Sun Java System Application Server Enterprise Edition 8.2 管理ガイド

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

Part No: 820–0850
2007 年 3 月

070530@17466

目次

図目次

表目次

例目次

はじめに

Sun Java System Application Server Enterprise Edition 8.2 管理者ガイド では、設定、監視、セキュリティー、リソース管理、および Web サービス管理を含む Application Server のシステム管理について説明します。

ここでは、Sun Java™ System Application Server のマニュアルセット全体の内容と表記規則について説明します。

Application Server のマニュアルセット

Application Server のマニュアルセットは、配備の計画とシステムのインストールについて説明しています。スタンドアロンの Application Server のマニュアルの Uniform Resource Locator (URL) は、`http://docs.sun.com/app/docs/coll/1310.4` です。Application Server の概要については、次の表に示している順に各マニュアルを参照してください。

表 P–1　Application Server のマニュアルセットの内容

マニュアル名	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。サポートされるハードウェア、オペレーティングシステム、Java Development Kit (JDK™)、およびデータベースドライバの包括的な表ベースの概要を含みます。
『Quick Start Guide』	Application Server 製品の使用を開始するための手順。
『Installation Guide』	ソフトウェアとそのコンポーネントのインストール。
『配備計画ガイド』	最適な方法で確実に Application Server を導入するための、システムニーズや企業ニーズの分析。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。
『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。開発ツール、セキュリティー、デバッグ、配備、ライフサイクルモジュールの作成などについての情報も提供します。

表 P–1 Application Server のマニュアルセットの内容 (続き)

マニュアル名	説明
『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。
『管理ガイド』	管理コンソールからの、Application Server サブシステムおよびコンポーネントの設定、管理、および配備。
『高可用性 (HA) 管理ガイド』	高可用性データベースのための、インストール後の設定と管理に関する解説。
『Administration Reference』	Application Server 設定ファイル `domain.xml` の編集。
『アップグレードと移行』	新しい 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 または 7 からの移行)。このマニュアルでは、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。
『パフォーマンスチューニングガイド』	パフォーマンスを向上させるための Application Server の調整。
『トラブルシューティングガイド』	Application Server の問題の解決。
『Error Message Reference』	Application Server のエラーメッセージの解決。
『Reference Manual』	Application Server で使用できるユーティリティーコマンド。マニュアルページのスタイルで記述されています。`asadmin` コマンド行インタフェースも含みます。

関連マニュアル

Application Server は、単体で購入することが可能です。あるいは、ネットワークまたはインターネット環境にわたって分散しているエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャーである Java ES のコンポーネントとして購入することもできます。Application Server を Java ES のコンポーネントとして購入した場合は、`http://docs.sun.com/coll/1286.2` にあるシステムマニュアルをよく読むことをお勧めします。Java ES およびそのコンポーネントに関するすべてのマニュアルの URL は `http://docs.sun.com/prod/entsys.5` です。

その他の Sun Java System サーバーのマニュアルとしては、次のマニュアルを参照してください。

- Message Queue のマニュアル
- Directory Server のマニュアル
- Web Server のマニュアル

さらに、次のリソースが役立つことがあります。

- 『J2EE 1.4 Specifications (`http://java.sun.com/j2ee/1.4/docs/index.html`)』
- 『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)』

デフォルトのパスとファイル名

次の表は、このマニュアルで使用するデフォルトのパスやファイル名について説明したものです。

表 P–2 デフォルトのパスとファイル名

プレースホルダ	説明	デフォルト値

install-dir	Application Server のベースインストールディレクトリを表します。	Solaris™ プラットフォームへの Sun Java Enterprise System (Java ES) インストールの場合:
		`/opt/SUNWappserver/appserver`
		Linux プラットフォームへの Java ES 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	すべてのドメインを含むディレクトリを表します。	Solaris プラットフォームへの Java ES インストールの場合:
		`/var/opt/SUNWappserver/domains/`
		Linux プラットフォームへの Java ES インストールの場合:
		`/var/opt/sun/appserver/domains/`
		そのほかのすべてのインストールの場合:
		install-dir`/domains/`

表 P–2 デフォルトのパスとファイル名 (続き)

プレースホルダ	説明	デフォルト値
domain-dir	ドメインのディレクトリを表します。 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}`	*domain-root-dir*/*domain-dir*
instance-dir	サーバーインスタンスのディレクトリを表します。	*domain-dir*/*instance-dir*

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–3 表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.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。 `machine_name% you have mail.`
`AaBbCc123`	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区別して示します。	`machine_name%` **`su`** `Password:`
aabbcc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャー・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun%` **`grep '^#define \`** **`XV_VERSION_STRING'`**

コード例は次のように表示されます。

- C シェル

  ```
  machine_name% command y|n [filename]
  ```

- C シェルのスーパーユーザー

  ```
  machine_name# command y|n [filename]
  ```

- Bourne シェルおよび Korn シェル

  ```
  $ command y|n [filename]
  ```

- Bourne シェルおよび Korn シェルのスーパーユーザー

  ```
  # command y|n [filename]
  ```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

記号の規則

次の表は、この用語集で使用される記号の一覧です。

表 P–4 記号の規則

記号	説明	例	意味
[ ]	省略可能な引数やコマンドオプションが含まれます。	ls [-l]	-l オプションは必須ではありません。
{ | }	必須のコマンドオプションの選択肢を囲みます。	-d {y|n}	-d オプションには y 引数か n 引数のいずれかを使用する必要があります。
${ }	変数参照を示します。	${com.sun.javaRoot}	com.sun.javaRoot 変数の値を参照します。

表 P–4 記号の規則 (続き)

記号	説明	例	意味
-	同時に押すキーを示します。	Control-A	Control キーを押しながら A キーを押します。
+	順番に押すキーを示します。	Ctrl+A+N	Control キーを押してから放し、それに続くキーを押します。
→	グラフィカルユーザーインタフェースでのメニュー項目の選択順序を示します。	「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。

マニュアル、サポート、およびトレーニング

Sun のサービス	URL	内容
マニュアル	http://jp.sun.com/documentation/	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	http://jp.sun.com/supportraining/	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

◆ ◆ ◆ 第 1 章

1 概要

この章では Sun Java System Application Server の管理について説明します。Application Server の管理には、アプリケーションの配備、ドメイン、サーバーインスタンス、およびリソースの作成と設定、ドメインとサーバーインスタンスの制御 (起動と停止)、パフォーマンスの監視と管理、問題の診断とトラブルシューティングなどの多くの作業が含まれます。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

Sun Java System Application Server は、サーバーサイド Java アプリケーションおよび Web サービスの開発と配信用の Java 2 Platform, Enterprise Edition (J2EE プラットフォーム) 1.4 互換プラットフォームを提供します。主な機能には、スケーラブルなトランザクション管理、コンテナ管理による持続性ランタイム、パフォーマンス Web サービス、クラスタリング、高可用性、セキュリティー、統合機能などがあります。

Application Server は次の Edition が提供されています。

- Platform Edition は無償で配布され、ソフトウェア開発および部門レベルの本稼動環境を構築するために使用できます。
- Enterprise Edition はミッションクリティカルサービスと大規模な本稼働環境向けに設計されています。これは、ロードバランサプラグインとクラスタ管理によって、水平方向のスケーラビリティーとサービスの連続性をサポートします。さらに、Enterprise Edition では高可用性データベース (HADB) による信頼性の高いセッション状態管理もサポートしています。

ここでは、次の内容について説明します。

Application Server とは

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サポートするプラットフォームです。一方、開発者は 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 Platform Edition は、開発、本稼働配備、および再配布を自由に行えます。再配布の詳細については、`http://www.sun.com/software/products/appsrvr/appsrvr_oem.xml` を参照してください。

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。Application Server のインフラストラクチャーは、さまざまなタイプの分散アプリケーションの配備をサポートし、サービス指向アーキテクチャー (SOA) に基づいてアプリケーションを構築するために最適な基盤です。SOA はアプリケーションサービスを最大限に再利用することを目的とした設計方法論です。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。

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

  クラスタに Application Server インスタンスを追加することによってシステムの容量が増加し、水平的なスケーリングが実現されます。サービスを中断せずに、クラスタに Application Server インスタンスを追加することができます。HTTP、RMI/IIOP、および JMS ロードバランスシステムは、クラスタ内の正常な Application Server インスタンスに要求を分散させます。
- 高可用性 - 可用性 はフェイルオーバー機能を指します。1 台のサーバーインスタンスが停止すると、クラスタの別のサーバーインスタンスが、障害が発生したインスタンスのセッションを引き継ぎ、クライアントへのサービスをシームレスに続行します。セッションの情報は、高可用性データベース (HADB) に格納されます。HADB は、持続的な HTTP セッションとステートフルセッション Beans をサポートします。

Application Server のアーキテクチャー

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

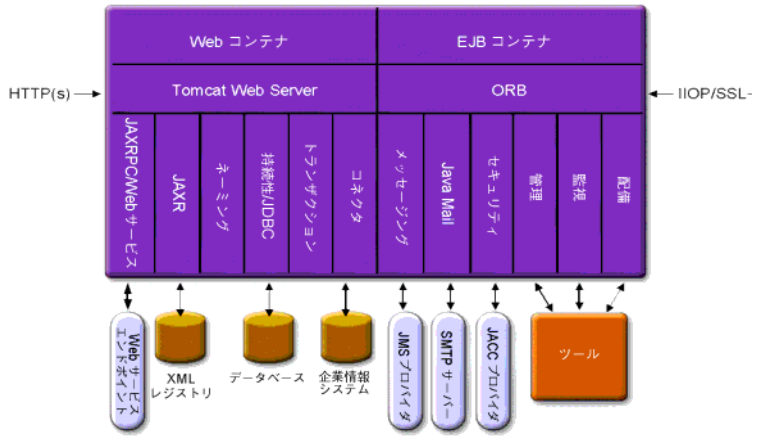

図 1–1　Application Server のアーキテクチャー

- コンテナ - コンテナは、J2EE コンポーネントのセキュリティーやトランザクション管理などのサービスを提供する実行環境です。図 1–1 は、2 つのタイプの J2EE コンテナ、Web および EJB を示しています。JSP ページやサーブレットなどの Web コンポーネントは、Web コンテナ内で実行されます。Enterprise Java Beans 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 Java Bean クライアントは、IIOP または IIOP/SSL (セキュア) プロトコルを介して ORB (Object Request Broker) と通信します。Application Server には、HTTP、HTTPS、IIOP、および IIOP/SSL プロトコル用の別個のリスナーがあります。各リスナーは、固有のポート番号を排他的に使用しています。
- **Web** サービス - J2EE プラットフォームでは、Java API for XML-Based RPC (JAX-RPC) によって実装された Web サービスを提供する Web アプリケーションを配備できます。J2EE アプリケーションやコンポーネントは、ほかの Web サービスのクライアントにすることもできます。アプリケーションは Java API for XML Registries (JAXR) を介して XML レジストリにアクセスします。
- アプリケーションのサービス - J2EE プラットフォームは、コンテナがアプリケーションのサービスを提供するように設計されています。図 1–1 は、次のサービスを示しています。
  - ネーミング - ネーミングおよびディレクトリサービスは、オブジェクトに名前をバインドします。J2EE アプリケーションは、JNDI 名を探してオブジェクトを検出します。JNDI は Java Naming and Directory Interface API の略です。

  - セキュリティー - Java Authorization Contract for Containers (JACC) は、J2EE コンテナ用に定義された一連のセキュリティー規約です。クライアントの ID に基づいて、コンテナはコンテナのリソースおよびサービスに対するアクセスを制限します。
- トランザクション管理 - トランザクションは作業の分割不能な単位です。たとえば、銀行口座間での資金の振り替えがトランザクションにあたります。トランザクション管理サービスは、トランザクションが完全に終了するか、またはロールバックされるようにします。

外部システムへのアクセス

J2EE プラットフォームでは、アプリケーションが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 バンドルには、Java DB データベース管理システムが含まれています。
- メッセージング - メッセージングは、ソフトウェアコンポーネント間またはアプリケーション間の通信メソッドです。メッセージングクライアントは、ほかのどのクライアントともメッセージの送受信を行います。アプリケーションは Java Messaging Service (JMS) API を介してメッセージングプロバイダにアクセスします。Application Server には JMS プロバイダが組み込まれています。
- コネクタ - J2EE コネクタアーキテクチャーでは、J2EE アプリケーションと既存のエンタープライズ情報システム (EIS) との統合が可能です。アプリケーションは、コネクタまたはリソースアダプタと呼ばれる移行可能な J2EE コンポーネントを介して EIS にアクセスします。
- JavaMail - JavaMail API を介して、アプリケーションは電子メールを送受信するために SMTP サーバーに接続します。
- サーバー管理 - 図 1-1 の右下に、Application Server の管理インタフェースを示しています。管理ツールはこれらのインタフェースを使用して、Application Server と通信します。

管理用ツール

Sun Java System Application Server の管理にはさまざまなツールと API を使用できます。

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。管理コンソール を使用するには、管理サーバー (ドメイン管理サーバーまたは DAS とも呼ばれる) が稼動している必要があります。管理コンソールを起動するには、管理サーバーのホスト名とポート番号が必要です。デフォルトの管理サーバーのデフォルトの管理サーバーポート番号は 4849 です。さらに、管理コンソールにログインするには、管理ユーザー名とパスワードが必要です。詳細については、該当する節を参照してください。

管理コンソールを起動するには、Web ブラウザで次のように入力します。

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

次に例を示します。

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

管理コンソール を管理サーバーが稼動しているマシンで実行する場合は、ホスト名として `localhost` を指定できます。

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

コマンド行インタフェース (asadmin ユーティリティー)

`asadmin` ユーティリティーは Sun Java System Application Server のコマンド行インタフェースです。管理コンソール で提供されている一連の同じ管理タスクを実行できます。`asadmin` ユーティリティーは、シェルでコマンドプロンプトから起動するか、ほかのスクリプトやプログラムから呼び出すことができます。`asadmin` ユーティリティーは *install-dir*`/bin` ディレクトリにインストールされます。Solaris でのデフォルトの Sun Java System Application Server インストールのルートディレクトリは `/opt/SUNWappserver` です。

`asadmin` ユーティリティーを起動するには、*install-dir*`/bin` ディレクトリに移動し、次のように入力します。

```
$ asadmin
```

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

```
asadmin> help
```

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

```
$ asadmin help
```

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

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

指定したコマンドの asadmin help 情報が、コマンドの UNIX マニュアルページに表示されます。これらのマニュアルページは、Web の 『Sun Java System Application Server Enterprise Edition 8.2 Reference Manual』 で HTML でも入手できます。

JConsole

Java 2 Platform Standard Edition 5.0 では、Java 監視および管理コンソール (JConsole) が導入されました。JConsole は Sun Java System Application Server の監視に使用します。JConsole の「リモート」タブまたは「詳細」タブを使用して、Application Server に接続できます。

- 「リモート」タブ: ユーザー名、パスワード、管理サーバーホスト、および JMS ポート番号 (デフォルトで 8686) を識別し、「接続」を選択します。
- 「詳細」タブ: JMXServiceURL をサービス (jmx:rmi:///jndi/rmi://host:jms-port/jmxrmi) として識別し、「接続」を選択します。JMXServerURL は server.log ファイルに書き込まれるほか、ドメイン作成コマンドのコマンドウィンドウに出力されます。

Application Server Management Extension (AMX)

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

Application Server Management Extension の使用の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide』 の第 16 章「Using the Java Management Extensions (JMX) API」 を参照してください。

Application Server のコマンドと概念

Sun Java System Application Server は 1 つまたは複数のドメインから構成されます。ドメインは管理上の境界であり、コンテキストです。各ドメインには管理サーバー (ドメイン管理サーバーまたは DAS とも呼ばれる) が関連付けられ、0 またはそれ以上のスタンドアロンインスタンスまたはクラスタ、あるいはその両方から構成されています。各クラスタには、1 つ以上の同機種サーバーインスタンスが含まれます。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。ドメイン内のサーバーインスタンス (スタンドアロンでもクラスタ構成でも) は異なる物理ホストで実行できます。

ここでは、次の内容について説明します。

ドメイン

ドメインは同時に管理されるインスタンスのグループです。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。管理境界に加えて、ドメインは基本的なセキュリティー構造を提供し、これによって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。

Sun Java System Application Server インストーラにより、デフォルトの管理ドメイン (`domain1` という名前) が作成されます。さらに、関連するドメイン管理サーバー (`server` という名前) も作成されます。管理サーバーポート番号を指定する必要があります。デフォルトの管理サーバーポートは 4849 です。インストーラは管理ユーザー名とパスワードも入力するよう求めます。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。

ドメイン管理サーバー (DAS)

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。管理コンソール は特定の DAS と通信し、関連するドメインを管理します。管理コンソール の各セッションにより、特定のドメインを設定し、管理できます。

ドメイン管理サーバー (DAS) は管理アプリケーションのホスト専用に設計されたアプリケーションサーバーインスタンスです。DAS は管理者を認証し、管理ツールからの要求を受け入れ、ドメイン内のサーバーインスタンスと通信して、要求を実行します。DAS は管理サーバーまたはデフォルトサーバーと呼ばれることもあります。デフォルトサーバーと呼ばれる理由は、Sun Java System Application Server のインストール時に作成される唯一のサーバーインスタンスで、配備に使用できるからです。DAS は単に追加の管理機能を備えたサーバーインスタンスです。

管理コンソール の各セッションでは、単一のドメインを設定し、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、ほかのドメインを管理する必要があります。管理コンソール の URL を指定する場合は、管理するドメインに関連付けられた DAS のポート番号を使用してください。

クラスタ

クラスタ は、一連の同じアプリケーション、リソース、および設定情報を共有するサーバーインスタンスの集まりに名前を付けたものです。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。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、サーバーインスタンスのロードバランスが容易になります。また、インスタンスレベルのフェイルオーバーによって、高可用性を実現します。管理上の観点では、クラスタは仮想エンティティーを表し、そのエンティティーでクラスタを構成するすべてのインスタンスに基づいたクラスタ上での操作 (アプリケーションの配備など) を実行します。

ノードエージェント

インスタンスのリモートライフサイクル管理を容易にするには、ドメインの各ノードに、軽量エージェント (JMX ランタイムのみをホストするなど) が必要です。この主な目的は、DAS の指示どおりに、サーバーインスタンスを起動、停止、作成することです。さらに、ノードエージェントはウォッチドッグとして機能し、障害の発生したプロセスを再起動します。DAS と同様に、ノードエージェントは特定の管理操作にのみ必要で、高可用性を期待するべきではありません。ただし、ノードエージェントは「常時稼働」コンポーネントであるため、ネィティブ O/S ノードブートストラップ (Solaris/Linux `inetd` または Windows サービスとしてなど) によって起動するように設定する必要があります。ノードエージェントは DAS には必要ありません。

サーバーインスタンス

サーバーインスタンスは、単一のノードで J2EE 1.4 Application Server をホストする単一の J2EE 互換 Java 仮想マシン (JVM) です。ドメインの各サーバーインスタンスは一意の名前を持ちます。クラスタ化されたサーバーインスタンスはクラスタのメン

バーであり、親クラスタからすべてのアプリケーション、リソース、および設定を受け取るため、クラスタのすべてのインスタンスは均一になります。クラスタ化されていないサーバーインスタンスはクラスタに属さないため、アプリケーション、リソース、および設定で独立したセットを使用します。

アプリケーションサーバーインスタンスは、アプリケーション配備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ます。DAS 以外のサーバーインスタンスには必ず、その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。

トポロジにリモートサーバーインスタンス (DAS 以外のサーバーインスタンス) が含まれる場合は、リモートサーバーインスタンスを管理し、補助するためのノードエージェントを作成します。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のコマンドを使用します。図 1–2 は、アプリケーションサーバーインスタンスの詳細を示しています。

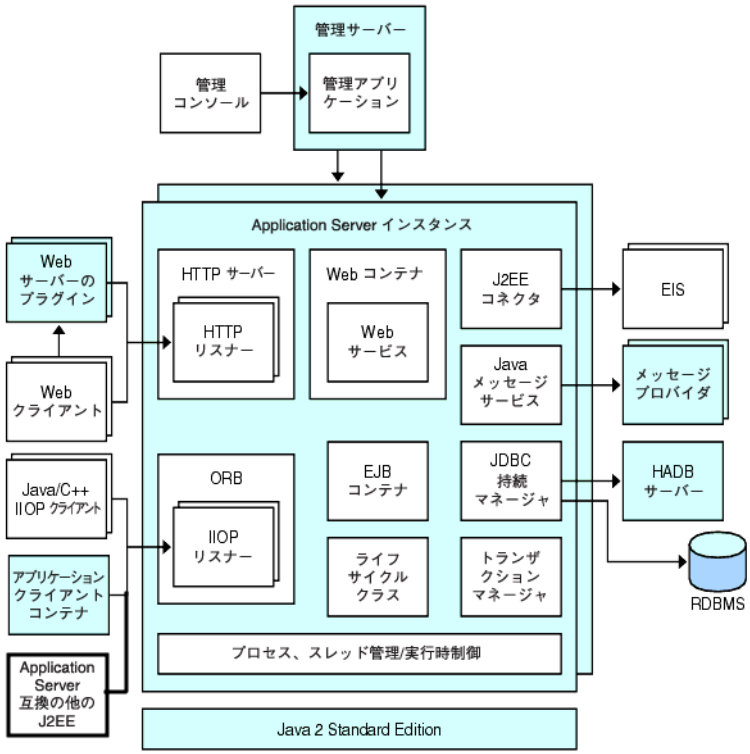

図 1–2　Application Server インスタンス

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。多くのユーザーにとっては、1 つの

アプリケーションサーバーインスタンスがあれば十分です。ただし、使用している環境によって追加のアプリケーションサーバーインスタンスを作成することが必要な場合があります。たとえば、開発環境内で異なるアプリケーションサーバーインスタンスを使用して、異なる Application Server 設定でテストしたり、異なるアプリケーション配備を比較およびテストできます。アプリケーションサーバーインスタンスは簡単に追加または削除できるため、これらを使用して、一時的にサンドボックス領域を作成して試用することができます。

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に対し、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、第 12 章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。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。

アプリケーションサーバーのコマンド

Application Server の管理には、ドメイン、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作成、設定、制御、管理などのタスクが含まれます。ここでは、次の内容について説明します。

ドメインの作成

ドメインは、`create-domain` コマンドを使用して作成します。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の入力を求めます。

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

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

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

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

domain-root-dir`/mydomain`

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

ドメインの削除

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

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

ドメインの一覧表示

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

```
$ asadmin list-domains
```

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

ドメインの起動

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

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

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

ドメインが 1 つだけの場合は、ドメイン名を省略できます。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 でのデフォルトのドメインの起動

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。

ドメインの停止

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

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

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

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

Windows でのデフォルトのドメインの停止

「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。

クラスタの作成

クラスタを作成するには create-cluster コマンドを使用します。次の例では、mycluster という名前のクラスタを作成します。管理サーバーホストは myhost、サーバーポートは 1234、管理ユーザー名は admin です。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin create-cluster --host myhost --port 1234 --user admin mycluster
```

コマンドの完全な構文を確認するには、asadmin help create-cluster と入力してください。

クラスタの起動

クラスタを起動するには start-cluster コマンドを使用します。次の例では mycluster という名前のクラスタを起動します。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin start-cluster --host myhost --port 1234 --user admin mycluster
```

myhost は管理サーバーホスト、1234 は管理ポート、admin は管理ユーザー名です。

コマンドの完全な構文を確認するには、asadmin help start-cluster と入力してください。クラスタを起動すると、クラスタのすべてのサーバーインスタンスが起動します。インスタンスを含まないクラスタは起動できません。

クラスタの停止

クラスタを停止するには stop-cluster コマンドを使用します。次の例では mycluster という名前のクラスタを停止します。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin stop-cluster --host myhost --port 1234 --user admin mycluster
```

myhost は管理サーバーホスト、1234 は管理ポート、admin は管理ユーザー名です。

コマンドの完全な構文を確認するには、asadmin help stop-cluster と入力してください。クラスタを停止すると、クラスタのすべてのサーバーインスタンスが停止します。インスタンスを含まないクラスタは停止できません。

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 create-node-agent コマンドを使用します。次の例では mynodeagent という名前のノードエージェントを作成します。管理サーバーホストは myhost、管理サーバーポートは 1234、管理ユーザー名は admin です。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin create-node-agent --host myhost --port 1234 --user admin mynodeagent
```

コマンドの完全な構文を確認するには、asadmin help create-node-agent と入力してください。

ノードエージェントの起動

ノードエージェントを起動するには start-node-agent コマンドを使用し、ノードエージェント名を指定します。たとえば、ノードエージェント mynodeagent を起動するには、次のように入力します。

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

コマンドの完全な構文を確認するには、asadmin help start-node-agent と入力してください。

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 stop-node-agent コマンドを使用し、ノードエージェント名を指定します。たとえば、ノードエージェント mynodeagent を停止するには、次のように入力します。

```
$ asadmin stop-node-agent mynodeagent
```

コマンドの完全な構文を確認するには、asadmin help stop-node-agent と入力してください。

インスタンスの作成

サーバーインスタンスを作成するには create-instance コマンドを使用します。次の例では、myinstance という名前のインスタンスを作成します。管理サーバーホストは myhost、管理サーバーポートは 1234、管理ユーザー名は admin です。このコマンドは、管理パスワードの入力を求めます。

次の例では myinstance という名前のクラスタ化されたサーバーインスタンスを作成します。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin create-instance --host myhost --port 1234
--user admin --cluster mycluster --nodeagent mynodeagent myinstance
```

myhost は管理サーバーホストで、管理ポートは 1234、管理ユーザー名は admin、このサーバーインスタンスが属するクラスタは mycluster、このサーバーインスタンスを管理するノードエージェントは mynodeagent です。

コマンドの完全な構文を確認するには、asadmin help create-instance と入力してください。

スタンドアロンサーバーインスタンスを作成する場合は、--cluster オプションを指定しません。

次の例では、mynodeagent という名前のノードエージェントで管理される myinstance という名前のスタンドアロンサーバーインスタンスを作成します。

```
$ asadmin create-instance --host myhost --port 1234
--user admin --nodeagent mynodeagent myinstance
```

インスタンスの起動

サーバーインスタンスを起動するには start-instance コマンドを使用します。次の例では、myinstance という名前のサーバーインスタンスを起動します。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin start-instance --host myhost --port 1234 --user admin  myinstance
```

管理サーバーホストは myhost、管理ポートは 1234、管理ユーザー名は admin です。サーバーインスタンス myinstance はクラスタ化することもスタンドアロンにすることもできます。

コマンドの完全な構文を確認するには、asadmin help start-instance と入力してください。

インスタンスの停止

サーバーインスタンスを停止するには stop-instance コマンドを使用します。次の例では myinstance という名前のインスタンスを停止します。このコマンドは、管理パスワードの入力を求めます。

```
$ asadmin stop-instance --host myhost --port 1234 --user admin  myinstance
```

管理サーバーホストは myhost、管理ポートは 1234、管理ユーザー名は admin です。サーバーインスタンス myinstance はクラスタ化することもスタンドアロンにすることもできます。

コマンドの完全な構文を確認するには、asadmin help stop-instance と入力してください。

インスタンスの再起動

サーバーインスタンスを再起動するには、インスタンスを停止してから、再起動します。

ドメイン管理サーバーの再作成

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

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

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

▼ DAS を移行する

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

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

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

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

      ./bundle-filename `-console`

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

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

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

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

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

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

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

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

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

次に例を示します。

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

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

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

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

たとえば、machine1 を検索して、machine3 に置き換えるとします。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 変更前:

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

変更後:

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

8 **machine3** の復元されたドメインを起動します。

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

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

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

11 **machine2** のノードエージェントを再起動します。

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

システム管理者のパスワードの変更

管理パスワードをリセットするには、ドメイン内のすべてのノードエージェントを停止する必要があります。これにより、関連付けられたすべてのサーバーインスタンスが停止します。すべてのサーバーインスタンスとノードエージェントが停止し、ドメイン管理サーバー (DAS) のみが稼動しています。

ここで、次のように管理ユーザーのパスワードを変更できます。

1. コマンド行インタフェースを使用して管理パスワードを変更します。

   ```
   asadmin update-file-user --authrealmname admin-realm ... --userpassword
   newpassword <admin-user-name>
   ```

2. 管理コンソールを使用して管理パスワードを変更します。

   管理サーバーの「設定」ノード >「セキュリティー」>「レルム」>「admin-realm」>「ユーザーを管理」とたどり、ユーザー ID を選択し、「ファイルレルムユーザーの編集」でパスワードを変更します。

   管理パスワードの変更に成功したことを示すメッセージが表示されます。

3. 新しいパスワードでドメイン管理サーバ (DAS) を再起動するには、次の手順に従います。

   コマンド行インタフェースの使用。asadmin start-domain --user admin --password newpassword domain1

   次のような設定が考えられます。2 つのノードエージェント (i1na、c1-na) と 3 つのインスタンス (c1 という名前の同じクラスタに属する c1i1 と c1i2、およびスタンドアロンサーバーインスタンス i1) があるドメイン。

4. 新しいパスワードでインスタンスを起動しないでノードエージェントを再起動します。

   次に例を示します。

   ```
   asadmin start-node-agent --user admin --password newpassword
   --startinstances=false i1-na asadmin
   ```

   ```
   asadmin start-node-agent --user admin --password newpassword
   --startinstances=false c1-na
   ```

5. サーバーとクラスタを再起動します。

   ```
   asadmin start-node-agent --user admin --password newpassword ... c1
   ```

   ```
   asadmin start-node-agent --user admin --password newpassword i1
   ```

Application Server の設定

Sun Java System Application Server の設定は domain.xml ファイルに保存されます。domain.xml は Application Server の設定の状態を表すドキュメントです。このドキュメントは特定の管理ドメインの中央リポジトリです。このドキュメントには、Application Server ドメインモデルの XML 表現が格納されます。domain.xml の内容は、ドメイン DTD の形式で表現された仕様によって管理されています。

ここでは、次の内容について説明します。

Application Server 設定の変更

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

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

手順については、37 ページの「ドメインの再起動」を参照してください。

動的設定が有効のときに次の設定変更を行う場合は、その設定変更を有効にするためにサーバーを再起動する必要はありません。

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

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

Application Server のポート

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

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

リスナー	デフォルトのポート番号	説明
管理サーバー	4849	ドメインの管理サーバーには、管理コンソールと asadmin ユーティリティーを使ってアクセスします。管理コンソールの場合は、ブラウザの URL にポート番号を指定します。リモートから asadmin コマンドを実行する場合は、--port オプションを使用してポート番号を指定します。
HTTP	8080	Web サーバーはポート上で HTTP 要求を待機します。配備された Web アプリケーションとサービスにアクセスするために、クライアントはこのポートに接続します。
HTTPS	8181	セキュア通信用に設定された Web アプリケーションは、個別のポートで待機します。
IIOP (Internet Inter-ORB Protocol)	3700	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	3820/3890	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。
JMX	8686	JMX コネクタは DAS との通信にほかのポートを使用します。

J2SE ソフトウェアの変更

Application Server は Java 2 Standard Edition (J2SE) ソフトウェアに依存します。Application Server をインストールすると、J2SE ソフトウェアのディレクトリが指定されます。J2SE ソフトウェアの変更の手順については、第 17 章を参照してください。

◆ ◆ ◆ 第 2 章

2 アプリケーションの配備

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

配備のライフサイクル

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

1. 初期の配備

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

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

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

管理コンソール ではなく asadmin deploy コマンドを使用して、ドメインに配備することもできます。アプリケーションやモジュールをドメインだけに配備する場合、アプリケーションやモジュールはドメインの中央リポジトリ内に存在しますが、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。

