Sun Java™ System

Application Server Enterprise Edition 8.1 管理ガイド

2005Q1

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

Part No: 819-1551

目 次

はじめに

このマニュアルでは、Application Server の設定および管理方法について説明します。この前書きには次のトピックに関する情報が含まれています。

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。次の URL にアクセスしてください。http://docs.sun.com/app/docs/prod/entsys.05q1?l=ja#hic です。

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 1 書体の表記規則

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:`

表 1　書体の表記規則 (続き)

書体	意味	例
AaBbCc123 (イタリック)	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。	これらを *class* オプションと呼びます。 ファイルは、*install-dir*/bin ディレクトリに置かれています。

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 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。
-	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 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
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 : *domain_root_dir*/
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 3　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*

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 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:¥

関連マニュアル

http://docs.sun.com℠ 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

『JDBC Database Access With Java:A Tutorial and Annotated Reference (Java Series)』Graham Hamilton (著)、Rick Cattell (著)、Maydene Fisher (著)

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。

次の表は、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です。

表 5　　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』	Sun Java SystemApplication 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。
『Deployment Planning Guide』	システムニーズと企業の評価による、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の確認。アプリケーションサーバー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。

表 5　マニュアルセットの内容 (続き)

タイトル	説明
『Developer's Guide』	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、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、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、Application Server のサブシステムとコンポーネントの設定、管理、配備。
『High Availability Administration Guide』	Sun Java System Application Server の高可用性機能の設定と管理。
『Administration Reference』	Sun Java System Application Server 設定ファイル `domain.xml` の編集。
『Upgrade and Migration Guide』	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。
『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。
『Reference Manual』	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/app/docs/coll/MessageQueue_05q1_ja
- Directory Server のマニュアル
  http://docs.sun.com/app/docs/coll/DirectoryServer_05q1_ja
- Web Server のマニュアル
  http://docs.sun.com/app/docs/coll/WebServer_05q1_ja

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/
- プロフェッショナルサービス
  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/
- 開発者情報
  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 にアクセスしてください。

関連するサードパーティの 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 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 2005Q1 管理ガイド』で、Part No は 819-1551 です。

ご意見をお寄せください

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、企業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、セキュリティ、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、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。クラスタ内の各アプリケーションサーバーインスタンスは、それぞれ同じように設定され、同じアプリケーションが配備されています。

  水平的なスケーリングは、クラスタに Application Server インスタンスを追加して、その結果システムの容量を増やすことによって実現されます。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** - 高可用性によって、クラスタ内の Application Server インスタンスに対するフェイルオーバーが可能になります。1 つの Application Server インスタンスが停止すると、別の Application Server インスタンスが、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を引き継ぎます。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

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について説明します。

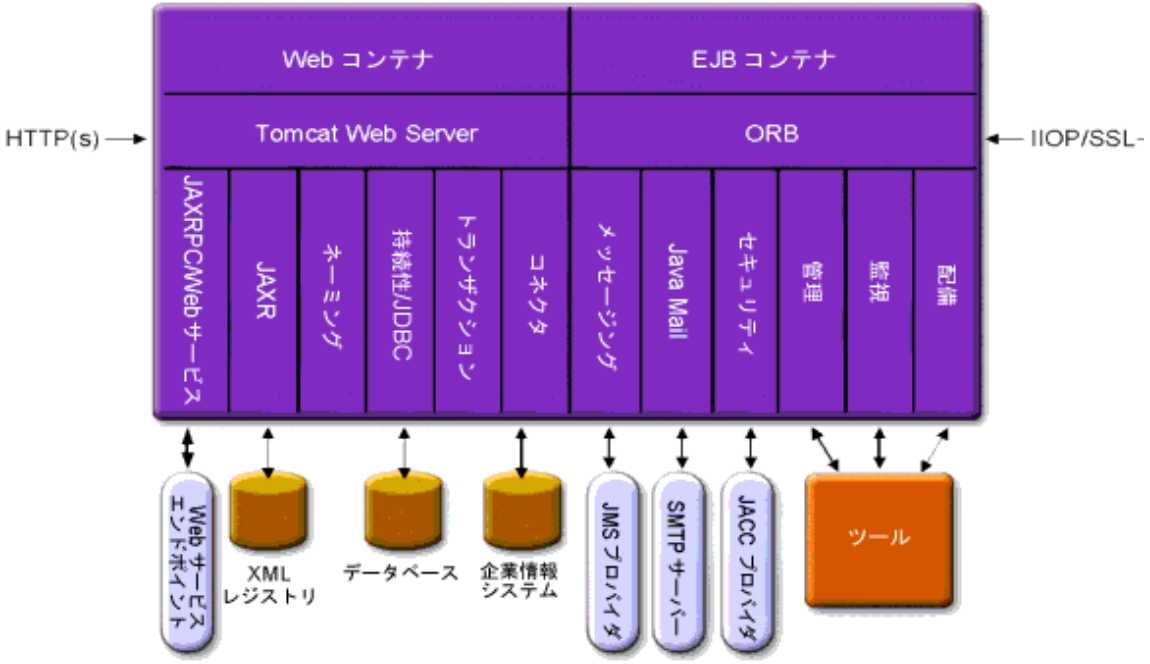


図 1-1　Application Server アーキテクチャ

- **コンテナ** - コンテナは、J2EE コンポーネントのセキュリティ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時環境です。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 プロトコルに個別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 に、次のサービスを示します。
  - ネーミング - ネーミング・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 - 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アプリケーションサーバー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後も存在するためです。Application Server には PointBase DBMS がバンドルされています。
- **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 Connector アーキテクチャでは、J2EE アプリケーションと既存の EIS (Enterprise Information System) 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。
  - 管理コンソール
  - asadmin ユーティリティ
  - 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、サーバーに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。

http://hostname:port

次に例を示します。

http://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「アプリケーションサーバー管理コンソール」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

asadmin ユーティリティは、コマンド行ツールです。asadmin ユーティリティと、このユーティリティ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リスト表示するには、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)

Sun Java System 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 (MBean) を監視しています。

Application Server Management Extension の使用方法の詳細については、『Sun Java System Application Server Developer's Guide』の「Using the Java Management Extensions (JMX) API」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、論理的または物理的ユニットです。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構造を提供し、これによってさまざまな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、さまざまな組織および管理者が 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

マシンで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 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。

Windows でデフォルトのドメインを起動するには

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` コマンドは使用できません。

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

- 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

「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 の移行の手順

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 台目のマシンにインストールされているのと同じアプリケーションサーバーを、3 台目のマシンにインストールして、3 台目のマシンを設定します。

   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。

   - コマンド行 (対話型) モードを使用して、アプリケーションサーバー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。

- 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に基づき、まったく同一のインストールパスを持つ (つまり、両方のマシンで *install_dir* を使用する) 2 台のマシンでのみサポートされます。

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 でファイル転送してもかまいません。
3. `asadmin restore-domain` コマンドを実行して、ZIP ファイルを 3 番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 install_root/domains/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、*domain1* をバックアップするドメインの名前と置き換えてください。
5. 3 台目のマシンの `domain.xml` で、プロパティのホスト値を変更します。
6. 3 台目のマシンの *install_root/domains/domain1*`/config/domain.xml` を更新します。

   次に例を示します。

   machine1 を検索して、*machine3* と置き換えます。したがって、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 nodeagent の下のプロパティで、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 プロパティ値を変更します。
11. machine2 の nodeagent 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するようにしてください。

Application Server インスタンス

Application Server インスタンスについて

Sun Java System 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 Sun Java System Application Server のインスタンスには、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 Sun Java System 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、「JVM の一般設定」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。

Sun Java System 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。

  図 1-2 は、アプリケーションサーバーインスタンスを詳細に示しています。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。
- **クラスタの定義と使用** - 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。

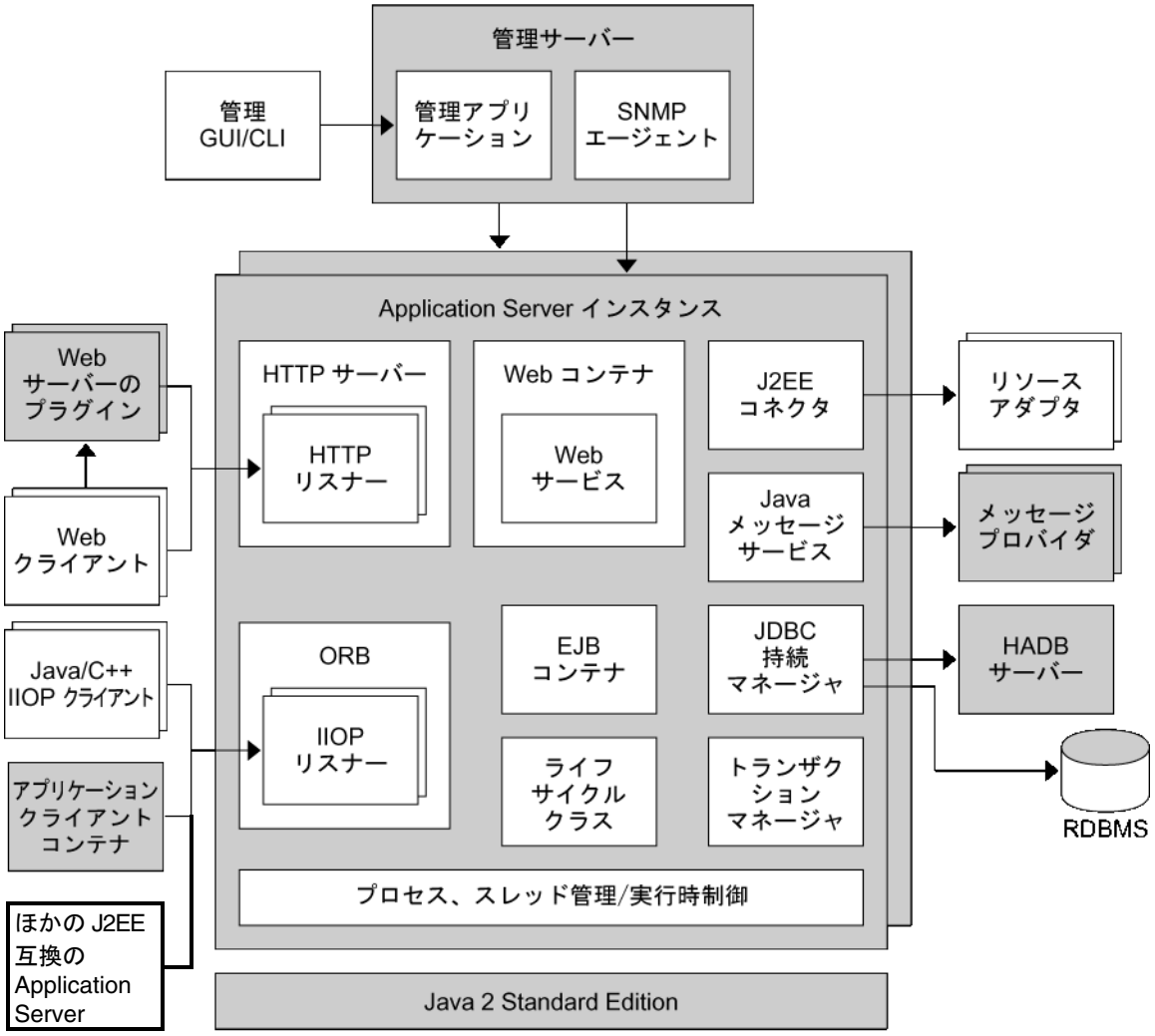


図 1-2　Sun Java System Application Server インスタンス

スタンドアロンインスタンスについて

Sun Java System 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。

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」タブ : インスタンス固有のプロパティを設定します。
- 「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を設定します。

アプリケーションの管理

「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。

アプリケーションを配備するには、次の手順に従います。

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. 「deploy」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)。
   - コネクタモジュール : EIS (Enterprise Information System) 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。

リソースの管理

「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。

新しいリソースタイプを作成するには、次の手順に従います。

1. 必要なリソースのチェックボックスを選択します。
2. 「new」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: コンテナ管理による持続性 Beans (下位互換性のために必要) 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 : メールと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: メールと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 EIS (Enterprise Information System) 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を設定できます。これらのプロパティにより、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

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は *domain_root_dir*/*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_root_dir*/domain1/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。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 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。

「アプリケーション設定」ページから自動配備設定を実行するには、次のようにします。

1. 「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。
2. 「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。
3. 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。
4. 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。
5. 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。

追加のプロパティの設定

「プロパティを追加」ボタンをクリックして、追加する設定を指定します。

ドメイン属性の設定

次のドメイン属性プロパティが利用可能です。

表 1-1　ドメイン属性値

プロパティ	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
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 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。

インスタンス固有設定プロパティ

インスタンス固有の設定プロパティは、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には、次のようにします。

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の追加または削除

プロパティを追加するには、次のようにします。

- 「プロパティを追加」ボタンをクリックして、追加する設定を指定します。

リソースを設定するために、次のプロパティ属性名および値のペアを利用できます。

表 1-2　プロパティ属性名および値のペア

プロパティ	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。

プロパティを削除するには、次のようにします。

1. 削除するプロパティをクリックします。
2. 「プロパティを削除」ボタンをクリックします。

インスタンスの作成

インスタンスを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「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. 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。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を指定します。
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。

同機能を持つ `asadmin` コマンド : `create-instance`

インスタンスの起動

インスタンスを起動するには、次の手順を実行します。

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」ノードを開きます。
2. 起動したいインスタンスを選択します。
3. 「一般」タブで、「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。

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。

インスタンスの停止

インスタンスを停止するには、次の手順を実行します。

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」ノードを開きます。
2. 停止したいインスタンスを選択します。
3. 「一般」タブで、「停止」をクリックしてインスタンスを停止します。

同機能を持つ asadmin コマンド : stop-instance

サーバーインスタンスのシャットダウン

管理サーバーをシャットダウンするには、次の手順に従います。

1. ツリーコンポーネントで、「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

詳細については、「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

表 1-3 は Application Server のポートリスナーについて示しています。

表 1-3 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 ユーティリティを使ってアクセスします。管理コンソールには、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	8081	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。

表 1-3　ポートを使用する Application Server リスナー (続き)

リスナー (Listener)	デフォルトのポート番号	説明
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。

ポート番号の表示

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」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の変更

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 「server-config (管理設定)」ノードを開きます。
3. 「HTTP サービス」ノードを開きます。
4. 「HTTP リスナー」ノードを開きます。
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 ポートの変更

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」ノードを開きます。
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 ポートの変更

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 「server-config (管理設定)」ノードを開きます。
3. 「ORB」ノードを開きます。
4. 「IIOP リスナー」ノードを開きます。
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た JMX コネクタの設定

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。

JMX コネクタを設定するには、次の操作を行います。

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。
    a. 特定のインスタンスを設定するには、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設定を設定するには、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 コネクタ設定の編集

「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。
    a. 特定のインスタンスを設定するには、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設定を設定するには、default-config ノードを選択します。

3. 「管理サービス」ノードを開いて、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 バージョン 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。Transport Layer Security を有効にする場合、「管理サービスの JMX コネクタのセキュリティの設定」にある手順に従って、SSL セクションに記入します。

J2SE ソフトウェアの変更

Application Server は J2SE (Java 2 Standard Edition) 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、「JVM の一般設定」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には、次のようにします。

1. ヘルプブラウザウィンドウ内で、右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。

探している情報が見つからない場合は、次の URL にあるオンラインで入手可能な『Application Server 管理ガイド』を調べてください。

```
http://docs.sun.com/
```

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 Developer's Guide』- このガイドには Application Server に固有の開発情報が説明されています。『Developer's Guide』は次の URL で入手できます。

  ```
  http://docs.sun.com/
  ```

- `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の構文と例が掲載されています。これらの HTML ページは次の URL で入手できます。

  ```
  http://docs.sun.com/
  ```

- 『Application Server リリースノート』- 次の URL で入手できます。

  ```
  http://docs.sun.com/
  ```

- 『Getting Started With J2EE Connectors』- このマニュアルには、コネクタ (リソースアダプタ)、接続プール、およびコネクタリソースを設定するための手順が書かれています。

  ```
  http://java.sun.com/j2ee/connector/
  ```

- 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

クラスタの設定

この章では、管理コンソールを使用してクラスタ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。

クラスタについて

クラスタとは

クラスタは、1 つの論理エンティティとして一体となって動作するゼロまたは 1 つ以上のサーバーインスタンスの集まりです。クラスタは、1 つ以上の J2EE アプリケーションに対して実行環境を提供します。

Sun Java System Application Server Enterprise Edition 8.1 環境でのクラスタの使用により、次が提供されます。

- 高可用性 : クラスタ内のサーバーインスタンスに対するフェイルオーバーを可能にすることで実現します。1 つのサーバーインスタンスが停止すると、別のサーバーインスタンスが、利用できないサーバーインスタンスが処理していた要求を引き継ぎます。

- スケーラビリティ : クラスタにサーバーインスタンスを追加できるようにし、それによってシステムの能力が増強されることによって実現します。Application Server のロードバランサは、クラスタ内の利用可能なサーバーインスタンスに要求を分配します。管理者はより多くのサーバーインスタンスをクラスタに追加しているので、処理の中断の必要はありません。

クラスタには次のプロパティがあります。

- クラスタ内のインスタンスが、すべて同じ設定を参照します。
- クラスタ内のすべてのインスタンスが、J2EE アプリケーションの EAR ファイル、Web モジュールの WAR ファイル、または EJB JAR ファイルなど、配備されたアプリケーションの同じセットを所有します。
- クラスタ内のすべてのインスタンスが同じリソースのセットを所有しており、結果として同じ JNDI 名前空間を構成します。

ドメイン内のすべてのクラスタが一意の名前を持ちます。また、この名前は、すべてのノードエージェント名、サーバーインスタンス名、クラスタ名、および設定名の間でも一意である必要があります。この名前を `domain` に使用してはいけません。管理者は、アプリケーションの配備やリソースの作成など、クラスタ化されていないサーバーインスタンスで実行する操作と同じ操作をクラスタ上で実行します。

クラスタ、ノードエージェント、およびサーバーインスタンスの使用の概要については、『配備計画ガイド』を参照してください。ロードバランサの詳細については、「ロードバランスとフェイルオーバーの設定」を参照してください。

クラスタのタイプ

2 種類のクラスタ、すなわち、スタンドアロンクラスタおよび共有クラスタがあります。

- スタンドアロンクラスタには、ほかのどのサーバーインスタンスやクラスタとも共有していない独自の設定があります。デフォルトで、この設定の名前は *cluster_name*`-config` です。この場合、*cluster_name* はクラスタの名前を表します。
- 共有クラスタは、1 つ以上のクラスタまたはクラスタ化されていないインスタンスと設定を共有します。

クラスタ、インスタンス、ロードバランサ、およびセッション

Application Server でのクラスタ、サーバーインスタンス、ロードバランサ、およびセッションの関係は次のとおりです。

- サーバーインスタンスは、クラスタの一部である必要はありません。ただし、クラスタの一部でないインスタンスは、1 つのインスタンスから別のインスタンスへとセッション状態を移すことによって得られる高可用性を利用することはできません。
- クラスタ内のサーバーインスタンスは、別のマシンまたは同じマシンでホストされます。すなわち、異なるマシンにまたがるサーバーインスタンスを 1 つのクラスタにグループ化できます。
- 特定のロードバランサは、複数のクラスタにあるサーバーインスタンスに要求を転送できます。ロードバランサのこの機能を使って、サービスを中断することなく、オンラインアップグレードを実行できます。詳細については、67 ページの「複数のクラスタを使用してのサービス中断のないオンラインアップグレードサービス」を参照してください。
- 単一のクラスタは、複数のロードバランサから要求を受信できます。クラスタが、2 つ以上のロードバランサからサービスを受ける場合、各ロードバランサで、まったく同一に、クラスタを設定する必要があります。
- 各セッションは、特定のクラスタに関連づけられます。これが意味することは、1 つのアプリケーションは、複数のクラスタに配備できますが、セッションは同じクラスタ内のサーバーインスタンスに対してのみフェイルオーバーされるということです。
- Application Server は、HTTP セッションとステートフルセッション Bean (SFSB) セッションに対するフェイルオーバーをサポートします。HTTP セッション内に保存される特定の J2EE オブジェクト参照のフェイルオーバーもサポートします。

  HTTP セッションのフェイルオーバーの詳細については、157 ページの「可用性とセッション持続性の設定」を参照してください。

したがってクラスタは、そのクラスタ内のサーバーインスタンスがフェイルオーバーしたときには、安全境界として機能します。ロードバランサを使って、サービスを停止することなく、Application Server 内のコンポーネントをアップグレードすることができます。詳細については、67 ページの「複数のクラスタを使用してのサービス中断のないオンラインアップグレードサービス」を参照してください。

クラスタに関する管理コンソールタスク

クラスタの作成

クラスタを作成するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを選択します。
2. 「クラスタ」ページで、「新規」をクリックします。「クラスタの作成」ページが表示されます。
3. 「名前」フィールドで、クラスタの名前を入力します。名前は次の条件を満たす必要があります。
   - 大文字と小文字、数字、下線、ハイフン、およびピリオド (.) だけで構成される
   - すべてのノードエージェント名、サーバーインスタンス名、クラスタ名、および設定名の間で一意である
   - `domain` は使用しない
4. 「構成」フィールドで、ドロップダウンリストから設定を選択します。

   スタンドアロンクラスタを作成するには、`default-config` を選択し、「選択している設定のコピーを作成します」ラジオボタンをオンのままにします。デフォルト設定のコピーは、*cluster_name*`-config` という名前になります。

   別の設定を参照するには、ドロップダウンリストから設定を選択して、「選択している設定を参照します」ラジオボタンをオンにします。設定が別のクラスタで使用されている場合、このアクションにより共有クラスタが作成されます。

5. ここで、サーバーインスタンスを追加するか、またはクラスタが作成されるまで待機することができます。クラスタのサーバーインスタンスを作成する前に、まず 1 つまたは複数のノードエージェントまたはノードエージェントのプレースホルダを作成します。詳細については、110 ページの「ノードエージェントのプレースホルダの作成」を参照してください。

   サーバーインスタンスを作成するには、次の手順を実行します。

   a. 「サーバーインスタンスを作成」セクションで、「追加」をクリックします。

   b. 「インスタンス名」フィールドにインスタンスの名前を入力します。

   c. 「ノードエージェント」ドロップダウンリストからノードエージェントを選択します。

6. 「了解」をクリックします。

7. 表示される「クラスタを正常に作成」ページで「了解」をクリックします。

クラスタ、サーバーインスタンス、およびノードエージェントを管理する方法の詳細については、104 ページの「ノードエージェントの配備」を参照してください。

同機能を持つ asadmin コマンド : `create-cluster`

クラスタの設定

クラスタを設定するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。

2. クラスタのノードを選択します。「一般情報」ページで、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、クラスタ化されたサーバー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、クラスタ化されたサーバーインスタンスを停止します。
   - 「EJB タイマーを移行」をクリックして、停止されたサーバーインスタンスからクラスタ内の別のサーバーインスタンスに EJB タイマーを移行します。

同機能を持つ asadmin コマンド : `start-cluster`、`stop-cluster`、`migrate-timers`

EJB タイマーの移行

サーバーインスタンスが異常に、または突然実行を停止した場合、そのサーバーインスタンス上にインストールされた EJB タイマーを、クラスタ内の実行中サーバーインスタンスに移動する必要があります。これを実行するには、次の手順を実行します。

1. 「ソース」ドロップダウンリストから、タイマーの移行元である停止されたサーバーインスタンスを選択します。
2. オプションとして、「送信先」ドロップダウンリストから、タイマーを移行する先の実行中サーバーインスタンスを選択します。このフィールドを空白のままにすると、実行中のサーバーインスタンスがランダムに選択されます。
3. 「了解」をクリックします。
4. 送信先サーバーインスタンスを停止して再起動します。

ソースサーバーインスタンスが実行中の場合、または送信先サーバーインスタンスが実行中でない場合はエラーメッセージが表示されます。

同機能を持つ `asadmin` コマンド : `migrate-timers`

クラスタのサーバーインスタンスの作成

クラスタのサーバーインスタンスを作成する前に、まずノードエージェントまたはノードエージェントのプレースホルダを作成します。詳細については、110 ページの「ノードエージェントのプレースホルダの作成」を参照してください。

クラスタのサーバーインスタンスを作成するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. クラスタのノードを選択します。
3. 「インスタンス」タブをクリックして、「クラスタ化されたサーバーインスタンス」ページを表示します。
4. 「新規」をクリックして、「クラスタ化されたサーバーインスタンスの作成」ページを表示します。
5. 「名前」フィールドで、サーバーインスタンスの名前を入力します。
6. 「ノードエージェント」ドロップダウンリストからノードエージェントを選択します。
7. 「了解」をクリックします。

同機能を持つ `asadmin` コマンド : `create-instance`

クラスタ化されたサーバーインスタンスの設定

クラスタ化されたサーバーインスタンスをいったん作成したあとで変更するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. サーバーインスタンスを含むクラスタのノードを開いて、編集するサーバーインスタンスのノードを選択します。
3. 「一般情報」ページで、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、実行するインスタンスを停止します。
    - 「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。
    - 「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。
    - 「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。
    - 「プロパティ」タブをクリックして、インスタンスのポート番号を変更します。
    - 「監視」タブをクリックして、監視プロパティを変更します。

サーバーインスタンスに対して次のような処理を実行することもできます。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. サーバーインスタンスを含むクラスタ用のノードを開きます。
3. 「インスタンス」タブをクリックして、「クラスタ化されたサーバーインスタンス」ページを表示します。このページで、次のタスクを実行できます。
    - インスタンス用のチェックボックスを選択し、「削除」、「起動」、「停止」をクリックします。
    - インスタンスの名前をクリックして、「一般情報」ページを表示します。

クラスタ化されたサーバーインスタンスをクラスタから削除するには、インスタンス名の横にあるチェックボックスを選択して、「削除」をクリックします。

クラスタのアプリケーションの設定

クラスタのアプリケーションを設定するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. クラスタのノードを選択します。
3. 「アプリケーション」タブをクリックして、「アプリケーション」ページを表示します。このページで、次のタスクを実行できます。
    - アプリケーションの横にあるチェックボックスを選択して、「有効」または「無効」を選択し、クラスタのアプリケーションを有効または無効にします。
    - 「配備」ドロップダウンリストから、配備するアプリケーションのタイプを選択します。表示される「配備」ページで、アプリケーションを指定します。
    - 「フィルタ」ドロップダウンリストから、リストに表示するアプリケーションのタイプを選択します。

アプリケーションを編集するには、アプリケーション名をクリックします。

クラスタのリソースの設定

クラスタのリソースを設定するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. クラスタのノードを選択します。
3. 「リソース」タブをクリックして、「リソース」ページを表示します。このページで、次のタスクを実行できます。
    - リソースの横にあるチェックボックスを選択して、「有効」または「無効」を選択し、リソースを広域的に有効または無効にします。このアクションはリソースを削除しません。
    - 「新規」ドロップダウンリストから、作成するリソースのタイプを選択します。リソースを作成するときには、必ずクラスタをターゲットとして指定します。
    - 「フィルタ」ドロップダウンリストから、リストに表示するリソースのタイプを選択します。

リソースを編集するには、リソース名をクリックします。

クラスタの削除

クラスタを削除するには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを選択します。
2. 「クラスタ」ページで、削除するクラスタの横にあるチェックボックスを選択します。
3. 「削除」をクリックします。

同機能を持つ asadmin コマンド : `delete-cluster`

複数のクラスタを使用してのサービス中断のないオンラインアップグレードサービス

ロードバランサと複数のクラスタを使用して、サービスを停止することなく、Application Server 内のコンポーネントをアップグレードできます。たとえば、コンポーネントとして、JVM、Application Server、または Web アプリケーションが可能です。

このタスクを実行するには、次を実行します。

1. クラスタの「一般情報」ページで「クラスタの停止」ボタンを使って、クラスタの 1 つを停止します。
2. そのクラスタでコンポーネントをアップグレードします。
3. クラスタの「一般情報」ページで「クラスタの開始」ボタンを使って、クラスタを開始します。
4. ほかのクラスタで、1 つずつプロセスを繰り返します。

1 つのクラスタ内のセッションは、別のクラスタ内のセッションにフェイルオーバーしないので、1 つのバージョンのコンポーネントを実行しているサーバーインスタンスから、異なるバージョンのコンポーネントを実行している (別のクラスタ内の) 別のサーバーインスタンスへのセッションフェイルオーバーによって、バージョンミスマッチが発生するリスクはありません。クラスタは、そのクラスタ内のサーバーインスタンスがフェイルオーバーしたときには、安全境界としてこのように機能します。

注 このアプローチは、次のケースでは利用できません。

- 高可用性データベース (HADB) のスキーマを変更する場合。詳細については、『Sun Java System Application Server High Availability Administration Guide』の「Administering High Availability Database」を参照してください。
- アプリケーションデータベーススキーマに対する変更を含むアプリケーションアップグレードを実行する場合。

警告 クラスタ内のすべてのサーバーインスタンスは一緒にアップグレードします。そうでないと、1 つのインスタンスから、異なるバージョンのコンポーネントを実行するインスタンスへフェイルオーバーするセッションによって、バージョンミスマッチが発生するリスクがあります。

第 3 章

ロードバランスとフェイルオーバーの設定

この章では、Sun Java System Application Server の HTTP 要求のロードバランスの設定方法について説明します。また、ロードバランサによって制御されるサーバーインスタンス間のフェイルオーバーの設定方法についても説明します。さらに、RMI-IIOP ロードバランスとフェイルオーバーについても説明します。

この章には次の節が含まれています。

HTTP ロードバランスとフェイルオーバーについて

HTTP ロードバランスとフェイルオーバー

ロードバランスの目的は、スタンドアロンまたはクラスタ化された複数の Sun Java System Application Server インスタンスの間でワークロードを均等に分散させ、その結果、システム全体のスループットを増加することです。

ロードバランサを使用しても、1 つのインスタンスから別のインスタンスへのフェイルオーバーを要求できます。HTTP セッションの情報を持続させるために、HTTP セッションの持続性を設定します。詳細については、「可用性とセッション持続性の設定」を参照してください。

