスも破棄されます。この際，JSP ファイルが，jspDestroy メソッドを実装している場合，jspDestroy メソッドが実行されます。
- javax.servlet.ServletContext に登録されたオブジェクトは破棄されます。

5. セッション情報（javax.servlet.http.HttpSession オブジェクトに登録したオブジェクト）をシリアライズ※し，セッション情報ファイルに出力する

セッション情報を引き継ぐかどうかを設定できます。セッション情報を引き継ぎたい場合に指定してください。詳細については，「(2)(d)　リロード時のセッション引き継ぎ（任意)」を参照してください。

6. Web アプリケーション単位のクラスローダを新規に作成する

Web アプリケーションのリロード処理が実行されると，Web アプリケーション単位のクラスローダが新たに作成され，リロード後のリクエスト処理で使用されます。すでにロードされているサーブレットについては，destroy メソッドが実行され，インスタンスを破棄します。

7. セッション情報ファイルからセッション情報を読み込み，デシリアライズ※する

セッション情報ファイルからセッション情報を読み込み，リロード後のクラスローダ上でデシリアライズします。セッション情報の引き継ぎを設定している場合だけ実施されます。

8. セッション情報ファイルを削除する

セッション引き継ぎを指定している場合だけ実施されます。

9. Web アプリケーションの開始処理をする

リロード後は，初回アクセス時に更新後のサーブレットのインスタンスが作成され，init メソッドが実行されます。

10. リクエスト処理を再開する

3.で実行待ちにしていたリクエストの処理を再開します。

注※

シリアライズ処理，およびデシリアライズ処理の時間は，リロードを実行している Web アプリケーション上で生成されたセッション数，javax.servlet.http.HttpSession オブジェクトに登録したオブジェクトの容量などに依存します。

(2)　リロードの動作に関する設定

アプリケーションの更新検知とリロード機能は，デフォルトの設定では実行されません。必要に応じて実行するように設定してください。

この機能では，次の動作をカスタマイズできます。

- 更新検知のインターバル
- リソース更新用インターバル
- リロード遅延実行（任意）
- リロード時のセッション引き継ぎ（任意）

リロードの動作に関する設定は，Web コンテナサーバのプロパティ，または Web コンテナ上で動作する Web アプリケーションプロパティとして設定します。アプリケーションの更新検知とリロードの設定については，「3.7.4　Web コンテナサーバの動作設定のカスタマイズ」を参照してください。

(a)　更新検知のインターバル

Web アプリケーションの更新を検知する間隔を設定できます。更新検知は，デーモンスレッドによって実行されます。デフォルトでは 1 秒間隔となっています。

更新検知インターバルの値を大きくすると，構成ファイルを監視する間隔が長くなり，ファイル更新後のリロードの反映が遅くなります。また，値を小さくすると，リロードの反映が早くなります。

ポイント

更新検知の対象となるファイル数が増えると，更新検知処理のオーバーヘッドが大きくなり，CPU 使用率が高くなります。このような場合は，更新検知インターバルを変更することによって，性能への影響が小さくなります。更新検知インターバルの値は大きくすることをお勧めします。

(b)　リソース更新用インターバル

Web アプリケーションの更新を検知してから，処理中のリクエスト数の監視を実施するまでの時間を設定できます。

Web コンテナは，ファイル（リソース）の更新を検知すると，更新するファイルをコピーし，処理中のリクエスト数の監視を開始します。処理中のリクエストの処理が完了すると，コピーしたファイルをロードします。ファイルの容量が大きい場合，ネットワークを経由した場合や，ファイルが複数ある場合には，ファイルコピーなどの処理時間が長くなることがあります。このとき，ファイルコピーが完了する前に処理中のリクエストがなくなると，ファイルコピー中にロードが開始されて，ロードに失敗するおそれがあります。これを回避するため，ファイルコピーに掛かる時間を考慮してコピーが完了してからロードが開始されるように，構成ファイルの更新を検知してから処理中のリクエスト数の監視を開始するまでの時間をリソース更新用インターバルとして設定しておくことができます。

ポイント

- リソース更新用インターバルには，実際に掛かるコピー時間にゆとりを持たせた時間を設定することをお勧めします。
- 複数のファイルを同時に更新する場合，複数のファイルの更新に掛かる時間を算出して，設定してください。

(c)　リロード遅延実行（任意）

ファイル（リソース）の更新を検知してから，リロードを開始するまでの時間を設定できます。リロード遅延実行の時間を設定している場合に，Web アプリケーションの更新を検知すると，リロードの開始を遅延

実行に指定した時間まで遅らせます。指定した時間内に処理中のリクエストがなくなるか，または指定した時間がくると，アプリケーションのサービスを停止してリロードを開始します。これによって，処理中のリクエストの実行待ち時間を減らすことができるため，アプリケーションのサービス停止期間を最小限にできます。

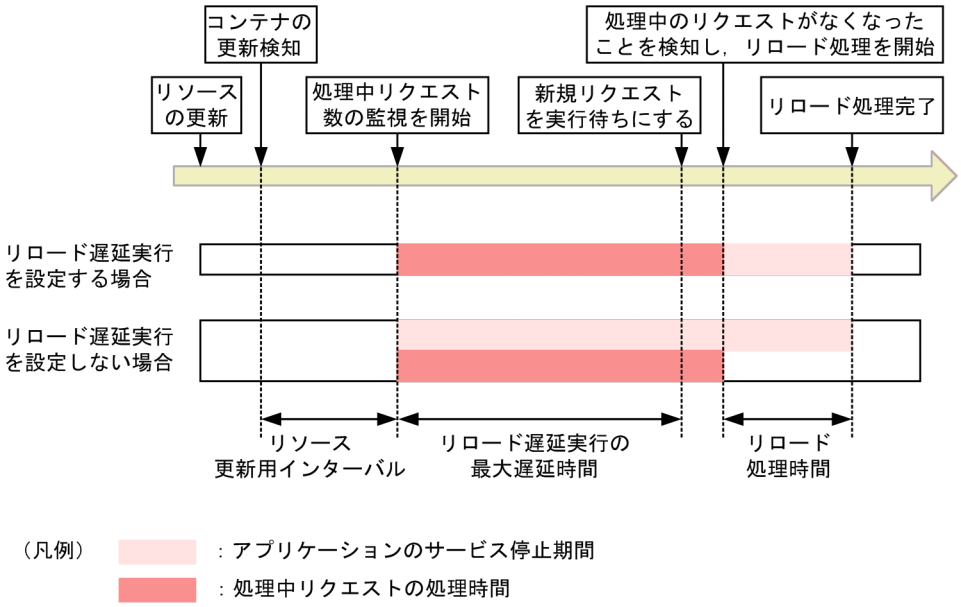
なお，デフォルトでは，リロード遅延実行が設定されていません。デフォルトでは次のような動作となります。

1. Web コンテナがリソースの更新を検知すると，アプリケーションのサービスを停止する
2. リクエストを次のように処理する
   - 新規のリクエスト：実行待ちにする
   - 処理中のリクエスト：すべてのリクエストを処理する
3. 処理中のリクエストの処理が完了すると，リロードが実施される

リロードの遅延実行が設定されていないと，アプリケーションのサービス停止の期間が長くなります。これを回避するため，リロード遅延実行を設定することをお勧めします。

リロード遅延実行を設定した場合と設定しない場合の，アプリケーションのサービス停止期間の違いを，次の図に示します。

図 3-2　アプリケーションのサービスの停止期間の違い



(d)　リロード時のセッション引き継ぎ（任意）

Web アプリケーションのリロード実行前に生成したセッション情報を，リロード後も継続して利用するかどうかを設定できます。これによって，リロード前に javax.servlet.http.HttpSession オブジェクトに登録されているオブジェクトを，リロード後も引き続き利用できるようになります。セッションの情報は，シリアライズされてセッション情報ファイルに出力されます。セッション情報ファイルは，リロード処理開始時に作成されます。そして，リロード後にセッション情報がクラスローダ上でデシリアライズされ，セッション情報ファイルはリロード処理完了時に削除されます。

なお，セッション引き継ぎ機能を使用しない場合，リロード処理が実行されるとセッション情報はすべて失われます。

(e)　注意事項

Web アプリケーションのリロード機能を使用する際の，注意について説明します。

● リロード機能の使用に関する注意

- Web アプリケーションのリロード機能を使用する場合，web.xml に<error-page>タグを定義しないでください。
- Web アプリケーションのリロード機能を使用する場合，Web アプリケーションから JNI を利用することはできません。リロード処理では Web アプリケーション単位のクラスローダを置き換えますが，JNI の仕様では，異なるクラスローダ上で同じネイティブライブラリをロードできないという制限があるためです。
- Web アプリケーションのリロード機能は，Web アプリケーション単位のクラスローダを入れ替えることによって，Web コンテナ起動中の Web アプリケーションの入れ替えを実現しています。そのため，リロード後にリロード前の Web アプリケーション単位のクラスローダ，または Web アプリケーション単位のクラスローダでロードされたクラスが参照されるとメモリリークが発生します。

  次に，該当する場合を示します。

  - スレッドの生成

    スレッドはコンテキストクラスローダを保持しますが，デフォルトでは親スレッドのコンテキストクラスローダを保持します。Web アプリケーション実行時のスレッドは，Web アプリケーション単位のクラスローダをコンテキストクラスローダとしているため，生成されたスレッドは Web アプリケーション単位のクラスローダをコンテキストクラスローダとします。そのため，リロード後も該当スレッドが存在すると，リロード前の Web アプリケーション単位のクラスローダが解放されないでメモリリークが発生します。
  - java.lang.Thread クラスまたはその派生クラスのインスタンス生成

    java.lang.Thread クラスまたはその派生クラス（以降，スレッドクラス）のインスタンスを作成すると，所属するスレッドグループから該当スレッドクラスのインスタンスが参照されます。スレッドグループからの参照は，スレッドの実行を終了（run メソッドの完了）すると解放されます。そのため，Web アプリケーション内でスレッドクラスのインスタンスを生成し，run メソッドを実行しなかった場合は，該当スレッドクラスのインスタンスが解放されません。また，スレッドはコンテキストクラスローダとして Web アプリケーション単位のクラスローダを保持しているため，Web アプリケーション単位のクラスローダが解放されないでメモリリークが発生します。
  - スレッドローカル変数の使用

    Web コンテナではリクエストを処理するスレッドはプールされていて，Web コンテナ終了時まで終了しません。そのため，スレッドローカル変数に Web アプリケーションに含まれるクラスのインスタンスを格納すると，メモリリークが発生します。
  - Web アプリケーション外からの参照

    usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）の web.add.class.path キーで指定したクラスパスに存在するクラスは，システムクラスローダからロードされます。リロード実行時に該当クラスはアンロードされません。

    このため，該当クラスのクラス変数から Web アプリケーション単位のクラスローダ，または Web アプリケーション単位のクラスローダでロードされたクラスを参照すると，メモリリークが発生します。
- Web アプリケーションのリロード機能と，JSP の再コンパイル機能を併用する場合，次のことに注意してください。
  - JSP ファイルだけを更新した場合，Web アプリケーションのリロードは実行されません。

- JSP ファイルの再コンパイルが実行されている場合には，リロード処理での JSP の再コンパイルは停止されることがあります。この場合，JSP ファイルの再コンパイルは，リロード完了後の該当する JSP に対して初めてアクセスするときに実行されます。また，web.xml の<load-on-startup>タグに該当する JSP ファイルが指定されている場合は，リロード処理内の Web アプリケーションの初期化処理で実行されます。

● リソースの更新に関する注意

- 次の場合にはメモリ不足が発生するおそれがあるため，リソースの更新はリクエストのピーク時など負荷の高いときを避けて実施してください。
  - リロードの処理は，クラスローダを再作成してクラスをロードし直すため，メモリを多く消費します（ただし，消費するメモリ使用量は，Web アプリケーションの実装に依存します）。そのため，例えば，複数の Web アプリケーションで，同時にリロード処理を繰り返すと，メモリ不足が発生するおそれがあります。
  - リロード処理によって不要になったリソースが JavaVM から解放されるタイミングは，JavaVM の GC の実行のタイミングに依存します。リクエストのピーク時など，負荷が高いときにリロードを繰り返すと，GC を実行するタイミングがなく，メモリ不足が発生するおそれがあります。
- 複数のリソースを更新する場合，リソース更新用インターバルに，複数のリソースの更新（ファイルコピー）に掛かる時間を指定してください。なお，いったん，リソースを更新したら，リロード処理が完了する前に別のリソースを更新しないようにしてください。
- ロード済みのリソース（ファイル）を削除した場合も Web アプリケーションに対する変更と判断されるため，リロードが実行されます。このため，不用意にリソースを削除しないよう，注意してください。

● リロード処理実行時の注意

- Web アプリケーションのリロード処理が実行されると，Web アプリケーション単位のクラスローダを置き換えるため，リロード前に作成されたインスタンスをリロード後のサーブレットから使用できません。
- 更新前のサーブレットが javax.servlet.ServletContext に登録したオブジェクトはクリア（削除）されます。更新前のサーブレットが生成したセッションは破棄され，更新後のサーブレットにアクセスすると新しいセッションが生成されます。さらに，Web アプリケーションのリロード時には，そのときロードされている JSP もリロードされます。
- Web アプリケーションのリロード機能を使用すると，更新検知処理のためのオーバーヘッドが発生し，監視対象のファイル数の増加に伴ってオーバーヘッドが大きくなるため，更新検知間隔を変更して性能への影響を小さくする必要があります。
- リロード処理中に，cjstopweb コマンドで Web コンテナサーバに対して終了要求を発行した場合は，次のように動作します。
  - リロード処理内の Web アプリケーションの初期化処理の開始前の場合
    リロード処理を中断し，終了処理が実行されます。
  - リロード処理内の Web アプリケーションの初期化処理の開始後の場合
    リロード処理完了後に，Web コンテナサーバの終了処理が実行されます。このとき，リロード処理による実行待ちのリクエストはエラーになります。

● リロードの処理時間に関する注意

リロード処理時間は，Web アプリケーションの実装に大きく依存します。このため，次の点に注意が必要です。

- セッション情報の容量は，シリアライズ処理時間，デシリアライズ処理時間に大きく影響します。シリアライズ対象となるセッション情報は，必要十分な容量にしてください。
- リロード処理では，Web アプリケーションの終了処理，初期化処理が実行されます。終了処理，初期化処理が長い場合，リロードの処理時間も長くなり，Web アプリケーションのサービス停止期間にも影響があるため注意してください。ただし，リロード処理で初期化処理が実行されるのは，web.xml の <load-on-startup>タグを設定したサーブレットや JSP だけです。<load-on-startup>タグを設定していないサーブレットや JSP の初期化処理は，リロード完了後の初回アクセス時に実行されます。

● セッション引き継ぎ機能を使用する場合の注意

- リロード時のセッション引き継ぎ機能を使用する場合，javax.servlet.http.HttpSession に登録するセッション情報は，シリアライズできるオブジェクトである必要があります。
- HttpSession オブジェクトに登録したオブジェクトから参照するオブジェクトは，transient で宣言されているか，またはシリアライズできるオブジェクトである必要があります。以降，参照するオブジェクトも同様です。シリアライズできるオブジェクトについての詳細は，J2SE の仕様書を参照してください。
- セッション情報としてシリアライズできないオブジェクトを登録した場合は，エラーが発生します。エラーの内容はオブジェクトによって異なります。
  - javax.servlet.http.HttpSession に登録したオブジェクトの場合
    HttpSession オブジェクトに登録したオブジェクトが，シリアライズできないオブジェクトである場合，該当するオブジェクトだけを破棄します。破棄されたオブジェクトは，リロードが完了したあとに使用することはできません。また，該当するオブジェクトが，javax.servlet.http.HttpSessionBindingListener インタフェースを実装していても，Web コンテナからアンバインドされたイベントは通知されません。
  - javax.servlet.http.HttpSession に登録したオブジェクトから参照されるオブジェクトの場合
    HttpSession に登録したオブジェクトから参照されるオブジェクトがシリアライズできないオブジェクトの場合，すべてのセッション情報が破棄されます。HttpSession に登録したオブジェクトから参照されるオブジェクトが，HttpSession に登録したオブジェクトの場合も同様に，すべてのセッション情報が破棄されます。
- シリアライズ対象となるセッションに登録したオブジェクトのクラス，およびそこから参照されるオブジェクトのクラスを，デシリアライズできない構成に更新してリロードを実行した場合，セッションのデシリアライズに失敗します。デシリアライズに失敗すると，Web アプリケーション上のすべてのセッション情報が破棄されるので注意してください。

(3)　JSP ファイルの再コンパイル

Web コンテナに配置された JSP ファイルの更新日時がロード時と異なる場合，再コンパイルされるように設定できます。ただし，JSP ファイルから生成された Java ファイル，またはクラスファイルを更新しても，再コンパイルの対象にはなりません。

JSP の再コンパイル機能は，デフォルトの設定では実行されません。必要に応じて実行するように設定してください。

この機能では，次の動作をカスタマイズできます。

- 更新検知のインターバル
- JSP ファイル更新用インターバル

JSP ファイルの再コンパイルに関する設定は，Web コンテナサーバのプロパティ，または Web コンテナ上で動作する Web アプリケーションプロパティとして設定します。アプリケーションの更新検知とリロードの設定については，「3.7.4　Web コンテナサーバの動作設定のカスタマイズ」を参照してください。

(a)　更新検知のインターバル

JSP ファイルの更新を検知する間隔を指定できます。更新検知は，デーモンスレッドによって実行されます。デフォルトでは 1 秒間隔となっています。

ポイント

更新検知の対象となるファイル数が増えると，更新検知処理のオーバーヘッドが大きくなり，CPU 使用率またはメモリ使用量が増加します。このような場合は，更新検知インターバルを変更することによって，性能への影響が小さくなります。更新検知インターバルの値は大きくすることをお勧めします。

(b)　JSP ファイル更新用インターバル

JSP ファイル更新用インターバルとは，JSP ファイルの更新を検知してから，JSP ファイルをコンパイルするまでの間隔を指します。

- 設定の指針
  - JSP ファイル更新用インターバルには，実際にファイルのコピーに掛かる時間よりも長い時間を設定することをお勧めします。
  - 複数のファイルを同時に更新する場合，複数のファイルの更新に掛かる時間を算出して，設定してください。

(c)　注意事項

- JSP ファイルの再コンパイル機能を使用すると，更新検知処理のためのオーバーヘッドが発生します。監視対象の JSP ファイル数が増えると，オーバーヘッドも大きくなります。この場合は，更新検知間隔を長くすることで，性能への影響を少なくできます。
- include ディレクティブを使用してインクルードされた JSP ファイルを更新しても，インクルード元の JSP ファイルは再コンパイルされません。再コンパイルを実行する場合には，インクルード元の JSP ファイルを更新してください。

3.4.3　JSP ファイルのコンパイル結果の保持

JSP は，Web コンテナ上でコンパイルされ，Java ソースファイルおよびクラスファイルが生成されます。Web コンテナでは，JSP のコンパイル結果である，Java ソースファイルおよびクラスファイルを，Web コンテナサーバの再起動時に保持するかどうか設定できます。

JSP のコンパイル結果の保持機能は，デフォルトの設定では実行されません。必要に応じて実行するように設定してください。

JSP のコンパイル結果の保持に関する設定は，Web コンテナサーバのプロパティをカスタマイズして設定します。設定については，「3.7.4　Web コンテナサーバの動作設定のカスタマイズ」を参照してください。

ここでは，コンパイル結果が生成，削除されるタイミングについて説明します。また，Web アプリケーションがアンデプロイされた場合，および JSP ファイルが更新された場合の JSP コンパイル結果の動作についても説明します。

(1)　コンパイル結果の生成

JSP のコンパイル結果は，次のタイミングで生成されます。

- JSP への初回アクセス時
- DD（web.xml）で JSP に対し<load-on-startup>が指定されている Web アプリケーションの開始時

(2)　コンパイル結果の削除

JSP のコンパイル結果は，次のタイミングで削除されます。

- Web コンテナサーバの起動時※1
- Web コンテナサーバの終了時※2

注※1

次のどちらかの場合にだけ削除されます。

- JSP コンパイル結果を保持しない設定にしている場合
- JSP 用テンポラリディレクトリに JSP コンパイル結果が保持されている場合
  これは，サーバが強制終了された場合に備えて実施されます。

注※2

JSP コンパイル結果保持を無効にした場合だけ削除します。

(3)　Web アプリケーションアンデプロイ時の動作

サーブレットエンジンモードの場合，Web アプリケーションをアンデプロイしても JSP コンパイル結果は削除されません。このため，JSP のコンパイル結果を保持する設定で Web コンテナを稼働したあとに JSP コンパイル結果が不要となった場合，または Web アプリケーションをアンデプロイした場合は，JSP のコンパイル結果を手動で削除する必要があります。削除方法について次に示します。

削除方法

JSP コンパイル結果の削除をするには，Web コンテナサーバ終了後に，削除対象の Web アプリケーションの JSP 用テンポラリディレクトリに作成される Web アプリケーション単位のディレクトリ以下を削除します。JSP 用テンポラリディレクトリは，usrconf.properties で設定します。

(4)　JSP ファイルを更新する場合の動作

サーブレットエンジンモードでは，JSP ファイルを直接編集，またはファイルの上書きで更新できます。JSP コンパイル結果を保持している状態では，JSP の初回ロード時に，保持されたクラスファイルをロードする場合と，更新された JSP ファイルのコンパイル結果であるクラスファイルをロードする場合があります。どちらのクラスファイルをロードするかについては，更新された JSP ファイルと，保持されたクラスファイルの更新日時によって次のように決まります。

- 更新された JSP ファイルの更新日時が，保持されたクラスファイルの更新日時より新しい場合，保持されたクラスファイルは使用されません。このため，JSP のコンパイルが実行され，コンパイル後のクラスファイルがロードされます。
- 更新された JSP ファイルの更新日時が，保持されたクラスファイルの更新日時よりも古い場合，保持されたクラスファイルがそのままロードされます。

(5)　JSP ファイルがインクルードされた場合の注意事項

include ディレクティブを使用してインクルードされた JSP ファイルを更新しても，インクルード元の JSP ファイルは再コンパイルされません。再コンパイルを実行する場合には，インクルード元の JSP ファイルを更新してください。

3.4.4　アプリケーションのイベントリスナ

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

- セッションオブジェクトの新規生成
- セッションオブジェクトのシリアライズ前※
- セッションオブジェクトのデシリアライズ後※
- セッションオブジェクトの抹消
- セッションオブジェクトの属性の追加，削除，変更
- サーブレットコンテキストオブジェクトの生成
- サーブレットコンテキストオブジェクトの抹消
- サーブレットコンテキストオブジェクトの属性の追加，削除，変更

注※

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

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

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

3.4.5　リクエストおよびレスポンスのフィルタリング

サーブレット／JSP の，リクエストやレスポンスに対してフィルタリングをする機能です。この機能によって，サーブレット／JSP の実行前のリクエスト，または実行後のレスポンスをラップして，データの変更，リソースに対するトレースの取得などができます。

アプリケーションサーバでは，アプリケーションサーバ独自の機能を使用するためのサーブレットフィルタを提供しています。アプリケーションサーバが提供するサーブレットフィルタを，**built-in フィルタ**といいます。

アプリケーションサーバで提供しているサーブレットフィルタ（built-in フィルタ）のうち，サーブレットエンジンモードで使用できるのは，HTTP レスポンスの圧縮機能を使用するための built-in フィルタです。

サーブレットエンジンモードで使用できる built-in フィルタの種類と，Web アプリケーションに built-in フィルタを組み込むことによって使用できる機能を次の表に示します。

表 3-17　サーブレットエンジンモードで使用できる built-in フィルタの種類

built-in フィルタの種類	機能
HTTP レスポンス圧縮フィルタ	サーブレット，JSP，および静的コンテンツへの HTTP リクエストに対する HTTP レスポンスを gzip 形式に圧縮します。

なお，built-in フィルタは，HTTP リクエストおよび HTTP レスポンスへの作用や，フィルタ連鎖の順序制約などを考慮して，Web アプリケーションに組み込む必要があります。built-in フィルタに関する制約，および HTTP レスポンス圧縮フィルタについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.9.1　アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ)」を参照してください。

3.4.6　Web コンテナのセキュリティ管理

Web コンテナのセキュリティ管理は，次の内容を除き，J2EE サーバモードの場合と同様です。このため，セキュリティ管理の概要については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を参照してください。

ここでは，J2EE サーバモードとの差異のある機能について説明します。

(1)　ロールに基づく Web コンテナのユーザ認証

サーブレットエンジンモードの場合，Web コンテナのユーザ認証で使用するユーザ情報の管理方法が，J2EE サーバモードとは異なります。

Web コンテナでは，ユーザ情報として，各ユーザのユーザ名，パスワード，および所属するロールを保持，管理します。これらの情報を保持，管理するために，サーブレットエンジンモードでは，次に示す場所に各ユーザのユーザ名，パスワード，および所属するロールを記述します。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥web-users.xml
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/web-users.xml

記述例を次に示します。

```
<web-users>
  <user name="user1"  password="user1" roles="regular" />
  <user name="admin" password="admin" roles="admin" />
</web-users>
```

なお，web-users.xml の詳細については，「3.11.4　web-users.xml（ユーザ認証情報定義ファイル)」を参照してください。

(2)　SecurityManager による Web コンテナの実行時の保護

Java2 の SecurityManager に基づいた Web コンテナの実行時保護を実現します。これによって，次に示すような現象を防ぐことができます。

- 内部で System.exit()などを発行するような，不正なサーブレット，JSP によって Web コンテナ全体が勝手に終了する。

- システムプロパティなどを勝手に書き換えるような不正なサーブレット，JSP によって，Web コンテナの実行に不具合が生じる。

サーブレットエンジンモードの場合，-security オプションを指定して，cjstartweb コマンドを実行することで Web コンテナの保護が有効になります。なお，Web コンテナの実行時の保護機能を使用すると，サーブレットまたは JSP ファイルから，EJB を呼び出すことができないので注意してください。

3.4.7　EJB コンテナとの連携

ここでは，サーブレットエンジンモードの場合の EJB コンテナとの連携について説明します。

(1)　Enterprise Bean の呼び出し方法

Web コンテナは EJB コンテナと連携して，J2EE サーバとして動作します。なお，EJB コンテナとの連携については，J2EE サーバモードの場合と同じです。

ここでは，サーブレットエンジンモード固有の Enterprise Bean の呼び出し方法についてだけ説明します。これ以外の Enterprise Bean の呼び出し方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「3.4　Enterprise Bean の呼び出し」を参照してください。また，Enterprise Bean 呼び出しの準備については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「3.4.2　Enterprise Bean を呼び出すための実装」を参照してください。

Enterprise Bean を呼び出すサーブレット／JSP が Web コンテナサーバで動作している場合

Enterprise Bean を呼び出すサーブレットが Web コンテナサーバで動作しているとき，次の例に示すように Enterprise Bean を呼び出します。例を次に示します。

例：

```
Context ctx = new InitialContext();
Object o =
 ctx.lookup("HITACHI_EJB/SERVERS/MyServer/EJB/APName/Cart");
CartHome h = (CartHome)PortableRemoteObject.narrow(o, CartHome.class);
Cart c = h.create();
c.call();
```

(2)　JSP からの Enterprise Bean を呼び出すときの RMI-IIOP 通信のタイムアウトを設定するときの注意

Enterprise Bean を呼び出すとき，RMI-IIOP 通信にタイムアウトを設定できます。RMI-IIOP 通信のタイムアウト時間の設定方法は，Enterprise Bean がどこから呼び出されるかによって異なります。

Enterprise Bean を呼び出す EJB クライアントの形態が JSP やサーブレットの場合で，サーブレットエンジンモードで動作する Web コンテナを利用するときは，クライアント側の JSP，サーブレットが動作する Web コンテナサーバのプロパティをカスタマイズして設定します。設定については，「3.7.4　Web コンテナサーバの動作設定のカスタマイズ」を参照してください。また，RMI-IIOP 通信のタイムアウトの概要については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「2.11.5 RMI-IIOP 通信のタイムアウト」を参照してください。

3.4.8　データベースとの接続

サーブレットエンジンモードで Web コンテナを利用している場合，Web アプリケーションから DABroker Library を通じ，Sun JDBC API 仕様に従ったデータベースアクセスができます。ただし，使用時には次に示す点に注意する必要があります。

- java.sql パッケージで定義されたクラス，インタフェースだけが使用できます。

- java.sql.DriverManager クラスの getConnection メソッドを使ってデータベースコネクションを取得します。
- コネクションシェアリング，および分散トランザクションはサポートしていません。
- コネクションプール機能を使用できます。コネクションプールの詳細については，「3.6.3(2)　パフォーマンスチューニングのための機能」を参照してください。

DABroker Library を使用したデータベースアクセスの詳細については，「4.2　DABroker Library を使用したデータベース接続の概要」を参照してください。

3.4.9　Web コンテナによるスレッドの作成

Web コンテナでは，Web サーバ連携のためのスレッド，Web アプリケーションのためのスレッドなどを作成します。これらのスレッドに対して，システムのリソースが十分であるか検討します。

ここでは，サーブレットエンジンモードの場合の，Web コンテナが作成するスレッドについて説明します。

(1)　作成されるスレッドの種類と数

Web コンテナが作成するスレッドを次に示します。

(a)　Web サーバ連携のためのスレッド

●　Web コンテナの簡易 Web サーバを使用する場合（任意）

作成するスレッド

TCP のコネクション接続要求を受信するスレッド，および受信したリクエスト処理用スレッドを作成します。

スレッドの数

TCP のコネクション接続要求を受信するスレッドは，1 スレッドです。簡易 Web サーバが受信したリクエスト処理用スレッドは，最小 5 スレッド，最大 100 スレッドです。

リクエスト処理用スレッドは，起動時に 5 スレッド作成し，同時実行スレッド数が起動済みスレッド数を超えた場合，100 スレッドまで作成されます。

なお，サーブレットエンジンモードの場合，Web コンテナの簡易 Web サーバの使用は任意となります。

●　Web サーバ連携機能を使用する場合

作成するスレッド

リダイレクタからのコネクション接続要求を受信するスレッド，およびリダイレクタから受信したリクエスト処理用スレッドを作成します。

スレッドの数

リダイレクタからのコネクション接続要求を受信するスレッドは，1 スレッドです。リダイレクタから受信したリクエスト処理用スレッドは，Web サーバと Web コンテナとのコネクション数分スレッドを作成します。

リダイレクタから受信したリクエスト処理用スレッドは，起動時に webserver.connector.ajp13.max_threads に指定された数分作成します（webserver.connector.ajp13.max_threads のデフォルトは 10）。

(b)　Web アプリケーションのためのスレッド

Web アプリケーション単位に次のスレッドを作成します。最小 1 スレッド，最大 3 スレッドです。

- セッションの有効期限監視スレッド
- Web アプリケーションのリロード監視スレッド
  （webserver.context.reloadable=true を設定した場合）
- JSP の更新監視スレッド
  （webserver.jsp.recompilable=true を設定した場合）

(c)　メインスレッド

なお，サーブレットエンジンモードの場合，メインスレッドとして 1 スレッド作成します。

(2)　作成されるスレッドの総数

Web コンテナがデフォルトの設定でプロセス起動時に作成するスレッドの総数を次に示します。ただし，この数値には Web コンテナ以外，および JavaVM が作成するスレッドは含まれていません。

作成するスレッドの総数は，レスポンス送信時のタイムアウトを設定しているかどうかによって異なります。

レスポンス送信時のタイムアウトを設定しているとき

スレッド総数=A ＋ B ＋ C ＋ D ＋ E

レスポンス送信時のタイムアウトを設定していないとき

スレッド総数=A ＋ B ＋ C ＋ D

（凡例）

A：Web コンテナの簡易 Web サーバを使用する場合のスレッド数
B：Web サーバ連携機能を使用する場合のスレッド数
C：Web アプリケーション単位のスレッド数
D：メインスレッド
E：レスポンス送信時のタイムアウトを監視するためのスレッド数

このため，プロセス起動時のスレッド総数は次のようになります。

レスポンス送信時のタイムアウトを設定しているときのスレッド総数
=6 ＋ 11 ＋（1×Web アプリケーション数）＋ 1 ＋ 1
=19 ＋ Web アプリケーション数

レスポンス送信時のタイムアウトを設定していないときのスレッド総数
=6 ＋ 11 ＋（1×Web アプリケーション数）＋ 1
=18 ＋ Web アプリケーション数

プロセス起動後，Web サーバとのコネクション数，簡易 Web サーバの同時実行数によってスレッド数が増加します。

3.4.10　同時実行スレッド数の制御

Web コンテナでは，マルチスレッドでサーブレットのリクエストを処理します。このとき，同時に実行できるスレッド数に上限を設定できます。これによって，スラッシングなどによる，パフォーマンスの低下を

防止できます。また，適切なスレッド数を設定することで，アクセス状況に従ったパフォーマンスチューニングができます。

サーブレットエンジンモードの場合，同時に実行できるスレッド数は，Web コンテナ単位で制御されます。Web アプリケーション単位でのスレッド数の制御および URL グループ単位でのスレッド数の制御はできません。

Web コンテナ単位での同時実行スレッド数の制御には，J2EE サーバモードとの機能差異はありません。このため，機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.16　Web コンテナ単位での同時実行スレッド数の制御」を参照してください。

3.4.11　ユーザスレッドで使用できるアプリケーションサーバの機能

ここでは，サーブレットエンジンモードの場合，ユーザスレッドで使用できるアプリケーションサーバの機能について説明します。ユーザスレッドの使用方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「6.2　サーブレットおよび JSP の実装時の注意事項」を参照してください。

アプリケーションサーバで提供される各機能について，ユーザスレッドで使用できるかどうかを次に示します。

(1)　サーブレット API

サーブレット API をユーザスレッドで使用する場合，リクエストオブジェクトおよびレスポンスオブジェクトは使用できません。リクエスト処理スレッドでだけ使用してください。詳細は，サーブレット仕様書の Thread Safety の項を参照してください。

(2)　リソース接続およびトランザクションサービス

サーブレットエンジンモードで使用できるリソース接続およびトランザクションサービスとして提供される機能が，サーブレット／JSP およびユーザスレッドで使用できるかどうかを次の表に示します。

表 3-18　リソース接続およびトランザクション管理の機能の使用可否（ユーザスレッド）

分類／機能名		サーブレット／JSP	ユーザスレッド
コネクションプーリング	Web コンテナコネクションプールによるコネクションプーリング	－	－
	コネクションプールのウォーミングアップ	○	○
Web コンテナコネクションプールによるコネクション取得リトライ		－	－
コネクションプールクリア		○	○
コネクションのクローズ・解放	コネクション強制クローズ	×	×

（凡例）○：使用できる　×：使用できない　－：該当しない

ユーザスレッドでリソース接続およびトランザクションサービスを使用する場合の注意事項を示します。

- SingleThreadModel インタフェースを実装したサーブレットで生成したスレッド上で，トランザクションの開始と決着，およびコネクションの取得と解放はできません。
- スレッド生成時にトランザクションを引き継ぐことはできません。

- トランザクションは，同じスレッド上で開始または決着してください。
- スレッド間でコネクションを渡すことはできません。コネクションを使用すると，動作不正となります。
- ユーザスレッドでコネクションを取得した場合，コネクションを取得したスレッド上で確実にコネクションをクローズしてください。

(3)　ログ運用および障害検知

サーブレットエンジンモードで使用できるログ運用および障害検知として提供される機能が，サーブレット／JSP およびユーザスレッドで使用できるかどうかを次の表に示します。

表 3-19　ログ運用および障害検知の機能の使用可否（ユーザスレッド）

分類／機能名	サーブレット／JSP	ユーザスレッド
ユーザログ出力	○	○
性能解析トレース	○	○

（凡例）○：使用できる

(4)　J2EE アプリケーション運用

J2EE アプリケーション運用機能として提供される機能のうち，Web アプリケーションの通常停止は，サーブレット／JSP およびユーザスレッドで使用できます。ただし，コンテナでは，ユーザスレッドを停止しません。

(5)　コンテナ拡張ライブラリ

コンテナ拡張ライブラリは，サーブレット／JSP およびユーザスレッドのどちらでも使用できます。

3.4.12　ドメイン名指定でのトップページの表示

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

ドメイン名指定でのトップページの表示は，J2EE サーバモードと同じです。ドメイン名指定でのトップページの表示については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.9　ドメイン名指定でのトップページの表示」を参照してください。

ただし，サーブレットエンジンモードの場合は，Web アプリケーションのコンテキストルートを「ROOT」にしてデプロイする必要があります。

サーブレットエンジンモードの場合にドメイン名指定でのトップページの表示をするための設定については，「3.7.7　Web アプリケーションの設定」を参照してください。

3.4.13　静的コンテンツのキャッシュ

一度アクセスした静的コンテンツをメモリ上にキャッシュし，2 回目以降のアクセスではキャッシュからブラウザにレスポンスを返すことで，静的コンテンツのレスポンスタイムを短縮できます。

静的コンテンツのキャッシュは，J2EE サーバモードの場合と同じです。静的コンテンツのキャッシュについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.21　静的コンテンツのキャッシュ」を参照してください。

なお，サーブレットエンジンモードの場合，リロード機能または JSP ファイルの再コンパイル機能が有効なときには，静的コンテンツのキャッシュは無効になります。

3.4.14　簡易 Web サーバ機能

サーブレットエンジンモードの場合，簡易 Web サーバは，Web アプリケーションの動作確認用に利用できます。

! 注意事項

簡易 Web サーバは，本番運用時にサーブレット，JSP を実行する Web サーバとしては利用できませんので注意してください。

簡易 Web サーバを利用するための設定は，Web コンテナサーバのプロパティをカスタマイズして設定します。設定については，「3.7.4　Web コンテナサーバの動作設定のカスタマイズ」を参照してください。

3.5　サーブレットエンジンモードで動作するアプリケーションの実装

ここでは，サーブレットエンジンモードで動作するアプリケーションの実装時に考慮する必要があることについて説明します。

3.5.1　利用できるアプリケーションサーバの API の種類

サーブレットエンジンモードでは，アプリケーションサーバが提供する API のうち，次の API を使用できます。

- 統合ユーザ管理フレームワークで使用する API
- 統合ユーザ管理フレームワークで使用するタグライブラリ
- ユーザログ機能で使用する API
- 性能解析トレースで使用する API
- JavaVM で使用する API
- DABroker Library で使用する API

これらの API のうち，統合ユーザ管理フレームワークに関する API およびタグライブラリについては，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を参照してください。それ以外の API については，マニュアル「アプリケーションサーバ リファレンス API 編」を参照してください。

3.5.2　web.xml 編集時の注意事項

ここでは，サーブレットエンジンモードの場合の，Web アプリケーション作成時の注意事項について説明します。

(1)　サーブレットエンジンモードの場合

サーブレットエンジンモードの場合で，web.xml を編集するときには次の内容に注意してください。

なお，これ以外の注意事項については，1.4 モードとの差異はありません。このほかの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.4　web.xml 編集時の注意事項」を参照してください。

- **セキュリティロール使用時の設定について**

  web.xml の<security-role>タグを使用する場合，実行環境の web-users.xml 上で<security-role>の<role-name>と同じ名称でロールを定義してください。

- **web.xml の記述内容とサーバの動作**

  Servlet 2.3 に対応したアプリケーションの場合で，サーブレットエンジンモードで起動したときの，web.xml の記述内容とサーバの動作を次の表に示します。

  表 3-20　web.xml の記述内容とサーブレットエンジンモードの場合のサーバの動作

web.xml の記述内容	サーバの動作
プロパティに webserver.xml.validate=false を設定している場合で，DTD の仕様で必須となっているタグを記述しないとき	無視して実行されるか，リクエスト実行時にエラーとなります。 必須のタグは，必ず記述してください。

web.xml の記述内容	サーバの動作
<filter>タグに含まれる<filter-name>タグ要素を空要素にし，<filter-mapping>タグに含まれる<filter-name>タグ要素も空要素にした場合	指定されたフィルタが実行されます。
<listener>タグに含まれる<listener-class>タグ要素を空要素にした場合	サーバ起動時に該当する Web アプリケーションが無効となります。
プロパティに webserver.xml.validate=false を設定している場合で，DTD の仕様で一つだけ指定が許されているタグを複数記述したとき	後ろに記述された値を使用して正常に動作することがありますが，一つだけ指定が許されているタグを複数記述しないでください。
要素が同じ<filter-mapping>タグを複数記述した場合	記述された回数分の doFilter メソッドが呼び出されます。
親タグと，その下に要素としてキーとなるタグとそのほかの情報を持つタグを複数記述した場合で，キーとなるタグの要素が同じでそのほかの情報が異なるとき	あとに記述された親タグを有効にします。
プロパティに webserver.xml.validate=false を設定している場合で，DTD の仕様で規定されていない順序でタグを記述した場合	正常に動作することがありますが，規定されている順序でタグを記述するようにしてください。
<jsp-file>タグに「/」で始まらない文字列を記述した場合	サーバ起動時にエラーになります。
次に示す<url-pattern>タグに，「*.」以外の「/」で始まらない文字列を記述した場合 • <servlet-mapping>-<url-pattern> • <security-constraint>-<web-resource-collection>-<url-pattern>	サーバ起動時にエラーになります。
<mime-mapping>タグに含まれる<extension>タグ要素に空文字を記述した場合	設定を無視して起動されます。
<mime-mapping>タグに含まれる<mime-type>タグ要素に空文字を記述した場合	正常に動作します。
<error-page>タグに含まれる<error-code>タグ要素に空文字を記述した場合	設定を無視して起動されます。
<transport-guarantee>タグに空文字を記述した場合	「CONFIDENTIAL」として動作します。
<form-login-page>タグまたは<form-error-page>タグに空文字を記述した場合	サーバ起動時にエラーになります。
<env-entry-type>タグ，<ejb-ref-type>タグ，または<ejb-ref-type>タグに空文字を設定した場合	正常に起動します。
<security-constraint>を設定して<auth-method>を省略した場合	サーバ起動時にエラーになります。

- **サーブレットエンジンモードで使用できないタグ**

  web.xml に記述する次のタグは，JNDI 名前空間に登録するため，サーブレットエンジンモードでは使用できません。J2EE サーバモードで有効になります。

- env-entry
- ejb-ref
- ejb-local-ref
- resource-ref
- resource-env-ref

- **Servlet 2.4 以降の仕様でサポートされない web.xml の要素**

  アプリケーションサーバでは，次に示す Servlet 2.4 以降の仕様の web.xml の要素は定義できません。定義した場合，デプロイ時にエラーになります。

  - message-destination
  - message-destination-ref
  - service-ref

  また，サーブレットエンジンモードの場合，次のプロパティの定義で web.xml およびタグライブラリ・ディスクリプタ（TLD ファイル）のチェックを無効化すると，Servlet 2.3 仕様の web.xml に Servlet 2.4 以降の仕様の要素を記述してもデプロイ時にエラーは通知されません。無効な要素として無視されます。

  webserver.xml.validate=false

  なお，Servlet 2.4 以降の仕様で追加，変更された仕様を，アプリケーションサーバ上で使用する場合の，上記以外の注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.5　Servlet 2.4 以降で追加，変更された仕様についての注意事項（web.xml）」を参照してください。

3.5.3　そのほかの注意事項

(1)　データベースとアプリケーション間の文字コード変換

サーブレットエンジンモードの場合，JDBC コネクションを取得するとき，ENCODELANG キーに該当値を設定したプロパティを作成し，JDBC ドライバの java.sql.Driver.connect メソッドまたは java.sql.DriverManager.getConnection メソッドの引数に渡してください。

(2)　アプリケーションサーバ独自の例外の扱い

HttpSession オブジェクト数の上限が設定できないサーブレットエンジンモードでは，HttpSessionLimitExceededException クラスは利用できません。

この例外クラスは，HttpSession オブジェクト数の上限が設定できる，J2EE サーバモードの場合に利用できます。

3.6　サーブレットエンジンモードでのリソース接続

ここでは，サーブレットエンジンモードでのリソース接続について説明します。

3.6.1　接続できるリソースとトランザクションの管理方法

サーブレットエンジンモードの場合に接続できるリソースの種類とトランザクションの管理方法について説明します。

(1)　接続できるリソース

サーブレットエンジンモードで接続できるリソースについて次の表に示します。

表 3-21　接続できるリソース（サーブレットエンジンモード）

リソースの種類		サーブレットエンジンモード
データベース	HiRDB	○
	Oracle	○
	SQL Server	×
	XDM/RD E2	×
データベース上のキュー	HiRDB	×
	Oracle	×
OpenTP1	SPP	×
	TP1/Message Queue	×
SMTP サーバ		×
JavaBeans リソース		×

（凡例）○：接続できる　×：接続できない

参考

リソースアダプタが Non-Managed 環境での使用に対応している場合には，接続できる場合があります。ただし，アプリケーションサーバによるコネクションプーリングやトランザクション管理は利用できません。

(2)　トランザクションの管理方法

サーブレットエンジンモードの場合，トランザクションはアプリケーションサーバで管理されません。

3.6.2　データベース接続の前提条件

サーブレットエンジンモードでデータベースに接続する場合の前提条件を，データベースの種類ごとに示します。

サーブレットエンジンモードの場合，データベース接続には，Web コンテナコネクションプール機能を使用します。

なお，サーブレットエンジンモードでは，次のデータベースには接続できません。

- Oracle10g 以降
- SQL Server
- XDM/RD E2

(1)　HiRDB の場合

- **使用できる JDBC ドライバ**
  使用できる JDBC ドライバは DABroker Library となります。
  DABroker Library を使用する場合，HiRDB クライアントをインストールする必要があります。
- **接続方法**
  Web コンテナコネクションプール機能を使用して接続します。
  なお，サーブレットエンジンモードの場合，トランザクションはアプリケーションサーバで管理されません。

(2)　Oracle9i の場合

- **使用できる JDBC ドライバ**
  使用できる JDBC ドライバは DABroker Library となります。Oracle JDBC Thin Driver では接続できません。
  JDBC ドライバとして DABroker Library を使用する場合，Oracle クライアントをインストールする必要があります。
- **接続方法**
  Web コンテナコネクションプール機能を使用して接続します。
  なお，サーブレットエンジンモードの場合，トランザクションはアプリケーションサーバで管理されません。

3.6.3　Web コンテナコネクションプール機能によるデータベース接続

サーブレットエンジンモード上で動作する Web アプリケーションからデータベースへ接続する場合，Web コンテナコネクションプーリング機能を利用してデータベースへの接続をプールできます。これによって，コネクション生成時のパフォーマンスを向上できます。

Web コンテナコネクションプール機能での，コネクションの取得方法には次の 2 とおりがあります。

1. java.sql.DriverManager.getDriver を使って java.sql.Driver のインスタンスを取得し，java.sql.Driver.connect を使ってコネクションを取得する方法
2. java.sql.DriverManager.getConnection を使ってコネクションを取得する方法

なお，コネクションプールの詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.1　コネクションプーリング」を参照してください。

(1)　Web コンテナコネクションプール機能の利用方法

Web コンテナコネクションプール機能を利用するためには，管理機能用の Web アプリケーションをデプロイし，プール管理情報を登録する必要があります。プール管理情報の登録をするためのコネクションプーリングの設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.1　コネクションプーリング」を参照してください。

また，コネクションの取得処理は Web アプリケーションで実施するため，ユーザ作成のアプリケーションに，コネクションを操作するための記述が必要になります。

コネクションを取得するには，次の 2 とおりの方法があります。同時に多数のユーザが接続するようなシステムの場合は，ドライバマネージャ内で待ちが発生するため，取得方法 1 の記述をお勧めします。

取得方法 1

```
String className = "com.hitachi.software.web.jdbc.WccpDriver";
String url = "jdbc:hitachi:wccp://DBID=<データベースID>";

Properties info = new Properties();
info.put("user", "<ユーザID>");
info.put("password", "<パスワード>");

Class.forName(className);
Driver drv = DriverManager.getDriver(url);
Connection con = drv.connect(url, info);
con.close();
```

取得方法 2

```
String className = "com.hitachi.software.web.jdbc.WccpDriver";
String url = "jdbc:hitachi:wccp://DBID=<データベースID>";
String user = "<ユーザID>";
String password = "<パスワード>";

Class.forName(className);
Connection con = DriverManager.getConnection(url, user, password);
con.close();
```

(2)　パフォーマンスチューニングのための機能

サーブレットエンジンモードでデータベースに接続する場合に，パフォーマンスチューニングのために使用できる機能を，次の表に示します。

表 3-22　サーブレットエンジンモードでデータベースに接続する場合に使用できるパフォーマンスチューニングのための機能

機能	使用可否
コネクションプーリング	○
コネクションシェアリング	×
コネクションのアソシエーション	×
ステートメントプーリング	×
ライトトランザクション	×
インプロセストランザクションサービス	×
DataSource オブジェクトのキャッシング	×
DB Connector のコンテナ管理でのサインオンの最適化	×
受信バッファのプーリング	×

（凡例）○：使用できる　×：使用できない

サーブレットエンジンモードの場合のコネクションプーリングについて次に説明します。

コネクションプーリング

サーブレットエンジンモードの場合，プーリングの基本機能として，コネクションプールの生成初期化，コネクションの取得や解放などの機能を提供しています。

- コネクションプールの生成および初期化

  コネクションプールが生成・初期化されるタイミングは，リソースのスタート処理時または監視・制御機能で提供する API のクラスのインスタンスの生成時です。ただし，この時点ではコネクションは生成されません。

  コネクションが最初に生成されるのは，最初のコネクション取得要求があったときです。以降，プールの最小値までコネクションをプールします。この動作は，非同期で実施されます。

- コネクションの取得と解放

  コネクションの取得要求があると，プールから未使用のコネクションを選択し，コネクションの状態を使用中にしてから，クライアントに渡します。

  また，コネクションの解放要求があると，使用中になっているコネクションの状態を未使用に設定して，コネクションをプールに返します。なお，コネクションの状態が，再利用できない状態に設定されている場合は，データベースとの接続を切断して，コネクションを削除します。この場合，コネクションはプールに返されません。

- 例外が発生したコネクションの破棄

  java.sql.Connection のような JDBC インタフェースの API で SQLException 例外が発生すると，コネクションクローズで該当コネクションの接続が維持されていても，コネクションプールに戻さないで破棄します。その結果，Web コンテナコネクションプーリングを使用している場合（java.sql.Connection インタフェースだけ該当する）に，余分なコネクション生成が発生して性能に影響を与えることがあります。

(3)　フォールトトレランスのための機能

サーブレットエンジンモードでデータベースに接続する場合に，フォールトトレランスのために使用できる機能を，次の表に示します。

表 3-23　サーブレットエンジンモードでデータベースに接続する場合に使用できるフォールトトレランスのための機能

機能	使用可否
コネクションの障害検知	×
コネクション枯渇時のコネクション取得待ち	×
コネクションの取得リトライ※	○
コネクションプールの情報表示	○
コネクションプールのクリア	○
コネクションの強制クローズ	○
コネクションの自動クローズ	×
コネクションスイーパ	×
トランザクションタイムアウト	×
トランザクションリカバリ	×
障害調査用 SQL の出力	×

機能	使用可否
クラスタコネクションプール	×

（凡例）○：使用できる　×：使用できないまたは該当しない

注※　Web コンテナコネクションプールのコネクション取得リトライができます。

以降，サーブレットエンジンモードで使用できるフォールトトレランスのための機能について説明します。

なお，それぞれの機能の概要および目的については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15　フォールトトレランスのための機能」を参照してください。ここでは，サーブレットエンジンモードで特有の説明についてだけ説明します。

コネクションの取得リトライ

J2EE サーバモードとの機能差はありません。マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.15.3　コネクションの取得リトライ」を参照してください。

コネクションプールの情報表示

コネクションプール内のコネクションの情報を，コマンドを使用して表示できます。コネクションプール情報の表示には，cjweblistpool コマンドを使用します。コマンドの詳細については，「cjweblistpool（プール管理情報の一覧表示)」を参照してください。

コネクションプールのクリア

データベースサーバのダウンなどによって，コネクションが切断された場合，プールされているコネクションはコネクションプール内に残ってしまいます。これらのコネクションは，cjwebclearpool コマンドによって削除できます。コマンドの詳細については，「cjwebclearpool（プール内のコネクションのクリア)」を参照してください。

コネクションの強制クローズ

アプリケーションの Connection.close メソッドの記述もれによってクローズされないコネクションは，コネクションプール内で使用中状態のまま再利用されません。これらのコネクションは，cjwebclosepool コマンドを使用して強制クローズできます。コマンドの詳細については，「cjwebclosepool（プール内のコネクションの強制切断)」を参照してください。

なお，ユーザスレッドでは使用できません。

3.7　サーブレットエンジンモードの利用方法

互換用のサーバの動作モードとして，Web コンテナだけを使用するサーブレットエンジンモードがあります。ここでは，サーブレットエンジンモードの利用について説明します。

なお，旧バージョンでサーブレットエンジンモードを使用していた場合には，J2EE サーバモードに移行することをお勧めします。

3.7.1　サーブレットエンジンモードでのシステム設計

ここでは，システム構成の設計およびパフォーマンスチューニングについて説明します。

(1)　システム構成の設計

ここでは，サーブレットエンジンモードで使用できるシステム構成について説明します。

(a)　システム設計の準備

システム設計作業を始める前に，次のことを明確にする必要があります。

- アプリケーションおよびプロセスの構成
- 運用方法

● アプリケーションおよびプロセスの構成

サーブレットエンジンモードでは，サーブレットと JSP だけで構成されるアプリケーション（Web アプリケーション）が使用できます。

また，サーブレットエンジンモードでアプリケーションサーバを動作させるシステムでは，次に示すプロセスが必要となります。

- Web サーバ※
- Web コンテナサーバ
- PRF デーモン

注※
　アプリケーションサーバに含まれる Web サーバは，HTTP Server です。

これらのプロセスは，アプリケーションサーバによって提供され，アプリケーションサーバをインストールしたマシンで起動できます。

サーブレットエンジンモードの場合，Web クライアントからのリクエストを，リダレクタと連携した Web サーバ（Web サーバ連携）を利用して処理できます。利用できる Web サーバは，HTTP Server または Microsoft IIS です。

● 運用方法

サーブレットエンジンモードでは，Management Server を利用した運用，およびクラスタソフトウェアと連携したシステムの運用はできません。サーブレットエンジンモードで運用する場合，システムの構築には，アプリケーションサーバが提供するコマンドと定義ファイルを使用します。

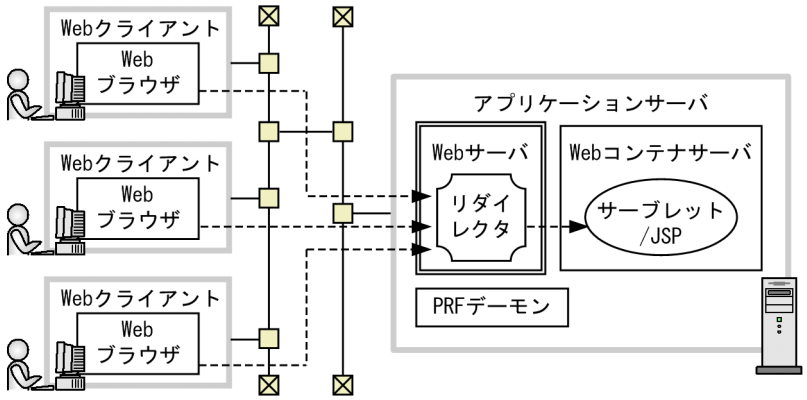
(b)　サーブレットエンジンモードのシステムの構成

サーブレットエンジンモードを使用して，アプリケーションサーバのシステムを設計する場合，Web コンテナサーバを使用した構成になります。Web コンテナサーバは，サーブレットエンジンモードで動作しているアプリケーションサーバです。

システム構成の特徴

Web コンテナサーバの構成の例を次の図に示します。

図 3-3　Web コンテナサーバの構成の例



（凡例）

----►　：リクエストの流れ

これ以外の凡例については，マニュアル「アプリケーションサーバ システム設計ガイド」の「3.2　システム構成の説明について」を参照してください。

特徴

- サーブレットと JSP だけが動作します。
- サーブレットと JSP を対象にした，負荷分散機またはリダイレクタを使用した負荷分散構成，および Web サーバとアプリケーションサーバを分離した構成が実現できます。
- Management Server を利用した運用はできません。

リクエストの流れ

サーブレットと JSP に対するクライアントからのリクエストは，Web ブラウザから Web サーバを経由して送られます。

それぞれのマシンで起動するプロセス

それぞれのマシンに必要なソフトウェアとプロセスについて説明します。

アプリケーションサーバマシン

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

起動するプロセスは次のとおりです。

- Web サーバ
- Web コンテナサーバ
- PRF デーモン

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

Web クライアントマシン

Web クライアントマシンには，Web ブラウザが必要です。

(2)　パフォーマンスチューニング

ここでは，サーブレットエンジンモードで使用できるアプリケーションのチューニング項目について説明します。サーブレットエンジンモードでは，アプリケーションがサーブレットと JSP だけで構成される Web アプリケーションが使用できます。サーブレットエンジンモードで使用できるアプリケーションのチューニング項目を次の表に示します。

表 3-24　サーブレットエンジンモードで使用できるアプリケーションのチューニング項目

チューニング項目	利用できる機能	参照先マニュアル	参照先
同時実行数の最適化	Web アプリケーションでの同時実行スレッド数制御（Web コンテナ単位）	アプリケーションサーバ システム設計ガイド	8.3.4
データベースアクセス方法の最適化	Web コンテナコネクションプーリング	このマニュアル	(a)
タイムアウトの設定	Web フロントシステムでのタイムアウトの設定	アプリケーションサーバ システム設計ガイド	8.6.2
Web アプリケーションの動作の最適化	静的コンテンツと Web アプリケーションの配置の切り分け		8.7.1
	静的コンテンツのキャッシュ		8.7.2
	リダイレクタによるリクエストの振り分け（Web サーバ連携の場合）		8.7.3

これらのチューニング項目のうち，サーブレットエンジンモードで動作する場合に異なる機能について説明します。

(a)　Web コンテナコネクションプーリングを使用する

サーブレットエンジンモード上で動作する Web アプリケーションからデータベースへ接続する場合，Web コンテナコネクションプールを使用します。このとき，Web コンテナコネクションプーリング機能を利用してデータベースへの接続をプールできます。これによって，コネクション生成時のパフォーマンスを向上できます。

Web コンテナコネクションプーリングを使用する場合，最初のコネクション取得要求があったときに，プール数の最小値までコネクションが生成され，プールされます。クライアントからコネクション取得要求があると，プールから未使用のコネクションが選択され，クライアントに渡されます。コネクションの解放要求があると，使用中になっているコネクションの状態が未使用に設定され，コネクションがプールに返されます。なお，コネクションの状態が，再利用できない状態に設定されている場合は，データベースとの接続が切断され，コネクションが削除されます。この場合，コネクションはプールに返されません。

Web コンテナコネクションプーリングを使用する場合は，チューニングパラメタを設定する必要があります。Web コンテナコネクションプーリングのチューニングパラメタは，プール管理情報設定ファイルに設定します。設定した内容は，cjwebeditpool コマンドで登録します。プール管理情報設定ファイルについては，「3.11.6　プール管理情報設定ファイル」を，cjwebeditpool コマンドについては，「cjwebeditpool (プール管理情報の登録と変更)」を参照してください。

Web コンテナコネクションプーリングのチューニングパラメタを次の表に示します。

表 3-25　Web コンテナコネクションプーリングのチューニングパラメタ

設定項目	設定個所
プールするコネクションの最小値	<MinimumSize>タグ
プールするコネクションの最大値	<MaximumSize>タグ
コネクションを取得できなかった場合にリトライするまでのアプリケーションの待ち時間	<RetryInterval>タグ
コネクションを取得できなかった場合のリトライ回数	<RetryCount>タグ

3.7.2　サーブレットエンジンモードでのシステム構築

サーブレットエンジンモードを使用する場合のインストール，システム構築の流れ，システム構築で使用するファイルについて説明します。なお，旧バージョンでサーブレットエンジンモードを使用していた場合の移行情報については，「3.9　サーブレットエンジンモードの場合の移行」を参照してください。

(1)　インストールと初期設定

サーブレットエンジンモードの場合，インストールと初期設定は，J2EE サーバの場合と同じです。J2EE サーバの場合のインストールと初期設定の概要およびインストール後のディレクトリ構成については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。なお，サーブレットエンジンモードの場合には，次のように読み替えてください。

- J2EE サーバ→Web コンテナサーバ
- cjsetup コマンド→cjwebsetup コマンド
- cjstartsv コマンド→cjstartweb コマンド
- cjstopsv コマンド→cjstopweb コマンド
- cjdumpsv コマンド→cjdumpweb コマンド

ここでは，J2EE サーバモードの場合と異なる内容について説明します。

Web コンテナサーバの場合の，インストール方法ごとのユーザ環境の扱いを次の表に示します。

表 3-26　インストール方法ごとのユーザ環境の扱い（Web コンテナサーバの場合）

インストール方法	Web コンテナサーバのユーザ定義	Web アプリケーション配置ディレクトリ	リダイレクタのユーザ定義
新規インストール	インストーラが，デフォルトの Web コンテナサーバをセットアップし，ユーザ定義を作成します。	インストーラが，デフォルトの Web コンテナサーバをセットアップし，Web アプリケーション配置ディレクトリを作成します。	インストーラが，デフォルトの定義を作成します。
上書きインストール	引き継ぎます。	引き継ぎます。	引き継ぎます。
旧バージョンからのアップグレードインストール	引き継ぎます。	引き継ぎます。	引き継ぎます。

(a)　インストールとアンインストール

Component Container を上書きインストールすると，前の設定が自動的に引き継がれます。なお，デフォルトの Web コンテナサーバ（ホスト名称のサーバ）のディレクトリ（<製品のインストールディレクトリ>¥CC¥web¥containers¥<ホスト名称>，または/opt/Cosminexus/CC/web/containers/<ホスト名称>）がない場合には，サーバが自動的にセットアップされます。

(b)　環境変数の設定

サーブレットエンジンモードの場合，環境変数の設定は，J2EE サーバモードで，CTM およびトランザクションサービスの両方を使用しないときの設定と同じです。詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「付録 H　システムの環境変数」を参照してください。

(c)　見積もりと設定

Web コンテナサーバで使用するリソースの見積もり方法を次に示します。

- スレッド数

  最大スレッド数 ＝ 8 ＋ A ＋ B ＋ C ＋ D ＋ E ＋ F

  （凡例）

  - A：Web サーバの同時接続クライアント数（複数の Web サーバがある場合は合計）

    リクエスト処理中に Web サーバが再起動された場合，再起動後に多くのリクエストを受けると，この値を超えるスレッドが生成される場合がある
  - B：簡易 Web サーバへの同時接続クライアント数（ただし，簡易 Web サーバへの同時接続クライアント数が 5 以下の場合は 5，100 以上の場合は 100 を指定する）
  - C：デプロイ済みの Web アプリケーションの数
  - D：簡易 Web サーバを有効にしている場合は 1，無効にしている場合は 0
  - E：Web アプリケーションのリロード機能を使用する Web アプリケーション数（ただし，usrconf.properties の webserver.context.reloadable キーに true を指定している場合は，Web アプリケーションのリロード機能を使用する Web アプリケーション数を指定する）
  - F：JSP ファイルの再コンパイル機能を使用する Web アプリケーション数（ただし，usrconf.properties の webserver.jsp.recompilable キーに true を指定している場合は，JSP ファイルの再コンパイル機能を使用する Web アプリケーション数を指定する）
- ファイルディスクリプタ数

  最大ファイルディスクリプタ数 ＝ （55 ＋ A ＋ B ＋ C ＋ D×2 ＋ E ＋ F）／ 0.8

  （凡例）

  - A：Web サーバの同時接続クライアント数（ただし，Web サーバの同時接続クライアント数が Web コンテナの最大同時実行スレッド数（usrconf.properties の webserver.connector.ajp13.max_threads キーに指定した値）より大きい場合は，Web サーバの同時接続クライアント数+1 を指定する）
  - B：Web コンテナの最大同時実行スレッド数
  - C：簡易 Web サーバを有効にしている場合は 1，無効にしている場合は 0
  - D：簡易 Web サーバへの同時接続クライアント数
  - E：Web アプリケーションに含む JAR ファイルの数
  - F：usrconf.cfg の add.class.path キーに指定した JAR ファイルの数

(d)　作業ディレクトリ

Web コンテナサーバの作業ディレクトリについては，「3.7.8　Web コンテナサーバの作業ディレクトリ」を参照してください。

(2)　システム構築の流れ

サーブレットエンジンモードでのシステム構築の流れを次の図に示します。

図 3-4　システム構築の流れ（サーブレットエンジンモードの場合）

注

それぞれの参照先については，次の表に示した個所を参照してください。

なお，図中のマニュアル名は略称です。

項番	参照先マニュアル	参照個所
1.	このマニュアル	3.7.3

項番	参照先マニュアル	参照個所
2.	アプリケーションサーバ 機能解説 セキュリティ管理機能編	7.2.6
	HTTP Server	全体
3.	このマニュアル	3.7.4
4.	このマニュアル	3.7.5
5.	アプリケーションサーバ 機能解説 基本・開発編（Web コンテナ）	4 章
6.	アプリケーションサーバ 機能解説 セキュリティ管理機能編	5 章
7.	このマニュアル	3.7.7
8.	アプリケーションサーバ 機能解説 運用／監視／連携編	2 章

図中の 1.～8.について説明します。

1. **サーバをセットアップします。**

   デフォルトの設定のままで利用する場合は，セットアップ不要です。インストール時にセットアップ済みの Web コンテナサーバとは別のサーバ名称を使用したい場合，Web コンテナサーバをセットアップしてください。

2. **Web サーバの設定をします。**

   使用する Web サーバとして，HTTP Server，または Microsoft IIS（Windows の場合）の設定をします。

3. **サーバをカスタマイズします。**

   デフォルトの設定のままで利用する場合は，カスタマイズ不要です。Web コンテナサーバが使用するネーミングサービス，JavaVM の起動オプションなどを変更したい場合や，コンテナ拡張ライブラリ，コネクションプーリングなど特定の機能を使用したい場合には，アプリケーションサーバが提供するユーザ定義ファイルやコマンドを使用して，Web コンテナサーバの動作を設定，カスタマイズします。

4. **トラブルシューティングの資料取得の設定をします。**

   デフォルトの設定のままで利用する場合は，設定不要です。ただし，OS の統計情報，ユーザダンプ（Windows の場合）または core ダンプ（UNIX の場合），JavaVM の GC のログは，事前に取得のための設定をしておかないと，取得できません。これらの資料はトラブルシューティングで必要となるため，取得することをお勧めします。

   これらの資料を取得する場合や，ほかの資料の取得設定を変更したい場合に，トラブルシューティングの資料取得の設定をしてください。

5. **Web サーバとの連携の設定をします。**

   Web サーバと連携しない場合は，設定不要です。リダイレクタを利用したリクエストの振り分け，通信タイムアウト，エラーページのカスタマイズなど，Web サーバと連携するための設定をします。

6. **統合ユーザ管理機能の設定をします。**

   統合ユーザ管理機能を使用しない場合は，設定不要です。アプリケーションサーバのシステムにログインするユーザを統合管理するための設定をします。ユーザ認証情報を格納する LDAP ディレクトリサーバや，統合ユーザ管理のコンフィグレーションファイルを設定します。

7. **Web アプリケーションの設定をします。**

   WAR ファイルを Web コンテナサーバにデプロイします。

8. **システムを起動して，システムの動作を確認します。**

参考

運用支援機能を持つプログラムを利用することで，システムで運用操作の一括実行や自動実行などの運用方法を実現することもできます。この場合には，図 3-4 の環境構築に加えて，運用支援機能を持つプログラムの設定が必要になります。運用支援機能を持つほかのプログラムを使用する場合には，使用しているプログラムのマニュアルなどを参照して，環境を構築してください。

(3)　システム構築で使用するファイル

ここでは，システムの構築で使用するファイルのうち，主なファイルについて説明します。ここで説明していないファイルについては，システム構築の手順の中で適宜説明します。

! 注意事項

ファイルを変更する場合は，Web コンテナサーバを停止してください。Web コンテナサーバの稼働中にファイルの内容を変更した場合の動作は保証されません。

- usrconf.properties（ユーザプロパティファイル）

  Web コンテナサーバのユーザプロパティを定義するファイルです。

  ファイルの格納場所を次に示します。

  - Windows の場合

    <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.properties

  - UNIX の場合

    /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.properties

- usrconf.cfg（オプション定義ファイル）

  Web コンテナサーバの JavaVM の起動オプションを定義するファイルです。JavaVM メモリ空間のサイズや割合などを指定する JavaVM オプションは，このファイルの add.jvm.arg キーに指定します。

  ファイルの格納場所を次に示します。

  - Windows の場合

    <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.cfg

  - UNIX の場合

    /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.cfg

- web.policy（セキュリティポリシーファイル）

  Web コンテナサーバのセキュリティポリシーを定義するファイルです。

  ファイルの格納場所を次に示します。

  - Windows の場合

    <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥web.policy

  - UNIX の場合

    /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/web.policy

- hitachi_web.properties（Web アプリケーションプロパティファイル）

  Web アプリケーションごとのプロパティを Web コンテナサーバのプロパティに指定するファイルです。

  ファイルの格納場所を次に示します。

  - Windows の場合

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

- UNIX の場合
  <Web アプリケーションの WEB-INF ディレクトリ>/hitachi_web.properties

hitachi_web.properties を変更した場合，WAR ファイルを作成，インポートし直してください。

- **web-users.xml（ユーザ認証情報定義ファイル）**
  ユーザ名，パスワード，所属するロールなどのユーザ認証情報を定義するファイルです。
  ファイルの格納場所を次に示します。
  - Windows の場合
    <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥web-users.xml
  - UNIX の場合
    /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/web-users.xml

3.7.3　Web コンテナサーバのセットアップ

Web コンテナサーバは，製品のインストール時にセットアップ済みです。デフォルトの設定では，Web コンテナサーバ名はホスト名です。デフォルトの設定を変更したい場合には，Web コンテナサーバをセットアップしてください。また，同じホスト内で複数の Web コンテナサーバを起動する場合には，それぞれの Web コンテナサーバに別のサーバ名を割り当てる必要があるため，Web コンテナサーバをセットアップしてください。

Web コンテナサーバはサーバ名称によって識別されます。各コマンドの引数にサーバ名称を指定することで，複数の Web コンテナを開始したり，管理したりできます。また，Web コンテナサーバのサーバ名称は，J2EE サーバの名称とは独立して管理されます。

Web コンテナサーバでは，次に示すファイル，ディレクトリがサーバ名称ごとに独立に管理されます。

- ユーザ定義ファイル
- Web アプリケーション配置ディレクトリ
- ログファイル格納ディレクトリ
- 一時ファイル格納ディレクトリ

インストール時には，インストールされたマシンのホスト名がデフォルトのサーバ名称として設定されます。新たにサーバ名称を設定するには，cjwebsetup コマンドを実行します。なお，UNIX の場合，コマンドの実行には，root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。

- Windows の場合

```
"<製品のインストールディレクトリ>¥CC¥web¥bin¥cjwebsetup サーバ名称"
```

- UNIX の場合

```
# /opt/Cosminexus/CC/web/bin/cjwebsetup サーバ名称
```

なお，同一マシン内では，同じサーバ名称で cjwebsetup コマンドを複数回実行してはいけません。実行した場合エラーになります。Web コンテナサーバを再セットアップする場合，cjwebsetup -d <サーバ名称>でサーバの環境をいったん削除してから再度セットアップコマンドを実行してください。cjwebsetup コマンドについては，「cjwebsetup（Web コンテナサーバのセットアップとアンセットアップ)」を参照してください。

同一マシンで複数の Web コンテナを同時に起動する場合，使用するポート番号が重ならないように，usrconf.properties のユーザ定義を正しく設定する必要があります。また，webserver.connector.ajp13.port キーで設定されているポート番号に対し，同時に起動する Web コンテナ間で異なる値を使用するように設定してください。webserver.connector.http.port キーについても，同様に異なるポート番号を設定するか，または webserver.connector.http.enabled キーに false を設定することで，ポート番号を使用しないようにする必要があります。さらに，webserver.shutdown.port キーについても，ポート番号が重ならないように注意する必要があります。

3.7.4　Web コンテナサーバの動作設定のカスタマイズ

ここでは，Web コンテナサーバの動作設定のカスタマイズについて説明します。

デフォルトで設定されている Web コンテナサーバの動作設定を変更したい場合には，ユーザ定義ファイルを編集して，Web コンテナサーバの動作をカスタマイズする必要があります。例えば，Web コンテナサーバが使用するコンテナ，ログなどの設定を変更したい場合には，usrconf.properties を編集してください。また，Web コンテナサーバの JavaVM の起動オプションなどを変更したい場合には，usrconf.cfg を編集してください。

(1)　Web コンテナサーバの動作設定のカスタマイズの方法

Web コンテナサーバの動作設定のカスタマイズでは，ユーザ定義ファイルを編集して，サーバ名で識別される Web コンテナサーバ単位に，Web コンテナサーバの動作をカスタマイズします。

(a)　カスタマイズの前提

カスタマイズ対象の Web コンテナサーバのセットアップが完了していることが前提になります。また，UNIX の場合，カスタマイズには root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。

(b)　カスタマイズ対象のユーザ定義ファイル

次の五つのユーザ定義ファイルをテキストエディタなどで編集して，Web コンテナサーバの動作をカスタマイズします。

- usrconf.properties（ユーザプロパティファイル）
- usrconf.cfg（オプション定義ファイル）
- web.policy（セキュリティポリシーファイル）
- hitachi_web.properties（Web アプリケーションプロパティファイル）
- web-users.xml（ユーザ認証情報定義ファイル）

これらのユーザ定義ファイルの詳細については，次の個所を参照してください。

- 3.11.3　usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル）
- 3.11.2　usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）
- 3.11.5　web.policy（SecurityManager 定義ファイル）
- 3.11.1　hitachi_web.properties（Web コンテナサーバ用 Web アプリケーションプロパティファイル）
- 3.11.4　web-users.xml（ユーザ認証情報定義ファイル）

(c)　Web コンテナサーバのカスタマイズで設定できる主な項目

Web コンテナサーバのカスタマイズでユーザ定義ファイルを編集して設定できる項目のうち，主な項目とその項目を設定するユーザ定義ファイルおよびキーを次の表に示します。キーの詳細，およびここで説明していないキーについては，次の個所を参照してください。

- 3.11　Web コンテナサーバで使用するファイル
- 3.11.1　hitachi_web.properties（Web コンテナサーバ用 Web アプリケーションプロパティファイル）
- 3.11.2　usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）
- 3.11.3　usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル）
- 3.11.4　web-users.xml（ユーザ認証情報定義ファイル）
- 3.11.5　web.policy（SecurityManager 定義ファイル）

表 3-27　ユーザ定義ファイルのカスタマイズで設定できる主な項目（サーブレットエンジンモードの場合）

分類	項目	設定ファイル	設定内容
ネーミングサービス	ネーミングのキャッシング	usrconf.properties	次のキーを指定します。 • ejbserver.jndi.cache キーに，ネーミングでのキャッシングを有効にするかどうかを指定します。 • ejbserver.jndi.cache.interval キーに，キャッシュクリアの間隔を設定します。 • ejbserver.jndi.cache.interval.clear.option キーに，キャッシュクリアの範囲を指定します。
	ネーミングサービスの通信タイムアウト	usrconf.properties	ejbserver.jndi.request.timeout キーに，ネーミングサービスとの通信タイムアウト時間を指定します。
コンテナ	コンテナ拡張ライブラリ	usrconf.cfg	usrconf.cfg で，コンテナ拡張ライブラリ用の JAR ファイルのパスを Web コンテナサーバのシステムクラスパスに追加します。詳細については，「(2)　コンテナ拡張ライブラリの設定」を参照してください。
	RMI-IIOP 通信のタイムアウト	usrconf.properties	ejbserver.rmi.request.timeout キーに，RMI-IIOP 通信のクライアントとサーバ間の通信タイムアウト時間を指定します。
	Web コンテナ単位での同時実行スレッド数の制御	usrconf.properties	webserver.connector.ajp13.max_threads キーに，Web コンテナ単位での同時実行スレッド数を指定します。
	Web コンテナの通信タイムアウト	usrconf.properties	webserver.connector.ajp13.receive_timeout キーに，Web サーバからのリクエスト待ち時間（秒）を設定します。詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.6.5　リクエスト送受信時の通信タイムアウトの設定（Smart

分類	項目	設定ファイル	設定内容
コンテナ	Web コンテナの通信タイムアウト	usrconf.properties	Composer 機能を使用しない場合)」を参照してください。
	JSP ファイルのコンパイル結果の保持	usrconf.properties	次のキーを指定します。 • webserver.work.clean キーに，コンパイル結果を保持するかどうかを指定します。 • webserver.work.directory キーに，コンパイル結果の出力先を指定します。 設定方法については,「(3)　JSP ファイルのコンパイル結果の保持の設定」を参照してください。
	静的コンテンツのキャッシュ	－	サーブレットエンジンモードの場合は，Web アプリケーションの DD（web.xml）を直接編集して静的コンテンツのキャッシュの設定をします。DD（web.xml）を直接編集して，Web コンテナ単位で静的コンテンツのキャッシュ機能を使用するかどうか, キャッシュを許可する静的コンテンツのメモリサイズ, およびファイルサイズの上限値を指定します。静的コンテンツのキャッシュ機能を使用する場合の DD の定義については，マニュアル「アプリケーションサーバ機能解説 基本・開発編(Web コンテナ)」の「2.21　静的コンテンツのキャッシュ」を参照してください。
	アプリケーションの更新検知とリロード	usrconf.properties hitachi_web.properties	Web アプリケーションが更新された場合に検知する機能，および JSP ファイルが更新された場合にリロードする設定を，usrconf.properties または hitachi_web.properties に指定します。設定方法については,「(4)　アプリケーションの更新検知とリロードの設定」を参照してください。
	リモートインタフェースでの通信障害発生時の EJB クライアントの動作	usrconf.properties	Web コンテナサーバが EJB クライアントの場合に, ejbserver.container.rebindpolicy キーに障害発生時の動作を指定します。
	Web コンテナコネクションプーリング機能	プール管理情報設定ファイル	Web アプリケーションからデータベースへ接続する場合，Web コンテナコネクションプーリング機能を利用してデータベースへの接続をプールできます。設定方法については,「(5)　コネクションプーリングの設定」を参照してください。
Web サーバ	通信路の暗号化レベルの要請	usrconf.properties	webserver.connector.redirect_https.port キーに, 転送先の https のポート番号を指定します。 また，通信路の暗号化レベルを要請するには，さらに，通信路の暗号化レベルの要請を DD (WEB-INF/web.xml) 内に指定する必要があります。

分類	項目	設定ファイル	設定内容
Web サーバ	簡易 Web サーバ機能を使用するかどうかの設定	usrconf.properties	webserver.connector.http.enabled キーに，簡易 Web サーバ機能を使用するかどうかを指定します。デフォルトでは使用しない設定になっています。
	簡易 Web サーバへのアクセス許可	usrconf.properties	webserver.connector.http.permitted.hosts キーに，簡易 Web サーバへのアクセスを許可するホストの IP アドレスまたはホスト名を指定します。 デフォルトでは次のように設定されています。 **ポート番号** 8080 同一ホスト内で複数の Web コンテナを開始する場合，各 Web コンテナで使用するポート番号が重複しないように，次に示すユーザ定義用カスタマイズキーで設定してください。 **ドキュメントルート** • Windows の場合 <製品のインストールディレクトリ>¥CC¥server¥public • UNIX の場合 /opt/Cosminexus/CC/server/public
ログ	ログの出力先や出力レベルの変更	usrconf.cfg， usrconf.properties	usrconf.cfg の web.server.log.directory キーに，ログの出力先を指定します。また，usrconf.properties の ejbserver.logger から始まるキーに，ログサイズとログレベルを指定します。詳細については，「3.7.5(2)　Web コンテナサーバのログ取得の設定」を参照してください。
	Web アプリケーションのユーザログ出力	usrconf.properties， web.policy	ejbserver.application から始まるキーに，ログの出力先，ログレベル，ログ面数，使用するフィルタ，フォーマッタなどを指定します。また，web.policy ファイルにセキュリティポリシーを指定します。詳細については，「3.7.5(10)　Web アプリケーションのユーザログ出力の設定」を参照してください。
ユーザスレッドの生成	ユーザスレッド生成のための権限	web.policy	cjstartweb コマンドに-security オプションを指定し，Web コンテナサーバの実行時保護機能を有効にした場合，ユーザがプログラムの中で明示して生成するスレッド（ユーザスレッド）を生成するためには，対象となるサーブレットや JSP にスレッドの生成権限を与える必要があります。 詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.14.2　ユーザスレッド生成のための権限の設定」を参照してください。ただし，サーブレットエンジンモードの場合は，server.policy ファイルではなく web.policy ファイルに設定してください。

（凡例）－：該当しない

(2)　コンテナ拡張ライブラリの設定

コンテナ拡張ライブラリは，Enterprise Bean，サーブレット，JSP が利用する共通のライブラリです。コンテナ拡張ライブラリを利用することで，Enterprise Bean，サーブレット，JSP から共通のライブラリを呼び出せるようになります。

ここでは，コンテナ拡張ライブラリを使用するための設定について説明します。機能の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.　コンテナ拡張ライブラリ」を参照してください。なお，コンテナ拡張ライブラリが JNI を利用する場合は，サーバ起動・停止フック機能を使用します。サーバ起動・停止フック機能の実装については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.4　サーバ起動・停止フック機能」を参照してください。

コンテナ拡張ライブラリを使用するためには，次の設定が必要です。

1. コンテナ拡張ライブラリ用の JAR ファイルを作成します。
   コンテナ拡張ライブラリについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「14.2　コンテナ拡張ライブラリの利用」を参照してください。
2. Web コンテナサーバ用の usrconf.cfg の web.add.class.path キーで，手順 1.で作成した JAR ファイルのパスを Web コンテナサーバのシステムクラスパスに指定します。
3. コンテナ拡張ライブラリから JNI 機能を使用する場合は，Web コンテナサーバ用の usrconf.cfg の add.library.path キーで，JNI 用ライブラリの検索パスを指定します。
4. コンテナ拡張ライブラリから JNI 機能を使用する場合は，Web コンテナサーバ用の usrconf.properties の ejbserver.application.InitTermProcessClasses キーに，サーバ起動・停止フック機能のクラス名を指定します。

ファイルの詳細については，「usrconf.cfg（Web コンテナサーバ用オプション定義ファイル)」を参照してください。

ユーザ定義ファイルの設定例を示します。この例では，コンテナ拡張ライブラリの JAR ファイルは「extended_container.jar」で，そのコンテナ拡張ライブラリが JNI を使用して「extended_container.dll」(UNIX の場合，extended_container）を呼び出します。

Web コンテナサーバ用の usrconf.cfg の設定例を次に示します。

- Windows の場合

```
# system classpath
web.add.class.path=c:¥jar¥extended_container.jar

# ADM service
web.adm.service=false

# library path
add.library.path=c:¥lib
```

- UNIX の場合

```
# system classpath
web.add.class.path=/work/classes/extended_container.jar

# library path
add.library.path=/work/lib
```

(3)　JSP ファイルのコンパイル結果の保持の設定

ここでは，JSP コンパイル結果の保持に関する設定について説明します。

(a)　JSP コンパイル結果保持の設定方法

JSP のコンパイル結果は，デフォルトでは保持する設定となっています。この設定は，usrconf.properties の webserver.work.clean キーでの指定で，変更できます。

デフォルトでは，次のように設定されています。

webserver.work.clean=false

webserver.work.clean キーに false が設定されていると，コンパイル結果を保持します。サーバ終了時に JSP コンパイル結果を削除する場合は，webserver.work.clean キーに true を設定します。

usrconf.properties については，「usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル)」を参照してください。

(b)　JSP コンパイル結果の出力先の設定

JSP コンパイル結果は，JSP 用テンポラリディレクトリ下に出力されます。JSP 用テンポラリディレクトリは，usrconf.properties の webserver.work.directory キーに設定します。デフォルトの出力先を次に示します。

デフォルトの出力先

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥work
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/work

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

Web アプリケーション単位の出力先

JSP 用テンポラリディレクトリ下に作成される Web アプリケーション単位のディレクトリは，コンテキストルート名を基に次の規則に従ったディレクトリ名となります。

- コンテキストルート名にドル記号（$），パーセント（%），プラス記号（+）が含まれる場合は，次に示す文字に変換します。なお，サーブレットエンジンモードの場合は，コンテキストルート名にスラッシュ（/）は使用できません。

変換前の文字	変換後の文字
$	$24
%	$25
+	$2b

例：

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

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥work¥J2EE_AP1$25WEB_AP1_war
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/work/J2EE_AP1$25WEB_AP1_war

! 注意事項

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

(4)　アプリケーションの更新検知とリロードの設定

サーブレットエンジンモードでは，Web アプリケーション開発時のテストを支援する機能として，Web コンテナ上のアプリケーションの更新を検知し，更新したアプリケーションをリロードする機能を備えています。

ここでは，アプリケーションの更新を検知し，リロードする機能を使用する場合の設定について説明します。また，JSP ファイルが更新された場合に，更新後の JSP を再コンパイルする機能の設定についても説明します。

(a)　リロードの動作に関する設定

Web アプリケーションのリロード機能を使用するためには，次に示す設定が必要となります。設定内容と設定ファイルについて説明します。

● リロード機能を使用するための設定

Web アプリケーションのリロード機能を使用する場合は，usrconf.properties または hitachi_web.properties に次の指定をします。

webserver.context.reloadable=true

● 更新検知のインターバル

Web アプリケーションの更新を検知する間隔を指定できます。更新検知は，デーモンスレッドによって実行されます。デフォルトでは 1 秒間隔となっています。

デーモンスレッドの実行間隔を変更する場合，usrconf.properties の webserver.context.check_interval キーに値（単位：秒）を指定します。

ポイント

更新検知の対象となるファイル数が増えると，更新検知処理のオーバーヘッドが大きくなり，CPU 使用率が高くなります。このような場合は，更新検知インターバルを変更することによって，性能への影響が小さくなります。更新検知インターバルの値は大きくすることをお勧めします。

● リソース更新用インターバル

リソース更新用インターバルとは，Web アプリケーションの更新を検知してから，処理中のリクエスト数の監視を実施するまでの時間を指します。

リソース更新用インターバルは，usrconf.properties または hitachi_web.properties の webserver.context.update.interval キーに値（単位：秒）を指定します。

● リロード遅延実行（任意）

リロード遅延実行機能を使用する場合は，ファイル（リソース）の更新を検知してから，リロードを開始するまでの時間を設定します。

リロード遅延実行は，usrconf.properties または hitachi_web.properties の webserver.context.reload_delay_timeout キーに値（単位：秒）を指定します。

● リロード時のセッション引き継ぎ（任意）

Web アプリケーションのリロード実行前に生成したセッション情報を，リロード後も継続して利用するかどうかを設定します。

セッション情報を引き継ぐかどうかの指定は，usrconf.properties または hitachi_web.properties の webserver.context.reload_enabled キーに指定します。セッション情報を引き継ぐ場合は「true」，引き継がない場合は「false」を指定します。デフォルトは「false」で，セッション情報は引き継ぎません。

なお，セッション情報を引き継ぐ場合，セッション情報は，シリアライズされて，**セッション情報ファイル**に出力されます。次に，セッション情報ファイルの出力先と，出力先ディレクトリの変更方法について説明します。

セッション情報ファイルの出力先

セッション情報ファイルはデフォルトでは次のディレクトリに作成されます。

ディレクトリ

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名>¥work¥<コンテキストルート名>¥cjwebsession.dat
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/work/<コンテキストルート名>/cjwebsession.dat

なお, 出力先ディレクトリは Web コンテナサーバ起動時に作成されます。作成されたディレクトリは，Web コンテナサーバを終了しても削除されません。ディレクトリが不要になった場合は，手動で削除してください。

セッション情報ファイルの出力先ディレクトリの変更

セッション情報ファイルの出力先は変更できます。変更するには，usrconf.properties の webserver.session.work.directory キーに，変更後の出力先ディレクトリを指定します。

出力先のディレクトリを変更するときの設定例を次に示します。

出力先ディレクトリ変更の設定例

- Windows の場合（C:¥tmp¥session_work に変更する場合）
  webserver.session.work.directory=C:/tmp/session_work
- UNIX の場合（/usr/tmp/session_work に変更する場合）
  webserver.session.work.directory=/usr/tmp/session_work

なお，この例の場合，コンテキストルートが“examples”の Web アプリケーションのセッション情報ファイルは，次に示すパスに生成されます。

- Windows の場合
  C:¥tmp¥session_work¥examples¥cjwebsession.dat
- UNIX の場合
  /usr/tmp/session_work/examples/cjwebsession.dat

● 設定ファイル

リロードに関する設定は，usrconf.properties または hitachi_web.properties のどちらかで設定します。なお，両方のファイルに設定している場合は，hitachi_web.properties の指定が優先されます。設定項目と，設定できるファイルを次の表に示します。

表 3-28　リロードに関する設定項目と設定ファイル

設定項目	ファイル名	
	usrconf.properties	hitachi_web.properties
リロード機能を使用するための設定	○	○
更新検知のインターバル	○	×
リソース更新用インターバル	○	○
リロード遅延実行	○	○
セッション情報の引き継ぎ	○	○
セッション情報ファイルの出力先	○	×

（凡例）○：設定できる　×：設定できない

usrconf.properties については，「usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル)」を参照してください。hitachi_web.properties については，「hitachi_web.properties（Web コンテナサーバ用 Web アプリケーションプロパティファイル)」を参照してください。

(b)　JSP ファイルの再コンパイルに関する設定

JSP ファイルの再コンパイル機能を使用するためには，次に示す設定が必要となります。設定内容と設定ファイルについて説明します。

● 再コンパイル機能を使用するための設定

JSP ファイルの再コンパイル機能を使用する場合は，usrconf.properties または hitachi_web.properties に次の内容を指定します。

webserver.jsp.recompilable=true

● 更新検知のインターバル

JSP ファイルの更新を検知する間隔を変更する場合は，usrconf.properties の webserver.jsp.check_interval キーに値（秒単位）を指定します。デフォルトでは 1 秒間隔となっています。

● JSP ファイル更新用インターバル

JSP ファイル更新用インターバルとは，JSP ファイルの更新を検知してから，JSP ファイルをコンパイルするまでの間隔を指します。

JSP ファイル更新用インターバルは，usrconf.properties または hitachi_web.properties の webserver.jsp.update.interval キーに値（単位：秒）を指定します。

● 設定ファイル

JSP ファイルの再コンパイルに関する設定は，usrconf.properties または hitachi_web.properties のどちらかで設定します。なお，両方のファイルに設定している場合は，hitachi_web.properties の指定が優先されます。設定項目と，設定できるファイルを次の表に示します。

表 3-29　JSP ファイルの再コンパイルに関する設定項目と設定ファイル

設定項目	ファイル名	
	usrconf.properties	hitachi_web.properties
JSP ファイル再コンパイル機能を使用するための設定	○	○
更新検知のインターバル	○	×
JSP ファイル更新用インターバル	○	○

（凡例）○：設定できる　×：設定できない

(5)　コネクションプーリングの設定

サーブレットエンジンモード上で動作する Web アプリケーションからデータベースへ接続する場合，Web コンテナコネクションプーリング機能を利用してデータベースへの接続をプールできます。これによって，コネクション生成時のパフォーマンスを向上できます。Web コンテナコネクションプーリング機能を利用するには，管理機能用の Web アプリケーションをデプロイし，プール管理情報を登録してください。

Web コンテナコネクションプーリング機能を利用するための手順を次に示します。

1. プール管理情報設定ファイルを作成します。

プール管理情報設定ファイルは，プール管理情報を登録するときに使用するファイルです。プール管理情報設定ファイルには次の情報を記述します。

- 一意のデータベース ID
  プール管理情報の登録で登録した一意のデータベース ID を指定します。
- DABroker Library のクラスおよびデータベース URL（DABroker Library を利用する場合）
  クラス名には，「Class.forName」に渡すクラス名として「com.hitachi.software.web.jdbc.WccpDriver」を指定します。また，URL には，データベース URL として，「jdbc:hitachi:wccp://DBID=<データベース URL>」を指定します。
- コネクションの最小値と最大値など

プール管理情報設定ファイルについては，「3.11.6　プール管理情報設定ファイル」を参照してください。

2. 管理機能用の Web アプリケーションをデプロイします。

管理機能用の Web アプリケーションの格納場所を次に示します。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥jdbc¥cjwebconpool.war
- UNIX の場合
  /opt/Cosminexus/CC/web/jdbc/cjwebconpool.war

Web アプリケーションのデプロイについては，「3.7.7(1)　Web アプリケーションのデプロイ」を参照してください。

3. Web コンテナサーバを起動します。

Web コンテナサーバの起動については，「3.8.2　システムの起動」を参照してください。

4. cjwebeditpool コマンドの引数にプール管理情報設定ファイルを指定して，プール管理情報を登録します。

cjwebeditpool コマンドについては，「cjwebeditpool（プール管理情報の登録と変更)」を参照してください。

5. ユーザ作成の 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デプロイについては，「3.7.7(1)　Web アプリケーションのデプロイ」を参照してください。

なお，コネクションの取得処理は，Web アプリケーションで実施します。このため，ユーザ作成の Web アプリケーションに，コネクションを操作するための記述が必要になります。Web アプリケーションの記述方法については，「3.6.3　Web コンテナコネクションプール機能によるデータベース接続」を参照してください。

3.7.5　トラブルシューティングの資料取得の設定

ここでは，トラブルシューティングの資料を取得するための設定方法について説明します。

(1)　資料取得の設定の概要

トラブルシューティングに必要な資料のうち，一部の資料は，運用を開始する前に資料取得のための設定をしておく必要があります。例えば，OS の統計情報，ユーザダンプ（Windows の場合）または core ダンプ（UNIX の場合），JavaVM の GC のログなどは，事前に取得のための設定をしておかないと，取得できません。これらの資料はトラブルシューティングで必要となるため，取得することをお勧めします。

デフォルトの設定で取得できるようになっている資料については，特に設定は不要ですが，ログの出力先やサイズなどを変更したい場合には，ユーザ定義ファイルを編集して，設定を変更してください。

トラブルシューティングの資料取得のための設定を次の表に示します。

表 3-30　トラブルシューティングの資料取得のための設定（サーブレットエンジンモードの場合）

資料の種類	資料取得のための設定	設定の要否	設定方法の参照先
性能解析トレースファイル	事前に，取得のための設定をしておく必要はありません。日常的なシステム運用の作業で，cprfed コマンドを実行するときに取得条件などを指定します。性能解析トレースファイルの取得については，「3.8.4　システムの処理性能の解析」を参照してください。	×	−
Web コンテナサーバのログ	Web コンテナサーバ用の usrconf.cfg と usrconf.properties で，ログの出力レベル，サイズ，面数などを設定します。	△	(2)
リダイレクタのログ	mod_jk.conf（HTTP Server の場合）または isapi_redirect.conf（Microsoft IIS の場合）で，ログの出力レベル，サイズ，面数などを設定します。	△	(3)

資料の種類	資料取得のための設定	設定の要否	設定方法の参照先
TPBroker のトレースファイル	Web コンテナサーバ用の usrconf.cfg と usrconf.properties で，トレースファイルの出力先や面数などを設定します。	△	(4)
DABroker Library のログ	環境設定ユティリティまたは DABroker Library 動作環境定義ファイルで，ログファイルサイズなどを設定します。	△	(5)
OS の統計情報	Windows の場合，Windows のシステムモニタでシステムリソースのパフォーマンスデータ取得の設定をします。	○	(6)
ユーザダンプ	Windows の場合，ワトソン博士または環境変数（CJMEMDUMP_PATH）で，ユーザダンプ取得の設定をします。	○	(7)
core ダンプ	UNIX の場合，usrconf.cfg やシェルコマンドで，core ファイル取得の設定をします。	○	(8)
JavaVM の資料	Web コンテナサーバ用の usrconf.cfg で，JavaVM のスレッドダンプや JavaVM ログ（JavaVM ログファイル）の出力方法や出力内容などの設定をします。	○	(9)
Web アプリケーションのユーザログ	Web コンテナサーバ用の usrconf.properties で，ロガーやハンドラの設定，ログの出力レベル，サイズ，面数などを設定します。また，web.policy で，セキュリティポリシーを設定します。 設定方法は，J2EE アプリケーションの場合と同じです。Web アプリケーションのユーザログ出力については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「9. アプリケーションのユーザログ出力」を参照してください。ただし，サーブレットエンジンモードの場合は，簡易構築定義ファイルではなく，Web コンテナサーバ用の usrconf.properties で設定します。設定時には，簡易構築定義ファイルで設定するパラメタを，Web コンテナサーバ用の usrconf.properties のキーに読み替えてください。Web コンテナサーバ用の usrconf.properties については，「usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル)」を参照してください。	○	(10)

（凡例）

○：資料を取得する場合には，設定が必要

△：デフォルトの設定をカスタマイズする場合だけ，設定が必要

×：設定は不要

－：該当しない

なお，次のログはログ出力先を変更できません。

- セットアップ時に作成される install.log，移行コマンドなどのログファイル
- 次のディレクトリに出力される Java のスレッドダンプファイル
  - Windows の場合
    <製品のインストールディレクトリ>¥CC¥web¥<サーバ名称>
  - UNIX の場合
    /opt/Cosminexus/CC/web/<サーバ名称>

(2)　Web コンテナサーバのログ取得の設定

Web コンテナサーバのログの出力先，ログサイズ，ログレベルの変更について説明します。変更できる項目と，項目に対応するユーザ定義ファイルとキーを次の表に示します。

表 3-31　Web コンテナサーバのログ取得の設定項目

項目	対応するユーザ定義ファイルとキー
ログの出力先	usrconf.cfg の web.server.log.directory キー
ログサイズ	usrconf.properties の ejbserver.logger.channels.define.<チャネル名>.filenum キー usrconf.properties の ejbserver.logger.channels.define.<チャネル名>.filesize キー
ログレベル	usrconf.properties の ejbserver.logger.enabled.*キー

ファイルおよびキーについては，「3.11　Web コンテナサーバで使用するファイル」を参照してください。

(a)　ログの出力先の変更

デフォルトのログ出力先を次に示します。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs

ログの出力先を変更する場合は，usrconf.cfg の web.server.log.directory キーでログの出力先ディレクトリを指定します。

設定例

- Windows の場合
  web.server.log.directory=C:¥CClogs¥web¥YourServer
- UNIX の場合
  web.server.log.directory=/CClogs/web/YourServer

カレントディレクトリ

ログ出力先を相対パスで指定する場合のカレントディレクトリを次に示します。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>

注意事項

- ログの出力先を変更した場合は，Web コンテナサーバを起動する前に，変更後のログの出力先ディレクトリを作成しておいてください。変更後のログの出力先ディレクトリがない場合は，Web コンテナサーバのサーバ起動時に KDJE40024-E のメッセージが出力されて異常終了します。
- 同じマシン内で複数の Web コンテナサーバを起動させている場合は，ログ出力先が同じディレクトリにならないように，ディレクトリにサーバ名称を含めるなど，サーバごとにユニークなディレ

クトリ名になるようにしてください。なお，キーの値に同じディレクトリを指定した場合は，動作の保証はしません。

- ログの出力先を変更して作業ディレクトリ以外にログを出力する場合，ログファイルはサーバのアンセットアップ時に削除されません。ログファイルを削除したい場合には，手動で削除してください。
- ユーザ定義ファイルでログの出力先を設定していても，次の設定が優先されますので，注意してください。

  usrconf.cfg に，add.jvm.arg=-XX:HitachiJavaLog:<JavaVM の保守情報および GC のログ出力先>キーを指定している場合は，JavaVM の保守情報および GC のログ出力先に設定したディレクトリに，JavaVM の保守情報および GC のログファイルが出力されます。
- ログ出力先には，UNC 名を含むパスは指定できません。

(b)　ログサイズの変更

ログサイズを変更する場合は，usrconf.properties で，ログファイルの面数，およびログファイル 1 面当たりの最大サイズを設定します。

ログファイル面数の変更

ログファイルの面数は，usrconf.properties の ejbserver.logger.channels.define.<チャネル名>.filenum キーで設定します。

記述例

```
ejbserver.logger.channels.define.MessageLogFile.filenum=3
```

ログファイル 1 面当たりの最大サイズの変更

ログファイル 1 面当たりの最大サイズは（単位：バイト），usrconf.properties の ejbserver.logger.channels.define.<チャネル名>.filesize キーで設定します。

記述例

```
ejbserver.logger.channels.define.MessageLogFile.filesize=2097152
```

(c)　ログレベルの変更

ログレベルは，ログの重要度を表します。ログレベルには，「Error」，「Warning」，「Information」，「Debug」の四つがあります。ログレベルを設定すると，設定したレベルのログが出力されます。デフォルトでは，Error レベルのログだけが取得されます。通常はデフォルトのまま利用してください。

ログレベルは，Web コンテナサーバ用の usrconf.properties の ejbserver.logger.enabled.*キーで設定します。

ejbserver.logger.enabled.*=<レベル名>

レベル名には，「Error」，「Warning」，「Information」，「Debug」の文字列を一つ，または複数設定します。複数設定する場合には，レベル名の文字列の間をコンマ（,）で区切ります。

記述例

1. ejbserver.logger.enabled.*=Error
2. ejbserver.logger.enabled.*=Error,Warning
3. ejbserver.logger.enabled.*=Error,Warning,Information
4. ejbserver.logger.enabled.*=Error,Warning,Information,Debug

注意事項

- 記述例の 1.，2.，3.，4.，の順に，取得できるログの件数が増加していきます。複数のログレベルを設定してログを取得すると，性能が劣化し，ログファイルの面の切り替えが頻繁に起こるようになります。
- レベル名に「Error」，「Warning」，「Information」，「Debug」以外の文字列，または空の値を設定した場合は，KDJE90009-W のメッセージが出力されます（Error レベルのログは取得されます）。

ログレベルの推奨設定

ログレベルの推奨設定を次に示します。

- 通常運用時

  「ejbserver.logger.enabled.*=Error」を指定します。
- 通常運用時（verbose）

  通常運用時よりも詳細な情報を取得する場合には，「ejbserver.logger.enabled.*=Error,Warning」を指定します。
- テスト時

  「ejbserver.logger.enabled.*=Error,Warning,Information」を指定します。
- 障害調査時

  「ejbserver.logger.enabled.*=Error,Warning,Information,Debug」を指定します。

(3)　リダイレクタのログ取得の設定

リダイレクタの出力するログの出力先やログ出力レベルなどの変更について説明します。なお，ファイルの詳細およびキーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.2　isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）」を参照してください。

リダイレクタが出力するログは，メッセージログと保守用トレースログの二つに分けて出力されます。メッセージログには，障害発生時のログなどのメッセージが出力されます。保守用トレースログには，HTTP リクエストを受信し，レスポンスを送信するまでのトレース情報が出力されます。

保守用トレースログのログサイズは，見積もりの計算式に従い，障害発生時から障害を検知しログを収集するまで保持されるサイズにカスタマイズすることをお勧めします。

注意事項

- ログファイル初期化処理の開始時および終了時には，メッセージログにそれぞれ KDJE41029-I，KDJE41030-I のメッセージが出力されます。Web サーバの起動時には，ログファイル初期化処理が正常に終了したことを確認するため，メッセージログに KDJE41029-I，KDJE41030-I のメッセージが出力されていることを確認してください。
- UNIX の場合，mod_jk.conf に設定されたファイルサイズ，およびファイル面数のログファイルは，Web サーバとの連携の設定後の初回起動時に作成されます。ログファイルの作成は，ログファイルサイズおよびファイル面数に比例した処理時間が必要となります。ログファイルサイズを大きく設定した場合，またはファイル面数を増やした場合などは，ログファイルの作成処理に数分掛かる場合があります。

(a)　HTTP Server の場合

HTTP Server を使用している場合は，mod_jk.conf でリダイレクタのログ取得の設定を変更できます。なお，UNIX の場合，設定を変更できるのは，root 権限（Component Container 管理者を設定していない

場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）を持つユーザです。

● 変更した設定を有効にする方法

変更した設定を有効にする方法について，OS ごとに説明します。

Windows の場合

変更したリダイレクタのログ取得の設定を有効にするには，HTTP Server を再起動してください。

UNIX の場合

変更したリダイレクタのログ取得の設定を有効にする方法は，ファイルサイズまたはファイル面数の変更の有無によって異なります。

ファイルサイズまたはファイル面数を変更する場合

ファイルサイズまたはファイル面数を変更する場合に，変更したリダイレクタのログ取得の設定を有効にする方法を次に示します。

1. HTTP Server を停止します。
2. ログファイルおよび管理ファイルを移動または削除します。

   メッセージログファイルの場合：メッセージログファイルおよび HNTRLib が使用する管理ファイル（<JkLogFilePrefix の設定値>.mm）を移動，または削除します。

   保守用トレースログファイルの場合：保守用トレースログファイルおよび HNTRLib が使用する管理ファイル（<JkTrcaeLogFilePrefix の設定値>.mm）を移動，または削除します。
3. HTTP Server を再起動します。

なお，HNTRLib が使用する管理ファイルのデフォルトの格納場所は次のとおりです。

/opt/Cosminexus/CC/web/redirector/logs/mmap

ファイルサイズまたはファイル面数を変更しない場合

ファイルサイズまたはファイル面数を変更しない場合は，HTTP Server を再起動してください。

● 変更できる設定項目

変更できる設定項目と，項目に対応するユーザ定義ファイルとキーを次の表に示します。

表 3-32　HTTP Server でのリダイレクタのログ取得の設定項目（サーブレットエンジンモードの場合）

項目	mod_jk.conf のキー
メッセージログ，保守用トレースログのログ出力レベル	JkLogLevel
メッセージログ出力先	JkLogFileDir
メッセージログファイルサイズ	JkLogFileSize
メッセージログファイル面数	JkLogFileNum
メッセージログファイル名のプリフィックス	JkLogFilePrefix
保守用トレースログ出力の有無	JkTraceLog
保守用トレースログ出力先	JkTraceLogFileDir※
保守用トレースログファイルサイズ	JkTraceLogFileSize
保守用トレースログファイル面数	JkTraceLogFileNum

項目	mod_jk.conf のキー
保守用トレースログファイル名のプリフィックス	JkTrcaeLogFilePrefix※

注※　保守用トレースログの出力先とプリフィックスには，同じ値を指定しないでください。同じ値を指定した場合は，リダイレクタが動作しません。

メッセージログ，および保守用トレースログのデフォルトの出力先

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥redirector¥logs
- UNIX の場合
  /opt/Cosminexus/CC/web/redirector/logs

ログレベルのレベル名

ログレベルのレベル名には，重要度の高い順に「emerg」※，「error」，「info」，「debug」のどれかを指定できます。

- emerg：通常運用時に指定します。常に出力する必要がある重要度の高いメッセージだけが出力されます。※
- error：通常運用時に指定します。
- info：テスト時に指定します。
- debug：障害調査時に指定します。

ログレベルは，単独で指定してください（コンマ区切りで複数のログレベルを指定しないでください）。あるログレベルを指定すると，指定されたログレベルおよびそれよりも重要度の高いログが出力されます。

注※　Windows の場合に指定できます。

ログサイズの見積もりの計算式

GET リクエスト，POST リクエストの 1HTTP アクセス当たりのログサイズは，次の計算式で見積もってください。

- GET による 1HTTP アクセス当たりのログサイズ
  ＝ 1.75 ＋（平均レスポンスボディサイズ / 16）* 0.25
- POST による 1HTTP アクセス当たりのログサイズ
  ＝ 2.00 ＋（平均 POST データサイズ / 16）* 0.5 ＋（平均レスポンスボディサイズ / 16）* 0.25

なお，単位はすべてキロバイト，除算結果の小数点以下の値はすべて切り捨てです。また，計算式ではリクエスト URL とクエリー文字列の平均サイズは，それぞれ 100 バイトとしています。リクエスト URL とクエリー文字列の平均サイズが 100 バイトを超えた場合は，計算式の最初の値に，超える分のサイズを加えて計算してください。

(b)　Microsoft IIS の場合

Microsoft IIS を使用している場合は，isapi_redirect.conf でリダイレクタのログ取得の設定を変更できます。

注意事項

isapi_redirect.conf に設定されたファイルサイズ，およびファイル面数のログファイルは，Web サーバとの連携の設定後の初回リクエスト時に作成されます。ログファイルの作成は，ログファイルサイズおよびファイル面数に比例した処理時間が必要となります。ログファイルサイズを大きく設定した場

合，またはファイル面数を増やした場合などは，ログファイルの作成処理に数分掛かる場合があります。

Web サーバとの連携の設定後およびログファイルの設定変更後の初回リクエストは，運用を開始する前に実施しておくことをお勧めします。

● 変更した設定を有効にする方法

変更したリダイレクタのログ取得の設定を有効にする方法は，Microsoft IIS のバージョンによって異なります。

ファイルサイズまたはファイル面数を変更する場合に，変更したリダイレクタのログ取得の設定を有効にするための手順を次に示します。

1. Microsoft IIS を停止します。
2. ログファイルおよび管理ファイルを移動または削除します。

   メッセージログファイルの場合：メッセージログファイルおよび HNTRLib が使用する管理ファイル（<log_file_prefix の設定値>.mm）を移動，または削除します。

   保守用トレースログファイルの場合：保守用トレースログファイルおよび HNTRLib が使用する管理ファイル（< trace_log_file_prefix の設定値>.mm）を移動，または削除します。

   なお，HNTRLib が使用する管理ファイルのデフォルトの格納場所は次のとおりです。

   <製品 のインストールディレクトリ>¥CC¥web¥redirector¥logs¥mmap
3. Microsoft IIS を起動します。

   ファイルサイズおよびファイル面数を変更しない場合に，変更したリダイレクタのログ取得の設定を有効にするための手順を次に示します。
4. Microsoft IIS を再起動するか，またはワーカプロセスのリサイクルを行います。

   Microsoft IIS の「インフォメーション サービス マネージャ」を使用して，ワーカプロセスのリサイクルができます。

● 変更できる設定項目

変更できる設定項目と，項目に対応するユーザ定義ファイルとキーを次の表に示します。なお，Microsoft IIS のバージョンによっては，キーに指定する値が異なります。isapi_redirect.conf については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.2　isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル)」を参照してください。

表 3-33　Microsoft IIS でのリダイレクタのログ取得の設定項目（サーブレットエンジンモードの場合）

項目	isapi_redirect.conf のキー
メッセージログ，保守用トレースログのログ出力レベル	log_level
メッセージログ出力先	log_file_dir
メッセージログファイルサイズ	log_file_size
メッセージログファイル面数	log_file_num
メッセージログファイル名のプリフィックス	log_file_prefix
保守用トレースログ出力の有無	trace_log
保守用トレースログ出力先	trace_log_file_dir※

項目	isapi_redirect.conf のキー
保守用トレースログファイルサイズ	trace_log_file_size
保守用トレースログファイル面数	trace_log_file_num
保守用トレースログファイル名のプリフィックス	trace_log_file_prefix※

注※　保守用トレースログの出力先とプリフィックスには，同じ値を指定しないでください。同じ値を指定した場合は，リダイレクタが動作しません。

メッセージログ，および保守用トレースログのデフォルトの出力先

<製品のインストールディレクトリ>¥CC¥web¥redirector¥logs

ログレベルのレベル名

ログレベルのレベル名には，重要度の高い順に「emerg」，「error」，「info」，「debug」のどれかを指定できます。

- emerg：通常運用時に指定します。常に出力する必要がある重要度の高いメッセージだけが出力されます。
- error：通常運用時に指定します。
- info：テスト時に指定します。
- debug：障害調査時に指定します。

ログレベルは，単独で指定してください（コンマ区切りで複数のログレベルを指定しないでください）。あるログレベルを指定すると，指定されたログレベルおよびそれよりも重要度の高いログが出力されます。

ログサイズの見積もりの計算式

GET リクエスト，POST リクエストの 1HTTP アクセス当たりのログサイズは，次の計算式で見積もってください。

- GET による 1HTTP アクセス当たりのログサイズ
  ＝ 2.25 ＋ (平均レスポンスボディサイズ / 16) * 0.25
- POST による 1HTTP アクセス当たりのログサイズ
  ＝ 2.50 ＋ (平均 POST データサイズ / 16) * 0.5 ＋ (平均レスポンスボディサイズ / 16) * 0.25

なお，単位はすべてキロバイト，除算結果の小数点以下の値はすべて切り捨てです。また，計算式ではリクエスト URL とクエリー文字列の平均サイズは，それぞれ 100 バイトとしています。リクエスト URL とクエリー文字列の平均サイズが 100 バイトを超えた場合は，計算式の最初の値に，超える分のサイズを加えて計算してください。

(4)　TPBroker のログ取得の設定

TPBroker のトレースファイルの出力先，ファイル数，エントリ数の変更について説明します。

TPBroker のトレースファイルのデフォルトの出力先は次のとおりです。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥TPB¥logj
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/TPB/logj

変更できる項目と，項目に対応するユーザ定義ファイルとキーを次に示します。

項目	対応するユーザ定義ファイルとキー
トレースファイルの出力先	usrconf.cfg の web.server.log.directory キー または usrconf.properties の vbroker.orb.htc.tracePath キー
トレースファイルのファイル数	usrconf.properties の vbroker.orb.htc.comt.fileCount キー
トレースファイルのエントリ数	usrconf.properties の vbroker.orb.htc.comt.entryCount キー

ファイルおよびキーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

注意事項

- トレースファイルの出力先を変更する場合は，変更後のトレースファイル出力先ディレクトリのサブディレクトリとして comtrc と mdltrc をあらかじめ作成しておく必要があります。出力先を変更すると，変更後のログ出力先ディレクトリ下の comtrc と mdltrc にトレースファイルが出力されます。
- usrconf.cfg の web.server.log.directory キーと，usrconf.properties の vbroker.orb.htc.tracePath キーをどちらも設定した場合は，usrconf.properties の vbroker.orb.htc.tracePath キーの設定が優先されます。

(5)　DABroker Library のログ取得の設定

環境設定ユティリティまたは DABroker Library 動作環境定義ファイルで，ログファイルサイズなどを設定します。環境設定ユティリティまたは DABroker Library 動作環境定義ファイルの設定については，「4.3　DABroker Library の初期設定」を参照してください。

(6)　OS の統計情報取得の設定

Windows のシステムモニタを使用して，システムリソースのパフォーマンスデータを取得できます。システムリソースのパフォーマンスデータを取得するための設定方法は，運用管理ポータルでシステムを構築する場合の設定方法と同じです。詳細については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」およびマニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.16　OS の統計情報取得の設定」を参照してください。

(7)　ユーザダンプ取得の設定

ユーザダンプを取得するための設定は，OS ごとに異なります。

● Windows XP の場合

ユーザダンプを取得するためには，ワトソン博士を使用できるように設定しておいてください。ユーザダンプを取得するための設定方法は，運用管理ポータルでシステムを構築する場合の設定方法と同じです。詳細については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」およびマニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.17　ユーザダンプ取得の設定」を参照してください。

● Windows Server 2012，Windows Server 2008，Windows 8，Windows 7，または Windows Vista の場合

cjstopweb コマンドの-fd オプションを使用してユーザダンプを取得する場合には，環境変数「CJMEMDUMP_PATH」にユーザダンプの出力先ディレクトリを指定しておいてください。ユーザダンプのファイル名は，cjmemdump.dmp です。

環境変数「CJMEMDUMP_PATH」の設定例を次に示します。

環境変数「CJMEMDUMP_PATH」の設定例

```
set CJMEMDUMP_PATH=C:¥temp
```

この例の場合，C:¥temp の下に cjmemdump.dmp が作成されます。

なお，環境変数「CJMEMDUMP_PATH」を指定する場合には，次の点に注意してください。

- 保存先のディスクに十分な空き容量があることを確認してください。ユーザダンプのファイルサイズは Web コンテナサーバの実メモリ所要量以上になります。
- 日本語などのマルチバイト文字を含むディレクトリを指定しないでください。ユーザダンプの出力に失敗する場合があります。
- ユーザダンプの出力先ディレクトリには，存在するディレクトリを指定してください。

(8)　core ダンプ取得の設定

core ダンプを取得するための設定方法は，運用管理ポータルでシステムを構築する場合の設定方法と同じです。詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.18　core ダンプ取得の設定」を参照してください。ただし，サーブレットエンジンモードの場合は，簡易構築定義ファイルではなく，Web コンテナサーバ用の usrconf.cfg で設定します。設定時には，簡易構築定義ファイルで設定するパラメタを，Web コンテナサーバ用の usrconf.cfg のキーに読み替えてください。Web コンテナサーバ用の usrconf.cfg については，「usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）」を参照してください。

(9)　JavaVM の資料取得の設定

ここでは，次に示す JavaVM の資料を取得するための設定について説明します。

- JavaVM のスレッドダンプ
- JavaVM ログ（JavaVM ログファイル）

(a)　JavaVM のスレッドダンプ取得の設定

JavaVM のスレッドダンプを取得するための設定について説明します。

JavaVM のスレッドダンプに出力される内容は，usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）の add.jvm.arg キーに指定している JavaVM の起動オプションによって異なります。

JavaVM のスレッドダンプを出力する場合に指定するオプションを次の表に示します。

表 3-34　JavaVM のスレッドダンプを取得するために指定するオプション（サーブレットエンジンモードの場合）

オプション	説明
-XX:+HitachiThreadDump	**拡張スレッドダンプ**を出力します。デフォルトの状態では，出力するように設定されています。
-XX:+HitachiThreadDumpToStdout	拡張スレッドダンプを標準出力に出力します。デフォルトの状態では，出力するように設定されています。
-XX:+HitachiThreadDumpWithHashCode	スレッド情報にスレッドのハッシュコードを出力します。デフォルトの状態では，出力するように設定されています。
-XX:+HitachiThreadDumpWithCpuTime	スレッド情報にスレッドを開始してからのユーザ CPU 時間，およびカーネル CPU 時間を出力します。デフォルトの状態では，出力するように設定されています。
-XX:+HitachiThreadDumpWithBlockCount	スレッド情報にスレッドが処理をブロックした回数，および処理が待ち状態になった回数を出力します。デフォルトの状態では，出力するように設定されています。
-XX:+HitachiOutOfMemoryAbort -XX:+HitachiOutOfMemoryAbortThreadDump	どちらも設定されている場合は，OutOfMemoryError によって強制終了した時に，スレッドダンプを出力します。ただし，J2SE クラスライブラリで C ヒープ不足が発生した場合，および JavaVM の処理中に C ヒープ不足が発生した場合は，スレッドダンプを出力しません。
-XX: +HitachiOutOfMemoryAbortThreadDumpWithJHeapProf	OutOfMemoryError によって強制終了した時に出力されるスレッドダンプに，クラス別統計情報を出力します。デフォルトでは出力されません。

(b)　JavaVM ログ取得の設定

JavaVM ログを取得するための設定について説明します。

JavaVM ログとは，製品が標準の JavaVM に追加した拡張オプションを使用して取得できるログです。標準の JavaVM よりも，多くのトラブルシュート情報が取得できます。このログファイルを，**JavaVM ログファイル**といいます。

JavaVM ログファイルを取得するためには，usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）の add.jvm.arg キーに JavaVM の起動オプションを指定します。

JavaVM ログファイルを取得するために指定するオプションを次の表に示します。

表 3-35　JavaVM ログ（JavaVM ログファイル）を取得するために指定するオプション（サーブレットエンジンモードの場合）

オプション	説明
-XX:+HitachiOutOfMemoryStackTrace -XX:+HitachiVerboseGC -XX:+HitachiJavaClassLibTrace	三つのオプションのうちどれかを指定している場合に，JavaVM ログファイルを出力します。 • -XX:+HitachiOutOfMemoryStackTrace を指定している場合は，例外情報とスタックトレースを JavaVM ログファイルに出力します。なお，このオプションを指定すると，-XX:+HitachiOutOfMemorySize および-XX:+HitachiOutOfMemoryCause も同時に指定されます。

オプション	説明
-XX:+HitachiOutOfMemoryStackTrace -XX:+HitachiVerboseGC -XX:+HitachiJavaClassLibTrace	• -XX:+HitachiVerboseGC を指定している場合は，GC が発生したときに，拡張 verbosegc 情報を JavaVM ログファイルに出力します。 • -XX:+HitachiJavaClassLibTrace を指定している場合は，クラスライブラリのスタックトレースを JavaVM ログファイルに出力します。

必要に応じて，次の拡張オプションを指定して，JavaVM ログファイルの出力方法や出力内容を設定してください。

- JavaVM ログファイルのファイルサイズやファイル数
- 拡張 verbosegc 機能オプション
- OutOfMemoryError 発生時の拡張機能オプション
- クラスライブラリトレース機能オプション
- ローカル変数情報出力機能オプション
- スワップ領域の予約オプション　など

JavaVM のオプションについては，「usrconf.cfg（Web コンテナサーバ用オプション定義ファイル)」を参照してください。

(c)　拡張 verbosegc 機能オプションの設定

ここでは，JavaVM の保守情報（Java ヒープの情報）および GC のログに拡張 verbosegc 情報を出力するための設定について説明します。次の表に示すオプションで，拡張 verbosegc 情報の出力を有効にして，拡張 verbosegc 情報の出力形式を指定してください。

表 3-36　拡張 verbosegc 情報を出力するために指定するオプション（サーブレットエンジンモードの場合）

オプション	説明
-XX:+HitachiVerboseGC	拡張 verbosegc 情報を JavaVM ログファイルに出力します。
-XX:+HitachiVerboseGCPrintDate	拡張 verbosegc 情報の各行に日付を出力します。
-XX:+HitachiVerboseGCCpuTime※	GC の開始から終了までの間で，GC を実行したスレッドの CPU 利用時間を出力します。CPU 利用時間は，ユーザモードで費やした CPU 時間とカーネルモードで費やした CPU 時間に分けて出力されます。
-XX:HitachiVerboseGCIntervalTime=<時間間隔(秒)>	拡張 verbosegc 情報の出力間隔を指定します。
-XX:+HitachiVerboseGCPrintCause	拡張 verbosegc 情報に，GC が発生した要因を出力します。
-XX:+HitachiOutputMilliTime	拡張 verbosegc 情報の各行に日時（ミリ秒まで）を出力します。
-XX:+HitachiCommaVerboseGC	拡張 verbosegc 情報を CSV 形式で出力します。
-XX: +HitachiVerboseGCPrintTenuringDistribution	Survivor 領域の年齢分布情報を出力します。出力形式や出力情報については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「9.11　Survivor 領域の年齢分布情報出力機能」を参照してください。

オプション	説明
-XX: +HitachiVerboseGCPrintJVMInternalMemory	JavaVM 内部で管理しているヒープ情報を JavaVM ログファイルに出力します。
-XX:+HitachiVerboseGCPrintThreadCount	Java スレッドの数を監視するために，Java スレッドの数を JavaVM ログファイルに出力します。
-XX:+HitachiVerboseGCPrintDeleteOnExit	java.io.File.deleteOnExit()を呼び出したことによって JavaVM が確保した累積のヒープサイズとメソッドの呼び出し回数を，JavaVM ログファイルに出力します。

注※

HP-UX の場合，このオプションは使用できません。オプションを指定しても無視されます。

拡張 verbosegc 情報から，そのサーバで必要とする Java ヒープ領域サイズ，Metaspace 領域サイズなどを見積もるための情報を取得できます。

(10)　Web アプリケーションのユーザログ出力の設定

ここでは，Web アプリケーションが出力するログを，トレース共通ライブラリ形式で出力するための設定方法について説明します。Web アプリケーションのユーザログを出力しない場合は，この設定は不要です。

アプリケーションサーバでは，トレース共通ライブラリを使用して，Web アプリケーションが出力するログをアプリケーションサーバの構成ソフトウェアが出力するログと同じ形式（トレース共通ライブラリ形式）で出力できます。これによって，システムのログと Web アプリケーションのログを同じ形式で扱うことができるようになり，システム全体のログ運用の信頼性とログ出力の性能を高められます。

Web アプリケーションが出力するログを，トレース共通ライブラリ形式で出力する場合は，Web アプリケーションを開発する際に，ログ出力処理を設定する必要があります。また，usrconf.properties を編集して，ハンドラからのログの出力先，ログレベル，ログ面数，使用するフィルタ，フォーマッタなどを指定する必要があります。

ユーザログを出力するための設定方法については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「9.　アプリケーションのユーザログ出力」を参照してください。ただし，次の点に注意してください。ただし，サーブレットエンジンモードの場合は，簡易構築定義ファイルではなく，Web コンテナサーバ用の usrconf.properties で設定します。設定時には，簡易構築定義ファイルで設定するパラメタを，Web コンテナサーバ用の usrconf.properties のキーに読み替えてください。Web コンテナサーバ用の usrconf.properties については，「usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル）」を参照してください。

- サーブレットエンジンモードの場合は，usrconf.properties の格納場所が異なります。サーブレットエンジンモードの usrconf.properties の格納場所を次に示します。
  - Windows の場合
    <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.properties
  - UNIX の場合
    /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.properties
- サーブレットエンジンモードの場合は，セキュリティポリシーを web.policy に指定してください。web.policy の格納場所を次に示します。
  - Windows の場合

<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥web.policy

- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/web.policy

3.7.6　統合ユーザ管理機能の設定

ここでは，サーブレットエンジンモードの場合の，統合ユーザ管理機能の設定について説明します。統合ユーザ管理機能を使用しない場合は，この設定は不要です。

(1)　統合ユーザ管理機能の設定手順

統合ユーザ管理機能の設定手順について説明します。

アプリケーションサーバでは，アプリケーションサーバで構築したシステムにログインするユーザを統合管理できます。統合ユーザ管理では，それぞれの J2EE アプリケーションが管理しているユーザの情報が関連づけて管理され，一度のログイン処理でさまざまな J2EE アプリケーションにログインできるようになります。統合ユーザ管理機能を使用する場合は，ユーザ認証情報を格納する LDAP ディレクトリサーバや，統合ユーザ管理のコンフィグレーションファイルを設定する必要があります。

統合ユーザ管理機能を使用するための設定手順は，運用管理ポータルを使用してシステムを構築するときの設定手順とほとんど同じです。運用管理ポータルを使用したシステム構築の場合の統合ユーザ管理機能の設定手順については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」を参照してください。

(2)　統合ユーザ管理機能の設定方法

統合ユーザ管理機能の設定方法について説明します。

統合ユーザ管理機能を使用するための設定方法は，運用管理ポータルを使用してシステムを構築するときの設定方法とほとんど同じです。運用管理ポータルを使用したシステム構築の場合の統合ユーザ管理機能の設定方法については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「5. 統合ユーザ管理による認証」を参照してください。ただし，サーブレットエンジンモードの場合では，設定方法が少し異なります。設定方法の差異を次に示します。

(a)　サーバ名について

サーブレットエンジンモードの場合は，「J2EE サーバ」を「Web コンテナサーバ」と読み替えてください。

(b)　ユーザ情報の登録について

サーブレットエンジンモードの場合は，統合ユーザ管理 GUI（Management Server の運用管理ポータルの「リポジトリ管理」）でのユーザ情報の登録はできません。ユーザ情報は，コマンド，ユーザ認証ライブラリまたはシングルサインオンライブラリの API を使用して登録してください。

(c)　JavaVM のプロパティの設定について

統合ユーザ管理機能を使用する場合は，JavaVM の起動時に，J2EE サーバの usrconf.properties と web.policy の設定が必要です。

詳細は，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「5.21　JavaVM のプロパティの設定」を参照してください。

(d)　ファイルのデプロイについて

サーブレットエンジンモードの場合は，デプロイする統合ユーザ管理用の「EAR ファイル（uastartup.ear)」を「WAR ファイル（uastartup.war)」と読み替えてください。また，WAR ファイルのデプロイについては，「3.7.7(1)　Web アプリケーションのデプロイ」を参照してください。
uastartup.war は，次に示すディレクトリにインストールされています。

- Windows の場合
  <製品のインストールディレクトリ>¥manager¥config
- UNIX の場合
  /opt/Cosminexus/manager/config

(e)　JSP タグライブラリを使用している場合の設定について

Web コンテナサーバにデプロイ済みの Web アプリケーションで，JSP タグライブラリを使用している場合には，ファイルのコピーや編集が必要です。

JSP タグライブラリ用の JAR ファイル（uatags.jar）とタグライブラリ記述子ファイル（uatags.tld）をコピーして，Web アプリケーションの DD（web.xml）を編集してください。手順を次に示します。

1. uatags.jar を，<デプロイ先ディレクトリ>¥<Web アプリケーション名>¥WEB-INF¥lib（Windows の場合），または<デプロイ先ディレクトリ>/<Web アプリケーション名>/WEB-INF/lib（UNIX の場合）にコピーします。
   例えば，uCosminexus Portal Framework を使用する場合，<Web アプリケーション名>は，Portal です。
2. uatags.tld を，<デプロイ先ディレクトリ>¥<Web アプリケーション名>¥WEB-INF（Windows の場合），または<デプロイ先ディレクトリ>/<Web アプリケーション名>/WEB-INF（UNIX の場合）にコピーします。
3. web.xml の適切な位置に次の記述を追加します。

```
<taglib>
<taglib-uri>http://cosminexus.com/admin/auth/uatags</taglib-uri>
<taglib-location>/WEB-INF/uatags.tld</taglib-location>
</taglib>
```

web.xml は次の場所に格納されています。

- Windows の場合
  <デプロイ先ディレクトリ>¥<Web アプリケーション名>¥WEB-INF¥web.xml
- UNIX の場合
  <デプロイ先ディレクトリ>/<Web アプリケーション名>/WEB-INF/web.xml

<デプロイ先ディレクトリ>については，「3.7.8　Web コンテナサーバの作業ディレクトリ」を参照してください。

参考

JSP タグライブラリを使用した Web アプリケーションの作成については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「5.11　タグライブラリを使用したユーザ認証の実装」を参照してください。なお，JSP タグライブラリを使用する場合は，上記の手順 3.に従って web.xml を編集してください。

3.7.7　Web アプリケーションの設定

サーブレットエンジンモードの環境では，Web アプリケーションを実行する前に，Web コンテナサーバにデプロイします。Web アプリケーションのデプロイについては，「(1)　Web アプリケーションのデプロイ」を参照してください。Web アプリケーションのアンデプロイについては，「(2)　Web アプリケーションのアンデプロイ」を参照してください。また，Web アプリケーションへのサーブレットフィルタの追加については，「(3)　Web アプリケーションに組み込むサーブレットフィルタの設定」を参照してください。

なお，Web アプリケーションは，Developer や IDE などを使用して，あらかじめ作成しておいてください。Developer や IDE による Web アプリケーションの作成については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

(1)　Web アプリケーションのデプロイ

Web アプリケーションのデプロイ方法は，次の 2 種類あります。

- WAR ファイルのコピーによるデプロイ
- ディレクトリのコピーによるデプロイ

それぞれのデプロイ方法について説明します。また，ルートコンテキストを使用し，ドメイン名を指定して Web アプリケーションのトップページを表示させる方法についても説明します。

(a)　WAR ファイルのコピーによるデプロイ

WAR ファイルのコピーによるデプロイの手順を次に示します。

1. WAR ファイルを次に示す場所にコピーします。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥webapps"
   - UNIX の場合
     /opt/Cosminexus/CC/web/containers/<サーバ名称>/webapps
2. cjstartweb コマンドを実行して，Web コンテナサーバを起動します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstartweb <サーバ名称>"
   - UNIX の場合
     /opt/Cosminexus/CC/web/bin/cjstartweb <サーバ名称>

   これによって，WAR ファイルが自動展開され，Web アプリケーションを利用できるようになります。デプロイ後は，手順 1.で示した場所に WAR ファイル名の拡張子（.war）を除いた名前のディレクトリが作成されます。このディレクトリ名が，コンテキスト名として使用されます。

(b)　ディレクトリのコピーによるデプロイ

ディレクトリのコピーによるデプロイの手順を次に示します。

1. 次に示す場所に Web アプリケーションのディレクトリ階層をコピーします。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥webapps"
   - UNIX の場合

/opt/Cosminexus/CC/web/containers/<サーバ名称>/webapps

2. cjstartweb コマンドを実行して，Web コンテナサーバを起動します。

   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstartweb <サーバ名称>"
   - UNIX の場合
     /opt/Cosminexus/CC/web/bin/cjstartweb <サーバ名称>

   これによって，Web アプリケーションを利用できるようになります。コンテキスト名としては，コピーされたディレクトリ名がそのまま使用されます。

(c)　ドメイン名指定でのトップページの表示

デプロイした Web アプリケーションにアクセスするとき，URL にドメイン名を指定するだけで，Web アプリケーションのトップページを表示させるには，次の設定が必要です。

- リダイレクタの URL マッピング定義で，ルートコンテキストにリクエストが振り分けられるように設定する必要があります。リダイレクタでの設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.2　Web サーバ（リダイレクタ）によるリクエストの振り分け」を参照してください。
  なお，サーブレットエンジンモードの場合は，Web アプリケーションをデプロイするときにルートコンテキストを指定します。
- index.html や index.jsp などの Web アプリケーションのトップページをルートコンテキストに配置する必要があります。Web アプリケーションのコンテキストルートを「ROOT」にしてデプロイしてください。

  WAR ファイルのコピーによるデプロイの場合

  WAR ファイル名を「ROOT.war」にしてデプロイしてください。
  なお，拡張子（.war）を除いた WAR の名前はすべて大文字の「ROOT」にします。「Root」や「root」のように小文字を含んでいる場合は，拡張子（.war）を除いた WAR の名前がコンテキストルートとなります。例えば，WAR ファイル名が「Root.war」の場合，ルートコンテキストではなく，コンテキストルートが「Root」のコンテキストになります。

  ディレクトリのコピーによるデプロイの場合

  ディレクトリ名を「ROOT」にしてデプロイしてください。
  なお，ディレクトリ名「ROOT」はすべて大文字にします。「Root」や「root」のように小文字を含んでいる場合は，指定したディレクトリの名前がコンテキストルートとなります。例えば，ディレクトリ名が「Root」の場合，ルートコンテキストではなく，コンテキストルートが「Root」のコンテキストになります。

(2)　Web アプリケーションのアンデプロイ

Web アプリケーションのアンデプロイ方法は，次の 2 種類あります。

- WAR ファイルの削除によるアンデプロイ
- ディレクトリの削除によるアンデプロイ

それぞれのアンデプロイ方法について説明します。

(a)　WAR ファイルの削除によるアンデプロイ

WAR ファイルの削除によるアンデプロイの手順を次に示します。

1. cjstopweb コマンドを実行して，Web コンテナサーバを停止します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstopweb <サーバ名称>"
   - UNIX の場合
     /opt/Cosminexus/CC/web/bin/cjstopweb <サーバ名称>
2. 該当する WAR ファイル，および次に示す場所に自動展開されたディレクトリを削除します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥webapps"
   - UNIX の場合
     /opt/Cosminexus/CC/web/containers/<サーバ名称>/webapps
3. Web コンテナサーバを再起動します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstartweb <サーバ名称>"
   - UNIX の場合
     /opt/Cosminexus/CC/web/bin/cjstartweb <サーバ名称>

これによって，Web アプリケーションのアンデプロイが有効になります。

(b)　ディレクトリの削除によるアンデプロイ

ディレクトリの削除によるアンデプロイの手順を次に示します。

1. cjstopweb コマンドを実行して，Web コンテナサーバを停止します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstopweb <サーバ名称>"
   - UNIX の場合
     /opt/Cosminexus/CC/web/bin/cjstopweb <サーバ名称>
2. 次に示す場所にコピーしたディレクトリを削除します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥webapps"
   - UNIX の場合
     /opt/Cosminexus/CC/web/containers/<サーバ名称>/webapps
3. Web コンテナサーバを再起動します。
   - Windows の場合
     "<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstartweb <サーバ名称>"
   - UNIX の場合
     /opt/Cosminexus/CC/web/bin/cjstartweb <サーバ名称>

これによって，Web アプリケーションのアンデプロイが有効になります。

(3)　Web アプリケーションに組み込むサーブレットフィルタの設定

アプリケーションサーバでは，アプリケーションサーバ独自の機能を使用するためのサーブレットフィルタを提供しています。アプリケーションサーバが提供するサーブレットフィルタを built-in フィルタといいます。

アプリケーションサーバで提供している built-in フィルタのうち, サーブレットエンジンモードで使用できるフィルタは，HTTP レスポンス圧縮フィルタだけです。

各 built-in フィルタの処理内容，および built-in フィルタの組み込み順序については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.9.1　アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ)」を参照してください。

サーブレットエンジンモードの場合は，Web アプリケーションの DD（web.xml）を直接編集してフィルタを組み込む必要があります。built-in フィルタの組み込みについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.10.3　HTTP レスポンス圧縮フィルタを使用するアプリケーションの実装」を参照してください。

3.7.8　Web コンテナサーバの作業ディレクトリ

Web コンテナサーバは作業ディレクトリ中にサーバ名や Web アプリケーションのコンテキストルート名，WAR ファイル中のファイルなどを用いてディレクトリやファイルを生成します。ディレクトリ名やファイル名，パス全体の長さには上限があり，作業ディレクトリ全体がパス全体の上限に収まらないと Web コンテナサーバは正常に動作できません。ここでは，Web コンテナサーバの作業ディレクトリの構成と作業ディレクトリのパス長の見積もり式を説明します。

(1)　作業ディレクトリの構成

Web コンテナサーバの作業ディレクトリは，Web コンテナサーバが動作する上で必要な作業領域です。この領域には，ユーザ定義ファイル，ログ，Web アプリケーションなどの情報が格納されます。

Web コンテナサーバには，次の作業ディレクトリがあります。

- Web アプリケーション
- JSP 用一時ディレクトリ

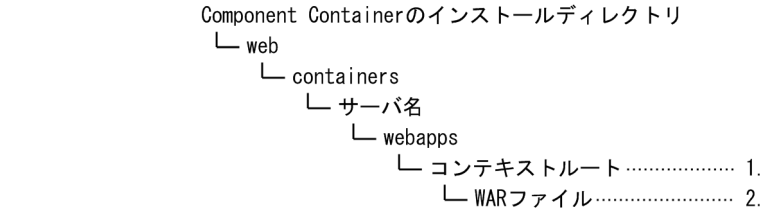
上記の作業ディレクトリの構成についてそれぞれ説明します。

ディレクトリ構成を図で示したあと，各ディレクトリを説明します。なお，図中の番号と説明の番号は対応しています。

(a)　Web アプリケーション

Web アプリケーションのディレクトリ構成を次の図に示します。

図 3-5　Web アプリケーションのディレクトリ構成

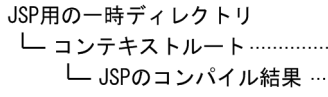


1. WAR ファイル名の拡張子（.war）を除いた名前のディレクトリが作成されます。
2. WAR ファイルに含まれているファイルがここに解凍されます。

(b)　JSP 用一時ディレクトリ

JSP 用一時ディレクトリのディレクトリ構成を次の図に示します。

図 3-6　JSP 用一時ディレクトリのディレクトリ構成



1. WAR ファイル名の拡張子（.war）を除いた名前のディレクトリが作成されます。
2. JSP をコンパイルして作成されたクラスファイルが格納されます。JSP のファイル名に英数字以外の文字が含まれている場合は，それぞれ 6 文字の文字列に置き換えられます。また，ファイル名が数字で始まる場合は 1 文字付加されます。

(2)　作業ディレクトリのパス長の見積もり式

作業ディレクトリのパス長の見積もり式は次のようになります（単位：バイト）。これらの式がすべて満たされるように，各種パラメタを調整してください。なお，見積もり式中の「最大パス長」※とは，OS で規定されているパス長の制限のことです。OS のパス長の制限については，ご使用の OS のドキュメントなどで確認してください。

- A+B+C+D+25≦最大パス長
- A'+C+E+2≦最大パス長

注※

ご使用の OS が Windows の場合，最大パス長は 259 バイトです。

（凡例）

A：インストールディレクトリのパスの長さ

A'：JSP 用一時ディレクトリまでのパスの長さ

B：最も長いサーバ名の長さ

C：最も長いコンテキストルートの長さ

D：WAR ファイル内で最も長いパスの長さ

E：最も長い JSP のファイル名（WAR ファイル内のパスを含む。英数字以外を含む場合は英数字以外の文字×6 で換算。JSP ファイルが数字で始まる場合は 1 を加算）の長さ

3.8　サーブレットエンジンモードでのシステム運用

ここでは，サーブレットエンジンモードを使用してシステムを運用する場合の方法について説明します。

3.8.1　Component Container 管理者による運用に関する留意事項

UNIX で Component Container 管理者を設定している場合の留意事項について説明します。
Component Container 管理者とは，UNIX を使用している場合に，Web コンテナサーバの構築・運用に関する操作を実行できる権限を持つユーザです。Component Container 管理者を設定することで，root 権限を持たないユーザでも， Web コンテナサーバの起動・停止やサーバ管理コマンドの実行など，システム運用時の操作を実行できるようになります。

! 注意事項

Component Container 管理者を設定している場合，Component Container 管理者が実行できる操作は，一部の操作を除き，スーパーユーザでは操作できなくなるので注意してください。詳細については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.4　Component Container 管理者を設定するときの注意事項（UNIX の場合)」を参照してください。

なお，Component Container 管理者の設定，および Component Container 管理者を設定している場合にパーミッションが変更されるディレクトリについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.4　Component Container 管理者を設定するときの注意事項（UNIX の場合)」を参照してください。

Component Container 管理者が実施できるシステム運用時の操作について説明します。

(1)　Component Container 管理者が実施できるシステム運用時の操作

Component Container 管理者を設定した場合に，スーパーユーザおよび Component Container 管理者が実施できるシステム運用時の操作を次の表に示します。

表 3-37　Component Container 管理者が実施できるシステム運用時の操作

操作	スーパーユーザ	Component Container 管理者
Web コンテナサーバの起動／停止 （cjstartweb コマンド， cjstopweb コマンド）	×	○
サーバ管理コマンドの実行	○	○
スレッドダンプの取得（cjdumpweb コマンド）	○	○
OS 状態情報の取得（cjgetsysinfo コマンド）	○	○

（凡例）○：できる　×：できない

なお，Component Container 管理者を設定しない場合は，スーパーユーザが運用管理者となるため，Component Container 管理者の操作もスーパーユーザが実施します。

3.8.2　システムの起動

ここでは，サーブレットエンジンモードでのシステムの起動の手順について説明します。

なお，データベース，LDAP ディレクトリサーバなどの関連製品は起動済みであることを前提に説明します。データベース，LDAP ディレクトリサーバなどを使用している場合は，必要に応じてあらかじめ起動しておいてください。システムの起動手順を次に示します。

1. PRF デーモンを起動します。
   cprfstart コマンドで PRF デーモンを起動します。起動方法については，「(1)　PRF デーモンの起動」を参照してください。
2. Web コンテナサーバを起動します。
   cjstartweb コマンドで Web コンテナサーバを起動します。起動方法については，「(2)　Web コンテナサーバの起動」を参照してください。
3. Web サーバを起動します。
   起動方法については，ご使用の Web サーバのマニュアルを参照してください。
4. Web アプリケーションの動作を確認します。
   システムが起動できたら，Web アプリケーションが正常に動作することを確認してください。

(1)　PRF デーモンの起動

PRF デーモンの起動手順を次に示します。

1. cd コマンドでカレントディレクトリを移動します。
   - Windows の場合

```
cd "<製品のインストールディレクトリ>¥PRF¥bin"
```

   - UNIX の場合

```
# cd /opt/Cosminexus/PRF/bin
```

2. cprfstart コマンドで PRF デーモンを起動します。
   - Windows の場合

```
cprfstart
```

   - UNIX の場合

```
# cprfstart
```

なお，cprfstart コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cprfstart（PRF デーモンの開始)」を参照してください。

(2)　Web コンテナサーバの起動

Web コンテナサーバの起動方法について説明します。

Web コンテナサーバを起動するには，次に示すコマンドを実行します。

- Windows の場合

```
"<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstartweb サーバ名称"
```

  なお，Windows の場合，コマンドの実行には，Administrator 権限が必要です。

- UNIX の場合

```
"/opt/Cosminexus/CC/web/bin/cjstartweb <サーバ名称>"
```

なお，UNIX の場合，コマンドの実行には，root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。

Web コンテナサーバのプロセス名は「cjstartweb」です。cjstopweb コマンドで Web コンテナサーバを停止するまで，このプロセスは存在します。

同一ホストで複数の Web コンテナを同時に起動する場合のポート番号については，Web コンテナサーバのセットアップで設定します。詳細については，「3.7.3　Web コンテナサーバのセットアップ」を参照してください。

なお，SecurityManager による Web コンテナの保護をするには，-security オプションを追加してください。

3.8.3　システムの停止

ここでは，サーブレットエンジンモードのシステムの停止手順について説明します。

なお，データベース，LDAP ディレクトリサーバなどを使用している場合は，必要に応じて停止してください。システムの停止手順を次に示します。

1. **Web サーバを停止します。**
   停止方法については，ご使用の Web サーバのマニュアルを参照してください。
2. **Web コンテナサーバを停止します。**
   cjstopweb コマンドで Web コンテナサーバを停止します。停止方法については，「(1)　Web コンテナサーバの停止」を参照してください。
3. **PRF デーモンを停止します。**
   cprfstop コマンドで PRF デーモンを停止します。停止方法については，「(2)　PRF デーモンの停止」を参照してください。

(1)　Web コンテナサーバの停止

Web コンテナサーバの停止方法について説明します。

Web コンテナサーバを停止するには，次に示すコマンドを実行します。

- Windows の場合

```
"<製品のインストールディレクトリ>¥CC¥web¥bin¥cjstopweb サーバ名称"
```

なお，Windows の場合，コマンドの実行には，Administrator 権限が必要です。

- UNIX の場合

```
"/opt/Cosminexus/CC/web/bin/cjstopweb <サーバ名称>"
```

なお，UNIX の場合，コマンドの実行には，root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。

Web コンテナサーバを停止しても，Web コンテナサーバの停止処理が完了しない場合があります。この場合，Web コンテナサーバを強制的に停止する必要があります。Web コンテナサーバを強制停止する場合，cjstopweb コマンドに-f オプションまたは-fd オプションを指定して実行します。-fd オプションを指定すると，Web コンテナサーバの障害情報を取得したあと，Web コンテナサーバを強制停止します。

(2)　PRF デーモンの停止

PRF デーモンの停止手順を次に示します。

1. cd コマンドでカレントディレクトリを移動します。

- Windows の場合

```
cd ”<製品のインストールディレクトリ>¥PRF¥bin”
```

- UNIX の場合

```
# cd /opt/Cosminexus/PRF/bin
```

2. cprfstop コマンドで PRF デーモンを停止します。

cprfstart コマンドを実行したユーザが実行してください

- Windows の場合

```
cprfstop
```

- UNIX の場合

```
# cprfstop
```

PRF デーモンを停止しても，PRF デーモンの停止処理が完了しない場合があります。この場合，PRF デーモンを強制的に停止する必要があります。PRF デーモンを強制停止する場合，cprfstop コマンドに-Force オプションを指定して実行します。

なお，cprfstop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cprfstop（PRF デーモンの停止)」を参照してください。

3.8.4　システムの処理性能の解析

ここでは，サーブレットエンジンモードで動作するシステムでの処理性能の解析方法について説明します。

アプリケーションサーバで構築したシステムの処理性能は，クライアントからデータベースなどの EIS まで，およびその処理結果がクライアントに返却されるまでのリクエストの一連の処理で，リダイレクタや Web コンテナなどから出力されるトレース情報を基に解析できます。トレース情報を出力するリダイレクタや Web コンテナなどを，**機能レイヤ**といいます。サーブレットエンジンモードで動作するシステムでのリクエストの処理過程では，次に示す機能レイヤからトレース情報が出力されます。

- リダイレクタ
- Web コンテナ

各機能レイヤでは決まった処理のポイント（**トレース取得ポイント**）でトレース情報を出力します。各機能レイヤが出力するトレース情報は，パフォーマンストレーサ（Performance Tracer）の PRF デーモンによって，トレースファイル（**PRF トレースファイル**）に出力されます。PRF デーモンにはトレース取得レベル（標準または詳細）を設定できます。PRF デーモンに設定するトレース取得レベルを **PRF トレース取得レベル**といいます。各機能レイヤのトレース取得ポイントおよび PRF トレース取得レベルの詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「8.　性能解析トレースのトレース取得ポイントと PRF トレース取得レベル」を参照してください。

なお，各機能レイヤのトレース情報はバイナリ形式で PRF トレースファイルに出力されます。システムの処理性能を解析する際は，PRF トレースファイルをテキスト形式（CSV 形式）に変換したファイル（**性能解析トレースファイル**）を使用します。

サーブレットエンジンモードの場合，システムの処理性能を解析する作業は，パフォーマンストレーサのコマンドで実行します。

システムの処理性能を解析する場合に実施する作業を次に示します。詳細については，参照先の説明を参照してください。

表 3-38　システムの処理性能を解析する場合に実施する作業（サーブレットエンジンモードの場合）

作業の目的	作業の概要	パフォーマンストレーサのコマンド	参照先マニュアル	参照先
性能解析トレースファイルを収集する	PRF トレースファイルを出力して，CSV 形式に変換します。	cprfflush cprfed	－	－
収集した性能解析トレースファイルを利用してシステムの処理性能を解析する	Web サーバのレスポンスタイムを解析します。	－※	アプリケーションサーバ機能解説 保守／移行編	7.7.2
	アプリケーションサーバ内でのリクエストの処理状況を調査します。	－※		7.7.3
	セッションのライフサイクルを調査します。	－※		7.7.4
	タイムアウトしたトランザクションを特定します。	－※		7.7.5
	タイムアウトしたリクエストを特定します。	－※		7.7.6
	ルートアプリケーション情報を利用してログを調査します。	－※		7.7.7
	データベースとの接続中にトラブルが発生したコネクションを特定します。	－※		7.7.8

（凡例）－：該当しない

注※

cprfed コマンドを使用して出力された性能解析トレースファイルを使用します。

cprfflush の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cprfflush (バッファ内容の強制ファイル出力)」を参照してください。cprfed の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cprfed（性能解析トレース情報の編集出力)」を参照してください。

3.8.5　トラブルシューティング

ここでは，サーブレットエンジンモードでシステムを運用している場合に発生したトラブルに対処する方法について説明します。

(1)　対処の手順（サーブレットエンジンモードの場合）

ここでは，アプリケーションサーバのシステムの運用中に，トラブルが発生した場合の対処の手順について説明します。

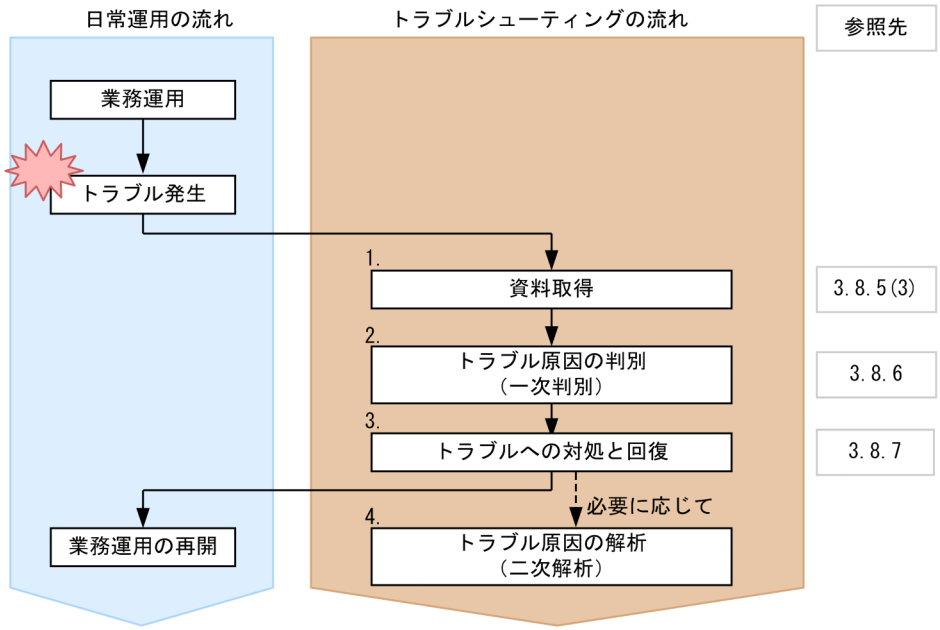
なお，トラブル発生時に取得する情報のうち，次に示す情報を取得するためには，運用開始前の準備が必要です。

- JavaVM ログ
  このファイルには，JavaVM の GC のログも出力されます。
- ユーザダンプ（Windows の場合）
- core ダンプ（UNIX の場合）
- OS の統計情報（Windows の場合）

これらの情報については，必要に応じてシステム構築時に取得準備をしてください。詳細については，「3.7.5　トラブルシューティングの資料取得の設定」を参照してください。

システムの運用中にトラブルが発生した場合，次の図に示す手順で対処します。

図 3-7　トラブル発生時の対処の流れ



1. 資料取得

トラブルシューティングに必要な資料をすべて取得します。

取得が必要な資料については，「(2)　取得が必要な資料の種類」を参照してください。なお，メモリダンプは，Web コンテナサーバを再起動する場合にだけ必要な資料です。

また，資料の取得方法については，「(3)　資料の取得」を参照してください。

2. トラブル原因の判別（一次判別）

トラブルの原因を判別します。判別には，取得した資料を使用します。資料の調査方法については，「3.8.6　資料の調査」を参照してください。

3. トラブルへの対処と回復

トラブルに対処して，回復します。

なお，トラブルの種類ごとの対処方法については，「3.8.7　トラブルへの対処と回復」を参照してください。

4. トラブル原因の解析（二次解析）

必要に応じてトラブル原因をさらに分析します。なお，この解析作業は保守員が実施します。

(2)　取得が必要な資料の種類

ここでは，トラブル発生時に取得が必要な資料の種類を示します。

トラブル発生時に取得が必要な資料の種類については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「2.4　取得が必要な資料の種類」を参照してください。

ただし，サーブレットエンジンモードの場合，snapshot ログの取得方法と，デフォルトの設定で snapshot ログとして取得できる情報の種類が異なります。

また，資料を個別で取得するとき，次の資料の取得内容または取得方法が，運用管理ポータルを使用して構築したシステムの場合と異なります。

- アプリケーションサーバのログ
- 性能解析トレース

(3)　資料の取得

ここでは，トラブル発生時の資料の取得方法について説明します。

トラブルシュートに必要な資料は，次のどちらかの方法で取得できます。

- snapshotlog コマンドを利用して一括取得する
- それぞれの情報を個別に収集する

snapshot ログは snapshotlog コマンドを実行して収集します。snapshotlog コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「snapshotlog（snapshot ログの収集）」を参照してください。

それぞれの情報を個別に収集する場合，次の資料を取得します。

- アプリケーションサーバのログ
  - Component Container のログ
  - Performance Tracer のログ
  - DABroker Library のログ
  - Web アプリケーションのログ
- 性能解析トレース
- JavaVM のスレッドダンプ
- JavaVM の GC ログ
- メモリダンプ
- アプリケーションサーバの定義情報
- Web コンテナサーバの作業ディレクトリの内容
- Web サーバログ
- そのほかの情報

それぞれの情報の取得方法については「3.8.9　資料の取得方法」を参照してください。

3.8.6　資料の調査

ここでは，トラブルシューティングで，取得した資料を調査して，トラブルの原因を判別する方法について説明します。

トラブルシューティングに必要な資料の調査方法は，J2EE サーバモードの場合と同様です。

資料の調査方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「5.　トラブルの分析」を参照してください。

3.8.7　トラブルへの対処と回復

ここでは，次のようなトラブルが発生した場合に実施する，トラブルの内容に応じた対処方法および回復方法について説明します。

- 構成ソフトウェアのプロセスが異常終了した場合
- データベースと接続中にトラブルが発生した場合
- JavaVM が異常終了した場合

(1)　構成ソフトウェアのプロセスが異常終了した場合（サーブレットエンジンモードの場合）

システムの運用中に構成ソフトウェアが異常終了して再起動を実行する場合，異常終了した構成ソフトウェアの起動順序に依存関係があるときは，前提となる構成ソフトウェアが起動していることを確認してから再起動する必要があります。前提となる構成ソフトウェアは，システムの運用形態によって異なります。以降で，構成ソフトウェアの起動順序の依存関係，および構成ソフトウェアの再起動方法について説明します。