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。

自動配備

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

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

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

注 – 自動配備は、デフォルトのサーバーインスタンスにのみ利用可能です。

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

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、208 ページの「詳細設定」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
```

deploytool ユーティリティーの使用

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

J2EE アーカイブファイルのタイプ

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

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

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

命名規約

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

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

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

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

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

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

◆ ◆ ◆ 第 3 章

JDBC リソース

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。

この章の内容は次のとおりです。

JDBC リソースの作成

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。Java EE 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。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、リソースを識別する一意の JNDI 名を指定します。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。

管理コンソールを使用して JDBC リソースを作成するには、「リソース」>「JDBC リソース」の順に選択します。次のようにリソース設定を指定します。

- JNDI 名: 一意の名前を指定します。図書館内でカード目録によって図書を編成し、位置を表すのと同様に、JNDI 名は、分散型コンピューティング環境内でコンポーネントを編成し、位置を示します。そのため、JNDI 名は JDBC リソースにアクセスする重要な方法になります。慣例により、名前は `jdbc/` 文字列で始まります。次に例を示します。`jdbc/payrolldb`. スラッシュを忘れないでください。
- プール名: 新しい JDBC リソースに関連した接続プールを選択します。
- 説明: リソースについての短い説明を入力します。
- 各クラスタノードの Status (列挙型)リソースを利用不可にする場合は、「有効」チェックボックスの選択を解除します。デフォルトでは、リソースは作成すると同時に利用可能 (有効) です。

コマンド行ユーティリティーを使用して JDBC を作成するには、`create-jdbc-resource` コマンドを使用します。

JDBC 接続プールの作成

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。

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

JDBC ドライバおよびデータベースベンダーに固有の特定のデータを入力する必要があります。処理を開始する前に、次の情報を集めます。

- データベースベンダー名
- `javax.sql.DataSource` (ローカルトランザクションのみ) や `javax.sql.XADataSource` (グローバルトランザクション) などのリソースタイプ
- データソースクラス名: JDBC ドライバにリソースタイプとデータベースのデータソースクラスがある場合、「データソースクラス名」フィールドの値が必要です。

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

インストールした JDBC ドライバで指定されているとおりに、一般設定の値を定義します。これらの設定は、Java プログラミング言語で記述されたクラスやインタフェースの名前です。

パラメータ	説明
データソースクラス名	`DataSource` API、`XADataSource` API、あるいはその両方を実装するベンダー固有のクラス名。このクラスは JDBC ドライバにあります。
リソースタイプ	選択肢には `javax.sql.DataSource` (ローカルトランザクションに限る)、`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。

さらに、プール内に存在する一連の物理データベース接続を定義する必要があります。アプリケーションが接続を要求すると、接続はプールから削除され、アプリケーションが接続を解放すると、接続はプールに返されます。

パラメータ	説明
初期および最小プールサイズ	プール内の接続の最小数。この値は、プールを最初に作成したり、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。

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

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

パラメータ	説明
接続検証	必要なチェックボックスを選択して、接続検証を有効にします。
検証方法	アプリケーションサーバーは、auto-commit、metadata、および table の 3 つの方法でデータベース接続を検証できます。 auto-commit と metadata - アプリケーションサーバーは、`con.getAutoCommit()` 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。

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

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

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。Java EE アプリケーションは、JDBC API を介してリレーショナルデータベースにアクセスします。アプリケーションがデータベースにアクセスするには、接続を取得する必要があります。

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

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

   リソースの JNDI 名を取得すると、ネーミングおよびディレクトリサービスが JDBC リソースを検索します。JDBC リソースはそれぞれ接続プールを指定します。

2. JDBC リソースを経由して、アプリケーションはデータベース接続を取得します。

   バックグラウンドで、アプリケーションサーバーはデータベースに対応した接続プールから物理接続を取得します。プールは、データベース名 (URL)、ユーザー名、およびパスワードなどの接続属性を定義します。

3. データベースに接続すると、アプリケーションはデータベースのデータの読み込み、変更、および追加を行うことができます。

   アプリケーションは JDBC API を呼び出すことにより、データベースにアクセスします。JDBC ドライバはアプリケーションの JDBC 呼び出しをデータベースサーバーのプロトコルに変換します。

4. データベースへのアクセスが完了すると、アプリケーションは接続を終了します。

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

データベースアクセスの設定

データベースアクセスを設定するには、まずサポートされているデータベース製品をインストールする必要があります。Application Server がサポートするデータベース製品のリストについては、リリースノートを参照してください。次に、データベース製品の JDBC ドライバをインストールし、ドライバの JAR ファイルがドメインのサーバーインスタンスにアクセスできるようにする必要があります。次に、データベース、データベースの接続プール、および接続プールをポイントする JDBC リソースを作成します。

JDBC ドライバをインストールしたら、ドライバが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するように統合する必要があります。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
  1. ドライバの JAR および ZIP ファイルを *domain-dir* `/lib` ディレクトリにコピーするか、またはドライバのクラスファイルを *domain-dir*`/lib/ext` ディレクトリにコピーします。
  2. 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
  1. 管理コンソール で、目的の設定の JVM 設定を選択します。
  2. ドライバの JAR ファイルの完全修飾パス名を識別します。
  3. 設定を保存して、サーバーを再起動します。

持続マネージャーリソース

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

管理コンソールを使用して、持続マネージャーリソースを作成するには、「リソース」ノード>「持続マネージャー」ノード>「持続マネージャー」ページの順に選択します。コマンド行ユーティリティーを使用するには、`create-persistence-resource` コマンドを使用します。

◆ ◆ ◆ 第 4 章

Java Message Service (JMS) リソースの設定

この章では、Java Message Service (JMS) API を使用するアプリケーションのリソースを設定する方法について説明します。次の項があります。

JMS リソースについて

Application Server の JMS プロバイダ

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

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

JMS リソース

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

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

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

- 接続ファクトリリソースを作成することによって、接続ファクトリを作成します
- 次の 2 つのオブジェクトを作成して、送信先を作成します
  - 物理的送信先
  - 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 に関する章では、この 2 つの通信タイプについての詳細および JMS のほかの側面が説明されています
(`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html` を参照)。

リソースを作成する順序は重要ではありません。

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

- 接続ファクトリ JNDI 名は `resource-ref` または `mdb-connection-factory` 要素に指定します。

- 送信先リソース JNDI 名は、メッセージ駆動型 Bean の `ejb` 要素と `message-destination` 要素に指定します。
- 物理送信先名は、Enterprise JavaBean 配備記述子の `message-driven` 要素または `message-destination-ref` 要素のいずれかにある `message-destination-link` 要素に指定します。さらに、`message-destination` 要素にも指定します。`message-destination-ref` 要素は、新しいアプリケーションで推奨されない `resource-env-ref` 要素から置き換わります。Application Server 配備記述子の `message-destination` 要素で、物理送信先名と送信先リソース名をリンクします。

JMS リソースとコネクタリソースの関係

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

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

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

JMS 接続ファクトリ

JMS 接続ファクトリは、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。これらの管理対象オブジェクトは、`ConnectionFactory`、`QueueConnectionFactory`、および `TopicConnectionFactory` インタフェースを実装します。 Application Server 管理コンソールを使用して、JMS 接続ファクトリを作成、編集、または削除できます。新しい JMS 接続ファクトリの作成では、ファクトリのコネクタ接続プールとコネクタリソースも作成します。

コマンド行ユーティリティーを使用して、JMS 接続ファクトリを管理するには、`create-jms-resource`、`list-jms-resources`、または `delete-jms-resource` コマンドを使用します。

JMS 送信先リソース

JMS 送信先は、メッセージのリポジトリとして機能します。管理コンソールを使用して、JMS 送信先リソースを作成、変更、または削除できます。新しい JMS 送信先リソースを作成するには、「リソース」>「JMS リソース」>「送信先リソース」の順に選択します。「送信先リソース」ページで、次を指定できます。

- リソースの JNDI 名。JMS リソースのネーミングサブコンテキストのプレフィックス `jms/` を使用することをお勧めします。次に例を示します。`jms/Queue`
- リソースタイプ。`javax.jms.Topic` または `javax.jms.Queue` です。
- 送信先リソースの追加プロパティー。これらのすべての設定と追加のプロパティーの詳細については、管理コンソールのオンラインヘルプを参照してください。

コマンド行ユーティリティーを使用して、JMS 送信先を管理するには、`create-jms-resource` または `delete-jms-resource` コマンドを使用します。

ヒント – `asadmin create-jms-resource` コマンドの `addresslist` プロパティー (`host:mqport,host2:mqport,host3:mqport` の形式で) を指定するには、`\\` を使用して、`:` をエスケープします。たとえば、`host1\\:mqport,host2\\:mqport,host3\\:mpqport` のようになります。

エスケープ文字の使用の詳細については、`asadmin`(8) のマニュアルページを参照してください。

JMS 物理送信先

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

管理コンソールから物理送信先を作成するには、「設定」>「物理送信先」の順に選択します。「現在の物理送信先」ページで、物理送信先の名前を指定し、送信先のタイプを選択します。これは `topic` または `queue` です。「物理送信先」ページのフィールドとプロパティ?の詳細については、管理コンソールのオンラインヘルプを参照してください。

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

コマンド行ユーティリティーを使用して、JMS 物理送信先を管理するには、`create-jmsdest`、`flush-jmsdest`、または `delete-jmsdest` コマンドを使用します。

JMS プロバイダ

JMS プロバイダの一般プロパティーの設定

管理コンソールの「JMS サービス」ページを使用して、すべての JMS 接続で使用するプロパティーを設定します。管理コンソールで、「設定」>「Java メッセージサービス」の順に選択します。「JMS サービス」ページで、次の JMS サービスの一般設定を制御できます。

- 「起動タイムアウト」の間隔を選択します。これは、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 待機する時間を示します。
- 「JMS サービス」のタイプを選択します。これは、JMS サービスをローカルホストで管理するか、リモートホストで管理するかを指定します。
- 「起動引数」を指定して、JMS サービスの起動をカスタマイズします。
- 「再接続」チェックボックスを選択して、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。
- 「再接続間隔」を秒数で指定します。この間隔は、`AddressList` で指定した各アドレスおよびリストの次のアドレスへの試行に適用されます。間隔が短すぎると、ブローカ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。
- 再接続の試行回数を指定します。このフィールドに、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。
- デフォルトの JMS ホストを選択します。
- 「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。
- 「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。
- デフォルト以外のスキームまたはサービスを使用する場合は、「MQ スキーム」および「MQ サービス」フィールドに、Message Queue アドレススキーム名と Message Queue 接続サービス名を入力します。

これらのすべてのプロパティーの値は実行時にも更新できます。ただし、更新された値を取得するのは、プロパティーの更新後に作成された接続ファクトリのみで

す。既存の接続ファクトリは元のプロパティー値のままになります。さらに、ほとんどすべての値を有効にするには、アプリケーションサーバーを再起動する必要があります。アプリケーションサーバーを再起動しなくても更新可能なプロパティーは、デフォルトの JMS ホストのみです。

コマンド行ユーティリティーを使用して JMS プロバイダを管理するには、`set` または `jms-ping` コマンドを使用します。

リモートサーバーへのアクセス

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

外部 JMS プロバイダ

JMS の汎用リソースアダプタは Java EE Connector 1.5 リソースアダプタで、IBM Websphere MQ、Tibco EMS、および Sonic MQ などの外部 JMS プロバイダの JMS クライアントライブラリをラップできるため、任意の JMS プロバイダを Sun Java System Application Server などの Java EE 1.4 アプリケーションサーバーに統合します。アダプタは Java EE 1.4 アプリケーションサーバーの管理ツールを使用して配備および設定可能な `.rar` アーカイブです。

JMS の汎用リソースアダプタの設定

アプリケーションサーバーの管理ツールを使用して、JMS の汎用リソースアダプタを配備および設定できます。ここでは、Sun Java System Application Server を使用して、JMS の汎用リソースアダプタを設定する方法を説明します。概して、リソースアダプタを設定して、JMS プロバイダが XA をサポートするかどうかを示すことができます。さらに、JMS プロバイダで可能な統合のモードを示すこともできます。リソースアダプタでは、2 つの統合のモードをサポートしています。最初のモードは、統合の手段として JNDI を使用します。この場合、管理対象オブジェクトを JMS プロバイダの JNDI ツリーに設定し、汎用リソースアダプタがそれらを検索し、使用します。このモードが統合に適切でない場合は、JMS 管理対象オブジェクト JavaBean クラスの Java リフレクションを統合のモードとして使用することもできます。Sun Java System Application Server の管理コンソールまたは CLI を使用して、リソースアダプタを設定できます。これは、他のリソースアダプタの設定と変わりありません。

汎用リソースアダプタの設定

リソースアダプタを配備する前に、アプリケーションサーバーで JMS クライアントライブラリを使用できるようにします。一部の JMS プロバイダでは、クライアントライブラリにネイティブライブラリも含まれている場合があります。そのような場合は、これらのネイティブライブラリもアプリケーションサーバー JVM から使用できるようにします。

1. コネクタモジュールを配備する場合と同じように、汎用リソースアダプタを配備します。

   この手順については、管理コンソールのオンラインヘルプを参照してください。配備時に、汎用リソースアダプタの場所として `install-dir/lib/addons/resourceadapters/genericjmsra/genericra.rar` を指定してください。さらに、64 ページの「リソースアダプタのプロパティー」の節で説明するプロパティーを指定する必要があります。

2. コネクタ接続プールを作成します。

   この手順については、管理コンソールのオンラインヘルプを参照してください。「コネクタ接続プールを作成」ページの「リソースアダプタ」コンボボックスから、`genericra` を選択します。さらに、「接続定義」コンボボックスで、`javax.jms.QueueConnectionFactory` を選択します。また、68 ページの「`ManagedConnectionFactory` のプロパティー」の節で説明するプロパティーを指定します。

3. コネクタリソースを作成します。

   この詳細な手順については、管理コンソールのオンラインヘルプを参照してください。「コネクタリソースを作成」ページで、前の手順で作成したプールを選択します。

4. 管理対象オブジェクトリソースを作成します。

   この詳細な手順については、管理コンソールのオンラインヘルプを参照してください。「管理オブジェクトリソースを作成」ページで、「リソースアダプタ」として `genericra` を、「リソースタイプ」として `javax.jms. Queue` を選択します。「次へ」をクリックし、2 番目のページで、「プロパティーを追加」をクリックします。「追加プロパティー」テーブルで、値 `Name\\=clientQueue` を持つ `DestinationProperties` という新しいプロパティーを指定します。その他のプロパティーについては、68 ページの「管理対象オブジェクトリソースのプロパティー」を参照してください。

5. Sun Java System Application Server セキュリティーポリシーを次のように変更します。

   - `sjsas_home/domains/domain1/config/server.policy` を変更して、`java.util.logging.LoggingPermission "control"` を追加します。

- sjsas_home/lib/appclient/client.policy を変更して、permission javax.security.auth.PrivateCredentialPermission "javax.resource.spi.security.PasswordCredential * \"*\"","read"; を追加します。

リソースアダプタのプロパティー

次の表に、リソースアダプタの作成時に使用するプロパティーを示します。

プロパティー名	有効な値	デフォルト値	説明
ProviderIntegrationMode	javabean/jndi	javabean	リソースアダプタと JMS クライアントの統合のモードを指定します。
ConnectionFactory ClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging.ConnectionFactory	なし	JMS クライアントの javax.jms.ConnectionFactory 実装のクラス名。ProviderIntegrationMode が javabean の場合に使用します。
QueueConnectionFactory ClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging.QueueConnectionFactory	なし	JMS クライアントの javax.jms.QueueConnectionFactory 実装のクラス名。これは ProviderIntegrationMode が javabean の場合に使用します。
TopicConnectionFactory ClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging.TopicConnectionFactory	なし	JMS クライアントの javax.jms.TopicConnectionFactory 実装のクラス名。これは ProviderIntegrationMode が javabean として指定されている場合に使用します。
XAConnectionFactory ClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging.XAConnectionFactory	なし	JMS クライアントの >javax.jms.ConnectionFactory 実装のクラス名。これは ProviderIntegrationMode が javabean として指定されている場合に使用します。

プロパティー名	有効な値	デフォルト値	説明
XAQueueConnectionFactory ClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging. XAQueueConnectionFactory	なし	JMS クライアントの javax.jms.XAQueueConnectionFactory 実装のクラス名。これは ProviderIntegrationMode が javabean として指定されている場合に使用します。
XATopicConnectionFactory ClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging. XATopicConnectionFactory	なし	JMS クライアントの javax.jms.XATopicConnectionFactory 実装のクラス名。これは ProviderIntegrationMode が javabean の場合に使用します。
TopicClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging.Topic	なし	JMS クライアントの javax.jms.Topic 実装のクラス名。これは ProviderIntegrationMode が javabean の場合に使用します。
QueueClassName	アプリケーションサーバークラスパスで使用可能なクラス名。たとえば次のようになります。 com.sun.messaging.Queue	なし	JMS クライアントの javax.jms.Queue 実装のクラス名これは ProviderIntegrationMode が javabean として指定されている場合に使用します。
SupportsXA	True/false	FALSE	JMS クライアントが XA をサポートするかどうかを指定します。
ConnectionFactoryProperties	コンマで区切られた名前と値のペア。	なし	これは javabean プロパティー名と JMS クライアントの ConnectionFactory の値を指定します。これは ProviderIntegrationMode が javabean の場合にのみ必要です。
JndiProperties	コンマで区切られた名前と値のペア。	なし	これは JMS プロバイダの JNDI への接続に使用する JNDI プロバイダのプロパティーを指定します。これは ProviderIntegrationMode が jndi の場合にのみ使用します。

プロパティー名	有効な値	デフォルト値	説明
CommonSetterMethodName	メソッド名	なし	これは、一部の JMS ベンダーが管理対象オブジェクトにプロパティーを設定するために使用する一般的な setter メソッド名を指定します。このプロパティーは ProviderIntegrationMode が javabean の場合にのみ使用します。Sun Java System Message Queue の場合、このプロパティーの名前は setProperty になります。
UserName	JMS ユーザーの名前	なし	JMS プロバイダに接続するためのユーザー名。
Password	JMS ユーザーのパスワード。	なし	JMS プロバイダに接続するためのパスワード。

プロパティー名	有効な値	デフォルト値	説明
RMPolicy	ProviderManaged または OnePerPhysicalConnection	ProviderManaged	XAResource の isSameRM メソッドは、トランザクションマネージャーで、2 つの XAResources によって表されたリソースマネージャーインスタンスが同じであるかどうかを判断するために使用されます。 RMPolicy を ProviderManaged (デフォルト値) に設定すると、JMS プロバイダが、汎用リソースアダプタの RMPolicy ラッパーおよび XAResource ラッパーが isSameRM 呼び出しをメッセージキュープロバイダの XA リソースの実装に単に委任するかどうかを JMS プロバイダが決定します。これは、大半のメッセージキュー製品で最適に機能するべきです。 IBM MQ シリーズなどの XAResource の一部の実装では、物理接続ごとに 1 つのリソースマネージャーを使用します。これにより、単一のトランザクションで同じキューマネージャーに対するインバウンド通信とアウトバウンド通信がある (たとえば MDB が送信先に応答を送信するなど) 場合に問題が発生します。 RMPolicy が OnePerPhysicalConnection に設定されている場合、汎用リソースアダプタの XAResource ラッパーの実装の isSameRM は、ラッパー対象オブジェクトに委任する前に、両方の XAResources が同じ物理接続を使用するかどうかを確認します。このプロパティーの詳細については、 Glassfish Web サイトの「Issue Tracker database」の問題番号 5 を参照してください。

ManagedConnectionFactory のプロパティー

ManagedConnectionFactory プロパティーは connector-connection-pool の作成時に指定します。リソースアダプタの作成時に指定されたすべてのプロパティーは、ManagedConnectionFactory でオーバーライドできます。 ManagedConnectionFactory でのみ使用可能な追加のプロパティーを次に示します。

プロパティー名	有効な値	デフォルト値	説明
ClientId	有効なクライアント ID	なし	JMS 1.1 仕様に指定されている ClientID
ConnectionFactoryJndiName	JNDI 名	なし	JMS プロバイダの JNDI ツリーにバインドされた接続ファクトリの JNDI 名。管理者は、JMS プロバイダ自体にすべての接続ファクトリプロパティー (clientID を除く) を指定するようにしてください。このプロパティー名は ProviderIntegratinMode が jndi の場合にのみ使用されます。
ConnectionValidationEnabled	true/false	FALSE	true に設定した場合、リソースアダプタは例外リスナーを使用して、接続の例外をキャッチし、CONNECTION_ERROR_OCCURED イベントをアプリケーションサーバーに送信します。

管理対象オブジェクトリソースのプロパティー

このプロパティーは、管理対象オブジェクトリソースの作成時に指定します。リソースアダプタのすべてのプロパティーは、管理対象リソースオブジェクトでオーバーライドできます。管理対象オブジェクトでのみ使用可能な追加のプロパティーを次に示します。

プロパティー名	有効な値	デフォルト値	説明
DestinationJndiName	JNDI 名	なし	JMS プロバイダの JNDI ツリーにバインドされた送信先の JNDI 名。管理者は JMS プロバイダ自体にすべてのプロパティーを指定するようにしてください。このプロパティー名は ProviderIntegrationMode が jndi の場合にのみ使用されます。
DestinationProperties	コンマで区切られた名前と値のペア	なし	これは JMS クライアントの送信先の javabean プロパティー名と値を指定します。ProviderIntegrationMode が javabean である場合にのみ必要です。

有効化仕様プロパティー

このプロパティーは、activation-config-properties として MDB の Sun 固有の配備記述子に指定されています。すべてのリソースアダプタのプロパティーは有効化仕様でオーバーライドできます。有効化仕様でのみ使用可能な追加のプロパティーを次に示します。

プロパティー名	有効な値	デフォルト値	説明
MaxPoolSize	整数	8	並行メッセージ配信用のリソースアダプタによって、内部で作成されるサーバーセッションプールの最大サイズ。これは MDB オブジェクトの最大プールサイズに等しくなるべきです。
MaxWaitTime	整数	3	リソースアダプタは、その内部プールからサーバーセッションを取得するために、このプロパティーに指定された秒単位の時間を待機します。この制限を超えると、メッセージ配信が失敗します。

プロパティー名	有効な値	デフォルト値	説明
SubscriptionDurability	持続性または非持続性	非持続性	JMS 1.1 仕様に指定されている SubscriptionDurability
SubscriptionName		なし	JMS 1.1 仕様に指定されている SubscriptionName
MessageSelector	有効なメッセージセレクタ	なし	JMS 1.1 仕様に指定されている MessageSelector
ClientID	有効なクライアント ID	なし	JMS 1.1 仕様に指定されている ClientID
ConnectionFactoryJndiName	有効な JNDI 名	なし	JMS プロバイダで作成された接続ファクトリの JNDI 名。この接続ファクトリはリソースアダプタが接続を作成し、メッセージを受け取るために使用します。ProviderIntegrationMode が jndi と設定されている場合にのみ使用します。
DestinationJndiName	有効な JNDI 名	なし	JMS プロバイダで作成された送信先の JNDI 名。この送信先は、リソースアダプタが接続を作成し、メッセージを受け取るために使用します。ProviderIntegrationMode が jndi と設定されている場合にのみ使用します。
DestinationType	javax.jms.Queue または javax.jms.Topic	null	MDB が待機する送信先のタイプ。
DestinationProperties	コンマで区切られた名前と値のペア	なし	これは JMS クライアントの送信先の javabean プロパティー名と値を指定します。ProviderIntegrationMode が javabean である場合にのみ必要です。
RedeliveryAttempts	integer		MDB でメッセージによって実行時例外が発生した場合に、メッセージが配信される回数。
RedeliveryInterval	秒単位での時間		MDB でメッセージによって実行時例外が発生した場合に、配信を繰り返す間隔。

プロパティー名	有効な値	デフォルト値	説明
SendBadMessagesToDMD	true/false	false	配信の試行回数を超えた場合に、リソースアダプタがデッドメッセージ送信先にメッセージを送信すべきかどうかを示します。
DeadMessageDestination JndiName	有効な JNDI 名。	なし	JMS プロバイダによって作成された送信先の JNDI 名。これは、デッドメッセージのターゲット送信先です。これは ProviderIntegrationMode が jndi の場合にのみ使用します。
DeadMessageDestination ClassName	送信先オブジェクトのクラス名。	なし	ProviderIntegrationMode が javabean の場合に使用します。
DeadMessageDestination Properties	コンマで区切られた名前と値のペア	なし	これは JMS クライアントの送信先の javabean プロパティー名と値を指定します。これは ProviderIntegrationMode が javabean の場合のみ必要です。
ReconnectAttempts	integer		例外リスナーが接続時のエラーをキャッチした場合に試行される再接続の回数。
ReconnectInterval	秒単位での時間		再接続の間隔。

◆ ◆ ◆ 第 5 章

JavaMail リソースの設定

Application Server には JavaMail API が含まれています。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 には、電子メッセージの読み取り、作成、送信のための機能が備えられています。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 を使用して、アプリケーションに電子メール機能を追加できます。JavaMail は Java アプリケーションから、ネットワークまたはインターネット上の IMAP (Internet Message Access Protocol) および SMTP (メール転送プロトコル) 対応のメールサーバーにアクセスできます。メールサーバー機能はないため、JavaMail を使用するにはメールサーバーにアクセスする必要があります。

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

この章は、次の節で構成されています。

JavaMail セッションの作成

Application Server で JavaMail を設定して使用するには、Application Server 管理コンソールでメールセッションを作成します。これにより、サーバー側コンポーネントとアプリケーションは、JavaMail サービスに割り当てられたセッションプロパティを使用して、JNDI を使用して JavaMail サービスにアクセスできます。メールセッションを作成する際に、管理コンソールで、メールホスト、トランスポートプロトコルとストアプロトコル、およびデフォルトのメールユーザーを指定できるため、JavaMail を使用するコンポーネントはこれらのプロパティーを設定する必要がありません。Application Server は単一のセッションオブジェクトを作成して、JNDI を介してセッションオブジェクトを必要とするすべてのコンポーネントに使用できるようにするため、電子メールを大量に使用するアプリケーションで役立ちます。

管理コンソールを使用して、JavaMail セッションを作成するには、「リソース」>「JavaMail セッション」の順に選択します。次のように JavaMail 設定を指定します。

- JNDI 名: メールセッションの一意の名前。JavaMail リソースには、ネーミングサブコンテキストプレフィックス `mail/` を使用します。次に例を示します。
  `mail/MySession`
- メールホスト: デフォルトのメールサーバーのホスト名。プロトコル固有のホストプロパティーが提供されていない場合、Store オブジェクトと Transport オブジェクトの接続メソッドはこの値を使用します。この名前は実際のホスト名として解決可能でなければいけません。
- デフォルトユーザー: メールサーバーに接続する際に指定するユーザー名。プロトコル固有の username プロパティーが提供されていない場合、Store オブジェクトと Transport オブジェクトの接続メソッドはこの値を使用します。
- デフォルトの返信用アドレス: 次の形式のデフォルトユーザーの電子メールアドレス。*username*@*host.domain* パターンを使用して名前を付けることができます。
- 説明: コンポーネントの説明文を記述します。
- セッション：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。

さらに、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、次の詳細設定を定義します。

- ストアプロトコル: 使用するストアオブジェクトの通信方法を定義します。デフォルトのストアプロトコルは `imap` です。
- ストアプロトコルクラス: 目的のストアプロトコルを実装するストア通信方法クラスを指定します。デフォルトのストアプロトコルクラスは `com.sun.mail.imap.IMAPStore` です。
- トランスポートプロトコル: トランスポート通信方法を識別します。デフォルトのトランスポートプロトコルは `smtp` です。
- トランスポートプロトコルクラス: トランスポートクラスの通信方法を定義します。デフォルトのトランスポートプロトコルクラスは `com.sun.mail.smtp.SMTPTransport` です。
- デバッグ: 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、このチェックボックスにチェックマークを付けます。JavaMail のログレベルを `FINE` またはそれ以上に設定した場合、デバッグ出力が生成され、システムのログファイルに取り込まれます。ログレベルの設定の詳細については、156 ページの「カスタムログレベルを設定する」を参照してください。
- 追加プロパティー: プロトコル固有のホストやユーザー名のプロパティーなどアプリケーションに必要なプロパティーを作成します。JavaMail API マニュアルには、使用可能なプロパティーのリストがあります。

◆ ◆ ◆ 第 6 章

6 JNDI リソース

Java Naming and Directory Interface (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 Application Programming Interface (API) です。Java EE 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。

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

この章の内容は次のとおりです。

J2EE ネームサービス

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

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

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

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

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

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

ネーミング参照とバインディング情報

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

リソースの JNDI 名とリソース参照名とは同じではありません。このネーミングへのアプローチでは、配備前に 2 つの名前をマップする必要がありますが、同時にコンポーネントをリソースから分離します。この分離により、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。

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

表 6–1　JNDI 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ー接続ファクトリ
java:comp/env/ejb	EJB 参照
java:comp/UserTransaction	UserTransaction 参照
java:comp/env/mail	JavaMail セッション接続ファクトリ
java:comp/env/url	URL 接続ファクトリ
java:comp/env/jms	JMS 接続ファクトリと送信先
java:comp/ORB	アプリケーションコンポーネント間で共有された ORB インスタンス

カスタムリソースの使用

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

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

外部 JNDI リポジトリおよびリソースの使用

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

外部 JNDI リソースの使用例を示します。

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

◆ ◆ ◆ 第 7 章

7 コネクタリソース

コネクタモジュールとは、アプリケーションがエンタープライズ情報システム (EIS) と対話することを可能にする Java 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 Java 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。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 9 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。

この章では、次の内容について説明します。

コネクタ接続プール

次の表では、接続プールの設定について説明しています。

パラメータ	説明
初期および最小プールサイズ	プール内の接続の最小数。この値は、プールを最初に作成したり、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。

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」メニューから「なし」を選択すると、リソースアダプタがローカルのリソースマネージャーまたはJTAトランザクションをサポートせず、`XAResource`または`LocalTransaction`インタフェースを実装しないことを示します。JAXRリソースアダプタの場合は、「トランザクションサポート」メニューから「なし」を選択する必要があります。JAXRリソースアダプタは、ローカルトランザクションまたはJTAトランザクションをサポートしていません。 ローカルトランザクションサポートは、リソースアダプタが`LocalTransaction`インタフェースを実装することにより、ローカルトランザクションをサポートすることを意味します。ローカルトランザクションはリソースマネージャーの内部で管理され、外部トランザクションマネージャーは一切関与しません。 XAトランザクションサポートは、リソースアダプタが`LocalTransaction`および`XAResource`インタフェースを実装することにより、ローカルのリソースマネージャーとJTAトランザクションをサポートすることを意味します。XAトランザクションは、リソースマネージャーの外部にあるトランザクションマネージャーによって制御され、協調動作します。ローカルトランザクションはリソースマネージャーの内部で管理され、外部トランザクションマネージャーは一切関与しません。
コネクタの検証	接続プールをアプリケーションに渡す前に検証する場合は、「有効」チェックボックスを選択します。

接続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)を配備する必要があります。コネクタモジュールは、管理コンソールを使用するか、`asadmin`コマンドを使用することで配備できます。`asadmin`コマンドの詳細については、`asadmin`(1M)を参照してください。

管理コンソールで接続プールを表示、作成、編集、削除するには、「リソース」>「コネクタ」>「コネクタ接続プール」の順にクリックします。コネクタ接続プールにはプロパティー(名前と値のペア)を追加できます。または、次の`asadmin`コマンドを使用して、接続プールを作成および削除することもできます。

- `create-connector-connection-pool`(1)
- `delete-connector-connection-pool`(1)

コネクタリソース

コネクタリソースは、アプリケーションにエンタープライズ情報システム (EIS) への接続を提供します。どのコネクタリソースも、接続プールに関連付けられています。コネクタリソースを表示、作成、編集、または削除するには、管理コンソールで「リソース」>「コネクタ」>「コネクタリソース」の順にクリックします。または、次の `asadmin` コマンドを使用して、接続リソースを作成および削除することもできます。

- `create-connector-resource`(1)
- `delete-connector-resource`(1)

管理対象オブジェクトのリソース

管理対象オブジェクトは、リソースアダプタとその関連の EIS に固有のパーサーへのアクセスなど、特化した機能をアプリケーションに提供します。管理対象オブジェクトを表示、作成、編集、または削除するには、管理コンソールで「リソース」>「コネクタ」>「管理オブジェクトリソース」の順にクリックします。

- `create-admin-object`(1)
- `delete-admin-object-1`(1)

◆ ◆ ◆ 第 8 章

J2EE コンテナ

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

J2EE コンテナのタイプ

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、26 ページの「Application Server のアーキテクチャー」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、エンティティー Beans、および メッセージ駆動型 Beans があります。セッション Beans は一時的なオブジェクトやプロセスを表し、通常は 1 つのクライアントが使用します。エンティティー Beans は通常データベースに保持されている持続性データを表します。メッセージ駆動型 Beans は、メッセージを非同期でアプリケーションモジュールやサービスに渡すために使われます。

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

J2EE コンテナの設定

一般的な Web コンテナ設定の設定

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

Web コンテナセッションの設定

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

セッションタイムアウト値の設定

管理コンソール を使用して、HTTP セッションのタイムアウト値を設定します。セッションタイムアウト値は、HTTP セッションが有効である時間を表します。

管理コンソール で、「設定」>「Web コンテナ」>「セッションプロパティー」の順にクリックします。「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。

セッションタイムアウト値の設定方法の詳細については、管理コンソール で「ヘルプ」をクリックしてください。

マネージャープロパティーの設定

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

管理コンソール でセッションマネージャーの設定を変更するには、「設定」>「Web コンテナ」>「マネージャープロパティー」の順にクリックします。

- 設定するインスタンスを選択します。

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

  すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。

「マネージャープロパティー」タブで、次のプロパティーを設定します。

- リープ間隔の値。「リープ間隔」フィールドで、非アクティブな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。
- 最大セッション数の値。「最大セッション数」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。
- 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。
- セッション ID ジェネレータクラス名の値。

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

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

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

public interface UuidGenerator {

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

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

マネージャープロパティーの設定方法の詳細については、管理コンソールで「ヘルプ」をクリックしてください。

ストアプロパティーの設定

セッションストアデータの保存先を指定するには、管理コンソールで、「設定」>「Web コンテナ」>「ストアプロパティー」の順にクリックします。

- 設定するインスタンスを選択します。

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

  すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
- リープ間隔の値を設定します。

  「リープ間隔」のフィールドで、非アクティブな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。
- セッションデータの保存先のディレクトリを指定します。

セッションストアプロパティーの設定方法の詳細については、管理コンソールで「ヘルプ」をクリックしてください。

仮想サーバー設定の設定

Application Server をインストールすると、Application Server インスタンスに対してデフォルトの仮想サーバーが作成されます。この仮想サーバーのデフォルトの `docroot` は、*instance-dir*`domains/domain1/docroot` に作成されます。これは、*instance_name* `/docroot` と同期化されます。仮想サーバーは、追加で作成するそれぞれの Application Server インスタンスに対して作成されます。

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。

- 87 ページの「セッション格納位置」
- 87 ページの「EJB プール設定の設定」
- 87 ページの「EJB キャッシュ設定の設定」

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。

セッション格納位置

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

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

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

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

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

EJB プール設定の設定

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 とエンティティー Beans だけに適用されます。

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

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

EJB キャッシュ設定の設定

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 とエンティティー Beans だけです。

キャッシュされた Enterprise JavaBeans は、アクティブ、アイドル、または非活性化のうち、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。

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、87 ページの「EJB プール設定の設定」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駆動型 Beans のプールを保持しています。

このプールの設定を調整するには、次の手順に従います。

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。

◆ ◆ ◆ 第 9 章

9 セキュリティーの設定

セキュリティーとはデータの保護に関することであり、ストレージ内または伝送中のデータへの不正アクセスやそうしたデータの破損をどのようにして防止するか、ということです。Application Server には、J2EE 標準に基づく動的で拡張可能なセキュリティーアーキテクチャーがあります。標準実装されているセキュリティー機能には、暗号化、認証と承認、および公開鍵インフラストラクチャーがあります。Application Server は Java セキュリティーモデルをベースに構築され、アプリケーションが安全に動作できるサンドボックスを使用するため、システムやユーザーにリスクが及ぶ可能性がありません。この章では、以下の内容について説明します。

アプリケーションおよびシステムセキュリティーについて

概して、2 種類のアプリケーションセキュリティーがあります。

- 「プログラムによるセキュリティー」。開発者が記述したアプリケーションコードがセキュリティー動作を処理します。管理者が、このメカニズムを操作する必要はまったくありません。一般的に、プログラムによるセキュリティーは、J2EE コンテナで管理するのではなく、アプリケーションのセキュリティー設定をハードコード化するのでお勧めできません。

- 「宣言によるセキュリティー」。Application Server のコンテナがアプリケーションの配備記述子によりセキュリティーを処理します。宣言によるセキュリティーは、配備記述子を直接または deploytool などのツールで編集することによって操作できます。配備記述子はアプリケーションの開発後に変更可能なので、宣言によるセキュリティーの方が柔軟性に富んでいます。

アプリケーションによるセキュリティーのほかに、Application Server システムのアプリケーション全体に影響するシステムセキュリティーもあります。

プログラムによるセキュリティーはアプリケーション開発者により制御されるため、このマニュアルでは説明していません。宣言によるセキュリティーについては、このマニュアルである程度説明しています。このマニュアルは、主にシステム管理者を対象としているため、システムセキュリティーを中心に説明しています。

セキュリティー管理用ツール

Application Server には次のセキュリティー管理用ツールがあります。

- 管理コンソール。サーバー全体のセキュリティー設定、ユーザー、グループ、レルムの管理、およびほかのシステム全体のセキュリティータスクの実行に使用するブラウザベースのツール。管理コンソールの概要については、29 ページの「管理用ツール」を参照してください。管理コンソール で実行可能なセキュリティータスクの概要については、104 ページの「管理コンソールによるセキュリティーの管理」を参照してください。
- asadmin。管理コンソールと同じタスクの多くを行うコマンド行ツール。管理コンソールからできない操作でも、asadmin を使用して操作できる場合があります。コマンドプロンプトまたはスクリプトのいずれかから asadmin コマンドを実行して、繰り返しのタスクを自動化します。asadminの概要については、29 ページの「管理用ツール」を参照してください。
- deploytool。個別のアプリケーションセキュリティーを制御するために、アプリケーション配備記述子を編集するグラフィカルパッケージ化ツールおよび配備ツール。deploytool はアプリケーション開発者を対象にしているため、このマニュアルでは使用方法の詳細な説明はしていません。deploytool の使用手順については、このツールのオンラインヘルプおよび http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html の『The J2EE 1.4 Tutorial』を参照してください。

Java 2 プラットフォーム、J2SE (Java 2 Standard Edition) には、次の 2 つのセキュリティー管理用ツールがあります。

- keytool。デジタル証明書および鍵のペアの管理に使用するコマンド行ユーティリティー。keytool は、certificate レルムのユーザー管理に使用します。
- policytool。システム全体の Java セキュリティーポリシー管理に使用するグラフィカルユーティリティー。管理者が policytool を使用することはほとんどありません。

keytool、policytool、およびその他の Java セキュリティーツールの使用方法の詳細については、http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html#security の「Java 2 SDK Tools and Utilities」を参照してください。

Enterprise Edition では、NSS (Network Security Services) を実装した 2 つのセキュリティー管理ツールも利用可能です。NSS の詳細については、http://www.mozilla.org/projects/security/pki/nss/ を参照してください。それらのセキュリティー管理ツールは次のとおりです。

- certutil。証明書および鍵データベースの管理に使用されるコマンド行ユーティリティー。
- pk12util。証明書または鍵データベースと PKCS12 形式のファイル間における鍵と証明書のインポートおよびエクスポートに使用されるコマンド行ユーティリティー。

certutil、pk12util、およびその他の NSS セキュリティーツールの使用方法の詳細については、http://www.mozilla.org/projects/security/pki/nss/tools の「NSS Security Tools」を参照してください。

パスワードのセキュリティー管理

Application Server のこのリリースでは、特定のドメインの仕様が収められるファイル domain.xml 内に、Sun Java System Message Queue ブローカのパスワードが、あらかじめ平文で格納されています。このパスワードを収めた domain.xml ファイルの要素は、jms-host 要素の admin-password 属性です。このパスワードは変更不可能なので、インストールの際、セキュリティーに重大な影響は与えません。

ただし、管理コンソールを使用してユーザーやリソースを追加し、そのユーザーやリソースにパスワードを割り当ててください。このパスワードの中には、たとえばデータベースにアクセスするためのパスワードなど、平文で domain.xml ファイルに記述されているものがあります。このようなパスワードを平文で domain.xml ファイルに保持すると、セキュリティー上の危険を引き起こす可能性があります。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。セキュリティーパスワードを管理する手順は、次のトピックを参照してください。

- 91 ページの「domain.xml ファイル内のパスワードを暗号化する」
- 92 ページの「エンコード化されたパスワードを含むファイルの保護」
- 92 ページの「マスターパスワードの変更」
- 93 ページの「マスターパスワードとキーストアを操作する」
- 94 ページの「管理パスワードの変更」

domain.xml ファイル内のパスワードを暗号化する

domain.xml ファイル内のパスワードを暗号化するには、次の手順に従います。

1. domain.xml ファイルが格納されているディレクトリ (デフォルトでは *domain-dir*/config) から、次の asadmin コマンドを実行します。

   ```
   asadmin create-password-alias --user admin alias-name
   ```

   以下にその例を示します。

   ```
   asadmin create-password-alias --user admin jms-password
   ```

   パスワードプロンプトが表示されます。この場合は admin です。詳細については、create-password-alias、list-password-aliases、delete-password-alias コマンドのマニュアルページを参照してください。
2. domain.xml のパスワードの削除および置き換えを行います。これは、asadmin set コマンドを使用して行います。このような目的での set コマンドの使用例は、次のとおりです。

   ```
   asadmin set --user admin server.jms-service.jms-host.
   default_JMS_host.admin-password='${ALIAS=jms-password}'
   ```

   注 – エイリアスのパスワードは、例に示すように単一引用符で囲みます。
3. 該当するドメインの Application Server を再起動します。

エンコード化されたパスワードを含むファイルの保護

ファイルの中にはエンコード化されたパスワードを含むものがあり、ファイルシステムのアクセス権を使用しての保護が必要になります。これらのファイルには次のものが含まれます。

- *domain-dir*/master-password

  このファイルにはエンコード化されたマスターパスワードが含まれているので、ファイルシステムのアクセス権 600 で保護する必要があります。
- asadmin への --passwordfile 引数を使用して、引数として渡すために作成されたすべてのパスワードファイル。これらは、ファイルシステムのアクセス権 600 で保護する必要があります。

マスターパスワードの変更

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ー全体の要なので、ユーザーが必要に応じて手動で入力した

り、またはファイルに隠蔽したりすることができます。これは、システムで最高の機密データです。ユーザーは、このファイルを削除することで、強制的に MP の入力を要求できます。マスターパスワードが変更されると、Java JCEKS タイプのキーストアである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の変更手順は次のとおりです。

1. ドメインの Application Server を停止します。新旧のパスワードの入力を促す `asadmin change-master-password` コマンドを使用して、依存するすべての項目を再暗号化してください。次に例を示します。

