Sun Java System Application Server 9.1 管理ガイド

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

Part No: 820–4604
2007 年 12 月

080609@20490

目次

図目次

表目次

例目次

はじめに

『Sun Java System Application Server 9.1 管理ガイド』では、Application Server サブシステムおよびコンポーネントを管理コンソールから設定、管理、および配備する手順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に関する情報と表記規則について説明しています。

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。Application Server マニュアルの URL (Uniform Resource Locator) は、`http://docs.sun.com/coll/1343.4` です。Application Server への導入としては、次の表に示されている順序で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

マニュアルタイトル	説明
『Documentation Center』	タスクや主題ごとに整理された Application Server のマニュアルのトピック。
『Release Notes』	ソフトウェアとマニュアルに関する最新情報。サポートされているハードウェア、オペレーティングシステム、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。
『Deployment Planning Guide』	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。
『Application Deployment Guide』	アプリケーションおよびアプリケーション コンポーネントの Application Server への配備。配備記述子に関する情報を含みます。

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

マニュアルタイトル	説明
『Developer's Guide』	Application Server 上で動作することを目的とし、Java EE コンポーネントおよび API のオープン Java スタンダードモデルに準拠した、Java 2 Platform, Enterprise Edition (Java EE プラットフォーム) アプリケーションの作成と実装。開発者ツール、セキュリティー、デバッグ、ライフサイクルモジュールの作成に関する情報を含みます。
『Java EE 5 Tutorial』	Java EE 5 プラットフォームテクノロジと API を使用した Java EE アプリケーションの開発。
『Java WSIT Tutorial』	Web サービス相互運用性テクノロジ (WSIT) を使用した Web アプリケーションの開発。WSIT テクノロジを使用する方法、時期、および理由と、各テクノロジがサポートする機能およびオプションについて説明します。
『管理ガイド』	設定、監視、セキュリティー、資源管理、および Web サービス管理を含む Application Server のシステム管理。
『高可用性 (HA) 管理ガイド』	高可用性 (HA) データベースのためのインストール後の設定と管理の手順。
『Administration Reference』	Application Server 設定ファイル `domain.xml` の編集。
『Upgrade and Migration Guide』	旧バージョンの Application Server からのアップグレード、または競合するアプリケーションサーバーからの Java EE アプリケーションの移行。このガイドでは、直前の製品リリースとの違いと、製品仕様との互換性がなくなる可能性のある設定オプションについても説明します。
『Performance Tuning Guide』	パフォーマンスを向上させるための Application Server の調整。
『Troubleshooting Guide』	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を含みます。

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 Sun Java™ Enterprise System (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/1286.3` 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。

その他のスタンドアロンの Sun Java System サーバー製品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 マニュアル (http://docs.sun.com/coll/1343.4)
- Directory Server マニュアル (http://docs.sun.com/coll/1224.1)
- Web Server マニュアル (http://docs.sun.com/coll/1308.3)

Application Server に付属するパッケージ用の Javadoc™ ツールのリファレンスは http://glassfish.dev.java.net/nonav/javaee5/api/index.html にあります。さらに、次のリソースが役立つことがあります。

- 『Java EE 5 Specifications (http://java.sun.com/javaee/5/javatech.html)』
- 『Java EE Blueprints (http://java.sun.com/reference/blueprints/index.html)』

NetBeans™ 統合開発環境 (IDE) でのエンタープライズアプリケーションの作成については、http://www.netbeans.org/kb/55/index.html を参照してください。

Application Server に付属する Java DB データベースについては、http://developers.sun.com/javadb/ を参照してください。

GlassFish Samples プロジェクトは、広範な Java EE テクノロジの実例を示すサンプルアプリケーションを集めたものです。GlassFish Samples は Java EE Software Development Kit (SDK) に付属しています。また、https://glassfish-samples.dev.java.net/ の GlassFish Samples プロジェクトページからも入手できます。

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されているデフォルトのパス名と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

as-install	Application Server のベースインストールディレクトリを表します。	Solaris™ オペレーティングシステムへの Java ES インストールの場合:
		`/opt/SUNWappserver/appserver`
		Linux オペレーティングシステムへの Java ES インストールの場合:
		`/opt/sun/appserver/`
		Solaris および Linux 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 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 への Java ES インストールの場合:
		`/var/opt/SUNWappserver/domains/`
		Linux 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:
		as-install`/domains/`

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

次の表に、このマニュアルで使用する記号の表記規則を示します。

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。
-	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

Sun Java™ System 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)、プロファイルとクラスタの管理、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。

Application Serverの概要

Sun Java System Application Server は Java EE アプリケーションおよび Java Web サービスの開発と配備用の Java EE 互換サーバーを提供します。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。ここでは、次の内容について説明します。

使用法プロファイル

すべての管理ドメインは、そのドメインの機能を特定する使用法プロファイルに関連付けられます。Application Server には次のプロファイルが用意されています。

- 開発者: このプロファイルは、ドメインが開発環境で実行されており、開発中のアプリケーションに NSS キーストア機能やクラスタ化機能 (負荷分散やセッション持続など) が必要でない場合に使用します。

- クラスタ: このプロファイルは、クラスタを作成する必要があるが、高可用性データベース (HADB) や NSS キーストアは必要としない場合に使用します。
- エンタープライズ: このプロファイルは、HADB や NSS を必要とする場合に使用します。このプロファイルは、HADB および NSS を個別にインストールする場合、または Java Enterprise System (JES) の一部として Application Server をインストールする場合にのみ使用できます。Application Server 9.1 でエンタープライズプロファイルを使用する方法については、31 ページの「エンタープライズプロファイルの使用」を参照してください。

注 – Application Server 8.x Enterprise Edition からのアップグレードは、エンタープライズプロファイルでのみサポートされます。Application Server 8.x Platform Edition からアップグレードする場合は、開発者プロファイルを使用します。アップグレード処理の詳細については、『Sun Java System Application Server 9.1 Update 1 Upgrade and Migration Guide』の第 2 章「Upgrading an Application Server Installation」を参照してください。

ドメインは、事前に設定されたランタイムをユーザーアプリケーションに提供します。使用法プロファイルにより、インストールされた Application Server 自体の実行バイナリと、実行環境の設定を区別しやすくなります。つまり、プロファイルにより、Application Server の 1 つの実行バイナリを使用して、特定のニーズに合った異なるドメインを作成できます。たとえば、場合によっては最新の Java EE 仕様を理解するために Application Server を使用する開発者もいます。そのような開発者には、厳格なセキュリティー設定は必要ありません。一方、本稼動環境でアプリケーションを配備するユーザーには、当然ながらセキュリティー保護された環境が必要です。

表 1–1 に、各プロファイルで使用できる機能の一覧を示します。

表 1–1 各プロファイルで使用できる機能

機能	開発者プロファイル	クラスタプロファイル	エンタープライズプロファイル
セキュリティーストア	JKS	JKS	NSS
クラスタ化/スタンドアロンインスタンス	使用不可	利用可能	利用可能
セキュリティーマネージャー	無効	有効	有効
HADB	使用不可	使用不可	利用可能
負荷分散	使用不可	利用可能	利用可能
ノードエージェント	使用不可	利用可能	利用可能

エンタープライズプロファイルの使用

エンタープライズプロファイルを使用するには、次のタスクを実行します。

1. NSS および HADB を個別にダウンロードしてインストールします。
2. `asenv.conf` ファイルを次のように変更します。
   - `AS_HADB` のポイント先を、HADB のインストールフォルダにします。
   - `AS_NSS` のポイント先を、NSS 共有オブジェクトが使用可能なフォルダにします。
   - `AS_NSS_BIN` のポイント先を、`certutil` などの NSS バイナリが格納されたフォルダにします。

以前のドメインの Application Server 9.1 へのアップグレード

`start-domain` コマンドを使用すると、Application Server 8.x または 9.0 のドメインを Application Server 9.1 にアップグレードできます。ドメインをアップグレードするには、次のいずれかの方法を使用します。

- Application Server バイナリのインプレースアップグレードを実行する。

  以前のバージョンの Application Server をポイントするドメインで `start-domain` を実行すると、`asadmin` によって `asupgrade` コマンドが呼び出され、自動的にドメインのインプレースアップグレードが実行されます。
- Application Server バイナリのサイドバイサイドアップグレードを実行する。

  以前のインストールのドメインで `start-domain` を実行します。`asupgrade` コマンドによって、ドメインは最新の Application Server インストールのドメインルートにアップグレードされます。このシナリオでは、アップグレードのターゲットディレクトリは、`asenv.conf` 内の `AS_DEF_DOMAINS_PATH` に定義されます。

Application Server とは

Application Server は、Web パブリッシングからエンタープライズ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。Application Server を使用して、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 9.1 のクラスタプロファイルとエンタープライズプロファイルは、高度なクラスタリング技術とフェイルオーバー技術を提供します。これらの機能により、スケーラブルで高い可用性を備えた Java EE アプリケーションを実行できます。

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

クラスタに Application Server インスタンスを追加することで水平方向への拡張が実現され、それによってシステムの処理能力が向上します。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 のロードバランスにより、クラスタ内の正常な Application Server インスタンスに要求を分散させます。

- 高可用性 - 可用性を有効にすると、クラスタ内の Application Server インスタンスをフェイルオーバーによって保護できます。1 つのアプリケーションサーバーインスタンスが停止すると、利用できなくなったサーバーに割り当てられていたセッションは別の Application Server インスタンスに引き継がれます。セッションの情報は、高可用性データベース (HADB) に格納されます。HADB は、持続的な HTTP セッションとステートフルセッション Beans をサポートします。

Application Server のアーキテクチャー

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

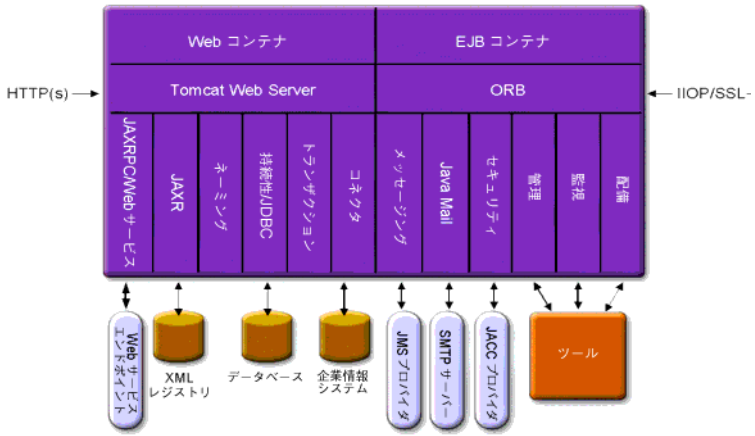

図 1–1　Application Server のアーキテクチャー

- コンテナ - コンテナは、Java EE 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、Web コンテナおよび EJB コンテナという、2 つのタイプの Java EE コンテナを示しています。JSP ページやサーブレットなどの Web コンポーネントは、Web コンテナ内で実行されます。EJB テクノロジのコンポーネントである Enterprise JavaBeans は、EJB コンテナ内で実行されます。

- クライアントアクセス - 実行時に、ブラウザクライアントはインターネット上で使われているプロトコルである HTTP で Web サーバーと通信することにより Web アプリケーションにアクセスします。HTTPS プロトコルは、セキュア通信を必要とするアプリケーションのためにあります。Enterprise Bean クライアントは、IIOP または IIOP/SSL (セキュア) プロトコルを介して ORB (Object Request Broker) と通信します。Application Server には、HTTP、HTTPS、IIOP、および IIOP/SSL プロトコル用の別個のリスナーがあります。各リスナーは、固有のポート番号を排他的に使用しています。
- **Web** サービス - Java EE 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。Java EE 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 - Java EE 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。Java EE 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、Java EE 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

Java EE 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。Java EE プラットフォームでは、次の API およびコンポーネントを介して外部システムにアクセスできます。

- **JDBC** - データベース管理システム (DBMS) は、データの格納、編成、および検索機能を提供します。大部分のビジネスアプリケーションは、アプリケーションが JDBC API 経由でアクセスするリレーショナルデータベースにデータを格納します。データベース内の情報は、多くの場合、持続性があるとされています。これは、ディスク上に保存され、アプリケーションを終了したあとも存在するためです。Application Server バンドルには、Java DB データベースが含まれています。

- 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手段です。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 - Java EE コネクタアーキテクチャーでは、Java EE 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 Java EE コンポーネントを介して EIS にアクセスします。
- **JavaMail** - JavaMail API を介して、アプリケーションは電子メールを送受信するために SMTP サーバーに接続します。
- サーバー管理 - 図 1-1 の右下に、Application Server の管理者によって実行されるタスクの一部が示されています。たとえば、管理者は、アプリケーションを配備 (インストール) し、サーバーのパフォーマンスを監視します。これらのタスクは Application Server が提供する管理ツールを使用して実行します。

管理用ツール

Application Server では、次の管理ツールおよび API を使用できます。

- 34 ページの「管理コンソール」
- 35 ページの「コマンド行インタフェース (asadmin ユーティリティー)」
- 36 ページの「JConsole」
- 36 ページの「Application Server Management Extension (AMX)」

管理コンソール

管理コンソールは、ナビゲートしやすいインタフェースとオンラインヘルプを装備したブラウザベースのツールです。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わかっていなければなりません。ポート番号については、Application Server のインストール時に選択したポート番号か、デフォルトポートの 4848 が使用されます。また、ユーザー名とマスターパスワードについても、インストール時に指定したものが必要です。

管理コンソールを起動するには、Web ブラウザで次のように入力します。

```
http://hostname:port
```

次に例を示します。

```
http://kindness.sun.com:4848
```

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

Windows 環境で管理コンソールを起動するには、「スタート」メニューから「管理コンソール」を起動します。

インストールプログラムにより、`domain1` という名前のデフォルト管理ドメインがデフォルトポート番号 4848 で生成されます。また、ドメイン管理サーバー (DAS) とは分離したインスタンスも作成されます。インストール後は、管理ドメインを作成して追加できます。各ドメインには、一意のポート番号を持ったドメイン管理サーバーがそれぞれ必要です。管理コンソールの URL を指定する場合は、ドメインの管理ポート番号も指定する必要があります。

DAS とは異なるサーバー上で稼動するリモートサーバーインスタンスを管理する場合は、ノードエージェントを作成し、リモートサーバーインスタンスを容易に管理できるようにします。サーバーインスタンスの作成、起動、停止、および削除は、ノードエージェントの役割です。ノードエージェントを設定するには、コマンド行インタフェース (CLI) のコマンドを使用します。

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。`asadmin` ユーティリティーと、このユーティリティーに関連するコマンドを使用して、管理コンソールで提供されている一連の同じ管理タスクを実行します。Solaris でのデフォルトのインストール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、*as-install*/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リスト表示するには、次のように入力します。

```
asadmin> help
```

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

```
$ asadmin help
```

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

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

指定したコマンドの asadmin help 情報が、UNIX のマニュアルページの形式で表示されます。これらのマニュアルページは、HTML 形式または PDF 形式の『Sun Java System Application Server 9.1 Reference Manual』でも参照できます。

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) を指定し、「接続」を選択します。
- 「詳細」タブ: JMXServiceURL のサービス URL (jmx:rmi:///jndi/rmi://host:jms-port/jmxrmi) を指定し、「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 (AMX) は、すべての Application Server 設定を表示する API です。また、AMX は 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 9.1 Developer's Guide』の第 20 章「Using the Application Server Management Extensions」を参照してください。

Application Server のコマンドと概念

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。管理上の境界線であることに加え、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、さまざまな組織や管理者が 1 つの Application Server インストールを共有できます。各ドメインには、固有の設定、ログファイル、およびアプリケーションの配備領域があり、これらはほかのドメインとは無関係です。1 つのドメインの設定が変更されても、ほかのドメインの設定は影響を受けません。

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。インストール時には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 4848 です。インストーラは管理ユーザー名とマスター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) は管理対象アプリケーションの制御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付け、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、クラスタへの操作 (アプリケーションの配備など) は、そのクラスタを構成するすべてのインスタンスに反映されます。

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 ランタイムのみで稼動できるなど) 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、単一のノードの Application Server 上で稼動する単一の Java EE 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。次の図は、アプリケーションサーバーインスタンスの詳細を示しています。アプリケーションサーバーインスタンスは、Application Server のクラスタリング、ロードバランス、セッションの持続性といった各機能の基本を構成するものです。

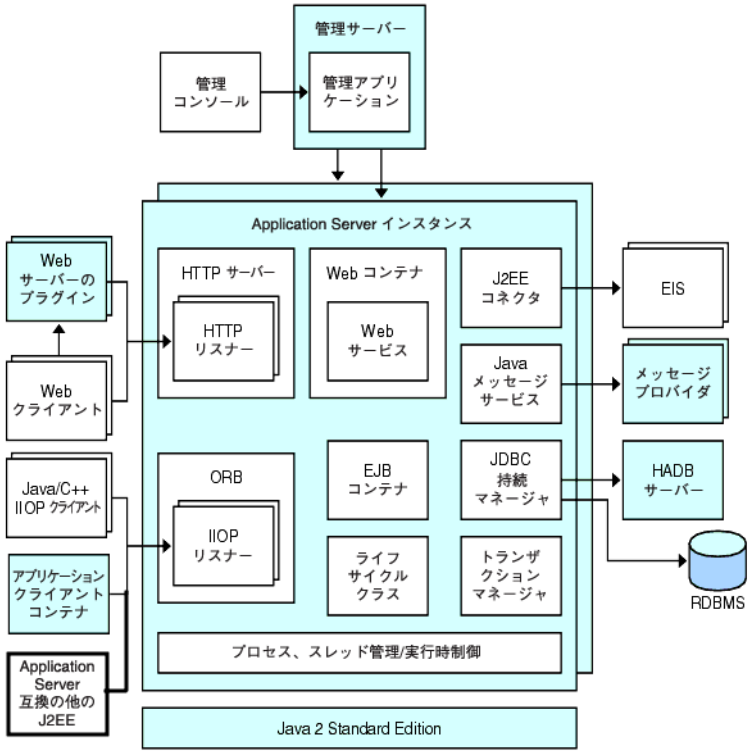

図 1–2　Application Server インスタンス

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、要件は満たされるでしょう。ただし、環境によっては、1 つ以上の追加のアプリケーションサーバーインスタンスを作成する場合があります。たとえば、開発環境で、異なるアプリケーションサーバーインスタンスがあれば、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したりすることができます。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、企業または個人のドメイン名、IP アドレス、いくつかの管理機能を提供できます。ユーザーにとっては、ハードウェアを持つことも、サーバーの基本的な保守を行うこともなく、自分の Web サーバーを所有しているのとほぼ同じです。このような仮想サーバーは、複数のアプリケーションサーバーインスタンスにまたがりません。仮想サーバーの詳細については、第 13 章HTTP サービスの設定を参照してください。

運用上において、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、処理中の要求が失われる可能性があります。

Application Server インスタンスの定義

アプリケーションサーバーインスタンスは、アプリケーション実行環境の基本単位となります。各インスタンスは 1 つのドメインに属し、それぞれに固有のディレクトリ構造、設定、および配備されたアプリケーションが含まれます。各サーバーインスタンスには、Java EE プラットフォームの Web および EJB コンテナも含まれます。新しいサーバーインスタンスは、そのインスタンスが置かれるマシン上で稼動するノードエージェントと関連付ける必要があります。

注 – 開発者プロファイルを持つドメインでは Application Server インスタンスを作成できません。開発者プロファイルを持つドメインに含めることのできるインスタンスは、デフォルトで生成される `server1` のみです。複数のインスタンスを作成するには、クラスタプロファイルでドメインを作成する必要があります。ドメインの作成については、`create-domain` コマンドのマニュアルページを参照するか、また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。

サーバーインスタンスには次の 3 つのタイプがあります。

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

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

サーバーの一般情報

開発者プロファイルを持つドメインの場合、管理コンソールの「アプリケーションサーバー」パネル内の「一般」タブからは、次のタスクを実行できます。

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

  この操作ではログファイルのローテーションは予約されるのみで、実際のローテーションは、次にログファイルにエントリが書き込まれたときに行われます。ローテーションは、デフォルトのサーバー (DAS) に対してはただちに実行されますが、ほかのスタンドアロンサーバーに対しては遅れます。
- 「JNDI ブラウズ」をクリックして、実行中のインスタンスの JNDI ツリーをブラウズします。
- 「トランザクションの回復」をクリックして、未完了のトランザクションを回復します。

さらに、次のタブを選択して、追加のタスクを実行できます。

- 「アプリケーション」タブ: 選択したアプリケーションを配備します。
- 「JVM 設定」タブ: Application Server で使用する JVM 一般設定を設定します。
- 「リソース」タブ: 選択したリソースを管理します。
- 「プロパティー」タブ: インスタンス固有のプロパティーを設定します。
- 「ログ」タブ: Application Server で使用するログレベルを設定します。
- 「監視」タブ: JVM、サーバー、スレッドプール、HTTP サービス、トランザクションサービスの監視データを表示します。
- 「詳細」タブ: 配備するアプリケーション用の一般プロパティーを設定します。