(a)　構成ソフトウェアの起動順序の依存関係

アプリケーションサーバで構築したシステムを構成する構成ソフトウェアの起動順序の依存関係について説明します。

サーブレットエンジンモードの場合のプロセスの依存関係を次に示します。

表 3-39　サーブレットエンジンモードの場合のプロセスの依存関係

プロセスの種類	前提プロセス
PRF デーモン	－
Web コンテナサーバ	PRF デーモン
Web サーバ	－

（凡例）－：前提プロセスがない

(b)　プロセスの再起動方法

システムでプロセスが異常終了した場合のプロセスの再起動方法について説明します。

サーブレットエンジンモードの場合のプロセスの再起動（回復）の手順は次のとおりです。

表 3-40　サーブレットエンジンモードの場合の再起動の手順

異常プロセス	開始コマンド	再起動（回復）手順
データベースサーバ	－	DB サーバを再起動してください。

異常プロセス	開始コマンド	再起動（回復）手順
PRF デーモン	cprfstart	パフォーマンストレーサを再起動してください。
Web コンテナサーバ	cjstartweb	Web コンテナサーバを再起動してください。
Web サーバ	−	Web サーバを再起動してください。

（凡例）

−：使用する製品によって開始コマンドが異なる，または該当する開始コマンドがない

(2)　データベースと接続中にトラブルが発生した場合（サーブレットエンジンモードの場合）

データベースとの接続中に，アプリケーションサーバの構成ソフトウェアである DABroker Library でトラブルが発生した場合，DABroker Library によってデータベースとの接続は切り離されます。また，DABroker Library を強制終了した場合も同様です。

これらのトラブルへの対処方法は，J2EE サーバモードの場合と同様です。対処方法については，「4.6.7 データベースと接続中にトラブルが発生した場合」を参照してください。

(3)　JavaVM が異常終了した場合（サーブレットエンジンモードの場合）

JavaVM では，次のようなトラブルが発生する場合があります。

- C ヒープ不足
- 内部論理エラー
- OutOfMemoryError

これらのトラブルへの対処方法は，J2EE サーバモードの場合と同様です。対処方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「2.5.7　JavaVM が異常終了した場合」を参照してください。

3.8.8　トラブルシューティングに関連する留意事項

トラブルシューティングに関連する留意事項は，J2EE サーバモードの場合と同様です。留意事項については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「2.6　トラブルシューティングに関連する留意事項」を参照してください。

3.8.9　資料の取得方法

(1)　Component Container のログ

Component Container のログとして取得する資料の出力先を次に示します。それぞれの資料の詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「5.2　アプリケーションサーバのログ」を参照してください。

表 3-41　Web コンテナサーバおよびリダイレクタのログの出力先（デフォルト）

分類	内容	ログ出力先およびログファイル名※1	デフォルトのサイズ×面数	チャネル名
メッセージログ	稼働ログ	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjweb_message[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjweb_message[n].log	1MB×2	WebMessageLogFile
	リダイレクタのメッセージログ	ディレクトリ※2 • Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥redirector¥logs • UNIX の場合 /opt/Cosminexus/CC/web/redirector/logs	4MB×5	−
		ファイル名 • Windows の場合 isapi_redirect[n].log（Microsoft IIS） hws_redirect[n].log（HTTP Server） • UNIX の場合 hws_redirect[n].log（HTTP Server）		
ユーザログ	Web サーブレットログ※3	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥web_servlet[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/web_servlet[n].log	4MB×4	WebServletLogFile
	ユーザ出力ログ※4	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥user_out[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/user_out[n].log	1MB×2	UserOutLogFile
	ユーザエラーログ※4	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥user_err[n].log • UNIX の場合	1MB×2	UserErrLogFile

分類	内容	ログ出力先およびログファイル名※1	デフォルトのサイズ×面数	チャネル名
ユーザログ	ユーザエラーログ※4	/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/user_err[n].log	1MB×2	UserErrLogFile
	JavaVM の保守情報および GC のログ	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥javalog[nn].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/javalog[nn].log	256KB×4	−
例外ログ	障害発生時の例外情報	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjweb_exception[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjweb_exception[n].log	12MB×4	WebExceptionLogFile
保守用ログ	保守情報	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjweb_maintenance[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjweb_maintenance[n].log	1MB×2	WebMaintenanceLogFile
	Web コンテナの保守情報	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjwebcontainer[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjwebcontainer[n].log	1MB×2	WebContainerLogFile
	起動プロセス標準出力情報※5	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjstdout.log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjstdout.log	−	−

分類	内容	ログ出力先およびログファイル名※1	デフォルトのサイズ×面数	チャネル名
保守用ログ	起動プロセス標準エラー情報※5	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjstderr.log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjstderr.log	−	−
	終了プロセス情報	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjweb_shutdown[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjweb_shutdown[n].log	1MB×2	WebShutdownLogFile
	Web コンテナ保守用トレースログ	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥cjweb_access[n].log • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/cjweb_access[n].log	2MB×16	WebAccessLogFile
	リダイレクタの保守用トレースログ	ディレクトリ※2 • Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥redirector¥logs • UNIX の場合 /opt/Cosminexus/CC/web/redirector/logs	16MB×2	−
		ファイル名※2 • Windows の場合 hws_rd_trace[n].log（HTTP Server） isapi_rd_trace[n].log（Microsoft IIS） • UNIX の場合 hws_rd_trace[n].log		

（凡例）−：該当しない

注

チャネル名は，ログの出力先を識別する名称です。ログの属性（サイズ，面数）を変更する場合のキー値として使用します。

注※1

[n]：ログファイル名の［n］部分には，面の番号（1 から面数（最大 16）まで）が付きます。また，［nn］の部分には 01～99 の通し番号が付きます。

注※2

ディレクトリ，ファイル名は，isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル），または mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）で設定できます。リダイレクタのログ取得の設定については，「3.7.5　トラブルシューティングの資料取得の設定」を参照してください。isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.2　isapi_redirect.conf（Microsoft IIS 用リダイレクタ動作定義ファイル)」を参照してください。mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「9.3　mod_jk.conf（HTTP Server 用リダイレクタ動作定義ファイル)」を参照してください。

注※3

サーブレット，JSP で発生した例外のスタックトレースについても出力されます。

注※4

標準出力にも出力されます。また，サーブレット，JSP での System.out，および System.err への出力はユーザ出力ログ，およびユーザエラーログに出力されます。

注※5

起動プロセス情報だけを取得するログです。主に Web コンテナサーバの起動または終了時に出力されるため，オンライン中に出力されることはほとんどありません。ファイルのサイズが上限に達したときは，<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs 下（Windows の場合），または/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs 下（UNIX の場合）の cjstdout_save.log または cjstderr_save.log に退避されます。すでに cjstdout_save.log または cjstderr_save.log があったときは，上書きされます。

! 注意事項

Web サーバとして Microsoft IIS を使用している場合，リダイレクタのログファイルは，Web サーバへの初回アクセス時に，isapi_redirect.conf で指定したファイルサイズで，指定した面数分，すべて作成されます。このとき，トレースファイルは空白(0x20)で初期化されます。デフォルトのファイルサイズと面数で運用している場合，ログの出力先に指定したディレクトリに，52 メガバイトのディスク容量が必要になります。

なお，Web コンテナサーバおよびリダイレクタのログの出力先を変更している場合は，次の表に示す，ログの出力先を設定するユーザ定義ファイルを参照して出力先を確認してください。

表 3-42　ログの出力先を設定するユーザ定義ファイル（サーブレットエンジンモードの場合）

分類	ユーザ定義ファイルの格納場所
Web コンテナサーバ	格納場所は，usrconf.cfg ファイルの web.server.log.directory キーで指定します。キーのデフォルトは，「<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs」（Windows の場合），または「/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs」（UNIX の場合）です。
リダイレクタ（Windows の場合）	HTTP Server の場合 ［メッセージログ］ <製品のインストールディレクトリ>¥CC¥web¥redirector¥mod_jk.conf の JkLogFileDir キー キーのデフォルトは，「<製品のインストールディレクトリ>¥CC¥web¥redirector¥logs」です。 ［保守用トレースログ］ <製品のインストールディレクトリ>¥CC¥web¥redirector¥mod_jk.conf の JkTraceLogFileDir キー

分類	ユーザ定義ファイルの格納場所
リダイレクタ（Windows の場合）	キーのデフォルトは，「<製品のインストールディレクトリ>¥CC¥web¥redirector¥logs」です。 Microsoft IIS の場合 ［メッセージログ］ <製品のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.conf の log_file_dir キー キーのデフォルトは，「<製品のインストールディレクトリ>¥CC¥web¥redirector¥logs」です。 ［保守用トレースログ］ <製品のインストールディレクトリ>¥CC¥web¥redirector¥isapi_redirect.conf の trace_log_file_dir キー キーのデフォルトは，「<製品のインストールディレクトリ>¥CC¥web¥redirector¥logs」です。
リダイレクタ（UNIX の場合）	メッセージログ /opt/Cosminexus/CC/web/redirector/mod_jk.conf の JkLogFileDir キー キーのデフォルトは，「/opt/Cosminexus/CC/web/redirector/logs」です。 保守用トレースログ /opt/Cosminexus/CC/web/redirector/mod_jk.conf の JkTraceLogFileDir キー キーのデフォルトは，「/opt/Cosminexus/CC/web/redirector/logs」です。

ログの出力先の変更方法など，トラブルシューティングの資料取得の設定については，「3.7.5　トラブルシューティングの資料取得の設定」を参照してください。

(a)　Performance Tracer のログ

Performance Tracer のログの出力先を次に示します。

それぞれの資料の詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「5.2 アプリケーションサーバのログ」を参照してください。

表 3-43　Performance Tracer のログの出力先

内容	出力先ディレクトリ
PRF デーモンおよび PRF コマンドのログ	• Windows の場合 <環境変数 PRFSPOOL の設定ディレクトリ>¥log¥<PRF 識別子>¥ctmlog[n] • UNIX の場合 $PRFSPOOL/log/<PRF 識別子>/ctmlog[n]
モジュールトレース	• Windows の場合 <環境変数 PRFSPOOL の設定ディレクトリ>¥utt¥umt • UNIX の場合 $PRFSPOOL/utt/umt

内容	出力先ディレクトリ
構造化例外発生ログ	• Windows の場合 <環境変数 PRFSPOOL の設定ディレクトリ>¥oslttrc • UNIX の場合 $PRFSPOOL/oslttrc

注

[n]には「01」または「02」が表示されます。

(b)　DABroker Library のログ

それぞれの資料の詳細については，「4.6.3　DABroker Library のスプール情報の内容」を参照してください。

DABroker Library では，トラブルシューティングに使用するログとして，スプール情報を取得します。

次のディレクトリ下のすべてのファイルをコピーして保存してください。

- Windows の場合
  <DABroker Library の運用ディレクトリ>¥spool
- UNIX の場合
  <DABroker Library の運用ディレクトリ>/spool

参考

DABroker Library では，スプール情報のほかに次の JDBC ドライバのトレースも取得できます。

- JDBC インタフェースメソッドトレース
- DABroker Library とのイベントトレース

ただし，JDBC ドライバで次のトレースを取得するためには，アプリケーション開発時にあらかじめ指定されている必要があります。トレースを取得するための情報の設定については，「4.7　DABroker Library を使用したデータベース接続の実装」を参照してください。

(c)　Web アプリケーションのログ

Web アプリケーションのログは，トレース共通ライブラリ形式で出力するように設定している場合に取得できます。

Web アプリケーションのログの出力先については，usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル）の ejbserver.application.userlog.CJLogHandler.<ハンドラ名>.path キーの値を参照してください。なお，<ハンドラ名>には，キーの値を区別するためのハンドラ名が指定されています。

usrconf.properties ファイルの格納場所を次に示します。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.properties
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.properties

なお，Web アプリケーションのユーザログ出力の設定については，「3.7.5(10)　Web アプリケーションのユーザログ出力の設定」を参照してください。

(2)　性能解析トレースの取得（サーブレットエンジンモードの場合）

性能解析トレースの収集方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.　性能解析トレースを使用した性能解析」を参照してください。セッショントレースも同様に取得できます。

性能解析トレースの格納場所は次のとおりです。

- Windows の場合
  <環境変数 PRFSPOOL の設定ディレクトリ>¥utt¥prf¥<PRF 識別子>
- UNIX の場合
  $PRFSPOOL/utt/prf/<PRF 識別子>

(3)　JavaVM のスレッドダンプの取得（サーブレットエンジンモードの場合）

ここでは，JavaVM のスレッドダンプを取得する方法について説明します。

なお，Web コンテナサーバの起動オプションとして-XX:+HitachiThreadDump が設定されている場合，**拡張スレッドダンプ**を取得できます。-XX:+HitachiThreadDumpToStdout が設定されている場合，スレッドダンプは標準出力にも出力されます。

また，-XX:+HitachiOutOfMemoryAbort が設定されている場合に，-XX:+HitachiOutOfMemoryAbortThreadDump が設定されていると，OutOfMemoryError によって強制終了した時に，スレッドダンプが出力されます。ただし，次の場合は除きます。

- J2SE クラスライブラリ内で C ヒープ不足が発生した場合
- JavaVM の処理中に C ヒープ不足が発生した場合

cjstartweb プロセスが存在する場合のスレッドダンプの取得方法を次に示します。

- Windows の場合
  cjstartweb プロセスが存在する場合，次を実行します（3 秒おきに 10 回程度）。

```
<製品のインストールディレクトリ>¥CC¥web¥bin¥cjdumpweb <Webコンテナサーバ名>
```

- UNIX の場合
  cjstartweb プロセスが存在する場合，cjdumpweb コマンドまたは kill コマンドを使用して Web コンテナサーバのスレッドダンプを取得します（3 秒おきに 10 回程度）。
  - cjdumpweb コマンドを使用する場合

```
/opt/Cosminexus/CC/web/bin/cjdumpweb <Webコンテナサーバ名>
```

  - kill コマンドを使用する場合

```
ps -ef | grep cjstartweb
kill -3 <Webコンテナサーバのプロセス ID>
```

  スレッドダンプは，「/opt/Cosminexus/CC/web/containers/<サーバ名称>/javacore*.txt」，およびサーバ標準出力ログ（cjstdout.log）に出力されます。

また，javacore<プロセス番号>.<コマンド実行日時>.txt ファイルの出力先は，環境変数 JAVACOREDIR で変更できます。ただし，指定したディレクトリへの書き込みに失敗した場合は，カレントディレクトリに出力されます。このディレクトリにも出力できなかった場合は，標準エラー出力にだけ出力されます。

参考

スレッドダンプが出力されると，標準出力に，次のメッセージが出力されます。そのあと，java プログラムの実行が継続されます。このメッセージは，-XX:+HitachiThreadDumpToStdout の設定とは関係なく出力されます。

```
Writing Java core to <ファイル名（フルパス）>...OK
```

(4)　JavaVM の GC ログの取得（サーブレットエンジンモードの場合）

JavaVM の GC ログは，JavaVM または Web コンテナサーバ起動前にログ出力先が設定されている場合にだけ取得できます。

ユーザ定義ファイルを使用して設定している場合は，usrconf.cfg ファイルの web.server.log.directory のキーを確認してください。

また，実行中の Java プロセスで GC を発生させたい場合は，javagc コマンドを実行してください。**javagc コマンド**は，トラブル発生時の要因調査のほか，1 トランザクション当たりのメモリ使用量の測定，メモリリークの調査，アプリケーションのデバッグなどの目的でも使用できます。

実行中の Java プロセスを指定して GC を発生させる場合，javagc コマンドは，次の形式で実行します。なお，これ以外に指定できるオプションについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「javagc（GC の強制発生)」を参照してください。

```
javagc -p プロセスID
```

実行結果として，次のログが出力されます。なお，これは，オプションとして，-XX:+HitachiVerboseGCPrintCause が指定されている場合の例です。

- Windows の場合

```
[VGC]<Wed Mar 17 00:42:30 2004>(Skip Full:0,Copy:0)[ Full GC 149K->149K(1984K),0.0786038
secs] [ DefNew::Eden:264K->0K(512K)] [ DefNew::Survivor:0K->63K(64K)] [ Tenured:
85K->149K(1408K)] [Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space: 356K(388K, 388K)-
>356K(388K, 388K)] [ cause:System.gc]
```

- UNIX の場合

```
[VGC]<Wed Mar 17 00:42:30 2004>(Skip Full:0,Copy:0)[ Full GC 149K->149K(1984K),0.0786038
secs] [ DefNew::Eden:264K->0K(512K)] [ DefNew::Survivor:0K->63K(64K)] [ Tenured:
85K->149K(1408K)] [Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space: 356K(388K, 388K)-
>356K(388K, 388K)] [ cause:JavaGC Command]
```

javagc コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「javagc（GC の強制発生)」を参照してください。

なお，-XX:+HitachiVerboseGCPrintCause オプションは，usrconf.cfg に指定します。オプションの設定方法については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「-XX:[+|-]HitachiVerboseGCPrintCause（GC 要因内容出力オプション)」を参照してください。

(5)　メモリダンプの取得（Web コンテナサーバを再起動する場合）

メモリダンプの取得の取得方法について OS ごとに説明します。

(a)　Windows の場合

Web コンテナサーバを再起動するときに，メモリダンプを取得します。メモリダンプは，システムでトラブルが発生したときに，保守員が障害解析するために使用します。

Web コンテナサーバが稼働している場合（cjstartweb プロセスが存在する場合），タスクマネージャから cjstartweb のプロセス ID を取得します。userdump.exe <プロセス ID> を実行し，メモリダンプ（user.dump）を採取します。

(b)　UNIX の場合

Web コンテナサーバを再起動するときに，メモリダンプとして，Web コンテナサーバの core ダンプを取得します。メモリダンプは，システムでトラブルが発生したときに，保守員が障害解析するために使用します。

cjstartweb プロセスがダウンした場合，「/opt/Cosminexus/CC/web/containers/<サーバ名称>/core*」に出力された core ダンプを取得します。

core ダンプのファイル名は，cjstartweb プロセスの再起動時に「core.<出力日時※>」（AIX，HP-UX の場合）または「core.<プロセス ID>.<出力日時※>」（Linux の場合）にリネームされます。cjstartweb プロセスの再起動時に core ダンプが上書き保存されないため，障害発生時の core ダンプを保存できます。

注※

出力日時は「YYMMDDhhmmss」の形式で出力されます。

YY：西暦（下 2 けた）　MM：月（2 けた）　DD：日（2 けた）

hh：時（2 けた[24 時間表記]）　mm：分（2 けた）　ss：秒（2 けた）

なお，保存する core ダンプは上限値を設定できます。cjstartweb プロセスの再起動時に「/opt/Cosminexus/CC/web/containers/<サーバ名称>/」に出力される core ダンプファイルの合計が上限値を超えた場合，出力日時が古い順に削除されます。上限値は，usrconf.cfg の web.server.corefilenum キーに指定します。なお，ファイルの削除は cjstartweb プロセスの再起動時に実行されます。

core ダンプを取得したあとに，core ダンプからスタックトレース情報だけを取得する場合は，javatrace コマンドを実行してください。スタックトレース情報は，JavaVM の異常終了の原因を究明するために必要な情報です。スタックトレース情報の取得方法は，J2EE サーバモードの場合と同様です。マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.18　JavaVM のスタックトレース情報」を参照してください。

core ダンプは次のような場合にも取得できます。それぞれの場合の取得方法について説明します。

- cjstartweb プロセスが稼働している状態で core ダンプを取得する場合
- 実行中の Java プロセスで core ダンプとスレッドダンプを同時に取得する場合

cjstartweb プロセスが稼働している状態で core ダンプを取得する場合

cjstartweb プロセス（Web コンテナサーバ）が稼働している状態で core ダンプを取得する場合，cjstartweb プロセスのプロセス ID を確認して，kill コマンドを実行します。kill コマンドは次の形式で実行してください。なお，kill コマンドを実行するとプロセスが終了するため，再起動する前に kill コマンドを実行することを推奨します。

```
kill -6 <cjstartwebプロセスID>
```

実行中の Java プロセスで core ダンプとスレッドダンプを同時に取得する場合

実行中の Java プロセスで core ダンプとスレッドダンプを同時に取得する方法は，J2EE サーバモードの場合と同様です。マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.9　メモリダンプ」を参照してください。

(6)　アプリケーションサーバの定義情報の取得（サーブレットエンジンモードの場合）

アプリケーションサーバの定義を取得します。この情報を使用して，発生した障害によって定義の設定を確認します。

Web コンテナサーバに関する定義情報

次のディレクトリ下に格納されているファイル一式を取得します。

Windows の場合

- <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf 下すべて

UNIX の場合

- /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf 下すべて

DABroker Library に関する定義情報

次のファイルを取得します。

Windows の場合

- <DABroker Library 運用ディレクトリ>¥conf¥dasys.conf（DABroker Library 動作環境定義ファイル）
- <DABroker Library 運用ディレクトリ>¥conf¥dabenv（接続先データベース定義ファイル）

UNIX の場合

- <DABroker Library 運用ディレクトリ>/conf/dasys.conf（DABroker Library 動作環境定義ファイル）
- <DABroker Library 運用ディレクトリ>/conf/dabenv（接続先データベース定義ファイル）

なお，トラブルシューティング用に保存するファイルからは，パスワードなどの公開できない情報を削除することをお勧めします。

(7)　Web コンテナサーバの作業ディレクトリの内容取得（サーブレットエンジンモードの場合）

システムでトラブルが発生した場合に，保守員による原因究明のため作業ディレクトリを調査に使用する場合があります。

- 環境変数「PRFSPOOL」の設定ディレクトリ下すべて※
- <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥webapps 下すべて（Windows の場合）
- /opt/Cosminexus/CC/web/containers/<サーバ名称>/webapps 下すべて（UNIX の場合）

注※

環境変数「PRFSPOOL」未設定の場合は，「<製品のインストールディレクトリ>¥PRF¥spool 下すべて」（Windows の場合），または「/opt/Cosminexus/PRF/spool 下すべて」（UNIX の場合）になります。

設定を確認の上，トラブルが発生した場合に保存しておいてください。

(8)　Web サーバログの取得

システムで使用している Web サーバ（HTTP Server または Microsoft IIS）のログを取得します。

HTTP Server を使用している場合，ログの格納場所は「<製品のインストールディレクトリ>¥httpsd¥logs」（Windows の場合），「/opt/hitachi/httpsd/logs」（UNIX の場合）です。

Microsoft IIS を使用している場合，ログの格納場所は「C:¥inetpub¥logs」です（C:の部分にはシステムドライブを指定します）。

(9)　そのほかの情報の取得（サーブレットエンジンモードの場合）

そのほかの情報を取得するための方法は，すべて J2EE サーバモードの場合と同様です。参照先を次に示します。

表 3-44　取得する情報の種類と参照先

取得する情報	参照先マニュアル	参照先
JavaVM ログ	アプリケーションサーバ 機能解説 保守／移行編	4.10
JavaVM 出力メッセージログ（標準出力またはエラーリポートファイル）		4.11
OS の状態情報と OS のログ		4.12
OS の統計情報		4.13

3.9　サーブレットエンジンモードの場合の移行

サーブレットエンジンモードの場合の，旧バージョンからアプリケーションサーバ 09-70 への移行方法について説明します。移行の対象となるバージョンを次に示します。

- アプリケーションサーバ 06-00
- アプリケーションサーバ 06-02（Windows だけのバージョン）
- アプリケーションサーバ 06-50
- アプリケーションサーバ 06-51
- アプリケーションサーバ 06-70
- アプリケーションサーバ 06-71
- アプリケーションサーバ 07-00
- アプリケーションサーバ 07-10
- アプリケーションサーバ 07-50
- アプリケーションサーバ 07-60
- アプリケーションサーバ 08-00
- アプリケーションサーバ 08-50
- アプリケーションサーバ 08-53
- アプリケーションサーバ 08-70
- アプリケーションサーバ 09-00
- アプリケーションサーバ 09-50
- アプリケーションサーバ 09-51
- アプリケーションサーバ 09-60

なお，このマニュアルでは，次に示す構成ソフトウェアの移行を中心に説明します。

- Component Container
- Performance Tracer

旧バージョンからの移行を行う場合，使用しているアプリケーションサーバの全構成ソフトウェアを同時に移行する必要があります。上記で示していないほかの構成ソフトウェア（以降，他構成ソフトウェアと呼びます）の移行については，各構成ソフトウェアのドキュメントを参照してください。

移行手順の中で，J2EE サーバモードの場合と同じ内容については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「10.　旧バージョンのアプリケーションサーバからの移行（J2EE サーバモードの場合）」で詳細を確認してください。なお，サーブレットエンジンモードの場合は，「J2EE サーバ」を「Web コンテナサーバ」と読み替えてください。

3.9.1　アプリケーションサーバの移行の概要

アプリケーションサーバの移行の概要について説明します。

(1)　移行のためのアップグレードインストール

旧バージョンからアプリケーションサーバ 09-70 へ移行するためには，アップグレードインストールを行います。アップグレードインストールとは，インストール先のマシンに旧バージョンのアプリケーションサーバがインストール済みの場合のインストールです。

なお，アップグレードインストールの場合は，<製品のインストールディレクトリ>を旧バージョンのインストールディレクトリから変更できません。旧バージョンがインストールされているディレクトリに上書きインストールされます。

(2)　移行時に引き継がれる情報

アプリケーションサーバを旧バージョンから 09-70 に移行する場合，旧バージョンで使用されていた，アプリケーションサーバのインストールディレクトリ以下に配置される定義情報，DD 情報，構成情報などのユーザ資産は，移行コマンドなどで必要に応じて自動変換され，09-70 で引き続き使用されます（一部，手動での定義修正が必要な場合もあります）。

なお，移行コマンドを実行すると，移行元バージョンから移行先バージョン間のすべてのバージョンごとに移行処理が実施されます。例えば，08-00 から 09-70 へ移行する場合，08-00 から 08-50 への移行，08-50 から 08-53 への移行，08-53 から 08-70 への移行，08-70 から 09-00 への移行，09-00 から 09-50 への移行，09-50 から 09-70 への移行という処理が実施されます。

Component Container で引き継がれる定義情報，DD 情報，構成情報などを次の表に示します。他構成ソフトウェアについては，それぞれのドキュメントを参照してください。

表 3-45　Component Container で移行時に引き継がれる情報（サーブレットエンジンモードの場合）

機能	引き継がれる情報	ファイル名称，または配置場所
Web コンテナサーバ	オプション定義ファイル	usrconf.cfg
	ユーザプロパティファイル	usrconf.properties
	Web アプリケーションプロパティファイル	hitachi_web.properties
	ユーザ認証情報定義ファイル	web-users.xml
	Security Manager 定義ファイル	web.policy
	作業ディレクトリ • 定義情報 • アプリケーション • DD 情報	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers 以下 • UNIX の場合 /opt/Cosminexus/CC/web/containers 以下
Web サーバ連携	HTTP Server 用リダイレクタ動作定義ファイル	mod_jk.conf
	Microsoft IIS 用リダイレクタ動作定義ファイル（Windows だけのファイル）	isapi_redirect.conf
	Microsoft IIS 用マッピング定義ファイル（Windows だけのファイル）	uriworkermap.properties
	ワーカ定義ファイル	workers.properties

機能	引き継がれる情報	ファイル名称，または配置場所
統合ユーザ管理	JAAS のコンフィグレーションファイル	jaas.conf
	統合ユーザ管理のコンフィグレーションファイル	ua.conf
	カスタムログインモジュール格納用ディレクトリ	• Windows の場合 <製品のインストールディレクトリ>¥manager¥modules • UNIX の場合 /opt/Cosminexus/manager/modules

3.9.2　アプリケーションサーバの移行の手順

次の手順に従って，旧バージョンからアプリケーションサーバ 09-70 へ移行してください。

1. 旧バージョンからの仕様変更の確認
2. 移行前の確認
3. 定義などのバックアップ
4. 他構成ソフトウェアの移行準備
5. 全構成ソフトウェアのアップグレードインストール
6. Web コンテナサーバの移行コマンドの実行
7. 手動による定義などの修正
8. 他構成ソフトウェアの移行実施
9. 移行後の確認

それぞれの手順について説明します。

(1)　旧バージョンからの仕様変更の確認

旧バージョンからの仕様変更を確認します。機能の変更や追加に伴い，旧バージョンで使用している機能，使用条件，使用方法によっては，次の作業が必要となります。

- 操作手順の変更
- 運用手順の変更
- アプリケーション，および DD の変更

なお，互換性を保つためのオプションが提供されている場合もありますが，推奨のオプションではありません。詳細については，「3.9.3　旧バージョンからの仕様変更の確認」を参照してください。

(2)　移行前の確認

移行前にアプリケーションサーバの動作環境，動作状態などを確認します。詳細については，「3.9.4　移行前の確認」を参照してください。

(3)　定義などのバックアップ

移行前に定義ファイルなどをバックアップします。詳細については，「3.9.5　定義などのバックアップ」を参照してください。

(4)　他構成ソフトウェアの移行準備

次に示す構成ソフトウェア以外の，他構成ソフトウェアの移行を準備します。

- Component Container
- Performance Tracer

移行の準備の詳細はそれぞれの他構成ソフトウェアのドキュメントを参照してください。なお，ここで準備した情報を基に，「(8)　他構成ソフトウェアの移行実施」で他構成ソフトウェアを移行してください。

(5)　全構成ソフトウェアのアップグレードインストール

使用しているアプリケーションサーバの全構成ソフトウェアをアップグレードインストールします。

(6)　Web コンテナサーバの移行コマンドの実行

定義情報，DD 情報などを継続して利用できるようにするため，Web コンテナサーバの移行コマンドを実行します。実行する移行コマンドは cjenvupdate コマンドで，J2EE サーバを移行する場合と同じです。

移行コマンドの実行については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「10.8 J2EE サーバまたはバッチサーバの移行コマンドの実行」を参照してください。このとき，J2EE サーバを Web コンテナサーバと読み替えてください。

(7)　手動による定義などの修正

手動で定義などを修正します。機能の変更，追加に伴い，互換性を保つためのオプションが提供されています。旧バージョンで使用している機能，使用方法，使用条件によっては，次の作業が必要となります。

- ユーザプロパティファイルなどの定義ファイルの変更
- Java のユーザクラスパスの変更
- 環境変数の変更

詳細については，「3.9.6　手動による定義などの修正」を参照してください。

(8)　他構成ソフトウェアの移行実施

他構成ソフトウェアを移行します。移行の詳細はそれぞれの他構成ソフトウェアのドキュメントを参照してください。なお，「(4)　他構成ソフトウェアの移行準備」で準備した情報を基に移行を実施してください。

(9)　移行後の確認

移行後にアプリケーションサーバの動作確認を実施します。

3.9.3　旧バージョンからの仕様変更の確認

旧バージョンからの仕様変更について説明します。

- アプリケーションサーバ Version 6 からの仕様変更
  アプリケーションサーバ Version 6 からの仕様変更を次の表に示します。

表 3-46　アプリケーションサーバ Version 6 からの仕様変更（サーブレットエンジンモードの場合）

分類	仕様変更の項目	06-71/ 06-70	06-51/ 06-50	06-02/ 06-00
全体	(1)　J2SE のバージョン	○	○	○
	(2)　TPBroker の運用支援機能の非サポート	○	○	○
Web コンテナサーバ	(3)　JavaVM のタイプ	−	−	○
	(4)　JSP から生成されたサーブレット用コンパイラのバージョン	○	○	○
	(5)　デフォルトのエラーページの内容	−	○	○
	(6)　カスタムタグのスクリプト変数定義の変更	○	○	○
	(7)　<jsp:useBean>タグの class 属性に対するエラーチェックの変更	○	○	○
	(8)　タグの属性値の Expression チェックの変更	○	○	○
	(9)　タグの属性値に指定する Expression に関する変更	○	○	○
	(10)　taglib ディレクティブの prefix 属性に関する変更	○	○	○
	(11)　JSP での$の扱いの変更	○	○	○
	(12)　Web コンテナがサポートする Servlet 仕様のバージョン	○	○	○
	(13)　サーブレットのデフォルトマッピングのデフォルトでの無効化	○	○	○
	(14)　javax.servlet.GenericServlet クラスでスローされる例外の変更	○	○	○
	(15)　javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッド呼び出し時の動作変更	○	○	○
	(16)　javax.servlet.GenericServlet クラスの init(ServletConfig)メソッドと destroy()メソッドでのログ出力	○	○	○
	(17)　-XX:SurvivorRatio のデフォルト値の変更	○	○	○
	(18)　JSP から生成されたサーブレット用コンパイラのバージョン	○	○	○
	(19)　POST リクエストのフォームデータの最大サイズの制限	○	○	○
	(20)　サーブレットのサービスメソッド実行中の HTTP セッションのタイムアウト	○	○	○

（凡例）○：仕様変更あり　－：該当しない

- アプリケーションサーバ Version 7 からの仕様変更

アプリケーションサーバ Version 7 からの仕様変更を次の表に示します。

表 3-47　アプリケーションサーバ Version 7 からの仕様変更（サーブレットエンジンモードの場合）

分類	仕様変更の項目	07-60	07-50	07-10	07-00
Web コンテナサーバ	(12)　Web コンテナがサポートする Servlet 仕様のバージョン	○	○	○	○
	(13)　サーブレットのデフォルトマッピングのデフォルトでの無効化	○	○	○	○
	(14)　javax.servlet.GenericServlet クラスでスローされる例外の変更	○	○	○	○
	(15)　javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッド呼び出し時の動作変更	○	○	○	○
	(16)　javax.servlet.GenericServlet クラスの init(ServletConfig)メソッドと destroy()メソッドでのログ出力	○	○	○	○
	(17)　-XX:SurvivorRatio のデフォルト値の変更	○	○	○	○
	(18)　JSP から生成されたサーブレット用コンパイラのバージョン	○	○	○	○
	(19)　POST リクエストのフォームデータの最大サイズの制限	○	○	○	○

（凡例）○：仕様変更あり

- アプリケーションサーバ Version 8 からの仕様変更

アプリケーションサーバ Version 8 からの仕様変更を次の表に示します。

表 3-48　アプリケーションサーバ Version 8 からの仕様変更（サーブレットエンジンモードの場合）

分類	仕様変更の項目	08-70	08-53	08-50	08-00
Web コンテナサーバ	(18)　JSP から生成されたサーブレット用コンパイラのバージョン	－	－	－	○
	(19)　POST リクエストのフォームデータの最大サイズの制限	－	－	－	○
	(20)　サーブレットのサービスメソッド実行中の HTTP セッションのタイムアウト	○	○	○	

（凡例）○：仕様変更あり　－：該当しない

- アプリケーションサーバ Version 9 からの仕様変更

アプリケーションサーバ Version 9 からの仕様変更を次の表に示します。

表 3-49　アプリケーションサーバ Version 9 からの仕様変更（サーブレットエンジンモードの場合）

分類	仕様変更の項目	09-00
Web コンテナサーバ	(21)　サーバ ID のデフォルト値の変更	○

（凡例）　○：仕様変更あり

次に仕様変更の項目の詳細について説明します。なお，表の「仕様変更の項目」の項番は，説明の項番と対応しています。

(1)　J2SE のバージョン

変更内容は，J2EE サーバモードの場合と同じです。

(2)　TPBroker の運用支援機能の非サポート

07-00 以降では，07-00 より前まで提供していた TPBroker の運用支援機能は使用できません。システムの各プロセスを一括起動，停止，および監視する場合には，Management Server を使用してください。

Web コンテナサーバを TPBroker の運用支援機能で使用していた場合，次のどちらかで運用してください。

- J2EE サーバモードで管理する。
- Web コンテナサーバを論理ユーザサーバとして定義して運用する。

(3)　JavaVM のタイプ

変更内容は，J2EE サーバモードの場合と同じです。

(4)　JSP から生成されたサーブレット用コンパイラのバージョン

変更内容は，J2EE サーバモードの場合と同じです。

(5)　デフォルトのエラーページの内容

変更内容は，J2EE サーバモードの場合と同じです。

(6)　カスタムタグのスクリプト変数定義の変更

変更内容は，J2EE サーバモードの場合と同じです。

(7)　<jsp:useBean>タグの class 属性に対するエラーチェックの変更

変更内容は，J2EE サーバモードの場合と同じです。

(8)　タグの属性値の Expression チェックの変更

変更内容は，J2EE サーバモードの場合と同じです。

(9)　タグの属性値に指定する Expression に関する変更

変更内容は，J2EE サーバモードの場合と同じです。

(10)　taglib ディレクティブの prefix 属性に関する変更

変更内容は，J2EE サーバモードの場合と同じです。

(11)　JSP での$の扱いの変更

変更内容は，J2EE サーバモードの場合と同じです。

(12)　Web コンテナがサポートする Servlet 仕様のバージョン

変更内容は，J2EE サーバモードの場合と同じです。

(13)　サーブレットのデフォルトマッピングのデフォルトでの無効化

変更内容は，J2EE サーバモードの場合と同じです。

(14)　javax.servlet.GenericServlet クラスでスローされる例外の変更

変更内容は，J2EE サーバモードの場合と同じです。

(15)　javax.servlet.ServletRequest インタフェースの setCharacterEncoding メソッド呼び出し時の動作変更

変更内容は，J2EE サーバモードの場合と同じです。

(16)　javax.servlet.GenericServlet クラスの init(ServletConfig)メソッドと destroy()メソッドでのログ出力

変更内容は，J2EE サーバモードの場合と同じです。

(17)　-XX:SurvivorRatio のデフォルト値の変更

変更内容は，J2EE サーバモードの場合と同じです。

(18)　JSP から生成されたサーブレット用コンパイラのバージョン

製品のバージョンによって，インストールされる JDK が異なります。Web コンテナサーバでは，インストールされている javac コンパイラのデフォルトバージョンの Java 言語仕様に従って JSP から生成されたサーブレットのソースファイルをコンパイルします。

コンパイルする Java 言語仕様のバージョンを次の表に示します。

表 3-50　コンパイルする Java 言語仕様のバージョン

アプリケーションサーバのバージョン	JDK のバージョン	コンパイル Java 言語仕様バージョン
Version 6	JDK 1.4	J2SE 1.3
Version 7	JDK 5.0	J2SE 5.0
Version 8	JDK 6	Java SE 6
	JDK 5.0	J2SE 5.0

! 注意事項

移行前と同様の Java 言語仕様に従って JSP から生成されたサーブレットのソースファイルをコンパイルする場合は，usrconf.properties（Web コンテナサーバ用ユーザプロパティファイル）の

webserver.jsp.compile.backcompat キーに「1.2」「1.3」「1.4」「1.5」または「5」を設定します。このプロパティは，インストールしている JDK のコンパイルでエラーが発生する場合だけ使用してください。

(19)　POST リクエストのフォームデータの最大サイズの制限

POST リクエストのフォームデータ（Content-Type ヘッダの値が application/x-www-form-urlencoded の場合の POST データ）の最大サイズがデフォルトで 2MB に制限されます。アップグレードインストールをした場合，すでにセットアップされている Web コンテナサーバには，POST リクエストのフォームデータの最大サイズを無制限にする設定（webserver.connector.limit.max_post_form_data=-1）が自動的に実施されます。

(20)　サーブレットのサービスメソッド実行中の HTTP セッションのタイムアウト

アプリケーションサーバでは，Web アプリケーションのバージョンに関係なく，Web アプリケーションでのリクエスト処理を実行している間は，HTTP セッションはタイムアウトされません。

ただし，07-00 より前の Web コンテナでは，Web アプリケーション単位の同時実行スレッド数制御機能による実行待ち中であっても HTTP セッションのタイムアウトが発生します。

(21)　サーバ ID のデフォルト値の変更

変更内容は，J2EE サーバモードの場合と同じです。

3.9.4　移行前の確認

移行前に，アプリケーションサーバの動作環境，および動作状態を確認します。次に確認項目を示します。

- 旧バージョンのアプリケーションサーバが終了していることを確認してください。アプリケーションサーバの全構成ソフトウェアを終了させます。
- Microsoft IIS を使用している場合は，Microsoft IIS を終了させます。

3.9.5　定義などのバックアップ

移行前に定義ファイルなどのバックアップを取ります。バックアップを取るディレクトリを次の表に示します。

表 3-51　バックアップを取るディレクトリ（サーブレットエンジンモードの場合）

機能	バックアップを取るディレクトリ
統合ユーザ管理	• Windows の場合 <製品のインストールディレクトリ>¥manager • UNIX の場合 /opt/Cosminexus/manager
Web サーバ連携	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥redirector • UNIX の場合 /opt/Cosminexus/CC/web/redirector
Web コンテナサーバ	• Windows の場合 <製品のインストールディレクトリ>¥CC¥web¥containers

機能	バックアップを取るディレクトリ
Web コンテナサーバ	• UNIX の場合 /opt/Cosminexus/CC/web/containers

3.9.6　手動による定義などの修正

旧バージョンから移行する場合に，手動で定義などの修正が必要な項目について説明します。

- **手動で定義などの修正が必要な項目（アプリケーションサーバ Version 6 からの移行）**

  アプリケーションサーバ Version 6 から移行する場合に，手動で定義などの修正が必要な項目を次の表に示します。

  表 3-52　手動で定義などの修正が必要な項目（サーブレットエンジンモードの場合のアプリケーションサーバ Version 6 から移行）

分類	修正項目	06-71/ 06-70	06-51/ 06-50	06-02/ 06-00
Web コンテナサーバ	(1)　セキュリティポリシーファイルの変更	○	○	○
	(2)　統合ユーザ管理のタグライブラリの入れ替え	○	○	○
	(3)　デフォルトエラーページのスタックトレース表示	－	○	○
	(4)　JSP のコンパイル結果の削除	○	○	○
	(5)　リダイレクタへのデータ送信処理のタイムアウト追加	○	○	○
	(6)　ユーザスレッドの生成に必要な権限の追加	－	○	○
	(7)　セッション ID へのサーバ ID の付加	－	○	○
	(8)　javax.servlet.ServletException で生成したオブジェクトに対する initCause メソッドの呼び出し	○	○	○
Web サーバ連携	(9)　リダイレクタのログ出力先変更	－	－	○
	(10)　Web コンテナへのデータ送信処理のタイムアウト追加	○	○	○
	(11)　リダイレクタの保守用トレースログの面数変更	○	○	－

（凡例）　○：修正が必要　－：該当しない

- **手動で定義などの修正が必要な項目（アプリケーションサーバ Version 7 からの移行）**

  アプリケーションサーバ Version 7 から移行する場合に，手動で定義などの修正が必要な項目を次の表に示します。

  表 3-53　手動で定義などの修正が必要な項目（サーブレットエンジンモードの場合のアプリケーションサーバ Version 7 から移行）

分類	修正項目	07-60	07-50	07-10	07-00
Web コンテナサーバ	(8)　javax.servlet.ServletException で生成したオブジェクトに対する initCause メソッドの呼び出し	○	○	○	○

（凡例）　○：修正が必要

次に修正項目の詳細について説明します。なお，表の「修正項目」の項番は，説明の項番と対応しています。

- **手動で定義などの修正が必要な項目（アプリケーションサーバ Version 8 からの移行）**
  アプリケーションサーバ Version 8 から移行する場合に，手動で定義などの修正が必要な項目はありません。
- **手動で定義などの修正が必要な項目（アプリケーションサーバ Version 9 からの移行）**
  アプリケーションサーバ Version 9 から移行する場合に，手動で定義などの修正が必要な項目はありません。

(1)　セキュリティポリシーファイルの変更

変更方法は，J2EE サーバモードの場合と同じです。

- Windows の場合
  <製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥web.policy
- UNIX の場合
  opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/web.policy

(2)　統合ユーザ管理のタグライブラリの入れ替え

変更方法は，J2EE サーバモードの場合と同じです。

(3)　デフォルトエラーページのスタックトレース表示

変更方法は，J2EE サーバモードの場合と同じです。

(4)　JSP のコンパイル結果の削除

07-00 以降では，サーブレット／JSP のバージョンが変更になりました。このため，07-00 より前の Web コンテナサーバの usrconf.properties ファイルで，webserver.work.clean=false を設定して JSP コンパイル結果を保持していた場合，07-00 以降へ移行後，07-00 より前で生成されたコンパイル結果を手動で削除してください。

次のどちらかの方法で，JSP コンパイル結果を削除します。

- usrconf.properties で webserver.work.clean=true を設定して，Web コンテナサーバを再起動する。
- JSP コンパイル結果を手動で削除する。

JSP コンパイル結果を手動で削除する方法については，「3.4.3　JSP ファイルのコンパイル結果の保持」を参照してください。

(5)　リダイレクタへのデータ送信処理のタイムアウト追加

変更方法は，J2EE サーバモードの場合と同じです。

(6)　ユーザスレッドの生成に必要な権限の追加

変更方法は，J2EE サーバモードの場合と同じです。なお，サーブレットエンジンモードの場合は，「server.policy ファイル」を「web.policy ファイル」と読み替えてください。

- Windows の場合

<製品のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥web.policy

- UNIX の場合
  opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/web.policy

(7)　セッション ID へのサーバ ID の付加

変更内容は，J2EE サーバモードの場合と同じです。

(8)　javax.servlet.ServletException で生成したオブジェクトに対する initCause メソッドの呼び出し

コンストラクタ ServletException(String,Throwable)および ServletException(Throwable)で生成した ServletException オブジェクトに対して，initCause(Throwable)を呼び出せません。
initCause(Throwable)を呼び出したい場合は，ユーザプロパティファイルに webserver.servlet_api.exception.getCause.backcompat=true を設定する必要があります。

(9)　リダイレクタのログ出力先変更

変更方法は，J2EE サーバモードの場合と同じです。

(10)　Web コンテナへのデータ送信処理のタイムアウト追加

変更方法は，J2EE サーバモードの場合と同じです。

(11)　リダイレクタの保守用トレースログの面数変更

変更方法は，J2EE サーバモードの場合と同じです。

3.9.7　Web コンテナサーバの移行コマンドで自動的に追加される定義

旧バージョンからアプリケーションサーバ 09-70 へ移行する場合，Web コンテナサーバのユーザプロパティファイルや作業ディレクトリ内の DD 情報は，旧バージョンのものが 09-70 の環境でもそのまま継続使用されます。ただし，新バージョンで追加された機能，およびデフォルト値が変更された機能があり，互換性を維持するために，Web コンテナサーバの移行コマンドがユーザプロパティファイルのプロパティや作業ディレクトリ内の DD 情報を部分的に変換します。移行コマンド実行時に追加されるキーと値について説明します。

● 移行コマンドで自動的に追加される定義（アプリケーションサーバ Version 6 からの移行）

移行コマンド実行時に追加されるキーと値を次の表に示します。

表 3-54　Web コンテナサーバの usrconf.properties に自動的に追加される定義（アプリケーションサーバ Version 6 からの移行）

追加されるキーと値	旧バージョンでの環境条件	06-71/06-70	06-51/06-50	06-02/06-00
webserver.container.servlet.default_mapping.enabled=true	−	○	○	○
ejbserver.logger.channels.define.WebServletLogFile.filenum=2	このキーがない場合	○	○	○

追加されるキーと値	旧バージョンでの環境条件	06-71/06-70	06-51/06-50	06-02/06-00
ejbserver.logger.channels.define.WebServletLogFile.filesize=1048576	このキーがない場合	○	○	○
ejbserver.logger.channels.define.WebExceptionLogFile.filenum=2	このキーがない場合	○	○	○
ejbserver.logger.channels.define.WebExceptionLogFile.filesize=1048576	このキーがない場合	○	○	○

（凡例）　○：定義が自動的に追加されることを示す　－：該当しない

表 3-55　Web コンテナサーバの usrconf.cfg に自動的に追加される定義（アプリケーションサーバ Version 6 からの移行）

追加されるキーと値	旧バージョンでの環境条件	06-71/06-70	06-51/06-50	06-02/06-00
add.jvm.arg=-XX:HitachiJavaLogFileSize=256k	このキーがない場合	○	○	○

（凡例）　○：定義が自動的に追加されることを示す　－：該当しない

表 3-56　Web コンテナサーバの server.policy に自動的に追加される定義（アプリケーションサーバ Version 7 からの移行）

追加される定義	旧バージョンでの環境条件	06-71/06-70	06-51/06-50	06-02/06-00

grant codeBase "file:${cosminexus.home}/common/lib/*" {
permission java.security.AllPermission;
};
	このキーがない場合	○	○	○

（凡例）　○：定義が自動的に追加されることを示す

● 移行コマンドで自動的に追加される定義（アプリケーションサーバ Version 7 からの移行）

移行コマンド実行時に追加されるキーと値を次の表に示します。

表 3-57　Web コンテナサーバの usrconf.properties に自動的に追加される定義（アプリケーションサーバ Version 7 からの移行）

追加されるキーと値	旧バージョンでの環境条件	07-60	07-50	07-10	07-00
webserver.container.servlet.default_mapping.enabled=true	－	○	○	○	○

（凡例）　○：定義が自動的に追加されることを示す　－：該当しない

表 3-58　Web コンテナサーバの server.policy に自動的に追加される定義（アプリケーションサーバ Version 7 からの移行）

追加される定義	旧バージョンでの環境条件	07-60	07-50	07-10	07-00

grant codeBase "file:${cosminexus.home}/common/lib/*" {
permission java.security.AllPermission;
};
	このキーがない場合	○	○	○	○

（凡例）　○：定義が自動的に追加されることを示す

● 移行コマンドで自動的に追加される定義（アプリケーションサーバ Version 8 からの移行）

表 3-59　Web コンテナサーバの usrconf.properties に自動的に追加される定義（アプリケーションサーバ Version 8 からの移行）

追加されるキーと値	旧バージョンでの環境条件	08-70	08-53	08-50	08-00
webserver.connector.limit.max_post_form_data=-1	このキーがない場合	－	－	－	○

（凡例）　○：定義が自動的に追加されることを示す　－：該当しない

表 3-60　Web コンテナサーバの server.policy に自動的に追加される定義（アプリケーションサーバ Version 8 からの移行）

追加される定義	旧バージョンでの環境条件	08-70	08-53	08-50	08-00

grant codeBase "file:${cosminexus.home}/common/lib/*" {
permission java.security.AllPermission;
};
	このキーがない場合	○	○	○	○

（凡例）　○：定義が自動的に追加されることを示す

● 移行コマンドで自動的に追加される定義（アプリケーションサーバ Version 9 からの移行）

アプリケーションサーバ Version 9 からの移行で自動的に追加される定義はありません。

3.10　Web コンテナサーバで使用するコマンド

サーブレットエンジンモードでは，次の種類のコマンドを使用できます。

- 性能解析トレースで使用するコマンド
- JavaVM で使用するコマンド
- ログの運用で使用するコマンド
- 統合ユーザ管理で使用するコマンド
- Web アプリケーションで使用するコマンド
- Web コンテナサーバで使用するコマンド

これらのコマンドのうち，「Web コンテナサーバで使用するコマンド」以外のコマンドについては，J2EE サーバモードで使用するコマンドと同じです。マニュアル「アプリケーションサーバ リファレンス コマンド編」，およびマニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を参照してください。

Web コンテナサーバで使用するコマンドの一覧を，次の表に示します。

表 3-61　Web コンテナサーバで使用するコマンドの一覧

コマンド名称	分類	概要
cjdumpweb	Web コンテナサーバのスレッドダンプの取得	Web コンテナサーバで障害が発生した場合などに，JavaVM のスレッドダンプを取得します。
cjenvsetup	Component Container 管理者のセットアップ	インストールファイルの属性を変更して，root 権限を持つユーザ以外で運用できるようにします。 このコマンドは，UNIX 用です。 このコマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjenvsetup（Component Container 管理者のセットアップ）」を参照してください。
cjgetsysinfo	OS 状態情報の取得	OS の状態情報を取得します。 このコマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetsysinfo（OS 状態情報の取得）」を参照してください。
cjstartweb	Web コンテナサーバの開始	Web コンテナサーバを開始します。
cjstopweb	Web コンテナサーバの停止	Web コンテナサーバを停止します。
cjwebchangedbid	データベース ID の変更	プール管理情報のデータベース ID を変更します。
cjwebclearpool	プール内のコネクションのクリア	データベースのコネクションをクローズして，クローズしたコネクションをプールから削除します。
cjwebclosepool	プール内のコネクションの強制切断	データベースのコネクションをプールから削除して，クローズします。
cjwebdeletepool	プール管理情報の削除	データベースのプール管理情報を削除します。
cjwebeditpool	プール管理情報の登録と変更	プール管理情報を新規登録または変更します。

コマンド名称	分類	概要
cjweblistpool	プール管理情報の一覧表示	プール管理情報の一覧を表示します。
cjwebsetup	Web コンテナサーバのセットアップとアンセットアップ	Web コンテナの環境をセットアップまたはアンセットアップします。

セットアップ系コマンド（cjwebsetup）は単独で実行してください。ほかのセットアップ系コマンド（cjenvsetup，cjenvupdate，cjrarupdate，cjsetup など）と同時に実行しないでください。

Web コンテナサーバで使用する各コマンドの入力形式，機能などを次に示します。

コマンドの格納先

Web コンテナサーバで使用するコマンドは，次のディレクトリに格納されています。

cjdumpweb コマンド，cjstartweb コマンド，cjstopweb コマンド，cjwebsetup コマンド

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥bin¥
- UNIX の場合
  /opt/Cosminexus/CC/web/bin/

それ以外のコマンド

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥bin¥
- UNIX の場合
  /opt/Cosminexus/CC/admin/bin/

3.10.1　cjdumpweb（Web コンテナサーバのスレッドダンプの取得）

(1)　形式

```
cjdumpweb [<サーバ名称>]
```

(2)　機能

Web ブラウザおよび EJB クライアントに応答が返らない場合，Web コンテナサーバで障害が発生した場合，アプリケーションの挙動を調査したい場合などに，JavaVM のスレッドダンプを取得します。

コマンドを実行すると，サーバ標準出力ログ（cjstdout.log），および次のファイルにスレッドダンプが出力されます。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥javacore*.txt
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/javacore*.txt

(3)　引数

<サーバ名称>

Web コンテナのサーバ名称を指定します。サーバ名称には，半角英数字（0～9，A～Z，a～z），アンダースコア（_）またはハイフン（-）が使用できます。サーバ名称の先頭の文字は英数字でなければなりません。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

(5)　注意事項

- UNIX の場合，cjdumpweb コマンドの実行には，root 権限または Component Container 管理者の権限が必要です。
- コマンドの引数にサーバ名称を指定する場合，cjwebsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。
- Web コンテナサーバを Manager から起動している場合，または Windows のサービスとして起動している場合，スレッドダンプの取得に失敗するおそれがあります。

3.10.2　cjstartweb（Web コンテナサーバの開始）

(1)　形式

```
cjstartweb [<サーバ名称>] [-security]
```

(2)　機能

サーバ名称の Web コンテナサーバを開始します。

(3)　引数

<サーバ名称>

開始する Web コンテナのサーバ名称を指定します。サーバ名称には，半角英数字（0～9，A～Z，a～z），アンダースコア（_）またはハイフン（-）が使用できます。サーバ名称の先頭の文字は英数字でなければなりません。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

-security

Web コンテナの実行時保護機能を有効にします。

(4)　戻り値

0：

正常終了しました。

1 以上：

異常終了しました。

(5)　ユーザ定義ファイル

- Windows の場合
  - <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.cfg
  - <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.properties
- UNIX の場合
  - /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.cfg
  - /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.properties

(6)　注意事項

- Windows の場合，cjstartweb コマンドの実行には，Administrator 権限が必要です。
- UNIX の場合，cjstartweb コマンドの実行には，root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。
- コマンドの引数にサーバ名称を指定する場合，cjwebsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。

3.10.3　cjstopweb（Web コンテナサーバの停止）

(1)　形式

```
cjstopweb [<サーバ名称>]
          [-f|-fd [<スレッドダンプ取得間隔> <スレッドダンプ取得回数>]]
```

(2)　機能

サーバ名称の Web コンテナサーバを停止します。

(3)　引数

<サーバ名称>

停止する Web コンテナサーバのサーバ名称を指定します。サーバ名称には，半角英数字（0～9，A～Z，a～z），アンダースコア（_）またはハイフン（-）が使用できます。サーバ名称の先頭の文字は英数字でなければなりません。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

-f

Web コンテナサーバを強制停止します。

-fd

Web コンテナサーバの障害情報を取得したあと，Web コンテナサーバを強制停止します。

<スレッドダンプ取得間隔>

スレッドダンプを取得する間隔を 1～60 の整数（単位：秒）で指定します。省略した場合の初期値は 3 秒です。

<スレッドダンプ取得回数>

スレッドダンプを取得する回数を 1～60 の整数（単位：回）で指定します。省略した場合の初期値は 10 回です。

取得する障害情報を次に示します。

1. OS の稼働状況

   Windows の場合の出力先：

   <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥cjsysinfo.<出力時間>.txt

   UNIX の場合の出力先：

   /opt/Cosminexus/CC/web/containers/<サーバ名称>/cjsysinfo.<出力時間>.txt

   <出力時間>は，yymmddHHMMSS の形式になります。

   なお，出力内容については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetsysinfo（OS 状態情報の取得）」を参照してください。

2. JavaVM のスレッドダンプ

   <スレッドダンプ取得間隔>，<スレッドダンプ取得回数>で指定した間隔と回数でスレッドダンプを取得します。Web コンテナサーバプロセスの状態によっては指定間隔，指定回数でスレッドダンプが出力されない場合もあります。なお，出力内容については，「cjdumpweb（Web コンテナサーバのスレッドダンプの取得）」を参照してください。

3. Web コンテナサーバのクラッシュダンプ（Windows の場合）または core ファイル（UNIX の場合）

   最後のスレッドダンプ取得要求後，スレッドダンプ取得間隔時間経過後にクラッシュダンプまたは core ファイルを出力します。

   クラッシュダンプの出力先：

   Windows XP と Windows Server 2012，Windows Server 2008，Windows 8，Windows 7 または Windows Vista ではクラッシュダンプの出力先は異なります。

   ・Windows XP の場合

   <ワトソン博士のクラッシュダンプ出力先>※¥user.dmp

   注※：ワトソン博士のクラッシュダンプ作成オプションの設定が必要です。

   ・Windows Server 2012，Windows Server 2008，Windows 8，Windows 7 または Windows Vista の場合

   <環境変数 CJMEMDUMP_PATH で指定したディレクトリ>※¥cjmemdump.dmp

   注※：環境変数 CJMEMDUMP_PATH が指定されていないか，指定したディレクトリが存在しない場合，クラッシュダンプは出力されません。また，指定したディレクトリにクラッシュダンプがすでに存在する場合，クラッシュダンプは上書きされます。

   core ファイルの出力先：

   /opt/Cosminexus/CC/web/containers/<サーバ名称>/core※

   注※

   Linux の場合，core.<強制停止された Web コンテナサーバのプロセス ID>

(4)　戻り値

0：

正常終了しました。

1 以上：
　異常終了しました。

(5)　ユーザ定義ファイル

- Windows の場合
  - <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.cfg
  - <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥usrconf.properties
- UNIX の場合
  - /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.cfg
  - /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/usrconf.properties

(6)　注意事項

- Web コンテナサーバを停止させるには，必ず cjstopweb コマンドを使用してください。Ctrl+C キーやタスクマネージャ（Windows の場合）から停止しないでください。
- cjstopweb コマンドで Web コンテナサーバを停止する場合，サーブレットまたは JSP のサービスメソッドが実行中のときは，該当するメソッドが終了するまで Web コンテナサーバは停止しません。なお，cjstopweb コマンドは，Web コンテナサーバの終了を待たないで，すぐに終了します。
- サーブレットまたは JSP のサービスメソッドが実行中の状態で cjstopweb コマンドを実行した場合，Web コンテナサーバを停止できないことがあります。この場合，-f オプションまたは-fd オプションを指定して，再度 cjstopweb コマンドを実行することで Web コンテナサーバを強制停止します。
- Windows の場合，cjstopweb コマンドの実行には，Administrator 権限が必要です。
- UNIX の場合，cjstopweb コマンドの実行には，root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。
- コマンドの引数にサーバ名称を指定する場合，cjwebsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。
- Windows Server 2012, Windows Server 2008, Windows 8, Windows 7 または Windows Vista で-fd オプションを指定する場合，環境変数 CJMEMDUMP_PATH は次のように設定してください。
  - 環境変数 CJMEMDUMP_PATH で指定した保存先のディスクに十分な空き容量があることを確認してください。クラッシュダンプのファイルサイズは WEB コンテナサーバの実メモリ所要量以上になります。
  - 環境変数 CJMEMDUMP_PATH に日本語などのマルチバイト文字を含むディレクトリを指定しないでください。クラッシュダンプの出力に失敗する場合があります。
  - クラッシュダンプの出力先ディレクトリには，存在するディレクトリを指定してください。

3.10.4　cjwebchangedbid（データベース ID の変更）

(1)　形式

```
cjwebchangedbid [-host <ホスト名称>] [-port <ポート番号>]
                -olddbid <データベースID> -newdbid <データベースID>
```

(2)　機能

-olddbid に指定されたデータベース ID に該当するプール管理情報のデータベース ID を，-newdbid に指定されたデータベース ID に変更します。-olddbid と-newdbid に同じデータベース ID を指定された場合，処理は実行しません。

使用中のプールのデータベース ID を変更した場合は，Web コンテナを再起動した時に変更内容が有効となります。

(3)　引数

-host <ホスト名称>

接続先のホスト名称として，コマンドからのリクエスト要求を受け付けるサーブレットが動作しているホスト名称を指定します。指定を省略した場合は，localhost が仮定されます。

-port <ポート番号>

コマンドからのリクエスト要求を受け付けるサーブレットが使用するポート番号として，サーブレットが管理されている Web コンテナのポート番号を指定します。指定を省略した場合は，8080 が仮定されます。

-olddbid <データベース ID>

変更の対象となる現在のデータベース ID を指定します。データベース ID には，半角英数字（0～9，A～Z，a～z）またはアンダースコア（_）が使用できます。

-newdbid <データベース ID>

変更後のデータベース ID を指定します。データベース ID には，半角英数字（0～9，A～Z，a～z）またはアンダースコア（_）が使用できます。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

(5)　入力例

```
cjwebchangedbid -olddbid db_1 -newdbid db_2
```

(6)　注意事項

- 引数は，コマンド名の直後に指定する必要があります。引数は，コマンド名より後ろにあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<データベースID> -olddbid）はできません。

3.10.5　cjwebclearpool（プール内のコネクションのクリア）

(1)　形式

```
cjwebclearpool [-host <ホスト名称>] [-port <ポート番号>]
               [-mode {normal|plan}] [-dbid <データベースID>]
```

(2)　機能

動作種別を選択して，プール内のすべてのコネクションまたは指定されたデータベースのコネクションをクローズします。クローズしたコネクションは，プールから削除します。

このコマンドは，データベースサーバが異常終了した場合やネットワークが切断された場合などの，システム障害時に実行するコマンドです。このコマンドでは，プールからコネクションを削除してクローズすることだけを実行します。新たなコネクションは生成しません。

(3)　引数

-host <ホスト名称>

接続先のホスト名称として，コマンドからのリクエスト要求を受け付けるサーブレットが動作しているホスト名称を指定します。指定を省略した場合は，localhost が仮定されます。

-port <ポート番号>

コマンドからのリクエスト要求を受け付けるサーブレットが使用するポート番号として，サーブレットが管理されている Web コンテナのポート番号を指定します。指定を省略した場合は，8080 が仮定されます。

-mode {normal|plan}

使用中のコネクションに対するクリア動作の種別を指定します。

- normal（通常モード）
  ユーザが使用中のコネクションはプールの外へ移して，プール内からはクリアされます。データベースとの接続はそのまま維持されますが，再利用はできません。
- plan（計画モード）
  ユーザが使用中のコネクションはクリアしません。ただし，再利用はできません。

-dbid <データベース ID>

削除対象となるプール管理情報のデータベース ID を指定します。データベース ID には，半角英数字（0～9，A～Z，a～z）またはアンダースコア（_）が使用できます。

省略すると，指定した動作種別で，すべてのプール管理情報を対象に処理が実行されます。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

(5)　入力例

```
cjwebclearpool -mode normal -dbid db_1
```

(6)　注意事項

- 引数は，コマンド名の直後に指定する必要があります。引数は，コマンド名より後ろにあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<データベース ID> -dbid）やオプション名と値を非対応にすること（例：-mode <データベース ID> -dbid normal）はできません。

3.10.6　cjwebclosepool（プール内のコネクションの強制切断）

(1)　形式

コネクション ID を指定する場合

```
cjwebclosepool [-host <ホスト名称>] [-port <ポート番号>]
               -dbid <データベースID> -type connectionid
               -cid <コネクションID>
```

タイムアウト時間を指定する場合

```
cjwebclosepool [-host <ホスト名称>] [-port <ポート番号>]
               -dbid <データベースID> -type timeout
               -time <タイムアウト時間>
```

(2)　機能

指定されたコネクション ID に該当するコネクション，または指定されたタイムアウト時間を超過したコネクションをプールから削除して，クローズします。

指定されたタイムアウト時間を超過したコネクションをプールから削除する場合，プールまたはデータベースに対して getConnection メソッド，SQL ステートメントなどが実行されてからの経過時間が，指定された時間を超えているコネクションが対象となります。

このコマンドは，コネクションが使用中のまま長時間放置されている場合などに使用します。コネクション ID または経過時間をキーにして削除して，プール内のコネクションが［Minimum size］に達していない場合は，新たにプールにコネクションを生成します。

(3)　引数

-host <ホスト名称>

接続先のホスト名称として，コマンドからのリクエスト要求を受け付けるサーブレットが動作しているホスト名称を指定します。指定を省略した場合は，localhost が仮定されます。

-port <ポート番号>

コマンドからのリクエスト要求を受け付けるサーブレットが使用するポート番号として，サーブレットが管理されている Web コンテナのポート番号を指定します。指定を省略した場合は，8080 が仮定されます。

-dbid <データベース ID>

削除対象となるプール管理情報のデータベース ID を指定します。データベース ID には，半角英数字（0～9，A～Z，a～z）またはアンダースコア（_）が使用できます。

-type {connectionid|timeout}

コネクションを削除する基準の種別を指定します。

- connectionid
  コネクション ID を基に，プールからコネクションを削除します。
- timeout
  タイムアウト時間を基に，プールからコネクションを削除します。

-cid <コネクション ID>

削除対象となるコネクションのコネクション ID を指定します。

-time <タイムアウト時間>

コネクションを削除する際のタイムアウト時間を指定します。コネクション最終アクセス時刻から，指定時間を超過したコネクションが削除対象となります。指定は秒単位で，指定できる値は 0≦指定値≦2147483647 の範囲になります。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

(5)　入力例

コネクション ID を指定する場合

```
cjwebclosepool -dbid db_1 -type connectionid -cid connection1
```

タイムアウト時間を指定する場合

```
cjwebclosepool -dbid db_1 -type timeout -time 36000
```

(6)　注意事項

- 引数は，コマンド名の直後に指定する必要があります。引数は，コマンド名より後ろにあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<データベース ID> -dbid）やオプション名と値を非対応にすること（例：-dbid timeout -type <データベース ID>）はできません。

3.10.7　cjwebdeletepool（プール管理情報の削除）

(1)　形式

```
cjwebdeletepool [-host <ホスト名称>] [-port <ポート番号>]
                -dbid <データベースID>
```

(2)　機能

指定されたデータベース ID に該当するプール管理情報を削除します。

使用中のプール管理情報が削除された場合，動作中のプールには影響を与えません。Web コンテナを再起動した時に有効になります。

(3)　引数

-host <ホスト名称>

接続先のホスト名称として，コマンドからのリクエスト要求を受け付けるサーブレットが動作しているホスト名称を指定します。指定を省略した場合は，localhost が仮定されます。

-port <ポート番号>

コマンドからのリクエスト要求を受け付けるサーブレットが使用するポート番号として，サーブレットが管理されている Web コンテナのポート番号を指定します。指定を省略した場合は，8080 が仮定されます。

-dbid <データベース ID>

削除対象となるプール管理情報のデータベース ID を指定します。データベース ID には，半角英数字（0～9，A～Z，a～z）またはアンダースコア（_）が使用できます。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

(5)　注意事項

- 引数は，コマンド名の直後に指定する必要があります。引数は，コマンド名より後ろにあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<データベース ID> -dbid）はできません。

3.10.8　cjwebeditpool（プール管理情報の登録と変更）

(1)　形式

```
cjwebeditpool [-host <ホスト名称>] [-port <ポート番号>]
              -c <プール管理情報設定ファイルパス>
```

(2)　機能

プール管理情報を，新規登録または変更します。

登録されていないデータベース ID をプール管理情報設定ファイルに定義した場合，プール管理情報を新規登録します。すでに登録されているデータベース ID をプール管理情報設定ファイルに定義した場合，プール管理情報を変更します。

プール管理情報設定ファイルに，データベース ID，クラス名および URL 名が指定されていない場合は，プール管理情報を登録できません。

プール管理情報設定ファイルは，あらかじめエディタなどを使って作成しておく必要があります。プール管理情報設定ファイルについては，「3.11.6　プール管理情報設定ファイル」を参照してください。

プール管理情報の変更は，データベース ID をキーにして実行されます。登録されていないデータベース ID を指定した場合，プール管理情報が新規に作成，登録されます。

(3)　引数

-host <ホスト名称>

接続先のホスト名称として，コマンドからのリクエスト要求を受け付けるサーブレットが動作しているホスト名称を指定します。指定を省略した場合は，localhost が仮定されます。

-port <ポート番号>

コマンドからのリクエスト要求を受け付けるサーブレットが使用するポート番号として，サーブレットが管理されている Web コンテナのポート番号を指定します。指定を省略した場合は，8080 が仮定されます。

-c <プール管理情報設定ファイルパス>

読み込み対象となるプール管理情報ファイル設定ファイルのパスを指定します。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

(5)　注意事項

- 引数は，コマンド名の直後に指定する必要があります。引数は，コマンド名より後ろにあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<プール管理情報設定ファイルパス> -c）はできません。
- プール管理情報を新規に登録する場合は，プール管理情報設定ファイルに<DatabaseID>，<ClassName>，<URL>を必ず指定してください。
- プール管理情報を変更する場合は，プール管理情報設定ファイルに<DatabaseID>を必ず指定してください。
- プール管理情報設定ファイルの<ClassName>と<URL>は対で指定してください。
- プール管理情報設定ファイルのそれぞれのタグには，次の条件を満たす値を指定してください。
  - <UserCount> ≧ 0
  - <MinimumSize> ≦ <MaximumSize>かつ<MaximumSize> ≧ 1
  - <MinimumSize> ≧ 0
  - <MaximumSize> ≧ 0
  - <RetryInterval> ≧ 0
  - <RetryCount> ≧ 0
- 新規登録または変更されたプール管理情報に対して，まだ一度もコネクション取得要求がない場合，コネクション取得要求することで新規登録または変更したプール管理情報が反映されたプールが生成されます。
- すでに使用中のプール管理情報を変更した場合，コマンド実行時に<DatabaseID>，<ClassName>，<URL>以外の項目が変更されます。<DatabaseID>，<ClassName>，<URL>については，Webコンテナを再起動してプールを再作成した時に有効になります。
- 新規登録時に<DatabaseID>，<ClassName>，<URL>以外の項目を省略した場合は，デフォルトの値が設定されます。
- プール管理情報設定ファイルについては，「3.11.6　プール管理情報設定ファイル」を参照してください。

3.10.9　cjweblistpool（プール管理情報の一覧表示）

(1)　形式

データベースリストを表示する場合

```
cjweblistpool [-host <ホスト名称>] [-port <ポート番号>]
              -listtype dblist [-f <ファイルパス>]
```

プールリストを表示する場合

```
cjweblistpool [-host <ホスト名称>] [-port <ポート番号>]
              -dbid <データベースID> -listtype poollist
              [-f <ファイルパス>]
```

コネクションリストを表示する場合

```
cjweblistpool [-host <ホスト名称>] [-port <ポート番号>]
              -dbid <データベースID> -userid <ユーザID>
              -listtype connectionlist [-f <ファイルパス>]
```

すべてのプール情報を表示する場合

```
cjweblistpool [-host <ホスト名称>] [-port <ポート番号>]
              -listtype all [-f <ファイルパス>]
```

(2)　機能

プール管理情報の一覧を表示します。表示できるプール管理情報を次に示します。

データベースリスト

データベースリスト（データベース ID，データベースクラス，データベース URL，ユーザ数，コネクション最小数，コネクション最大数，リトライ待ち時間，リトライ回数，およびプールの数）を表示します。または，指定されたファイルに出力します。

プールリスト

プールリスト（ユーザ ID，コネクション取得待ちカウンタ，およびコネクションの数）と，キー項目のデータベース ID を表示します。または，指定されたファイルに出力します。

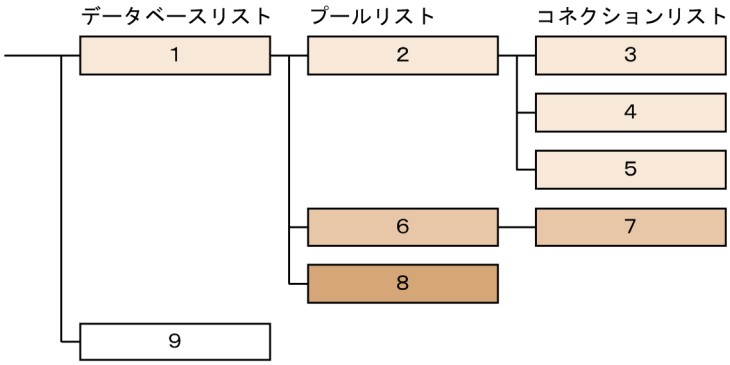
コネクションリスト

コネクションリスト（コネクション ID，使用中フラグ，再利用可否フラグ，コネクション生成時刻，コネクション最終アクセス時刻）と，キー項目のデータベース ID，ユーザ ID およびコネクション ID を表示します。または，指定されたファイルに出力します。

すべてのプール管理情報

上記のすべてのプール管理情報を一覧表示します。または，指定されたファイルに出力します。

表示順序は次の図中の番号に示した順番になります。一つのデータベースリストに関連するプールリストが表示され終わってから，次のデータベースリストが表示されます。また，一つのプールリストに関連するプールリストに関連するコネクションリストが表示され終わってから，次のプールリストが表示されます。



なお，データベースリスト，プールリストおよびコネクションリストに出力される項目については，-listtype オプションのオプション引数に，dblist，poollist，connectionlist を指定した場合と同じです。

(3)　引数

-host <ホスト名称>

接続先のホスト名称として，コマンドからのリクエスト要求を受け付けるサーブレットが動作しているホスト名称を指定します。指定を省略した場合は，localhost が仮定されます。

-port <ポート番号>

コマンドからのリクエスト要求を受け付けるサーブレットが使用するポート番号として，サーブレットが管理されている Web コンテナのポート番号を指定します。指定を省略した場合は，8080 が仮定されます。

-listtype {dblist|poollist|connectionlist|all}

表示対象となるプール管理情報の種別を指定します。

- dblist
  データベースリストを一覧表示します。
- poollist
  プールリストを一覧表示します。
- connectionlist
  コネクションリストを一覧表示します。
- all
  すべてのプール管理情報を一覧表示します。

-f <ファイルパス>

プール管理情報の出力先ファイルのパスを指定します。

-dbid <データベース ID>

処理対象となるプール管理情報のデータベース ID を指定します。データベース ID には，半角英数字（0～9，A～Z，a～z）またはアンダースコア（_）が使用できます。

-userid <ユーザ ID>

処理対象となるプール管理情報のユーザ ID を指定します。

(4)　入力例

データベースリストを表示する場合

```
cjweblistpool -listtype dblist -f db1.list
```

プールリストを表示する場合

```
cjweblistpool -dbid db_1 -listtype poollist -f pool1.list
```

コネクションリストを表示する場合

```
cjweblistpool -dbid db_1 -userid user1 -listtype connectionlist -f connection1.list
```

すべてのプール情報を表示する場合

```
cjweblistpool -listtype all -f all.list
```

(5)　出力形式

データベースリストを表示する場合

```
<DatabaseID=<データベースID1>><ClassName=<データベースクラス>><URL=<データベース
URL>><UserCount=<ユーザ数>><MinimumSize=<コネクション最小数>><MaximumSize=<コネクション最大
数>><RetryInterval=<リトライ待ち時間>><RetryCount=<リトライ回数>><PoolCount=<プールの数>>
<DatabaseID=<データベースID2>><ClassName=<データベースクラス>><URL=<データベース
URL>><UserCount=<ユーザ数>><MinimumSize=<コネクション最小数>><MaximumSize=<コネクション最大
数>><RetryInterval=<リトライ待ち時間>><RetryCount=<リトライ回数>><PoolCount=<プールの数>>
・・・
・・・
```

プールリストを表示する場合

```
<DatabaseID=<データベースID1>><UserID=<ユーザID1>><WaitCount=<コネクション取得待ちカウンタ
>><ConnectionCount=<コネクションの数>>
<DatabaseID=<データベースID1>><UserID=<ユーザID2>><WaitCount=<コネクション取得待ちカウンタ
>><ConnectionCount=<コネクションの数>>
<DatabaseID=<データベースID2>><UserID=<ユーザID3>><WaitCount=<コネクション取得待ちカウンタ
>><ConnectionCount=<コネクションの数>>
・・・
・・・
```

コネクションリストを表示する場合

```
<DatabaseID=<データベースID1>><UserID=<ユーザID1>><ConnectionID=<コネクション
ID1>><ConnectionBusy=<使用中フラグ>><ConnectionReuse=<再利用可否フラグ>><ConnectionTime=<コ
ネクション生成時刻>><LastTime=<最終アクセス時刻>>
<DatabaseID=<データベースID1>><UserID=<ユーザID2>><ConnectionID=<コネクション
ID2>><ConnectionBusy=<使用中フラグ>><ConnectionReuse=<再利用可否フラグ>><ConnectionTime=<コ
ネクション生成時刻>><LastTime=<最終アクセス時刻>>
<DatabaseID=<データベースID2>><UserID=<ユーザID3>><ConnectionID=<コネクション
ID3>><ConnectionStatus=NotUsed><ReuseStatus=NotReuse><ConnectionTime=2003/11/25
11:36:00.616><LastTime=2003/11/25 11:40:00.313>
<DatabaseID=<データベースID2>><UserID=<ユーザID3>><ConnectionID=<コネクション
ID4>><ConnectionStatus=Used><ReuseStatus=Reuse><ConnectionTime=2003/11/25
11:37:00.256><LastTime=2003/11/25 11:37:00.256>
・・・
・・・
```

すべてのプール管理情報を表示する場合

```
<DatabaseID=<データベースID1>><ClassName=<データベースクラス>><URL=<データベース
URL>><UserCount=<ユーザ数>><MinimumSize=<コネクション最小数>><MaximumSize=<コネクション最大
数>><RetryInterval=<リトライ待ち時間>><RetryCount=<リトライ回数>><PoolCount=<プールの数>>
<DatabaseID=<データベースID1>><UserID=<ユーザID1>><WaitCount=<コネクション取得待ちカウンタ
>><ConnectionCount=<コネクションの数>>
<DatabaseID=<データベースID1>><UserID=<ユーザID1>><ConnectionID=<コネクション
ID1>><ConnectionStatus=<使用中フラグ>><ReuseStatus=<再利用可否フラグ>><ConnectionTime=<コネ
クション生成時刻>><LastTime=<最終アクセス時刻>>
<DatabaseID=<データベースID1>><UserID=<ユーザID2>><WaitCount=<コネクション取得待ちカウンタ
>><ConnectionCount=<コネクションの数>>
<DatabaseID=<データベースID1>><UserID=<ユーザID2>><ConnectionID=<コネクション
ID2>><ConnectionStatus=<使用中フラグ>><ReuseStatus=<再利用可否フラグ>><ConnectionTime=<コネ
クション生成時刻>><LastTime=<最終アクセス時刻>>
<DatabaseID=<データベースID2>><ClassName=<データベースクラス>><URL=<データベース
URL>><UserCount=<ユーザ数>><MinimumSize=<コネクション最小数>><MaximumSize=<コネクション最大
数>><RetryInterval=<リトライ待ち時間>><RetryCount=<リトライ回数>><PoolCount=1>
<DatabaseID=<データベースID2>><UserID=<ユーザID3>><WaitCount=<コネクション取得待ちカウンタ
>><ConnectionCount=2>
<DatabaseID=<データベースID2>><UserID=<ユーザID3>><ConnectionID=<コネクション
ID3>><ConnectionStatus=NotUsed><ReuseStatus=NotReuse><ConnectionTime=2003/11/25
11:36:00.616><LastTime=2003/11/25 11:40:00.313>
```

```
<DatabaseID=<データベースID2>><UserID=<ユーザID3>><ConnectionID=<コネクション
ID4>><ConnectionStatus=Used><ReuseStatus=Reuse><ConnectionTime=2003/11/25
11:37:00.256><LastTime=2003/11/25 11:37:00.256>
・・・
・・・
```

(6)　戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

(7)　注意事項

- 引数は，コマンド名の直後に指定する必要があります。引数は，コマンド名より後ろにあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<データベース ID> -dbid）やオプション名と値を非対応にすること（例：-dbid timeout -type <データベース ID>）はできません。

3.10.10　cjwebsetup（Web コンテナサーバのセットアップとアンセットアップ）

(1)　形式

セットアップの場合

```
cjwebsetup［<サーバ名称>］
```

アンセットアップの場合

```
cjwebsetup -d <サーバ名称>
```

(2)　機能

指定したサーバ名称の Web コンテナの環境をセットアップまたはアンセットアップします。

-d オプションを指定した場合，次に示す Web コンテナの環境（ディレクトリおよびファイル）の<サーバ名称>以下を削除します。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/

(3)　引数

<サーバ名称>

セットアップまたはアンセットアップする Web コンテナのサーバ名称を指定します。

サーバ名称には，半角英数字（0～9，A～Z，a～z），アンダースコア（_）またはハイフン（-）が使用できます。サーバ名称の先頭の文字は英数字でなければなりません。また，サーバ名称は，「3.7.8 Web コンテナサーバの作業ディレクトリ」を参照して見積もった範囲内の長さで指定できます。

サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。ただし，アンセットアップ時は省略できません。また，ホスト名称にサーバ名称として使用できない文字が含まれている場合はエラーになります。

-d

Web コンテナの環境を削除する場合に指定します。

(4)　戻り値

0：

正常終了しました。

1：

異常終了しました。

(5)　注意事項

- セットアップ済みのサーバ名称を指定して再セットアップ（cjwebsetup <サーバ名称>）を行うと，エラーとなります。
- Windows の場合，サーバ名称の大文字・小文字は区別されません。作成済みのサーバ名称と，大文字・小文字の区別以外は一致している文字列を cjwebsetup のサーバ名称として指定した場合，次のように動作します。
  - セットアップの場合(cjwebsetup <サーバ名称>)
    セットアップに失敗します。
  - アンセットアップの場合(cjwebsetup -d <サーバ名称>)
    すでに作成済みである，<サーバ名称>と大文字・小文字の区別以外は一致しているサーバが削除されます。
- Web コンテナサーバの開始または停止コマンドを実行する場合，これらのコマンドにサーバ名称を引数として指定するときは，cjwebsetup コマンドで作成したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。
- 起動中の Web コンテナサーバに対して，-d オプションで削除しないでください。
- Windows の場合，削除対象のファイルの属性が読み取り専用，または使用中のときに，-d オプションで削除しないでください。
- UNIX の場合，cjwebsetup コマンドの実行には，root 権限（Component Container 管理者を設定していない場合），または Component Container 管理者の権限（Component Container 管理者を設定している場合）が必要です。
- UNIX の場合，削除対象のディレクトリ，ファイルのアクセス権の設定および使用状態に関係なく，すべて削除します。
- Web コンテナサーバを再セットアップする場合，cjwebsetup -d <サーバ名称>でサーバの環境をいったん削除してから再度セットアップコマンドを実行してください。

3.11　Web コンテナサーバで使用するファイル

サーブレットエンジンモードでは，次のコマンドを使用できます。

- Web サーバ連携で使用するファイル
- ログの運用で使用するファイル
- 統合ユーザ管理で使用するファイル
- Web アプリケーションで使用するファイル
- DABroker Library で使用するファイル
- Web コンテナサーバで使用するファイル

これらのファイルのうち，「Web コンテナサーバで使用するファイル」以外のファイルについては，J2EE サーバモードで使用するファイルと同じです。

マニュアル「アプリケーションサーバ リファレンス コマンド編」，およびマニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」を参照してください。

Web コンテナサーバのオプション定義ファイル（usrconf.cfg）に設定する JavaVM 拡張オプションについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「16.　JavaVM 起動オプション」を参照してください。

Web コンテナサーバで使用するファイルの一覧を，次の表に示します。

表 3-62　Web コンテナサーバで使用するファイルの一覧

ファイル名	分類	概要	参照先
hitachi_web.properties	Web コンテナサーバ用 Web アプリケーションプロパティファイル	Web アプリケーション個別のプロパティを指定します。	3.11.1
usrconf.cfg	Web コンテナサーバ用オプション定義ファイル	Web コンテナサーバ全体でのシステムプロパティを指定します。	3.11.2
usrconf.properties	Web コンテナサーバ用ユーザプロパティファイル	Web コンテナサーバ全体でのシステムプロパティを指定します。	3.11.3
web-users.xml	ユーザ認証情報定義ファイル	各ユーザのユーザ名，パスワード，および所属するロールを記述します。	3.11.4
web.policy	SecurityManager 定義ファイル	SecurityManager を設定して，Java 2 から導入されたセキュリティポリシーに基づく Web コンテナの実行時保護をします。	3.11.5
（任意）	プール管理情報設定ファイル	プール管理情報を記述します。	3.11.6

3.11.1　hitachi_web.properties（Web コンテナサーバ用 Web アプリケーションプロパティファイル）

(1)　形式

次のようにキーを指定します。

〈キー名称〉＝〈値〉

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。
- 値が存在しない行を定義した場合，その行は無視されます。

(2)　ファイルの格納先

- Windows の場合
  <Web アプリケーションの WEB-INF ディレクトリ>¥
- UNIX の場合
  <Web アプリケーションの WEB-INF ディレクトリ>/

(3)　機能

Web アプリケーション個別のプロパティを指定します。なお，Web コンテナサーバ全体でのシステムプロパティ設定は，usrconf.properties で指定します。usrconf.properties と WEB-INF/hitachi_web.properties で同じキーが指定されている場合は，WEB-INF/hitachi_web.properties の値が優先されます。

(4)　指定できるキー

指定できるキーとデフォルト値を次に示します。このキーに不正な値を指定した場合, 動作は保証されません。

キー名称	内容	デフォルト値
webserver.context.reload_delay_timeout	リロードの遅延実行機能を使用するかどうかを，-2147483648～2147483647 の整数で指定します。 • リロードの遅延実行機能を使用し，最大遅延期間（単位：秒）を設定する場合は，1 以上の整数で指定します。 • リロードの遅延実行機能を使用し，最大遅延期間（単位：秒）を設定しない場合は，負の整数で指定します。 • リロードの遅延実行機能を使用しない場合は，0 を指定します。	0
webserver.context.reloadable	Web アプリケーションのクラスファイルの更新を自動検知するかどうかを指定します。 true を指定した場合： 更新を自動検知します。 false を指定した場合： 更新を自動検知しません。	false

キー名称	内容	デフォルト値
webserver.context.update.interval	更新するファイルのコピーに必要な時間（単位：秒）をインターバルとして指定します。0～2147483647 の整数で指定します。 なお，インターバルは余裕のある値にしてください。 リソースの更新を検知してから指定したインターバルのあとに，処理中リクエストの監視を開始します。	0
webserver.jsp.keepgenerated	JSP ファイルをコンパイルした結果得られた Java ファイルを保持するかどうかを指定します。 true を指定した場合： 　Java ファイルを保持します。 false を指定した場合： 　Java ファイルを保持しません。	false
webserver.jsp.recompilable	JSP ファイルが更新されたときに自動的に再コンパイルするかどうかを指定します。 true を指定した場合： 　自動的に再コンパイルします。 false を指定した場合： 　自動的に再コンパイルしません。	false
webserver.jsp.update.interval	更新する JSP ファイルのコピーに必要な時間（単位：秒）をインターバルとして指定します。0～2147483647 の整数で指定します。 なお，インターバルは余裕のある値にしてください。 JSP ファイルの更新を検知してから指定したインターバルのあとに，JSP の再コンパイルが実行されます。	0
webserver.session.persistence.enabled	リロード時のセッション継続機能を使用するかどうかを指定します。 true を指定した場合： 　セッション継続機能は有効になります。 false を指定した場合： 　セッション継続機能は無効になります。	false
webserver.xml.validate	Web アプリケーションのデプロイメント・ディスクリプタ（web.xml ファイル），およびタグライブラリ・ディスクリプタ（TLD ファイル）が DTD で記述されたスキーマに沿って書かれているかチェックするかどうかを指定します。 true を指定した場合： 　チェックします。 false を指定した場合： 　チェックしません。	true

(5)　記述例

```
webserver.context.reload_delay_timeout=0
webserver.context.reloadable=false
webserver.context.update.interval=0
webserver.jsp.keepgenerated=false
webserver.jsp.recompilable=false
webserver.jsp.update.interval=0
webserver.session.persistence.enabled=true
webserver.xml.validate=true
```

3.11.2　usrconf.cfg（Web コンテナサーバ用オプション定義ファイル）

(1)　形式

次のようにキーを指定します。

〈キー名称〉＝〈値〉

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。
- 値が存在しない行を定義した場合，その行は無視されます。
- 値の後ろには，空白やコメントなどの文字列は追加できません。追加した場合，不正な値と解釈されます。
  （例）〈キー名称〉＝〈値〉#〈コメント〉
- 実行時の文字エンコーディングと異なる文字エンコーディングでは記述できません。

(2)　ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/

(3)　機能

Web コンテナサーバを実行する JavaVM の起動オプションを指定します。

Web コンテナサーバの稼働中に，このファイルの内容を変更した場合，変更した内容は，次回 Web コンテナサーバを起動したときに反映されます。

(4)　指定できるキー

指定できるキーとデフォルト値を次に示します。このキーに不正な値を指定した場合は，動作は保証されません。

ポイント

ここでは，次の内容について説明します。

- **Web コンテナサーバ用 usrconf.cfg ファイルで指定できるキーの概要**
  指定できるキーのうち，J2EE サーバ用 usrconf.cfg ファイルと共通のキーについては，ここでは詳細を説明しません。
  共通のキーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.3　usrconf.cfg（J2EE サーバ用オプション定義ファイル)」を参照してください。参照する場合には，「J2EE サーバ」，「J2EE アプリケーション」を「Web コンテナサーバ」，「Web アプリケーション」と読み替えてください。
  なお，J2EE サーバ用のキーと指定内容が異なる場合には，ここでキーの詳細について説明します。
- **J2EE サーバ用のキーとの差異（J2EE サーバ用 usrconf.cfg ファイルのキーと指定内容が異なる場合）**
  Web コンテナサーバ用のキーと J2EE サーバ用のキーの指定内容が異なる場合には，キーの詳細について説明します。差異の有無は「差異」列に表記します。

表 3-63　usrconf.cfg に指定できるキーとデフォルト値（Web コンテナサーバ）

キー名称	差異	内容	デフォルト値
add.jvm.arg	あり	指定されたオプションを使って JavaVM を起動します。 システムプロパティを指定したい場合は，"-D"を利用せずに，usrconf.properties ファイルに指定します。 複数のオプションを指定する場合，同じキー名称を使用して，次のように複数回指定してください。 （指定例） add.jvm.arg= -Xms128m add.jvm.arg= -Xmx256m なお，次のように複数のオプションを指定できません。指定をした場合，JavaVM の初期化に失敗します。 （指定例） add.jvm.arg=-Xms128m -Xmx256m add.jvm.arg で指定できる JavaVM のオプションについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「16.1　JavaVM 拡張オプションの一覧」および「16.5　Application Server で指定できる Java HotSpot VM のオプション」を参照してください。 ただし，Web コンテナサーバでは G1GC をサポートしていません。「-XX:+UseG1GC」を指定しないでください。 add.jvm.arg キーで定義されていないオプションのデフォルト値はマニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「16.4　Application Server で指定できる Java HotSpot VM のオプションのデフォルト値」を参照してください。	デフォルト値については「3.11.2(5) Web コンテナサーバでの JavaVM オプションのデフォルト値」を参照してください。
cpp.library.version	なし	プロセス内で使用する libstdc++ライブラリのバージョンを指定します。なお，このキーは，Linux 用です。 指定できる文字列を次に示します。なお，09-00 以降で設定できるのは「6」だけです。 6： libstdc++.so.6 ライブラリを使用します。 Red Hat Enterprise Linux 以外のプラットフォームで設定しても無効です。	6
web.add.class.path	独自	コンテナ拡張ライブラリ用の JAR を指定します。複数のオプションを指定する場合，同じキー名称を使用して，複数回指定してください。 スペースを含むパスを指定する場合，パスをダブルクォーテーション（"）で囲む必要はありません。また，ダブルクォーテーションでの指定は有効となりません。 注意事項 このキーで追加したクラスパスを使用する Web アプリケーションは，J2EE 1.3 仕様に従わないアプリケーションになります。将来的に J2EE 環境での実行を予定しているアプリケーションでは，このキーを使用しないで，WAR ファイルの WEB-INF/lib 以下に，Web アプリケーションで使用するすべての JAR ファイルを格納してください。	なし

キー名称	差異	内容	デフォルト値
add.library.path※1	なし	JNI 用の共有ライブラリを指定します。コンテナ拡張ライブラリが JNI を利用する場合だけ指定します。	−
web.server.corefilenum（UNIX 用）	独自	Web コンテナサーバ再起動時に残す core ファイルの数を 0～16 の整数で指定します。なお，このキーは，UNIX 用です。 次に示す対象 core ファイルのうち，作成日時が新しいものから指定した数のファイルを残し，それ以外のファイルは削除されます。通常は Web コンテナサーバダウン時に出力する core ファイル数の 1 を指定します。 対象 core ファイル /opt/Cosminexus/CC/web/containers/<サーバ名称>/core* • 再起動時にすべての core ファイルを削除する場合は，0 を指定します。 • 手動による運用または Management Server を使用する場合は，1 以上を指定します。 対象 core ファイルのうち，上書き，または削除される可能性のある core ファイルについては再起動時にリネームされます。 Linux（プロセス ID 付きの設定）の場合 core.<core 生成プロセス ID> → core.<core 生成プロセス ID>.<core ファイル作成日時> 上記以外の場合 core → core.<core ファイル作成日時> <core ファイル作成日時>の形式は，yymmddHHMMSS です。 なお，core ファイルのリネーム，または削除に失敗した場合はメッセージ KDJE40047-E を出力し，Web コンテナサーバ起動処理を中止して異常終了します。 また，次のことに注意してください。 • 対象 core ファイルと同じ名称のファイルを作成した場合は削除されます。 • core ファイルの削除は Web コンテナサーバ再起動時に処理されるため，再起動するまでは定義数以上の core ファイルが出力されている場合があります。 • 定義数+n 分の core ファイルが残るためディスク容量に注意してください（n:再起動されるまでに Web コンテナサーバが出力する core ファイル数）。	1
web.server.log.directory	独自	指定されているディレクトリをログファイル※2 の出力先として使用します。パス長は，1～200 バイトで指定します。Windows の場合，UNC 名を含むパスは指定できません。UNIX の場合，nfs マウントされたディスク上へのパスは指定できません。 このキーを指定する場合，サーバ開始前にログ出力先ディレクトリを作成してください。存在しないディレクトリが指定された場合は，J2EE サーバまたは Web コンテナサーバ	• Windows の場合 <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs • UNIX の場合

キー名称	差異	内容	デフォルト値
web.server.log.directory	独自	開始時に KDJE40024-E のメッセージが出力され，異常終了します。 ほかの J2EE サーバや Web コンテナサーバとログ出力先が，同じディレクトリにならないように注意してください。同じディレクトリを指定した場合，動作は保証されません。 なお，このキーを指定して，デフォルトの出力先以外にログを出力する場合，変更先ディレクトリ下のログファイルはサーバをアンセットアップする時に削除されません。ログファイルを削除したい場合には，手動で削除してください。	/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs
web.server.log.stdout.filesize	独自	cjstdout.log ファイルのサイズの上限値を 4096～2147483647 の範囲（単位：バイト）の整数で指定します。	1048576
web.server.log.stderr.filesize	独自	cjstderr.log ファイルのサイズの上限値を 4096～2147483647 の範囲（単位：バイト）の整数で指定します。	1048576
jvm.type	なし	使用する JavaVM のタイプを指定します。	－

（凡例）

あり：

Web コンテナサーバ用 usrconf.cfg ファイルのキーと J2EE サーバ用 usrconf.cfg ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

なし：

Web コンテナサーバ用 usrconf.cfg ファイルと J2EE サーバ用 usrconf.cfg ファイルで共通のキーです。キーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.3　usrconf.cfg (J2EE サーバ用オプション定義ファイル)」を参照してください。

独自：

Web コンテナサーバ用 usrconf.cfg ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

－：

Web コンテナサーバ用 usrconf.cfg ファイルと J2EE サーバ用 usrconf.cfg ファイルで共通のデフォルト値です。デフォルト値の詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.3 usrconf.cfg（J2EE サーバ用オプション定義ファイル)」を参照してください。

注※1

cjstartsv コマンドまたは cjstartweb コマンド実行時，add.library.path と OS の環境変数（LIBPATH など）の両方が設定されている場合，add.library.path の設定が優先されます。

注※2

次に示すログファイルが変更の対象になります。

ログの種類	内容	ログ取得ディレクトリおよびログファイル名
メッセージログ	稼働ログ	• Windows の場合 <web.server.log.directory キーの値>¥cjweb_message?.logs • UNIX の場合 <web.server.log.directory キーの値>/cjweb_message?.logs
	ログ稼働ログ	• Windows の場合 <web.server.log.directory キーの値>¥cjlogger.log • UNIX の場合 <web.server.log.directory キーの値>/cjlogger.log

ログの種類	内容	ログ取得ディレクトリおよびログファイル名
ユーザログ	Web サーブレットログ	• Windows の場合 <web.server.log.directory キーの値>¥web_servlet?.log • UNIX の場合 <web.server.log.directory キーの値>/web_servlet?.log
	ユーザ出力ログ	• Windows の場合 <web.server.log.directory キーの値>¥user_out?.log • UNIX の場合 <web.server.log.directory キーの値>/user_out?.log
	ユーザエラーログ	• Windows の場合 <web.server.log.directory キーの値>¥user_err?.log • UNIX の場合 <web.server.log.directory キーの値>/user_err?.log
	JavaVM の保守情報および GC のログ	• Windows の場合 <web.server.log.directory キーの値>¥javalog??.log • UNIX の場合 <web.server.log.directory キーの値>/javalog??.log
保守用ログ	保守情報	• Windows の場合 <web.server.log.directory キーの値>¥cjweb_maintenance?.log • UNIX の場合 <web.server.log.directory キーの値>/cjweb_maintenance?.log
	障害発生時の例外情報	• Windows の場合 <web.server.log.directory キーの値>¥cjweb_exception?.log • UNIX の場合 <web.server.log.directory キーの値>/cjweb_exception?.log
	Web コンテナの保守情報	• Windows の場合 <web.server.log.directory キーの値>¥cjwebcontainer?.log • UNIX の場合 <web.server.log.directory キーの値>/cjwebcontainer?.log
	終了プロセス情報	• Windows の場合 <web.server.log.directory キーの値>¥cjweb_shutdown?.log • UNIX の場合 <web.server.log.directory キーの値>/cjweb_shutdown?.log
	起動プロセス標準出力情報	• Windows の場合 <web.server.log.directory キーの値>¥cjstdout.log • UNIX の場合 <web.server.log.directory キーの値>/cjstdout.log
	起動プロセス標準出力情報バックアップ	• Windows の場合 <web.server.log.directory キーの値>¥cjstdout_save.log • UNIX の場合 <web.server.log.directory キーの値>/cjstdout_save.log

ログの種類	内容	ログ取得ディレクトリおよびログファイル名
保守用ログ	起動プロセス標準エラー情報	• Windows の場合 <web.server.log.directory キーの値>¥cjstderr.log • UNIX の場合 <web.server.log.directory キーの値>/cjstderr.log
	起動プロセス標準エラー情報バックアップ	• Windows の場合 <web.server.log.directory キーの値>¥cjstderr_save.log • UNIX の場合 <web.server.log.directory キーの値>/cjstderr_save.log
	Web コンテナ保守用トレースログ	• Windows の場合 <web.server.log.directory キーの値>¥cjweb_access?.log • UNIX の場合 <web.server.log.directory キーの値>/cjweb_access?.log
TPBroker のトレース	TPBroker のトレース情報	• Windows の場合 <web.server.log.directory キーの値>¥TPB 配下 • UNIX の場合 <web.server.log.directory キーの値>/TPB 配下

(5)　Web コンテナサーバでの JavaVM オプションのデフォルト値

Web コンテナサーバでの Java VM オプションのデフォルト値を示します。

- -Xms128m
- -Xmx256m
- -XX:MetaspaceSize=64m
- -XX:MaxMetaspaceSize=64m
- -XX:SurvivorRatio=8
- -XX:+HitachiOutOfMemoryStackTrace
- -XX:HitachiJavaLog:<web.server.log.directory の指定値>
- -XX:HitachiJavaLogFileSize=4m
- -XX:-HitachiThreadDumpToStdout
- -XX:+HitachiOutOfMemoryAbort
- -XX:+HitachiVerboseGC
- -XX:+HitachiVerboseGCPrintCause
- -XX:+HitachiOutputMilliTime
- -XX:+HitachiJavaClassLibTrace
- -XX:HitachiJavaClassLibTraceLineSize=1024
- -XX:+HitachiLocalsSimpleFormat
- -XX:+HitachiTrueTypeInLocals
- -XX:+HitachiLocalsInStackTrace

(6)　記述例

- Windows の場合

```
#java vm options
add.jvm.arg=-Xms128m
add.jvm.arg=-Xmx256m
add.jvm.arg=-XX:MetaspaceSize=64m
add.jvm.arg=-XX:MaxMetaspaceSize=64m
add.jvm.arg=-XX:+HitachiVerboseGC
add.jvm.arg=-XX:HitachiVerboseGCIntervalTime=600

# system classes
web.add.class.path=c:¥mylib¥classes

# ADM service

# web.server.log.directory
web.server.log.directory=c:¥worklogdir
```

- UNIX の場合

```
#java vm options
add.jvm.arg=-Xms128m
add.jvm.arg=-Xmx256m
add.jvm.arg=-XX:MetaspaceSize=64m
add.jvm.arg=-XX:MaxMetaspaceSize=64m
add.jvm.arg=-XX:+HitachiVerboseGC
add.jvm.arg=-XX:HitachiVerboseGCIntervalTime=600

# system classes
web.add.class.path=/opt/mylib/classes

# web.server.log.directory
web.server.log.directory=/CClogs/web/servername

# web.server.corefilenum
web.server.corefilenum=3
```

(7)　注意事項

- コマンドプロンプトで設定されている CLASSPATH 環境変数は，Web コンテナサーバに渡りませんが，そのほかの環境変数（PATH など）は渡ります。
- OS のロケールと異なる文字エンコーディングで記述しないでください。また，OS のロケールに UTF-8 を使用する場合，BOM 付き UTF-8 では記述しないでください。
- usrconf.properties と usrconf.cfg の add.jvm.arg の-D に同じキーが指定されている場合は，usrconf.properties の値が優先されます。

3.11.3　usrconf.properties (Web コンテナサーバ用ユーザプロパティファイル)

(1)　形式

J2SE のプロパティファイル形式です。

次のようにキーを指定します。

〈キー名称〉 = 〈値〉

指定方法

- 改行までが値になります。
- #で始まる行はコメントとみなされます。

- 値が存在しない行を定義した場合，その行は無視されます。
- 値の後ろには，空白やコメントなどの文字列は追加できません。追加した場合，不正な値と解釈されます。
  （例）〈キー名称〉＝〈値〉#〈コメント〉
- 記載する文字は Java の仕様に従って，ISO 8859-1 文字エンコーディングを使用してください。
- Java の仕様に従わない形式の場合，Web コンテナサーバの起動に失敗することがあります。
  （例）不正な Unicode エスケープシーケンスを含む場合（”¥u”に続く文字列が適切な Unicode 16 進数の値ではない）。

(2)　ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/

(3)　機能

Web コンテナサーバ全体でのシステムプロパティを指定します。なお，Web アプリケーション個別の定義は，WEB-INF/hitachi_web.properties で指定します。usrconf.properties と WEB-INF/hitachi_web.properties で同じキーが指定されている場合は，WEB-INF/hitachi_web.properties の値が優先されます。また，usrconf.properties と usrconf.cfg の add.jvm.arg の-D に同じキーが指定されている場合は，usrconf.properties の値が優先されます。

Web コンテナサーバの稼働中に，このファイルの内容を変更した場合，変更した内容は，次回 Web コンテナサーバを起動したときに反映されます。

(4)　Web コンテナサーバの予約済みキー

usrconf.properties 内では，ユーザが独自のキーを新たに定義することもできます。定義されたキーは，java.util.Properties クラスを使用することで，Web アプリケーション内で値を取り出せます。ただし，Web コンテナサーバでは，次に示す接頭子で始まるキーを内部的に利用するため，アプリケーションでこれらの接頭子で始まるキーを利用してはいけません。

- ejbserver.*
- webserver.*
- appclient.*
- j2eeserver.*

(5)　Web コンテナサーバのカスタマイズ用キー

Web コンテナサーバでは，次に示すシステムプロパティのキーに値を設定することで，Web コンテナサーバの動作をカスタマイズできます。このキーに不正な値を指定した場合，動作は保証されません。

ここでは，キーを次の分類に分けて説明します。

- ejbserver.application から始まるキー
- ejbserver.client から始まるキー
- ejbserver.container から始まるキー

- ejbserver.jndi から始まるキー
- ejbserver.jta から始まるキー
- ejbserver.logger から始まるキー
- ejbserver.rmi から始まるキー
- ejbserver.server から始まるキー
- ejbserver.stdoutlog から始まるキー
- java から始まるキー
- vbroker から始まるキー
- webserver.connector から始まるキー
- webserver.container から始まるキー
- webserver.context から始まるキー
- webserver.errorpage から始まるキー
- webserver.http から始まるキー
- webserver.jsp から始まるキー
- webserver.servlet から始まるキー
- webserver.session から始まるキー
- webserver.shutdown から始まるキー
- webserver.static から始まるキー
- webserver.work から始まるキー
- webserver.xml から始まるキー

ポイント

ここでは，次の内容について説明します。

- **Web コンテナサーバ用 usrconf.properties ファイルで指定できるキーの概要**

  指定できるキーのうち，J2EE サーバ用 usrconf.properties ファイルと共通のキーについては，ここでは詳細を説明しません。

  共通のキーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.4　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。Web コンテナサーバの場合には，「J2EE サーバ」，「J2EE アプリケーション」を「Web コンテナサーバ」，「Web アプリケーション」と読み替えてください。

  なお，J2EE サーバ用のキーと指定内容が異なる場合には，ここでキーの詳細についても説明します。

- **J2EE サーバ用のキーとの差異（J2EE サーバ用 usrconf.properties ファイルのキーと指定内容が異なる場合)**

  Web コンテナサーバ用のキーと J2EE サーバ用のキーの指定内容が異なる場合には，キーの詳細について説明します。差異の有無は「差異」列に表記します。

- **Web コンテナサーバ用 usrconf.properties ファイル独自のキーの詳細**

  Web コンテナサーバ用 usrconf.properties ファイル独自のキーについては，ここでキーの詳細を説明します。

(a)　ejbserver.application から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.application から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.application.InitTermProcessClasses	なし	コンテナ拡張ライブラリを使用する場合に，サーバ起動・停止フックのクラス名を指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.appname	なし	AppName のフィールドに出力するデフォルトのアプリケーション名を指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.count	なし	ログファイルの面数を指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.encoding	なし	出力する文字列のエンコーディングを指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.filter	なし	使用するフィルタを指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.formatter	なし	使用するフォーマッタを指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.level	なし	ログの取得レベルの上限を指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.limit	なし	ログファイルの容量（単位：バイト）を指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.msgid	なし	MsgID のフィールドに出力するデフォルトのメッセージ ID を指定します。	－
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.path	あり	ログファイル名のプリフィックスを 1～255 バイトの範囲で指定します。パスは，相対パスで指定します※2。なお，パス名の終端に，半角数値を指定しないでください。 ここで指定したプリフィックスに"xx（1～16 の整数）.log"が付加された値が，トレース情報ファイルの名称になります。 パスに指定された半角英文字の大文字と小文字は区別されません。また，パス値の長さは，次の A+B+C の値が範囲に収まるよう指定してください。 • A（Windows の場合）：<ログ出力先ルート(web.server.log.directory の値)>¥user¥（デフォルトは<Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥user¥） • A（UNIX の場合）：<ログ出力先ルート(web.server.log.directory の値)>/user/（デフォルトは/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/user/）	－

キー名称	差異	内容	デフォルト値
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.path	あり	• B：ユーザの指定したプリフィックスの長さ • C："xx.log"の 6 文字 このほか，パスを指定するときには次の点に注意してください。 • Properties ファイルには native の 2 バイトコードなどは使用できないため，native2ascii などを利用して Unicode の文字列を取得して指定してください。 • 絶対パスで指定しないでください。また，相対パスの".. ¥"（UNIX の場合，"../"）は使用しないでください。 • パスに日本語文字などを含む場合は，その文字を処理できるロケール設定がされている環境下で JavaVM が動作している必要があります。 • パスをほかのハンドラと同一に設定することはできません。そのような設定にした場合，そのハンドラは作成されません。	−
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>※1.separator	なし	CJSimpleFormatter を利用して，1 文でメッセージを出力するためのセパレータを指定します。	−
ejbserver.application.userlog.loggers	なし	使用するロガーの名称を指定します。	−
ejbserver.application.userlog.Logger.<ロガー名称>※3.filter	なし	ロガーで使用するフィルタを指定します。	−
ejbserver.application.userlog.Logger.<ロガー名称>※3.handlers	なし	指定のロガー名称に対して接続（addHandler(Handler)）するハンドラクラスを指定します。	−
ejbserver.application.userlog.Logger.<ロガー名称>※3.level	なし	ロガーのログ出力レベルを指定します。	−
ejbserver.application.userlog.Logger.<ロガー名称>※3.useParentHandlers	なし	ログレコードを受け取ったロガーから，親ロガーの接続しているハンドラへログレコードを伝播させるかどうかを指定します。	−

（凡例）

あり：

Web コンテナサーバ用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

−：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

注

ユーザログ機能についてのプロパティ定義（ejbserver.application.InitTermProcessClasses キー以外）で使用できるそれぞれの文字セットで指定できる値は，次のとおりです。

- **クラス文字セット（クラス名の定義に使用する文字セット）**
  半角英数字（a～z，A～Z，0～9），ピリオド（.），ドル記号（$）およびアンダースコア（_）
- **半角文字セット**
  半角英数字（a～z，A～Z，0～9），ピリオド（.），ドル記号（$），アンダースコア（_），ハイフン（-），ストローク（|），コロン（:），アンパサンド（&），単価記号（@），シャープ（#）およびパーセント（%）

指定した値が，使用できる文字セットの条件を満たさない，または指定できる値の範囲を満たさない不正な値の場合は，警告メッセージが出力され，その時点で有効なデフォルトの値が採用されます。その時点で有効なデフォルトの値とは，<ハンドラ名称>=default の場合のそのキーの値，または「デフォルト値」欄のデフォルト値を指します。

注※1

ハンドラ名称はプロパティの値を区別するために使用します。半角英数字で始まる 1～1024 バイトのクラス文字セットで指定してください。"default"を指定した場合は，すべての CJLogHandler が共通のデフォルト値として認識するプロパティを示します。

注※2

出力先の起点は，次の規則に従ってください。

<web.server.log.directory の値>¥user または<web.server.log.directory の値>/user

web.server.log.directory の値は，web.server.log.directory の仕様に従ってください。Web コンテナサーバの web.server.log.directory のデフォルトの値を次に示します。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<コンテナサーバ名>¥logs¥user¥<ユーザ指定値>
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<コンテナサーバ名>/logs/user/<ユーザ指定値>

注※3

ロガー名称は，Logger.getLogger（<ロガー名称>）でロガーのインスタンスを取得するときに指定する"."付きの名称で，あらかじめ，ejbserver.application.userlog.loggers プロパティで宣言する必要があります。ロガー名称については，java.util.logging.Logger の仕様に従ってください。

半角英数字で始まる 1～1024 バイトのクラス文字セットで指定してください。"default"を指定した場合は，すべての CJLogHandler が共通のデフォルト値として認識するプロパティを示します。

なお，ロガー名称で識別されるプロパティは，ユーザログ機能のロガークラスの初期化で使用されます。

ロガー名称の最後を.handlers で終了させた場合，ejbserver.application.userlog.Logger.<ロガー名称>.handlers のプロパティ指定時に紛らわしくなるため，ロガー名称は".handlers"で終了させないことを推奨します。

(b)　ejbserver.client から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.client から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.client.ctm.RequestPriority	なし	CTM 内のキューに滞留したリクエストを引き出す時のプライオリティ（優先順位）を指定します。	−

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

−：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(c)　ejbserver.container から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.container から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.container.rebindpolicy	なし	EJB メソッドの呼び出し時に通信障害が発生した場合の，EJB クライアント側でのコネクションの再接続動作とリクエストの再送動作を指定します。	−

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

−：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(d)　ejbserver.jndi から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.jndi から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.jndi.cache	あり	Enterprise Bean を呼び出す場合だけ指定します。 ネーミングでのキャッシングをするかどうかを指定します。 キャッシングをする場合は on を，キャッシングをしない場合は off を指定します。	−
ejbserver.jndi.cache.interval	あり	Enterprise Bean を呼び出す場合だけ指定します。 ネーミングでのキャッシングをする場合，キャッシュをクリアする間隔（単位：秒）を指定します。 ただし，1～2147483647 以外の値を指定した場合，クリアしません。 ネーミングサービスの障害検知機能を使用している場合は，短い値（60 秒）を指定することを推奨します。	−
ejbserver.jndi.cache.interval.clear.option	あり	インターバル経過後のネーミングでのキャッシュ領域に対する動作を決定します。	−

キー名称	差異	内容	デフォルト値
ejbserver.jndi.cache.interval.clear.option	あり	refresh を指定した場合： 　キャッシュの全領域をクリアします。 check を指定した場合： 　無効なキャッシュをクリアします。	−
ejbserver.jndi.cache.reference	なし	DataSource オブジェクトのキャッシング機能を使用するかどうかを指定します。	−
ejbserver.jndi.namingservice.group.list	なし	ラウンドロビン検索をする場合に，検索対象になる論理ネーミングサービスのグループを定義します。	−
ejbserver.jndi.namingservice.group.<Specify group name>.providerurls	なし	各グループに属するネーミングサービスのルート位置を指定します。	−
ejbserver.jndi.request.timeout	なし	ネーミングサービスとの通信タイムアウト時間（単位：秒）を指定します。	−

（凡例）

あり：

Web コンテナサーバ用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

−：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(e)　ejbserver.jta から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.jta から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.jta.pooldeffile.scramble.enabled	独自	DD ファイルをスクランブル化するかどうかを指定します。 true を指定した場合： 　スクランブル化します。 false を指定した場合： 　スクランブル化しません。	false

（凡例）

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

(f)　ejbserver.logger から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.logger から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.logger.channels.define.<チャネル名>※.filenum	なし	Web コンテナサーバのログファイルの面数を指定します。	• チャネル名が WebAccessLogFile の場合 16 • チャネル名が WebExceptionLogFile または WebServletLogFile の場合 4 • チャネル名が上記以外の場合 2
ejbserver.logger.channels.define.<チャネル名>※.filesize	なし	Web コンテナサーバのログファイルのサイズ（単位：バイト）を指定します。	• チャネル名が WebAccessLogFile の場合 2097152 • チャネル名が WebExceptionLogFile の場合 12582912 • チャネル名が WebServletLogFile の場合 4194304 • チャネル名が上記以外の場合 1048576
ejbserver.logger.enabled.*	なし	Web コンテナサーバのログレベルを指定します。	Error
ejbserver.logger.rotationTime	なし	J2EE サーバのログの出力先ファイルを切り替える時刻を HHMMSS の形式で指定します。 指定がない場合は，ログのファイルサイズで出力先を切り替えます。 なお，このキーを指定した場合も，ファイルサイズの指定は有効になります。このキーに指定した時刻になったとき，または出力先ファイルが指定したファイルサイズになったときに，出力先ファイルが切り替わります。	なし
ejbserver.logger.rotationStyle	なし	J2EE サーバのログの出力先ファイル切り替え時のファイル名付与規則を指定します。 SHIFT を指定した場合： シフトモードでファイル名を付与します。 ファイルの合計数は，指定した出力先ファイル面数＋ 1（現在の出力先ファイル分）となります。 WRAP を指定した場合： ラップアラウンドモードでファイル名を付与します。	WRAP

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

注※

チャネル名称として次に示す名称を設定できます。

WebMessageLogFile，WebMaintenanceLogFile，WebExceptionLogFile，WebContainerLogFile，
WebShutdownLogFile，WebServletLogFile，UserOutLogFile，UserErrLogFile，WebAccessLogFile

資料の取得については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「2.3　資料の取得」を参照してください。

(g)　ejbserver.rmi から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.rmi から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.rmi.request.timeout	なし	クライアントとサーバ間の通信タイムアウト時間（単位：秒）を指定します。	－

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(h)　ejbserver.server から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.server から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.server.prf.PRFID	なし	PRF 識別子を指定します。	－

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(i)　ejbserver.stdoutlog から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる ejbserver.stdoutlog から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
ejbserver.stdoutlog.autoflush	なし	ユーザ出力ログおよびユーザエラーログの自動フラッシュ機能を有効にするかどうかを指定します。	－

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(j)　java から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる java から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
java.naming.factory.initial	なし	JNDI の InitialContext がデレゲートするコンテキストの実装クラスのファクトリクラスを指定します。	－
java.naming.provider.url	独自	Enterprise Bean を呼び出す場合だけ J2EE サーバ上の EJB コンテナと通信する場合，CORBA ネーミングサービスのホスト名称とポート番号を指定します。この CORBA ネーミングサービスは，Web コンテナサーバから呼び出される J2EE サーバがネーミングサービスとして利用しているものです。	corbaname::localhost:900

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(k)　vbroker から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる vbroker から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
vbroker.orb.htc.comt.entryCount	なし	TPBroker 通信トレースファイル 1 個当たりのエントリ数の上限値を指定します。	－
vbroker.orb.htc.comt.fileCount	なし	TPBroker 通信トレースファイルの個数の上限値を指定します。	－
vbroker.orb.htc.tracePath	なし	TPBroker のトレースファイル出力先のパスを指定します。	• Windows の場合 <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥logs¥TPB¥logj • UNIX の場合

キー名称	差異	内容	デフォルト値
vbroker.orb.htc.tracePath	なし	TPBroker のトレースファイル出力先のパスを指定します。	/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs/TPB/logj

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(l)　webserver.connector から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.connector から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.connector.ajp12.backlog	なし	旧バージョンとの互換性を保つためのキーです。 webserver.connector.ajp13.backlog として使用されます。指定できる値については，webserver.connector.ajp13.backlog キーの説明を参照してください。 このキーと webserver.connector.ajp13.backlog の両方が指定されていた場合，このキーの値が使用されます。	－
webserver.connector.ajp12.max_threads	なし	旧バージョンとの互換性を保つためのキーです。 webserver.connector.ajp13.max_threads として使用されます。指定できる値については，webserver.connector.ajp13.max_threads キーの説明を参照してください。 このキーと webserver.connector.ajp13.max_threads の両方が指定されていた場合，このキーの値が使用されます。 1025 以上の値が設定されていた場合，webserver.connector.ajp13.max_threads のデフォルト値の 100 が使用されます。	－
webserver.connector.ajp12.port	なし	旧バージョンとの互換性を保つためのキーです。 webserver.connector.ajp13.port として使用されます。指定できる値については，webserver.connector.ajp13.port キーの説明を参照してください。 このキーと webserver.connector.ajp13.port の両方が指定されていた場合，このキーの値が使用されます。	－
webserver.connector.ajp13.backlog	なし	リダイレクタからの接続要求の最大の待ち行列数を指定します。※1	－
webserver.connector.ajp13.bind_host	なし	Web サーバ連携で使用する IP アドレスまたはホスト名称を指定します。※2	－
webserver.connector.ajp13.max_threads	あり	Web コンテナがリクエストを処理する同時実行数を指定します。※3	－

キー名称	差異	内容	デフォルト値
webserver.connector.ajp13.max_threads	あり	1～1024 の整数で指定します。 指定されたリクエストの同時実行数分のスレッドがサーバ起動時に生成されます。	－
webserver.connector.ajp13.port	なし	Web サーバとの通信に使用するポート番号を指定します。※4	－
webserver.connector.ajp13.receive_timeout	なし	リクエスト受信処理のリダイレクタへのデータ要求処理で，リダイレクタからの応答を待つ時間（通信タイムアウト値）を指定します。	－
webserver.connector.ajp13.send_timeout	なし	レスポンス送信処理のタイムアウト値を指定します。	－
webserver.connector.http.bind_host	なし	簡易 Web サーバで使用するローカル IP アドレス，または解決できるローカルホスト名称を指定します。	－
webserver.connector.http.enabled	独自	簡易 Web サーバを使用するかどうかを指定します。 true を指定した場合： 簡易 Web サーバを有効にします。 false を指定した場合： 簡易 Web サーバを無効にします。	• 新規インストールの場合： false • バージョンアップインストールの場合： true
webserver.connector.http.permitted.hosts	なし	簡易 Web サーバへのアクセスを許可するホストの IP アドレスまたはホスト名を指定します。	－
webserver.connector.http.port	独自	簡易 Web サーバのポート番号を 1～65535 の整数で指定します。※4	8080
webserver.connector.limit.max_parameter_count	なし	リクエストパラメタ数の上限値を-1～2147483647 の整数で指定します。	－
webserver.connector.limit.max_post_form_data	なし	POST リクエストのフォームデータ(Content-Type ヘッダの値が application/x-www-form-urlencoded の場合の POST データ)の最大サイズ（単位：バイト）を-1～2147483647 の整数で指定します。	2097152
webserver.connector.redirect_https.port	なし	Web サーバが使用している HTTPS のポート番号を指定します。※4	－

（凡例）

あり：

Web コンテナサーバ用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

注※1

有効な最大値は実行するプラットフォームで指定できる Socket の Listen キューの最大値となります。実際の Listen キューの最大値は OS によって異なるため，詳細は各 OS の listen 関数についてのマニュアルを参照してください。

注※2

同一ホストで実行している Web サーバと Web サーバ連携をしている構成の場合に，このプロパティにローカルホスト名称，または IP アドレスを指定し，リダイレクタのワーカホスト名称に localhost などのループバックアドレスを指定したとき，Web コンテナは Web サーバからのリクエストを受信できません。

注※3

有効な最大値は実行するプラットフォームに依存します。

また，Web サーバに到着するリクエストの一部が Web コンテナに転送されるため，Web コンテナの同時実行数は Web サーバの最大同時接続数より小さく設定する必要があります。なお，データベース操作をするサーブレットや JSP については，データベースコネクションの数よりも多くの多重度は得られないため，Web コンテナの同時実行数を増やす場合は，利用できるデータベースコネクションの数も増やす必要があります。

性能のチューニング時には，次に示す関係を常に考慮して，各パラメタの値を調整してください。

〈Webサーバの最大同時接続数〉 ≧ 〈Webコンテナ単位の最大同時実行スレッド数〉 ≧ 〈データベースコネクションの数〉

Web サーバでの処理の同時接続数については Web サーバのマニュアル，データベースコネクションについては DABroker Library のドキュメントを参照してください。

注※4

すでにほかのアプリケーションで使用または確保されているポート番号は指定できません。また，複数の Web コンテナサーバで Web サーバとの通信に使用するポートのポート番号に同じ値を設定しないでください。同一のポート番号を指定した Web コンテナサーバは cjstartweb コマンドで複数起動できません。

(m)　webserver.container から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.container から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.container.ac.logEnabled	なし	Web コンテナの保守用のトレースログを出力するかどうかを指定します。	－
webserver.container.server_id.enabled	なし	サーバ ID の Cookie を HTTP レスポンスに付加するかどうかを指定します。	－
webserver.container.server_id.name	なし	サーバ ID の Cookie を HTTP レスポンスに付加する場合に，Cookie の名前を指定します。	－
webserver.container.server_id.value	なし	サーバ ID の Cookie を HTTP レスポンスに付加する場合に，Cookie の値を指定します。	Web コンテナを実行するホストの IP アドレスと Web サーバとの通信に使用するポート番号から生成する文字列

キー名称	差異	内容	デフォルト値
webserver.container.servlet.default_mapping.enabled	なし	サーブレットのデフォルトマッピングを有効にするかどうかを指定します。	−

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

−：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(n)　webserver.context から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.context から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.context.check_interval	独自	Web アプリケーションの更新検知インターバルを 1～2147483647 の範囲（単位：秒）の整数で指定します。 Web アプリケーションのリロード機能が無効の場合，このプロパティは設定できません。設定した場合無視されます。 数値以外の文字列，範囲外の数値または空文字列を指定した場合はメッセージを出力し，デフォルト値が設定されます。	1
webserver.context.reload_delay_timeout	独自	Web アプリケーションのリロードの遅延実行機能を使用するかどうかを指定します。 Web アプリケーションのリロード機能が無効の場合，このプロパティは設定できません。設定した場合無視されます。 リロードの遅延実行機能を使用するかどうかは，-2147483648～2147483647 の整数で指定します。 • リロードの遅延実行機能を使用し，最大遅延期間（単位：秒）を設定する場合は，1 以上の整数で指定します。 • リロードの遅延実行機能を使用し，最大遅延期間（単位：秒）を設定しない場合は，負の整数を指定します。 • リロードの遅延実行機能を使用しない場合は，0 を指定します。	0
webserver.context.reloadable	独自	Web アプリケーションのクラスファイルの更新を自動検知するかどうかを指定します。 true を指定した場合： 更新を自動検知します。 false を指定した場合： 更新を自動検知しません。	false
webserver.context.update.interval	独自	更新するファイルのコピーに必要な時間のインターバルを 0～2147483647 の範囲（単位：秒）の整数で指定します。なお，インターバルは余裕のある値にしてください。リソースの更新を検知してから指定したインターバルのあとに，処理中リクエストの監視が開始されます。	0

キー名称	差異	内容	デフォルト値
webserver.context.update.interval	独自	Web アプリケーションのリロード機能が無効の場合，このプロパティは設定できません。設定した場合無視されます。 数値以外の文字列や範囲外の数値を指定した場合はメッセージを出力し，デフォルト値が設定されます。	0

（凡例）

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

(o)　webserver.errorpage から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.errorpage から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.errorpage.stack_trace.enabled	なし	例外発生時，デフォルトのエラーページにスタックトレースを出力するかどうかを指定します。	－

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(p)　webserver.http から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.http から始まるキーについて次に示します。

キー名称	差異	内容	デフォルト値
webserver.http.request.uri_decode.enabled	なし	サーブレットパスおよび追加のパス情報をデコードするかどうかを指定します。 true を指定した場合： 　デコードします。 false を指定した場合： 　デコードしません。	false

(q)　webserver.jsp から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.jsp から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.jsp.additional.import.list	なし	JSP コンパイル時に暗黙にインポートしたいクラス名（完全修飾名のクラス名または，「パッケージ名.*」）を指定します。複数のクラ	なし

キー名称	差異	内容	デフォルト値
webserver.jsp.additional.import.list	なし	ス名を指定する場合は，クラス名とクラス名の間をコンマ（,）で区切ります。存在しないクラス名や，クラスパスの通っていないクラス名など不正な値を指定した場合，メッセージを出力し，コンパイルエラーになります。	なし
webserver.jsp.check_interval	なし	JSP の更新検知インターバルを 0～2147483647 の範囲（単位：秒）の整数で指定します。 0 を指定した場合，更新の自動検知は行われません。	1
webserver.jsp.compile.backcompat	なし	JSP から生成されたサーブレットの java ソースのコンパイル時に，java ソースコードのバージョンを指定します。	−
webserver.jsp.keepgenerated	あり	JSP ファイルのコンパイルで生成された java ファイルを保持するかどうかを指定します。 次の値を指定します。 true を指定した場合： 　java ファイルを保持します。 false を指定した場合： 　java ファイルを保持しません。	−
webserver.jsp.recompilable	独自	JSP ファイルが更新されたときに自動的に再コンパイルするかどうかを指定します。 true を指定した場合： 　自動的に再コンパイルします。 false を指定した場合： 　自動的に再コンパイルしません。	false
webserver.jsp.translation.backcompat.customAction.declareVariable	なし	複数のカスタムタグで，スクリプト変数名とスクリプト変数のスコープが重複する場合，JSP ファイルから生成された Java コードで 2 回目のカスタムタグに対応するスクリプト変数の変数宣言を出力するかどうかを指定します。	−
webserver.jsp.translation.backcompat.taglib.noCheckPrefix	なし	taglib ディレクティブの前に，taglib ディレクティブで指定した prefix を使用したカスタムタグを記述しているかチェックするかどうかを指定します。	−
webserver.jsp.translation.backcompat.tag.noCheckRtexprvalue	なし	Expression が指定できないタグの属性値に Expression が指定されているかどうか検証するかどうかを指定します。	−
webserver.jsp.translation.backcompat.tag.rtexprvalueTerminate	なし	タグの属性値が，「"<%=」（または「'<%=」）で開始しており，「%>"」(「'<%」で開始した場合は「%>'」)で終了していない属性値の「"」(または「'」) で囲まれた値を文字列として扱うかどうかを指定します。	−
webserver.jsp.translation.backcompat.useBean.noCheckClass	なし	JSP トランスレーション時に<jsp:useBean>タグのクラス属性値のチェック処理を実行するかどうかを指定します。	−
webserver.jsp.update.interval	独自	JSP のリロード監視対象ファイルのコピーに必要な時間のインターバルを 0～2147483647 の範囲（単位：秒）の整数で指定します。 なお，インターバルは余裕のある値にしてください。JSP ファイルの更新を検知してから指定したインターバルのあとに，JSP の再コンパイルが実行されます。	0

キー名称	差異	内容	デフォルト値
webserver.jsp.update.interval	独自	JSP の再コンパイル機能が無効の場合，このプロパティは設定できません。設定した場合無視されます。 数値以外の文字列や範囲外の数値を指定した場合はメッセージを出力し，デフォルト値が設定されます。	0

（凡例）

あり：

Web コンテナサーバ用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(r)　webserver.servlet から始まるキー

webserver.servlet から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.servlet_api.exception.getCause.backcompat	なし	javax.servlet.ServletException のコンストラクタ ServletException（String, Throwable）および ServletException（Throwable）で指定した根本原因の例外を getCause()により取得できないようにします。 同様に javax.servlet.jsp.JspException のコンストラクタ JspException（String, Throwable）および JspException（Throwable）で指定した根本原因の例外を getCause()により取得できないようにします。 true を指定した場合： 根本原因の例外を取得できません。 false を指定した場合： 根本原因の例外を取得できます。	－

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(s)　webserver.session から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.session から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.session.persistence.enabled	独自	リロード時のセッション継続機能を使用するかどうかを指定します。 true を指定した場合： 　セッション継続機能は有効になります。 false を指定した場合： 　セッション継続機能は無効になります。 Web アプリケーションのリロード機能が無効の場合，このプロパティは設定できません。設定した場合無視されます。なお，このプロパティはサーブレットエンジンモードでだけ有効です。	false
webserver.session.server_id.enabled	なし	セッション ID にサーバ ID を付加するかどうかを指定します。	－
webserver.session.server_id.value	なし	セッション ID に付加するサーバ ID を指定します。	Web コンテナを実行するホストの IP アドレスと Web サーバとの通信に使用するポート番号から生成する文字列
webserver.session.work.directory	独自	セッション情報ファイルの出力先ディレクトリを絶対パスで指定します。 指定したディレクトリには Web アプリケーション単位にコンテキストルート名のディレクトリを作成し，作成したディレクトリ下にセッション情報ファイルを作成します。 作成したセッション情報ファイルは，リロード処理完了後に削除されます。webserver.session.work.clean=false が設定されている場合は削除されません。 このプロパティを設定する場合，サーバ単位で出力先が重複しないように注意してください。 リロード時のセッション継続機能が無効の場合，このプロパティは設定できません。なお，このプロパティはサーブレットエンジンモードでだけ有効です。	• Windows の場合 <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥work • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/work

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(t)　webserver.shutdown から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.shutdown から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.shutdown.port	独自	Web コンテナが終了用のメッセージを受信するために使用するポートを指定します。※ 1～65535 の整数で指定します。	8005

（凡例）

独自：

Web コンテナサーバ用 usrconf.properties ファイル固有のキーです。キーの詳細については，「内容」の列を参照してください。

注※

すでにほかのアプリケーションで使用，または確保されているポート番号は指定できません。また，複数の Web コンテナサーバで Web サーバとの通信に使用するポートのポート番号に同じ値を設定しないでください。同一のポート番号を指定した Web コンテナサーバは cjstartweb コマンドで複数起動できません。

(u)　webserver.static から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.static から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.static_content.cache.enabled	なし	静的コンテンツキャッシュ機能の有効／無効／強制的な無効を指定します。	－
webserver.static_content.cache.size	なし	静的コンテンツキャッシュ機能が有効な場合，メモリにキャッシュできるサイズ（単位：バイト）の上限を指定します。	－
webserver.static_content.cache.filesize.threshold	なし	静的コンテンツキャッシュ機能が有効な場合，キャッシュできるファイルサイズ（単位：バイト）を指定します。	－

（凡例）

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(v)　webserver.work から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.work から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.work.clean	なし	JSP 用テンポラリディレクトリ下に生成したディレクトリをサーバ起動時，および終了時に削除するかどうかを指定します。	－

キー名称	差異	内容	デフォルト値
webserver.work.directory	あり	JSP 用一時ディレクトリを指定します。絶対パスによるディレクトリ名を指定します。※	• Windows の場合 <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥work • UNIX の場合 /opt/Cosminexus/CC/web/containers/<サーバ名称>/work

（凡例）

あり：

Web コンテナサーバ用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

なし：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のキーです。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

注※

ディレクトリのパスの区切り記号には，"/"を使用してください。例えば，work ディレクトリとして C:¥temp¥work を設定する場合，次のように指定します。

```
webserver.work.directory=c:/temp/work
```

また，使用する OS によってファイル名長に制限があります。Windows では 250 バイト程度となっています。このため，ディレクトリ階層の深い場所でサーブレットや JSP を実行できません。特に JSP ファイルから生成されるファイル名は Web アプリケーションで配置されたディレクトリ階層の深さに比例して長くなり，JSP ファイルに対する Java ファイルを生成できないことがあります。これを回避するためには Web アプリケーションの配置ディレクトリの階層を浅くしたり，プロパティ webserver.work.directory に絶対パスで浅いディレクトリ階層を指定したりすることで回避してください。Application Server の作業ディレクトリについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1　J2EE サーバの作業ディレクトリ」を参照してください。

(w)　webserver.xml から始まるキー

Web コンテナサーバ用 usrconf.properties ファイルで指定できる webserver.xml から始まるキーを次に示します。

キー名称	差異	内容	デフォルト値
webserver.xml.validate	あり	Servlet 2.2，および Servlet 2.3 の Web アプリケーションについて，デプロイメント・ディスクリプタ（web.xml ファイル），およびタグライブラリ・ディスクリプタ（TLD ファイル）が，DTD で記述されたスキーマに沿って記述されているかチェックするかどうかを指定します。 次の値を指定します。 true を指定した場合： サーブレットエンジンモードのときは，Web アプリケーションのデプロイメント・ディスクリプタ（web.xml ファイル），およびタグライブラリ・ディスクリプタ（TLD ファイル）が DTD	－

キー名称	差異	内容	デフォルト値
webserver.xml.validate	あり	で記述されたスキーマに沿って書かれているかどうかをチェックします。 false を指定した場合： チェックしません。	－

（凡例）

あり：

Web コンテナサーバ用 usrconf.properties ファイルのキーと J2EE サーバ用 usrconf.properties ファイルのキーに差異があります。詳細については，「内容」の列を参照してください。

－：

Web コンテナサーバ用 usrconf.properties ファイルと J2EE サーバ用 usrconf.properties ファイルで共通のデフォルト値です。

(6)　記述例

```
ejbserver.logger.enabled.*=Error,Warning,Information,Debug
ejbserver.logger.channels.define.WebMessageLogFile.filenum=2
ejbserver.logger.channels.define.WebMessageLogFile.filesize=1048576

ejbserver.application.InitTermProcessClasses=InitTermClass1,InitTermClass2

webserver.connector.http.enabled=false
webserver.connector.http.port=8080
webserver.shutdown.port=8005
webserver.connector.ajp13.port=8007
webserver.connector.ajp13.max_threads=10
webserver.connector.ajp13.backlog=100
webserver.connector.ajp13.receive_timeout=600
webserver.work.directory=c:/work
```

3.11.4　web-users.xml（ユーザ認証情報定義ファイル）

(1)　形式

XML1.0 形式です。

(2)　ファイルの格納先

- Windows の場合

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

- UNIX の場合

  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/

(3)　機能

各ユーザのユーザ名，パスワード，および所属するロールを記述します。このファイルに不正な値を設定した場合，動作は保証されません。

(4)　指定できる属性

属性の指定形式

```
<web-users>
  <user name="ユーザ名" password="パスワード" roles="ロール名"/>
  <user name=・・・/>
</web-users>
```

属性の説明

name

ユーザのログイン名を記述します。

password

ユーザのパスワードを記述します。

roles

ユーザの所属するロールを記述します。コンマで区切ることで，複数のロールを定義できます。

(5)　記述例

```
<web-users>
  <user name="admin" password="admin" roles="admin"/>
</web-users>
```

3.11.5　web.policy（SecurityManager 定義ファイル）

(1)　形式

J2SE のプロパティファイル形式です。

(2)　ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥containers¥<サーバ名称>¥usrconf¥
- UNIX の場合
  /opt/Cosminexus/CC/web/containers/<サーバ名称>/usrconf/

(3)　機能

SecurityManager を設定します。SecurityManager 設定ファイルによって，Java 2 から導入されたセキュリティポリシーに基づく Web コンテナの実行時保護をします。例えば，次に示す現象を防止します。

- 内部で System.exit()などを発行するような，不正なサーブレット，JSP によって，Web コンテナ全体が勝手に終了する。
- システムプロパティなどを勝手に書き替えるような，不正なサーブレット，JSP によって，Web コンテナの実行に不具合が生じる。

なお，cjstartweb コマンドに，-security オプションを付けることで，Web コンテナ実行時保護機能が有効になります。また，このファイルに不正な値を設定した場合，動作は保証されません。

Web コンテナサーバの稼働中に，このファイルの内容を変更した場合，変更した内容は，次回 Web コンテナサーバを起動したときに反映されます。

(4)　記述例

使用されるポリシーファイルの内容を次に示します。

```
// (1)
// Grant all permissions to the java extensions
grant codeBase "file:${java.home}/lib/ext/-" {
permission java.security.AllPermission;
};

// (2)
```

```
// Grant all permissions to the java tools, etc
// Note: java.home is the jre, not the installation dir for the jdk
grant codeBase "file:${java.home}/../lib/*" {
permission java.security.AllPermission;
};


// (3)
// Grant all permissions to anything loaded from the
// EJB server itself
grant codeBase "file:${ejbserver.install.root}/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${tpbroker.java.home}/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/DABJ/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/manager/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/c4web/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/c4web/exlib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/jaxp/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/CTM/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/PRF/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/wss/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${cosminexus.home}/XMLSEC/lib/*" {
permission java.security.AllPermission;
};
grant codeBase "file:${hntrlib.home}/classes/*" {
grant codeBase "file:${program.files.home}/Hitachi/HNTRLib2/classes/*" {
permission java.security.AllPermission;
};

// (4)
// webapp policy
grant codeBase "file:${webserver.install.root}/containers/${webserver.serverName}
/webapps/-" {
permission java.lang.RuntimePermission "loadLibrary.*";
permission java.lang.RuntimePermission "queuePrintJob";
permission java.net.SocketPermission "*", "connect";
permission java.io.FilePermission "<<ALL FILES>>", "read, write";
permission java.util.PropertyPermission "*", "read";
};

// (5)
// Grant minimal permissions to everything else:
grant  {
permission java.util.PropertyPermission "*", "read";
permission java.lang.RuntimePermission "queuePrintJob";
permission java.net.SocketPermission "*", "connect";
};
```

記述例の（1）～（5）について説明します。

(1)

JDK の ext ディレクトリ下のクラスファイルに対して次の権限を許可します。

- すべてのアクセス権を許可

(2)

JDK の lib ディレクトリ下のクラスファイルに対して次の権限を許可します。

- すべてのアクセス権を許可

(3)

Web コンテナサーバが使用するクラスファイルに対して次の権限を許可します。

- すべてのアクセス権を許可

(4)

Web アプリケーションのクラスファイルに対して次の権限を許可します。

- すべてのライブラリのロードを許可
- 印刷ジョブ要求を許可
- すべてのソケット通信に対してネットワーク接続を許可
- すべてのファイルに対し読み取り，および書き込みを許可
- すべてのプロパティ情報の読み取りを許可

(5)

すべてのクラスファイルに対して次の権限を許可します。

- すべてのプロパティ情報の読み取りを許可
- 印刷ジョブ要求を許可
- すべてのソケット通信に対してネットワーク接続を許可

(5)　注意事項

サーバをセットアップするときに生成された web.policy ファイルには，Web コンテナサーバを動作させるために最低限必要な権限が記述されています。生成された web.policy ファイルの記述行の削除および変更はしないでください。

3.11.6　プール管理情報設定ファイル

プール管理情報設定ファイルは，次に示す DOCTYPE 宣言を持ちます。プール管理情報設定ファイルは，Web コンテナのコネクションプール機能を設定する場合に使用するファイルです。

```
<!DOCTYPE hitachi-webconpool-configuration PUBLIC '-//Hitachi, Ltd.//DTD webconpool Configuration 6.0//EN' 'file:///<Application Server のインストールディレクトリ>/CC/admin/dtds/hitachi-webconpool-configuration_6_0.dtd'>
```

! 注意事項

サンプルファイルで提供している属性ファイルなどをサーバ管理コマンド以外の方法で使用する場合，DOCTYPE 宣言内の URL の指定を URI に書き直す必要があります。

例

変更前

```
<!DOCTYPE hitachi-webconpool-configuration PUBLIC '-//Hitachi, Ltd.//DTD webconpool Configuration 6.0//EN' 'file:///C:/Program%20Files/Hitachi/Cosminexus/CC/admin/dtds/hitachi-webconpool-configuration_6_0.dtd'>
```

変更後

<!DOCTYPE hitachi-webconpool-configuration PUBLIC '-//Hitachi, Ltd.//DTD webconpool Configuration 6.0//EN' 'file:///<Application Server のインストールディレクトリ>/CC/admin/dtds/hitachi-webconpool-configuration_6_0.dtd'>

プール管理情報設定ファイルは，次に示すタグを持つ xml ファイルです。

タグ名	出現パターン	説明
<hitachi-webconpool-configuration>	1 回	ルートタグ。
┣ <DatabaseID>	1 回	DBMS 名。 一意の名称で指定します。 DBMS 名には，英数字，およびアンダースコア「_」を使用できます。
┣ <ClassName>	0 または 1 回	利用する JDBC 実装クラス名。
┣ <URL>	0 または 1 回	利用するデータベースにアクセスするためのデータベース URL。
┣ <UserCount>	0 または 1 回	プール機能を使用できるユーザ数。 0～2147483647 ユーザ数≧0 となるように指定します。 0 を指定するとプール機能を使用できるユーザ数に制限がなくなります。 省略した場合は 0 が設定されます。
┣ <UserID>	0 または 1 回	コネクション取得要求時に，省略された場合に使用されるユーザ ID。 省略した場合は空文字が設定されます。
┣ <Password>	0 または 1 回	コネクション取得要求時に，省略された場合に使用されるパスワード。 省略した場合は空文字が設定されます。
┣ <MinimumSize>	0 または 1 回	プールに割り当てるデータベースコネクションの最小数。 0～2147483647 省略した場合は 0 を設定します。

タグ名	出現パターン	説明
├ <MaximumSize>	0 または 1 回	プールで使用できるデータベースコネクションの最大数。 0～2147483647 0 を指定すると無制限にコネクションを使用できます。ただし，データベースで許容される最大コネクション数によって最大数は制限されます。 省略した場合は 0 が設定されます。
├ <RetryInterval>	0 または 1 回	プールからコネクションを取得するときに，コネクション取得のリトライをするまでのアプリケーションの待ち時間（単位：秒）。 0～2147483647 省略した場合は 0 が設定されます。
└ <RetryCount>	0 または 1 回	プールからコネクションを取得できなかった場合のリトライ回数。 0～2147483647 コネクションが取得できない場合に，リトライを行わないようにするには 0 を指定します。 省略した場合は 0 が設定されます。

cjwebeditpool コマンドでプール管理情報を登録・変更するときに使用するプール管理情報設定ファイルは，あらかじめエディタなどを使って作成したものを使用するか，次のディレクトリにあるサンプルファイルを使用します。サンプルファイルでは，デフォルトの設定内容が記述されています。

ディレクトリ	ファイル
• Windows の場合 <Application Server のインストールディレクトリ>¥CC¥examples¥webpoolconf • UNIX の場合 /opt/Cosminexus/CC/examples/webpoolconf	Webconpool.xml

4 基本・開発機能の互換機能 (DABroker Library を使用したデータベース接続)

この章では，基本・開発機能の互換機能のうち，DABroker Library を使用したデータベース接続について説明します。

4.1　この章の構成

この章では，基本・開発機能の互換機能のうち，DABroker Library を使用したデータベース接続について説明します。

この章の構成を次の表に示します。

表 4-1　この章の構成（DABroker Library を使用したデータベース接続）

分類	タイトル	参照先
解説	DABroker Library を使用したデータベース接続の概要	4.2
	DABroker Library の初期設定	4.3
	データベースとアプリケーション間の文字コードの変換	4.4
	受信バッファのプーリングの設定	4.5
	DABroker Library を使用する場合のトラブルシュート	4.6
	DABroker Library を使用したデータベース接続の実装	4.7
	DB Connector を使用したデータベースとの接続	4.8
	DB Connector for Reliable Messaging を使用したデータベース上のキューへの接続	4.9
実装	DABroker Library で使用する API の一覧	4.10
	Driver クラス	4.11
	Connection クラス	4.12
	Statement クラス	4.13
	PreparedStatement クラス	4.14
	CallableStatement クラス	4.15
	ResultSet クラス	4.16
	ResultSetMetaData クラス	4.17
	DatabaseMetaData クラス	4.18
	DataSource クラス	4.19
	Blob インタフェース	4.20
設定	DABroker Library で使用するコマンドの一覧	4.21
	dabsetup	4.22
	DABroker Library で使用するファイルの一覧	4.23
	DABroker Library 動作環境定義ファイル	4.24
	共通設定項目	4.25
	拡張データベースアクセストレース設定項目	4.26

分類	タイトル	参照先
設定	リモートアクセス設定項目	4.27
	for C++設定項目	4.28
	HiRDB の設定項目	4.29
	ORACLE の設定項目	4.30
	Connector 属性ファイルでの設定	4.31
注意事項	DABroker Library を使用する場合の注意事項	4.32

注　「運用」について，この章での説明はありません。

4.2　DABroker Library を使用したデータベース接続の概要

DABroker Library は，データベース連携機能を提供する構成ソフトウェアです。JDBC インタフェースをサポートしたデータベースにアクセスするライブラリを提供します。これによって，複数の DBMS に一つのインタフェースでアクセスできる統一的なデータベースアクセスを実現できます。

4.2.1　DABroker Library を使用してデータベースに接続する方法

アプリケーションサーバで DABroker Library を使用してデータベースに接続する方法には，次の 2 種類があります。

- DABroker Library の API を使用して実装したアプリケーションによってデータベースに接続する方法
- アプリケーションサーバが提供するリソースアダプタ（DB Connector または DABroker Library）を使用してデータベースに接続する方法

この章の構成と，データベースへの接続方法ごとに参照する節について，次の表に示します。

表 4-2　データベースへの接続方法ごとに参照する節

タイトル	データベースへの接続方法		参照先
	API を使用して実装したアプリケーションを使用	リソースアダプタを使用	
DABroker Library を使用したデータベース接続の概要	○	○	4.2※
Cosminexus DABroker Library の初期設定	○	○	4.3
データベースとアプリケーション間の文字コードの変換	○	○	4.4
受信バッファのプーリングの設定	○	○	4.5
DABroker Library を使用する場合のトラブルシュート	○	○	4.6
DABroker Library を使用したデータベース接続の実装	○	－	4.7
DB Connector を使用したデータベースとの接続	－	○	4.8
DB Connector for Reliable Messaging を使用したデータベース上のキューへの接続	－	○	4.9
DABroker Library を使用する場合の注意事項	○	○	4.32

（凡例）○：参照する　－：参照しない

注※　この節です。

4.2.2　接続できるデータベース

DABroker Library を JDBC ドライバとして使用して J2EE サーバから接続できるデータベース，および接続に使用するインタフェースの対応を，次の表に示します。なお，DABroker Library で提供する JDBC ドライバは，JDBC2.1 の規格に準拠しています。

表 4-3　J2EE サーバから接続できるデータベース

データベース	JDBC インタフェース		JMS インタフェースおよび JDBC インタフェース	
	ローカルトランザクション	グローバルトランザクション	ローカルトランザクション	グローバルトランザクション
HiRDB Version 8	○	○	○	○
HiRDB Version 7	○	○	○	○
XDM/RD E2※	○	－	－	－
Oracle 10g R2 10.2.0	○	○	－	－
Oracle 10g 10.1.0	○	○	－	－
Oracle9i R2 9.2.0	○	○	－	－
Oracle9i 9.0.1	○	－	－	－

（凡例）○：使用できます。　－：使用できません。

注※　Database Connection Server 経由で接続します。HiRDB クライアントが前提です。

！ 注意事項

Oracle9i 9.2.0 を使用する場合は，Oracle9i 9.2.0.4 以降を使用してください。

4.2.3　DABroker Library のメモリ所要量

ここでは，DABroker Library のメモリ所要量の計算式について説明します。

DABroker Library のメモリ所要量の計算式を次に示します。

メモリ所要量＝128＋データベース同時接続数
×〔386＋A×｛20＋B＋定義列数×2＋C×2＋D×E＋F×（G＋H）｝〕＋I

（凡例）

A：並列に使用しているカーソル数

並列に使用しているカーソル数は，一つの Connection オブジェクトから派生して，同時に使用する Statement インスタンス数，PreparedStatement インスタンス数，CallableStatement インスタンス数，ResultSet インスタンス数の総和です。

B：bufSize の値

JDBC を使用してデータベースに接続するときに指定する bufSize の値です。

C：SQL のサイズ

SQL のサイズは，KB（キロバイト）単位に切り上げて算出してください。

D：?パラメタのサイズ

?パラメタのサイズは，一つの SQL 文に指定する，?パラメタのデータ長の総和です。?パラメタに指定する列属性ごとのデータ長を次の表に示します。

表 4-4　パラメタ長一覧（ネイティブライブラリ内）

データベース種別	列属性	データ長（バイト）	備考
HiRDB	INTEGER	12	−
	SMALLINT	8	−
	LARGE DECIMAL(m, n)	↓（M/2）↓＋7	• M は指定した値の精度を表します。 • 4 バイト単位で切り上げます。
	FLOAT または DOUBLE PRECISION	16	−
	SMALLFLT または REAL	12	−
	CHAR(n)，MCHAR(n)，NCHAR(n)，VARCHAR(n)，MVARCHAR(n)，NVARCHAR(n)	指定した実データ長+8	4 バイト単位で切り上げます。
	DATE	16	−
	TIME	16	−
	TIMESTAMP(p)	p=0，p=2 の場合：28 p=4，p=6 の場合：32	−
	BINARY(n)または BLOB(n)	指定した実データ長＋ 4	−
ORACLE，Oracle8i	FLOAT	56	−
	DATE	64	−
	その他のデータ型	指定した実データ長＋ 54	4 バイト単位で切り上げます。

E：配列更新の配列数

配列更新の配列数は，HiRDB の「配列を使用した INSERT 機能」，「配列を使用した UPDATE 機能」，または「配列を使用した DELETE 機能」を使用する場合は addBatch メソッドの実行回数，そのほかの場合は 1 となります。

F：検索データのサイズ

検索データのサイズは，検索する列のデータ長の総和（1 行分）です。

ただし，接続するデータベースが Oracle であり，CLOB，NCLOB，BLOB，BFILE，LONG LAW，LONG のどれかの列を一つ以上含む検索の場合，検索データのサイズはほかの列に関係なく，bufSize の値となります。

列属性ごとのデータ長を次の表に示します。

表 4-5　検索する列のデータ長一覧（ネイティブライブラリ内）

データベース種別	列属性		データ長（バイト）	備考
HiRDB	INTEGER		4	−
	SMALLINT		2	−
	LARGE DECIMAL(m, n)		m + 2	−
	FLOAT または DOUBLE PRECISION		8	−
	SMALLFLT または REAL		4	−
	CHAR(n)または MCHAR(n)		n + 1	−
	NCHAR(n)		n×2 + 1	−
	VARCHAR(n)または MVARCHAR(n)		n + 2	−
	NVARCHAR(n)		n×2 + 2	−
	DATE		11	−
	TIME		9	−
	TIMESTAMP(p)		• p=0 の場合：20 • p=0 以外の場合：21 + p	−
	BINARY(n)または BLOB(n)	HiRDB の位置付け子機能を使用する場合	12	−
		HiRDB の位置付け子機能を使用しない場合	（n と bufSize の値を比較してより小さい方の値）+ 8	bufSize の値とは，JDBC を使用してデータベースに接続するときに指定する bufSize の値です。
ORACLE, Oracle8i	NUMBER		44	−
	VARCHAR2(n)，CHAR(n)，VARCHAR2(n)または NCHAR(n)		n + 1	−
	MLSLABEL(n)または RAW(n)		n	−
	ROWID		9	−
	FLOAT		8	−
	DATE		20	−

G：FETCH サイズ

FETCH サイズは，Statement，または ResultSet クラスの setFetchSize メソッドの指定値です。ただし，指定値が 0，または指定を省略した場合は，次のようになります。

- 指定値が 0 または指定を省略した場合の FETCH サイズ= 64KB/検索データのサイズ

  計算の結果 0 になる場合は，1 に繰り上げられます。

H：データベース用データ領域

データベース用データ領域は，接続データベースが HiRDB の場合は 0，そのほかのデータベースの場合は 1 となります。

I：JDBC のメモリサイズ

JDBC のメモリサイズは，次の値です。

- JDBC のメモリサイズ＝（Connection オブジェクトサイズ
  ＋ Statement オブジェクトサイズ
  ＋ ResultSet オブジェクトサイズ
  ＋ PreparedStatement オブジェクトサイズ
  ＋ CallableStatement オブジェクトサイズ
  ＋?パラメタの実データサイズ
  ＋検索データのトータルサイズ
  ＋ SQL サイズ
  ＋ bufSize の値によるデータサイズ
  ＋トレースサイズ）/1,024

JDBC のメモリサイズの計算式に使用するデータのサイズについて，(1)から(10)で説明します。

(1)　Connection オブジェクトサイズ

Connection オブジェクトサイズ（単位バイト）の計算式を次に示します。

Connectionオブジェクトサイズ＝1223500※1＋（80364※1＋bufSizeの値※2×1024）×Connectionインスタンス数※3

注

Connection オブジェクトサイズとして，次のどちらかのタイミングで「bufSize に指定した値 × 生成したコネクション数分のメモリ」を Java ヒープ領域に確保します。

- コネクションプールのウォーミングアップ機能使用時
- 初回 getConnection()発行時

bufSize に必要以上に大きな値を設定すると，Connection オブジェクトサイズ分のメモリを確保したときに Java ヒープを使い切り，OutOfMemoryError が発生するおそれがあります。

HiRDB の BLOB 型データや BINARY 型データへアクセスする場合は，bufSize に大きな値を指定しないで，HiRDB の位置付け子機能を使用することを検討してください。HiRDB の位置付け子機能については，マニュアル「HiRDB 解説」を参照してください。