```
asadmin change-master-password>
Please enter the master password>
Please enter the new master password>
Please enter the the new master password again>
```

2. Application Server を再起動します。

注意 – この時点で、実行中のサーバーインスタンスを開始してはいけません。対応するノードエージェントの SMP が変更されるまでは、決して実行中のサーバーインスタンスを再起動しないでください。SMP が変更される前にサーバーインスタンスを再起動すると、起動に失敗します。

3. 各ノードエージェントおよび関連するサーバーを 1 つずつ停止します。`asadmin change-master-password` コマンドをもう一度実行してから、ノードエージェントおよび関連するサーバーを再起動してください。
4. すべてのノードエージェントで対応が終了するまで、次のノードエージェントで同様の作業を継続します。このようにして、継続的な変更作業が完了します。

マスターパスワードとキーストアを操作する

マスターパスワードは、セキュリティー保護されたキーストアのパスワードです。新しいアプリケーションサーバードメインが作成されると、新しい自己署名付き証明書が生成されて、関連キーストアに格納されます。このキーストアは、マスターパスワードでロックされます。マスターパスワードがデフォルトではない場合、`start-domain` コマンドにより、マスターパスワードが要求されます。正しいマスターパスワードが入力されると、ドメインが起動します。

ドメインに関連付けられたノードエージェントが作成されると、ノードエージェントはデータをドメインと同期化させます。その間に、キーストアも同期化されます。このノードエージェントによって制御されるサーバーインスタンスは、キーストアを開く必要があります。このストアは、ドメイン作成処理で作成されたストアと基本的に同一であるため、同一のマスターパスワードでのみ開くことができます。マスターパスワード自体は、同期化されることはなく、同期中にノードエー

ジェントには転送されません。しかし、ローカルでノードエージェントが利用できるようにする必要があります。このため、ノードエージェントの作成または起動、あるいはその両方の場合に、マスターパスワードが要求され、かつドメインの作成や起動のときに入力したパスワードと同じものを入力する必要があるのです。マスターパスワードがドメインで変更された場合は、ドメインと関連付けられている各ノードエージェントでも同じ手順で変更する必要があります。

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、91 ページの「パスワードのセキュリティー管理」を参照してください。管理パスワードの暗号化は強く推奨されています。管理パスワードを暗号化する前に変更する場合は、`asadmin set` コマンドを使用してください。このような目的での `set` コマンドの使用例は、次のとおりです。

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

管理コンソールを使って管理パスワードを変更することもできます。その手順を次に示します。

管理コンソールを使って管理パスワードを変更するには、「設定」ノード > 設定するインスタンス >「セキュリティー」ノード >「レルム」ノード >「`admin-realm`」ノードの順に選択し、レルムページを編集します。

認証と承認について

認証と承認は、アプリケーションサーバーセキュリティーの中心的な概念です。ここでは、認証と承認に関連する次の項目について説明します。

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

エンティティーの認証

「認証」とは、あるエンティティー (ユーザー、アプリケーション、またはコンポーネント) が別のエンティティーが主張している本人であることを確認する方法です。エンティティーは、「セキュリティー資格」を使用して自らを認証します。資格には、ユーザー名、パスワード、デジタル証明書などが含まれます。

通常、認証はユーザー名とパスワードでユーザーがアプリケーションにログインすることを意味していますが、アプリケーションがサーバーのリソースを要求すると

き、セキュリティー資格を提供する EJB を指す場合もあります。普通、サーバーやアプリケーションはクライアントに認証を要求しますが、さらにクライアントもサーバーに自らの認証を要求できます。双方向で認証する場合、これを相互認証と呼びます。

エンティティーが保護対象リソースにアクセスを試行する場合、Application Server はそのリソースに対して設定されている認証メカニズムを使用してアクセスを認可するかどうかを決定します。たとえば、ユーザーが Web ブラウザでユーザー名およびパスワードを入力でき、アプリケーションがその資格を確認する場合、そのユーザーは認証されます。それ以降のセッションで、ユーザーはこの認証済みのセキュリティー ID に関連付けられます。

94 ページの「エンティティーの認証」で説明しているように、Application Server は 4 種類の認証をサポートします。アプリケーションは、配備記述子で使用する認証タイプを指定します。`deploytool` を使ってアプリケーションの認証メソッドを設定する方法の詳細については、`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html` の『J2EE 1.4 Tutorial』を参照してください。

表 9–1　Application Server の認証メソッド

認証メソッド	通信プロトコル	説明	ユーザー資格の暗号化
基本	HTTP (オプションで SSL)	サーバーのビルトインポップアップログインダイアログボックスを使用します。	SSL を使用しないかぎりありません。
フォームベース	HTTP (オプションで SSL)	アプリケーションが独自仕様のカスタムログインおよびエラーページを提供します。	SSL を使用しないかぎりありません。
クライアント証明書	HTTPS (HTTP over SSL)	サーバーは公開鍵証明書を使用してクライアントを認証します。	SSL

シングルサインオンの確認

シングルサインオンとは、1 つの仮想サーバーインスタンスの複数のアプリケーションがユーザー認証状態を共有することを可能にするものです。シングルサインオンによって、1 つのアプリケーションにログインしたユーザーは、同じ認証情報が必要なほかのアプリケーションに暗黙的にログインするようになります。

シングルサインオンはグループに基づいています。配備記述子が同じ「グループ」を定義し、かつ同じ認証メソッド (基本、フォーム、ダイジェスト、証明書) を使用するすべての Web アプリケーションはシングルサインオンを共有します。

デフォルトでシングルサインオンは、Application Server に定義された仮想サーバーで有効です。

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、97 ページの「ユーザー、グループ、ロール、およびレルムについて」を参照してください。

JACC プロバイダの指定

JACC (Java Authorization Contract for Containers) は J2EE 1.4 仕様の一部で、プラグイン可能な承認プロバイダ用のインタフェースを定義しています。これによって、管理者は認証を行うためにサードパーティー製のプラグインモジュールを設定できます。

デフォルトで、Application Server は JACC 仕様に準拠する単純なファイルベースの承認エンジンを提供します。その他のサードパーティー製の JACC プロバイダを指定することもできます。

JACC プロバイダは JAAS (Java Authentication and Authorization Service) の API を使用します。JAAS によって、サービスが認証およびユーザーに対するアクセス制御を行うことが可能になります。これは、標準 PAM (Pluggable Authentication Module) フレームワークの Java テクノロジバージョンを実装しています。

認証および承認の決定の監査

Application Server は、「監査モジュール」によってすべての認証および承認の決定の監査トレールを提供します。Application Server は、デフォルトの監査モジュールのほか、監査モジュールのカスタマイズ機能も提供します。カスタム監査モジュールの開発については、『Application Server Developer's Guide』の「Configuring an Audit Module」の節を参照してください。

メッセージセキュリティーの設定

「メッセージセキュリティー」によって、サーバーは、メッセージレイヤーで Web サービスの呼び出しおよび応答をエンドツーエンドで認証できます。Application Server は、SOAP レイヤーのメッセージセキュリティープロバイダを使用して、メッセージセキュリティーを実装します。メッセージセキュリティープロバイダは、要求メッセージおよび応答メッセージに必要な認証のタイプなどの情報を提供します。サポートされている認証には次のタイプが含まれます。

- ユーザー名とパスワード認証を含む送信者認証。

- XML デジタル署名を含むコンテンツ認証。

このリリースには、2 つのメッセージセキュリティープロバイダが付属しています。メッセージセキュリティープロバイダは、SOAP レイヤーの認証用に設定されます。設定可能なプロバイダには、`ClientProvider` と `ServerProvider` があります。

メッセージレイヤーセキュリティーのサポートは、プラグイン可能な認証モジュールの形式で Application Server とそのクライアントコンテナに統合されています。Application Server では、メッセージレイヤーセキュリティーはデフォルトで無効になっています。

メッセージレベルのセキュリティーは、Application Server 全体または特定のアプリケーションあるいはメソッドに対して設定できます。Application Server レベルのメッセージセキュリティーの設定については、第 10 章を参照してください。アプリケーションレベルのメッセージセキュリティーの設定については、『Developer's Guide』の「Securing Applications」の章で説明されています。

ユーザー、グループ、ロール、およびレルムについて

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

- 97 ページの「ユーザー」:「Application Server で定義される」個別の ID。一般に、ユーザーとは、人物、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。
- 98 ページの「グループ」:「Application Server で定義される」一連のユーザー。共通の特性に基づいて分類されます。
- 98 ページの「ロール」:アプリケーションによって定義される指定した承認レベル。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。
- 99 ページの「レルム」:ユーザーとグループの情報、および関連するセキュリティー資格を含むリポジトリ。レルムは、「セキュリティーポリシードメイン」とも呼ばれます。

注 – ユーザーおよびグループは Application Server 全体で指定されますが、各アプリケーションは独自のロールを定義します。アプリケーションがパッケージ化されて配備される場合、次の図に例示されているように、アプリケーションはユーザーまたはグループとロールとの間のマッピングを指定します。

ユーザー

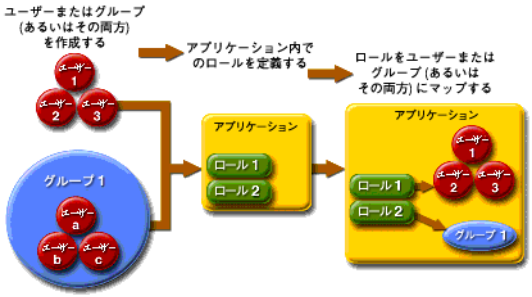

図 9–1 ロールマッピング

「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ことによって、多数のユーザーのアクセスを容易に制御できるようになります。

ロール

「ロール」は、ユーザーが、どのアプリケーションのどの部分にアクセスできるかと、何を実行できるかを定義します。つまり、ロールによってユーザーの承認レベルが決まります。

たとえば、人事アプリケーションの場合、電話番号とメールアドレスにはすべての社員がアクセスできますが、給与情報にアクセスできるのは管理職だけです。このアプリケーションでは少なくとも 2 つのロールが定義されます。`employee` と `manager` です。そして、`manager` ロールのユーザーだけに給与情報の表示が許可されます。

ロールはアプリケーション内での役割を定義するのに対し、グループはある方法で関連付けられているユーザーの集まりに過ぎません。この点で、ロールとユーザーグループは異なります。たとえば、人事アプリケーションには、`full-time`、`part-time`、および `on-leave` などのグループがありますが、これらすべてのグループのユーザーは `employee` ロールに所属します。

ロールはアプリケーション配備記述子で定義されます。反対に、グループはサーバーおよびレルム全体に対して定義されます。アプリケーション開発者または配備担当者は、配備記述子により各アプリケーション内で、ロールを 1 つまたは複数のグループに割り当てます。

レルム

「セキュリティーポリシードメイン」または「セキュリティードメイン」とも呼ばれている「レルム」とは、サーバーによって共通のセキュリティーポリシーが定義および適用される範囲のことです。実際には、レルムとはサーバーがユーザーおよびグループの情報を格納するリポジトリです。

Application Server では、3 つのレルムが事前に設定されています。`file` (初期のデフォルトレルム)、`certificate`、および `admin-realm` です。さらに、`ldap` レルム、`solaris` レルム、またはカスタムレルムも設定できます。アプリケーションは、その配備記述子でレルムを指定して使用できます。レルムを指定しない場合、Application Server はデフォルトレルムを使用します。

`file` レルムでは、サーバーはユーザー資格を `keyfile` という名前のファイルにローカルで格納します。管理コンソールを使用して `file` レルムのユーザーを管理できます。

`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、100 ページの「証明書および SSL の概要」を参照してください。

`admin-realm` は `FileRealm` でもあり、管理者ユーザーの資格を `admin-keyfile` という名前のファイルにローカルで格納します。`file` レルムでユーザーを管理するのと同じ方法で、管理コンソールを使用してこのレルムのユーザーを管理してください。

`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、管理コンソールのオンラインヘルプを参照してください。

証明書および SSL の概要

この節では、次の項目について説明します。

デジタル証明書について

「デジタル証明書」(または単に証明書) とは、インターネット上の人物やリソースを一意に識別する電子ファイルです。さらに証明書は 2 つのエンティティー間の安全で機密保護された通信を可能にします。

個人により使用される個人証明書や SSL (Secure Sockets Layer) テクノロジでサーバーとクライアント間の安全なセッションを確立するために使用されるサーバー証明書など、さまざまな種類の証明書があります。SSL の詳細については、101 ページの「SSL (Secure Sockets Layer) について」を参照してください。

証明書は「公開鍵暗号化」に基づき、意図した受信者だけが解読できるようデジタルの「鍵」(非常に長い数値) のペアを使用して「暗号化」、または符号化します。そして受信者は、情報を「復号化」して解読します。

鍵のペアには公開鍵と非公開鍵が含まれます。所有者は公開鍵を配布して、だれでも利用できるようにします。しかし、所有者は非公開鍵を決して配布せず、常時秘密にしておきます。鍵は数学的に関連しているので、1 つの鍵で暗号化されたデータは、そのペアのもう 1 つの鍵でだけ復号化ができます。

証明書とはパスポートのようなものです。所有者を識別し、その他の重要な情報を提供します。証明書は、「証明書発行局」(CA) と呼ばれる、信頼できるサードパーティーが発行します。CA はパスポートセンターに似ています。CA は、証明書の所有者の身元を確認したあと、偽造や改ざんができないように証明書に署名します。いったん CA が証明書に署名すると、所有者は ID の証明としてこれを提出することで、暗号化され、機密保護された通信を確立できます。

最も重要な点は、証明書によって所有者の公開鍵が所有者の ID と結び付けられることです。パスポートが写真とその所有者についての個人情報を結び付けるように、証明書は公開鍵とその所有者についての情報を結び付けます。

公開鍵のほかに、通常、証明書には次のような情報が含まれています。

- 所有者の名前、および証明書を使用する Web サーバーの URL や個人のメールアドレスなどその他の識別情報。
- 証明書が発行された CA の名前。
- 有効期限の日付。

デジタル証明書は、X.509 形式の技術仕様で管理されます。certificate レルムのユーザ ID を検証するために、 `certificate` 認証サービスは X.509 証明書の共通名フィールドを主体名として使用して、X.509 証明書を検証します。

証明書チェーンについて

Web ブラウザは、ブラウザが自動的に信頼する一連の「ルート」CA 証明書で事前に設定されます。別の場所で発行されたすべての証明書は、有効性を検証するために「証明書チェーン」を備えている必要があります。証明書チェーンとは、最後がルート CA 証明書で終わる、継続的な CA によって発行される一連の証明書です。

証明書が最初に生成される場合、それは「自己署名付き」証明書です。自己署名付き証明書とは、発行者 (署名者) が被認証者 (公開鍵が証明書で認証されているエンティティー) と同じものです。所有者は、証明書の署名要求 (CSR) を CA に送信するとき、その応答をインポートし、自己署名付き証明書が証明書のチェーンによって置き換えられます。チェーンの元の部分には、被認証者の公開鍵を認証する CA によって発行された証明書 (応答) があります。このチェーンの次の証明書は、CA の公開鍵を認証するものです。通常、これは自己署名付き証明書 (つまり、自らの公開鍵を認証する CA からの証明書) およびチェーンの最後の証明書です。

CA が証明書のチェーンに戻ることができる場合もあります。この場合、チェーンの元の証明書は同じ (キーエントリの公開鍵を認証する、CA によって署名された証明書) ですが、チェーン 2 番目の証明書が、CSR の送信先の CA の公開鍵を認証する、異なる CA によって署名された証明書です。そして、チェーンのその次の証明書は 2 番目の鍵を認証する証明書というように、自己署名付き「ルート」証明書に到達するまで続きます。こうして、チェーンの最初以降の各証明書は、チェーンの前にある証明書の署名者の公開鍵を認証します。

SSL (Secure Sockets Layer) について

「SSL」(Secure Sockets Layer) とは、インターネットの通信およびトランザクションのセキュリティー保護で最も普及している標準仕様です。Web アプリケーションは HTTPS (HTTP over SSL) を使用します。HTTPS は、サーバーとクライアント間のセキュアで機密保護された通信を確保するため、デジタル証明書を使用します。SSL 接続では、クライアントとサーバーの両方が送信前にデータを暗号化し、受信するとそれを復号化します。

クライアントの Web ブラウザがセキュアなサイトに接続する場合、次のように「SSL ハンドシェーク」が行われます。

- ブラウザはネットワークを介してセキュアなセッションを要求するメッセージを送信します。通常は、`http` ではなく `https` で始まる URL を要求します。
- サーバーは、公開鍵を含む証明書を送信することで応答します。

- ブラウザは、サーバーの証明書が有効であること、またサーバーの証明書が証明書をブラウザのデータベースに持つ信頼されている CA によって署名されていることを検証します。さらに、CA の証明書の有効期限が切れていないことも検証します。
- 証明書が有効な場合、ブラウザは 1 回限りの一意の「セッション鍵」を生成し、サーバーの公開鍵でそれを暗号化します。そして、ブラウザは暗号化されたセッション鍵をサーバーに送信し、両方でコピーを持てるようにします。
- サーバーは、非公開鍵を使用してメッセージを復号化し、セッション鍵を復元します。

ハンドシェークの後、クライアントは Web サイトの ID を検証し、クライアントと Web サーバーだけがセッション鍵のコピーを持ちます。これ以降、クライアントとサーバーはセッション鍵を使用して互いにすべての通信を暗号化します。こうすると、通信は確実にセキュアになります。

SSL 標準の最新バージョンは TLS (Transport Layer Security) と呼ばれています。Application Server は、SSL (Secure Sockets Layer) 3.0 および TLS (Transport Layer Security) 1.0 暗号化プロトコルをサポートしています。

SSL を使用するには、セキュアな接続を受け付ける各外部インタフェースまたは IP アドレスの証明書を、Application Server が所持しておく必要があります。ほとんどの Web サーバーの HTTPS サービスは、デジタル証明書がインストールされるまで実行されません。109 ページの「keytool ユーティリティーを使って証明書を生成する」の手順に従って、Web サーバーが SSL 用に使用できるデジタル証明書を設定してください。

暗号化方式について

「暗号化方式」とは、暗号化と復号化に使用される暗号化アルゴリズムです。SSL および TLS プロトコルは、サーバーとクライアントでお互いを認証するために使用される多くの暗号化方式のサポート、証明書の送信、およびセッション鍵の確立を行います。

安全度は、暗号化方式によって異なります。クライアントとサーバーは異なる暗号化方式群をサポートできます。SSL および TLS プロトコルから暗号化方式を選択してください。クライアントとサーバーは安全な接続のために、双方で通信に使用可能であるもっとも強力な暗号化方式を使用します。そのため、通常は、すべての暗号化方式を有効にすれば十分です。

名前ベースの仮想ホストの使用方法

セキュアなアプリケーションに名前ベースの仮想ホストを使用すると、問題が発生する場合があります。これは、SSL プロトコル自体の設計上の制約です。クライアントブラウザがサーバーの証明書を受け付ける SSL ハンドシェークは、HTTP 要求がア

クセスされる前に行われる必要があります。その結果、認証より前に仮想ホスト名を含む要求情報を特定できないので、複数の証明書を単一の IP アドレスに割り当てできません。

単一の IP すべての仮想ホストが同じ証明書に対して認証を必要とする場合、複数の仮想ホストを追加しても、サーバーの通常の SSL 動作を妨害する可能性はありません。ただし、証明書 (主に正式な CA の署名済みの証明書が該当) に表示されているドメイン名がある場合、ほとんどのブラウザがサーバーのドメイン名をこのドメイン名と比較することに注意してください。ドメイン名が一致しない場合、これらのブラウザは警告を表示します。一般的には、アドレスベースの仮想ホストだけが本稼働環境の SSL で広く使用されています。

ファイアウォールについて

「ファイアウォール」は、2 つ以上のネットワーク間のデータフローを制御し、ネットワーク間のリンクを管理します。ファイアウォールは、ハードウェア要素およびソフトウェア要素で構成できます。この節では、一般的ないくつかのファイアウォールアーキテクチャーとその設定について説明します。ここでの情報は、主に Application Server に関係するものです。特定のファイアウォールテクノロジの詳細については、使用しているファイアウォールのベンダーのドキュメントを参照してください。

一般的には、クライアントが必要な TCP/IP ポートにアクセスできるようにファイアウォールを設定します。たとえば、HTTP リスナーがポート 8080 で動作している場合は、HTTP 要求をポート 8080 だけで受け付けるようにファイアウォールを設定します。同様に、HTTPS 要求が ポート 8181 に設定されている場合は、HTTPS 要求をポート 8181 で受け付けるようにファイアウォールを設定する必要があります。

インターネットから EJB モジュールへ直接の RMI-IIOP (Remote Method Invocations over Internet Inter-ORB Protocol) アクセスが必要な場合は、同様に RMI-IIOP リスナーポートを開きますが、これにはセキュリティー上のリスクが伴うので、使用しないことを強くお勧めします。

二重のファイアウォールのアーキテクチャーでは、HTTP および HTTPS トランザクションを受け付けるように外部ファイアウォールを設定する必要があります。また、ファイアウォールの背後の Application Server と通信する HTTP サーバープラグインを受け付けるように内部ファイアウォールを設定する必要があります。

管理コンソールによるセキュリティーの管理

管理コンソールは、セキュリティーの次の側面を管理する手段を提供します。

サーバーセキュリティー設定

「セキュリティー設定」ページでは、デフォルトレルム、匿名ロール、およびデフォルトの主体ユーザー名とパスワードの指定を含む、サーバー全体のプロパティーを設定します。

レルムおよび file レルムユーザー

レルムの概念については、97 ページの「ユーザー、グループ、ロール、およびレルムについて」で紹介しています。

- 新しいレルムの作成
- 既存のレルムの削除
- 既存のレルムの設定変更
- `file` レルムのユーザーの追加、変更、および削除
- デフォルトレルムの設定

JACC プロバイダ

JACC プロバイダについては、96 ページの「JACC プロバイダの指定」で紹介しています。管理コンソールでは、次のタスクを実行します。

- 新しい JACC プロバイダの追加
- 既存の JACC プロバイダの削除または変更

監査モジュール

監査モジュールについては、96 ページの「認証および承認の決定の監査」で紹介しています。監査は、エラーやセキュリティー違反などの重大なイベントが発生した場合に、それを後から調べることができるようにイベントを記録するメソッドです。Application Server のログにすべての認証イベントが記録されます。完全なアクセスログには、Application Server で行われるすべてのアクセスイベントが連続して記録されます。

管理コンソールでは、次のタスクを実行します。

- 新しい監査モジュールの追加
- 既存の監査モジュールの削除または変更

メッセージセキュリティー

メッセージセキュリティーの概念については、96 ページの「メッセージセキュリティーの設定」で紹介しています。管理コンソールでは、次のタスクを実行します。

- メッセージセキュリティーの有効化
- メッセージセキュリティープロバイダの設定
- 既存のメッセージセキュリティー設定またはプロバイダの削除または設定

これらのタスクの詳細は、管理コンソール のオンラインヘルプを参照してください。

HTTP および IIOP リスナーのセキュリティー

HTTP サービスの各仮想サーバーは、1 つまたは複数の「HTTP リスナー」を介してネットワーク接続を提供します。

Application Server は、CORBA (Common Object Request Broker Architecture) オブジェクトをサポートしており、これはネットワークを介しての通信をするために IIOP (Internet Inter-Orb Protocol) を使用します。「IIOP リスナー」は、EJB コンポーネントのリモートクライアントおよびほかの CORBA ベースのクライアントから受信する接続を受け付けます。IIOP リスナーについての一般情報については、150 ページの「IIOP リスナー」を参照してください。

管理コンソールでは、次のタスクを実行します。

- 新しい HTTP または IIOP リスナーの作成と、それを使用するセキュリティーの指定
- 既存のHTTP または IIOP リスナーのセキュリティー設定の変更

管理サービスのセキュリティー

管理サービスは、サーバーインスタンスが通常のインスタンスか、ドメイン管理サーバー (DAS) か、あるいは組み合わせかを決定します。管理サービスを使用して、JSR-160 準拠のリモート JMX コネクタを設定してください。これは、ドメイン管理サーバーとノードエージェントとの間の通信を処理し、ノードエージェントは、リモートサーバーインスタンスの代わりに、ホストマシンのサーバーインスタンスを管理します。

管理コンソールでは、次のタスクを実行します。

- 管理サービスの管理
- JMX コネクタの編集
- JMX コネクタのセキュリティー設定の変更

セキュリティーマップ

管理コンソール を使用して、次のセキュリティーマッピングタスクを実行します。

- 既存のコネクタ接続プールへのセキュリティーマップの追加
- 既存のセキュリティーマップの削除または設定

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 JSSE (Java Secure Socket Extension) または NSS (Network Security Services) 形式のデジタル証明書が生成されます。デフォルトでは、Application Server は *domain-dir*`/config` ディレクトリの証明書データベースに、証明書情報を格納します。

- キーストアファイル。`key3.db` には、非公開鍵を含む Application Server の証明 書が格納されます。キーストアファイルはパスワードで保護されています。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、112 ページの「`certutil` ユーティリティーの使用」を参照してください。

  各キーストアエントリには一意のエイリアスがあります。インストール後の Application Server キーストア内には、エイリアス `s1as` を持つ単一のエントリが含まれています。

- トラストストアファイル。cert8.db には、ほかのエンティティーの公開鍵を含む Application Server の信頼できる証明書が格納されます。信頼できる証明書では、サーバーは証明書の公開鍵が証明書の所有者に属していることを確認しています。信頼できる証明書には、通常、証明書発行局 (CA) の証明書も含まれています。

  Platform Edition では、サーバー側で、Application Server は keytool を使用して証明書とキーストアを管理する JSSE 形式を使用します。Enterprise Edition では、サーバー側で、Application Server は certutil を使用して非公開鍵と証明書を格納する NSS データベースを管理する NSS を使用します。これらの両方の Edition で、クライアントサイド (アプリケーションクライアントまたはスタンドアロン) では、JSSE 形式を使用します。

  デフォルトで、Application Server は、サンプルアプリケーションで開発目的のために動作するキーストアおよびトラストストアを使用して設定されています。本稼動環境のために、証明書エイリアスを変更し、トラストストアにほかの証明書を追加し、キーストアおよびトラストストアファイルの名前と場所を変更する必要が生ずる可能性があります。

証明書ファイルの場所の変更

開発用として提供されているキーストアファイルとトラストストアファイルは、*domain-dir*/config ディレクトリに格納されています。

管理コンソールを使用して「server-config」ノード >「JVM 設定」>「JVM オプション」タブの順に展開し、証明書ファイルの新しい場所の値のフィールドを追加または変更します。

```
-Dcom.sun.appserv.nss.db=${com.sun.aas.instanceRoot}/NSS-database-directory
```

ここで、*NSS-database-directory* は NSS データベースの場所です。

JSSE (Java Secure Socket Extension) ツールの使用

keytool を使用して、JSSE (Java Secure Socket Extension) デジタル証明書を設定および操作します。Platform Edition、Enterprise Edition のどちらの場合も、クライアント側 (アプリケーションクライアントまたはスタンドアロン) では JSSE 形式が使用されます。

J2SE SDK に同梱されている keytool を使用すれば、管理者は、公開鍵と非公開鍵のペアおよび関連する証明書を管理できます。さらに、ユーザーは、通信接続先の公開鍵を証明書の形式でキャッシュできます。

keytool を実行するには、J2SE の /bin ディレクトリがパスの中に設定されているか、またはツールへのフルパスがコマンド行に存在するように、シェルの環境を設

定する必要があります。keytool の詳細については、keytool のドキュメント (http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html) を参照してください。

keytool ユーティリティーの使用

次の例は、JSSE ツールによる証明書処理に関する使用方法を示したものです。

- RSA 鍵アルゴリズムを使ってタイプ JKS のキーストア内に自己署名付き証明書を作成する。RSA は RSA Data Security, Inc. が開発した公開鍵暗号化技術です。この略語は、この技術の開発者である Rivest、Shamir、および Adelman を表しています。

  ```
  keytool  -genkey -noprompt -trustcacerts -keyalg RSA -alias ${cert.alias}
  -dname  ${dn.name} -keypass ${key.pass} -keystore ${keystore.file}
  -storepass ${keystore.pass}
  ```

  証明書を作成する別の例については、109 ページの「keytool ユーティリティーを使って証明書を生成する」を参照してください。
- デフォルトの鍵アルゴリズムを使ってタイプ JKS のキーストア内に自己署名付き証明書を作成する。

  ```
  keytool -genkey -noprompt -trustcacerts -alias ${cert.alias} -dname
  ${dn.name} -keypass ${key.pass} -keystore ${keystore.file} -storepass
  ${keystore.pass}
  ```

  証明書に署名する例については、110 ページの「keytool ユーティリティーを使ってデジタル証明書に署名する」を参照してください。
- タイプ JKS のキーストアで利用可能な証明書を表示する。

  ```
  keytool -list -v  -keystore ${keystore.file} -storepass ${keystore.pass}
  ```
- タイプ JKS のキーストア内の証明書情報を表示する。

  ```
  keytool  -list -v  -alias ${cert.alias} -keystore ${keystore.file}
  -storepass ${keystore.pass}
  ```
- RFC/テキスト形式の証明書を JKS ストア内にインポートする。証明書は、バイナリエンコーディングではなく、Internet RFC (Request for Comments) 1421 標準によって定義された印刷可能なエンコーディング形式を使って格納されることがしばしばあります。Base 64 エンコーディングとしても知られるこの証明書形式を使用すれば、電子メールなどの機構を使って証明書をほかのアプリケーションにエクスポートしやすくなります。

  ```
  keytool -import -noprompt -trustcacerts -alias ${cert.alias} -file
  ${cert.file} -keystore ${keystore.file} -storepass ${keystore.pass}
  ```

- タイプ JKS のキーストア内の証明書を PKCS7 形式でエクスポートする。「Public Key Cryptography Standards #7, Cryptographic Message Syntax Standard」によって定義された応答形式には、発行される証明書に加え、それをサポートする証明書チェーンも含まれます。

```
keytool  -export -noprompt  -alias ${cert.alias} -file ${cert.file}
-keystore ${keystore.file} -storepass ${keystore.pass}
```

- タイプ JKS のキーストア内の証明書を RFC/テキスト形式でエクスポートする。

```
keytool  -export -noprompt -rfc  -alias ${cert.alias} -file
${cert.file} -keystore ${keystore.file} -storepass ${keystore.pass}
```

- タイプ JKS のキーストアから証明書を削除する。

```
keytool  -delete -noprompt -alias ${cert.alias}  -keystore ${keystore.file}
-storepass ${keystore.pass}
```

  キーストアから証明書を削除する別の例については、111 ページの「keytool ユーティリティーを使って証明書を削除する」を参照してください。

keytool ユーティリティーを使って証明書を生成する

keytool を使用して証明書の生成、インポート、およびエクスポートを行います。デフォルトでは、keytool は実行元のディレクトリにキーストアファイルを作成します。

1. 証明書を実行すべきディレクトリに移動します。

   証明書の生成は常に、キーストアファイルとトラストストアファイルが格納されたディレクトリ (デフォルトでは *domain-dir*/config) 内で行います。これらのファイルの場所を変更する方法については、107 ページの「証明書ファイルの場所の変更」を参照してください。

2. 次の keytool コマンドを入力することで、キーストアファイル keystore.jks 内に証明書を生成します。

```
keytool -genkey -alias keyAlias-keyalg RSA
 -keypass changeit
 -storepass changeit
-keystore keystore.jks
```

   keyAlias には任意の一意名を指定します。キーストアまたは非公開鍵のパスワードをデフォルト以外の値に変更した場合には、前述のコマンドの changeit をその新しいパスワードで置き換えてください。

   プロンプトが表示され、keytool が証明書の生成に使用するユーザーの名前、組織、およびその他の情報の入力を求められます。

3. 次の keytool コマンドを入力することで、生成された証明書をファイル server.cer (または client.cer でもよい) にエクスポートします。

```
keytool -export -alias keyAlias-storepass changeit
 -file server.cer
 -keystore keystore.jks
```

4. 認証局によって署名された証明書が必要な場合は、110 ページの「keytool ユーティリティーを使ってデジタル証明書に署名する」を参照してください。
5. トラストストアファイル cacerts.jks を作成し、そのトラストストアに証明書を追加するには、次の keytool コマンドを入力します。

```
keytool -import -v -trustcacerts
-alias keyAlias
 -file server.cer
-keystore cacerts.jks
 -keypass changeit
```

6. キーストアまたは非公開鍵のパスワードをデフォルト以外の値に変更した場合には、前述のコマンドの changeit をその新しいパスワードで置き換えてください。

   このツールは、証明書に関する情報を表示し、その証明書を信頼するかどうかをユーザーに尋ねます。
7. yes と入力し、続いて Enter キーを押します。

   すると、keytool から次のようなメッセージが表示されます。

```
Certificate was added to keystore
[Saving cacerts.jks]
```

8. Application Server を再起動します。

keytool ユーティリティーを使ってデジタル証明書に署名する

デジタル証明書の作成後、所有者はそれに署名して偽造を防止する必要があります。E コマースのサイト、または ID の認証が重要であるサイトは、既知の証明書発行局 (CA) から証明書を購入できます。認証に心配がない場合、たとえば、非公開のセキュアな通信だけが必要な場合などは、CA 証明書の取得に必要な時間と費用を節約して、自己署名付き証明書を使用してください。

1. 証明書の鍵のペアを生成するため、CA の Web サイトの指示に従います。
2. 生成された証明書の鍵のペアをダウンロードします。

   キーストアファイルとトラストストアファイルが格納されたディレクトリ (デフォルトでは *domain-dir*/config ディレクトリ) 内に、証明書を保存します。107 ページの「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. keytool を使用して、証明書をローカルのキーストア、および必要に応じてローカルのトラストストアにインポートします。

