管理者ガイド

Sun™ ONE Application Server

Version 7, Update 1

816-6857-10
2003 年 3 月

目 次

本書について

このマニュアルでは、Sun™ ONE Application Server 7 の設定および管理の方法について説明します。このマニュアルの対象読者は、WWW 経由でより多くの顧客にクライアントサーバーアプリケーションを提供しようと考えている企業の IT 管理者です。

この章には次の節があります。

- マニュアルの概要
- マニュアルの構成
- マニュアルの表記規則
- 製品ラインの概要
- マニュアルの使用法
- 製品サポート

マニュアルの概要

このマニュアルでは、Sun ONE Application Server の設定および管理の方法について説明します。サーバーの設定後、このマニュアルを使ってサーバーの保守を行います。

マニュアルの構成

このマニュアルは、4 部構成になっています。巻末には索引が付いています。最初の第 1 部「サーバーの基礎知識とグローバル設定の管理」では、製品の概要を説明します。第 2 部「サーバーインスタンスの管理」では、管理サーバーの使用方法や、すべてのサーバーインスタンスに影響を及ぼすその他のサーバー機能の使用方法を紹介します。

管理サーバーの基本的な使用方法を理解したら、第 3 部「HTTP サーバーの機能と仮想サーバーの管理」でプログラムと設定スタイルの使用方法を参照してください。

最後の「付録」では、国際化の問題、サーバーの拡張機能、Sun ONE Application Server コマンド行インタフェースなど、さまざまな項目を取り上げて説明します。

第 1 部：サーバーの基礎知識とグローバル設定の管理

Sun ONE Application Server の概要を説明します。次の各章で構成されています。

- 第 1 章「Sun ONE Application Server 管理入門」では、Sun ONE Application Server の概要を説明します。
- 第 2 章「管理サーバーの設定」では、管理サーバーの管理方法を説明します。
- 第 3 章「管理ドメインの設定」では、複数ドメインの使用方法を説明します。

第 2 部：サーバーインスタンスの管理

サーバーインスタンスの設定方法、管理方法、および使用方法を概念的な側面と実際的な側面から詳しく説明します。次の各章で構成されています。

- 第 4 章「アプリケーションサーバーインスタンスの使用」では、Sun ONE Application Server のサーバー設定の設定方法を説明します。
- 第 5 章「ログの使用」では、基本的なログ機能と、Sun ONE Application Server で使用できるログの特徴および機能について説明します。
- 第 6 章「Sun ONE Application Server の監視」では、監視方法と、Sun ONE Application Server で使用できる SNMP (Simple Network Management Protocol) の特徴と機能について説明します。
- 第 7 章「Web サーバープラグインの設定」では、Sun ONE Application Server による HTTP 要求の処理方法と、Sun ONE Application Server での Web サーバープラグインの設定方法および使用方法を説明します。

- 第 8 章「J2EE コンテナの設定」では、EJB (Enterprise Java Beans) や MDB (Message Driven Bean : メッセージ駆動型 Bean) などの J2EE アプリケーションコンポーネントの実行時サポートを提供するコンテナの設定方法および使用方法を説明します。
- 第 9 章「トランザクションサービスの使用」では、データベーストランザクションとその使用方法および管理方法を説明します。
- 第 10 章「ネーミングとリソースの設定」では、J2EE リソースの設定方法を説明します。
- 第 11 章「JMS サービスの使用」では、ネイティブの JMS プロバイダである Sun ONE Message Queue に組み込まれた JMS サービスについて説明します。さらに、このサービスを管理するために必要な情報を提供します。
- 第 12 章「Corba/IIOP クライアント用のサーバーの設定」では、Sun ONE Application Server 環境で、RMI/IIOP プロトコルを使って CORBA ベースのクライアントをサポートする方法を説明します。
- 第 13 章「アプリケーションの配備」では、Sun ONE Application Server にアプリケーションを配備する方法を説明します。

第 3 部 : HTTP サーバーの機能と仮想サーバーの管理

プログラムや設定スタイルの管理インタフェースの使用に関する情報を提供します。次の各章で構成されています。

- 第 14 章「HTTP 機能の設定」では、Sun ONE Application Server の HTTP 関連機能の設定方法を説明します。
- 第 15 章「仮想サーバーの使用」では、Sun ONE Application Server による仮想サーバーの設定方法および管理方法を説明します。
- 第 16 章「仮想サーバーコンテンツの管理」では、サーバーコンテンツの設定方法および管理方法を説明します。

第 4 部 : 付録

参考になる情報を提供します。次の各付録で構成されています。

- 付録 A「コマンド行インタフェースの使用」では、ユーザーインタフェース画面の代わりにコマンド行ユーティリティを使用する方法を説明します。

- 付録 B「サードパーティ製品の著作権について」では、著作権に関する追加情報を提供します。

マニュアルの表記規則

この節では、このマニュアルの表記規則について説明します。

- 一般的な表記規則
- ディレクトリ名の表記規則

一般的な表記規則

このマニュアルは、次の表記規則に従っています。

- **ファイルとディレクトリのパス**は、UNIX の形式で表記します (ディレクトリ名を「/」記号で区切って表記)。Windows バージョンでは、ディレクトリパスについては UNIX と同じですが、ディレクトリの区切り記号にはスラッシュではなく円記号を使用します。
- URL は次の書式で記述します。

  http://*server.domain*/*path*/*file*.html

  server はアプリケーションを実行するサーバー名、domain はユーザーのインターネットドメイン名、path はサーバー上のディレクトリの構造、file は個別のファイル名を示します。URL の斜体文字の部分は可変部分です。
- **フォント**は、次のように使い分けます。
  - モノスペースフォントは、サンプルコード、コードの一覧表示、API および言語要素 (関数名、クラス名など)、ファイル名、パス名、ディレクトリ名、および HTML タグに使います。
  - 斜体文字はコード変数に使います。
  - 斜体文字は、変数および可変部分、およびリテラルに使われる文字にも使います。
  - **太字**は、段落の先頭またはリテラルに使われる文字の強調に使います。
- このマニュアルでは、ほとんどのプラットフォームの**インストールルートディレクトリ**を *install_dir* と記述します。例外については、21 ページの「ディレクトリ名の表記規則」を参照してください。

  デフォルトでは、**ほとんどの**プラットフォームの *install_dir* は次の場所になります。

- Solaris 8 のパッケージベースでない評価バージョンのインストール :

  ユーザーのホームディレクトリ `/sun/appserver7`

- Solaris にバンドルされていない評価用以外のバージョンのインストール :

  `/opt/SUNWappserver7`

- Windows のインストール :

  `C:¥Sun¥AppServer7`

上記のプラットフォームで *default_config_dir* および *install_config_dir* は、*install_dir* と同義です。例外と追加情報については、21 ページの「ディレクトリ名の表記規則」を参照してください。

- このマニュアルでは、**インスタンスルートディレクトリ**は、*instance_dir* と記述します。これは以下のパスの省略形式です。

  default_config_dir/*domains*/*domain*/*instance*

- このマニュアルを通じて、特に明記のない限り、すべての **UNIX 固有の表記**は、Linux オペレーティングシステムにも適用されます。

ディレクトリ名の表記規則

Solaris 8 および 9 のパッケージに含まれる製品のインストール、および Solaris 9 バンドル版のインストールでは、アプリケーションサーバーのファイルはデフォルトで複数のルートディレクトリにまたがって保存されます。ここでは、これらのディレクトリについて説明します。

- **Solaris 9 バンドル版のインストール**では、デフォルトのインストールディレクトリは次のように表記されます。

  - *install_dir* は `/usr/appserver/` を示します。このディレクトリにはインストールイメージの静的な要素が保存されます。ユーティリティ、実行可能ファイル、およびアプリケーションサーバーを構成するライブラリは、すべてここに保存されます。
  - *default_config_dir* は `/var/appserver/domains` を示します。このディレクトリは、作成したドメインのデフォルトの保存場所です。
  - *install_config_dir* は `/etc/appserver/config` を示します。このディレクトリには、ライセンスなどのインストール全体に適用される設定情報や、このインストール用に設定した管理ドメインのマスターリストが保存されます。

- **Solaris 8 および 9 パッケージベースのアンバンドルのインストール (評価バージョン以外)** では、デフォルトのインストールディレクトリは次のように表記されます。

- *install_dir* は /opt/SUNWappserver7 を示します。このディレクトリにはインストールイメージの静的な要素が保存されます。ユーティリティ、実行可能ファイル、およびアプリケーションサーバーを構成するライブラリは、すべてここに保存されます。
- *default_config_dir* は /var/opt/SUNWappserver7/domains を示します。このディレクトリは、作成したドメインのデフォルトの保存場所です。
- *install_config_dir* は /etc/opt/SUNWappserver7/config を示します。このディレクトリには、ライセンスなどのインストール全体に適用される設定情報や、このインストール用に設定した管理ドメインのマスターリストが保存されます。

製品ラインの概要

Sun ONE Application Server 7 は J2EE 1.3 仕様に準拠したアプリケーションサーバーです。また、新しい Java Web Service 規格や標準の HTTP サーバープログラミング機能もサポートしています。本稼動環境と開発環境の両方に幅広く対応するため、次の 3 種類のアプリケーションサーバーが用意されています。

- Platform Edition
- Standard Edition
- Enterprise Edition

Platform Edition

Platform Edition は、Sun ONE Application Server 7 製品ラインの中核を成す製品です。この製品は無償で提供されます。J2EE 1.3 仕様に準拠した高パフォーマンスで小さな実行時環境を備えているため、基礎的な運用開発に適しています。また、こうした実行時環境をサードパーティアプリケーションに組み込むこともできます。Web サービス対応の Platform Edition には、Sun ONE Web Server や Sun ONE Message Queue で実証済みの技術が組み込まれています。

Platform Edition の配備先は、単一アプリケーションサーバーインスタンスに限られています。このアプリケーションサーバーインスタンスは、Java プラットフォーム 用の仮想マシン、すなわち Java 仮想マシン (JVM™) になります。Platform Edition では、複数層の配備トポロジがサポートされます。ただし、Web サーバー層のプロキシはロードバランスを行いません。さらに、管理ユーティリティを使用できるのは、ローカルクライアントに限定されています。

Platform Edition は、Solaris 9 に統合されています。

Standard Edition

この『入門ガイド』で、対象としているエディションです。Platform Edition の機能に加えて、拡張されたリモート管理機能が追加されています。拡張された管理機能、リモートコマンド行、および Web ベースの管理機能はすべて、Standard Edition に組み込まれています。また、Web サーバー層のプロキシにより Web アプリケーションのトラフィックを分割する機能もあります。Standard Edition では、1 台のマシンに複数のアプリケーションサーバーインスタンス (JVM) を設定できます。

Enterprise Edition

Enterprise Edition では、アプリケーションサーバープラットフォームの基本機能に、高可用性機能、ロードバランス機能、およびクラスタ機能が追加されています。これにより、要求の多い J2EE ベースのアプリケーションもスムーズに配備できます。Standard Edition より高度な管理機能により、複数のインスタンスの配備、複数のマシンへの配備にも対応しています。

クラスタリング機能では、複数のアプリケーションサーバーインスタンスの複製をグループとして構成し、クライアント要求のロードバランスを行うことができます。Enterprise Edition では、外部ロードバランサとロードバランスを行う Web 層ベースのプロキシが両方ともサポートされます。また、HTTP セッション、ステートフルセッション Bean インスタンス、および Java Message Service (JMS) リソースのフェイルオーバー機能があります。「Always On (常時配信)」という独自の高可用性データベーステクノロジを、高可用性 (HA) 持続ストアの基盤として採用しています。

製品の詳細は、Sun Microsystems の Web サイト の Sun ONE Application Server のページを参照してください。

マニュアルの使用法

このマニュアルは、PDF 形式または HTML 形式でも入手できます。

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

次の表は、Sun ONE Application Server のマニュアルに記述されているタスクと概念を示しています。左側の列にタスクと概念、右側の列に参照するマニュアルを示します。

Sun ONE Application Server マニュアルの概要

情報の内容	参照するマニュアル
ソフトウェアおよびマニュアルの最新情報	リリースノート
サポート対象のプラットフォームと環境	プラットフォーム
アプリケーションサーバーの紹介。アプリケーションサーバーの新機能、評価 (Evaluation) バージョンのインストール、アーキテクチャの概要など	入門ガイド
Sun ONE Application Server とそのコンポーネント (サンプルアプリケーション、管理インタフェース、Sun ONE Message Queue など) のインストール	インストールガイド
Sun ONE Application Server 7 の Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。アプリケーション設計、開発ツール、セキュリティ、アセンブリ、配備、デバッグ、ライフサイクルモジュールの作成に関する情報など	開発者ガイド
Sun ONE Application Server 7 の Web アプリケーション向け Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。Web アプリケーションプログラミングの概念とタスクの説明、サンプルコード、実装のヒント、関連資料の紹介など	Web アプリケーション開発者ガイド
Sun ONE Application Server 7 のエンタープライズ Bean 向け Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。EJB プログラミングの概念とタスクの説明、サンプルコード、実装のヒント、関連資料の紹介など	Enterprise JavaBeans 開発者ガイド
Sun ONE Application Server 7 上で J2EE アプリケーションにアクセスするクライアントの作成	Developer's Guide to Clients
Web サービスの作成	Developer's Guide to Web Services

Sun ONE Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル
JDBC、JNDI、JTS、JMS、JavaMial、リソース、コネクタなどの J2EE 機能	Developer's Guide to J2EE Features and Services
カスタム NSAPI プラグインの作成方法	NSAPI Developer's Guide
次の管理タスクの実行 • 管理インタフェースとコマンド行インタフェースの使用 • サーバーの作業環境の設定 • 管理ドメインの使用 • サーバーインスタンスの使用 • サーバーの稼動状況の監視およびログ記録 • Web サーバープラグインの設定 • Java Messaging Service の設定 • J2EE 機能の使用 • CORBA ベースのクライアント機能の設定 • データベース接続の設定 • トランザクション管理の設定 • Web コンテナの設定 • アプリケーションの配備 • 仮想サーバーの管理	管理者ガイド
サーバー設定ファイルの編集	管理者用設定ファイルリファレンス
Sun ONE Application Server 7 運用環境のセキュリティの設定および管理。一般的なセキュリティ、証明書、および SSL/TLS 暗号化に関する情報など。HTTP サーバーベースのセキュリティについても説明	セキュリティ管理者ガイド
Sun ONE Application Server 7 用の J2EE CA コネクタのサービスプロバイダ実装の設定と管理。管理ツール、DTD に関する情報やサンプル XML ファイルなど	J2EE CA Service Provider Implementation Administrator's Guide
Netscape Application Server バージョン 2.1 から新しい Sun ONE Application Server 7 プログラミングモデルへのアプリケーションの移行。Sun ONE Application Server に付属するオンラインバンクアプリケーションの移行サンプルなど	サーバーアプリケーションの移行および再配備

Sun ONE Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル
Sun ONE Message Queue の使用法	Sun ONE Message Queue については次の URL を参照 : http://docs.sun.com/

製品サポート

ご使用のシステムに問題が発生した場合は、次のいずれかの方法でカスタマサポートにお問い合わせください。

- 次のオンラインサポート Web サイトをご利用ください。

  `http://www.sun.com/supportraining/`

- 保守契約を結んでいるお客様の場合は、専用ダイヤルをご利用ください。

サポートのご依頼の前に、次の情報を用意してください。サポート担当がお客様の問題を解決するために必要な情報です。

- 問題が発生した箇所や動作への影響など、問題の具体的な説明
- マシン機種、OS バージョン、および、問題の原因と思われるパッチやそのほかのソフトウェアなどの製品バージョン
- 問題を再現するための具体的な手順の説明
- エラーログやコアダンプ

第 1 部

サーバーの基礎知識とグローバル設定の管理

第 1 章「Sun ONE Application Server 管理入門」

第 2 章「管理サーバーの設定」

第 3 章「管理ドメインの設定」

第 1 章

Sun ONE Application Server 管理入門

この章では、Sun ONE Application Server の管理の基本について説明します。サーバーに関する一般的な情報、サーバーのユーザーインタフェースにアクセスする方法、およびオンラインヘルプにアクセスする方法について取り上げます。

この章には次の節が含まれます。

- Sun ONE Application Server について
- Solaris バンドル版の設定
- 管理インタフェースの使用
- コマンド行インタフェースの使用
- 管理サーバーへのアクセス
- アプリケーションサーバーインスタンスへのアクセス
- Sun ONE Studio の使用
- 設定ファイルについて
- ライセンスコマンドの使用

Sun ONE Application Server について

Sun ONE Application Server は、多種多様なサーバー、クライアント、およびデバイスを対象とした電子商取引アプリケーションサービスを開発、配備、管理するための堅牢な J2EE プラットフォームです。主な機能に、トランザクション管理、パフォーマンス、スケーラビリティ、セキュリティ、およびアプリケーションの統合があります。

Sun ONE Application Server は、Web パブリッシングから企業規模のトランザクション処理までを幅広くサポートします。一方、開発者は Sun ONE Application Server を利用して、JavaServer Pages (JSPTM)、Java サーブレット、Enterprise Java BeansTM (EJBTM) 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。

Sun ONE Application Server は次の管理者用の基本ツールを含みます。

- 複数の管理ドメイン。これにより、複数の管理者がそれぞれ独自のアプリケーションサーバーインスタンスセットを作成および管理できる
- 管理機能を提供する管理サーバー (各ドメインに 1 つ)
- グラフィカルユーザーインタフェース (管理インタフェース)。サーバー管理に使用する
- コマンド行インタフェース。管理インタフェース と同じタスクを実行できる

これらのツールを使用して、次の管理機能を実行できます。

- ドメインの管理
- サーバーインスタンスの管理
- アプリケーションの配備
- サーバーの監視
- ログファイルの使用
- リソースの管理
- Message Queue サーバーの管理
- トランザクションサービスの使用
- Corba/IIOP クライアントの使用
- Web サーバープラグインの設定
- J2EE コンテナの設定
- HTTP サーバー機能の管理

Sun ONE Application Server のアーキテクチャと機能、および初期手順の詳細については、『Sun ONE Application Server 入門ガイド』を参照してください。

Solaris バンドル版の設定

このマニュアルは、Sun ONE Application Server の 2 種類のインストール、つまり Solaris 9 バンドル版とアンバンドル版を対象とします。Solaris 9 インストールの一部として入手した Sun ONE Application Server は、Solaris バンドル版です。スタンドアロンコピーの Sun ONE Application Server は、アンバンドル版です。

注 Sun ONE Application Server の Solaris アンバンドル版または Windows 版を使用している場合は、この節を参照せずに 33 ページの「管理インタフェースの使用」に進んでください。

Sun ONE Application Server の Solaris 9 バンドル版を使用している場合は、追加の設定を行なってから、サーバーを起動する必要があります。

Sun ONE Application Server のアンバンドル版をインストールすると、インストールプロセスの一部として、ドメイン、管理サーバー、およびサーバーインスタンスが自動的に作成されます。

Solaris 9 バンドル版では、他の手順と同時に、これらのアイテムを手動で作成する必要があります。これらの初期手順を行うと、その他の管理ドメインやサーバーインスタンスを追加することを含め、Sun ONE Application Server のすべての機能を利用できるようになります。

この節では次の項目について説明します。

- 管理ドメインの作成
- 管理サーバーの起動
- アプリケーションサーバーインスタンスの作成
- アプリケーションの配備

管理ドメインの作成

複数の管理ドメインを作成すると、複数の管理ユーザーがそれぞれ独自のドメインを作成し、管理できます。ドメインはインスタンスのセットで、1 つのシステムにインストールされたバイナリの共通セットから作成されます。各ドメインには、管理サーバーが 1 つずつあります。

新しいドメインの作成時には、次の項目を指定します。

- 管理サーバーのポート番号。アンバンドル版をインストールした場合のデフォルト値は 4848

- 管理ユーザーの名前とパスワード。パスワードは、管理インタフェース へのアクセス、またはコマンド行インタフェースの実行のいずれかで、管理者が管理サーバーにアクセスする場合に必要となる
- ドメインの位置

ドメインの作成には、コマンド行インタフェースで `asadmin` ユーティリティの `create-domain` コマンドを使用する必要があります。管理ドメインの作成方法の詳細については、第 3 章「管理ドメインの設定」を参照してください。

管理サーバーの起動

管理ドメインの作成時に、管理サーバーが作成されます。管理サーバーは Sun ONE Application Server の特殊なインスタンスで、管理インタフェースとコマンド行インタフェースの管理機能を提供します。

管理インタフェース、またはコマンド行インタフェースの多くのコマンドを使用するには、管理サーバーが実行中である必要があります。管理サーバーの起動方法の詳細については、45 ページの「管理サーバーの起動」を参照してください。

アプリケーションサーバーインスタンスの作成

ドメインを作成し、管理サーバーを起動した後には、アプリケーションサーバーインスタンスを作成する必要があります。個々のアプリケーションサーバーインスタンスの J2EE 設定、J2EE リソース、アプリケーション配備領域、サーバー構成設定は独立しています。

アプリケーションサーバーインスタンスの作成には、管理インタフェースまたはコマンド行インタフェースを使用します。サーバーインスタンスは、ドメイン内のフォルダに作成されます。

アンバンドル版では、`server1` と呼ばれるサーバーインスタンスが、インストール時に作成されます。このマニュアル内の例では、`server1` が多く登場します。

アプリケーションサーバーインスタンスの作成方法の詳細については、76 ページの「アプリケーションサーバーインスタンスの追加」を参照してください。

アプリケーションの配備

ドメインの作成、管理サーバーの起動、アプリケーションサーバーインスタンスの追加が終わると、作成したインスタンスにアプリケーションを配備する必要があります。詳細については、第 13 章「アプリケーションの配備」を参照してください。

管理インタフェースの使用

管理インタフェースを使用すると、サーバーのあらゆる部分を設定できます。この節には次の項目があります。

- 管理インタフェースへのアクセス
- タブの使用
- ボタンの使用
- オンラインヘルプへのアクセス
- 管理インタフェースの終了

注 いくつかのサーバー設定とその対応する管理インタフェースは、現在も開発が進んでいます。不安定なインタフェースは、次期製品リリースでより安全で安定したものに置き換えられる可能性があります。ほとんどのサーバー設定および管理インタフェースがそのまま残るか互換性を持ったまま変更されますが、いくつかの点で互換性が保たれない場合もあります。今後のリリースでの製品マニュアルでは、必要に応じて非互換性に関しては明確に記述する予定です。

管理インタフェースへのアクセス

Sun ONE Application Server の管理インタフェースは、管理サーバーと呼ばれる HTTP サーバー上で動作します。アンバンドル版を使用している場合、管理サーバーは、Sun ONE Application Server と同時にインストールされます。バンドル版をインストールする場合は、管理ドメインと管理サーバーを作成する必要があります。詳細については、31 ページの「Solaris バンドル版の設定」を参照してください。

管理インタフェースを使用するには、管理サーバーを実行する必要があります。管理サーバーの起動方法の詳細については、45 ページの「管理サーバーの起動」を参照してください。

Sun ONE Application Server のインストール時またはドメインの作成時には、選択した管理サーバーのポート番号か、デフォルトのポート番号 4848 が使用されます。管理インタフェースにアクセスするには、Web ブラウザで次の URL を指定します。

http://*hostname.domain*:*port*/

次に例を示します。

http://austen.sun.com:4848/

Sun ONE Application Server がインストールされているマシンから Application Server にアクセスするときは、次の URL を使用できます。

http://*localhost*:4848

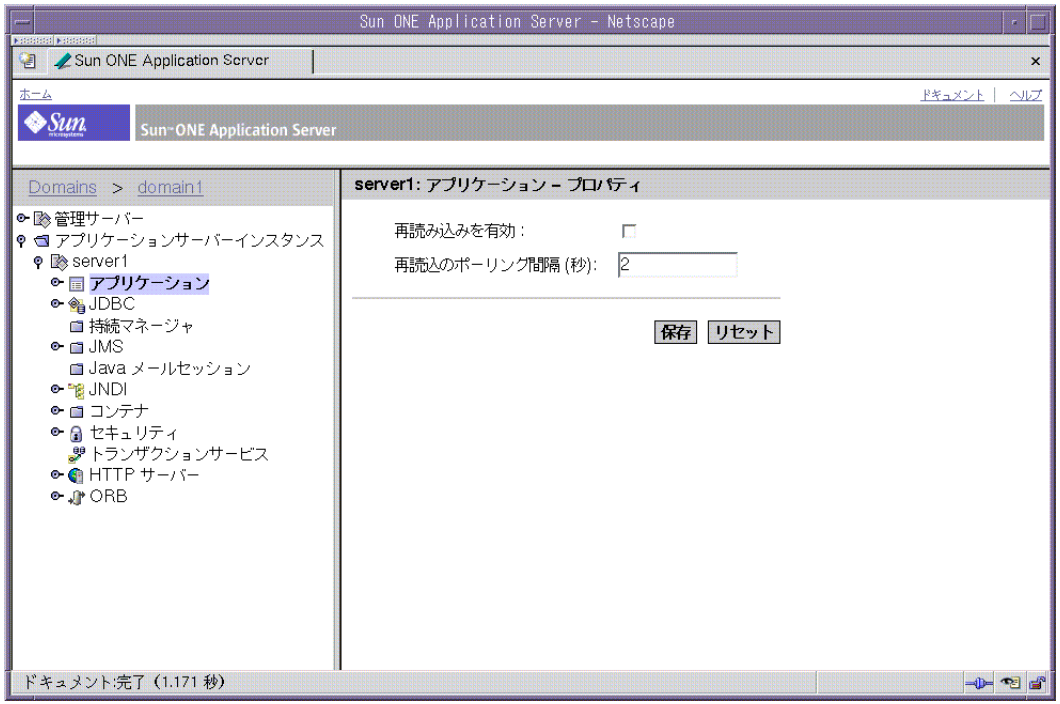
プロンプトが表示されたら、インストール時に設定したユーザー名とパスワード、またはドメインと管理サーバーを作成したときに設定したユーザー名とパスワードを入力します。

ブラウザにアクセスできる限り、リモートで管理インタフェースにアクセスできます。ネットワークを介してサーバーにアクセスできれば、どのマシンからでもアクセスできます。

Windows では、「スタート」メニューから「プログラム」、「Sun Microsystems」、「Sun ONE Application Server 7」、「Start Admin Console」の順に選ぶと、管理インタフェースにアクセスできます。

次の図は、管理インタフェースを示しています。

Sun ONE Application Server の管理インタフェース



左側のペインには、Sun ONE Application Server で設定できるすべての項目がツリー表示されます。管理インタフェースを使用するには、左側のペインから、どれか 1 つの項目をクリックします。右側のペインに、クリックした項目に対応するページが表示されます。

左側のペインの項目の横に展開または折りたたみの記号が表示されている場合、この記号をクリックすると、項目が展開され、サブ項目が表示されます。ツリー項目が展開されていないときは次の記号が表示されます。

折りたたみ記号

ツリー項目が展開されているときは次の記号が表示されます。

展開記号

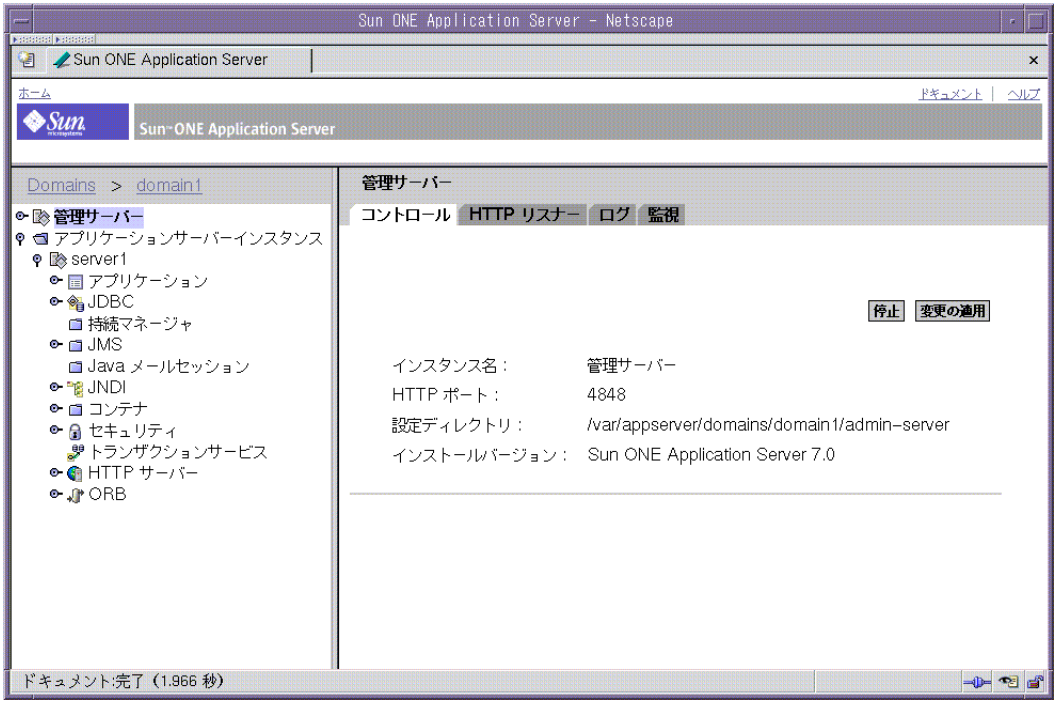
タブの使用

管理インタフェースには、その他のページに移動するタブがあります。これらのタブは、右側のペインの最上部にある、独立したペインに表示されます。

タブを使用するには、タブ名をクリックします。この操作で、別のページに直接移動できる場合と、このタブから移動できるページのリストがタブ名の下に表示される場合があります。ページのリストが表示される場合は、いずれかのページ名をクリックすると、そのページに移動できます。

次の図は、タブが表示されている管理インタフェースを示しています。

タブが表示されている管理インタフェース



ボタンの使用

管理インタフェースでは、次に示す標準ボタンを使用できます。

管理インタフェースの標準ボタン

ボタン	実行される内容
取り消し	変更内容を保存しないで前のページに戻る
削除	項目の削除。削除する項目は、項目の横の「選択」をクリックして選択
新規	新しい項目を作成するページを表示。たとえば、「アプリケーションサーバーインスタンス」ページで「新規」をクリックすると、「新しいインスタンスを作成する」ページが表示される
了解	入力した内容の保存。Sun ONE Application Server の設定に対する変更を有効にするには、変更内容を適用することが必要 詳細については、78 ページの「アプリケーションサーバーインスタンスの変更の適用」を参照

管理インタフェースの標準ボタン (続き)

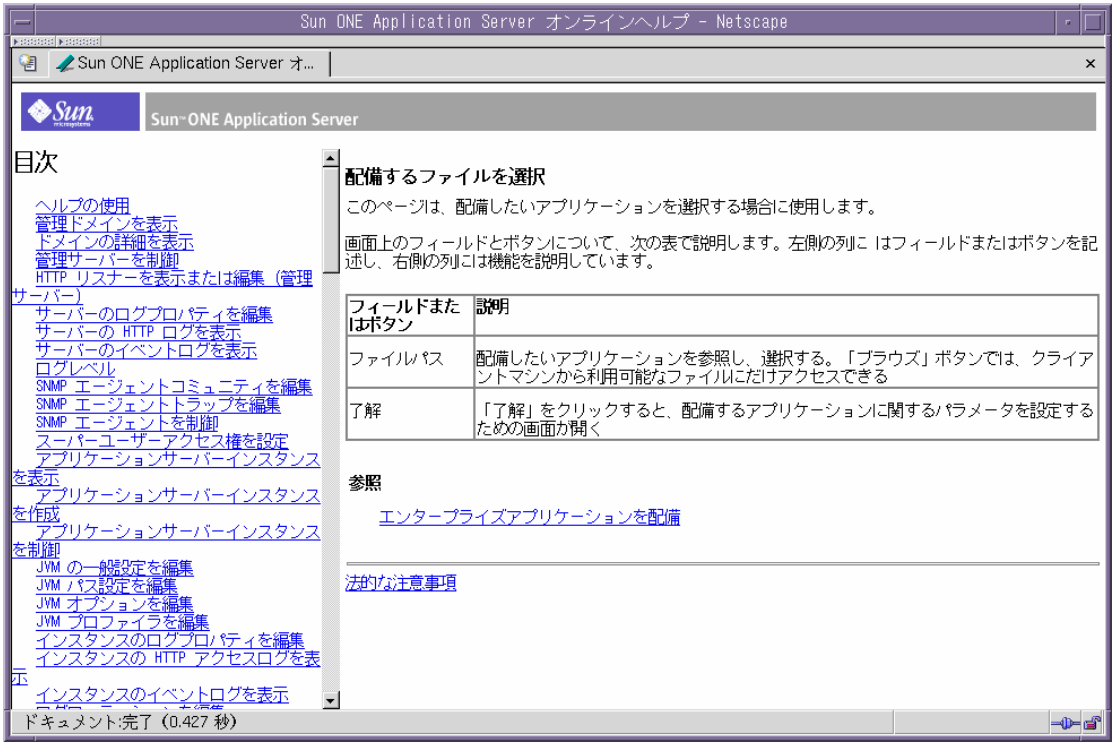
ボタン	実行される内容
リセット	ページに表示されている値をデフォルト値に戻す
保存	入力した内容の保存。Sun ONE Application Server の設定に対する変更を有効にするには、変更内容を適用することが必要 詳細については、78 ページの「アプリケーションサーバーインスタンスの変更の適用」を参照

このほか、画面ごとに必要なボタンを使用できます。

オンラインヘルプへのアクセス

管理インタフェースの各ページに関するヘルプには、管理インタフェースの最上部のバナーにある「ヘルプ」ボタンをクリックするとアクセスできます。オンラインヘルプには、表示されているページの使用方法や、ページ内のフィールドに入力する内容についての説明が記載されています。

オンラインヘルプ



ヘルプウィンドウの左側のペインにある目次から、その他のページのヘルプも表示できます。ヘルプの使用方法については、オンラインヘルプ目次の最初のトピック「ヘルプを使用する」を選択してください。

管理インタフェースの終了

管理インタフェースを終了またはログアウトするボタンはありません。終了するには、管理インタフェースへのアクセスに使用しているブラウザを閉じます。また、マシン上で実行されている同じブラウザのその他のインスタンスを閉じます。

コマンド行インタフェースの使用

Sun ONE Application Server には、コマンド行インタフェースが付属しています。使用できるのは、asadmin ユーティリティとその関連コマンドです。これらのコマンド行インタフェースでは、管理インタフェースで実行するタスクをすべて実行できます。たとえば、アプリケーションサーバーインスタンスの起動と停止、サーバーの設定、およびアプリケーションの配備といったタスクがあります。

コマンド行インタフェースは、シェル内のコマンドプロンプトから使用できるほか、スクリプトやプログラムから呼び出すこともできます。これらのコマンドを使って、繰り返し実行する管理タスクを自動化することもできます。

コマンド行インタフェースの詳しい使用方法とコマンドの一覧については、付録 A「コマンド行インタフェースの使用」を参照してください。

管理サーバーへのアクセス

管理サーバーは、管理インタフェースとその管理機能、およびコマンド行インタフェースを提供する Sun ONE Application Server の特殊なインスタンスです。管理サーバーはこれらインタフェースやツールの設定、配備、および監視の各機能を管理するため、各機能の動作には管理サーバーが動作していることが必要です。