注 – 開発者プロファイルで管理コンソールを実行している場合は、「インスタンスを起動」オプション、「リソース」および「プロパティー」のタブを使用できません。

Application Server 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 5000 で、管理ユーザー名は `admin` です。このコマンドは、管理パスワードおよびマスターパスワードの入力を求めます。

```
$ asadmin create-domain --adminport 5000 –user admin 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*`:5000`

Application Server 9.1 では、各ドメインはそれぞれがプロファイルを持ちます。プロファイルについては、29 ページの「使用法プロファイル」を参照してください。ドメインのプロファイルは、作成時に選択することができます。また、「一般」タブの「クラスタサポートを追加」をクリックすると、開発者プロファイルからクラスタプロファイルに変更できます。ドメインのプロファイルを指定するには、`create-domain` コマンドで `--profile` オプションを使用します。`--profile` オプションを使用してプロファイルを明示的に指定しない場合、そのドメインにはデフォルトのプロファイルが関連付けられます。デフォルトのプロファイルは、`asadminenv.conf` ファイル内の `AS_ADMIN_PROFILE` で定義されます。

注意 – HADB およびネットワークセキュリティーサービス (NSS) キーストアがない場合は、エンタープライズドメインを作成しないでください。HADB および NSS がないと、エンタープライズドメインを起動できません。

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。この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。パスワードオプションを省略した場合は、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 9.1」>「デフォルトサーバーを起動」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、ドメイン名を省略できます。コマンドの完全な構文を確認するには、`asadmin help stop-domain` と入力してください。

管理コンソールからドメインを停止するには、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。

Windows でのデフォルトドメインの停止

「スタート」メニューで、「プログラム」>「Sun Microsystems」>「Application Server 9.1」>「デフォルトサーバー停止」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
```

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、通常は管理パスワードの入力を求めます。ただし、--savemasterpassword オプションが指定されていないか、または false の場合、このコマンドはマスターパスワードの入力を求めません。

```
$ 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 という名前のサーバー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 (machine3)

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。現在のドメインをバックアップするには、`asadmin backup-domain` を使用します。

▼ 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。対話型コマンド行モードを有効にするには、次のように `console` オプションを使用してインストールプログラムを呼び出します。

./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 *as-install* 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、ドメインの再作成時には、ドメイン名を元のドメイン名と同一にする必要があります。

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

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

次に例を示します。

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

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

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

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

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

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

変更後:

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

7 変更前:

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

変更後:

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

8 **machine3** で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 *as-install*/nodeagents/*nodeagent*/agent/config/das.properties で、agent.das.host プロパティー値を変更します。

11 **machine2** のノードエージェントを再起動します。

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

Application Server 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。

- 49 ページの「Application Server 設定の変更」
- 50 ページの「Application Server のポート」

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
- 次の JDBC 接続プールプロパティーの変更
  - datasource-classname
  - JDBC ドライバベンダー固有のプロパティー
  - associate-with-thread
  - lazy-connection-association
  - lazy-connection-enlistment
- 次のコネクタ接続プールプロパティーの変更
  - resource-adapter-name
  - connection-definition-name
  - transaction-support
  - ベンダー固有のプロパティー
  - associate-with-thread
  - lazy-connection-association
  - lazy-connection-enlistment

サーバーの再起動の手順については、44 ページの「ドメインの再起動」を参照してください。

その他の設定変更はサーバーを稼動させたままで動的に有効化されます。次の設定変更を行う場合は、サーバーを再起動させ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 1–2 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	4848	ドメイン管理サーバーには、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。管理コンソールには、ブラウザの URL にポート番号を指定します。リモートから `asadmin` コマンドを実行する場合は、`--port` オプションを使用してポート番号を指定します。
HTTP	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	セキュリティー保護された通信用に設定された Web アプリケーションは、個別のポートで待機します。
IIOP		EJB コンポーネントであるエンタープライズ Beans のリモートクライアントは IIOP リスナー経由で Beans にアクセスします。
IIOP_SSL		セキュリティー保護された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_MUTUALAUTH		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。

◆ ◆ ◆ 第 2 章

Java Business Integration

Java Business Integration (JBI) は、Java Community Process (JCP) の下でサービス指向アーキテクチャー (SOA) を実装する方法として開発された標準である「JSR 208 specification for Java Business Integration (`http://www.jcp.org/en/jsr/detail?id=208`)」の実装です。

JBI は、Web Services Description Language (WSDL) 2.0 に直接基づいたサービスモデルを使用して相互に作用するプラグインコンポーネントのための環境を定義します。これらのプラグインコンポーネントは、サービスプロバイダ、サービスコンシューマ、またはその両方として機能します。

JBI 実行環境の主要コンポーネントおよびそれらのライフサイクル状態の管理の詳細については、Application Server 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。JBI コマンドの使用については、『Sun Java System Application Server 9.1 Reference Manual』を参照してください。

JBI 環境

次の各節で、JBI 環境の主要コンポーネントについて説明します。

JBI コンポーネント

サービスエンジン

サービスエンジンは、ローカルサービス (つまり、JBI 環境内部のサービス) を提供し、ローカルまたはリモートサービスを消費するコンポーネントです。

バインディングコンポーネント

バインディングコンポーネントは、JBI 環境の外部にあるコンシューマまたはプロバイダのプロキシです。バインディングコンポーネントは、通常、FTP、JMS、SMTP などの標準的な通信プロトコル、または SAP や webSphereMQ などの外部サービスの呼び出しに基づいています。

JBI コンポーネントには、次のライフサイクル状態があります。

- 起動済み
- 停止済み
- シャットダウン

JBI ランタイムは、JBI コンポーネントのライフサイクル状態を持続します。アプリケーションサーバーがシャットダウンしてから再起動すると、JBI コンポーネントはアプリケーションサーバーがシャットダウンした時点の状態に戻ります。

注 – JBI ランタイムは、JBI コンポーネントの「適切な」状態に戻ろうとします。たとえば、JBI コンポーネントを起動しようとしたが、コンポーネント内のエラーのために起動しなかったとします。Application Server を再起動すると、JBI ランタイムはコンポーネントの起動を再試行します。

JBI コンポーネントに対して次の操作を実行できます。詳細な手順については、管理コンソールにログオンし、JBI ノードに移動して、「コンポーネント」をクリックし、次に「ヘルプ」をクリックしてください。

- 特定のライフサイクル状態ごとに JBI コンポーネントを表示する。
- JBI コンポーネントをインストールする。
- JBI コンポーネントをアンインストールする。
- JBI コンポーネントのライフサイクル状態を管理する。
- JBI コンポーネントの一般プロパティーを表示する。
- JBI コンポーネントの設定情報を表示する。
- JBI コンポーネントの記述子を表示する。
- JBI コンポーネントのロガーを管理する。

JBI コンポーネントロガー

Application Server 管理コンソールを使用して、JBI コンポーネントのログ レベルを管理できます。JBI コンポーネントの中には、複数のロガーを提供するものや、ロガーを提供しないものもありますが、コンポーネント全体では常に 1 つのロガーレベルが表示されます。ただし、ロガーレベルの設定が有効になるのは、コンポーネントがデフォルトの名前に基づいてロガーを実装している場合だけです。ログレベルの指定に関しては、JBI コンポーネントのプロバイダから提供されるドキュメントもあわせて参照してください。

注 – JBI コンポーネントのログレベルは、多くの場合、JBI ロガーなどの親ロガーから継承されます。親のログレベルを表示および設定するには、開発者プロファイルを持つドメインの場合、管理コンソールで「アプリケーションサーバー」パネルを選択します。次に、「ログ」タブを選択し、「ログレベル」タブを選択します。JBI モジュールのドロップダウンリストで親の JBI ログレベルの表示および設定を行います。

サービスアセンブリ

サービスアセンブリは、アプリケーションのために特定のサービスを共同で提供または消費するターゲットコンポーネントをプロビジョニングするサービスユニットの集合です。通常、サービスアセンブリは開発ツール環境 (NetBeans Enterprise Pack で提供される環境など) で作成されます。

サービスアセンブリには、次のライフサイクル状態があります。

- 起動済み
- シャットダウン
- 停止済み

JBI ランタイムは、サービスアセンブリのライフサイクル状態を持続します。アプリケーションサーバーがシャットダウンしてから再起動すると、サービスアセンブリはアプリケーションサーバーがシャットダウンした時点の状態に戻ります。

注 – JBI ランタイムは、サービスアセンブリの「適切な」状態に戻ろうとします。たとえば、サービスアセンブリを起動しようとしたが、サービスアセンブリ内のエラーのために起動しなかったとします。Application Server を再起動すると、JBI ランタイムはサービスアセンブリの起動を再試行します。

サービスアセンブリに対して次の操作を実行できます。詳細な手順については、管理コンソールにログオンし、JBI ノードに移動して、「サービスアセンブリ」をクリックし、次に「ヘルプ」をクリックしてください。

- サービスアセンブリを表示する。ライフサイクル状態によるソートとフィルタリングがサポートされます。
- サービスアセンブリを配備する。
- サービスアセンブリの配備を取り消す。
- サービスアセンブリのライフサイクルを管理する。
- サービスアセンブリの一般プロパティーを表示する。
- サービスアセンブリの記述子を表示する。

共用ライブラリ

共用ライブラリは、1 つのコンポーネントに占有されない Java クラスを提供し、通常は複数の JBI コンポーネントによって共有されます。たとえば、Java EE サービスエンジンには WSDL 共用ライブラリが必要です。

共用ライブラリに対して次の操作を実行できます。詳細な手順については、管理コンソールにログオンし、JBI ノードに移動して、「共用ライブラリ」をクリックし、次に「ヘルプ」をクリックしてください。

- すべての共用ライブラリを表示する。
- 共用ライブラリをインストールする。
- 共用ライブラリの一般プロパティーを表示する。
- 共用ライブラリの記述子を表示する。
- 共用ライブラリをアンインストールする。

JBI 記述子

サービスアセンブリ、JBI コンポーネント、および共用ライブラリの記述子ファイル (`jbi.xml`) は、次の情報を提供します。

- サービスアセンブリ: サービスアセンブリに含まれるサービスユニットと各サービスユニットのターゲットを示すリスト。サービスユニットによっては、接続エンドポイントに関する情報が示される場合もあります。
- JBI コンポーネント: JBI コンポーネントのタイプ (バインディングコンポーネントまたはサービスエンジン)、コンポーネントの説明、コンポーネントに関連するクラスパスの情報、およびコンポーネントが依存する共用ライブラリの名前を示すリスト。
- 共用ライブラリ: 共用ライブラリの名前と、共用ライブラリを格納するアーカイブファイル (`.jar` ファイル) またはクラスファイルサブディレクトリの名前を示すリスト。

◆ ◆ ◆ 第 3 章

JDBC リソース

この章では、データベースにアクセスするアプリケーションで必要な、JDBC リソースの設定方法について説明します。この章で説明する内容は次のとおりです。

JDBC リソース

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

JDBC リソース (データソース) は、アプリケーションにデータベースへ接続する手段を提供します。通常、管理者は、ドメインに配備されたアプリケーションがアクセスする各データベースの JDBC リソースを作成します。ただし、1 つのデータベースに複数の JDBC リソースを作成できます。

JDBC リソースを作成するには、リソースを識別する一意の JNDI 名を指定します。「JNDI 名とリソース」の節を参照してください。通常、JDBC リソースの JNDI 名は `java:comp/env/jdbc` サブコンテキストにあります。たとえば、給与データベースのリソースの JNDI 名は `java:comp/env/jdbc/payrolldb` と名付けることができます。すべてのリソース JNDI 名は `java:comp/env` サブコンテキストにあるので、管理コンソールの JDBC リソースにある JNDI 名を指定するときは、`jdbc/`*name* だけを入力します。たとえば、給与データベースには `jdbc/payrolldb` を指定します。

JDBC 接続プール

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

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

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

関連項目

- 55 ページの「JDBC リソース」
- 56 ページの「JDBC リソースと接続プールの協調動作について」
- 58 ページの「JDBC 接続プールの編集」

JDBC リソースと接続プールの協調動作について

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

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

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

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

   バックグラウンドで、アプリケーションサーバーはデータベースに対応した接続プールから物理接続を取得します。プールは、データベース名 (URL)、ユーザー名、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、次の手順に従います。

1. サポートされたデータベース製品をインストールします。

   Application Server がサポートするデータベース製品のリストについては、リリースノートを参照してください。
2. データベース製品の JDBC ドライバをインストールします。
3. ドメインのサーバーインスタンスにアクセスできるドライバの JAR ファイルを作成します。
4. データベースを作成します。

   通常、アプリケーションプロバイダがデータベースを作成し移行するためのスクリプトを配信します。
5. データベースの接続プールを作成します。
6. 接続プールを示す JDBC リソースを作成します。

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

1. 共通クラスローダーにアクセスできるドライバを作成します。

   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。
3. ドライバの JAR ファイルの完全修飾パス名が認識されます。

JDBC 接続プールについて

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

JDBC 接続プールの編集

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

1. 一般設定を変更します。

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

パラメータ	説明
データソースクラス名	`DataSource` API または `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。

2. プール設定を変更します。

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

パラメータ	説明
初期および最小プールサイズ	プール内の接続の最小数。この値は、プールを最初に作成したり、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	プールのサイズを最大プールサイズおよび最小プールサイズに対して拡大および縮小すると、一括処理でプールサイズが変更されます。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作成と再利用が遅れ、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	接続を要求するアプリケーションが接続タイムアウトになるまでに待つ時間。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。

3. 接続検証設定を変更します。

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

パラメータ	説明
接続検証	「必須」チェックボックスにチェックを付けて、接続検証を有効にします。

パラメータ	説明
検証方法	アプリケーションサーバーは、次の 3 つの方法でデータベース接続を検証できます。auto-commit、metadata、および table。 auto-commit と meta-data - アプリケーションサーバーは、`con.getAutoCommit()` メソッドと `con.getMetaData()` メソッドを呼び出して接続を検証します。 注 – 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、すでに頻繁にアクセスされている表を使用しないでください。
テーブル名	「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のテーブル名を指定します。
すべての障害で	「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。
コンポーネントの呼び出し側以外を許可	サーブレットフィルタやライフサイクルモジュールなど、コンポーネント以外の呼び出し側が使用するためのプールを有効にする場合は、このチェックボックスにチェックマークを付けます。

4. トランザクション設定を変更します。

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

パラメータ	説明
非トランザクション接続	Application Server がすべての非トランザクション接続を返すようにする場合は、このチェックボックスにチェックマークを付けます。

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るようにします。指定しない場合、接続には 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と、プールから取得されるすべての接続が同じ遮断レベルになります。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の遮断レベルに戻されます。

5. プロパティーを変更します。

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

JDBC 接続プールの詳細属性の編集

接続リークを診断しやすくし、使い勝手を向上させるため、Application Server 9.1 には接続プールをその作成時点で設定するための新しい属性がいくつか用意されています。

1. 「詳細」タブを開いて、次の属性を指定します。

属性	説明
名前	プロパティーを編集する JDBC 接続プールの名前。ただし、プール名は変更できません。
文のタイムアウト	異常に長い時間実行されているクエリーを終了させるまでの時間 (秒数)。Application Server は、作成された文に対して「QueryTimeout」を設定します。デフォルト値の -1 は、この属性が無効であることを示します。
JDBC オブジェクトをラップ	true に設定すると、アプリケーションは Statement、PreparedStatement、CallableStatement、ResultSet、DatabaseMetaData に対してラップされた JDBC オブジェクトを取得します。デフォルト値は false です。

2. 次の表の説明に従って、接続設定を指定します。

属性	説明
最大で 1 回検証	経過するまでに 1 回しか接続が検証されない時間 (秒)。これは、1 つの接続による検証要求の数を減らすのに役立ちます。デフォルト値の 0 は、接続の検証が無効であることを示します。
リークタイムアウト	接続プール内の接続リークを追跡する時間 (秒)。デフォルト値の 0 は、接続リーク追跡が無効であることを示します。接続リーク追跡を有効にすると、「リソースの監視」タブに接続リークの数に関する統計情報が表示されます。このタブを表示するには、「Application Server」→「監視」→「リソース」の順に選択します。
リーク再要求	このオプションを有効にすると、リーク接続の追跡が完了したあとで、リークした接続がプールにリストアされます。
作成再試行回数	新しい接続の作成に失敗した場合に行われる試行の回数。デフォルト値の 0 は、接続の再試行が行われないことを示します。
再試行間隔	接続作成の試行の間隔 (秒数) を指定します。デフォルト値は 10 秒です。この属性は、「作成再試行回数」が 1 以上の場合にのみ使用されます。
使用時にのみ接続を登録	リソースが実際にメソッドで使用される場合にのみリソースをトランザクションに登録するには、このオプションを有効にします。
使用時にのみ関連付け	接続に対して操作が実行された時点ではじめて接続が関連付けられます。また、トランザクションが完了してコンポーネントメソッドが終了したときに関連付けが解除されるため、物理的な接続を再利用しやすくなります。デフォルト値は false です。
スレッドとの関連付け	接続をスレッドに関連付けるには、このオプションを有効にします。これにより、同じスレッドが接続を必要とするときは、そのスレッドにすでに関連付けられている接続を再利用できるため、プールから接続を取得するオーバーヘッドが発生しません。デフォルト値は false です。

接続のマッチング	プールに対する接続マッチングのオン/オフを切り替えるには、このオプションを使用します。プール内の接続が常に同じ種類であり、プールから選択した接続とリソースアダプタを照合する必要がない場合は、false に設定できます。デフォルト値は false です。
最大接続使用数	接続をプールで再利用する回数を指定します。指定された回数だけ再利用されると、その接続は閉じられます。これは、たとえば、文リークを回避するのに役立ちます。デフォルト値の 0 は、接続が再利用されないことを示します。

JDBC リソースについて

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ときには、次の項目を指定してください。

1. JNDI 名。規則により、名前は `jdbc/` 文字列で始まります。次に例を示します。`jdbc/payrolldb`。スラッシュを忘れないでください。
2. 新しい JDBC リソースに関連付ける接続プールを選択します。
3. リソースの設定を指定します。
4. クラスタプロファイルまたはエンタープライズプロファイルの場合、リソースが利用できるターゲット (クラスタおよびスタンドアロンサーバーインスタンス) を指定します。

各 JDBC ドライバの設定

Application Server 9.1 は、対応する JDBC ドライバを持つすべてのデータベース管理システムに接続できるように設計されています。次の JDBC ドライバとデータベースの組み合わせがサポートされます。これらの組み合わせは、Application Server 9.1 でテスト済みであり、J2EE と互換性があることが確認されています。これらは CMP でもサポートされます。

- 64 ページの「Derby Type 4 ドライバ」
- 65 ページの「DB2 データベース用の Sun Java System JDBC ドライバ」
- 66 ページの「Oracle 8.1.7 および 9.x データベース用の Sun Java System JDBC ドライバ」
- 66 ページの「Microsoft SQL Server データベース用の Sun Java System JDBC ドライバ」

- 67 ページの「Sybase データベース用の Sun Java System JDBC ドライバ」
- 67 ページの「IBM DB2 8.1 Type 2 ドライバ」
- 68 ページの「Sybase ASE 12.5 データベース用の JConnect Type 4 ドライバ」
- 68 ページの「MM MySQL Type 4 ドライバ (非 XA)」

現在サポートされている JDBC ドライバの最新リストについては、『Sun Java System Application Server 9.1 リリースノート』を参照してください。

その他の JDBC ドライバは、Application Server 9.1 で使用可能ですが、J2EE との互換性テストはまだ完了していません。Sun では、それらのドライバの製品サポートは提供していませんが、Application Server 9.1 での使用についての限定サポートを提供しています。

- 69 ページの「MM MySQL Type 4 ドライバ (XA のみ)」
- 70 ページの「Oracle 8.1.7 および 9.x データベース用の Inet Oraxo JDBC ドライバ」
- 71 ページの「Microsoft SQL Server データベース用の Inet Merlia JDBC ドライバ」
- 71 ページの「Sybase データベース用の Inet Sybelux JDBC ドライバ」
- 72 ページの「Oracle 8.1.7 および 9.x 用の Oracle Thin Type 4 ドライバ」
- 73 ページの「Oracle 8.1.7 および 9.x データベース用の OCI Oracle Type 2 ドライバ」
- 74 ページの「IBM Informix Type 4 ドライバ」
- 74 ページの「CloudScape 5.1 Type 4 ドライバ」

JDBC ドライバの統合方法、および管理コンソールまたはコマンド行インタフェースを使って設定を実装する方法の詳細については、『Sun Java System Application Server 9.1 管理ガイド』を参照してください。

注 – Oracle データベースユーザーが `capture-schema` コマンドを実行するには、そのユーザーがスキーマの所有者でないかぎり、ANALYZE ANY TABLE 特権が必要です。この特権は、データベース管理者がユーザーに付与します。`capture-schema` については、『Sun Java System Application Server 9.1 Reference Manual』を参照してください。

Derby Type 4 ドライバ

Derby JDBC ドライバは、デフォルトで Application Server に含まれています。ただし、Solaris バンドル版のインストールには Derby は含まれていません。したがって、Solaris バンドル版のインストール以外では、この JDBC ドライバを Application Server に組み込む必要はありません。

この Derby ドライバの JAR ファイルは `derbyclient.jar` です。

次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。

- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `Derby`
- 「データソースクラス名」: 次のいずれかを指定します。

