Sun Java System Application Server Enterprise Edition 8.1 2005Q2 管理ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–3510

060420@14774

目次

図目次

表目次

例目次

はじめに

この管理ガイドでは、Application Server のサブシステムとコンポーネントを、管理コンソールから設定、管理、配備する方法について説明します。

対象読者

この管理ガイドは、本稼働環境の情報技術管理者を対象としています。対象読者は、次のトピックに詳しいことを前提としています。

- 基本的なシステム管理のタスク
- ソフトウェアのインストール
- Web ブラウザの使用
- データベースサーバーの起動
- 端末ウィンドウでのコマンドの発行

お読みになる前に

Application Server は、Sun Java™ Enterprise System のコンポーネントであり、ネットワークおよびインターネット環境に分散したエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャーです。Sun Java Enterprise System に付属のマニュアルをよく読んで理解してください。マニュアルは、次の URL から入手できます。`http://docs.sun.com/app/docs/prod/entsys.05q4#hic`

内容の紹介

このガイドの構成は、Application Server を管理するためのブラウザベースのツールである管理コンソールの配置に対応しています。各章は、コンセプトの説明に始まり、管理コンソールで特定のタスクを実行する方法を説明する、手順の節が続きます。

表記上の規則

この節の表は、このマニュアルで使用されている表記規則を示しています。

表記上の規則

次の表に、このマニュアルの書体の表記の種類を示します。

表 P–1 表記上の規則

書体	意味	例
`AaBbCc123` (モノスペース)	API および言語要素、HTML タグ、Web サイト URL、コマンド名、ファイル名、ディレクトリパス名、画面上のコンピュータ出力、サンプルコード。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `% You have mail.`
`AaBbCc123` (太字のモノスペース)	ユーザーが入力する文字を、画面上のコンピュータ出力とは区別して表示します。	`%` **`su`** `Password:`
AaBbCc123 (イタリック)	マニュアルのタイトル、新規用語、強調する単語。 実際の名前または値に置き換えられるコマンドのプレースホルダまたはパス名。	ユーザーズガイドの第 6 章を参照してください。 これらを *class* オプションと呼びます。 ファイルを保存しないでください。 このファイルは、*install-dir*`/bin` ディレクトリにあります。

記号

次の表に、このマニュアルに使用されている記号の表記規則を示します。

表 P–2 記号の表記規則

記号	説明	例	意味
`[ ]`	コマンドオプションの選択肢が含まれます。	`ls [-l]`	`-l` オプションは必須ではありません。
`{ | }`	必要なコマンドオプションの選択肢のセットが含まれます。	`-d {y|n}`	`-d` オプションとともに、`y` 引数または `n` 引数を指定する必要があります。

表 P–2 記号の表記規則　(続き)

記号	説明	例	意味
-	同時に実行する複数のキーストロークを結び付けます。	Control-A	コントロールキーを押しながらAキーを押します。
+	連続して実行する複数のキーストロークを結び付けます。	Ctrl+A+N	コントロールキーを押して放し、続いて次のキーを押します。
>	グラフィカルユーザーインタフェースのメニューの選択を表示します。	「ファイル」>「新規」>「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

デフォルトのパスとファイル名

次の表に、このマニュアルのデフォルトのパスとファイル名を示します。

表 P–3 デフォルトのパスとファイル名

項	説明

install-dir	デフォルトでは、Application Server インストールディレクトリは次の場所に置かれます。
	■ Solaris プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/opt/SUNWappserver/appserver`
	■ Linux プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/opt/sun/appserver/`
	■ Solaris および Linux プラットフォームへのインストールで、ルートユーザーでない場合:
	ユーザーのホームディレクトリ`/SUNWappserver`
	■ Solaris および Linux プラットフォームへのインストールで、ルートユーザーである場合:
	`/opt/SUNWappserver`
	■ Windows のすべてのインストールの場合:
	SystemDrive`:\Sun\AppServer`

表P–3デフォルトのパスとファイル名　(続き)

項	説明

domain-root-dir	デフォルトでは、すべてのドメインを含むディレクトリは次の場所に置かれます。
	■ Solaris プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/var/opt/SUNWappserver/domains/`
	■ Linux プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/var/opt/sun/appserver/domains/`
	■ そのほかのすべてのインストールの場合:
	install-dir`/domains/`

domain-dir	デフォルトでは、各ドメインディレクトリは次の場所に置かれます。 *domain-root-dir*`/`*domain-dir* 設定ファイルには、次のように表される *domain-dir* があります。 `${com.sun.aas.instanceRoot}`
instance-dir	デフォルトでは、各インスタンスディレクトリは次の場所に置かれます。 *domain-dir*`/`*instance-dir*

シェルプロンプト

次の表は、このマニュアルのシェルプロンプトを示します。

表P–4シェルプロンプト

シェル	プロンプト
UNIX または Linux の C シェル	*machine-name*`%`
UNIX または Linux の C シェルスーパーユーザー	*machine-name*`#`
UNIX または Linux の Bourne シェルと Korn シェル	`$`
UNIX または Linux の Bourne シェルおよび Korn シェルスーパーユーザー	`#`
Windows コマンド行	`C:\`

記号

次の表に、このマニュアルに使用されている記号の表記規則を示します。

表 P–5 記号の表記規則

記号	説明	例	意味
[ ]	コマンドオプションの選択肢が含まれます。	ls [-l]	-l オプションは必須ではありません。
{ | }	必要なコマンドオプションの選択肢のセットが含まれます。	-d {y|n}	-d オプションとともに、y 引数または n 引数を指定する必要があります。
-	同時に実行する複数のキーストロークを結び付けます。	Control-A	コントロールキーを押しながら A キーを押します。
+	連続して実行する複数のキーストロークを結び付けます。	Ctrl+A+N	コントロールキーを押して放し、続いて次のキーを押します。
>	グラフィカルユーザーインタフェースのメニューの選択を表示します。	「ファイル」>「新規」>「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

デフォルトのパスとファイル名

次の表に、このマニュアルのデフォルトのパスとファイル名を示します。

表P–6デフォルトのパスとファイル名

項	説明

install-dir	デフォルトでは、Application Server インストールディレクトリは次の場所に置かれます。
	■ Solaris プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/opt/SUNWappserver/appserver`
	■ Linux プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/opt/sun/appserver/`
	■ Solaris および Linux プラットフォームへのインストールで、ルートユーザーでない場合:
	ユーザーのホームディレクトリ`/SUNWappserver`
	■ Solaris および Linux プラットフォームへのインストールで、ルートユーザーである場合:
	`/opt/SUNWappserver`
	■ Windows のすべてのインストールの場合:
	SystemDrive`:\Sun\AppServer`

domain-root-dir	デフォルトでは、すべてのドメインを含むディレクトリは次の場所に置かれます。
	■ Solaris プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/var/opt/SUNWappserver/domains/`
	■ Linux プラットフォームへの Sun Java Enterprise System インストールの場合:
	`/var/opt/sun/appserver/domains/`
	■ そのほかのすべてのインストールの場合:
	install-dir`/domains/`

domain-dir	デフォルトでは、各ドメインディレクトリは次の場所に置かれます。 *domain-root-dir*`/`*domain-dir* 設定ファイルには、次のように表される *domain-dir* があります。 `${com.sun.aas.instanceRoot}`

表 P–6 デフォルトのパスとファイル名　(続き)

項	説明
instance-dir	デフォルトでは、各インスタンスディレクトリは次の場所に置かれます。 *domain-dir*/*instance-dir*

関連マニュアル

http://docs.sun.com[SM] Web サイトでは、Sun が提供しているオンラインマニュアルを参照することができます。アーカイブをブラウズすることも、特定のマニュアルのタイトルまたは主題を検索することもできます。

正式な仕様の URL のディレクトリを入手するには、*install-dir*/docs/index.htm にアクセスしてください。さらに、次のリソースが役立つことがあります。

一般的な **J2EE** 情報**:**

『*J2EE 1.4 Tutorial*』 http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

『*J2EE Blueprints*』 http://java.sun.com/reference/blueprints/index.html

『*Core J2EE Patterns:Best Practices and Design Strategies*』 Deepak Alur (著)、 John Crupi (著)、 Dan Malks (著)、 Prentice Hall Publishing

『*Java Security*』 Scott Oaks (著)、 O'Reilly Publishing

サーブレットと **JSP** ファイルを使用したプログラミング**:**

『*Java Servlet Programming*』 Jason Hunter (著)、 O'Reilly Publishing

『*Java Threads, 2nd Edition*』 Scott Oaks (著)、 Henry Wong (著)、 O'Reilly Publishing

EJB コンポーネントを使用したプログラミング**:**

『*Enterprise JavaBeans*』 Richard Monson-Haefel (著)、 O'Reilly Publishing

JDBC を使用したプログラミング**:**

『*Database Programming with JDBC and Java*』 George Reese (著)、 O'Reilly Publishing

Java を使用した *JDBC* データベースアクセス*:*『*A Tutorial and Annotated Reference (Java Series)*』 Graham Hamilton (著)、 Rick Cattell (著)、 Maydene Fisher (著)

Javadoc:

Application Server に付属するパッケージの Javadoc は、*install-dir*/docs/api にあります。

このマニュアルセットの内容

Sun Java System Application Server マニュアルは、Portable Document Format (PDF) およびハイパーテキストマークアップ言語 (HTML) のオンラインファイルとして入手可能です。

次の表に、Application Server の主要なマニュアルセットに含まれるマニュアルの概要を示します。

表 P–7 このマニュアルセットの内容

タイトル	説明
『リリースノート』	ソフトウェアとマニュアルに関する最新情報。サポートされているハードウェア、オペレーティングシステム、JDK、および JDBC/RDBMS の包括的な表ベースの概要を含みます。
『クイックスタートガイド』	Application Server 製品の使用を開始するための手順。
『インストールガイド』	Application Server ソフトウェアとそのコンポーネントのインストール。
『配備計画ガイド』	管理サイトに最適な方法で Sun Java System Application Server を配備しているかどうかを確認するための、システムニーズと企業の評価。Application Server を配備する際に注意すべき一般的な問題や懸案事項についても説明しています。
『開発者ガイド』	J2EE コンポーネントおよび API 用のオープン Java 標準モデルに従い、Sun Java System Application Server 上で実行することを目的とする Java 2 Platform, Enterprise Edition (J2EE プラットフォーム) アプリケーションの作成と実装。開発者ツール、セキュリティー、アセンブリ、配備、デバッグ、ライフサイクルモジュールの作成に関する一般情報を含みます。
『J2EE 1.4 Tutorial』	J2EE アプリケーションの開発のための J2EE 1.4 プラットフォーム技術および API の使用、および Sun Java System Application Server でのアプリケーションの配備。
『管理ガイド』	管理コンソールからの、Sun Java System Application Server のサブシステムとコンポーネントの設定、管理、および配備。
『高可用性 (HA) 管理ガイド』	高可用性データベースのインストール後の設定方法と管理方法。
『Administration Reference』	Sun Java System Application Server 設定ファイル `domain.xml` の編集。
『アップグレードと移行』	新しい Sun Java System Application Server プログラミングモデルへのアプリケーションの移行。特に Application Server 6.x および 7 からの移行。このガイドでは、製品仕様の非互換性をもたらす可能性のある、隣接した製品リリース間の相違点や設定オプションについても説明しています。

表 P–7 このマニュアルセットの内容　(続き)

タイトル	説明
『Performance Tuning Guide』	パフォーマンスを向上させるための Sun Java System Application Server のチューニング。
『Troubleshooting Guide』	Sun Java System Application Server の問題解決。
『Error Message Reference』	Sun Java System Application Server エラーメッセージの解決。
『リファレンスマニュアル』	マニュアルページスタイルで記述された Sun Java System Application Server で利用可能なユーティリティーコマンド。コマンド行インタフェース `asadmin` を含みます。

その他のサーバーのマニュアル

ほかのサーバーのマニュアルを参照するには、次の URL にアクセスしてください。

- Message Queue のマニュアル http://docs.sun.com/db?p=prod/s1.s1msgqu (`http://docs.sun.com/db?p=prod/s1.s1msgqu`)
- Directory Server のマニュアル http://docs.sun.com/coll/DirectoryServer_04q2 (`http://docs.sun.com/coll/DirectoryServer_04q2`)
- Web Server のマニュアル http://docs.sun.com/coll/S1_websvr61_en (`http://docs.sun.com/coll/S1_websvr61_en`)

Sun のリソースにオンラインでアクセスする

製品ダウンロード、プロフェッショナルサービス、パッチとサポート、および、詳細な開発情報については、次の URL にアクセスしてください。

- ダウンロードセンター http://wwws.sun.com/software/download/ (`http://wwws.sun.com/software/download/`)
- プロフェッショナルサービス http://www.sun.com/service/sunps/sunone/index.html (`http://www.sun.com/service/sunps/sunone/index.html`)
- Sun Enterprise Service、Solaris パッチ、およびサポート http://sunsolve.sun.com/ (`http://sunsolve.sun.com/`)
- 開発者情報 http://developers.sun.com/prodtech/index.html (`http://developers.sun.com/prodtech/index.html`)

Sun テクニカルサポートの連絡先

製品のマニュアルに記載されていない、技術上の問題が発生した場合、http://www.sun.com/service/contacting (http://www.sun.com/service/contacting) にアクセスしてください。

関連するサードパーティーの Web サイトの参照

このマニュアルに記載されたサードパーティーの Web サイトの利用可能性について Sun は責任を負いません。これらのサイトやリソースを通して入手される内容、広告、製品、およびその他の資料について、Sun は保証することも、責任を負うこともありません。これらのサイトやリソースを通して入手される内容、物品、およびサービスを使用または信用することにより発生する、実際の、または申し立てられている損害や損失について、Sun は賠償責任などいかなる責任も負いません。

ご意見をお寄せください

Sun では、マニュアルの改善に努力しており、お客様からのご意見やご提案を歓迎いたします。

ご意見をお送りいただくには、http://docs.sun.com (http://docs.sun.com) にアクセスして、「コメントの送信」をクリックしてください。オンラインフォームに、マニュアルのタイトルと Part No を記入してください。Part No は、マニュアルのタイトルページまたは上部に記載された 7 桁または 9 桁の番号です。たとえば、このマニュアルのタイトルは『Sun Java System Application Server 2005Q2 管理ガイド』で、Part No は 819-3510 です。

◆ ◆ ◆ 第 1 章

概要

この章では、Sun Java™ System Application Server について説明し、基本的な管理タスクを紹介します。この章には次の節が含まれています。

Sun Java System Application Server について

Application Server とは

Application Server は、エンタープライズアプリケーションの開発、配備、および管理のための堅牢な J2EE プラットフォームを提供します。主な機能に、トランザクション管理、パフォーマンス、スケーラビリティー、セキュリティー、および統合があります。Application Server は、Web パブリッシングから企業規模のトランザクション処理までを幅広くサポートします。一方、開発者は Application Server を利用して、JavaServer Pages (JSP)、Java サーブレット、および Enterprise JavaBeans (EJB) テクノロジをベースにしたアプリケーションを構築できます。

Application Server Enterprise Edition は、高度なクラスタリング技術とフェイルオーバー技術を提供します。これらの機能により、スケーラブルで高い可用性を備えた J2EE アプリケーションを実行できます。

- クラスタリング - クラスタは、1 つの論理エンティティーとして一体となって動作するアプリケーションサーバーインスタンスの集まりです。クラスタ内の各 Application Server インスタンスは同じように設定され、各インスタンスには同じアプリケーションが配備されています。

  クラスタに Application Server インスタンスを追加することによってシステムの容量が増加し、水平的なスケーリングが実現されます。サービスを中断せずに、クラスタに Application Server インスタンスを追加することができます。HTTP、RMI/IIOP、および JMS ロードバランスシステムは、クラスタ内の正常な Application Server インスタンスに要求を分散させます。
- 高可用性 - 可用性を有効にすると、クラスタ内の Application Server インスタンスをフェイルオーバーによって保護できます。1 つのアプリケーションサーバーインスタンスが停止すると、利用できなくなったサーバーに割り当てられていたセッションは別の Application Server インスタンスに引き継がれます。セッションの情報は、高可用性データベース (HADB) に格納されます。HADB は、持続的な HTTP セッションとステートフルセッション Beans をサポートします。

Application Server のアーキテクチャー

ここでは、図 1–1 に示す Application Server のハイレベルアーキテクチャーについて説明します。

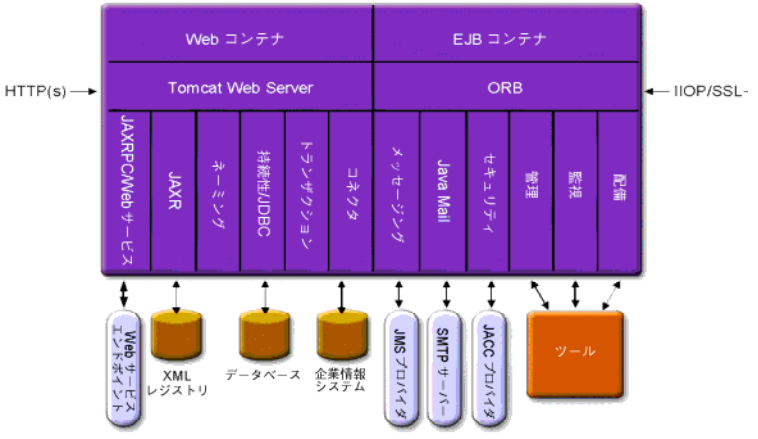

図 1–1 Application Server のアーキテクチャー

- コンテナ - コンテナは、J2EE コンポーネントのセキュリティーやトランザクション管理などのサービスを提供する実行環境です。図 1–1 は、2 つのタイプの J2EE コンテナ、Web および EJB を示しています。JSP ページやサーブレットなどの Web コンポーネントは、Web コンテナ内で実行されます。EJB テクノロジのコンポーネントである Enterprise JavaBeans は、EJB コンテナ内で実行されます。

- クライアントアクセス - 実行時に、ブラウザクライアントはインターネット上で使われているプロトコルである HTTP で Web サーバーと通信することにより Web アプリケーションにアクセスします。HTTPS プロトコルは、セキュア通信を必要とするアプリケーションのためにあります。Enterprise Bean クライアントは、IIOP または IIOP/SSL (セキュア) プロトコルを介して ORB (Object Request Broker) と通信します。Application Server には、HTTP、HTTPS、IIOP、および IIOP/SSL プロトコル用の別個のリスナーがあります。各リスナーは、固有のポート番号を排他的に使用しています。
- **Web** サービス - J2EE プラットフォームでは、Java API for XML-Based RPC (JAX-RPC) によって実装された Web サービスを提供する Web アプリケーションを配備できます。J2EE アプリケーションやコンポーネントは、ほかの Web サービスのクライアントにすることもできます。アプリケーションは Java API for XML Registries (JAXR) を介して XML レジストリにアクセスします。
- アプリケーションのサービス - J2EE プラットフォームは、コンテナがアプリケーションのサービスを提供するように設計されています。図 1–1 は、次のサービスを示しています。
  - ネーミング - ネーミングおよびディレクトリサービスは、オブジェクトに名前をバインドします。J2EE アプリケーションは、JNDI 名を探してオブジェクトを検出します。JNDI は Java Naming and Directory Interface API の略です。
  - セキュリティー - Java Authorization Contract for Containers (JACC) は、J2EE コンテナ用に定義された一連のセキュリティー規約です。クライアントの ID に基づいて、コンテナはコンテナのリソースおよびサービスに対するアクセスを制限します。
- トランザクション管理 - トランザクションは作業の分割不能な単位です。たとえば、銀行口座間での資金の振り替えがトランザクションにあたります。トランザクション管理サービスは、トランザクションが完全に終了するか、またはロールバックされるようにします。

外部システムへのアクセス

J2EE プラットフォームでは、アプリケーションが Application Server の外部にあるシステムにアクセスできます。アプリケーションは、リソースと呼ばれるオブジェクトを介してこれらのシステムにアクセスします。管理者はリソース設定を行う必要があります。J2EE プラットフォームでは、次の API およびコンポーネントを介して外部システムにアクセスできます。

- **JDBC** - データベース管理システム (DBMS) は、データの格納、編成、および検索機能を提供します。大部分のビジネスアプリケーションは、アプリケーションが JDBC API 経由でアクセスするリレーショナルデータベースにデータを格納します。データベース内の情報は、多くの場合、持続性があるとされています。これは、ディスク上に保存され、アプリケーションを終了したあとも存在するためです。Application Server バンドルには PointBase DBMS が組み込まれています。
- メッセージング - メッセージングは、ソフトウェアコンポーネント間またはアプリケーション間の通信メソッドです。メッセージングクライアントは、ほかのどのクライアントともメッセージの送受信を行います。アプリケーションは Java Messaging

Service (JMS) API を介してメッセージングプロバイダにアクセスします。Application Server には JMS プロバイダが組み込まれています。

- コネクタ - J2EE コネクタアーキテクチャーでは、J2EE アプリケーションと既存のエンタープライズ情報システム (EIS) との統合が可能です。アプリケーションは、コネクタまたはリソースアダプタと呼ばれる移行可能な J2EE コンポーネントを介して EIS にアクセスします。
- **JavaMail** - JavaMail API を介して、アプリケーションは電子メールを送受信するために SMTP サーバーに接続します。
- サーバー管理 - 図 1-1 の右下に、Application Server の管理者によって実行されるタスクの一部が示されています。たとえば、管理者は、アプリケーションを配備 (インストール) し、サーバーのパフォーマンスを監視します。これらのタスクは Application Server が提供する管理ツールを使用して実行します。

管理用ツール

- Application Server には、次の 3 つの管理ツールが含まれます。
  - 34 ページの「管理コンソール」
  - 35 ページの「asadmin ユーティリティー」
  - 35 ページの「Application Server Management Extension (AMX)」

管理コンソール

管理コンソールは、ナビゲートしやすいインタフェースとオンラインヘルプを装備したブラウザベースのツールです。このマニュアルでは、管理コンソールの使用手順を順を追って説明します。管理コンソールを使用するには、管理サーバーが稼動している必要があります。

Application Server をインストールするときに、サーバーのポート番号を選択します。選択しなかった場合は、デフォルトポートの 4849 が使用されます。また、ユーザー名とマスターパスワードも指定します。

管理コンソールを起動するには、Web ブラウザで次のように入力します。

```
https://hostname:port
```

次に例を示します。

```
https://kindness.sun.com:4849
```

管理コンソールを Application Server がインストールされたマシンで実行する場合は、ホスト名として `localhost` を指定します。

Windows で、「スタート」メニューから「Application Server 管理コンソール」を起動します。

インストールプログラムにより、domain1 という名前のデフォルト管理ドメインがデフォルトポート番号 4849 で生成されます。また、ドメイン管理サーバー (DAS) とは分離したインスタンスも作成されます。インストール後は、管理ドメインを作成して追加できます。各ドメインは、一意のポート番号を持ったドメイン管理サーバーをそれぞれ持っています。管理コンソールの URL を指定する場合は、管理するドメインのポート番号を使用してください。

設定にリモートサーバーインスタンスが含まれる場合は、ノードエージェントを作成してリモートサーバーインスタンスを容易に管理できるようにします。サーバーインスタンスの作成、起動、停止、および削除は、ノードエージェントの役割です。ノードエージェントを設定するには、コマンド行インタフェース (CLI) のコマンドを使用します。

asadmin ユーティリティー

asadmin ユーティリティーは、コマンド行ツールです。asadmin ユーティリティーと、このユーティリティーに関連するコマンドを使用して、管理コンソールで実行可能な一連の同じタスクを実行します。たとえば、ドメインの起動と停止、サーバーの設定、アプリケーションの配備などを実行します。

シェルのコマンドプロンプトからこれらのコマンドを使用するか、または別のスクリプトやプログラムからこれらのコマンドを呼び出します。これらのコマンドを使用して、定型管理タスクを自動化します。

asadmin ユーティリティーを起動するには、次のように入力します。

```
$ asadmin
```

asadmin 内で使用可能なコマンドを一覧表示するには、次のように入力します。

```
asadmin> help
```

シェルのコマンドプロンプトで、asadmin コマンドを次のように実行することもできます。

```
$ asadmin help
```

コマンドの構文と例を表示するには、help のあとにコマンド名を入力します。次に例を示します。

```
asadmin> help create-jdbc-resource
```

指定したコマンドの asadmin help 情報が、コマンドの UNIX マニュアルページに表示されます。これらのマニュアルページは HTML 形式でも利用できます。

Application Server Management Extension (AMX)

Application Server Management Extension は、すべての Application Server 設定を表示する API であり、AMX インタフェースを実装する、使いやすいクライアント側の動的なプロキシとして JMX 管理対象 Beans を監視しています。

Application Server Management Extension の使用についての詳細は、『Application Server 開発者ガイド』の JMX に関する章を参照してください。

Application Server の設定

Application Server の設定

Application Server ドメインは、管理者によるシステム設定の管理を容易にするための作成される、論理的または物理的なユニットです。ドメインは、インスタンスとノードエージェントを含む、より小さなユニットに分割されます。サーバーインスタンスは、単一の物理マシンで Application Server を実行する単一の Java 仮想マシン (JVM) です。各ドメインには 1 つ以上のインスタンスがあります。また、適切に機能するためにドメインには、インスタンス用に少なくとも 1 つの関連するノードエージェントが必要です。ドメインをグループ化してクラスタを生成できます。クラスタでは、管理者によるソフトウェアとハードウェアのグループ管理が可能です。

ドメインの設定

管理ドメインは基本的なセキュリティー構造を提供し、これによって各種の管理者がアプリケーションサーバーインスタンスの特定のグループ (ドメイン) を管理できます。サーバーインスタンスを個別のドメインにグループ化することにより、さまざまな組織や管理者が 1 つの Application Server インストールを共有できます。各ドメインには、固有の設定、ログファイル、およびアプリケーションの配備領域があり、これらはほかのドメインとは無関係です。1 つのドメインの設定が変更されても、ほかのドメインの設定は影響を受けません。

それぞれの管理コンソールセッションでは、ドメインを設定および管理できます。複数のドメインを作成している場合は、追加の管理コンソールセッションを起動して、各ドメインを管理する必要があります。各ドメインは、一意のポート番号を持ったドメイン管理サーバー (DAS) を持っています。各管理ドメインは、複数のアプリケーションサーバーインスタンスを保有できます。ただし、アプリケーションサーバーインスタンスは 1 つのドメインにのみ属することができます。Application Server がインストールされると、`domain1` という名前の管理ドメインが自動的に作成されます。

ドメインの作成

ドメインは、create-domain コマンドによって作成されます。次のコマンド例では、mydomain というドメインを作成します。管理サーバーが待機するポートは 1234 で、管理ユーザー名は hanan です。このコマンドは、管理パスワードおよびマスターパスワードをプロンプトします。

```
$ asadmin create-domain --adminport 80 --adminuser hanan mydomain
```

mydomain ドメインの管理コンソールをブラウザ内で起動するには、次の URL を入力します。

```
http://hostname:80
```

前述の create-domain の例の場合、ドメインのログファイル、設定ファイル、および配備されたアプリケーションは次のディレクトリに置かれます。

domain-root-dir/mydomain

ドメインのディレクトリを別の位置に作成するには、--domaindir オプションを指定します。コマンドの完全な構文を確認するには、asadmin help create-domain と入力してください。

ドメインの削除

ドメインは、asadmin delete-domain コマンドによって削除されます。ドメインを管理できる OS ユーザー (またはルート) だけが、このコマンドを正常に実行できます。たとえば、mydomain というドメインを削除するには、次のコマンドを入力します。

```
$ asadmin delete-domain mydomain
```

ドメインの一覧表示

マシン上に作成されているドメインを asadmin list-domains コマンドを使用して参照できます。デフォルトの *domain-root-dir* ディレクトリ内のドメインを一覧表示するには、次のコマンドを入力します。

```
$ asadmin list-domains
```

別のディレクトリに作成されているドメインを一覧表示するには、--domaindir オプションを指定します。

ドメインの起動

ドメインの起動時に、管理サーバーとアプリケーションサーバーインスタンスが起動されます。アプリケーションサーバーインスタンスは、一度起動すると常時稼動となり、要求を待機して受け付けます。各ドメインは、別々に起動する必要があります。

ドメインを起動するには、asadmin start-domain コマンドを入力し、ドメイン名を指定します。たとえば、デフォルトのドメイン (domain1) を起動するには、次のように入力します。

```
$ asadmin start-domain --user admin domain1
```

ドメインが 1 つだけの場合は、ドメイン名を省略します。コマンドの完全な構文を確認するには、asadmin help start-domain と入力してください。パスワードデータを省略した場合は、入力するように要求されます。

asadmin start-domain domain1 コマンドは、サーバーをインストールしたユーザーが実行するようにしてください。インストールユーザー以外が実行した場合、.asadmintruststore が、インストールユーザーのホームディレクトリから、実行ユーザーのホームディレクトリに移動またはコピーされます。

このファイルが、インストールユーザーのホームディレクトリから実行ユーザーのホームディレクトリに (コピーではなく) 移動された場合、アップグレードやインストールを行うユーザーのホームディレクトリに .asadmintruststore ファイルが存在しなくなってしまうため、アプリケーションのアップグレードが正常に実行できなくなる可能性があります。

削除された .asadmintruststore ファイルの復元

ドメインの .asadmintruststore ファイルが削除されてしまった場合、新しいファイルを簡単に作成できます。

▼ 新しい .asadmintruststore ファイルを作成する

1. ローカルの asadmin start-domain コマンドを使用して、管理するドメインを起動します。

   ローカルの asadmin コマンドでドメインを起動する場合、.asadmintruststore ファイルは必要ありません。

2. リモートの asadmin コマンドのいずれかを実行します。

   リモートの asadmin コマンドでは、--user、--passwordfile (--password)、--host、および --port オプションを指定し、またターゲットドメインが実行中である必要があります。

3. 確認画面で「**y**」と入力して、新しいドメイン証明書を受け入れます。

Windows でデフォルトのドメインを起動するには、次の手順に従います。

Windows の「スタート」メニューで、「プログラム」 - >「Sun Microsystems」 - >「Application Server」 - >「管理サーバーを起動」を選択します。

サーバーまたはドメインの再起動

サーバーの再起動の手順はドメインの再起動と同じです。ドメインまたはサーバーを再起動するには、ドメインをいったん停止してから起動します。

ドメインの停止

ドメインを停止すると、そのドメインの管理サーバーとアプリケーションサーバーインスタンスがシャットダウンします。ドメインを停止すると、そのサーバーインスタンスは新しい接続を受け付けなくなり、未完了の接続がすべて完了するまで待機します。サーバーインスタンスはシャットダウンプロセスを完了しなければならないため、これには数秒間かかります。ドメインの停止処理中は、管理コンソールおよびほとんどの `asadmin` コマンドが使用できません。

ドメインを停止するには、`asadmin stop-domain` コマンドを入力し、ドメイン名を指定します。たとえば、デフォルトのドメイン (`domain1`) を停止するには、次のように入力します。

```
$ asadmin stop-domain domain1
```

ドメインが 1 つだけの場合は、ドメイン名を省略します。コマンドの完全な構文を確認するには、`asadmin help stop-domain` と入力してください。

管理コンソールを使用してドメインを停止するには、次の手順に従います。

- ツリーコンポーネントで、スタンドアロンインスタンスノードの下にあるサーバー (管理サーバー) を選択します。
- 「一般情報」ページで、「インスタンスの停止」をクリックします。

Windows でデフォルトのドメインを停止するには、次の手順に従います。

「スタート」メニューで、「プログラム」 - >「Sun Microsystems」 - >「Application Server」 - >「管理サーバーを停止」を選択します。

ドメイン管理サーバーの再作成

ミラーリングを行うため、および、ドメイン管理サーバー (DAS) の有効なコピーを提供するためには、次のものを用意する必要があります。

- 元の DAS を含むマシン 1 台 (machine1)
- アプリケーションを実行してクライアントの要求を満たすサーバーインスタンスを持つクラスタを含む 2 台目のマシン (machine2)。クラスタは、1 台目のマシンの DAS を使用して設定されます。
- 1 台目のマシンがクラッシュした場合に DAS を再作成する必要がある 3 台目のバックアップマシン (マシン 3)

注 – 1 台目のマシンの DAS のバックアップを維持する必要があります。`asadmin backup-domain` を使用して、現在のドメインをバックアップしてください。

▼ DAS を移行する

ドメイン管理サーバーを 1 台目のマシン (machine1) から 3 台目のマシン (machine3) に移行するには、次の手順が必要です。

1 **1 台目のマシンと同様に、Application Server を 3 台目のマシンにインストールします。**

この処理は、DAS が 3 台目のマシンに正常に復元されて、パスの競合を発生させないために必要です。

a. コマンド行 (対話型) モードを使用して、**Application Server** 管理パッケージをインストールします。対話型のコマンド行モードを有効にするには、`console` オプションを次のように指定してインストールプログラムを起動します。

./bundle-filename `-console`

コマンド行インタフェースを使用してインストールを行うには、ルートのアクセス権が必要です。

b. オプションの選択を解除して、デフォルトのドメインをインストールします。

バックアップされたドメインの復元は、同じアーキテクチャーおよびまったく同じインストールパスを持つ 2 台のマシンでのみサポートされます (すなわち両方のマシンが同じ *install-dir* と *domain-root-dir* を使用する)。

2 **1 台目のマシンのバックアップ ZIP ファイルを、3 台目のマシンの *domain-root-dir* にコピーします。FTP でファイルを転送することもできます。**

3 `asadmin restore-domain` コマンドを実行して、**ZIP** ファイルを **3** 台目のマシンに復元します。

```
asadmin restore-domain --filename domain-root-dir/sjsas_backup_v00001.zip domain1
```

任意のドメインをバックアップできます。ただし、ドメインの再作成中は、ドメイン名が元のドメイン名と同一でなければなりません。

4 **3** 台目のマシンで *domain-root-dir*/domain1/generated/tmp ディレクトリのアクセス権を変更して、**1** 台目のマシンの同じディレクトリのアクセス権と一致させます。

このディレクトリのデフォルトのアクセス権は、?drwx------? (または 700) です。

次に例を示します。

```
chmod 700 domain-root-dir/domain1/generated/tmp
```

前述の例では、domain1 をバックアップすると仮定しています。ドメインを別の名前でバックアップする場合は、この domain1 をバックアップするドメインの名前に置き換えてください。

5 **3** 台目のマシンの domain.xml で、プロパティーのホスト値を変更します。

6 **3** 台目のマシンの *domain-root-dir*/domain1/config/domain.xml を更新します。

たとえば、machine1 を検索して、machine3 に置き換えるとします。その場合は、次のように変更します。

```
<jmx-connector><property name=client-hostname value=machine1/>...
```

変更後:

```
<jmx-connector><property name=client-hostname value=machine3/>...
```

7 次のように変更します。

```
<jms-service... host=machine1.../>
```

変更後:

```
<jms-service... host=machine3.../>
```

8 **machine3** の復元されたドメインを起動します。

```
asadmin start-domain --user admin-user --password admin-password domain1
```

9 **machine2** のノードエージェントのプロパティーで、**DAS** ホストの値を変更します。

10 **machine2** の *install-dir*/nodeagents/*nodeagent*/agent/config/das.properties で、agent.das.host プロパティー値を変更します。

11 **machine2** のノードエージェントを再起動します。

注 – asadmin start-instance コマンドを使用してクラスタインスタンスを起動し、復元したドメインと同期させます。

Application Server インスタンス

Application Server インスタンスについて

Application Server は、インストール時に server という名前のアプリケーションサーバーインスタンスを作成します。必要に応じて、サーバーインスタンスを削除して、異なる名前で新しいインスタンスを作成できます。

各 Application Server インスタンスには、固有の J2EE 設定、J2EE リソース、アプリケーションの配備領域、およびサーバー設定があります。1 つのアプリケーションサーバーインスタンスを変更しても、ほかのアプリケーションサーバーインスタンスへは影響しません。1 つの管理ドメイン内に多数のアプリケーションサーバーインスタンスを保有できます。

多くのユーザーに対して、1 つのアプリケーションサーバーインスタンスが、そのニーズを満たします。ただし、環境によっては、1 つ以上の追加のアプリケーションサーバーインスタンスを追加して作成する場合があります。たとえば、開発環境内で異なるアプリケーションサーバーインスタンスを使用して、異なる Application Server 設定でテストしたり、異なるアプリケーション配備を比較およびテストできます。アプリケーションサーバーインスタンスを簡単に追加または削除できるので、それを利用して、一時的な「サンドボックス」領域を作成して、開発中に実験できます。

さらに、各アプリケーションサーバーインスタンスに対して、仮想サーバーを作成することもできます。単一のインストールされているアプリケーションサーバーインスタンス内で、企業または個人のドメイン名、IP アドレス、いくつかの管理機能を提供できます。ユーザーにとっては、ハードウェアを持つことも、サーバーの基本的な保守を行うこともなく、自分の Web サーバーを所有しているのとほぼ同じです。このような仮想サーバーは、複数のアプリケーションサーバーインスタンスにまたがりません。仮想サーバーの詳細については、327 ページの「JVM の一般設定を行う」を参照してください。

実践配備においては、複数のアプリケーションサーバーインスタンスの代わりに仮想サーバーをさまざまな用途に応じて使用できます。ただし、仮想サーバーがニーズを満たさない場合、複数のアプリケーションサーバーインスタンスを使用することも可能です。

Application Server インスタンスは、自動的には起動しません。一度インスタンスを起動すると、停止するまで、そのインスタンスは機能します。アプリケーションサーバーインスタンスを停止すると、そのアプリケーションサーバーインスタンスは新しい接続を受

け付けなくなり、未完了の接続がすべて完了するまで待機します。マシンがクラッシュしたり、オフラインになったりすると、サーバーは終了して、処理中だった要求が失われる可能性があります。

Application Server インスタンスの定義

アプリケーションサーバーインスタンスは、アプリケーション開発の基礎を形成します。各インスタンスは 1 つのドメインに属し、それぞれに固有のディレクトリ構造、設定、および配備されたアプリケーションが含まれます。各サーバーインスタンスには、J2EE プラットフォームの Web および EJB コンテナも含まれます。新しいサーバーインスタンスには必ず、インスタンスが置かれるマシンを定義するノードエージェント名に対する参照が含まれる必要があります。

作成可能なサーバーインスタンスには、次の 3 つのタイプがあります。個々のサーバーインスタンスは 1 つのタイプのみに該当します。

- スタンドアロンサーバーインスタンスの場合、設定はほかのサーバーインスタンスやクラスタとは共有されません。
- 共有サーバーインスタンスの場合、設定はほかのインスタンスやクラスタと共有されます。
- クラスタ化されたサーバーインスタンスの場合、設定はクラスタ内のほかのインスタンスと共有されます。

  クラスタは、アプリケーション、リソース、および設定情報の同じセットを共有するサーバーインスタンスの集まりです。1 つのサーバーインスタンスは 1 つのクラスタにのみ属することが可能です。特に注目すべき点は、クラスタを使用すると、複数のマシン間で負荷が分散されることによってロードバランスが容易になり、インスタンスレベルのフェイルオーバーによって高可用性を実現できることです。

図 1–2 は、アプリケーションサーバーインスタンスの詳細を示しています。アプリケーションサーバーインスタンスは、Application Server Enterprise Edition のクラスタリング、ロードバランス、セッションの持続性といった各機能の基本を構成するものです。

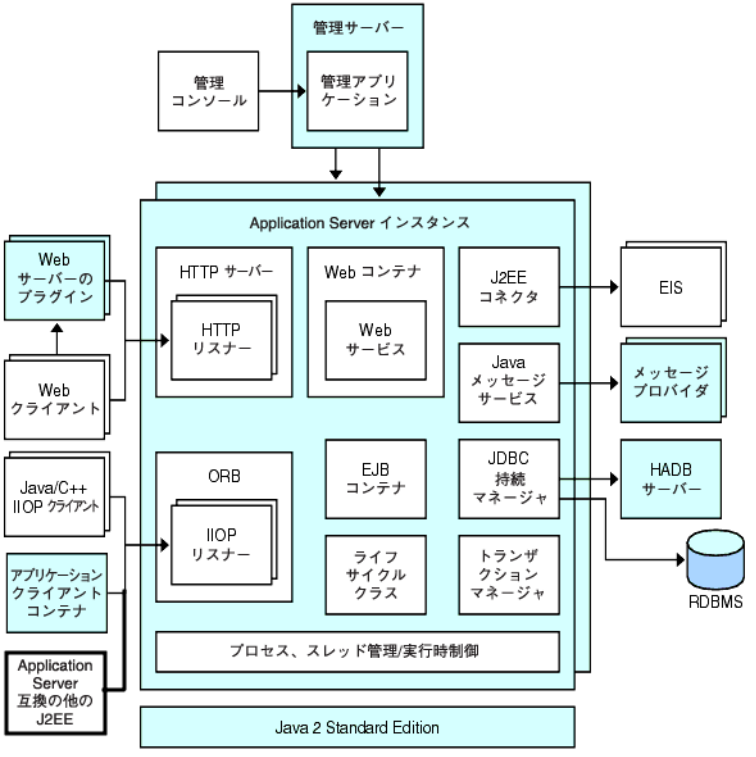

図 1–2 Application Server インスタンス

スタンドアロンインスタンスについて

Application Server インスタンスは、自動的には起動しません。一度インスタンスを起動すると、停止するまで、そのインスタンスは機能します。アプリケーションサーバーインスタンスを停止すると、そのアプリケーションサーバーインスタンスは新しい接続を受け付けなくなり、未完了の接続がすべて完了するまで待機します。マシンがクラッシュしたり、オフラインになったりすると、サーバーは終了して、処理中だった要求が失われる可能性があります。

関連項目

- 50 ページの「インスタンスを作成する」
- 51 ページの「インスタンスを起動する」
- 52 ページの「インスタンスを停止する」
- 52 ページの「トランザクションの回復」

一般サーバー情報の表示

「一般」タブから、次のタスクを実行できます。

- 「インスタンスを起動」をクリックして、インスタンスを起動します。
- 「インスタンスの停止」をクリックして、インスタンスを停止します。
- 「ログファイルを表示」をクリックして、サーバーのログビューアを開きます。
- 「ログファイルをローテーション」をクリックして、インスタンスのログファイルをローテーションします。

  このアクションは、ログファイルのローテーションをスケジュールします。実際のローテーションは、次にログファイルがエントリに書き込まれたときに行われます。ローテーションは、デフォルトのサーバー (DAS) に対してはただちに実行されますが、ほかのスタンドアロンサーバーに対しては遅れます。
- 「JNDI ブラウズ」をクリックして、実行中のインスタンスの JNDI ツリーをブラウズします。
- 「トランザクションの回復」をクリックして、未完了のトランザクションを回復します。

さらに、次のタブを選択して、追加のタスクを実行できます。

- 「アプリケーション」タブ: 選択したアプリケーションを配備します。
- 「リソース」タブ: 選択したリソースを管理します。
- 「プロパティー」タブ: インスタンス固有のプロパティーを設定します。
- 「監視」タブ: JVM、サーバー、スレッドプール、HTTP サービス、トランザクションサービスの監視データを表示します。
- 「詳細」タブ: アプリケーションを配備するための一般的なプロパティーを設定します。

▼ アプリケーションを配備する

「アプリケーション」タブでは、インスタンスと関連付けられたアプリケーションのうち選択したものを有効化または無効化したり、配備したりできます。

1 必要なアプリケーションのチェックボックスを選択します。

2 「配備」ドロップダウンメニューから、配備するアプリケーションモジュールのタイプを選択します。

- エンタープライズアプリケーション: EAR (Enterprise Application Archive) ファイルまたはディレクトリ内の J2EE アプリケーション。
- Web アプリケーション: WAR (Web アプリケーションアーカイブ) ファイルまたはディレクトリ内にパッケージ化されている JavaServer Pages (JSP)、サーブレット、HTML ページなどの Web リソースの集まり。

- EJB モジュール: EJB JAR (Java Archive) ファイルまたはディレクトリに含まれる 1 つまたは複数の Enterprise JavaBeans (EJB コンポーネント)。
- コネクタモジュール: エンタープライズ情報システム (EIS) に接続し、RAR (Resource Adapter Archive) ファイルまたはディレクトリにパッケージ化されます。
- ライフサイクルモジュール: サーバーのライフサイクルの 1 つまたは複数のイベントによって起動されると、タスクを実行します。
- アプリケーションクライアントモジュール: J2EE アプリケーションクライアント JAR ファイルとも呼ばれ、クライアントのサーバー側ルーチンを含んでいます。

▼ 新しいリソースタイプを作成する

「リソース」タブでは、リソースタイプを有効または無効にしたり、新規に作成してインスタンスと関連付けしたりできます。

1 必要なリソースのチェックボックスを選択します。

2 「新規」ドロップダウンメニューから、作成するリソースのタイプを選択し、該当するインスタンスと関連付けます。

- JDBC: アプリケーションにデータベースへ接続する手段を提供します。
- 持続マネージャー: 下位互換性のために必要な、コンテナ管理による持続性 Beans を使用したアプリケーションのために必要です。
- JMS 接続ファクトリ: アプリケーションがプログラムでほかの JMS オブジェクトを作成できるようにするオブジェクトです。
- JMS 送信先: メールおよびメッセージングアプリケーションを構築するための、プラットフォームにもプロトコルにも依存しないフレームワークを提供する JavaMail API 内のメールセッションを表します。
- JavaMail: メールおよびメッセージングアプリケーションを構築するための、プラットフォームにもプロトコルにも依存しないフレームワークを提供します。
- カスタム: 定義済みの JNDI サブコンテキスト、リソースタイプ、およびファクトリクラスを含む、非標準のリソースを表します。
- 外部: LDAP (Lightweight Directory Access Protocol) リポジトリ内の外部リソースオブジェクトを検出するためのアプリケーションを有効にします。
- コネクタ: アプリケーションにエンタープライズ情報システム (EIS) への接続を提供するプログラムオブジェクト。
- 管理オブジェクト: JSR-160 準拠のリモート JMX コネクタを設定します。

管理サーバーの詳細設定

管理サーバーの詳細設定では、アプリケーションを配備するための一般プロパティーを設定できます。これらのプロパティーにより、配備されているアプリケーションに加えられた変更の検出や、変更されたクラスの再読み込みを確実に実行し、監視できます。

アプリケーション設定の実行

動的再読み込みを有効にすると、サーバーは配備されたアプリケーションのファイル内の変更を定期的にチェックし、変更のあるアプリケーションを自動的に再読み込みします。動的再読み込みは、変更したコードをすぐにテストできるため、開発環境で役に立ちます。しかし、本稼働環境では、動的再読み込みはパフォーマンスを低下させる可能性があります。

動的再読み込みは、開発環境を対象としています。この機能は、本稼働環境の機能であるセッションの持続性とは互換性がありません。動的な配備が有効になっている場合は、セッションの持続性を有効にしないでください。

注 – 動的再読み込みは、デフォルトのサーバーインスタンスにのみ利用可能です。

「アプリケーション設定」ページから動的再読み込みを設定するには、次を設定します。

- 再読み込み: 「有効」チェックボックスで動的再読み込みを有効または無効にします。
- 再読込のポーリング間隔: 配備されているアプリケーション内の変更をサーバーがチェックする頻度を指定します。
- 管理セッションタイムアウト: 管理セッションがタイムアウトし、再度ログインするまでの時間を指定します。

▼ 自動配備の設定を行う

自動配備機能を使うと、事前にパッケージ化されたアプリケーションやモジュールを *domain-dir*/`autodeploy` ディレクトリにコピーすることで配備できます。

たとえば、`hello.war` という名前のファイルを *domain-dir*/`autodeploy` ディレクトリにコピーします。アプリケーションの配備を取り消すには、`autodeploy` ディレクトリから `hello.war` ファイルを削除します。

自動配備機能は、開発環境を対象としています。この機能は、本稼働環境の機能であるセッションの持続性とは互換性がありません。自動配備が有効になっている場合は、セッションの持続性を有効にしないでください。

注 – 自動配備は、デフォルトのサーバーインスタンスにのみ利用可能です。

1 「アプリケーション設定」ページに移動します。

2 「有効」チェックボックスを選択または選択解除して、自動配備を有効または無効にします。

3 「自動配備のポーリング間隔」フィールドで、アプリケーションやモジュールファイルの自動配備ディレクトリをサーバーが確認する頻度を指定します。
ポーリング間隔を変更しても、アプリケーションやモジュールの配備にかかる時間には影響ありません。

4 自動配備ディレクトリでアプリケーションを構築したディレクトリを指定してあれば、ファイルをデフォルトの自動配備ディレクトリにコピーする必要はありません。
デフォルトでは、複数のサーバーインスタンスのディレクトリを手動で変更する必要をなくすために、変数が使用されます。

5 配備の前にベリファイアを実行するには、「ベリファイアの有効化」チェックボックスにチェックマークを付けます。
ベリファイアはファイルの構造とコンテンツを調べます。大きなアプリケーションの検証は時間がかかる可能性があります。

6 JSP ページを事前にコンパイルするには、「JSP」チェックボックスにチェックマークを付けます。
このチェックボックスを選択しない場合、JSP ページは最初のアクセスの実行時にコンパイルされます。コンパイルは時間がかかる可能性があるので、本稼働環境ではこのチェックボックスにチェックマークを付けてください。

7 「プロパティーを追加」ボタンをクリックして、追加する設定値を指定します。

ドメイン属性の設定

次のドメイン属性プロパティーが利用可能です。

表 1–1 ドメイン属性値

プロパティー	定義
com.sun.aas.installRoot	アプリケーションサーバーがインストールされているディレクトリ
com.sun.aas.instanceRoot	サーバーインスタンス用のトップレベルディレクトリ

表 1–1 ドメイン属性値　(続き)

プロパティー	定義
com.sun.aas.hostName	ホスト (マシン) の名前
com.sun.aas.javaRoot	J2SE インストールディレクトリ
com.sun.aas.imqLib	Sun Java System Message Queue ソフトウェアのライブラリディレクトリ
com.sun.aas.configName	サーバーインスタンスによって使用されている設定の名前
com.sun.aas.instanceName	サーバーインスタンスの名前。このプロパティーは default-config には利用できませんが、カスタマイズされた設定には使用できます。
com.sun.aas.clusterName	クラスタの名前。このプロパティーは、クラスタ化されたサーバーインスタンスにのみ設定されます。このプロパティーは default-config には利用できませんが、カスタマイズされた設定には使用できます。
com.sun.aas.domainName	ドメインの名前。このプロパティーは default-config には利用できませんが、カスタマイズされた設定には使用できます。

インスタンス固有の設定プロパティー

インスタンス固有の設定プロパティーは、そのインスタンスの値をオーバーライドします。

注 – デフォルト値は、インスタンスにバインドされている設定に定義されます。

▼ デフォルト値に値を戻す

1. オーバーライド値を削除します。
2. 「保存」をクリックします。
   オーバーライド値が設定されていなければ、デフォルト値が使用されます。

▼ インスタンスプロパティーを追加する

◗ 「プロパティーを追加」ボタンをクリックして、追加する設定値を指定します。
リソースを設定するために、次のプロパティー属性名および値のペアを利用できます。

プロパティー	定義
HTTP_LISTENER_PORT	このポートを使用して HTTP 要求を待機します。このプロパティーは、http-listener-1 のポート番号を指定します。有効な値は 1 ～ 65535 です。UNIX では、ポート 1 ～ 1024 で待機するソケットを作成するには、スーパーユーザー権限が必要です。
HTTP_SSL_LISTENER_PORT	このポートを使用して HTTPS 要求を待機します。このプロパティーは、http-listener-2 のポート番号を指定します。有効な値は 1 ～ 65535 です。UNIX では、ポート 1 ～ 1024 で待機するソケットを作成するには、スーパーユーザー権限が必要です。
IIOP_LISTENER_PORT	このプロパティーは、orb-listener-1 が待機する IIOP 接続の ORB リスナーポートを指定します。
IIOP_SSL_LISTENER_PORT	このポートは、セキュアな IIOP 接続用に使用します。
JMX_SYSTEM_CONNECTOR_PORT	このプロパティーは、JMX コネクタが待機するポート番号を指定します。有効な値は 1 ～ 65535 です。UNIX では、ポート 1 ～ 1024 で待機するソケットを作成するには、スーパーユーザー権限が必要です。
IIOP_SSL_MUTUALAUTH_PORT	このプロパティーは、SSL_MUTUALAUTH と呼ばれる IIOP リスナーが待機する IIOP 接続の ORB リスナーポートを指定します。

▼ プロパティーを削除する

1 削除するプロパティーをクリックします。

2 「プロパティーを削除」ボタンをクリックします。

▼ インスタンスを作成する

1 ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを選択します。

2 「スタンドアロンサーバーインスタンス」ページで、「新規」をクリックします。

3 「名前」フィールドで、新しいインスタンスの一意の名前を特定します。

4 「ノードエージェント」を選択します。

ノードエージェントの起動は、ノードエージェントのホストマシン上で `asadmin start-node-agent` コマンドを使用して行い、作成するサーバーインスタンスがそのノードエージェントと関連付けられるようにする必要があります。

5 必要な設定を選択します。

- 既存の設定を参照します。新しい設定は追加されません。
- 既存の設定のコピーを作成します。サーバーインスタンスまたはクラスタを追加すると、新しい設定が追加されます。

6 別の設定からコピーするには、新規インスタンスの作成時に設定値を指定します。

デフォルトでは、default-config 設定からコピーした設定を使用して新しいインスタンスが作成されます。

7 サーバーインスタンスの場合、新しい設定には *instance-name*-`config` という名前が付けられます。

`default-config` 設定はデフォルト設定であり、スタンドアロンサーバーインスタンスを作成するためのテンプレートとして機能します。クラスタ化されていないサーバーインスタンスまたはクラスタは、default-config 設定を参照できません。この設定は、新しい設定を作成するためにコピーできるだけです。デフォルト設定を編集して、コピーした新しい設定が正しく初期設定されているかどうか確認します。

参考 同機能を持つ asadmin コマンド

`create-instance`

▼ インスタンスを起動する

1 ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを展開します。

2 起動したいインスタンスを選択します。

3 「一般」タブで、「インスタンスを起動」をクリックしてインスタンスを起動します。

インスタンスを正常に起動するには、`asadmin start-node-agent` コマンドを使用して、インスタンスと関連付けられているノードエージェントを起動する必要があります。

インスタンスが起動すると、「一般」タブから次のタスクが実行可能になります。

- 「インスタンスの停止」をクリックして、インスタンスを停止します。
- 「**JNDI** ブラウズ」をクリックして、そのインスタンスの **JNDI** エントリを表示します。

- 「ログファイルを表示」をクリックしてログビューアを表示し、ログのオプションを指定します。
- 「ログファイルをローテーション」をクリックします。
- 「トランザクションの回復」をクリックして、未完了のトランザクションを回復します。

参考 同機能を持つ asadmin コマンド

`start-instance`

トランザクションの回復

トランザクションは、サーバークラッシュまたはリソースマネージャークラッシュのいずれかにより未完了になる可能性があります。これらの未完了トランザクションを完了させ、障害を回復させる必要があります。Application Server は、これらの障害を回復し、サーバーの起動時にそのトランザクションを完了するように設計されています。

選択されたサーバーが実行中の場合、回復は同じサーバーによって行われます。選択されたサーバーが実行中でない場合、選択した送信先サーバーが回復を実施します。

▼ インスタンスを停止する

1. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを展開します。
2. 停止したいインスタンスを選択します。
3. 「一般」タブで、「インスタンスの停止」をクリックしてインスタンスを停止します。

参考 同機能を持つ asadmin コマンド

`stop-instance`

▼ 管理サーバーをシャットダウンする

1. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを選択します。
2. 管理サーバーインスタンスを選択します。
3. 「停止」をクリックします。
   管理サーバーをシャットダウンするかどうかを確認するダイアログが表示されます。

設定変更

Application Server 設定の変更

次の設定変更を実行した場合は、変更を有効にするためにサーバーを再起動する必要があります。

- JVM オプションの変更
- ポート番号の変更
- HTTP、IIOP、および JMS サービスの管理
- スレッドプールの管理

手順については、39 ページの「サーバーまたはドメインの再起動」を参照してください。

動的設定を使用すると、ほとんどの変更はサーバーが実行している間に有効になります。次の設定を変更した場合は、サーバーを再起動しないでください。

- アプリケーションの配備と配備取り消し
- JDBC、JMS、Connector のリソース、およびプールの追加または削除
- ログレベルの変更
- ファイルレルムユーザーの追加
- 監視レベルの変更
- リソースとアプリケーションの有効化と無効化

`asadmin reconfig` コマンドは推奨されなくなり、不要になったことに注意してください。設定の変更は、サーバーに対して動的に適用されます。

Application Server のポート

次の表に、Application Server のポートリスナーを示します。

表 1–2 ポートを使用する Application Server リスナー

リスナー (Listener)	デフォルトのポート番号	説明
管理サーバー	4849	ドメインの管理サーバーには、管理コンソールと asadmin ユーティリティーを使ってアクセスします。管理コンソールの場合は、ブラウザの URL にポート番号を指定します。リモートから asadmin コマンドを実行する場合は、--port オプションを使用してポート番号を指定します。
HTTP	8080	Web サーバーはポート上で HTTP 要求を待機します。配備された Web アプリケーションとサービスにアクセスするために、クライアントはこのポートに接続します。
HTTPS	8181	セキュア通信用に設定された Web アプリケーションは、個別のポートで待機します。
IIOP		EJB コンポーネントである Enterprise JavaBeans のリモートクライアントは IIOP リスナー経由で Beans にアクセスします。
IIOP、SSL		セキュア通信用に設定された IIOP リスナーは、ほかのポートを使用します。
IIOP、SSL、および相互認証		相互 (クライアントおよびサーバー) 認証用に設定された IIOP リスナーは、もう一方のポートを使用します。

▼ ポート番号を表示する

1 ツリーコンポーネントで、スタンドアロンインスタンスノードの下にあるインスタンスを選択します。

2 「プロパティー」タブを選択します。

3 「インスタンス固有」ページで、デフォルトのポート番号を確認します。構成を設定して、これらの値をオーバーライドできます。

▼ 管理サーバーポートを変更する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 「**server-config (管理設定)**」ノードを展開します。

3 「**HTTP** サービス」ノードを展開します。

4 「**HTTP** リスナー」ノードを展開します。

5 「**admin-listener**」ノードを選択します。

6 「**HTTP** リスナーを編集」ページで、「リスナーポート」フィールドの値を変更します。

7 サーバーを再起動します。

▼ HTTP ポートを変更する

1 ツリーコンポーネントで「**HTTP** サービス」ノードを展開します。

2 「**HTTP** リスナー」ノードを展開します。

3 変更するポート番号の **HTTP** リスナーを選択します。

4 「**HTTP** リスナーを編集」ページで、「リスナーポート」フィールドの値を変更します。

5 「保存」をクリックします。

6 サーバーを再起動します。

▼ IIOP ポートを変更する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 「**server-config** (管理設定)」ノードを展開します。

3 「**ORB**」ノードを展開します。

4 「**IIOP** リスナー」ノードを展開します。

5 変更するポート番号のリスナーを選択します。

6 「**IIOP** リスナーを編集」ページで、「リスナーポート」フィールドの値を変更します。

7 「保存」をクリックします。

8 サーバーを再起動します。

▼ 管理サービスを使用して JMX コネクタを設定する

管理サービスを使用して、JSR-160 準拠のリモート JMX コネクタを設定してください。これは、ドメイン管理サーバーとノードエージェントとの間の通信を処理し、ノードエージェントは、リモートサーバーインスタンスの代わりに、ホストマシンのサーバーインスタンスを管理します。

管理サービスは、サーバーインスタンスが通常のインスタンスか、ドメイン管理サーバー (DAS) か、あるいは組み合わせかを決定します。DAS は、ユーザーアプリケーション要求を処理する能力はあるものの、ユーザーアプリケーションとリソースが、DAS に対しては配備されないことを除いて、J2EE サーバーインスタンスと似ています。DAS と J2EE サーバーインスタンス間の唯一の大きな違いは、DAS は、サーバーインスタンスの同質ユニットであるクラスタの一部にはならない点です。

1 ツリーから「設定」を選択します。

2 設定するインスタンスを選択します。

   a. 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス **server** の場合は、`server-config` ノードを選択します。

   b. `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 ツリーから「管理サービス」を選択します。

4 「タイプ」ドロップダウンメニューから、管理サービスを設定する **DAS**、**DAS** とサーバー、またはサーバーを選択します。**DAS** とサーバーを選択することは、**DAS** を選択することと同じです。サーバーを選択すると、**DAS** 以外のサーバーインスタンスが選択されます。

5 「**JMX** コネクタ名」フィールドに、内部的に使用する **JMX** コネクタの名前を入力します。コネクタの名前は、**system** です。

▼ JMX コネクタの設定を編集する

「JMX コネクタを編集」画面で、JSR 160 準拠の JMX コネクタの設定を編集できます。

1 ツリーから「設定」を選択します。

2 設定するインスタンスを選択します。

   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス **server** の場合は、`server-config` ノードを選択します。

   - `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 「管理サービス」ノードを展開して、内部的に使用される **JMX** コネクタである「システム」をクリックします。

4. **JMX** コネクタサーバーのポートを入力します。
   JMX サービスの URL は、JSR 160 1.0 仕様によって定義されているプロトコル、ポート、およびアドレスの関数です。

5. この **JMX** コネクタがサポートするプロトコルを入力します。
   Application Server version 8.1 は、`rmi_jrmp` プロトコルのみをサポートします。

6. 「レルム名」フィールドに、特別な管理レルムを表す名前を入力します。
   すべての認証は、このレルムによって処理されます。

7. 「有効」チェックボックスを選択して、**JMX** コネクタで **Transport Layer Security** が使用されるように指示します。

J2SE ソフトウェアの変更

Application Server は Java 2 Standard Edition (J2SE) ソフトウェアに依存します。Application Server をインストールすると、J2SE ソフトウェアのディレクトリが指定されます。J2SE ソフトウェアの変更手順については、327 ページの「JVM の一般設定を行う」を参照してください。

オンラインヘルプの利用

管理コンソールのオンラインヘルプには、操作内容に関連した情報が表示されます。右上の「ヘルプ」リンクをクリックすると、ヘルプブラウザウィンドウに、現在の「管理コンソール」ページに関連したトピックが表示されます。現在のページにヘルプ情報がない場合は、「オンラインヘルプの使用」トピックが表示されます。

オンラインヘルプには、文脈依存ではない概念トピックが含まれています。これらのトピックの 1 つを表示するには、ヘルプブラウザウィンドウの目次から選択します。

▼ 前のヘルプ画面に戻る

1. ヘルプブラウザウィンドウ内で、マウスの右ボタンをクリックして選択メニューを表示します。

2. 「戻る」を選択します。

詳細情報

- Sun Microsystems の世界規模でのトレーニング - Sun とその公認トレーニングセンターでは、Web ベースのコースおよび 60 か国以上に配置された 250 箇所を超えるトレーニングサイトを通じて、毎年 250,000 人を超える受講生がトレーニングを受けています。詳細については、次の Web サイトを参照してください。`http://training.sun.com/`
- 『J2EE 1.4 Tutorial』- 開発者を対象に書かれたこのチュートリアルには、JMS の設定、JavaMail リソースの設定、およびセキュリティー管理のための管理手順が説明されています。チュートリアルにアクセスするには、次の URL に移動してください。`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`
- 『Application Server 開発者ガイド』- このガイドには Application Server に固有の開発情報が記載されています。
- `asadmin` マニュアルページ - HTML 形式で入手できるこれらのページには、`asadmin` ユーティリティーコマンドを含むすべてのアプリケーションサーバーユーティリティーの構文と例が記載されています。
- 『*Application Server* リリースノート』
- docs.sun.com: Sun 製品マニュアル - 次のサイトから当社のすべての製品マニュアルを検索し、アクセスできます。`http://docs.sun.com/`
- J2EE 1.4 Documentation ページ - 公開 Web サイト上のこのページには、J2EE 1.4 プラットフォームのテクニカルマニュアルへのリンクがあります。`http://java.sun.com/j2ee/1.4/docs/`
- 『クイックスタートガイド』- このマニュアルには、簡単な Web アプリケーションの配備と実行の方法が説明されています。このガイドは *install-dir*`/docs/QuickStart.html` ファイルにあります。