```
keytool -import -v -trustcacerts
-alias keyAlias
 -file server.cer
-keystore cacerts.jks
 -keypass changeit
-storepass changeit
```

キーストアまたは非公開鍵のパスワードがデフォルト以外の値である場合には、前述のコマンドの `changeit` をその新しいパスワードで置き換えてください。

5. Application Server を再起動します。

keytool ユーティリティーを使って証明書を削除する

既存の証明書を削除するには、`keytool -delete` コマンドを使用します。次に例を示します。

```
keytool -delete
 -alias keyAlias
 -keystore keystore-name
 -storepass password
```

NSS (Network Security Services) ツールの使用

Enterprise Edition の場合、サーバー側では、NSS (Network Security Services) デジタル証明書を使って非公開鍵と証明書を格納するデータベースを管理します。クライアント側 (アプリケーションクライアントまたはスタンドアロン) では、107 ページの「JSSE (Java Secure Socket Extension) ツールの使用」で説明した JSSE 形式を使用します。

NSS (Network Security Services) を使ってセキュリティーを管理するためのツールは、次のとおりです。

- `certutil`。証明書および鍵データベースの管理に使用されるコマンド行ユーティリティー。`certutil` ユーティリティーの使用例については、112 ページの「certutil ユーティリティーの使用」を参照してください。
- `pk12util`。証明書または鍵データベースと PKCS12 形式のファイル間における鍵と証明書のインポートおよびエクスポートに使用されるコマンド行ユーティリティー。`pk12util` ユーティリティーの使用例については、114 ページの「pk12util ユーティリティーによる証明書のインポートとエクスポート」を参照してください。
- `modutil`。`secmod.db` ファイル内またはハードウェアトークン内の PKCS #11 モジュール情報を管理するためのコマンド行ユーティリティー。`modutil` ユーティリティーの使用例については、115 ページの「modutil による PKCS11 モジュールの追加と削除」を参照してください。

これらのツールは *install-dir*/lib/ ディレクトリに格納されています。NSS セキュリティーツールの場所を指し示すために、次の各環境変数が使用されます。

- LD_LIBRARY_PATH =${*install-dir*}/lib
- ${*os.nss.path*}

例に含まれる証明書の共通名 (CN) は、クライアントまたはサーバーの名前です。また、この CN は、SSL ハンドシェーク中に証明書の名前とその証明書の生成元であるホスト名とを比較する目的でも使用されます。SSL ハンドシェーク中に証明書名とホスト名が一致しなかった場合、警告または例外が生成されます。いくつかの例では便宜上、証明書の共通名 CN=localhost が使用されていますが、これは、すべてのユーザーが、実際のホスト名に基づいて新しい証明書を作成することなしにその証明書を使用できるようにするためです。

次の各節の例は、NSS ツールによる証明書処理に関する使用方法を示したものです。

- 112 ページの「certutil ユーティリティーの使用」
- 114 ページの「pk12util ユーティリティーによる証明書のインポートとエクスポート」
- 115 ページの「modutil による PKCS11 モジュールの追加と削除」

certutil ユーティリティーの使用

certutil を実行する前に必ず、このユーティリティーを実行するために必要なライブラリの場所が LD_LIBRARY_PATH で指定されていることを確認してください。この場所は、asenv.conf (製品全体の設定ファイル) の AS_NSS_LIB の値から特定できます。

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 NSS コマンド行ユーティリティーです。このユーティリティーは、cert8.db ファイルで、証明書の一覧表示、生成、変更、または削除を行い、key3.db ファイルで、パスワードの作成または変更、新しい公開鍵と非公開鍵のペアの生成、鍵データベースのコンテンツの表示、または鍵のペアの削除を行うこともできます。

通常、鍵と証明書の管理プロセスは鍵データベース内の鍵の作成から始まり、証明書データベース内の証明書の生成と管理に続きます。次のドキュメントでは、certutil ユーティリティーの構文を含む、NSS による証明書および鍵データベースの管理について説明しています。http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html。

次の箇条書きの各項目は、NSS および JSSE セキュリティーツールを使って証明書の作成または管理、あるいはその両方を行う例を示したものです。

- 自己署名付きのサーバー証明書およびクライアント証明書を生成する。この例では、CN は hostname.domain.[com|org|net|...] の形式でなければなりません。

この例では、*domain-dir*/config です。serverseed.txt ファイルと clientseed.txt ファイルには、任意のランダムテキストを含めることができます。このランダムテキストは、鍵ペア生成時に使用されます。

```
certutil -S -n $SERVER_CERT_NAME -x -t "u,u,u"
 -s "CN=$HOSTNAME.$HOSTDOMAIN, OU=Java Software, O=Sun Microsystems Inc.,
    L=Santa Clara, ST=CA, C=US"
 -m 25001 -o  $CERT_DB_DIR/Server.crt
 -d $CERT_DB_DIR -f passfile &lt;$CERT_UTIL_DIR/serverseed.txt
```

クライアント証明書を生成する。この証明書も自己署名付き証明書です。

```
certutil -S -n $CLIENT_CERT_NAME  -x -t "u,u,u"
 -s "CN=MyClient, OU=Java Software, O=Sun Microsystems Inc.,
    L=Santa Clara, ST=CA, C=US"
 -m 25002 -o  $CERT_DB_DIR/Client.crt
 -d $CERT_DB_DIR -f passfile &lt;$CERT_UTIL_DIR/clientseed.txt
```

- 前述の項目で生成された証明書を検証する。

```
certutil -V -u V -n $SERVER_CERT_NAME -d  $CERT_DB_DIR
certutil -V -u C -n $CLIENT_CERT_NAME -d  $CERT_DB_DIR
```

- 利用可能な証明書を表示する。

```
certutil -L -d  $CERT_DB_DIR
```

- RFC テキスト形式の証明書を NSS 証明書データベースにインポートする。

```
certutil -A -a -n ${cert.nickname} -t ${cert.trust.options}
-f ${pass.file} -i ${cert.rfc.file}
-d ${admin.domain.dir}/${admin.domain}/config
```

- NSS 証明書データベース内の証明書を RFC 形式でエクスポートする。

```
certutil -L -a -n ${cert.nickname} -f ${pass.file}
-d ${admin.domain.dir}/${admin.domain}/config > cert.rfc
```

- NSS 証明書データベースから証明書を削除する。

```
certutil -D -n ${cert.nickname} -f ${pass.file}
-d ${admin.domain.dir}/${admin.domain}/config
```

- 証明書を NSS データベースから JKS 形式に移動する。

```
certutil -L -a -n ${cert.nickname}
-d ${admin.domain.dir}/${admin.domain}/config > cert.rfc
keytool  -import -noprompt -trustcacerts -keystore ${keystore.file}
-storepass ${keystore.pass} -alias ${cert.alias} -file cert.rfc
```

pk12util ユーティリティーによる証明書のインポートとエクスポート

証明書または鍵データベースと PKCS12 形式のファイル間における鍵と証明書のインポートおよびエクスポートに使用されるコマンド行ユーティリティーは、pk12util です。PKCS12 は、「Public-Key Cryptography Standards (PKCS) #12, Personal Information Exchange Syntax Standard」です。pk12util ユーティリティーの詳細については、http://www.mozilla.org/projects/security/pki/nss/tools/pk12util.html を参照してください。

- PKCS12 形式の証明書を NSS 証明書データベースにインポートする。

```
pk12util -i ${cert.pkcs12.file} -k ${certdb.pass.file}
-w ${cert.pass.file} -d ${admin.domain.dir}/${admin.domain}/config
```

- PKCS12 形式の証明書を NSS 証明書データベーストークンモジュールにインポートする。

```
pk12util -i ${cert.pkcs12.file} -h ${token.name} -k ${certdb.pass.file}
-w ${cert.pass.file} -d ${admin.domain.dir}/${admin.domain}/config
```

- NSS 証明書データベース内の証明書を PKCS12 形式でエクスポートする。

```
pk12util -o -n ${cert.nickname} -k ${pass.file} -w${cert.pass.file}
-d ${admin.domain.dir}/${admin.domain}/config
```

- NSS 証明書データベーストークンモジュール内の証明書を PKCS12 形式でエクスポートする (ハードウェアアクセラレータ構成で有用)。

```
pk12util -o -n ${cert.nickname} -h ${token.name} -k ${pass.file}
-w ${cert.pass.file} -d ${admin.domain.dir}/${admin.domain}/config
```

- PKCS12 証明書を JKS 形式に変換する (Java ソースが必要)。

```
&lt;target name="convert-pkcs12-to-jks" depends="init-common">
   &lt;delete file="${jks.file}" failonerror="false"/>
   &lt;java classname="com.sun.enterprise.security.KeyTool">
      &lt;arg line="-pkcs12"/>
      &lt;arg line="-pkcsFile ${pkcs12.file}"/>
      &lt;arg line="-pkcsKeyStorePass ${pkcs12.pass}"/>
      &lt;arg line="-pkcsKeyPass ${pkcs12.pass}"/>
      &lt;arg line="-jksFile ${jks.file}"/>
      &lt;arg line="-jksKeyStorePass ${jks.pass}"/>
      &lt;classpath>
         &lt;pathelement path="${s1as.classpath}"/>
         &lt;pathelement path="${env.JAVA_HOME}/jre/lib/jsse.jar"/>
      &lt;/classpath>
   &lt;/java>
&lt;/target>
```

modutil による PKCS11 モジュールの追加と削除

「セキュリティーモジュールデータベースツール」である modutil は、secmod.db ファイル内またはハードウェアトークン内の PKCS #11 (Cryptographic Token Interface Standard) モジュール情報を管理するためのコマンド行ユーティリティーです。このツールを使用して、PKCS #11 モジュールを追加および削除し、パスワードを変更し、デフォルトを設定し、モジュールの内容を表示し、スロットを使用可または使用不可にし、FIPS-140-1 準拠を有効または無効にし、暗号化操作にデフォルトのプロバイダを割り当てることができます。また、このツールを使用すれば、key3.db、cert7.db、および secmod.db セキュリティーデータベースファイルを作成することもできます。このツールの詳細については、http://www.mozilla.org/projects/security/pki/nss/tools/modutil.html を参照してください。

- 新しい PKCS11 モジュールまたはトークンを追加する。

```
modutil -add ${token.module.name} -nocertdb -force -mechanisms RSA:DSA:RC4:DES
-libfile ${SCA.lib.path} -dbdir ${admin.domain.dir}/${admin.domain}/config
```

- NSS ストアから PKCS11 モジュールを削除する。

```
modutil -delete ${token.module.name} -nocertdb -force -mechanisms RSA:DSA:RC4:DES
-libfile ${SCA.lib.path} -dbdir ${admin.domain.dir}/${admin.domain}/config
```

- NSS ストア内で利用可能なトークンモジュールを一覧表示する。

```
modutil -list  -dbdir ${admin.domain.dir}/${admin.domain}/config
```

Application Server でのハードウェア暗号化アクセラレータの使用

ハードウェアアクセラレータトークンを使用すると、暗号化のパフォーマンスを向上させたり、セキュリティー保護された鍵ストレージ機能を備えたりすることができます。また、スマートカードを使用したモバイル用のセキュリティー保護された鍵ストレージを提供することもできます。

Java 2 Platform, Standard Edition (J2SE プラットフォーム) 5.0 で Sun Java System Application Server 8.1 および 8.2 の Standard Edition または Enterprise Edition を実行する場合は、SSL 通信や TLS 通信用の PKCS#11 トークンと、鍵および PKCS#11 トークンを管理するための Network Security Services (NSS) ツールの使用をサポートします。この節では、Application Server によるサポートの提供方法と、関連設定の手順を説明します。

J2SE 5.0 PKCS#11 プロバイダは、簡単に Application Server ランタイムと統合できます。これらのプロバイダによって、ハードウェアアクセラレータなどの PKCS#11 トークンを Application Server で使用して、高速なパフォーマンスを実現したり、SSL 通信や TLS 通信での非公開鍵の継承を防いだりすることができます。

ここでは、次の内容について説明します。

ハードウェア暗号化アクセラレータの設定について

Sun Java System Application Server 8.1 および 8.2 の Standard Edition または Enterprise Edition は、Sun Crypto Accelerator 1000 (SCA-1000) および SCA-4000 でテストされています。

Application Server, は、J2SE 5.0 と併用することにより、PKCS#11 トークンと通信できます。Application Server にパッケージされているのは、NSS PKCS#11 トークンライブラリ (NSS 内部 PKCS#11 モジュール用、一般に NSS ソフトトークンと呼ばれる) と、NSS コマンド行管理ツールです。詳細は、111 ページの「NSS (Network Security Services) ツールの使用」を参照してください。

NSS ツールを使用して PKCS#11 トークンと J2SE PKCS#11 プロバイダに鍵と証明書を作成し、実行時にトークンの鍵と証明書にアクセスします。PKCS#11 プロバイダは、暗号化サービスプロバイダで、ネイティブ PKCS#11 ライブラリのラッパーとして動作します。一般に、PKCS#11 トークンは、ネイティブ PKCS#11 インタフェースを使用してすべてのハードウェアとソフトウェアを参照します。ハードウェアトークンは、ハードウェアアクセラレータやスマートカードなどの物理デバイスに実装された PKCS#11 トークンです。ソフトウェアトークンは、完全にソフトウェアに実装された PKCS#11 トークンです。

注 – Application Server を J2SE 1.4.x プラットフォームで実行する場合、PKCS#11 トークン 1 つだけ、つまり NSS ソフトトークンがサポートされます。

Microsoft Windows 環境では、NSS ライブラリの位置 *AS_NSS* と NSS ツールディレクトリ *AS_NSS_BIN* を PATH 環境変数に追加してください。簡単にするために、この節では UNIX のコマンドだけを使用して手順を説明します。必要に応じて UNIX 変数を Windows 変数に置き換えるようにしてください。

ハードウェア暗号化アクセラレータの設定は、主に次の 2 つの手順に分かれます。

PKCS#11 トークンの設定

ここでは、NSS セキュリティーツール `modutil` を使用して PKCS#11 トークンを設定する方法について説明します。次の手順で PKCS#11 トークンを設定します。

次のコマンドを入力します。すべて 1 行に入力してください。

```
modutil -dbdir AS_NSS_DB -nocertdb -force -add moduleName -libfile
 absolute_path_of_pkcs11_library -mechanisms list_of_security_mechanisms
```

ここで *AS_NSS_DB* は NSS データベースのディレクトリになり、ドメイン管理サーバー (DAS) を使用する場合には `AS_DOMAIN_CONFIG` と同じになります。

たとえば、ハードウェアアクセラレータトークンを設定するには、次のように入力します。すべて 1 行に入力してください。

```
modutil -dbdir AS_NSS_DB -nocertdb -force -add "Sun Crypto Accelerator" -libfile
 /opt/SUNWconn/crypto/lib/libpkcs11.so -mechanisms RSA:DSA:RC4:DES
```

この例のハードウェアアクセラレータは SCA–1000 暗号化アクセラレータです。対応する PKCS#11 ライブラリは、デフォルトでは `/opt/SUNWconn/crypto/lib/libpkcs11.so` にあります。

`mechanisms` は、トークンで利用可能な暗号化メカニズムの完全なリストにしてください。利用可能な暗号化メカニズムの一部だけを使用する方法については、120 ページの「J2SE 5.0 PKCS#11 プロバイダの設定」を参照してください。サポートされるすべてのメカニズムのリストについては、NSS セキュリティーツールのサイト (`http://www.mozilla.org/projects/security/pki/nss/tools`) にある `modutil` のドキュメントを参照してください。

次の例では、トークンのインストール時に指定したトークン名が`mytoken` であるとします。

ハードウェアアクセラレータが正しく設定されていることを確認するには、次のコマンドを入力します。

```
modutil -list -dbdir AS_NSS_DB
```

標準出力表示は次のようになります。

```
Using database directory /var/opt/SUNWappserver/domains/domain1/config ...

Listing of PKCS#11 Modules
-----------------------------------------------------------
  1. NSS Internal PKCS#11 Module
         slots: 2 slots attached
        status: loaded

         slot: NSS Internal Cryptographic Services
        token: NSS Generic Crypto Services

         slot: NSS User Private Key and Certificate Services
        token: NSS Certificate DB

  2. Sun Crypto Accelerator
```

```
        library name: /opt/SUNWconn/crypto/lib/libpkcs11.so
         slots: 1 slot attached
        status: loaded

         slot: Sun Crypto Accelerator:mytoken
        token: mytoken
-----------------------------------------------------------
```

鍵と証明書の管理

ここでは、`certutil` と `pk12util` を使用して鍵や証明書を作成および管理する一般的な手順について説明します。`certutil` および `pk12util` の詳細については、111 ページの「NSS (Network Security Services) ツールの使用」、および NSS セキュリティーツールのサイト (`http://www.mozilla.org/projects/security/pki/nss/tools`) を参照してください。

注 – Java ランタイムの *JAVA_HOME*`/jre/lib/security` ディレクトリにある `java.security` プロパティーファイルで PKCS#11 プロバイダを設定することで、J2SE `keytool` ユーティリティーを使用して鍵や証明書を管理することもできます。`keytool` の使用の詳細は、`http://java.sun.com/j2se/1.5.0/docs/guide/secuirty/p11guide.html` の『Java PKCS#11 Reference Guide』を参照してください。

この節で説明する内容は、次のとおりです。

- 118 ページの「鍵や証明書の一覧表示」
- 119 ページの「非公開鍵と証明書の操作」

鍵や証明書の一覧表示

- 設定済み PKCS#11 トークンの鍵や証明書を表示するには、次のコマンドを実行します。

  ```
  certutil -L -d AS_NSS_DB [-h tokenname]
  ```

  たとえば、デフォルトの NSS ソフトトークンの内容を表示するには、次のように入力します。

  ```
  certutil -L -d AS_NSS_DB
  ```

  標準出力表示は次のようになります。

```
verisignc1g1                T,c,c
verisignc1g2                T,c,c
verisignc1g3                T,c,c
verisignc2g3                T,c,c
verisignsecureserver        T,c,c
verisignc2g1                T,c,c
verisignc2g2                T,c,c
verisignc3g1                T,c,c
verisignc3g2                T,c,c
verisignc3g3                T,c,c
s1as                        u,u,u
```

出力には、左側の列にトークンの名前、右側の列に 3 つの信頼属性が表示されます。Application Server 証明書の場合、通常は `T,c,c` です。信頼のレベルが 1 つしかない J2SE `java.security.KeyStore` API とは異なり、NSS テクノロジには信頼のレベルが複数あります。Application Server では、基本的に 1 番目の信頼属性に着目します。この信頼属性は、このトークンが SSL を使用する方法を示します。この属性の意味は次のとおりです。

`T` は、認証局 (CA) がクライアント証明書の発行に対して信頼されていることを表します。
`u` は、認証や署名で証明書や鍵を使用できることを表します。
`u,u,u` という属性の組み合わせは、非公開鍵がデータベースに存在することを表します。

- ハードウェアトークン `mytoken` の内容を表示するには、次のコマンドを実行します。

```
certutil -L -d AS_NSS_DB -h mytoken
```

ハードウェアトークンのパスワードが求められます。標準出力表示は次のようになります。

```
Enter Password or Pin for "mytoken":
mytoken:Server-Cert                                        &#9;u,u,u
```

非公開鍵と証明書の操作

自己署名付き証明書の作成や、証明書のインポート/エクスポートには、`certutil` を使用します。非公開鍵をインポートまたはエクスポートするには、`pk12util` ユーティリティーを使用します。詳細は、111 ページの「NSS (Network Security Services) ツールの使用」を参照してください。

注意 – Application Server では、`certutil` や `modutil` などの NSS ツールを使用して NSS パスワードを直接変更しないでください。そのようにすると、Application Server のセキュリティーデータが破損する可能性があります。

J2SE 5.0 PKCS#11 プロバイダの設定

Application Server は実行時に PKCS#11 トークン内にある鍵や証明書へのアクセスに、J2SE PKCS#11 プロバイダを使用します。デフォルトでは、Application Server では NSS ソフトトークン用に J2SE PKCS#11 プロバイダが設定されます。ここでは、J2SE PKCS#11 プロバイダのデフォルト設定をオーバーライドする方法について説明します。

Application Server では、PKCS#11 トークンごとに次のデフォルト PKCS#11 設定パラメータが生成されます。

- デフォルトの NSS ソフトトークン用の設定

```
name=internal
library=${com.sun.enterprise.nss.softokenLib}
nssArgs="configdir='${com.sun.appserv.nss.db}'
 certPrefix='' keyPrefix='' secmod='secmod.db'"
slot=2
omitInitialize = true
```

- SCA 1000 ハードウェアアクセラレータ用の設定

```
name=HW1000
library=/opt/SUNWconn/crypto/lib/libpkcs11.so
slotListIndex=0
omitInitialize=true
```

これらの設定は、『Java PKCS#11 Reference Guide』で説明されている構文に従います。

注 – name パラメータは、固有でなければならない場合を除き、必要ではありません。J2SE 5.0 の一部の以前のバージョンでは、英数字のみ使用できます。

デフォルトの設定パラメータをオーバーライドするには、カスタム設定ファイルを作成します。たとえば、SCA–1000 で RSA 暗号化方式と RSA 鍵ペアジェネレータを明示的に無効にすることができます。RSA 暗号化方式と RSA 鍵ペアジェネレータを無効にする詳細は、`http://www.mozilla.org/projects/security/pki/nss/tools` を参照してください。

カスタム設定ファイルを作成するには、次の手順に従います。

1. 次のコードを記述した *install-dir*/mypkcs11.cfg という設定ファイルを作成して保存します。

```
name=HW1000
library=/opt/SUNWconn/crypto/lib/libpkcs11.so
slotListIndex=0
disabledMechanisms = {
&#9;CKM_RSA_PKCS
&#9;CKM_RSA_PKCS_KEY_PAIR_GEN
}
omitInitialize=true
```

2. 必要に応じて NSS データベースを更新します。この場合は、RSA を無効にするために NSS データベースを更新します。

   以下のコマンドを実行します。

```
modutil -undefault "Sun Crypto Accelerator" -dbdir AS_NSS_DB -mechanisms RSA
```

   mechanisms リストのアルゴリズム名は、デフォルト設定のアルゴリズム名とは異なります。NSS で有効な mechanisms のリストについては、NSS セキュリティーツールのサイト (http://www.mozilla.org/projects/security/pki/nss/tools) にある modutil のドキュメントを参照してください。

3. 次のように、適切な位置にプロパティーを追加して、この変更でサーバーを更新します。

```
&lt;property name="mytoken" value="&InstallDir;/mypkcs11.cfg"/>
```

   プロパティーの位置は、次のいずれかにします。

   - プロバイダが DAS またはサーバーインスタンス用である場合は、関連 <security-service> の下にプロパティーを追加します。
   - プロバイダがノードエージェント用である場合は、domain.xml ファイルで関連 <node-agent> 要素の下にプロパティーを追加します。

4. Application Server を再起動します。

   カスタマイズされた設定が再起動後に有効になります。

詳細情報

- Java 2 Standard Edition のセキュリティーについては、http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html を参照してください。
- 『J2EE 1.4 Tutorial』の「Security」の章については、http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html を参照してください。

- 『管理ガイド』の第 10 章。
- 『Developer's Guide』の「Securing Applications」の章。

◆ ◆ ◆ 第 10 章

メッセージセキュリティーの設定

この章の一部の内容は、セキュリティーと Web サービスに関する基本概念の理解を前提としてます。この章を読む前にこれらの概念について学ぶには、121 ページの「詳細情報」に記載されているリソースを参照してください。

この章では、Application Server で Web サービスのメッセージレイヤーセキュリティーを設定する方法について説明します。この章の内容は、次のとおりです。

メッセージセキュリティーの概要

「メッセージセキュリティー」を使用する場合、メッセージ内にセキュリティー情報が挿入され、その情報がメッセージとともにネットワークレイヤー経由でメッセージの送信先に届けられます。メッセージセキュリティーは、『J2EE 1.4 Tutorial』の「Security」の章で説明されているトランスポートレイヤーセキュリティーとは異なり、メッセージトランスポートからメッセージ保護を分離して伝送後もメッセージを保護されたままにするために使用することができます。

「Web Services Security: SOAP Message Security (WS-Security)」は、米国 Sun Microsystems, Inc. を含むすべての主要な Web サービステクノロジプロバイダによって共同開発された、相互運用可能な Web サービスセキュリティーを実現するための OASIS 国際標準です。WS-Security のメッセージセキュリティーメカニズムは、SOAP 経由で送信される Web サービスメッセージを XML 暗号化と XML デジタル署名を使ってセキュリティー保護する、というものです。WS-Security 仕様には、X.509 証明書、SAML アサーション、ユーザー名/パスワードなどの各種セキュリティートークンを使って SOAP Web サービスメッセージの認証および暗号化を実現する方法が規定されています。

WS-Security 仕様については、
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf を参照してください。

Application Server のメッセージセキュリティーの理解

Application Server は、Web サービスのクライアント側コンテナとサーバー側コンテナにおいて、WS-Security 標準に対する統合化されたサポートを提供します。この機能は統合化されているため、Application Server のコンテナがアプリケーションに代わって Web サービスセキュリティーを適用します。また、そうしたセキュリティーで Web サービスアプリケーションを保護する際、アプリケーションの実装を変更する必要はありません。Application Server は、これを実現する目的で、SOAP レイヤーメッセージセキュリティープロバイダとメッセージ保護ポリシーを、コンテナおよびコンテナ内に配備されたアプリケーションにバインドする機能を提供しています。

メッセージセキュリティーの責任の割り当て

Application Server でメッセージセキュリティー設定の主要責任者として期待されるのは、124 ページの「システム管理者」ロールと125 ページの「アプリケーション配備担当者」ロールです。場合によっては、125 ページの「アプリケーション開発者」もその責任の一端を担うことがありますが、通常は、システム管理者またはアプリケーション配備者のいずれかのロールが既存アプリケーションをセキュリティー保護し、開発者が関与することも、実装が変更されることもありません。次の各節では、各種ロールの責任を定義します。

システム管理者

システム管理者は次の責任を負います。

- Application Server 上のメッセージセキュリティープロバイダの設定。
- ユーザーデータベースの管理。
- キーストアおよびトラストストアファイルの管理。
- 暗号化を使用し、バージョン 1.5.0 より前のバージョンの Java SDK を実行している場合の JCE (Java Cryptography Extension) プロバイダの設定。
- サンプルサーバーのインストール。ただし、これを行うのは、`xms` サンプルアプリケーションを使ってメッセージレイヤー Web サービスセキュリティーの使用方法を示す場合だけです。

システム管理者は、管理コンソールを使用してサーバーセキュリティーの設定を管理し、コマンド行ツールを使用して証明書データベースを管理します。Platform Edition の証明書と非公開鍵は、キーストア内に格納され、`keytool` を使って管理されます。Standard Edition と Enterprise Edition の証明書と非公開鍵は、NSS データベース

内に格納され、certutil を使って管理されます。このマニュアルは主にシステム管理者を対象にしています。メッセージセキュリティータスクの概要については、130 ページの「メッセージセキュリティーのための Application Server の設定」を参照してください。

アプリケーション配備担当者

アプリケーション配備担当者は次の責任を負います。

- 必要なすべてのアプリケーション固有メッセージ保護ポリシーをアプリケーションアセンブリ時に指定 (それらのポリシーが上流行程の役割 (開発者またはプログラマ) によって指定されていなかった場合)。
- Sun 固有の配備記述子を変更し、アプリケーション固有メッセージ保護ポリシー情報 (message-security-binding 要素) を Web サービスエンドポイントとサービス参照に指定。

これらのセキュリティータスクについては、『Developers' Guide』の「Securing Applications」の章で説明されています。この章へのリンクについては、121 ページの「詳細情報」を参照してください。

アプリケーション開発者

アプリケーション開発者はメッセージセキュリティーを有効にできますが、そのようにする責任はありません。メッセージセキュリティーの設定をシステム管理者が行う場合、すべての Web サービスがセキュリティー保護されます。コンテナにバインドされているプロバイダまたは保護ポリシーと異なるものをアプリケーションにバインドする必要がある場合、アプリケーション配備担当者がメッセージセキュリティーの設定を行います。

アプリケーション開発者またはプログラマは次の責任を負います。

- アプリケーション固有メッセージ保護ポリシーがアプリケーションで必要かどうかの判断。必要な場合、その必要なポリシーがアプリケーションアセンブリで指定されているかどうかの確認。それにはアプリケーション配備担当者に連絡します。

セキュリティートークンとセキュリティーメカニズムについて

WS-Security 仕様は、セキュリティートークンを使って SOAP Web サービスメッセージを認証および暗号化するための拡張可能なメカニズムを提供します。Application Server とともにインストールされる SOAP レイヤーメッセージセキュリティープロバイダを使えば、ユーザー名 / パスワードセキュリティートークンと X.509 証明書セキュリティートークンによる SOAP Web サービスメッセージの認証と暗号化を行え

ます。Application Server の今後のリリースでは、SAML アサーションなどのほかのセキュリティートークンを採用したプロバイダも追加される予定です。

ユーザー名トークンについて

Application Server は、SOAP メッセージ内で「ユーザー名トークン」を使ってメッセージ「送信者」の認証 ID を確立します。パスワードが埋め込まれたユーザー名トークンを含むメッセージの受信者は、そのメッセージの送信者がそのトークンによって識別されるユーザーとして振る舞うことを許可されているかどうかを検証するために、その送信者がユーザーの秘密情報 (パスワード) を知っているかどうかを確認します。

ユーザー名トークンを使用する場合、有効なユーザーデータベースを Application Server 上に設定する必要があります。

デジタル署名について

Application Server は、XML デジタル署名を使ってメッセージの「コンテンツ」に認証 ID をバインドします。クライアントはデジタル署名を使用して、呼び出し元 ID を確立します。この方法は、トランスポートレイヤーセキュリティーが使用されている場合に、基本認証または SSL クライアント証明書認証が同じ目的で使用される方法に類似しています。デジタル署名は、メッセージコンテンツのソースを認証するためにメッセージ受信者によって検証されます (このソースはメッセージ送信者と異なる可能性がある。)

デジタル署名を使用する場合、有効なキーストアおよびトラストストアファイルを Application Server 上に設定する必要があります。このトピックの詳細については、106 ページの「証明書ファイルについて」を参照してください。

暗号化について

暗号化の目的は、対象読者だけが理解できるようにデータを変更することです。これは、元のコンテンツを暗号化された要素に置き換えることにより行われます。公開鍵暗号方式に関して言えば、暗号化はメッセージを読み取ることができる関係者の ID を確立するために使用されます。

暗号化を使用する場合は、暗号化をサポートする JCE プロバイダがインストールされている必要があります。このトピックの詳細については、132 ページの「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ーまたはメッセージのコンテンツを定義したエンティティーの ID がメッセージ内で確立され、その ID をメッセー

ジ受信者が認証できる、という要件を表します。受信者認証ポリシーは、メッセージを受信可能なエンティティーの ID をメッセージ送信者が確立できるようにメッセージが送信される、という要件を表します。プロバイダは、特定のメッセージセキュリティーメカニズムを適用することで、SOAP Web サービスメッセージにおけるメッセージ保護ポリシーを実現します。

要求と応答に対するメッセージ保護ポリシーが定義されるのは、特定のプロバイダがコンテナ内に設定される時です。また、アプリケーションまたはアプリケーションクライアントの Sun 固有の配備記述子内で、アプリケーション固有のメッセージ保護ポリシー (Web サービスのポートまたは操作の粒度でのポリシー) を設定することも可能です。いずれにせよ、メッセージ保護ポリシーを定義する場合、クライアントの要求と応答に対するメッセージ保護ポリシーは、サーバーのそれと一致する (等しい) 必要があります。アプリケーション固有のメッセージ保護ポリシーの定義方法の詳細については、『Developers' Guide』の「Securing Applications」の章を参照してください。

メッセージセキュリティー用語の解説

次に、このマニュアルで使用する用語について説明します。これらの概念については、130 ページの「メッセージセキュリティーのための Application Server の設定」でも説明しています。

- 認証レイヤー

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

- 認証プロバイダ

  Application Server のこのリリースでは、Application Server は、「認証プロバイダ」を呼び出して SOAP メッセージレイヤーセキュリティーを処理します。

  - 「クライアント側プロバイダ」は、署名またはユーザー名/パスワードを使って要求メッセージのソース ID を確立したり、対象の受信者だけがメッセージを参照できるように暗号化を使って要求メッセージを保護したりします。また、クライアント側プロバイダは、受信した応答を正常に復号化することで、その許可された受信者としてコンテナを確立したり、応答内のパスワードまたは署名を検証してその応答に関連付けられたソース ID を認証したりもします。Application Server 内に設定されているクライアント側プロバイダを使えば、ほかのサービスのクライアントとして機能するサーバー側コンポーネント (サーブレットと EJB コンポーネント) によって送信される要求メッセージと受信される応答メッセージを保護することができます。
  - 「サーバー側プロバイダ」は、受信した要求を正常に復号化することで、その許可された受信者としてコンテナを確立したり、要求内のパスワードまたは署名を検証してその要求に関連付けられたソース ID を認証したりします。また、サーバー側プロバイダは、署名またはユーザー名/パスワードを使って応

答メッセージのソース ID を確立したり、対象の受信者だけがメッセージを参照できるように暗号化を使って要求メッセージを保護したりもします。「サーバー側プロバイダ」を呼び出すのはサーバー側コンテナだけです。

- デフォルトサーバープロバイダ

  「デフォルトサーバープロバイダ」は、特定のサーバープロバイダがバインドされていない任意のアプリケーションに対して呼び出されるサーバープロバイダを識別するために使用されます。「デフォルトサーバープロバイダ」は「デフォルトプロバイダ」とも呼ばれます。

- デフォルトクライアントプロバイダ

  「デフォルトクライアントプロバイダ」は、特定のクライアントプロバイダがバインドされていない任意のアプリケーションに対して呼び出されるクライアントプロバイダを識別するために使用されます。

- 要求ポリシー

  「要求ポリシー」は、認証プロバイダが実行する要求処理に関連付けられた認証ポリシー要件を定義します。ポリシーはメッセージ送信者の順序で送信されます。この順序には、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ー保護

Application Server 上に配備された Web サービスをセキュリティー保護するには、アプリケーションの配備先コンテナ、またはそのアプリケーションがサービスを提供する Web サービスエンドポイントのいずれかに対し、SOAP レイヤーメッセージセキュリティープロバイダとメッセージ保護ポリシーをバインドします。Application Server のクライアント側コンテナで SOAP レイヤーメッセージセキュリティー機能を設定するには、クライアントコンテナ、またはクライアントアプリケーションによって宣言されたポータブルサービス参照のいずれかに対し、SOAP レイヤーメッセージセキュリティープロバイダとメッセージ保護ポリシーをバインドします。

Application Server のインストール時に、SOAP レイヤーメッセージセキュリティープロバイダが Application Server のクライアント側コンテナとサーバー側コンテナ内に設定され、コンテナまたはコンテナ内に配備された個々のアプリケーションまたはクライアントからバインドして利用できるようになります。インストール中、プロバイダにはある単純なメッセージ保護ポリシーが設定されます。このポリシーをコンテナまたはコンテナ内のアプリケーションまたはクライアントにバインドした場

合、すべての要求メッセージと応答メッセージに含まれるコンテンツのソースが、XML デジタル署名によって認証されるようになります。

Application Server の管理インタフェースを使えば、既存のプロバイダをバインドして Application Server のサーバー側コンテナから利用できるようにしたり、プロバイダが適用するメッセージ保護ポリシーを変更したり、別のメッセージ保護ポリシーを備えた新しいプロバイダ設定を作成したりできます。アプリケーションクライアントコンテナの SOAP メッセージレイヤーセキュリティー設定でも、これと同様の管理操作を実行できます。それらについては、136 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」で定義しています。

Application Server では、メッセージレイヤーセキュリティーはデフォルトで無効になっています。Application Server のメッセージレイヤーセキュリティーを設定するには、130 ページの「メッセージセキュリティーのための Application Server の設定」に要約されている手順に従ってください。Application Server 上に配備されたすべての Web サービスアプリケーションを Web サービスセキュリティーで保護するには、135 ページの「メッセージセキュリティーのためのプロバイダの有効化」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ーが適用されるようになります。

アプリケーション固有の Web サービスセキュリティーの設定

アプリケーション固有の Web サービスセキュリティー機能をアプリケーションアセンブリで設定するには、アプリケーションの Sun 固有の配備記述子内で message-security-binding 要素を定義します。これらの message-security-binding 要素は、特定のプロバイダまたはメッセージ保護ポリシーを Web サービスエンドポイントまたはサービス参照に関連付けるために使用されます。また、この要素を修飾することで、それらのプロバイダやポリシーが対応するエンドポイントまたは参照サービスの特定のポートやメソッドに適用されるようにすることも可能です。

アプリケーション固有のメッセージ保護ポリシーの定義方法の詳細については、『Developers' Guide』の「Securing Applications」の章を参照してください。121 ページの「詳細情報」に、この章へのリンクがあります。

サンプルアプリケーションのセキュリティー保護

Application Server には、xms という名前のサンプルアプリケーションが付属しています。xms アプリケーションは、J2EE EJB エンドポイントと Java サーブレットエンドポイントの両方を使って実装された、単純な Web サービスです。両エンドポイントは同一のサービスエンドポイントインタフェースを共有しています。このサービスエンドポイントインタフェースには、単一の操作 sayHello が定義されています。この操作は、文字列引数を 1 つ受け取り、その呼び出し引数の前に Hello が付加された String を返します。

xms サンプルアプリケーションは、Application Server の WS-Security 機能を使って既存の Web サービスアプリケーションをセキュリティー保護する方法を示す目的で提供されています。サンプルに付属する手順では、Application Server の WS-Security 機能を有効にして xms アプリケーションを保護する方法が説明されています。また、このサンプルは、WS-Security 機能をアプリケーションに直接バインドする方法 (129 ページの「アプリケーション固有の Web サービスセキュリティーの設定」を参照) も示しています。

xms サンプルアプリケーションは次のディレクトリにインストールされます。*install-dir*/samples/webservices/security/ejb/apps/xms/。

xms サンプルアプリケーションのコンパイル、パッケージ化、および実行に関する詳細については、『Developers' Guide』の「Securing Applications」の章を参照してください。

メッセージセキュリティーのための Application Server の設定

要求および応答ポリシー設定のアクション

次の表は、メッセージ保護ポリシーの設定と、その結果として WS-Security SOAP メッセージセキュリティープロバイダによって実行されるメッセージセキュリティー処理を示したものです。

表 10–1　メッセージ保護ポリシーと WS-Security SOAP メッセージセキュリティー処理との対応づけ

メッセージ保護ポリシー	結果として実行される WS-Security SOAP メッセージ保護処理
auth-source="sender"	メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に wsse:UsernameToken (パスワード付き) が格納されます。
auth-source="content"	SOAP メッセージ本体のコンテンツが署名されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内にメッセージ本体の署名が ds:Signature として格納されます。
auth-source="sender" auth-recipient="before-content" OR auth-recipient="after-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に wsse:UsernameToken (パスワード付き) と xenc:EncryptedKey が格納されます。xenc:EncryptedKey には、SOAP メッセージ本体の暗号化に使用した鍵が格納されます。この鍵は受信者の公開鍵で暗号化されます。
auth-source="content" auth-recipient="before-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。xenc:EncryptedData が署名されます。 メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に xenc:EncryptedKey と ds:Signature として格納されます。 xenc:EncryptedKey には、SOAP メッセージ本体の暗号化に使用した鍵が格納されます。この鍵は受信者の公開鍵で暗号化されます。
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" OR 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。

表 10–1 メッセージ保護ポリシーと WS-Security SOAP メッセージセキュリティー処理との対応づけ (続き)

メッセージ保護ポリシー	結果として実行される WS-Security SOAP メッセージ保護処理
ポリシーを何も指定しない。	モジュールはセキュリティー処理を一切行いません。

その他のセキュリティー機能を設定する

Application Server は、SOAP 処理レイヤー内に統合化されたメッセージセキュリティープロバイダを使用して、メッセージセキュリティーを実装します。メッセージセキュリティープロバイダは、Application Server のその他のセキュリティー機能に依存します。

1. バージョン 1.5.0 より前のバージョンの Java SDK を使用し、暗号化技術を使用する場合は、JCE プロバイダを設定します。
2. JCE プロバイダの設定については、132 ページの「JCE プロバイダの設定」を参照してください。
3. 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。
4. 必要に応じて証明書と非公開鍵を管理します。

次の手順

Application Server の機能の設定が完了し、メッセージセキュリティープロバイダがそれらの機能を使用できるようになると、Application Server とともにインストールされたプロバイダを有効にできます。その手順については、135 ページの「メッセージセキュリティーのためのプロバイダの有効化」を参照してください。

JCE プロバイダの設定

J2SE 1.4.x に付属している JCE (Java Cryptography Extension) プロバイダは、RSA 暗号化をサポートしていません。通常、WS-Security で定義されている XML 暗号化は RSA 暗号化に基づいているため、WS-Security を使って SOAP メッセージを暗号化するには、RSA 暗号化をサポートする JCE プロバイダをダウンロードおよびインストールする必要があります。

注 – RSA は RSA Data Security, Inc. が開発した公開鍵暗号化技術です。この略語は、この技術の開発者である Rivest、Shamir、および Adelman を表しています。

Java SDK バージョン 1.5 で Application Server を実行している場合は、JCE プロバイダは正しく設定されています。Java SDK バージョン 1.4.x で Application Server を実行している場合は、次のようにJCE プロバイダを JDK 環境の一部として静的に追加できます。

1. JCE プロバイダの JAR (Java ARchive) ファイルをダウンロードし、インストールします。

   次の URL で、RSA 暗号化をサポートする JCE プロバイダのリストが提供されています。`http://java.sun.com/products/jce/jce14_providers.html`。
2. JCE プロバイダの JAR ファイルを *java-home*`/jre/lib/ext/` にコピーします。
3. Application Server を停止します。

   Application Server を停止せずにこの手順の最後で再起動した場合、JCE プロバイダは Application Server に認識されません。
4. 任意のテキストエディタで *java-home*`/jre/lib/security/java.security` プロパティーファイルを編集します。このファイルに、前述の手順でダウンロードした JCE プロバイダを追加します。

   `java.security` ファイルに、このプロバイダを追加する詳細手順が含まれています。基本的には、類似したプロパティーを持つ場所に次の形式の行を追加する必要があります。

   ```
   security.provider.n=provider-class-name
   ```

   この例では、*n* は、Application Server がセキュリティープロバイダを評価する際に使用する優先順位を示します。ここで追加した JCE プロバイダには、*n* を 2 に設定します。

   たとえば、Legion of the Bouncy Castle JCE プロバイダをダウンロードした場合は、次のような行を追加します。

   ```
   security.provider.2=org.bouncycastle.jce.provider.
      BouncyCastleProvider
   ```

   Sun セキュリティープロバイダが、値 1 の最高の優先順位に設定されていることを確認してください。

   ```
   security.provider.1=sun.security.provider.Sun
   ```

   各レベルにセキュリティープロバイダがただ 1 つだけ設定されるように、ほかのセキュリティープロバイダのレベルを下位に調整します。

   次に示す例は、必要な JCE プロバイダを提供し、既存のプロバイダを正しい位置に保持する `java.security` ファイルのサンプルです。

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=org.bouncycastle.jce.provider.
   ```

```
    BouncyCastleProvider
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.rsajca.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=sun.security.jgss.SunProvider
```