管理サーバーのプロパティを設定するには、管理インタフェースにアクセスします。左側のペインの「管理サーバー」をクリックすると、管理サーバーの設定情報が表示されます。

管理サーバーの詳細については、第 2 章「管理サーバーの設定」を参照してください。

アプリケーションサーバーインスタンスへのアクセス

Sun ONE Application Server のインスタンスは、複数作成できます。アプリケーションサーバーインスタンスごとに、他のアプリケーションサーバーインスタンスに依存しない独自の設定、リソース、アプリケーション配備領域を設けることができます。あるアプリケーションサーバーインスタンスの設定に変更を加えても、別のアプリケーションサーバーインスタンスの設定は変更されません。管理インタフェースには、作成されたすべてのアプリケーションサーバーインスタンスの項目が表示されます。アンバンドル版のインストール時には、アプリケーションサーバーインスタンスが 1 つ作成されます。その後、必要に応じて新しいインスタンスを作成できます。

Solaris 9 バンドル版では、アプリケーションサーバーインスタンスは自動的に作成されません。詳細については、31 ページの「Solaris バンドル版の設定」を参照してください。

アプリケーションサーバーインスタンスの詳細については、第 4 章「アプリケーションサーバーインスタンスの使用」を参照してください。

Sun ONE Studio の使用

アプリケーションを配備するには、管理インタフェースおよびコマンド行インタフェースを使用するほかに、Sun ONE Studio 4 を使用することもできます。Sun ONE Studio の詳細については、『Sun ONE Studio 4, Enterprise Edition for Java with Application Server 7 チュートリアル』を参照してください。このマニュアルは、http://docs.sun.com で見ることができます。

設定ファイルについて

管理インタフェースまたはコマンド行インタフェースによってサーバーの設定に変更を加えると、管理サーバーは基になる設定ファイルを変更します。これらの設定ファイルには、管理サーバーと個々のアプリケーションサーバーインスタンスの設定情報が含まれています。

設定ファイルとその内容の詳細については、『Sun ONE Application Server 管理者用構成ファイルリファレンス』を参照してください。

ライセンスコマンドの使用

Sun ONE Application Server を購入し、インストールすると、該当する種類のライセンスが自動的にインストールされます。ライセンスの種類とそれぞれの制限については、『Sun ONE Application Server インストールガイド』を参照してください。

Sun ONE Application Server には、インストール後にライセンスを管理するためのコマンド行ユーティリティが含まれます。

Windows マシンでは、`asadmin` ユーティリティの `install-license` コマンドを使用すると、インストール後にライセンスをアップグレードできます。このコマンドは、Sun ONE Application Server をインストールしたマシンでローカルに実行する必要があります。ライセンスをインストールする前に、実行中のアプリケーションサーバーをすべて停止する必要があります。

このコマンドの構文は次のとおりです。

```
asadmin install-license
```

Solaris パッケージベースのアンバンドル版では、次の構文を使用してライセンスをインストールします。

`pkgadd -d` *full_path* SUNWaslco

例を示します。

`pkgadd -d /`*install_dir*`/pkg SUNWaslco`

Solaris 9 バンドル版では、次の構文を使用してライセンスをインストールします。

`pkgadd -d` *full_path* SUNWaslc

現在のライセンスに関する情報を取得するには、`display-license` コマンドを使用します。構文は次のとおりです。

`asadmin display-license [--user` *admin_user*`] [--password` *admin_password*`] [--passwordfile` *password_file*`] [--host` *localhost*`] [--port` *admin_port*`] [--local=`*true/false*`]`

このコマンドは、ローカルオプションの値により、ローカルでもリモートでも実行できます。たとえば、ホストとポート番号に関してデフォルト値を使用し、このコマンドをローカルマシンで実行する場合は、次の構文になります。

```
asadmin display-license --local
```

出力には、評価 (Evaluation) バージョンなど現在インストールされているライセンスの種類、期限が設定されている場合は有効期限、ライセンスで許可されている管理サーバーごとのインスタンス数、およびリモート管理が許可されているかどうかについての内容が示されます。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。asadmin の詳しい使用方法については、付録 A「コマンド行インタフェースの使用」を参照してください。

第 2 章

管理サーバーの設定

管理サーバーは、管理インタフェースとその管理機能、およびコマンド行インタフェースを提供する Sun ONE Application Server の特殊なインスタンスです。管理サーバーは、Sun ONE Application Server の設定機能、配備機能、監視機能を管理します。この章では、管理サーバーの設定方法を説明します。

この章では次のトピックについて説明します。

- 管理サーバーについて
- 管理サーバーの起動
- 管理サーバーの停止
- 管理サーバーの設定へのアクセス
- 管理サーバーの制御設定の表示
- 管理サーバーへの変更の適用
- 管理サーバーの HTTP リスナーの設定
- SNMP、ログ、セキュリティのプリファレンスの設定

管理サーバーについて

管理サーバーは、管理インタフェースとコマンド行インタフェースの管理機能を提供する Sun ONE Application Server の特殊なインスタンスです。管理サーバーは、これらのインタフェースの設定機能、配備機能、監視機能を管理します。また、管理インタフェースのページを提供します。管理インタフェースを使用し、ほとんどのコマンドをコマンド行インタフェースから実行する場合には、管理サーバーが稼働している必要があります。

管理サーバーは、Sun ONE Application Server (アンバンドル版) をインストールした場合、またはドメインを新規作成した場合にインストールされます。各ドメインの管理サーバーは 1 つだけですが、管理サーバーの管理対象となるアプリケーションサーバーインスタンスは、複数作成できます。管理サーバーを使って、設定内容や配備されたアプリケーションのほか、個々のアプリケーションサーバーインスタンスのサーバー機能にアクセスできます。

管理ドメインの詳細については、第 3 章「管理ドメインの設定」を参照してください。

Sun ONE Application Server のアンバンドル版を使う場合は、Sun ONE Application Server のインストール時に管理サーバーのポート番号を指定します。指定しない場合はデフォルトの 4848 が適用されます。

Solaris 9 にバンドルされている Sun ONE Application Server を使う場合は、ドメインと管理サーバーを手動で作成する必要があります。Solaris 9 のバンドル版の設定については、31 ページの「Solaris バンドル版の設定」を参照してください。

管理インタフェースにアクセスするときは、Web ブラウザに次のように入力します。

http://*hostname.domain*:*port*/

この *domain* は Sun ONE Application Server の管理ドメインではなく、実際のドメイン名を表します。

次に例を示します。

http://austen.sun.com:4848/

Sun ONE Application Server がインストールされているマシンから Application Server にアクセスするときは、次の URL を使用できます。

http://localhost:4848

ユーザー名とパスワードを入力する必要があります。

管理サーバーの起動

管理サーバーの起動または再起動の方法については、次のいずれかを参照してください。

- startserv スクリプトの使用
- コマンド行インタフェースの使用
- 「サービス」ウィンドウの使用 (Windows の場合)
- 「スタート」メニューの使用 (Windows の場合)

startserv スクリプトの使用

起動スクリプトを使って、サーバーを起動する際、サーバーが 1024 より小さい番号のポートで実行されている場合は root として、それ以外の場合は root またはそのサーバーのユーザーアカウントでログインします (UNIX) 。コマンド行を使って次のディレクトリに移動します。

install_dir/domains/*domain_dir*/admin-server/bin

install_dir はサーバーをインストールしたディレクトリ、*domain_dir* は管理ドメインのディレクトリです。

UNIX 環境では、次のように入力します。

```
./startserv
```

末尾にオプションパラメータ `-i` を指定することもできます。通常、サーバーはバックグラウンド処理として実行されます。`-i` オプションを指定すると、サーバーがバックグラウンド処理に切り替わることはありません。このオプションは、`/etc/inittab` を使ってサーバーを実行している場合に便利です。

Windows 環境では、`startserv.bat` ファイルを実行します。

注	`startserv` コマンドは、サーバーの実行中は失敗します。`startserv` コマンドを実行する前に、サーバーを停止してください。サーバーの起動に失敗する場合は、再起動する前にプロセスを強制終了してください。

コマンド行インタフェースの使用

コマンド行インタフェースの asadmin ユーティリティには、start-domain コマンドが付属しています。このコマンドを使って、Application Server および関連するすべてのアプリケーションサーバーインスタンスを起動できます。このコマンドは、ローカルでしか実行できません。したがって、Sun ONE Application Server をインストールしたマシンで実行する必要があります。コマンド引数は不要です。

start-domain コマンドを使って管理ドメイン内のすべてのインスタンスを起動することもできます。構文は次のとおりです。

```
asadmin start-domain [--domain domain-name]
```

コマンド行インタフェースの操作方法については、コマンド行インタフェースのオンラインヘルプ、および付録 A「コマンド行インタフェースの使用」を参照してください。

「サービス」ウィンドウの使用 (Windows の場合)

Windows の「サービス」コントロールパネルからサーバーを起動するには、次の手順に従います。

1. コントロールパネルの「管理ツール」をクリックします。
2. 「サービス」をクリックします。
3. サービスのリストをスクロールし、「Application Server 7.0 管理サーバー」サービスをダブルクリックします。

   対象ドメインの「管理サーバー」を選択します。「サービス」ウィンドウの「名前」列には、Sun App Server Admin Server として管理サーバーの名前 (*your_domain_name*:admin-server) が表示されます。
4. 「起動」をクリックします。
5. 「了解」をクリックします。

これで、コンピュータの起動時に Application Server 7.0 管理サーバーが自動的に起動するように設定されます。

「スタート」メニューの使用 (Windows の場合)

Windows の「スタート」メニューを使ってサーバーを起動するには、次の手順に従います。

1. 「スタート」メニューから「プログラム」を選択します。

2. 「Sun Microsystems」を選択します。
3. 「Sun ONE Application Server 7」を選択します。
4. 「Start Application Server」をクリックします。

管理サーバーの停止

管理サーバーは、起動のあと継続して実行され、要求を待機したり受け入れたりします。管理サーバーのログ機能の設定や、HTTP リスナーが待機するポートを変更する場合は、いったん管理サーバーを停止して再起動します。

ユーザーが管理サーバーを停止すると、管理サーバーは新しい接続の受け入れを停止します。それから、管理サーバーは未処理の接続がすべて完了するまで待機します。管理サーバーの停止中は、管理インタフェースもコマンド行インタフェースも使用できません。

サーバーを停止する方法については、次のいずれかを参照してください。

- 管理インタフェースの使用
- stopserv スクリプトによる停止
- コマンド行インタフェースによる停止
- 「サービス」ウィンドウによる停止 (Windows の場合)

サーバーの停止後、停止プロセスが完了するまで数秒かかります。

管理インタフェースの使用

管理インタフェースから管理サーバーを停止するには、次の手順に従います。

1. 左側のペインの「管理サーバー」をクリックします。
2. 「コントロール」タブをクリックします。
3. 「停止」をクリックします。

   このリンクをクリックすると、管理サーバーがただちに停止します。新しい画面が表示されることはありません。

stopserv スクリプトによる停止

管理サーバーを手動で停止するには、コマンドプロンプトで次のディレクトリに移動します。

install_dir/domains/*domain_dir*/admin-server/bin

install_dir はサーバーをインストールしたディレクトリ、*domain_dir* はドメインのディレクトリです。

UNIX 環境では、次のように入力します。

```
./stopserv
```

このコマンドは、サーバーを実行しているユーザーとして実行する必要があります。

サーバー起動時に、`*/etc*/inittab` ファイルを使用した場合は、サーバーを停止する前に `*/etc*/inittab` ファイルからサーバーを起動するための行を削除し、`kill -1` を実行します。それ以外の場合は、停止したサーバーは自動的に再起動します。

Windows 環境を使用している場合は、`stopserv.bat` ファイルを実行します。

コマンド行インタフェースによる停止

コマンド行インタフェース `asadmin` ユーティリティを使って管理サーバーを停止することもできます。この場合は、`shutdown` コマンドを実行します。このコマンドはローカルでもリモートでも実行できます。コマンド引数は不要です。

コマンド行インタフェース `asadmin` を使って、管理サーバーと関連するすべてのアプリケーションサーバーインスタンスを停止することもできます。この場合は、`stop-appserv` コマンドを実行します。このコマンドは、ローカルでしか実行できません。したがって、Sun ONE Application Server をインストールしたマシンで実行する必要があります。コマンド引数は不要です。

`stop-domain` コマンドを使ってドメインを停止することで管理サーバーを停止することもできます。このコマンドを実行すると、デフォルトの設定では管理サーバーを含むそのドメインのすべてのインスタンスが停止されます。ドメイン内の、管理サーバーを除くすべてのインスタンスを停止するように設定することもできます。このコマンドの構文は次のとおりです。

`asadmin stop-domain [--user` *admin_user*`] [--password` *admin_password*`] [--host` *admin_host*`] [--port` *admin_port*`] [--local=`*true/false*`] [--domain` *domain_name*`] [--adminserv=`*true/false*`] [--passwordfile` *file_name*`] [--secure | -s]`

local オプションを指定すると、コマンドはローカルで実行されます。
--adminserv=false オプションを指定すると、コマンドを実行しても管理サーバーは停止しません。ただし、デフォルトの設定どおりに --adminserv を true に設定すると、管理サーバーも停止されます。

コマンド行インタフェースの操作方法については、コマンド行インタフェースのオンラインヘルプ、および付録 A「コマンド行インタフェースの使用」を参照してください。

「サービス」ウィンドウによる停止 (Windows の場合)

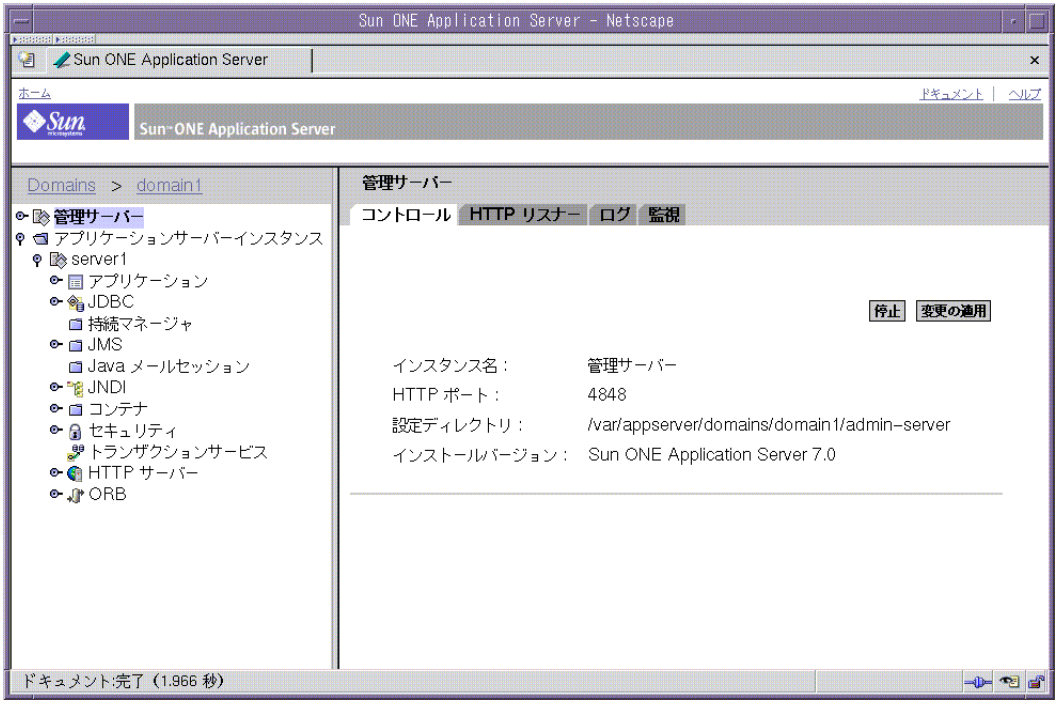
「サービス」ウィンドウから管理サーバーを停止するには、次の手順に従います。

1. コントロールパネルの「管理ツール」をクリックします。
2. 「サービス」をクリックします。
3. サービスのリストをスクロールし、「Sun Application Server 7 Admin Server」サービスをダブルクリックします。
4. 「停止」をクリックします。
5. 「了解」をクリックします。

管理サーバーの設定へのアクセス

管理サーバーの設定にアクセスするには、管理インタフェースの左側のペインの「管理サーバー」をクリックします。右側のペインに管理サーバーの設定情報が表示されます。なお、この情報は、機能別に複数のタブに分けられています。

管理サーバーのユーザーインタフェース



タブをクリックすると、その機能別の設定情報が表示されます。タブをクリックする操作によって、直接ページが表示される場合もあれば、表示できるページのリストが表示される場合もあります。

この章では、「コントロール」タブと「HTTP リスナー」タブについて説明します。監視および SNMP の設定については、第 6 章「Sun ONE Application Server の監視」を参照してください。ログについては、第 5 章「ログの使用」を参照してください。

管理サーバーの制御設定の表示

管理サーバーの制御設定には、インスタンス名 (管理サーバー)、管理サーバーの実行ポート、設定ファイルの格納先ディレクトリ、および実行する Sun ONE Application Server ソフトウェアのバージョン情報があります。

これらの設定を表示するには、次の手順に従います。

1. 左側のペインの「管理サーバー」をクリックします。
2. 「コントロール」タブをクリックします。

管理サーバーへの変更の適用

管理インタフェースやコマンド行インタフェースを使って変更した管理サーバーの設定情報を有効にするには、変更内容を適用する必要があります。この処理は「サーバーの再設定」とも呼ばれます。変更を適用すると、最後に変更を適用したときから現在までの変更内容がすべて有効になります。

変更の適用が必要な管理サーバーの設定情報に変更を加えたときは、左側のペインのツリーに表示される「管理サーバー」の横に黄色のアイコンが表示されます。

警告アイコン

設定の変更を適用するには、次の手順を実行します。

1. 左側のペインの「管理サーバー」をクリックします。
2. 「コントロール」タブをクリックします。
3. 「変更の適用」をクリックします。

   変更が適用されると、画面にメッセージが表示されます。

管理サーバーの HTTP リスナーの設定

サーバーは、HTTP リスナーから受け取った要求を処理します。

Sun ONE Application Server のアンバンドルバージョンでは、管理サーバーの HTTP リスナー `http-listener-1` は、インストール時に自動的に作成されます。この HTTP リスナーは、IP アドレス 0.0.0.0 を使用します。また、ユーザーがインストール時に管理サーバーのポートとして指定したポートを使用します。デフォルトのポート番号は 4848 です。デフォルトの HTTP リスナーを削除することはできません。

ドメイン内の管理サーバーインスタンスでは、HTTP リスナーだけが `http-listener-1` という ID を持ちます。つまり、管理サーバーのインスタンスを作成すると、HTTP リスナーだけがある特定時点で HTTP または HTTPS プロトコルとして機能できることになります。これは、管理サーバーに対する HTTP 接続と HTTPS 接続を同時に持てないことを意味します。Solaris 9 バンドル版の設定方法の詳細については、31 ページの「Solaris バンドル版の設定」を参照してください。

管理サーバーの SSL/TLS セキュリティ設定の有効化と設定は、HTTP リスナーで行います。

さらに、HTTP リスナー内のアクセプタスレッド (受け入れスレッド) の数を指定します。受け入れスレッドは、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入れられ、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、常に十分な数の受け入れスレッドを確保しておくのが理想的ですが、システムに負荷がかかり過ぎない数に抑える必要があります。デフォルトの受け入れスレッド数は 1 です。システム上の CPU ごとに受け入れスレッドを 1 つずつ確保するのが適切です。パフォーマンスに問題があるときは、この値を調整できます。パフォーマンスの詳細については、『Sun ONE Application Server パフォーマンスおよびチューニングガイド』を参照してください。

管理サーバーの HTTP リスナーの設定を編集するには、次の手順に従います。

1. 左側のペインの「管理サーバー」をクリックします。
2. 「HTTP リスナー」タブをクリックします。
3. 設定内容に必要な変更を加え、「了解」をクリックします。

HTTP リスナーの詳細については、オンラインヘルプを参照してください。

SNMP、ログ、セキュリティのプリファレンスの設定

SNMP の設定については、第 6 章「Sun ONE Application Server の監視」を参照してください。ログについては、第 5 章「ログの使用」を参照してください。セキュリティ設定については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

第 3 章

管理ドメインの設定

この章では、Sun ONE Application Server による管理ドメインの設定方法と管理方法を説明します。

この章では次のトピックについて説明します。

- 管理ドメインについて
- ドメインの設定
- ドメインレジストリの再作成

管理ドメインについて

管理ドメインは、異なる管理者がマシン上のアプリケーションサーバーインスタンスをグループ (ドメイン) 化して管理するための、基本的なセキュリティ構造を提供します。このようにアプリケーションサーバーインスタンスを分割すると、それぞれ異なる管理者が存在する別々の組織によって 1 台のマシンを共有することが可能になります。

Sun ONE Application Server では、各アプリケーションサーバーインスタンスは、いずれかのドメインのメンバーとなります。ドメインが複数あることが必須ではありませんが、複数のドメインを設定することによって便利な機能を利用できます。

管理セキュリティは、基になるオペレーティングシステムのセキュリティメカニズムを使った (つまり、ファイルへのアクセス権を介した) ローカルコマンドによって確立されます。リモートコマンドの場合は、特定の管理サーバーと通信するために、ユーザー名とパスワードをペアで使用することでセキュリティが確保されます。管理ドメインでは、これら以外のセキュリティ構造は利用されません。

この節では、次の項目について説明します。

- 管理ドメインの実装
- ディレクトリ構造

- プロセスとポートの構造

管理ドメインの実装

ドメインは、ファイル、オペレーティングシステムのプロセス、およびポートを使って実装されます。各ドメインには、一意の名前が付けられます。

ディレクトリ構造

インストールごとに、すべてのドメインに共有されるファイル (設定ファイル、実行可能ファイルなど) があります。ここで重要なのは、各ドメインに固有のファイルです。

ドメインすべてに固有のファイルは、ドメインディレクトリと呼ばれる共通のルートディレクトリを共有します。ルートディレクトリの名前にはドメイン名が付けられます。ドメインディレクトリの下には、インスタンスごとに 1 つのディレクトリが作成され、それぞれ各インスタンスの名前が付けれられます。さらに、各インスタンスディレクトリには、インスタンスに固有のファイルが格納されます。

ドメインディレクトリは、ファイルシステム内のどこにでも構築できます (ただし、セキュリティ権限などオペレーティングシステムレベルの制約に従う)。ユーザーが場所を指定しない限り、ドメインディレクトリはデフォルトディレクトリの下 (ドメインディレクトリと呼ばれる) に構築されます。ユーザーは別の位置にドメインディレクトリを作成することも可能です。

プロセスとポートの構造

ドメインは、実行中に、オペレーティングシステムプロセスとポートを消費します。特に、ドメインの管理サーバーを含むドメイン内で実行中の各インスタンスには、プロセスとポートが 1 つずつ存在します。

ドメインの設定

ドメイン専用のコマンドを使うと、ドメインを作成、削除、一覧表示、開始、および停止できます。

ドメインの作成、削除、および開始はローカルだけで実行できますが、ドメインの一覧表示と停止はローカルでもリモートでも実行できます。

削除、開始、および停止の各コマンドにはすべてドメイン名を指定します。ドメインが 1 つだけ存在する場合、ドメイン名の指定は省略できます。複数のドメインが設定されている環境でドメイン名を指定せずにコマンドを実行すると、エラーが発生します。

この節には次の項目があります。

- ドメインの作成
- ドメインの削除
- ドメインの一覧表示
- ドメインの開始
- ドメインの停止

ドメインの作成

ドメインの作成には、`create-domain` コマンドを使用します。このコマンドはローカルだけで使用できます。

構文

`asadmin create-domain [--path` *domain_path*`] [--sysuser` *sys_user*`] [--passwordfile` *file_name*`] --adminport` *port_number* `--adminuser` *admin_user* `--adminpassword` *password domain_name*

例 デフォルトの位置にドメインを作成する

```
$ asadmin create-domain --adminport 123 --adminuser MyAdmin
--adminpassword MyPassword MyDomain
```

この例では、デフォルトの位置 (ドメインディレクトリ) に MyDomain というドメインが作成されます。管理サーバーはポート 123 で待機し、管理ユーザー名は MyAdmin で、パスワードは MyPassword となります。ドメインディレクトリおよびその配下のファイルは、このコマンドを実行したオペレーティングシステムのユーザーに所有されます。また、オペレーティングシステムのプロセスは、このコマンドを実行したユーザーとして実行されます。

既に MyDomain というドメインが存在する場合は、エラーメッセージが返されます。

コマンド行でパスワードを使用するのではなく (セキュリティ上の問題になる)、パスワードをファイルに書き込んでおき、--passwordfile オプションを使って受け渡すことができます。

例 デフォルトの位置以外にドメインを作成する

```
$ asadmin create-domain --path $HOME --adminport 123 --adminuser
MyAdmin --adminpassword MyPassword MyDomain
```

この例は 1 番目の例と似ていますが、デフォルトのドメインディレクトリにではなく、ユーザーの $HOME ディレクトリに作成される点が異なります。

例 ユーザーを変更してドメインを作成する (UNIX のみ)

```
# asadmin  create-domain --user AnotherUser --adminport 123
--adminuser MyAdmin --adminpassword MyPassword MyDomain
```

この例は 1 番目の例に似ていますが、ドメインおよびその中に含まれるファイルが AnotherUser というユーザーに所有される点が異なります。オペレーティングシステムのプロセスも、このユーザーとして実行されます。

--sysuser オプションを使用すると、他のユーザーが管理するドメインを構築できます。このオプションを指定するには、create-domain コマンドを実行するユーザーが root である必要があります。

UNIX プラットフォームでのユーザーのアクセス権

root 以外のユーザーが管理ドメインを作成または削除するには、ドメインの設定ファイルに対する書き込み権を持つ UNIX グループにそのユーザーの ID を追加する必要があります。

1. インストール全体に有効なドメイン設定ファイルに適用される UNIX グループを作成します。たとえば、asadmin という名前の UNIX グループです。
2. インストール全体に有効なドメイン設定ファイルを /etc/appserver に格納して、新しく作成した UNIX グループが所有するようにします。

   ファイル名は、domains.bin と domains.lck です。たとえば、これらのファイルは変更後、次のようになります。

   ```
   -rw-r--r-- 1 root asadmin 0 Sep 18 14:34 domains.bin
   ```

   ```
   -rw-r--r-- 1 root asadmin 0 Sep 18 14:34 domains.lck
   ```

3. 新しく作成した UNIX グループがこれらのファイルに対して書き込みアクセスできるように設定にします。この例では、アクセス権は次のようになります。

   ```
   -rw-rw-r-- 1 root asadmin 0 Sep 18 14:34 domains.bin
   ```

   ```
   -rw-rw-r-- 1 root asadmin 0 Sep 18 14:34 domains.lck
   ```

4. ユーザー ID を UNIX グループに追加します。

上記とは反対に、インストール全体に適用する設定ファイルへの書き込みアクセスを root ユーザー以外に与えたくない場合は、ユーザーではなく管理ドメインを作成することもできます。新しい管理ドメインを作成するには、`--sysuser` オプションによって、ドメインのディレクトリとファイルを所有する UNIX ユーザー ID を指定し、`--path` オプションによって、管理ドメインを作成する位置を指定します。この例については、58 ページの「例 ユーザーを変更してドメインを作成する (UNIX のみ)」を参照してください。

管理ドメインをユーザー ID を基に作成すると、そのユーザーは新しいアプリケーションサーバーインスタンスを作成できるようになり、作成したアプリケーションサーバーインスタンスに対するさまざまな管理操作を行うことができます。ユーザー ID は、管理ドメイン設定ファイルへの書き込み権限を持つ UNIX グループに所属している必要はありません。UNIX グループへの登録が必要となるのは、管理ドメインを作成または削除するユーザーだけです。

ドメインの削除

ドメインの削除には、`delete-domain` コマンドを使用します。ドメインを管理できるオペレーティングシステムユーザー (または root) だけがこのコマンドを実行できます。このコマンドはローカルだけで使用できます。

構文

```
asadmin delete-domain [domain_name]
```

例 ドメインを削除する

```
$ asadmin delete-domain MyDomain
```

この例では、ローカルマシン上の MyDomain というドメインが削除されます。

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表示するには、`list-domains` コマンドを使用します。

このコマンドは、ローカルとリモートの両方で使用できます。

構文

```
asadmin list-domains [--host host] [--port port] [--password password]
[--user user]
```

例 ローカルマシン上のドメインを一覧表示する

```
$ asadmin list-domains
domain1     [/opt/ias/build/domains/domain1]
```

例 リモートオプションを使用して、ローカルマシン上のドメインを一覧表示する

```
$ asadmin list-domains --user admin --password password --host
localhost --port 4848
domain1     [/opt/ias/build/domains/domain1]
```

ドメインの開始

ドメインの開始には、start-domain コマンドを使用します。このコマンドは、ドメインの管理サーバーと、ドメイン内のすべてのインスタンスを開始します。

このコマンドは、ローカルだけで実行できます。

構文

```
asadmin start-domain [--domain domain_name]
```

例 マシン上に 1 つだけあるドメインを開始する

```
$ asadmin start-domain
インスタンス domain1:admin-server が起動しました
インスタンス domain1:server1 が起動しました
ドメイン domain1 が起動しました。
```

ドメインの停止

ドメインの停止には、stop-domain コマンドを使用します。ユーザーはドメイン内のすべてのインスタンスを停止できます。また、管理サーバー以外の全インスタンスを停止することもできます。そのようにすると、ドメインのリモート管理を継続できます。

このコマンドは、ローカルとリモートの両方で実行できます。

構文

```
asadmin stop-domain [--user admin_user] [--password admin_password]
[--host host_name] [--port port_name] [-- local=false] [--domain
domain_name] [--adminserv=true] [--passwordfile file_name] [--secure |
-s]
```

例ドメイン内の管理サーバーインスタンス以外の全インスタンスを停止する

```
$ asadmin stop-domain --user admin --password password --host
localhost --port 4848 --adminserv=false --domain domain1

DomainStoppedRemotely
```

ドメインレジストリの再作成

各ドメインの詳細情報 (名前、位置、使用されるポートなど) は、実装上、ドメインレジストリと呼ばれるファイルに記録されています。

ドメインレジストリの変更や使用はシステムの管理に使用するコマンドにカプセル化されているため、通常はドメインディレクトリを直接操作する必要はありません。ただし、ドメインレジストリはファイルであるため、スクリプトの実行が不正な場合や、誰かが不注意にレジストリを削除してしまった場合などに破損する可能性があります。破損した場合は、ファイルを再作成する必要があります。

注 ドメインレジストリには、asadmin コマンドを使って、コマンド行インタフェースからアクセスできます。

レジストリが破損した場合、次の手順に従って、レジストリを再作成してください。

1. すべてのドメイン、およびドメインが格納されているディレクトリ (デフォルトまたはデフォルト以外) の一覧を入手します。
2. 各ディレクトリの名前を変更します (たとえば、各ディレクトリ名に ".bak" のサフィックスを追加)。
3. ポートやパスワードなどのデフォルト値を使って、元の位置に各ドメインを再度作成します。
4. 新しいドメインディレクトリをすべて削除し、元のディレクトリと置き換えます。
5. 各ドメインに対して、reconfig コマンドを実行します。これで、古いドメインの値によってドメインレジストリが更新されます。

第 2 部

サーバーインスタンスの管理

第 4 章「アプリケーションサーバーインスタンスの使用」

第 5 章「ログの使用」

第 6 章「Sun ONE Application Server の監視」

第 7 章「Web サーバープラグインの設定」

第 8 章「J2EE コンテナの設定」

第 9 章「トランザクションサービスの使用」

第 10 章「ネーミングとリソースの設定」

第 11 章「JMS サービスの使用」

第 12 章「Corba/IIOP クライアント用のサーバーの設定」

第 13 章「アプリケーションの配備」

第 4 章

アプリケーションサーバーインスタンスの使用

この章では、Sun ONE Application Server のインスタンスを作成、削除、設定、起動、および停止する方法について説明します。

この章では次のトピックについて説明します。

- アプリケーションサーバーインスタンスについて
- アプリケーションサーバーインスタンスの起動と停止
- アプリケーションサーバーインスタンスのデバッグモードでの起動
- 終了タイムアウトの設定
- アプリケーションサーバーインスタンスの自動再起動 (UNIX)
- アプリケーションサーバーインスタンスの手動再起動 (UNIX)
- ウォッチドッグについて
- アプリケーションサーバーインスタンスの追加
- アプリケーションサーバーインスタンスの削除
- アプリケーションサーバーインスタンスの変更の適用
- アプリケーションサーバーインスタンスの状態の表示
- JVM 設定
- ログ設定と監視設定
- アプリケーションサーバーインスタンスの詳細設定の変更

アプリケーションサーバーインスタンスについて

アンバンドル版の Sun ONE Application Server をインストールすると、`server1` というアプリケーションサーバーインスタンスが 1 つ作成されます。`server1` インスタンスを削除して、任意の名前のインスタンスを新しく作成することもできます。

Solaris 9 のバンドル版を使う場合は、サーバーインスタンスを作成する必要があります。詳細については、31 ページの「Solaris バンドル版の設定」を参照してください。

個々のアプリケーションサーバーインスタンスの J2EE 設定、J2EE リソース、アプリケーション配備領域、サーバー構成設定は独立しています。あるインスタンスに変更を加えても、ほかのインスタンスに影響はありません。1 つの管理ドメインに複数のアプリケーションサーバーインスタンスを持たせることができます。1 つのドメインでは、すべてのサーバーインスタンスの管理サーバーは同じです。ドメインの詳細については、第 3 章「管理ドメインの設定」を参照してください。

多くのユーザーのニーズは、アプリケーションサーバーインスタンスが 1 つあれば満たされます。しかし、ユーザーの環境によっては、複数のアプリケーションサーバーインスタンスを作成したい場合もあります。たとえば、開発環境では、複数のアプリケーションサーバーインスタンスを使って、何種類かの Sun ONE Application Server 設定をテストしたり、アプリケーションの配備の方法を比較検討することがあります。この場合、追加および削除の簡単なアプリケーションサーバーインスタンスの特性を活かして、「サンドボックス」という一時領域を作成し、さまざまなテストを実施できます。

さらに、アプリケーションサーバーインスタンスごとに仮想サーバーを作成することもできます。インストール済みの単一の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、その他の管理機能を提供できます。このとき、ユーザーは、あたかも独自の Web サーバーであるかのように仮想サーバーを使用できます。ハードウェアや基本的なサーバーメンテナンスは必要ありません。ただし、ある仮想サーバーを複数のアプリケーションサーバーインスタンスで使用することはできません。仮想サーバーの詳細については、第 15 章「仮想サーバーの使用」を参照してください。

実際の配備作業では、目的に合わせて、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使用できます。仮想サーバーがユーザーのニーズに合わない場合は、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できます。

アプリケーションサーバーインスタンスの起動と停止

Sun ONE Application Server は、自動的には起動しません。しかし、いったんユーザーの手で起動すると、インスタンスは停止するまで継続して実行されます。アプリケーションサーバーインスタンスを停止すると、新しい接続を受け付けなくなり、処理中のすべての接続が完了するまで待機します。ユーザーのマシンがクラッシュしたり、オフラインになったりした場合、サーバーは停止します。