注※1

オブジェクトを構成する大きさであり，実行するプラットフォームおよび JDK のバージョンにより若干値が異なる場合があります。

注※2

bufSize のデフォルト値は 64 です。

注※3

Connection インスタンス数とは，同時接続数のことです。

(2)　Statement オブジェクトサイズ

Statement オブジェクトサイズ（単位バイト）の計算式を次に示します。

Statementオブジェクトサイズ＝1000※1＋1400※1×Statementインスタンス数※2

注※1

オブジェクトを構成する大きさであり，実行するプラットフォームおよび JDK のバージョンにより若干値が異なる場合があります。

注※2

Statement インスタンス数とは，システム全体で同時に使用する Statement 数です。システム全体で Statement オブジェクトを使用しないときは 0 になります。

(3)　ResultSet オブジェクトサイズ

ResultSet オブジェクトサイズは，接続先のデータベースが Oracle8i 以降で，BLOB，CLOB の列属性がある表を検索した場合と，それ以外の処理を実行した場合で異なります。それぞれの場合の，ResultSet オブジェクトサイズ（単位バイト）の計算式を次に示します。

1. 接続先のデータベースが Oracle8i 以降で，BLOB，CLOB の列属性がある表を検索した場合

   ResultSetオブジェクトサイズ＝

   2400※1＋｛3400※1＋MIN（ROW_SIZEの値※2×1024×1024，BLOBおよびCLOBの実データ長）×2｝×ResultSetインスタンス数※3

2. 1.以外の処理を実行した場合

   ResultSetオブジェクトサイズ＝2400※1＋3400※1×ResultSetインスタンス数※3

   注※1

   オブジェクトを構成する大きさであり，実行するプラットフォームおよび JDK のバージョンにより若干値が異なる場合があります。

   注※2

   デフォルト値は 16 です。

   注※3

   ResultSet インスタンス数とは，システム全体で同時に使用する ResultSet 数です。システム全体で，検索 SQL を実行しない，または ResultSet オブジェクトを生成しないときは 0 になります。

(4)　PreparedStatement オブジェクトサイズ

PreparedStatement オブジェクトサイズ（単位バイト）の計算式を次に示します。

PreparedStatementオブジェクトサイズ＝

79900※1＋4700※1×PreparedStatementインスタンス数※2

注※1

オブジェクトを構成する大きさであり，実行するプラットフォームおよび JDK のバージョンにより若干値が異なる場合があります。

注※2

PreparedStatement インスタンス数とは，全体で同時に使用する PreparedStatement 数です。システム全体で PreparedStatement オブジェクトを使用しないときは 0 になります。

(5)　CallableStatement オブジェクトサイズ

CallableStatement オブジェクトサイズ（単位バイト）の計算式を次に示します。

CallableStatementオブジェクトサイズ＝

65000※1＋4700※1×CallableStatementインスタンス数※2

注※1

オブジェクトを構成する大きさであり，実行するプラットフォームおよび JDK のバージョンにより若干値が異なる場合があります。

注※2

CallableStatement インスタンス数とは，全体で同時に使用する CallableStatement 数です。システム全体で CallableStatement オブジェクトを使用しないときは 0 になります。

(6)　?パラメタの実データサイズ

?パラメタの実データサイズ（単位バイト）の計算式を次に示します。

?パラメタの実データサイズ＝

9000※1＋180※1×?パラメタ数※2＋?パラメタのトータル長※2

注※1

オブジェクトを構成する大きさであり，実行するプラットフォームおよび JDK のバージョンにより若干値が異なる場合があります。

注※2

?パラメタ数と?パラメタのトータル長は，システム全体で同時に指定する，?パラメタ数と?パラメタのトータル長です。?パラメタのトータル長は，実データの長さの総和です。?パラメタに指定する列属性ごとの実データ長を次の表に示します。

表 4-6　パラメタ長一覧（JDBC 内）

データベース種別	列属性	データ長（バイト）
HiRDB	INTEGER	4
	SMALLINT	2
	LARGE DECIMAL(m, n)	↓（m/2）↓+1
	FLOAT または DOUBLE PRECISION	8
	SMALLFLT または REAL	4
	上記以外	指定した実データ長
ORACLE，Oracle8i	NUMBER	指定した実データを数値文字列で表現したときの実際の長さ
	FLOAT	8
	上記以外	指定した実データ長

(7)　検索データのトータルサイズ

検索データのトータルサイズ（単位バイト）は，実際のレコード長＋ 4×（取得する列数＋ 1）を 1 件のレコード長とし，検索した件数の総和になります。実際のレコード長は列属性ごとの実際のデータ長の総和です。

なお，検索データのトータルサイズは，ResultSet のタイプがスクロール可能のときだけ必要です。それ以外は 0 になります。

列属性ごとの実際のデータ長を次の表に示します。

表 4-7　検索する列のデータ長一覧（JDBC 内）

データベース種別	列属性	データ長（バイト）
HiRDB	INTEGER	4
	SMALLINT	2
	LARGE DECIMAL(m, n)	↓(m/2)↓+1
	FLOAT または DOUBLE PRECISION	8
	SMALLFLT または REAL	4
	上記以外	データベースに格納している実データ長
ORACLE，Oracle8i	NUMBER	データベースに格納している実データを数値文字列で表現したときの実際の長さ
	FLOAT	8
	上記以外	データベースに格納している実データ長

(8)　SQL サイズ

実際の SQL のサイズが bufSize の値×1024 バイト以上のときだけ必要です。それ以外は 0 になります。なお，bufSize のデフォルト値は 64 です。

(9)　bufSize の値によるデータサイズ

bufSize の値によるデータサイズ（単位バイト）の計算式を次に示します。なお，計算式は，受信バッファプール数指定機能の使用の有無によって異なります。

- 受信バッファプール数指定機能を使用しない場合

  データサイズ＝bufSizeの値※×1024
  ×（同一コネクション内に存在する，検索SQL以外を実行するStatementインスタンス数
  ＋同一コネクション内に存在する，検索SQL以外を実行するPreparedStatementインスタンス数
  ＋同一コネクション内に存在する，CallableStatementインスタンス数
  ＋同一コネクション内に存在する，ResultSetインスタンス数）

- 受信バッファプール数指定機能を使用する場合

  データサイズ＝bufSizeの値※×1024
  ×（同一コネクション内で同時に使用する，検索SQL以外を実行するStatementインスタンス数
  ＋同一コネクション内で同時に使用する，検索SQL以外を実行するPreparedStatementインスタンス数
  ＋同一コネクション内で同時に使用する，CallableStatementインスタンス数
  ＋同一コネクション内で同時に存在する，ResultSetインスタンス数）

  注※　bufSize のデフォルト値は 64 です。

(10)　トレースサイズ

トレースサイズの計算式（単位バイト）を次に示します。

トレースサイズ＝2048×メモリ情報取得量

メモリ情報取得量とは，環境変数 DAB_ONMEMNUM の値です。デフォルト値は 1000 です。

4.3　DABroker Library の初期設定

ここでは，DABroker Library を使用する場合の初期設定について，OS ごとに説明します。

なお，以降，この章では，DABroker Library のインストールディレクトリのことを，**DABroker Library 運用ディレクトリ**といいます。

初期設定では，インストールおよびインストール後の環境設定を実施します。DABroker Library を使用する場合のインストール後の環境設定の流れを次に示します。

1. DABroker Library のセットアップ（UNIX の場合）
2. DABroker Library の動作環境の設定
3. データベース接続のための設定
4. J2EE アプリケーション運用のための設定

なお，これらの設定は，次の環境構築が済んでいることを前提にしています。

- DBMS を使用できる環境が構築されていること
- 開発した J2EE アプリケーションを組み込むホスト，および DABroker Library を組み込むホストをネットワーク上で使用できる環境が構築されていること

設定後，DABroker Library を使用したプロセス（J2EE サーバなど）が起動しているときには，設定を変更できません。該当するプロセスを停止してから，設定を変更してください。

4.3.1　DABroker Library のインストールおよび動作環境の設定（Windows の場合）

ここでは，Windows で DABroker Library を使用するためのインストールおよび動作環境の設定について説明します。

(1)　インストール

製品のインストールには，インストーラを使用します。

なお，Windows の場合には，すべての構成製品のインストールが終わると，再起動を促すダイアログが表示されます。ただし，DABroker Library の設定後にも再起動が必要になるため，DABroker Library の設定が終了したあとに再起動することをお勧めします。

DABroker Library をインストールしたあとは，システム環境変数 Path に<DABroker Library インストールディレクトリ>¥lib を設定し，システムを再起動する必要があります。なお，システム環境変数の設定，およびシステムの再起動を実施しないと，各ユティリティが使用できません。必ず実施してください。

(2)　インストール後の DABroker Library のディレクトリ構成

Windows の場合のインストール後の DABroker Library のディレクトリ構成を次の表に示します。これらのディレクトリは，<アプリケーションサーバのインストールディレクトリ>下に作成されます。

表 4–8　DABroker Library のディレクトリ構成（Windows の場合）

ディレクトリ		説明
DAB	−	DABroker Library のインストールディレクトリです。

ディレクトリ			説明
DAB	bin		ヘルプファイル格納用ディレクトリです。
	lib		ライブラリ格納用ディレクトリです。
	msg		メッセージファイル格納用ディレクトリです。
	spool	－	トレースファイル，ログファイル格納用ディレクトリです。
		dabcltrc	－
		dabdump	－
		db_access	－
	tmp		作業ディレクトリです。
DABJ	－		DABroker Library の JDBC ドライバのインストールディレクトリです。

（凡例）－：該当しません。

(3)　DABroker Library の動作環境の設定（Windows の場合）

DABroker Library の動作環境は，DABroker Library の環境設定ユティリティで設定します。

環境設定ユティリティにはオンラインヘルプがありますので，設定内容の詳細については，オンラインヘルプを参照してください。［スタート］メニューの［プログラム］から［Cosminexus※］－［DABrokerLibrary］－［ヘルプ］を選択すると，DABroker Library のオンラインヘルプが表示されます。

注※
　プログラムフォルダ名を変更している場合，任意のプログラムフォルダから，選択してください。

また，環境設定ユティリティでは，トレース取得の設定ができます。DABroker Library で取得できるトレースについては，「4.6.5　DABroker Library で取得できるトレース」を参照してください。

ここでは，DABroker Library の動作環境の設定について説明します。

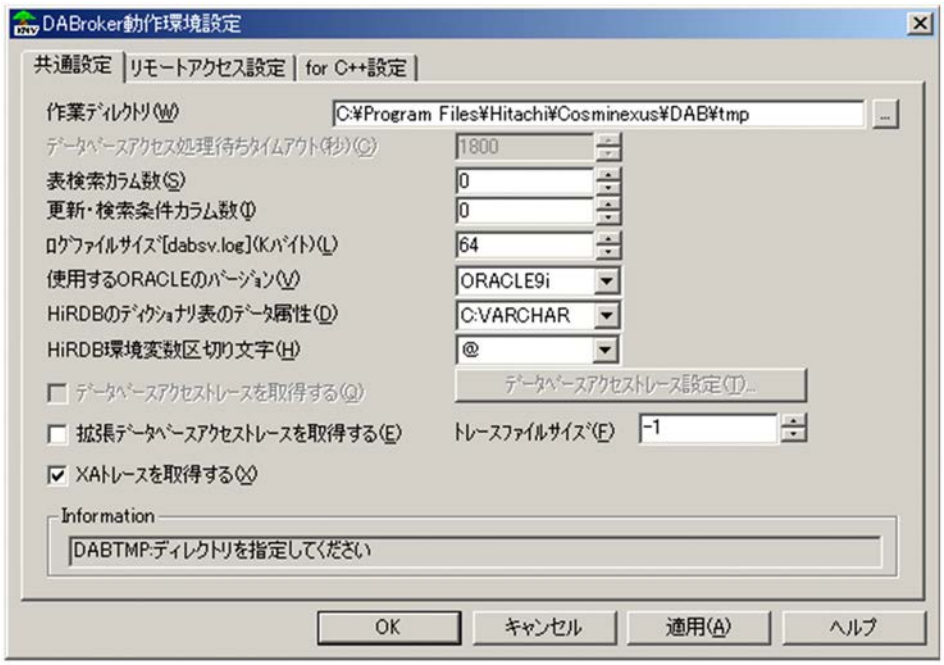
！ 注意事項

DABroker Library の動作環境で設定する各ディレクトリには，ネットワークドライブを指定しないでください。

(a)　DABroker Library の動作環境の設定手順

DABroker Library の動作環境の設定手順を次に示します。

1. **環境変数を設定します。**
   システム環境変数の PATH に「<アプリケーションサーバのインストールディレクトリ>¥DAB¥lib」を設定してください。設定されていない場合，環境設定ユティリティは起動できません。
2. **［スタート］メニューの［プログラム］から［Cosminexus］－［DABrokerLibrary］－［環境設定］を選択します。**
   ［DABroker 動作環境設定］ダイアログが表示されます。



3. [DABroker 動作環境設定] ダイアログで必要事項を設定して，[適用] ボタンをクリックします。

   [DABroker 動作環境設定] ダイアログには次の三つのタブがあります。

   - [共通設定] タブ
   - [リモートアクセス設定] タブ
   - [for C++設定] タブ

   各タブの設定内容については，以降で説明します。

4. [DABroker 動作環境設定] ダイアログでの必要事項の設定が終わったら，[OK] ボタンをクリックしてダイアログを閉じます。
5. システムを再起動します。

(b)　[共通設定] タブの設定内容

ここでは，[共通設定] タブの設定内容について説明します。DABroker Library を使用する場合の [共通設定] タブの設定内容を次に示します。

1. 「作業ディレクトリ」に，作業ディレクトリ（例：「C:¥Program Files¥Hitachi¥Cosminexus¥DAB¥tmp」）が表示されていることを確認します。
2. 「使用する ORACLE のバージョン」に，Oracle のバージョン（例：ORACLE9i）を指定します。
3. 拡張データベースアクセストレースを取得する場合は，[拡張データベースアクセストレースを取得する] チェックボックスをチェックします。
4. 拡張データベースアクセストレースを取得する場合は，[データベースアクセストレース設定] ボタンをクリックします。

   [アクセストレース設定] ダイアログが表示されます。

5. [アクセストレース設定] ダイアログで，取得しない情報の項目のチェックを外して，[OK] ボタンをクリックします。

   [DABroker 動作環境設定] ダイアログに戻ります。

6. XA トレースを取得する場合は，[XA トレースを取得する] チェックボックスをチェックします。

7. [適用] ボタンをクリックします。

(c) ［リモートアクセス設定］タブの設定内容

ここでは，［リモートアクセス設定］タブの設定内容について説明します。DABroker Library を使用する場合の［リモートアクセス設定］タブの設定内容を次に示します。

1. 次の項目を設定します。
   - HiRDB の BLOB 型データ受取バッファサイズ
   - ORACLE の LONG 型データ受取バッファサイズ
   - 漢字コードセット
2. [適用] ボタンをクリックします。

(d) ［for C++設定］タブの設定内容

ここでは，［for C++設定］タブの設定内容について説明します。DABroker Library がインストールされている環境で，DABroker for C++をインストールし，DABroker for C++を使用してデータベースにアクセスする場合の設定です。

［for C++設定］タブでは，DABroker for C++を使用して開発した J2EE アプリケーションからのアクセスでエラーが発生した場合の，DABroker for C++のイベントトレースの取得について設定します。エラーが発生した場合にイベントトレースを取得しておくことで，トレースを参照してエラーに対処できるようになります。

DABroker Library を使用する場合の［for C++設定］タブの設定内容を次に示します。

1. イベントトレースの取得の有無，取得する場合のトレース取得サイズやトレース取得レベルを設定します。
2. 取得するトレースの種別を設定します。
3. [適用] ボタンをクリックします。

DABroker for C++のイベントトレースの詳細については，マニュアル「DABroker for C++」を参照してください。

(e) 環境設定ユティリティとファイルとの対応

環境設定ユティリティで設定した内容は，DABroker Library 動作環境定義ファイルに反映されます。環境設定ユティリティと DABroker Library 動作環境定義ファイルの対応を次の表に示します。

表 4-9　環境設定ユティリティと DABroker Library 動作環境定義ファイルの対応

環境設定ユティリティ		DABroker Library 動作環境定義ファイルの項目名
タブ名	項目名	
［共通設定］タブ	作業ディレクトリ	DABTMP
	データベースアクセス処理終了待ちタイムアウト	DABDBA_TIMEOUT
	表検索カラム数	DABSELECTCOLCNT
	更新・検索条件カラム数	DABSELECTINFCNT

環境設定ユティリティ		DABroker Library 動作環境定義ファイルの項目名
タブ名	項目名	
［共通設定］タブ	ログファイルサイズ［dabsv.log］	DABSVLOGSIZE
	使用する ORACLE のバージョン	DABORACLE_TYPE
	HiRDB のディクショナリ表のデータ属性	DABSQL_HiRDB_DICTIONARY_DATATYPE
	HiRDB 環境変数区切り文字	DAB_HIRDB_DBINF_ESC
	拡張データベースアクセストレースを取得する	DABEXSQL_TRC_LINE
	トレースファイルサイズ	
	XA トレースを取得する	DABXA_LOG_ERROR
［リモートアクセス設定］タブ	HiRDB の BLOB 型データ受取バッファサイズ	DABHIRDBA_BLOBBUFSIZE
	ORACLE の LONG 型データ受取バッファサイズ	DABORADBA_LONGBUFSIZE
	漢字コードセット	DABLANGMODE
［for C++設定］タブ	DABroker for C++のトレース取得サイズ	DABCPP_EVTTRC_SIZE
	DABroker for C++のトレース取得レベル	DABCPP_EVTTRC_LEVEL
	DABroker for C++のトレース取得種別	DABCPP_EVTTRC_nn

環境設定ユティリティの［アクセストレース設定］ダイアログの項目と DABroker Library 動作環境定義ファイルの対応を次の表に示します。

表 4-10 ［アクセストレース設定］ダイアログの項目と DABroker Library 動作環境定義ファイルの対応

項目名（アクセストレースの種類）	DABroker Library 動作環境定義ファイルの項目名
CANCEL	DABSQL_TRC_CANCEL
CLOSE	DABSQL_TRC_CLOSE
COMMIT	DABSQL_TRC_COMMIT
CONNECT	DABSQL_TRC_CONNECT
DECLARE	DABSQL_TRC_DECLARE
DESCRIBE	DABSQL_TRC_DESCRIBE
DISCONNECT	DABSQL_TRC_DISCONNECT
EXECUTE	DABSQL_TRC_EXECUTE
FETCH	DABSQL_TRC_FETCH
LOGGING	DABSQL_TRC_LOGGING

項目名（アクセストレースの種類）	DABroker Library 動作環境定義ファイルの項目名
LOGLESS	DABSQL_TRC_LOGLESS
PREPARE	DABSQL_TRC_PREPARE
ROLLBACK	DABSQL_TRC_ROLLBACK
OPEN	DABSQL_TRC_OPEN
AUTO_COMMIT_OFF	DABSQL_TRC_AUTO_COMMIT_OFF
AUTO_COMMIT_ON	DABSQL_TRC_AUTO_COMMIT_ON
ATTACH	DABSQL_TRC_ATTACH
ATTRGET	DABSQL_TRC_ATTRGET
ATTRSET	DABSQL_TRC_ATTRSET
BIND	DABSQL_TRC_BIND
CHARFORM	DABSQL_TRC_CHARFORM
CHARID	DABSQL_TRC_CHARID
DEFINE	DABSQL_TRC_DEFINE
DESALLOC	DABSQL_TRC_DESALLOC
DESCANY	DABSQL_TRC_DESCANY
DESFREE	DABSQL_TRC_DESFREE
DETACH	DABSQL_TRC_DETACH
ENVCREATE	DABSQL_TRC_ENVCREATE
ERRGET	DABSQL_TRC_ERRGET
GETPIECE	DABSQL_TRC_GETPIECE
HDLALLOC	DABSQL_TRC_HDLALLOC
HDLFREE	DABSQL_TRC_HDLFREE
LOBCLOSE	DABSQL_TRC_LOBCLOSE
LOBLEN	DABSQL_TRC_LOBLEN
LOBOPEN	DABSQL_TRC_LOBOPEN
LOBREAD	DABSQL_TRC_LOBREAD
LOBWRITE	DABSQL_TRC_LOBWRITE
PARAMGET	DABSQL_TRC_PARAMGET
SESBEGIN	DABSQL_TRC_SESBEGIN
SESEND	DABSQL_TRC_SESEND
SETPIECE	DABSQL_TRC_SETPIECE

項目名（アクセストレースの種類）	DABroker Library 動作環境定義ファイルの項目名
GETDIAG	DABSQL_TRC_GETDIAG
GET_INFO	DABSQL_TRC_GET_INFO
SET_INFO	DABSQL_TRC_SET_INFO
EXECUTE_DIRECT	DABSQL_TRC_EXECUTE_DIRECT

DABroker Library 動作環境定義ファイルについては，「4.24　DABroker Library 動作環境定義ファイル」を参照してください。

(4)　データベース接続のための設定

DABroker Library がインストールされている環境で，DABroker for C++をインストールし，DABroker for C++を使用してデータベースに接続する場合には，接続先データベースの定義が必要です。

DABroker for C++を使用してデータベースに接続する場合は，接続先データベース定義ユティリティで設定したデータベース種別名，およびデータベース名を使用してデータベースにアクセスします。接続先データベース定義ユティリティで設定した情報を使用すると，接続条件などが変更されてもアプリケーションを変更する必要がありません。ただし，OTS 機能，または JTS 機能を使用する場合は，接続先データベース定義ユティリティで設定した情報を使用できません。

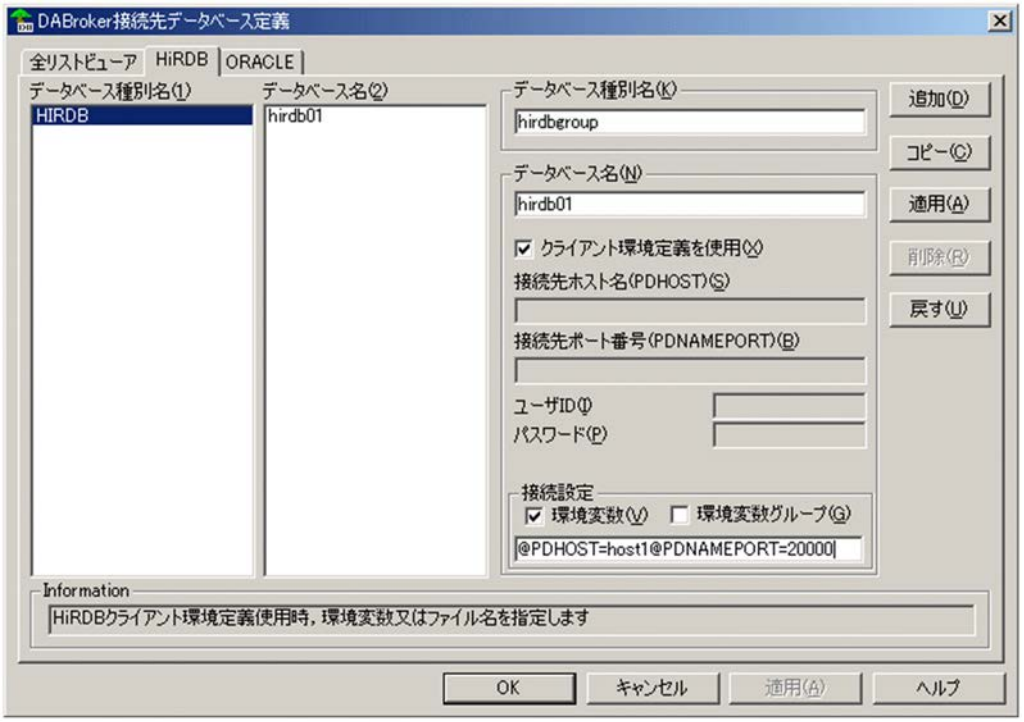
(a)　接続先データベース定義の設定

接続先データベース定義ユティリティは，プログラムフォルダに登録された「DABroker 接続先データベース定義」から起動します。「DABroker 接続先データベース定義」は，すべてのユーザが操作できます。

接続先データベース定義ユティリティで設定した内容は，接続先データベース定義ファイル（<DABroker Library 運用ディレクトリ>¥conf¥dabenv）に出力されます。なお，DABroker Library 運用ディレクトリは，DABroker Library のインストールディレクトリ（<アプリケーションサーバのインストールディレクトリ>¥DAB）です。

接続先データベース定義の設定手順を次に示します。

1. プログラムフォルダに登録された［接続先データベース定義］を起動します。
   ［DABroker 接続先データベース定義］ダイアログが表示されます。



2. 使用するデータベースの製品名が記載されたタブを選択し，接続情報を設定します。

次の情報を定義します。

- データベース種別名

  使用するデータベースに対して付ける任意の名称です。

- データベース名

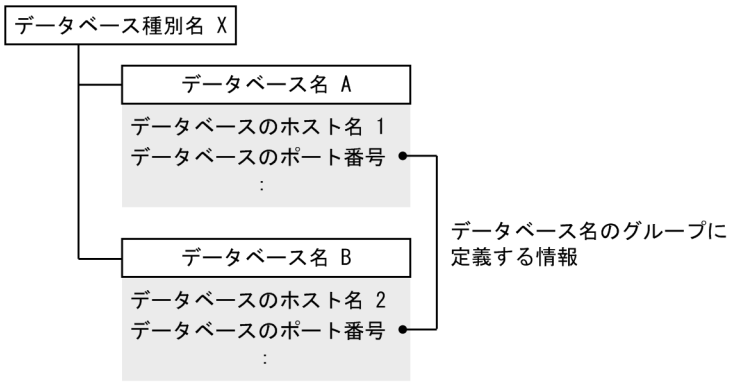
  データベースごとに異なる情報の集まりに対して付ける任意の名称です。

- データベース名のグループに定義する情報

  データベース名のグループに定義する情報は，データベースごとに異なります。詳細については，「(b)　データベース名のグループに定義する情報」を参照してください。

3. [接続先データベース定義] ダイアログでの必要事項の設定が終わったら，[OK] ボタンをクリックしてダイアログを閉じます。

なお，[接続先データベース定義] ダイアログで設定する，データベース種別名，データベース名，およびデータベース名のグループに定義する情報は，次のような親子関係になります。



この例で示すように，データベース名は，同一データベース種別名に対して，複数定義できます。例えば，ホスト名が異なる二つの HiRDB を，一つのデータベース種別として定義し，アプリケーションでは，データベース名でホストの異なる HiRDB を使い分けられます。このような場合の［接続先データベース定義］ダイアログでの設定方法を次に示します。

- **データベース種別名，データベース名を新規登録する場合**
  設定後，［追加］ボタンを選択します。
- **同一データベース種別名に対して，別のデータベース名を登録する場合**
  設定後，［追加］ボタンを選択します。
- **登録されている情報を変更する場合**
  登録内容を変更したあと，［追加］ボタンを選択します。
- **データベース種別名またはデータベース名を削除する場合**
  削除するデータベース種別またはデータベース名を選択して［削除］ボタンを選択してください。

(b)　データベース名のグループに定義する情報

データベース名のグループに定義する情報について，データベースごとに説明します。

● HiRDB または XDM/RD E2 を使用する場合

HiRDB または XDM/RD E2 を使用する場合に，データベース名のグループに定義する情報を次の表に示します。

表 4-11　データベース名のグループに定義する情報（HiRDB または XDM/RD E2 を使用する場合）

設定項目名	指定内容
データベース種別名	使用するデータベースに対して付ける任意の名称を指定します。設定規則を次に示します。 • 接続先データベース定義ユティリティ内でユニークな名称にしてください。 • 23 文字まで指定できます。2 バイト文字，半角カタカナ文字は使用しないでください。
データベース名	データベースごとに異なる情報の集まりに対して付ける任意の名称を指定します。設定規則を次に示します。 • 同一データベース種別名内でユニークな名称にしてください。 • 23 文字まで指定できます。2 バイト文字，半角カタカナ文字は使用しないでください。
接続先ホスト名（DBHOST）	HiRDB のホスト名を指定します。
接続先ポート番号（DBNAME）	ポート番号を指定します。
ユーザ ID（USERID）	接続ユーザ ID を指定します。※
パスワード（PASSWD）	パスワードを指定します。※
環境変数	HiRDB クライアントへ渡す環境変数を指定します。設定規則を次に示します。 • 1,024 文字まで指定できます。2 バイト文字，および半角カタカナ文字は使用しないでください。 • 各環境変数の先頭に，必ず区切り文字（@）を記述してください。

設定項目名	指定内容
環境変数	• PDHOST（ホスト名称，または IP アドレス），および PDNAMEPORT（接続するポート番号）は，必ず先頭か末尾で指定してください。 • PDHOST，および PDNAMEPORT は，重複して指定しないでください。PDHOST，および PDNAMEPORT 以外の環境変数は，DABroker Library で設定した値をそのまま使えます。 • DABroker Library では，指定した環境変数の妥当性はチェックしません。
環境変数グループ	HiRDB クライアントへ渡す環境変数グループ名を指定します。設定規則を次に示します。 • 30 文字まで指定できます。2 バイト文字，および半角カタカナ文字は使用しないでください。 • DABroker Library では，指定した環境変数グループが有効かどうかはチェックしません。

注※

J2EE アプリケーションでユーザ ID とパスワードを指定していない場合は，ここで指定したユーザ ID，パスワードを使用してデータベースにアクセスします。DABroker Library では，接続先データベース定義ファイルの出力時にパスワードを暗号化していないため，パスワードは J2EE アプリケーション側で指定するようにしてください。

なお，接続先データベース定義ユティリティで複数定義したデータベース種別名がユニークでも，実際のデータベース種別が同じ場合，同じプログラムから同時に接続できません。

● Oracle を使用する場合

Oracle を使用する場合に，データベース名のグループに定義する情報を次の表に示します。

表 4-12　データベース名のグループに定義する情報（Oracle を使用する場合）

設定項目名	指定内容
データベース種別名	使用するデータベースに対して付ける任意の名称を指定します。設定規則を次に示します。 • 接続先データベース定義ユティリティ内でユニークな名称にしてください。 • 23 文字まで指定できます。2 バイト文字，半角カタカナ文字は使用しないでください。
データベース名	データベースごとに異なる情報の集まりに対して付ける任意の名称を指定します。設定規則を次に示します。 • 同一データベース種別名内でユニークな名称にしてください。 • 23 文字まで指定できます。2 バイト文字，半角カタカナ文字は使用しないでください。
リスナー名（DBNAME）	SQL*NET 経由でアクセスする Oracle のリスナー名称を指定します。ただし，リモートアクセスしない場合は省略します。
ユーザ ID（USERID）	接続ユーザ ID を指定します。※
パスワード（PASSWD）	パスワードを指定します。※

注※

J2EE アプリケーションでユーザ ID とパスワードを指定していない場合は，ここで指定したユーザ ID，パスワードを使用してデータベースにアクセスします。DABroker Library では，接続先データベース定義ファイルの出力時にパスワードを暗号化していないため，パスワードは J2EE アプリケーション側で指定するようにしてください。

なお，接続先データベース定義ユティリティで複数定義したデータベース種別名がユニークでも，実際のデータベース種別が同じ場合，同じプログラムから同時に接続できません。

(5)　J2EE アプリケーション運用のための設定

J2EE アプリケーションを運用する前に，次の設定をしてください。

- J2EE アプリケーションの組み込み
- J2EE アプリケーションのインタフェースの定義
- サーバで動作する J2EE アプリケーションの動作環境の設定

(a)　J2EE アプリケーションの組み込み

DABroker Library が組み込まれたサーバに J2EE アプリケーションを格納します。また，次に示すクラスパスを指定します。

- <アプリケーションサーバのインストールディレクトリ>¥DABJ¥JdbcDbpsv.jar
- <アプリケーションサーバのインストールディレクトリ>¥DABJ¥JdbcDbpsvEx.jar※

  注※

  DABroker Library で JDBC2.0 拡張機能を使用する場合に，このパスを指定してください。

ここでは，例として，「<アプリケーションサーバのインストールディレクトリ>¥DABJ¥JdbcDbpsvEx.jar」をクラスパスとして指定します。

簡易構築定義ファイルでは，物理ティアの定義（<tier>タグの定義）の J2EE サーバの定義で，<configuration>タグにクラスパスの設定を追加します。なお，指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6.2　簡易構築定義ファイルの指定内容」を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<model-definition xmlns="http://www.cosminexus.com/mngsvr/schema/ModelDefinition-2.5">
  <!-- Webシステムの属性定義 -->
  <web-system>
    <name>MyWebSystem</name>
      :
    <!-- 物理ティア(combined-tier)の定義 -->
    <tier>
      <tier-type>combined-tier</tier-type>
      <!-- J2EEアプリケーションの組み込みの設定 -->
      <configuration>
        <logical-server-type>j2ee-server</logical-server-type>
        <param>
          <param-name>add.class.path</param-name>
          <param-value><アプリケーションサーバのインストールディレクトリ>¥DABJ¥JdbcDbpsvEx.jar</param-value>
        </param>
      </configuration>
    </tier>
      :
  </web-system>
  <host>
    :
</model-definition>
```

注　太字の部分が編集部分です。

(b)　J2EE アプリケーションのインタフェースの定義

J2EE アプリケーションのインタフェースを定義します。

(c)　サーバで動作する J2EE アプリケーションの動作環境の設定

DABroker Library では，JNI を使って DABroker の提供するネイティブライブラリを使用します。DABroker Library をサーバで動作する Java アプリケーションとして動作させるときは，次の設定が必要です。

- ネイティブライブラリの動作環境の設定
- データベースクライアントライブラリの動作環境の設定

● ネイティブライブラリの動作環境の設定

ネイティブライブラリは JDBC ドライバを実行する（呼び出す）プロセス上で読み込まれて動作するので，ネイティブライブラリ実行のための動作環境を設定する必要があります。

設定内容

システム環境変数の PATH に「<DABroker Library 運用ディレクトリ>¥lib」を設定します。

● データベースクライアントライブラリの動作環境の設定

ネイティブライブラリは，データベースにアクセスする場合に，各データベースが提供するクライアントライブラリを呼び出します。データベースクライアントライブラリの動作環境の設定を次に示します。

1. データベースライブラリのパスを設定します。

   特別な設定は必要ありません。ただし，接続するデータベースによっては動作環境の設定が必要な場合があります。詳細については，接続するデータベースのマニュアルなどを参照してください。

   PATH に，次の値を設定します。

   - HiRDB，または XDM/RD E2 の場合：HiRDB クライアントライブラリのパス名
   - Oracle の場合：Oracle ライブラリ（libclntsh）のパス名

2. データベースの提供する環境変数を設定します。

   JDBC ドライバを実行する（呼び出す）プロセス上で有効な環境変数として，各データベースの提供する環境変数を設定する必要があります。

   詳細については，「4.3.3　データベースの設定」，接続するデータベースのマニュアルなどを参照してください。

4.3.2　DABroker Library のインストールおよび動作環境の設定（UNIX の場合）

ここでは，UNIX で DABroker Library を使用するためのインストールおよび動作環境の設定について説明します。

(1)　インストール

製品のインストールには，インストーラを使用します。

使用する Java VM によって，異なった環境変数を設定する必要があることがあります。設定する項目については，各 Java VM のドキュメントを参照してください。

(2)　インストール後の DABroker Library のディレクトリ構成

UNIX の場合のインストール後の DABroker Library のディレクトリ構成を次の表に示します。

表 4–13　DABroker Library のディレクトリ構成（UNIX の場合）

ディレクトリ		説明
/opt/Cosminexus	DABJ	DABroker Library の JDBC ドライバのインストールディレクトリです。
/opt/DABroker	−	DABroker Library のインストールディレクトリです。
	bin	実行コマンド格納用ディレクトリです。
	lib	ライブラリ格納用ディレクトリです。
	obj	オブジェクトファイル格納用ディレクトリです。
	AGT	エージェント処理用ディレクトリです。
	AGTGRP	グループ管理用ディレクトリです。
	msg	メッセージファイル格納用ディレクトリです。
	sample	サンプルファイル格納用ディレクトリです。
	conf	定義ファイル格納用ディレクトリです。
	spool	トレースファイル，ログファイル格納用ディレクトリです。

（凡例）−：該当しません。

! 注意事項

DABroker Library をアンインストールする場合，/opt/Cosminexus/DABJ は削除されないので，手動で削除してください。

(3)　DABroker Library のセットアップ

セットアップでは，DABroker Library を管理するユーザを OS に登録し，DABroker Library の運用ディレクトリを作成します。セットアップの手順を次に示します。

1. **DABroker Library を管理するユーザ（UNIX ログインユーザ）を OS に登録します。**

   ユーザ ID 登録後，パスワードも登録してください。登録方法は OS のマニュアルを参照してください。ここで登録したユーザを，**DABroker Library の管理ユーザ**と呼びます。DABroker Library の管理ユーザに与えられる権限は，次のとおりです。

   - DABroker Library で提供するファイルおよびディレクトリの所有者としてのアクセス権
   - DABroker Library の運用コマンドの実行権

   注　Oracle にアクセスする場合は，登録するユーザの UID を「3」以上にしてください。

2. **DABroker Library のインストールディレクトリのグループおよびユーザを，手順 1.で登録した管理グループおよびユーザに変更します。**

3. **DABroker Library セットアップコマンド（dabsetup）を実行します。**

   スーパーユーザで，/opt/DABroker/bin ディレクトリにある dabsetup を実行します。

   (例)　dabsetup /home

   dabsetup コマンドは，DABroker Library 運用ディレクトリを指定するコマンドです。このコマンドを実行すると，インストールディレクトリのディレクトリおよびファイルが，<パス名>に指定した運用ディレクトリに移動されます。<パス名>を省略した場合，運用ディレクトリは DABroker Library

のインストールディレクトリ（/opt/DABroker）となります。また，dabsetup コマンドに-d オプションを指定すると，運用ディレクトリの削除もできます。

! 注意事項

AIX の場合で，Component Container を C ロケールで動作させるときは，環境変数「LC__FASTMSG」に「false」が指定されているか確認してください。DABroker Library を使用して DB にアクセスし，かつ Component Container を C ロケールで動作させる場合には，環境変数「LC__FASTMSG」に「false」を指定する必要があります。

(4)　DABroker Library の動作環境の設定（UNIX の場合）

DABroker Library の動作環境は，DABroker Library 動作環境定義ファイルで設定します。また，DABroker Library 動作環境定義ファイルでは，トレース取得の設定ができます。DABroker Library で取得できるトレースについては，「4.6.5　DABroker Library で取得できるトレース」を参照してください。

ここでは，DABroker Library の動作環境の設定について説明します。

DABroker Library の動作環境は，DABroker Library 動作環境定義ファイルで設定します。動作環境定義ファイルは，インストール時に提供されるサンプルを使用して作成します。

- 提供されるサンプルファイル
  /opt/DABroker/sample/dasysconf
- 動作環境定義ファイルの格納先
  <DABroker Library 運用ディレクトリ>/conf

DABroker Library の動作環境の設定手順を次に示します。

1. **サンプルファイルを動作環境定義ファイルの格納先へコピーします。**
2. **コピーしたサンプルファイルを編集します。**
   必要に応じて，標準値の変更や値の追加をしてください。
   - 共通設定項目（作業ディレクトリ，Oracle のバージョン，拡張データベースアクセストレースの取得の有無など）
   - リモートアクセス設定項目（データの受け取りバッファサイズなど）
   - HiRDB 設定項目
   - Oracle 設定項目

   なお，DABroker Library 動作環境定義ファイルは，「DAB」ブロックと「OTHERS」ブロックで構成されます。

   「DAB」ブロック

   共通設定項目やリモートアクセス設定項目を指定するブロックです。DABroker Library はこのブロックの設定項目をチェックするため，不正な名称などを指定していた場合はエラーになります。

   「OTHERS」ブロック

   HiRDB 設定項目や Oracle 設定項目を指定するブロックです。DABroker Library はこのブロックの設定項目をチェックしません。指定したすべての項目が有効となります。

   DABroker Library 動作環境定義ファイルについては，「4.24　DABroker Library 動作環境定義ファイル」を参照してください。
3. **ファイルを保存します。**

なお，作成済みの DABroker Library 動作環境定義ファイルの内容を変更する場合は，J2EE サーバを停止してください。

(5)　データベース接続のための設定

ここでは，DABroker Library を使用してデータベースに接続するために必要な設定について説明します。

- HiRDB の言語モードの設定
- DABroker Library および JDBC での言語モードの設定

(a)　HiRDB の言語モードの設定

データベースとして HiRDB を使用する場合は，言語モードを設定してください。DABroker Library の言語モードと HiRDB の言語モードの対応を次の表に示します。なお，DABroker Library の言語モードとは，DABroker Library 動作環境定義ファイルの DAB_LANG（LANG 環境変数）のことです。

表 4-14　DABroker Library の言語モードと HiRDB の言語モードの対応

DABroker Library の言語モード			HiRDB の言語モード			
OS	言語モード	DAB_LANG の指定値	SJIS	EUC 日本語	ASCII	UTF-8
AIX	SJIS	Ja_JP	○	−	−	−
		Ja_JP.IBM-932	○	−	−	−
		Ja_JP.IBM-943	○	−	−	−
	EUC	ja_JP	−	○	−	−
		ja_JP.IBM-eucJP	−	○	−	−
	ASCII	C※	−	−	○	−
	UTF-8	UTF-8	−	−	−	○
HP-UX	SJIS	ja_JP.SJIS※	○	−	−	−
	EUC	ja_JP.EUC	−	○	−	−
		ja_JP.UJIS	−	○	−	−
		ja_JP.eucJP	−	○	−	−
	ASCII	C	−	−	○	−
	UTF-8	UTF-8	−	−	−	○
Linux	EUC	ja_JP	−	○	−	−
		ja_JP.ujis	−	○	−	−
		japanese	−	○	−	−
		japanese.euc	−	○	−	−
		ja_JP.eucjp	−	○	−	−
		Ja_JP	−	○	−	−

DABroker Library の言語モード			HiRDB の言語モード			
OS	言語モード	DAB_LANG の指定値	SJIS	EUC 日本語	ASCII	UTF-8
Linux	ASCII	C※	−	−	○	−
	UTF-8	UTF-8	−	−	−	○

（凡例）　○：使用できる　−：使用できない

注※　DAB_LANG の標準値を示します。

(b)　DABroker Library および JDBC での言語モードの設定

ENCODELANG，または setEncodLang に指定する言語モードは，DABroker Library の言語モードと合わせる必要があります。ENCODELANG，または setEncodLang の指定方法について次に説明します。

OS の言語モードと DABroker Library の言語モードが同じ場合

ENCODELANG，または setEncodLang の指定は必要ありません。

OS の言語モードと DABroker Library の言語モードが異なる場合

ENCODELANG，または setEncodLang に，DABroker Library の言語モードと同等の値を指定する必要があります。

注意事項

- DABroker Library の言語モードと，データベースの言語モードは必ず合わせてください。
- データベースの言語モードを DABroker Library がサポートしていない場合，DABroker Library の言語モードには ASCII モードを設定し，ENCODELANG または setEncodLang にデータベースの言語モードを指定してください。

(6)　J2EE アプリケーション運用のための設定

J2EE アプリケーションを運用する前に，次の設定をしてください。

- J2EE アプリケーションの組み込み
- J2EE アプリケーションのインタフェースの定義
- サーバで動作する J2EE アプリケーションの動作環境の設定

(a)　J2EE アプリケーションの組み込み

DABroker Library が組み込まれたサーバに J2EE アプリケーションを格納します。また，環境変数「CLASSPATH」に次に示すパスを追加します。

- /opt/Cosminexus/DABJ/JdbcDbpsv.jar
- /opt/Cosminexus/DABJ/JdbcDbpsvEx.jar※

  注※

  DABroker Library で JDBC2.0 拡張機能を使用する場合に，このパスを指定してください。

(b)　J2EE アプリケーションのインタフェースの定義

J2EE アプリケーションのインタフェースを定義します。

(c)　サーバで動作する J2EE アプリケーションの動作環境の設定

DABroker Library では，JNI を使って DABroker の提供するネイティブライブラリを使用します。DABroker Library をサーバで動作する Java アプリケーションとして動作させるときは，次の設定が必要です。

- ネイティブライブラリの動作環境の設定
- データベースクライアントライブラリの動作環境の設定

● ネイティブライブラリの動作環境の設定

ネイティブライブラリは JDBC ドライバを実行する（呼び出す）プロセス上で読み込まれて動作するので，ネイティブライブラリ実行のための動作環境を設定する必要があります。

設定内容

ライブラリ検索パスを指定する環境変数に，「<DABroker Library 運用ディレクトリ>/lib」を設定します。設定する環境変数は OS によって異なります。

- LIBPATH（AIX の場合）
- LD_LIBRARY_PATH（HP-UX および Linux の場合）

● データベースクライアントライブラリの動作環境の設定

ネイティブライブラリは，データベースにアクセスする場合に，各データベースが提供するクライアントライブラリを呼び出します。データベースクライアントライブラリの動作環境の設定を次に示します。

1. データベースライブラリのパスを設定します。

   特別な設定は必要ありません。ただし，接続するデータベースによっては動作環境の設定が必要な場合があります。詳細については，接続するデータベースのマニュアルなどを参照してください。

   ライブラリ検索パスを指定する環境変数に，次の値を指定します。設定する環境変数は OS によって異なります。

   - LIBPATH（AIX の場合）
   - LD_LIBRARY_PATH（HP-UX および Linux の場合）

   使用している OS の環境変数に次のパスを設定してください。

   - HiRDB，または XDM/RD E2 の場合：HiRDB クライアントライブラリのパス名
   - Oracle の場合：Oracle ライブラリ（libclntsh）のパス名

2. データベースの提供する環境変数を設定します。

   JDBC ドライバを実行する（呼び出す）プロセス上で有効な環境変数として，各データベースの提供する環境変数を設定する必要があります。

   詳細については，「4.3.3　データベースの設定」，接続するデータベースのマニュアルなどを参照してください。

4.3.3　データベースの設定

ここでは，DABroker Library を使用してデータベースに接続する場合に必要な設定について説明します。

DABroker Library を使用する場合に接続できるデータベースと，必要な設定についての参照先を次に示します。

表 4-15　DABroker Library を使用する場合に接続できるデータベースと，必要な設定についての参照先

使用できるデータベース	必要な設定についての参照先
HiRDB	(1)
Oracle	(2)
XDM/RD E2	(3)

(1)　HiRDB の設定

DABroker Library を使用して HiRDB に接続する場合の設定について説明します。なお，データベースとして HiRDB を使用する場合は，HiRDB をインストールして初期設定を済ませておいてください。詳細については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

ここでは，DABroker Library を使用して，HiRDB で J2EE リソースを扱う場合に必要な次の設定について説明します。

- 環境変数の設定
- 環境変数のグループ登録
- タイムアウトの設定

なお，同一の環境変数を環境変数グループと環境変数の両方に設定した場合は，環境変数グループで設定した環境変数が優先されます。

(a)　環境変数の設定

次の環境変数を設定する必要があります。環境変数の詳細については，マニュアル「HiRDB UAP 開発ガイド」を参照してください。

PATH（Windows の場合）

HiRDB のクライアントライブラリとして，次の値を指定します。通常，HiRDB のインストーラによってシステム環境変数に設定されます。

<HiRDB のホームディレクトリ>¥client¥lib

LIBPATH（AIX の場合）

HiRDB のクライアントライブラリとして，次の値を指定します。

<HiRDB のホームディレクトリ>/client/lib

LD_LIBRARY_PATH（HP-UX および Linux の場合）

HiRDB のクライアントライブラリとして，次の値を指定します。

<HiRDB のホームディレクトリ>/client/lib

PDXAMODE※1

「1」を指定します。

PDTXACANUM※1

（HiRDB にアクセスする J2EE コンポーネントの同時実行数＋ 1）×（J2EE コンポーネントでアクセスする可能性がある HiRDB 数）を指定します。

PDHOST※2

データベースのホスト名または IP アドレスを指定します。

PDUSER※2

データベースのユーザ名およびデータベースのパスワードを指定します。

PDNAMEPORT※2

データベースのポート番号を指定します。

PDSWAITTIME※2

リクエスト間隔タイムアウトを指定します。

PDCWAITTIME※2

レスポンスタイムアウトを指定します。

PDSWATCHTIME※2

「0」を指定します。コネクションプーリング機能を使用する場合は，「0」を指定してください。

注※1

グローバルトランザクションを使用する場合に設定します。この環境変数は，HiRDB のクライアント環境変数グループには登録できません。

注※2

これらの環境変数は，HiRDB のクライアント環境変数グループとして登録することもできます。登録方法については，「(b)　環境変数のグループ登録」で説明します。

(b)　環境変数のグループ登録

環境変数のグループは，次の方法で登録します。

- Windows の場合

  HiRDB のクライアント環境変数登録ツールで「システムグループ」を選択して，環境変数のグループを登録します。この場合，DB Connector のプロパティ定義では，ここで登録したグループ名を指定してください。

- UNIX の場合

  HiRDB のクライアント環境変数グループの設定ファイルに登録します。この場合，DB Connector のプロパティ定義では，ここで登録した環境変数グループの設定ファイルのパスを指定してください。

なお，環境変数のグループ登録の詳細については，マニュアル「HiRDB UAP 開発ガイド」を参照してください。

(c)　タイムアウトの設定

HiRDB のタイムアウトは，次に示す HiRDB のクライアント環境変数に設定します。

- PDSWAITTIME
- PDCWAITTIME
- PDCONNECTWAITTIME
- PDNBLOCKWAITTIME

HiRDB のクライアント環境変数の設定については，「(a)　環境変数の設定」を参照してください。

タイムアウトの設定については，マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6.6 データベースでのタイムアウトを設定する」，マニュアル「HiRDB システム定義」，およびマニュアル「HiRDB UAP 開発ガイド」を参照してください。

(2)　Oracle の設定

DABroker Library を使用して Oracle に接続する場合の設定について説明します。なお，データベースとして Oracle を使用する場合は，Oracle をインストールして初期設定を済ませておいてください。詳細については，Oracle のマニュアルを参照してください。

! 注意事項

Oracle9i 以降は Oracle8i の機能範囲内でサポートしています。Oracle9i 以降でサポートされた新たな固有の機能は使用できません。

ここでは，DABroker Library を使用して，Oracle で J2EE リソースを扱う場合に必要な次の設定について説明します。

- 環境変数の設定
- トランザクションリカバリを行うための権限の設定
- タイムアウトの設定

(a)　環境変数の設定

次の環境変数を設定する必要があります。

PATH（Windows の場合）

Oracle のクライアントライブラリとして，次の値を指定します。通常，Oracle のインストーラによってシステム環境変数に設定されます。

<Oracle のホームディレクトリ>¥bin

LIBPATH（AIX の場合）

Oracle のクライアントライブラリとして，次の値を指定します。

<Oracle のホームディレクトリ>/lib32

LD_LIBRARY_PATH（HP-UX および Linux の場合）

Oracle のクライアントライブラリとして，次の値を指定します。

<Oracle のホームディレクトリ>/lib32

NLS_LANG（UNIX の場合）

Oracle のクライアントで使用する言語を設定します。

設定値の例

```
japanese_japan.ja16sjis
Japanese_Japan.UTF8
```

(b)　トランザクションリカバリを行うための権限の設定

Oracle への接続でグローバルトランザクションを使用する場合には，Oracle の接続ユーザにトランザクションリカバリを行うための権限を付与する必要があります。

付与する権限

- SYS.DBA_PENDING_TRANSACTIONS の SELECT 権限
- FORCE ANY TRANSACTION 権限

権限の付与の方法

次のどちらかの方法で権限を付与します。

- Oracle Enterprise Manager コンソールを使用する

- sqlplus を使用する

次に，Oracle 9.2.0 で権限を付与する場合の例を示します。なお，権限を付与する方法の詳細については，Oracle のマニュアルを参照してください。

- Oracle Enterprise Manager コンソールを使用する場合の例

  Oracle Enterprise Manager コンソールを使用して権限を付与する場合の手順を次に示します。

  1. sys ユーザでデータベースに接続します。
  2. [セキュリティ] − [ユーザー] − [<権限を付与するユーザ名>] − [オブジェクト権限] を開きます。
  3. [SYS] − [ビュー] − [DBA_PENDING_TRANSACTIONS] を選択して，SELECT 権限を付与します。

- sqlplus を使用する場合の例

  sqlplus を使用して権限を付与する場合の手順を次に示します。

  1. sys ユーザでデータベースに接続します。
  2. 次の SQL 文を発行します。

     grant select on DBA_PENDING_TRANSACTIONS to <権限を付与するユーザ名>;

     commit;

(c)　タイムアウトの設定

必要に応じて，Oracle のタイムアウトを設定してください。タイムアウトは，Oracle のサーバ定義の DISTRIBUTED_LOCK_TIMEOUT パラメタに設定します。

なお，このほか，XAOpenString の SesTm パラメタの設定がタイムアウトに影響します。このパラメタは，チューニングできません。

詳細については，マニュアル「アプリケーションサーバ システム設計ガイド」の「8.6　タイムアウトを設定する」またはマニュアル「アプリケーションサーバ システム設計ガイド」の「9.3　タイムアウトを設定する」，および Oracle のマニュアルを参照してください。

(d)　動的パフォーマンスビューの参照権の設定

Oracle に接続するユーザは，動的パフォーマンスビューの参照権を持っている必要があります。次のどちらかの方法で Oracle に接続するユーザに動的パフォーマンスビューの参照権を設定してください。

- 「GRANT SELECT_CATALOG_ROLE TO <ユーザ名>;」を実行する。
- 「GRANT SELECT ON V_$INSTANCE TO <ユーザ名>;」，「GRANT SELECT ON V_$PROCESS TO <ユーザ名>;」，および「GRANT SELECT ON V_$SESSION TO <ユーザ名>;」を実行する。

(e)　OS 認証を使用する場合の設定（Windows の場合）

DABroker Library のサービスは，Windows のローカルシステムアカウントを使用しています。そのため，OS 認証をするユーザが Windows のローカルシステムアカウントでない場合は，DABroker Library のサービスアカウントを使用する OS 認証に変更してください。なお，OS 認証を使用できるのは，Java を使用してデータベースアクセスする場合だけです。

(3)　XDM/RD E2 の設定

DABroker Library を使用して XDM/RD E2 に接続する場合の設定は，HiRDB Type4 JDBC Driver を使用して接続する場合の設定と同じです。HiRDB Type4 JDBC Driver を使用して XDM/RD E2 に接続する場合の設定については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 L XDM/RD E2 の設定」を参照してください。

4.4　データベースとアプリケーション間の文字コードの変換

アプリケーションからデータベースに文字データを格納するときには，Unicode からデータベースに設定されている文字コードに変換して格納する必要があります。文字コード変換は JDBC ドライバによって行われますが，次に示す内容に従って，あらかじめ JDBC ドライバに使用する文字コードを設定しておく必要があります。

ここでは，データベースへの接続方法ごとの文字コードの設定方法について説明します。

(1)　API を使用して実装したアプリケーションでデータベースに接続する場合

JDBC コネクションを取得するとき，ENCODELANG キーに該当値を設定したプロパティを作成し，DABroker Library の java.sql.Driver.connect メソッドまたは java.sql.DriverManager.getConnection メソッドの引数に渡してください。

(2)　DB Connector を使用してデータベースに接続する場合

Connector 属性ファイルの encodLang プロパティで指定します。

ベーシックモードでは，ejbserver.jdbc.propertyInfo.ENCODELANG プロパティに使用する文字コードを指定してください。このプロパティに値が指定されている場合，J2EE サーバが JDBC コネクションを取得するときに，ENCODELANG キーに該当値を設定したプロパティを作成し，DABroker Library の java.sql.Driver.connect の引数に渡します。

4.5　受信バッファのプーリングの設定

DABroker Library を使用する場合，実行した SQL に対して返却される実行結果を格納する領域をプールして再利用できます。この領域を受信バッファといいます。また，受信バッファをプールするための領域を受信バッファプールといいます。受信バッファをプールすることによって，SQL 実行時に受信バッファが頻繁に作成，破棄されることで発生する負荷を削減できます。

また，プールする受信バッファの数を指定できるため，ステートメントプーリング使用時のメモリ占有量を抑制できます。

ここでは，受信バッファの管理と，受信バッファプール数の指定について説明します。

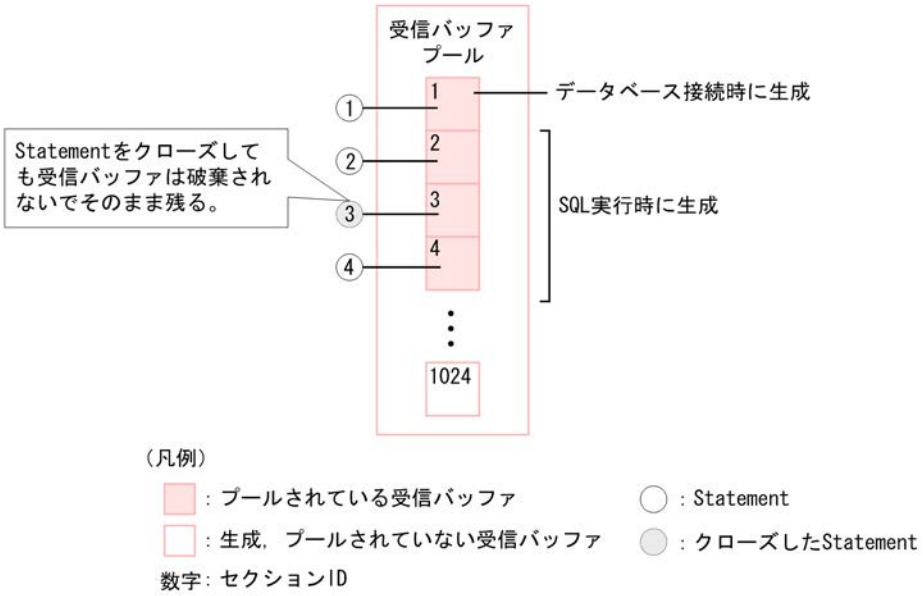
4.5.1　受信バッファの管理

受信バッファの管理では，プールする受信バッファ数を指定する場合と指定しない場合とで，受信バッファの再利用のしかたが異なります。ここでは，プールする受信バッファ数を指定しない，デフォルトの場合について説明します。受信バッファ数を指定する場合については，「4.5.2　受信バッファプール数の指定」を参照してください。

受信バッファは，Statement ごとに受信バッファプールに生成されます。生成された受信バッファはセクション ID で管理され，同一の Statement で再利用されます。なお，ここでは，Statement，PreparedStatement，および CallableStatement を Statement と総称します。

受信バッファプールの管理の概要を次の図に示します。

図 4-1　受信バッファプールの管理の概要（受信バッファプール数を指定しない場合）



受信バッファが生成されるタイミング

Statement の受信バッファはデータベースとの接続時に一つ生成され，そのセクション ID は「1」になります。これ以降は SQL 実行時に受信バッファが生成されます。受信バッファが生成されると，受信バッファプールに登録されます。

受信バッファの保持期間

受信バッファプールに登録された受信バッファは，コネクションがクローズされるまで保持されます。ステートメントをクローズしても，受信バッファは破棄されません。

プールする受信バッファの数

プールできる受信バッファの数は，最大 1,024 個です。

SQL を実行するメソッド

次のメソッドを実行すると SQL が実行され，受信バッファが生成されます。

表 4–16　4SQL を実行するメソッド

クラス	メソッド
Connection	prepareCall(String sql)
	prepareCall(String sql, int resultSetType, int resultSetConcurrency)
	prepareStatement(String sql)
	prepareStatement(String sql, int resultSetType, int resultSetConcurrency)
DatabaseMetaData	getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
	getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
	getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
	getSchemas()
	getTableTypes()
	getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
	getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)
	getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
	getPrimaryKeys(String catalog, String schema, String table)
	getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)
Statement	executeBatch()
	execute(String sql)
	executeQuery(String sql)
	executeUpdate(String sql)
PreparedStatement	execute()
	executeQuery()
	executeUpdate()

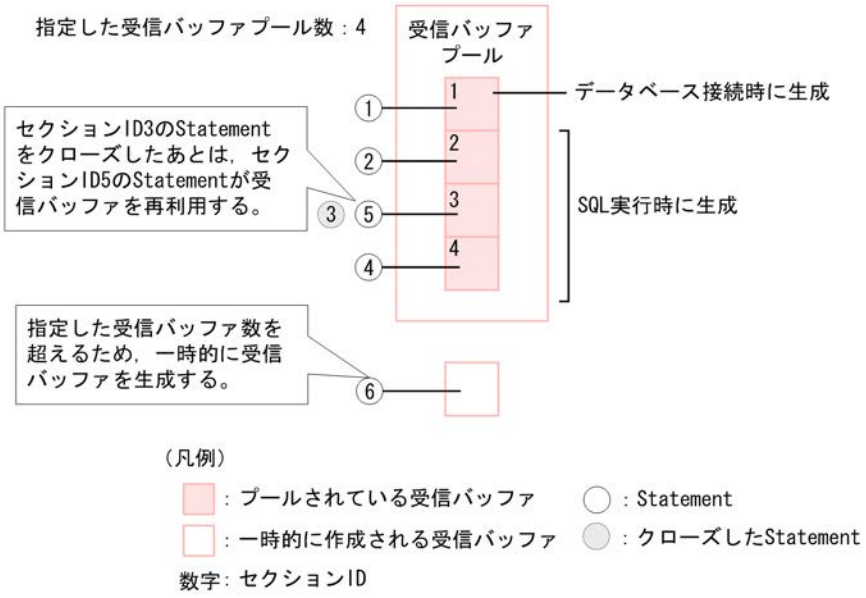
4.5.2　受信バッファプール数の指定

受信バッファプールに登録してプールする受信バッファの数を指定できます。この数のことを受信バッファプール数といいます。受信バッファプール数を指定することによって SQL 実行時の性能を向上でき，メモリ占有量を軽減できます。

受信バッファプール数は，1～1,024 個の範囲で指定できます。受信バッファプール数を指定すると，同一コネクション内で受信バッファが再利用されます。そのため，セッション ID が異なる Statement でも，同一コネクションであれば受信バッファを再利用できます。受信バッファが，指定した受信バッファプール数よりも多く必要になった場合は，一時的に受信バッファを生成し，DABroker Library から実行結果のデータを受信したあとに破棄します。このような一時的に生成した受信バッファは，受信バッファプールには登録されません。

受信バッファプール数を指定した場合の，受信バッファプールの管理の概要を次の図に示します。

図 4-2　受信バッファプールの管理の概要（受信バッファプール数を指定した場合）



受信バッファが生成されるタイミング

受信バッファが生成されるタイミングは，受信バッファプール数を指定しない場合と同じです。詳細については，「4.5.1　受信バッファの管理」を参照してください。

受信バッファを取得または返却するタイミング

受信バッファプールから受信バッファを取得するタイミング，および受信バッファを受信バッファプールに返却するタイミングは，SQL を実行するメソッドによって異なります。

各メソッドの動作は次の三つに分類されます。

表 4-17　メソッドの動作の分類

動作	説明
1	メソッド実行時に受信バッファを受信バッファプールから取得する。メソッド終了後，受信バッファを受信バッファプールに戻す。
2	動作 2 に分類されるメソッドのうち，最初に実行するメソッドで受信バッファを受信バッファプールから取得する。メソッドが終了しても，受信バッファは受信バッファプールに戻さない。

動作	説明
3	動作 2 に分類されるメソッドで取得した受信バッファを受信バッファプールに戻す。新たな受信バッファは取得しない。

受信バッファを再利用する際に受信バッファを取得または返却するメソッド，およびメソッドの動作を次の表に示します。動作の列の数字は，上記の表と対応しています。

表 4-18　SQL を実行するメソッド，およびメソッドの動作

クラス	メソッド	動作
Connection	prepareCall(String sql)	1
	prepareCall(String sql, int resultSetType, int resultSetConcurrency)	1
	prepareStatement(String sql)	1
	prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	1
DatabaseMetaData	getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	1
	getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	1
	getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	1
	getSchemas()	1
	getTableTypes()	1
	getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	1
	getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	1
	getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	1
	getPrimaryKeys(String catalog, String schema, String table)	1
	getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	1
Statement	executeBatch()	1
	execute(String sql)	1
	executeQuery(String sql)	1
	executeUpdate(String sql)	1
	close()※	3
PreparedStatement	execute()	1
	executeQuery()	1

クラス	メソッド	動作
PreparedStatement	executeUpdate()	1
ResultSet	absolute(int row)	2
	afterLast()	2
	first()	2
	next()	2
	close()	3

注※　ResultSet#close()メソッドを実行していない場合だけ使用できます。

(1)　受信バッファプール数の指定方法

受信バッファプール数は，次のどれかの方法で指定します。また，これらを同時に指定した場合，1，2，3の優先度順に決定されます。

1. DataSource#setBufferPoolSize()メソッドで指定する。
2. 接続するデータベースの設定で，getConnection メソッドおよび connect メソッドの引数 Properties info に，キーワード「BUF_POOL_SIZE」を指定する。
3. データベースの各種情報の設定で，引数 URL に「BUF_POOL_SIZE」を指定する。

指定方法の詳細については，「4.7.4　データベース接続の設定」を参照してください。

(2)　受信バッファプール数の目安

受信バッファプール数は，次の式で求められる数以上の数を目安に設定します。

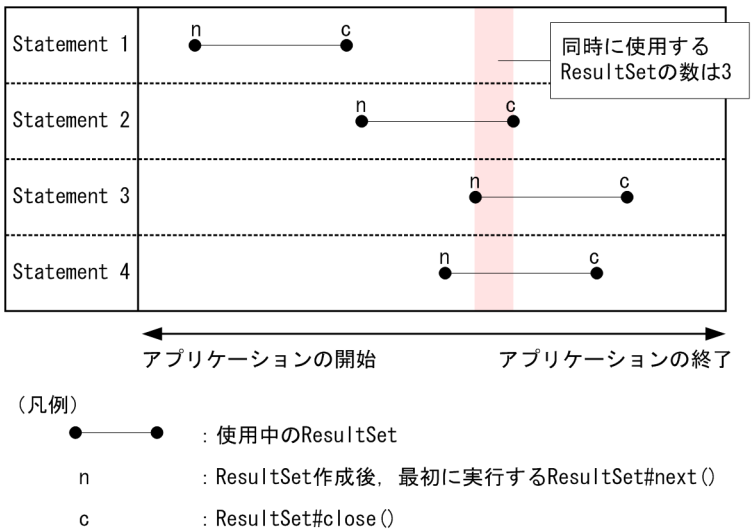
検索処理に必要な受信バッファ数+更新処理に必要な受信バッファ数

検索処理に必要な受信バッファ数

アプリケーションの中で同時に使用する ResultSet の数です。

実行する Statement が 4 個のアプリケーションの場合の，ResultSet の数の例を次の図に示します。

図 4-3　アプリケーション中で同時に使用する ResultSet の数



この図の場合，アプリケーションの開始から終了までの間に同時に使用する ResultSet の数は 3 のため，検索処理に必要な受信バッファ数は 3 となります。

更新処理に必要な受信バッファ数

次のどちらかで求めます。

- 更新処理を単一スレッドで逐次実行する場合は，1
- 更新処理をマルチスレッドで同時実行する場合は，アプリケーションの中で同時実行する更新 SQL の数

実際に必要な受信バッファ数が，この式を基に設定した受信バッファ数を超えた場合，受信バッファを新たに生成し，使用後に破棄します。このとき，生成した受信バッファは受信バッファプールに登録されません。この場合，オブジェクトの作成，破棄が頻発するため，性能が劣化することがあります。ただし，設定した受信バッファ数を超えた分の受信バッファは使用後に破棄するため，式を基に設定した場合よりメモリ占有量は軽減します。

4.6　DABroker Library を使用する場合のトラブルシュート

ここでは，DABroker Library を使用する場合のトラブルシュートについて説明します。

4.6.1　snapshot ログとして収集できる資料

DABroker Library の資料は，snapshot ログとして収集できます。

snapshot ログとして収集できる資料については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「付録 A　snapshot ログの収集対象一覧」を参照してください。

4.6.2　DABroker Library で取得できるログの種類と出力先

DABroker Library では，トラブルシューティングに使用するログとして，スプール情報を取得します。スプール情報として取得できるのは，次の情報です。

- メッセージログ
- 拡張データベースアクセストレース
- エラートレース
- Exception トレースログ
- イベントトレース（DABroker for C++をインストールしている場合）
- XA トレース

また，これらのログは，アプリケーションサーバのログにトレース共通ライブラリ形式（マルチプロセス）で出力されます。トレース共通ライブラリ形式のログの出力形式および出力項目については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「5.2.1　トレース共通ライブラリ形式のログの出力形式と出力項目」を参照してください。また，トレース共通ライブラリ形式のログを参照する場合の注意事項については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「5.2.2　トレース共通ライブラリ形式のログを参照する場合の注意」を参照してください。

「<DABroker Library の運用ディレクトリ>¥spool 下」（Windows の場合），または「<DABroker Library の運用ディレクトリ>/spool 下」（UNIX の場合）のすべてのファイルをコピーして保存してください。<DABroker Library の運用ディレクトリ>とは，「<アプリケーションサーバのインストールディレクトリ>¥DAB」（Windows の場合），または「/opt/DABroker」（UNIX の場合）のことです。なお，環境設定によって出力先を変更している場合は，変更した出力先のファイルをコピーして保存してください。

DABroker Library のスプール情報のうち，Exception トレースログは，運用開始前に取得のための設定をしておく必要があります。Exception トレースログ取得の設定については，「4.6.4　資料取得のための設定」を参照してください。

！ 注意事項

HP-UX でメッセージログファイルへの書き込みが失敗した場合，書き込みに失敗要因のメッセージと，メッセージログファイルへ書き込もうとしたメッセージをシステムログに出力します。この場合，システムログに対応していないメッセージを書き込むことがあるので，注意してください。

4.6.3　DABroker Library のスプール情報の内容

DABroker Library のスプール情報の調査方法について説明します。

DABroker Library のスプール情報には，次の種類があります。これらの情報は，<DABroker Library の運用ディレクトリ>¥spool（Windows の場合），または<DABroker Library の運用ディレクトリ>/spool（UNIX の場合）に出力されています。

表 4-19　DABroker Library のスプール情報

スプール情報の種類	対応するファイル
メッセージログ	dabsv.log dabsv.logold
拡張データベースアクセストレース	db_access ディレクトリ下のファイル
エラートレース	dabevttrcerr1.txt（ログ出力用ファイル） dabevttrcerr2.txt（バックアップ用ファイル）※1
Exception トレースログ	J2EE サーバ名称+MTrc.log および J2EE サーバ名称+MTrcOLD.log
イベントトレース※2	dabevttrcXXX1 および dabevttrcXXX2
XA トレース	db_xainfo.log

注※1　これらのほか，ワークファイルが，dabevttrcerr1.txt.lock という名称で作成されます。

注※2　DABroker for C++をインストールしている場合に出力されます。

それぞれの情報について説明します。

また，このほか，DABroker Library のログとしては，次の JDBC ドライバのトレースが取得できます。これらのトレースは，アプリケーション開発時に設定されている場合だけ出力されます。

- JDBC ドライバのトレース（JDBC インタフェースメソッドトレース）
- JDBC ドライバのトレース（DABroker Library とのイベントトレース）

(1)　メッセージログ

DABroker Library のメッセージが出力されたファイルです。

(2)　拡張データベースアクセストレース

データベースへの接続から切断までのアクセス情報を出力したトレースです。データベースへの接続ごとに出力されます。

拡張データベースアクセストレースは，次のように利用できます。

- データベースのアクセス中に発生したトラブルの迅速な切り分け
- 24 時間稼働に対応する J2EE アプリケーションの稼働状態でのトラブル調査
- マルチスレッド環境で J2EE アプリケーションが動作している場合のスレッドごとのデータベースアクセス履歴取得
- コネクションプーリングなどの機能を使用している場合のデータベースへの接続ごとのデータベースアクセス履歴取得

- トラブルが再現調査できない場合の常時トレース取得

なお，出力するイベントの種類や，トレースファイルの最大容量などは，DABroker Library の［環境設定ユティリティ］，または DABroker Library 動作環境定義ファイルで定義しておきます。DABroker Library の動作環境の設定については，Windows の場合は「4.3.1(3) DABroker Library の動作環境の設定（Windows の場合）」，UNIX の場合は「4.3.2(4) DABroker Library の動作環境の設定（UNIX の場合）」を参照してください。DABroker Library 動作環境定義ファイルの詳細については，「4.24 DABroker Library 動作環境定義ファイル」を参照してください。

(a)　出力形式

拡張データベースアクセストレースのファイル出力形式を次に示します。*斜体*は可変の情報です。拡張データベースアクセストレースは，次の 2 種類の情報で構成されます。

- ヘッダ情報
- 個々のトレース情報

個々のトレース情報は，出力項目を示すタイトル行の下に出力されます。

- Windows の場合

```
*-------------------------------------------------------*
*- DataBase Access Information (DRV)                   -*
*- Date  YYYY/MM/DD hh:mm:ss.nnnnnn                    -*
*-------------------------------------------------------*
DABroker Connect ID    : AAAAA(BBBBB)
Process ID             : CCCCC
UserID                 : DDDDD
Client Name            : EEEEE
(DB別出力情報)
Lang Mode              : GGGGG

THREAD-ID  CID   EVT   START-TIME   END-TIME   RETCODE  BLOCKCNT  (Windows QueryPerformance Counter)
HiRDB_CONNECTION_ID
 (トレース情報)
(SQL)SS…SS
 (トレース情報)
```

- UNIX の場合

```
*-------------------------------------------------------*
*- DataBase Access Information (DRV)                   -*
*- Date  YYYY/MM/DD hh:mm:ss.nnnnnn                    -*
*-------------------------------------------------------*
DABroker Connect ID    : AAAAA(BBBBB)
Process ID             : CCCCC
UserID                 : DDDDD
Client Name            : EEEEE
(DB別出力情報)
Lang Mode              : GGGGG

THREAD-ID  CID   EVT   START-TIME   END-TIME   RETCODE
BLOCKCNT   HiRDB_CONNECTION_ID
 (トレース情報)
(SQL)SS…SS
 (トレース情報)
```

(b)　出力項目

拡張データベースアクセストレースの出力項目について，次の表に示します。*斜体*は可変の情報です。

表 4-20　拡張データベースアクセストレースの出力項目

項目	内容
ヘッダ情報	

項目	内容	
DRV	DABroker Library のデータベースアクセスドライバの種別に従い，次の識別子が出力されます。 • HiRDB 接続の場合：HiRDB Driver • Oracle8i 接続の場合：ORACLE8i Driver※（Oracle Call Interface 8i を使用） • Oracle7 接続の場合：ORACLE7 Driver（Oracle Call Interface 7 を使用）	
Date:*YYYY/MM/DD hh:mm:ss.nnnnnn*	ファイル作成日時（YYYY/MM/DD hh:mm:ss.nnnnnn）が出力されます。 • YYYY：西暦年，MM：月，DD：日 • hh：時，mm：分，ss：秒，nnnnnn：マイクロ秒	
DABroker Connect ID:*AAAAA*(*BBBBB*)	データベース接続識別子が出力されます。	
Process ID:*CCCCC*	実行プロセス ID が出力されます。	
UserID:*DDDDD*	接続ユーザ ID が出力されます。 DABroker Library で ORACLE の OS 認証機能を使用した場合は，「OS authority」となります。	
Client Name:*EEEEE*	J2EE アプリケーション名称が出力されます。名称が設定されていない場合は空白になります。	
(DB 別出力情報)	接続データベースによって異なる情報が出力されます。	
	HiRDB の場合	• PDHOST： HiRDB 接続時の PDHOST の設定内容です。 未設定の場合は，"Not Indicator"が出力されます。 • PDNAMEPORT： HiRDB 接続時の PDNAMEPORT の設定内容です。 未設定の場合は，"Not Indicator"が出力されます。
	Oracle の場合	• SQLNET： Oracle リスナー名が出力されます。 未設定の場合は，"Not Indicator"が出力されます。
Lang Mode:*GGGGG*	実行言語モードが出力されます。	
個々のトレース情報		
THREAD-ID	実行スレッド ID が出力されます。	
CID	カーソル番号が出力されます。	
EVT	DABroker Library がデータベースに要求した処理内容を示す文字列が出力されます。データベースアクセストレースおよび拡張データベースアクセストレースの内容が含まれます。 DABroker Library の［環境設定ユティリティ］または DABroker Library 動作環境定義ファイルで設定された内容のイベントが出力されます。なお，すべてのイベントを出力しない設定にしている場合は，ヘッダだけが出力されます。なお，何か一つでもイベントを出力する設定にしている場合は，GET_DIAG イベントは必ず出力されます。 出力されるイベントを次に示します。 • ATTACH：サーバとのアクセスパス作成要求を実行します。 • ATTRGET：ハンドル属性の取得要求を実行します。	

項目	内容
EVT	• ATTRSET：ハンドル属性の設定要求を実行します。 • AUTOCOMOFF：AUTO COMMIT モードの解除要求を実行します。 • AUTOCOMON：AUTO COMMIT モードの設定要求を実行します。 • BIND：パラメタのバインド要求を実行します。 • CHARFORM：キャラクタセットフォームの取得要求を実行します。 • CHARID：キャラクタセット ID の取得要求を実行します。 • DEFINE：パラメタの定義要求を実行します。 • DESALLOC：記述子の領域確保を実行します。 • DESC_OUT：前処理した SQL の出力情報を返します。 • DESC_IN ：前処理した SQL の入力情報を返します。 • DESCANY：オブジェクトの記述要求を実行します。 • DESFREE：記述子の領域解放を実行します。 • DETACH：データベースアクセスパス削除要求を実行します。 • ENVCREATE：環境作成要求を実行します。 • ERRGET：エラー情報取得要求を実行します。 • EXECDIRECT：SQL の前処理後の実行処理です。 • GET_DIAG：エラー詳細情報の取得処理を実行します。 • GETINFO：接続後の動作情報の取得要求を実行します。 • GETPIECE：ピース情報の取得要求を実行します。 • HDLALLOC：ハンドルの領域確保を実行します。 • HDLFREE：ハンドルの領域解放を実行します。 • LOBCLOSE：BFILE データのクローズ要求を実行します。 • LOBLEN：LOB データの長さ取得要求を実行します。 • LOBOPEN：BFILE データのオープン要求を実行します。 • LOBREAD：LOB データの読み込み要求を実行します。 • LOBWRITE：LOB データの書き込み要求を実行します。 • PARAMGET：パラメタ情報の取得要求を実行します。 • SESBEGIN：セッション開始要求を実行します。 • SESEND：セッション終了要求を実行します。 • SETINFO：接続後の動作情報の設定要求を実行します。 • SETPIECE：ピース情報の設定要求を実行します。
START-TIME	アクセス開始日時（YYYY/MM/DD hh:mm:ss.nnnnnn）が出力されます。 • YYYY：西暦年，MM：月，DD：日 • hh：時，mm：分，ss：秒，nnnnnn：マイクロ秒
END-TIME	アクセス終了日時（YYYY/MM/DD hh:mm:ss.nnnnnn）が出力されます。 • YYYY：西暦年，MM：月，DD：日 • hh：時，mm：分，ss：秒，nnnnnn：マイクロ秒
RETCODE	データベースから返されるリターンコードが出力されます。
BLOCKCNT	FETCH 時の行取得件数が出力されます。

項目	内容
(Windows QueryPerformance Counter)	Windows の場合に，高分解パフォーマンス情報が表示されます。
HiRDB_CONNECTION_ID	コネクション ID（aaaa:bbbb:cccc）が出力されます。 • aaaa： フロントエンドサーバ名（HiRDB/Parallel Server 使用時）またはシングルサーバ名（HiRDB/Single Server 使用時） • bbbb： aaaa で示すサーバの接続通番 • cccc： aaaa で示すサーバのプロセス ID。 コネクション ID は，HiRDB クライアントのバージョンが 07-01 以降で，かつサーバが HiRDB である場合だけ出力されます。サーバが HiRDB で，かつ HiRDB クライアントのバージョンが 07-01 よりも前の場合は，ヘッダ（HiRDB_CONNECTION_ID）だけが出力され，コネクション ID は出力されません。また，サーバが XDM/RD E2 の場合は，ヘッダ（HiRDB_CONNECTION_ID）およびコネクション ID は出力されません。なお，この情報は，CONNECT，COMMIT，ROLLBACK，PREPARE，EXECUTE，DIRECT を実行したとき，最新の情報に更新されます。
(SQL)*SS…SS*	クライアントから要求のあった SQL の内容が出力されます。この項目は，必要に応じて出力されます。

注※

DABroker Library から Oracle8i，Oracle9i および Oracle10g の機能を使用する場合に出力されます。

(c)　ファイル名称規則

拡張データベースアクセストレースのファイルは，次のディレクトリに作成されます。

- Windows の場合
  <DABroker Library 運用ディレクトリ>¥spool¥db_access
- UNIX の場合
  <DABroker Library 運用ディレクトリ>/spool/db_access

拡張データベースアクセストレースのファイル名称は，次の規則に従います。*斜体は*可変の情報です。

spdb_*XXXXXXXX*_*YYYYYYYY*_*HHHHHHHHHHHHHHHH*_*<接続データベース名称>ZZZZZ*.*IIIIII*

ファイル名称中の文字列について次に示します。

表 4–21　拡張データベーストレースのファイル名称の規則

項目	意味
XXXXXXXX	実行プロセス ID が設定されます。
YYYYYYYY	データベース接続識別子が設定されます。
HHHHHHHHHHHHHHHH	実行プロセスごとにユニークな接続識別子が設定されます。
<接続データベース名称>	接続データベース名称です。指定したデータベース種別によって，次のどれかが設定されます。

項目	意味
<接続データベース名称>	• HiRDB の場合：HiRDB • Oracle8i 接続の場合：ORACLE8i • Oracle7 の場合：ORACLE7
ZZZZZ	識別順序番号が設定されます。 同じ実行プロセス ID で，かつ同じデータベース接続識別子のファイルがある場合に，00000 から 32767 までの番号を付加します。
IIIII	ファイル拡張子が設定されます。 • log：最新のトレースファイル • logold：1 世代前のトレースファイル

(3)　エラートレース

エラートレースは，ログファイルに出力される DABroker Library のエラー情報です。

エラートレースとして出力される情報を次に示します。

- **エラー事象を示す情報**
  エラーメッセージなどが出力されます。
- **データベース接続情報**
  データベース接続に関連する情報が出力されます。
- **履歴情報**
  エラーが発生する前に要求を受けた情報，前回コミットまたはロールバックが実行された時間，SQL 文の実行要求処理中の実行中カーソルでの要求ごとの履歴情報などが出力されます。

(a)　エラー事象を示す情報

エラー事象を示す情報の出力形式および出力項目について説明します。

出力形式

エラー事象を示す情報のファイル出力形式を次に示します。*斜体*は可変の情報です。

YYYY/MM/DD hh:mm:ss.sss TID=*AAAA SS..SS*

エラー事象を示す情報の出力例を次に示します。

```
2004/05/07 13:30:09.555  TID=4040 KFPA11561-E Specified authorization identifier has no connect privilege
```

出力項目

エラー事象を示す情報の出力項目について，次の表に示します。

表 4-22　エラートレースのエラー事象を示す情報の出力項目

項目	意味
YYYY/MM/DD hh:mm:ss.sss	エラー発生日時が出力されます。 *YYYY*：西暦年，*MM*：月，*DD*：日 *hh*：時，*mm*：分，*ss*：秒，*sss*：ミリ秒
AAAA	エラー発生スレッド ID（16 進数値）が出力されます。
SS...SS	エラーメッセージが出力されます。

(b)　データベース接続情報

データベース接続情報の出力形式および出力項目について説明します。

出力形式

エラートレースのデータベース接続情報のファイル出力形式を次に示します。*斜体*は可変の情報です。

YYYY/MM/DD hh:mm:ss.sss PID=*BBBB* InitID=*CCCC* ConID=*DDDD* CName=*EEEE* DBID=*FFFF* CID=*GGGG*(*HHHH*,*IIII*)

エラートレースのデータベース接続情報の出力例を次に示します。

```
2004/05/07 13:30:21.232 PID=1752 InitID=701 ConID=10701 CName=DABLIBAP DBID=1 CID=1(0,0)
```

出力項目

エラートレースのデータベース接続情報のファイル出力項目について，次の表に示します。

表 4-23　エラートレースのデータベース接続情報の出力項目

項目	意味
YYYY/MM/DD hh:mm:ss.sss	データベース接続日時が出力されます。 *YYYY*：西暦年，*MM*：月，*DD*：日 *hh*：時，*mm*：分，*ss*：秒，*sss*：ミリ秒
BBBB	プロセス ID（10 進数値）が出力されます。
CCCC	DABroker Library 接続 ID（16 進数値）が出力されます。
DDDD	データベース接続識別子（16 進数値）が出力されます。
EEEE	J2EE アプリケーションの名称が出力されます。ただし，名称なしの場合は空白となります。
FFFF	データベース ID（10 進数値）が出力されます。 1：HiRDB 7：Oracle8i 8：Oracle9i
GGGG	実行中カーソル ID（10 進数値）が出力されます。
HHHH	保守用情報が出力されます。
IIII	保守用情報が出力されます。

(c)　履歴情報

履歴情報の出力形式および出力項目について説明します。

出力形式

エラートレースの履歴情報のファイル出力形式を次に示します。

ヘッダ「START-TIME EXEC-TIME TID COUNT REQ-CODE DATA」の下に，次に示す形式でトレース情報が出力されます。*斜体*は可変の情報です。

YYYY/MM/DD hh:mm:ss.sss mmmm:ss.sss JJJJJJJJ KKKKKKKK LLLL MMMM NNNNNNNN 00000000 PP..PP

エラートレースの履歴情報の出力例を以下に示します。

```
START-TIME              EXEC-TIME    TID    COUNT  REQ-CODE  DATA
2004/05/07 15:37:16.733  0:00.030     1712       7 1022 NOW  00000000 00000001
2004/05/07 15:36:18.369  0:00.000     1712       6 1022 OLD  00000001 00000002
2004/05/07 15:35:59.191  0:00.281     1712       2 1021 PRE  00000000 00000002
```

```
INSERT INTO "HS
2004/05/07 15:36:18.369  0:00.000     1712        6 1022 EXE  00000001 00000002
```

出力項目

エラートレースの履歴情報のファイル出力項目について，次の表に示します。

表 4-24　エラートレースの履歴情報の出力項目

項目	意味
YYYY/MM/DD hh:mm:ss.sss	要求開始日時が出力されます。 *YYYY*：西暦年，*MM*：月，*DD*：日 *hh*：時，*mm*：分，*ss*：秒，*sss*：ミリ秒
mmmm:ss.sss	要求実行時間が出力されます。 *mmmm*：分，*ss*：秒，*sss*：ミリ秒
JJJJJJJ	要求元スレッド ID（16 進数値）が出力されます。
KKKKKKKK	コネクト要求からの累計要求カウント数（16 進数値）が出力されます。FET の場合は，SQL 文実行時の FET の要求回数 0xFFFFFFFF を超えた場合は，0 から循環します。
LLLL	要求コード（10 進数値）要求コードの詳細は，「表 4-25　要求コード・応答コード一覧」を参照してください。
MMMM	大きくわけて次の 3 種類に分類されます。 • 共通履歴情報 NOW：実行中の情報 OLD：一つ前の要求の情報 • SQL 文の実行要求時に指定されたカーソルの履歴情報 SPRE：SQL PREPARE 処理 EXE：SQL EXECUTE 処理 FET1：SQL FETCH 処理初回 FET：SQL FETCH 処理 CLS：SQL CLOSE 処理 • コミット/ロールバック情報 CMT：コミット処理 RBK：ロールバック処理 詳細は，「表 4-26　各履歴種別による DATA 内容」を参照してください。
NNNNNNNN	情報 1（16 進数値） 詳細は，「表 4-26　各履歴種別による DATA 内容」を参照してください。
OOOOOOOO	情報 2（16 進数値） 詳細は，「表 4-26　各履歴種別による DATA 内容」を参照してください。
PP..PP	詳細情報（任意文字列） 詳細は，「表 4-26　各履歴種別による DATA 内容」を参照してください。

データベース接続情報に出力される，要求コードおよび応答コードの一覧を次の表に示します。

表 4-25　要求コード・応答コード一覧

要求コード・応答コード（*LLLL*）	種別
1001	コネクション解放
1002	データベース接続
1003	データベース切り離し
1004	Commit
1005	Rollback
1006	表一覧取得準備
1007	表一覧取得
1008	表一覧取得後処理
1009	列一覧取得準備
1010	列一覧取得
1011	列一覧取得後処理
1012	索引一覧取得準備
1013	索引一覧取得
1014	索引一覧取得後処理
1015	表権限一覧取得準備
1016	表権限一覧取得
1017	表権限一覧取得後処理
1018	列権限一覧取得準備
1019	列権限一覧取得
1020	列権限一覧取得後処理
1021	SQL 実行準備
1022	SQL 実行
1023	SQL 実行結果取得
1024	SQL 実行後処理
1035	SQL 実行結果取得オプション
1036	BLOB ファイル削除
1037	プライマリキー一覧取得準備
1038	プライマリキー一覧取得
1039	プライマリキー一覧取得後処理
1040	BLOB，CLOB および BFILE 分割取得

要求コード・応答コード（*LLLL*）	種別
1041	自動 Commit 設定
1042	自動 Commit 解除
1050	XA トランザクション制御
1501	ストアドプロシジャ一覧取得準備
1502	ストアドプロシジャ一覧取得
1503	ストアドプロシジャ一覧取得後処理
1504	ストアドプロシジャ列一覧取得準備
1505	ストアドプロシジャ列一覧取得
1506	ストアドプロシジャ列一覧取得後処理
1507	ストアドプロシジャパラメタ定義情報取得
1508	ストアドプロシジャ実行準備
1509	ストアドプロシジャ実行
1510	ストアドプロシジャ拡張実行
2001	正常応答
2002	異常応答

データベース接続情報に出力される，各履歴種別による DATA 内容の一覧を次の表に示します。

表 4-26　各履歴種別による DATA 内容

履歴種別(*MMMM*)	情報 1(*NNNNNNNN*)	情報 2(*OOOOOOOO*)	詳細情報(*PP..PP*)
PRE	検索カラム数	?パラメタ定義数	SQL の先頭 16 バイト（マルチバイト文字コードの途中の場合はその文字は出力しません）
EXE	SQL 実行件数※	?パラメタ指定数	－
FET1	要求時検索件数	最大検索件数	－
FET	累計検索件数	要求時検索件数	－
CLS	0 固定	0 固定	－
CMT	－	－	－
RBK	－	－	－
NOW	今回の要求コードで取得した情報	－	－
OLD	前回の要求コードで取得した情報	－	－

（凡例）

－：該当しない

注※

INSERT，UPDATE，DELETE 実行時の件数

(4)　Exception トレースログ

Exception トレースログは，メソッド実行時に障害が発生した場合に障害要因を分析するためのトレース情報です。障害発生時の問題個所の切り分けをするときに有効です。メソッドの呼び出し情報，メソッドの戻り値の情報，および例外（Exception）の情報が，製品名称などが出力されたヘッダと合わせて出力されます。

Exception トレースログとして出力される情報を次に示します。

表 4-27　Exception トレースログとして出力される情報

情報の種類	内容
ヘッダ	出力情報の通番
	製品名称
メソッドの呼び出し情報	ログの取得時間
	スレッド識別情報
	メソッドの呼び出し識別情報
	クラス名，オブジェクト ID，メソッド名
	コネクションハンドル
	セクション ID（カーソル ID と同じことを表現しています）
	メソッドの引数
メソッドの戻り情報	ログの取得時間
	スレッド識別情報
	メソッドの戻り識別情報
	クラス名，オブジェクト ID，メソッド名
	コネクションハンドル
	セクション ID（カーソル ID と同じことを表現しています）
	メソッドの引数
Exception の情報	ログの取得時間
	スレッド識別情報
	Exception のスタックトレース

Exception トレースログは，SQLException，BatchUpdateException，または XAException が発生したときに，環境変数に設定した設定情報を基に，Exception が throw される前にファイルに出力されます。Exception トレースを出力するメソッドと，ログ出力に関連する環境変数については，「4.6.2　DABroker Library で取得できるログの種類と出力先」を参照してください。

(a)　出力形式と出力項目

Exception トレースログの出力形式を，トレース取得レベル 5 の場合の出力例を基に説明します。

```
　（続き）
2004/07/06 23:07:26.676 Thread[main,5,main]@1259414:[R][JdbcDbpsvStatement@1e4cbc4.execute]
                        HANDLE(406397096) : SID(2)
                        Return=true
2004/07/06 23:07:26.676 Thread[main,5,main]@1259414:[R][JdbcDbpsvStatement@1e4cbc4.executeQuery]
                        HANDLE(406397096) : SID(2)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvResultSet@3eca90
2004/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][JdbcDbpsvResultSet@3eca90.close]
                        HANDLE(406397096) : SID(2)
2004/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][JdbcDbpsvConnection@82c01f.commit]
                        HANDLE(406397096) : SID(0)
2004/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][JdbcDbpsvConnection@82c01f.commit]
                        HANDLE(406397096) : SID(0)
2004/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][JdbcDbpsvResultSet@3eca90.close]
                        HANDLE(406397096) : SID(2)
2004/07/06 23:07:28.332 Thread[Thread-
0,5,main]@30090737:[E][JdbcDbpsvConnection@82c01f.prepareStatement(1)]
                        HANDLE(406397096) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2004/07/06 23:07:28.332 Thread[Thread-
0,5,main]@30090737:[R][JdbcDbpsvConnection@82c01f.prepareStatement(1)]
                        HANDLE(406397096) : SID(3)
Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement@2808b3
2004/07/06 23:07:28.348 Thread[Thread-
1,5,main]@5462872:[E][JdbcDbpsvConnection@82c01f.prepareStatement(1)]
                        HANDLE(406397096) : SID(0)
                        sql=DELETE FROM SEINO_TABLE WHERE I1 = ?
2004/07/06 23:07:30.098 Thread[Thread-
1,5,main]@5462872:[R][JdbcDbpsvConnection@82c01f.prepareStatement(1)]
                        HANDLE(406397096) : SID(4)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement@922804
2004/07/06 23:07:30.332 Thread[Thread-2,5,main]@25253977:[E][JdbcDbpsvConnection@82c01f.rollback]
                        HANDLE(406397096) : SID(0)
2004/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977:[R][JdbcDbpsvConnection@82c01f.rollback]
                        HANDLE(406397096) : SID(0)
2004/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977:[E][JdbcDbpsvConnection@82c01f.close]
                        HANDLE(406397096) : SID(0)
2004/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977:[R][JdbcDbpsvConnection@82c01f.close]
                        HANDLE(406397096) : SID(0)
　（続く）
```

```
（続き）
                                                                12
2004/07/06 23:07:42.535 Thread[Thread-1,5,main]@5462872:JP.co.Hitachi.soft.DBPSV_Driver.SQLException:
KFDJ00002-E Statement or Connection is closed.[JdbcDbpsvPreparedStatement.setInt]
at JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement.setInt(JdbcDbpsvPreparedStatement.java:1363)
at Main3.run(Main3.java:35)
[2] Cosminexus DABroker for Java , DABroker for Java Version 2 02-11-/D
2004/07/06 23:07:25.723 Thread[Thread-
0,5,main]@1259414:[E][JdbcDbpsvConnection@82c01f.prepareStatement(1)]
                        HANDLE(406488568) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2004/07/06 23:07:25.770 Thread[Thread-1,5,main]@33409388:[E][JdbcDbpsvConnection@82c01f.rollback]
                        HANDLE(406488568) : SID(0)
2004/07/06 23:07:25.770 Thread[Thread-1,5,main]@33409388:[R][JdbcDbpsvConnection@82c01f.rollback]
                        HANDLE(406488568) : SID(0)
2004/07/06 23:07:25.770 Thread[Thread-
2,5,main]@11730319:[E][JdbcDbpsvConnection@82c01f.prepareStatement(1)]
                        HANDLE(406488568) : SID(0)
                        sql=SELECT ** FROM SEINO_TABLE
2004/07/06 23:07:25.863 Thread[Thread-2,5,main]@11730319:JP.co.Hitachi.soft.DBPSV_Driver.SQLException:
KFDJ00001-E Error occurred at server.[JdbcDbpsvResultSQLExecute.SQLPrepare]
OperationType : 2002
ReturnCode    : -100
ErrorCode     : -105
WarningInfo   : 0
ErrorMsg      : KFPA11105-E Invalid token "*" after token "*"
PreparedSQL   : SELECT
SectionID     : 1
at
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvResultSQLExecute.SQLPrepare(JdbcDbpsvResultSQLExecute.java:1413)
at
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvStatement.prepare(JdbcDbpsvStatement.java:2536)
          at
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement.<init>(JdbcDbpsvPreparedStatement.java:158)
          at
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection.prepareStatement(JdbcDbpsvConnection.java:590)
at Main4.run(Main4.java:53)
```

出力される内容について次の表で示します。項番は出力例中の番号と対応しています。

表 4-28　Exception トレースログの出力項目

項番	区分	表示	意味
1	ヘッダ	[XXXX]	出力情報の通番を示します。
2		Cosminexus DABroker for Java , DABroker for Java Version 2 XX-XX-XX	DABroker Library のログであることを示します。
3	共通	XXXX/XX/XX XX:XX:XX.XXX	ログの取得時間を示します。
4		Thread[XXXX,XXXXX,XXXX]@XXXX	スレッド識別情報を示します。
5		[JdbcDbpsvXXXX@XXXX.XXXX]	クラス名，オブジェクト ID，メソッド名を示します。
6		HANDLE(XXXX)	ConnectionHandle を示します。
7		SID(XXX)	セクション ID（カーソル ID と同じことを表現しています）を示します。
8	メソッドの呼び出し情報	[E]	メソッドの呼び出し識別情報を示します。
9		XXXX=XXXX	メソッドの引数を示します。

項番	区分	表示	意味
10	メソッドの戻り情報	[R]	メソッドからの戻り識別情報を示します。
11		Return=XXXX	メソッドの戻り値を示します。
12	Exception 情報	JP.co.Hitachi.soft.DBPSV_Driver.XXXXException:XXXX	Exception のスタックトレースを表示します。

(b)　解析方法

ここでは，(1)で示した出力例を使用して，Exception トレースログの解析方法について説明します。

Exception トレースログは，次の手順で解析します。

1. 解析するログの通番から，ログ情報を抜き出します。
2. ログ情報を，Thread 識別情報で分類して，ログごとに分割します。
3. ログの取得時間によって，ログ情報を時系列に並べます。

手順 1.～3.を実施すると，次のような表を作成できます。

時間	スレッド 1	スレッド 2	スレッド 3	スレッド 4
	Thread[main, 5,main] @1259414	Thread[Thread-0,5, main] @30090737	Thread[Thread-1,5, main] @5462872	Thread[Thread-2,5, main] @25253977
2004/07/06 23:07:09.129	JdbcDbpsvConnection @82c01f.createStatement(1)	－	－	－
2004/07/06 23:07:09.160	JdbcDbpsvStatement @1e4cbc4.execute	－	－	－
2004/07/06 23:07:14.285	JdbcDbpsvConnection @82c01f.commit	－	－	－
2004/07/06 23:07:14.301	JdbcDbpsvConnection @82c01f.prepareStatement(1)	－	－	－
2004/07/06 23:07:26.567	JdbcDbpsvConnection @82c01f.commit	－	－	－
2004/07/06 23:07:26.567	JdbcDbpsvStatement @1e4cbc4.executeQuery	－	－	－
2004/07/06 23:07:26.567	JdbcDbpsvStatement	－	－	－

時間	スレッド 1	スレッド 2	スレッド 3	スレッド 4
	Thread[main,5,main] @1259414	Thread[Thread-0,5,main] @30090737	Thread[Thread-1,5,main] @5462872	Thread[Thread-2,5,main] @25253977
2004/07/06 23:07:26.567	@1e4cbc4.execute	－	－	－
2004/07/06 23:07:28.332	JdbcDbpsvResultSet @3eca90.close	JdbcDbpsvConnection @82c01f.prepareStatement(1)	－	－
2004/07/06 23:07:28.332	JdbcDbpsvConnection @82c01f.commit	－	－	－
2004/07/06 23:07:28.348	－	－	JdbcDbpsvConnection @82c01f.prepareStatement(1)	－
2004/07/06 23:07:30.332	－	－	－	JdbcDbpsvConnection @82c01f.rollback
2004/07/06 23:07:42.098	－	－	－	JdbcDbpsvConnection @82c01f.close
2004/07/06 23:07:42.535	－	－	SQLException 発生 KFDJ00002-E Statement or Connection is closed.	－

（凡例）

－：該当しない

4. Exception エラーの内容を確認します。

2004/07/06 23:07:42.535 のスレッド 3 で SQLException が発生しています。メッセージの内容から，この時点で Statement オブジェクト，または Connection オブジェクトがすでにクローズされていることによるエラーだとわかります。

5. 時系列にオブジェクトの流れを確認します。

2004/07/06 23:07:09.129 のスレッド 1，2004/07/06 23:07:28.332 のスレッド 2，2004/07/06 23:07:28.348 のスレッド 3，2004/07/06 23:07:30.332 のスレッド 4 の Connection オブジェクトのオブジェクト ID が同じであることから，四つのスレッドが同一コネクションで処理されていることがわかります。

6. エラーの原因となっている個所を探します。

四つのスレッドが同一コネクションであることがわかったため，Statement.close メソッド，または Connection.close メソッドを実行している個所を探します。この例の場合は，スレッド 4 が 2004/07/06 23:07:42.098 で Connection.close メソッドを実行しています。

このことから，2004/07/06 23:07:42.535 のスレッド 3 で発生した SQLException の原因は，スレッド 4 が 2004/07/06 23:07:42.098 で Connection.close メソッドを実行していたためであることが，わかります。

(c)　トレース取得レベルごとの出力例

トレース取得レベルごとの出力例を次に示します。

トレース取得レベル 1 の場合の出力例

```
[1] Cosminexus DABroker for Java , DABroker for Java Version 2 02-11-/D
2004/07/21 17:59:41.001 Thread[main,5,main]@2113601537 [E][JdbcDbpsvXADataSource@59920001.getXAConnection(1)]
                        HANDLE(0) : SID(0)
2004/07/21 17:59:41.020 Thread[main,5,main]@2113601537 [E][JdbcDbpsvXAConnection@8bf8002.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 17:59:41.554 Thread[main,5,main]@2113601537 [R][JdbcDbpsvXAConnection@8bf8002.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@33690002
2004/07/21 17:59:41.554 Thread[main,5,main]@2113601537
 [R][JdbcDbpsvXADataSource@59920001.getXAConnection(1)]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAConnection@8bf8002
2004/07/21 17:59:41.557 Thread[main,5,main]@2113601537 [E][JdbcDbpsvXAConnection@8bf8002.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 17:59:41.557 Thread[main,5,main]@2113601537 [R][JdbcDbpsvXAConnection@8bf8002.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@33690002
2004/07/21 17:59:41.561 Thread[main,5,main]@2113601537 [E][JdbcDbpsvXAConnection@8bf8002.getConnection]
                        HANDLE(0) : SID(0)
2004/07/21 17:59:41.649 Thread[main,5,main]@2113601537 [R][JdbcDbpsvXAConnection@8bf8002.getConnection]
                        HANDLE(883163352) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection@78688002
2004/07/21 17:59:41.649 Thread[main,5,main]@2113601537 [E][JdbcDbpsvConnection@78688002.prepareStatement(1)]
                        HANDLE(883163352) : SID(0)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 17:59:41.819 Thread[main,5,main]@2113601537 [R][JdbcDbpsvConnection@78688002.prepareStatement(1)]
                        HANDLE(883163352) : SID(1)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement@d334002
2004/07/21 17:59:41.977 Thread[main,5,main]@2113601537 [E][JdbcDbpsvConnection@78688002.close]
                        HANDLE(883163352) : SID(0)
2004/07/21 17:59:42.886 Thread[main,5,main]@2113601537 [R][JdbcDbpsvConnection@78688002.close]
                        HANDLE(883163352) : SID(0)
2004/07/21 17:59:42.945 Thread[main,5,main]@2113601537 [E][JdbcDbpsvConnection@78688002.createStatement(1)]
                        HANDLE(883163352) : SID(0)
2004/07/21 17:59:42.946 Thread[main,5,main]@2113601537:JP.co.Hitachi.soft.DBPSV_Driver.SQLException:
KFDJ00006-E Connection is closed.[JdbcDbpsvConnection.createStatement(1)]
        at JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection.createStatement(JdbcDbpsvConnection.java:529)
        at TRC_ex.main(TRC_ex.java:81)
```

トレース取得レベル 2 の場合の出力例

```
[1] Cosminexus DABroker for Java , DABroker for Java Version 2 02-11-/D
2004/07/21 19:00:56.992 Thread[main,5,main]@2113866091 [E][JdbcDbpsvXADataSource@5996096b.getXAConnection(1)]
                        HANDLE(0) : SID(0)
2004/07/21 19:00:57.012 Thread[main,5,main]@2113866091 [E][JdbcDbpsvXAConnection@8bb8968.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 19:00:57.561 Thread[main,5,main]@2113866091 [R][JdbcDbpsvXAConnection@8bb8968.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@336d0968
2004/07/21 19:00:57.561 Thread[main,5,main]@2113866091 [R][JdbcDbpsvXADataSource@5996096b.getXAConnection(1)]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAConnection@8bb8968
2004/07/21 19:00:57.565 Thread[main,5,main]@2113866091 [E][JdbcDbpsvXAConnection@8bb8968.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 19:00:57.565 Thread[main,5,main]@2113866091 [R][JdbcDbpsvXAConnection@8bb8968.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@336d0968
2004/07/21 19:00:57.569 Thread[main,5,main]@2113866091 [E][JdbcDbpsvXAConnection@8bb8968.getConnection]
                        HANDLE(0) : SID(0)
2004/07/21 19:00:57.660 Thread[main,5,main]@2113866091 [R][JdbcDbpsvXAConnection@8bb8968.getConnection]
                        HANDLE(883163368) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection@7862c968
2004/07/21 19:00:57.660 Thread[main,5,main]@2113866091 [E][JdbcDbpsvConnection@7862c968.prepareStatement(1)]
                        HANDLE(883163368) : SID(0)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 19:00:57.779 Thread[main,5,main]@2113866091 [R][JdbcDbpsvConnection@7862c968.prepareStatement(1)]
```

```
                        HANDLE(883163368) : SID(1)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement@d4dc968
2004/07/21 19:00:57.840 Thread[main,5,main]@2113866091 [E][JdbcDbpsvPreparedStatement@d4dc968.executeUpdate(1)]
                        HANDLE(883163368) : SID(1)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 19:00:57.854 Thread[main,5,main]@2113866091 [R][JdbcDbpsvPreparedStatement@d4dc968.executeUpdate(1)]
                        HANDLE(883163368) : SID(1)
                        Return=1
2004/07/21 19:00:57.854 Thread[main,5,main]@2113866091 [E][JdbcDbpsvConnection@7862c968.close]
                        HANDLE(883163368) : SID(0)
2004/07/21 19:00:57.886 Thread[main,5,main]@2113866091 [R][JdbcDbpsvConnection@7862c968.close]
                        HANDLE(883163368) : SID(0)
2004/07/21 19:00:57.916 Thread[main,5,main]@2113866091 [E][JdbcDbpsvConnection@7862c968.createStatement(1)]
                        HANDLE(883163368) : SID(0)
2004/07/21 19:00:57.917 Thread[main,5,main]@2113866091:JP.co.Hitachi.soft.DBPSV_Driver.SQLException: KFDJ00006-
E Connection is closed.[JdbcDbpsvConnection.createStatement(1)]
        at JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection.createStatement(JdbcDbpsvConnection.java:529)
        at TRC_ex.main(TRC_ex.java:81)
```

（凡例）

下線は，トレース取得レベル 1 ではトレースが出力されないメソッドの情報です。

トレースレベル 3 の場合の出力例

```
[1] Cosminexus DABroker for Java , DABroker for Java Version 2 02-11-/D
2004/07/21 19:01:28.755 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXADataSource@5996095b.getXAConnection(1)]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:28.774 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAConnection@8bb8958.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:29.329 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAConnection@8bb8958.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@336d0958
2004/07/21 19:01:29.329 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXADataSource@5996095b.getXAConnection(1)]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAConnection@8bb8958
2004/07/21 19:01:29.332 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAConnection@8bb8958.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:29.332 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAConnection@8bb8958.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@336d0958
2004/07/21 19:01:29.333 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAResource@336d0958.start]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4958
                        flags=0
2004/07/21 19:01:29.372 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAResource@336d0958.start]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:29.389 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAConnection@8bb8958.getConnection]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:29.477 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAConnection@8bb8958.getConnection]
                        HANDLE(883163368) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection@78b60958
2004/07/21 19:01:29.478 Thread[main,5,main]@2113866075 [E][JdbcDbpsvConnection@78b60958.prepareStatement(1)]
                        HANDLE(883163368) : SID(0)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 19:01:29.571 Thread[main,5,main]@2113866075 [R][JdbcDbpsvConnection@78b60958.prepareStatement(1)]
                        HANDLE(883163368) : SID(1)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement@13934958
2004/07/21 19:01:29.614 Thread[main,5,main]@2113866075 [E]
[JdbcDbpsvPreparedStatement@13934958.executeUpdate(1)]
                        HANDLE(883163368) : SID(1)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 19:01:29.624 Thread[main,5,main]@2113866075 [R]
[JdbcDbpsvPreparedStatement@13934958.executeUpdate(1)]
                        HANDLE(883163368) : SID(1)
                        Return=1
2004/07/21 19:01:29.624 Thread[main,5,main]@2113866075 [E][JdbcDbpsvConnection@78b60958.close]
                        HANDLE(883163368) : SID(0)
2004/07/21 19:01:29.656 Thread[main,5,main]@2113866075 [R][JdbcDbpsvConnection@78b60958.close]
                        HANDLE(883163368) : SID(0)
2004/07/21 19:01:29.656 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAResource@336d0958.end]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4958
                        flags=67108864
2004/07/21 19:01:29.658 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAResource@336d0958.end]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:29.659 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAResource@336d0958.prepare]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4958
2004/07/21 19:01:29.675 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAResource@336d0958.prepare]
                        HANDLE(0) : SID(0)
```

```
                        Return=0
2004/07/21 19:01:29.675 Thread[main,5,main]@2113866075 [E][JdbcDbpsvXAResource@336d0958.commit]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4958
                        onePhase=false
2004/07/21 19:01:29.681 Thread[main,5,main]@2113866075 [R][JdbcDbpsvXAResource@336d0958.commit]
                        HANDLE(0) : SID(0)
2004/07/21 19:01:29.681 Thread[main,5,main]@2113866075 [E][JdbcDbpsvConnection@78b60958.createStatement(1)]
                        HANDLE(883163368) : SID(0)
2004/07/21 19:01:29.682 Thread[main,5,main]@2113866075:JP.co.Hitachi.soft.DBPSV_Driver.SQLException: KFDJ00006-
E Connection is closed.[JdbcDbpsvConnection.createStatement(1)]
        at JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection.createStatement(JdbcDbpsvConnection.java:529)
        at TRC_ex.main(TRC_ex.java:81)
```

（凡例）

太字は，トレース取得レベル 1 ではトレースが出力されないメソッドの情報です。

下線は，トレース取得レベル 2 ではトレースが出力されないメソッドの情報です。

トレース取得レベル 5 の場合の出力例

```
[1] Cosminexus DABroker for Java , DABroker for Java Version 2 02-11-/D
2004/07/21 18:58:47.122 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXADataSource@5996092a.getXAConnection(1)]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:47.141 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAConnection@8bb8929.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:47.705 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAConnection@8bb8929.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@336d0929
2004/07/21 18:58:47.706 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXADataSource@5996092a.getXAConnection(1)]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAConnection@8bb8929
2004/07/21 18:58:47.709 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAConnection@8bb8929.getXAResource]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:47.709 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAConnection@8bb8929.getXAResource]
                        HANDLE(0) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvXAResource@336d0929
2004/07/21 18:58:47.710 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAResource@336d0929.start]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4929
                        flags=0
2004/07/21 18:58:47.750 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAResource@336d0929.start]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:47.770 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAConnection@8bb8929.getConnection]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:47.858 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAConnection@8bb8929.getConnection]
                        HANDLE(883163368) : SID(0)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection@78454929
2004/07/21 18:58:47.858 Thread[main,5,main]@2113866026 [E][JdbcDbpsvConnection@78454929.prepareStatement(1)]
                        HANDLE(883163368) : SID(0)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 18:58:47.952 Thread[main,5,main]@2113866026 [R][JdbcDbpsvConnection@78454929.prepareStatement(1)]
                        HANDLE(883163368) : SID(1)
                        Return=JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvPreparedStatement@d2ec929
2004/07/21 18:58:47.994 Thread[main,5,main]@2113866026 [E][JdbcDbpsvPreparedStatement@d2ec929.executeUpdate(1)]
                        HANDLE(883163368) : SID(1)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 18:58:47.994 Thread[main,5,main]@2113866026 [E][JdbcDbpsvPreparedStatement@d2ec929.execute(1)]
                        HANDLE(883163368) : SID(1)
                        sql=INSERT INTO TRC_ex VALUES(?)
2004/07/21 18:58:48.004 Thread[main,5,main]@2113866026 [R][JdbcDbpsvPreparedStatement@d2ec929.execute(1)]
                        HANDLE(883163368) : SID(1)
                        Return=false
2004/07/21 18:58:48.004 Thread[main,5,main]@2113866026 [R][JdbcDbpsvPreparedStatement@d2ec929.executeUpdate(1)]
                        HANDLE(883163368) : SID(1)
                        Return=1
2004/07/21 18:58:48.004 Thread[main,5,main]@2113866026 [E][JdbcDbpsvConnection@78454929.close]
                        HANDLE(883163368) : SID(0)
2004/07/21 18:58:48.037 Thread[main,5,main]@2113866026 [R][JdbcDbpsvConnection@78454929.close]
                        HANDLE(883163368) : SID(0)
2004/07/21 18:58:48.037 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAResource@336d0929.end]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4929
                        flags=67108864
2004/07/21 18:58:48.039 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAResource@336d0929.end]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:48.039 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAResource@336d0929.prepare]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4929
2004/07/21 18:58:48.053 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAResource@336d0929.prepare]
```

```
                        HANDLE(0) : SID(0)
                        Return=0
2004/07/21 18:58:48.053 Thread[main,5,main]@2113866026 [E][JdbcDbpsvXAResource@336d0929.commit]
                        HANDLE(0) : SID(0)
                        xid=MyXid@5ffe4929
                        onePhase=false
2004/07/21 18:58:48.059 Thread[main,5,main]@2113866026 [R][JdbcDbpsvXAResource@336d0929.commit]
                        HANDLE(0) : SID(0)
2004/07/21 18:58:48.059 Thread[main,5,main]@2113866026
 [E][JdbcDbpsvConnection@78454929.createStatement(1)]
                        HANDLE(883163368) : SID(0)
2004/07/21 18:58:48.060 Thread[main,5,main]@2113866026:JP.co.Hitachi.soft.DBPSV_Driver.SQLException: KFDJ00006-
E Connection is closed.[JdbcDbpsvConnection.createStatement(1)]
        at JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvConnection.createStatement(JdbcDbpsvConnection.java:529)
        at TRC_ex.main(TRC_ex.java:81)
```

（凡例）

太字は，トレース取得レベル 1 ではトレースが出力されないメソッドの情報です。

下線は，トレース取得レベル 2 ではトレースが出力されないメソッドの情報です。

(5)　イベントトレース

イベントトレースは，DABroker for C++をインストールしている場合に出力されます。

DABroker for C++の機能を使ってデータベースにアクセスする場合のイベントトレースが出力されます。イベントトレースは，次のファイルに取得されます。

- Windows の場合
  <DABroker Library 運用ディレクトリ>¥spool¥dabevttrccpp1
- UNIX の場合
  <DABroker Library 運用ディレクトリ>/spool/dabevttrccpp1

なお，イベントトレースファイルが満杯になった場合，ラップアラウンドでトレースが出力されます。この場合は，バックアップファイル（dabevttrccpp2）が作成され，1 世代だけ（dabevttrccpp1）が管理されます。

取得するイベントトレースの設定については，「4.7.4　データベース接続の設定」を参照してください。

(a)　出力形式

イベントトレースの出力形式について説明します。

ヘッダ「LEVEL VPID PID TID DATE TIME EVENT」の下に，次に示す形式でトレース情報が出力されます。

レベル 仮想プロセスID プロセスID スレッドID 日付 時刻 動作結果

イベントトレースの出力例を次に示します。

```
 LEVEL    VPID     PID  TID  DATE       TIME           EVENT
40000010 00120000 711a 0001 2004/06/01 11:38:29.300000 イベント1
40000010 00120000 711a 0002 2004/06/01 11:38:30.100000 イベント2
40000010 00120000 711b 0001 2004/06/01 11:38:30.150000 イベント3
40000010 00120000 711b 0002 2004/06/01 11:38:30.250000 イベント4
```

(b)　出力項目

イベントトレースの出力項目について，次の表に示します。

表 4-29　DABroker Library のイベントトレースの出力項目

ヘッダ	項目	説明
LEVEL	レベル	トレースに出力したレベルが出力されます。
VPID	仮想プロセス ID	仮想プロセス ID が出力されます。
PID	プロセス ID	プロセス ID が出力されます。
TID	スレッド ID	スレッド ID が出力されます。
DATE	日付	取得した日付が出力されます。
TIME	時刻	取得した時刻が出力されます。
EVENT	動作結果	実行時の動作結果のログが出力されます。

(6)　XA トレースの取得形式

XA インタフェースでのトラブルシュート情報は，XA トレースファイルとして次のファイルに出力されます。

- Windows の場合
  <DABroker Library 運用ディレクトリ>¥spool¥db_xainfo.log
- UNIX の場合
  <DABroker Library 運用ディレクトリ>/spool/db_xainfo.log

(a)　出力形式

XA トレースファイルの出力形式を次に示します。*斜体*は可変の情報です。

日付 時刻 プロセスID:スレッドID イベント種別:XA関数名 rmid=*rrrr* flags=*ffff* RMName="*nnnn*"
xa_info="*iiii*" xid=[*ffff*:*gggg*:*bbbb*:*dddd*] count=*cccc* return code=*rrrr*

XA トレースファイルの出力例を次に示します。

```
2004/10/20 10:27:12.156000 [226:225] [end :xa_commit  ] rmid=1 flags=0
RMName="Oracle_XA"
xid=[00000001:10:8:313939392D31302D323031303A32363A34371906B8003C1BAB0000000
0000094000000060000000100000003C0000000200000003C0000000400000081000000080000
03C0000000B0000003C0000006000000003C2A1B1C2D0303B600373A0A190AB8003D390BA7002
B57BB0023591201B7002B1906B60032B600403A072AB400]
  return code = -4
2004/10/20 10:27:19.947000 [226:225] [end :xa_recover  ] rmid=1 flags=0
count=1 RMName="Oracle_XA"
xid=[00000000:0:0:000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000]
  return code = -5
```

(b)　出力項目

XA トレースファイルの出力項目について，次の表に示します。

表 4-30　DABroker Library の XA トレースの出力項目

項目名	説明
日付	出力した年月日（YYYY/MM/DD）が出力されます。 • YYYY：西暦年，MM：月，DD：日
時刻	出力した時刻（hh:mm:ss:nnnnnn）が表示されます。 • hh：時，mm：分，ss：秒，nnnnnn：マイクロ秒
[プロセス ID：スレッド ID]	プロセス ID とスレッド ID が出力されます。
[イベント種別：XA 関数名]	イベント種別と，XA 関数の名前が出力されます。 • イベント種別 次のどちらかが出力されます。 "call"：関数呼び出しであることを示します。 "end"：関数リターンであることを示します。 • XA 関数名 XA 関数名が出力されます。
rmid=*rrrr*	RMID が出力されます。
flags=*ffff*	フラグパラメタが出力されます。
RMName="*nnnn*"	RM 名称が出力されます。
xa_info="*iii* "	XA 文字列が出力されます。 この項目は，XA 関数が，xa_open()，または xa_close()の場合だけ，出力されます。
xid=[*ffff:gggg:bbbb:dddd*]	XID が出力されます。 この項目は，XA 関数が，xa_open()，または xa_close()以外の場合だけ，出力されます。 *ffff*：フォーマット ID が出力されます。 *gggg*：グローバルトランザクション ID 長が出力されます。 *bbbb*：ブランチ修飾子長が出力されます。 *dddd*：XID データが出力されます。
count=*cccc*	リカバリ情報取得カウントが出力されます。 この項目は，XA 関数が xa_recover()の場合だけ，出力されます。
return code ＝*rrrr*	XA 関数の戻り値が出力されます。 この項目は，イベント種別が"end"の場合だけ出力されます。

(7)　JDBC ドライバのトレース（JDBC インタフェースメソッドトレース）

J2EE アプリケーションまたはバッチアプリケーション開発時に，DABroker Library の API を使用して，JDBC ドライバのトレースを出力するように設定できます。

ここでは，JDBC ドライバのトレースのうち，JDBC インタフェースメソッドのトレースについて説明します。

(a)　出力形式

JDBC インタフェースメソッドのトレースの出力形式を次に示します。*斜体*は可変の情報です。

```
[DABroker for Java Version 2] [JDBC Interface メソッド状態][JdbcDbpsvクラス名.メソッド名] 引数または戻り値=値
```

JDBC インタフェースメソッドのトレースの出力例を次の図に示します。

```
[DABroker for Java Version 2][JDBC Interface Entry ][JdbcDbpsvDriver.connect]
[DABroker for Java Version 2]              url=jdbc:hitachi:dbpsv://ikn0205/DB=hirdb
[DABroker for Java Version 2]              info={user=jdbc, password=jdbc,
UAPNAME=sample, JDBC_IF=ON, SV_EVENT=ON, TRC_NO=1000}
[DABroker for Java Version 2][JDBC Interface Return][JdbcDbpsvDriver.connect]
[DABroker for Java Version 2]              Return=JP.co.Hitachi.soft.DBPSV_Driver.
JdbcDbpsvConnection@1cee91
                                    :
```

(b)　出力項目

JDBC インタフェースメソッドのトレースの出力項目について，次の表に示します。

表 4-31　JDBC インタフェースメソッドのトレースの出力項目

項目	説明
[DABroker for Java Version 2]	DABroker Library のトレースであることを示します。
[JDBC Interface *メソッド状態*]	使用した JDBC インタフェースメソッドの状態を示します。 Entry：メソッドの呼び出し Return：メソッドの戻り
[JdbcDbpsv *クラス名.メソッド名*]	使用した JDBC インタフェースメソッドの「クラス名.メソッド名」を示します。
引数または戻り値=値	該当する「クラス名.メソッド名」の引数の値，または戻り値が出力されます。 引数の場合，*引数名=値*の形式で出力されます。 戻り値の場合，Return=*値*の形式で出力されます。

(8)　JDBC ドライバのトレース（DABroker Library とのイベントトレース）

J2EE アプリケーションまたはバッチアプリケーション開発時に，DABroker Library の API を使用して，JDBC ドライバのトレースを出力するように設定できます。

ここでは，JDBC ドライバと DABroker Library とのイベントトレースについて説明します。

(a)　出力形式

DABroker Library とのイベントトレースの出力形式を次に示します。

[DABroker for Java Version 2][Server Event *イベント*]*送信データ種別*

DABroker Library とのイベントトレースの出力例を次に示します。

```
[DABroker for Java Version 2][Server Event Get]Result(0,0) SIZE:91
[DABroker for Java Version 2][Server Event Put]Connect(Database) SIZE:112
                                    :
```

(b)　出力項目

DABroker Library とのイベントトレースの出力項目について，次の表に示します。

表 4–32　DABroker とのイベントトレースの出力項目

項目	説明	
[DABroker for Java Version 2]	DABroker Library のトレースであることを示します。	
[Server Event *イベント*]	DABroker Library でのイベントを示します。	
送信データ種別	送受信データ種別が，*aa…aa <bb…bb>（cc…cc,dd…dd）SIZE：XXX* の形式で出力されます	
	aa…aa	オペレーション種別が出力されます。 表示されるオペレーション種別とその意味は次のとおりです。 • Disconnect(Connection)：DABroker Library とのコネクション解放要求 • Connect(Database)：データベースとの接続要求 • Disconnect(Database)：データベースとの接続解放要求 • Commit：トランザクションのコミット要求 • Rollback：トランザクションのロールバック要求 • LIST(Table)：表一覧取得要求 • LIST(Column)：列一覧取得要求 • LIST(Index)：索引一覧取得要求 • LIST(TablePrivileges)：表権限一覧取得要求 • LIST(ColumnPrivileges)：列権限一覧取得要求 • LIST(Procedure)：プロシジャ一覧取得要求 • LIST(ProcedureColumn)：プロシジャ列一覧取得要求 • EXT-CMD：SQL の実行要求 • EXE(Procedure)：プロシジャの実行要求 • Cancel：非同期キャンセル要求 • Result：正常応答 • Error：エラー応答 • Result(Cancel)：正常応答（非同期キャンセル） • Error(Cancel)：エラー応答（非同期キャンセル） • XA：CALL：XA オペレーションの実行 • XA：AUTOCOMMIT<ON>：XALocalCommitMode（true）の実行 • XA：AUTOCOMMIT<OFF>：XALocalCommitMode（false）の実行
	bb…bb	付加情報が出力されます。 オペレーション種別が，LIST（xxxxx）および EXT-CMD の場合に表示されます。
	cc…cc	リターンコードが出力されます。
	dd…dd	詳細コードが出力されます。
	XXX	DABroker Library でのイベントが Put の場合は，送信したデータのサイズが出力されます。

項目	説明	
送信データ種別	*XXX*	DABroker Library でのイベントが Get の場合は，受け取ったデータのサイズが出力されます。

4.6.4　資料取得のための設定

DABroker Library のログは，トラブルシューティングに必要な資料です。

環境設定ユティリティまたは DABroker Library 動作環境定義ファイルで，ログファイルサイズなどを設定してください。なお，デフォルトの資料取得の設定から変更しない場合は，設定は不要です。

DABroker Library では，JDBC ドライバのトレース情報やスプール情報を取得できます。また，障害発生時には，アプリケーションサーバの定義を取得してください。これらの情報を使用して，障害が発生したときに設定されていた定義内容を確認します。

ここでは，これらの情報を取得するための設定について説明します。

(1)　JDBC ドライバのトレース情報の取得

DABroker Library では，スプール情報のほかに次の JDBC ドライバのトレースも取得できます。

- JDBC インタフェースメソッドトレース
- DABroker Library のイベントトレース

ただし，これらのトレースを取得するためには，アプリケーション開発時に JDBC ドライバのトレース取得のための設定をしておく必要があります。設定方法については，「4.7.4　データベース接続の設定」を参照してください。

(2)　スプール情報の取得

DABroker Library では，スプール情報として，メッセージログ，拡張データベースアクセストレース，Exception トレースログなどを取得できます。環境設定ユティリティ（Windows の場合）または DABroker Library 動作環境定義ファイル（UNIX の場合）で，メッセージログ，拡張データベースアクセストレースのファイルサイズなどを設定してください。環境設定ユティリティまたは DABroker Library 動作環境定義ファイルの設定については，Windows の場合は「4.3.1(3) DABroker Library の動作環境の設定（Windows の場合)」，UNIX の場合は「4.3.2(4) DABroker Library の動作環境の設定（UNIX の場合)」を参照してください。

また，Exception トレースログを取得するためには，運用開始前に取得のための設定が必要です。ここでは，Exception トレースログ取得の設定について説明します。

Exception トレースログ取得の設定では，次の内容を設定します。

- トレース取得レベル
- ファイル出力先
- ファイル出力情報量
- メモリ内取得情報量

それぞれの設定方法について説明します。

● トレース取得レベル

Exception トレースログは，メソッド実行時に，SQLException，BatchUpdateException，または XAException が発生した場合に出力されます。Exception トレースログを出力するメソッドには，メソッドごとにトレース取得レベルが定義されています。出力するトレース取得レベルを指定することで，目的に応じた内容のトレースを出力できるようになります。

- **設定方法**

  トレース取得レベルは，次の環境変数で指定します。

環境変数名	説明
DAB_TRACELEVEL	トレース取得レベルとして，1～5 を指定します。ただし，トレース取得レベル 4 は現在未使用であり，4 が指定された場合は 3 と同じ動きとなります。 • ここでは，どのトレース取得レベルまでログを取得するかを指定します。例えば，トレース取得レベル 3 を指定した場合は，トレース取得レベル 1 および 2 の内容が取得できます。トレース取得レベル 5 が指定された場合は，内部呼び出しを含め，ログ取得対象メソッドをすべて取得します。 • 指定を省略した場合，または不正な値を指定した場合は，トレース取得レベルとして，1 を仮定します。 • ログを取得するメソッドは，指定された値で取得されるメソッドすべてとなります。 • 指定値の前後にスペースがあった場合は，スペースを削除した値を指定値として扱います。

- **トレース取得レベルと取得できるログの内容**

  Exception トレースログのトレース取得レベルと取得できるログの内容を次の表に示します。

  表 4–33　Exception トレースログのトレース取得レベル

トレース取得レベル	取得できるログの内容	説明
1	• データベースとの接続と切断 • ステートメントの作成とクローズ • 静的 SQL の実行	標準のトレース取得レベルです。コネクションとステートメントの状態と実行順序を調査したい場合に指定してください。
2	• トレース取得レベル 1 の内容 • トランザクションのコミット，およびロールバック • データベースへのアクセス要求 • ResultSet クラスの処理 • 各オブジェクトの処理	データベースのアクセス要求の状態と実行順序を調査したい場合に指定してください。ソースとログの比較が容易になります。
3	• トレース取得レベル 1 および 2 の内容 • XA 要求	XA 要求の状態と実行順序を調査したい場合に指定してください。
5	• トレース取得レベル 1，2 および 3 の内容 • JDBC ドライバの内部呼び出し	内部処理の状態と実行順序を調査したい場合に指定してください。

- **Exception トレースログを出力するメソッドのトレース取得レベル**

  Exception トレースログを出力するメソッドのトレース取得レベルを次の表に示します。

表 4-34　Exception トレースログを取得するメソッドのトレース取得レベル

クラス名	メソッド名	トレース取得レベル	備考
CallableStatement	boolean execute()※1	2	Statement クラス，および PreparedStatement クラスのメソッドをオーバーライドしたメソッド。
	ResultSet executeQuery()※1	2	
	int executeUpdate()※1	2	
	boolean execute(String sql)※2	1	
	int[] executeBatch()	2	
	ResultSet executeQuery(String sql)※2	1	
	int executeUpdate(String sql)※2	1	
Connection	void close()	1	−
	void commit()	2	−
	Statement createStatement()※1	1	−
	Statement createStatement(int resultSetType, int resultSetConcurrency)※2	1	−
	DatabaseMetaData getMetaData()	2	−
	CallableStatement prepareCall(String sql)※1	1	−
	CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)※2	1	−
	PreparedStatement prepareStatement(String sql)※1	1	−
	PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)※2	1	−
	void rollback()	2	−
	void setAutoCommit(boolean autoCommit)	2	−
DatabaseMetaData	ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	2	−
	ResultSet getCatalogs()	2	−
	ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	2	−

クラス名	メソッド名	トレース取得レベル	備考
DatabaseMetaData	ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	2	−
	Connection getConnection()	2	−
	ResultSet getCrossReference(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)	2	−
	ResultSet getExportedKeys(String catalog, String schema, String table)	2	−
	ResultSet getImportedKeys(String catalog, String schema, String table)	2	−
	ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	2	−
	ResultSet getPrimaryKeys(String catalog, String schema, String table)	2	−
	ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	2	−
	ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	2	−
	ResultSet getSchemas()	2	−
	ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	2	−
	ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	2	−
	ResultSet getTableTypes()	2	−
	ResultSet getTypeInfo()	2	−
	ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	2	−
	ResultSet getVersionColumns(String catalog, String schema, String table)	2	−

クラス名	メソッド名	トレース取得レベル	備考
Driver	Connection connect(String url, Properties info)	1	−
PreparedStatement	boolean execute()※1	2	−
	ResultSet executeQuery()※1	2	−
	int executeUpdate()※1	2	−
	ResultSetMetaData getMetaData()	2	−
	boolean execute(String sql)※2	1	Statement クラスのメソッドをオーバーライドしたメソッド。
	int[] executeBatch()	2	
	ResultSet executeQuery(String sql)※2	1	
	int executeUpdate(String sql)※2	1	
ResultSet	boolean absolute(int row)	2	データベースにアクセスするときだけ取得します。
	void afterLast()	2	
	void beforeFirst()	2	
	void close()	2	−
	boolean first()	2	データベースにアクセスするときだけ取得します。
	ResultSetMetaData getMetaData()	2	−
	Statement getStatement()	2	−
	boolean last()	2	データベースにアクセスするときだけ取得します。
	boolean next()	2	
	boolean relative(int rows)	2	
Statement	void cancel()	2	−
	void close()	1	−
	boolean execute(String sql)	1	−
	int[] executeBatch()	2	−
	ResultSet executeQuery(String sql)	1	−
	int executeUpdate(String sql)	1	−
	ResultSet getResultSet()	2	−
DataSource	Connection getConnection()※1	1	−
	Connection getConnection(String username, String password)※2	1	−

クラス名	メソッド名	トレース取得レベル	備考
PooledConnection	void close()	1	−
	Connection getConnection()	1	−
ConnectionPoolDataSource	PooledConnection getPooledConnection()※1	1	−
	PooledConnection getPooledConnection(String username, String password) ※2	1	−
XAConnection	XAResource getXAResource()	1	−
XADataSource	XAConnection getXAConnection()※1	1	−
	XAConnection getXAConnection(String user, String password)※2	1	−
XAResource	void commit(Xid xid, boolean onePhase)	3	−
	void end(Xid xid, int flags)	3	−
	void forget(Xid xid)	3	−
	boolean isSameRM(XAResource xares)	3	−
	int prepare(Xid xid)	3	−
	Xid[] recover(int flag)	3	−
	void rollback(Xid xid)	3	−
	void start(Xid xid, int flags)	3	−

（凡例）−：該当しない

注※1　メソッド名は，「メソッド名(1)」と出力されます。

注※2　メソッド名は，「メソッド名(2)」と出力されます。

● ファイル出力先

Exception トレースログを出力するファイルの出力先を設定できます。

- **設定方法**

  ファイル出力先は，次の環境変数で指定します。

環境変数名	説明
DAB_FILEDIR	Exception トレースログの出力先のディレクトリを絶対パスで指定します。 • 指定を省略した場合は，次の場所に Exception トレースログが出力されます。 Windows の場合 <DABroker Library 運用ディレクトリ>¥spool UNIX の場合

環境変数名	説明
DAB_FILEDIR	<DABroker Library 運用ディレクトリ>/spool • ファイルの出力先として，<ファイルの出力先ディレクトリ>¥<J2EE サーバ名称+MTrc.log>のバイト数が 259 バイトを超える値は指定しないでください。環境変数の設定値のバイト数の上限は，1,024 バイトです。上限を超えた場合は，先頭から 1,024 バイト分だけが有効になります。 • 指定値の前後にスペースがあった場合は，スペースを削除した値を指定値として扱います。

● ファイル出力情報量

Exception トレースログを出力するファイルの出力情報量を設定できます。

• 設定方法

ファイル出力情報量は，次の環境変数で指定します。

環境変数名	説明
DAB_FILEOUTNUM	ファイル出力情報量として，メモリ内取得情報をファイルに出力する回数の上限を指定します。指定できる範囲は，1～50 です。 • 指定を省略した場合，または不正な値を指定した場合は，出力情報量として，5 を仮定します。指定された値が 1 以下の場合は 1, 50 以上の場合は 50 を出力情報量として仮定します。 • 指定した上限を超えて出力しようとすると，カレントファイルは 1 世代前のファイルとなり，新しくカレントファイルが作成されて，情報が出力されます。 • 環境変数の設定値のバイト数の上限は，10 バイトです。上限を超えた場合には，先頭から 10 バイト分だけが有効になります。 • 指定値の前後にスペースがあった場合は，スペースを削除した値を指定値として扱います。

• 情報量の上限

一つのファイルに出力する情報量の上限（単位：行）は，次の算出式で求められます。

情報量の上限＝（DAB_ONMEMNUMの指定値）×（DAB_FILEOUTNUMの指定値）

なお，Exception トレースログを出力するためのファイルサイズの概算（単位：キロバイト）は，次の算出式で求められます。

出力するファイルサイズ＝2×メモリ内取得情報量（DAB_ONMEMNUMの指定値）×ファイル出力情報量（DAB_FILEOUTNUMの指定値）×2

● メモリ内取得情報量

Exception トレースログを出力するファイルのメモリ内取得情報量を設定できます。

• 設定方法

メモリ内取得情報量は，次の環境変数で指定します。

環境変数名	説明
DAB_ONMEMNUM	メモリ内取得情報量として，メモリ内で保持する情報量の上限を指定します。指定できる範囲は 500～10,000（単位は，行※）です。

環境変数名	説明
DAB_ONMEMNUM	• 指定を省略した場合，または不正な値を指定した場合は，メモリ内取得情報量として，1,000 を仮定します。指定された範囲が 500 以下の場合は 500，10,000 以上の場合は 10,000 をメモリ内取得情報量として仮定します。 • 指定した上限を超えて保持しようとすると，ラップアラウンドします。 • 環境変数の設定値のバイト数の上限は，10 バイトです。上限を超えた場合には，先頭から 10 バイト分だけが有効になります。 • 指定値の前後にスペースがあった場合は，スペースを削除した値を指定値として扱います。

注※　1 行とは，メソッドの呼び出し情報，メソッドの戻り情報，および Exception 情報のそれぞれで一度に取得する情報のことです。

- **メモリ所要量の算出式**

  Exception トレースログを出力するために必要となるメモリ所要量（単位：キロバイト）は，次の算出式で求められます。

  メモリ所要量＝2×メモリ内取得情報量（DAB_ONMEMNUMの指定値）

(3)　DABroker Library に関する定義情報

トラブルが発生した際に，次のファイルを取得するようにしてください。

Windows の場合

- <DABroker Library 運用ディレクトリ>¥conf¥dasys.conf（DABroker Library 動作環境定義ファイル）
- <DABroker Library 運用ディレクトリ>¥conf¥dabenv（接続先データベース定義ファイル）

UNIX の場合

- <DABroker Library 運用ディレクトリ>/conf/dasys.conf（DABroker Library 動作環境定義ファイル）
- <DABroker Library 運用ディレクトリ>/conf/dabenv（接続先データベース定義ファイル）

トラブルシューティング用に保存するファイルからは，パスワードなどの公開できない情報を削除することをお勧めします。

4.6.5　DABroker Library で取得できるトレース

DABroker Library の動作環境設定で，必要に応じて，トレース取得の設定をしてください。取得できるトレースを次に示します。

- 拡張データベースアクセストレース
- XA トレース

拡張データベースアクセストレース

通常の運用ではトレースは採取しませんが，障害などの不具合の調査が必要なときに，拡張データベースアクセストレースを取得します。

拡張データベースアクセストレースは，データベースへの接続から切断までのアクセス情報を出力したトレースです。データベースへの接続ごとに出力されます。

拡張データベースアクセストレースは，次のように利用できます。

- データベースのアクセス中に発生したトラブルの迅速な切り分け
- 24 時間稼働に対応する J2EE アプリケーションの稼働状態でのトラブル調査
- マルチスレッド環境で J2EE アプリケーションが動作している場合のスレッドごとのデータベースアクセス履歴取得
- コネクションプーリングなどの機能を使用している場合のデータベースへの接続ごとのデータベースアクセス履歴取得
- トラブルが再現調査できない場合の常時トレース取得

XA トレース

XA トレースは XA インタフェースでエラーを検知した場合に，エラートレースを取得します。XA インタフェースでエラーが発生した場合にエラーの内容をトレースできるように，XA トレースの取得を設定してください。

4.6.6　DABroker Library のトレースファイル運用時の留意事項

ここでは，DABroker Library のトレースファイル運用時の留意事項について説明します。

(1)　拡張データベースアクセストレースファイル

拡張データベースアクセストレースファイル運用時の注意事項を次に示します。

- 拡張データベースアクセストレースでは，DABroker Library 動作環境定義ファイルで設定した行数を超える出力があった場合，現在のトレースファイルがバックアップファイルとして別名で保存されます。トレース内容は，トレースファイルの先頭から再び出力されます。
- バックアップファイルは 1 世代前までしか残りません。このため，すでにバックアップファイルがある場合に，現在のトレースファイルが設定した行数を超えると，既存のバックアップファイルが上書きされます。複数世代のバックアップファイルを保存する必要がある場合は，あらかじめ別のディレクトリに退避してください。
- トレースファイルは実行しているコネクションの数だけ作成されます。
- トレースファイルは自動的に削除されません。ユーザが不要になった時点で削除してください。
- ハードディスクの空き容量が十分でない場合，トレースの出力が停止されることがあります。

(2)　エラートレースファイル

エラートレースファイル運用時の注意事項を次に示します。

- dabevttrcerr1.txt ファイルのサイズが 512 キロバイトを超えている場合にログを出力すると，dabevttrcerr1.txt ファイルをバックアップ用ファイル（dabevttrcerr2.txt）に名称を変更したあと，新規作成した dabevttrcerr1.txt ファイルにログを出力します。
- バックアップ用ファイルは 1 世代前までだけ残せます。バックアップファイルを保存する必要がある場合は，あらかじめ別のディスクに退避してください。
- ハードディスクの空き容量が十分でない場合，ログの出力を停止します。

(3)　Exception トレースログのファイル

Exception トレースログのファイル運用時の注意事項を次に示します。

- Exception トレースログのファイルは 1 世代前まで保持されます。ファイル出力時にすでにカレントファイルと 1 世代前のファイルが存在する場合は，1 世代前のファイルが削除され，カレントファイルが 1 世代前のファイルに置き換えられます。
- ファイルの出力先は，環境変数 DAB_FILEDIR で設定できます。

  デフォルトは，<DABroker Library 運用ディレクトリ>¥spool（Windows の場合），または <DABroker Library 運用ディレクトリ>/spool（UNIX の場合）です。

  <ファイルの出力先ディレクトリ>¥J2EE サーバ名称+MTrc.logl（Windows の場合），または<ファイルの出力先ディレクトリ>/J2EE サーバ名称+MTrc.logl（UNIX の場合）のバイト数が 259 バイトを超えないように注意してください。また，<ファイルの出力先ディレクトリ>¥J2EE サーバ名称 +MTrcOLD.log（Windows の場合），または<ファイルの出力先ディレクトリ>/J2EE サーバ名称 +MTrcOLD.log（UNIX の場合）のバイト数が 259 バイトを超える場合も注意が必要です。

  259 バイトを超えた場合の動作は，次のようになります。

  - <ファイルの出力先ディレクトリ>¥J2EE サーバ名称+MTrc.log のバイト数が 259 バイトを超えた場合，ログファイル，および 1 世代前のファイルが作成されません。
  - <ファイルの出力先ディレクトリ>¥<J2EE サーバ名称+MTrc.log>のバイト数が 259 バイトを超えない場合に，<ファイルの出力先ディレクトリ>¥<J2EE サーバ名称+MTrcOLD.log>のバイト数が 259 バイトを超えたときは，ログファイルは作成されますが，1 世代前のファイルが作成されません。この場合，ファイル出力情報量を超えても 1 世代前のファイルが作成されないで，カレントファイルへ出力され続けます。
- ファイルの作成や出力に失敗した場合，ログファイルに情報が出力されません。このとき，J2EE アプリケーションへのエラーメッセージの返却や，リトライは実行されません。エラーになった要因が解決されると，ログファイルにファイルが出力されるようになります。

  ログファイル初回接続時にエラーが発生した場合に考えられる要因を次に示します。

  - ディレクトリが存在しない，またはディレクトリに対する権限がない
  - カレントファイルを 1 世代前のファイルに移動できない
  - 1 世代前のファイルを削除できない

  ログファイル出力時にエラーが発生した場合に考えられる要因を次に示します。

  - ディレクトリが存在しない，またはディレクトリに対する権限がない
  - カレントファイルを作成または出力できない
  - カレントファイルを 1 世代前のファイルに移動できない
  - 1 世代前のファイルを削除できない

(4)　JDBC ドライバのトレースファイル

JDBC ドライバのトレースファイル運用時の注意事項を次に示します。

- JDBC ドライバ内でトレース情報の数がエントリ数を超えた場合は，トレースの古い順にトレース情報が上書きされます。
- JDBC インタフェースメソッドトレース（キーワード：JDBC_IF）は，Entry および Return でそれぞれ 1 エントリのトレース領域を使用します。
- DABroker とのイベントトレース（キーワード：SV_EVENT）は，送信または受信イベントでそれぞれ 1 エントリのトレース領域を使用します。

4.6.7　データベースと接続中にトラブルが発生した場合

データベースとの接続中に，アプリケーションサーバの構成ソフトウェアである DABroker Library でトラブルが発生した場合，DABroker Library によってデータベースとの接続は切り離されます。また，DABroker Library を強制終了した場合も同様です。

このとき，次の処理が実行されます。

- DABroker Library が異常終了した場合
  データベースで実行中の処理はロールバックされます。
- DBMS が異常終了した場合
  データベースで実行中の処理は，DBMS の制御に従って処理されます。

トラブルへの対応後，必要に応じてデータベースに再接続してください。

また，データベースアクセス，DABroker Library または JDBC ドライバでエラーが発生した場合，例外として SQLException がスローされます。この SQLException オブジェクトの reason には，次の項目が設定されています。トラブルへの対処に利用してください。

- JDBC ドライバのメッセージ
- エラーを検知したクラス名称とメソッド名称
- DABroker Library からのメッセージ（メッセージがある場合）

なお，次のエラーが発生した場合は，SQLException はスローされません。

- Driver.connection メソッドでデータベースに接続できない場合
- URL 指定で null が返却される場合
  DABroker Library で対象にしていない URL が指定された場合などが該当します。

この場合は，JDBC インタフェースの仕様で決まっている処理が実行されます。

なお，データベースとして HiRDB，または Oracle を利用している場合，性能解析トレースファイルに出力されたコネクション ID と，DABroker Library や HiRDB クライアント，または Oracle クライアントのログファイルやトレースファイルに出力されたコネクション ID を突き合わせて確認することで，トラブルが発生したコネクションを確認できます。確認方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「付録 B　データベースと接続中にトラブルが発生したコネクションの特定」を参照してください。

! 注意事項

07-01 以前のバージョンの HiRDB に接続していて，グローバルトランザクションとローカルトランザクションが混在している場合，性能解析トレースに出力されるコネクション ID と実際の接続状態に不整合が生じます。

例えば，コネクションの障害検知機能を無効にして，次のようなグローバルトランザクションとローカルトランザクションが混在している処理を実行する場合，6.～8.および 9.～13.の処理で性能解析トレースに出力される情報と実際の接続状態に不整合が生じます。

```
1. DataSource.getConnection()    : xa_open
2. UserTransaction.begin()       : xa_start
3. SQL実行                       --- グローバルトランザクション
4. UserTransaction.commit()      : xa_end
5. Connection.close()