5. ファイルを保存して終了します。
6. Application Server を再起動します。

メッセージセキュリティーの設定

メッセージセキュリティーを使用できるように Application Server を設定する手順のほとんどは、管理コンソールまたは `asadmin` コマンド行ツールを使用するか、あるいはシステムファイルを手動で編集することで実現できます。一般に、システムファイルの編集はお勧めできません。なぜなら、Application Server が適切に動作しなくなるような変更を間違って施してしまう可能性があるからです。したがって、できるだけ、管理コンソールによる Application Server の設定手順を最初に示し、その後に `asadmin` ツールコマンドによる手順を示しています。システムファイルを手動で編集する手順は、管理コンソールと `asadmin` に同等の方法が存在しない場合にだけ示しています。

メッセージレイヤーセキュリティーのサポートは、プラグイン可能な認証モジュールの形式で Application Server とそのクライアントコンテナに統合されています。Application Server では、メッセージレイヤーセキュリティーはデフォルトで無効になっています。次の各節では、メッセージセキュリティー設定とプロバイダを有効化、作成、編集、および削除する方法について、詳しく説明します。

- 135 ページの「メッセージセキュリティーのためのプロバイダの有効化」
- 135 ページの「メッセージセキュリティープロバイダを設定する」
- 136 ページの「メッセージセキュリティープロバイダの作成」
- 136 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」
- 137 ページの「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」
- 138 ページの「詳細情報」

ほとんどの場合、上記の管理操作を実行したあとで Application Server を再起動する必要があります。特に、操作実行時に Application Server 上にすでに配備されていたアプリケーションに管理上の変更を適用したい場合に Application Server の再起動が必要となります。

メッセージセキュリティーの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ーを有効にするには、サーバー側でデフォルトで使用されるプロバイダを指定する必要があります。メッセージセキュリティーのデフォルトプロバイダを有効にする場合、Application Server 上に配備された Web サービスクライアントが使用するプロバイダも有効にする必要があります。クライアントが使用するプロバイダを有効にする方法については、136 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」を参照してください。

配備済みエンドポイントからの Web サービス呼び出しに対するメッセージセキュリティーを有効にするには、デフォルトクライアントプロバイダを指定する必要があります。Application Server のデフォルトクライアントプロバイダを有効にした場合、Application Server 内に配備されたエンドポイントから呼び出されるすべてのサービスが、メッセージレイヤーセキュリティー用に正しく設定されていることを確認する必要があります。

コマンド行ユーティリティーを使用するには、次の手順に従います。

- 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ープロバイダを設定する

プロバイダの再設定は通常、そのメッセージ保護ポリシーを変更するために行われますが、プロバイダのタイプ、実装クラス、およびプロバイダ固有の設定プロパティーも変更可能です。

コマンド行ユーティリティーを使用して、応答ポリシーを設定する場合は、次のコマンドの `request` という単語を `response` に置き換えてください。

- 要求ポリシーをクライアントに追加して、認証元を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
```

```
provider-config.ClientProvider.request-policy.auth_source=
sender | content
```

- 要求ポリシーをサーバーに追加して、認証元を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ServerProvider.request-policy.auth_source=
sender | content
```

- 要求ポリシーをクライアントに追加して、認証受信者を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ClientProvider.request-policy.auth_recipient=
before-content | after-content
```

- 要求ポリシーをサーバーに追加して、認証受信者を設定します。

```
asadmin  set --user  admin-user --port admin-port
server-config.security-service.message-security-config.SOAP.
provider-config.ServerProvider.request-policy.auth_recipient=
before-content | after-content
```

メッセージセキュリティープロバイダの作成

管理コンソールを使用して既存のプロバイダを設定するには、「設定」ノード > 設定するインスタンス >「セキュリティー」ノード >「メッセージセキュリティー」ノード >「SOAP」ノード >「プロバイダ」タブの順に選択します。

メッセージセキュリティープロバイダの作成方法については、管理コンソールのオンラインヘルプを参照してください。

アプリケーションクライアントのメッセージセキュリティーを有効にする

クライアントプロバイダのメッセージ保護ポリシーは、通信相手となるサーバー側プロバイダのメッセージ保護ポリシーと等しくなるように設定する必要があります。Application Server のインストール時に設定された (しかしまだ有効化されていない) プロバイダでは、すでにそうなっています。

クライアントアプリケーションのメッセージセキュリティーを有効にするには、アプリケーションクライアントコンテナの Application Server 固有の設定を変更します。

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ーを実現するには、サーバーとクライアントの両方で要求ポリシーと応答ポリシーが有効化されている必要があります。クライアントおよびサーバーのポリシーを設定する場合は、クライアントポリシーがアプリケーションレベルのメッセージのバインドで要求および応答保護のサーバーポリシーと一致する必要があります。

アプリケーションクライアント設定の要求ポリシーを設定するには、136 ページの「アプリケーションクライアントのメッセージセキュリティーを有効にする」の説明に従って、アプリケーションクライアントコンテナの Application Server 固有の設定を変更します。アプリケーションクライアント設定ファイル内で `request-policy` 要素と `response-policy` 要素を次のように追加することで、要求ポリシーを設定します。

その他のコードは参照用に用意されています。実際のインストールでは、その他のコードが若干異なっている可能性があります。それらを変更しないでください。

```
<client-container>
  <target-server name="your-host" address="your-host"
      port="your-port"/>
  <log-service file="" level="WARNING"/>
  <message-security-config auth-layer="SOAP"
      default-client-provider="ClientProvider">
    <provider-config
        class-name="com.sun.enterprise.security.jauth.ClientAuthModule"
        provider-id="ClientProvider" provider-type="client">
      <request-policy auth-source="sender | content"
        auth-recipient="after-content | before-content"/>
      <response-policy auth-source="sender | content"
        auth-recipient="after-content | before-content"/>
       <property name="security.config"
           value="install-dir/lib/appclient/wss-client-config.xml"/>
    </provider-config>
  </message-security-config>
</client-container>
```

`auth-source` の有効な値には、`sender` と `content` があります。`auth-recipient` の有効な値には、`before-content` と `after-content` があります。これらの値をさまざまに組み合わせた結果を記述した表については、130 ページの「要求および応答ポリシー設定のアクション」を参照してください。

要求または応答ポリシーを指定しない場合は、この要素を空白のままにします。次に例を示します。

```
<response-policy/>
```

詳細情報

- Java 2 Standard Edition のセキュリティーの説明については、`http://java.sun.com/j2se/1.4.2/docs/guide/security/index.html` を参照してください。
- 『J2EE 1.4 Tutorial』の「Security」の章については、`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html` を参照してください。
- 『管理ガイド』の第 9 章。
- 『Developer's Guide』の「Securing Applications」の章。
- 『Oasis Web Services Security: SOAP Message Security (WS-Security)』仕様については、http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf を参照してください。
- 『OASIS Web Services Security Username Token Profile 1.0』については、http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf を参照してください。
- 『OASIS Web Services Security X.509 Certificate Token Profile 1.0』については、http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf を参照してください。
- XML-Signature Syntax and Processing ドキュメントについては、`http://www.w3.org/TR/xmldsig-core/` を参照してください。
- XML Encryption Syntax and Processing ドキュメントについては、`http://www.w3.org/TR/xmlenc-core/` を参照してください。

◆ ◆ ◆ 第 11 章

11 トランザクション

トランザクションを使用すると、1 つ以上のステップがそれ以上分割不可能な作業単位にまとめられるため、データの完全性と整合性が保証されます。この章の内容は次のとおりです。

トランザクションとは

トランザクションは、すべて正常に完了することが必要なアプリケーションで、周到に用意された一連のアクションです。正常に完了しない場合、各アクションで行われたすべての変更が取り消されます。たとえば、当座預金から普通預金に資金を移動するのは次の手順を実行するトランザクションになります。

1. 当座預金口座にその移動をカバーするだけの金額があるかどうかを確認します。
2. 当座預金に十分なお金が入っている場合は、当座預金の金額を借り方に記帳します。
3. その金額を普通預金口座の貸し方に記帳します。
4. その移動を当座預金口座ログに記録します。
5. その移動を普通預金口座ログに記録します。

これらの手順のいずれが失敗すると、先行する手順によって行われた変更がすべて取り消されます。当座預金口座と普通預金口座はこのトランザクションが始まる前と同じ状態になる必要があります。このイベントは「ロールバック」と呼ばれます。すべての手順が正常に完了すると、トランザクションは「コミット」状態になります。トランザクションはコミットかロールバックのどちらかで終了します。

J2EE テクノロジのトランザクション

J2EE テクノロジのトランザクション処理には、次の 5 つの関係要素が含まれます。

- トランザクションマネージャー
- Application Server
- リソースマネージャー (複数可)
- リソースアダプタ (複数可)
- ユーザーアプリケーション

これらの各エンティティーは、次に説明する API や機能を実装することにより、信頼性のあるトランザクション処理を実現しています。

- トランザクションマネージャーは、トランザクション境界、トランザクションリソース管理、同期化、およびトランザクションコンテキスト伝達のサポートに必要なサービスと管理機能を提供します。
- Application Server は、トランザクション状態管理を含むアプリケーションランタイム環境のサポートに必要なインフラストラクチャーを提供します。
- リソースマネージャーは、リソースアダプタを介して、リソースへのアプリケーションアクセスを提供します。リソースマネージャーは、特定のトランザクションリソースインタフェースを実装することで分散トランザクションに参加します。このインタフェースは、トランザクションマネージャーがトランザクションの関連付け、トランザクションの完了、および回復作業を伝達する際に使用されます。このようなリソースマネージャーの例としては、リレーショナルデータベースサーバーがあります。
- リソースアダプタはシステムレベルのソフトウェアライブラリで、リソースマネージャーへ接続するためにアプリケーションサーバーまたはクライアントが使用します。通常、リソースアダプタはリソースマネージャーに固有です。リソースアダプタはライブラリとして使用可能で、クライアントのアドレス空間内で使用されます。そのようなリソースアダプタの一例として、JDBC ドライバが挙げられます。
- アプリケーションサーバー環境で動作するように開発されたトランザクションユーザーアプリケーションは、JNDI を使用してトランザクションデータソースおよびトランザクションマネージャー (オプション) を検索します。アプリケーションは、エンタープライズ Bean 用の宣言的なトランザクション属性設定や、プログラムによる明示的なトランザクション境界を使用することがあります。

トランザクションの回復

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

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。

管理コンソールを使用して、トランザクションを回復するようにApplication Server を設定できます。この作業の詳細な手順については、管理コンソールのオンラインヘルプを参照してください。

トランザクションのタイムアウト値

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。この作業の詳細な手順については、管理コンソールのオンラインヘルプを参照してください。

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。

キーポイント間隔

キーポイント処理によって、トランザクションログファイルが圧縮されます。キーポイント間隔とは、ログに対して実行されるキーポイント処理の間のトランザクション数のことです。キーポイント処理によって、トランザクションログファイルのサイズを小さくすることができます。キーポイント間隔を大きくすると (例: 2048)、トランザクションログファイルが大きくなりますが、キーポイント処理が少なくなるのでパフォーマンスが向上する可能性があります。キーポイント間隔を小さくすると (例: 256)、ログファイルのサイズが小さくなりますが、キーポイント処理が多くなるので、パフォーマンスがわずかに低下します。

◆ ◆ ◆ 第 12 章

12 HTTP サービスの設定

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。 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。

次の内容について説明します。

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、IP アドレス 0.0.0.0 を指定すると、HTTP リスナーは設定されたすべての IP アドレスをマシンの特定のポートで待機できます。また、各リスナーに一意の IP アドレスを指定した上で、同一ポートを使用することもできます。

HTTP リスナーは IP アドレスとポート番号の組み合わせであるため、IP アドレスが同じでポート番号が異なる HTTP リスナーや (例: 1.1.1.1:8081 および 1.1.1.1:8082)、IP アドレスが異なっていてポート番号が同じ HTTP リスナー (例: 1.1.1.1:8081 および 1.2.3.4:8081。ただし、マシンがこれら両方のアドレスに応答するように設定されている場合) を複数使用することができます。

ただし、HTTP リスナーに単一のポート上ですべての IP アドレスを待機する 0.0.0.0 を使用する場合は、この同じポート上に、特定の IP アドレスを待機する HTTP リスナーを作成できません。たとえば、HTTP リスナーが 0.0.0.0:8080 (ポート 8080 のすべての IP アドレス) を使用する場合、別の HTTP リスナーが 1.2.3.4:8080 を使用することはできません。

通常、Application Server が稼動するシステムでアクセスできる IP アドレスは 1 つだけであるため、HTTP リスナーは、ポートが異なる 0.0.0.0 IP アドレスを通常使用し、役割ごとに異なるポート番号を使用します。システムが複数の IP アドレスにアクセスできる場合は、各アドレスを異なる役割に使用できます。

デフォルトでは、Application Server を起動すると、次の HTTP リスナーが準備されます。

- `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` ではセキュリティーが無効になり、`http-listener-2` ではセキュリティーが有効になります。
- 仮想サーバー `__asadmin` に関連付けられた HTTP リスナー `admin-listener`。このリスナーでは、セキュリティーが有効になります。

これらのリスナーはすべて、Application Server のインストールの間に HTTP サーバーポート番号として指定された IP アドレス 0.0.0.0 とポート番号を使用します。Application Server がポート番号のデフォルト値をそのまま使用した場合、`http-listener-1` はポート 8080、`http-listener-2` はポート 8181、`admin-listener` はポート 4849 を使用します。

各 HTTP リスナーはデフォルトの仮想サーバーを持ちます。デフォルトの仮想サーバーは、HTTP リスナーがその HTTP リスナーに関連付けられたどの仮想サーバーともホストコンポーネントが一致しないすべての要求 URL をルーティングする宛先のサーバーです。仮想サーバーと HTTP リスナーの関連付けは、仮想サーバーの `http-listeners` 属性に HTTP リスナーを指定することで行われます。

さらに、HTTP リスナー内のアクセプタスレッドの数を指定します。アクセプタスレッドは、接続を待機するスレッドです。アクセプタスレッドによって受け付けられ、接続キューと呼ばれるキューに入れられた接続は、ワークスレッドによって取り出されます。新しい要求が着信したときにいつでも対応できるように、常に十分な数のアクセプタスレッドを設定しておきますが、システムに負荷がかかり過ぎない数に抑える必要もあります。接続キューには、アクセプタスレッドによって受け付けられた新しい接続と、キープアライブ接続管理サブシステムによって管理される持続接続の両方が格納されます。

一連の要求処理スレッドが、接続キューから受信 HTTP 要求を取り出し、それらの要求を処理します。これらのスレッドは、HTTP ヘッダーを解析し、適切な仮想サーバーを選択し、要求処理エンジンを実行して要求を処理します。処理すべき要求がなくなったあと、その接続が `HTTP/1.1` を使用するか `Connection: keep-alive`

ヘッダーを送信することで持続可能になっていた場合、要求処理スレッドは、その接続がアイドル状態にあると判断し、その接続をキープアライブ接続管理サブシステムに渡します。

キープアライブサブシステムは、そうしたアイドル状態の接続を定期的にポーリングし、活動中の接続が見つかるとそれらを接続キュー内に格納し、さらに処理できるようにします。要求処理スレッドは、そのキューから再び接続を取り出し、その要求を処理します。キープアライブサブシステムはマルチスレッド化されています。なぜなら、このサブシステム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、元のセキュリティー保護されていないポートを SSL 有効ポートに置き換えた新しい URL がロケーション応答ヘッダーに指定された応答を返します。クライアントは、この新しい URL に接続します。

また、HTTP リスナーのセキュリティーを有効にするかどうか、あるいは、どのセキュリティーの種類を使用するか (例: SSL プロトコルや暗号化方式の種類) も指定します。

Application Server に配備された Web アプリケーションにアクセスするには、Web アプリケーション用に指定したコンテキストルートとともに、`http://localhost:8080/` (または、セキュリティー保護されたアプリケーションでは `https://localhost:8181/`) という URL を使用します。管理コンソールにアクセスするには、URL `https://localhost:4849/` または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。

◆ ◆ ◆ 第 13 章

13 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。

CORBA

Application Server は、相互運用性を確実にする一連の標準的なプロトコルおよび形式をサポートします。これらのプロトコルの中には、CORBA で定義されているものがあります。

CORBA (Common Object Request Broker Architecture) モデルのベースになっているのは、明確に定義されたインタフェースを介して分散型のオブジェクトやサーバーにサービスを要求するクライアントです。こうしたクライアントは、リモートメソッド要求の形式でオブジェクトに対して要求を発行します。リモートメソッド要求では、実行する必要のある操作に関する情報が伝送されます。この情報には、サービスプロバイダのオブジェクト名 (オブジェクト参照) と、存在する場合は、起動メソッドのパラメータが含まれます。CORBA は、オブジェクトの登録、オブジェクトの配置、オブジェクトのアクティブ化、要求の多重分離、エラー処理、整列化、操作のディスパッチをはじめとするネットワークプログラミングのタスクを自動的に処理します。

ORB とは

ORB (Object Request Broker) は、CORBA の中枢となるコンポーネントです。ORB は、オブジェクトの特定と検索、接続管理、およびデータと要求の配信に必要なインフラストラクチャーを提供します。

個々の CORBA オブジェクトが相互に対話することはありません。その代わりに、リモートスタブを介して、ローカルマシンで実行されている ORB に要求を送ります。次に、ローカルの ORB が、IIOP (Internet Inter-Orb Protocol) を使ってほかのマシン上の ORB へ要求を転送します。リモート ORB は、適切なオブジェクトを検出し、要求を処理して、結果を返します。

アプリケーションやオブジェクトでは、RMI-IIOP により、IIOP を RMI (Remote Method Invocation) として使用することが可能になっています。エンタープライズ Bean (EJB モジュール) のリモートクライアントは、RMI-IIOP を介して Application Server と通信します。

IIOP リスナー

IIOP リスナーは、Enterprise JavaBeans のリモートクライアントおよびほかの CORBA ベースのクライアントから受信する接続を受け付ける待機ソケットです。Application Server では、複数の IIOP リスナーを設定できます。各リスナーに対して、ポート番号、ネットワークアドレス、およびオプションでセキュリティー属性を指定してください。

ORB の操作

IIOP リスナーを作成するには、管理コンソールで「設定」>「ORB」>「IIOP リスナー」の順に選択し、「新規」をクリックします。または、次の asadmin コマンドを使用して IIOP リスナーを作成します。`create-iiop-listener`(1) および `create-ssl`(1)。

IIOP リスナーを編集するには、管理コンソールで「設定」>「ORB」>「IIOP リスナー」の順に選択し、変更対象のリスナーを選択します。設定を変更します。ポート番号を変更した場合は、サーバーを再起動します。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。

IIOP リスナーを削除するには、管理コンソールで「設定」>「ORB」>「IIOP リスナー」の順に選択し、削除対象のリスナーを選択します。または、`delete-iiop-listener`(1) コマンドを使用します。

`ORBCommunicationsRetryTimeout` プロパティーは、ORB クライアントが到達不可能な ORB バックエンドへの接続の確立を試行する秒数を指定します。デフォルト値は `60`

秒です。このデフォルト設定では、ORB バックエンドが到達不可能な場合、ログに大量の CORBA 例外が記録されたり、ネットワーク使用率が高くなったりすることがあります。

そのような場合は、`ORBCommunicationsRetryTimeout` を低い値に設定してください。

サードパーティー製の ORB

Sun Java System Application Server は、サードパーティー製の ORB ソフトウェアと併用できます。そのようなサードパーティー製 ORB をサポートするには、サーバー側の設定を変更する必要があります。

サードパーティー製 ORB のサポートを実装するには、`domain.xml` ファイルと `server.policy` ファイルを編集する必要があります。サードパーティー製 ORB のサンプルを設定する方法の詳細は、「Configuring Sun Java System Application Server for Third-Party ORBs」(`http://developers.sun.com/prodtech/appserver/reference/techart/orb.html`) を参照してください。

◆ ◆ ◆ 第 14 章

14 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。パフォーマンスの向上に役立つように、Application Server は 1 つまたは複数のスレッドプールを維持します。特定のスレッドプールを、コネクタモジュールと ORB に割り当てることができます。

1 つのスレッドプールで、複数のコネクタモジュールおよびエンタープライズ Beans を処理できます。要求スレッドは、アプリケーションコンポーネントへのユーザーの要求を処理します。サーバーは要求を受け取ると、スレッドプールから使用可能なスレッドにその要求を割り当てます。スレッドはクライアントの要求を実行し、結果を返します。たとえば、現在ビジー状態のシステムリソースが必要な場合、スレッドはリソースが解放されるのを待ってから、リソースの使用を要求に許可します。

アプリケーションからの要求用に確保するスレッドの最小数と最大数を指定できます。スレッドプールはこれら 2 つの値の間で動的に調整されます。サーバーは、指定された最小スレッドプールサイズに従って、アプリケーション要求用に確保するスレッドを割り当てます。その数は、指定された最大スレッドプールサイズまで増加できます。

プロセスで使用可能なスレッドの数を増やすと、プロセスが同時に応答できるアプリケーション要求数が多くなります。

1 つのリソースアダプタやアプリケーションが Application Server のすべてのスレッドを占有している場合、Application Server のスレッドを複数のスレッドプールに分割することで、スレッド不足を防止してください。

この章の内容は、次のとおりです。

スレッドプールの構成

管理コンソール を使用してスレッドプールを作成するには、「設定」>「スレッドプール」>「現在のプール」>「新規」の順に選択します。

- 「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。
- 「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。
- 「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。
- 「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。
- 「作業キューの数」フィールドに、このスレッドプールが処理する作業キューの合計数を入力します。
- Application Server を再起動します。

スレッドプールの作成に関する詳細については、管理コンソール で「ヘルプ」をクリックしてください。

スレッドプールをコマンド行から作成するには、`asadmin` コマンドの `create-threadpool`(1) を使用します。

管理コンソール を使用してスレッドプールの設定を編集するには、「設定」>「スレッドプール」>「現在のプール」の順に選択し、設定するプールを選択します。選択したスレッドプールの値を変更して保存し、Application Server を再起動します。

スレッドプールの編集の詳細については、管理コンソール で「ヘルプ」をクリックしてください。

管理コンソール を使用してスレッドプールを削除するには、「設定」>「スレッドプール」>「現在のプール」の順に選択します。削除するスレッドプール名を確認し、「削除」をクリックします。

Application Server を再起動します。

スレッドプールをコマンド行から削除するには、`asadmin` コマンドの `delete-threadpool`(1) を使用します。

◆ ◆ ◆ 第 15 章

15 ロギングの設定

この章では、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。この章は、次の節で構成されます。

ログレコード

Application Server は、JSR 047 に指定されている「Java 2 platform Logging API」を使用します。Application Server のログメッセージはサーバーログ内に記録されます。このサーバーログの場所は通常、*domain-dir*/logs/server.log です。

domain-dir/logs ディレクトリには、サーバーログのほかに別の 2 種類のログも格納されます。access サブディレクトリには HTTP サービスアクセスログ、tx サブディレクトリにはトランザクションサービスログがあります。これらのログについては、管理コンソール のオンラインヘルプ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-e8.1|javax.enterprise.
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取ることができます。`SEVERE`、`WARNING`、`INFO`、`CONFIG`、`FINE`、`FINER`、および `FINEST` 。
- *ProductName-Version* は `sun-appserver-ee8.1` です。
- *LoggerName* はログモジュールのソースを識別する階層ロガーの名前空間で、この場合は `javax.enterprise.system.core` です。
- *Key Value Pairs* はキー名と値で、通常は `_ThreadID=14;` のようなスレッド ID です。
- *Message* は、ログメッセージのテキストです。Application Server のすべての `SEVERE` メッセージと `WARNING` メッセージ、および多くの `INFO` メッセージは、モジュールコードと数値から構成されるメッセージ ID (この場合は `CORE5004`) で始まります。

このログレコード形式は、将来のリリースでは変更または拡張される可能性があります。

カスタムログレベルを設定する

この節では、`java.util.logging` パッケージを使用して Application Server のロギングサブシステムにアクセスするアプリケーションのカスタムロギングレベルを設定する方法について説明します。

`java.util.logging` パッケージは、ロガーインスタンスを作成できる階層型の名前空間を提供します。特定のロギングレコードが Application Server インスタンスのログファイルに出力されるかどうかは、ログレコードのログレベルと指定したログレベルによって異なります。

Application Server のロガー設定では、20 を超えるロギングモジュールが提供され、Application Server 独自の内部ロギングを詳細に制御できます。また、モジュール名とそのモジュールで使用するロギングレベルを指定することで、カスタムログモジュールを追加作成するオプションもあります。

ここで重要なのは、ロガーは静的な名前であり、継承されないということです。そのため、カスタムロガーが `com.someorg.app` という名前で設定された場合、アプリケーションがロガー `com.someorg.app.submodule` を検索しても、`com.someorg.app` から

の設定を継承するロガーは提供されません。代わりに com.someorg.app.submodule には、INFO レベルまたはそれ以上のレベルでロギングするように設定されたデフォルトのロガーがあります。

アプリケーションでロガーの継承を使用する必要がある場合は、Application Server を実行するために使用している Java ランタイムの logging.properties ファイルを編集して設定することができます。たとえば、次のエントリを logging.properties ファイルに追加すると、FINE レベルを継承する com.someorg.app.submodule が作成されます。

```
com.someorg.app.level = FINE
```

Java ロギング API の詳細は、http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/package-summary.html にある Java ドキュメントや、その他の java.util.logging クラスのドキュメントを参照してください。

ロガー名前空間の階層

Application Server では、モジュールごとにロガーが用意されています。次の表では、管理コンソール の「ログレベル」ページに表示されるとおりに、アルファベット順でモジュールの名前と各ロガーの名前空間を示します。表内の最後の 3 つのモジュールは、「ログレベル」ページには表示されません。

表 15–1　Application Server ロガー名前空間

モジュール名	名前空間
管理	javax.enterprise.system.tools.admin
クラスローダー	javax.enterprise.system.core.classloading
CMP	javax.enterprise.system.container.cmp
構成	javax.enterprise.system.core.config
コネクタ	javax.enterprise.resource.resourceadapter
CORBA	javax.enterprise.resource.corba
導入	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

表 15–1 Application Server ロガー名前空間 (続き)

モジュール名	名前空間
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
Root	javax.enterprise
SAAJ	javax.enterprise.resource.webservices.saaj
セキュリティー	javax.enterprise.system.core.security
サーバー	javax.enterprise.system
同期 (Enterprise Edition に限る)	javax.ee.enterprise.system.tools.synchronization
ユーティリティー	javax.enterprise.system.util
ベリファイア	javax.enterprise.system.tools.verifier
Web コンテナ	javax.enterprise.system.container.web
コアシステムサポート	javax.enterprise.system.core
システム出力 (System.out.println)	javax.enterprise.system.stream.out
システムエラー (System.err.println)	javax.enterprise.system.stream.err

◆ ◆ ◆ 第 16 章

コンポーネントとサービスの監視

監視機能を使用して Application Server のサーバーインスタンスに配備されている各種コンポーネントおよびサービスの実行時状態を把握します。実行時コンポーネントとプロセスに関する情報を使用して、チューニングに関わるパフォーマンスボトルネックを識別し、処理能力を計画し、障害を見積もり、障害の場合の原因を分析して、期待通りの機能性を確保できます。

監視をオンにすると、オーバーヘッドの増大によりパフォーマンスが低下します。

この章の内容は次のとおりです。

監視の概要

Application Server を監視するには、次の手順を実行します。

1. 管理コンソールまたは `asadmin` ツールを使用して、特定のサービスおよびコンポーネントの監視を有効にします。

   この手順の詳細については、185 ページの「監視の有効化と無効化」を参照してください。

2. 管理コンソールまたは `asadmin` ツールを使用して、特定のサービスおよびコンポーネントの監視データを表示します。

   この手順の詳細については、186 ページの「監視データの表示」を参照してください。

監視可能なオブジェクトのツリー構造について

Application Server は、ツリー構造を使って監視可能なオブジェクトを追跡します。監視オブジェクトのツリーは動的であり、インスタンス内におけるコンポーネントの追加、更新、削除に応じて変更されます。ツリー内のルートオブジェクトは、server などのサーバーインスタンス名です。Platform Edition では、1 つのサーバーインスタンスしか使用できません。

次のコマンドを実行すると、ツリーのトップレベルが表示されます。

```
asadmin> list --user adminuser --monitor server
server.applications
server.http-service
server.connector-service
server.jms-service
server.jvm
server.orb
server.resources
server.thread-pools
```

次の各節では、これらのサブツリーについて説明します。

- 160 ページの「アプリケーションのツリー」
- 161 ページの「HTTP サービスのツリー」
- 162 ページの「リソースのツリー」
- 162 ページの「コネクタサービスのツリー」
- 162 ページの「JMS サービスのツリー」
- 163 ページの「ORB のツリー」
- 163 ページの「スレッドプールのツリー」

アプリケーションのツリー

次の図に、エンタープライズアプリケーションの各種コンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。詳細については、164 ページの「EJB コンテナの統計」を参照してください。

例 16–1 アプリケーションノードのツリー構造