◆ ◆ ◆ 第 2 章

2 アプリケーションの配備

この章では、Application Server に J2EE アプリケーションを配備 (インストール) する方法について説明します。この章には次の節が含まれています。

配備について

配備のライフサイクル

Application Server をインストールしてドメインを起動したら、J2EE アプリケーションとモジュールを配備 (インストール) できます。配備中およびアプリケーションの変更の際、アプリケーションとモジュールには次のような作業を行います。

1. 初期の配備

   アプリケーションまたはモジュールを配備する前に、ドメインを起動します。

   アプリケーションまたはモジュールを、特定のスタンドアロンサーバーインスタンスまたはクラスタに配備 (インストール) します。アプリケーションとモジュールはアーカイブファイルにパッケージ化されているので、配備中はアーカイブファイル名を指定します。デフォルトでは、デフォルトのサーバーインスタンス `server` に配備されます。

サーバーインスタンスまたはクラスタに配備する場合、アプリケーションやモジュールはドメインの中央リポジトリ内に存在し、ターゲットとして配備したクラスタまたはサーバーインスタンスによって参照されます。

管理コンソールではなく `asadmin deploy` コマンドを使用して、ドメインに配備することもできます。アプリケーションやモジュールをドメインだけに配備する場合、アプリケーションやモジュールはドメインの中央リポジトリ内に存在しますが、59 ページの「配備のライフサイクル」の説明に従って参照を追加するまでは、サーバーインスタンスまたはクラスタによって参照されません。

配備は動的です。アプリケーションを使用可能にするために、アプリケーションまたはモジュールの配備後にサーバーインスタンスを再起動する必要はありません。再起動しても、すべての配備アプリケーションとモジュールはそのまま配備され、使用することができます。

2. 有効化または無効化

   デフォルトでは、配備されているアプリケーションやモジュールは有効になっています。つまり、アクセス可能なサーバーインスタンスやクラスタに配備されている場合、実行可能で、クライアントがアクセスできる状態になっています。アクセスを抑制するには、アプリケーションやモジュールを無効化します。無効化されたアプリケーションやモジュールはドメインからアンインストールされてはいないので、配備後は簡単に有効化できます。

3. 配備されているアプリケーションやモジュールのターゲットの追加または削除

   配備の完了したアプリケーションやモジュールは、中央リポジトリ内に存在し、複数のサーバーインスタンスやクラスタによる参照が可能になります。最初は、ターゲットとして配備したサーバーインスタンスやクラスタがアプリケーションやモジュールを参照します。

   アプリケーションやモジュールの配備後、それを参照するサーバーインスタンスやクラスタを変更するには、管理コンソールを使用してアプリケーションやモジュールのターゲットを変更するか、または `asadmin` ツールを使用してアプリケーションの参照を変更します。アプリケーション自体は中央リポジトリに格納されるため、ターゲットを追加または削除すると、さまざまなターゲット上にある同じバージョンのアプリケーションが追加または削除されます。ただし、複数のターゲットに配備されているアプリケーションを、1 つのターゲット上で有効にして、ほかのターゲット上で無効にすることができます。したがって、アプリケーションがあるターゲットによって参照されていても、そのターゲット上で有効にされないかぎり、ユーザーはアプリケーションを使用できません。

4. 再配備

   配備されているアプリケーションやモジュールを置換するには、これらを再配備します。再配備すると、以前に配備されたアプリケーションやモジュールは配備が自動的に取り消され、新しいアプリケーションやモジュールと置き換えられます。

   管理コンソールから再配備した場合、再配備されたアプリケーションやモジュールはドメインに配備されます。動的再設定が有効になっている場合、アプリケーションやモジュールを参照するスタンドアロンまたはクラスタ化されたすべてのサーバーインスタンスは、自動的に新しいバージョンを受信します。`asadmin deploy` コマンドを使用して再配備する場合、ターゲットとして `domain` を指定します。

本稼動環境では、段階的アップグレードを使用して、処理が中断されない状態でアプリケーションをアップグレードします。詳細については、「段階的アップグレードについて」を参照してください。

5. 配備取消し

   アプリケーションまたはモジュールをアンインストールするには、これらの配備を取り消します。

J2EE アーカイブファイルのタイプ

ソフトウェアプロバイダは、アプリケーションやモジュールをアーカイブファイルにパッケージ化します。アプリケーションやモジュールを配備するには、アーカイブファイルの名前を指定します。アーカイブファイルのコンテンツと構造は J2EE プラットフォームの仕様で定義されています。J2EE アーカイブファイルの種類は次のとおりです。

- Web アプリケーションアーカイブ (WAR): WAR ファイルは、サーブレットや JSP などの Web コンポーネントと、静的な HTML ページ、JAR ファイル、タグライブラリ、およびユーティリティークラスで構成されます。WAR ファイル名の拡張子は `.war` です。
- EJB JAR: EJB JAR ファイルには、EJB テクノロジが使用するコンポーネントである 1 つまたは複数の Enterprise JavaBeans が含まれます。EJB JAR ファイルには、Enterprise JavaBeans で必要なユーティリティークラスも含まれます。EJB JAR ファイル名の拡張子は `.jar` です。
- J2EE アプリケーションクライアント JAR: この JAR ファイルには、RMI/IIOP を使用して Enterprise JavaBeans などのサーバー側コンポーネントにアクセスする J2EE アプリケーションクライアントのコードが含まれます。管理コンソールでは、J2EE アプリケーションクライアントを「アプリケーションクライアント」と呼びます。J2EE アプリケーションクライアントである JAR ファイル名の拡張子は `.jar` です。
- リソースアダプタアーカイブ (RAR): RAR ファイルはリソースアダプタを保持します。リソースアダプタは、J2EE Connector Architecture 仕様によって定義されており、Enterprise JavaBeans、Web コンポーネント、およびアプリケーションクライアントがリソースや外部エンタープライズシステムにアクセスできるようにする、移行可能なコンポーネントです。通常、リソースアダプタはコネクタと呼ばれます。RAR ファイル名の拡張子は `.rar` です。
- エンタープライズアプリケーションアーカイブ (EAR): EAR ファイルは 1 つまたは複数の WAR、EJB JAR、RAR、または J2EE アプリケーションクライアント JAR ファイルを保持します。EAR ファイル名の拡張子は `.ear` です。

ソフトウェアプロバイダは、アプリケーションを 1 つの EAR ファイルまたは個別の WAR、EJB JAR、およびアプリケーションクライアント JAR ファイルにアセンブルすることが可能です。管理ツールでは、配備ページとコマンドはすべての種類のファイルで同じです。

ネーミング規則

1 つのドメイン内では、配備されているアプリケーションやモジュールの名前が一意である必要があります。

- 管理コンソールを使用して配備する場合は、「アプリケーション名」フィールドで名前を指定します。
- `asadmin deploy` コマンドを使用して配備する場合は、アプリケーションやモジュールのデフォルト名は、配備される JAR ファイルのプレフィックスになります。たとえば、`hello.war` ファイルの場合、Web アプリケーション名は `hello` となります。デフォルト名をオーバーライドするには、`--name` オプションを指定します。

異なるタイプのモジュールが、アプリケーション内で同じ名前を使用できます。アプリケーションが配備されると、個々のモジュールを保持するディレクトリの名前には `_jar`、`_war`、および `_rar` サフィックスが使用されます。アプリケーション内の同じタイプのモジュールは、一意の名前にする必要があります。また、データスキーマのファイル名は、アプリケーション内で一意の名前にする必要があります。

モジュールのファイル名、EAR ファイル名、`ejb-jar.xml` ファイルの `<module-name>` 部分に見られるモジュール名、および `ejb-jar.xml` ファイルの `<ejb-name>` 部分に見られる EJB 名には、Java パッケージと同様のネーミングスキームを使用することをお勧めします。このパッケージと同様のネーミングスキームの使用により、名前の競合を防げます。このネーミング方法の利点は、Application Server だけでなく、ほかの J2EE Application Server にも当てはまります。

EJB コンポーネントの JNDI 検索名も一意である必要があります。一貫性のあるネーミング規則の確立が役立つ場合があります。たとえば、アプリケーション名とモジュール名を EJB 名に追加するのは、名前を一意にする 1 つの方法です。この場合、`mycompany.pkging.pkgingEJB.MyEJB` は、アプリケーション `pkging.ear` にパッケージ化されたモジュール `pkgingEJB.jar` 内にある EJB の JNDI 名を表します。

パッケージとファイル名に、オペレーティングシステムでは不正なスペースや文字を含めないようにする必要があります。

アプリケーションの配備に関する管理コンソールタスク

▼ エンタープライズアプリケーションを配備する

エンタープライズアプリケーションは、WAR ファイルや EJB JAR ファイルなど、任意のタイプの J2EE スタンドアロンモジュールを含むアーカイブファイルの一種である EAR ファイルにパッケージ化されています。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「エンタープライズアプリケーション」ノードを選択します。

3 「エンタープライズアプリケーション」ページで、「配備」をクリックします。

4 「配備」ページで、**EAR** ファイルを配備する場所を指定します。

サーバーマシンとは、Application Server のドメイン管理サーバーを実行しているホストです。クライアントマシンとは、ブラウザを介して管理コンソールを表示しているホストです。

- ファイルがクライアントマシンにある場合、またはクライアントマシンからファイルにアクセス可能な場合は、ラジオボタンをクリックして、**Application Server** にアップロードするパッケージファイルを指定します。

  「ブラウズ」をクリックしてファイルを検索するか、またはファイルへの完全パスを入力します。

- ファイルがサーバーマシンにある場合、または分割ディレクトリからパッケージ化されていないアプリケーションを配備する場合は、ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

  ファイルまたはディレクトリへの完全パス名を入力します。分割ディレクトリからの配備は高度な開発者用なので、本稼働環境ではお勧めできません。

5 「次へ」をクリックして「**Enterprise** アプリケーションを配備」ページを表示します。

6 「**Enterprise** アプリケーションを配備」ページで、アプリケーションの設定値を指定します。

a. 「アプリケーション名」フィールドで、ファイル名のプレフィックスであるデフォルト名を保持するか、または別の名前を入力します。

ファイルのアップロードを選択した場合は、デフォルト名が表示されます。アプリケーション名は一意である必要があります。

b. 配備後には利用できないようにアプリケーションを無効にする場合は、「無効」ラジオボタンをオンにします。

デフォルトでは、アプリケーションは配備すると同時に利用可能になります。

c. アプリケーションがすでに配備されている場合は、「再配備」チェックボックスを選択して、再配備します。そうでない場合、エラーが表示されます。

また、別のアプリケーション名を選択して、新しい名前で配備することもできます。

d. 配備の前にファイルの構造やコンテンツを検証するには、「ベリファイア」チェックボックスにチェックマークを付けます。

大きなアプリケーションの検証は時間がかかる可能性があります。ファイルの破壊や移行不能が想定される場合は検証を行なってください。

e. **JSP** ページを事前にコンパイルするには、「**JSP**」チェックボックスにチェックマークを付けます。

このチェックボックスを選択しない場合、JSP ページは最初のアクセスの実行時にコンパイルされます。コンパイルは時間がかかる可能性があるので、本稼働環境ではこのチェックボックスにチェックマークを付けてください。

f. 高可用性の設定を選択します。

アプリケーションの高可用性を有効にするには、「可用性」チェックボックスを選択します。1 つのアプリケーションで可用性を有効にする場合、それより高いすべてのレベル (指定した設定および Web コンテナまたは EJB コンテナ) も同様に有効にする必要があります。

g. アプリケーションを配備するターゲットを選択します。

利用可能なターゲットのリストから、1 つまたは複数のターゲットを選択して「追加」をクリックします。ターゲットを選択しない場合、アプリケーションはデフォルトのサーバーインスタンス server に配備されます。

再配備の場合は、ターゲットを選択しないでください。ここで選択した内容は、すべて無視されます。クラスタやスタンドアロンインスタンスの動的再設定が有効になっている場合、配備されているアプリケーションを参照する、ターゲットのクラスタ化またはスタンドアロンのすべてのサーバーインスタンスは、新しく再配備されたアプリケーションを自動的に参照します。サービスを中断せずにアプリケーションを再配備する方法の詳細については、「アプリケーションのアップグレード」を参照してください。

h. **RMI** スタブを生成するかどうかを選択します。

RMI スタブの生成を選択すると、静的 RMI-IIOP スタブが生成され、クライアント JAR ファイルに配置されます。

7 「了解」をクリックしてアプリケーションを配備します。

参考 同機能を持つ asadmin コマンド

`deploy`

▼ 配備されているエンタープライズアプリケーションを編集する

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「エンタープライズアプリケーション」ノードを展開します。

3 配備されているアプリケーションのノードを選択します。

4 「エンタープライズアプリケーション」ページで、説明を変更します。

5 「**Enterprise Edition**」で、高可用性を有効または無効にします。

1 つのアプリケーションで可用性を有効にする場合、それより高いすべてのレベル (指定した設定および Web コンテナまたは EJB コンテナ) も同様に有効にする必要があります。

▼ Web アプリケーションを配備する

Web アプリケーションは、サーブレットや JSP ファイルなどのコンポーネントを含むアーカイブファイルの一種である WAR ファイルにパッケージ化されています。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「**Web** アプリケーション」ノードを選択します。

3 「**Web** アプリケーション」ページで、「配備」をクリックします。

4 「配備」ページで、**WAR** ファイルを配備する場所を指定します。

サーバーマシンとは、Application Server のドメイン管理サーバーを実行しているホストです。クライアントマシンとは、ブラウザを介して管理コンソールを表示しているホストです。

- ファイルがクライアントマシンにある場合、またはクライアントマシンからファイルにアクセス可能な場合は、ラジオボタンをクリックして、Application Server にアップロードするパッケージファイルを指定します。

  「ブラウズ」をクリックしてファイルを検索するか、またはファイルへの完全パスを入力します。

- ファイルがサーバーマシンにある場合、または分割ディレクトリからパッケージ化されていないアプリケーションを配備する場合は、ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

  ファイルまたはディレクトリへの完全パス名を入力します。分割ディレクトリからの配備は高度な開発者用なので、本稼働環境ではお勧めできません。

5 「次へ」をクリックして「**Web** アプリケーションを配備」ページを表示します。

6 「**Web** アプリケーションを配備」ページで、アプリケーションの設定を指定します。

a. 「アプリケーション名」フィールドで、ファイル名のプレフィックスであるデフォルト名を保持するか、または別の名前を入力します。

ファイルのアップロードを選択した場合は、デフォルト名が表示されます。アプリケーション名は一意である必要があります。

b. 「コンテキストルート」フィールドに、**Web** アプリケーションを識別する文字列を入力します。

Web アプリケーションの URL では、コンテキストルートはポート番号の直後に続きます (`http://host:port/context-root/...`)。コンテキストルートがスラッシュで始まるようにしてください。たとえば次のようにします。`/hello`

c. 配備後には利用できないようにアプリケーションを無効にする場合は、「無効」ラジオボタンをオンにします。

デフォルトでは、アプリケーションは配備すると同時に利用可能になります。

d. アプリケーションがすでに配備されている場合は、「再配備」チェックボックスを選択して、再配備します。そうでない場合、エラーが表示されます。

また、別のアプリケーション名を選択して、新しい名前で配備することもできます。

e. 配備の前にファイルの構造やコンテンツを検証するには、「ベリファイア」チェックボックスにチェックマークを付けます。

大きなアプリケーションの検証は時間がかかる可能性があります。ファイルの破壊や移行不能が想定される場合は検証を行なってください。

f. **JSP** ページを事前にコンパイルするには、「**JSP**」チェックボックスにチェックマークを付けます。

このチェックボックスを選択しない場合、JSP ページは最初のアクセスの実行時にコンパイルされます。コンパイルは時間がかかる可能性があるので、本稼働環境ではこのチェックボックスにチェックマークを付けてください。

g. 高可用性の設定を選択します。

アプリケーションの高可用性を有効にするには、「可用性」チェックボックスを選択します。1 つのアプリケーションで可用性を有効にする場合、それより高いすべてのレベル (指定した設定および Web コンテナまたは EJB コンテナ) も同様に有効にする必要があります。

h. アプリケーションを配備するターゲットを選択します。

利用可能なターゲットのリストから、1 つまたは複数のターゲットを選択して「追加」をクリックします。ターゲットを選択しない場合、アプリケーションはデフォルトのサーバーインスタンス `server` に配備されます。

再配備の場合は、ターゲットを選択しないでください。ここで選択した内容は、すべて無視されます。クラスタやスタンドアロンインスタンスの動的再設定が有効になっている場合、配備されているアプリケーションを参照する、ターゲットのクラスタ化またはスタンドアロンのすべてのサーバーインスタンスは、新しく再配備されたアプリケーションを自動的に参照します。サービスを中断せずにアプリケーションを再配備する方法の詳細については、「段階的アップグレードについて」を参照してください。

i. **RMI** スタブを生成するかどうかを選択します。
RMI スタブの生成を選択すると、静的 RMI-IIOP スタブが生成され、クライアント JAR ファイルに配置されます。

7 「了解」をクリックしてアプリケーションを配備します。

参考 同機能を持つ asadmin コマンド

`deploy`

▼ 配備されている **Web** アプリケーションを起動する

アプリケーションを配備すると、管理コンソールから起動することができます。アプリケーションを起動するには、サーバーと HTTP リスナーが実行されている必要があります。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「**Web** アプリケーション」をクリックします。

3 **Web** アプリケーションの起動リンクをクリックします。

4 「**Web** アプリケーションへのリンク」ページのリンクをクリックして、アプリケーションを起動します。

▼ **EJB** モジュールを配備する

EJB モジュールは、EJB JAR ファイルとも呼ばれ、Enterprise JavaBeans を含んでいます。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「**EJB** モジュール」ノードを選択します。

3 「**EJB** モジュール」ページで、「配備」をクリックします。

4 「配備」ページで、**JAR** ファイルを配備する場所を指定します。

サーバーマシンとは、Application Server のドメイン管理サーバーを実行しているホストです。クライアントマシンとは、ブラウザを介して管理コンソールを表示しているホストです。

- ファイルがクライアントマシンにある場合、またはクライアントマシンからファイルにアクセス可能な場合は、ラジオボタンをクリックして、Application Server にアップロードするパッケージファイルを指定します。

  「ブラウズ」をクリックしてファイルを検索するか、またはファイルへの完全パスを入力します。
- ファイルがサーバーマシンにある場合、または分割ディレクトリからパッケージ化されていないアプリケーションを配備する場合は、ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

  ファイルまたはディレクトリへの完全パス名を入力します。分割ディレクトリからの配備は高度な開発者用なので、本稼働環境ではお勧めできません。

5 「次へ」をクリックして「**EJB** モジュールを配備」ページを表示します。

6 「**EJB** モジュールを配備」ページで、モジュールの設定を指定します。

a. 「アプリケーション名」フィールドで、ファイル名のプレフィックスであるデフォルト名を保持するか、または別の名前を入力します。

ファイルのアップロードを選択した場合は、デフォルト名が表示されます。アプリケーション名は一意である必要があります。

b. 配備後には利用できないようにモジュールを無効にする場合は、「無効」ラジオボタンをオンにします。

デフォルトでは、モジュールは配備すると同時に利用可能になります。

c. モジュールがすでに配備されている場合は、「再配備」チェックボックスを選択して、再配備します。そうでない場合は、エラーが表示されます。

また、別のアプリケーション名を選択して、新しい名前で配備することもできます。

d. 配備の前にファイルの構造やコンテンツを検証するには、「ベリファイア」チェックボックスにチェックマークを付けます。

大きなアプリケーションの検証は時間がかかる可能性があります。ファイルの破壊や移行不能が想定される場合は検証を行なってください。

e. 高可用性の設定を選択します。

モジュールの高可用性を有効にするには、「可用性」チェックボックスを選択します。1 つのモジュールで可用性を有効にする場合、それより高いすべてのレベル (指定した設定および Web コンテナまたは EJB コンテナ) も同様に有効にする必要があります。

f. モジュールを配備するターゲットを選択します。

利用可能なターゲットのリストから、1 つまたは複数のターゲットを選択して「追加」をクリックします。ターゲットを選択しない場合、モジュールはデフォルトのサーバーインスタンス server に配備されます。

再配備の場合は、ターゲットを選択しないでください。ここで選択した内容は、すべて無視されます。クラスタやスタンドアロンインスタンスの動的再設定が有効になっている場合、配備されているモジュールを参照する、ターゲットのクラスタ化またはスタンドアロンのすべてのサーバーインスタンスは、新しく再配備されたモジュールを自動的に参照します。サービスを中断せずにモジュールを再配備する方法の詳細については、「段階的アップグレードについて」を参照してください。

g. **RMI** スタブを生成するかどうかを選択します。

RMI スタブの生成を選択すると、静的 RMI-IIOP スタブが生成され、クライアント JAR ファイルに配置されます。

7 「了解」をクリックしてモジュールを配備します。

参考 同機能を持つ asadmin コマンド

`deploy`

▼ コネクタモジュールを配備する

コネクタはリソースアダプタとも呼ばれ、RAR ファイルというアーカイブファイルの一種にパッケージ化されています。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「コネクタモジュール」ノードを選択します。

3 「コネクタモジュール」ページで、「配備」をクリックします。

4 「配備」ページで、**RAR** ファイルを配備する場所を指定します。

サーバーマシンとは、Application Server のドメイン管理サーバーを実行しているホストです。クライアントマシンとは、ブラウザを介して管理コンソールを表示しているホストです。

- ファイルがクライアントマシンにある場合、またはクライアントマシンからファイルにアクセス可能な場合は、ラジオボタンをクリックして、Application Server にアップロードするパッケージファイルを指定します。

  「ブラウズ」をクリックしてファイルを検索するか、またはファイルへの完全パスを入力します。

- ファイルがサーバーマシンにある場合、またはパッケージ化されていないモジュールを分割ディレクトリから配備する場合は、ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

  ファイルまたはディレクトリへの完全パス名を入力します。分割ディレクトリからの配備は高度な開発者用なので、本稼働環境ではお勧めできません。

5 「次へ」をクリックして「コネクタモジュールを配備」ページを表示します。

6 「コネクタモジュールを配備」ページで、モジュールの設定を指定します。

a. 「アプリケーション名」フィールドで、ファイル名のプレフィックスであるデフォルト名を保持するか、または別の名前を入力します。

ファイルのアップロードを選択した場合は、デフォルト名が表示されます。アプリケーション名は一意である必要があります。

b. 「スレッドプール ID」フィールドで、配備するリソースアダプタのスレッドプールを指定します。

デフォルトでは、Application Server は、デフォルトスレッドプールのすべてのリソースアダプタからの作業要求を処理します。このフィールドを使用して、特定のユーザーが作成したスレッドプールを関連付け、リソースアダプタからの作業要求を処理します。

c. 配備後には利用できないようにモジュールを無効にする場合は、「無効」ラジオボタンをオンにします。

デフォルトでは、モジュールは配備すると同時に利用可能になります。

コネクタモジュールを有効または無効にすると、コネクタリソースとそのモジュールをポイントする接続プールも有効または無効にできます。

d. モジュールがすでに配備されている場合は、「再配備」チェックボックスを選択して、再配備します。そうでない場合は、エラーが表示されます。

また、別のアプリケーション名を選択して、新しい名前で配備することもできます。

e. 配備の前にファイルの構造やコンテンツを検証するには、「ベリファイア」チェックボックスにチェックマークを付けます。

大きなアプリケーションの検証は時間がかかる可能性があります。ファイルの破壊や移行不能が想定される場合は検証を行なってください。

f. リソースアダプタに追加プロパティーが指定されている場合は、それらが表示されます。

この表を使用して、これらのプロパティーのデフォルト値を変更します。

g. モジュールを配備するターゲットを選択します。

利用可能なターゲットのリストから、1 つまたは複数のターゲットを選択して「追加」をクリックします。ターゲットを選択しない場合、モジュールはデフォルトのサーバーインスタンス server に配備されます。

再配備の場合は、ターゲットを選択しないでください。ここで選択した内容は、すべて無視されます。クラスタやスタンドアロンインスタンスの動的再設定が有効になっている場合、配備されているモジュールを参照する、ターゲットのクラスタ化またはスタンドアロンのすべてのサーバーインスタンスは、新しく再配備されたモジュールを自動的に参照します。サービスを中断せずにモジュールを再配備する方法の詳細については、「段階的アップグレードについて」を参照してください。

7 「了解」をクリックしてモジュールを配備します。

参考 同機能を持つ asadmin コマンド

`deploy`

▼ ライフサイクルモジュールを作成する

ライフサイクルモジュールは、サーバーライフサイクルの 1 つまたは複数のイベントによってトリガーされると、タスクを実行します。このようなサーバーイベントには次のものがあります。

- 初期化
- 起動
- サービス要求に対する準備
- シャットダウン

ライフサイクルモジュールは J2EE 仕様の一部ではなく、Application Server の拡張です。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「ライフサイクルモジュール」ノードを選択します。

3 「ライフサイクルモジュール」ページで、「新規」をクリックします。

4 「ライフサイクルモジュールを作成」ページで、次の設定を指定します。

a. 「名前」フィールドに、モジュールの機能を示す名前を入力します。

b. 「クラス名」フィールドに、ライフサイクルモジュールのクラスファイルの完全修飾名を入力します。

c. ライフサイクルを含む **JAR** ファイルがサーバーのクラスパスにある場合は、「クラスパス」フィールドを空白のままにしておきます。そうでない場合は、完全修飾パスを入力します。

クラスパスを指定しない場合は、
domain-dir/applications/lifecycle-module/*module-name* のクラスをアンパックする必要があります。クラスパスを指定する場合は、何もする必要はありません。

d. 「読み込み順序」フィールドに、**100** 以上でオペレーティングシステムの MAXINT 値未満の整数を入力します。

この整数によって、サーバーの起動時にライフサイクルモジュールが読み込まれる順番が決定します。モジュールに指定された数値が小さいほど、早く読み込まれます。

e. サーバーを起動すると、すでに配備されたライフサイクルモジュールが読み込まれます。

デフォルトでは、読み込みが失敗した場合も、サーバーは起動操作を継続します。読み込みが失敗したときにサーバーが起動しないようにするには、「読込み時の障害」チェックボックスにチェックマークを付けます。

f. 配備後には利用できないようにモジュールを無効にする場合は、「無効」ラジオボタンをオンにします。

ライフサイクルモジュールはサーバーの起動時に呼び出されるため、ライフサイクルモジュールを無効にしても、実際にはサーバーインスタンスが再起動されるまで無効になりません。

g. モジュールを配備するターゲットを選択します。

利用可能なターゲットのリストから、1 つまたは複数のターゲットを選択して「追加」をクリックします。ターゲットを選択しない場合、モジュールはデフォルトのサーバーインスタンス server に配備されます。

5 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

create-lifecycle-module

▼ アプリケーションクライアントモジュールを配備する

アプリケーションクライアントモジュールは、J2EE アプリケーションクライアント JAR ファイルとも呼ばれ、クライアントのサーバー側ルーチンを含んでいます。

1 ツリーコンポーネントで、「アプリケーション」ノードを展開します。

2 「アプリケーションクライアントモジュール」ノードを選択します。

3 「アプリケーションクライアントモジュール」ページで、「配備」をクリックします。

4 「配備」ページで、**JAR** ファイルを配備する場所を指定します。

サーバーマシンとは、Application Server のドメイン管理サーバーを実行しているホストです。クライアントマシンとは、ブラウザを介して管理コンソールを表示しているホストです。

- ファイルがクライアントマシンにある場合、またはクライアントマシンからファイルにアクセス可能な場合は、ラジオボタンをクリックして、Application Server にアップロードするパッケージファイルを指定します。

  「ブラウズ」をクリックしてファイルを検索するか、またはファイルへの完全パスを入力します。

- ファイルがサーバーマシンにある場合、またはパッケージ化されていないモジュールを分割ディレクトリから配備する場合は、ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

  ファイルまたはディレクトリへの完全パス名を入力します。分割ディレクトリからの配備は高度な開発者用なので、本稼働環境ではお勧めできません。

5 「次へ」をクリックして「アプリケーションクライアントモジュールを配備」ページを表示します。

6 「アプリケーションクライアントモジュールを配備」ページで、モジュールの設定を指定します。

a. 「アプリケーション名」フィールドで、ファイル名のプレフィックスであるデフォルト名を保持するか、または別の名前を入力します。

   ファイルのアップロードを選択した場合は、デフォルト名が表示されます。アプリケーション名は一意である必要があります。

b. モジュールがすでに配備されている場合は、「再配備」チェックボックスを選択して、再配備します。そうでない場合は、エラーが表示されます。

   また、別のアプリケーション名を選択して、新しい名前で配備することもできます。

c. 配備の前にファイルの構造やコンテンツを検証するには、「ベリファイア」チェックボックスにチェックマークを付けます。

   大きなアプリケーションの検証は時間がかかる可能性があります。ファイルの破壊や移行不能が想定される場合は検証を行なってください。

d. モジュールを配備するターゲットを選択します。

   利用可能なターゲットのリストから、1 つまたは複数のターゲットを選択して「追加」をクリックします。ターゲットを選択しない場合、モジュールはデフォルトのサーバーインスタンス server に配備されます。

再配備の場合は、ターゲットを選択しないでください。ここで選択した内容は、すべて無視されます。クラスタやスタンドアロンインスタンスの動的再設定が有効になっている場合、配備されているモジュールを参照する、ターゲットのクラスタ化またはスタンドアロンのすべてのサーバーインスタンスは、新しく再配備されたモジュールを自動的に参照します。サービスを中断せずにモジュールを再配備する方法の詳細については、「段階的アップグレードについて」を参照してください。

e. RMI スタブを生成するかどうかを選択します。

RMI スタブの生成を選択すると、静的 RMI-IIOP スタブが生成され、クライアント JAR ファイルに配置されます。

クライアント側ルーチンでは次のことを行います。

- 通常、アプリケーションプロバイダは、クライアント側ルーチンを含む JAR ファイルを出荷します。
- アプリケーションプロバイダは、`asadmin deploy` コマンドの `--retrieve` オプションを指定することにより、クライアント側スタブを取得できます。

7 「了解」をクリックしてモジュールを配備します。

参考 同機能を持つ asadmin コマンド

`deploy`

▼ 配備用のアーカイブファイルを指定する

アプリケーションまたはモジュールのページで「配備」をクリックして、「配備」ページにアクセスします。「配備」ページで、アプリケーションまたはモジュールがパッケージ化されているアーカイブファイルの場所を指定します。

サーバーマシンとは、Application Server のドメイン管理サーバーを実行しているホストです。クライアントマシンとは、ブラウザを介して管理コンソールを表示しているホストです。

1 ファイルがクライアントマシンにある場合、またはクライアントマシンからファイルにアクセス可能な場合は、ラジオボタンをクリックして、**Application Server** にアップロードするパッケージファイルを指定します。

「ブラウズ」をクリックしてファイルを検索するか、またはファイルへの完全パスを入力します。

2 ファイルがサーバーマシンにある場合、または分割ディレクトリからパッケージ化されていないアプリケーションを配備する場合は、ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

ファイルまたはディレクトリへの完全パス名を入力します。分割ディレクトリからの配備は高度な開発者用なので、本稼働環境ではお勧めできません。

アプリケーションの一覧表示、配備取消し、および有効化に関する管理コンソールタスク

▼ 配備されているアプリケーションを一覧表示する

1. ツリーコンポーネントで、「アプリケーション」ノードを展開します。
2. アプリケーションまたはモジュールタイプのノードを展開します。
   配備されているアプリケーションまたはモジュールの詳細を表示するには、次の手順のいずれかに従います。
   - ツリーコンポーネントで、アプリケーションまたはモジュールのノードを選択します。
   - ページで、「アプリケーション名」列のエントリを選択します。

参考 同機能を持つ asadmin コマンド

`list-components`

▼ サブコンポーネントを一覧表示する

エンタープライズアプリケーション、Web アプリケーション、EJB モジュール、およびコネクタモジュールにはサブコンポーネントが含まれています。たとえば、Web アプリケーションには 1 つまたは複数のサーブレットが含まれる場合があります。

1. ツリーコンポーネントで、「アプリケーション」ノードを展開します。
2. 記述子を表示するアプリケーションまたはモジュールタイプのノードを展開します。
3. 配備されているアプリケーションまたはモジュールのノードを選択します。
4. 「アプリケーション」または「モジュール」ページで、「サブコンポーネント」表が表示されます。

参考 同機能を持つ asadmin コマンド

`list-sub-components`

▼ 配備されているアプリケーションのモジュールの記述子を表示する

エンタープライズアプリケーション、Web アプリケーション、EBJ モジュール、コネクタモジュール、およびアプリケーションクライアントモジュールについては、モジュールの配備記述子を表示できます。

1. ツリーコンポーネントで、「アプリケーション」ノードを展開します。
2. 記述子を表示するアプリケーションまたはモジュールタイプのノードを選択します。
3. 配備されているアプリケーションまたはモジュールのノードを選択します。
4. 「記述子」タブを選択します。
5. 記述子ファイルのテキストを表示するには、ファイル名をクリックします。
   ページがファイルのコンテンツを表示します。この情報は読み取り専用です。

▼ アプリケーションまたはモジュールの配備を取り消す

アプリケーションまたはモジュールの配備を取り消すと、それがドメインからアンインストールされ、すべてのインスタンスからそのアプリケーションまたはモジュールに対する参照が削除されます。

1. ツリーコンポーネントで、「アプリケーション」ノードを展開します。
2. 配備を取り消すアプリケーションまたはモジュールタイプのノードを選択します。
3. 配備されているアプリケーションを一覧している表で、配備を取り消すアプリケーションまたはモジュールのチェックボックスにチェックマークを付けます。
4. 「配備取消し」をクリックします。

参考 同機能を持つ asadmin コマンド

`undeploy`

▼ アプリケーションまたはモジュールを有効または無効にする

配備されているアプリケーションやモジュールが有効である場合、クライアントはアクセスできます。無効の場合は、配備されているもののクライアントからはアクセスできません。デフォルトでは、「すべてのターゲットを有効にする」ラジオボタンがオンになっているので、アプリケーションまたはモジュールは配備すると有効になります。

1. ツリーコンポーネントで、「アプリケーション」ノードを展開します。
2. アプリケーションタイプのノードを展開します。
3. 配備されているアプリケーションまたはモジュールを有効または無効にするには、配備されているアプリケーションまたはモジュールの横にあるチェックボックスを選択します。

   1 つのターゲットでアプリケーションを有効にするには、次の手順に従います。

   a. アプリケーションのノードを選択します。
   b. 「ターゲット」タブをクリックします。
   c. 配備されているアプリケーションまたはモジュールの横にあるチェックボックスを選択します。
4. 「有効」または「無効」を選択します。

   これらのボタンを使用すると、すべてのターゲットでアプリケーションを有効化または無効化できます。

参考 同機能を持つ asadmin コマンド

`enable` および `disable`

▼ アプリケーションターゲットを管理する

アプリケーションまたはモジュールの配備後、ターゲットを管理することによって、アプリケーションまたはモジュールを参照するサーバーインスタンスやクラスタを管理します。

1. ツリーコンポーネントで、「アプリケーション」ノードを展開します。
2. アプリケーションタイプのノードを展開します。
3. 配備されているアプリケーションのノードを選択します。

4 「ターゲット」タブを選択します。

5 特定のターゲットインスタンスまたはクラスタ上のアプリケーションを有効化または無効化するには、ターゲットの横にあるチェックボックスをクリックして、「有効」または「無効」をクリックします。

6 アプリケーションのターゲットを追加または削除するには、「ターゲットの管理」を選択します。

7 ターゲットを追加または削除して、「了解」をクリックします。
アプリケーションが、修正されたターゲット一覧で使用できるようになります。

参考 同機能を持つ asadmin コマンド

`create-application-ref` および `delete-application-ref`

▼ 追加の仮想サーバーに配備する

ターゲットのサーバーインスタンスまたはクラスタにアプリケーションやモジュールを配備したら、それを追加の仮想サーバーと関連付けることができます。

1 配備されているアプリケーションまたはモジュールの「ターゲット」ページで、ターゲットの横にある「仮想サーバーの管理」をクリックします。

2 使用可能な仮想サーバーのリストに仮想サーバーターゲットを追加または削除します。

3 「了解」をクリックします。

複数のターゲットへの再配備

アプリケーションが複数のターゲット (スタンドアロンサーバーインスタンスまたはクラスタ) に配備されている場合、複数のターゲットへの再配備には 2 とおりの方法があります。次のいずれかの方法を使用して、アプリケーションを参照するすべてのサーバーインスタンスが必ず最新バージョンを受信するようにします。

開発環境

開発環境では、単にアプリケーションを再配備するだけです。アプリケーションはドメインに再配備され、動的再設定がターゲットサーバーインスタンスに対して有効になっている場合、アプリケーションを参照するすべてのターゲットは自動的に新しいバージョンを受信します。動的再設定は、デフォルトでは有効になっています。動的再設定がサーバーインスタンスに対して有効になっていない場合、サーバーインスタンスは再起動されるまで旧バージョンを使用し続けます。

本稼働環境

本稼働環境では、「段階的アップグレードについて」で説明されている手順に従います。

▼ 動的再読み込みを設定する

動的再読み込みを有効にすると、サーバーは配備されているアプリケーションの変更を定期的にチェックし、変更のあるアプリケーションを自動的に再読み込みします。変更は、手動で作成する `.reload` と呼ばれるファイルの日付の変更によって示されます。アプリケーションは、*domain-dir*`/applications/j2ee-modules` *module-name* または *domain-dir*`/applications/j2ee-apps/` *app-name* にインストールする必要があります。

次に例を示します。

```
/opt/SUNWappserver/domain/domain1/applications/j2ee-modules/webapps-simple
```

動的再読み込みは、変更したコードをすぐにテストできるため、開発環境で役に立ちます。しかし、本稼働環境では、動的再読み込みはパフォーマンスを低下させる可能性があります。

注 – 動的再読み込みは、デフォルトのサーバーインスタンスにのみ利用可能です。

動的再読み込みは、開発環境を対象としています。この機能は、本稼働環境の機能であるセッションの持続性とは互換性がありません。動的再読み込みが有効になっている場合は、セッションの持続性を有効にしないでください。

1. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを展開します。
2. **Server (管理サーバー)** をクリックします。
3. 「詳細」をクリックします。
4. 「アプリケーション設定」ページで、次の事項を設定します。
   - 再読み込み: 「有効」チェックボックスで動的再読み込みを有効または無効にします。
   - 再読込のポーリング間隔: 配備されているアプリケーション内の変更をサーバーがチェックする頻度を指定します。
   - 管理セッションタイムアウト: 管理セッションがタイムアウトし、再度ログインするまでの時間を指定します。

次の手順　動的再読み込みを使用してすべてのアプリケーションが動的に再読み込みされるようにシステムを設定したら、.reload と呼ばれるファイルを作成して、アプリケーションの

ディレクトリに配置します。このファイルには何もコンテンツがありません。アプリケーションの変更時に、UNIX の touch コマンドを使用するなどしてファイルの日付を変更すると、この変更が自動的に再読み込みされます。

開発者のための開発方法

関連項目

- 80 ページの「自動配備を使用する」
- 80 ページの「自動配備を使用する」
- 81 ページの「パッケージ化されていないアプリケーションをディレクトリから配備する」
- 82 ページの「deploytool ユーティリティーの使用」
- 82 ページの「配備計画の使用」

▼ 自動配備を使用する

自動配備機能を使うと、事前にパッケージ化されたアプリケーションやモジュールを *domain-dir*/autodeploy ディレクトリにコピーすることで配備できます。

たとえば、hello.war という名前のファイルを *domain-dir* /autodeploy ディレクトリにコピーします。アプリケーションの配備を取り消すには、autodeploy ディレクトリから hello.war ファイルを削除します。

管理コンソールまたは asadmin ツールを使用して、アプリケーションの配備を取り消すこともできます。この場合、アーカイブファイルはそのままになります。

注 – 自動配備は、デフォルトのサーバーインスタンスにのみ利用可能です。

自動配備機能は、開発環境を対象としています。この機能は、本稼働環境の機能であるセッションの持続性とは互換性がありません。自動配備が有効になっている場合は、セッションの持続性を有効にしないでください。

1. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを展開します。
2. **Server (**管理サーバー**)** をクリックします。
3. 「詳細」をクリックします。
4. 「アプリケーション設定」ページで、次の事項を設定します。
   a. 「有効」チェックボックスを選択または選択解除して、自動配備を有効または無効にします。

b. 「自動配備のポーリング間隔」フィールドで、アプリケーションやモジュールファイルの自動配備ディレクトリをサーバーが確認する頻度を指定します。

ポーリング間隔を変更しても、アプリケーションやモジュールの配備にかかる時間には影響ありません。

c. 自動配備ディレクトリでアプリケーションを構築したディレクトリを指定してあれば、ファイルをデフォルトの自動配備ディレクトリにコピーする必要はありません。

デフォルトは、サーバーインスタンスのルートディレクトリにある `autodeploy` というディレクトリです。

デフォルトでは、複数のサーバーインスタンスのディレクトリを手動で変更する必要をなくすために、変数が使用されます。これらの変数の詳細については、331 ページの「詳細ドメイン属性を設定する」を参照してください。

d. 配備の前にベリファイアを実行するには、「ベリファイア」を選択します。

ベリファイアはファイルの構造とコンテンツを調べます。大きなアプリケーションの検証は時間がかかる可能性があります。

e. JSP ページをプリコンパイルするには、「プリコンパイル」を選択します。

このチェックボックスを選択しない場合、JSP ページは最初のアクセスの実行時にコンパイルされます。コンパイルは時間がかかる可能性があるので、本稼働環境ではこのチェックボックスにチェックマークを付けてください。

▼ パッケージ化されていないアプリケーションをディレクトリから配備する

この機能は高度な開発者を対象としています。

ディレクトリ配備は、デフォルトのサーバーインスタンス (server) への配備にだけ使用します。クラスタまたはスタンドアロンサーバーインスタンスへの配備には使用できません。

パッケージ化されていないアプリケーションやモジュールを含むディレクトリを、分割ディレクトリと呼ぶことがあります。ディレクトリのコンテンツは、対応する J2EE アーカイブファイルのコンテンツと一致する必要があります。たとえば、ディレクトリから Web アプリケーションを配備する場合、ディレクトリのコンテンツは対応する WAR ファイルと同じになる必要があります。必要なディレクトリコンテンツの詳細については、適切な仕様を参照してください。

分割ディレクトリ内で配備記述子ファイルを直接変更できます。

環境が動的再読み込みを使用するように設定されている場合は、配備されたアプリケーションをディレクトリから動的に再読み込みすることもできます。詳細については、79 ページの「動的再読み込みを設定する」を参照してください。

1 管理コンソールで、配備プロセスを開始します。65 ページの「Web アプリケーションを配備する」を参照してください。

2 「配備」ページで、次の事項を設定します。

a. ラジオボタンをクリックして、サーバーからアクセス可能なパッケージファイルまたはディレクトリパスを指定します。

b. 「ファイルまたはディレクトリ」フィールドで、分割ディレクトリの名前を入力します。

参考 同機能を持つ asadmin コマンド

`deploydir`

deploytool ユーティリティーの使用

ソフトウェア開発者を対象として設計された `deploytool` ユーティリティーは、J2EE アプリケーションおよびモジュールをパッケージ化し、配備します。`deploytool` の使用手順については、『J2EE 1.4 Tutorial』を参照してください。

配備計画の使用

この機能は高度な開発者を対象としています。

配備計画は、Application Server に固有の配備記述子だけを含む JAR ファイルです。このような配備記述子、たとえば `sun-application.xml` などについては、『Application Server 開発者ガイド』で説明されています。配備計画は、JSR 88: J2EE Application Deploymentの実装の一部です。配備計画を使用して、Application Server に固有の配備記述子を含まないアプリケーションやモジュールを配備します。

配備計画を使用して配備を行うには、`asadmin deploy` コマンドの `--deploymentplan` オプションを指定します。たとえば、次のコマンドは、`mydeployplan.jar` ファイルによって指定される計画に従って、`myrosterapp.ear` ファイルのエンタープライズアプリケーションを配備します。

```
$ asadmin deploy --user admin ---deploymentplan mydeployplan.jar myrosterapp.ear
```

エンタープライズアプリケーション (EAR) の配備計画ファイルでは、`sun-application.xml` ファイルがルートに置かれています。各モジュールの配備記述子は、構文 *module-name.sun-dd-name* に従って格納されています。*sun-dd-name* は、モジュールタイプによって異なります。モジュールに CMP マッピングファイルが含まれる場合、ファイルは *module-name*`.sun-cmp-mappings.xml` という名前になります。`.dbschema` ファイルはルートレベルに格納されていて、スラッシュ (/) はシャープ記号 (#) に置き換えられます。次のリストは、エンタープライズアプリケーション (EAR) の配備計画ファイルの構造を示しています。

```
$ jar -tvf mydeployplan.jar
420 Thu Mar 13 15:37:48 PST 2003 sun-application.xml
370 Thu Mar 13 15:37:48 PST 2003 RosterClient.war.sun-web.xml
418 Thu Mar 13 15:37:48 PST 2003 roster-ac.jar.sun-application-client.xml
1281 Thu Mar 13 15:37:48 PST 2003 roster-ejb.jar.sun-ejb-jar.xml
2317 Thu Mar 13 15:37:48 PST 2003 team-ejb.jar.sun-ejb-jar.xml
3432 Thu Mar 13 15:37:48 PST 2003 team-ejb.jar.sun-cmp-mappings.xml
84805 Thu Mar 13 15:37:48 PST 2003 team-ejb.jar.RosterSchema.dbschema
```

Web アプリケーションまたはモジュールファイルの配備計画では、Application Server に固有の配備記述子がルートレベルにあります。スタンドアロンの EJB モジュールに CMP Bean が含まれる場合、配備計画には、ルートレベルの `sun-cmp-mappings.xml` ファイルや `.dbschema` ファイルが含まれます。次のリストでは、配備計画が CMP bean を示しています。

```
$ jar r -tvf myotherplan.jar
3603 Thu Mar 13 15:24:20 PST 2003 sun-ejb-jar.xml
3432 Thu Mar 13 15:24:20 PST 2003 sun-cmp-mappings.xml
84805 Thu Mar 13 15:24:20 PST 2003 RosterSchema.dbschema
```

◆ ◆ ◆ 第 3 章

JDBC リソース

この章では、データベースにアクセスするアプリケーションで必要な、JDBC リソースの設定方法について説明します。この章には次の節が含まれています。

JDBC リソースと接続プールについて

JDBC リソース

データを格納、編成、取得するために、ほとんどのアプリケーションはリレーショナルデータベースを使用します。J2EE アプリケーションは、JDBC API を介してリレーショナルデータベースにアクセスします。

JDBC リソース (データソース) は、アプリケーションにデータベースへ接続する手段を提供します。通常、管理者は、ドメインに配備されたアプリケーションがアクセスする各データベースの JDBC リソースを作成します。ただし、データベース用に複数の JDBC リソースを作成できます。

JDBC リソースを作成するには、リソースを識別する一意の JNDI 名を指定します。「JNDI 名とリソース」の節を参照してください。通常、JDBC リソースの JNDI 名は `java:comp/env/jdbc` サブコンテキストにあります。たとえば、給与データベースの JNDI 名は `java:comp/env/jdbc/payrolldb` にある可能性があります。すべてのリソース JNDI 名

は java:comp/env サブコンテキストにあるので、管理コンソールの JDBC リソースにある JNDI 名を指定するときは、jdbc/*name* だけを入力します。たとえば、給与データベースには jdbc/payrolldb を指定します。

JDBC 接続プール

JDBC リソースを作成するには、関連した接続プールを指定します。複数の JDBC リソースで 1 つの接続プールを指定できます。

JDBC 接続プールとは、特定のデータベースのための再利用可能な接続のグループです。新しい物理接続をそれぞれ作成するには時間がかかるので、パフォーマンスの向上のためにサーバーは利用可能な接続のプールを保持しています。アプリケーションが接続を要求すると、プールから 1 つの接続が取得されます。アプリケーションが接続を閉じると、接続はプールに返されます。

接続プールのプロパティーは、データベースベンダーによっては異なる場合もあります。共通のプロパティーには、データベースの名前 (URL)、ユーザー名、およびパスワードがあります。

JDBC リソースと接続プールの協調動作について

データを格納、編成、取得するために、ほとんどのアプリケーションはリレーショナルデータベースを使用します。J2EE アプリケーションは、JDBC API を介してリレーショナルデータベースにアクセスします。アプリケーションがデータベースにアクセスするには、接続を取得する必要があります。

実行時に、アプリケーションがデータベースに接続されると次のことが行われます。

1. JNDI API を介して呼び出しを行うことにより、アプリケーションはデータベースに関連した JDBC リソース (データソース) を取得します。

   リソースの JNDI 名を取得すると、ネーミングおよびディレクトリサービスが JDBC リソースを検索します。JDBC リソースはそれぞれ接続プールを指定します。
2. JDBC リソースを経由して、アプリケーションはデータベース接続を取得します。

   バックグラウンドで、アプリケーションサーバーはデータベースに対応した接続プールから物理接続を取得します。プールは、データベース名 (URL)、ユーザー名、およびパスワードなどの接続属性を定義します。
3. データベースに接続すると、アプリケーションはデータベースのデータの読み込み、変更、および追加を行うことができます。

   アプリケーションは JDBC API を呼び出すことにより、データベースにアクセスします。JDBC ドライバはアプリケーションの JDBC 呼び出しをデータベースサーバーのプロトコルに変換します。
4. データベースへのアクセスが完了すると、アプリケーションは接続を終了します。

   アプリケーションサーバーは接続を接続プールに返します。接続がプールに戻されると、次のアプリケーションがその接続を利用できるようになります。

データベースアクセスの設定

▼ データベースアクセスを設定する

1. サポートされたデータベース製品をインストールします。
   Application Server がサポートするデータベース製品のリストについては、リリースノートを参照してください。
2. データベース製品の **JDBC** ドライバをインストールします。
3. ドメインのサーバーインスタンスにアクセスできるドライバの **JAR** ファイルを作成します。**87** ページの「**JDBC** ドライバを統合する」を参照してください。
4. データベースを作成します。
   通常、アプリケーションプロバイダがデータベースを作成し移行するためのスクリプトを配信します。
5. データベースの接続プールを作成します。**88** ページの「**JDBC** 接続プールを作成する」を参照してください。
6. 接続プールを示す **JDBC** リソースを作成します。**93** ページの「**JDBC** リソースを作成する」を参照してください。

▼ JDBC ドライバを統合する

JDBC ドライバはアプリケーションの JDBC 呼び出しをデータベースサーバーのプロトコルに変換します。

◗ **JDBC** ドライバを管理ドメインに統合するには、次のどれかを実行します。

- 共通クラスローダーにアクセスできるドライバを作成する
  a. ドライバの **JAR** および **ZIP** ファイルを *domain-dir* `/lib` ディレクトリにコピーするか、またはドライバのクラスファイルを *domain-dir*`/lib/ext` ディレクトリにコピーします。
  b. ドメインを再起動します。

- システムクラスローダーにアクセスできるドライバを作成する

  a. 管理コンソールのツリービュー (左側の区画) で、「設定」を選択します。

  b. **default-config** など、希望する設定を選択します。

  c. 「**JVM** 設定」を選択します。

  d. 「**JVM** 設定」ページで、「パス設定」タブをクリックします。

  e. 「クラスパスのサフィックス」フィールドで、ドライバの **JAR** ファイルの完全修飾パス名を入力します。

  f. 「保存」をクリックします。

  g. サーバーを再起動します。

JDBC 接続プールについて

▼ JDBC 接続プールを作成する

JDBC 接続プールとは、特定のデータベースのための再利用可能な接続のグループです。管理コンソールでプールを作成すると、管理者は実際には特定のデータベースへの接続の項目を定義していることになります。

始める前に　プールを作成するには、まず JDBC ドライバをインストールして統合する必要があります。87 ページの「JDBC ドライバを統合する」を参照してください。

「接続プールを作成」ページを構築する際は、JDBC ドライバおよびデータベースベンダーに固有の特定のデータを入力する必要があります。処理を開始する前に、次の情報を集めます。

- データベースベンダー名
- `javax.sql.DataSource` (ローカルトランザクションのみ) や `javax.sql.XADataSource` (グローバルトランザクション) などのリソースタイプ
- データソースクラス名
- データベース名 (URL)、ユーザー名、およびパスワードなどの必要なプロパティー

1. 管理コンソールのツリービュー(左側の区画)で、「リソース」ノードを展開します。
2. 「リソース」ノードで、「**JDBC**」ノードを展開します。
3. 「**JDBC**」ノードで、「接続プール」ノードを選択します。
4. 「接続プール」ページで、「新規」をクリックします。
5. 最初の「接続プールを作成」ページで、次の一般設定を指定します。
   a. 「名前」フィールドで、プールの論理名を入力します。
      JDBC リソースの作成時にこの名前を指定します。
   b. 「リソースタイプ」コンボボックスからエントリを選択します。
   c. 「データベースベンダー」コンボボックスからエントリを選択します。
6. 「次へ」をクリックします。
7. **2** 番目の「接続プールを作成」ページで、「データソースクラス名」フィールドの値を指定します。
   JDBC ドライバに前のページで指定したリソースタイプとデータベースベンダーのデータソースクラスがある場合は、「データソースクラス名」フィールドの値が提示されます。
8. 「次へ」をクリックします。
9. 最後に **3** 番目の「接続プールを作成」ページで、次のタスクを実行します。
   a. 「一般設定」セクションでその値が正しいことを確認します。
   b. 「プール設定」、「接続検証」フィールド、および「トランザクション遮断」セクション用に、デフォルト値を保持します。
      これらの設定はあとで変更するのがもっとも便利です。90 ページの「JDBC 接続プールを編集する」を参照してください。
   c. 「追加プロパティー」テーブルで、データベース名 **(URL)**、ユーザー名、およびパスワードなど、必要なプロパティーを追加します。
10. 「完了」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-jdbc-connection-pool`

▼ JDBC 接続プールを編集する

「JDBC 接続プールを編集」ページは、名前を除く既存プールのすべての設定を変更する手段を提供します。

1 ツリーコンポーネントで、「リソース」ノードを展開します。

2 「リソース」ノードで、「**JDBC**」ノードを展開します。

3 「**JDBC**」ノードで、「接続プール」ノードを展開します。

4 編集するプールのノードを選択します。

5 「コネクタ接続プールを編集」ページで、必要な変更を行います。

a. 一般設定を変更します。

一般設定の値は、インストールした固有の JDBC ドライバにより異なります。これらの設定は、Java プログラミング言語で記述されたクラスやインタフェースの名前です。

パラメータ	説明
データソースクラス名	`DataSource` API、`XADataSource` API、あるいはその両方を実装するベンダー固有のクラス名。このクラスは JDBC ドライバにあります。
リソースタイプ	選択肢には `javax.sql.DataSource`(ローカルトランザクションのみ)、`javax.sql.XADataSource`(グローバルトランザクション)、および `java.sql.ConnectinPoolDataSource`(ローカルトランザクション、パフォーマンス向上の可能性あり)があります。

b. プール設定を変更します。

一連の物理データベース接続はプール内にあります。アプリケーションが接続を要求すると、接続はプールから削除され、アプリケーションが接続を解放すると、接続はプールに返されます。

パラメータ	説明
初期および最小プールサイズ	プール内の接続の最小数。この値は、プールを最初に作成したり、Application Server を起動したりするときの、プールに含まれる接続の数も指定します。
最大プールサイズ	プールに含まれる接続の最大数。

パラメータ	説明
プールサイズ変更量	プールのサイズが最小プールサイズに近づくと、プールサイズが一括処理で変更されます。この値は一括処理での接続の数を指定します。この値を過大に設定すると接続の再利用が遅れ、過小に設定すると効率が落ちます。
アイドルタイムアウト	プールで接続がアイドル状態のままでいられる最長時間を指定します。この時間を過ぎると、接続はプールから削除されます。
最大待ち時間	接続タイムアウトになる前に接続を要求するアプリケーションが待つ時間。デフォルトの待ち時間は長いので、アプリケーションがハングアップしているように見える可能性があります。

c. 接続検証設定を変更します。

オプションで、アプリケーションサーバーは接続が渡される前にそれを検証することができます。この検証により、ネットワークやデータベースサーバーに障害が発生してデータベースが利用できなくなった場合でも、アプリケーションサーバーが自動的にデータベース接続を再確立できます。接続の検証は追加オーバーヘッドとなるため、パフォーマンスに若干の影響が生じます。

パラメータ	説明
接続検証	必要なチェックボックスを選択して、接続検証を有効にします。
検証方法	アプリケーションサーバーは、auto-commit、metadata、および table の 3 つの方法でデータベース接続を検証できます。 auto-commit と metadata - アプリケーションサーバーは、`con.getAutoCommit()` メソッドと `con.getMetaData()` メソッドを呼び出して接続を検証します。ただし、多くの JDBC ドライバでは、これらの呼び出しの結果をキャッシュしているので、常に信頼のある検証が行われるとは限りません。呼び出しがキャッシュされるかどうかについて、ドライバベンダーに問い合わせる必要があります。 table - アプリケーションは指定したデータベース表に問い合わせます。表は実在し、アクセス可能である必要がありますが、行は必要ありません。多くの行を持つ既存の表や、頻繁にアクセスされる表を使用しないでください。
表名	「検証方法」コンボボックスで表を選択した場合は、ここでデータベース表の名前を指定します。
すべて障害で	「すべての接続を閉じる」のチェックボックスを選択してある場合、1 つの接続が失敗すると、アプリケーションサーバーはプールに含まれるすべての接続を閉じ、それらを再確立します。そのチェックボックスを選択していない場合は、使用されるときにだけ個々の接続が再確立されます。

d. トランザクション遮断設定を変更します。

データベースは通常多くのユーザーが同時にアクセスするため、あるトランザクションがデータを読み込もうとするときに別のトランザクションが同じデータを更新する可能性があります。トランザクションの遮断レベルは、更新されるデータがほかのトランザクションに見える度合いを定義します。遮断レベルの詳細については、データベースベンダーのマニュアルを参照してください。

パラメータ	説明
トランザクション遮断	プールの接続のトランザクション遮断レベルを選択できます。指定しない場合、接続は JDBC ドライバによって設定されるデフォルトの遮断レベルがプールに適用されます。
遮断レベルを保証	遮断レベルを指定した場合にだけ適用されます。「保証」チェックボックスを選択する場合は、プールから取得されるすべての接続が同じ遮断レベルを持ちます。たとえば、最後の使用時に `con.setTransactionIsolation` を使って接続の遮断レベルをプログラム的に変更した場合、このメカニズムによって状態が指定された遮断レベルに戻されます。

e. プロパティーを変更します。

「追加プロパティー」テーブルで、データベース名 (URL)、ユーザー名、およびパスワードなど、必要なプロパティーを指定できます。データベースベンダーによってプロパティーが異なるため、詳細については、ベンダーのマニュアルを調べてください。

6 「保存」をクリックします。

参照
- 88 ページの「JDBC 接続プールを作成する」
- 92 ページの「接続プール設定を検証する」
- 93 ページの「JDBC 接続プールを削除する」

▼ 接続プール設定を検証する

1 データベースサーバーを起動します。

2 「**Ping**」をクリックします。

管理コンソールがデータベースに接続を試みます。エラーメッセージが表示される場合は、データベースサーバーが再起動しているかどうかを確認します。

▼ JDBC 接続プールを削除する

1 ツリーコンポーネントで、「リソース」ノードを展開します。

2 「リソース」ノードで、「**JDBC**」ノードを展開します。

3 「**JDBC**」ノードで、「接続プール」ノードを選択します。

4 「接続プール」ページで、削除するプールのチェックボックスにチェックマークを付けます。

5 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

```
delete-jdbc-connection-pool
```

JDBC リソースについて

▼ JDBC リソースを作成する

JDBC リソース (データソース) は、アプリケーションにデータベースへ接続する手段を提供します。

始める前に JDBC リソースを作成する前に、まず JDBC 接続プールを作成します。88 ページの「JDBC 接続プールを作成する」を参照してください。

1 ツリーコンポーネントで、「リソース」ノードを展開します。

2 「リソース」ノードで、「**JDBC**」ノードを展開します。

3 「**JDBC**」ノードで、「**JDBC** リソース」ノードを選択します。

4 「**JDBC** リソース」ページで、「新規」をクリックします。

5 「**JDBC** リソースを作成」ページで、リソースの設定を指定します。

a. 「**JNDI** 名」フィールドに、名前を入力します。
慣例により、名前は `jdbc/` 文字列で始まります。次に例を示します。`jdbc/payrolldb`。スラッシュを忘れないでください。

b. 「プール名」コンボボックスから、新しい **JDBC** リソースに関連付けられた接続プールを選択します。

c. リソースを利用不可にする場合は、「有効」チェックボックスの選択を解除します。
デフォルトでは、リソースは作成すると同時に利用可能 (有効) です。

d. 「説明」フィールドで、リソースの簡単な説明を入力します。

e. 「ターゲット」セクションで、リソースが利用できるターゲット (クラスタおよびスタンドアロンサーバーインスタンス) を指定します。
左側の希望するターゲットを選択し、「追加」をクリックして選択したターゲットのリストに追加します。

6 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-jdbc-resource`

▼ JDBC リソースを編集する

1 ツリーコンポーネントで、「リソース」ノードを展開します。

2 「リソース」ノードで、「**JDBC**」ノードを展開します。

3 「**JDBC**」ノードで、「**JDBC** リソース」ノードを展開します。

4 編集する **JDBC** リソースのノードを選択します。

5 「**JDBC** リソースを編集」ページで、次のタスクを実行できます。

a. 「プール名」コンボボックスから、別の接続プールを選択します。

b. 「説明」フィールドで、リソースの簡単な説明を変更します。

c. チェックボックスを選択または選択解除して、リソースを有効または無効にします。

d. 「ターゲット」タブを選択して、リソースが利用できるターゲット **(**クラスタおよびスタンドアロンサーバーインスタンス**)** を変更します。

リスト内の既存のターゲットのチェックボックスを選択し、「有効」をクリックしてそのターゲットのリソースを有効にするか、または「無効」をクリックしてそのターゲットのリソースを無効にします。

「ターゲットの管理」をクリックして、リストにターゲットを追加または削除します。「ターゲットの管理」ページで、左側の利用可能なリストから希望するターゲットを選択し、「追加」をクリックして選択したターゲットのリストに追加します。「削除」をクリックして、選択したリストからターゲットを削除します。

「了解」をクリックして、利用可能なターゲットの変更を保存します。

6 「保存」をクリックして、編集を適用します。

▼ JDBC リソースを削除する

1 ツリーコンポーネントで、「リソース」ノードを展開します。

2 「リソース」ノードで、「**JDBC**」ノードを展開します。

3 「**JDBC**」ノードで、「接続プール」ノードを選択します。

4 「接続プール」ページで、削除するプールのチェックボックスにチェックマークを付けます。

5 「削除」をクリックします。

▼ JDBC リソースを有効または無効にする

1 ツリーコンポーネントで、「**JDBC** リソース」ノードまたは「スタンドアロンインスタンス」を展開し、「サーバーインスタンス」ノードの「リソース」タブを選択します。

2 「リソース」ページで、有効または無効にするリソースのチェックボックスにチェックマークを付けます。

3 「有効」または「無効」を選択します。

持続マネージャーリソースについて

▼ 持続マネージャーリソースを作成する

この機能は下位互換性のために必要です。Application Server のバージョン 7 で実行するには、コンテナ管理による持続性 Beans (EJB コンポーネントのタイプの 1 つ) を使用したアプリケーションの持続マネージャーリソースが必要でした。代わりに JDBC リソースを使用することをお勧めします。