6. DataSource.getConnection()
7. SQL実行                       --- ローカルトランザクション
8. Connection.close()

9. DataSource.getConnection()    : xa_open
10. UserTransaction.begin()      : xa_start
11. SQL実行                      --- グローバルトランザクション
12. UserTransaction.commit()     : xa_end
13. Connection.close()
```

- 6.～8.の処理では，ローカルトランザクションのコネクション ID は出力されずに，グローバルトランザクションのコネクション ID が出力されます。
- 9.～13.の処理では，グローバルトランザクションのコネクション ID は出力されずに，ローカルトランザクションのコネクション ID が出力されます。

このように，ローカルトランザクションのコネクション ID や，ローカルトランザクション実行後のグローバルトランザクションのコネクション ID を追跡することはできません。なお，7.のローカルトランザクションの処理で，コネクションが切り替わりますが，アプリケーションサーバには通知されません。

グローバルトランザクションとローカルトランザクションの混在は，使用する DBConnector のトランザクションサポートレベルが XATransaction の場合に，トランザクションを開始する前にコネクションを取得して SQL を実行したときにも発生します。

4.7　DABroker Library を使用したデータベース接続の実装

ここでは，DABroker Library が提供する機能を使用してデータベース接続をする場合の実装方法や注意事項について説明します。

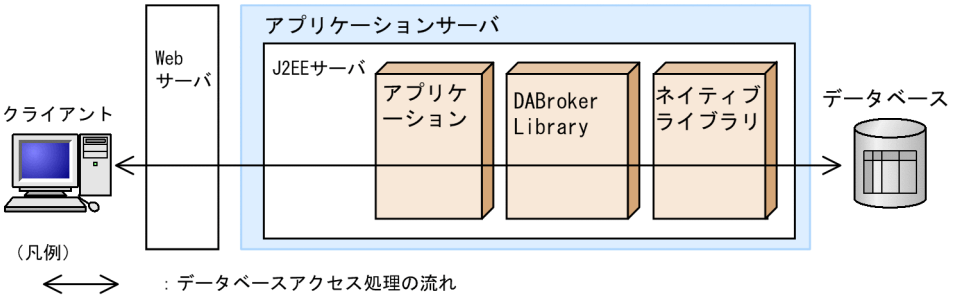
4.7.1　DABroker Library で開発するアプリケーション

DABroker Library は，JDBC ドライバと DABroker のデータベース接続部品から構成されたデータベースに接続するミドルウェアです。

DABroker Library を利用して開発したアプリケーションは，JDK が対応する WWW ブラウザで動作します。開発したアプリケーションを組み込むマシンには，DABroker Library が提供する JDBC ドライバが必要です。DABroker Library はサーバで動作するアプリケーションだけサポートしています。

アプリケーションサーバを利用したデータベース接続の流れを次の図に示します。

図 4-4　アプリケーションサーバを利用したデータベース接続



DABroker Library で開発したアプリケーションは，アプリケーションサーバに組み込んで使用します。アプリケーションからの要求は，アプリケーションサーバ上にある JDBC ドライバを経由し，DABroker Library で処理されます。

アプリケーションサーバのアプリケーションの動作時には，DABroker Library で提供するネイティブライブラリがアプリケーションサーバ上に動的にロードされます。ネイティブライブラリがロードされると，Java 標準のネイティブライブラリインタフェースである，JNI を使用してアプリケーションと DABroker Library を接続します。ネイティブライブラリを使用することで，アプリケーションと DABroker Library との接続処理を高速化し，高性能なデータベース接続を実現します。また，同一マシン上にアプリケーション，JDBC ドライバ，DABroker Library があるため，通信処理のオーバーヘッドがなく，Java アプレットやクライアントで動作するアプリケーションに比べて高性能な処理を実現します。

4.7.2　JDBC2.0 の概要

ここでは，DABroker Library で提供する，JDBC2.0 基本規格に沿った機能の概要と注意事項について説明します。JDBC1.0 が提供するクラスとメソッドの機能，および JDBC2.0 で規格化されていない DBP ドライバだけの機能については，マニュアル「アプリケーションサーバ リファレンス API 編」を参照してください。

(1)　ResultSet の拡張

JDBC2.0 基本規格では，ResultSet（ResultSet クラス）の拡張機能として，「スクロール」と「並行処理」が追加されています。

(a)　スクロールタイプ

ResultSet のスクロールタイプには，次の 3 種類があります。

- **順方向専用型**
  JDBC1.0 からの標準のスクロールタイプです。ResultSet 内を，順方向（上から下）にだけスクロールできます。
- **スクロール非反映型**
  JDBC2.0 で導入されたスクロールタイプです。ResultSet 内を，順方向，逆方向にスクロールできます。また，現在の位置からの相対位置指定の移動，絶対位置への移動もできます。
  「非反映型」とは，ResultSet が開かれている間に加えられた変更が，その ResultSet に反映されないことを意味します。つまり，基盤となるデータの静的なビューを提供するだけで，ResultSet に含まれる行，その順序，および列の値は，ResultSet の作成時に固定されます。
- **スクロール反映型**
  JDBC2.0 で導入されたスクロールタイプです。DABroker Library ではサポートしていません。
  ResultSet が開かれている間に加えられた変更が，その ResultSet に反映される点がスクロール非反映型とは異なります。
  「変更の反映」という点では，「その ResultSet 自身による変更の反映」，「同一トランザクション内での，ほかの ResultSet での変更の反映」，「ほかのトランザクションによる変更の反映」などがありますが，どこまで保証するかはドライバの実装レベル，および DBMS のトランザクション遮断レベルに依存します。

(b)　並行処理タイプ

ResultSet の並行処理タイプには，次の種類があります。

- **読み取り専用型**
  JDBC1.0 からの標準の並行処理タイプです。ResultSet からの更新はできません。
- **更新可能型**
  JDBC2.0 で導入された並行処理タイプです。ResultSet からの更新（UPDATE，INSERT，DELETE）ができます。DABroker Library ではサポートしていません。

(c)　ResultSet タイプ

スクロールタイプと並行処理タイプを組み合わせることで，ResultSet には 6 種類のタイプがあります。DABroker Library では，このうち 2 種類をサポートしています。ResultSet タイプは，Connection クラスの createStatement メソッド，prepareStatement メソッド，または prepareCall メソッドで，Statement クラス（またはそのサブクラス）のインスタンスを取得する時に指定します。

6 種類の ResultSet タイプと DABroker Library でのサポートを，次の表に示します。

表 4-35　ResultSet のタイプ

ResultSet タイプ		DABroker Library でのサポート
スクロールタイプ	並行処理タイプ	
順方向専用型	読み取り専用型	○
	更新可能型	×
スクロール非反映型	読み取り専用型	○
	更新可能型	×
スクロール反映型	読み取り専用型	×
	更新可能型	×

（凡例）

○：サポートしている

×：サポートしていない

サポートしていない ResultSet タイプが指定された場合，エラーにはしないで，指定された ResultSet タイプにいちばん近い ResultSet タイプを使用して Statement クラス（またはそのサブクラス）のインスタンスを生成します。また，警告メッセージを Connection クラスの SQLWarning に格納します。

更新可能型の並行処理タイプはサポートしていないため，ResultSet クラスに使用できないメソッドがあります。これらの使用できないメソッドが呼び出された場合は，無条件に SQLException を throw します。どのメソッドが使用できないかについては，「4.16　ResultSet クラス」を参照してください。

(d)　スクロール型 ResultSet 使用時の注意点

スクロール型 ResultSet では，DABroker Library 内ですべての検索データをキャッシングします。データ量が大量な場合は，メモリ不足や性能劣化が発生する可能性が大きくなります。このため，スクロール型 ResultSet を使用する場合は，「SQL に条件を付加する」，「Statement クラスの setMaxRows メソッドを呼び出す」など，検索データ量をあらかじめ抑制してください。

(2)　バッチ更新

JDBC2.0 基本規格では，Statement クラス，PreparedStatement クラス，および CallableStatement クラスにバッチ更新機能が追加されています。バッチ更新機能を使うと，複数の SQL，または複数のパラメタ値を登録し，一括で実行できます。

注意

バッチ更新機能を使用する場合，Connection クラスの AUTO コミットモードを OFF にしてください。これは，バッチ更新の途中でエラーが発生した場合，そのトランザクションの有効・無効をアプリケーション側で制御する必要があるためです。AUTO コミットモードが ON（初期状態）の場合，バッチ更新の途中でエラーが発生しても，エラーが発生する一つ前までの SQL 実行は有効となるため，注意が必要です。アプリケーションサーバで JTA トランザクションを使用している場合，AUTO コミットモードは OFF になります。

(a)　Statement クラスでのバッチ更新

概要

- 複数の更新系 SQL を addBatch メソッドで登録します。

- 登録した更新系 SQL を，executeBatch メソッドを使用して一括で実行します。
- 一括実行の結果として，それぞれの更新系 SQL によって更新された行数の配列を返します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を throw します。
- 登録した SQL 中に検索系 SQL がある場合，executeBatch メソッド呼び出し時に BatchUpdateException を throw します。

注意事項

DABroker とのインタフェース上，複数の SQL を一括実行できないため，DABroker Library では，登録された SQL を逐次実行することでバッチ更新を実現しています。

(b)　PreparedStatement クラスでのバッチ更新

概要

- PreparedStatement インスタンス生成時に指定した，更新系 SQL に対する？パラメタを，通常の手順（setXXX メソッド）で設定します。
- addBatch メソッドで，？パラメタのセットを登録します。
- 登録した複数セットの？パラメタを，executeBatch メソッドで一括実行します。
- 一括実行の結果として，それぞれの？パラメタのセットで更新された行数の配列を返します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を throw します。
- PreparedStatement インスタンス生成時に指定した SQL が検索系 SQL である場合，executeBatch メソッド呼び出し時に BatchUpdateException を throw します。

注意事項

定義系および制御系 SQL については，addBatch メソッドを実行しないでください。

(c)　CallableStatement クラスでのバッチ更新

概要

- CallableStatement インスタンス生成時に指定した，SQL に対する？パラメタを，通常の手順（setXXX メソッド）で設定します。
- addBatch メソッドで，入力パラメタのセットを登録します。
- 登録した複数セットの入力パラメタを，executeBatch メソッドで一括実行します。
- 一括実行の結果として，それぞれの入力パラメタのセットで実行された，SQL の返却値（更新行数）の配列を返します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を throw します。
- CallableStatement インスタンス生成時に指定した SQL が更新行数を返さない場合，および出力パラメタか入出力パラメタを持つ場合，executeBatch メソッド呼び出し時に BatchUpdateException を throw します。

注意事項

定義系および制御系 SQL については，addBatch メソッドを実行しないでください。

(d)　HiRDB の場合のバッチ更新

HiRDB の場合のバッチ更新処理は，次の 2 種類に分けられます。

- 逐次実行
  複数の SQL に分けて実行する更新処理です。

- 一括実行
  HiRDB の配列更新機能を使用して実行する更新処理です。

HiRDB の配列更新機能とは，データベースにより提供される次の機能を指します。

- HiRDB の配列を使用した INSERT 機能
- HiRDB の配列を使用した UPDATE 機能
- HiRDB の配列を使用した DELETE 機能
- HiRDB クライアント経由 XDM／RD E2 で使用する XDM／RD E2 の配列を使用した INSERT 機能
- HiRDB クライアント経由 XDM／RD E2 で使用する XDM／RD E2 の配列を使用した UPDATE 機能
- HiRDB クライアント経由 XDM／RD E2 で使用する XDM／RD E2 の配列を使用した DELETE 機能

HiRDB の配列更新機能では，複数の?パラメタセットを配列形式で扱い，一度に処理することができます。そのため，データベースとの通信回数が減り，大量のデータを高速に更新できます。

HiRDB の配列更新機能を使用するには，BlockUpdate 設定を true にします。ただし，接続する HiRDB のバージョンによって使用できない場合があります。HiRDB の配列更新機能の詳細については，マニュアル「HiRDB UAP 開発ガイド」およびマニュアル「HiRDB SQL リファレンス」を参照してください。

● addBatch メソッド－executeBatch メソッド使用時の仕様

次に，バッチ更新時に使用する addBatch メソッド－executeBatch メソッド使用時の仕様を，BlockUpdate 設定が true の場合と false の場合に分けて示します。

表 4–36　addBatch メソッド－executeBatch メソッド使用時の仕様（BlockUpdate 設定が true，HiRDB の場合）

実行する SQL 文	DABroker の仕様（処理方法）			
	HiRDB Version 6	HiRDB Version 7 07-00 以降かつ 07-00-/K 以前	HiRDB Version 7 07-00-/L 以降かつ HiRDB Version 7 07-01 未満	HiRDB Version 7 07-01 以降
INSERT 文（VALUES 句指定）	逐次実行	一括実行	一括実行	一括実行
INSERT 文（SELECT 句指定）	逐次実行	KFPA19408-E	逐次実行	一括実行
UPDATE 文	逐次実行	KFPA19408-E	逐次実行	一括実行
DELETE 文	逐次実行	KFPA19408-E	逐次実行	一括実行
CALL 文（OUT または INOUT パラメタあり）	KFDJ00010-E	KFDJ00010-E	KFDJ00010-E	KFDJ00010-E
CALL 文（IN パラメタだけ）	逐次実行	逐次実行	逐次実行	逐次実行

（凡例）
一括実行：HiRDB の配列更新機能を使用して SQL を一括実行する
逐次実行：HiRDB の配列更新機能を使用しないで SQL を分けて実行する

KFPA19408-E：エラー（KFPA19408-E）が発生する。HiRDB の配列更新機能は使用できない

KFDJ00010-E：エラー（KFDJ00010-E）が発生する。JDBC2.0 規格では，executeBatch メソッドで更新行数を返さないもの（SELECT など）が含まれていた場合，例外となる

表 4-37　addBatch メソッド－executeBatch メソッド使用時の仕様（BlockUpdate 設定が false または指定なし，HiRDB の場合）

実行する SQL 文	DABroker の仕様（処理方法）
INSERT 文（VALUES 句指定）	逐次実行
INSERT 文（SELECT 句指定）	逐次実行
UPDATE 文	逐次実行
DELETE 文	逐次実行
SELECT 文	KFDJ00010-E
CALL 文（OUT または INOUT パラメタあり）	KFDJ00010-E
CALL 文（IN パラメタだけ）	逐次実行

（凡例）

逐次実行：HiRDB の配列更新機能を使用しないで SQL を分けて実行する

KFDJ00010-E：エラー（KFDJ00010-E）が発生する。JDBC2.0 規格では，executeBatch メソッドで更新行数を返さないもの（SELECT など）が含まれていた場合，例外となる

表 4-38　addBatch メソッド－executeBatch メソッド使用時の仕様（BlockUpdate 設定が true，HiRDB クライアント経由 XDM/RD E2 の場合）

実行する SQL	DABroker の仕様（処理方式）		
	HiRDB クライアント 08-02 未満	HiRDB クライアント 08-02 以降	
		XDM／RD E2 (11-03 未満)	XDM／RD E2 (11-03 以降)
INSERT 文（VALUES 句指定）	逐次実行	逐次実行	一括実行
INSERT 文（SELECT 句指定）	逐次実行	逐次実行	一括実行
UPDATE 文	逐次実行	逐次実行	一括実行
DELETE 文	逐次実行	逐次実行	一括実行
SELECT 文	KFDJ00010-E	KFDJ00010-E	KFDJ00010-E
CALL 文（OUT，または INOUT パラメタ有）	KFDJ00010-E	KFDJ00010-E	KFDJ00010-E
CALL 文（IN パラメタだけ）	逐次実行	逐次実行	逐次実行

（凡例）

一括実行：HiRDB の配列更新機能を使用して SQL を一括実行する

逐次実行：HiRDB の配列更新機能を使用しないで SQL を分けて実行する

KFDJ00010-E：エラー（KFDJ00010-E）が発生する。JDBC2.0 規格では，executeBatch メソッドで更新行数を返さないもの（SELECT など）が含まれている場合，例外となる

表 4-39　addBatch メソッド－executeBatch メソッド使用時の仕様（BlockUpdate 設定が false または指定なし，HiRDB クライアント経由 XDM/RD E2 の場合）

実行する SQL 文	DABroker の仕様
INSERT 文（VALUES 句指定）	逐次実行
INSERT 文（SELECT 句指定）	逐次実行
UPDATE 文	逐次実行
DELETE 文	逐次実行
SELECT 文	KFDJ00010-E
CALL 文（OUT，または INOUT パラメタあり）	KFDJ00010-E
CALL 文（IN パラメタだけ）	逐次実行

（凡例）

逐次実行：HiRDB の配列更新機能を使用しないで SQL を分けて実行する

KFDJ00010-E：エラー（KFDJ00010-E）が発生する。JDBC2.0 規格では，executeBatch メソッドで更新行数を返さないもの（SELECT など）が含まれている場合，例外となる

● HiRDB の配列更新機能の使用方法

HiRDB の配列更新機能の使用方法を次に示します。

- BlockUpdate 設定を true にします。
  Connect 時に，プロパティとして BLOCK_UPDATE=TRUE を指定するか（DataSource を使用する場合は setBlockUpdate(true)），JdbcDbpsvPreparedStatement の setBlockUpdate(true)を指定します。
- パラメタが一つ以上ある SQL を前処理します。
- PreparedStatement クラス，または CallableStatement クラスの set メソッドで，?パラメタに対する値を設定します。
- addBatch メソッドで，?パラメタセットを追加します。
- executeBatch メソッドで，バッチ更新を実行します。

● HiRDB の配列更新機能使用時の注意事項

- 共通項目
  サポートする HiRDB 全バージョン共通の注意事項を次に示します。
  - HiRDB の配列更新機能を使用するには，接続時のプロパティ情報，またはデータソースの BlockUpdate を true で設定するか，PreparedStatement の setBlockUpdate メソッドを true に設定する必要があります。
  - HiRDB の配列更新機能は，?パラメタを持つ SQL 文に対して有効です。プロシジャや Statement クラスの addBatch(String sql)を使用した場合は有効となりません。
  - パラメタセットを登録する場合，PreparedStatement，または CallableStatement の addBatch メソッドを使用してください。Statement の addBatch(String sql)を使用するとエラーになります。
  - HiRDB の配列更新機能は，addBatch メソッドで登録したパラメタセットが 2 件以上ある場合に有効です。1 件の場合は HiRDB の配列更新機能を使用しないで実行されます。

- 実行する SQL には，?パラメタが一つ以上必要です。?パラメタがない場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。
- XA 環境下の HiRDB に接続してバッチ更新を実行する場合，setAutoCommit(false)を設定する必要があります。setAutoCommit(false)を設定しないで実行すると HiRDB でエラーになります。
- DECIMAL 型データをパラメタセットに設定する場合，配列に指定する DECIMAL 型データの精度，および位取りは HiRDB の表定義属性に合わせられます。整数部のけた数が HiRDB の表定義の整数部のけた数より大きいと，けたあふれが発生してエラーになります。
- SMALLINT 型，INTEGER 型，または DECIMAL 型の列に，setInt，または setString を使用して範囲外のデータを設定するとエラーになります。
- 配列更新の途中でエラーが発生した場合，エラーが発生する直前までの実行もすべて無効になります。
- ?パラメタに HiRDB の繰り返し列（複数要素を格納できる列）を指定している場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。
- setNull メソッドで，HiRDB の表定義の列属性が CHAR，NCHAR，MCHAR，DATE，TIME，または TIMESTAMP の場合，列属性と同じ java.sql.Types を指定していると，HiRDB の配列更新機能は使用されないで，逐次実行されます。
- setNull メソッドで，HiRDB の表定義の列属性と異なる java.sql.Types を指定している場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。ただし，次の場合は，HiRDB の配列更新機能が実行されます。

  ・HiRDB の表定義の列属性が CHAR，NCHAR，または MCHAR で，java.sql.Types に java.sqlTypes.VARCHAR を指定した場合

  ・HiRDB の表定義の列属性が DATE，TIME，または TIMESTAMP で，java.sql.Types に java.sql.Types.CHAR を指定した場合

- HiRDB のバージョンが 07-00 の場合の注意事項

  HiRDB のバージョンが 07-00 の場合，次に示すバージョン固有の注意事項があります。共通項目と合わせて注意してください。

  - HiRDB のバージョンが 07-00 の場合，使用できる HiRDB の配列更新機能は，「配列を使用した INSERT 機能（INSERT VALUE 句だけ）」です。「配列を使用した INSERT 機能（INSERT VALUE 句以外）」，「配列を使用した UPDATE 機能」，および「配列を使用した DELETE 機能」は使用できません。
  - ?パラメタに HiRDB の BLOB 型の列，または定義長 32,001 バイト以上の BINARY 型の列を指定している場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。
  - WITH DEFAULT 指定の BLOB 型の列，または WITH DEFAULT 指定で定義長 32,001 バイト以上の BINARY 型の列の場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。
  - HiRDB の行単位トリガを定義した表に対して更新する場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。

- HiRDB のバージョンが 07-01 以降の場合の注意事項

  HiRDB のバージョンが 07-01 以降の場合，次に示すバージョン固有の注意事項があります。共通項目と合わせて注意してください。

  - 「配列を使用した INSERT 機能」，「配列を使用した UPDATE 機能」，および「配列を使用した DELETE 機能」が使用できます。
  - ?パラメタにデータ長が 32,001 バイト以上の BINARY データを指定している場合，HiRDB の配列更新機能は使用されないで，逐次実行されます。

- HiRDB の BLOB 型列に対して 32,001 バイト以上のデータを指定している場合，HiRDB の配列更新機能は使用され，逐次実行されます。

(e)　Oracle の場合のバッチ更新

ここでは，Oracle の場合のバッチ更新について説明します。

● addBatch メソッド－executeBatch メソッド使用時の仕様

次に，PreparedStatement および CallableStatement での addBatch メソッド－executeBatch メソッド使用時の仕様を，BlockUpdate 設定が true の場合，false の場合に分けて示します。

表 4-40　addBatch メソッド－executeBatch メソッド使用時の仕様（BlockUpdate 設定が true，Oracle の場合）

実行する SQL 文	DABroker の仕様（処理方法）
INSERT 文（VALUES 句指定）	逐次実行
INSERT 文（SELECT 句指定）	逐次実行
UPDATE 文	逐次実行
DELETE 文	逐次実行
SELECT 文	KFDJ00010-E
BEGIN 文（OUT または INOUT パラメタあり）	KFDJ00010-E
BEGIN 文（IN パラメタだけ）	逐次実行

（凡例）

逐次実行：SQL を分けて実行する

KFDJ00010-E：エラー（KFDJ00010-E）が発生する。JDBC2.0 規格では，executeBatch メソッドで更新行数を返さないもの（SELECT など）が含まれていた場合，例外となる

表 4-41　addBatch メソッド－executeBatch メソッド使用時の仕様（BlockUpdate 設定が false または指定なし，Oracle の場合）

実行する SQL 文	DABroker の仕様（処理方法）
INSERT 文（VALUES 句指定）	逐次実行
INSERT 文（SELECT 句指定）	逐次実行
UPDATE 文	逐次実行
DELETE 文	逐次実行
SELECT 文	KFDJ00010-E
BEGIN 文（OUT または INOUT パラメタあり）	KFDJ00010-E
BEGIN 文（IN パラメタだけ）	逐次実行

（凡例）

逐次実行：SQL を分けて実行する

KFDJ00010-E：エラー（KFDJ00010-E）が発生する。JDBC2.0 規格では，executeBatch メソッドで更新行数を返さないもの（SELECT など）が含まれていた場合，例外となる

● addBatch メソッド使用時の注意事項

- 実行する SQL に?パラメタを一つ以上含めてください。
- 配列に指定するデータ型が異なる場合にはエラーとなります。
- アプリケーションサーバから使用する場合は，PreparedStatement クラスの setBlockUpdate メソッドは使用できません。

(3)　JDBC SQL タイプの扱い

JDBC2.0 基本規格では，幾つかの新しい JDBC SQL タイプが追加されています。追加された JDBC SQL タイプと DABroker Library でのサポートを，次の表に示します。

表 4-42　追加された JDBC SQL タイプと DABroker Library でのサポート

追加された JDBC SQL タイプ	DABroker Library でのサポート
BLOB	○
CLOB	△
ARRAY	×
REF	×
DISTINCT	×
STRUCT	×
JAVA OBJECT	×

（凡例）

○：Oracle9i，Oracle10g，および HiRDB でサポートしている

△：Oracle9i，および Oracle10g でだけサポートしている

×：サポートしていない

(a)　検索データ取得時のデータ変換

追加された JDBC SQL タイプと，ResultSet クラスの getXXX メソッドとのデータ変換について，次の表に示します。変換できない JDBC SQL タイプに対して getXXX メソッドが呼び出された場合は，SQLException を throw します。接続データベースがどの JDBC SQL タイプをサポートしているかは，「4.7.7　データ型の対応」を参照してください。

なお，JDBC2.0 規格では getUnicodeStream メソッドが推奨されないメソッドとなったため，代わりに getCharacterStream が追加されています。

表 4-43　ResultSet クラスの getXXX メソッドと JDBC SQL タイプとのデータ変換 （1）

getXXX メソッド	JDBC SQL タイプ												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
getByte	○	◎	○	○	○	○	○	○	○	○	○	○	○
getShort	○	○	◎	○	○	○	○	○	○	○	○	○	○
getInt	○	○	○	◎	○	○	○	○	○	○	○	○	○
getLong	○	○	○	○	◎	○	○	○	○	○	○	○	○
getFloat	○	○	○	○	○	○	◎	○	○	○	○	○	○
getDouble	○	○	○	○	○	◎	○	◎	○	○	○	○	○
getBigDecimal	○	○	○	○	○	○	○	○	◎	◎	■	○	○
getBoolean	◎	○	○	○	○	○	○	○	○	○	○	○	○
getString	○	○	○	○	○	○	○	○	○	○	◎	◎	○
getBytes	−	−	−	−	−	−	−	−	−	−	−	−	−
getDate※1	−	−	−	−	−	−	−	−	−	−	○	○	○
getTime※1	−	−	−	−	−	−	−	−	−	−	○	○	○
getTimestamp※1	−	−	−	−	−	−	−	−	−	−	○	○	○
getAsciiStream	−	−	−	−	−	−	−	−	−	−	○	○	◎
getUnicodeStream	−	−	−	−	−	−	−	−	−	−	○	○	◎
getBinaryStream	−	−	−	−	−	−	−	−	−	−	−	−	−
getObject	○	○	○	○	○	○	○	○	○	○	○	○	○
getCharacter Stream	−	−	−	−	−	−	−	−	−	−	○	○	◎
getArray （未サポート）	−	−	−	−	−	−	−	−	−	−	−	−	−
getBlob※2	−	−	−	−	−	−	−	−	−	−	−	−	−
getClob※3	−	−	−	−	−	−	−	−	−	−	−	−	−

getXXX メソッド	JDBC SQL タイプ												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
getRef （未サポート）	－	－	－	－	－	－	－	－	－	－	－	－	－

（凡例）

◎：完全に変換できる

○：変換できる。ただし，変換元のデータの内容によってはデータの欠落や変換エラーとなることがある

■：設定値が定義長に満たない場合，エラーが発生する

－：変換できない

注※1

引数に java.util.Calendar オブジェクトを指定する getDate，getTime および getTimestamp メソッドを実行すると，データベースに格納されたデータが指定した java.util.Calendar オブジェクトのタイムゾーンであるとして，実行しているアプリケーションのデフォルトのタイムゾーンに変換して取得します。

注※2

Oracle9i，Oracle10g，および HiRDB でサポートしています。

注※3

Oracle9i，および Oracle10g でサポートしています。

表 4-44　ResultSet クラスの getXXX メソッドと JDBC SQL タイプとのデータ変換 （2）

getXXX メソッド	JDBC SQL タイプ											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVAOBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
getByte	−	−	−	−	−	−	−	−	−	−	−	−
getShort	−	−	−	−	−	−	−	−	−	−	−	−
getInt	−	−	−	−	−	−	−	−	−	−	−	−
getLong	−	−	−	−	−	−	−	−	−	−	−	−
getFloat	−	−	−	−	−	−	−	−	−	−	−	−
getDouble	−	−	−	−	−	−	−	−	−	−	−	−
getBigDecimal	−	−	−	−	−	−	−	−	−	−	−	−
getBoolean	−	−	−	−	−	−	−	−	−	−	−	−
getString	○	○	○	○	○	○	−	−	−	−	−	−
getBytes	−	−	−	◎	◎	○	−	−	−	−	−	−
getDate※1	◎	−	○	−	−	−	−	−	−	−	−	−
getTime※1	−	◎	○	−	−	−	−	−	−	−	−	−
getTimestamp※1	○	−	◎	−	−	−	−	−	−	−	−	−
getAsciiStream	−	−	−	○	○	○	−	−	−	−	−	−
getUnicodeStream	−	−	−	○	○	○	−	−	−	−	−	−
getBinaryStream	−	−	−	○	○	◎	−	−	−	−	−	−
getObject	○	○	○	○	○	○	▲	▲	△	▼	▼	△
getCharacter Stream	−	−	−	○	○	○	−	−	−	−	−	−
getArray （未サポート）	−	−	−	−	−	−	−	−	▲	−	−	−

getXXX メソッド	JDBC SQL タイプ											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVAOBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
getBlob※2	－	－	－	－	－	○※4	－	－	－	●	－	－
getClob※3	－	－	－	－	－	－	－	－	－	－	●	－
getRef（未サポート）	－	－	－	－	－	－	－	－	－	－	－	▲

（凡例）

◎：完全に変換できる

○：変換できる。ただし，変換元のデータの内容によってはデータの欠落や変換エラーとなることがある

●：完全に変換できる組み合わせは，接続するデータベースが Oracle9i または Oracle10g の場合以外では，JDBC SQL タイプが未サポートのため使えない

▼：変換できる組み合わせは，接続するデータベースが Oracle9i または Oracle10g の場合以外では，JDBC SQL タイプが未サポートのため使えない

▲：完全に変換できる組み合わせは，JDBC SQL タイプが未サポートのため使えない

△：変換できる組み合わせは，JDBC SQL タイプが未サポートのため使えない

－：変換できない

注※1

引数に java.util.Calendar オブジェクトを指定する getDate，getTime および getTimestamp メソッドを実行すると，データベースに格納されたデータが指定した java.util.Calendar オブジェクトのタイムゾーンであるとして，実行しているアプリケーションのデフォルトのタイムゾーンに変換して取得します。

注※2

Oracle9i，Oracle10g，および HiRDB でサポートしています。

注※3

Oracle9i，および Oracle10g でサポートしています。

注※4

HiRDB でだけサポートしています。

(b)　？パラメタ設定時のデータ変換

追加された JDBC SQL タイプと，PreparedStatement クラスの setXXX メソッドとのデータ変換について，次の表に示します。JDBC SQL タイプをサポートしていない場合，setXXX メソッドは SQLException を throw します。接続データベースがどの JDBC SQL タイプをサポートしているかは，「4.7.7　データ型の対応」を参照してください。

なお，JDBC2.0 規格では，setUnicodeStream メソッドが推奨されないメソッドとなったため，代わりに setCharacterStream が追加されています。

表 4-45　追加された JDBC SQL タイプと setXXX メソッド

setXXX メソッド	データ変換される JDBC SQL タイプ
setCharacterStream	CHAR，VARCHAR，または LONGVARCHAR
setRef（未サポート）	REF
setBlob※1	BLOB，LONGVARBINARY
setClob※2	CLOB
setArray（未サポート）	ARRAY

注※1

Oracle9i，Oracle10g，および HiRDB でサポートしています。

注※2

Oracle9i，および Oracle10g でサポートしています。

表 4-46　setXXX メソッドと JDBC SQL タイプとのデータ変換 （1）

setXXX メソッド	JDBC SQL タイプ												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
setByte	○	◎	○	○	○	○	○	○	○	○	○	○	○
setShort	○	○	◎	○	○	○	○	○	○	○	○	○	○
setInt	○	○	○	◎	○	○	○	○	○	○	○	○	○
setLong	○	○	○	○	◎	○	○	○	○	○	○	○	○
setFloat	○	○	○	○	○	○	◎	○	○	○	○	○	○
setDouble	○	○	○	○	○	◎	○	◎	○	○	○	○	○

setXXX メソッド	JDBC SQL タイプ												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
setBigDecimal	○	○	○	○	○	○	○	○	◎	◎	○	○	○
setBoolean	◎	○	○	○	○	○	○	○	○	○	○	○	○
setString	○	○	○	○	○	○	○	○	○	○	◎	◎	○
setBytes	−	−	−	−	−	−	−	−	−	−	−	−	−
setDate	−	−	−	−	−	−	−	−	−	−	○	○	○
setTime	−	−	−	−	−	−	−	−	−	−	○	○	○
setTimestamp	−	−	−	−	−	−	−	−	−	−	○	○	○
setAsciiStream	−	−	−	−	−	−	−	−	−	−	○※4	○※4	○※4
setUnicodeStream※1	−	−	−	−	−	−	−	−	−	−	○	○	○
setBinaryStream	−	−	−	−	−	−	−	−	−	−	−	−	−
setObject	○	○	○	○	○	○	○	○	○	○	○	○	○
setCharacterStream	−	−	−	−	−	−	−	−	−	−	○※5	○※5	○※5
setArray （未サポート）	−	−	−	−	−	−	−	−	−	−	−	−	−
setBlob※2	−	−	−	−	−	−	−	−	−	−	−	−	−
setClob※3	−	−	−	−	−	−	−	−	−	−	−	−	−
setRef （未サポート）	−	−	−	−	−	−	−	−	−	−	−	−	−

（凡例）

◎：完全に変換できる

○：変換できる。ただし，変換元のデータの内容によってはデータの欠落や変換エラーとなることがある

－：変換できない

注※1

Oracle9i，Oracle10g，および HiRDB でサポートしています。

注※2

Oracle9i，および Oracle10g でサポートしています。

注※3

長さが 0 のデータを設定するとエラーになります。

注※4

接続データベースが Oracle の場合，長さが 0 のデータを設定すると，NULL 値に変換します。

注※5

java.io.Reader オブジェクトから取得できるデータの長さが引数で指定した長さより短い場合，次のように引数で指定した長さまで 0 を補完します。

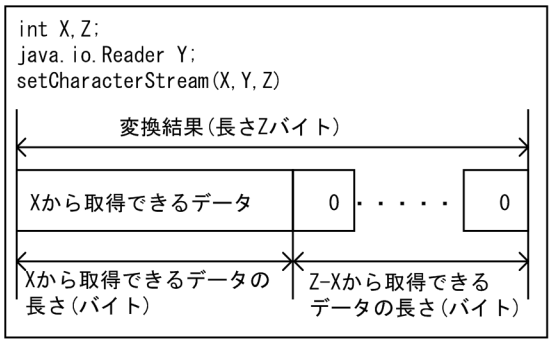


表 4-47　setXXX メソッドと JDBC SQL タイプとのデータ変換（2）

setXXX メソッド	JDBC SQL タイプ											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVAOBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
setByte	－	－	－	－	－	－	－	－	－	－	－	－
setShort	－	－	－	－	－	－	－	－	－	－	－	－
setInt	－	－	－	－	－	－	－	－	－	－	－	－
setLong	－	－	－	－	－	－	－	－	－	－	－	－
setFloat	－	－	－	－	－	－	－	－	－	－	－	－

setXXX メソッド	JDBC SQL タイプ											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVAOBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
setDouble	−	−	−	−	−	−	−	−	−	−	−	−
setBigDecimal	−	−	−	−	−	−	−	−	−	−	−	−
setBoolean	−	−	−	−	−	−	−	−	−	−	−	−
setString	○	○	○	○	○	○	−	−	−	−	−	−
setBytes	−	−	−	◎	◎	○	−	−	−	−	−	−
setDate	◎	−	○	−	−	−	−	−	−	−	−	−
setTime	−	◎	○	−	−	−	−	−	−	−	−	−
setTimestamp	○	−	◎	−	−	−	−	−	−	−	−	−
setAsciiStream	−	−	−	○	○※4	○※4	−	−	−	−	−	−
setUnicodeStream※3	−	−	−	○	○	○	−	−	−	−	−	−
setBinaryStream	−	−	−	○	○※4	○※4	−	−	−	−	−	−
setObject	○	○	○	○	○	○	▲	▲	△	▼	▼	△
setCharacterStream	−	−	−	○※5	○※5	○※5	−	−	−	−	−	−
setArray（未サポート）	−	−	−	−	−	−	−	−	▲	−	−	−
setBlob※1	−	−	−	−	−	○※1	−	−	−	●※2※4	−	−

setXXX メソッド	JDBC SQL タイプ											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVAOBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
setClob※2	－	－	－	－	－	－	－	－	－	－	●※4	－
setRef（未サポート）	－	－	－	－	－	－	－	－	－	－	－	▲

（凡例）

◎：完全に変換できる

○：変換できる。ただし，変換元のデータの内容によってはデータの欠落や変換エラーとなることがある

●：完全に変換できる組み合わせは，接続するデータベースが Oracle9i または Oracle10g の場合以外では，JDBC SQL タイプが未サポートのため使えない

▼：変換できる組み合わせは，接続するデータベースが Oracle9i または Oracle10g の場合以外では，JDBC SQL タイプが未サポートのため使えない

▲：完全に変換できる組み合わせは，JDBC SQL タイプが未サポートのため使えない

△：変換できる組み合わせは，JDBC SQL タイプが未サポートのため使えない

－：変換できない

注※1

Oracle9i，Oracle10g，および HiRDB でサポートしています。

注※2

Oracle9i，および Oracle10g でサポートしています。

注※3

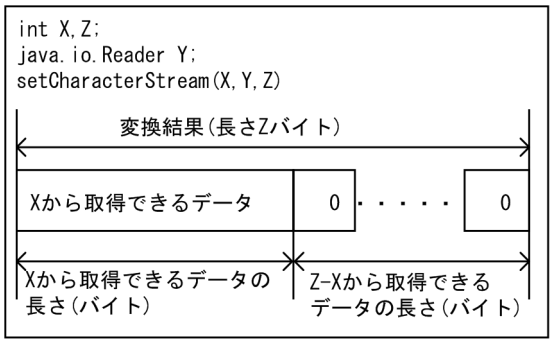
長さが 0 のデータを設定するとエラーになります。

注※4

接続データベースが Oracle である場合，長さが 0 であるデータを設定すると NULL 値に変換します。

注※5

java.io.Reader オブジェクトから取得できるデータの長さが引数で指定した長さより短い場合，次のように引数で指定した長さまで 0 を補完します。



4.7.3　JDBC2.0 拡張機能の概要

ここでは，DABroker Library で提供する JDBC2.0 拡張機能の概要を説明します。各メソッドでの制限事項については，「4.10　DABroker Library で使用する API の一覧」を参照してください。

(1)　JNDI 連携

JNDI 連携とは，DBMS との接続情報を JNDI に登録し，登録した接続情報を使用して DBMS と接続する機能です。一度接続情報を登録すれば，DBMS との接続時に接続情報を設定する必要がなくなります。
JNDI 連携の操作については，「4.7.5　JDBC2.0 拡張機能を使用したデータベース接続の設定」を参照してください。

(2)　接続プール

接続プール機能は，DataSource オブジェクトと JNDI を使用してデータベースと接続した場合だけ使用できます。

データベースとの接続をためる（プール）機能の概要について説明します。

(a)　概要

DBMS との接続をプールします。この機能は，DBMS との接続および解放によるオーバーヘッドを縮小し，サーバアプリケーションの性能向上に役立ちます。

- アプリケーションに影響を与えないため，接続プールの有無を意識する必要がない。
- システム構築時に，接続プール機能を任意に選択できる（ユーザが作成する，アプリケーションサーバのベンダが提供するものを使用する，または JDBC のベンダが提供するものを使用する）。
- 接続プール機能では，アプリケーションとのインタフェースとして，DataSource クラスを提供する。
- JDBC2.0 拡張機能では，接続プール機能とのインタフェースとして，ConnectionPoolDataSource クラスと PooledConnection クラスを提供する。
- ConnectionPoolDataSource クラスは，JDBC2.0 拡張機能が提供する DataSource クラスと同様に，接続情報の設定および取得用のメソッドを提供する。

(b)　関連するクラス

接続プール機能に関連するクラスを次の表に示します。

表 4-48　接続プールに関連するクラス

クラス	概要
javax.sql. ConnectionEventListener	• 接続プール機能が提供 • 接続プール機能で，接続の切断，SQL エラーなどをこのクラスのオブジェクトから検知※1
javax.sql.DataSource	• 接続プール機能が提供 • データベース接続時のアプリケーションとのインタフェース • 接続プールを制御 • JNDI に登録して使用
javax.sql. ConnectionPoolDataSource	• JDBC2.0 拡張機能が提供※2 • データベース接続に必要な接続情報の設定および取得用のメソッドを提供 • 接続プール機能が使用 • JNDI に登録して使用 • このクラスのオブジェクトから PooledConnection オブジェクトを取得
javax.sql.PooledConnection	• JDBC2.0 拡張機能が提供※2 • 接続プール機能が使用 • このクラスのオブジェクトがプール対象 • このクラスのオブジェクトから Connection オブジェクトを取得

注※1

connectionErrorOccurred が返されるのは次のエラーコードの場合です。

接続データベースが HiRDB：-720，-722，-723，-728，-732，-932，-1700

接続データベースが Oracle：28，1012，3113，3114，12571

注※2

JDBC2.0 拡張機能が提供するパッケージ名称とクラス名称を次に示します。

種類	名称
パッケージ	JP.co.Hitachi.soft.DBPSV_Driver
ConnectionPoolDataSource クラス	JdbcDbpsvConnectionPoolDataSource
PooledConnection クラス	JdbcDbpsvPooledConnection

(3)　分散トランザクション機能

分散トランザクション機能とは，Sun Microsystems の提唱する Java Transaction API（JTA）規格を基に，トランザクションマネージャと連携する，接続プール機能の拡張機能です。

分散トランザクション機能は，XADataSource クラスと JNDI を使用してデータベースと接続した場合だけ使用できます。

分散トランザクション機能の概要について説明します。

(a)　概要

JTS および JTA との連携によって，Sun Microsystems の提唱する Java Transaction API（JTA）規格に沿った，分散トランザクションを構築できます。

- システム構築時に，トランザクション連携機能を任意に選択できます（ユーザが作成する，APServer ベンダ提供のものを使用する，または JDBC ベンダが提供するものを使用する）。
- トランザクション連携機能は，接続プールの拡張機能として，トランザクションマネージャが提供する，JTA や JTS を使用します。
- トランザクション連携機能では，アプリケーションとのインタフェースとして，DataSource クラスを提供します。
- JDBC2.0 拡張機能では，トランザクション連携機能とのインタフェースとして，XADataSource クラスと XAConnection クラスを提供します。また，トランザクションマネージャとのインタフェースとして，XAResource クラスを提供します。
- XADataSource クラスは，JDBC2.0 拡張機能が提供する DataSource クラスと同様に，接続情報の設定および取得用のメソッドを提供します。

接続プールの場合と同様に，アプリケーションが使用する Connection オブジェクトは XAConnection クラスが生成しますが，PooledConnection クラスや JDBC2.0 拡張機能が提供する DataSource クラスが生成する Connection オブジェクトと比べて，次の点が異なります。

- 接続情報に XALocalCommitMode(true)を設定していない場合には，XADataSource クラス使用時，AutoCommit のモードは OFF になります。また，AutoCommit のモードを ON にしたり（Connection クラスの setAutoCommit(true)メソッドの発行），Connection クラスの commit メソッド，または rollback メソッドを実行したりすると，SQLException となります。ただし，XALocalCommitMode(true)をプロパティに設定している場合だけ AutoCommit のモードは ON となります。
- 一つの XAResource に対して複数の Connection クラスを生成してもトランザクションは一つになります。XAResource，Connection は一対一で使用するようにしてください。トランザクションがローカルトランザクション※1 の場合，Connection クラスの setAutoCommit(true)メソッド，commit メソッド，および rollback メソッドを呼び出せます。同じ XAResource を使用するトランザクションがグローバルトランザクション※2 を開始すると，自動的に AutoCommit のモードは OFF になります。再び，トランザクションがローカルトランザクションに移行すると，開始前の AutoCommit のモードに変更されます。複数の Connection を使用して，トランザクションがローカルトランザクションのときに setAutoCommit メソッドで AutoCommit のモードを変更する場合は，すべての Connection クラスのオブジェクトに対してそれぞれ setAutoCommit メソッドを呼び出して AutoCommit のモードを統一させる必要があります。現在のトランザクションがローカルトランザクションで，AutoCommit のモードが OFF の状態で新たに Connection を生成した場合，新しい Connection クラスのオブジェクトに対して setAutoCommit(false)を実行してください。

  注※1

  ローカルトランザクションとは，XAResource クラスの start 実行前，または XAResource クラスの end 実行後の状態にあるトランザクションのことです。

  注※2

  グローバルトランザクションとは，XAResource クラスの start 実行後，または XAResource クラスの end を実行するまでの状態にあるトランザクションのことで，トランザクションマネージャがトランザクション制御を行っている状態を指します。

- 次の状態のとき，Connection クラスの close メソッドを実行しても，接続しているデータベースによってトランザクションが決着されないことがあります。
  - 接続情報に XALocalCommitMode(true)を設定している。
  - AutoCommit のモードを OFF にしている。
  - ローカルトランザクション内で SQL を実行し，Connection クラスの commit メソッド，または rollback メソッドを実行しない。

  トランザクションを決着させるには，Connection クラスの close メソッドを実行する前に，明示的に Connection クラスの commit メソッド，または rollback メソッドを実行する必要があります。
- HiRDB の配列更新機能を使用するには，setXALocalCommitMode(true)，および setBlockUpdate(true)に設定して HiRDB に接続後，AutoCommit(false)を指定する必要があります。HiRDB に setXALocalCommitMode(true)を設定して接続し，HiRDB の配列更新機能を使用する例を次に示します。

  なお，HiRDB の配列更新機能については，「4.7.2(2)　バッチ更新」を参照してください。

```
XADataSource xads = new XADataSource();
xads.setDatabaseName("HIRDB");
xads.setXALocalCommitMode(true);
                .
                .
Connection con = xaconnection.getConnection();
String sql = "INSERT INTO SAMPLE VALUES(?)";
PreparedStatement pst = con.prepareStatement(sql);
con.setAutoCommit(false);
pst.setInt(1,10);
pst.addBatch();
pst.setInt(1,20);
pst.addBatch();
pst.executeBatch();
con.commit();
```

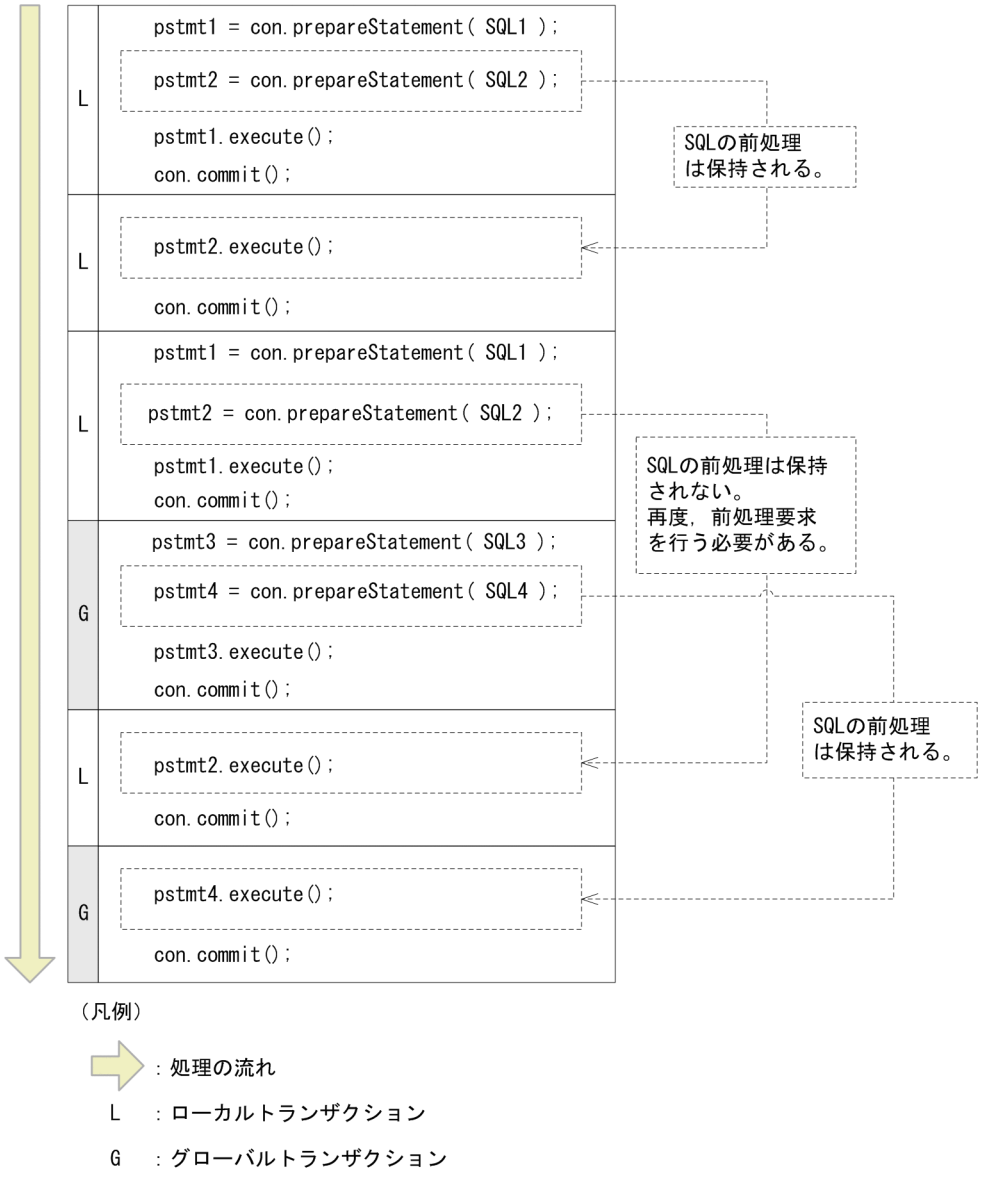
- DABroker Library は，SQL の前処理を複数の Commit，Rollback にわたって保持します。

  この機能を XA 環境下で使用できるデータベースは，HiRDB のバージョンが 07-01 以降です。また，SQL の前処理の保持機能が使用できるのは，グローバルトランザクション，およびローカルトランザクションを混在せずに使用している場合だけです。

  ローカルトランザクションとグローバルトランザクションを混在させて使用すると，ローカルトランザクション SQL の前処理の保持機能は使用できません。

  XA 環境下でローカルトランザクションとグローバルトランザクションが混在した JavaUP の場合の前処理保持状態を次に示します。

図 4-5　XA 環境下でローカルトランザクションとグローバルトランザクションが混在した JavaUP の場合の前処理保持状態



(b)　関連するクラス

分散トランザクション機能に関連するクラスを次の表に示します。

表 4-49　分散トランザクションに関連するクラス

クラス	概要
javax.sql. ConnectionEventListener	• トランザクション連携機能が提供 • トランザクション連携機能で，接続の切断，SQL エラーなどをこのクラスのオブジェクトから検知※1
javax.sql.DataSource	• トランザクション連携機能が提供 • データベース接続時のアプリケーションとのインタフェース

クラス	概要
javax.sql.DataSource	• トランザクションマネージャとの連携および接続プールを制御 • JNDI に登録して使用
javax.sql.XADataSource	• JDBC2.0 拡張機能が提供※2 • データベース接続に必要な接続情報の設定および取得用のメソッドを提供 • トランザクション連携機能が使用 • JNDI に登録して使用 • このクラスのオブジェクトから XAConnection オブジェクトを取得
javax.sql.XAConnection	• JDBC2.0 拡張機能が提供※2 • PooledConnection クラスのサブクラス • トランザクション連携機能が使用 • このクラスのオブジェクトがプール対象 • このクラスのオブジェクトから Connection オブジェクトを取得
javax.transaction.xa. XAResource	• JDBC2.0 拡張機能が提供※2 • トランザクションマネージャが使用する XA 関連メソッドを提供
javax.transaction.xa.Xid	• JDBC2.0 拡張機能※2 およびトランザクションマネージャが提供 • XAResource クラスの引数および戻り値として使用

注※1

connectionErrorOccurred が返されるのは次のエラーコードの場合です。

接続データベースが HiRDB：-720，-722，-723，-728，-732，-932，-1700

接続データベースが Oracle：28，1012，3113，3114，12571

注※2

JDBC2.0 拡張機能が提供するパッケージ名称とクラス名称を次に示します。

種類	名称
パッケージ	JP.co.Hitachi.soft.DBPSV_Driver
XADataSource クラス	JdbcDbpsvXADataSource
XAConnection クラス	JdbcDbpsvXAConnection
XAResource クラス	JdbcDbpsvXAResource
Xid クラス	JdbcDbpsvXid

4.7.4　データベース接続の設定

DABroker Library で開発するアプリケーションでは，次の項目を設定します。ただし，トレースを取得するための情報の設定は任意です。

- 使用する JDBC ドライバの設定
- 接続するデータベースの設定
- データベースに接続するための各種情報の設定

- トレースを取得するための情報の設定（任意）

ここでは，それぞれの設定内容について説明します。

なお，アプリケーション運用時に必要となる，アプリケーションの組み込みやインタフェースの定義については，「4.3.1(5)　J2EE アプリケーション運用のための設定」を参照してください。

(1)　使用する JDBC ドライバの設定

JDBC ドライバのロードに，次のドライバ名称およびパッケージ名称を設定します。なお，DABroker Library で提供する JDBC ドライバは，JDBC2.1 の規格に準拠しています。

設定項目	設定値
ドライバ名称	JdbcDbpsvDriver
パッケージ名称	JP.co.Hitachi.soft.DBPSV_Driver

ドライバ名称およびパッケージ名称は，次のどちらかの方法で設定してください。

- Class.forName メソッドでの設定
  Class.forName メソッドでアプリケーションから JDBC ドライバを登録します。
  アプリケーション内に次のように記述してください。

```
Class.forName("JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDriver")
```

- システム（java.lang.System）プロパティの jdbc.drivers での指定
  システムプロパティに JDBC ドライバを登録します。
  システムプロパティでの指定方法は，Java の各処理系の仕様に依存します。
  次に，システムプロパティへの登録方法を示します。
  - Appletviewer での指定－Java アプレットの場合
    [JAVA_HOME]/.hotjava/properties ファイルに次の記述を追加してください。複数のドライバを使用する場合は，「:（コロン）」で区切って記述してください。なお，［JAVA_HOME］とは，環境変数 PATH に設定した値のことです。［JAVA_HOME］は，Java 実行環境によって異なります。

```
jdbc.drivers="JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDriver"
```

  - Java のメソッドでの設定
    System.setProperty メソッドを使用してシステムプロパティに登録してください。次のように記述してください。

```
System.setProperty("jdbc.drivers", "JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDriver") ;
```

JDBC ドライバの使用方法の詳細については，JDBC 関連ドキュメントを参照してください。

(2)　接続するデータベースの設定

DABroker Library で開発するアプリケーションは，DABroker に接続してデータベースに接続します。DABroker およびデータベースは，次のどちらかの方法で設定してください。

- Driver.connect メソッドの直接呼び出し
- DriverManager.getConnection メソッドの使用

同時に多数のユーザが接続するシステムの場合，ドライバマネージャを使用する接続方法では，ドライバマネージャ内で待ちが発生することがあります。多数のユーザが同時に接続することが想定されるシステムでは，Driver.connect メソッドの直接呼び出しの接続方法でアプリケーションを開発することをお勧めします。

- Driver.connect メソッドの直接呼び出しによる接続方法

  次のように，Driver クラスの connect メソッドを直接呼び出すことができます。

```
Driver drv = new JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDriver();
Connection con = drv.connect(String url,Properties info);
```

- DriverManager.getConnection メソッドの使用による接続方法

  DABroker との接続は，DriverManager.getConnection メソッドを使用します。引数のパターンは，次のどちらかです。

```
DriverManager.getConnection(String url,String user,String password);
DriverManager.getConnection(String url,Properties info);
```

次の表に，DriverManager.getConnection メソッド，および Driver.connect メソッドの引数の内容を示します。

表 4-50　getConnection メソッドおよび connect メソッドの引数の内容

引数	設定内容	指定
String url	データベースに接続するための各種情報の設定用 URL 構文 （「(3)　データベースに接続するための各種情報の設定」参照）	必須
String user	データベース接続のユーザ ID※1	必須
String password	データベース接続のパスワード※1	任意
Properties info	データベース接続のユーザ ID※1 （キーワード："user"）	必須
	データベース接続のパスワード※1 （キーワード："password"）	任意
	アプリケーションの名称※2 （キーワード："UAPNAME"）	任意
	DABroker からの受信データのバッファ長※3 （キーワード："BUF_SIZE"）	任意
	トレース取得の有無 （キーワード："JDBC_IF，SV_EVENT，TRC_NO"， 「(4)　トレースを取得するための情報の設定」参照）	任意
	エンコード文字形態※4 （キーワード："ENCODELANG"）	任意
	JDBC で取り扱うバッファ長※5 （キーワード："ROW_SIZE"）	任意
	OS 認証機能使用の有無※6 （キーワード："DAB_OSAUTH"）	任意

引数	設定内容	指定
Properties info	警告の保持※7 （キーワード："SQLWARNING_IGNORE"）	任意
	HiRDB のカーソルの保持※8 （キーワード："HIRDB_CURSOR"）	任意
	複数のパラメタセットの処理方法※9 （キーワード："BLOCK_UPDATE"）	任意
	Execute Direct 機能使用の有無※10 （キーワード："EXECUTE_DIRECT_MODE"）	任意
	LONGVARBINARY 型データアクセスモード※11 （キーワード："LONGVARBINARY_ACCESS"）	任意
	DABroker からの受信データのバッファプール数※12 （キーワード："BUF_POOL_SIZE"）	任意

注※1

データベースに接続するためのユーザ ID，およびパスワードは，あらかじめデータベースに登録しておいてください。

なお，ユーザ ID だけを指定し，パスワードの指定を省略した場合でも，データベースに接続できます。ただし，ユーザ ID の指定を省略し，パスワードだけを指定した場合は，データベースに接続できません（データベースの仕様に従っています）。

注※2

開発するアプリケーションを一意に識別するための名称です。DABroker で取得する，アプリケーションとの通信トレースの PAPNAME に使用されます。指定を省略した場合は，「DABroker for Java Version 2」が使用されます。

アプリケーションの名称は，30 バイト以内の文字列で指定してください。31 バイト以上の文字列を指定した場合，31 バイト以降は切り捨てられます。

注※3

DABroker からの受信データを格納するためのバッファ長を，キロバイト単位の数字文字列で指定します。BLOB データなどの長大データを使用する場合，想定されるデータ長以上の値をこの項目に指定します。

指定値

1～16000

指定を省略した場合，数字以外を指定した場合，または 1 より小さい値を指定した場合は 64 が仮定されます。16000 より大きい値を指定した場合は 16000 が仮定されます。

値が仮定されたときは，Exception トレースログに値が仮定されたことを表すメッセージ（KFDJ01009-W）が出力されます。出力される契機は DriverManager.getConnection メソッドなどで Connection オブジェクトを取得するときです。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「KFDJ01009-W」を参照してください。

ただし，BUF_SIZE の上限が 16,000 キロバイトのため，BLOB データなどの 16,000 キロバイトを超えるデータに一度にアクセスすることはできません。データベース種別が HiRDB の場合は，指定したバイト数を取り出す STRING のインタフェース（BLOB データの部分抽出機能）があるため，ここで指定したサイズ以下のサイズに分割して取り出してください。BLOB データの部分抽出

機能の詳細については，マニュアル「HiRDB UAP 開発ガイド」，およびマニュアル「HiRDB SQL リファレンス」を参照してください。

注※4

String クラスで文字変換に使用するエンコード文字形態を指定します。

指定値

Java がサポートしている文字エンコーディング：

MS932 など，指定したエンコード文字形態を使用します。

OFF：

JavaVM がサポートしている標準エンコードを使用します。

次に DABroker の漢字コードセットの値と JDBC 内のエンコーディングの対応を示します。

DABroker（UNIX）の DAB_LANG の値，または DABroker（Windows）の DABLANGMODE の漢字コードセットの値	JDBC 内の文字エンコーディング
SJIS（シフト JIS）モード	VM の標準エンコードを使用します。
EUC モード	EUCJIS
ASCII モード	8859_1
UTF-8 モード	UTF-8

指定を省略した場合，OFF が仮定されます。

注※5

JDBC で取り扱うバッファ長を指定します。指定はメガバイト単位の数字文字列で指定します。利用するデータベースの 1 レコードの長さより大きい値を指定してください。

指定値

16～512

指定を省略した場合，数字以外を指定した場合，または 16 より小さい値を指定した場合は 16 が仮定されます。

512 より大きい値を指定した場合は 512 が仮定されます。

注※6

OS 認証機能を使用したデータベース接続をするかどうか指定します。

指定値

TRUE ：使用します。

FALSE：使用しません。

指定を省略した場合は，FALSE が仮定されます。この指定は，Oracle9i 9.2.0 に接続する場合だけ有効です。それ以外のバージョンの Oracle に対して指定した場合は動作を保証しません。接続するデータベースが Oracle 以外の場合は，指定値に関係なく，OS 認証機能を使用しません。

注※7

データベースから返される警告を，Connection クラスで保持するかどうかを指定します。

指定値

TRUE ：保持しません。

FALSE：保持します。

指定を省略した場合は，FALSE が仮定されます。

注※8

HiRDB での検索時，カーソルが複数の Commit にわたって有効かどうかを指定します。

指定値

TRUE：カーソルは保持されます。アプリケーションは，続けて Fetch できます（LOCK TABLE UNTIL DISCONNECT が前提です）。

FALSE：カーソルはクローズされます。

指定を省略した場合は，FALSE が仮定されます。Oracle では，カーソルは常に複数の Commit，または Rollback にわたって有効です。通常，Fetch 時，DABroker に Fetch データをバッファリングするため，Commit 直後の Fetch でエラーが発生するとは限りません。

MetaData を取得する場合，「TRUE」を指定して LOCK TABLE UNTIL DISCONNECT をしないで，複数の Commit にわたって Fetch を行うとエラーが発生します。MetaData を取得する場合は，「TRUE」を設定しないでください。

MetaData を取得する場合，「FALSE」，またはデフォルトの状態で ResultSet の Fetch 実行中に Commit を実行するとエラーが発生します。

この機能については，マニュアル「HiRDB SQL リファレンス」を参照してください。

注※9

接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを指定します。

指定値

TRUE ：一度に処理します。

FALSE：パラメタセットを一つずつ分割して処理します。

指定を省略した場合は，FALSE が仮定されます。この指定は，接続データベースが HiRDB のときだけ有効です。HiRDB 以外のデータベースの場合は，常にパラメタセットを一つずつ分割して処理します。また，実際にパラメタセットが一度に処理されるかどうかは，DABroker および HiRDB の仕様に従います。

注※10

接続データベースが HiRDB の場合，INSERT，UPDATE，DELETE など，Statement クラスを使用したデータベースの更新で HiRDB の EXECUTE DIRECT 機能を使用するかどうかを指定します。

指定値

TRUE ：EXECUTE DIRECT 機能を使用します。

FALSE：EXECUTE DIRECT 機能を使用しません。

指定を省略した場合は，FALSE が仮定されます。この指定は，接続データベースが HiRDB のときだけ有効であり，HiRDB 以外のデータベースの場合は，常に EXECUTE DIRECT 機能を使用しません。

注※11

LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースのアクセス方法を指定します。

指定値

REAL：実データでアクセスします（デフォルト）。

LOCATOR：HiRDB の位置づけ子（locator）機能を使用してアクセスします。

上記以外：「REAL」を指定したものとします。なお，次のどれかの場合，「LOCATOR」を指定しても「REAL」が指定されたと見なします。

- 接続先 DB が HiRDB 以外

- 接続先 DB である HiRDB の HiRDB サーバ，または HiRDB クライアントライブラリのバージョンが 07-00 以前

LONGVARBINARY_ACCESS を指定した場合の注意事項を次に示します。

- BUF_SIZE と LONGVARBINARY_ACCESS を指定した場合

  BUF_SIZE および LONGVARBINARY_ACCESS の指定によって，BLOB／BINARY データの取得方法が変化します。BLOB／BINARY データの取得方法については，「(3)(b)　URL に指定する各種情報」の「BLOB／BINARY データの取得方法」を参照してください。

- 実行性能

  LONGVARBINARY_ACCESS に「REAL」を指定すると，データベースへのアクセスは，ResultSet.next または CallableStatement.execute 実行時に 1 回行われるのに対し，「LOCATOR」を指定すると，ResultSet.next または CallableStatement.execute 実行時と，getBytes などのデータを取得するメソッド実行時の 2 回行われます。そのため，「LOCATOR」を指定すると「REAL」を指定した場合に比べて性能が低下する場合があります。

- AUTO コミット

  LONGVARBINARY_ACCESS が「LOCATOR」でかつ，AUTO コミットが有効の場合，ストアドプロシジャの BLOB／BINARY 型出力パラメタを受け取れません（getBytes などでデータを取得できません）。ストアドプロシジャの出力パラメタを受け取るには AUTO コミットを無効にしてください。

- トランザクション終了後のデータ操作

  LONGVARBINARY_ACCESS が「LOCATOR」の場合，SQL の実行結果取得（ResultSet.next，CallableStatement.execute）からデータ操作（Blob.getBytes，InputStream.read など）までにトランザクションが終了すると，データ操作ができません。また，HIRDB_CURSOR が「TRUE」であってもトランザクション終了後のデータ操作は実行できません。

  そのため，データ操作はトランザクション終了前に行ってください。

注※12

DABroker からの受信データを格納するためのバッファプール数を設定します。指定しない場合は，0（デフォルト）が仮定されます。受信バッファプール数指定機能については，「4.5.2　受信バッファプール数の指定」を参照してください。

指定値

- 0（デフォルト）

  受信バッファプール数指定機能を使用しません。受信バッファをすべてプールします。

- 1～1024

  引数に指定された値を受信バッファプール数に設定します。ただし，接続するデータベースが HiRDB または ORACLE 以外の場合は，受信バッファプール数は 1～64 になります。この場合，64 より大きい値を指定しても 64 個までしかプールできません。

- 上記以外

  0 より小さい値を指定した場合，または数字以外を指定した場合は 0 が仮定されます。1024 より大きい値を設定した場合は 1024 が仮定されます。

  値が仮定されたときは，Exception トレースログにメッセージ（KFDJ01011-W）が出力されます。このメッセージが出力される契機は DriverManager.getConnection メソッドなどで Connection オブジェクトを取得するときです。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「KFDJ01011-W」を参照してください。

  ただし，SQLWarnning は作成されません。

(3)　データベースに接続するための各種情報の設定

ここでは，DABroker およびデータベースの各種情報の設定について説明します。

(a)　URL の指定形式

引数 URL は，次の形式で指定します。

なお，任意に指定する項目については，[]（角括弧）で示しています。

```
jdbc:hitachi:dbplib://DB =データベース種別[,DBID = データベースの接続情報][,DBENV = 接続先データベース定義情報][,DBHOST = データベース(HiRDB)のホスト名][,BUF_SIZE = 受信バッファ長][,ENCODELANG = 変換文字指定][,HIRDB_CURSOR = HiRDBカーソル動作モード]
```

- 必須の指定を省略した場合は，エラー（SQLException を throw する）となります。
- URL で指定する項目の中または項目の間でスペースを指定すると，指定値が無効となります。

(b)　URL に指定する各種情報

各種情報に指定する内容を，次に示します。

● URL での指定項目

表 4-51　URL での指定項目

各種情報	指定内容
データベース種別 (DB=)	DABroker から接続するデータベースの種別を指定します。 **指定値**：表 4-52 参照 指定は必須です。
データベースの 接続情報 (DBID=)	接続するデータベース情報を指定します。 **指定値**：表 4-53 参照
接続先データベース定義情報 (DBENV=)	DABroker の接続先データベース定義情報を次の形式で指定します。 **データベース種別名：データベース名** データベース種別が DABroker－Database Connection Server の場合は，必ずこの項目を指定します。それ以外のデータベース種別の場合は，ここでの指定値は無視されます。
データベースのホスト名 (DBHOST=)	データベースの種別に HiRDB を指定する場合，HiRDB のホスト名を指定します。なお，HiRDB 以外のデータベースを指定する場合，ここでの指定値は無視されます。 **指定値**：ホスト名 指定は任意です。
受信バッファ長 (BUF_SIZE=)	DABroker からの受信データを格納するためのバッファ長を，キロバイト単位の数字文字列で指定します。BLOB データなどの長大データを使用する場合，想定されるデータ長以上の値をこの項目に指定します。 **指定値**：1～16000 指定を省略した場合，数字以外を指定した場合，または 1 より小さい値を指定した場合は 64 が仮定されます。16000 より大きい値を指定した場合は 16000 が仮定されます。 値が仮定されたときは，Exception トレースログに値が仮定されたことを表すメッセージ（KFDJ01009-W）が出力されます。出力される契機は

各種情報	指定内容
受信バッファ長 （BUF_SIZE=）	DriverManager.getConnection メソッドなどで Connection オブジェクトを取得するときです。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「KFDJ01009-W」を参照してください。 ただし，BUF_SIZE の上限が 16,000 キロバイトのため，BLOB データなどの 16,000 キロバイトを超えるデータに一度にアクセスすることはできません。データベース種別が HiRDB の場合は，指定したバイト数を取り出す STRING のインタフェース（BLOB データの部分抽出機能）があるため，ここで指定したサイズ以下のサイズに分割して取り出してください。BLOB データの部分抽出機能の詳細については，マニュアル「HiRDB UAP 開発ガイド」，およびマニュアル「HiRDB SQL リファレンス」を参照してください。
受信バッファプール数 （BUF_POOL_SIZE）	DABroker からの受信データを格納するためのバッファプール数を設定します。指定しない場合は，0（デフォルト）が仮定されます。受信バッファプール数指定機能については，「4.5.2　受信バッファプール数の指定」を参照してください。 **指定値：** 0（デフォルト） 受信バッファプール数指定機能を使用しません。受信バッファをすべてプールします。 1～1024 引数に指定された値を受信バッファプール数に設定します。ただし，接続するデータベースが HiRDB または ORACLE 以外の場合は，受信バッファプール数は 1～64 になります。この場合，64 より大きい値を指定しても 64 個までしかプールできません。 上記以外 0 より小さい値を指定した場合，または数字以外を指定した場合は 0 が仮定されます。1024 より大きい値を設定した場合は 1024 が仮定されます。 値が仮定されたときは，Exception トレースログにメッセージ（KFDJ01011-W）が出力されます。このメッセージが出力される契機は DriverManager.getConnection メソッドなどで Connection オブジェクトを取得するときです。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「KFDJ01011-W」を参照してください。 ただし，SQLWarnning は作成されません。
変換文字指定 （ENCODELANG=）	String クラスで文字変換に使用するエンコード文字形態を指定します。 **指定値：**Java がサポートしている文字エンコーディング（MS932 など），または OFF（JavaVM がサポートしている標準エンコードを使用します） 指定は任意です。指定を省略した場合，OFF が仮定されます。
HiRDB カーソル動作モード （HIRDB_CURSOR=）	HiRDB での検索時，カーソルが複数の Commit にわたって有効かどうかを指定します。 **指定値：** TRUE カーソルは保持されます。アプリケーションは，続けて Fetch できます（LOCK TABLE UNTIL DISCONNECT が前提です）。 FALSE カーソルはクローズされます。 指定を省略した場合は，FALSE が仮定されます。Oracle では，カーソルは常に複数の Commit，または Rollback にわたって有効です。通常，Fetch 時，

各種情報	指定内容
HiRDB カーソル動作モード（HIRDB_CURSOR=）	DABroker に Fetch データをバッファリングするため，Commit 直後の Fetch でエラーが発生するとは限りません。 MetaData を取得する場合，「TRUE」を指定して LOCK TABLE UNTIL DISCONNECT をしないで，複数の Commit にわたって Fetch を行うとエラーが発生します。MetaData を取得する場合は，「TRUE」を設定しないでください。 MetaData を取得する場合，「FALSE」またはデフォルトの状態で，ResultSet の Fetch 実行中に Commit を実行するとエラーが発生します。 接続データベースが HiRDB で検索 SQL（SELECT 文）に「UNTIL DISCONNECT」を記述した場合は，「TRUE」を指定するか，DataSource クラスの setHiRDBCursorMode メソッドで「true」を設定してください。設定しない場合，Commit をわたった検索中の ResultSet クラスオブジェクトに対して close メソッドを実行してもカーソルをクローズしません。 この機能を使用する場合，マニュアル「HiRDB SQL リファレンス」を参照してください。

● データベースの指定値

表 4-52　データベース種別の指定値

データベース種別	データベース種別の指定値
Oracle	oracle または ORACLE
Oracle8i※	oracle8i または ORACLE8I
HiRDB	hirdb または HIRDB

注

- 指定値の文字列は，大文字，小文字のどちらでもかまいません。
- 「=」の前後に，空白またはタブ文字を指定した場合は，エラー（SQLException を throw する）となります。

注※

データベース種別に Oracle8i の指定値を指定して，DABroker 環境設定の「使用する Oracle のバージョン」に Oracle9i を指定した場合は Oracle9i に，Oracle10g を指定した場合は Oracle10g に接続できます。

表 4-53　データベースの接続情報の指定値

データベース種別	データベースの接続情報の指定内容	指定の省略可否
Oracle	SQL*Net の接続文字列※1	任意※1
Oracle8i		
HiRDB	HiRDB のシステムポート番号※1※2	任意※1

注

- 「=」の前後に，空白またはタブ文字を指定した場合は，エラー（SQLException を throw する）となります。
- 「DB=データベース種別」との区切りに「,（コンマ）」を使用します。

- 「,（コンマ）」を 2 回以上指定した場合は，最後の「,（コンマ）」以降がデータベースの接続情報の指定となります。
- データの区切りに「,（コンマ）」以外のキーを使用した場合は，データベース接続情報の指定は無効となります。

注※1

指定を省略する場合は，DABroker の動作環境定義ファイルでの指定が有効になります。

注※2

HiRDB クライアントの環境変数グループ名を指定する場合は，グループ名の先頭に@を付加します。

（例 1）

Windows の場合

HiRDB クライアントの環境変数グループ名が HiRDB_ENV_GROUP のときは，次のように指定します。

DBID=@DABENVGRP=HiRDB_ENV_GROUP

（例 2）

UNIX の場合

HiRDB クライアントの環境変数グループ名のパスが/HiRDB_P/Client/HiRDB.ini のときは，次のように指定します。

DBID=@DABENVGRP=/HiRDB_P/Client/HiRDB.ini

DBID=@DABENVGRP=＜環境変数グループ名＞を指定する場合は，指定内容に半角の空白を含めないでください。指定内容が次に示す例のどれかに該当する場合，DBID は正しく設定されません。

（例）

DBID=@DABENVGRP=＜環境変数グループ名＞の指定内容に半角スペースを含みます。OS は Windows で，環境変数グループ名は HiRDB_ENV_GROUP とします。

半角の空白を△で示します。

DBID△=@DABENVGRP=HiRDB_ENV_GROUP

DBID=△@DABENVGRP=HiRDB_ENV_GROUP

DBID=@△DABENVGRP=HiRDB_ENV_GROUP

DBID=@DABENVGRP△=HiRDB_ENV_GROUP

DBID=@DABENVGRP=△HiRDB_ENV_GROUP

DBID=@DABENVGRP=HiRDB_ENV_GROUP△

● BLOB／BINARY データの取得方法

LONGVARBINARY_ACCESS を指定した場合の各メソッド実行時の動作を次に示します。

接続先データベースに HiRDB を指定した場合

表 4-54　BLOB／BINARY データの取得方法（1）

指定値	CallableStatement.execute および ResultSet.next	CallableStatement.getBytes および ResultSet.getBytes	Blob.getBytes
REAL	BLOB／BINARY データの全体を接続先データベースから取得します。	ResultSet.next で取得した BLOB／BINARY データを使用します。	ResultSet.next で取得した BLOB／BINARY データから，引数で指定された範囲を切り出して取得します。

指定値	CallableStatement.execute および ResultSet.next	CallableStatement.getBytes および ResultSet.getBytes	Blob.getBytes
LOCATOR	BLOB／BINARY データの全体ではなく，接続先データベース内の BLOB／BINARY データを示すロケータを取得します。	BLOB／BINARY データの全体を接続先データベースから BUF_SIZE×1,024 バイト単位に分割して取得します。	引数で指定された範囲の BLOB／BINARY データを接続先データベースから BUF_SIZE×1,024 バイト単位に分割して取得します。

表 4-55　BLOB／BINARY データの取得方法（2）

指定値	CallableStatement.getBinaryStream および ResultSet.getBinaryStreamBlob.getBinaryStream	Blob.length	Blob.position
REAL	getBinaryStream によって取得した InputStream の read メソッドを実行した場合，ResultSet.next で取得した BLOB／BINARY データから抽出して取得します。	ResultSet.next メソッドで取得した BLOB／BINARY データから，データ長を取得します。	ResultSet.next で取得した BLOB／BINARY データから，検索パターンに一致する位置を取得します。
LOCATOR	getBinaryStream によって取得した InputStream の read メソッドを実行した場合，接続先データベースからデータを取得します。	接続先データベースからデータ長を取得します。	接続先データベースから，検索パターンに一致する位置を取得します。

表 4-56　BLOB／BINARY データの取得方法（3）

指定値	InputStream.available※	InputStream.skip※
REAL	ロケータの示す実データの長さ以下の値を返します。	ロケータの示す実データの長さ以下の値を返します。
LOCATOR	BUF_SIZE×1,024 バイト以下の値を返します。	BUF_SIZE×1,024 バイト以下の値を返します。

注※

CallableStatement.getBinaryStream，ResultSet.getBinaryStream，または Blob.getBinaryStream で取得した InputStream です。

接続先データベースに Oracle を指定した場合

表 4-57　BLOB／BINARY データの取得方法（4）

指定値	CallableStatement.execute および ResultSet.next	CallableStatement.getBytes および ResultSet.getBytes	Blob.getBytes
REAL LOCATOR	BLOB／BINARY データの全体を接続先データベースから BUF_SIZE ×1,024 バイト単位に分割して取得します。	ResultSet.next で取得した BLOB／BINARY データを使用します。	ResultSet.next で取得した BLOB／BINARY データから，引数で指定された範囲を切り出して取得します。

表 4-58　BLOB／BINARY データの取得方法（5）

指定値	CallableStatement.getBinaryStream および ResultSet.getBinaryStreamBlob.getBinaryStream	Blob.length	Blob.position
REAL	ResultSet.next で取得した BLOB／BINARY データを使用します。	ResultSet.next で取得した BLOB／BINARY データから，データ長を取得します。	ResultSet.next で取得した BLOB／BINARY データから，検索パターンに一致する位置を取得します。
LOCATOR			

表 4-59　BLOB／BINARY データの取得方法（6）

指定値	InputStream.available※	InputStream.skip※
REAL	ロケータの示す実データの長さ以下の値を返します。	ロケータの示す実データの長さ以下の値を返します。
LOCATOR		

注※

CallableStatement.getBinaryStream，ResultSet.getBinaryStream，または Blob.getBinaryStream で取得した InputStream です。

(4)　トレースを取得するための情報の設定

JDBC ドライバでは，次のトレースを取得できます。トレースの出力内容については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.　性能解析トレースを使用した性能解析」を参照してください。

- JDBC インタフェースメソッドトレース
- DABroker とのイベントトレース

(a)　トレースの取得および出力ストリームの設定方法

次の項目を指定して，トレースの取得を設定してください。次の項目が指定されていない場合は，トレースを取得できません。

1. DriverManager.getConnection メソッドまたは Driver.connect メソッドの引数 Properties info でのトレース取得有無の指定
2. DriverManager.setLogWriter メソッドでの，トレースの出力ストリームの指定

1.の Properties info で指定する値を次の表に示します。1.のメソッドと引数については，「(3)　データベースに接続するための各種情報の設定」を参照してください。

表 4-60　トレースを取得するための指定値

トレースの種類およびトレース関連項目	Properties info のキーワード	Properties info の値
JDBC インタフェースメソッドトレース※1	JDBC_IF	指定値 ON：トレースを取得する OFF：トレースを取得しない

トレースの種類 およびトレース関連項目	Properties info の キーワード	Properties info の値
DABroker とのイベント トレース※1	SV_EVENT	**指定値** ON：トレースを取得する OFF：トレースを取得しない
各トレースのエントリ数※2	TRC_NO	**デフォルト値** 500（エントリ） **指定範囲** 10～1000（エントリ）

注※1

ON または OFF 以外の値を指定した場合は「トレースを取得しない」設定となります。

注※2

数字以外の値を指定した場合は「トレースを取得しない」設定となります。

(b)　指定したストリームへの出力タイミング

トレースは，各トレースのエントリ数分，JDBC ドライバで保持されます。

保持されたトレースは，次のタイミングで DriverManager.setLogWriter メソッドで指定した出力ストリームに出力されます。

- Connection.Close メソッドの呼び出し（正常終了時）
- エラー発生時（SQLException の throw 時）

(c)　JDBC ドライバでのトレース取得時の注意事項

- JDBC ドライバは，JDBC ドライバ内でトレース情報の数がエントリ数を超えた場合は，トレースの古い順にトレース情報を上書きします。
- JDBC インタフェースメソッドトレース（キーワード：JDBC_IF）は，Entry および Return でそれぞれ 1 エントリのトレース領域を使用します。
- DABroker とのイベントトレース（キーワード：SV_EVENT）は，送信または受信イベントでそれぞれ 1 エントリのトレース領域を使用します。

4.7.5　JDBC2.0 拡張機能を使用したデータベース接続の設定

JDBC2.0 拡張機能で提供する JNDI 連携機能を使用した，データベースへ接続する操作について説明します。この機能はアプリケーションサーバの J2EE サーバモード，DB Connector を使用する場合は，アプリケーションで設定する必要はありません。

(1)　DataSource オブジェクトの生成

接続情報の設定は，DataSource オブジェクトで実行します。

DataSource オブジェクトの生成で必要な，ドライバ拡張機能のパッケージ名称および DataSource クラス名称を次に示します。

指定項目	指定値
パッケージ名称	JP.co.Hitachi.soft.DBPSV_Driver
DataSource クラス名称	JdbcDbpsvDataSource

DataSource オブジェクトの生成例を次に示します。

```
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDataSource  ds = null ;
ds = new JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDataSource() ;
```

(2)　データベースに接続するための各種情報の設定

データベースに接続するための各種情報は，JDBC2.0 拡張機能で提供するメソッドで設定します。提供するメソッドを次の表に示します。

表 4-61　各種情報を設定するメソッド

メソッド	機能
setNetworkProtocol	DABroker との接続種別を設定します。
setServerName	接続する DABroker のホスト名を設定します。
setPortNumber	接続する DABroker のポート番号を設定します。
setDatabaseName	接続するデータベースの種別を設定します。
setDescription	接続するデータベースに必要な接続付加情報を設定します。
setDBHostName	接続する HiRDB のホスト名を設定します。
setDBEnv	DABroker の接続先データベース定義情報を設定します。
setEncodLang	エンコード文字形態を設定します。
setJDBC_IF_TRC	JDBC インタフェースメソッドトレースの取得の有無を設定します。
setSV_EVENT_TRC	DABroker とのイベントトレースの取得の有無を設定します。
setTRC_NO	トレースのエントリ数を設定します。
setUapName	アプリケーション名称を設定します。
setBufSize	DABroker からの受信データのバッファ長を設定します。
setUser	データベース接続のユーザ ID を設定します。
setPassword	データベース接続のパスワードを設定します。
setXAOpenString※	XA_OPEN 文字列を設定します。
setXACloseString※	XA_CLOSE 文字列を設定します。
setRMID※	リソースマネージャの識別子を設定します。
setXAThreadMode※	XA 使用時のスレッドモードを設定します。
setXALocalCommitMode※	XA 使用時，データベースのオートコミットを有効にするかどうかを設定します。

メソッド	機能
setRowSize	JDBC で取り扱うバッファ長を指定します。
setOSAuthorize	OS 認証機能を使用してデータベースに接続するかどうかを設定します。
setSQLWarningIgnore	データベースから返される警告を Connection クラスで保持するかどうかの情報を設定します。
setHiRDBCursorMode	接続データベースが HiRDB の場合，検索時にカーソルが複数の Commit にわたって有効かどうかを設定します。
setNotErrorOccurred	connectionErrorOccurred が呼ばれるかどうかを設定します。
setBlockUpdate	接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。
setExecuteDirectMode	接続データベースが HiRDB の場合，Statement クラスを使用したデータベースの更新で HiRDB の Execute Direct 機能を使用するかどうかを設定します。
setLONGVARBINARY_Access	HiRDB で LONGVARBINARY（列属性 BLOB，列属性 BINARY）へのアクセス方法を設定します。
setBufferPoolSize	DABroker からの受信データのバッファプール数を設定します。

注※

JdbcDbpsvXADataSource クラスでだけ提供するメソッドです。

JDBC2.0 拡張機能では，接続情報を取得するためのメソッドも提供しています。接続情報を取得するためのメソッドを次の表に示します。

表 4-62　各種情報を取得するメソッド

メソッド	機能
getNetworkProtocol	DABroker との接続種別を取得します。
getServerName	接続する DABroker のホスト名を取得します。
getPortNumber	接続する DABroker のポート番号を取得します。
getDatabaseName	接続するデータベースの種別を取得します。
getDescription	接続するデータベースに必要な接続付加情報を取得します。
getDBHostName	接続する HiRDB のホスト名を取得します。
getDBEnv	DABroker の接続先データベース定義情報を取得します。
getEncodLang	エンコード文字形態を取得します。
getJDBC_IF_TRC	JDBC インタフェースメソッドトレースの取得の有無を取得します。
getSV_EVENT_TRC	DABroker とのイベントトレースの取得の有無を取得します。
getTRC_NO	トレースのエントリ数を取得します。
getUapName	アプリケーション名称を取得します。
getBufSize	DABroker からの受信データのバッファ長を取得します。

メソッド	機能
getUser	データベース接続のユーザ ID を取得します。
getPassword	データベース接続のパスワードを取得します。
getXAOpenString※	XA_OPEN 文字列を取得します。
getXACloseString※	XA_CLOSE 文字列を取得します。
getRMID※	リソースマネージャの識別子を取得します。
getXAThreadMode※	XA 使用時のスレッドモードを取得します。
getXALocalCommitMode※	XA 使用時，データベースのオートコミットを有効にしているかどうかの設定を取得します。
getRowSize	JDBC で取り扱うバッファ長を取得します。
getOSAuthorize	OS 認証機能を使用してデータベースに接続するかどうかの設定を取得します。
getSQLWarningIgnore	データベースから返される警告を Connection クラスで保持するかどうかの情報を取得します。
getHiRDBCursorMode	接続データベースが HiRDB の場合，検索時にカーソルが複数の Commit，または Rollback にわたって有効かどうかの設定を取得します。
getNotErrorOccurred	connectionErrorOccurred が呼ばれるかどうかの設定を取得します。
getBlockUpdate	接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかの設定を取得します。
getExecuteDirectMode	接続データベースが HiRDB の場合，Statement クラスを使用したデータベースの更新で HiRDB の Execute Direct 機能を使用するかどうかの設定を取得します。
getLONGVARBINARY_Access	HiRDB で LONGVARBINARY（列属性 BLOB，列属性 BINARY）へのアクセス方法を取得します。
getBufferPoolSize	DABroker からの受信データのバッファプール数を取得します。

注※

JdbcDbpsvXADataSource クラスでだけ提供するメソッドです。

各メソッドの機能および文法については，マニュアル「アプリケーションサーバ リファレンス API 編」を参照してください。

(3)　JNDI への情報の登録

接続情報を設定した DataSource オブジェクトを，JNDI に登録します。JNDI は実行環境によって，幾つかのサービスプロバイダを選択できます。

File System サービスプロバイダを使用する JNDI への登録例を次に示します。ほかのサービスプロバイダへの登録については，JNDI のドキュメントを参照してください。なお，JNDI に登録するときの論理名称は，"jdbc"というサブコンテキスト（登録例では jdbc/TestDataSource）の下に登録するようにしてください。

```
// JDBC2.0拡張機能が提供するDataSourceオブジェクトを生成する
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDataSource  ds;
ds = new JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDataSource();

// 接続情報を設定する
   :
// システムプロパティを取得する
Properties sys_prop = System.getProperties();

// File Systemサービスプロバイダのプロパティを設定する
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");

// File Systemサービスプロバイダで使用するディレクトリを設定する
// (この場合, c:¥JNDI_DIRの下に登録される)
sys_prop.put(Context.PROVIDER_URL, "file:c:"+File.separator + "JNDI_DIR");

//システムプロパティを更新する
System.setProperties(sys_prop);

// JNDIを初期化する
Context ctx = new InitialContext();

// JDBC2.0拡張機能が提供するDataSourceオブジェクトを,
// jdbc/TestDataSourceという論理名称でJNDIに登録する
ctx.bind("jdbc" + File.separator + "TestDataSource", ds);
   :
```

(4)　JNDI からの情報の取得

データベースに接続するときに，JNDI に登録した DataSource オブジェクトを取得します。

File System サービスプロバイダを使用する JNDI から，DataSource オブジェクトを取得する例を次に示します。ほかのサービスプロバイダから取得する方法については，JNDI のドキュメントを参照してください。

```
// システムプロパティを取得する
Properties sys_prop = System.getProperties();

// File Systemサービスプロバイダのプロパティを設定する
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");

// File Systemサービスプロバイダで使用するディレクトリを設定する
// (この場合, c:¥JNDI_DIRの下に登録されている)
sys_prop.put(Context.PROVIDER_URL, "file:c:"+File.separator + "JNDI_DIR");

// システムプロパティを更新する
System.setProperties(sys_prop);

// JNDIを初期化する
Context ctx = new InitialContext();

//jdbc/TestDataSourceという論理名称のオブジェクトをJNDIから取得する
Object obj = ctx.lookup("jdbc" + File.separator + "TestDataSource");

// 取り出したオブジェクトを，DataSourceクラスの型にキャストする
DataSource ds = (DataSource)obj;
   :
```

(5)　データベースへの接続

JNDI に登録した DataSource オブジェクトを取得後，DataSource オブジェクトに対し，getConnection メソッドを呼び出して，データベースへ接続します。

getConnection メソッドの呼び出し例を次に示します。

```
DataSource ds

// JNDIからDataSourceオブジェクトを取得する
   :
```

```
// getConnectionメソッドを呼び出す
Connection con = ds.getConnection("USERID","PASSWORD")※
```

注※

getConnection メソッドの引数（ユーザ ID，パスワード）は省略できます。引数を指定した場合は，DataSource オブジェクトに設定した接続情報よりも優先されます。

必要な接続情報が DataSource オブジェクトに設定されていない場合，接続情報の内容が不正な場合，データベースとの接続に失敗した場合は，getConnection メソッドは SQLException を throw します。

JNDI から DataSource オブジェクトを取得したあと，接続情報を設定し直すことができます。設定し直す場合は，DataSource オブジェクトを JDBC2.0 拡張機能が提供する DataSource クラスの型にキャストします。キャストする例を次に示します。

```
DataSource ds
JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDataSource  dbp_ds;

// JNDIからDataSourceオブジェクトを取得する
   :
// DataSourceオブジェクトを，JDBC拡張機能が提供する
// DataSourceクラスの型にキャストする
dbp_ds = (JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDataSource)ds;

// 接続情報を再設定する
   :
```

(6)　JDBC2.0 拡張機能で提供する機能のトレースを取得するための情報の指定

JDBC2.0 拡張機能では，次のトレースを取得できます。トレースの出力内容については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「7.　性能解析トレースを使用した性能解析」を参照してください。

- JDBC インタフェースメソッドトレース
- DABroker とのイベントトレース

(a)　トレースの取得

トレースを取得する場合は，次の項目を設定してください。次の項目が設定されていない場合は，トレースを取得できません。

- JdbcDbpsvDataSource，JdbcDbpsvConnectionPoolDataSource または JdbcDbpsvXADataSource クラスの setLogWriter メソッドでの有効なログストリームの指定
- setJDBC_IF_TRC，setSV_EVENT_TRC および setTRC_NO メソッドで，トレース取得の指定

(b)　トレース取得時の注意事項

setLogWriter メソッドで指定したログストリームは，JNDI には登録されません。トレースの取得は，JNDI から DataSource オブジェクトを取得するたびに，setLogWriter メソッドで指定してください。

4.7.6　エラー時の JDBC ドライバの処理

データベース接続，DABroker，または JDBC ドライバでエラーが発生した場合は，SQLException を throw します。SQLException オブジェクトの reason には，次の項目が設定されます。

- JDBC ドライバのメッセージ
- エラーを検知したクラス名称.メソッド名称

- DABroker からのメッセージ※

注※
DABroker からのメッセージは，存在する場合だけ設定されます。

なお，次のようなエラーの場合は，SQLException を throw しないで，JDBC インタフェースの仕様で決まっている処理を続行します。

- Driver.connection メソッドでデータベースに接続できない場合
- URL 指定で null が返される場合（DABroker Library で対象としていない URL が指定された場合など）

4.7.7　データ型の対応

ここでは，JDBC ドライバのデータ型について説明します。

JDBC ドライバでは，JDBC のデータ型と DABroker でサポートするデータベースのデータ型をマッピングして処理を実行しています。JDBC のデータ型と DABroker でサポートするデータ型は完全に一致していません。

なお，DABroker Library でサポートする文字型データのコード体系は，ASCII，シフト JIS，EUCJIS，UTF-8，および Chinese Simple です。

(1)　JDBC のデータ型と DABroker でサポートするデータ型の対応

次に，JDBC のデータ型と DABroker でサポートするデータ型の対応をデータベースごとに示します。

(a)　HiRDB の場合

表 4-63　HiRDB の場合のデータ型の対応

DABroker でサポートするデータ型	JDBC のデータ型
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT, DOUBLE PRECISION	FLOAT
SMALLFLT, REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	CHAR
NVARCHAR	VARCHAR
MCHAR	CHAR
MVARCHAR	VARCHAR
DATE	DATE
TIME	TIME

DABroker でサポートするデータ型	JDBC のデータ型
TIMESTAMP	TIMESTAMP
BLOB，BINARY LARGE OBJECT	LONGVARBINARY

(b)　Oracle の場合

表 4-64　Oracle の場合のデータ型の対応

DABroker でサポートするデータ型	JDBC のデータ型
VARCHAR2	VARCHAR
NUMBER	NUMERIC，DECIMAL，BIT，TINYINT，SMALLINT，INTEGER，BIGINT
FLOAT	FLOAT
LONG	LONGVARCHAR
ROWID	CHAR
DATE	DATE，TIME，TIMESTAMP
RAW	VARBINARY
LONG RAW	LONGVARBINARY
CHAR	CHAR
NCHAR※	CHAR
NVARCHAR2※	VARCHAR
CLOB※	CLOB
NCLOB※	CLOB
BLOB※	BLOB
BFILE※	BLOB

注※

接続データベースが Oracle9i，または Oracle10g の場合だけ使用できます。

(2)　データ型の注意事項

ここでは，データ型についての注意事項を示します。

- データ型のマッピングは，次のメソッドでデータベースに接続する場合は，データ型の変換を JDBC ドライバで実行します。次に示すメソッド実行時のデータ型の扱いについては，JDBC の仕様に依存します。
  - ResultSet クラスの getXXX メソッド
  - PreparedStatement クラスの setXXX メソッド
  - CallableStatement クラスの setXXX メソッド

なお，setXXX メソッド実行時は，JDBC ドライバでデータを保持し，execute 実行時にデータを DABroker に転送して，通信回数を削減しています。したがって，setXXX で発生するようなエラーが execute の実行時に発生する場合もあります。

- DABroker がサポートしていないデータ型に対して接続を要求した場合（Statement.execute 発行時など）は，DABroker からエラーが返ります。その後，DABroker Library は，SQLException を throw します。
- DABroker Library がサポートしていない列の属性を含む列情報を取得した場合，DATA_TYPE に OTHER，TYPE_NAME に OTHER がリターンされます。

4.7.8　データベース接続の実装時の注意事項

ここでは，データベース接続の実装時の注意事項について説明します。

- SQL にエスケープシーケンスを利用しない場合, setEscapeProcessing メソッドに false を設定すると SQL 変換をしません。このため，最適な処理速度となります。
- PooledConnection.getConnection メソッドで取得した Connection オブジェクトに対して，その Connection オブジェクトから派生した Statement オブジェクト, PreparedStatement オブジェクト, および CallableStatement オブジェクトの作成および実行中に Connection.close メソッドを実行すると，これらのオブジェクトの作成および実行処理が終了するまで，Connection.close メソッドの処理を待つことになります。そのため，これらのオブジェクトの作成および実行処理に長時間掛かっている場合は，Connection.close メソッドの処理時間も長くなります。
- PooledConnection オブジェクトによって生成された connectionErrorOccurred イベント内で, PooledConnection.getConnection メソッドで取得した Connection オブジェクトに対して, Connection.close メソッドを実行しないでください。connectionErrorOccurred イベント内で Connection.close メソッドを実行すると，JDBC に対するすべての要求が返ってこなくなることがあります。
- Windows の場合に，Statement.cancel メソッドで実行中の SQL を取り消すとき，タイミングによっては実行中の SQL が取り消されないことがあります。Statement.cancel メソッドを呼び出しても，実行中の SQL が取り消されない場合は，再度，Statement.cancel メソッドを呼び出してください。また，接続するデータベースが HiRDB の場合，cancel メソッド実行後，使用中のコネクションで生成したステートメントオブジェクトをすべてクローズして，再度ステートメントオブジェクトを作成してください。
- Windows の場合，PreparedStatement クラスおよび CallableStatement クラスの setObject メソッドに，InputStream クラスおよび Reader クラス（派生クラスも含む）のオブジェクトは指定できません。指定するとエラーになります。
- Windows で，HiRDB の BLOB 列および BINARY 列のデータを取得する場合，データベースアクセス方法は次のどちらかになります。この指定値によっては，JavaVM でのメモリ取得の性能に大きく影響します。
  - REAL（一括でデータを取得する）
  - LOCATOR（データを BUF_SIZE（デフォルト値は 64 キロバイト）分で分割してデータを取得する）

  「REAL」を指定すると，最大データ長分の BUF_SIZE を設定する必要があります。この場合，BLOB 列および BINARY 列以外の SQL 実行時にも BUF_SIZE 分のメモリを取得するため, ほかの SQL の実行性能にも影響を与えます。

「LOCATOR」を指定すると，データベースへのアクセスは，ResultSet.next()，または CallableStatment.execute()実行時と，getBytes などのデータを取得するメソッド実行時の 2 回行われます。そのため，「REAL」を指定した場合に比べて性能が低下することがあります。

データベースアクセス方法は，BLOB 列および BINARY 列で取得する実データの最大データ長によって設定してください。BLOB 列および BINARY 列を取得し，最大データ長が 1 メガバイト以上のときは「LOCATOR」を設定することをお勧めします。ただし，JavaVM でのメモリ確保時の性能に大きく差が生じるため，1 メガバイトでも「REAL」と設定した方が，性能が良いこともあります。

- setBinaryStream メソッドまたは setAsciiStream メソッドの引数※1 に指定したストリームデータのデータ長 L1※2 と，x.available メソッドで取得できる値 L2 が L1 ＞ L2 の場合，setBinaryStream メソッドはストリームデータの読み込み領域として引数 length で指定したサイズ（単位：バイト）のメモリを確保します。

  引数 length を大きくし過ぎると，データ長 L1 に関係なくメモリ不足になるおそれがあります。また，「引数 length ＞データ長 L1」の場合，読み込み処理終了後に読み込み領域をデータ長 L1 バイトで再確保するため，性能が劣化するおそれがあります。このため，引数 length の値は，データ長 L1 以下にしてください。

  注※1

  setBinaryStream メソッドまたは setAsciiStream メソッドの引数とは，setBinaryStream(int parameterIndex,InputStream x,int length)，または setAsciiStream(int parameterIndex,InputStream x,int length)のことです。

  注※2

  データ長 L1 は，InputStream クラスの read メソッドが，-1 をリターンするまで読み込めるデータの長さです。

- HP-UX の場合，DABroker Library は，64 ビットアドレッシングモードで作成しています。
- HP-UX の場合，XA インタフェース使用時，start メソッドおよび end メソッドを実行する際には次の点に注意してください。
  - start メソッド実行時，オートコミットモードの状態に関係なく，オートコミットモードを on にします。また，オートコミットモードの状態に関係なく，DABroker Library に対してオートコミットモードを off にするように要求します。
  - end メソッド実行後は，start メソッドを実行する前のオートコミットモードの状態に戻ります。
- HP-UX の場合に接続データベースが ORACLE のとき，XADataSource 使用して，xa_recover(XAResource.TMSTARTRSCAN | XAResoure.TMENDRSCAN)を実行すると，ORACLE の仕様上, XAResource.TMENDRSCAN 実行時に-5 の xa エラーが発生します。この場合，DABroker Library ではこのエラーを返しません。ただし，XA ログにはエラーの発生が出力されます

4.8　DB Connector を使用したデータベースとの接続

ここでは，DB Connector を使用したデータベースとの接続について説明します。

なお，DABroker Library で使用する DB Connector は，JDBC 2.0 仕様をサポートしています。

4.8.1　DB Connector で接続できるデータベース

DABroker Library を使用する場合，DB Connector を使用して接続できるデータベースには，HiRDB，Oracle および XDM/RD E2 があります。

4.8.2　DB Connector（RAR ファイル）の種類

DABroker Library で使用できる DB Connector の RAR ファイルの種類を次の表に示します。これらの RAR ファイルは，J2EE サーバおよびバッチサーバで使用できます。

表 4-65　DABroker Library で使用できる DB Connector の RAR ファイルの種類

RAR ファイル	接続できるデータベース	説明
DBConnector_DABJ_CP.rar	HiRDB Oracle XDM/RD E2	トランザクション管理をしない場合，またはローカルトランザクションを使用する場合（トランザクションのサポートレベルに NoTransaction または LocalTransaction を指定する場合）に使用します。 DABroker Library の ConnectionPoolDataSource を使用してデータベースに接続します。
DBConnector_DABJ_XA.rar※	HiRDB Oracle	グローバルトランザクションを使用する場合（トランザクションのサポートレベルに XATransaction を指定する場合）に使用します。DABroker Library の XADataSource を使用してデータベースに接続します。

注　新規に，DB Connector の RAR ファイルを使用する場合，アプリケーションサーバで提供する Connector 属性ファイルのテンプレートファイルを使用して，プロパティを定義できます。提供しているテンプレートファイルについては，「4.31.2　Connector 属性ファイルのテンプレートファイル」を参照してください。

注※　ライトトランザクション機能は有効にできません。

RAR ファイルは，サーバ管理コマンドを使用して操作します。サーバ管理コマンドを使用して RAR ファイルを操作する方法については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」を参照してください。

RAR ファイルの種類に応じて，指定できるトランザクションサポートレベルが異なります。また，ライトトランザクションを使用できるかどうかも異なります。

RAR ファイルごとに指定できるトランザクションサポートレベルを次の表に示します。

表 4-66　RAR ファイルごとに使用できるトランザクションサポートレベル（DB Connector）

使用する DB Connector（RAR ファイル）	トランザクションサポートレベル	ライトトランザクション	
		有効	無効
DBConnector_DABJ_CP.rar	NoTransaction LocalTransaction	○	○
DBConnector_DABJ_XA.rar	XATransaction	−	○

（凡例）　○：使用できる　　−：該当しない

4.8.3　必要なモジュール

DABroker Library を使用してデータベースに接続する場合に必要なモジュールのうち，アプリケーションサーバ以外によって提供されるものを次の表に示します。

表 4-67　データベースに接続する場合に必要なモジュール

接続するデータベース	必要なモジュール
HiRDB	HiRDB Run Time
Oracle	Oracle Client
XDM/RD E2	HiRDB Run Time

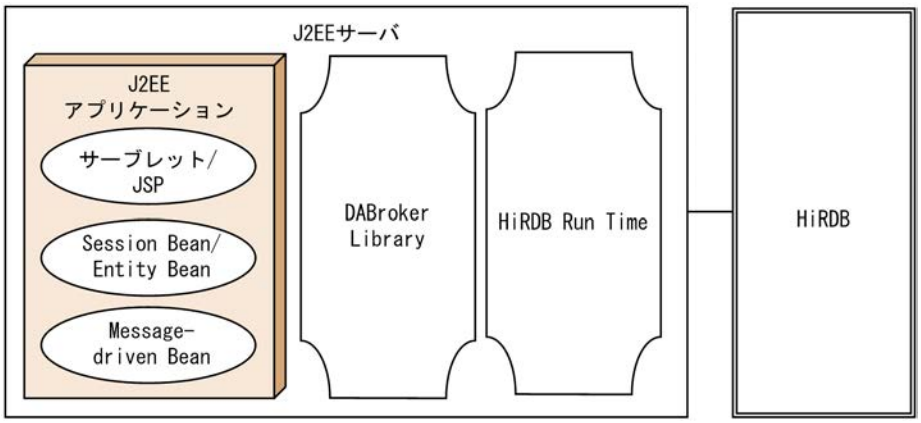
4.8.4　データベースにアクセスする場合の構成

DB Connector を使用して，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean からデータベースにアクセスする場合の構成を次に示します。

- HiRDB にアクセスする場合の構成

  HiRDB に接続する場合，J2EE サーバと同じマシンに，HiRDB Run Time が必要となります。HiRDB にアクセスする場合の構成を次の図に示します。

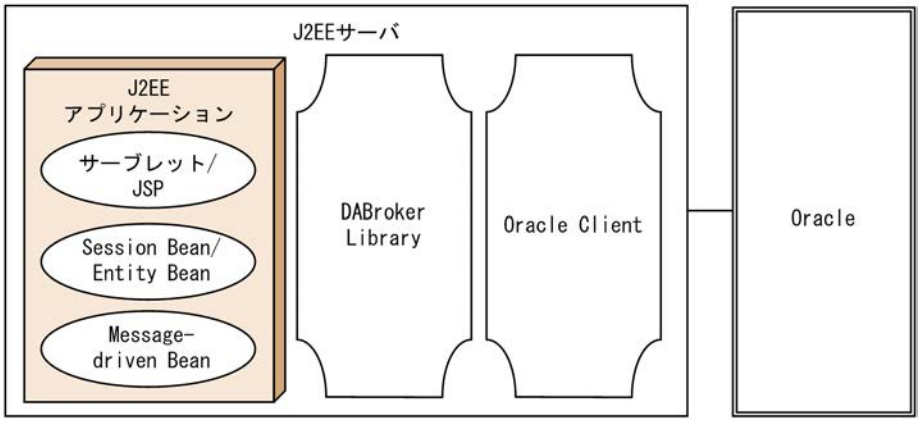
  図 4-6　リソースアダプタを使用して HiRDB にアクセスする場合の構成



- Oracle にアクセスする場合の構成

  Oracle に接続する場合，J2EE サーバと同じマシンに Oracle Client が必要となります。Oracle にアクセスする場合の構成を次の図に示します。

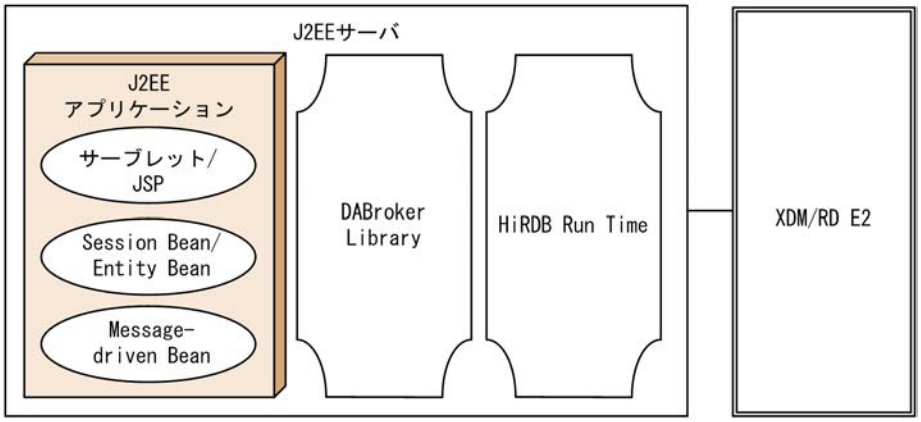
図 4-7　リソースアダプタを使用して Oracle にアクセスする場合の構成



- XDM/RD E2 にアクセスする場合の構成

  XDM/RD E2 に接続する場合，J2EE サーバと同じマシンに HiRDB Run Time が必要となります。XDM/RD E2 にアクセスする場合の構成を次の図に示します。

  図 4-8　リソースアダプタを使用して XDM/RD E2 にアクセスする場合の構成（DABroker Library を使用するとき）



4.8.5　DB Connector のインポート

DB Connector のインポートは，cjimportres コマンドで実行します。インポートする RAR ファイルは，DBConnector_DABJ_CP.rar または DBConnector_DABJ_XA.rar です。

インポート手順については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2.1　DB Connector のインポート」を参照してください。

4.8.6　DB Connector のプロパティ定義

DB Connector のプロパティ定義の手順については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2.2　DB Connector のプロパティ定義」を参照してください。

ここでは，次の DB Connector について，コンフィグレーションプロパティの設定例を示します。

- DBConnector_DABJ_CP.rar で，Oracle，HiRDB，または XDM/RD E2 を使用する場合
- DBConnector_DABJ_XA.rar で，Oracle または HiRDB を使用する場合

(1)　DBConnector_DABJ_CP.rar で，Oracle，HiRDB，または XDM/RD E2 を使用する場合の設定例

DBConnector_DABJ_CP.rar で，Oracle，HiRDB，または XDM/RD E2 を使用する場合の，コンフィグレーションプロパティの設定例を，次の表に示します。

表 4-68　データベースとして Oracle，HiRDB，または XDM/RD E2 を使用する場合のコンフィグレーションプロパティの設定例（DBConnector_DABJ_CP.rar の場合）

項目名	Oracle の場合の設定例	HiRDB の場合の設定例	XDM/RD E2 の場合の設定例
networkProtocol	lib	lib	lib
databaseName	ORACLE8I※1	HIRDB	HIRDB
description	＜ネット・サービス名＞	＜ HiRDB ポート番号＞※2	＜データベースコネクションサーバのサーバスケジュール番号＞※2
DBHostName	－	＜ HiRDB ホスト名＞	＜ XDM/RD E2 ホスト名＞
loginTimeout	0	0	0
serverName	－	－	－
portNumber	40179	40179	40179
DBEnv	－	－	－
encodLang	－	－	－
JDBC_IF_TRC	false	false	false
SV_EVENT_TRC	false	false	false
TRC_NO	500	500	500
uapName	－	－	－
bufSize	64※3	64※3	64※3
rowSize	16	16	16
OSAuthorize	false	false	false
HiRDBCursorMode	false	false	false
blockUpdate	false	false	false
executeDirectMode	false	false	false
SQLWarningIgnore	false	false	false
LONGVARBINARY_Access	REAL	REAL	REAL
bufferPoolSize	0	0	0
PreparedStatementPoolSize	10	10	10※4
CallableStatementPoolSize	10	10	10※4

項目名	Oracle の場合の設定例	HiRDB の場合の設定例	XDM/RD E2 の場合の設定例
CancelStatement	true	true	false
ConnectionIDUpdate	false	false	false
logLevel	ERROR	ERROR	ERROR

（凡例）　－：設定は不要

注※1　DABroker Library の環境設定で，使用する Oracle のバージョンに Oracle9i が設定されていれば，Oracle9i に接続できます。また，使用する Oracle のバージョンに Oracle10g が設定されていれば，Oracle10g に接続できます。

注※2　HiRDB クライアントの環境変数グループ名（Windows の場合）または環境変数グループの設定ファイルのパス（UNIX の場合）を指定することもできます。

注※3　次のどちらかのタイミングで，「bufSize に指定した値 × 生成したコネクション数分のメモリ」を Java ヒープ領域に確保します。

・コネクションプールのウォーミングアップ機能使用時

・初回 getConnection()発行時

bufSize に必要以上に大きな値を設定すると，メモリを確保したときに Java ヒープを使い切り，OutOfMemoryError が発生するおそれがあります。HiRDB の BLOB 型データや BINARY 型データへアクセスする場合は，bufSize に大きな値を指定しないで，HiRDB の位置付け子機能を使用することを検討してください。HiRDB の位置付け子機能については，マニュアル「HiRDB 解説」を参照してください。

注※4　XDM/RD E2 11-02 より前のバージョンに接続する場合は，0 を設定してください。

(2)　DBConnector_DABJ_XA.rar で，Oracle または HiRDB を使用する場合の設定例

DBConnector_DABJ_XA.rar で，Oracle または HiRDB を使用する場合の，コンフィグレーションプロパティの設定例を，次の表に示します。

表 4-69　データベースとして Oracle または HiRDB を使用する場合のコンフィグレーションプロパティの設定例（DBConnector_DABJ_XA.rar の場合）

項目名	Oracle の場合の設定例	HiRDB の場合の設定例
networkProtocol	lib	lib
databaseName	ORACLE8I※1	HIRDB
description	＜ネット・サービス名＞	＜環境変数グループ識別子＞※2
DBHostName	－	＜ HiRDB ホスト名＞
XAOpenString	Oracle_XA+DB=＜ネット・サービス名＞+SqlNet=＜ネット・サービス名＞+Acc=P/＜ DB のユーザ名※3 ＞/＜ DB のパスワード※4 ＞ +SesTm※5=300+Threads=true	Windows の場合 ＜環境変数グループ識別子※6 ＞ +＜環境変数グループ名※7 ＞ UNIX の場合 ＜環境変数グループ識別子※6 ＞ +＜環境変数グループの設定ファイルのパス※7 ＞
loginTimeout	0	0
serverName	－	－
portNumber	40179	40179

項目名	Oracle の場合の設定例	HiRDB の場合の設定例
DBEnv	−	−
encodLang	−	−
JDBC_IF_TRC	false	false
SV_EVENT_TRC	false	false
TRC_NO	500	500
uapName	−	−
bufSize	64※8	64※8
rowSize	16	16
OSAuthorize	false	false
HiRDBCursorMode	false	false
blockUpdate	false	false
executeDirectMode	false	false
SQLWarningIgnore	false	false
LONGVARBINARY_Access	REAL	REAL
bufferPoolSize	0	0
XACloseString	−	−
RMID	1	1※9
XAThreadMode	true	true
XALocalCommitMode	true	true
PreparedStatementPoolSize	10	10
CallableStatementPoolSize	10	10
CancelStatement	true	false
ConnectionIDUpdate	false	false
logLevel	ERROR	ERROR

（凡例）　−：設定は不要

注※1　DABroker Library の環境設定で，使用する Oracle のバージョンに Oracle9i が設定されていれば，Oracle9i に接続できます。また，使用する Oracle のバージョンに Oracle10g が設定されていれば，Oracle10g に接続できます。

注※2　J2EE サーバ内でユニークな 4 バイトの文字列を指定します。

注※3　［User］フィールドに入力した値を指定します。

注※4　［Password］フィールドに入力した値を指定します。

注※5　XA Open String の SesTm に指定する値は，トランザクションタイムアウトの値よりも必ず大きく設定してください。

注※6　［Description］フィールドに入力した値を指定します。

注※7　HiRDB の環境変数グループ名（Windows の場合）または環境変数グループの設定ファイルのパス（UNIX の場合）を指定します。詳細については，「4.3.3　データベースの設定」を参照してください。

注※8　次のどちらかのタイミングで，「bufSize に指定した値 × 生成したコネクション数分のメモリ」を Java ヒープ領域に確保します。

・コネクションプールのウォーミングアップ機能使用時

・初回 getConnection()発行時

bufSize に必要以上に大きな値を設定すると，メモリを確保したときに Java ヒープを使い切り，OutOfMemoryError が発生するおそれがあります。HiRDB の BLOB 型データや BINARY 型データへアクセスする場合は，bufSize に大きな値を指定しないで，HiRDB の位置付け子機能を使用することを検討してください。HiRDB の位置付け子機能については，マニュアル「HiRDB 解説」を参照してください。

注※9　リソースマネジャの識別子を指定します。J2EE サーバ内でユニークな 1～2147483647 の数値で指定します。

(3)　ステートメントプーリングの設定についての注意事項

DABroker Library を使用する場合にステートメントプーリングを設定するときの注意事項について説明します。なお，ステートメントプーリングの機能および設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.4　ステートメントプーリング」および「3.14.10　実行環境での設定」を参照してください。

(a)　PreparedStatement，CallableStatement のプールサイズの制限

DABroker Library を使用する場合，PreparedStatement，CallableStatement のプールサイズに制限があります。

- PreparedStatementPoolSize と CallableStatementPoolSize の合計値は，DABroker の最大ステートメント数以下に設定してください。

  合計値が最大ステートメント数を超えた場合には，PreparedStatementPoolSize と CallableStatementPoolSize にデフォルト値が使用されます。PreparedStatementPoolSize，CallableStatementPoolSize は，リソース数が最大値を超えないように設定してください。一つのコネクションで利用できるリソース（Statement，PreparedStatement，CallableStatement，DatabaseMetaData から取得した ResultSet）の総数は，最大値が決められています。リソース数の最大値については，「4.12　Connection クラス」を参照してください。

  最大値を超えた場合には，ユーザアプリケーションプログラムに例外が通知されます。例外が通知された場合，次のように対処してください。

  - 「PreparedStatementPoolSize」，「CallableStatementPoolSize」の値を小さくする。
  - ユーザアプリケーションプログラムで，リソースが不要になったらすぐに解放するようにする。

  指定できるプールサイズについては，「4.31.1　DB Connector に設定する<config-property>タグに指定できるプロパティ」を参照してください。

(b)　XDM/RD E2 に接続する場合のバージョン

XDM/RD E2 に接続するする場合は，XDM/RD E2 11-02 以降のバージョンと DABroker Library 03-13 以降を使用する場合にだけステートメントプーリング機能を使用できます。

(c)　Oracle に接続する場合にステートメントプーリング機能とコネクションの障害検知機能を併用するとき

コネクションの障害検知に使用される PreparedStatement がコネクションごとに一つプーリングされます。そのため，PreparedStatementPoolSize，CallableStatementPoolSize を決定する際には，コネクションの障害検知用の PreparedStatement もリソース数の見積もりに加えて，CallableStatementPoolSize を最大ステートメント数未満に設定してください。例えば，DABroker Library を使用している場合，PreparedStatementPoolSize=0，CallableStatementPoolSize=＜最大ス

テートメント数＞と設定して，プーリングされている CallableStatement の数が最大ステートメント数に達したとき，コネクションの障害検知が実行されると，DABroker Library の一つのコネクションで利用できるリソース数の最大値を超えるため，例外が発生します。例外が発生すると障害が発生したと判断されるので，そのコネクションはコネクションプールから削除され，同時にステートメントプールも破棄されます。つまり，ステートメントプーリング機能を使用する意味がなくなってしまいます。したがって，コネクションの障害検知機能を使用する場合には，CallableStatementPoolSize を最大ステートメント数未満に設定してください。

4.8.7　データベースと接続するときの注意事項

ここでは，DABroker Library を使用して HiRDB または Oracle に接続する場合の注意事項を示します。

(1)　HiRDB と接続する場合の注意事項

HiRDB と接続する場合の注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.6.5　HiRDB と接続する場合の前提条件と注意事項」を参照してください。

(a)　SQL の前処理保持機能

commit()メソッドをわたっても※保持される PreparedStatement オブジェクトは，DELETE 文，INSERT 文，PURGE 文，TABLE 文，UPDATE 文，CALL 文，および SELECT 文で生成したものだけです。ただし，HiRDB クライアント経由で XDM/RD E2 を使用する場合に PURGE TABLE 文で生成した PreparedStatement オブジェクトは除きます。

これらの SQL 以外で生成された PreparedStatement オブジェクトは，commit()メソッド実行時に無効となります。無効になった PreparedStatement オブジェクトで SQL を実行するとエラーになります。実行例を次に示します。

実行例

```
PrepareStatement pstmt1 = con.prepareStatement(""lock table tb1"");
PrepareStatement pstmt2 = con.prepareStatement(""lock table tb2"");
pstmt1.execute();  //エラーにならない
con.commit();
pstmt2.execute();  //エラーになる
pstmt1.close();
pstmt2.close();
```

注※　commit()メソッドによる明示的なコミットのほかに，次の処理も該当します。

- 自動コミットによる暗黙的なコミット
- 定義系 SQL の実行
- PURGE TABLE 文の実行
- rollback メソッドによる明示的なロールバック
- SQL 実行エラーによる暗黙的なロールバック

(2)　Oracle と接続する場合の注意事項

ここでは，DABroker Library を使用して Oracle に接続する場合の注意事項について説明します。

(a)　使用できる J2EE コンポーネント

DABroker Library では，次のコンポーネントを使用できます。

- サーブレット／JSP

- Stateless Session Bean
- Stateful Session Bean
- Entity Bean（BMP）
- Entity Bean（CMP2.0）
- Message-driven Bean

(b)　型マッピング

DABroker Library を使用する場合の，Java の型と Oracle の SQL 型のマッピングを次の表に示します。

表 4-70　Java の型と Oracle の SQL 型のマッピング

メソッド	Oracle の SQL 型									
	CHAR	VARCHAR2	NUMBER	DATE	TIMESTAMP	CLOB	LONG	BLOB	RAW	LONG RAW
set/getByte	○	○	○	−	−	−	○	−	−	−
set/getShort	○	○	○	−	−	−	○	−	−	−
set/getInt	○	○	○	−	−	−	○	−	−	−
set/getLong	○	○	○	−	−	−	○	−	−	−
set/getFloat	○	○	○	−	−	−	○	−	−	−
set/getDouble	○	○	○	−	−	−	○	−	−	−
set/getBigDecimal	○	○	○	−	−	−	○	−	−	−
set/getBoolean	−	−	−	−	−	−	−	−	−	−
set/getString	○	○	−	○	−	−	○	−	−	−
set/getBytes	−	−	−	−	−	−	−	−	○	○
set/getDate	○	○	−	○	−	−	○	−	−	−
set/getTime	○	○	−	○	−	−	○	−	−	−
set/getTimestamp	○	○	−	○	−	−	○	−	−	−
set/getClob	−	−	−	−	−	○	−	−	−	−
set/getBlob	−	−	−	−	−	−	−	○	−	−
set/getObject	○	○	○	○	−	○	○	○	○	○

（凡例）

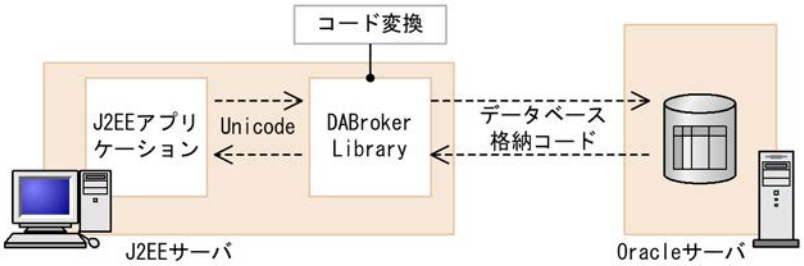
○：java.sql.PreparedStatement#setXXX および java.sql.ResultSet#getXXX などでマッピングできる

−：マッピングできない

(c)　日本語文字コード変換のための設定

DABroker Library を使用する場合に文字コード変換が実施される場所を次の図に示します。

図 4–9　DABroker Library を使用する場合に文字コード変換が実施される場所



J2EE サーバ上の JDBC ドライバ部分では，JavaVM のコンバータによって，Unicode からデータベース格納コード，およびデータベース格納コードから Unicode への変換が実施されます。

DABroker Library のエンコーディングセットと Oracle サーバのデータベースキャラクタセットを同じにしておく必要があります。

(d)　データソースの設定

Oracle に対して XA 接続でアクセスする場合，データソースの設定では，次の規則で XA Open String を指定する必要があります。

- 指定する文字列は，Oracle のマニュアルに記載されている規則に従って指定してください。キーワードなどの区切り文字が不正な場合は，Oracle ライブラリ内でアプリケーションエラーが発生する場合がありますのでご注意ください。
- 指定する文字列には必ず，「Threads=true」を指定してください。「Threads=true」が指定されていない場合は，Oracle ライブラリ内でアプリケーションエラーが発生，または SQL 実行時にエラーが発生することがありますのでご注意ください。

4.9　DB Connector for Reliable Messaging を使用したデータベース上のキューへの接続

Reliable Messaging を使用したメッセージの非同期通信をする場合，DB Connector for Reliable Messaging を使用してデータベース上のキューに接続します。

ここでは，DABroker Library を使用する場合のデータベース上のキューとの接続について説明します。

機能の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.7　データベース上のキューとの接続」を参照してください。また，DB Connector for Reliable Messaging の設定の詳細については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.1　リソースアダプタの設定の概要」，およびマニュアル「Reliable Messaging」の「2.7　DB Connector for Reliable Messaging の機能」を参照してください。

4.9.1　DB Connector for Reliable Messaging で接続できるデータベース

DABroker Library を使用する場合，DB Connector for Reliable Messaging と Reliable Messaging を使用して接続できるデータベースは HiRDB です。

4.9.2　DB Connector for Reliable Messaging（RAR ファイル）の種類

RAR ファイルは，サーバ管理コマンドを使用して操作します。サーバ管理コマンドを使用して RAR ファイルを操作する方法については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」を参照してください。

DABroker Library で使用できる DB Connector for Reliable Messaging の RAR ファイルの種類を次の表に示します。

表 4-71　DABroker Library で使用できる DB Connector for Reliable Messaging の RAR ファイルの種類

RAR ファイル	接続できるデータベース	説明
DBConnector_DABJ_CP_Cosminexus_RM.rar	HiRDB	トランザクション管理をしない場合，またはローカルトランザクションを使用する場合（トランザクションのサポートレベルに NoTransaction または LocalTransaction を指定する場合）に使用します。
DBConnector_DABJ_XA_Cosminexus_RM.rar※	HiRDB	グローバルトランザクションを使用する場合（トランザクションのサポートレベルに XATransaction を指定する場合）に使用します。

注※　ライトトランザクション機能は有効にできません。

RAR ファイルの種類に応じて，指定できるトランザクションサポートレベルが異なります。また，ライトトランザクションを使用できるかどうかも異なります。

RAR ファイルごとに指定できるトランザクションサポートレベルを次の表に示します。

表 4-72　RAR ファイルごとに使用できるトランザクションサポートレベル(DB Connector for Reliable Messaging)

使用する DB Connector for Reliable Messaging (RAR ファイル)	トランザクションサポートレベル	ライトトランザクション	
		有効	無効
DBConnector_DABJ_CP_Cosminexus_RM.rar	NoTransaction LocalTransaction	○	○
DBConnector_DABJ_XA_Cosminexus_RM.rar	XATransaction	−	○

（凡例）　○：使用できる　　−：該当しない

4.9.3　キューに接続する場合に必要なモジュール

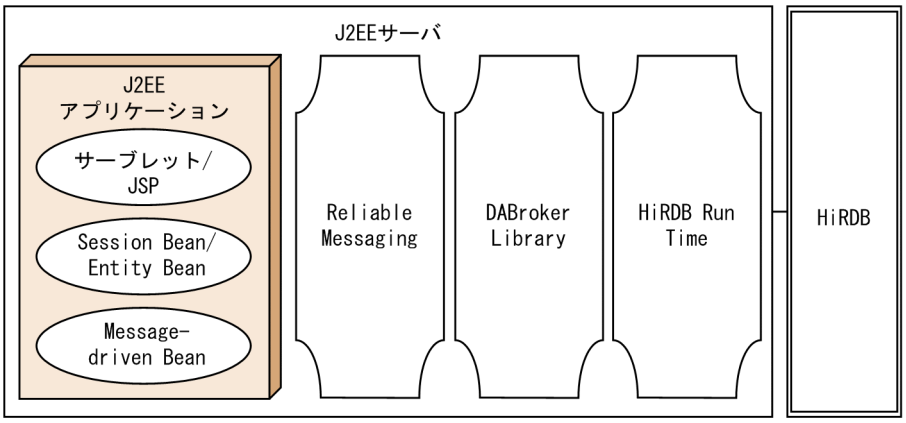
DABroker Library を使用してデータベースのキューに接続する場合に必要なモジュールは，DB Connector の場合と同じです。「4.8.3　必要なモジュール」を参照してください。

4.9.4　データベースにアクセスする場合の構成

DB Connector for Reliable Messaging を使用して，サーブレット，JSP，Session Bean，Entity Bean または Message-driven Bean からデータベースにアクセスする場合の構成を次に示します。

Reliable Messaging を使用してデータベースにアクセスする場合の構成を次の図に示します。

図 4-10　Reliable Messaging を使用してデータベースにアクセスする場合の構成



4.9.5　DB Connector for Reliable Messaging のインポートおよびプロパティ定義

DB Connector for Reliable Messaging のインポートおよびプロパティ定義の手順については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.4.1　リソースアダプタのインポート」および「4.4.2　リソースアダプタのプロパティ定義（Connector 1.0 の場合)」を参照してください。

4.9.6　データベース上のキューと接続するときの注意事項

DB Connector for Reliable Messaging と Reliable Messaging を使用してデータベースに接続する場合の注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.7.4　接続できるデータベース」を参照してください。

4.10　DABroker Library で使用する API の一覧

DABroker Library で使用する API の一覧を，次の表に示します。

表 4-73　DABroker Library で使用する API の一覧

クラス名	主な機能
Driver クラス	• データベースの接続 • 指定された URL の妥当性チェック • DriverManager.getConnection メソッドで指定する接続プロパティの情報取得 • ドライバのバージョン情報の返却
Connection クラス	• Statement クラス，PreparedStatement クラス，および CallableStatement クラスのオブジェクト生成 • トランザクションのコミットまたはロールバック • オートコミットモードの設定
Statement クラス	• SQL の実行 • 検索結果として，ResultSet（ResultSet オブジェクト）の生成 • 更新結果として，更新行数の返却 • 最大検索行数の設定 • 検索制限時間の設定
PreparedStatement クラス	• ?パラメタ付き SQL の実行 • ?パラメタの設定 • 検索結果として，ResultSet オブジェクトの生成および返却 • 更新結果として，更新行数の返却
CallableStatement クラス	• ストアドプロシジャの実行 • INPUT および INOUT パラメタの設定（PreparedStatement クラスの setXXX メソッドを使用） • INOUT および OUTPUT パラメタの登録 • INOUT および OUTPUT パラメタ値の取得
ResultSet クラス	• 行単位の ResultSet 内の移動 • 結果データの返却 • 検索結果データが NULL 値かどうかの通知
ResultSetMetaData クラス	• ResultSet の各列に対する，データ型およびデータ長のメタ情報の返却
DatabaseMetaData クラス	• 接続データベースに関する各種情報の返却 • 表一覧，列一覧などの一覧系情報を，ResultSet に格納して返却
DataSource クラス	• データベースの接続情報の設定および取得
Blob インタフェース	• Blob データの取得

4.11　Driver クラス

説明

Driver クラスでは，主に次の機能を提供します。

- データベースの接続
- 指定された URL の妥当性チェック
- DriverManager.getConnection メソッドで指定する接続プロパティの情報取得
- ドライバのバージョン情報の返却

Driver クラスの提供する各メソッドの詳細と使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

Driver クラスを使用したデータベース接続の設定については，「4.7.4　データベース接続の設定」を参照してください。

制限事項

DABroker Library で使用する Driver クラスの制限事項を，次に示します。

表 4-74　Driver クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
acceptURL	URL 構文の必須項目が設定されている場合は true を返却します。必須項目が不足している場合は false を返却します。	○	○
getPropertyInfo	引数 info に与えられた以外に設定可能なプロパティ情報の配列を返却します。	○	○
jdbcCompliant	無条件に true を返却します。	○	○

（凡例）

○：該当します。

4.12　Connection クラス

説明

Connection クラスでは，主に次の機能を提供します。

- Statement クラス，PreparedStatement クラス，および CallableStatement クラスのオブジェクト生成
- トランザクションのコミットまたはロールバック
- オートコミットモードの設定

Connection クラスの提供する各メソッドの詳細，および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

注意事項

- Connection のクローズ

  Connection の close メソッドを発行しないでアプリケーションを終了すると，DABroker Library 側にリソースが残る場合があります。このため，SQL 実行などでエラーの発生によって終了する場合は，catch ブロックまたは finally ブロック内で close メソッドを発行してください。

  Connection の close メソッドの発行でエラーが発生した場合，SQLException をスローしません。また，プーリング使用時（ConnectionPoolDataSource を使用した接続），および XA 使用時（XADataSource を使用した接続）は Connection の close メソッド実行でデータベースとの物理的な切断はしません。

- カタログ

  DABroker Library では，接続データベース種別にかかわらず，カタログをサポートしていません。このため，getCatalog メソッドは無条件に null を返却し，setCatalog メソッドは何もしません。

- アクセスモード

  DABroker Library では，アクセスモードの変更をサポートしていないため，isReadOnly メソッドは無条件に false を返却し，setReadOnly メソッドは何もしません。

- トランザクション分離モード

  DABroker Library では，トランザクション分離モードの変更をサポートしていません。このため，getTransactionIsolation メソッドでは，常に TRANSACTION_READ_UNCOMMITTED を返却し（データベースの仕様を表しているものではありません），setTransactionIsolation メソッドは TRANSACTION_READ_UNCOMMITTED 以外が指定された場合，SQLException をスローします。

- データベースアクセスリソース数の制限

  DABroker Library を利用するアプリケーションでは，一つのコネクションで利用できるリソース数の最大は 1024 個です。ここでいうリソースとは，Statement クラス，PreparedStatement クラス，CallableStatement クラスのオブジェクト，および DatabaseMetaData クラスの中で一覧系情報を ResultSet に格納するメソッドの同時実行数を指します。

- オートコミット

  次のすべての条件に一致する更新 SQL（INSERT，UPDATE および DELETE）を実行する場合は，事前に検索中の ResultSet オブジェクトまたは検索中の ResultSet オブジェクトを生成した Statement（継承したクラスも含みます）オブジェクトに対して close メソッドを実行してください。

1. コネクションプーリングを使用してデータベースに接続している，または XA を用いてデータベースに接続している場合

2. 接続データベースが HiRDB の場合
3. ホールダブルカーソルを使用していない場合
4. Statement クラスの setExecuteDirectMode メソッドを，引数 Mode に false を指定して実行している場合
5. オートコミットを有効にしている場合

参考

close メソッドを実行しないで更新 SQL を実行した場合，更新 SQL 実行後 ResultSet オブジェクトをクローズした時に HiRDB でエラーが発生することがあります。この場合，SQLException によるエラーの通知はされませんが，HiRDB のエラーログ，SQL トレースおよび DABroker Library の拡張データベースアクセストレースに HiRDB のエラーメッセージ（KFPA11501-E）を出力します。

制限事項

DABroker Library で使用する Connection クラスの制限事項を，次に示します。

表 4-75　Connection クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
close	通常接続時はデータベースとの接続を解除します。プーリング使用時（ConnectionPoolDataSource を使用した接続），および XA 使用時（XADataSource を使用した接続）は物理的な切断はしません。Connection.close メソッドの実行でエラーが発生した場合は SQLException をスローしません。	○	○
createStatement	更新結果を反映する ResultSet をサポートしていないため，ResultSet タイプに TYPE_SCROLL_SENSITIVE を指定した場合，TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。また，並行処理タイプに CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。	−	○
getCatalog	無条件に null を返却します。	○	○
getTransactionIsolation	無条件に TRANSACTION_READ_UNCOMMITTED を返却します。	○	○
getTypeMap	ユーザ定義型をサポートしていないため，無条件に SQLException をスローします。	−	○
isReadOnly	無条件に false を返却します。	○	○
prepareCall	更新結果を反映する ResultSet をサポートしていないため，ResultSet タイプに TYPE_SCROLL_SENSITIVE を指定した場合，TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。また，並行処理タイプに CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。	−	○
prepareStatement	更新結果を反映する ResultSet をサポートしていないため，ResultSet タイプに TYPE_SCROLL_SENSITIVE を指定した場合，TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。また，並行処理タイプに	−	○

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
prepareStatement	CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。	－	○
setCatalog	使用できません。	○	○
setReadOnly	使用できません。	○	○
setTransactionIsolation	使用できません。	○	○
setTypeMap	ユーザ定義型をサポートしていないため，無条件に SQLException をスローします。	－	○

（凡例）

○：該当します。

－：該当しません。

4.13　Statement クラス

説明

Statement クラスでは，主に次の機能を提供します。

- SQL の実行
- 検索結果として，ResultSet（ResultSet オブジェクト）の生成
- 更新結果として，更新行数の返却
- 最大検索行数の設定
- 検索制限時間の設定

Statement クラスの提供する各メソッドの詳細，および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

注意事項

- マルチスレッド

  一つの Statement オブジェクトを複数のスレッドで使用する場合，「SQL の実行～ResultSet の取得～ResultSet のクローズ」という一連の処理を，スレッドごとにシリアライズする必要があります。シリアライズしないで並行して処理した場合の動作は保証しません。

  各スレッドには，それぞれ別の Statement オブジェクトを割り当ててください。

- 複数の ResultSet

  DABroker Library では，複数の ResultSet を返却する機能をサポートしていません。このため，getMoreResults メソッドは無条件に false を返却し，現在オープンしている ResultSet が存在する場合，その ResultSet をクローズします。

- カーソル名称

  DABroker Library では，位置決めされた更新および削除をサポートしていないため，setCursorName メソッドは何もしません。

- 検索制限時間

  DABroker Library では，検索の時間監視はできません。このため，setQueryTimeout メソッドで指定した値は無効となります。

- setFetchSize メソッドの使用について

  setFetchSize メソッドが有効になるのは，通常の検索の場合だけです。ResultSetMetaData や DatabaseMetaData の情報取得では有効になりません。

  CLOB や BLOB などの LONGVARCHAR・LONGVARBINARY 型データを含むテーブルの検索では，setFetchSize メソッドの設定値は有効になりません。

  setFetchSize メソッドを使用する場合，データ取得用のバッファサイズは setFetchSize メソッドの指定値，および検索するカラムの定義長などによって変化します。そのため，setFetchSize メソッドでの指定値が大き過ぎる場合や検索するカラムの定義長の合計が極端に大きくなる場合，メモリ不足が発生する可能性があるので注意が必要です。

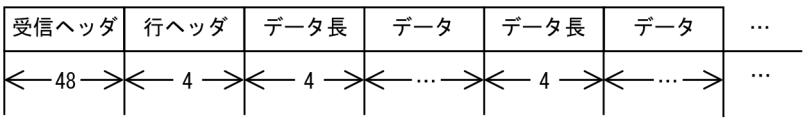
  setFetchSize メソッドでの指定値が 1 以上の場合，必要となるバッファ領域は自動的に確保されます。この場合，どの程度の領域を必要とするかの計算方法を，次の表に示します。

  バッファ領域に指定できる最大値は 2147483647 バイトです。最大値を超えないようにしてください。

  ただし，DABroker Library の最小バッファサイズが 64 キロバイトのため，領域サイズが 64 キロバイトより小さい場合は，64 キロバイトに設定されます。また，領域サイズが接続時にプロパティ

の BUF_SIZE，または DataSource の setBufSize メソッドで指定した値を超えた場合は，setFetchSize の指定値に必要な領域サイズが有効となります。

表 4–76　setFetchSize メソッド使用で必要となるバッファ領域

DBMS	データ型	必要となるバッファ領域（単位：バイト）
共通	 受信ヘッダ ＋（（行ヘッダ ＋ データ長 ＋ データ ＋ …）＊ 検索件数）	
HiRDB	int，integer，smallflt，real	4
	date	10
	smallint	2
	dec，decimal(m.n)	(m ＋ 1) / 2
	float，double precision	8
	char(n)，mchar(n)，rowid(n)，varchar(n)，mvarchar(n)	n
	nchar(n)，nvarchar(n)	n * 2
	time	8
	timestamp(n)	7 ＋ n / 2
ORACLE	varchar2(n)，char(n)，raw(n)，mlslabel(n)	n
	number	22
	float，rowid	8
	date	20

（例 1）

HiRDB 接続で FetchSize=3，検索カラムが varchar(10)，integer の場合

受信ヘッダ(48)+((行ヘッダ(4)+データ長(4)+varchar データ(10)+データ長(4)+integer データ(4))*3)

となり，必要となるバッファ領域は 126 バイトとなります。この場合，合計が 64 キロバイト以下のため，確保される領域は 64 キロバイトになります。

（例 2）

HiRDB 接続で FetchSize=100，検索カラムが varchar(1000)，integer の場合

受信ヘッダ(48)+((行ヘッダ(4)+データ長(4)+varchar データ(1000)+integer データ(4))*100)

となり，101248 バイトのバッファ領域が確保されます。確保される領域が，プロパティの BUF_SIZE，または DataSource の setBufSize の指定値を超えた場合は，setFetchSize の指定値に必要な領域サイズが有効となります。

- Statement のクローズ

  プーリング使用時（ConnectionPoolDataSource を使用した接続），および XA 使用時（XADataSource を使用した接続）で Statement の close メソッドの発行でエラーが発生した場合，SQLException をスローしません。また，プーリング使用時，および XA 使用時で Statement の close メソッド実行中にデータベースとの物理的な切断でエラーが発生してコネクションプーリ

ングが使用できなくなった場合，ConnectionEventListener.connectionErrorOccurred は発生しません。

- 非同期キャンセル

  接続データベースが HiRDB または ORACLE の場合，cancel メソッドを使用して非同期キャンセルを発行できます。ただし，XADataSource を使用した接続の場合，非同期キャンセルの実行は接続データベースの仕様に従います（HiRDB は XADataSource を使用した接続時の非同期キャンセルをサポートしていません。そのため非同期キャンセル要求は有効になりません）。

  また，HiRDB クライアント経由 XDM/RD E2 接続時の非同期キャンセルはサポートしていません。

制限事項

DABroker Library で使用する Statement クラスの制限事項を，次に示します。

表 4-77　Statement クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
cancel	接続データベース種別が HiRDB または ORACLE の場合だけ有効です。	○	○
close	プーリング使用時（ConnectionPoolDataSource 使用した接続），および XA 使用時（XADataSource を使用した接続）で Statement.close メソッド実行中にエラーが発生した場合，SQLException をスローしません。また，プーリング使用時，および XA 使用時で Statement.close メソッド実行中にデータベースとの物理的な切断でエラーが発生してコネクションプーリングが使用できなくなった場合，ConnectionEventListener.connectionErrorOccurred は発生しません。	○	○
getFetchSize	0，または setFetchSize メソッドで指定された値が setMaxRows メソッドで指定された値より小さい場合は setFetchSize メソッドで指定された値を，setFetchSize メソッドで指定された値が setMaxRows メソッドで指定された値より大きい場合は setMaxRows メソッドで指定された値を返却します。	－	○
getMaxFieldSize	setMaxFieldSize が指定されていない場合 0 を返却します。	○	○
getMaxRows	setMaxRows が指定されていない場合は 0 を返却します。	○	○
getMoreResults	無条件に結果なし=false を返却します。	○	○
setCursorName	使用できません。	○	○
setFetchDirection	FETCH_FORWARD 以外を指定した場合，SQLException をスローします。	－	○
setFetchSize	接続データベース種別が HiRDB，または ORACLE の場合で，CLOB や BLOB などの LONG VARCHAR・LONG VARBINARY 型データ列の検索を含まないときだけ有効となります。	－	○
setMaxFieldSize	0 より小さい値を指定した場合，SQLException をスローします。	○	○
setMaxRows	0 より小さい値を指定した場合，SQLException をスローします。	○	○

（凡例）

○：該当します。

－：該当しません。

DABroker Library 提供メソッド

Statement クラスで提供している，DABroker Library だけの機能を次に示します。

メソッド一覧

メソッド名	機能
getExecuteDirectMode メソッド	接続データベースが HiRDB の場合，Statement クラスを使用したデータベースの更新で，HiRDB の Execute Direct 機能を使用するかどうかの設定情報を取得します。
setExecuteDirectMode メソッド	接続データベースが HiRDB の場合，Statement クラスを使用したデータベースの更新で，HiRDB の Execute Direct 機能を使用するかどうかを設定します。

getExecuteDirectMode メソッド

説明

接続データベースが HiRDB の場合，INSERT，UPDATE，DELETE など Statement クラスを使用したデータベースの更新で，HiRDB の Execute Direct 機能を使用するかどうかの設定情報を取得します。

形式

```
public boolean getExecuteDirectMode();
```

パラメタ

なし

例外

なし

戻り値

boolean：

Execute Direct 機能を使用するかどうかを次の値で取得します。

- true
  Execute Direct 機能を使用します。
- false
  Execute Direct 機能を使用しません。

setExecuteDirectMode メソッド

説明

接続データベースが HiRDB の場合，INSERT，UPDATE，DELETE など Statement クラスを使用したデータベースの更新で，HiRDB の Execute Direct 機能を使用するかどうかを設定します。

このメソッドが呼び出されない場合は，false を設定します。

なお，この指定は接続データベースが HiRDB の場合だけ有効です。HiRDB 以外のデータベースの場合は，指定の有無にかかわらず Execute Direct 機能を使用しません。

形式

```
public void setExecuteDirectMode(boolean Mode);
```

パラメタ

Mode：

HiRDB の Execute Direct 機能を使用するかどうかを次の値で設定します。

- true
  Execute Direct 機能を使用します。
- false
  Execute Direct 機能を使用しません。

例外

なし

戻り値

なし

4.14　PreparedStatement クラス

説明

PreparedStatement クラスでは，主に次の機能を提供します。

- パラメタ付き SQL の実行
- パラメタの設定
- 検索結果として，ResultSet オブジェクトの生成および返却
- 更新結果として，更新行数の返却

PreparedStatement クラスの提供する各メソッドの詳細および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

なお，PreparedStatement クラスは Statement クラスのサブクラスです。Statement クラスの機能をすべて継承します。

注意事項

- PreparedStatement クラスは Statement クラスのサブクラスであるため，Statement クラスの注意事項はすべて該当します。Statement クラスの注意事項を参照してください。
- HiRDB を使用する場合での TIME 型，DATE 型，TIMESTAMP 型の列に対するデータ変換処理

  TIME 型，DATE 型，または TIMESTAMP 型の列に対して，setTime，setDate，setTimestamp および setString メソッドを使用してデータが設定された場合，HiRDB のデータ型に応じてデータが変換されます。列のデータ型と，メソッドとの組み合わせによる，変換処理について次の表に示します。なお，実際に存在しない日時を指定した場合は，JavaVM の返す値となります。

表 4–78　TIME 型，DATE 型，TIMESTAMP 型と setXXX メソッドとの変換

setXXX メソッド	HiRDB 列属性		
	TIME	DATE	TIMESTAMP
setTime(Time Obj)※1	設定値をデータベースに格納します。	HiRDB のエラーを返します。	設定値 hh:mm:ss[.000000]の前に 1970-01-01 を付加したデータをデータベースに格納します。
setDate(Date Obj)※2	HiRDB のエラーを返します。	設定値をデータベースに格納します。	設定値 yyyy-MM-DD の後ろに 00:00:00[.000000]を付加したデータをデータベースに格納します。
setTimestamp(Timestamp Obj)※3	HiRDB のエラーを返します。	設定値から yyyy-MM-DD を抜き出したデータをデータベースに格納します。	設定値をデータベースに格納します。
setString(hh:mm:ss 形式の文字列)	設定値をデータベースに格納します。	HiRDB のエラーを返します。	HiRDB のエラーを返します。
setString(yyyy-MM-DD 形式の文字列)	HiRDB のエラーを返します。	設定値をデータベースに格納します。	HiRDB のエラーを返します。
setString(yyyy-MM-DD hh:mm:ss[.ffffff]形式の文字列)※4	HiRDB のエラーを返します。	HiRDB のエラーを返します。	設定値をデータベースに格納します。

注※1　Time Obj：java.sql.Time オブジェクト「時：分：秒」の値を持つオブジェクト。

注※2　Date Obj：java.sql.Date オブジェクト「年－月－日」の値を持つオブジェクト。

注※3　Timestamp Obj：java.sql.Timestamp オブジェクト「年－月－日　時：分：秒．ナノ秒」の値を持つオブジェクト。

注※4　[.ffffff]：HiRDB の Timestamp 型の精度によって，小数点以下のけた数が変わります。

- HiRDB を使用する場合での setXXX メソッド使用時のオーバーフローチェック

  XXX は，?パラメタの適切な型です。HiRDB には，値によってはオーバーフローが発生するデータ型が存在します。HiRDB のデータ型と，メソッドとの組み合わせによるオーバーフローの発生の有無について，次の表に示します。オーバーフロー発生時は例外を返します。

表 4-79　HiRDB のデータ型と setXXX メソッドの組み合わせによるオーバーフローの発生の有無

setXXX メソッド	HiRDB のデータ型											
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR	VARCHAR	DATE	TIME	TIMESTAMP	BINARY	BLOB
setByte	○	○	○	○	×	－	－	－	－	－	－	－
setShort	○	○	○	○	×	－	－	－	－	－	－	－
setInt	×	○	○	○	×	－	－	－	－	－	－	－
setLong	×	×	○	○	×	－	－	－	－	－	－	－
setFloat	×	×	○	○	×	－	－	－	－	－	－	－
setDouble	×	×	○	×	×	－	－	－	－	－	－	－
setBigDecimal	×	×	○	○	×	－	－	－	－	－	－	－
setBoolean	○	○	○	○	×	－	－	－	－	－	－	－
setString	×	×	×	×	×	－	－	×	×	×	－	－
setBytes	－	－	－	－	－	－	－	－	－	－	－	－
setDate	－	－	－	－	－	－	－	○	－	×	－	－
setTime	－	－	－	－	－	－	－	－	○	×	－	－
setTimestamp	－	－	－	－	－	－	－	－	－	○	－	－

（凡例）

○：値にかかわらずオーバーフローしません。

×：値によってはオーバーフローする場合があります。

－：この組み合わせでは使用できません。

表 4-80　HiRDB のデータ型と setObject メソッドの組み合わせによるオーバーフローの発生の有無

setObject メソッド	HiRDB のデータ型											
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR	VARCHAR	DATE	TIME	TIMESTAMP	BINARY	BLOB
Byte	○	○	○	○	×	－	－	－	－	－	－	－
Short	○	○	○	○	×	－	－	－	－	－	－	－

setObject メソッド	HiRDB のデータ型											
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR	VARCHAR	DATE	TIME	TIMESTAMP	BINARY	BLOB
Integer	×	○	○	○	×	−	−	−	−	−	−	−
Long	×	×	○	○	×	−	−	−	−	−	−	−
Decimal	×	×	○	○	×	−	−	−	−	−	−	−
Float	×	×	○	○	×	−	−	−	−	−	−	−
Double	×	×	○	×	×	−	−	−	−	−	−	−
Boolean	○	○	○	○	×	−	−	−	−	−	−	−
String	×	×	×	×	×	−	−	×	×	×	−	−
Date	−	−	−	−	−	−	−	○	−	−	−	−
Time	−	−	−	−	−	−	−	−	○	−	−	−
Timestamp	−	−	−	−	−	−	−	−	−	○	−	−
byte[]	−	−	−	−	−	−	−	−	−	−	−	−
BLOB	−	−	−	−	−	−	−	−	−	−	−	−
CLOB	−	−	−	−	−	−	−	−	−	−	−	−
Array	−	−	−	−	−	−	−	−	−	−	−	−
Ref	−	−	−	−	−	−	−	−	−	−	−	−
Struct	−	−	−	−	−	−	−	−	−	−	−	−

（凡例）

○：値にかかわらずオーバーフローしません。

×：値によってはオーバーフローする場合があります。

−：この組み合わせでは使用できません。

- Commit または Rollback にわたっての SQL の前処理の保持機能

  DABroker Library は DBMS が HiRDB または ORACLE の場合，SQL の前処理を Commit または Rollback にわたって保持します。このため，HiRDB の場合，SELECT，INSERT，DELETE，UPDATE，PURGE および CALL のどれかでアクセスする次のスキーマ資源に対して，ほかのユーザが定義系 SQL 文を発行すると，スキーマ資源にアクセスしていたコネクションを DISCONNECT するまでの間，定義系 SQL は排他待ちの状態になります。

  ・表

  ・ビュー

  ・ストアドプロシジャ

  ・ストアドファンクション

  ・抽象データ型

参考

XADataSource を使用して，バージョンが 07-01 より前の HiRDB に接続した場合，この機能は有効になりません。バージョンが 07-01 以降の HiRDB に接続する場合に，この機能が有効になります。
DABroker Library を使用したデータベース接続の実装については，「4.7　DABroker Library を使用したデータベース接続の実装」を参照してください。

Component Container でコネクションをプーリングしている場合，定義系 SQL を発行する前に次のコマンドを実行してください。Component Container の cjclearpool コマンドを実行することで，コネクションプールからコネクションを破棄できます。

```
cjclearpool 〈サーバ名称〉 -mode normal -resall
```

この場合，<サーバ名称>で指定した J2EE サーバまたはバッチサーバに接続しているすべてのデータソースについてコネクションが削除されます。J2EE アプリケーションまたはバッチアプリケーションでコネクションをクローズしたときに，物理コネクションもクローズされます。

cjclearpool コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclearpool（コネクションプール内のコネクション削除）」を参照してください。

制限事項

DABroker Library で使用する PreparedStatement クラスの制限事項を，次に示します。

表 4-81　PreparedStatement クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
setArray	SQL 配列型をサポートしていないため，無条件に SQLException をスローします。	－	○
setBlob	接続データベースが Oracle9i，または Oracle10g 以外の場合，SQL BLOB 型をサポートしていないため，無条件に SQLException をスローします。	－	○
setClob	接続データベースが Oracle9i，または Oracle10g 以外の場合，SQL CLOB 型をサポートしていないため，無条件に SQLException をスローします。	－	○
setNull	ユーザ定義型をサポートしていないため，無条件に SQLException をスローします。	－	○
setRef	SQL 構造化型をサポートしていないため，無条件に SQLException をスローします。	－	○

（凡例）

○：該当します。

－：該当しません。

DABroker Library 提供メソッド

PreparedStatement クラスで提供している，DABroker Library だけの機能を次に示します。

メソッド一覧

メソッド名	機能
getBlockUpdate メソッド	接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかの設定情報を取得します。
setBlockUpdate メソッド	接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。

getBlockUpdate メソッド

説明

接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかの設定情報を取得します。

形式

```
public boolean getBlockUpdate();
```

パラメタ

なし

例外

なし

戻り値

boolean：

複数のパラメタセットを一度に処理するかどうかを次の値で取得します。

- true
  複数のパラメタセットを一度に処理します。
- false
  複数のパラメタセットを一つずつ分割して処理します。

注意事項

- この機能は，HiRDB の配列を使用した更新機能を使用します。このため，HiRDB の配列を使用した更新機能の使用条件に満たない場合はエラーとなります。なお，配列を使用した更新機能の詳細については，マニュアル「HiRDB UAP 開発ガイド」の配列を使用した機能，およびマニュアル「HiRDB SQL リファレンス」の EXECUTE 文（SQL の実行形式 2），INSERT 文（行挿入）形式 3，DELETE 文（行削除）形式 2，UPDATE 文（行更新）形式 3，形式 4 を参照してください。
- この機能は，HiRDB に対する INSERT，UPDATE，DELETE 処理以外には使用できません。
- 複数のパラメタセット各列のデータ型はすべて同じにしてください。複数のパラメタセット各列のデータ型が異なる場合，更新処理実行時にエラーになります。
- DECIMAL 型データを挿入する場合，配列に指定する DECIMAL データの精度および位置取りはすべて同じにしてください。精度および位置取りが異なる場合はエラーになります。
- 複数のパラメタセットを一度に処理する場合に，バッチ更新の途中でエラーが発生すると，エラーが発生する直前までの更新処理はすべて無効になります。

setBlockUpdate メソッド

説明

接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。

このメソッドが呼び出されない場合は，false を設定します。ただし，データベース接続時に DriverManager クラスの getConnection メソッドの引数に BLOCK_UPDATE=true を指定した場合，このメソッドが呼び出されないときに設定される値は，true になります。

形式

```
public void setBlockUpdate(boolean Mode);
```

パラメタ

Mode：

複数のパラメタセットを一度に処理するかどうかを次の値で設定します。

- true
  複数のパラメタセットを一度に処理します。
- false
  複数のパラメタセットを一つずつ分割して処理します。

例外

なし

戻り値

なし

注意事項

- この指定は，接続データベースが HiRDB の場合だけ有効です。HiRDB 以外のデータベースの場合は，指定の有無にかかわらず，複数のパラメタセットを一つずつ分割して処理します。
- 実際にパラメタセットが一度に処理されるかどうかは，DABroker Library および HiRDB の仕様に従います。
- この機能は，HiRDB の配列を使用した更新機能を使用します。このため，HiRDB の配列を使用した更新機能の使用条件に満たない場合はエラーとなります。なお，配列を使用した更新機能の詳細については，マニュアル「HiRDB UAP 開発ガイド」の配列を使用した機能，およびマニュアル「HiRDB SQL リファレンス」の EXECUTE 文（SQL の実行形式 2），INSERT 文（行挿入）形式 3，DELETE 文（行削除）形式 2，UPDATE 文（行更新）形式 3，形式 4 を参照してください。
- この機能は，HiRDB に対する INSERT，UPDATE，DELETE 処理以外には使用できません。
- 複数のパラメタセット各列のデータ型はすべて同じにしてください。複数のパラメタセット各列のデータ型が異なる場合，更新処理実行時にエラーになります。
- DECIMAL 型データを挿入する場合，配列に指定する DECIMAL データの精度および位置取りはすべて同じにしてください。精度および位置取りが異なる場合はエラーになります。
- 複数のパラメタセットを一度に処理する場合，バッチ更新の途中でエラーが発生すると，エラーが発生する直前までの更新処理はすべて無効になります。

4.15　CallableStatement クラス

説明

CallableStatement クラスでは，主に次の機能を提供します。なお，CallableStatement クラスは PreparedStatement クラスのサブクラスです。PreparedStatement クラスおよび Statement クラスの機能をすべて継承します。

- ストアドプロシジャの実行
- INPUT および INOUT パラメタの設定（PreparedStatement クラスの setXXX メソッドを使用）
- INOUT および OUTPUT パラメタの登録
- INOUT および OUTPUT パラメタ値の取得

CallableStatement クラスの提供する各メソッドの詳細および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

注意事項

- CallableStatement クラスは PreparedStatement クラスのサブクラスであるため，PreparedStatement クラス，および Statement クラスの注意事項はすべて該当します。PreparedStatement クラスおよび Statement クラスの注意事項を参照してください。
- ResultSet を伴うストアドプロシジャ

  DABroker Library では，ResultSet を伴うストアドプロシジャをサポートしていません。このため，getMoreResults メソッドは無条件に false を返却し，getResultSet メソッドは無条件に null を返却します。
- ストアドプロシジャの OUT パラメタ取得

  setXXX メソッド，および registerOutParameter メソッドで設定した情報は，execute メソッド，executeUpdate メソッド，または executeQuery メソッド実行後，getXXX メソッドを使用して必要な OUT パラメタをすべて取得し，それ以降その実行結果が不要になるまでは変更しないでください。OUT パラメタを取り出すときにパラメタの設定情報を参照するため，途中で clearParameter メソッドでパラメタの設定情報をクリアしたり，setXXX メソッド，および registerOutParameter メソッドでパラメタ情報を再設定したりすると，getXXX メソッドでエラーが発生し OUT パラメタの取得ができなくなります。

  （例）OUT パラメタを取得する前に clearParameter メソッドを実行した場合

  （実行する TEST_PROC(?)の?パラメタは REAL 型の OUT パラメタとする）

```
cstmt = con.prepareCall("{CALL TEST_PROC(?)}");
cstmt.registerOutParameter(1, Types.REAL);  // OUTパラメタ情報の設定
cstmt.execute();   // プロシジャ実行
cstmt.clearParameter();   // パラメタ情報のクリア
float float_value = cstmt.getFloat(1);   // OUTパラメタの取得，エラーの発生
```

  発生するエラー：KFDJ05006-E

  Output attribute is not able to acuire information because it does not exist in a parameter.
- DECIMAL 型使用時の注意事項

  DECIMAL 型の OUT パラメタ，または INOUT パラメタを設定するとき，小数点以下のけた数（以下スケール値）を受け入れない形式の registerOutParameter(int parameterIndex，int sqlType)メソッドを使用するとスケール値は 0 とみなされます。

  また，INOUT パラメタへの設定は最後に実行したメソッドのスケール値が有効となります。そのため DECIMAL 型の INOUT パラメタを設定する場合は，スケール値を受け入れない形式の registerOutParameter(int parameterIndex，int sqlType)メソッドを先に設定したあとに

setBigDecimal(int parameterIndex，BigDecimal x)メソッドを設定，または setBigDecimal(int parameterIndex， BigDecimal x)メソッドを設定したあとにスケール値を受け入れる形式の registerOutParameter(int parameterIndex，int sqlType，int scale)メソッドを設定してください。このとき，setBigDecimal メソッドに指定する java.math.BigDecimal オブジェクトのスケール値，および registerOutParameter メソッドに指定するスケール値はデータベースに定義しているスケール値と同じにする必要があります。

（例 1）registerOutParameter メソッドを先に設定する場合(定義長(15,5)とする)

```
cstmt.registerOutParameter(1, Types.DECIMAL);
cstmt.setBigDecimal(1, new java.math.BigDecimal("123.45000"));
```

（例 2）scale 付き registerOutParameter メソッドを使用する場合(定義長(15,5)とする)