```
applications
   |--- application1
   |    |--- ejb-module-1
   |    |        |--- ejb1 *
   |    |                 |--- cache (エンティティー/sfsb) *
   |    |                 |--- pool (ステートレスセッション/メッセージ駆動型/エンティティー Bean 用) *
   |    |                 |--- methods
   |    |                      |---method1 *
```

例 16–1 アプリケーションノードのツリー構造 (続き)

```
|    |                        |---method2 *
|    |                   |--- stateful-session-store (sfsb)*
|    |                   |--- timers (ステートレスセッション/エンティティー/mdb) *
|    |--- web-module-1
|    |        |--- virtual-server-1 *
|    |                        |---servlet1  *
|    |                        |---servlet2  *
|--- standalone-web-module-1
|    |        |----- virtual-server-2 *
|    |                        |---servlet3 *
|    |                        |---servlet4 *
|    |        |----- virtual-server-3 *
|    |                        |---servlet3 *(ほかの仮想サーバーと同一のサーブレット)
|    |                        |---servlet5 *
|--- standalone-ejb-module-1
|    |        |--- ejb2 *
|    |                   |--- cache (エンティティー/sfsb) *
|    |                   |--- pool (ステートレスセッション/メッセージ駆動型/エンティティー Bean 用) *
|    |                   |--- methods
|    |                        |--- method1 *
|    |                        |--- method2 *
|--- application2
```

HTTP サービスのツリー

HTTP サービスのノードを、次の図に示します。監視情報が利用可能なノードには、アスタリスク (*) を付けています。169 ページの「HTTP サービスの統計」を参照してください。

例 16–2 HTTP サービスの図 (Platform Edition 版)

```
http-service
         |--- virtual-server-1
              |--- http-listener-1 *
              |--- http-listener-2 *
         |--- virtual-server-2
              |--- http-listener-1 *
              |--- http-listener-2 *
```

例 16–3 HTTP サービスの図 (Enterprise Edition 版)

```
http-service *
             |---connection-queue *
             |---dns *
             |---file-cache *
```

例 16–3 HTTP サービスの図 (Enterprise Edition 版) (続き)

```
            |---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。170 ページの「JDBC 接続プールの統計」を参照してください。

例 16–4 リソースの図

```
resources
        |---connection-pool1(connector-connection-pool、jdbc のいずれか)*
        |---connection-pool2(connector-connection-pool、jdbc のいずれか)*
```

コネクタサービスのツリー

コネクタサービスノードには、コネクタ接続プールなどのプールの監視可能な属性が格納されます。次の図に、各種コネクタサービスコンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。171 ページの「JMS サービスおよびコネクタサービスの統計」を参照してください。

例 16–5 コネクタサービスの図

```
connector-service
        |--- resource-adapter-1
        |        |-- connection-pools
        |        |        |-- pool-1 (このプールのすべてのプール状態)
        |        |-- work-management (このリソースアダプタのすべての作業管理状態)
```

JMS サービスのツリー

JMS サービスノードには、コネクタ接続プールなどのプールの監視可能な属性が格納されます。次の図に、各種 JMS サービスコンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。

例 16–6　JMS サービスの図

```
jms-service
        |-- connection-factories (リソースアダプタの世界では AKA 接続プールと呼ばれる)
        |        |-- connection-factory-1 (この接続ファクトリのすべての接続ファクトリ状態)
        |-- work-management (この MQ リソースアダプタのすべての作業管理状態)
```

ORB のツリー

ORB ノードには、接続マネージャーの監視可能な属性が格納されます。次の図に、各種 ORB コンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。173 ページの「ORB の接続マネージャーの統計」を参照してください。

例 16–7　ORB の図

```
orb
        |--- connection-managers
        |        |--- connection-manager-1 *
        |        |--- connection-manager-1 *
```

スレッドプールのツリー

スレッドプールノードには、接続マネージャーの監視可能な属性が格納されます。次の図に、各種 ORB コンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。173 ページの「スレッドプールの統計」を参照してください。

例 16–8　スレッドプールの図

```
thread-pools
        |    |--- thread-pool-1 *
        |    |--- thread-pool-2 *
```

監視対象のコンポーネントとサービスの統計

この節では、利用可能な監視統計について説明します。

EJB コンテナの統計

EJB 統計を次の表に示します。

表 16–1　EJB 統計

Attribute Name（属性名）	データ型	説明
createcount	CountStatistic	特定の EJB に対する create メソッドの呼び出し回数。
removecount	CountStatistic	特定の EJB に対する remove メソッドの呼び出し回数。
pooledcount	RangeStatistic	プールされた状態にあるエンティティー Bean の数。
readycount	RangeStatistic	実行可能状態にあるエンティティー Bean の数。
messagecount	CountStatistic	特定のメッセージ駆動型 Bean に対して受信されたメッセージの数。
methodreadycount	RangeStatistic	MethodReady 状態にあるステートフルまたはステートレスセッション Beans の数。
passivecount	RangeStatistic	Passive 状態にあるステートフルセッション Beans の数。

EJB メソッド呼び出しに関して利用可能な統計を、次の表に示します。

表 16–2　EJB メソッドの統計

Attribute Name（属性名）	データ型	説明
methodstatistic	TimeStatistic	特定の操作の呼び出し回数。その呼び出しにかかった合計時間など。

表 16–2 EJB メソッドの統計 (続き)

Attribute Name（属性名）	データ型	説明
totalnumerrors	CountStatistic	メソッド実行時に例外が発生した回数。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。
totalnumsuccess	CountStatistic	メソッドが正常に実行された回数。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。
executiontime	CountStatistic	成功または失敗した最後の操作実行時にメソッド実行に費やされた時間 (ミリ秒)。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。

EJB セッションストアに関する統計を、次の表に示します。

表 16–3 EJB セッションストアの統計

Attribute Name（属性名）	データ型	説明
currentSize	RangeStatistic	現在ストア内に存在している、非活性化またはチェックポイント化されたセッションの数。
activationCount	CountStatistic	ストアから活性化されたセッションの数。
activationSuccessCount	CountStatistic	ストアからの活性化に成功したセッションの数
activationErrorCount	CountStatistic	成功または失敗した最後の操作実行時にメソッド実行に費やされた時間 (ミリ秒)。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。

表 16–3 EJB セッションストアの統計 (続き)

Attribute Name（属性名）	データ型	説明
passivationCount	CountStatistic	このストアを使って非活性化されたセッションの数。
passivationSuccessCount	CountStatistic	このストアを使って正常に非活性化されたセッションの数。
passivationErrorCount	CountStatistic	このストアを使って非活性化できなかったセッションの数。
expiredSessionCount	CountStatistic	期限切れによりこのストアから削除されたセッションの数。
passivatedBeanSize	CountStatistic	このストアによって非活性化されたバイト数の合計 (合計、最小、最大を含む)。
passivationTime	CountStatistic	Beans のストアへの非活性化に要した時間 (合計、最小、最大を含む)。
checkpointCount (EE のみ)	CountStatistic	このストアを使ってチェックポイント化されたセッションの数。
checkpointSuccessCount (EE のみ)	CountStatistic	正常にチェックポイント化されたセッションの数。
checkpointErrorCount (EE のみ)	CountStatistic	チェックポイント化できなかったセッションの数。
checkpointedBeanSize (EE のみ)	ValueStatistic	ストアによってチェックポイント化された Beans の合計数。
checkpointTime (EE のみ)	TimeStatistic	Beans のストアへのチェックポイント化に要した時間。

EJB プールに関して利用可能な統計を、次の表に示します。

表 16–4 EJB プールの統計

Attribute Name（属性名）	データ型	説明
numbeansinpool	BoundedRangeStatistic	関連付けられたプール内の EJB 数。これにより、プールがどのように変化しているかがわかります。
numthreadswaiting	BoundedRangeStatistic	未使用 Beans を取得するために待機しているスレッドの数。これは、要求の輻輳の可能性を示します。

表 16–4 EJB プールの統計 (続き)

Attribute Name（属性名）	データ型	説明
totalbeanscreated	CountStatistic	関連付けられたプール内でデータ収集開始後に作成された Beans の数。
totalbeansdestroyed	CountStatistic	関連付けられたプール内でデータ収集開始後に破棄された Beans の数。
jmsmaxmessagesload	CountStatistic	メッセージ駆動型 Bean のサービスを提供するために JMS セッション内に一度にロード可能なメッセージの最大数。デフォルトは 1。メッセージ駆動型 Beans 用のプールにのみ適用されます。

EJB キャッシュに関して利用可能な統計を、次の表に示します。

表 16–5 EJB キャッシュの統計

Attribute Name（属性名）	データ型	説明
cachemisses	BoundedRangeStatistic	ユーザー要求に対する Bean がキャッシュ内で見つからなかった回数。
cachehits	BoundedRangeStatistic	ユーザー要求に対するエントリがキャッシュ内で見つかった回数。
numbeansincache	BoundedRangeStatistic	キャッシュ内の Beans 数。これは現在のキャッシュサイズです。
numpassivations	CountStatistic	非活性化された Bean の数。ステートフルセッション Beans にのみ適用されます。
numpassivationerrors	CountStatistic	非活性化中に発生したエラーの数。ステートフルセッション Beans にのみ適用されます。
numexpiredsessionsremoved	CountStatistic	クリーンアップスレッドによって削除された期限切れセッションの数。ステートフルセッション Beans にのみ適用されます。
numpassivationsuccess	CountStatistic	非活性化が正常に終了した回数。ステートフルセッション Beans にのみ適用されます。

タイマーに関して利用可能な統計を、次の表に示します。

表 16–6 タイマーの統計

Statistic	データ型	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、160 ページの「アプリケーションのツリー」に示したオブジェクトツリー内に含まれます。Web コンテナの統計は、個々の Web アプリケーションごとに表示されます。Web コンテナのサーブレットに関して利用可能な統計を 168 ページの「Web コンテナの統計」に、Web モジュールに関して利用可能な統計を 168 ページの「Web コンテナの統計」に、それぞれ示します。

表 16–7 Web コンテナ (サーブレット) の統計

Statistic	単位	データ型	コメント
errorcount	番号	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	番号	CountStatistic	その時点までに処理された要求の合計数。

Web モジュールに関して利用可能な統計を、168 ページの「Web コンテナの統計」に示します。

表 16–8 Web コンテナ (Web モジュール) の統計

Statistic	データ型	コメント
jspcount	CountStatistic	この Web モジュール内に読み込まれた JSP ページの数。
jspreloadcount	CountStatistic	この Web モジュール内に再読み込みされた JSP ページの数。
sessionstotal	CountStatistic	この Web モジュールに対して作成されたセッションの合計数。

表 16–8　Web コンテナ (Web モジュール) の統計　　(続き)

Statistic	データ型	コメント
activesessionscurrent	CountStatistic	この Web モジュールで現在アクティブになっているセッションの数。
activesessionshigh	CountStatistic	この Web モジュールで同時にアクティブになれるセッションの最大数。
rejectedsessionstotal	CountStatistic	この Web モジュールで拒否されたセッションの合計数。これは、最大許可セッション数がすでにアクティブになっていたために作成されなかったセッションの数です。
expiredsessionstotal	CountStatistic	この Web モジュールで期限切れになったセッションの合計数。
sessionsize (EE のみ)	AverageRangeStatistic	この Web モジュールのセッションのサイズ。値は high、low、average のいずれかです。ただし、直列化されたセッションの場合はバイト値になります。
containerlatency (EE のみ)	AverageRangeStatistic	応答時間要求全体の Web コンテナ部分の応答時間。値は high、low、average のいずれかです。
sessionpersisttime (EE のみ)	AverageRangeStatistic	この Web モジュールの HTTP セッション状態のバックエンドストアへの持続化に要した時間 (ミリ秒値、low、high、average のいずれか)。
cachedsessionscurrent (EE のみ)	CountStatistic	この Web モジュールで現在メモリ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、169 ページの「HTTP サービスの統計」に示します。この統計は Platform Edition のみに適用されます。Enterprise Edition の場合の HTTP サービス統計については、179 ページの「PWC (Production Web Container) の統計」を参照してください。

表 16–9 HTTP サービスの統計 (Platform Edition のみに適用)

Statistic	単位	データ型	コメント
bytesreceived	バイト	CountStatistic	各要求プロセッサが受信したバイトの累計値。
bytessent	バイト	CountStatistic	各要求プロセッサが送信したバイトの累計値。
currentthreadcount	番号	CountStatistic	リスナースレッドプール内に現在存在している処理スレッドの数。
currentthreadsbusy	番号	CountStatistic	要求処理用リスナースレッドプール内で現在使用されている要求処理スレッドの数。
errorcount	番号	CountStatistic	エラー回数の累計値。これは、応答コードが 400 以上になった場合の回数を表します。
maxsparethreads	番号	CountStatistic	存在可能な未使用応答処理スレッドの最大数。
minsparethreads	番号	CountStatistic	存在可能な未使用応答処理スレッドの最小数。
maxthreads	番号	CountStatistic	リスナーが作成する要求処理スレッドの最大数。
maxtime	ミリ秒	CountStatistic	スレッド処理時間の最大値。
processing-time	ミリ秒	CountStatistic	各要求の処理に要した時間の累計値。この処理時間は、要求処理時間を要求数で割って得られた平均値です。
request-count	番号	CountStatistic	その時点までに処理された要求の合計数。

JDBC 接続プールの統計

JDBC リソースを監視することで、パフォーマンスを測定するとともに、実行時のリソースの使用状況を把握します。JDBC 接続の作成はコストのかかる処理であり、アプリケーションのパフォーマンス上のボトルネックになることが多いため、JDBC 接続プールで新しい接続がどのように解放/作成されているかや、特定のプールから接続を取得するために待機しているスレッドがどれくらい存在するかを監視することが不可欠です。

JDBC 接続プールに関して利用可能な統計を、次の表に示します。

表 16–10 JDBC 接続プールの統計

Statistic	単位	データ型	説明
numconnfailedvalidation	番号	CountStatistic	開始時刻から前回のサンプリング時刻までの間に検証に失敗した接続プール内の接続の合計数。
numconnused	番号	RangeStatistic	接続の使用状況に関する統計を提供します。現在使用されている合計接続数に加え、過去に使用された接続の最大数 (ハイウォーターマーク) に関する情報も提供します。
numconnfree	番号	RangeStatistic	前回のサンプリング時点におけるプール内の未使用接続の合計数。
numconntimedout	番号	BoundedRangeStatistic	開始時刻から前回のサンプリング時刻までの間にタイムアウトしたプール内の接続の合計数。
averageconnwaittime	番号	CountStatistic	コネクタ接続プールに対する接続要求が成功した場合の平均接続待ち時間を示します。
waitqueuelength	番号	CountStatistic	サービスを受けるためにキュー内で待機している接続要求の数。
connectionrequestwaittime		RangeStatistic	接続要求の最長待ち時間と最短待ち時間。現在の値は、プールのサービスを最後に受けた要求の待ち時間を示します。
numconncreated	ミリ秒	CountStatistic	前回のリセット後に作成された物理接続の数。
numconndestroyed	番号	CountStatistic	前回のリセット後に破棄された物理接続の数。
numconnacquired	番号	CountStatistic	プールから取得された論理接続の数。
numconnreleased	番号	CountStatistic	プールに解放された論理接続の数。

JMS サービスおよびコネクタサービスの統計

コネクタ接続プールに関して利用可能な統計を、171 ページの「JMS サービスおよびコネクタサービスの統計」に示します。コネクタ作業管理に関する統計を、171 ページの「JMS サービスおよびコネクタサービスの統計」に示します。

表 16–11 コネクタ接続プールの統計

Statistic	単位	データ型	説明
numconnfailedvalidation	番号	CountStatistic	開始時刻から前回のサンプリング時刻までの間に検証に失敗した接続プール内の接続の合計数。
numconnused	番号	RangeStatistic	接続の使用状況に関する統計を提供します。現在使用されている合計接続数に加え、過去に使用された接続の最大数 (ハイウォーターマーク) に関する情報も提供します。
numconnfree	番号	RangeStatistic	前回のサンプリング時点におけるプール内の未使用接続の合計数。
numconntimedout	番号	CountStatistic	開始時刻から前回のサンプリング時刻までの間にタイムアウトしたプール内の接続の合計数。
averageconnwaittime	番号	CountStatistic	接続プールからサービスを受けるまでにかかった平均接続待ち時間。
waitqueuelenght	番号	CountStatistic	サービスを受けるためにキュー内で待機している接続要求の数。
connectionrequestwaittime		RangeStatistic	接続要求の最長待ち時間と最短待ち時間。現在の値は、プールのサービスを最後に受けた要求の待ち時間を示します。
numconncreated	ミリ秒	CountStatistic	前回のリセット後に作成された物理接続の数。
numconndestroyed	番号	CountStatistic	前回のリセット後に破棄された物理接続の数。
numconnacquired	番号	CountStatistic	プールから取得された論理接続の数。
numconnreleased	番号	CountStatistic	プールに解放された論理接続の数。

コネクタ作業管理に関して利用可能な統計を、171 ページの「JMS サービスおよびコネクタサービスの統計」に示します。

表 16–12 コネクタ作業管理の統計

Statistic	データ型	説明
activeworkcount	RangeStatistic	コネクタによって実行された作業オブジェクトの数。

表 16–12 コネクタ作業管理の統計 (続き)

Statistic	データ型	説明
waitqueuelength	RangeStatistic	実行される前にキュー内で待機している作業オブジェクトの数。
workrequestwaittime	RangeStatistic	作業オブジェクトが実行されるまでの最長待ち時間と最短待ち時間。
submittedworkcount	CountStatistic	コネクタモジュールによって送信された作業オブジェクトの数。
rejectedworkcount	CountStatistic	Application Server によって拒否された作業オブジェクトの数。
completedworkcount	CountStatistic	完了した作業オブジェクトの数。

ORB の接続マネージャーの統計

ORB の接続マネージャーに関して利用可能な統計を、173 ページの「ORB の接続マネージャーの統計」に示します。

表 16–13 ORB の接続マネージャーの統計

Statistic	単位	データ型	説明
connectionsidle	番号	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	番号	CountStatistic	ORB への接続のうち、使用中のものの合計数を提供します。
totalconnections	番号	BoundedRangeStatistic	ORB への接続の合計数。

スレッドプールの統計

スレッドプールに関して利用可能な統計を、次の表に示します。

表 16–14 スレッドプールの統計

Statistic	単位	データ型	説明
averagetimeinqueue	ミリ秒	RangeStatistic	キュー内の要求が処理されるまでの平均待ち時間 (ミリ秒)。
averageworkcompletion-time	ミリ秒	RangeStatistic	1 つの作業の平均完了時間 (ミリ秒)。
currentnumberofthreads	番号	BoundedRangeStatistic	要求処理スレッドの現在の数。

表 16–14 スレッドプールの統計 (続き)

Statistic	単位	データ型	説明
numberofavailablethreads	番号	CountStatistic	利用可能なスレッドの数。
numberofbusythreads	番号	CountStatistic	ビジー状態のスレッドの数。
totalworkitemsadded	番号	CountStatistic	その時点までに作業キューに追加された作業項目の合計数。

トランザクションサービスの統計

トランザクションサービスを使えば、クライアントはトランザクションサブシステムをフリーズできます。フリーズすると、トランザクションをロールバックしたり、フリーズ時点で処理中であったトランザクションを特定したりできます。トランザクションサービスに関して利用可能な統計を、次の表に示します。

表 16–15 トランザクションサービスの統計

Statistic	データ型	説明
activecount	CountStatistic	現在アクティブなトランザクションの数。
activeids	StringStatistic	現在アクティブなトランザクションの ID。それらの各トランザクションは、トランザクションサービスのフリーズ後にロールバックすることができます。
committedcount	CountStatistic	コミットされたトランザクションの数。
rolledbackcount	CountStatistic	ロールバックされたトランザクションの数。
state	StringStatistic	トランザクションがフリーズされたかどうかを示します。

Java 仮想マシン (JVM) の統計

JVM の監視可能な属性は、常に有効になっています。JVM に関して利用可能な統計を、次の表に示します。

表 16–16 JVM の統計

Statistic	データ型	説明
heapsize	BoundedRangeStatistic	JVM のメモリーヒープサイズの上限と下限の間にある常駐メモリーフットプリント。

表 16–16 JVM の統計 (続き)

Statistic	データ型	説明
uptime	CountStatistic	JVM の稼働時間。

J2SE 5.0 の JVM 統計

Application Server がバージョン 5.0 以上の J2SE 上で動作するように設定されている場合、JVM から追加の監視情報を取得できます。監視レベルを「低」に設定すると、この追加情報の表示が有効になります。監視レベルを「高」に設定すると、さらにシステム内の各ライブスレッドに関する情報も表示されます。J2SE 5.0 で利用可能な追加監視機能の詳細については、『Monitoring and Management for the Java Platform』というタイトルの文書を参照してください。この文書は http://java.sun.com/j2se/1.5.0/docs/guide/management/ で利用可能になっています。

J2SE 5.0 の監視ツールについては、http://java.sun.com/j2se/1.5.0/docs/tooldocs/#manage を参照してください。

J2SE 5.0 の JVM で利用可能なクラス読み込み関連の統計を、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–17 J2SE 5.0 の JVM 統計- クラス読み込み

Statistic	データ型	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、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–18 J2SE 5.0 の JVM 統計 - コンパイル

Statistic	データ型	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、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–19　J2SE 5.0 の JVM 統計 - ガベージコレクション

Statistic	データ型	説明
collectioncount	CountStatistic	実行されたコレクションの合計回数。
collectiontime	CountStatistic	コレクション時間の累計値 (ミリ秒)。

J2SE 5.0 の JVM で利用可能なメモリー関連の統計を、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–20　J2SE 5.0 の JVM 統計 - メモリ

Statistic	データ型	説明
objectpending finalizationcount	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。
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、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–21　J2SE 5.0 の JVM 統計 - オペレーティングシステム

Statistic	データ型	説明
arch	StringStatistic	オペレーティングシステムのアーキテクチャー。
availableprocessors	CountStatistic	JVM が使用できるプロセッサの数。
name	StringStatistic	オペレーティングシステムの名前。

表 16–21　J2SE 5.0 の JVM 統計 - オペレーティングシステム　　(続き)

Statistic	データ型	説明
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、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–22　J2SE 5.0 の JVM 統計 - ランタイム

Statistic	データ型	説明
name	StringStatistic	実行中の JVM を表す名前
vmname	StringStatistic	JVM 実装の名前。
vmvendor	StringStatistic	JVM 実装のベンダー。
vmversion	StringStatistic	JVM 実装のバージョン。
specname	StringStatistic	JVM 仕様の名前。
specvendor	StringStatistic	JVM 仕様のベンダー。
specversion	StringStatistic	JVM 仕様のバージョン。
managementspecversion	StringStatistic	JVM が実装している管理仕様のバージョン。
classpath	StringStatistic	システムクラスローダーがクラスファイルの検索時に使用するクラスパス。
librarypath	StringStatistic	Java のライブラリパス。
bootclasspath	StringStatistic	ブートストラップクラスローダーがクラスファイルの検索時に使用するクラスパス。
inputarguments	StringStatistic	JVM に渡された入力引数。main メソッドに対する引数は含みません。
uptime	CountStatistic	JVM の稼働時間 (ミリ秒)。

J2SE 5.0 の JVM で利用可能な ThreadInfo 関連の統計を、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–23　J2SE 5.0 の JVM 統計 - ThreadInfo

Statistic	データ型	説明
threadid	CountStatistic	スレッドの ID。
threadname	StringStatistic	スレッドの名前

表 16–23 J2SE 5.0 の JVM 統計 - ThreadInfo (続き)

Statistic	データ型	説明
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。
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、175 ページの「J2SE 5.0 の JVM 統計」に示します。

表 16–24 J2SE 5.0 の JVM 統計 - スレッド

Statistic	データ型	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。

表 16–24　J2SE 5.0 の JVM 統計 - スレッド　(続き)

Statistic	データ型	説明
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。
monitordeadlocked threads	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。

- 179 ページの「PWC (Production Web Container) の統計」、PWC 仮想サーバー
- 179 ページの「PWC (Production Web Container) の統計」、PWC 要求
- 179 ページの「PWC (Production Web Container) の統計」、PWC ファイルキャッシュ
- 179 ページの「PWC (Production Web Container) の統計」、PWC キープアライブ
- 179 ページの「PWC (Production Web Container) の統計」、PWC DNS
- 179 ページの「PWC (Production Web Container) の統計」、PWC Thread Pool
- 179 ページの「PWC (Production Web Container) の統計」、PWC 接続キュー
- 179 ページの「PWC (Production Web Container) の統計」、PWC HTTP サービス

PWC 仮想サーバーに関する統計を、179 ページの「PWC (Production Web Container) の統計」に示します。

表 16–25　PWC 仮想サーバーの統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
id	StringStatistic	仮想サーバーの ID。
mode	StringStatistic	仮想サーバーのモード。unknown、active のいずれかを選択できます。
hosts	StringStatistic	この仮想サーバーからサービスを受けているホストの名前。
interfaces	StringStatistic	仮想サーバーに設定されたインタフェース (リスナー) のタイプ。

PWC 要求に関して利用可能な統計を、次の表に示します。

表 16–26　PWC 要求の統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
method	StringStatistic	要求で使用されたメソッド。
uri	StringStatistic	最後に処理された URI。
countrequests	CountStatistic	処理された要求の数。
countbytestransmitted	CountStatistic	送信バイト数。この情報が利用不可能な場合は 0
countbytesreceived	CountStatistic	受信バイト数。この情報が利用不可能な場合は 0。
ratebytesreceived	CountStatistic	サーバーで定義されたある期間内に受信されたデータの受信速度。この情報が利用不可能な場合は 0
maxbytestransmissionrate	CountStatistic	サーバーで定義されたある期間内に送信されたデータの最大送信速度。この情報が利用不可能な場合は 0。
countopenconnections	CountStatistic	現在開いている接続の数。この情報が利用不可能な場合は 0。
maxopenconnections	CountStatistic	同時に開ける接続の最大数。この情報が利用不可能な場合は 0。
count2xx	CountStatistic	コード 2XX の応答の合計数。
count3xx	CountStatistic	コード 3XX の応答の合計数。
count4xx	CountStatistic	コード 4XX の応答の合計数。
count5xx	CountStatistic	コード 5XX の応答の合計数。
countother	CountStatistic	その他の応答コードを含む応答の合計数。
count200	CountStatistic	コード 200 の応答の合計数。
count302	CountStatistic	コード 302 の応答の合計数。
count304	CountStatistic	コード 304 の応答の合計数。
count400	CountStatistic	コード 400 の応答の合計数。
count401	CountStatistic	コード 401 の応答の合計数。
count403	CountStatistic	コード 403 の応答の合計数。
count404	CountStatistic	コード 404 の応答の合計数。
count503	CountStatistic	コード 503 の応答の合計数。

キャッシュ情報セクションでは、ファイルキャッシュの使用状況に関する情報が提供されます。PWC ファイルキャッシュに関する統計を、次の表に示します。

表 16–27　PWC ファイルキャッシュの統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
flagenabled	CountStatistic	ファイルキャッシュが有効になっているかどうかを示します。有効な値は、0 (no)、1 (yes) のいずれかです。
secondsmaxage	CountStatistic	有効なキャッシュエントリの最大有効期間 (秒)。
countentries	CountStatistic	現在のキャッシュエントリの数。単一のキャッシュエントリは単一の URI を表します。
maxentries	CountStatistic	同時に存在可能なキャッシュエントリの最大数。
countopenentries	CountStatistic	特定のオープンファイルに関連付けられたエントリの数。
maxopenentries	CountStatistic	特定のオープンファイルに同時に関連付けることのできるキャッシュエントリの最大数。
sizeheapcache	CountStatistic	キャッシュコンテンツ格納用のヒープ領域。
maxheapcachesize	CountStatistic	キャッシュファイルコンテンツ格納用のヒープ領域の最大サイズ。
sizemmapcache	CountStatistic	メモリにマップされたファイルのコンテンツ用として使用するアドレス空間。
maxmmapcachesize	CountStatistic	ファイルキャッシュがメモリにマップされたファイルのコンテンツ用として使用するアドレス空間の最大サイズ。
counthits	CountStatistic	キャッシュ検索の成功回数。
countmisses	CountStatistic	キャッシュ検索の失敗回数。
countinfohits	CountStatistic	ファイル情報検索の成功回数。
countinfomisses	CountStatistic	キャッシュファイル情報検索の失敗回数。
countcontenthits	CountStatistic	キャッシュファイルコンテンツのヒット数。
countcontentmisses	CountStatistic	ファイル情報検索の失敗回数。

このセクションでは、サーバーの HTTP レベルキープアライブシステムに関する情報が提供されます。PWC キープアライブに関して利用可能な統計を、次の表に示します。

表 16–28　PWC キープアライブの統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
countconnections	CountStatistic	キープアライブモードの接続の数。
maxconnections	CountStatistic	同時に存在可能なキープアライブモード接続の最大数。
counthits	CountStatistic	キープアライブモードの接続が有効な要求を生成した回数の合計。
countflushes	CountStatistic	キープアライブ接続がサーバーによって閉じられた回数。
countrefusals	CountStatistic	サーバーがキープアライブスレッドに接続を渡せなかった回数。その原因はおそらく、持続接続が多すぎたことにあります。
counttimeouts	CountStatistic	クライアント接続が何の活動も見られないままタイムアウトに達し、サーバーがそのキープアライブ接続を終了した回数。
secondstimeout	CountStatistic	アイドル状態のキープアライブ接続が閉じられるまでの時間 (秒)。

DNS キャッシュでは、IP アドレスと DNS 名がキャッシュされます。サーバーの DNS キャッシュはデフォルトで無効になっています。単一のキャッシュエントリは、単一の IP アドレスまたは単一の DNS 名の検索を表します。PWC DNS に関して利用可能な統計を、次の表に示します。

表 16–29　PWC DNS の統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
flagcacheenabled	CountStatistic	DNS キャッシュが有効 (オン) になっているかどうかを示します。0 (off)、1 (on) のいずれかです。
countcacheentries	CountStatistic	キャッシュ内に現在存在している DNS エントリの数。
maxcacheentries	CountStatistic	キャッシュ内に格納できる DNS エントリの最大数。
countcachehits	CountStatistic	DNS キャッシュ検索の成功回数。
countcachemisses	CountStatistic	DNS キャッシュ検索の失敗回数。
flagasyncenabled	CountStatistic	非同期 DNS 検索が有効 (オン) になっているかどうかを示します。0 (off)、1 (on) のいずれかです。
countasyncnamelookups	CountStatistic	非同期 DNS 名検索の合計回数。

表 16–29　PWC DNS の統計 (EE のみ)　　(続き)

Attribute Name（属性名）	データ型	説明
countasyncaddrlookups	CountStatistic	非同期 DNS アドレス検索の合計回数。
countasynclookupsinprogress	CountStatistic	処理中の非同期検索の数。

PWC スレッドプールに関する統計を、次の表に示します。

表 16–30　PWC スレッドプールの統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
id	StringStatistic	スレッドプールの ID。
countthreadsidle	CountStatistic	現在アイドル状態になっている要求処理スレッドの数。
countthreads	CountStatistic	要求処理スレッドの現在の数。
maxthreads	CountStatistic	同時に存在可能な要求処理スレッドの最大数。
countqueued	CountStatistic	このスレッドプールの処理待ちキューに格納されている要求の数。
peakqueued	CountStatistic	キュー内に同時に格納された要求の最大数。
maxqueued	CountStatistic	キュー内に同時に格納可能な要求の最大数。

接続キューとは、処理される前の要求が格納されるキューのことです。接続キューの統計は、キュー内のセッション数や接続が受け付けられるまでの平均遅延時間などを示します。PWC 接続キューに関する統計を、次の表に示します。

表 16–31　PWC 接続キューの統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
id	StringStatistic	接続キューの ID。
counttotalconnections	CountStatistic	受け付けられた接続の合計数。
countqueued	CountStatistic	キュー内に現在存在している接続の数。
peakqueued	CountStatistic	キュー内に同時に存在していた接続の最大数。
maxqueued	CountStatistic	接続キューの最大サイズ。
countoverflows	CountStatistic	キューがいっぱいになったために接続を格納できなかった回数。

表 16–31　PWC 接続キューの統計 (EE のみ)　　(続き)

Attribute Name（属性名）	データ型	説明
counttotalqueued	CountStatistic	キューに格納された接続の合計数。1 つの接続がキュー内に複数回格納される可能性があります。このため、counttotalqueued の値は、counttotalconnections の値以上になります。
tickstotalqueued	CountStatistic	接続がキュー内で費やした合計ティック数。ティックは、システムに依存する時間の単位です。
countqueued1minuteaverage	CountStatistic	キュー内接続数の過去 1 分間における平均値。
countqueued5minuteaverage	CountStatistic	キュー内接続数の過去 5 分間における平均値。
countqueued15minuteaverage	CountStatistic	キュー内接続数の過去 15 分間における平均値。

PWC HTTP サービスに関する統計を、次の表に示します。

表 16–32　PWC HTTP サービスの統計 (EE のみ)

Attribute Name（属性名）	データ型	説明
id	StringStatistic	HTTP サービスのインスタンス名。
versionserver	StringStatistic	HTTP サービスのバージョン番号。
timestarted	StringStatistic	HTTP サービスが起動された時刻 (GMT)。
secondsrunning	CountStatistic	HTTP サービス起動後の経過時間 (秒)。
maxthreads	CountStatistic	各インスタンス内のワークスレッドの最大数。
maxvirtualservers	CountStatistic	各インスタンス内に設定可能な仮想サーバーの最大数。
flagprofilingenabled	CountStatistic	HTTP サービスのパフォーマンスプロファイリングが有効になっているかどうか。有効な値は 0、1 のいずれかです。
flagvirtualserver overflow	CountStatistic	maxvirtualservers を超える仮想サーバーが設定されているかどうかを示します。これを 1 に設定すると、すべての仮想サーバーで統計が追跡されなくなります。
load1minuteaverage	CountStatistic	要求負荷の過去 1 分間における平均値。
load5minuteaverage	CountStatistic	要求負荷の過去 5 分間における平均値。
load15minuteaverage	CountStatistic	要求負荷の過去 15 分間における平均値。

表 16–32　PWC HTTP サービスの統計 (EE のみ)　　(続き)

Attribute Name（属性名）	データ型	説明
ratebytestransmitted	CountStatistic	サーバーで定義されたある期間内に送信されたデータの送信速度。この情報が利用不可能な場合、結果は 0 になります。
ratebytesreceived	CountStatistic	サーバーで定義されたある期間内に受信されたデータの受信速度。この情報が利用不可能な場合、結果は 0 になります。

監視の有効化と無効化

管理コンソールで監視レベルを設定するには、「設定」ノード > サーバーインスタンスのノード >「監視」ページの順に選択し、監視レベルを変更するサービスの値を選択します。デフォルトでは、すべてのコンポーネントとサービスで監視がオフになっています。

監視レベルの設定方法の詳細は、管理コンソールのオンラインヘルプを参照してください。

コマンド行で asadmin set を使用すると、さまざまな Application Server コンポーネントの監視をオンにできます。たとえば、次のコマンドを実行すると、HTTP サービスの監視をオンにできます。

```
asadmin> set --user admin-user
server.monitoring-service.module-monitoring-levels.http-service=HIGH
```

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 = HIGH
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
```

set コマンドを使用して、監視をオフにします。

たとえば、次のコマンドを実行すると、HTTP サービスの監視が無効になります。