```
org.apache.derby.jdbc.ClientDataSource
org.apache.derby.jdbc.ClientXADataSource
```

- 「プロパティー」:
  - **user** - データベースユーザーを指定します。

    これは、Derby が認証を使用するように設定されている場合にのみ必要です。Derby は、デフォルトでは認証を使用しません。ユーザーを指定すると、その名前が、テーブルが属するスキーマの名前になります。
  - **password** - データベースパスワードを指定します。

    これは、Derby が認証を使用するように設定されている場合にのみ必要です。
  - **databaseName** - データベースの名前を指定します。
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します (デフォルトと異なる場合のみ)。
- 「**URL**」: `jdbc:derby://` *serverName*`:` *portNumber*`/`*databaseName* `;create=true`

  `;create=true` の部分は、データベースが存在していない場合にそのデータベースを作成する場合にのみ含めます。

DB2 データベース用の Sun Java System JDBC ドライバ

このドライバの JAR ファイルは `smbase.jar`、`smdb2.jar`、および `smutil.jar` です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `DB2`
- 「データソースクラス名」: `com.sun.sql.jdbcx.db2.DB2DataSource`
- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します。
  - **databaseName** - 必要に応じて設定します。
  - **user** - 必要に応じて設定します。

- **password** - 必要に応じて設定します。
- 「**URL**」: `jdbc:sun:db2://` *serverName*`:` *portNumber*`;databaseName=`*databaseName*

Oracle 8.1.7 および 9.x データベース用の Sun Java System JDBC ドライバ

このドライバの JAR ファイルは `smbase.jar`、`smoracle.jar`、および `smutil.jar` です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `Oracle`
- 「データソースクラス名」: `com.sun.sql.jdbcx.oracle.OracleDataSource`
- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します。
  - **SID** - 必要に応じて設定します。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
- 「**URL**」: `jdbc:sun:oracle://` *serverName*`[:` *portNumber*`][;SID=`*databaseName* `]`

Microsoft SQL Server データベース用の Sun Java System JDBC ドライバ

このドライバの JAR ファイルは `smbase.jar`、`smsqlserver.jar`、および `smutil.jar` です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `Microsoft SQL Server`
- 「データソースクラス名」: `com.sun.sql.jdbcx.sqlserver.SQLServerDataSource`
- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。

- **selectMethod** - cursor に設定します。
- 「**URL**」: jdbc:sun:sqlserver:// *serverName*[: *portNumber*]

Sybase データベース用の Sun Java System JDBC ドライバ

このドライバの JAR ファイルは smbase.jar、smsybase.jar、および smutil.jar です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: Sybase
- 「データソースクラス名」: com.sun.sql.jdbcx.sybase.SybaseDataSource
- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します。
  - **databaseName** - 必要に応じて設定します。これは任意指定です。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
- 「**URL**」: jdbc:sun:sybase:// *serverName*[: *portNumber*]

IBM DB2 8.1 Type 2 ドライバ

この DB2 ドライバの JAR ファイルは db2jcc.jar、db2jcc_license_cu.jar、および db2java.zip です。次のように環境変数を設定します。

```
LD_LIBRARY_PATH=/usr/db2user/sqllib/lib:${j2ee.home}/lib
DB2DIR=/opt/IBM/db2/V8.1
DB2INSTANCE=db2user
INSTHOME=/usr/db2user
VWSPATH=/usr/db2user/sqllib
THREADS_FLAG=native
```

次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: DB2
- 「データソースクラス名」: com.ibm.db2.jcc.DB2SimpleDataSource

- 「プロパティー」:
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
  - **databaseName** - 必要に応じて設定します。
  - **driverType** - 2 に設定します。
  - **deferPrepares** - false に設定します。

Sybase ASE 12.5 データベース用の JConnect Type 4 ドライバ

この Sybase ドライバの JAR ファイルは jconn2.jar です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: Sybase
- 「データソースクラス名」: 次のいずれかを指定します。

```
com.sybase.jdbc2.jdbc.SybDataSource
com.sybase.jdbc2.jdbc.SybXADataSource
```

- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
  - **databaseName** - 必要に応じて設定します。完全な URL ではなく、データベース名のみを指定してください。
  - **BE_AS_JDBC_COMPLIANT_AS_POSSIBLE** - true に設定します。
  - **FAKE_METADATA** - true に設定します。

MM MySQL Type 4 ドライバ (非 XA)

この MySQL ドライバの JAR ファイルは mysql-connector-java-*version*-bin-g.jar です。たとえば、mysql-connector-java-3.1.12-bin-g.jar などです。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。

- 「データベースベンダー」: MySQL
- 「データソースクラス名」: 次のように指定します。

  ```
  com.mysql.jdbc.jdbc2.optional.MysqlDataSource
  ```

- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **port** - データベースサーバーのポート番号を指定します。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
  - **databaseName** - 必要に応じて設定します。
  - **URL** - グローバルトランザクションを使用する場合は、serverName、port、および databaseName の代わりにこのプロパティーを設定できます。

    MM MySQL Type 4 ドライバには、必須の relaxAutoCommit プロパティーを設定する手段が用意されていないので、**URL** プロパティーを次のように設定することにより間接的に設定してください。

    ```
    jdbc:mysql://host:port/database?relaxAutoCommit="true"
    ```

MM MySQL Type 4 ドライバ (XA のみ)

この MySQL ドライバの JAR ファイルは mysql-connector-java-*version*-bin-g.jar です。たとえば、mysql-connector-java-3.1.12-bin-g.jar などです。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: MySQL
- 「データソースクラス名」: 次のように指定します。

  ```
  com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
  ```

- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **port** - データベースサーバーのポート番号を指定します。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
  - **databaseName** - 必要に応じて設定します。

- **URL** - グローバルトランザクションを使用する場合は、serverName、port、および databaseName の代わりにこのプロパティーを設定できます。

  MM MySQL Type 4 ドライバには、必須の relaxAutoCommit プロパティーを設定する手段が用意されていないので、**URL** プロパティーを次のように設定することにより間接的に設定してください。

  ```
  jdbc:mysql://host:port/database?relaxAutoCommit="true"
  ```

Oracle 8.1.7 および 9.x データベース用の Inet Oraxo JDBC ドライバ

この Oracle ドライバの JAR ファイルは Oranxo.jar です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: Oracle
- 「データソースクラス名」: com.inet.ora.OraDataSource
- 「プロパティー」:
  - **user** - データベースユーザーを指定します。
  - **password** - データベースパスワードを指定します。
  - **serviceName** - データベースの URL を指定します。構文は次のとおりです。

    ```
    jdbc:inetora:server:port:dbname
    ```

    次に例を示します。

    ```
    jdbc:inetora:localhost:1521:payrolldb
    ```

    この例では、localhost は Oracle サーバーを実行しているマシンのホスト名、1521 は Oracle サーバーのポート番号、payrolldb はデータベースの SID です。データベース URL の構文の詳細については、Oracle のドキュメントを参照してください。
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **port** - データベースサーバーのポート番号を指定します。
  - **streamstolob** - BLOB または CLOB データ型のサイズが 4K バイトを超え、このドライバが CMP で使用される場合は、このプロパティーを true に設定してください。

- **xa-driver-does-not-support-non-tx-operations** - true の値に設定します。これは任意指定です。同じ接続プールから非 XA 接続と XA 接続の両方が取得される場合にのみ必要です。パフォーマンスが低下する可能性があります。

  このプロパティーを設定する代わりに、非 XA 接続用とXA 接続用の 2 つの接続プールを作成することもできます。

Microsoft SQL Server データベース用の Inet Merlia JDBC ドライバ

この Microsoft SQL Server ドライバの JAR ファイルは Merlia.jar です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: Microsoft SQL Server
- 「データソースクラス名」: com.inet.tds.TdsDataSource
- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **port** - データベースサーバーのポート番号を指定します。
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。

Sybase データベース用の Inet Sybelux JDBC ドライバ

この Inet Sybase ドライバの JAR ファイルは Sybelux.jar です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: Sybase
- 「データソースクラス名」: com.inet.syb.SybDataSource
- 「プロパティー」:
  - **serverName** - データベースサーバーのホスト名または IP アドレスを指定します。
  - **portNumber** - データベースサーバーのポート番号を指定します。
  - **user** - 必要に応じて設定します。

- **password** - 必要に応じて設定します。
- **databaseName** - 必要に応じて設定します。完全な URL ではなく、データベース名のみを指定してください。

Oracle 8.1.7 および 9.x 用の Oracle Thin Type 4 ドライバ

この Oracle ドライバの JAR ファイルは `ojdbc14.jar` です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `Oracle`
- 「データソースクラス名」: 次のいずれかを指定します。

```
oracle.jdbc.pool.OracleDataSource
oracle.jdbc.xa.client.OracleXADataSource
```

- 「プロパティー」:
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
  - **URL** - 次の構文を使用して完全なデータベース URL を指定します。

    ```
    jdbc:oracle:thin:[user/password]@host[:port]/service
    ```

    次に例を示します。

    ```
    jdbc:oracle:thin:@localhost:1521:customer_db
    ```

  - **xa-driver-does-not-support-non-tx-operations** - `true` の値に設定します。これは任意指定です。同じ接続プールから非 XA 接続と XA 接続の両方が取得される場合にのみ必要です。パフォーマンスが低下する可能性があります。

    このプロパティーを設定する代わりに、非 XA 接続用と XA 接続用の 2 つの接続プールを作成することもできます。

注– グローバルトランザクションの復旧が正しく動作するためには、トランザクションサービスで oracle-xa-recovery-workaround プロパティーを設定する必要があります。詳細については、153 ページの「特定のデータベースに関する問題の回避方法」を参照してください。

このドライバを使用する場合、2000 バイトを超えるデータを列に挿入することはできません。この問題を回避するには、OCI ドライバ (JDBC Type 2) を使用します。

Oracle 8.1.7 および 9.x データベース用の OCI Oracle Type 2 ドライバ

この OCI Oracle ドライバの JAR ファイルは ojdbc14.jar です。LD_LIBRARY_PATH を介して共用ライブラリが利用可能であること、および ORACLE_HOME プロパティーが設定されていることを確認してください。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: Oracle
- 「データソースクラス名」: 次のいずれかを指定します。

```
oracle.jdbc.pool.OracleDataSource
oracle.jdbc.xa.client.OracleXADataSource
```

- 「プロパティー」:
  - **user** - 必要に応じて設定します。
  - **password** - 必要に応じて設定します。
  - **URL** - 次の構文を使用して完全なデータベース URL を指定します。

    ```
    jdbc:oracle:oci:[user/password]@host[:port]/service
    ```

    次に例を示します。

    ```
    jdbc:oracle:oci:@localhost:1521:customer_db
    ```

  - **xa-driver-does-not-support-non-tx-operations** - true の値に設定します。これは任意指定です。同じ接続プールから非 XA 接続と XA 接続の両方が取得される場合にのみ必要です。パフォーマンスが低下する可能性があります。

    このプロパティーを設定する代わりに、非 XA 接続用と XA 接続用の 2 つの接続プールを作成することもできます。

IBM Informix Type 4 ドライバ

次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `Informix`
- 「データソースクラス名」: 次のいずれかを指定します。

```
com.informix.jdbcx.IfxDataSource
com.informix.jdbcx.IfxXADataSource
```

- 「プロパティー」:
  - **`serverName`** - Informix データベースサーバー名を指定します。
  - **`portNumber`** - データベースサーバーのポート番号を指定します。
  - **`user`** - 必要に応じて設定します。
  - **`password`** - 必要に応じて設定します。
  - **`databaseName`** - 必要に応じて設定します。これは任意指定です。
  - **`IfxIFXHost`** - データベースサーバーのホスト名または IP アドレスを指定します。

CloudScape 5.1 Type 4 ドライバ

この CloudScape ドライバの JAR ファイルは `db2j.jar`、`db2jtools.jar`、`db2jcview.jar`、`jh.jar`、`db2jcc.jar`、および `db2jnet.jar` です。次のように接続プールを設定します。

- 「名前」: あとで JDBC リソースを設定するときにこの名前を使用します。
- 「リソースタイプ」: 適切な値を指定します。
- 「データベースベンダー」: `Cloudscape`
- 「データソースクラス名」: `com.ibm.db2.jcc.DB2DataSource`
- 「プロパティー」:
  - **`user`** - 必要に応じて設定します。
  - **`password`** - 必要に応じて設定します。
  - **`databaseName`** - 必要に応じて設定します。

◆ ◆ ◆ 第 4 章

Java Message Service (JMS) リソースの設定

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 管理タスクには、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の高度なタスクには、*MQ-as-install*/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 Administration Guide』を参照してください。

この章では、Java Message Service (JMS) API を使用するアプリケーションのリソースを設定する方法について説明します。この章の内容は次のとおりです。

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。通常、JMS アプリケーションは 1 つ以上の接続ファクトリと 1 つ以上の宛先を使用します。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

『*Java EE 5 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/javaee/5/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 章コネクタリソースを参照してください。

JMS 接続ファクトリ

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 プロパティーを (host:mqport,host2:mqport,host3:mqport の形式で) 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」>「Java メッセージサービス」>「物理送信先」の順に選択します。「新規」をクリックし、「新しい物理送信先」ページで、物理送信先の名前を指定し、送信先のタイプを topic または queue から選択します。「新しい物理送信先」ページのフィールドと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。

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きに、Message Queue は、送信先リソースの Name 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 プロバイダのプロパティーの設定

管理コンソールの「Java メッセージサービス」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。「Java メッセージサービス」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」チェックボックスにチェックマークを付けて、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。これは、AddressList で指定した各アドレスおよびリストの次のアドレスへの試行に適用されます。間隔が短すぎると、ブローカにリカバリする時間が与えられません。間隔が長すぎると、再接続が許容できない遅延を示す場合があります。
- 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 5 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 5 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

Application Server 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。

全体として、JMS プロバイダが XA をサポートするかどうかを、リソースアダプタの設定により選択できます。さらに、JMS プロバイダで可能な統合のモードを選択することもできます。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。

管理コンソールまたはコマンド行を使用して、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。
2. コネクタ接続プールを作成します。
3. コネクタリソースを作成します。
4. 管理対象オブジェクトリソースを作成します。
5. Application Server の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。
ConnectionFactoryClassName	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。
QueueConnectionFactoryClassName	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。ProviderIntegrationMode が javabean の場合に使用します。
TopicConnectionFactoryClassName	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。ProviderIntegrationMode が javabean と指定されている場合に使用します。
XAConnectionFactoryClassName	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。ProviderIntegrationMode が javabean と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
XAQueueConnection FactoryClassName	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.XA QueueConnectionFactory	なし	JMS クライアントの javax.jms.XAQueueConnectionFactory 実装のクラス名。ProviderIntegrationMode が javabean と指定されている場合に使用します。
XATopicConnection FactoryClassName	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.XA TopicConnectionFactory	なし	JMS クライアントの javax.jms.XATopicConnectionFactory 実装のクラス名。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。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ProviderIntegrationMode が javabean と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。
ConnectionFactory Properties	コンマで区切られた名前と値のペア	なし	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。ProviderIntegrationMode が javabean である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	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。ProviderIntegrationMode が jndi である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	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。`ProviderIntegrationMode` が `javabean` である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。

ManagedConnectionFactory 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
ConnectionFactory JndiName	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 ProviderIntegrationMode が jndi の場合にのみ使用されます。
ConnectionValidation Enabled	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	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	同時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。
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。

プロパティー名	有効な値	デフォルト値	説明
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	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	整数		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。
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。
DeadMessageDestinationJndiName	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。
DeadMessageDestinationClassName	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。
DeadMessageDestinationProperties	コンマで区切られた名前と値のペア	なし	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	整数		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。

プロパティー名	有効な値	デフォルト値	説明
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と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 は、Java プラットフォームのオプションパッケージとして実装され、J2EE プラットフォームの一部としても使用できます。

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

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

ここでは、次の内容について説明します。

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ーを使用して、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な名前。JavaMail リソースのネーミングサブコンテキストプレフィックス mail/ を使用することをお勧めします。次に例を示します。`mail/MySession`
- 「メールホスト」: デフォルト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」: メールサーバーへの接続時に渡されるユーザー名。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」: デフォルトユーザーの電子メールアドレス。次の形式で入力します。*username@host.domain*
- 「説明」: コンポーネントの説明を入力します。
- 「状態」: 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通信の方法を定義します。デフォルトでは、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メソッドクラスを指定します。デフォルトでは、ストアプロトコルクラスは `com.sun.mail.imap.IMAPStore` です。
- 「転送プロトコル」: トランスポート通信の方法を指定します。デフォルトでは、トランスポートプロトコルは `smtp` です。
- 「転送プロトコルクラス」: トランスポートクラスの通信メソッドを定義します。デフォルトでは、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、追加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 のログレベルが「FINE」以上に設定されている場合は、デバッグ出力が生成され、それがシステムログファイルに記録されます。

- 「追加プロパティ」: 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

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。

この節の内容は、次のとおりです。

Java EE 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、Java EE 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。Java EE 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。

Java EE 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。

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

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

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

ネーミング参照とバインディング情報

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

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

別のリソースにアクセスする必要があっても、名前を変更する必要がなくなります。この柔軟性により、既存のコンポーネントから Java EE 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 Java EE 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、Java EE リソースの JNDI 接続ファクトリリソースを設定し、これらのリソースにアクセスする方法を説明します。

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

外部 JNDI リポジトリとリソースの使用

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、Java スキーマのように一般的な Java オブジェクトを LDAP サーバーに格納できます。外部 JNDI リソース要素を使用すると、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 element specifies how to access J2EE resources
 -- stored in an external JNDI repository. The following example
 -- illustrates how to access a java object stored in LDAP.
 -- factory-class element specifies the JNDI InitialContext factory that
 -- needs to be used to access the resource factory. property element
 -- corresponds to the environment applicable to the external JNDI context
 -- and jndi-lookup-name refers to the JNDI name to lookup to fetch the
 -- designated (in this case the java) object.
 -->
<!-- external-jndi-resource 要素は、外部 JNDI リポジトリに格納されて
-- いる Java EE 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

この章では、エンタープライズ情報システム (EIS) へのアクセスに使用されるコネクタの設定方法について説明します。この章で説明する内容は次のとおりです。

コネクタの概要

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 EE 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、および非リレーショナルデータベースなどです。ほかの Java EE モジュールと同様に、コネクタモジュールをインストールするには、これを配備する必要があります。

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

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

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

コネクタ接続プールの管理

コネクタ接続プールを作成、編集、および削除するには、管理コンソールで「リソース」>「コネクタ」>「コネクタ接続プール」の順にクリックします。コネクタ接続プールの管理手順の詳細については、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。

▼ EISアクセスをセットアップする

1. コネクタを配備 (インストール) します。コネクタの配備手順の詳細については、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。
2. コネクタの接続プールを作成します。
3. 接続プールに関連付けるコネクタリソースを作成します。

コネクタリソースの管理

コネクタリソースを作成、編集、および削除するには、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。コネクタリソースの管理手順の詳細については、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。

管理対象オブジェクトリソースの管理

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

管理対象オブジェクトリソースを作成、編集、および削除するには、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。管理対象オブジェクトリソースの管理手順の詳細については、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。

◆ ◆ ◆ 第 8 章

J2EE コンテナ

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

この章では、次のコンテナについて説明します。

Web コンテナ

Web コンテナは、Web アプリケーションを稼動させる Java EE 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

エンタープライズ Beans (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、エンタープライズ Beans 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。

エンタープライズ Beans には、次の 3 つのタイプがあります。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 です。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。

◆ ◆ ◆ 第 9 章

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、Java EE 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、次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。配備記述子はアプリケーションの開発後に変更可能なので、宣言によるセキュリティーの方が柔軟性に富んでいます。

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

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

セキュリティー管理用ツール

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

- 管理コンソール。サーバー全体のセキュリティー設定、ユーザー、グループ、レルムの管理、およびほかのシステム全体のセキュリティータスクの実行に使用するブラウザベースのツール。管理コンソールの概要については、34 ページの「管理用ツール」を参照してください。セキュリティータスクの概要については、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。
- `asadmin`。管理コンソールと同じタスクの多くを行うコマンド行ツール。管理コンソールからできない操作でも、`asadmin` を使用して操作できる場合があります。コマンドプロンプトまたはスクリプトのいずれかから `asadmin` コマンドを実行して、繰り返しのタスクを自動化します。`asadmin`の概要については、34 ページの「管理用ツール」を参照してください。

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。

エンタープライズプロファイルでは、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 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。このパスワードはインストール時には変更不可能なので、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た場合、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るものもあります。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。

- 103 ページの「domain.xml ファイル内のパスワードの暗号化」
- 104 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」
- 104 ページの「マスターパスワードの変更」
- 105 ページの「マスターパスワードとキーストアの操作」
- 106 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の暗号化

domain.xml ファイル内のパスワードを暗号化するには、次の手順を実行します。

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
```

次に例を示します。

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

パスワードプロンプトが表示されます。この場合は admin です。詳細については、create-password-alias、list-password-aliases、delete-password-alias コマンドのマニュアルページを参照してください。

2. domain.xml のパスワードの削除および置き換えを行います。これは、asadmin set コマンドを使用して行います。このような目的での set コマンドの使用例は、次のとおりです。