この場合、それまで処理中だったすべての要求は失われます。アプリケーションサーバーインスタンスは、次のいずれかの方法で起動および停止できます。

- 管理インタフェースの「起動」ボタンと「停止」ボタンの使用
- start-instance コマンドと stop-instance コマンドの使用
- Windows サービスの使用 (Windows の場合)
- startserv スクリプトと stopserv スクリプトの使用

注 サーバーにセキュリティモジュールをインストールしている場合は、起動または停止の前に適切なパスワードの入力を求められます。

サーバー証明書をインストールしている環境では、管理者は、Sun ONE Application Server の起動時にキーデータベースのパスワードを入力する必要があります。Sun ONE Application Server をユーザーの介入なしで再起動できるようにするには、`password.conf` ファイルにパスワードを保存する必要があります。このファイルとキーデータベースが危険にさらされないようにするためには、システムが充分にセキュリティ保護されている場合だけに行なってください。`password.conf` の詳しい作成方法および使用方法については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

注 Sun ONE Application Server を UNIX 環境にインストールした場合、オペレーティングシステムによってデフォルトで制限されているメモリーやファイル記述子にアクセスしなければならないことがあります。サーバーを起動できないときは、`ulimit` コマンドを使って、オペレーティングシステムによるリソースの制限がないかどうかを確認してください。詳細については、ご使用のオペレーティングシステムの `ulimit` のマニュアルページを参照してください。

管理インタフェースの「起動」ボタンと「停止」ボタンの使用

管理インタフェースを使ってサーバーを起動または停止するには、次の手順に従います。

1. 左側のペインの「アプリケーションサーバーインスタンス」の下に表示されているインスタンス名をクリックします。
2. 右側のペインの「起動」または「停止」をクリックするか、「一般」タブで「起動」または「停止」をクリックします。
3. アプリケーションサーバーインスタンスが正常に起動または停止すると、メッセージが表示されます。

start-instance コマンドと stop-instance コマンドの使用

コマンド行インタフェースユーティリティ asadmin を使うと、コマンドプロンプトまたはスクリプトからアプリケーションサーバーインスタンスを起動および停止できます。start-instance コマンドおよび stop-instance コマンドを使います。

これらのコマンドの構文は次のとおりです。

start-instance [--user *admin_user*] [--password *admin_password*] [--host *admin_host*] [--port *admin_port*] [--local=*true/false*] [--domain *domain_name*] [--debug=*true/false*] [--passwordfile *file_name*] [--secure | -s] *instance_name*

stop-instance [--user *admin_user*] [--password *admin_password*] [--host *admin_host*] [--port *admin_port*] [--local=*true/false*] [--domain *domain_name*] [--passwordfile *file_name*] [--secure | -s] *instance_name*

コマンドの実行時に local オプションを指定すると、管理サーバーを介することなくサーバーを起動または停止できます。local オプションを指定するときは、host、port、user、および password (または passwordfile) オプションを指定する必要はありません。

これらのコマンドの構文については、asadmin のヘルプを参照してください。asadmin の使用方法については、付録 A「コマンド行インタフェースの使用」を参照してください。

Windows サービスの使用 (Windows の場合)

Windows の「サービス」コントロールパネルからサーバーを起動できます。

次の手順に従います。

1. コントロールパネルの「管理ツール」をクリックします。
2. 「サービス」をクリックします。
3. サービスのリストをスクロールし、使用しているサーバーのサービスをダブルクリックします。

   サービス名は「Sun Application Server (*domain_name:instance_name*)」の形式になります。たとえば、Sun Application Server (domain1:server1) のようになります。
4. 「起動」または「停止」をクリックします。
5. 「了解」をクリックします。

startserv スクリプトと stopserv スクリプトの使用

`startserv` スクリプトおよび `stopserv` スクリプトを使うには、コマンド行プロンプトから次のディレクトリに移動します。

instance_dir/bin

install_dir はサーバーのインストール先ディレクトリ、*domain_dir* はドメインディレクトリ、*instance_dir* は起動するインスタンスの名前です。

UNIX 環境では、次のように入力します。

```
./startserv
```

サーバーが 1024 より小さい番号のポートで実行されている場合は root として、それ以外の場合は root またはそのサーバーのユーザーアカウントでログインします。

末尾にオプションパラメータ `-i` を指定することもできます。`-i` オプションを指定すると、サーバーが `inittab` モードで実行され、サーバーの強制終了時やクラッシュ時に `inittab` がサーバーを再起動します。このオプションを指定すると、サーバーがバックグラウンド処理に切り替わることはありません。

注	`startserv` コマンドは、サーバーの実行中は失敗します。`startserv` コマンドを実行する前に、サーバーを停止してください。サーバーの起動に失敗する場合は、再起動する前にプロセスを強制終了してください。

Windows 環境では、次のように入力します。

```
startserv
```

手動でサーバーを停止する場合は、コマンド行プロンプトを使って次のディレクトリに移動します。

instance_dir/bin

install_dir はサーバーのインストール先ディレクトリ、*instance_dir* は起動するインスタンスの名前です。

UNIX 環境では、次のように入力します。

```
./stopserv
```

サーバー起動時に、/etc/inittab ファイルを使用した場合は、サーバーを停止する前に /etc/inittab ファイルからサーバーを起動するための行を削除し、`kill -1 1` を実行します。それ以外の場合は、停止したサーバーは自動的に再起動します。

Windows 環境では、次のように入力します。

```
stopserv
```

アプリケーションサーバーインスタンスのデバッグモードでの起動

J2EE アプリケーションをデバッグする場合は、アプリケーションサーバーインスタンスをデバッグモードで実行できます。

デバッグモードでサーバーを起動するには、次の手順に従います。

1. 管理インタフェースにアクセスし、デバッグモードで起動するアプリケーションサーバーインスタンス名をクリックします。
2. 「一般」タブをクリックします。
3. 「デバッグモードで起動または再起動」の横のチェックボックスをクリックします。
4. アプリケーションサーバーインスタンスを再起動します。

デバッグモードにすると、JVM 設定が変更されます。「デバッグを有効」がオンになり、「デバッグオプション」が変更されます。JVM デバッグオプションの詳細については、『Java Platform Debugger Options』(http://java.sun.com/products/jpda/doc/conninv.html) を参照してください。

コマンド行インタフェースを使ってアプリケーションサーバーインスタンスをデバッグモードで起動する場合は、debug オプションを true に設定し asadmin ユーティリティの start-instance コマンドを実行します。コマンド構文の詳細については、コマンド行インタフェースのオンラインヘルプを参照してください。

終了タイムアウトの設定

アプリケーションサーバーインスタンスを停止すると、新しい接続の受け入れが停止します。その後、アプリケーションサーバーインスタンスは、未処理の接続がすべて完了するまで待機します。タイムアウトになるまでサーバーが待機する時間は、init.conf ファイルで設定できます。このファイルは *instance_dir*/config/ にあります。デフォルトでは 30 秒に設定されています。この値を変更するには、init.conf に次の行を追加します。

TerminateTimeout *seconds*

seconds は、タイムアウトになるまでサーバーが待機する秒数を表します。

この値を変更する利点は、接続の処理が完了するまでサーバーが待機する時間が、長くなることです。ただし、サーバーは応答していないクライアントに接続されていることがあるため、終了タイムアウト値を大きくすると、サーバーのシャットダウンにかかる時間が長くなる可能性があります。

アプリケーションサーバーインスタンスの自動再起動 (UNIX)

次のいずれかの方法で、アプリケーションサーバーインスタンスを再起動できます。

- /etc/inittab ファイルからの自動再起動

  使用中の UNIX/Linux が System V から派生したものではない場合、/etc/inittab ファイルは使用できません。

- マシンの再起動時に、/etc/rc2.d 内のデーモンによる自動再起動
- 手動での再起動。67 ページの「アプリケーションサーバーインスタンスの起動と停止」および 77 ページの「アプリケーションサーバーインスタンスの削除」を参照

この節には次の項目があります。

- 自動再起動について
- /etc/inittab による再起動 (UNIX)

- システムの RC スクリプトによる自動再起動 (UNIX)

自動再起動について

/etc/rc.local ファイルや /etc/inittab ファイルは、インストールスクリプトでは編集できません。テキストエディタで編集してください。編集方法がわからない場合は、システム管理者に問い合わせるか、使用システムのマニュアルを参照してください。

通常、SSL が有効なサーバーは起動の前にパスワードを要求するため、これらのファイルでは起動できません。パスワードをプレーンテキストでファイルに保存すれば、SSL が有効なサーバーを自動的に起動できますが、この方法はお勧めしません。

警告 SSL が有効なサーバーの startserv スクリプトにプレーンテキストでパスワードを保存すると、セキュリティ上きわめて危険です。ファイルにアクセスできるユーザーなら誰でも、SSL が有効なサーバーのパスワードにアクセスできます。SSL が有効なサーバーのパスワードをプレーンテキストで保存する前に、セキュリティ上の危険性を考慮しておく必要があります。

通常、サーバーの startserv スクリプト、キーペアファイル、キーパスワードの所有者は root ユーザーです。root 以外のユーザーがサーバーをインストールした場合は、そのユーザーが所有者になります。これらのスクリプト、ファイル、パスワードへの読み取りおよび書き込み権を持つのは、その所有者だけです。

/etc/inittab による再起動 (UNIX)

inittab を使ってサーバーを再起動するには、/etc/inittab ファイルに次のテキストを 1 行で追加します。

```
http:2:respawn:install_dir/path_to_domain_dir/instance_dir/bin/startserv -start
-i
```

install_dir はサーバーのインストール先ディレクトリ、*path_to_domain_dir* はドメインディレクトリのパス、*instance_dir* はサーバーのディレクトリです。

-i オプションを指定すると、サーバーがバックグラウンド処理に切り替わることはありません。

この行は、サーバーを停止する前に削除する必要があります。削除しないと、サーバーが自動的に再起動してしまいます。

システムの RC スクリプトによる自動再起動 (UNIX)

/etc/rc.local、または使用システムでこれに相当するスクリプトを使用する場合は、/etc/rc.local ファイルに次の行を追加します。

install_dir/*path_to_domain_dir/instance_dir*/bin/startserv

install_dir はサーバーのインストール先ディレクトリ、*path_to_domain_dir* はドメインディレクトリのパス、*instance_dir* はアプリケーションサーバーインスタンスの名前です。

アプリケーションサーバーインスタンスの手動再起動 (UNIX)

UNIX 環境では、インスタンスを手動で再起動するオプションも利用できます。サーバーインスタンスの停止後の起動とは異なり、再起動時はウォッチドッグプログラムは停止されません。ウォッチドッグの詳細については、75 ページの「ウォッチドッグについて」を参照してください。

注 設定ファイルを編集して手動で変更を加えたときは、管理インタフェースの「変更を適用」ボタンをクリックする、または asadmin reconfig コマンドの keepmanualchanges オプションを true に設定して、サーバーを再起動する前に変更を適用する必要があります。変更の適用については、78 ページの「アプリケーションサーバーインスタンスの変更の適用」を参照してください。

次の項目では、サーバーインスタンスを再起動する 3 種類の方法について説明します。

- 「再起動」ボタンによるサーバーインスタンスの再起動 (UNIX)
- restart-instance コマンドによるサーバーインスタンスの再起動 (UNIX)
- restartserv スクリプトによるサーバーインスタンスの再起動 (UNIX)

「再起動」ボタンによるサーバーインスタンスの再起動 (UNIX)

管理インタフェースを使ってサーバーインスタンスを再起動するには、次の手順を実行します。

1. 左側のペインの「アプリケーションサーバーインスタンス」の下で、再起動するインスタンスの名前をクリックします。
2. 右側のペインの「再起動」をクリックします。
3. アプリケーションサーバーインスタンスが正常に再起動すると、メッセージが表示されます。

restart-instance コマンドによるサーバーインスタンスの再起動 (UNIX)

コマンド行インタフェースユーティリティ `asadmin` を使うことで、コマンド行またはスクリプトからアプリケーションサーバーインスタンスを起動および停止できます。この場合は `restart-instance` コマンドを使います。このコマンドの構文は次のとおりです。

`restart-instance` [`--user` *admin_user*] [`--password` *admin_password*] [`--host` *admin_host*] [`--port` *admin_port*] [`--local=`*true/false*] [`--domain` *domain_name*] [`--passwordfile` *file_name*] [`--secure` | `-s`] *instance_name*

コマンドの実行時に `local` オプションを指定すると、管理サーバーを介することなくサーバーを再起動できます。

これらのコマンドの構文については、`asadmin` のヘルプを参照してください。`asadmin` の使用方法については、付録 A「コマンド行インタフェースの使用」を参照してください。

restartserv スクリプトによるサーバーインスタンスの再起動 (UNIX)

`restartserv` スクリプトを使うには、コマンド行プロンプトから次のディレクトリに移動します。

instance_dir/bin

install_dir はサーバーのインストール先ディレクトリ、*domain_dir* はドメインディレクトリ、*instance_dir* は起動するインスタンスの名前です。

次のように入力します。

```
./restartserv
```

サーバーが 1024 より小さい番号のポートで実行されている場合は root として、それ以外の場合は root またはそのサーバーのユーザーアカウントでログインします。

ウォッチドッグについて

ウォッチドッグ (UNIX では `appserv-wdog`、Windows では `appservd-wdog.exe`) は、Sun ONE Application Server に付属するプログラムです。このプログラムは次のタスクを実行します。

- サーバーを起動する
- サーバーを停止する
- SSL/TLS が有効な場合に、サーバーの起動時に信頼データベースのパスワードを管理者に要求する
- サーバーが停止した場合に再起動する

ウォッチドッグはバックグラウンドで実行され、ユーザーによる操作を必要としません。このため、設定や設定の変更は必要ありません。管理サーバーを含め、アプリケーションサーバーインスタンスごとに 1 つのウォッチドッグが実行されます。

UNIX 環境では、各ウォッチドッグがアプリケーションサーバー (`appservd`) の基本プロセス用にプロセスを生成し、そのプロセスが要求を受け付ける `appservd` プロセスを生成します。ウォッチドッグはサーバーを起動するため、ウォッチドッグのプロセス ID が *instance_dir*/logs の下の `pid` ログファイルに記録されます。

注

`UNIX` 環境の `appservd` プロセス : Windows 環境ではアプリケーションサーバーのインスタンスごとに 1 つの `appservd` プロセスが起動しますが、UNIX 環境ではインスタンスごとに 2 つの `appservd` プロセスが起動します。

UNIX 環境で起動する 2 つの `appservd` プロセスのうち、1 つは「基本」プロセスと呼ばれ、もう 1 つは「ワーカー」プロセスと呼ばれます。ワーカープロセスはアプリケーションからの要求を実際に処理し、基本プロセスは全体を制御するコントローラとして機能します。アプリケーションサーバーの将来のリリースでは、アプリケーションサーバーインスタンスごとにワーカープロセス数を定義するオプションを用意する予定です。今回のリリースでは、アプリケーションサーバーインスタンスに定義できるワーカープロセスは 1 つだけです。

アプリケーションサーバーインスタンスの追加

管理インタフェースを使ってアプリケーションサーバーインスタンスを追加するには、次の手順に従います。

1. 管理インタフェースにアクセスし、左側のペインの「アプリケーションサーバーインスタンス」をクリックします。
2. 「一般」タブをクリックします。
3. 「アプリケーションサーバーインスタンス」ページの「新規」をクリックします。
4. 「新規のインスタンス作成」ページにインスタンス名とポート番号を指定します。

   この管理サーバーおよびドメインに固有のインスタンス名を指定してください。また、マシン上のその他のプロセスによって使用されていないポート番号を指定してください。

   UNIX 環境では、インスタンスを実行する UNIX ユーザーも指定できます。
5. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

コマンド行インタフェースを使ってアプリケーションサーバーインスタンスを追加する場合は、`asadmin` ユーティリティの `create-instance` コマンドを使用します。構文は次のとおりです。

```
asadmin create-instance [--user admin_user] [--password admin_password]
[--host host] [--port port] [--sysuser sys_user] [--domain domain_name]
[--local=true/false] [--passwordfile file_name] [--secure | -s]
--instanceport instance_port instance_name
```

コマンドの実行時に `local` オプションを指定すると、管理サーバーを介することなくサーバーを再起動できます。`sysuser` オプションは、UNIX 環境だけで利用できます。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

アプリケーションサーバーインスタンスの削除

管理ドメインからアプリケーションサーバーインスタンスを削除できます。削除する前に、そのアプリケーションサーバーインスタンスが不要であることを確認してください。いったん削除したインスタンスを元に戻すことはできません。

管理インタフェースを使ってマシンからアプリケーションサーバーインスタンスを削除するには、次の手順に従います。

1. 管理インタフェースにアクセスし、削除するアプリケーションサーバーインスタンスの名前をクリックします。
2. 「一般」タブをクリックします。
3. 「削除」をクリックします。

詳細については、オンラインヘルプを参照してください。

コマンド行インタフェースを使ってアプリケーションサーバーインスタンスを削除する場合は、`asadmin` ユーティリティの `delete-instance` コマンドを使用します。構文は次のとおりです。

`asadmin delete-instance` [`--user` *admin_user*] [`--password` *admin_password*] [`--host` *admin_host*] [`--port` *admin_port*] [`--domain` *domain_name*] [`--local=`*true/false*] [`--passwordfile` *file_name*] [`--secure` | `-s`] *instance_name*

コマンドの実行時に `local` オプションを指定すると、管理サーバーを介することなくサーバーを削除できます。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

アプリケーションサーバーインスタンスの変更の適用

管理インタフェースまたはコマンド行インタフェースを使って設定情報を変更しても、*server_instance*/config/backup に保存されている特別なファイルに変更内容が記録されるだけで、自動的には適用されません。管理インタフェースおよびコマンド行インタフェースは、上記ディレクトリに保存されているファイルに記録された設定値を表示します。変更内容は、適用するまで反映されません。変更の適用は「サーバーの再設定」とも呼ばれます。変更を適用すると、最後に変更を適用したときから現在までの変更内容がすべて有効になります。インスタンスを再起動しても、変更内容は自動的に適用されません。

変更の適用が必要なサーバーインスタンスの設定に変更を加えたときは、左側のペインのツリーに表示されるアプリケーションサーバーインスタンスの横、サーバーインスタンスへのアクセス時に表示されるバナー、およびサーバーインスタンスのメインページに黄色のアイコンが表示されます。

警告アイコン

管理インタフェースを使ってアプリケーションサーバーインスタンスに変更を適用するには、次の手順に従います。

1. 管理インタフェースにアクセスして、再設定するアプリケーションサーバーインスタンス名をクリックします。
2. 「一般」タブをクリックします。
3. 「変更の適用」をクリックします。

   変更が適用されると、画面にメッセージが表示されます。

コマンド行インタフェースを使ってアプリケーションサーバーインスタンスを再設定する場合は、`asadmin` ユーティリティの `reconfig` コマンドを使用します。構文は次のとおりです。

`asadmin reconfig --user` *admin_user* `[--password` *admin_password*`] [--host` *admin_host*`] [--port` *admin_port*`] [--passwordfile` *file_name*`] [--secure | -s] [--discardmanualchanges=`*true/false* `| --keepmanualchanges=`*true/false*`]` *instance_name*

設定ファイルを編集して手作業で変更を加えた際、再設定時に編集内容を維持するときは、keepmanualchanges=true オプションを使う必要があります (デフォルトは false)。discardmanualchanges=true に設定すると、手作業で加えた変更はすべて破棄されます。discardmanulchanges=false (デフォルト) と keepmanualchanges=true は同じ意味ではありません。keepmanualchanges=false に設定した場合は、discardmanualchanges オプションを指定しない場合と同じ意味となります。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。asadmin の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

変更を適用したあと、サーバーを再起動しないと変更内容が有効にならないプロパティもあります。設定ファイル init.conf および obj.conf のすべてのプロパティ、server.xml ファイルの一部のプロパティがこれに該当します。これらのファイルの詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

変更を適用する上でサーバーの再起動が必要となるときは、左側のペインのツリーに表示されるアプリケーションサーバーインスタンスの横、サーバーインスタンスへのアクセス時に表示されるバナー、およびサーバーインスタンスのメインページに黄色の警告アイコンが表示されます。ページのバナーとメインページには、サーバーの再起動が必要なことを示すメッセージが表示されます。サーバーインスタンスを再起動すると、黄色の警告アイコンは表示されなくなります。

次の server.xml の設定の変更は、再起動を必要としません。

- J2EE アプリケーション (EAR ファイル)、EJB モジュール (JAR ファイル)、Web モジュール (WAR ファイル)、およびコネクタ (RAR ファイル) の配備、配備取消し、再配備。これらの設定には、変更の適用は必要ない
- J2EE アプリケーション (EAR ファイル)、EJB モジュール (JAR ファイル)、Web モジュール (WAR ファイル)、およびコネクタ (RAR ファイル) の有効化と無効化
- リソースの作成、更新、削除
- EJB コンテナまたは MDB コンテナの監視設定 (true または false)
- HTTP および Web コンテナ機能の変更 (server.xml 内の http-service、web-container、およびこれらのサブ要素の変更)

アプリケーションサーバーインスタンスの状態の表示

管理インタフェースでは、アプリケーションサーバーインスタンスの基本的な設定のほか、サーバーが動作しているかどうかを確認できます。

アプリケーションサーバーインスタンスの状態を確認するには、次の手順に従います。

1. 左側のペインで、アプリケーションサーバーインスタンス名をクリックします。
2. 右側のペインの「一般」タブをクリックします。

   サーバーのホスト名、ポート番号、インストールディレクトリ、Sun ONE Application Server ソフトウェアのバージョンとともに、サーバーが実行中であるかどうかが表示されます。

コマンド行インタフェースを使ってアプリケーションサーバーインスタンスの状態を確認する場合は、`asadmin` ユーティリティの `show-instance-status` コマンドを使用します。状態は、起動中、実行中、停止中、停止のいずれかです。このコマンドの構文は次のとおりです。

`asadmin show-instance-status --user` *admin_user* `[--password` *admin_password*`] [--host` *admin_host*`] [--port` *admin_port*`] [--passwordfile` *file_name*`] [--secure | -s]` *instance_name*

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

JVM 設定

アプリケーションサーバーインスタンスの JVM (Java Virtual Machine) を設定できます。この設定には、Java ホームの場所、コンパイラオプション、デバッグオプション、プロファイラ情報が含まれます。設定を変更する理由の 1 つに、パフォーマンスの向上があります。パフォーマンスの詳細については、『Sun ONE Application Server パフォーマンスおよびチューニングガイド』を参照してください。

この節では、次の項目について説明します。

- 一般設定
- パス設定
- JVM オプションの設定
- JVM プロファイラの設定
- コマンド行インタフェースによる JVM の設定

一般設定

管理インタフェースを使って JVM の一般設定を行うには、次の手順を実行します。

1. 左側のペインで、アプリケーションサーバーインスタンス名をクリックします。
2. 右側のペインの「JVM」タブをクリックします。
3. 「一般」をクリックします。
4. 「Java ホーム」を設定します。

   Java ホームは、JDK (Java Developer's Kit) がインストールされているディレクトリを示すパスです。Sun ONE Application Server は、Sun JDK 1.4.0_02 以降をサポートしています。
5. デバッグを有効化するかどうかを選択し、デバッグオプションを設定します。

   デバッグオプションは、
   http://java.sun.com/products/jpda/doc/conninv.html#Invocation で参照できます。
6. 「rmic オプション」を選択します。

   「rmic オプション」フィールドには、アプリケーションの配備時に RMI コンパイラに渡される rmic オプションが表示されます。`-keepgenerated` オプションを指定すると、スタブとタイ用に生成されるソースが保存されます。rmic コマンドの詳細は、『Sun ONE Application Server Enterprise Java Beans 開発者ガイド』を参照してください。
7. 「保存」をクリックします。

パス設定

管理インタフェースを使って JVM のパス設定を行うには、次の手順を実行します。

1. 左側のペインで、アプリケーションサーバーインスタンス名をクリックします。
2. 右側のペインの「JVM」タブをクリックします。
3. 「パス設定」をクリックします。
4. システムのクラスパスのサフィックスを選択します。

5. 環境クラスパスを無視するかどうかを選択します。

   クラスパスを無視しない場合、CLASSPATH 環境変数が読み込まれ、Sun ONE Application Server のクラスパスに適用されます。CLASSPATH 環境変数は、クラスパスサフィックスの後に適用され、末尾に追加されます。

   開発環境では、クラスパスを使用する必要があります。本稼働環境では、環境変数による影響を避けるために、このクラスパスを無視する必要があります。

6. ネイティブライブラリパスのプレフィックスとサフィックスを設定します。

   ネイティブライブラリパスは、Application Server インストールディレクトリに基づくネイティブ共有ライブラリの相対パス、標準の JRE ネイティブライブラリパス、シェル環境設定 (UNIX では LD_LIBRARY_PATH)、およびプロファイラ要素に指定したパスの連結によって自動的に作成されます。これは組み合わせによって作成されるため、サーバーの設定には明示的に表示されません。

7. 「保存」をクリックします。

JVM オプションの設定

管理インタフェースを使って JVM コマンド行オプションを設定するには、次の手順を実行します。

1. 左側のペインで、アプリケーションサーバーインスタンス名をクリックします。
2. 右側のペインの「JVM」タブをクリックします。
3. 「JVM」オプションをクリックします。
4. JVM オプションを追加するときは、画面上部のテキストフィールドに入力し、「追加」をクリックします。
5. JVM オプションを削除するときは、削除するオプションのとなりのチェックボックスをクリックし、「削除」をクリックします。
6. JVM オプションを編集するときは、「JVM オプション」フィールドのテキストを編集し、「保存」をクリックします。

JVM オプションの詳細は、http://java.sun.com/docs/hotspot/VMOptions.html で参照できます。

JVM プロファイラの設定

管理インタフェースを使って JVM プロファイラを設定するには、次の手順を実行します。

1. 左側のペインで、アプリケーションサーバーインスタンス名をクリックします。
2. 右側のペインの「JVM」タブをクリックします。
3. 「プロファイラ」をクリックします。
4. プロファイラの名前、クラスパスとネイティブライブラリパス、有効にするかどうかを指定します。
5. プロファイラの JVM オプションを追加するときは、画面上部のテキストフィールドに入力し、「追加」をクリックします。
6. プロファイラの JVM オプションを削除するときは、削除するオプションのとなりのチェックボックスをクリックし、「削除」をクリックします。
7. プロファイラの JVM オプションを編集するときは、「JVM オプション」フィールドのテキストを編集し、「保存」をクリックします。

プロファイラの詳細は、『Sun ONE Application Server 開発者ガイド』を参照してください。

コマンド行インタフェースによる JVM の設定

コマンド行インタフェースの `asadmin` ユーティリティを使って JVM を設定するには、次のコマンドを使います。

インスタンスのすべての属性を確認するには、次のコマンドを使います。

```
asadmin> get server_instance.java-config.*
```

server1 の `classpathprefix` という属性を確認するには、次のコマンドを使います。

```
asadmin> get server1.java-config.classpathprefix
```

server1 の `classpathprefix` という属性を設定するには、次のコマンドを使います。

```
asadmin> set server1.java-config.classpathprefix=com.sun
```

上記すべての例は、環境変数にユーザー、パスワード、ホスト、ポートが設定されていることを前提としています。すべての属性のリストは、付録 A「コマンド行インタフェースの使用」を参照してください。

コマンド行インタフェースの `asadmin` ユーティリティを使って JVM オプションを設定するには、次のコマンドを使います。

```
asadmin> create-jvm-options --user admin_user [--password admin_password]
[--host host] [--port port] [--secure | -s] [--instance instance_name]
[--profiler=true/false]
(jvm_option_name=jvm_option_value) [:jvm_option_name=jvm_option_name] *
```

```
asadmin> delete-jvm-options --user admin_user [--password admin_password]
[--host host] [--port port] [--secure | -s] [--instance instance_name]
[--profiler=true/false]
(jvm_option_name=jvm_option_value) [:jvm_option_name=jvm_option_name] *
```

注 : コロンで区切ることで、複数の JVM オプションを入力できます。プロファイラに適用するオプションでは、`--profiler` を `true` に設定します。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

ログ設定と監視設定

「ログ」タブと「監視」タブで行うログと監視の設定については、それぞれ別の章で説明します。ログについては、第 5 章「ログの使用」を参照してください。監視および SNMP の設定については、第 6 章「Sun ONE Application Server の監視」を参照してください。

アプリケーションサーバーインスタンスの詳細設定の変更

アプリケーションサーバーインスタンスの詳細設定には、インスタンスのロケール (文字セットや言語の設定を決定する)、サーバーのログファイルのパス、アプリケーションが配備されているディレクトリのパス、および非活性化された Beans や持続的な HTTP セッションが格納されるセッションストアディレクトリのパスなどがあります。

さらに、アプリケーションの再読み込みや、再読み込みのポーリング間隔を有効にすることもできます。アプリケーションを動的に再読み込みすると、自動的に変更内容のチェックが行われ、変更があった場合は自動的にバージョンが更新されます。通常、動的再読み込みは、本稼働環境ではなく開発環境で行います。ポーリング間隔は、Application Server がアプリケーションの更新をチェックする時間間隔です。

管理インタフェースを使ってアプリケーションサーバーインスタンスの設定を変更するには、次の手順に従います。

1. 左側のペインで、アプリケーションサーバーインスタンス名をクリックします。
2. アプリケーションサーバーインスタンスのページの「詳細」タブをクリックします。
3. フィールドに適切な値を入力します。
4. 「保存」をクリックします。

コマンド行インタフェースの asadmin ユーティリティを使ってサーバーインスタンスの詳細設定を変更するには、get コマンドと set コマンドを使います。

インスタンスのすべての属性を確認するときは、次のコマンドを使います。

```
asadmin get instance_name.*
```

次に例を示します。

```
asadmin get "server1.*"
```

server1 の logRoot という属性を確認するには、次のコマンドを使います。

```
asadmin get server1.logRoot
```

server1 の logRoot という属性を設定するには、次のコマンドを使います。

```
asadmin set server1.logRoot=/space/log
```

上記すべての例は、環境変数にユーザー、パスワード、ホスト、ポートが設定されていることを前提としています。コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。asadmin の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

第 5 章

ログの使用

この章では、Sun ONE Application Server のログ機能について解説します。また、ログを利用可能なコンポーネントについても説明します。

この章では次のトピックについて説明します。

- ログについて
- UNIX および Windows の各プラットフォームでのログ
- ログレベルの使用
- 仮想サーバーとログについて
- ロガーについて
- クライアントサイドのログについて
- アプリケーションログ出力およびサーバーログ出力のリダイレクト
- ログファイルの管理
- コマンド行インタフェースによるログの設定
- 管理インタフェースによるログの設定
- エラーログ指令の設定
- アクセスログファイルの表示
- イベントログファイルの表示
- ログのプリファレンスの設定
- ログアナライザの実行
- イベントの表示 (Windows 2000 Professional)

ログについて

アプリケーションでログを使用すると、デバッグと診断のための便利なツールとなります。また、開発者の生産性を高めるためにも使用できます。アプリケーションサーバーの独自のログ出力を利用すると、サーバーの設定や配備に関する問題の特定と診断を行うことができます。

Sun ONE Application Server のログでは、Java のログ API が使用されます。Sun ONE Application Server は、`logs` ディレクトリにある 2 つのログファイル `access.log` および `server.log` にロギング情報を収集して保存します。独自のログファイルにログを収集することもできます。

ログに記録されたメッセージは、単なるメッセージ以上の情報を提供します。たとえば、次のような追加情報があります。

- イベントの日時
- イベントのログレベル。Loglevel ID または名前で指定された Appserver
- プロセス ID (PID)。appserv プロセスの PID
- 仮想サーバー ID (vsid)。メッセージを生成した vsid (オプション)
- メッセージ ID。サブシステムおよび 4 桁の整数値
- メッセージデータ

追加メッセージ情報の種類と順序は、ログに使用されるプラットフォームと、そのプラットフォームで有効なログサービスによって変わります。ログメッセージの仮想サーバー ID を有効にする方法については、111 ページの「ログサービスの設定」を参照してください。

UNIX および Windows の各プラットフォームでのログ

この節では、ログファイルの作成方法について説明します。次の項目で説明します。

- server.log でのデフォルトのログ
- syslog を使用したログ
- Windows イベントログを使用したログ

server.log でのデフォルトのログ

UNIX と Windows のどちらのプラットフォームでも、ログファイルは `log` サブディレクトリの `server.log` に作成されます。インスタンスのサーバーコンポーネントと仮想サーバーのすべてから得られたログが、このファイルに収集されます。

デフォルトでは、サーバー全体のログレベルを設定できます。ただし、デフォルトのログレベルをオーバーライドして、特定のサブシステムについて設定することも可能です。また、`stdout` と `stderr` をサーバーのイベントログへリダイレクトすることや、オペレーティングシステムのシステムログにログの出力することもできます。さらに、`stdout` と `stderr` の内容をサーバーのイベントログに出力することもできます。デフォルトでは、ログメッセージは、指定されたサーバーログファイルと `stderr` に送られます。

その他、ログメッセージに仮想サーバー ID を加える機能もあります。これは、複数の仮想サーバーを使用する際、同じログファイルへメッセージを記録するのに便利な機能です。ログメッセージをシステムログに書き込むこともできます。その場合、`server.log` ファイルではログされません。ログの生成と管理には、UNIX では `syslog` ログサービスが使用され、Windows ではシステムログサービスが使用されます。

また、`server.xml` 属性を使用して、このファイルの内容を管理することもできます。`server.xml` ファイルの詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

server.log の例

`server.log` の例を次に示します。

タイムスタンプ、ログレベル、(PID vsid(オプション)): メッセージ ID: messageID: メッセージ

```
[01/Aug/2002:11:39:31] INFO ( 1224): CORE1116:Sun ONE Application
Server 7.0
```

```
[01/Aug/2002:11:39:36] INFO ( 1224):CORE5076:Using [Java HotSpot(TM)
Server VM, Version 1.4.0_02-20020712] from [Sun Microsystems Inc.]

[01/Aug/2002:11:39:50] INFO ( 1224):JMS5023: JMS service
successfully started. Instance Name = domain1_server1, Home =
[D:¥install_7_29¥imq¥bin].

[01/Aug/2002:11:39:53] INFO ( 1224):CIS0056:Creating TCP
ServerConnection at [EndPoint
[IIOP_CLEAR_TEXT:192.18.145.66:3700:false]]

[01/Aug/2002:11:39:53] INFO ( 1224):CIS0057:Created TCP
ServerConnection at [EndPoint
[IIOP_CLEAR_TEXT:192.18.145.66:3700:false]]

[01/Aug/2002:11:39:54] INFO ( 1224):CIS0054:Creating TCP Connection
from [-] to [EndPoint [IIOP_CLEAR_TEXT:192.18.145.66:3700:false]]
```

注

コンパイル済み JSP に関するログメッセージのリダイレクト

コンパイル済み JSP に関するログメッセージは、デフォルトで管理サーバーのログファイル ({*domain_root*}/{*domain_name*}/admin-server/logs/server.log) 内に格納されます。

すべてのメッセージが同じファイルに記録されるため、コンパイル済み JSP を使ってアプリケーションを配備する際に発生した例外やエラーが、その共通のログファイル内の膨大なメッセージの中で見失われる危険性があります。複数のアプリケーションを指定されたドメインに属する複数のインスタンスに配備する場合、管理サーバーのログメッセージを注意深く調べて、特定のアプリケーションの JSP に関する例外が発生していないか確認する必要があります。これは冗長な作業です。