```
asadmin> set --user admin-user
server.monitoring-service.module-monitoring-levels.http-service=OFF
```

また、ほかのコンポーネントの監視を無効にするには、set コマンドを使用し、監視レベルに OFF を指定します。

監視データの表示

管理コンソールで監視データを表示するには、スタンドアロンのサーバーインスタンスの「監視」ページに移動し、監視が有効なサーバーインスタンスに配備済みのコンポーネントや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。

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。監視データの表示方法の詳細は、管理コンソールのオンラインヘルプを参照してください。

コマンド行ユーティリティーを使用して監視データを表示するには、asadmin list および asadmin get コマンドに続けて監視可能なオブジェクトのドット表記名を使用します。手順を次に示します。

1. 監視可能なオブジェクトの名前を表示するには、asadmin list コマンドを使用します。

   たとえば、サーバーインスタンスで監視が有効なアプリケーションコンポーネントおよびサブシステムのリストを表示するには、次のコマンドを端末ウィンドウに入力します。

   ```
   asadmin> list --user adminuser --monitor server
   ```

   前述のコマンドは、監視が有効なアプリケーションコンポーネントおよびサブシステムの次のようなリストを返します。

   ```
   server.resources
   server.connector-service
   server.orb
   server.jms-service
   server.jvm
   server.applications
   server.http-service
   server.thread-pools
   ```

2. 監視が有効なアプリケーションコンポーネントまたはサブシステムの監視統計を表示するには、asadmin get コマンドを使用します。

   統計を取得するには、asadmin get コマンドを端末ウィンドウに入力して、前述の手順の list コマンドで表示された名前を指定します。次の例では、特定のオブジェクトのサブシステムからすべての属性を取得します。

```
asadmin> get --user adminuser --monitor server.jvm.*
```

   このコマンドは次の属性およびデータを返します。

```
server.jvm.dotted-name = server.jvm
server.jvm.heapsize-current = 21241856
server.jvm.heapsize-description = Provides statistical information about
   the JVM's memory heap size.
server.jvm.heapsize-highwatermark = 21241856
server.jvm.heapsize-lastsampletime = 1080232913938
server.jvm.heapsize-lowerbound = 0
server.jvm.heapsize-lowwatermark = 0
server.jvm.heapsize-name = JvmHeapSize
server.jvm.heapsize-starttime = 1080234457308
server.jvm.heapsize-unit = bytes
server.jvm.heapsize-upperbound = 518979584
server.jvm.uptime-count = 1080234457308
server.jvm.uptime-description = Provides the amount of time the JVM has
   been running.
server.jvm.uptime-lastsampletime = 1080234457308
server.jvm.uptime-name = JvmUpTime
server.jvm.uptime-starttime = 1080232913928
server.jvm.uptime-unit = milliseconds
```

ドット表記名とその指定方法について

asadmin list コマンドと asadmin get コマンドでは、監視可能オブジェクトのドット表記名を指定します。すべての子オブジェクトのアドレス指定にはドット (.) 文字が区切り文字として使用され、それらの名前は「ドット表記名」と呼ばれます。子ノードが単独タイプの場合、その監視オブジェクトタイプを指定するだけで、そのオブジェクトを指定できます。それ以外の場合は、type.name 形式の名前を指定する必要があります。

たとえば、http-service は、有効な監視可能オブジェクトタイプの 1 つであり、単独タイプです。インスタンス server の http-service を表す単独タイプの子ノードを指定する場合、ドット表記名は次のようになります。

```
server.http-service
```

もう 1 つ例を挙げます。applications は、有効な監視可能オブジェクトタイプですが、単独タイプではありません。たとえば、アプリケーション PetStore を表す、単独タイプでない子ノードを指定するには、ドット表記名は次のようになります。

```
server.applications.petstore
```

また、監視可能なオブジェクトの特定の属性も、ドット表記名で指定します。たとえば、http-service には、bytesreceived-lastsampletime という名前の監視可能な属性があります。次の名前は、bytesreceived 属性を指定していることになります。

```
server.http-service.server.http-listener-1.
  bytesreceived-lastsampletime
```

管理者は、asadmin list コマンドと asadmin get コマンドの有効なドット表記名を覚えておく必要はありません。list コマンドを使えば、利用可能な監視可能オブジェクトが表示され、ワイルドカードパラメータ付きの get コマンドを使えば、任意 の監視可能オブジェクトで利用可能なすべての属性を確認することができます。

list コマンドと get コマンドでドット表記名を使用する場合、根本的に次のことを前提としています。

- list コマンドでドット表記名の後にワイルドカード (*) が指定されていなかった場合、現在のノードの直接の子ノードが結果として返される。たとえば、list --user *adminuser* --monitor server を実行した場合、server ノードに属するすべての直接の子ノードが一覧表示される。
- list コマンドでドット表記名の後に .* 形式のワイルドカードが指定されていた場合、現在のノードの子ノード階層ツリーが結果として返される。たとえば、list --user *adminuser* --monitor server.applications.* を実行した場合、applications のすべての子ノードに加え、それらの配下にある子ノードなども一覧表示される。
- list コマンドで **dottedname*、*dotted* * *name*、**dotted name** * のいずれかの形式でドット表記名の前後にのワイルドカードが指定されていた場合、そのマッチングパターンによって生成された正規表現にマッチするすべてのノードとそれらの子ノードが、結果として返される。
- get コマンドの末尾に「.*」、「*」のいずれかが指定されていた場合、マッチング対象の現在のノードに属する属性とその値のセットが、結果として返される。

詳細については、194 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

list コマンドの例

list コマンドは、指定されたサーバーインスタンス名で現在監視されているアプリケーションコンポーネントやサブシステムに関する情報を提供します。このコマン

ドを使えば、特定のサーバーインスタンスの監視可能なコンポーネントやそのサブコンポーネントを表示できます。list のより詳しい例については、194 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

例 1

```
asadmin> list --user admin-user --monitor server
```

前述のコマンドは、監視が有効なアプリケーションコンポーネントおよびサブシステムの次のようなリストを返します。

```
server.resources
server.orb
server.jvm
server.jms-service
server.connector-service
server.applications
server.http-service
server.thread-pools
```

また、指定されたサーバーインスタンス内で現在監視されているアプリケーションを一覧表示することも可能です。これは、get コマンドを使って特定のアプリケーションの特定の監視統計を取得する場合に便利です。

例 2

```
asadmin> list --user admin-user --monitor server.applications
```

次の結果が返されます。

```
server.applications.adminapp
 server.applications.admingui
server.applications.myApp
```

get コマンドの例

get コマンドは、次の監視対象情報を取得します。

- 特定のコンポーネントまたはサブシステム内で監視されているすべての属性
- 特定のコンポーネントまたはサブシステム内で監視されている特定の属性

  特定のコンポーネントまたはサブシステムに存在しない属性が要求された場合、エラーが返されます。同様に、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、194 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

例 1

特定のオブジェクトのすべての属性をサブシステムから取得します。

```
asadmin> get --user admin-user --monitor server.jvm.*
```

次の結果が返されます。

```
server.jvm.dotted-name= server.jvm
server.jvm.heapsize-current = 21241856
server.jvm.heapsize-description = Provides statistical information about
   the JVM's memory heap size.
server.jvm.heapsize-highwatermark = 21241856
server.jvm.heapsize-lastsampletime = 1080232913938
server.jvm.heapsize-lowerbound = 0
server.jvm.heapsize-lowwatermark = 0
server.jvm.heapsize-name = JvmHeapSize
server.jvm.heapsize-starttime = 1080234457308
server.jvm.heapsize-unit = bytes
server.jvm.heapsize-upperbound = 518979584
server.jvm.uptime-count = 1080234457308
server.jvm.uptime-description = Provides the amount of time the JVM has
   been running.
server.jvm.uptime-lastsampletime = 1080234457308
server.jvm.uptime-name = JvmUpTime
server.jvm.uptime-starttime = 1080232913928
server.jvm.uptime-unit = milliseconds
```

例 2

特定の J2EE アプリケーションからすべての属性を取得します。

```
asadmin> get --user admin-user --monitor server.applications.myJ2eeApp.*
```

次の結果が返されます。

```
No matches resulted from the wildcard expression.
CLI137 Command get failed.
```

J2EE アプリケーションレベルで公開されている監視可能な属性が存在しないため、このような応答が表示されました。

例 3

特定のサブシステムから特定の属性を取得します。

```
asadmin> get --user admin-user --monitor server.jvm.uptime-lastsampletime
```

次の結果が返されます。

```
server.jvm.uptime-lastsampletime = 1093215374813
```

例 4

特定のサブシステム属性内から未知の属性を取得します。

```
asadmin> get --user admin-user --monitor server.jvm.badname
```

次の結果が返されます。

```
No such attribute found from reflecting the corresponding Stats
interface: [badname]
CLI137 Command get failed.
```

PetStore サンプルを使用する

次の例は、asadmin ツールを監視目的でどのように使えばよいかを示したものです。

あるユーザーが、Application Server 上に配備済みのサンプル Petstore アプリケーションに含まれる特定のメソッドの呼び出し回数を調査しようとしています。その配備先インスタンスの名前は、server です。list コマンドと get コマンドを併用することで、そのメソッドの目的の統計情報にアクセスします。

1. Application Server と asadmin ツールを起動します。
2. いくつかの有用な環境変数を設定することで、それらの値をコマンドごとに入力しないですむようにします。

   ```
   asadmin> export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=admin123
   asadmin>export AS_ADMIN_HOST=localhost AS_ADMIN_PORT=4849
   ```
3. インスタンス server の監視可能なコンポーネントを一覧表示します。

   ```
   asadmin> list --user adminuser --monitor server*
   ```

   次のような出力結果が返されます。

   ```
   server
   server.applications
   server.applications.CometEJB
   server.applications.ConverterApp
   server.applications.petstore
   server.http-service
   server.resources
   server.thread-pools
   ```

   この監視可能なコンポーネントの一覧には、thread-pools、http-service、resources、および配備済みで有効化されているすべての applicationsが含まれています。

4. PetStore アプリケーションの監視可能なサブコンポーネントを一覧表示します (--monitor の代わりに -m を使用可能)。

   ```
   asadmin> list -m server.applications.petstore
   ```

   次の結果が返されます。

   ```
   server.applications.petstore.signon-ejb_jar
   server.applications.petstore.catalog-ejb_jar
   server.applications.petstore.uidgen-ejb_jar
   server.applications.petstore.customer-ejb_jar
   server.applications.petstore.petstore-ejb_jar
   server.applications.petstore.petstore\.war
   server.applications.petstore.AsyncSenderJAR_jar
   server.applications.petstore.cart-ejb_jar
   ```

5. Petstore アプリケーションの EJB モジュール signon-ejb_jar の監視可能なサブコンポーネントを一覧表示します。

   ```
   asadmin> list -m server.applications.petstore.signon-ejb_jar
   ```

   次の結果が返されます。

   ```
   server.applications.petstore.signon-ejb_jar.SignOnEJB
   server.applications.petstore.signon-ejb_jar.UserEJB
   ```

6. Petstore アプリケーションの EJB モジュール signon-ejb_jar のエンティティー Bean UserEJB に含まれる監視可能なサブコンポーネントを一覧表示します。

   ```
   asadmin> list -m server.applications.petstore.signon-ejb_jar.UserEJB
   ```

   次の結果が返されます (ドット表記名はスペースの関係で削除してある)。

   ```
   server.applications.petstore.signon-ejb_jar.UserEJB.bean-cache
   server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods
   server.applications.petstore.signon-ejb_jar.UserEJB.bean-pool
   ```

7. PetStore アプリケーションの EJB モジュール signon-ejb_jar のエンティティー Bean UserEJB に含まれるメソッド getUserName 内の監視可能なサブコンポーネントを一覧表示します。

   ```
   asadmin> list -m
   server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.getUserName
   ```

   次の結果が返されます。

   ```
   Nothing to list at server.applications.petstore.signon-ejb_jar.
   UserEJB.bean-methods.getUserName. To get the valid names beginning with a
   string, use the wildcard "*" character. For example, to list all names
   that begin with "server", use "list server*".
   ```

8. メソッドに対する監視可能なサブコンポーネントは存在しません。メソッド `getUserName` の監視可能なすべての統計を取得します。

```
asadmin> get -m
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.getUserName.*
```

次の結果が返されます。

```
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-description = Provides the time in milliseconds
   spent during the last successful/unsuccessful attempt to execute the
   operation.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-lastsampletime = 1079981809259
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-name = ExecutionTime
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-unit = count
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-description = Provides the number of times an
   operation was called, the total time that was spent during the
   invocation and so on.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-lastsampletime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-maxtime = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-mintime = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-name = ExecutionTime
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-totaltime = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.methodstatistic-unit =
 server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-description = Provides the total number of errors
   that occured during invocation or execution of an operation.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
```

```
getUserName.totalnumerrors-lastsampletime = 1079981809273
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-name = TotalNumErrors
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumerrors-unit = count
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-count = 0
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-description = Provides the total number of
   successful invocations of the method.
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-lastsampletime = 1079981809255
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-name = TotalNumSuccess
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-starttime = 1079980593137
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.totalnumsuccess-unit = count
```

9. また、実行回数など、特定の統計を取得するには、次のようなコマンドを使用します。

```
asadmin> get -m server.applications.petstore.signon-ejb_jar.
UserEJB.bean-methods.getUserName.executiontime-count
```

次の結果が返されます。

```
server.applications.petstore.signon-ejb_jar.UserEJB.bean-methods.
getUserName.executiontime-count = 1
```

すべてのレベルにおける list コマンドと get コマンドの予想出力

次の各表は、ツリーの各レベルにおけるコマンド、ドット表記名、および対応する出力を示したものです。

表 16–33 トップレベル

コマンド	ドット表記名	出力
`list -m`	`server`	`server.applicationsserver.thread-poolsserver.resourcesserver.http-serviceserver.transaction-serviceserver.orb.connection-managersserver.orb.connection-managers.orb\.Connections\.Inbound\.AcceptedConnectionsserver.jvm`
`list -m`	`server.*`	このノードから下の子ノード階層。
`get -m`	`server.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。

次の表に、アプリケーションレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–34 アプリケーションレベル

コマンド	ドット表記名	出力

`list -m`	`server.applications`
	または、
	`*applications`
		`appl1app2web-module1_warejb-module2_jar...`

`list -m`	`server.applications.*`
	または、
	`*applications.*`
		このノードから下の子ノード階層。

`get -m`	`server.applications.*`
	または、
	`*applications.*`
		このノードに属性が存在しないことを示すメッセージだけが表示されます。

次の表に、アプリケーションレベルのスタンドアロンモジュールとエンタープライズアプリケーションのコマンド、ドット表記名、および対応する出力を示します。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール

コマンド	ドット表記名	出力
list -m	server.applications.app1 または、 *app1 注意: このレベルが適用可能なのは、エンタープライズアプリケーションが配備されている場合だけです。スタンドアロンモジュールが配備されている場合には適用できません。	ejb-module1_jarweb-module2_warejb-module3_jarweb-module3_war...
list -m	server.applications.app1.* または、 *app1.*	このノードから下の子ノード階層。
get -m	server.applications.app1.* または、 *app1.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.applications.app1.ejb-module1_jar または、 *ejb-module1_jar または、 server.applications.ejb-module1_jar	bean1bean2bean3...
list -m	server.applications.app1.ejb-module1_jar または、 *ejb-module1_jar または、 server.applications.ejb-module1_jar	このノードから下の子ノード階層。
get -m	server.applications.app1.ejb-module1_jar.* または、 *ejb-module1_jar.* または、 server.applications.ejb-module1_jar.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール (続き)

コマンド	ドット表記名	出力
list -m	server.applications.app1.ejb-module1_jar.bean1 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	次の子ノード一覧が表示されます。 bean-poolbean-cachebean-method
list -m	server.applications.app1.ejb-module1_jar.bean1 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	子ノードの階層とこのノードとそれより下のすべての子ノードの全属性の一覧。
get -m	server.applications.app1.ejb-module1_jar.bean1.* 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	次の属性とそれらの関連付けられた値が表示されます。 CreateCount_CountCreateCount_ DescriptionCreateCount_ LastSampleTimeCreateCount_ NameCreateCount_ StartTimeCreateCount_ UnitMethodReadyCount_ CurrentMethodReadyCount_ DescriptionMethodReadyCount_ HighWaterMarkMethodReadyCount_ LastSampleTimeMethodReadyCount_ LowWaterMarkMethodReadyCount_ NameMethodReadyCount_ StartTimeMethodReadyCount_ UnitRemoveCount_CountRemoveCount_ DescriptionRemoveCount_ LastSampleTimeRemoveCount_ NameRemoveCount_StartTimeAttribute RemoveCount_Unit
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-pool 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	属性は表示されず、server.applications.appl.ejb-module1_jar.bean1-cache には表示すべき情報がないことを示すメッセージが表示されます。特定の文字列で始まる有効な名前を取得するには、ワイルドカード (*) 文字を使用します。たとえば、server で始まるすべての名前を一覧表示するには、list server* を使用します。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール (続き)

コマンド	ドット表記名	出力
get -m	server.applications.app1.ejb-module1_jar.bean1.bean-pool.* 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	表 1-4 で説明した EJB プール属性に対応する属性と値の一覧。
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-cache 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	属性は表示されず、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.ejb-module1_jar.bean1.bean-cache.* 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	表 1-5 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。
list -m	server.applications.app1.ejb-module1_jar.bean1.bean-method.method1 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	属性は表示されず、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.ejb-module1_jar.bean1.bean-method.method1.* 注意: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	表 1-2 で説明した EJB メソッド属性に対応する属性と値の一覧。
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。

表 16–35 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール (続き)

コマンド	ドット表記名	出力
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。

次の表に、HTTP サービスレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–36 HTTP サービスレベル

コマンド	ドット表記名	出力
list -m	server.http-service	仮想サーバーの一覧。
get -m	server.http-service.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.http-service.server	HTTP リスナーの一覧。
get -m	server.http-service.server.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.http-service.server.http-listener1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.http-service.server.*	HTTP サービス属性に対応する属性と値の一覧。

次の表に、スレッドプールレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–37 スレッドプールレベル

コマンド	ドット表記名	出力
list -m	server.thread-pools	スレッドプール名の一覧。
get -m	server.thread-pools.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.thread-pools.orb\.threadpool\.thread-pool-1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。

表 16–37 スレッドプールレベル (続き)

コマンド	ドット表記名	出力
get -m	server.thread-pools..orb\.threadpool\.thread-pool-1.*	スレッドプール属性に対応する属性と値の一覧。

次の表に、リソースレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–38 リソースレベル

コマンド	ドット表記名	出力
list -m	server.resources	プール名の一覧。
get -m	server.resources.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.resources.jdbc-connection-pool-pool.connection-pool1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.resources.jdbc-connection-pool-pool.connection-pool1.*	接続プール属性に対応する属性と値の一覧。

次の表に、トランザクションサービスレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–39 トランザクションサービスレベル

コマンド	ドット表記名	出力
list -m	server.transaction-service	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.transaction-service.*	トランザクションサービス属性に対応する属性と値の一覧。

次の表に、ORB レベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–40　ORB レベル

コマンド	ドット表記名	出力
`list -m`	`server.orb`	`server-orb.connection-managers`
`get -m`	`server.orb.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。
`list -m`	`server.orb.connection-managers`	1 つまたは複数の ORB 接続マネージャー名。
`get -m`	`server.orb.connection-managers.*`	このノードに属性が存在しないことを示すメッセージだけが表示されます。
`list -m`	`server.orb.connection-managers.orb\.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.*`	ORB 接続マネージャー属性に対応する属性と値の一覧。

次の表に、JVM レベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 16–41　JVM レベル

コマンド	ドット表記名	出力
`list -m`	`server.jvm`	属性は表示されず、代わりに次のメッセージが表示されます。「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。」
`get -m`	`server.jvm.*`	JVM 属性に対応する属性と値の一覧。

JConsole の使用

ここでは、次の内容について説明します。

- 202 ページの「JConsole から Application Server への接続のセキュリティーを有効にする」
- 203 ページの「JConsole を Application Server に接続する前提条件」
- 203 ページの「JConsole を Application Server に接続する」
- 204 ページの「安全に JConsole を Application Server に接続する」

Application Server の管理と監視は、JMX をベースにしています。つまり、管理対象コンポーネントは、MBean で表されます。Java 2 Standard Edition (J2SE) 5.0 を使用すると、JVM を監視したり、JVM MBean を表示したりして、状況を理解することができ

ます。このインストゥルメンテーションを公開するために、Application Server ではシステム JMX コネクタサーバーという標準 JMX コネクタサーバーの設定を提供します。Application Server の起動時に、この JMX コネクタサーバーのインスタンスを起動し、信頼できるクライアントにインストゥルメンテーションを公開します。

Java Monitoring and Management Console (JConsole) は、JMX バックエンドを管理できる一般的な JMX コネクタです。JConsole (http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jconsole.html) は、J2SE 5.0 からの標準 JDK ディストリビューションの一部として利用できます。JConsole の詳細は、http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html を参照してください。

Application Server で使用できるように JConsole を設定すると、Application Server は JMX コネクタのサーバー側となり、JConsole は JMX コネクタの優先クライアント側となります。

JConsole から Application Server への接続のセキュリティーを有効にする

Application Server 、つまり JMX コネクタサーバー側への接続方法は、接続のトランスポート層のセキュリティーによって若干異なります。サーバー側がセキュリティー保護されている (トランスポート層のセキュリティーが保証されている) 場合、クライアント側で実行する設定があります。

- デフォルトでは、Application Server の Platform Edition のシステム JMX コネクタサーバー側はセキュリティー保護されていません。
- デフォルトでは、Application Server の Enterprise Edition のシステム JMX コネクタサーバー側はセキュリティー保護されています。
- 通信に使用されるプロトコルは、RMI/JRMP です。JMX コネクタのセキュリティーが有効な場合、使用されるプロトコルは SSL 上の RMI/JRMP です。

注 – SSL 上の RMI では、クライアントが目的のサーバーと通信できるようにするための追加チェックは行われません。そのため、JConsole の使用時は、悪意のあるホストにユーザー名とパスワードを送信している可能性が常にあります。セキュリティーが安全であるかどうかの確認は、管理者に完全に委ねられています。

Platform Edition ドメインを appserver.sun.com のようなマシンにインストールすると、管理サーバーまたは単にドメインである DAS (ドメイン管理サーバー) の domain.xml に次のようなエントリが含まれます。

```
<!- – The JSR 160 "system-jmx-connector" – –><jmx-connector accept-all="false"
address="0.0.0.0" auth-realm-name="admin-realm" enabled="true" name="system"
port="8686" protocol="rmi_jrmp" security-enabled="false"/> <!- – The JSR 160
"system-jmx-connector" – –>
```

JMX コネクタの `security-enabled` フラグは *false* です。Enterprise Edition が稼働している場合、または Platform Edition の JMX コネクタのセキュリティーを有効にした場合、このフラグは *true* に設定されます。

```
<!- – The JSR 160 "system-jmx-connector" – –><jmx-connector accept-all="false"
address="0.0.0.0" auth-realm-name="admin-realm" enabled="true" name="system"
port="8686" protocol="rmi_jrmp" security-enabled="true"/>
...</jmx-connector><!- – The JSR 160 "system-jmx-connector" – –>
```

JConsole を Application Server に接続する前提条件

JConsole の設定は、2 つに分かれます。サーバー側とクライアント側です。Application Server ドメインは、強力な Solaris サーバーである `appserver.sun.com` と呼ばれるマシンにインストールされます。これがサーバー側です。

クライアント側にも Application Server のインストールがあります。ここでは、クライアント側は Windows マシンで、Java SE 5.0 と Application Server がインストールされているものとします。

注 – クライアント側で Application Server のインストールが必要になるのは、Application Server ドメインのリモートマシン上でセキュリティーが有効な場合 (Enterprise Edition のデフォルト) だけです。前述の Solaris マシンで Application Server Platform Edition ドメインを管理する場合、このクライアントマシンに Application Server のインストールは必要ありません。

同じマシン上にサーバー側とクライアント側がある場合、`localhost` を使用してホスト名を指定できます。

JConsole を Application Server に接続する

この節では、JMX コネクタでセキュリティーを有効にしないで JConsole を Application Server に接続する方法について説明します。デフォルトでは、Application Server Platform Edition でセキュリティーは有効ではありません。

1. `appserver.sun.com` でドメインを起動します。
2. *JDK_HOME*`/bin/jconsole` を実行して JConsole を起動します。
3. JConsole の「エージェントに接続」タブで、ユーザー名、パスワード、ホスト名、およびポート (デフォルトは 8686) を入力します。

ユーザー名は管理ユーザーの名前、パスワードはドメインの管理パスワードを参照します。

4. 「接続」をクリックします。

   JConsole ウィンドウの各種タブに、MBean、VM 情報などが表示されます。

安全に JConsole を Application Server に接続する

この節では、JMX コネクタでセキュリティーを有効にして JConsole を Application Server に接続する方法について説明します。デフォルトでは、Application Server Enterprise Edition でセキュリティーは有効です。この手順は、Platform Edition の JMX コネクタでセキュリティーを有効にした場合に使用してください。

1. クライアントマシン (JConsole がインストールされている) に Application Server をインストールします。

   この作業が必要になるのは、信頼するドメイン管理サーバーのサーバー証明書の場所を JConsole に対して通知するためです。この証明書を取得するには、*remote* `asadmin` コマンドを 1 回以上呼び出しますが、そのためには Application Server のローカルインストールが必要です。

2. `appserver.sun.com` で Application Server Enterprise Edition を起動します。

   これは Enterprise Edition ドメインであるため、システム JMX コネクタサーバーはセキュリティー保護されています。

3. ローカル Application Server インストールから *install-dir*`/bin/asadmin list --user admin --secure=true --host appserver.sun.com --port 4849` を実行します。4849 はサーバーの管理ポートです。

   この例では `asadmin list` コマンドを選択していますが、任意のリモート `asadmin` コマンドを実行できます。ここで、`appserver.sun.com` の DAS から送信される証明書を受け入れることを要求されます。

4. `y` を押して、`appserver.sun.com` のドメイン管理サーバーから送信される証明書を受け入れます。

   サーバーの証明書は、クライアントマシンのホームディレクトリにある `.asadmintruststore` ファイルに格納されます。

注 – サーバーマシンとクライアントマシンが同じである場合、この手順は必要ありません。つまり、JConsole も `appserver.sun.com` で稼働している場合です。

---

5. 次の JConsole コマンドを使用して、DAS のトラストストアの場所を JConsole に通知します。

   JDK-dir`/bin/jconsole.exe -J-Djavax.net.ssl.trustStore="C:/Documents and Settings/`*user*`/.asadmintruststore"`

これで、この証明書は JConsole によって自動的に信頼されます。

6. *JDK_HOME*/bin/jconsole を実行して JConsole を起動します。
7. JConsole の「エージェントに接続」タブで、ユーザー名、パスワード、ホスト名、およびポート (デフォルトは 8686) を入力します。

   ユーザー名は管理ユーザーの名前、パスワードはドメインの管理パスワードを参照します。
8. 「接続」をクリックします。

   JConsole ウィンドウの各種タブに、MBean、VM 情報などが表示されます。

◆ ◆ ◆ 第 17 章

Java 仮想マシンと詳細設定

Java 仮想マシン (JVM) は、コンパイル済みの Java プログラムでバイトコードを実行する、インタプリタ型の処理エンジンです。JVM は Java バイトコードをホストマシンのネイティブ命令に変換します。Java プロセスの 1 つであるアプリケーションサーバーには JVM が必要であり、JVM がアプリケーションサーバーを実行し、アプリケーションサーバー上で稼働する Java アプリケーションをサポートします。JVM 設定は、アプリケーションサーバー設定の一部です。

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。次の項があります。

JVM 設定の調整

アプリケーションサーバーを設定する一環として、Java 仮想マシンの使用を拡張する設定を定義します。管理コンソールを使用して JVM の設定を変更するには、「アプリケーションサーバー」>「JVM 設定」タブの順に選択し、次のように JVM の一般設定を定義します。

- Java ホーム: Java 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 Java SE ソフトウェアに依存します。

  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 Java EE 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。

- Javac オプション: Java プログラミング言語コンパイラのコマンド行オプションを入力します。Application Server は EJB コンポーネントの配備時にコンパイラを実行します。

- デバッグ: JPDA (Java Platform Debugger Architecture) によるデバッグを設定するときは、このフィールドの「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。
- デバッグオプション: デバッグが有効なときに JVM に渡される JPDA オプションを指定します。
- RMI コンパイルオプション: `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。
- バイトコードプリプロセッサ: 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。

詳細設定

管理コンソールを使用して詳細なアプリケーション設定を行うには、「アプリケーションサーバー」>「詳細」タブ>「アプリケーション設定」タブの順に選択し、次のようにアプリケーション設定を行います。

- 再読み込み: アプリケーションの動的再読み込みを有効にするには、このチェックボックスにチェックマークを付けます。

  動的再読み込みが有効な場合 (デフォルト)、アプリケーションやモジュールのコードや配備記述子を変更しても再配備する必要はありません。変更した JSP またはクラスファイルをアプリケーションやモジュールの配備ディレクトリにコピーするだけです。サーバーは変更を定期的に確認し、変更があるとアプリケーションを自動的かつ動的に再配備します。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。また、再読み込みが行われると、その転送時のセッションが無効になります。クライアントはセッションを再起動する必要があります。
- 再読込のポーリング間隔: アプリケーションやモジュールのコード変更をチェックし、動的に再読み込みする間隔を定義します。デフォルトは 2 です。
- 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てから、停止している分数を指定します。

また、配備設定を次のように定義します。

- 自動配備: アプリケーションの自動配備を有効にするには、このチェックボックスにチェックマークを付けます。

自動配備では、アプリケーションやモジュールのファイル (JAR、WAR、RAR、または EAR) を特殊なディレクトリにコピーします。ファイルは Application Server によって自動的に配備されます。

- 自動配備のポーリング間隔: アプリケーションやモジュールのコード変更をチェックし、動的に再読み込みする間隔を指定します。デフォルトは 2 です。
- ベリファイア: 配備記述子ファイルを確認するには、ベリファイアの「有効」チェックボックスにチェックマークを付けます。必要があれば、ユーザーを追加します。
- プリコンパイル: JSP ファイルを事前にコンパイルするには、プリコンパイルの「有効」チェックボックスにチェックマークを付けます。

◆ ◆ ◆ 第 18 章

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。

* (アスタリスク) は、ドット表記名の任意の場所で使用できます。これは正規表現におけるワイルドカード文字のような役割を果たします。ワイルドカード文字を使用すると、ドット表記名のすべての部分を折りたたむことができるという利点があります。たとえば、this.is.really.long.hierarchy などの長形式のドット表記名を、th*.hierarchy に短縮することができます。ただし、常に . を使用して名前を区切ります。* によって、ドット表記名の全体を一覧表示できます。

この付録の内容は次のとおりです。

トップレベル要素

domain.xml ファイル内のすべてのトップレベル要素で、次の条件が満たされている必要があります。

- サーバー、設定、クラスタ、またはノードエージェントの名前はそれぞれ一意である必要があります。
- サーバー、設定、クラスタ、またはノードエージェントに「domain」という名前を付けることはできません。
- サーバーインスタンスに「agent」という名前を付けることはできません。

次の表に、トップレベル要素と対応するドット表記名プレフィックスを示します。

要素名	ドット表記名プレフィックス
applications	`domain.applications`
resources	`domain.resources`
configurations	`domain.configs`
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.node-agents`
lb-configs	`domain.lb-configs`
system-property	`domain.system-property`

次の 2 つのレベルの別名が利用可能です。

1. 1 つ目のレベルの別名を使えば、プレフィックス `domain.servers` または `domain.clusters` を使わずにサーバーインスタンスまたはクラスタの属性にアクセスできます。したがって、たとえば、`server1` という形式のドット表記名は、ドット表記名 `domain.servers.server1` (`server1` は特定のサーバーインスタンス) にマッピングされます。
2. 2 つ目のレベルの別名を使えば、特定のクラスタまたはスタンドアロンサーバーインスタンス (ターゲット) の設定、アプリケーション、およびリソースを参照できます。

次の表に、サーバー名またはクラスタ名で始まるドット表記名と、それらの別名であるドメイン配下のトップレベル名を示します。

ドット表記名	別名	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。

ドット表記名	別名	説明
target.resources.*	domain.resources.*	この別名の解決結果は、*target* によって参照されるすべての jdbc-connection-pool、connector-connection-pool、resource-adapter-config、およびその他のすべてのリソースになります。

次の表に、サーバー名またはクラスタ名で始まるドット表記名と、それらの別名である、そのサーバーまたはクラスタによって参照されている設定内のトップレベル名を示します。

ドット表記名	別名
target.http-service	*config-name*.http-service
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ーを取得する際のドット表記名属性は、*clustered-instance-name*.*system-property* ではなく、`domain.servers.`*clustered-instance-name*.*system-property* のように記述してください。

asadmin コマンド

`asadmin get`、`set`、および `list` コマンドは、Application Server の抽象階層に対するナビゲーションメカニズムを提供するために、連携して動作します。階層には、設定と監視の 2 つがあり、これらのコマンドはこの両方に対して機能します。`list` コマンドでは、読み取り専用または変更可能な属性を持つ管理コンポーネントの完全修飾のドット表記名で表示されます。

設定階層は、変更可能な属性を提供します。一方、監視階層にある管理コンポーネントの属性は純粋に読み取り専用です。設定階層は、大まかにドメインのスキーマドキュメントに基づいています。`list` コマンドを使用すると、必要な階層内の特定の管理コンポーネントに到達できます。次に、`get` および `set` コマンドを呼び出すと、管理コンポーネントの属性の名前と値を取得したり、値を設定することができます。ワイルドカード (*) オプションを使用すると、指定した完全修飾のドット表記名の中から一致するものをすべて取得することができます。`get`、`set`、および `list` コマンドの使用例については、次のマニュアルページを参照してください。

`get`(1)
`set`(1)
`list`(1)

◆ ◆ ◆ 第 19 章

asadmin ユーティリティー

Application Server には、asadmin という名前のコマンド行管理ユーティリティーが含まれています。asadmin ユーティリティーは、Application Server の起動と停止のほかに、ユーザー、リソース、およびアプリケーションの管理にも使用されます。

この章の内容は次のとおりです。

asadmin コマンドの使用法

asadmin ユーティリティーを使用すると、Application Server の管理タスクを実行できます。この asadmin ユーティリティーは、管理コンソールの代わりに使用できます。

asadmin ユーティリティーは、ユーザーの実行する操作やタスクを特定するコマンドを呼び出します。これらのコマンドでは大文字と小文字が区別されます。短形式のオプションの引数にはダッシュ (-) が、長形式のオプションの引数には二重ダッシュ (--) が付いています。オプションによって、ユーティリティーによるコマンドの実行方法を制御します。オプションでも大文字と小文字を区別します。機能のオン/オフを切り替えるブール型のオプションを除いて、大部分のオプションには引数値が必要です。オペランドは引数値の後ろに表示され、空白、タブ、または二重ダッシュ (--) で区切られます。asadmin ユーティリティーでは、オプションとその値の後ろに続くものをオペランドとして処理します。

例 19–1 構文の例

asadmin *command* [-short_option] [*short_option_argument*]* [--long_option [*long_option_argument*]* [*operand*]*