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

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、強制的にマスター

パスワード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 domain-name
マスターパスワードを入力してください>
新しいマスターパスワードを入力してください>
新しいマスターパスワードをもう一度入力してください>
```

2. Application Server を再起動します。

注意 – この時点で、実行中のサーバーインスタンスを再起動してはいけません。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の操作

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、103 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用して管理パスワードを変更する手順については、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。

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。

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。

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。

表 9–1 Application Server の認証方法

認証方法	通信プロトコル	説明	ユーザー資格の暗号化
BASIC	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。
FORM	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。
CLIENT-CERT	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、かつ同じ認証方法 (BASIC、FORM、CLIENT-CERT) 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、109 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 Java EE 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。

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 章メッセージセキュリティーの設定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』を参照してください。

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。

- 110 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。
- 110 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。
- 110 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例えられます。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かかわらず、適切な鍵が使用される場合だけ対応します。
- 111 ページの「レルム」:ユーザーとグループの情報、および関連するセキュリティー資格を含むリポジトリ。レルムは、「セキュリティーポリシードメイン」とも呼ばれます。

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

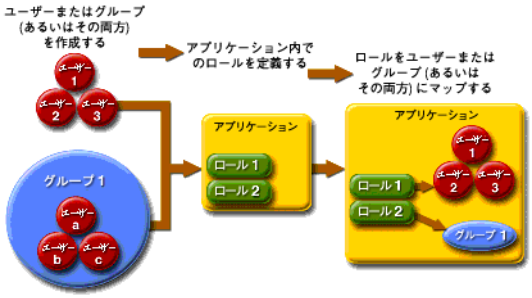

図 9–1 ロールマッピング

ユーザー

「ユーザー」とは、Application Server によって定義された個人またはアプリケーションプログラムの ID です。ユーザーは 1 つのグループと関連付けできます。Application Server の認証サービスは、複数のレルムでユーザーを管理できます。

グループ

「J2EE グループ」(または単にグループ) は、肩書きや顧客のプロファイルなど、共通の特性で分類されたユーザーのカテゴリです。たとえば、E コマースアプリケーションのユーザーは `CUSTOMER` グループに属しますが、お得意様は `PREFERRED` グループに属します。ユーザーをグループに分類すると、大量のユーザーによるアクセスを容易に制御できます。

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 には、次の 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、`JDBC` 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、113 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。

`JDBC` レルムでは、サーバーはデータベースからユーザー資格を取得します。Application Server は、データベース情報および設定ファイル内の有効化された JDBC レルムオプションを使用します。

`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。

▼ Java EE アプリケーションの JDBC レルムを設定する

Application Server では、接続プールの代わりに JDBC レルムにユーザーの資格を指定できます。接続プールの代わりに JDBC レルムを使用すると、ほかのアプリケーションがユーザー資格のデータベース表を参照するのを防止できます。ユーザーの資格とは、ユーザーの名前とパスワードです。

注 – JDBC レルムでは、デフォルトでは平文によるパスワードの保存はサポートされません。通常の状況では、パスワードを平文で保存しないでください。

---

1. レルムのユーザー資格を格納するデータベース表を作成します。
   データベース表の作成方法は、使用しているデータベースによって異なります。

2. 手順 1 で作成したデータベース表にユーザーの資格を追加します。
   データベース表にユーザーの資格を追加する方法は、使用しているデータベースによって異なります。

3. JDBC レルムを作成します。
   この作業には管理コンソール GUI を使用します。JDBC レルムの作成手順については、管理コンソール GUI のオンラインヘルプを参照してください。

4. 手順 3 で作成したレルムをアプリケーションのレルムとして指定します。
   レルムを指定するには、アプリケーションの該当する配備記述子を変更します。

   - **Enterprise Archive (EAR)** ファイルのエンタープライズアプリケーションの場合は、`sun-application.xml` ファイルを変更します。
   - **Web Application Archive (WAR)** ファイルの **Web** アプリケーションの場合は、`web.xml` ファイルを変更します。
   - **EJB JAR** ファイルのエンタープライズ **Bean** の場合は、`sun-ejb-jar.xml` ファイルを変更します。

   レルムの指定方法の詳細については、『Sun Java System Application Server 9.1 Developer's Guide』の「How to Set a Realm for an Application or Module」を参照してください。

5 レルム内のユーザーにセキュリティーロールを割り当てます。

ユーザーにセキュリティーロールを割り当てるには、手順 4 で変更した配備記述子に `security-role-mapping` 要素を追加します。

次の例は、ユーザー `Calvin` にセキュリティーロール `Employee` を割り当てる `security-role-mapping` 要素を示しています。

```
<security-role-mapping>
    <role-name>Employee</role-name>
    <principal-name>Calvin</principal-name>
  </security-role-mapping>
```

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。

- 113 ページの「デジタル証明書について」
- 114 ページの「SSL (Secure Sockets Layer) について」

デジタル証明書について

「デジタル証明書」(または単に証明書) とは、インターネット上の人物やリソースを一意に識別する電子ファイルです。さらに証明書は 2 つのエンティティ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、114 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。

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

公開鍵のほかに、通常、証明書には次のような情報が含まれています。

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

デジタル証明書は、X.509 形式の技術仕様で管理されます。certificate レルムのユーザー ID を検証するために、 `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もっと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 回かぎりの一意の「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。120 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

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。

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、123 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。

  開発者プロファイルでは、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。クラスタおよびエンタープライズプロファイルでは、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。いずれのプロファイルでも、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。

証明書ファイルの新しい場所の値フィールドを追加または変更するには、管理コンソールを使用します。

```
-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。開発者プロファイルの場合、Application Server はサーバー側で、JSSE 形式を使って証明書とキーストアを管理します。すべてのプロファイルで、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 形式を使用します。

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、120 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、121 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、122 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、117 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。デフォルトのキーパスワードエイリアスは「s1as」です。

   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、121 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。117 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

クラスタおよびエンタープライズプロファイルの場合、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、118 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、123 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、124 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、125 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 *as-install*`/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 =${`*as-install*`}/lib`
- `${`*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。

- 123 ページの「certutil ユーティリティーの使用」
- 124 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」
- 125 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。

Sun Java System Application Server は、SSL 通信および 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 は、Sun Crypto Accelerator 1000 (SCA-1000) および SCA-4000 でテスト済みです。

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、122 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 トークンは 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。

- 127 ページの「PKCS#11 トークンの設定」
- 131 ページの「J2SE 5.0 PKCS#11 プロバイダの設定」

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、131 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、122 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/security/p11guide.html` の『Java PKCS#11 Reference Guide』を参照してください。

ここで説明する内容は次のとおりです。

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、122 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 *as-install*`/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。

◆ ◆ ◆ 第 10 章

10 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、『Java EE 5.0 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、134 ページの「システム管理者」ロールと135 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、135 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。開発者プロファイルとクラスタプロファイルでは、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。一方、エンタープライズプロファイルでは、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、140 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。

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。トランスポートレイヤーセキュリティーを使用する場合も、基本認証または SSL クライアント証明書認証を使う場合も、呼び出し元 ID を確立する方法はほぼ同様です。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、117 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、142 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、『Developer's 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、140 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による順序で示されるので、「コンテンツのあと」ではメッセージ受信者が署名の検証前にメッセージを復号化することを意味します。

- 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による順序で示されるので、「コンテンツのあと」ではメッセージ受信者が署名の検証前にメッセージを復号化することを意味します。

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、146 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の有効化」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、140 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、144 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 (アプリケーション構築上で) 設定するには、その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の詳細については、『Sun Java System Application Server 9.1 Developer's Guide』の第 5 章「Securing Applications」を参照してください。

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 (139 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。*as-install*/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

次の表に、メッセージ保護ポリシーの設定と、その設定の結果として WS-Security SOAP メッセージセキュリティープロバイダが実行するメッセージセキュリティー処理を示します。

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

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。

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

メッセージ保護ポリシー	結果として実行される WS-Security SOAP メッセージ保護処理
auth-source="content"	SOAP メッセージ本体のコンテンツが署名されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内にメッセージ本体の署名が ds:Signature として格納されます。
auth-source="sender" auth-recipient="before-content" または auth-recipient="after-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。メッセージに wsse:Security ヘッダーが格納され、そのヘッダー内に wsse:UsernameToken (パスワード付き) と xenc:EncryptedKey が格納されます。また、xenc:EncryptedKey には SOAP メッセージ本文の暗号化に使用する鍵が含まれます。この鍵は、受信者の公開鍵内で暗号化されています。
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 は署名されています。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。また、xenc:EncryptedKey には SOAP メッセージ本文の暗号化に使用する鍵が含まれます。この鍵は、受信者の公開鍵内で暗号化されています。
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。また、xenc:EncryptedKey には SOAP メッセージ本文の暗号化に使用する鍵が含まれます。この鍵は、受信者の公開鍵内で暗号化されています。
auth-recipient="before-content" または auth-recipient="after-content"	SOAP メッセージ本体のコンテンツが暗号化され、その結果得られた xend:EncryptedData で置換されます。メッセージには、xenc:EncryptedKey を含む wsse:Security ヘッダーが含まれます。また、xenc:EncryptedKey には SOAP メッセージ本文の暗号化に使用する鍵が含まれます。この鍵は、受信者の公開鍵内で暗号化されています。
ポリシーを何も指定しない。	モジュールはセキュリティー処理を一切行いません。

その他のセキュリティー機能の設定

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、142 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、144 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 Version 1.5 以上で 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。