```
cstmt.setBigDecimal(1, new java.math.BigDecimal("123.45000"));
cstmt.registerOutParameter(1,Types.DECIMAL,5);
```

- データベースから 0 バイトデータを取得する場合の注意事項

  データベースから 0 バイトデータを取得すると，null が返されることがあります。発生条件を次に示します。

HiRDB の場合

次の条件が重なった場合に発生します。

- データベースに格納されている 0 バイトデータを，ストアドプロシジャの OUT パラメタ，または INOUT パラメタで取得する場合。
- 列のデータ型が次のどれかの場合。

  VARCHAR
  NVARCHAR
  MVARCHAR
  BINARY
  BLOB

Oracle の場合

次の条件が重なった場合に発生します。

- データベースに格納されている 0 バイトデータを，ResultSet クラス，ストアドプロシジャの OUT パラメタ，または INOUT パラメタで取得する場合。
- 列のデータ型が 0 バイトデータを格納できるデータ型の場合。

制限事項

DABroker Library で使用する CallableStatement クラスの制限事項を，次に示します。

表 4-82　CallableStatement クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
getArray	SQL 配列型をサポートしていないため，無条件に SQLException をスローします。	−	○
getBlob	接続データベースが Oracle9i，または Oracle10g 以外の場合，SQL BLOB 型をサポートしていないため，無条件に SQLException をスローします。	−	○

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
getClob	接続データベースが Oracle9i，または Oracle10g 以外の場合，SQL CLOB 型をサポートしていないため，無条件に SQLException をスローします。	－	○
getRef	SQL 構造化型をサポートしていないため，無条件に SQLException をスローします。	－	○

（凡例）

○：該当します。

－：該当しません。

4.16　ResultSet クラス

説明

ResultSet クラスでは，主に次の機能を提供します。

- 行単位の ResultSet 内の移動
- 結果データの返却
- 検索結果データが NULL 値かどうかの通知

ResultSet クラスの提供する各メソッドの詳細および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

注意事項

- マルチスレッド

  一つの ResultSet オブジェクトを複数のスレッドで並行して使用する場合の動作は保証しません。

  一つの ResultSet オブジェクトは一つのスレッドで処理してください。
- setFetchSize メソッドの使用について

  Statement クラスの注意事項を参照してください。
- HiRDB のホールダブルカーソル機能の使用について

  DABroker Library では，HiRDB のホールダブルカーソル機能を使用することによって，複数のコミットにわたってカーソルを保持できます。使用するためには，接続時にプロパティまたは URL に HIRDB_CURSOR=true を設定するか，DataSource クラスの setHiRDBCursorMode(true)を設定する必要があります。
- ResultSet のクローズ

  プーリング使用時（ConnectionPoolDataSource を使用した接続），および XA 使用時（XADataSource を使用した接続）で ResultSet の close メソッド実行中にデータベースとの物理的な切断でエラーが発生しコネクションプーリングが使用できなくなった場合，ConnectionEventListener.connectionErrorOccurred は発生しません。

制限事項

DABroker Library で使用する ResultSet クラスの制限事項を，次に示します。

表 4–83　ResultSet クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
close	ステートメントのクローズ時，または次の SQL 実行時に暗黙的にクローズします。プーリング使用時（ConnectionPoolDataSource を使用した接続），および XA 使用時（XADataSource を使用した接続）で ResultSet.close メソッド実行中にデータベースとの物理的な切断でエラーが発生しコネクションプーリングが使用できなくなった場合 ConnectionEventListener.connectionErrorOccurred は発生しません。	○	○
cancelRowUpdates	更新可能型 ResultSet をサポートしていないため，無条件に SQLException をスローします。	−	○
deleteRow			

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
findColumn	指定された列名が ResultSet オブジェクトに含まれていないため列インデックスが取得できない場合，SQLException をスローします。	○	○
absolute	absolute(-2)と指定した場合，カーソルは最終行の前の行へ移動します。	－	○
afterLast	結果セットに行がない場合，何も処理しません。	－	○
beforeFirst			
cancelRowUpdates	更新可能型 ResultSet をサポートしていないため，無条件に SQLException をスローします。	－	○
deleteRow			
getArray	SQL 配列型をサポートしていないため，無条件に SQLException をスローします。	－	○
getBlob	接続データベースが HiRDB，Oracle9i，または Oracle10g 以外の場合，SQL LONGVARBINARY 型および SQL BLOB 型をサポートしていないため，無条件に SQLException をスローします。	－	○
getClob	接続データベースが Oracle9i，または Oracle10g 以外の場合，SQL CLOB 型をサポートしていないため，無条件に SQLException をスローします。	－	○
getFetchSize	0，または setFetchSize メソッドで指定された値が setMaxRows メソッドで指定された値より小さい場合は setFetchSize メソッドで指定された値を，setFetchSize メソッドで指定された値が setMaxRows メソッドで指定された値より大きい場合は setMaxRows メソッドで指定された値を返却します。	－	○
getRef	SQL 構造化型をサポートしていないため，無条件に SQLException をスローします。	－	○
isFirst	結果セットに行がない場合 false を返却します。	－	○
isLast	結果セットに行がない場合 false を返却します。	－	○
insertRow	更新可能型 ResultSet をサポートしていないため，無条件に SQLException をスローします。	－	○
moveToCurrentRow			
moveToInsertRow			
previous	カーソルの移動方向が next メソッドと逆方向になります。	－	○
refreshRow	更新可能型 ResultSet をサポートしていないため，無条件に SQLException をスローします。	－	○
relative	結果セットに行がない場合 SQLException をスローします。カーソルが先頭行の前，または最終行の後ろにある場合は有効として処理します。処理結果が結果セットの先頭行，最終行を超える場合はそれぞれ先頭行の前，最終行の後ろへ移動します。	－	○

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
rowDeleted	更新可能型 ResultSet をサポートしていないため，無条件に SQLException をスローします。	−	○
rowInserted			
rowUpdated			
setFetchDirection	FETCH_FORWARD 以外を指定した場合，SQLException をスローします。	−	○
setFetchSize	接続データベース種別が HiRDB，または ORACLE の場合で，CLOB や BLOB などの LONG VARCHAR・LONG VARBINARY 型データ列の検索を含まないときだけ有効となります。	−	○
updateAsciiStream	更新可能型 ResultSet をサポートしていないため，無条件に SQLException をスローします。	−	○
updateBigDecimal			
updateBinaryStream			
updateBoolean			
updateByte			
updateBytes			
updateCharacterStream			
updateDate			
updateDouble			
updateFloat			
updateInt			
updateLong			
updateNull			
updateObject			
updateRow			
updateShort			
updateString			
updateTime			
updateTimestamp			

（凡例）

○：該当します。

−：該当しません。

4.17　ResultSetMetaData クラス

説明

ResultSetMetaData クラスでは，主に次の機能を提供します。

- ResultSet の各列に対する，データ型およびデータ長のメタ情報の返却

ResultSetMetaData クラスの提供する各メソッドの詳細および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

制限事項

DABroker Library で使用する ResultSetMetaData クラスの制限事項を，次に示します。

表 4-84　ResultSetMetaData クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
getCatalogName	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に null を返却します。	○	○
getColumnClassName	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に false を返却します。	－	○
getColumnDisplaySize	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。	○	○
getColumnLabel	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，カラムのラベル（カラムヘッダ）をサポートしていないため，カラム名を返却します。	○	○
getColumnName	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。データベースから返される列名称を返却します。	○	○
getColumnType	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。	○	○
getPrecision	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。列属性が DECIMAL，NUMERIC の場合は精度を返却し，それ以外は列長を返却します。	○	○
getScale	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。列属性が DECIMAL，NUMERIC の場合は小数点以下のけた数を返却し，それ以外は 0 を返却します。	○	○
getSchemaName getTableName	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に null を返却します。	○	○
isAutoIncrement isCaseSensitive	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に false を返却します。	○	○

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
isCurrency isDefinitelyWritable	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に false を返却します。	○	○
isNullable	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。	○	○
isReadOnly	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に false を返却します。	○	○
isSearchable	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。	○	○
isSigned	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。列属性が数値属性の場合 true を返却し，それ以外の場合は false を返却します。	○	○
isWritable	column が 0 以下，またはカラム数を超えている場合は SQLException をスローします。それ以外の場合，無条件に false を返却します。	○	○

（凡例）

○：該当します。

－：該当しません。

4.18　DatabaseMetaData クラス

説明

DatabaseMetaData クラスでは，主に次の機能を提供します。

- 接続データベースに関する各種情報の返却
- 表一覧，列一覧などの一覧系情報を，ResultSet に格納して返却

DatabaseMetaData クラスの提供する各メソッドの詳細および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

制限事項

DABroker Library で使用する DatabaseMetaData クラスの制限事項を，次に示します。

表 4-85　DatabaseMetaData クラスの制限事項

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
dataDefinitionIgnoredInTransactions	無条件に false を返却します。	○	○
deletesAreDetected	更新結果を反映する ResultSet をサポートしていないため，無条件に false を返却します。	−	○
getBestRowIdentifier	行を一意に識別するテーブルの最適なカラムに関する記述を返却します（返す結果は常に 0 件）。	○	○
getCatalogs	カタログ名称に関する記述を返却します（返す結果は常に 0 件）。	○	○
getCrossReference	主キーテーブルの主キーカラムを参照する外部キーテーブル中の，外部キーカラムに関する記述を返却します（返す結果は常に 0 件）。	○	○
getDatabaseProductVersion	無条件に null を返却します。	○	○
getDefaultTransactionIsolation	無条件に TRANSACTION_READ_UNCOMMITTED を返却します。	○	○
getExportedKeys	主キーのカラムを参照する，外部キーのカラムに関する記述を返却します（返す結果は常に 0 件）。	○	○
getIdentifierQuoteString	無条件に引用符(")を返却します。	○	○
getImportedKeys	外部キーのカラムを参照する，主キーのカラムに関する記述を返却します（返す結果は常に 0 件）。	○	○
getMaxConnections	無条件に 0 を返却します。	○	○
getMaxStatements	接続データベースが HiRDB または Oracle の場合，無条件に 1024 を，それ以外の場合は，64 を返却します。	○	○

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
getUDTs	ユーザ定義型に関する記述を返却します（返す結果は常に 0 件）。	−	○
getVersionColumns	自動的に更新されるカラムに関する記述を返却します（返す結果は常に 0 件）。	○	○
isReadOnly	アクセスモードを変更できないため，無条件に false を返却します。	○	○
insertsAreDetected	更新結果を反映する ResultSet をサポートしていないため，無条件に false を返却します。	−	○
othersDeletesAreVisible			
othersInsertsAreVisible			
othersUpdatesAreVisible			
ownDeletesAreVisible			
ownInsertsAreVisible			
ownUpdatesAreVisible			
storesLowerCaseQuotedIdentifiers	無条件に false を返却します。	○	○
supportsANSI92EntryLevelSQL	無条件に true を返却します。	○	○
supportsANSI92FullSQL	無条件に false を返却します。	○	○
supportsANSI92IntermediateSQL			
supportsCatalogsInIndexDefinitions			
supportsCatalogsInPrivilegeDefinitions			
supportsMixedCaseIdentifiers			
supportsMultipleResultSets			
supportsMultipleTransactions	無条件に true を返却します。	○	○
supportsPositionedDelete	無条件に false を返却します。	○	○
supportsPositionedUpdate			
supportsSelectForUpdate			
supportsSchemasInDataManipulation	無条件に true を返却します。	○	○
supportsTransactionIsolationLevel	与えられたトランザクションアイソレーションレベルが TRANSACTION_READ_UNCOMMITTED の場合，true を返却します。	○	○
supportsTransactions	無条件に true を返却します。	○	○
usesLocalFilePerTable	無条件に false を返却します。	○	○
usesLocalFiles			

メソッド名	制限事項	JDBC1.0 での制限	JDBC2.0 での制限
supportsOpenCursorsAcrossCommit	HiRDB 接続時 表 4-86 を参照してください。 Oracle 接続時 常に true を返却します。 XDM/RD E2 接続時 XDM/RD E2 11-01 以前の場合，常に false を返却します。 XDM/RD E2 11-02 以降の場合，HiRDB 接続時と同じです。詳細については，表 4-86 を参照してください。	－	○
supportsOpenCursorsAcrossRollback			
supportsOpenStatementsAcrossCommit			
supportsOpenStatementsAcrossRollback			
supportsBatchUpdates	無条件に true を返却します。	－	○
supportsResultSetConcurrency	ResultSet タイプが TYPE_FORWARD_ONLY または TYPE_SCROLL_INSENSITIVE で，並行処理タイプが CONCUR_READ_ONLY の場合，true を返却します。	－	○
updatesAreDetected	更新結果を反映する ResultSet をサポートしていないため，無条件に false を返却します。	－	○

（凡例）

○：該当します。

－：該当しません。

表 4-86　HiRDB 接続時の supportsOpenXXXX メソッドの戻り値

接続時の HIRDB_CURSOR の指定値	
FALSE	TRUE
supportsOpenCursorsAcrossCommit=false supportsOpenCursorsAcrossRollback=false supportsOpenStatementsAcrossCommit=true supportsOpenStatementsAcrossRollback=true	supportsOpenCursorsAcrossCommit=true supportsOpenCursorsAcrossRollback=false supportsOpenStatementsAcrossCommit=true supportsOpenStatementsAcrossRollback=true

4.19　DataSource クラス

説明

DataSource クラスでは，次の機能を提供します。

- データベースの接続情報の設定および取得

このクラスのパッケージ名は，JP.co.Hitachi.soft.DBPSV_Driver です。

JdbcDbpsvDataSource クラス

JDBC2.0 拡張機能で提供する，JNDI 連携機能を使用したデータベースへアクセスするためのインタフェースです。

表 4-87　JdbcDbpsvDataSource クラスのメソッド一覧

メソッド名	機能
getBlockUpdate メソッド	接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかの設定を取得します。
getBufferPoolSize メソッド	受信バッファプール数を取得します。
getBufSize メソッド	DABroker Library からの受信データのバッファ長を取得します。
getDatabaseName メソッド	接続するデータベースの種別を取得します。
getDBEnv メソッド	DABroker Library の接続先データベース定義情報を取得します。
getDBHostName メソッド	接続する HiRDB のホスト名を取得します。
getDescription メソッド	接続するデータベースに必要な接続付加情報を取得します。
getEncodLang メソッド	エンコード文字形態を取得します。
getExecuteDirectMode メソッド	接続データベースが HiRDB の場合，Statement クラスを使用したデータベースの更新で HiRDB の Execute Direct 機能を使用するかどうかの設定を取得します。
getHiRDBCursorMode メソッド	接続データベースが HiRDB の場合，検索時にカーソルが複数の Commit または Rollback にわたって有効かどうかの設定を取得します。
getJDBC_IF_TRC メソッド	JDBC インタフェースメソッドトレースの取得の有無を取得します。
getLONGVARBINARY_Access メソッド	HiRDB で LONGVARBINARY（列属性 BLOB，列属性 BINARY）へのアクセス方法を取得します。
getNetworkProtocol メソッド	DABroker Library との接続種別を取得します。
getNotErrorOccurred メソッド	connectionErrorOccurred が呼ばれるかどうかの設定を取得します。
getOSAuthorize メソッド	OS 認証機能を使用してデータベースに接続するかどうかの設定を取得します。
getPassword メソッド	データベースアクセスのパスワードを取得します。
getPortNumber メソッド	接続する DABroker Library のポート番号を取得します。
getRowSize メソッド	JDBC で取り扱うバッファ長を取得します。
getServerName メソッド	接続する DABroker Library のホスト名を取得します。

メソッド名	機能
getSQLWarningIgnore メソッド	データベースから返される警告を Connection クラスで保持するかどうかの情報を取得します。
getSV_EVENT_TRC メソッド	DABroker Library とのイベントトレースの取得の有無を取得します。
getTRC_NO メソッド	トレースのエントリ数を取得します。
getUapName メソッド	アプリケーション名称を取得します。
getUser メソッド	データベースアクセスのユーザ ID を取得します。
setBlockUpdate メソッド	接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。
setBufferPoolSize メソッド	受信バッファプール数を設定します。
setBufSize メソッド	DABroker Library からの受信データのバッファ長を設定します。
setDatabaseName メソッド	接続するデータベースの種別を設定します。
setDBEnv メソッド	DABroker Library の接続先データベース定義情報を設定します。
setDBHostName メソッド	接続する HiRDB のホスト名を設定します。
setDescription メソッド	接続するデータベースに必要な接続付加情報を設定します。
setEncodLang メソッド	エンコード文字形態を設定します。
setExecuteDirectMode メソッド	接続データベースが HiRDB の場合，Statement クラスを使用したデータベースの更新で HiRDB の Execute Direct 機能を使用するかどうかを設定します。
setHiRDBCursorMode メソッド	接続データベースが HiRDB の場合，検索時にカーソルが複数の Commit または Rollback にわたって有効かどうかを設定します。
setJDBC_IF_TRC メソッド	JDBC インタフェースメソッドトレースの取得の有無を設定します。
setLONGVARBINARY_Access メソッド	HiRDB で LONGVARBINARY（列属性 BLOB，列属性 BINARY）へのアクセス方法を設定します。
setNetworkProtocol メソッド	DABroker Library との接続種別を設定します。
setNotErrorOccurred メソッド	connectionErrorOccurred が呼ばれるかどうかを設定します。
setOSAuthorize メソッド	OS 認証機能を使用してデータベースに接続するかどうかを設定します。
setPassword メソッド	データベースアクセスのパスワードを設定します。
setPortNumber メソッド	接続する DABroker Library のポート番号を設定します。
setRowSize メソッド	JDBC で取り扱うバッファ長を指定します。
setServerName メソッド	接続する DABroker Library のホスト名を設定します。
setSQLWarningIgnore メソッド	データベースから返される警告を Connection クラスで保持するかどうかの情報を設定します。
setSV_EVENT_TRC メソッド	DABroker Library とのイベントトレースの取得の有無を設定します。
setTRC_NO メソッド	トレースのエントリ数を設定します。

メソッド名	機能
setUapName メソッド	アプリケーション名称を設定します。
setUser メソッド	データベースアクセスのユーザ ID を設定します。

JdbcDbpsvXADataSource クラス

JDBC2.0 拡張機能で提供する，トランザクション連携のためのインタフェースです。

表 4–88　JdbcDbpsvXADataSource クラスのメソッド一覧

メソッド名	機能
getRMID メソッド	リソースマネジャの識別子を取得します。
getXACloseString メソッド	XA_CLOSE 文字列を取得します。
getXALocalCommitMode メソッド	XA 使用時，データベースのオートコミットを有効にしているかどうかの設定を取得します。
getXAOpenString メソッド	XA_OPEN 文字列を取得します。
getXAThreadMode メソッド	XA 使用時のスレッドモードを取得します。
setRMID メソッド	リソースマネジャの識別子を設定します。
setXACloseString メソッド	XA_CLOSE 文字列を設定します。
setXALocalCommitMode メソッド	XA 使用時，データベースのオートコミットを有効にするかどうかを設定します。
setXAOpenString メソッド	XA_OPEN 文字列を設定します。
setXAThreadMode メソッド	XA 使用時のスレッドモードを設定します。

getBlockUpdate メソッド

説明

接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかの設定情報を取得します。

形式

```
public boolean getBlockUpdate();
```

パラメタ

なし

例外

なし

戻り値

boolean：

複数のパラメタセットを一度に処理するかどうかを次の値で取得します。

- true
  複数のパラメタセットを一度に処理します。
- false
  複数のパラメタセットを一つずつ分割して処理します。

getBufferPoolSize メソッド

説明

setBufferPoolSize メソッドで設定した，受信バッファプール数を取得します。setBufferPoolSize メソッドで受信バッファプール数が設定されていない場合は，0 を取得します。

このメソッドは，受信バッファプール数が設定されている場合に使用します。受信バッファプール数の設定については，「4.5　受信バッファのプーリングの設定」を参照してください。

形式

```
public int getBufferPoolSize();
```

パラメタ

なし

例外

なし

戻り値

int：
　受信バッファプール数を取得します。

getBufSize メソッド

説明

setBufSize メソッドで設定した，DABroker Library からの受信データを格納するためのバッファ長を取得します。

setBufSize メソッドで受信バッファ長が設定されていない場合は，64 を取得します。

形式

```
public int getBufSize();
```

パラメタ

なし

例外

なし

戻り値

int：
　バッファ長を取得します。

getDatabaseName メソッド

説明

setDatabaseName メソッドで設定された，接続するデータベースの種別を取得します。

接続データベース種別が設定されていない場合は，null を取得します。

形式

```
public String getDatabaseName();
```

パラメタ

なし

例外

なし

戻り値

String：
　接続するデータベースの種別を取得します。

getDBEnv メソッド

説明

setDBEnv メソッドで設定された，接続するデータベースに対する接続先データベース定義情報を取得します。

接続先データベース定義情報が設定されていない場合は，null を取得します。

形式

```
public String getDBEnv();
```

パラメタ

なし

例外

なし

戻り値

String：
　DABroker Library に設定してある，接続先データベース定義情報を取得します。

getDBHostName メソッド

説明

setDBHostName メソッドで設定された，接続する HiRDB のホスト名を取得します。

HiRDB のホスト名が設定されていない場合は，null を取得します。

形式

```
public String getDBHostName();
```

パラメタ

なし

例外

なし

戻り値

String：
　HiRDB のホスト名を取得します。

getDescription メソッド

説明

setDescription メソッドで設定された，接続するデータベースに必要な接続付加情報を取得します。

接続付加情報が設定されていない場合は，null を取得します。

形式

```
public String getDescription();
```

パラメタ

なし

例外

なし

戻り値

String：
　接続付加情報を取得します。

getEncodLang メソッド

説明

エンコード文字形態を取得します。

形式

```
public String getEncodLang();
```

パラメタ

なし

例外

なし

戻り値

String：
　setEncodLang メソッドで設定した，文字エンコーディング情報を取得します。

getExecuteDirectMode メソッド

説明

接続データベースが HiRDB の場合，INSERT，UPDATE，DELETE など，Statement クラスを使用したデータベースの更新で，HiRDB の Execute Direct 機能を使用するかどうかの設定情報を取得します。

形式

```
public boolean getExecuteDirectMode();
```

パラメタ

なし

例外

なし

戻り値

boolean：
　Execute Direct 機能を使用するかどうかを次の値で取得します。

- true
  Execute Direct 機能を使用します。
- false
  Execute Direct 機能を使用しません。

getHiRDBCursorMode メソッド

説明

HiRDB で検索時にカーソルが複数の Commit にわたって有効かどうかの設定情報を取得します。

MetaData を取得する場合，true を指定して LOCK TABLE UNTIL DISCONNECT をしないで，複数の Commit にわたって Fetch をするとエラーが発生します。MetaData を取得する場合は，true を設定しないでください。

MetaData を取得する場合，false またはデフォルトの状態で ResultSet の Fetch 実行中に Commit を実行するとエラーが発生します。

この機能を使用する場合，マニュアル「HiRDB SQL リファレンス」の DECLARE CURSOR（カーソル宣言）形式 1 を参照してください。

形式

```
public boolean getHiRDBCursorMode();
```

パラメタ

なし

例外

なし

戻り値

boolean：

HiRDB で検索時にカーソルが複数の Commit にわたって有効かどうかの設定情報を次の値で取得します。

- true
  カーソルは保持されます。アプリケーションは続けて Fetch できます（LOCK TABLE UNTIL DISCONNECT が前提です）。
- false
  カーソルはクローズされますが，ステートメントは有効です。アプリケーションは，Prepare しないで，再度 Execute できます。

getJDBC_IF_TRC メソッド

説明

setJDBC_IF_TRC メソッドで設定した，JDBC インタフェースメソッドトレースの取得の有無を取得します。

形式

```
public boolean getJDBC_IF_TRC();
```

パラメタ

なし

例外

なし

戻り値

boolean：

トレースの取得の有無を次の値で取得します。

- true
  取得します。
- false
  取得しません。

getLONGVARBINARY_Access メソッド

説明

setLONGVARBINARY_Access メソッドで指定された，LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースアクセス方法の設定情報を返却します。

形式

```
public String getLONGVARBINARY_Access();
```

パラメタ

なし

例外

なし

戻り値

String：

LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースアクセス方法の設定情報を取得します。何も設定されていない場合，"REAL"を返却します。

- "REAL"
  実データでアクセスします。
- "LOCATOR"
  HiRDB の位置付け子（locator）機能を使用してアクセスします。

getNetworkProtocol メソッド

説明

setNetworkProtocol メソッドで設定された，DABroker Library との接続種別を取得します。

接続種別が設定されていない場合は，null を取得します。

形式

```
public String getNetworkProtocol();
```

パラメタ

なし

例外

なし

戻り値

String：
　接続種別を取得します。

getNotErrorOccurred メソッド

説明

ConnectionEventListener.connectionErrorOccurred の発生を抑止するかどうかの設定情報を取得します。

形式

```
public boolean getNotErrorOccurred();
```

パラメタ

なし

例外

なし

戻り値

boolean：
　ConnectionEventListener.connectionErrorOccurred を発生させるかどうかの設定情報を次の値で取得します。

- true
  connectionErrorOccurred を発生させません。
- false
  connectionErrorOccurred を発生させます。

getOSAuthorize メソッド

説明

OS 認証機能を使用してデータベースに接続するかどうかの設定情報を取得します。

このメソッドは，setOSAuthorize(boolean Mode)メソッドで設定した内容を取得します。

setOSAuthorize(boolean Mode)メソッドが呼び出されなかった場合は，false が設定されます。

形式

```
public boolean getOSAuthorize();
```

パラメタ

なし

例外

なし

戻り値

boolean：

OS 認証機能を使用するかどうかの設定情報を次の値で取得します。

- true
  OS 認証機能を使用します。
- false
  OS 認証機能を使用しません。

getPassword メソッド

説明

setPassword メソッドで設定した，データベースアクセスのパスワードを取得します。

setPassword メソッドでパスワードが設定されていない場合は，null を取得します。setPassword メソッドでパスワードを設定し，ユーザ ID とパスワードを引数に持つ getConnection メソッドを呼び出した場合は，getConnection メソッドで指定したパスワードを取得します。

形式

```
public String getPassword();
```

パラメタ

なし

例外

なし

戻り値

String：

パスワードを取得します。

getPortNumber メソッド

説明

setPortNumber メソッドで設定された，DABroker Library のポート番号を取得します。

ポート番号が設定されていない場合は，「40179」を取得します。

形式

```
public int getPortNumber();
```

パラメタ

なし

例外

なし

戻り値

int：
　DABroker Library のポート番号を取得します。

getRowSize メソッド

説明

JDBC で取り扱うバッファ長を取得します。

形式

```
public int getRowSize();
```

パラメタ

なし

例外

なし

戻り値

int：
　バッファ長を取得します。指定を省略した場合は，16 を取得します。

getServerName メソッド

説明

setServerName メソッドで設定された，DABroker Library のホスト名を取得します。

DABroker Library のホスト名が設定されていない場合は，null を取得します。

形式

```
public String getServerName();
```

パラメタ

なし

例外

なし

戻り値

String：
　DABroker Library のホスト名または IP アドレスを取得します。

getSQLWarningIgnore メソッド

説明

データベースから返される警告を Connection クラスで保持するかどうかの設定情報を取得します。

形式

```
public boolean getSQLWarningIgnore();
```

パラメタ

なし

例外

なし

戻り値

boolean：
　警告を保持しないかどうかの設定情報を次の値で取得します。

- true
  警告を保持しません。
- false
  警告を保持します。

getSV_EVENT_TRC メソッド

説明

setSV_EVENT_TRC メソッドで設定した，DABroker Library とのイベントトレースの取得の有無を取得します。

形式

```
public boolean getSV_EVENT_TRC();
```

パラメタ

なし

例外

なし

戻り値

boolean：

トレースの取得の有無を次の値で取得します。

- true
  取得します。
- false
  取得しません。

getTRC_NO メソッド

説明

setTRC_NO メソッドで設定した，トレースのエントリ数を取得します。

setTRC_NO メソッドでエントリ数が指定されていない場合は，100 を取得します。

形式

```
public int getTRC_NO();
```

パラメタ

なし

例外

なし

戻り値

int：

トレースのエントリ数を取得します。

getUapName メソッド

説明

setUapName メソッドで設定した，アプリケーション名称を取得します。

setUapName メソッドでアプリケーション名称が設定されていない場合は，JDBC ドライバの製品名称を取得します。

形式

```
public String getUapName();
```

パラメタ

なし

例外

なし

戻り値

String：
　アプリケーション名称を取得します。

getUser メソッド

説明

setUser メソッドで設定した，データベースアクセスのユーザ ID を取得します。

setUser メソッドでユーザ ID が設定されていない場合は，null を取得します。

setUser メソッドでユーザ ID を設定し，ユーザ ID とパスワードを引数に持つ getConnection メソッドを呼び出した場合は，getConnection メソッドで指定したユーザ ID を取得します。

形式

```
public String getUser();
```

パラメタ

なし

例外

なし

戻り値

String：
　ユーザ ID を取得します。

setBlockUpdate メソッド

説明

接続データベースが HiRDB の場合，?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。

このメソッドが呼び出されない場合は，false を設定します。

この指定は，接続データベースが HiRDB の場合だけ有効です。HiRDB 以外のデータベースの場合は，指定の有無にかかわらず，複数のパラメタセットを一つずつ分割して処理します。DABroker Library を使用してデータベースに接続するための各種情報の設定については，「4.3　DABroker Library の初期設定」を参照してください。

形式

```
public void setBlockUpdate(boolean Mode);
```

パラメタ

Mode：

複数のパラメタセットを一度に処理するかどうかを次の値で設定します。

- true
  複数のパラメタセットを一度に処理します。
- false
  複数のパラメタセットを一つずつ分割して処理します。

例外

なし

戻り値

なし

setBufferPoolSize メソッド

説明

受信バッファプール数を設定します。

0 を設定した場合，受信バッファプール数は設定されません。この場合，すべての受信バッファをプールします。0 未満の値を設定した場合，またはこのメソッドで受信バッファプール数を明示的に設定していない場合は，0 を仮定して，すべての受信バッファをプールします。

1025 以上の値を指定した場合は，1024 を仮定します。

値を仮定した場合は，Exception トレースログにメッセージ（KFDJ30010-W）を出力します。ただし，SQLWarning は作成されません。メッセージの詳細については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「KFDJ30010-W」を参照してください。受信バッファプール数の設定については，「4.5　受信バッファのプーリングの設定」を参照してください。

形式

```
public void setBufferPoolSize(int size);
```

パラメタ

size：

受信バッファプール数を設定するかどうかを指定します。また，受信バッファプール数を設定する場合，受信バッファプールの数を指定します。

次の値で指定します。

- 0
  受信バッファプール数を指定しません。この場合，すべての受信バッファがプールされます。これは，明示的に受信バッファプール数を指定しない場合と同じ動作です。
- 1～1024
  受信バッファプール数を指定する場合に，プールする受信バッファの数を 1～1024 の範囲で指定します。

例外

なし

戻り値

なし

setBufSize メソッド

説明

DABroker Library からの受信データを格納するためのバッファ長を設定します。

このメソッドが呼び出されない場合は，受信バッファ長として 64 を仮定します。

1 より小さい値を設定した場合は 64 を仮定します。

16000 より大きい値を指定した場合は 16000 を仮定します。

値を仮定した場合は，Exception トレースログにメッセージ（KFDJ30009-W）を出力します。メッセージは，DataSource.getConnection()メソッド，PooledConnection.getConnection()メソッド，または XAConnection.getConnection()メソッドによって，Connection オブジェクトを取得するときに出力します。メッセージの詳細については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「KFDJ30009-W」を参照してください。

BLOB データなどの長大データを使用する場合，想定されるデータ以上の値を指定してください。

このメソッドの指定値の上限が 16000 キロバイトのため，BLOB データなど，16000 キロバイトを超える長大データにはアクセスできません。データベース種別が HiRDB の場合は，指定したバイト数を取り出す STRING のインタフェース（BLOB データの部分抽出機能）があるため，ここで指定したサイズ以下のサイズに分割してから取り出してください。BLOB データの部分抽出機能の詳細については，マニュアル「HiRDB UAP 開発ガイド」，およびマニュアル「HiRDB SQL リファレンス」を参照してください。

形式

```
public void setBufSize(int buf_size);
```

パラメタ

buf_size：
　バッファ長を 1～16000 の範囲（単位：キロバイト）で指定します。

例外

なし

戻り値

なし

setDatabaseName メソッド

説明

接続するデータベースの種別を設定します。

このメソッドは，データベースとの接続時に必ず呼び出します。

形式

```
public void setDatabaseName(String database_name);
```

パラメタ

database_name：

接続するデータベースの種別を指定します。

接続データベース種別	設定する内容※1
HiRDB	"HIRDB"
ORACLE	"ORACLE"
Oracle8i※2	"ORACLE8I"

注※1

設定する内容には，大文字，小文字および大文字・小文字の混在する内容を指定できます。

注※2

接続データベース種別に Oracle8i を指定して，DABroker Library 環境設定の「使用する ORACLE のバージョン」に ORACLE9i を指定した場合，Oracle9i に，ORACLE10g を指定した場合，Oracle10g に接続できます。

例外

引数の内容が null の場合，または接続データベース種別に上記の内容以外が指定された場合は，SQLException をスローします。

戻り値

なし

setDBEnv メソッド

説明

DABroker Library の接続先データベース定義情報を設定します。

接続するデータベースに対応する，DABroker Library の接続先データベース定義情報を次の形式で指定します。

データベース種別名：データベース名

このメソッドは，接続するデータベースが DABroker Library-Database Connection Server 経由の場合，必ず呼び出します。接続するデータベースが DABroker Library-Database Connection Server 経由以外の場合は，設定は無視されます。

形式

```
public void setDBEnv(String db_env);
```

パラメタ

db_env：
　接続先データベース定義情報を指定します。

例外

なし

戻り値

なし

setDBHostName メソッド

説明

接続する HiRDB のホスト名を設定します。

このメソッドで指定した値は，接続データベース種別が HiRDB のときだけ有効です。ただし，接続付加情報に HiRDB クライアントの環境変数グループ名を指定している場合は，値をこのメソッドで指定しても有効となりません。

形式

```
public void setDBHostName(String db_host_name);
```

パラメタ

db_host_name：
　HiRDB のホスト名を指定します。

例外

なし

戻り値

なし

setDescription メソッド

説明

接続するデータベースに必要な接続付加情報を設定します。

接続データベース種別	設定する内容	設定の要否
HiRDB	HiRDB システムのポート番号，または HiRDB クライアントの環境変数グループ名※1，※3	任意
ORACLE	SQL*Net の接続文字列※1，※2	任意
Oracle8i		

注※1

省略時には，DABroker Library の動作環境定義の指定値が有効となります。

注※2

XA 使用時には，XA_OPEN 文字列で指定したデータベース名を指定します。

注※3

HiRDB クライアントの環境変数グループ名を指定する場合は，グループ名の先頭に@を付加します。

（例）

- Windows の場合

  HiRDB クライアントの環境変数グループ名が HiRDB_ENV_GROUP のときは，次のように指定します。

```
@DABENVGRP=HiRDB_ENV_GROUP
```

- UNIX の場合

  HiRDB クライアントの環境変数グループ名のパスが/HiRDB_P/Client/HiRDB.ini のときは，次のように指定します。

```
@DABENVGRP=/HiRDB_P/Client/HiRDB.ini
```

@DABENVGRP=＜環境変数グループ名＞を指定する場合は，指定内容に半角の空白を含めないでください。指定内容が次に示す例のどれかに該当する場合，DBID は正しく設定されません。

（例）

@DABENVGRP=＜環境変数グループ名＞の指定内容に半角スペースを含みます。環境変数グループ名は HiRDB_ENV_GROUP（Windows の場合），または/HiRDB_P/Client/HiRDB.ini（UNIX の場合）とします。

半角の空白を△で示します。

- Windows の場合

```
@△DABENVGRP=HiRDB_ENV_GROUP
@DABENVGRP△=HiRDB_ENV_GROUP
@DABENVGRP=△HiRDB_ENV_GROUP
@DABENVGRP=HiRDB_ENV_GROUP△
```

- UNIX の場合

```
@△DABENVGRP=/HiRDB_P/Client/HiRDB.ini
@DABENVGRP△=/HiRDB_P/Client/HiRDB.ini
@DABENVGRP=△/HiRDB_P/Client/HiRDB.ini
@DABENVGRP=/HiRDB_P/Client/HiRDB.ini△
```

形式

```
public void setDescription(String description);
```

パラメタ

description：

接続付加情報を指定します。

例外

なし

戻り値

なし

setEncodLang メソッド

説明

エンコード文字形態を設定します。

String オブジェクトの取得時に unicode から変換する場合，JavaVM の標準エンコードを利用しないで，指定したエンコードを利用します。

なお，UNIX の場合，setEncodLang に指定する言語モードは，DABroker Library の言語モードと合わせる必要があります。言語モードは，DABroker Library 動作環境定義ファイルの共通設定項目の DAB_LANG（LANG 環境変数）で指定します。DABroker Library 動作環境定義ファイルについては，「4.24　DABroker Library 動作環境定義ファイル」を参照してください。

形式

```
public void setEncodLang(String encode_lang);
```

パラメタ

encode_lang：
　データ変換で使用する文字エンコーディング情報を指定します。

例外

なし

戻り値

なし

setExecuteDirectMode メソッド

説明

接続データベースが HiRDB の場合，INSERT，UPDATE，DELETE など，Statement クラスを使用したデータベースの更新で，HiRDB の Execute Direct 機能を使用するかどうかを設定します。

このメソッドが呼び出されない場合は，false を設定します。

この指定は接続データベースが HiRDB の場合だけ有効です。HiRDB 以外のデータベースの場合は，指定の有無にかかわらず，Execute Direct 機能を使用しません。

形式

```
public void setExecuteDirectMode(boolean Mode);
```

パラメタ

Mode：

HiRDB の Execute Direct 機能を使用するかどうかを次の値で設定します。

- true
  Execute Direct 機能を使用します。
- false
  Execute Direct 機能を使用しません。

例外

なし

戻り値

なし

setHiRDBCursorMode メソッド

説明

HiRDB で検索時にカーソルが複数の Commit にわたって有効かどうかを設定します。このメソッドが呼び出されない場合は，false が設定されます。

なお，この機能を使用する場合，マニュアル「HiRDB SQL リファレンス」の DECLARE CURSOR（カーソル宣言）形式 1 を参照してください。

形式

```
public void setHiRDBCursorMode(boolean Mode);
```

パラメタ

Mode：

HiRDB で検索時にカーソルが複数の Commit にわたって有効かどうかを次の値で設定します。

- true
  カーソルは保持されます。アプリケーションは続けて Fetch できます（LOCK TABLE UNTIL DISCONNECT が前提です）。
- false
  カーソルはクローズされますが，ステートメントは有効です。アプリケーションは，Prepare しないで，再度 Execute できます。

例外

なし

戻り値

なし

注意事項

- 接続データベースが HiRDB 以外の場合，指定値は無視されます。
- ORACLE では，常にカーソルは複数の Commit，または Rollback にわたって有効です。
- SELECT，INSERT，DELETE，UPDATE，PURGE および CALL のどれかでアクセスする次のスキーマ資源に対して，ほかのユーザが定義系 SQL 文を発行すると，スキーマ資源にアクセスしていたコネクションを DISCONNECT するまでの間，定義系 SQL は排他待ちの状態になります。
  - 表
  - ビュー
  - ストアドプロシジャ
  - ストアドファンクション
  - 抽象データ型
- XADataSource を使用して，バージョンが 07-01 より前の HiRDB に接続した場合，true にしても有効になりません。

true を設定する場合の注意事項

- MetaData を取得する場合，true を指定して LOCK TABLE UNTIL DISCONNECT をしないで，複数の Commit にわたって Fetch をするとエラーが発生します。MetaData を取得する場合は，true を設定しないでください。
- 接続データベースが HiRDB の場合でかつ，検索 SQL（SELECT 文）に「UNTIL DISCONNECT」を記述した場合は，必ず接続時のプロパティまたは URL に HIRDB_CURSOR=true を設定するか，このメソッドで true を設定してください。これらの設定をしなかった場合，Commit をわたった検索中の ResultSet クラスオブジェクトに対して close メソッドを実行してもカーソルをクローズしません。
- 次の表にはアクセスできません。アクセスした場合，HiRDB のエラーになります。
  - 分散表
  - ディクショナリ表
  - 抽象データ型を含む表
  - 外部表，または外部表を基表とするビュー表
  - 関数呼び出しを指定して導出した名前付きの導出表（ビュー表，WITH 句の問い合わせ）
- カーソルをオープンする前に，そのカーソルを使用する表に対して UNTIL DISCONNECT 指定の LOCK TABLE 文を実行しておく必要があります。
- ROLLBACK を発行すると，使用しているカーソルはすべて閉じられます。
- 排他オプションに WITHOUT LOCK WAIT，または WITHOUT LOCK NOWAIT を指定できません。また，クライアント環境変数でデータ保証レベル(PDISLLVL)に 0，または 1 を指定した場合でも 2（WITH SHARE LOCK 相当）を仮定します。
- カーソルがオープンしている場合，同じコネクションで定義系 SQL は実行できません。また，カーソルがクローズしている場合，同じコネクションで定義系 SQL を実行すると SQL の前処理は無効になります。
- SELECT 文を実行し，その SELECT 文中で使用している表に対して PURGE TABLE 文を実行すると，カーソルはクローズされます。
- オープンしているカーソルで指定した表を，同じコネクションの別のステートメントでオープンすることはできません。

- 次のオペランドを指定する場合は，LOCK 文に IN EXCLUSIVE MODE を指定してください。
  - WITH EXCLUSIVE LOCK
  - FOR UPDATE 句

false を設定する場合の注意事項

- 通常，Fetch 時，DABroker Library に Fetch データをバッファリングするため，Commit 直後の Fetch でエラーが発生するとは限りません。
- MetaData を取得する場合，false またはデフォルトの状態で ResultSet の Fetch 実行中に，Commit を実行するとエラーが発生します。
- SELECT，INSERT，DELETE，UPDATE，PURGE および CALL のどれかでアクセスする次のスキーマ資源に対して，ほかのユーザが定義系 SQL 文を発行すると，スキーマ資源にアクセスしていたコネクションを DISCONNECT するまでの間，定義系 SQL は排他待ちの状態になります。
  - 表
  - ビュー
  - ストアドプロシジャ
  - ストアドファンクション

setJDBC_IF_TRC メソッド

説明

JDBC インタフェースメソッドトレースの取得の有無を設定します。

このメソッドが呼び出されない場合は，false を設定します。

なお，別途 setLogWriter メソッドで有効なログストリームを指定する必要があります。

形式

```
public void setJDBC_IF_TRC(boolean flag);
```

パラメタ

flag：

トレースの取得の有無を次の値で指定します。

- true
  取得します。
- false
  取得しません。

例外

なし

戻り値

なし

setLONGVARBINARY_Access メソッド

説明

LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースアクセス方法を設定します。

なお，次の場合，"LOCATOR"を指定しても"REAL"が指定されたものとみなされます。

- 接続先データベースが HiRDB 以外の場合。
- 接続先データベースである HiRDB の HiRDB サーバまたは HiRDB クライアントライブラリのバージョンが 07-00 以前の場合。

形式

```
public void setLONGVARBINARY_Access(String Mode);
```

パラメタ

String Mode：

LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースアクセス方法を，次の値で設定します。デフォルトは，"REAL"です。

- "REAL"
  実データでアクセスします。
- "LOCATOR"
  HiRDB の位置づけ子（locator）機能を使用してアクセスします。

例外

なし

戻り値

なし

setNetworkProtocol メソッド

説明

DABroker Library との接続種別を設定します。なお，このメソッドは，データベースとの接続時に必ず呼び出します。

形式

```
public void setNetworkProtocol(String network_protocol);
```

パラメタ

network_protocol：

接続種別を指定します。

指定できる接続種別を，次に示します。

接続種別	設定する内容※
ネイティブライブラリ接続（ローカルアクセス）	"lib"

注※

設定する内容には，大文字，小文字および大文字・小文字の混在する内容を指定できます。設定できる種別はネイティブライブラリ接続だけです。

例外

引数の内容が null の場合，または接続種別に指定できる内容以外が指定された場合は，SQLException をスローします。

戻り値

なし

setNotErrorOccurred メソッド

説明

ConnectionEventListener.connectionErrorOccurred の発生を抑止するかどうかを設定します。

ConnectionPooledDataSource，XADataSource 使用時に致命的な接続エラーが発生した場合，ConnectionEventListener.connectionErrorOccurred が呼ばれますが，このメソッドで，その呼び出しを抑止できます。

このメソッドが呼び出されない場合は，false が設定されます。

形式

```
public void setNotErrorOccurred(boolean Mode);
```

パラメタ

Mode：

connectionErrorOccurred の発生を抑止するかどうかを次の値で設定します。

- true
  connectionErrorOccurred を発生させません。
- false
  connectionErrorOccurred を発生させます。

例外

なし

戻り値

なし

注意事項

- 通常は，このメソッドを呼び出さないか，false を設定してください。

- ORACLE の FailOver 機能，または HiRDB の再接続機能を使用する場合は，true を設定してください。

setOSAuthorize メソッド

説明

OS 認証機能を使用してデータベースに接続するかどうかを設定します。

このメソッドが呼び出されない場合は，false が設定されます。

なお，Oracle に対する OS 認証機能を使用する場合は，アプリケーションサーバ，または Web コンテナサーバを OS 認証で使用するユーザで起動する必要があります。

このメソッドは，Oracle9i 9.2.0 または Oracle10g 10.1.0 に接続する場合だけ有効です。それ以外のバージョンの Oracle に接続した場合は動作を保証しません。接続するデータベースが ORACLE 以外の場合は，指定値に関係なく，OS 認証機能を使用しません。

形式

```
public void setOSAuthorize(boolean Mode);
```

パラメタ

Mode：

OS 認証機能を使用するかどうかを次の値で指定します。

- true
  OS 認証機能を使用します。
- false
  OS 認証機能を使用しません。

例外

なし

戻り値

なし

setPassword メソッド

説明

データベースアクセスのパスワードを設定します。

パスワードは，getConnection メソッドの引数でも指定できます。

このメソッドでパスワードを設定し，ユーザ ID とパスワードを引数に持つ getConnection メソッドを呼び出した場合は，getConnection メソッドで指定したパスワードが優先されます。

形式

```
public void setPassword(String password);
```

パラメタ

password：
　パスワードを指定します。

例外

なし

戻り値

なし

setPortNumber メソッド

説明

接続する DABroker Library のポート番号を設定します。

このメソッドが呼び出されない場合は，ポート番号として「40179」を使用します。

形式

```
public void setPortNumber(int port_no);
```

パラメタ

port_no：
　DABroker Library のポート番号を指定します。

例外

引数の内容が 0 より小さい場合，SQLException をスローします。

戻り値

なし

setRowSize メソッド

説明

JDBC で取り扱うバッファ長を，メガバイト単位の数字文字列で指定します。

指定を省略した場合，数字以外を指定した場合，または 16 より小さい値を指定した場合は 16 を仮定します。

512 より大きい値を指定した場合は 512 を仮定します。

形式

```
public void setRowSize(int row_size);
```

パラメタ

row_size：
　バッファ長を 16～512 の範囲のメガバイト単位で指定します。

例外

なし

戻り値

なし

setServerName メソッド

説明

接続する DABroker Library のホスト名，または IP アドレスを設定します。

形式

```
public void setServerName(String server_name);
```

パラメタ

server_name：
　DABroker Library のホスト名または IP アドレスを指定します。

例外

引数の内容が null の場合，SQLException をスローします。

戻り値

なし

setSQLWarningIgnore メソッド

説明

データベースから返される警告を Connection クラスで保持するかどうかの情報を設定します。このメソッドが呼び出されない場合は，false が設定されます。

形式

```
public void setSQLWarningIgnore(boolean Mode);
```

パラメタ

Mode：
　警告を保持するかどうかを次の値で設定します。

- true
  警告を保持しません。

- false
  警告を保持します。

例外

なし

戻り値

なし

setSV_EVENT_TRC メソッド

説明

DABroker Library とのイベントトレースの取得の有無を設定します。

このメソッドが呼び出されない場合は，false を設定します。

なお，別途 setLogWriter メソッドで有効なログストリームを指定する必要があります。

形式

```
public void setSV_EVENT_TRC(boolean flag);
```

パラメタ

flag：
トレースの取得の有無を次の値で指定します。

- true
  取得します。
- false
  取得しません。

例外

なし

戻り値

なし

setTRC_NO メソッド

説明

トレースのエントリ数を設定します。

このメソッドが呼び出されない場合は，トレースのエントリ数は 500 となります。

10～1000 以外の値が設定された場合は，トレースを取得しません。

形式

```
public void setTRC_NO(int trc_no);
```

パラメタ

trc_no：
　トレースのエントリ数を 10～1000 の範囲で指定します。

例外

なし

戻り値

なし

setUapName メソッド

説明

アプリケーション名称を設定します。

このメソッドが呼び出されない場合は，アプリケーション名称として，JDBC ドライバの製品名称を DABroker Library に通知します。

設定されたアプリケーション名称は，DABroker Library のデータベースアクセストレースの PAPNAME，および拡張データベースアクセストレースの Client Name に出力されます。

形式

```
public void setUapName(String uap_name);
```

パラメタ

uap_name：
　アプリケーション名称を指定します。

例外

なし

戻り値

なし

setUser メソッド

説明

データベースアクセスのユーザ ID を設定します。

ユーザ ID は，getConnection メソッドの引数でも指定できます。

このメソッドでユーザ ID を設定し，ユーザ ID とパスワードを引数に持つ getConnection メソッドを呼び出した場合は，getConnection メソッドで指定したユーザ ID が優先されます。

形式

```
public void setUser(String user);
```

パラメタ

user：
　ユーザ ID を指定します。

例外

なし

戻り値

なし

getRMID メソッド

説明

setRMID メソッドで設定した，リソースマネジャの識別子を取得します。

setRMID メソッドで識別子が設定されていない場合は，1 を取得します。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public int getRMID();
```

パラメタ

なし

例外

なし

戻り値

int：
　リソースマネジャの識別子を取得します。

getXACloseString メソッド

説明

setXACloseString メソッドで設定した，XA_CLOSE 文字列を取得します。

setXACloseString メソッドで XA_CLOSE 文字列が設定されていない場合は，null を取得します。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public String getXACloseString();
```

パラメタ

なし

例外

なし

戻り値

String：
　XA_CLOSE 文字列を取得します。

getXALocalCommitMode メソッド

説明

XA 使用時，トランザクションが分散トランザクションでないとき，データベースのオートコミットモードを有効にしているかどうかの設定情報を取得します。

形式

```
public boolean getXALocalCommitMode();
```

パラメタ

なし

例外

なし

戻り値

boolean：
　XA 使用時のオートコミットの設定情報を次の値で取得します。

- true
  データベースのオートコミットが有効です。
- false
  データベースのオートコミットは無効です。

getXAOpenString メソッド

説明

setXAOpenString メソッドで設定した，XA_OPEN 文字列を取得します。

setXAOpenString メソッドで XA_OPEN 文字列が設定されていない場合は，null を取得します。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public String getXAOpenString();
```

パラメタ

なし

例外

なし

戻り値

String：
　XA_OPEN 文字列を取得します。

getXAThreadMode メソッド

説明

setXAThreadMode メソッドで設定した，XA 使用時のスレッドモードを取得します。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public boolean getXAThreadMode();
```

パラメタ

なし

例外

なし

戻り値

boolean：
　XA 使用時のスレッドモードを次の値で取得します。

- true
  マルチスレッドモード
- false
  シングルスレッドモード

setRMID メソッド

説明

リソースマネジャの識別子を設定します。

このメソッドが呼び出されない場合は，識別子は 1 となります。

複数のリソースマネジャを同時に使用する場合，リソースマネジャごとに一意な識別子を設定してください。

リソースマネジャの識別子については，各 DBMS のマニュアルを参照してください。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public void setRMID(int rmid);
```

パラメタ

rmid：
　リソースマネジャの識別子を，1 以上の正の数値で指定します。

例外

引数の内容が 1 より小さい場合，SQLException をスローします。

戻り値

なし

setXACloseString メソッド

説明

XA_CLOSE 文字列を設定します。なお，XA_CLOSE 文字列については，各 DBMS のマニュアルを参照してください。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public void setXACloseString(String xa_string);
```

パラメタ

xa_string：
　XA_CLOSE 文字列を指定します。

例外

なし

戻り値

なし

setXALocalCommitMode メソッド

説明

XA 使用時，トランザクションが分散トランザクションでないとき，データベースのオートコミットモードを有効にするかどうかを設定します。

このメソッドが呼び出されない場合は，false を設定します。

使用するアプリケーションサーバで FullJTA（J2ee1.3）の機能を使用する場合は，必ず true を設定してください。

形式

```
public void setXALocalCommitMode(boolean AutoCommitMode);
```

パラメタ

AutoCommitMode：

オートコミットを次の値で設定します。

- true
  データベースのオートコミットを有効にします。
- false
  データベースのオートコミットを無効にします。

例外

なし

戻り値

なし

setXAOpenString メソッド

説明

XA_OPEN 文字列を設定します。なお，XA_OPEN 文字列については，各 DBMS のマニュアルを参照してください。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public void setXAOpenString(String xa_string);
```

パラメタ

xa_string：
　XA_OPEN 文字列を指定します。

例外

なし

戻り値

なし

setXAThreadMode メソッド

説明

XA 使用時のスレッドモードを設定します。

このメソッドが呼び出されない場合は，false を設定します。

リソースマネジャが提供する XA ライブラリがマルチスレッド対応で，アプリケーションがマルチスレッドで動作する場合は，true を設定してください。

このメソッドは，JdbcDbpsvXADataSource クラスでだけ提供しています。

形式

```
public void setXAThreadMode(boolean mode);
```

パラメタ

mode：
　XA 使用時のスレッドモードを次の値で指定します。

- true
  マルチスレッドモード
- false
  シングルスレッドモード

例外

なし

戻り値

なし

4.20　Blob インタフェース

説明

Blob インタフェースでは，主に次の機能を提供します。

- Blob データの取得

Blob インタフェースの提供する各メソッドの詳細，および使用方法については，JavaSoft 提供の JDBC 関連ドキュメントを参照してください。

注意事項

- ResultSet の getBlob メソッド，または CallableStatement の getBlob メソッドで取得した Blob オブジェクト

  ResultSet の getBlob メソッド，または CallableStatement の getBlob メソッドで取得した Blob オブジェクトを使用する場合，ロケータアクセスであるかどうかで動作が異なります。

ロケータアクセスでない場合

ResultSet の getBlob メソッド，または CallableStatement の getBlob メソッドで取得した時点でのデータを?パラメタの値とします。

ロケータアクセスの場合

setBlob()メソッドを呼び出したとき，内部で Blob.getBytes(1, (int)(Blob.length()))を実行します。?パラメタの値は Blob.getBytes(1, (int)(Blob.length()))で得られたデータとなります。

制限事項

次のメソッドは未サポートです。呼び出しをすると SQLException をスローします。

- setBinaryStream
- setBytes
- truncate

4.21　DABroker Library で使用するコマンドの一覧

DABroker Library で使用するコマンドの一覧を，次の表に示します。

表 4–89　DABroker Library で使用するコマンドの一覧

コマンド名	主な機能
dabsetup コマンド	• DABroker Library のセットアップ

4.22　dabsetup（DABroker Library のセットアップ）

UNIX の場合，DABroker Library をセットアップするには，dabsetup コマンドを使用します。

なお，dabsetup コマンドを実行する前に，次の操作が必要です。

- DABroker Library の管理ユーザの登録
- DABroker Library インストールディレクトリ（/opt/DABroker）の属性（ユーザ，グループ）の変更

DABroker Library のセットアップについては，「4.3　DABroker Library の初期設定」を参照してください。

コマンドの格納先

```
/opt/DABroker/bin/
```

(1)　形式

```
dabsetup [-d] [<パス名>]
```

(2)　機能

DABroker Library の運用ディレクトリを設定します。dabsetup コマンドを実行すると，インストールディレクトリのディレクトリおよびファイルが，<パス名>に指定した運用ディレクトリに移動されます。

また，-d オプションを指定することで，運用ディレクトリを削除できます。

(3)　引数

-d

DABroker Library 運用ディレクトリ下の DABroker Library システムを，OS から削除します。
DABroker Library 運用ディレクトリを設定する場合は，このオプションを指定しないでください。

<パス名>

DABroker Library を運用したい任意のディレクトリを指定します。

ここで指定したディレクトリ下に"DABroker"ディレクトリが作成されて，インストールディレクトリのディレクトリおよびファイルが移動されます。

指定を省略した場合，DABroker Library 運用ディレクトリは DABroker Library のインストールディレクトリ（/opt/DABroker）となります。なお，リモートファイルシステム上のディレクトリ，リモートファイルシステム，またはローカルファイルとリンクしたディレクトリは指定できません。

(4)　入力例

DABroker Library 運用ディレクトリを設定する場合

```
dabsetup /home
```

DABroker Library システムを削除する場合

```
dabsetup -d /home
```

(5)　注意事項

- dabsetup コマンドは，スーパーユーザで実行してください。
- dabsetup コマンドは，DABroker Library が停止中に実行してください。

- dabsetup コマンドで DABroker Library を再度セットアップする場合，運用ディレクトリの DABroker Library の動作状態は初期状態になります。ただし，ユーザが作成した DABroker Library 動作環境定義ファイル，ログファイル，およびトレースファイルなどはセットアップされません。
- 運用ディレクトリとして指定したパスが，ファイルとして存在する場合，dabsetup コマンドの実行を中止します。
- 運用ディレクトリとして指定したパスに，DABroker Library が提供するディレクトリと同一名称のファイル，または DABroker Library が提供するファイルと同一名称のディレクトリが存在する場合，dabsetup コマンドの実行を中止します。
- 運用ディレクトリとして指定したパスが，ディレクトリとして存在する場合，上書きするかどうかを確認します。なお，上書きすることを選択しても，指定したディレクトリに存在する DABroker Library がセットアップしないファイル（ユーザが作成した DABroker Library 動作環境定義ファイル，ログファイル，およびトレースファイルなど）は保証されます。
- dabsetup コマンドの実行が失敗した場合，dabsetup コマンドの実行を終了して，dabsetup コマンドを実行する前の状態に戻します。

(6)　DABroker Library 運用ディレクトリの削除について

dabsetup コマンドでの運用ディレクトリの削除に成功した場合

- インストールディレクトリに, dabsetup コマンドなどのシステムで必要なファイルが残ります。インストールディレクトリに残ったファイルは，インストーラで削除してください（アンインストールしてください）。
- ユーザが作成した DABroker Library 動作環境定義ファイルは, dabsetup コマンドでは削除されません。必要に応じて削除してください。

dabsetup コマンドでの運用ディレクトリの削除に失敗した場合

- 残ったディレクトリおよびファイルを OS の rm コマンドで削除してください。

4.23　DABroker Library で使用するファイルの一覧

DABroker Library で使用するファイルの一覧を，次の表に示します。

表 4-90　DABroker Library で使用するファイルの一覧

ファイル名称	分類	概要	参照先
dasysconf※	DABroker Library 動作環境定義ファイル	DABroker Library の動作に関する情報を定義します。	4.24

注※

Windows の場合は，環境設定ユーティリティを使用して DABroker Library の動作環境を設定したときに，このファイルが作成されます。DABroker Library の動作環境の設定については，「4.3　DABroker Library の初期設定」を参照してください。

4.24　DABroker Library 動作環境定義ファイル

(1)　形式

DABroker Library 動作環境定義ファイルの形式について，OS ごとに説明します。

(a)　Windows の場合

環境設定ユーティリティを使用して DABroker Library の動作環境を設定したときに，次のように環境変数が設定されます。

〈環境変数〉=〈値〉

DABroker Library 動作環境定義ファイルに設定される項目については，「4.25　共通設定項目」，「4.26　拡張データベースアクセストレース設定項目」，「4.27　リモートアクセス設定項目」，および「4.28　for C++設定項目」を参照してください。

(b)　UNIX の場合

次のように環境変数を指定します。

〈環境変数〉=〈値〉

指定方法

- #で始まる行はコメントとみなされます。
- 「=」の前後にスペースまたはタブがある場合はエラーとなります。
- 値に半角スペースを使用する場合は，'（シングルクォーテーション）で値を囲んで指定してください。
  （例）DABTMP='/tmp/da b'
- 値の後ろに半角スペース，タブおよび#以外の文字を記載した場合は，エラーとなります。
- 値には，「$設定項目」で指定する値を引き継ぐことはできません。

このファイルは DAB ブロックと OTHERS ブロックで構成されます。各ブロックの設定内容を次に示します。

DAB ブロック

DABroker Library の動作環境を設定します。

DABroker Library では，設定項目をチェックし，不正な名称などを指定した場合はエラーになります。なお，ここで指定する項目については，「4.25　共通設定項目」，「4.26　拡張データベースアクセストレース設定項目」，および「4.27　リモートアクセス設定項目」を参照してください。

OTHERS ブロック

DBMS などの動作環境を設定します。

DABroker Library では，設定項目をチェックしません。このセクションに指定されたすべての項目を有効とします。なお，ここで指定する項目については，「4.29　HiRDB の設定項目」および「4.30　ORACLE の設定項目」を参照してください。

(2)　ファイルの格納先

- Windows の場合
  <DABroker Library 運用ディレクトリ>¥conf

- UNIX の場合
  <DABroker Library 運用ディレクトリ>/conf

<DABroker Library 運用ディレクトリ>は，<DABroker Library インストールディレクトリ>です。

UNIX の場合，サンプルファイル（/opt/DABroker/sample/dasysconf）をコピーしてファイルを作成してください。

(3)　機能

DABroker Library の動作に関する情報を定義します。

UNIX の場合，DABroker Library 動作環境定義ファイルで定義する内容は，シェルの環境変数として設定することもできます。その場合，DABroker Library 動作環境定義ファイルで設定した値よりもシェルの環境変数として設定した値が優先されます。

(4)　注意事項

- 値を変更しない設定項目は削除しないでください。
- 設定項目の export は不要です。
- DABroker Library の起動中は，DABroker Library 動作環境定義ファイルを変更しないでください。

4.25　共通設定項目

接続するデータベースに関係なく，共通に設定する項目です。

「4.27　リモートアクセス設定項目」で設定する，次の項目についても有効になります。

- Windows の場合
  - HiRDB の BLOB 型データ受取バッファサイズ
  - ORACLE の LONG 型データ受取バッファサイズ
- UNIX の場合
  - DABHIRDBA_BLOBBUFSIZE（HiRDB の BLOB 型データ受取バッファサイズ）
  - DABORADBA_LONGBUFSIZE（ORACLE の LONG 型データ受取バッファサイズ）

ここでの設定内容を，次に示します。

DAB_LANG（LANG 環境変数）

この項目は，Windows では設定されません。

標準値

- AIX，または Linux の場合
  C
- HP-UX の場合
  ja_JP.SJIS

指定範囲

表 4-91　LANG 環境変数に指定できる値

言語モード	AIX の場合	HP-UX の場合	Linux の場合
SJIS モード	Ja_JP Ja_JP.IBM-932 Ja_JP.IBM-943	ja_JP.SJIS	ja_JP.SJIS japanese.sjis
日本語 EUC モード	ja_JP ja_JP.IBM-eucJP	ja_JP.EUC ja_JP.UJIS ja_JP.eucJP	ja_JP ja_JP.eucjp ja_JP.ujis japanese japanese.euc Ja_JP
ASCII モード	C	C	C
UTF-8 モード	UTF-8	UTF-8※1	UTF-8

注※1　DABroker Library の使用環境の locale（LANG,LC_ALL）には，必ず使用する UTF-8 のロケールを設定してください。

注※2　HiRDB 使用時に指定できます。

説明

HiRDB，または Oracle 使用時に使用する母国語（日本語，英語など）および文字コード（SJIS コードなど）を設定します。

各データベースで使用する言語モードは，必ず DABroker Library の言語モードと合わせてください。データベースと DABroker Library の言語モードが異なっている場合の動作は保証されません。

DABroker Library を日本語 EUC モード，ASCII モード，または UTF-8 モードで動作させた場合，メッセージは英文で出力されます。

注意事項

DABroker Library の LANG 環境変数に SJIS モードが設定されている場合，JDBC では JavaVM の標準エンコードセットを利用して文字変換を実施します。このため，データベースを SJIS モードで構築し，JavaVM が EUC モードのエンコードセットを標準としている場合，データベースには EUC モードで格納されます。データベースに SJIS モードで格納するためには，プロファイルまたは URL に「ENCODELANG=SJIS」を設定します。

DABTMP（作業ディレクトリ）

標準値

- Windows の場合
  <DABroker Library 運用ディレクトリ>¥tmp
- UNIX の場合
  <DABroker Library 運用ディレクトリ>/tmp

指定範囲

パーティションを圧迫しない，任意のディレクトリを指定します。

ただし，Windows の場合，ネットワークドライブ上のディレクトリは指定できません。また，指定したディレクトリのアクセス権限として，システムにフルコントロールを指定してください。

UNIX の場合，DABroker Library の管理ユーザは，ディレクトリにアクセスできる権限（READ および WRITE）が必要です。

説明

DABroker Library の作業ディレクトリを指定します。このディレクトリは，予約処理用の SQL の格納および，処理途中で作成される一時ファイルの格納などに使用されます。

なお，UNIX の場合，ここで/tmp を指定したら，OS の再起動時に，/tmp ディレクトリ下に作成されたすべてのファイルは削除されます。

DABSVLOGSIZE（ログファイルサイズ［dabsv.log］）

標準値

64（キロバイト）

指定範囲

0 または 64～32767（キロバイト）

説明

DABroker Library のログを取得するファイルの容量を設定します。ログは，<DABroker Library 運用ディレクトリ>¥spool¥dabsv.log（UNIX の場合，<DABroker Library 運用ディレクトリ>/spool/dabsv.log）に出力されます。

- 0 を設定した場合

ファイルの容量は無限大になります。

- 64～32767 を設定した場合

  設定した値がファイルの容量になります。

注意事項

ログファイルが満杯になったら，ラップアラウンドでログを取得します。ラップアラウンドした場合，1 世代前のバックアップファイル（dabsv.logold）を作成し，1 世代だけを管理します。

DABSELECTCOLCNT（表検索カラム数）

標準値

0

指定範囲

0 または 1～2147483647

説明

表検索用の領域として確保する表カラムの個数を指定します。

- 0 を設定した場合

  DBMS に対して表カラムの個数を要求する SQL を発行します。DBMS から返された結果を基に，表検索用の領域を確保します。

- 1～2147483647 を設定した場合

  表カラムの個数が指定した値より少ない場合，DBMS に対して表カラムの個数を要求する SQL を発行しません。このため，SQL を発行する場合より処理性能が向上します。ただし，検索を実行するたびに次に示す大きさのメモリが必要になるため，動作環境を考慮して値を設定してください。よく検索する表のカラム数を大きく超える値を設定した場合，性能劣化の原因になることがあります。

  HiRDB，ORACLE の場合：1 カラム当たり 128 バイト

  表カラムの個数が指定した値より多い場合，DBMS に対して表カラムの個数を要求する SQL を発行します。DBMS から返された結果を基に，表検索用の領域を確保します。

DABSELECTINFCNT（更新・検索条件カラム数）

標準値

0

指定範囲

0 または 1～2147483647

説明

表の更新・検索用の領域として確保する，入力項目（パラメタ）の個数を指定します。

- 0 を設定した場合

  DBMS に対して入力項目の個数を要求する SQL を発行します。DBMS から返された結果を基に，表の更新・検索用の領域を確保します。

- 1～2147483647 を設定した場合

  入力項目の個数が指定した値より少ない場合，DBMS に対して入力項目の個数を要求する SQL を発行しません。このため，SQL を発行する場合より処理性能が向上します。ただし，更新・検索を実行するたびに次に示す大きさのメモリが必要になるため，動作環境を考慮して値を設定してくだ

さい。よく更新・検索する表の入力項目の個数を大きく超える値を設定した場合，性能劣化の原因になることがあります。

HiRDB，ORACLE の場合：1 カラム当たり 128 バイト

入力項目の個数が指定した値より多い場合，DBMS に対して入力項目の個数を要求する SQL を発行します。DBMS から返された結果を基に，表の更新・検索用の領域を確保します。

DABORACLE_TYPE（使用する ORACLE のバージョン）

標準値

- Windows の場合
  ORACLE9i
- AIX，または Linux の場合
  817
- HP-UX の場合
  805

指定範囲

- Windows の場合
  ORACLE7，ORACLE9i または ORACLE10g
- AIX，または Linux の場合
  920，または 101
- HP-UX の場合
  901，920，または 101

説明

使用する ORACLE のバージョンを設定します。

901：Oracle9i R9.0.1

920：Oracle9i R9.2.0

101：Oracle10g R10.1.0

DABSQL_HiRDB_DICTIONARY_DATATYPE（HiRDB のディクショナリ表のデータ属性）

標準値

C

指定範囲

M または C

説明

HiRDB のディクショナリ表の列属性を指定します。

- M を設定した場合
  混在文字データ（MVARCHAR）として処理します。
- C を設定した場合
  文字データ（VARCHAR）として処理します。

注意事項

ここでの指定と HiRDB で使用されているディクショナリ表のデータ属性が異なる場合は，結果不正や SQL エラーになる場合があります。

DAB_HIRDB_DBINF_ESC（HiRDB 環境変数区切り文字）

標準値

@

指定範囲

@ ! $ % & * + - / : ? ¥ |

説明

HiRDB 環境変数区切り文字を指定します。

注意事項

特に問題がない場合，区切り文字には@を使用してください。

区切り文字は必ず半角 1 文字で指定してください。全角文字は指定できません。

DABroker Library から HiRDB クライアント環境変数グループを指定してデータベースと接続する場合は，標準値「@」を指定してください。

DABEXSQL_TRC_LINE（拡張データベースアクセストレースを取得する）

標準値

- Windows の場合
  OFF
- UNIX の場合
  -1

指定範囲

- Windows の場合
  OFF または ON
- UNIX の場合
  -1，0，または 1024～32767（行）

説明

拡張データベースアクセストレースを取得するかどうかを設定します。

Windows の場合

- OFF（チェックなし）を設定した場合
  拡張データベースアクセストレースを取得しません。
- ON（チェックあり）を設定した場合
  拡張データベースアクセストレースを取得します。この場合，トレースファイルサイズを指定します。

なお，拡張データベースアクセストレースのファイルの出力行数は，トレースファイルサイズの設定値に従います。

UNIX の場合

- -1 を設定した場合
  拡張データベースアクセストレースを取得しません。
- 0 を設定した場合
  拡張データベースアクセストレースを取得します。出力行数は，無限大となります。
- 1024～32767 を設定した場合
  拡張データベースアクセストレースを取得します。出力行数は，設定した数値分となります。

拡張データベースアクセストレースファイルは，1 行に 134 バイトまで出力され，出力行数はここでの設定値に従います。ただし，ヘッダは行数に含まれません。

トレースファイルサイズ

この項目は，UNIX では設定不要です。

標準値

1024（単位：行）

指定範囲

0，または 1024～32767

説明

DABEXSQL_TRC_LINE（拡張データベースアクセストレースを取得する）を ON にした場合，トレースファイルサイズを指定します。

拡張データベースアクセストレースのトレースファイルサイズを指定します。

- 0 を設定した場合
  出力行数は，無限大となります。
- 1024～32767 を設定した場合
  拡張データベースアクセストレースを取得します。出力行数は，設定した数値分となります。

なお，拡張データベースアクセストレースのファイルは，1 行に 134 バイトまで出力されます。拡張データベースアクセストレースのファイルの出力行数は，ここでの設定値に従います。ただし，ヘッダは行数に含まれません。

DABXA_LOG_ERROR（XA トレースを取得する）

標準値

ON

指定範囲

ON または OFF

説明

XA インタフェースでエラーを検知した場合，エラートレースを取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  エラートレースを取得しません。
- ON（チェックあり）を設定した場合
  エラートレースを取得します。

XA トレースは，DABroker Library 運用ディレクトリ下の spool ディレクトリに db_xainfo.log というファイル名で取得されます。XA トレースファイル（db_xainfo.log）はテキスト形式なので，任意のテキストエディタで参照してください。

XA トレースファイルは，運用期間に応じてファイルサイズが増加するため，定期的に削除する必要があります。

4.26　拡張データベースアクセストレース設定項目

拡張データベースアクセストレースで取得する項目の設定内容を次に示します。

ここでの設定は，DABEXSQL_TRC_LINE（拡張データベースアクセストレースを取得する）に 0 または 1024～32767 を指定した場合に有効になります。

すべての項目を設定する必要はありません。省略した場合，標準値が仮定されます。また，「ON」が設定された項目に該当する，すべてのデータベースへの要求がなかった場合，拡張データベースアクセストレースにはヘッダ（出力項目名）だけが出力されます。

DABSQL_TRC_CANCEL（CANCEL）

標準値

ON

指定範囲

ON または OFF

説明

DBMS に対する処理中断要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  取得しません。
- ON（チェックあり）を設定した場合
  取得します。

DABSQL_TRC_CLOSE（CLOSE）

標準値

ON

指定範囲

ON または OFF

説明

カーソルを閉じる要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  取得しません。
- ON（チェックあり）を設定した場合
  取得します。

DABSQL_TRC_COMMIT（COMMIT）

標準値

ON

指定範囲

ON または OFF

説明

トランザクションの正常終了，更新を有効にする要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  取得しません。
- ON（チェックあり）を設定した場合
  取得します。

DABSQL_TRC_CONNECT（CONNECT）

標準値

ON

指定範囲

ON または OFF

説明

データベースとの接続要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  取得しません。
- ON（チェックあり）を設定した場合
  取得します。

DABSQL_TRC_DECLARE（DECLARE）

標準値

ON

指定範囲

ON または OFF

説明

前処理した SELECT に対してカーソルを宣言する要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  取得しません。
- ON（チェックあり）を設定した場合
  取得します。

DABSQL_TRC_DESCRIBE（DESCRIBE）

標準値

ON

指定範囲

ON または OFF

説明

前処理した SQL の情報を取得する要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合

取得しません。

- ON（チェックあり）を設定した場合
  取得します。

DABSQL_TRC_DISCONNECT（DISCONNECT）

標準値

ON

指定範囲

ON または OFF

説明

トランザクションを正常終了，切り離し要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_EXECUTE（EXECUTE）

標準値

ON

指定範囲

ON または OFF

説明

前処理した SQL の実行要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_FETCH（FETCH）

標準値

ON

指定範囲

ON または OFF

説明

行を読み込んでカーソルを進める要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_LOGGING（LOGGING）

標準値

ON

指定範囲

ON または OFF

説明

LOGGING 要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_LOGLESS（LOGLESS）

標準値

ON

指定範囲

ON または OFF

説明

LOGLESS 要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_PREPARE（PREPARE）

標準値

ON

指定範囲

ON または OFF

説明

SQL の前処理要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_ROLLBACK（ROLLBACK）

標準値

ON

指定範囲

ON または OFF

説明

トランザクションを取り消し，更新を無効にする要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_OPEN（OPEN）

標準値

ON

指定範囲

ON または OFF

説明

カーソルの位置づけ要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_ATTACH（ATTACH）

標準値

ON

指定範囲

ON または OFF

説明

サーバとのアクセスパス作成要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_ATTRGET（ATTRGET）

標準値

ON

指定範囲

ON または OFF

説明

ハンドルの属性取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_ATTRSET（ATTRSET）

標準値

ON

指定範囲

ON または OFF

説明

ハンドルの属性設定要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_BIND（BIND）

標準値

ON

指定範囲

ON または OFF

説明

パラメタのバインド要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_CHARFORM（CHARFORM）

標準値

ON

指定範囲

ON または OFF

説明

キャラクタセットフォームの取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_CHARID（CHARID）

標準値

ON

指定範囲

ON または OFF

説明

キャラクタセット ID の取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_DEFINE（DEFINE）

標準値

ON

指定範囲

ON または OFF

説明

パラメタの定義要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_DESALLOC（DESALLOC）

標準値

ON

指定範囲

ON または OFF

説明

記述子の領域確保の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_DESCANY（DESCANY）

標準値

ON

指定範囲

ON または OFF

説明

オブジェクトの記述要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_DESFREE（DESFREE）

標準値

ON

指定範囲

ON または OFF

説明

記述子の領域解放の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_DETACH（DETACH）

標準値

ON

指定範囲

ON または OFF

説明

サーバとのアクセスパス削除要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_ENVCREATE（ENVCREATE）

標準値

ON

指定範囲

ON または OFF

説明

環境作成要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_ERRGET（ERRGET）

標準値

ON

指定範囲

ON または OFF

説明

エラー情報取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_GETPIECE（GETPIECE）

標準値

ON

指定範囲

ON または OFF

説明

ピース情報取得要求の情報を取得するかどうかを指定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_HDLALLOC（HDLALLOC）

標準値

ON

指定範囲

ON または OFF

説明

ハンドルの領域確保の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_HDLFREE（HDLFREE）

標準値

ON

指定範囲

ON または OFF

説明

ハンドルの領域解放の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_LOBCLOSE（LOBCLOSE）

標準値

ON

指定範囲

ON または OFF

説明

BFILE データのクローズ要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_LOBLEN（LOBLEN）

標準値

ON

指定範囲

ON または OFF

説明

LOB データの長さ取得要求の情報を取得するかどうかを指定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_LOBOPEN（LOBOPEN）

標準値

ON

指定範囲

ON または OFF

説明

BFILE データのオープン要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_LOBREAD（LOBREAD）

標準値

ON

指定範囲

ON または OFF

説明

LOB データの読み込み要求の情報を取得するかどうかを指定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_LOBWRITE（LOBWRITE）

標準値

ON

指定範囲

ON または OFF

説明

LOB データの書き込み要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_PARAMGET（PARAMGET）

標準値

ON

指定範囲

ON または OFF

説明

パラメタ情報取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_SESBEGIN（SESBEGIN）

標準値

ON

指定範囲

ON または OFF

説明

セッション開始要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_SESEND（SESEND）

標準値

ON

指定範囲

ON または OFF

説明

セッション終了要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_SETPIECE（SETPIECE）

標準値

ON

指定範囲

ON または OFF

説明

ピース情報設定要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

この項目は，DABroker Library のデータベース種別に Oracle8i を設定している場合だけ有効です。

DABSQL_TRC_GETDIAG（GETDIAG）

標準値

ON

指定範囲

ON または OFF

説明

エラー情報取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_AUTO_COMMIT_OFF（AUTO COMMIT OFF）

標準値

ON

指定範囲

ON または OFF

説明

オートコミットモード解除要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_AUTO_COMMIT_ON（AUTO COMMIT ON）

標準値

ON

指定範囲

ON または OFF

説明

オートコミットモード設定要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_GET_INFO（GET INFO）

標準値

ON

指定範囲

ON または OFF

説明

接続後の動作情報取得要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_SET_INFO（SET INFO）

標準値

ON

指定範囲

ON または OFF

説明

接続後の動作情報設定要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

DABSQL_TRC_EXECUTE_DIRECT（EXECUTE DIRECT）

標準値

ON

指定範囲

ON または OFF

説明

SQL の前処理後に実行する要求の情報を取得するかどうかを設定します。

- OFF（チェックなし）を設定した場合
  情報を取得しません。
- ON（チェックあり）を設定した場合
  情報を取得します。

4.27　リモートアクセス設定項目

「4.25　共通設定項目」で設定した項目以外に設定できる項目です。ここでの設定内容を，次に示します。

DABHIRDBA_BLOBBUFSIZE（HiRDB の BLOB 型データ受取バッファサイズ）

標準値

4096（バイト）

指定範囲

0，または 1～2147483647（バイト）

説明

HiRDB から BLOB 型データを抽出する場合の受け取りサイズを指定します。

このデータ型の場合，HiRDB では最大 2147483647 バイトのデータの存在が考えられますが，DABroker Library ではここで指定した値までのデータを受け取り，以降のデータは切り捨てます。

0 を指定した場合，DBMS の定義長分の領域長を仮定します。ただし，実際に格納されているデータの大きさにかかわらず，無条件に定義長分の領域が確保されるため，メモリ効率が悪くなることがあります。格納されているデータの最大長がわかっている場合は，その値を指定することを推奨します。

DABORADBA_LONGBUFSIZE（ORACLE の LONG 型データ受取バッファサイズ）

標準値

4096（バイト）

指定範囲

0，または 1～2147483647（バイト）

説明

ORACLE から，LONG 型データを抽出する場合の受け取りサイズを指定します。

このデータ型の場合，ORACLE では最大 2147483647 バイトのデータの存在が考えられますが，DABroker Library ではここで指定した値までのデータを受け取り，以降のデータは切り捨てます。

0 を指定した場合，DBMS の定義長分の領域長を仮定します。ただし，実際に格納されているデータの大きさにかかわらず，無条件に定義長分の領域が確保されるため，メモリ効率が悪くなることがあります。格納されているデータの最大長がわかっている場合は，その値を指定することを推奨します。

この項目は，データベース種別に Oracle8i を設定している場合は設定しません。

DABLANGMODE（漢字コードセット）

この項目は，UNIX では設定不要です。

標準値

SJIS（シフト JIS モード）

指定範囲

SJIS（シフト JIS モード），EUC（日本語 EUC モード），ASCII（ASCII モード），または UTF-8（UTF-8 モード）

説明

DABroker Library が稼働する言語モードを設定します。

各言語モードで操作できるデータベースを次に示します。

表 4-92　各言語モードで操作できるデータベース

データベース	DABroker Library での設定			
	シフト JIS モード	日本語 EUC モード	ASCII モード	UTF-8 モード
HiRDB	○	○	○	○
ORACLE	○	○	○	○

（凡例）

○：操作できます。

各データベースで使用する言語モードは，必ず DABroker Library の言語モードと合わせてください。データベースと DABroker Library の言語モードが異なっている場合は動作を保証しません。

DABroker Library を日本語 EUC モード，ASCII モード，または UTF-8 モードで動作させた場合，メッセージは英文で出力されます。

4.28　for C++設定項目

DABroker for C++のイベントトレースについて設定します。なお，for C++設定項目は，Windows で DABroker for C++を使用してデータベースに接続する場合の設定項目です。UNIX では，この項目は設定不要です。

トレースの取得レベル，取得種別など，イベントトレースの詳細については，マニュアル「DABroker for C++」を参照してください。

DABCPP_EVTTRC_SIZE（トレース取得サイズ）

標準値

32（キロバイト）

指定範囲

-1，0，または 10～32767（キロバイト）

説明

イベントトレースを取得するかどうかを指定します。また，取得する場合は，トレースを取得するファイルの容量を設定します。なお，トレースファイルは，<DABroker Library 運用ディレクトリ>¥spool¥dabevttrccpp1 に作成されます。

- -1 を設定した場合
  イベントトレースを取得しません。
- 0 を設定した場合
  イベントトレースを取得します。
  この場合，ファイルの容量を制限しません。
- 10～32767 を設定した場合
  イベントトレースを取得します。
  この場合，設定した値がファイルの容量になります。

注意事項

指定した容量のトレースファイルを作成したらラップアラウンドでトレースを取得します。ラップアラウンドした場合は，1 世代前のバックアップファイル（dabevttrccpp2）を作成し，常に 1 世代だけを管理します。

DABCPP_EVTTRC_LEVEL（トレース取得レベル）

標準値

0

指定範囲

0 または 1

説明

イベントトレースの取得レベルを設定します。

値が大きいほど詳細な情報が取得できます。

4.29　HiRDB の設定項目

接続するデータベースに HiRDB を使用する場合の設定項目です。Windows では，この項目は設定されません。

ここでの設定内容を，次に示します。

PDCLTPATH

トレースファイル格納ディレクトリ

PDHOST

ホスト名

ここで設定した値と異なるホスト名でアプリケーションが接続を要求した場合は，アプリケーションで指定されたホスト名が有効になります。

PDNAMEPORT

HiRDB のシステムポート番号

ここで設定した値と異なるポート番号でアプリケーションが接続を要求した場合は，アプリケーションで指定されたポート番号が有効になります。

PDSWAITTIME

サーバ最大待ち時間

PDSQLTRACE

SQL トレースファイルのサイズ

PDUAPERLOG

エラーログファイルのサイズ

注意事項

- HiRDB のバージョンによっては，必要な設定項目が異なることがありますのでご注意ください。
- PDHOST と PDNAMEPORT の値は，HiRDB の環境で設定されている値を指定してください。そのほかの設定項目では，DABroker Library で設定した値が有効になります。なお，設定内容については，マニュアル「HiRDB UAP 開発ガイド」を参照してください。

4.30　ORACLE の設定項目

接続するデータベースに ORACLE を使用する場合の設定項目です。Windows では，この項目は設定されません。

ここでの設定内容を，次に示します。

ORACLE_HOME

ORACLE のホームディレクトリ

ORACLE_SID

ORACLE のデータベース識別子

ここで設定した値と異なるデータベース識別子でアプリケーションが接続を要求した場合は，アプリケーションで指定されたデータベース識別子が有効になります。

NLS_LANG

母国語指定

注意事項

各設定項目には，ORACLE の環境で設定されている値を指定してください。なお，設定内容については，ORACLE のマニュアルを参照してください。

4.31　Connector 属性ファイルでの設定

DB Connector を使用してデータベースに接続する場合に使用する Connector 属性ファイルの設定内容について説明します。

4.31.1　DB Connector に設定する<config-property>タグに指定できるプロパティ

DABroker Library を使用して，Oracle または HiRDB に接続する場合に，DB Connector に設定する<config-property>タグに指定できる値について説明します。

- DBConnector_DABJ_CP.rar または DBConnector_DABJ_CP_Cosminexus_RM.rar
  トランザクション管理をしない場合，またはローカルトランザクションを使用する場合に使用します。また，Reliable Messaging と連携する場合は，DBConnector_DABJ_CP_Cosminexus_RM.rar を使用します。
- DBConnector_DABJ_XA.rar または DBConnector_DABJ_XA_Cosminexus_RM.rar
  グローバルトランザクションを使用する場合に使用します。また，Reliable Messaging と連携する場合は，DBConnector_DABJ_XA_Cosminexus_RM.rar を使用します。

それぞれのファイルで指定できるプロパティを次の表に示します。

表 4-93　DBConnector_DABJ_CP.rar または DBConnector_DABJ_CP_Cosminexus_RM.rar を使用する場合に指定できるプロパティ

config-property-name	config-property-type	config-property-value
blockUpdate	java.lang.Boolean	接続データベースが HiRDB の場合，？パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。設定された値は，DABroker Library の setBlockUpdate メソッドに渡されます。 • true を指定した場合 複数のパラメタセットを一度に処理します。 • false を指定した場合 複数のパラメタセットを一つずつ分割して処理します。 デフォルト値は false です。
bufferPoolSize	java.lang.Integer	受信データのバッファをプールする最大数を設定します。設定された値は，DABroker Library の setBufferPoolSize メソッドに渡されます。 デフォルト値は 0 です。
bufSize	java.lang.Integer	DABroker Library からの受信データのバッファ長を，1～16000 の範囲（単位：キロバイト）で設定します。設定された値は，DABroker Library の setBufSize メソッドに渡されます。デフォルト値は 64 です。
CallableStatementPoolSize	java.lang.Integer	コネクションプールに割り当てられるコネクションごとの CallableStatement のプール数を設定します。0～1024 の整数で指定します。デフォルト値は 10 です。また，PreparedStatementPoolSize と CallableStatementPoolSize の合計値は，1024 以下に設定

config-property-name	config-property-type	config-property-value
CallableStatementPoolSize	java.lang.Integer	してください。有効範囲外の値を指定した場合は，デフォルト値が設定されます。
CancelStatement	java.lang.Boolean	トランザクションタイムアウトやアプリケーション強制停止時に，Statement クラス，CallableStatement クラスおよび PreparedStatement クラスで実行中の SQL をキャンセルするかどうかを指定します。 • true を指定した場合 実行中の SQL をキャンセルします。 • false を指定した場合 実行中の SQL をキャンセルしません。 デフォルト値は true です。 Oracle に接続する場合で，データベースのサーバまたはクライアントのどちらかが Windows プラットフォームで，専用サーバ接続をする場合，false を設定してください。
ConnectionIDUpdate	java.lang.Boolean	コネクション ID を DataSource#getConnection メソッドごとに更新するかどうかを指定します。 • true を指定した場合 DataSource#getConnection メソッドのたびにコネクション ID を生成します。 • false を指定した場合 DataSource#getConnection メソッドで新規の物理コネクションを確立したときにコネクション ID を生成し，そのあとは更新しません。 デフォルト値は false です。 このプロパティは，接続するデータベースが Oracle の場合に指定できます。
databaseName	java.lang.String	接続するデータベースの種別を設定します。設定された値は，DABroker Library の setDatabaseName メソッドに渡されます。 • HIRDB HiRDB および XDM/RD E2 に接続する場合に指定します。 • ORACLE，ORACLE8I Oracle に接続する場合に指定します。 DBConnector_DABJ_CP_Cosminexus_RM.rar の場合，デフォルト値は HIRDB です。
DBEnv	java.lang.String	DABroker Library の接続先データベース定義情報を設定します。設定された値は，DABroker Library の setDBEnv メソッドに渡されます。
DBHostName	java.lang.String	接続する HiRDB のホスト名を設定します。設定された値は，DABroker Library の setDBHostName メソッドに渡されます。
description	java.lang.String	接続するデータベースに必要な接続付加情報を設定します。設定された値は，DABroker Library の setDescription メソッドに渡されます。

config-property-name	config-property-type	config-property-value
description	java.lang.String	• HIRDB の場合 HiRDB システムのポート番号，または HiRDB クライアントの環境変数グループ名を指定します。 • ORACLE の場合 SQL*Net の接続文字列を指定します。
encodLang	java.lang.String	エンコード文字形態を設定します。設定された値は，DABroker Library の setEncodLang メソッドに渡されます。
executeDirectMode	java.lang.Boolean	接続データベースが HiRDB の場合，Statement クラスを使用した INSERT，UPDATE，DELETE などのデータベースの更新で，HiRDB の EXECUTE DIRECT 機能を使用するかどうかを設定します。設定された値は，DABroker Library の setExecuteDirectMode メソッドに渡されます。 • true を指定した場合 Execute Direct 機能を使用します。 • false を指定した場合 Execute Direct 機能を使用しません。 デフォルト値は false です。
HiRDBCursorMode	java.lang.Boolean	HiRDB で検索時にカーソルが Commit，または Rollback にわたって有効かどうかを設定します。設定された値は，DABroker Library の setHiRDBCursorMode メソッドに渡されます。 • true を指定した場合 カーソルは保持されます。アプリケーションは続けて Fetch することができます。 • false を指定した場合 カーソルはクローズされますが，ステートメントは有効です。アプリケーションは，Prepare しないで，再度 Execute できます。 デフォルト値は false です。
JDBC_IF_TRC	java.lang.Boolean	JDBC インタフェースメソッドトレースの取得の有無を設定します。設定された値は，DABroker Library の setJDBC_IF_TRC メソッドに渡されます。 • true を指定した場合 トレースを取得します。 • false を指定した場合 トレースを取得しません。 なお，true を指定した場合，Driver が出力するトレースは，Component Container からセットされる PrintWriter と，ユーザがコネクションファクトリ（javax.sql.DataSource インタフェース）の setLogWriter()メソッドで指定した PrintWriter に出力します。デフォルト値は false です。
linkedResourceAdapterName	java.lang.String	連携する Reliable Messaging リソースアダプタの表示名を指定します。

config-property-name	config-property-type	config-property-value
linkedResourceAdapterName	java.lang.String	DBConnector_DABJ_CP_Cosminexus_RM.rar を使用する場合に指定できます。
loginTimeout	java.lang.Integer	データベースへの接続試行中に，データソースが待機する最長時間（単位：秒）を設定します。デフォルト値は 0 です。
logLevel	java.lang.String	DB Connector が出力するログ・トレースのレベルを指定します。 次の値が指定できます。 • 0 または ERROR • 10 または WARNING • 20 または INFORMATION デフォルトは，0 または ERROR です。
LONGVARBINARY_Access	java.lang.String	LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースアクセス方法を設定します。設定された値は，DABroker Library の setLONGVARBINARY_Access メソッドに渡されます。 • REAL • LOCATOR デフォルト値は REAL です。
networkProtocol	java.lang.String	DABroker Library との接続種別を設定します。設定された値は，DABroker Library の setNetworkProtocol メソッドに渡されます。 • "lib"
OSAuthorize	java.lang.Boolean	OS 認証機能を使用してデータベースに接続するかどうかを設定します。設定された値は，DABroker Library の setOSAuthorize メソッドに渡されます。 • true を指定した場合 OS 認証機能を使用します。 • false を指定した場合 OS 認証機能を使用しません。 デフォルト値は false です。
portNumber	java.lang.Integer	接続する DABroker Library のポート番号を設定します。設定された値は，DABroker Library の setPortNumber メソッドに渡されます。デフォルト値は 40179 です。
PreparedStatementPoolSize	java.lang.Integer	コネクションプールに割り当てられるコネクションごとの PreparedStatement のプール数を設定します。0～1024 の整数で指定します。デフォルト値は 10 です。また，PreparedStatementPoolSize と CallableStatementPoolSize の合計値は，1024 以下に設定してください。有効範囲外の値を指定した場合は，デフォルト値が設定されます。
rowSize	java.lang.Integer	DABroker Library が JDBC で取り扱うバッファ長を，16～512 の範囲（単位：メガバイト）で設定します。設定された

config-property-name	config-property-type	config-property-value
rowSize	java.lang.Integer	値は，DABroker Library の setRowSize メソッドに渡されます。デフォルト値は 16 です。
serverName	java.lang.String	接続する DABroker Library のホスト名または IP アドレスを設定します。設定された値は，DABroker Library の setServerName メソッドに渡されます。
SQLWarningIgnore	java.lang.Boolean	データベースから返される警告を Connection クラスで保持するかどうかを設定します。設定された値は，DABroker Library の setSQLWarningIgnore メソッドに渡されます。 • true を指定した場合 警告を保持しません。 • false を指定した場合 警告を保持します。 デフォルト値は false です。
SV_EVENT_TRC	java.lang.Boolean	DABroker Library とのイベントトレースの取得の有無を設定します。設定された値は，DABroker Library の setSV_EVENT_TRC メソッドに渡されます。 • true を指定した場合 トレースを取得します。 • false を指定した場合 トレースを取得しません。 なお，true を指定した場合，Driver が出力するトレースは，Component Container からセットされる PrintWriter と，ユーザがコネクションファクトリ（javax.sql.DataSource クラス）の setLogWriter()メソッドで指定した PrintWriter に出力します。デフォルト値は false です。
TRC_NO	java.lang.Integer	トレースのエントリ数を，10～1000 の範囲で設定します。設定された値は，DABroker Library の setTRC_NO メソッドに渡されます。デフォルト値は 500 です。
uapName	java.lang.String	アプリケーション名称を設定します。設定された値は，DABroker Library の setUapName メソッドに渡されます。

表 4-94　DBConnector_DABJ_XA.rar または DBConnector_DABJ_XA_Cosminexus_RM.rar を使用する場合に指定できるプロパティ

config-property-name	config-property-type	config-property-value
blockUpdate	java.lang.Boolean	接続データベースが HiRDB の場合，？パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。設定された値は，DABroker Library の setBlockUpdate メソッドに渡されます。 • true を指定した場合 複数のパラメタセットを一度に処理します。 • false を指定した場合 複数のパラメタセットを一つずつ分割して処理します。 デフォルト値は false です。

config-property-name	config-property-type	config-property-value
bufferPoolSize	java.lang.Integer	受信データのバッファをプールする最大数を設定します。設定された値は，DABroker Library の setBufferPoolSize メソッドに渡されます。 デフォルト値は 0 です。
bufSize	java.lang.Integer	DABroker Library からの受信データのバッファ長を，1～16000 の範囲（単位：キロバイト）で設定します。設定された値は，DABroker Library の setBufSize メソッドに渡されます。デフォルト値は 64 です。
CallableStatementPoolSize	java.lang.Integer	コネクションプールに割り当てられるコネクションごとの CallableStatement のプール数を設定します。0～1024 の整数で指定します。デフォルト値は 10 です。また，PreparedStatementPoolSize と CallableStatementPoolSize の合計値は，1024 以下に設定してください。有効範囲外の値を指定した場合は，デフォルト値が設定されます。
CancelStatement	java.lang.Boolean	トランザクションタイムアウトやアプリケーション強制停止時に，Statement クラス，CallableStatement クラスおよび PreparedStatement クラスで実行中の SQL をキャンセルするかどうかを指定します。 • true を指定した場合 実行中の SQL をキャンセルします。 • false を指定した場合 実行中の SQL をキャンセルしません。 HiRDB に接続する場合は，false を指定してください。 Oracle に接続する場合で，データベースのサーバまたはクライアントのどちらかが Windows プラットフォームで，専用サーバ接続をする場合，false を設定してください。
ConnectionIDUpdate	java.lang.Boolean	コネクション ID を DataSource#getConnection メソッドごとに更新するかどうかを指定します。 • true を指定した場合 DataSource#getConnection メソッドのたびにコネクション ID を生成します。 • false を指定した場合 最初の DataSource#getConnection メソッドでコネクション ID を生成し，そのあとは更新しません。 デフォルト値は false です。 このプロパティは，接続するデータベースが Oracle の場合に指定できます。
databaseName	java.lang.String	接続するデータベースの種別を設定します。設定された値は，DABroker Library の setDatabaseName メソッドに渡されます。 次の値が指定できます。 • HIRDB HiRDB に接続する場合に指定します。 • ORACLE，ORACLE8I

config-property-name	config-property-type	config-property-value
databaseName	java.lang.String	Oracle に接続する場合に指定します。 DBConnector_DABJ_XA_Cosminexus_RM.rar の場合，デフォルト値は HIRDB です。
DBEnv	java.lang.String	DABroker Library の接続先データベース定義情報を設定します。設定された値は，DABroker Library の setDBEnv メソッドに渡されます。
DBHostName	java.lang.String	接続する HiRDB のホスト名を設定します。設定された値は，DABroker Library の setDBHostName メソッドに渡されます。
description	java.lang.String	接続するデータベースに必要な接続付加情報を設定します。設定された値は，DABroker Library の setDescription メソッドに渡されます。 • HIRDB の場合 HiRDB クライアントの環境変数グループ名を指定します。 • ORACLE の場合 SQL*Net の接続文字列を指定します。
encodLang	java.lang.String	エンコード文字形態を設定します。設定された値は，DABroker Library の setEncodLang メソッドに渡されます。
executeDirectMode	java.lang.Boolean	接続データベースが HiRDB の場合，Statement クラスを使用した INSERT， UPDATE， DELETE などのデータベースの更新で，HiRDB の EXECUTE DIRECT 機能を使用するかどうかを設定します。設定された値は，DABroker Library の setExecuteDirectMode メソッドに渡されます。 • true を指定した場合 Execute Direct 機能を使用します。 • false を指定した場合 Execute Direct 機能を使用しません。 デフォルト値は false です。
HiRDBCursorMode	java.lang.Boolean	HiRDB で検索時にカーソルが Commit，または Rollback にわたって有効かどうかを設定します。設定された値は，DABroker Library の setHiRDBCursorMode メソッドに渡されます。 • true を指定した場合 カーソルは保持されます。アプリケーションは続けて Fetch することができます。 • false を指定した場合 カーソルはクローズされますが，ステートメントは有効です。アプリケーションは，Prepare しないで，再度 Execute できます。 デフォルト値は false です。

config-property-name	config-property-type	config-property-value
JDBC_IF_TRC	java.lang.Boolean	JDBC インタフェースメソッドトレースの取得の有無を設定します。設定された値は，DABroker Library の setJDBC_IF_TRC メソッドに渡されます。 • true を指定した場合 トレースを取得します。 • false を指定した場合 トレースを取得しません。 なお，true を指定した場合，Driver が出力するトレースは，Component Container からセットされる PrintWriter と，ユーザがコネクションファクトリ（javax.sql.DataSource クラス）の setLogWriter()メソッドで指定した PrintWriter に出力します。デフォルト値は false です。
linkedResourceAdapterName	java.lang.String	連携する Reliable Messaging リソースアダプタの表示名を指定します。 DBConnector_DABJ_XA_Cosminexus_RM.rar を使用する場合に指定できます。
loginTimeout	java.lang.Integer	データベースへの接続試行中に，データソースが待機する最長時間（単位：秒）を設定します。デフォルト値は 0 です。
logLevel	java.lang.String	DB Connector が出力するログ・トレースのレベルを指定します。 次の値が指定できます。 • 0 または ERROR • 10 または WARNING • 20 または INFORMATION デフォルト値は，0 または ERROR です。
LONGVARBINARY_Access	java.lang.String	LONGVARBINARY（列属性 BLOB，列属性 BINARY）のデータベースアクセス方法を設定します。設定された値は，DABroker Library の setLONGVARBINARY_Access メソッドに渡されます。 • REAL • LOCATOR デフォルト値は REAL です。
networkProtocol	java.lang.String	DABroker Library との接続種別を設定します。設定された値は，DABroker Library の setNetworkProtocol メソッドに渡されます。 • "lib"
OSAuthorize	java.lang.Boolean	OS 認証機能を使用してデータベースに接続するかどうかを設定します。設定された値は，DABroker Library の setOSAuthorize メソッドに渡されます。 • true を指定した場合 OS 認証機能を使用します。 • false を指定した場合 OS 認証機能を使用しません。

config-property-name	config-property-type	config-property-value
OSAuthorize	java.lang.Boolean	デフォルト値は false です。
portNumber	java.lang.Integer	接続する DABroker Library のポート番号を設定します。設定された値は，DABroker Library の setPortNumber メソッドに渡されます。デフォルト値は 40179 です。
PreparedStatementPoolSize	java.lang.Integer	コネクションプールに割り当てられるコネクションごとの PreparedStatement のプール数を設定します。0～1024 の整数で指定します。デフォルト値は 10 です。また，PreparedStatementPoolSize と CallableStatementPoolSize の合計値は，1024 以下に設定してください。有効範囲外の値を指定した場合は，デフォルト値が設定されます。
RMID	java.lang.Integer	リソースマネジャの識別子を，1 以上の正の整数で設定します。設定された値は，DABroker Library の setRMID メソッドに渡されます。デフォルト値は 1 です。
rowSize	java.lang.Integer	DABroker Library が JDBC で取り扱うバッファ長を，16～512 の範囲（単位：メガバイト）で設定します。設定された値は，DABroker Library の setRowSize メソッドに渡されます。デフォルト値は 16 です。
serverName	java.lang.String	接続する DABroker Library のホスト名を設定します。設定された値は，DABroker Library の setServerName メソッドに渡されます。
SQLWarningIgnore	java.lang.Boolean	データベースから返される警告を Connection クラスで保持するかどうかを設定します。設定された値は，DABroker Library の setSQLWarningIgnore メソッドに渡されます。 • true を指定した場合 警告を保持しません。 • false を指定した場合 警告を保持します。 デフォルト値は false です。
SV_EVENT_TRC	java.lang.Boolean	DABroker Library とのイベントトレースの取得の有無を設定します。設定された値は，DABroker Library の setSV_EVENT_TRC メソッドに渡されます。 • true を指定した場合 トレースを取得します。 • false を指定した場合 トレースを取得しません。 なお，true を指定した場合，Driver が出力するトレースは，Component Container からセットされる PrintWriter と，ユーザがコネクションファクトリ（javax.sql.DataSource クラス）の setLogWriter()メソッドで指定した PrintWriter に出力します。デフォルト値は false です。
TRC_NO	java.lang.Integer	トレースのエントリ数を設定します。設定された値は，DABroker Library の setTRC_NO メソッドに渡されます。デフォルト値は 500 です。

config-property-name	config-property-type	config-property-value
uapName	java.lang.String	アプリケーション名称を設定します。設定された値は，DABroker Library の setUapName メソッドに渡されます。
XACloseString	java.lang.String	XA_CLOSE 文字列を設定します。設定された値は，DABroker Library の setXACloseString メソッドに渡されます。
XALocalCommitMode	java.lang.Boolean	XA 使用時，トランザクションが分散トランザクションでないとき，データベースのオートコミットモードを有効にするかどうかを設定します。設定された値は DABroker Library の setXALocalCommitMode メソッドに渡されます。 • true を指定した場合 データベースのオートコミットを有効にします。 • false を指定した場合 データベースのオートコミットを無効にします。 デフォルト値は true です。 なお，false を指定した場合，リソースの接続テスト(cjtestres コマンド)，コネクション障害検知機能が正常に動作しません。リソースの接続テスト(cjtestres コマンド)，コネクション障害検知機能を使用する場合，false は指定しないでください。 また，DBConnector_DABJ_XA_Cosminexus_RM.rar の場合，false は指定しないでください。
XAOpenString	java.lang.String	XA_OPEN 文字列を設定します。設定された値は，DABroker Library の setXAOpenString メソッドに渡されます。
XAThreadMode	java.lang.Boolean	XA 使用時のスレッドモードを設定します。設定された値は，DABroker Library の setXAThreadMode メソッドに渡されます。 • true を指定した場合 マルチスレッドモード • false を指定した場合 シングルスレッドモード デフォルト値は true です。

4.31.2　Connector 属性ファイルのテンプレートファイル

DB Connector のテンプレートファイルについて説明します。

ファイルの格納先

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥templates¥
- UNIX の場合
  /opt/Cosminexus/CC/admin/templates/

テンプレートファイル

Component Container が提供している DB Connector のテンプレートファイルを次に示します。

表 4-95　DB Connector のテンプレートファイル一覧

テンプレートファイル名	説明
DBConnector_DABJ_CP_cfg.xml	DABroker Library 用 DB Connector の Connector 属性ファイルのテンプレートファイル（ローカルトランザクション用）。
DBConnector_DABJ_XA_cfg.xml	DABroker Library 用 DB Connector の Connector 属性ファイルのテンプレートファイル（グローバルトランザクション用）。
DBConnector_DABJ_CP_Cosminexus_RM_cfg.xml	DABroker Library 用 DB Connector for Reliable Messaging の Connector 属性ファイルのテンプレートファイル（ローカルトランザクション用）。
DBConnector_DABJ_XA_Cosminexus_RM_cfg.xml	DABroker Library 用 DB Connector for Reliable Messaging の Connector 属性のテンプレートファイル（グローバルトランザクション用）。

4.32　DABroker Library を使用する場合の注意事項

ここでは，DABroker Library を使用する場合の注意事項を示します。

4.32.1　DABroker Library を使用する場合に使用できる J2EE コンポーネントおよび機能についての注意事項

ここでは，データベース接続で使用できる機能について説明します。

データベース接続で使用できる J2EE コンポーネントおよび機能を，データベースごとに次の表に示します。

表 4-96　使用できる J2EE コンポーネントおよび機能

項目		HiRDB	Oracle		XDM/RD E2
			Oracle9i	Oracle10g	
J2EE コンポーネント	Servlet/JSP	○	○	○	○
	Stateless Session Bean	○	○	○	○
	Stateful Session Bean	○	○	○	○
	Entity Bean（BMP）	○	○	○	×
	Entity Bean（CMP1.1）	○	○	×	×
	Entity Bean（CMP2.0）	○※1	○	×	×
	Message-driven Bean（onMessage メソッドからのデータベースアクセス）	○	○	○	○
使用できる機能	コネクションプーリング	○	○	○	○
	コネクションプールのウォーミングアップ	○	○	○	○
	コネクションプールの情報表示（cjlistpool コマンド）	○	○	○	○
	コネクションプールのクリア（cjclearpool コマンド）	○	○	○	○
	リソースへの接続テスト	○	○	○	○
	コネクションの障害検知	○※2	○	○	○
	ステートメントプーリング	○※3	○	○	○※3，※4
	ステートメントキャンセル	○※5	△※6	△※6	○※7
	ステートメント setQueryTimeout メソッド	×	×	×	×
	コネクション ID の PRF トレース出力	○	○	○	○※7

項目		HiRDB	Oracle		XDM/RD E2
			Oracle9i	Oracle10g	
使用できる機能	障害調査用 SQL の出力	○	○	○	○
	クラスタコネクションプール	×	×	×	×

（凡例）○：使用できる　　×：使用できない　　△：一部制限あり

注※1　CMP2.0 の機能である，CMR 機能は利用できません。

注※2　DABroker Library を使用して接続する場合，HiRDB のバージョンが 07-01 以前でグローバルトランザクション（XATransaction）のときは使用できません。

注※3　HiRDB のバージョンが 07-01 以前の場合，アプリケーションサーバが管理するトランザクションで使用するときだけ，ステートメントプーリング機能が使用できます。これ以外の場合はエラーが発生します。また，HiRDB 自動再接続機能とステートメントプーリング機能の併用はできません。

注※4　XDM/RD E2 11-02 以降のバージョンと DABroker Library 03-13 以降を使用する場合にだけステートメントプーリングを使用できます。

注※5　グローバルトランザクション（XATransaction）のときは使用できません。

注※6　データベースのサーバまたはクライアントが Windows で，接続形式が専用接続であるときは，使用できません。

注※7　ステートメントキャンセル，およびコネクション ID の PRF トレース出力は，DABroker 経由の場合，XDM/RD E2 11-02 以降で使用できます。

! 注意事項

HiRDB のバージョンが 07-01 以前で，トランザクションサポートレベルに XATransaction を使用する場合，トランザクション外でコネクションを使用してデータベースにアクセスするとコネクションが二つ使用され，HiRDB への接続が同時に利用しているユーザ数の 2 倍になります。HiRDB のシステム共通定義の pd_max_users オペランドに，同時に利用しているユーザ数の 2 倍の値を指定してください。

4.32.2　トランザクションリカバリを使用する場合の注意事項

Oracle でリカバリをする場合で，JDBC ドライバに DABroker Library を使用するときは，ユーザに次の権限が必要です。

- SYS.DBA_PENDING_TRANSACTIONS への SELECT 権限
- FORCE ANY TRANSACTION 権限

4.32.3　CMP フィールドとデータベースのマッピングの注意事項

CMP フィールドとデータベースのマッピングは，データベースの種類ごとに異なります。詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「2.4.2　CMP フィールドとデータベースのマッピング」を参照してください。ここでは，DABroker Library の場合の注意事項について説明します。

(1)　HiRDB の場合のマッピング

次の表に示すデータ型で扱えるデータサイズの上限は JDBC ドライバの上限に依存します。

表 4-97　CMP でのフィールドとデータベースのマッピング（HiRDB 使用時）

Java データ型	JDBC データ型	SQL データ型
byte[]	LONGVARBINARY	BLOB

Java データ型	JDBC データ型	SQL データ型
Serializable な型	LONGVARBINARY	BLOB

[HiRDB BLOB 最大値]は 2147483647 バイトです。ただし，DABroker Library を使用する場合，DABHIRDBA_BLOBBUFSIZE（HiRDB の BLOB 型データ受取バッファサイズ）指定値を超過したデータは切り捨てられます。

(2)　Oracle の場合のマッピング一覧

次の表に示すデータ型で扱えるデータサイズの上限は JDBC ドライバの上限に依存します。

表 4-98　CMP でのフィールドとデータベースのマッピング（Oracle 使用時）

Java データ型	JDBC データ型	SQL データ型
byte[]	LONGVARBINARY	LONG RAW
Serializable な型	LONGVARBINARY	LONG RAW

[Oracle LONG RAW 最大値]は，2 ギガバイトです。ただし，扱えるデータサイズの上限は JDBC ドライバの上限に依存します。DABroker Library を使用する場合：DABORADBA_LONGBUFSIZE（ORACLE の LONG 型データ受取バッファサイズ）指定値を超過したデータは切り捨てられます。

4.32.4　複数の構築済み実行環境を切り替える場合の注意事項

複数の構築済み実行環境を切り替える場合の DABroker Library のファイルの扱いについての注意事項を説明します。

(1)　退避環境作成時に退避対象となるファイル

退避環境作成時に退避対象となる DABroker Library の製品ファイルを次に示します。

- /opt/Cosminexus/DABJ 以下すべて
- /opt/DABroker/.HTC_2340.inf
- /opt/DABroker/.dabrokerenv
- <DABroker 運用ディレクトリ>※/bin 以下すべて
- <DABroker 運用ディレクトリ>※/lib 以下すべて
- <DABroker 運用ディレクトリ>※/msg 以下すべて
- <DABroker 運用ディレクトリ>※/obj 以下すべて
- <DABroker 運用ディレクトリ>※/sample 以下すべて
- <DABroker 運用ディレクトリ>※/conf 以下すべて
- <DABroker 運用ディレクトリ>※/cpp 以下すべて

注※

<DABroker 運用ディレクトリ>のデフォルトは，<DABroker Library のインストールディレクトリ>です。<DABroker 運用ディレクトリ>にデフォルト以外を指定している場合も退避対象となります。

DABroker Library 以外のファイルについては，「7.2.4(2) 退避環境作成時に退避対象となるファイル」を参照してください。

(2)　退避作業が不要なファイル

次の表に示すファイルは，退避対象外ですが，退避作業は必須ではありません。これらのファイルは，アプリケーションサーバの動作に影響はありません。必要に応じて退避してください。なお，退避作業が不要なファイルの場合，実行環境の回復時に退避したファイルを戻す作業は不要です。

表 4–99　退避作業が不要なファイル

項番	ファイルまたはディレクトリ名	説明	構成ソフトウェア
1	<DABroker 運用ディレクトリ>※/AGT	エージェント処理用ディレクトリ	DABroker Library
2	<DABroker 運用ディレクトリ>※/AGTGRP	グループ管理用ディレクトリ	DABroker Library
3	<DABroker 運用ディレクトリ>※/spool 以下すべて	トレースファイル，ログファイルの格納用ディレクトリ	DABroker Library

注※

<DABroker 運用ディレクトリ>のデフォルトは，<DABroker Library のインストールディレクトリ>です。
<DABroker 運用ディレクトリ>にデフォルト以外を指定している場合も退避対象外です。

DABroker Library 以外のファイルについては，「7.2.3(1) 退避対象外のファイルの退避」を参照してください。

4.32.5　UTF-8 を使用する場合の注意事項（Windows の場合）

Windows で言語モードとして UTF-8 を使用する場合は，環境設定ユティリティの［リモートアクセス設定］タブの［漢字コードセット］に「UTF-8」を設定してください。なお，UTF-8 で操作できるのは，Java を使用してデータベースアクセスする場合だけです。

4.32.6　SQL 実行時の注意事項（Windows の場合）

DABroker Library は JDBC2.1 の規格に準拠します。Windows の場合，次に示す SQL の実行では SQLException を発行します。

- executeQuery メソッドでの更新系 SQL の実行
- executeUpdate メソッドでの参照系 SQL の実行

4.32.7　HiRDB クライアント経由で XDM/RD E2 を使用する場合の注意事項

HiRDB クライアント経由で XDM/RD E2 を使用する場合の注意事項について説明します。

- HiRDB クライアント経由で XDM/RD E2 に接続する場合に，UTF-8 環境下の HiRDB クライアントでアクセスするときは，NCHAR および NVARCHAR 属性のデータを使用できません。
- HiRDB クライアント経由で XDM/RD E2 に接続して一覧情報取得処理※を実行する場合は，XDM/RD E2 で HiRDB 形式ディクショナリビュー表を作成しておく必要があります。HiRDB 形式ディクショナリビュー表については，マニュアル「VOS3　データマネジメントシステム　XDM E2 系　XDM/RD E2　使用の手引－運用編－」を参照してください。

  注※　DatabaseMetaData クラスに含まれる次のメソッドを実行すると，一覧情報取得処理が実行されます。

- getTables
- getColumns
- getProcedures
- getTablePrivileges
- getColumnPrivileges
- getProcedureColumns
- getIndexInfo
- getPrimaryKeys
- getTableTypes

4.32.8　Windows Server 2008 または Windows Vista 使用時の注意事項

Windows Server 2008 または Windows Vista の環境で DABroker Library を使用する場合，次の操作は管理者権限のあるユーザで実行してください。

- DABroker Library のインストールまたはアンインストール
- 環境設定ユティリティ
- 接続先データベース定義ユティリティ
- データベースへのアクセス

5 基本・開発機能の互換機能（そのほかの機能）

この章では，基本・開発機能の互換機能のうち，簡易 Web サーバ機能，EJB クライアントアプリケーションのログのサブディレクトリ専有モード，および EJB 2.1 と Servlet 2.4 でのアノテーションの利用について説明します。

5.1　この章の構成

この章では，基本・開発機能の互換機能のうち，次の機能について説明します。

- 簡易 Web サーバ機能
- EJB クライアントアプリケーションのログのサブディレクトリ専有モード
- EJB 2.1 と Servlet 2.4 でのアノテーションの利用

この章の構成を次の表に示します。

表 5-1　基本・開発機能の互換機能（そのほかの機能）と参照先

機能	参照先
簡易 Web サーバ機能	5.2
EJB クライアントアプリケーションのログのサブディレクトリ専有モード	5.3
EJB 2.1 と Servlet 2.4 でのアノテーションの利用	5.4

5.2　簡易 Web サーバ機能

Web コンテナを HTTP サーバとして，スタブなどのダウンロードに利用します。この機能は，常に利用するように設定されています。

この節では，簡易 Web サーバ機能について説明します。

この節の構成を次の表に示します。

表 5-2　この節の構成（簡易 Web サーバ機能）

分類	タイトル	参照先
解説	簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能）	5.2.1
	簡易 Web サーバへのアクセスの制御	5.2.2
設定	実行環境での設定	5.2.3
注意事項	サーブレットおよび JSP 実装時の注意事項	5.2.4

注　「実装」および「運用」について，この節での説明はありません。

! 注意事項

- 簡易 Web サーバは，本番運用時の Web サーバとして利用できません。
- 簡易 Web サーバでは負荷分散機のヘルスチェックは実施できません。
- 簡易 Web サーバでは，Web コンテナでエラーが発生します。エラーが発生した際のエラーステータスコードについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「付録 A エラーステータスコード」を参照してください。
- 簡易 Web サーバを使用する場合，HTTP セッションに関するオブジェクトおよび Web コンテナとリダイレクタ間の通信に使用するオブジェクトに対して明示管理ヒープ機能を使用できません。
- Web アプリケーション単位の同時実行数制御を実行している場合，Web コンテナ上で動作する Web アプリケーションごとに，同時にリクエストを処理するスレッド数を設定できます。Web アプリケーション単位の同時実行スレッド数制御は簡易 Web サーバに対しても有効になります。

5.2.1　簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能）

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

! 注意事項

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

- 簡易 Web サーバで利用する IP アドレスには，ループバックアドレスを指定することはできません。簡易 Web サーバで利用する IP アドレスにループバックアドレスを指定した場合は，ワイルドカードアドレスを使用してサーバが起動されます。

5.2.2　簡易 Web サーバへのアクセスの制御

簡易 Web サーバにアクセスできるホストを制限できます。簡易 Web サーバへのアクセスを許可されていないホストからのアクセスがあった場合，J2EE サーバはそのアクセスを拒否※します。デフォルトでは，ローカルホストからのアクセスだけを許可します。ホストを制限するには，アクセスを許可するホストの，ホスト名，または IP アドレスを設定します。このとき，ホスト名や IP アドレスの代わりにアスタリスク（*）を指定すると，全ホストからのアクセスを許可する設定になります。

注※

J2EE サーバがアクセスを拒否すると，Web ブラウザが自動で再接続をリトライすることがあります。再接続のリトライ回数は，Web ブラウザによって異なるため，1 度のアクセスで，アクセスが拒否されたことを示すメッセージが複数出力されることがあります。

5.2.3　実行環境での設定

実行環境での設定について説明します。

(1)　J2EE サーバの設定

簡易 Web サーバでの IP アドレス指定（バインド先アドレス設定機能）または簡易 Web サーバへのアクセスの制御を使用する場合，J2EE サーバの設定が必要です。

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

簡易構築定義ファイルでの簡易 Web サーバの機能の定義について次の表に示します。

表 5-3　簡易構築定義ファイルでの簡易 Web サーバの機能の定義

項目	指定するパラメタ	設定内容
Web サーバで使用するホストの固定	webserver.connector.http.bind_host	簡易 Web サーバの IP アドレスまたはホスト名を指定します。
管理用サーバ	webserver.connector.http.permitted.hosts	管理用サーバへのアクセスを許可するホストの IP アドレスまたはホスト名を指定します※。
	ejbserver.http.port	管理用サーバのポート番号を指定します。

注※　アクセスを許可するホストをホスト名で指定した場合，J2EE サーバ起動時にホスト名の名前が解決されるため，次のことに注意する必要があります。

- hosts ファイル，または DNS などでホスト名を解決できるようにする必要があります。解決できない場合はデフォルトの設定でサーバが起動されます。
- ホスト名の解決を J2EE サーバの起動時にするため，サーバの起動に時間が掛かることがあります。また，起動後に変更された IP アドレスは反映されないことがあります。

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

(2)　バッチサーバの設定

usrconf.properties（バッチサーバ用ユーザプロパティファイル）での指定内容を次に示します。ファイルの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「3.3 usrconf.properties（バッチサーバ用ユーザプロパティファイル)」を参照してください。

キー名称	概要	デフォルト値
webserver.connector.http.bind_host	簡易 Web サーバで使用するローカル IP アドレス，または解決できるローカルホスト名称を指定します。	なし
webserver.connector.http.permitted.hosts	簡易 Web サーバへのアクセスを許可するホストの IP アドレス（10 進表記）またはホスト名を指定します。	なし

(3)　運用管理ポータルについての設定

mserver.properties（Management Server 環境設定ファイル）に設定します。ファイルの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「10.7　mserver.properties（Management Server 環境設定ファイル)」を参照してください。

キー名称	内容	指定可能値	デフォルト	VR
com.cosminexus.mngsvr.compat.monitor_view	運用管理ポータルの［論理サーバの運用監視］画面に次の項目を表示させるかどうかを指定します。 • 管理用サーバ（簡易 Web サーバ）の情報 なお，この項目は互換用の項目です。 true を指定した場合： 表示項目を表示します。 false を指定した場合： 表示項目を表示しません。	次のどちらかを指定します。 • true • false	false	06-70

5.2.4　サーブレットおよび JSP 実装時の注意事項

(1)　javax.servlet.ServletRequest インタフェースの getServerName メソッドおよび getServerPort メソッドの戻り値について

アプリケーションサーバでは，getServerName メソッド，および getServerPort メソッドの戻り値は，HTTP リクエストと，使用する機能の組み合わせによって得られます。簡易 Web サーバとの組み合わせの場合の戻り値を次の表に示します。

表 5–4　getServerName メソッド，および getServerPort メソッドの戻り値（簡易 Web サーバとの組み合わせの場合）

HTTP リクエスト		getServerName メソッドの戻り値	getServerPort メソッドの戻り値
Host ヘッダの有無	リクエストラインの URI の種類		
あり	絶対 URI	Host ヘッダのホスト名	Host ヘッダのポート番号
	相対 URI	Host ヘッダのホスト名	Host ヘッダのポート番号
あり	絶対 URI	J2EE サーバのホスト名または IP アドレス※	簡易 Web サーバのポート番号
	相対 URI	J2EE サーバのホスト名または IP アドレス※	簡易 Web サーバのポート番号

注※　java.net.InetAddress.getLocalHost メソッド，または getHostName メソッドの戻り値となります。

これ以外の注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「6.2.1　サーブレットおよび JSP 実装時共通の注意事項」を参照してください。

(2)　サーブレットを実装するときの入出力ストリーム利用時の注意

ServletInputStream の available メソッドは，簡易 Web サーバ機能を使用している場合，常に 0 を返します。それ以外の場合，バッファの状態に依存した値を返します。

5.3　EJB クライアントアプリケーションのログのサブディレクトリ専有モード

この節では，EJB クライアントアプリケーションのログのサブディレクトリ専有モードについて説明します。

この節の構成を次の表に示します。

表 5-5　この節の構成（EJB クライアントアプリケーションのログのサブディレクトリ専有モード）

分類	タイトル	参照先
解説	サブディレクトリ専有モードとは	5.3.1
	ログ出力先のサブディレクトリ数の管理	5.3.2
	サブディレクトリ専有モードの場合のシステムログの出力先	5.3.3
設定	動作モードの指定	5.3.4
運用	サブディレクトリ数の管理で使用するコマンド（cjcldellog（システムログ出力先サブディレクトリの削除））	5.3.5
注意事項	サブディレクトリ専有モード使用時の注意事項	5.3.6

注　「実装」について，この節での説明はありません。

5.3.1　サブディレクトリ専有モードとは

EJB クライアントアプリケーションのシステムログは，EJB クライアントアプリケーションのプロセス単位で出力されます。システムログでは，プロセスごとにログ出力先のサブディレクトリを作成するか，複数のプロセスでログ出力先のサブディレクトリを共有することができます。プロセスごとにログ出力先のサブディレクトリを作成する動作モードのことを**サブディレクトリ専有モード**といいます。

サブディレクトリ専有モードは 06-50 よりも前のバージョンとの互換用に使用するモードであるため，EJB クライアントアプリケーションを新規作成する場合は，**サブディレクトリ共有モード**を使用することをお勧めします。

！ 注意事項

- cjclstartap コマンドで EJB クライアントアプリケーションを実行する場合は，サブディレクトリ共有モードを使用してください。
- EJB クライアントアプリケーションのユーザログ機能を使用する場合は，サブディレクトリ共有モードを使用してください。

サブディレクトリ専有モードの動作を次の表に示します。

表 5-6　サブディレクトリ専有モードの動作

項目	サブディレクトリ専有モード
複数のプロセスでのサブディレクトリ共有の可否	共有できません。
ログ管理ファイル作成	作成されません。
ejbserver.client.ejb.log キー※のデフォルト値	ejb

項目	サブディレクトリ専有モード
ejbserver.client.log.appid キー※のデフォルト値	EJB クライアントアプリケーション識別 ID
ejbserver.client.log.directorynum キー※の指定	ejbserver.client.log.appid キーを指定した場合は，無効です。
ejbserver.logger.channels.define.<チャネル名>.filenum キー※に指定できる面数	1～16
ejbserver.logger.channels.define.<チャネル名>.filesize キー※に指定できる容量（バイト）	4,096～2,147,483,647

注※　EJB クライアントアプリケーションの開始時に指定するシステムプロパティです。

5.3.2　ログ出力先のサブディレクトリ数の管理

サブディレクトリ専有モードを使用している場合は，ログ出力先のサブディレクトリ数を管理できます。

EJB クライアントアプリケーションのシステムログは, EJB クライアントアプリケーション単位に作成されるログ出力先ディレクトリ（ejbserver.client.log.directory キー，および ejbserver.client.ejb.log キーに指定したディレクトリ）下の，サブディレクトリに格納されます。EJB クライアントアプリケーションを複数起動すると, 起動したプロセス分のサブディレクトリが生成されます。このサブディレクトリが単調増加しないように管理できます。

サブディレクトリ数は, EJB クライアントアプリケーションのプロセス単位に作成されるサブディレクトリの数となります。ただし, EJB クライアントアプリケーション単位に作成されるログ出力先ディレクトリ下にユーザ作成のサブディレクトリがある場合, ユーザ作成のサブディレクトリの数もサブディレクトリとして数えられます。

デフォルトでは，ログ出力先のサブディレクトリ数を 8 個までとし，古いサブディレクトリから削除されるようになっています。サブディレクトリ数を変更する場合には，ejbserver.client.log.directorynum キーで変更してください。EJB クライアントアプリケーションのシステムログの取得については, マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.5　EJB クライアントアプリケーションのシステムログ」を参照してください。

5.3.3　サブディレクトリ専有モードの場合のシステムログの出力先

EJB クライアントアプリケーションのシステムログは, ejbserver.client.log.directory キーに指定したディレクトリ下の, EJB クライアントアプリケーションのプロセス単位に作成されるサブディレクトリ下に格納されます。EJB クライアントアプリケーションのプロセス単位のサブディレクトリは, ejbserver.client.log.directory キーに指定したディレクトリ下の, ejbserver.client.ejb.log キーに指定したディレクトリ下に作成されます。デフォルトでは,「<製品のインストールディレクトリ>¥CC¥client¥logs¥ejb」（Windows の場合），または「/opt/Cosminexus/CC/client/logs/ejb」（UNIX の場合）の下にサブディレクトリが作成されます。

EJB クライアントアプリケーションのシステムログ出力先のサブディレクトリ名を次に示します。

表 5-7　EJB クライアントアプリケーションのシステムログ出力先のサブディレクトリ名

コマンド名	サブディレクトリ名
vbj コマンド	cjcl<日時>_<プロセス ID>

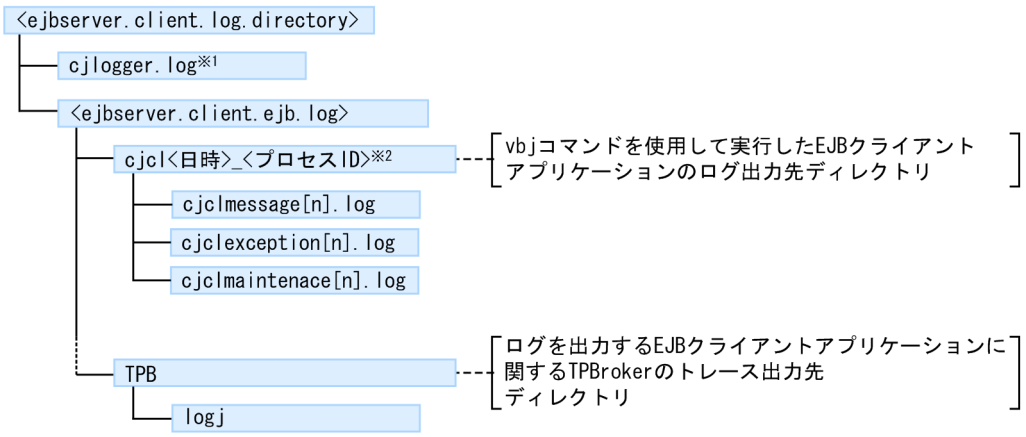
注

<日時>には，EJB クライアントアプリケーションを開始した日時が「mddhhmmssmmm」の形式で表示されます。日時に表示される項目は次のとおりです。なお，月以外は 10 進数で表示されます。

m：月（16 進数）　dd：日付　hh：時間　mm：分　ss：秒　mmm：ミリ秒

EJB クライアントアプリケーションのシステムログ出力先のディレクトリ構成例を次に示します。

図 5-1　サブディレクトリ専有モードの EJB クライアントアプリケーションのシステムログの出力先



注
　ファイル名の[n]は，1～（指定したログの面数）の通し番号になります。

注※1
　cjlogger.logは，EJBクライアントアプリケーションのプロパティの設定などに
　エラーがある場合だけ出力されます。

注※2
　EJBクライアントアプリケーションのシステムプロパティの
　ejbserver.client.log.appidキーにサブディレクトリ名を指定している場合，
　指定した名称でサブディレクトリが作成されます。

サブディレクトリは，次のように作成および運用されます。

- EJB クライアントアプリケーション単位に作成するサブディレクトリは，上限値を設定できます。EJB クライアントアプリケーションが終了すると，サブディレクトリに client_process_end という空のファイルが作成されます。上限値を超えた場合は，このファイルの作成時刻が古い順にサブディレクトリが削除されます。なお，サブディレクトリにログを出力している EJB クライアントアプリケーションがどれも終了していない（client_process_end ファイルが作成されていない）場合は，上限値を超えてサブディレクトリが作成されます。作成されたサブディレクトリがあとで上限値まで削除されることはありません。
- ejbserver.client.log.directory キーに指定したディレクトリ下の，ejbserver.client.ejb.log キーに指定したディレクトリ下にユーザ作成のサブディレクトリがある場合，EJB クライアントアプリケーション単位に作成するサブディレクトリと同様に，サブディレクトリとして数えられます。
- EJB クライアントアプリケーションが異常終了すると，client_process_end ファイルが作成されないことがあるため，異常終了した EJB クライアントアプリケーションのサブディレクトリは上限値を超えても削除されないことがあります。
- 一つの EJB クライアントアプリケーションを一つだけ起動する場合（同時実行しない場合）には，その EJB クライアントアプリケーション固有のシステムログ出力先ディレクトリを作成して，常にそのディ

レクトリに出力する運用ができます。このサブディレクトリは，上限値の指定に関係なく，削除されません。

- サブディレクトリを削除する場合，cjcldellog コマンドを使用すると，指定した更新日または更新時間より前に作成されたサブディレクトリを一括して削除できます。なお，cjcldellog コマンドは，Windows の場合は Windows のタスク，UNIX の場合は at コマンドまたは crontab コマンドを利用して定期的に実行してください。

EJB クライアントアプリケーションのログ出力の設定については，マニュアル「アプリケーションサーバ 機能解説 拡張編」の「9.10　EJB クライアントアプリケーションのユーザログ出力の設定（cjclstartap コマンドを使用する場合）」，およびマニュアル「アプリケーションサーバ 機能解説 拡張編」の「9.11　EJB クライアントアプリケーションのユーザログ出力の実装と設定（vbj コマンドを使用する場合）」を参照してください。

5.3.4　動作モードの指定

EJB クライアントアプリケーションでサブディレクトリ専有モードを使用するためには，vbj コマンド実行時のクラスパスに JAR ファイルを指定します。

指定する JAR ファイルの種類と配置場所を次に示します。なお，JAR ファイルには製品提供クラスが含まれます。

JAR ファイルの種類

- hntrlib2j64.jar
  HP-UX（IPF）の場合および Linux（IPF）の場合に指定します。
- hntrlib2j.jar
  上記以外の OS を使用している場合に指定します。

JAR ファイルの配置場所

- Windows の場合
  <Program Files>¥Hitachi¥HNTRLib2¥classes
- UNIX の場合
  /opt/hitachi/HNTRLib2/classes

なお，EJB クライアントアプリケーションから呼び出すアプリケーションの数，内容に関係なく，指定する JAR ファイルのファイル名と配置場所は固定です。

また，サブディレクトリ専有モードを使用する場合，次のキーを使用して，EJB クライアントアプリケーションのプロセス単位で作成されるログ出力先サブディレクトリ数を指定できます。

このキーは，Java アプリケーション用ユーザプロパティファイル（usrconf.properties）に指定できます。また，vbj コマンドでも指定できます（システムの実行環境に従って値を指定するか，または指定を省略できます）。

なお，このキーは，Java アプリケーション用 usrconf.properties ファイル固有のキーです。

キー名称	内容	デフォルト値
ejbserver.client.log.directorynum	Java アプリケーションのログ出力先のサブディレクトリ数を，1～16 の整数値で指定します。	なし

キー名称	内容	デフォルト値
ejbserver.client.log.directorynum	指定したサブディレクトリ数を超えた場合は，いちばん古い client_process_end ファイルがあるサブディレクトリが削除されます。なお，0 を指定した場合，サブディレクトリ数は無制限になり，サブディレクトリは削除されません。 ejbserver.client.log.directory に通常値のディレクトリを使用する場合は，このキーを指定しないでください。指定がない場合や 0～16 以外の値を指定した場合は，KDJE51002-W のメッセージが cjlogger.log ファイルに出力され，通常値が使用されます。 Java アプリケーションを同時に 17 個以上起動する場合は，このキーには 0 を指定してください。この場合，サブディレクトリの管理には cjcldellog コマンドを使用してください。 また，複数の Java アプリケーションを同時に起動する場合，KDJE51005-W のメッセージが頻繁に出力されるときには，サブディレクトリ数がこのキー値を超えていることがあります。この場合，このキーに 0 を指定して，cjcldellog コマンドを使用してサブディレクトリを管理してください。なお，このメッセージが出力される場合も，Java アプリケーションは問題なく動作します。 cjcldellog コマンドについては，「5.3.5　サブディレクトリ数の管理で使用するコマンド（cjcldellog（システムログ出力先サブディレクトリの削除））」を参照してください。 なお，次の場合，このキーの指定は無効になります。 • ejbserver.client.log.appid キーを指定した場合 • サブディレクトリ共有モードの場合	なし

usrconf.properties（Java アプリケーション用ユーザプロパティファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「14.3　usrconf.properties（Java アプリケーション用ユーザプロパティファイル)」を参照してください。

5.3.5　サブディレクトリ数の管理で使用するコマンド（cjcldellog（システムログ出力先サブディレクトリの削除））

サブディレクトリ数の管理には，次のコマンドを使用します。

表 5-8　サブディレクトリ数の管理で使用するコマンド

コマンド名称	分類	概要
cjcldellog	Java アプリケーションのシステムログ出力先サブディレクトリの削除	vbj コマンドで起動した Java アプリケーションのシステムログ出力先サブディレクトリを削除します。

コマンドの格納先

次のディレクトリに格納されています。

- Windows の場合

  <Application Server のインストールディレクトリ>¥CC¥client¥bin¥※
- UNIX の場合

  /opt/Cosminexus/CC/client/bin/

注※

Client を使用している場合は，格納ディレクトリの「<Application Server のインストールディレクトリ>¥CC¥」を，「<Application Server のインストールディレクトリ>¥CCL¥」と読み替えてください。

以降にコマンドの文法を示します。

● 形式

```
cjcldellog -t 〈時間>h|〈日数>d [-f] 〈ログ出力先ディレクトリ名称>
          [〈ログ出力先ディレクトリ名称〉 ...]
```

● 機能

引数に指定したログ出力先ディレクトリに，指定した日数または時間より前の更新日時のサブディレクトリがあった場合に，削除します。なお，-f を指定しない場合は，起動中の Java アプリケーションのサブディレクトリは削除しません。

なお，このコマンドは，vbj コマンドで起動した Java 環境で実行するコマンドです。

このコマンドは，タスク（Windows の場合），または at コマンドや crontab コマンド（UNIX の場合）を使用して定期的に実行してください。

● 引数

-t <時間>h

更新時間が何時間よりも前のサブディレクトリを削除するかを 1～8760 の範囲（単位：時間）で指定します。例えば，2 を指定した場合，更新時間が 2 時間前よりも古いサブディレクトリが削除されます。

-t <日数>d

更新日が何日よりも前のサブディレクトリを削除するかを 1～365 の範囲（単位：日）で指定します。例えば，2 を指定した場合，更新日が 2 日よりも古いサブディレクトリが削除されます。

-f

指定された日時より古いサブディレクトリすべてを削除対象とします。省略した場合は，client_process_end ファイルが存在するサブディレクトリだけが削除対象になります。

この引数を使用すると，Java アプリケーション起動中のサブディレクトリも削除対象としてしまうおそれがあるため，すべての Java アプリケーションが終了していることを確認してからコマンドを実行してください。

<ログ出力先ディレクトリ名称>

削除対象にするサブディレクトリの上位ディレクトリである，ログ出力先ディレクトリを，フルパス名で指定します。ログ出力先ディレクトリを複数指定するときは，空白で区切ってください。また，複数のログ出力先ディレクトリを指定した場合，最後のディレクトリだけにワイルドカード（*）を指定できます。

なお，Windows の場合，空白を含むパスを指定するときは，パス全体を""で囲む必要があります。

指定できるパスの例を次に示します。

- Windows の場合

```
"C:¥Program Files¥Hitachi¥Cosminexus¥CC¥client¥logs¥ejb"
C:¥work¥logs¥ejb*
C:¥work¥logs¥ejb*log
C:¥work¥logs¥*log
```

- UNIX の場合

```
/work/logs/ejb*
/work/logs/ejb*log
/work/logs/*log
```

● 戻り値

0：

正常終了しました。

1：

異常終了しました。

● 入力例

7 日より前の更新日時のサブディレクトリをすべて削除する場合

- Windows の場合

```
cjcldellog -t 7d -f c:¥work¥logs
```

- UNIX の場合

```
cjcldellog -t 7d -f /work/logs
```

12 時間より前の更新日時のサブディレクトリで client_process_end ファイルが存在するサブディレクトリを削除する場合

- Windows の場合

```
cjcldellog -t 12h c:¥work¥logs
```

- UNIX の場合

```
cjcldellog -t 12h /work/logs
```

複数のディレクトリ下にある 12 時間より前の更新日時のサブディレクトリで，client_process_end ファイルが存在するサブディレクトリを削除する場合

- Windows の場合

```
cjcldellog -t 12h c:¥work¥logs¥ejb1 c:¥work¥logs¥ejb2 c:¥work¥logs¥ejb3
```

- UNIX の場合

```
cjcldellog -t 12h /work/logs/ejb1 /work/logs/ejb2 /work/logs/ejb3
```

“ejb”で始まるディレクトリにある 12 時間より前の更新日時のサブディレクトリで，client_process_end ファイルが存在するサブディレクトリを削除する場合

- Windows の場合

```
cjcldellog -t 12h c:¥work¥logs¥ejb*
```

- UNIX の場合

```
cjcldellog -t 12h /work/logs/ejb*
```

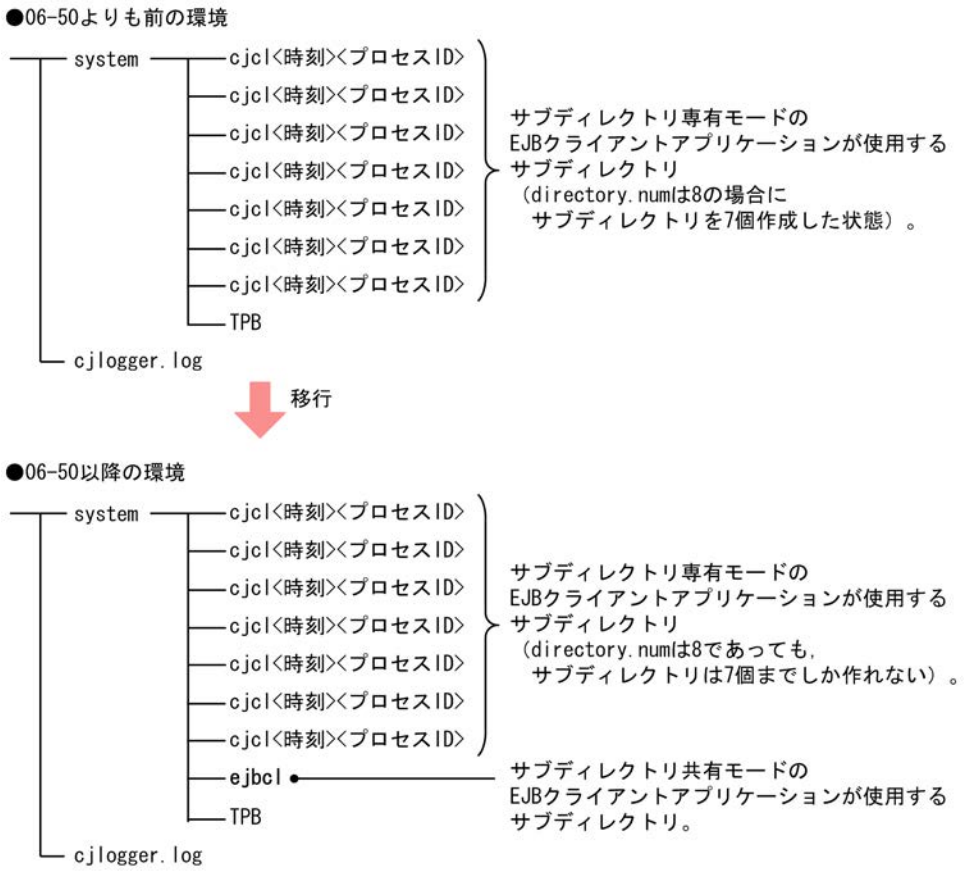
● 注意事項

- ログ出力サブディレクトリ配下にユーザがディレクトリを作成した場合，このコマンドではサブディレクトリ下のログファイルは削除しますが，ユーザ作成ディレクトリは削除しません。この場合，サブディレクトリとユーザ作成ディレクトリが削除されないで残ります。
- このコマンドでは作成後 1 時間以上経過していないログは削除できません。
- サブディレクトリ共有モードで動作しているアプリケーションが使用しているサブディレクトリを削除しないでください。削除した場合，以降のログ出力が正しく行われなくなることがあります。

5.3.6　サブディレクトリ専有モード使用時の注意事項

- EJB クライアントアプリケーションの同時起動の多重度が 17 多重以上の場合は，ejbserver.client.log.directorynum キーに 0 を指定してください。cjcldellog コマンドを使用してサブディレクトリ数を管理してください。
- EJB クライアントアプリケーションを複数同時に起動すると，KDJE51005-W のメッセージが出力されることがあります。EJB クライアントアプリケーションの動作には問題ありませんが，サブディレクトリ数が ejbserver.client.log.directorynum キーに指定した値を超えている場合があります。このメッセージが頻繁に出力される場合は，ejbserver.client.log.directorynum キーの値を 0 にし，cjcldellog コマンドを使用してサブディレクトリ数を管理してください。ログ出力先のサブディレクトリ数を管理できるのは，サブディレクトリ専有モードを使用している場合だけです。サブディレクトリ共有モードを使用している場合は，サブディレクトリ数の管理はできません。
- サブディレクトリ共有モードで動作している EJB クライアントアプリケーションがある環境では，サブディレクトリ専有モードで開始する EJB クライアントアプリケーションで，システムプロパティの ejbserver.client.ejb.log キーに"system"を指定しないでください。サブディレクトリ共有モードでは ejbserver.client.ejb.log キーのデフォルト値として"system"を使用するため，サブディレクトリ数が正しく管理できなくなります。
- すでにアプリケーションサーバ 06-50 よりも前のバージョンで構築した環境で，サブディレクトリ専有モードで動作させている EJB クライアントアプリケーションを，システムプロパティ ejbserver.client.ejb.log キーに"system"を指定した状態で 06-50 以降に移行した場合，サブディレクトリ専有モードの EJB クライアントアプリケーションとサブディレクトリ共有モードの EJB クライアントアプリケーションが混在することになります。この場合，次の図のような状態になります。

図 5-2　サブディレクトリ専有モードで動作させている EJB クライアントアプリケーションがある環境にサブディレクトリ共有モードで動作する EJB クライアントアプリケーションを混在させた場合のディレクトリ構成



このように，混在した環境では，サブディレクトリ専有モードの EJB クライアントアプリケーションが作成できるサブディレクトリの数が，サブディレクトリ共有モードの EJB クライアントアプリケーションが使用するディレクトリの数（1 個）分，少なくなります。なお，すでに directorynum で指定した数分のサブディレクトリが作成された状態で移行した場合は，サブディレクトリ専有モードで使用するサブディレクトリの数が減ることはありません。

- cjclstartap コマンドはサブディレクトリ共有モードだけで使用してください。サブディレクトリ専有モードで cjclstartap コマンドを実行して EJB クライアントを起動した場合，EJB クライアント起動コマンドの稼働時の情報（cjclstartap[n].log）が正しく表示されません。
- javalog[n].log はサブディレクトリ共有モードだけで使用できます。サブディレクトリ専有モードに設定した場合，ログは正しく出力されません。
- サブディレクトリ専有モードを使用している場合，Java アプリケーションに指定するシステムプロパティの次のキーについては，usrconf.properties（Java アプリケーション用ユーザプロパティファイル）のデフォルト値がサブディレクトリ共有モードの場合と一部異なります。異なるキーとデフォルト値を次に示します。

キー名称	デフォルト値
ejbserver.client.ejb.log	ejb
ejbserver.client.log.appid	<EJB クライアントアプリケーション識別 ID>※

注※　サブディレクトリ専有モードの通常値として使用される EJB クライアントアプリケーション識別 ID には，vbj コマンドの場合，Java アプリケーションの開始時刻とプロセス ID が使用されます。

- サブディレクトリ専有モードを使用している場合，Java アプリケーションに指定するシステムプロパティの次のキーについては，指定できる範囲がサブディレクトリ共有モードの場合と一部異なります。異なるキーと指定できる範囲を次に示します。

キー名称	指定できる範囲
ejbserver.logger.channels.define.<チャネル名>※.filenum	1～16 の整数値で指定します。
ejbserver.logger.channels.define.<チャネル名>※.filesize	4096～2147483647 の整数値で指定します。

注※

チャネル名称として次に示す名称を設定できます。

ClientMessageLogFile，ClientExceptionLogFile，ClientMaintenanceLogFile，EJBContainerLogFile，UserOutLogFile，UserErrLogFile

資料の取得については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「2.3　資料の取得」を参照してください。

- サブディレクトリ専有モードの場合，Java アプリケーションに指定するシステムプロパティとして次のプロパティを指定しても無視されます。
  - ejbserver.client.log.lockInterval
  - ejbserver.client.log.lockRetryCount

5.4　EJB 2.1 と Servlet 2.4 でのアノテーションの利用

アプリケーションサーバでは，EJB 2.1 と Servlet 2.4 でアプリケーションサーバ独自のアノテーションを利用できます。この節では，EJB 2.1 と Servlet 2.4 でのアノテーションの利用について説明します。

この節の構成を次の表に示します。

表 5–9　この節の構成（EJB 2.1 と Servlet 2.4 でのアノテーションの利用）

分類	タイトル	参照先
解説	ロード対象のクラスとロード時に必要なクラスパス	5.4.1
	EJB 2.1 と Servlet 2.4 でのアノテーション参照抑止機能	5.4.2
実装	javax.annotation パッケージに含まれるアノテーションのサポート範囲	5.4.3
	javax.ejb パッケージに含まれるアノテーションのサポート範囲	5.4.4
設定	実行環境での設定（J2EE サーバ単位の設定）	5.4.5
	アノテーション参照抑止機能の設定変更	5.4.6
注意事項	アノテーションの利用時の注意事項	5.4.7

注　「運用」について，この節での説明はありません。

5.4.1　ロード対象のクラスとロード時に必要なクラスパス

アノテーション情報は，次のような操作を実行したときに読み込まれます。

- リソースの追加（cjaddapp）※
- インポート時（cjimportapp）
- インポート時（cjimportres）
- インポート時（cjimportlibjar）※
- 開始時（cjstartapp）
- リプレース時（cjreplaceapp）
- リロード時（cjreloadapp）

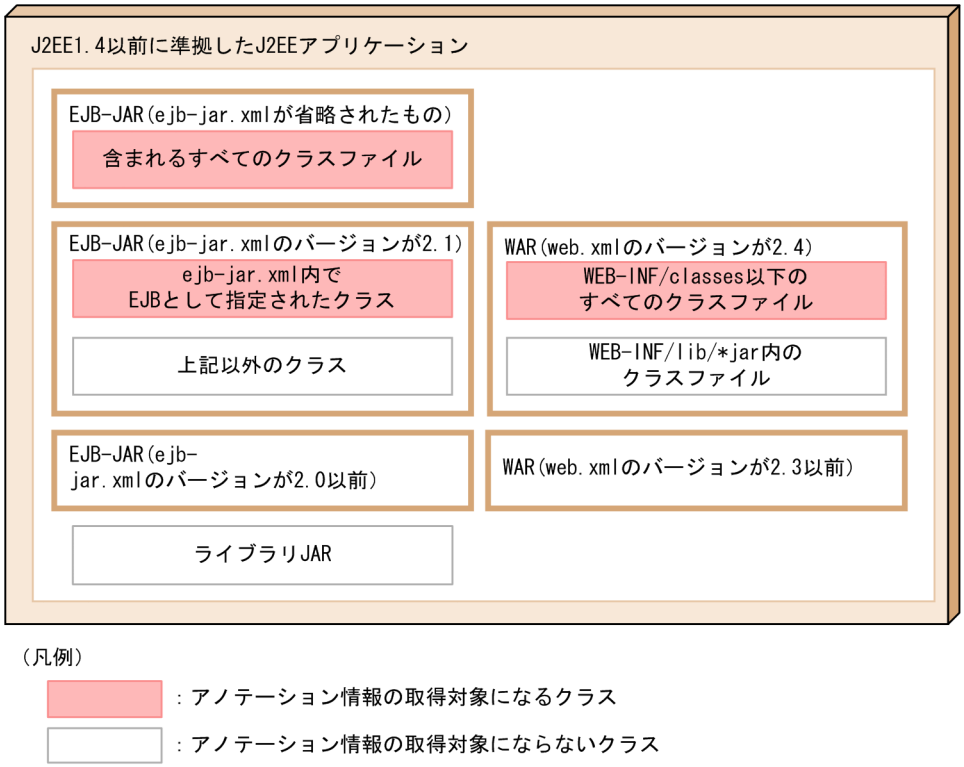
注※

対象となるのはライブラリ JAR のクラスです。

アノテーション情報を読み込むためには，まずクラスをロードする必要があります。

J2EE1.4 以前に準拠した J2EE アプリケーションでアノテーション情報を読み込むためにロードするクラスを次の図に示します。

図 5-3　アノテーション情報を読み込むためにロードするクラス



EJB-JAR（ejb-jar.xml のバージョンが 2.1）に含まれるクラスファイルまたは WAR（web.xml のバージョンが 2.4）に含まれるクラスファイルからアノテーション情報を読み込む場合のアノテーションは，アプリケーションサーバ独自の仕様です。また，アプリケーションサーバでは，アプリケーションサーバ独自の仕様によって，J2EE 1.4 の場合でも ejb-jar.xml の省略とアノテーションの使用ができます。

なお，J2EE 1.4 以前に準拠した J2EE アプリケーションの場合，EJB-JAR（ejb-jar.xml のバージョンが 2.1，または ejb-jar.xml が省略されたもの）や WAR（web.xml のバージョンが 2.4）からライブラリ JAR を使用することはできません。指定しても無視されます。

アノテーション情報を取得するためにクラスをロードするかどうかは，各モジュールのバージョン，および J2EE サーバで設定されているプロパティに依存します。

モジュールのバージョンが EJB-JAR（2.1）または WAR（2.4）の場合のアノテーション情報を取得するかどうかの条件を次の表に示します。

表 5-10　モジュールのバージョンが EJB-JAR（2.1）または WAR（2.4）の場合

モジュール		J2EE サーバのプロパティ ejbserver.deploy.applications.metadata_complete	
		false またはプロパティを省略時	true
EJB-JAR（2.1）		○	－
WAR（2.4）	WEB-INF/classes 以下	○	－
	WEB-INF/lib/JAR ファイル	－	－

（凡例）

○：アノテーション情報を取得する

－：アノテーション情報を取得しない

5.4.2　EJB 2.1 と Servlet 2.4 でのアノテーション参照抑止機能

アノテーション参照抑止機能は，J2EE アプリケーション中のアノテーションの参照（解析処理）を実施するかどうかを設定する機能です。アノテーション参照抑止機能を使用すると，アノテーションを使用しない場合に，アノテーションの解析処理を抑止することができます。

EJB 2.1 または Servlet 2.4 でのアノテーションの利用は，アプリケーションサーバで拡張された機能です。このため，アノテーションを利用している場合に，アノテーションの参照を抑止したいときは，アプリケーションサーバ独自のアノテーション参照抑止機能を使用します。

なお，EJB 2.1 または Servlet 2.4 でのアノテーションの参照抑止機能は，J2EE サーバ単位だけで設定できます。モジュール単位の参照抑止は設定できません。また，J2EE サーバ単位のアノテーションの参照抑止機能を有効にした場合，アノテーションを記述した EJB 2.1 または Servlet 2.4 のモジュールを含む J2EE アプリケーションは使用できなくなります。

5.4.3　javax.annotation パッケージに含まれるアノテーションのサポート範囲

javax.annotation パッケージのアノテーションの適用範囲を説明します。ここでは，WAR ファイル（Servlet 2.4 対応）および EJB-JAR ファイル（EJB2.1 対応）のコンポーネントごとに記述できるアノテーションを説明します。

(1)　EJB-JAR ファイル（EJB2.1 対応）

EJB-JAR ファイルに記述できるアノテーションの一覧を示します。

表 5–11　EJB-JAR ファイル（EJB2.1 対応）に記述できるアノテーション（javax.annotation パッケージ）

アノテーション名	Enterprise Bean				例外クラス	その他のクラス
	インタフェース	Session Bean	Entity Bean	Message-driven Bean		
@Resource※	－	○	－	×	－	－
@Resources※	－	○	－	×	－	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

(2)　WAR ファイル（Servlet 2.4 対応）

WAR ファイルに記述できるアノテーションの一覧を示します。

表 5-12　WAR ファイル（Servlet 2.4 対応）に記述できるアノテーション（javax.annotation パッケージ）

アノテーション名	Servlet 仕様			JSP 仕様				その他のクラス
	サーブレット	サーブレットフィルタ	イベントリスナ	JSP ファイル	タグハンドラ		タグライブラリイベントリスナ	
					クラシックタグハンドラ	シンプルタグハンドラ		
@Resource※1	○	○	○	－	○	○※2	×	－
@Resources※1	○	○	○	－	○	○	×	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※1

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

注※2

Servlet2.4 仕様の Web アプリケーションではメソッドおよびフィールドに対する DI ができません。

5.4.4　javax.ejb パッケージに含まれるアノテーションのサポート範囲

javax.ejb パッケージのアノテーションの適用範囲を説明します。ここでは，WAR ファイル（Servlet 2.4 対応）および EJB-JAR ファイル（EJB2.1 対応）のコンポーネントごとに記述できるアノテーションを説明します。

(1)　WAR ファイル（Servlet 2.4 対応）

WAR ファイルに記述できるアノテーションの一覧を示します。

表 5-13　WAR ファイル（Servlet 2.4 対応）に記述できるアノテーション（javax.ejb パッケージ）

アノテーション名	Servlet 仕様			JSP 仕様				その他のクラス
	サーブレット	サーブレットフィルタ	イベントリスナ	JSP ファイル	タグハンドラ		タグライブラリイベントリスナ	
					クラシックタグハンドラ	シンプルタグハンドラ		
@EJB※1	○	○	○	－	○	○※2	×	－
@EJBs※1	○	○	○	－	○	○	×	－

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

－：標準仕様で対応していない。

注※1

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

注※2

Servlet2.4 仕様の Web アプリケーションではメソッドおよびフィールドに対する DI ができません。

(2)　EJB-JAR ファイル（EJB2.1 対応）

EJB-JAR ファイルに記述できるアノテーションの一覧を示します。

表 5-14　EJB-JAR ファイル（EJB2.1 対応）に記述できるアノテーション（javax.ejb パッケージ）

アノテーション名	Enterprise Bean				例外クラス	その他のクラス
	インタフェース	Session Bean	Entity Bean	Message-driven Bean		
@EJB※	−	○	−	×	−	−
@EJBs※	−	○	−	×	−	−

（凡例）

○：対応する。

×：アプリケーションサーバでは対応しない。

−：標準仕様で対応していない。

注※

アプリケーションサーバでは，EJB2.1 仕様，または Servlet2.4 仕様のアプリケーションでもサポートしています。

5.4.5　実行環境での設定（J2EE サーバ単位の設定）

EJB 2.1 または Servlet 2.4 のモジュールに対するアノテーション参照抑止機能の定義は，J2EE サーバ単位に設定します。設定した内容は，その J2EE サーバにデプロイされているすべての J2EE アプリケーションに適用されます。

J2EE サーバ単位でアノテーションの参照を抑止する設定は，簡易構築定義ファイルを編集して行います。論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，次のパラメタで設定します。

キー名称	内容	省略値	導入されたバージョン
ejbserver.deploy.applications.metadata_complete	アノテーションを記述できるアプリケーションのうち，次のアプリケーションでアノテーション参照抑止機能を有効にするかどうかを指定します。 • EJB 2.1 • Servlet 2.4 true を指定した場合： アノテーションを参照しません。 false を指定した場合： アノテーションを参照します。	false	07-50

このパラメタは，usrconf.properties（J2EE サーバ用ユーザプロパティファイル）にも指定できます。

アノテーションの参照を抑止する設定をした場合，設定した J2EE サーバの起動時に，アノテーションの参照を抑止することを示すメッセージが出力されます。J2EE サーバ起動前または起動後にアノテーションの

参照を抑止しているかどうかの設定を確認する場合は，簡易構築定義ファイルの<configuration>タグ内のパラメタ「ejbserver.deploy.applications.metadata_complete」に設定された値を参照してください。

! 注意事項

J2EE サーバで ejbserver.deploy.applications.metadata_complete に true を指定した場合，アノテーションを記述した EJB 2.1 または Servlet 2.4 のモジュールを含む J2EE アプリケーションは使用できなくなります。

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

usrconf.properties（J2EE サーバ用ユーザプロパティファイル）については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.4　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。

5.4.6　アノテーション参照抑止機能の設定変更

アノテーション参照抑止機能の設定を変更する場合について説明します。

(1)　モジュール単位の設定（DD の metadata-complete 属性の値）を変更する場合

DD の metadata-complete 属性を変更する場合は，metadata-complete 属性を変更したアプリケーションを作成して，J2EE サーバに再度インポートする必要があります。

metadata-complete 属性を変更した場合，リデプロイ機能（-replaceDD オプションを指定した cjreplaceapp コマンド）によってアプリケーションを入れ替えようとすると，エラーになり，入れ替え処理は中断されます。

(2)　J2EE サーバ単位の設定（簡易構築定義ファイルの ejbserver.deploy.applications.metadata_complete の値）を変更する場合

アプリケーションインポート後に ejbserver.deploy.applications.metadata_complete の値を変更する場合について説明します。

true から false に変更した場合は，アプリケーションのアノテーションが参照されるようになります。特に対処は不要です。

false から true に変更した場合は，アノテーションが参照されなくなります。これに伴い，アプリケーションが正常に動作しなくなることがあります。アプリケーションが正常に動作しなくなった場合は，次の対処を実施してください。

アーカイブ形式のアプリケーションの場合

アノテーション参照抑止機能の設定を見直してください。または，J2EE アプリケーションの実装および DD の設定を見直して，アノテーションを参照しなくても正しく動作するように設定を変更してください。変更後，アプリケーションを再度インポートしてください。