管理コンソールではなく、asadmin ツールを使用して、HTTP ロードバランスを設定します。

HTTP ロードバランスに対する要件

HTTP 要求にロードバランサプラグインを使用するには、次の要件を満たす必要があります。

- Sun Java System Application Server Enterprise Edition がインストールされていること。
- Web サーバーがインストールおよび設定されていること。

  詳細については、73 ページの「HTTP ロードバランスのための Web Server の設定」を参照してください。
- ロードバランサプラグインがインストールされていること。
- Application Server が設定されていること。
  - Application Server ロードバランスに関わる Application Server インスタンスまたはクラスタが作成されていること。
  - ロードバランスに関わるすべての Application Server インスタンスまたはクラスタに、アプリケーションが配備されていること。
  - ロードバランスに関わるサーバーインスタンスとクラスタが同種の環境を確保していること。これは、通常、サーバーインスタンスが同じサーバー設定を参照し、同じアプリケーションを配備していることを意味します。

割り当て要求と非割り当て要求について

最初に HTTP クライアントからロードバランサに要求が入ってきた時点で、その要求は新規セッションの要求となります。新しいセッションに対する要求は、非割り当て要求と呼ばれます。ロードバランサはこの要求を、ラウンドロビンアルゴリズムに従って、クラスタ内のアプリケーションサーバーインスタンスにルーティングします。詳細については、71 ページの「HTTP ロードバランスアルゴリズム」を参照してください。

セッションがアプリケーションサーバーインスタンスに作成されると、ロードバランサは、そのセッションに関するすべての後続要求を特定のインスタンスにだけルーティングします。既存のセッションに対する要求は、割り当てまたはスティッキ要求と呼ばれます。

HTTP ロードバランスアルゴリズム

Sun Java System Application Server のロードバランサは、スティッキラウンドロビンアルゴリズムを使用して、着信 HTTP および HTTPS 要求をロードバランスします。特定のセッションに対するすべての要求は、同じアプリケーションサーバーインスタンスに送信されます。スティッキロードバランサでは、セッションデータは、クラスタ内の全インスタンスに分散されるのではなく、単一のアプリケーションサーバーにキャッシュされます。

したがって、スティッキラウンドロビンスキーマでは、パフォーマンス上、大きなメリットが得られます。これは、純粋なラウンドロビンによる、より均一化された分散負荷によるメリット以上のものです。

スティッキラウンドロビンロードバランスアルゴリズムについて

新しい HTTP 要求がロードバランサプラグインに送信されると、単純なラウンドロビンスキームに基づいてアプリケーションサーバーインスタンスに転送されます。次にこの要求は、Cookie または URL を明示的に書き換えることによって、この特定のアプリケーションサーバーに「固定」されます。

スティッキ情報から、ロードバランサプラグインは、まず、以前に要求が転送されたインスタンスを判断します。そのインスタンスが正常であるとわかると、ロードバランサプラグインは、要求をその特定のアプリケーションサーバーインスタンスに転送します。したがって、特定のセッションに対するすべての要求が同じアプリケーションサーバーインスタンスに送信されます。

ロードバランサプラグインは次の方法を使ってセッションのスティッキ度を判断します。

- Cookie に基づいた方法
- URL の明示的書き換えによる方法

Cookie に基づいた方法

Cookie に基づいた方法では、ロードバランサプラグインは、個別の Cookie を使って、ルート情報を記録します。

注	Cookie に基づいた方法を使用するには、HTTP クライアントが Cookie をサポートしている必要があります。

URL の明示的書き換えによる方法

URL を明示的に書き換えて、スティッキ情報を URL に追加します。この方法は、HTTP クライアントが Cookie をサポートしない場合でも機能します。

ロードバランスとフェイルオーバーのサンプルアプリケーション

次のディレクトリには、ロードバランスとフェイルオーバーを示すサンプルアプリケーションが含まれています。

install_dir`/samples/ee-samples/highavailability`

install_dir`/samples/ee-samples/failover`

`ee-samples` ディレクトリには、サンプルを実行する環境を設定するための情報も含まれています。

HTTP ロードバランス設定の概要

`asadmin` ツールを使用して、環境に対するロードバランスを設定します。次の手順に従います。

1. Web サーバーと Application Server インスタンスとクラスタの両方、またはどちらかのインストールと設定など、70 ページの「HTTP ロードバランスに対する要件」を完了します。
2. `asadmin` コマンドの `create-http-lb-config` を使用して、ロードバランサ設定を作成します。
3. 作成したロードバランサのクラスタとスタンドアロンサーバーインスタンスへの参照を追加し、`asadmin create-http-lb-ref` を使用して管理するようにします。

   ターゲットを指定してロードバランサ設定を作成しており、そのターゲットが、ロードバランサが参照する唯一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である場合は、この手順を飛ばしてください。

4. `asadmin enable-http-lb-server` を使用してロードバランサが実行するクラスタまたはスタンドアロンサーバーインスタンスの参照を有効にします。
5. `asadmin enable-http-lb-application` を使用してロードバランスするアプリケーションを有効にします。

   これらのアプリケーションは、ロードバランサが参照するクラスタまたはスタンドアロンインスタンスで使用するために、事前に配備および有効にしておく必要があります。ロードバランスを有効にする手順は、使用可能にする手順とは別です。
6. `asadmin create-health-checker` を使用して、診断プログラムを作成します。

   診断プログラムは、正常でないサーバーインスタンスを監視し、それらが正常に戻ったときにロードバランサが新しい要求を送信できるようにします。
7. `asadmin export-http-lb-config` を使用して、ロードバランサ設定ファイルを生成します。

   このコマンドは、Sun Java System Application Server に同梱されているロードバランサプラグインと一緒に使用する設定ファイルを生成します。
8. ロードバランサ設定ファイルを、ロードバランサプラグイン設定ファイルが格納されている Web サーバーの `config` ディレクトリにコピーします。

HTTP ロードバランスのための Web Server の設定

Web サーバーの設定について

ロードバランサプラグインインストールプログラムは、Web サーバーの設定ファイルに対していくつかの変更を加えます。これらの変更は、Web サーバーによって異なります。

注 ロードバランサプラグインは、サポートされている Web サーバーを実行するマシン上に、Sun Java System Application Server Enterprise Edition とともにインストールすることも、または個別にインストールすることもできます。

インストール手順の詳細については、『Sun Java System Application Server Installation Guide』を参照してください。

Sun Java System Web Server に対する変更

インストールプログラムは、Sun Java System Web Server の設定ファイルに対して次のような変更を加えます。

1. 次のようなロードバランサプラグイン固有のエントリを、Web サーバーインスタンスの `magnus.conf` ファイルに追加します。

```
##EE lb-plugin
Init fn="load-modules"
shlib="web_server_install_dir/plugins/lbplugin/bin/libpassthrough.so"
funcs="init-passthrough,service-passthrough,name-trans-passthrough"
Thread="no"

Init fn="init-passthrough"

##end addition for EE lb-plugin
```

2. 次のようなロードバランサプラグイン固有のエントリを、Web サーバーインスタンスの `obj.conf` ファイルに追加します。

```
<Object name=default>

NameTrans fn="name-trans-passthrough" name="lbplugin"
config-file="web_server_install_dir/web_server_instance/config/loadbalancer.xml"

<Object name="lbplugin">
ObjectType fn="force-type" type="magnus-internal/lbplugin"
PathCheck fn="deny-existence" path="*/WEB-INF/*"
Service type="magnus-internal/lbplugin" fn="service-passthrough"
Error reason="Bad Gateway" fn="send-error"
uri="$docroot/badgateway.html"
</object>
```

lbplugin は、Object を一意に識別する名前であり、*web_server_install_dir*/*web_server_instance*/config/loadbalancer.xml は、ロードバランサが動作するように設定されている仮想サーバーの XML 設定ファイルの場所です。

インストールが完了したら、72 ページの「HTTP ロードバランス設定の概要」で説明されている方法でロードバランサを設定します。

Apache Web サーバーに対する変更

ロードバランサプラグインを Apache にインストールする前に、付録 A 「Apache Web サーバーのコンパイルと設定」の Apache のコンパイルと設定に関する情報を参照してください。

インストーラによって行われる変更

ロードバランサプラグインインストールプログラムは、必要なファイルを、Web サーバーのルートディレクトリの下の libexec (Apache 1.3) または modules (Apache 2.0) フォルダに抽出します。このプログラムはまた、次のようなロードバランサプラグイン固有のエントリを、Web サーバーインスタンスの httpd.conf ファイルに追加します。

```
<VirtualHost machine_name:443>

##Addition for EE lb-plugin

LoadFile /usr/lib/libCstd.so.1

LoadModule apachelbplugin_module libexec/mod_loadbalancer.so
#AddModule mod_apachelbplugin.cpp
<IfModule mod_apachelbplugin.cpp>
        config-file webserver_instance/conf/loadbalancer.xml
locale en
</IfModule>

<VirtualHost machine_ip_address>
DocumentRoot "webserver_instance/htdocs"
ServerName server_name
</VirtualHost>

##END EE LB Plugin ParametersVersion 7
```

注

- Apache 1.3 で、複数の Apache の子プロセスが動作している場合、各プロセスは固有のロードバランスラウンドロビンシーケンスを使用しています。

  たとえば、Apache の子プロセスが 2 つ動作していて、ロードバランスプラグインが 2 つのアプリケーションサーバーインスタンスに対してロードバランスする場合、最初の要求はインスタンス #1 に送信され、2 番目の要求もインスタンス #1 に送信されます。3 番目の要求はインスタンス #2 に送信され、4 番目の要求も同じくインスタンス #2 に送信されます。このパターンが繰り返されます (インスタンス 1、インスタンス 1、インスタンス 2、インスタンス 2、以下同様)。

  この動作は、インスタンス 1、インスタンス 2、インスタンス 1、インスタンス 2、というユーザーが予想する順序とは異なります。Sun Java System Application Server では、Apache 用のロードバランスプラグインは Apache プロセスごとにロードバランサインスタンスをインスタンス化して、独立したロードバランスシーケンスを作成します。

- `--with-mpm=worke` オプションを使用してコンパイルした場合、Apache 2.0 は動作をマルチスレッド化します。

インストール後の変更

Apache Web サーバーは、ロードバランサプラグインを使用して正常に動作するために、セキュリティ保護されたモードで実行される必要があります。*apache_install_dir* の下に `sec_db_files` というディレクトリを作成して、*application_server_domain_dir*`/config/`*security_db_files* を *apache_install_dir*`/`*sec_db_files* にコピーします。

Microsoft Windows に対する追加の変更

プラグインのインストール後に Microsoft Windows 上で Apache を実行している場合は、一部の環境変数を変更する必要があります。

「スタート」->「設定」->「コントロールパネル」->「システム」->「詳細」->「環境変数」->「システム環境変数」を選択して、Path 環境変数に新しいパスを追加します。Path 変数に、次の内容が含まれるように編集します。

application_server_install_dir`/bin`

さらに、Apache Web サーバーを起動する前に、環境変数 NSPR_NATIVE_THREADS_ONLY を 1 に設定します。

「環境変数」ウィンドウで、「システム環境変数」の下の「新規」をクリックします。次の名前と値のペアを入力します。

変数名 : NSPR_NATIVE_THREADS_ONLY

変数値 : 1

マシンを再起動します。

Microsoft IIS に対する変更

Microsoft Internet Information Services (IIS) がロードバランサプラグインを使用するように設定するには、Windows Internet Services Manager の特定のプロパティを変更します。Internet Services Manager は、「コントロールパネル」フォルダの「管理ツール」フォルダに置かれています。

Sun Java System Application Server をインストールしたら、次の変更を行います。

1. Internet Services Manager を開きます。
2. プラグインを有効にする Web サイトを選択します。この Web サイトは通常、デフォルトの Web サイトとして指定されます。
3. この Web サイト上で右クリックして「プロパティ」を選択し、「プロパティ」ノートブックを開きます。
4. 新しい ISAPI フィルタを追加するには、「ISAPI フィルタ」タブを開いて「追加」をクリックし、次の手順に従います。
   a. 「フィルタ名」フィールドに、「Application Server」と入力します。
   b. 「実行ファイル」フィールドに、「`C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.dll`」と入力します。
   c. 「了解」をクリックして、「プロパティ」ノートブックを閉じます。
5. 新しい仮想ディレクトリを作成および設定します。
   a. デフォルトの Web サイト上で右クリックして「新規」を選択し、「仮想ディレクトリ」を選択します。

      「仮想ディレクトリの作成ウィザード」が開きます。
   b. 「エイリアス」フィールドに、「`sun-passthrough`」と入力します。
   c. 「ディレクトリ」フィールドに、「`C:¥Inetpub¥wwwroot¥sun-passthrough`」と入力します。
   d. 「実行パーミッション」チェックボックスにチェックマークを付けます。ほかのすべてのパーミッション関連のチェックボックスは、チェックしないでおきます。
   e. 「完了」をクリックします。

6. システムの Path 環境変数に、sun-passthrough.dll ファイルおよび *application_server_install_dir*/bin のパスを追加します。マシンを再起動します。
7. Web サーバーを停止してから起動して、新しい設定を反映させます。

   Web サーバーを停止するには、Web サイト上で右クリックして「停止」を選択します。Web サーバーを起動するには、Web サイト上で右クリックして「起動」を選択します。

   次に、Web ブラウザに以下のように入力して Web アプリケーションのコンテキストルートにアクセスします。

   http://*webserver_name*/*web_application*

   webserver_name は Web サーバーのホスト名または IP アドレスで、/*web_application* は C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.properties ファイルに一覧表示したコンテキストルートです。Web サーバー、ロードバランサプラグイン、および Application Server が正常に動作していることを確認します。

インストーラは、sun-passthrough.properties 内に次のプロパティを自動的に設定します。デフォルト値は変更可能です。

表 3-1　自動的に設定される Microsoft IIS の sun-passthrough プロパティ

プロパティ	定義	デフォルト値
lb-config-file	ロードバランサ設定ファイルへのパス	*IIS_www_root*¥sun-passthrough¥loadbalancer.xml
log-file	ロードバランサログファイルへのパス	*IIS_www_root*¥sun-passthrough¥lb.log
log-level	Web サーバーのログレベル	INFO

複数の Web サーバーインスタンスの設定

Sun Java System Application Server インストーラでは、1 台のマシンに複数のロードバランサプラグインをインストールできません。1 台のマシンの 1 つまたは複数のクラスタ内に、ロードバランサプラグインとともに複数の Web サーバーを置くには、いくつかの手順を手動で実行してロードバランサプラグインを設定する必要があります。

1. 74 ページの「Sun Java System Web Server に対する変更」、75 ページの「Apache Web サーバーに対する変更」、および 77 ページの「Microsoft IIS に対する変更」で説明されている方法で、ロードバランサプラグインを使用する新しい Web サーバーインスタンスを設定します。

2. 既存の Web サーバーインスタンスの config ディレクトリから、sun-loadbalancer_1_1.dtd ファイルを新しいインスタンスの config ディレクトリにコピーします。
3. 同じロードバランサ設定を使用するために、既存の Web サーバーインスタンスの config ディレクトリから、loadbalancer.xml ファイルを新しいインスタンスの config ディレクトリにコピーします。
4. 異なるロードバランサ設定を使用する場合は、次の手順に従います。
   a. asadmin create-http-lb-config を使用して、新しいロードバランサ設定を作成します。
   b. asadmin export http-lb-config を使用して、新しい設定を loadbalancer.xml ファイルにエクスポートします。
   c. loadbalancer.xml ファイルを、新しい Web サーバーの config ディレクトリにコピーします。

   ロードバランサ設定を作成し、loadbalancer.xml ファイルにエクスポートすることについて詳しくは、「HTTP ロードバランサ設定タスク」を参照してください。

HTTP ロードバランサ設定タスク

HTTP ロードバランサ設定の作成

ロードバランサ設定は、ロードバランサを定義する `domain.xml` ファイル内で名前を付けられている設定です。

ロードバランス設定は非常に柔軟性があります。

- 各ロードバランサ設定は、関連する複数のロードバランサを持つことができます。ただし、1 つのロードバランサには、1 つのロードバランサ設定しかできません。
- ロードバランサがサービスを提供するドメインは 1 つだけですが、ドメインは関連する複数のロードバランサを持つことができます。

`asadmin` コマンド `create-http-lb-config` を使用して、設定を作成します。次のパラメータを指定します。

- 応答タイムアウト

  サーバーインスタンスが応答を返すまでの秒単位のタイムアウト。タイムアウト時間内に応答が着信しない場合、サーバーが正常でないと判断されます。デフォルトは 60 です。

- HTTPS ルーティング

  ロードバランサに対する HTTPS 要求の結果が、サーバーインスタンスに対する HTTPS または HTTP 要求となるかどうかを指定します。

  詳細については、87 ページの「HTTP および HTTPS セッションのフェイルオーバーの設定」を参照してください。

- 再読み込み間隔

  ロードバランサ設定ファイル `loadbalancer.xml` に対する変更をチェックする間隔。チェックによって変更が検出されると、設定ファイルが再読み込みされます。値 0 の場合、再読み込みを無効にします。

  詳細については、85 ページの「動的再設定の有効化」を参照してください。

- 監視

  ロードバランサで監視が有効かどうかを指定します。

  詳細については、90 ページの「HTTP ロードバランサプラグインの監視」を参照してください。

- ルート Cookie

  ロードバランサプラグインがルート情報を記録するために使用する Cookie の名前を指定します。HTTP クライアントは Cookie をサポートする必要があります。Cookie を格納する前にブラウザが確認してくるように設定すると、Cookie の名前は「JROUTE」となります。

- ターゲット

  ロードバランサ設定のターゲットを指定します。ターゲットを指定すると、設定に参照を追加した場合と同じ結果になります。ターゲットは、クラスタまたはスタンドアロンインスタンスです。

詳細については、create-http-lb-config、delete-http-lb-config、および list-http-lb-configs のドキュメントを参照してください。

HTTP ロードバランサ参照の作成

ロードバランサでスタンドアロンのサーバーまたはクラスタに対する参照を作成する場合、ロードバランサが制御するターゲットサーバーまたはクラスタの一覧に、参照先のサーバーおよびクラスタが追加されます。この場合でも、参照先のサーバーまたはクラスタに対する要求を負荷分散するには、enable-http-lb-server を使用してそのサーバーまたはクラスタを有効化する必要があります。ターゲットを指定してロードバランサ設定を作成した場合、そのターゲットはすでに参照として追加されています。

create-http-lb-ref を使用して参照を作成します。ロードバランサ設定名と、ターゲットサーバーインスタンスまたはクラスタを指定する必要があります。

参照を削除するには、delete-http-lb-ref を使用します。参照を削除する前に、disable-http-lb-server を使用して参照先のサーバーまたはクラスタを無効にする必要があります。

詳細については、create-http-lb-ref および delete-http-lb-ref のドキュメントを参照してください。

ロードバランスのためのサーバーインスタンスの有効化

サーバーインスタンスまたはクラスタへの参照を作成したら、enable-http-lb-server を使用してサーバーインスタンスまたはクラスタを有効にします。ロードバランサ設定の作成時にサーバーインスタンスまたはクラスタをターゲットとして使用した場合は、それを有効にする必要があります。

詳細については、enable-http-lb-server のドキュメントを参照してください。

ロードバランスのためのアプリケーションの有効化

ロードバランサによって管理されるすべてのサーバーは、アプリケーションの同じセットが配備されているなど、同一の設定である必要があります。アプリケーションが配備されてアクセス可能になると (配備手順の実行中または完了後)、ロードバランスを有効にする必要があります。アプリケーションでロードバランスが有効化されていない場合、そのアプリケーションが配備されているサーバーへの要求が負荷分散およびフェイルオーバーされていても、アプリケーションへの要求は負荷分散およびフェイルオーバーされません。

アプリケーションを有効にする際に、アプリケーション名とターゲットを指定します。ロードバランサが複数のターゲット (2 つのクラスタなど) を管理している場合は、すべてのターゲットでアプリケーションを有効にしてください。

詳細については、`enable-http-lb-application` のオンラインヘルプを参照してください。

新しいアプリケーションを配備する場合にも、アプリケーションでロードバランスを有効にして、再度ロードバランサ設定をエクスポートする必要があります。

HTTP 診断プログラムの作成

ロードバランサの診断プログラムは、設定されている Application Server インスタンスで正常ではないとしてマークされているものをすべて、定期的にチェックします。診断プログラムは必須ではありませんが、このプログラムが存在しない場合、または無効になっている場合は、正常でないインスタンスの定期的な診断プログラムは実行されません。

ロードバランサの診断プログラムメカニズムは、HTTP を使用してアプリケーションサーバーと通信します。診断プログラムは、指定された URL に HTTP 要求を送信し、応答を待ちます。HTTP 応答ヘッダー内の状態コードが 100 ～ 500 の間であれば、インスタンスが正常であることを示します。

診断プログラムの作成

診断プログラムを作成するには、`asadmin create-http-health-checker` コマンドを使用します。次のパラメータを指定します。

- url

  ロードバランサが健康状態を判断するためにチェックするリスナーの URL を指定します。デフォルトは `"/"` です。

- `interval`

インスタンスの診断プログラムを実行する間隔を秒単位で指定します。デフォルトは 30 秒です。0 を指定すると、診断プログラムが無効になります。

- timeout

  正常だとみなされるリスナーが応答を受け取るまでのタイムアウトを秒単位で指定します。デフォルトは 10 秒です。

アプリケーションサーバーインスタンスが正常でないとマークされている場合、診断プログラムが正常ではないインスタンスをポーリングして、インスタンスが正常になったかどうかを判断します。診断プログラムは、指定された URL を使用して正常でないアプリケーションサーバーインスタンスをすべてチェックし、それらが正常な状態に戻っているかどうかを判断します。

診断プログラムにより、正常ではないインスタンスが正常になったことが確認されると、そのインスタンスが正常なインスタンスのリストに加えられます。

詳細については、`create-http-health-checker` および `delete-http-health-checker` のドキュメントを参照してください。

正常なインスタンス用診断プログラムの追加プロパティ

`create-http-health-checker` によって作成された診断プログラムは、正常ではないインスタンスのみをチェックします。正常なインスタンスを定期的にチェックするには、エクスポートした `loadbalancer.xml` ファイルに追加のプロパティをいくつか設定します。

注 これらのプロパティは、`loadbalancer.xml` ファイルをエクスポート後に手動で編集することによってのみ設定できます。同機能を持つ asadmin コマンドはありません。

正常なインスタンスをチェックするには、次のプロパティを設定します。

表 3-2 診断プログラムのプロパティ

プロパティ	定義
active-healthcheck-enabled	サーバーインスタンスが正常であるかどうかを調べるために、それらに対して Ping を実行するかどうかを示す true/false フラグ。サーバーインスタンスに対して Ping を実行するには、このフラグを true に設定します。
number-healthcheck-retries	ロードバランサの診断プログラムが、応答しないサーバーインスタンスを正常でないとマークするまでに、それらに対して Ping を実行する回数を指定します。有効な範囲は 1 ～ 1000 です。デフォルト値は 3 に設定します。

loadbalancer.xml ファイルを編集して、プロパティを設定します。次に例を示します。

```
<property name="active-healthcheck-enabled" value="true"/>
<property name="active-healthcheck-retries" value="3"/>
```

これらのプロパティを追加し、続いて loadbalancer.xml ファイルをふたたび編集およびエクスポートする場合、新しくエクスポートされた設定には追加のプロパティが含まれないため、これらを再度ファイルに追加する必要があります。

ロードバランサ設定ファイルのエクスポート

Sun Java System Application Server に同梱されているロードバランスプラグインは、loadbalancer.xml という設定ファイルを使用します。asadmin ツールを使用して、domain.xml ファイルにロードバランサ設定を作成します。ロードバランス環境を設定したら、その環境をファイルにエクスポートします。

1. asadmin コマンド export-http-lb-config を使用して、loadbalancer.xml ファイルをエクスポートします。

   特定のロードバランサ設定の loadbalancer.xml ファイルをエクスポートします。パスまたは別のファイル名を指定できます。ファイル名を指定しない場合、ファイルには loadbalancer.xml.*load_balancer_config_name* という名前が付けられます。パスを指定しない場合、ファイルは *application_server_install_dir*/domains/*domain_name*/generated ディレクトリに作成されます。

   Windowd でパスを指定する場合は、パスを引用符で囲みます。たとえば、"c:¥sun¥AppServer¥loadbalancer.xml" のように指定します。

2. エクスポートしたロードバランサ設定ファイルを、Web サーバーの設定ディレクトリにコピーします。

   たとえば、Sun Java System Web Server の場合、コピー先は *web_server_root*/config となります。

   Web サーバーの設定ディレクトリ内のロードバランサ設定ファイルには、loadbalancer.xml という名前を付ける必要があります。loadbalancer.xml.*load_balancer_config_name* などの別の名前を付けた場合は、変更する必要があります。

HTTP ロードバランサ設定の変更

サーバーに対する参照の作成または削除、新しいアプリケーションの配備、サーバーまたはアプリケーションの有効化 / 無効化などによって HTTP ロードバランサ設定を変更する場合は、ロードバランサ設定ファイルをふたたびエクスポートして、Web サーバーの `config` ディレクトリにコピーします。詳細については、84 ページの「ロードバランサ設定ファイルのエクスポート」を参照してください。

ロードバランサプラグインは、ロードバランサ設定で指定した再読み込み間隔に従って、更新された設定を定期的にチェックします。指定した時間が経過して、ロードバランサが新しい設定ファイルを検出した場合は、その設定を使用して再読み込みが開始されます。

動的再設定の有効化

動的な再設定が有効になっている場合、ロードバランサプラグインは設定の更新を定期的にチェックします。動的再設定を有効にするには、次の手順に従います。

- ロードバランサ設定を作成する際に動的再設定を有効にするには、`asadmin create-http-lb-config` の実行時に `--reloadinterval` オプションを使用します。

  このオプションは、ロードバランサ設定ファイル `loadbalancer.xml` に対する変更のチェックの間隔を設定します。値 0 の場合、再読み込みを無効にします。デフォルトでは、動的再読み込みが有効化され、間隔は 60 秒に設定されています。

- 無効にしていた動的再読み込みを有効にする、または、再読み込み間隔を変更する場合は、`asadmin set` コマンドを使用します。

これらの設定を変更したら、ロードバランサ設定ファイルを再度エクスポートし、Web サーバーの `config` ディレクトリにコピーします。

以前は無効に設定されていた動的再設定を有効にするには、Web サーバーを再起動する必要があります。

注

- ロードバランサがそれ自体の再設定を試みているときにハードディスク読み込みエラーが発生した場合、ロードバランサは現在メモリに格納されている設定を使用します。ロードバランサはまた、既存の設定を上書きする前に、変更された設定データが必ず DTD に準拠するようにします。

  ディスク読み込みエラーが発生すると、Web サーバーのエラーログファイルに警告メッセージが記録されます。

  Sun Java System Web Server のエラーログは、次の場所にあります。
  web_server_install_dir`/`*webserver_instance*`/logs/`

サーバーインスタンスまたはクラスタの無効化 (停止)

何らかの理由でアプリケーションサーバーを停止する前には、インスタンスが要求の処理を完了する必要があります。サーバーインスタンスまたはクラスタを正常に無効にするプロセスは、停止と呼ばれます。

ロードバランサは、アプリケーションサーバーインスタンスを停止するために、次のポリシーを使用します。

- あるインスタンス (スタンドアロンまたはクラスタの 1 部分) が削除され、タイムアウトが経過していない場合、スティッキ要求はインスタンスに配信され続けます。ただし、新しい要求は無効化されたインスタンスに送信されません。
- タイムアウトを経過すると、インスタンスは無効化されます。ロードバランサからインスタンスへのすべてのオープン接続が閉じられます。このインスタンスに固定されている一部のセッションが無効化されなかった場合でも、ロードバランサはこのインスタンスに要求を送りません。ロードバランサはスティッキ要求を別の正常なインスタンスにフェイルオーバーします。

サーバーインスタンスまたはクラスタを無効にするには、次の手順に従います。

1. `asadmin disable-http-lb-server` を実行して、タイムアウトを分単位で設定します。
2. `asadmin export-http-lb-config` を使用して、ロードバランサ設定ファイルをエクスポートします。
3. エクスポートした設定を Web サーバーの `config` ディレクトリにコピーします。
4. サーバーインスタンスを停止します。

アプリケーションの無効化 (停止)

Web アプリケーションの配備を取り消す前に、アプリケーションが要求の処理を完了する必要があります。アプリケーションを正常に無効にするプロセスは、停止と呼ばれます。

ロードバランサは、アプリケーションを停止するために、次のポリシーを使用します。

- あるアプリケーションが無効化され、タイムアウトが経過していない場合、ロードバランサは無効にしたアプリケーションに対して新しい要求を転送しません。これらの要求は Web サーバーに返されます。スティッキ要求は、タイムアウトが経過するまで転送し続けられます。
- タイムアウトを経過すると、アプリケーションは無効化されます。ロードバランサは、スティッキ要求などのこのアプリケーションに対する要求は受け付けません。

ロードバランサが参照するすべてのサーバーインスタンスまたはクラスタから、あるアプリケーションを無効にする場合、無効化されたアプリケーションのユーザーは、アプリケーションが再度有効化されるまでサービスを受けられません。

1 つのサーバーインスタンスまたはクラスタからアプリケーションを無効にして、別のサーバーインスタンスまたはクラスタでは有効にする場合、ユーザーは引き続きアプリケーションにアクセスできます。

アプリケーションを無効にするには、次の手順に従います。

1. `asadmin disable-http-lb-application` を実行して、タイムアウト (分単位)、無効にするアプリケーションの名前、および、無効化を実行するターゲットクラスタまたはインスタンスを指定します。
2. `asadmin export-http-lb-config` を使用して、ロードバランサ設定ファイルをエクスポートします。
3. エクスポートした設定を Web サーバーの `config` ディレクトリにコピーします。

HTTP および HTTPS セッションのフェイルオーバーの設定

HTTP/HTTPS セッションが接続されていた元のアプリケーションサーバーインスタンスが無効化された場合、ロードバランサプラグインは、そのセッションを別のアプリケーションサーバーインスタンスにフェイルオーバーします。この節では、HTTP/HTTPS ルーティングとセッションフェイルオーバーを有効にするようにロードバランサプラグインを設定する方法について説明します。