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

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

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

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

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

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

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=org.bouncycastle.jce.provider.
      BouncyCastleProvider
   security.provider.3=com.sun.net.ssl.internal.ssl.Provider
   security.provider.4=com.sun.rsajca.Provider
   security.provider.5=com.sun.crypto.provider.SunJCE
   security.provider.6=sun.security.jgss.SunProvider
   ```
5. ファイルを保存して、閉じます。
6. Application Server を再起動します。

メッセージセキュリティーの設定

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。

- 144 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」
- 145 ページの「メッセージセキュリティープロバイダの設定」
- 146 ページの「メッセージセキュリティープロバイダの作成」
- 146 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の有効化」
- 146 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」
- 147 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する場合に 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、146 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の有効化」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して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の設定

プロバイダの再設定は、プロバイダタイプ、実装クラス、およびプロバイダ固有の設定プロパティーを変更するために実行することもできますが、通常はメッセージ保護ポリシーを変更するために実行します。

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」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の有効化

クライアントプロバイダのメッセージ保護ポリシーは、通信相手となるサーバー側プロバイダのメッセージ保護ポリシーと等しくなるように設定する必要があります。Application Server がインストールされるとき、プロバイダはデフォルトでそのように設定されます (ただし、有効化はされていない)。

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による順序で示されるので、「コンテンツのあと」ではメッセージ受信者が署名の検証前にメッセージを復号化することを意味します。

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、146 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の有効化」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。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="as-install/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、140 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。

- 『Java EE 5.0 Tutorial』の「Security」の章については、`http://java.sun.com/javaee/5/docs/tutorial/doc/index.html` を参照してください。
- 『管理ガイドの』 の章。
- 『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 診断サービスの設定

診断サービスを使用すると、サーバーやそのアプリケーションの実行時パフォーマンスをわかりやすい形式で表示および制御でき、障害の発生時にその障害を診断および特定できます。

この章で説明する内容は次のとおりです。

診断フレームワークとは

Application Server 診断フレームワークは、アプリケーションサーバーの標準ライフサイクル内で実行するサービスのセットを定義および実装するための監視フレームワークです。診断サービスでは、稼働しているサーバーやサーバーが配備するアプリケーションによって生成される診断データの定義、作成、収集、およびそれらのデータへのアクセスが可能です。

診断サービスフレームワーク

診断サービスは、アプリケーションサーバーインスタンスの設定の詳細を報告します。例外やパフォーマンスの問題、その他の予期しない結果が起きたときなど、アプリケーションサーバーの問題を診断するのに役立ちます。管理コンソールの「診断サービス」で、次の操作を実行できます。

- 「チェックサムの計算」: `appserver_install_dir/ lib`、`appserver_install_dir/etc`、および `appserver_install_dir/bin` ディレクトリにある Application Server のバイナリファイルのうち、選択したファイルのチェックサムを収集します。
- 「設定の確認」: `domain.xml` や `server.policy` などの設定ファイルを取り込みます。

- 「インストールログの取り込み」: Application Server のバージョン番号やパッチ ID、インストール時に生成されたログファイルの内容など、インストールに関係する詳細情報。インストールディレクトリの絶対パスは、同じマシンに複数のインストール環境がある場合に、収集するインストールログファイルの判別に使用されます。`config/asenv.conf` の内容は、DAS のインストールフォルダとノードエージェントからコピーされます。

  インストールに関係する詳細情報は、ファイルベースでのインストールに関してのみ収集されます。
- 「システム情報の取り込み」: デフォルトでは、次のシステム情報が収集されます。
  - ネットワーク設定
  - OS の詳細
  - ハードウェア情報
- 「アプリケーション配備記述子の取り込み」: `ejb-jar.xml`、`sun-ejb-jar.xml`、`web.xml`、`sun-web.xml` などの配備記述子。
- 「ログレベル」:
- 「ログエントリ」:

  生成される診断レポートに含めるログエントリの数。

診断レポートの生成

診断レポートは、管理コンソールの「診断」タブでの設定内容に基づいて生成されます。生成されたレポートでは、機密データは「機密プロパティー」テーブルにリストされているとおりに表示されます。

◆ ◆ ◆ 第 12 章

12 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 (Unit of Work) 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。

トランザクションについて

トランザクションとは

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

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

これらの手順のいずれが失敗すると、先行する手順によって行われた変更がすべて取り消されます。当座預金口座と普通預金口座はこのトランザクションが始まる前と同じ状態になる必要があります。このイベントは「ロールバック」と呼ばれま

す。すべての手順が正常に完了すると、トランザクションは「コミット」状態になります。トランザクションはコミットかロールバックのどちらかで終了します。

関連項目

- 152 ページの「J2EE テクノロジのトランザクション」
- 153 ページの「トランザクションの設定」

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。

関連項目

- 151 ページの「トランザクションとは」
- 153 ページの「トランザクションの設定」

特定のデータベースに関する問題の回避方法

Application Server には、次の JDBC ドライバの回復実装に関するいくつかの既知の問題に対する回避方法が用意されています。明示的に無効にしないかぎり、それらの回避方法が使用されます。

- Oracle thin ドライバ - `XAResource.recover` メソッドは、入力フラグに関係なく、繰り返し同じ未確定 Xid セットを返します。XA 仕様に従って、トランザクションマネージャーは最初に TMSTARTSCAN でこのメソッドを呼び出したあと、TMNOFLAGS で、Xid が返されなくなるまで繰り返しこのメソッドを呼び出します。`XAResource.commit` メソッドにもいくつかの問題があります。

  Application Server の回避方法を無効にするには、`oracle-xa-recovery-workaround` プロパティー値を `false` に設定します。プロパティーの設定方法の詳細については、154 ページの「Application Server のトランザクションからの回復方法を設定する」を参照してください。

注 – これらの回避方法は、特定の JDBC ドライバに対応するものではありません。

トランザクションに関する管理コンソールタスク

Application Server は、管理コンソールの設定に基づいてトランザクションを処理します。

トランザクションの設定

この節では、トランザクションの設定方法について説明します。

- 154 ページの「Application Server のトランザクションからの回復方法を設定する」
- 155 ページの「トランザクションタイムアウト値を設定する」
- 155 ページの「トランザクションログの位置を設定する」
- 156 ページの「キーポイント間隔を設定する」

トランザクションに関する追加情報については、次の各節を参照してください。

- 151 ページの「トランザクションとは」
- 152 ページの「J2EE テクノロジのトランザクション」

▼ Application Server のトランザクションからの回復方法を設定する

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

復元を行なっている間に一部のリソースにアクセスできない場合は、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ている可能性があります。

トランザクションが複数のサーバーにわたっている場合は、トランザクションを開始したサーバーがトランザクションの結果を取得しようとしてほかのサーバーに問い合わせる場合があります。ほかのサーバーにアクセスできない場合、そのトランザクションは「特殊な結果判別」フィールドを使用してその結果を判別します。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「トランザクションサービス」ノードを選択します。
4. 未完了なトランザクションの復元を有効にするには、「再起動時」フィールドで「回復」にチェックマークを付けます。
5. 「再試行タイムアウト」フィールドに、**Application Server** がアクセスできないサーバーに対して接続を試みる時間を秒単位で設定します。デフォルト値は **10** 分 **(600** 秒**)** です。
6. 「特殊な結果判別」フィールドに、トランザクションでアクセスできないサーバーのポリシーを設定します。
   このフィールドを「コミット」に設定する適切な理由がないかぎり、「特殊な結果判別」を「ロールバック」のままにしておきます。未確定なトランザクションのコミットは、アプリケーションのデータの整合性を損なう可能性があります。

7 必要なプロパティーがあれば設定します。
「プロパティーを追加」ボタンをクリックし、「名前」フィールドと「値」フィールドに名前と値を入力し、「名前」フィールドの横にあるボックスにチェックマークを付けて、そのプロパティーを有効にします。

8 「保存」をクリックします。

9 **Application Server** を再起動します。

▼ トランザクションタイムアウト値を設定する

デフォルトでは、サーバーはトランザクションをタイムアウトしないようになっています。つまり、サーバーはトランザクションの完了を待機し続けます。トランザクションのタイムアウト値を設定して、トランザクションが設定された時間内に完了しない場合、Application Server はトランザクションをロールバックします。

1 ツリーコンポーネントで、「設定」ノードを選択します。

2 設定するインスタンスを選択します。

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

3 「トランザクションサービス」ノードを選択します。

4 「トランザクションタイムアウト」フィールドに、トランザクションがタイムアウトする秒数を入力します。
トランザクションタイムアウトのデフォルト値は 0 秒です。これにより、トランザクションのタイムアウトは無効になります。

5 「保存」をクリックします。

6 **Application Server** を再起動します。

▼ トランザクションログの位置を設定する

トランザクションログは、関連リソースのデータの整合性を維持して障害を回復するために、各トランザクションについての情報を記録します。トランザクションログは、「トランザクションログの位置」フィールドで指定したディレクトリの `tx` サブディレクトリに保存されます。これらのログは人間が読み取れるものではありません。

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。
   - すべてのインスタンスのデフォルト値を設定するには、`default-config` ノードを選択します。
3. 「トランザクションサービス」ノードを選択します。
4. 「トランザクションログの位置」フィールドに、トランザクションログの位置を入力します。

   `tx` サブディレクトリが作成され、トランザクションログがそのディレクトリの下に保存されます。

   デフォルト値は `${com.sun.aas.instanceRoot}/logs` です。`${com.sun.aas.instanceRoot}` 変数はインスタンスの名前であり、Application Server インスタンスの起動時に設定されます。`${com.sun.aas.instanceRoot}` の値を表示するには、「実際の値」をクリックします。
5. 「保存」をクリックします。
6. **Application Server** を再起動します。

▼ キーポイント間隔を設定する

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

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 設定するインスタンスを選択します。
   - 特定のインスタンスを設定するには、そのインスタンスの設定ノードを選択します。たとえば、デフォルトインスタンス `server` の場合は、`server-config` ノードを選択します。

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

3. 「トランザクションサービス」ノードを選択します。
4. 「キーポイント間隔」フィールドに、キーポイント処理間のトランザクション数を入力します。
   デフォルト値は 2048 です。
5. 「保存」をクリックします。
6. **Application Server** を再起動します。

◆ ◆ ◆ 第 13 章

13 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。

この章では、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。

- `server`: 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
- `__asadmin`: 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。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 4848 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、ワーカー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。Application Server では、アクセプタスレッドと要求処理 (ワーカー) スレッドの区別はありません。各 HTTP リスナースレッドが、要求の受け付けと処理を行います。このため、Application Server のデフォルト設定で、HTTP リスナーは 50 個のアクセプタスレッドを使用します。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、そのキューからふたたび接続を取り出し、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:4848/` またはそのデフォルトコンテキストルートである `http://localhost:4848/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 14 章

Web サービスの管理

この章では、Application Server による Web サービス管理について説明します。管理コンソールおよび asadmin ツールで、Web サービスの配備、テスト、および管理を行うことができます。複雑な Web サービスをすばやく視覚化して理解し、監視および管理することができます。ドメインに配備されたすべての Web サービスを、Java EE アプリケーションや、EJB のようなアプリケーションコンポーネントと同じように表示できます。

さらに、次の操作も可能です。

- Web サービスの応答時間や呼び出し回数をリアルタイムで追跡してグラフに表示する。
- 応答時間やスループットの障害などの境界条件に対してアラートを生成する。
- Web サービスの呼び出し内容を XML で表示する。
- XSLT を使用して実行時にメッセージを変換する。

この章の内容は次のとおりです。

Web サービスの概要

Web サービスは、クライアントが Simple Object Access Protocol (SOAP) などの XML ベースプロトコルを使用してアクセスし、HTTP などのインターネットプロトコルを介して送信されるアプリケーションです。クライアントは、Web Services Definition Language (WSDL) ファイルなどの XML アーティファクトを使って定義されたインタフェースとバインディングを通して Web サービスアプリケーションにアクセスします。

eXtensible Markup Language (XML) は、World Wide Web Consortium (W3C) によって開発された規格であり、Web サービス構築の基盤の 1 つです。XML によって、Web サービスとクライアントが共通の言語で互いに通信できます。XML は、シンプルで柔軟な、テキストベースのマークアップ言語です。XML データは、山括弧で囲まれたタグを使用してマーク付けされます。タグ内には、そのタグでマーク付けするデータの意味が含まれます。このようなマークアップにより、異なるシステム間で容易にデータを交換できます。

文書型定義 (Document Type Definitions、DTD) または XML スキーマ定義 (XML Schema Definition、XSD) には、XML ドキュメントの構造が記述されます。これには、対応する XML ドキュメントで使用できるタグや、それらのタグの順序などの情報が含まれます。

XSLT (eXtensible Stylesheet Language Transformation の略) は、XML ドキュメントをある形式から別の形式に変換するために使用されます。

Web サービスの規格

Simple Object Access Protocol (SOAP) は、Web サービスに共通のメッセージング形式を提供します。SOAP によって、互いに未知のオブジェクトがメッセージを交換できます。SOAP では、XML ベースのデータコード化形式と HTTP を使用してメッセージを転送します。SOAP は、プログラミング言語にも動作プラットフォームにも依存せず、エンドポイントで特定のテクノロジを一切必要としません。

Universal Description, Discovery, and Integration (UDDI) は、Web サービスの登録、登録解除、および検索を行うための標準的な方法を提供します。電話システムのイエローページと同様に、UDDI レジストリによって、提供者はサービスを登録し、要求者はサービスを検索できます。要求者がサービスを見つけたあとは、要求者と提供者の間でレジストリが果たす役割はもうありません。

Web Services Description Language (WSDL) は、Web サービスの詳細を指定する標準的な方法を定義します。これは汎用 XML スキーマであり、Web サービスのインタフェース、バインディング、および配備に関するその他の詳細を指定できます。このようなサービスの詳細を指定する標準的な方法があることで、クライアントは予備知識がなくても Web サービスを使用できます。

ebXML (Electronic Business using eXtensible Markup Language) は、企業がインターネットを介してビジネスを行うための一連の仕様です。OASIS (Organization for the Advancement of Structured Information Standards) が ebXML 仕様を管理しています。

Java EE Web サービスの規格

Java APIs for XML Processing (JAXP) は、XML ドキュメントの解析や処理を行うための、ベンダー中立の一連の軽量 API です。JAXP によって、規格に準拠するすべての

XML パーサーを Web サービスに「プラグイン」できます。外部パーサーが「プラグイン」されていない場合、JAXP は自身に実装された XML パーサーを使用します。

Java API for XML-based Remote Procedure Calls (JAX-RPC) は、クライアントサーバー遠隔手続き呼び出しに XML ベースプロトコルを使用します。JAX-RPC は、SOAP ベースの相互運用可能で移植性のある Web サービスを可能にします。開発者は、JAX-RPC プログラミングモデルを使用して、SOAP ベースの Web サービスエンドポイントとそれに対応する WSDL 記述、およびクライアントを開発します。JAX-RPC ベースの Web サービスは、Java ベース以外のクライアントと対話できます。同様に、JAX-RPC ベースのクライアントは、Java ベース以外の Web サービスと対話できます。

Java API for XML Registries (JAXR) は、ビジネスレジストリにアクセスするための Java API です。JAXR には、UDDI およびその他のレジストリ仕様 (ebXML など) をサポートする柔軟なアーキテクチャーが備わっています。JAXR クライアント (スタンドアロン Java アプリケーションや J2EE コンポーネントなど) は、JAXR プロバイダが提供する JAXR API の実装を使用して、ビジネスレジストリにアクセスします。JAXR プロバイダは 2 つの部分から構成されます。レジストリ固有の API 実装を提供するレジストリ固有の JAXR プロバイダと、レジストリのタイプに依存しない API のレジストリ機能を実装するプラグイン可能な JAXR プロバイダです。プラグイン可能なプロバイダは、レジストリ固有プロバイダの詳細をクライアントから見えないようにします。

SOAP with Attachments API for Java (SAAJ) により、開発者は、SOAP 1.1 仕様と SOAP with Attachments note に準拠するメッセージを作成および処理できます。SAAJ プロバイダは、添付ファイル付きの SOAP メッセージを処理するための抽象化オブジェクトを提供します。上級開発者は、SAAJ を使用して、SOAP メッセージで直接アプリケーションを動作させることができます。添付ファイルには、完全な XML ドキュメント、XML フラグメント、または MIME タイプの添付ファイルを使用できます。また、SAAJ により、開発者は、ほかの MIME タイプのサポートを有効にできます。JAX-RPC などの JAX テクノロジでは、内部的に SAAJ を使用することで、SOAP の複雑さを開発者に感じさせないようにしています。SAAJ には次の機能があります。

- 要求/応答型の同期メッセージング: クライアントはメッセージを送信し、応答を待ちます。
- 一方向の非同期メッセージング: クライアントはメッセージを送信し、応答を待たずにその処理を続けます。

Web サービスの配備とテスト

Application Server では、Web サービスの配備とテストを簡単に行えます。

Web サービスの配備

エンタープライズアプリケーションと同じように、エンタープライズアーカイブ (EAR) で Web サービスを配備します。

また、POJO (Plain Old Java Object) によって Web サービスを実装することもできます。POJO Web サービスを配備するには、自動配備機能を使用します。これを行うには、サービスを自動配備ディレクトリにドラッグ＆ドロップします。Application Server によって、自動的に適切な Web XML ファイルが生成され、Web サービスが配備されます。

管理コンソールで、「Application Server」 > 「Web サービス」|「一般」の順に選択すると、配備済みの Web サービスの一覧を表示できます。

配備済み Web サービスの表示

管理コンソールで Web サービスをテストするには、「アプリケーション」>「Web サービス」>「*web-service-name*」|「一般」の順に選択します。管理コンソールに、次のような Web サービスの属性が表示されます。

- 名前: Web サービスの名前。
- エンドポイントアドレス URI: Web サービスのエンドポイントの URI。
- アプリケーション: リンクをクリックすると、Web アプリケーションまたはエンタープライズアプリケーションのプロパティーが表示されます。
- WSDL: リンクをクリックすると、Web サービスの WSDL ファイルが表示されます。
- モジュール名: Web サービスの WAR ファイルまたは EAR ファイルの名前。
- マッピングファイル: リンクをクリックすると、Java WSDL マッピングファイルが表示されます。
- Webservices.xml: リンクをクリックすると、webservices.xml ファイルが表示されます。
- 実装タイプ: SERVLET または EJB。
- 実装クラス名:
- 配備記述子:

Web サービスのテスト

管理コンソールで、Web サービスをテストし、問題を診断できます。汎用テストサーブレットで、配備済み Web サービスに対して ping を実行できます。メソッドの呼び出しごとに SOAP メッセージが表示されます。

管理コンソールで Web サービスをテストするには、「アプリケーション」>「Web サービス」>「*web-service-name*」|「一般」の順に選択し、「テスト」ボタンをクリックします。

Web サービスのセキュリティー

SOAP メッセージ層セキュリティーのサポートは、WS-Security の SAML トークンプロファイルに基づきます。Web サービスの改ざん防止監査機能も提供されます。

Web サービスレジストリの使用

注 – Application Server には、内部レジストリが付属していません。Web サービスを内部レジストリに発行するには、レジストリを Application Server にダウンロードし、インストールする必要があります。Web サービスを外部レジストリに発行するには、外部レジストリのアドレスを指定します。

レジストリの追加

管理コンソールで Web サービスレジストリを追加または削除するには、「Application Server」>「Web サービス」|「レジストリ」の順に選択します。このページで、レジストリアクセスポイント (RAP) を作成します。レジストリを追加する場合、次のパラメータを指定します。

- 「JNDI 名」: レジストリの接続リソースプール (JNDI) 名。このコネクタリソースの JNDI 名は、レジストリの JNDI 名です。
- 追加するレジストリの種類を選択します。UDDI 3.0、ebXML のいずれかです。
- 「発行 URL」および「照会 URL」: レジストリの発行用アドレスと照会用アドレスをそれぞれ指定します。形式は http://<hostname>/<path of registry installation> です。
- レジストリのユーザー名とパスワード。

次の手順によって、レジストリ JNDI 名が作成されます。

- レジストリと通信できるリソースアダプタが作成されます。

- アプリケーションサーバーのコンテキストでは、JAXR リソースアダプタは UDDI レジストリと通信するように事前設定されています。また、SOA レジストリリソースアダプタモジュールをダウンロードすることもできます。SOA レジストリは、Sun 独自の ebXML レジストリです。
- リソースアダプタを使用して、接続リソースプールを作成します。
- 作成した接続プールを使用して、コネクタリソースを作成します。

レジストリへの Web サービスの発行

管理コンソールで Web サービスを発行するには、「アプリケーション」>「Web サービス」>「*web-service-name*」|「発行」の順に選択します。

「Web サービスの発行」画面で、Web サービスの発行対象のレジストリを 1 つ以上選択し、「発行」をクリックします。利用可能なレジストリをすべて発行するには、「すべて追加」ボタンをクリックします。

レジストリ内でこの Web サービスを表示するカテゴリを入力します。各カテゴリを区切るには、コンマを使用します。カテゴリは、使用中のレジストリに定義されます。この Web サービスの説明を入力します。UDDI レジストリに発行する場合、Oganization の名前を入力します。

ロードバランサを使用する場合は、ロードバランサのホスト名、ポート番号、および SSL ポート番号を入力します。Web サービスを外部レジストリに発行する場合、WSDL はインターネットを通じて検出されますが、これらのオプションによって WSDL ファイルに指定されたホスト名とポート名はロードバランサのホスト名とポート名に置き換えられます。

Web サービスを発行解除するには、「Web サービスの発行」画面で、Web サービスを発行解除するレジストリを選択し、「発行解除」をクリックします。

XSLT フィルタによるメッセージの変換

Web サービスのエンドポイントに XSLT 変換規則を適用できます。これによって、Web サービスの要求と応答に関して詳細な制御を行うことができます。1 つの Web サービスエンドポイントメソッドに複数の XSLT 規則を適用できます。また、各変換の適用順序も設定できます。XSLT ファイルはすべて、中央リポジトリの `generated/xml/`*appOrModule* ディレクトリに格納されます。これらの変換規則は、リモートサーバーインスタンスと同期されます。

変換規則を SOAP 要求や SOAP 応答に適用できます。

管理コンソールで Web サービス操作に適用する変換規則を追加するには、「アプリケーション」>「Web サービス」>「*web-service-name*」|「変換」の順に選択します。「追加」をクリックします。

この Web サービスのエンドポイントに利用可能な変換規則のリストが表示されます。

変換規則が定義されている XSLT ファイルの場所を参照します。生成された XSLT ファイルはすべて `generated/xml/`*application or module name*`/` ディレクトリに格納されます。

Web サービスのエンドポイントに複数の変換規則を追加する場合、変換規則は追加した順序で適用されます。

変換規則を有効にするには、「変換規則」ページで、その規則のチェックボックスにチェックマークを付けてから、「有効」をクリックします。規則を無効にするには、「無効」をクリックします。

変換規則を削除するには、「変換規則」ページで、その規則のチェックボックスにチェックマークを付けてから、「削除」をクリックします。これによって、変換規則がリストから削除されます。変換規則が Web サービスエンドポイントに適用されている場合、その変換規則は自動的に無効になります。ただし、XSLT ファイルはファイルパスの示す場所に残ります。ほかの Web サービスエンドポイントでは、この XSLT ファイルを使用できます。

Web サービスの監視

管理コンソールで、Web サービス処理の統計情報を追跡してグラフィック表示したり、Web サービスで送受信されたメッセージを表示したりできます。

Web サービスの監視を有効にするには、管理コンソールで、「アプリケーション」>「Web サービス」>「*web-service-name*」|「監視」|「設定」の順に選択します。

「監視の設定」ページで、監視レベルを設定します。

- LOW - Web サービスの応答時間、スループット、要求の合計数、および障害を監視します。メソッドレベルの監視は行いません。
- HIGH - 1 秒あたりの要求数、平均応答時間、およびスループット属性を追跡および監視するメッセージを追加します。
- OFF - 監視を無効にします。

「メッセージ履歴」の値を入力します。デフォルトは 25 です。「リセット」ボタンをクリックして、すべての統計情報をクリアーします。移動平均が再始動されます。

Web サービス統計の表示

Application Server9.1 は、Web サービス処理の統計情報を追跡し、グラフィック表示する機能を提供します。

監視統計を表示するには、「アプリケーション」>「Web サービス」>「*web-service-name*」|「監視」|「統計」の順にクリックします。利用できる統計情報は次のとおりです。

- すべての正常処理または異常処理のミリ秒単位の応答時間 (最大、最小、平均)。
- スループット
- 要求の合計数
- 障害の合計数および障害発生元エンドポイントの URI
- 認証失敗の合計数
- 承認成功の合計数

Web サービスメッセージの監視

Web サービスのエンドポイントに関するメッセージ (デフォルトは 25) を表示するように Web サービスを設定することもできます。これらのメッセージは、リモートサーバーインスタンスのメモリーに格納されます。SOAP 要求、応答、HTTP ヘッダー情報の詳細が表示されます。

Web サービスメッセージを監視するには、「アプリケーション」>「Web サービス」>「*web-service-name*」|「監視」|「メッセージ」の順にクリックします。

有効にすると、Web サービスエンドポイントに関する最新のメッセージ (デフォルトでは 25 件) を表示できます。これらのメッセージはリモートサーバーインスタンスのメモリーに保持されます。これには、SOAP 要求と応答の詳細および HTTP ヘッダー情報が含まれます。

Web サービスに関して受信したメッセージのリストが表示されます。表示されるメッセージの数は、監視の設定によって決まります。

フィルタを選択して、正常処理のメッセージまたは異常処理のメッセージだけを表示することもできます。

◆ ◆ ◆ 第 15 章

ORB (Object Request Broker) の設定

この章では、ORB (Object Request Broker) と IIOP リスナーの設定方法について説明します。内容は次のとおりです。

Object Request Broker の概要

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 の設定

ORB を設定するには、管理コンソールで「設定」タブをクリックします。設定するインスタンスに対応する ORB タブをクリックします。

IIOP リスナーの管理

IIOP リスナーを作成、編集、および削除するには、管理コンソールで「設定」タブをクリックします。設定するインスタンスに対応する ORB タブをクリックします。「IIOP リスナー」タブを選択します。手順の詳細については、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。

◆ ◆ ◆ 第 16 章

16 スレッドプール

Java 仮想マシン (JVM) は、1 回の実行で多数のスレッドをサポートできます。パフォーマンスに役立つように、Application Server は 1 つまたは複数のスレッドプールを維持します。特定のスレッドプールを、コネクタモジュールと ORB に割り当てることができます。

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

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

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

1 つのリソースアダプタまたはアプリケーションだけが Application Server のスレッドをすべて占有している場合、Application Server のスレッドを異なるスレッドプールに分割して、スレッド不足を防止してください。

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。
- 「最小スレッドプールサイズ」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前もって作成されます。
- 「最大スレッドプールサイズ」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ワークキューの総数を入力します。
- 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。

また、`asadmin`コマンドの`create-threadpool`を使用して、コマンド行からスレッドプールを作成することもできます。

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。

また、`asadmin`コマンドの`delete-threadpool`を使用して、コマンド行からスレッドプールを削除することもできます。

◆ ◆ ◆ 第 17 章

17 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。この章の内容は次のとおりです。

ロギングについて

ログレコード

Application Server は、JSR 047 に指定されている「Java EE 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。ログをローテーションするときに、Application Server は `server.log` という名前で新しい空のファイルを作成し、古いファイルの名前を `server.log_`*date* に変更します。*date* はファイルがローテーションされた日付と時刻になります。

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、153 ページの「トランザクションの設定」を参照してください。

Application Server のコンポーネントがログ出力を生成します。アプリケーションコンポーネントもログ出力を生成できます。

アプリケーションコンポーネントは、Apache Commons ロギングライブラリを使ってメッセージをロギングしてもかまいません。ただし、ログ設定を効率的に行いたい場合は、プラットフォーム標準の JSR 047 API を使用することをお勧めします。

ログレコードは次の統一形式に従います。

```
[#|yyyy-mm-ddThh:mm:ss.SSS-Z|Log Level|ProductName-Version|LoggerName|Key Value Pairs|Message|#]
```

次に例を示します。

```
[#|2006-10-21T13:25:53.852-0400|INFO|sun-appserver9.1|javax.enterprise.
system.core|_ThreadID=13;|CORE5004: Resource Deployed:
[cr:jms/DurableConnectionFactory].|#]
```

この例について説明します。

- `[#` と `#]` はレコードの開始と終了をマーク付けします。
- 垂直バー (`|`) はレコードのフィールドを区切ります。
- `2006-10-21T13:25:53.852-0400` 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-appserver9.1`です。
- *LoggerName* はログモジュールのソースを識別する階層ロガーの名前空間で、この場合は `javax.enterprise.system.core` です。
- *Key Value Pairs* はキー名と値で、通常は `_ThreadID=14;` のようなスレッド ID です。
- *Message* は、ログメッセージのテキストです。Application Server のすべての `SEVERE` メッセージと `WARNING` メッセージ、および多くの `INFO` メッセージは、モジュールコードと数値から構成されるメッセージ ID (この場合は `CORE5004`) で始まります。

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

ロガー名前空間の階層

Application Server は各モジュールのロガーを提供します。次の表では、管理コンソールの「ログレベル」ページに表示されるとおりに、アルファベット順でモジュールの名前と各ロガーの名前空間を示します (181 ページの「ログレベルの設定」を参照)。表内の最後の 3 つのモジュールは、「ログレベル」ページには表示されません。

表 17–1　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`

表 17–1 Application Server ロガー名前空間 *(続き)*

モジュール名	名前空間
クラスローダー	javax.enterprise.system.core.classloading
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
グループ管理サービス (クラスタおよびエンタープライズプロファイルのみ)	javax.ee.enterprise.system.gms
JavaMail	javax.enterprise.resource.javamail
JAXR	javax.enterprise.resource.webservices.registry
JAXRPC	javax.enterprise.resource.webservices.rpc
JAXWS	javax.enterprise.resource.webservices.javaws
JBI	com.sun.jbi
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
持続	oracle.toplink.essentials、 javax.enterprise.resource.jdo、 javax.enterprise.system.container.cmp
ノードエージェント (クラスタおよびエンタープライズプロファイルのみ)	javax.ee.enterprise.system.nodeagent
ルート	javax.enterprise
SAAJ	javax.enterprise.resource.webservices.saaj
セキュリティー	javax.enterprise.system.core.security
自己管理	javax.enterprise.system.core.selfmanagement
サーバー	javax.enterprise.system
同期 (クラスタおよびエンタープライズプロファイルのみ)	javax.ee.enterprise.system.tools.synchronization

表 17–1 Application Server ロガー名前空間 (続き)

モジュール名	名前空間
Util	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 org.apache.catalina org.apache.coyote org.apache.jasper

ロギングの設定

ここでは、次の内容について説明します。

ログの一般設定

管理コンソールを使用してログの一般設定を設定するには、次の手順に従います。

- 開発者プロファイルの場合は、「Application Server」→「ログ」→「一般」の順に選択します。
- クラスタプおよびエンタープライズプロファイルの場合は、「設定」→「設定」→「ログ設定」→「一般」の順に選択します。

「一般」ページで適切な値を入力し、必要に応じてログをカスタマイズします。Application Server を停止して再起動します。

各設定パラメータの設定の詳細については、管理コンソールで「ヘルプ」をクリックしてください。

これらのログ設定を `asadmin` で設定するには、`get` および `set` コマンドを使用します。

ログレベルの設定

管理コンソールを使用してログレベルを設定するには、次の手順に従います。

- 開発者プロファイルの場合は、「Application Server」→「ログ」→「ログレベル」の順に選択します。
- クラスタおよびエンタープライズプロファイルの場合は、「設定」→「設定」→「ログ」→「ログ設定」→「ログレベル」の順に選択します。

このページで、一覧表示されたモジュールのログレベルを設定します。アプリケーションロガーのログレベルを設定するには、「追加プロパティー」領域を使用します。モジュールロガーの一覧については、178 ページの「ロガー名前空間の階層」を参照してください。

各設定パラメータの設定の詳細については、管理コンソールで「ヘルプ」をクリックしてください。

これらのログ設定を `asadmin` で設定するには、`get` および `set` コマンドを使用します。

サーバーログの表示

ログファイルを表示するには、次の手順に従います。

- 開発者プロファイルの場合は、「Application Server」→「ログ」→「ログファイルを表示」の順に選択します。
- クラスタおよびエンタープライズプロファイルの場合は、「設定」→「設定」→「ロガーの設定」→「一般」の順に選択し、「ログファイルを表示」をクリックします。

設定内容に基づいたログ結果を表示するには、「検索基準」領域に示されているオプションを使用します。

- インスタンス名: ドロップダウンリストからインスタンス名を選択して、そのサーバーインスタンスのログを表示します。デフォルトは現在のサーバーインスタンスです。
- ログファイル: ドロップダウンリストからログファイル名を選択して、そのログのコンテンツを表示します。デフォルトは `server.log` です。
- タイムスタンプ: 最新のメッセージを表示するには、「最新」 (デフォルト) を選択します。特定期間内のメッセージだけを表示するには、「特定範囲」を選択して、表示される「開始」と「終了」フィールドに日付と時刻を入力します。時刻の値の構文は次の形式を必ず使用してください (*SSS* はミリ秒)。

  `hh:mm:ss.SSS`

次に例を示します。