展開ディレクトリ形式のアプリケーションの場合

アノテーション参照抑止機能の設定を見直してください。または，J2EE アプリケーションの実装および DD の設定を見直して，アノテーションを参照しなくても正しく動作するように設定を変更してください。変更後，アプリケーションを再開始してください。

5.4.7　アノテーションの利用時の注意事項

EJB 2.1，Servlet 2.4 のモジュールに定義されたアノテーションの情報は，DD に定義しても上書きはされません。

これらのモジュールに定義できるのは，アプリケーションサーバ独自のアノテーションです。DD によって上書きすることはできません。

6 拡張機能の互換機能（メモリセッションフェイルオーバ機能）

この章では，拡張機能の互換機能のうち，メモリセッションフェイルオーバ機能について説明します。

6.1　この章の構成

ここでは，メモリセッションフェイルオーバ機能について説明します。この機能を使用すると，アプリケーションで実行中のセッション情報が J2EE サーバとは別のプロセス（SFO サーバ）で管理され，J2EE サーバで障害が発生したときにそのセッション情報がほかの J2EE サーバに引き渡されます。これによって，リクエストがほかの J2EE サーバに転送された場合でも，障害発生前の状態で業務を続行できるようになります。

この章の構成を次の表に示します。

表 6-1　この章の構成（メモリセッションフェイルオーバ機能）

分類	タイトル	参照先
解説	メモリセッションフェイルオーバ機能の概要	6.2
	前提条件	6.3
	グローバルセッションを利用したセッション管理（メモリセッションフェイルオーバ機能）	6.4
	グローバルセッション情報のサイズの見積もり	6.5
	HTTP Cookie によるグローバルセッションの制御	6.6
	URL 書き換えによるグローバルセッションの制御	6.7
	グローバルセッションの制御方法が混在する場合のシステム構成	6.8
	メモリセッションフェイルオーバ機能で実施される処理	6.9
	メモリセッションフェイルオーバ機能使用時の障害検知	6.10
実装	アプリケーションの実装	6.11
	DD での定義	6.12
	cosminexus.xml での定義	6.13
設定	J2EE サーバの設定	6.14
	SFO サーバの設定	6.15
	Web アプリケーションの設定	6.16
	Web アプリケーションの設定情報の削除	6.17

注　「運用」および「注意事項」について，この章での説明はありません。

6.2　メモリセッションフェイルオーバ機能の概要

メモリセッションフェイルオーバ機能は，セッション情報を SFO サーバに冗長化し，システムの可用性を高めるための機能です。

ここでは，メモリセッションフェイルオーバ機能の処理の概要，および適用手順の概要を説明します。

6.2.1　処理の概要

メモリセッションフェイルオーバ機能を使用すると，リクエストによるセッションの生成処理が発生したときに，引き継ぎ対象に指定したセッション情報を SFO サーバに冗長化できます。そのため，J2EE サーバまたは Web サーバで障害が発生した場合も，ほかの J2EE サーバで SFO サーバに格納されているセッション情報を基にセッションを再生成し，正常に業務を続行できます。

なお，SFO サーバで障害が発生した場合は，すべてのセッション情報は削除されます。また，SFO サーバでの障害発生後にリクエストを受け付けた場合は，SFO サーバへのセッション情報の冗長化は実施されません。

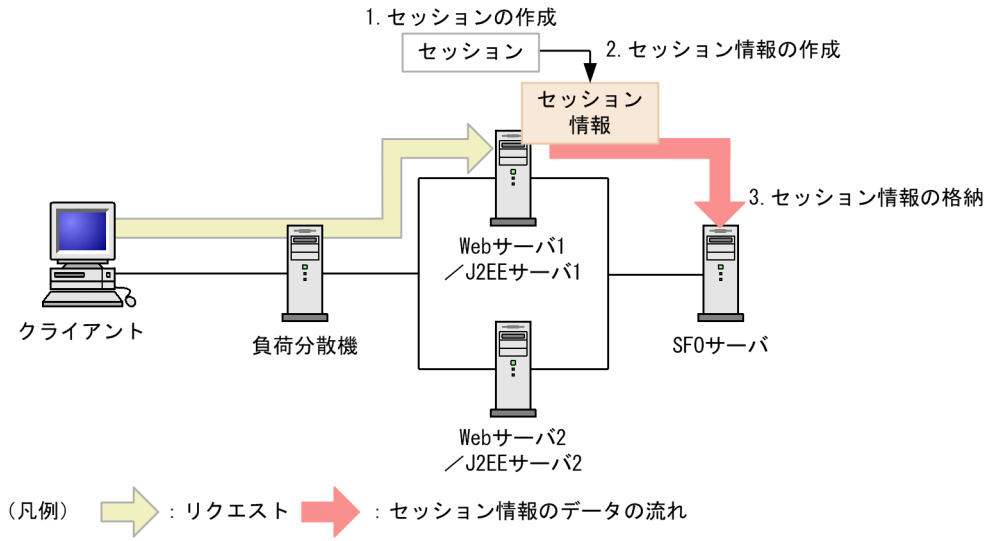
セッション情報の格納の流れ，および障害が発生した場合の処理について説明します。

(1)　セッション情報の格納の流れ

メモリセッションフェイルオーバ機能を使用すると，リクエストによるセッションの作成処理が発生したときに，処理の延長上でセッション情報が SFO サーバに格納されます。

セッション情報の格納の流れを次の図に示します。

図 6-1　セッション情報の格納の流れ



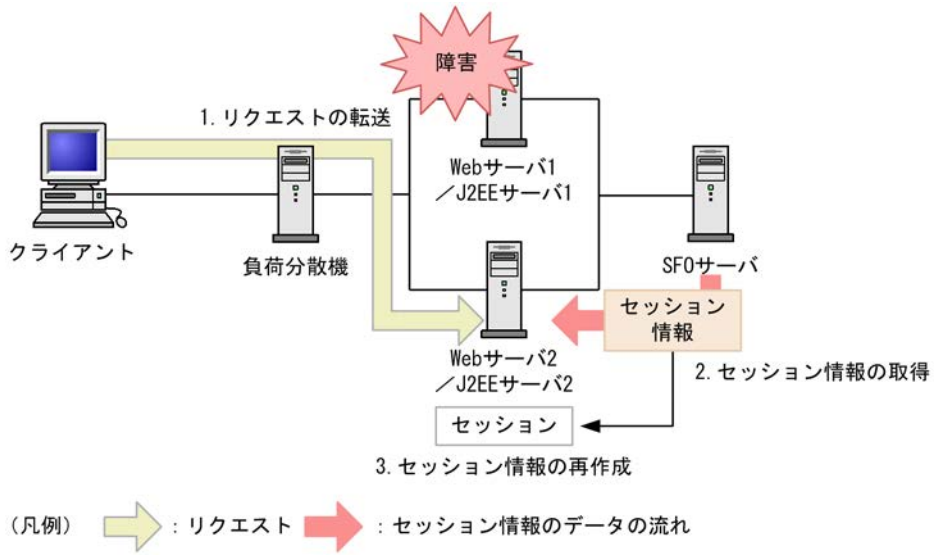
Web サーバがクライアントからセッションの作成が必要なリクエストを受け取ると，J2EE サーバでセッションが作成されます（1.）。セッションの作成処理の延長上で，セッション情報が作成され（2.），セッション情報が SFO サーバに格納されます（3.）。

(2)　Web サーバまたは J2EE サーバで障害が発生した場合の処理

Web サーバまたは J2EE サーバで障害が発生した場合，SFO サーバに格納されているセッション情報を基に，ほかの J2EE サーバでセッションを再作成し，正常に業務を続行できます。

Web サーバまたは J2EE サーバで障害が発生した場合の処理について，次の図に示します。

図 6-2　Web サーバまたは J2EE サーバで障害が発生した場合の処理



Web サーバ 1／J2EE サーバ 1 で障害が発生すると，負荷分散機によって Web サーバ 2 へリクエストが転送されます（1.）。転送先の J2EE サーバ 2 でリクエストを処理するとき，リクエストに関連づけられたセッションが存在しないため，SFO サーバからセッション情報を取得して（2.），セッションを再作成します（3.）。

セッションが正常に引き継がれ，障害発生前の状態で業務を続行できます。

(3)　データベースセッションフェイルオーバ機能とメモリセッションフェイルオーバ機能の機能差異

データベースセッションフェイルオーバ機能とメモリセッションフェイルオーバ機能では HTTP セッションの情報の格納先が異なります。そのため，二つの機能には次の表に示す機能差異があります。

表 6-2　データベースセッションフェイルオーバ機能とメモリセッションフェイルオーバ機能の機能差異

比較項目	機能差異	
	データベースセッションフェイルオーバ機能	メモリセッションフェイルオーバ機能
HTTP セッションの情報の格納先	データベース	SFO サーバ（メモリ）
J2EE サーバに障害が発生した際の動作	障害発生直前の状態から業務が再開できます。	障害発生直前の状態から業務が再開できます。
HTTP セッションの情報の格納先に障害が発生した際の動作	J2EE サーバ上のセッション情報を操作して業務を続行できます。	システム内のすべての HTTP セッションが削除され，業務は再開できません。
システムの可用性	データベースセッションフェイルオーバ機能が優れています。	

比較項目	機能差異	
	データベースセッションフェイルオーバ機能	メモリセッションフェイルオーバ機能
リクエスト処理性能	メモリセッションフェイルオーバ機能が優れています。	

メモリセッションフェイルオーバ機能とデータベースセッションフェイルオーバ機能は同時に使用できません。機能差異を踏まえて，システムに適した機能を選択してください。なお，メモリセッションフェイルオーバ機能とデータベースセッションフェイルオーバ機能を同時に有効にする設定をした場合，アプリケーションの開始に失敗します。

6.2.2　適用手順

メモリセッションフェイルオーバ機能を使用する場合に必要な，環境構築の準備と各種設定について説明します。

(1)　環境構築の準備

メモリセッションフェイルオーバ機能を使用する場合に環境構築の準備として実施する項目について，実施内容および参照先を次の表に示します。

表 6-3　メモリセッションフェイルオーバ機能を使用する環境構築の準備として実施する項目の実施内容および参照先

実施順序	実施項目	実施内容	参照先
1	前提条件の確認	前提となる構成および設定を確認します。	6.3
2	グローバルセッション情報のサイズの見積もり	グローバルセッション情報のサイズを見積もります。見積もった値は Web アプリケーションの設定(グローバルセッション情報の最大サイズの設定)で必要になります。	6.5

(2)　必要な設定

メモリセッションフェイルオーバ機能を使用したシステムを構築する場合，J2EE サーバ，SFO サーバ，および Web アプリケーションの設定が必要です。また，Web サーバ連携を使用し，かつ URL 書き換えを使用する場合，Web サーバの設定が必要です。

メモリセッションフェイルオーバ機能を使用する場合に必要な設定について，設定内容および参照先を次の表に示します。

表 6-4　メモリセッションフェイルオーバ機能使用時に必要な設定項目の設定内容および参照先

設定順序	設定項目	設定内容	参照先
1	Web アプリケーションの設定※	Web アプリケーションの設定は，DD（web.xml）および cosminexus.xml に定義します。 DD に定義する項目 • フィルタマッピング定義	6.12, 6.13

設定順序	設定項目	設定内容	参照先
1	Web アプリケーションの設定※	• グローバルセッション情報に関する設定 cosminexus.xml に定義する項目 フィルタ定義	6.12, 6.13
2	J2EE サーバの設定（必須）	次の設定をします。 • グローバルセッション情報を格納する SFO サーバの設定 • SFO サーバとの通信タイムアウトの設定 • リクエストラインの最大サイズの設定（URL 書き換えを使用し，かつインプロセス HTTP サーバを使用する場合）	6.7.5, 6.14
3	SFO サーバの設定（必須）	次の設定をします。 • SFO サーバの基本設定 • SFO サーバの起動オプションの設定	6.15
4	Web サーバの設定	リクエストラインの最大サイズの設定をします（Web サーバ連携を使用し，かつ URL 書き換えを使用する場合）。	6.7.5

注※　Web アプリケーションの設定は開発環境で実施します。実行環境で属性ファイルを使用した Web アプリケーションの設定をしたい場合は「6.16　Web アプリケーションの設定」を参照してください。

SFO サーバに登録した Web アプリケーションの設定情報を削除する方法については，「6.17　Web アプリケーションの設定情報の削除」を参照してください。

6.3　前提条件

メモリセッションフェイルオーバ機能を利用するための前提条件について説明します。

6.3.1　前提となる構成

メモリセッションフェイルオーバ機能を使用するには，負荷分散機の使用が前提となります。メモリセッションフェイルオーバ機能は，負荷分散機を使用した，複数の J2EE サーバにリクエストを振り分けるシステム構成の場合に使用できます。

メモリセッションフェイルオーバ機能を使用するためのシステム構成については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ システム設計ガイド」の「3.10.4　SFO サーバがシステムに複数存在する構成（メモリセッションフェイルオーバ機能）」
- マニュアル「アプリケーションサーバ システム設計ガイド」の「3.10.5　SFO サーバがシステムに一つだけ存在する構成（メモリセッションフェイルオーバ機能）」

6.3.2　前提となる設定

メモリセッションフェイルオーバ機能を使用するためには，次の設定が必要です。

- 前提機能の設定
  - HttpSession オブジェクト数の上限値の指定機能

    有効となる HttpSession オブジェクトの数に上限値を設定する機能です。なお，メモリセッションフェイルオーバ機能を使用する場合は，上限値に，1 以上の有効な値を設定する必要があります。HttpSession オブジェクト数の上限値の設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.5　HttpSession オブジェクト数の上限値の設定」を参照してください。

    なお，開発環境でグローバルセッション情報のサイズ見積もり機能を有効にする場合，HttpSession オブジェクト数の上限値の指定は不要です。グローバルセッション情報のサイズ見積もり機能については，「6.5　グローバルセッション情報のサイズの見積もり」を参照してください。
  - セッション ID へのサーバ ID の付加機能

    HttpSession のセッション ID に，Web コンテナごとに異なるサーバ ID を付加する機能です。セッション ID へのサーバ ID の付加については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.6　セッション ID および Cookie へのサーバ ID の付加」を参照してください。
- HTTP セッションのスティッキー（Sticky）の設定

  メモリセッションフェイルオーバ機能を使用するには，負荷分散機に HTTP セッションのスティッキーを設定する必要があります。

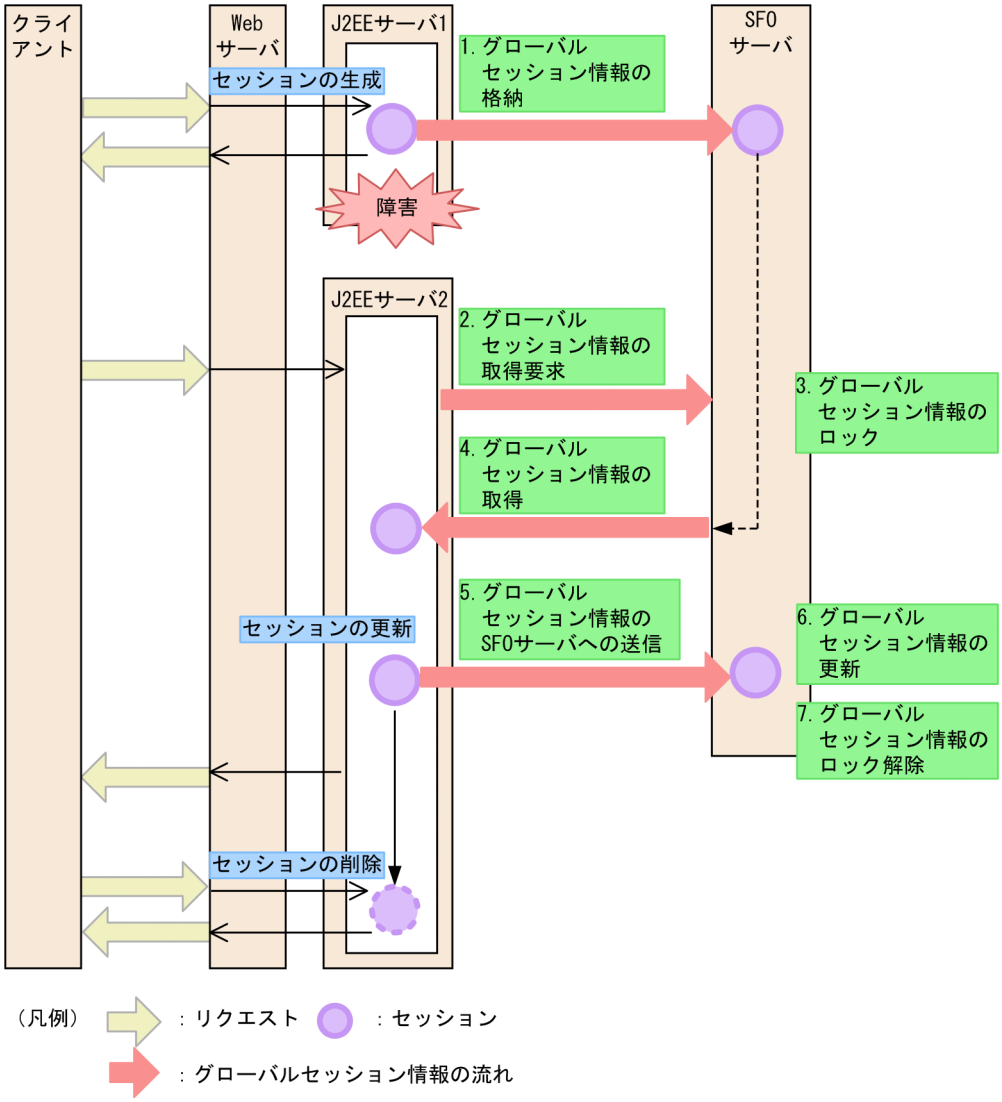
  HTTP セッションのスティッキーを設定しない場合，HTTP セッションを保持したリクエストの振り分け先が固定されません。そのため，HTTP セッションを保持したリクエストを受け付けるたびに HTTP セッションの引き継ぎ処理が実施され，性能が劣化するおそれがあります。

6.4　グローバルセッションを利用したセッション管理（メモリセッションフェイルオーバ機能）

メモリセッションフェイルオーバ機能を使用する場合，グローバルセッション情報は，SFO サーバで管理されます。SFO サーバとは，システム内のグローバルセッション情報を管理するための J2EE サーバです。

セッション情報の引き継ぎの流れについて次の図で説明します。

図 6-3　セッション情報引き継ぎの流れ（メモリセッションフェイルオーバ機能）



1. J2EE サーバ 1 で HTTP セッションの生成後，グローバルセッション情報は SFO サーバに格納されます。
2. J2EE サーバ 1 に障害が発生すると，処理を引き継いだ J2EE サーバ 2 では，SFO サーバに対して，グローバルセッション ID を基にグローバルセッション情報の取得を要求します。

   グローバルセッション ID とは，グローバルセッション情報を一意に識別するための ID です。
3. J2EE サーバ 2 では，セッションに対するリクエスト処理を開始する際に，グローバルセッション情報の同期をとるため，グローバルセッション情報に対してロックを掛けます。

4. SFO サーバは，J2EE サーバ 2 にグローバルセッション情報を渡します。
5. J2EE サーバ 2 でのリクエスト処理が完了すると，リクエスト処理で更新されたグローバルセッション情報は，SFO サーバに送信されます。
6. SFO サーバ側で，グローバルセッション情報が更新されます。
7. ロックが解除されます。

流れの中で出てきた項目について，以降の項で説明します。

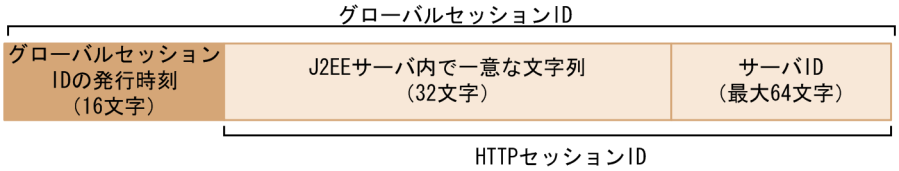
6.4.1　グローバルセッション ID

グローバルセッション ID とは，グローバルセッションを管理するための ID です。グローバルセッション ID によって，システム内でグローバルセッション情報を一意に識別できます。グローバルセッション ID は，HttpSession オブジェクトが作成されるごとに，HTTP セッション ID に関連づけて発行されます。J2EE サーバに障害が発生し，セッション情報が引き継がれるときにも，グローバルセッション ID が変わることはありません。

グローバルセッション ID は，グローバルセッション ID の発行時刻，および HTTP セッション ID で構成されています。

グローバルセッション ID の形式を次の図に示します。

図 6-4　グローバルセッション ID の形式



グローバルセッション ID の発行時刻は，GMT（グリニッジ標準時）の 1970 年 1 月 1 日 0 時からの通算ミリ秒数（8 バイトのデータを 16 進数で表現したもの）です。J2EE サーバで障害が発生したときに，障害発生前と障害発生後に作成されたグローバルセッション ID を必ず一意にするために付加されています。

HTTP セッション ID は，J2EE サーバ内で一意な文字列と，サーバ ID で構成されています。サーバ ID とは，セッション ID へのサーバ ID 付加機能で設定される ID です。セッション ID へのサーバ ID 付加機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.6 セッション ID および Cookie へのサーバ ID の付加」を参照してください。

6.4.2　グローバルセッション情報

メモリセッションフェイルオーバ機能を使用する場合，グローバルセッション情報は SFO サーバで管理されます。

SFO サーバでグローバルセッション情報として保持するセッション情報には，次の設定ができます。

- グローバルセッション情報として引き継ぐセッション情報の範囲
- グローバルセッション情報として指定できるセッション情報のデータ型
- グローバルセッション情報の種別（書き換え可能にするかまたは読み込み専用にするか）

これらの設定は，セッションフェイルオーバ用フィルタに設定します。設定については，「6.12　DD での定義」を参照してください。

次に，各設定内容について説明します。

(1)　グローバルセッション情報として引き継ぐセッション情報の範囲

SFO サーバでは，引き継ぎ対象として選択したセッション情報だけがグローバルセッション情報として保持されます。グローバルセッション情報として選択できるセッション情報は，HTTP セッションに登録された一部，またはすべてです。引き継ぎ対象として選択するセッション情報は，グローバルセッション情報の種別ごとに指定できます。グローバルセッション情報の種別については，「(4)　グローバルセッション情報の種別」を参照してください。

グローバルセッション情報選択時の注意事項

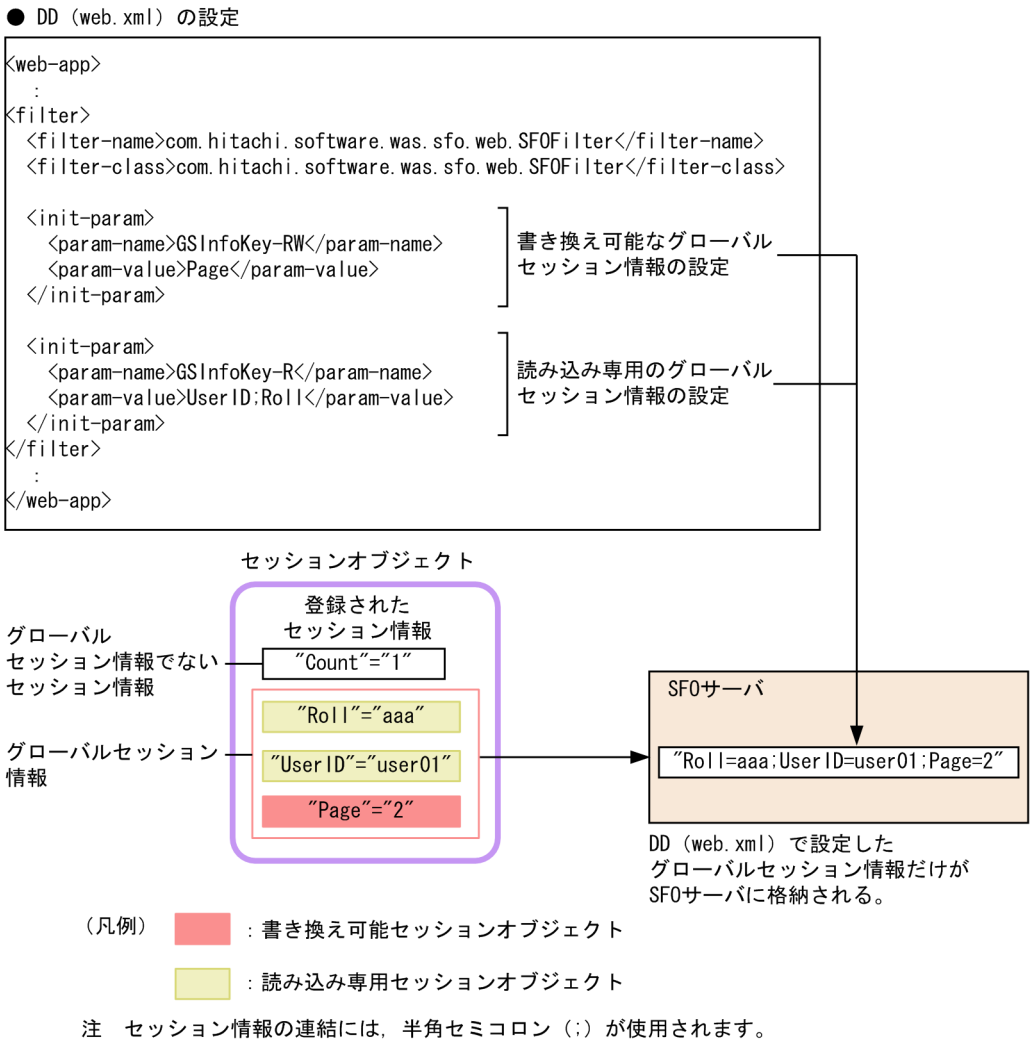
パフォーマンス向上のために，グローバルセッション情報には，必要最低限の HTTP セッション情報を指定してください。

(a)　HTTP セッションに登録された一部のセッション情報を引き継ぐ場合

HTTP セッションに登録されたセッション情報のうち，一部のセッション情報をグローバルセッション情報として引き継ぐ場合には，あらかじめ，DD（web.xml）で引き継ぎ対象とするセッション情報のキー名称を指定する必要があります。この場合，Web アプリケーション内で動的にキー名称を作成するセッション情報をグローバルセッションとして指定することはできません。

HTTP セッションに登録された一部のセッション情報を引き継ぐ例を次の図に示します。

図 6-5　HTTP セッションに登録された一部のセッション情報を引き継ぐ例



注　セッション情報の連結には，半角セミコロン（;）が使用されます。

(b)　HTTP セッションに登録されたすべてのセッション情報を引き継ぐ場合

HTTP セッションに登録されたセッション情報を，グローバルセッション情報としてすべて引き継ぐ場合には，あらかじめ，DD（web.xml）で引き継ぎ対象とするセッション情報のキー名称を設定する必要はありません。ただし，すべてのセッション情報を引き継ぐ設定をする必要があります。

! 注意事項

HTTP セッションに登録されたすべてのセッション情報をグローバルセッション情報として引き継ぐと，引き継ぐ必要がない情報が HTTP セッションに登録されている場合に，リクエスト処理のパフォーマンスが低下することがあります。このため，グローバルセッション情報には，必要な HTTP セッション情報だけ指定して引き継ぐことをお勧めします。

(2)　グローバルセッション情報として指定できるセッション情報のデータ型

(a)　データ型の種類

グローバルセッション情報として指定できるセッション情報には，次のデータ型のオブジェクトが指定できます。

- java.lang.String クラスのオブジェクト
- byte 型配列
- 直列化可能クラスのオブジェクト

java.lang.String クラス以外のデータ型のオブジェクトを引き継ぎ対象とする場合は，あらかじめ，DD（web.xml）でデータ型の引き継ぎを許可する設定が必要です。DD（web.xml）でデータ型の設定を省略した場合は，java.lang.String クラスのオブジェクトだけが引き継ぎ可能なセッション情報となります。

参考

既存の J2EE アプリケーションにメモリセッションフェイルオーバ機能を適用するときは，引き継ぎをしたいセッション情報のデータ型を確認してください。指定したデータ型以外のオブジェクトの扱いについては，「(3)　引き継ぎ対象外のオブジェクトの扱い」を参照してください。

(b)　直列化可能クラスのオブジェクトを引き継ぎ対象にする場合の注意事項

- 直列化可能クラスは，java.io.Serializable インタフェースを実装したクラスです。java.io.ObjectOutputStream クラスの writeObject()メソッドによってシリアライズできます。直列化可能クラスのオブジェクトを引き継ぎ対象にすると，グローバルセッション情報は，セッション情報を java.io.ObjectOutputStream クラスでシリアライズして作成されます。この場合，引き継ぐセッション情報のデータ型が，java.lang.String クラスのオブジェクト，または byte 型配列であっても，データ型に関係なく，直列化可能クラスのオブジェクトとしてシリアライズされます。

  セッション情報のシリアライズ処理では，属性ごとにシリアライズしたあとに，作成したグローバルセッション情報のサイズをチェックします。シリアライズ後の一時的なデータは，GSInfosLengthMax に指定したグローバルセッション情報の最大サイズに関係なく作成されます。このため，見積もった値を超える巨大なオブジェクトが HTTP セッションに登録されることがあります。この場合，グローバルセッション情報の作成中にメモリ不足（OutOfMemoryError エラー）が発生するおそれがあるので，注意してください。
- メモリセッションフェイルオーバ機能では，次に示す直列化可能クラスのオブジェクトを引き継ぎ対象としてサポートしています。
  - J2EE アプリケーションで提供されるクラスのオブジェクト
  - J2SE で提供されるクラスのオブジェクト

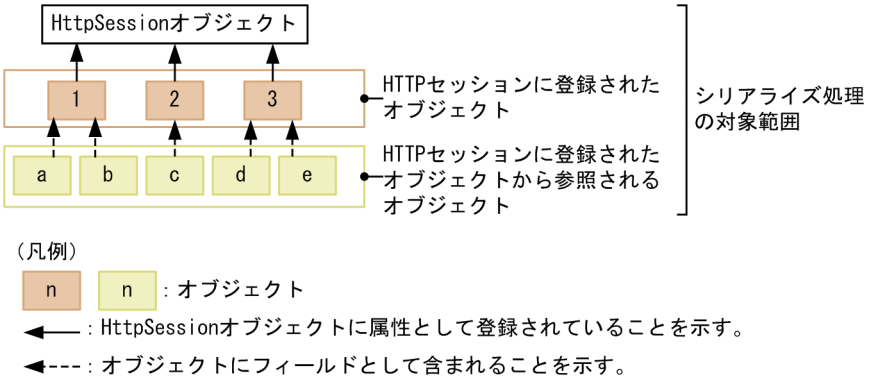
  ただし，引き継ぎ処理では，HTTP セッションに登録された直列化可能クラスのオブジェクトが，メモリセッションフェイルオーバ機能でサポートされているかオブジェクトかどうかはチェックされません。このため，メモリセッションフェイルオーバ機能でサポートしていないオブジェクトを HTTP セッションに登録している場合には，次のどちらかを実施して，このオブジェクトが引き継がれないようにしてください。
  - DD（web.xml）で引き継ぎ対象とするオブジェクトのキー名称を設定する。
  - オブジェクトが引き継がれないように J2EE アプリケーションを変更する。

(c)　データ型選択時の注意事項

セッション情報が直列化可能クラスのオブジェクトである場合，グローバルセッション情報の更新，および引き継ぎ処理で，オブジェクトのシリアライズまたはデシリアライズ処理が必要になります。シリアライズ処理は，引き継ぎ対象のオブジェクトだけでなく，このオブジェクトから参照されるオブジェクトすべてを対象とします。このため，引き継ぐ必要がない情報を含むクラスの場合，性能が低下することがあります。

シリアライズ処理の対象範囲の例を次の図に示します。

図 6-6　シリアライズ処理の対象範囲の例



この図では，HTTP セッションに登録されたオブジェクト（1～3）にフィールドとして含まれているオブジェクト（a～e）も，シリアライズ処理の対象に含まれます。

パフォーマンス向上のためには，グローバルセッション情報を java.lang.String クラスまたは byte 型配列のオブジェクトに限定することをお勧めします。

(3)　引き継ぎ対象外のオブジェクトの扱い

グローバルセッション情報として引き継ぎ対象外となるオブジェクトが HTTP セッションに登録された場合の扱いについて説明します。引き継ぎ対象とするセッション情報のデータ型および範囲の設定によって，オブジェクトの扱いが異なります。

また，直列化可能クラスのオブジェクトの場合，セッション情報の引き継ぎ時にシリアライズ処理またはデシリアライズ処理を実施します。これらの処理に成功するかどうかによってもオブジェクトの扱いが異なります。

(a)　java.lang.String クラスまたは byte 型配列のオブジェクトが引き継ぎ対象である場合

リクエスト処理完了時点で，HTTP セッションに java.lang.String クラスまたは byte 型配列以外のオブジェクトが登録されている場合，引き継ぎ対象とするセッション情報の範囲の設定によって，セッション情報を引き継ぐかどうかが異なります。HTTP セッションに引き継ぎ対象外のオブジェクトが登録されている場合のセッション情報の引き継ぎ可否を次の表に示します。

表 6-5　HTTP セッションに引き継ぎ対象以外のオブジェクトが登録されている場合のセッション情報の引き継ぎ可否

引き継ぎ対象とするセッション情報の範囲	リクエスト処理完了時点の動作	セッション情報の引き継ぎ可否	
		引き継ぎ対象外のオブジェクト	引き継ぎ対象のオブジェクト
一部のセッション情報※を引き継ぐ場合	KDJE34226-E のメッセージが出力され，エラーが通知されます。 HTTP セッションに登録されたすべての書き換え可能グローバルセッション情報は削除され，リクエストが処理されます。この場合，KDJE34219-W のメッセージが Warning レベルで出力されます。	セッション情報は引き継がれません。	エラーが通知される原因となった HTTP セッションに登録された，すべてのセッション情報が引き継がれません。

引き継ぎ対象とするセッション情報の範囲	リクエスト処理完了時点の動作	セッション情報の引き継ぎ可否	
		引き継ぎ対象外のオブジェクト	引き継ぎ対象のオブジェクト
すべてのセッション情報を引き継ぐ場合	KDJE34255-W のメッセージが Warning レベルで出力されます。	セッション情報は引き継がれません。	セッション情報は SFO サーバに冗長化されて，引き継がれます。

注※　DD（web.xml）で指定したキーで登録されたセッション情報です。

(b)　直列化可能クラスのオブジェクトが引き継ぎ対象である場合

リクエスト処理完了時点で，HTTP セッションに引き継ぎ対象外のオブジェクト登録されている場合，引き継ぎ対象とするセッション情報の設定範囲によって，セッション情報を引き継ぐかどうかが異なります。

また，直列化可能クラスのオブジェクトの場合，セッション情報の引き継ぎ時にシリアライズ処理またはデシリアライズ処理を実施します。これらの処理に成功するかどうかによってもオブジェクトの扱いが異なります。

- HTTP セッションに登録された一部のセッション情報を引き継ぐ場合

  DD（web.xml）で指定したキーで登録されたセッション情報は，java.io.Serializable インタフェースを実装した直列化可能クラスでなければなりません。リクエスト処理完了時点で，java.io.Serializable インタフェースを実装していないクラスのオブジェクトや，直列化可能クラス以外のオブジェクトが登録されていた場合にはエラーとなります。この場合，HTTP セッションに登録されたすべてのセッション情報が引き継がれません。

  HTTP セッションに登録されたオブジェクトによるセッション情報の引き継ぎ可否を次の表に示します。

表 6-6　HTTP セッションに登録されたオブジェクトの内容によるセッション情報の引き継ぎ可否（一部のセッション情報を引き継ぐ場合）

項番	HTTP セッションに登録されたオブジェクトの内容		リクエスト処理完了時点の動作	セッション情報の引き継ぎ可否	
	java.io.Serializable インタフェースの実装の有無	シリアライズの成功/失敗		引き継ぎ対象外のオブジェクト	引き継ぎ対象のオブジェクト
1	java.io.Serializable インタフェースの実装あり	シリアライズの成功	リクエストは処理されます。	－（該当しません）	セッション情報は SFO サーバに冗長化されて，引き継がれます。
2		シリアライズの失敗	KDJE34256-E のメッセージが出力され，エラーが通知されます。 HttpSession オブジェクトに登録されたすべての書き換え可能グローバルセッション情報は削除され，リクエストが処理されます。この場合，KDJE34219-W のメッセージが Warning レベルで出力されます。	セッション情報は引き継がれません。	エラーが通知される原因となった HTTP セッションに登録された，すべてのセッション情報は引き継がれません。

項番	HTTP セッションに登録されたオブジェクトの内容		リクエスト処理完了時点の動作	セッション情報の引き継ぎ可否	
	java.io.Serializable インタフェースの実装の有無	シリアライズの成功/失敗		引き継ぎ対象外のオブジェクト	引き継ぎ対象のオブジェクト
3	java.io.Serializable インタフェースの実装なし	－（シリアライズできません）	KDJE34226-E のメッセージが出力され，エラーが通知されます。 HTTP セッションに登録されたすべての書き換え可能グローバルセッション情報は削除され，リクエストが処理されます。この場合，KDJE34219-W のメッセージが Warning レベルで出力されます。	セッション情報は引き継がれません。	エラーが通知される原因となった HTTP セッションに登録された，すべてのセッション情報は引き継がれません。

- HTTP セッションに登録されたすべてのセッション情報を引き継ぐ場合

  リクエスト処理完了時点で，HTTP セッションに java.io.Serializable インタフェースを実装していないクラスのオブジェクトや，直列化可能クラス以外のオブジェクトが登録されていた場合，これらのオブジェクトは無視されて引き継がれません。

  HTTP セッションに登録されたオブジェクトによるセッション情報の引き継ぎ可否を次の表に示します。

表 6–7　HTTP セッションに登録されたオブジェクトの内容によるセッション情報の引き継ぎ可否（すべてのセッション情報を引き継ぐ場合）

項番	HTTP セッションに登録されたオブジェクトの内容		リクエスト処理完了時点の動作	セッション情報の引き継ぎ可否	
	java.io.Serializable インタフェースの実装の有無	シリアライズの成功/失敗		引き継ぎ対象外のオブジェクト	引き継ぎ対象のオブジェクト
1	java.io.Serializable インタフェースの実装あり	シリアライズの成功	リクエストは処理されます。	－（該当しません）	セッション情報は SFO サーバに冗長化されて，引き継がれます。
2		シリアライズの失敗	KDJE34256-E のメッセージが出力され，エラーが通知されます。 HttpSession オブジェクトに登録されたすべての書き換え可能グローバルセッション情報は削除され，リクエストが処理されます。この場合，KDJE34219-W のメッセージが Warning レベルで出力されます。	セッション情報は引き継がれません。	エラーが通知される原因となった HTTP セッションに登録された，すべてのセッション情報は引き継がれません。
3	java.io.Serializable インタフェースの実装なし	－（シリアライズできません）	KDJE34255-W のメッセージが出力されます。	セッション情報は無視され，引き継がれません。	セッション情報は SFO サーバに冗長化されて，引き継がれます。

- SFO サーバから J2EE サーバへグローバルセッション情報を引き継ぐ場合

J2EE サーバ間でセッション情報を引き継ぐ場合，J2EE サーバは，SFO サーバからグローバルセッション情報を取得します。この時，グローバルセッション情報のデシリアライズ処理を実施します。デシリアライズ処理によるグローバルセッション情報の引き継ぎ可否を次の表に示します。

表 6-8　デシリアライズ処理によるグローバルセッション情報の引き継ぎ可否

項番	デシリアライズの成功/失敗	引き継ぎ先の J2EE サーバの処理	セッション情報の引き継ぎ可否
1	デシリアライズの成功	SFO サーバから引き継いだグローバルセッション情報を使用してリクエストは処理されます。	SFO サーバから J2EE サーバへグローバルセッション情報が引き継がれます。
2	デシリアライズの失敗	グローバルセッション情報およびセッション情報を削除され，KDJE34258-E および KDJE34214-E のメッセージが出力されます。 J2EE サーバ間のセッション情報の引き継ぎに失敗するため，リクエストは HTTP セッションがない状態で処理されます。	SFO サーバから J2EE サーバへグローバルセッション情報は引き継がれません。

シリアライズおよびデシリアライズに失敗する原因

シリアライズおよびデシリアライズに失敗する原因について，次に示します。

シリアライズに失敗する原因

java.io.Serializable インタフェースを実装したクラスのオブジェクトがシリアライズに失敗する原因として，次の 2 点が考えられます。

- HTTP セッションに登録された直列化可能クラスのオブジェクトから参照されたオブジェクトが，直列化可能クラス以外のクラスであった。
- writeObject（java.io.ObjectOutputStream out）メソッドが実装されていて，かつ writeObject（java.io.ObjectOutputStream out）メソッドでシリアライズ時に例外が発生した。

デシリアライズに失敗する原因

java.io.Serializable インタフェースを実装したクラスのオブジェクトがデシリアライズに失敗する原因として，次の 2 点が考えられます。

- アプリケーションが，シリアライズ処理時とデシリアライズ処理時で異なっていて，デシリアライズ処理が失敗する原因となる変更がアプリケーションに加えられている。
- readObject（java.io.ObjectOutputStream in）メソッドが実装されていて，かつ readObject（java.io.ObjectOutputStream in）メソッドでデシリアライズ時に例外が発生した。

(4)　グローバルセッション情報の種別

保持するセッション情報には，グローバルセッション情報の種別を設定します。グローバルセッション情報の種別には，次の 2 種類があります。

- 書き換え可能グローバルセッション情報

  書き換え可能グローバルセッション情報とは，HttpSession オブジェクトに登録されたあと，変更が必要となるセッション情報です。書き換え可能グローバルセッション情報は，SFO サーバに障害が発生した場合，破棄されます。

HttpSession オブジェクトへの登録後，更新や削除をする必要があり，リクエスト処理の推移とともに，更新される可能性のあるセッション情報は，書き換え可能グローバルセッション情報に設定してください。

- **読み込み専用グローバルセッション情報**

  読み込み専用グローバルセッション情報とは，HttpSession オブジェクトに登録されたあと，変更されないセッション情報です。読み込み専用グローバルセッション情報は，SFO サーバに障害が発生しても，そのまま使用できます。

  HttpSession オブジェクトに登録した情報に対して，更新や削除をする必要のないセッション情報は，読み込み専用グローバルセッション情報に設定してください。例えば，ログイン情報など，一度登録すると更新する必要のない情報を読み込み専用グローバルセッション情報に設定すると，SFO サーバに障害が発生した場合でも，ユーザに対して再ログインを要求する必要がなくなるので，ユーザへの影響を軽減できます。

(5)　SFO サーバでのグローバルセッション情報の管理方法

SFO サーバでは，HttpSession オブジェクトに含まれるグローバルセッション情報は連結されて管理されます。直列化可能クラスのオブジェクトが引き継ぎ対象でない場合，セッション情報の連結には，区切り文字として半角セミコロン（;）が使用されます。直列化可能クラスのオブジェクトが引き継ぎ対象である場合は，連結するセッション情報がシリアライズ後の情報であるため，セッション情報の連結に区切り文字は使用されません。

なお，連結されたグローバルセッション情報のサイズには上限があります。上限を超えると，エラーになります。また，上限を超えたサイズのグローバルセッション情報は，SFO サーバでは管理されないので注意してください。グローバルセッション情報の最大サイズは，Web アプリケーションの DD（web.xml）で指定します。メモリセッションフェイルオーバ機能を使用する場合に必要な定義については，「6.12.1 DD の作成」を参照してください。また，開発環境でグローバルセッション情報のサイズを見積もる場合には，グローバルセッション情報のサイズ見積もり機能が使用できます。グローバルセッション情報のサイズ見積もり機能については，「6.5　グローバルセッション情報のサイズの見積もり」を参照してください。

また，デプロイ後の Web アプリケーションの場合，Web アプリケーションを含む J2EE アプリケーションの属性（プロパティ）として設定できます。デプロイ後の Web アプリケーションのセッションフェイルオーバ用フィルタの設定については，「6.12　DD での定義」を参照してください。

6.4.3　グローバルセッションを利用するためのシステム構成要素

ここでは，グローバルセッションを利用するためのシステム構成要素である，次のサーバプロセス，アプリケーションおよびフィルタについて説明します。

- SFO サーバ
- SFO サーバアプリケーション
- セッションフェイルオーバ用フィルタ

SFO サーバアプリケーションは，SFO サーバ上で動作するアプリケーションです。セッションフェイルオーバ用フィルタは，J2EE サーバで動作する J2EE アプリケーションに設定するサーブレットフィルタです。

(1)　SFO サーバ

SFO サーバとは，グローバルセッション情報を管理するサーバプロセスです。メモリセッションフェイルオーバ機能を使用するためには，システム内に SFO サーバを配置する必要があります。

● SFO サーバの役割

SFO サーバは，J2EE サーバの J2EE アプリケーション内で生成されたグローバルセッション情報を管理する機能を持つサーバプロセスです。実体は J2EE サーバです。

J2EE アプリケーションが稼働している J2EE サーバに障害が発生して，J2EE サーバ間でグローバルセッション情報の引き継ぎが必要になると，引き継がれる側の J2EE サーバは，SFO サーバに対してグローバルセッション情報の取得要求をします。

SFO サーバは，SFO サーバアプリケーションだけを実行するための J2EE サーバです。SFO サーバアプリケーション以外のアプリケーションを実行しないでください。

また，SFO サーバでは，SFO サーバアプリケーションを登録するために，CORBA ネーミングサービスを使用します。CORBA ネーミングサービスは SFO サーバのインプロセスで動作させてください。

● SFO サーバの複数配置

SFO サーバに障害が発生すると，SFO サーバ上で管理されているグローバルセッション情報はすべて削除されます※。SFO サーバの障害発生に備えて，システム内に SFO サーバを複数配置することもできます。

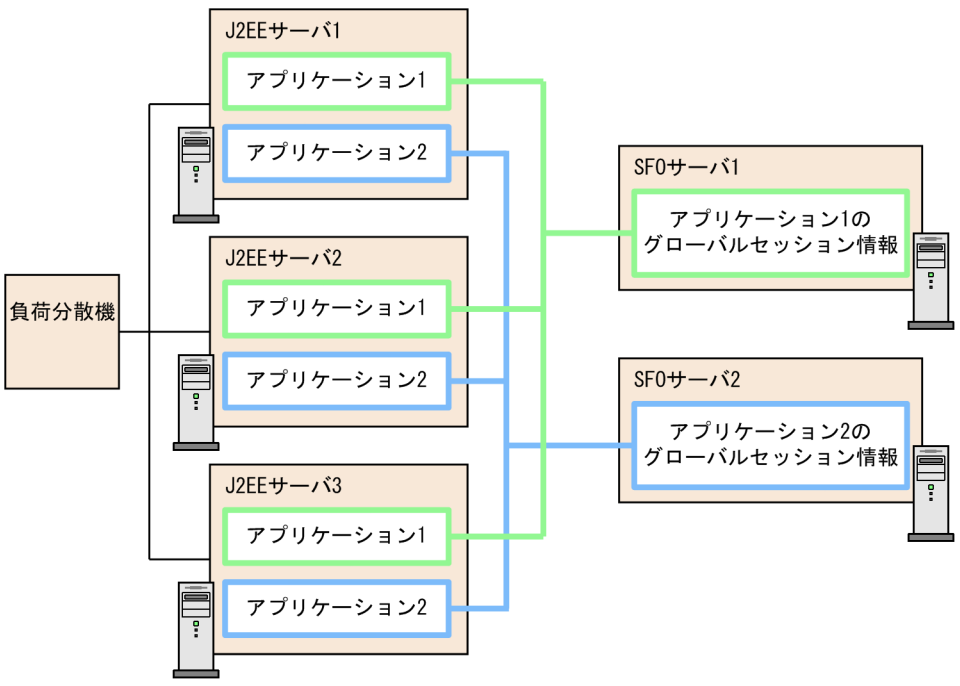
注※

削除されるのは，書き換え可能グローバルセッション情報です。読み込み専用のグローバルセッション情報は削除されません。書き換え可能，および読み込み専用のグローバルセッション情報については，「6.4.2　グローバルセッション情報」を参照してください。

SFO サーバを複数配置するときの推奨構成

SFO サーバを複数配置する構成では，J2EE サーバごとに SFO サーバを配置するのではなく，J2EE サーバ内の Web アプリケーションごとに SFO サーバを割り当てることをお勧めします。これによって，SFO サーバに障害が発生した場合の影響範囲を，障害が発生した SFO サーバを使用していた Web アプリケーションだけにとどめることができます。配置例を次の図に示します。

図 6-7　複数の SFO サーバの配置例



図のように，アプリケーション 1 のグローバルセッション情報を SFO サーバ 1 で，アプリケーション 2 のグローバルセッション情報を SFO サーバ 2 で管理している場合，SFO サーバ 1 に障害が発生しても，影響範囲はアプリケーション 1 だけで済みます。また，ほかの Web アプリケーションに影響を与えることなく，SFO サーバの再起動を伴う Web アプリケーションの設定変更をすることもできます。

● SFO サーバでの J2EE サーバの切断検知

SFO サーバでは，SFO サーバと一度接続した J2EE サーバについて，切断検知ができます。J2EE サーバの切断は，Stateful Session Bean のタイムアウトで検知できます。J2EE サーバとの接続が切断された場合，デフォルトで最大 6 分後に J2EE サーバの切断を検知できます。なお，SFO サーバが J2EE サーバの切断を検知する前に，一度切断された J2EE サーバが，切断前に SFO サーバに格納したセッション情報を取得するために SFO サーバに再接続することがあります。この場合，SFO サーバでは，J2EE サーバがいったん切断されたことを検知し，再接続をします。

(2)　SFO サーバアプリケーション

SFO サーバ上では，SFO サーバで使用する J2EE アプリケーションが稼働しています。この J2EE アプリケーションを，**SFO サーバアプリケーション**といいます。

SFO サーバアプリケーションは，グローバルセッション情報を管理する機能が実装されたアプリケーションです。EJB で実装されています。

SFO サーバアプリケーションはアプリケーションサーバで提供しています。SFO サーバをセットアップするときに，SFO サーバアプリケーションを J2EE サーバにインポートしてください。

ポイント

Smart Composer 機能を使用してシステムを構築する場合は，sfo-tier を使用してシステムを構築すると，SFO サーバアプリケーションが SFO サーバに自動的にインポートされます。

(3)　セッションフェイルオーバ用フィルタ

セッションフェイルオーバ用フィルタは，J2EE 仕様で規定された javax.servlet.Filter インタフェースを持つクラスです。J2EE アプリケーションで実行中のセッション情報を管理するサーブレットフィルタで，J2EE サーバで障害が発生したときに，管理しているセッション情報をほかの J2EE サーバに引き継ぎます。セッションフェイルオーバ用フィルタでは，セッション情報を管理し，J2EE サーバ間のセッションの情報を引き継ぐために，次の内容を設定します。

- グローバルセッション情報
- グローバルセッション情報の最大サイズ
- グローバルセッション情報を格納する SFO サーバの名称
- グローバルセッションの制御方法
- グローバルセッション ID の送信に使用する Cookie の名称
- グローバルセッション ID の送信に使用する URL のパスパラメタ名
- グローバルセッション情報のロック待ちタイムアウト
- byte 型配列であるセッション情報の引き継ぎ許可
- 直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可

セッションフェイルオーバ用フィルタは，メモリセッションフェイルオーバ機能の対象にする J2EE アプリケーションに追加して使用します。

6.4.4　グローバルセッション情報が引き継がれるセッションの条件

ここでは，メモリセッションフェイルオーバ機能を使用できるセッションの条件と，サーバ障害が発生したときのグローバルセッション情報の引き継ぎについて説明します。

メモリセッションフェイルオーバ機能では，次の二つの条件を満たしているセッションの場合に，グローバルセッション情報が引き継がれます。

- SFO サーバが正常に稼働している状態で作成されたセッションであること。
- 障害が発生した場所がクライアントから送信されるリクエストの間で，どの J2EE サーバからも使用されていないセッションであること。

これらの条件を満たしていない場合は，グローバルセッション情報は引き継がれず，破棄されます。

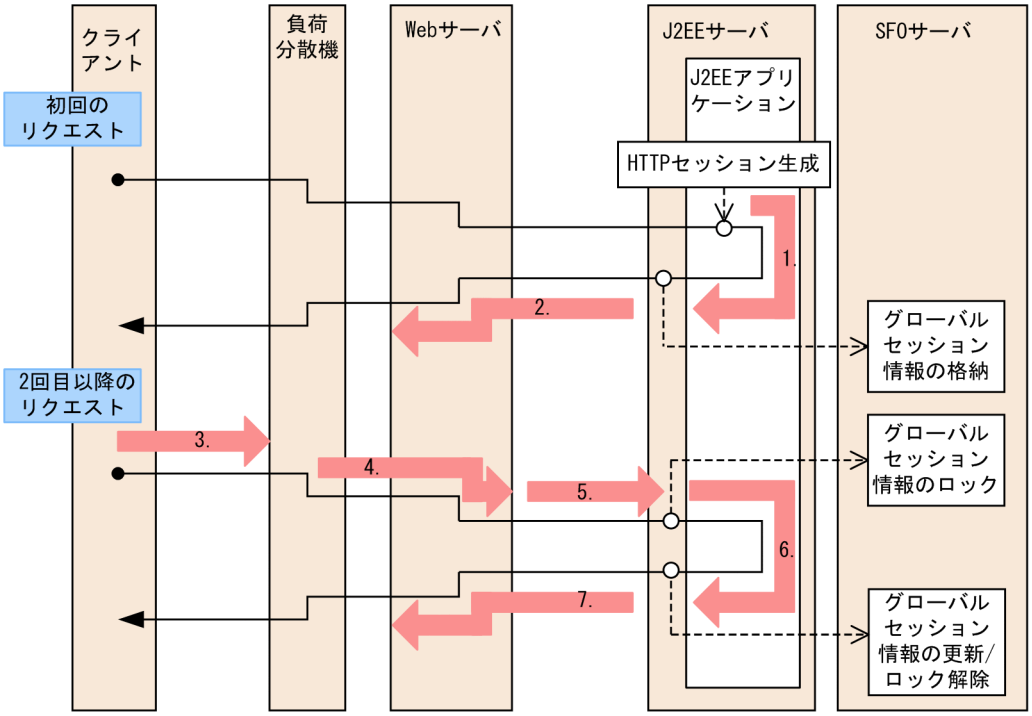
また，障害発生時に J2EE サーバで使用しているセッションについても，グローバルセッション情報は引き継がれません。障害が発生した J2EE サーバで処理されていたリクエストは，エラーとしてクライアントに返され，セッションが切断されます。

次に，Web サーバ，J2EE サーバ，SFO サーバで障害が発生したときの，リクエストの処理位置による，グローバルセッション情報の引き継ぎの有無について説明します。

(1)　Web サーバで障害が発生した場合

Web サーバで障害が発生した場合，障害発生時のリクエストの処理の位置にかかわらず，リクエスト自体は正常に処理され，グローバルセッション情報が引き継がれます。障害発生時のリクエストの処理位置と，グローバルセッション情報の引き継ぎについて次の図に示します。

図 6-8　障害発生時のリクエストの処理位置とグローバルセッション情報の引き継ぎ（Web サーバでの障害発生の場合）



（凡例）　：リクエストの処理。障害発生時，グローバルセッション情報が引き継がれる。

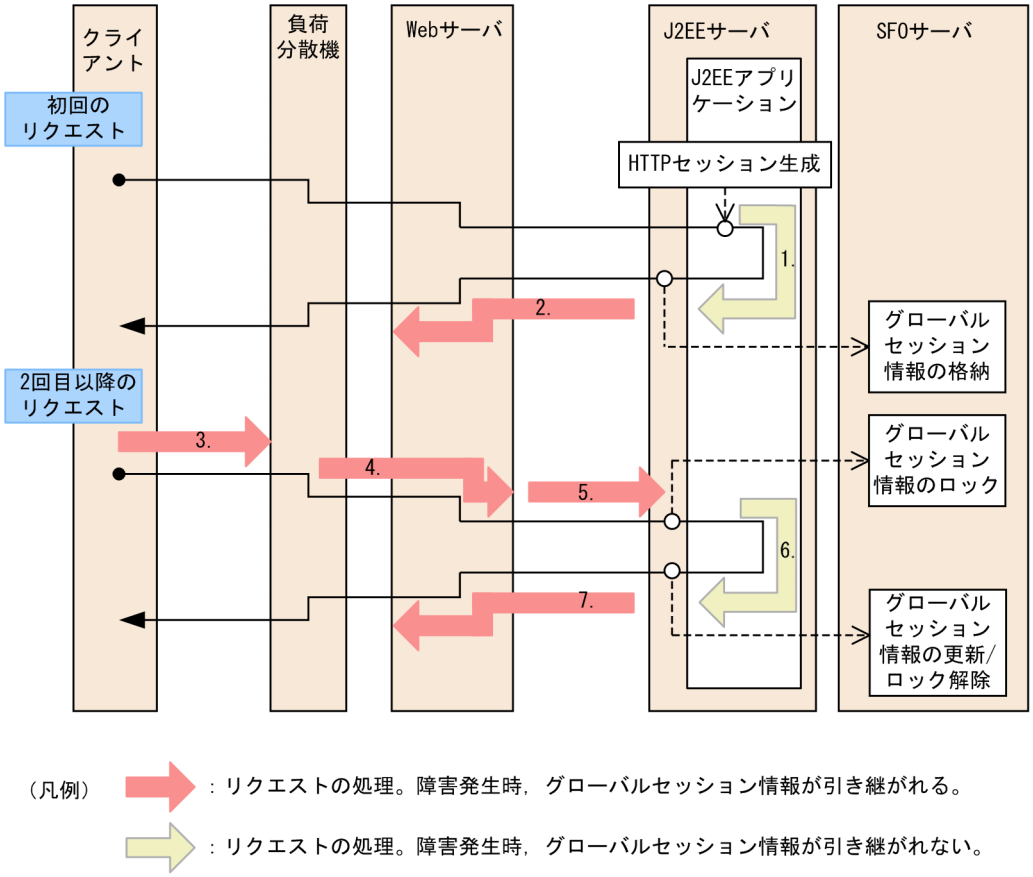
クライアントへのレスポンスについて

- 図中の項番 1.，2.，4.，5.，6.，7.のリクエスト処理中に障害が発生した場合
  クライアントへのレスポンスはエラーとなります。
- 図中の項番 3.のリクエスト処理中に障害が発生した場合
  クライアントへのレスポンスは正常に返ります。

(2)　J2EE サーバで障害が発生した場合

J2EE サーバで障害が発生した場合，障害発生時のリクエストの処理の位置が J2EE サーバ上にあるときは，グローバルセッション情報が破棄され，引き継ぎできません。障害発生時のリクエストの処理位置と，グローバルセッション情報の引き継ぎについて次の図に示します。

図 6-9　障害発生時のリクエストの処理位置とグローバルセッション情報の引き継ぎ（J2EE サーバでの障害発生の場合）



クライアントへのレスポンスについて

- 図中の項番 1.，4.，5.，6.のリクエスト処理中に障害が発生した場合
  クライアントへのレスポンスはエラーとなります。
- 図中の項番 2.，3.，7.のリクエスト処理中に障害が発生した場合
  クライアントへのレスポンスは正常に返ります。

(3)　SFO サーバで障害が発生した場合

SFO サーバで障害が発生した場合，グローバルセッションはすべて切断されます。また，SFO サーバでの障害発生後のリクエストは，書き換え可能グローバルセッション情報は破棄されて，処理されます。

6.4.5　サーバの状態とグローバルセッション情報の引き継ぎ

サーバの停止時，およびサーバの障害発生時のセッション情報の引き継ぎについて説明します。

ここでは，次の状態の場合のセッション情報の引き継ぎについて説明します。

- J2EE サーバまたは SFO サーバ停止時のセッション情報の引き継ぎ
- Web サーバ，J2EE サーバまたは SFO サーバでの障害発生時のセッション情報の引き継ぎ

(1)　J2EE サーバ停止時のセッション情報の引き継ぎ

J2EE サーバを停止すると，J2EE サーバ内の HTTP セッションは破棄されますが，SFO サーバ上にあるグローバルセッション情報は破棄されません。このため，別の J2EE サーバにグローバルセッション情報を引き継いだり，停止した J2EE サーバの再起動後にグローバルセッション情報を引き継いだりできます。

なお，SFO サーバ上のグローバルセッション情報は，SFO サーバを再起動したときに破棄されます。

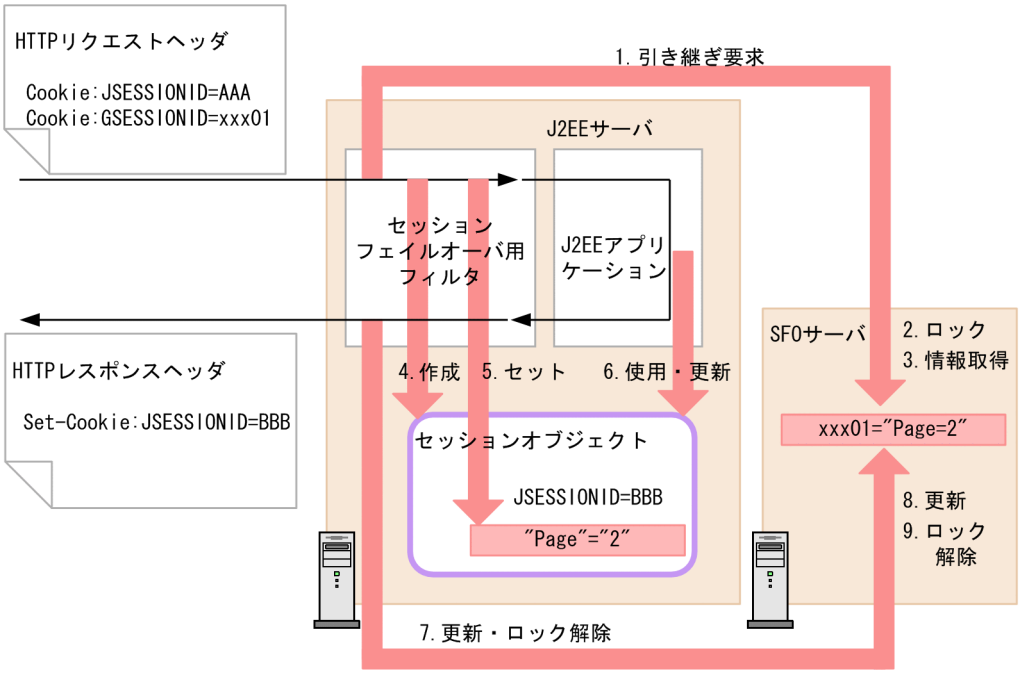
(2)　SFO サーバ停止時のセッション情報の引き継ぎ

SFO サーバを停止すると，SFO サーバ上のグローバルセッション情報はすべて破棄されます。SFO サーバの停止も，SFO サーバで障害が発生したときと同様に，J2EE サーバ上の HttpSession オブジェクトに含まれる書き換え可能グローバルセッション情報も削除されるので注意してください。

(3)　Web サーバまたは J2EE サーバの障害発生時のセッション情報の引き継ぎ

Web サーバまたは J2EE サーバで障害が発生したとき，リクエストは，別の J2EE サーバに転送されます。このとき，グローバルセッション情報の引き継ぎ処理が発生します。Web サーバまたは J2EE サーバで障害が発生したときの，セッション情報の引き継ぎについて次の図に示します。なお，この図は，HTTP リクエストヘッダの Cookie ヘッダにグローバルセッション ID があるが，Cookie ヘッダに含まれる JSESSIONID に対応する HttpSession オブジェクトが J2EE サーバにない場合の例です。

図 6-10　Web サーバまたは J2EE サーバの障害発生時のセッション情報の引き継ぎ



（凡例）　：書き換え可能グローバルセッション情報

→　：リクエストまたはレスポンスの流れ

注1　CookieヘッダおよびSet-Cookieヘッダのグローバルセッション IDのHTTP Cookieの名称は，「GSESSIONID」がデフォルトとなります。

注2　CookieヘッダのJSESSIONIDは，J2EEで規定されたHTTPセッションのセッション IDを HTTP Cookieとして付けるときのHTTP Cookie名称です。

1. SFO サーバにグローバルセッション ID に対応するグローバルセッション情報の引き継ぎ要求をします。

2. グローバルセッション情報を検索し，ロックします。
3. グローバルセッション情報を J2EE サーバに転送します。
4. 新規 HttpSession オブジェクトを作成します。
5. 3.で取得したグローバルセッション情報を，4.で作成した新規 HttpSession オブジェクトに格納します。
6. J2EE アプリケーション内の処理で，引き継がれたグローバルセッション情報を使用します。
7. J2EE アプリケーション処理の終了後，セッションフェイルオーバ用フィルタでグローバルセッション情報を SFO サーバに送信します。
8. SFO サーバ上のグローバルセッション情報を更新します。
9. SFO サーバ上のグローバルセッション情報のロックを解除します。

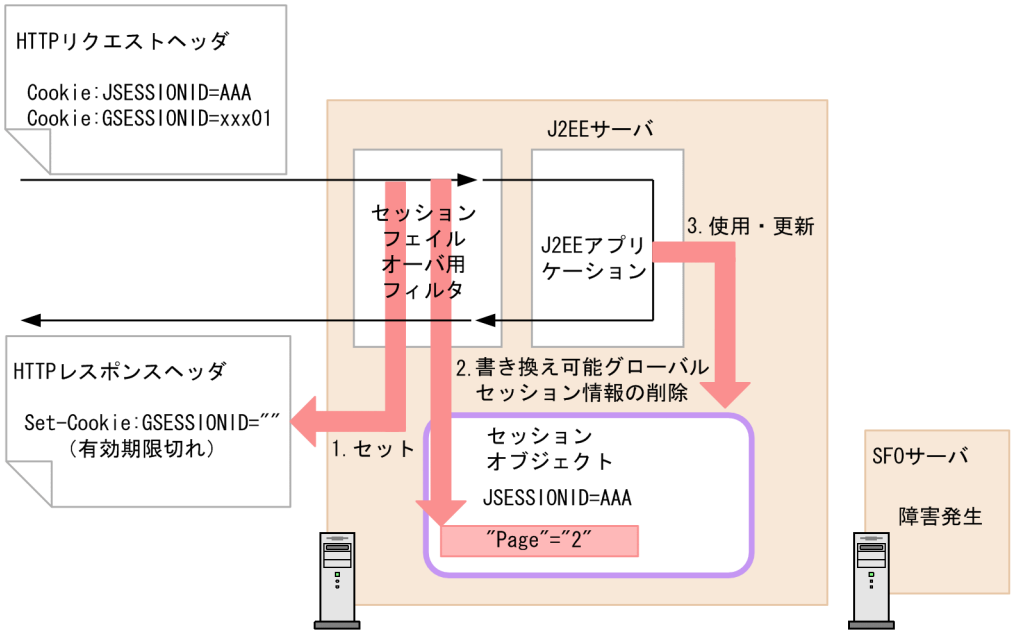
(4)　SFO サーバの障害発生時のセッション情報の引き継ぎ

SFO サーバに障害が発生している間は，メモリセッションフェイルオーバ機能が無効になり，HTTP セッションによるセッション管理で業務を続けます。このため，SFO サーバに障害が発生している間は，グローバルセッションは生成されません。

SFO サーバの障害発生中に作成された HttpSession オブジェクトは，SFO サーバが復旧すると，最初にリクエストが送信された時点で，対応するグローバルセッションが生成され，SFO サーバに格納されます。

SFO サーバに障害が発生すると，J2EE サーバ上にある HttpSession オブジェクト内の書き換え可能グローバルセッション情報を削除します。SFO サーバ障害発生時のグローバルセッション情報の削除処理の動作を次の図で説明します。

図 6-11　SFO サーバ障害発生時のグローバルセッション情報の削除処理



（凡例）　：書き換え可能グローバルセッション情報

：リクエストまたはレスポンスの流れ

注1　CookieヘッダおよびSet-CookieヘッダのグローバルセッションIDのHTTP Cookieの名称は，「GSESSIONID」がデフォルトとなります。

注2　CookieヘッダのJSESSIONIDは，J2EEで規定されたHTTPセッションのセッションIDをHTTP Cookieとして付けるときのHTTP Cookie名称です。

1. グローバルセッション ID の HTTP Cookie 情報を削除するよう，HTTP レスポンスヘッダの Set-Cookie ヘッダに，有効期限切れを示す情報をセットします。
2. HttpSession オブジェクトに含まれる書き換え可能グローバルセッション情報を削除します。
3. J2EE アプリケーション内の処理では，書き換え可能グローバルセッション情報だけが削除された HttpSession オブジェクトが使用されます。

参考

書き換え可能グローバルセッション情報の削除は，SFO サーバの障害時以外にも，ロック待ちタイムアウトが発生した場合，および無効となったグローバルセッション ID を使用するリクエストが送信された場合も実施されます。

SFO サーバに障害が発生したときの，グローバルセッション ID の発行，HttpSession オブジェクトの最大数，およびグローバルセッション情報は次のようになります。

● SFO サーバの障害発生時のグローバルセッション ID の発行について

SFO サーバに障害が発生した場合，メモリセッションフェイルオーバ機能は無効になります。このため，HttpSession オブジェクトが生成されても，グローバルセッション ID は発行されません。なお，SFO サーバが回復すると，最初に実施されるロック処理でグローバルセッション ID が発行されます。ロックが解除されたときに SFO サーバにグローバルセッション情報が格納されます。

● SFO サーバに障害が発生した場合の HttpSession オブジェクトの最大数について

SFO サーバに障害が発生した場合，メモリセッションフェイルオーバ機能は無効になります。SFO サーバが停止していると，J2EE サーバでは，SFO サーバで管理できるグローバルセッション情報の最大数を超えて，HTTP セッション数の上限まで HttpSession オブジェクトが生成されます。

SFO サーバが回復すると，リクエスト処理で使用されたセッション情報から順に SFO サーバに格納されます。このとき，SFO サーバに格納できない分の HttpSession オブジェクトについては，リクエストを受信した時点で破棄され，新規リクエストとして再度生成されて，J2EE アプリケーションに送られます。

● SFO サーバに障害が発生した場合のグローバルセッション情報について

SFO サーバに障害が発生したあとのグローバルセッション情報について，種類ごとに説明します。なお，グローバルセッションの種類の概要については，「6.4.2　グローバルセッション情報」を参照してください。

書き換え可能グローバルセッション情報の場合

SFO サーバに障害が発生した場合，SFO サーバによる管理ができなくなるため，書き換え可能グローバルセッション情報は使用できなくなります。

読み込み専用グローバルセッション情報の場合

継続して使用できます。

SFO サーバで障害が発生したあとのリクエストは，書き換え可能グローバルセッション情報を削除した HttpSession オブジェクトで処理を続行します。クライアントへのレスポンスも正常に返ります。

また，セッションフェイルオーバ用フィルタが SFO サーバに対してロック要求をした延長で SFO サーバに障害が発生した場合は，次のように処理されます。

- 書き換え可能グローバルセッション情報の削除をしたあとの HttpSession オブジェクトに対して新しいグローバルセッション ID が作成されます。
- SFO サーバに対するロック処理が実行されます。

- J2EE アプリケーションが処理されます。

この処理が発生した場合，HttpSession オブジェクトの属性に次に示すキー名称と値が登録されます。

- キー名称：com.hitachi.software.was.sfo.gsession.status
- 値：unavailable

なお，この属性は，SFO サーバの復旧後，グローバルセッション情報が SFO サーバに格納されると削除されます。また，この属性は，SFO サーバに冗長化されないため，引き継ぎの対象になりません。

● SFO サーバに障害が発生した場合の HTTP セッション

SFO サーバに障害が発生したあとでも，Web アプリケーションで作成された新しい HTTP セッションは使用できます。ただし，メモリセッションフェイルオーバ機能は無効になっているため，SFO サーバには格納されません。

なお，SFO サーバ復旧後，SFO サーバに障害が発生しているときに作成された HTTP セッションは，リクエスト処理で使用された HTTP セッションから順に，SFO サーバに格納され，メモリセッションフェイルオーバ機能が有効になります。このとき，SFO サーバの障害発生中に登録された書き換え可能グローバルセッション情報や読み込み専用グローバルセッション情報は，そのまま SFO サーバに格納されます。

6.4.6　グローバルセッションの制御方法

ここでは，グローバルセッションの制御方法について説明します。

(1)　グローバルセッションの制御方法の種類

グローバルセッションは，次のどちらかの方法で制御できます。

- HTTP Cookie による制御

  グローバルセッション ID を HTTP Cookie によって管理する制御方法です。J2EE サーバから Web クライアントにレスポンスを返すときに，グローバルセッション ID を含む HTTP Cookie を設定します。次に Web クライアントが J2EE サーバにリクエストを送るときには，その HTTP Cookie を送信します。これによって，グローバルセッションの制御を実現します。

- URL 書き換えによる制御

  グローバルセッション ID を URL のパスパラメタに含めて管理する制御方法です。

  J2EE サーバが Web クライアントに返すページ遷移用の URL を書き換えるときに，グローバルセッション ID を含めてエンコードします。次に Web クライアントが J2EE サーバにリクエストを送るときには，このエンコードされた URL を使用します。これによって，グローバルセッションの制御を実現します。

使用するクライアントの種類などに応じて，適切な制御方法を選択してください。それぞれの制御方法の詳細については，「6.6　HTTP Cookie によるグローバルセッションの制御」および「6.7　URL 書き換えによるグローバルセッションの制御」を参照してください。

制御方法は，セッションフェイルオーバ用フィルタに設定します。設定方法については，「6.12　DD での定義」を参照してください。

参考

両方の制御方法を使用する設定をしている場合に，HTTP Cookie および URL のパスパラメタの両方にグローバルセッション ID が含まれているときの動作

セッションフェイルオーバ用フィルタでの制御方法の設定では，次の設定ができます。

- HTTP Cookie を使用することを示す設定（cookie）
- URL 書き換えを使用することを示す設定（url-rewriting）
- 両方の制御方法を使用することを示す設定（both）

「both」を設定している場合に，HTTP Cookie および URL のパスパラメタの両方にグローバルセッション ID が含まれているときには，HTTP Cookie に含まれるグローバルセッション ID が有効になります。このとき，URL のパスパラメタに含まれているグローバルセッション ID は無効になります。

なお，URL 書き換えによるグローバルセッション制御は，J2EE サーバおよび SFO サーバのバージョンが 07-50 以降の場合に使用できます。

(2)　グローバルセッション情報の制御方法の選択指針

HTTP Cookie によって制御できる場合は，HTTP Cookie による制御を選択してください。

URL 書き換えによる制御をする場合，リクエストおよびレスポンスの両方に，HTTP セッションのセッション ID とグローバルセッション ID が含まれることになります。この方法は，一般的にセッションハイジャックなどの危険性が高まるといわれています。

HTTP Cookie を使用できないクライアントを使用するシステムなど，特定のシステム以外では，URL 書き換えによる制御は使用しないことをお勧めします。

(3)　注意事項

Web クライアントから受信したリクエストが，次の条件をすべて満たす場合，グローバルセッションが作成されません。

- HTTP セッションが存在する。
- Web アプリケーションの DD に，グローバルセッションの制御方法として，HTTP Cookie，または URL 書き換えのどちらかを使用すると設定している（「both」を設定していない）。
- すでに存在する HTTP セッションの制御方法が，Web アプリケーションに設定したグローバルセッションの制御方法と異なる。

これらの条件に当てはまるリクエストに関連づけられた HTTP セッションは，メモリセッションフェイルオーバ機能の対象にはなりません。この場合，リクエストの受信時に KDJE34524-W のメッセージが出力されます。

ただし，グローバルセッションが作成されない場合も，HTTP セッションは継続されます。

6.4.7　グローバルセッション情報の排他制御

ここでは，SFO サーバで管理しているグローバルセッション情報の排他制御について説明します。

メモリセッションフェイルオーバ機能では，複数の J2EE サーバから同じグローバルセッション情報を同時に更新されることを抑止するために，ロックを使用して排他制御を実現します。

J2EE サーバは，セッションに対するリクエスト処理を開始するときに，SFO サーバ上のグローバルセッション情報にロックを設定します。また，リクエスト処理が完了した時点でロックを解除します。

別の J2EE サーバによってすでにロックが設定されているグローバルセッション情報をロックしようとした場合は，設定されているロックが解除されるまで，処理が待たされます。これを，**ロック待ち**といいます。

なお，ロック待ちには，タイムアウト時間を設定できます。タイムアウトが発生した場合，J2EE サーバは，該当するグローバルセッション情報が使用できないものと判断します。この場合，J2EE サーバは，SFO サーバ上の書き換え可能グローバルセッション情報を破棄して，J2EE アプリケーションを呼び出し，リクエスト処理を実行します。

ロック待ちのタイムアウト時間は，Web アプリケーション単位で設定できます。設定方法については，「6.12.1　DD の作成」を参照してください。

参考

異なる J2EE サーバに振り分けられた場合，グローバルセッション情報のロック機能を使用して，更新業務を実行する J2EE アプリケーションの排他制御も実現できます。

更新業務を実行する J2EE アプリケーションでは，ダブルクリックなどによってリクエストが二重に送信されると，正常に処理が実行できなくなる場合があります。このため，J2EE アプリケーションでは，セッションなどを利用して，このような場合に二重に処理するのを抑止する必要があります。

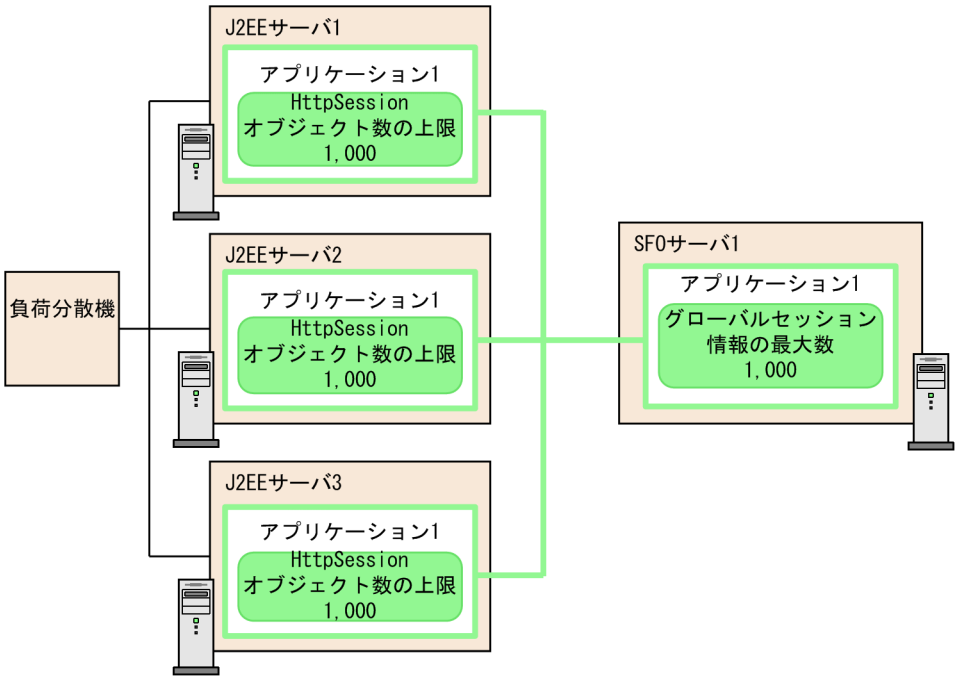
メモリセッションフェイルオーバ機能を使用している場合，ダブルクリックなどによって二重送信されたリクエストが異なる J2EE サーバに送信されたときには，グローバルセッション情報のロック設定時にロック待ちが発生します。これによって，処理が二重に実施されるのを防げます。

6.4.8　グローバルセッション情報の最大数

SFO サーバで管理できるグローバルセッション情報の最大数は，SFO サーバと接続している Web アプリケーションの HttpSession オブジェクト数の上限値となります。

Web アプリケーションの HttpSession オブジェクト数の上限値と，SFO サーバで管理できるグローバルセッション情報の最大数の関係を次の図に示します。

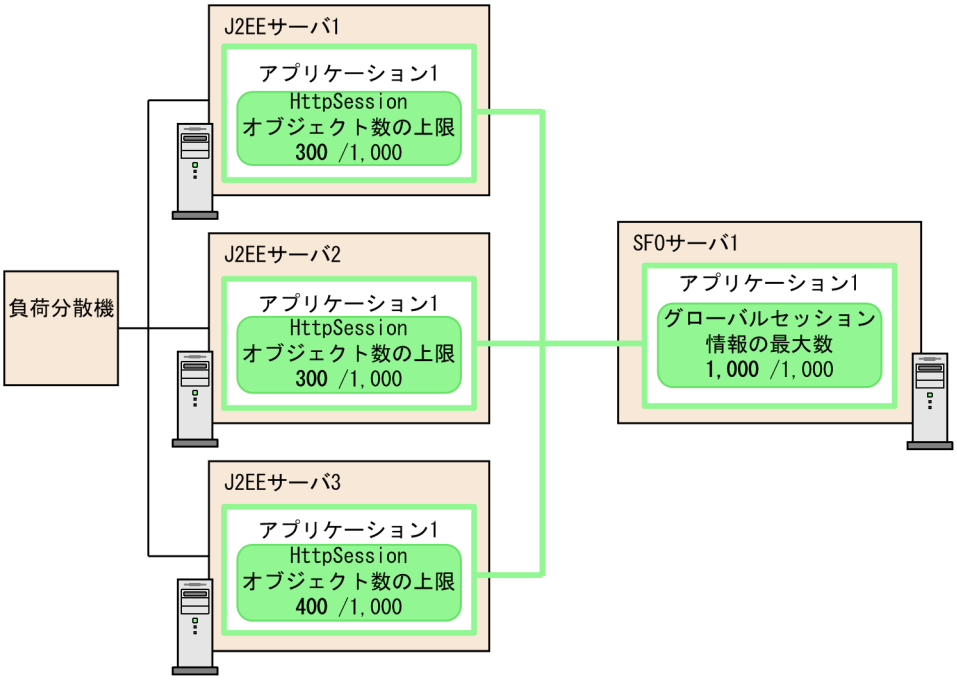
図 6-12　SFO サーバで管理できるグローバルセッション情報の最大数



図のように，各 Web アプリケーションの，HttpSession オブジェクト数の上限値に 1,000 が設定されている場合，SFO サーバで管理できるグローバルセッション情報の最大数も 1,000 となります。

しかし，実際には SFO サーバは，複数の J2EE サーバで生成された HTTP セッションを一括して管理するため，Web アプリケーションの HttpSession オブジェクト数の上限値に達する前に，SFO サーバ内のグローバルセッション情報の数が最大数に達することがあります。次の図で説明します。

図 6-13　HttpSession オブジェクト数の上限と SFO サーバのグローバルセッション情報の最大数との関係



SFO サーバで管理するグローバルセッション情報の数は，各 Web アプリケーションの HttpSession オブジェクト数の合計となるため，図のように，Web アプリケーションの HttpSession オブジェクト数が上限値に達していなくても，SFO サーバのグローバルセッション情報の数が最大数に達する場合があります。このとき，J2EE サーバ上の J2EE アプリケーション内で，javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドを呼び出すと，IllegalStateException が送出されます。

なお，HttpSession オブジェクト数の上限値設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.5　HttpSession オブジェクト数の上限値の設定」を参照してください。

6.5　グローバルセッション情報のサイズの見積もり

ここでは，グローバルセッション情報を見積もるための機能について説明します。

グローバルセッション情報のサイズは，あらかじめ最大サイズを見積もって，DD（web.xml）に定義する必要があります。しかし，次の場合，セッション情報のサイズを見積もることは困難です。

- 直列化可能クラスのオブジェクトの場合，HTTP セッションに登録されたオブジェクトをシリアライズして作成される。
- java.lang.String クラスのオブジェクトの場合，データの内容が非 ASCII 文字を含むときは，文字種によって文字列長が変わる。

グローバルセッション情報のサイズ見積もり機能は，開発環境でグローバルセッション情報のサイズを見積もるための機能です。グローバルセッション情報を作成してそのサイズを Web サーブレットログに出力します。この機能は開発環境で使用してください。このとき，SFO サーバのセットアップや設定，および J2EE サーバでの設定は不要です。

！ 注意事項

グローバルセッション情報のサイズ見積もり機能は，運用環境では使用しないでください。この機能を使用すると，セッション情報の引き継ぎは無効となり，メモリセッションフェイルオーバ機能が有効になりません。グローバルセッション情報のサイズ見積もり機能使用時に無効となる処理については，「6.5.3　グローバルセッション情報のサイズ見積もり機能使用時に無効となる処理」を参照してください。

6.5.1　グローバルセッション情報のサイズ見積もり機能を有効にするための設定（見積もり時に使用する設定）

グローバルセッション情報のサイズ見積もり機能を有効にする場合は，簡易構築定義ファイルと，DD（web.xml）で設定が必要になります。

- **簡易構築定義ファイルでのグローバルセッション情報のサイズ見積もり機能を有効にする設定**

  論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で，webserver.sfo.check_size.mode パラメタに「on」を設定します。グローバルセッション情報のサイズ見積もり機能を無効にする場合は，このパラメタに「off」を設定してください。簡易構築ファイルおよび指定するパラメタの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

  グローバルセッション情報のサイズ見積もり機能を使用する場合の簡易構築定義ファイルのパラメタの設定要否を次の表に示します。

表 6-9　グローバルセッション情報のサイズ見積もり機能を使用する場合の簡易構築定義ファイルのパラメタの設定要否

簡易構築定義ファイルのパラメタ	パラメタの設定要否
webserver.sfo.sfo_servers	無効※
webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.naming	無効※
webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.connectionTimeout	無効※
webserver.sfo.check_size.mode	必須

簡易構築定義ファイルのパラメタ	パラメタの設定要否
webserver.session.server_id.enabled	任意
webserver.session.server_id.value	任意
webserver.container.server_id.enabled	任意
webserver.container.server_id.name	任意
webserver.container.server_id.value	任意
webserver.connector.inprocess_http.limit.max_request_line	任意

（凡例）

必須：必ず指定することを示す。省略した場合はエラーとなり，SFO サーバ用アプリケーションを開始できない。

無効：指定しても無効になることを示す。設定した値は使用されない。

任意：省略できることを示す。省略した場合はデフォルトの値が設定される。

注※

パラメタに設定した値は読み込みまれません。このため，不正な値を設定した場合も検出されません。

参考

Smart Composer 機能を使用しない場合に，グローバルセッション情報のサイズ見積もり機能を有効にするときは，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）で webserver.sfo.check_size.mode キーに「on」を設定します。webserver.sfo.check_size.mode キーについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.4　usrconf.properties（J2EE サーバ用ユーザプロパティファイル）」を参照してください。

- DD（web.xml）でのセッションフェイルオーバ用フィルタの定義

  メモリセッションフェイルオーバ機能の設定をします。DD（web.xml）でのセッションフェイルオーバ用フィルタの定義については，「6.12.1　DD の作成」を参照してください。グローバルセッション情報のサイズ見積もり機能では，DD（web.xml）に定義したメモリセッションフェイルオーバ機能の設定を基にグローバルセッション情報が作成されます。

6.5.2　グローバルセッション情報のサイズの見積もり方法

この機能を使用すると，メモリセッションフェイルオーバ機能が有効な Web アプリケーションに対してリクエストを送信するたびに，グローバルセッション情報のサイズを示すメッセージが Web サーブレットログに出力されます。グローバルセッション情報のサイズを出力するメッセージを次の表に示します。

表 6-10　グローバルセッション情報のサイズを出力するメッセージ

メッセージ ID	出力レベル	出力するタイミング	出力する内容
KDJE34260-I	Error	リクエストごと	リクエスト一つ分に対応するグローバルセッション情報のサイズ
KDJE34261-I	Error	HTTP セッションに登録されるセッション情報ごと	キー名称，区切り文字などを含むセッション情報一つ分に対応するデータのサイズ

各メッセージの出力例を次に示します。この例では，キー名称が Attribute1 と Attribute2 の二つのセッション情報からグローバルセッション情報を作成します。

KDJE34261-I　The session information was serialized.（J2EE application = App01，context root = /test，request URL = http://host01/test/TestServlet，attribute name = Attribute1，class name = app.MyObject1※，size（byte）= **36**，HTTP session ID = 01234567aaaabbbbccccddddeeeeffffserver1，global session ID = 0123456789abcdef01234567aaaabbbbccccddddeeeeffffserver1）

KDJE34261-I　The session information was serialized.（J2EE application = App01，context root = /test，request URL = http://host01/test/TestServlet，attribute name = Attribute2，class name = app.MyObject2※，size（byte）= **25**，HTTP session ID = 01234567aaaabbbbccccddddeeeeffffserver1，global session ID = 0123456789abcdef01234567aaaabbbbccccddddeeeeffffserver1）

KDJE34260-I　The global session information was created.（J2EE application = App01，context root = /test，request URL = http://host01/test/TestServlet，size（byte）= **65**，HTTP session ID = 01234567aaaabbbbccccddddeeeeffffserver1，global session ID = 0123456789abcdef01234567aaaabbbbccccddddeeeeffffserver1）

注　**太字**の部分に，グローバルセッション情報のサイズを見積もるための値が出力されます。各メッセージの size（byte）に出力される内容の詳細については，表 6-11 を参照してください。また，各メッセージの詳細については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。

注※　クラス名に出力される内容は，属性値に対して java.lang.Class クラスの getName()メソッドで得られた文字列です。なお，KDJE34261-I のメッセージは，引き継ぎの対象であるセッション情報が HTTP セッションにない場合は出力されません。また，KDJE34260-I のメッセージは，引き継ぎの対象であるセッション情報が HTTP セッションにあるかどうかに関係なく，HTTP セッションがあれば出力されます。

これらのメッセージに出力されるサイズ（size（byte））の値を基に，グローバルセッション情報のサイズを見積もります。見積もった値が，SFO サーバで冗長化できるグローバルセッション情報のサイズのデフォルト値（1,024 バイト）を超えている場合は，DD（web.xml）でグローバルセッション情報の最大サイズの設定を変更する必要があります。グローバルセッション情報の最大サイズの設定方法については，「6.12.2(2)　グローバルセッション情報の最大サイズ（GSInfosLengthMax)」を参照してください。

なお，KDJE34260-I および KDJE34261-I のメッセージに出力されるサイズの値は，直列化可能クラスのオブジェクトを引き継ぎ対象としているかどうかによって異なります。KDJE34260-I および KDJE34261-I のメッセージに出力されるサイズ（size（byte））の詳細を次の表に示します。

表 6-11　KDJE34260-I および KDJE34261-I のメッセージに出力されるサイズ（size（byte））の詳細

メッセージ ID の種類	直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可（GSInfoTypeSerializableEnable）の設定	
	true（直列化可能クラスのオブジェクトが引き継ぎ対象の場合）	false（直列化可能クラスのオブジェクトが引き継ぎ対象外の場合）
KDJE34260-I	次の値を合計したサイズが出力されます。 • HTTP セッションの有効期間情報の長さ • 各属性のサイズ（KDJE34261-I のメッセージで出力したサイズ）の合計	次の値を合計したサイズが出力されます。 • HTTP セッションの有効期間情報の長さ • 各属性を連結する区切り文字分のサイズ（各 1 バイト） • 各属性のサイズ（KDJE34261-I のメッセージで出力したサイズ）の合計
KDJE34261-I	次の値を合計したサイズが出力されます。 • 属性名をシリアライズしたデータのサイズ • 属性値をシリアライズしたデータのサイズ • java.io.ObjectOutputStream クラスが書き込むマジックナンバー，およびバージョン情報のデータ分のサイズ※1	次の値を合計したサイズが出力されます。 • 属性名のデータのサイズ • 属性値のデータのサイズ • 属性名と属性値の区切り文字のサイズ（1 バイト）

メッセージ ID の種類	直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可（GSInfoTypeSerializableEnable）の設定	
	true（直列化可能クラスのオブジェクトが引き継ぎ対象の場合）	false（直列化可能クラスのオブジェクトが引き継ぎ対象外の場合）
KDJE34261-I	次の値を合計したサイズが出力されます。 • 属性名をシリアライズしたデータのサイズ • 属性値をシリアライズしたデータのサイズ • java.io.ObjectOutputStream クラスが書き込むマジックナンバー，およびバージョン情報のデータ分のサイズ※1	• byte 型配列の管理情報の長さ※2

注※1　HTTP セッションに登録された属性のうち，最初にシリアライズされる属性のサイズにだけ含まれます。

注※2　byte 型配列であるセッション情報が引き継ぎ対象の場合にだけ含まれます。

6.5.3　グローバルセッション情報のサイズ見積もり機能使用時に無効となる処理

グローバルセッション情報のサイズ見積もり機能を使用すると，グローバルセッション情報は作成しますが，セッション情報の引き継ぎ処理は無効となります。この場合，SFO サーバにセッション情報が冗長化されません。このほかにも，メモリセッションフェイルオーバ機能で実行する処理で無効となるものがあります。グローバルセッション情報のサイズ見積もり機能使用時に，無効となる処理を次に示します。

- J2EE サーバと SFO サーバとのネゴシエーション処理
- SFO サーバへのグローバルセッション情報の冗長化
- グローバルセッション情報の引き継ぎ処理

6.5.4　グローバルセッション情報のサイズ見積もり機能使用時の注意事項

グローバルセッション情報のサイズ見積もり機能使用時の注意事項を次に示します。

- メモリセッションフェイルオーバ機能で引き継ぎ可能なサイズの上限は，HTTP セッションの有効期間情報などのサイズを含めて，1,048,576（1M）バイトです。グローバルセッション情報のサイズ見積もり機能使用時に，この上限値を超えた場合は，KDJE34262-E のメッセージが出力されます。それ以降のグローバルセッション情報の作成処理は中断されます。この場合，KDJE34260-I のメッセージは出力されません。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。
- グローバルセッション情報のサイズ見積もり機能を有効にしている場合は，DD（web.xml）の GSInfosLengthMax パラメタで，グローバルセッション情報の最大サイズを指定しても無視されます。このため，この機能を無効にして運用を開始する前に，KDJE34260-I のメッセージに出力されたグローバルセッション情報のサイズが，GSInfosLengthMax パラメタで指定したサイズを超えていないことを確認する必要があります。GSInfosLengthMax パラメタについては，「6.12.2(2)　グローバルセッション情報の最大サイズ（GSInfosLengthMax）」を参照してください。メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。
- グローバルセッション情報のサイズ見積もり機能を使用すると，Web アプリケーション開始時に J2EE サーバと SFO サーバとのネゴシエーション処理は実行されません。このため，KDJE34216-I のメッ

セージ（SFO サーバとの接続開始），および KDJE34205-I のメッセージ（SFO サーバとの接続完了）は出力されません。これらのメッセージの代わりに，KDJE34263-I のメッセージ（グローバルセッション情報の冗長化無効）がエラーレベルで出力されます。各メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「7.2　KDJE30000 から KDJE39999 までのメッセージ」を参照してください。

- 同じ HTTP セッションを使用する複数のリクエストを同時に送信した場合，処理中のリクエスト処理スレッドがあるときはグローバルセッション情報が作成されません。このため，グローバルセッション情報のサイズは見積もりません。この場合，最後のリクエスト処理スレッドが処理を終えた時にグローバルセッション情報のサイズを見積もって，メッセージを出力します。
- グローバルセッション情報のサイズ見積もり機能を有効にする場合，HttpSession オブジェクト数の上限値の指定は不要です。HttpSession オブジェクト数の上限値の設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.5　HttpSession オブジェクト数の上限値の設定」を参照してください。
- グローバルセッション情報のサイズ見積もり機能を有効にする場合，J2EE サーバが使用するメモリ量が増加します。J2EE サーバが使用するメモリ増加量を求める式を次に示します。

J2EE サーバが使用するメモリ増加量（メガバイト）

＝ 10×1.5 ＋ 1.5※

注※　Web コンテナ単位の最大同時実行スレッド数が 10 以上の場合は，「Web コンテナ単位の最大同時実行スレッド数×1.5 ＋ 1.5」となります。Web コンテナ単位の最大同時実行スレッド数については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.16　Web コンテナ単位での同時実行スレッド数の制御」を参照してください。

6.6　HTTP Cookie によるグローバルセッションの制御

この節では，HTTP Cookie によるグローバルセッションの制御について説明します。

6.6.1　HTTP Cookie による制御の概要

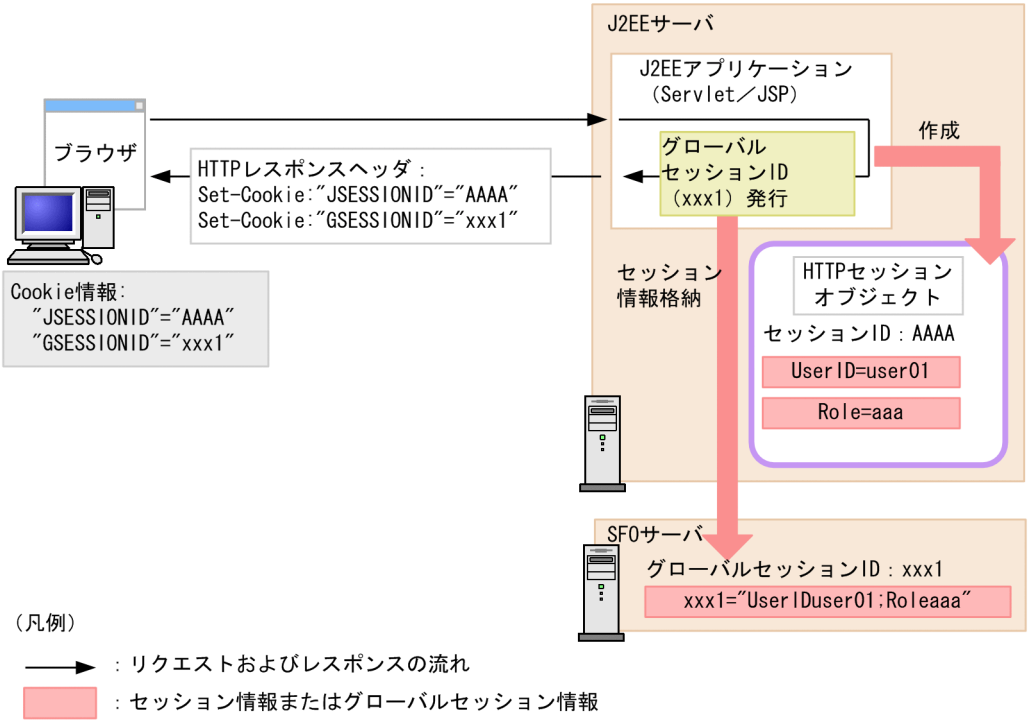
HTTP Cookie による制御では，Web クライアントから J2EE サーバにリクエストを送信するときに，グローバルセッション ID を示す HTTP Cookie を送信します。

グローバルセッション ID は，HTTP レスポンスヘッダの Set-Cookie ヘッダに設定され，Web クライアントに送信されます。この情報は，Web クライアントで，Cookie 情報として管理されます。

J2EE サーバでは，グローバルセッション ID を含む HTTP Cookie を送信してきた Web クライアントに対して，グローバルセッションを継続します。

HTTP Cookie によるグローバルセッションの制御の概要を次の図に示します。

図 6-14　HTTP Cookie によるグローバルセッションの制御の概要



Web クライアントからのリクエストを受信すると，J2EE サーバの J2EE アプリケーション（Servlet／JSP）内で，HTTP セッションオブジェクトの作成とともに，グローバルセッション ID が発行されます。発行されたグローバルセッション ID は，Web クライアントに送信される HTTP レスポンスヘッダに設定されます。同時に，SFO サーバに，グローバルセッション情報とともに格納されます。

6.6.2　HTTP Cookie による制御の前提条件

HTTP Cookie によってグローバルセッションを制御する場合，Web クライアントとして，HTTP Cookie を使用できるクライアントが必要です。また，Web クライアントのブラウザでは，HTTP Cookie セッションを有効にする必要があります。

6.6.3　グローバルセッション ID を送信する HTTP Cookie の名称

グローバルセッション ID は，HTTP レスポンスヘッダの Set-Cookie ヘッダに設定されます。使用される HTTP Cookie の名称は，Web アプリケーション単位に設定できます。デフォルトの設定の場合，「GSESSIONID」になります。設定方法については，「6.14　J2EE サーバの設定」を参照してください。

! 注意事項

グローバルセッション ID を送信する HTTP Cookie の名称には，次の名称は指定できません。なお，大文字と小文字は区別されません。

1. HTTP セッションのセッション ID で使用する HTTP Cookie の名称
   Web コンテナが自動で設定する HTTP Cookie の名称は「JSESSIONID」です。この名称はアプリケーションサーバのバージョンが 09-00 以降の場合，変更できます。HTTP Cookie の名称を変更した場合の注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.9 セッション管理の注意事項」を参照してください。
2. J2EE サーバのサーバ ID 付加機能によってサーバ ID を付加した HTTP Cookie の名称と同じ名称
3. 「csfcfc」

サーバ ID 付加機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.6　セッション ID および Cookie へのサーバ ID の付加」を参照してください。

上記の 1 または 2 と同じ名称を使用すると，セッションフェイルオーバ用フィルタの初期化処理でエラーとなり，アプリケーションが開始できなくなります。なお，セッションフェイルオーバ用フィルタの初期化処理については，「6.9.3　J2EE アプリケーション開始時の処理」を参照してください。

6.6.4　サーブレットコンテキストへの属性追加

グローバルセッション ID が使用する HTTP Cookie の名称は，Web アプリケーション開始時に，サーブレットコンテキストの属性として，「com.cosminexus.software.was.sfo.GIDCookieName」の名称で登録されます。このとき，Web アプリケーションに，javax.servlet.ServletContextAttributeListener インタフェースを実装したクラスが登録されていた場合，attributeAdded メソッドが呼び出されます。

6.6.5　HTTP Cookie の有効期限

グローバルセッション ID の送信に使用する HTTP Cookie の有効期限には，デフォルト値である-1 が設定されています。このため，HTTP Cookie の情報は，ブラウザのプロセスの終了と同時に消失します。

6.6.6　グローバルセッション破棄に伴う HTTP Cookie 情報の破棄

ブラウザ内の HTTP Cookie 情報は，デフォルトの設定ではブラウザのプロセス終了とともに消失します。また，J2EE アプリケーション内で明示的にセッションを破棄することでグローバルセッション情報が破棄されると，ブラウザ内にある HTTP Cookie 情報も破棄されます。ブラウザ内の HTTP Cookie 情報は，HTTP レスポンスヘッダに対して有効期限の切れた HTTP Cookie を設定することで破棄されます。

なお，タイムアウトによるグローバルセッション情報の破棄の場合は，ブラウザ内の HTTP Cookie 情報は明示的に破棄されません。クライアントから無効になったグローバルセッション ID が HTTP Cookie として送られてきたときに，クライアント内の HTTP Cookie 情報が削除されます。

6.7　URL 書き換えによるグローバルセッションの制御

この節では，URL 書き換えによるグローバルセッションの制御について説明します。

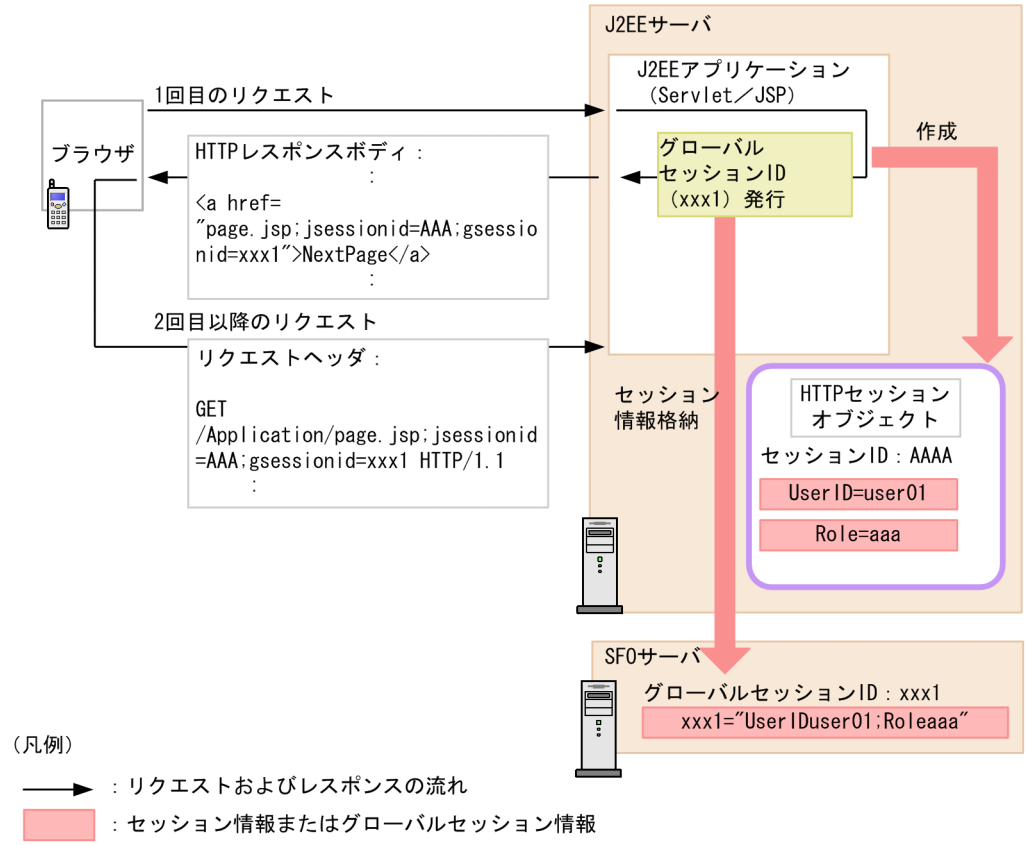
6.7.1　URL 書き換えによる制御の概要

URL 書き換えによるグローバルセッションの制御方法は，HTTP セッションを URL 書き換えによって制御する場合の方法と同じです。J2EE サーバが Web アプリケーション内のサーブレットや JSP でリンク URL をエンコードするときに，グローバルセッション ID を含めてエンコードします。

Web クライアントからの 2 回目以降のリクエスト送信時には，グローバルセッション ID を含めてエンコードされた URL が使用されます。J2EE サーバはリクエストヘッダに含まれる URL の情報を基に，グローバルセッションの制御を実行します。

URL 書き換えによるグローバルセッションの制御の概要を次の図に示します。

図 6-15　URL 書き換えによるグローバルセッションの制御の概要



Web クライアントからのリクエストを受信すると，J2EE サーバの J2EE アプリケーション（Servlet／JSP）内で，HTTP セッションオブジェクトの作成とともに，グローバルセッション ID が発行されます。発行されたグローバルセッション ID は，Web クライアントに送信される HTTP レスポンスボディに設定されます。同時に，SFO サーバに，グローバルセッション情報とともに格納されます。

6.7.2　URL 書き換えによる制御の前提条件

URL 書き換えによる制御の前提条件について説明します。

URL 書き換えによってグローバルセッションを制御するためには，Web アプリケーションが URL 書き換えに対応している必要があります。Web アプリケーションが URL 書き換えに対応していない場合，この制御方法ではグローバルセッションを制御できません。

URL 書き換えに対応した Web アプリケーションの実装方法については，「6.11.2　URL 書き換えによる制御をする場合のアプリケーションの実装」を参照してください。

! 注意事項

URL 書き換えによるグローバルセッション情報の制御に対応したシステムは，Smart Composer 機能では構築できません。Smart Composer 機能で構築するシステムは，HTTP Cookie によるグローバルセッション情報の制御だけに対応しています。グローバルセッション情報の制御に URL 書き換えを使用するシステムは，運用管理ポータル，またはファイル編集とコマンド実行で構築してください。運用管理ポータルで構築する場合は，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「3.　J2EE アプリケーションを実行するシステムの構築と削除」を参照してください。ファイル編集とコマンド実行の場合は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」，およびマニュアル「アプリケーションサーバ リファレンス コマンド編」の「付録 B　J2EE アプリケーションを実行するシステムの構築と削除」を参照してください。

6.7.3　グローバルセッション ID の送信に使用する URL のパスパラメタ

ここでは，グローバルセッション ID の送信に使用する URL のパスパラメタについて説明します。

グローバルセッション ID は，URL のパスパラメタ名として設定されます。使用されるパスパラメタ名は，Web アプリケーション単位に設定できます。デフォルトの設定の場合，「gsessionid」になります。パスパラメタ名は，大文字と小文字を区別されます。設定方法については，「6.14　J2EE サーバの設定」を参照してください。

! 注意事項

HTTP セッションのセッション ID で使用する URL のパスパラメタ名は，大文字，小文字に関係なく，URL のパスパラメタ名に指定できません。これと同じ名称を使用すると，セッションフェイルオーバ用フィルタの初期化処理でエラーとなり，アプリケーションが開始できなくなります。

セッションフェイルオーバ用フィルタの初期化処理については，「6.9.3　J2EE アプリケーション開始時の処理」を参照してください。

なお，Web コンテナが自動で設定する URL のパスパラメタ名は「jsessionid」です。この名称はアプリケーションサーバのバージョンが 09-00 以降の場合，変更できます。URL のパスパラメタ名を変更した場合の注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.9 セッション管理の注意事項」を参照してください。

Web コンテナは，リクエストの URL にグローバルセッション ID を示すパスパラメタが含まれる場合，その ID をグローバルセッション ID として扱い，グローバルセッションを引き継ぎます。

ただし，次の条件のどれかに該当する場合は，URL のパスパラメタに含まれるグローバルセッション ID が無効になります。

URL のパスパラメタに含まれるグローバルセッション ID が無効になる場合

- リクエストの HTTP Cookie にグローバルセッション ID が含まれている場合
- リクエストの HTTP Cookie に HTTP セッションのセッション ID が含まれている場合
- リクエストの URL が HTTP セッションのセッション ID を表すパスパラメタを含んでいない場合

なお，Web アプリケーション内で URL を取得する ServletAPI を使用した場合，取得できる URL にはグローバルセッション ID を示すパスパラメタは含まれません。URL に含まれるパスパラメタがグローバルセッション ID として無効な場合も同様です。

リクエスト URL を取得する ServletAPI については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.3　セッションの管理方法」を参照してください。

6.7.4　URL 書き換えによって制御する場合の負荷分散

ここでは，URL 書き換えによってグローバルセッションを制御する場合に，負荷分散機で負荷分散を実現する方法について説明します。

HTTP Cookie を使用できない Web クライアントを使用する場合，負荷分散をするときにセッションを継続するための方法として，次のような方法があります。

- サーバが埋め込む識別情報によってセッションを継続する方法
- HTTP リクエストに含まれるクライアント個別の識別情報によってセッションを継続する方法

それぞれの方法について説明します。

(1)　サーバが埋め込む識別情報によるセッションの継続

Web サーバまたはアプリケーションサーバがコンテンツに埋め込む識別情報を使用して，振り分け先を決め，セッションを継続する方法です。負荷分散機には，この識別情報を基にリクエストの振り分け先のサーバを決めるように設定します。

サーバが埋め込む識別情報は，一般的に次のフィールドに設定されます。

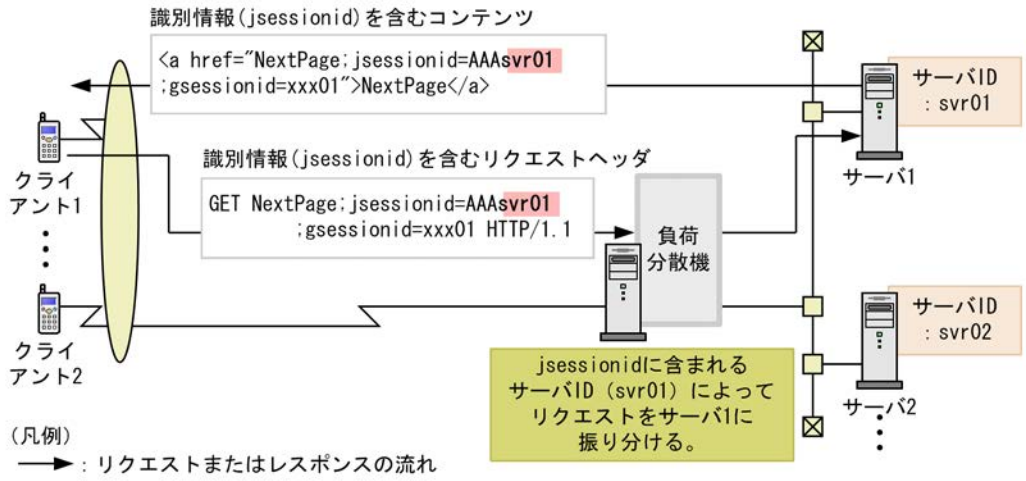
- URL のパスの一部
- URL のパスパラメタ
- URL のクエリストリング
- サーバから送信する HTML のフォームの hidden フィールド

メモリセッションフェイルオーバ機能を使用している場合，識別情報として，サーバ ID を使用できます。サーバ ID を識別情報として使用する場合，アプリケーションの実装を変更しなくても，負荷分散時のセッションの継続が実現できます。サーバ ID は，セッション ID の構成要素です。URL のパスパラメタに含まれています。サーバ ID については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.6　セッション ID および Cookie へのサーバ ID の付加」を参照してください。

なお，サーバ ID 以外を識別情報として使用する場合は，識別情報をコンテンツに埋め込む処理をアプリケーションで作りこむ必要があります。

サーバが埋め込む識別情報を使用して負荷分散を実行する処理の概要を次の図に示します。

図 6-16　サーバが埋め込む識別情報を使用した負荷分散の概要



クライアント 1 は，リクエストを送信するときに，サーバ 1 から送信された識別情報をリクエストヘッダに含めて送信します。負荷分散機は，この識別情報を基に，振り分け先のサーバを選択します。これによって，セッションを継続できます。

(2)　HTTP リクエストに含まれるクライアント個別の識別情報によるセッションの継続

Web クライアントが送信するクライアント個別の識別情報に基づいて，振り分け先を決め，セッションを継続する方法です。負荷分散機には，この識別情報を基にリクエストの振り分け先のサーバを決めるように設定します。

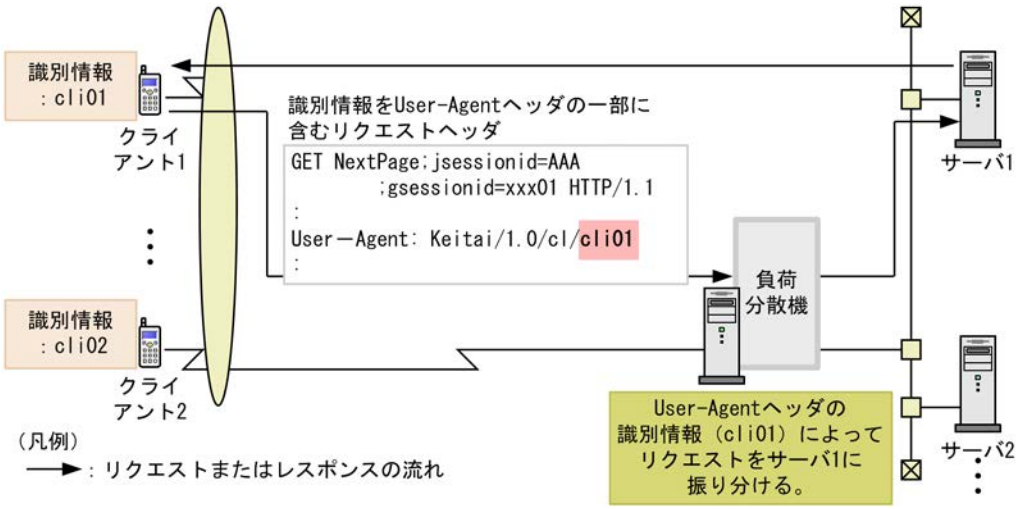
クライアント個別の識別情報は，一般的に次のフィールドに設定されます。

- HTTP リクエストの User-Agent ヘッダ
- クライアントが独自に定義した HTTP リクエストヘッダ

なお，この方法を使用する場合，クライアント側の設定で，クライアント個別の識別情報をリクエストと同時に送信するように設定しておく必要があります。

クライアントの個別識別情報を使用して負荷分散を実行する処理の概要を次の図に示します。

図 6-17　クライアントの個別識別情報を使用した負荷分散の概要



クライアント 1 は，リクエストを送信するときに，クライアント固有の識別情報を User-Agent ヘッダに含めて送信します。負荷分散機は，この識別情報を基に，振り分け先のサーバを選択します。これによって，セッションを継続できます。

6.7.5　リクエストラインの最大サイズの設定

URL 書き換えによってグローバルセッションを制御する場合の，リクエストラインの最大サイズの設定について説明します。

URL 書き換えによってグローバルセッションを制御する場合，URL のパスパラメタには，HTTP セッションのセッション ID とグローバルセッション ID が含まれます。HTTP リクエストのリクエストラインのサイズは，サーバ側およびクライアント側でそれぞれ上限があります。このため，あらかじめリクエストラインの最大サイズを算出して，それを基に，必要に応じてリクエストラインの最大サイズを設定してください。

ここでは，URL 書き換えによってグローバルセッションを制御する場合のリクエストラインのサイズの算出方法と，リクエストラインのサイズの上限値を設定する方法について説明します。

(1)　リクエストラインの文字列長の算出と最大サイズの設定

URL 書き換えによってグローバルセッションを制御する場合の，リクエストラインの最大サイズの設定について説明します。

URL 書き換えによってグローバルセッションを制御する場合，URL のパスパラメタには，HTTP セッションのセッション ID とグローバルセッション ID が含まれます。HTTP リクエストのリクエストラインのサイズは，サーバ側およびクライアント側でそれぞれ上限があります。

URL 書き換えによってグローバルセッションを制御する場合には，あらかじめリクエストラインの最大サイズを算出して，それを基にサーバおよびクライアントでのリクエストラインの最大サイズを設定しておく必要があります。

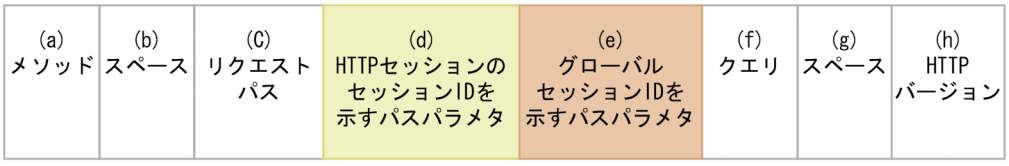
URL 書き換えを使用する場合のリクエストラインの文字列長の算出方法，およびリクエストラインの最大サイズの設定方法については，「6.7.5　リクエストラインの最大サイズの設定」を参照してください。

(2)　URL 書き換え使用時のリクエストラインのサイズの算出

リクエストライン全体のサイズは，リクエストラインを構成する各項目のサイズを基に算出してください。

URL 書き換えをする場合のリクエストラインの構成を次の図に示します。URL 書き換えで使用するパスパラメタは，(d)と(e)に挿入されます。

図 6-18　URL 書き換えをする場合のリクエストラインの構成



リクエストラインを構成する各項目について説明します。(a)～(h)は図中の文字と対応しています。

表 6-12　URL 書き換えをする場合のリクエストラインを構成する項目

図中の文字	項目	説明	例
(a)	メソッド	リクエストのメソッドです。	GET POST
(b)	スペース	区切りを示す空白文字です。 1 バイト固定です。	△
(c)	リクエストパス	宛先を示すパスです。	http://www.host1:1234/example/test.jsp/example/test.jsp
(d)	HTTP セッションのセッション ID を示すパスパラメタ	URL 書き換えによって URL に付加された，HTTP セッションのセッション ID を示すパスパラメタです。	;jsessionid=01234567aaaabbbbccccddddeeeeffffserver1
(e)	グローバルセッション ID を示すパスパラメタ	URL 書き換えによって URL に付加された。グローバルセッション ID を示すパスパラメタです。	;gsessionid=0123456789abcdef01234567aaaabbbbccccddddeeeeffffserver1
(f)	クエリ	リクエストに付加されたクエリ文字列です。	?a=b&x=y
(g)	スペース	区切りを示す空白文字です。 1 バイト固定です。	△
(h)	HTTP バージョン	HTTP のバージョンを示す文字列です。	HTTP/1.1

（凡例）

△：空白文字

なお，リクエストラインおよび URL の構成についての詳細は，RFC 2616 および RFC 2396 のドキュメントを参照してください。

表 6-12 の内容のうち，URL 書き換えによって付加される(d)と(e)のパスパラメタのサイズの算出方法について説明します。

(d)の HTTP セッションのセッション ID を示すパスパラメタのサイズの算出方法

```
HTTPセッションのセッションIDを示すパスパラメタのサイズ(バイト)
 = ";jsessionid=" ＋ <HTTPセッションのセッションID>
 =12バイト ＋ <HTTPセッションのセッションIDのサイズ(バイト)>※
```

注※

HTTP セッションのセッション ID の文字列長については，マニュアル「アプリケーションサーバ機能解説 基本・開発編(Web コンテナ)」の「2.7　セッション管理機能」を参照してください。また，URL のパスパラメタの名称を「jsessionid」から変更している場合は，次の値の合計がパスパラメタのサイズとなります。

- パスパラメタの文字数
- セミコロン（;），およびイコール（=）の文字数
- HTTP セッションのセッション ID の文字列長

(e)のグローバルセッション ID を示すパスパラメタのサイズの算出方法

```
グローバルセッションIDの送信に使用するパスパラメタの文字列長(バイト)
 = ";" ＋ <グローバルセッションIDの送信に使用するパスパラメタ名のサイズ>
   ＋ "=" ＋ <グローバルセッションID>
```

```
= 1バイト ＋ <GIDURLParamNameパラメタに設定した文字列のサイズ(バイト)>
  ＋ 1バイト ＋ <グローバルセッションIDのサイズ(バイト)>※
```

注※

グローバルセッション ID のサイズ（バイト）は，次の式で算出してください。

```
グローバルセッションIDのサイズ(バイト)
 = <HTTPセッションのセッションIDのサイズ(バイト)> ＋ 16バイト
```

(3)　リクエストラインの最大サイズの算出と設定

リクエストラインの最大サイズの設定方法について説明します。「(2)　URL 書き換え使用時のリクエストラインのサイズの算出」で算出した値を基に，最大サイズを設定してください。

設定方法は，使用する Web サーバによって異なります。

● Web サーバ連携で HTTP Server を使用する場合

HTTP Server を使用する場合は，HTTP Server の定義ファイル (httpsd.conf) の **LimitRequestLine ディレクティブ**で設定します。デフォルトの設定では，8,190 バイトです。

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

● Web サーバ連携で Microsoft IIS を使用する場合

Microsoft IIS を使用する場合のリクエストラインの最大サイズの設定方法については，Microsoft IIS のドキュメントを参照してください。

● インプロセス HTTP サーバを使用する場合

インプロセス HTTP サーバを使用する場合，リクエストラインの最大サイズは，簡易構築定義ファイルで設定します。

パラメタ名は，webserver.connector.inprocess_http.limit.max_request_line です。
webserver.connector.inprocess_http.limit.max_request_line の値には，リクエストラインの終端を示す改行文字である，(CR(0x0d)＋ LF(0x0A))の 2 バイトを含めた値を指定してください。デフォルトの設定では，8,190 バイトです。

簡易構築定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4.6　簡易構築定義ファイル」を参照してください。

6.7.6　URL 書き換えによって制御する場合の注意事項

ここでは，URL 書き換えによってグローバルセッションを制御する場合の注意事項について説明します。

URL 書き換えを実行する Servlet API を呼び出すと，URL のパスパラメタにはその時点で存在するグローバルセッションのグローバルセッション ID が付加されます。このため，次の場合には，セッションを正しく継続できません。

- セッションを作成する前に URL 書き換えを実行する Servlet API を呼び出した場合
- URL 書き換えを実行する Servlet API を呼び出したあとでセッションを再作成した場合

6.8　グローバルセッションの制御方法が混在する場合のシステム構成

この節では，グローバルセッションの制御方法が混在する場合のシステム構成について説明します。なお，グローバルセッションの制御方法が混在しない場合のシステム構成については，マニュアル「アプリケーションサーバ システム設計ガイド」の「3.10　セッション情報の引き継ぎを検討する」を参照してください。

グローバルセッションの制御方法が混在するシステムでは，Web アプリケーションを制御方法ごとに分けるかどうかによって，システム構成および設定内容が異なります。

! 注意事項

URL 書き換えによってセッションを制御する方法には，セッションハイジャックなどの脆弱性があるといわれています。URL 書き換えを使用する場合，脆弱性への対策は，それぞれのシステムで確実に実施するようにしてください。

対策としては，次のようなものが考えられます。

- HTTP Cookie を使用する Web アプリケーションでは，URL 書き換えを抑止して，セッション ID を URL に付加しないようにする。
- URL 書き換えを実行する Web アプリケーションでは，HTTP Cookie をサポートしている Web クライアントからのリクエストは処理しない。

なお，この節の説明は，URL 書き換えでセッションを制御する場合の脆弱性について理解し，適切な対応をしていることを前提にした上で，次のシステム構成について説明しています。

- HTTP Cookie を使用する Web アプリケーションと，URL 書き換えを使用する Web アプリケーションを，それぞれ専用の Web アプリケーションとして使用する場合のシステム構成
- HTTP Cookie を使用する Web アプリケーションと，URL 書き換えを使用する Web アプリケーションを，専用の Web アプリケーションにはしないで，両方の制御方法で使用する場合のシステム構成

6.8.1　制御方法ごとに Web アプリケーションを分ける場合

ここでは，グローバルセッションの制御方法ごとに Web アプリケーションを分ける場合のシステム構成について説明します。

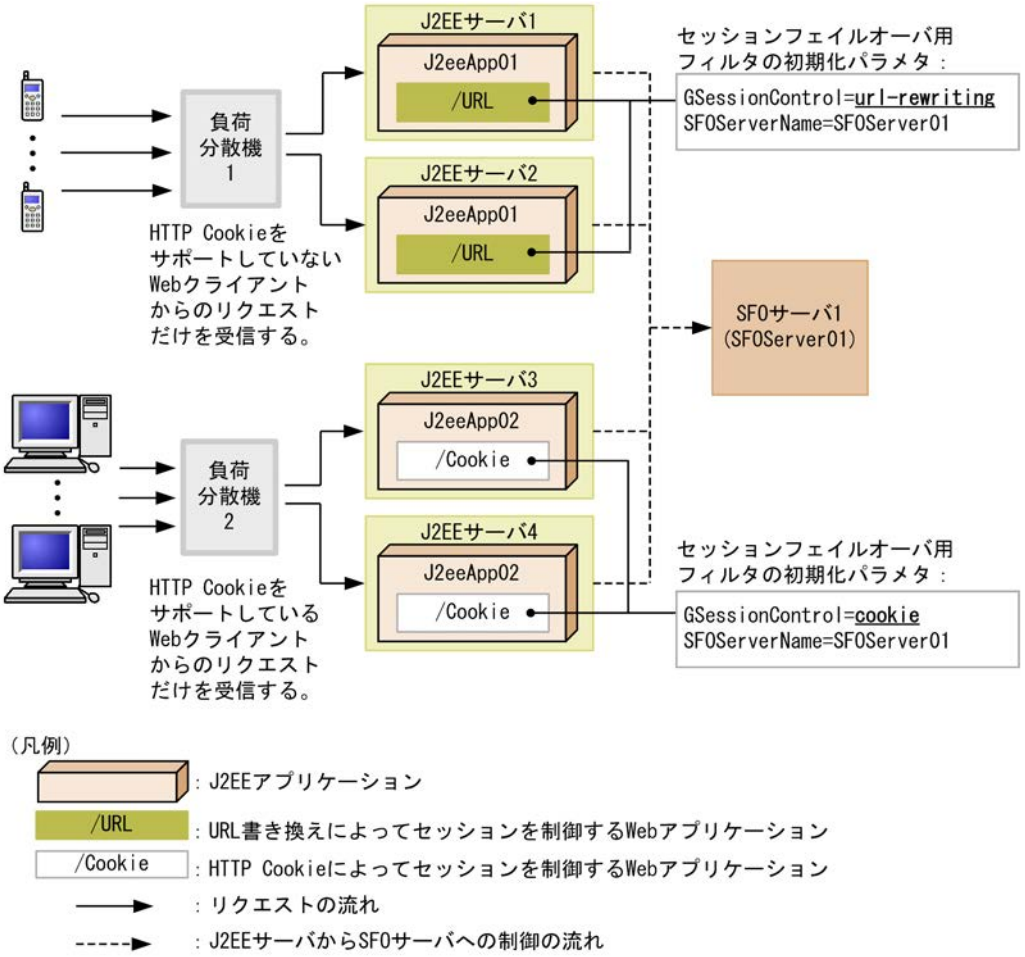
次の 2 種類のシステム構成について説明します。

- リクエストの受信ポートを，Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合
- リクエストの受信ポートを，Web クライアントが HTTP Cookie を使用できるかどうかによって分けない場合

(1)　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合

リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合のシステム構成の例を次に示します。

図 6-19　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合（制御方法を Web アプリケーションごとに分けるとき）



この例の場合は，HTTP Cookie を使用できない Web クライアントからのリクエストを受信する J2EE サーバと，HTTP Cookie を使用できる Web クライアントからのリクエストを受信する J2EE サーバを分けて配置しています。

J2EE サーバ 1 および J2EE サーバ 2 は，HTTP Cookie を使用できない Web クライアントからのリクエストを受信する J2EE サーバです。これらの J2EE サーバには，URL 書き換えによってセッションを制御する Web アプリケーションを配置します。

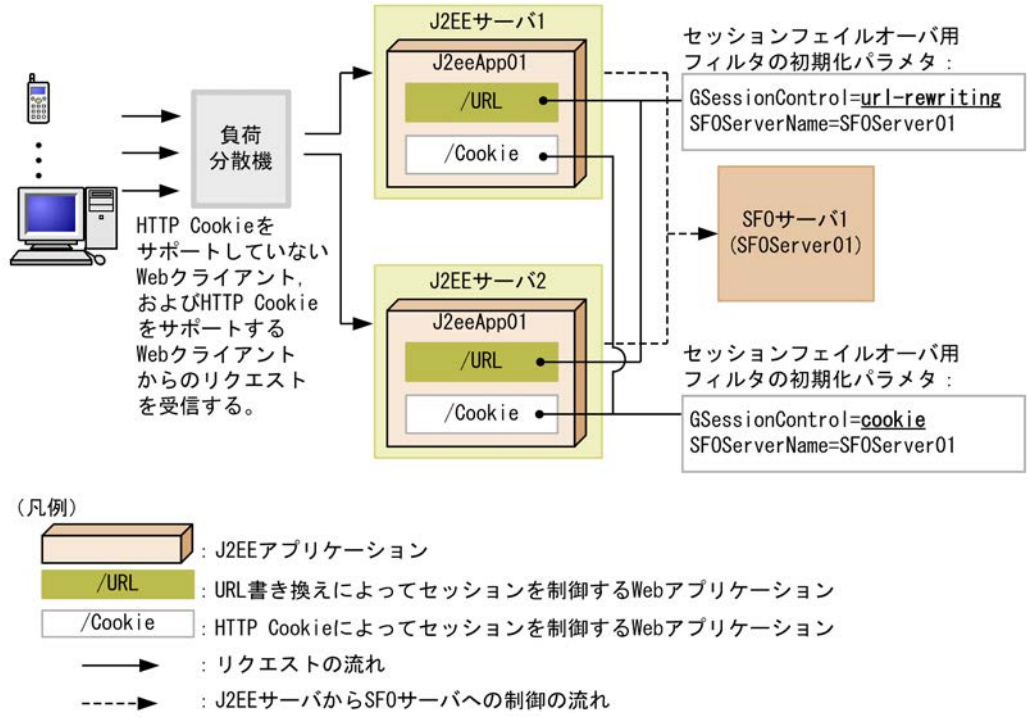
また，J2EE サーバ 3 および J2EE サーバ 4 は，HTTP Cookie を使用できる Web クライアントからのリクエストを受信する J2EE サーバです。これらの J2EE サーバには，HTTP Cookie を使用してセッションを制御する Web アプリケーションを配置します。

なお，グローバルセッションの制御方法に関係なく，J2EE サーバ 1 から J2EE サーバ 4 までで作成したすべてのグローバルセッション情報は，SFO サーバ 1（SFOServer01）で冗長化して管理します。

(2)　リクエストの受信ポートを HTTP Cookie の使用できるかどうかによって分けない場合

リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分けない場合のシステム構成の例を次に示します。

図 6-20　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分けない場合（制御方法を Web アプリケーションごとに分けるとき）



この例の場合は，HTTP Cookie を使用できない Web クライアントからのリクエストを受信する J2EE サーバと，HTTP Cookie を使用できる Web クライアントからのリクエストを受信する J2EE サーバを分けないで配置しています。

この場合は，URL 書き換えによってセッションを制御する Web アプリケーションと HTTP Cookie によってセッションを制御する Web アプリケーションの両方を，それぞれの J2EE サーバに配置する必要があります。これは，HTTP Cookie をサポートしていない Web クライアントと，HTTP Cookie をサポートしている Web クライアントの両方からのリクエストが，それぞれの J2EE サーバに対して送信されるためです。

なお，この場合，グローバルセッションの制御方法が異なる Web アプリケーションを一つの J2EE アプリケーションにまとめて配置しても，異なる J2EE アプリケーションとして配置してもかまいません。この例の場合は，同一の J2EE アプリケーション内に異なる制御方法の Web アプリケーションを配置しています。

グローバルセッションの制御方法は，Web アプリケーションそれぞれに対して，設定してください。

なお，グローバルセッションの制御方法に関係なく，J2EE サーバ 1 および J2EE サーバ 2 で作成したすべてのグローバルセッション情報は，SFO サーバ 1（SFOServer01）で冗長化して管理します。

6.8.2　制御方法ごとに Web アプリケーションを分けない場合

ここでは，グローバルセッションの制御方法ごとに Web アプリケーションを分けない場合のシステム構成として，次の 2 種類の構成について説明します。

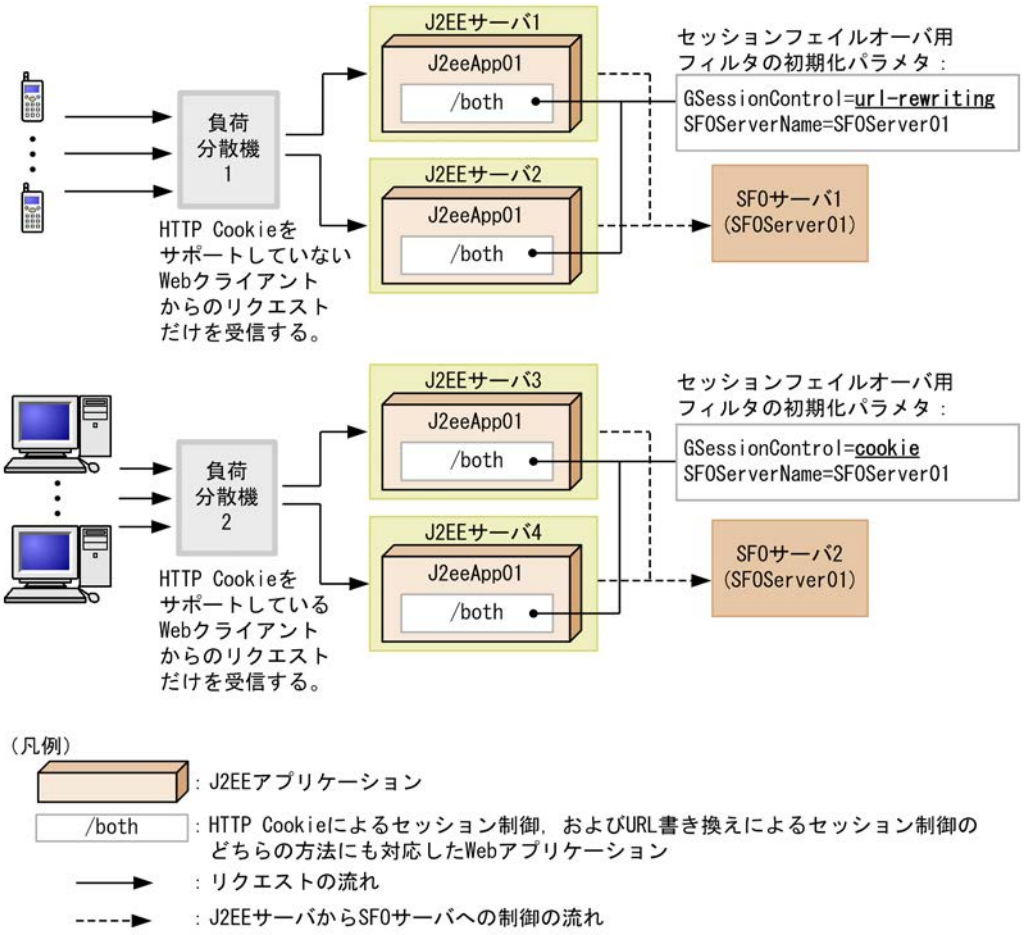
- リクエストの受信ポートを，Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合

- リクエストの受信ポートを，Web クライアントが HTTP Cookie を使用できるかどうかによって分けない場合

(1)　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合

グローバルセッションの制御方法に関係なく同一の Web アプリケーションを使用する場合に，リクエストが Web クライアントの HTTP Cookie サポート有無によって分かれているときの，システムの構成例を次に示します。

図 6-21　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分ける場合（制御方法を Web アプリケーションごとに分けないとき）



それぞれの J2EE サーバに配置された Web アプリケーションは，HTTP Cookie によるセッション制御，および URL 書き換えによるセッション制御の両方に対応しています。ただし，グローバルセッションの制御方法ごとにリクエストの受信ポートを分けているため，実際には Web アプリケーションごとに Web クライアントの種類が決まります。このため，それぞれの Web アプリケーションには，リクエストを送信する Web クライアントが HTTP Cookie を使用できるかどうかに応じて，グローバルセッションの制御方法を設定します。

図 6-21 の場合は，J2EE サーバ 1 および J2EE サーバ 2 が HTTP Cookie を使用できない Web クライアントからのリクエストを受け付けます。また，J2EE サーバ 3 と J2EE サーバ 4 が HTTP Cookie を使用できる Web クライアントからのリクエストを受け付けます。

ポイント

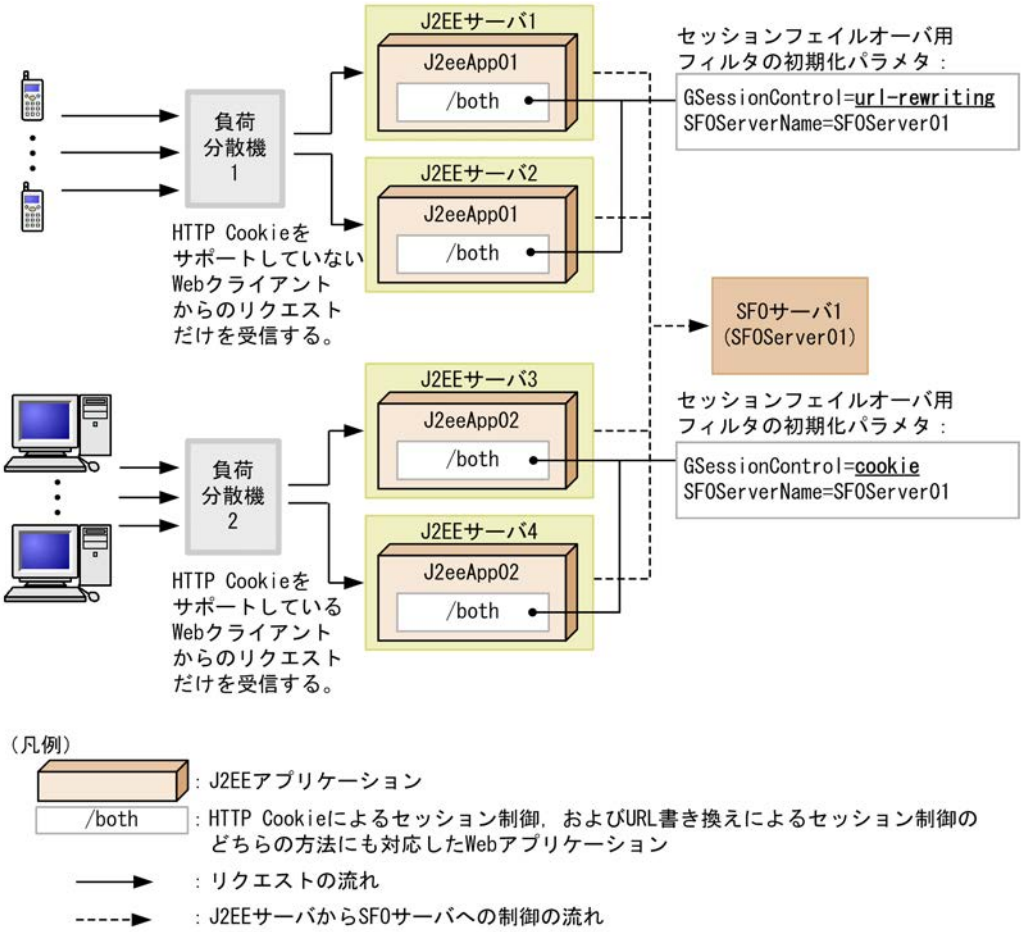
冗長化した同じ Web アプリケーションに対して，異なるグローバルセッションの制御方法を設定した場合，同一の SFO サーバ上でのネゴシエーションは失敗します。このため，この例では，SFO サーバを，グローバルセッションの制御方法ごとに二つのプロセスとしています。

どの SFO サーバを使用するかは，それぞれの J2EE サーバで設定します。接続先の SFO サーバの設定方法については，「6.14　J2EE サーバの設定」を参照してください。

また，同じ Web アプリケーションに対して異なるグローバルセッションの制御方法を混在させて管理する場合に，SFO サーバを 1 プロセスだけで運用したいときには，制御方法ごとに，J2EE アプリケーションの名称を変更する方法があります。メモリセッションフェイルオーバ機能では，J2EE アプリケーションのアプリケーションが一致するかどうかによって，Web アプリケーションを同じアプリケーションであると判断します。このため，J2EE アプリケーションの名称を変更することで，異なる Web アプリケーションだと判断され，ネゴシエーションが失敗しなくなります。

異なるグローバルセッションの制御方法を混在させて管理する場合に，SFO サーバを 1 プロセスだけで運用する場合のシステム構成例を次の図に示します。

図 6-22　異なるグローバルセッションの制御方法を混在させて管理する場合に，SFO サーバを 1 プロセスだけで運用する場合のシステム構成例

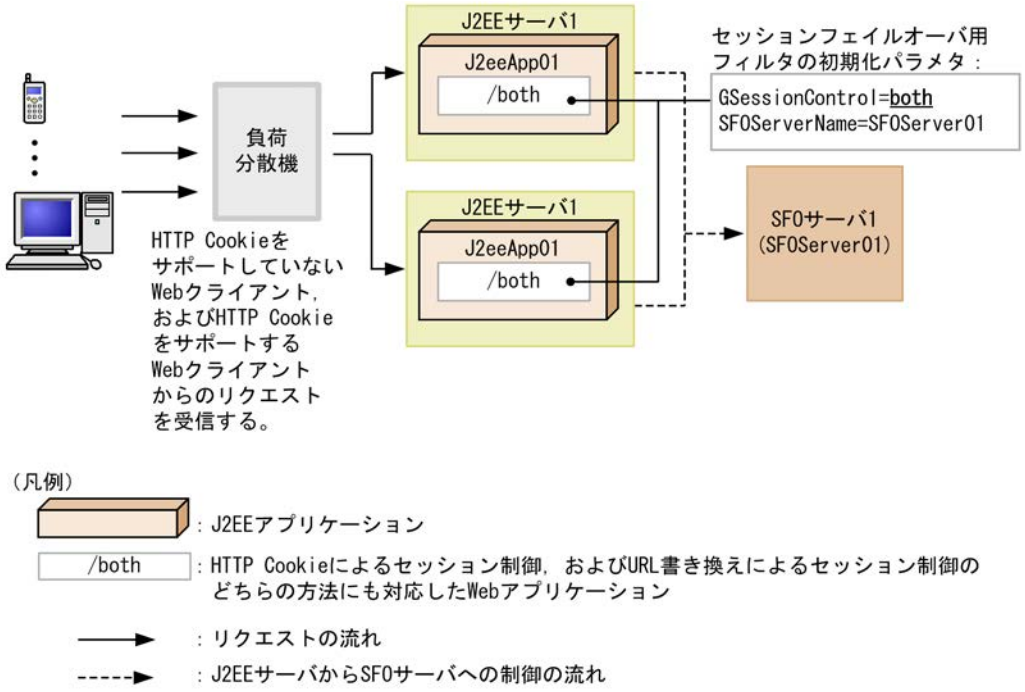


この例では，グローバルセッションの制御方法ごとに，J2EE アプリケーションの名称を変更しています。

(2)　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分けない場合

グローバルセッションの制御方法に関係なく同一の Web アプリケーションを使用する場合に，リクエストが Web クライアントの HTTP Cookie サポート有無によって分かれていないときの，システムの構成例を次に示します。

図 6-23　リクエストの受信ポートを Web クライアントが HTTP Cookie を使用できるかどうかによって分けない場合（制御方法を Web アプリケーションごとに分けないとき）



この例の場合は，リクエストの受信ポートがグローバルセッションの制御方法によって分かれていないため，それぞれの Web アプリケーションには，HTTP Cookie を使用したリクエストと URL 書き換えを利用したリクエストの両方が送信されます。これに対応するため，それぞれの Web アプリケーションでは，セッションの制御方法として「both」を設定します。

なお，グローバルセッションの制御方法に関係なく，J2EE サーバ 1 および J2EE サーバ 2 で作成したすべてのグローバルセッション情報は，SFO サーバ 1（SFOServer01）で冗長化して管理します。

6.9　メモリセッションフェイルオーバ機能で実施される処理

この節では，メモリセッションフェイルオーバ機能で実施される処理について説明します。

参考

この節の 6.9.1 から 6.9.6 までの説明は，グローバルセッションの制御方法として，HTTP Cookie を使用した場合の処理について説明しています。

それぞれの処理でのグローバルセッションの制御処理については，URL 書き換えを使用した場合も同様です。ただし，グローバルセッションの制御に URL 書き換えを使用する場合，HTTP Cookie を使用する場合と比べて，次の処理が異なります。

- グローバルセッションの制御に使用するグローバルセッション ID は，URL のパスパラメタから取得されます。
- グローバルセッションの制御に HTTP Cookie を使用しないため，グローバルセッションを削除した場合，およびグローバルセッションが無効な場合に，グローバルセッションの HTTP Cookie 情報の削除処理が実行されません。
- HTTP セッションの URL 書き換え処理が実行されるときに，HTTP セッションのセッション ID に加えて，グローバルセッション ID が付加されます。なお，HTTP セッションの URL 書き換え処理は，Web アプリケーションでの URL エンコード用の Servlet API の呼び出しを実行した場合に実行されます。

これらの相違点を含めた処理の流れについては，「6.9.7　URL 書き換えを使用する場合の処理の違い」を参照してください。

6.9.1　メモリセッションフェイルオーバ機能で処理が実施されるタイミング

メモリセッションフェイルオーバ機能では，次に示す時点でそれぞれ処理が実施されます。

- J2EE サーバ起動時の処理
  J2EE サーバ起動時には，SFO サーバの状態を監視するためのスレッドが作成されます。
- J2EE アプリケーション開始時の処理
  J2EE アプリケーション開始時には，セッションフェイルオーバ用フィルタの初期化処理が実施されます。
- リクエスト実行時の処理
  リクエスト処理実施時には，グローバルセッション情報の格納，更新，削除がされます。

メモリセッションフェイルオーバ機能で実施される処理について，以降の項で説明します。

また，HttpSession オブジェクトの有効期限が切れた場合の処理，無効なグローバルセッション ID が送られた場合の処理についても説明します。

6.9.2　J2EE サーバ起動時の処理

J2EE サーバでは，SFO サーバの状態を監視するためのスレッドを作成します。これを，**SFO サーバ監視スレッド**といいます。SFO サーバ監視スレッドは，Web アプリケーションごとに一つ作成され，各 Web アプリケーションで使用する SFO サーバのヘルスチェックを実施します。ヘルスチェックは，SFO サーバアプリケーションで用意している，コネクションの状態監視用のビジネスメソッドを呼び出して実施します。コネクションの状態監視のインターバルは 3 秒です。また，コネクションの状態監視用のビジネスメ

ソッドの呼び出しのタイムアウトには，SFO サーバとの通信タイムアウト時間に設定した時間が適用されます。タイムアウトが発生した場合，SFO サーバはダウンしていると判断され，リトライはされません。

なお，SFO サーバアプリケーションは，Application Server で提供しています。SFO サーバアプリケーションの概要については，「6.4.3(1)　SFO サーバ」を参照してください。また，SFO サーバ監視スレッドの設定については，「6.14　J2EE サーバの設定」を参照してください。

6.9.3　J2EE アプリケーション開始時の処理

ここでは，J2EE アプリケーション開始時の処理の流れについて説明します。また，J2EE アプリケーション開始時に実行される J2EE サーバと SFO サーバのネゴシエーション処理についても説明します。

(1)　J2EE アプリケーション開始時の処理の流れ

J2EE アプリケーション実行前に，セッションフェイルオーバ用フィルタの初期化処理が実施されます。初期化処理では，次の内容が実施されます。

1. **web.xml のメモリセッションフェイルオーバ機能に関する設定の妥当性がチェックされます。**

   設定内容に問題がある場合は，フィルタの初期化処理でエラーとなり，Web アプリケーションは開始されません。

2. **web.xml の設定に問題がない場合，J2EE サーバから SFO サーバへの接続が実施されます。**

   接続先の SFO サーバは，J2EE サーバごとに設定されています。設定方法については，「6.14　J2EE サーバの設定」を参照してください。

   なお，接続に失敗した場合は，次の点を確認してください。

   - SFO サーバが起動しているか
   - SFO サーバとの通信路に障害が発生していないか
   - J2EE サーバに設定されている，SFO サーバが登録されている CORBA ネーミングサービスの IP アドレスまたはポート番号に誤りがないか
   - SFO サーバとして使用する J2EE サーバの起動コマンドである"cjstartsv"コマンドの実行時に"-nosecurity"オプションが指定されているか

3. **SFO サーバとの接続に成功すると，SFO サーバとのネゴシエーションが実施されます。**

   web.xml で設定した内容に問題がない場合，SFO サーバとのネゴシエーションが実施されます。ネゴシエーション処理に使用される情報については，「(2)　J2EE サーバと SFO サーバのネゴシエーションで使用される情報」を参照してください。

   ネゴシエーションに失敗した場合の対処については，「(3)　ネゴシエーションが失敗した場合の対処」を参照してください。

4. **セッションフェイルオーバ用フィルタの初期化処理完了後，Web アプリケーションの開始時にメモリセッションフェイルオーバ機能の設定内容を示す KDJE34265-I のメッセージが出力されます。**

ポイント

グローバルセッションの制御に HTTP Cookie を使用する Web アプリケーションの場合，ネゴシエーションが成功すると，web.xml で設定したグローバルセッション ID が使用する HTTP Cookie 名称が，サーブレットコンテキストに属性として登録されます。属性の名称は，「com.hitachi.software.was.sfo.GIDCookieName」です。

このとき，Servlet 仕様で提供されるリスナ，および javax.servlet.ServletContextAttributeListener インタフェースを実装したリスナを Web アプリケーションに定義していた場合，attributeAdded メソッドの呼び出しがされます。

グローバルセッションの制御方法の設定，および HTTP Cookie の名称の設定方法については，「6.12　DD での定義」を参照してください。

● 注意事項

- SFO サーバの稼働状態に異常があったり，SFO サーバが稼働するマシンのメモリ使用量に異常があったりして，60 秒以内にネゴシエーション処理が完了しなかった場合，J2EE サーバは SFO サーバとのネゴシエーションに失敗したと判断して，SFO サーバとの接続を切断します。

  なお，SFO サーバ側でネゴシエーションが成功していても，J2EE サーバとの接続が切断されていることを検知した時点で，グローバルセッション情報の保持を終了します。

  なお，J2EE サーバの接続検知については，「6.4.3(1)　SFO サーバ」を参照してください。
- SFO サーバと J2EE サーバとのネゴシエーションが成功すると，SFO サーバには，Web アプリケーションの設定情報が登録されます。Web アプリケーションの情報を変更して，SFO サーバに接続すると，SFO サーバには変更前の Web アプリケーションの情報があるため，ネゴシエーションに失敗します。そのため，SFO サーバに登録された Web アプリケーションの設定情報を削除してから Web アプリケーションを開始する必要があります。なお，設定情報の削除については，「6.17　Web アプリケーションの設定情報の削除」を参照してください。
- SFO サーバが起動していない状態で J2EE アプリケーションを開始していた場合，または J2EE サーバとの接続中に SFO サーバを再起動した場合，J2EE サーバは，起動または再起動後の SFO サーバに接続します。SFO サーバとの接続完了後，アプリケーション開始と同様のネゴシエーションが実行されます。

  なお，このときにネゴシエーションに失敗した Web アプリケーションは，以降，SFO サーバへの接続はしません。この場合，メモリセッションフェイルオーバ機能が無効なまま処理が続行されます。

  メモリセッションフェイルオーバ機能を有効にするためには，J2EE アプリケーションを一度停止してください。該当する Web アプリケーションのメモリセッションフェイルオーバ機能に関する設定を正しくした上で，J2EE アプリケーションを再度開始してください。

(2)　J2EE サーバと SFO サーバのネゴシエーションで使用される情報

J2EE アプリケーションの開始処理の中で実施される，ネゴシエーションで使用される情報について説明します。

セッションフェイルオーバ用フィルタは，ネゴシエーション処理を実行するときに，Web アプリケーションの情報を SFO サーバに送信します。SFO サーバでは，送信された情報のうちネゴシエーションで使用する情報について，冗長化されているほかの J2EE サーバの Web アプリケーションの情報と比較します。これらの情報が，ほかの J2EE サーバの Web アプリケーションの情報と異なる場合，フィルタの初期化処理でエラーとなり，Web アプリケーションは開始されません。

ネゴシエーション処理で使用される情報として，セッションフェイルオーバ用フィルタによって SFO サーバに送信される情報を次の表に示します。また，その中でほかの J2EE サーバと同じ設定である必要がある情報も示します。

表 6-13　ネゴシエーション処理で使用される情報

セッションフェイルオーバ用フィルタによって SFO サーバに送信される情報	同じ設定にする情報
J2EE アプリケーション名	○
Web アプリケーションのコンテキストルート名	−
HttpSession 数の上限値	○

セッションフェイルオーバ用フィルタによって SFO サーバに送信される情報	同じ設定にする情報
セッション情報の最大サイズ	○
DD に定義された HTTP セッションの有効期間	○
グローバルセッションの制御方法	○
グローバルセッション ID の送信に使用する HTTP Cookie の名称	○※1
グローバルセッション ID の送信に使用する URL のパスパラメタ名	○※2
SFO サーバ内のロック待ちでのタイムアウト時間	○
書き換え可能グローバルセッション情報のキー名称	○
読み込み専用グローバルセッション情報のキー名称	○
J2EE サーバに設定された Http セッションのセッション ID へ付加するサーバ ID	－
byte 型配列であるセッション情報の引き継ぎ許可	○※3
直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可	○

（凡例）

○：ほかの J2EE サーバと同じ設定にする必要がある。

－：ほかの J2EE サーバと同じ設定でなくてもかまわない。

注※1

グローバルセッションの制御方法が HTTP Cookie を使用する方法の場合に同じ設定である必要があります。URL 書き換えを使用する場合，同じ設定でなくてもかまいません。

注※2

グローバルセッションの制御方法が URL 書き換えを使用する方法の場合に同じ設定である必要があります。HTTP Cookie を使用する場合，同じ設定でなくてもかまいません。

注※3

直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象とする場合は，同じ設定でなくてもかまいません。

また，セッションフェイルオーバ用フィルタによって SFO サーバに送信される情報のうち，HttpSession のセッション ID に付けられるサーバ ID は，J2EE サーバに対してそれぞれ一意となる値が設定されている必要があります。このため，すでに接続されている J2EE サーバに設定されたサーバ ID と同じ値である場合は，ネゴシエーションが失敗し，Web アプリケーションは開始されません。

(3)　ネゴシエーションが失敗した場合の対処

ネゴシエーションで使用する情報の不一致によってネゴシエーションに失敗した場合，J2EE サーバのメッセージログ，および SFO サーバのメッセージログにエラーメッセージが出力されます。エラーの原因となった個所の設定を見直し，再度 Web アプリケーションをデプロイしてください。