HTTP セッションの持続性の設定については、「可用性とセッション持続性の設定」を参照してください。

この節では、次の項目について説明します。

- HTTPS ルーティングについて
- HTTPS ルーティングの設定
- HTTP/HTTPS 要求のロードバランスにおける既知の問題

HTTPS ルーティングについて

HTTP または HTTPS のどちらでも、すべての着信要求は、ロードバランサプラグインによってアプリケーションサーバーインスタンスにルーティングされます。ただし、HTTPS ルーティングが有効になっている場合、ロードバランサプラグインは HTTPS ポートのみを使用して HTTPS 要求をアプリケーションサーバーに転送します。HTTPS ルーティングは、新しい要求とスティッキ要求の両方について実行されます。

HTTPS 要求が受信され、処理中のセッションがない場合、ロードバランサプラグインは設定されている HTTPS ポートを使用して使用可能なアプリケーションサーバーインスタンスを選択し、要求をそのインスタンスに転送します。

処理中の HTTP セッションで、同じセッションに対して新しい HTTPS 要求が受信された場合、HTTP セッション中に保存されたセッションおよびスティッキ情報を使用して HTTPS 要求がルーティングされます。新しい HTTPS 要求は、最後の HTTP 要求が処理された同じサーバーにルーティングされます。ただし、HTTPS ポートが使用されます。

HTTPS ルーティングの設定

`create-http-lb-config` コマンドの `httpsrouting` オプションは、ロードバランスに関わるすべてのアプリケーションサーバーに対して HTTPS ルーティングが有効か無効かを制御します。このオプションが false に設定されている場合、すべての HTTP および HTTPS 要求は HTTP として転送されます。新しいロードバランサ設定を作成する場合、または、作成後に `asadmin set` コマンドを使用して変更する場合には、このオプションを true に設定してください。

注

- HTTPS ルーティングを動作させるには、1 つまたは複数の HTTPS リスナーを設定する必要があります。
- `https-routing` が `true` に設定されていて、クラスタ内に正常な HTTPS リスナーが存在していない状態で新しい要求またはスティッキ要求が着信した場合、その要求はエラーを生成します。

HTTP/HTTPS 要求のロードバランスにおける既知の問題

次のリストでは、HTTP/HTTPS 要求の処理に関するロードバランサの制限について説明します。

- あるセッションが HTTP 要求と HTTPS 要求を組み合わせて使用する場合、最初の要求は必ず HTTP 要求にする必要があります。最初の要求が HTTPS 要求の場合、そのあと HTTP 要求を続けられません。これは、HTTPS セッションに関連付けられている Cookie がブラウザによって返されないからです。ブラウザは、異なる 2 つのプロトコルを異なる 2 つのサーバーと解釈し、新しいセッションを開始します。

  この制限は、`httpsrouting` が `true` に設定されている場合のみ有効です。

- あるセッションに HTTP 要求と HTTPS 要求の組み合わせが含まれる場合、アプリケーションサーバーインスタンスは HTTP リスナーと HTTPS リスナーの両方を使用して設定される必要があります。

  この制限は、`httpsrouting` が `true` に設定されている場合のみ有効です。

- あるセッションに HTTP 要求と HTTPS 要求の組み合わせが含まれる場合、アプリケーションサーバーインスタンスは、標準ポート番号、すなわち HTTP には 80、HTTPS には 443 を使用する HTTP および HTTPS リスナーによって設定される必要があります。

  この制限は、`httpsrouting` の値セットとは無関係に有効です。

べき等 URL の設定

配備されたアプリケーションの可用性を強化するには、ロードバランサによって処理されたすべてのアプリケーションサーバーインスタンスに、失敗したべき等の HTTP 要求を再試行する環境を設定します。このオプションは、検索要求の再試行など、読み取り専用の要求に使用されます。

べき等要求とは再試行時にアプリケーションに変更や不整合をもたらさないタイプの要求です。HTTP の場合、GET などの一部のメソッドはべき等的ですが、POST などその他のメソッドはそうではありません。べき等 URL の再試行では、サーバーまたはデータベースの値が変更されてはいけません。ユーザーが受信する応答が変更されるだけです。

べき等要求の例としては、検索エンジンクエリーやデータベースクエリーがあります。基礎となる原則は、再試行によってデータの更新や変更が発生しないことです。

`sun-web.xml` ファイルに、べき等 URL を設定します。ロードバランサ設定をエクスポートする場合、べき等 URL の情報は自動的に `loadbalancer.xml` ファイルに追加されます。

べき等 URL の設定の詳細については、『Developer's Guide』を参照してください。

HTML エラーページの設定

独自のエラーページやエラーページへの URL を指定して、それらをエンドユーザーに対して表示することができます。エラーページを指定すると、エラー報告に関して設定されているその他のメカニズムはすべて無効になります。

`sun-web.xml` ファイルに、HTML エラーページを設定します。ロードバランサ設定をエクスポートする場合、HTML エラーページの情報は自動的に `sun-web.xml` ファイルから `loadbalancer.xml` ファイルに追加されます。

HTML エラーページの設定の詳細については、『Developer's Guide』を参照してください。

HTTP ロードバランサプラグインの監視

ログメッセージの設定

ロードバランサプラグインは、Web サーバーのログメカニズムを使用してログメッセージを記録します。Application Server のデフォルトのログレベルは、Sun Java System Web Server (INFO)、Apache Web Server (WARN)、および Microsoft IIS (INFO) のデフォルトのログレベルに設定されています。アプリケーションサーバーのログレベル - FINE、FINER、および FINEST は、Web サーバーの DEBUG レベルに対応します。

これらのログメッセージは Web サーバーのログファイルに書き込まれます。これらは raw データ形式で、スクリプトを使用して解析されるかまたは表計算ドキュメントにインポートされて、必要なメトリックスを計算します。

ログメッセージのタイプ

ロードバランサプラグインは、次の 3 種類のログメッセージセットを生成します。

ロードバランサコンフィギュレータログメッセージ

これらのメッセージは、べき等 URL とエラーページ設定を使用している場合に記録されます。

べき等 URL のパターン設定の出力には、次の情報が含まれます。

- ログレベルが FINE に設定されている場合 :

  CONFxxxx: IdempotentUrlPattern によって、Web モジュール <web-module> の <url-pattern> に対する <no-of-retries> が設定されました

- ログレベルが SEVERE に設定されている場合 :

  CONFxxxx: loadbalancer.xml の Web モジュール <web-module> に対するべき等 URL 要素 <url-pattern> のエントリが重複しています

- ログレベルが WARN に設定されている場合 :

  CONFxxxx: Web モジュール <web-module> の IdempotentUrlPatternData <url-pattern> が無効です

エラーページの URL 設定の出力には、次の情報が含まれます (ログレベルが WARN に設定されている場合)。

CONFxxxx: Web モジュール <web-module> の error-url が無効です

要求ディスパッチおよび実行時ログメッセージ

これらのログメッセージは、要求が負荷分散およびディスパッチされている間に生成されます。

- 各メソッドの起動の標準的なログの出力には、次の情報が含まれます (ログレベルが FINE に設定されている場合)。

  ROUTxxxx: ルーターメソッド <method_name> を実行しています

- 各メソッドの起動のルーターログの出力には、次の情報が含まれます (ログレベルが INFO に設定されている場合)。

  ROUTxxxx: べき等要求 <Request-URL> に対する別の ServerInstance の選択に成功しました

- 実行時ログの出力には、次の情報が含まれます (ログレベルが INFO に設定されている場合)。

  RNTMxxxx: べき等の <GET/POST/HEAD> 要求 <Request-URL> を再試行しています

コンフィギュレータエラーメッセージ

これらのエラーは、参照先のカスタムエラーページがなくなっているなど、設定上の問題がある場合に表示されます。

- ログレベルが INFO に設定されている場合 :

  ROUTxxxx: 非べき等要求 <Request-URL> は、再試行されません

  例 : ROUTxxxx: 非べき等要求 http://sun.com/addToDB?x=11&abc=2 は、再試行されません

- ログレベルが FINE に設定されている場合 :

  RNTMxxxx: Web モジュール <web-module> に対するカスタムエラー URL またはページ <error-url> が、無効または不明です

例 : RNTMxxxx: Web モジュール test に対するカスタムエラー URL またはページ myerror1xyz が、無効または不明です

監視の設定

次の手順を実行して、ロードバランサプラグインログメッセージを有効にします。

1. Web サーバーのログオプションを設定します。
   a. Sun Java System Web Server の管理コンソールで、「Magnus Editor」タブを表示します。

      「Log Verbose」オプションを 「On」に設定します。

   b. Apache Web Server の場合は、ログレベルを DEBUG に設定します。

   c. Microsoft IIS の場合は、sun-passthrough.properties ファイルでログレベルを FINE に設定します。
2. ロードバランサ設定の「監視」オプションを true に設定します。

   asadmin create-http-lb-config コマンドを使用して最初にロードバランサ設定を作成する際に監視を true に設定するか、asadmin set コマンドを使用して後から true に設定します。デフォルトでは、監視は無効になっています。

ロードバランサプラグインは、次の情報をログに記録します。

- すべての要求の開始 / 停止情報。
- 要求が正常ではないインスタンスから正常なインスタンスにフェイルオーバーする際の、フェイルオーバー要求の情報。
- すべての診断プログラムサイクルの最後にある正常ではないインスタンスのリスト。

注 ロードバランサプラグインでログが有効になっていて、Web サーバーのログレベルが DEBUG かまたは verbose メッセージを出力するように設定されている場合、ロードバランサは Web サーバーのログファイルに HTTP セッション ID を記録します。したがって、ロードバランサプラグインをホストしている Web サーバーが DMZ 内にある場合、本稼動環境では DEBUG または同等のログレベルを使用しないでください。

ログレベル DEBUG を使用する必要がある場合は、loadbalancer.xml で require-monitor-data プロパティを false に設定して、ロードバランサのログを無効にしてください。

メッセージの監視について

ロードバランサプラグインのログメッセージの形式は、次のとおりです。

- HTTP 要求の開始時には、次の情報が含まれます。

  RequestStart スティッキ（新規）< 要求 ID> < タイムスタンプ > <URL>

  タイムスタンプ値には、1970 年 1 月 1 日からの時間をミリ秒単位で指定します。次に例を示します。

  ```
  RequestStart 新規 123456 602983
  http://austen.sun.com/Webapps-simple/servlet/Example1
  ```

- HTTP 要求の最後には、RequestExit メッセージが次のように表示されます。

  RequestStart スティッキ（新規）< 要求 ID> < タイムスタンプ > <URL> < リスナー ID> < 応答時間 >< エラーに関する障害の原因 >（障害発生時）

  次に例を示します。

  ```
  RequestExit 新規 123456 603001
  http://austen.sun.com/Webapps-simple/servlet/Example1
  http://austen:2222 18
  ```

  注 RequestExit メッセージでは、< 応答時間 > は、要求の合計ターンアラウンドタイムをロードバランサプラグインの側からミリ秒単位で表します。

- 正常ではないインスタンスのリストは、次のとおりです。

  UnhealthyInstances < クラスタ ID> < タイムスタンプ > < リスナー ID>, < リスナー ID>...

  次に例を示します。

  ```
  UnhealthyInstances cluster1 701923 http://austen:2210,
  http://austen:3010
  ```

- フェイルオーバー要求のリストは、次のとおりです。

  FailedoverRequest < 要求 ID> < タイムスタンプ > <URL> < セッション ID> < フェイルオーバーリスナー ID> < 正常でないリスナー ID>

  次に例を示します。

  ```
  FailedoverRequest 239496 705623
  http://austen.sun.com/Apps/servlet/SessionTest 16dfdac3c7e80a40
  http://austen:4044 http://austen:4045
  ```

アプリケーションのアップグレード

段階的アップグレードについて

ユーザーへのサービスを停止することなくアプリケーションをアップグレードするには、1 つのサーバーまたはクラスタ上にあるアプリケーションを一度にアップグレードします。クラスタはバージョンが混在している環境を透過的に維持しますが、ユーザーはアップグレードが行われていることに気づきません。このタイプのアップグレードを段階的アップグレードと呼びます。

段階的アップグレードは、旧新バージョンのアプリケーションに互換性があり、両方を同時に実行できる場合のみ可能です。セッション情報には互換性が必要です。混在モードの段階的アップグレードを、単一のスタンドアロンクラスタまたは複数のクラスタで実行します。

混在モードの環境での段階的アップグレードは、データベーススキーマの変更など、アプリケーションに大きな変更がある場合は、実行できません。そのような場合は、アップグレード中にアプリケーションを停止してください。アプリケーションの配備を取り消して、アップグレードしたアプリケーションを同じ名前で再配備します。

単一のスタンドアロンクラスタでのアップグレード

ほかのクラスタと設定を共有しないクラスタである単一のスタンドアロンクラスタでアプリケーションをアップグレードするには、次のようにします。

1. 旧バージョンのアプリケーションを保存するか、ドメインをバックアップします。

   ドメインをバックアップするには、`asadmin backup-domain` コマンドを使用します。

2. クラスタの動的再設定が有効の場合はオフにします。

   管理コンソールで、次の手順を実行します。

   a. 「設定」ノードを開きます。

   b. クラスタの設定の名前をクリックします。

   c. 「システムプロパティの設定」ページで、「動的再設定を有効」ボックスのチェックをはずします。

   d. 「保存」をクリックします。

   `asadmin` と同機能を持つコマンドは、`asadmin set` です。構文は次のとおりです。

```
asadmin set --user user --passwordfile password_file
cluster_name-config.dynamic-reconfiguration-enabled=false
```

3. ターゲットの domain に対して、アップグレードしたアプリケーションを再配備します。管理コンソールを使って再配備する場合、ドメインが自動的にターゲットになります。動的再設定が無効なので、旧アプリケーションがクラスタで実行し続けます。
4. asadmin enable-http-lb-application を使用して、インスタンスに対して再配備アプリケーションを有効にします。
5. asadmin disable-http-lb-server を使用して、1 つのサーバーインスタンスを無効にします。
6. asadmin export-http-lb-config を使用して、ロードバランサ設定ファイルをエクスポートします。
7. エクスポートした設定ファイルを Web サーバーインスタンスの設定ディレクトリにコピーします。たとえば、Sun Java System Web Server の場合、コピー先は *web_server_install_dir*/https-*host-name*/config/loadbalancer.xml となります。
8. タイムアウトが終了するまで、待機します。ロードバランサのログファイルを監視して、インスタンスがオフラインであることを確認します。
9. クラスタ内のほかのインスタンスが実行中の間に、無効になっていたサーバーインスタンスを再起動します。再起動により、サーバーはドメインと同期し、アプリケーションを更新します。
10. 再起動したサーバー上でアプリケーションをテストし、正しく動作していることを確認します。
11. asadmin enable-http-lb-server を使用して、サーバーインスタンスを有効にします。
12. asadmin export-http-lb-config を使用して、ロードバランサ設定ファイルをエクスポートします。
13. 設定ファイルを Web サーバーインスタンスの設定ディレクトリにコピーします。
14. クラスタ内の各インスタンスに対して、手順 5 ～手順 13 を繰り返します。
15. すべてのサーバーインスタンスに新しいアプリケーションがあり、それらのインスタンスが実行中である場合、再びクラスタに対して動的再設定を有効にします。

2 つのクラスタのアップグレード

1. 旧バージョンのアプリケーションを保存するか、ドメインをバックアップします。

   ドメインをバックアップするには、asadmin backup-domain コマンドを使用します。
2. 両方のクラスタの動的再設定が有効の場合はオフにします。

管理コンソールで、次の手順を実行します。

a. 「設定」ノードを開きます。

b. 1 つのクラスタの設定の名前をクリックします。

c. 「システムプロパティの設定」ページで、「動的再設定を有効」ボックスのチェックをはずします。

d. 「保存」をクリックします。

e. 2 番目のクラスタに対して上記手順を繰り返します。

`asadmin` と同機能を持つコマンドは、`asadmin set` です。構文は次のとおりです。

`asadmin set --user` *user* `--passwordfile` *password_file* *cluster_name*`-config.dynamic-reconfiguration-enabled=false`

3. ターゲットの `domain` に対して、アップグレードしたアプリケーションを再配備します。管理コンソールを使って再配備する場合、ドメインが自動的にターゲットになります。動的再設定が無効なので、旧アプリケーションがクラスタで実行し続けます。
4. `asadmin enable-http-lb-application` を使用して、クラスタに対して再配備したアプリケーションを有効にします。
5. `asadmin disable-http-lb-server` を使用して、ロードバランサから 1 つのクラスタを無効にします。
6. `asadmin export-http-lb-config` を使用して、ロードバランサ設定ファイルをエクスポートします。
7. エクスポートした設定ファイルを Web サーバーインスタンスの設定ディレクトリにコピーします。たとえば、Sun Java System Web Server の場合、コピー先は *web_server_install_dir*`/https-`*host-name*`/config/loadbalancer.xml` となります。
8. タイムアウトが終了するまで、待機します。ロードバランサのログファイルを監視して、クラスタがオフラインであることを確認します。
9. ほかのクラスタが実行中の間に、無効となっていたクラスタを再起動します。再起動により、クラスタはドメインと同期し、アプリケーションを更新します。
10. 再起動したクラスタ上でアプリケーションをテストし、正しく動作していることを確認します。
11. `asadmin enable-http-lb-server` を使用して、クラスタを有効にします。
12. `asadmin export-http-lb-config` を使用して、ロードバランサ設定ファイルをエクスポートします。
13. 設定ファイルを Web サーバーインスタンスの設定ディレクトリにコピーします。
14. ほかのクラスタ内に対して、手順 5 〜手順 13 を繰り返します。

15. すべてのサーバーインスタンスに新しいアプリケーションがあり、それらのインスタンスが実行中である場合、再び両方のクラスタに対して動的再設定を有効にします。

RMI-IIOP ロードバランスとフェイルオーバーについて

RMI-IIOP ロードバランスとフェイルオーバーに対する要件

Sun Javaa System Application Server は、RMI-IIOP を使用して、リモート EJB 参照とネームサービスオブジェクトの高可用性を提供します。これらの機能を使用する前に、環境が次の要件を満たす必要があります。

- Sun Java System Application Server Enterprise Edition がインストールされていること。
- 2 つ以上のアプリケーションサーバーインスタンスに 1 つのクラスタが存在していること。

  詳細については、「クラスタの設定」を参照してください。
- J2EE アプリケーションが、ロードバランスに関わるすべてのアプリケーションサーバーインスタンスとクラスタに対して配備されていること。
- RMI-IIOP クライアントアプリケーションで、ロードバランスが有効であること。

  ロードバランスは、次の RMI-IIOP クライアントでサポートされます。

  - アプリケーションサーバーインスタンスに配備されている EJB にアクセスするアプリケーションクライアントコンテナ (ACC) で実行する Java アプリケーション。
  - ACC で実行するのではなく、アプリケーションサーバーインスタンスに配備されている EJB にアクセスする Java アプリケーション。

  RMI-IIOP ベースのアプリケーションを有効にするのに必要な設定は、クライアントのタイプに依存します。ロードバランスに対する RMI-IIOP クライアントアプリケーションの設定の詳細については、『Sun Java System Application Server Developer's Guide』を参照してください。

RMI-IIOP フェイルオーバーおよびロードバランスの詳細については、『Sun Java System Application Server High Availability Administration Guide』を参照してください。

注 SSL を使用した RMI-IIOP フェイルオーバーはサポートされません。

RMI-IIOP ロードバランスおよびフェイルオーバーアルゴリズムについて

Sun Java System Application Server は、RMI-IIOP パス上のリモート EJB 参照およびネームサービスオブジェクトのロードバランスに、ランダム化およびラウンドロビンアルゴリズムを採用します。

RMI-IIOP クライアントは最初に新しい `InitialContext` オブジェクトを作成すると、そのクライアントで利用可能な Application Server IIOP エンドポイントのリストが、ランダムに選ばれます。その `InitialContext` オブジェクトに対して、ロードバランサは、検索要求とほかの `InitialContext` 操作を、リストの最初のエンドポイントに命令します。最初のエンドポイントが利用できない場合、リストの 2 番目のエンドポイントが使用され、以下同様です。

クライアントが続けて新しい `InitialContext` オブジェクトを作成するたびに、エンドポイントリストがローテーションし、異なる IIOP エンドポイントが `InitialContext` 操作で使われます。

`InitialContext` オブジェクトによって確保される参照から Beans を入手または作成する場合、それらの Beans は、`InitialContext` オブジェクトに割り当てられた IIOP エンドポイントを処理する Application Server インスタンスで作成されます。それらの Beans に対する参照には、クラスタ内のすべての Application Server インスタンスの IIOP エンドポイントアドレスが含まれます。

プライマリエンドポイントは、Bean を検索または作成するのに使用される `InitialContext` エンドポイントに対応する Bean エンドポイントです。クラスタ内のほかの IIOP エンドポイントは、代替エンドポイントとして指定されています。Bean のプライマリエンドポイントが利用できなくなると、その Bean での追加の要求は、代替エンドポイントの 1 つにフェイルオーバーされます。

RMI-IIOP のサンプルアプリケーション

次のディレクトリには、ACC とともに、または ACC なしで RMI-IIOP フェイルオーバーを使用する方法を示すサンプルアプリケーションが含まれています。

install_dir/samples/ee-samples/failover/apps/sfsbfailover

ACC とともに、または ACC なしでアプリケーションを実行する方法については、このサンプルに付属している index.html ファイルを参照してください。ee-samples ディレクトリには、サンプルを実行するための環境の設定方法に関する情報も含まれます。

第 4 章

ノードエージェントの設定

この章では、Application Server のノードエージェント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。

- ノードエージェントについて
- ノードエージェントに関する管理コンソールタスク
- asadmin ツールのノードエージェント関連タスク

ノードエージェントについて

- ノードエージェント
- ノードエージェントのプレースホルダ
- ノードエージェントの配備
- ノードエージェントとドメイン管理サーバーとの同期化
- ノードエージェントログの表示
- 管理コンソールと asadmin ツールから利用可能なタスク

ノードエージェント

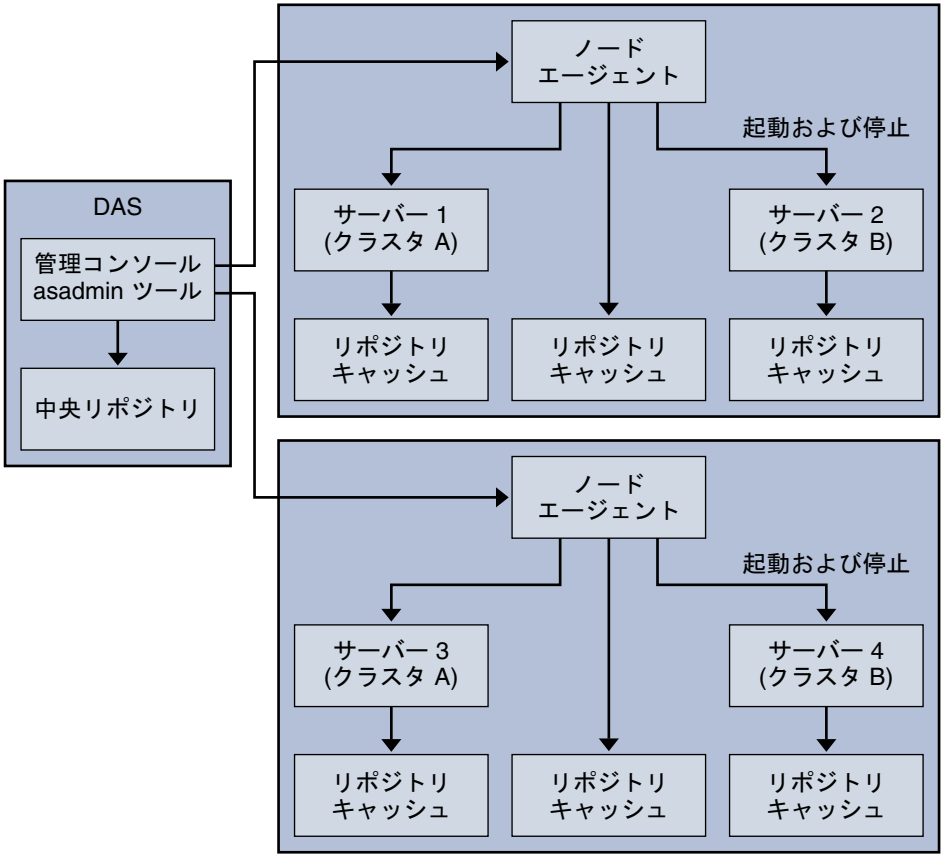
ノードエージェントは、ドメイン管理サーバー (DAS) をホストするマシンを含む、サーバーインスタンスをホストするすべてのマシンに必要な軽量エージェントです。ノードエージェントは次の機能を実行します。

- ドメイン管理サーバーの指示により、サーバーインスタンスの起動、停止、作成、または削除を行います。
- 障害の発生したサーバーインスタンスを再起動します。

- 障害の発生したサーバーのログファイルを表示します。
- 各サーバーインスタンスのローカル設定リポジトリとドメイン管理サーバーの中央リポジトリを同期化します。各ローカルリポジトリには、そのサーバーインスタンスまたはノードエージェントに関する情報のみが含まれます。

次の図は、ノードエージェントの全体的なアーキテクチャを示しています。

図 4-1 ノードエージェントのアーキテクチャ



ノードエージェントの自動作成

Application Server をインストールすると、マシンのホスト名を持つノードエージェントがデフォルトで作成されます。このノードエージェントは、実行する前に、ローカルマシン上で手動で起動する必要があります。

ノードエージェントとサーバーインスタンスの管理

ノードエージェントを実行していない場合でも、サーバーインスタンスを作成および管理できます。ただし、ノードエージェントを使用してサーバーインスタンスを起動および停止するには、ノードエージェントが実行中である必要があります。

ノードエージェントを停止すると、ノードエージェントが管理するサーバーインスタンスも停止します。

追加ノードエージェント

ノードエージェントは 1 つのドメインを処理します。マシンが複数のドメインで実行されるインスタンスをホストする場合は、複数のノードエージェントを実行する必要があります。

ノードエージェントのプレースホルダ

プレースホルダを使用するノードエージェントが存在しなくても、サーバーインスタンスを作成、削除することは可能です。プレースホルダは、ノードエージェント自体がノードエージェントのローカルシステムに作成される前に、ドメイン管理サーバー (DAS) 上で作成されたノードエージェントの設定です。

注 プレースホルダノードエージェントを作成すると、それを使用してドメインにインスタンスを作成できます。ただし、インスタンスを起動する前に、`asadmin` コマンドを使用して、インスタンスが配置されるマシン上に実際のノードエージェントをローカルに作成し、起動する必要があります。詳細については、114 ページの「ノードエージェントの作成」と、115 ページの「ノードエージェントの起動」を参照してください。

ノードエージェントの配備

ノードエージェントは次の 2 つの方法のどちらかを使用して、設定し、配備します。

- オンライン配備 : 設定前に、トポロジが構成されていて、ドメイン用のハードウェアが用意されている場合。
- オフライン配備 : 完全な環境を設定する前に、ドメインとサーバーインスタンスを設定する場合。

ノードエージェントの配備の前に

ノードエージェントを配備する前に、次を実行します。

1. ドメイン管理サーバーをインストールします。
2. ドメイン管理サーバーを起動します。

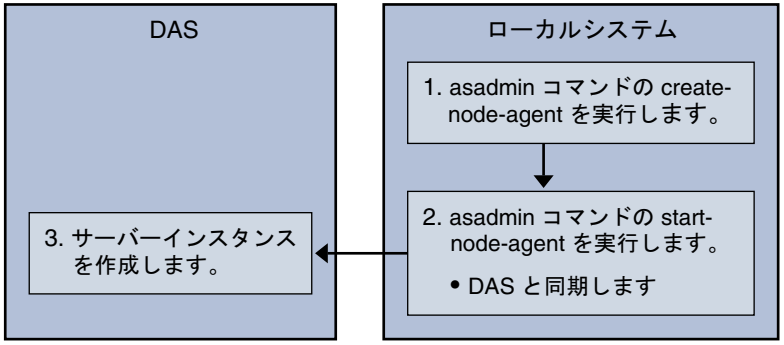
ドメイン管理サーバーが起動し、実行中になったら、オンラインまたはオフライン配備を開始します。

オンライン配備

設定を開始する前に、トポロジが構成されていて、ドメイン用のハードウェアが用意されている場合にドメインを設定するには、オンライン配備を使用します。

次の図は、ノードエージェントのオンライン配備の概要を示しています。

図 4-2 ノードエージェントのオンライン配備



オンライン配備を設定するには、次を実行します。

1. サーバーインスタンスをホストするすべてのマシンにノードエージェントをインストールします。

インストールプログラム、または asadmin コマンドの create-node-agent を使用します。マシンに複数のエージェントが必要な場合は、asadmin コマンドの create-node-agent を使用してエージェントを作成します。

詳細については、114 ページの「ノードエージェントの作成」を参照してください。

2. asadmin コマンドの start-node-agent を使用して、ノードエージェントを起動します。

   起動すると、ノードエージェントはドメイン管理サーバー (DAS) と通信します。それが DAS に到達すると、DAS にノードエージェントに対する設定が作成されます。設定が作成されると、管理コンソールでノードエージェントを表示できます。

   詳細については、115 ページの「ノードエージェントの起動」を参照してください。

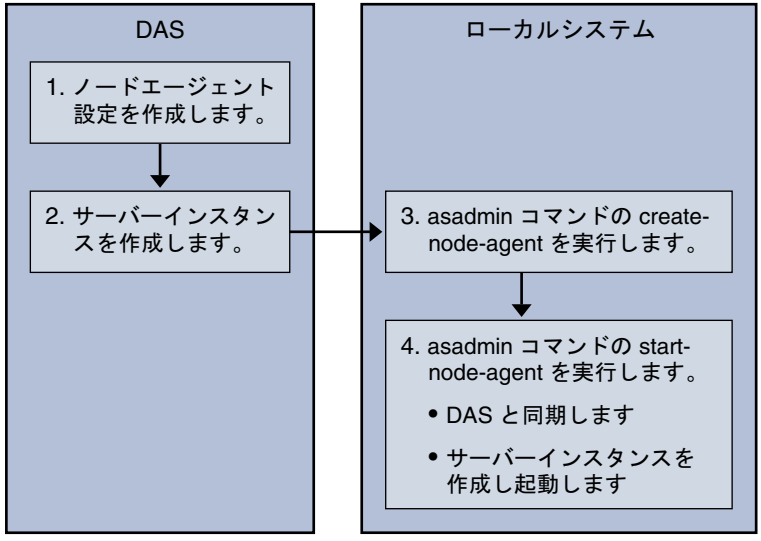
3. サーバーインスタンスとクラスタの作成、アプリケーションの配備を含むドメインを設定します。