```
17:10:00.000
```

「開始」フィールドの値が「終了」フィールドの値よりも遅い場合は、エラーメッセージが表示されます。

- ログレベル: ログレベルでメッセージをフィルタリングするには、ドロップダウンリストからログレベルを選択します。デフォルトでは、サーバーログにある選択したログレベルおよびさらに重大なレベルのすべてのメッセージが表示に含まれます。選択したレベルのメッセージだけを表示するには、「より重度の高いメッセージをこれ以上含めない」というラベルの付いたチェックボックスを選択します。

  表示したいメッセージを確実にサーバーログに表示するには、最初に「ログレベル」ページで適切なログレベルを設定してください。181 ページの「ログレベルの設定」を参照してください。

  ログレベルに基づくログメッセージのフィルタリングを選択する場合、指定したフィルタリング基準と一致するメッセージだけが表示されます。しかし、このフィルタリングは、どのメッセージがサーバーログに記録されるかには影響しません。

  最新の 40 エントリが「ログ設定」と「ログレベル」ページで指定した設定で表示されます。

  最新のメッセージが最後に表示されるようにメッセージを並べ替えるには、タイムスタンプヘッダーのとなりの矢印をクリックします。

  形式設定済みのメッセージを表示するには、次の印が付いたリンクをクリックします。

  (詳細)

  「ログエントリの詳細」というウィンドウが現れて、形式設定済みメッセージを表示します。

  エントリのリストの末尾で、ボタンをクリックしてログファイルの古いエントリまたは新しいエントリを表示します。

  「検索基準」領域で「詳細検索」をクリックして、ログビューアの詳細設定を行います。「詳細オプション」フィールドは次のように使用します。

- ロガー — モジュール別にフィルタリングするには、ドロップダウンリストから 1 つ以上の名前空間を選択します。shift-click または control-click を使い、複数の名前空間を選択します。

  上位レベルの名前空間を 1 つ選ぶと、その下のすべての名前空間が選択されます。たとえば、`javax.enterprise.system` を選択すると、その名前空間の下にあるすべてのモジュールのロガーも選択されます。`javax.enterprise.system.core`、`javax.enterprise.system.tools.admin` などです。

- カスタムロガー — 特定のアプリケーションに固有のロガーのメッセージを表示するには、テキストフィールドで 1 行に 1 つずつロガー名を入力します。アプリケーションに複数のモジュールがある場合は、そのいずれかまたはすべてを表示できます。たとえば、使用しているアプリケーションに次の名前のロガーがあるとします。

  ```
  com.mycompany.myapp.module1
  com.mycompany.myapp.module2
  com.mycompany.myapp.module3
  ```

  アプリケーション内のすべてのモジュールのメッセージを表示するには、`com.mycompany.myapp` と入力します。`module2` のメッセージだけを表示するには、`com.mycompany.myapp.module2` と入力します。

  1 つ以上のカスタムロガーを指定した場合、Application Server モジュールのメッセージは、「ロガー」領域で明示的に指定されるときだけ表示されます。
- 名前と値のペア — 特定のスレッドの出力を表示するには、テキストフィールドにスレッドのキー名と値を入力します。キー名は `_ThreadID` です。次に例を示します。

  ```
  _ThreadID=13
  ```

  `com.mycompany.myapp.module2` がいくつかのスレッドで実行されるとします。1 つのスレッドの出力だけを表示するようにログビューアを修正するには、「カスタムロガー」フィールドでモジュールのロガーを指定してからこのフィールドにスレッド ID を指定します。
- 表示 — 一度に 40 メッセージ (デフォルト) 以上を表示するには、ドロップダウンリストからほかの可能な値 (100、250、または 1000) を選択します。

  スタックトレースを表示するには、「過度に長いメッセージを制限」チェックボックスのチェックマークを外します。デフォルトでは、スタックトレースはビューアに表示されませんが、メッセージの「(詳細)」リンクをクリックすると表示できます。

  「基本検索」をクリックして、「詳細オプション」領域を非表示にします。

◆ ◆ ◆ 第 18 章

18 コンポーネントとサービスの監視

この章では、Application Server の管理コンソールを使ってコンポーネントを監視する方法について説明します。この章の内容は次のとおりです。

監視について

Application Server での監視

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

管理ルールを使用して、サーバーに関する潜在的な問題を通知することもできます。さらにオプションで、サーバーが一定のパフォーマンスしきい値に達したときに措置を講じることもできます。詳細については、第 19 章管理ルールの設定を参照してください。

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、211 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、213 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。開発者プロファイルでは、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。

- 187 ページの「アプリケーションのツリー」
- 187 ページの「HTTP サービスのツリー」
- 188 ページの「リソースのツリー」
- 188 ページの「コネクタサービスのツリー」
- 188 ページの「JMS サービスのツリー」
- 189 ページの「ORB のツリー」
- 189 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、190 ページの「EJB コンテナの統計」を参照してください。

例 18–1 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/ステートフルセッション Bean) *
   |    |                 |--- pool (ステートレスセッション/メッセージ駆動型/エンティティー Bean 用) *
   |    |                 |--- methods
   |    |                      |---method1 *
   |    |                      |---method2 *
   |    |                 |--- stateful-session-store (ステートフルセッション Bean)*
   |    |                 |--- 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/ステートフルセッション Bean) *
   |    |                 |--- 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。196 ページの「HTTP サービスの統計」を参照してください。

例 18–2 HTTP サービスの図

```
http-service *
             |---connection-queue *
             |---dns *
             |---file-cache *
             |---keep-alive *
             |---pwc-thread-pool *
             |---virtual-server-1*
             |        |--- request *
             |---virtual-server-2*
             |        |--- request *
```

リソースのツリー

リソースノードには、JDBC 接続プールやコネクタ接続プールなどのプールの監視可能な属性が格納されます。次の図に、各種リソースコンポーネントのトップノードと子ノードを示します。監視統計が利用可能なノードには、アスタリスク (*) を付けています。196 ページの「JDBC 接続プールの統計」を参照してください。

例 18–3 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。197 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 18–4 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 18–5　JMS サービスの図

```
jms-service
        |-- connection-factories (リソースアダプタの世界では接続プールと呼ばれるもの)
        |        |-- 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。199 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 18–6　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。199 ページの「スレッドプールの統計」を参照してください。

例 18–7　スレッドプールの図

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

監視対象のコンポーネントとサービスの統計について

この節では、利用可能な監視統計について説明します。

EJB コンテナの統計

EJB コンテナの統計を次の表に示します。

EJB 統計を次の表に示します。

表 18–1　EJB 統計

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 18–2　EJB メソッドの統計

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。
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 18–3　EJB セッションストアの統計

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

表 18–3 EJB セッションストアの統計 (続き)

属性名	データ型	説明
activationErrorCount	CountStatistic	成功または失敗した最後の操作実行時にメソッド実行に費やされた時間 (ミリ秒)。この情報は、EJB コンテナの監視が有効になっている場合に、ステートレスおよびステートフルのセッション Beans とエンティティー Beans に対して収集されます。
passivationCount	CountStatistic	このストアを使って非活性化されたセッションの数。
passivationSuccessCount	CountStatistic	このストアを使って正常に非活性化されたセッションの数。
passivationErrorCount	CountStatistic	このストアを使って非活性化できなかったセッションの数。
expiredSessionCount	CountStatistic	期限切れによりこのストアから削除されたセッションの数。
passivatedBeanSize	CountStatistic	このストアによって非活性化されたバイト数の合計 (合計、最小、最大を含む)。
passivationTime	CountStatistic	Beans のストアへの非活性化に要した時間 (合計、最小、最大を含む)。
checkpointCount (エンタープライズプロファイルのみ)	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 (エンタープライズプロファイルのみ)	CountStatistic	正常にチェックポイント化されたセッションの数。
checkpointErrorCount (エンタープライズプロファイルのみ)	CountStatistic	チェックポイント化できなかったセッションの数。
checkpointedBeanSize (エンタープライズプロファイルのみ)	ValueStatistic	ストアによってチェックポイント化されたバイト数の合計。
checkpointTime (エンタープライズプロファイルのみ)	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 18–4 EJB プールの統計

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、要求が過剰である可能性を示します。
totalbeanscreated	CountStatistic	関連付けられたプール内でデータ収集開始後に作成された Beans の数。
totalbeansdestroyed	CountStatistic	関連付けられたプール内でデータ収集開始後に破棄された Beans の数。
jmsmaxmessagesload	CountStatistic	メッセージ駆動型 Bean のサービスを提供するために JMS セッション内に一度にロード可能なメッセージの最大数。デフォルトは 1。メッセージ駆動型 Beans 用のプールにのみ適用されます。

EJB キャッシュに関して利用可能な統計を、次の表に示します。

表 18–5 EJB キャッシュの統計

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。

表 18–5 EJB キャッシュの統計 (続き)

属性名	データ型	説明
numexpiredsessionsremoved	CountStatistic	クリーンアップスレッドによって削除された期限切れセッションの数。ステートフルセッション Beans にのみ適用されます。
numpassivationsuccess	CountStatistic	非活性化が正常に終了した回数。ステートフルセッション Beans にのみ適用されます。

タイマーに関して利用可能な統計を、次の表に示します。

表 18–6 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、187 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表 18–7 に、Web モジュールに関して利用可能な統計を表 18–8 に示します。

表 18–7 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、194 ページの「Web コンテナの統計」に示します。

表 18–8　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。
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	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。
sessionpersisttime	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)。

表 18–8 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
cachedsessionscurrent	CountStatistic	この Web モジュールで現在メモリー内にキャッシュされているセッションの数。
passivatedsessionscurrent	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

Sun Java System Application Server 8.x Platform Edition で提供されていた HTTP サービス統計はより高機能な PWC (Production Web Container) の統計に置き換えられました。PWC については 205 ページの「PWC (Production Web Container) の統計」 を参照してください。

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 18–9 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	CountStatistic	前回のサンプリング時点におけるプール内の未使用接続の合計数。

表 18–9 JDBC 接続プールの統計 (続き)

Statistic	単位	データ型	説明
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、表 18–10 に示します。コネクタ作業管理に関する統計を、表 18–11 に示します。

表 18–10 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。

表 18–10 コネクタ接続プールの統計 (続き)

Statistic	単位	データ型	説明
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、次の図に示します。

表 18–11 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。
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。

表 18–11 コネクタ作業管理の統計 (続き)

Statistic	データ型	説明
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、次の図に示します。

表 18–12 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 18–13 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。
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 18–14 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 18–15 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。
uptime	CountStatistic	JVM の稼働時間。

Java SE の JVM 統計

Java SE では、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。Java SE 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/javase/ja/6/docs/ja/ で利用可能になっています。

Java SE の監視ツールについては、http://java.sun.com/javase/6/docs/technotes/tools/#manage を参照してください。

Java SE の JVM で利用可能なクラス読み込み関連の統計を、次の図に示します。

表 18–16　Java SE 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。

Java SE の JVM で利用可能なコンパイル関連の統計を、次の図に示します。

表 18–17　Java SE 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)。

Java SE の JVM で利用可能なガベージコレクション関連の統計を、次の図に示します。

表 18–18　Java SE 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)。

Java SE の JVM で利用可能なメモリー関連の統計を、次の図に示します。

表 18–19　Java SE の JVM 統計 - メモリー

Statistic	データ型	説明
objectpendingfinalizationcount	CountStatistic	ファイナライズを保留しているオブジェクトの概算数。
initheapsize	CountStatistic	JVM が最初に要求したヒープのサイズ。
usedheapsize	CountStatistic	現在使用されているヒープのサイズ。
maxheapsize	CountStatistic	メモリー管理用として使用可能なメモリーの最大サイズ (バイト)。
committedheapsize	CountStatistic	JVM 用としてコミットされたメモリーのサイズ (バイト)。
initnonheapsize	CountStatistic	JVM が最初に要求した非ヒープ領域のサイズ。
usednonheapsize	CountStatistic	現在使用されている非ヒープ領域のサイズ。
maxnonheapsize	CountStatistic	メモリー管理用として使用可能なメモリーの最大サイズ (バイト)。
committednonheapsize	CountStatistic	JVM 用としてコミットされたメモリーのサイズ (バイト)。

Java SE 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、次の図に示します。

表 18–20　Java SE 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。
version	StringStatistic	オペレーティングシステムのバージョン。

Java SE の JVM で利用可能なランタイム関連の統計を、次の図に示します。

表 18–21　Java SE 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)。

Java SE の JVM で利用可能な ThreadInfo 関連の統計を、次の図に示します。

表 18–22　Java SE 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
threadstate	StringStatistic	スレッドの状態。
blockedtime	CountStatistic	このスレッドが BLOCKED 状態に入ったあと経過した時間 (ミリ秒)。スレッド競合監視が無効になっている場合は、-1 が返されます。

表 18–22 Java SE の JVM 統計 - ThreadInfo (続き)

Statistic	データ型	説明
blockedcount	CountStatistic	このスレッドが BLOCKED 状態に入った合計回数。
waitedtime	CountStatistic	スレッドが WAITING 状態に入ったあと経過した時間 (ミリ秒)。スレッド競合監視が無効になっている場合は、-1 が返されます。
waitedcount	CountStatistic	スレッドが WAITING 状態または TIMED_WAITING 状態になった合計回数。
lockname	StringStatistic	このスレッドが獲得をブロックされている監視ロック、またはこのスレッドが Object.wait メソッド経由で通知されるのを待っている監視ロックの文字列表現。
lockownerid	CountStatistic	このスレッドのブロック対象オブジェクトの監視ロックを保持しているスレッドの ID。
lockownername	StringStatistic	このスレッドのブロック対象オブジェクトの監視ロックを保持しているスレッドの名前。
stacktrace	StringStatistic	このスレッドに関連付けられているスタックトレース。

Java SE の JVM で利用可能なスレッド関連の統計を、次の図に示します。

表 18–23 Java SE 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。

表 18–23 Java SE の JVM 統計 - スレッド (続き)

Statistic	データ型	説明
daemonthreadcount	CountStatistic	ライブデーモンスレッドの現在の数。
allthreadids	StringStatistic	すべてのライブスレッド ID のリスト。
currentthreadcputime	CountStatistic	CPU 時間の測定が有効になっている場合は、現在のスレッドに対する CPU 時間 (ナノ秒)。CPU 時間の測定が無効になっている場合は、-1 が返されます。
monitordeadlockedthreads	StringStatistic	監視デッドロックが発生しているスレッド ID のリスト。

PWC (Production Web Container) の統計

次の各表に示す PWC コンポーネントおよびサービスの統計が利用可能です。

- 表 18–24
- 表 18–25
- 表 18–26
- 表 18–27
- 表 18–28
- 表 18–29
- 表 18–30
- 表 18–31

PWC 仮想サーバーに関する統計を、次の表に示します。

表 18–24 PWC 仮想サーバーの統計

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 18–25 PWC 要求の統計

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。

表 18–25 PWC 要求の統計 (続き)

属性名	データ型	説明
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 18–26 PWC ファイルキャッシュの統計

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	メモリーにマップされたファイルのコンテンツ用として使用するアドレス空間。
maxmmapcachesize	CountStatistic	ファイルキャッシュがメモリーにマップされたファイルのコンテンツ用として使用するアドレス空間の最大サイズ。
counthits	CountStatistic	キャッシュ検索の成功回数。
countmisses	CountStatistic	キャッシュ検索の失敗回数。
countinfohits	CountStatistic	ファイル情報検索の成功回数。
countinfomisses	CountStatistic	キャッシュファイル情報検索の失敗回数。

表 18–26 PWC ファイルキャッシュの統計 (続き)

属性名	データ型	説明
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 18–27 PWC キープアライブの統計

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 18–28　PWC DNS の統計

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。
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 18–29　PWC スレッドプールの統計

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。

表 18–29 PWC スレッドプールの統計 (続き)

属性名	データ型	説明
maxqueued	CountStatistic	キュー内に同時に格納可能な要求の最大数。

接続キューとは、処理される前の要求が格納されるキューのことです。接続キューの統計は、キュー内のセッション数や接続が受け付けられるまでの平均遅延時間などを示します。PWC 接続キューに関する統計を、次の表に示します。

表 18–30 PWC 接続キューの統計

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。
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 18–31　PWC HTTP サービスの統計

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カースレッドの最大数。
maxvirtualservers	CountStatistic	各インスタンス内に設定可能な仮想サーバーの最大数。
flagprofilingenabled	CountStatistic	HTTP サービスのパフォーマンスプロファイリングが有効になっているかどうか。有効な値は 0、1 のいずれかです。
flagvirtualserveroverflow	CountStatistic	maxvirtualservers を超える仮想サーバーが設定されているかどうかを示します。これを 1 に設定すると、すべての仮想サーバーで統計が追跡されなくなります。
load1minuteaverage	CountStatistic	要求負荷の過去 1 分間における平均値。
load5minuteaverage	CountStatistic	要求負荷の過去 5 分間における平均値。
load15minuteaverage	CountStatistic	要求負荷の過去 15 分間における平均値。
ratebytestransmitted	CountStatistic	サーバーで定義されたある期間内に送信されたデータの送信速度。この情報が利用不可能な場合、結果は 0 になります。
ratebytesreceived	CountStatistic	サーバーで定義されたある期間内に受信されたデータの受信速度。この情報が利用不可能な場合、結果は 0 になります。

監視の有効化と無効化

ここでは、次の内容について説明します。

管理コンソールを使用した監視レベルの設定

管理コンソールを使用して監視を設定するには、次の手順に従います。

- 開発者プロファイルの場合は、「設定」→「監視」の順に選択します。
- クラスタおよびエンタープライズプロファイルの場合は、「設定」→「設定」→「監視」の順に選択します。

デフォルトでは、すべてのコンポーネントおよびサービスについて監視はオフになっています。監視をオンにするには、コンボボックスから「LOW」または「HIGH」を選択します。監視をオフにするには、コンボボックスから「OFF」を選択します。

監視の設定の詳細については、管理コンソールで利用可能なオンラインヘルプを参照してください。

▼ asadmin を使用して監視レベルを設定する

1 get コマンドを使って監視が現在有効になっているサービスとコンポーネントを確認します。

```
asadmin> get --user admin-user server.monitoring-service.module-monitoring-levels.*
```

次の結果が返されます。

```
server.monitoring-service.module-monitoring-levels.
connector-connection-pool = OFF
server.monitoring-service.module-monitoring-levels.
connector-service = OFF
server.monitoring-service.module-monitoring-levels.ejb-container = OFF
server.monitoring-service.module-monitoring-levels.http-service = OFF
server.monitoring-service.module-monitoring-levels.jdbc-connection-pool = OFF
server.monitoring-service.module-monitoring-levels.jms-service = OFF
server.monitoring-service.module-monitoring-levels.jvm = OFF
server.monitoring-service.module-monitoring-levels.orb = OFF
server.monitoring-service.module-monitoring-levels.thread-pool = OFF
server.monitoring-service.module-monitoring-levels.transaction-service = OFF
server.monitoring-service.module-monitoring-levels.web-container = OFF
```

2 set コマンドを使って監視を有効にします。

たとえば、HTTP サービスの監視を有効にするには、次のようにします。

```
asadmin> set --user admin-user
server.monitoring-service.module-monitoring-levels.http-service=LOW
```

監視を無効にするには、set コマンドを使って監視レベルに OFF を指定します。

監視データの表示

管理コンソールでの監視データの表示

開発者プロファイルの場合、監視データを表示するには、「Application Server」→「監視」の順に選択します。

クラスタおよびエンタープライズプロファイルの場合、スタンドアロンインスタンスの監視データを表示するには、「スタンドアロンインスタンス」→「インスタンス」→「監視」の順に選択します。クラスタ化されたインスタンスの監視データを表示するには、「クラスタ」→「クラスタ」→「インスタンス」→「監視」の順に選択します。

JVM、サーバー、アプリケーション、スレッドプール、HTTP サービス、トランザクションサービス、ログ統計、および呼び出しフロー統計の監視データを選択して表示できます。これらのコンポーネントやサービスの構成を示した図については、186 ページの「監視可能なオブジェクトのツリー構造について」を参照してください。

監視の表示または設定の詳細については、管理コンソールで利用可能なオンラインヘルプを参照してください。

各コンポーネントやサービスの属性の詳細については、189 ページの「監視対象のコンポーネントとサービスの統計について」を参照してください。

asadmin ツールによる監視データの表示

ここでは、次の内容について説明します。

▼ asadmin monitor コマンドを使用して監視データを表示する

asadmin で監視データを表示する方法は 2 とおりあります。最初の方法では、monitor コマンドを使用します。このコマンドは、一般的な監視統計を出力します。統計情報を絞り込むためのオプションと、出力をコンマ区切り (CSV) ファイルに取り込むためのオプションがあります。

1 監視データを表示するには、**monitor** コマンドを使用して、監視データのタイプを指定します。監視データのタイプには、httplistener、keepalive、filecache、connectionqueue、jdbcpool、jvm、threadpool 、servlet、connection、connectorpool、endpoint、entitybean、messagedriven、statefulsession、statelesssession、httpservice、webmodule があります。

たとえば、server 上の jvm のデータを表示するには、次のように入力します。

```
asadmin>monitor --type jvm --user adminuser server

                                JVM Monitoring
UpTime(ms)                                        HeapSize(bytes)
current                   min         max         low         high        count
327142979                 0           531628032   0           45940736    45940736
```

2 監視データを表示して **CSV** ファイルに出力を送信するには、**filename** オプションを使用します。次に例を示します。

```
asadmin> monitor --type jvm --filename myoutputfile --user adminuser server
```

▼ asadmin get および asadmin list コマンドを使用して監視データを表示する

ほとんどの状況で monitor コマンドを使用できます。ただし、monitor コマンドでは、監視可能なすべてのオブジェクトの完全なリストは表示できません。asadmin ツールを使用して監視可能なすべてのデータを表示するには、asadmin list および asadmin get コマンドに続けて監視可能なオブジェクトのドット表記名を使用します。手順を次に示します。

1 監視可能なオブジェクトの名前を表示するには、asadmin list コマンドを使用します。

たとえば、サーバーインスタンスで監視が有効なアプリケーションコンポーネントおよびサブシステムのリストを表示するには、次のコマンドを端末ウィンドウに入力します。

```
asadmin> list --user adminuser --monitor server
```

前述のコマンドは、監視が有効なアプリケーションコンポーネントおよびサブシステムの次のようなリストを返します。

```
server.resources
server.connector-service
```

```
server.orb
server.jms-service
server.jvm
server.applications
server.http-service
server.thread-pools
```

list コマンドのその他の使用例については、217 ページの「list コマンドと get コマンドの例」を参照してください。list コマンドで使用できるドット表記名の詳細については、216 ページの「ドット表記名とその指定方法について」を参照してください。