1. ツリーコンポーネントで、「リソース」ノードを展開します。
2. 「リソース」ノードで、「持続マネージャー」ノードを選択します。
3. 「持続マネージャー」ページで、「新規」をクリックします。
4. 「持続マネージャーを作成」ページで、次の設定を指定します。

   a. 「**JNDI** 名」フィールドに、名前を入力します。
   次に例を示します。 `jdo/mypm`。スラッシュを忘れないでください。

   b. 「ファクトリクラス」フィールドで、このリソースで提供されているデフォルトクラスを保持するか、またはほかの実装クラスを入力します。

   c. 「接続プール」コンボボックスから、新しい持続性マネージャーリソースが属する接続プールを選択します。

   d. 新しい持続マネージャーリソースを無効にするには、「有効」チェックボックスを選択解除します。
   デフォルトでは、新しい持続マネージャーリソースが有効になります。

   e. 「ターゲット」セクションで、リソースが利用できるターゲット (クラスタおよびスタンドアロンサーバーインスタンス) を指定します。
   左側の希望するターゲットを選択し、「追加」をクリックして選択したターゲットのリストに追加します。

5. 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-persistence-resource`

▼ 持続マネージャーリソースを編集する

1. 「持続マネージャープロパティーを編集」タブから、「プロパティーを追加」を選択します。
   「追加プロパティー」テーブルに新しい行が追加されます。
2. 希望するプロパティーと値を追加します。

▼ リソースターゲットを管理する

1. 「ターゲット」タブを選択して、リソースが存在するターゲット(クラスタおよびスタンドアロンサーバーインスタンス)を変更します。
2. リスト内の既存のターゲットのチェックボックスを選択し、「有効」をクリックしてそのターゲットのリソースを有効にするか、または「無効」をクリックしてそのターゲットのリソースを無効にします。
3. 「ターゲットの管理」をクリックして、リストにターゲットを追加または削除します。
   「ターゲットの管理」ページで、左側の利用可能なリストから希望するターゲットを選択し、「追加」をクリックして選択したターゲットのリストに追加します。「削除」をクリックして、選択したリストからターゲットを削除します。
4. 「了解」をクリックして、利用可能なターゲットの変更を保存します。
5. 「保存」をクリックします。

▼ 持続マネージャーリソースを削除する

1. ツリーコンポーネントで、「持続マネージャー」ノードを展開します。
2. 「持続マネージャー」ノードを選択します。
3. 「持続マネージャー」ページで、削除する持続マネージャーのチェックボックスにチェックマークを付けます。
4. 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-persistence-resource`

▼ 接続マネージャーリソースを有効または無効にする

1. ツリーコンポーネントで、「持続マネージャー」ノードを展開します。
2. 有効または無効にするリソースのチェックボックスにチェックマークを付けます。
3. 「有効」または「無効」を選択します。

◆ ◆ ◆ 第 4 章

Java Message Service (JMS) リソースの設定

この章では、Java Message Service (JMS) API を使用するアプリケーションのリソースを設定する方法について説明します。この章には次の節が含まれています。

JMS リソースについて

Application Server の JMS プロバイダ

Application Server は、Sun Java System Message Queue (従来の Sun ONE Message Queue) を Application Server に統合することによって Java Message Service (JMS) API を実装します。基本的な JMS API 管理タスクの場合は、Application Server の管理コンソールを使用します。Message Queue クラスタの管理など、高度なタスクの場合は、*MQ-install-dir*`/imq/bin` ディレクトリに用意されたツールを使用します。

Message Queue の管理の詳細については、『Message Queue 管理ガイド』を参照してください。

JMS リソース

JMS (Java Message Service) API は、次の 2 種類の管理対象オブジェクトを使用します。

- 接続ファクトリ。アプリケーションがプログラムでほかの JMS オブジェクトを作成できるようにするオブジェクトです。
- 送信先。メッセージのリポジトリとして機能します。

オブジェクトは管理された上で作成され、その作成方法は JMS の実装に固有になります。Application Server で、次のタスクを実行します。

- 接続ファクトリリソースを作成することによって、接続ファクトリを作成します。
- 次の 2 つのオブジェクトを作成して、送信先を作成します。
  - 物理送信先
  - 物理送信先を参照する送信先リソース

JMS アプリケーションは、JNDI API を使用して接続ファクトリと送信先リソースにアクセスします。JMS アプリケーションは、通常接続ファクトリと送信先を少なくとも 1 つずつ使います。作成するリソースを確認するには、アプリケーションを理解したり、アプリケーションの開発者の意見を確認したりすることをお勧めします。

接続ファクトリには次の 3 つのタイプがあります。

- ポイントツーポイント通信で使用する `QueueConnectionFactory` オブジェクト
- パブリッシュ - サブスクライブ通信で使用する `TopicConnectionFactory` オブジェクト
- ポイントツーポイント通信とパブリッシュ - サブスクライブ通信の両方で使用できる `ConnectionFactory` オブジェクト。新しいアプリケーションでの使用をお勧めします。

送信先には次の 2 種類があります。

- ポイントツーポイント通信で使用する `Queue` オブジェクト
- パブリッシュ - サブスクライブ通信で使用する `Topic` オブジェクト

『*J2EE 1.4 Tutorial*』の JMS に関する章では、この 2 つの通信タイプについての詳細および JMS のほかの側面が説明されています (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html` を参照)。

リソースを作成する順序は重要ではありません。

J2EE アプリケーションでは、次の手順に従って Application Server の配備記述子に接続ファクトリリソースと送信先リソースを指定します。

- 接続ファクトリ JNDI 名は `resource-ref` または `mdb-connection-factory` 要素に指定します。
- 送信先リソース JNDI 名は、メッセージ駆動型 Bean の `ejb` 要素と `message-destination` 要素に指定します。
- 物理送信先名は、Enterprise JavaBean 配備記述子の `message-driven` 要素または `message-destination-ref` 要素のいずれかにある `message-destination-link` 要素に指定します。さらに、`message-destination` 要素にも指定します。`message-destination-ref` 要素は、新しいアプリケーションで推奨されない

`resource-env-ref` 要素から置き換わります。Application Server 配備記述子の `message-destination` 要素で、物理送信先名と送信先リソース名をリンクします。

JMS リソースとコネクタリソースの関係

Application Server は、`jmsra` という名前のシステムリソースアダプタを使用して JMS を実装します。JMS リソースを作成すると、Application Server がコネクタリソースも自動的に作成します。コネクタリソースは、管理コンソールのツリービューに表示される「コネクタ」ノードの下に表示されます。

ユーザーが作成する各 JMS 接続ファクトリに対して、Application Server はコネクタ接続プールとコネクタリソースを作成します。ユーザーが作成する個々の JMS 送信先に対して、Application Server は管理オブジェクトリソースを作成します。ユーザーがJMS リソースを削除するときに、Application Server はコネクタリソースを自動的に削除します。

「JMS リソース」ノードの代わりに管理コンソールの「コネクタ」ノードを使用して、JMS システムリソースアダプタ用のコネクタリソースを作成できます。 詳細については、第 7 章を参照してください。

JMS 接続ファクトリに関する管理コンソールタスク

▼ JMS 接続ファクトリリソースを作成する

1. ツリーコンポーネントで、「リソース」ノードを展開し、次に「**JMS** リソース」ノードを展開します。
2. 「接続ファクトリ」ノードを選択します。
3. 「**JMS** 接続ファクトリ」ページで、「新規」をクリックします。
   「JMS 接続ファクトリを作成」ページが表示されます。
4. 「**JNDI** 名」フィールドに、接続ファクトリの名前を入力します。
   JMS リソースのネーミングサブコンテキストのプレフィックス `jms/` を使用することをお勧めします。次に例を示します。`jms/ConnectionFactory1`
5. 「タイプ」ドロップダウンリストから、`javax.jms.ConnectionFactory`、`javax.jms.QueueConnectionFactory`、または `javax.jms.TopicConnectionFactory` を選択します。

6 実行時にリソースを有効にするには、「有効」チェックボックスにチェックマークを付けます。

7 「詳細」セクションで、接続ファクトリの属性として必要な値を変更します。

これらの属性の詳細については、104 ページの「JMS 接続ファクトリリソースを編集する」を参照してください。Application Server は、接続ファクトリを作成するコネクタ接続プールにこれらの属性を適用します。

8 **JMS** 接続ファクトリリソースでは、トランザクションサポートの値を次のように指定します。

- トランザクションスコープ内で複数のリソースの使用が必要となるトランザクションとして使用可能なリソースに `XATransaction` **(**デフォルト値**)** を指定します。

  たとえば、このリソースには、JDBC リソース、コネクタリソース、またはその他の JMS 接続ファクトリリソースも含まれます。この値により、最高の柔軟性が提供されます。`XATransaction` として設定されるリソースは、2 フェーズコミットオペレーションに関与します。

- トランザクションスコープ内でリソースが **1** つだけ必要となるトランザクションか、複数の **XA** リソースが必要となる分散トランザクションの直前のエージェントのいずれかとして使用可能なリソースに `LocalTransaction` を指定します。

  この値により、大幅なパフォーマンスの向上が得られます。`LocalTransaction` として設定されるリソースは、2 フェーズコミットオペレーションに関与しません。

- トランザクションにまったく関与しないリソースに `NoTransaction` を指定します。

  この設定は JMS アプリケーションでの使用に限られます。

9 「追加プロパティー」セクションで、アプリケーションに必要なプロパティーの値を指定します。次の表には、使用可能なプロパティーが一覧表示されています。

プロパティー名	説明
`ClientId`	永続的なサブスクライバが使用する接続ファクトリのクライアント ID を指定します。

プロパティー名	説明
AddressList	アプリケーションが通信するメッセージブローカインスタンスまたはインスタンスの名前 (およびオプションでポート番号) を指定します。リスト内の各アドレスは接続用のホスト名、およびオプションでホストポートと接続サービスを指定します。たとえば、earth や earth:7677 などの値を指定できます。メッセージブローカがデフォルト (7676) 以外のポートで実行している場合は、ポート番号を指定します。プロパティー設定で、クラスタ化された環境の複数のホストおよびポートが指定されている場合、AddressListBehavior プロパティーが RANDOM に設定されていないかぎり、リストで最初に利用できるホストが使用されます。 詳細については、『*Message Queue Developer's Guide for Java Clients*』を参照してください。 デフォルト: ローカルホストおよびデフォルトポート番号 (7676) です。クライアントは、ローカルホストのポート 7676 のブローカへの接続を試行します。
MessageServiceAddressList	AddressList と同じです。このプロパティー名は推奨されていません。代わりに AddressList を使用します。
UserName	接続ファクトリのユーザー名。 デフォルト: guest
Password	接続ファクトリのパスワード。 デフォルト: guest
ReconnectEnabled	有効 (値を true) にすると、接続が失われたときに、クライアントランタイムがメッセージサーバー (または AddressList で指定したアドレスのリスト) に再接続を試みるように指定します。 デフォルト: true
ReconnectAttempts	クライアントランタイムがリストの次のアドレスを試行する前に、AddressList に指定した各アドレスへの接続 (または再接続) を試行する回数を指定します。値 -1 は、再試行回数が無制限であることを示します。クライアントランタイムは、接続が成功するまで最初のアドレスへの接続を試みます。 デフォルト: 3
ReconnectInterval	再接続を試行する間隔をミリ秒単位で指定します。この間隔は、AddressList で指定した各アドレスおよびリストの次のアドレスへの試行に適用されます。間隔が短すぎると、ブローカがリカバリする時間がなくなります。間隔が長すぎると、再接続が許容できない遅延を示す場合があります。 デフォルト: 30000

プロパティー名	説明
AddressListBehavior	接続の試行を AddressList 属性で指定したアドレスの順序 (PRIORITY) で行うか、またはランダムな順序 (RANDOM) で行うかを指定します。 RANDOM は、再接続で AddressList から任意のアドレスが選択されたことを意味します。多数のクライアントが同じ接続ファクトリを使用して接続を試行する可能性がある場合は、RANDOM を指定すると、すべてのクライアントが同じアドレスに接続しないようにすることができます。 PRIORITY は、再接続が常に AddressList に指定した最初のサーバーのアドレスへの接続を試行し、最初のブローカが利用できない場合にのみほかのアドレスを使用することを意味します。 デフォルト: RANDOM
AddressListIterations	接続の確立 (または再確立) のために、クライアントランタイムが AddressList を介して反復する回数を指定します。値 -1 は試行回数が無制限であることを示します。 デフォルト: 3

10 「ターゲット」領域で、次を実行します。

a. 「選択可能」リストから、リソースを使用するアプリケーションが配備されるターゲットを選択します。
選択可能なターゲットには、利用可能なクラスタおよびサーバーインスタンスと、デフォルトのサーバーインスタンス server が含まれます。

b. 「追加」をクリックして、ターゲットを「選択」リストに移動します。

11 「了解」をクリックして、接続ファクトリを保存します。

参考 同機能を持つ asadmin コマンド

```
create-jms-resource
```

▼ JMS 接続ファクトリリソースを編集する

1 ツリーコンポーネントで、「リソース」ノードを展開し、次に「JMS リソース」ノードを展開します。

2 「接続ファクトリ」ノードを展開します。

3 編集する接続ファクトリを選択します。

4 「JMS 接続ファクトリを編集」ページで、次のタスクを実行できます。

- 「説明」フィールドのテキストの変更。
- 「有効」チェックボックスの選択または選択解除による、リソースの有効化または無効化。
- 「詳細」セクションでの属性の値の変更。
- プロパティーの追加、削除、または変更。

5 オプションで、「ターゲット」タブをクリックして、「**JMS** 接続ファクトリリソースターゲット」ページを表示します。このページで、次を実行します。

a. 「ターゲットの管理」をクリックして、「リソースターゲットの管理」ページを開きます。

このページで、「選択可能」列と「選択」列の間でターゲットを移動します。「選択」列に、リソースを使用するアプリケーションが配備されるターゲットを必ず配置します。選択可能なターゲットには、利用可能なクラスタおよびサーバーインスタンスと、デフォルトのサーバーインスタンス `server` が含まれます。「了解」をクリックして変更を保存します。

b. ターゲットのチェックボックスを選択して、「有効」または「無効」をクリックし、そのターゲットのリソースを有効または無効にします。

6 「保存」をクリックして変更を保存します。

▼ JMS 接続ファクトリリソースを削除する

1 ツリーコンポーネントで、「リソース」ノードを展開し、次に「**JMS** リソース」ノードを展開します。

2 「接続ファクトリ」ノードを選択します。

3 「**JMS** 接続ファクトリ」ページで、削除する接続ファクトリ名の横にあるチェックボックスにチェックマークを付けます。

4 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-jms-resource`

JMS 送信先リソースに関する管理コンソールタスク

▼ JMS 送信先リソースを作成する

1 ツリーコンポーネントで、「リソース」ノードを展開し、次に「**JMS** リソース」ノードを展開します。

2 「送信先リソース」ノードを選択します。

3 「**JMS** 送信先リソース」ページで、「新規」をクリックします。「**JMS** 送信先リソースを作成」ページが表示されます。

4 「**JNDI** 名」フィールドに、リソースの名前を入力します。
JMS リソースのネーミングサブコンテキストのプレフィックス `jms/` を使用することをお勧めします。次に例を示します。`jms/Queue`

5 「タイプ」ドロップダウンリストから、`javax.jms.Topic` または `javax.jms.Queue` を選択します。

6 実行時にリソースを有効にするには、「有効」チェックボックスにチェックマークを付けます。

7 「追加プロパティー」セクションで、プロパティーの値を指定します。
次の表には、使用可能なプロパティーが一覧表示されています。

プロパティー名	説明
`Name`	(必須) リソースが参照する物理送信先の名前。
`Description`	物理送信先の説明。

8 「ターゲット」領域で、次を実行します。

a. 「選択可能」リストから、リソースを使用するアプリケーションが配備されるターゲットを選択します。
選択可能なターゲットには、利用可能なクラスタおよびサーバーインスタンスと、デフォルトのサーバーインスタンス `server` が含まれます。

b. 「追加」をクリックして、ターゲットを「選択」リストに移動します。

9 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-jms-resource`

▼ JMS 送信先リソースを編集する

1 ツリーコンポーネントで、「リソース」ノードを展開し、次に「**JMS** リソース」ノードを展開します。

2 「送信先リソース」ノードを展開します。

3 編集する送信先リソースを選択します。

4 「**JMS** 送信先リソースを編集」ページで、次のタスクを実行できます。

- リソースのタイプの変更。
- 「説明」フィールドのテキストの変更。
- 「有効」チェックボックスの選択または選択解除による、リソースの有効化または無効化。
- **Name** または **Description** プロパティーの追加、削除、または変更。

5 「保存」をクリックして変更を保存します。

6 オプションで、「ターゲット」タブをクリックして、「**JMS** 送信先リソースターゲット」ページを表示します。このページで、次を実行します。

a. 「ターゲットの管理」をクリックして、「リソースターゲットの管理」ページを開きます。

このページで、「選択可能」列と「選択」列の間でターゲットを移動します。「選択」列に、リソースを使用するアプリケーションが配備されるターゲットを必ず配置します。選択可能なターゲットには、利用可能なクラスタおよびサーバーインスタンスと、デフォルトのサーバーインスタンス `server` が含まれます。「了解」をクリックして変更を保存します。

b. ターゲットのチェックボックスを選択して、「有効」または「無効」をクリックし、そのターゲットのリソースを有効または無効にします。

▼ JMS 送信先リソースを削除する

1 ツリーコンポーネントで、「リソース」ノードを展開し、次に「**JMS** リソース」ノードを展開します。

2 「送信先リソース」ノードを選択します。

3 「**JMS** 送信先リソース」ページで、削除する送信先リソース名のチェックボックスにチェックマークを付けます。

4 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-jms-resource`

JMS 物理送信先に関する管理コンソールタスク

▼ JMS 物理送信先を作成する

本稼動環境では、必ず物理送信先を作成する必要があります。ただし、開発およびテスト段階では、この手順は不要です。アプリケーションが最初に送信先リソースにアクセスすると、Message Queue は、送信先リソースの名前プロパティーで指定した物理送信先を自動的に作成します。物理送信先は一時的なものなので、Message Queue の設定プロパティーで指定した期限が切れると効力を失います。

1 ツリーコンポーネントで、「設定」ノードを展開し、「**Java** メッセージサービス」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 「物理送信先」ノードを選択します。

4 「物理送信先」ページで、「新規」をクリックします。
「物理転送先の作成」ページが表示されます。

5 「物理送信先名」フィールドに、送信先の名前 (PhysicalQueue など) を入力します。

6 「タイプ」ドロップダウンリストから、topic または queue を選択します。

7 「追加プロパティー」セクションで、「プロパティーを追加」をクリックしてプロパティーを追加します。
次の表に、現在使用可能な 1 つのプロパティーを示します。

プロパティー名	説明
maxNumActiveConsumers	キュー送信先からの負荷分散された配信でアクティブ化できるコンシューマの最大数。値 -1 は、この数が無制限であることを示します。デフォルトは、送信先がスタンドアロンサーバーインスタンスに対して作成される場合は 1 であり、クラスタに対して作成される場合は -1 です。

このプロパティーの値を変更するか、またはほかの物理送信先プロパティーを指定するには、*MQ-install-dir*/imq/bin/imqcmd コマンドを使用します。詳細については、『Message Queue 管理ガイド』を参照してください。

8 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

create-jmsdest

システムの送信先

「物理送信先」ページに、有効期限が切れて配信不可能なメッセージをリダイレクトするシステムの送信先 (mq.sys.dmq という名前のキュー) が表示されます。この送信先に対して、送信先リソース、コンシューマ、およびブラウザを作成できます。この送信先を削除したり、メッセージを送信したりすることはできません。

▼ JMS 物理送信先を削除する

1 ツリーコンポーネントで、「設定」ノードを展開し、「Java メッセージサービス」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- default-config のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、default-config ノードを選択します。

3 「物理送信先」ノードを選択します。

4 「物理送信先」ページで、削除する送信先名のとなりのチェックボックスにチェックマークを付けます。

5 「削除」をクリックします。
システムの送信先 mq.sys.dmq を削除しようとすると、エラーメッセージが表示されます。

参考 同機能を持つ asadmin コマンド

delete-jmsdest

JMS プロバイダに関する管理コンソールタスク

▼ JMS プロバイダの一般プロパティーを設定する

「JMS サービス」ページを使用して、すべての JMS 接続で使用するプロパティーを設定します。

JMS サービスの設定の詳細については、『Application Server 開発者ガイド』を参照してください。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。

- `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 「**Java** メッセージサービス」ノードを選択して、「**JMS** サービス」ページを開きます。

4 起動が中止されないように **JMS** サービスが開始するのを **Application Server** が待機する時間を変更するには、「起動時のタイムアウト」フィールドの値を編集します。

処理速度の遅いシステムやオーバーロードしたシステムでは、デフォルト値 (60) を大きくします。

5 「タイプ」ドロップダウンリストから、次のとおり選択します。

- ローカルホストの JMS サービスにアクセスするには、`LOCAL` (`server-config` 設定のデフォルト) を選択します。JMS サービスは、Application Server によって起動および管理されます。
- ほかのシステムまたはクラスタの JMS サービスにアクセスするには、`REMOTE` (`default-config` 設定のデフォルト) を選択します。`REMOTE` を選択すると、JMS サービスは次のサーバーの起動時に Application Server によって起動されません。その代わり、JMS サービスは Message Queue によって起動および管理されるため、Message Queue ブローカを別に起動する必要があります。ブローカの起動については、『*Message Queue* 管理ガイド』を参照してください。この値を選択し、かつリモートホストを使用している場合は、116 ページの「JMS ホストを編集する」で説明する手順に従って、リモートホストの名前を指定します。

6 「起動引数」フィールドに、**JMS** サービスの起動をカスタマイズする引数を入力します。

MQ-install-dir`/imq/bin/imqbrokerd` コマンドで使用できる任意の引数を使用します。

7 「再接続」チェックボックスを使用して、接続が失われたときに **JMS** サービスがメッセージサーバーまたは **AddressList** で指定したアドレスのリストに再接続を試みるように指定します。

デフォルトで、再接続は有効です。

8 「再接続の間隔」フィールドに、再接続を試行する間隔を秒数で入力します。

この間隔は、AddressList で指定した各アドレスおよびリストの次のアドレスへの試行に適用されます。間隔が短すぎると、ブローカにリカバリする時間が与えられません。間隔が長すぎると、再接続が許容できない遅延を示す場合があります。

デフォルト値は 60 秒です。

9 「再接続の試行」フィールドで、クライアントランタイムがリストの次のアドレスを試行する前に、**AddressList** に指定した各アドレスへの接続 **(**または再接続**)** を試行する回数を入力します。

値 -1 は、再試行回数が無制限であることを示します。クライアントランタイムは、接続が成功するまで最初のアドレスへの接続を試みます。デフォルト値は 3 です。

10 「デフォルト **JMS** ホスト」ドロップダウンリストからホストを選択します。デフォルトは `default_JMS_host` です。

11 「アドレスリストの動作」ドロップダウンリストで、接続の試行を **AddressList** で指定したアドレスの順序 (`priority`) で行うか、またはランダムな順序 (`random`) で行うかを選択します。

`priority` は、再接続が常に AddressList に指定した最初のサーバーのアドレスへの接続を試行し、最初のブローカが利用できない場合だけにほかのアドレスを使用することを意味します。

多数のクライアントが同じ接続ファクトリを使用して接続を試行する場合は、すべてのクライアントが同じアドレスに接続しないように `random` を指定します。

デフォルトは `random` です。

12 「アドレスリストの繰り返し」フィールドで、接続の確立または再確立のために、**AddressList** を介して **JMS** サービスが反復する回数を入力します。

値 -1 は試行回数が無制限であることを示します。

デフォルト値は 3 です。

13 デフォルト以外のスキームまたはサービスを使用する場合は、「**MQ** スキーム」および「**MQ** サービス」フィールドに、**Message Queue** アドレススキーム名と **Message Queue** 接続サービス名を入力します。

メッセージサービスのアドレスのフル構文は次のとおりです。

scheme`://`*address-syntax*

ここで、*scheme* と *address_syntax* は次の表に示すとおりです。

「MQ スキーム」と「MQ サービス」については、次の表の最初の 2 列に値が表示されています。

スキーム名	接続サービス	説明	アドレス構文
`mq`	`jms` と `ssljms`	Message Queue クライアントランタイムは、指定したホストとポートで MQ ポートマッパーへの接続を確立します。ポートマッパーは動的に確立された接続サービスポートのリストを返し、次に Message Queue クライアントランタイムは指定された接続サービスをホストするポートへの接続を確立します。	[*hostName*][`:` *port*][`/`*serviceName*] デフォルト: *hostName* = `localhost`、*port* = `7676`、*serviceName* = `jms` デフォルトは `jms` 接続サービスだけに適用されます。`ssljms` 接続サービスの場合、すべての変数を指定する必要があります。 例: `mq:MyHost:7677/ssljms`

スキーム名	接続サービス	説明	アドレス構文
mqtcp	jms	Message Queue クライアントランタイムは、MQ ポートマッパーをバイパスして、指定したホストとポートに TCP 接続を確立します。	*hostName*:*port* /jms 例: mqtcp:localhost:7676/jms
mqssl	ssljms	Message Queue クライアントランタイムは、MQ ポートマッパーをバイパスして、指定したホストとポートにセキュリティー保護された SSL 接続を確立します。	*hostName*:*port* /ssljms 例: mqssl:localhost:7676/ssljms
http	httpjms	Message Queue クライアントランタイムは、指定された URL の Message Queue トンネルサーブレットに HTTP 接続を確立します。ブローカは、Message Queue の『管理ガイド』で説明されているとおり、HTTP トンネルサーブレットにアクセスするように設定する必要があります。	*hostName*:*port* /*contextRoot*/tunnel 複数のブローカインスタンスが同じトンネルサーブレットを使用している場合、無作為に選択されたブローカインスタンスではなく、特定のブローカインスタンスに接続するための構文は次のとおりです。http://*hostName*:*port*/*contextRoot* /tunnel?*serverName*=*hostName*:*instanceName*
https	httpsjms	Message Queue クライアントランタイムは、指定された Message Queue トンネルサーブレット URL にセキュリティー保護された HTTPS 接続を確立します。ブローカは、Message Queue の『管理ガイド』で説明されているとおり、HTTPS トンネルサーブレットにアクセスするように設定する必要があります。	*hostName*:*port* /*contextRoot*/tunnel 複数のブローカインスタンスが同じトンネルサーブレットを使用している場合、無作為に選択されたブローカインスタンスではなく、特定のブローカインスタンスに接続するための構文は次のとおりです。http://*hostName*:*port*/*contextRoot* /tunnel?*serverName*=*hostName*:*instanceName*

14 「追加プロパティー」セクションで、「プロパティーを追加」をクリックしてプロパティーを追加します。

次の表に、利用可能な Sun Java System Message Queue ブローカの設定プロパティーを一覧表示します。

プロパティー名	説明
instance-name	完全な Message Queue ブローカインスタンス名を指定します。デフォルトは imqbroker です。
instance-name-suffix	完全な Message Queue ブローカインスタンス名に追加するサフィックスを指定します。サフィックスは、下線文字 (_) によってインスタンス名と区切られます。たとえば、インスタンス名が imqbroker の場合、サフィックス xyz を追加して、インスタンス名を imqbroker_xyz に変更します。

プロパティー名	説明
append-version	true の場合、下線文字 (_) が先行するメジャーバージョンおよびマイナーバージョン番号を完全な Message Queue ブローカインスタンス名に追加します。たとえば、インスタンス名が imqbroker の場合、バージョン番号を追加して、インスタンス名を imqbroker_8_0 に変更します。デフォルトは false です。

15 「保存」をクリックして変更を保存するか、または「デフォルトを読み込み」をクリックしてサービスのデフォルト値を復元します。

参考 リモートサーバーへのアクセス

プロバイダとホストをリモートシステムに変更すると、すべての JMS アプリケーションがリモートサーバーで実行するようになります。ローカルサーバーと 1 つまたは複数のリモートサーバーを使用するには、リモートサーバーにアクセスする接続を作成する AddressList プロパティーを使用して、接続ファクトリリソースを作成します。101 ページの「JMS 接続ファクトリリソースを作成する」を参照してください。

▼ JMS サービスの設定を検証する

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- default-config のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、default-config ノードを選択します。

3 「**Java** メッセージサービス」ノードを選択して、「**JMS** サービス」ページを開きます。

4 「**Ping**」をクリックします。

JMS サービスが稼働している場合は、「Ping が成功しました: JMS サービスは稼働中です」というメッセージが表示されます。

参考 同機能を持つ asadmin コマンド

jms-ping

▼ JMS ホストを作成する

Application Server Platform Edition では、`default_JMS_host` というデフォルト名を持つ JMS ホストが 1 つだけ存在することが想定されています。追加のホストを作成することは可能ですが、Application Server はそれらのホストを認識できません。

1. ツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。
3. 「**Java** メッセージサービス」ノードを展開します。
4. 「**JMS** ホスト」ノードを選択します。
5. 「**JMS** ホスト」ページで、「新規」をクリックします。「**JMS** ホストを作成」ページが表示されます。
6. 「名前」フィールドにホスト名を入力します。次に例を示します。

   ```
   NewJmsHost
   ```

7. 「ホスト」フィールドに、**JMS** ホストを実行するシステムの名前 (`localhost` またはローカルあるいはリモートシステムの名前) または **IP (Internet Protocol)** アドレスを入力します。
8. 「ポート」フィールドに、**JMS** サービスのポート番号を入力します。

   使用する JMS サービスをデフォルト以外のポートで実行する場合にのみ、このフィールドを変更してください。デフォルトのポートは 7676 です。
9. 「管理ユーザー名」フィールドと「管理パスワード」フィールドに、**Message Queue** ブローカのユーザー名およびパスワードを入力します。

   これらは、Application Server のユーザー名およびパスワードとは異なります。これらのフィールドを編集するのは、*MQ-install-dir*`/imq/bin/imqusermgr` コマンドを使って MQ ブローカの値を変更した場合に限ります。デフォルト値は `admin` と `admin` です。
10. 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-jms-host`

▼ JMS ホストを編集する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- default-config のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、default-config ノードを選択します。

3 「Java メッセージサービス」ノードを展開します。

4 「JMS ホスト」ノードを選択します。

5 「JMS ホスト」ページで、編集するホストを選択します。

6 「JMS ホストを編集」ページで、次のタスクを実行できます。

- 「ホスト」フィールドでの、ホスト名または IP (Internet Protocol) アドレスの変更。
- 「ポート」フィールドでの、JMS サービスのポート番号の変更。
- 「管理ユーザー名」フィールドと「管理パスワード」フィールドの値の変更。

7 「保存」をクリックして変更を保存するか、または「デフォルトを読み込み」をクリックしてホストのデフォルト値を復元します。

▼ JMS ホストを削除する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- default-config のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、default-config ノードを選択します。

3 「Java メッセージサービス」ノードを展開します。

4 「**JMS** ホスト」ノードを選択します。

5 「**JMS** ホスト」ページで、削除するホストのとなりのチェックボックスにチェックマークを付けます。

6 「削除」をクリックします。

注意 – JMS ホストを全部は削除しないでください。全部削除すると、Application Server が再起動しなくなってしまいます。少なくとも 1 つの JMS ホストを残しておく必要があります。

参考 同機能を持つ asadmin コマンド

`delete-jms-host`

◆ ◆ ◆ 第 5 章

JavaMail リソースの設定

この章では、JavaMail API を使用するアプリケーションのリソースを設定する方法について説明します。この章には次の節が含まれています。

JavaMail について

JavaMail API はメールシステムをモデル化する一連の抽象 API です。この API は、メールおよびメッセージングアプリケーションを構築するための、プラットフォームにもプロトコルにも依存しないフレームワークを提供します。JavaMail API では電子メールの送受信機能が提供されます。サービスプロバイダは特定のプロトコルを実装します。

JavaMail API は、Java プラットフォームのオプションパッケージとして実装され、また J2EE プラットフォームの一部としても利用できます。

Application Server には、JavaMail API とともに、アプリケーションコンポーネントがインターネットを介して電子メール通知を送信したり IMAP や POP3 メールサーバーからの電子メールを読んだりするための JavaMail サービスプロバイダが含まれています。

JavaMail API の詳細については、JavaMail Web サイト (`http://java.sun.com/products/javamail/`) を参照してください。

JavaMail に関する管理コンソールタスク

▼ JavaMail セッションを作成する

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「**JavaMail** セッション」ノードを選択します。

2 「**JavaMail** セッション」ページで、「新規」をクリックします。
「JavaMail セッションを作成」ページが表示されます。

3 「**JNDI** 名」フィールドに、セッション名を入力します。
JavaMail リソースのネーミングサブコンテキストプレフィックス `mail/` を使用することをお勧めします。次に例を示します。`mail/MySession`

4 「メールホスト」フィールドに、デフォルトメールサーバーのホスト名を入力します。
プロトコル固有のホストプロパティーが提供されていない場合、Store オブジェクトと Transport オブジェクトの接続メソッドはこの値を使用します。この名前は実際のホスト名として解決可能でなければいけません。

5 「デフォルトユーザー」フィールドで、メールサーバーへの接続時に渡すユーザー名を入力します。
プロトコル固有の username プロパティーが提供されていない場合、Store オブジェクトと Transport オブジェクトの接続メソッドはこの値を使用します。

6 「デフォルトの返信用アドレス」フィールドで、デフォルトユーザーの電子メールアドレスを *username*@*host*.*domain* の形式で入力します。

7 このときメールセッションを有効にしない場合は、「有効」チェックボックスを選択解除します。

8 デフォルト以外のストアやトランスポートプロトコルを使用するように **Application Server** のメールプロバイダを設定し直した場合にのみ、「詳細」フィールドでフィールド値を変更します。
デフォルトで、ストアプロトコルは `imap`、ストアプロトコルクラスは `com.sun.mail.imap.IMAPStore`、トランスポートプロトコルは `smtp`、トランスポートプロトコルクラスは `com.sun.mail.smtp.SMTPTransport` になっています。

9 このメールセッションのプロトコルトレースなど、ほかのデバッグ出力を有効にするには、「デバッグ」チェックボックスにチェックマークを付けます。
JavaMail のログレベルを `FINE` またはそれ以上に設定した場合、デバッグ出力が生成され、システムのログファイルに取り込まれます。ログレベルの設定の詳細については、273 ページの「ログレベルを設定する」を参照してください。

10 プロトコル固有のホストや **username** プロパティーなど、アプリケーションが必要とするプロパティーを追加するには、「追加プロパティー」フィールドで「プロパティーを追加」をクリックします。

JavaMail API マニュアルには、使用可能なプロパティーのリストがあります (`http://java.sun.com/products/javamail/javadocs/index.html`)。

11 「ターゲット」領域で、次を実行します。

a. 「選択可能」リストから、リソースを使用するアプリケーションが配備されるターゲットを選択します。

選択可能なターゲットには、利用可能なクラスタおよびサーバーインスタンスと、デフォルトのサーバーインスタンス `server` が含まれます。

b. 「追加」をクリックして、ターゲットを「選択」リストに移動します。

12 「了解」をクリックして、セッションを保存します。

参考 同機能を持つ asadmin コマンド

`create-javamail-resource`

▼ JavaMail セッションを編集する

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「**JavaMail** セッション」ノードを選択します。

2 「**JavaMail** セッション」ページで、編集するセッションを選択します。

3 「**JavaMail** セッションを編集」ページで、次のタスクを実行できます。

- 「メールホスト」、「デフォルトユーザー」、「デフォルトの返信用アドレス」、および「説明」フィールドの値の変更。
- 「有効」チェックボックスの選択または選択解除による、リソースの有効化または無効化。
- 「詳細」フィールドの値の変更。
- プロパティーの追加、削除、または変更。

4 「ターゲット」タブをクリックして、「**JavaMail** セッションターゲット」ページを表示します。このページで、次を実行します。

   a. 「ターゲットの管理」をクリックして、「リソースターゲットの管理」ページを開きます。

   このページで、「選択可能」列と「選択」列の間でターゲットを移動します。「選択」列に、リソースを使用するアプリケーションが配備されるターゲットを必ず配置します。選択可能なターゲットには、利用可能なクラスタおよびサーバーインスタンスと、デフォルトのサーバーインスタンス `server` が含まれます。「了解」をクリックして変更を保存します。

   b. ターゲットのチェックボックスを選択して、「有効」または「無効」をクリックし、そのターゲットのリソースを有効または無効にします。

5 「保存」をクリックして変更を保存するか、または「デフォルトを読み込み」をクリックしてメールセッションのデフォルト値を復元します。

▼ JavaMail セッションを削除する

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「**JavaMail** セッション」ノードを選択します。

2 「**JavaMail** セッション」ページで、削除するセッション名のとなりのチェックボックスにチェックマークを付けます。

3 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-javamail-resource`

◆ ◆ ◆ 第 6 章

JNDI リソース

Java Naming and Directory Interface (JNDI) について

この節では、Java Naming and Directory Interface (JNDI) について説明します。JNDI は、さまざまな種類のネーミングおよびディレクトリサービスにアクセスするための API (Application Programming Interface) です。J2EE コンポーネントは、JNDI 検索メソッドを起動することによってオブジェクトを検出します。

この節では、次の項目について説明します。

JNDI 名とリソース

JNDI は、Java Naming and Directory Interface API の略語です。API を呼び出すことにより、アプリケーションはリソースとほかのプログラムオブジェクトを検出します。リソースとは、データベースサーバーやメッセージングシステムなどのシステムへの接続を提供するプログラムオブジェクトです。JDBC リソースはデータソースと呼ばれる場合もあります。それぞれのリソースオブジェクトは人間が理解しやすい JNDI 名という一意の名前で識別されます。リソースオブジェクトと JNDI 名は、Application Server に含まれているネーミングおよびディレクトリサービスによって相互にバインドされています。新しいリソースを作成すると、JNDI に新しい名前とオブジェクトのバインドが入力されます。

J2EE ネームサービス

JNDI 名は人間が理解しやすいオブジェクトの名前です。これらの名前は、J2EE サーバーが提供するネームサービスとディレクトリサービスによってオブジェクトにバインドされます。J2EE コンポーネントは JNDI API を介してこのサービスにアクセスするので、通常オブジェクトはその JNDI 名を使用します。たとえば、PointBase データベースの JNDI 名は `jdbc/Pointbase` となります。Application Server は、起動時に設定ファイルから情報を読み込み、JNDI データベース名を自動的に名前空間に追加します。

J2EE アプリケーションクライアント、Enterprise JavaBeans、および Web コンポーネントは、JNDI ネーミング環境にアクセスする必要があります。

アプリケーションコンポーネントのネーミング環境は、配備またはアセンブリの際に、アプリケーションコンポーネントのビジネスロジックのカスタマイズを可能にするメカニズムです。このアプリケーションコンポーネントの環境を使用することにより、アプリケーションコンポーネントのソースコードにアクセスしたり、このソースコードを変更したりせずに、アプリケーションコンポーネントをカスタマイズできます。

J2EE コンテナはアプリケーションコンポーネントの環境を実装し、この環境をアプリケーションコンポーネントのインスタンスに JNDI ネーミングコンテキストとして提供します。アプリケーションコンポーネントの環境は次のとおり使用されます。

- アプリケーションコンポーネントのビジネスメソッドは JNDI インタフェースを使用して環境にアクセスします。アプリケーションコンポーネントプロバイダは、実行時にその環境に用意されるとアプリケーションコンポーネントが想定する、環境エントリのすべてを配備記述子で宣言します。
- コンテナは、アプリケーションコンポーネントの環境を格納する JNDI ネーミングコンテキストの実装を提供します。また、コンテナは、配備担当者が各アプリケーションコンポーネントの環境を作成し、管理するツールも提供します。
- 配備担当者は、コンテナが提供するツールを使用して、アプリケーションコンポーネントの配備記述子で宣言された環境エントリを初期化します。配備担当者は環境エントリの値を設定し、変更します。
- コンテナは、実行時にアプリケーションコンポーネントのインスタンスで、環境ネーミングコンテキストを利用できるようにします。アプリケーションコンポーネントのインスタンスは、JNDI インタフェースを使用して環境エントリの値を取得します。

各アプリケーションコンポーネントは、自身の一連の環境エントリを定義します。同じコンテナ内のすべてのアプリケーションコンポーネントのインスタンスは、同じ環境エントリを共有します。アプリケーションコンポーネントのインスタンスは、実行時に環境を変更することはできません。

ネーミング参照とバインディング情報

リソース参照は、リソース用にコード化されたコンポーネントの名前を識別する配備記述子の要素です。具体的には、コード化された名前はリソースの接続ファクトリを参照します。次の節で説明する例では、リソース参照名は `jdbc/SavingsAccountDB` です。

リソースの JNDI 名とリソース参照名とは同じではありません。このネーミングへのアプローチでは、配備前に 2 つの名前をマップする必要がありますが、同時にコンポーネントをリソースから分離します。この分離により、あとでコンポーネントが別のリソースにアクセスする必要があっても、名前を変更する必要がなくなります。この柔軟性により、既存のコンポーネントから J2EE アプリケーションを簡単にアセンブルすることが可能になります。

次の表には、Application Server が使用する J2EE リソースの JNDI 検索と関連する参照が一覧表示されています。

表 6–1 JNDI 検索と関連する参照

JNDI 検索名	関連する参照
java:comp/env	アプリケーション環境エントリ
java:comp/env/jdbc	JDBC データソースリソースマネージャー接続ファクトリ
java:comp/env/ejb	EJB 参照
java:comp/UserTransaction	UserTransaction 参照
java:comp/env/mail	JavaMail セッション接続ファクトリ
java:comp/env/url	URL 接続ファクトリ
java:comp/env/jms	JMS 接続ファクトリと送信先
java:comp/ORB	アプリケーションコンポーネント間で共有された ORB インスタンス

カスタムリソースについて

カスタムリソースの使用

カスタムリソースはローカルの JNDI リポジトリにアクセスし、外部リソースは外部 JNDI リポジトリにアクセスします。両方のリソースのタイプが、ユーザー指定のファクトリクラス要素、JNDI 名属性などを必要とします。この節では、J2EE リソースの JNDI 接続ファクトリリソースを設定し、これらのリソースにアクセスする方法を説明します。

Application Server では、`list-jndi-entities` と同様に、リソースを作成、削除、一覧表示することができます。

▼ カスタムリソースを作成する

1. 管理コンソールの左側の区画で、**JNDI** 設定を変更する **Application Server** インスタンスを開きます。

2. 「**JNDI**」タブを開き、「カスタムリソース」をクリックします。
   カスタムリソースがすでに作成されている場合、それらが右側の区画に表示されます。新しいカスタムリソースを作成するには、「新規」をクリックします。「JNDI」タブを開き、「新規」をクリックします。新しいカスタムリソースを追加するページが表示されます。

3. 「**JNDI** 名」フィールドで、リソースへのアクセスに使用する名前を入力します。
   この名前は JNDI ネームサービスに登録されます。

4. 「リソースタイプ」フィールドで、上記の例に示すとおり完全修飾タイプの定義を入力します。
   リソースタイプの定義は、`xxx.xxx` の形式に従います。

5. 「ファクトリクラス」フィールドで、作成するカスタムリソースのファクトリクラス名を入力します。
   「ファクトリクラス」はファクトリクラスのユーザー指定の名前です。このクラスは `javax.naming.spi.ObjectFactory` インタフェースを実装します。

6. 「説明」フィールドで、作成するリソースの説明を入力します。
   この説明は文字列値で、最大 250 文字を入力することができます。

7. 「追加プロパティー」セクションで、プロパティー名およびプロパティー値を追加します。

8. 「カスタムリソースを有効」チェックボックスにチェックマークを付けて、カスタムリソースを有効にします。

9. 「了解」をクリックして、カスタムリソースを保存します。
   クラスタまたはスタンドアロンインスタンスにカスタムリソースが配備されている場合、「ターゲット」タブを使用してターゲットを管理できます。「ターゲット」タブは、カスタムリソースの作成後に表示されます。ターゲット名を入力して「了解」をクリックし、ターゲットを設定します。

参考 同機能を持つ asadmin コマンド

`create-custom-resource`

▼ カスタムリソースを編集する

1. 管理コンソールの左側の区画で、**JNDI** 設定を変更する **Application Server** インスタンスを開きます。
2. 「**JNDI**」を開き、「カスタムリソース」を選択します。
   カスタムリソースがすでに作成されている場合、それらが右側の区画に表示されます。
3. 右側の区画のファイル名をクリックします。
4. 「リソースタイプ」フィールド、「ファクトリクラス」フィールド、または「説明」フィールドを編集します。
5. 「カスタムリソースを有効」チェックボックスにチェックマークを付けて、カスタムリソースを有効にします。
6. 「保存」をクリックして、カスタムリソースの変更を保存します。

▼ カスタムリソースを削除する

1. 管理コンソールの左側の区画で、「**JNDI**」タブを開きます。
2. 「カスタムリソース」をクリックします。
   カスタムリソースがすでに作成されている場合、それらが右側の区画に表示されます。
3. 削除するリソース名の横にあるボックスをクリックします。
4. 「削除」をクリックします。カスタムリソースが削除されます。

参考 同機能を持つ asadmin コマンド

`delete-custom-resource`

カスタムリソースの一覧表示

カスタムリソースを一覧表示するには、`asadmin list-custom-resources` コマンドを入力します。たとえば、ホスト `plum` 上のカスタムリソースを一覧表示するには、次のように入力します。

```
$asadmin list-custom-resources --host plum target6
```

コマンドの完全なコンテキストを確認するには、`asadmin help list-custom-resources` と入力してください。

外部 JNDI リポジトリおよびリソースについて

外部 JNDI リポジトリおよびリソースの使用

通常、Application Server で実行中のアプリケーションは、外部 JNDI リポジトリに格納されているリソースにアクセスする必要があります。たとえば、一般的な Java オブジェクトは、Java スキーマのように LDAP サーバーに格納できます。外部 JNDI リソースの要素を使用すると、このような外部リソースリポジトリを設定できます。外部 JNDI ファクトリは、`javax.naming.spi.InitialContextFactory` インタフェースを実装する必要があります。

外部 JNDI リソースの使用例を示します。

```
<resources>
 <!-- external-jndi-resource 要素は、外部 JNDI リポジトリに格納されて
-- いる J2EE リソースへのアクセス方法を指定します。次の例は、
-- LDAP に格納されている Java オブジェクトへのアクセス方法を示します。
-- factory-class 要素は、リソースファクトリへのアクセスに使用される
-- JNDI InitialContext ファクトリを指定します。property 要素は
-- 外部 JNDI コンテキストに適用可能な環境に対応します。
-- jndi-lookup-name は、指定の (この例では Java) オブジェクトを検出して
-- フェッチするための JNDI 名を参照します。
 -->
  <external-jndi-resource jndi-name="test/myBean"
      jndi-lookup-name="cn=myBean"
      res-type="test.myBean"
      factory-class="com.sun.jndi.ldap.LdapCtxFactory">
    <property name="PROVIDER-URL" value="ldap://ldapserver:389/o=myObjects" />
    <property name="SECURITY_AUTHENTICATION" value="simple" />
    <property name="SECURITY_PRINCIPAL", value="cn=joeSmith, o=Engineering" />
    <property name="SECURITY_CREDENTIALS" value="changeit" />
  </external-jndi-resource>
</resources>
```

▼ 外部リソースを作成する

1. 管理コンソールの左側の区画で、**JNDI** 設定を変更する **Application Server** インスタンスを開きます。
2. 「**JNDI**」を開き、「外部リソース」を選択します。
   外部リソースがすでに作成されている場合、それらが右側の区画に表示されます。
3. 新しい外部リソースを作成するには、「新規」をクリックします。
4. 「**JNDI** 名」フィールドで、リソースへのアクセスに使用する名前を入力します。
   この名前は JNDI ネームサービスに登録されます。
5. 「リソースタイプ」フィールドで、上記の例に示すとおり完全修飾タイプの定義を入力します。
   リソースタイプの定義は、`xxx.xxx` の形式に従います。
6. 「**JNDI** 検索」フィールドで、外部リポジトリを検索する **JNDI** 値を入力します。
   たとえば、外部リポジトリに接続し、Bean クラスをテストする外部リソースを作成する場合、「JNDI 検索」は次のようになります。`cn=`*testmybean*
7. 「ファクトリクラス」フィールドで、**JNDI** ファクトリクラスの外部リポジトリを `com.sun.jndi.ldap` のように入力します。
   このクラスは `javax.naming.spi.ObjectFactory` インタフェースを実装します。
8. 「説明」フィールドには、作成するリソースの説明を入力します。
   この説明は文字列値で、最大 250 文字を入力することができます。
9. 「追加プロパティー」セクションで、プロパティー名およびプロパティー値を追加します。
10. 「外部リソースを有効」チェックボックスにチェックマークを付けて、外部リソースを有効にします。
11. 「了解」をクリックして、外部リソースを保存します。
    クラスタまたはスタンドアロンインスタンスに外部リソースが配備されている場合、「ターゲット」タブを使用してターゲットを管理できます。「ターゲット」タブは、外部リソースの作成後に表示されます。ターゲット名を入力して「了解」をクリックし、ターゲットを設定します。

参考 同機能を持つ asadmin コマンド

`create-jndi-resource`

▼ 外部リソースを編集する

1. 管理コンソールの左側の区画で、**JNDI** 設定を変更する **Application Server** インスタンスを開きます。
2. 「**JNDI**」を開き、「外部リソース」を選択します。
   外部リソースがすでに作成されている場合、それらが右側の区画に表示されます。
3. 外部リソースを編集するには、右側の区画のファイル名をクリックします。
4. 「リソースタイプ」フィールド、「**JNDI** 検索」フィールド、「ファクトリクラス」フィールド、または「説明」フィールドを編集します。
5. 「外部リソースを有効」チェックボックスにチェックマークを付けて、外部リソースを有効にします。
6. 「保存」をクリックして、外部リソースの変更を保存します。

▼ 外部リソースを削除する

1. 管理コンソールの左側の区画で、「**JNDI**」タブを開きます。
2. 「外部リソース」をクリックします。
   外部リソースがすでに作成されている場合、それらが右側の区画に表示されます。
3. 削除するリソース名の横にあるボックスをクリックします。
4. 「削除」をクリックします。外部リソースが削除されます。

参考 同機能を持つ asadmin コマンド

`delete-jndi-resource`

外部リソースの一覧表示

外部リソースを一覧表示するには、`asadmin list-jndi-resources` コマンドを入力して JNDI 名を指定します。たとえば、外部リソースを一覧表示するには、次のように入力します。

```
$asadmin list-jndi-resources --user adminuser --host plum jndi_name_test
```

コマンドの完全なコンテキストを確認するには、`asadmin help list-jndi-resources` と入力してください。

◆ ◆ ◆ 第 7 章

7 コネクタリソース

この章では、エンタープライズ情報システム (EIS) へのアクセスに使用されるコネクタの設定方法について説明します。この章には次の節が含まれています。

コネクタについて

コネクタモジュールとは、アプリケーションがエンタープライズ情報システム (EIS) と対話することを可能にする J2EE コンポーネントであり、リソースアダプタとも呼ばれます。EIS ソフトウェアにはさまざまな種類のシステムが含まれています。ERP (Enterprise Resource Planning)、メインフレームトランザクション処理、非リレーショナルデータベースなどです。ほかの J2EE モジュールと同様に、コネクタモジュールをインストールするには、これを配備する必要があります。

コネクタ接続プールとは、特定の EIS のための再利用可能な接続のグループです。コネクタ接続プールを作成するには、プールに関連付けられたコネクタモジュール (リソースアダプタ) を指定します。

コネクタリソースとは、アプリケーションに EIS への接続を提供するプログラムオブジェクトです。コネクタリソースを作成するには、JNDI 名と関連する接続プールを指定します。複数のコネクタリソースで 1 つの接続プールを指定できます。アプリケーションはリソースの JNDI 名を検索してその位置を確定します。JNDI の詳細については、「JNDI 名とリソース」の節を参照してください。EIS 用コネクタリソースの JNDI 名は、通常 `java:comp/env/`*`eis-specific`* サブコンテキストにあります。

Application Server は、コネクタモジュール (リソースアダプタ) を使って JMS を実装します。「JMS リソースとコネクタリソースの関係」を参照してください。

コネクタ接続プールに関する管理コンソールタスク

▼ EIS アクセスを設定する

1. コネクタを配備 (インストール) します。**69** ページの「コネクタモジュールを配備する」を参照してください。
2. コネクタの接続プールを作成します。**132** ページの「コネクタ接続プールを作成する」を参照してください。
3. 接続プールに関連付けられたコネクタリソースを作成します。**135** ページの「コネクタリソースを作成する」を参照してください。

▼ コネクタ接続プールを作成する

始める前に　プールを作成する前に、プールに関連付けられたコネクタモジュール (リソースアダプタ) を配備します。新しいプールに指定する値は、配備したコネクタモジュールにより異なります。

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。
2. 「コネクタ接続プール」ノードを選択します。
3. 「コネクタ接続プール」ページで、「新規」をクリックします。
4. 最初の「コネクタ接続プールを作成」ページで、次の設定値を指定します。
   a. 「名前」フィールドで、プールの論理名を入力します。
      コネクタリソースの作成時にこの名前を指定します。
   b. 「リソースアダプタ」コンボボックスからエントリを選択します。
      コンボボックスには配備したリソースアダプタ (コネクタモジュール) のリストが表示されます。
5. 「次へ」をクリックします。

6 次の「コネクタ接続プールを作成」ページで、「接続の定義」コンボボックスから値を選択します。

コンボボックスの選択はリソースアダプタにより異なります。通常は、ConnectionFactory のタイプとして EIS に接続するファクトリインスタンスを指定します。

7 「次へ」をクリックします。

8 最後に 3 番目の「コネクタ接続プールを作成」ページで、次のタスクを実行します。

a. 「一般設定」セクションでその値が正しいことを確認します。

b. 「プール設定」セクションのフィールドに、デフォルト値を保持できます。

これらの設定はあとで変更できます。「コネクタ接続プールの編集」を参照してください。

c. 「追加プロパティー」テーブルで、必要なプロパティーを追加します。

前の「コネクタ接続プールを作成」ページでは、「接続の定義」コンボボックスから値を選択しました。このクラスはサーバーのクラスパス内にあり、「追加プロパティー」テーブルにはデフォルトプロパティーが表示されます。

9 「完了」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-connector-connection-pool`

▼ コネクタ接続プールを編集する

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。

2 「コネクタ接続プール」ノードを開きます。

3 編集するプールのノードを選択します。

4 「コネクタ接続プールを編集」ページで、プールに含まれる接続の数を制御する設定を変更できます。次の表を参照してください。

パラメータ	説明
初期および最小プールサイズ	プール内の接続の最小数。この値は、プールを最初に作成したり、Application Server を起動したりするときの、プールに含まれる接続の数も指定します。
最大プールサイズ	プールに含まれる接続の最大数。
プールサイズ変更量	プールのサイズが最小プールサイズに近づくと、プールサイズが一括処理で変更されます。この値は一括処理での接続の数を指定します。過大な値を設定すると接続の再利用が遅れ、過小な値を設定すると効率が落ちます。
アイドルタイムアウト	プールで接続がアイドル状態のままでいられる最長時間を指定します。この時間を過ぎると、接続はプールから削除されます。
最大待ち時間	接続タイムアウトになる前に接続を要求したアプリケーションが待つ時間。デフォルトの待ち時間は長いので、アプリケーションがハングアップしているように見える可能性があります。
すべて障害で	「すべての接続を閉じる」のチェックボックスを選択してある場合、1 つの接続が失敗すると、アプリケーションサーバーはプールに含まれるすべての接続を閉じ、それらを再確立します。そのチェックボックスを選択していない場合は、使用されるときにだけ個々の接続が再確立されます。
トランザクションサポート	トランザクションサポートのリストを使用して、接続プールのトランザクションサポートのタイプを選択します。選択したトランザクションサポートは、この接続プールに下位互換性があるように関連付けられたリソースアダプタのトランザクションサポート属性をオーバーライドします。つまり、リソースアダプタに指定したレベルより低いトランザクションレベルまたはリソースアダプタに指定したレベルと同じトランザクションレベルをサポートし、それより高いレベルを指定することはできません。 トランザクションサポートオプションには次のものが含まれます。 「トランザクションサポート」メニューから「なし」を選択すると、リソースアダプタがローカルのリソースマネージャーまたは JTA トランザクションをサポートせず、`XAResource` または `LocalTransaction` インタフェースを実装しないことを示します。 ローカルトランザクションサポートは、リソースアダプタが `LocalTransaction` インタフェースを実装することにより、ローカルトランザクションをサポートすることを意味します。ローカルトランザクションはリソースマネージャーの内部で管理され、外部トランザクションマネージャーは一切関与しません。 XA トランザクションサポートは、リソースアダプタが `LocalTransaction` および `XAResource` インタフェースを実装することにより、ローカルのリソースマネージャーと JTA トランザクションをサポートすることを意味します。XA トランザクションは、リソースマネージャーの外部にあるトランザクションマネージャーによって制御され、協調動作します。ローカルトランザクションはリソースマネージャーの内部で管理され、外部トランザクションマネージャーは一切関与しません。

5 「追加プロパティー」テーブルで、名前と値のペアを指定します。

指定するプロパティーは、プールが使用するリソースアダプタにより異なります。配備担当者がこのテーブルを使用して指定した名前と値のペアは、リソースアダプタベンダーが定義したプロパティーのデフォルト値をオーバーライドするために使用できます。

6 「セキュリティーマップ」タブ付き区画で、接続プールのセキュリティーマップを作成または変更します。

セキュリティーマップの作成方法の詳細については、198 ページの「セキュリティーマップについて」を参照してください。

7 「保存」をクリックします。

▼ コネクタ接続プールを削除する

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。

2 「コネクタ接続プール」ノードを選択します。

3 「コネクタ接続プール」ページで、削除するプールのチェックボックスにチェックマークを付けます。

4 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-connector-connection-pool`

コネクタリソースに関する管理コンソールタスク

▼ コネクタリソースを作成する

コネクタリソース (データソース) はアプリケーションに EIS への接続を提供します。

始める前に コネクタリソースを作成する前に、まずコネクタ接続プールを作成します。

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。
2. 「コネクタリソース」ノードを開きます。
3. 「コネクタリソース」ページで、「新規」をクリックします。
4. 「コネクタリソースを作成」ページで、リソースの設定を指定します。
   a. 「**JNDI** 名」フィールドに、一意の名前を入力します。次に例を示します。`eis/myERP`。
      スラッシュを忘れないでください。
   b. 「プール名」コンボボックスから、新しいコネクタリソースが属する接続プールを選択します。
   c. リソースを利用不可に変更するには、「すべてのターゲットを無効」ラジオボタンをオンにします。
      デフォルトでは、リソースは作成すると同時に利用可能 (有効) です。
   d. このページの「ターゲット」セクションで、「選択可能」フィールドからコネクタリソースが配置されるドメイン、クラスタ、またはサーバーインスタンスを選択して「追加」をクリックします。
      「選択」フィールドに一覧表示されたドメイン、クラスタ、またはサーバーインスタンスのいずれかにコネクタリソースを配備しない場合は、同フィールドから該当するものを選択して「削除」をクリックします。
5. 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-connector-resource`

▼ コネクタリソースを編集する

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。
2. 「コネクタリソース」ノードを開きます。
3. 編集するコネクタリソースのノードを選択します。

4 「コネクタリソースを編集」ページで、「プール名」メニューから別の接続プールを選択することができます。

5 「ターゲット」タブ付き区画で、「ターゲットの管理」をクリックして、コネクタリソースが配備されているターゲットを編集することができます。
ターゲットの詳細については、135 ページの「コネクタリソースを作成する」を参照してください。

6 「保存」をクリックして、編集を適用します。

▼ コネクタリソースの削除

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。

2 「コネクタリソース」ノードを選択します。

3 「コネクタリソース」ページで、削除するリソースのチェックボックスにチェックマークを付けます。

4 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-connector-resource`

▼ コネクタサービスを設定する

「コネクタサービス」画面を使用して、このクラスタまたはサーバーインスタンスに配備されたすべてのリソースアダプタのコネクタコンテナを設定します。

1 ツリーから「設定」を選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス **server** の場合は、**server-config** ノードを選択します。
- **default-config** のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、**default-config** ノードを選択します。

3 「コネクタサービス」ノードを選択します。

4 「シャットダウンタイムアウト」フィールドで、シャットダウンのタイムアウトを秒単位で指定します。

アプリケーションサーバーがコネクタモジュールのインスタンスの `ResourceAdapter.stop` メソッドの完了を待機する秒数 (整数) を入力します。指定したシャットダウンタイムアウトよりも長い時間がかかるリソースアダプタはアプリケーションサーバーに無視され、シャットダウン手順が続行されます。デフォルトのシャットダウンタイムアウトは 30 秒です。「デフォルトを読み込み」をクリックして、このクラスタまたはサーバーインスタンスに配備されたリソースアダプタのデフォルトのシャットダウンタイムアウトを選択します。

管理対象オブジェクトリソースに関する管理コンソールタスク

▼ 管理対象オブジェクトリソースを作成する

リソースアダプタ (コネクタモジュール) にパッケージ化されている管理対象オブジェクトは、アプリケーションの特殊な機能を提供します。たとえば、管理対象オブジェクトは、リソースアダプタおよびそれに関連付けられた EIS に固有なパーサーへのアクセスを提供できます。オブジェクトは管理対象に指定することができます。つまり、管理者により設定可能です。オブジェクトを設定するには、「作成」または「管理オブジェクトリソースを編集」ページで、名前と値のプロパティーペアを追加します。管理対象オブジェクトリソースを作成すると、その管理対象オブジェクトが JNDI 名に関連付けられます。

Application Server はリソースアダプタを使って JMS を実装します。作成する各 JMS 送信先に対して、Application Server は管理対象オブジェクトリソースを自動的に作成します。

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。

2 「管理オブジェクトリソース」ノードを開きます。

3 「コネクタリソース」ページで、「新規」をクリックします。

4 「管理オブジェクトリソース」ページで、次の設定を指定します。

a. 「**JNDI** 名」フィールドで、リソースを識別する一意の名前を入力します。

b. 「リソースタイプ」フィールドで、リソースの **Java** タイプを入力します。

c. 「リソースアダプタ」コンボボックスから、管理対象オブジェクトが含まれるリソースアダプタを選択します。

d. チェックボックスを選択または選択解除して、リソースを有効または無効にします。

e. 「次へ」をクリックします。

5 次の「管理オブジェクトリソースを作成」ページで、次のタスクを実行できます。

a. 名前と値のプロパティーペアで管理対象オブジェクトを設定するには、「プロパティーを追加」をクリックします。

b. このページの「ターゲット」セクションで、「選択可能」フィールドから管理対象オブジェクトが配置されるドメイン、クラスタ、またはサーバーインスタンスを選択して「追加」をクリックします。

「選択」フィールドに一覧表示されたドメイン、クラスタ、またはサーバーインスタンスのいずれかへの管理対象オブジェクトの配備を取り消す場合は、同フィールドから該当するものを選択して「削除」をクリックします。

6 「完了」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-admin-object`

▼ 管理対象オブジェクトリソースを編集する

1 ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。

2 「管理対象オブジェクトリソース」ノードを開きます。

3 編集する管理対象オブジェクトリソースのノードを選択します。

4 「管理対象オブジェクトリソースを編集」ページで、「管理対象オブジェクトリソースの作成」で指定した値を変更します。

5 「ターゲット」タブ付き区画で、「ターゲットの管理」をクリックして、管理対象オブジェクトが配備されているターゲットを編集することができます。

ターゲットの詳細については、138 ページの「管理対象オブジェクトリソースを作成する」を参照してください。

6 「保存」をクリックして、編集を適用します。

▼ 管理対象オブジェクトリソースを削除する

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「コネクタ」ノードを開きます。
2. 「管理対象オブジェクトリソース」ノードを選択します。
3. 「管理対象オブジェクトリソース」ページで、削除するリソースのチェックボックスにチェックマークを付けます。
4. 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

```
delete-admin-object
```

◆ ◆ ◆ 第 8 章

J2EE コンテナ

この章では、サーバーに含まれている J2EE コンテナの設定方法について説明します。この章には次の節が含まれています。

J2EE コンテナについて

この節では、Application Server に含まれている J2EE コンテナについて説明します。

J2EE コンテナのタイプ

J2EE コンテナは J2EE アプリケーションコンポーネントの実行時サポートを提供します。J2EE アプリケーションコンポーネントは、コンテナのプロトコルとメソッドを使用して、サーバーが提供するほかのアプリケーションコンポーネントとサービスにアクセスします。Application Server は、アプリケーションクライアントコンテナ、アプレットコンテナ、Web コンテナ、およびEJB コンテナを提供します。コンテナを示す図については、32 ページの「Application Server のアーキテクチャー」を参照してください。