オフライン配備

オフラインアプローチによって、個々のローカルマシンを設定する前に、ドメインの項目を簡単に定義および再編成できます。

次の図は、オフライン配備手順の概要を示しています。

図 4-3　ノードエージェントのオフライン配備



ローカルマシンでノードエージェントを設定する前に、ドメインとサーバーインスタンスを設定するには、次の手順を実行します (オフライン設定)。

1. ドメイン管理サーバーにプレースホルダノードエージェントを作成します。

   詳細については、110 ページの「ノードエージェントのプレースホルダの作成」を参照してください。

2. サーバーインスタンスとクラスタを作成して、アプリケーションを配備します。
3. サーバーインスタンスをホストするすべてのマシンにノードエージェントをインストールします。

   インストールプログラム、または `asadmin` コマンドの `create-node-agent` を使用します。ノードエージェントには、以前に作成したプレースホルダノードエージェントと同じ名前を付ける必要があります。

   詳細については、114 ページの「ノードエージェントの作成」を参照してください。

4. `asadmin` コマンドの `start-node-agent` を使用して、ノードエージェントを起動します。

   ノードエージェントが起動すると、ドメイン管理サーバーにバインドされ、以前にノードエージェントに関連付けられたサーバーインスタンスを作成します。

   詳細については、115 ページの「ノードエージェントの起動」を参照してください。

ノードエージェントとドメイン管理サーバーとの同期化

設定データがドメイン管理サーバーのリポジトリ (中央リポジトリ) に格納されると同時にノードエージェントのローカルマシンにキャッシュされるため、これら 2 つは同期する必要があります。

ノードエージェントの同期化

初めてノードエージェントが起動すると、中央リポジトリの最新情報の要求をドメイン管理サーバー (DAS) に送信します。ノードエージェントが DAS に正常に接続され、設定情報を取得すると、ノードエージェントは DAS にバインドされます。

DAS にプレースホルダノードエージェントを作成した場合、ノードエージェントが初めて起動するときに、ノードエージェントは DAS の中央リポジトリから設定を取得します。

最初の起動時に、DAS が実行されていないため、ノードエージェントが DAS に到達できない場合、ノードエージェントは停止し、バインドされないままの状態になります。

ドメインのノードエージェントの設定が変更された場合、ノードエージェントを実行するローカルマシンのノードエージェントと自動的に通信します。

DAS のノードエージェント設定を削除すると、次に同期するときにノードエージェント自体が停止し、削除待ちとしてマーク付けされます。ローカルの asadmin コマンドの delete-node-agent を使用して、ノードエージェントを手動で削除してください。

サーバーインスタンスの同期化

管理コンソールまたは asadmin ツールを使用して、サーバーインスタンスを明示的に起動する場合、サーバーインスタンスは中央リポジトリと同期します。この同期が失敗すると、サーバーインスタンスは起動しません。

ノードエージェントが、管理コンソールまたは asadmin ツールによる明示的な要求なしにサーバーインスタンスを起動する場合、サーバーインスタンスのリポジトリキャッシュは同期しません。サーバーインスタンスは、キャッシュに格納された設定によって実行されます。

大きなアプリケーションの同期化

同期化の必要な大きなアプリケーションが使用環境に含まれる場合、または使用できるメモリが制限されている場合は、JVM オプションを調整してメモリの使用を制限できます。この調整によって、メモリ不足によるエラーを受信する可能性は低くなります。インスタンス同期化 JVM ではデフォルトの設定が使用されますが、JVM オプションを設定してそれらを変更することもできます。

INSTANCE-SYNC-JVM-OPTIONS プロパティを使用して、JVM オプションを設定します。このプロパティを設定するコマンドは次のとおりです。

```
asadmin set
domain.node-agent.node_agent_name.property.INSTANCE-SYNC-JVM-OPTIONS="JVM_options"
```

次に例を示します。

```
asadmin set domain.node-agent.node0.property.INSTANCE-SYNC-JVM-OPTIONS="-Xmx32m
-Xss2m"
```

この例では、ノードエージェントは node0、JVM オプションは -Xmx32m -Xss2m です。

JVM オプションの詳細については、次を参照してください。

http://java.sun.com/docs/hotspot/VMOptions.html

注 ノードエージェントの設定にプロパティが追加されたり変更されてもノードエージェントは自動的に同期化されないため、INSTANCE-SYNC-JVM-OPTIONS プロパティの変更後、ノードエージェントを再起動してください。

ノードエージェントログの表示

各ノードエージェントは固有のログファイルを持っています。ノードエージェント関連の問題がある場合、次の場所にあるログファイルを参照します。

node_agent_dir/*node_agent_name*/agent/logs/server.log.

ノードエージェントログが、サーバーのログを参照して問題に関する詳細なメッセージを調べるように指示する場合もあります。

サーバーログの場所は以下のとおりです。

node_agent_dir/*node_agent_name*/*server_name*/logs/server.log

node_agent_dir のデフォルトの位置は *domain_root_dir/../nodeagents* です。

管理コンソールと asadmin ツールから利用可能なタスク

ノードエージェントについては、ノードエージェントを実行するシステムで一部のタスクをローカルに実施する必要がありますが、その他はドメイン管理サーバーで実施できます。ローカルに実施する必要があるタスクは、ノードエージェントが存在するマシンで実行する asadmin ツールからのみ利用できます。ドメイン管理サーバーで機能するタスクは、管理コンソールと asadmin ツールから利用できます。

次の表は、タスクとそれを実行する場所の概要です。

表 4-1 管理コンソールと asadmin コマンドから利用可能なタスク

タスク	管理コンソール	asadmi コマンド
ノードエージェントのプレースホルダおよび設定をドメイン管理サーバーに作成します。	「現在のノードエージェントのプレイスホルダ」ページ。	create-node-agent-config
ノードエージェントを作成します。	利用不可。	create-node-agent
ノードエージェントを起動します。	利用不可。	start-node-agent
ノードエージェントを停止します。	利用不可。	stop-node agent

表 4-1　管理コンソールと asadmin コマンドから利用可能なタスク (続き)

タスク	管理コンソール	asadmi コマンド
ドメイン管理サーバーからノードエージェント設定を削除します。	「ノードエージェント」ページ。	delete-node-agent-config
ローカルマシンからノードエージェントを削除します。	利用不可。	delete-node-agent
ノードエージェント設定を編集します。	「ノードエージェント」ページ。	set
ノードエージェントを一覧表示します。	「ノードエージェント」ページ。	list-node-agents

ノードエージェントに関する管理コンソールタスク

一般ノードエージェント情報の表示

エージェントまたはノードエージェントのプレースホルダが作成されてから、その設定を表示するには次を実行します。

1. ツリーコンポーネントで、「ノードエージェント」ノードを選択します。
2. ノードエージェントの名前をクリックします。

   ノードエージェントが既に存在するのに、ここに表示されない場合は、ノードエージェントのホストマシンで、`asadmin start-node-agent` を使用して、ノードエージェントを起動します。詳細については、115 ページの「ノードエージェントの起動」を参照してください。
3. ノードエージェントのホスト名をチェックします。

   ホスト名が「不明なホスト」の場合、ノードエージェントはドメイン管理サーバー (DAS) と初期接続をしていません。

4. ノードエージェントの状態をチェックします。

   「稼動中」：ノードエージェントが正常に作成され、現在実行中です。

   「停止中」：次の状況のどちらかです。

   - ノードエージェントはローカルマシンで作成されているが、起動していない。
   - ノードエージェントは起動したが、その後停止した。

   「ランデブーを待機しています」：ノードエージェントは、ローカルマシンで作成されていないプレースホルダです。

   ノードエージェントの作成と起動については、114 ページの「ノードエージェントの作成」と、115 ページの「ノードエージェントの起動」を参照してください。

5. 起動時にインスタンスを起動するかどうかを選択します。

   ノードエージェントが起動するときに、ノードエージェントに関連するサーバーインスタンスが自動的に起動するようにするには「Yes」を選択します。インスタンスを手動で起動するには、「No」を選択します。

6. ノードエージェントがドメイン管理サーバーと接続したかどうかを確認します。

   ノードエージェントがドメイン管理サーバーと接続していない場合、正常に起動していません。

7. ノードエージェントに関連するサーバーインスタンスを管理します。

   ノードエージェントが実行中の場合、インスタンス名の横にあるチェックボックスをクリックし、「起動」または「停止」をクリックしてインスタンスを起動または停止します。

ノードエージェントのプレースホルダの作成

ノードエージェントは、ノードエージェントをホスティングするマシン上にローカルに作成する必要があるため、ノードエージェントのプレースホルダは、管理コンソールを使用する場合にだけ作成できます。このプレースホルダは、ノードエージェントが存在しない場合のノードエージェントの設定です。

プレースホルダを作成したら、ノードエージェントをホスティングするマシン上で、`asadmin` コマンドの `create-node-agent` を使用して作成を完了します。詳細については、114 ページの「ノードエージェントの作成」を参照してください。

1. ツリーコンポーネントで、「ノードエージェント」ノードを選択します。
2. 「ノードエージェント」ページで、「新規」をクリックします。
3. 「現在のノードエージェントプレイスホルダ」ページで、新規ノードエージェントの名前を入力します。

名前は、ドメインのすべてのノードエージェント名、サーバーインスタンス名、クラスタ名、および設定名の間で一意である必要があります。

4. 「了解」をクリックします。

   新規ノードエージェントのプレースホルダが「ノードエージェント」ページにリスト表示されます。

同機能を持つ asadmin コマンド : create-node-agent-config

ノードエージェントの設定の削除

管理コンソールを使用する場合にだけ、ノードエージェントの設定をドメインから削除できます。実際のノードエージェントは削除できません。ノードエージェント自体を削除するには、ノードエージェントのローカルマシンで asadmin コマンドの delete-node-agent を実行します。詳細については、116 ページの「ノードエージェントの削除」を参照してください。

ノードエージェントの設定を削除する前に、ノードエージェントの実行を停止し、関連するインスタンスを破棄する必要があります。ノードエージェントを停止するには、asadmin コマンドの stop-node-agent を使用します。詳細については、116 ページの「ノードエージェントの停止」を参照してください。

1. ツリーコンポーネントで、「ノードエージェント」ノードを選択します。
2. 「ノードエージェント」ページで、削除するノードエージェントの横にあるチェックボックスを選択します。
3. 「削除」をクリックします。

同機能を持つ asadmin コマンド : delete-node-agent-config

ノードエージェントの設定の編集

ノードエージェントの設定を編集するには、次の手順を実行します。

1. ツリーコンポーネントで、「ノードエージェント」ノードを開きます。
2. 編集するノードエージェントの設定を選択します。
3. 「ノードエージェントの一般情報」ページで、エージェントの起動時にエージェントのサーバーインスタンスを起動するかどうかを選択します。このページから、手動でのインスタンスの起動または停止もできます。

この設定がプレースホルダノードエージェント用である場合、`asadmin create-node-agent` を使用して実際のノードエージェントを作成するときに、この設定が引き継がれます。ノードエージェントの作成に関する情報は、114 ページの「ノードエージェントの作成」を参照してください。

この設定が既存のノードエージェント用である場合、ノードエージェントの設定情報が自動的に同期されます。

ノードエージェントのレルムの編集

ノードエージェントに接続されたユーザーの認証レルムを設定します。管理ユーザーだけがノードエージェントにアクセスできます。

1. ツリーコンポーネントで、「ノードエージェント」ノードを開きます。
2. 編集するノードエージェントの設定を選択します。
3. 「認証レルム」タブをクリックします。
4. 「ノードエージェントのレルムの編集」ページで、レルムを入力します。

   デフォルトは、ノードエージェントの作成時に作成された `admin-realm` です。別のレルムを使用するには、ドメインによって制御されるすべてのコンポーネントまたは正常に通信しないコンポーネントのレルムを置き換えます。
5. 「クラス名」フィールドで、レルムを実装する Java クラスを指定します。
6. 必要なプロパティを追加します。

   認証レルムは、特定の実装によって必要とするものが異なるプロバイダ固有のプロパティが必要です。

ノードエージェントの JMX 対応リスナーの編集

ノードエージェントは、JMX を使用してドメイン管理サーバーと通信します。このため、JMX 要求とその他のリスナー情報を待機するポートが必要です。

1. ツリーコンポーネントで、「ノードエージェント」ノードを開きます。
2. 編集するノードエージェントの設定を選択します。
3. 「JMX」タブをクリックします。
4. 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、「アドレス」フィールドに「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。
5. 「ポート」フィールドで、待機するノードエージェントの JMX コネクタのポート番号を入力します。IP アドレスが「0.0.0.0」の場合、ポート番号は一意のものである必要があります。
6. 「JMX プロトコル」フィールドで、JMX コネクタがサポートするプロトコルを入力します。

   デフォルトは rmi_jrmp です。
7. 「すべてのアドレスを許可」の横にあるチェックボックスをクリックして、すべての IP アドレスに接続できるようにします。

   ノードエージェントは、ネットワークカードに関連付けられた特定の IP アドレスを待機するか、またはすべての IP アドレスを待機します。すべてのアドレスを許可すると、「待機するホストアドレス」プロパティに値「0.0.0.0」が設定されます。
8. 「レルム名」フィールドで、リスナーの認証を処理するレルムの名前を入力します。

このページの「セキュリティ」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を使用するように設定します。

安全なリスナーを設定するには、次の手順を実行します。

1. 「セキュリティ」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。

   デフォルトで、セキュリティが有効になります。
2. 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。
3. 「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」の章を参照してください。
4. 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、または両方のプロトコルにチェックマークを付ける必要があります。

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。

5. 「保存」をクリックします。

asadmin ツールのノードエージェント関連タスク

ノードエージェントの作成

ノードエージェントを作成するには、ノードエージェントを実行するマシンで `asadmin` コマンドの `create-node-agent` をローカルに実行します。

次に例を示します。

```
$ asadmin create-node-agent --host myhost --port 4849 ---user admin
nodeagent1
```

ここで、`myhost` はドメイン管理サーバー (DAS) ホスト名、`4849` は DAS ポート番号、`admin` は DAS ユーザー、および `nodeagent1` は、作成しているノードエージェントの名前です。

ノードエージェントのデフォルト名は、ノードエージェントを作成するホストの名前です。

ノードエージェントのプレースホルダをすでに作成している場合、ノードエージェントプレースホルダと同じ名前を使用して、関連したノードエージェントを作成します。ノードエージェントのプレースホルダを作成しておらず、DAS が起動していて到達可能である場合、`create-node-agent` コマンドは DAS 上にノードエージェント設定 (プレースホルダ) も作成します。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

注

次の場合は、DNS に到達可能なホスト名を指定する必要があります。

1. ドメインがサブネットの境界を超える場合。つまり、ノードエージェントとドメイン管理サーバー (DAS) が sun.com と java.com などの異なるドメインにある場合
2. DNS に登録されていないホスト名を持つ DHCP マシンを使用している場合

ドメインおよびノードエージェントの作成時に、次のとおりドメインおよびノードエージェントのホスト名を明示的に指定して、DNS に到達可能なホスト名を指定します。

```
create-domain --domainproperties domain.hostName=DAS-host-name
```

```
create-node-agent --host DAS-host-name --agentproperties
remoteclientaddress=node-agent-host-name
```

別の解決法は、プラットフォームに特定のホスト名およびアドレス解決を定義する、hosts ファイルを更新し、ホスト名を正しい IP アドレスに解決することです。ただし、DHCP 使用して再接続する時に、異なる IP アドレスを割り当てられる可能性があります。その場合、各サーバーでホスト解決ファイルを更新する必要があります。

ノードエージェントの起動

ノードエージェントがサーバーインスタンスを管理するには、ノードエージェントを実行する必要があります。ノードエージェントを起動するには、ノードエージェントが存在するシステムで asadmin コマンドの start-node-agent をローカルに実行します。

次に例を示します。

```
$ asadmin start-node-agent --user admin nodeagent1
```

ここで、admin は管理ユーザー、nodeagent1 は起動しているノードエージェントです。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

ノードエージェントの停止

ノードエージェントを停止するには、ノードエージェントが存在するシステムで asadmin コマンドの stop-node-agent をローカルに実行します。stop-node-agent は、ノードエージェントが管理するすべてのサーバーインスタンスを停止します。

次に例を示します。

```
$ asadmin stop-node-agent nodeagent1
```

ここで、nodeagent1 はノードエージェントです。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

ノードエージェントの削除

ノードエージェントを削除するには、ノードエージェントが存在するシステムで asadmin コマンドの delete-node-agent をローカルに実行します。

ノードエージェントを削除する前に、ノードエージェントを停止する必要があります。ノードエージェントが起動しない場合、またはドメイン管理サーバーに正常に接続できない (バインドされない) 場合も、ノードエージェントを削除できます。

次に例を示します。

```
$ asadmin delete-node-agent nodeagent1
```

ここで、nodeagent1 はノードエージェントです。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

ノードエージェントを削除する際には、管理コンソールまたは asadmin delete-node-agent-config コマンドのいずれかを使用して、ドメイン管理サーバーからノードエージェントの設定も削除する必要があります。

第 5 章

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の 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、手順 3 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、94 ページの「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、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クラスとともに、サーブレットおよび JSP などの Web コンポーネントから構成されます。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: JAR ファイルには、RMI/IIOP を使用して Enterprise JavaBeans など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。
- Enterprise 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する 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、`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の使用をお勧めします。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は、Sun Java System Application Server だけでなく、ほかの J2EE アプリケーションサーバーにも適用されます。

EJB の JNDI ルックアップ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の配備

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。

エンタープライズアプリケーションを配備 (インストール) 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、「アプリケーション」ノードを開きます。
2. 「Enterprise アプリケーション」ノードを選択します。
3. 「Enterprise 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、アプリケーションサーバー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。

a. 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 を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。

b. 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. 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。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。

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。ターゲットは、クラスタまたはスタンドアロンサーバーインスタンスです。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、94 ページの「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、`client.jar` 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。

同機能を持つ `asadmin` コマンド : `deploy`

Web アプリケーションの配備

Web アプリケーションは、サーブレットや JSP ページ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ます。

Web アプリケーションを配備 (インストール) 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、「アプリケーション」ノードを開きます。
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、アプリケーションサーバー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。

a. 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 を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。

b. 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. 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。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。

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。ターゲットは、クラスタまたはスタンドアロンサーバーインスタンスです。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、94 ページの「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、`client.jar` 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。

同機能を持つ `asadmin` コマンド : `deploy`

配備されている Web アプリケーションの起動

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。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
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。

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。

EJB モジュールの配備

EJB JAR ファイルとも呼ばれる EJB モジュールには、Enterprise JavaBeans が含まれています。

EJB モジュールを配備 (インストール) 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、「アプリケーション」ノードを開きます。
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、アプリケーションサーバー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。

   a. 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 を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。

   b. 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. 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。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。

   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、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。ターゲットは、クラスタまたはスタンドアロンサーバーインスタンスです。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、94 ページの「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、`client.jar` 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。

同機能を持つ `asadmin` コマンド : `deploy`

コネクタモジュールの配備

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。

コネクタモジュールを配備 (インストール) 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、「アプリケーション」ノードを開きます。
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、アプリケーションサーバー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。

   a. 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 を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。

   b. 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、Sun Java System 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. 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。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。

   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、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ってください。

f. リソースアダプタに追加プロパティ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。ターゲットは、クラスタまたはスタンドアロンサーバーインスタンスです。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、94 ページの「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。

同機能を持つ `asadmin` コマンド : `deploy`

ライフサイクルモジュールの作成

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起動されると、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、Sun Java System Application Server の拡張です。

ライフサイクルモジュールを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
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、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_root_dir*/*domains*/*domain*/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。

      この整数が、サーバーの起動時にライフサイクルモジュールがロードされる順番を決定します。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ロードされます。デフォルトでは、ロードが失敗した場合も、サーバーは起動操作を継続します。ロード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. 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。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。

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。ターゲットは、クラスタまたはスタンドアロンサーバーインスタンスです。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、94 ページの「段階的アップグレードについて」を参照してください。

5. 「了解」をクリックします。

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の配備

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。

アプリケーションクライアントモジュールを配備 (インストール) 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、「アプリケーション」ノードを開きます。
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、アプリケーションサーバー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。

a. 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 を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。

b. 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、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。ターゲットは、クラスタまたはスタンドアロンサーバーインスタンスです。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、94 ページの「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、`client.jar` 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。

クライアントサイド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、クライアントサイドルーチンを含む 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、クライアントサイドスタブを取得できます。

同機能を持つ `asadmin` コマンド : `deploy`

アプリケーションの一覧表示、配備の取り消し、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の一覧表示

配備されているアプリケーションを一覧表示するには、次の手順に従います。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
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開きます。

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。

同機能を持つ `asadmin` コマンド : `list-components`

サブコンポーネントのリスト

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。

アプリケーションまたはモジュールのサブコンポーネントを一覧するには、次の手順に従います。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
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開きます。
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。

同機能を持つ asadmin コマンド : `list-components`

配備されているアプリケーションのモジュール記述子の表示

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では、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」ノードを開きます。
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。

アプリケーションの配備取り消し

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。

アプリケーションまたはモジュールの配備を取り消すには、次の手順に従います。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
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。

同機能を持つ asadmin コマンド : undeploy

アプリケーションの有効化と無効化

配備されているアプリケーションやモジュールが有効の場合、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。

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にするには、次の手順に従います。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
2. アプリケーションタイプのノードを開きます。
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。

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。

1. ツリーコンポーネントで、「アプリケーション」ノードを開きます。
2. アプリケーションタイプのノードを開きます。
3. アプリケーションのノードを選択します。
4. 「ターゲット」タブをクリックします。

5. 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。
6. 「有効」または「無効」を選択します。

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の管理

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」ノードを開きます。
2. アプリケーションタイプのノードを開きます。
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。