2 監視が有効なアプリケーションコンポーネントまたはサブシステムの監視統計を表示するには、asadmin get コマンドを使用します。

統計を取得するには、asadmin get コマンドを端末ウィンドウに入力して、前述の手順の list コマンドで表示された名前を指定します。次の例では、特定のオブジェクトのサブシステムからすべての属性を取得します。

```
asadmin> get --user adminuser --monitor server.jvm.*
```

このコマンドは次の属性およびデータを返します。

```
server.jvm.dotted-name = server.jvm
server.jvm.heapsize-current = 21241856
server.jvm.heapsize-description = Provides statistical information about
   the JVM's memory heap size.
server.jvm.heapsize-highwatermark = 21241856
server.jvm.heapsize-lastsampletime = 1080232913938
server.jvm.heapsize-lowerbound = 0
server.jvm.heapsize-lowwatermark = 0
server.jvm.heapsize-name = JvmHeapSize
server.jvm.heapsize-starttime = 1080234457308
server.jvm.heapsize-unit = bytes
server.jvm.heapsize-upperbound = 518979584
server.jvm.uptime-count = 1080234457308
server.jvm.uptime-description = Provides the amount of time the JVM has
   been running.
server.jvm.uptime-lastsampletime = 1080234457308
server.jvm.uptime-name = JvmUpTime
server.jvm.uptime-starttime = 1080232913928
server.jvm.uptime-unit = milliseconds
```

get コマンドのその他の使用例については、217 ページの「list コマンドと get コマンドの例」を参照してください。get コマンドで使用できるドット表記名の詳細については、216 ページの「ドット表記名とその指定方法について」を参照してください。

ドット表記名とその指定方法について

asadmin list コマンドと asadmin get コマンドでは、監視可能オブジェクトのドット表記名を指定します。すべての子オブジェクトのアドレス指定にはドット (.) 文字が区切り文字として使用され、それらの名前は「ドット表記名」と呼ばれます。子ノードが単独タイプの場合、その監視オブジェクトタイプを指定するだけで、そのオブジェクトを指定できます。それ以外の場合は、type.name 形式の名前を指定する必要があります。

たとえば、http-service は、有効な監視可能オブジェクトタイプの 1 つであり、単独タイプです。インスタンス server の http-service を表す単独タイプの子ノードを指定する場合、ドット表記名は次のようになります。

```
server.http-service
```

もう 1 つ例を挙げます。applications は、有効な監視可能オブジェクトタイプですが、単独タイプではありません。たとえば、アプリケーション PetStore を表す、単独タイプでない子ノードを指定するには、ドット表記名は次のようになります。

```
server.applications.petstore
```

また、監視可能なオブジェクトの特定の属性も、ドット表記名で指定します。たとえば、http-service には、bytesreceived-lastsampletime という名前の監視可能な属性があります。次の名前は、bytesreceived 属性を指定していることになります。

```
server.http-service.server.http-listener-1.
  bytesreceived-lastsampletime
```

管理者は、asadmin list コマンドと asadmin get コマンドの有効なドット表記名を覚えておく必要はありません。list コマンドを使えば、利用可能な監視可能オブジェクトが表示され、ワイルドカードパラメータ付きの get コマンドを使えば、任意 の監視可能オブジェクトで利用可能なすべての属性を確認することができます。

list コマンドと get コマンドでドット表記名を使用する場合、根本的に次のことを前提としています。

- list コマンドでドット表記名の後にワイルドカード (*) が指定されていなかった場合、現在のノードの直接の子ノードが結果として返される。たとえば、list --user *adminuser* --monitor server を実行した場合、server ノードに属するすべての直接の子ノードが一覧表示される。
- list コマンドでドット表記名の後に .* 形式のワイルドカードが指定されていた場合、現在のノードの子ノード階層ツリーが結果として返される。たとえば、list --user *adminuser* --monitor server.applications.* を実行した場合、applications のすべての子ノードに加え、それらの配下にある子ノードなども一覧表示される。

- `list` コマンドで *`*dottedname`*、*`dotted`* `*` *`name`*、**`dotted name *`** のいずれかの形式でドット表記名の前後にのワイルドカードが指定されていた場合、そのマッチングパターンによって生成された正規表現にマッチするすべてのノードとそれらの子ノードが、結果として返される。
- `get` コマンドの末尾に「`.*`」、「`*`」のいずれかが指定されていた場合、マッチング対象の現在のノードに属する属性とその値のセットが、結果として返される。

詳細については、223 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

list コマンドと get コマンドの例

ここでは、次の内容について説明します。

- 217 ページの「list --user admin-user --monitor コマンドの例」
- 218 ページの「get --user admin-user --monitor コマンドの例」

list --user admin-user --monitor コマンドの例

`list` コマンドは、指定されたサーバーインスタンス名で現在監視されているアプリケーションコンポーネントやサブシステムに関する情報を提供します。このコマンドを使えば、特定のサーバーインスタンスの監視可能なコンポーネントやそのサブコンポーネントを表示できます。`list` のより詳しい例については、223 ページの「すべてのレベルにおける list コマンドと get コマンドの予想出力」を参照してください。

例 1

```
asadmin> list --user admin-user --monitor server
```

前述のコマンドは、監視が有効なアプリケーションコンポーネントおよびサブシステムの次のようなリストを返します。

```
server.resources
server.orb
server.jvm
server.jms-service
server.connector-service
server.applications
server.http-service
server.thread-pools
```

また、指定されたサーバーインスタンス内で現在監視されているアプリケーションを一覧表示することも可能です。これは、`get` コマンドを使って特定のアプリケーションの特定の監視統計を取得する場合に便利です。

例 2

```
asadmin> list --user admin-user --monitor server.applications
```

次の結果が返されます。

```
server.applications.adminapp
 server.applications.admingui
server.applications.myApp
```

より包括的な例については、219 ページの「PetStore サンプルを使用する」を参照してください。

get --user admin-user --monitor コマンドの例

このコマンドは、次の監視対象情報を取得します。

- 特定のコンポーネントまたはサブシステム内で監視されているすべての属性
- 特定のコンポーネントまたはサブシステム内で監視されている特定の属性

  特定のコンポーネントまたはサブシステムに存在しない属性が要求された場合、エラーが返されます。同様に、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、223 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 Java EE 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.myJavaEEApp.*
```

次の結果が返されます。

```
No matches resulted from the wildcard expression.
CLI137 Command get failed.
```

Java EE 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=4848
```

3 インスタンス server の監視可能なコンポーネントを一覧表示します。

```
asadmin> list --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 **18–32** 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 **18–33** 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...`

表 18–33 アプリケーションレベル (続き)

コマンド	ドット表記名	出力
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 18–34 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール

コマンド	ドット表記名	出力
list -m	server.applications.app1 または *app1 注: このレベルが適用可能なのは、エンタープライズアプリケーションが配備されている場合だけです。スタンドアロンモジュールが配備されている場合には適用できません。	ejb-module1_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...

表 18–34 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール (続き)

コマンド	ドット表記名	出力
list -m	server.applications.app1.ejb-module1_jar または *ejb-module1_jar または server.applications.ejb-module1_jar	このノードから下の子ノード階層。
get -m	server.applications.app1.ejb-module1_jar.* または *ejb-module1_jar.* または server.applications.ejb-module1_jar.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.applications.app1.ejb-module1_jar.bean1 注: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	次の子ノード一覧が表示されます。 bean-poolbean-cachebean-method
list -m	server.applications.app1.ejb-module1_jar.bean1 注: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	子ノードの階層とこのノードとそれより下のすべての子ノードの全属性の一覧。

表 18–34 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.*`
	注: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では `app1`) は表示されません。
		次の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: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では `app1`) は表示されません。	属性は表示されず、`server.applications.appl.ejb-module1_jar.bean1-cache` には表示すべき情報がないことを示すメッセージが表示されます。特定の文字列で始まる有効な名前を取得するには、ワイルドカード (*) 文字を使用します。たとえば、`server` で始まるすべての名前を一覧表示するには、`list server*` を使用します。
`get -m`	`server.applications.app1.ejb-module1_jar.bean1.bean-pool.*` 注: 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 18–4 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: 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。

表 18–34 アプリケーション - エンタープライズアプリケーションとスタンドアロンモジュール (続き)

コマンド	ドット表記名	出力
get -m	server.applications.app1.ejb-module1_jar.bean1.bean-cache.* 注: スタンドアロンモジュールでは、アプリケーション名を含むノード (この例では app1) は表示されません。	表 18–5 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: 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: 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 18–2 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。
get -m	server.applications.app1.web-module1_war.virtual_server.servlet1.*	表 18–7 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 18–35 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。

表 18–35 HTTP サービスレベル (続き)

コマンド	ドット表記名	出力
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 18–36 スレッドプールレベル

コマンド	ドット表記名	出力
list -m	server.thread-pools	スレッドプール名の一覧。
get -m	server.thread-pools.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.thread-pools.orb\.threadpool\.thread-pool-1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.thread-pools..orb\.threadpool\.thread-pool-1.*	表 18–13 で説明したスレッドプール属性に対応する属性と値の一覧。

次の表に、リソースレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 18–37 リソースレベル

コマンド	ドット表記名	出力
list -m	server.resources	プール名の一覧。
get -m	server.resources.*	このノードに属性が存在しないことを示すメッセージだけが表示されます。
list -m	server.resources.jdbc-connection-pool-pool.connection-pool1	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.resources.jdbc-connection-pool-pool.connection-pool1.*	表 18–9 で説明した接続プール属性に対応する属性と値の一覧。

次の表に、トランザクションサービスレベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 18–38 トランザクションサービスレベル

コマンド	ドット表記名	出力
list -m	server.transaction-service	属性は表示されず、代わりに「get --monitor コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
get -m	server.transaction-service.*	表 18–14 で説明したトランザクションサービス属性に対応する属性と値の一覧。

次の表に、ORB レベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 18–39 ORB レベル

コマンド	ドット表記名	出力
list -m	server.orb	server-orb.connection-managers

表 18–39 ORB レベル (続き)

コマンド	ドット表記名	出力
`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.*`	表 18–12 で説明した ORB 接続マネージャー属性に対応する属性と値の一覧。

次の表に、JVM レベルに対するコマンド、ドット表記名、および対応する出力を示します。

表 18–40 JVM レベル

コマンド	ドット表記名	出力
`list -m`	`server.jvm`	属性は表示されず、代わりに「`get --monitor` コマンドを使用して、このノードの属性と値を表示してください。」というメッセージが表示されます。
`get -m`	`server.jvm.*`	表 18–15 で説明した 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。

- 231 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」
- 232 ページの「JConsole を Application Server に接続する前提条件」

- 233 ページの「JConsole を Application Server に接続する」
- 233 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 テクノロジをベースにしています。つまり、Application Server の JVM で実行している MBeanServer では、管理対象コンポーネントは MBean で表されます。

Java SE 5 では、Platform MBean Server を含めること、および JVM を設定するための MBean を含めることにより、JVM の管理と監視を拡張します。Application Server は、これらの拡張機能を利用して、MBean を Platform MBean Server に登録します。JMX コネクタクライアントには、JVM MBean と Application Server MBean が統合表示されます。

すべての MBean を表示するために、Application Server にはシステム 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が用意されています。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が起動します。規格に準拠する JMX コネクタクライアントはすべて、このコネクタサーバーを使用してサーバーに接続できます。

Java SE には、MBean Server に接続し、そこに登録されている MBean を表示するためのツールも用意されています。JConsole は一般的な JMX コネクタクライアントであり、標準 Java SE ディストリビューションの一部として利用できます。JConsole の詳細については、`http://java.sun.com/javase/6/docs/technotes/guides/management/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。233 ページの「JConsole を Application Server に接続する」に、正常な接続を作成する方法が示されています。

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。

- Application Server の開発者プロファイルの場合、デフォルトでは、セキュリティー保護されていないシステム JMX コネクタサーバーが設定されます。
- Application Server のクラスタおよびエンタープライズプロファイルの場合、デフォルトでは、セキュリティー保護されたシステム JMX コネクタサーバーが設定されます。

- 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。

開発者プロファイルドメインを `appserver.sun.com` のようなマシンにインストールすると、ドメイン管理サーバー (DAS) の `domain.xml` ファイル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。クラスタまたはエンタープライズプロファイルが稼働している場合、または開発者プロファイルの 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。この例では、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 6.0 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 (クラスタおよびエンタープライズプロファイルのデフォルト) だけです。前述の Solaris マシンで Application Server 開発者プロファイル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

この手順では、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、開発者プロファイルの Application Server のセキュリティーは有効になっていません。

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

この手順では、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。クラスタまたはエンタープライズプロファイルの Application Server のセキュリティーは、デフォルトで有効になっています。この手順は、開発者プロファイルの 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、リモートの `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** を起動します。

これはクラスタまたはエンタープライズ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。開発者プロファイルの JMX コネクタのセキュリティーを有効にするには、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。

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 4848 を実行します。**4848** 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。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 **DAS** 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、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"

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 19 章

19 管理ルールの設定

この章では、定期的な管理作業の自動化、実行時のさまざまな状況に対応したアプリケーションサーバーの自己調整、および障害の発生防止による可用性の向上を目的とした管理ポリシーの設定について説明します。また、カスタマイズ可能な定義済み管理ルールである自己管理テンプレートについても説明します。

ここでは、次の内容について説明します。

管理ルールについて

管理ルールを設定すると、定期的な管理作業の自動化、実行時のさまざまな状況に対応したアプリケーションサーバーの自己調整、および障害の発生防止による可用性の向上を実現できます。管理ルールには、指定したイベントの発生時や設定したしきい値への到達時に行うアクションが含まれます。指定したイベントに基づいて修正アクションを自動的に実行できる管理ルールを設定できます。

管理ルールは、イベントとアクションの 2 つの部分から構成されます。

- イベントは、JMX 通知機構を使用して定義済みのアクションをトリガーします。
- アクションは、関連付けられたイベントが発生したときにトリガーされます。アクションは、`javax.management.NotificationListener` を実装した通知リスナーである MBean です。

たとえば、イベントが EJB ロガーによって記録された SEVERE メッセージであれば、アクションによってログメッセージの内容を含むアラートを管理者に送信できます。イベントが発生すると、イベントデータが `javax.management.Notification` の `userData` 部分の一部として渡されます。

ルールに指定したアクションは、カスタム MBean として実装する必要があります。そのため、管理ルールを設定する前に、イベント通知を受信したら適切なアク

ションを実行するように設計したカスタム MBean を配備してください。カスタム MBean の開発とその配備については、『Sun Java System Application Server 9.1 Developer's Guide』の第 14 章「Developing Custom MBeans」を参照してください。

Application Server には、いくつかの便利なイベントが用意されており、通知を発行するカスタム MBean を記述することで、それらのイベントをさらに拡張できます。プロパティーを変更することにより、各イベントをさらにカスタマイズできます。

使用可能なイベントタイプは次のとおりです。

- 監視イベント: MBean の属性を監視します。監視イベントは、`javax.management.monitor` パッケージと同様の機能を持っています。監視イベントは、Java SE 5 の `javax.management.monitor` が行う単純な属性の監視に加えて、複雑な属性の監視もサポートしています。
- 通知イベント: カスタム MBean からのイベントを通知します。これらのイベントを使ってカスタムイベントを作成することにより、イベント辞書を拡張します。通知を発行できる MBeans は、すべてイベントとして指定できます。
- システムイベント:
  - ライフサイクル: サーバーの起動、シャットダウン、および終了を示すイベント。
  - ログ: 指定されたロガーがログエントリを書き込んだときにトリガーされるイベント。たとえば、EJB コンテナロガーが SEVERE ログエントリを記録したときに管理者にアラートを送信する管理ルールを作成できます。
  - タイマー: 指定された日時や間隔などでトリガーされるイベント。これらのイベントは、`javax.management.timer` パッケージと同様の機能を持っています。
  - トレース: HTTP/IIOP 要求メソッド、EJB メソッド、および Web メソッドの入口と出口でトリガーされるイベント。たとえば、サーブレットとの対話をログに記録するために使用するサーブレットフィルタを、Web メソッドの入口および出口イベントを使った管理ルールとして設計できます。
  - クラスタ: クラスタまたはインスタンスが起動、停止、または失敗したときにトリガーされるイベント。これらのイベントでは、グループ管理システムのクラスタ監視が使用されます。

管理ルールの設定

管理コンソールで管理ルールを設定するには、次の手順に従います。

- 開発者プロファイルの場合は、「設定」→「管理ルール」の順に選択します。
- クラスタおよびエンタープライズプロファイルの場合は、「設定」→「設定」→「管理ルール」の順に選択します。

注 – 管理ルールをグローバルに有効にするには、このページで「すべての規則」にチェックマークを付けます。管理ルールがグローバルに有効になっていない場合は、どの管理ルールも実行されません。

また、個別の管理ルールを有効にするには、このページで、有効にするルールの横にあるボックスをクリックし、「有効」をクリックします。

ルールの MBean も、ターゲットで有効にする必要があります。MBean を有効にするには、「カスタム MBean」→「MBean」の順に選択します。「カスタム MBean を編集」ページで、「ターゲット」タブをクリックして「カスタム MBean ターゲット」ページにアクセスします。このページで、一部またはすべてのターゲットで MBean を有効にできます。

詳細については、オンラインヘルプを参照してください。

コマンド行から管理ルールを作成するには、`create-management-rule` コマンドを使用します。管理ルールのプロパティーを設定するには、`get` および `set` コマンドを使用します。管理ルールを表示および削除するには、`list-management-rules` および `delete-management-rule` コマンドを使用します。詳細については、各コマンドのオンラインヘルプまたは『Sun Java System Application Server 9.1 Reference Manual』を参照してください。

◆ ◆ ◆ 第 20 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 Application Server には JVM が必要であり、JVM が Application Server を実行し、Application Server 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。この章の内容は次のとおりです。

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。EJB コンポーネントの配備時に、Application Server はコンパイラを実行します。

- 「デバッグ」: JPDA (Java Platform Debugger Architecture) によるデバッグを設定するときは、「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」: デバッグを有効にしたときに 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」: このチェックボックスを選択して、アプリケーションの動的再読み込みを有効にします。

  動的再読み込みが有効になっている場合は (デフォルトでは有効)、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たときにアプリケーションやモジュールを再配備する必要はありません。変更された JSP またはクラスファイルをアプリケーションまたはモジュールの配備ディレクトリにコピーするだけで十分です。サーバーは定期的に変更を確認し、変更が見つかると、自動的かつ動的にアプリケーションを再配備します。この機能は、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ているときは、その転送時のセッションが無効になります。クライアントのセッションを再起動する必要があります。
- 「再読み込みのポーリング間隔」: アプリケーションとモジュールにコードの変更がないかをチェックして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 秒です。
- 「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するまでの非活動の分数を指定します。

また、配備設定を次のように定義します。

- 「自動配備」: このチェックボックスを選択して、アプリケーションの自動配備を有効にします。

自動配備では、アプリケーションやモジュールファイル (JAR、WAR、RAR、または EAR) が特別なディレクトリにコピーされ、そこで 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」: アプリケーションとモジュールにコードの変更がないかをチェックして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 秒です。
- 「ベリファイア」: 配備記述子ファイルを検証するには、「ベリファイアを有効」ボックスにチェックマークを付けます。これは任意指定です。
- 「プリコンパイル」: JSP ファイルをプリコンパイルするには、「プリコンパイルを有効」ボックスにチェックマークを付けます。

◆ ◆ ◆ 付 録 A

ドメインまたはノードエージェントの自動再起動

マシンの再起動が必要になった場合など、ドメインまたはノードエージェントが予想外に停止される場合にそなえて、ドメインまたはノードエージェントが自動的に再起動されるようにシステムを設定することが可能です。

この付録では、次の項目について説明します。

Solaris 10 での自動再起動

Solaris 10 ユーザーは、`asadmin create-service` コマンドを使用して、ノードエージェントまたはドメイン管理サーバー (DAS) を再起動するサービスを作成できます。作成したサービスでは、Solaris サービス管理機能 (SMF) が使用されます。

サービスが再起動するプロセスは、そのサービスが DAS またはノードエージェントのどちらを再起動するかによって異なります。

- DAS を再起動するサービスのプロセスは、`asadmin start-domain` です。
- ノードエージェントを再起動するサービスのプロセスは、`asadmin start-node-agent` です。

サービスはプロセスに、そのプロセスを実行するユーザーの特権を付与します。`asadmin create-service` コマンドを使用して SMF サービスを作成する場合、デフォルトのユーザーはスーパーユーザーです。別のユーザーがプロセスを実行する必要がある場合は、`method_credential` にそのユーザーを指定します。

プロセスを Solaris OS の特権ポートにバインドする場合、そのプロセスには net_privaddr 特権が必要です。Solaris OS の特権ポートは、1024 より小さいポート番号です。

ユーザーが net_privaddr 特権を持っているかどうかを確認するには、そのユーザーとしてログインし、ppriv -l | grep net_privaddr コマンドを入力します。

asadmin create-service コマンドを実行するには、solaris.smf.* 認証が必要です。この認証の設定方法については、useradd および usermod のマニュアルページを参照してください。さらに次のディレクトリツリーでの書き込み権も必要です。/var/svc/manifest/application/SUNWappserver。通常、スーパーユーザーはこれらの権限をどちらも持っています。また、svccfg、svcs、auths などの Solaris 10 管理コマンドが PATH で使用できなければなりません。このコマンドの詳細については、create-service(1) のマニュアルページを参照してください。

構文は次のとおりです。