Web コンテナ

Web コンテナは、Web アプリケーションをホストする J2EE コンテナです。Web コンテナは、サーブレットと JSP (JavaServer Pages) の実行環境を開発者に提供することにより、Web サーバーの機能を拡張します。

EJB コンテナ

Enterprise JavaBeans (EJB コンポーネント) は、ビジネスロジックを含む Java プログラミング言語サーバーコンポーネントです。EJB コンテナは、Enterprise JavaBeans へのローカルアクセスとリモートアクセスを提供します。

Enterprise JavaBeans には、セッション Beans、エンティティー Beans、およびメッセージ駆動型 Beans の 3 つのタイプがあります。セッション Beans は、一時的なオブジェクトやプロセスを表し、通常は 1 つのクライアントが使用します。エンティティー Beans は、通常データベースに保持されている持続性データを表します。メッセージ駆動型 Beans は、メッセージを非同期でアプリケーションモジュールやサービスに渡すために使われます。

コンテナの機能は、Enterprise JavaBean を作成したり、ほかのアプリケーションコンポーネントが Enterprise JavaBean にアクセスできるように Enterprise JavaBean をネームサービスにバインドしたり、承認されたクライアントだけが Enterprise JavaBean メソッドにアクセスできるようにしたり、Bean の状態を持続的記憶領域に保存したり、Bean の状態をキャッシュしたり、必要に応じて Bean を活性化したり非活性化したりすることです。

J2EE コンテナに関する管理コンソールタスク

一般的な Web コンテナ設定の設定

このリリースでは、管理コンソールに Web コンテナのコンテナ全体に関する設定はありません。

Web コンテナセッションの設定

この節では、Web コンテナの HTTP セッション設定について説明します。HTTP セッションは、持続ストアに書き込まれた状態データを持つ独自の Web セッションです。

▼ セッションタイムアウト値を設定する

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「**Web** コンテナ」ノードを選択します。

4 「セッションプロパティー」タブをクリックします。

5 「セッションタイムアウト」フィールドで、セッションが有効である秒数を入力します。

6 「保存」をクリックします。

▼ マネージャープロパティーを設定する

セッションマネージャーを使用して、セッションを作成および破棄する方法、セッション状態を格納する場所、およびセッションの最大数を設定できます。

セッションマネージャーの設定を変更するには、次の手順に従います。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「**Web** コンテナ」ノードを選択します。

4 「マネージャープロパティー」タブをクリックします。

5 リープ間隔の値を設定します。
「リープ間隔」フィールドで、非アクティブセッションデータがストアから削除されるまでの秒数を指定します。

6 最大セッションの値を設定します。
「最大セッション」フィールドで、許容されるセッションの最大数を指定します。

7 セッションファイル名の値を設定します。
「セッションファイル名」フィールドで、セッションデータを格納するファイルを指定します。

8 セッション ID ジェネレータクラス名の値を設定します。
「セッション ID ジェネレータクラス名」フィールドで、一意のセッション ID を生成するカスタムクラスを指定できます。サーバーインスタンスごとに 1 つのセッション ID ジェネレータクラスだけを作成できます。クラスタ内のすべてのインスタンスは、セッションキーの競合を防止するために、同じセッション ID ジェネレータを使用する必要があります。

カスタムセッション ID ジェネレータクラスは、次のとおり `com.sun.enterprise.util.uuid.UuidGenerator` インタフェースを実装する必要があります。

```
package com.sun.enterprise.util.uuid;

public interface UuidGenerator {

    public String generateUuid();
    public String generateUuid(Object obj);  //obj はセッションオブジェクト
}
```

このクラスは Application Server のクラスパスになければいけません。

9 「保存」をクリックします。

▼ ストアプロパティーを設定する

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「Web コンテナ」ノードを選択します。

4 「ストアプロパティー」タブをクリックします。

5 リープ間隔を設定します。
「リープ間隔」フィールドで、非アクティブセッションデータがストアから削除されるまでの秒数を指定します。

6 「保存」をクリックします。

一般的な EJB 設定の設定

この節では、サーバー上のすべての Enterprise JavaBean コンテナに適用される、次の設定を説明します。

- 145 ページの「セッション格納位置」
- 145 ページの「EJB プールを設定する」
- 146 ページの「EJB キャッシュを設定する」

デフォルト値をコンテナ別にオーバーライドするには、`sun-ejb-jar.xml` ファイル内で Enterprise JavaBeans の値を調整します。詳細については、『Application Server 開発者ガイド』を参照してください。

セッション格納位置

「セッション格納位置」フィールドは、ファイルシステムで非活性化された Beans と持続的な HTTP セッションが保存されるディレクトリを指定します。

非活性化された Beans とは、ファイルシステムのファイルに状態を書き込まれた Enterprise JavaBeans です。非活性化された Beans は、通常、特定の時間内のアイドル状態にあり、現在クライアントによってアクセスされていません。

非活性化された Beans と同じく、持続的な HTTP セッションはファイルシステム上のファイルに状態を書き込まれた個別の Web セッションです。

「コミットオプション」フィールドで、コンテナがトランザクション間の非活性化されたエンティティー Bean インスタンスをキャッシュする方法を指定します。

オプション B はトランザクション間のエンティティー Bean インスタンスをキャッシュし、デフォルトで選択されます。オプション C はキャッシュを無効にします。

▼ EJB プールを設定する

Beans の作成によってパフォーマンスに影響を受けることなく、クライアントの要求に応答するために、コンテナは Enterprise JavaBeans のプールを保持します。これらの設定は、ステートレスセッション Beans とエンティティー Beans だけに適用されます。

配備した Enterprise JavaBeans を使用するアプリケーションでパフォーマンス上の問題がある場合は、プールを作成したり、既存のプールで保持される Beans の数を増やしたりすることによって、アプリケーションのパフォーマンスを向上させることができます。

デフォルトで、コンテナは Enterprise JavaBeans のプールを保持しています。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「**EJB** コンテナ」ノードを選択します。

4 「初期および最小プールサイズ」フィールドの「プール設定」で、コンテナがプールで作成する **Beans** の最小数を入力します。

5 「最大プールサイズ」フィールドで、コンテナが一度にプール内に保持する **Beans** の最大数を入力します。

6 「プールサイズ変更量」フィールドに、「プールアイドルタイムアウト」フィールドで指定した時間を超えて **Beans** がアイドル状態になった場合にプールから削除される **Beans** の数を入力します。

7 「プールアイドルタイムアウト」フィールドに、プール内の **Bean** がプールから削除される前にアイドル状態でいられる時間を秒単位で入力します。

8 「保存」をクリックします。

9 **Application Server** を再起動します。

▼ EJB キャッシュを設定する

コンテナは、もっともよく使われる Enterprise JavaBeans の Enterprise JavaBean データのキャッシュを保持します。これにより、コンテナはその Enterprise JavaBeans のデータに対するほかのアプリケーションモジュールからの要求により速く応答できます。この節が適用されるのは、ステートフルセッション Beans とエンティティー Beans だけです。

キャッシュされた Enterprise JavaBeans は、アクティブ、アイドル、非活性化の 3 つのうち、いずれかの状態になっています。アクティブな Enterprise JavaBean には、現在クライアントがアクセスしています。アイドル Enterprise JavaBeans のデータは現在キャッシュに

ありますが、この Bean にアクセスしているクライアントはありません。非活性化 Bean のデータは一時的に保存されていて、クライアントが Bean を要求した場合はキャッシュに読み込まれます。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「**EJB** コンテナ」ノードを選択します。

4 「最大キャッシュサイズ」フィールドで最大キャッシュサイズを調整します。

Bean の作成と破棄のオーバーヘッドをなくすために、キャッシュする Beans の最大数を大きくします。ただし、キャッシュを大きくした場合、サーバーはより大きなメモリーとリソースを消費します。キャッシュ設定に対して動作環境が十分であることを確認してください。

5 「キャッシュのサイズ変更量」フィールドで、キャッシュのサイズ変更量を調整します。

キャッシュされた Beans の最大数に達すると、コンテナは複数の非活性化 Beans を、デフォルトで 32 に設定されたバックアップストアから削除します。

6 「キャッシュアイドルタイムアウト」フィールドで、エンティティー **Beans** にスケジュールされたキャッシュクリーンアップの間隔を秒単位で調整します。

キャッシュされたエンティティー Bean が一定時間アイドル状態になった場合、Bean は非活性化されます。つまり、その Bean の状態がバックアップストアに書き込まれます。

7 「削除タイムアウト」フィールドで、ステートフルセッション **Beans** をキャッシュストアまたは非活性化ストアから削除する時間を秒単位で調整します。

8 「選択内容の削除ポリシー」フィールドで、コンテナがステートフルセッション **Beans** を削除するために使用するポリシーを設定します。

「選択内容の削除ポリシー」フィールドに設定されたポリシーに基づいて、コンテナは削除するステートフルセッション Beans を決定します。コンテナがキャッシュから Beans を削除するために使えるポリシーには次の 3 つがあります。

- 最近使用されていない (NRU)
- ファーストインファーストアウト (FIFO)
- 最近の使用頻度がもっとも低い (LRU)

NRU ポリシーでは、最近使われていない Bean を削除します。FIFO ポリシーでは、キャッシュ内でもっとも古い Bean を削除します。LRU ポリシーでは、最近もっともアクセスされていない Bean を削除します。デフォルトでは、コンテナがNRU ポリシーを使うように設定されています。

エンティティー Beans は常に FIFO ポリシーを使って削除されます。

9 「保存」をクリックします。

10 **Application Server** を再起動します。

メッセージ駆動型 **Bean** 設定の設定

メッセージ駆動型 Beans のプールは、145 ページの「EJB プールを設定する」で説明したセッション Beans のプールと似ています。デフォルトで、コンテナはメッセージ駆動型 Beans のプールを保持しています。

このプールの設定を調整するには、次の手順に従います。

▼ **MDB** プールを設定する

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「**EJB** コンテナ」ノードを選択します。

4 「**MDB** 設定」タブをクリックします。

5 「初期および最小プールサイズ」フィールドの「プール設定」で、コンテナがプールで作成するメッセージ **Beans** の最小数を入力します。

6 「最大プールサイズ」フィールドで、コンテナが一度にプール内に保持する **Beans** の最大数を入力します。

7 「プールサイズ変更量」フィールドに、「プールアイドルタイムアウト」フィールドで指定した時間を超えて **Beans** がアイドル状態になった場合にプールから削除される **Beans** の数を入力します。

8 「プールアイドルタイムアウト」フィールドに、プール内の **Bean** がプールから削除される前にアイドル状態でいられる時間を秒単位で入力します。

9 「保存」をクリックします。

10 **Application Server** を再起動します。

EJB タイマーサービス設定の設定

タイマーサービスは、Enterprise JavaBeans により通知やイベントをスケジュールするのに使われ、Enterprise JavaBean コンテナが提供する持続的なトランザクション通知サービスです。ステートフルセッション Beans 以外の Enterprise JavaBeans はすべて、タイマーサービスからの通知を受信できます。このサービスによって設定されたタイマーは、サーバーのシャットダウンや再起動では破棄されません。

▼ タイマーサービスを設定する

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「**EJB** コンテナ」ノードを選択します。

4 「**EJB** タイマーサービス」タブをクリックします。

5 「最小配信間隔」フィールドで、最小配信間隔をミリ秒単位で設定します。

最小配信間隔とは、次のタイマーの有効期限が切れて特定のタイマーの発生が可能になるまでのミリ秒単位の時間です。この間隔の設定が短すぎると、サーバーがオーバーロードする可能性があります。

6 「最大再配信回数」フィールドで、タイマーサービスが通知の配信を試みる最大回数を設定します。

7 「再配信間隔」フィールドで、再配信を試みる間隔をミリ秒単位で設定します。

8 「保存」をクリックします。

9 **Application Server** を再起動します。

▼ タイマーサービスで外部データベースを使用する

デフォルトで、タイマーサービスはタイマーを格納するために埋め込みデータベースを使います。

1 93 ページの「**JDBC** リソースを作成する」の説明に従って、データベースの **JDBC** リソースを設定します。

2 「タイマーデータソース」フィールドで、そのリソースの **JNDI** 名を入力します。

3 「保存」をクリックします。

4 **Application Server** を再起動します。

PointBase や Oracle のタイマーデータベース作成ファイルのサンプルは、*install-dir*`/lib/install/databases/` で提供されています。

◆ ◆ ◆ 第 9 章

セキュリティーの設定

この章では、アプリケーションサーバーの主なセキュリティー概念について説明したあと、Application Server のセキュリティーの設定方法について説明します。この章では、次の項目について説明します。

Application Server のセキュリティーについて

セキュリティーの概要

セキュリティーとはデータの保護に関することであり、ストレージ内または伝送中のデータへの不正アクセスやそうしたデータの破損をどのようにして防止するか、ということです。Application Server には、J2EE 標準に基づく動的で拡張可能なセキュリティーアーキテクチャーがあります。標準実装されているセキュリティー機能には、暗号化、

認証と承認、および公開鍵インフラストラクチャーがあります。Application Server は Java セキュリティーモデルをベースに構築され、アプリケーションが安全に動作できるサンドボックスを使用するため、システムやユーザーにリスクが及ぶ可能性がありません。説明する項目は次のとおりです。

アプリケーションおよびシステムセキュリティーについて

概して、2 種類のアプリケーションセキュリティーがあります。

- 「プログラムによるセキュリティー」。開発者が記述したアプリケーションコードがセキュリティー動作を処理します。管理者が、このメカニズムを操作する必要はまったくありません。一般的に、プログラムによるセキュリティーは、J2EE コンテナで管理するのではなく、アプリケーションのセキュリティー設定をハードコード化するのでお勧めできません。
- 「宣言によるセキュリティー」。Application Server のコンテナがアプリケーションの配備記述子によりセキュリティーを処理します。宣言によるセキュリティーは、配備記述子を直接または `deploytool` などのツールで編集することによって操作できます。配備記述子はアプリケーションの開発後に変更可能なので、宣言によるセキュリティーの方が柔軟性に富んでいます。

アプリケーションによるセキュリティーのほかに、Application Server システムのアプリケーション全体に影響するシステムセキュリティーもあります。

プログラムによるセキュリティーはアプリケーション開発者により制御されるため、このマニュアルでは説明していません。宣言によるセキュリティーについては、このマニュアルである程度説明しています。このマニュアルは、主にシステム管理者を対象としているため、システムセキュリティーを中心に説明しています。

セキュリティー管理用ツール

Application Server には次のセキュリティー管理用ツールがあります。

- 管理コンソール。サーバー全体のセキュリティー設定、ユーザー、グループ、レルムの管理、およびほかのシステム全体のセキュリティータスクの実行に使用するブラウザベースのツール。管理コンソールの概要については、34 ページの「管理用ツール」を参照してください。管理コンソールで実行可能なセキュリティータスクの概要については、166 ページの「管理コンソールによるセキュリティーの管理」を参照してください。
- `asadmin`。管理コンソールと同じタスクの多くを行うコマンド行ツール。管理コンソールからできない操作でも、`asadmin` を使用して操作できる場合があります。コマンドプロンプトまたはスクリプトのいずれかから `asadmin` コマンドを実行して、繰り返しのタスクを自動化します。`asadmin`の概要については、34 ページの「管理用ツール」を参照してください。

- deploytool。個別のアプリケーションセキュリティーを制御するために、アプリケーション配備記述子を編集するグラフィカルパッケージ化ツールおよび配備ツール。deploytool はアプリケーション開発者を対象にしているため、このマニュアルでは使用方法の詳細な説明はしていません。deploytool の使用手順については、このツールのオンラインヘルプおよび http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html の『J2EE 1.4 Tutorial』を参照してください。

Java 2 プラットフォーム、J2SE (Java 2 Standard Edition) には、次の 2 つのセキュリティー管理用ツールがあります。

- keytool。デジタル証明書および鍵のペアの管理に使用するコマンド行ユーティリティー。keytool は、certificate レルムのユーザー管理に使用します。
- policytool。システム全体の Java セキュリティーポリシー管理に使用するグラフィカルユーティリティー。管理者が policytool を使用することはほとんどありません。

keytool、policytool、およびその他の Java セキュリティーツールの使用方法の詳細については、http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html#security の「Java 2 SDK Tools and Utilities」を参照してください。

Enterprise Edition では、NSS (Network Security Services) を実装した 2 つのセキュリティー管理ツールも利用可能です。NSS の詳細については、http://www.mozilla.org/projects/security/pki/nss/ を参照してください。それらのセキュリティー管理ツールは次のとおりです。

- certutil。証明書および鍵データベースの管理に使用されるコマンド行ユーティリティー。
- pk12util。証明書または鍵データベースと PKCS12 形式のファイル間における鍵と証明書のインポートおよびエクスポートに使用されるコマンド行ユーティリティー。

certutil、pk12util、およびその他の NSS セキュリティーツールの使用方法の詳細については、http://www.mozilla.org/projects/security/pki/nss/tools の「NSS Security Tools」を参照してください。

パスワードのセキュリティー管理

Application Server のこのリリースでは、特定のドメインの仕様が収められるファイル domain.xml 内に、Sun Java System Message Queue ブローカのパスワードが、あらかじめ平文で格納されています。このパスワードを収めた domain.xml ファイルの要素は、jms-host 要素の admin-password 属性です。このパスワードは変更不可能なので、インストールの際、セキュリティーに重大な影響は与えません。

ただし、管理コンソールを使用してユーザーやリソースを追加し、そのユーザーやリソースにパスワードを割り当ててください。このパスワードの中には、たとえばデータベースにアクセスするためのパスワードなど、平文で domain.xml ファイルに記述されているものがあります。このようなパスワードを平文で domain.xml ファイルに保持すると、セキュリティー上の危険を引き起こす可能性があります。次の手順を実行すると、admin-password 属性やデータベースパスワードを含む domain.xml のすべてのパスワードを暗号化できます。

▼ domain.xml 内のパスワードを暗号化する

1 domain.xml ファイルが格納されているディレクトリ (デフォルトでは *domain-dir*/config) から、次の asadmin コマンドを実行します。

```
asadmin create-password-alias --user admin alias-name
```

次に例を示します。

```
asadmin create-password-alias --user admin jms-password
```

パスワードプロンプトが表示されます。この場合は admin です。詳細については、create-password-alias、list-password-aliases、delete-password-alias コマンドのマニュアルページを参照してください。

2 domain.xml のパスワードの削除および置き換えを行います。これは、asadmin set コマンドを使用して行います。このような目的での set コマンドの使用例は、次のとおりです。

```
asadmin set --user admin server.jms-service.jms-host.
default_JMS_host.admin-password=${ALIAS=jms-password}
```

3 該当するドメインの **Application Server** を再起動します。

エンコード化されたパスワードを含むファイルの保護

ファイルの中にはエンコード化されたパスワードを含むものがあり、ファイルシステムのアクセス権を使用しての保護が必要になります。これらのファイルには次のものが含まれます。

- *domain-dir*/master-password

  このファイルにはエンコード化されたマスターパスワードが含まれているので、ファイルシステムのアクセス権 600 で保護する必要があります。

- asadmin への --passwordfile 引数を使用して、引数として渡すために作成されたすべてのパスワードファイル。これらは、ファイルシステムのアクセス権 600 で保護する必要があります。

▼ マスターパスワードを変更する

マスターパスワード (MP) とは、全体で共有するパスワードです。これを認証に使用したり、ネットワークを介して送信したりすることは決してありません。このパスワードはセキュリティー全体の要なので、ユーザーが必要に応じて手動で入力したり、またはファイルに隠蔽したりすることができます。これは、システムで最高の機密データです。ユーザーは、このファイルを削除することで、強制的に MP の入力を要求できます。マスターパスワードが変更されると、マスターパスワードキーストアに再保存されます。

1. ドメインの **Application Server** を停止します。新旧のパスワードの入力を促す `asadmin change-master-password` コマンドを使用して、依存するすべての項目を再暗号化してください。次に例を示します。

```
asadmin change-master-password>
Please enter the master password>
Please enter the new master password>
Please enter the the new master password again>
```

2. **Application Server** を再起動します。

注意 – この時点で、実行中のサーバーインスタンスを開始してはいけません。対応するノードエージェントの SMP が変更されるまでは、決して実行中のサーバーインスタンスを再起動しないでください。SMP が変更される前にサーバーインスタンスを再起動すると、起動に失敗します。

3. 各ノードエージェントおよび関連するサーバーを 1 つずつ停止します。`asadmin change-master-password` コマンドをもう一度実行してから、ノードエージェントおよび関連するサーバーを再起動してください。

4. すべてのノードエージェントで対応が終了するまで、次のノードエージェントで同様の作業を継続します。このようにして、継続的な変更作業が完了します。

▼ 管理パスワードを変更する

管理パスワードの暗号化については、153 ページの「パスワードのセキュリティー管理」を参照してください。管理パスワードの暗号化は強く推奨されています。管理パスワードを暗号化する前に変更する場合は、`asadmin set` コマンドを使用してください。このような目的での `set` コマンドの使用例は、次のとおりです。

```
asadmin set --user admin
server.jms-service.jms-host.default_JMS_host.admin-password=new_pwd
```

管理コンソールを使って管理パスワードを変更することもできます。その手順を次に示します。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2. 設定するインスタンスを選択します。

   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「レルム」ノードを展開します。

5 admin-realm ノードを選択します。

6 「レルムを編集」ページで、「ユーザーを管理」ボタンをクリックします。

7 admin という名前のユーザーを選択します。

8 新しいパスワードを入力し、そのパスワードをもう一度入力します。

9 「保存」をクリックして保存するか、「閉じる」をクリックして保存しないで閉じます。

セキュリティーの責任の割り当て

セキュリティーの責任は次のように割り当てます。

- 156 ページの「アプリケーション開発者」
- 156 ページの「アプリケーション配備担当者」
- 157 ページの「システム管理者」

アプリケーション開発者

アプリケーション開発者は次の責任を負います。

- アプリケーションコンポーネントに対するロールおよびロールベースのアクセス制限の指定。
- アプリケーションの認証メソッドの定義およびセキュリティー保護が行われるアプリケーションの各部の指定。

アプリケーション開発者は、deploytool などのツールを使用してアプリケーション配備記述子を編集できます。これらのセキュリティータスクの詳細については、『J2EE 1.4 Tutorial』の「Security」の章 (http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html) を参照してください。

アプリケーション配備担当者

アプリケーション配備担当者は次の責任を負います。

- ユーザーまたはグループ、あるいはその両方のセキュリティーロールへのマッピング。
- 特定の配備シナリオの要件に適合するための、コンポーネントメソッドのアクセスに必要な権限の定義。

アプリケーション配備担当者は、deploytool などのツールを使用してアプリケーション配備記述子を編集できます。これらのセキュリティータスクの詳細については、『J2EE 1.4 Tutorial』の「Security」の章 (http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html) を参照してください。

システム管理者

システム管理者は次の責任を負います。

- セキュリティーレルムの設定。
- ユーザーアカウントおよびグループの管理。
- 監査ログの管理。
- サーバー証明書の管理およびサーバーの **SSL (Secure Sockets Layer)** の使用の設定。
- コネクタ接続プールのセキュリティーマップ、その他の JACC プロバイダなど、その他のシステム全体のセキュリティー機能の管理。

システム管理者は、管理コンソールを使ってサーバーのセキュリティー設定を管理し、certutil を使って証明書を管理します。このマニュアルは主にシステム管理者を対象にしています。

認証と承認について

認証と承認は、アプリケーションサーバーセキュリティーの中心的な概念です。ここでは、認証と承認に関連する次の項目について説明します。

- 157 ページの「エンティティーの認証」
- 159 ページの「ユーザーの承認」
- 159 ページの「JACC プロバイダの指定」
- 159 ページの「認証および承認の決定の監査」
- 159 ページの「メッセージセキュリティーの設定」

エンティティーの認証

「認証」とは、あるエンティティー (ユーザー、アプリケーション、またはコンポーネント) が別のエンティティーが主張している本人であることを確認する方法です。エンティティーは、「セキュリティー資格」を使用して自らを認証します。資格には、ユーザー名、パスワード、デジタル証明書などが含まれます。

通常、認証はユーザー名とパスワードでユーザーがアプリケーションにログインすることを意味していますが、アプリケーションがサーバーのリソースを要求するとき、セキュリティー資格を提供する EJB を指す場合もあります。普通、サーバーやアプリケーションはクライアントに認証を要求しますが、さらにクライアントもサーバーに自らの認証を要求できます。双方向で認証する場合、これを相互認証と呼びます。

エンティティーが保護対象リソースにアクセスを試行する場合、Application Server はそのリソースに対して設定されている認証メカニズムを使用してアクセスを認可するかどうかを決定します。たとえば、ユーザーが Web ブラウザでユーザー名およびパスワードを入力でき、アプリケーションがその資格を確認する場合、そのユーザーは認証されます。それ以降のセッションで、ユーザーはこの認証済みのセキュリティー ID に関連付けられます。

157 ページの「エンティティーの認証」で説明しているように、Application Server は 4 種類の認証をサポートします。アプリケーションは、配備記述子で使用する認証タイプを指定します。`deploytool` を使ってアプリケーションの認証メソッドを設定する方法の詳細については、`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html` の『J2EE 1.4 Tutorial』を参照してください。

表 9–1 Application Server の認証メソッド

認証メソッド	通信プロトコル	説明	ユーザー資格の暗号化
基本	HTTP (オプションで SSL)	サーバーのビルトインポップアップログインダイアログボックスを使用します。	SSL を使用しないかぎりありません。
フォームベース	HTTP (オプションで SSL)	アプリケーションが独自仕様のカスタムログインおよびエラーページを提供します。	SSL を使用しないかぎりありません。
クライアント証明書	HTTPS (HTTP over SSL)	サーバーは公開鍵証明書を使用してクライアントを認証します。	SSL

シングルサインオンの確認

シングルサインオンとは、1 つの仮想サーバーインスタンスの複数のアプリケーションがユーザー認証状態を共有することを可能にするものです。シングルサインオンによって、1 つのアプリケーションにログインしたユーザーは、同じ認証情報が必要なほかのアプリケーションに暗黙的にログインするようになります。

シングルサインオンはグループに基づいています。配備記述子が同じ「グループ」を定義し、かつ同じ認証メソッド (基本、フォーム、ダイジェスト、証明書) を使用するすべての Web アプリケーションはシングルサインオンを共有します。

デフォルトでシングルサインオンは、Application Server に定義された仮想サーバーで有効です。シングルサインオンを無効にする方法については、195 ページの「シングルサインオン (SSO) を設定する」を参照してください。

ユーザーの承認

いったんユーザーが認証されると、「承認」のレベルによってどのような操作が可能かが決まります。ユーザーの承認は、ユーザーの「ロール」に基づいています。たとえば、人事管理アプリケーションでは、管理者には社員全員の個人情報を見ることを承認し、社員には自身の個人情報だけを見ることを承認します。ロールの詳細については、160 ページの「ユーザー、グループ、ロール、およびレルムについて」を参照してください。

JACC プロバイダの指定

JACC (Java Authorization Contract for Containers) は J2EE 1.4 仕様の一部で、プラグイン可能な承認プロバイダ用のインタフェースを定義しています。これによって、管理者は認証を行うためにサードパーティー製のプラグインモジュールを設定できます。

デフォルトで、Application Server は JACC 仕様に準拠する単純なファイルベースの承認エンジンを提供します。その他のサードパーティー製の JACC プロバイダを指定することもできます。

JACC プロバイダは JAAS (Java Authentication and Authorization Service) の API を使用します。JAAS によって、サービスが認証およびユーザーに対するアクセス制御を行うことが可能になります。これは、標準 PAM (Pluggable Authentication Module) フレームワークの Java テクノロジバージョンを実装しています。

認証および承認の決定の監査

Application Server は、「監査モジュール」によってすべての認証および承認の決定の監査トレールを提供します。Application Server は、デフォルトの監査モジュールのほか、監査モジュールのカスタマイズ機能も提供します。カスタム監査モジュールの開発の情報については、『Application Server Developer's Guide』を参照してください。

メッセージセキュリティーの設定

「メッセージセキュリティー」によって、サーバーは、メッセージレイヤーで Web サービスの呼び出しおよび応答をエンドツーエンドで認証できます。Application Server は、SOAP レイヤーのメッセージセキュリティープロバイダを使用して、メッセージセキュリティーを実装します。メッセージセキュリティープロバイダは、要求メッセージおよび応答メッセージに必要な認証のタイプなどの情報を提供します。サポートされている認証には次のタイプが含まれます。

- ユーザー名とパスワード認証を含む送信者認証。
- XML デジタル署名を含むコンテンツ認証。

このリリースには、2 つのメッセージセキュリティープロバイダが付属しています。メッセージセキュリティープロバイダは、SOAP レイヤーの認証用に設定されます。設定可能なプロバイダには、`ClientProvider` と `ServerProvider` があります。

メッセージレイヤーセキュリティーのサポートは、プラグイン可能な認証モジュールの形式で Application Server とそのクライアントコンテナに統合されています。Application Server では、メッセージレイヤーセキュリティーはデフォルトで無効になっています。

メッセージレベルのセキュリティーは、Application Server 全体または特定のアプリケーションあるいはメソッドに対して設定できます。Application Server レベルのメッセージセキュリティーの設定については、第 10 章を参照してください。アプリケーションレベルのメッセージセキュリティーの設定については、『Developer's Guide』の「Securing Applications」の章で説明されています。

ユーザー、グループ、ロール、およびレルムについて

Application Server は次のエンティティーに対して認証および承認ポリシーを実施します。

- 161 ページの「ユーザー」:「Application Server で定義される」個別の ID。一般に、ユーザーとは、人物、エンタープライズ Bean などのソフトウェアコンポーネント、またはサービスを意味します。認証されたユーザーを「主体」と呼ぶ場合もあります。また、ユーザーが「被認証者」と呼ばれる場合もあります。
- 161 ページの「グループ」:「Application Server で定義される」一連のユーザー。共通の特性に基づいて分類されます。
- 161 ページの「ロール」:アプリケーションによって定義される名前付きの承認レベル。ロールは錠を開ける鍵にたとえられます。多くの人が鍵のコピーを所持している場合があります。錠は、だれがアクセスを求めるかに関わらず、適切な鍵が使用される場合だけ対応します。
- 162 ページの「レルム」:ユーザーとグループの情報、および関連するセキュリティー資格を含むリポジトリ。レルムは、「セキュリティーポリシードメイン」とも呼ばれます。

注 – ユーザーおよびグループは Application Server 全体で指定されますが、各アプリケーションは独自のロールを定義します。アプリケーションがパッケージ化されて配備される場合、次の図に例示されているように、アプリケーションはユーザーまたはグループとロールとの間のマッピングを指定します。

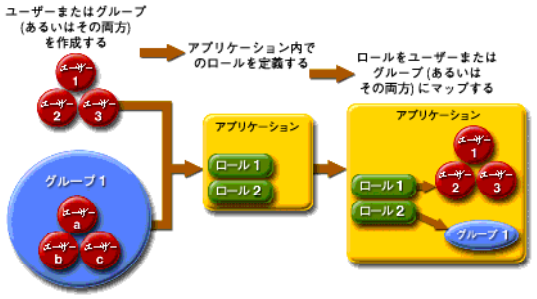

図 9–1 ロールマッピング

ユーザー

「ユーザー」とは、Application Server によって定義された個人またはアプリケーションプログラムの ID です。ユーザーは 1 つのグループと関連付けできます。Application Server の認証サービスは、複数のレルムでユーザーを管理できます。

グループ

「J2EE グループ」(または単にグループ) は、肩書きや顧客のプロファイルなど、共通の特性で分類されたユーザーのカテゴリです。たとえば、E コマースアプリケーションのユーザーは `CUSTOMER` グループに属しますが、お得意様は `PREFERRED` グループに属します。ユーザーをグループに分類すると、ユーザーからの大量のアクセスを制御することが容易になります。

ロール

「ロール」は、ユーザーが、どのアプリケーションのどの部分にアクセスできるかと、何を実行できるかを定義します。つまり、ロールによってユーザーの承認レベルが決まります。

たとえば、人事アプリケーションの場合、電話番号とメールアドレスにはすべての社員がアクセスできますが、給与情報にアクセスできるのは管理職だけです。このアプリケーションでは少なくとも 2 つのロールが定義されます。`employee` と `manager` です。そして、`manager` ロールのユーザーだけに給与情報の表示が許可されます。

ロールはアプリケーション内での役割を定義するのに対し、グループはある方法で関連付けられているユーザーの集まりに過ぎません。この点で、ロールとユーザーグループは異なります。たとえば、人事アプリケーションには、`full-time`、`part-time`、および `on-leave` などのグループがありますが、これらすべてのグループのユーザーは `employee` ロールに所属します。

ロールはアプリケーション配備記述子で定義されます。反対に、グループはサーバーおよびレルム全体に対して定義されます。アプリケーション開発者または配備担当者は、配備記述子により各アプリケーション内で、ロールを 1 つまたは複数のグループに割り当てます。

レルム

「セキュリティーポリシードメイン」または「セキュリティードメイン」とも呼ばれている「レルム」とは、サーバーによって共通のセキュリティーポリシーが定義および適用される範囲のことです。実際には、レルムとはサーバーがユーザーおよびグループの情報を格納するリポジトリです。

Application Server では、3 つのレルムが事前に設定されています。`file` (初期のデフォルトレルム)、`certificate`、および `admin-realm` です。さらに、`ldap` レルム、`solaris` レルム、またはカスタムレルムも設定できます。アプリケーションは、その配備記述子でレルムを指定して使用できます。レルムを指定しない場合、Application Server はデフォルトレルムを使用します。

`file` レルムでは、サーバーはユーザー資格を `keyfile` という名前のファイルにローカルで格納します。管理コンソールを使用して `file` レルムのユーザーを管理できます。詳細については、180 ページの「file レルムユーザーの管理」を参照してください。

`certificate` レルムでは、サーバーはユーザー資格を証明書データベースに格納します。`certificate` レルムを使用する際、サーバーは HTTP プロトコルを使う証明書を使用して Web クライアントを認証します。証明書の詳細については、163 ページの「証明書および SSL の概要」を参照してください。

`admin-realm` は `FileRealm` でもあり、管理者ユーザーの資格を `admin-keyfile` という名前のファイルにローカルで格納します。`file` レルムでユーザーを管理するのと同じ方法で、管理コンソールを使用してこのレルムのユーザーを管理してください。詳細については、180 ページの「file レルムユーザーの管理」を参照してください。

`ldap` レルムでは、サーバーは Sun Java System Directory Server などの LDAP (Lightweight Directory Access Protocol) サーバーからユーザー資格を取得します。LDAP とは、一般のインターネットまたは会社のイントラネットのどちらであっても、ネットワークでの組織、個人、およびファイルやデバイスなどその他のリソースの検出をだれにでもできるようにするプロトコルです。`ldap` レルムのユーザーおよびグループの管理については、LDAP サーバーのドキュメントを参照してください。

`solaris` レルムでは、サーバーは Solaris オペレーティングシステムからユーザー資格を取得します。このレルムは Solaris 9 OS 以降でサポートされています。`solaris` レルムのユーザーおよびグループの管理については、Solaris のドキュメントを参照してください。

カスタムレルムとは、リレーショナルデータベースやサードパーティー製のコンポーネントなどその他のユーザー資格のリポジトリです。詳細については、178 ページの「カスタムレルムの作成」を参照してください。

証明書および SSL の概要

この節では、次の項目について説明します。

デジタル証明書について

「デジタル証明書」(または単に証明書) とは、インターネット上の人物やリソースを一意に識別する電子ファイルです。さらに証明書は 2 つのエンティティー間の安全で機密保護された通信を可能にします。

個人により使用される個人証明書や SSL (Secure Sockets Layer) テクノロジでサーバーとクライアント間の安全なセッションを確立するために使用されるサーバー証明書など、さまざまな種類の証明書があります。SSL の詳細については、164 ページの「SSL (Secure Sockets Layer) について」を参照してください。

証明書は「公開鍵暗号化」に基づき、意図した受信者だけが解読できるようデジタルの「鍵」 (非常に長い数値) のペアを使用して「暗号化」、または符号化します。そして受信者は、情報を「復号化」して解読します。

鍵のペアには公開鍵と非公開鍵が含まれます。所有者は公開鍵を配布して、だれでも利用できるようにします。しかし、所有者は非公開鍵を決して配布せず、常時秘密にしておきます。鍵は数学的に関連しているので、1 つの鍵で暗号化されたデータは、そのペアのもう 1 つの鍵でだけ復号化ができます。

証明書とはパスポートのようなものです。所有者を識別し、その他の重要な情報を提供します。証明書は、「証明書発行局」(CA) と呼ばれる、信頼できるサードパーティーが発行します。CA はパスポートセンターに似ています。CA は、証明書の所有者の身元を確認したあと、偽造や改ざんができないように証明書に署名します。いったん CA が証明書に署名すると、所有者は ID の証明としてこれを提出することで、暗号化され、機密保護された通信を確立できます。

最も重要な点は、証明書によって所有者の公開鍵が所有者の ID と結び付けられることです。パスポートが写真とその所有者についての個人情報を結び付けるように、証明書は公開鍵とその所有者についての情報を結び付けます。

公開鍵のほかに、通常、証明書には次のような情報が含まれています。

- 所有者の名前、および証明書を使用する Web サーバーの URL や個人のメールアドレスなどその他の識別情報。
- 証明書が発行された CA の名前。
- 有効期限の日付。

デジタル証明書は、X.509 形式の技術仕様で管理されます。certificate レルムのユーザ ID を検証するために、 `certificate` 認証サービスは X.509 証明書の共通名フィールドを主体名として使用して、X.509 証明書を検証します。

証明書チェーンについて

Web ブラウザは、ブラウザが自動的に信頼する一連の「ルート」CA 証明書で事前に設定されます。別の場所で発行されたすべての証明書は、有効性を検証するために「証明書チェーン」を備えている必要があります。証明書チェーンとは、最後がルート CA 証明書で終わる、継続的な CA によって発行される一連の証明書です。

証明書が最初に生成される場合、それは「自己署名付き」証明書です。自己署名付き証明書とは、発行者 (署名者) が被認証者 (公開鍵が証明書で認証されているエンティティー) と同じものです。所有者は、証明書の署名要求 (CSR) を CA に送信するとき、その応答をインポートし、自己署名付き証明書が証明書のチェーンによって置き換えられます。チェーンの元の部分には、被認証者の公開鍵を認証する CA によって発行された証明書 (応答) があります。このチェーンの次の証明書は、CA の公開鍵を認証するものです。通常、これは自己署名付き証明書 (つまり、自らの公開鍵を認証する CA からの証明書) およびチェーンの最後の証明書です。

CA が証明書のチェーンに戻ることができる場合もあります。この場合、チェーンの元の証明書は同じ (キーエントリの公開鍵を認証する、CA によって署名された証明書) ですが、チェーン 2 番目の証明書が、CSR の送信先の CA の公開鍵を認証する、異なる CA によって署名された証明書です。そして、チェーンのその次の証明書は 2 番目の鍵を認証する証明書というように、自己署名付き「ルート」証明書に到達するまで続きます。こうして、チェーンの最初以降の各証明書は、チェーンの前にある証明書の署名者の公開鍵を認証します。

SSL (Secure Sockets Layer) について

「SSL」(Secure Sockets Layer) とは、インターネットの通信およびトランザクションのセキュリティー保護で最も普及している標準仕様です。Web アプリケーションは HTTPS (HTTP over SSL) を使用します。HTTPS は、サーバーとクライアント間のセキュアで機密保護された通信を確保するため、デジタル証明書を使用します。SSL 接続では、クライアントとサーバーの両方が送信前にデータを暗号化し、受信するとそれを復号化します。

クライアントの Web ブラウザがセキュアなサイトに接続する場合、次のように「SSL ハンドシェーク」が行われます。

- ブラウザはネットワークを介してセキュアなセッションを要求するメッセージを送信します。通常は、`http` ではなく `https` で始まる URL を要求します。
- サーバーは、公開鍵を含む証明書を送信することで応答します。
- ブラウザは、サーバーの証明書が有効であること、またサーバーの証明書が証明書をブラウザのデータベースに持つ信頼されている CA によって署名されていることを検証します。さらに、CA の証明書の有効期限が切れていないことも検証します。
- 証明書が有効な場合、ブラウザは 1 回限りの一意の「セッション鍵」を生成し、サーバーの公開鍵でそれを暗号化します。そして、ブラウザは暗号化されたセッション鍵をサーバーに送信し、両方でコピーを持てるようにします。
- サーバーは、非公開鍵を使用してメッセージを復号化し、セッション鍵を復元します。

ハンドシェークの後、クライアントは Web サイトの ID を検証し、クライアントと Web サーバーだけがセッション鍵のコピーを持ちます。これ以降、クライアントとサーバーはセッション鍵を使用して互いにすべての通信を暗号化します。こうすると、通信は確実にセキュアになります。

SSL 標準の最新バージョンは TLS (Transport Layer Security) と呼ばれています。Application Server は、SSL (Secure Sockets Layer) 3.0 および TLS (Transport Layer Security) 1.0 暗号化プロトコルをサポートしています。

SSL を使用するには、セキュアな接続を受け付ける各外部インタフェースまたは IP アドレスの証明書を、Application Server が所持しておく必要があります。ほとんどの Web サーバーの HTTPS サービスは、デジタル証明書がインストールされるまで実行されません。204 ページの「keytool ユーティリティーを使って証明書を生成する」の手順に従って、Web サーバーが SSL 用に使用できるデジタル証明書を設定してください。

暗号化方式について

「暗号化方式」とは、暗号化と復号化に使用される暗号化アルゴリズムです。SSL および TLS プロトコルは、サーバーとクライアントでお互いを認証するために使用される多くの暗号化方式のサポート、証明書の送信、およびセッション鍵の確立を行います。

安全度は、暗号化方式によって異なります。クライアントとサーバーは異なる暗号化方式群をサポートできます。SSL および TLS プロトコルから暗号化方式を選択してください。クライアントとサーバーは安全な接続のために、双方で通信に使用可能であるもっとも強力な暗号化方式を使用します。そのため、通常は、すべての暗号化方式を有効にすれば十分です。

名前ベースの仮想ホストの使用方法

セキュアなアプリケーションに名前ベースの仮想ホストを使用すると、問題が発生する場合があります。これは、SSL プロトコル自体の設計上の制約です。クライアントブラウザがサーバーの証明書を受け付ける SSL ハンドシェークは、HTTP 要求がアクセスされる前に行われる必要があります。その結果、認証より前に仮想ホスト名を含む要求情報を特定できないので、複数の証明書を単一の IP アドレスに割り当てできません。

単一の IP すべての仮想ホストが同じ証明書に対して認証を必要とする場合、複数の仮想ホストを追加しても、サーバーの通常の SSL 動作を妨害する可能性はありません。ただし、証明書 (主に正式な CA の署名済みの証明書が該当) に表示されているドメイン名がある場合、ほとんどのブラウザがサーバーのドメイン名をこのドメイン名と比較することに注意してください。ドメイン名が一致しない場合、これらのブラウザは警告を表示します。一般的には、アドレスベースの仮想ホストだけが本稼働環境の SSL で広く使用されています。

ファイアウォールについて

「ファイアウォール」は、2 つ以上のネットワーク間のデータフローを制御し、ネットワーク間のリンクを管理します。ファイアウォールは、ハードウェア要素およびソフトウェア要素で構成できます。この節では、一般的ないくつかのファイアウォールアーキテクチャーとその設定について説明します。ここでの情報は、主に Application Server に関係するものです。特定のファイアウォールテクノロジの詳細については、使用しているファイアウォールのベンダーのドキュメントを参照してください。

一般的には、クライアントが必要な TCP/IP ポートにアクセスできるようにファイアウォールを設定します。たとえば、HTTP リスナーがポート 8080 で動作している場合は、HTTP 要求をポート 8080 だけで受け付けるようにファイアウォールを設定します。同様に、HTTPS 要求が ポート 8181 に設定されている場合は、HTTPS 要求をポート 8181 で受け付けるようにファイアウォールを設定する必要があります。

インターネットから EJB モジュールへ直接の RMI-IIOP (Remote Method Invocations over Internet Inter-ORB Protocol) アクセスが必要な場合は、同様に RMI-IIOP リスナーポートを開きますが、これにはセキュリティー上のリスクが伴うので、使用しないことを強くお勧めします。

二重のファイアウォールのアーキテクチャーでは、HTTP および HTTPS トランザクションを受け付けるように外部ファイアウォールを設定する必要があります。また、ファイアウォールの背後の Application Server と通信する HTTP サーバープラグインを受け付けるように内部ファイアウォールを設定する必要があります。

管理コンソールによるセキュリティーの管理

管理コンソールは、セキュリティーの次の側面を管理する手段を提供します。

サーバーセキュリティー設定

「セキュリティー設定」ページでは、デフォルトレルム、匿名ロール、およびデフォルトの主体ユーザー名とパスワードの指定を含む、サーバー全体のプロパティーを設定します。詳細については、169 ページの「セキュリティー設定を設定する」を参照してください。

レルムおよび file レルムユーザー

レルムの概念については、160 ページの「ユーザー、グループ、ロール、およびレルムについて」で紹介しています。

- 新しいレルムの作成
- 既存のレルムの削除
- 既存のレルムの設定変更
- `file` レルムのユーザーの追加、変更、および削除
- デフォルトレルムの設定

これらのタスクの詳細については、171 ページの「レルムに関する管理コンソールタスク」を参照してください。

JACC プロバイダ

JACC プロバイダについては、159 ページの「JACC プロバイダの指定」で紹介しています。管理コンソールでは、次のタスクを実行します。

- 新しい JACC プロバイダの追加
- 既存の JACC プロバイダの削除または変更

これらのタスクの詳細については、184 ページの「JACC プロバイダに関する管理コンソールタスク」を参照してください。

監査モジュール

監査モジュールについては、159 ページの「認証および承認の決定の監査」で紹介しています。監査は、エラーやセキュリティー違反などの重大なイベントが発生した場合に、それを後から調べることができるようにイベントを記録するメソッドです。Application Server のログにすべての認証イベントが記録されます。完全なアクセスログには、Application Server で行われるすべてのアクセスイベントが連続して記録されます。

管理コンソールでは、次のタスクを実行します。

- 新しい監査モジュールの追加
- 既存の監査モジュールの削除または変更

これらのタスクの詳細については、187 ページの「監査モジュールに関する管理コンソールタスク」を参照してください。

メッセージセキュリティー

メッセージセキュリティーの概念については、159 ページの「メッセージセキュリティーの設定」で紹介しています。管理コンソールでは、次のタスクを実行します。

- メッセージセキュリティーの有効化
- メッセージセキュリティープロバイダの設定
- 既存のメッセージセキュリティー設定またはプロバイダの削除または設定

これらのタスクの詳細については、第 10 章を参照してください。

HTTP および IIOP リスナーのセキュリティー

HTTP サービスの各仮想サーバーは、1 つまたは複数の「HTTP リスナー」を介してネットワーク接続を提供します。HTTP サービスおよび HTTP リスナーについての一般情報については、239 ページの「HTTP サービスとは」を参照してください。

Application Server は、CORBA (Common Object Request Broker Architecture) オブジェクトをサポートしており、これはネットワークを介しての通信をするために IIOP (Internet Inter-Orb Protocol) を使用します。「IIOP リスナー」は、EJB コンポーネントのリモートクライアントおよびほかの CORBA ベースのクライアントから受信する接続を受け付けます。IIOP リスナーについての一般情報については、260 ページの「IIOP リスナー」を参照してください。

管理コンソールでは、次のタスクを実行します。

- 新しい HTTP または IIOP リスナーの作成と、それを使用するセキュリティーの指定
- 既存のHTTP または IIOP リスナーのセキュリティー設定の変更

これらのタスクの詳細については、192 ページの「リスナーと JMX コネクタに関する管理コンソールタスク」を参照してください。

管理サービスのセキュリティー

管理サービスは、サーバーインスタンスが通常のインスタンスか、ドメイン管理サーバー (DAS) か、あるいは組み合わせかを決定します。管理サービスを使用して、JSR-160 準拠のリモート JMX コネクタを設定してください。これは、ドメイン管理サーバーとノードエージェントとの間の通信を処理し、ノードエージェントは、リモートサーバーインスタンスの代わりに、ホストマシンのサーバーインスタンスを管理します。

管理コンソールでは、次のタスクを実行します。

- 管理サービスの管理
- JMX コネクタの編集
- JMX コネクタのセキュリティー設定の変更

これらのタスクの詳細については、194 ページの「管理サービスの JMX コネクタのセキュリティーを設定する」を参照してください。

セキュリティーマップ

コネクタ接続プールのセキュリティーマップの概念については、198 ページの「セキュリティーマップについて」で紹介しています。管理コンソールでは、次のタスクを実行します。

- 既存のコネクタ接続プールへのセキュリティーマップの追加
- 既存のセキュリティーマップの削除または設定

これらのタスクの詳細については、197 ページの「コネクタ接続プールに関する管理コンソールタスク」を参照してください。

セキュリティーに関する管理コンソールタスク

▼ セキュリティー設定を設定する

管理コンソールの「セキュリティー」ページでは、システム全体のセキュリティー設定をさまざまに設定できます。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを選択します。
「セキュリティー」ページが表示されます。

4 必要に応じて値を変更します。
次の表では、一般的なセキュリティーオプションについて説明しています。

設定	説明
監査ログ	選択すると監査ログが有効になります。有効な場合、サーバーは「監査モジュール」設定で指定されたすべての監査モジュールのロードおよび実行を行います。無効な場合、サーバーは監査モジュールにアクセスしません。デフォルトは無効です。
デフォルトレルム	サーバーが認証用に使用する有効な (デフォルト) レルム。アプリケーションは、配備記述子で異なるレルムを指定しないかぎり、このレルムを使用します。設定されているすべてのレルムがこのリストに表示されます。デフォルトの初期レルムは、`file` レルムです。
匿名ロール	デフォルトまたは匿名ロールの名前。匿名ロールはすべてのユーザーに割り当てられます。アプリケーションは、配備記述子でこのロールを使用して任意の人に承認を付与することができます。

設定	説明
デフォルト主体	デフォルトのユーザー名を指定します。サーバーは、主体が指定されない場合にこれを使用します。このフィールドに値を入力する場合は、「デフォルト主体のパスワード」フィールドに対応する値を入力します。 この属性は通常のサーバー動作には不要です。
デフォルト主体のパスワード	「デフォルト主体」フィールドで指定したデフォルト主体のパスワード。 この属性は通常のサーバー動作には不要です。
JACC	設定されている JACC プロバイダのクラス名。184 ページの「JACC プロバイダを作成する」を参照してください
監査モジュール	コンマで区切られている、監査モジュールプロバイダのクラスのリスト。ここで表示されるモジュールは事前に設定しておく必要があります。監査ログが有効な場合、この設定で監査モジュールが表示される必要があります。デフォルトで、サーバーは `default` という名前の監査モジュールを使用します。新しい監査モジュールの作成方法については、187 ページの「監査モジュールを作成する」を参照してください。

5 「追加プロパティー」セクションに、**JVM (Java Virtual Machine)** に渡すための追加プロパティーを入力します。

有効なプロパティーは、「デフォルトレルム」フィールドで選択したレルムのタイプによって異なります。有効なプロパティーは、次の項目で説明されています。

- 179 ページの「file および admin-realm レルムの編集」
- 179 ページの「certificate レルムの編集」
- 177 ページの「solaris レルムの作成」
- 176 ページの「ldap レルムの作成」
- 178 ページの「カスタムレルムの作成」

6 「保存」を選択して変更を保存するか、または「デフォルトを読込み」を選択してデフォルト値を復元します。

▼ 管理ツールへのアクセスを許可する

`asadmin` グループのユーザーだけが、管理コンソールおよび `asadmin` コマンド行ユーティリティーにアクセスできます。

この管理ツールへのアクセスをユーザーに許可するには、ユーザーを `admin-realm` レルムの `asadmin` グループに追加してください。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「レルム」ノードを展開します。

5 admin-realm ノードを選択します。

6 「レルムを編集」ページで、「ユーザーを管理」ボタンをクリックします。

インストール後当初は、インストールの際に入力した管理者ユーザー名およびパスワードが admin-keyfile という名前のファイルに表示されます。このユーザーは、デフォルトで Application Server を変更する権限を付与される asadmin グループに属します。ユーザーに Application Server の管理者権限を付与する場合にかぎり、ユーザーをこのグループに割り当てます。

ユーザーを admin-realm レルムに追加しても、ユーザーに割り当てるグループが asadmin 以外の場合、ユーザーの情報は admin-keyfile という名前のファイルに記述されますが、そのユーザーには file レルムの管理ツールまたはアプリケーションへのアクセス権はありません。

7 「新規」をクリックして、新しいユーザーを admin-realm レルムに追加します。

8 正確な情報を「ユーザー ID」、「パスワード」、および「グループ」フィールドに入力します。

Application Server に変更を加えるユーザーを承認するには、「グループリスト」に asadmin グループを含めます。

9 「了解」をクリックして、このユーザーを admin-realm レルムに追加するか、または「取消し」をクリックして保存せずに終了します。

レルムに関する管理コンソールタスク

▼ レルムを作成する

Application Server では、3 つのレルムが事前に設定されています。file、certificate、および admin-realm です。さらに、ldap レルム、solaris レルム、またはカスタムレルムも作成できます。一般に、1 つのサーバー上には各タイプのレルムが 1 つずつ存在しますが、Application Server 上には file レルムが 2つ存在します。file と admin-realm です。これらは、異なる 2 つの目的に使用される同じタイプの 2 つのレルムです。また、システムの各仮想サーバーで異なる証明書データベースを持つこともできます。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
   - すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。
3. 「セキュリティー」ノードを展開します。
4. 「レルム」ノードを選択します。
5. 「レルム」ページで、「新規」をクリックします。
   「レルムを作成」ページが表示されます。
6. 「名前」フィールドにレルムの名前を入力します。
7. 作成するレルムのクラス名を指定します。
   有効な選択肢を次の表に示します。

レルム名	クラス名
file	com.sun.enterprise.security.auth.realm.file.FileRealm
certificate	com.sun.enterprise.security.auth.realm.certificate.CertificateRealm
ldap	com.sun.enterprise.security.auth.realm.ldap.LDAPRealm
solaris	com.sun.enterprise.security.auth.realm.solaris.SolarisRealm
custom	ログインレルムクラスの名前

8 レルムに必要なプロパティーおよび希望するオプションのプロパティーを追加します。

a. 「プロパティーを追加」をクリックします。

b. 「名前」フィールドに、プロパティーの名前を入力します。

- `file` レルムのプロパティーについては、179 ページの「file および admin-realm レルムの編集」を参照してください。
- `certificate` レルムのプロパティーについては、179 ページの「certificate レルムの編集」を参照してください。
- `ldap` レルムのプロパティーについては、176 ページの「ldap レルムの作成」を参照してください。
- `solaris` レルムのプロパティーについては、177 ページの「solaris レルムの作成」を参照してください。
- カスタムレルムのプロパティーについては、178 ページの「カスタムレルムの作成」を参照してください。

c. 「値」フィールドに、プロパティーの値を入力します。

9 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-auth-realm`

▼ レルムを編集する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「レルム」ノードを展開します。

5 既存のレルムの名前を選択します。
「レルムを編集」ページが表示されます。

6 必要に応じて既存のプロパティーとその値を編集します。

7 さらにプロパティーを追加するには、「プロパティーを追加」ボタンをクリックします。

ページに新しい行が表示されます。有効なプロパティー名およびプロパティー値を入力します。

- `file` レルムのプロパティーについては、179 ページの「file および admin-realm レルムの編集」を参照してください。
- `certificate` レルムのプロパティーについては、179 ページの「certificate レルムの編集」を参照してください。
- `ldap` レルムのプロパティーについては、176 ページの「ldap レルムの作成」を参照してください。
- `solaris` レルムのプロパティーについては、177 ページの「solaris レルムの作成」を参照してください。
- カスタムレルムのプロパティーについては、178 ページの「カスタムレルムの作成」を参照してください。

8 「保存」をクリックして変更を保存します。

▼ レルムを削除する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「レルム」ノードを選択します。

5 削除するレルムの横にあるボックスをクリックします。

6 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-auth-realm`

▼ デフォルトレルムを設定する

「デフォルトレルム」とは、アプリケーションの配備記述子がレルムを指定しない場合に、Application Server が認証と承認に使用するレルムです。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。
3. 「セキュリティー」ノードを選択します。
   「セキュリティー」ページが表示されます。
4. 「デフォルトレルム」フィールドで、ドロップダウンリストから必要なレルムを選択します。
5. 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」をクリックして変更を削除し、**Application Server** のデフォルト値を復元します。
6. コンソールに「再起動が必要です」と表示される場合は、サーバーを再起動します。

特定のレルムに関する追加情報

この節では、次の項目について説明します。

ldap レルムの作成

ldap レルムは、LDAP サーバーからの情報を使用して認証を行います。ユーザー情報には、ユーザー名、パスワード、およびユーザーが属するグループが含まれます。LDAP レルムを使用するには、ユーザーおよびグループを LDAP ディレクトリで事前に定義しておいてください。

LDAP レルムを作成するには、172 ページの「レルムを作成する」の手順に従って新しいレルムを追加したあと、次の表に示したプロパティーを追加します。

表 9–2 ldap レルムに必要なプロパティー

プロパティー名	説明	値
directory	ディレクトリサーバーの LDAP URL。	ldap://*hostname*:*port* という形式の LDAP URL。例: ldap://myldap.foo.com:389。
base-dn	ユーザーデータの場所のベース DN (Distinguished Name)。ツリー範囲検索が実行されるため、ユーザーデータのレベルより上に置かれます。検索ツリーが小さければ小さいほど、パフォーマンスが向上します。	検索用のドメイン。例: dc=siliconvalley, dc=BayArea, dc=sun, dc=com。
jaas-context	このレルムに使用するログインモジュールのタイプ。	ldapRealm が必須です。

ldap レルムのオプションのプロパティーを、次の表に示します。

表 9–3 ldap レルムのオプションのプロパティー

プロパティー名	説明	デフォルト
search-filter	ユーザー検索に使用される検索フィルタ。	uid=%s (%s はサブジェクト名に展開される)。
group-base-dn	グループデータの場所のベース DN。	base-dn と同じですが、必要に応じて変更可能です。
group-search-filter	ユーザーのグループメンバーシップ検索に使用する検索フィルタ。	uniquemember=%d (%d はユーザー要素 DN に展開される)。
group-target	グループ名エントリを含む LDAP の属性名。	CN
search-bind-dn	search-filter 検索を実行するディレクトリの認証に使用するオプション DN。匿名検索を実行できないディレクトリにのみ必要になります。	

表 9–3 ldap レルムのオプションのプロパティー　(続き)

プロパティー名	説明	デフォルト
search-bind-password	search-bind-dn で指定した DN の LDAP パスワード。	

例

たとえば、次のように LDAP ユーザー、Joe Java が LDAP ディレクトリで定義されているとします。

```
uid=jjava,ou=People,dc=acme,dc=com
uid=jjava
givenName=joe
objectClass=top
objectClass=person
objectClass=organizationalPerson
objectClass=inetorgperson
sn=java
cn=Joe Java
```

ldap レルムの作成または編集をする際には、例示したコードを使用して、次の表で示す値を入力できます。

表 9–4 ldap レルムの値の例

プロパティー名	プロパティー値
directory	サーバーへの LDAP URL。例: ldap://ldap.acme.com:389
base-dn	ou=People,dc=acme,dc=com. より上位に置くことも可能です (例: dc=acme、dc=com)。ただし、トラバースするツリーの範囲が広ければ、それだけパフォーマンスが低下します。
jaas-context	ldapRealm

solaris レルムの作成

solaris レルムは、システム設定で指定されているとおり、基礎となる Solaris ユーザーデータベースからユーザーおよびグループの情報を取得します。solaris レルムは、認証のための基礎となる PAM インフラストラクチャーを呼び出します。設定されている PAM モジュールに root 権限が必要な場合、ドメインはこのレルムを使用するため root として実行される必要があります。詳細については、セキュリティーサービスに関する Solaris ドキュメントを参照してください。

solaris には、1 つの必須プロパティー jaas-context があり、これは使用するログインモジュールのタイプを指定します。プロパティー値は solarisRealm である必要があります。

注 – solaris レルムは Solaris 9 以降でのみサポートされています。

カスタムレルムの作成

4 つのビルトインレルムに加えて、ユーザーデータを、リレーショナルデータベースなどのほかの何らかの方法で格納するカスタムレルムも作成できます。カスタムレルムの作成は、このマニュアルの対象外です。詳細については、Application Server の『Developer's Guide』の「Securing Applications」の章を参照してください。

管理者として知っておく必要のある主な事柄は、カスタムレルムが、JAAS (Java Authentication and Authorization Service) パッケージから派生した `LoginModule` と呼ばれるクラスによって実装されていることです。

▼ カスタムレルムを作成する

1. 172 ページの「レルムを作成する」で概説した手順に従い、カスタムレルムの名前と `LoginModule` クラスの名前を入力します。

   `myCustomRealm` などの一意で任意の名前が、カスタムレルムに使用できます。

2. 次の表に示すカスタムレルムのプロパティーを追加します。

プロパティー名	プロパティー値
jaas-context	`LoginModule` クラス名。例: `simpleCustomRealm`
auth-type	レルムの説明。例:「カスタムレルムの分かりやすい例」。

3. 「了解」をクリックします。

4. ドメインのログイン設定ファイル *domain-dir*`/config/login.conf` を編集し、このファイルの最後に **JAAS** `LoginModule` の完全修飾クラス名を次のように追加します。

   ```
   realmName {
       fully-qualified-LoginModule-classname required;
   };
   ```

   次に例を示します。

   ```
   myCustomRealm {
       com.foo.bar.security.customrealm.simpleCustomLoginModule required;
   };
   ```

5. `LoginModule` クラスと依存するすべてのクラスを、ディレクトリ *domain-dir*`/lib/classes` にコピーします。

6 コンソールに「再起動が必要です」と表示される場合は、サーバーを再起動します。

7 レルムが正常にロードされたことを確認します。

domain-dir/logs/server.logを開き、サーバーがレルムを読み込んだことを確認します。サーバーは、レルムの init() メソッドを呼び出す必要があります。

certificate レルムの編集

certificate レルムは、SSL 認証をサポートしています。このレルムは、Application Server のセキュリティーコンテキスト内にユーザー ID を設定したあと、トラストストアファイルとキーストアファイル内の暗号的に検証されたクライアント証明書からユーザーデータを取得し、それをそのユーザー ID に格納します (201 ページの「証明書ファイルについて」を参照)。これらのファイルにユーザーを追加するには、certutil を使用します。

J2EE コンテナは、certificate レルムを使用して、証明書からの各ユーザーの DN (Distinguished Name) に基づいた承認処理を行います。DN とは、その公開鍵を証明書が識別するエンティティーの名前です。この名前には、X.500 標準が使用され、インターネット全体で一意であるように意図されています。キーストアおよびトラストストアの詳細については、 certutil のドキュメント (206 ページの「NSS (Network Security Services) ツールの使用」)を参照してください。

次の表は、certificate レルムのオプションのプロパティーの一覧です。

表 9–5 certificate レルムのオプションのプロパティー

プロパティー	説明
assign-groups	グループ名のコンマで区切られたリスト。有効な証明書を提出するすべてのクライアントがこのグループに割り当てられます。たとえば、employee,manager など。この場合、これらはユーザーグループの名前です。
jaas-context	このレルムに使用するログインモジュールのタイプ。certificate レルムでは、この値は必ず certificateRealm にする必要があります。

file および admin-realm レルムの編集

サーバーは、file レルムの keyfile および admin-realm レルムの admin-keyfile という名前のファイルに、すべてのユーザー、グループ、およびパスワードの情報を保持します。どちらの場合も、file プロパティーを使って keyfile の場所を指定します。次の表に、file レルムに必要なプロパティーを示します。