同機能を持つ asadmin コマンド : create-application-ref、delete-application-ref

追加の仮想サーバーへの配備

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られます。

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

本稼動環境では、94 ページの「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。

動的再読み込みの有効化と無効化

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_root_dir/domain1/applications/j2ee-modules_or_j2ee-apps/app_or_module_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。

```
AppServer/domain/domain1/applications/j2ee-module/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。

動的再読み込みを設定するには、次の手順に従います。

1. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを開きます。
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。

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、ファイルの日付を変更すると (たとえば、UNIX の `touch` コマンドを使用)、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

自動配備の使用

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は *domain_root_dir*/*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_root_dir*/`domain1/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。

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。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 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。

自動配備機能を設定するには、次の手順に従います。

1. ツリーコンポーネントで、「スタンドアロンインスタンス」ノードを開きます。
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、「ドメイン属性の設定」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。
   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。

ディレクトリからのパッケージ化されていないアプリケーションの配備

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、137 ページの「動的再読み込みの有効化と無効化」を参照してください。

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には、次の手順に従います。

1. 管理コンソールで、配備プロセスを開始します。123 ページの「Web アプリケーションの配備」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。

同機能を持つ `asadmin` コマンド : `deploydir`

deploytool ユーティリティの使用

開発者を対象として設計された `deploytool` ユーティリティは、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、『The 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 Developer's Guide』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 ファイルがルートとして配置されています。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、sun-cmp-mappings.xml ファイルと .dbschema ファイルがルートレベルに含まれます。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 6 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設定の一般手順

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』の「詳細情報」の節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。「JDBC ドライバの統合」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。「JDBC 接続プールの作成」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。「JDBC リソースの作成」を参照してください。

JDBC ドライバの統合

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
  - ドライバの JAR および ZIP ファイルを *domain_root_dir*/*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_root_dir*/*domain_dir*/`lib/ext` ディレクトリにコピーします。
  - 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
  - 管理コンソールのツリービューで、Application Server を選択します。
  - default-config など、希望する設定を選択します。
  - 「JVM 設定」を選択します。
  - 「JVM 設定」ページで、「パス設定」タブをクリックします。
  - 「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。
  - 「保存」をクリックします。
  - サーバーを再起動します。

JDBC 接続プールについて

JDBC 接続プールの作成

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。

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。

「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に限る)
  `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

JDBC 接続プールを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開きます。
2. 「リソース」ノードで、「JDBC」ノードを開きます。
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最も便利です。「JDBC 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」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、必要なプロパティを追加します。

10.「完了」をクリックします。

同機能を持つ asadmin コマンド : `create-jdbc-connection-pool`

JDBC 接続プールの編集

「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。

「コネクタ接続プールを編集」ページにアクセスするには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開きます。
2. 「リソース」ノードで、「JDBC」ノードを開きます。
3. 「JDBC」ノードで、「接続プール」ノードを開きます。
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。

   変更可能な設定の詳細については、次の各節を参照してください。
6. 「保存」をクリックします。

一般設定

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。

表 6-1　JDBC 接続プールの一般設定

パラメータ	説明
データソースクラス名	DataSource API、ConnectionPoolDataSource API、および XADataSource API 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に限る)、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。

プール設定

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。

表 6-2　JDBC 接続プールのプール設定

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、アプリケーションサーバー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。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。

接続検証

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。

表 6-3　JDBC 接続プールの接続検証設定

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。

表 6-3　JDBC 接続プールの接続検証設定 (続き)

パラメータ	説明
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() 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。

トランザクション遮断

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。

表 6-4　JDBC 接続プールのトランザクション遮断設定

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。

表 6-4　JDBC 接続プールのトランザクション遮断設定 (続き)

パラメータ	説明
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遮断レベルに戻されます。

プロパティ

「追加プロパティ」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。

接続プール設定の検証

接続プール設定を検証するには、次の手順に従います。

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」ノードを開きます。
2. 「リソース」ノードで、「JDBC」ノードを開きます。
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。

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 リソースの作成

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。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。

JDBC リソースを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開きます。
2. 「リソース」ノードで、「JDBC」ノードを開きます。
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。規則により、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. 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。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。
    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。

同機能を持つ `asadmin` コマンド : `create-jdbc-resource`

JDBC リソースの編集

1. ツリーコンポーネントで、「リソース」ノードを開きます。
2. 「リソース」ノードで、「JDBC」ノードを開きます。
3. 「JDBC」ノードで、「JDBC リソース」ノードを開きます。
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」ノードを開きます。
2. 「リソース」ノードで、「JDBC」ノードを開きます。
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、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。
5. 「削除」をクリックします。

JDBC リソースの有効化と無効化

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」を開き、「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。

持続マネージャリソースについて

- 持続マネージャリソースの作成
- 持続マネージャリソースの編集
- リソースターゲットの管理
- 持続マネージャリソースの削除
- 持続マネージャリソースの有効化と無効化

持続マネージャリソースの作成

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) を使用したアプリケーションの持続マネージャリソースが必要でした。

持続マネージャリソースを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開きます。
2. 「リソース」ノードで、「持続マネージャ」ノードを選択します。
3. 「持続マネージャ」ページで、「新規」をクリックします。
4. 「持続マネージャを作成」ページで、次の設定を指定します。
    a. 「JNDI 名」フィールドに、一意の名前を入力します。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リソースが属する接続プールを選択します。

d. デフォルトでは、新しい持続マネージャリソースが有効になります。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。

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。

同機能を持つ asadmin コマンド : `create-persistence-resource`

持続マネージャリソースの編集

既存の持続マネージャリソースのプロパティを編集するには、次の手順に従います。

1. 「持続マネージャプロパティを編集」タブから、「プロパティを追加」を選択します。

   「追加プロパティ」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と値を追加します。

リソースターゲットの管理

リソースターゲットを管理するには、次の手順に従います。

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。

持続マネージャリソースの削除

1. ツリーコンポーネントで、「持続マネージャ」ノードを開きます。
2. 「持続マネージャ」ノードを選択します。
3. 「持続マネージャ」ページで、削除する持続マネージャ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。

同機能を持つ asadmin コマンド : `delete-persistence-resource`

持続マネージャリソースの有効化と無効化

1. ツリーコンポーネントで、「持続マネージャ」ノードを開きます。
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 7 章

可用性とセッション持続性の設定

この章では、Sun Java™ System Application Server Enterprise Edition 環境でセッション持続性と可用性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。

- 可用性とセッション持続性について
- 可用性設定に関する管理コンソールタスク

可用性とセッション持続性について

- セッション持続性が必要な理由
- セッション持続性設定の概要
- 可用性のレベル
- HTTP セッション状態のシングルサインオンの可用性
- サンプルアプリケーション

セッション持続性が必要な理由

アプリケーションセッションが進行していくと、従来型データベースに格納されていないセッションの一部のデータが出現することがあります。このようなデータの例として、ショッピングカートの中身があります。Sun Java System Application Server は、このセッションデータをリポジトリ内に保存、または持続させる機能を提供し、アプリケーションサーバーインスタンスに障害が発生しても、セッションの状態が復元されて情報を損失せずにセッションを継続できるようにしています。

J2EE アプリケーションでは、セッションデータは通常 HTTP セッションまたはステートフルセッション Bean (SFSB) セッションに格納されます。Sun Java System Application Server は、HTTP セッションと SFSB セッションの状態の持続性をサポートします。HTTP セッションおよび SFSB セッション内に保存される特定の J2EE オブジェクト参照のフェイルオーバーもサポートします。『Developer's Guide』を参照してください。

Sun Java System Application Server に含まれている高可用性データベース (HADB) は、セッションデータの高可用性を提供する持続ストアとして機能します。

セッション持続性設定の概要

セッション持続性を正しく設定するためには、以下の手順を表示される順に実行してください。あとの手順には、前提条件として前の手順が含まれる場合があります。

1. クラスタの HADB データベースを作成します。『Reference Manual』のコマンド `configure-ha-cluster` の説明を参照してください。
2. クラスタの HTTP ロードバランスを設定します。第 3 章「ロードバランスとフェイルオーバーの設定」を参照してください。
3. セッション持続性をサポートする必要があるアプリケーションサーバーインスタンスおよび Web または EJB コンテナの可用性を有効にして、セッション持続性の設定を行います。次の方法のうち 1 つを選択します。
   - 161 ページの「可用性設定に関する管理コンソールタスク」を参照してください。
   - 『Reference Manual』のコマンド `configure-ha-persistence` の説明を参照してください。
4. 可用性を有効にしない場合、SFSB のファイルシステムのセッションストアを必要に応じて変更できます。161 ページの「可用性が無効の場合の SFSB セッションストアの設定」を参照してください。
5. クラスタ内の各サーバーインスタンスを再起動します。
6. 可用性を必要とする特定の SFSB の可用性を有効にして、セッション状態にチェックポイントを設定する必要がある方法を選択します。『Developer's Guide』を参照してください。
7. 高可用性を必要とする各 Web モジュールを分散可能にします。『Developer's Guide』を参照してください。
8. 配備中に、J2EE アプリケーション、Web モジュール、または EJB モジュールの可用性を有効にします。管理コンソールで、可用性を有効にするチェックボックスをチェックするか、または `deploy` コマンドを、`--availabilityenabled` オプションを `true` にして使用します。

注 セッション持続性は、動的配備、動的再読み込み、および自動配備とは互換性がありません。これらの配備機能は、開発環境を対象としていて、本稼動環境は対象としていません。これらの機能を無効にする方法については、第 5 章「アプリケーションの配備」を参照してください。

注 インスタンスが現在要求を処理中の場合、インスタンスをいったん停止してから再起動して、インスタンスが要求を処理する時間が十分に取れるようにします。詳細については、「サーバーインスタンスまたはクラスタの無効化 (停止)」を参照してください。

可用性のレベル

可用性は、5 つの異なるレベルで有効にできます。

1. デフォルトで有効になっているサーバーインスタンス
2. デフォルトで有効になっている Web または EJB コンテナ
3. デフォルトで無効になっているアプリケーション
4. デフォルトで無効になっているスタンドアロンの Web または EJB モジュール
5. デフォルトで無効になっている SFSB

可用性をある特定のレベルで有効にするには、それより上のすべてのレベルでも有効にする必要があります。たとえば、アプリケーションレベルで可用性を有効にするには、サーバーインスタンスレベルおよびコンテナレベルでも有効にする必要があります。

ある特定のレベルの可用性は、デフォルトでは 1 つ上のレベルに設定されます。たとえば、可用性がコンテナレベルで有効になっている場合、デフォルトではアプリケーションレベルで有効になります。

可用性がサーバーインスタンスレベルで無効になっている場合、ほかのすべてのレベルで有効にしても反映されません。可用性がサーバーインスタンスレベルで有効になっている場合、明示的に無効化しないかぎり、すべてのレベルで有効になります。

HTTP セッション状態のシングルサインオンの可用性

単一のアプリケーションサーバーインスタンスでは、ユーザーがある 1 つのアプリケーションによって認証されると、同じインスタンス上で動作しているほかのアプリケーションに対して個別に再認証を行う必要はありません。これをシングルサインオンといいます。シングルサインオンの詳細については、235 ページの「シングルサインオンの確認」を参照してください。

HTTP セッションがクラスタ内のほかのインスタンスにフェイルオーバーした場合でも、シングルサインオンが機能し続けるようにするには、シングルサインオン情報が HADB に対して持続される必要があります。まず、サーバーインスタンスと Web コンテナの可用性を有効にして、次にシングルサインオン状態の持続性を有効にします。162 ページの「サーバーインスタンスレベルの可用性の設定」を参照してください。

単一の名前とパスワードの組み合わせによってアクセス可能なアプリケーションは、シングルサインオングループを構成します。

シングルサインオングループに属するアプリケーションに対応する HTTP セッションでは、1 つのセッションがタイムアウトになった場合、ほかのセッションは無効化されず、引き続き有効となります。これは、1 つのセッションがタイムアウトしてもほかのセッションの可用性には影響しないからです。

この動作の当然の結果として、あるセッションがタイムアウトして、セッションを実行していた同じブラウザウィンドウから対応するアプリケーションにアクセスを試みる場合、再度認証を行う必要はありません。ただし、新しいセッションが作成されます。

シングルサインオングループに属するショッピングカートアプリケーションの例を挙げます。このグループにはほかに 2 つのアプリケーションが含まれます。ほかの 2 つのアプリケーションのセッションタイムアウト値は、ショッピングカートアプリケーションのセッションタイムアウト値を上回るものと仮定します。ショッピングカートアプリケーションのセッションがタイムアウトして、セッションを実行していた同じブラウザウィンドウからショッピングカートアプリケーションの実行を試みる場合、再度認証を行う必要はありません。ただし、以前のショッピングカートは失われていて、新しいショッピングカートを作成する必要があります。ほかの 2 つのアプリケーションは、ショッピングカートアプリケーションを実行していたセッションのタイムアウト後も変わらず動作し続けます。

同様に、ほかの 2 つのアプリケーションのどちらかに対応するセッションがタイムアウトしたとします。セッションを実行していた同じブラウザウィンドウからアプリケーションに接続している間は、再度認証を行う必要はありません。

注 この動作は、セッションがタイムアウトした場合にのみ当てはまります。シングルサインオンが有効になっていて、`HttpSession.invalidate()` を使用してセッションの 1 つを無効にする場合、シングルサインオングループに属するすべてのアプリケーションのセッションが無効になります。シングルサインオングループに属する任意のアプリケーションへのアクセスを試みる場合、再認証が必要であり、アプリケーションにアクセスするクライアントに対して新しいセッションが作成されます。

サンプルアプリケーション

次のディレクトリには、HTTP および SFSB セッションの持続性を示すサンプルアプリケーションが含まれています。

install_dir`/samples/ee-samples/highavailability`
install_dir`/samples/ee-samples/failover`

可用性設定に関する管理コンソールタスク

- 可用性が無効の場合の SFSB セッションストアの設定
- サーバーインスタンスレベルの可用性の設定
- Web コンテナレベルの可用性の設定
- EJB コンテナレベルの可用性の設定

可用性が無効の場合の SFSB セッションストアの設定

可用性が無効になっている場合、ローカルファイルシステムは SFSB 状態の非活性化に使用されますが、持続性には使用されません。SFSB 状態が格納される場所を変更するには、EJB コンテナのセッション格納位置の設定を変更します。221 ページの「一般的な EJB 設定の設定」を参照してください。

サーバーインスタンスレベルの可用性の設定

管理コンソールを使用して、サーバーインスタンスレベルの可用性を有効または無効にするには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 編集する設定のノードを開きます。
3. 「可用性サービス」ノードを選択します。
4. 「可用性サービス」ページに移動します。
5. 「可用性サービス」ボックスにチェックマークを付けて、インスタンスレベルの可用性を有効にします。無効にするには、このボックスのチェックマークを外します。

   セッションの持続性のために HADB への接続に使用する JDBC リソースを変更した場合、格納プール名を変更できます。詳細については、『Reference Manual』のコマンド `configure-ha-cluster` の説明を参照してください。
6. 「保存」ボタンをクリックします。
7. 「インスタンス」ノードを開きます。
8. サーバーインスタンスを選択します。
9. 「サーバーインスタンス」ページに移動します。
10. サーバーを再起動します。

Web コンテナレベルの可用性の設定

個別の Web アプリケーションの可用性を有効にする、または可用性設定をオーバーライドするには、`sun-web.xml` ファイルの設定を使用します。詳細については、『Developer's Guide』を参照してください。

管理コンソールを使用して、Web コンテナの可用性を有効または無効にするには、次の手順に従います。

1. 「Web コンテナの可用性」タブを選択して、「可用性サービス」ボックスにチェックマークを付けます。無効にするには、このボックスのチェックマークを外します。次のオプション設定を変更することもできます。
   - 持続性のタイプ : 可用性が有効化されている Web アプリケーションのセッションの持続性メカニズムを指定します。使用できる値は、`memory` (持続性なし) `file` (ファイルシステム)、および `ha` (HADB) です。可用性が有効の場合、デフォルトは `ha` です。可用性が無効の場合、デフォルトは `memory` です。セッションの持続性が必要となる本稼動環境では、`ha` を使用します。

持続性のタイプを memory に設定すると、サーバーインスタンスが正常にシャットダウンする場合に、sessionFilename プロパティを使用して HTTP セッション状態が格納されるファイルシステムの場所を指定できます。このオプションは内部テストには役立ちますが、本稼働環境ではサポートされません。

持続性のタイプを file に設定すると、ディレクトリのプロパティを使用して HTTP セッション状態が格納されるファイルシステムの場所を指定できます。ファイルシステムに対する持続性は内部テストには役立ちますが、本稼働環境ではサポートされません。

- 持続性の頻度 : セッション状態を格納する頻度を指定します。持続性のタイプが ha の場合にのみ適用できます。使用できる値は次のとおりです。
  - web-method - セッション状態は、各 Web 要求の終了時に、クライアントに応答を返信する前に格納されます。このモードでは、障害発生時にセッション状態を完全に更新するための最良の保証が得られます。デフォルトです。
  - time-based - セッション状態が、reapIntervalSeconds ストアプロパティによって設定された頻度でバックグラウンドに格納されます。このモードでは、セッション状態が必ずしも完全に更新される保証は得られません。ただし、各要求後に状態が格納されないので、パフォーマンスが大幅に向上します。このプロパティの設定については、220 ページの「ストアプロパティの設定」を参照してください。
- 持続性の範囲 : セッション状態を格納する範囲を指定します。持続性のタイプが ha の場合にのみ適用できます。使用できる値は次のとおりです。
  - session - 常にすべてのセッション状態が格納されます。このモードでは、セッションデータを分散可能な Web アプリケーションに正しく格納するための最良の保証が得られます。デフォルトです。
  - modified-session - セッション状態が変更された場合、すべてのセッション状態が格納されます。HttpSession.setAttribute() または HttpSession.removeAttribute() が呼び出された場合に、セッションが変更されたと見なします。属性が変更されるたびに、必ず setAttribute() を呼び出す必要があります。これは J2EE 仕様の要件ではありませんが、このモードを正しく動作させるために必要になります。
  - modified-attribute - 変更されたセッション属性だけが格納されます。このモードを正しく動作させるには、次のガイドラインに従う必要があります。

    セッション状態が変更されるたびに、setAttribute() を呼び出します。

    属性間で相互参照しないようにします。別個の各属性キーにあるオブジェクトグラフを直列化し、別々に格納します。別個の各キーにあるオブジェクト間に相互参照がある場合は、正常な直列化および非直列化は行われません。

    複数の属性間、または少なくとも読み取り専用属性と変更可能な属性間でセッション状態を分散します。

- シングルサインオン状態 : このボックスにチェックマークを付けて、シングルサインオン状態の持続性を有効にします。無効にするには、このボックスのチェックマークを外します。
- HTTP セッションストア : セッションの持続性のために HADB への接続に使用する JDBC リソースを変更した場合、HTTP セッションストアを変更できます。詳細については、『Reference Manual』のコマンド `configure-ha-cluster` の説明を参照してください。

2. 「保存」ボタンをクリックします。
3. セッション持続性に影響する追加のオプション設定を変更するには、219 ページの「Web コンテナセッションの設定」を参照してください。
4. 「インスタンス」ノードを開きます。
5. サーバーインスタンスを選択します。
6. 「サーバーインスタンス」ページに移動します。
7. サーバーを再起動します。

EJB コンテナレベルの可用性の設定

可用性を有効化し、個々のステートフルセッション Bean (SFSB) のチェックポイントを設定する方法を選択するには、`sun-ejb-jar.xml` ファイルの設定を使用します。詳細については、『Developer's Guide』を参照してください。

管理コンソールを使用して、EJB コンテナの可用性を有効または無効にするには、次の手順に従います。

1. 「EJB コンテナの可用性」タブを選択して、「可用性サービス」ボックスにチェックマークを付けます。無効にするには、このボックスのチェックマークを外します。次のオプション設定を変更することもできます。
   - HA 持続性のタイプ : 可用性が有効化されている SFSB のセッションの持続性および非活性化メカニズムを指定します。使用できる値は、`file` (ファイルシステム) と `ha` (HADB) です。セッションの持続性が必要となる本稼動環境では、デフォルトの `ha` を使用します。
   - SFSB 持続性のタイプ: 可用性が有効化されていない SFSB の非活性化メカニズムを指定します。使用できる値は、`file` (デフォルト) と `ha` です。

     いずれかの持続性のタイプを `file` に設定すると、EJB コンテナによって非活性化されたセッション Bean が格納されるファイルシステムの場所が指定されます。221 ページの「一般的な EJB 設定の設定」を参照してください。ファイルシステムに対するチェックポイントは内部テストには役立ちますが、本稼働環境ではサポートされません。

- SFSB ストアプール名 : セッションの持続性のために HADB への接続に使用する JDBC リソースを変更した場合、SFSB ストアプール名を変更できます。詳細については、『Reference Manual』のコマンド `configure-ha-cluster` の説明を参照してください。

2. 「保存」ボタンをクリックします。
3. 「インスタンス」ノードを開きます。
4. サーバーインスタンスを選択します。
5. 「サーバーインスタンス」ページに移動します。
6. サーバーを再起動します。

第 8 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
- JMS 接続ファクトリに関する管理コンソールタスク
- JMS 送信先リソースに関する管理コンソールタスク
- JMS 物理送信先に関する管理コンソールタスク
- JMS プロバイダに関する管理コンソールタスク

JMS リソースについて

- Application Server の JMS プロバイダ
- JMS リソース
- JMS リソースとコネクタリソースの関係

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り 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、*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、『Sun Java System 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
  - 物理的送信先
  - 物理的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 についての章では、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廃止された 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、管理コンソールツリーに表示される「コネクタ」ノードの下に表示されます。

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。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。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 11 章「コネクタリソース」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の作成

JMS 接続ファクトリリソースを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「JMS リソース」ノードを開きます。
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。次に例を示します。

   ```
   jms/ConnectionFactory1
   ```

   JMS 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 `jms/` を使用することをお勧めします。
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、204 ページの「コネクタ接続プールの編集」の表「コネクタ接続プールのプール設定」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。

   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の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のリソースが 1 つだけ必要となるトランザクション、または複数の 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。

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

表 8-1　JMS 接続ファクトリの追加プロパティ

プロパティ名	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。
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、可能な値には earth や earth:7677 があります。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、『Sun Java System 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。このプロパティ名は廃止されました。代わりに AddressList を使用します。

表 8-1　JMS 接続ファクトリの追加プロパティ (続き)

プロパティ名	説明
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 (値が true) な場合、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。これは、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

表 8-1　JMS 接続ファクトリの追加プロパティ (続き)

プロパティ名	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

9. 「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。
10. 「了解」をクリックして、接続ファクトリを保存します。

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の編集

JMS 接続ファクトリリソースを編集するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「JMS リソース」ノードを開きます。
2. 「接続ファクトリ」ノードを開きます。
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の削除

JMS 接続ファクトリリソースを削除するには、次の手順に従います。

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

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 送信先リソースの作成

JMS 送信先リソースを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「JMS リソース」ノードを開きます。
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。次に例を示します。

   ```
   jms/Queue
   ```

   JMS 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 jms/ を使用することをお勧めします。
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」セクションで、プロパティの値を指定します。次の表には、使用可能なプロパティが一覧表示されています。

表 8-2　JMS 送信先リソースの追加プロパティ

プロパティ名	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。

同機能を持つ asadmin コマンド : create-jms-resource

JMS 送信先リソースの編集

JMS 送信先リソースを編集するには、次の手順に従います。

1. ツリーコンポーネントで、「リソース」ノードを開き、次に「JMS リソース」ノードを開きます。
2. 「送信先リソース」ノードを開きます。
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 送信先リソースの削除

JMS 送信先リソースを削除するには、次の手順に従います。

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

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 物理送信先の作成

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 の設定プロパティで指定した期限が切れると効力を失います。

JMS 物理送信先を作成するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開き、「物理送信先」ノードを選択します。の前にきます。つまり、「Java メッセージサービス」ノードを開きます。次に「物理送信先」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。
   a. 特定のインスタンスを設定するには、インスタンスの設定ノードを選択します。たとえば、デフォルトのインスタンス `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設定を設定するには、`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」セクションで、「プロパティを追加」をクリックしてプロパティを追加します。次の表に、現在使用可能な 1 つのプロパティを示します。

表 8-3　JMS 物理送信先の追加プロパティ

プロパティ名	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。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、デフォルトは 1 であり、クラスタに対して作成される場合、デフォルトは -1 です。

このプロパティの値を変更するか、またはほかの物理送信先プロパティを指定するには、*install_dir*/imq/bin/imqcmd コマンドを使用します。詳細については、『Sun Java System 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。

「物理送信先」ページに、有効期限が切れ、配信不可能の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。

同機能を持つ asadmin コマンド : create-jmsdest

JMS 物理送信先の削除

JMS 物理送信先を削除するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開き、「Java メッセージサービス」ノードを開きます。次に「物理送信先」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。
   a. 特定のインスタンスを設定するには、インスタンスの設定ノードを選択します。たとえば、デフォルトのインスタンス 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設定を設定するには、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。

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の設定

「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を設定します。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。
   a. 特定のインスタンスを設定するには、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設定を設定するには、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 を選択する場合、JMS サービスは次のサーバーの起動時には 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、『Sun Java System 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、185 ページの「JMS ホストの編集」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。*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。これは、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. 非デフォルト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 MQ 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* は次の表に示します。

    「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。

表 8-4　メッセージサーバーのアドレススキームと構文

スキーム名	接続サービス	説明	アドレス構文
mq	jms と ssljms	MQ 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 MQ クライアントランタイムが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*] デフォルトは次のとおり *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
mqtcp	jms	MQ 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

表 8-4　メッセージサーバーのアドレススキームと構文 (続き)

スキーム名	接続サービス	説明	アドレス構文
mqssl	ssljms	MQ 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 を確立します。	*hostName*:*port*/ssljms 例 : mqssl:localhost:7676/ssljms
http	httpjms	MQ クライアントランタイムは、指定された URL の MQ 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、MQ 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構文は次のとおりです。 http://*hostname*:*port*/*contextRoot*/tunnel?*serverName*=*hostName*:*instanceName*
https	httpsjms	MQ クライアントランタイムは、指定された MQ トンネルサーブレット URL にセキュリティー保護された HTTP 接続を確立します。ブローカは、MQ 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」セクションで、「プロパティを追加」をクリックしてプロパティを追加します。次の表に、利用可能な Message Queue ブローカの設定プロパティのリストを示します。

表 8-5　JMS プロバイダの追加プロパティ

プロパティ名	説明
instance-name	完全な Sun Java System 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。

表 8-5　JMS プロバイダの追加プロパティ (続き)

プロパティ名	説明
instance-name-suffix	完全な Sun Java System 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。
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 Sun Java System 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。

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。

    「Ping」をクリックして、JMS サービスが起動し、実行されているかどうかを確認します。JMS サービスが起動し、実行されている場合、「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。

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 プロパティを使用して、接続ファクトリリソースを作成します。

JMS サービスの設定の詳細については、『Sun Java System Application Server Developer's Guide』を参照してください。

同機能を持つ asadmin コマンド : jms-ping

JMS ホストの作成

JMS ホストを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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、インスタンスの設定ノードを選択します。たとえば、デフォルトのインスタンス `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設定を設定するには、`default-config` ノードを選択します。
3. 「Java メッセージサービス」ノードを開きます。
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。JMS サービスを非デフォルト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、MQ ブローカのユーザー名とパスワードを入力します。これらは、Application Server のユーザー名と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、*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。

同機能を持つ `asadmin` コマンド : `create-jms-host`

JMS ホストの編集

JMS ホストを編集するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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、インスタンスの設定ノードを選択します。たとえば、デフォルトのインスタンス `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設定を設定するには、`default-config` ノードを選択します。
3. 「Java メッセージサービス」ノードを開きます。
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 ホストの削除

JMS ホストを削除するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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、インスタンスの設定ノードを選択します。たとえば、デフォルトのインスタンス `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設定を設定するには、`default-config` ノードを選択します。
3. 「Java メッセージサービス」ノードを開きます。
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。

同機能を持つ `asadmin` コマンド : `delete-jms-host`

第 9 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

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、メールと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 ウェブサイト (`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 セッションの作成

JavaMail セッションを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「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。次に例を示します。

   ```
   mail/MySession
   ```

   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。
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. Application Server のメールプロバイダが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。

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含まれます。ログレベルの設定の詳細については、348 ページの「ログレベルの設定」を参照してください。

9. プロトコル固有のホストや username プロパティなど、アプリケーションで必要なプロパティを追加するには、「追加プロパティ」フィールドで「プロパティを追加」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のリストがあります (http://java.sun.com/products/javamail/javadocs/index.html)。
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、セッションを保存します。

同機能を持つ asadmin コマンド : create-javamail-resource

JavaMail セッションの編集

JavaMail セッションを編集するには、次の手順に従います。

1. ツリーコンポーネントで、「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 セッションの削除

JavaMail セッションを削除するには、次の手順に従います。

1. ツリーコンポーネントで、「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。

同機能を持つ `asadmin` コマンド : `delete-javamail-resource`

第 10 章

JNDI リソース

この章では、管理コンソールを使用して JNDI 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。

- Java Naming and Directory Interface (JNDI) について
- カスタムリソースについて
- 外部 JNDI リポジトリおよびリソースについて

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 ルックアップ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
- J2EE ネームサービス
- ネーミング参照とバインディング情報

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サービス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。Sun Java System 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ネームスペースに追加します。

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、後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。

表 10-1 には、Sun Java System Application Server が使用する J2EE リソースの JNDI ルックアップと関連する参照が一覧表示されています。

表 10-1 JNDI ルックアップと関連する参照

JNDI ルックアップ名	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

表 10-1　JNDI ルックアップと関連する参照 (続き)

JNDI ルックアップ名	関連する参照
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 はもとより、リソースを作成、削除、一覧表示することができます。

カスタムリソースの作成

カスタムリソースを作成するには、次の手順に従います。

1. 管理コンソールの左側の区画で、JNDI 設定を変更する Sun Java System 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 ネーミング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する必要があります。
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」セクションで、プロパティ名およびプロパティ値を追加します。
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。

同機能を持つ asadmin コマンド : create-custom-resource

カスタムリソースの編集

カスタムリソースを編集するには、次の手順に従います。

1. 管理コンソールの左側の区画で、JNDI 設定を変更する Sun Java System 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。カスタム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。
3. 「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。
4. 「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。
5. 「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。

カスタムリソースの削除

カスタムリソースを削除するには、次の手順に従います。

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。カスタムリソースを削除するには、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。
3. 「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。

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 ist-custom-resources` コマンドを入力します。たとえば、カスタムリソースをホスト plum 上で一覧表示するには、次のように入力します。

```
$asadmin list-custom-resource --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、Sun Java System 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 エレメントは、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。
-- ファクトリクラスの要素は、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。プロパティ要素は
-- 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>
```

外部リソースの作成

外部リソースを作成するには、次の手順に従います。

1. 管理コンソールの左側の区画で、JNDI 設定を変更する Sun Java System 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。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。
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 ネーミング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. 「JNDI ルックアップ」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 ルックアップ」は次のようになります。
   `cn=`*`testmybean`*
6. 「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。
7. 「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。
8. 「追加プロパティ」セクションで、プロパティ名およびプロパティ値を追加します。
9. 「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。
10. 「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。

同機能を持つ asadmin コマンド : `create-jndi-resource`

外部リソースの編集

外部リソースを編集するには、次の手順に従います。

1. 管理コンソールの左側の区画で、JNDI 設定を変更する Sun Java System 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。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。
3. 「リソースタイプ」フィールド、「JNDI ルックアップ」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。
4. 「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。
5. 「保存」をクリックして、外部リソースの変更を保存します。

外部リソースの削除

外部リソースを削除するには、次の手順に従います。

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。外部リソースを削除するには、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。
3. 「削除」をクリックします。外部リソースが削除されます。

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 -- target 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 11 章

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、接続プール、およびリソース

コネクタモジュールとは、アプリケーションが EIS (Enterprise Information System) 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。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のタスク

EIS アクセス設定の一般手順

1. コネクタを配備 (インストール) します。「コネクタモジュールの配備」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。「コネクタ接続プールの作成」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。「コネクタリソースの作成」を参照してください。

コネクタ接続プールの作成

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。

接続プールを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「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「接続プールを作成」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「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「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」テーブルで、必要なプロパティを追加します。

      前の「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」テーブルにはデフォルトプロパティが表示されます。
9. 「完了」をクリックします。

同機能を持つ `asadmin` コマンド : `create-connector-connection-pool`

- コネクタモジュール、接続プール、およびリソース
- コネクタモジュールの配備

コネクタ接続プールの編集

「コネクタ接続プールを編集」ページを使うと、プールの設定や追加のプロパティを変更できます。

「コネクタ接続プールを編集」ページにアクセスするには、次の手順に従います。

1. ツリーコンポーネントで、「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。表 11-1 を参照してください。

表 11-1 コネクタ接続プールのプール設定

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、アプリケーションサーバー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。

表 11-1 コネクタ接続プールのプール設定

パラメータ	説明
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の内部で管理され、外部トランザクションマネージャ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の外部にあるトランザクションマネージャによって制御され、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の内部で管理され、外部トランザクションマネージャは一切関与しません。

5. 「追加プロパティ」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。このテーブルを使用して配備担当者が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. 「セキュリティマップ」タブ付き区画で、接続プールのセキュリティマップを作成または変更します。セキュリティマップの作成方法の詳細については、「セキュリティマップについて」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。

同機能を持つ asadmin コマンド : `delete-connector-connection-pool`

コネクタリソースのタスク

- コネクタリソースの作成
- コネクタリソースの編集
- コネクタリソースの削除
- コネクタサービスの設定

コネクタリソースの作成

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。コネクタリソースを作成する前に、まず接続プールを作成します。

コネクタリソースを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「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. 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。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。

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。

同機能を持つ `asadmin` コマンド : `create-connector-resource`

コネクタリソースの編集

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、「コネクタリソースの作成」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。

同機能を持つ asadmin コマンド : `delete-connector-resource`

コネクタサービスの設定

「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。

コネクタコンテナを設定するには、次の手順に従います。

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。
   a. 特定のインスタンスを設定するには、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。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、アプリケーションサーバーが待機する秒数 (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のタスク

管理対象オブジェクトリソースの作成

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 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。

管理対象オブジェクトリソースを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「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。

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の編集

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、「管理対象オブジェクトリソースの作成」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. 「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。

同機能を持つ asadmin コマンド : `delete-admin-object`

第 12 章

名前付き設定の管理

この章では、Application Server における名前付きサーバー設定の追加、変更、および使用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。

名前付き設定について

名前付き設定

名前付き設定とは、一連のサーバー設定情報です。この情報には、HTTP リスナー、ORB/IIOP リスナー、JMS ブローカ、EJB コンテナ、セキュリティ、ロギング、および監視などの設定情報が含まれます。アプリケーションやリソースは、名前付き設定では定義されません。

設定は管理ドメインに作成されます。ドメイン内の複数のサーバーインスタンスが同じ設定を参照したり、別個の設定を使用したりできます。

クラスタでは、クラスタのインスタンスで均質の環境が確保されるように、クラスタ内のすべてのサーバーインスタンスがクラスタの設定を継承します。

名前付き設定には数多くの必須設定情報が含まれるため、既存の名前付き設定をコピーして新しい設定を作成します。設定情報を変更しないかぎり、新規に作成された設定はコピーした設定と同じです。

クラスタとインスタンスには、クラスタまたはインスタンスが設定を使用する方法に応じて次の 3 つのタイプがあります。

- **スタンドアロン** : スタンドアロンのサーバーインスタンスまたはクラスタは、ほかのサーバーインスタンスと設定を共有しません。つまり、ほかのサーバーインスタンスまたはクラスタは名前付き設定を参照しません。
- **共有** : 共有サーバーインスタンスまたはクラスタは、ほかのサーバーインスタンスまたはクラスタと設定を共有します。つまり、複数のインスタンスまたはクラスタが同じ名前付き設定を参照します。
- **クラスタ化** : クラスタ化されたサーバーインスタンスはクラスタの設定を継承します。

default-config 設定

`default-config` 設定は、スタンドアロンサーバーインスタンスまたはスタンドアロンクラスタの設定を作成するテンプレートとして機能する特殊な設定です。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。

インスタンスまたはクラスタの作成時に作成された設定

新しいサーバーインスタンスまたは新しいクラスタを作成する場合は、次のどちらかを実行します。

- 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。

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クラスタまたはインスタンスが作成されます。別の設定からコピーするには、新規インスタンスまたはクラスタの作成時に設定を指定します。

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。クラスタの場合、新しい設定には *cluster-name*`-config` という名前が付けられます。

一意のポート番号と設定

同じホストマシン上の複数のインスタンスが同じ設定を参照する場合、各インスタンスは独自のポート番号を待機する必要があります。たとえば、ポート 80 の HTTP リスナーを使用する名前付き設定を 2 つのサーバーインスタンスが参照する場合、ポートの競合により、どちらかのサーバーインスタンスが起動できなくなります。一意のポートが使用されるように、個々のサーバーインスタンスが待機するポート番号を定義するプロパティを変更します。

ポート番号に次の原則を適用します。

- 個々のサーバーインスタンスのポート番号は、最初に設定から継承されます。
- サーバーインスタンスの作成時にポートがすでに使用されている場合は、継承されたデフォルト値をインスタンスレベルでオーバーライドして、ポートの競合を防止します。
- インスタンスが設定を共有しているものと仮定します。設定はポート番号 n を使用します。同じ設定を使用するマシンで新しいインスタンスを作成する場合、新しいインスタンスにはポート番号 n+1 が割り当てられます (使用可能な場合)。この番号が使用できない場合は、n+1 の次に使用可能なポートが選択されます。
- 設定のポート番号を変更する場合、そのポート番号を継承するサーバーインスタンスは変更されたポート番号を自動的に継承します。
- インスタンスのポート番号を変更し、続いて設定のポート番号を変更する場合、インスタンスのポート番号は変更されません。

名前付き設定に関する管理コンソールタスク

名前付き設定の作成

名前付き設定を作成するには、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」ページで、「新規」をクリックします。
3. 「設定の作成」ページで、一意の設定の名前を入力します。
4. 設定を選択して、コピーします。

   `default-config` 設定は、スタンドアロンサーバーインスタンスまたはスタンドアロンクラスタを作成するときに使用するデフォルトの設定です。

同機能を持つ `asadmin` コマンド : `copy-config`

名前付き設定のプロパティの編集

名前付き設定を編集するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 名前付き設定のノードを選択します。
3. 「システムプロパティの設定」ページで、動的再設定を有効にするかどうかを選択します。

   有効な場合は、設定に対する変更は、サーバーを再起動することなくサーバーインスタンスに適用されます。
4. プロパティの追加、プロパティの現在の値の変更、またはプロパティの削除を実行します。

   すでに定義されているプロパティはポートです。複数のサーバーインスタンスがある場合、ポート番号は一意にする必要があります。

表 12-1 に、あらかじめ定義されたプロパティとその説明のリストを示します。

表 12-1 名前付き設定のプロパティ

プロパティ名	説明
HTTP_LISTENER_PORT	このプロパティは、http-listener-1 のポート番号を指定します。有効な値は 1 ～ 65535。UNIX で、ポート 1 ～ 1024 を待機するソケットを作成する場合はスーパーユーザー権限が必要です。

表 12-1　名前付き設定のプロパティ (続き)

プロパティ名	説明
HTTP_SSL_LISTENER_PORT	このプロパティは、http-listener-2 のポート番号を指定します。有効な値は 1 ～ 65535。UNIX で、ポート 1 ～ 1024 を待機するソケットを作成する場合はスーパーユーザー権限が必要です。
IIOP_SSL_LISTENER_PORT	このプロパティは、SSL 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。
IIOP_LISTENER_PORT	このプロパティは、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。
JMX_SYSTEM_CONNECTOR_PORT	このプロパティは、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。UNIX で、ポート 1 ～ 1024 を待機するソケットを作成する場合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。

5. 設定に関連するすべてのインスタンスの現在のプロパティの値を編集するには、「インスタンス値」をクリックします。

同機能を持つ `asadmin` コマンド : `set`

設定を参照するインスタンスのポート番号の編集

名前付き設定を参照する各インスタンスは、最初にその設定からポート番号を継承します。ポート番号はシステムで一意である必要があるため、継承されたポート番号をオーバーライドする必要があります。

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 名前付き設定のノードを選択します。
3. 「システムプロパティの設定」ページで、編集するポート番号の横にある「インスタンス値」をクリックします。

   たとえば、SSL-port プロパティの横にある「インスタンス値」をクリックすると、設定を参照するすべてのサーバーインスタンスの SSL-port の値が表示されます。
4. ポートの値を変更して、「保存」をクリックします。

同機能を持つ asadmin コマンド : set

名前付き設定のターゲットの表示

名前付き設定のターゲットを表示するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 名前付き設定のノードを選択します。

   「システムプロパティの設定」ページに、設定を使用するすべてのターゲットのリストが表示されます。クラスタ設定の場合、これらのターゲットはクラスタです。インスタンス設定の場合、これらのターゲットはインスタンスです。

名前付き設定の削除

名前付き設定を削除するには、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」ページで、削除する名前付き設定のチェックボックスにチェックマークを付けます。

   default-config 設定は削除できません。
3. 「削除」をクリックします。

同機能を持つ asadmin コマンド : delete-config

第 13 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、「Application Server のアーキテクチャ」の節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 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 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。

コンテナの機能としては、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 をネーミングサービスにバインドしたり、認証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 の状態を持続性データに保存したり、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があります。

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、管理コンソールの Web コンテナのためのコンテナ全体の設定はありません。

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。
   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. すべてのインスタンスの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. 「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」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。

マネージャプロパティの設定

セッションマネージャを使用して、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。
   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. 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. 「マネージャプロパティ」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 is the
session object

}
```