したがって、コンパイル済み JSP を使って配備されたアプリケーションに関するメッセージは、管理サーバーの server.log ファイルではなく、サーバーインスタンスの server.log ファイルに記録することをお勧めします。

Sun ONE Application Server インスタンスの server.log ファイルにログメッセージをリダイレクトするには、管理インタフェースでログファイルのパスを変更します。詳細については、「ログサービスの設定」を参照してください。

syslog を使用したログ

ログの一元化が必要となるような安定した動作環境には、syslog の使用が適しています。診断やデバッグのためにログの出力が頻繁に要求されるような環境では、個々のサーバーインスタンスや仮想サーバーのログのほうが管理が容易になります。

注

- サーバーインスタンスおよび管理サーバーのログデータをすべて 1 つのファイルに格納すると、読み取りやデバッグが困難になります。問題なく動作している配備済みアプリケーションだけに、syslog のマスターログファイルを使用することをお勧めします。
- ログに記録されたメッセージには、Solaris デーモンアプリケーションによるその他のログがすべて混在します。

syslog ログファイルと syslogd を連動させ、システムログデーモンと共に使用する場合は、syslog.conf ファイル設定して、次のことを実行できます。

- 適切なシステムログにメッセージを記録する
- メッセージをシステムコンソールに書き出す
- ログに記録されたメッセージを、ユーザーのリストや、ネットワークを介して別のホストの syslogd に転送する

注

Sun ONE Application Server をインストールした時点では、サーバーのログサービス要素属性 use-system-logging は無効です。つまり、デフォルトでは、ログは UNIX の syslog または Windows の Windows イベントログに収集されません。syslog や Windows イベントログにログを収集するには、この属性を server.xml の Server 要素で有効に設定する必要があります。『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。use-system-logging を設定する前に、103 ページの「ログファイルの管理」を参照してください。

syslog の設定

/etc ディレクトリの syslog.conf を設定して、重要度の低いメッセージを個別のファイルに収集すると、管理や読み取りが容易になります。

syslog を設定するには、次の手順に従います。

1. 重要度の低いメッセージを個別のファイルに保存するために、Solaris の syslog.conf ファイルに次のコマンドを追加します。

```
daemon.debug  /var/adm/iasdebug
```

注 Windows のイベントログにログメッセージを収集するときには、INFO、WARNING、SEVERE、ALERT、FATAL の各レベルのメッセージだけが記録されます。

2. syslogd にハングアップシグナルを送ります。それには、次のコマンドを使用します。

```
kill -HUP <PID syslogd>
```

3. 管理インタフェースで管理サーバーに移動し、「システムログに書き込み」オプションを選択します。変更を保存し、適用します。管理サーバーを再起動して、変更を有効にします。

Solaris syslog.conf ファイルの設定例を次に示します。

```
#ident"@(#)syslog.conf1.5 98/12/14 SMI"/* SunOS 5.0 */
#
# Copyright (c) 1991-1998 by Sun Microsystems, Inc.
# All rights reserved.
#
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words. Also, within ifdef's, arguments
# containing commas must be quoted.
#
*.err;kern.notice;auth.notice/dev/sysmsg
*.err;kern.debug;mail.crit/var/adm/messages
daemon.info;daemon.err;daemon.debug;daemon.alert;daemon.crit;daemon
.warning/var/adm/iaslog
daemon.debug/var/adm/iasdebug
#daemon.notice;        /var/adm/iaslognotice
#daemon.warning;       /var/adm/iaslogwarning
#daemon.alert;         /var/adm/iaslogalert
#daemon.err;           /var/adm/iaslogerr
```

```
#*.alert;kern.err;daemon.err operator
#*.alert            root
*.emerg             *

# if a non-loghost machine chooses to have authentication messages
# sent to the loghost machine, un-comment out the following line:
#auth.noticeifdef(`LOGHOST', /var/log/authlog, @loghost)

mail.debugifdef(`LOGHOST', /var/log/syslog, @loghost)
#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
ifdef(`LOGHOST', ,
user.err         /dev/sysmsg
user.err         /var/adm/messages
user.alert       'root, operator'
user.emerg       *
)
```

詳細については、`syslog.conf` のマニュアルページを参照してください。

`syslog.conf` に変更を加えた場合、変更内容を適用するには、Sun ONE Application Server を再起動する必要があります。

`syslog` にログを行うと、Sun ONE Application Server のすべてのログが、その他のデーモンアプリケーションのログと同一のファイルに記録されます。このため、ログに記録されたメッセージには、Sun ONE Application Server 固有のメッセージと、特定のサーバーや仮想サーバーインスタンスのメッセージを区別するために、次のような情報が付加されます。

- 一意のメッセージ ID
- タイムスタンプ
- インスタンス名
- プログラム名 (`appservd` または `appserv-wdog`)
- プロセス ID (appserv プロセスの PID)

- スレッド ID (オプション)
- サーバー ID

サーバーインスタンスと仮想サーバーインスタンスの両方に対するログサービスを、server.xml ファイルで設定できます。仮想サーバーインスタンスのログサービス設定については、98 ページの「仮想サーバーとログについて」を参照してください。サーバーインスタンスのログサービス設定については、111 ページの「管理インタフェースによるログの設定」を参照してください。

ログレベルについては、適用されるサブシステムおよびコンポーネントの管理インタフェースで設定します。

UNIX の動作環境で使用される syslog のログメカニズムの詳細については、端末プロンプトで次の man コマンドを入力してください。

```
man syslog
man syslogd
man syslog.conf
```

syslog メッセージの例

syslog メッセージの例を次に示します。

タイムスタンプ , ホスト名 [インスタンス名], [サブシステム], [vsid], メッセージ ID, ログレベル , メッセージデータ

```
Jul 19 14:33:18 strange /usr/lib/nfs/lockd[164]: [ID 599441
daemon.info] Number of servers not specified. Using default of 20.

Jul 19 14:33:20 strange ntpdate[181]:[ID 558275 daemon.notice]
adjust time server 192.18.56.149 offset 0.06702 6 sec

Jul 19 14:38:13 strange xntpd[248]:[ID 204180 daemon.info]
synchronisation lost

Jul 19 14:38:47 strange server1 appservd[374]:[ID 702911
daemon.info] INFO (  374): CORE1116:Sun ONE Application Server 7.0

Jul 19 14:38:48 strange server1 appservd[374]:[ID 702911
daemon.info] FINE (  374):Collecting statistics for up to 1
processes with 128 threads, 200 listen sockets, and 1000 virtual
servers
```

Windows イベントログを使用したログ

Windows の動作環境で使用されるイベントログメカニズムの詳細については、Windows ヘルプシステムの索引で イベントログというキーワードを探してください。

ログレベルの使用

この節では、ログレベルと、Sun ONE Application Server の各サブシステムにログレベルを割り当てる方法について説明します。

次の項目があります。

- ログレベルについて
- syslog 設定に使用するログレベル

ログレベルについて

Sun ONE Application Server は、標準 JDK 1.4 のログレベルを使用して、ログでの情報選択を行います。Sun ONE Application Server には、標準 JDK のログレベルに加えて、`server.log` とのマッピングをより直観的にすることと、Solaris との統合を密接にすることを目的に設計されたログレベルが追加されています。

ログに記録されたメッセージは、`server.log` に配信されるとき、97 ページの「Sun ONE Application Server のログレベルの `server.log` へのマッピング」の定義に従って、ログレベルにマップします。

注　管理サーバーおよびデフォルトのアプリケーションサーバーインスタンスに対する `server.log` ファイル (または `syslog`) のデフォルトのログレベルは、INFO です。アプリケーションサーバーインスタンスでデフォルトのログレベルを使用する場合は、エラーと情報メッセージが記録されます。こうしたメッセージを記録しない場合は、`server.xml` ファイルのログレベル、または管理サーバーの管理インタフェースとデフォルトのサーバインスタンスに設定されているログレベルを、WARNING または SEVERE に変更します。

サーバーに対するデフォルトのログレベルは、log-service 要素によって設定できます。この設定は、ログレベルが「default」に設定されているすべての要素に反映されます。

ログを有効に設定すると、Sun ONE Application Server サブシステムごとに、ログレベルを割り当てることができます。ログレベルは、実行時に記録されるメッセージ情報の量を調整するのに役立ちます。ログレベルは、目的のサブシステムの `server.xml` ファイルで指定されます。ログレベルを指定するには、選択したサブシステムの管理インタフェースを使用するか、`server.xml` ファイルを直接編集して、選択したサブシステムに適切なログレベルを設定します。

警告 `server.xml` ファイルを手動で編集すると、構文エラーでサーバーの起動が失敗する可能性があります。設定ファイルを手動で編集する方法については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』の「設定ファイルの手動編集」を参照してください。

管理インタフェースでログレベルを設定する例は、101 ページの「JMS サービスのログレベル」の図に示されています。各サブシステムやコンポーネントに対するログレベルを `server.xml` ファイルで直接設定する方法については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

96 ページの「ログレベル」の表に示されているログレベルは、JDK 1.4 ロギング API 仕様の要件に合致しています。ただし、ALERT と FATAL は Sun ONE Application Server 固有のログレベルであり、JDK 1.4 ロギング API では実装されていません。

次の表は、Sun ONE Application Server のメッセージに割り当てられたログレベルを重要度の低いものから順に表しています。左側の列は Sun ONE Application Server でのログレベル指定を示し、右側の列は各ログレベルについての簡単な説明です。

ログレベル

ログレベル	説明
FINEST FINER FINE	デバッグメッセージの冗長性の程度を示すメッセージ。FINEST は最大の冗長性を示す
CONFIG	さまざまな静的設定情報に関連するメッセージ。特定の設定に関連する問題をデバッグする場合に使用できる
INFO	主にサーバー設定またはサーバーの状態に関する通知メッセージ。すぐに対処する必要があるエラーではない たとえば、設定の変更通知が受信され、メッセージブローカーに関する新しいトピックが作成されたというメッセージなどが記録される
WARNING	警告を示すメッセージ。通常、メッセージとともに例外が発行される
SEVERE	通常のアプリケーションの実行を阻害するような極めて重要なイベントを示すメッセージ
ALERT*	ユーザーに特定の対処を行うように警告するメッセージ
FATAL*	サーバーの実行を継続するには適していない致命的なエラーを示すメッセージ。サーバーのクラッシュの直前に表示される典型的なメッセージ

* Sun ONE Application Server に固有のログレベル

注 INFO よりも重大度の低いログレベル (FINEST、FINER、FINE、および CONFIG) のメッセージは、デバッグに関係する問題を解決するための情報が含まれ、テクニカルサポートに問い合わせる際に有効となります。通常、ログレベルが INFO より低いメッセージは、ローカライズされません。

syslog 設定に使用するログレベル

`syslog` の使用時に Sun ONE Application Server で設定できるログレベルを、次の表に示します。左側の列は Sun ONE Application Server でのログレベル指定を示し、右側の列は `syslog` 機能で対応するログレベルを示します。

Sun ONE Application Server のログレベルの `server.log` へのマッピング

Sun ONE Application Server	syslog レベル
FINEST	LOG_DEBUG
FINER	LOG_DEBUG
FINE	LOG_DEBUG
CONFIG	LOG_INFO
INFO (デフォルト)	LOG_INFO
WARNING	LOG_WARNING
SEVERE	LOG_ERR
ALERT	LOG_ALERT
FATAL	LOG_CRIT

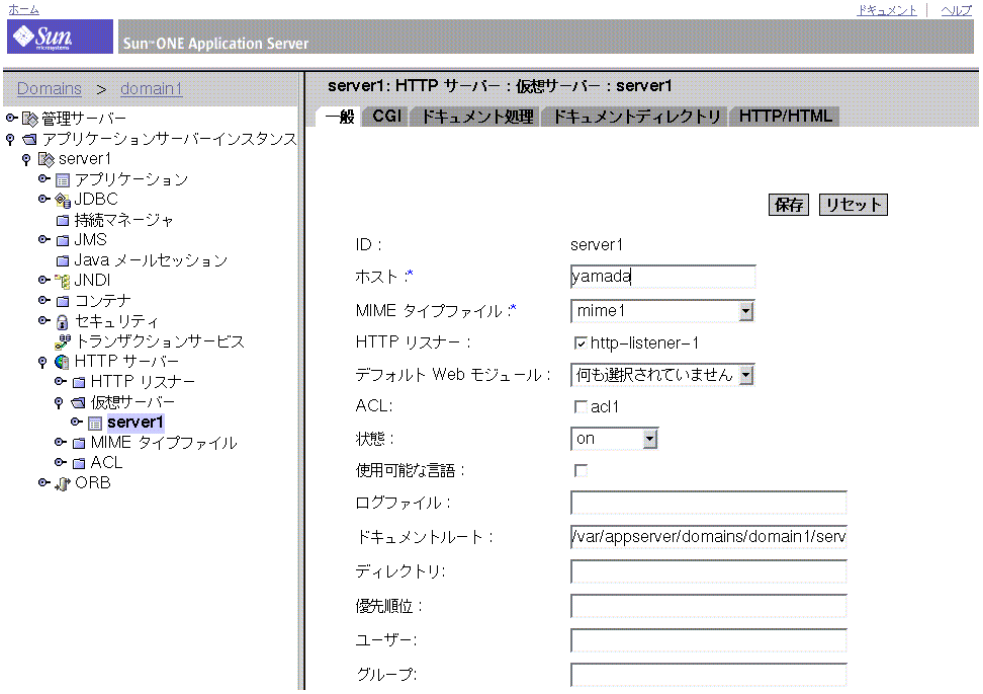
仮想サーバーとログについて

Sun ONE Application Server には、仮想サーバーインスタンスを設定できます。アプリケーションサーバーインスタンス内の各仮想サーバーには、独自の ID があり、独自のログファイルを設定することもできます。仮想サーバーごとにログファイルを割り当てると、特定のトランザクションおよびリソースのサーバーアクティビティを追跡しやすくなります。

管理インタフェースで仮想サーバーのログファイル名を指定するには、ディレクトリツリーから「HTTP サーバー」のリンクへ移動し、仮想サーバーのフォルダ下のサーバーインスタンス要素を開いて、右フレームに「一般」タブを表示します。「ログファイル」フィールドに、仮想サーバーのログファイルのパスと名前を入力します。99 ページの「仮想サーバーログファイル名の設定」の図は、設定する場所を示しています。

注　ログを有効にしてアプリケーションを実行すると、アプリケーションからのログメッセージには、仮想サーバー ID が記録されません。

仮想サーバーログファイル名の設定



ログに記録された複数の仮想サーバーからのメッセージを、単一のサーバーログファイルに直接収集することもできます。この場合は、`server.xml` ファイル内の `log-service` 要素の `log-virtual-server-id` を有効にすることもできます。これによって、異なる仮想サーバーからのログメッセージを区別できます。

```
<log-service level="FINEST" log-stdout="false" log-stderr="false"
echo-log-messages-to-stderr="false" create-console="false"
log-virtual-server-id="true" use-system-logging="false">

</log-service>

   <http-listener>

      <virtual-server-class>

         <virtual-server id="server1"
http-listeners="http-listener-1" hosts="strange" mime="mime1"
state="on" accept-language="false"/>

         <virtual-server id="server2" hosts="strange" mime="mime1"/>

      </virtual-server-class>

   </http-listener>
```

この例では、<log-service log-virtual-server-id="true"> によって、各ログメッセージに virtual_server_id が挿入されます。これによって、異なる仮想サーバーからのメッセージを区別することができます。virtual-server 要素に「log-file」の属性を指定しないと、すべての仮想サーバーから単一の同じファイルへメッセージが記録されます。

ロガーについて

ログの有効化と無効化は、サブシステムのレベルで選択できます。サブシステム単位のログ制御は、server.xml ファイルで指定します。『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。各サブシステムは、JDK 1.4 ログ API の要件を満たす独自のロガーを持ちます。

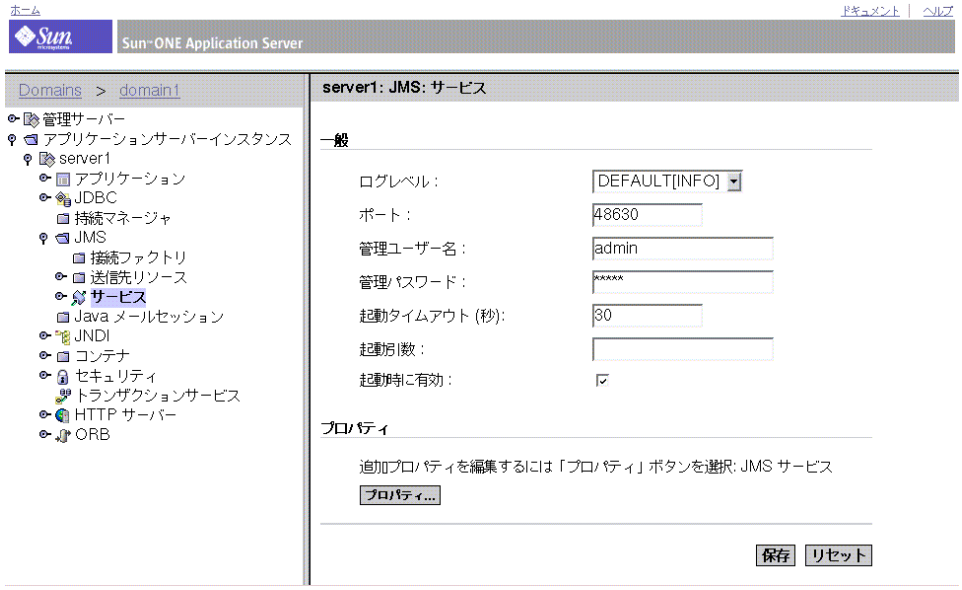
次の表では、左側の列はサブシステムを示し、右側の列は各サブシステムに対応する server.xml ファイルの要素を示します。

Sun ONE Application Server のサブシステムとその場所

サブシステム	要素
管理サーバー	<admin-service>
EJB コンテナ	<ejb-container>
Web コンテナ	<web-container>
MDB コンテナ	<mdb-container>
Sun ONE Message Queue (JMS サービス)	<jms-service>
セキュリティサービス	<security-service>
Java Transaction Service (JTS)	<transaction-service>
Object Request Broker (ORB)	<iiop-service>
デフォルトハンドラ[1]	<log-service>

1. デフォルトハンドラは、特定のサブシステム (ユーティリティクラスなど) に関連付けられていないすべての server.xml エントリに対するデフォルトのロガーになります。

JMS サービスのログレベル



注 Windows プラットフォームでは、Windows の `server.log` にログを収集するよう選択した場合、`INFO`、`WARNING`、`SEVERE`、`ALERT`、`FATAL` の各レベルのメッセージだけが Windows イベントログに記録されます。

96 ページの「ログレベル」の表は、Sun ONE Application Server のメッセージに割り当てられたログレベルを重要度の低いものから順に示しています。これらのログレベルは、JDK 1.4 ログ API 仕様の要件に合致しています。ただし、ALERT と FATAL は Sun ONE Application Server 固有のログレベルであり、JDK 1.4 ログ API ではサポートされていません。

クライアントサイドのログについて

ACC (Application Client Container) には独自のログサービスがあり、ログはローカルファイルにのみ記録されます。

通常、ACC はアプリケーションサーバーとは別のホストで、独自のプロセスによって動作します。そのため、独自のログインフラストラクチャと独自のログファイルを持ちます。ACC の設定は、`sun-acc.xml` ファイルに保持されます。

ACC のクライアントサブシステムログ要素は、`log-service` です。要素と属性を次の表に示します。それぞれ、指定のデフォルト値と値の範囲があります。

ACC のログ要素

要素	属性	説明
`log-service`	file	ACC ログファイル。空または存在しない場合、`stdout` に記録される
`log-service`	level	ACC ログレベル

`sun-acc.xml` ファイルの例は、『Sun ONE 管理者用設定ファイルリファレンス』に記載されています。

アプリケーションログ出力およびサーバーログ出力のリダイレクト

アプリケーションコンポーネントと J2EE アプリケーションの単体テストの実行時には、開発者のためにアプリケーションログとサーバーログを利用できるようにしておく必要があります。Windows プラットフォームでは、開発者はサーバーログメッセージをデスクトップのコマンドウィンドウに表示することを好みます。一方、UNIX プラットフォームでは、サーバーインスタンスを起動した端末ウィンドウ内で `stderr` を実行してログメッセージストリームを出力するか、`tail -f` コマンドを使ってログファイル内に書き込まれたログメッセージを表示するほうが、開発者にとって好都合です。

`server.xml` ファイル内の一部の属性を `stdout` および `stderr` に設定すると、記録されたメッセージをログファイルや端末ウィンドウに出力することができます。`stdout` および `stderr` の詳しい使用方法については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

ログサービスについては、111 ページの「ログサービスの設定」を参照してください。

ログファイルの管理

アクセスログファイルとイベントログファイル (server.log) が自動的にアーカイブされるように設定できます。指定した時刻、または指定した間隔で、ログがローテーションされます。Sun ONE Application Server は、古いログファイルに保存日時を含めた名前を付けて保存します。

注 複数の仮想サーバーを作成し、仮想サーバーごとにログファイルを関連付けることができますが、仮想サーバーごとのログのローテーションはサポートされていません。

たとえば、1 時間おきにアクセスログファイルをローテーションし、このファイルを「access.199907152400」という名前で保存するように設定することができます。ログファイルの名前は、年、月、日、および 24 時間形式の時間が連結された単一の文字列になります。ログアーカイブファイルの形式は、設定したログローテーションの種類によって異なります。

注 これらの機能は、主に Solaris 以外のプラットフォームに提供されます。

Solaris では、これらの機能はデフォルトでは有効になっていません。Solaris のネイティブのオペレーティングシステムログ管理機能 (Solaris 9 では logadm) を使用する必要があります。

ログローテーションには、オペレーティングシステムによって 4 種類の方法があります。以下、これらの方法について説明します。説明する項目は次のとおりです。

UNIX および Windows

- 内部デーモンログローテーション
- スケジューラベースのログローテーション

Solaris 9

- Solaris logadm ユーティリティの使用。詳細については、105 ページの「Solaris logadm ユーティリティを使用したローテーション」を参照してください。

Solaris (任意のバージョン)

- Solaris cron ユーティリティの使用。詳細については、108 ページの「Solaris cron ユーティリティを使用したローテーション」を参照してください。

内部デーモンログローテーション

内部デーモンログローテーションは、UNIX と Windows の両方のオペレーティングシステムで使用できます。内部デーモンログローテーションは HTTP デーモン内で実行され、サーバーインスタンスの起動時だけに設定できます。この方法でローテーションされるログは、次の形式で保存されます。

```
access.<YYYY><MM><DD><HHMM>
```

```
error.<YYYY><MM><DD><HHMM>
```

ログファイルをローテーションし、新しいログファイルの使用を開始するベースとして使用される時間を指定できます。たとえば、ローテーションの開始時刻が午前 0 時で、ローテーションの間隔が 1440 分 (1 日) の場合、現在時刻に関係なく保存直後に新しいログファイルが作成され、ローテーションの開始時刻まで情報を収集します。ログファイルは毎日午前 0 時にローテーションされるため、アクセスログのタイムスタンプは午前 0 時、ファイル名は `access.199907152400` のようになります。同様に、ローテーションの間隔を 240 分 (4 時間) に設定した場合、午前 0 時から 4 時間おきにログがローテーションされます。アクセスログファイルには、午前 0 時から午前 4 時まで、午前 4 時から午前 8 時まで、という順に 4 時間で収集された情報が保存されます。

ログローテーションが有効になっている場合は、サーバーの起動時にログファイルのローテーションが開始されます。ローテーションされる最初のログファイルでは、現在時刻から次のローテーション時刻までの間の情報が収集されます。前の例を使用して、開始時刻を午前 0 時に設定し、ローテーションの間隔を 240 分に設定した場合、現在時刻が午前 6 時とすると、ローテーションされる最初のログファイルには午前 6 時から午前 8 時の間に収集された情報が保存され、次のログファイルには午前 8 時から午後 12 時 (正午) までの間に収集された情報が保存されます。

スケジューラベースのログローテーション

スケジューラによるログローテーションでは、すぐにログファイルをアーカイブすることも、サーバーで特定の日の特定の時間にログファイルをアーカイブするように設定することもできます。ログファイルをその場でアーカイブするには、管理インタフェースの左側のペインで「管理サーバー」を選択します。次に、右のページの最上部にある「ログ」リンクをクリックします。次に、「ログローテーション」をクリックします。最後に、「アーカイブ」をクリックします。

スケジューラを使用する方法でローテーションされるログは、元のファイル名の後にローテーションされた日時が追加された名前で保存されます。たとえば、午後 4 時 30 分にローテーションされる `access` は `access.24Apr-0430PM` という名前になります。

ログローテーションは、サーバーの起動時に初期化されます。ローテーションを有効にすると、Sun ONE Application Server によってタイムスタンプ付きのアクセスログファイルが作成され、サーバーの起動時にローテーションが行われます。

ローテーションが開始されると、事前にスケジューリングされている「次のローテーション時刻」が過ぎてからアクセスログファイルまたはエラーログファイルに記録する必要のある要求やエラーが発生した場合、Sun ONE Application Server により新しいタイムスタンプ付きのログファイルが作成されます。

注	Windows プラットフォームの場合、および Solaris で `syslog` 以外のファイルでサーバーログを収集する場合は、このサーバーログをアーカイブする必要があります。

ログファイルをアーカイブし、`scheduler`d 制御メソッドを使用するには、管理インタフェースの左側のペインから「アプリケーションサーバーインスタンス」を選択してください。次に、右のページの最上部にある「ログ」リンクをクリックします。次に、「ログのローテーション」をクリックし「スケジュールベースのログローテーション」ボックスをクリックします。最後に、「了解」をクリックします。`scheduler` の現在の状態が示されます。

Solaris logadm ユーティリティを使用したローテーション

Solaris 9 オペレーティングシステムには、ログに記録されたメッセージを使って関数の配列を実行するユーティリティ `logadm` が組み込まれています。

Sun ONE Application Server では、Solaris cron ユーティリティからログローテーションタスクを実行する場合に、このユーティリティが役立ちます。109 ページの「Solaris cron ユーティリティを使用した logadm のスケジュール実行」を参照してください。

ログファイルについて、次に示すログローテーションの詳細を指定できます。

- システム上でローテーションする全ログファイル名
- ローテーション間隔
- ローテーションをトリガーする条件
- 保存するバックアップログファイル数
- 保存するバックアップログファイルの名前規則

上記の詳細は、次の場所にある `logadm.conf` ファイルで指定されます。

n /etc/logadm.conf

サンプルの logadm.conf ファイルを、次に示します。

```
# Copyright 2001-2002 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)logadm.conf 1.2 02/02/13 SMI"
#
# logadm.conf
#
# Default settings for system log file management.
# The -w option to logadm(1M) is the preferred way to write to this
# file,
# but if you do edit it by hand, use "logadm -V" to check it for
# errors.
#
# The format of lines in this file is:
# <logname> <;options>
# For each logname listed here, the default options to logadm
# are given. # are given. Options given on the logadm command line
override
# the defaults contained in this file.
# # logadm typically runs early every morning via an entry in
# root's crontab (see crontab(1)).
#
/var/log/syslog -C 8 -P 'Tue Jul 9 10:10:00 2002' -a 'kill -HUP `cat
/var/run/syslog.pid`' /var/adm/messages -C 4 -P 'Tue Jul 30 10:10:00
2002' -a
'kill -HUP `cat /var/run/syslog.pid`' /var/cron/log -c -s 512k -t
/var/cron/olog
/var/lp/logs/lpsched -C 2 -N -t '$file.$N'
#
# The entry below is used by turnacct(1M)
```

```
#
/var/adm/pacct -C 0 -N -a '/usr/lib/acct/accton pacct' -g adm -m 664
-o adm -p never
#
# The entry below will rotate SUN One application server's default
logfile
# every day provided the current logfile size is >= 512k. It will
compress
# the old log file before archiving it and also delete the old files
after 30
# days. The compression is done with gzip(1) and the resulting log
file has
# the suffix of .gz.
/var/appserver/domains/domain1/server1/logs/server.log -A 30d -s
512k -p 1d -z
```

上記とは別に、logadm コマンドを対話形式で呼び出すことにより、特定のファイルでログローテーションを起動することもできます。

次の例では、syslog をローテーションし、8 個のログファイルを保持します。古いログファイルは、/var/log ディレクトリではなく、/var/oldlogs ディレクトリに格納されます。

```
% logadm -C8 -t'/var/oldlogs/syslog.$n' /var/log/syslog
```

対話形式のコマンド行オプションを使用して、/etc/logadm.conf で指定されているファイルでローテーションを行うこともできます。また、オプションを変更することもできます。

/etc/logadm.conf ファイルとコマンド行の両方でオプションが指定されている場合は、/etc/logadm.conf ファイルで指定されているオプションが最初に適用されます。このため、コマンド行オプションが /etc/logadm.conf のオプションをオーバーライドします。次に例を示します。

```
% logadm /var/appserver/domains/domain1/server1/logs/server.log -p
now
```

上記のコマンドは、/etc/logadm.conf で設定されているファイルに対するすべてのオプションを使用して、指定されたファイルをローテーションします。

注　同時に複数のオプションを指定すると、オプション間で暗黙的に AND が適用されます。つまり、ログをローテーションするには、すべての条件が満たされる必要があります。

---

logadm ユーティリティとそのオプションの詳細については、次のように入力してマニュアルページを参照してください。

```
% man logadm
```

または

```
% logadm -h
```

Solaris cron ユーティリティを使用したローテーション

Solaris 8 では、cron ユーティリティを使用して、アプリケーションサーバーのログローテーションを実行できます。次のコマンドを使用します。

```
% crontab -e
```

環境変数 $EDITOR で定義されたエディタが起動し、cron エントリのリストが表示されます。

注 このコマンドでは、エディタの終了後、すぐに /etc/cron.d/logchecker スクリプトが実行されます。このスクリプトは、変更または追加された crontab のエントリを cron デーモンに送ります。cron デーモンはこの方法で追加されたエントリをただちにピックアップして、ログローテーションがすぐに起動します。

ログローテーションを有効にするために cron デーモンを再起動する必要はありません。

この節には次の項目があります。

- crontab エントリの形式について
- Solaris cron ユーティリティを使用した logadm のスケジュール実行

crontab エントリの形式について

crontab ファイルの各行には、6 個のフィールドを設定します。各フィールドは、空白またはタブで区切られます。最初の 5 個のフィールドは、次の項目を指定する整数値です。

- 分 (0-59)
- 時間 (0-23)
- 日付 (1-31)

- 月 (1-12)
- 曜日 (0-6、0 が日曜日)

この形式を使用して、アクセスログファイルとイベントログファイルのローテーション間隔と、ローテーションの繰り返しのスケジュールを設定します。例を示します。

0 0 * * 1-5

```
/opt/SUNWappserver7/appserver/domains/domain1/server1/bin/rotatelogs
```

0 12 * * 1-5

```
/opt/SUNWappserver7/appserver/domains/domain1/server1/bin/rotatelogs
```

0 * * * 1-5

```
/opt/SUNWappserver7/appserver/domains/domain1/mainserver/bin/rotate
logs
```

この例では、server1 のアクセスログファイルとイベントログファイルが、月曜日から金曜日までの毎日、午前 0 時と正午にローテーションされます。mainserver のログファイルは、月曜日から金曜日までの毎日、1 時間ごとにローテーションされます。

crontab ファイルは、/var/spool/cron/crontabs/ に格納されます。crontab ファイルの作成は、エンドユーザーとしても root としても行うことができます。ユーザーの権限によって、次のコマンドで見ることのできる crontab エントリは異なります。

```
% crontab -l username
```

Solaris cron ユーティリティを使用した logadm のスケジュール実行

cron コマンドは、指定された日時にコマンドを実行するプロセスを起動します。/var/spool/cron/crontabs ディレクトリの crontab ファイルで見つかった命令に基づいて、コマンドが定期的なスケジュールで設定されます。

cron で使用される定期的にスケジュールされたコマンドの例として、次の crontab エントリは logadm を毎日午前 0 時に起動します。

```
0 0 * * 0-6 logadm
```

ユーザーは、crontab(1) コマンドを使用する独自の crontab ファイルを送信できることに注意してください。

cron が実行するすべてのアクションに関するログを保持するには、CRONLOG=YES (デフォルト) を /etc/default/cron ファイルで指定する必要があります。/etc/cron.d/logchecker は、ログファイルがシステムの ulimit を超過していないかどうかをチェックするスクリプトです。超過している場合、ログファイルは /var/cron/olog に移動されます。

コマンド行インタフェースによるログの設定

サーバーインスタンスおよび仮想サーバーインスタンスのコマンド行から、ログサービスの内容を設定できます。

注	この節の例では、すべてのコマンドは、環境変数が設定済みであることを前提としています。

サーバーインスタンスのすべての log-service 属性を取得するには、次のように入力します。

```
asadmin> get instance_name.log-service.*
```

log-service 属性は、113 ページの「ログサービス属性」の表に説明されています。

サーバーインスタンス名を指定したコマンド例を次に示します。

```
asadmin> get server1.log-service.*
```

server1 インスタンスのログサービスに関する属性の一覧が表示されます。`set` コマンドを使用すると、表示された属性をそれぞれ設定できます。

仮想サーバーインスタンスの仮想サーバー ID をログに記録するには、端末プロンプトに次のコマンドを入力します。

```
asadmin> get instance_name.LogVirtualServerId
```

LogVirtualServerId の現在の状態が表示されます。状態が false の場合は、次のように `set` コマンドを使用して有効にできます。

```
asadmin> set instance_name.LogVirtualServerId=true
```

仮想サーバーインスタンスのログファイル名を設定するには、次の `set` コマンドを使用します。

```
asadmin> set instance_name.virtual-server.<virtual server
id>.logFile=<log file>
```

たとえば、次のようなログファイルの設定コマンドを発行できます。

```
asadmin> set
instance2.virtual-server.instance2.logFile=/space/IAs7se/appserver7/appse
rv/domains/domain1/instance2/logs/log
```

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。

`asadmin` の使用方法の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

管理インタフェースによるログの設定

この節では、Sun ONE Application Server の管理インタフェースを使って、サーバーワイド (グローバル) な要素、指令、およびアプリケーションコンポーネントに適用できるログサービスオプションを設定する方法を説明します。