表 9–6 file レルムに必要なプロパティー

プロパティー名	説明	デフォルト値
file	keyfile のフルパスおよび名前。	*domain-dir*/config/keyfile
jaas-context	このレルムに使用するログインモジュールのタイプ。	fileRealm だけが有効な値です。

keyfile は最初は空のため、file レルムを使用する前にユーザーを追加する必要があります。手順については、180 ページの「file レルムユーザーの管理」を参照してください。

admin-keyfile には最初、管理ユーザー名、暗号形式の管理パスワード、およびデフォルトで asadmin であるこのユーザーが属するグループが収められています。admin-realm にユーザーを追加する方法の詳細については、170 ページの「管理ツールへのアクセスを許可する」を参照してください。

注 – admin-realm のグループ asadmin のユーザーには、管理コンソールおよび asadmin ツールを使用する権限があります。このグループには、サーバーの管理権限のあるユーザーだけを追加してください。

NSS (Network Security Services) によるユーザーの管理

Enterprise Edition の場合にのみ、180 ページの「file レルムユーザーの管理」で説明したように管理コンソールを使ってユーザーを管理することもできますし、NSS ツールを使ってユーザーを管理することもできます。NSS (Network Security Services) とは、セキュリティーが有効なクライアントおよびサーバーアプリケーションのクロスプラットフォーム開発をサポートするよう設計された一連のライブラリです。NSS で構築されたアプリケーションは、SSL v2 および v3、TLS、PKCS #5、PKCS #7、PKCS #11、PKCS #12、S/MIME、X.509 v3 証明書およびその他のセキュリティー標準をサポートできます。詳細については、次の URL を参照してください。

- NSS (Network Security Services) については、http://www.mozilla.org/projects/security/pki/nss/
- NSS セキュリティーツールについては、http://www.mozilla.org/projects/security/pki/nss/tools/
- NSS の概要については、http://www.mozilla.org/projects/security/pki/nss/overview.html

file レルムユーザーの管理

file レルムユーザーは管理コンソールで管理します。file レルムのユーザーおよびグループは keyfile で表示され、その場所は file プロパティーで指定されます。

注 – この手順を使用して、admin-realm を含む任意の file レルムにユーザーを追加することもできます。この節で言及されている file レルムの代わりに、ターゲットレルムの名前を代入するだけです。

file レルムの各ユーザーは、特定の「J2EE グループ」に所属できます。J2EE グループは、共通の特性に基づいて分類されるユーザーのカテゴリです。たとえば、E コマースアプリケーションの顧客は CUSTOMER グループに属しますが、お得意様は PREFERRED グループに属します。ユーザーをグループに分類すると、ユーザーからの大量のアクセスを制御することが容易になります。

Application Server のインストール後の当初は、ユーザーはインストールの際に入力した管理者だけです。デフォルトで、このユーザーは Application Server を変更する権限を付与する admin-realm レルムの asadmin グループに属します。このグループに割り当てられるすべてのユーザーは、管理者権限が付与されます。つまり、asadmin ツールおよび管理コンソールへのアクセス権があります。

file レルムのユーザーを管理するには、次の各タスクを実行します。

- 181 ページの「「ファイルユーザー」ページにアクセスする」
- 182 ページの「ユーザーを追加する」
- 182 ページの「ユーザー情報を編集する」
- 183 ページの「ユーザーを削除する」

▼ 「ファイルユーザー」ページにアクセスする

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
   - すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。
3. 「セキュリティー」ノードを展開します。
4. 「レルム」ノードを展開します。
5. file ノードを選択します。
6. 「レルムを編集」ページで、「ユーザーを管理」ボタンをクリックします。
   「ファイルユーザー」ページが表示されます。このページで、次のタスクを実行します。

- 182 ページの「ユーザーを追加する」
- 182 ページの「ユーザー情報を編集する」
- 183 ページの「ユーザーを削除する」

▼ ユーザーを追加する

1. 「新規」をクリックして、新しいユーザーを `file` レルムに追加します。
2. 「ファイルユーザー」ページで次の情報を入力します。
   - ユーザー **ID** *(必須)* - ユーザーの名前。
   - パスワード *(必須)* - ユーザーのパスワード。
   - パスワードの再入力 *(必須)* - ユーザーのパスワードの確認用再入力。
   - グループリスト *(オプション)* - ユーザーが属するグループのコンマで区切られたリスト。これらのグループをほかの場所で定義する必要はありません。
3. 「了解」をクリックして、このユーザーを `file` レルムのユーザーのリストに追加します。「取消し」をクリックすると保存せずに終了します。

参考 同機能を持つ asadmin コマンド

`create-file-user`

▼ ユーザー情報を編集する

1. 「ユーザー **ID**」列で、変更するユーザーの名前をクリックします。
   「ファイルレルムユーザーの編集」ページが表示されます。
2. 「パスワード」および「パスワードの確認」フィールドに新しいパスワードを入力して、ユーザーのパスワードを変更します。
3. 「グループ リスト」フィールドのグループを追加または削除して、ユーザーが属するグループを変更します。
   グループ名をコンマで区切ってください。グループを事前に定義する必要はありません。
4. 「保存」をクリックして、このユーザーを `file` レルムのユーザーのリストに保存します。
   「閉じる」をクリックすると保存せずに終了します。

▼ ユーザーを削除する

1. 削除するユーザーの名前の左側にあるチェックボックスを選択します。
2. 「削除」をクリックします。
3. 「閉じる」をクリックすると「レルムを編集」ページに戻ります。

参考 同機能を持つ asadmin コマンド

`delete-file-user`

相互認証の設定

- 183 ページの「アプリケーションの相互 SSL 認証の有効化」
- 183 ページの「すべてのアプリケーションの相互認証を有効にする」

相互認証では、サーバーとクライアント側の両方で認証が有効です。相互認証をテストするには、有効な証明書を持つクライアントが存在している必要があります。相互認証の詳細については、『J2EE 1.4 Tutorial』(`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`) の「Security」の章を参照してください。

アプリケーションの相互 SSL 認証の有効化

特定のアプリケーションで相互認証を有効にするには、`deploytool` を使用して認証のメソッドを `Client-Certificate` に設定してください。`deploytool` の使用方法の詳細については、『J2EE 1.4 Tutorial』(`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`) の「Security」の章を参照してください。

▼ すべてのアプリケーションの相互認証を有効にする

Application Server は、HTTPS 認証に `certificate` レルムを使用します。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「レルム」ノードを展開します。

5 certificate レルムを選択します。

6 「プロパティーを追加」ボタンをクリックします。

a. 「名前」フィールドに、clientAuth を入力します。

b. 「値」フィールドに、true を入力します。

7 「保存」をクリックします。

8 コンソールに「再起動が必要です」と表示される場合は、**Application Server** を再起動します。

サーバーの再起動の後、certificate レルムを使用するすべてのアプリケーションでクライアント認証が必要になります。

JACC プロバイダに関する管理コンソールタスク

▼ JACC プロバイダを作成する

JACC (Java Authorization Contract for Containers) は J2EE 1.4 仕様の一部で、プラグイン可能な承認プロバイダ用のインタフェースを定義しています。これによって、管理者は認証を行うためにサードパーティー製の「プラグイン」モジュールを設定できます。デフォルトで、Application Server は JACC に準拠する単純なファイルベースの承認エンジンを提供します。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。

- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「JACC プロバイダ」ノードを選択します。

5 「JACC プロバイダ」ページで、「新規」をクリックします。

6 「JACC プロバイダを作成」ページで、次を入力します。

- 名前 – このプロバイダの識別に使用する名前。
- ポリシーの設定 – ポリシー設定ファクトリを実装するクラスの名前。デフォルトプロバイダは、com.sun.enterprise.security.provider.PolicyConfigurationFactoryImpl を使用します。
- ポリシープロバイダ – ポリシーファクトリを実装するクラスの名前。デフォルトプロバイダは、com.sun.enterprise.security.provider.PolicyWrapper を使用します。

7 「プロパティーを追加」ボタンをクリックして、プロバイダにプロパティーを追加します。有効なプロパティーは次のとおりです。

- repository – ポリシーファイルを含むディレクトリ。デフォルトプロバイダの場合、この値は ${com.sun.aas.instanceRoot}/generated/policy になります。

8 「了解」をクリックしてこの設定を保存するか、「取消し」をクリックして保存しないで終了します。

▼ JACC プロバイダを編集する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「JACC プロバイダ」ノードを展開します。

5 編集する **JACC** プロバイダのノードを選択します。

6 「**JACC** プロバイダを編集」ページで、必要なプロバイダ情報の変更を行います。

- ポリシーの設定 – ポリシー設定ファクトリを実装するクラスの名前。
- ポリシープロバイダ – ポリシーファクトリを実装するクラスの名前。

7 プロパティーを追加するには、「追加」ボタンをクリックします。プロパティーの名前および値を入力します。有効なエントリは次のとおりです。

- `repository` – ポリシーファイルを含むディレクトリ。デフォルトプロバイダの場合、この値は `${com.sun.aas.instanceRoot}/generated/policy` になります。

8 既存のプロパティーを削除するには、プロパティーの左側のチェックボックスをクリックして、「プロパティーを削除」をクリックします。

9 「保存」をクリックして保存するか、ブラウザの「戻る」ボタンをクリックして保存しないで取り消します。

▼ JACC プロバイダを削除する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「**JACC** プロバイダ」ノードを選択します。

5 削除する **JACC** プロバイダの左側のチェックボックスをクリックします。

6 「削除」をクリックします。

▼ 有効な JACC プロバイダを設定する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを選択します。
「セキュリティー」ページが表示されます。

4 「**JACC**」フィールドに、サーバーが使用する **JACC** プロバイダの名前を入力します。
使用可能な JACC プロバイダが分からない場合は、ツリーの「JACC プロバイダ」コンポーネントを展開して、設定されたすべての JACC プロバイダを表示します。

5 「保存」を選択して変更を保存するか、または「デフォルトを読込み」を選択してデフォルト値を復元します。

6 コンソールに「再起動が必要です」と表示される場合は、**Application Server** を再起動します。

監査モジュールに関する管理コンソールタスク

▼ 監査モジュールを作成する

Application Server は単純なデフォルト監査モジュールを提供します。詳細については、191 ページの「デフォルト監査モジュールを使用する」を参照してください。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「監査モジュール」ノードを選択します。

5 「監査モジュール」ページで、「新規」をクリックします。

6 「監査モジュールを作成」ページで、次の情報を入力します。

- 名前 – この監査モジュールの識別に使用する名前。
- クラス名 – このモジュールを実装するクラスの完全修飾名。デフォルトの監査モジュールのクラス名は、com.sun.enterprise.security.Audit です。

7 このモジュールに **JVM** プロパティーを追加するには、「プロパティーを追加」をクリックします。各プロパティーの名前および値を指定します。有効なプロパティーは次のとおりです。

- auditOn - この実装クラスを有効にするかどうかを指定します。有効な値は true および false です。

8 「了解」をクリックしてエントリを保存するか、「取消し」をクリックして保存しないで終了します。

▼ 監査モジュールを編集する

監査モジュールはデフォルトではオンになりません。監査モジュールを有効にする方法の詳細については、190 ページの「監査ログを有効または無効にする」を参照してください。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。

- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「監査モジュール」ノードを展開します。

5 編集する「監査モジュール」ノードをクリックします。

6 「監査モジュールを編集」ページで、必要に応じてクラス名を変更します。

7 「追加」ボタンを選択して、プロパティーの名前と値を入力し、モジュールの任意のプロパティーをさらに入力します。有効なプロパティーは次のとおりです。

- auditOn - この監査モジュールを使用するかどうかを指定します。有効な値は true および false です。

8 変更する名前または値を選択して、変更を直接テキストフィールドに入力し、任意の既存のプロパティーを変更します。

9 プロパティーの左側のチェックボックスを選択して、「プロパティーを削除」をクリックし、プロパティーを削除します。

10 「保存」をクリックして保存するか、ブラウザの「戻る」ボタンをクリックして保存しないで取り消します。

▼ 監査モジュールを削除する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。

- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「監査モジュール」ノードを選択します。

5 削除する監査モジュールの左側のチェックボックスをクリックします。

6 「削除」をクリックします。

▼ 監査ログを有効または無効にする

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを選択します。
「セキュリティー」ページが表示されます。

4 ログを有効にするには、「監査ログ」チェックボックスを選択します。ログを無効にするには、選択を解除してください。
このオプションを選択すると、監査モジュールが読み込まれ、Application Server の監査ライブラリによって監査モジュールが監査ポイントで確実に呼び出されます。

5 監査ログを有効にする場合には、**190** ページの「有効な監査モジュールを設定する」の説明に従ってデフォルトの監査モジュールを指定します。

6 「保存」を選択して変更を保存します。

7 コンソールに「再起動が必要です」と表示される場合は、**Application Server** を再起動します。

▼ 有効な監査モジュールを設定する

始める前に サーバーが使用する監査モジュールを指定するには、まず、190 ページの「監査ログを有効または無効にする」の説明に従って監査ログを有効にします。

1 「監査モジュール」フィールドに、サーバーが使用する監査モジュールの名前を入力します。
事前に設定された監査モジュールは `default` と呼ばれます。この監査モジュールの `auditOn` が `true` に設定されていることを確認してください。その方法は、191 ページの「デフォルト監査モジュールを使用する」で説明しています。

2 「保存」を選択して変更を保存するか、「デフォルトを読込み」を選択して取り消します。

3 コンソールに「再起動が必要です」と表示される場合は、**Application Server** を再起動します。

▼ デフォルト監査モジュールを使用する

`default` 監査モジュールは、認証および承認の要求をサーバーログファイルに記録します。ログファイルの場所を変更する方法については、272 ページの「ログの一般設定を設定する」を参照してください。

認証ログエントリには、次の情報が含まれています。

- 認証を試みたユーザーの名前。
- アクセス要求を処理したレルム。
- 要求された Web モジュール URI または EJB コンポーネント。
- 要求の成功または失敗。

監査ログが有効かどうかにかかわらず、Application Server はすべての拒否認証イベントを記録します。

承認ログエントリには、次の情報が含まれています。

- 承認されたユーザーの名前 (ある場合)。
- 要求された Web URI または EJB コンポーネント。
- 要求の成功または失敗。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「セキュリティー」ノードを展開します。

4 「監査モジュール」ノードを展開します。

5 `default` ノードをクリックします。

6 `auditOn` プロパティーの値を `true` に設定します。

7 「保存」を選択して変更を保存します。

8 コンソールに「再起動が必要です」と表示される場合は、**Application Server** を再起動します。

リスナーと JMX コネクタに関する管理コンソールタスク

▼ HTTP リスナーのセキュリティーを設定する

HTTP サービスの各仮想サーバーは、1 つまたは複数の「HTTP リスナー」を介してネットワーク接続を提供します。管理コンソールで、新しい HTTP リスナーを作成し、既存の HTTP リスナーのセキュリティー設定を編集します。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「**HTTP** サービス」ノードを展開します。

4 「**HTTP** リスナー」ノードを選択します。

5 特定の **HTTP** リスナーを選択してその既存リスナーを編集するか、「新規」をクリックし、**255** ページの「**HTTP** リスナーを作成する」の手順に従って新しいリスナーを作成します。

6 **194** ページの「リスナーのセキュリティープロパティーを設定する」の手順に従ってセキュリティープロパティーを設定します。

7 「保存」をクリックして変更を保存するか、ブラウザの「戻る」ボタンをクリックして保存しないで取り消します。

参考 同機能を持つ asadmin コマンド

`create-http-listener`

▼ IIOP リスナーのセキュリティーを設定する

Application Server は、CORBA (Common Object Request Broker Architecture) オブジェクトをサポートしており、これはネットワークを介しての通信をするために IIOP (Internet Inter-Orb Protocol) を使用します。「IIOP リスナー」は、EJB コンポーネントのリモートクライアントおよびほかの CORBA ベースのクライアントから受信する接続を受け付けます。管理コンソールで、新しい IIOP リスナーを作成し、既存の IIOP リスナーの設定を編集します。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。
3. 「**ORB**」ノードを展開します。
4. 「**IIOP** リスナー」ノードを選択します。
5. 特定の **IIOP** リスナーを選択してそのリスナーを編集するか、「新規」をクリックし、261 ページの「**IIOP** リスナーを作成する」の手順に従って新しいリスナーを作成します。
6. 194 ページの「リスナーのセキュリティープロパティーを設定する」の手順に従ってセキュリティープロパティーを設定します。
7. 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」をクリックしてプロパティーのデフォルト値を復元します。

   新しいリスナーが作成されると、「IIOP リスナー」ページの「現在のリスナー」テーブルに、そのリスナーが表示されます。

参考 同機能を持つ asadmin コマンド

`create-iiop-listener`

▼ 管理サービスの JMX コネクタのセキュリティーを設定する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを展開します。

3 「管理サービス」ノードを展開します。

4 変更する管理サービスを選択します。

5 194 ページの「リスナーのセキュリティープロパティーを設定する」の手順に従ってセキュリティープロパティーを設定します。

6 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」をクリックしてプロパティーのデフォルト値を復元します。

▼ リスナーのセキュリティープロパティーを設定する

この手順は、HTTP リスナー、IIOP リスナー、および JMX コネクタのセキュリティープロパティーに適用されます。

1 「**HTTP** リスナーを編集」、「**IIOP** リスナーを編集」、または「**JMX** コネクタを編集」ページで、「**SSL**」というセクションに進みます。

2 「セキュリティー」フィールドの「有効」ボックスにチェックマークを付けて、このリスナーのセキュリティーを有効にします。このオプションを選択すると、有効にするセキュリティーのタイプを指定するため **SSL3** または **TLS** を選択して、証明書のニックネームを入力する必要があります。

3 このリスナーを使用する際、**Application Server** への認証を個々のクライアントに任せる場合は、「クライアント認証」フィールドの「有効」ボックスにチェックマークを付けます。

4 「有効」ボックスにチェックマークが付いている場合は、「証明書のニックネーム」フィールドにキーストアエイリアスを入力します。キーストアエイリアスは、既存のサーバーのキーペアと証明書を識別する単一の値です。デフォルトキーストアの証明書のニックネームは、`s1as` です。

「証明書のニックネーム」を検索するには、206 ページの「NSS (Network Security Services) ツールの使用」の説明に従って `certutil` ユーティリティーを使用します。

5 「有効」ボックスにチェックマークが付いている場合は、**SSL3** および **TLS** またはそのいずれかを選択します。デフォルトでは、**SSL3** と **TLS** のどちらも有効です。

6 必要に応じて、個別の暗号化方式群を有効にします。デフォルトでは、サポートされるすべての暗号化方式群が有効です。暗号化方式については、**165** ページの「暗号化方式について」を参照してください。

7 「保存」を選択して変更を保存するか、「デフォルトを読込み」を選択して取り消します。

仮想サーバーに関する管理コンソールセキュリティータスク

▼ シングルサインオン (SSO) を設定する

シングルサインオンによって、複数のアプリケーションがユーザーのサインオン情報を共有できるため、ユーザーはアプリケーションごとにサインオンする必要がなくなります。シングルサインオンを使用するアプリケーションでは、一度ユーザーを認証すると、その認証情報はその他のすべての関連アプリケーションに伝達されます。

シングルサインオンは、同じレルムおよび仮想サーバーに設定された Web アプリケーションに適用されます。

注 – シングルサインオンは、HTTP Cookie を使用して、各要求を保存されたユーザー ID に関連付けるトークンを送信するので、ブラウザクライアントが Cookie をサポートしている場合にのみ使用できます。

シングルサインオンは、次の規則に従って動作します。

- ユーザーが Web アプリケーションの保護対象リソースにアクセスする場合、サーバーはユーザーに、Web アプリケーションで定義されているメソッドを使用してユーザー自身を認証するよう要求します。

- 一度認証されると、Application Server は仮想サーバーの Web アプリケーション全体の承認決定用のユーザーに関連付けられたロールを使用するので、ユーザーを個別のアプリケーションごとに認証する必要はありません。
- ユーザーが 1 つの Web アプリケーションから明示的に、またはセッションの有効期限が切れたためにログアウトすると、すべての Web アプリケーションで、そのユーザーのセッションが無効になります。その後ユーザーが任意のアプリケーションの保護対象リソースにアクセスするには、ログインする必要があります。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを展開します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを展開します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを展開します。

3 「**HTTP** サービス」ノードを展開します。

4 「仮想サーバー」ノードを展開し、シングルサインオンのサポートのために設定が必要な仮想サーバーを選択します。

5 「プロパティーを追加」をクリックします。
空白のプロパティーエントリがリストの下部に追加されます。

6 「名前」フィールドに、`sso-enable` を入力します。

7 「値」フィールドに `false` を入力すると **SSO** が無効になり、`true` を入力すると **SSO** が有効になります。
デフォルトで、SSO は有効です。

8 「プロパティーを追加」をクリックし、該当する任意の **SSO** プロパティーを設定することで、その他のシングルサインオンのプロパティーの追加または変更を行います。
次の表では、仮想サーバーの有効な SSO プロパティーについて説明します。

プロパティー名	説明	値
sso-max-inactive-seconds	クライアントが活動を停止後、ユーザーのシングルサインオンの記録を削除可能にするまでの秒数。仮想サーバーの任意のアプリケーションへアクセスすることで、シングルサインオンの記録は有効なまま保持されます。	デフォルトは 300 秒 (5 分) です。値を大きくすると、ユーザーの持続時間も長くなりますが、サーバー上のメモリーの消費量も増大します。
sso-reap-interval-seconds	有効期限が切れたシングルサインオンの記録の削除を行う間隔 (秒単位)。	デフォルトは 60 です。

9 「保存」をクリックします。

10 コンソールに「再起動が必要です」と表示される場合は、**Application Server** を再起動します。

コネクタ接続プールに関する管理コンソールタスク

コネクタ接続プールについて

「コネクタモジュール」は、リソースアダプタとも呼ばれ、J2EE アプリケーションが EIS (Enterprise Information System) と対話することを可能にします。「コネクタリソース」はアプリケーションに EIS への接続を提供します。「コネクタ接続プール」とは、特定の EIS のための再利用可能な接続のグループです。

「セキュリティーマップ」は、J2EE ユーザーおよびグループと EIS ユーザーおよびグループ間のマッピングの作成を可能にします。管理コンソールを使用して、コネクタ接続プールのセキュリティーマップの作成、更新、一覧表示、および削除を行ってください。

注 – このコンテキストでは、ユーザーは主体と呼ばれます。EIS (Enterprise Information System) は情報を保持する任意のシステムです。これは、メインフレーム、メッセージングシステム、データベースシステム、またはアプリケーションのいずれかになります。

セキュリティーマップについて

セキュリティーマップを使用して、コンテナ管理トランザクションベースのシナリオで、アプリケーション (主体またはユーザーグループ) の呼び出し側 ID を適切な EIS 主体に割り当ててください。アプリケーション主体が EIS に要求を開始すると、アプリケーションサーバーは最初に、マッピングされたバックエンドのEIS 主体を特定するために、コネクタ接続プールに定義されたセキュリティーマップを使用して正確な主体を確認します。完全に一致するものがない場合、アプリケーションサーバーは、ワイルドカード文字の指定があればそれを使用して、マッピングされたバックエンドの EIS 主体を特定します。セキュリティーマップは、アプリケーションユーザーが、EIS の特定の ID として実行することを要求される EIS 動作を実行する必要がある場合に使用されます。

セキュリティーマップを管理するには、管理コンソールで次の各手順を実行してください。

- 198 ページの「セキュリティーマップを作成する」
- 199 ページの「セキュリティーマップを編集する」
- 200 ページの「セキュリティーマップを削除する」

▼ セキュリティーマップを作成する

コネクタ接続プールのセキュリティーマップは、アプリケーションユーザーおよびグループ (主体) を EIS 主体に割り当てます。アプリケーションユーザーが EIS の特定の ID を必要とする EIS 動作の実行を必要とする場合は、セキュリティーマップを使用してください。

1. 「リソース」ノードを展開します。
2. 「コネクタ」ノードを展開します。
3. 「コネクタ接続プール」ノードを選択します。
4. 特定のコネクタ接続プールを選択するために現在のプールのリストからその名前を選択するか、新しいコネクタ接続プールを作成するために現在のプールのリストから「新規」を選択し、**88** ページの「**JDBC** 接続プールを作成する」の手順に従います。
5. 「セキュリティーマップ」ページを選択します。
6. 「新規」をクリックして、新しいセキュリティーマップを作成します。
7. 「セキュリティーマップ」ページで、次のプロパティーを入力します。
   - 名前 – この特定のセキュリティーマップの参照に使用する名前を入力します。
   - ユーザーグループ – 適切な EIS 主体にマッピングされるアプリケーションの呼び出し側 ID。アプリケーション固有のユーザーグループのコンマで区切られたリストを入力するか、またはすべてのユーザーやすべてのユーザーグループを指定するワイルド

カードアスタリスク (*) を入力します。「主体」オプションまたは「ユーザーグループ」オプションを指定しますが、両方は指定しません。

- 主体 – 適切な EIS 主体にマッピングされるアプリケーションの呼び出し側 ID。アプリケーション固有の主体のコンマで区切られたリストを入力するか、またはすべての主体を指定するワイルドカードアスタリスク (*) を入力します。「主体」オプションまたは「ユーザーグループ」オプションを指定しますが、両方は指定しません。

8 「バックエンド主体」セクションで、次のプロパティーを入力します。

- ユーザー名 – EIS のユーザー名を入力します。EIS (Enterprise Information System) は情報を保持する任意のシステムです。これは、メインフレーム、メッセージングシステム、データベースシステム、またはアプリケーションである可能性があります。
- パスワード – EIS ユーザーのパスワードを入力します。

9 「了解」をクリックしてセキュリティーマップを作成するか、「取消し」をクリックして保存しないで取り消します。

参考 同機能を持つ asadmin コマンド

`create-connector-security-map`

▼ セキュリティーマップを編集する

1 「リソース」ノードを展開します。

2 「コネクタ」ノードを展開します。

3 「コネクタ接続プール」ノードを選択します。

4 現在のプールのリストから名前を選択することで、「コネクタ接続プール」を選択します。

5 「セキュリティーマップ」ページを選択します。

6 「セキュリティーマップ」ページで、現在のセキュリティーマップのリストからセキュリティーマップを選択します。

7 「セキュリティーマップを編集」ページで、必要に応じて次のプロパティーを変更します。

- ユーザーグループ – 適切な EIS 主体にマッピングされるアプリケーションの呼び出し側 ID。アプリケーション固有のユーザーグループのコンマで区切られたリストを入力するか、またはすべてのグループやすべてのユーザーグループを指定するワイルドカードアスタリスク (*) を入力します。「主体」オプションまたは「ユーザーグループ」オプションを指定しますが、両方は指定しません。

- 主体 – 適切な EIS 主体にマッピングされるアプリケーションの呼び出し側 ID。アプリケーション固有の主体のコンマで区切られたリストを入力するか、またはすべての主体を指定するワイルドカードアスタリスク (*) を入力します。「主体」オプションまたは「ユーザーグループ」オプションを指定しますが、両方は指定しません。

8 「バックエンド主体」セクションで、次のプロパティーを入力します。

- ユーザー名 – EIS のユーザー名を入力します。EIS (Enterprise Information System) は情報を保持する任意のシステムです。これは、メインフレーム、メッセージングシステム、データベースシステム、またはアプリケーションである可能性があります。
- パスワード – EIS ユーザーのパスワードを入力します。

9 「保存」をクリックして、セキュリティーマップの変更を保存します。

参考 有用な asadmin コマンド

`list-connector-security-maps` および `update-connector-security-maps`

▼ セキュリティーマップを削除する

1 「リソース」ノードを展開します。

2 「コネクタ」ノードを展開します。

3 「コネクタ接続プール」ノードを選択します。

4 現在のプールのリストから名前を選択することで、「コネクタ接続プール」を選択します。

5 「セキュリティーマップ」ページを選択します。

6 「セキュリティーマップ」ページで、削除するセキュリティーマップの名前の左側にあるチェックボックスをクリックします。

7 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-connector-security-map`

証明書と SSL の操作

証明書ファイルについて

Application Server をインストールすると、内部テストに適した NSS (Network Security Services) 形式のデジタル証明書が生成されます。デフォルトでは、Application Server は *domain-dir*/config ディレクトリの 証明書データベースに、証明書情報を格納します。

- キーストアファイル。key3.db には、非公開鍵を含む Application Server の証明 書が格納されます。キーストアファイルはパスワードで保護されます。パスワードを変更するには、asadmin change-master-password コマンドを使用します。certutil の詳細については、207 ページの「certutil ユーティリティーの使用」を参照してください。

  各キーストアエントリには一意のエイリアスがあります。インストール後の Application Server キーストア内には、エイリアス s1as を持つ単一のエントリが含まれています。

- トラストストアファイル。cert8.db には、ほかのエンティティーの公開鍵を含 む Application Server の信頼できる証明書が格納されます。信頼できる証明書では、サーバーは証明書の公開鍵が証明書の所有者に属していることを確認しています。信頼できる証明書には、通常、証明書発行局 (CA) の証明書も含まれています。

  Platform Edition では、サーバー側で、Application Server は keytool を使用して証明書とキーストアを管理する JSSE 形式を使用します。Enterprise Edition では、サーバー側で、Application Server は certutil を使用して非公開鍵と証明書を格納する NSS データベースを管理する NSS を使用します。これらの両方の Edition で、クライアントサイド (アプリケーションクライアントまたはスタンドアロン) では、JSSE 形式を使用します。

  デフォルトで、Application Server は、サンプルアプリケーションで開発目的のために動作するキーストアおよびトラストストアを使用して設定されています。本稼動環境のために、証明書エイリアスを変更し、トラストストアにほかの証明書を追加し、キーストアおよびトラストストアファイルの名前と場所を変更する必要が生ずる可能性があります。

▼ 証明書ファイルの場所を変更する

開発用として提供されているキーストアファイルとトラストストアファイルは、*domain-dir*/config ディレクトリに格納されています。

1. 管理コンソール ツリーコンポーネントで、「設定」ノードを展開します。
2. 「**server-config** (管理設定)」ノードを展開します。

3 「**JVM**設定」ノードを選択します。

4 「**JVM**オプション」タブをクリックします。

5 「**JVM**オプション」ページで、「値」フィールドの次の値を追加または変更して、証明書ファイルの新しい場所を反映させます。

```
-Dcom.sun.appserv.nss.db=${com.sun.aas.instanceRoot}/NSS-database-directory
```

ここで、*NSS-database-directory* はNSSデータベースの場所です。

6 「保存」をクリックします。

7 コンソールに「再起動が必要です」と表示される場合は、**Application Server**を再起動します。

JSSE (Java Secure Socket Extension) ツールの使用

`keytool` を使用して、JSSE (Java Secure Socket Extension) デジタル証明書を設定および操作します。Platform Editionの場合、Application Serverはサーバー側で、JSSE形式を使って証明書とキーストアを管理します。Platform Edition、Enterprise Editionのどちらの場合も、クライアント側 (アプリケーションクライアントまたはスタンドアロン) ではJSSE形式が使用されます。

J2SE SDKに同梱されている `keytool` を使用すれば、管理者は、公開鍵と非公開鍵のペアおよび関連する証明書を管理できます。さらに、ユーザーは、通信接続先の公開鍵を証明書の形式でキャッシュできます。

`keytool` を実行するには、J2SEの `/bin` ディレクトリがパスの中に設定されているか、またはツールへのフルパスがコマンド行に存在するように、シェルの環境を設定する必要があります。`keytool` の詳細については、`keytool` のドキュメント (`http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html`) を参照してください。

keytoolユーティリティーの使用

次の例は、JSSEツールによる証明書処理に関する使用方法を示したものです。

- RSA鍵アルゴリズムを使ってタイプJKSのキーストア内に自己署名付き証明書を作成する。RSAはRSA Data Security, Inc. が開発した公開鍵暗号化技術です。この略語は、この技術の開発者であるRivest、Shamir、およびAdelmanを表しています。

```
keytool  -genkey -noprompt -trustcacerts -keyalg RSA -alias ${cert.alias}
-dname  ${dn.name} -keypass ${key.pass} -keystore ${keystore.file}
-storepass ${keystore.pass}
```

証明書を作成する別の例については、204ページの「`keytool` ユーティリティーを使って証明書を生成する」を参照してください。

- デフォルトの鍵アルゴリズムを使ってタイプJKSのキーストア内に自己署名付き証明書を作成する。

```
keytool -genkey -noprompt -trustcacerts -alias ${cert.alias} -dname
${dn.name} -keypass ${key.pass} -keystore ${keystore.file} -storepass
${keystore.pass}
```

  証明書に署名する例については、205ページの「keytoolユーティリティーを使ってデジタル証明書に署名する」を参照してください。

- タイプJKSのキーストアで利用可能な証明書を表示する。

```
keytool -list -v  -keystore ${keystore.file} -storepass ${keystore.pass}
```

- タイプJKSのキーストア内の証明書情報を表示する。

```
keytool  -list -v  -alias ${cert.alias} -keystore ${keystore.file}
-storepass ${keystore.pass}
```

- RFC/テキスト形式の証明書をJKSストア内にインポートする。証明書は、バイナリエンコーディングではなく、Internet RFC (Request for Comments) 1421標準によって定義された印刷可能なエンコーディング形式を使って格納されることがしばしばあります。Base 64エンコーディングとしても知られるこの証明書形式を使用すれば、電子メールなどの機構を使って証明書をほかのアプリケーションにエクスポートしやすくなります。

```
keytool -import -noprompt -trustcacerts -alias ${cert.alias} -file
${cert.file} -keystore ${keystore.file} -storepass ${keystore.pass}
```

- タイプJKSのキーストア内の証明書をPKCS7形式でエクスポートする。「Public Key Cryptography Standards #7, Cryptographic Message Syntax Standard」によって定義された応答形式には、発行される証明書に加え、それをサポートする証明書チェーンも含まれます。

```
keytool  -export -noprompt  -alias ${cert.alias} -file ${cert.file}
-keystore ${keystore.file} -storepass ${keystore.pass}
```

- タイプJKSのキーストア内の証明書をRFC/テキスト形式でエクスポートする。

```
keytool  -export -noprompt -rfc  -alias ${cert.alias} -file
${cert.file} -keystore ${keystore.file} -storepass ${keystore.pass}
```

- タイプJKSのキーストアから証明書を削除する。

```
keytool  -delete -noprompt -alias ${cert.alias}  -keystore ${keystore.file}
-storepass ${keystore.pass}
```

  キーストアから証明書を削除する別の例については、205ページの「keytoolユーティリティーを使って証明書を削除する」を参照してください。

▼ keytoolユーティリティーを使って証明書を生成する

keytoolを使用して証明書の生成、インポート、およびエクスポートを行います。デフォルトでは、keytoolは実行元のディレクトリにキーストアファイルを作成します。

1. 証明書を実行すべきディレクトリに移動します。

   証明書の生成は常に、キーストアファイルとトラストストアファイルが格納されたディレクトリ(デフォルトでは*domain-dir*/config)内で行います。これらのファイルの場所を変更する方法については、201 ページの「証明書ファイルの場所を変更する」を参照してください。

2. 次のkeytoolコマンドを入力することで、キーストアファイルkeystore.jks内に証明書を生成します。

   ```
   keytool -genkey -alias keyAlias-keyalg RSA
    -keypass changeit
    -storepass changeit
   -keystore keystore.jks
   ```

   *keyAlias*には任意の一意名を指定します。キーストアまたは非公開鍵のパスワードをデフォルト以外の値に変更した場合には、前述のコマンドのchangeitをその新しいパスワードで置き換えてください。

   プロンプトが表示され、keytoolが証明書の生成に使用するユーザーの名前、組織、およびその他の情報の入力を求められます。

3. 次のkeytoolコマンドを入力することで、生成された証明書をファイルserver.cer(またはclient.cerでもよい)にエクスポートします。

   ```
   keytool -export -alias keyAlias-storepass changeit
    -file server.cer
    -keystore keystore.jks
   ```

4. 認証局によって署名された証明書が必要な場合は、**205** ページの「keytoolユーティリティーを使ってデジタル証明書に署名する」を参照してください。

5. トラストストアファイルcacerts.jksを作成し、そのトラストストアに証明書を追加するには、次のkeytoolコマンドを入力します。

   ```
   keytool -import -v -trustcacerts
   -alias keyAlias
    -file server.cer
   -keystore cacerts.jks
    -keypass changeit
   ```

   キーストアまたは非公開鍵のパスワードをデフォルト以外の値に変更した場合には、前述のコマンドのchangeitをその新しいパスワードで置き換えてください。

   このツールは、証明書に関する情報を表示し、その証明書を信頼するかどうかをユーザーに尋ねます。

6 yes と入力し、続いて **Enter** キーを押します。

すると、keytool から次のようなメッセージが表示されます。

```
Certificate was added to keystore
[Saving cacerts.jks]
```

7 **Application Server** を再起動します。

▼ keytool ユーティリティーを使ってデジタル証明書に署名する

デジタル証明書の作成後、所有者はそれに署名して偽造を防止する必要があります。E コマースのサイト、または ID の認証が重要であるサイトは、既知の証明書発行局 (CA) から証明書を購入できます。認証に心配がない場合、たとえば、非公開のセキュアな通信だけが必要な場合などは、CA 証明書の取得に必要な時間と費用を節約して、自己署名付き証明書を使用してください。

1 証明書の鍵のペアを生成するため、**CA** の **Web** サイトの指示に従います。

2 生成された証明書の鍵のペアをダウンロードします。

キーストアファイルとトラストストアファイルが格納されたディレクトリ (デフォルトでは *domain-dir*/config ディレクトリ) 内に、証明書を保存します。201 ページの「証明書ファイルの場所を変更する」を参照してください。

3 使用しているシェルで、証明書を含むディレクトリに変更します。

4 keytool を使用して、証明書をローカルのキーストア、および必要に応じてローカルのトラストストアにインポートします。

```
keytool -import -v -trustcacerts
-alias keyAlias
 -file server.cer
-keystore cacerts.jks
 -keypass changeit
-storepass changeit
```

キーストアまたは非公開鍵のパスワードがデフォルト以外の値である場合には、前述のコマンドの changeit をその新しいパスワードで置き換えてください。

5 **Application Server** を再起動します。

keytool ユーティリティーを使って証明書を削除する

既存の証明書を削除するには、keytool -delete コマンドを使用します。次に例を示します。

```
keytool -delete
 -alias keyAlias
```

```
-keystore keystore-name
-storepass password
```

-delete コマンドで利用可能なオプションの完全な一覧については、http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html の keytool ドキュメントを参照してください。

NSS (Network Security Services) ツールの使用

Enterprise Edition の場合、サーバー側では、NSS (Network Security Services) デジタル証明書を使って非公開鍵と証明書を格納するデータベースを管理します。クライアント側 (アプリケーションクライアントまたはスタンドアロン) では、202 ページの「JSSE (Java Secure Socket Extension) ツールの使用」で説明した JSSE 形式を使用します。

NSS (Network Security Services) を使ってセキュリティーを管理するためのツールは、次のとおりです。

- certutil。証明書および鍵データベースの管理に使用されるコマンド行ユーティリティー。certutil ユーティリティーの使用例については、207 ページの「certutil ユーティリティーの使用」を参照してください。
- pk12util。証明書または鍵データベースと PKCS12 形式のファイル間における鍵と証明書のインポートおよびエクスポートに使用されるコマンド行ユーティリティー。pk12util ユーティリティーの使用例については、208 ページの「pk12util ユーティリティーによる証明書のインポートとエクスポート」を参照してください。
- modutil。secmod.db ファイル内またはハードウェアトークン内の PKCS #11 モジュール情報を管理するためのコマンド行ユーティリティー。modutil ユーティリティーの使用例については、209 ページの「modutil による PKCS11 モジュールの追加と削除」を参照してください。

これらのツールは *install-dir*/lib/ ディレクトリに格納されています。NSS セキュリティーツールの場所を指し示すために、次の各環境変数が使用されます。

- LD_LIBRARY_PATH =${*install-dir*}/lib
- ${*os.nss.path*}

例に含まれる証明書の共通名 (CN) は、クライアントまたはサーバーの名前です。また、この CN は、SSL ハンドシェーク中に証明書の名前とその証明書の生成元であるホスト名とを比較する目的でも使用されます。SSL ハンドシェーク中に証明書名とホスト名が一致しなかった場合、警告または例外が生成されます。いくつかの例では便宜上、証明書の共通名 CN=localhost が使用されていますが、これは、すべてのユーザーが、実際のホスト名に基づいて新しい証明書を作成することなしにその証明書を使用できるようにするためです。

次の各節の例は、NSS ツールによる証明書処理に関する使用方法を示したものです。

- 207 ページの「certutil ユーティリティーの使用」
- 208 ページの「pk12util ユーティリティーによる証明書のインポートとエクスポート」

- 209 ページの「modutil による PKCS11 モジュールの追加と削除」

certutil ユーティリティーの使用

証明書データベースツールの certutil は、Netscape Communicator の cert8.db および key3.db データベースファイルを作成し、変更することができる NSS コマンド行ユーティリティーです。このユーティリティーは、cert8.db ファイルで、証明書の一覧表示、生成、変更、または削除を行い、key3.db ファイルで、パスワードの作成または変更、新しい公開鍵と非公開鍵のペアの生成、鍵データベースのコンテンツの表示、または鍵のペアの削除を行うこともできます。

通常、鍵と証明書の管理プロセスは鍵データベース内の鍵の作成から始まり、証明書データベース内の証明書の生成と管理に続きます。次のドキュメントでは、certutil ユーティリティーの構文を含む、NSS による証明書および鍵データベースの管理について説明していま
す。http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html。

次の箇条書きの各項目は、NSS および JSSE セキュリティーツールを使って証明書の作成または管理、あるいはその両方を行う例を示したものです。

- 自己署名付きのサーバー証明書およびクライアント証明書を生成する。この例では、CN は hostname.domain.[com|org|net|...] の形式でなければなりません。

  この例では、*domain-dir*/config です。serverseed.txt ファイルと clientseed.txt ファイルには、任意のランダムテキストを含めることができます。このランダムテキストは、鍵ペア生成時に使用されます。

  ```
  certutil -S -n $SERVER_CERT_NAME -x -t "u,u,u"
   -s "CN=$HOSTNAME.$HOSTDOMAIN, OU=Java Software, O=Sun Microsystems Inc.,
      L=Santa Clara, ST=CA, C=US"
   -m 25001 -o  $CERT_DB_DIR/Server.crt
   -d $CERT_DB_DIR -f passfile <$CERT_UTIL_DIR/serverseed.txt
  ```

  クライアント証明書を生成する。この証明書も自己署名付き証明書です。

  ```
  certutil -S -n $CLIENT_CERT_NAME  -x -t "u,u,u"
   -s "CN=MyClient, OU=Java Software, O=Sun Microsystems Inc.,
      L=Santa Clara, ST=CA, C=US"
   -m 25002 -o  $CERT_DB_DIR/Client.crt
   -d $CERT_DB_DIR -f passfile <$CERT_UTIL_DIR/clientseed.txt
  ```

- 前述の項目で生成された証明書を検証する。

  ```
  certutil -V -u V -n $SERVER_CERT_NAME -d  $CERT_DB_DIR
  certutil -V -u C -n $CLIENT_CERT_NAME -d  $CERT_DB_DIR
  ```

- 利用可能な証明書を表示する。

  ```
  certutil -L -d  $CERT_DB_DIR
  ```

- RFC テキスト形式の証明書を NSS 証明書データベースにインポートする。

```
certutil -A -a -n ${cert.nickname} -t ${cert.trust.options}
-f ${pass.file} -i ${cert.rfc.file}
-d ${admin.domain.dir}/${admin.domain}/config
```

- NSS 証明書データベース内の証明書を RFC 形式でエクスポートする。

```
certutil -L -a -n ${cert.nickname} -f ${pass.file}
-d ${admin.domain.dir}/${admin.domain}/config > cert.rfc
```

- NSS 証明書データベースから証明書を削除する。

```
certutil -D -n ${cert.nickname} -f ${pass.file}
-d ${admin.domain.dir}/${admin.domain}/config
```

- 証明書を NSS データベースから JKS 形式に移動する。

```
certutil -L -a -n ${cert.nickname}
-d ${admin.domain.dir}/${admin.domain}/config > cert.rfc
keytool  -import -noprompt -trustcacerts -keystore ${keystore.file}
-storepass ${keystore.pass} -alias ${cert.alias} -file cert.rfc
```

pk12util ユーティリティーによる証明書のインポートとエクスポート

証明書または鍵データベースと PKCS12 形式のファイル間における鍵と証明書のインポートおよびエクスポートに使用されるコマンド行ユーティリティーは、`pk12util` です。PKCS12 は、「Public-Key Cryptography Standards (PKCS) #12, Personal Information Exchange Syntax Standard」です。`pk12util` ユーティリティーの詳細については、`http://www.mozilla.org/projects/security/pki/nss/tools/pk12util.html` を参照してください。

- PKCS12 形式の証明書を NSS 証明書データベースにインポートする。

```
pk12util -i ${cert.pkcs12.file} -k ${certdb.pass.file}
-w ${cert.pass.file} -d ${admin.domain.dir}/${admin.domain}/config
```

- PKCS12 形式の証明書を NSS 証明書データベーストークンモジュールにインポートする。

```
pk12util -i ${cert.pkcs12.file} -h ${token.name} -k ${certdb.pass.file}
-w ${cert.pass.file} -d ${admin.domain.dir}/${admin.domain}/config
```

- NSS 証明書データベース内の証明書を PKCS12 形式でエクスポートする。

```
pk12util -o -n ${cert.nickname} -k ${pass.file} -w${cert.pass.file}
-d ${admin.domain.dir}/${admin.domain}/config
```

- NSS 証明書データベーストークンモジュール内の証明書を PKCS12 形式でエクスポートする (ハードウェアアクセラレータ構成で有用)。

```
pk12util -o -n ${cert.nickname} -h ${token.name} -k ${pass.file}
-w ${cert.pass.file} -d ${admin.domain.dir}/${admin.domain}/config
```

- PKCS12 証明書を JKS 形式に変換する (Java ソースが必要)。

```
<target name="convert-pkcs12-to-jks" depends="init-common">
   <delete file="${jks.file}" failonerror="false"/>
   <java classname="com.sun.enterprise.security.KeyTool">
      <arg line="-pkcs12"/>
      <arg line="-pkcsFile ${pkcs12.file}"/>
      <arg line="-pkcsKeyStorePass ${pkcs12.pass}"/>
      <arg line="-pkcsKeyPass ${pkcs12.pass}"/>
      <arg line="-jksFile ${jks.file}"/>
      <arg line="-jksKeyStorePass ${jks.pass}"/>
      <classpath>
         <pathelement path="${s1as.classpath}"/>
         <pathelement path="${env.JAVA_HOME}/jre/lib/jsse.jar"/>
      </classpath>
   </java>
</target>
```

modutil による PKCS11 モジュールの追加と削除

「セキュリティーモジュールデータベースツール」である modutil は、secmod.db ファイル内またはハードウェアトークン内の PKCS #11 (Cryptographic Token Interface Standard) モジュール情報を管理するためのコマンド行ユーティリティーです。このツールを使用すれば、PKCS #11 モジュールの追加と削除、パスワードの変更、デフォルトの設定、モジュールの内容表示、スロットの有効化または無効化、FIPS-140-1 準拠の有効化または無効化、および暗号化操作用デフォルトプロバイダの割り当てを行えます。また、このツールを使用すれば、key3.db、cert7.db、および secmod.db セキュリティーデータベースファイルを作成することもできます。このツールの詳細については、http://www.mozilla.org/projects/security/pki/nss/tools/modutil.html を参照してください。

- 新しい PKCS11 モジュールまたはトークンを追加する。

```
modutil -add ${token.module.name} -nocertdb -force -mechanisms RSA:DSA:RC4:DES
-libfile ${SCA.lib.path} -dbdir ${admin.domain.dir}/${admin.domain}/config
```

- NSS ストアから PKCS11 モジュールを削除する。

```
modutil -delete ${token.module.name} -nocertdb -force -mechanisms RSA:DSA:RC4:DES
-libfile ${SCA.lib.path} -dbdir ${admin.domain.dir}/${admin.domain}/config
```

- NSS ストア内で利用可能なトークンモジュールを一覧表示する。

```
modutil -list  -dbdir ${admin.domain.dir}/${admin.domain}/config
```

詳細情報

- Java 2 Standard Edition のセキュリティーについては、http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html を参照してください。
- 『J2EE 1.4 Tutorial』の「Security」の章については、http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html を参照してください。
- 『管理ガイド』の第 10 章。
- 『Developer's Guide』の「Securing Applications」の章。

◆ ◆ ◆ 第 10 章

10 メッセージセキュリティーの設定

この章では、Application Server で Web サービスのメッセージレイヤーセキュリティーを設定する方法について説明します。この章では、次の項目について説明します。

- 151 ページの「Application Server のセキュリティーについて」
- 169 ページの「セキュリティーに関する管理コンソールタスク」

この章の一部の内容は、セキュリティーと Web サービスに関する基本概念の理解を前提としてます。この章を読む前にこれらの概念について学ぶには、210 ページの「詳細情報」に記載されているリソースを参照してください。

メッセージセキュリティーについて

- 151 ページの「セキュリティーの概要」
- 212 ページの「Application Server のメッセージセキュリティーの理解」
- 216 ページの「Web サービスのセキュリティー保護」
- 217 ページの「サンプルアプリケーションのセキュリティー保護」
- 218 ページの「メッセージセキュリティーのための Application Server の設定」

メッセージセキュリティーの概要

「メッセージセキュリティー」を使用する場合、メッセージ内にセキュリティー情報が挿入され、その情報がメッセージとともにネットワークレイヤー経由でメッセージの送信先に届けられます。メッセージセキュリティーは、『J2EE 1.4 Tutorial』の「Security」の章で説明されているトランスポートレイヤーセキュリティーとは異なり、メッセージトランスポートからメッセージ保護を分離して伝送後もメッセージを保護されたままにするために使用することができます。

「Web Services Security: SOAP Message Security (WS-Security)」は、米国 Sun Microsystems, Inc. を含むすべての主要な Web サービステクノロジプロバイダによって共同開発された、相互運用可能な Web サービスセキュリティーを実現するための OASIS 国際標準で

す。WS-Security のメッセージセキュリティーメカニズムは、SOAP 経由で送信される Web サービスメッセージを XML 暗号化と XML デジタル署名を使ってセキュリティー保護する、というものです。WS-Security 仕様には、X.509 証明書、SAML アサーション、ユーザー名/パスワードなどの各種セキュリティートークンを使って SOAP Web サービスメッセージの認証および暗号化を実現する方法が規定されています。

WS-Security 仕様については、
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf を参照してください。

Application Server のメッセージセキュリティーの理解

Application Server は、Web サービスのクライアント側コンテナとサーバー側コンテナにおいて、WS-Security 標準に対する統合化されたサポートを提供します。この機能は統合化されているため、Application Server のコンテナがアプリケーションに代わって Web サービスセキュリティーを適用します。また、そうしたセキュリティーで Web サービスアプリケーションを保護する際、アプリケーションの実装を変更する必要はありません。Application Server は、これを実現する目的で、SOAP レイヤーメッセージセキュリティープロバイダとメッセージ保護ポリシーを、コンテナおよびコンテナ内に配備されたアプリケーションにバインドする機能を提供しています。

メッセージセキュリティーの責任の割り当て

Application Server でメッセージセキュリティー設定の主要責任者として期待されるのは、157 ページの「システム管理者」ロールと156 ページの「アプリケーション配備担当者」ロールです。場合によっては、156 ページの「アプリケーション開発者」もその責任の一端を担うことがありますが、通常は、システム管理者またはアプリケーション配備者のいずれかのロールが既存アプリケーションをセキュリティー保護し、開発者が関与することも、実装が変更されることもありません。次の各節では、各種ロールの責任を定義します。

- 157 ページの「システム管理者」
- 156 ページの「アプリケーション配備担当者」
- 156 ページの「アプリケーション開発者」

システム管理者

システム管理者は次の責任を負います。

- Application Server 上のメッセージセキュリティープロバイダの設定。
- ユーザーデータベースの管理。
- キーストアおよびトラストストアファイルの管理。
- 暗号化を使用し、バージョン 1.5.0 より前のバージョンの Java SDK を実行している場合の JCE (Java Cryptography Extension) プロバイダの設定。

- サンプルサーバーのインストール。ただし、これを行うのは、xms サンプルアプリケーションを使ってメッセージレイヤー Web サービスセキュリティーの使用方法を示す場合だけです。

システム管理者は、管理コンソールを使用してサーバーセキュリティーの設定を管理し、コマンド行ツールを使用して証明書データベースを管理します。Platform Edition の証明書と非公開鍵は、キーストア内に格納され、keytool を使って管理されます。Standard Edition と Enterprise Edition の証明書と非公開鍵は、NSS データベース内に格納され、certutil を使って管理されます。このマニュアルは主にシステム管理者を対象にしています。メッセージセキュリティータスクの概要については、218 ページの「メッセージセキュリティーのための Application Server の設定」を参照してください。

アプリケーション配備担当者

アプリケーション配備担当者は次の責任を負います。

- 必要なすべてのアプリケーション固有メッセージ保護ポリシーをアプリケーションアセンブリ時に指定 (それらのポリシーが上流行程の役割 (開発者またはプログラマ) によって指定されていなかった場合)。
- Sun 固有の配備記述子を変更し、アプリケーション固有メッセージ保護ポリシー情報 (message-security-binding 要素) を Web サービスエンドポイントとサービス参照に指定。

これらのセキュリティータスクについては、『Developers' Guide』の「Securing Applications」の章で説明されています。この章へのリンクについては、210 ページの「詳細情報」を参照してください。

アプリケーション開発者

アプリケーション開発者はメッセージセキュリティーを有効にできますが、そのようにする責任はありません。メッセージセキュリティーの設定をシステム管理者が行う場合、すべての Web サービスがセキュリティー保護されます。コンテナにバインドされているプロバイダまたは保護ポリシーと異なるものをアプリケーションにバインドする必要がある場合、アプリケーション配備担当者がメッセージセキュリティーの設定を行います。

アプリケーション開発者またはプログラマは次の責任を負います。

- アプリケーション固有メッセージ保護ポリシーがアプリケーションで必要かどうかの判断。必要な場合、その必要なポリシーがアプリケーションアセンブリで指定されているかどうかの確認。それにはアプリケーション配備担当者に連絡します。

セキュリティートークンとセキュリティーメカニズムについて

WS-Security 仕様は、セキュリティートークンを使って SOAP Web サービスメッセージを認証および暗号化するための拡張可能なメカニズムを提供します。Application Server とともにインストールされる SOAP レイヤーメッセージセキュリティープロバイダを使え

ば、ユーザー名 / パスワードセキュリティートークンと X.509 証明書セキュリティートークンによる SOAP Web サービスメッセージの認証と暗号化を行えます。Application Server の今後のリリースでは、SAML アサーションなどのほかのセキュリティートークンを採用したプロバイダも追加される予定です。

ユーザー名トークンについて

Application Server は、SOAP メッセージ内で「ユーザー名トークン」を使ってメッセージ「送信者」の認証 ID を確立します。パスワードが埋め込まれたユーザー名トークンを含むメッセージの受信者は、そのメッセージの送信者がそのトークンによって識別されるユーザーとして振る舞うことを許可されているかどうかを検証するために、その送信者がユーザーの秘密情報 (パスワード) を知っているかどうかを確認します。

ユーザー名トークンを使用する場合、有効なユーザーデータベースを Application Server 上に設定する必要があります。このトピックの詳細については、173 ページの「レルムを編集する」を参照してください。

デジタル署名について

Application Server は、XML デジタル署名を使ってメッセージの「コンテンツ」に認証 ID をバインドします。クライアントはデジタル署名を使用して、呼び出し元 ID を確立します。この方法は、トランスポートレイヤーセキュリティーが使用されている場合に、基本認証または SSL クライアント証明書認証が同じ目的で使用される方法に類似しています。デジタル署名は、メッセージコンテンツのソースを認証するためにメッセージ受信者によって検証されます (このソースはメッセージ送信者と異なる可能性がある。)

デジタル署名を使用する場合、有効なキーストアおよびトラストストアファイルを Application Server 上に設定する必要があります。このトピックの詳細については、201 ページの「証明書ファイルについて」を参照してください。

暗号化について

暗号化の目的は、対象読者だけが理解できるようにデータを変更することです。これは、元のコンテンツを暗号化された要素に置き換えることにより行われます。公開鍵暗号方式に関して言えば、暗号化はメッセージを読み取ることができる関係者の ID を確立するために使用されます。

暗号化を使用する場合は、暗号化をサポートする JCE プロバイダがインストールされている必要があります。このトピックの詳細については、220 ページの「JCE プロバイダを設定する」を参照してください。

メッセージ保護ポリシーについて

メッセージ保護ポリシーは、要求メッセージ処理と応答メッセージ処理に対して定義され、ソース認証または受信者認証に関する要件として表現されます。ソース認証ポリシーは、メッセージを送信したエンティティーまたはメッセージのコンテンツを定義し

たエンティティーの ID がメッセージ内で確立され、その ID をメッセージ受信者が認証できる、という要件を表します。受信者認証ポリシーは、メッセージを受信可能なエンティティーの ID をメッセージ送信者が確立できるようにメッセージが送信される、という要件を表します。プロバイダは、特定のメッセージセキュリティーメカニズムを適用することで、SOAP Web サービスメッセージにおけるメッセージ保護ポリシーを実現します。

要求と応答に対するメッセージ保護ポリシーが定義されるのは、特定のプロバイダがコンテナ内に設定される時です。また、アプリケーションまたはアプリケーションクライアントの Sun 固有の配備記述子内で、アプリケーション固有のメッセージ保護ポリシー (Web サービスのポートまたは操作の粒度でのポリシー) を設定することも可能です。いずれにせよ、メッセージ保護ポリシーを定義する場合、クライアントの要求と応答に対するメッセージ保護ポリシーは、サーバーのそれと一致する (等しい) 必要があります。アプリケーション固有のメッセージ保護ポリシーの定義方法の詳細については、『Developers' Guide』の「Securing Applications」の章を参照してください。

メッセージセキュリティー用語の解説

次に、このマニュアルで使用する用語について説明します。これらの概念については、218 ページの「メッセージセキュリティーのための Application Server の設定」でも説明しています。

- 認証レイヤー

  「認証レイヤー」とは、認証処理を実行する必要のあるメッセージレイヤーです。Application Server は、SOAP レイヤーにおいて Web サービスメッセージセキュリティーを適用します。

- 認証プロバイダ

  Application Server のこのリリースでは、Application Server は、「認証プロバイダ」を呼び出して SOAP メッセージレイヤーセキュリティーを処理します。

  - 「クライアント側プロバイダ」は、署名またはユーザー名/パスワードを使って要求メッセージのソース ID を確立したり、対象の受信者だけがメッセージを参照できるように暗号化を使って要求メッセージを保護したりします。また、クライアント側プロバイダは、受信した応答を正常に復号化することで、その許可された受信者としてコンテナを確立したり、応答内のパスワードまたは署名を検証してその応答に関連付けられたソース ID を認証したりもします。Application Server 内に設定されているクライアント側プロバイダを使えば、ほかのサービスのクライアントとして機能するサーバー側コンポーネント (サーブレットと EJB コンポーネント) によって送信される要求メッセージと受信される応答メッセージを保護することができます。
  - 「サーバー側プロバイダ」は、受信した要求を正常に復号化することで、その許可された受信者としてコンテナを確立したり、要求内のパスワードまたは署名を検証してその要求に関連付けられたソース ID を認証したりします。また、サーバー側プロバイダは、署名またはユーザー名/パスワードを使って応答メッセージのソース ID を確立したり、対象の受信者だけがメッセージを参照できるように暗号化を使って要求メッセージを保護したりもします。「サーバー側プロバイダ」を呼び出すのはサーバー側コンテナだけです。

- デフォルトサーバープロバイダ

  「デフォルトサーバープロバイダ」は、特定のサーバープロバイダがバインドされていない任意のアプリケーションに対して呼び出されるサーバープロバイダを識別するために使用されます。「デフォルトサーバープロバイダ」は「デフォルトプロバイダ」とも呼ばれます。

- デフォルトクライアントプロバイダ

  「デフォルトクライアントプロバイダ」は、特定のクライアントプロバイダがバインドされていない任意のアプリケーションに対して呼び出されるクライアントプロバイダを識別するために使用されます。

- 要求ポリシー

  「要求ポリシー」は、認証プロバイダが実行する要求処理に関連付けられた認証ポリシー要件を定義します。ポリシーはメッセージ送信者の順序で送信されます。この順序では、コンテンツのあとで暗号化するという要件は、メッセージ受信者がメッセージの復号化を行なってから署名の検証を想定することを意味しています。

- 応答ポリシー

  「応答ポリシー」は、認証プロバイダが実行する応答処理に関連付けられた認証ポリシー要件を定義します。ポリシーはメッセージ送信者の順序で送信されます。この順序では、コンテンツのあとで暗号化するという要件は、メッセージ受信者がメッセージの復号化を行なってから署名の検証を想定することを意味しています。

Web サービスのセキュリティー保護

Application Server 上に配備された Web サービスをセキュリティー保護するには、アプリケーションの配備先コンテナ、またはそのアプリケーションがサービスを提供する Web サービスエンドポイントのいずれかに対し、SOAP レイヤーメッセージセキュリティープロバイダとメッセージ保護ポリシーをバインドします。Application Server のクライアント側コンテナで SOAP レイヤーメッセージセキュリティー機能を設定するには、クライアントコンテナ、またはクライアントアプリケーションによって宣言されたポータブルサービス参照のいずれかに対し、SOAP レイヤーメッセージセキュリティープロバイダとメッセージ保護ポリシーをバインドします。

Application Server のインストール時に、SOAP レイヤーメッセージセキュリティープロバイダが Application Server のクライアント側コンテナとサーバー側コンテナ内に設定され、コンテナまたはコンテナ内に配備された個々のアプリケーションまたはクライアントからバインドして利用できるようになります。インストール中、プロバイダにはある単純なメッセージ保護ポリシーが設定されます。このポリシーをコンテナまたはコンテナ内のアプリケーションまたはクライアントにバインドした場合、すべての要求メッセージと応答メッセージに含まれるコンテンツのソースが、XML デジタル署名によって認証されるようになります。

Application Server の管理インタフェースを使えば、既存のプロバイダをバインドして Application Server のサーバー側コンテナから利用できるようにしたり、プロバイダが適用するメッセージ保護ポリシーを変更したり、別のメッセージ保護ポリシーを備えた新し

いプロバイダ設定を作成したりできます。これらの操作については、169 ページの「セキュリティーに関する管理コンソールタスク」で定義しています。アプリケーションクライアントコンテナの SOAP メッセージレイヤーセキュリティー設定でも、これと同様の管理操作を実行できます。それらについては、230 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」で定義しています。

Application Server では、メッセージレイヤーセキュリティーはデフォルトで無効になっています。Application Server のメッセージレイヤーセキュリティーを設定するには、218 ページの「メッセージセキュリティーのための Application Server の設定」に要約されている手順に従ってください。Application Server 上に配備されたすべての Web サービスアプリケーションを Web サービスセキュリティーで保護するには、222 ページの「メッセージセキュリティーのプロバイダを有効にする」の手順に従ってください。

上記の手順 (Application Server の再起動が必要な場合もあり) を実行し終わると、Application Server 上に配備されたすべての Web サービスアプリケーションに Web サービスセキュリティーが適用されるようになります。

アプリケーション固有の Web サービスセキュリティーの設定

アプリケーション固有の Web サービスセキュリティー機能をアプリケーションアセンブリで設定するには、アプリケーションの Sun 固有の配備記述子内で message-security-binding 要素を定義します。これらの message-security-binding 要素は、特定のプロバイダまたはメッセージ保護ポリシーを Web サービスエンドポイントまたはサービス参照に関連付けるために使用されます。また、この要素を修飾することで、それらのプロバイダやポリシーが対応するエンドポイントまたは参照サービスの特定のポートやメソッドに適用されるようにすることも可能です。

アプリケーション固有のメッセージ保護ポリシーの定義方法の詳細については、『Developers' Guide』の「Securing Applications」の章を参照してください。210 ページの「詳細情報」に、この章へのリンクがあります。