   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。

ストアプロパティの設定

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。
   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. 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」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。

- セッション格納位置
- プール設定
- キャッシュ設定

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 Developer's Guide』を参照してください。このガイドへのリンクについては、「詳細情報」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 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 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 はキャッシングを無効にします。

プール設定

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 Beans にだけ適用されます。

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、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。

Enterprise JavaBeans のコンテナのプールの設定を調整するには、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。
   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. 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。

キャッシュ設定

コンテナは、最もよく使われる 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 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、または非活性化の 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。

キャッシュされた Enterprise JavaBeans の設定を調整するには、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。
   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. 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 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 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、最近もっとも使われていない 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 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、「一般的な EJB 設定の設定」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 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。

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。
   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. 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。

タイマーサービスの設定

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。
   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. 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の外部データベースの使用

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。

タイマーの格納に外部データベースを使うには次の手順に従います。

1. 152 ページの「JDBC リソースの作成」で説明されているとおり、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 14 章

セキュリティの設定

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概念について説明したあと、Sun Java System Application Server 8.1 2005Q1 のセキュリティ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 のセキュリティについて
- セキュリティに関する管理コンソールタスク
- レルムに関する管理コンソールタスク
- JACC プロバイダに関する管理コンソールタスク
- 監査モジュール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に関する管理コンソールタスク
- リスナーと JMX コネクタに関する管理コンソールタスク
- コネクタ接続プールに関する管理コンソールタスク
- 証明書と SSL の操作
- 詳細情報

Application Server のセキュリティについて

- セキュリティの概要
- 認証と承認について
- ユーザー、グループ、ロール、およびレルムについて
- 証明書および SSL の概要
- ファイアウォールについて
- 管理コンソールによるセキュリティの管理

セキュリティの概要

セキュリティとはデータの保護に関することであり、保存または伝送中にデータへの不正なアクセスやデータの損傷を防止する方法です。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アーキテクチャがあります。標準実装されているセキュリティ機能には、暗号化、認証と承認、および公開鍵インフラストラクチャがあります。Application Server は Java セキュリティモデルをベースに構築され、アーキテクチャ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があります。

- プログラムによるセキュリティでは、開発者が記述したアプリケーションコードによりセキュリティ動作を処理します。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では、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を処理します。宣言によるセキュリティは、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もあります。

プログラムによるセキュリティ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を中心に説明しています。

セキュリティ管理用ツール

Application Server には次のセキュリティ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。管理コンソールの一般的な基本情報については、「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タスクの概要については、「管理コンソールによるセキュリティの管理」を参照してください。
- asadmin。多くの管理コンソールと同じタスク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 の一般的な基本情報については、「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を詳細に説明しません。deploytool の使用手順については、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。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 セキュリティポリシー管理に使用するグラフィカルユーティリティ。管理者が、policytool を使用することはほとんどありません。

keytool、policytool、およびほかの Java セキュリティ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。
- 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。

certutil、pk12util、およびその他の NSS セキュリティ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管理

Application Server の今回のリリースの場合、ファイル domain.xml には特定のドメインの仕様が収められ、最初に平文で IMQ ブローカのパスワードが収められています。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リースを追加し、このユーザーやリリ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。

1. domain.xml ファイルがあるディレクトリ (デフォルトでは *domain_root_dir*/*domain_dir*/config) から、次の asadmin コマンドを実行します。

   ```
   asadmin create-password-alias < エイリアス名 >
   ```

   次に例を示します。

   ```
   asadmin create-password-alias 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。この目的の set コマンドの使用例は次のとおりです。

   ```
   asadmin set
   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_root_dir*/*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。

- `--passwordfile` 引数を使用する引数として、`asadmin` へ渡すために作成されたすべてのパスワードファイルは、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の変更

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。

マスターパスワードを変更するには、次の手順に従う必要があります。

1. ドメインの Application Server を停止します。新旧のパスワードをプロンプトする `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>
マスターパスワードを入力してください >
新しいマスターパスワードを入力してください >
新しいマスターパスワードをもう一度入力してください >
```

2. Application Server を再起動します。

警告 : 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、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。

管理パスワードの変更

管理パスワードの暗号化については、「パスワードのセキュリティ管理」で説明されています。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。この目的の `set` コマンドの使用例は次のとおりです。

```
asadmin set
server.jms-service.jms-host.default_JMS_host.admin-password=new_pwd
```

管理パスワードは、管理コンソールを使用しても変更できます。管理コンソールを使用して管理パスワードを変更するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを開きます。
4. 「レルム」ノードを開きます。
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。

セキュリティの責任の割り当て

セキュリティの責任は次のように割り当てます。

- アプリケーション開発者
- アプリケーション配備担当者
- 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。このセキュリティタスクの詳細については、次の URL に掲載されている『The J2EE 1.4 Tutorial』の「Security」の章で説明しています。

`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

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。このセキュリティタスクの詳細については、次の URL に掲載されている『The J2EE 1.4 Tutorial』の「Security」の章で説明しています。

`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

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機能の管理。

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の設定を管理し、`certutil` を使用し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の中心的なコンセプトです。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。

- エンティティの認証
- ユーザーの承認
- JACC プロバイダの指定
- 認証および承認の決定の監査
- メッセージセキュリティの設定

エンティティの認証

認証とは、あるエンティティ (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。エンティティは、セキュリティ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 ID に関連付けられます。

Application Server は、表 14-1 で説明されているように、4 つのタイプ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` を使用し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、次の URL で『The J2EE 1.4 Tutorial』を参照してください。

`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

表 14-1 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。

表 14-1　Application Server の認証メソッド (続き)

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
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の無効化については、「シングルサインオン (SSO) の設定」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、「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。『Developer's Guide』へのリンクの詳細については、「詳細情報」を参照してください。

メッセージセキュリティの設定

メッセージセキュリティによって、サーバーは、メッセージレイヤで Web サービスの呼び出しおよび応答のエンドツーエンドを認証できます。Application Server は、SOAP レイヤのメッセージセキュリティプロバイダを使用して、メッセージセキュリティを実装します。メッセージセキュリティプロバイダは、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プロバイダが付属しています。メッセージセキュリティプロバイダは、SOAP レイヤ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。

メッセージレイヤセキュリティのサポートは、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、メッセージレイヤセキュリティ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の設定については、「メッセージセキュリティの設定」で説明されています。アプリケーションレベルのメッセージセキュリティの設定については、『Developer's Guide』の「Securing Applications」の章で説明されています。

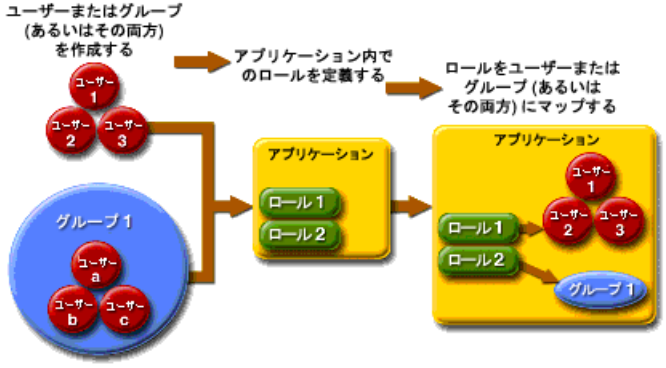
ユーザー、グループ、ロール、およびレルムについて

Application Server は次のエンティティに対して認証および承認ポリシーを実施します。

- ユーザー : Application Server で定義されている個別の ID。一般に、ユーザーとは、人物、Enterprise JavaBeans 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。
- グループ : 共通の特性によって分類され、Application Server で定義されたユーザーの集まり。
- 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。
- 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資格を含むリポジトリ。レルムは、セキュリティ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。

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

ロールでは、ユーザーがどのアプリケーション、および各アプリケーションのどの部分にアクセスできるか、また何ができるかを定義します。つまり、ロールで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には、少なくとも次の 2 つのロールが必要です。それは、employee および manager で、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

セキュリティポリシードメインまたはセキュリティドメインとも呼ばれているレルムとは、共通のセキュリティポリシーが定義および適用されている範囲のことです。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、次の 3 つのレルムが事前に設定されています。file (デフォルトの初期レルム)、certificate、および admin-realm です。さらに、ldap、solaris、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、「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、「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、「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、「カスタムレルムの作成」または『Developer's Guide』の「Securing Applications」の章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
- SSL (Secure Sockets Layer) について

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。詳細については、「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鍵 (非常に長い番号) 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、ほか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、証明書所有者の ID を確証し、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` レルムのユーザー ID を検証するため、認証サービスは、主体名として X.509 証明書の共通名フィールドを使用して 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証明書は、有効性を検証するため、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) 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スタンスまたは 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。「サーバー証明書の生成」で説明されている手順を使用して、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の管理

管理コンソールでは、セキュリティ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設定
- レルムおよび file レルムユーザー
- JACC プロバイダ
- 監査モジュール
- メッセージセキュリティ
- HTTP および IIOP リスナーのセキュリティ
- 管理サービスのセキュリティ
- セキュリティマップ

サーバーセキュリティ設定

「セキュリティ設定」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を設定します。詳細については、「セキュリティ設定の設定」を参照してください。

レルムおよび file レルムユーザー

レルムのコンセプトについては、「ユーザー、グループ、ロール、およびレルムについて」で説明されています。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、「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、「JACC プロバイダの指定」で説明されています。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、「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、「認証および承認の決定の監査」で説明されています。監査は、エラーやセキュリティ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。すべての認証イベントは、Application Server のログに記録されます。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、「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

メッセージセキュリティのコンセプトについては、「メッセージセキュリティの設定」で説明されています。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の有効化
- メッセージセキュリティプロバイダの設定
- 既存のメッセージセキュリティ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、「メッセージセキュリティの設定」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

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、「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 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、「IIOP リスナー」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の指定
- 既存の HTTP または IIOP リスナーのセキュリティ設定の変更

これらのタスクの詳細については、「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

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、DAS (domain administration server) 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設定の変更

これらのタスクの詳細については、「管理サービスの JMX コネクタのセキュリティの設定」を参照してください。

セキュリティマップ

コネクタ接続プールのセキュリティマップのコンセプトは、「セキュリティマップについて」で説明されています。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マップの追加

- 既存のセキュリティ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、「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。

セキュリティに関する管理コンソールタスク

セキュリティ設定の設定

管理コンソールの「セキュリティ」ページでは、システム全体のセキュリティ設定をさまざまに設定できます。

これらの設定を編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを選択します。

   「セキュリティ」ページが表示されます。
4. 必要に応じて値を変更します。セキュリティの一般的なオプションについては、表 14-2 で説明します。

表 14-2 一般的なセキュリティ設定

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。

表 14-2　一般的なセキュリティ設定 (続き)

設定	説明
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。
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。JACC プロバイダの追加については、「JACC プロバイダの作成」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については、「監査モジュールの作成」を参照してください。

5. 「追加プロパティ」セクションに、JVM (Java Virtual Machine) に渡すための追加プロパティを入力します。

   有効なプロパティは、「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。有効なプロパティは、次の項目で説明されています。

   - file および admin-realm レルムの編集
   - certificate レルムの編集
   - solaris レルムの作成

- ldap レルムの作成
- 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「デフォルトを読み込み」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制御

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。これを行うには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。
   b. 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 ノードを開きます。
3. 「セキュリティ」ノードを開きます。
4. 「レルム」ノードを開きます。
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

レルムの作成

Application Server では、次の 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、solaris、またはカスタムレルムも作成できます。通常、サーバーには各タイプのレルムが 1 つずつありますが、Application Server では、次の 2 つの file レルムがあります。file および 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。

セキュリティレルムを作成するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。
   b. 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 ノードを開きます。

3. 「セキュリティ」ノードを開きます。
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. 作成されたレルムのクラス名を指定します。表 14-3 に有効な値を示します。

表 14-3　レルムクラス名に有効な値

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および希望するオプションのプロパティを追加します。

   プロパティを追加するには、次の手順に従います。

   a. 「プロパティを追加」をクリックします。

   b. 「名前」フィールドに、プロパティの名前を入力します。

   - `file` レルムのプロパティの詳細については、「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の詳細については、「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の詳細については、「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の詳細については、「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の詳細については、「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、プロパティの値を入力します。

9. 「了解」をクリックします。

同機能を持つ asadmin コマンド : create-auth-realm

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。

レルムを作成するには、新しいレルムの追加について説明している「レルムの作成」の手順に従い、表 14-4 で示されているプロパティを追加します。

表 14-4　ldap レルムに必要なプロパティ

プロパティ名	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は、表 14-5 に示されています。

表 14-5　ldap レルムのオプションのプロパティ

プロパティ名	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

表 14-5　ldap レルムのオプションのプロパティ (続き)

プロパティ名	説明	デフォルト
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。	
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、表 14-6 で示す値を入力できます。

表 14-6　ldap レルムの値の例

プロパティ名	プロパティ値
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 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 ServerDeveloper'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。

カスタムレルムを使用するには、Application Server を次のように設定してください。

1. 「レルムの作成」の手順のアウトラインに従い、カスタムレルムの名前および 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. 表 14-7 に示されたプロパティを追加します。

表 14-7 カスタムレルムに有効なプロパティ

プロパティ名	プロパティ値
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_root_dir*/*domain_name*/config/login.conf を編集し、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 クラスおよび依存するすべてのクラスを、ディレクトリ *domain_root_dir*/*domain_name*/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。

   サーバーがレルムをロードしたことを確かめるため、*domain_root_dir*/*domain_name*/logs/server.log を確認してください。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。

レルムの編集

レルムを編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。
   b. 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 ノードを開きます。
3. 「セキュリティ」ノードを開きます。
4. 「レルム」ノードを開きます。
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とその値を編集します。

   file レルムのプロパティについては、「file および admin-realm レルムの編集」を参照してください。file レルムのユーザーを管理するには、「ユーザーを管理」ボタンをクリックします。詳しくは、「file レルムユーザーの管理」を参照してください。

certificate レルムのプロパティの詳細については、「certificate レルムの編集」を参照してください。

7. さらにプロパティを追加するには、「プロパティを追加」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。有効なプロパティ名およびプロパティ値を入力します。設定可能なオプションのプロパティについては次の各表を参照してください。
   - 表 14-4「ldap レルムに必要なプロパティ」
   - 表 14-5「ldap レルムのオプションのプロパティ」
   - 表 14-7「カスタムレルムに有効なプロパティ」
   - 表 14-8「file レルムに必要なプロパティ」
   - 表 14-9「certificate レルムのオプションのプロパティ」
8. 「保存」をクリックして変更を保存します。

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 プロパティで keyfile の場所が指定されています。表 14-8 に、file レルムに必要なプロパティを示します。

表 14-8 file レルムに必要なプロパティ

プロパティ名	説明	デフォルト値
file	keyfile の完全パスおよび名前。	*domain_root_dir*/*domain-name*/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。詳細については、「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 へのユーザーの追加の詳細については、「管理ツールへのアクセス制御」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、「file レルムユーザーの管理」の説明に従って管理コンソールを使用するか、または NSS ツールを使用して、ユーザーを管理できます。NSS (Network Security Services) とは、セキュリティ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 (Network Security Services) セキュリティツールについては、
  http://www.mozilla.org/projects/security/pki/nss/tools/
- NSS (Network Security Services) 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` プロパティで指定されます。

注 : この手順を使用して、ユーザーを `admin-realm` を含む任意のレルムに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` レルムのユーザーは、共通の特性で分類されるユーザーのカテゴリである J2EE グループに属することができます。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。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、`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` レルムユーザーを管理するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。

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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。

b. 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 ノードを開きます。

3. 「セキュリティ」ノードを開きます。
4. 「レルム」ノードを開きます。
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。

- ユーザーの追加
- ユーザーの編集
- ユーザーの削除

ユーザーの追加

「ファイルユーザー」ページで、次の手順に従って新しいユーザーを追加してください。

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。これらのグループは、ほかの場所で定義され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。

同機能を持つ asadmin コマンド : create-file-user

ユーザーの編集

「ファイルユーザー」ページで、次の手順に従ってユーザーの情報を変更してください。

1. 「ユーザー ID」列で、ユーザーの名前をクリックして変更します。

「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。

ユーザーの削除

「ファイルユーザー」ページで、次の手順に従ってユーザーを削除してください。

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。

同機能を持つ `asadmin` コマンド : `delete-file-user`

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。このレルムでは、Application Server のセキュリティコンテキストにユーザー ID が設定され、トラストストアとキーストアファイルのクライアント証明書から暗号を使用して取得されたユーザーデータが入力されます。「証明書ファイルについて」を参照してください。`certutil` を使用して、これらのファイルにユーザーを追加してください。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 ユーティリティについて」に記載した certutil のドキュメントを参照してください。

表 14-9 は、`certificate` レルムのオプションのプロパティを示しています。

表 14-9 `certificate` レルムのオプションのプロパティ

プロパティ	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、値は必ず `certificateRealm` にする必要があります。

相互認証の設定

- すべてのアプリケーションに対する相互認証の有効化
- アプリケーションの相互 SSL 認証の有効化

相互認証では、サーバーとクライアントサイド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、次の URL にある『The J2EE 1.4 Tutorial』の「Security」の章を参照してください。

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

すべてのアプリケーションに対する相互認証の有効化

Application Server は、HTTPS 認証に `certificate` レルムを使用します。

このレルムを使用するすべてのアプリケーションについて相互認証を指定するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを開きます。
4. 「レルム」ノードを開きます。
5. 「`certificate`」レルムを選択します。
6. 「プロパティを追加」ボタンをクリックします。
   - 「名前」フィールドに、`clientAuth` を入力します。
   - 「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、相互認証では `certificate` を使用するすべてのアプリケーションに対してクライアント認証が必要になります。

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、次の URL にある『The J2EE 1.4 Tutorial』の「Security」の章を参照してください。

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

レルムの削除

レルムを削除するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。
   b. 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 ノードを開きます。
3. 「セキュリティ」ノードを開きます。
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。

同機能を持つ asadmin コマンド : delete-auth-realm

デフォルトレルムの設定

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。

デフォルトレルムを設定するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。

b. 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 ノードを開きます。

3. 「セキュリティ」ノードを選択します。

   「セキュリティ」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「デフォルトを読み込み」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。

JACC プロバイダに関する管理コンソールタスク

JACC プロバイダの作成

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。

JACC プロバイダを作成するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。

2. 設定するインスタンスを開きます。

   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。

   b. 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 ノードを開きます。

3. 「セキュリティ」ノードを開きます。

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. 「プロパティを追加」ボタンをクリックして、プロバイダにプロパティを追加します。有効なプロパティは次のとおりです。
   - `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では、この値は *domain_root_dir*`/`*domain_dir*`/generated/policy` です。
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 プロバイダの編集

JACC プロバイダを編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを開きます。
4. 「JACC プロバイダ」ノードを開きます。
5. 編集する 「JACC プロバイダ」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では、この値は `${com.sun.aas.instanceRoot}/generated/policy` です。
8. 既存のプロパティを削除するには、プロパティの左側のチェックボックスをクリックして、「プロパティを削除」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 プロバイダの削除

JACC プロバイダを削除するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを開きます。
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 プロバイダの設定

JACC プロバイダを指定するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを選択します。

   「セキュリティ」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「デフォルトを読み込み」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

監査モジュールの作成

Application Server では、単純なデフォルト監査モジュールを提供しています。詳細については、「デフォルト監査モジュールの使用」を参照してください。

新しい監査モジュールを作成するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを開きます。
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 プロパティを追加するには、「プロパティを追加」をクリックします。各プロパティの名前および値を指定します。有効なプロパティは次のとおりです。
   - `auditOn` - この実装クラスを有効にするかどうかを指定。有効な値は `true` および `false`。
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。

監査モジュールの編集

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化する方法の詳細については、「監査ログの有効化と無効化」を参照してください。

監査モジュールを編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「セキュリティ」ノードを開きます。
4. 「監査モジュール」ノードを開きます。
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、プロパティの名前と値を入力し、任意のプロパティをさらに入力します。有効なプロパティは次のとおりです。
   - `auditOn` - この監査モジュールを使用するかどうかを指定。有効な値は `true` および `false`。
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を変更します。
9. プロパティの左側のチェックボックスを選択して、「プロパティを削除」をクリックし、プロパティを削除します。
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。

監査モジュールの削除

監査モジュールを削除するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。

b. 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 ノードを開きます。

3. 「セキュリティ」ノードを開きます。
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。

監査ログの有効化と無効化

サーバーが使用する監査モジュールを指定するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。
   b. 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 ノードを開きます。
3. 「セキュリティ」ノードを選択します。

   「セキュリティ」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、監査モジュールのロードが行われ、監査モジュールは監査ポイントで Application Server の監査ライブラリによって確実に呼び出されます。
5. 監査ログを有効にする場合、「有効な監査モジュールの設定」で説明されているように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の設定

サーバーが使用する監査モジュールを指定するには、「監査ログの有効化と無効化」で説明されているとおり監査ログを有効にして、次の手順に従います。

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、「デフォルト監査モジュールの有効化と無効化」に説明されているとおり true に設定されていることを確認します。
2. 「保存」を選択して変更を保存するか、「デフォルトを読み込み」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の使用

`default` 監査モジュールは、サーバーログファイルへの認証および承認を記録します。ログファイルの場所の変更については、「ログの一般設定」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。

- 認証するユーザーの名前。
- アクセス要求を処理するレルム。
- 要求された Web モジュールまたは 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。

デフォルト監査モジュールの有効化と無効化

ログの有効化に加えて、必要な特定の監査モジュールで必要とされるプロパティを設定してください。デフォルトの監査モジュールの場合は、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。

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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。

b. 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 ノードを開きます。

3. 「セキュリティ」ノードを開きます。
4. 「監査モジュール」ノードを開きます。
5. 「default」ノードをクリックします。
6. auditOn プロパティの値を 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の設定

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設定を編集します。

既存の HTTP リスナーのセキュリティ設定を編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。

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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。

b. 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 ノードを開きます。

3. 「HTTP サービス」ノードを開きます。
4. 「HTTP リスナー」ノードを選択します。
5. HTTP リスナーを選択して既存のリスナーを編集するか、または「新規」をクリックして、「HTTP リスナーの作成」の手順に従って新しいリスナーを作成します。
6. 「リスナーのセキュリティプロパティの設定」の手順に従って、セキュリティプロパティを設定します。
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。

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の設定

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 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。

IIOP リスナーのセキュリティプロパティを編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス server の場合は、server-config ノードを開きます。
   b. 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 ノードを開きます。
3. 「ORB」ノードを開きます。
4. 「IIOP リスナー」ノードを選択します。
5. IIOP リスナーを選択してリスナーを編集するか、または「新規」をクリックして、「IIOP リスナーの作成」の手順に従って新しいリスナーを作成します。
6. 「リスナーのセキュリティプロパティの設定」の手順に従って、セキュリティプロパティを設定します。

7. 「保存」をクリックして変更を保存するか、または「デフォルトを読み込み」をクリックしてプロパティ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。

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の設定

管理サービスの JMX コネクタのセキュリティプロパティを編集するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
   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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。
   b. 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` ノードを開きます。
3. 「管理サービス」ノードを開きます。
4. 変更する管理サービスを選択します。
5. 「リスナーのセキュリティプロパティの設定」の手順に従って、セキュリティプロパティを設定します。
6. 「保存」をクリックして変更を保存するか、または「デフォルトを読み込み」をクリックしてプロパティのデフォルト値を復元します。

リスナーのセキュリティプロパティの設定

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プロパティを設定するには、次の共通の手順に従います。

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」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けられている場合は、「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、`slas` です。

   「証明書のニックネーム」を検索するには、「certutil ユーティリティについて」で説明している `certutil` ユーティリティを使用してください。
5. 「有効」ボックスにチェックマークが付けられている場合は、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、「暗号化方式について」で説明されています。
7. 「保存」を選択して変更を保存するか、「デフォルトを読み込み」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タスク

シングルサインオン (SSO) の設定

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、ユーザーが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場合は、すべての Web アプリケーションのセッションが無効になります。したがって、ユーザーは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ためにログインする必要があります。

Application Server では、シングルサインオンはデフォルトで有効です。シングルサインオンを無効にしたり、ほかのプロパティを設定したりするには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
2. 設定するインスタンスを開きます。
    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開きます。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを開きます。

b. 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 ノードを開きます。

3. 「HTTP サービス」ノードを開きます。
4. 「仮想サーバー」ノードを開いて、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. 「プロパティを追加」をクリックします。

   空白のプロパティ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. 「プロパティを追加」をクリックし、該当する任意の SSO プロパティを設定することで、その他のシングルサインオンのプロパティの追加または変更を行います。表 14-10 に、有効な SSO プロパティを示します。

表 14-10 仮想サーバーの SSO プロパティ

プロパティ名	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パージ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)。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パージを行う間隔 (秒単位)。	デフォルトは 60。

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マップの更新、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。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がこれに使用できます。

セキュリティマップについて

セキュリティマップを使用して、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、コネクタ接続プールに定義されたセキュリティ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、ワイルドカードの文字仕様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。

セキュリティマップの作成

コネクタ接続プールのセキュリティマップは、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、セキュリティマップを使用してください。

所定のコネクタ接続プールのセキュリティマップを作成するには、次の手順に従います。

1. 「リソース」ノードを開きます。
2. 「コネクタ」ノードを開きます。
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してコネクタ接続プールを選択するか、または「コネクタ接続プールの作成」の手順に従って、現在のプールのリストから「新規」を選択して新しい接続プールを作成します。
5. 「セキュリティマップ」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マップを作成します。
7. 「セキュリティマップ」ページで、次のプロパティを入力します。
   - **名前** - この特定のセキュリティ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。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がこれに使用できます。
   - **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。

同機能を持つ `asadmin` コマンド : `create-connector-security-map`

セキュリティマップの編集

所定のコネクタ接続プールのセキュリティマップを変更するには、次の手順に従います。

1. 「リソース」ノードを開きます。
2. 「コネクタ」ノードを開きます。
3. 「コネクタ接続プール」ノードを選択します。
4. 現在のプールから、名前を指定して「コネクタ接続プール」を選択します。
5. 「セキュリティマップ」ページを選択します。
6. 「セキュリティマップ」ページで、現在のセキュリティマップのリストからセキュリティマップを選択します。
7. 「セキュリティマップを編集」ページで、必要に応じて次のプロパティを変更します。
   - **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。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がこれに使用できます。
   - **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、セキュリティマップの変更を保存します。

参考になる asadmin コマンド : `list-connector-security-maps`、`update-connector-security-maps`

セキュリティマップの削除

所定のコネクタ接続プールのセキュリティマップを削除するには、次の手順に従います。

1. 「リソース」ノードを開きます。
2. 「コネクタ」ノードを開きます。
3. 「コネクタ接続プール」ノードを選択します。
4. 現在のプールから、名前を指定して「コネクタ接続プール」を選択します。
5. 「セキュリティマップ」ページを選択します。
6. 「セキュリティマップ」ページで、削除するセキュリティ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。

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 形式のデジタル証明書が生成されます。デフォルトで、Application Server は *domain_root_dir*/*domain_name*/`config` ディレクトリの次の 2 つのファイルに証明書情報を格納しています。

- **キーストアファイル** - デフォルトでは、`key3.db` という名前で、非公開鍵を含む Application Server のデジタル証明書が入っています。キーストアファイルはパスワードで保護されています。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、「certutil ユーティリティについて」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、Application Server キーストアにはエイリアス `s1as` を持つ単一のエントリができます。

- **トラストストアファイル** - デフォルトでは、`cert8.db` という名前で、ほかのエンティティの公開鍵を含む Application Server の信頼できる証明書が収められています。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 JSEE 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、J2EE 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の変更

開発用に準備されているキーストアおよびトラストストアファイルは、*domain_root_dir*/*domain_name*/`config` ディレクトリに格納されています。キーストアおよびトラストストアファイルの名前または場所、あるいはその両方を変更するには、次の手順に従います。

1. 管理コンソールツリーで、「設定」を開きます。
2. 「server-config (管理設定)」ノードを開きます。
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*

   NSS_database_directory は、キーストアファイルおよびトラストストアファイルが格納されているディレクトリの名前です。

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。

keytool ユーティリティについて

keytool を使用して、Platform Edition で JSSE デジタル証明書の設定および取り扱いを行います。J2SE SDK には keytool が同梱されているので、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、コマンド行で表示される必要があるツールへのパスまたは完全パスの中にあるよう、シェル環境が設定される必要があります。keytool については、次の URL にある keytool のドキュメントを参照してください。

http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html

certutil ユーティリティについて

certutil を使用して、**Enterprise Edition** でのみ NSS デジタル証明書の設定および取り扱いを行います。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コマンド行ユーティリティです。このユーティリティは、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 URL に一覧表示されているドキュメントでは、certutil ユーティリティの構文を含む、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

証明書または鍵データベースと PKCS12 形式のファイル間の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は pk12util です。pk12util ユーティリティの詳細については、次の URL を参照してください。

http://www.mozilla.org/projects/security/pki/nss/tools/pk12util.html

certutil、pk12util、およびその他の NSS セキュリティ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。

これらのツールは、*install_dir*/lib/ ディレクトリにあります。

サーバー証明書の生成

certutil 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。その実行方法の詳細については、「certutil ユーティリティについて」を参照してください。

デジタル証明書の署名

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。

CA からの証明書の使用

CA によって署名されたデジタル証明書を使用するには、次の手順に従います。

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。

   証明書をキーストアおよびトラストストアファイルを含むディレクトリに保存します。このディレクトリは、デフォルトで *domain_root_dir*/*domain-dir*/config です。この場所を変更する手順については、「証明書ファイルの場所の変更」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. certutil 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。
5. Application Server を再起動します。

certutil の使用方法の詳細については、次の URL にある certutil のドキュメントを参照してください。

http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html

証明書の削除

既存の証明書を削除するには、certutil ユーティリティを使用します。certutil ユーティリティの詳細については、「certutil ユーティリティについて」を参照してください。

詳細情報

- Java 2 Standard Edition のセキュリティの説明は、次の URL で入手できます。
  http://java.sun.com/j2se/1.4.2/docs/guide/security/index.html
- 『J2EE 1.4 Tutorial』の「Security」の章は、次の URL で入手できます。
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
- 『管理ガイド』の「メッセージセキュリティの設定」の章
- 『Developer's Guide』の「Securing Applications」の章

第 15 章

メッセージセキュリティの設定

この章では、Sun Java System Application Server 8.1 2005Q1 で Web サービスのメッセージレイヤセキュリティ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。

- メッセージセキュリティについて
- メッセージセキュリティ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、「詳細情報」に記載したリソースを参照してください。

メッセージセキュリティについて

- メッセージセキュリティの概要
- Application Server におけるメッセージセキュリティについて
- Web サービスのセキュリティ保護
- サンプルアプリケーションのセキュリティ保護
- メッセージセキュリティのための Application Server の設定

メッセージセキュリティの概要

メッセージセキュリティを使用する場合、メッセージ内にセキュリティ情報が挿入され、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セキュリティとは異なります。メッセージセキュリティを使用した場合、メッセージトランスポートからメッセージ保護が分離されるため、伝送後もメッセージは保護されたままになります。

「Web Services Security: SOAP Message Security (WS-Security)」は、米国 Sun Microsystems, Inc. を含むすべての Web サービステクノロジ主要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 仕様は次の URL で参照することができます。

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Application Server におけるメッセージセキュリティについて

Sun Java System Application Server 8.1 2005Q1 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を適用します。また、そうしたセキュリティで 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 レイヤメッセージセキュリティ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の責任の割り当て

Sun Java System Application Server 8.1 2005Q1 でメッセージセキュリティ設定の主要責任者として期待されるのは、「システム管理者」ロールと「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、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。

- システム管理者
- アプリケーション配備担当者
- 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 Web サービスセキュリティの使用方法を示したい場合だけです。

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。PE 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。SE と EE 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のタスクの概要については、「メッセージセキュリティのための 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 (つまり、`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、「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。

セキュリティトークンとセキュリティメカニズムについて

WS-Security 仕様は、セキュリティトークンを使って 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 にインストールされている SOAP レイヤメッセージセキュリティ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トークンと X509 証明書セキュリティトークンによる SOAP Web サービスメッセージの認証および暗号化を行えます。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 (つまりパスワード) 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、「レルムの編集」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セキュリティ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ります。

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、「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、「JCE プロバイダの設定」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。「詳細情報」に、この章へのリンクが掲載されています。

メッセージセキュリティ用語の解説

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。また、概念については、「メッセージセキュリティのための Application Server の設定」で説明されています。

- 認証レイヤ

  認証レイヤとは、認証処理を実行する必要のあるメッセージレイヤです。Application Server は、SOAP レイヤにおいて Web サービスメッセージセキュリティを適用します。

- 認証プロバイダ

  Sun Java Systems Application Server の今回のリリースでは、Application Server は認証プロバイダを呼び出して、SOAP メッセージレイヤのセキュリティを処理します。

  - 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 (つまり、サーブレットと EJB) 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。この順序には、メッセージ受信者が復号化を行ってから署名を検証するという想定を示すコンテンツの後で暗号化するという要件があります。

- 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。この順序には、メッセージ受信者が復号化を行ってから署名を検証するという想定を示すコンテンツの後で暗号化するという要件があります。

Web サービスのセキュリティ保護

Application Server 上に配備された Web サービスをセキュリティ保護するには、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 レイヤメッセージセキュリティ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 レイヤメッセージセキュリティ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 レイヤメッセージセキュリティ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 レイヤメッセージセキュリティプロバイダが 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、「メッセージセキュリティ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 メッセージレイヤセキュリティ設定でも、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、「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の有効化」で定義しています。

Application Server では、メッセージレイヤセキュリティ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セキュリティを設定するには、「メッセージセキュリティのための 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で保護するには、「メッセージセキュリティのためのプロバイダの有効化」と「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の有効化」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。「詳細情報」に、この章へのリンクが掲載されています。

サンプルアプリケーションのセキュリティ保護

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、単純な操作 `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先頭に `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保護する方法を示すために提供されています。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ことで WS-Security 機能が特定のアプリケーションにだけ適用されるようにする方法 (「アプリケーション固有の Web サービスセキュリティの設定」を参照) も示しています。

`xms` サンプルアプリケーションは、ディレクトリ *install_dir*¥`samples¥webservices¥security¥ejb¥apps¥xms¥` 内にインストールされます。

`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。「詳細情報」に、この章へのリンクが掲載されています。

メッセージセキュリティのための Application Server の設定

Application Server は、SOAP 処理レイヤ内に統合化されたメッセージセキュリティプロバイダを使用して、メッセージセキュリティを実装します。メッセージセキュリティプロバイダは、Application Server のその他のセキュリティ機能に依存します。

そうしたその他の機能を設定するには、次の手順に従います。

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、「JCE プロバイダの設定」で説明されています。

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、「レルムの編集」で説明されています。

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、「証明書ファイルについて」で説明されています。

Application Server の機能の設定が完了し、メッセージセキュリティプロバイダからそれらの機能を利用できるようになると、Application Server にインストールされたプロバイダを有効化できます。その手順については、「メッセージセキュリティのためのプロバイダの有効化」で説明されています。

JCE プロバイダの設定

J2SE 1.4.x に付属している Java Cryptography Extension (JCE) 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、次の手順に従って JCE プロバイダを JDK 環境の一部として静的に追加してください。

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>=< プロバイダのクラス名 >
   ```