なお，ネゴシエーション中に，次に示す要因でエラーが発生した場合，ネゴシエーションは失敗しますが，メモリセッションフェイルオーバ機能が無効な状態で Web アプリケーションの処理は続行されます。

- SFO サーバのメモリが不足している
- ネゴシエーション処理のタイムアウトが発生した
- ネゴシエーション中に SFO サーバまたは通信路に障害が発生した

- 必要な機能を SFO サーバがサポートしていない（SFO サーバのバージョンが J2EE サーバのバージョンと合っていない）

エラーが発生した場合は，J2EE サーバおよび SFO サーバで出力されたメッセージログと SFO サーバの状態を確認して，エラーの原因を取り除いてください。

なお，エラー発生後，J2EE サーバは Web アプリケーションの処理を続行しながら，定期的に SFO サーバへの接続を試みます。J2EE サーバが SFO サーバに接続ができた場合，再度ネゴシエーション処理が実行されます。

ネゴシエーションが成功すると，メモリセッションフェイルオーバ機能が有効になります。

6.9.4　リクエスト実行時の処理

ここでは，リクエスト実行時のグローバルセッションフェイルオーバ用フィルタの処理と，グローバルセッション情報の格納，更新，削除処理について説明します。

(1)　リクエスト実行時のセッションフェイルオーバ用フィルタの処理

セッションフェイルオーバ用フィルタでは，リクエスト実行前には，グローバルセッション情報をロックし，必要に応じて引き継ぎ処理をします。また，リクエスト処理後には，グローバルセッション情報を格納または更新して，ロックを解除します。

(a)　グローバルセッション情報のロックについて

異なる J2EE サーバから同じグローバルセッション情報を同時に更新されることを抑止するため，セッションフェイルオーバ用フィルタでは，SFO サーバ上のグローバルセッション情報のロックをします。

1. J2EE アプリケーションでのリクエスト処理で，SFO サーバのグローバルセッション情報を使用すると，SFO サーバ上のグローバルセッション情報はロックされます。
   このとき，別の J2EE サーバがグローバルセッション情報を使用しようとすると，ロックの解除待ちとなります。
2. グローバルセッション情報を使用している J2EE アプリケーションでリクエスト処理が完了すると，SFO サーバでのロックが解除されます。
   ロック解除時に，ロックの解除待ちとなっている J2EE アプリケーションがある場合は，ロックが解除されたグローバルセッション情報を使用してリクエスト処理をします。

グローバルセッション情報にロックを設定することによって，J2EE アプリケーションでのダブルクリックなどによって二重送信されたリクエストが別の J2EE サーバに送信された場合の誤動作も防止できます。

なお，ロックの解除待ちには，タイムアウトを設定できます。解除待ちでタイムアウトした場合，該当するグローバルセッション情報は使用できなくなります。このとき，セッションは破棄され，新規リクエストとして J2EE アプリケーションを呼び出します。

(b)　セッション情報の引き継ぎ処理について

セッション情報の引き継ぎ処理では，新規に HttpSession オブジェクトを生成し，セッション情報を登録します。Servlet 仕様で提供されているセッションに関する次の三つのリスナを J2EE アプリケーションで定義している場合，リスナへの通知が実施されます。

呼び出されるリスナの種類について，システム内で発生するイベントごとに次の表に示します。

表 6-14　呼び出されるリスナの種類

システム内で発生するイベント	呼び出されるリスナ	HttpSession オブジェクト，およびグローバルセッションへの操作
ログイン	HttpSessionListener	• HttpSession オブジェクトの作成 • グローバルセッションの作成
J2EE アプリケーション内でのセッションの使用	HttpSessionAttributeListener HttpSessionBindingListener	• HttpSession オブジェクトへのセッション情報の格納 • HttpSession オブジェクトからのセッション情報の削除
ログアウト	HttpSessionListener HttpSessionAttributeListener HttpSessionBindingListener	• HttpSession オブジェクトの破棄 • グローバルセッションの破棄
障害発生によるグローバルセッション情報の引き継ぎ		• HttpSession オブジェクトの作成 • HttpSession オブジェクトへのグローバルセッション情報の格納
ほかのサーバに引き継がれた HttpSession オブジェクトの有効期限切れ		• HttpSession オブジェクトの削除

(2)　グローバルセッション情報の格納処理

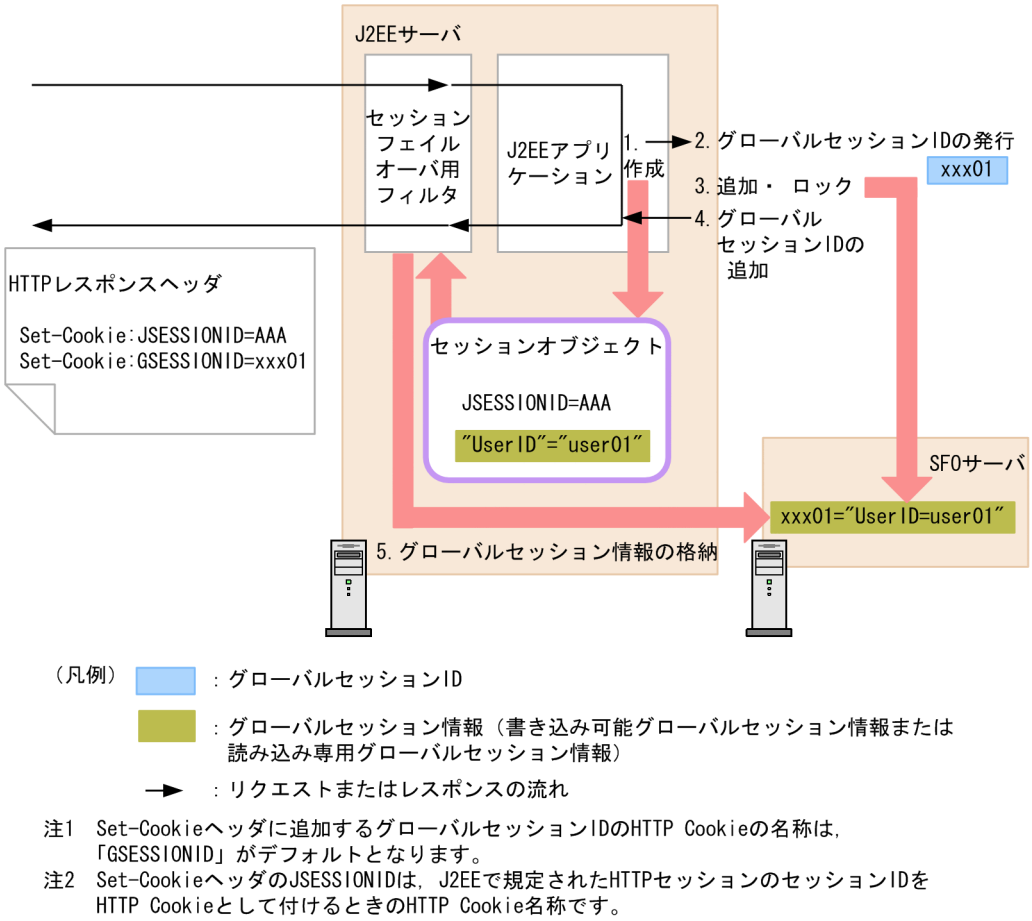
グローバルセッションは，HTTP セッションと同時に生成されます。グローバルセッションが生成されると，グローバルセッション情報も生成されます。このとき，HttpSession オブジェクトに対してセッション情報が未登録である場合も，グローバルセッション情報は生成されます。ただし，セッション情報が未登録であるという情報が生成されます。

J2EE アプリケーションで HttpSession オブジェクトの作成後，グローバルセッション情報が SFO サーバに格納されるまでの処理について，次の図に示します。

参考

次の図とその説明は，グローバルセッションの制御方法として，HTTP Cookie を使用している場合の例です。URL 書き換えを使用している場合の処理の違いについては，「6.9.7　URL 書き換えを使用する場合の処理の違い」を参照してください。

図 6-24　グローバルセッション情報の格納処理



注1　Set-Cookieヘッダに追加するグローバルセッションIDのHTTP Cookieの名称は，「GSESSIONID」がデフォルトとなります。
注2　Set-CookieヘッダのJSESSIONIDは，J2EEで規定されたHTTPセッションのセッションIDをHTTP Cookieとして付けるときのHTTP Cookie名称です。

1. J2EE アプリケーション内の処理で，HttpSession オブジェクトが作成されます。
   javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドが呼び出され，HttpSession オブジェクトが作成されます。
2. 作成された HttpSession オブジェクトに対応する，グローバルセッション ID が発行されます。
3. SFO サーバに新しいグローバルセッション情報を追加します。追加されたグローバルセッション情報はロックされます。
4. HTTP レスポンスのヘッダの Set-Cookie ヘッダに，グローバルセッション ID が追加されます。
5. SFO サーバにグローバルセッション情報を送信し，格納します。

なお，HTTP セッションで使用する Set-Cookie ヘッダの path 属性はコンテキストルートが固定となります。

(3)　グローバルセッション情報の更新処理

J2EE アプリケーションでセッションを使用し，グローバルセッション情報が更新されると，SFO サーバ内のグローバルセッション情報も更新する必要があります。

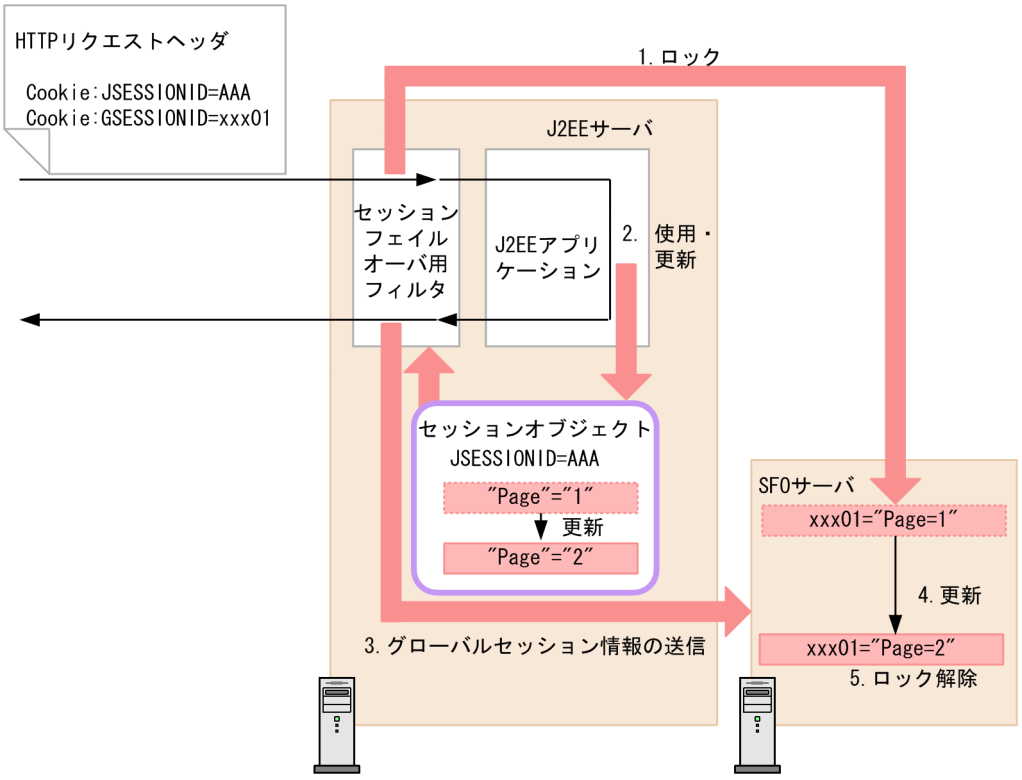
なお，グローバルセッション情報の更新中は，グローバルセッション情報にロックが掛かります。ロックが掛かると，ほかの J2EE サーバからのグローバルセッション情報の操作ができなくなります。

グローバルセッション情報の更新処理について，次の図に示します。

参考

次の図とその説明は，グローバルセッションの制御方法として，HTTP Cookie を使用している場合の例です。URL 書き換えを使用している場合の処理の違いについては，「6.9.7　URL 書き換えを使用する場合の処理の違い」を参照してください。

図 6-25　グローバルセッション情報の更新処理



注1　Cookieヘッダのグローバルセッション IDのHTTP Cookieの名称は，「GSESSIONID」がデフォルトとなります。
注2　CookieヘッダのJSESSIONIDは，J2EEで規定されたHTTPセッションのセッション IDをHTTP Cookieとして付けるときのHTTP Cookie名称です。

1. SFO サーバ上のグローバルセッション情報にロックが掛かります。
   J2EE アプリケーションの処理の前に，グローバルセッション情報にロックが掛かります。
2. J2EE アプリケーション内の処理で HttpSession オブジェクトを使用すると，HttpSession オブジェクトに登録されたグローバルセッション情報が更新されます。
3. J2EE アプリケーション処理の終了後，SFO サーバにセッション情報を送信します。
4. SFO サーバ上のグローバルセッション情報を最新に更新します。
5. SFO サーバ上のグローバルセッション情報のロックを解除します。

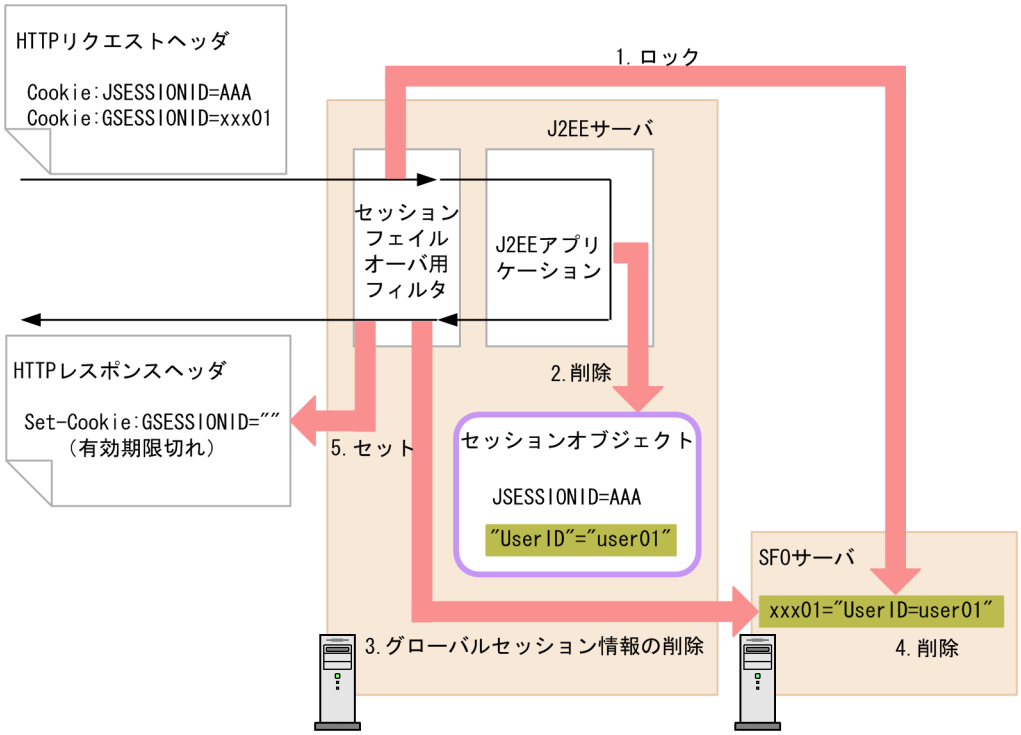
(4)　グローバルセッション情報の削除処理

J2EE アプリケーション内で，javax.servlet.http.HttpSession インタフェースの invalidate メソッドが呼び出される場合の，明示的なグローバルセッション情報の削除の処理について，次の図に示します。

参考

次の図とその説明は，グローバルセッションの制御方法として，HTTP Cookie を使用している場合の例です。URL 書き換えを使用している場合の処理の違いについては，「6.9.7　URL 書き換えを使用する場合の処理の違い」を参照してください。

図 6-26　グローバルセッション情報の明示的な削除



（凡例）　：グローバルセッション情報（書き込み可能グローバルセッション情報または読み込み専用グローバルセッション情報）

→　：リクエストまたはレスポンスの流れ

注1　CookieヘッダおよびSet-CookieヘッダのグローバルセッションIDのHTTP Cookieの名称は，「GSESSIONID」がデフォルトとなります。

注2　CookieヘッダのJSESSIONIDは，J2EEで規定されたHTTPセッションのセッションIDをHTTP Cookieとして付けるときのHTTP Cookie名称です。

1. SFO サーバ上のグローバルセッション情報にロックが掛かります。

   J2EE アプリケーションの処理の前に，グローバルセッション情報にロックが掛かります。

2. J2EE アプリケーション内の処理で，明示的に HttpSession オブジェクトを破棄します。

   javax.servlet.http.HttpSession インタフェースの invalidate メソッドを呼び出して，オブジェクトを破棄します。

3. J2EE アプリケーション処理の終了後，SFO サーバにグローバルセッション情報の削除要求を送信します。

4. SFO サーバ上のグローバルセッション情報が削除されます。

5. クライアント上にあるグローバルセッション ID の HTTP Cookie 情報を削除するように設定されます。

   HTTP レスポンスヘッダに，HTTP Cookie を期限切れでセットします。ただし，J2EE アプリケーションで HTTP レスポンスがすでにコミットされていた場合は，期限切れでのセットは実施されません。

クライアント上のグローバルセッション ID の HTTP Cookie 情報が削除されるのは，レスポンスがコミットされていなかった場合だけです。J2EE アプリケーションからセッションフェイルオーバ用フィルタに処理が戻った時点で，すでにレスポンスがコミットしている場合は，クライアント上のグローバルセッション ID の HTTP Cookie 情報は削除されません。この場合，次に送信するリクエストにグローバルセッション情報が Cookie として付けられた時点で，HTTP Cookie 情報の削除処理が実施されます。

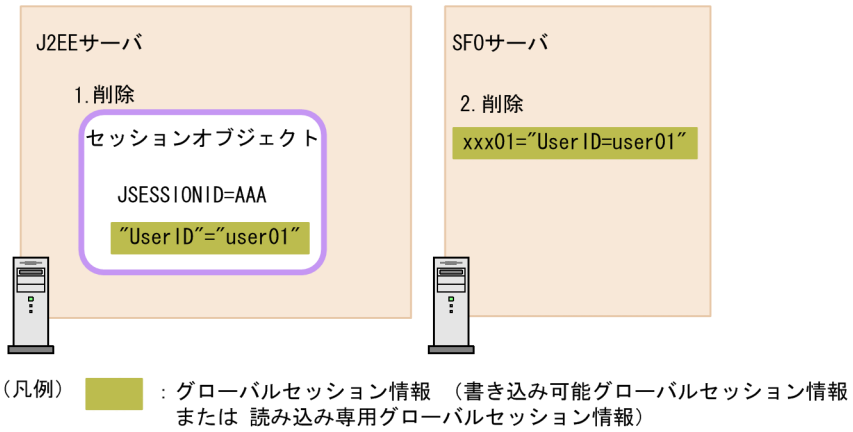
6.9.5　HttpSession オブジェクトの有効期限が切れた場合の処理

HttpSession オブジェクトに設定されている有効期限が切れるまでの間に，HttpSession オブジェクトを使用するリクエストが発生しなかった場合は，HttpSession オブジェクトは J2EE サーバによって削除されます。また，このとき，HttpSession オブジェクトに対応づけられた SFO サーバ上のグローバルセッション情報も削除されます。

有効期限が切れているときには，グローバルセッション情報がロックされているかどうかにかかわらず，HttpSession オブジェクトは削除されます。このため，J2EE サーバでグローバルセッション情報を使用中の場合や，J2EE サーバのダウンによってロックされたままのグローバルセッション情報も，有効期限が切れると削除されます。

HttpSession オブジェクトの有効期限が切れた場合の処理の流れについて，次の図に示します。

図 6-27　HttpSession オブジェクトの有効期限が切れた場合の処理



1. 有効期限が切れると，HttpSession オブジェクトは削除されます。

   J2EE サーバ内では，HttpSession オブジェクトの有効期限が監視されています。有効期限が切れると，HttpSession オブジェクトが削除されます。

2. 有効期限の切れた SFO サーバ上のグローバルセッション情報も削除されます。

   SFO サーバ内では，グローバルセッション情報の有効期限が監視されています。有効期限が切れると，SFO サーバ上のグローバルセッション情報が削除されます。

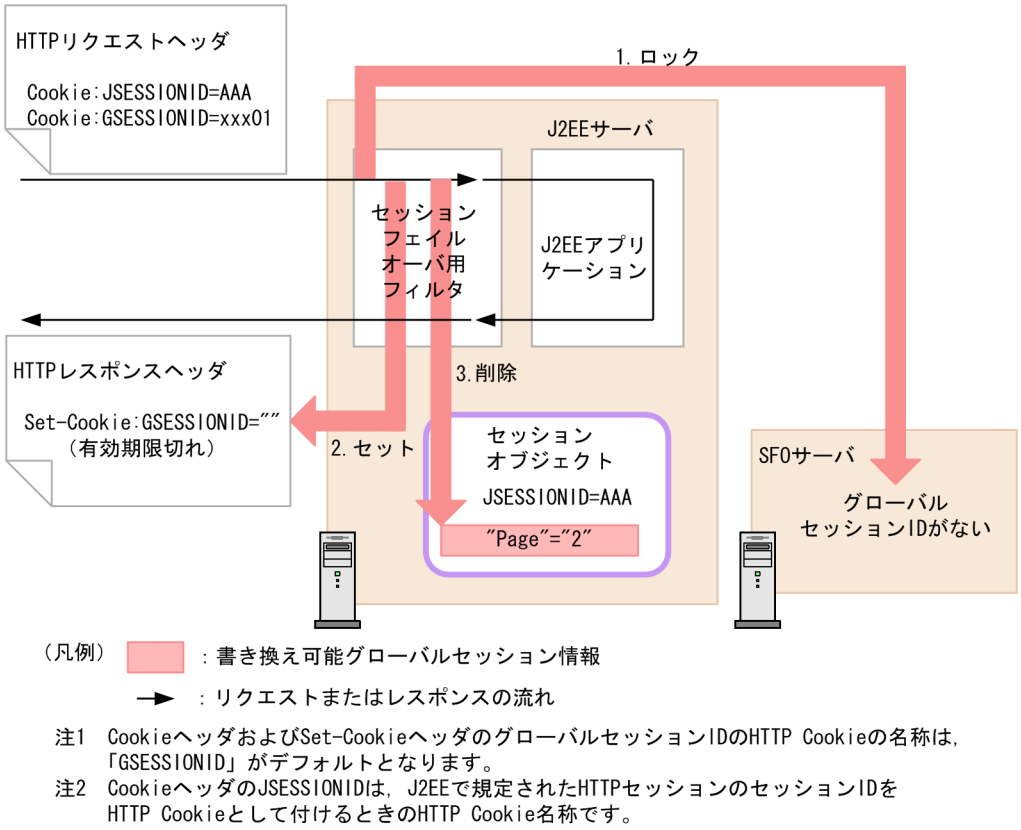
6.9.6　無効なグローバルセッション ID が送られた場合の処理

SFO サーバにない，グローバルセッション ID がリクエストとして送られてきた場合の処理の流れについて，次の図に示します。

参考

次の図とその説明は，グローバルセッションの制御方法として，HTTP Cookie を使用している場合の例です。URL 書き換えを使用している場合の処理の違いについては，「6.9.7　URL 書き換えを使用する場合の処理の違い」を参照してください。

図 6-28　無効なグローバルセッション ID が送られた場合の処理



注1　CookieヘッダおよびSet-Cookieヘッダのグローバルセッション IDのHTTP Cookieの名称は，「GSESSIONID」がデフォルトとなります。

注2　CookieヘッダのJSESSIONIDは，J2EEで規定されたHTTPセッションのセッション IDを HTTP Cookieとして付けるときのHTTP Cookie名称です。

1. グローバルセッション情報のロックが失敗します。

   SFO サーバに対して，グローバルセッション情報のロック要求がありますが，SFO サーバ内にない，無効なグローバルセッション ID が送信されてきたため，ロックが失敗します。

2. HTTP レスポンスヘッダに，Set-Cookie ヘッダを期限切れでセットします。

   クライアントにあるグローバルセッション ID の HTTP Cookie 情報を削除するよう，Set-Cookie ヘッダを期限切れでセットします。

3. HttpSession オブジェクトがある場合は，書き換え可能グローバルセッション情報が削除されます。

なお，3.の処理以降，J2EE アプリケーション内の処理で，javax.servlet.http.HttpServletRequest インタフェースの getSession メソッドが呼び出されると，グローバルセッションが新規に作成されます。

6.9.7　URL 書き換えを使用する場合の処理の違い

ここでは，URL 書き換えを使用する場合の処理と，HTTP Cookie を使用する場合の処理との違いについて説明します。

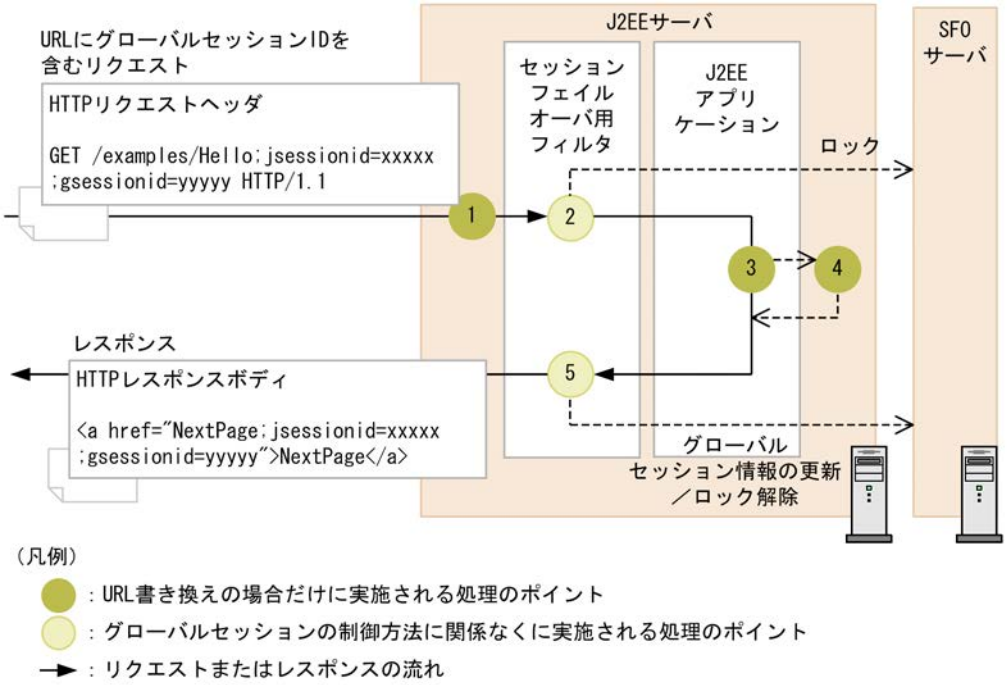
グローバルセッションの制御方法として URL 書き換えを使用している場合，HTTP Cookie を使用して制御している場合の処理に加えて，URL 書き換えのための処理として，次の処理が追加されます。

- URL のパスパラメタの解析
- Web アプリケーションでの Servlet API の呼び出しによる，グローバルセッション ID を含む URL のエンコード

また，グローバルセッションを削除した場合，およびグローバルセッションが無効な場合に，グローバルセッションの HTTP Cookie 情報の削除処理は実行されません。

URL 書き換えする場合に追加される処理について，次の図に示します。

図 6-29　URL 書き換えする場合に追加される処理



各ポイントでの処理内容について説明します。説明の番号は図中の番号と対応しています。なお，1.，3.，4.の処理が，URL 書き換えの場合だけに実行される処理です。

1. J2EE サーバによって，Web クライアントから送信されたリクエスト URL のパスパラメタが解析されます。
2. セッションフェイルオーバ用フィルタによって，SFO サーバ上のグローバルセッション情報がロックされます。

   なお，この処理は，グローバルセッションの制御方法に関係なく実行されます。
3. J2EE アプリケーション内の Web アプリケーションで，URL 書き換えのためのエンコードを実行する Servlet API が呼び出されます。

   この処理は，URL 書き換えの場合だけ実行されます。
4. J2EE サーバによって，グローバルセッション ID が URL にエンコードされます。なお，エンコードは HTTP セッションのセッション ID とともに実行されます。
5. セッションフェイルオーバ用フィルタによって，SFO サーバ上のグローバルセッション情報が更新され，グローバルセッション情報のロックが解除されます。

   なお，この処理は，グローバルセッションの制御方法に関係なく実行されます。

6.10　メモリセッションフェイルオーバ機能使用時の障害検知

メモリセッションフェイルオーバ機能を使用している場合に，Web サーバ，J2EE サーバ，および SFO サーバで発生した障害は，次に示す二つの方法で検知できます。

- Management Server の稼働状況監視による障害検知

  Management Server の稼働状況監視を使用して，Web サーバ，J2EE サーバ，および SFO サーバの稼働状況を監視できます。稼働状況の監視については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「3.　稼働情報の監視（稼働情報収集機能）」を参照してください。

- 負荷分散機のヘルスチェック機能による障害検知

  負荷分散機の機能を使用して，システム内の J2EE サーバに対してヘルスチェックを実施できます。ヘルスチェックで，J2EE サーバに問題を検知した場合，該当する J2EE サーバにリクエストを転送している Web サーバへの負荷分散を中止します。

  また，J2EE サーバへのチェックの際に Web サーバを経由するため，Web サーバでの障害検知もできます。

  SFO サーバで発生した障害については，SFO サーバおよび J2EE サーバが出力したメッセージログを基に，障害検知を実施します。

6.11　アプリケーションの実装

この節では，メモリセッションフェイルオーバ機能を使用するアプリケーションを開発するときに留意することについて説明します。また，URL 書き換えによる制御をする場合のアプリケーションの実装方法，およびアプリケーションを実行するときに追加される属性についても説明します。

6.11.1　実装時の留意事項

メモリセッションフェイルオーバ機能を使用するアプリケーションを開発する場合に，留意する点について説明します。

(1)　セッションフェイルオーバ用フィルタの呼び出し順序

セッションフェイルオーバ用フィルタは，サーブレットフィルタとして提供されます。セッションフェイルオーバ用フィルタは，ほかのサーブレットフィルタよりも前に呼び出される必要があります。HTTP レスポンス圧縮フィルタ，およびユーザ作成のフィルタよりも前に配置してください。フィルタの組み込みについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.9.2　フィルタ連鎖の推奨例」を参照してください。

(2)　JSP で暗黙的に作成される HTTP セッションの扱い

メモリセッションフェイルオーバ機能を有効にしたアプリケーションでは，セッションを作成すると必ずグローバルセッション ID が発行され，グローバルセッション情報を使用しないサーブレットまたは JSP であっても，「グローバルセッション情報が存在しない」という情報が SFO サーバに冗長化されます。

JSP では，デフォルトで HttpSession オブジェクトが作成されます。メモリの使用量や，SFO サーバとの通信による負荷を考慮し，セッションを必要としない JSP では明示的に HttpSession オブジェクトを作成しない設定にしてください。HttpSession オブジェクトを作成しない設定には，page ディレクティブの session 属性を使用してください。

(3)　セッション管理に使用する情報の選択

HTTP セッションのセッション管理には，グローバルセッション情報を使用するようにしてください。

メモリセッションフェイルオーバ機能では，障害発生時にグローバルセッション情報は引き継げますが，HTTP セッションは新しく作成されます。このため，HTTP セッションのセッション ID はグローバルセッション情報の引き継ぎの前後で異なります。したがって，HTTP セッションのセッション ID を基に制御を実行するアプリケーションは，グローバルセッション情報を基に制御を実行するように変更する必要があります。

(4)　障害発生時の動作を考慮した処理の実装

メモリセッションフェイルオーバ機能を使用するアプリケーションでは，障害発生時にグローバルセッション情報の引き継ぎ処理や，書き換え可能グローバルセッション情報の削除処理が発生します。

アプリケーションでは，これらを想定した処理を実装する必要があります。

(a)　セッション情報の引き継ぎを考慮した処理

J2EE サーバや Web サーバに障害が発生すると，リクエストが別の J2EE サーバに転送されて，セッション情報が引き継がれます。アプリケーションが引き継ぎ先の J2EE サーバで問題なく業務を続けるためには，必要な情報をすべてグローバルセッション情報として設定しておく必要があります。

グローバルセッション情報として設定できる情報については，「6.4.2　グローバルセッション情報」を参照してください。また，グローバルセッション情報の引き継ぎについては，「6.4.5　サーバの状態とグローバルセッション情報の引き継ぎ」を参照してください。

(b)　HTTP セッションに同一のオブジェクトが複数回登録されている場合を考慮した処理

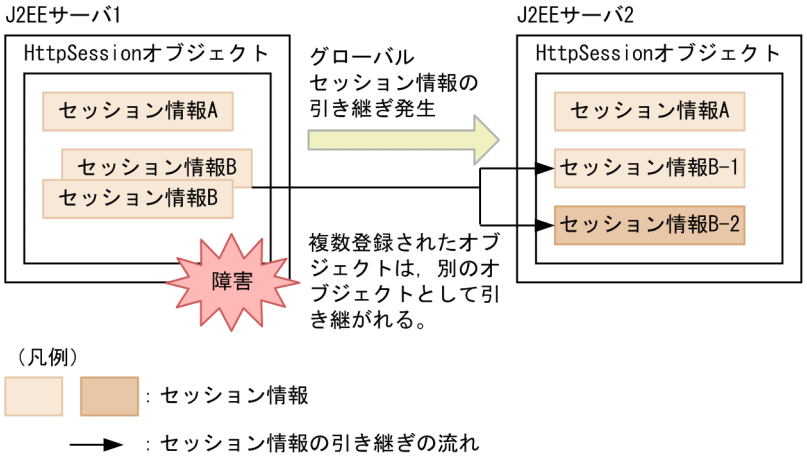
HttpSession オブジェクトに同じオブジェクトが複数登録されていた場合に，引き継ぎ後の HttpSession オブジェクトに登録されるオブジェクトが異なるので，注意が必要です。

- **一つの HttpSession オブジェクトに同じオブジェクトが複数登録されている場合**

  一つの HttpSession オブジェクトに，同じ java.lang.String クラスのオブジェクトが複数登録されている場合に，グローバルセッション情報の引き継ぎが発生すると，複数登録していたオブジェクトは別々のオブジェクトとして生成されます。

  一つの HttpSession オブジェクトに同じオブジェクトが複数登録されている場合の引き継ぎの例を次の図に示します。

  図 6-30　一つの HttpSession オブジェクトに同じオブジェクトが複数登録されている場合の引き継ぎの例



  この図では，J2EE サーバ 1 上の HttpSession オブジェクトに，java.lang.String クラスのオブジェクトであるセッション情報 B が複数登録されています。J2EE サーバ 1 で障害が発生して，J2EE サーバ 2 へセッション情報が引き継がれた場合，J2EE サーバ 2 上の HttpSession オブジェクトには，複数登録されていたセッション情報 B が，それぞれ別のセッション情報 B-1 とセッション情報 B-2 として生成されます。セッション情報 B-1 およびセッション情報 B-2 は，インスタンスは異なりますが，内容は同じです。

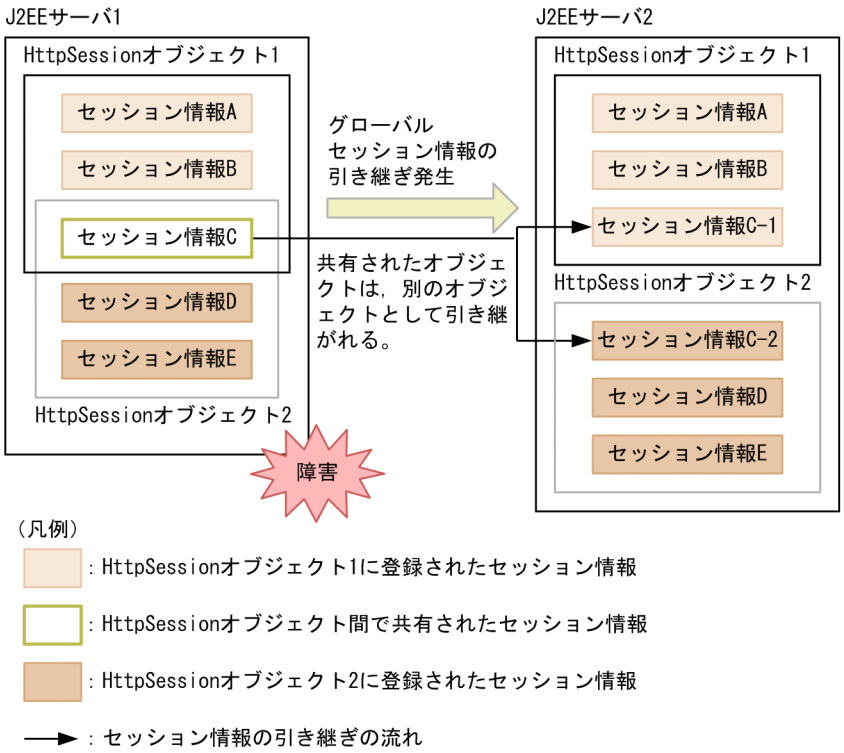
- **異なる HTTP セッションに同一のオブジェクトが登録されている場合**

  グローバルセッション情報は HTTP セッション単位に作成されます。直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象に設定している場合も，HTTP セッションごとに引き継ぎます。

  異なる HttpSession オブジェクトで同一のオブジェクトをセッション情報として共有している場合に，グローバルセッション情報の引き継ぎが発生したときは，オブジェクトが共有されません。別々のオブジェクトとして生成されます。

  異なる HttpSession オブジェクトに同一のオブジェクトが登録されている場合の引き継ぎの例を次の図に示します。

図 6-31　異なる HttpSession オブジェクトに同一のオブジェクトが登録されている場合の引き継ぎの例



この図では，J2EE サーバ 1 上の HttpSession オブジェクト 1 および HttpSession オブジェクト 2 で，同一のオブジェクトのセッション情報 C を共有しています。J2EE サーバ 1 で障害が発生して，J2EE サーバ 2 へセッション情報が引き継がれた場合，J2EE サーバ 2 上の HttpSession オブジェクト 1 および HttpSession オブジェクト 2 には，共有されていたセッション情報 C が，それぞれ別のセッション情報 C-1 とセッション情報 C-2 として生成されます。セッション情報 C-1 およびセッション情報 C-2 は，インスタンスは異なりますが，内容は同じです。

(c)　書き換え可能グローバルセッション情報の削除を考慮した処理

SFO サーバに障害が発生した場合など，書き換え可能グローバルセッション情報の削除が発生したとき，HttpSession オブジェクトには読み込み専用グローバルセッション情報とグローバルセッション情報でないセッション情報が残ります。

リクエスト処理を実行する前に，書き換えグローバルセッション情報が削除されていないかどうかを確認する処理を実行して，削除されている場合には適切な処理を実行してください。

例えば，グローバルセッション情報が次のように設定されている場合があります。

- 読み込み専用グローバルセッション情報にログイン情報が設定されている。
- 書き換え可能グローバルセッション情報に作業中の情報が設定されている。

この場合，書き換え可能グローバルセッション情報が削除されると，HttpSession オブジェクトに登録されたセッション情報の内容はログイン直後の状態に戻ります。この状態から業務を再開するためには，オブジェクトを再作成する処理や，javax.servlet.http.HttpServletResponse インタフェースの sendRedirect メソッドを使用してログイン直後のページに戻す処理などが必要になります。

J2EE アプリケーション内での書き換え可能グローバルセッション情報が削除されているかどうかは，HttpSession オブジェクトに登録した書き換え可能グローバルセッション情報があるかどうかで確認できます。確認した結果，あるはずの書き換え可能グローバルセッション情報がなくなっていた場合，書き換え可能グローバルセッション情報が削除されたと判断できます。

(5)　HTTP セッションの破棄に伴う動作

HTTP セッションを破棄するアプリケーション内でレスポンスのコミットを実行しても，クライアント上のグローバルセッション ID の HTTP Cookie 情報は削除されません。この場合，コミット後にリクエストを送信するとき，グローバルセッション ID がリクエストヘッダの Cookie として付加されて，J2EE サーバ，SFO サーバのメッセージログに次のログメッセージが出力されます。

- J2EE サーバで出力されるメッセージ
  KDJE34234-W（出力レベル：Warning）
- SFO サーバで出力されるメッセージ
  KDJE34017-W（出力レベル：Warning）

ただし，これらのメッセージが出力された場合も，J2EE サーバ，SFO サーバでの実際のリクエスト処理，およびメモリセッションフェイルオーバ機能への影響はありません。なお，クライアント上のグローバルセッション ID の HTTP Cookie 情報を削除する処理は，この時点で実行されます。以降のリクエスト処理ではこの HTTP Cookie は付加されません。

また，javax.servlet.http.HttpServletResponse インタフェースの sendRedirect メソッドを使用した場合も，レスポンスがコミットされて，リダイレクト先アドレスへのアクセス時に KDJE34234-W のメッセージが出力されます。このとき，リダイレクト先アドレスが FORM 認証を使用しているページへのアドレスであった場合，J2EE サーバのメッセージログには KDJE34234-W のメッセージではなく，KDJE34229-W のメッセージ（出力レベル：Warning）が出力されます。ただし，この場合も，認証機能およびメモリセッションフェイルオーバ機能への影響はありません。

(6)　セッション情報の引き継ぎが発生した場合の認証情報の扱い

障害発生時に認証情報を引き継ぐ場合は，Basic 認証によって認証処理をする必要があります。Basic 認証を使用すると，認証された J2EE サーバと異なる J2EE サーバにリクエストが振り分けられた場合でも，再度認証する必要はありません。グローバルセッション情報は正常に引き継がれ，業務を続けられます。

サーブレット API で定義されているフォーム認証によって認証処理をしている場合，認証された J2EE サーバと異なる J2EE サーバにリクエストが振り分けられた場合は，認証情報が引き継がれていても，再度認証が必要です。

Basic 認証および Form 認証については，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「6.2　DD の設定による Web コンテナのユーザ認証」を参照してください。

(7)　メモリセッションフェイルオーバ機能使用によるサーブレット API の動作

メモリセッションフェイルオーバ機能を使用する場合と，使用しない場合とで動作が変わるサーブレット API があります。次に，メモリセッションフェイルオーバ機能によって動作に変更がある API および注意点を示します。

表 6-15　メモリセッションフェイルオーバ機能と関係するサーブレット API

項番	クラス名またはインタフェース名※1	API 名	注意点
1	ServletContext	getAttributeNames	HTTP Cookie によるグローバルセッション制御が有効な場合に動作が変わります。グローバルセッション ID に使用する HTTP Cookie 名称をサーブレットコンテキストに属性として登録します。グローバルセッション ID は，セッションフェイルオーバ用フィルタ開始時に初期化パラメタに設定された ID です。getAttributeNames で取得できる属性名にこの属性の名称が含まれます。セッションフェイルオーバ用フィルタが登録する属性については，「6.11.3　アプリケーションを実行するときに追加される属性」を参照してください。
2	ServletContextAttributeListener	attributeAdded	HTTP Cookie によるグローバルセッション制御が有効な場合に動作が変わります。グローバルセッション ID に使用する HTTP Cookie 名称をサーブレットコンテキストに属性として登録するために，javax.servlet.ServletContextAttributeListener クラスの attributeAdded メソッドが呼び出されます。グローバルセッション ID は，セッションフェイルオーバ用フィルタ開始時に初期化パラメタに設定された ID です。サーブレットコンテキストに登録する属性については，「6.11.3　アプリケーションを実行するときに追加される属性」を参照してください。
3	HttpServletRequest	getCookies	HTTP Cookie によるグローバルセッション制御が有効な場合に動作が変わります。グローバルセッションが発行されると，グローバルセッション ID が Cookie ヘッダにセットされます。HTTP Cookie の名称は，GIDCookieName パラメタに指定した値です。
4		getHeader	
5		getHeaderNames	
6		getHeaders	
7		getSession	HTTP セッション作成時に，J2EE サーバの HTTP セッション数に余裕がある場合でも，SFO サーバでグローバルセッション数が上限に達すると，IllegalStateException 例外が発生します。
8	HttpServletResponse	encodeRedirectURL	URL 書き換えによるグローバルセッション制御が有効な場合に動作が変わります。グローバルセッション ID を URL に含めてエンコードします。グローバルセッション ID は，HTTP セッション ID とともにエンコードされます。この処理は必要に応じて実行されます。
9		encodeURL	
10		encodeRedirectUrl※2	
11		encodeUrl※2	
12	HttpSession	getAttribute	書き換え可能グローバルセッション情報の削除処理が発生すると，Web コンテナによって属性が追加されます。Web コンテナが追加する属性については，「6.11.3　アプリケーションを実行するときに追加される属性」を参照してください。
13		getAttributeNames	
14		getCreationTime	引き継ぎが発生すると，引き継ぎ処理によって HttpSession オブジェクトが作成されるため，引き継ぎ発生前と異なる値が返されます。
15		getId※3	

項番	クラス名またはインタフェース名※1	API 名	注意点
16	HttpSession	getLastAccessedTime	引き継ぎ発生後，初回のリクエスト時には前回アクセスの時刻を返すことができません。
17		isNew※4	引き継ぎによって，HttpSession オブジェクトが作成されると，戻り値が「true」となります。
18	HttpSessionAttributeListener	リスナについては，「6.9.4　リクエスト実行時の処理」を参照してください。	
19	HttpSessionBindingListener		
20	HttpSessionListener		
21	HttpServletRequestWrapper	動作については，javax.servlet.http.HttpServletRequest インタフェースの動作を参考にしてください。	

注※1

クラス名およびインタフェース名は，「javax.servlet.http.」を省略しています。

注※2

Servlet 2.1 仕様で非推奨とされている API です。

注※3

HTTP セッションのセッション ID を基に制御が行われる Web アプリケーションは，グローバルセッション情報を基に制御が行われるように変更する必要があります。

注※4

isNew メソッドの戻り値によって制御が行われる Web アプリケーションでは，HttpSession オブジェクトを新規に作成した場合，必ず登録するセッション情報を設けてください。そのセッション情報を読み込み専用グローバルセッション情報とし，isNew メソッドを使用する代わりにそのセッションが HttpSession オブジェクトに登録されているかどうかで新規セッションかどうかを判断するようにしてください。

(8)　JSP での<jsp:useBean>タグの使用

JSP で<jsp:useBean>タグを使用する時，scope 属性を"session"とすることで，JavaBeans が HttpSession オブジェクトの属性として登録されます。

<jsp:useBean>タグの scope 属性を"session"とする場合には，次の設定が必要になります。

- 引き継ぎ対象とするセッション情報のデータ型に，直列化可能クラスのオブジェクトを設定する。
- <jsp:useBean>タグの id タグに設定した名称が引き継ぎ対象となるように，id タグに設定した名称をグローバルセッション情報として使用する情報（キー）に設定する。

6.11.2　URL 書き換えによる制御をする場合のアプリケーションの実装

URL 書き換えによってグローバルセッションを制御する場合の，Web アプリケーションの実装方法について説明します。

(1)　使用する Servlet API

Web アプリケーションを実装する際には，次に示す Servlet API を使用して実装してください。

なお，グローバルセッション ID のエンコードは，グローバルセッションが存在している場合に，Web アプリケーション内でこれらの Servlet API が呼び出され，HTTP セッションのセッション ID のエンコードが必要と判断されたときにだけ実行されます。

- サーブレットや JSP 内で同一アプリケーションへのリンク先 URL をレスポンスに出力する場合，次の API を使用してリンク先 URL を出力してください。
  - javax.servlet.http.HttpServletResponse インタフェースの encodeURL(java.lang.String url)メソッド
  - javax.servlet.http.HttpServletResponse インタフェースの encodeUrl(java.lang.String url)メソッド

  ただし，encodeUrl(java.lang.String url)メソッドは非推奨の API です。
- サーブレットや JSP 内で javax.servlet.http.HttpServletResponse インタフェースの sendRedirect()メソッドによって同一アプリケーションへのリダイレクトを行う場合は，リダイレクト先の URL を次の API を使用して取得してください。
  - javax.servlet.http.HttpServletResponse インタフェースの encodeRedirectURL(java.lang.String url)メソッド
  - javax.servlet.http.HttpServletResponse インタフェースの encodeRedirectUrl(java.lang.String url)メソッド

  ただし，encodeRedirectUrl(java.lang.String url)メソッドは非推奨の API です。

これらの ServletAPI を使用しないで Web アプリケーションを実装している場合，グローバルセッション ID の URL 書き換えは実行されません。この場合，HTTP セッションおよびグローバルセッションは継続されません。

(2)　JSP の記述例

URL 書き換えによってグローバルセッションを制御する場合の JSP の記述例，および出力されるレスポンスの内容を示します。

JSP の記述内容

```
<a href="<%= response.encodeURL("nextPage.jsp") %>">link</a>
```

なお，この例は，次の前提に従っています。

- HTTP セッションのセッション ID は"xxxxx"である。
- グローバルセッション ID は"yyyyy"である。
- グローバルセッション ID の送信に使用する URL のパスパラメタ名は"gsessionid"である。

出力されるレスポンスの内容

```
<a href="nextPage.jsp;jsessionid=xxxxx;gsessionid=yyyyy">link</a>
```

! 注意事項

次に示すように，URL 書き換えを使用しないでパスパラメタを設定するアプリケーションの場合，グローバルセッションの URL 書き換えが実行されません。この場合，メモリセッションフェイルオーバ機能が有効になりません。

URL 書き換えによるグローバルセッションの制御が有効にならない例（JSP の場合）

```
<a href="nextPage.jsp;jsessionid=<%=session.getId()%>">link</a>
```

6.11.3　アプリケーションを実行するときに追加される属性

メモリセッションフェイルオーバ機能を使用するアプリケーションでは，動作するために必要な情報をWebコンテナから取得するために，javax.servlet.ServletContextオブジェクトの属性を使用します。

アプリケーションの起動時には，次の表に示す属性が，Webコンテナによってjavax.servlet.ServletContextオブジェクトの属性に追加されます。

表6-16　Webコンテナによって ServletContextの属性に追加される属性

キー名称	値の説明
com.hitachi.software.web.application-dd.J2eeApplicationName	J2EEアプリケーション名
com.hitachi.software.web.application-dd.contextRoot	コンテキストルート名
com.hitachi.software.web.dd.session-max-inactive-interval	DD（web.xml）に設定されたHTTPセッションの有効期間※1
com.hitachi.software.web.hitachi-runtime-dd.http-session-max-number	HTTPセッション数の上限値設定機能で設定されたHTTPセッション数の上限値
com.hitachi.software.web.dd.filter-class	DD（web.xml）の最上部に定義されたフィルタマッピングのクラス名
com.hitachi.software.web.dd.url-pattern	DD（web.xml）の最上部に定義されたフィルタマッピングのURLパターン※2※3

注※1

DD（web.xml）にHTTPセッションの有効期間を設定しなかった場合は，デフォルトの値が登録されます。

注※2

<url-pattern>タグでなく，<servlet-name>タグが設定された場合は追加されません。

注※3

DD（web.xml）内にセッションフェイルオーバ用フィルタを使用するフィルタマッピングの定義が複数あった場合，URLパターンの代わりに「<multi mapped>」の文字列が登録されます。

これらの属性の追加はメモリセッションフェイルオーバ機能の有効，無効に関係なく，J2EEサーバモードでWebアプリケーションを開始する時には必ず実行されます。

なお，javax.servlet.ServletContextAttributeListenerインタフェースを実装したクラスを使用した場合は，上記属性の追加，削除するためのattributeAddedメソッド，attributeRemovedメソッド，およびattributeReplacedメソッドの呼び出しは実行されません。

6.12　DD での定義

ここでは，テンプレートを利用した DD の作成と，DD に定義する項目について説明します。

6.12.1　DD の作成

セッション情報の引き継ぎを行う Web アプリケーションの設定では，Web アプリケーションにセッションフェイルオーバ用フィルタを追加して，セッション情報の引き継ぎに関する項目やフィルタマッピングを定義します。フィルタの設定の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.9.1　アプリケーションサーバが提供するサーブレットフィルタ (built-in フィルタ)」も参照してください。なお，Servlet 3.0 以降では，API でフィルタを定義できますが，セッションフェイルオーバ用フィルタは API で定義できません。

次に示すディレクトリに，メモリセッションフェイルオーバ機能で使用するフィルタ定義およびフィルタマッピングの定義が記述されたテンプレートの DD（web.xml）が用意してあります。

テンプレート格納ディレクトリ

<Application Serverのインストールディレクトリ>¥CC¥sfo¥sfo_war

このファイルをコピーして Web アプリケーションの DD を作成します。すでに存在する Web アプリケーションにメモリセッションフェイルオーバ機能を適用する場合，テンプレートから DD にセッションフェイルオーバ用フィルタのフィルタ定義およびフィルタマッピングの定義をコピーし，必要な定義を記述します。

! 注意事項

タグの指定を省略する場合，タグの記述も含めて省略する必要があります（<init-param>タグ～</init-param>タグの記述を省略します）。次のようにタグを記述して値だけ省略した場合，空白文字列を指定したと判断されてエラーとなります。

間違った省略方法の例

```
      (省略)
        :
<init-param>
<param-name>LockTimeout</param-name>
<param-value></param-value>
</init-param>
        :
      (省略)
```

テンプレートの内容を次に示します。

図 6-32　テンプレートの内容（メモリセッションフェイルオーバ機能の DD）（1/2）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
    All Rights Reserved. Copyright (C) 2004, 2007, Hitachi, Ltd.
-->
<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_2_4.xsd"
    version="2.4">

    <!- The filter for Session Fail Over -->
    <filter>                                                          フィルタ
        <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>  定義
        <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
        <init-param>
            <param-name>GSInfoKey-RW</param-name>
            <param-value>page</param-value>
        </init-param>

        <init-param>
            <param-name>GSInfoKey-R</param-name>
            <param-value>userId;roll</param-value>
        </init-param>

        <init-param>
            <param-name>GSInfosLengthMax</param-name>
            <param-value>1024</param-value>
        </init-param>

        <init-param>
            <param-name>SFOServerName</param-name>
            <param-value>sfo_server1</param-value>
        </init-param>

        <init-param>
            <param-name>GSessionControl</param-name>                  編集が
            <param-value>cookie</param-value>                         必要な
        </init-param>                                                 定義内容

        <!--
          Use this parameter only when the GSessionControl parameter is
          set to "cookie" or "both".
        -->

        <init-param>
            <param-name>GIDCookieName</param-name>
            <param-value>GSESSIONID</param-value>
        </init-param>

        <!--
          Use this parameter only when the GSessionControl parameter is
          set to "url-rewriting" or "both".
        -->
        <!--
        <init-param>
            <param-name>GIDURLParamName</param-name>
            <param-value>gsessionid</param-value>
        </init-param>
        -->
```

図 6-33　テンプレートの内容（メモリセッションフェイルオーバ機能の DD）(2/2)

```
        <init-param>
            <param-name>LockTimeout</param-name>
            <param-value>30</param-value>
        </init-param>

        <init-param>
            <param-name>GSInfoTypeByteArrayEnable</param-name>
            <param-value>false</param-value>
        </init-param>

        <init-param>
            <param-name>GSInfoTypeSerializableEnable</param-name>
            <param-value>false</param-value>
        </init-param>
    </filter>
    <!-- Specify the filter tags here. -->


    <!-- The filter mappings for the Session Fail Over-->
    <filter-mapping>
        <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- Specify the filter-mapping tags here. -->

</web-app>
```

次の定義の値は固定となります。テンプレートの値をそのまま使用してください。

- フィルタ定義の<filter-name>タグと<filter-class>タグ
- フィルタマッピング定義の<filter-name>タグと<url-pattern>タグ

! 注意事項

フィルタマッピング（<filter-mapping>タグ）を定義する場合，次の点に注意してください。

- フィルタマッピング定義を複数定義する場合，セッションフェイルオーバ用フィルタのマッピング定義を最初に記述してください。最初に記述しない場合はアプリケーションを開始できません。
- セッションフェイルオーバ用フィルタのマッピング定義は複数定義しないでください。複数定義した場合はアプリケーションを開始できません。
- Web アプリケーションのバージョンが Servlet 2.4 以降の場合，<filter-mapping>タグの要素の<dispatcher>タグを指定しないでください。<dispatcher>タグで「FORWARD」，「INCLUDE」，および「ERROR」を指定した場合，Web アプリケーションの開始時にセッションフェイルオーバ用フィルタの初期化段階でエラーが発生し，アプリケーションは開始できません。

6.12.2　DD に定義する項目

グローバルセッション情報として登録する情報や，使用する SFO サーバの情報などは，<filter>タグ内の初期化パラメタである，<init-param>タグ内を編集して定義します。

参考

<init-param>タグ内の情報は，アプリケーションを J2EE サーバにインポートしたあとで，サーバ管理コマンドを使用して設定することもできます。

DD の<init-param>タグ内に定義する内容を次の表に示します。なお，<param-value>に指定する値の詳細については，(1)以降で説明します。

表 6-17　DD の<init-param>タグ内に定義する内容

定義する項目		<param-name>タグに指定する要素	<param-value>に指定する内容
グローバルセッション情報として使用する情報	書き換え可能な情報	GSInfoKey-RW	HTTP セッションに登録する情報の中から，グローバルセッション情報として登録する情報を選択して定義します。 文字列を指定します。※1 「;」（半角セミコロン）を含んだキーは指定できません。 複数指定する場合は，「;」（半角セミコロン）で区切って指定します。 なお，同じ名称のキーは複数回指定できません。GSInfoKey-RW と GSInfoKey-R に同じ名称を指定することもできません。※2 また，「*」（半角アスタリスク）だけを指定すると，HTTP セッションに登録されたすべての情報がグローバルセッション情報として登録されます。ただし，複数指定の中に「*」を指定している場合，「*」はキーとして扱われます。「*」は，GSInfoKey-RW と GSInfoKey-R の両方に指定できません。 GSInfoKey-RW と GSInfoKey-R に指定する値と引き継ぎ対象となるセッション情報の関係については，「(1)　グローバルセッション情報として使用する情報（GSInfoKey-RW，GSInfoKey-R)」を参照してください。
	読み込み専用の情報	GSInfoKey-R	
グローバルセッション情報の最大サイズ		GSInfosLengthMax	グローバルセッション情報の最大サイズ（文字）を指定します。 DD には，3～1048576 の整数値で指定します。
グローバルセッション情報を格納する SFO サーバ		SFOServerName	使用する SFO サーバ名を指定します。 DD には，J2EE サーバ用の usrconf.properties の webserver.sfo.sfo_servers キーで指定されている SFO サーバの中から選択して指定します。
グローバルセッションの制御方法		GSessionControl	グローバルセッション情報の制御方法を指定します。 指定できる値は，「cookie」，「url-rewriting」または「both」です。 このパラメタに指定した値については，大文字と小文字が区別して扱われます。
グローバルセッション ID の送信に使用する HTTP Cookie の名称		GIDCookieName	グローバルセッション ID を送信する HTTP Cookie の名称を指定します。 DD には，16 文字までの文字列で指定します。 使用できる文字は，A～Z，a～z および 0～9 の英数字です。ただし，HTTP Cookie の名称の識別では，大文字と小文字は区別されません。 また，HTTP セッションのセッション ID が使用する HTTP Cookie の名称※3 や J2EE サーバが使用するサーバ ID，または「csfcfc」と同じ値は指定できません。
グローバルセッション ID の送信に使用する URL のパスパラメタ名		GIDURLParamName	グローバルセッション ID の送信に使用する URL のパスパラメタ名を指定します。 DD には，16 文字までの文字列で指定します。

定義する項目		<param-name>タグに指定する要素	<param-value>に指定する内容
グローバルセッション ID の送信に使用する URL のパスパラメタ名		GIDURLParamName	使用できる文字は，A～Z，a～z および 0～9 の英数字です。URL のパスパラメタの識別では，大文字と小文字は区別されます。 また，URL のパスパラメタ名には，HTTP セッションのセッション ID が使用する URL のパスパラメタ名※4 を指定できません。なお，URL のパスパラメタ名の識別では，大文字と小文字は区別されません。 また，Web アプリケーションで独自の URL のパスパラメタを使用している場合は，Web アプリケーションで使用している URL のパスパラメタと異なる名称をこのパラメタに，指定してください。
グローバルセッション情報のロック待ちタイムアウト時間		LockTimeout	SFO サーバでのロック待ちタイムアウト時間を秒単位で指定します。 DD には，1～3600 の整数値を指定します。
引き継ぎ可能なセッション情報のデータ型	byte 型配列であるセッション情報の引き継ぎ許可	GSInfoTypeByteArrayEnable	byte 型配列のセッション情報を引き継ぎ対象とするかどうかを指定します。 true byte 型配列のセッション情報を引き継ぎ対象とします。 false byte 型配列のセッション情報を引き継ぎ対象としません。byte 型配列を使用した場合は，グローバルセッション情報の更新時にエラーとなります。 true および false 以外の文字列を指定した場合は，デフォルト値の false が適用されます。 このパラメタに指定した値について，大文字，小文字は区別されません。また，このパラメタに指定した値は，GSInfoTypeSerializableEnable パラメタに「true」を指定している場合は無視されます。
	直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可	GSInfoTypeSerializableEnable	直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象とするかどうかを指定します。 true 直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象とします。 false 直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象としません。 このパラメタに指定した値について，大文字，小文字は区別されません。

注※1　キー名称の先頭と最後の空白は省略されます。省略された結果，""（文字列）になったキー名称は無視されます。

注※2　同じキー名称を指定した場合，Web アプリケーションを開始できません。

注※3　Web コンテナが自動で設定する HTTP Cookie の名称は「JSESSIONID」です。この名称はアプリケーションサーバのバージョンが 09-00 以降の場合，変更できます。HTTP Cookie の名称を変更した場合の注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.9　セッション管理の注意事項」を参照してください。

注※4　Web コンテナが自動で設定する URL のパスパラメタ名は「jsessionid」です。この名称はアプリケーションサーバのバージョンが 09-00 以降の場合，変更できます。URL のパスパラメタ名を変更した場合の注意事項について

は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.9　セッション管理の注意事項」を参照してください。

これらの定義項目は，使用する機能によって設定するかどうかが異なります。使用する機能ごとの DD の<init-param>タグ内に定義する要素のデフォルト値と設定要否を次の表に示します。

表 6-18　使用する機能ごとの DD の<init-param>タグ内に定義する要素のデフォルト値と設定要否

<param-name>タグに指定する要素	デフォルト値	使用する機能	
		メモリセッションフェイルオーバ機能の設定要否	グローバルセッション情報のサイズ見積もり機能の設定要否
GSInfoKey-RW	なし	任意※1	任意※1
GSInfoKey-R	なし	任意※1	任意※1
GSInfosLengthMax	1024	任意	無効※2
SFOServerName	webserver.sfo.sfo_servers キーに指定されている SFO サーバ数によって異なります。※3	任意	無効※4
GSessionControl	cookie	任意	任意
GIDCookieName	GSESSIONID	任意	任意
GIDURLParamName	gsessionid	任意	任意
LockTimeout	60	任意	無効※2
GSInfoTypeByteArrayEnable	false	任意	任意
GSInfoTypeSerializableEnable	false	任意	任意

（凡例）

任意：省略できることを示します。省略した場合はデフォルトの値が設定されます。

無効：指定しても無効になることを示します。設定した値は使用されません。

注※1　GSInfoKey-RW と GSInfoKey-R の両方を省略することはできません。

注※2　パラメタに不正な値を指定した場合はエラーとなり，Web アプリケーションは開始できません。

注※3　セッションフェイルオーバ用フィルタのサーバ定義名に指定されている SFO サーバ名が一つの場合は，その SFO サーバ名がデフォルトになります。複数指定されている場合は，先頭に指定されている SFO サーバ名がデフォルトになります。セッションフェイルオーバ用フィルタのサーバ定義名は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.sfo.sfo_servers パラメタで指定します。

注※4　指定した SFO サーバ名が，簡易構築定義ファイルで指定されているセッションフェイルオーバ用フィルタのサーバ定義名にない場合もエラーになりません。セッションフェイルオーバ用フィルタのサーバ定義名は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内に，webserver.sfo.sfo_servers パラメタで指定します。

なお，グローバルセッションの制御方法によって，指定する必要があるパラメタが異なります。

グローバルセッションの制御方法と指定が必要なパラメタの対応を次の表に示します。

表 6-19　グローバルセッションの制御方法と指定が必要なパラメタの対応

<param-name>タグに指定する要 (パラメタ)	HTTP Cookie によって制御する場合	URL 書き換えによって制御する場合
GSInfoKey-RW	○	○
GSInfoKey-R	○	○
GSInfosLengthMax	○	○
SFOServerName	○	○
GSessionControl	○※	◎
GIDCookieName	○	－
GIDURLParamName	－	○
LockTimeout	○	○
GSInfoTypeByteArrayEnable	○	○
GSInfoTypeSerializableEnable	○	○

（凡例）

◎：指定が必要です。省略できません。

○：指定が必要です。省略した場合は，デフォルト値が有効になります。

－：指定は不要です。

注※

デフォルト値は「cookie」です。このため，省略した場合も HTTP Cookie によってグローバルセッションが制御されます。

注意

- グローバルセッション情報として使用する情報は，一つ以上指定する必要があります。それ以外の値の定義は任意です。デフォルト値がある項目については，省略するとデフォルト値が使用されます。
- パラメタ値を省略する場合，<init-param>タグごと省略してください。次のようにタグを残して値だけを省略した場合，空文字列を指定したと判断されて，エラーになります。

  間違った省略方法の例

```
        :
<init-param>
<param-name>GSInfosLengthMax</param-name>
<param-value></param-value>
</init-param>
        :
```

メモリセッションフェイルオーバ機能を使用するための DD の定義内容は，すべての冗長化したアプリケーションで同一である必要があります。定義内容が不正な場合，アプリケーション開始処理がエラーになります。メッセージは，KDJE34202-E のメッセージが出力されます。

(1)　グローバルセッション情報として使用する情報（GSInfoKey-RW，GSInfoKey-R）

HTTP セッションに登録する情報の中から，グローバルセッション情報として登録する情報を選択して定義します。HTTP セッションに登録する情報すべてをグローバルセッション情報として引き継ぐ場合は，「*」を指定します。

ただし，グローバルセッション情報として，HTTP セッションに登録されたすべてのセッション情報を指定すると，引き継ぐ必要のない情報が HTTP セッションに登録されている場合にリクエスト処理のパフォーマンスが低下することがあります。このため，必要最小限のセッション情報をグローバルセッション情報として SFO サーバに引き継ぐように設定することをお勧めします。必要最小限のグローバルセッション情報を引き継ぐことで，SFO サーバのパフォーマンスの向上を図れます。

グローバルセッション情報は，書き換え可能グローバルセッション情報または読み込み専用グローバルセッション情報として指定します。HTTP セッションに登録後，変更が必要なセッション情報は，書き換え可能グローバルセッション情報に指定します。HTTP セッションに登録後，変更が不要なセッション情報は，読み込み専用グローバルセッションに指定します。グローバルセッション情報については，「6.4.2　グローバルセッション情報」を参照してください。

書き換え可能グローバルセッション情報の場合は，パラメタの GSInfoKey-RW で指定します。読み込み専用グローバルセッション情報の場合は，GSInfoKey-R で指定します。

GSInfoKey-RW および GSInfoKey-R パラメタに指定した文字列は，「;」（半角セミコロン）で分割したあと，それぞれの値の先頭と最後の連続した空白文字※を削除した結果がキー名称として使用されます。空白文字※を削除した結果が「""」（空文字列）になったキー名称は無視されます。GSInfoKey-RW および GSInfoKey-R パラメタに指定した文字列に有効となるキー名称がない場合は，KDJE34202-E のメッセージが出力され，エラーとなります。

注※

空白文字とは，半角スペース，タブ，LF（ラインフィード（0x0a)），CR（キャリッジリターン（0x0d)）または FF（フォームフィード（0x0c)）のことを指します。

GSInfoKey-RW および GSInfoKey-R パラメタに指定する値と引き継ぎ対象となるセッション情報の関係を次の表に示します。

表 6-20　GSInfoKey-RW および GSInfoKey-R パラメタに指定する値と引き継ぎ対象となるセッション情報の関係

項番	パラメタに指定する値	指定例	指定例で引き継ぎ対象となるセッション情報
1	「;」（半角セミコロン）で区切った複数の文字列	"user;roll"	"user"および"roll"のキーで登録されたセッション情報
2	キーの前，またはあとに空白を含んだ文字列	"user△;△roll△△;password"	"user"，"roll"，および"password"のキーで登録されたセッション情報（空白文字を削除した結果，キー名称となります）
3	キーの途中に空白を含んだ文字列	"user;user△id"	"user"および"user△id"のキーで登録されたセッション情報
4	「*」（半角アスタリスク）だけ	"*"	HTTP セッションに登録されたすべてのセッション情報※
5	前，またはあとに空白を含んだ「*」（半角アスタリスク）	"△△*△△"	HTTP セッションに登録されたすべてのセッション情報※（空白文字を削除した結果，「*」となります）

項番	パラメタに指定する値	指定例	指定例で引き継ぎ対象となるセッション情報
6	「*」（半角アスタリスク）を含んだ複数の文字列	"user;*"	"user"および"*"のキーで登録されたセッション情報
7	「*」(半角アスタリスク) 以外のキーが空文字列	"*;"	"*"のキーで登録されたセッション情報
8	空文字列	""	エラーとなり，KDJE34202-E のメッセージが出力されます。
9	「△」だけ	"△△"	エラーとなり，KDJE34202-E のメッセージが出力されます（空白文字を削除した結果，空文字列となります)。
10	「;」（半角セミコロン）だけ	";;"	エラーとなり，KDJE34202-E のメッセージが出力されます（有効となるキー名称がありません)。
11	「△」を「;」（半角セミコロン）で区切った文字列	";△;"	エラーとなり，KDJE34202-E のメッセージが出力されます(空白文字を削除した結果, 有効となるキー名称がありません)。

（凡例）△：空白文字を示します。

注※

GSInfoKey-RW パラメタで「*」を指定している場合に，GSInfoKey-R パラメタでキーを指定しているときは，GSInfoKey-R パラメタに指定したキーで登録されたセッション情報は含まれません。また，GSInfoKey-R パラメタで「*」を指定している場合に，GSInfoKey-RW パラメタでキーを指定しているときは，GSInfoKey-RW パラメタに指定したキーで登録されたセッション情報は含まれません。

(2)　グローバルセッション情報の最大サイズ（GSInfosLengthMax）

SFO サーバに格納するグローバルセッション情報の一つ当たりの最大サイズを指定します。SFO サーバで冗長化できるグローバルセッション情報のサイズは，デフォルトでは 1,024 バイトまでです。冗長化するグローバルセッション情報の文字数がこの値を超える場合は，定義した内容を変更する必要があります。

また，グローバルセッション情報の最大サイズを小さくすると，SFO サーバで使用するメモリサイズを少なくできます。実際に格納するグローバルセッション情報の最大サイズを考慮して，適切なサイズを指定してください。

! 注意事項

グローバルセッション情報のサイズ見積もり機能を有効にしている場合，GSInfosLengthMax パラメタの指定は無視されます。このため，グローバルセッション情報のサイズ見積もり機能を無効にして運用を開始する前に，KDJE34260-I のメッセージで出力されたグローバルセッション情報のサイズが，GSInfosLengthMax パラメタで指定したサイズを超えていないことを確認してください。

グローバルセッション情報のサイズは，次の式で算出できます。直列化可能クラスのオブジェクトであるセッション情報の引き継ぎを許可しているかどうか（GSInfoTypeSerializableEnable パラメタの指定）によって，見積もり式が異なります。

(a)　直列化可能クラスのオブジェクトであるセッション情報の引き継ぎを許可していない場合

グローバルセッション情報のサイズ＝
（各グローバルセッション情報のキーの文字列長合計）
＋（各 HTTP セッションに登録されたオブジェクトのサイズの最大長合計）
＋（区切り文字数）※1

＋（HTTP セッションの有効期間情報の長さ）※2

注※1

区切り文字数は，2×（設定したグローバルセッション情報の数）－1 で算出できます。これは，区切り文字には，グローバルセッション情報間の区切り文字と，各グローバルセッション情報のキーと値の区切り文字が含まれるためです。

注※2

HTTP セッションの有効期間情報の長さは 4 バイトです。

ただし，グローバルセッション情報が java.lang.String クラスの場合，文字列を UTF-8 の文字エンコーディングで URL エンコードして冗長化します（URL エンコードには java.net.URLEncoder クラスを使用します）。そのため，グローバルセッション情報の文字列長は，セッション情報の文字列長と異なることがあります。文字種ごとの URL エンコード後の文字列長を次の表に示します。

表 6-21　文字種ごとの URL エンコード後の文字列長

項番	文字種		文字列長（バイト）
1	ASCII 文字	英数字（0～9 および a～Z）および次に示す特殊文字 • ピリオド（.） • ハイフン（-） • アスタリスク（*） • アンダースコア（_）	1
		上記以外の ASCII 文字	3
2	JIS X 0201 のすべての非 ASCII 文字		6 または 9
3	JIS X 0208 のすべての非 ASCII 文字		6 または 9
4	JIS X 0212 のすべての非 ASCII 文字		6 または 9
5	Windows-31J の拡張文字		9
6	Unicode の補助文字（基本多言語面以外の文字）		12
7	JIS X 0213 の第 3，第 4 水準漢字		9 または 12

このため，グローバルセッション情報のサイズは，セッション情報の文字列を文字種ごとに URL エンコード後の文字列長に変換してから，それらを合計して見積もる必要があります。

例えば，n 種類のグローバルセッション情報を指定した場合は，グローバルセッション情報の最大サイズは次の式で算出できます。

グローバルセッション情報のサイズ＝
（グローバルセッション情報 1 のキーの文字列長）
＋（HTTP セッションに登録されたオブジェクト 1 の最大サイズ）
＋（グローバルセッション情報 2 のキーの文字列長）
＋（HTTP セッションに登録されたオブジェクト 2 の最大サイズ）
：
＋（グローバルセッション情報 n のキーの文字列長）
＋（HTTP セッションに登録されたオブジェクト n の最大サイズ）
＋（2×n-1）

＋4

参考

グローバルセッション情報のサイズ見積もり機能を使用すると，HTTP セッションに含まれるセッション情報ごとのバイト数が出力できます。出力されたサイズ情報を参考にして，グローバルセッション情報の最大サイズに適切な値を設定してください。グローバルセッション情報のサイズ見積もり機能については，「6.5　グローバルセッション情報のサイズの見積もり」を参照してください。

(b)　直列化可能クラスのオブジェクトであるセッション情報の引き継ぎを許可している場合

グローバルセッション情報のサイズ＝
（各グローバルセッション情報のキーのシリアライズ後のバイト数合計）
＋（各 HTTP セッションに登録されたオブジェクトのシリアライズ後のバイト数合計）
＋（HTTP セッションの有効期間情報の長さ）※

注
シリアライズは，一つの HTTP セッションに対して，一つの java.io.ObjectOutputStream のオブジェクトを使用して実施します。

注※
HTTP セッションの有効期間情報の長さは 4 バイトです。

例えば，n 種類のグローバルセッション情報を設定した場合は，グローバルセッション情報の最大サイズは次の式で算出できます。

グローバルセッション情報のサイズ＝
（グローバルセッション情報 1 のキーのシリアライズ後のバイト数）
＋（HTTP セッションに登録されたオブジェクト 1 のシリアライズ後のバイト数）
＋（グローバルセッション情報 2 のキーのシリアライズ後のバイト数）
＋（HTTP セッションに登録されたオブジェクト 2 のシリアライズ後のバイト数）
：
＋（グローバルセッション情報 n のキーのシリアライズ後のバイト数）
＋（HTTP セッションに登録されたオブジェクト n のシリアライズ後のバイト数）
＋4

注
シリアライズは，一つの HTTP セッションに対して，一つの java.io.ObjectOutputStream のオブジェクトを使用して実施します。

参考

グローバルセッション情報のサイズ見積もり機能を使用すると，各情報のシリアライズ後のバイト数が出力できます。出力されたサイズ情報を参考にして，グローバルセッション情報の最大サイズに適切な値を設定してください。グローバルセッション情報のサイズ見積もり機能については，「6.5　グローバルセッション情報のサイズの見積もり」を参照してください。

(3)　グローバルセッション情報を格納する SFO サーバ（SFOServerName）

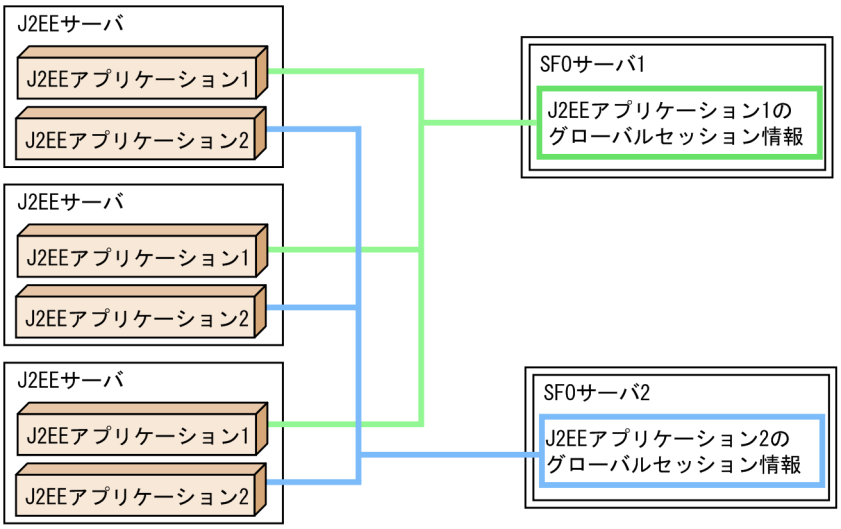
グローバルセッション情報を格納する SFO サーバのサーバ名を指定します。サーバ名は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内のパラメタ webserver.sfo.sfo_servers に指定した SFO サーバ名から選択します。

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

ポイント

システム内に複数の SFO サーバを配置する場合，メモリセッションフェイルオーバ機能を使用する Web アプリケーション単位に，SFO サーバを割り当てることをお勧めします。Web アプリケーション単位に SFO サーバを割り当てることによって，SFO サーバで障害が発生したときの影響範囲を特定の J2EE アプリケーションに限定させることができます。システム内に複数の SFO サーバを配置する場合の設定例を次の図に示します。

図 6-34　システム内に複数の SFO サーバを配置する場合の設定例



(4)　グローバルセッションの制御方法（GSessionControl）

グローバルセッションの制御方法を指定します。指定した値に応じて，Web クライアントに対するグローバルセッション ID の送信方法，および Web クライアントからのグローバルセッション ID の受信方法が決まります。

指定できる値と意味を次の表に示します。

表 6-22　GSessionControl の指定値の意味

指定値	意味
cookie	グローバルセッションの制御に HTTP Cookie を使用します。
url-rewriting	グローバルセッションの制御に URL 書き換えを使用します。
both	グローバルセッションの制御に HTTP Cookie および URL 書き換えの両方を使用します。 Web クライアントが HTTP Cookie をサポートしているかどうかを判別して，必要に応じて HTTP Cookie による制御と URL 書き換えによる制御を切り替えます。

また，GSessionControl に指定した値によって，次の制御が変わります。

- Web クライアントから J2EE サーバに送信されるグローバルセッション ID の扱われ方
- J2EE サーバから Web クライアントへのレスポンスの設定内容

GSessionControl に指定する値とグローバルセッションの制御方法の関係を，次の表に示します。

表 6-23　GSessionControl に指定する値とグローバルセッションの制御方法の関係

GSessionControl に指定する値	Web クライアントから J2EE サーバに送信されるリクエストに含まれるグローバルセッション ID		J2EE サーバから Web クライアントに送信されるレスポンスの内容	
	HTTP Cookie に含まれるグローバルセッション ID	URL のパスパラメタに含まれるグローバルセッション ID	グローバルセッション生成時に HTTP Cookie を付加するかどうか	ServletAPI による URL 書き換えが有効になるかどうか
cookie	有効	無効	付加する	無効
url-rewriting	無効	有効	付加しない	有効
both	有効※1	有効※1	付加する	有効※2

注※1

クライアントから送信されたリクエストが，HTTP Cookie および URL のパスパラメタの両方にグローバルセッション ID を含んでいる場合は，HTTP Cookie で送信されたグローバルセッション ID が有効になります。

注※2

Web クライアントが HTTP Cookie をサポートしている場合は，URL 書き換えは実行しません。

HTTP Cookie または URL 書き換えを使用したグローバルセッションの制御については，「6.6　HTTP Cookie によるグローバルセッションの制御」および「6.7　URL 書き換えによるグローバルセッションの制御」を参照してください。

(5)　グローバルセッション ID の送信に使用する HTTP Cookie の名称 (GIDCookieName)

グローバルセッション ID を Web ブラウザなどのクライアントに通知するときに使用する HTTP Cookie の名称を指定します。なお，グローバルセッション ID とは，グローバルセッション情報の識別に使用される情報です。

Cookie の名称は，Web アプリケーション単位に指定できます。ただし，次に示す Cookie の名称と同じ値を指定することはできません。

- JSESSIONID

  HTTP セッションのセッション ID の送信に使用する Cookie の名称です。
- HTTP セッションを生成したリクエストのレスポンスに付加されるサーバ ID の Cookie の名称

  HTTP セッションを生成したリクエストのレスポンスにサーバ ID の Cookie を付加する機能を有効にしている場合に確認が必要です。

  この機能を有効にしているかどうか，および付加されるサーバ ID の Cookie の名称については，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の<configuration>タグ内で，次のパラメタの値を参照してください。
  - webserver.container.server_id.enabled
  - webserver.container.server_id.name

なお，グローバルセッション ID の送信に使用する Cookie の名称が，これらの Cookie の名称と重複した場合，Web アプリケーションに定義したセッションフェイルオーバ用フィルタ（J2EE 仕様で規定された javax.servlet.Filter インタフェースを実装したクラス）の開始処理でエラーとなり，Web アプリケーションを実行できません。

(6)　グローバルセッション ID の送信に使用する URL のパスパラメタ名（GIDURLParamName）

グローバルセッション ID を Web ブラウザなどのクライアントに通知するときに使用する URL のパスパラメタ名を指定します。

GSessionControl パラメタに「cookie」を指定した場合，このパラメタに設定した値は無効になります。なお，GSessionControl パラメタに「cookie」を指定した場合，GIDURLParamName パラメタに設定した値はチェックされません。このため，このパラメタに不正な値を設定した場合も，検出はされません。

! 注意事項

Web アプリケーション内で使用する URL のパスパラメタと同じ名称を，このパラメタで指定しないでください。指定した場合，メモリセッションフェイルオーバ機能が正しく動作しません。

URL 書き換えを使用したグローバルセッションの制御については，「6.7　URL 書き換えによるグローバルセッションの制御」を参照してください。

(7)　グローバルセッション情報のロック待ちタイムアウト時間（LockTimeout）

すでにロックされた状態のセッションに対して，別の J2EE サーバからロック要求があった場合に，ロック待ちがタイムアウトになるまでの時間を指定します。

J2EE サーバから SFO サーバに対してグローバルセッション情報のロック取得要求をしたあとに，J2EE サーバに障害が発生して停止した場合，SFO サーバ上のグローバルセッション情報がロックされたままの状態になってしまうおそれがあります。グローバルセッション情報のロック待ちタイムアウト時間は，このような場合に，J2EE サーバに障害が発生したと判断するために使用されます。

また，次のような場合にロックされたままの状態になったときにも，このタイムアウト時間が有効になります。これらの状態は，負荷分散機を使用している環境で発生することがあります。

- ページにフレームを使用している場合など，同じセッションを使用するリクエストが同時に送信され，それぞれのリクエストが負荷分散機のスケジュール機能によって異なる J2EE サーバに分散されて転送されたとき
- ユーザの操作によってリクエストが二重に送信された場合など，リクエストを送信して応答が返されるよりも前に次のリクエストを送信したときに，それぞれのリクエストが負荷分散機のスケジュール機能によって異なる J2EE サーバに分散されて転送されたとき

フレームを使用するアプリケーションの場合は，リクエスト処理が終わるまでのロック待ち時間も考慮して，タイムアウト時間を定義してください。

(8)　引き継ぎ可能なセッション情報のデータ型（GSInfoTypeByteArrayEnable，GSInfoTypeSerializableEnable）

引き継ぎ可能なセッション情報のデータ型には，java.lang.String クラスのオブジェクト，byte 型配列，および直列化可能クラスのオブジェクトがあります。

データ型が byte 型配列，または直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象とする場合には，次の設定を有効にします。

- byte 型配列であるセッション情報の引き継ぎ許可
- 直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可

これらの設定を無効にした場合は，java.lang.String クラスのオブジェクトだけが引き継ぎ可能なセッション情報となります。

(a)　byte 型配列であるセッション情報の引き継ぎ許可

byte 型配列のセッション情報を引き継ぎ対象とするかどうかを GSInfoTypeByteArrayEnable パラメタに指定します。「true」を指定した場合は，java.lang.String クラスのオブジェクトに加え，byte 型配列のセッション情報も引き継ぎ対象となります。

「false」を指定した場合は，byte 型配列のセッション情報は引き継ぎ対象外となります。引き継ぎ対象外のオブジェクトが登録された場合の扱いについては，「6.4.2(3)　引き継ぎ対象外のオブジェクトの扱い」を参照してください。

(b)　直列化可能クラスのオブジェクトであるセッション情報の引き継ぎ許可

直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象とするかどうかを GSInfoTypeSerializableEnable パラメタに指定します。「true」を指定した場合は，直列化可能クラスのオブジェクトであるセッション情報が引き継ぎ対象となります。この場合，java.lang.String クラスのオブジェクト，および byte 型配列のセッション情報も区別なく，直列化可能クラスのオブジェクトであるセッション情報としてシリアライズして，グローバルセッション情報を作成します。

「false」を指定した場合は，直列化可能クラスのオブジェクトであるセッション情報は引き継ぎ対象外となります。引き継ぎ対象外のオブジェクトが登録された場合の扱いについては，「6.4.2(3)　引き継ぎ対象外のオブジェクトの扱い」を参照してください。

(c)　引き継ぎ可能なセッション情報のデータ型を指定する場合の注意事項

GSInfoTypeByteArrayEnable パラメタと GSInfoTypeSerializableEnable パラメタの指定値によって，引き継ぎ対象となるデータ型が異なります。各パラメタに指定する値と引き継ぎ対象となるデータ型の関係を次の表に示します。

表 6-24　GSInfoTypeByteArrayEnable および GSInfoTypeSerializableEnable パラメタに指定する値と引き継ぎ対象となるデータ型の関係

GSInfoTypeByteArrayEnable パラメタ※1 に指定する値	GSInfoTypeSerializableEnable パラメタ※2 に「true」を指定	GSInfoTypeSerializableEnable パラメタ※2 に「false」を指定
true	直列化可能クラスのオブジェクト	java.lang.String クラスのオブジェクトまたは byte 型配列
false	直列化可能クラスのオブジェクト	java.lang.String クラスのオブジェクト

注※1　byte 型配列のセッション情報を引き継ぎ対象とするかどうかを指定するパラメタです。

注※2　直列化可能クラスのオブジェクトであるセッション情報を引き継ぎ対象とするかどうかを指定するパラメタです。

なお，引き継ぎ対象外のオブジェクトが登録された場合の扱いについては，「6.4.2(3)　引き継ぎ対象外のオブジェクトの扱い」を参照してください。

(9)　統合ユーザ管理機能を使用する場合の設定

統合ユーザ管理機能を使用する場合に必要な設定について説明します。

(a)　統合ユーザ管理機能を使用する場合のグローバルセッション情報

統合ユーザ管理機能を使用する場合のセッション情報は，HTTP セッションに登録したあとで変更が必要になります。書き換え可能なグローバルセッション情報（GSInfoKey-RW パラメタ）を選択してください。また，GSInfoKey-RW パラメタには，「com.cosminexus.admin.auth.user_data」を設定してください。

(b)　統合ユーザ管理機能を使用する場合のグローバルセッション情報の最大サイズ

統合ユーザ管理機能を使用する場合のグローバルセッション情報の最大サイズの見積もり式を次に示します。

```
グローバルセッション情報の最大サイズ
    ＝統合ユーザ管理で使用する値
    ＋ユーザアプリケーションで使用する値
```

統合ユーザ管理で使用する値は，統合ユーザ管理フレームワークの API を使用する場合とタグライブラリを使用する場合で異なります。また，パスワード保存機能を使用するかどうかによっても異なります。それぞれの場合の統合ユーザ管理で使用する値の計算式を次に示します。

- API を使用し，パスワード保存機能を使用する場合

```
統合ユーザ管理で使用する値
    ＝｛1,500＋（レルム名の長さ＋ユーザ名の最大長＋パスワードの最大長
    ＋1,200）×レルム数｝×2
```

- API を使用し，パスワード保存機能を使用しない場合

```
統合ユーザ管理で使用する値
    ＝｛1,500＋（レルム名の長さ＋ユーザ名の最大長＋1,200）×レルム数｝×2
```

- タグライブラリを使用し，パスワード保存機能を使用する場合

```
統合ユーザ管理で使用する値
    ＝｛1,500＋（レルム名の長さ＋ユーザ名の最大長
    ＋loginタグのid属性値の長さ＋loginタグのentry属性値の長さ
    ＋パスワードの最大長＋1,200）×レルム数｝×2
```

- タグライブラリを使用し，パスワード保存機能を使用しない場合

```
統合ユーザ管理で使用する値
    ＝｛1,500＋（レルム名の長さ＋ユーザ名の最大長
    ＋loginタグのid属性値の長さ＋loginタグのentry属性値の長さ
    ＋1,200）×レルム数｝×2
```

ポイント

計算式中の 1,500 と 1,200 は固定値です。統合ユーザ管理の管理情報を保存するために必要な値です。この値はオブジェクトをシリアライズしたときのバイト数の最大値を基に算出しています。

！ 注意事項

セッションフェイルオーバ用フィルタの設定に誤りがある場合，セッションフェイルオーバが正しく行われませんので注意してください。

統合ユーザ管理では，GIDCookieName パラメタが示す値を取得できたときに，統合ユーザ管理でのメモリセッションフェイルオーバ機能を有効にします。ただし，統合ユーザ管理からはセッションフェイルオーバ用フィルタの内容が正しいかをチェックできません。このため，フィルタの設定に誤りがあった場合，実際にフェイルオーバが発生するまで誤りに気づかないおそれがあります。

6.12.3　DD での定義例

ここでは，DD の定義例として，HTTP Cookie による制御の場合，URL 書き換えによる制御の場合，および制御方法に both を指定する場合を紹介します。

(1)　HTTP Cookie による制御の場合

メモリセッションフェイルオーバ機能を使用する場合の DD の定義例を示します。この例は，HTTP Cookie によってグローバルセッションを制御し，byte 型配列のセッション情報を引き継ぐ場合の例です。

```
      :
<!--The filter for Session Fail Over -->
<filter>
   <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
   <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
   <init-param>
     <param-name>GSInfoKey-RW</param-name>
     <param-value>page</param-value>
   </init-param>

   <init-param>
     <param-name>GSInfoKey-R</param-name>
     <param-value>userId;roll</param-value>
   </init-param>

   <init-param>
     <param-name>GSInfosLengthMax</param-name>
     <param-value>44</param-value>
   </init-param>

   <init-param>
     <param-name>SFOServerName</param-name>
     <param-value>sfo_server1</param-value>
   </init-param>

   <init-param>
     <param-name>GSessionControl</param-name>
     <param-value>cookie</param-value>
   </init-param>

   <init-param>
     <param-name>GIDCookieName</param-name>
     <param-value>SFOSESSIONID</param-value>
   </init-param>

   <init-param>
     <param-name>LockTimeout</param-name>
     <param-value>180</param-value>
   </init-param>

   <init-param>
     <param-name>GSInfoTypeByteArrayEnable</param-name>
     <param-value>true</param-value>
   </init-param>
</filter>

<!-- Specify the filter tags here. -->
<filter>
   <filter-name>UserFilter</filter-name>
   <filter-class>filter.UserFilter</filter-class>
</filter>

<!-- The filter mappings for the Session Fail Over-->
<filter-mapping>
   <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- Specify the filter-mapping tags here. -->
<filter-mapping>
   <filter-name>UserFilter</filter-name>
   <url-pattern>/servlet/*</url-pattern>
</filter-mapping>
      :
</web-app>
```

この例で定義される内容は，次のとおりです。

- 書き換え可能グローバルセッション情報として，「page」の値を格納します。
- 読み込み専用グローバルセッション情報として，「userId」および「roll」の 2 種類の値を格納します。
- グローバルセッション情報の最大サイズとして，49 を設定します。
  最大サイズは，次の式で算出します。ここでは，page（4 文字）の値は 1～2 文字の ASCII 文字列，userId（6 文字）の値は 1～12 バイトの byte 型配列，roll（4 文字）の値は 1～7 文字の ASCII 文字列とします。
  （4 ＋ 6 ＋ 4）＋（2 ＋ 7 ＋（12 ＋ 5））＋（2×3-1）＋ 4 ＝ 49
- 使用する SFO サーバの名称は，sfo_server1 とします。
- グローバルセッション ID のキー名称は，SFOSESSIONID とします。
- SFO サーバでのロック待ちタイムアウト時間は，180 秒とします。
- byte 型配列のセッション情報を引き継ぎます。

(2)　URL 書き換えによる制御の場合

メモリセッションフェイルオーバ機能を使用する場合の DD の定義例を示します。この例は，URL 書き換えによってグローバルセッションを制御する場合の例です。

なお，ここでは，URL 書き換えによる制御の場合に HTTP Cookie による制御と異なる設定だけについて示します。

```
      :
<!--The filter for Session Fail Over -->
<filter>
   <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
   <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
      :
   <init-param>
     <param-name>GSessionControl</param-name>
     <param-value>url-rewriting</param-value>
   </init-param>

   <init-param>
     <param-name>GIDURLParamName</param-name>
     <param-value>sfogid</param-value>
   </init-param>
      :
</filter>
```

この例では，グローバルセッション ID の送信に使用する URL のパスパラメタ名は，「sfogid」としています。

(3)　制御方法に both を指定する場合

メモリセッションフェイルオーバ機能を使用する場合の DD の定義例を示します。この例は，制御方法として GSessionControl に「both」を指定する場合の例です。both は，Web クライアントが HTTP Cookie をサポートしているかどうかを判別して，必要に応じて HTTP Cookie による制御と URL 書き換えによる制御を切り替える方法です。

なお，ここでは，both を指定する場合に HTTP Cookie による制御と異なる設定だけについて示します。

```
      :
<filter>
   <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
   <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
      :
   <init-param>
```

```
      <param-name>GSessionControl</param-name>
      <param-value>both</param-value>
    </init-param>

    <init-param>
      <param-name>GIDCookieName</param-name>
      <param-value>SFOSESSIONID</param-value>
    </init-param>

    <init-param>
      <param-name>GIDURLParamName</param-name>
      <param-value>sfogid</param-value>
    </init-param>
      :
 </filter>
```

この例では，グローバルセッション ID の送信に使用する HTTP Cookie の名称は「SFOSESSIONID」としています。また，グローバルセッション ID の送信に使用する URL のパスパラメタ名は，「sfogid」としています。

(4)　統合ユーザ管理機能を使用しない場合

統合ユーザ管理機能を使用しない場合の DD でのフィルタ属性の記述例を次に示します。この例は，HTTP Cookie を使用してグローバルセッションを制御し，byte 型配列のセッション情報を引き継ぐ場合の例です。

- DD でのフィルタ属性の記述例

```
    （省略）
      :
<filter>
    <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
    <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
    <init-param>
        <param-name>GSInfoKey-RW</param-name>
        <param-value>page</param-value>
    </init-param>

    <init-param>
        <param-name>GSInfoKey-R</param-name>
        <param-value>userId;roll</param-value>
    </init-param>

    <init-param>
        <param-name>GSInfosLengthMax</param-name>
        <param-value>44</param-value>
    </init-param>

    <init-param>
        <param-name>SFOServerName</param-name>
        <param-value>sfo_server1</param-value>
    </init-param>

    <init-param>
        <param-name>GSessionControl</param-name>
        <param-value>cookie</param-value>
    </init-param>

    <init-param>
        <param-name>GIDCookieName</param-name>
        <param-value>SFOSESSIONID</param-value>
    </init-param>

    <init-param>
        <param-name>LockTimeout</param-name>
        <param-value>180</param-value>
    </init-param>

    <init-param>
        <param-name>GSInfoTypeByteArrayEnable</param-name>
        <param-value>true</param-value>
    </init-param>
      :
  （省略）
```

- DD での WAR 属性の記述例

```
      （省略）
        :
<filter-mapping>
    <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
        :
      （省略）
```

この記述例で設定される内容は，次のとおりです。

- 書き換え可能なグローバルセッション情報として，「page」の値を格納します。
- 読み取り専用のグローバルセッション情報として，「UserID」および「Roll」の 2 種類の値を格納します。
- グローバルセッション情報の最大サイズとして，49 を設定します。

  最大サイズは，次の式で算出します。なお，ここでは，page（4 文字）の値は 1～2 文字の ASCII 文字列，UserID（6 文字）の値は 1～12 バイトの byte 型配列，roll（4 文字）の値は 1～7 文字の ASCII 文字列とします。

  （4 ＋ 4 ＋ 6）＋（2 ＋ 7 ＋（12 ＋ 5））＋（2×3-1）＋ 4 ＝ 49
- グローバルセッション情報を格納する SFO サーバの名称は，「sfo_server1」とします。
- グローバルセッションの制御には，HTTP Cookie を使用します。
- グローバルセッション ID の送信に使用する Cookie の名称は，「SFOSESSIONID」とします。
- グローバルセッション情報のロック待ちタイムアウトは，「180 秒」とします。
- byte 型配列のセッション情報を引き継ぐため，GSInfoTypeByteArrayEnable パラメタに true を指定します。

(5)　統合ユーザ管理機能を使用する場合

統合ユーザ管理機能を使用する場合の DD の記述例を次に示します。この例は，HTTP Cookie を使用してグローバルセッションを制御している場合の例です。

- DD でのフィルタ属性の記述例

```
      （省略）
        :
<filter>
    <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
    <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
    <init-param>
        <param-name>GSInfoKey-RW</param-name>
        <param-value>com.cosminexus.admin.auth.user_data</param-value>
    </init-param>

    <init-param>
        <param-name>GSInfoKey-R</param-name>
        <param-value></param-value>
    </init-param>

    <init-param>
        <param-name>GSInfosLengthMax</param-name>
        <param-value>10380</param-value>
    </init-param>

    <init-param>
        <param-name>SFOServerName</param-name>
        <param-value>sfo_server1</param-value>
    </init-param>

    <init-param>
        <param-name>GSessionControl</param-name>
        <param-value>cookie</param-value>
```

```
    </init-param>

    <init-param>
        <param-name>GIDCookieName</param-name>
        <param-value>GSESSIONID</param-value>
    </init-param>

    <init-param>
        <param-name>LockTimeout</param-name>
        <param-value>30</param-value>
    </init-param>

    <init-param>
        <param-name>GSInfoTypeByteArrayEnable</param-name>
        <param-value>false</param-value>
    </init-param>

</filter>
        :
      （省略）
```

- DD での WAR 属性の記述例

```
      （省略）
        :
<filter-mapping>
    <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
        :
      （省略）
```

この記述例で設定される内容は，次のとおりです。

- 書き換え可能なグローバルセッション情報として，「com.cosminexus.admin.auth.user_data」の値を格納します。
- グローバルセッション情報の最大サイズとして，10,380 文字を設定します。

  最大サイズは，次の式で算出します。

  {1,500 +(10 + 20 + 1,200)×3} ×2 = 10,380

  なお，ここでは，次の条件で最大サイズを算出しています。

  - API を使用する。
  - パスワード保存機能は使用しない。
  - レルム名の長さ：10
  - ユーザ名の最大長：20
  - レルム数（シングルサインオンを使用するシステムの数）：3
- グローバルセッション情報を格納する SFO サーバの名称は，「sfo_server1」とします。
- グローバルセッションの制御には，HTTP Cookie を使用します。
- グローバルセッション ID の送信に使用する Cookie の名称は，「GSESSIONID」とします。
- グローバルセッション情報のロック待ちタイムアウトは，「30」秒とします。
- グローバルセッション情報として，byte 型配列を使用しないため，GSInfoTypeByteArrayEnable パラメタに false を指定します。

6.13　cosminexus.xml での定義

メモリセッションフェイルオーバ機能を使用するための定義は，cosminexus.xml の<war>タグ内に指定します。

Web アプリケーションにセッションフェイルオーバ用フィルタを組み込むために，追加するフィルタ名称を指定します。<filter>-<filter-name>タグに次に示す固定値を指定してください。

設定値（固定値）

com.hitachi.software.was.sfo.web.SFOFilter

指定するタグの詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「2.2.7　Filter 属性の詳細」を参照してください。

ポイント

セッション情報の引き継ぎを行う Web アプリケーションを含む J2EE アプリケーションの開始後に Web アプリケーションの設定を変更する場合，J2EE アプリケーションを再開始する前に，SFO サーバに登録された Web アプリケーションの設定情報を削除する必要があります。SFO サーバに登録された Web アプリケーションの設定情報の削除方法については，「6.17　Web アプリケーションの設定情報の削除」を参照してください。

6.14　J2EE サーバの設定

メモリセッションフェイルオーバ機能を使用する場合，J2EE サーバの設定が必要です。

6.14.1　J2EE サーバの設定内容

セッション情報の引き継ぎを行う J2EE サーバの設定内容について説明します。

セッション情報の引き継ぎを行う J2EE サーバの設定では，簡易構築定義ファイルを編集して次に示す設定をします。

- **グローバルセッション情報を格納する SFO サーバの設定**

  セッション情報の引き継ぎを行う J2EE サーバのグローバルセッション情報を格納する SFO サーバの名称，および SFO サーバ用の CORBA ネーミングサービスの IP アドレスを設定します。

- **SFO サーバとの通信タイムアウトの設定**

  J2EE サーバと SFO サーバとの通信タイムアウトを設定します。

  通信タイムアウト時間の設定では，FullGC などによる応答の遅延を考慮し，SFO サーバで発生した障害以外の原因でタイムアウトが発生しないように値を設定してください。

  SFO サーバとの通信タイムアウトは，J2EE サーバが SFO サーバに障害が発生したと判断するために使用される時間です。設定したタイムアウト時間を超えると，J2EE サーバでは，SFO サーバは障害のため停止していると判断し，リトライを実行しません。

  次の場合に，J2EE サーバが SFO サーバとの通信タイムアウトを検知して，SFO サーバに障害が発生したと判断します。

  - SFO サーバに障害が発生して SFO サーバとの通信が完了しなかった場合
  - FullGC などで J2EE サーバが高負荷状態となり，通信タイムアウトに設定した時間内に SFO サーバとの通信が完了しなかった場合

  通信タイムアウトが発生して SFO サーバとの接続が一時的に切断された場合，再接続が発生します。

  ! 注意事項

  通信タイムアウトが発生して SFO サーバとの接続が切断された場合の注意事項

  - SFO サーバに障害が発生して停止した場合，ビジネスメソッドの呼び出しができなくなります。

    SFO サーバアプリケーションで用意しているコネクションの状態監視用のビジネスメソッドは，SFO サーバのヘルスチェックを実施するために SFO サーバ監視スレッドによって呼び出されます。また，リクエスト実行時には，SFO サーバ上のグローバルセッション情報を操作するためのビジネスメソッドが呼び出されます。SFO サーバ監視スレッドは，J2EE サーバで作成されます。
  - SFO サーバとの接続が切断された場合は，SFO サーバがダウンした場合と同じ状態となります。切断中に J2EE サーバがリクエストを受信した場合，そのリクエストの HTTP セッションに登録された書き換え可能グローバルセッション情報は削除されます。

- **HTTP セッションのセッション ID へのサーバ ID の付加**

  HTTP セッションのセッション ID にサーバ ID を付加する機能を有効にして，HTTP セッションのセッション ID に付加するサーバ ID を設定します。

- **サーバ ID の Cookie の付加**

  HTTP Cookie によってグローバルセッションを制御する場合に，HTTP セッションを生成したリクエストのレスポンスにサーバ ID の Cookie を付加する機能を有効にして，付加するサーバ ID の Cookie の名称を設定します。

- **リクエストラインの最大サイズの設定**

URL 書き換えによってグローバルセッションを制御する場合に，リクエストラインの最大サイズを設定します。URL 書き換えによってグローバルセッションを制御する場合，URL のパスパラメタには，HTTP セッションのセッション ID とグローバルセッション ID が含まれます。HTTP リクエストのリクエストラインのサイズには，上限があります。このため，あらかじめリクエストラインの最大サイズを算出して，それを基にリクエストラインの最大サイズを設定する必要があります。

なお，リクエストラインの最大サイズを J2EE サーバに設定するのは，インプロセス HTTP サーバを使用している場合です。HTTP Server または Microsoft IIS を使用している場合は，それぞれの Web サーバに設定します。

設定値の考え方および各 Web サーバへの設定方法については，「6.7.5　リクエストラインの最大サイズの設定」を参照してください。

! 注意事項

J2EE サーバ間でのセッション情報の引き継ぎを行うために編集した，簡易構築定義ファイルのパラメタに不正な値が指定されていたり，必要なパラメタの指定が不足していたりした場合，J2EE サーバの起動時ではなく，セッション情報の引き継ぎ時にエラーが発生し，セッション情報の引き継ぎを行う Web アプリケーションを開始できません。

6.14.2　簡易構築定義ファイルでの定義

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

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

簡易構築定義ファイルでのメモリセッションフェイルオーバ機能の定義について次の表に示します。

表 6-25　簡易構築定義ファイルでのメモリセッションフェイルオーバ機能の定義

項目	指定するパラメタ	設定内容
グローバルセッション情報を格納する SFO サーバの設定	webserver.sfo.sfo_servers	HTTP セッションの情報の引き継ぎを行う J2EE サーバのグローバルセッション情報を格納する SFO サーバの名称※1（セッションフェイルオーバ用フィルタのサーバ定義名）を指定します。 • 複数の SFO サーバにグローバルセッション情報を格納する場合 すべての SFO サーバの名称（セッションフェイルオーバ用フィルタのサーバ定義名）を指定します。 • SFO サーバに対して複数のセッションフェイルオーバ用フィルタのサーバ定義名を指定する場合 それぞれの SFO サーバに異なるセッションフェイルオーバ用フィルタのサーバ定義名を指定してください。複数の SFO サーバに同一のセッションフェイルオーバ用フィルタのサーバ定義名を指定することはできません。 • 複数の SFO サーバを指定する場合 一つ目に指定された SFO サーバが，Web アプリケーションでの指定が省略された場合にデフォルトで使用する SFO サーバとなります。
	webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.naming	webserver.sfo.sfo_servers パラメタに指定した SFO サーバで起動する CORBA ネーミングサービスの IP アドレス（またはホスト名）およびポート番号を指定します。

項目	指定するパラメタ	設定内容
グローバルセッション情報を格納する SFO サーバの設定	webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.naming	複数の SFO サーバにグローバルセッション情報を格納する場合，それぞれの SFO サーバで起動する CORBA ネーミングサービスの IP アドレス（またはホスト名）およびポート番号を指定します。
SFO サーバとの通信タイムアウトの設定	webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.connectionTimeout	J2EE サーバと SFO サーバとの通信タイムアウトを指定します。 複数の SFO サーバにグローバルセッション情報を格納する場合，それぞれの SFO サーバとの通信タイムアウトを指定します。
セッション ID へのサーバ ID の付加	webserver.session.server_id.enabled	HTTP セッションのセッション ID にサーバ ID を付加するかどうかを指定します。J2EE サーバ間で HTTP セッションの情報の引き継ぎを行う場合，「true」を指定します。
	webserver.session.server_id.value※2	HTTP セッションのセッション ID に付加するサーバ ID を指定します。 指定するサーバ ID は，各 J2EE サーバに対して一意な値としてください。HTTP セッションのセッション ID に設定されるサーバ ID が，ほかの J2EE サーバのサーバ ID と重複した場合，SFO サーバとのネゴシエーションに失敗し，セッション情報の引き継ぎを行う Web アプリケーションは開始できません。
サーバ ID の Cookie の付加（HTTP Cookie を使用する場合）※3	webserver.container.server_id.enabled	HTTP セッションを生成したリクエストのレスポンスにサーバ ID の Cookie を付加するかどうかを指定します。同一セッションのリクエストを，同一の Web コンテナに転送する場合，「true」を指定します。
	webserver.container.server_id.name	HTTP セッションを生成したリクエストのレスポンスにサーバ ID の Cookie を付加する場合に，Cookie の名称を指定します。 Cookie に設定するサーバ ID は，サーブレットや JSP で設定する Cookie の名称，および Web コンテナが自動設定する Cookie の名称「JSESSIONID」と重複しないように注意してください。
	webserver.container.server_id.value※2	HTTP セッションを生成したリクエストのレスポンスにサーバ ID の Cookie を付加する場合に，Cookie の値を指定します。
リクエストラインの最大サイズの設定（URL 書き換えを使用する場合）※1	webserver.connector.inprocess_http.limit.max_request_line	リクエストラインの最大サイズを設定します。 なお，このパラメタはインプロセス HTTP サーバを使用している場合に設定するパラメタです。設定値の算出方法と，HTTP Server または Microsoft IIS を使用している場合の設定個所については，「6.7.5　リクエストラインの最大サイズの設定」を参照してください。

注※1

SFO サーバの名称（セッションフェイルオーバ用フィルタのサーバ定義名）は，セッション情報の引き継ぎを行う J2EE サーバと Web アプリケーション間での SFO サーバの識別に使用する名称です。指定したセッションフェイルオーバ用フィルタのサーバ定義名に SFO サーバ（論理サーバ）を関連づけて利用します。ここで指定したセッションフェイルオーバ用フィルタのサーバ定義名は，セッション情報の引き継ぎを行う Web アプリケーションの設定で使用します。セッション情報の引き継ぎを行う Web アプリケーションの設定については，「6.12　DD での定義」を参照してください。

注※2

webserver.session.server_id.value および webserver.container.server_id.value のデフォルト値には，サーバ環境の情報が含まれます。この値はクライアントに送信されるため，サーバ環境の情報を含まない値を明示的に指定することをお勧めします。

注※3

新たに負荷分散機を導入して，Web コンテナの機能で Cookie によるリクエストの振り分けをする場合，HTTP セッションを生成したリクエストのレスポンスにサーバ ID の Cookie を付加する機能を有効にして，付加するサーバ ID の Cookie の名称および Cookie の値を設定します。

既存のアプリケーションで負荷分散機用に Cookie の情報を設定している場合などは設定不要です。

6.14.3　簡易構築定義ファイルのパラメタの設定要否

メモリセッションフェイルオーバ機能を使用する場合の簡易構築定義ファイルのパラメタの設定要否を次の表に示します。

表 6-26　メモリセッションフェイルオーバ機能を使用する場合の簡易構築定義ファイルのパラメタの設定要否

簡易構築定義ファイルのパラメタ	パラメタの設定要否
webserver.sfo.sfo_servers	必須
webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.naming	必須
webserver.sfo.sfo_server.<セッションフェイルオーバ用フィルタのサーバ定義名>.connectionTimeout	任意
webserver.session.server_id.enabled	任意
webserver.session.server_id.value	任意
webserver.container.server_id.enabled	任意
webserver.container.server_id.name	任意
webserver.container.server_id.value	任意
webserver.connector.inprocess_http.limit.max_request_line	任意

（凡例）

必須：必ず指定することを示す。省略した場合はエラーとなり，SFO サーバ用アプリケーションを開始できない。

任意：省略できることを示す。省略した場合はデフォルトの値が設定される。

6.14.4　定義内容の詳細（webserver.sfo から始まるパラメタ）

メモリセッションフェイルオーバ機能を使用する場合に指定するキーの詳細について次に示します。なお「省略値」とは，キーの指定がない場合に仮定される値です。

「VR」とは，キーが導入・変更されたアプリケーションサーバのバージョンです。

キー名称	内容	省略値	VR
webserver.sfo.negotiation.ignore_serverId	ネゴシエーションで，J2EE サーバに設定された HttpSession のセッション ID に付加するサーバ ID が，すでに接続済みの J2EE サーバの設定と一致した場合に，エラーとするかどうかを指定します。 true を指定した場合： 　エラーとします。	false	−

キー名称	内容	省略値	VR
webserver.sfo.negotiation.ignore_serverId	false を指定した場合： 　エラーとしません。 このパラメタは，HA モニタを使用したクラスタ環境だけで設定してください。それ以外の環境では設定しないでください。	false	−
webserver.sfo.check_size.mode	グローバルセッション情報のサイズ見積もり機能を有効にするかどうかを指定します。 on を指定した場合： 　グローバルセッション情報のサイズ見積もり機能が有効になります。 off を指定した場合 　グローバルセッション情報のサイズ見積もり機能が無効になります。	off	07-60
webserver.sfo.sfo_server.<SFO サーバ名>※1.connectionTimeout	J2EE サーバから SFO サーバに通信した際のタイムアウト時間を，1～3600 の整数（単位：秒）で指定します。	30	06-50 07-00
webserver.sfo.sfo_server.<SFO サーバ名>※1.naming	<SFO サーバ名>が指す SFO サーバのインプロセスで起動した CORBA ネーミングサービスの IP アドレスまたはホスト名，およびポート番号を指定します。 IP アドレスは 10 進表記で，ポート番号は 1～65535 の整数で指定します。区切り文字には半角コロン（:）を使用します。 指定形式： 　<IPアドレス>:<ポート番号> 　<ホスト名>:<ポート番号> webserver.sfo.sfo_servers キーで指定した，すべての SFO サーバについて設定が必要です。設定されていない SFO サーバが存在した場合，セッションフェイルオーバを使用するアプリケーションは開始できません。 簡易構築定義ファイルで指定できる値を次に示します。 • IPv4 アドレス:ポート番号 • ホスト名:ポート番号 • @myhost:ポート番号	なし	07-50
webserver.sfo.sfo_servers※2	J2EE サーバ上の Web アプリケーションが使用する，すべての SFO サーバ名を指定します。1024 文字までの文字列が指定できます。半角英数字（A～Z，a～z，0～9）またはアンダースコア（_）が使用できます。 SFO サーバを複数指定する場合は，半角コンマ（,）で区切ります。ただし，一つの SFO サーバ名に指定できる文字列長は 32 文字までです。このとき，一つ目に指定された SFO サーバ名がデフォルトとなります。 指定形式： 　<SFOサーバ名>[,<SFOサーバ名>] 同じ SFO サーバ名を複数回指定することはできません。 セッションフェイルオーバ機能を使用しない場合，このプロパティは無効です。	なし	06-50 07-00

（凡例）

－：08-00 より前のバージョンを示します。

注※1

webserver.sfo.sfo_servers キーで指定した SFO サーバ名。webserver.sfo.sfo_servers キーに指定されていない SFO サーバ名を指定した場合，そのキーは無効になります。

注※2

param-value に値を複数指定する際は，コンマ「,」で区切って指定します。

（例）

<param-value>値 1,値 2,値 3</param-value>

このパラメタは，usrconf.properties（J2EE サーバ用ユーザプロパティファイル）にも指定できます。

6.14.5　簡易構築定義ファイルの指定例

セッション情報の引き継ぎに関連する簡易構築定義ファイルの指定例を次に示します。

```
  :
<tier>
  <tier-type>combined-tier</tier-type>
  <configuration>
    <logical-server-type>j2ee-server</logical-server-type>
    <param>
      <param-name>webserver.container.server_id.enabled</param-name>
      <param-value>true</param-value>
    </param>
    <param>
      <param-name>webserver.sfo.sfo_servers</param-name>
      <param-value>sfo_server1,sfo_server2</param-value>
    </param>
    <param>
      <param-name>webserver.sfo.sfo_server.sfo_server1.naming</param-name>
      <param-value>host1:910</param-value>
    </param>
    <param>
      <param-name>webserver.sfo.sfo_server.sfo_server1.connectionTimeout</param-name>
      <param-value>5</param-value>
    </param>
    <param>
      <param-name>webserver.sfo.sfo_server.sfo_server2.naming</param-name>
      <param-value>host2:911</param-value>
    </param>
    <param>
      <param-name>webserver.sfo.sfo_server.sfo_server2.connectionTimeout</param-name>
      <param-value>10</param-value>
    </param>
      :
  </configuration>
    :
</tier>
<tier>
  <tier-type>sfo-tier</tier-type>
  <configuration>
    <logical-server-type>sfo-server</logical-server-type>
    <param>
      :
  </configuration>
  <configuration>
    <logical-server-type>performance-tracer</logical-server-type>
    <param>
      :
  </configuration>
</tier>
  :
```

この記述例で設定される内容は，次のとおりです。

- SFO サーバとして，「sfo_server1」および「sfo_server2」の二つを使用します。

- Web アプリケーションにグローバルセッション情報を格納する SFO サーバを指定していない場合，デフォルトで「sfo_server1」を使用します。
- 「sfo_server1」の稼働ホストは「host1」であり，使用する CORBA ネーミングサービスのポート番号は「910」です。
- 「sfo_server1」との通信タイムアウトを「5 秒」に設定し，通信時間が 5 秒を超えた場合は「sfo_server1」で障害が発生したと判断します。
- 「sfo_server2」の稼働ホストは「host2」であり，使用する CORBA ネーミングサービスのポート番号は「911」です。
- 「sfo_server2」との通信タイムアウトを「10 秒」に設定し，通信時間が 10 秒を超えた場合は「sfo_server2」で障害が発生したと判断します。

6.15　SFO サーバの設定

SFO サーバが使用するネーミングサービス，PRF デーモン，ログ，セキュリティなどの基本設定や，SFO サーバの JavaVM の起動オプションの設定を変更する場合に，必要に応じて実施してください。

! 注意事項

CTM を使用している場合，CTM を使用したリクエストのスケジューリングを無効にする必要があります。論理 SFO サーバ（sfo-server）の<configuration>タグ内に，ejbserver.ctm.enabled で「true」が指定されている場合，「false」に変更してください。

6.15.1　簡易構築定義ファイルでの定義

SFO サーバの設定は，簡易構築定義ファイルで実施します。メモリセッションフェイルオーバ機能の定義は，簡易構築定義ファイルの論理 SFO サーバ（sfo-server）の<configuration>タグ内に指定します。

簡易構築定義ファイルでのメモリセッションフェイルオーバ機能の定義について次の表に示します。なお，ここで説明していないパラメタは編集しないでください。

表 6-27　簡易構築定義ファイルでのメモリセッションフェイルオーバ機能の定義

項目	指定するパラメタ	設定内容
CORBA ネーミングサービス	ejbserver.naming.port	利用する CORBA ネーミングサービスのポート番号を指定します。
EJB コンテナ	ejbserver.container.remove.scan.interval	Stateful Session Bean のタイムアウトやプールの状態を監視するスレッドの起動間隔を指定します。
Web サーバ	• ejbserver.http.port • webserver.connector.http.permitted.hosts • webserver.connector.http.bind_host	簡易 Web サーバのポート番号，簡易 Web サーバへのアクセスを許可するホストや，簡易 Web サーバで使用するホストを指定します。
	• webserver.connector.ajp13.port • webserver.connector.ajp13.bind_host • webserver.connector.ajp13.max_threads※1	Web サーバとの通信ポート番号や，Web サーバ連携で使用するホストを指定します。
PRF デーモン	ejbserver.server.prf.PRFID	利用する PRF デーモンを指定します。
リソース枯渇監視	• ejbserver.watch.enabled • ejbserver.watch.memory.enabled • ejbserver.watch.memory.interval • ejbserver.watch.memory.threshold • ejbserver.watch.memory.writefile.enabled • ejbserver.watch.fileDescriptor.enabled※2 • ejbserver.watch.fileDescriptor.interval※2 • ejbserver.watch.fileDescriptor.threshold※2 • ejbserver.watch.fileDescriptor.writefile.enabled※2 • ejbserver.watch.thread.enabled	リソース枯渇監視機能の使用の有無，リソースの監視間隔やしきい値を指定します。 リソース枯渇監視機能の設定については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「4.3.3　実行環境での設定」を参照してください。

項目	指定するパラメタ	設定内容
リソース枯渇監視	• ejbserver.watch.thread.interval • ejbserver.watch.thread.threshold • ejbserver.watch.thread.writefile.enabled • ejbserver.watch.threaddump.enabled • ejbserver.watch.threaddump.interval • ejbserver.watch.threaddump.threshold • ejbserver.watch.threaddump.writefile.enabled • ejbserver.watch.defaultRequestQueue.enabled※3 • ejbserver.watch.defaultRequestQueue.interval • ejbserver.watch.defaultRequestQueue.threshold • ejbserver.watch.defaultRequestQueue.writefile.enabled※3	リソース枯渇監視機能の使用の有無，リソースの監視間隔やしきい値を指定します。 リソース枯渇監視機能の設定については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「4.3.3　実行環境での設定」を参照してください。
SFO サーバのログ	• ejbserver.logger.enabled.* • ejbserver.logger.channels.define.＜チャネル名＞※4.filenum • ejbserver.logger.channels.define.＜チャネル名＞※4.filesize • ejbserver.logger.systemlog.enabled	SFO サーバのログのファイル面数，ファイルサイズや，ログレベルを指定します。また，SFO サーバのログのイベントログ（UNIX の場合，syslog）への出力の有無を指定します。
スレッドダンプ出力	ejbserver.server.threaddump.filenum	J2EE サーバが障害を検知して自発的にスレッドダンプを出力する場合の スレッドダンプのファイル数の上限を指定します。
TPBroker のトレース	• vbroker.orb.htc.comt.entryCount • vbroker.orb.htc.comt.fileCount • vbroker.orb.htc.tracePath	TPBroker のトレースファイルの出力先，ファイル数や，エントリ数を指定します。
TPBroker のオプション	• vbroker.se.iiop_tp.host • vbroker.se.iiop_tp.scm.iiop_tp.listener.port	EJB コンテナの通信ポート，および IP アドレスを固定する場合に，任意の値を指定します。

注※1

SFO サーバでは Web アプリケーションでリクエストを処理することがないため，このパラメタに指定する値を小さくすることで SFO サーバが作成するスレッド数を減らすことができます。値に最小値「1」を設定することをお勧めします。

注※2

UNIX の場合に指定できるパラメタです。

注※3

SFO サーバでは Web アプリケーションを実行しないため，HTTP リクエスト実行待ちキュー監視は不要です。値に「false」を設定して監視を無効にすることをお勧めします。

注※4

チャネル名には，種別ごとのチャネル名を指定します。

MessageLogFile，MaintenanceLogFile，ExceptionLogFile，ConsoleLogFile，EJBContainerLogFile，
MemoryWatchLogFile，FileDescriptorWatchLogFile，ThreadWatchLogFile，RequestQueueWatchLogFile

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

6.15.2　SFO サーバの起動オプションの設定

SFO サーバでは，Web アプリケーションのグローバルセッション情報の冗長化を開始する前に，グローバルセッション情報を格納するために必要な領域を確保します。このとき確保したメモリは，Web アプリケーションのグローバルセッション情報の冗長化が終了するまで解放されないで，JavaVM の Tenured 領域に格納されます。

JavaVM では，Tenured 領域の使用状態や，New 領域の最大値に対する Tenured 領域の空き容量などを基に，FullGC を発生させます。SFO サーバでは，FullGC の発生を防ぐために，Web アプリケーションのグローバルセッション情報の冗長化で使用する Tenured 領域メモリサイズや，SFO サーバが使用する Tenured 領域のメモリサイズなど，Tenured 領域に格納されるオブジェクトのメモリサイズを基に，JavaVM の Tenured 領域，New 領域のサイズをチューニングする必要があります。

JavaVM の Tenured 領域，New 領域のサイズは，簡易構築定義ファイルの論理 SFO サーバ（sfo-server）の<configuration>タグ内に，JavaVM 起動パラメタで定義します。JavaVM 起動パラメタの指定内容を次に示します。

<param-name>タグ

add.jvm.arg

<param-value>タグ

- -Xmx（設定値）
- -XX:NewRatio=（設定値）

! 注意事項

JavaVM の領域をチューニングする場合は，JavaVM の GC ログを確認して，SFO サーバの運用で FullGC が発生しないことを確認してください。

Web アプリケーションのグローバルセッション情報の冗長化で使用する Tenured 領域のメモリサイズ，および SFO サーバが使用する Tenured 領域のメモリサイズの見積もり式を次に示します。

- **Web アプリケーションのグローバルセッション情報の冗長化で使用する Tenured 領域のメモリサイズの見積もり式**

```
A＝ {(B＋500)×C×1.2÷(1,024×1,024)} ＋D×5
```

（凡例）

A：Web アプリケーションのグローバルセッション情報の冗長化で使用する Tenured 領域のメモリサイズ（単位：メガバイト）

B：グローバルセッション情報の最大サイズ（単位：バイト）※1

C：HTTP セッション数の上限値※2

D：接続する J2EE サーバの数

注※1

Web アプリケーションのフィルタ属性ファイルまたは DD（web.xml）の GSInfosLengthMax パラメタに指定した値です。グローバルセッション情報の最大サイズの詳細については，「6.12.2(2)　グローバルセッション情報の最大サイズ（GSInfosLengthMax）」を参照してください。

注※2

Web アプリケーションの WAR 属性ファイルの<http-session-max-number>タグに指定する値です。HTTP セッション数の上限値の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.7.5　HttpSession オブジェクト数の上限値の設定」を参照してください。

- SFO サーバが使用する Tenured 領域のメモリサイズの見積もり式

SFOサーバが使用するTenured領域のメモリサイズ＝A＋50

（凡例）

A：各 Web アプリケーションのグローバルセッション情報の冗長化で使用する Tenured 領域のメモリサイズの合計（単位：メガバイト）

6.16　Web アプリケーションの設定

メモリセッションフェイルオーバ機能を使用する場合，Web アプリケーションの設定が必要です。

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

ポイント

セッション情報の引き継ぎを行う Web アプリケーションを含む J2EE アプリケーションの開始後に Web アプリケーションの設定を変更する場合，J2EE アプリケーションを再開始する前に，SFO サーバに登録された Web アプリケーションの設定情報を削除する必要があります。SFO サーバに登録された Web アプリケーションの設定情報の削除方法については，「6.17　Web アプリケーションの設定情報の削除」を参照してください。

実行環境での Web アプリケーションの設定は，サーバ管理コマンドおよび属性ファイルで実施します。メモリセッションフェイルオーバ機能の定義には，フィルタ属性ファイルおよび WAR 属性ファイルを使用します。

フィルタ属性ファイルおよび WAR 属性ファイルで指定するタグは，DD または cosminexus.xml と対応しています。DD（web.xml）および cosminexus.xml での定義については，「6.12　DD での定義」および「6.13　cosminexus.xml での定義」を参照してください。

属性ファイルを編集してフィルタを組み込む場合，フィルタ属性ファイルおよび WAR 属性ファイルに，Web アプリケーションにセッションフェイルオーバ用フィルタを組み込むための設定をします。属性ファイルの取得，編集した属性ファイルの設定の手順については，マニュアル「アプリケーションサーバ機能解説 基本・開発編(Web コンテナ)」の「2.9.1　アプリケーションサーバが提供するサーブレットフィルタ（built-in フィルタ）」を参照してください。

6.17　Web アプリケーションの設定情報の削除

セッション情報の引き継ぎを行う Web アプリケーションを含む J2EE アプリケーションの開始時に，SFO サーバに Web アプリケーションの設定情報が登録されます。このため，J2EE アプリケーションの開始後に Web アプリケーションの設定を変更する場合，J2EE アプリケーションを再開始する前に，SFO サーバに登録された Web アプリケーションの設定情報を削除する必要があります。

ここでは，Web アプリケーションの設定情報を削除するときの手順について説明します。

Web アプリケーションの設定情報の削除手順を次の図に示します。

図 6-35　Web アプリケーションの設定情報の削除手順

図中の「構築・運用ガイド」はマニュアル「アプリケーションサーバ システム構築・運用ガイド」，「AP 設定操作ガイド」はマニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」を示します。

図中の 1.～5.について説明します。

1. 設定を変更する Web アプリケーションを含む J2EE アプリケーションをすべて停止します。

   J2EE アプリケーションの停止方法については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「10.2.2　J2EE アプリケーションの停止」を参照してください。

2. SFO サーバが配置されているサービスユニットを停止します。

   サービスユニット内の SFO サーバも停止されます。サービスユニットの停止には，Smart Composer 機能のコマンドを使用します。サービスユニットの停止方法については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.34　システムを停止する（CUI 利用時）」を参照してください。

3. Web アプリケーションの設定を変更します。

   セッション情報の引き継ぎを行う Web アプリケーションの設定については，「6.12　DD での定義」および「6.13　cosminexus.xml での定義」を参照してください。

参考

サービスユニット開始後に Web アプリケーションの設定変更をしたい場合は，「6.16　Web アプリケーションの設定」を参照してください。

4. 手順 2 で停止したサービスユニットを開始します。

サービスユニット内の SFO サーバも開始されます。SFO サーバの再開始によって, SFO サーバに登録されている Web アプリケーションの設定情報は, すべて削除されます。サービスユニットの開始には, Smart Composer 機能のコマンドを使用します。サービスユニットの開始方法については, マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.24　システムを起動する（CUI 利用時)」を参照してください。

! 注意事項

SFO サーバを再開始すると，SFO サーバに冗長化されているグローバルセッション情報もすべて削除されるため注意が必要です。また，SFO サーバで複数の Web アプリケーションのグローバルセッション情報を冗長化している場合，すべての Web アプリケーションの設定情報，およびグローバルセッション情報が削除されるため注意してください。

5. Web アプリケーションの設定変更が完了したら，J2EE アプリケーションを再開始します。

J2EE アプリケーションの開始方法については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「10.2.1　J2EE アプリケーションの開始」を参照してください。

7 拡張機能の互換機能（そのほかの機能）

この章では，拡張機能の互換機能のうち，複数の構築済み実行環境の切り替え，テストモード，および cjjsp2java コマンドについて説明します。

7.1　この章の構成

この章では，拡張機能の互換機能のうち，次の機能について説明します。

- 複数の構築済み実行環境の切り替え
- J2EE アプリケーションのテスト機能
- JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java）

この章の構成を次の表に示します。

表 7-1　拡張機能の互換機能（そのほかの機能）と参照先

タイトル	参照先
複数の構築済み実行環境の切り替え	7.2
J2EE アプリケーションのテスト機能	7.3
JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java）	7.4

7.2　複数の構築済み実行環境の切り替え

アプリケーションサーバでは，1 台のマシンに複数の構築済み実行環境を共存させ，それぞれの実行環境を切り替える機能を提供しています。構築済み実行環境の切り替えで使用できる機能には，退避機能，回復機能，削除機能，および退避環境一覧出力機能があります。構築済み実行環境の切り替えによって，同一製品のバージョンやパッチの違い，設定内容の違いなどによるシステムの動作を，1 台のマシンで検証できます。なお，1 台のマシンで同時に動作できる実行環境は一つです。

この節の構成を次の表に示します。

表 7-2　この節の構成（複数の構築済み実行環境の切り替え）

分類	タイトル	参照先
解説	構築済み実行環境の切り替えの概要	7.2.1
	構築済み実行環境の切り替えの流れ	7.2.2
設定	構築済み実行環境を切り替えるための準備	7.2.3
	構築済み実行環境の切り替え手順	7.2.4
運用	構築済み実行環境の切り替えで使用するコマンド（cosmienv（構築済み実行環境の退避，回復，削除，一覧出力)）	7.2.5
	構築済み実行環境の切り替えで出力するログの取得	7.2.6
	構築済み実行環境の切り替えで出力するログの出力形式と出力項目（UNIX の場合）	7.2.7
注意事項	構築済み実行環境を切り替える場合の注意事項	7.2.8

注　「実装」について，この節での説明はありません。

! 注意事項

構築済み実行環境の切り替えは，UNIX 環境の場合に使用できます。退避機能と回復機能は，同じ OS の同じバージョンのマシン環境上で実行してください。切り替え対象となる実行環境を構築する製品は，Application Server です。ただし，構成ソフトウェアのうち，Component Container - Redirector は構築済み実行環境の切り替え対象外です。

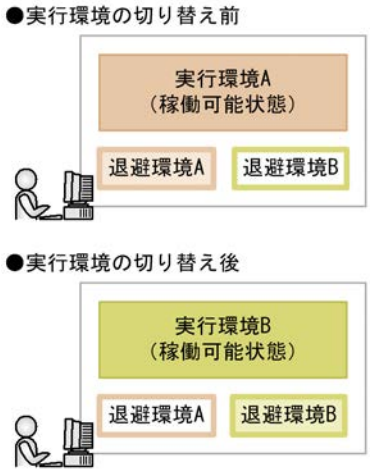
7.2.1　構築済み実行環境の切り替えの概要

ここでは，構築済み実行環境の切り替えの概要について説明します。

構築済み実行環境を切り替える場合，構築済みの実行環境ごとに退避環境を作成します。**退避環境**は，実行環境を圧縮して，稼働可能状態から退避状態（稼働できない状態）に遷移した環境です。複数の退避環境のうち，使用したい退避環境を稼働可能状態に回復して，実行環境を切り替えます。

複数の構築済み実行環境の切り替えの概要を次の図に示します。

図 7-1　複数の構築済み実行環境の切り替えの概要



この図では，実行環境 A に対する退避環境 A と，実行環境 B に対する退避環境 B をそれぞれ作成して，実行環境を切り替えています。作成済みの退避環境は，一覧で出力したり，不要となった場合に削除したりできます。

次に，構築済み実行環境を切り替える場合の前提条件，切り替え時に使用する機能，および取得するログファイルについて説明します。

(1)　前提条件

切り替え対象となる実行環境を構築する製品は，Application Server です。構築済み実行環境の切り替えは，UNIX 環境の場合に使用できます。退避対象となる実行環境は，システムの構築が完了済みで，各サーバやサービスを起動していない状態（稼働可能状態）であることが前提です。

また，構築済み実行環境から退避環境を作成する場合に，実行環境は TAR 形式または CPIO 形式に圧縮されます。このため，実行環境の切り替えは，TAR 形式または CPIO 形式に対応している環境で実行する必要があります。

(2)　構築済み実行環境の切り替えで使用する機能

構築済み実行環境を切り替える場合，次の四つの機能を使用します。

- 退避機能

  構築済み実行環境を TAR 形式または CPIO 形式のファイルに圧縮して，退避環境を作成します。退避機能を実行しても，構築済み実行環境はそのまま残ります。退避環境は，複数作成でき，退避機能実行時に指定する識別子で管理できます。

- 回復機能

  退避環境を回復します。回復機能を実行しても，回復元の退避環境はそのまま残ります。実行環境の回復完了後に，回復元の退避環境を削除したい場合は，削除機能を実行してください。回復機能実行前には Application Server が提供するすべての製品はアンインストール済みである必要があります。

- 削除機能

  退避環境を削除します。退避環境が不要となった場合に実施します。

- 退避環境一覧出力機能

  現在退避している退避環境の一覧を出力します。

  退避環境一覧出力機能では，次の項目を出力します。

- 退避環境の識別子
- 退避環境の格納先パス
- 退避環境を作成した日付と時刻
- 退避環境に退避した製品の形名，バージョン情報および製品名

これらの機能は，cosmienv コマンドに，機能に対応する引数を指定して実行します。cosmienv コマンドの詳細，および退避環境一覧の出力形式については，「7.2.5　構築済み実行環境の切り替えで使用するコマンド（cosmienv（構築済み実行環境の退避，回復，削除，一覧出力））」を参照してください。

(3)　構築済み実行環境の切り替え実行時に障害が発生した場合の対処

構築済み実行環境を切り替える場合は，cosmienv コマンドを実行します。cosmienv コマンド実行時に障害が発生した場合は，メッセージと，ログファイルの内容を基に障害を解析してください。ログファイルは機能ごとに取得され，次の場所に格納されます。

構築済み実行環境の切り替えで取得するログファイルの格納場所

/opt/Cosminexus/env/log

機能ごとに出力されるログファイルを次の表に示します。

表 7-3　機能ごとに出力されるログファイル

機能の種類	ログファイル名
退避機能	cosmienv_s[n].log
回復機能	cosmienv_r[n].log
削除機能	cosmienv_d[n].log

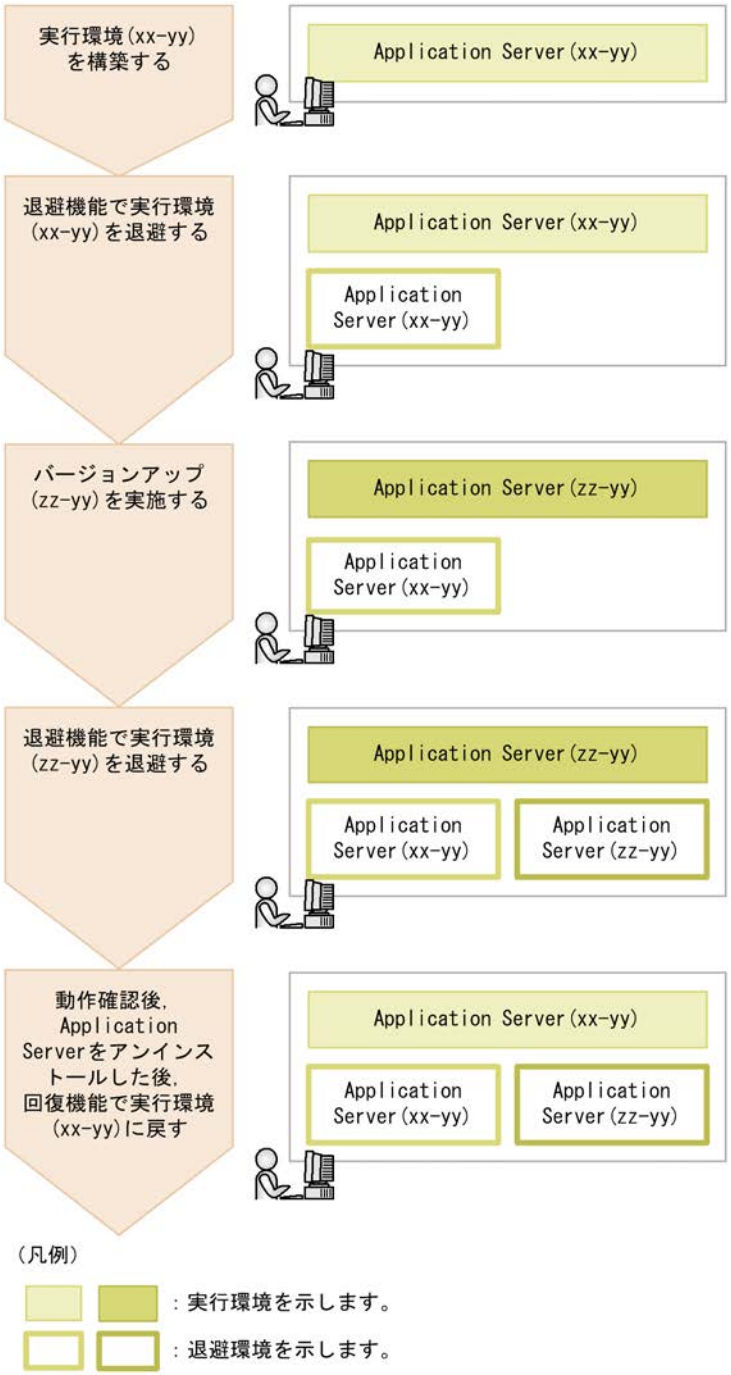
なお，退避環境一覧出力機能の場合，ログファイルは出力されません。

ログファイルの内容については，「7.2.7　構築済み実行環境の切り替えで出力するログの出力形式と出力項目（UNIX の場合）」を参照してください。また，メッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。

7.2.2　構築済み実行環境の切り替えの流れ

ここでは，構築済み実行環境を切り替える場合の流れについて説明します。構築済み実行環境の切り替えの流れを次の図に示します。

図 7-2　構築済み実行環境の切り替えの流れ



この図では，1 台のマシンで，実行環境のバージョンアップを実施し，バージョンアップ後の実行環境で動作確認をしたあと，バージョンアップ前の実行環境に戻しています。この場合，退避機能を使用して，各バージョンの実行環境を退避しているので，動作確認中に障害や不具合が発生しても，再インストールや環境設定などの作業を実施しないで，退避した実行環境に回復できます。

退避機能を使用すると，次に示す実行環境を 1 台のマシンに共存できます。

- 製品のバージョンが異なる実行環境
- 製品のバージョンが同じで，設定が異なる実行環境

- 製品のバージョンが同じで，適用しているパッチが異なる実行環境

7.2.3　構築済み実行環境を切り替えるための準備

ここでは，構築済み実行環境を切り替えるために必要な準備について説明します。

準備として次の作業が必要です。

- 退避対象外のファイルの退避
- ディスク容量の確認

(1)　退避対象外のファイルの退避

構築済み実行環境を切り替える場合，cosmienv コマンドを使用して，構築済み実行環境から退避環境を作成します。cosmienv コマンドによって作成される退避環境には，ユーザ任意の定義ファイルや，ログファイル，プロパティの設定によって作成されたファイルなどは含まれません。このため，cosmienv コマンドを実行する前に，退避対象外のファイルのうち，アプリケーションサーバの動作に必要なファイルは，あらかじめ退避しておく必要があります。

なお，実行環境の回復時には，退避したファイルを回復した実行環境に戻す作業が必要になります。

(a)　退避作業が必要なファイル

退避対象外のファイルのうち，退避作業が必要なファイルと作業方法を次の表に示します。これらのファイルは，アプリケーションサーバの動作に影響があるため，必ず退避作業を実施してください。なお，実行環境の回復時には，退避したファイルを回復した実行環境の退避元フォルダに戻す作業が必要になります。

表 7-4　退避作業が必要なファイルと作業方法

項番	ファイルまたはディレクトリ名	作業方法	備考	構成ソフトウェア
1	簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の webserver.connector.inprocess_http.error_custom.<エラーページカスタマイズ定義名>.file パラメタに指定しているファイル	退避してください。	エラーステータスコードに対応するファイルを送信する場合に，レスポンスボディとしてクライアントに返すファイル	Component Container
2	簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の webserver.connector.inprocess_http.redirect.<リダイレクト定義名>.file で指定しているファイル	退避してください。	特定のファイルをレスポンスとしてクライアントに返す場合にレスポンスボディとして使用するファイル	Component Container
3	展開ディレクトリ形式のアプリケーション	退避してください。	－	Component Container
4	コンテナ拡張ライブラリ	退避してください。	－	Component Container
5	アプリケーションサーバの構築・運用・アプリケーション開発で使用するファイルの形式や定義で，パスにデフォルト以外を設定しているディレクトリおよびファイル	退避してください。	－	Component Container

項番	ファイルまたはディレクトリ名	作業方法	備考	構成ソフトウェア
6	Reliable Messaging の管理情報テーブル	退避してください。	DBMS 上に作成した Reliable Messaging の管理情報テーブル	Reliable Messaging
7	Reliable Messaging の QueueConfigFileName プロパティに指定しているファイル	プロパティを指定している場合は，退避してください。	Reliable Messaging のキュー定義ファイル	Reliable Messaging
8	Reliable Messaging の RMTRTransferControlDir プロパティに指定しているディレクトリ	プロパティを指定している場合は，退避してください。	Reliable Messaging で使用するクライアント定義ファイルを格納したディレクトリ	Reliable Messaging
9	Reliable Messaging の QueueMakeFileName プロパティに指定しているファイル	プロパティを指定している場合は，退避してください。	Reliable Messaging のキュー作成ファイル	Reliable Messaging
10	Web サービスセキュリティの環境設定ファイル（cwsscfg.properties）の cwss.binding.KeyLocator.KeystoreDir プロパティに指定しているディレクトリ以下すべて	プロパティを指定している場合は，退避してください。	署名付与/検証で使用するキーストアファイルを格納するディレクトリ	Web Services - Security
11	Web サービスセキュリティの環境設定ファイル（cwsscfg.properties）の cwss.binding.KeyLocator.CertificateDir プロパティに指定しているディレクトリ以下すべて	プロパティを指定している場合は，退避してください。	証明書検証で使用する証明書ファイルを格納するディレクトリ	Web Services - Security
12	Web サービスセキュリティの環境設定ファイル（cwsscfg.properties）の cwss.binding.KeyLocator.SecretKeyDir プロパティに指定しているディレクトリ以下すべて	プロパティを指定している場合は，退避してください。	暗号化/復号化で使用する共通鍵ファイルを格納するディレクトリ	Web Services - Security
13	環境変数「VBROKER_ADM」または TPBroker の vbroker.orb.admDir プロパティに/opt/Cosminexus/TPB/adm 以外を設定している場合に，環境変数「VBROKER_ADM」または vbroker.orb.admDir プロパティに設定しているディレクトリ以下にある agentaddr，localaddr，HVMGTEE_DEF，htc.props，htc.clienthandleraddr，および HVIORB_DEF	環境変数またはプロパティを指定している場合は，退避してください。	ユーザ作成の定義ファイル	TPBroker
14	環境変数「OSAGENT_ADDR_FILE」または vbroker.agent.addrFile プロパティに指定しているファイル	環境変数またはプロパティを指定している場合は，退避してください。	ユーザ作成の定義ファイル	TPBroker

項番	ファイルまたはディレクトリ名	作業方法	備考	構成ソフトウェア
15	環境変数「OSAGENT_LOCAL_FILE」，または vbroker.agent.localFile プロパティに指定しているファイル	環境変数またはプロパティを指定している場合は，退避してください。	ユーザ作成の定義ファイル	TPBroker
16	-ORBpropStorage オプション，または vbroker.orb.propStorage プロパティで指定しているファイル	オプションまたはプロパティを指定している場合は，退避してください。	ユーザ作成の定義ファイル	TPBroker
17	httpsd.conf で指定した各種設定ファイル（Include ディレクティブ，TypeConfig ディレクティブ，SSL による暗号化，認証の定義など）のうち，opt/hitachi/httpsd 以外にあるファイルすべて	opt/hitachi/httpsd 以外を指定している場合は，退避してください。	－	HTTP Server
18	httpsd.conf で指定したコンテンツディレクトリのうち，opt/hitachi/httpsd 以外にあるディレクトリすべて	opt/hitachi/httpsd 以外を指定している場合は，退避してください。	－	HTTP Server

(b)　退避作業が不要なファイル

退避対象外のファイルのうち，退避作業が不要なファイルを次の表に示します。これらのファイルは，アプリケーションサーバの動作に影響はありません。必要に応じて退避してください。なお，退避作業が不要なファイルの場合，実行環境の回復時に退避したファイルを戻す作業は不要です。

表 7-5　退避作業が不要なファイル

項番	ファイルまたはディレクトリ名	説明	構成ソフトウェア
1	/opt/Cosminexus/XMLSEC/logs/*	Web Services - Security に含まれる XML Security - Core のログ	Web Services - Security
2	/opt/Cosminexus/env 以下すべて	構築済み実行環境の切り替え機能のコマンド（cosmienv コマンド）の格納先ディレクトリ	－
3	<製品の作業ディレクトリ>/ejb/<サーバ名称>/logs 以下すべて /opt/Cosminexus/CC/server/public/ejb/<サーバ名称>/logs 以下すべて /opt/Cosminexus/CC/server/public/public_old/ejb/<サーバ名称>/logs 以下すべて	J2EE サーバのログ出力先ディレクトリ※	Component Container
4	<製品の作業ディレクトリ>/ejb/<サーバ名称>/以下の core*，および javacore* /opt/Cosminexus/CC/server/public/ejb/<サーバ名称>/以下の core*，および javacore*	J2EE サーバの core ファイル，および JavaCore ファイル※	Component Container

項番	ファイルまたはディレクトリ名	説明	構成ソフトウェア
4	/opt/Cosminexus/CC/server/public/public_old/ejb/<サーバ名称>/以下の core*，および javacore*	J2EE サーバの core ファイル，および JavaCore ファイル※	Component Container
5	<製品の作業ディレクトリ>/ejb/<サーバ名称>/stats 以下すべて /opt/Cosminexus/CC/server/public/ejb/<サーバ名称>/stats 以下すべて /opt/Cosminexus/CC/server/public/public_old/ejb/<サーバ名称>/stats 以下すべて	J2EE サーバと SFO サーバの稼働情報ファイルの格納先ディレクトリ※	Component Container
6	/opt/Cosminexus/CC/logs 以下すべて	Component Container のログ格納用ディレクトリ	Component Container
7	/opt/Cosminexus/CC/admin/logs 以下すべて	サーバ管理コマンドのログ格納用ディレクトリ	Component Container
8	/opt/Cosminexus/CC/client/logs 以下すべて	EJB クライアントアプリケーションのログ出力用ディレクトリ	Component Container
9	/opt/Cosminexus/CC/web/containers/<サーバ名称>/以下の core*，および javacore*	Web コンテナサーバの core ファイル，および JavaCore ファイル	Component Container
10	/opt/Cosminexus/CC/web/containers/<サーバ名称>/logs 以下すべて	Web コンテナサーバのログ出力先ディレクトリ	Component Container
11	/opt/Cosminexus/manager/log 以下すべて	Component Container の運用管理機能のログ出力先ディレクトリ	Component Container
12	/opt/Cosminexus/manager/tmp	Component Container の運用管理機能の一時的なワークディレクトリ	Component Container
13	/opt/Cosminexus/manager/bin/以下の core*，および javacore*	Component Container の運用管理機能の core ファイル，および JavaCore ファイル	Component Container
14	/opt/Cosminexus/sinagent/log	サーバ通信エージェントのログ出力先ディレクトリ	Component Container
15	/opt/Cosminexus/CC/web/redirector/logs および/opt/Cosminexus/CC/web/redirector/servers/<論理 Web サーバ名>/logs 以下すべて	リダイレクタのメッセージログ出力先ディレクトリおよび保守用トレースログ出力先ディレクトリ	Component Container
16	<製品の作業ディレクトリ>/ejb/<サーバ名称>/otsstatus 以下すべて /opt/Cosminexus/CC/server/public/ejb/<サーバ名称>/otsstatus 以下すべて /opt/Cosminexus/CC/server/public/public_old/ejb/<サーバ名称>/otsstatus 以下すべて	インプロセストランザクションサービスのステータスファイルの格納ディレクトリ※	Component Container
17	/var/spool/TPBrokerOTM 以下すべて	ドメイン情報ファイル格納ディレクトリ	Component Transaction Monitor

項番	ファイルまたはディレクトリ名	説明	構成ソフトウェア
18	CTMSPOOL 環境変数で指定したディレクトリ以下すべて	CTMSPOOL ディレクトリ	Component Transaction Monitor
19	PRFSPOOL 環境変数で指定したディレクトリ以下すべて	PRFSPOOL ディレクトリ	Performance Tracer
20	/opt/Cosminexus/RM/logs 以下すべて	Reliable Messaging コマンドのログ出力先ディレクトリ	Reliable Messaging
21	/opt/Cosminexus/c4web/logs 以下すべて	Component Container の SOAP アプリケーション開発支援機能のログ出力先ディレクトリ	Web Services - Security
22	/opt/Cosminexus/TPB/adm/hgtfifo/以下すべて	バーボースログ出力用一時ディレクトリ	TPBroker
23	/opt/Cosminexus/TPB/adm/impl_dir/以下すべて	インプリメンテーションリポジトリデータ作成ディレクトリ	TPBroker
24	/opt/Cosminexus/TPB/log/以下すべて	C++ORB 機能トレース出力ディレクトリ	TPBroker
25	/opt/Cosminexus/TPB/logj/以下すべて	JavaORB 機能トレース出力ディレクトリ	TPBroker
26	/opt/hitachi/httpsd/logs 以下すべて	HTTP Server のログ出力先ディレクトリ	HTTP Server
27	/opt/hitachi/httpsd/servers/<サーバ名称>/logs 以下すべて	HTTP Server のログ出力先ディレクトリ	HTTP Server
28	/opt/Cosminexus/wss/logs 以下すべて	Web Services - Security のログ出力先ディレクトリ	Web Services - Security

（凡例）－：Application Server の製品が該当します。

注　表内のディレクトリの意味を次に示します。

- <製品の作業ディレクトリ>

  <製品の作業ディレクトリ>は，簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の ejb.public.directory パラメタに設定されているディレクトリです。デフォルト値は，「/opt/Cosminexus/CC/server/public」です。

注※　次のディレクトリも含まれます。

- <製品の作業ディレクトリ>を変更する前に使用したデフォルトの作業ディレクトリ
- アップグレードインストール後に移行した際に作成されるデフォルトのバックアップディレクトリ

(2)　ディスク容量の確認

構築済み実行環境から退避環境を作成する場合，退避対象ファイルをすべてアーカイブしたファイルが，退避環境として作成されます。このため，実行するマシンに十分なディスク容量があるかを確認してください。退避環境に必要なディスク容量は，構築済みの実行環境の状態によって変わります。製品の新規インストール直後の状態の必要なディスク占有量についてはリリースノートをご確認ください。

注意事項

簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の ejbserver.distributedtx.ots.status.directory1 パラメタ，または ejbserver.distributedtx.ots.status.directory2 パラメタで指定したディレクトリは，相対パスで指定している場合は退避されます。この場合，ディスク容量は，ejbserver.distributedtx.ots.status.directory1 パラメタ，または ejbserver.distributedtx.ots.status.directory2 パラメタで指定しているディレクトリが使用するディスク容量を加えた値で見積もる必要があります。

7.2.4　構築済み実行環境の切り替え手順

ここでは，構築済み実行環境を切り替える場合の手順と，退避環境作成時に退避されるファイルについて説明します。

(1)　構築済み実行環境の切り替え手順の例

ここでは，1 台のマシンに実行環境 A を構築し，実行環境 B にバージョンアップして動作検証したあと，実行環境 A に戻す手順を例に説明します。

各実行環境の設定内容の例を次の表に示します。

表 7-6　各実行環境の設定内容の例

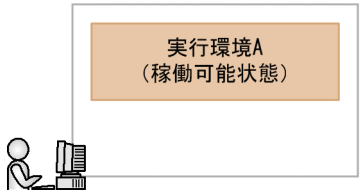
設定内容	実行環境 A	実行環境 B
退避環境の格納先	/home/CosmiBackup	
退避対象外のファイルの格納先	/home/CosmiBackup/apsvAfiles	/home/CosmiBackup/apsvBfiles
退避環境の識別子	apsvA	apsvB

実行環境の切り替え手順を次に示します。

1. 実行環境 A を構築します。

   Application Server をインストールして，環境設定を実施し，システムを構築します。

   実行環境 A が稼働可能状態となります。



2. 実行環境 A の退避対象外のファイルを任意の場所に格納します。

   ここでは，「/home/CosmiBackup/apsvAfiles」に格納します。退避対象外のファイルについては，「7.2.3(1) 退避対象外のファイルの退避」を参照してください。

3. 実行環境 A を退避するために必要なディスク容量を確認します。

   製品ごとに必要なディスク容量については，「7.2.3(2) ディスク容量の確認」を参照してください。

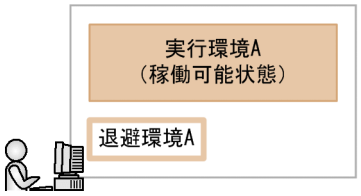
4. cosmienv コマンドに引数-s を指定して，実行環境 A の退避環境を作成します。

   コマンドの入力例を次に示します。

   cosmienv -s -I apsvA -P /home/CosmiBackup

   退避環境作成時に退避されるファイルについては，「7.2.4(2) 退避環境作成時に退避対象となるファイル」を参照してください。

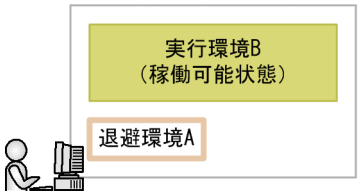
cosmienv コマンドを実行しても，実行環境 A は稼働可能状態のまま残ります。



5. 実行環境 B を構築します。

実行環境 A にアップグレードインストールと環境設定を実施します。

実行環境 B が稼働可能状態となり，実行環境 A は退避環境だけとなります。



6. 実行環境 B の退避対象外のファイルを任意の場所に格納します。

ここでは，「/home/CosmiBackup/apsvBfiles」に格納します。退避対象外のファイルについては，「7.2.3(1) 退避対象外のファイルの退避」を参照してください。

7. 実行環境 B を退避するために必要なディスク容量を確認します。

製品ごとに必要なディスク容量については，「7.2.3(2) ディスク容量の確認」を参照してください。

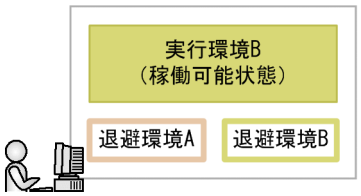
8. cosmienv コマンドに引数-s を指定して，実行環境 B の退避環境を作成します。

コマンドの入力例を次に示します。