```
asadmin create-service [--name service-name] [--type das|node-agent]
--passwordfile password-file [--serviceproperties serviceproperties]
domain-or-node-agent-configuration-directory
```

たとえば、domain1 に対して domain1 という名前のサービスを作成する場合は、次の手順に従います。

1. 次のコマンドを実行します。

   asadmin create-service --type das --passwordfile password.txt /appserver/domains/domain1

   これで、ドメイン domain1 を自動的に再起動するサービスが作成されます。このコマンドにより、バックグラウンドで、テンプレートからマニフェストファイルが作成されて検証され、そのファイルがサービスとしてインポートされます。

   注 – 特定の Application Server ドメインにデフォルトのユーザー特権を与えないようにする場合は、サービスのマニフェストを変更し、サービスを再インポートします。ユーザーの特権を調べるには、そのユーザーとしてログインし、ppriv -l コマンドを入力します。

   ---

2. サービスが作成されたら、次の svcadm enable コマンドを使用してサービスを有効にします。

   svcadm enable /appserver/domains/domain1

3. 有効にしたあと、ドメインが停止した場合は、SMF によって再起動されます。

サービスを管理するときに、次の Solaris コマンドが役に立ちます。

- auths
- smf_security
- svcadm
- svccfg
- rbac
- useradd
- usermod

これらのコマンドの詳細については、各コマンドのマニュアルページを参照してください。

Solaris 9 および Linux プラットフォーム上での inittab による自動再起動

Solaris 9 または Linux プラットフォーム上でドメインを再起動するには、/etc/inittab ファイルにテキストを 1 行追加します。

/etc/rc.local またはこれに相当するファイルを使用している場合は、必要な asadmin コマンドを呼び出す行を /etc/rc.local に追加します。

たとえば、opt/SUNWappserver ディレクトリにインストールされた Application Server の domain1 を、password.txt という名前のパスワードファイルを使って再起動するには、次のテキストを追加します。

```
das:3:respawn:/opt/SUNWappserver/bin/asadmin start-domain --user admin
--passwordfile /opt/SUNWappserver/password.txt domain1
```

このテキストは 1 行で記述してください。先頭の 3 文字はこのプロセスに対する一意の指示子ですが、これは変更可能です。

ノードエージェントを再起動する場合の構文も、これと似ています。たとえば、opt/SUNWappserver ディレクトリにインストールされた Application Server の agent1 を、password.txt という名前のパスワードファイルを使って再起動するには、次のテキストを追加します。

```
das:3:respawn:/opt/SUNWappserver/bin/asadmin start-node-agent --user admin
--passwordfile /opt/SUNWappserver/password.txt agent1
```

Microsoft Windows プラットフォーム上での自動再起動

Microsoft Windows 上で自動的に再起動するには、Windows サービスを作成し、ユーザーがログアウトするときにサービスがシャットダウンされないようにします。

Windows サービスの作成

Sun Java System Application Server に同梱されている実行可能ファイル `appservService.exe` と `appserverAgentService.exe` を、Microsoft が提供するサービス制御コマンド (`sc.exe`) と組み合わせて使用します。

`sc.exe` コマンドは Windows XP に含まれており、Windows インストールディレクトリの `system32` サブディレクトリ (通常は `C:\windows\system32` か `C:\winnt\system32` のいずれか) に格納されています。このマニュアルの執筆時点では、Windows 2000 の `sc.exe` が `ftp://ftp.microsoft.com/reskit/win2000/sc.zip` からダウンロード可能となっています。`sc.exe` の使用方法の詳細については、http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndllpro/html/msdn_scmslite.asp を参照してください。

`appservService.exe` と `appservAgentService.exe` の使用方法は次のとおりです。

```
C:\winnt\system32\sc.exe create service-name binPath= \"fully-qualified-path-to-appservService.exe
\"fully-qualified-path-to-asadmin.bat start-command\"
\"fully-qualified-path-to-asadmin.bat stop-command\""
start= auto DisplayName= "display-name"
```

注 – `binpath` と等号 (=) の間にスペースは入れません。等号とパスの間にはスペースが必要です。

たとえば、パスワードファイル `C:\Sun\AppServer\password.txt` を使ってドメイン `domain1` を開始および停止するサービス `SunJavaSystemAppServer DOMAIN1` を作成するには、次のようにします。

```
C:\windows\system32\sc.exe create domain1 binPath=
"C:\Sun\AppServer\lib\appservService.exe \"C:\Sun\AppServer\bin\asadmin.bat
start-domain --user admin --passwordfile C:\Sun\AppServer\password.txt domain1\"
\"C:\Sun\AppServer\bin\asadmin.bat stop-domain domain1\"" start= auto
DisplayName= "SunJavaSystemAppServer DOMAIN1"
```

ノードエージェント `agent1` を開始および停止するサービスを作成するには、次のようにします。

```
C:\windows\system32\sc.exe create agent1 binPath=
"C:\Sun\AppServer\lib\appservAgentService.exe \"C:\Sun\AppServer\bin\asadmin.bat
```

```
start-node-agent --user admin --passwordfile C:\Sun\AppServer\password.txt agent1\"
\"C:\Sun\AppServer\bin\asadmin.bat stop-node-agent agent1\"" start= auto
DisplayName= "SunJavaSystemAppServer AGENT1"
```

注 – `binPath=` パラメータの一部として入力された開始コマンドと停止コマンドは、正しい構文で記述されている必要があります。確認するには、それらのコマンドをコマンドプロンプトから実行します。コマンドを実行してもドメインまたはノードエージェントが正常に開始または停止しない場合、そのサービスは正しく動作していません。

注 – `asadmin` の `start/stop` コマンドとサービスの開始/停止を混在させないでください。両者を混在させると、サーバーの状態の同期が取れなくなります。たとえば、サーバーのコンポーネントが実行されていないのに「コンポーネントが開始された」と表示されたりします。こうした状況を避けるには、サービス使用時には常に、`sc.exe` コマンドを使ってコンポーネントを開始および停止するようにしてください。

---

sc.exe create コマンドでサービスが正しく作成されなかった場合は、そのサービスを削除し、もう一度実行してみてください。サービスを削除するには、`sc.exe delete` "*service-name*" コマンドを使用します。

ユーザーのログアウト時にサービスがシャットダウンされないようにする

デフォルトでは、Java VM は、オペレーティングシステムのシャットダウンまたはユーザーのログアウトが行われることを示すシグナルを Windows からキャッチし、Java VM 自身を完全にシャットダウンします。この動作により、ユーザーが Windows からログアウトすると Application Server サービスがシャットダウンされます。ユーザーがログアウトするときにサービスがシャットダウンされないようにするには、`-Xrs` Java VM オプション (`http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/java.html#Xrs`)を設定します。

`-Xrs` Java VM オプションを設定するには、*as-install*`\domains\`*domain-name*`\config\domain.xml` ファイル内の、Java VM オプションを定義するセクションに次の行を追加します。

```
<jvm-options>-Xrs</jvm-options>
```

Application Server サービスが稼働している場合、変更を有効にするには、そのサービスを停止して再起動します。

注 – Windows 2003 Server インストールでは、 `-Xrs` オプションを `domain.xml` ファイル追加しても、サービスのシャットダウンを防止できないことがあります。この場合は、次のように、このオプションを *as-install*`\lib\processLauncher.xml` ファイルに追加します。

```
<process name="as-service-name">
   ...
   <sysproperty key="-Xrs"/>
   ...
```

自動再起動時のセキュリティー

クラスタプロファイルまたはエンタープライズプロファイルを使用している場合は、Application Server を自動的に再起動するときに、管理パスワードとマスターパスワードが必要です。開発者プロファイルを使用している場合は、これらのパスワードは必要ありません。

クラスタプロファイルおよびエンタープライズプロファイルで必要なパスワードとマスターパスワードは、次のいずれかの方法で処理します。

- Microsoft Windows 上で、ユーザーにパスワードを尋ねるようにサービスを設定します。
  1. サービスコントロールパネルで、作成したサービスをダブルクリックします。
  2. 「プロパティー」ウィンドウの「ログオン」タブをクリックします。
  3. 「デスクトップとの対話をサービスに許可」にチェックマークを付け、必要なパスワードに対するプロンプトがコンポーネント起動時に表示されるようにします。

     ログインしてプロンプトを表示させ、入力時にエントリがエコーバックされないことを確認する必要があります。これがサービスオプションを使用する際のもっとも安全な方法ですが、この方法の場合、ユーザーが関与しないとサービスが利用可能になりません。

     デスクトップとの対話オプションを設定しなかった場合、サービスは「開始保留」状態のままになり、ハングアップしたように見えます。この状態から抜け出すには、このサービスのプロセスを終了してください。
- Windows または UNIX 上で、`--savemasterpassword=true` オプションを使ってドメインを作成し、管理パスワード格納用のパスワードファイルを作成します。コンポーネント起動時に、`--passwordfile` オプションを使ってパスワードが格納されたファイルを指定します。

  次に例を示します。

  1. ドメイン作成時にマスターパスワードを保存します。次の構文では、ユーザーは管理パスワードとマスターパスワードの入力を求められます。

```
asadmin create-domain --adminport 4848 --adminuser admin
--savemasterpassword=true --instanceport 8080 domain1
```

2. Windows の場合は、サービスを作成します。その際、パスワードファイルを使って管理パスワードを提供します。

```
C:\windows\system32\sc.exe create domain1 binPath=
"C:\Sun\AppServer\lib\appservService.exe \"C:\Sun\AppServer\bin\asadmin.bat
start-domain --user admin
--passwordfile C:\Sun\AppServer\password.txt domain1\"
\"C:\Sun\AppServer\bin\asadmin.bat stop-domain domain1\"" start= auto
DisplayName= "SunJavaSystemAppServer DOMAIN1"
```

   パスワードファイル `password.txt` のパスは、`C:\Sun\AppServer\password.txt` です。このファイルには、パスワードが次の形式で格納されています。

```
AS_ADMIN_password=password
```

   たとえば、パスワードが `adminadmin` の場合、次のようになります。

```
AS_ADMIN_password=adminadmin
```

3. UNIX の場合、`inittab` ファイルに追加する行の中で、`--passwordfile` オプションを使用します。

```
das:3:respawn:/opt/SUNWappserver/bin/asadmin start-domain --user admin
--passwordfile /opt/SUNWappserver/password.txt domain1
```

   パスワードファイル `password.txt` のパスは、`/opt/SUNWappserver/password.txt` です。このファイルには、パスワードが次の形式で格納されています。

```
AS_ADMIN_password=password
```

   たとえば、パスワードが `adminadmin` の場合、次のようになります。

```
AS_ADMIN_password=adminadmin
```

◆ ◆ ◆ 付 録 B

domain.xml のドット表記名属性

この付録では、Mbean とその属性を指定するために使用可能なドット表記名属性について説明します。`domain.xml` ファイル内のすべての要素は対応する MBean を持ちます。これらの名前は、「個々の名前をピリオドで区切る」という構文規則に従うため、「ドット表記名」と呼ばれます。

この付録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`

要素名	ドット表記名プレフィックス
servers	`domain.servers` この要素に含まれるすべてのサーバーは、*server-name* としてアクセス可能です。ここで、*server-name* は、server サブ要素の name 属性の値です。
clusters	`domain.clusters` この要素に含まれるすべてのクラスタは、*cluster-name* としてアクセス可能です。ここで、*cluster-name* は、cluster サブ要素の name 属性の値です。
node-agents	`domain.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	コメント
target`.applications.*`	`domain.applications.*`	この別名の解決結果は、*target* のみによって参照されるアプリケーションになります。
target`.resources.*`	`domain.resources.*`	この別名の解決結果は、*target* によって参照されるすべての `dbc-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。

◆ ◆ ◆ 付録 C

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。

この章で説明する内容は次のとおりです。

asadmin ユーティリティー

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、管理者インタフェース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サブコマンドを呼び出します。サブコマンドは大文字と小文字を区別します。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サブ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。

asadmin 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、対話型オプションまたは非対話型オプションで実行できます。デフォルトでは、対話型オプションが有効になっています。対話型オプションでは、必要な引数を入力するように求められます。対話型オプションは、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 つのサブ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 で対話型オプションを使用できます。multimode のサブコマンド (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サブコマンドは、対話型オプションでは実行できません。

ローカルサブコマンドは、管理サーバーが存在しなくても実行できます。ただし、サブ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。リモートサブコマンドの実行は、管理サーバーに接続してサブ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。すべてのリモートサブコマンドで、次のオプションが必須になります。

- -u --user 認証されたドメインアプリケーションサーバーの管理ユーザー名。

- `--passwordfile` ドメインアプリケーションサーバーのパスワードを格納したファイル。形式は **AS_ADMIN_PASSWORD=***password* です。*password* には、実際の管理者パスワードを指定します。
- `-H --host` ドメインアプリケーションサーバーが稼働しているマシン名。
- `-p --port` 管理要求を待機しているドメインアプリケーションサーバーのポート番号。デフォルトのポート番号は 4848 です。
- `-s --secure` true の場合、SSL/TLS を使用してドメインアプリケーションサーバーと通信します。
- `-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。
- `-I --interactive` true (デフォルト) に設定した場合、必須パスワードオプションのみの入力が要求されます。
- `-h --help` コマンドに関するヘルプテキストが表示されます。

ローカルまたはリモートで実行できるサブコマンドの場合、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、そのサブコマンドはリモートモードで実行されます。さらに、ローカルまたはリモートで実行できるサブコマンドの場合、`--local` オプションが true に設定されていると、そのサブコマンドはローカルで実行されます。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、デフォルトでサブコマンドはローカルで実行されます。`--local` オプションが true に設定されていると、ローカルの `--host`、`--port`、`--user`、および `--passwordfile` の設定が指定されている場合でも、それらより優先されます。サブコマンドはローカルモードで実行されます。

ローカルで実行できるサブコマンドでは、対象のドメインを指定する `--domain` オプションを使用でき、ドメインが 1 つだけの場合は、そのドメインがデフォルトドメインとみなされます。複数のドメインが存在する場合、`--domain` オプションは必須オプションになります。ローカルまたはリモートで実行できるサブコマンドの場合、リモートで `--host`、`--port`、`--user`、および `--passwordfile` オプションを指定して実行すると、`--domain` オプションは無視されます。サブコマンドがリモートモードで実行される場合、`--domain` オプションは無視されます。ドメインごとに 1 つの管理インスタンスがあるため、1 つのマシンに複数のドメインがある場合、ローカルでの実行するときはドメインを指定し、リモートで実行するときはそのドメインの管理インスタンスの `--host`、`--port`、`--user`、および `--passwordfile` オプションを指定してください。

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、ファイルからサブ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 C–1 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
```

`AS_ADMIN_PASSWORD` がグローバル環境にエクスポートされている場合、`--passwordfile` オプションを指定すると、`--password` オプションの使用に関する警告が表示されます。この警告が表示されないようにするには、**`AS_ADMIN_PASSWORD`** の設定を取り消します。マスターパスワードは、コマンド行または環境変数では伝達されませんが、`passwordfile` で指定できます。

`--secure` オプションを使用するには、`set` コマンドを使用して、`domain.xml` 内の `admin http-listener` でセキュリティーの `--enabled` フラグを有効にします。`asadmin` のサブコマンドを使用して作成や削除を行った場合、新しく作成したコマンドを有効にするには、サーバーを再起動する必要があります。サーバーを再起動するには、`start-domain` コマンドを使用します。

Solaris プラットフォーム上の Application Server のコマンド行インタフェースサブ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` ユーティリティーサブ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。サブコマンドを指定すると、そのサブコマンドの使用法が表示されます。サブコマンドを指定せずに `--help` オプションを実行すると、使用可能なすべてのサブコマンドの一覧が表示されます。

リモートコマンドの共通オプション

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 C–1 リモートコマンドの必須オプション

オプション	定義
`--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。

表C–1 リモートコマンドの必須オプション (続き)

オプション	定義
--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。デフォルトのポート番号は4848です。
--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_MAPPEDPASSWORD、AS_ADMIN_USERPASSWORD、AS_ADMIN_ALIASPASSWORD 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。
--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。
--interactive	true (デフォルト) に設定した場合、必須パスワードオプションのみの入力が要求されます。
--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。
--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。

表 C–1 リモートコマンドの必須オプション (続き)

オプション	定義
--help	コマンドに関するヘルプテキストが表示されます。

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。

get、set、list 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、configuration と monitoring 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。

configuration 階層は、変更可能な属性を提供します。一方、monitoring 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。configuration 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、 monitoring 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。ナビゲーション可能な階層および管理コンポーネントの詳細な説明については、例を参照してください。

アプリケーションサーバー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。
- 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 (monitoring と configuration) 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、抽象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、appserver 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C–2 サーバーのライフサイクルコマンド

コマンド	定義
`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。アプリケーションサーバー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`スクリプト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。

表C–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
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。
create-instance	ローカルまたはリモートマシン上に新しいサーバーインスタンスを作成します。
delete-instance	サーバーインスタンスを削除します。このコマンドは、リモートまたはローカルで実行できます。ユーザーの認証には、管理サーバー用に指定されたパスワードを使用します。また、削除するインスタンスは、管理サーバーが処理するドメイン内にすでに存在していなければなりません。削除操作は取り消せないため、このコマンドは慎重に使用してください。

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C–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。

表 C–3 リストおよびステータスコマンド (続き)

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。
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設定解除するか、マルチモード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。

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 C–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。
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。

表 C–4 配備コマンド (続き)

コマンド	定義
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。

バージョンコマンド

バージョンコマンドを使用すると、バージョン文字列を返したり、すべての asadmin コマンドを一覧表示したり、ライセンスファイルをインストールしたりできます。

表 C–5 バージョンコマンド

コマンド	定義
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。
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。
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。

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 C–6 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。

表 C–6 Message Queue コマンド (続き)

コマンド	定義
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 C–7 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。

表 C–7 リソース管理コマンド (続き)

コマンド	定義
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。
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作成します。
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。

表 C–7 リソース管理コマンド (続き)

コマンド	定義
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、あと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、エンタープライズサーバー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。

設定コマンド

設定コマンドを使用すると、IIOP リスナー、ライフサイクルモジュール、HTTP および HTTP リスナー、プロファイラ、およびその他のサブシステムを構築できます。

ここでは、次の内容について説明します。

HTTP および IIOP 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 C–8 IIOP リスナーコマンド

コマンド	定義
create-http-listener	新しい HTTP 待機ソケットを追加します。
delete-http-listener	指定した HTTP リスナーを削除します。

表 C–8 IIOP リスナーコマンド (続き)

コマンド	定義
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。

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 C–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、アプリケーションサーバー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。
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 SSL コマンド

プロファイラおよび SSL コマンドを使用すると、プロファイラおよび SSL クライアント設定を管理できます。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 C–10 プロファイラおよびSSL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-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。

JVM オプションおよび仮想サーバーコマンド

JVM オプションおよび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 C–11 JVM オプションおよび仮想サーバーコマンド

コマンド	定義
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。
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。

スレッドプールおよび認証レルムコマンド

スレッドプールおよび認証レルム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C–12 スレッドプールおよび認証レルムコマンド

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。
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。

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C–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。

表 C–13 トランザクションコマンド (続き)

コマンド	定義
unpublish-from-registry	
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。

レジストリコマンド

レジストリコマンドを使用すると、Web サービスのアーティファクトを発行または発行解除できます。

表 C–14 レジストリコマンド

コマンド	定義
publish-to-registry	レジストリに 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	

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 C–15 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。

表C–15 ユーザー管理コマンド (続き)

コマンド	定義
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ファイルユーザーおよびグループを管理します。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。

ルールおよび監視コマンド

ルールおよび監視コマンドを使用すると、規則を管理し、サーバーを監視できます。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C–16 ルールおよび監視コマンド

コマンド	定義
create-management-rule	アプリケーションサーバー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。
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、要求または応答に適用できます。
delete-transformation-rule	指定した Web サービスの XSLT 変換規則を削除します。
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。

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 C–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、アプリケーションサーバー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 C–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。
`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 C–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 サービスの監視属性または `maxhistory` 属性を設定します。

表 C–19 Webサービスコマンド (続き)

コマンド	定義
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、要求または応答に適用できます。
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 サービスレジス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 C–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。

表 C–20 セキュリティーコマンド (続き)

コマンド	定義
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-map	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。

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、アプリケーションサーバーのセキュリティーを確保することができます。

表 C–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。

表 C–21 パスワードコマンド (続き)

コマンド	定義
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。

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 C–22 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 C–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 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。

サービスコマンド

サービスコマンドを使用すると、ドメイン管理サーバー (DAS) の起動を設定できます。

表 C–24 サービスコマンド

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。

プロパティーコマンド

共有サーバーインスタンスでは、参照される設定に定義された属性の上書きが頻繁に必要になります。サーバーインスタンスの任意の設定属性を、対応する名前のシステムプロパティーによって上書きできます。システムプロパティーコマンドを使用して、これらの共有サーバーインスタンスを管理します。

表 C–25 プロパティーコマンド

コマンド	定義
create-system-property	ドメイン、設定、またはサーバーインスタンスのシステムプロパティーを一度に 1 つずつ作成します。
delete-system-property	ドメイン、設定、またはサーバーインスタンスのシステムプロパティーを 1 つずつ削除します。
list-system-properties	ドメイン、設定、またはサーバーインスタンスのシステムプロパティーを表示します。

索引

れ

ろ