   この例では、<*n*> は、Application Server がセキュリティ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。

メッセージレイヤセキュリティのサポートは、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、メッセージレイヤセキュリティ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。

- メッセージセキュリティのためのプロバイダの有効化
- メッセージセキュリティプロバイダの設定
- メッセージセキュリティプロバイダの作成
- メッセージセキュリティ設定の削除
- メッセージセキュリティプロバイダの削除
- 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の有効化

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のためのプロバイダの有効化

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の詳細については、「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の有効化」で説明されています。

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、メッセージレイヤセキュリティ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。

Application Server のデフォルトプロバイダを有効にするには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
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. 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」ノードを開きます。
4. 「メッセージセキュリティ」ノードを開きます。
5. 「SOAP」ノードを選択します。
6. 「メッセージセキュリティ」タブを選択します。
7. 「メッセージセキュリティ設定の編集」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、その有効化し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、「メッセージセキュリティプロバイダの設定」を参照し、この手順で有効化されたメッセージセキュリティプロバイダの設定を変更する方法を確認してください。

同機能を持つ asadmin コマンドは次のとおりです。

- 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プロバイダの設定

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も変更可能です。メッセージセキュリティプロバイダを再設定するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
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. 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」ノードを開きます。
4. 「メッセージセキュリティ」ノードを開きます。
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プロバイダを選択します。Application Server には `ClientProvider` と `ServerProvider` が同梱されています。
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.xml.wss.provider.ClientSecurityAuthModule` インタフェースを実装している必要があります。サーバー側プロバイダは、`com.sun.xml.wss.provider.ServerSecurityAuthModule` インタフェースを実装している必要があります。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。この順序には、メッセージ受信者が復号化を行ってから署名を検証するという想定を示すコンテンツの後で暗号化するという要件があります。

- **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プロバイダによってメッセージ保護ポリシーの結果として実行されるアクションについては、「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。この順序には、メッセージ受信者が復号化を行ってから署名を検証するという想定を示すコンテンツの後で暗号化するという要件があります。

    - **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プロバイダによってメッセージ保護ポリシーの結果として実行されるアクションについては、「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. 「プロパティを追加」ボタンをクリックして、プロパティ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_root_dir*/*domain_dir*/config/wss-server-config.xml。

12. 「保存」をクリックします。

同機能を持つ asadmin コマンドを次に示します。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プロバイダの作成

新しいメッセージセキュリティプロバイダを作成するには、次の手順に従います。既存のプロバイダを設定するには、「メッセージセキュリティプロバイダの設定」で説明する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」ノードを開きます。
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. 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」ノードを開きます。
4. 「メッセージセキュリティ」ノードを開きます。
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.xml.wss.provider.ClientSecurityAuthModule` インタフェースを実装している必要があります。サーバー側プロバイダは、`com.sun.xml.wss.provider.ServerSecurityAuthModule` インタフェースを実装している必要があります。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プロバイダによってメッセージ保護ポリシーの結果として実行されるアクションについては、「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プロバイダによってメッセージ保護ポリシーの結果として実行されるアクションについては、「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. 「プロパティを追加」ボタンをクリックして、プロパティ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_root_dir*/*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。

同機能を持つ asadmin コマンド : create-message-security-provider

要求および応答ポリシー設定のアクション

表 15-1 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処理を示したものです。

表 15-1 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。

表 15-1　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="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 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="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処理を一切行いません。

メッセージセキュリティ設定の削除

メッセージセキュリティ設定を削除するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
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. 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」ノードを開きます。
4. 「メッセージセキュリティ」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プロバイダの削除

メッセージセキュリティプロバイダを削除するには、次の手順に従います。

1. 管理コンソールのツリーコンポーネントで、「設定」ノードを開きます。
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. 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」ノードを開きます。
4. 「メッセージセキュリティ」ノードを開きます。
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。

同機能を持つ `asadmin` コマンド : `delete-message-security-provider`

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の有効化

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 Sun Java System 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のデフォルトクライアントプロバイダを有効にするには、次の手順に従います。

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_root_dir*/*domain_dir*/config/sun-acc.xml にある 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、次の**太字**のテキストを追加して、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。太字で表示されていないコードは、インストールによって若干異なりますが、太字で表示されている以外のテキストは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.xml.wss.provider.ClientSecurityAuthModule"
      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。この順序には、メッセージ受信者が復号化を行ってから署名を検証するという想定を示すコンテンツの後で暗号化するという要件があります。

メッセージセキュリティを実現するには、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、「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の有効化」で説明されているとおり、アプリケーションクライアントコンテナの Sun Java System 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で、次の**太字**のテキストを追加して、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。太字で表示されていないコードは、インストールによって若干異なりますが、太字で表示されている以外のテキストは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.xml.wss.provider.ClientSecurityAuthModule"
      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。これらの値のさまざまな組み合わせの結果を示す表が、「要求および応答ポリシー設定のアクション」に示されています。

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 のセキュリティの説明は、次の URL で入手できます。
  http://java.sun.com/j2se/1.4.2/docs/guide/security/index.html
- 『J2EE 1.4 Tutorial』の「Security」の章は、次の URL で入手できます。
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
- 『管理ガイド』の「セキュリティの設定」の章。
- 『Developer's Guide』の「Securing Applications」の章。
- 『Oasis Web Services Security: SOAP Message Security (WS-Security)』仕様は、次の URL で入手できます。
  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
- 『OASIS Web Services Security Username Token Profile 1.0』は、次の URL で入手できます。
  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
- 『OASIS Web Services Security X.509 Certificate Token Profile 1.0』は、次の URL で入手できます。
  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
- 『XML-Signature Syntax and Processing』ドキュメントは、次の URL で入手できます。
  http://www.w3.org/TR/xmldsig-core/
- 『XML Encryption Syntax and Processing』ドキュメントは、次の URL で入手できます。
  http://www.w3.org/TR/xmlenc-core/

第 16 章

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
- トランザクションに関する管理コンソールタスク

トランザクションについて

- トランザクションとは
- J2EE テクノロジのトランザクション

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
- アプリケーションサーバー
- リソースマネージャ
- リソースアダプタ
- ユーザーアプリケーション

これらの各エンティティは、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。
- アプリケーションサーバーは、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を提供します。
- リソースマネージャは、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。トランザクションリソースインタフェースによって実装された分散トランザクション内のリソースマネージャの関係要素が動作します。トランザクションマネージャは、トランザクションリソースインタフェースを使用して、トランザクションの関連付け、トランザクションの完了、およびリカバリと通信します。このようなリソースマネージャの例としては、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。このようなリソースアダプタの例としては、JDBC ドライバがあります。
- 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 (オプション) を検索します。このアプリケーションは、Enterprise JavaBeans の宣言によるトランザクション属性、または明示的なプログラムによるトランザクション境界を使用します。

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、次のトランザクション属性の設定方法を説明します。

- トランザクションリカバリ
- トランザクションタイムアウト
- トランザクションログ

トランザクションリカバリ

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て障害を回復することは重要です。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。

Application Server がトランザクションをリカバリする方法を設定するには、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。
   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. 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。

トランザクションタイムアウト

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。

タイムアウト値を設定するには、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。
   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. 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

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。

トランザクションログの位置を設定するには、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。
   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. 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} の値を参照するには、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。キーポイント間隔は、ログ上のキーポイント処理間のトランザクション数です。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。

キーポイント間隔を設定するには、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。

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. 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 17 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。123 ページの「Web アプリケーションの配備」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 システムでは、これらのドメインを `/etc/hosts` ファイルに追加します (`/etc/nsswitch.conf` ファイルの `hosts` の設定に `files` が含まれる場合)。

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、HTTP リスナーは、IP アドレス 0.0.0.0 を指定することで、マシンの特定のポートに設定されているすべての IP アドレスを待機できます。これとは反対に、HTTP リスナーは同一ポートを使用する特定の一意の IP アドレスを各リスナーに指定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す。このため、IP アドレスが同じでポート番号の異なる 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は同じ 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 アドレスを待機) 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。

- server という仮想サーバーに関連付けられた http-listener-1 および http-listener-2 という 2 つの HTTP リスナー。http-listener-1 ではセキュリティが無効になり、http-listener-2 ではセキュリティが有効になります。
- __asadmin という仮想サーバーに関連付けられた admin-listener という HTTP リスナー。このリスナーではセキュリティが有効になります。

これらのリスナーはどれも IP アドレス 0.0.0.0 を使用します。また、Application Server のインストール時に指定された HTTP サーバー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 リスナーに関連する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 が HTTP リスナーによってルーティングされるサーバーです。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、このサブシステムは数万個の接続を管理する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 (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、普通、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。
- 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通常の HTTP ポートから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、元の非セキュアポートが SSL 有効ポートに置き換えられた新しい 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を有効にするかどうか、あるいは、どのセキュリティ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/` (セキュリティ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 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 サービスの設定

HTTP サービスを設定するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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を設定できます。

   次の表には、これらのプロパティが一覧表示されています。

表 17-1　HTTP サービスのプロパティ

プロパティ名	説明	デフォルト値
`traceEnabled`	true に設定すると、TRACE 処理が有効化されます。このプロパティを 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

表 17-1　HTTP サービスのプロパティ (続き)

プロパティ名	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 (ミリ秒) を指定します。	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 (バイト) を指定します。	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 (ミリ秒) を指定します。	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後のスリープ時間の上限 (ミリ秒) を指定します。	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。このプロパティを false に設定すると、HTTP サービスの監視を有効化しても実際には有効にならなくなります。	true

表 17-1　HTTP サービスのプロパティ (続き)

プロパティ名	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サイズ (バイト) を指定します。	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 (秒) を指定します。	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の設定

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て、設定します。これらのログは、*domain_root_dir*/*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*.txt

「デフォルトを読み込み」をクリックして、デフォルト値を読み込みます。これらのログのローテーションプロパティを変更するには、次の手順に従います。

- 「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。
- 「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。
- 「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。

- 「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。
- 「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。

表 17-2 アクセスログ形式のトークン値

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 リクエスト行	`%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%`
データヘッダー	`%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%`

- 「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設定に戻します。

HTTP サービスの要求処理スレッドの設定

HTTP サービスの要求処理スレッドを設定する場合に、このページを使用します。

- 「デフォルトを読み込み」をクリックして、デフォルト値を読み込みます。
- 「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。
- 「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。
- 「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。
- 「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。
- 「バッファ長」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のサイズ (バイト) を入力します。デフォルトは 4096 バイトです。
- 「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の設定

HTTP サービスのキープアライブサブシステムを設定する場合に、このページを使用します。

- 「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。
- 「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。
- 「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。
- 「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。
- 「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設定に戻します。

関連項目

- HTTP リスナー

HTTP サービスの接続プールの設定

HTTP サービスの接続キューの接続プールを設定する場合に、このページを使用します。

- 「デフォルトを読み込み」をクリックして、デフォルト値を読み込みます。
- 「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。
- 「キューサイズ」フィールドに、接続キューの最大サイズ (バイト) を入力します。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。
- 「受信バッファサイズ」フィールドに、HTTP リスナーの受信バッファのサイズを入力します。デフォルトは 4096 です。
- 「送信バッファサイズ」フィールドに、HTTP リスナーの送信バッファのサイズを入力します。デフォルトは 8192 です。
- 「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設定に戻します。

HTTP サービスの HTTP プロトコルの設定

HTTP サービスの HTTP プロトコルを設定する場合に、このページを使用します。

- 「デフォルトを読み込み」をクリックして、デフォルト値を読み込みます。
- 「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。
- 「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。
- 「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。
- 「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。

- 「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。
- 「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設定に戻します。

HTTP サービスの HTTP ファイルキャッシュの設定

HTTP サービスの HTTP ファイルキャッシュを設定する場合に、このページを使用します。

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。

- 「デフォルトを読み込み」をクリックして、デフォルト値を読み込みます。
- 「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。
- 「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。
- 「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 (秒) を入力します。デフォルトは 30 秒です。
- 「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。
- 「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。
- 「ファイルサイズ上限 (中)」フィールドに、メモリ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 (バイト) を入力します。デフォルトは 537,600 バイトです。
- 「ファイルサイズ (中)」フィールドに、メモリ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 (バイト) を入力します。デフォルトは 10,485,760 バイトです。
- 「ファイルサイズ上限 (小)」フィールドに、メモリに読み込めるファイルの最大サイズ (バイト) を入力します。デフォルトは 2048 バイトです。
- 「ファイルサイズ (小)」フィールドに、メモリに読み込まれるすべてのファイルの合計サイズ (バイト) を入力します。デフォルトは 1,048,576 バイトです。

- 「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。
- 「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

仮想サーバーの作成

仮想サーバーを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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 のコピーを使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」ノードを開きます。
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することは決してしないでください。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、これが使用されます。

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_root_dir*/*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」では、仮想サーバーのプロパティを追加する場合は「プロパティを追加」をクリックします。プロパティ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 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。

13. 「了解」をクリックして、仮想サーバーを保存します。

次の表には、使用可能なプロパティが一覧表示されています。

表 17-3 仮想サーバーのプロパティ

プロパティ名	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_root_dir*/*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_root_dir*/*domain_dir*/logs/access。

表 17-3　仮想サーバーのプロパティ (続き)

プロパティ名	説明
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。

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 プロパティ 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。

同機能を持つ asadmin コマンド : create-virtual-server

仮想サーバーの編集

仮想サーバーを編集するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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」ノードを開きます。
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。
   - プロパティを、追加、削除、または変更する。
7. 「保存」をクリックして変更を保存します。

仮想サーバーの削除

仮想サーバーを削除するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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」ノードを開きます。

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。

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 リスナーの作成

HTTP リスナーを作成するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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 のコピーを使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」ノードを開きます。
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化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」フィールドには、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、次の操作を任意で行います。
    - 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場合。
      - 着信した要求に適用されるセキュリティ制約が SSL 伝送を必要とする場合。

      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化されています。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 Developer's Guide』の「Deployment Descriptor Files」の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、56 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。

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を使用するように設定できます。

安全なリスナーを設定するには、次の手順を実行します。

1. 「セキュリティ」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。
2. 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。
3. 「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」の章を参照してください。
4. 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。

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。

同機能を持つ asadmin コマンド : `create-http-listener`、`create-ssl`

HTTP リスナーの編集

HTTP リスナーを編集するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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」ノードを開きます。
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 リスナーの削除

HTTP リスナーを削除するには、次の手順に従います。

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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」ノードを開きます。
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。

同機能を持つ `asadmin` コマンド : `delete-http-listener`

第 18 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 IIOP の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属性を指定してください。詳細については、336 ページの「IIOP リスナーの作成」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 の設定

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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. 「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、340 ページの「スレッドプールの作成」および 339 ページの「Application Server のスレッドプール」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 リスナーの作成

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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. 「ORB」ノードを開きます。
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」セクションで、アプリケーションに必要なプロパティの値を指定します。
11. 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。

このページの「セキュリティ」セクションでは、リスナーが SSL、TLS、あるいはこの両方のセキュリティを使用するように設定できます。

安全なリスナーを設定するには、次の手順を実行します。

1. 「セキュリティ」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。

2. 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。
3. 「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。
4. 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。
   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。
5. 「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。

同機能を持つ asadmin コマンド : `create-iiop-listener`、`create-ssl`

IIOP リスナーの編集

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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. 「ORB」ノードを開きます。
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。変更できるフィールドの説明については、336 ページの「IIOP リスナーの作成」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 リスナーの削除

1. ツリーコンポーネントで、「設定」ノードを開きます。
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` のコピーを使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. 「ORB」ノードを開きます。
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。

同機能を持つ `asadmin` コマンド : `delete-iiop-listener`

第 19 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。

Application Server のスレッドプール

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またはアプリケーションが Application Server のすべてのスレッドを占めている場合、Application Server のスレッドを異なるスレッドプールに分割して、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

スレッドプールの作成

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。
   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. 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ワークキューの総数を入力します。
10. 「了解」をクリックします。
11. Application Server を再起動します。

同機能を持つ asadmin コマンド : create-threadpool

スレッドプールの編集

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。
   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. 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ワークキューの総数を入力します。
9. 「保存」をクリックします。
10. Application Server を再起動します。

スレッドプールの削除

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。
   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. 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。

同機能を持つ `asadmin` コマンド : `delete-threadpool`

第 20 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
- ロギングに関する管理コンソールタスク

ロギングについて

- ログレコード
- ロガー名前空間の階層

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、通常 *domain_root_dir*/*domain_dir*/logs/server.log にあるサーバーログに記録されます。

domain_root_dir/*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、320 ページの「HTTP サービスのアクセスログの設定」および 309 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-ddThh: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-ee8.1|javax.ente
rprise.system.core|_ThreadID=13;|CORE5004: Resource Deployed:
[cr:jms/DurableConnectionFactory].|#]
```

この例で、

- `[#` 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とることができます。`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。すべての Application Server の `SEVERE` および `WARNING` メッセージおよび多くの `INFO` メッセージは、モジュールコードおよび数値から構成されるメッセージ ID で始まっています。この場合は、`CORE5004` です。

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 は各モジュールのロガーを提供します。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 (348 ページの「ログレベルの設定」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 20-1 Application Server ロガー名空間

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`

表 20-1　Application Server ロガー名空間 (続き)

モジュール名	名前空間
CMP	javax.enterprise.system.container.cmp
設定	javax.enterprise.system.core.config
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	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	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

表 20-1　Application Server ロガー名空間 (続き)

モジュール名	名前空間
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

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。
   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 のコピーを使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. 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。デフォルトで、その場所は *domain_root_dir*/*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。カスタムハンドラの場合は、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、343 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。カスタムフィルタの場合は、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したら、新規に空のファイル `server.log` を作成し、古いファイルを `server.log_`*`date`* に名称変更します。ここで *`date`* はファイルがローテーションされた日付と時刻になります。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 分または複数の分数を指定すると、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。

ログレベルの設定

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。
   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` のコピーを使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. 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」セクションを使用します。プロパティ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「デフォルトを読み込み」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」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。

サーバーログの表示

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。

「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、「SEVERE 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。348 ページの「ログレベルの設定」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、次のような、その名前空間の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。

- **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 21 章

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 Sun Java System Application Server Enterprise Edition 8.1 2005Q1 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 ツールのいずれかを使用して、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、「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 ツールのどちらかを使用して、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、「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ない)。

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 --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。

- アプリケーションのツリー
- HTTP サービスのツリー
- リソースのツリー
- コネクタサービスのツリー
- JMS サービスのツリー
- ORB のツリー

- 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、「EJB コンテナの統計」と「Web コンテナの統計」を参照してください。

図 21-1　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 /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
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 /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 /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
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。「HTTP サービスのツリー」を参照してください。

図 21-2 HTTP サービスの図 (PE 版)