この節には次の項目があります。

- ログサービスの設定
- アプリケーションサーバーコンポーネントおよびサブシステムのログ設定
- エラーログ指令の設定

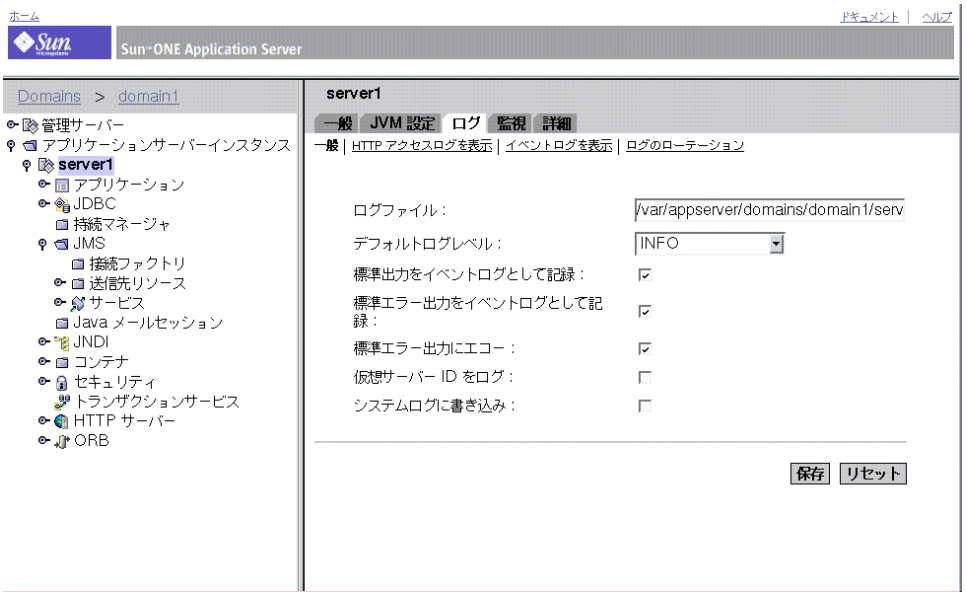
ログサービスの設定

ログサービスは、`server.xml` ファイルの J2EE サービス要素カテゴリ内の要素で指定します。『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。ログサービスは、次のログファイルを含むシステムログサービスを設定するために使用されます。

- サーバーログ
- アクセスログ
- トランザクションログ
- 仮想サーバーログ

システムログサービスの設定では、ログサービス要素の属性値を指定します。

サービスインスタンスのログサービス管理



112 ページの「サービスインスタンスのログサービス管理」の図で示されるように、管理インタフェースでは、次に示すログサービス要素の属性を設定できます。

- ログファイル
- デフォルトログレベル
- 標準出力をイベントログとして記録
- 標準エラー出力をイベントログとして記録
- 標準エラー出力にエコー
- コンソールを作成
- 仮想サーバー ID をログ
- システムログに書き込み

ログサービスのリンクには、管理インタフェースの左側のペインで、サーバーインスタンスのツリー階層を展開するとアクセスできます。次の表に、設定可能な属性とそのデフォルト値および許値の範囲を示します。

ログサービス属性

属性	デフォルト値	説明
file	server.log[1]	省略可能。サーバーログの名前または場所をオーバーライドする。サーバーログが格納されているファイルおよびディレクトリは、サーバーを実行するユーザーアカウントに関係なく、常に書き込み可能な状態にしておくことが必要
level	INFO	省略可能。別の要素によってサーバーログに記録されるメッセージのデフォルトタイプを制御する。有効な値 (降順) は以下のとおり。FINEST、FINER、FINE、CONFIG、INFO、WARNING、SEVERE、ALERT、FATAL 各値は、それよりも低いレベルの値のすべてのメッセージをログに記録する。たとえば、FINEST ではすべてのメッセージが記録され、FATAL では FATAL メッセージだけが記録される。デフォルト値は INFO であり、INFO、WARNING、SEVERE、ALERT、FATAL のすべてのメッセージがログに記録される
log-stdout	True	省略可能。true の場合、stdout 出力がサーバーログにリダイレクトされる。有効な値は on、off、yes、no、1、0、true、false
log-stderr	True	省略可能。true の場合、stderr 出力がサーバーログにリダイレクトされる。有効な値は on、off、yes、no、1、0、true、false
echo-log-messages-to-stderr	True	省略可能。true の場合、サーバーログに加えてログメッセージが stderr に送信される。有効な値は on、off、yes、no、1、0、true、false
create-console	False	省略可能。true の場合、stderr 出力用に Windows オペレーティングシステムのコンソールウィンドウが作成される。有効な値は on、off、yes、no、1、0、true、false
log-virtual-server-id	False	省略可能。true の場合、仮想サーバー ID が仮想サーバーログに表示される。複数の virtual-server 要素が同じログファイルを共有している場合に有用
use-system-log	False	true の場合、UNIX syslog サービスまたは Windows イベントログを使ってログの作成と管理が行われる

1. server 要素の log-root 属性で指定されたディレクトリ

アプリケーションサーバーコンポーネントおよびサブシステムのログ設定

この節では、ログを有効にし、Sun ONE Application Server コンポーネントおよびサブシステムのログレベルを選択する方法について説明します。Java トランザクションサービスコンポーネントには、複数のログファイルがあります。ログレベルの設定方法は、ほとんどのコンポーネントおよびサブシステムに共通です。そのため、ログレベルを選択する手順は、指定したコンポーネントおよびサブシステムのグループについて 1 回だけ記載します。

次のコンポーネントおよびサブシステムでは、サーバーメッセージのログ方法を選択できます。これらのコンポーネントおよびサブシステムの詳細については、このマニュアルで説明されているその他の項目を参照してください。

- ORB -「CORBA ベースのクライアントのサポートの設定」
- Web コンテナ -「J2EE サービスの設定」
- EJB コンテナ -「J2EE サービスの設定」
- MDB コンテナ -「J2EE サービスの設定 (EJB コンテナ内)」
- Java トランザクションサービス -「J2EE サービスの設定」
- JMS サービス -「Java Message Service」
- 仮想サーバー -「仮想サーバーの使用」

ログレベルの指定方法

ORB、Web コンテナ、EJB コンテナ、MDB コンテナ (EJB コンテナ内)、Java トランザクションサービス、および JMS サービスのログレベルを指定するには、次の手順に従います。

1. 管理インタフェースの左側のペインで、アプリケーションサーバーインスタンスを展開し、編集するコンポーネントおよびサブシステムを表示します。
2. 編集するコンポーネントまたはサブシステムのリンクをクリックします。
3. 管理インタフェースの右側のペインで、「ログレベル」ドロップダウンリストから、いずれかのログレベルパラメータを選択します。ログレベルについては、95 ページの「ログレベルについて」を参照してください。

ログファイルの指定方法 (仮想サーバー)

ログファイルを指定するには、次の手順に従います。

1. 管理インタフェースの左側のペインで、アプリケーションサーバーインスタンスを展開し、HTTP サーバーサブシステムを表示します。

2. 「HTTP サーバー」リンクをクリックします。
3. 「仮想サーバー」リンクをクリックします。
4. 編集するサーバーインスタンスのリンクをクリックします。
5. 管理インタフェースの右側のペインの「一般」タブで、「ログファイル」フィールドに編集するディレクトリパスとファイル名を入力します。

トランザクションログの場所の指定方法 (Java トランザクションサービス)

トランザクションログの場所を指定するには、次の手順に従います。

1. 管理インタフェースの左側のペインで、アプリケーションサーバーインスタンスを展開し、トランザクションサービスサブシステムを表示します。
2. 「トランザクションサービス」リンクをクリックします。
3. 管理インタフェースの右側のペインの「詳細」フィールドグループで、「トランザクションログの位置」フィールドに編集するディレクトリパスとファイル名を入力します。

エラーログ指令の設定

Sun ONE Application Server の `init.conf` ファイルには、エラーログ指令が含まれています。次の指令があります。

- **エラーログの日時形式** : `ErrorLogDateFormat` 指令では、サーバーログで使用する日付形式を指定する
- **ログのフラッシュ間隔** : `LogFlushInterval` は、アクセスログがメモリから `access.log` ファイルへフラッシュされるまでの最大間隔 (秒単位) を決定する
- **Pid ログ** : `PidLog` では、ベースサーバープロセスのプロセス ID (pid) を記録するファイルを指定する。サーバーサポートプログラムによっては、このログがサーバールートの `logs/pid` にあると見なす場合もある

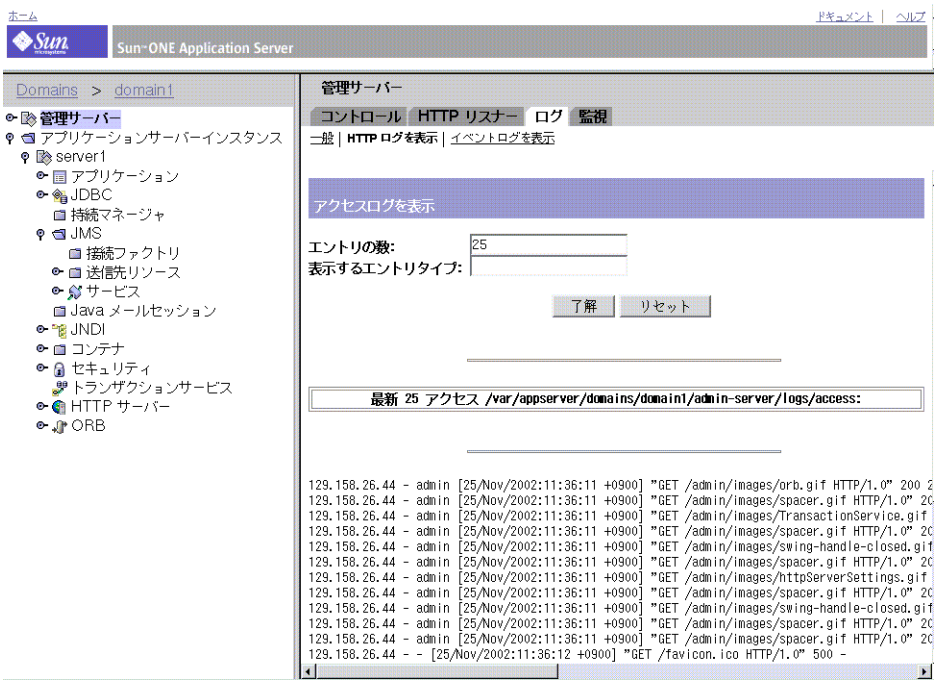
`init.conf` の全指令の詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

アクセスログファイルの表示

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のどちらの HTTP ログファイルも表示できます。

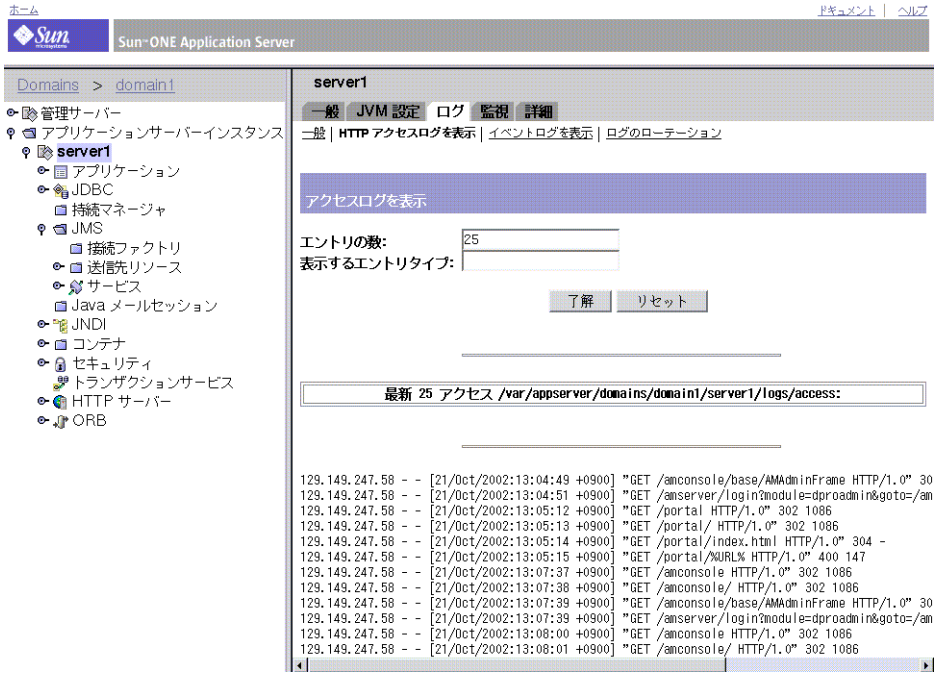
管理サーバーの HTTP ログを表示するには、管理インタフェースの左側のペインで「管理サーバー」を選択し、右側のページで「ログ」タブを選択します。「HTTP アクセスログを表示」リンクが表示されます。このリンクを選択すると、設定済みのアクセスログが表示されます。表示されるログの例は、116 ページの「管理サーバーの HTTP アクセスログ表示」の図に示されています。

管理サーバーの HTTP アクセスログ表示



アプリケーションサーバーインスタンスのアクセスログを表示するには、管理インタフェースの左側のペインで表示するサーバーインスタンスをクリックします。右側のペインで「ログ」タブをクリックします。表示したいログのリンクをクリックすると、該当のサーバーインスタンスのアクティブな設定済みアクセスログが表示されます。117 ページの「アプリケーションサーバーインスタンスのアクセスログ表示」の図に例を示します。

アプリケーションサーバーインスタンスのアクセスログ表示

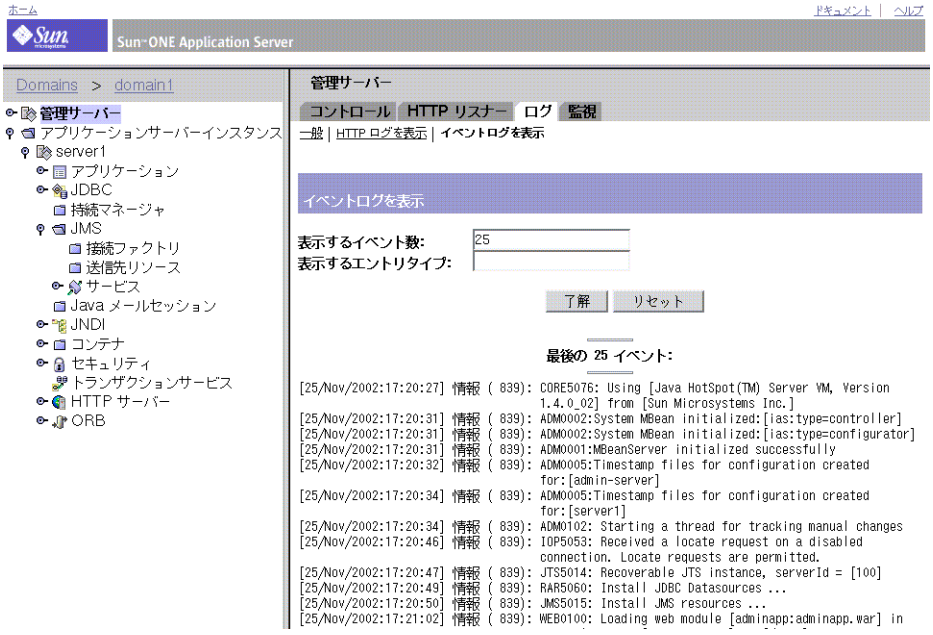


イベントログファイルの表示

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のどちらのアクティブなイベントログファイルも表示できます。

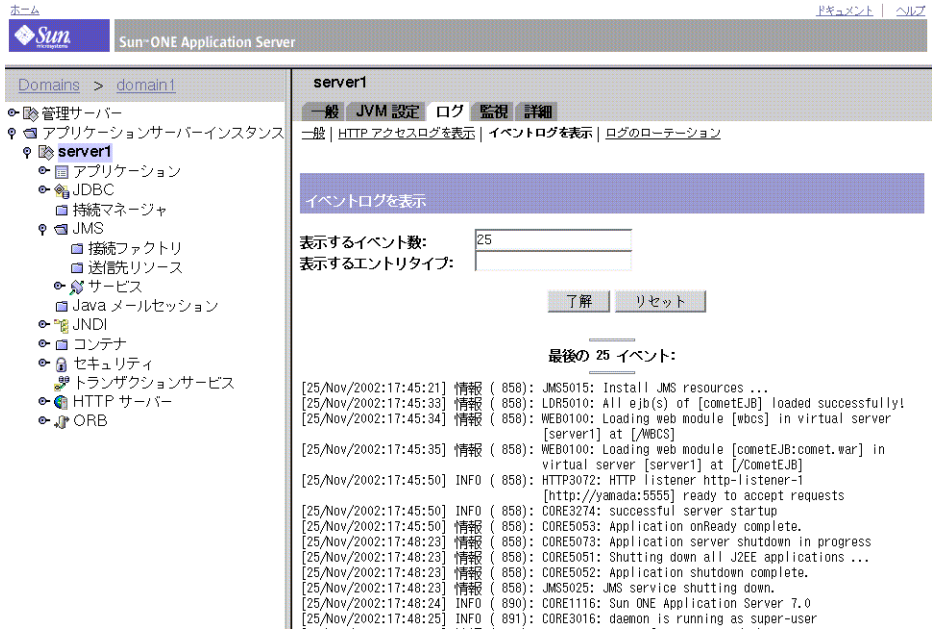
管理サーバーのイベントログを表示するには、左側のペインで「管理サーバー」を選択し、右側のページで「ログ」タブを選択します。「イベントログを表示」リンクが表示されます。このリンクを選択すると、設定済みのイベントログが表示されます。表示されるログの例は、118 ページの「管理サーバーのイベントログ表示」の図に示されています。

管理サーバーのイベントログ表示



管理サーバーインスタンスのイベントログを表示するには、管理インタフェースの左側のペインでサーバーインスタンスを選択し、右側のペインで「ログ」タブを選択します。「イベントログを表示」リンクが表示されます。このリンクを選択すると、設定済みのイベントログが表示されます。表示されるログの例は、119 ページの「アプリケーションサーバーインスタンスのイベントログ表示」の図に示されています。

アプリケーションサーバーインスタンスのイベントログ表示



ログのプリファレンスの設定

インストール時に、サーバーに対して access という名前のアクセスログファイルが作成されます。アクセスをログに記録するかどうか、どのような形式でログを行うか、およびクライアントがリソースにアクセスした場合にサーバーでそのクライアントのドメイン名を検索する必要があるかどうかを指定することによって、リソースに対するアクセスログをカスタマイズできます。

複数の仮想サーバーに対して 1 つのログファイルを使用するには、server.xml ファイルの LogVsId をイベントログに関して有効に設定する必要があります。詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。LogVsID の設定は、管理インタフェースの「管理サーバー」の「ログ」タブでも行うことができます。

管理インタフェースで「仮想サーバー ID をログ」を有効にするには、次の手順に従います。変更内容は管理サーバーの再起動後に反映されます。

1. 管理インタフェースの左側のペインで「管理サーバー」をクリックします。
2. 右側のページで「ログ」タブをクリックします。
3. 「仮想サーバー ID をログ」のチェックボックスをクリックします。
4. 「保存」ボタンをクリックして、変更を Sun ONE Application Server に適用します。

この設定に対する変更を有効にするには、Sun ONE Application Server を再起動する必要があります。

ログアナライザの実行

flexanlg は、ログファイルの報告に使用するログアナライザツールです。ログアナライザは、syslog 以外のファイルにログを記録する場合にのみ使用できます。

ログアナライザを使用すると、アクティビティの要約情報、もっとも頻繁にアクセスされる URL、サーバーがもっとも頻繁にアクセスされる時間など、デフォルトサーバーの統計情報を生成できます。ログアナライザでは、デフォルトサーバー以外の仮想サーバーの統計情報を生成できません。ただし、116 ページの「アクセスログファイルの表示」で説明されているように、各仮想サーバーの統計情報を参照することは可能です。

注 ログアナライザを実行する前に、サーバーログのローテーションを行う必要があります。詳細については、103 ページの「ログファイルの管理」を参照してください。

コマンド行からログアナライザコマンドを実行するには、flexanlg ツールを実行します。このツールは *install_dir*/bin/flexanlg ディレクトリにあります。

flexanlg を実行するには、コマンドプロンプトに次のコマンドとオプションを入力します。

```
flexanlg [ -P ] [-n name] [-x] [-r] [-p order] [-i file]* [ -m
metafile ]* [ o file][ c opts] [-t opts] [-l opts] [-h help]
```

コマンドオプション (* の付いたオプションは繰り返し可能)

-i *filename*

入力ログファイル （ 複数可 ）

-P

プロキシログ形式

-n *servername*

サーバーの名前

-x

HTML で出力

-r

IP アドレスをホスト名で解決

-p [c, t, l]

出力順。デフォルトでは、カウント数 (c)、時間統計情報 (t)、一覧 (l) の順

-m *filename*

メタファイル （ 複数可 ）

-o *filename*

出力ログファイル。デフォルトは stdout

-c [h, n, r, f, e, u, o, k, c, z]

以下の項目のカウント数。デフォルトは h, n, r, e , u, o, k, c

- h: 合計ヒット数
- n: 304 Not Modified 状態コード （ ローカルコピーを使用 ）
- r: 302 Found 状態コード （ リダイレクト ）
- f: 404 Not Found 状態コード （ ドキュメントが見つからない ）
- e: 500 Server Error 状態コード （ 設定ミス ）
- u: 一意の URL の合計数
- o: 一意のホストの合計数

k: 転送された合計 K バイト数

c: キャッシュによって保存された合計 K バイト数

z: どの項目もカウントしない

-t [sx, mx, hx, xx, z]

一般的な統計情報を検索。デフォルトは s5m5h24x10

s (number): ログの上位 number 個を秒単位で検索

m (number): ログの上位 number 個を分単位で検索

h (number): ログの上位 number 個を時間単位で検索

u (number): ログの上位 number 個をユーザー単位で検索

a (number): ログの上位 number 個をユーザーエージェント単位で検索

r (number): ログの上位 number 個を参照元単位で検索

x (number): ログの上位 number 個をその他のキーワードで検索

z: 一般的な統計情報を検索しない

-l [cx, hx]

指定されたサブオプションの一覧を作成。デフォルトは c+3h5

c (x, +x): 最も頻繁にアクセスされた URL

x: x 個のエントリのみ一覧表示

+x: アクセス回数が x を超えた場合のみ一覧表示

h (x, +x): サーバーに最も頻繁にアクセスしたホストまたは IP アドレス

x: x 個のエントリのみ一覧表示

+x: アクセス回数が x を超えた場合のみ一覧表示

z: 一覧を作成しない

例 : flexanlg コマンドの使用

```
flexanlg -i
/var/opt/SUNQappserver7/domains/domain1/server1/logs/access
```

注 ログアナライザを実行する前に、サーバーログのアーカイブを行う必要があります。

イベントの表示 (Windows 2000 Professional)

Sun ONE Application Server は、`server.log` ファイルにエラーのログを記録し、イベントビューアには深刻なシステムエラーのログを記録します。イベントビューアでは、システム上のイベントを監視できます。イベントビューアには、基本設定の問題によって発生したエラーを表示できます。基本設定による問題は、エラーログが記録される前に発生することがあります。

イベントビューアを使用するには、次の手順に従います。

1. 「スタート」メニューから、「プログラム」、「管理ツール」の順に選択します。「管理ツール」プログラムグループの「イベントビューア」を選択します。
2. 「ログ」メニューから「アプリケーション」を選択します。

   イベントビューアにアプリケーションログが表示されます。Sun ONE Application Server のエラーには、`https-serverid` というソースラベルが付いています。
3. こうしたラベルをログ内で検索するには、「表示」メニューから「検索」を選択します。アップデートされたログエントリを表示するには、「表示」メニューから「最新の情報に更新」を選択します。

イベントビューアの詳細については、ご使用のシステムのマニュアルを参照してください。

第 6 章

Sun ONE Application Server の監視

この章では、監視の方法と、Sun ONE Application Server で使用できる SNMP (Simple Network Management Protocol) の特徴および機能について説明します。

この章には次の節が含まれます。

- Sun ONE Application Server の監視について
- CLI を使用した監視データの抽出
- CLI によるトランザクションサービスの管理
- HTTP サービス品質の使用
- SNMP について
- SNMP の設定
- SNMP マスターエージェントの有効化と起動

Sun ONE Application Server の監視について

システム上の重要なデータポイントから稼働統計情報を収集することにより、Sun ONE Application Server を監視できます。統計情報は、サーバーが処理している要求の数と、その処理状況を示します。個々の仮想サーバーに関する統計情報や、アプリケーションサーバーインスタンス全体に関する統計情報を表示できます。Sun ONE Application Server の監視には、`asadmin` ユーティリティや SNMP を使用します。

この節では次の項目について説明します。

- 統計情報
- SNMP
- HTTP サーバーの監視
- アプリケーションコンポーネントとサブシステムの監視

- サービス品質 (QOS)

統計情報

HTTP サーバーなどほとんどの Sun ONE Application Server のアプリケーションコンポーネントとサブシステムの統計情報は、特に監視機能を有効に設定しなくても、常に収集されます。ただし、サブシステムでの監視を明示的に有効に設定した場合や、それと同等の機能を有効に設定した場合だけ収集される統計情報もあります。これらの統計情報には、次のデータポイントが含まれます。

- EJB メソッドの統計情報
- アクティブなトランザクション
- 接続 (QOS が有効な場合のみ)
- DNS (DNS キャッシュが有効な場合のみ)

管理インタフェースからアプリケーションのサブシステムまたはコンポーネントの監視機能を有効にする方法については、127 ページの「アプリケーションコンポーネントとサブシステムの監視」を参照してください。

サーバーモニターを通してサーバーが多数の要求を処理していることが判明した場合、要求数に合わせてサーバー設定またはシステムのネットワークカーネルを調整する必要があります。サーバー設定の調整方法の詳細については、『Sun ONE Application Server パフォーマンスおよびチューニングガイド』を参照してください。

SNMP

Sun ONE Application Server は、SNMP (Simple Network Management Protocol) を使用する情報収集ツールによって、ネットワーク管理のための情報を提供します。SNMP は、ネットワークで管理情報や監視情報を交換するためのプロトコルです。SNMP を使用するエージェントと呼ばれるプログラムが、ネットワーク上のさまざまなデバイス (ハブ、ルーター、ブリッジなど) を監視します。別のプログラムが、エージェントから受け取ったデータを収集します。オペレーションの監視によって作成されたデータベースは MIB (management information base) と呼ばれます。このデータを使用して、ネットワーク上のすべてのデバイスが適切に動作していることをチェックします。

SNMP で監視できるのは HTTP サーバーだけですが、コマンド行インタフェース (CLI) を使用するとすべてのコンポーネントとシステムを監視できます。

SNMP の詳細については、157 ページの「SNMP について」および 165 ページの「SNMP の設定」を参照してください。

HTTP サーバーの監視

HTTP サーバーの監視は、デフォルトで有効になっているため、有効に設定する作業は不要です。HTTP サーバーの監視は XML ファイルをベースにしていて、管理可能な 3 種類の属性一式には、asadmin コマンドを使用してアクセスできます。この XML ファイルの要素、サブ要素、および属性については、142 ページの「監視可能な HTTP サーバー要素」および 144 ページの「監視可能な HTTP サーバー属性」を参照してください。

注 SNMP では、HTTP サーバーの統計情報だけを利用できます。HTTP サーバーも含めた Sun ONE Application Server のすべてのサブシステムに関する統計情報を利用するには、コマンド行インタフェースを使用します。

asadmin の使用方法の詳細については、411 ページの「コマンド行インタフェースの使用」を参照してください。

アプリケーションコンポーネントとサブシステムの監視

Sun ONE Application Server のサブシステムやコンポーネントには、関連する統計情報が常に収集されているために、監視を有効に設定する必要がないものもあります。たとえば、コンテナなどのアプリケーションコンポーネントでは、監視機能を有効にしても無効にしても構いません。監視機能を有効にすると、常時収集される統計情報に加えて、すべての EJB メソッドに関する追加の統計情報も収集されます。JDBC 接続プールの監視は、常に有効になっています。接続プールは最初にアクセスされたときに初期化され、その後は常に関連する統計情報が監視されます。

監視対象となるデータポイントの詳細については、137 ページの「監視可能な属性名」を参照してください。

管理インタフェースまたはコマンド行インタフェース (CLI) を使って、選択した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監視機能を有効にすることができます。たとえば、CLI から EJB コンテナの監視機能を有効に設定するには、端末ウィンドウで次にコマンドを入力します。

```
asadmin set server1.ejb-container.monitoringEnabled=true
asadmin reconfig server1
```

server1 はインスタンス名です。

上記と同等の機能には、「コンテナ (Containers)」ノードの下の管理インタフェースからアクセスできます。

この節では次の項目について説明します。

- コンテナサブシステムの監視
- ORB サービスの監視
- トランザクションサービスの監視

コンテナサブシステムの監視

EJB コンテナの場合、監視機能を有効にすると、エンティティ Beans、ステートフルセッション Beans、およびステートレスセッション Beans のメソッドに関連する統計情報が収集されます。次の統計情報があります。

- 合計エラー数
- 合計呼び出し数
- 合計成功数
- ミリ秒単位の実行時間 (最終のメソッド呼び出しが対象)

コンテナサブシステムでは、その他すべての統計情報が常に収集されます。監視されるデータポイントには、次のような統計情報が含まれます。

- プール内のステートレス Beans の初期数、最小数、および最大数
- キャッシュ内のステートフル Beans とエンティティ Beans の最小数およびユーザー設定数
- キャッシュ内のステートレス Beans の最小数およびユーザー設定数
- 作成された Beans 数と破棄された Beans 数
- その他の関連統計情報

ORB サービスの監視

ORB サービスの場合、監視されるデータポイントには、ORB 接続および ORB スレッドプール用に収集される統計情報が含まれます。ORB 統計情報は常に収集されるため、ORB サービスの監視機能を有効に設定する必要はありません。

トランザクションサービスの監視

Java トランザクションサービス (JTS) の場合、監視されるデータポイントには、次の統計情報が含まれます。

- 完了したトランザクションの合計
- ロールバックされたトランザクションの合計
- 処理中のトランザクションの合計

- 処理中のトランザクションのリスト

詳細については、151 ページの「CLI によるトランザクションサービスの管理」を参照してください。

サービス品質 (QOS)

サービス品質は、サーバーインスタンスの仮想サーバークラス、または仮想サーバーに対して設定するパフォーマンスの制限です。たとえば、ISP (Internet Service Provider) であれば、許可する帯域幅に応じて仮想サーバーの課金額を変えたい場合があります。この場合、帯域幅の量と接続数に制限を課すことができます。

Sun ONE Application Server が提供するサービス品質を使用すると、次の項目に関して実行時のサーバーの効率を判断できます。

- 起動時間
- サーバートラフィック、および帯域幅に対するトラフィックの影響
- ライブデータと静的データの分析
- その他のデータ要素

詳細については、151 ページの「CLI によるトランザクションサービスの管理」を参照してください。

CLI を使用した監視データの抽出

コマンド行インタフェース (CLI) で asadmin コマンドに list コマンドや get コマンドを使用すると、監視されたデータを抽出できます。

注	set コマンドは、トランザクションサービスの監視を設定する場合だけ使用されます。151 ページの「CLI によるトランザクションサービスの管理」を参照してください。

この節では次の項目について説明します。

- list --monitor コマンド
- get --monitor コマンド
- CLI ネームマッピング
- HTTP サーバーの監視可能オブジェクト

list --monitor コマンド

list コマンドは、指定されたサーバーインスタンス名について、現在監視されているアプリケーションコンポーネントおよびサブシステムに関する情報を提供します。このコマンドを使用すると、サーバーインスタンスで監視可能なコンポーネントおよびサブコンポーネントを表示できます。

例

```
asadmin>list --monitor server1
```

監視機能が有効になっている次の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が返されます。

```
iiop-service
transaction-service
application.converter
application.myApp
http-server
```

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できます。この機能は、get コマンドを使ってアプリケーションから特定の監視統計情報を取得する際に役立ちます。

例

```
asadmin> list --monitor server1.application
```

戻り値

```
converter
myApp
```

より詳細な例については、133 ページの「Petstore の例」を参照してください。

get --monitor コマンド

このコマンドは、次の監視情報を取得します。

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

特定のコンポーネントまたはサブシステムについて、要求された属性が存在しない場合は、エラーが返されます。同様に、コンポーネントまたはサブシステムについて要求された特定の属性がアクティブでない場合も、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、132 ページの「CLI ネームマッピング」を参照してください。

例 1

特定の属性について、サブシステムからすべての属性を取得する例

```
asadmin> get --monitor server1.iiop-service.orb.system.orb-connection.*

total-inbound-connections=1
total-outbound-connections=1
```

例 2

J2EE アプリケーションからすべての属性を取得する例

```
asadmin> get --monitor server1.application.converter.*

Attribute name(s) not found
```

J2EE アプリケーションレベルで公開されている監視可能な属性はないため、コマンドは失敗します。

例 3

サブシステムから特定の属性を取得する例

```
asadmin> get --monitor server1.transaction-service.inflight-tx

Attribute name = inflight-tx Value = No active transaction found.
```

例 4

サブシステム属性内に存在しない属性を取得する例

```
asadmin> get --monitor server1.iiop-service.orb.system.orb-connection.bad-name
             Could not get the attribute
Execution failed for the command: get --monitor
server1.iiop-service.orb-connection.bad-name
```

CLI ネームマッピング

Sun ONE Application Server では、ツリー構造によって、監視対象オブジェクトを探すことができます。ツリー内のノードごとに名前とタイプがあります。タイプが単独の場合は、親ノードの下には、そのタイプのノードが 1 だけ存在します。ツリー内のノードタイプの詳細については、135 ページの「監視可能なオブジェクトタイプ」を参照してください。

ツリーのルートオブジェクトは、Sun ONE Application Server のインスタンス名で表されます。たとえば、`server1` と名前付けされているインスタンスのルートの監視オブジェクトは、次のようになります。

```
server1
```

すべての子オブジェクトのアドレスは、ドット (.) で区切って指定されます。子ノードが単独の場合、その監視オブジェクトのアドレスを指定するには、オブジェクトのタイプだけが必要となります。単独でない場合は、オブジェクトのアドレス指定には、`type.name` の形式の名前が必要となります。

たとえば、単独の有効な監視可能オブジェクトタイプとしては、`http-server` などがあります。インスタンス `server1` の `http-server` を表す単体の子ノードをアドレス指定する場合、名前は次のようになります。

```
server1.http-server
```

また、単独ではない有効な監視可能オブジェクトタイプとしては、`application` などがあります。アプリケーション Petstore を表す単独ではない子ノードをアドレス指定する場合、名前は次のようになります。

```
server1.application.petstore
```

CLI 名では、監視可能オブジェクトの特定の属性をアドレス指定することもできます。たとえば、`http-server` には、監視可能属性の `summary` があります。次の名前は、summary 属性のアドレス指定となります。

```
server1.http-server.summary
```

監視オブジェクトが公開する属性名に対するネーミング規則は定められていません。

CLI で使用するための有効な名前が分からない場合もあります。list コマンドを使用すると、有効な監視可能オブジェクトを調べることができます。get コマンドにワイルドカードのパラメータを指定すると、どの監視可能オブジェクトについても、すべての有効な属性を調べることができます。

次の例では、クライアントのネームマッピングのシナリオを示します。

Petstore の例

server1 という名前のアプリケーションサーバーインスタンスに配備された Petstore アプリケーションのメソッドに対して、何回の呼び出しがあったかについて調査します。list コマンドと get コマンドを組み合わせて使用して、該当するメソッドの統計情報にアクセスします。

1. マルチモードで CLI を呼び出します。
2. 次のように、有用な環境変数を設定して、コマンドを使用するたびに同じ変数を入力する手間を省きます。

   ```
   asadmin>export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=admin123

   asadmin>export AS_ADMIN_HOST=localhost AS_ADMIN_PORT=4848
   ```

3. 次のコマンドを入力して、インスタンス server1 の監視可能なコンポーネントを一覧表示します。

   ```
   asadmin>list --monitor server1

   出力
   iiop-service
   transaction-service
   application.CometEJB
   application.ConverterApp
   application.petstore
   http-server
   resources
   ```

   監視可能なコンポーネントの一覧には、iiop-service、http-server、transaction-service、resources、およびすべての配備済み (かつ有効な) アプリケーションが含まれます。