サンプルアプリケーションのセキュリティー保護

Application Server には、`xms` という名前のサンプルアプリケーションが付属しています。`xms` アプリケーションは、J2EE EJB エンドポイントと Java サーブレットエンドポイントの両方を使って実装された、単純な Web サービスです。両エンドポイントは同一のサービスエンドポイントインタフェースを共有しています。このサービスエンドポイントインタフェースには、単一の操作 `sayHello` が定義されています。この操作は、文字列引数を 1 つ受け取り、その呼び出し引数の前に `Hello` が付加された `String` を返します。

`xms` サンプルアプリケーションは、Application Server の WS-Security 機能を使って既存の Web サービスアプリケーションをセキュリティー保護する方法を示す目的で提供されています。サンプルに付属する手順では、Application Server の WS-Security 機能を有効にして `xms` アプリケーションを保護する方法が説明されています。また、このサンプルは、WS-Security 機能をアプリケーションに直接バインドする方法 (217 ページの「アプリケーション固有の Web サービスセキュリティーの設定」を参照) も示しています。

xms サンプルアプリケーションは次のディレクトリにインストールされます。
install-dir/samples/webservices/security/ejb/apps/xms/。

xms サンプルアプリケーションのコンパイル、パッケージ化、および実行に関する詳細については、『Developers' Guide』の「Securing Applications」の章を参照してください。

メッセージセキュリティーのための Application Server の設定

要求および応答ポリシー設定のアクション

次の表は、メッセージ保護ポリシーの設定と、その結果として WS-Security SOAP メッセージセキュリティープロバイダによって実行されるメッセージセキュリティー処理を示したものです。

表 10–1 メッセージ保護ポリシーと WS-Security SOAP メッセージセキュリティー処理との対応づけ

メッセージ保護ポリシー	結果として実行される WS-Security SOAP メッセージ保護処理
auth-source="sender"	メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に wsse:UsernameToken (パスワード付き) が格納されます。
auth-source="content"	SOAP メッセージ本体のコンテンツが署名されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内にメッセージ本体の署名が ds:Signature として格納されます。
auth-source="sender" auth-recipient="before-content" または auth-recipient="after-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に wsse:UsernameToken (パスワード付き) と xenc:EncryptedKey が格納されます。xenc:EncryptedKey には、SOAP メッセージ本体の暗号化に使用した鍵が格納されます。この鍵は受信者の公開鍵で暗号化されます。
auth-source="content" auth-recipient="before-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。xenc:EncryptedData が署名されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に xenc:EncryptedKey と ds:Signature として格納されます。xenc:EncryptedKey には、SOAP メッセージ本体の暗号化に使用した鍵が格納されます。この鍵は受信者の公開鍵で暗号化されます。

表 10–1 メッセージ保護ポリシーと WS-Security SOAP メッセージセキュリティー処理との対応づけ (続き)

メッセージ保護ポリシー	結果として実行される WS-Security SOAP メッセージ保護処理
auth-source="content" auth-recipient="after-content"	SOAP メッセージ本体のコンテンツが、署名されたあと暗号化され、その結果得られた xend:EncryptedData で置換されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に xenc:EncryptedKey と ds:Signature が格納されます。xenc:EncryptedKey には、SOAP メッセージ本体の暗号化に使用した鍵が格納されます。この鍵は受信者の公開鍵で暗号化されます。
auth-recipient="before-content" または auth-recipient="after-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に xenc:EncryptedKey が格納されます。xenc:EncryptedKey には、SOAP メッセージ本体の暗号化に使用した鍵が格納されます。この鍵は受信者の公開鍵で暗号化されます。
ポリシーを何も指定しない。	モジュールはセキュリティー処理を一切行いません。

▼ その他のセキュリティー機能を設定する

Application Server は、SOAP 処理レイヤー内に統合化されたメッセージセキュリティープロバイダを使用して、メッセージセキュリティーを実装します。メッセージセキュリティープロバイダは、Application Server のその他のセキュリティー機能に依存します。

1. バージョン 1.5.0 より前のバージョンの Java SDK を使用し、暗号化技術を使用する場合は、JCE プロバイダを設定します。

   JCE プロバイダの設定については、220 ページの「JCE プロバイダを設定する」を参照してください。

2. ユーザー名トークンを使用する場合は、必要に応じてユーザーデータベースを設定します。ユーザー名およびパスワードトークンを使用する場合は、適切なレルムを設定し、このレルムに適切なユーザーデータベースを設定する必要があります。

   ユーザーデータベースの設定については、173 ページの「レルムを編集する」を参照してください。

3. 必要に応じて証明書と非公開鍵を管理します。

   証明書と非公開鍵の管理については、201 ページの「証明書ファイルについて」を参照してください。

次の手順 Application Server の機能の設定が完了し、メッセージセキュリティープロバイダがそれらの機能を使用できるようになると、Application Server とともにインストールされたプロバイダを有効にできます。その手順については、222 ページの「メッセージセキュリティーのプロバイダを有効にする」を参照してください。

▼ JCE プロバイダを設定する

J2SE 1.4.x に付属している JCE (Java Cryptography Extension) プロバイダは、RSA 暗号化をサポートしていません。通常、WS-Security で定義されている XML 暗号化は RSA 暗号化に基づいているため、WS-Security を使って SOAP メッセージを暗号化するには、RSA 暗号化をサポートする JCE プロバイダをダウンロードおよびインストールする必要があります。

注 – RSA は RSA Data Security, Inc. が開発した公開鍵暗号化技術です。この略語は、この技術の開発者である Rivest、Shamir、および Adelman を表しています。

Java SDK バージョン 1.5 で Application Server を実行している場合は、JCE プロバイダは正しく設定されています。Java SDK バージョン 1.4.x で Application Server を実行している場合は、次のようにJCE プロバイダを JDK 環境の一部として静的に追加できます。

1. **JCE プロバイダの JAR (Java ARchive) ファイルをダウンロードし、インストールします。**

   次の URL で、RSA 暗号化をサポートする JCE プロバイダのリストが提供されています。`http://java.sun.com/products/jce/jce14_providers.html`。

2. **JCE プロバイダの JAR ファイルを** *java-home*`/jre/lib/ext/` **にコピーします。**

3. **Application Server を停止します。**

   Application Server を停止せずにこの手順の最後で再起動した場合、JCE プロバイダは Application Server に認識されません。

4. **任意のテキストエディタで** *java-home*`/jre/lib/security/java.security` **プロパティーファイルを編集します。このファイルに、前述の手順でダウンロードした JCE プロバイダを追加します。**

   `java.security` ファイルに、このプロバイダを追加する詳細手順が含まれています。基本的には、類似したプロパティーを持つ場所に次の形式の行を追加する必要があります。

   `security.provider.`*n*`=`*provider-class-name*

   この例では、*n* は、Application Server がセキュリティープロバイダを評価する際に使用する優先順位を示します。ここで追加した JCE プロバイダには、*n* を 2 に設定します。

   たとえば、Legion of the Bouncy Castle JCE プロバイダをダウンロードした場合は、次のような行を追加します。

   ```
   security.provider.2=org.bouncycastle.jce.provider.
      BouncyCastleProvider
   ```

   Sun セキュリティープロバイダが、値 1 の最高の優先順位に設定されていることを確認してください。

   ```
   security.provider.1=sun.security.provider.Sun
   ```

各レベルにセキュリティープロバイダがただ 1 つだけ設定されるように、ほかのセキュリティープロバイダのレベルを下位に調整します。

次に示す例は、必要な JCE プロバイダを提供し、既存のプロバイダを正しい位置に保持する `java.security` ファイルのサンプルです。

```
security.provider.1=sun.security.provider.Sun
security.provider.2=org.bouncycastle.jce.provider.
   BouncyCastleProvider
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.rsajca.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=sun.security.jgss.SunProvider
```

5. ファイルを保存して、閉じます。

6. **Application Server** を再起動します。

メッセージセキュリティーに関する管理コンソールタスク

メッセージセキュリティーを使用できるように Application Server を設定する手順のほとんどは、管理コンソールまたは `asadmin` コマンド行ツールを使用するか、あるいはシステムファイルを手動で編集することで実現できます。一般に、システムファイルの編集はお勧めできません。なぜなら、Application Server が適切に動作しなくなるような変更を間違って施してしまう可能性があるからです。したがって、できるだけ、管理コンソールによる Application Server の設定手順を最初に示し、その後に `asadmin` ツールコマンドによる手順を示しています。システムファイルを手動で編集する手順は、管理コンソールと `asadmin` に同等の方法が存在しない場合にだけ示しています。

メッセージレイヤーセキュリティーのサポートは、プラグイン可能な認証モジュールの形式で Application Server とそのクライアントコンテナに統合されています。Application Server では、メッセージレイヤーセキュリティーはデフォルトで無効になっています。次の各節では、メッセージセキュリティー設定とプロバイダを有効化、作成、編集、および削除する方法について、詳しく説明します。

- 222 ページの「メッセージセキュリティーのプロバイダを有効にする」
- 224 ページの「メッセージセキュリティープロバイダを設定する」
- 226 ページの「メッセージセキュリティープロバイダの作成」
- 228 ページの「メッセージセキュリティー設定を削除する」
- 229 ページの「メッセージセキュリティープロバイダを削除する」
- 230 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」

ほとんどの場合、上記の管理操作を実行したあとで Application Server を再起動する必要があります。特に、操作実行時に Application Server 上にすでに配備されていたアプリケーションに管理上の変更を適用したい場合に Application Server の再起動が必要となります。

▼ メッセージセキュリティーのプロバイダを有効にする

Application Server 上に配備された Web サービスエンドポイントのメッセージセキュリティーを有効にするには、サーバー側でデフォルトで使用されるプロバイダを指定する必要があります。メッセージセキュリティーのデフォルトプロバイダを有効にする場合、Application Server 上に配備された Web サービスクライアントが使用するプロバイダも有効にする必要があります。クライアントが使用するプロバイダを有効にする方法については、230 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」を参照してください。

配備済みエンドポイントからの Web サービス呼び出しに対するメッセージセキュリティーを有効にするには、デフォルトクライアントプロバイダを指定する必要があります。Application Server のデフォルトクライアントプロバイダを有効にした場合、Application Server 内に配備されたエンドポイントから呼び出されるすべてのサービスが、メッセージレイヤーセキュリティー用に正しく設定されていることを確認する必要があります。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「セキュリティー」ノードを展開します。
4. 「メッセージセキュリティー」ノードを展開します。
5. 「**SOAP**」ノードを選択します。
6. 「メッセージセキュリティー」タブを選択します。

7 「メッセージセキュリティー設定の編集」ページで、特定のプロバイダがバインドされていないすべてのアプリケーションに対して、サーバー側で使用されるプロバイダとクライアント側で使用されるプロバイダを指定します。

それには、次の各オプションプロパティーを変更します。

- デフォルトプロバイダ – 特定のサーバープロバイダがバインドされていない任意のアプリケーションに対して呼び出されるサーバープロバイダの ID。

  デフォルトでは、Application Server に対して、プロバイダの設定は何も選択されていません。サーバー側のプロバイダを特定するには、`ServerProvider` を選択します。null オプションの選択は、サーバー側ではメッセージセキュリティープロバイダがデフォルトで何も呼び出されないことを意味します。

  通常、このフィールドでは `ServerProvider` を選択します。

- デフォルトクライアントプロバイダ – 特定のクライアントプロバイダがバインドされていない任意のアプリケーションに対して呼び出されるクライアントプロバイダの ID。

  デフォルトでは、Application Server に対して、プロバイダの設定は何も選択されていません。クライアント側のプロバイダを特定するには、`ClientProvider` を選択します。null オプションの選択は、クライアント側ではメッセージセキュリティープロバイダがデフォルトで何も呼び出されないことを意味します。

  通常、このフィールドでは null を選択します。`ClientProvider` は、Application Server 上に配備された Web サービスエンドポイントからの Web サービス呼び出しに適用するデフォルトプロバイダとメッセージ保護ポリシーを有効にする場合に選択します。

8 「保存」をクリックします。

9 特定のクライアントプロバイダまたはサーバープロバイダを有効にしたあとで、その有効になったプロバイダのメッセージ保護ポリシーを変更する場合には、**224** ページの「メッセージセキュリティープロバイダを設定する」を参照し、この手順で有効にしたメッセージセキュリティープロバイダを変更する方法を確認してください。

参考 同機能を持つ asadmin コマンド

- デフォルトサーバープロバイダを指定するには、次のコマンドを実行します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
default_provider=ServerProvider
```

- デフォルトクライアントプロバイダを指定するには、次のコマンドを実行します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
default_client_provider=ClientProvider
```

▼ メッセージセキュリティープロバイダを設定する

プロバイダの再設定は通常、そのメッセージ保護ポリシーを変更するために行われますが、プロバイダのタイプ、実装クラス、およびプロバイダ固有の設定プロパティーも変更可能です。

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「セキュリティー」ノードを展開します。

4 「メッセージセキュリティー」ノードを展開します。

5 「**SOAP**」ノードを選択します。

6 「プロバイダ」タブを選択します。

7 編集するメッセージセキュリティープロバイダを選択します。

Application Server には `ClientProvider` と `ServerProvider` が付属しています。

8 「プロバイダ設定の編集」の「プロバイダ設定」セクションで、次のプロパティーが変更に使用できます。

- プロバイダタイプ – `client`、`server`、または `client-server` を選択して、プロバイダがクライアント認証プロバイダ、サーバー認証プロバイダ、あるいはこの両方 (クライアント - サーバープロバイダ) のいずれかとして使用されるようにします。
- クラス名 - プロバイダの Java 実装クラスを入力します。クライアント認証プロバイダは、`com.sun.enterprise.security.jauth.ClientAuthModule` インタフェースを実装する必要があります。サーバー側プロバイダは、`com.sun.enterprise.security.jauth.ServerAuthModule` インタフェースを実装する必要があります。プロバイダはどちらのインタフェースも実装可能ですが、プロバイダタイプに対応したインタフェースを実装している必要があります。

9 「プロバイダ設定の作成」ページの「要求ポリシー」セクションで、必要に応じて、次のオプションの値を入力します。

これらのプロパティーはオプションですが、指定されない場合、認証はメッセージの要求にまったく適用されません。

「要求ポリシー」は、認証プロバイダが実行する要求処理に関連付けられた認証ポリシー要件を定義します。ポリシーはメッセージ送信者の順序で送信されます。この順序では、コンテンツのあとで暗号化するという要件は、メッセージ受信者がメッセージの復号化を行なってから署名の検証を想定することを意味しています。

- 認証元– `sender`、`content`、または null (空白オプション) を選択して、メッセージレイヤー送信者認証 (ユーザー名、パスワードなど)、コンテンツ認証 (デジタル署名など) の要件を定義するか、あるいはメッセージの要求に認証を適用しないようにします。null が指定される場合、要求のソース認証は必須ではありません。
- 認証受信者– `beforeContent` または `afterContent` を選択し、XML 暗号化などを使用して、送信者に対する要求メッセージの受信者のメッセージレイヤー認証要件を定義します。値が指定されない場合、デフォルトでは `afterContent` に指定されます。

SOAP メッセージセキュリティープロバイダがメッセージ保護ポリシーに従った決定として実行されるアクションについては、218 ページの「要求および応答ポリシー設定のアクション」を参照してください。

10 「プロバイダ設定の作成」ページの「応答ポリシー」セクションで、必要に応じて、次のオプションの値を入力します。

これらのプロパティーはオプションですが、指定されない場合、認証はメッセージの応答にまったく適用されません。

「応答ポリシー」は、認証プロバイダが実行する応答処理に関連付けられた認証ポリシー要件を定義します。ポリシーはメッセージ送信者の順序で送信されます。この順序では、コンテンツのあとで暗号化するという要件は、メッセージ受信者がメッセージの復号化を行なってから署名の検証を想定することを意味しています。

- 認証元 – `sender`、`content`、または null (空白オプション) を選択して、メッセージの応答に適用されるように、メッセージレイヤー送信者認証 (ユーザー名、パスワードなど) またはコンテンツ認証 (デジタル署名など) の要件を定義します。null が指定される場合、応答のソース認証は必須ではありません。
- 認証受信者 – `beforeContent` または `afterContent` を選択し、XML 暗号化などを使用して、送信者に対する応答メッセージの受信者のメッセージレイヤー認証要件を定義します。値が指定されない場合、デフォルトでは `afterContent` に指定されます。

SOAP メッセージセキュリティープロバイダがメッセージ保護ポリシーに従った決定として実行されるアクションについては、218 ページの「要求および応答ポリシー設定のアクション」を参照してください。

11 「プロパティーを追加」ボタンをクリックして、プロパティーをさらに追加します。

Application Server に同梱されるプロバイダは、次に表示するプロパティーをサポートします。ほかのプロバイダが使用されている場合、プロパティーおよび有効な値の詳細については、該当するドキュメントを参照してください。

- `server.config` – サーバー設定情報を収める XML ファイルのディレクトリおよびファイル名。たとえば、*domain-dir*`/config/wss-server-config.xml`。

12 「保存」をクリックします。

参考 同機能を持つ asadmin コマンド

応答ポリシーを設定するには、次のコマンドの request を response に置き換えます。

- 要求ポリシーをクライアントに追加して、認証元を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ClientProvider.request-policy.auth_source=
sender | content
```

- 要求ポリシーをサーバーに追加して、認証元を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ServerProvider.request-policy.auth_source=
sender | content
```

- 要求ポリシーをクライアントに追加して、認証受信者を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ClientProvider.request-policy.auth_recipient=
before-content | after-content
```

- 要求ポリシーをサーバーに追加して、認証受信者を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ServerProvider.request-policy.auth_recipient=
before-content | after-content
```

▼ メッセージセキュリティープロバイダの作成

既存のプロバイダを設定する場合は、224 ページの「メッセージセキュリティープロバイダを設定する」の手順に従ってください。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、default-config ノードを選択します。

3 「セキュリティー」ノードを展開します。

4 「メッセージセキュリティー」ノードを展開します。

5 「**SOAP**」ノードを選択します。

6 「プロバイダ」タブを選択します。

7 「プロバイダ設定」ページで、「新規」をクリックします。

8 「プロバイダ設定の作成」ページの「プロバイダ設定」セクションで、次を入力します。

- デフォルトプロバイダ – このフィールドの横にあるボックスをオンにして、新しいメッセージセキュリティープロバイダを、特定のプロバイダがバインドされていない任意のアプリケーションに対して呼び出されるプロバイダとして指定します。このプロバイダが、デフォルトクライアントプロバイダ、デフォルトサーバープロバイダ、その両方のいずれになるかは、「プロバイダタイプ」の値によって決まります。
- プロバイダタイプ – `client`、`server`、または `client-server` を選択して、プロバイダがクライアント認証プロバイダ、サーバー認証プロバイダ、あるいはこの両方 (クライアント - サーバープロバイダ) のいずれかとして使用されるようにします。
- プロバイダ **ID** - このプロバイダ設定の ID を入力します。この名前は「現在のプロバイダ設定」リストに表示されます。
- クラス名 - プロバイダの Java 実装クラスを入力します。クライアント認証プロバイダは、`com.sun.enterprise.security.jauth.ClientAuthModule` インタフェースを実装する必要があります。サーバー側プロバイダは、`com.sun.enterprise.security.jauth.ServerAuthModule` インタフェースを実装する必要があります。プロバイダはどちらのインタフェースも実装可能ですが、プロバイダタイプに対応したインタフェースを実装している必要があります。

9 「プロバイダ設定の作成」ページの「要求ポリシー」セクションで、必要に応じて、次のオプションの値を入力します。

これらのプロパティーはオプションですが、指定されない場合、認証はメッセージの要求にまったく適用されません。

- 認証元– `sender`、`content`、または null (空白オプション) を選択して、メッセージレイヤー送信者認証 (ユーザー名、パスワードなど)、コンテンツ認証 (デジタル署名など) の要件を定義するか、あるいはメッセージの要求に認証を適用しないようにします。null が指定される場合、要求のソース認証は必須ではありません。
- 認証受信者– `beforeContent` または `afterContent` を選択し、XML 暗号化などを使用して、送信者に対する要求メッセージの受信者のメッセージレイヤー認証要件を定義します。値が指定されない場合、デフォルトでは `afterContent` に指定されます。

SOAP メッセージセキュリティープロバイダがメッセージ保護ポリシーに従った決定として実行されるアクションについては、218 ページの「要求および応答ポリシー設定のアクション」を参照してください。

10 「プロバイダ設定の作成」ページの「応答ポリシー」セクションで、必要に応じて、次のオプションの値を入力します。

これらのプロパティーはオプションですが、指定されない場合、認証はメッセージの応答にまったく適用されません。

- 認証元 – `sender`、`content`、または null (空白オプション) を選択して、メッセージの応答に適用されるように、メッセージレイヤー送信者認証 (ユーザー名、パスワードなど) またはコンテンツ認証 (デジタル署名など) の要件を定義します。null が指定される場合、応答のソース認証は必須ではありません。
- 認証受信者 – `beforeContent` または `afterContent` を選択し、XML 暗号化などを使用して、送信者に対する応答メッセージの受信者のメッセージレイヤー認証要件を定義します。値が指定されない場合、デフォルトでは `afterContent` に指定されます。

SOAP メッセージセキュリティープロバイダがメッセージ保護ポリシーに従った決定として実行されるアクションについては、218 ページの「要求および応答ポリシー設定のアクション」を参照してください。

11 「プロパティーを追加」ボタンをクリックして、プロパティーをさらに追加します。

Application Server に同梱されるプロバイダは、次に表示するプロパティーをサポートします。ほかのプロバイダが使用されている場合、プロパティーおよび有効な値の詳細については、該当するドキュメントを参照してください。

- `server.config` – サーバー設定情報を収める XML ファイルのディレクトリおよびファイル名。たとえば、*domain-dir*`/config/wss-server-config.xml`。

12 「了解」をクリックしてこの設定を保存するか、「取消し」をクリックして保存しないで終了します。

参考 同機能を持つ asadmin コマンド

`create-message-security-provider`

▼ メッセージセキュリティー設定を削除する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「セキュリティー」ノードを展開します。

4 「メッセージセキュリティー」ノードを選択します。

5 削除する「メッセージセキュリティー設定」の左側のチェックボックスをクリックします。

6 「削除」をクリックします。

▼ メッセージセキュリティープロバイダを削除する

1 管理コンソールのツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「セキュリティー」ノードを展開します。

4 「メッセージセキュリティー」ノードを展開します。

5 「**SOAP**」ノードを選択します。

6 「プロバイダ」ページを選択します。

7 削除する「プロバイダ設定」の左側のチェックボックスをクリックします。

8 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-message-security-provider`

▼ アプリケーションクライアントのメッセージセキュリティーを有効にする

クライアントプロバイダのメッセージ保護ポリシーは、通信相手となるサーバー側プロバイダのメッセージ保護ポリシーと等しくなるように設定する必要があります。Application Server のインストール時に設定された (しかしまだ有効化されていない) プロバイダでは、すでにそうなっています。

クライアントアプリケーションのメッセージセキュリティーを有効にするには、アプリケーションクライアントコンテナの Application Server 固有の設定を変更します。

1. クライアントコンテナ記述子に依存するすべてのクライアントアプリケーションを停止します。

2. テキストエディタで、*domain-dir*`/config/sun-acc.xml` に格納されている **Sun** アプリケーションクライアントコンテナ記述子を開きます。

3. このファイルに `default-client-provider` 要素を追加して、アプリケーションクライアントのデフォルトクライアントプロバイダを有効にします。

   その他のコードは、クライアントアプリケーションのメッセージセキュリティーを有効にするコードを配置すべき場所を示すためにあります。実際のインストールでは、その他のコードが若干異なっている可能性があります。それらを変更しないでください。

```
<client-container>
  <target-server name="your-host" address="your-host"
      port="your-port"/>
  <log-service file="" level="WARNING"/>
  <message-security-config auth-layer="SOAP"
      default-client-provider="ClientProvider">
    <provider-config
        class-name="com.sun.enterprise.security.jauth.ClientAuthModule"
        provider-id="ClientProvider" provider-type="client">
      <request-policy auth-source="sender"/>
      <response-policy/>
        <property name="security.config"
          value="C:/Sun/AppServer/lib/appclient/wss-client-config.xml"/>
    </provider-config>
  </message-security-config>
</client-container>
```

また、クライアントコンテナ内に設定されたメッセージセキュリティープロバイダは、非公開鍵と信頼できる証明書にアクセスする必要もあります。それには、アプリケーションクライアントの起動スクリプト内で、次のシステムプロパティーに適切な値を指定します。

```
-Djavax.net.ssl.keyStore

-Djavax.net.ssl.trustStore
```

アプリケーションクライアント設定の要求および応答ポリシーの設定

「要求および応答ポリシー」は、認証プロバイダが実行する要求および応答処理に関連付けられた認証ポリシー要件を定義します。ポリシーはメッセージ送信者の順序で送信されます。この順序では、コンテンツのあとで暗号化するという要件は、メッセージ受信者がメッセージの復号化を行なってから署名の検証を想定することを意味しています。

メッセージセキュリティーを実現するには、サーバーとクライアントの両方で要求ポリシーと応答ポリシーが有効化されている必要があります。クライアントおよびサーバーのポリシーを設定する場合は、クライアントポリシーがアプリケーションレベルのメッセージのバインドで要求および応答保護のサーバーポリシーと一致する必要があります。

アプリケーションクライアント設定の要求ポリシーを設定するには、230 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」の説明に従って、アプリケーションクライアントコンテナの Application Server 固有の設定を変更します。アプリケーションクライアント設定ファイル内で `request-policy` 要素と `response-policy` 要素を次のように追加することで、要求ポリシーを設定します。

その他のコードは参照用に用意されています。実際のインストールでは、その他のコードが若干異なっている可能性があります。それらを変更しないでください。

```
<client-container>
  <target-server name="your-host" address="your-host"
      port="your-port"/>
  <log-service file="" level="WARNING"/>
  <message-security-config auth-layer="SOAP"
      default-client-provider="ClientProvider">
    <provider-config
        class-name="com.sun.enterprise.security.jauth.ClientAuthModule"
        provider-id="ClientProvider" provider-type="client">
      <request-policy auth-source="sender | content"
        auth-recipient="after-content | before-content"/>
      <response-policy auth-source="sender | content"
        auth-recipient="after-content | before-content"/>
```

```
        <property name="security.config"
            value="install-dir/lib/appclient/wss-client-config.xml"/>
    </provider-config>
  </message-security-config>
</client-container>
```

auth-source の有効な値には、sender と content があります。auth-recipient の有効な値には、before-content と after-content があります。これらの値をさまざまに組み合わせた結果を記述した表については、218 ページの「要求および応答ポリシー設定のアクション」を参照してください。

要求または応答ポリシーを指定しない場合は、この要素を空白のままにします。次に例を示します。

```
<response-policy/>
```

詳細情報

- Java 2 Standard Edition のセキュリティーの説明については、http://java.sun.com/j2se/1.4.2/docs/guide/security/index.html を参照してください。
- 『J2EE 1.4 Tutorial』の「Security」の章については、http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html を参照してください。
- 『管理ガイド』の第 9 章。
- 『Developer's Guide』の「Securing Applications」の章。
- 『Oasis Web Services Security: SOAP Message Security (WS-Security)』仕様については、http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf を参照してください。
- 『OASIS Web Services Security Username Token Profile 1.0』については、http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf を参照してください。
- 『OASIS Web Services Security X.509 Certificate Token Profile 1.0』については、http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf を参照してください。
- XML-Signature Syntax and Processingドキュメントについては、http://www.w3.org/TR/xmldsig-core/ を参照してください。
- XML Encryption Syntax and Processingドキュメントについては、http://www.w3.org/TR/xmlenc-core/ を参照してください。

◆ ◆ ◆ 第 11 章

11 トランザクション

トランザクションを使用すると、1 つ以上のステップがそれ以上分割不可能な作業単位にまとめられるため、データの完全性と整合性が保証されます。この章には次の節が含まれています。

トランザクションについて

トランザクションとは

トランザクションは、すべて正常に完了することが必要なアプリケーションで、周到に用意された一連のアクションです。正常に完了しない場合、各アクションで行われたすべての変更が取り消されます。たとえば、当座預金から普通預金に資金を移動するのは次の手順を実行するトランザクションになります。

1. 当座預金口座にその移動をカバーするだけの金額があるかどうかを確認します。
2. 当座預金に十分なお金が入っている場合は、当座預金の金額を借り方に記帳します。
3. その金額を普通預金口座の貸し方に記帳します。
4. その移動を当座預金口座ログに記録します。
5. その移動を普通預金口座ログに記録します。

これらの手順のいずれが失敗すると、先行する手順によって行われた変更がすべて取り消されます。当座預金口座と普通預金口座はこのトランザクションが始まる前と同じ状態になる必要があります。このイベントは「ロールバック」と呼ばれます。すべての手順が正常に完了すると、トランザクションは「コミット」状態になります。 トランザクションはコミットかロールバックのどちらかで終了します。

J2EE テクノロジのトランザクション

J2EE テクノロジのトランザクション処理には、次の 5 つの関係要素が含まれます。

- トランザクションマネージャー
- Application Server
- リソースマネージャー (複数可)
- リソースアダプタ (複数可)
- ユーザーアプリケーション

これらの各エンティティーは、次に説明する API や機能を実装することにより、信頼性のあるトランザクション処理を実現しています。

- トランザクションマネージャーは、トランザクション境界、トランザクションリソース管理、同期化、およびトランザクションコンテキスト伝達のサポートに必要なサービスと管理機能を提供します。
- Application Server は、トランザクション状態管理を含むアプリケーションランタイム環境のサポートに必要なインフラストラクチャーを提供します。
- リソースマネージャーは、リソースアダプタを介して、リソースへのアプリケーションアクセスを提供します。リソースマネージャーは、特定のトランザクションリソースインタフェースを実装することで分散トランザクションに参加します。このインタフェースは、トランザクションマネージャーがトランザクションの関連付け、トランザクションの完了、および回復作業を伝達する際に使用されます。このようなリソースマネージャーの例としては、リレーショナルデータベースサーバーがあります。
- リソースアダプタはシステムレベルのソフトウェアライブラリで、リソースマネージャーへ接続するためにアプリケーションサーバーまたはクライアントが使用します。通常、リソースアダプタはリソースマネージャーに固有です。リソースアダプタはライブラリとして使用可能で、クライアントのアドレス空間内で使用されます。そのようなリソースアダプタの一例として、JDBC ドライバが挙げられます。
- アプリケーションサーバー環境で動作するように開発されたトランザクションユーザーアプリケーションは、JNDI を使用してトランザクションデータソースおよびトランザクションマネージャー (オプション) を検索します。アプリケーションは、エンタープライズ Bean 用の宣言的なトランザクション属性設定や、プログラムによる明示的なトランザクション境界を使用することがあります。

トランザクションに関する管理コンソールタスク

Application Server は、管理コンソールの設定に基づいてトランザクションを処理します。

トランザクションの設定

この節では、トランザクションの設定方法について説明します。

- 235 ページの「Application Server のトランザクションからの回復方法を設定する」

- 236 ページの「トランザクションのタイムアウト値を設定する」
- 236 ページの「トランザクションログの場所を設定する」
- 237 ページの「キーポイント間隔を設定する」

トランザクションに関する追加情報については、次の各節を参照してください。

- 233 ページの「トランザクションとは」
- 234 ページの「J2EE テクノロジのトランザクション」

▼ Application Server のトランザクションからの回復方法を設定する

トランザクションは、サーバークラッシュまたはリソースマネージャークラッシュのいずれかにより未完了になる可能性があります。これらの未完了トランザクションを完了させ、障害を回復させる必要があります。Application Server は、これらの障害を回復し、サーバーの起動時にそのトランザクションを完了するように設計されています。

リカバリを行っている間にリソースにアクセスできなくなった場合は、トランザクションを回復しようとしてサーバーの再起動が遅れた可能性があります。

トランザクションが複数のサーバーにわたっている場合は、トランザクションを開始したサーバーがトランザクションの結果を取得しようとしてほかのサーバーに問い合わせる場合があります。ほかのサーバーにアクセスできない場合、そのトランザクションは「特殊な結果判別」フィールドを使用してその結果を判別します。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「トランザクションサービス」ノードを選択します。
4. 未完了なトランザクションのリカバリを有効にするには、「再起動時」フィールドで「回復」にチェックマークを付けます。
5. 「再試行タイムアウト」フィールドに、**Application Server** がアクセスできないサーバーに対して接続を試みる時間を秒単位で設定します。デフォルト値は **10** 分 **(600** 秒**)** です。

6 「特殊な結果判別」フィールドに、トランザクションでアクセスできないサーバーのポリシーを設定します。

このフィールドを「コミット」に設定する適切な理由がないかぎり、「特殊な結果判別」を「ロールバック」のままにしておきます。未確定なトランザクションのコミットは、アプリケーションのデータの整合性を損なう可能性があります。

7 「保存」をクリックします。

8 **Application Server** を再起動します。

▼ トランザクションのタイムアウト値を設定する

デフォルトでは、サーバーはトランザクションをタイムアウトしないようになっています。つまり、サーバーはトランザクションの完了を待機し続けます。トランザクションのタイムアウト値を設定して、トランザクションが設定された時間内に完了しない場合、Application Server はトランザクションをロールバックします。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3 「トランザクションサービス」ノードを選択します。

4 「トランザクションタイムアウト」フィールドに、トランザクションがタイムアウトする秒数を入力します。

トランザクションタイムアウトのデフォルト値は 0 秒です。これにより、トランザクションのタイムアウトは無効になります。

5 「保存」をクリックします。

6 **Application Server** を再起動します。

▼ トランザクションログの場所を設定する

トランザクションログは、関連リソースのデータの整合性を維持して障害を回復するために、各トランザクションについての情報を記録します。トランザクションログは、「トランザクションログの位置」フィールドで指定したディレクトリの `tx` サブディレクトリに保存されます。これらのログは人間が読み取れるものではありません。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- すべてのインスタンスのデフォルト値を設定するには、default-config ノードを選択します。

3 「トランザクションサービス」ノードを選択します。

4 「トランザクションログの位置」フィールドに、トランザクションログの位置を入力します。

tx サブディレクトリが作成され、トランザクションログがそのディレクトリの下に保存されます。

デフォルト値は ${com.sun.aas.instanceRoot}/logs です。${com.sun.aas.instanceRoot} 変数はインスタンスの名前であり、Application Server インスタンスの起動時に設定されます。${com.sun.aas.instanceRoot} の値を表示するには、「実際の値」をクリックします。

5 「保存」をクリックします。

6 **Application Server** を再起動します。

▼ キーポイント間隔を設定する

キーポイント処理によって、トランザクションログファイルが圧縮されます。キーポイント間隔とは、ログに対して実行されるキーポイント処理の間のトランザクション数のことです。キーポイント処理によって、トランザクションログファイルのサイズを小さくすることができます。キーポイント間隔を大きくすると (例: 2048)、トランザクションログファイルが大きくなりますが、キーポイント処理が少なくなるのでパフォーマンスが向上する可能性があります。キーポイント間隔を小さくすると (例: 256)、ログファイルのサイズが小さくなりますが、キーポイント処理が多くなるので、パフォーマンスがわずかに低下します。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。

- すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

3. 「トランザクションサービス」ノードを選択します。
4. 「キーポイント間隔」フィールドに、キーポイント処理間のトランザクション数を入力します。
   デフォルト値は 2048 です。
5. 「保存」をクリックします。
6. **Application Server** を再起動します。

◆ ◆ ◆ 第 12 章

HTTP サービスの設定

この章では、Application Server の HTTP サービスコンポーネントの仮想サーバーと HTTP リスナーを設定する方法について説明します。

HTTP サービスについて

HTTP サービスとは

HTTP サービスは、Web アプリケーションの配備機能を提供する Application Server のコンポーネントで、配備された Web アプリケーションに HTTP クライアントがアクセスできるようにします。65 ページの「Web アプリケーションを配備する」を参照してください。これらの機能は、仮想サーバーと HTTP リスナーという 2 種類の関連オブジェクトによって提供されます。

仮想サーバー

仮想サーバーは、複数のインターネットドメイン名を同一の物理サーバーでホスティングするためのオブジェクトで、仮想ホストとも呼ばれます。同一物理サーバーにホスティングされるすべての仮想サーバーは、その物理サーバーの IP (Internet Protocol) アドレスを共有します。仮想サーバーは、サーバーのドメイン名 (`www.aaa.com` など) と、Application Server が稼動するサーバーを関連付けます。

注 – インターネットドメインと Application Server の管理ドメインを混同しないでください。

たとえば、ある物理サーバーで次のドメインをホスティングすると仮定します。

```
www.aaa.com
www.bbb.com
www.ccc.com
```

また、www.aaa.com、www.bbb.com、www.ccc.com には、それぞれに関連付けられた Web モジュール web1、web2、web3 があるものとします。

つまり、その物理サーバーでは、次のすべての URL が処理されます。

```
http://www.aaa.com:8080/web1
http://www.bbb.com:8080/web2
http://www.ccc.com:8080/web3
```

最初の URL は仮想ホスト www.aaa.com、2 番目の URL は仮想ホスト www.bbb.com、3 番目の URL は仮想ホスト www.ccc.com にそれぞれマッピングされます。

一方、www.bbb.com には web3 が登録されていないため、次の URL は 404 リターンコードのエラーとなります。

```
http://www.bbb.com:8080/web3
```

このマッピングが機能するには、www.aaa.com、www.bbb.com、www.ccc.com のすべてを物理サーバーの IP アドレスとして解決する必要があります。これをネットワークの DNS サーバーに登録しなければなりません。さらに、UNIX システムで、/etc/nsswitch.conf ファイルの hosts の設定に files が含まれる場合は、これらのドメインを /etc/hosts ファイルに追加します。

Application Server を起動すると、次の 2 つの仮想サーバーが自動的に起動されます。

- ユーザー定義のすべての Web モジュールをホスティングする仮想サーバー server
- すべての管理関連 Web モジュール (具体的には管理コンソール) をホスティングする仮想サーバー __asadmin。このサーバーの使用は制限されています。つまり、ユーザーがこの仮想サーバーに Web モジュールを配備することはできません。

本稼動環境以外での Web サービスの開発、テスト、配備で必要となる仮想サーバーは、通常、server だけです。ただし本稼動環境では、同一物理サーバー上でユーザーと顧客のそれぞれが専用の Web サーバーを持つように見せる機能をホスティングするため、通常は追加の仮想サーバーも使用されます。

HTTP リスナー

各仮想サーバーは、1 つまたは複数の HTTP リスナーを通じてサーバーとクライアントの間の接続を提供します。各 HTTP リスナーは、IP アドレス、ポート番号、サーバー名、およびデフォルトの仮想サーバーを持つ待機ソケットです。

HTTP リスナーは、ポート番号と IP アドレスの一意の組み合わせを持つ必要があります。たとえば、IP アドレス 0.0.0.0 を指定すると、HTTP リスナーは設定されたすべての IP アドレスをマシンの特定のポートで待機できます。また、各リスナーに一意の IP アドレスを指定した上で、同一ポートを使用することもできます。

HTTP リスナーは IP アドレスとポート番号の組み合わせであるため、IP アドレスが同じでポート番号が異なる HTTP リスナーや (例: 1.1.1.1:8081 および 1.1.1.1:8082)、IP アドレスが異なっていてポート番号が同じ HTTP リスナー (例: 1.1.1.1:8081 および 1.2.3.4:8081。ただし、マシンがこれら両方のアドレスに応答するように設定されている場合) を複数使用することができます。

ただし、HTTP リスナーに単一のポート上ですべての IP アドレスを待機する 0.0.0.0 を使用する場合は、この同じポート上に、特定の IP アドレスを待機する HTTP リスナーを作成できません。たとえば、HTTP リスナーが 0.0.0.0:8080 (ポート 8080 のすべての IP アドレス) を使用する場合、別の HTTP リスナーが 1.2.3.4:8080 を使用することはできません。

通常、Application Server が稼動するシステムでアクセスできる IP アドレスは 1 つだけであるため、HTTP リスナーは、ポートが異なる 0.0.0.0 IP アドレスを通常使用し、役割ごとに異なるポート番号を使用します。システムが複数の IP アドレスにアクセスできる場合は、各アドレスを異なる役割に使用できます。

デフォルトでは、Application Server を起動すると、次の HTTP リスナーが準備されます。

- 仮想サーバー `server` に関連付けられた 2 つの HTTP リスナー `http-listener-1` および `http-listener-2`。`http-listener-1` ではセキュリティーが無効になり、`http-listener-2` ではセキュリティーが有効になります。
- 仮想サーバー `__asadmin` に関連付けられた HTTP リスナー `admin-listener`。このリスナーでは、セキュリティーが有効になります。

これらのリスナーはすべて、Application Server のインストールの間に HTTP サーバーポート番号として指定された IP アドレス 0.0.0.0 とポート番号を使用します。Application Server がポート番号のデフォルト値をそのまま使用した場合、`http-listener-1` はポート 8080、`http-listener-2` はポート 8181、`admin-listener` はポート 4849 を使用します。

各 HTTP リスナーはデフォルトの仮想サーバーを持ちます。デフォルトの仮想サーバーは、HTTP リスナーがその HTTP リスナーに関連付けられたどの仮想サーバーともホストコンポーネントが一致しないすべての要求 URL をルーティングする宛先のサーバーです。仮想サーバーと HTTP リスナーの関連付けは、仮想サーバーの `http-listeners` 属性に HTTP リスナーを指定することで行われます。

さらに、HTTP リスナー内のアクセプタスレッドの数を指定します。アクセプタスレッドは、接続を待機するスレッドです。アクセプタスレッドによって受け付けられ、接続キューと呼ばれるキューに入れられた接続は、ワークスレッドによって取り出されま

す。新しい要求が着信したときにいつでも対応できるように、常に十分な数のアクセプタスレッドを設定しておきますが、システムに負荷がかかり過ぎない数に抑える必要もあります。接続キューには、アクセプタスレッドによって受け付けられた新しい接続と、キープアライブ接続管理サブシステムによって管理される持続接続の両方が格納されます。

一連の要求処理スレッドが、接続キューから受信 HTTP 要求を取り出し、それらの要求を処理します。これらのスレッドは、HTTP ヘッダーを解析し、適切な仮想サーバーを選択し、要求処理エンジンを実行して要求を処理します。処理すべき要求がなくなったあと、その接続が `HTTP/1.1` を使用するか `Connection: keep-alive` ヘッダーを送信することで持続可能になっていた場合、要求処理スレッドは、その接続がアイドル状態にあると判断し、その接続をキープアライブ接続管理サブシステムに渡します。

キープアライブサブシステムは、そうしたアイドル状態の接続を定期的にポーリングし、活動中の接続が見つかるとそれらを接続キュー内に格納し、さらに処理できるようにします。要求処理スレッドは、そのキューから再び接続を取り出し、その要求を処理します。キープアライブサブシステムはマルチスレッド化されています。なぜなら、このサブシステムによって管理される接続の数が数万個に及ぶ可能性があるからです。効率的なポーリングテクニックに基づいて多数の接続がより少数の接続を含むサブセットへと分割され、どの接続で要求の準備が整ったか、あるいはどの接続のアイドル時間が閉じてもよいほど十分長い時間になったか、つまり、最大許容キープアライブタイムアウトを超えたかが判断されます。

HTTP リスナーのサーバー名は、サーバーがクライアントに送信する URL にリダイレクトの一部として表示されるホスト名です。この属性は、サーバーが自動的に生成する URL には影響しますが、サーバーに格納されているディレクトリやファイルの URL には影響しません。サーバーがエイリアスを使っている場合、通常は、この名前はエイリアス名です。クライアントが `Host:` ヘッダーを送信する場合、HTTP リスナーのサーバー名の代わりにホスト名がリダイレクトに指定されます。

リダイレクトポートを指定して、元の要求に指定されているポート番号とは異なるポート番号を使用します。リダイレクトは、次のいずれかの状況で行われます。

- リソースが別の位置に移動され、クライアントのアクセス対象のリソースが指定の URL に存在しない場合、サーバーは 404 を返す代わりに指定の応答コードを返し、応答のロケーションヘッダーに新しい位置を含めることで、クライアントを新しい位置にリダイレクトします。
- 通常の HTTP ポートで SSL などによって保護されているリソースにクライアントがアクセスを試みる場合、サーバーは要求を SSL 有効ポートにリダイレクトします。この場合、サーバーは、元のセキュリティー保護されていないポートを SSL 有効ポートに置き換えた新しい URL がロケーション応答ヘッダーに指定された応答を返します。クライアントは、この新しい URL に接続します。

また、HTTP リスナーのセキュリティーを有効にするかどうか、あるいは、どのセキュリティーの種類を使用するか (例: SSL プロトコルや暗号化方式の種類) も指定します。

Application Server に配備された Web アプリケーションにアクセスするには、Web アプリケーション用に指定したコンテキストルートとともに、`http://localhost:8080/` (また

は、セキュリティー保護されたアプリケーションでは https://localhost:8181/) という URL を使用します。管理コンソールにアクセスするには、URL https://localhost:4849/ またはそのデフォルトコンテキストルートである https://localhost:4849/asadmin/ を使用します。

仮想サーバーは既存の HTTP リスナーを指定する必要があり、ほかの仮想サーバーによってすでに使用されている HTTP リスナーを指定できないことから、新しい仮想サーバーを作成するときは、事前に少なくとも 1 つの HTTP リスナーを作成します。

HTTP サービスに関する管理コンソールタスク

▼ HTTP サービスを設定する

1. ツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
   - default-config のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、default-config ノードを選択します。
3. 「**HTTP** サービス」ノードを選択します。
4. 「**HTTP** サービス」ページでは、サービスのすべての **HTTP** リスナーに適用されるプロパティーを設定できます。
   次の表には、これらのプロパティーが一覧表示されています。

プロパティー名	説明	デフォルト値
traceEnabled	true に設定すると、TRACE 処理が有効になります。このプロパティーを false に設定すると、Application Server がクロスサイトスクリプティング攻撃の影響を受けにくくなります。	false

プロパティー名	説明	デフォルト値
monitoringCacheEnabled	true に設定すると、Application Server は、統計クエリーへの回答用に HTTP サービスの統計のローカル値をキャッシュします。この値を使用すると、パフォーマンスが向上します。 false に設定すると、Application Server は、統計値ごとに HTTP サービスに対するクエリーを発行します。	true
monitoringCacheRefreshInMillis	監視キャッシュの更新間隔をミリ秒単位で指定します。	5000
sslCacheEntries	キャッシュ可能な SSL セッションの数を指定します。上限はありません。	10000
sslSessionTimeout	SSL2 セッションがタイムアウトするまでの秒数を指定します。	100
ssl3SessionTimeout	SSL3 セッションがタイムアウトするまでの秒数を指定します。	86400
sslClientAuthDataLimit	クライアント証明書ハンドシェークフェーズでバッファリングされるアプリケーションデータの最大サイズをバイト単位で指定します。	1048576
sslClientAuthTimeout	クライアント証明書ハンドシェークフェーズがタイムアウトするまでの秒数を指定します。	60
keepAliveQueryMeanTime	希望するキープアライブ応答時間をミリ秒単位で指定します。	100
keepAliveQueryMaxSleepTime	キープアライブ接続に新しい要求の有無をポーリングしたあとのスリープ時間の上限をミリ秒単位で指定します。	100
stackSize	ネイティブスレッドの最大スタックサイズを指定します。	OS/マシンに依存
statsProfilingEnabled	false に設定すると、HTTP サービスによる監視統計の記録が無効になり、パフォーマンスが向上します。このプロパティーを false に設定すると、HTTP サービスの監視を有効にしても実際には有効にならなくなります。	true

プロパティー名	説明	デフォルト値
chunkedRequestBufferSize	要求データのチャンク解除に対するデフォルトバッファーサイズをバイト単位で指定します。	8192
chunkedRequestTimeoutSeconds	要求データのチャンク解除に対するデフォルトタイムアウトを秒単位で指定します。	60
dnsCacheEnabled	true に設定すると、DNS キャッシュに関する統計をユーザーが監視できるようになります。このプロパティーが有効になるのは、「HTTP プロトコル」タブの「DNS 検索」ボックスが選択されている場合だけです。それ以外の場合、このプロパティーの設定は無視されます。	false

5 「アクセスログ」タブをクリックして、アクセスログのローテーションを設定します。
その他のタブをクリックして、要求処理、キープアライブサブシステム、接続プール、HTTP プロトコル、および HTTP ファイルキャッシュを設定します。

6 「保存」をクリックします。

▼ HTTP サービスのアクセスログを設定する

このページを使用して、仮想サーバーのアクセスログのローテーションを有効にし、設定します。これらのログは、*domain-dir*/logs/access ディレクトリにあり、次のように命名されます。*virtual-server-name* _access_log%YYYY;%MM;%DD-%hh;h%mm;m%ss;s

「デフォルト」をクリックして、デフォルト値を読み込みます。

1 「ファイルローテーション」ボックスにチェックマークを付けて、ファイルローテーションを有効にします。
デフォルトで、ファイルローテーションは有効です。

2 「ローテーションポリシー」ドロップダウンリストから、ポリシーを選択します。
使用可能なポリシーは time だけです。

3 「ローテーション間隔」フィールドに、数値を入力してアクセスログのローテーション間の分数を指定します。
このフィールドは、「ローテーションポリシー」が time の場合にのみ有効です。デフォルト値は 1440 分です。

4 「ローテーションサフィックス」フィールドに、文字列値を入力して、ローテーション後にログファイル名に追加されるサフィックスを指定します。
デフォルトは、`%YYYY;%MM;%DD;-%hh;h%mm;m%ss;s` です。

5 「形式」フィールドに、文字列値を入力してアクセスログの形式を指定します。
次の表に示されている形式を使用してください。デフォルトの形式は、`%client.name% %auth-user-name% %datetime% %request% %status% %response.length%` です。

データ	トークン
クライアントホスト名	`%client.name%`
クライアント DNS	`%client.dns%`
システム日付	`%datetime%`
全 HTTP 要求行	`%request%`
状態	`%status%`
応答コンテンツ長	`%response.length%`
リファラーヘッダー	`%header.referer%`
ユーザーエージェント	`%header.user-agent%`
HTTP メソッド	`%http-method%`
HTTP URI	`%http-uri%`
HTTP クエリー文字列	`%query-str%`
HTTP プロトコルバージョン	`%http-version%`
アクセプトヘッダー	`%header.accept%`
日付ヘッダー	`%header.date%`
If-Modified-Since ヘッダー	`%header.if-mod-since%`
承認ヘッダー	`%header.auth%`
RFC 2616 で定義された任意の有効な HTTP ヘッダー値 (any も有効なヘッダー値。ここでは変数として指定されている)	`%header.`*any*`%`
承認ユーザーの名前	`%auth-user-name%`
Cookie の値	`%cookie.value%`
仮想サーバーの ID	`%vs.id%`

6 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」を選択してデフォルト設定に戻します。

▼ HTTP サービスの要求処理スレッドを設定する

1 「デフォルトを読込み」をクリックして、デフォルト値を読み込みます。

2 「スレッド数」フィールドに、要求処理スレッドの最大数を示す数値を入力します。
デフォルトは 128 です。

3 「初期スレッド数」フィールドに、サーバー起動時に利用可能な要求処理スレッドの数を入力します。
デフォルトは 48 です。

4 「スレッドの増分」フィールドに、要求数が初期スレッド数を超えた場合に追加される要求処理スレッドの数を入力します。
デフォルトは 10 です。

5 「要求タイムアウト」フィールドに、要求がタイムアウトするまでの秒数を入力します。
デフォルトは 30 秒です。

6 「バッファー長」フィールドに、要求処理スレッドが要求データの読み取り時に使用するバッファーのサイズをバイト単位で入力します。
デフォルトは 4096 バイトです。

7 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」を選択してデフォルト設定に戻します。

▼ HTTP サービスのキープアライブサブシステムを設定する

1 「デフォルトを読込み」をクリックして、デフォルト値を読み込みます。

2 「スレッド数」フィールドに、使用するキープアライブスレッドの数を入力します。
デフォルトは 1 です。

3 「最大接続数」フィールドに、保持される持続接続の最大数を入力します。
デフォルトは 256 です。

4 「タイムアウト」フィールドに、キープアライブ接続を開いたままにしておく最大秒数を入力します。
デフォルトは 30 秒です。

5 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」を選択してデフォルト設定に戻します。

▼ HTTP サービスの接続プールを設定する

1 「デフォルトを読込み」をクリックして、デフォルト値を読み込みます。

2 「最大保留カウント」フィールドに、**HTTP** リスナーに対して許可する保留接続の最大数を入力します。
デフォルトは 4096 です。

3 「キューサイズ」フィールドに、接続キューの最大サイズをバイト単位で入力します。
この値は、サーバーが保持できる未処理接続の最大数も指定します。デフォルトは 4096 です。

4 「受信バッファーサイズ」フィールドに、**HTTP** リスナーの受信バッファーのサイズを入力します。
デフォルトは 4096 です。

5 「送信バッファーサイズ」フィールドに、**HTTP** リスナーの送信バッファーのサイズを入力します。
デフォルトは 8192 です。

6 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」を選択してデフォルト設定に戻します。

▼ HTTP サービスの HTTP プロトコルを設定する

1 「デフォルトを読込み」をクリックして、デフォルト値を読み込みます。

2 「バージョン」フィールドに、使用する **HTTP** プロトコルのバージョン (`HTTP/1.0`、`HTTP/1.1` のいずれか) を入力します。
デフォルトは `HTTP/1.1` です。

3 「**DNS** 検索」ボックスを選択してクライアントに対する **DNS** エントリの検索を有効にします。
デフォルトは false です。

4 「**SSL**」ボックスのチェックマークを外してサーバーのセキュリティーをグローバルに無効にします。

この値を true のままにしておけば、セキュリティーが有効になっているリスナーで SSL を使用できます。デフォルトは true です。

5 「強制応答タイプ」フィールドに、拡張子に一致する利用可能な **MIME** マッピングが見つからない場合に使用する応答タイプを入力します。

デフォルト値は、`text/html; charset=iso-8859-1` です。

6 「デフォルト応答タイプ」フィールドに、デフォルト応答タイプを入力します。

デフォルト値は、`text/html; charset=iso-8859-1` です。値は、コンテンツタイプ、エンコーディング、言語、および文字セットから構成される、セミコロンで区切られた文字列になります。

7 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」を選択してデフォルト設定に戻します。

▼ HTTP サービスの HTTP ファイルキャッシュを設定する

ファイルキャッシュには静的コンテンツが格納されるため、サーバーはそうしたコンテンツに対する要求をすばやく処理できます。

1 「デフォルトを読込み」をクリックして、デフォルト値を読み込みます。

2 「グローバル」ボックスにチェックマークを付けてファイルキャッシュを有効にします。

デフォルトは true です。

3 「ファイル転送」ボックスにチェックマークを付けることで、**Windows** 上で `TransmitFileSystem` メソッドを有効にします。

デフォルトは false です。

4 「最大有効期間」フィールドに、有効なキャッシュエントリの最大有効期間を秒単位で入力します。

デフォルトは 30 秒です。

5 「最大ファイル数」フィールドに、ファイルキャッシュ内のファイルの最大数を入力します。

デフォルトは 1024 です。

6 「ハッシュ初期サイズ」フィールドに、ハッシュバケットの初期数を入力します。
デフォルトは 0 です。

7 「ファイルサイズ上限 (中)」フィールドに、メモリーがマップされたファイルとしてキャッシュできるファイルの最大サイズをバイト単位で入力します。
デフォルトは 537,600 バイトです。

8 「ファイルサイズ (中)」フィールドに、メモリーにマップされたファイルとしてキャッシュされるすべてのファイルの合計サイズをバイト単位で入力します。
デフォルトは 10,485,760 バイトです。

9 「ファイルサイズ上限 (小)」フィールドに、メモリーに読み込めるファイルの最大サイズをバイト単位で入力します。
デフォルトは 2048 バイトです。

10 「ファイルサイズ (小)」フィールドに、メモリーに読み込まれるすべてのファイルの合計サイズをバイト単位で入力します。
デフォルトは 1,048,576 バイトです。

11 「ファイルキャッシュを有効化」ドロップダウンリストから ON または OFF を選択することで、ファイルサイズがファイルサイズ上限 (中) より小さい場合にファイルコンテンツのキャッシュを有効または無効にします。
デフォルトは ON です。

12 「保存」をクリックして変更を保存するか、または「デフォルトを読込み」を選択してデフォルト設定に戻します。

仮想サーバーに関する管理コンソールタスク

▼ 仮想サーバーを作成する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。

- `default-config` のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、`default-config` ノードを選択します。

3 「**HTTP** サービス」ノードを展開します。

4 「仮想サーバー」ノードを選択します。

5 「仮想サーバー」ページで「新規」をクリックします。
「仮想サーバーを作成」ページが表示されます。

6 「**ID**」フィールドで、仮想サーバーの一意の名前を入力します。
この値は、仮想サーバーの内部的な識別に使用されます。これが HTTP クライアント側に表示されることはありません。HTTP クライアント側に表示するホスト名は、「ホスト」フィールドに指定する必要があります。

7 「ホスト」フィールドに、サーバーが稼動するマシンのホスト名 **(1** つまたは複数**)** を入力します。
ネットワークの DNS サーバー (UNIX システムでは `/etc/hosts` ファイル) に登録されている、実際のホスト名または仮想ホスト名を使用します。

8 「状態」セクションで、「オン」、「オフ」、または「無効」を選択します。
デフォルトは「オン」です。

9 「**HTTP** リスナー」フィールドは空のまま残します。
このフィールドは、HTTP リスナーを作成し、このサーバーと関連付けたときに自動的に設定されます。

このフィールドを使用するときは、既存の HTTP リスナーを指定する必要があります。ただし、別の仮想サーバーで使用されているリスナーを指定しないでください。指定した場合、サーバーログにエラーが表示されます。HTTP リスナーは、作成時に既存の仮想サーバーと関連付ける必要があるため、既存のすべてのリスナーはすでに別の仮想サーバーによって使用されています。

10 「デフォルト **Web** モジュール」ドロップダウンリストから、仮想サーバーに配備されているその他の **Web** モジュールにマッピングできないすべての要求に対応する配備済み **Web** モジュールを選択します **(**配備されている場合**)**。
デフォルトの Web モジュールが指定されない場合は、コンテキストルートが空の Web モジュールが使用されます。コンテキストルートが空の Web モジュールが存在しない場合は、システムのデフォルトの Web モジュールが作成され、使用されます。

11 「ログファイル」フィールドに、この仮想サーバーからのログメッセージが記録されるファイルのパス名を入力します。
デフォルトのサーバーログ *domain-dir*`/logs/server.log` にログメッセージを送信する場合は、このフィールドを空のまま残します。

12 「追加プロパティー」で、仮想サーバーのプロパティーを追加する場合は「プロパティーを追加」をクリックします。

プロパティーを追加するかどうかに関係なく、新しいサーバーはデフォルトプロパティー `docroot` および `accesslog` を持ち、それぞれにデフォルトの値が設定されます。

次の表には、使用可能な仮想サーバーのプロパティーが一覧表示されています。

プロパティー名	説明
`docroot`	サーバーのルートドキュメントディレクトリへの絶対パス。 デフォルトは *domain-dir*`/docroot`。
`accesslog`	サーバーのアクセスログへの絶対パス。 デフォルトは *domain-dir*`/logs/access`。
`sso-enabled`	false の場合、この仮想サーバーに対するシングルサインオンは無効となり、ユーザーは仮想サーバー上のアプリケーションごとに個別に認証を行う必要があります。 Application Server 上のアプリケーション間でのシングルサインオンは、サーブレットと JSP ページによってサポートされます。この機能により、複数のアプリケーションが同一のサインオン情報を共有できるため、ユーザーはアプリケーションごとにサインオンする必要がなくなります。 デフォルトは true。
`sso-max-inactive-seconds`	クライアントが活動を停止後、何秒後にユーザーのシングルサインオンの記録をパージ可能にするかを指定します。シングルサインオンは同一仮想サーバー上の複数のアプリケーションに適用されるので、これらのアプリケーションのいずれかにアクセスすることでシングルサインオンの記録は有効なまま確保されます。 デフォルトは 300 秒 (5 分)。値を大きくするとユーザーのシングルサインオンの持続時間は長くなりますが、サーバー上のメモリー消費量も増加します。
`sso-reap-interval-seconds`	有効期限が切れたシングルサインオンの記録のパージを行う間隔を秒単位で指定します。 デフォルトは 60。

プロパティー名	説明

allowLinking	true の場合、この仮想サーバー上に配備されているすべての Web アプリケーションに対し、シンボリックリンクのリソースが提供されます。個々の Web アプリケーションでこの設定をオーバーライドできます。それには、sun-web.xml ファイル内で sun-web-app のプロパティー allowLinking を次のように指定します。
	`<sun-web-app>`
	`  <property name="allowLinking"`
	`  value="{true|false}"/>`
	`</sun-web-app>`
	デフォルトは true。

13 「了解」をクリックして、仮想サーバーを保存します。

参考 同機能を持つ asadmin コマンド

`create-virtual-server`

▼ 仮想サーバーを編集する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- default-config のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、default-config ノードを選択します。

3 「**HTTP** サービス」ノードを展開します。

4 「仮想サーバー」ノードを選択します。

5 編集する仮想サーバーを選択します。

6 「仮想サーバーを編集」ページで、次のタスクを実行できます。

- 「ホスト」フィールドのホスト名の変更。
- 「状態」設定の値の変更。

- **HTTP** リスナーの追加または削除。
- 「デフォルト **Web** モジュール」の選択の変更。
- 「ログファイル」の値の変更。
- プロパティーの追加、削除、または変更。

7 「保存」をクリックして変更を保存します。

▼ 仮想サーバーを削除する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、`default-config` ノードを選択します。

3 「**HTTP** サービス」ノードを展開します。

4 「仮想サーバー」ノードを選択します。

5 「仮想サーバー」ページで、削除する仮想サーバーの名前の隣のチェックボックスにチェックマークを付けます。

6 「削除」をクリックします。

仮想サーバー `__asadmin` を削除することもできますが、お勧めできません。削除する場合は、必要時に設定を復元できるように Application Server の `domain.xml` ファイルに含まれる `virtual-server` 要素を事前に安全な場所にコピーしておいてください。

参考 同機能を持つ asadmin コマンド

`delete-virtual-server`

HTTP リスナーに関する管理コンソールタスク

▼ HTTP リスナーを作成する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、`default-config` ノードを選択します。

3 「**HTTP** サービス」ノードを展開します。

4 「**HTTP** リスナー」ノードを選択します。

5 「**HTTP** リスナー」ページで「新規」をクリックします。
「HTTP リスナーを作成」ページが表示されます。

6 「名前」フィールドに、リスナーの名前を入力します。

7 サーバーの再起動時にリスナーを有効にしないようにするには、「リスナー」フィールドの「有効」ボックスのチェックマークを外します。

8 単一ポート番号を使用して、サーバーのすべての **IP** アドレスを待機するようにリスナーを設定するときは、「ネットワークアドレス」に「**0.0.0.0**」を入力します。それ以外の場合は、サーバーの有効な **IP** アドレスを入力します。

9 「リスナーポート」フィールドに、ネットワークアドレス **0.0.0.0** を使用する場合は一意のポート番号を指定し、別の **IP** アドレスを使用する場合は任意のポート番号を指定します。

10 「デフォルト仮想サーバー」ドロップダウンリストから仮想サーバーを選択します。

11 「サーバー名」フィールドに、サーバーがクライアントに送信する **URL** で使用されるホスト名を入力します。サーバーがエイリアスを使っている場合、この名前はエイリアス名です。

サーバーがエイリアスを使用していない場合は、このフィールドを空のまま残します。

12 「詳細」セクションでは、次のいずれかの操作を行います。

- 要求を別のポートにリダイレクトするには、「リダイレクトポート」フィールドに値を入力します。Application Server は、次の 2 つの条件が存在する場合、要求を自動的にリダイレクトします。
  - このリスナーが非 SSL 要求をサポートしている。
  - 着信した要求に適用されるセキュリティー制約によって SSL 伝送が必要である。

  デフォルトでは、Application Server は元の要求に指定されているポート番号を使用します。
- アクセプタスレッドの数を変更します。
- 「Powered By」チェックボックスからチェックマークを外し、サーブレットが生成する HTTP 応答ヘッダー内に `X-Powered-By: Servlet/2.4` ヘッダーが含まれないようにします。