```
http-service
        |--- virtual-server-1
           |--- http-listener-1 *
           |--- http-listener-2 *
        |--- virtual-server-2
           |--- http-listener-1 *
           |--- http-listener-2 *
```

図 21-3 HTTP サービスの図 (EE 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。「JDBC 接続プールの統計」と「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。

図 21-4 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。「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。

図 21-5　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。

図 21-6　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。「ORB の接続マネージャの統計」を参照してください。

図 21-7　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。「スレッドプールの統計」を参照してください。

図 21-8 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
- Web コンテナの統計
- HTTP サービスの統計
- JDBC 接続プールの統計
- JMS サービスおよびコネクタサービスの統計
- ORB の接続マネージャの統計
- スレッドプールの統計
- トランザクションサービスの統計
- Java 仮想マシン (JVM) の統計
  - J2SE 5.0 の JVM 統計
- PWC (Production Web Container) の統計

EJB コンテナの統計

EJB 統計を表 21-1 に示します。

表 21-1 EJB 統計

属性名	データタイプ	説明
`createcount`	Count Statistic	特定の EJB に対する `create` メソッドの呼び出し回数。

表 21-1　EJB 統計 (続き)

属性名	データタイプ	説明
removecount	Count Statistic	特定の EJB に対する remove メソッドの呼び出し回数。
pooledcount	Range Statistic	プールされた状態にあるエンティティ Bean の数。
readycount	Range Statistic	実行可能状態にあるエンティティ Bean の数。
messagecount	Count Statistic	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	Range Statistic	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	Range Statistic	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、表 21-2 に示します。

表 21-2　EJB メソッドの統計

属性名	データタイプ	説明
methodstatistic	Time Statistic	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。
totalnumerrors	Count Statistic	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 Beans に対して収集されます。
totalnumsuccess	Count Statistic	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 Beans に対して収集されます。
executiontime	Count Statistic	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 Beans に対して収集されます。

EJB セッションストアに対する統計を、表 21-3 に示します。

表 21-3　EJB セッションストアの統計

属性名	データタイプ	説明
currentSize	Range Statistic	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	Count Statistic	ストアから活性化されたセッションの数。
activationSuccessCount	Count Statistic	ストアからの活性化に成功したセッションの数
activationErrorCount	Count Statistic	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 Beans に対して収集されます。
passivationCount	Count Statistic	このストアを使って非活性化されたセッションの数。
passivationSuccessCount	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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）	Count Statistic	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）	Count Statistic	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）	Count Statistic	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）	Value Statistic	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）	Time Statistic	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、表 21-4 に示します。

表 21-4　EJB プールの統計

属性名	データタイプ	説明
numbeansinpool	Bounded Range Statistic	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	Bounded Range Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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、表 21-5 に示します。

表 21-5　EJB キャッシュの統計

属性名	データタイプ	説明
cachemisses	Bounded Range Statistic	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	Bounded Range Statistic	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	Bounded Range Statistic	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	Count Statistic	非活性化の数。ステートフルセッション Beans にのみ適用されます。
numpassivationerrors	Count Statistic	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。
numexpiredsessionsremoved	Count Statistic	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。

表 21-5　EJB キャッシュの統計 (続き)

属性名	データタイプ	説明
numpassivationsuccess	Count Statistic	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、表 21-6 に示します。

表 21-6　タイマーの統計

統計	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、図 21-1 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表 21-7 に、Web モジュールに関して利用可能な統計を表 21-8 に、それぞれ示します。

表 21-7　Web コンテナ (サーブレット) の統計

統計	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 モジュールに対する統計を、表 21-8 に示します。

表 21-8　Web コンテナ (Web モジュール) の統計

統計	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。

表 21-8　Web コンテナ (Web モジュール) の統計 (続き)

統計	データタイプ	補足説明
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)	AverageRange Statistic	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)	AverageRange Statistic	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。
sessionpersisttime (EE のみ)	AverageRange Statistics	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、表 21-9 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、表 21-32 を参照してください。

表 21-9　HTTP サービスの統計 (Platform Edition のみに適用)

統計	単位	データタイプ	補足説明
bytesreceived	バイト	Count Statistic	各要求プロセッサが受信したバイトの累計値。
bytessent	バイト	Count Statistic	各要求プロセッサが送信したバイトの累計値。
currentthreadcount	数値	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	存在可能な未使用応答処理スレッドの最大数。
minsparethreads	数値	Count Statistic	存在可能な未使用応答処理スレッドの最小数。
maxthreads	数値	Count Statistic	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	Count Statistic	スレッド処理時間の最大値。
processing-time	ミリ秒	Count Statistic	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	Count Statistic	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、表 21-10 に示します。

表 21-10　JDBC 接続プールの統計

統計	単位	データタイプ	説明
numconnfailedvalidation	数値	Count Statistic	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	Range Statistic	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	Range Statistic	Count Statistic	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	Count Statistic	Bounded Range Statistic	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	Count Statistic	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	Count Statistic	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		Range Statistic	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	Count Statistic	前回のリセット後に破棄された物理接続の数。

表 21-10　JDBC 接続プールの統計 (続き)

統計	単位	データタイプ	説明
numconnacquired	数値	Count Statistic	プールから取得された論理接続の数。
numconnreleased	数値	Count Statistic	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、表 21-11 に示します。コネクタ作業管理に対する統計を、表 21-12 に示します。

表 21-11　コネクタ接続プールの統計

統計	単位	データタイプ	説明
numconnfailed validation	数値	Count Statistic	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	Range Statistic	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	Range Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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。
connectionrequest waittime		Range Statistic	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。

表 21-11　コネクタ接続プールの統計 (続き)

統計	単位	データタイプ	説明
numconndestroyed	数値	Count Statistic	前回のリセット後に破棄された物理接続の数。
numconnacquired	数値	Count Statistic	プールから取得された論理接続の数。
numconnreleased	数値	Count Statistic	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、表 21-12 に示します。

表 21-12　コネクタ作業管理の統計

統計	データタイプ	説明
activeworkcount	Range Statistic	コネクタによって実行された作業オブジェクトの数。
waitqueuelength	Range Statistic	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	Range Statistic	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	Count Statistic	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	Count Statistic	Application Server によって拒否された作業オブジェクトの数。
completedworkcount	Count Statistic	完了した作業オブジェクトの数。

ORB の接続マネージャの統計

ORB の接続マネージャに関して利用可能な統計を、表 21-13 に示します。

表 21-13　ORB の接続マネージャの統計

統計	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。

表 21-13　ORB の接続マネージャの統計 (続き)

統計	単位	データタイプ	説明
totalconnections	数値	BoundedRange Statistic	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、表 21-14 に示します。

表 21-14　スレッドプールの統計

統計	単位	データタイプ	説明
averagetimeinqueue	ミリ秒	RangeStatistics	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	RangeStatistics	1 つの作業の平均完了時間 (ミリ秒)。
currentnumberofthreads	数値	BoundedRange Statistic	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、表 21-15 に示します。

表 21-15　トランザクションサービスの統計

統計	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。

表 21-15　トランザクションサービスの統計 (続き)

統計	データタイプ	説明
activeids	String Statistic	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	String Statistic	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、表 21-16 に示します。

表 21-16　JVM の統計

統計	データタイプ	説明
heapsize	BoundedRange Statistic	JVM のメモリヒープサイズの上限と下限の間にある常駐メモリフットプリント。
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。この文書は次の URL で利用可能になっています。

http://java.sun.com/j2se/1.5.0/docs/guide/management/

J2SE 5.0 の監視ツールについては、次の URL を参照してください。

http://java.sun.com/j2se/1.5.0/docs/tooldocs/#manage

J2SE 5.0 の JVM で利用可能なクラス読み込み関連の統計を、表 21-17 に示します。

表 21-17　J2SE 5.0 の JVM 統計 - クラス読み込み

統計	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、表 21-18 に示します。

表 21-18　J2SE 5.0 の JVM 統計 - コンパイル

統計	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、表 21-19 に示します。

表 21-19　J2SE 5.0 の JVM 統計 - ガベージコレクション

統計	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関連の統計を、表 21-20 に示します。

表 21-20　J2SE 5.0 の JVM 統計 - メモリ

統計	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。
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。

表 21-20　J2SE 5.0 の JVM 統計 - メモリ (続き)

統計	データタイプ	説明
maxheapsize	CountStatistic	メモリ管理用として使用可能なメモリの最大サイズ (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	メモリ管理用として使用可能なメモリの最大サイズ (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、表 21-21 に示します。

表 21-21　J2SE 5.0 の JVM 統計 - オペレーティングシステム

統計	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。
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、表 21-22 に示します。

表 21-22　J2SE 5.0 の JVM 統計 - ランタイム

統計	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。

表 21-22　J2SE 5.0 の JVM 統計 - ランタイム (続き)

統計	データタイプ	説明
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、表 21-23 に示します。

表 21-23　J2SE 5.0 の JVM 統計 - ThreadInfo

統計	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。
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。

表 21-23　J2SE 5.0 の JVM 統計 - ThreadInfo (続き)

統計	データタイプ	説明
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、表 21-24 に示します。

表 21-24　J2SE 5.0 の JVM 統計 - スレッド

統計	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。
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。

- 表 21-25、PWC 仮想サーバー
- 表 21-26、PWC 要求
- 表 21-27、PWC ファイルキャッシュ
- 表 21-28、PWC キープアライブ
- 表 21-29、PWC DNS
- 表 21-30、PWC スレッドプール
- 表 21-31、PWC 接続キュー
- 表 21-32、PWC HTTP サービス

PWC 仮想サーバーの統計を、表 21-25 に示します。

表 21-25　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	String Statistic	仮想サーバーの ID。
mode	String Statistic	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	String Statistic	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	String Statistic	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、表 21-26 に示します。

表 21-26　PWC 要求の統計 (EE のみ)

属性名	データタイプ	説明
method	String Statistic	要求で使用されたメソッド。
uri	String Statistic	最後に処理された URI。
countrequests	Count Statistic	処理された要求の数。

表 21-26　PWC 要求の統計 (EE のみ) (続き)

属性名	データタイプ	説明
countbytestransmitted	Count Statistic	送信バイト数。この情報が利用不可能な場合は 0
countbytesreceived	Count Statistic	受信バイト数。この情報が利用不可能な場合は 0。
ratebytesreceived	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	コード 2XX の応答の合計数。
count3xx	Count Statistic	コード 3XX の応答の合計数。
count4xx	Count Statistic	コード 4XX の応答の合計数。
count5xx	Count Statistic	コード 5XX の応答の合計数。
countother	Count Statistic	その他の応答コードを含む応答の合計数。
count200	Count Statistic	コード 200 の応答の合計数。
count302	Count Statistic	コード 302 の応答の合計数。
count304	Count Statistic	コード 304 の応答の合計数。
count400	Count Statistic	コード 400 の応答の合計数。
count401	Count Statistic	コード 401 の応答の合計数。

表 21-26　PWC 要求の統計 (EE のみ) (続き)

属性名	データタイプ	説明
count403	Count Statistic	コード 403 の応答の合計数。
count404	Count Statistic	コード 404 の応答の合計数。
count503	Count Statistic	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の統計を、表 21-27 に示します。

表 21-27　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	Count Statistic	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	Count Statistic	有効なキャッシュエントリの最大有効期間 (秒)。
countentries	Count Statistic	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	Count Statistic	同時に存在可能なキャッシュエントリの最大数。
countopenentries	Count Statistic	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	Count Statistic	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	Count Statistic	キャッシュコンテンツ格納用のヒープ領域。
maxheapcachesize	Count Statistic	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	Count Statistic	メモリ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	Count Statistic	ファイルキャッシュがメモリ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。

表 21-27　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
counthits	Count Statistic	キャッシュ検索の成功回数。
countmisses	Count Statistic	キャッシュ検索の失敗回数。
countinfohits	Count Statistic	ファイル情報検索の成功回数。
countinfomisses	Count Statistic	キャッシュファイル情報検索の失敗回数。
countcontenthits	Count Statistic	キャッシュファイルコンテンツのヒット数。
countcontentmisses	Count Statistic	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、表 21-28 に示します。

表 21-28　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	Count Statistic	キープアライブモードの接続の数。
maxconnections	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	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、表 21-29 に示します。

表 21-29　PWC DNS の統計 (EE のみ)

属性名	データタイプ	説明
flagcacheenabled	Count Statistic	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	Count Statistic	キャッシュ内に現在存在している DNS エントリの数。
maxcacheentries	Count Statistic	キャッシュ内に格納できる DNS エントリの最大数。
countcachehits	Count Statistic	DNS キャッシュ検索の成功回数。
countcachemisses	Count Statistic	DNS キャッシュ検索の失敗回数。
flagasyncenabled	Count Statistic	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	Count Statistic	非同期 DNS 名検索の合計回数。
countasyncaddrlookups	Count Statistic	非同期 DNS アドレス検索の合計回数。
countasynclookupsinprogress	Count Statistic	処理中の非同期検索の数。

PWC スレッドプールの統計を、表 21-30 に示します。

表 21-30　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	String Statistic	スレッドプールの ID。
countthreadsidle	Count Statistic	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	Count Statistic	要求処理スレッドの現在の数。

表 21-30　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
maxthreads	Count Statistic	同時に存在可能な要求処理スレッドの最大数。
countqueued	Count Statistic	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	Count Statistic	キュー内に同時に格納された要求の最大数。
maxqueued	Count Statistic	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 接続キューの統計を、表 21-31 に示します。

表 21-31　PWC 接続キューの統計 (EE のみ)

属性名	データタイプ	説明
id	String Statistic	接続キューの ID。
counttotalconnections	Count Statistic	受け付けられた接続の合計数。
countqueued	Count Statistic	キュー内に現在存在している接続の数。
peakqueued	Count Statistic	キュー内に同時に存在していた接続の最大数。
maxqueued	Count Statistic	接続キューの最大サイズ。
countoverflows	Count Statistic	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	Count Statistic	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	Count Statistic	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。

表 21-31　PWC 接続キューの統計 (EE のみ) (続き)

属性名	データタイプ	説明
countqueued1minuteaverage	Count Statistic	キュー内接続数の過去 1 分間における平均値。
countqueued5minuteaverage	Count Statistic	キュー内接続数の過去 5 分間における平均値。
countqueued15minuteaverage	Count Statistic	キュー内接続数の過去 15 分間における平均値。

PWC HTTP サービスの統計を、表 21-32 に示します。

表 21-32　PWC HTTP サービスの統計 (EE のみ)

属性名	データタイプ	説明
id	String Statistic	HTTP サービスのインスタンス名。
versionserver	String Statistic	HTTP サービスのバージョン番号。
timestarted	String Statistic	HTTP サービスが起動された時刻 (GMT)。
secondsrunning	Count Statistic	HTTP サービス起動後の経過時間 (秒)。
maxthreads	Count Statistic	各インスタンス内のワークスレッドの最大数。
maxvirtualservers	Count Statistic	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	Count Statistic	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	Count Statistic	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	Count Statistic	要求負荷の過去 1 分間における平均値。
load5minuteaverage	Count Statistic	要求負荷の過去 5 分間における平均値。

表 21-32　PWC HTTP サービスの統計 (EE のみ) (続き)

属性名	データタイプ	説明
load15minuteaverage	Count Statistic	要求負荷の過去 15 分間における平均値。
ratebytestransmitted	Count Statistic	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	Count Statistic	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た監視レベルの設定

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. ツリーで、「設定」ノードを開きます。
   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、常時監視可能な Java 仮想マシン (JVM) 以外のすべてのコンポーネントや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、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」は存在しないので、「追加プロパティ」の表は無視してかまいません。

同機能を持つ `asadmin` コマンド : `set`。HTTP サービスの監視機能をオンにするときには、次の `asadmin` コマンドを使用してください。

```
asadmin> set --user admin_user
server.monitoring-service.module-monitoring-levels.http-service=LOW
```

asadmin ツールを使用した監視の設定

特定のコンポーネントまたはサービスに対して、その監視をオフにしたり、特定の監視レベルを設定したりするには、「管理コンソールを使用した監視レベルの設定」で説明した管理コンソールを使用できますが、この節で説明する `asadmin` ツールを使用することもできます。

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

- 管理コンソールでの監視データの表示
- asadmin ツールによる監視データの表示

管理コンソールでの監視データの表示

Application Server 管理コンソールを使用してサーバーインスタンスに配備されているコンポーネントまたはサービスの監視データを表示するには、次の手順に従います。各コンポーネントまたはサービスの属性の詳細については、「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、`server (Admin Server)` などの「スタンドアロンインスタンス」ノードを開きます。
   b. リストから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については、「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、「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、「管理コンソールを使用した監視レベルの設定」または「asadmin ツールを使用した監視の設定」を参照してください。

4. 「監視」ページの「アプリケーション」タブを選択して、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については、「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、「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、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については、「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、「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。トランザクションサービスの監視を有効にするには、「監視を設定」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。

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ときには、次の asadmin コマンドを使用してください。

```
asadmin> get --monitor server.jvm.*
```

asadmin ツールによる監視データの表示

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。asadmin ツールを使用して監視データを表示する一般的な設定をするには、次の手順に従います。

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 --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例については、「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、「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 --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例については、「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、「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 の list コマンドと 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名前はドット (.) 文字を区切り文字として使って指定されるため、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、単独タイプでない子ノードであるアプリケーション PetStore 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。この bytesreceived 属性を指定する名前は、次のようになります。

```
server.http-service.server.http-listener-1.
  bytesreceived-lastsampletime
```

管理者は、asadmin の list コマンドと 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、特定の監視可能オブジェクトで利用可能なすべて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ます。たとえば、list --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ます。
- 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ます。たとえば、list --monitor server.applications.* を実行した場合、applications のすべての子ノードに加え、それらの子ノード以下の階層にある子ノードもすべて一覧表示されます。
- list コマンドでドット表記名の前後に *dottedname、*dotted*name、dotted name** のいずれかの形式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、結果として返されます。
- 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、結果として返されます。

詳細については、「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。

- list --monitor コマンドの例
- get --monitor コマンドの例
- Petstore の例

list --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と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、「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 --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 --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、「Petstore の例」を参照してください。

get --monitor コマンドの例

このコマンドで取得できる監視対象情報は、次のとおりです。

- 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、特定の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、「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 --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 --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 --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 --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 の例

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たがっています。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=4848
   ```

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 --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 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 Bean `UserEJB` のメソッド `getUserName` 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.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 21-33　トップレベル

コマンド	ドット表記名	出力

`list -m`	`server`	`server.applications`
		`server.thread-pools`
		`server.resources`
		`server.http-service`
		`server.transaction-service`
		`server.orb.connection-managers`
		`server.orb.connection-managers.orb¥.Connections¥.In bound¥.`
		`AcceptedConnections`
		`server.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。

表 21-34 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 21-34 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`
		`appl1`
		`app2`
		`web-module1_war`
		`ejb-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。

表 21-35 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 21-35 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 : 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_jar`
		`web-module2_war`
		`ejb-module3_jar`
		`web-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。

表 21-35 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 または *ejb-module1_jar または server.applications.ejb-module1_jar	bean1 bean2 bean3 ...
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。
list -m	server.applications.app1.ejb-module1_jar.bean1 **注** : 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-pool bean-cache bean-method
list -m	server.applications.app1.ejb-module1_jar.bean1 **注** : 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。

表 21-35 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
get -m	server.applications.app1.ejb-module1_jar.bean1.*	次の属性とそれらの関連付けられた値が表示されます。
	注 : 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。	CreateCount_Count CreateCount_Description CreateCount_LastSampleTime CreateCount_Name CreateCount_StartTime CreateCount_Unit MethodReadyCount_Current MethodReadyCount_Description MethodReadyCount_HighWaterMark MethodReadyCount_LastSampleTime MethodReadyCount_LowWaterMark MethodReadyCount_Name MethodReadyCount_StartTime MethodReadyCount_Unit RemoveCount_Count RemoveCount_Description RemoveCount_LastSampleTime RemoveCount_Name RemoveCount_StartTime RemoveCount_Unit
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-pool	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 : 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-pool.*	表 1-4 で説明した EJB プール属性に対応する属性と値の一覧。
	注 : 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-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 : 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 21-35 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
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 : 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 : 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 : 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 1-7 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。

表 21-36 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 21-36 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。

表 21-37 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 21-37 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 1-14 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。

表 21-38 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 21-38 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。

表 21-39 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 21-39 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。

表 21-40 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 21-40 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。
`list -m`	`server.orb.connection-managers`	1 つまたは複数の ORB 接続マネージャ名。
`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。

表 21-40　ORB レベル (続き)

コマンド	ドット表記名	出力
list -m	server.orb.connection-managers.orb¥.Connections¥.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¥.Connections¥.Inbound¥.AcceptedConnections.*	表 1-13 で説明した ORB 接続マネージャ属性に対応する属性と値の一覧。

表 21-34 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 21-41　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を無効にするには、次の方法のいずれかを使用します。

1. 管理コンソールを使って JMX コネクタのセキュリティを無効にします。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」を開きます。
   b. 「ノードエージェント」を選択します。
   c. JMX タブの SSL セクションで、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。

PE 版では、JMX コネクタがデフォルトで無効になっています。このため、PE 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 : `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 22 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

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、この設定を行うときは注意が必要です。

Application Server が使用する JVM の一般設定を行うには、次の手順を実行します。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 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 (「詳細情報」セクションのリンクを参照)。

   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。

   EJB コンポーネントの配備時に Application Server は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 Developer's Guide』の「Debugging J2EE Applications」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、「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 Developer's Guide』の「Debugging J2EE Applications」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 クラスパス設定の設定

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。

Application Server の JVM クラスパスを設定するには、次の操作を行います。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 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 のクラスパスを調べるには、「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確認します。

   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。「JDBC ドライバの統合」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 オプションの設定

「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を指定します。

JVM オプションを設定するには、次の操作を行います。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 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、次の URL を参照してください。

- http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html
- http://java.sun.com/docs/hotspot/VMOptions.html

セキュリティマネージャの無効化

Application Server のセキュリティマネージャを無効にすると、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 承認および認証の機能は、セキュリティマネージャが無効になっている場合でもなお機能します。開発環境ではセキュリティマネージャを無効にできますが、実稼動環境では決してセキュリティマネージャを無効にしないでください。

セキュリティマネージャを無効にするには、次の手順に従います。

1. 管理コンソールの「JVM オプション」ページに進みます。

   詳細については、「JVM オプションの設定」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 プロファイラ設定の設定

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。

Application Server のプロファイラを設定するには、次の操作を行います。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 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 Developer's Guide』の「Debugging J2EE Applications」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、「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

詳細ドメイン属性の設定

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」ノードを開き、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。

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ます。Sun Java System 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、『Sun Java System 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。

最小要件

この節では、Apache 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 で実行されるようにコンパイルし、構築する必要があります。

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

Microsoft Windows プラットフォームの要件については、次の URL を参照してください。
http://httpd.apache.org/docs/windows.html#req
http://httpd.apache.org/docs/win_compiling.html

その他の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、以下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、そのバージョンは `gcc` 3.0 以降である必要があります。

注

- その他の C 言語のコンパイラを使用するには、その C 言語のコンパイラのパスを設定して、PATH 環境変数のユーティリティを使用可能にします。次に例を示します。

  ```
  exportLD_LIBRARY_PATH=$LD_LIBRARY_PATH:appserver_installdir/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

Microsoft Windows プラットフォームの要件については、次の URL を参照してください。

http://httpd.apache.org/docs-2.0/platform/windows.html

その他の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)
- `httpd-2.0.49` (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、以下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、そのバージョンは gcc 3.0 以降である必要があります。

注

- その他の C 言語のコンパイラを使用するには、その C 言語のコンパイラのパスを設定して、PATH 環境変数のユーティリティを使用可能にします。次に例を示します。

  ```
  export
  LD_LIBRARY_PATH=app_server_install_dir/lib:$LD_LIBRARY_PATH
  ```

  この例は `sh` に対するものです。
- 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 のインストール

Microsoft Windows プラットフォームで Apache をコンパイルし、インストールする手順については、次の Web サイトを参照してください。

Apache 1.3 の場合 :

http://httpd.apache.org/docs/win_compiling.html

Apache 2 の場合 :

http://httpd.apache.org/docs-2.0/platform/win_compiling.html

その他のプラットフォームで SSL 対応の Apache Web サーバーをコンパイルし、設定し、インストールするには、次の手順に従います。この例では Apache 1.3.29 のコンパイルとビルドが説明されていますが、同じ手順が Apache 2 にも適用されます。

注 mod_ssl、OpenSSL、および Apache の展開は、同じディレクトリレベルで行ってください。

Open SSL のコンパイルとビルド

Linux とともにインストールされた OpenSSL のバージョンが 0.9.7d である場合、その Linux 上で次の手順を実行する必要はありません。

OpenSSL の詳細については、次を参照してください。

http://www.openssl.org/

openssl-0.9.7d ソースをアンパックし、次の手順を実行します。

1. `cd openssl-0.9.7d`
2. `./config`
3. `make`
4. `make test`
5. `make install`

OpenSSL をソースからビルドする方法の詳細については、openssl ディレクトリ内の INSTALL ファイルを参照してください。

mod_ssl による Apache の設定

この節の内容は Apache 1.3 だけに当てはまります。Apache 2.0 をインストールする場合は、ここを飛ばして 414 ページの「Apache のコンパイルとビルド」に進んでください。

mod_ssl の詳細については、次を参照してください。

http://www.modssl.org/

1. apache_1.3.29 ソース配付をダウンロードします。

   このソース配布をアンパックします。ソース配布は圧縮されたアーカイブとして提供されます。apache_1.3.29 の場合、ソース配付のアーカイブ名は、apache_1.3.29.tar.gz となります。

2. 次のコマンドを使用して、アーカイブを解凍して展開します。

   ```
   tar -zxvf apache_1.3.29.tar.gz
   ```

   このコマンドを実行すると、apache_1.3.29 という名前のディレクトリが現在の作業用ディレクトリ内に作成されます。

3. mod_ssl-2.8.14-1.3.29 ソース をアンパックします。

4. `cd mod_ssl-2.8.14-1.3.29`

5. `./configure --with-apache=../apache_1.3.29 --with-ssl=../openssl-0.9.7d --prefix=`*install path* `--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。このコマンドを実行すると、画面上に数行のメッセージが出力されます。

このコマンドを実行すると、システム設定に基づいてビルドに使用する make ファイルが作成されます。configure でエラーが発生した場合、一部のヘッダーファイルやユーティリティプログラムがなくなることがあります。処理を続行する前にそれらをインストールしてください。

Apache のコンパイルとビルド

Apache をコンパイルおよびビルドする手順は、Apache のバージョンごとに異なります。

- Apache 1.3 のコンパイルとビルド
- Apache 2 のコンパイルとビルド

Apache 1.3 のコンパイルとビルド

この手順では、413 ページの「mod_ssl による Apache の設定」で説明した `--prefix` 属性で指定された場所に Apache をインストールします。

1. Linux の場合、`src/MakeFile` の `End of automatically generated section` の後に次の行を含めます。

   ```
   LIBS+= -licuuc -licui18n -lnspr4 -lpthread -lxerces-c -lsupport
   -lnsprwrap -lns-httpd40
   ```

   ```
   LDFLAGS+= -L/appserver_installdir/lib
   ```

2. Linux の場合、Application Server のインストールディレクトリを `LD_LIBRARY_PATH` に含めます。

   ```
   export LD_LIBRARY_PATH=/app_server_install_dir/lib:$LD_LIBRARY_PATH
   ```

3. 次の手順に従って、`make` コマンドを使用して Apache をコンパイルします。

   a. `mod_ssl` ディレクトリへの移動。

   b. `make`

   c. `make certificate`

   d. `make install`

   注 `make certificate` コマンドは、セキュリティ保護されたパスワードを要求します。このパスワードは、セキュリティ保護された Apache を起動するために必要です。忘れないでください。

   `make install` コマンドは、プロセスが Apache ソースコードをコンパイルし、Apache をリンクしていることを示す複数行の出力を画面に表示します。通常、このプロセスはエラーを発生せずに終了します。ただし、何らかのエラーが発生した場合は、Apache のすべてのライブラリファイルとユーティリティプログラムが正しくダウンロードされているかどうかを確認してください。

   使用する環境に適した値を *apache_install_path*`/conf/httpd.conf` ファイルに入力して、Apache のインストールを設定します。

Apache 2 のコンパイルとビルド

1. Apache 2_0_*NN* のソース配付をダウンロードします。

   NN は、「52」などのマイナーバージョンを表します。

2. このソース配布をアンパックします。

   ソース配布は圧縮されたアーカイブとして提供されます。Apache 2_0_*NN* の場合、ソース配付のアーカイブ名は、`httpd-2_0_NN.tar.gz` となります。

3. 次のコマンドを使用して、アーカイブを解凍して展開します。

   ```
   tar -zxvf httpd-2_0_NN.tar.gz
   ```

   このコマンドを実行すると、`httpd-2_0_NN` という名前のディレクトリが現在の作業用ディレクトリ内に作成されます。

4. `cd httpd-2_0_NN` を実行します。

5. `./configure --with-ssl=`*open_ssl_install_path* `--prefix=`*install_path* `--enable-ssl --enable-so` を実行します。

6. Linux 場合、*apache_src*`/build/config_vars.mk` を変更し、次の行を追加します。

   ```
   EXTRA_LIBS += -licuuc -licui18n -lnspr4 -lpthread -lxerces-c
   -lsupport -lnsprwrap -lns-httpd40

   LDFLAGS+=-L<appserver install dir>/lib
   ```

7. Linux の場合、Application Server のインストールディレクトリを `LD_LIBRARY_PATH` に含めます。

   ```
   export LD_LIBRARY_PATH=/app_server_install_dir/lib:$LD_LIBRARY_PATH
   ```

8. 次の手順に従って、`make` コマンドを使用して Apache をコンパイルします。

   `httpd-2_0_NN` ディレクトリから次のコマンドを実行します。

   a. `make`

   b. `make install`

   `make install` コマンドは、プロセスが Apache ソースコードをコンパイルし、Apache をリンクしていることを示す複数行の出力を画面に表示します。通常、このプロセスはエラーを発生せずに終了します。ただし、何らかのエラーが発生した場合は、Apache のすべてのライブラリファイルとユーティリティプログラムが正しくダウンロードされているかどうかを確認してください。

   使用する環境に適した値を *apache_install_path*`/conf/httpd.conf` ファイルに入力して、Apache のインストールを設定します。

注 エラーが発生した場合、Application Server インストールディレクトリを PATH に含めてみてください。

```
export LD_LIBRARY_PATH=/app_server_install_dir/lib
```

あるいは、OpenSSL ライブラリを追加します。次に例を示します。

```
export
LD_LIBRARY_PATH=/openssl_install_dir/lib:/app_server_install_dir/lib
```

注 Apache 2 の場合、証明書を手動で作成およびインストールする必要があります。詳細については、Apache のマニュアルを参照してください。

Apache の起動と停止

Apache には、Apache の起動、停止、および再起動を容易にする `apachectl` という名前のスクリプトが含まれています。

- 次のコマンドを実行して、Apache を起動します。

  apache_install_dir`/bin/apachectl start`

- 次のコマンドを実行して、Apache を SSL モードで起動します。

  apache_install_dir`/bin/apachectl startssl`

- Apache を停止するには、次のコマンドを実行します。

  apache_install_dir`/bin/apachectl stop`

Apache を起動したら、インストールをテストします。Apache の起動後、「http://*server_name*:*port_number*/」というアドレスを Web ブラウザから入力します。インストールに成功し、Apache が実行されている場合は、テストページが表示されます。

Apache のインストールが完了したら、75 ページの「Apache Web サーバーに対する変更」を参照し、プラグインのインストール中とインストール後における Apache 設定に関する情報を確認してください。

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、次のテキストを使用します。

```
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、次のテキストを使用します。

```
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、C:¥winnt¥system32 のいずれかのディレクトリ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、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 と 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されない場合、そのサービスは正しく動作しません。

注 asadmin の開始 / 停止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

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。
  a. 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。
  b. 「プロパティ」ウィンドウの「ログオン」タブをクリックします。
  c. 「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最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。

  a. 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
  ```

  b. Windows の場合、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
  ```

  c. UNIX の場合、`inittab` ファイルに追加した行内で、`--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
  ```

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。

1. 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。
2. サーバー、設定、クラスタ、またはノードエージェントに「domain」という名前を付けてはいけません。
3. サーバーインスタンスに「agent」という名前を付けてもかまいません。

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。

表 C-1　トップレベル要素

要素名	ドット表記名プレフィックス
applications	domain.applications
resources	domain.resources
configurations	domain.configs

表 C-1　トップレベル要素 (続き)

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。
clusters	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.note-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 にマッピングされます (server1 は特定のサーバーインスタンスを表す)。
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。

表 C-2　ドメイン配下のドット表記名サーバー名

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。

表 C-3　サーバーまたはクラスタによって参照されている設定のドット表記名

ドット表記名	別名
target.http-service	*config-name*.http-service
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、別名を使用すべきではありません。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を取得する際のドット表記名属性は、domain.servers.clustered-instance-name.*system-property* のように記述すべきです。clustered-instance-name.*system-property* のように記述すべきではありません。

索引

E

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

あ

い

え

お

か

き

く

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

ね

の

は

ひ

ふ

へ

ほ

ま

め

も

よ

ら

り

る

れ

ろ

わ