4. 次のコマンドを入力して、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 を使用できます。

   ```
   asadmin>list -m server1.application.petstore

   出力
   ejb-module.signon-ejb_jar
   ejb-module.catalog-ejb_jar
   ejb-module.uidgen-ejb_jar
   ```

```
ejb-module.customer-ejb_jar
ejb-module.petstore-ejb_jar
ejb-module.AsyncSenderJAR_jar
ejb-module.cart-ejb_jar
```

5. 次のコマンドを入力して、Petstore アプリケーションの EJB モジュール sigon-ejb_jar 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 server1.application.petstore.ejb-module.signon-ejb_jar
```

出力

```
entity-bean.UserEJB
stateless-session-bean.SignOnEJB
```

6. 次のコマンドを入力して、Petstore アプリケーションの EJB モジュール sigon-ejb_jar のためのエンティティ Bean UserEJB に含まれる監視可能なサブコンポーネントを一覧表示します。

```
asadmin>list -m
server1.application.petstore.ejb-module.signon-ejb_jar.entity-bean.UserEJB
```

出力

```
bean-method.create0
bean-method.findByPrimaryKey1
bean-method.remove2
bean-method.getUserName3
bean-method.setPassword4
bean-method.getPassword5
bean-method.matchPassword6
bean-method.remove7
bean-method.isIdentical8
bean-method.getEJBLocalHome9
bean-method.getPrimaryKey10
bean-pool
bean-cache
```

7. 次のコマンドを入力して、Petstore アプリケーションの EJB モジュール sigon-ejb_jar にあるエンティティ Bean UserEJB のメソッド getUserName3 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
server1.application.petstore.ejb-module.signon-ejb_jar.entity-bean.UserEJB.bean-m
ethod.getUserName3
```

出力

```
No monitorable entities for element
server1.application.petstore.ejb-module.signon-ejb_jar.entity-bean.UserEJB.bean-m
ethod.getUserName3
```

8. メソッドには監視可能なサブコンポーネントはありません。次のコマンドを入力して、メソッド getUserName3 の監視可能な統計情報をすべて取得します。

```
asadmin>get -m server1.application.petstore.ejb-module.
signon-ejb_jar.entity-bean.UserEJB.bean-method.getUserName3.*
method-name = public abstract java.lang.String
com.sun.j2ee.blueprints.signon.user.ejb.UserLocal.getUserName()
total-num-errors = 0
total-num-success = 2
execution-time-millis = 1
total-num-calls = 2
```

9. 次のコマンドを入力して、実行時間など特定の統計情報を取得することもできます。

```
asadmin>get -m server1.application.petstore.ejb-module.
signon-ejb_jar.entity-bean.UserEJB.bean-method.getUserName3.execution-time-millis
execution-time-millis = 1
```

監視可能なオブジェクトタイプ

監視に使用するオブジェクトのツリーには、複数のノードが含まれています。ノードとはオブジェクトツリー内の特定のエントリで、タイプ、名前、および親ノードによって一意に識別されます。ノードタイプには単独のものがあり、その場合は親ノードの下にノードのタイプが 1 つだけあることを意味します。名前は、単独のノードには必要ありません。

単独ではないタイプのノードには、名前が必要です。「インスタンス名」の列に、有効な名前空間を示します。

次の表に、さまざまなノードタイプ間の有効な親と子の関係についてのツリー構造と、いくつかのノードタイプについては名前空間を示します。

監視オブジェクトタイプ

ノードタイプ	単独	リーフ	子ノードのタイプ	インスタンス名
root	Yes	No	http-server iiop-service resources transaction-service application standalone-ejb-module	
http-server	Yes	No	virtual-server process	
virtual-server	Yes	Yes		
process	Yes	Yes		

監視オブジェクトタイプ (続き)

ノードタイプ	単独	リーフ	子ノードのタイプ	インスタンス名
iiop-service	Yes	Yes	orb	
orb	No	No	orb-connection orb-thread-pool	system がシステム ORB のために予約されている。すべてのユーザー ORB が TCP エンドポイントから派生する名前を取得する
orb-connection	Yes	Yes		
orb-thread-pool	Yes	Yes		
resources	Yes	No	jdbc-connection-pool	
jdbc-connection-pool	No	Yes		名前は、接続プール作成時にユーザーが指定するものと同じ
transaction-service	Yes	Yes		
application	No	No	ejb-module	server.xml に登録されたアプリケーションの名前
ejb-module	No	No	stateless-session-bean stateful-session-bean entity-bean message-driven-bean	EJB モジュールの名前。EJB JAR 名から派生する
standalone-ejb-module	No	No	stateless-session-bean stateful-session-bean entity-bean message-driven-bean	server.xml に登録されたスタンドアロン EJB モジュールの名前
stateless-session-bean	No	No	bean-pool bean-method	配備記述子の Bean 名
stateful-session-bean	No	No	bean-cache bean-method	配備記述子の Bean 名
entity-bean	No	No	bean-cache bean-pool bean-method	配備記述子の Bean 名
message-driven-bean	No	No	bean-pool bean-method	配備記述子の Bean 名
bean-pool	Yes	Yes		
bean-cache	Yes	Yes		

監視オブジェクトタイプ (続き)

ノードタイプ	単独	リーフ	子ノードのタイプ	インスタンス名
bean-method	No	Yes		メッセージ駆動型 Beans の onMessage。別のエンタープライズ Beans 内のメソッド用に、数値のサフィックスを付けたメソッド名。サフィックスは、オーバーロードされたメソッドを特定するために必要

監視可能な属性名

すべての監視可能なオブジェクトについて、監視可能な属性名を公開する必要があるわけではありません。オブジェクトには、その他のオブジェクトをグループ化するためだけに使用されるものもあります。Sun ONE Application Server では、http-server ノード以外は、ツリーのリーフノードだけが属性を持ちます。http-server ノードタイプは、子ノードと属性を持ちます。次の表では、さまざまなノードについての有効な監視可能属性名を示します。

http-server

属性名	データ型	説明
summary	文字列 (形式設定済み)	HTTP サーバーの要約。仮想サーバーとプロセスが含まれる 注 : 形式設定済みの文字列で公開されるデータについては、142 ページの「HTTP サーバーの監視可能オブジェクト」を参照してください。

virtual-server

属性名	データ型	説明
<vs-id>	文字列 (形式設定済み)	仮想サーバーの情報。各アプリケーションサーバーインスタンスには、1 つ以上の仮想サーバーを設定できる。http-server の要約属性から、仮想サーバー ID の一覧を取得できる。get コマンドパラメータに server1.http-server.virtual-server.<vs-id> という形式を指定すると、特定の仮想サーバーに関する統計情報を検索できる。get コマンドパラメータに server1.http-server.virtual-server.* という形式を指定すると、すべての仮想サーバーに関する統計情報を検索できる 注 : 形式設定済みの文字列で公開されるデータについては、142 ページの「HTTP サーバーの監視可能オブジェクト」を参照してください。

Process

属性名	データ型	説明
<pid>	文字列 (形式設定済み)	プロセスの情報。各アプリケーションサーバーインスタンスに対して、1 つのプロセスがある。http-server の要約属性から、プロセス ID を取得できる。get コマンドパラメータに server1.http-server.process.<pid> という形式を指定すると、プロセスの統計情報を取得できる 注 : 形式設定済みの文字列で公開されるデータについては、142 ページの「HTTP サーバーの監視可能オブジェクト」を参照してください。

orb-connection

属性名	データ型	説明
total-inbound-connections	整数	ORB への受信総接続数
total-outbound-connections	整数	ORB への送信総接続数

orb-thread-pool

属性名	データ型	説明
thread-pool-size	整数	ORB スレッドプールのスレッド総数
waiting-thread-count	整数	スレッドプール内で受信待機中のスレッド数

jdbc-connection-pool

属性名	データ型	説明
total-threads-waiting	整数	JDBC 接続を待機するスレッド総数
total-outbound-connections	整数	JDBC 接続検証の失敗総数
total-connections-timed-out	整数	タイムアウトになった接続要求の総数

transaction-service

属性名	データ型	説明
total-tx-completed	整数	完了したトランザクションの総数
total-tx-rolled-back	整数	ロールバックされたトランザクションの総数
total-tx-inflight	整数	処理中のトランザクション (ライブトランザクション) の総数
isFrozen	文字列	トランザクションシステムがフリーズしているかどうか (true または false)
inflight-tx	文字列 (形式設定済み)	処理中のトランザクションのリスト

bean-pool

属性名	データ型	説明
max-pool-size	整数	プール内の Bean インスタンスの最大数

bean-pool (続き)

属性名	データ型	説明
steady-pool-size	整数	プール内に通常保持される Bean インスタンス数。プールを作成すると、最初に steady-pool-size のサイズでインスタンスが保持される。プールからインスタンスが削除されると非同期で補充されるため、プールサイズは steady-pool-size に指定された値以上になる
pool-resize-quantity	整数	max-pool-size を上限とする増分、または steady-pool-size を下限とする減少分
idle-timeout-in-seconds	整数	プールクリーニングスレッドを実行する間隔を定義する。現在のサイズが通常プールサイズを超えていないかどうかをチェックし、pool-resize-quantity の要素を削除する。現在のサイズが steady-pool-size よりも小さい場合、プールサイズは min (current-pool-size+pool + resize-quantity, max-pool-size) を上限として、pool-resize-quantity だけ増加する。pool-idle-timeout-in-seconds を超過してアクセスされていないオブジェクトだけが削除対象となる
num-beans-in-pool	整数	プール内の利用可能な Beans 数
num-threads-waiting	整数	利用可能な Bean を待機しているスレッド数
total-beans-created	整数	これまでに作成された Beans 数
total-beans-destroyed	整数	これまでに削除された Beans 数
jms-max-messages-load	整数	メッセージ駆動型 Bean による処理で JMS セッションに一度に読み込む最大メッセージ数。デフォルト値は 1。メッセージ駆動型 Beans 用のプールだけに適用される

bean-cache

属性名	データ型	説明
cache-resize-quantity (resize-quantity)	整数	キャッシュ内の Beans 数が max-cache-size に等しくなったとき、つまりキャッシュでオーバーフローが発生したときに、キャッシュサイズを縮小する量
cache-misses	整数	ユーザーから要求された Bean がキャッシュ内で見つからなかった回数

bean-cache (続き)

属性名	データ型	説明
idle-timeout-in-seconds	整数	キャッシュクリーナスレッドのスケジュール間隔。このクリーナは、キャッシュ内ですべての Beans を検査し、cache-idle-timeout-in-seconds の期間アクセスされていない Beans を非活性化する
cache-hits	整数	ユーザーから要求されたエントリがキャッシュ内で見つかった回数
total-beans-in-cache	整数	キャッシュ内の Beans 数。キャッシュの現在のサイズ
max-beans-in-cache	整数	キャッシュ内に保持される Beans の最大数。この値を超えると、キャッシュのオーバーフローが発生する
num-passivations	整数	非活性化の数。ステートフルセッション Beans だけに適用される
num-passivation-errors	整数	非活性化中に発生したエラーの回数。ステートフルセッション Beans だけに適用される
num-expired-sessions-removed	整数	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数。ステートフルセッション Beans だけに適用される
num-passivation-success	整数	非活性化の成功回数。ステートフルセッション Beans だけに適用される

bean-method

属性名	データ型	説明
method-name	文字列	完全指定のメソッド名
total-num-calls	整数	メソッドが呼び出された回数。EJB コンテナに対する監視機能が true に設定されている場合は、ステートレスセッション Beans、ステートフルセッション Beans、およびエンティティ Beans について収集される。メッセージ駆動型 Bean コンテナに対する監視機能が有効に設定されている場合は、メッセージ駆動型 Beans について収集される
total-num-errors	整数	例外が発生したメソッドの実行回数。EJB 設定で監視機能が有効に設定されている場合は、ステートレスセッション Beans、ステートフルセッション Beans、およびエンティティ Beans について収集される。MDB 設定で監視機能が有効に設定されている場合は、メッセージ駆動型 Beans について収集される

bean-method (続き)

属性名	データ型	説明
total-num-success	整数	メソッドの実行が成功した回数。EJB コンテナに対する監視機能が true に設定されている場合は、ステートレスセッション Beans、ステートフルセッション Beans、およびエンティティ Beans について収集される。メッセージ駆動型 Bean コンテナに対する監視機能が有効に設定されている場合は、メッセージ駆動型 Beans について収集される
execution-time-millis	Long	最後に成功したメソッドの実行に要した時間。EJB コンテナに対する監視機能が有効に設定されている場合は、ステートレスセッション Beans、ステートフルセッション Beans、およびエンティティ Beans について収集される。メッセージ駆動型 Bean コンテナに対する監視機能が有効に設定されている場合は、メッセージ駆動型 Beans について収集される

HTTP サーバーの監視可能オブジェクト

HTTP サーバーの監視可能属性名 summary は、Server 要素の属性値とそのサブ要素に関する要約情報を出力します。要約情報には、各サブ要素の数や、各サブ要素の属性値も含まれます。HTTP サーバーの virtual-server 属性は、VirtualServer 要素の属性とその各サブ要素の詳細を出力します。process 属性は、Process 要素の属性値とその各サブ要素の詳細を出力します。

NSAPI パフォーマンスプロファイルを有効にして、Profile 要素と ProfileBucket 要素の統計情報を取得するには、『Sun ONE Application Server Developer's Guide to NSAPI』を参照してください。

パフォーマンスチューニングに監視統計情報を使用する方法については、『Sun ONE Application Server パフォーマンスおよびチューニングガイド』を参照してください。

監視可能な HTTP サーバー要素

次の表に、HTTP サーバーの監視可能な要素を示します。

監視可能な HTTP サーバー要素

要素名	サブ要素	説明
Server	ConnectionQueue ThreadPool Profile Process VirtualServer	サーバーインスタンス

監視可能な HTTP サーバー要素 (続き)

要素名	サブ要素	説明
ConnectionQueue	なし	要求が送信される前に保持されるキュー。Sun ONE Application Server 7 には接続キューが 1 個だけある
ThreadPool	なし	スレッドプール。init.conf ファイルで定義する
Profile	なし	NSAPI パフォーマンスプロファイルバケット
Process	ConnectionQueueBucket ThreadPoolBucket DnsBucket DnsBucket KeepaliveBucket CacheBucket Thread	サーバーインスタンス内の単一サーバープロセス
ConnectionQueueBucket	なし	特定の ConnectionQueue に属する統計情報を追跡する
ThreadPoolBucket	ThreadPoolBucket	特定の ThreadPool に属する統計情報を追跡する
DnsBucket	なし	DNS 統計情報を追跡する
KeepaliveBucket	なし	キープアライブ (持続的接続) の統計情報
CacheBucket	なし	ファイルキャッシュ (NSFC) の統計情報を追跡する
Thread	RequestBucket ProfileBucket	スレッドを処理する要求を説明する
VirtualServer	RequestBucket ProfileBucket	仮想サーバーを説明する
RequestBucket	なし	要求に関連する統計情報を追跡する
ProfileBucket		Profile に属する統計情報を追跡する

監視可能な HTTP サーバー属性

次の表に、HTTP サーバーの監視可能な属性を示します。

Server

属性名	値	説明
`Id`		サーバーインスタンス ID (`server1` など)
`VersionServer`		Sun ONE Application Server のバージョンを示す文字列
`TimeStarted`	GMT	サーバーインスタンスの起動時刻
`SecondsRunning`		サーバーインスタンスが起動されてからの秒数
`TicksPerSecond`		1 秒あたりのタイマー刻み数。この値はシステムに依存する
`MaxProcs`		プロセスの最大数
`MaxThreads`		処理中のスレッドの最大数
`MaxVirtualServers`		追跡される仮想サーバーの最大数
`FlagProfilingEnabled`	0 (off)、1 (on)	NSAPI パフォーマンスプロファイルが有効 (on) であるかどうかを示す
`FlagVirtualServerOverflow`	0 (no)、1 (yes)	`MaxVirtualServers` を超える仮想サーバーを設定する (yes) かどうかを示す。この属性を 1 に設定しても、統計情報がすべての仮想サーバーについて追跡されるわけではない
`LoadMinuteAverage`		1 分間の平均読み込み
`Load5MinuteAverage`		5 分間の平均読み込み
`Load15MinuteAverage`		15 分間の平均読み込み
`RateBytesTransmitted`	1 秒あたりのバイト数	サーバー定義の間隔で送信されるデータの速度。この情報を利用できない場合は 0
`RateBytesReceived`	1 秒あたりのバイト数	サーバー定義の間隔で受信されるデータの速度。この情報を利用できない場合は 0

ConnectionQueue

属性名	値	説明
`Id`		接続キュー ID

ThreadPool

属性名	値	説明
Id		スレッドプール ID
属性名		スレッドプールのシンボリック名

Profile

属性名	値	説明
Id		NSAPI パフォーマンスプロファイルバケット ID
属性名		NSAPI パフォーマンスプロファイルバケットのシンボリック名
説明		NSAPI パフォーマンスプロファイルバケットの説明

Process

属性名	値	説明
Pid		プロセスを一意に特定するオペレーティングシステムのプロセス識別子
Mode	unknown active	プロセスがアクティブの場合は active が表示される
TimeStarted	GMT	プロセスの起動時刻
CountConfigurations		設定が読み込まれた回数。この情報が利用できない場合は 0
SizeVirtual	K バイト	プロセスに使用された仮想メモリのサイズ
SizeResident	K バイト	プロセスに使用された常駐メモリのサイズ
FractionSystemMemoryUsage		プロセスに使用されたシステムメモリの部分

ConnectionQueueBucket

属性名	値	説明
ConnectionQueue		ConnectionQueue 要素の ID
CountTotalConnection		これまでに受け入れた新しい接続の総数
CountQueued		現在キューに入れられている接続数
PeakQueued		同時にキューに入れられた最大接続数
MaxQueued		キューに入れることのできる最大接続数
CountOverflow		接続によってキューがいっぱいになった回数
CountTotalQueued		キューに入れられた接続の総数。1 つの接続が複数回キューに入れられることもあるため、CountTotalQueued は CountTotalConnections 以上の値になる
TicksTotalQueued		接続がキューに入れられていたタイマー刻みの合計。タイマー刻みはシステムに依存する時間単位。TicksPerSecond を参照

ThreadPoolBucket

属性名	値	説明
Thread-pool		ThreadPool 要素の ID
CountThreadsIdle		現在アイドル状態の要求処理スレッド数
CountThreads		要求処理スレッド数
MaxThreads		同時に存在できる要求処理スレッドの最大数
CountQueued		スレッドプールで処理するためにキューに入れられた要求の数
PeakQueued		同時にキューに入れられた最大要求数
MaxQueued		キューに入れることのできる最大要求数

DnsBucket

属性名	値	説明
FlagCacheEnabled	0 (off)、1 (on)	DNS キャッシュが有効 (on) であるかどうかを示す
CountCacheEntries		現在キャッシュ内にある DNS エントリ数
MaxCacheEntries		キャッシュに収容できる DNS エントリの最大数
CountCacheHits		DNS キャッシュの検索に成功した回数
CountCacheMisses		DNS キャッシュの検索に失敗した回数
FlagAsyncEnabled	0 (off)、1 (on)	非同期 DNS 検索が有効 (on) であるかどうかを示す
CountAsyncNameLookups		非同期 DNS 名検索が実行された合計回数
CountAsyncAddrLookups		非同期 DNS アドレス検索が実行された合計回数
CountAsyncLookupsInProgress		現在実行中の非同期 DNS 検索の合計回数

KeepaliveBucket

属性名	値	説明
CountConnections		現在キープアライブモードにある接続の数
MaxConnections		同時にキープアライブとなる接続の最大数
CountHits		キープアライブモードの接続が有効な要求を作成した合計回数
CountFlushes		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
CountTimeouts		キープアライブ接続がタイムアウトになった回数
SecondsTimeouts		サーバーがアイドル状態のキープアライブ接続を閉じるまでの秒数
CountRefusals		キープアライブ接続がサーバーによって拒否された回数

CacheBucket

属性名	値	説明
FlagEnabled	0 (off)、1 (on)	ファイルキャッシュが有効 (on) であるかどうかを示す
SecondsMaxAge	秒数	ファイルキャッシュエントリの最大有効期間
CountEntries		現在ファイルキャッシュ内にあるエントリ数
MaxEntries		同時にファイルキャッシュに収容できるキャッシュエントリの最大数
CountOpenEntries		開かれているファイルに関連付けられたエントリの数
MaxOpenEntries		同時にファイルキャッシュに収容できる開かれたファイルに関連付けられたキャッシュエントリの最大数
SizeHeapCache	バイト数	キャッシュされたファイルコンテンツに使用されるヒープの量
MaxHeapCacheSize	バイト数	キャッシュされたファイルコンテンツのためにファイルキャッシュが使用するヒープの最大量
SizeMmapCache	バイト数	メモリにマップされたファイルコンテンツに使用されるアドレス空間の量
MaxMmapCacheSize	バイト数	メモリにマップされたファイルコンテンツのためにファイルキャッシュが使用するアドレス空間の最大量
CountHits		キャッシュエントリの検索に成功した回数
CountMisses		キャッシュエントリの検索に失敗した回数
CountInfoHits		ファイル情報の検索に成功した回数
CountInfoMisses		ファイル情報の検索に失敗した回数
CountContentHits		コンテンツの検索に成功した回数
CountContentMisses		コンテンツの検索に失敗した回数

Thread

属性名	値	説明
Mode	unknown、idle、DNS、request、processing、response、updating	最後に検知されたスレッドの状態
TimeStarted	GMT	スレッドの起動時刻
ConnectionQueue		スレッドが処理している ConnectionQueue の ID

VirtualServer

属性名	値	説明
Id		仮想サーバー ID
Mode	unknown、active	仮想サーバーがアクティブの場合は active が表示される
Hosts		仮想サーバーが配信するソフトウェア仮想サーバーのホスト名 (www.foo.com foo.com foo.isp.com など)
Interfaces		仮想サーバーを設定したインタフェース (リスナー)。192.168.1.2:80 192.168.1.2:443 など

RequestBucket

属性名	値	説明
CountRequests		処理された要求数
CountBytesReceived		受信バイト数。この情報が利用できない場合は 0
CountBytesTransmitted		送信バイト数。この情報が利用できない場合は 0
RateBytesTransmitted	1 秒あたりのバイト数	サーバー定義の間隔で送信されたデータの速度。この情報を利用できない場合は 0
MaxByteTransmissionRate		サーバー定義の間隔で送信されたデータの最大速度。この情報を利用できない場合は 0
CountOpenConnections		開かれている接続数。この情報が利用できない場合は 0
MaxOpenConnections		開かれる最大接続数。この情報が利用できない場合は 0
Count2xx		送信された 200 レベルの応答数

RequestBucket (続き)

属性名	値	説明
Count3xx		送信された 300 レベルの応答数
Count4xx		送信された 400 レベルの応答数
Count5xx		送信された 500 レベルの応答数
CountOther		送信された 200、300、400、および 500 以外のレベルの応答数
Count200		送信された 200 レベルの応答数
Count302		送信された 302 レベルの応答数
Count304		送信された 304 レベルの応答数
Count400		送信された 400 レベルの応答数
Count401		送信された 401 レベルの応答数
Count403		送信された 403 レベルの応答数
Count404		送信された 404 レベルの応答数
Count503		送信された 503 レベルの応答数

ProfileBucket

属性名	値	説明
Profile		Profile 要素の ID
Countcalls		NSAPI SAF の呼び出し数
CountRequests		処理された要求数
TicksDispatch		要求のディスパッチに費やされたタイマー刻み数。タイマー刻みはシステムに依存する時間単位。TicksPerSecond を参照
TicksFunction		NSAPI SAF に費やされたタイマー刻み数。タイマー刻みはシステムに依存する時間単位。TicksPerSecond を参照

CLI によるトランザクションサービスの管理

set コマンドを使用すると、監視対象とする JTS の統計情報を管理できます。

例 1

ロールバックリストにトランザクションを追加するには (その結果ロールバックトランザクションまたは指定のトランザクションになる)、次のように set コマンドを実行します。

```
set --monitor server1.transaction-service.rollback-list=txnid1
```

例 2

トランザクションサービスを凍結するには、次のように set コマンドを実行します。

```
set --monitor server1.transaction-service.freeze=true
```

JTS の統計情報を収集する場合に監視できる属性については、表「トランザクションサービス」を参照してください。これらの属性は、132 ページの「CLI ネームマッピング」で説明した規則に従ってコマンド行から設定できます。

Java トランザクションサービスの詳細については、第 9 章「トランザクションサービスの使用」を参照してください。

HTTP サービス品質の使用

トラフィックのカウント方法と帯域幅の再計算頻度は、次の設定によって管理します。

- 再計算間隔 - 帯域幅を計算する頻度 (ミリ秒単位)
- メトリック間隔 - トラフィック計算でデータを使用する時間

管理インタフェースでは、サーバーインスタンスまたは仮想サーバーのクラスに対する、サーバーレベルまたはクラスレベルの設定を有効にすることができます。ただし、個々の仮想サーバーごとに設定をオーバーライドすることもできます。

この節には次の項目があります。

- サービス品質 (QOS) の例
- サービス品質 (QOS) の設定
- obj.conf ファイルへの必要な変更
- サービス品質に関する既知の制限事項

サービス品質 (QOS) の例

次の例では、サービス品質の情報を収集および計算する方法を示します。

- サーバーのメトリック間隔は 30 秒
- サーバーは 0 秒で起動する
- 1 秒の時点で、HTTP 接続によって、サーバーとの間に 5000 バイトのトラフィックが生成される
- このあと、それ以上の接続は行われない。30 秒の時点で、最後の 30 秒間の合計トラフィックは 5000 バイト
- 32 秒の時点で、メトリック間隔の 30 秒よりも古いトラフィックである 1 秒の時点で生成されたトラフィックが廃棄される。その結果、この時点での最後の 30 秒間の合計トラフィックは 0

再計算間隔も同様に機能します。このサーバーの再計算間隔は 100 ミリ秒です。

前の例に引き続き、帯域幅は 100 ミリ秒ごとに再計算されます。この計算は、トラフィック量とメトリック間隔に基づいて行われます。

- 0 秒の時点では、帯域幅の 1 回目の計算が行われる。この時点での合計トラフィックは 0 で、測定時間の 30 秒で割ると、帯域幅は 0
- 1 秒の時点では、帯域幅の 10 回目 (1000 ミリ秒 /100 ミリ秒) の計算が行われる。この時点での合計トラフィックは 5000 バイト。これを 30 秒で割ると、帯域幅は 5000/30 = 166 バイト / 秒
- 30 秒の時点では、帯域幅の 300 回目の計算が行われる。この時点での合計トラフィックは 5000 バイト。これを 30 秒で割ると、帯域幅は 5000/30 = 166 バイト / 秒
- 32 秒の時点では、帯域幅の 320 回目の計算が行われる。この時点でのトラフィックは 0 バイト (トラフィックを生成した接続が古くなってカウントされなくなるため)。これを 30 秒で割ると、帯域幅は 0 バイト / 秒

サービス品質 (QOS) の設定

サーバーインスタンスまたは仮想サーバーのクラスについてのサービス品質は、管理インタフェースで設定されます。

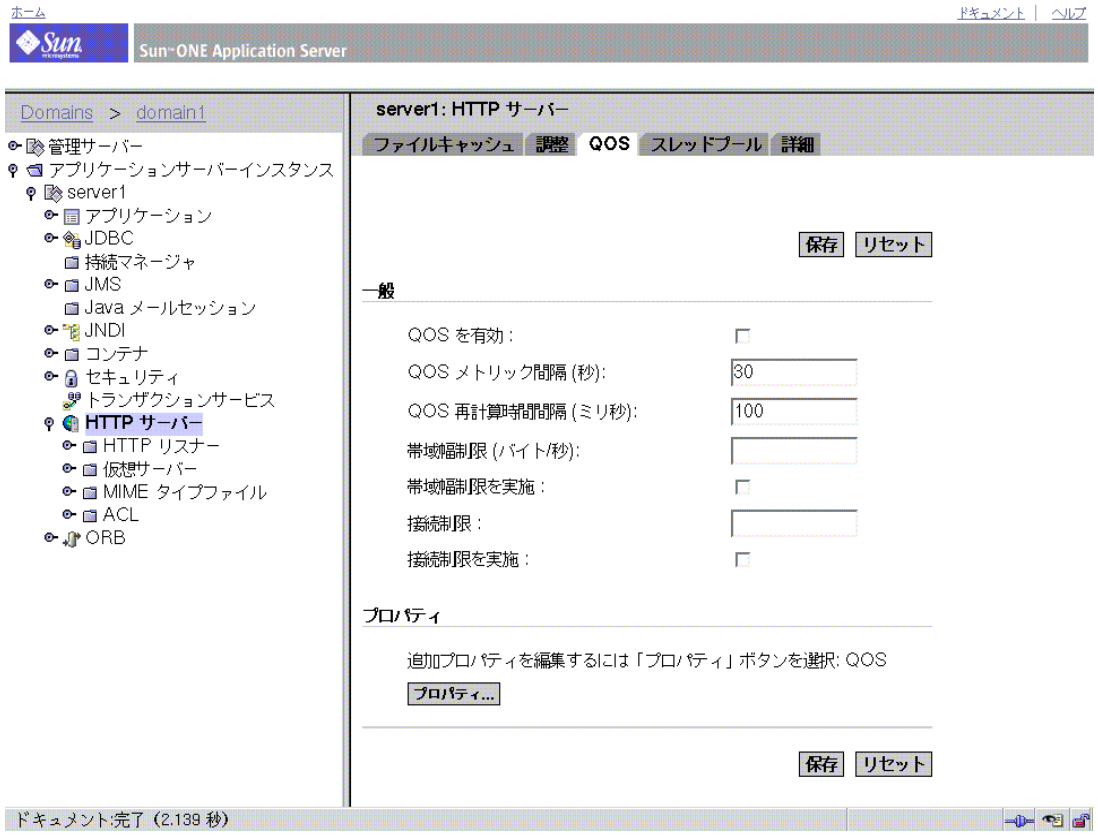
注 サービス品質の設定内容を有効にするためには、`obj.conf` ファイルの Server Application Function (SAF) も設定する必要があります。155 ページの「obj.conf ファイルへの必要な変更」を参照してください。

サービス品質を設定するには、次の手順に従います。

1. 左のペインで「アプリケーションサーバーインスタンス」ノードを選択します。
2. サーバーインスタンスノードを展開して、「HTTP サーバー」ノードを表示します。
3. 「HTTP サーバー」ノードをクリックして、「QOS」タブを表示します。
4. 「QOS」タブをクリックします。

   次のページでは、サービス品質の一般的な設定と、「プロパティ」ボタンが表示されています。

仮想サーバーインスタンスの「QOS」タブ



5. この HTTP サーバーに対するサービス品質を有効にするために、「QOS を有効」をクリックします。

   注 : デフォルトでは、サービス品質は無効になっています。サービス品質を有効にすると、サーバーのオーバーヘッドがわずかに増えます。

6. 「QOS メトリック間隔」を指定します。

   メトリック間隔は、サーバートラフィック計算中にデータがサンプリングされる時間 (秒単位) です。デフォルト値は 30 秒です。

   サイズの大きいファイルを転送することが多い場合は、このフィールドの値を大きくします (数分またはそれ以上)。サイズの大きいファイルを転送する際、メトリック間隔が短いと、許容帯域幅がすべて占有される可能性があります。この場合、最大帯域幅の設定が有効になっていると接続が拒否されます。帯域幅はメトリック間隔によって平均化されるため、間隔を長くすれば、サイズの大きいファイルによるトラフィックスパイクを防ぐことができます。

   帯域幅の制限値が使用可能な帯域幅よりもはるかに小さい場合 (たとえば、帯域幅の制限値が 1M バイト / 秒で、バックボーンとの接続が 1G バイト / 秒の場合) は、メトリック間隔を短くする必要があります。

   注 : 転送する静的ファイルのサイズが大きいという問題の解決策と、帯域幅の制限値が使用可能な帯域幅よりもはるかに小さいという問題の解決策は相反しています。

7. どちらの問題を調整するかを決定する必要があります。

   「QOS 再計算時間間隔」を指定します。再計算時間間隔は、すべてのサーバー、クラス、および仮想サーバーの帯域幅の計算間隔を示すミリ秒数です。デフォルトは 100 ミリ秒です。

8. 「帯域幅制限」を指定します。

   これは、サーバーインスタンスに対する最大帯域幅 (バイト / 秒) です。ある程度「QOS メトリック間隔」と依存関係があります。

9. 最大帯域幅の設定を強制するかどうかを選択します。

   最大帯域幅を強制する場合は、帯域幅の制限値に達したとき、それ以上の接続が拒否されます。

   最大帯域幅を強制しない場合は、制限値を超えたとき、サーバーのイベントログにメッセージが記録されます。

10. 「接続制限」を指定します。

    これは、同時に処理できる要求の数です。

11. 接続制限の設定を強制するかどうかを選択します。

    最大接続数を強制する場合は、制限値に達したとき、それ以降の接続が拒否されます。最大接続数を強制しない場合は、制限値を超えたとき、サーバーのイベントログにメッセージが記録されます。

12. この指定はオプションです。サービス品質に関する追加の名前 - 値ペアを指定するには、「プロパティ」ボタンをクリックします。

    サービス品質のプロパティで有効な名前 - 値ペアの一覧については、オンラインヘルプを参照してください。

13. 「保存」をクリックして、サーバーインスタンスへの変更をコミットします。

14. 左ペインでアプリケーションサーバーインスタンスを選択してサーバーインスタンスにアクセスし、「変更を適用」をクリックします。

obj.conf ファイルへの必要な変更

サービス品質を強制するには、指令 (ディレクティブ) を `obj.conf` ファイルに追加して、次の Server Application Function (SAF) を呼び出す必要があります。

- AuthTrans `qos-handler`
- Error `qos-error`

AuthTrans 指令 `qos-handler` を正しく動作させるためには、デフォルトのオブジェクト内で最初の `AuthTrans` として設定する必要があります。サービス品質ハンドラには、仮想サーバー、仮想サーバークラス、グローバルサーバーの現在の統計情報を調べ、エラーを返して制限値を強制する働きがあります。Sun ONE Application Server には、`qos-handler` という組み込みのサービス品質ハンドラ SAF のサンプルが付属しています。この SAF は、制限値に達した時刻を記録したあと、サーバーに 503 `Server busy` エラーを返して、NSAPI で処理されるようにします。

Sun ONE Application Server には、`qos-error` という組み込みのエラー SAF のサンプルも付属しています。このサンプルは、503 エラーの原因となった制限値と、その制限値を決定付けた統計値を示すエラーページを返します。

SAF とその使用方法の詳細については、『Sun ONE Application Server Developer's Guide to NSAPI』を参照してください。