```
cosmienv -s -I apsvB -P /home/CosmiBackup
```

退避環境作成時に退避されるファイルについては，「7.2.4(2) 退避環境作成時に退避対象となるファイル」を参照してください。

cosmienv コマンドを実行しても，実行環境 B は稼働可能状態のまま残ります。



9. 実行環境 B で動作検証をします。

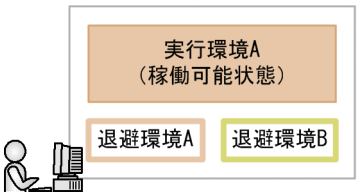
10. 実行環境 B（Application Server）をアンインストールします。

11. cosmienv コマンドに引数-r を指定して，実行環境 A に戻します。

コマンドの入力例を次に示します。

```
cosmienv -r -I apsvA
```

実行環境 A が稼働可能状態となり，実行環境 B は退避環境だけとなります。回復元の退避環境 A はそのまま残ります。



12. 手順 2.で退避したファイルを該当個所（退避元ディレクトリ）に格納します。

退避対象外のファイルを実行環境 A の退避元ディレクトリにそれぞれ格納します。

cosmienv コマンドの詳細については，「7.2.5　構築済み実行環境の切り替えで使用するコマンド（cosmienv（構築済み実行環境の退避，回復，削除，一覧出力））」を参照してください。

(2)　退避環境作成時に退避対象となるファイル

退避環境作成時に退避対象となるファイルを次の表に示します。

表 7-7　退避環境作成時に退避対象となるファイル

項番	ファイルまたはディレクトリ名	説明	構成ソフトウェア
1	/opt/Cosminexus/XMLSEC	Web Services - Security に含まれる XML Security - Core の製品，設定ファイルなど	Web Services - Security
2	/opt/Cosminexus/.HTC_xxxx.inf	製品情報ファイル	−
3	/opt/Cosminexus/common 以下すべて	共通モジュール群	−
4	/opt/Cosminexus/CC	Component Container のインストールディレクトリ	Component Container
5	/opt/Cosminexus/CWC	Component Container の Web アプリケーション開発に利用する機能のインストールディレクトリ	Component Container
6	/opt/Cosminexus/c4web	Component Container の SOAP アプリケーション開発支援機能，SOAP 通信基盤のファイルがインストールされているディレクトリ	Component Container
7	/opt/Cosminexus/manager	Component Container の運用管理機能のインストールディレクトリ	Component Container
8	/sbin/init.d/MngSvr	インストール時に作成される Component Container の運用管理機能の rc スクリプトファイル（HP-UX の場合）	Component Container
9	/sbin/init.d/AdminAgent	インストール時に作成される Component Container の運用管理機能の rc スクリプトファイル（HP-UX の場合）	Component Container
10	/etc/rc.d/init.d/MngSvr	インストール時に作成される Component Container の運用管理機能の rc スクリプトファイル（Linux の場合）	Component Container

項番	ファイルまたはディレクトリ名	説明	構成ソフトウェア
11	/etc/rc.d/init.d/AdminAgent	インストール時に作成される Component Container の運用管理機能の rc スクリプトファイル（Linux の場合）	Component Container
12	/opt/Cosminexus/CTM/bin 以下すべて	Component Transaction Monitor のロードモジュール	Component Transaction Monitor
13	/opt/Cosminexus/CTM/lib 以下すべて	Component Transaction Monitor のライブラリ	Component Transaction Monitor
14	/opt/Cosminexus/CTM/.HTC_xxxx.inf	Component Transaction Monitor の製品情報ファイル	Component Transaction Monitor
15	/opt/Cosminexus/PRF/bin 以下すべて	Performance Tracer のロードモジュール	Performance Tracer
16	/opt/Cosminexus/PRF/lib 以下すべて	Performance Tracer のライブラリ	Performance Tracer
17	/opt/Cosminexus/PRF/.HTC_xxxx.inf	Performance Tracer の製品情報ファイル	Performance Tracer
18	/opt/Cosminexus/jaxp	XML Processor のインストールディレクトリ	XML Processor
19	/opt/Cosminexus/jdk	Developer's Kit for Java のインストールディレクトリ	Developer's Kit for Java
20	/opt/Cosminexus/RM	Reliable Messaging のインストールディレクトリ	Reliable Messaging
21	/opt/Cosminexus/TPB 以下すべて	TPBroker インストールディレクトリ	TPBroker
22	/opt/hitachi/httpsd	HTTP Server のインストールディレクトリ	HTTP Server
23	簡易構築定義ファイルの論理 J2EE サーバ（j2ee-server）の ejb.public.directory パラメタで指定するディレクトリパス	J2EE サーバの作業ディレクトリ	Component Container
24	/opt/Cosminexus/wss	Web Services - Security のインストールディレクトリ	Web Services - Security
25	/opt/Cosminexus/sinagent	サーバ通信エージェントのインストールディレクトリ	Component Container
26	/etc/rc.d/init.d/SINaviAgent	インストール時に作成されるサーバ通信エージェントの rc スクリプトファイル（Linux の場合）	Component Container

（凡例）－：Application Server の製品が該当します。

7.2.5　構築済み実行環境の切り替えで使用するコマンド（cosmienv（構築済み実行環境の退避，回復，削除，一覧出力））

構築済み実行環境の切り替えでは，次のコマンドを使用します。

表 7-8　構築済み実行環境の切り替えで使用するコマンド

コマンド名称	分類	概要
cosmienv	構築済み実行環境の退避，回復，削除，一覧出力	実行環境の退避・回復を行います。

コマンドの格納先

構築済み実行環境の切り替えで使用するコマンドは，次のディレクトリに格納されています。

/opt/Cosminexus/env/bin/

以降にコマンドの文法を示します。

● 形式

```
cosmienv <オプション> [-I<識別子>]
                      [-P<退避パス>]
                      [-f]
```

● 機能

実行環境の退避・回復，退避環境の削除または一覧を表示します。

● 引数

<オプション>

-s

構築済み実行環境を退避します。

-d

退避環境を削除します。

-r

退避環境を回復します。

-l

現在の退避環境の一覧を表示します。

-h

cosmienv コマンドのヘルプを表示します。

-I<識別子>

退避環境を識別するための識別子を指定します。

20 文字までの半角英数字で指定します。

-s，-d，-r オプションを使用する場合には，識別子の指定は必須です。

-P<退避パス>

構築済み実行環境を退避するパス名をフルパスで指定します。

半角空白，全角空白，タブ文字を含むパス名は指定できません。

構築済み実行環境の退避先は <退避パス>/<識別子> となります。

-s オプションを使用する場合には，退避パスの指定は必須です。

-f

-r オプションを指定した場合に，退避環境を強制的に回復することを指定します。

-f オプションを指定した場合，Application Server のインストール先にファイルが残っていても，退避環境の回復が実行されます。

-f オプションを指定しない場合，Application Server のインストール先にファイルが残っていると，退避環境の回復は実行されません。

-r オプション以外のオプションと組み合わせて指定した場合，無効です。

● 出力形式

退避環境の一覧表示

```
aa…aa bb…bb cccc/dd/ee ff:gg:hh ii…ii jj…jj kk…kk
```

- aa…aa　退避機能実行時に指定した識別子
- bb…bb　退避機能実行時に指定した退避パス
- cccc　退避機能実行時の時刻（年）
- dd　退避機能実行時の時刻（月）
- ee　退避機能実行時の時刻（日）
- ff　退避機能実行時の時刻（時）
- gg　退避機能実行時の時刻（分）
- hh　退避機能実行時の時刻（秒）
- ii…ii　退避した製品の形名
- jj…jj　退避した製品の VV-RR-SS
- kk…kk　退避した製品名

● 入力例・出力例

入力例

退避環境の一覧を表示します。

```
cosmienv -l
```

出力例

```
KEUC00001-I The cosmienv -l command will now start.
UCAPSE001 /home/CosmiBackup 2007/03/23 15:02:36 P-1M43-7K71 07-50 uCosminexus Application
Server Enterprise
UCAPSS001 /home/CosmiBackup 2007/03/23 16:06:08 P-1M43-7D71 07-50 uCosminexus Application
Server Standard
KEUC00002-I The cosmienv -l command ended successfully.
```

● 戻り値

コマンドの戻り値を次の表に示します。

表 7-9　cosmienv コマンドの戻り値一覧

戻り値	意味
0	正常終了しました。
128	異常終了しました。

戻り値	意味
129	構築済み実行環境の退避・回復の元情報が正しくありません。
130	構築済み実行環境の退避・回復の後処理で障害が発生しました。
131	構築済み実行環境の退避・回復のための情報が正しくありません。
133	ディスクのアクセスでエラーが発生しました。
135	ファイルのオープンでエラーが発生しました。
138	引数の指定に誤りがあります。
139	製品がサポートしていない機能を指定しています。
140	コマンド実行中です。
141	コマンドの実行環境が正しくありません。
142	コマンドの実行権限がありません。
143	製品の構成ソフトウェアで障害が発生しました。 構成ソフトウェア※の戻り値が付与されています。

注※　障害が発生した構成ソフトウェアについては，コマンドで出力されるログファイルで確認してください。

戻り値「143」で付与される構成ソフトウェア別の戻り値を次の表に示します。

表 7-10　構成ソフトウェア別の戻り値一覧

構成ソフトウェア	戻り値	意味
Component Container	145	HNTRLib のインストールに失敗しました。
	146	退避に失敗しました。
	147	異常終了しました。
Component Transaction Monitor	129	退避・回復の元情報が正しくありません。
	130	退避・回復の後処理でエラーが発生しました。
	131	退避・回復のための情報が正しくありません。
	138	引数の指定に誤りがあります。
	142	実行権限がありません。
	146	退避先ディレクトリがありません。
	147	そのほかのエラーが発生しました。
	148	ファイルおよびディレクトリの削除に失敗しました。
DABroker Library	128	異常終了しました。
	131	退避・回復の元情報が正しくありません。
	138	引数の指定に誤りがあります。
	144	DABroker Library が実行中です。

構成ソフトウェア	戻り値	意味
Developer's Kit for Java	131	構築済み実行環境の退避・回復のための情報が正しくありません。
	138	引数の指定に誤りがあります。
	144	回復に失敗しました。
	145	/opt/Cosminexus/jdk がシンボリックリンクファイルです。
	146	そのほかのエラーが発生しました。
Performance Tracer	129	退避・回復の元情報が正しくありません。
	130	退避・回復の後処理でエラーが発生しました。
	131	退避・回復のための情報が正しくありません。
	138	引数の指定に誤りがあります。
	142	実行権限がありません。
	145	HCCLib のインストールに失敗しました。
	146	退避先のディレクトリがありません。
	147	そのほかのエラーが発生しました。
	148	ファイルおよびディレクトリの削除に失敗しました。
Reliable Messaging	129	退避・回復の元情報が正しくありません。
	130	退避・回復の後処理でエラーが発生しました。
	144	HNTRLib2 のインストールに失敗しました。
	145	退避先のディレクトリがありません。
	146	そのほかのエラーが発生しました。
TPBroker	129	退避・回復の元情報が正しくありません。
	130	退避・回復の後処理でエラーが発生しました。
	131	退避・回復のための情報が正しくありません。
	133	ディスクのアクセスでエラーが発生しました。
	135	ファイルのオープンでエラーが発生しました。
	144	HCCLib のインストールに失敗しました。
	145	HNTRLib2 のインストールに失敗しました。
	146	退避に失敗しました。
	147	PP インストーラ情報の取得・設定に失敗しました。
Web Services - Security	129	退避・回復の元情報が正しくありません。
	130	退避・回復の後処理でエラーが発生しました。

構成ソフトウェア	戻り値	意味
Web Services - Security	131	退避・回復のための情報が正しくありません。
	138	引数の指定に誤りがあります。
	146	退避に失敗しました。
	147	そのほかのエラーが発生しました。
	148	ファイルおよびディレクトリの削除に失敗しました。
XML Processor	128	異常終了しました。
	138	引数の指定に誤りがあります。
	144	退避に失敗しました。
	145	HNTRLib2 のインストールに失敗しました。
HTTP Server	128	異常終了しました。
	138	引数の指定に誤りがあります。
	144	退避に失敗しました。

● **注意事項**

- このコマンドはスーパーユーザで実行してください。
- 退避機能実行時は，アプリケーションサーバが提供するすべてのプログラムが停止中であることを確認してください。
- 回復機能実行時は，アプリケーションサーバが提供するすべての製品がアンインストール済みであることを確認してください。
- すべての製品をアンインストールし，かつアプリケーションサーバのインストール先にファイルやディレクトリが残った状態で回復機能を実行したい場合は，-f オプションを指定してください。この場合，回復処理によってインストール先に残ったファイルが更新されることがあります。
- 退避機能で退避する範囲以外でアプリケーションサーバの動作に必要なものは，ユーザで退避してください。退避機能で退避する範囲およびユーザで退避が必要なファイルについては，「7.2.3(1) 退避対象外のファイルの退避」を参照してください。

7.2.6　構築済み実行環境の切り替えで出力するログの取得

構築済み実行環境を cosmienv コマンドで切り替えた場合に出力されるログファイルについて説明します。cosmienv コマンドで出力するログファイルは，コマンド実行時に指定する引数（機能）によって異なります。各ログファイルの出力先は「/opt/Cosminexus/env/log」です。ログファイルのファイルサイズは 100 キロバイト以内です。また，ファイル面数は 2 面です。コマンド実行時に指定する引数（機能）ごとのログファイルを次に示します。

- 引数に-s を指定した場合（退避機能の場合）
  cosmienv_s[2].log
- 引数に-d を指定した場合（削除機能の場合）
  cosmienv_d[2].log

- 引数に-r を指定した場合（回復機能の場合）
  cosmienv_r[2].log

7.2.7　構築済み実行環境の切り替えで出力するログの出力形式と出力項目（UNIX の場合）

構築済み実行環境を cosmienv コマンドで切り替えた場合に出力される，ログファイルの出力形式および出力項目について説明します。

cosmienv コマンドのログは，独自形式で出力されます。

cosmienv コマンド実行時に出力されるログファイルには，cosmienv_s[2].log，cosmienv_d[2].log，および cosmienv_r[2].log があります。これらのファイルの出力形式および出力項目はすべて同じです。

cosmienv コマンド実行時に出力されるログファイルの出力形式は次のとおりです。

```
 YYYY/MM/DD HH:MM:SS 製品の形名 製品名 VVRRSS START
 YYYY/MM/DD HH:MM:SS 構成ソフトウェアの形名 START 起動時のコマンドと引数
[YYYY/MM/DD HH:MM:SS 構成ソフトウェアの保守情報]
 YYYY/MM/DD HH:MM:SS 構成ソフトウェアの形名 END 構成ソフトウェアの戻り値
  :
 YYYY/MM/DD HH:MM:SS 製品の形名 製品名 VVRRSS END cosmienvコマンドの戻り値
```

cosmienv コマンドのログの出力項目を次に示します。

表 7-11　cosmienv コマンドのログの出力項目

項目名	説明
YYYY/MM/DD	cosmienv コマンドを実行した日付が出力されます。
HH:MM:SS	cosmienv コマンドを実行した時刻が出力されます。
製品の形名	退避，回復，または削除の対象とした実行環境の形名が出力されます。
製品名	退避，回復，または削除の対象とした実行環境の製品名が出力されます。
VVRRSS	退避，回復，または削除の対象とした実行環境の製品のバージョン情報が出力されます。
構成ソフトウェアの形名	退避，回復，または削除の対象とした実行環境の製品に含まれる構成ソフトウェアの形名が出力されます。 構成ソフトウェアの情報は，製品に含まれる構成ソフトウェアの数分出力されます。
起動時のコマンドと引数	内部処理（構成ソフトウェアの処理）を起動したコマンドと，引数が出力されます。
構成ソフトウェアの保守情報	内部処理（構成ソフトウェアの処理）実行時の保守情報が出力されます。
構成ソフトウェアの戻り値※	内部処理（構成ソフトウェアの処理）の実行結果が出力されます。
cosmienv コマンドの戻り値※	cosmienv コマンドの実行結果が出力されます。

注※　構成ソフトウェアおよび cosmienv コマンドの戻り値については，「7.2.5　構築済み実行環境の切り替えで使用するコマンド（cosmienv（構築済み実行環境の退避，回復，削除，一覧出力））」を参照してください。

退避機能を実行した場合の出力例を次に示します。この例では，退避機能実行時に，構成ソフトウェアの処理でディスク容量不足を検出しています。

```
2007/05/20 14:37:22 P-1M43-7K71 uCosminexus Application Server 07-50 START
2007/05/20 14:37:22 P-1M3Z-7151 START 2007/05/20 14:37:22 P-1M43-7K71 uCosminexus Application Server 07-50
```

```
START
2007/05/20 14:37:22 P-1M3Z-7151 START cosmippenv -s -P/home/CosmiBackup/UCAPSE001 -ZCosmiInside
2007/05/20 14:37:24 P-1M3Z-7151 END 0
2007/05/20 14:37:47 P-1M41-E171 START cosmippenv -s -P/home/CosmiBackup/UCAPSE001 -ZCosmiInside
2007/05/20 14:37:52 httpsd tar: tape write error: No space left on device
2007/05/20 14:37:53 P-1M41-E171 END 128
2007/05/20 14:37:54 P-1M43-7K71 uCosminexus Application Server 07-50 END 143
```

7.2.8　構築済み実行環境を切り替える場合の注意事項

ここでは，構築済み実行環境を切り替える場合の注意事項について説明します。

- 退避機能および回復機能を実行する場合，次に示すファイルは，退避および回復されません。
  - ユーザ任意の定義ファイル
  - ログファイル
  - プロパティの設定によって作成されるファイル

  これらのファイルのうち，退避機能実行前に退避したり，回復機能実行後に回復したりする作業が必要なファイルがあります。手動で退避・回復が必要なファイルについては，「7.2.3(1)(a) 退避作業が必要なファイル」を参照してください。
- CTM ドメインマネジャが異常終了したあとに，構築済み実行環境の切り替えを実施すると，CTM ドメインマネジャの起動時に KFCT70604-E メッセージを出力して，起動に失敗する場合があります。CTM ドメインマネジャの起動に失敗した場合は，次のどちらかを実施してください。
  - ctmdmstart コマンドに，-CTMAutoForce オプションまたは-CTMForceStart を指定して，CTM ドメインマネジャを起動してください。
  - CTMSPOOL 環境変数で指定したディレクトリ以下を削除してから，CTM ドメインマネジャを起動してください。
- HP-UX 11i V2（IPF）で構築済み実行環境を切り替える場合は，OS のパッチ PHCO_31633 がインストールされていないことを確認してください。

7.3　J2EE アプリケーションのテスト機能

J2EE アプリケーションのテスト機能には，J2EE アプリケーションの動作モードとして，本番稼働できる J2EE アプリケーションの動作モードである**通常モード**と，テスト専用の動作モードである**テストモード**の二つがあります。

この節では，テストモードについて説明します。

この節の構成を次の表に示します。

表 7-12　この節の構成（J2EE アプリケーションのテスト機能）

分類	タイトル	参照先
解説	テストモードとは	7.3.1
	テストモードで動作している J2EE アプリケーションに対する処理	7.3.2
	テストモードで動作している J2EE アプリケーションに対してできる動作	7.3.3
	テストモードで動作している J2EE アプリケーション情報の取得	7.3.4
運用	テストモードでの J2EE アプリケーションの動作確認（新規作成時）	7.3.5
	テストモードでの J2EE アプリケーションの動作確認（入れ替え時）	7.3.6
	テストモードによる J2EE アプリケーションの実行手順	7.3.7
	アプリケーションのモードの切り替えで使用するコマンド（cjchmodapp（アプリケーションのモード切り替え））	7.3.8
注意事項	テストモードでの J2EE アプリケーション呼び出し時の注意事項	7.3.9
	テストモード使用時のそのほかの注意事項	7.3.10

注　「実装」,「設定」について，この節での説明はありません。

7.3.1　テストモードとは

テストモードとは，J2EE アプリケーションをテスト用に動作させるための，専用の動作モードです。J2EE アプリケーションの新規開発時や，J2EE アプリケーションの入れ替え時など，J2EE アプリケーションを本番環境でテストするときに利用します。なお，テストモードの動作の対象となるのは，Web アプリケーションと EJB です。

テストモードには次のような特徴があります。

- **稼働中の J2EE サーバ上にテスト用の J2EE アプリケーションを配置して，テストを実施できます。**
  テストモードで J2EE アプリケーションを動作させても，稼働中の J2EE アプリケーションに影響を与えることはありません。このため，本番環境で J2EE アプリケーションのテストを実施できます。
- **J2EE アプリケーションを容易に本番環境に移行できます。**
  テストモードで動作している J2EE アプリケーションは，テスト終了後，容易に本番運用に移行できます。移行のときには，DD などの設定を変更する必要はありません。

(1)　テストモード使用の前提条件

テストモードで J2EE アプリケーションを動作させるときの前提条件を次に示します。

- 同一 J2EE サーバでは，同じ名称の J2EE アプリケーション（表示名が同じ J2EE アプリケーション）は，通常モードとテストモードでそれぞれ一つだけ開始できます。
- テストモードの J2EE アプリケーションと，通常モードの J2EE アプリケーション間で，相互呼び出しはできません。
- Connector 1.5 仕様に準拠した Inbound リソースアダプタは，テストモードで使用できません。
- CDI コンポーネントを含む J2EE アプリケーションはテストモードで使用できません。テストモードで J2EE アプリケーションを開始した場合，KDJE42396-E が出力され，開始に失敗します。

(2)　J2EE アプリケーションの動作モードの変更

テストモードから通常モードへの変更，および通常モードからテストモードへの変更時には，J2EE アプリケーションをいったん停止する必要があります。モードの変更後，J2EE アプリケーションを開始してください。なお，モードを変更するとき，標準 DD などの設定変更は不要です。

(3)　テストモードを使用するための設定

テストモードにするかどうかは，J2EE アプリケーション単位で設定します。テストモードでの J2EE アプリケーションの設定については，「7.3.5　テストモードでの J2EE アプリケーションの動作確認（新規作成時）」を参照してください。

(4)　テストモードで動作中のアプリケーションに対してできる操作

テストモードで動作する J2EE アプリケーションのカスタマイズおよび運用操作は，サーバ管理コマンドで実行します。テストモードでの J2EE アプリケーションの運用については，「7.3.3　テストモードで動作している J2EE アプリケーションに対してできる動作」を参照してください。

7.3.2　テストモードで動作している J2EE アプリケーションに対する処理

ここでは，テストモードで実行している J2EE アプリケーションに対するクライアントからの処理について説明します。

(1)　処理の流れ

テストモードおよび通常モードで実行している J2EE アプリケーションがあるときのアプリケーションサーバの動作について，CTM を使用する場合と使用しない場合に分けて，図に示します。

図 7-3　処理の流れ（CTM を使用する場合）

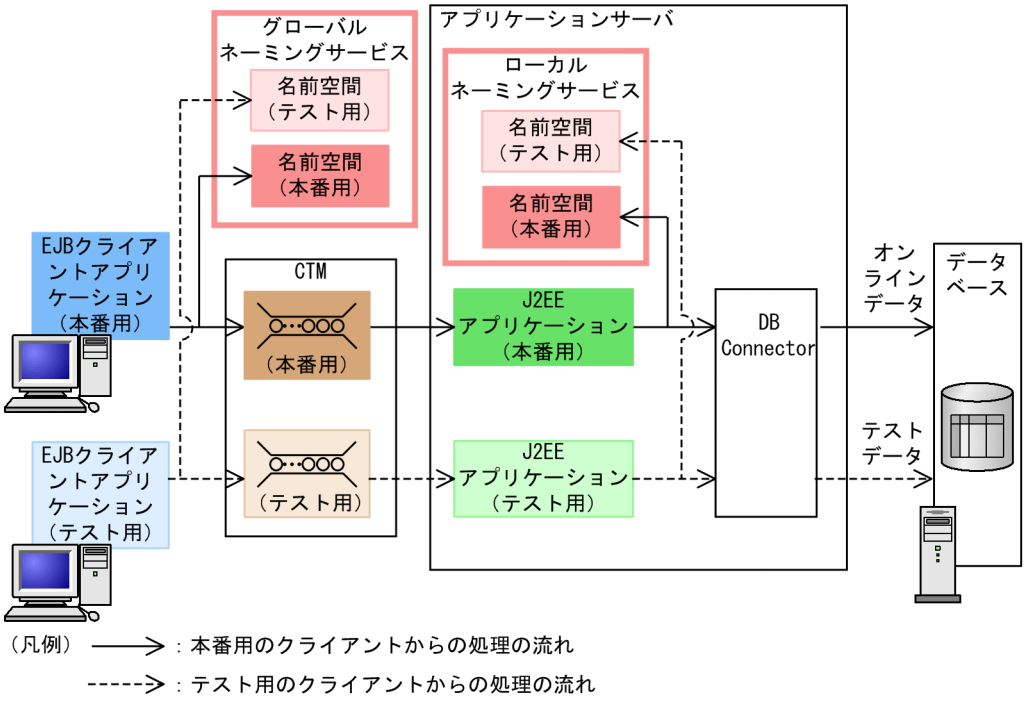
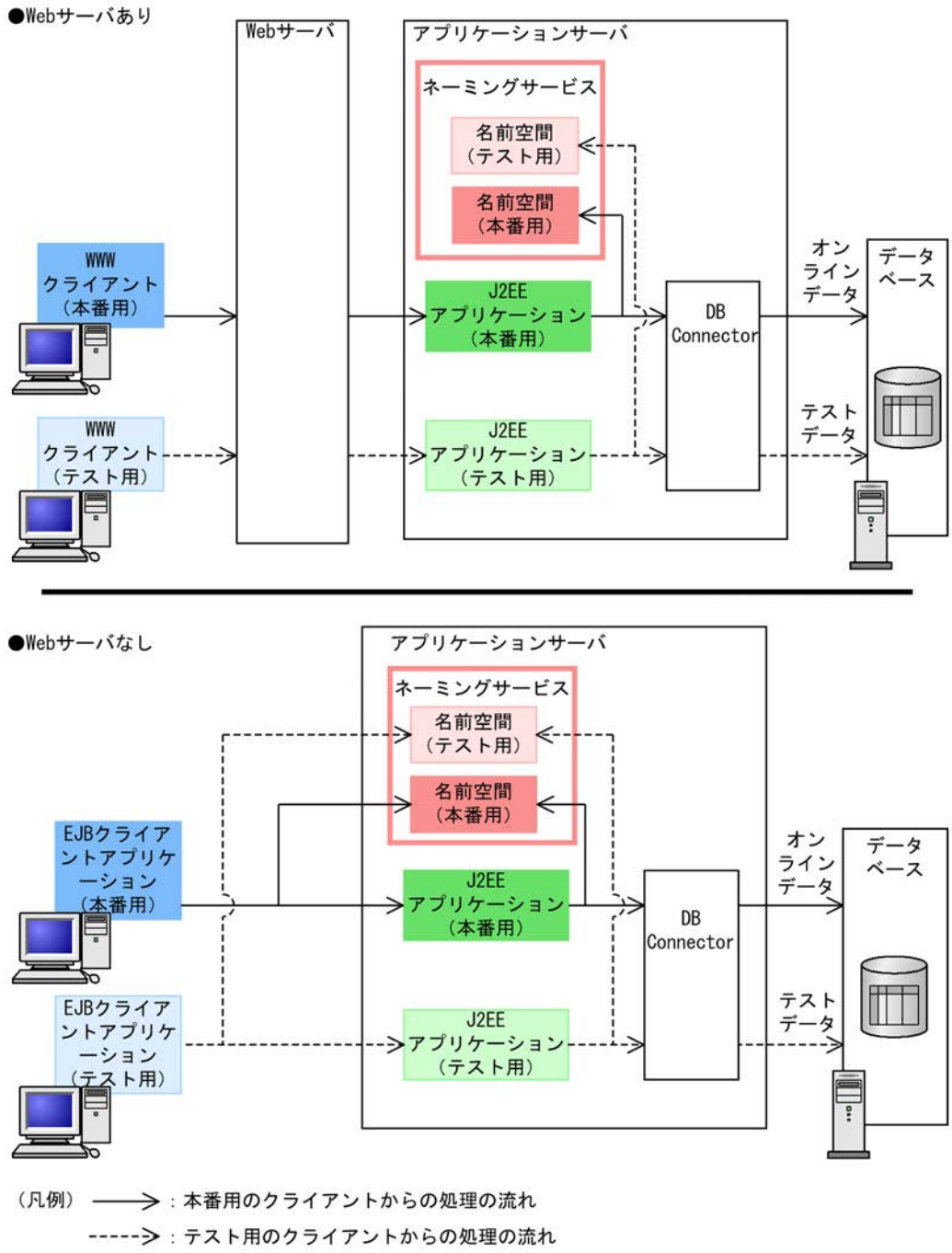


図 7-4　処理の流れ



図について説明します。

クライアント

クライアントは，本番用とテスト用とで別に用意します。

J2EE アプリケーション

通常モードで動作中の J2EE アプリケーションのほかに，テストモードで J2EE アプリケーションをデプロイします。

名前空間

CTM を使用する場合，グローバルネーミングサービス，およびローカルネーミングサービスの名前空間は，本番用とテスト用で別に用意します。CTM を使用しない場合のネーミングサービスも同様に，本番用とテスト用を別に用意します。

クライアントから検索する名前空間を本番用とテスト用とで切り分けることで，EJB の呼び出しの切り替えをします。

CTM のスケジュールキュー（CTM を使用する場合だけ）

CTM のスケジュールキューは，本番用の J2EE アプリケーションとテスト用の J2EE アプリケーションで別に生成します。

DB Connector およびデータベース

J2EE アプリケーションがデータベースを使用する場合は，本番用に定義した DB Connector を使用します。テスト用のクライアントや J2EE アプリケーションから，テスト用のデータを生成することで，データベースでの確認をします。

参考

なお，テストモードの J2EE アプリケーションに含まれる Web アプリケーションから Web サーバ上の静的コンテンツを利用する場合には，次のどちらかの方法で静的コンテンツにアクセスしてください。

1. Web アプリケーションから Web サーバ上の静的コンテンツに相対パスでアクセスする場合，あらかじめ Web サーバ上のディレクトリ構成をテストモードのコンテキストルート名を含むディレクトリ構成として静的コンテンツを配置し，アクセスするようにしてください。
2. Web アプリケーションから Web サーバ上の静的コンテンツに絶対パスでアクセスする場合，あらかじめ Web サーバ上で本番用のディレクトリ構成下に配置された静的コンテンツにアクセスするようにしてください。

(2)　テストモードで動作している Web アプリケーションの呼び出し

テストモードで動作している場合，テスト用のクライアントからは，テスト用の URL で Web アプリケーションに接続します。

J2EE アプリケーションをテストモードで開始すると，コンテキストルートに「_test_」のプリフィックスが付いた，テスト用のコンテキストルートが生成されます。このコンテキストルートを含む URL が，テスト用の URL になります。例えば，コンテキストルートが「example」で，本番用の URL が「http://www.*xxx*.co.jp/example/index.jsp」である場合，テスト用の URL は「http://www.*xxx*.co.jp/_test_/example/index.jsp」となります。

! 注意事項

通常モードで，コンテキストルート名が「/_test_」で始まる Web アプリケーションを使用している場合，コンテキストルート名が「/_test_」で始まらない Web アプリケーションをテストモードで実行すると，コンテキストルート名が重複したり，またはコンテキストルート名のパスの構成要素が包含関係となったりする場合があります。

例えば，通常モードでコンテキストルート名が「/_test_/example」の Web アプリケーション 1 を使用している場合に，コンテキストルート名が「/example」の Web アプリケーション 2 は，テストモードではコンテキストルート名に「/_test_」が付加されるためコンテキストルート名が Web アプリケーション 1 と重複することになり実行できません。

また，コンテキストルート名が「/example/app」の Web アプリケーション 3 を，テストモードで実行すると，コンテキストルート名は「/_test_/example/app」となり，Web アプリケーション 1 とはコンテキストルート名のパスの構成要素が包含関係となります。この場合，テストモードで実行することはできますが，包含されるコンテキストルート名を持つ Web アプリケーション 1 にアクセスできない場合があります。

(3)　テストモードで動作している EJB の呼び出し

EJB では，EJB ホームオブジェクトおよび EJB ローカルホームオブジェクトの取得には，JNDI を利用します。テストモードの場合，通常モード用とは別に名前空間を設けることで JNDI でモードの切り替えをして，EJB を呼び出します。

J2EE アプリケーションをテストモードで開始すると，EJB ホームオブジェクトおよび EJB ローカルホームオブジェクトなどは，JNDI で自動変換してから，テストモード用の名前空間に登録および検索されます。クライアント側では，テスト用の名前空間を検索して，EJB ホームオブジェクトや EJB ローカルホームオブジェクトなどを取得します。JNDI が lookup 呼び出し元クライアントの J2EE アプリケーションのモードを判別して，検索名の変換を自動で実施するので，クライアント側では，テスト用の検索名を意識する必要はありません。

また，テストモードを通常モードに変更しても，lookup の呼び出し元のソースや，WAR，EAR などの DD の変更は必要ありません。

！ 注意事項

EJB クライアントアプリケーションの場合は，通常モードとテストモードの区別がありません。このため，JNDI では検索名の自動変換はされませんので，クライアント側ではテストモードの検索名を意識して J2EE アプリケーションを作成する必要があります。

また，J2EE アプリケーションに Entity Bean（CMP2.0）を含む場合，J2EE アプリケーションのテスト機能を利用できません。

(4)　J2EE アプリケーション外からテストモードで実行されている Enterprise Bean のルックアップをする場合の名称の指定方法

J2EE アプリケーション外からテストモードで実行されている Enterprise Bean をルックアップする場合，テストモード用の名前空間を意識してルックアップする必要があります。通常モードで実行される Enterprise Bean を「HITACHI_EJB/」で始まる文字列でルックアップする場合，同じ Enterprise Bean がテストモードで実行されるときには「$HITACHI_TEST/HITACHI_EJB/」で始まる文字列でルックアップします。また，通常モードで実行される Enterprise Bean を「MyCart」でルックアップする場合，同じ Enterprise Bean がテストモードで実行されるときには，「$HITACHI_TEST/MyCart」でルックアップする必要があります。

(5)　EJB クライアントアプリケーションからルックアップを使用して UserTransaction を取得する場合の検索文字列

EJB クライアントアプリケーションから UserTransaction をルックアップする場合に，テストモードの J2EE アプリケーションを呼び出すときには，次の検索文字列を指定します。

```
$HITACHI_TEST/HITACHI_EJB/SERVERS/<サーバ名称>/SERVICES/UserTransaction
```

(6)　ユーザ指定名前空間機能でテストモードを使用する場合の注意

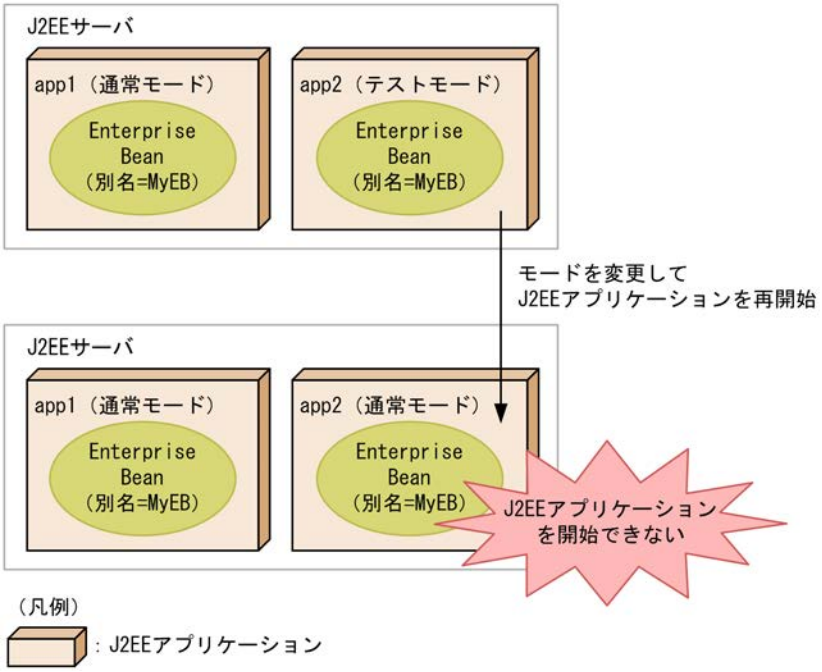
テストモードを使用している場合，1.と 2.で示すオブジェクトの別名に，同じ名前を付けることができます。

1. テストモードで動作している J2EE アプリケーションに含まれる，リモートインタフェースを持つ Enterprise Bean
2. 通常モードで動作している J2EE アプリケーションに含まれる，リモートインタフェースを持つ Enterprise Bean，または J2EE リソース

ただし，これらのオブジェクトに重複した別名を付けている場合に，どちらかの J2EE アプリケーションの動作モードを変更して開始しようとすると，エラーが発生します。

エラーが発生する例を次の図に示します。

図 7-5　J2EE アプリケーションの動作モードを変更した場合にエラーが発生する例



動作モードを変更した場合に J2EE アプリケーションを起動できなかったときは，重複しない別名に変更してください。

(7)　CTM を使用する場合の注意事項

CTM を使用する場合は，次の内容に注意してください。

(a)　スケジュールキューの名称

スケジュールキューの名称は，通常モードの場合，デフォルトでは，J2EE アプリケーションの名称になります。通常モードのキュー名称との衝突を回避するため，テストモードでは，スケジュールキューの名称を，通常モードのキュー名称の先頭に，プリフィックス「_test_」が付けられます。CTM で提供するコマンドを使用してテスト用のキューを操作する場合は，キュー名称を意識する必要があります。

なお，通常モードとテストモードの J2EE アプリケーションの間では，CTM のスケジュールキューを共有することはできません。

(b)　EJB ホームオブジェクトの JNDI 登録名

J2EE アプリケーションをテストモードで開始するとき，ローカルネーミングサービスと同様に CTM が利用するグローバルネーミングサービスにもテスト用の名前空間を形成し，EJB ホームオブジェクトを登録します。

(8)　テストモードで起動された J2EE アプリケーションに対するそのほかの注意事項

- テストモードで起動されたアプリケーション内のオブジェクトは Portable Global JNDI 名の登録はできません。

- J2EE アプリケーションの名称を変更する場合，すでに変更後の名称と同じ名称の J2EE アプリケーションがある場合，その名称には変更できません。大文字，小文字の違いによる区別はできません。ただし，通常モードまたはテストモードなどの運用時のモードが異なるときは，それぞれのモードで一つずつ同じ名称の J2EE アプリケーションを登録できます。

7.3.3　テストモードで動作している J2EE アプリケーションに対してできる動作

ここでは，**テストモード**での J2EE アプリケーションの運用について説明します。

J2EE アプリケーションを新しく登録する場合や，すでに動作中の J2EE アプリケーションを入れ替えるとき，本番運用を開始する前に，本番環境で問題なく動作することを確認したいことがあります。このような場合にテストモードを使用します。

テストモードでの運用は，J2EE アプリケーション単位で実行できます。テストモードから本番運用時のモード（**通常モード**）に変更する場合，J2EE アプリケーションの設定やカスタマイズをし直す必要はありません。また，一つの本番環境で表示名が同じ J2EE アプリケーションを，通常モードとテストモードで同時に開始できます。これによって，既存の J2EE アプリケーションでクライアントにサービスを提供しながら，入れ替える予定の J2EE アプリケーションの動作を同じ環境で確認できます。

テストモードで運用できるのは，J2EE アプリケーションだけです。J2EE アプリケーション内で DB Connector などのリソースアダプタを使用する場合は，本番時に使用するものを使用してください。また，Web サーバ上の静的コンテンツについても，本番用に構築した Web サーバ上に配置して使用してください。

J2EE アプリケーションのテストモードでの動作確認には，サーバ管理コマンドを使用します。

テストモードで動作中の J2EE アプリケーションに対してできる操作を次に示します。

- インポート
- J2EE アプリケーションの開始
- J2EE アプリケーションの停止（通常停止，および強制停止）
- J2EE アプリケーションの削除
- J2EE アプリケーションの設定の追加，変更，および削除
- モード変更
- リデプロイ
- リロード

Management Server では，J2EE アプリケーションをテストモードでインポートして動作確認することはできません。ただし，次の場合には，Management Server の運用管理コマンド（mngsvrutil）で，テストモードで動作している J2EE アプリケーションの稼働情報が参照できます。

- Management Server に登録されている J2EE アプリケーションを，サーバ管理コマンドを使用してテストモードで動作させた場合
- Management Server に登録されている J2EE アプリケーションと同名の J2EE アプリケーションを，サーバ管理コマンドを使用してテストモードで動作させた場合

運用管理コマンド（mngsvrutil）では，テストモードで動作しているJ2EEアプリケーションを操作できません。テストモードでJ2EEアプリケーションを操作したい場合は，サーバ管理コマンドを実行してください。ただし，テストモードの場合も，運用管理コマンドで稼働情報を出力できます。

なお，J2EEアプリケーションのモードが，サーバ管理コマンドでテストモードに変更された場合，運用管理コマンドにサブコマンド「get」を指定して取得したJ2EEアプリケーションの稼働情報のfullyQualifiedNameに，「_TEST」という文字列が付いて出力されます。

また，次の表に示すコマンドは，テストモードで実行しているJ2EEアプリケーションに対して実行できます。cjlistthreadコマンド以外のコマンドでは，-testオプションを指定することで，コマンドの実行対象がJ2EEアプリケーションになります。cjlistthreadコマンドは，テストモードのJ2EEアプリケーションだけを対象にするためのオプションはありません。

テストモードのJ2EEアプリケーションに対して実行できるサーバ管理コマンドと実行時の形式を次の表に示します。各コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2. J2EEサーバで使用するコマンド」を参照してください。

表7-13　テストモードのJ2EEアプリケーションに対して実行できるサーバ管理コマンドと実行時の形式

コマンド	テストモードでの動作	形式
cjclearpool	テストモードのJ2EEアプリケーションにに含まれる指定されたデータソース（JDBC）またはリソースアダプタのコネクションをクリアします。	リソースアダプタ（J2EEアプリケーションに含めて使用）の場合 cjclearpool［<サーバ名称>］［-nameserver <プロバイダURL>］-type connector［-mode {normal｜plan}］［-test］-name <アプリケーション名> -resname <リソースアダプタ表示名>［-resname <リソースアダプタ表示名> ...］
cjdeleteapp	テストモードのJ2EEアプリケーションを削除できます。	cjdeleteapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名>［-name <J2EEアプリケーション名> ...］
cjdeletelibjar	ライブラリJARをテストモードのJ2EEアプリケーションから削除します。	cjdeletelibjar［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <アプリケーション名> -f <ライブラリJAR名>［-f <ライブラリJAR名> ...］
cjgencmpsql	テストモードのJ2EEアプリケーションのCMP2.x Entity Bean用のSQL文を生成します。	cjgencmpsql［サーバ名称］［-nameserver <プロバイダURL>］［-test］-name <アプリケーション名>［-resname <EJB-JARの表示名/Entity Beanの表示名>］
cjgetappprop	テストモードのJ2EEアプリケーションのアプリケーション属性，WAR属性およびEJB-JAR属性を取得できます。	アプリケーションの場合 cjgetappprop［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名>［-encoding <エンコーディング名>］-c <属性ファイルパス> EJB-JARファイル，WARファイル，またはRARファイルの場合 cjgetappprop［<サーバ名称>］［-nameserver <プロバイダURL>］[-test］-name <J2EEアプリケーション名> -type {ejb|war|rar} -resname <リソース表示名>[/<下位リソース表示名>］［-encoding <エンコーディング名>］-c <属性ファイルパス> すべての情報を取得する場合

コマンド	テストモードでの動作	形式
cjgetappprop	テストモードの J2EE アプリケーションのアプリケーション属性，WAR 属性および EJB-JAR 属性を取得できます。	cjgetappprop [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <J2EEアプリケーション名> -type all [-encoding <エンコーディング名>] -c <属性ファイルパス>
cjgetstubsjar	テストモードの J2EE アプリケーションから RMI-IIOP スタブおよびインタフェースを取得できます。	cjgetstubsjar [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <アプリケーション名> -d <スタブおよびインタフェース格納先ディレクトリパス>
cjimportapp	J2EE アプリケーションをテストモードでインポートできます。インポートは，実行時情報を含む形式（ZIP ファイル）または実行時情報を含まない形式（EAR ファイル）のどちらも実行できます。	アーカイブ形式のアプリケーションをインポートする場合 cjimportapp [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] [-nodelete] -f <EARファイルのパス> [-f <EARファイルのパス>...] 展開ディレクトリ形式のアプリケーションとしてインポートする場合（アプリケーションディレクトリ） cjimportapp [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -a <アプリケーションディレクトリパス> 展開ディレクトリ形式のアプリケーションとしてインポートする場合（標準のアプリケーションファイル（拡張子.ear）またはアプリケーションサーバのアプリケーション実行環境情報ファイル（拡張子.zip）） cjimportapp [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] [-nodelete] -f <EARファイルのパス> -d <展開ディレクトリパス>
cjimportlibjar	テストモードの J2EE アプリケーションにライブラリ JAR をインポートします。	cjimportlibjar [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <アプリケーション名> -f <ライブラリJARファイルパス> [-f <ライブラリJARファイルパス> ...]
cjlistapp	J2EE サーバに含まれるすべての J2EE アプリケーションのうち，テストモードの J2EE アプリケーションの一覧を表示できます。また，テストモードの J2EE アプリケーションに含まれる WAR ファイルおよび EJB-JAR ファイルの一覧を表示できます。	アプリケーションに含まれる EJB-JAR ファイルまたは WAR ファイルの場合 cjlistapp [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <J2EEアプリケーション名> -type {ejb|war} [-resname <表示名>] アプリケーションに含まれる RAR ファイルの場合 cjlistapp [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <J2EEアプリケーション名> -type rar [-spec | -resname <表示名> -outbound| -resname <表示名> -inbound| -resname <表示名> -listenertype <メッセージリスナのタイプ名>]
cjlistlibjar	テストモードの J2EE アプリケーションに含まれるライブラリ JAR の一覧を標準出力に出力します。	cjlistlibjar [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <アプリケーション名>
cjlistpool	テストモードの J2EE アプリケーションに含まれるリソースアダプタのコネクションプールの状態を標準出力に出力します。	アプリケーションに含めて使用されるリソースアダプタを指定する場合 cjlistpool [<サーバ名称>] [-nameserver <プロバイダURL>] [-test] -name <アプリケーション名> -resname <リソースアダプタの表示名> [-resname <リソースアダプタの表示名> ...]

コマンド	テストモードでの動作	形式
cjlistthread	テストモードの J2EE アプリケーションのスレッド一覧を表示できます。	cjlistthread［<サーバ名称>］［-nameserver <プロバイダURL>］
cjreloadapp	テストモードの J2EE アプリケーションをリロードできます。	cjreloadapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <アプリケーション名>［-t <強制リロード開始までのタイムアウト時間>］
cjrenameapp※	テストモードの J2EE アプリケーションの名称を変更します。	cjrenameapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <アプリケーション名> -newname <変更後アプリケーション名>
cjreplaceapp	テストモードの J2EE アプリケーションをリデプロイできます。	cjreplaceapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <アプリケーション名> -f <ファイルパス>［-t <タイムアウト時間>］［-replaceDD］
cjsetappprop	テストモードの J2EE アプリケーションのアプリケーション属性，WAR 属性および EJB-JAR 属性を設定できます。	アプリケーションの場合 cjsetappprop［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名> -c <属性ファイルパス> EJB-JAR ファイル，WAR ファイル，または RAR ファイルの場合 cjsetappprop［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名> -type {ejb|war|rar} -resname <リソース表示名>[/<下位リソース表示名>］-c <属性ファイルパス> すべての情報を設定する場合 cjsetappprop［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名> -type all -c <属性ファイルパス>
cjstartapp	J2EE アプリケーションをテストモードで開始できます。	cjstartapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名>［-jspc］
cjstopapp	テストモードで開始した J2EE アプリケーションを停止できます。	通常停止を行う場合 cjstopapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名>［-t <タイムアウト時間>］ 通常停止がタイムアウトしたあと，強制停止を手動で行う場合 cjstopapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名> -cancel 通常停止がタイムアウトしたあと，強制停止を自動で行う場合 cjstopapp［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <J2EEアプリケーション名> -t <タイムアウト時間> -force
cjstopthread	テストモードの J2EE アプリケーションに対してメソッドキャンセルを実行できます。	cjstopthread［<サーバ名称>］［-nameserver <プロバイダURL>］-tid <スレッドID>［-tid <スレッドID> ...］
cjtestres	テストモードの J2EE アプリケーションに含まれる J2EE リソース	アプリケーションに含めて使用するリソースアダプタの場合

コマンド	テストモードでの動作	形式
cjtestres	アダプタ，メールおよびデータソースの接続テストをします。	cjtestres［<サーバ名称>］［-nameserver <プロバイダURL>］［-test］-name <アプリケーション名> -type rar -resname <表示名>［-resname <表示名> ...］

注※　テストモードと通常モードで，それぞれに一意のアプリケーション名称を登録できます。同一モードで同一アプリケーション名称が存在する場合は，エラーとなります。このとき，アプリケーション名の大文字・小文字は区別されません。

7.3.4　テストモードで動作している J2EE アプリケーション情報の取得

テストモードで動作している J2EE アプリケーションについて，Management Server を使用すると統計情報の取得ができます。また，性能解析トレースを使用することで，J2EE アプリケーションのモードを確認できます。

(1)　統計情報の取得

通常モードの J2EE アプリケーションと同様に，テストモードの J2EE アプリケーションでも，統計情報を取得できます。統計情報の取得には，Management Server を使用します。

! 注意事項

テストモードで動作している J2EE アプリケーションの場合，Management Server は統計情報の取得にだけ利用できます。Management Server を使用して，テストモードで動作する J2EE アプリケーションに対して操作することはできません。

(2)　性能解析トレースの利用による J2EE アプリケーションの動作モードの確認

テストモードの J2EE アプリケーションの呼び出しと，通常モードの J2EE アプリケーションの相互呼び出しはできません。性能解析トレースを使用すると，クライアントからのリクエストがテストモードの J2EE アプリケーションの呼び出しか，通常モードの J2EE アプリケーションの呼び出しかを判断できるので，テストモードと通常モードの J2EE アプリケーションを呼び出していないかどうか，確認できます。

Web アプリケーション呼び出し時と EJB 呼び出し時のモードの判別方法について説明します。なお，トレース取得ポイントの詳細と取得情報については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「8.　性能解析トレースのトレース取得ポイントと PRF トレース取得レベル」を参照してください。

(a)　Web アプリケーション呼び出し時のモードの判別

Web アプリケーション呼び出しでモードの判別をするときの，トレース取得ポイントとモードの判別方法について次に示します。

トレース取得ポイント

- Web コンテナでのリクエスト取得・リクエストヘッダ解析完了時点
- リクエスト処理完了時点

モードの判別方法

上記のトレース取得ポイントで取得された出力情報で，オペレーション名に出力される URL のコンテキストルートに「_test_」が含まれる場合は，テストモードとなります。

(b)　EJB 呼び出し時のモードの判別

EJB 呼び出しでモードの判別をするときの，トレース取得ポイントとモードの判別方法について次に示します。

トレース取得ポイント

- JNDI での検索処理開始時点

  EJB 呼び出しの切り替えは JNDI で実施しているため，JNDI での検索処理開始時の情報を取得します。

モードの判別方法

上記のトレース取得ポイントで取得された出力情報で，オペレーション名に出力される検索文字列に「$HITACHI_TEST」が含まれる場合は，テストモードとなります。

7.3.5　テストモードでの J2EE アプリケーションの動作確認（新規作成時）

ここでは，新規作成した J2EE アプリケーションをテストモードで実行して問題なく動作することを確認してから，J2EE アプリケーションの動作モードを通常モードに切り替えて，本番運用を開始する場合の操作について説明します。すでに動作中の J2EE アプリケーションを入れ替える場合には，動作中の J2EE アプリケーションの停止や退避などの操作が必要になります。J2EE アプリケーションの入れ替えと保守については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「5.6.3　J2EE アプリケーションの入れ替えと保守」を参照してください。

なお，Web アプリケーションを含む J2EE アプリケーションをテストモードで運用する場合，あらかじめ次の作業が必要です。

1. リダイレクタの設定（URL のマッピング）

   通常モードのコンテキストルートを含む URL のマッピングに加えて，テストモードのコンテキストルートを含む URL のマッピングを設定してください。Web アプリケーションを含む J2EE アプリケーションをテストモードで運用する場合，Web アプリケーションのコンテキストルートにプリフィックス「_test_」を付けたものが，コンテキストルートとして使用されます。例えば，通常モードのコンテキストルートが「/jsptobean」の場合，テストモードのコンテキストルートは「/_test_/jsptobean」になります。

   URL のマッピングについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.3　URL パターンでのリクエストの振り分け」を参照してください。

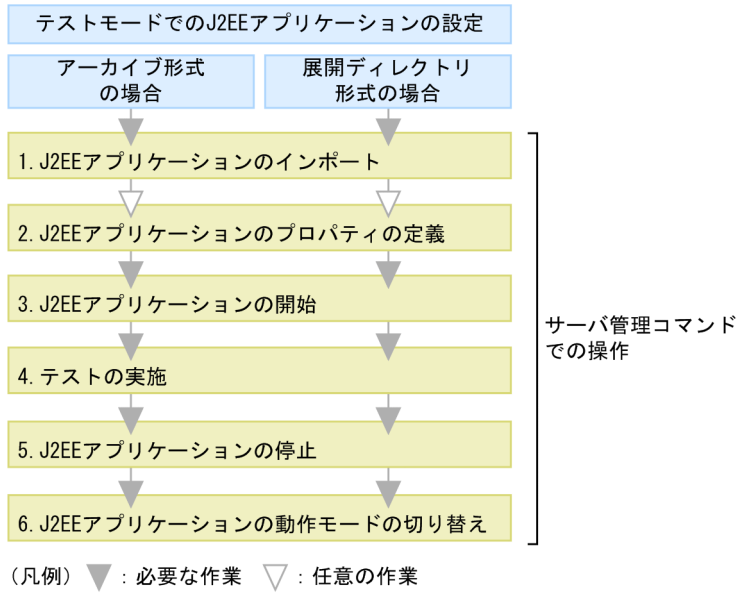
2. Web サーバの再起動

   Web サーバを再起動して，リダイレクタの設定を有効にしてください。

ここでは，上記の作業が済んでいることを前提に，テストモードでの J2EE アプリケーションの設定について説明します。

テストモードでの J2EE アプリケーションの設定の流れを次の図に示します。

図 7-6　テストモードでの J2EE アプリケーションの設定の流れ



図中の 1.～6.について説明します。

1. サーバ管理コマンドを使用して，J2EE アプリケーションをテストモードでインポートします。

   J2EE アプリケーションのインポートには，cjimportapp コマンドで-test を指定します。cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

2. 必要に応じて，サーバ管理コマンドを使用して，J2EE アプリケーションのプロパティを定義します。

   J2EE アプリケーションのプロパティの定義には，cjgetappprop コマンドおよび cjsetappprop コマンドで-test を指定します。cjgetappprop -test で各属性ファイルを取得し，ファイル編集後に，cjsetappprop -test で編集内容を反映させます。

   J2EE アプリケーションのプロパティの設定内容については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「9.1　J2EE アプリケーションのプロパティ設定の概要」を参照してください。

   cjgetappprop コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetappprop（アプリケーションの属性の取得）」を，cjsetappprop コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetappprop（アプリケーションの属性設定）」を参照してください。

   各属性ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.　J2EE アプリケーションの設定で使用する属性ファイル」を参照してください。

3. サーバ管理コマンドを使用して，J2EE アプリケーションをテストモードで開始します。

   cjstartapp コマンドで-test を指定して，J2EE アプリケーションをテストモードで運用します。cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

4. テストを実施します。

   テストモード用に設定した特定のクライアントから適宜操作を実行し，テストを実施して問題ないことが確認できたら，手順 5.に進みます。

テスト結果に問題があった場合は，J2EE アプリケーションを修正して，手順 1.からもう一度実行してください。なお，同じ名称の J2EE アプリケーションは同じモードで登録できません。このため，テストが失敗したテストモードの J2EE アプリケーションは，cjdeleteapp コマンドで-test を指定して削除してください。cjdeleteapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeleteapp（J2EE アプリケーションの削除)」を参照してください。

ポイント

- テストモードで動作している J2EE アプリケーションは，テストモード用に設定した特定のクライアントからだけ利用できます。Web アプリケーションを含む J2EE アプリケーションをテストモードで運用する場合，Web アプリケーションのコンテキストルートにプリフィックス「_test_」を付けたものが，コンテキストルートとして使用されます。例えば，コンテキストルートが「/jsptobean」で，本番運用での URL が「http://www.xxx.co.jp/jsptobean/account.jsp」の場合，テストモードでの URL は，「http://www.xxx.co.jp/_test_/jsptobean/account.jsp」になります。テスト用のクライアントからはこの URL を使用して J2EE アプリケーションに対して操作を実行してください。
- EJB を含む J2EE アプリケーションをテストモードで運用する場合，EJB はテスト用の名前空間に登録されます。EJB クライアントアプリケーションから EJB を呼び出す場合には，通常モード用の名前空間のルートコンテキスト名で始まる検索文字列「HITACHI_EJB/…」ではなく，テストモード用の名前空間のルートコンテキスト名で始まる検索文字列「$HITACHI_TEST/…」を指定して lookup を実行してください。

5. **サーバ管理コマンドを使用して，テストモードで運用している J2EE アプリケーションを停止します。**

   cjstopapp コマンドで-test を指定して，J2EE アプリケーションを停止します。cjstopapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstopapp（J2EE アプリケーションの停止)」を参照してください。

6. **サーバ管理コマンドを使用して，テストモードで運用していた J2EE アプリケーションの動作モードを通常モードに変更します。**

   cjchmodapp コマンドで-test -mode normal を指定して，J2EE アプリケーションの動作モードを通常モードに変更します。cjchmodapp コマンドについては，「7.3.8　アプリケーションのモードの切り替えで使用するコマンド（cjchmodapp（アプリケーションのモード切り替え))」を参照してください。

サーバ管理コマンドでの操作については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「3.　サーバ管理コマンドの基本操作」を参照してください。また，コマンドや属性ファイルについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」，またはマニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」を参照してください。

7.3.6　テストモードでの J2EE アプリケーションの動作確認（入れ替え時）

J2EE アプリケーションを入れ替えるときに，J2EE アプリケーションをテストモードで開始する手順の例を示します。なお，この手順では，入れ替え元の J2EE アプリケーションは，J2EE サーバ上で開始されていることを前提としています。

準備

Web アプリケーションを含む J2EE アプリケーションをテストモードで運用する場合，あらかじめ次の作業が必要です。

- リダイレクタの設定（URL のマッピング）

  通常モードのコンテキストルートを含む URL のマッピングに加えて，テストモードのコンテキストルートを含む URL のマッピングを設定してください。Web アプリケーションを含む J2EE アプリケーションをテストモードで運用する場合，Web アプリケーションのコンテキストルートにプリフィックス「_test_」を付けたものが，コンテキストルートとして使用されます。例えば，通常モー

ドのコンテキストルートが「/jsptobean」の場合，テストモードのコンテキストルートは「/_test_/jsptobean」になります。

URL パターンでのリクエスト振り分けの設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「4.3　URL パターンでのリクエストの振り分け」を参照してください。

- Web サーバの再起動

  Web サーバを再起動して，リダイレクタの設定を有効にしてください。

手順

1. J2EE アプリケーションをテストモードでインポートします。

   J2EE アプリケーションのインポートには，サーバ管理コマンド（cjimportapp）を使用します。

   実行形式と実行例を次に示します。

   実行形式

```
cjimportapp <J2EEサーバ名> -test -f <EARファイルのパス>
```

   実行例

```
cjimportapp MyServer -test -f App1.ear
```

2. インポートした J2EE アプリケーションのプロパティ設定とカスタマイズを実行します。

   サーバ管理コマンド（cjgetappprop および cjsetappprop）を使用して実行します。それぞれのコマンドには-test オプションを指定して，テストモードで実行します。

   cjgetappprop コマンドの実行形式と実行例を次に示します。

   実行形式

```
cjgetappprop <J2EEサーバ名> -test -name <J2EEアプリケーション名> -type all -c <アプリケーション統合属性ファイルの出力先パス>
```

   実行例

```
cjgetappprop MyServer -test -name App1 -type all -c App1Prop.xml
```

   属性ファイルはテキストエディタなどで編集します。

   cjsetappprop コマンドの実行形式と実行例を次に示します。

   実行形式

```
cjsetappprop <J2EEサーバ名> -test -name <J2EEアプリケーション名> -type all -c <編集したアプリケーション統合属性ファイルのパス>
```

   実行例

```
cjsetappprop MyServer -test -name App1 -type all -c App1Prop.xml
```

3. J2EE アプリケーションをテストモードで開始します。

   cjstartapp コマンドの実行形式と実行例を次に示します。

   実行形式

```
cjstartapp <J2EEサーバ名> -test -name <J2EEアプリケーション名>
```

   実行例

```
cjstartapp MyServer -test -name App1
```

4. テストを実施します。

   テスト用のクライアントから適宜操作を実行して，テストします。問題ないことが確認できたら，手順 5.に進みます。

テスト結果に問題があった場合は，J2EE アプリケーションを修正して，手順 1.からもう一度実行してください。なお，同じ名称の J2EE アプリケーションは同じモードで登録できません。このため，テストが失敗したテストモードの J2EE アプリケーションは削除してください。テストモードの J2EE アプリケーションの削除は，サーバ管理コマンド（cjdeleteapp）に-test オプションを付けた形式で実行してください。

ポイント

- テストモードで動作している J2EE アプリケーションは，テストモード用に設定した特定のクライアントからだけ利用できます。Web アプリケーションを含む J2EE アプリケーションをテストモードで運用する場合，Web アプリケーションのコンテキストルートにプリフィックス「_test_」を付けたものが，コンテキストルートとして使用されます。例えば，コンテキストルートが「/jsptobean」で，本番運用での URL が「http://www.xxx.co.jp/jsptobean/account.jsp」の場合，テストモードでの URL は，「http://www.xxx.co.jp/_test_/jsptobean/account.jsp」になります。テスト用のクライアントからはこの URL を使用して J2EE アプリケーションに対して操作を実行してください。
- EJB を含む J2EE アプリケーションをテストモードで運用する場合，EJB はテスト用の名前空間に登録されます。EJB クライアントアプリケーションから EJB を呼び出す場合には，通常モード用の名前空間のルートコンテキスト名で始まる検索文字列「HITACHI_EJB/…」ではなく，テストモード用の名前空間のルートコンテキスト名で始まる検索文字列「$HITACHI_TEST/…」を指定して lookup を実行してください。

5. テストモードで運用している J2EE アプリケーションを停止します。

実行形式

```
cjstopapp 〈J2EEサーバ名〉 -test -name 〈J2EEアプリケーション名〉
```

実行例

```
cjstopapp MyServer -test -name App1
```

6. 通常モードで運用している J2EE アプリケーションを停止します。

なお，J2EE アプリケーションを入れ替えではなく新規にインポートした場合，この操作は不要です。

実行形式

```
cjstopapp 〈J2EEサーバ名〉 -name 〈J2EEアプリケーション名〉
```

実行例

```
cjstopapp MyServer -name App1
```

7. 通常モードで運用していた J2EE アプリケーションの名称を変更します。

なお，J2EE アプリケーションを入れ替えではなく新規にインポートした場合，この操作は不要です。

また，この操作は，インポートしたファイルの世代を管理したい場合などに，必要に応じて実行してください。例えば，「App1」という J2EE アプリケーションの場合，「App1_bak」などに変更します。

実行形式

```
cjrenameapp 〈J2EEサーバ名〉 -name 〈変更前のJ2EEアプリケーション名〉 -newname 〈変更後のJ2EEアプリケーション名〉
```

実行例

```
cjrenameapp MyServer -name App1 -newname App1_bak
```

8. テストモードで運用していた J2EE アプリケーションのモードを通常モードに変更します。

実行形式

```
cjchmodapp 〈J2EEサーバ名〉 -test -name 〈テストモードで動作を確認したJ2EEアプリケーション名〉 -mode normal
```

実行例

```
cjchmodapp MyServer -test -name App1 -mode normal
```

9. 手順 8.でモードを変更した J2EE アプリケーションを開始します。

実行形式

```
cjstartapp 〈J2EEサーバ名〉 -name 〈J2EEアプリケーション名〉
```

実行例

```
cjstartapp MyServer -name App1
```

7.3.7　テストモードによる J2EE アプリケーションの実行手順

テストモードによる J2EE アプリケーションの実行は，アーカイブ形式，展開ディレクトリ形式のどちらのアプリケーションでも実行できます。

テストモードで J2EE アプリケーションを実行することで，J2EE アプリケーションを本番環境で問題なく実行できるかを確認します。

次の二つの場合について説明します。

- 新規作成した J2EE アプリケーションをテストモードで実行します。
- J2EE アプリケーションを入れ替える場合にテストモードを使用します。

(1)　新規作成した J2EE アプリケーションをテストモードで実行する

新規作成した J2EE アプリケーションをテストモードで実行します。また，本番環境で問題なく実行できることを確認してから通常モードに切り替えます。

1. J2EE アプリケーションをテストモードでインポートします。

   J2EE アプリケーションをテストモードでインポートするには，-test オプションを指定した cjimportapp コマンドを実行します。

2. J2EE アプリケーションをテストモードで開始します。

   J2EE アプリケーションをテストモードで開始するには，-test オプションを指定した cjstartapp コマンドを実行します。

3. テストを実施します。

   実際に J2EE アプリケーションを操作してテストします。問題がないことを確認したら次の手順に進みます。

   テスト結果に問題があった場合は，J2EE アプリケーションを修正して，手順 1.からもう一度実行してください。なお，同じ名称の J2EE アプリケーションは同じモードで登録できません。このため，テストが失敗したテストモードの J2EE アプリケーションは削除してください。テストモードの J2EE アプリケーションを削除するには，-test オプションを指定した cjdeleteapp コマンドを実行します。

4. テストモードで実行している J2EE アプリケーションを停止します。

   テストモードで実行中の J2EE アプリケーションを停止するには，-test オプションを指定した cjstopapp コマンドを実行します。

5. 次に示すコマンドを実行して，J2EE アプリケーションのモードを，テストモードから通常モードへ切り替えます。

実行形式

```
cjchmodapp ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -test -name <アプリケーション名> -mode normal
```

実行例

```
cjchmodapp MyServer -test -name App1 -mode normal
```

cjchmodapp コマンドの詳細については，「7.3.8　アプリケーションのモードの切り替えで使用するコマンド（cjchmodapp（アプリケーションのモード切り替え））」を参照してください。

6. テストモードから通常モードへ切り替えた J2EE アプリケーションを開始します。

(2)　J2EE アプリケーションを入れ替える場合にテストモードを使用する

J2EE アプリケーションを入れ替える前に，入れ替える J2EE アプリケーションをテストモードで実行します。また，本番環境で問題なく実行できることを確認してから，既存の J2EE アプリケーションと入れ替えます。

! 注意事項

J2EE サーバ内で同じアプリケーションディレクトリを持つ複数の J2EE アプリケーションは共存できません。展開ディレクトリ形式のアプリケーションの場合はテストモードのアプリケーションと通常モードのアプリケーションでアプリケーションディレクトリを分けてください。

1. J2EE アプリケーションをテストモードでインポートします。

J2EE アプリケーションをテストモードでインポートするには，-test オプションを指定した cjimportapp コマンドを実行します。

2. J2EE アプリケーションをテストモードで開始します。

J2EE アプリケーションをテストモードで開始するには，-test オプションを指定した cjstartapp コマンドを実行します。

3. テストを実施します。

実際にアプリケーションを操作してテストします。問題がないことを確認したら次の手順に進みます。

テスト結果に問題があった場合は，J2EE アプリケーションを修正して，手順 1.からもう一度実行してください。なお，同じ名称の J2EE アプリケーションは同じモードで登録できません。このため，テストが失敗したテストモードの J2EE アプリケーションは削除してください。テストモードの J2EE アプリケーションの削除は，-test オプションを指定した cjdeleteapp コマンドを実行します。

4. テストモードで実行している J2EE アプリケーションを停止します。

テストモードで実行中の J2EE アプリケーションを停止するには，-test オプションを指定した cjstopapp コマンドを実行します。

5. 通常モードで実行している J2EE アプリケーションを停止します。

6. 通常モードで実行していた J2EE アプリケーションの名称を変更します。

この操作は，インポートしたファイルの世代を管理したい場合などに，必要に応じて実行してください。例えば，「App1」という J2EE アプリケーションの場合，「App1_bak」などに変更します。

マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「10.6　J2EE アプリケーション名の変更」を参照してください。

7. 次に示すコマンドを実行して，テストモードで実行していた J2EE アプリケーションのモードを，テストモードから通常モードへ切り替えます。

実行形式

```
cjchmodapp ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -test -name <アプリケーション名> -mode normal
```

実行例

```
cjchmodapp MyServer -test -name App1 -mode normal
```

8. テストモードから通常モードへ切り替えた J2EE アプリケーションを開始します。

7.3.8　アプリケーションのモードの切り替えで使用するコマンド（cjchmodapp（アプリケーションのモード切り替え））

アプリケーションのモード切り替えには，次のコマンドを使用します。

表 7-14　アプリケーションのモード切り替えで使用するコマンド

コマンド名称	分類	概要	コマンドの系統
cjchmodapp	アプリケーションのモード切り替え	J2EE アプリケーションの動作モードを変更します。	更新

コマンドの格納先

次のディレクトリに格納されています。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥bin¥
- UNIX の場合
  /opt/Cosminexus/CC/admin/bin/

以降にコマンドの文法を示します。

● 形式

アプリケーションをテストモードから通常モードに変更する場合

```
cjchmodapp [<サーバ名称>] [-nameserver <プロバイダURL>] -test
          -name <アプリケーション名> -mode normal
```

アプリケーションを通常モードからテストモードに変更する場合

```
cjchmodapp [<サーバ名称>] [-nameserver <プロバイダURL>]
          -name <アプリケーション名> -mode test
```

● 機能

J2EE アプリケーションの動作モードを，テストモードから通常モードへ，または通常モードからテストモードへ変更します。

● 引数

<サーバ名称>

接続先 J2EE サーバ名称を指定します。サーバ名称を省略したときは，ホスト名称がサーバ名称として使用されます。

-nameserver <プロバイダ URL>

CORBA ネーミングサービスへのアクセスプロトコル，CORBA ネーミングサービスが稼働しているホスト名，およびそれが使用しているポート番号を次に示す形式で指定します。

<プロトコル名称>::<ホスト名称>:<ポート番号>

指定内容の詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「2.1(2)　プロバイダ URL について」を参照してください。

-test

テストモードのアプリケーションに対してコマンドを実行する場合に指定します。

-name <アプリケーション名>

モードを変更する J2EE アプリケーション名を指定します。

-mode {test|normal}

アプリケーションの動作モードを指定します。

- test
  テストモードに変更します。
- normal
  通常モードに変更します。このオプションを指定する場合，-test オプションの指定が必要です。

● 入力例

アプリケーションをテストモードから通常モードに変更する場合

```
cjchmodapp MyServer -test -name App1 -mode normal
```

アプリケーションを通常モードからテストモードに変更する場合

```
cjchmodapp MyServer -name App1 -mode test
```

● 戻り値

0：

正常終了しました。

1：

異常終了しました。

2：

排他エラーによってコマンドを実行できません。

3：

タイムアウトが発生しました。

9：

管理者特権がないため，コマンドが実行できません（Windows Server 2012，Windows Server 2008，Windows 8，Windows 7 または Windows Vista の場合）。

● 注意事項

- コマンドの引数にサーバ名称を指定する場合，コマンド名の直後にする必要があります。そのほかの引数は，サーバ名称より後ろ（サーバ名称を省略した場合はコマンド名の後ろ）にあれば順序は任意です。ただし，オプション名と値の順序の入れ替え（例：<アプリケーション名> -name）やオプション名と値を非対応にすること（例：-nameserver <アプリケーション名> -name <プロバイダ URL>）はできません。
- コマンドの引数にサーバ名称を指定する場合，cjsetup コマンドで指定したサーバ名称と大文字・小文字を一致させた文字列を指定する必要があります。
- WAR アプリケーションに対してモードの変更はできません。-name オプションに WAR アプリケーションを指定した場合，このコマンドは異常終了します。

7.3.9　テストモードでの J2EE アプリケーション呼び出し時の注意事項

アプリケーションの動作モードには，本番稼働できる**通常モード**と，テスト専用の**テストモード**の二つがあります。

テストモードは，アプリケーションの新規開発や，アプリケーションの入れ替えなど，アプリケーションをテストするときに利用します。なお，テストの対象となるのは，Web アプリケーションと Enterprise Bean です。

(1)　テストモードでの Web アプリケーション呼び出しについての注意事項

アプリケーションの作りによっては，テストモードでの Web アプリケーション呼び出し機能が利用できないことがあります。注意事項を次に説明します。

(a)　ほかの Web アプリケーションとのリンクを作成する場合

テストモードで起動する J2EE アプリケーションに複数の Web アプリケーションが含まれている場合，その Web アプリケーションから，同じくテストモードで起動された Web アプリケーションへのリンク（URL または URI）を作成する処理が含まれることがあります。この場合，サーブレット API でほかの Web アプリケーションのコンテキストルートは取得できないため，Web アプリケーションで固定またはユーザプログラム内の定義で指定していると考えられます。

この場合，テストモードの Web アプリケーションにアクセスし，作成されたページのリンクにアクセスすると，テストモードの Web アプリケーションではなく，通常モードの Web アプリケーションにアクセスするので注意してください。

(b)　静的コンテンツ内のリンクがコンテキストルートからのパスで記述されている場合

静的コンテンツ内で記述しているリンクが，相対パスではなくコンテキストルートからのパスで記述してある場合，通常モードの Web アプリケーションにアクセスしてしまいます。

(c)　サーブレット API を利用する場合

サーブレット API で使用するメソッドによっては，次に示す注意が必要になる場合があります。

注意事項 A：

引数にコンテキストルート名をハードコーディングしたり，戻り値として特定のコンテキストルート名が返されることを期待したりしないでください。

注意事項 B：

戻り値としてファイルシステム上の実際のパスが返されます。特定のファイルパスが返されることを期待しないでください。

注意事項 A および注意事項 B に該当するメソッドを，次の表に示します。

表 7-15　注意が必要なサーブレット API

パッケージ	クラス	メソッド	注意事項	
			A	B
javax.servlet	ServletContext	getContext(java.lang.String uripath)	○	−
		getResource(java.lang.String path)	○	−
		getRealPath(java.lang.String path)	−	○

パッケージ	クラス	メソッド	注意事項	
			A	B
javax.servlet.http	HttpServletRequest	getContextPath()	○	−
		getPathTranslated()	○	−
		getRequestURI()	○	−
		getRequestURL()	○	−
		getPathTranslated()	−	○
	HttpServletResponse	encodeRedirectUrl(java.lang.String url)	○	−
		encodeRedirectURL(java.lang.String url)	○	−
		encodeUrl(java.lang.String url)	○	−
		encodeURL(java.lang.String url)	○	−
		sendRedirect(java.lang.String location)	○	−
	Cookie	getPath()	○	−
		setPath(java.lang.String uri)	○	−
	HttpServletRequestWrapper	getContextPath()	○	−
		getPathTranslated()	○	−
		getRequestURI()	○	−
		getRequestURL()	○	−
		getPathTranslated()	−	○
	HttpServletResponseWrapper	encodeRedirectUrl(java.lang.String url)	○	−
		encodeRedirectURL(java.lang.String url)	○	−
		encodeUrl(java.lang.String url)	○	−
		encodeURL(java.lang.String url)	○	−
		sendRedirect(java.lang.String location)	○	−
	HttpUtils	getRequestURL(HttpServletRequest req)	○	−

（凡例）

○：注意事項に該当することを示します。

−：注意事項に該当することを示します。

(2)　テストモードでの Enterprise Bean 呼び出しについての注意事項

Enterprise Bean の場合，一般に EJB ホームオブジェクトおよび EJB ローカルホームオブジェクトの取得に JNDI を利用します。したがって，通常モード用とは別にテストモード用の名前空間を設け，JNDI で通常／テストのモード切り替えを実施することで，テストモードでの Enterprise Bean の呼び出しを実現します。テストモードでの Enterprise Bean の呼び出しについての注意事項を次に示します。

(a)　Enterprise Bean を開始する場合

Enterprise Bean を含む J2EE アプリケーションをテストモードで開始するとき，テストモードで開始された Enterprise Bean の EJB ホームオブジェクト，および EJB ローカルホームオブジェクトなどをテストモード用の名前空間に登録します。テストモード用名前空間への登録／検索については，JNDI で自動変換して実行するため，J2EE アプリケーションに本番／テストのモード変更が発生しても，lookup メソッド呼び出し元のソースや，WAR，Enterprise Bean などの DD 定義の変更は不要です。

(b)　クライアントから接続する場合

クライアントでは，テストモード用の名前空間を検索して EJB ホームオブジェクト，および EJB ローカルホームオブジェクトなどを取得します。

J2EE アプリケーション形態の Enterprise Bean のクライアント（例：JSP／サーブレットや Enterprise Bean）では，JNDI が lookup メソッド呼び出し元（クライアント）の J2EE アプリケーションのモードを判別して自動的に検索名を変換するため，クライアント側ではテストモードの検索名を意識しなくて済みます。

一方，J2EE アプリケーション形態以外の Enterprise Bean のクライアント（例：EJB クライアントアプリケーション）では，JNDI では自動的に検索名を変換しません。そのため，クライアント側では，テストモードの検索名を意識し，テスト用のクライアントを作成してください（J2EE アプリケーション形態以外の Enterprise Bean のクライアントには，通常モード／テストモードの区別がありません。このため，JNDI では，検索名の自動変換は実行されません）。

(c)　ダイナミッククラスローディング機能を使用する場合

ダイナミッククラスローディング機能を使用する場合，一つの J2EE サーバ上で通常モードとテストモードの J2EE アプリケーションを，同じ名称で同時に開始できません。例えば，すでに通常モードで J2EE アプリケーションが動作しているときにテストモードの J2EE アプリケーションを開始しようとすると，開始に失敗します。

テストモードで J2EE アプリケーションを動作させる場合は，ダイナミッククラスローディング機能を使用しないことをお勧めします。

7.3.10　テストモード使用時のそのほかの注意事項

- mngsvrutil コマンドのサブコマンド start を指定して J2EE アプリケーションをテストモードで開始することはできません。
- mngsvrutil コマンドのサブコマンド start を指定してテストモードで開始されている J2EE アプリケーションを停止することはできません。
- ユーザスレッドではテストモードでの J2EE アプリケーションの実行機能を使用できません。

7.4　JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック（cjjsp2java）

cjjsp2java コマンドを使用して，JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックができます。

この節の構成を次の表に示します。

表 7-16　この節の構成（JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェック）

分類	タイトル	参照先
実装	JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックで使用するコマンド（cjjsp2java（JSP ソースのチェック））	7.4.1
注意事項	JSP 実装時の注意事項	7.4.2

注　「解説」「設定」および「運用」について，この節での説明はありません。

7.4.1　JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックで使用するコマンド（cjjsp2java（JSP ソースのチェック））

JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックでは，次のコマンドを使用します。

表 7-17　JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックで使用するコマンド

コマンド名称	分類	概要
cjjsp2java	JSP ソースのチェック	JSP ファイルのシンタックスチェックをします。

コマンドの格納先

JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ソースのチェックで使用するコマンドは，次のディレクトリに格納されています。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥web¥bin¥
- UNIX の場合
  /opt/Cosminexus/CC/web/bin/

以降にコマンドの文法を示します。

● 形式

```
cjjsp2java [-d <ディレクトリ>|-d:non] [-classpath <クラスパス>]
           [-root <ディレクトリ> [[-taglib <uri> <location>]...]]
           <JSPファイル名>...
```

● 機能

JSP1.1 仕様および JSP1.2 仕様に準拠した JSP ファイルから Java ソースを生成し，JSP ファイルのシンタックスチェックをします。

なお，JSP2.0 仕様以降に準拠した JSP ファイルのシンタックスチェックをする場合は，cjjspc コマンドを使用してください。

● 引数

-d <ディレクトリ>

生成した java ソースファイルの出力先ディレクトリを指定します。このオプションを指定しなかった場合は，コマンドを実行したカレントディレクトリ下に生成されます。

-root オプションを指定していない場合，-d オプションで指定したディレクトリの直下に java ソースファイルが生成されます。同一ファイル名の JSP ファイルが複数指定された場合, java ソースファイルも同一名称となるため，先に生成された java ソースファイルは，あとから生成された java ソースファイルに上書きされます。-root オプションを指定している場合，-d オプションで指定したディレクトリ下に-root オプションで指定した web アプリケーションのルートディレクトリから JSP ファイルまでのパスが生成され，そのディレクトリ下に java ソースファイルが生成されます。

-d:non

java ソースファイルを生成しません。JSP ファイルの確認だけを実施します。

-classpath <クラスパス>

JSP ファイルから java ファイルを生成する時に必要となるクラスのクラスパスを絶対パスで指定します。次のクラスパスが必要です。

- タグライブラリの実装クラス（タグライブラリバリデータの実装クラスも含む）
- <jsp:useBean>タグで指定した JavaBeans の実装クラス

複数のパスを指定する場合は，";" で区切って指定します（UNIX の場合，":"）。

なお，-root オプションを使用し，指定したディレクトリ下の WEB-INF¥classes¥（UNIX の場合，WEB-INF/classes/）ディレクトリ，および WEB-INF¥lib¥（UNIX の場合，WEB-INF/lib/）ディレクトリ下に必要な class ファイル，jar ファイルを配置する場合，このオプションで指定する必要はありません。

-root <ディレクトリ>

Web アプリケーションのルートディレクトリを指定します。JSP ファイルで include ディレクティブを使用する場合，タグライブラリを使用する場合，または java ソースファイル生成時に必要なクラスパスを指定する場合に，このオプションを指定します。-root オプションを指定した場合，指定した Web アプリケーションのルートディレクトリ下に JSP ファイルを配置する必要があります。Web アプリケーションのルートディレクトリ下以外にある JSP ファイルを指定した場合は無視されます。

include ディレクティブを使用する場合

include ディレクティブの page 属性に"/"から始まるパスを指定した場合，このオプションで指定されたルートディレクトリからインクルード対象の JSP ファイル，または静的コンテンツが検索されます。また，page 属性に"/"から始まるパスを指定していても，-root オプションを指定していない場合は，コマンドに指定した JSP ファイルのディレクトリから"/"以下のファイルが検索されます。

タグライブラリを使用する場合

-taglib オプションの<location>に指定されたパス，または-taglib オプションを使用しないで JSP ファイル内に直接記述された tld ファイルの検索に使用されます。タグライブラリを使用する場合，-root オプションは必ず指定してください。

tld ファイルが jar ファイルに含まれる場合，その jar ファイルの検索に使用されます。この場合，使用する jar ファイルは指定した Web アプリケーションのルートディレクトリ下の WEB-INF¥lib¥（UNIX の場合，WEB-INF/lib/）ディレクトリ下に配置する必要があります。

クラスパスを指定する場合

java ファイル生成時にクラスパスの指定が必要な場合，Web アプリケーションのルートディレクトリ下の WEB-INF¥classes¥（UNIX の場合，WEB-INF/classes/）ディレクトリ，または WEB-

INF¥lib¥（UNIX の場合，WEB-INF/lib/）ディレクトリ下に class ファイル，jar ファイルを配置します。クラスパスの指定が必要な場合については，-classpath オプションの説明を参照してください。

-taglib <uri> <location>

タグライブラリを使用する場合に，<uri>および<location>に，それぞれ web.xml に定義する<taglib-uri>タグ，<taglib-location>タグの内容を指定します。-taglib オプションを指定する場合，-root オプションで Web アプリケーションのルートディレクトリを指定する必要があります。

複数のタグライブラリを使用する場合，-taglib <uri> <location>の形式で複数指定します。

<JSP ファイル名>

java ソースファイルを生成する JSP ファイルを指定します。複数の JSP ファイルを指定する場合は，空白で区切って指定します。

複数の JSP ファイルを指定した場合，同名の JSP ファイルがあるときは，先に生成された java ソースファイルは，あとから生成された java ソースファイルに上書きされます。

複数の JSP ファイルを指定した場合，指定した順に java ソースが生成されます。途中でエラーを検出した場合は java ソースファイルの生成が中断されます。

● 戻り値

0：

正常終了しました。

1：

オプション解析エラーです。

2 以上：

異常終了しました。

● 注意事項

- このコマンドで生成された java ソースファイルは，javac コマンドに-encoding オプションを指定してコンパイルすることで<%! ... %>タグ，<% ... %>タグ，または<%= ... %>タグで記述した java 言語ソースコードのコンパイル確認もできます。ただし，生成された java ソースファイルから，javac コマンドで作成した class ファイルは実行できません。
- また，cjjsp2java コマンドで生成したソースファイルをコンパイルするには，次に示す JAR ファイルをクラスパスに指定する必要があります。
  - <Application Server のインストールディレクトリ>¥CC¥lib¥ejbserver.jar
  - <Application Server のインストールディレクトリ>¥CC¥lib¥hitj2ee.jar

7.4.2　JSP 実装時の注意事項

taglib ディレクティブの uri 属性で指定する URI と TLD ファイルのマッピングで，TLD ファイルの<uri>要素で指定した URI とその TLD ファイル自身をマッピングする場合に，URI の重複が検出されたときには，メッセージが Web アプリケーション単位で出力され，該当するマッピングは無視されます。

ここでは，出力されるメッセージに対する cjjsp2java コマンドの場合だけに固有の出力条件について説明します。

出力されるメッセージ

KDJE39314-W

出力条件

TLD ファイルで記述した<uri>要素の内容が，コマンド引数-taglib に指定した uri の内容，またはほかの TLD ファイルで記述した<uri>要素の内容と重複したときに出力されます。

これ以外の出力条件については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「6.2.6　JSP 実装時の注意事項」を参照してください。

付録

付録 A　各バージョンでの主な機能変更

ここでは，09-70 よりも前のアプリケーションサーバの各バージョンでの主な機能の変更について，変更目的ごとに説明します。09-70 での主な機能変更については，「1.4　アプリケーションサーバ 09-70 での主な機能変更」を参照してください。

説明内容は次のとおりです。

- アプリケーションサーバの各バージョンで変更になった主な機能と，その概要を説明しています。機能の詳細については「参照先マニュアル」の「参照個所」の記述を確認してください。「参照先マニュアル」および「参照個所」には，その機能についての 09-70 のマニュアルでの主な記載個所を記載しています。

「参照先マニュアル」に示したマニュアル名の「アプリケーションサーバ」は省略しています。

付録 A.1　09-60 での主な機能変更

(1)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 A-1　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
G1GC への対応	G1GC を選択できるようになりました。	システム設計ガイド	7.15
		リファレンス 定義編(サーバ定義)	16.5
圧縮オブジェクトポインタ機能への対応	圧縮オブジェクトポインタ機能を使用できるようになりました。	機能解説 保守／移行編	9.16

(2)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 A-2　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ファイナライズ滞留解消機能の追加	ファイナライズ処理の滞留を解消でき，OS 資源の解放遅れなどの発生を抑止できるようになりました。	機能解説 保守／移行編	9.17

(3)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 A–3　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
ログファイルの非同期出力機能の追加	ログのファイル出力を非同期でできるようになりました。	リファレンス 定義編(サーバ定義)	16.2

付録 A.2　09-50 での主な機能変更

(1)　開発生産性の向上

開発生産性の向上を目的として変更した項目を次の表に示します。

表 A–4　開発生産性の向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Eclipse セットアップの簡略化	GUI を利用して Eclipse 環境をセットアップできるようになりました。	アプリケーション開発ガイド	1.1.5，2.4
ユーザ拡張性能解析トレースを使ったデバッグ支援	ユーザ拡張性能解析トレース設定ファイルを開発環境で作成できるようになりました。	アプリケーション開発ガイド	1.1.3，6.5

(2)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 A–5　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境でのシステム構成パターンの拡充	仮想化環境で使用できるティアの種類（http-tier，j2ee-tier および ctm-tier）が増えました。これによって，次のシステム構成パターンが構築できるようになりました。 • Web サーバと J2EE サーバを別のホストに配置するパターン • フロントエンド（サーブレット，JSP）とバックエンド（EJB）を分けて配置するパターン • CTM を使用するパターン	仮想化システム構築・運用ガイド	1.1.2

(3)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 A-6　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JDBC 4.0 仕様への対応	DB Connector で JDBC 4.0 仕様の HiRDB Type4 JDBC Driver，および SQL Server の JDBC ドライバに対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	3.6.3
Portable Global JNDI 名での命名規則の緩和	Portable Global JNDI 名に使用できる文字を追加しました。	機能解説 基本・開発編 (コンテナ共通機能)	2.4.3
Servlet 3.0 仕様への対応	Servlet 3.0 の HTTP Cookie の名称，および URL のパスパラメタ名の変更が，Servlet 2.5 以前のバージョンでも使用できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.7
Bean Validation と連携できるアプリケーションの適用拡大	CDI や UAP でも Bean Validation を使って検証できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	10 章
JavaMail への対応	JavaMail 1.4 に準拠した API を使用したメール送受信機能を利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	8 章
javacore コマンドが使用できる OS の適用拡大	javacore コマンドを使って，Windows のスレッドダンプを取得できるようになりました。	リファレンス コマンド編	javacore (スレッドダンプの取得／Windows の場合)

(4)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 A-7　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
コードキャッシュ領域の枯渇回避	システムで使用しているコードキャッシュ領域のサイズを確認して，領域が枯渇する前にしきい値を変更して領域枯渇するのを回避できるようになりました。	システム設計ガイド	7.2.6
		機能解説 保守／移行編	5.7.2, 5.7.3
		リファレンス 定義編 (サーバ定義)	16.1, 16.2, 16.4
明示管理ヒープ機能の効率的な適用への対応	自動解放処理時間を短縮し，明示管理ヒープ機能を効率的に適用するための機能として，Explicit ヒープに移動するオブジェクトを制御できる機能を追加しました。 • Explicit メモリブロックへのオブジェクト移動制御機能 • 明示管理ヒープ機能適用除外クラス指定機能 • Explicit ヒープ情報へのオブジェクト解放率情報の出力	システム設計ガイド	7.14.6
		機能解説 拡張編	8.2.2, 8.6.5, 8.10, 8.13.1, 8.13.3
		機能解説 保守／移行編	5.5

項目	変更の概要	参照先マニュアル	参照個所
クラス別統計情報の出力範囲拡大	クラス別統計情報を含んだ拡張スレッドダンプに，static フィールドを基点とした参照関係を出力できるようになりました。	機能解説 保守／移行編	9.6

(5)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 A-8　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
EADs セッションフェイルオーバ機能のサポート	EADs と連携してセッションフェイルオーバ機能を実現する EADs セッションフェイルオーバ機能をサポートしました。	機能解説 拡張編	5 章, 7 章
WAR による運用	WAR ファイルだけで構成された WAR アプリケーションを J2EE サーバにデプロイできるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.2.1
		機能解説 基本・開発編 (コンテナ共通機能)	13.9
		リファレンス コマンド編	cjimportwar (WAR アプリケーションのインポート)
運用管理機能の同期実行による起動と停止	運用管理機能（Management Server および運用管理エージェント）の起動および停止を，同期実行するオプションを追加しました。	機能解説 運用／監視／連携編	2.6.1, 2.6.2, 2.6.3, 2.6.4
		リファレンス コマンド編	adminagentctl(運用管理エージェントの起動と停止), mngautorun（自動起動および自動再起動の設定／設定解除), mngsvrctl (Management Server の起動／

項目	変更の概要	参照先マニュアル	参照個所
運用管理機能の同期実行による起動と停止	運用管理機能（Management Server および運用管理エージェント）の起動および停止を，同期実行するオプションを追加しました。	リファレンス コマンド編	停止／セットアップ）
明示管理ヒープ機能での Explicit メモリブロックの強制解放	javagc コマンドで，Explicit メモリブロックの解放処理を任意のタイミングで実行できるようになりました。	機能解説 拡張編	8.6.1，8.9
		リファレンス コマンド編	javagc（GC の強制発生）

(6)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 A-9　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
定義情報の取得	snapshotlog（snapshot ログの収集）コマンドで定義ファイルだけを収集できるようになりました。	機能解説 保守／移行編	2.3
		リファレンス コマンド編	snapshotlog（snapshot ログの収集）
cjenvsetup コマンドのログ出力	Component Container 管理者のセットアップ（cjenvsetup コマンド）の実行情報がメッセージログに出力されるようになりました。	システム構築・運用ガイド	4.1.4
		機能解説 保守／移行編	4.20
		リファレンス コマンド編	cjenvsetup（Component Container 管理者のセットアップ）
BIG-IP v11 のサポート	使用できる負荷分散機の種類に BIG-IP v11 が追加になりました。	システム構築・運用ガイド	4.7.2
		仮想化システム構築・運用ガイド	2.1
明示管理ヒープ機能のイベントログへの CPU 時間の出力	Explicit メモリブロック解放処理に掛かった CPU 時間が，明示管理ヒープ機能のイベントログに出力されるようになりました。	機能解説 保守／移行編	5.11.3

項目	変更の概要	参照先マニュアル	参照個所
ユーザ拡張性能解析トレースの機能拡張	ユーザ拡張性能解析トレースで，次の機能を追加しました。 • トレース対象の指定方法を通常のメソッド単位の指定方法に加えて，パッケージ単位またはクラス単位で指定できるようになりました。 • 使用できるイベント ID の範囲を拡張しました。 • ユーザ拡張性能解析トレース設定ファイルに指定できる行数の制限を緩和しました。 • ユーザ拡張性能解析トレース設定ファイルでトレース取得レベルを指定できるようになりました。	機能解説 保守／移行編	7.5.2, 7.5.3, 8.28.1
Session Bean の非同期呼び出し使用時の情報解析向上	PRF トレースのルートアプリケーション情報を使用して，呼び出し元と呼び出し先のリクエストを突き合わせることができるようになりました。	機能解説 基本・開発編（EJB コンテナ）	2.17.3

付録 A.3　09-00 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 A-10　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での構築・運用の操作対象単位の変更	仮想化環境の構築・運用時の操作対象単位が仮想サーバから仮想サーバグループへ変更になりました。仮想サーバグループの情報を定義したファイルを使用して，複数の仮想サーバを管理ユニットへ一括で登録できるようになりました。	仮想化システム構築・運用ガイド	1.1.2
セットアップウィザードによる構築環境の制限解除	セットアップウィザードを使用して構築できる環境の制限が解除されました。ほかの機能で構築した環境があってもアンセットアップされて，セットアップウィザードで構築できるようになりました。	システム構築・運用ガイド	2.2.7
構築環境の削除手順の簡略化	Management Server を使用して構築したシステム環境を削除する機能（mngunsetup コマンド）の追加によって，削除手順を簡略化しました。	システム構築・運用ガイド	4.1.37
		運用管理ポータル操作ガイド	3.6，5.4
		リファレンス コマンド編	mngunsetup（Management Server の構築環境の削除）

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 A-11　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 3.0 への対応	Servlet 3.0 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	6 章
EJB 3.1 への対応	EJB 3.1 に対応しました。	機能解説 基本・開発編 (EJB コンテナ)	2 章
JSF 2.1 への対応	JSF 2.1 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	3 章
JSTL 1.2 への対応	JSTL 1.2 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	3 章
CDI 1.0 への対応	CDI 1.0 に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	9 章
Portable Global JNDI 名の利用	Portable Global JNDI 名を利用したオブジェクトのルックアップができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	2.4
JAX-WS 2.2 への対応	JAX-WS 2.2 に対応しました。	Web サービス開発ガイド	1.1, 16.1.5, 16.1.7, 16.2.1, 16.2.6, 16.2.10, 16.2.12, 16.2.13, 16.2.14, 16.2.16, 16.2.17, 16.2.18, 16.2.20, 16.2.22, 19.1, 19.2.3, 37.2, 37.6.1, 37.6.2, 37.6.3
JAX-RS 1.1 への対応	JAX-RS 1.1 に対応しました。	Web サービス開発ガイド	1.1, 1.2.2, 1.3.2, 1.4.2, 1.5.1, 1.6, 2.3, 11 章, 12 章, 13 章, 17 章, 24 章, 39 章

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 A-12　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
SSL/TLS 通信での TLSv1.2 の使用	RSA BSAFE SSL-J を使用して，TLSv1.2 を含むセキュリティ・プロトコルで SSL/TLS 通信ができるようになりました。	機能解説 セキュリティ管理機能編	7.3

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 A-13　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web コンテナ全体の実行待ちキューの総和の監視	Web コンテナ全体の実行待ちキューの総和を稼働情報に出力して監視できるようになりました。	機能解説 運用／監視／連携編	3 章
アプリケーションの性能解析トレース（ユーザ拡張トレース）の出力	ユーザが開発したアプリケーションの処理性能を解析するための性能解析トレースを，アプリケーションの変更をしないで出力できるようになりました。	機能解説 保守／移行編	7 章
仮想化環境でのユーザスクリプトを使用した運用	任意のタイミングでユーザ作成のスクリプト（ユーザスクリプト）を仮想サーバ上で実行できるようになりました。	仮想化システム構築・運用ガイド	7.8
運用管理ポータル改善	運用管理ポータルの次の画面で，手順を示すメッセージを画面に表示するように変更しました。 • ［設定情報の配布］画面 • Web サーバ，J2EE サーバおよび SFO サーバの起動画面 • Web サーバクラスタと J2EE サーバクラスタの一括起動，一括再起動および起動画面	運用管理ポータル操作ガイド	10.11.1, 11.9.2, 11.10.2, 11.11.2, 11.11.4, 11.11.6, 11.12.2, 11.13.2, 11.13.4, 11.13.6
運用管理機能の再起動機能の追加	運用管理機能（Management Server および運用管理エージェント）で自動再起動が設定できるようになり，運用管理機能で障害が発生した場合でも運用が継続できるようになりました。また，自動起動の設定方法も変更になりました。	機能解説 運用／監視／連携編	2.4.1, 2.4.2, 2.6.3, 2.6.4
		リファレンス コマンド編	mngautorun（自動起動および自動再起動の設定／設定解除）

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 A-14　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
ログ出力時のファイル切り替え単位の変更	ログ出力時に，日付ごとに出力先のファイルを切り替えられるようになりました。	機能解説 保守／移行編	3.2.1
Web サーバの名称の変更	アプリケーションサーバに含まれる Web サーバの名称を HTTP Server に変更しました。	HTTP Server	－
BIG-IP の API（SOAP アーキテクチャ）を使用した直接接続への対応	BIG-IP（負荷分散機）で API（SOAP アーキテクチャ）を使用した直接接続に対応しました。 また，API を使用した直接接続を使用する場合に，負荷分散機の接続環境を設定する方法が変更になりました。	システム構築・運用ガイド	4.7.3，付録 K
		仮想化システム構築・運用ガイド	2.1，付録 C
		機能解説 セキュリティ管理機能編	8.2，8.4，8.5，8.6，18.2，18.3，18.4

（凡例）－：マニュアル全体を参照する

付録 A.4　08-70 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更した項目を次の表に示します。

表 A-15　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータル改善	運用管理ポータルの画面で，リソースアダプタの属性を定義するプロパティ（Connector 属性ファイルの設定内容）の設定，および接続テストができるようになりました。また，運用管理ポータルの画面で，J2EE アプリケーション（ear ファイルおよび zip ファイル）を Management Server にアップロードできるようになりました。	ファーストステップガイド	3.5
		運用管理ポータル操作ガイド	－
page/tag ディレクティブの import 属性暗黙インポート機能の追加	page/tag ディレクティブの import 属性暗黙インポート機能を使用できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.3.7
仮想化環境での JP1 製品に対する環境設定の自動化対応	仮想サーバへのアプリケーションサーバ構築時に，仮想サーバに対する JP1 製品の環境設定を，フックスクリプトで自動的に設定できるようになりました。	仮想化システム構築・運用ガイド	7.7.2
統合ユーザ管理機能の改善	ユーザ情報リポジトリでデータベースを使用する場合に，データベース製品の JDBC ドライバを使用して，データベースに接続できるようになりました。 DABroker Library の JDBC ドライバによるデータベース接続はサポート外になりました。 簡易構築定義ファイルおよび運用管理ポータルの画面で，統合ユーザ管理機能に関する設定ができるようになりました。	機能解説 セキュリティ管理機能編	5 章，14.3
		運用管理ポータル操作ガイド	3.5，10.9.1

項目	変更の概要	参照先マニュアル	参照個所
統合ユーザ管理機能の改善	また，Active Directory の場合，DN で日本語などの 2 バイト文字に対応しました。	運用管理ポータル操作ガイド	3.5, 10.9.1
HTTP Server 設定項目の拡充	簡易構築定義ファイルおよび運用管理ポータルの画面で，HTTP Server の動作環境を定義するディレクティブ（httpsd.conf の設定内容）を直接設定できるようになりました。	システム構築・運用ガイド	4.1.21
		運用管理ポータル操作ガイド	10.10.1
		リファレンス 定義編(サーバ定義)	4.13

（凡例）－：マニュアル全体を参照する

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更した項目を次の表に示します。

表 A–16　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ejb-jar.xml の指定項目の追加	ejb-jar.xml に，クラスレベルインターセプタおよびメソッドレベルインターセプタを指定できるようになりました。	機能解説 基本・開発編(EJB コンテナ)	2.15
パラレルコピーガーベージコレクションへの対応	パラレルコピーガーベージコレクションを選択できるようになりました。	リファレンス 定義編(サーバ定義)	16.5
Connector 1.5 仕様に準拠した Inbound リソースアダプタのグローバルトランザクションへの対応	Connector 1.5 仕様に準拠したリソースアダプタで Transacted Delivery を使用できるようにしました。これによって，Message-driven Bean を呼び出す EIS がグローバルトランザクションに参加できるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	3.16.3
TP1 インバウンドアダプタの MHP への対応	TP1 インバウンドアダプタを使用してアプリケーションサーバを呼び出す OpenTP1 のクライアントとして，MHP を使用できるようになりました。	機能解説 基本・開発編(コンテナ共通機能)	4 章
cjrarupdate コマンドの FTP インバウンドアダプタへの対応	cjrarupdate コマンドでバージョンアップできるリソースアダプタに FTP インバウンドアダプタを追加しました。	リファレンス コマンド編	2.2

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更した項目を次の表に示します。

表 A–17　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
データベースセッションフェイルオーバ機能の改善	性能を重視するシステムで，グローバルセッション情報を格納したデータベースのロックを取得しないモードを選択できるようになりました。また，データベースを更新しない，参照専用のリクエストを定義できるようになりました。	機能解説 拡張編	6 章

項目	変更の概要	参照先マニュアル	参照個所
OutOfMemory ハンドリング機能の対象となる処理の拡大	OutOfMemory ハンドリング機能の対象となる処理を追加しました。	機能解説 保守／移行編	2.5.7
		リファレンス 定義編(サーバ定義)	16.2
HTTP セッションで利用する Explicit ヒープの省メモリ化機能の追加	HTTP セッションで利用する Explicit ヒープのメモリ使用量を抑止する機能を追加しました。	機能解説 拡張編	8.11

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更した項目を次の表に示します。

表 A-18　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
仮想化環境での JP1 製品を使用したユーザ認証への対応（クラウド運用対応）	JP1 連携時に，JP1 製品の認証サーバを利用して，仮想サーバマネージャを使用するユーザを管理・認証できるようになりました。	仮想化システム構築・運用ガイド	1.2.2，3 章，4 章，5 章，6 章，7.9

(5)　そのほかの目的

そのほかの目的で変更した項目を次の表に示します。

表 A-19　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
負荷分散機への API（REST アーキテクチャ）を使用した直接接続の対応	負荷分散機への接続方法として，API（REST アーキテクチャ）を使用した直接接続に対応しました。 また，使用できる負荷分散機の種類に ACOS（AX2500）が追加になりました。	システム構築・運用ガイド	4.7.2，4.7.3
		仮想化システム構築・運用ガイド	2.1
		リファレンス 定義編(サーバ定義)	4.5
snapshot ログ収集時のタイムアウトへの対応と収集対象の改善	snapshot ログの収集が指定した時間で終了（タイムアウト）できるようになりました。一次送付資料として収集される内容が変更になりました。	機能解説 保守／移行編	付録 A

付録 A.5　08-53 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更された項目を次の表に示します。

表 A–20　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
さまざまなハイパーバイザに対応した仮想化環境の構築	さまざまなハイパーバイザを使用して実現する仮想サーバ上に，アプリケーションサーバを構築できるようになりました。 また，複数のハイパーバイザが混在する環境にも対応しました。	仮想化システム構築・運用ガイド	2 章，3 章，5 章

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更された項目を次の表に示します。

表 A–21　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
トランザクション連携に対応した OpenTP1 からの呼び出し	OpenTP1 からアプリケーションサーバ上で動作する Message-driven Bean を呼び出すときに，トランザクション連携ができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	4 章
JavaMail	POP3 に準拠したメールサーバと連携して，JavaMail 1.3 に準拠した API を使用したメール受信機能を利用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	8 章

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更された項目を次の表に示します。

表 A–22　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JavaVM のトラブルシュート機能強化	JavaVM のトラブルシュート機能として，次の機能が使用できるようになりました。 • OutOfMemoryError 発生時の動作を変更できるようになりました。 • JIT コンパイル時に，C ヒープ確保量の上限値を設定できるようになりました。 • スレッド数の上限値を設定できるようになりました。 • 拡張 verbosegc 情報の出力項目を拡張しました。	機能解説 保守／移行編	4 章，5 章，9 章

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更された項目を次の表に示します。

表 A–23　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
JP1/ITRM への対応	IT リソースを一元管理する製品である JP1/ITRM に対応しました。	仮想化システム構築・運用ガイド	1.3，2.1

(5)　そのほかの目的

そのほかの目的で変更された項目を次の表に示します。

表 A-24　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
Microsoft IIS 7.0 および Microsoft IIS 7.5 への対応	Web サーバとして Microsoft IIS 7.0 および Microsoft IIS 7.5 に対応しました。	－	－
HiRDB Version 9 および SQL Server 2008 への対応	データベースとして次の製品に対応しました。 • HiRDB Server Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 また，SQL Server 2008 に対応する JDBC ドライバとして，SQL Server JDBC Driver に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	3 章

（凡例）－：該当なし。

付録 A.6　08-50 での主な機能変更

(1)　導入・構築の容易性強化

導入・構築の容易性強化を目的として変更された項目を次の表に示します。

表 A-25　導入・構築の容易性強化を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Web サービスプロバイダ側での web.xml の指定必須タグの変更	Web サービスプロバイダ側での web.xml で，listener タグ，servlet タグ，および servlet-mapping タグの指定を必須から任意に変更しました。	リファレンス 定義編 (サーバ定義)	2.4
論理サーバのネットワークリソース使用	J2EE アプリケーションからほかのホスト上にあるネットワークリソースやネットワークドライブにアクセスするための機能を追加しました。	機能解説 運用／監視／連携編	1.2.3，5.2，5.7
サンプルプログラムの実行手順の簡略化	一部のサンプルプログラムを EAR 形式で提供することによって，サンプルプログラムの実行手順を簡略化しました。	ファーストステップガイド	3.5
		システム構築・運用ガイド	付録 M
運用管理ポータルの画面の動作の改善	画面の更新間隔のデフォルトを「更新しない」から「3 秒」に変更しました。	運用管理ポータル操作ガイド	7.4.1
セットアップウィザードの完了画面の改善	セットアップウィザード完了時の画面に，セットアップで使用した簡易構築定義ファイルおよび Connector 属性ファイルが表示されるようになりました。	システム構築・運用ガイド	2.2.6
仮想化環境の構築	ハイパーバイザを使用して実現する仮想サーバ上に，アプリケーションサーバを構築する手順を追加しました。※	仮想化システム構築・運用ガイド	3 章，5 章

注※

08-50 モードで構築する場合は，マニュアル「アプリケーションサーバ 仮想化システム構築・運用ガイド」の「付録 D　08-50 モードの仮想サーバマネージャを利用する場合の設定」を参照してください。

(2)　標準機能・既存機能への対応

標準機能・既存機能への対応を目的として変更された項目を次の表に示します。

表 A-26　標準機能・既存機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
OpenTP1 からの呼び出しへの対応	OpenTP1 からアプリケーションサーバ上で動作する Message-driven Bean を呼び出せるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	4 章
JMS への対応	JMS 1.1 仕様に対応した CJMS プロバイダ機能を使用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	7 章
Java SE 6 への対応	Java SE 6 の機能が使用できるようになりました。	機能解説 保守／移行編	5.5, 5.8.1
ジェネリクスの使用への対応	EJB でジェネリクスを使用できるようになりました。	機能解説 基本・開発編 (EJB コンテナ)	4.2.19

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更された項目を次の表に示します。

表 A-27　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
明示管理ヒープ機能の使用性向上	自動配置設定ファイルを使用して，明示管理ヒープ機能を容易に使用できるようになりました。	システム設計ガイド	7.2, 7.7.3, 7.11.5, 7.12.1
		機能解説 拡張編	8 章
データベースセッションフェイルオーバ機能の URI 単位での抑止	データベースセッションフェイルオーバ機能を使用する場合に，機能の対象外にするリクエストを URI 単位で指定できるようになりました。	機能解説 拡張編	5.6.1
仮想化環境での障害監視	仮想化システムで，仮想サーバを監視し，障害の発生を検知できるようになりました。	仮想化システム構築・運用ガイド	付録 D

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更された項目を次の表に示します。

表 A-28　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
管理ユーザアカウントの省略	運用管理ポータル，Management Server のコマンド，または Smart Composer 機能のコマンドで，ユーザのログイン ID お	システム構築・運用ガイド	4.1.15

項目	変更の概要	参照先マニュアル	参照個所
管理ユーザアカウントの省略	よびパスワードの入力を省略できるようになりました。	運用管理ポータル操作ガイド	2.2，7.1.1，7.1.2，7.1.3，8.1，8.2.1，付録 F.2
		リファレンス コマンド編	1.4，mngsvrctl (Management Server の起動／停止／セットアップ)，mngsvrutil (Management Server の運用管理コマンド)，8.3，cmx_admin_passwd (Management Server の管理ユーザアカウントの設定)
仮想化環境の運用	仮想化システムで，複数の仮想サーバを対象にした一括起動・一括停止，スケールイン・スケールアウトなどを実行する手順を追加しました。※	仮想化システム構築・運用ガイド	4 章，6 章

注※

08-50 モードで運用する場合は，マニュアル「アプリケーションサーバ 仮想化システム構築・運用ガイド」の「付録 D　08-50 モードの仮想サーバマネージャを利用する場合の設定」を参照してください。

(5)　そのほかの目的

そのほかの目的で変更された項目を次の表に示します。

表 A-29　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
Tenured 領域内不要オブジェクト統計機能	Tenured 領域内で不要となったオブジェクトだけを特定できるようになりました。	機能解説 保守／移行編	9.8
Tenured 増加要因の基点オブジェクトリスト出力機能	Tenured 領域内不要オブジェクト統計機能を使って特定した，不要オブジェクトの基点となるオブジェクトの情報を出力できるようになりました。		9.9
クラス別統計情報解析機能	クラス別統計情報を CSV 形式で出力できるようになりました。		9.10
論理サーバの自動再起動回数オーバー検知によるクラスタ系切り替え	Management Server を系切り替えの監視対象としているクラスタ構成の場合，論理サーバが異常停止状態(自動再起動回数をオーバーした状態，または自動再起動回数の設定が 0 のときに障害を検知した状態)になったタイミングでの系切り替えができるようになりました。	機能解説 運用／監視／連携編	18.4.3，18.5.3，20.2.2，20.3.3，20.3.4
ホスト単位管理モデルを対象とした系切り替えシステム	クラスタソフトウェアと連携したシステム運用で，ホスト単位管理モデルを対象にした系切り替えができるようになりました。		20 章

項目	変更の概要	参照先マニュアル	参照個所
ACOS（AX2000，BS320）のサポート	使用できる負荷分散機の種類に ACOS（AX2000，BS320）が追加になりました。	システム構築・運用ガイド	4.7.2，4.7.3，4.7.5，4.7.6，付録 K，付録 K.2
		リファレンス 定義編(サーバ定義)	4.5，4.6.2，4.6.4，4.6.5，4.6.6，4.10.1
CMT でトランザクション管理をする場合に Stateful Session Bean（SessionSynchronization）に指定できるトランザクション属性の追加	CMT でトランザクション管理をする場合に，Stateful Session Bean（SessionSynchronization）にトランザクション属性として Supports, NotSupported および Never を指定できるようになりました。	機能解説 基本・開発編(EJB コンテナ)	2.7.3
OutOfMemoryError 発生時の運用管理エージェントの強制終了	JavaVM で OutOfMemoryError が発生したときに，運用管理エージェントが強制終了するようになりました。	機能解説 保守／移行編	2.5.8
スレッドの非同期並行処理	TimerManager および WorkManager を使用して，非同期タイマ処理および非同期スレッド処理を実現できるようになりました。	機能解説 拡張編	10 章

付録 A.7　08-00 での主な機能変更

(1)　開発生産性の向上

開発生産性の向上を目的として変更された項目を次の表に示します。

表 A-30　開発生産性の向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
ほかのアプリケーションサーバ製品からの移行容易化	ほかのアプリケーションサーバ製品からの移行を円滑に実施するため，次の機能を使用できるようになりました。 • HTTP セッションの上限が例外で判定できるようになりました。 • JavaBeans の ID が重複している場合や，カスタムタグの属性名と TLD の定義で大文字・小文字が異なる場合に，トランスレーションエラーが発生することを抑止できるようになりました。	機能解説 基本・開発編(Web コンテナ)	2.3，2.7.5
cosminexus.xml の提供	アプリケーションサーバ独自の属性を cosminexus.xml に記載することによって，J2EE アプリケーションを J2EE サーバにインポート後，	機能解説 基本・開発編(コンテナ共通機能)	11.3

項目	変更の概要	参照先マニュアル	参照個所
cosminexus.xml の提供	プロパティの設定をしないで開始できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	11.3

(2)　標準機能への対応

標準機能への対応を目的として変更された項目を次の表に示します。

表 A-31　標準機能への対応を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
Servlet 2.5 への対応	Servlet 2.5 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	2.2, 2.5.4, 2.6, 6 章
JSP 2.1 への対応	JSP 2.1 に対応しました。	機能解説 基本・開発編 (Web コンテナ)	2.3.1, 2.3.3, 2.5, 2.6, 6 章
JSP デバッグ	MyEclipse を使用した開発環境で JSP デバッグができるようになりました。※	機能解説 基本・開発編 (Web コンテナ)	2.4
タグライブラリのライブラリ JAR への格納と TLD のマッピング	タグライブラリをライブラリ JAR に格納した場合に，Web アプリケーション開始時に Web コンテナによってライブラリ JAR 内の TLD ファイルを検索し，自動的にマッピングできるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.3.4
application.xml の省略	J2EE アプリケーションで application.xml が省略できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	11.4
アノテーションと DD の併用	アノテーションと DD を併用できるようになり，アノテーションで指定した内容を DD で更新できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	12.5
アノテーションの Java EE 5 標準準拠（デフォルトインターセプタ）	デフォルトインターセプタをライブラリ JAR に格納できるようになりました。また，デフォルトインターセプタから DI できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	11.4
@Resource の参照解決	@Resource でリソースの参照解決ができるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	12.4
JPA への対応	JPA 仕様に対応しました。	機能解説 基本・開発編 (コンテナ共通機能)	5 章, 6 章

注※　09-00 以降では，WTP を使用した開発環境で JSP デバッグ機能を使用できます。

(3)　信頼性の維持・向上

信頼性の維持・向上を目的として変更された項目を次の表に示します。

表 A-32　信頼性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
セッション情報の永続化	HTTP セッションのセッション情報をデータベースに保存して引き継げるようになりました。	機能解説 拡張編	5 章, 6 章

項目	変更の概要	参照先マニュアル	参照個所
FullGC の抑止	FullGC の要因となるオブジェクトを Java ヒープ外に配置することで，フ FullGC 発生を抑止できるようになりました。	機能解説 拡張編	8 章
クライアント性能モニタ	クライアント処理に掛かった時間を調査・分析できるようになりました。	－	－

（凡例）－：09-00 で削除された機能です。

(4)　運用性の維持・向上

運用性の維持・向上を目的として変更された項目を次の表に示します。

表 A-33　運用性の維持・向上を目的とした変更

項目	変更の概要	参照先マニュアル	参照個所
運用管理ポータルでのアプリケーション操作性向上	アプリケーションおよびリソースの操作について，サーバ管理コマンドと運用管理ポータルの相互運用ができるようになりました。	運用管理ポータル操作ガイド	1.1.3

(5)　そのほかの目的

そのほかの目的で変更された項目を次の表に示します。

表 A-34　そのほかの目的による変更

項目	変更の概要	参照先マニュアル	参照個所
無効な HTTP Cookie の削除	無効な HTTP Cookie を削除できるようになりました。	機能解説 基本・開発編 (Web コンテナ)	2.7.4
ネーミングサービスの障害検知	ネーミングサービスの障害が発生した場合に，EJB クライアントが，より早くエラーを検知できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	2.9
コネクション障害検知タイムアウト	コネクション障害検知タイムアウトのタイムアウト時間を指定できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3.15.1
Oracle11g への対応	データベースとして Oracle11g が使用できるようになりました。	機能解説 基本・開発編 (コンテナ共通機能)	3 章
バッチ処理のスケジューリング	バッチアプリケーションの実行を CTM によってスケジューリングできるようになりました。	機能解説 拡張編	4 章
バッチ処理のログ	バッチ実行コマンドのログファイルのサイズ，面数，ログの排他処理失敗時のリトライ回数とリトライ間隔を指定できるようになりました。	リファレンス 定義編 (サーバ定義)	3.6
snapshot ログ	snapshot ログの収集内容が変更されました。	機能解説 保守／移行編	付録 A.1，付録 A.2
メソッドキャンセルの保護区公開	メソッドキャンセルの対象外となる保護区リストの内容を公開しました。	機能解説 運用／監視／連携編	付録 C

項目	変更の概要	参照先マニュアル	参照個所
統計前のガーベージコレクション選択機能	クラス別統計情報を出力する前に，ガーベージコレクションを実行するかどうかを選択できるようになりました。	機能解説 保守／移行編	9.7
Survivor 領域の年齢分布情報出力機能	Survivor 領域の Java オブジェクトの年齢分布情報を JavaVM ログファイルに出力できるようになりました。	機能解説 保守／移行編	9.11
ファイナライズ滞留解消機能	JavaVM のファイナライズ処理の状態を監視して，処理の滞留を解消できるようになりました。	－	－
サーバ管理コマンドの最大ヒープサイズの変更	サーバ管理コマンドが使用する最大ヒープサイズが変更されました。	リファレンス 定義編(サーバ定義)	5.2，5.3
推奨しない表示名を指定された場合の対応	J2EE アプリケーションで推奨しない表示名を指定された場合にメッセージが出力されるようになりました。	メッセージ(構築／運用／開発用)	KDJE42374-W

（凡例）－：09-00 で削除された機能です。

付録 B　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

A

add.jvm.arg〔Web コンテナサーバ用オプション〕 230
add.library.path 148
add.library.path〔Web コンテナサーバ用オプション〕 231
afterCompletion メソッドからの Enterprise Bean の呼び出しについて 84

B

beforeCompletion メソッドからの Enterprise Bean の呼び出しについて 84
Blob インタフェース 464
built-in フィルタ 119, 172

C

CallableStatement クラス 416
CallableStatement クラスでのバッチ更新 344
cjchmodapp 686
cjcldellog 525
cjclosecn 80
cjdumpweb 210
cjjsp2java 691
cjstartweb 211
cjstopweb 212
cjwebchangedbid 214
cjwebclearpool 215
cjwebclosepool 217
cjwebdeletepool 218
cjwebeditpool 154, 219
cjweblistpool 221
cjwebsetup 143, 224
com.cosminexus.mngsvr.compat.monitor_view〔Management Server 環境設定ファイルのキー〕 519
Component Container 管理者 175
Connection クラス 402
Connector 属性ファイルのテンプレートファイル 507
cosmienv 660
cpp.library.version〔Web コンテナサーバ用オプション〕 230

D

DAB_HIRDB_DBINF_ESC 475
DAB_LANG 471
DABCPP_EVTTRC_LEVEL 495
DABCPP_EVTTRC_SIZE 495
DABEXSQL_TRC_LINE 475
DABHIRDBA_BLOBBUFSIZE 493
DABLANGMODE 493
DABORACLE_TYPE 474
DABORADBA_LONGBUFSIZE 493
DABroker Library 運用ディレクトリ 275
DABroker Library で開発するアプリケーション 341
DABroker Library で取得できるログの種類と出力先 304
DABroker Library で使用する API の一覧 400
DABroker Library で使用するコマンドの一覧 465
DABroker Library 動作環境定義ファイル 469
DABroker Library のイベントトレースの出力項目 325
DABroker Library の管理ユーザ 287
DABroker Library のスプール情報の内容 305
DABroker Library のセットアップ 287, 466
DABroker Library のディレクトリ構成 286
DABroker Library のディレクトリ構成〔windows の場合〕 275
DABroker Library の動作環境の設定（UNIX の場合） 288
DABroker Library の動作環境の設定（Windows の場合） 276
DABroker Library のトレースファイル運用時の留意事項 337
DABroker Library のメモリ所要量 267
［DABroker 動作環境設定］ダイアログ 276
DABSELECTCOLCNT 473
DABSELECTINFCNT 473
dabsetup 466
DABSQL_HiRDB_DICTIONARY_DATATYPE 474
DABSVLOGSIZE 472
DABTMP 472
DABXA_LOG_ERROR 476
dasysconf 468
DatabaseMetaData クラス 424
DataSource オブジェクトの生成 378
DataSource クラス 427

DBConnector_DABJ_CP.rar または DBConnector_DABJ_CP_Cosminexus_RM.rar を使用する場合に指定できるプロパティ　498
DBConnector_DABJ_XA.rar または DBConnector_DABJ_XA_Cosminexus_RM.rar を使用する場合に指定できるプロパティ　502
Driver クラス　401

E

ejbserver.application.InitTermProcessClasses〔Web コンテナサーバのカスタマイズ用キー〕238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.appname〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.count〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.encoding〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.filter〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.formatter〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.level〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.limit〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.msgid〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.path〔Web コンテナサーバのカスタマイズ用キー〕　238
ejbserver.application.userlog.CJLogHandler.<ハンドラ名称>.separator〔Web コンテナサーバのカスタマイズ用キー〕　239
ejbserver.application.userlog.Logger.<ロガー名称>.filter〔Web コンテナサーバのカスタマイズ用キー〕　239
ejbserver.application.userlog.Logger.<ロガー名称>.handlers〔Web コンテナサーバのカスタマイズ用キー〕　239
ejbserver.application.userlog.Logger.<ロガー名称>.level〔Web コンテナサーバのカスタマイズ用キー〕　239
ejbserver.application.userlog.Logger.<ロガー名称>.useParentHandlers〔Web コンテナサーバのカスタマイズ用キー〕　239
ejbserver.application.userlog.loggers〔Web コンテナサーバのカスタマイズ用キー〕　239
ejbserver.client.ctm.RequestPriority〔Web コンテナサーバのカスタマイズ用キー〕　241
ejbserver.client.log.directorynum〔Java アプリケーション〕　524
ejbserver.container.rebindpolicy〔Web コンテナサーバのカスタマイズ用キー〕　241
ejbserver.deploy.applications.metadata_complete〔J2EE サーバのカスタマイズ用キー〕　535
ejbserver.jdbc.propertyInfo.BUF_SIZE〔J2EE サーバのカスタマイズ用キー〕　60
ejbserver.jdbc.propertyInfo.ENCODELANG〔J2EE サーバのカスタマイズ用キー〕　60
ejbserver.jndi.cache　145
ejbserver.jndi.cache.interval　145
ejbserver.jndi.cache.interval.clear.option　145
ejbserver.jndi.cache.interval.clear.option〔Web コンテナサーバのカスタマイズ用キー〕　241
ejbserver.jndi.cache.interval〔Web コンテナサーバのカスタマイズ用キー〕　241
ejbserver.jndi.cache.reference〔Web コンテナサーバのカスタマイズ用キー〕　242
ejbserver.jndi.cache〔Web コンテナサーバのカスタマイズ用キー〕　241
ejbserver.jndi.namingservice.group.<Specify group name>.providerurls〔Web コンテナサーバのカスタマイズ用キー〕　242
ejbserver.jndi.namingservice.group.list〔Web コンテナサーバのカスタマイズ用キー〕　242
ejbserver.jndi.request.timeout　145
ejbserver.jndi.request.timeout〔Web コンテナサーバのカスタマイズ用キー〕　242
ejbserver.jta.pooldeffile.scramble.enabled〔Web コンテナサーバのカスタマイズ用キー〕　242
ejbserver.jta.TransactionManager.defaultTimeOut〔J2EE サーバのカスタマイズ用キー〕　60
ejbserver.jta.TransactionManager.defaultTimeOut〔Java アプリケーション〕　62
ejbserver.jta.TransactionManager.enlist.compatibleMode.enabled〔J2EE サーバのカスタマイズ用キー〕　60

ejbserver.logger.channels.define.<チャネル名>.filenum　156
ejbserver.logger.channels.define.<チャネル名>.filenum〔Web コンテナサーバのカスタマイズ用キー〕　243
ejbserver.logger.channels.define.<チャネル名>.filesize　156
ejbserver.logger.channels.define.<チャネル名>.filesize〔Web コンテナサーバのカスタマイズ用キー〕　243
ejbserver.logger.enabled.*　156
ejbserver.logger.enabled.*〔Web コンテナサーバのカスタマイズ用キー〕　243
ejbserver.logger.rotationStyle〔Web コンテナサーバのカスタマイズ用キー〕　243
ejbserver.logger.rotationTime〔Web コンテナサーバのカスタマイズ用キー〕　243
ejbserver.rmi.request.timeout　145
ejbserver.rmi.request.timeout〔Web コンテナサーバのカスタマイズ用キー〕　244
ejbserver.server.j2ee.feature　53
ejbserver.server.j2ee.feature〔J2EE サーバのカスタマイズ用キー〕　61
ejbserver.server.prf.PRFID〔Web コンテナサーバのカスタマイズ用キー〕　244
ejbserver.stdoutlog.autoflush〔Web コンテナサーバのカスタマイズ用キー〕　244
EJB コンテナとの連携　121
EJB 呼び出し時のモードの判別　679
Exception トレースログ　315
Exception トレースログ取得の設定　329
Exception トレースログとして出力される情報　315
Exception トレースログの出力項目　318
Exception トレースログのファイル　337

G

getBlockUpdate メソッド　414, 429
getBufferPoolSize メソッド　430
getBufSize メソッド　430
getDatabaseName メソッド　431
getDBEnv メソッド　431
getDBHostName メソッド　432
getDescription メソッド　432
getEncodLang メソッド　432
getExecuteDirectMode メソッド　408, 433
getHiRDBCursorMode メソッド　433
getJDBC_IF_TRC メソッド　434
getLONGVARBINARY_Access メソッド　435
getNetworkProtocol メソッド　435
getNotErrorOccurred メソッド　436
getOSAuthorize メソッド　436
getPassword メソッド　437
getPortNumber メソッド　437
getRMID メソッド　458
getRowSize メソッド　438
getServerName メソッド　438
getSQLWarningIgnore メソッド　439
getSV_EVENT_TRC メソッド　439
getTRC_NO メソッド　440
getUapName メソッド　440
getUser メソッド　441
getXACloseString メソッド　458
getXALocalCommitMode メソッド　459
getXAOpenString メソッド　459
getXAThreadMode メソッド　460

H

HiRDB 環境変数区切り文字　475
HiRDB の BLOB 型データ受取バッファサイズ　493
HiRDB の設定　292
HiRDB のディクショナリ表のデータ属性　474
HiRDB の場合のバッチ更新　344
hitachi_web.properties　142, 227
HTTP Cookie 情報の破棄　574
HTTP Cookie によるグローバルセッションの制御　573
HTTP Cookie による制御　564, 573
HTTP Cookie の有効期限　574
HttpSession オブジェクトの有効期限が切れた場合の処理　597

J

J2EE アプリケーション運用のための設定　285, 290
J2EE アプリケーション開始時の処理　589
J2EE アプリケーションのテスト機能　667
J2EE アプリケーションの動作モードの変更　668
J2EE アプリケーションを入れ替える場合にテストモードを使用する　685
J2EE サーバ起動時の処理〔セッションフェイルオーバ〕　588
J2EE サーバ停止時のセッション情報の引き継ぎ　561
J2EE サーバのコネクションクローズまたはコネクションの状態表示　80
J2EE サーバの設定　631
J2EE サーバ用ユーザプロパティファイル　60
java.naming.factory.initial〔Web コンテナサーバのカスタマイズ用キー〕　245

java.naming.provider.url〔Web コンテナサーバのカスタマイズ用キー〕　245
javagc コマンド　191
JavaVM のスレッドダンプ取得の設定　164
JavaVM ログ取得の設定　165
JavaVM ログファイル　165
javax.annotation パッケージに含まれるアノテーションのサポート範囲　533
javax.ejb パッケージに含まれるアノテーションのサポート範囲　534
Java アプリケーション用ユーザプロパティファイル　62
JDBC2.0 拡張機能で提供する機能のトレースを取得するための情報の指定　383
JDBC2.0 拡張機能の概要　360
JDBC2.0 拡張機能を使用したデータベース接続の設定　378
JDBC2.0 の概要　341
JdbcDbpsvDataSource クラス　427
JdbcDbpsvXADataSource クラス　429
JDBC SQL タイプの扱い　350
JDBC インタフェースメソッドのトレースの出力項目　327
JDBC ドライバのトレース（DABroker Library とのイベントトレース）　327
JDBC ドライバのトレース（JDBC インタフェースメソッドトレース）　326
JDBC ドライバのトレース情報の取得　329
JDBC ドライバのトレースファイル　338
JNDI からの情報の取得　382
JNDI への情報の登録　381
JNDI 連携　360
JSP コンパイル結果の出力先の設定〔サーブレットエンジンモード〕　149
JSP ソースのチェック　691
JSP タグライブラリ　169
JSP ファイル更新用インターバル　117, 152
JSP ファイルのコンパイル結果の保持　117, 146
JSP ファイルのコンパイル結果の保持の設定　149
JSP ファイルの再コンパイル　116
JSP ファイルの再コンパイルに関する設定項目と設定ファイル　153
JSP 用一時ディレクトリのディレクトリ構成　174
JTA によるトランザクション実装時の注意事項　39
jvm.type〔Web コンテナサーバ用オプション〕　232

L

LANG 環境変数　471
LimitRequestLine ディレクティブ　581

N

NLS_LANG　497

O

ORACLE_HOME　497
ORACLE_SID　497
Oracle と接続する場合の注意事項　395
ORACLE の LONG 型データ受取バッファサイズ　493
Oracle の設定　294
Oracle の場合のバッチ更新　349

P

PDCLTPATH　496
PDHOST　496
PDNAMEPORT　496
PDSQLTRACE　496
PDSWAITTIME　496
PDUAPERLOG　496
PreparedStatement クラス　410
PreparedStatement クラスでのバッチ更新　344
PRF トレース取得レベル　178
PRF トレースファイル　178

R

ResultSetMetaData クラス　422
ResultSet クラス　419
ResultSet タイプ　342
ResultSet の拡張　342
RMI-IIOP 通信のタイムアウト　145

S

SecurityManager 定義ファイル　257
SecurityManager による Web コンテナの実行時の保護　120
SessionSynchronization のメソッド呼び出しについての注意　83
setBlockUpdate メソッド　414, 441
setBufferPoolSize メソッド　442
setBufSize メソッド　443
setDatabaseName メソッド　444
setDBEnv メソッド　444
setDBHostName メソッド　445
setDescription メソッド　445
setEncodLang メソッド　447
setExecuteDirectMode メソッド　408, 447
setHiRDBCursorMode メソッド　448

setJDBC_IF_TRC メソッド　450
setLONGVARBINARY_Access メソッド　451
setNetworkProtocol メソッド　451
setNotErrorOccurred メソッド　452
setOSAuthorize メソッド　453
setPassword メソッド　453
setPortNumber メソッド　454
setRMID メソッド　461
setRowSize メソッド　454
setServerName メソッド　455
setSQLWarningIgnore メソッド　455
setSV_EVENT_TRC メソッド　456
setTRC_NO メソッド　456
setUapName メソッド　457
setUser メソッド　457
setXACloseString メソッド　461
setXALocalCommitMode メソッド　462
setXAOpenString メソッド　462
setXAThreadMode メソッド　463
SFO サーバ　555
SFO サーバアプリケーション　557
SFO サーバ停止時のセッション情報の引き継ぎ　561
SFO サーバでの J2EE サーバの切断検知　557
SFO サーバでのグローバルセッション情報の管理　555
SFO サーバに障害が発生した場合の HttpSession オブジェクトの最大数　563
SFO サーバに障害が発生した場合のグローバルセッション情報　563
SFO サーバの障害発生時のグローバルセッション ID の発行　563
SFO サーバの障害発生時のセッション情報の引き継ぎ　562
SFO サーバの設定　638
SFO サーバの複数配置　556
SFO サーバを複数配置するときの推奨構成　556
Stateful Session Bean 実装時の注意事項　83
Statement クラス　405
Statement クラスでのバッチ更新　343

U

uatags.jar　169
uatags.tld　169
URL 書き換えによって制御する場合の負荷分散　577
URL 書き換えによるグローバルセッションの制御　575
URL 書き換えによる制御　564
URL 書き換えによる制御の前提条件　575
URL 書き換えによる制御をする場合のアプリケーションの実装　606
usrconf.cfg　142, 229
usrconf.properties　60, 62, 142, 235

V

vbroker.orb.htc.comt.entryCount　163
vbroker.orb.htc.comt.entryCount〔Web コンテナサーバのカスタマイズ用キー〕　245
vbroker.orb.htc.comt.fileCount　163
vbroker.orb.htc.comt.fileCount〔Web コンテナサーバのカスタマイズ用キー〕　245
vbroker.orb.htc.tracePath　163
vbroker.orb.htc.tracePath〔Web コンテナサーバのカスタマイズ用キー〕　245

W

web.add.class.path　148
web.add.class.path〔Web コンテナサーバ用オプション〕　230
web.policy　142, 257
web.server.corefilenum〔Web コンテナサーバ用オプション〕　231
web.server.log.directory　147, 156, 163
web.server.log.directory〔Web コンテナサーバ用オプション〕　231
web.server.log.stderr.filesize〔Web コンテナサーバ用オプション〕　232
web.server.log.stdout.filesize〔Web コンテナサーバ用オプション〕　232
web.xml　169
web.xml の記述内容とサーバの動作　127
web.xml の記述内容とサーブレットエンジンモードの場合のサーバの動作　127
webserver.connector.ajp12.backlog〔Web コンテナサーバのカスタマイズ用キー〕　246
webserver.connector.ajp12.max_threads〔Web コンテナサーバのカスタマイズ用キー〕　246
webserver.connector.ajp12.port〔Web コンテナサーバのカスタマイズ用キー〕　246
webserver.connector.ajp13.backlog〔Web コンテナサーバのカスタマイズ用キー〕　246
webserver.connector.ajp13.bind_host〔Web コンテナサーバのカスタマイズ用キー〕　246
webserver.connector.ajp13.max_threads　145
webserver.connector.ajp13.max_threads〔Web コンテナサーバのカスタマイズ用キー〕　246

webserver.connector.ajp13.port〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.ajp13.receive_timeout 145
webserver.connector.ajp13.receive_timeout〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.ajp13.send_timeout〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.http.bind_host〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.http.bind_host〔バッチサーバ用のキー〕 519
webserver.connector.http.enabled 147
webserver.connector.http.enabled〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.http.permitted.hosts 147
webserver.connector.http.permitted.hosts〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.http.permitted.hosts〔バッチサーバ用のキー〕 519
webserver.connector.http.port〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.connector.limit.max_parameter_count〔Web コンテナサーバのカスタマイズ用キー〕247
webserver.connector.limit.max_post_form_data〔Web コンテナサーバのカスタマイズ用キー〕247
webserver.connector.redirect_https.port 146
webserver.connector.redirect_https.port〔Web コンテナサーバのカスタマイズ用キー〕 247
webserver.container.ac.logEnabled〔Web コンテナサーバのカスタマイズ用キー〕 248
webserver.container.server_id.enabled〔Web コンテナサーバのカスタマイズ用キー〕 248
webserver.container.server_id.name〔Web コンテナサーバのカスタマイズ用キー〕 248
webserver.container.server_id.value〔Web コンテナサーバのカスタマイズ用キー〕 248
webserver.container.servlet.default_mapping.enabled〔Web コンテナサーバのカスタマイズ用キー〕 249
webserver.context.check_interval〔Web コンテナサーバのカスタマイズ用キー〕 249
webserver.context.reload_delay_timeout〔Web コンテナサーバのカスタマイズ用キー〕 249
webserver.context.reload_delay_timeout〔Web コンテナサーバ用〕 227
webserver.context.reloadable〔Web コンテナサーバのカスタマイズ用キー〕 249
webserver.context.reloadable〔Web コンテナサーバ用〕 227
webserver.context.update.interval〔Web コンテナサーバのカスタマイズ用キー〕 249
webserver.context.update.interval〔Web コンテナサーバ用〕 228
webserver.errorpage.stack_trace.enabled〔Web コンテナサーバのカスタマイズ用キー〕 250
webserver.http.request.uri_decode.enabled〔Web コンテナサーバのカスタマイズ用キー〕 250
webserver.jsp.additional.import.list〔Web コンテナサーバのカスタマイズ用キー〕 250
webserver.jsp.check_interval〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.compile.backcompat〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.keepgenerated〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.keepgenerated〔Web コンテナサーバ用〕 228
webserver.jsp.recompilable〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.recompilable〔Web コンテナサーバ用〕 228
webserver.jsp.translation.backcompat.customAction.declareVariable〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.translation.backcompat.tag.noCheckRtexprvalue〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.translation.backcompat.tag.rtexprvalueTerminate〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.translation.backcompat.taglib.noCheckPrefix〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.translation.backcompat.useBean.noCheckClass〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.update.interval〔Web コンテナサーバのカスタマイズ用キー〕 251
webserver.jsp.update.interval〔Web コンテナサーバ用〕 228
webserver.servlet_api.exception.getCause.backcompat〔Web コンテナサーバのカスタマイズ用キー〕 252
webserver.session.persistence.enabled〔Web コンテナサーバのカスタマイズ用キー〕 253
webserver.session.persistence.enabled〔Web コンテナサーバ用〕 228

webserver.session.server_id.enabled〔Web コンテナサーバのカスタマイズ用キー〕 253
webserver.session.server_id.value〔Web コンテナサーバのカスタマイズ用キー〕 253
webserver.session.work.directory〔Web コンテナサーバのカスタマイズ用キー〕 253
webserver.sfo.check_size.mode〔J2EE サーバのカスタマイズ用キー〕 635
webserver.sfo.negotiation.ignore_serverId〔J2EE サーバのカスタマイズ用キー〕 634
webserver.sfo.sfo_server.<SFO サーバ名>.connectionTimeout〔J2EE サーバのカスタマイズ用キー〕 635
webserver.sfo.sfo_server.<SFO サーバ名>.naming〔J2EE サーバのカスタマイズ用キー〕 635
webserver.sfo.sfo_servers〔J2EE サーバのカスタマイズ用キー〕 635
webserver.shutdown.port〔Web コンテナサーバのカスタマイズ用キー〕 254
webserver.static_content.cache.enabled〔Web コンテナサーバのカスタマイズ用キー〕 254
webserver.static_content.cache.filesize.threshold〔Web コンテナサーバのカスタマイズ用キー〕254
webserver.static_content.cache.size〔Web コンテナサーバのカスタマイズ用キー〕 254
webserver.work.clean 146
webserver.work.clean〔Web コンテナサーバのカスタマイズ用キー〕 254
webserver.work.directory 146
webserver.work.directory〔Web コンテナサーバのカスタマイズ用キー〕 255
webserver.xml.validate〔Web コンテナサーバのカスタマイズ用キー〕 255
webserver.xml.validate〔Web コンテナサーバ用〕 228
Web アプリケーションに組み込むサーブレットフィルタ〔サーブレットエンジンモード〕 100
Web アプリケーションのアンデプロイ 171
Web アプリケーションのアンデプロイ〔サーブレットエンジンモード〕 107
Web アプリケーションの更新検知とリロード 109
Web アプリケーションの実行 107
Web アプリケーションの設定情報の削除 643
Web アプリケーションのディレクトリ構成 173
Web アプリケーションのデプロイ 107, 170
Web アプリケーションのデプロイと設定〔サーブレットエンジンモード〕 96
Web アプリケーションのユーザログ出力 147
Web アプリケーションのユーザログ出力の設定 167
Web アプリケーションのリロード 109
Web アプリケーションのリロードの条件 109
Web アプリケーションのリロード〔サーブレットエンジンモード〕 100
Web アプリケーション呼び出し時のモードの判別 678
Web コンテナコネクションプール機能の利用方法 131
Web コンテナサーバで使用するコマンド 209
Web コンテナサーバで使用するファイル 226
Web コンテナサーバの移行コマンドで自動的に追加される定義 206
Web コンテナサーバの開始 211
Web コンテナサーバのカスタマイズ用キー 236
Web コンテナサーバの起動 176
Web コンテナサーバの作業ディレクトリ 173
Web コンテナサーバのスレッドダンプの取得 210
Web コンテナサーバのセットアップ 143
Web コンテナサーバのセットアップとアンセットアップ 224
Web コンテナサーバの停止 177, 212
Web コンテナサーバの動作設定のカスタマイズ 144
Web コンテナサーバの予約済みキー 236
Web コンテナサーバのログ取得の設定 156
Web コンテナサーバ用 Web アプリケーションプロパティファイル 227
Web コンテナサーバ用オプション定義ファイル 229
Web コンテナサーバ用ユーザプロパティファイル 235
Web コンテナ単位での同時実行スレッド数の制御 145
Web コンテナのセキュリティ管理 120
Web コンテナの通信タイムアウト 145
Web サーバで使用するホストの固定 518
Web サーバの使用方法〔サーブレットエンジンモード〕 97
Web サーバまたは J2EE サーバの障害発生時のセッション情報の引き継ぎ 561
web-users.xml 143, 256

X

XA トレースの取得形式 325
XA トレースを取得する 476
XDM/RD E2 の設定 296

あ

アノテーション参照抑止機能の設定変更 536
アプリケーションサーバ 09-70 での主な機能変更 12

アプリケーションサーバの移行の概要〔サーブレットエンジンモードの場合〕 195
アプリケーションサーバの移行の手順〔サーブレットエンジンモードの場合〕 197
アプリケーションサーバの機能 1
アプリケーションのイベントリスナ 119
アプリケーションの更新検知とリロードの設定 150
アプリケーションの実行基盤としての機能 4
アプリケーションの実行基盤を運用・保守するための機能 5
アプリケーションのモード切り替え 686
アプリケーションを実行するときに追加される属性 608

い

移行前の確認〔サーブレットエンジンモードの場合〕 203
イベントトレース 324

え

エラー時の JDBC ドライバの処理 383
エラートレース 310
エラートレースのエラー事象を示す情報の出力項目 310
エラートレースのデータベース接続情報の出力項目 311
エラートレースの履歴情報の出力項目 312
エラートレースファイル 337

か

下位互換用のクラスローダ構成 51
回復機能 648
書き換え可能グローバルセッション情報 554
各実行環境の設定内容の例 656
拡張スレッドダンプ 165, 190
拡張データベースアクセストレース 305
拡張データベースアクセストレース設定項目
DABSQL_TRC_ATTACH（ATTACH） 482
DABSQL_TRC_ATTRGET（ATTRGET） 483
DABSQL_TRC_ATTRSET（ATTRSET） 483
DABSQL_TRC_AUTO_COMMIT_OFF（AUTO COMMIT OFF） 491
DABSQL_TRC_AUTO_COMMIT_ON（AUTO COMMIT ON） 491
DABSQL_TRC_BIND（BIND） 483
DABSQL_TRC_CANCEL（CANCEL） 478
DABSQL_TRC_CHARFORM（CHARFORM） 484
DABSQL_TRC_CHARID（CHARID） 484
DABSQL_TRC_CLOSE（CLOSE） 478
DABSQL_TRC_COMMIT（COMMIT） 478
DABSQL_TRC_CONNECT（CONNECT） 479
DABSQL_TRC_DECLARE（DECLARE） 479
DABSQL_TRC_DEFINE（DEFINE） 484
DABSQL_TRC_DESALLOC（DESALLOC） 485
DABSQL_TRC_DESCANY（DESCANY） 485
DABSQL_TRC_DESCRIBE（DESCRIBE） 479
DABSQL_TRC_DESFREE（DESFREE） 485
DABSQL_TRC_DETACH（DETACH） 486
DABSQL_TRC_DISCONNECT（DISCONNECT） 480
DABSQL_TRC_ENVCREATE（ENVCREATE） 486
DABSQL_TRC_ERRGET（ERRGET） 486
DABSQL_TRC_EXECUTE_DIRECT（EXECUTE DIRECT） 492
DABSQL_TRC_EXECUTE（EXECUTE） 480
DABSQL_TRC_FETCH（FETCH） 480
DABSQL_TRC_GET_INFO（GET INFO） 492
DABSQL_TRC_GETDIAG（GETDIAG） 491
DABSQL_TRC_GETPIECE（GETPIECE） 487
DABSQL_TRC_HDLALLOC（HDLALLOC） 487
DABSQL_TRC_HDLFREE（HDLFREE） 487
DABSQL_TRC_LOBCLOSE（LOBCLOSE） 488
DABSQL_TRC_LOBLEN（LOBLEN） 488
DABSQL_TRC_LOBOPEN（LOBOPEN） 488
DABSQL_TRC_LOBREAD（LOBREAD） 489
DABSQL_TRC_LOBWRITE（LOBWRITE） 489
DABSQL_TRC_LOGGING（LOGGING） 481
DABSQL_TRC_LOGLESS（LOGLESS） 481
DABSQL_TRC_OPEN（OPEN） 482
DABSQL_TRC_PARAMGET（PARAMGET） 489
DABSQL_TRC_PREPARE（PREPARE） 481

DABSQL_TRC_ROLLBACK（ROLLBACK） 482
DABSQL_TRC_SESBEGIN(SESBEGIN) 490
DABSQL_TRC_SESEND（SESEND） 490
DABSQL_TRC_SET_INFO(SET INFO) 492
DABSQL_TRC_SETPIECE(SETPIECE) 490
拡張データベースアクセストレースの出力項目 306
拡張データベースアクセストレースファイル 337
拡張データベースアクセストレースを取得する 475
拡張データベーストレースのファイル名称の規則 309
簡易 Web サーバ機能 126, 517
簡易 Web サーバ機能を使用するかどうかの設定 147
簡易 Web サーバへのアクセス許可 147
簡易 Web サーバへのアクセスの制御 518
漢字コードセット 493
管理機能用の Web アプリケーション 153
管理用サーバ 518

き

起動順序の依存関係 182
機能とマニュアルの対応 6
機能の分類 2
機能レイヤ 178
旧バージョンからの仕様変更の確認［サーブレットエンジンモードの場合］ 198

く

グローバルセッション ID 547
グローバルセッション ID の形式 547
グローバルセッション ID の送信に使用する URL のパスパラメタ 576
グローバルセッション ID を送信する HTTP Cookie の名称 574
グローバルセッション情報 547
グローバルセッション情報が引き継がれるセッションの条件 558
グローバルセッション情報の格納 593
グローバルセッション情報の更新 594
グローバルセッション情報のサイズの見積もり 568
グローバルセッション情報のサイズ見積もり機能 568
グローバルセッション情報の最大数 566
グローバルセッション情報の削除 595
グローバルセッション情報の種別 554
グローバルセッション情報の排他制御 565
グローバルセッション情報のロック 592
グローバルセッションの制御方法 564
グローバルセッションの制御方法が混在する場合のシステム構成 582
グローバルセッションを利用したセッション管理（メモリセッションフェイルオーバ機能） 546

け

検索データ取得時のデータ変換 350

こ

更新・検索条件カラム数 473
更新検知のインターバル 112, 117, 150, 152
構築済み実行環境の切り替えで出力するログの出力形式と出力項目（UNIX の場合） 665
構築済み実行環境の切り替えで出力するログの取得 664
構築済み実行環境の切り替え手順 656
構築済み実行環境の切り替えで使用するコマンド 660
構築済み実行環境の切り替えの概要 647
構築済み実行環境の切り替えの流れ 649
構築済み実行環境を切り替えるための準備 651
コネクションスイーパの動作 66
コネクションプーリングとコネクションスイーパの動作 65
コネクションプーリングの設定 153
コネクションプーリングの動作 65
コネクションを共有する場合の留意点 66
このマニュアルに記載している機能の説明 10
コンテキストルート 671
コンテキストルート名 671
コンテナ拡張ライブラリ 145
コンテナ拡張ライブラリの設定 148

さ

サーバの状態とグローバルセッション情報の引き継ぎ 560
サーブレットエンジンモード 85, 127
サーブレットエンジンモードで使用できる Web コンテナの機能 105
サーブレットエンジンモードでのシステム構築 138
サーブレットエンジンモードでのリソース接続 130
サーブレットエンジンモードと J2EE サーバモードの比較 95
サーブレットエンジンモードの場合の移行 195
サーブレットコンテキストへの属性追加 574
サーブレットと JSP のリファレンス定義 68
再コンパイル機能を使用するための設定 152
再コンパイルに関する設定 152
作業ディレクトリ 472
作業ディレクトリのパス長の見積もり式 174
削除機能 648

作成されるスレッドの種類と数　122
作成されるスレッドの総数　123
サブディレクトリ専有モード　521

し

システム内に複数の SFO サーバを配置する場合の設定例　620
システムの起動および停止方法〔サーブレットエンジンモード〕　103
システムの構築方法〔サーブレットエンジンモード〕　95
システムログの出力先　522
受信バッファのプーリングの設定　298
手動による定義などの修正〔サーブレットエンジンモードの場合〕　204
使用する JDBC ドライバの設定　366
使用する ORACLE のバージョン　474
使用するメモリ, スレッド数, ポート番号〔サーブレットエンジンモード〕　103
使用できる Java EE および J2EE の機能　20, 87
使用できるアプリケーションサーバの機能〔サーブレットエンジンモード〕　87
使用できるアプリケーションサーバの機能〔ベーシックモード〕　20
使用できる機能の一覧〔サーブレットエンジンモード〕　87
使用できる機能の一覧〔ベーシックモード〕　20
シリアライズおよびデシリアライズに失敗する原因　554
新規作成した J2EE アプリケーションをテストモードで実行する　684

す

スクロール型 ResultSet 使用時の注意点　343
スクロールタイプ　342

せ

静的コンテンツのキャッシュ　146
性能解析トレースの利用による J2EE アプリケーションの動作モードの確認　678
性能解析トレースファイル　178
セキュリティロール使用時の設定について　127
セッション情報の引き継ぎ処理　592
セッション情報ファイル　151
セッションフェイルオーバ用フィルタ　557
セッションフェイルオーバ用フィルタの処理　592
接続するデータベースの設定　366
接続できるリソース　32, 130
接続できるリソースとトランザクションの管理方法　32, 130
接続プール　360

た

ダイナミッククラスローディング機能　690
退避環境　647
退避環境一覧出力機能　648
退避環境作成時に退避対象となるファイル　658
退避機能　648
退避作業が必要なファイル　651
退避作業が必要なファイルと作業方法　651
退避作業が不要なファイル　512, 653

ち

チューニング項目　47

つ

通常モード　674
通信路の暗号化レベルの要請　146

て

定義などのバックアップ〔サーブレットエンジンモードの場合〕　203
ディスク容量の確認　655
データ型の対応　384
データソース　70
データソースから DB Connector への移行方法　42
データソース設定ファイル　62
データソース属性ファイル　63
データソースによるデータベースへの接続で使用できる機能　33
データソースのインポート　71
データソースの稼働情報監視　75
データソースの稼働情報監視（稼働情報）　78
データソースの稼働情報監視（基本情報）　77
データソースの削除　73
データソースの設定　53
データソースへのリファレンス定義　67
データベース ID の変更　214
データベース接続の前提条件　32, 130
データベース接続のための設定　281, 289
データベースとアプリケーション間の文字コード変換　50
データベースと接続中にトラブルが発生した場合 339
データベースとの接続　121
データベースとの接続〔サーブレットエンジンモード〕　98

データベースの設定　291
データベースへの接続　382
テストモード　674
テストモード使用の前提条件　667
テストモードで動作している EJB の呼び出し　672
テストモードで動作している J2EE アプリケーション情報の取得　678
テストモードで動作している J2EE アプリケーションに対してできる動作　674
テストモードで動作している J2EE アプリケーションに対する処理　668
テストモードで動作している Web アプリケーションの呼び出し　671
テストモードで動作中のアプリケーションに対してできる操作　668
テストモードでの Enterprise Bean 呼び出しについての注意事項　689
テストモードでの J2EE アプリケーションの動作確認（入れ替え時）　681
テストモードでの J2EE アプリケーションの動作確認（新規作成時）　679
テストモードでの J2EE アプリケーション呼び出し時の注意事項　688
テストモードでの Web アプリケーション呼び出しについての注意事項　688
テストモードによる J2EE アプリケーションの実行　684
テストモードを使用するための設定　668

と

統計情報の取得　678
統合ユーザ管理機能の設定　168
同時実行スレッド数の制御　123
ドメイン名指定でのトップページの表示　125, 171
トランザクションの管理方法　130
トレース共通ライブラリ　167
トレース共通ライブラリ形式　167
トレース取得サイズ　495
トレース取得ポイント　178
トレース取得レベル　495
トレースファイルサイズ　476
トレースを取得するための情報の設定　377

に

入出力ストリーム利用時の注意　520

ね

ネーミングサービスの通信タイムアウト　145
ネーミングのキャッシング　145
ネゴシエーションが失敗した場合の対処　591

は

バインド先アドレス設定機能　517
バッチ更新　343
パフォーマンスチューニング　47

ひ

表検索カラム数　473

ふ

プール管理情報設定ファイル　153, 259
プール管理情報の一覧表示　221
プール管理情報の削除　218
プール管理情報の登録と変更　219
プール内のコネクションの強制切断　217
プール内のコネクションのクリア　215
複数の構築済み実行環境の切り替え　647
プロセスの依存関係とプロセスの再起動方法　40
プロセスの再起動方法　182
分散トランザクション機能　361

へ

並行処理タイプ　342
ベーシックモード　17
ベーシックモードでデータベースに接続する場合の運用管理（互換用機能）　70
ベーシックモードでのリソース接続　32
ベーシックモードの概要　19
ベーシックモードの設定　53
ベーシックモードの場合のプロセスの依存関係　40
ベーシックモードの場合のプロセスの再起動方法　40

む

無効なグローバルセッション ID が送られた場合の処理　597

め

メッセージログ　305
メモリセッションフェイルオーバ機能　539, 540
メモリセッションフェイルオーバ機能使用時の障害検知　600

ゆ

ユーザスレッドの使用方法　83

ユーザ認証情報定義ファイル　256

よ

読み込み専用グローバルセッション情報　555

り

リクエスト実行時の処理　592
リクエストラインの最大サイズの設定　579
リソース更新用インターバル　112, 150
リソースへの接続テスト　38
リロード機能を使用するための設定　150
リロード時のセッション引き継ぎ　151
リロード処理の流れ　110
リロード遅延実行　112, 151
リロード時のセッション引き継ぎ　113
リロードに関する設定項目と設定ファイル　152

ろ

ロード対象のクラスとロード時に必要なクラスパス　531
ログサイズの変更　157
ログ出力先のサブディレクトリ数の管理　522
ログの出力先　188
ログの出力先の変更　156
ログの出力先や出力レベルの変更　147
ログの出力先を設定するユーザ定義ファイル（サーブレットエンジンモードの場合）　187
ログファイルサイズ［dabsv.log］　472
ログレベルの変更　157
ロック待ち　565