```
asadmin create-profiler -u admin --passwordfile password.txt myprofiler
```

Solaris プラットフォーム上の Application Server の asadmin ユーティリティーコマンドのマニュアルページにアクセスするには、MANPATH 環境変数に $AS_INSTALL/man を追加します。

asadmin ユーティリティーコマンドの全体的な使用法の情報は、--help オプションを呼び出すことで取得できます。コマンドを指定すると、そのコマンドの使用法に関する情報が表示されます。コマンドを指定せずに --help オプションを使用すると、使用可能なすべてのコマンドの一覧が表示されます。

例 19–2 help コマンドの例

asadmin --help では、一般的なヘルプが表示されます。

asadmin *command* --help では、特定のコマンドのヘルプが表示されます。

ここでは、次の内容について説明します。

- 217 ページの「マルチモードと対話型モード」
- 217 ページの「ローカルコマンド」
- 217 ページの「リモートコマンド」
- 219 ページの「パスワードファイル」

マルチモードと対話型モード

asadmin ユーティリティーは、コマンドシェルを呼び出して、またはマルチコマンドモード (multimode コマンドと呼ばれる) で使用できます。コマンドシェルの呼び出しを使用する場合は、コマンドシェルから asadmin ユーティリティーを呼び出します。asadmin がコマンドを実行し、終了します。マルチコマンドモードでは、asadmin が一度呼び出されると、asadmin が終了するまで、複数のコマンドを受け入れます。終了後は通常のコマンドシェルの呼び出しに戻ります。マルチコマンドモードで設定した環境変数は、multimode を終了するまで、あとに続くすべてのコマンドに使用されます。また、あらかじめ準備したコマンドの一覧をファイルまたは標準入力から渡す (パイプする) ことによって、コマンドを提供することもできます。また、マルチモードセッション内から multimode を呼び出すこともできます。2 つ目のマルチモード環境を終了すると、元のマルチモード環境に戻ります。

また、asadmin ユーティリティーは、対話型モードまたは非対話型モードで実行できます。デフォルトでは、対話型モードのオプションが有効になっています。対話型モードでは、必要な引数を入力するように求められます。対話型モードオプションは、すべての環境のコマンドシェルの呼び出しで使用することができます。コマンドプロンプトから一度に 1 つのコマンドを実行したり、ファイルから multimode で実行したりするときは、multimode で対話型モードオプションを使用できます。multimode のコマンド (入力ストリームからパイプされた場合) や、別のプログラムから呼び出されたコマンドは、対話型モードでは実行できません。

ローカルコマンド

ローカルコマンドは、管理サーバーが存在しなくても実行できます。ただし、コマンドを実行したり、インストールディレクトリやドメインディレクトリに対するアクセス権を得たりするには、ドメインをホストしているマシンにユーザーがログインする必要があります。

ローカルまたはリモートで実行できるコマンドの場合、環境内またはコマンド行のどちらかで、--host、--port、--user、および --passwordfile オプションのいずれか 1 つが設定されていれば、そのコマンドはリモートモードで実行されます。また、コマンド行または環境内で、ローカルオプションが何も設定されていない場合でも、デフォルトでコマンドはローカルで実行されます。

リモートコマンド

リモートコマンドの実行は、管理サーバーに接続してコマンドを実行することによって常に行われます。稼働中の管理サーバーが必要です。すべてのリモートコマンドで、次の共通オプションが必須になります。

表 19–1 リモートコマンドの必須オプション

短形式のオプション	オプション	定義
-H	--host	ドメイン管理サーバーの稼働しているマシン名。デフォルト値は、localhost です。
-p	--port	管理用の HTTP/S ポート。これは、ドメインを管理するためにブラウザで指定するポートです。たとえば、http://localhost:4848 などです。Platform Edition のデフォルトのポート番号は 4848 です。
-u	--user	認証されたドメイン管理サーバーの管理ユーザー名。asadmin login コマンドを使用してドメインに対して認証を行った場合、その後の操作では、この特定のドメインに対して --user オプションを指定する必要はありません。
	--passwordfile	--passwordfile オプションは、特定の形式でパスワードエントリを格納しているファイルの名前を指定します。パスワードのエントリには、パスワード名の前に AS_ADMIN_ というプレフィックス (大文字) を付ける必要があります。 たとえば、ドメイン管理サーバーのパスワードを指定するには、次の形式のエントリを使用します。AS_ADMIN_PASSWORD=*password* (この *password* は実際の管理者パスワード)。その他の指定できるパスワードには、AS_ADMIN_PASSWORD、AS_ADMIN_USERPASSWORD、AS_ADMIN_ALIASPASSWORD、AS_ADMIN_MAPPEDPASSWORD などがあります。 すべてのリモートコマンドでは、--passwordfile または asadmin login を使用するか、コマンドプロンプトによる対話形式で、ドメイン管理サーバーに対して認証を行うための管理パスワードを指定する必要があります。asadmin login コマンドを使用するのは、管理パスワードを指定するときだけです。リモートコマンド用に指定する必要があるその他のパスワードについては、--passwordfile を使用するか、コマンドプロンプトで入力します。 asadmin login コマンドを使用してドメインに対して認証を行った場合、その後の操作では、この特定のドメインに対して --passwordfile オプションを使用して管理パスワードを指定する必要はありません。ただし、これは AS_ADMIN_PASSWORD オプションにしか適用されません。なお、個別のコマンド (update-file-user など) が要求する場合は、その他のパスワード (AS_ADMIN_USERPASSWORD など) を指定する必要があります。 セキュリティー上の理由により、環境変数として指定されたパスワードは、asadmin によって読み取られません。
-s	--secure	true に設定した場合、SSL/TLS を使用してドメイン管理サーバーと通信します。

表 19–1 リモートコマンドの必須オプション (続き)

短形式のオプション	オプション	定義
-I	--interactive	true (デフォルト) に設定した場合、必須パスワードとユーザーオプションのみの入力が要求されます。
-t	--terse	出力データを簡潔にすることを示します。通常、人間が読みやすい文を避けて、スクリプトで使用するために整形されたデータを優先します。デフォルトは false です。
-e	--echo	true に設定すると、コマンド行の文が標準出力にエコーされます。デフォルトは false です。
-h	--help	コマンドに関するヘルプテキストが表示されます。

パスワードファイル

セキュリティーのため、コマンド行でパスワードを入力する代わりに、ファイルからコマンドのパスワードを設定することができます。--passwordfile オプションを使用すると、パスワードを格納したファイルを取得できます。このファイルの有効な内容は次のとおりです。

例 19–3 パスワードファイルの内容

```
AS_ADMIN_PASSWORD=value
AS_ADMIN_ADMINPASSWORD=value
AS_ADMIN_USERPASSWORD=value
AS_ADMIN_MASTERPASSWORD=value
```

multimode コマンド

multimode コマンドを使用すると、asadmin コマンドを処理できます。コマンド行インタフェースによってコマンドの入力が求められます。入力されたコマンドが実行され、コマンドの結果が表示されたあと、次のコマンドの入力が求められます。さらに、このモードで設定されたすべての asadmin オプション名は、後続のすべてのコマンドで使用されます。exit または quit を入力して multimode を終了するまで、環境を設定したり、コマンドを実行することができます。また、あらかじめ準備したコマンドの一覧をファイルまたは標準入力から渡す (パイプする) ことによって、コマンドを提供することもできます。*multimode* セッション内から multimode を呼び出すことができます。2 つ目の *multimode* 環境を終了すると、元の *multimode* 環境に戻ります。

multimode を呼び出すには、asadmin multimode と入力します。

list、get、set コマンド

asadmin list、get、および set コマンドは、Application Server のドット表記名の階層にナビゲーションメカニズムを提供するために、連携して動作します。階層には、設定と監視 の 2 つがあり、これらのコマンドは両方の階層に対して機能します。list コマンドでは、読み取り専用または変更可能な属性を持つ管理コンポーネントの完全修飾のドット表記名で表示されます。

設定階層は、変更可能な属性を提供します。一方、監視階層にある管理コンポーネントの属性は純粋に読み取り専用です。設定階層は、大まかにドメインのスキーマドキュメントに基づいていますが、監視階層は少し異なっています。

list コマンドを使用すると、必要な階層内の特定の管理コンポーネントに到達できます。次に、get および set コマンドを呼び出すと、すぐに管理コンポーネントの属性の名前と値を取得したり、値を設定することができます。ワイルドカード (*) オプションを使用すると、指定した完全修飾のドット表記名の中から、一致するものをすべて取得できます。

Application Server のドット表記名では、名前全体を複数部分に分けるための区切り文字として「.」(ピリオド) を使用します。これは、Unix ファイルシステムで、ファイルの絶対パス名のレベルを「/」を使用して区切る方法と同じです。get、set、および list コマンドによって受け入れられるドット表記名を形成する場合、次の規則が適用されます。特定のコマンドには追加のセマンティクスが適用されることに留意してください。

- . (ピリオド) は常に、名前を連続した 2 つの部分に区切ります。
- 名前の 1 つの部分は、通常、アプリケーションサーバーのサブシステムまたはその固有のインスタンス、あるいはその両方を特定します。次に例を示します。web-container、log-service、thread-pool-1 など。
- 名前の一部に . (ピリオド) が含まれている場合は、その「.」の前に \ (バックスラッシュ) を付けて、区切り文字として機能しないようにする必要があります。
- * (アスタリスク) は、ドット表記名の任意の場所で使用できます。これは正規表現におけるワイルドカード文字のような役割を果たします。また、* によって、ドット表記名のすべての部分を折りたたむことができます。
  「<classname>this.is.really.long.hierarchy </classname> 」のような長形式のドット表記名を「<classname>th*.hierarchy</classname>」に短縮することができます。ただし、. は常に名前の区切りに使われることに注意してください。
- Solaris で、* をオプション値やオペランドとして使用してコマンドを実行する場合は、引用符が必要です。
- ドット表記名の最上位のスイッチは --monitor または -m であり、所定のコマンド行で個別に指定されます。このスイッチが存在するかしないかによって、アプリケーションサーバー管理の 2 つの階層 (監視と設定) のどちらを選択するのかが示されます。

- ワイルドカード文字をまったく含まない完全なドット表記名を使用する場合は、list および get/set では、セマンティクスが少し異なります。
  - list コマンドは、この完全なドット表記名を、階層内の親ノードの完全な名前として処理します。この名前を list コマンドに与えると、そのレベルの直接の子ノードの名前が単に返されます。たとえば、list server.applications.web-module では、ドメインまたはデフォルトのサーバーに配備されたすべての Web モジュールが一覧表示されます。
  - get および set コマンドは、この完全なドット表記名を、ノードの属性の完全修飾名 (ノードのドット表記名そのものが、このドット表記名の最後の部分を削除したときに取得する名前となる) として処理し、その属性の値を取得または設定します。これはこのような属性が存在する場合です。したがって、最初からこれを実行することはできません。まず、階層内の特定のノードの属性名を見つけるために、ワイルドカード文字の * を使用する必要があります。たとえば、server.applications.web-module.JSPWiki.context-root* では、ドメインまたはデフォルトサーバーに配備された Web アプリケーションのコンテキストルートが返されます。

list コマンドは、これら 3 つのコマンドのナビゲーション機能では、必ず最初に来るものです。特定のアプリケーションサーバーのサブシステムの属性を set (設定) または get (取得) する場合は、そのドット表記名を知っておく必要があります。list コマンドを使用すると、サブシステムのドット表記名を見つけることができます。たとえば、/ で始まる大規模なファイルシステム内の特定のファイルの変更日 (属性) を検索する場合を考えます。最初に、そのファイルのファイルシステム内での場所を検索し、その属性を確認する必要があります。したがって、Application Server の階層を理解するための最初の 2 つのコマンドは、* list "*" と <command>* list * --monitor になります。これらのコマンドのソートされた出力を確認するには、get、set、または list コマンドのマニュアルページを参照してください。

サーバーのライフサイクルコマンド

サーバーのライフサイクルコマンドとは、ドメイン、サービス (DAS)、またはインスタンスを、作成、削除、起動、または停止するコマンドのことです。

表 19–2 サーバーのライフサイクルコマンド

コマンド	定義
`create-service`	無人の自動起動によって DAS が起動されるように設定します。このコマンドは、Solaris 10 では Service Management Facility (SMF) を使用します。これはローカルコマンドで、スーパーユーザー権限のある OS レベルのユーザーとして実行する必要があります。これは Solaris 10 でのみ使用可能です。サービスが作成されたら、ユーザーがサービスを起動、有効化、無効化、または停止する必要があります。DAS は、スーパーユーザーがアクセス権を持つフォルダに格納する必要があります。設定をネットワークファイルシステムに格納することはできません。サービスは、DAS の設定の存在するフォルダを所有する OS レベルのユーザーによって制御されるように作成されます。このコマンドを実行するには、`solaris.smf.*` の承認が必要です。
`create-domain`	ドメインの設定を作成します。ドメインとは管理用の名前空間のことです。どのドメインにも設定があり、その設定は一連のファイルに格納されます。Application Server の所定のインストールでは、任意の数のドメインを作成できます。それぞれのドメインには個別の管理アイデンティティーが与えられます。ドメインは、1 つずつ独立して存在しています。所定のシステムの `asadmin` ユーティリティーに対してアクセス権を持つユーザーは、ドメインを作成し、自分の選択するフォルダにその設定を格納することができます。デフォルトでは、ドメイン設定は *install_dir*`/domains` ディレクトリに作成されます。この場所をオーバーライドして、別の場所に設定を格納することもできます。
`delete-domain`	指定したドメインを削除します。ドメインはすでに存在して、停止している必要があります。
`start-domain`	ドメインを起動します。ドメインのディレクトリが指定されていない場合は、デフォルトの *install_dir*`/domains` ディレクトリにあるドメインが起動します。複数のドメインが存在する場合、*domain_name* オペランドを指定する必要があります。
`stop-domain`	指定したドメインのドメイン管理サーバーを停止します。
`restore-domain`	ドメイン下のファイルをバックアップディレクトリから復元します。
`list-domains`	ドメインを一覧表示します。ドメインのディレクトリが指定されていない場合は、デフォルトの *install_dir*`/domains` ディレクトリにあるドメインが表示されます。複数のドメインが存在する場合、*domain_name* オペランドを指定する必要があります。
`backup-domain`	指定したドメイン下のファイルをバックアップします。
`list-backups`	バックアップリポジトリ内のすべてのバックアップに関するステータス情報を表示します。

表 19–2 サーバーのライフサイクルコマンド (続き)

コマンド	定義
shutdown	管理サーバーと実行中のすべてのインスタンスをシャットダウンします。再起動するには、管理サーバーを手動で起動させる必要があります。

リストおよびステータスコマンド

リストおよびステータスコマンドは、配備されたコンポーネントのステータスを表示します。

表 19–3 リストおよびステータスコマンド

コマンド	定義
show-component-status	配備されたコンポーネントのステータスを取得します。ステータスは、サーバーから返された文字列で表現されます。ステータスを表す文字列は、*app-name* のステータスは enabled である、または *app-name* のステータスは disabled である、と表現されます。
list-components	配備されたすべての Java EE 5 コンポーネントを一覧表示します。--type オプションが指定されていない場合は、すべてのコンポーネントが表示されます。
list-sub-components	配備されたモジュール内か、配備されたアプリケーションのモジュール内にある EJB または サーブレットを一覧表示します。モジュールが指定されていない場合は、すべてのモジュールが表示されます。

配備コマンド

配備コマンドは、アプリケーションを配備したり、クライアントスタブを取得したりします。

表 19–4 配備コマンド

コマンド	定義
deploy	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、コネクタモジュール、またはアプリケーションクライアントモジュールを配備します。コンポーネントがすでに配備済みであるか、すでに存在している場合、--force オプションが true に設定されていれば、強制的に再配備されます。

表 19–4 配備コマンド (続き)

コマンド	定義
deploydir	アプリケーションを配備ディレクトリから直接配備します。配備ディレクトリには、Java EE 仕様に準拠する適切なディレクトリ階層と配備記述子が存在していなければなりません。
get-client-stubs	AppClient スタンドアロンモジュールまたは AppClient モジュールを含むアプリケーション用のクライアントスタブ JAR ファイルを、サーバーマシンからローカルディレクトリに取得します。このコマンドを実行する前に、アプリケーションまたはモジュールを配備済みにしてください。--retrieve オプションを使用して deploy コマンドの一部としてクライアントスタブを取得することもできます。
undeploy	指定した配備済みのコンポーネントを削除します。

Message Queue 管理コマンド

Message Queue 管理コマンドを使用すると、JMS 送信先を管理できます。

表 19–5 Message Queue コマンド

コマンド	定義
create-jmsdest	JMS 物理送信先を作成します。物理送信先とともに、create-jms-resource コマンドを使用して、物理送信先を指定する Name プロパティーを持つ JMS 送信先リソースを作成します。
delete-jmsdest	指定した JMS 送信先を削除します。
flush-jmsdest	指定したターゲットの JMS サービス設定の物理送信先から、メッセージをパージします。
list-jmsdest	JMS 物理送信先を一覧表示します。
jms-ping	JMS サービス (JMS プロバイダとも呼ばれる) が起動して稼働中かどうかを確認します。JMS サービスは、デフォルトでは Application Server の起動時に起動します。また、このコマンドは JMS サービス内のデフォルトの JMS ホストのみを ping します。組み込まれている JMS サービスに ping できない場合には、エラーメッセージが表示されます。

リソース管理コマンド

リソースコマンドを使用すると、アプリケーション内で使用されているさまざまなリソースを管理できます。

表 19–6 リソース管理コマンド

コマンド	定義
create-jdbc-connection-pool	新しい JDBC 接続プールを、指定した JDBC 接続プール名で登録します。
delete-jdbc-connection-pool	JDBC 接続プールを削除します。削除する JDBC 接続プールは、オペランドによって特定されます。
list-jdbc-connection-pools	作成済みの JDBC 接続プールを取得します。
create-jdbc-resource	JDBC リソースを新規作成します。
delete-jdbc-resource	指定した JNDI 名の JDBC リソースを削除します。
list-jdbc-resources	作成済みの JDBC リソースの一覧を表示します。
create-jms-resource	Java Message Service (JMS) 接続ファクトリリソースまたは JMS 送信先リソースを作成します。
delete-jms-resource	指定した JMS リソースを削除します。
list-jms-resources	既存の JMS リソース (送信先および接続ファクトリリソース) を一覧表示します。
create-jndi-resource	JNDI リソースを登録します。
delete-jndi-resource	指定した JNDI 名の JNDI リソースを削除します。
list-jndi-resources	既存のすべての JNDI リソースを特定します。
list-jndi-entries	JNDI ツリーを表示して照会します。
create-javamail-resource	JavaMail セッションリソースを作成します。
delete-javamail-resource	指定した JavaMail セッションリソースを削除します。
list-javamail-resources	既存の JavaMail セッションリソースを一覧表示します。
create-persistence-resource	持続性リソースを登録します。
delete-persistence-resource	持続性リソースを削除します。持続性リソースを削除すると、create-persistence-resource コマンドを使用して作成された JDBC リソースも一緒に削除されます。
list-persistence-resources	すべての持続性リソースを表示します。

表 19–6 リソース管理コマンド (続き)

コマンド	定義
create-custom-resource	カスタムリソースを作成します。カスタムリソースは、javax.naming.spi.ObjectFactory インタフェースを実装する、サーバー規模のカスタムリソースオブジェクトファクトリを指定します。
delete-custom-resource	カスタムリソースを削除します。
list-custom-resources	カスタムリソースを一覧表示します。
create-connector-connection-pool	指定した接続プール名で新しいコネクタ接続プールを追加します。
delete-connector-connection-pool	オペランド connector_connection_pool_name を使用して指定したコネクタ接続プールを削除します。
list-connector-connection-pools	作成済みのコネクタ接続プールを一覧表示します。
create-connector-resource	指定した JNDI 名でコネクタリソースを登録します。
delete-connector-resource	指定した JNDI 名のコネクタリソースを削除します。
list-connector-resources	すべてのコネクタリソースを取得します。
create-admin-object	指定した JNDI 名の管理対象オブジェクトを追加します。
delete-admin-object	指定した JNDI 名の管理対象オブジェクトを削除します。
list-admin-objects	すべての管理対象オブジェクトを一覧表示します。
create-resource-adapter-config	コネクタモジュールの設定情報を作成します。
delete-resource-adapter-config	domain.xml に作成されたコネクタモジュールの設定情報を削除します。
list-resource-adapter-configs	domain.xml 内のコネクタモジュールの設定情報を一覧表示します。
add-resources	指定した XML ファイル内に指定したリソースを作成します。*xml_file_path* は、作成するリソースを格納する XML ファイルへのパスです。DOCTYPE は、resources.xml ファイル内で *install_dir*/lib/dtds/sun-resources_1_2.dtd と指定するようにしてください。
ping-connection-pool	JDBC 接続プールとコネクタ接続プールの両方に対して、接続プールが使用可能かどうかをテストします。たとえば、後で配備する予定のアプリケーション用に JDBC 接続プールを新規作成した場合、そのアプリケーションを配備する前にこのコマンドを使用して JDBC プールをテストします。接続プールに ping する前に、認証された接続プールを作成し、Application Server またはデータベースが起動していることを確認する必要があります。

Application Server の設定コマンド

設定コマンドを使用すると、Application Server の操作を設定できます。ここでは、次の内容について説明します。

一般的な設定コマンド

ここで説明するコマンドによって、Application Server コンポーネントの設定を管理することができます。

表 19–7　一般的な設定コマンド

コマンド	定義
`enable`	指定したコンポーネントを有効にします。コンポーネントがすでに有効になっている場合は、再有効化されます。有効にするには、コンポーネントが配備済みである必要があります。コンポーネントが配備済みでない場合は、エラーメッセージが返されます。
`disable`	指定したコンポーネントを即座に無効にします。コンポーネントが配備済みである必要があります。コンポーネントが配備済みでない場合は、エラーメッセージが返されます。
`export`	後続のコマンド環境に対して、自動エクスポートの変数名にマークを付けます。指定した変数名の値を `unset` するか、`multimode` を終了しないかぎり、後続のコマンドはすべてその変数名の値を使用します。
`get`	属性の名前と値を取得します。
`set`	1 つ以上の設定可能な属性の値を設定します。
`list`	設定可能な要素を一覧表示します。Solaris で、* をオプション値やオペランドとして使用してコマンドを実行する場合は、引用符が必要です。
`unset`	マルチモード環境に対して設定した 1 つ以上の変数を削除します。変数と変数に関連付けられた値は、その環境内に存在しなくなります。

HTTP、IIOP、および SSL リスナーコマンド

HTTP および IIOP リスナーコマンドを使用して、リスナーを管理することができます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–8 IIOP リスナーコマンド

コマンド	定義
create-http-listener	新しい HTTP リスナーを追加します。
delete-http-listener	指定した HTTP リスナーを削除します。
list-http-listeners	既存の HTTP リスナーを一覧表示します。
create-iiop-listener	IIOP リスナーを作成します。
delete-iiop-listener	指定した IIOP リスナーを削除します。
list-iiop-listeners	既存の IIOP リスナーを一覧表示します。
create-ssl	選択した HTTP リスナー、IIOP リスナー、または IIOP サービス内で SSL 要素を作成および設定し、そのリスナーまたはサービス上でセキュリティー保護された通信ができるようにします。
delete-ssl	選択した HTTP リスナー、IIOP リスナー、または IIOP サービス内の SSL 要素を削除します。

ライフサイクルおよび監査モジュールコマンド

ライフサイクルおよび監査モジュールコマンドを使用すると、ライフサイクルモジュールや、監査機能を実装するオプションのプラグインモジュールを制御できるようになります。これらのコマンドは、リモートモードのみでサポートされています。

表 19–9 ライフサイクルモジュールコマンド

コマンド	定義
create-lifecycle-module	ライフサイクルモジュールを作成します。ライフサイクルモジュールによって、Application Server 環境内で短期または長期の Java ベースのタスクを実行する手段が提供されます。
delete-lifecycle-module	指定したライフサイクルモジュールを削除します。
list-lifecycle-modules	既存のライフサイクルモジュールを一覧表示します。
create-audit-module	監査機能を実装するプラグインモジュール用に、指定した監査モジュールを追加します。

表 19–9 ライフサイクルモジュールコマンド (続き)

コマンド	定義
delete-audit-module	指定した監査モジュールを削除します。
list-audit-modules	すべての監査モジュールを一覧表示します。

プロファイラおよびJVMオプションコマンド

プロファイラおよびJVMオプションコマンドを使用すると、プロファイラを管理し、プロファイラの要素を制御することができます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–10 プロファイラおよびJVMオプションコマンド

コマンド	定義
create-profiler	プロファイラ要素を作成します。サーバーインスタンスは、Java設定内のプロファイラ要素によって、特定のプロファイラと連動しています。プロファイラの変更時には、サーバーを再起動する必要があります。
delete-profiler	指定したプロファイラ要素を削除します。サーバーインスタンスは、Java設定内のプロファイラ要素によって、特定のプロファイラと連動しています。プロファイラの変更時には、サーバーを再起動する必要があります。
create-jvm-option	Java設定またはdomain.xmlファイルのプロファイラ要素に、JVMオプションを作成します。プロファイラ用に作成されたJVMオプションは、特定のプロファイラの実行に必要な設定を記録するために使用されます。新しく作成したJVMオプションを有効にするには、サーバーを再起動する必要があります。
delete-jvm-option	Java設定またはdomain.xmlファイルのプロファイラ要素から、JVMオプションを削除します。

仮想サーバーコマンド

仮想サーバーコマンドを使用すると、次のような要素を制御できます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–11 仮想サーバーコマンド

コマンド	定義
create-virtual-server	指定した仮想サーバーを作成します。Application Server で仮想化を行うことで、複数のホストアドレス上で待機している 1 つの HTTP サーバープロセスによって、複数の URL ドメインを処理できるようになります。アプリケーションを 2 つの仮想サーバーで使用できる場合は、同じ物理リソースプールを共有します。
delete-virtual-server	指定した仮想サーバー ID の仮想サーバーを削除します。
list-virtual-server	既存の仮想サーバーを一覧表示します。

スレッドプールコマンド

スレッドプールコマンドを使用すると、次のような要素を制御できます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–12 スレッドプールコマンド

コマンド	定義
create-threadpool	指定した名前付きのスレッドプールを作成します。プール内のスレッドの最大数および最小数、作業キューの数、およびスレッドのアイドルタイムアウトを指定できます。作成したスレッドプールは、IIOP 要求やリソースアダプタの作業管理要求のサービスに使用できます。作成したスレッドプールは、複数のリソースアダプタで使用できます。
delete-threadpool	指定した ID のスレッドプールを削除します。
list-threadpools	すべてのスレッドプールを一覧表示します。

トランザクションおよびタイマーコマンド

トランザクションおよびタイマーコマンドを使用すると、トランザクションおよびタイマーサブシステムを制御できます。これによって、実行中のトランザクションを中断できるようになります。これらのコマンドは、リモートモードのみでサポートされています。

表 19–13 トランザクションコマンド

コマンド	定義
freeze-transaction	実行中のすべてのトランザクションが中断している間、トランザクションサブシステムを凍結します。このコマンドは、実行中のトランザクションをロールバックする前に呼び出します。すでに凍結しているトランザクションサブシステムに対してこのコマンドを呼び出しても、効果はありません。
unfreeze-transaction	中断していた実行中のすべてのトランザクションを再開します。このコマンドは、すでに凍結しているトランザクションに対して呼び出します。
recover-transactions	保留中のトランザクションを手動で回復します。
rollback-transaction	指定したトランザクションをロールバックします。
list-timers	特定のサーバーインスタンスに備えられたタイマーを一覧表示します。

ユーザー管理コマンド

ユーザー管理コマンドは、ファイルレルム認証によってサポートされているユーザーを管理します。これらのコマンドは、リモートモードのみでサポートされています。

表 19–14 ユーザー管理コマンド

コマンド	定義
create-file-user	指定したユーザー名、パスワード、およびグループで、キーファイル内にエントリを作成します。コロン (:) で区切ることによって、複数のグループを作成することもできます。
delete-file-user	指定したユーザー名のエントリをキーファイル内から削除します。
update-file-user	指定した user_name、user_password、およびグループを使用して、キーファイル内の既存のエントリを更新します。コロン (:) で区切ることによって、複数のグループを入力することもできます。
list-file-users	ファイルレルム認証によってサポートされているファイルユーザーの一覧を作成します。
list-file-groups	ファイルレルム認証によってサポートされているユーザーおよびグループを一覧表示します。このコマンドでは、ファイルユーザー内の使用可能なグループが表示されます。

監視データコマンド

監視データコマンドを使用すると、サーバーを監視することができます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–15 監視データコマンド

コマンド	定義
`start-callflow-monitoring`	Web コンテナ、EJB コンテナ、および JDBC からデータを収集して相互に関連付け、要求の完全な呼び出しフロー/パスを提示します。`callflow-monitoring` がオンの場合のみ、データは収集されます。
`stop-callflow-monitoring`	要求の呼び出しフロー情報の収集を無効にします。

ルールコマンド

ルールコマンドを使用すると、サーバーの規則を管理できます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–16 ルールコマンド

コマンド	定義
`create-management-rule`	Application Server インストールや配備済みのアプリケーションをインテリジェントに自己管理するために、新しい管理規則を作成します。
`delete-management-rule`	指定した管理規則を削除します。
`list-management-rules`	使用可能な管理規則を一覧表示します。

データベースコマンド

データベースコマンドを使用すると、Java DB データベース (Apache Derby に基づく) を起動および停止することができます。これらのコマンドは、ローカルモードのみでサポートされています。

表 19–17 データベースコマンド

コマンド	定義
start-database	Application Server で使用可能な Java DB サーバーを起動します。このコマンドは、Application Server に配備されたアプリケーションの操作に対してのみ使用します。
stop-database	Java DB サーバーのプロセスを停止します。Java DB サーバーは Application Server で使用できます。

診断およびロギングコマンド

診断およびロギングコマンドは、Application Server による問題のトラブルシューティングに役立ちます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–18 診断およびロギングコマンド

コマンド	定義
`generate-diagnostic-report`	生成される HTML レポートには、アプリケーションサーバーインスタンスの設定詳細、ロギング詳細、またはプロセス固有の情報などの、アプリケーションサーバーのインストールの詳細情報へのポインタまたはナビゲーションリンクが含まれます。
`generate-jvm-report`	Domain Administration Service を含む、所定のターゲットインスタンスのスレッド (スタックトレースのダンプ)、クラス、およびメモリを表示します。このコマンドは、アプリケーションサーバーインスタンスプロセスでのみ機能します。このコマンドは、アプリケーションサーバープロセスへの `ctrl+break` または `kill -3` 信号の送信などの従来の手法に代わるものです。ターゲットサーバーインスタンスが実行されていない場合、このコマンドは機能しません。
`display-error-statistics`	前回のサーバーの再起動以降の `server.log` 内の重要なメッセージや警告を要約して一覧表示します。
`display-error-distribution`	モジュールレベルでインスタンスの `server.log` から配布されたエラーを表示します。
`display-log-records`	指定のタイムスタンプでの所定のモジュールに関するすべてのエラーメッセージを表示します。

Web サービスコマンド

Web サービスコマンドを使用すると、配備された Web サービスを監視し、変換規則を管理することができます。

表 19–19　Web サービスコマンド

コマンド	定義
configure-webservice-management	配備された Web サービスエンドポイントの監視属性または maxhistorysize 属性を設定します。
create-transformation-rule	Web サービス操作に適用できる XSLT 変換規則を作成します。この規則は、要求、応答、またはその両方に適用できます。
delete-transformation-rule	指定した Web サービスの XSLT 変換規則を削除します。
list-transformation-rules	指定した Web サービスのすべての変換規則を、適用された順に一覧表示します。
publish-to-registry	レジストリサーバーに Web サービスのアーティファクトを発行します。
unpublish-from-registry	レジストリサーバーから Web サービスのアーティファクトの発行を解除します。
list-registry-locations	設定済みの Web サービスエントリのアクセスポイントの一覧を表示します。

セキュリティーサービスコマンド

次のセキュリティーコマンドを使用して、コネクタ接続プールのセキュリティーマッピングを制御します。これらのコマンドは、リモートモードのみでサポートされています。

表 19–20 セキュリティーコマンド

コマンド	定義
create-connector-security-map	指定したコネクタ接続プールのセキュリティーマップを作成します。セキュリティーマップが存在しない場合は、新規に作成されます。また、コンテナ管理のトランザクションベースのシナリオでは、このコマンドを使用して、アプリケーションの呼び出し側アイデンティティー (主体またはユーザーグループ) を適切なエンタープライズ情報システム (EIS) の主体にマップします。1 つ以上の指定したセキュリティーマップをコネクタ接続プールに関連付けることができます。コネクタセキュリティーマップの設定では、ワイルドカード文字としてアスタリスク (*) を使用し、すべてのユーザーまたはすべてのユーザーグループを示すことができます。このコマンドを正常に実行するためには、最初にコネクタ接続プールを作成しておく必要があります。EIS は、組織のデータ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。
delete-connector-security-map	指定したコネクタ接続プールのセキュリティーマップを削除します。
update-connector-security-map	指定したコネクタ接続プールのセキュリティーマップを変更します。
list-connector-security-maps	指定したコネクタ接続プールに属するセキュリティーマップを一覧表示します。
create-message-security-provider	管理者は、特定のメッセージ層 (Application Server のパラメータおよびプロパティーを指定するファイル domain.xml の message-security-config 要素) の provider-config サブ要素を作成できます。
delete-message-security-provide	管理者は、特定のメッセージ層 (Application Server のパラメータおよびプロパティーを指定するファイル domain.xml の message-security-config 要素) の provider-config サブ要素を削除できます。
list-message-security-providers	管理者は、特定のメッセージ層 (domain.xml の message-security-config 要素) のすべてのセキュリティーメッセージプロバイダ (provider-config サブ要素) を一覧表示できます。
create-auth-realm	名前付き認証レルムを追加します。
delete-auth-realm	名前付き認証レルムを削除します。
list-auth-realms	既存の認証レルムを一覧表示します。

パスワードコマンド

パスワードコマンドを使用すると、パスワードを管理して、Application Server のセキュリティーを確保することができます。

表 19–21 パスワードコマンド

コマンド	定義
create-password-alias	パスワードのエイリアスを作成し、これを domain.xml に格納します。エイリアスは、${ALIAS=password-alias-password} という形式のトークンです。エイリアス名に対応するパスワードは、暗号化形式で格納されます。このコマンドでは、セキュリティー保護された対話型形式 (ユーザーがすべての情報の入力を求められる) と、スクリプトの処理しやすい形式 (パスワードがコマンド行で伝送される) の両方の形式が使用できます。
delete-password-alias	パスワードのエイリアスを削除します。
update-password-alias	名前付きターゲットにあるパスワードのエイリアス ID を更新します。
list-password-aliases	すべてのパスワードのエイリアスを一覧表示します。
change-admin-password	このリモートコマンドは、管理パスワードを変更します。このコマンドは対話型で、ユーザーは元の管理パスワードと新しい管理パスワードの両方の入力を求められます (確認入力も必要です)。
change-master-password	このローカルコマンドを使用して、マスターパスワードを変更します。このコマンドは対話型で、ユーザーは元のマスターパスワードと新しいマスターパスワードの両方の入力を求められます。サーバーが停止していないかぎり、このコマンドは機能しません。

domain.xml の検証コマンド

XML 検証コマンドは、domain.xml ファイルの内容を検証します。

表 19–22 domain.xml の検証コマンド

コマンド	定義
verify-domain-xml	domain.xml ファイルの内容を検証します。

カスタム MBean コマンド

MBean コマンドを使用すると、カスタム MBean を管理および登録できます。これらのコマンドは、リモートモードのみでサポートされています。

表 19–23 カスタム MBean コマンド

コマンド	定義
create-mbean	カスタム MBean を作成および登録します。ターゲットの MBeanServer (DAS) が実行されていない場合は、MBean は登録されません。
delete-mbean	カスタム MBean を削除します。ターゲットの MBeanServer が実行されていることを確認します。
list-mbeans	指定したターゲットのカスタム MBean を一覧表示します。

その他のコマンド

その他のコマンドを使用すると、Application Server の各種項目を管理できます。

表 19–24 その他のコマンド

コマンド	定義
login	ユーザーをドメインにログインさせます。(ローカルの) 各種マシン上でさまざまなアプリケーションサーバードメインが作成されている場合、これらの中の任意のマシンから asadmin を呼び出すことによって、任意の場所にあるドメインを (リモートで) 管理することができます。この機能は、特定のマシンが管理クライアントとして選択されており、そのマシンが複数のドメインやサーバーを管理しているような場合に特に役立ちます。任意の場所にあるドメインを管理するために使用される asadmin コマンドは、リモートコマンドと呼ばれます。asadmin login コマンドを使用すると、このようなリモートドメインの管理が簡単になります。login コマンドは対話型モードのみで実行されます。ここでは、管理ユーザー名とパスワードの入力が求められます。正常にログインしたら、ユーザーのホームディレクトリにファイル .asadminpass が作成されます。これは、--savelogin オプションの使用時に create-domain コマンドによって変更されるファイルと同じものです。このコマンドを実行するには、ドメインが実行されている必要があります。
version	バージョン情報を表示します。このコマンドによって、特定のユーザー/パスワード、およびホスト/ポートを使用して管理サーバーと通信できない場合は、ローカルでバージョンを取得し、警告メッセージを表示します。

表 19–24 その他のコマンド　(続き)

コマンド	定義
help	すべての asadmin ユーティリティーコマンドの一覧を表示します。コマンドを指定すると、そのコマンドの使用方法が表示されます。
install-license	Application Server の不正な使用を防止します。このコマンドを使用すると、ライセンスファイルをインストールできます。

索引

ろ