サービス品質に関する既知の制限事項

サービス品質の機能を使用するときは、次の制限事項に留意してください。

- サービス品質の機能では、アプリケーションレベルの HTTP 帯域幅だけが測定されます。HTTP 帯域幅は、次のようなさまざまな理由により、実際の TCP ネットワーク帯域幅とは異なる場合があります。
  - SSL が有効になっている場合、トラフィックにハンドシェークとクライアント証明書の交換が追加されますが、これらは計測されません。
  - 単方向または双方向のチャンクエンコーディングが有効になっている場合、チャンク層によってチャンクヘッダーが削除され、トラフィックにカウントされません。その他のヘッダーやプロトコル項目はカウントされます。
- サービス品質の機能では、`PR_TransmitFile` 呼び出しからのトラフィックを正確に測定できません。`PR_Send()/net_write`、`PR_Recv()/net_read` などの基本入出力操作では、通常、1 回のシステムコールで転送されるバイト数とバッファのサイズが等しくなり、呼び出しの時間も短いため、転送されたデータ量を帯域幅マネージャで迅速に計算できます。この機能は、動的なコンテンツアプリケーションの瞬間的な帯域幅の測定に最適です。しかし、`PR_TransmitFile` から転送されるデータの量は転送が終了するまで測定できません。

  サービス品質の機能は、`PR_TransmitFile` が短時間であれば適切に動作します。一方、`PR_TransmitFile` が長時間に及ぶ場合、たとえばダイアルアップユーザーがサイズの大きいファイルをダウンロードする場合などは、転送の完了時に転送された全体のデータ量が算入されます。次の再計算間隔では、その大規模な `PR_TransmitFile` が原因で、帯域幅マネージャによって計算される帯域幅の値が非常に大きくなります。このような場合、サーバーは、次のメトリック間隔まですべての要求を拒否することがあります。そして、帯域幅マネージャが期限切れになったファイル転送操作を終了したときに、ふたたび帯域幅の値が小さくなります。静的ファイルを長時間かけてダウンロードするようなサイトでは、メトリック間隔をデフォルトの 30 秒よりも長くする必要があります。
- 計算される帯域幅は、瞬間的なものではなく、一定の間隔で一定期間にわたって計算されるものなので、常に近似値になります。たとえば、メトリック間隔がデフォルトの 30 秒で、サーバーが 29 秒間アイドル状態だったとします。この場合、次の 1 秒間で、クライアントが帯域幅の制限値の 30 倍を使用することもあります。
- 帯域幅のサービス品質に関する統計情報は、サーバーが動的に再設定されると失われます。また、サービス品質の制限は、アクティブでない古い設定上で接続したスレッドには適用されません。これは、帯域幅マネージャのスレッドが、アクティブな設定の帯域幅の統計情報だけを計算するためです。クライアントが長時間ソケットを終了せず、ずっとアクティブになっている場合、サーバーはこのクライアントをタイムアウトにしません。このようなクライアントは、サーバーを動的に再設定しても、サービス品質の制限の影響を受けない場合があります。

- 同時に複数の接続が存在する場合、統計情報は、仮想サーバークラスやグローバルサーバーインスタンスとは異なった細分度で計算されます。個々の仮想サーバーの接続カウンタは、要求が解析されて仮想サーバーに配信された直後、アトミックに増分されます。そして、その要求の応答処理が終了した時点でアトミックに減分されます。このため、仮想サーバーの接続に関する統計情報は、どの時点でも常に正確です。

  ただし、仮想サーバークラスとグローバルサーバーインスタンスの接続に関する統計情報は、すぐには更新されません。これらの統計情報は、再計算間隔ごとに帯域幅マネージャのスレッドによって更新されます。仮想サーバークラスの接続数は、そのクラスのすべての仮想サーバー上の接続の合計数であり、グローバルサーバーインスタンスの接続数は、すべての仮想サーバークラス上の接続の合計数です。

  これらの値の計算方法によって、仮想サーバーの接続数は常に正確になります。また、接続数の制限を強制すると、制限を超えた接続は許可されません。仮想サーバークラスとサーバーインスタンスの値は、再計算間隔だけで計算されるため、接続数に比較すると正確ではありません。

SNMP について

SNMP (Simple Network Management Protocol) は、ネットワークの管理情報と監視情報を交換するために使用されるプロトコルです。管理対象デバイスとネットワークマネージメントステーション (NMS) 間のデータのやりとりは、SNMP によって行われます。ネットワーク上のホスト、ルーター、HTTP サーバー、その他のサーバーなど、SNMP を実行するすべてのデバイスが管理対象デバイスとなります。

この節では次の項目について説明します。

- ネットワーク管理ステーション (NMS)
- 管理情報ベース (MIB) オブジェクト
- SNMP メッセージ
- SNMP トラップの送信先
- SNMP エージェントコミュニティ

ネットワーク管理ステーション (NMS)

NMS (ネットワーク管理ステーション) は、特定のネットワークをリモート管理するマシンです。通常、NMS ソフトウェアには、収集されたデータをグラフに表示する機能や、そのデータを使ってサーバーが特定の許容範囲内で動作していることを確認する機能があります。

通常、NMS には強力なワークステーションを使用し、1 つ以上のネットワーク管理アプリケーションがインストールされます。HP OpenView のようなネットワーク管理アプリケーションでは、HTTP サーバーをはじめとする管理対象のデバイスに関する情報がグラフィカルに表示されます。たとえば、社内のどのサーバーが稼動または停止しているかを表示することや、受け取ったエラーメッセージの数と種類を表示することができます。このような情報は、Sun ONE Application Server で SNMP を使用する場合、サブエージェントとマスターエージェントという 2 種類のエージェントを使用して、NMS とサーバーの間で転送されます。

サブエージェントは、さまざまなドメインで実行しているサーバーインスタンスに関する情報を収集し、マスターエージェントに情報を渡します。マスターエージェントとサブエージェントは、Sun ONE Application Server のインストールごとに存在します。

注	SNMP の設定を変更したときは、「適用」ボタンをクリックしてから、SNMP サブエージェントを再起動する必要があります。

マスターエージェントは、さまざまなサブエージェントと NMS との間で情報を交換します。マスターエージェントは、Sun ONE Application Server のインストール時にインストールされます。

1 台のホストコンピュータに複数のサブエージェントをインストールできますが、マスターエージェントは 1 つしかインストールできません。たとえば、Sun ONE Directory Server、Sun ONE Application Server、Sun ONE Messaging Server を同じホストにインストールしている場合、各サーバーのサブエージェントは、同じマスターエージェントと通信します。

NMS は、サーバー情報の要求、またはサーバー MIB に保存されている変数値の変更のいずれかを行います。次に例を示します。

1. NMS が管理サーバーのマスターエージェントにメッセージを送信します。このメッセージは、データの要求 (GET メッセージ) か MIB の変数の設定命令 (SET メッセージ) です。
2. マスターエージェントは、受信したメッセージを適切なサブエージェントに転送します。
3. サブエージェントは、このデータを取得するか、MIB 内の変数を変更します。

4. サブエージェントは、マスターエージェントにデータまたは状態を報告します。マスターエージェントは、報告内容 (GET メッセージ) を NMS に返送します。
5. NMS は、ネットワーク管理アプリケーションを通して、データをテキストまたはグラフィックで表示します。

管理情報ベース (MIB) オブジェクト

Sun ONE Application Server には、ネットワーク上の管理情報や監視情報に関する変数が保存されています。マスターエージェントがアクセスできる変数は、管理対象オブジェクトと呼ばれます。これらのオブジェクトは、MIB (管理情報ベース) と呼ばれるツリー構造で定義されます。MIB によって、HTTP サーバーのネットワーク設定、状態、および統計情報へアクセスできます。SNMP を使用すると、これらの情報を NMS (ネットワーク管理ステーション) から確認できます。

MIB ツリーのトップレベルを見ると、インターネットオブジェクト識別子には次の 4 種類のサブツリーがあることがわかります。

- directory (1)
- mgmt (2)
- experimental (3)
- private (4)

private (4) のサブツリーには、enterprises (1) ノードが含まれます。enterprises (1) ノードの各サブツリーは、個々の企業 (独自の MIB 拡張を登録している組織) に割り当てられます。企業は、自社のサブツリーの下に製品固有のサブツリーを作成できます。企業によって作成された MIB は、enterprises (1) ノードの下に置かれます。

Sun ONE Application Server のサブエージェントは、SNMP 通信で使用する MIB を提供します。サーバーは、これらの変数が含まれたメッセージまたはトラップを送信することにより、重大なイベントを NMS に報告します。NMS はサーバーの MIB のデータをクエリできます。

Sun ONE Application Server ごとに独自の MIB が、*install_dir*/lib に格納されています。

Sun ONE Application Server の MIB は、appserv.mib というファイルです。この MIB には、Sun ONE Application Server のネットワーク管理に関する各種変数の定義が格納されています。

Sun ONE Application Server の MIB は、appserver 1 (as appserver7 OBJECT IDENTIFIER ::= {appserver 1 }) というオブジェクト識別子を持ち、*install_dir*/lib ディレクトリに格納されます。

Sun ONE Application Server の MIB を使用すると、Sun ONE Application Server に関する管理情報をリアルタイムで確認および監視できます。次の表に、appserv.mib ファイルに格納されている管理対象オブジェクトとその説明を示します。

appserv.mib の管理対象オブジェクトと説明

管理対象オブジェクト	説明
iwsCpuID	CPU 識別子
iwsCpuIdleTime	CPU のアイドル時間
iwsCpuKernelTime	CPU のカーネル時間
iwsCpuTable	Sun ONE Application Server の CPU
iwsCpuUserTime	CPU のユーザー時間
iwsInstanceTable	Sun ONE Application Server のインスタンス
iwsInstanceId	サーバーインスタンス識別子
iwsInstanceVersion	文字列 (SunONE-ApplicationServer-Enterprise/7 BB1-01/24/2001 17:15 (SunOS DOMESTIC) など)
iwsInstanceDescription	サーバーインスタンスの説明
iwsInstanceOrganization	サーバーインスタンスを管理する組織
iwsInstanceContact	サーバーインスタンスを管理する担当者の連絡先
iwsInstanceLocation	サーバーの場所
iwsInstanceStatus	サーバーインスタンスの状態
iwsInstanceUptime	サーバーの稼動時間
iwsInstanceDeathCount	サーバーインスタンスのプロセスが停止した回数
iwsInstanceRequests	サーバーインスタンスが処理した要求の数
iwsInstanceInOctets	サーバーインスタンスが受信したオクテット数。使用できる情報がない場合は 0
iwsInstanceOutOctets	サーバーインスタンスが送信したオクテット数。使用できる情報がない場合は 0
iwsInstanceCount2xx	サーバーインスタンスが発行した 200 番レベル (Successful) の応答数
iwsInstanceCount3xx	サーバーインスタンスが発行した 300 番レベル (Redirection) の応答数
iwsInstanceCount4xx	サーバーインスタンスが発行した 400 番レベル (Client Error) の応答数

appserv.mib の管理対象オブジェクトと説明 (続き)

管理対象オブジェクト	説明
iwsInstanceCount5xx	サーバーインスタンスが発行した 500 番レベル (Server Error) の応答数
iwsInstanceCountOther	サーバーインスタンスが発行したその他 (2xx、3xx、4xx、5xx 以外) の応答数
iwsInstanceCount200	サーバーインスタンスが発行した 200 (OK) の応答数
iwsInstanceCount302	サーバーインスタンスが発行した 302 (Moved Temporarily) の応答数
iwsInstanceCount304	サーバーインスタンスが発行した 304 (Not Modified) の応答数
iwsInstanceCount400	サーバーインスタンスが発行した 400 (Bad Request) の応答数
iwsInstanceCount401	サーバーインスタンスが発行した 401 (Unauthorized) の応答数
iwsInstanceCount403	サーバーインスタンスが発行した 403 (Forbidden) の応答数
iwsInstanceCount404	サーバーインスタンスが発行した 404 (Not Found) の応答数
iwsInstanceLoad1MinuteAverage	サーバーインスタンスを実行しているシステムの 1 分間の平均読み込み
iwsInstanceLoad5MinuteAverage	サーバーインスタンスを実行しているシステムの 5 分間の平均読み込み
iwsInstanceLoad15MinuteAverage	サーバーインスタンスを実行しているシステムの 15 分間の平均読み込み
iwsInstanceNetworkInOctets	ネットワーク上で送信された 1 秒間のオクテット数
iwsInstanceNetworkOutOctets	ネットワーク上で受信された 1 秒間のオクテット数
iwsVsTable	仮想サーバー
iwsVsId	仮想サーバー識別子
iwsVsRequests	仮想サーバーが処理した要求の数
iwsVsInOctets	仮想サーバーが受信したオクテット数
iwsVsOutOctets	仮想サーバーが送信したオクテット数
iwsVsCount2xx	仮想サーバーが発行した 200 番レベル (Successful) の応答数

appserv.mib の管理対象オブジェクトと説明 (続き)

管理対象オブジェクト	説明
iwsVsCount3xx	仮想サーバーが発行した 300 番レベル (Redirection) の応答数
iwsVsCount4xx	仮想サーバーが発行した 400 番レベル (Client Error) の応答数
iwsVsCount5xx	仮想サーバーが発行した 500 番レベル (Server Error) の応答数
iwsVsCountOther	仮想サーバーが発行したその他 (2xx、3xx、4xx、5xx 以外) の応答数
iwsVsCount200	仮想サーバーが発行した 200 (OK) の応答数
iwsVsCount302	仮想サーバーが発行した 302 (Moved Temporarily) の応答数
iwsVsCount304	仮想サーバーが発行した 304 (Not Modified) の応答数
iwsVsCount400	仮想サーバーが発行した 400 (Bad Request) の応答数
iwsVsCount401	仮想サーバーが発行した 401 (Unauthorized) の応答数
iwsVsCount403	仮想サーバーが発行した 403 (Forbidden) の応答数
iwsVsCount404	仮想サーバーが発行した 404 (Not Found) の応答数
iwsProcessTable	Sun ONE Application Server のプロセス
iwsProcessId	オペレーティングシステムのプロセス識別子
iwsProcessThreadCount	要求処理スレッド数
iwsProcessThreadIdle	現在アイドル状態の要求処理スレッド数
iwsProcessConnectionQueueCount	接続キュー内の接続数
iwsProcessConnectionQueuePeak	これまでに同時にキューに入れた最大接続数
iwsProcessConnectionQueueMax	接続キューに入れることができる最大接続数
iwsProcessConnectionQueueTotal	これまでに受け入れた接続数
iwsProcessConnectionQueueOverflows	接続キューのオーバーフローによって拒否された接続数
iwsProcessKeepaliveCount	キープアライブキュー内の接続数
iwsProcessKeepaliveMax	キープアライブキューに入れることができる最大接続数
iwsProcessSizeVirtual	K バイト単位のプロセスサイズ
iwsProcessSizeResident	K バイト単位のプロセス常駐サイズ
iwsProcessFractionSystemMemoryUsage	システムメモリ内のプロセスメモリ部分

appserv.mib の管理対象オブジェクトと説明 (続き)

管理対象オブジェクト	説明
iwsListenTable	Sun ONE Application Server 待機ソケット
iwsListenId	待機ソケット識別子
iwsListenAddress	ソケットが待機するアドレス
iwsListenPort	ソケットが待機するポート
iwsListenSecurity	暗号化のサポート
iwsThreadPoolCount	拒否された要求の数
iwsThreadPoolMax	キューに入れることができる最大要求数
iwsThreadPoolPeak	これまでに同時にキューに入れた最大の要求数
iwsThreadPoolTable	Sun ONE Application Server のスレッドプール
iwsVsCount503	発行された 503 (Unavailable) の応答数
iwsInstanceCount503	発行された 503 (Unavailable) の応答数

SNMP メッセージ

SNMP では、GET と SET の 2 種類のメッセージが定義されています。

各オブジェクトには、MIB 内で一意の識別子が割り当てられます。SNMP マネージャでオブジェクトにアクセスするには、その識別子を指定する GET コマンドおよび GETNEXT コマンドを発行します。プロキシエージェントは、指定されたオブジェクトの値を取得し、SNMP マネージャに転送します。ログに追加されたイベントがトラップフィルタの条件を満たしている場合、SNMP トラップが生成されます。トラップを生成しないイベントは、単に管理ログテーブルのエントリとして記録されます。これらには、標準の GET コマンドおよび GETNEXT コマンドで SNMP マネージャからアクセスできます。

GET メッセージと SET メッセージは、NMS からマスターエージェントに送信されます。管理インタフェースでは、このいずれかまたは両方のメッセージを使用できます。

SNMP は、プロトコルデータユニット (PDU) の形式でネットワーク情報をやり取りします。このユニットには、HTTP サーバーなどの管理対象デバイスに格納されている変数の情報が収められています。これらの変数は、必要に応じて NMS に報告される値とタイトルを含んでおり、管理対象オブジェクトとも呼ばれます。サーバーから NMS に送信されるプロトコルデータユニットを「トラップ」と呼びます。GET、SET、トラップの各メッセージの使用については、以降の各節で説明します。

SNMP トラップの送信先

SNMP トラップは、SNMP エージェントが NMS に送信するメッセージです。SNMP エージェントは、インタフェースの状態が稼動から停止に変わったときなどにトラップを送信します。SNMP エージェントは、NMS のアドレスを元にトラップの送信先を認識します。

SNMP マスターエージェントのトラップの送信先は、Sun ONE Application Server の管理インタフェースで設定できます。設定済みのトラップの送信先の表示、編集、および削除も可能です。管理インタフェースを使ってトラップの送信先を設定すると、実際に `CONFIG` ファイルが編集されます。

サーバーのサブエージェントは、重大なイベントが発生したとき、NMS にメッセージまたはトラップを送信します。次に例を示します。

1. サブエージェントがマスターエージェントに、サーバーの停止を通知します。
2. マスターエージェントは、イベントを報告するメッセージまたはトラップを NMS に送信します。
3. NMS は、ネットワーク管理アプリケーションを通して、情報をテキストまたはグラフィックで表示します。

SNMP トラップポートの設定方法については、168 ページの「SNMP マスターエージェントのインストール」を参照してください。

SNMP エージェントコミュニティ

SNMP エージェントコミュニティは、指定されたコミュニティに割り当てられたコミュニティ文字列と操作で構成されます。コミュニティ文字列は、SNMP エージェントが承認に使用する NMS 名を示すテキスト文字列です。NMS は、エージェントに送信するメッセージとともにコミュニティ文字列を送信します。

割り当てられる操作は、`get`、`set` のいずれか、または両方です。SNMP エージェントは、データ交換のために `get`、`set` のいずれか、または `get` と `set` の両方を実行する権限が NMS に与えられているかどうかを検証します。SNMP パケット内のコミュニティ文字列は秘匿されず、ASCII テキストで送信されます。

管理インタフェースを使用すると、指定されたコミュニティごとのコミュニティ文字列と許可された操作を設定および管理できます。SNMP エージェントコミュニティの設定方法については、168 ページの「SNMP マスターエージェントのインストール」を参照してください。

SNMP の設定

通常、SNMP を使用するには、システムにマスターエージェントと 1 個以上のサブエージェントをインストールし、実行している必要があります。サブエージェントを有効にする前に、マスターエージェントをインストールする必要があります。168 ページの「SNMP マスターエージェントのインストール」を参照してください。

SNMP の設定手順はシステムによって異なります。次の表に、さまざまな条件下での設定手順の概要を示します。実際の手順は、この章の後半で詳しく説明します。

サーバーの条件	手順 (詳細は次の各項で説明)
ネイティブエージェントが実行されていない	1. マスターエージェントを起動します。 2. システムにインストールされている各サーバーのサブエージェントを有効にします。
• ネイティブエージェントが実行されている • SMUX がサポートされていない • ネイティブエージェントを継続して使用する必要がない	1. 管理サーバーのマスターエージェントをインストールする前に、ネイティブエージェントを停止します。 2. マスターエージェントを起動します。 3. サーバーインスタンスごとに SNMP サブエージェントを設定します。
• ネイティブエージェントが実行されている • SMUX がサポートされていない • ネイティブエージェントを継続して使用する必要がある	1. プロキシ SNMP エージェントをインストールします。 2. プロキシ SNMP エージェントを起動します。 3. マスターエージェントのポート番号以外のポート番号を使って、ネイティブエージェントを再起動します。 4. マスターエージェントを起動します。 5. システムにインストールされている各サーバーのサブエージェントを有効にします。

最初に、次の点について確認します。

- SNMP エージェント (使用するオペレーティングシステムのネイティブエージェント) がシステムですでに稼動しているか
- 稼動している場合は、ネイティブ SNMP エージェントが SMUX 通信をサポートしているか

確認方法については、ご使用のシステムのマニュアルを参照してください。

注 管理サーバーの SNMP 設定の変更、新しいサーバーのインストール、または既存のサーバーの削除を行なった場合は、次の手順を実行する必要があります。

- (Windows 2000 の場合) Windows SNMP サービスまたはマシンを再起動する
- (UNIX の場合) 管理サーバーを使用して、SNMP マスターエージェントと SNMP サブエージェントを再起動する

この節では次の項目について説明します。

- プロキシ SNMP エージェントの使用 (UNIX/Linux)
- SNMP マスターエージェントのインストール

プロキシ SNMP エージェントの使用 (UNIX/Linux)

すでに実行中のネイティブエージェントを Sun ONE Application Server のマスターエージェントと同時に継続して使用する場合は、プロキシ SNMP エージェントを使用する必要があります。その前に、ネイティブマスターエージェントを停止します。詳しい手順については、ご使用のシステムのマニュアルを参照してください。

注 プロキシエージェントを使用するには、プロキシエージェントをインストールして起動する必要があります。さらに、Sun ONE Application Server のマスターエージェントが実行されているポート番号以外のポート番号を使って、ネイティブ SNMP エージェントを再起動する必要があります。

この節には次の項目があります。

- プロキシ SNMP エージェントのインストール
- プロキシ SNMP エージェントの起動
- ネイティブ SNMP デーモンの再起動

プロキシ SNMP エージェントのインストール

システム上で SNMP エージェントが稼動中で、ネイティブ SNMP デーモンを継続して使用する場合は、次の手順に従います。

1. SNMP マスターエージェントをインストールします。168 ページの「SNMP マスターエージェントのインストール」を参照してください。
2. プロキシ SNMP エージェントをインストールし、起動して、ネイティブ SNMP デーモンを再起動します。166 ページの「プロキシ SNMP エージェントの使用 (UNIX/Linux)」を参照してください。
3. SNMP マスターエージェントを起動します。171 ページの「SNMP マスターエージェントの有効化と起動」を参照してください。
4. サブエージェントを有効にします。177 ページの「サブエージェントの有効化」を参照してください。

SNMP プロキシエージェントをインストールするには、サーバーのルートディレクトリの *install_dir*/lib/snmp/sagt にある CONFIG ファイル (別の名前を付けることも可能) を編集して、SNMP デーモンの待機ポートを指定します。さらに、プロキシ SNMP エージェントが転送する MIB ツリーおよびトラップも指定します。

CONFIG ファイルの例を示します。

```
AGENT AT PORT 1161 WITH COMMUNITY public
SUBTREES 1.3.6.1.2.1.1,
       1.3.6.1.2.1.2,
       1.3.6.1.2.1.3,
       1.3.6.1.2.1.4,
       1.3.6.1.2.1.5,
       1.3.6.1.2.1.6,
       1.3.6.1.2.1.7,
       1.3.6.1.2.1.8
FORWARD ALL TRAPS;
```

プロキシ SNMP エージェントの起動

プロキシ SNMP エージェントを起動するには、コマンドプロンプトで次のように入力します。

```
# sagt -c CONFIG&
```

ネイティブ SNMP デーモンの再起動

プロキシ SNMP エージェントの起動後、CONFIG ファイルに指定されているポートでネイティブ SNMP デーモンを再起動します。

ネイティブ SNMP デーモンを再起動するには、コマンドプロンプトで次のように入力します。

```
# snmpd -P port_number
```

port_number は CONFIG ファイルに指定されているポート番号です。たとえば、Solaris プラットフォームで、前述した例の CONFIG ファイルのポート番号を使用する場合は、次のように入力します。

```
# snmpd -P 1161
```

SNMP マスターエージェントのインストール

注 管理インタフェースを使って、マスター SNMP エージェントをインストールして起動するには、サーバーが root として実行されている必要があります。

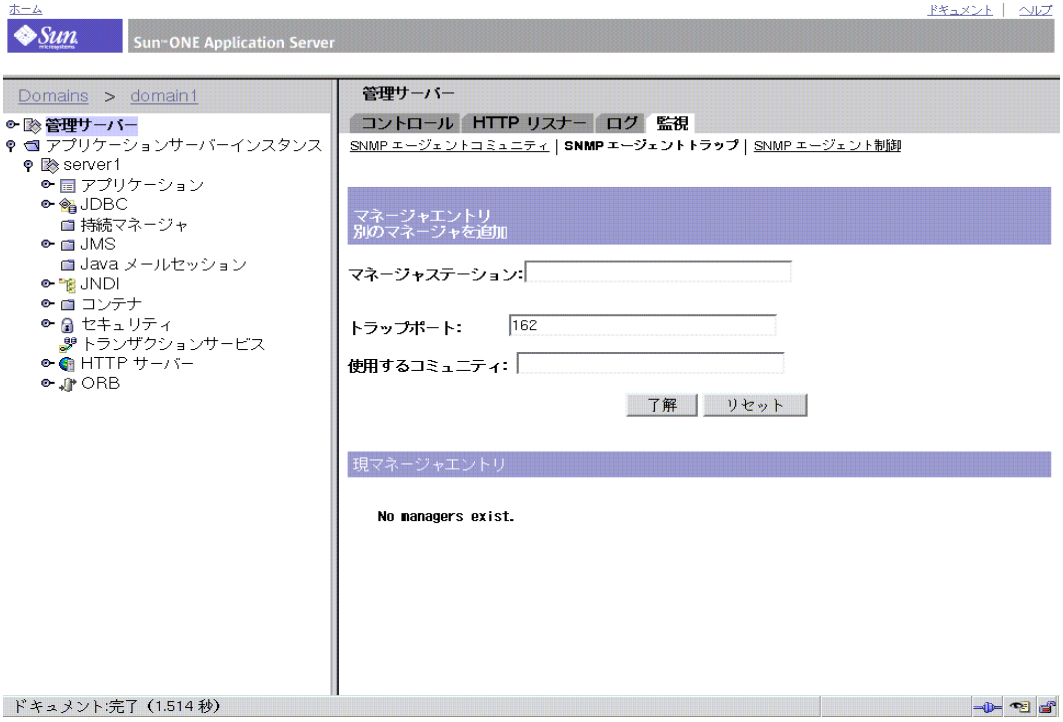
マスター SNMP エージェントをインストールするには、次の手順に従います。

1. root としてログインします。
2. SNMP デーモン (snmpd) がポート 161 で実行されているかどうかを確認します。

   SNMP デーモンが実行されていない場合は、Step 4 に進みます。

   SNMP デーモンが実行中の場合は、再起動の方法と、どの MIB ツリーがサポートされているかを確認してください。
3. SNMP デーモンが実行されている場合は、そのプロセスを強制終了します。
4. 管理インタフェースの左側のペインで、管理サーバーノードを選択します。
5. 「監視」タブを選択して、次の図で示される「SNMP エージェントトラップ」ページを表示します。

SNMP エージェントトラップ (SNMP Agent Trap) ページ

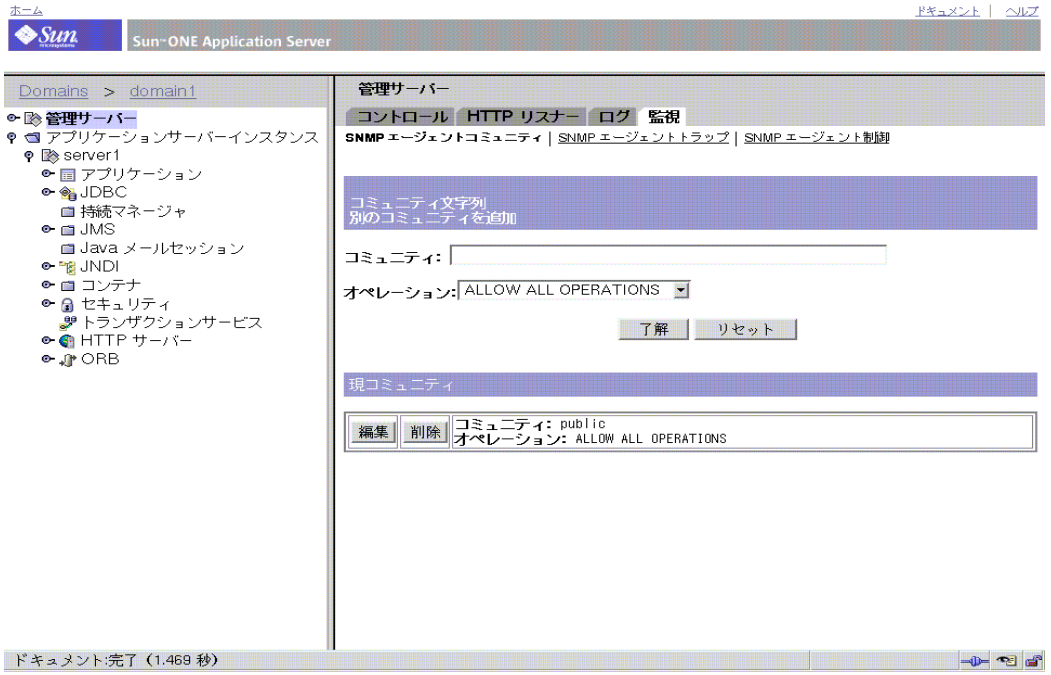


このページには、マネージャエントリの情報が表示されます。

6. ネットワーク管理ソフトウェアを実行しているシステムの名前を入力します。
7. ネットワーク管理システムがトラップを待機しているトラップポート番号を入力します。一般的なポート番号は 162 です。トラップの詳細については、164 ページの「SNMP トラップの送信先」を参照してください。
8. トラップで使用するコミュニティ文字列を入力します。コミュニティ文字列の詳細については、164 ページの「SNMP エージェントコミュニティ」を参照してください。
9. 「了解」をクリックします。
10. 「監視」タブの「SNMP エージェントコミュニティ」リンクをクリックします。

    次の図に示すコミュニティ文字列の情報が表示されます。

SNMP エージェントコミュニティ (SNMP Agent Community) ページ



11. マスターエージェントのコミュニティ文字列を入力します。
12. コミュニティの動作レベルを選択します。

    コミュニティが確立されると、このページの「現コミュニティ」という見出しの下のボタンで、コミュニティ設定の編集や削除を行うことができます。
13. 「了解」をクリックします。
14. 左ペインでアプリケーションサーバーインスタンスを選択してサーバーインスタンスにアクセスし、「変更を適用」をクリックします。

SNMP マスターエージェントの有効化と起動

マスターエージェントの動作は、CONFIG という名前のエージェント設定ファイルに定義されています。このファイルは手動で編集できます。SNMP サブエージェントを有効にする前に、マスター SNMP エージェントをインストールする必要があります。

注 マスターエージェントの再起動時に「システムエラーポートにバインドできませんでした」のようなバインドエラーメッセージが表示される場合は、`ps -ef | grep snmp` コマンドを使用して、`magt` が実行中であるかどうかを確認します。実行中である場合は、`kill -9` *pid* コマンドでこのプロセスを終了します。これで、SNMP の CGI が再度機能するようになります。

この節には次の項目があります。

- 別のポートを使用したマスターエージェントの起動
- SNMP マスターエージェントの手動設定
- マスターエージェントの CONFIG ファイルの編集
- sysContact 変数と sysLocation 変数の定義
- SNMP マスターエージェントの設定
- SNMP マスターエージェントの起動
- サブエージェントの有効化

別のポートを使用したマスターエージェントの起動

管理インタフェースでは、161 以外のポートで SNMP マスターエージェントを起動することはできません。別のポートを使用するには、次の手順に従って手動でマスターエージェントを起動してください。

1. *install_dir*`/lib/snmp/magt/CONFIG` を編集して、適切なポートを指定します。
2. 次のようにして起動スクリプトを実行します。

```
cd instance_root/admin-server ./start -shell
install_dir/lib/snmp/magt/magt
install_dir/lib/snmp/magt/CONFIG
install_dir/lib/snmp/magt/INIT
```

指定のポートでマスターエージェントが起動します。手動で起動した場合も、管理インタフェースで、このマスターエージェントが実行されていることを確認できます。

SNMP マスターエージェントの手動設定

SNMP マスターエージェントを手動で設定するには、次の手順に従います。

1. root としてログインします。
2. ポート 161 に実行中の SNMP デーモン (snmpd) があることを確認します。

   SNMP デーモンが実行中の場合は、再起動の方法と、どの MIB ツリーがサポートされているかを確認してください。その後、プロセスを強制終了します。
3. サーバーのルートディレクトリの lib/snmp/magt にある CONFIG ファイルを編集します。
4. CONFIG ファイルに sysLocation 変数と SysLocation 変数を定義します (オプション) 。173 ページの「sysContact 変数と sysLocation 変数の定義」を参照してください。

マスターエージェントの CONFIG ファイルの編集

CONFIG ファイルでは、マスターエージェントで動作するコミュニティおよびマネージャを定義します。マネージャの値は、有効なシステム名または IP アドレスにしてください。

基本的な CONFIG ファイルの例を示します。

```
COMMUNITY          public
                   ALLOW ALL OPERATIONS

MANAGER            manager_station_name
                   SEND ALL TRAPS TO PORT 162
                    WITH COMMUNITY public
```

sysContact 変数と sysLocation 変数の定義

CONFIG ファイルを編集して、MIB-II 変数の sysContact および sysLocation を指定する sysContact および sysLocation の初期値を追加できます。この例の sysContact および sysLocation の文字列が二重引用符で囲まれていることに注意してください。スペース、改行、タブなどが含まれている文字列は、二重引用符で囲む必要があります。16 進数の値を指定することもできます。

次に、sysContact 変数と sysLocation 変数が定義された CONFIG ファイルの例を示します。

```
COMMUNITY           public
                    ALLOW ALL OPERATIONS

MANAGER             nms2
                    SEND ALL TRAPS TO PORT 162
                    WITH COMMUNITY public

INITIAL             sysLocation "Server room
901 San Antonio Road
Palo Alto CA 94303
USA"

INITIAL             sysContact "John Doe
email:jdoe@sun.com"
```