  Java Servlet 2.4 仕様では、サーブレットが生成する応答にコンテナがこのヘッダーを追加できるように定義されています。同様に、JavaServer Pages™ (JSP) 2.0 仕様では、JSP テクノロジを使用する応答にオプションとして `X-Powered-By: JSP/2.0` ヘッダーを追加するように定義されています。Web アプリケーションでは、`X-Powered-By: JSP/2.0` ヘッダーを含めることがデフォルトで有効になっています。これらのヘッダーの目的は、Web サイトの管理者がサーブレットと JSP テクノロジの使用に関する統計データを収集することです。

  JSP ページの `X-Powered-By` ヘッダーの有効化と無効化に関する詳細ついては、『*Application Server* 開発者ガイド』の「配備記述子ファイル」という章を参照してください。このマニュアルへのリンクについては、58 ページの「詳細情報」を参照してください。

  本稼動環境では、`X-Powered-By` ヘッダーの生成を省略し、使用されている技術が知られないようにすることができます。

13 セキュリティー保護されていないリスナーを作成するには、「了解」をクリックします。

14 このページの「**SSL**」セクションで、リスナーが **SSL**、**TLS**、あるいはこの両方のセキュリティーを使用するように設定できます。

セキュリティー保護されたリスナーを設定するには、次の手順を実行します。

15 「セキュリティー」フィールドの「有効」ボックスにチェックマークを付けます。

16 サーバーへの認証をこのリスナーを使っている個々のクライアントに任せる場合は、「クライアント認証」フィールドの「有効」ボックスにチェックマークを付けます。

17 「証明書のニックネーム」フィールドに、既存サーバーの鍵ペアと証明書の名前を入力します。詳細については、「セキュリティー」の章を参照してください。

18 **SSL3/TLS** セクションでは次の手順を実行します。

a. リスナーで有効にするセキュリティープロトコルにチェックマークを付けます。**SSL3** と **TLS** のどちらかに、あるいはこの両方にチェックマークを付けます。

b. プロトコルが使用する暗号化方式にチェックマークを付けます。すべての暗号化方式を有効にするには、「サポートされるすべての暗号化方式群」にチェックマークを付けます。個別の暗号化方式を有効にすることもできます。

19 これにより、デフォルト仮想サーバーとして指定した仮想サーバーの「**HTTP** リスナー」フィールドにこのリスナーが表示されるようになります。

参考 同機能を持つ asadmin コマンド

`create-http-listener` および `create-ssl`

▼ HTTP リスナーを編集する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、`default-config` ノードを選択します。

3 「**HTTP** サービス」ノードを展開します。

4 「**HTTP** リスナー」ノードを選択します。

5 編集する **HTTP** リスナーを選択します。

6 「**HTTP** リスナーを編集」ページで、任意の設定を変更してください。

7 「保存」をクリックして変更を保存します。

▼ HTTP リスナーを削除する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルト設定を設定するには、`default-config` ノードを選択します。

3 「**HTTP** サービス」ノードを展開します。

4 「**HTTP** リスナー」ノードを選択します。

5 「**HTTP** リスナー」ページで、削除する **HTTP** リスナーの隣のチェックボックスにチェックマークを付けます。

6 「削除」をクリックします。

HTTP リスナー `http-listener-1`、`http-listener-2`、`admin-listener` を削除することもできますが、お勧めできません。削除する場合は、必要時に設定を復元できるように Application Server の `domain.xml` ファイルに含まれる `http-listener` 要素を事前に安全な場所にコピーしておいてください。

参考 同機能を持つ asadmin コマンド

`delete-http-listener`

◆ ◆ ◆ 第 13 章

ORB (Object Request Broker) の設定

この章では、ORB (Object Request Broker) と IIOP リスナーの設定方法について説明します。ここには次の節があります。

ORB (Object Request Broker) について

CORBA

Application Server は、相互運用性を確実にする一連の標準的なプロトコルおよび形式をサポートします。これらのプロトコルの中には、CORBA で定義されているものがあります。

CORBA (Common Object Request Broker Architecture) モデルのベースになっているのは、明確に定義されたインタフェースを介して分散型のオブジェクトやサーバーにサービスを要求するクライアントです。こうしたクライアントは、リモートメソッド要求の形式でオブジェクトに対して要求を発行します。リモートメソッド要求では、実行する必要のある操作に関する情報が伝送されます。この情報には、サービスプロバイダのオブジェクト名 (オブジェクト参照) と、存在する場合は、起動メソッドのパラメータが含まれます。CORBA は、オブジェクトの登録、オブジェクトの配置、オブジェクトのアクティブ化、要求の多重分離、エラー処理、整列化、操作のディスパッチをはじめとするネットワークプログラミングのタスクを自動的に処理します。

ORB とは

ORB (Object Request Broker) は、CORBA の中枢となるコンポーネントです。ORB は、オブジェクトの特定と検索、接続管理、およびデータと要求の配信に必要なインフラストラクチャーを提供します。

個々の CORBA オブジェクトが相互に対話することはありません。その代わりに、リモートスタブを介して、ローカルマシンで実行されている ORB に要求を送ります。次に、ローカルの ORB が、IIOP (Internet Inter-Orb Protocol) を使ってほかのマシン上の ORB へ要求を転送します。リモート ORB は、適切なオブジェクトを検出し、要求を処理して、結果を返します。

アプリケーションやオブジェクトでは、RMI-IIOP により、IIOP を RMI (Remote Method Invocation) として使用することが可能になっています。エンタープライズ Bean (EJB モジュール) のリモートクライアントは、RMI-IIOP を介して Application Server と通信します。

IIOP リスナー

IIOP リスナーは、Enterprise JavaBeans のリモートクライアントおよびほかの CORBA ベースのクライアントから受信する接続を受け付ける待機ソケットです。Application Server では、複数の IIOP リスナーを設定できます。各リスナーに対して、ポート番号、ネットワークアドレス、およびオプションでセキュリティー属性を指定してください。詳細については、261 ページの「IIOP リスナーを作成する」を参照してください。

ORB に関する管理コンソールタスク

▼ ORB を設定する

1. ツリーコンポーネントで、「設定」ノードを展開します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。
3. 「**ORB**」ノードを選択します。

4 「スレッドプール ID」ドロップダウンリストから、**ORB** が使用するスレッドプールを選択します。

ORB はスレッドプールを使用し、Enterprise JavaBeans のリモートクライアントおよび RMI-IIOP を介して通信するほかのクライアントからの要求に応答します。詳細については、265 ページの「スレッドプールについて」および 266 ページの「スレッドプールを作成する」を参照してください。

5 「最大メッセージ分割サイズ」フィールドに、**IIOP** メッセージの最大フラグメントサイズを設定します。

このサイズより大きいメッセージは分割されます。

6 「総接続数」フィールドに、すべての **IIOP** リスナーの入力接続の最大数を設定します。

7 **IIOP** クライアント認証が必要な場合には、「必須」チェックボックスを選択します。

8 「保存」をクリックして変更を保存するか、または「デフォルトを読み込み」を選択してデフォルト値を読み込みます。

9 サーバーを再起動します。

IIOP リスナーに関する管理コンソールタスク

▼ IIOP リスナーを作成する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 「**ORB**」ノードを展開します。

4 **IIOP** リスナーを選択します。

5 「新規」をクリックします。

6 「名前」フィールドにリスナーを特定する名前を入力します。

7 「ネットワークアドレス」フィールドに、リスナーのネットワークアドレスを入力します。
ここにはIPアドレスあるいは、DNSで解決可能なホスト名を指定することができます。

8 「リスナーポート」フィールドに、リスナーが待機するポートを入力します。

9 「リスナー」フィールドの「有効」ボックスにチェックマークを付けて、リスナーを有効にします。

10 「追加プロパティー」セクションで、アプリケーションに必要なプロパティーの値を指定します。

11 リスナーを作成するには、次の手順に従います。

- セキュアでないリスナーを作成するには、「了解」をクリックします。
- 安全なリスナーを設定するには、次の手順を実行します。

  a. 「セキュリティー」フィールドの「有効」ボックスにチェックマークを付けます。

  b. サーバーへの認証をこのリスナーを使っている個々のクライアントに任せる場合は、「クライアント認証」フィールドの「有効」ボックスにチェックマークを付けます。

  c. 「証明書のニックネーム」フィールドに、既存サーバーの鍵ペアと証明書の名前を入力します。

  d. **SSL3/TLS** セクションでは次の手順を実行します。

     i. リスナーで有効にするセキュリティープロトコルにチェックマークを付けます。**SSL3** および **TLS** のいずれか、または両方のプロトコルにチェックマークを付けます。

     ii. プロトコルが使用する暗号化方式にチェックマークを付けます。
     すべての暗号化方式を有効にするには、「サポートされるすべての暗号化方式群」にチェックマークを付けます。個別の暗号化方式を有効にすることもできます。

e. 「了解」をクリックします。
これで、「IIOP リスナー」ページの「現在のリスナー」テーブルに、リスナーが一覧表示されます。

参考 同機能を持つ asadmin コマンド

`create-iiop-listener` および `create-ssl`

▼ IIOP リスナーを編集する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 「**ORB**」ノードを展開します。

4 「**IIOP** リスナー」ノードを選択します。

5 「現在のリスナー」テーブルで、変更するリスナーを選択します。

6 リスナーの設定を変更します。
変更可能なフィールドについては、261 ページの「IIOP リスナーを作成する」を参照してください。

7 リスナーのポート番号を変更した場合は、サーバーを再起動してください。

▼ IIOP リスナーを削除する

1 ツリーコンポーネントで、「設定」ノードを展開します。

2 設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。

- `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3. 「**ORB**」ノードを展開します。
4. 「**IIOP** リスナー」ノードを選択します。
5. 「現在のリスナー」テーブルで、削除するリスナーにチェックマークを付けます。
6. 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-iiop-listener`

◆ ◆ ◆ 第 14 章

スレッドプール

この章では、スレッドプールを作成、編集、および削除する方法について説明します。ここには次の節があります。

スレッドプールについて

この節では、Application Server におけるスレッドプールの動作方法について説明します。

Java 仮想マシン (JVM) は、1 回の実行で多数のスレッドをサポートできます。パフォーマンスの向上に役立つように、Application Server は 1 つまたは複数のスレッドプールを維持します。特定のスレッドプールを、コネクタモジュールと ORB に割り当てることができます。

1 つのスレッドプールで、複数のコネクタモジュールおよびエンタープライズ Beans を処理できます。要求スレッドは、アプリケーションコンポーネントへのユーザーの要求を処理します。サーバーは要求を受け取ると、スレッドプールから使用可能なスレッドにその要求を割り当てます。スレッドはクライアントの要求を実行し、結果を返します。たとえば、現在ビジー状態のシステムリソースが必要な場合、スレッドはリソースが解放されるのを待ってから、リソースの使用を要求に許可します。

アプリケーションからの要求用に確保するスレッドの最小数と最大数を指定できます。スレッドプールはこれら 2 つの値の間で動的に調整されます。サーバーは、指定された最小スレッドプールサイズに従って、アプリケーション要求用に確保するスレッドを割り当てます。その数は、指定された最大スレッドプールサイズまで増加できます。

プロセスで使用可能なスレッドの数を増やすと、プロセスが同時に応答できるアプリケーション要求数が多くなります。

1 つのリソースアダプタやアプリケーションが Application Server のすべてのスレッドを占有している場合、Application Server のスレッドを複数のスレッドプールに分割することで、スレッド不足を防止してください。

スレッドプールに関する管理コンソールタスク

▼ スレッドプールを作成する

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「スレッドプール」ノードを選択します。
4. 「現在のプール」で、「新規」をクリックします。
5. 「スレッドプール **ID**」フィールドに、スレッドプールの名前を入力します。
6. 「最小プールサイズ」フィールドに、キューで要求を処理するスレッドプール内のスレッドの最小数を入力します。
   スレッドプールがインスタンス化されると、これらのスレッドが最前列に作成されます。
7. 「最大プールサイズ」フィールドに、キューで要求を処理するスレッドプール内のスレッドの最大数を入力します。
   これがそのスレッドプールに存在するスレッドの数の上限になります。
8. 「アイドルタイムアウト」フィールドに、プールからアイドルスレッドが削除されるアイドル時間の制限秒数を入力します。
9. 「作業キューの数」フィールドに、このスレッドプールが処理する作業キューの合計数を入力します。
10. 「了解」をクリックします。
11. **Application Server** を再起動します。

参考 同機能を持つ asadmin コマンド

`create-threadpool`

▼ スレッドプールを編集する

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「スレッドプール」ノードを選択します。
4. 「現在のプール」で変更するスレッドプールの名前を選択します。
5. 「最小プールサイズ」フィールドに、キューで要求を処理するスレッドプール内のスレッドの最小数を入力します。
   スレッドプールがインスタンス化されると、これらのスレッドが最前列に作成されます。
6. 「最大プールサイズ」フィールドに、キューで要求を処理するスレッドプール内のスレッドの最大数を入力します。
   これがそのスレッドプールに存在するスレッドの数の上限になります。
7. 「アイドルタイムアウト」フィールドに、プールからアイドルスレッドが削除されるアイドル時間の制限秒数を入力します。
8. 「作業キューの数」フィールドに、このスレッドプールが処理する作業キューの合計数を入力します。
9. 「保存」をクリックします。
10. **Application Server** を再起動します。

▼ スレッドプールを削除する

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「スレッドプール」ノードを選択します。
4. 「現在のプール」テーブルで、削除するスレッドプール名にチェックマークを付けます。
5. 「削除」をクリックします。
6. **Application Server** を再起動します。

参考 同機能を持つ asadmin コマンド

`delete-threadpool`

◆ ◆ ◆ 第 15 章

ロギングの設定

この章では、管理コンソールによるロギングの設定方法とサーバーログの表示方法について簡単に説明します。この章には次の節が含まれています。

ロギングについて

ログレコード

Application Server は、JSR 047 に指定されている「Java 2 platform Logging API」を使用します。Application Server のログメッセージはサーバーログ内に記録されます。このサーバーログの場所は通常、*domain-dir*`/logs/server.log` です。

domain-dir`/logs` ディレクトリには、サーバーログのほかに別の 2 種類のログも格納されます。`access` サブディレクトリには HTTP サービスアクセスログ、`tx` サブディレクトリにはトランザクションサービスログがあります。これらのログについては、245 ページの「HTTP サービスのアクセスログを設定する」および 234 ページの「トランザクションの設定」を参照してください。

Application Server のコンポーネントがログ出力を生成します。アプリケーションコンポーネントもログ出力を生成できます。

アプリケーションコンポーネントは、Apache Commons ロギングライブラリを使ってメッセージをロギングしてもかまいません。ただし、ログ設定を効率的に行いたい場合は、プラットフォーム標準の JSR 047 API を使用することをお勧めします。

ログレコードは次の統一形式に従います。

[#|*yyyy-mm-dd*T*hh*:*mm*:*ss*.*SSS-Z*|*Log Level*|*ProductName-Version*|*LoggerName*|*Key Value Pairs*|*Message*|#]

次に例を示します。

```
[#|2004-10-21T13:25:53.852-0400|INFO|sun-appserver-e8.1|javax.enterprise.
system.core|_ThreadID=13;|CORE5004: Resource Deployed:
[cr:jms/DurableConnectionFactory].|#]
```

この例について説明します。

- `[#` と `#]` はレコードの開始と終了をマーク付けします。
- 垂直バー (|) はレコードのフィールドを区切ります。
- `2004-10-21T13:25:53.852-0400` は日付と時刻を指定します。
- *Log Level* は `INFO` です。このレベルは次のいずれかの値を取ることができます。`SEVERE`、`WARNING`、`INFO`、`CONFIG`、`FINE`、`FINER`、および `FINEST`。
- *ProductName-Version* は `sun-appserver-ee8.1` です。
- *LoggerName* はログモジュールのソースを識別する階層ロガーの名前空間で、この場合は `javax.enterprise.system.core` です。
- *Key Value Pairs* はキー名と値で、通常は `_ThreadID=14;` のようなスレッド ID です。
- *Message* は、ログメッセージのテキストです。Application Server のすべての `SEVERE` メッセージと `WARNING` メッセージ、および多くの `INFO` メッセージは、モジュールコードと数値から構成されるメッセージ ID (この場合は `CORE5004`) で始まります。

このログレコード形式は、将来のリリースでは変更または拡張される可能性があります。

ロガー名前空間の階層

Application Server では、モジュールごとにロガーが用意されています。次の表では、管理コンソールの「ログレベル」ページに表示されるとおりに、アルファベット順でモジュールの名前と各ロガーの名前空間を示します (273 ページの「ログレベルを設定する」を参照)。表内の最後の 3 つのモジュールは、「ログレベル」ページには表示されません。

表 15–1 Application Server ロガー名前空間

モジュール名	名前空間
管理	`javax.enterprise.system.tools.admin`
クラスローダー	`javax.enterprise.system.core.classloading`
CMP	`javax.enterprise.system.container.cmp`
設定	`javax.enterprise.system.core.config`

表 15–1 Application Server ロガー名前空間　(続き)

モジュール名	名前空間
コネクタ	javax.enterprise.resource.resourceadapter
CORBA	javax.enterprise.resource.corba
配備	javax.enterprise.system.tools.deployment
EJB コンテナ	javax.enterprise.system.container.ejb
JavaMail	javax.enterprise.resource.javamail
JAXR	javax.enterprise.resource.webservices.registry
JAX-RPC	javax.enterprise.resource.webservices.rpc
JDO	javax.enterprise.resource.jdo
JMS	javax.enterprise.resource.jms
JTA	javax.enterprise.resource.jta
JTS	javax.enterprise.system.core.transaction
MDB コンテナ	javax.enterprise.system.container.ejb.mdb
ネーミング	javax.enterprise.system.core.naming
ノードエージェント (Enterprise Edition に限る)	javax.ee.enterprise.system.nodeagent
ルート	javax.enterprise
SAAJ	javax.enterprise.resource.webservices.saaj
セキュリティー	javax.enterprise.system.core.security
サーバー	javax.enterprise.system
同期 (Enterprise Edition に限る)	javax.ee.enterprise.system.tools.synchronization
ユーティリティー	javax.enterprise.system.util
ベリファイア	javax.enterprise.system.tools.verifier
Web コンテナ	javax.enterprise.system.container.web
コア	javax.enterprise.system.core
システム出力 (System.out.println)	javax.enterprise.system.stream.out
システムエラー (System.err.println)	javax.enterprise.system.stream.err

ロギングに関する管理コンソールタスク

▼ ログの一般設定を設定する

1 ツリーコンポーネントで、「ノードエージェント」ノードまたは「設定」ノードを開きます。

2 特定のノードエージェントを選択するか、設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
- `default-config` のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、`default-config` ノードを選択します。

3 ノードエージェントの場合は、「ロガーの設定」タブを選択します。設定の場合は、「ロガーの設定」ノードを選択します。

4 「ログ設定」ページで、次のフィールドを使用してログをカスタマイズします。

- ログファイル — サーバーログファイル用に別の名前や場所を指定するには、テキストフィールドに新しいパス名を入力します。デフォルトの場所は、*domain-dir*`/logs/server.log` です。
- アラーム — JMX フレームワークを通じて、`SEVERE` および `WARNING` メッセージをルーティングするには、「有効」チェックボックスにチェックマークを付けます。
- システムログへの書き込み — Solaris システムおよび Linux システムに限り、サーバーログのほかに `syslog` 機能にログ出力を送る場合は、「有効」チェックボックスにチェックマークを付けます。
- ログハンドラ — `server.log` あるいは `syslog` 以外の送信先にログを送る場合は、カスタムログハンドラを組み込むことができます。組み込まれるカスタムハンドラには、クラス `java.util.logging.Handler` (JSR 047 準拠の API) を拡張する必要があります。「ログハンドラ」フィールドに、ハンドラの絶対クラス名を入力します。Application Server のクラスパスにもそのハンドラクラスを置き、サーバーの起動時にハンドラがインストールされるようにします。カスタムハンドラからのログレコードは、269 ページの「ログレコード」に説明する形式になります。
- ログフィルタ — `server.log`、`syslog`、またはカスタムログハンドラで指定した送信先などに送信するログレコードをフィルタリングする場合は、カスタムログフィルタを組み込むことができます。組み込まれるカスタムフィルタは、インタフェース

java.util.logging.Filter を実装している必要があります。「ログフィルタ」フィールドに、フィルタの絶対クラス名を入力します。Application Server のクラスパスにもそのフィルタクラスを置き、サーバーの起動時にフィルタがインストールされるようにします。

- ファイルローテーションの制限 — サーバーログがバイト単位で指定されたサイズに達すると、新しい空のファイル server.log を作成し、古いファイルの名前を server.log_*date* に変更します。ここで、*date* は、ファイルのローテーションが発生した日付と時刻です。デフォルト値は 2M バイトです。上限の最小値は 500K バイトです。それより小さな値を指定すると、ファイルは 500K バイトに達した時点でローテーションされます。ログファイルのローテーションをオフにするには、この値を 0 に設定します。
- ファイルローテーションの時間制限 — 指定された分数に達すると、サーバーログはローテーションされます。デフォルト値は 0 です。これは「ファイルローテーションの制限」フィールドで指定されたサイズに達すると、ファイルがローテーションされることを意味しています。1 分以上の値を指定すると、時間制限がサイズ制限に優先されます。

5 「保存」をクリックして変更を保存します。「ログファイルを表示」をクリックして、サーバーログを表示します。

▼ ログレベルを設定する

1 ツリーコンポーネントで、「ノードエージェント」ノードまたは「設定」ノードを開きます。

2 特定のノードエージェントを選択するか、設定するインスタンスを選択します。

- 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス server の場合は、server-config ノードを選択します。
- default-config のコピーを利用する将来のインスタンスのためにデフォルトの設定値を設定するには、default-config ノードを選択します。

3 ノードエージェントの場合は、「ログレベル」タブを選択します。設定の場合は、「ロガーの設定」ノードを選択したあと、「ログレベル」タブを選択します。

4 「モジュールログレベル」ページで、対象モジュールまたはログレベルを変更するモジュールの横にあるドロップダウンリストから新しい値を選択します。

デフォルトのレベルは INFO で、それ以上のレベル (WARNING、SEVERE) のメッセージがログに表示されることを意味します。次の値のどれかを選択します (最高から最低まで表示)。

- SEVERE

- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST
- OFF

5 アプリケーションロガーのログレベルを設定するには、「追加プロパティー」セクションを使用します。

プロパティー名はロガー名前空間で、値は可能な 8 レベルのいずれか 1 つを指定します。たとえば、プロパティー名を samples.logging.simple.servlet に、その値を FINE に指定できます。

次のような CORBA モジュールのトランスポートサブモジュールなど、サブモジュールのログレベルを変更するときにもこの領域を使います。

javax.enterprise.resource.corba.*ORBId*.transport

6 「保存」をクリックして変更を保存するか、または「デフォルト」をクリックしてデフォルト値を復元します。

System.out.println への呼び出しは、ロガー名 javax.enterprise.system.stream.out を使って INFO レベルにログ記録されます。System.err.println への呼び出しは、ロガー名 javax.enterprise.system.stream.err を使って WARNING レベルにログ記録されます。これらのソースからのログをオフにするには、「追加プロパティー」セクションでロガー名の値を OFF に指定します。

ログレベル設定の変更はただちに有効になります。それらは domain.xml ファイルにも保存され、サーバーの再起動時に使用されます。

▼ サーバーログを表示する

1 ツリーコンポーネントで、ログを表示するサーバーインスタンスのノードを開きます。

2 「一般情報」ページで、「ログファイルの表示」をクリックします。

3 「検索条件」領域を使用して、ログビューアのカスタマイズおよびフィルタリングを行います。この基本フィールドは次のように使用します。

- インスタンス名 — ドロップダウンリストからインスタンス名を選択して、そのサーバーインスタンスのログを表示します。デフォルトは現在のサーバーインスタンスです。
- ログファイル — ドロップダウンリストからログファイル名を選択して、そのログのコンテンツを表示します。デフォルトは server.log です。

- タイムスタンプ — 最新のメッセージを表示するには、「最新」 (デフォルト) を選択します。特定期間内のメッセージだけを表示するには、「特定範囲」を選択して、表示される「開始」と「終了」フィールドに日付と時刻を入力します。時刻の値の構文は次の形式を必ず使用してください (*SSS* はミリ秒)。

  hh:*mm*:*ss*.*SSS*

  次に例を示します。

  ```
  17:10:00.000
  ```

  「開始」フィールドの値が「終了」フィールドの値よりも遅い場合は、エラーメッセージが表示されます。
- ログレベル — ログレベルでメッセージをフィルタリングするには、ドロップダウンリストからログレベルを選択します。デフォルトでは、サーバーログにある選択したログレベルおよびさらに重大なレベルのすべてのメッセージが表示に含まれます。選択したレベルのメッセージだけを表示するには、「より重度の高いメッセージをこれ以上含めない」というラベルの付いたチェックボックスを選択します。

  表示したいメッセージを確実にサーバーログに表示するには、最初に「ログレベル」ページで適切なログレベルを設定してください。273 ページの「ログレベルを設定する」を参照してください

  ログレベルに基づくログメッセージのフィルタリングを選択する場合、指定したフィルタリング基準と一致するメッセージだけが表示されます。しかし、このフィルタリングは、どのメッセージがサーバーログに記録されるかには影響しません。

  最新の 40 エントリが「ログ設定」と「ログレベル」ページで指定した設定で表示されます。

  最新のメッセージが最後に表示されるようにメッセージを並べ替えるには、タイムスタンプヘッダーのとなりの▲印をクリックします。

  形式設定済みのメッセージを表示するには、次の印が付いたリンクをクリックします。

  (詳細)

  「ログエントリの詳細」というウィンドウが現れて、形式設定済みメッセージを表示します。

  エントリのリストの末尾で、ボタンをクリックしてログファイルの古いエントリまたは新しいエントリを表示します。

  「検索条件」領域で「詳細検索」をクリックして、ログビューアの詳細設定を行います。「詳細オプション」フィールドは次のように使用します。
- ロガー — モジュール別にフィルタリングするには、ドロップダウンリストから 1 つ以上の名前空間を選択します。shift-click または control-click を使い、複数の名前空間を選択します。

高いレベルの名前空間を 1 つ選ぶと、その下のすべての名前空間が選択されます。たとえば、javax.enterprise.system を選択すると、その名前空間に下にあるすべてのモジュールのロガーも選択されます。javax.enterprise.system.core、javax.enterprise.system.tools.admin などです。

- カスタムロガー — 特定のアプリケーションに固有のロガーのメッセージを表示するには、テキストフィールドで 1 行に 1 つずつロガー名を入力します。アプリケーションに複数のモジュールがある場合は、そのいずれかまたはすべてを表示できます。たとえば、使用しているアプリケーションに次の名前のロガーがあるとします。

  ```
  com.mycompany.myapp.module1
  com.mycompany.myapp.module2
  com.mycompany.myapp.module3
  ```

  アプリケーション内のすべてのモジュールのメッセージを表示するには、com.mycompany.myapp と入力します。module2 のメッセージだけを表示するには、com.mycompany.myapp.module2 と入力します。

  1 つ以上のカスタムロガーを指定した場合、Application Server モジュールのメッセージは、「ロガー」領域で明示的に指定されるときだけ表示されます。

- 名前と値のペア — 特定のスレッドの出力を表示するには、テキストフィールドにスレッドのキー名と値を入力します。キー名は _ThreadID です。次に例を示します。

  ```
  _ThreadID=13
  ```

  com.mycompany.myapp.module2 がいくつかのスレッドで実行されるとします。1 つのスレッドの出力だけを表示するようにログビューアを修正するには、「カスタムロガー」フィールドでモジュールのロガーを指定してからこのフィールドにスレッド ID を指定します。

- 表示 — 一度に 40 メッセージ (デフォルト) 以上を表示するには、ドロップダウンリストからほかの可能な値 (100、250、または 1000) を選択します。

  スタックトレースを表示するには、「過度に長いメッセージを制限」チェックボックスのチェックマークを外します。デフォルトでは、スタックトレースはビューアに表示されません。表示するには、メッセージの (詳細) リンクをクリックします。

  「基本検索」をクリックして、「詳細オプション」領域を非表示にします。

◆ ◆ ◆ 第 16 章

16 コンポーネントとサービスの監視

この章では、Application Server の管理コンソールを使ってコンポーネントを監視する方法について説明します。この章には次の節が含まれています。

監視について

Application Server での監視

監視機能を使用して Application Server のサーバーインスタンスに配備されている各種コンポーネントおよびサービスの実行時状態を把握します。実行時コンポーネントとプロセスに関する情報を使用して、チューニングに関わるパフォーマンスボトルネックを識別し、処理能力を計画し、障害を見積もり、障害の場合の原因を分析して、期待通りの機能性を確保できます。

監視をオンにすると、オーバーヘッドの増大によりパフォーマンスが低下します。

監視の概要

Application Server を監視するには、次の手順を実行します。

1. 管理コンソールまたは asadmin ツールを使用して、特定のサービスおよびコンポーネントの監視を有効にします。

   この手順の詳細については、304 ページの「監視の有効化と無効化に関する管理コンソールタスク」を参照してください。
2. 管理コンソールまたは asadmin ツールを使用して、特定のサービスおよびコンポーネントの監視データを表示します。

   この手順の詳細については、306 ページの「監視データの表示に関する管理コンソールタスク」を参照してください。

監視可能なオブジェクトのツリー構造について

Application Server は、ツリー構造を使って監視可能なオブジェクトを追跡します。監視オブジェクトのツリーは動的であり、インスタンス内におけるコンポーネントの追加、更新、削除に応じて変更されます。ツリー内のルートオブジェクトは、server などのサーバーインスタンス名です。Platform Edition では、1 つのサーバーインスタンスしか使用できません。

次のコマンドを実行すると、ツリーのトップレベルが表示されます。

```
asadmin> list --user adminuser --monitor server
server.applications
server.http-service
server.connector-service
server.jms-service
server.jvm
server.orb
server.resources
server.thread-pools
```

次の各節では、これらのサブツリーについて説明します。

- 278 ページの「アプリケーションのツリー」
- 279 ページの「HTTP サービスのツリー」
- 280 ページの「リソースのツリー」
- 280 ページの「コネクタサービスのツリー」
- 280 ページの「JMS サービスのツリー」
- 281 ページの「ORB のツリー」
- 281 ページの「スレッドプールのツリー」

アプリケーションのツリー

次の図に、エンタープライズアプリケーションの各種コンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。詳細については、282 ページの「EJB コンテナの統計」を参照してください。

例 16–1 アプリケーションノードのツリー構造

```
applications
   |--- application1
   |    |--- ejb-module-1
   |    |        |--- ejb1 *
   |    |                 |--- cache (エンティティー/sfsb) *
   |    |                 |--- pool (ステートレスセッション/メッセージ駆動型/エンティティー Bean 用) *
   |    |                 |--- methods
   |    |                      |---method1 *
   |    |                      |---method2 *
   |    |                 |--- stateful-session-store (sfsb)*
   |    |                 |--- timers (ステートレスセッション/エンティティー/mdb) *
   |    |--- web-module-1
   |    |        |--- virtual-server-1 *
   |    |                      |---servlet1  *
   |    |                      |---servlet2  *
   |--- standalone-web-module-1
   |    |        |----- virtual-server-2 *
   |    |                      |---servlet3 *
   |    |                      |---servlet4 *
   |    |        |----- virtual-server-3 *
   |    |                      |---servlet3 *(ほかの仮想サーバーと同一のサーブレット)
   |    |                      |---servlet5 *
   |--- standalone-ejb-module-1
   |    |        |--- ejb2 *
   |    |                 |--- cache (エンティティー/sfsb) *
   |    |                 |--- pool (ステートレスセッション/メッセージ駆動型/エンティティー Bean 用) *
   |    |                 |--- methods
   |    |                      |--- method1 *
   |    |                      |--- method2 *
   |--- application2
```

HTTP サービスのツリー

HTTP サービスのノードを、次の図に示します。監視情報が利用可能なノードには、アスタリスク (*) を付けています。288 ページの「HTTP サービスの統計」を参照してください。

例 16–2 HTTP サービスの図 (Platform Edition 版)

```
http-service
         |--- virtual-server-1
              |--- http-listener-1 *
              |--- http-listener-2 *
         |--- virtual-server-2
              |--- http-listener-1 *
              |--- http-listener-2 *
```

例 16–3 HTTP サービスの図 (Enterprise Edition 版)

```
http-service *
             |---connection-queue *
             |---dns *
             |---file-cache *
             |---keep-alive *
             |---pwc-thread-pool *
             |---virtual-server-1*
             |        |--- request *
             |---virtual-server-2*
             |        |--- request *
```

リソースのツリー

リソースノードには、JDBC 接続プールやコネクタ接続プールなどのプールの監視可能な属性が格納されます。次の図に、各種リソースコンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。289 ページの「JDBC 接続プールの統計」を参照してください。

例 16–4 リソースの図

```
resources
        |---connection-pool1(connector-connection-pool、jdbc のいずれか)*
        |---connection-pool2(connector-connection-pool、jdbc のいずれか)*
```

コネクタサービスのツリー

コネクタサービスノードには、コネクタ接続プールなどのプールの監視可能な属性が格納されます。次の図に、各種コネクタサービスコンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。290 ページの「JMS サービスおよびコネクタサービスの統計」を参照してください。

例 16–5 コネクタサービスの図

```
connector-service
        |--- resource-adapter-1
        |        |-- connection-pools
        |        |        |-- pool-1 (このプールのすべてのプール状態)
        |        |-- work-management (このリソースアダプタのすべての作業管理状態)
```

JMS サービスのツリー

JMS サービスノードには、コネクタ接続プールなどのプールの監視可能な属性が格納されます。次の図に、各種 JMS サービスコンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。

例 16–6 JMS サービスの図

```
jms-service
        |-- connection-factories (リソースアダプタの世界では AKA 接続プールと呼ばれる)
        |        |-- connection-factory-1 (この接続ファクトリのすべての接続ファクトリ状態)
        |-- work-management (この MQ リソースアダプタのすべての作業管理状態)
```

ORB のツリー

ORB ノードには、接続マネージャーの監視可能な属性が格納されます。次の図に、各種 ORB コンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。292 ページの「ORB の接続マネージャーの統計」を参照してください。

例 16–7 ORB の図

```
orb
        |--- connection-managers
        |        |--- connection-manager-1 *
        |        |--- connection-manager-1 *
```

スレッドプールのツリー

スレッドプールノードには、接続マネージャーの監視可能な属性が格納されます。次の図に、各種 ORB コンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。292 ページの「スレッドプールの統計」を参照してください。

例 16–8 スレッドプールの図

```
thread-pools
        |    |--- thread-pool-1 *
        |    |--- thread-pool-2 *
```

監視対象のコンポーネントとサービスの統計について

この節では、利用可能な監視統計について説明します。

EJB コンテナの統計

EJB 統計を次の表に示します。

表 16–1 EJB 統計

属性名	データタイプ	説明
createcount	CountStatistic	特定の EJB に対する create メソッドの呼び出し回数。
removecount	CountStatistic	特定の EJB に対する remove メソッドの呼び出し回数。
pooledcount	RangeStatistic	プールされた状態にあるエンティティー Bean の数。
readycount	RangeStatistic	実行可能状態にあるエンティティー Bean の数。
messagecount	CountStatistic	特定のメッセージ駆動型 Bean に対して受信されたメッセージの数。
methodreadycount	RangeStatistic	MethodReady 状態にあるステートフルまたはステートレスセッション Beans の数。
passivecount	RangeStatistic	Passive 状態にあるステートフルセッション Beans の数。

EJB メソッド呼び出しに関して利用可能な統計を、次の表に示します。

表 16–2 EJB メソッドの統計

属性名	データタイプ	説明
methodstatistic	TimeStatistic	特定の操作の呼び出し回数。その呼び出しにかかった合計時間など。

表 16–2 EJB メソッドの統計　(続き)

属性名	データタイプ	説明
totalnumerrors	CountStatistic	メソッド実行時に例外が発生した回数。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。
totalnumsuccess	CountStatistic	メソッドが正常に実行された回数。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。
executiontime	CountStatistic	成功または失敗した最後の操作実行時にメソッド実行に費やされた時間 (ミリ秒)。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。

EJB セッションストアに関する統計を、次の表に示します。

表 16–3 EJB セッションストアの統計

属性名	データタイプ	説明
currentSize	RangeStatistic	現在ストア内に存在している、非活性化またはチェックポイント化されたセッションの数。
activationCount	CountStatistic	ストアから活性化されたセッションの数。
activationSuccessCount	CountStatistic	ストアからの活性化に成功したセッションの数

表 16–3 EJB セッションストアの統計　(続き)

属性名	データタイプ	説明
activationErrorCount	CountStatistic	成功または失敗した最後の操作実行時にメソッド実行に費やされた時間 (ミリ秒)。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。
passivationCount	CountStatistic	このストアを使って非活性化されたセッションの数。
passivationSuccessCount	CountStatistic	このストアを使って正常に非活性化されたセッションの数。
passivationErrorCount	CountStatistic	このストアを使って非活性化できなかったセッションの数。
expiredSessionCount	CountStatistic	期限切れによりこのストアから削除されたセッションの数。
passivatedBeanSize	CountStatistic	このストアによって非活性化されたバイト数の合計 (合計、最小、最大を含む)。
passivationTime	CountStatistic	Beans のストアへの非活性化に要した時間 (合計、最小、最大を含む)。
checkpointCount (EE のみ)	CountStatistic	このストアを使ってチェックポイント化されたセッションの数。
checkpointSuccessCount (EE のみ)	CountStatistic	正常にチェックポイント化されたセッションの数。
checkpointErrorCount (EE のみ)	CountStatistic	チェックポイント化できなかったセッションの数。
checkpointedBeanSize (EE のみ)	ValueStatistic	ストアによってチェックポイント化された Beans の合計数。
checkpointTime (EE のみ)	TimeStatistic	Beans のストアへのチェックポイント化に要した時間。

EJB プールに関して利用可能な統計を、次の表に示します。

表 16–4 EJB プールの統計

属性名	データタイプ	説明
numbeansinpool	BoundedRangeStatistic	関連付けられたプール内の EJB 数。これにより、プールがどのように変化しているかがわかります。
numthreadswaiting	BoundedRangeStatistic	未使用 Beans を取得するために待機しているスレッドの数。これは、要求が過剰である可能性を示します。
totalbeanscreated	CountStatistic	関連付けられたプール内でデータ収集開始後に作成された Beans の数。
totalbeansdestroyed	CountStatistic	関連付けられたプール内でデータ収集開始後に破棄された Beans の数。
jmsmaxmessagesload	CountStatistic	メッセージ駆動型 Bean のサービスを提供するために JMS セッション内に一度にロード可能なメッセージの最大数。デフォルトは 1。メッセージ駆動型 Beans 用のプールにのみ適用されます。

EJB キャッシュに関して利用可能な統計を、次の表に示します。

表 16–5 EJB キャッシュの統計

属性名	データタイプ	説明
cachemisses	BoundedRangeStatistic	ユーザー要求に対する Bean がキャッシュ内で見つからなかった回数。
cachehits	BoundedRangeStatistic	ユーザー要求に対するエントリがキャッシュ内で見つかった回数。
numbeansincache	BoundedRangeStatistic	キャッシュ内の Beans 数。これは現在のキャッシュサイズです。
numpassivations	CountStatistic	非活性化された Bean の数。ステートフルセッション Beans にのみ適用されます。
numpassivationerrors	CountStatistic	非活性化中に発生したエラーの数。ステートフルセッション Beans にのみ適用されます。

表 16–5 EJB キャッシュの統計　(続き)

属性名	データタイプ	説明
numexpiredsessionsremoved	CountStatistic	クリーンアップスレッドによって削除された期限切れセッションの数。ステートフルセッション Beans にのみ適用されます。
numpassivationsuccess	CountStatistic	非活性化が正常に終了した回数。ステートフルセッション Beans にのみ適用されます。

タイマーに関して利用可能な統計を、次の表に示します。

表 16–6 タイマーの統計

Statistic	データタイプ	説明
numtimerscreated	CountStatistic	システム内で作成されたタイマーの数。
numtimersdelivered	CountStatistic	システムによって配信されたタイマーの数。
numtimersremoved	CountStatistic	システムから削除されたタイマーの数。

Web コンテナの統計

Web コンテナは、278 ページの「アプリケーションのツリー」に示したオブジェクトツリー内に含まれます。Web コンテナの統計は、個々の Web アプリケーションごとに表示されます。Web コンテナのサーブレットに関して利用可能な統計を 286 ページの「Web コンテナの統計」に、Web モジュールに関して利用可能な統計を 286 ページの「Web コンテナの統計」に、それぞれ示します。

表 16–7 Web コンテナ (サーブレット) の統計

Statistic	単位	データタイプ	補足説明
errorcount	数値	CountStatistic	応答コードが 400 以上になった場合の累計件数。
maxtime	ミリ秒	CountStatistic	Web コンテナの要求待ち状態の最大継続時間。
processingtime	ミリ秒	CountStatistic	各要求の処理に要した時間の累計値。この処理時間は、要求処理時間を要求数で割って得られた平均値です。
requestcount	数値	CountStatistic	その時点までに処理された要求の合計数。

Web モジュールに関して利用可能な統計を、286 ページの「Web コンテナの統計」に示します。

表 16–8 Web コンテナ (Web モジュール) の統計

Statistic	データタイプ	補足説明
jspcount	CountStatistic	この Web モジュール内に読み込まれた JSP ページの数。
jspreloadcount	CountStatistic	この Web モジュール内に再読み込みされた JSP ページの数。
sessionstotal	CountStatistic	この Web モジュールに対して作成されたセッションの合計数。
activesessionscurrent	CountStatistic	この Web モジュールで現在アクティブになっているセッションの数。
activesessionshigh	CountStatistic	この Web モジュールで同時にアクティブになれるセッションの最大数。
rejectedsessionstotal	CountStatistic	この Web モジュールで拒否されたセッションの合計数。これは、最大許可セッション数がすでにアクティブになっていたために作成されなかったセッションの数です。
expiredsessionstotal	CountStatistic	この Web モジュールで期限切れになったセッションの合計数。
sessionsize (EE のみ)	AverageRangeStatistic	この Web モジュールのセッションのサイズ。値は high、low、average のいずれかです。ただし、直列化されたセッションの場合はバイト値になります。
containerlatency (EE のみ)	AverageRangeStatistic	応答時間要求全体の Web コンテナ部分の応答時間。値は high、low、average のいずれかです。

表 16–8 Web コンテナ (Web モジュール) の統計　(続き)

Statistic	データタイプ	補足説明
sessionpersisttime (EE のみ)	AverageRangeStatistic	この Web モジュールの HTTP セッション状態のバックエンドストアへの持続化に要した時間 (ミリ秒値、low、high、average のいずれか)。
cachedsessionscurrent (EE のみ)	CountStatistic	この Web モジュールで現在メモリー内にキャッシュされているセッションの数。
passivatedsessionscurrent (EE のみ)	CountStatistic	この Web モジュールで現在非活性化されているセッションの数。

HTTP サービスの統計

HTTP サービスに関して利用可能な統計を、288 ページの「HTTP サービスの統計」に示します。この統計は Platform Edition のみに適用されます。Enterprise Edition の場合の HTTP サービス統計については、298 ページの「PWC (Production Web Container) の統計」を参照してください。

表 16–9 HTTP サービスの統計 (Platform Edition のみに適用)

Statistic	単位	データタイプ	補足説明
bytesreceived	バイト	CountStatistic	各要求プロセッサが受信したバイトの累計値。
bytessent	バイト	CountStatistic	各要求プロセッサが送信したバイトの累計値。
currentthreadcount	数値	CountStatistic	リスナースレッドプール内に現在存在している処理スレッドの数。
currentthreadsbusy	数値	CountStatistic	要求処理用リスナースレッドプール内で現在使用されている要求処理スレッドの数。
errorcount	数値	CountStatistic	エラー回数の累計値。これは、応答コードが 400 以上になった場合の回数を表します。
maxsparethreads	数値	CountStatistic	存在可能な未使用応答処理スレッドの最大数。
minsparethreads	数値	CountStatistic	存在可能な未使用応答処理スレッドの最小数。

表 16–9 HTTP サービスの統計 (Platform Edition のみに適用)　(続き)

Statistic	単位	データタイプ	補足説明
maxthreads	数値	CountStatistic	リスナーが作成する要求処理スレッドの最大数。
maxtime	ミリ秒	CountStatistic	スレッド処理時間の最大値。
processing-time	ミリ秒	CountStatistic	各要求の処理に要した時間の累計値。この処理時間は、要求処理時間を要求数で割って得られた平均値です。
request-count	数値	CountStatistic	その時点までに処理された要求の合計数。

JDBC 接続プールの統計

JDBC リソースを監視することで、パフォーマンスを測定するとともに、実行時のリソースの使用状況を把握します。JDBC 接続の作成はコストのかかる処理であり、アプリケーションのパフォーマンス上のボトルネックになることが多いため、JDBC 接続プールで新しい接続がどのように解放/作成されているかや、特定のプールから接続を取得するために待機しているスレッドがどれくらい存在するかを監視することが不可欠です。

JDBC 接続プールに関して利用可能な統計を、次の表に示します。

表 16–10 JDBC 接続プールの統計

Statistic	単位	データタイプ	説明
numconnfailedvalidation	数値	CountStatistic	開始時刻から前回のサンプリング時刻までの間に検証に失敗した接続プール内の接続の合計数。
numconnused	数値	RangeStatistic	接続の使用状況に関する統計を提供します。現在使用されている合計接続数に加え、過去に使用された接続の最大数 (ハイウォーターマーク) に関する情報も提供します。
numconnfree	数値	RangeStatistic	前回のサンプリング時点におけるプール内の未使用接続の合計数。
numconntimedout	数値	BoundedRangeStatistic	開始時刻から前回のサンプリング時刻までの間にタイムアウトしたプール内の接続の合計数。

表 16–10 JDBC 接続プールの統計　(続き)

Statistic	単位	データタイプ	説明
averageconnwaittime	数値	CountStatistic	コネクタ接続プールに対する接続要求が成功した場合の平均接続待ち時間を示します。
waitqueuelength	数値	CountStatistic	サービスを受けるためにキュー内で待機している接続要求の数。
connectionrequestwaittime		RangeStatistic	接続要求の最長待ち時間と最短待ち時間。現在の値は、プールのサービスを最後に受けた要求の待ち時間を示します。
numconncreated	ミリ秒	CountStatistic	前回のリセット後に作成された物理接続の数。
numconndestroyed	数値	CountStatistic	前回のリセット後に破棄された物理接続の数。
numconnacquired	数値	CountStatistic	プールから取得された論理接続の数。
numconnreleased	数値	CountStatistic	プールに解放された論理接続の数。

JMS サービスおよびコネクタサービスの統計

コネクタ接続プールに関して利用可能な統計を、290 ページの「JMS サービスおよびコネクタサービスの統計」に示します。コネクタ作業管理に関する統計を、290 ページの「JMS サービスおよびコネクタサービスの統計」に示します。

表 16–11 コネクタ接続プールの統計

Statistic	単位	データタイプ	説明
numconnfailedvalidation	数値	CountStatistic	開始時刻から前回のサンプリング時刻までの間に検証に失敗した接続プール内の接続の合計数。
numconnused	数値	RangeStatistic	接続の使用状況に関する統計を提供します。現在使用されている合計接続数に加え、過去に使用された接続の最大数 (ハイウォーターマーク) に関する情報も提供します。
numconnfree	数値	RangeStatistic	前回のサンプリング時点におけるプール内の未使用接続の合計数。

表 16–11 コネクタ接続プールの統計　(続き)

Statistic	単位	データタイプ	説明
numconntimedout	数値	CountStatistic	開始時刻から前回のサンプリング時刻までの間にタイムアウトしたプール内の接続の合計数。
averageconnwaittime	数値	CountStatistic	接続プールからサービスを受けるまでにかかった平均接続待ち時間。
waitqueuelenght	数値	CountStatistic	サービスを受けるためにキュー内で待機している接続要求の数。
connectionrequestwaittime		RangeStatistic	接続要求の最長待ち時間と最短待ち時間。現在の値は、プールのサービスを最後に受けた要求の待ち時間を示します。
numconncreated	ミリ秒	CountStatistic	前回のリセット後に作成された物理接続の数。
numconndestroyed	数値	CountStatistic	前回のリセット後に破棄された物理接続の数。
numconnacquired	数値	CountStatistic	プールから取得された論理接続の数。
numconnreleased	数値	CountStatistic	プールに解放された論理接続の数。

コネクタ作業管理に関して利用可能な統計を、290 ページの「JMS サービスおよびコネクタサービスの統計」に示します。

表 16–12 コネクタ作業管理の統計

Statistic	データタイプ	説明
activeworkcount	RangeStatistic	コネクタによって実行された作業オブジェクトの数。
waitqueuelength	RangeStatistic	実行される前にキュー内で待機している作業オブジェクトの数。
workrequestwaittime	RangeStatistic	作業オブジェクトが実行されるまでの最長待ち時間と最短待ち時間。
submittedworkcount	CountStatistic	コネクタモジュールによって送信された作業オブジェクトの数。
rejectedworkcount	CountStatistic	Application Server によって拒否された作業オブジェクトの数。
completedworkcount	CountStatistic	完了した作業オブジェクトの数。

ORB の接続マネージャーの統計

ORB の接続マネージャーに関して利用可能な統計を、292 ページの「ORB の接続マネージャーの統計」に示します。

表 16–13 ORB の接続マネージャーの統計

Statistic	単位	データタイプ	説明
connectionsidle	数値	CountStatistic	ORB への接続のうち、アイドル状態のものの合計数を提供します。
connectionsinuse	数値	CountStatistic	ORB への接続のうち、使用中のものの合計数を提供します。
totalconnections	数値	BoundedRangeStatistic	ORB への接続の合計数。

スレッドプールの統計

スレッドプールに関して利用可能な統計を、次の表に示します。

表 16–14 スレッドプールの統計

Statistic	単位	データタイプ	説明
averagetimeinqueue	ミリ秒	RangeStatistic	キュー内の要求が処理されるまでの平均待ち時間 (ミリ秒)。
averageworkcompletion-time	ミリ秒	RangeStatistic	1 つの作業の平均完了時間 (ミリ秒)。
currentnumberofthreads	数値	BoundedRangeStatistic	要求処理スレッドの現在の数。
numberofavailablethreads	数値	CountStatistic	利用可能なスレッドの数。
numberofbusythreads	数値	CountStatistic	ビジー状態のスレッドの数。
totalworkitemsadded	数値	CountStatistic	その時点までに作業キューに追加された作業項目の合計数。

トランザクションサービスの統計

トランザクションサービスを使えば、クライアントはトランザクションサブシステムをフリーズできます。フリーズすると、トランザクションをロールバックしたり、フリーズ時点で処理中であったトランザクションを特定したりできます。トランザクションサービスに関して利用可能な統計を、次の表に示します。

表 16–15 トランザクションサービスの統計

Statistic	データタイプ	説明
activecount	CountStatistic	現在アクティブなトランザクションの数。
activeids	StringStatistic	現在アクティブなトランザクションの ID。それらの各トランザクションは、トランザクションサービスのフリーズ後にロールバックすることができます。
committedcount	CountStatistic	コミットされたトランザクションの数。
rolledbackcount	CountStatistic	ロールバックされたトランザクションの数。
state	StringStatistic	トランザクションがフリーズされたかどうかを示します。

Java 仮想マシン (JVM) の統計

JVM の監視可能な属性は、常に有効になっています。JVM に関して利用可能な統計を、次の表に示します。

表 16–16 JVM の統計

Statistic	データタイプ	説明
heapsize	BoundedRangeStatistic	JVM のメモリーヒープサイズの上限と下限の間にある常駐メモリーフットプリント。
uptime	CountStatistic	JVM の稼働時間。

J2SE 5.0 の JVM 統計

Application Server がバージョン 5.0 以上の J2SE 上で動作するように設定されている場合、JVM から追加の監視情報を取得できます。監視レベルを「低」に設定すると、この追加情報の表示が有効になります。監視レベルを「高」に設定すると、さらにシステム内の各ライブスレッドに関する情報も表示されます。J2SE 5.0 で利用可能な追加監視機能の詳細については、『Monitoring and Management for the Java Platform』というタイトルの文書を参照してください。この文書は
http://java.sun.com/j2se/1.5.0/docs/guide/management/ で利用可能になっています。

J2SE 5.0 の監視ツールについては、http://java.sun.com/j2se/1.5.0/docs/tooldocs/#manage を参照してください。

J2SE 5.0 の JVM で利用可能なクラス読み込み関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 **16–17** J2SE 5.0 の JVM 統計- クラス読み込み

Statistic	データタイプ	説明
`loadedclasscount`	CountStatistic	JVM 内に現在読み込まれているクラスの数。
`totalloadedclasscount`	CountStatistic	JVM の実行開始後に読み込まれたクラスの合計数。
`unloadedclasscount`	CountStatistic	JVM の実行開始後に JVM から読み込み解除されたクラスの数。

J2SE 5.0 の JVM で利用可能なコンパイル関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 **16–18** J2SE 5.0 の JVM 統計 - コンパイル

Statistic	データタイプ	説明
`totalcompilationtime`	CountStatistic	コンパイルに費やされた時間の累計 (ミリ秒)。

J2SE 5.0 の JVM で利用可能なガベージコレクション関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 **16–19** J2SE 5.0 の JVM 統計 - ガベージコレクション

Statistic	データタイプ	説明
`collectioncount`	CountStatistic	実行されたコレクションの合計回数。
`collectiontime`	CountStatistic	コレクション時間の累計値 (ミリ秒)。

J2SE 5.0 の JVM で利用可能なメモリー関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 **16–20** J2SE 5.0 の JVM 統計 - メモリー

Statistic	データタイプ	説明
`objectpendingfinalizationcount`	CountStatistic	ファイナライズを保留しているオブジェクトの概算数。

表 16–20 J2SE 5.0 の JVM 統計 - メモリー　(続き)

Statistic	データタイプ	説明
initheapsize	CountStatistic	JVM が最初に要求したヒープのサイズ。
usedheapsize	CountStatistic	現在使用されているヒープのサイズ。
maxheapsize	CountStatistic	メモリー管理用として使用可能なメモリーの最大サイズ (バイト)。
committedheapsize	CountStatistic	JVM 用としてコミットされたメモリーのサイズ (バイト)。
initnonheapsize	CountStatistic	JVM が最初に要求した非ヒープ領域のサイズ。
usednonheapsize	CountStatistic	現在使用されている非ヒープ領域のサイズ。
maxnonheapsize	CountStatistic	メモリー管理用として使用可能なメモリーの最大サイズ (バイト)。
committednonheapsize	CountStatistic	JVM 用としてコミットされたメモリーのサイズ (バイト)。

J2SE 5.0 の JVM で利用可能なオペレーティングシステム関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–21 J2SE 5.0 の JVM 統計 - オペレーティングシステム

Statistic	データタイプ	説明
arch	StringStatistic	オペレーティングシステムのアーキテクチャー。
availableprocessors	CountStatistic	JVM が使用できるプロセッサの数。
name	StringStatistic	オペレーティングシステムの名前。
version	StringStatistic	オペレーティングシステムのバージョン。

J2SE 5.0 の JVM で利用可能なランタイム関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–22 J2SE 5.0 の JVM 統計 - ランタイム

Statistic	データタイプ	説明
name	StringStatistic	実行中の JVM を表す名前
vmname	StringStatistic	JVM 実装の名前。
vmvendor	StringStatistic	JVM 実装のベンダー。
vmversion	StringStatistic	JVM 実装のバージョン。
specname	StringStatistic	JVM 仕様の名前。
specvendor	StringStatistic	JVM 仕様のベンダー。
specversion	StringStatistic	JVM 仕様のバージョン。
managementspecversion	StringStatistic	JVM が実装している管理仕様のバージョン。
classpath	StringStatistic	システムクラスローダーがクラスファイルの検索時に使用するクラスパス。
librarypath	StringStatistic	Java のライブラリパス。
bootclasspath	StringStatistic	ブートストラップクラスローダーがクラスファイルの検索時に使用するクラスパス。
inputarguments	StringStatistic	JVM に渡された入力引数。main メソッドに対する引数は含みません。
uptime	CountStatistic	JVM の稼働時間 (ミリ秒)。

J2SE 5.0 の JVM で利用可能な ThreadInfo 関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–23 J2SE 5.0 の JVM 統計 - ThreadInfo

Statistic	データタイプ	説明
threadid	CountStatistic	スレッドの ID。
threadname	StringStatistic	スレッドの名前。
threadstate	StringStatistic	スレッドの状態。
blockedtime	CountStatistic	このスレッドが BLOCKED 状態に入ったあと経過した時間 (ミリ秒)。スレッド競合監視が無効になっている場合は、-1 が返されます。

表 16–23 J2SE 5.0 の JVM 統計 - ThreadInfo　(続き)

Statistic	データタイプ	説明
blockedcount	CountStatistic	このスレッドが BLOCKED 状態に入った合計回数。
waitedtime	CountStatistic	スレッドが WAITING 状態に入ったあと経過した時間 (ミリ秒)。スレッド競合監視が無効になっている場合は、-1 が返されます。
waitedcount	CountStatistic	スレッドが WAITING 状態または TIMED_WAITING 状態になった合計回数。
lockname	StringStatistic	このスレッドが獲得をブロックされている監視ロック、またはこのスレッドが Object.wait メソッド経由で通知されるのを待っている監視ロックの文字列表現。
lockownerid	CountStatistic	このスレッドのブロック対象オブジェクトの監視ロックを保持しているスレッドの ID。
lockownername	StringStatistic	このスレッドのブロック対象オブジェクトの監視ロックを保持しているスレッドの名前。
stacktrace	StringStatistic	このスレッドに関連付けられているスタックトレース。

J2SE 5.0 の JVM で利用可能なスレッド関連の統計を、293 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–24 J2SE 5.0 の JVM 統計 - スレッド

Statistic	データタイプ	説明
threadcount	CountStatistic	ライブデーモンスレッドと非デーモンスレッドの現在の数。
peakthreadcount	CountStatistic	JVM 起動後またはピーク値リセット後におけるライブスレッドのピーク数。
totalstartedthreadcount	CountStatistic	JVM が起動されて以来、作成されたスレッド、起動されたスレッド、作成および起動されたスレッドの合計数。

表 16–24 J2SE 5.0 の JVM 統計 - スレッド　(続き)

Statistic	データタイプ	説明
daemonthreadcount	CountStatistic	ライブデーモンスレッドの現在の数。
allthreadids	StringStatistic	すべてのライブスレッド ID のリスト。
currentthreadcputime	CountStatistic	CPU 時間の測定が有効になっている場合は、現在のスレッドに対する CPU 時間 (ナノ秒)。CPU 時間の測定が無効になっている場合は、-1 が返されます。
monitordeadlockedthreads	StringStatistic	監視デッドロックが発生しているスレッド ID のリスト。

PWC (Production Web Container) の統計

Application Server の Enterprise Edition (EE) では、PWC の次のコンポーネントとサービスに関する統計が利用可能です。

- 298 ページの「PWC (Production Web Container) の統計」、PWC 仮想サーバー
- 298 ページの「PWC (Production Web Container) の統計」、PWC 要求
- 298 ページの「PWC (Production Web Container) の統計」、PWC ファイルキャッシュ
- 298 ページの「PWC (Production Web Container) の統計」、PWC キープアライブ
- 298 ページの「PWC (Production Web Container) の統計」、PWC DNS
- 192 ページの「リスナーと JMX コネクタに関する管理コンソールタスク」、PWC スレッドプール
- 298 ページの「PWC (Production Web Container) の統計」、PWC 接続キュー
- 298 ページの「PWC (Production Web Container) の統計」、PWC HTTP サービス

PWC 仮想サーバーに関する統計を、298 ページの「PWC (Production Web Container) の統計」に示します。

表 16–25 PWC 仮想サーバーの統計 (EE のみ)

属性名	データタイプ	説明
id	StringStatistic	仮想サーバーの ID。
mode	StringStatistic	仮想サーバーのモード。unknown、active のいずれかを選択できます。
hosts	StringStatistic	この仮想サーバーからサービスを受けているホストの名前。
interfaces	StringStatistic	仮想サーバーに設定されたインタフェース (リスナー) のタイプ。

PWC 要求に関して利用可能な統計を、次の表に示します。

表 16–26 PWC 要求の統計 (EE のみ)

属性名	データタイプ	説明
method	StringStatistic	要求で使用されたメソッド。
uri	StringStatistic	最後に処理された URI。
countrequests	CountStatistic	処理された要求の数。
countbytestransmitted	CountStatistic	送信バイト数。この情報が利用不可能な場合は 0
countbytesreceived	CountStatistic	受信バイト数。この情報が利用不可能な場合は 0。
ratebytesreceived	CountStatistic	サーバーで定義されたある期間内に受信されたデータの受信速度。この情報が利用不可能な場合は 0
maxbytestransmissionrate	CountStatistic	サーバーで定義されたある期間内に送信されたデータの最大送信速度。この情報が利用不可能な場合は 0。
countopenconnections	CountStatistic	現在開いている接続の数。この情報が利用不可能な場合は 0。
maxopenconnections	CountStatistic	同時に開ける接続の最大数。この情報が利用不可能な場合は 0。
count2xx	CountStatistic	コード 2XX の応答の合計数。
count3xx	CountStatistic	コード 3XX の応答の合計数。
count4xx	CountStatistic	コード 4XX の応答の合計数。
count5xx	CountStatistic	コード 5XX の応答の合計数。
countother	CountStatistic	その他の応答コードを含む応答の合計数。
count200	CountStatistic	コード 200 の応答の合計数。
count302	CountStatistic	コード 302 の応答の合計数。
count304	CountStatistic	コード 304 の応答の合計数。
count400	CountStatistic	コード 400 の応答の合計数。
count401	CountStatistic	コード 401 の応答の合計数。

表 16–26 PWC 要求の統計 (EE のみ)　(続き)

属性名	データタイプ	説明
count403	CountStatistic	コード 403 の応答の合計数。
count404	CountStatistic	コード 404 の応答の合計数。
count503	CountStatistic	コード 503 の応答の合計数。

キャッシュ情報セクションでは、ファイルキャッシュの使用状況に関する情報が提供されます。PWC ファイルキャッシュに関する統計を、次の表に示します。

表 16–27 PWC ファイルキャッシュの統計 (EE のみ)

属性名	データタイプ	説明
flagenabled	CountStatistic	ファイルキャッシュが有効になっているかどうかを示します。有効な値は、0 (no)、1 (yes) のいずれかです。
secondsmaxage	CountStatistic	有効なキャッシュエントリの最大有効期間 (秒)。
countentries	CountStatistic	現在のキャッシュエントリの数。単一のキャッシュエントリは単一の URI を表します。
maxentries	CountStatistic	同時に存在可能なキャッシュエントリの最大数。
countopenentries	CountStatistic	特定のオープンファイルに関連付けられたエントリの数。
maxopenentries	CountStatistic	特定のオープンファイルに同時に関連付けることのできるキャッシュエントリの最大数。
sizeheapcache	CountStatistic	キャッシュコンテンツ格納用のヒープ領域。
maxheapcachesize	CountStatistic	キャッシュファイルコンテンツ格納用のヒープ領域の最大サイズ。
sizemmapcache	CountStatistic	メモリーにマップされたファイルのコンテンツ用として使用するアドレス空間。
maxmmapcachesize	CountStatistic	ファイルキャッシュがメモリーにマップされたファイルのコンテンツ用として使用するアドレス空間の最大サイズ。
counthits	CountStatistic	キャッシュ検索の成功回数。
countmisses	CountStatistic	キャッシュ検索の失敗回数。
countinfohits	CountStatistic	ファイル情報検索の成功回数。

表 16–27 PWC ファイルキャッシュの統計 (EE のみ)　(続き)

属性名	データタイプ	説明
countinfomisses	CountStatistic	キャッシュファイル情報検索の失敗回数。
countcontenthits	CountStatistic	キャッシュファイルコンテンツのヒット数。
countcontentmisses	CountStatistic	ファイル情報検索の失敗回数。

このセクションでは、サーバーの HTTP レベルキープアライブシステムに関する情報が提供されます。PWC キープアライブに関して利用可能な統計を、次の表に示します。

表 16–28 PWC キープアライブの統計 (EE のみ)

属性名	データタイプ	説明
countconnections	CountStatistic	キープアライブモードの接続の数。
maxconnections	CountStatistic	同時に存在可能なキープアライブモード接続の最大数。
counthits	CountStatistic	キープアライブモードの接続が有効な要求を生成した回数の合計。
countflushes	CountStatistic	キープアライブ接続がサーバーによって閉じられた回数。
countrefusals	CountStatistic	サーバーがキープアライブスレッドに接続を渡せなかった回数。その原因はおそらく、持続接続が多すぎたことにあります。
counttimeouts	CountStatistic	クライアント接続が何の活動も見られないままタイムアウトに達し、サーバーがそのキープアライブ接続を終了した回数。
secondstimeout	CountStatistic	アイドル状態のキープアライブ接続が閉じられるまでの時間 (秒)。

DNS キャッシュでは、IP アドレスと DNS 名がキャッシュされます。サーバーの DNS キャッシュはデフォルトで無効になっています。単一のキャッシュエントリは、単一の IP アドレスまたは単一の DNS 名の検索を表します。PWC DNS に関して利用可能な統計を、次の表に示します。

表 16–29 PWC DNS の統計 (EE のみ)

属性名	データタイプ	説明
flagcacheenabled	CountStatistic	DNS キャッシュが有効 (オン) になっているかどうかを示します。0 (off)、1 (on) のいずれかです。
countcacheentries	CountStatistic	キャッシュ内に現在存在している DNS エントリの数。
maxcacheentries	CountStatistic	キャッシュ内に格納できる DNS エントリの最大数。
countcachehits	CountStatistic	DNS キャッシュ検索の成功回数。
countcachemisses	CountStatistic	DNS キャッシュ検索の失敗回数。
flagasyncenabled	CountStatistic	非同期 DNS 検索が有効 (オン) になっているかどうかを示します。0 (off)、1 (on) のいずれかです。
countasyncnamelookups	CountStatistic	非同期 DNS 名検索の合計回数。
countasyncaddrlookups	CountStatistic	非同期 DNS アドレス検索の合計回数。
countasynclookupsinprogress	CountStatistic	処理中の非同期検索の数。

PWC スレッドプールに関する統計を、次の表に示します。

表 16–30 PWC スレッドプールの統計 (EE のみ)

属性名	データタイプ	説明
id	StringStatistic	スレッドプールの ID。
countthreadsidle	CountStatistic	現在アイドル状態になっている要求処理スレッドの数。
countthreads	CountStatistic	要求処理スレッドの現在の数。
maxthreads	CountStatistic	同時に存在可能な要求処理スレッドの最大数。
countqueued	CountStatistic	このスレッドプールの処理待ちキューに格納されている要求の数。
peakqueued	CountStatistic	キュー内に同時に格納された要求の最大数。

表 16–30 PWC スレッドプールの統計 (EE のみ)　(続き)

属性名	データタイプ	説明
maxqueued	CountStatistic	キュー内に同時に格納可能な要求の最大数。

接続キューとは、処理される前の要求が格納されるキューのことです。接続キューの統計は、キュー内のセッション数や接続が受け付けられるまでの平均遅延時間などを示します。PWC 接続キューに関する統計を、次の表に示します。

表 16–31 PWC 接続キューの統計 (EE のみ)

属性名	データタイプ	説明
id	StringStatistic	接続キューの ID。
counttotalconnections	CountStatistic	受け付けられた接続の合計数。
countqueued	CountStatistic	キュー内に現在存在している接続の数。
peakqueued	CountStatistic	キュー内に同時に存在していた接続の最大数。
maxqueued	CountStatistic	接続キューの最大サイズ。
countoverflows	CountStatistic	キューがいっぱいになったために接続を格納できなかった回数。
counttotalqueued	CountStatistic	キューに格納された接続の合計数。1 つの接続がキュー内に複数回格納される可能性があります。このため、counttotalqueued の値は、counttotalconnections の値以上になります。
tickstotalqueued	CountStatistic	接続がキュー内で費やした合計ティック数。ティックは、システムに依存する時間の単位です。
countqueued1minuteaverage	CountStatistic	キュー内接続数の過去 1 分間における平均値。
countqueued5minuteaverage	CountStatistic	キュー内接続数の過去 5 分間における平均値。
countqueued15minuteaverage	CountStatistic	キュー内接続数の過去 15 分間における平均値。

PWC HTTP サービスに関する統計を、次の表に示します。

表 16–32 PWC HTTP サービスの統計 (EE のみ)

属性名	データタイプ	説明
id	StringStatistic	HTTP サービスのインスタンス名。
versionserver	StringStatistic	HTTP サービスのバージョン番号。
timestarted	StringStatistic	HTTP サービスが起動された時刻 (GMT)。
secondsrunning	CountStatistic	HTTP サービス起動後の経過時間 (秒)。
maxthreads	CountStatistic	各インスタンス内のワークスレッドの最大数。
maxvirtualservers	CountStatistic	各インスタンス内に設定可能な仮想サーバーの最大数。
flagprofilingenabled	CountStatistic	HTTP サービスのパフォーマンスプロファイリングが有効になっているかどうか。有効な値は 0、1 のいずれかです。
flagvirtualserveroverflow	CountStatistic	maxvirtualservers を超える仮想サーバーが設定されているかどうかを示します。これを 1 に設定すると、すべての仮想サーバーで統計が追跡されなくなります。
load1minuteaverage	CountStatistic	要求負荷の過去 1 分間における平均値。
load5minuteaverage	CountStatistic	要求負荷の過去 5 分間における平均値。
load15minuteaverage	CountStatistic	要求負荷の過去 15 分間における平均値。
ratebytestransmitted	CountStatistic	サーバーで定義されたある期間内に送信されたデータの送信速度。この情報が利用不可能な場合、結果は 0 になります。
ratebytesreceived	CountStatistic	サーバーで定義されたある期間内に受信されたデータの受信速度。この情報が利用不可能な場合、結果は 0 になります。

監視の有効化と無効化に関する管理コンソールタスク

▼ 管理コンソールを使用して監視レベルを設定する

1 「監視サービス」ページにアクセスします。それには、次の手順に従います。

a. ツリーコンポーネントで、「設定」ノードを展開します。

b. ツリーで、server-config など、監視の設定を行うサーバーインスタンスのノードを開きます。

c. ツリーで、「監視」を選択します。

2 「監視サービス」ページで、監視レベルを変更するコンポーネントまたはサービスの横にあるコンボボックスから適切な値を選択します。

デフォルトでは、すべてのコンポーネントとサービスで監視がオフになっています。監視をオンにするには、コンボボックスから「低」または「高」を選択します。監視をオフにするには、コンボボックスから「オフ」を選択します。次のコンポーネントおよびサービスに対して、監視機能をオンまたはオフにできます。

- **JVM** - Java 仮想マシンを監視するには、このオプションの監視レベルを「低」に設定します。
- **HTTP** サービス - すべてのHTTP リスナーと仮想サーバーを監視するには、このオプションの監視レベルを「低」に設定します。
- トランザクションサービス - トランザクションサブシステムを監視するには、このオプションの監視レベルを「低」に設定します。
- **JMS/**コネクタサービス - Java メッセージサービス (JMS) を監視するには、このオプションの監視レベルを「低」に設定します。
- **ORB** - Application Server コアとその接続マネージャーが使用するシステム ORB を監視するには、このオプションの監視レベルを「低」に設定します。
- **Web** コンテナ - すべての配備サーブレットを監視するには、このオプションの監視レベルを「低」に設定します。
- **EJB** コンテナ - すべての配備 EJB コンポーネント、EJB プール、および EJB キャッシュを監視するには、このオプションの監視レベルを「低」に設定します。EJB ビジネスメソッドも監視するには、このメソッドを「高」に設定します。
- **JDBC** 接続プール - すべての JDBC 接続プールを監視するには、このオプションの監視レベルを「低」に設定します。
- スレッドプール - すべてのスレッドプールを監視するには、このオプションの監視レベルを「低」に設定します。

3 「保存」をクリックします。

このリリースには「監視サービスの追加プロパティー」は存在しないので、「追加プロパティー」の表は無視してかまいません。

参考 同機能を持つ asadmin コマンド

set

たとえば、HTTP サービスの監視をオンにするには、次の asadmin コマンドを使用します。

```
asadmin> set --user admin-user
server.monitoring-service.module-monitoring-levels.http-service=LOW
```

▼ asadmin ツールを使用して監視を設定する

監視をオフにしたり、特定のコンポーネントやサービスの監視レベルを設定したりするには、304 ページの「管理コンソールを使用して監視レベルを設定する」で説明した管理コンソールを使用してもかまいません。

1. get コマンドを使って監視が現在有効になっているサービスとコンポーネントを確認します。

   ```
   asadmin> get --user admin-user server.monitoring-service.module-monitoring-levels.*
   ```

   次の結果が返されます。

   ```
   server.monitoring-service.module-monitoring-levels.
   connector-connection-pool = OFF
   server.monitoring-service.module-monitoring-levels.
   connector-service = OFF
   server.monitoring-service.module-monitoring-levels.ejb-container = OFF
   server.monitoring-service.module-monitoring-levels.http-service = OFF
   server.monitoring-service.module-monitoring-levels.jdbc-connection-pool = OFF
   server.monitoring-service.module-monitoring-levels.jms-service = OFF
   server.monitoring-service.module-monitoring-levels.jvm = OFF
   server.monitoring-service.module-monitoring-levels.orb = OFF
   server.monitoring-service.module-monitoring-levels.thread-pool = OFF
   server.monitoring-service.module-monitoring-levels.transaction-service = OFF
   server.monitoring-service.module-monitoring-levels.web-container = OFF
   ```

2. set コマンドを使って監視を有効にします。

   たとえば、HTTP サービスの監視を有効にするには、次のようにします。

   ```
   asadmin> set --user admin-user
   server.monitoring-service.module-monitoring-levels.http-service=LOW
   ```

   監視を無効にするには、set コマンドを使って監視レベルに OFF を指定します。

監視データの表示に関する管理コンソールタスク

▼ 管理コンソールで監視データを表示する

各コンポーネントやサービスの属性の詳細については、281 ページの「監視対象のコンポーネントとサービスの統計について」を参照してください。

1 「監視」ページにアクセスします。それには、次の手順に従います。

   a. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを展開します。

   b. リストから、**server (Admin Server)** など、特定のスタンドアロンサーバーインスタンスを選択します。

   c. 「監視」ページを選択します。

   d. 「監視」ページの「監視」タブを選択します。

2 「ビュー」リストから、サーバーインスタンスに配備された、監視が有効なコンポーネントまたはサービスを選択します。

   選択したコンポーネントまたはサービスの監視データが「ビュー」フィールドの下に表示されます。監視可能なプロパティーについては、281 ページの「監視対象のコンポーネントとサービスの統計について」を参照してください。

   このページでは、これらのコンポーネントおよびサービスに対して監視機能が有効な場合、JVM、サーバー、スレッドプール、HTTP サービスおよびトランザクションサービスの監視データを表示できます。これらのコンポーネントやサービスの構成を示した図については、278 ページの「監視可能なオブジェクトのツリー構造について」を参照してください。

3 監視したいコンポーネントまたはサービスがこのリスト内に見つからない場合は、「監視を設定」リンクを選択し、そのコンポーネントやサービスの監視を有効または無効にしてください。

   「オフ」を選択すると、コンポーネントまたはサービスの監視が無効になります。「低」または「高」を選択すると、コンポーネントまたはサービスの監視が有効になります。

   監視の有効化と無効化の詳細については、304 ページの「管理コンソールを使用して監視レベルを設定する」または306 ページの「asadmin ツールを使用して監視を設定する」を参照してください。

4 「アプリケーション」タブ (「監視」ページ) を選択して、サーバーインスタンスに配備され、監視が有効なアプリケーションコンポーネントの監視データを表示します。「アプリケーション」リストからアプリケーションを選択します。「コンポーネント」リストから特定のコンポーネントを選択します。

   アプリケーションまたはコンポーネントに対する監視データが表示されない場合、「監視を設定」リンクを選択し、そのコンポーネントまたはサービスに対する監視を有効または無効にしてください。アプリケーションを監視するには、それを実行するコンテナ

の監視をオンにします。たとえば、Web アプリケーションの場合は Web コンテナで、EJB アプリケーションの場合は EJB コンテナで、それぞれ「低」または「高」を選択します。

監視データがアプリケーションに対して表示されない場合、おそらくそのアプリケーションが存在しないか、または機能していません。アプリケーションの監視データは、アプリケーションが存在し、監視がそのアプリケーションで有効で、なおかつアプリケーションが機能している場合だけ使用できます。いったんアプリケーションが実行されると、監視レジストリに登録され、監視データが表示されます。

管理コンソールを使用すると、リモートのアプリケーションとインスタンスを監視できます。ただしそれには、リモートのインスタンスが実行されており、かつその設定がなされている必要があります。

選択したコンポーネントの監視データが選択したコンポーネントの下に表示されます。監視可能なプロパティーについては、281 ページの「監視対象のコンポーネントとサービスの統計について」を参照してください。アプリケーションに対するコンポーネントやサービスの構成を示した図については、278 ページの「監視可能なオブジェクトのツリー構造について」を参照してください。

5 「リソース」ページを選択して、サーバーインスタンスに配備され監視が有効なリソースの監視データを表示します。「ビュー」リストからリソースを選択します。

監視データを表示させたいリソースが見つからない場合は、「監視を設定」リンクを選択し、リソースの監視を有効または無効にします。

監視データがリソースに対して表示されない場合、おそらくそのリソースが存在しないか、または機能していません。リソースの監視データは、リソースが存在し、監視がそのリソースに「高」レベルで有効で、なおかつリソースが機能している場合だけ使用できます。たとえば、JDBC コネクタサービスが作成されていても、そのコネクタサービスを使用するアプリケーションがまだサービスからコネクタを要求していない場合、そのサービスはまだ作成されていないのでサービスは存在せず監視データも使用できません。いったん JDBC アプリケーションが実行され、サービスからコネクタを要求すると、そのサービスは監視レジストリに登録され監視データが表示されます。

選択したコンポーネントまたはサービスの監視データが「ビュー」フィールドの下に表示されます。監視可能なプロパティーについては、281 ページの「監視対象のコンポーネントとサービスの統計について」を参照してください。リソースに対するコンポーネントやサービスの構成を示した図については、278 ページの「監視可能なオブジェクトのツリー構造について」を参照してください。

6 「トランザクション」ページを選択して、トランザクションのロールバックおよびフリーズ時に処理中だったトランザクションの特定を行うためにトランザクションサブシステムをフリーズします。

7 トランザクションサービスの監視を有効にするには、「監視の設定」リンクを選択し、トランザクションサービスが「低」に設定されていることを確認します。

トランザクションをロールバックするためにトランザクションサービスをフリーズするには、「フリーズ」を選択します。トランザクションをロールバックするには、トランザクションのそばのチェックボックスを選択して「ロールバック」をクリックします。

参考 同機能を持つ asadmin コマンド

```
get --monitor
```

たとえば、JVM の監視データを表示するには、次の asadmin コマンドを使用します。

```
asadmin> get --user adminuser --monitor server.jvm.*
```

asadmin ツールによる監視データの表示

▼ asadmin ツールを使用して監視データを表示する

asadmin ツールを使用して監視データを表示するには、asadmin list および asadmin get コマンドに続けて監視可能なオブジェクトのドット表記名を使用します。手順を次に示します。

1 監視可能なオブジェクトの名前を表示するには、asadmin list コマンドを使用します。

たとえば、サーバーインスタンスで監視が有効なアプリケーションコンポーネントおよびサブシステムのリストを表示するには、次のコマンドを端末ウィンドウに入力します。

```
asadmin> list --user adminuser --monitor server
```

前述のコマンドは、監視が有効なアプリケーションコンポーネントおよびサブシステムの次のようなリストを返します。

```
server.resources
server.connector-service
server.orb
server.jms-service
server.jvm
server.applications
```

```
server.http-service
server.thread-pools
```

list コマンドのその他の使用例については、312 ページの「list コマンドと get コマンドの例」を参照してください。list コマンドで使用できるドット表記名の詳細については、310 ページの「ドット表記名とその指定方法について」を参照してください。

2 監視が有効なアプリケーションコンポーネントまたはサブシステムの監視統計を表示するには、asadmin get コマンドを使用します。

統計を取得するには、asadmin get コマンドを端末ウィンドウに入力して、前述の手順の list コマンドで表示された名前を指定します。次の例では、特定のオブジェクトのサブシステムからすべての属性を取得します。

```
asadmin> get --user adminuser --monitor server.jvm.*
```

このコマンドは次の属性およびデータを返します。

```
server.jvm.dotted-name = server.jvm
server.jvm.heapsize-current = 21241856
server.jvm.heapsize-description = Provides statistical information about
   the JVM’s memory heap size.
server.jvm.heapsize-highwatermark = 21241856
server.jvm.heapsize-lastsampletime = 1080232913938
server.jvm.heapsize-lowerbound = 0
server.jvm.heapsize-lowwatermark = 0
server.jvm.heapsize-name = JvmHeapSize
server.jvm.heapsize-starttime = 1080234457308
server.jvm.heapsize-unit = bytes
server.jvm.heapsize-upperbound = 518979584
server.jvm.uptime-count = 1080234457308
server.jvm.uptime-description = Provides the amount of time the JVM has
   been running.
server.jvm.uptime-lastsampletime = 1080234457308
server.jvm.uptime-name = JvmUpTime
server.jvm.uptime-starttime = 1080232913928
server.jvm.uptime-unit = milliseconds
```

get コマンドのその他の使用例については、312 ページの「list コマンドと get コマンドの例」を参照してください。get コマンドで使用できるドット表記名の詳細については、310 ページの「ドット表記名とその指定方法について」を参照してください。

ドット表記名とその指定方法について

asadmin list コマンドと asadmin get コマンドでは、監視可能オブジェクトのドット表記名を指定します。すべての子オブジェクトのアドレス指定にはドット (.) 文字が区切り文字として使用され、それらの名前は「ドット表記名」と呼ばれます。子ノードが単独タイプの場合、その監視オブジェクトタイプを指定するだけで、そのオブジェクトを指定できます。それ以外の場合は、type.name 形式の名前を指定する必要があります。

たとえば、http-service は、有効な監視可能オブジェクトタイプの 1 つであり、単独タイプです。インスタンス server の http-service を表す単独タイプの子ノードを指定する場合、ドット表記名は次のようになります。

```
server.http-service
```

もう 1 つ例を挙げます。applications は、有効な監視可能オブジェクトタイプですが、単独タイプではありません。たとえば、アプリケーション PetStore を表す、単独タイプでない子ノードを指定するには、ドット表記名は次のようになります。

```
server.applications.petstore
```

また、監視可能なオブジェクトの特定の属性も、ドット表記名で指定します。たとえば、http-service には、bytesreceived-lastsampletime という名前の監視可能な属性があります。次の名前は、bytesreceived 属性を指定していることになります。

```
server.http-service.server.http-listener-1.
  bytesreceived-lastsampletime
```

管理者は、asadmin list コマンドと asadmin get コマンドの有効なドット表記名を覚えておく必要はありません。list コマンドを使えば、利用可能な監視可能オブジェクトが表示され、ワイルドカードパラメータ付きの get コマンドを使えば、任意 の監視可能オブジェクトで利用可能なすべての属性を確認することができます。

list コマンドと get コマンドでドット表記名を使用する場合、根本的に次のことを前提としています。

- list コマンドでドット表記名の後にワイルドカード (*) が指定されていなかった場合、現在のノードの直接の子ノードが結果として返される。たとえば、list --user *adminuser* --monitor server を実行した場合、server ノードに属するすべての直接の子ノードが一覧表示される。
- list コマンドでドット表記名の後に .* 形式のワイルドカードが指定されていた場合、現在のノードの子ノード階層ツリーが結果として返される。たとえば、list --user *adminuser* --monitor server.applications.* を実行した場合、applications のすべての子ノードに加え、それらの配下にある子ノードなども一覧表示される。
- list コマンドで **dottedname*、*dotted* name*、**dotted name** のいずれかの形式でドット表記名の前後にのワイルドカードが指定されていた場合、そのマッチングパターンによって生成された正規表現にマッチするすべてのノードとそれらの子ノードが、結果として返される。
- get コマンドの末尾に「.*」、「*」のいずれかが指定されていた場合、マッチング対象の現在のノードに属する属性とその値のセットが、結果として返される。

詳細については、317 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

list コマンドと get コマンドの例

この節では、次の項目について説明します。

- 312 ページの「list --user admin-user --monitor コマンドの例」
- 313 ページの「get --user admin-user --monitor コマンドの例」

list --user admin-user --monitor コマンドの例

list コマンドは、指定されたサーバーインスタンス名で現在監視されているアプリケーションコンポーネントやサブシステムに関する情報を提供します。このコマンドを使えば、特定のサーバーインスタンスの監視可能なコンポーネントやそのサブコンポーネントを表示できます。list のより詳しい例については、317 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

例 1

```
asadmin> list --user admin-user --monitor server
```

前述のコマンドは、監視が有効なアプリケーションコンポーネントおよびサブシステムの次のようなリストを返します。

```
server.resources
server.orb
server.jvm
server.jms-service
server.connector-service
server.applications
server.http-service
server.thread-pools
```

また、指定されたサーバーインスタンス内で現在監視されているアプリケーションを一覧表示することも可能です。これは、get コマンドを使って特定のアプリケーションの特定の監視統計を取得する場合に便利です。

例 2

```
asadmin> list --user admin-user --monitor server.applications
```

次の結果が返されます。

```
server.applications.adminapp
 server.applications.admingui
server.applications.myApp
```

より包括的な例については、314 ページの「PetStore サンプルを使用する」を参照してください。

get --user admin-user --monitor コマンドの例

このコマンドは、次の監視対象情報を取得します。

- 特定のコンポーネントまたはサブシステム内で監視されているすべての属性
- 特定のコンポーネントまたはサブシステム内で監視されている特定の属性

  特定のコンポーネントまたはサブシステムに存在しない属性が要求された場合、エラーが返されます。同様に、コンポーネントまたはサブシステムのアクティブでない特定の属性が要求された場合も、エラーが返されます。

  get コマンドの使用方法の詳細については、317 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

例 1

特定のオブジェクトのすべての属性をサブシステムから取得します。

```
asadmin> get --user admin-user --monitor server.jvm.*
```

次の結果が返されます。

```
server.jvm.dotted-name= server.jvm
server.jvm.heapsize-current = 21241856
server.jvm.heapsize-description = Provides statistical information about
   the JVM's memory heap size.
server.jvm.heapsize-highwatermark = 21241856
server.jvm.heapsize-lastsampletime = 1080232913938
server.jvm.heapsize-lowerbound = 0
server.jvm.heapsize-lowwatermark = 0
server.jvm.heapsize-name = JvmHeapSize
server.jvm.heapsize-starttime = 1080234457308
server.jvm.heapsize-unit = bytes
server.jvm.heapsize-upperbound = 518979584
server.jvm.uptime-count = 1080234457308
server.jvm.uptime-description = Provides the amount of time the JVM has
   been running.
server.jvm.uptime-lastsampletime = 1080234457308
server.jvm.uptime-name = JvmUpTime
server.jvm.uptime-starttime = 1080232913928
server.jvm.uptime-unit = milliseconds
```

例 2

特定の J2EE アプリケーションからすべての属性を取得します。

```
asadmin> get --user admin-user --monitor server.applications.myJ2eeApp.*
```

次の結果が返されます。

```
No matches resulted from the wildcard expression.
CLI137 Command get failed.
```

J2EE アプリケーションレベルで公開されている監視可能な属性が存在しないため、このような応答が表示されました。

例 3

特定のサブシステムから特定の属性を取得します。

```
asadmin> get --user admin-user --monitor server.jvm.uptime-lastsampletime
```

次の結果が返されます。

```
server.jvm.uptime-lastsampletime = 1093215374813
```

例 4

特定のサブシステム属性内から未知の属性を取得します。

```
asadmin> get --user admin-user --monitor server.jvm.badname
```

次の結果が返されます。

```
No such attribute found from reflecting the corresponding Stats
interface: [badname]
CLI137 Command get failed.
```

▼ PetStore サンプルを使用する

次の例は、asadmin ツールを監視目的でどのように使えばよいかを示したものです。

あるユーザーが、Application Server 上に配備済みのサンプル Petstore アプリケーションに含まれる特定のメソッドの呼び出し回数を調査しようとしています。その配備先インスタンスの名前は、server です。list コマンドと get コマンドを併用することで、そのメソッドの目的の統計情報にアクセスします。

1. **Application Server** と asadmin ツールを起動します。
2. いくつかの有用な環境変数を設定することで、それらの値をコマンドごとに入力しないですむようにします。

   ```
   asadmin> export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=admin123
   asadmin>export AS_ADMIN_HOST=localhost AS_ADMIN_PORT=4849
   ```

3. インスタンス server の監視可能なコンポーネントを一覧表示します。

   ```
   asadmin> list --user adminuser --monitor server*
   ```

次のような出力結果が返されます。

```
server
server.applications
server.applications.CometEJB
server.applications.ConverterApp
server.applications.petstore
server.http-service
server.resources
server.thread-pools
```

この監視可能なコンポーネントの一覧には、thread-pools、http-service、resources、および配備済みで有効化されているすべての applicationsが含まれています。

4 PetStore アプリケーションの監視可能なサブコンポーネントを一覧表示します (--monitor の代わりに -m を使用可能)。

```
asadmin> list -m server.applications.petstore
```

次の結果が返されます。

```
server.applications.petstore.signon-ejb_jar
server.applications.petstore.catalog-ejb_jar
server.applications.petstore.uidgen-ejb_jar
server.applications.petstore.customer-ejb_jar
server.applications.petstore.petstore-ejb_jar
server.applications.petstore.petstore\.war
server.applications.petstore.AsyncSenderJAR_jar
server.applications.petstore.cart-ejb_jar
```

5 Petstore アプリケーションの **EJB** モジュール signon-ejb_jar の監視可能なサブコンポーネントを一覧表示します。

```
asadmin> list -m server.applications.petstore.signon-ejb_jar
```

次の結果が返されます。

```
server.applications.petstore.signon-ejb_jar.SignOnEJB
server.applications.petstore.signon-ejb_jar.UserEJB
```

6 Petstore アプリケーションの **EJB** モジュール signon-ejb_jar のエンティティー **Bean** UserEJB に含まれる監視可能なサブコンポーネントを一覧表示します。

```
asadmin> list -m server.applications.petstore.signon-ejb_jar.UserEJB
```

次の結果が返されます (ドット表記名はスペースの関係で削除してある)。

```
server.applications.petstore.signon-ejb_jar.UserEJB.bean-cache
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods
server.applications.petstore.signon-ejb_jar.UserEJB.bean-pool
```

7 PetStore アプリケーションの **EJB** モジュール signon-ejb_jar のエンティティー **Bean** UserEJB に含まれるメソッド getUserName 内の監視可能なサブコンポーネントを一覧表示します。

```
asadmin> list -m
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.getUserName
```

次の結果が返されます。

```
Nothing to list at server.applications.petstore.signon-ejb_jar.
UserEJB.bean-methods.getUserName. To get the valid names beginning with a
string, use the wildcard "*" character. For example, to list all names
that begin with "server", use "list server*".
```

8 メソッドに対する監視可能なサブコンポーネントは存在しません。メソッド getUserName の監視可能なすべての統計を取得します。

```
asadmin> get -m
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.getUserName.*
```

次の結果が返されます。

```
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-description = Provides the time in milliseconds
   spent during the last successful/unsuccessful attempt to execute the
   operation.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-lastsampletime = 1079981809259
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-name = ExecutionTime
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-unit = count
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-description = Provides the number of times an
   operation was called, the total time that was spent during the
   invocation and so on.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-lastsampletime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-maxtime = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-mintime = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-name = ExecutionTime
```

```
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-totaltime = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-unit =
 server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-description = Provides the total number of errors
   that occured during invocation or execution of an operation.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-lastsampletime = 1079981809273
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-name = TotalNumErrors
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-unit = count
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-description = Provides the total number of
   successful invocations of the method.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-lastsampletime = 1079981809255
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-name = TotalNumSuccess
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-unit = count
```

9 また、実行回数など、特定の統計を取得するには、次のようなコマンドを使用します。

```
asadmin> get -m server.applications.petstore.signon-ejb_jar.
UserEJB.bean-methods.getUserName.executiontime-count
```

次の結果が返されます。

```
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-count = 1
```

すべてのレベルにおける **list** コマンドと **get** コマンドの予想出力

次の各表は、ツリーの各レベルにおけるコマンド、ドット表記名、および対応する出力を示したものです。

表 **16–33** トップレベル

コマンド	ドット表記名	出力
`list -m`	`server`	`server.applicationsserver.thread-poolsserver.resourcesserver.http-serviceserver.transaction-serviceserver.orb.connection-managersserver.orb.connection-managers.orb\.Connections\.Inbound\.AcceptedConnectionsserver.jvm`
`list -m`	`server.*`	このノードから下の子ノード階層。
`get -m`	`server.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。

次の表に、アプリケーションレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 **16–34** アプリケーションレベル

コマンド	ドット表記名	出力

`list -m`	`server.applications`
	または
	`*applications`
		`appl1app2web-module1_warejb-module2_jar...`

`list -m`	`server.applications.*`
	または
	`*applications.*`
		このノードから下の子ノード階層。

`get -m`	`server.applications.*`
	または
	`*applications.*`
		このノードに属性が存在しないことを示すメッセージだけが表示されます。

次の表に、アプリケーションレベルのスタンドアロンモジュールとエンタープライズアプリケーションのコマンド、ドット表記名、および対応する出力を示します。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール

コマンド	ドット表記名	出力
list -m	server.applications.app1 または *app1 注意: このレベルが適用可能なのは、エンタープライズアプリケーションが配備されている場合だけです。スタンドアロンモジュールが配備されている場合には適用できません。	ejb-module1_jarweb-module2_warejb-module3_jarweb-module3_war
list -m	server.applications.app1.* または *app1.*	このノードから下の子ノード階層。
get -m	server.applications.app1.* または *app1.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.applications.app1.ejb-module1_jar または *ejb-module1_jar または server.applications.ejb-module1_jar	bean1bean2bean3...
list -m	server.applications.app1.ejb-module1_jar または *ejb-module1_jar または server.applications.ejb-module1_jar	このノードから下の子ノード階層。
get -m	server.applications.app1.ejb-module1_jar.* または *ejb-module1_jar.* または server.applications.ejb-module1_jar.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール (続き)

コマンド	ドット表記名	出力
list -m	server.applications.app1.ejb-module1_jar.bean1 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	次の子ノード一覧が表示されます。 bean-poolbean-cachebean-method
list -m	server.applications.app1.ejb-module1_jar.bean1 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	子ノードの階層とこのノードとそれより下のすべての子ノードの全属性の一覧。
get -m	server.applications.app1.ejb-module1_jar.bean1 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	次の属性とそれらの関連付けられた値が表示されます。 CreateCount_CountCreateCount_ DescriptionCreateCount_ LastSampleTimeCreateCount_ NameCreateCount_ StartTimeCreateCount_ UnitMethodReadyCount_ CurrentMethodReadyCount_ DescriptionMethodReadyCount_ HighWaterMarkMethodReadyCount_ LastSampleTimeMethodReadyCount_ LowWaterMarkMethodReadyCount_ NameMethodReadyCount_ StartTimeMethodReadyCount_ UnitRemoveCount_CountRemoveCount_ DescriptionRemoveCount_ LastSampleTimeRemoveCount_ NameRemoveCount_StartTimeAttribute RemoveCount_Unit
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-pool 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	属性は表示されず、 server.applications.appl.ejb-module1_jar.bean1-cache には表示すべき情報がないことを示すメッセージが表示されます。特定の文字列で始まる有効な名前を取得するには、ワイルドカード (*) 文字を使用します。たとえば、server で始まるすべての名前を一覧表示するには、list server* を使用します。
get -m	server.applications.app1.ejb-module1_jar.bean1.bean-pool 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	表 1–1 で説明した EJB プール属性に対応する属性と値の一覧。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール　(続き)

コマンド	ドット表記名	出力
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-cache	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。
get -m	server.applications.app1.ejb-module1_jar.bean1.bean-cache.*	表 1–5 で説明した EJB キャッシュ属性に対応する属性と値の一覧。 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-method.method1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。
get -m	server.applications.app1.ejb-module1_jar.bean1.bean-method.method1.*	表 1–2 で説明した EJB メソッド属性に対応する属性と値の一覧。 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。
list -m	server.applications.app1.web-module1_war	このモジュールに割り当てられた 1 つまたは複数の仮想サーバーが表示されます。
get -m	server.applications.app1.web-module1_war.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.applications.app1.web-module1_war.virtual-server	登録されているサーブレットの一覧が表示されます。
get -m	server.applications.app1.web-module1_war.virtual-server.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.applications.app1.web-module1_war.virtual-server.servlet1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.applications.app1.web-module1_war.virtual-server.servlet1.*	表 1–3 で説明した Web コンテナ (サーブレット) 属性に対応する属性と値の一覧。

次の表に、HTTP サービスレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–36 HTTP サービスレベル

コマンド	ドット表記名	出力
`list -m`	`server.http-service`	仮想サーバーの一覧。
`get -m`	`server.http-service.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。
`list -m`	`server.http-service.server`	HTTP リスナーの一覧。
`get -m`	`server.http-service.server.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。
`list -m`	`server.http-service.server.http-listener1`	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
`get -m`	`server.http-service.server.*`	表 1-9 で説明した HTTP サービス属性に対応する属性と値の一覧。

次の表に、スレッドプールレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–37 スレッドプールレベル

コマンド	ドット表記名	出力
`list -m`	`server.thread-pools`	スレッドプール名の一覧。
`get -m`	`server.thread-pools.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。
`list -m`	`server.thread-pools.orb\.threadpool\.thread-pool-1`	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
`get -m`	`server.thread-pools..orb\.threadpool\.thread-pool-1.*`	表 1-11 で説明したスレッドプール属性に対応する属性と値の一覧。

次の表に、リソースレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–38 リソースレベル

コマンド	ドット表記名	出力
list -m	server.resources	プール名の一覧。
get -m	server.resources.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.resources.jdbc-connection-pool-pool.connection-pool1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.resources.jdbc-connection-pool-pool.connection-pool1.*	表 1-10 で説明した接続プール属性に対応する属性と値の一覧。

次の表に、トランザクションサービスレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–39 トランザクションサービスレベル

コマンド	ドット表記名	出力
list -m	server.transaction-service	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.transaction-service.*	表 1-15 で説明したトランザクションサービス属性に対応する属性と値の一覧。

次の表に、ORB レベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–40 ORB レベル

コマンド	ドット表記名	出力
list -m	server.orb	server-orb.connection-managers
get -m	server.orb.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。

表 16–40 ORB レベル　(続き)

コマンド	ドット表記名	出力
list -m	server.orb.connection-managers	1 つまたは複数の ORB 接続マネージャー名。
get -m	server.orb.connection-managers.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.orb.connection-managers.orb\.ConnectionManager\.Inbound\.AcceptedConnections	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.orb.connection-managers.orb\.ConnectionManager\.Inbound\.AcceptedConnections\.*	表 1-3 で説明した ORB 接続マネージャー属性に対応する属性と値の一覧。

次の表に、JVM レベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–41 JVM レベル

コマンド	ドット表記名	出力
list -m	server.jvm	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.jvm.*	表 1-16 で説明した JVM 属性に対応する属性と値の一覧。

JConsole の使用

JConsole と Application Server を連携動作させるには、JMX コネクタのセキュリティーを無効にする必要があります。Application Server (SE/EE 版) の現行バージョンでは、セキュリティーがデフォルトで有効になっています。

▼ JMX コネクタのセキュリティーを無効にする

1 管理コンソールを使って **JMX** コネクタのセキュリティーを無効にします。管理コンソールからこれを行うには、次の手順に従います。

a. 「ノードエージェント」を展開します。

b. 「ノードエージェント」を選択します。

c. 「**JMX**」タブで、**SSL3** と **TLS** の選択を解除します。

d. 「保存」を選択します。

2 `asadmin` を使って **JMX** コネクタのセキュリティーを無効にします。端末ウィンドウまたはコマンドプロンプトからこれを行うには、次の手順に従います。

a. 次のコマンドを入力します。

```
asadmin set server.admin-service.jmx-connector.system.security-enabled=false
```

b. **DAS (Domain Application Server)** を再起動します。

Platform Edition 版では、JMX コネクタはデフォルトで無効になっています。このため、Platform Edition では設定を変更する必要はありません。

3 **JConsole** を起動し、「詳細」タブで **JMX URL**、ユーザー名、およびパスワードを入力してログインします。**JMX URL** の形式は次のとおりです。

```
service:jmx:rmi:///jndi/rmi://<your machine
name>:<port>/management/rmi-jmx-connector
```

注意: `server.log` 管理ファイルから正確な JMX URL を取得できます。それには、このファイル内で「`message ADM1501`」を検索してください。

◆ ◆ ◆ 第 17 章

Java 仮想マシンと詳細設定

この章では、Java 仮想マシン (JVM™) とその他の詳細設定の設定方法について説明します。この章には次の節が含まれています。

JVM 設定に関する管理コンソールタスク

▼ JVM の一般設定を行う

Application Server で必要とされる Java 仮想マシン (JVM) は、J2SE (Java 2 Standard Edition) ソフトウェアに含まれます。JVM の設定に誤りがあるとサーバーが稼動しなくなるため、この設定を変更するときは注意してください。

1. ツリーコンポーネントで、「設定」を選択します。
2. 対象となるサーバーインスタンスの「**JVM 設定**」ノードをクリックします。
3. デフォルトでは、タブの下で「一般」リンクがすでに選択されています。

4 「**JVM** 一般設定」ページで次の操作を行います。

a. 「**Java** ホーム」フィールドに、**J2SE (Java 2 Standard Edition)** ソフトウェアのインストールディレクトリの名前を入力します。

Application Server は J2SE ソフトウェアに依存します。指定した J2SE のバージョンがこのリリースでサポートされるかどうかについては、『リリースノート』を参照してください。58 ページの「詳細情報」セクションのリンクを参照してください。

注 – 存在しないディレクトリ名を入力したり、サポートされないバージョンの J2SE ソフトウェアのインストールディレクトリを指定したりした場合、Application Server は起動しません。

---

b. 「**Javac** オプション」フィールドに、**Java** プログラミング言語コンパイラのコマンド行オプションを入力します。

Application Server は EJB コンポーネントの配備時にコンパイラを実行します。

c. **JPDA (Java Platform Debugger Architecture)** によるデバッグを設定するときは、「デバッグ」フィールドの「有効」チェックボックスにチェックマークを付け、「デバッグオプション」フィールドにオプションを指定します。

JPDA はアプリケーション開発者によって使用されます。詳細については、『Application Server 開発者ガイド』の「J2EE アプリケーションのデバッグ」の章を参照してください。このガイドへのリンクについては、58 ページの「詳細情報」を参照してください。

d. 「**RMI** コンパイルオプション」フィールドに、`rmic` コンパイラのコマンド行オプションを入力します。

EJB コンポーネントの配備時に Application Server は `rmic` コンパイラを実行します。

e. 「バイトコードプリプロセッサ」フィールドに、クラス名のコンマ区切りリストを入力します。

各クラスは、`com.sun.appserv.BytecodePreprocessor` インタフェースを実装する必要があります。クラスは指定の順序で呼び出されます。

プロファイラなどのツールは、「バイトコードプリプロセッサ」フィールドの入力を必要とすることがあります。プロファイラは、サーバーパフォーマンスの分析に使用される情報を生成します。プロファイリングの詳細については、Application Server 開発者ガイドの「J2EE アプリケーションのデバッグ」の章を参照してください。

5 「保存」をクリックします。

6 サーバーを再起動します。

▼ JVM のクラスパス設定を行う

クラスパスは、Java 実行時環境がクラスやその他のリソースファイルを検索する JAR ファイルのリストです。

1 ツリーコンポーネントで、「**Application Server**」ノードを選択します。

2 対象となるサーバーインスタンスの「**JVM 設定**」ノードをクリックします。

3 タブの下にある「パス設定」リンクを選択します。

4 「**JVM** クラスパス設定」ページで次の操作を行います。

a. 「環境クラスパス」チェックボックスのデフォルトの選択内容を維持し、**CLASSPATH** 環境変数を無視します。

CLASSPATH 環境変数は、プログラミングの基礎練習では便利ですが、エンタープライズ環境での使用はお勧めできません。

b. **Application Server** のクラスパスを確認するには、「サーバークラスパス」フィールドの読み取り専用の内容を調べます。

c. サーバーのクラスパスの先頭に **JAR** ファイルを挿入するには、「クラスパスのプレフィックス」フィールドにファイルの完全パス名を入力します。

d. サーバーのクラスパスの最後に **JAR** ファイルを追加するには、「クラスパスのサフィックス」フィールドにファイルの完全パス名を入力します。

たとえば、データベースドライバの JAR ファイルを指定します。87 ページの「JDBC ドライバを統合する」を参照してください。

e. 「ネイティブライブラリパスのプレフィックス」および「ネイティブライブラリパスのサフィックス」の各フィールドには、ネイティブライブラリパスの先頭、または最後に追加するエントリを指定できます。

ネイティブライブラリパスは、ネイティブ共有ライブラリの相対パス、標準の JRE ネイティブライブラリパス、シェル環境設定 (UNIX では LD_LIBRARY_PATH)、および「JVM プロファイラ設定」ページに指定したパスを連結したものです。

5 「保存」をクリックします。

6 サーバーを再起動します。

▼ JVM オプションを設定する

「JVM オプション」ページでは、Application Server を実行する Java アプリケーション起動用ウィンドウ (java ツール) のオプションを指定できます。-D オプションは、Application Server に固有のプロパティーを指定します。

1. ツリーコンポーネントで、「**Application Server**」ノードを選択します。
2. 対象となるサーバーインスタンスの「**JVM 設定**」ノードをクリックします。
3. タブの下にある「**JVM オプション**」リンクを選択します。
4. 「**JVM オプション**」ページでオプションを変更するには、「値」フィールドを編集します。
5. オプションを追加するには、次の手順を実行します。
   a. 「**JVM オプションを追加**」をクリックします。
   b. 表示される空白行に、「値」フィールドの情報を入力します。
6. オプションを削除するには、次の手順を実行します。
   a. オプションの隣のボックスにチェックマークを付けます。
   b. 「削除」をクリックします。
7. 「保存」をクリックします。
8. サーバーを再起動します。

JVM オプションの詳細については、`http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html` および `http://java.sun.com/docs/hotspot/VMOptions.html` を参照してください。

▼ セキュリティーマネージャーを無効にする

Application Server のセキュリティーマネージャーを無効にすると、アプリケーションのタイプによってはパフォーマンスが向上する場合があります。J2EE の承認および認証機能は、セキュリティーマネージャーが無効になっている場合でも機能します。開発環境ではセキュリティーマネージャーを無効にできますが、実稼動環境ではセキュリティーマネージャーを無効にしないでください。

1. 管理コンソールの「**JVM オプション**」ページに進みます。
   手順については、330 ページの「JVM オプションを設定する」を参照してください。

2 「JVM オプション」ページで、このオプションを次のように削除します。

```
-Djava.security.policy
```

3 「保存」をクリックします。

4 サーバーを再起動します。

▼ JVM のプロファイラ設定を行う

プロファイラツールは、パフォーマンスを分析し、潜在的なボトルネックを特定するための情報を生成します。

1 ツリーコンポーネントで、「Application Server」ノードを選択します。

2 対象となるサーバーインスタンスの「JVM 設定」ノードをクリックします。

3 タブの下にある「プロファイラ」リンクを選択します。

4 「JVM プロファイラ設定」ページに指定する情報は、使用するプロファイラ製品によって異なります。
例と操作方法については、『Application Server 開発者ガイド』の「J2EE アプリケーションのデバッグ」の章を参照してください。このガイドへのリンクについては、58 ページの「詳細情報」を参照してください。

5 「保存」をクリックします。

6 サーバーを再起動します。

詳細設定に関する管理コンソールタスク

▼ 詳細ドメイン属性を設定する

1 ツリーコンポーネントで、「スタンドアロンインスタンス」を展開し、サーバインスタンスノードを選択します。

2 「詳細」タブを選択します。

3 「ドメイン属性」ページでは、次の設定を行えます。

a. 「アプリケーションルート」フィールドで、アプリケーションを配備するディレクトリの完全パスを指定します。

b. 「ログルート」フィールドで、サーバーインスタンスのログファイルを保持する場所を指定します。

c. 通常は、ホストのデフォルトロケールを使用するため、「ロケール」フィールドは空白のまま残します。

ロケールは、言語と地域の特定の組み合わせを指定する識別子です。たとえば、米国英語のロケールは `en_US` で、日本語のロケールは `ja_JP` です。英語以外のロケールを使用するには、Application Server を英語から別の言語に言語対応する必要があります。

4 「保存」をクリックします。

5 サーバーを再起動します。

◆ ◆ ◆ 第 18 章

Apache Web サーバーのコンパイルと設定

この付録では、Apache ソースコードのコンパイルと Apache Web サーバーインストールの設定をどのように行えば、Sun Java System Application Server ロードバランサプラグインを使用できるようになるかについて説明します。

適切な Apache ソースコードをダウンロードしてください。Application Server でサポートされる Apache Web サーバーのバージョンとプラットフォームの詳細については、Application Server リリースノートを参照してください。

この付録では、次の項目について説明します。

Apache のインストール

この節では、次の項目について説明します。

Apache 1.3 の最小要件

この節では、Apache 1.3 Web サーバーを正常にコンパイルしてロードバランサプラグインを実行するための最小要件を説明します。Apache ソースは、SSL と連携して動作するようにコンパイルし、ビルドする必要があります。

UNIX および Linux プラットフォームの要件は次のとおりです。

- `openssl-0.9.7d` (ソース)
- `mod_ssl-2.8.16-1.3.29` (ソース)
- `apache_1.3.29` (ソース)
- `gcc-3.3-sol9-sparc-local` パッケージ (Solaris 9 SPARC/x86 の場合)

- `flex-2.5.4a-sol9-sparc-local` パッケージ (Solaris 9 SPARC の場合)
- `flex-2.5.4a-sol9-intel-local` パッケージ (Solaris 9 x86 の場合)

さらに、Apache をコンパイルする前に、次の操作を実行する必要があります。

- Linux では、同じマシンに Sun Java System Application Server をインストールします。
- Solaris 8 では、`PATH` に `gcc` と `make` が存在することを確認します。
- Solaris 9 では、`PATH` に `gcc` バージョン 3.3 と `make` が存在し、`flex` がインストールされていることを確認します。
- Red Hat Enterprise Linux Advanced Server 2.1 上で `gcc` を使用する場合は、バージョンが `gcc` 3.0 以降である必要があります

注 –

- その他の C 言語コンパイラを使用するには、PATH 環境変数にその C 言語コンパイラと make ユーティリティーのパスを設定します。次に例を示します。

  export `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:`*install-dir*`/lib`
- これらのソフトウェアのソースは、`http://www.sunfreeware.com` で入手できます。

Apache 2 の最小要件

この節では、Apache 2 Web サーバーを正常にコンパイルしてロードバランサプラグインを実行するための最小要件を説明します。Apache ソースは、SSL と連携して動作するようにコンパイルし、ビルドする必要があります。

UNIX および Linux プラットフォームの要件は次のとおりです。

- `openssl-0.9.7e` (ソース)
- `httpd-2.0.49` (ソース)
- `gcc-3.3-sol9-sparc-local` パッケージ (Solaris SPARC の場合)
- `gcc-3.3-sol9-intel-local` パッケージ (Solaris x86 の場合)
- `flex-2.5.4a-sol9-sparc-local` パッケージ (Solaris SPARC の場合)
- `flex-2.5.4a-sol9-intel-local` パッケージ (Solaris x86 の場合)

さらに、Apache をコンパイルする前に、次の操作を実行する必要があります。

- Linux プラットフォームでは、同じマシンに Sun Java System Application Server をインストールします。
- Solaris オペレーティングシステムでは、`PATH` に `gcc` バージョン 3.3 と `make` が存在し、`flex` がインストールされていることを確認します。

- Solaris 10 オペレーティングシステムでは、OpenSSL の make を実行する前に、/usr/local/lib/gcc-lib/sparc-sun-solaris2.9/3.3/install-tools/mkheaders (Solaris SPARC) または /usr/local/lib/gcc-lib/i386-pc-solaris2.9/3.3/install-tools/mkheaders (Solaris x86) を実行します。
- Red Hat Enterprise Linux Advanced Server 2.1 上で gcc を使用する場合は、バージョンが gcc 3.0 以降である必要があります

注 –

- その他の C 言語コンパイラを使用するには、PATH 環境変数にその C 言語コンパイラと make ユーティリティーのパスを設定します。次に例を示します。export LD_LIBRARY_PATH= *install-dir*/lib:$LD_LIBRARY_PATH. この例は sh 用です。
- これらのソフトウェアのソースは、http://www.sunfreeware.com で入手できます。

▼ SSL 対応の Apache をインストールする

始める前に　Apache ソフトウェアがすでにダウンロードされ、圧縮解凍されている必要があります。

1 **OpenSSL** をコンパイルし、ビルドします。**OpenSSL** の詳細については、http://www.openssl.org/ を参照してください。

Linux とともにインストールされた OpenSSL のバージョンが 0.9.7e である場合は、Linux プラットフォーム上で次の手順を実行する必要はありません。

OpenSSL ソースをダウンロードし、展開します。

a. cd openssl-0.9.7e

b. make

c. make install

2 **Apache 1.3** の場合は、mod_ssl を使用して **Apache** を設定します。**Apache 2** の場合は、次の手順を実行する必要はありません。mod_ssl の詳細については、http://www.modssl.org/ を参照してください。

mod_ssl ソースを展開し、次の手順に従います。

a. cd mod_ssl-2.8.14-1.3.x

b. `./configure --with-apache=../apache_1.3.`*x* `--with-ssl=../openssl-0.9.7e --prefix=`*install-dir* `--enable-module=ssl --enable-shared=ssl --enable-rule=SHARED_CORE --enable-module=so` を実行します。

上記のコマンド例で指定しているディレクトリは、変数です。*prefix* 引数は Apache のインストール先を示します。バージョン番号の *x* は実際のバージョンを表します。

3 **Apache 2.0** の場合は、次のようにしてソースツリーを設定します。

a. `http-2.0_`*x* のディレクトリに移動します。

b. `./configure --with-ssl=`*open-ssl-install-path* `--prefix=`*install-dir* `--enable-ssl --enable-so` を実行します。

上記のコマンド例で指定しているディレクトリは、変数です。*prefix* 引数は Apache のインストール先を示します。バージョン番号の *x* は実際のバージョンを表します。

4 **Linux 2.1** 上の **Apache** については、コンパイルの前に次の手順を実行します。

a. `src/MakeFile` を開き、自動生成セクションの終わりを見つけます。

b. 自動生成セクションに続く **4** 行のあとに、次の行を追加します。

```
LIBS+= -licuuc -licui18n -lnspr4 -lpthread -lxerces-c -lsupport -lnsprwrap
-lns-httpd40

LDFLAGS+= -L/install-dir/lib -L/opt/sun/private/lib
```

注 – `-L/opt/sun/private/lib` の部分は、Application Server を Java Enterprise System インストールの一部としてインストールした場合にのみ必要です。

次に例を示します。

```
## (End of automatically generated section)##

CFLAGS=$(OPTIM) $(CFLAGS1) $(EXTRA_CFLAGS)LIBS=$(EXTRA_LIBS)
$(LIBS1)INCLUDES=$(INCLUDES1) $(INCLUDES0) $(EXTRA_INCLUDES)LDFLAGS=$(LDFLAGS1)
$(EXTRA_LDFLAGS)"

LIBS+= -licuuc -licui18n -lnspr4 -lpthread -lxerces-c -lsupport -lnsprwrap
-lns-httpd40LDFLAGS+= -L/install-dir/lib -L/opt/sun/private/lib
```

c. 環境変数 **LD_LIBRARY_PATH** を作成します。変数の値は、*install-dir* `/lib` **(**すべてのインストール**)** および *install-dir*`/lib:opt/sun/private/lib` **(Java Enterprise System** インストールの一部としてインストールした **Application Server** のみ**)** とします。

5 使用しているバージョンのインストール手順に従って **Apache** をコンパイルします。完全なマニュアルは、`http://httpd.apache.org/` にあります。

一般的な手順は次のとおりです。

a. `make`

b. `make certificate` **(Apache 1.3 のみ)**

c. `make install`

`make certificate` コマンドは、セキュリティー保護されたパスワードを要求します。このパスワードはセキュリティー保護された Apache を起動するために必要なので、忘れないでください。

6 使用する環境に合わせて **Apache** を設定します。

Apache のインストールが完了したら、プラグインのインストール後に「Modifications to Apache Web Server」を参照してください。

◆ ◆ ◆ 第 19 章

ドメインまたはノードエージェントの自動再起動

マシンの再起動が必要になった場合など、ドメインまたはノードエージェントが予想外に停止される場合にそなえて、ドメインまたはノードエージェントが自動的に再起動されるようにシステムを設定することが可能です。

この付録では、次の項目について説明します。

UNIX プラットフォーム上での自動再起動

UNIX プラットフォーム上でドメインを再起動するには、`/etc/inittab` ファイルにテキストを 1 行追加します。

たとえば、`opt/SUNWappserver` ディレクトリにインストールされた Application Server の `domain1` を、`password.txt` という名前のパスワードファイルを使って再起動するには、次のテキストを追加します。

```
das:3:respawn:/opt/SUNWappserver/bin/asadmin start-domain --user admin
--passwordfile /opt/SUNWappserver/password.txt domain1
```

このテキストは 1 行で記述してください。先頭の 3 文字はこのプロセスに対する一意の指示子ですが、これは変更可能です。

ノードエージェントを再起動する場合の構文も、これと似ています。たとえば、`opt/SUNWappserver` ディレクトリにインストールされた Application Server の `agent1` を、`password.txt` という名前のパスワードファイルを使って再起動するには、次のテキストを追加します。

```
das:3:respawn:/opt/SUNWappserver/bin/asadmin start-node-agent --user admin
--passwordfile /opt/SUNWappserver/password.txt agent1
```

Microsoft Windows プラットフォーム上での自動再起動

Microsoft Windows 上で自動的に再起動するには、Windows サービスを作成します。Sun Java System Application Server に同梱されている実行可能ファイル `appservService.exe` と `appserverAgentService.exe` を、Microsoft が提供するサービス制御コマンド (`sc.exe`) と組み合わせて使用します。

`sc.exe` コマンドは Windows XP に同梱されており、`C:\windows\system32` ディレクトリか `C:\winnt\system32` ディレクトリのいずれかに格納されています。このマニュアルの執筆時点では、Windows 2000 の `sc.exe` が `http:// ftp://ftp.microsoft.com/reskit/win2000/sc.zip` からダウンロード可能となっています。`sc.exe` の使用方法の詳細については、http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndllpro/html/msdn_scmslite.asp を参照してください。

`appservService.exe` および `appservAgentService.exe` の使用方法は次のとおりです。

```
C:\winnt\system32\sc.exe create service-name binPath= \"fully-qualified-path-to-appservService.exe
\"fully-qualified-path-to-asadmin.bat start-command\"
\"fully-qualified-path-to-asadmin.bat stop-command\""
start= auto DisplayName= "display-name"
```

たとえば、パスワードファイル `C:\Sun\AppServer\password.txt` を使ってドメイン `domain1` を開始および停止するサービス `SunJavaSystemAppServer DOMAIN1` を作成するには、次のようにします。

```
C:\windows\system32\sc.exe create domain1 binPath=
"C:\Sun\AppServer\lib\appservService.exe \"C:\Sun\AppServer\bin\asadmin.bat
start-domain --user admin --passwordfile C:\Sun\AppServer\password.txt domain1\"
\"C:\Sun\AppServer\bin\asadmin.bat stop-domain domain1\"" start= auto
DisplayName= "SunJavaSystemAppServer DOMAIN1"
```

ノードエージェント `agent1` を開始および停止するサービスを作成するには、次のようにします。

```
C:\windows\system32\sc.exe create agent1 binPath=
"C:\Sun\AppServer\lib\appservAgentService.exe \"C:\Sun\AppServer\bin\asadmin.bat
start-node-agent --user admin --passwordfile C:\Sun\AppServer\password.txt agent1\"
\"C:\Sun\AppServer\bin\asadmin.bat stop-node-agent agent1\"" start= auto
DisplayName= "SJESAS_SE8.1 AGENT1"
```

注 – `binPath=` パラメータの一部として入力された開始コマンドと停止コマンドは、正しい構文で記述されている必要があります。確認するには、それらのコマンドをコマンドプロンプトから実行します。コマンドを実行してもドメインまたはノードエージェントが正常に開始または停止しない場合、そのサービスは正しく動作していません。

注 – asadmin の start/stop コマンドとサービスの開始/停止を混在させないでください。両者を混在させると、サーバーの状態の同期が取れなくなります。たとえば、サーバーのコンポーネントが実行されていないのに「コンポーネントが開始された」と表示されたりします。こうした状況を避けるには、サービス使用時には常に、sc.exe コマンドを使ってコンポーネントを開始および停止するようにしてください。

自動再起動時のセキュリティー

起動時に必要となるパスワードとマスターパスワードは、次のいずれかの方法を使って処理します。

- Microsoft Windows 上で、ユーザーにパスワードを尋ねるようにサービスを設定します。
  1. サービスコントロールパネルで、作成したサービスをダブルクリックします。
  2. 「プロパティー」ウィンドウの「ログオン」タブをクリックします。
  3. 「デスクトップとの対話をサービスに許可」にチェックマークを付け、必要なパスワードに対するプロンプトがコンポーネント起動時に表示されるようにします。

     ログインしてプロンプトを表示させ、入力時にエントリがエコーバックされないことを確認する必要があります。これがサービスオプションを使用する際のもっとも安全な方法ですが、この方法の場合、ユーザーが関与しないとサービスが利用可能になりません。

     デスクトップとの対話オプションを設定しなかった場合、サービスは「開始保留」状態のままになり、ハングアップしたように見えます。この状態から抜け出すには、このサービスのプロセスを終了してください。
- Windows または UNIX 上で、--savemasterpassword=true オプションを使ってドメインを作成し、管理パスワード格納用のパスワードファイルを作成します。コンポーネント起動時に、--passwordfile オプションを使ってパスワードが格納されたファイルを指定します。

  次に例を示します。
  1. ドメイン作成時にマスターパスワードを保存します。次の構文では、ユーザーは管理パスワードとマスターパスワードの入力を求められます。

     ```
     asadmin create-domain --adminport 4848 --adminuser admin
     --savemasterpassword=true --instanceport 8080 domain1
     ```
  2. Windows の場合は、サービスを作成します。その際、パスワードファイルを使って管理パスワードを提供します。

```
C:\windows\system32\sc.exe create domain1 binPath=
"C:\Sun\AppServer\lib\appservService.exe \"C:\Sun\AppServer\bin\asadmin.bat
```

```
start-domain --user admin --passwordfile C:\Sun\AppServer\password.txt domain1\"
\"C:\Sun\AppServer\bin\asadmin.bat stop-domain domain1\"" start= auto
DisplayName= "SJESAS_PE8.1 DOMAIN1"
```

パスワードファイル `password.txt` のパスは、`C:\Sun\AppServer\password.txt` です。このファイルには、パスワードが次の形式で格納されています。

```
AS_ADMIN_password=password
```

たとえば、パスワードが `adminadmin` の場合、次のようになります。

```
AS_ADMIN_password=adminadmin
```

3. UNIX の場合、`inittab` ファイルに追加する行の中で、`--passwordfile` オプションを使用します。

```
das:3:respawn:/opt/SUNWappserver/bin/asadmin start-domain --user admin
--passwordfile /opt/SUNWappserver/password.txt domain1
```

パスワードファイル `password.txt` のパスは、`/opt/SUNWappserver/password.txt` です。このファイルには、パスワードが次の形式で格納されています。

```
AS_ADMIN_password=password
```

たとえば、パスワードが `adminadmin` の場合、次のようになります。

```
AS_ADMIN_password=adminadmin
```

◆ ◆ ◆ 第 20 章

20 domain.xml のドット表記名属性

この付録では、Mbean とその属性を指定するために使用可能なドット表記名属性について説明します。domain.xml ファイル内のすべての要素は対応する MBean を持ちます。これらの名前は、「個々の名前をピリオドで区切る」という構文規則に従うため、「ドット表記名」と呼ばれます。

この付録では、次の項目について説明します。

トップレベル要素

domain.xml ファイル内のすべてのトップレベル要素で、次の条件が満たされている必要があります。

- サーバー、設定、クラスタ、またはノードエージェントの名前はそれぞれ一意である必要があります。
- サーバー、設定、クラスタ、またはノードエージェントに「domain」という名前を付けることはできません。
- サーバーインスタンスに「agent」という名前を付けることはできません。

次の表に、トップレベル要素と対応するドット表記名プレフィックスを示します。

要素名	ドット表記名プレフィックス
アプリケーション	domain.applications
resources	domain.resources
configurations	domain.configs

要素名	ドット表記名プレフィックス
servers	`domain.servers` この要素に含まれるすべてのサーバーは、*server-name* としてアクセス可能です。ここで、*server-name* は、server サブ要素の name 属性の値です。
クラスタ	`domain.clusters` この要素に含まれるすべてのクラスタは、*cluster-name* としてアクセス可能です。ここで、*cluster-name* は、cluster サブ要素の name 属性の値です。
node-agents	`domain.node-agents`
lb-configs	`domain.lb-configs`
system-property	`domain.system-property`

次の 2 つのレベルの別名が利用可能です。

1. 1 つ目のレベルの別名を使えば、プレフィックス `domain.servers` または `domain.clusters` を使わずにサーバーインスタンスまたはクラスタの属性にアクセスできます。したがって、たとえば、`server1` という形式のドット表記名は、ドット表記名 `domain.servers.server1` (`server1` は特定のサーバーインスタンス) にマッピングされます。
2. 2 つ目のレベルの別名を使えば、特定のクラスタまたはスタンドアロンサーバーインスタンス (ターゲット) の設定、アプリケーション、およびリソースを参照できます。

次の表に、サーバー名またはクラスタ名で始まるドット表記名と、それらの別名であるドメイン配下のトップレベル名を示します。

ドット表記名	別名	補足説明
target`.applications.*`	`domain.applications.*`	この別名の解決結果は、*target* のみによって参照されるアプリケーションになります。
target`.resources.*`	`domain.resources.*`	この別名の解決結果は、*target* によって参照されるすべての `jdbc-connection-pool`、`connector-connection-pool`、`resource-adapter-config`、およびその他のすべてのリソースになります。

次の表に、サーバー名またはクラスタ名で始まるドット表記名と、それらの別名である、そのサーバーまたはクラスタによって参照されている設定内のトップレベル名を示します。

ドット表記名	別名
target`.http-service`	*config-name*`.http-service`

ドット表記名	別名
target.iiop-service	*config-name*.iiop-service
target.admin-service	*config-name*.admin-service
target.web-container	*config-name*.web-container
target.ejb-container	*config-name*.ejb-container
target.mdb-container	*config-name*.mdb-container
target.jms-service	*config-name*.jms-service
target.log-service	*config-name*.log-service
target.security-service	*config-name*.security-service
target.transaction-service	*config-name*.transaction-service
target.monitoring-service	*config-name*.monitoring-service
target.java-config	*config-name*.java-config
target.availability-service	*config-name*.availability-service
target.thread-pools	*config-name*.thread-pools

別名を使用しない要素

クラスタ化されたインスタンスでは、別名を使用しないでください。クラスタ化されたインスタンスの特定のシステムプロパティーを取得する際のドット表記名属性は、*clustered-instance-name*.*system-property* ではなく、domain.servers.*clustered-instance-name*.*system-property* のように記述してください。

索引

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

あ

い

え