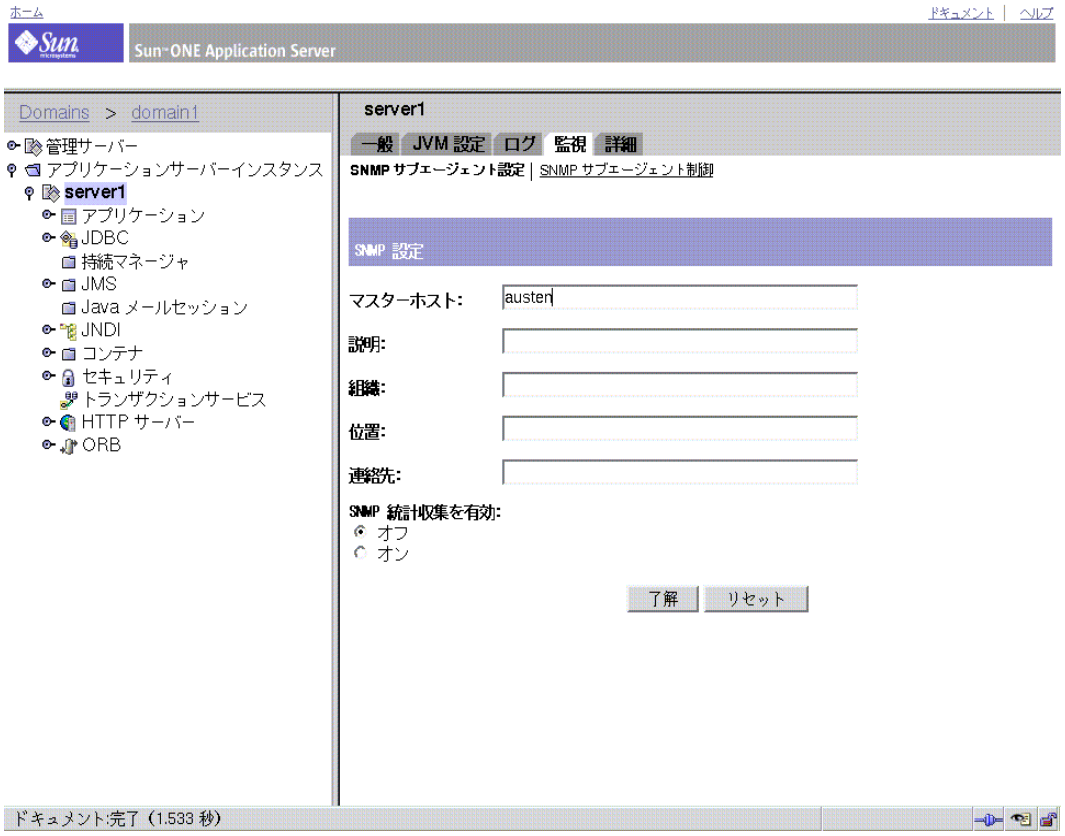
SNMP サブエージェントの設定

SNMP サブエージェントを設定するには、次の手順に従います。

1. 管理サーバーの左側のペインでサーバーインスタンスノードを選択します。
2. 右側のペインで「監視」タブを選択します。
3. 「SNMP サブエージェント設定」リンクを選択します。

   次に示すページが表示されます。

SNMP サブエージェント設定ページ



4. (UNIX のみ)「マスターホスト」フィールドにサーバーの名前とドメインを入力します。
5. サーバーの説明 (オペレーティングシステムの情報を含む) を入力します。
6. サーバーを管理する組織を入力します。
7. サーバーインスタンスの名前を入力します。
8. 「連絡先」フィールドに、サーバーの管理担当者の名前と連絡先を入力します。
9. 「SNMP 統計収集を有効」で「オン」を選択します。
10. 「了解」をクリックします。
11. 左ペインでアプリケーションサーバーインスタンスを選択してサーバーインスタンスにアクセスし、「変更を適用」をクリックします。

SNMP マスターエージェントの起動

SNMP マスターエージェントのインストール後、手動または管理インタフェースから管理サーバーを使用して SNMP マスターエージェントを起動できます。

SNMP マスターエージェントの手動による起動

マスターエージェントを手動で起動するには、コマンドプロンプトに次のように入力します。

```
# magt CONFIG INIT&
```

INIT ファイルは、システムの場所や連絡先情報など、MIB-II システムグループからの情報が格納された不揮発性ファイルです。INIT ファイルが存在しない場合、ファイルはマスターエージェントの初回の起動時に作成されます。

注 CONFIG ファイルに無効なマネージャ名が指定されている場合は、マスターエージェントの起動に失敗します。

標準以外のポートでマスターエージェントを手動で起動するには、次の 2 種類の方法のいずれかを使用してください。

方法 1 : CONFIG ファイルに、マスターエージェントがマネージャからの SNMP 要求を待機する各インタフェースのトランスポートマッピングを指定します。トランスポートマッピングを使うと、マスターエージェントは標準ポートと標準以外のポートで接続を受け入れることができます。また、マスターエージェントは、標準以外のポートで SNMP トラフィックを受け入れることができます。最大同時 SNMP 数は、1 プロセスあたりのオープンソケット数またはファイル記述子数に関するシステムの制限値によって決まります。トランスポートマッピングのエントリの例を示します。

```
TRANSPORT           extraordinary   SNMP
                    OVER UDP SOCKET
                     AT PORT 11161
```

CONFIG ファイルを手動で編集した後、コマンドプロンプトに次のように入力し、マスターエージェントを手動で起動します。

```
# magt CONFIG INIT&
```

方法 2 : /etc/services ファイルを編集して、マスターエージェントが標準ポートと標準以外のポートでも接続を受け入れることができるようにします。

管理サーバーによる SNMP マスターエージェントの起動

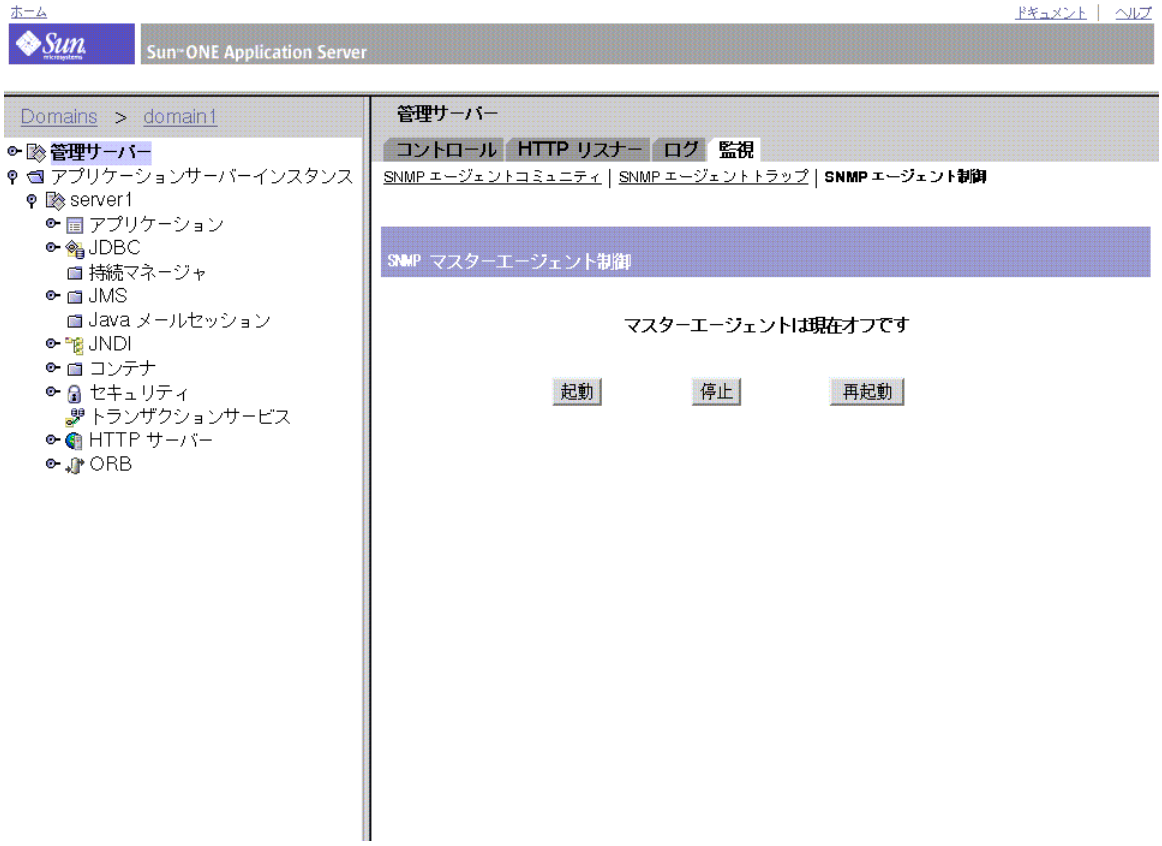
管理サーバーを使って SNMP マスターエージェントを起動するには、次の手順に従います。

注	SNMP マスターエージェントを起動するには、Sun ONE Application Server に root としてログインする必要があります。

1. 管理サーバーにログインします。
2. 左側のペインの管理サーバーノードを選択し、「監視」タブを選択します。
3. 右側のペイン最上部の「SNMP エージェント制御」リンクを選択します。

   次に示すページが表示されます。

SNMP エージェント制御ページ



4. 「起動」をクリックします。

「SNMP エージェント制御」ページで、SNMP マスターエージェントの停止と再起動を行うこともできます。

サブエージェントの有効化

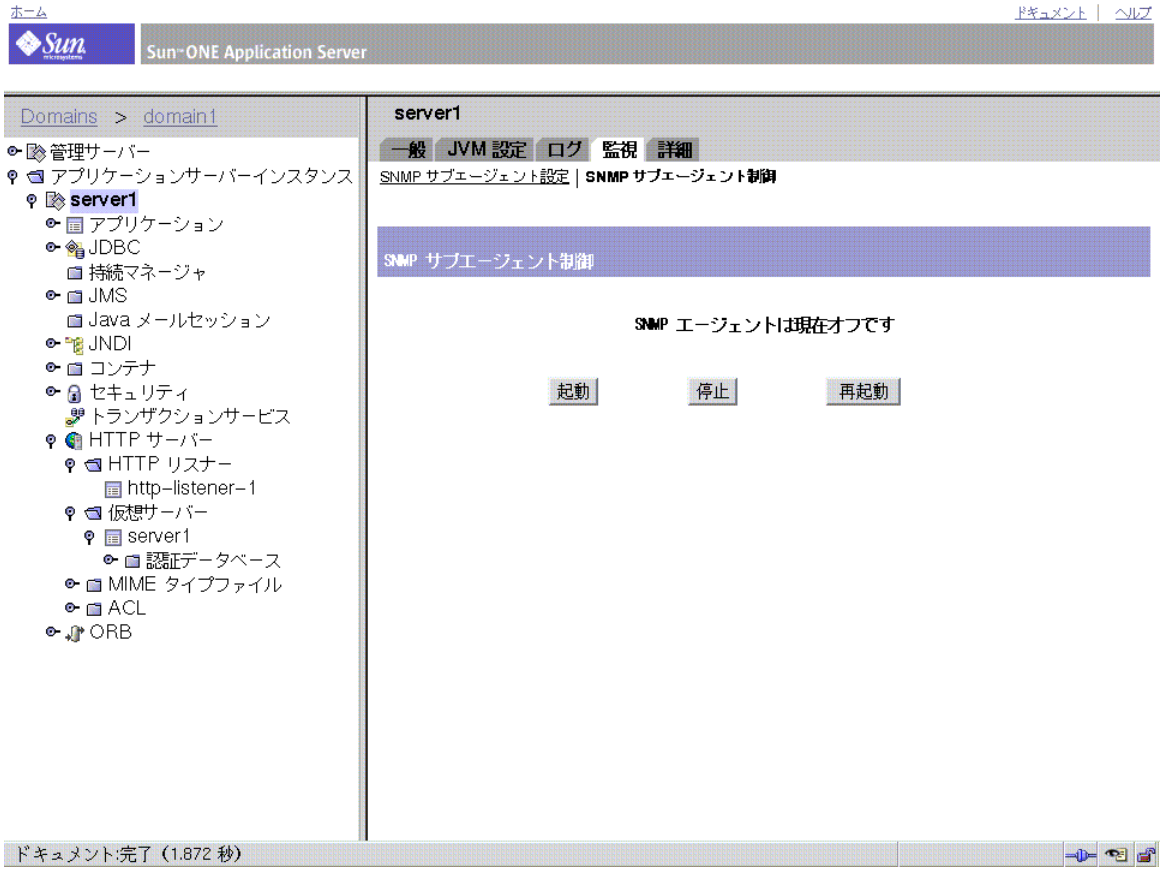
管理サーバーに付属するマスターエージェントをインストールしたら、マスターエージェントを起動する前に、サーバインスタンスのサブエージェントを有効にする必要があります。マスターエージェントの詳しいインストール方法については、168 ページの「SNMP マスターエージェントのインストール」を参照してください。

UNIX/Linux プラットフォームでは、サブエージェントを使用して SNMP 機能を停止できます。サブエージェントを停止してから、マスターエージェントを停止する必要があります。マスターエージェントを先に停止すると、サブエージェントを停止できなくなることがあります。その場合は、マスターエージェントを再起動し、サブエージェントを停止したあと、マスターエージェントを停止します。

SNMP サブエージェントを有効化するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスノードを展開します。
2. サーバーインスタンスを選択し、「監視」タブをクリックします。
3. 「SNMP サブエージェント制御」オプションを選択して、次の図で示されるページを表示します。

SNMP サブエージェント制御ページ



このページで、SNMP サブエージェントの起動、停止、再起動を行うことができます。サブエージェントの状態が、制御ボタンの上に表示されます。

Windows プラットフォームでは、Sun ONE Application Server の監視に Windows SNMP サービスが使用されます。このサービスは、「コントロールパネル / 管理ツール / サービス」から制御できます。

注	SNMP の設定を変更したときは、「適用」ボタンをクリックしてから、「SNMP サブエージェント制御」ページで SNMP サブエージェントを再起動する必要があります。

第 7 章

Web サーバープラグインの設定

この章では、Sun ONE Application Server による HTTP (HyperText Transfer Protocol) 要求の処理方法と、Sun ONE Application Server による Web サーバープラグインの設定方法および使用方法を説明します。また、Web サーバープラグインと Microsoft IIS および Apache Web サーバーの併用について、およびその設定方法についても説明します。

この章では次のトピックについて説明します。

- Web サーバープラグインについて
- クライアント要求の処理
- Web サーバープラグインの設定
- Web サーバープラグインの SAF
- Web サーバープラグインの使用
- Web サーバープラグインを使用するための Microsoft IIS の設定
- Apache サーバー用に Web サーバープラグインを設定

Web サーバープラグインについて

HTTP リバースプロキシプラグインを使うことで、ユーザーから Sun ONE Web Server または Sun ONE Application Server に指示を送り、特定の HTTP 要求を別のサーバーへ転送することができます。たとえば、特定の Web アプリケーションの要求を、インターネットに接続した Web サーバーから企業内のファイアウォールに転送するような設定が可能です。

Sun ONE Application Server 内では、Web サーバープラグインにより、1 つのサーバーインスタンスから別のサーバーインスタンスに HTTP (Web) 要求を転送できます。

Web サーバープラグインの機能は次のとおりです。

- プロキシサーバーからの接続をできるかぎり再利用する。これにより、着信要求を処理する際、新しい接続を確立する必要がなくなる
- Web サーバープラグインは、受信を開始した時点で、要求と応答のストリーミングを開始する。つまり、要求または応答が完全に収集された後に、これらをリモートサーバーへ転送する
- Web サーバープラグインは、同一リモートサーバーへの複数の送信 HTTP 接続を適切に管理する。Web サーバープラグインによる要求の転送用として確立された接続を「送信 HTTP 接続」と呼ぶ

Web サーバープラグインの機能を理解するためには、HTTP 要求の基礎知識、とりわけ Sun ONE Application Server が HTTP 要求を処理する方法を理解しておく必要があります。

クライアント要求の処理

Sun ONE Application Server は HTTP 要求を直接受け入れ、それに応答できるアプリケーションサーバーです。この節では、HTTP の基本的な概念と、Sun ONE Application Server が要求をどのように処理するかについて説明します。この節には次の項目があります。

- HTTP の基礎知識
- 要求処理プロセスの手順

HTTP の基礎知識

HTTP/1.1 プロトコルの機能を簡単にまとめます。

- クライアント (通常はブラウザ) はサーバーとの接続を確立し、要求を送信する
- サーバーは要求を処理し、応答を生成する。さらに `Connection: Close` ヘッダーを検出すると接続を終了する

要求は、`GET`、`POST` などのメソッドを指定する行、要求の対象となるリソースを示す URI (Universal Resource Identifier)、および HTTP プロトコルのバージョンを空白文字で区切った形式になります。

通常は、このあとに多数のヘッダー、ヘッダーの終わりを表す空白行が続き、さらに本文データが含まれる場合もあります。ヘッダーは、要求またはクライアントの本文データに関するさまざまな情報を提供します。一般に、ヘッダーは、`POST` メソッドと `PUT` メソッドにのみ送信されます。

次の例では、ブラウザがサーバー foo.com に要求を送信し、*/index.html* のリソースが返されます。この例では、本文データは送信されません。これは、要求のデータを取得するだけで送信しない GET メソッドが使用されているからです。

```
GET /index.html HTTP/1.0

User-agent:Mozilla

Accept:text/html, text/plain, image/jpeg, image/gif, */*

Host:foo.com
```

サーバーは要求を受信し、処理します。サーバーは複数の要求を同時に処理できるにもかかわらず、各要求を個別に処理します。各要求は、要求処理プロセスを設定する一連の手順に分割されます。

サーバーは、HTTP プロトコルのバージョン、HTTP 状態コード、および原因フレーズを空白文字で区切った形式で応答を生成します。通常は、このあとに多数のヘッダーが続きます。ヘッダーの終わりは空白行で表されます。そのあとに、応答の本文データが続きます。次に典型的な HTTP 応答を紹介します。

```
HTTP/1.0 200 OK
Server:Standard/7.0
Content-type:text/html
Content-length: 83

<HTML>
<HEAD><TITLE>Hello World</Title></HEAD>
<BODY>Hello World</BODY>
</HTML>
```

状態コードと原因フレーズから、サーバーが要求をどのように処理したかがわかります。通常は、要求の処理に成功したことを表す状態コード 200 が返されます。この場合、本文データには要求された項目が含まれます。別のサーバーまたはブラウザキャッシュへのリダイレクト、さまざまな種類の HTTP エラー (「404 Not Found」など) を表す結果コードが返されることもあります。

要求処理プロセスの手順

Sun ONE Application Server は、初期起動時に特定の初期化タスクを実行します。その後、ブラウザなどのクライアントからの HTTP 要求を待機します。要求を受信すると、最初に仮想サーバーを選択します。

仮想サーバーを選択すると、仮想サーバーの `obj.conf` ファイルにより、次の手順に従って要求の処理方法が指定されます。

1. AuthTrans (承認変換)

   要求とともに送信される承認情報 (名前、パスワードなど) を検証する

2. NameTrans (名前変換)

   論理 URI をローカルファイルシステムのパスに変換する

3. PathCheck (パスチェック)

   ローカルファイルシステムのパスの妥当性検査を実施し、要求元がこのファイルシステム上の所定のリソースに対してアクセス権を持っていることを確認する

4. ObjectType (オブジェクト型の識別)

   要求されたリソースの MIME (Multi-purpose Internet Mail Encoding) 型を特定する。たとえば、`text/html`、`image/gif` などの MIME 型がある

5. Service (応答の生成)

   応答を生成し、クライアントに返す

6. AddLog (ログエントリの追加)

   ログファイルにエントリを追加する

7. Error (サービス)

   前の手順でエラーが発生した場合にのみ実行される手順。エラーが発生した場合、サーバーはログにエラーメッセージを記録し、プロセスを中止する

Web サーバープラグインの設定

Web サーバープラグインの設定と動作は、一組の設定ファイルで決定されます。Sun ONE Application Server は、クライアントからの要求を処理する際、毎回これらのファイルに定義された設定情報を参照します。設定ファイルは、`obj.conf` と `init.conf` です。`obj.conf` ファイルは、たとえば `server1-obj.conf` のように、仮想サーバーの名前が最初につけられます。詳細については、371 ページの「obj.conf ファイル」を参照してください。

Sun ONE Application Server の各インスタンスは、サーバーが起動時に参照する固有の `init.conf` ファイルを持ちます。

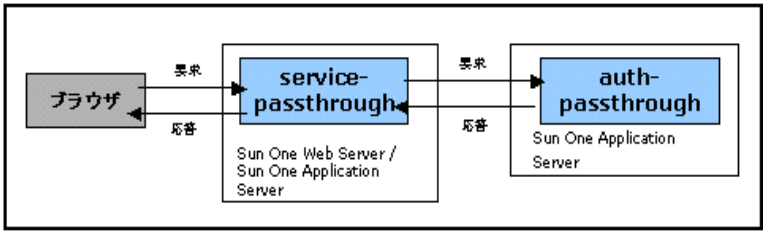
前の項目で説明したように、`obj.conf` 設定ファイルには、クライアントからの要求とこれに対する応答処理の各段階で実行する内容を、Sun ONE Application Server に指示する一連の指令が格納されています。各指令は、SAF (Server Application Function) を呼び出します。

Sun ONE Application Server の操作には、`obj.conf` ファイルが必要です。管理インタフェースを使ってサーバーに変更を加えると、システムにより、`obj.conf` が自動的に更新されます。

`init.conf` 設定ファイルは、初期化の際に、サーバーを設定する変数値を設定します。サーバーは、起動時に、このファイルに指定されている設定パラメータを実行します。詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

次の図は、Web ブラウザ、フロントエンドの Web サーバー、バックエンドのアプリケーションサーバー、および Web サーバープラグインの `service-passthrough` と `auth-passthrough` SAF の関係を示しています。

Web ブラウザ、Web サーバー、アプリケーションサーバー、Web サーバープラグイン SAF の関係



Web サーバープラグインの SAF

この節では、次の SAF (Server Application Functions) の機能と動作を説明します。

- init-passthrough
- auth-passthrough
- service-passthrough
- check-passthrough

init-passthrough

`init-passthrough` 関数は、Web サーバープラグインを初期化します。この関数は、Web サーバープラグインの使用を開始する前に呼び出す必要があります。

例

```
Init fn="load-modules" shlib="c:/plugins/passthrough.dll"
funcs="init-passthrough,auth-passthrough,check-passthrough,service-
passthrough" NativeThread="no"

Init fn="init-passthrough"
```

auth-passthrough

`auth-passthrough` SAF は、`AuthTrans-class` 指令に適用されます。

auth-passthrough 関数は、中間サーバー上で実行されている server-passthrough 関数によってコード化された、クライアント情報を求める着信 HTTP (Web) 要求を検査します。次のようなクライアント情報があります。

- 要求の発信元の IP アドレス
- 発信元クライアントで使用される SSL キーサイズ
- 発信元クライアントから提供される SSL クライアント証明書

auth-passthrough は、コード化されたクライアント情報を検出すると、着信要求が `service-passthrough` を実行している中間サーバーから転送されたものではなく、送信元クライアントから直接送信されたものとみなします。

これは、次のような 2 層配備のシナリオで便利です。

- Sun ONE Application Server インスタンスが、企業のファイアウォールの背後に配置された 2 番目のファイアウォールによって保護されている

- クライアントが S1AS インスタンスに直接接続することが許可されていない

そのようなネットワークアーキテクチャの場合、クライアントは必ず、プロキシプラグインが動作しているフロントエンドの Web サーバーに接続します。この Web サーバーは、要求を Sun ONE Application Server に転送します。したがって、Sun ONE Application Server が要求を受信できるのは、プロキシのホスト (ここでは Web サーバー) からだけであり、クライアントのホストから直接受信することはできません。これは、2 つのファイアウォールの背後に存在する Sun ONE Application Server インスタンス上に配備されたアプリケーションがクライアントの情報 (クライアントの IP アドレスなど) に対してクエリを実行すると、そのアプリケーションは、実際に要求を中継したホストであるプロキシホストの IP アドレスを取得することを意味しています。auth-passthrough SAF を使用すれば、代わりにリモートの (プロキシを介して) クライアント情報が返されるようにこの動作を変更することができます。

auth-passthrough は認証に使用できる情報 (要求の送信元の IP アドレスなど) を上書きできるので、信頼できるクライアントまたはサーバーだけに auth-passthrough を実行しているサーバーへの接続を許可することが重要です。予防措置として、auth-passthrough は、企業のファイアウォールで保護されているサーバーだけで実行することをお勧めします。インターネット経由でアクセスできるサーバーで、auth-passthrough SAF の実行を許可しないでください。auth-passthrough SAF は、発信元のクライアントに直接関連する情報が必要な場合にのみ使用してください。

上記のシナリオの場合、SSL クライアント認証を有効にできるのは Web サーバーに対してのみである点に注意してください。設定が正しく動作するようにするには、アプリケーションサーバーに対しては常に無効にします。

コマンドの例

```
AuthTrans fn="auth-passthrough"
```

service-passthrough

service-passthrough SAF は、Service-class 指令に適用されます。

service-passthrough SAF は、処理する要求をサーバーからサーバーへ転送します。service-passthrough SAF の設定により、要求の受信に使用された接続の種類に関係なく、リモートサーバーへの SSL 接続または SSL 以外の接続 (HTTPS または HTTP) を使用できます。service-passthrough SAF は、送信元クライアントに関する情報をコード化します。この情報は、リモートサーバー上で実行されている auth-passthrough によって復号化できます。

通常、service-passthrough 指令は、obj.conf 設定ファイル内でその他の指令と組み合わせて使用されます。次の例を参照してください。

```
<Object name="passthrough">
```

```
ObjectType fn="force-type" type="magnus-internal/passthrough"

Error reason="Bad Gateway" fn="send-error"
uri="$docroot/badgateway.html"

</Object>

<Object name="default">

....

NameTrans fn="assign-name" from="(/webapp1|/webapp1/*)" name="passthrough"

...

<</Object> >
```

バックエンドアプリケーションサーバーがダウンしている場合は、ユーザーには代わりにローカルの HTML ファイル `badgateway.html` が表示されます。
`service-passthrough` SAF を実行しているサーバーが、アクセス権を持っているファイルを提供し、拒否された要求だけをバックエンドアプリケーションに転送する必要がある場合は、`ObjectType` 行を次のように変更します。

```
ObjectType fn="check-passthrough" type="magnus-internal/passthrough"
```

check-passthrough

`check-passthrough` SAF は、`ObjectType-class` 指令に適用されます。

check-passthrough 関数は、要求されたリソース (HTML ドキュメント、GIF イメージなど) がローカルサーバー上で使用可能であるかどうかを確認します。要求されたリソースがローカルに存在しない場合、`check-passthrough` SAF は、要求を別のサーバーに転送し、`service-passthrough` SAF で処理することを示す型を設定します。

パラメータ :

`type` - (オプション) 要求されたリソースが存在しない場合に設定される型。デフォルトは `magnus-internal/passthrough`

例

```
ObjectType fn="check-passthrough"
```

Web サーバープラグインの使用

Sun ONE Web Server または Sun ONE Application Server インスタンスにプラグインを追加するには、次の手順に従います。

1. UNIX の場合は `libpassthrough.so` が *install_dir*`/plugins/passthrough/bin` ディレクトリに存在することを、Windows の場合は `passthrough.dll` ファイルが *install_dir*`/bin` ディレクトリに存在することを、それぞれ確認します。Sun ONE Web Server 6.0 にプラグインを追加するときは、Sun ONE Application Server 7 のインストールからプラグインをコピーする必要があります。

2. 設定ファイル *install_dir*`/config/init.conf` (Sun ONE Application Server 7 の場合) または *install_dir*`/config/magnus.conf` (Sun ONE Web Server 6.0 の場合) に、次の行を追加します。各行は `Init` で開始し、1 行で記述してください。

   Windows 環境では :

   ```
   Init fn="load-modules" shlib="c:/install_dir/bin/passthrough.dll"
   funcs="init-passthrough,auth-passthrough,check-passthrough,
   service-passthrough" NativeThread="no"

   Init fn="init-passthrough"
   ```

   UNIX 環境では :

   ```
   Init fn="load-modules"
   shlib="install_dir/plugins/passthrough/bin/libpassthrough.so"
   funcs="init-passthrough,auth-passthrough,check-passthrough,servi
   ce-passthrough" NativeThread="no"

   Init fn="init-passthrough"
   ```

3. 設定ファイル `config/`*appserver-server-instance*`-obj.conf` の先頭に次の行を追加して、`obj.conf` に passthrough `<Object>` を追加します。Sun ONE Application Server では、`config/`*<appserver-server-instance>*`-obj.conf` ファイル上で手順を実行する必要があります。

   ```
   <Object name="passthrough">
   ObjectType fn="force-type" type="magnus-internal/passthrough"
   Service type="magnus-internal/passthrough"
   fn="service-passthrough" servers="server"
   Error reason="Bad Gateway" fn="send-error"
   uri="$docroot/badgateway.html"
   <</Object> >
   ```

   server は、次の形式で記述した URL を意味します。

   `http://`*servername*`:port`

4. 設定ファイル *install_dir*/config/obj.conf で、デフォルトオブジェクトの先頭に次のような行を追加して、どの URI を転送するかを設定します。

   ```
   NameTrans fn="assign-name" from="(/uri|/uri/*)"
   name="passthrough"
   ```

   uri はリモートサーバー上に配備された Web アプリケーションのコンテキストルートです。passthrough は、手順 3 で指定した obj.conf 内の <Object> の名前です。

   次に例を示します。

   ```
   <Object name="default">
   ...
   NameTrans fn="assign-name" from="(/webapp1|/webapp1/*)"
   name="passthrough"
   ...
   <</Object> >
   ```

5. サーバーを再起動します。

注 プラグインライブラリの名前は、Solaris および Linux では libpassthrough.so です。Windows では、passthrough.dll です。

Web サーバープラグインを使用するための Microsoft IIS の設定

Web サーバープラグインと併用する場合の Microsoft Internet Information Services の設定には、Microsoft IIS を使用するための Web サーバープラグインの設定と、Web サーバープラグインを使用するための Microsoft IIS の設定が含まれます。

サーバープールを設定し、別のサーバー上で稼働する複数のアプリケーションを扱うこともできます。

この節では次の項目について説明します。

- IIS 用に Web サーバープラグインを設定
- Web サーバープラグイン用に IIS を設定
- 複数のサーバープールの設定
- sun-passthrough プロパティファイルの例

IIS 用に Web サーバープラグインを設定

IIS 用に Web サーバープラグインを設定するには、次の手順を実行します。

1. C:¥ のコマンド行プロンプトで次のコマンドを実行し、IIS の `wwwroot` ディレクトリの下に Web サーバープラグインのディレクトリを作成します。

   ```
   md ¥Inetpub¥wwwroot¥sun-passthrough
   ```

2. プラグインファイルを `C:¥Inetpub¥wwwroot¥sun-passthrough` ディレクトリにコピーします。

3. テキストエディタを使って、Sun ONE Application Server がインストールされているマシンの URL を `C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.properties` ファイルに追加します。

   テキストエディタを使って、次の情報を追加する必要があります。

   `server=http://`*`appservername`*`:`*`port`*

   appservername は Sun ONE Application Server がインストールされているマシンのホスト名または IP アドレス、`port` はそのマシンが待機しているポートの番号です (通常は 80)。

4. Sun ONE Application Server に処理させるコンテキストルートを `C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.properties` ファイルに追加します。

   これらのコンテキストルートは、Sun ONE Application Server に配備したアプリケーションのコンテキストルートに対応している必要があります。これらのコンテキストルートに対する要求は Sun ONE Application Server によって処理され、その他の要求は IIS Web サーバーによって処理されます。要求を Web アプリケーションに渡すコマンド行は、次のとおりです。

   `passthrough=/`*`webapplication`*

   /webapplication は Web アプリケーションのコンテキストルートです。すべての要求を Sun ONE Application Server に渡すには、次の行を追加します。

   ```
   passthrough=/
   ```

これで、Microsoft IIS のルートディレクトリに Web サーバープラグインを設定できました。プロセスを完了するには、次に Web サーバープラグインを使えるように Microsoft IIS を設定します。

Web サーバープラグイン用に IIS を設定

Web サーバープラグイン用に IIS を設定するには、Windows のインターネットサービスマネージャを開く必要があります。インターネットサービスマネージャは、「コントロールパネル」フォルダ内の「管理ツール」フォルダにあります。

インターネットサービスマネージャを開き、次のタスクを実行します。

1. プラグインを有効にする Web サイトを選択します。Web サイトの名前は、通常は 既定の Web サイトです。
2. Web サイトを右クリックし、「プロパティ」をクリックして「プロパティ」ノートブックを開きます。
3. 「ISAPI フィルタ」タブを開き、「追加」ボタンをクリックします。次の手順に従って新しい ISAPI フィルタを追加します。
   a. 「フィルタ名」フィールドに Sun ONE Application Server と入力します。
   b. 「実行ファイル」フィールドに `C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.dll` と入力します。
   c. 「OK」をクリックし、「プロパティ」ノートブックを閉じます。
4. 次に、新しい仮想ディレクトリを作成し、設定します。次の手順に従って、仮想ディレクトリを新規作成し、設定してください。
   a. 既定の Web サイトを右クリックし、「新規」、「仮想ディレクトリ」を順にクリックします。「仮想ディレクトリの作成ウィザード」が開きます。
   b. 「エイリアス」フィールドに `sun-passthrough` と入力します。
   c. 「ディレクトリ」フィールドに `C:¥Inetpub¥wwwroot¥sun-passthrough` と入力します。
   d. 実行権限のボックスにチェックマークをつけ、それ以外のすべてのパーミッションボックスのチェックマークを外します。
   e. 「完了」をクリックします。

5. 新しい設定を適用するには、Web サーバーを停止し、起動しなおす必要があります。Web ブラウザを停止するには、Web サイトを右クリックし、「停止」を選択します。Web ブラウザを起動するには、Web サイトを右クリックし、「起動」を選択します。

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

   `http://`*`webservername`*`/`*`webapplication`*

   webservername は Web サーバーのホスト名または IP アドレスです。*webapplication* は `C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.properties` に指定したコンテキストルートで、Web サーバー、Web サーバープラグイン、および Sun ONE Application Server の正常な動作の検証に使われます。

複数のサーバープールの設定

`sun-passthrough.properties` ファイルに複数サーバープールを設定することで、Web アプリケーションを複数のアプリケーションサーバーに区切ることができます。これにより、一部のアプリケーションを一部のサーバーセットで実行し、別のアプリケーションを別のサーバーセットで実行できます。各サーバープールには、アルファベットと数字を使って他と重複しない名前をつけます。188 ページの「Web サーバープラグインを使用するための Microsoft IIS の設定」で説明した方法で Microsoft IIS 用に Web サーバープラグインをインストールおよび設定したら、`C:¥Inetpub¥wwwroot¥sun-passthrough¥sun-passthrough.properties` ファイルを開き、関連する server プロパティと passthrough プロパティの行に、サーバープールにつけた一意の名前を追加します。サーバープール名の後にはピリオド (.) をつけます。

たとえば、`sun-passthrough.properties` ファイルの次の行は、2 つのサーバープールを定義しています。最初のサーバープールは `server-a` と、コンテキストルート `/app1` のサービス要求から構成されます。もう一方のサーバープールは `server-b` と、コンテキストルート `/app2` および `/app3` のサービス要求から構成されます。

```
server=http://server-a
passthrough=/app1
serverpool2.server=http://server-b
serverpool2.passthrough=/app2
serverpool2.passthrough=/app
```

sun-passthrough プロパティファイルの例

```
# Sun ONE Application Server Web サーバープラグイン（IIS 用）
#
# このファイルは、IIS 用の Sun ONE Application Server Web サーバープラグインの設定に使い
# ます。「#」から始まる行は無視されます。
# server
#
# server プロパティは、アプリケーションサーバーの URL を指定します。複数の server プロパティ
# を指定すると、プラグインは指定のアプリケーションサーバーに負荷を分散します。
#
server=http://localhost:8080
# passthrough
#
# passthrough プロパティは、Web アプリケーションのコンテキストルート（仮想ディレクトリ）を
# 指定します。指定したコンテキストルートの要求はアプリケーションサーバーに渡され、処理されます。
# 「passthrough=/」を指定すると、すべての要求がアプリケーションサーバーに渡され、処理されます。
#
# passthrough プロパティを指定するときは、詳細なものから順に入力します。たとえば、
# 「passthrough=/apps/app1」は「passthrough=/apps」の前に指定します。
#
# passthrough は、複数を指定できます。
#
#passthrough=/webapp
#passthrough=/servlets
#passthrough=*.jsp
passthrough=/
# prefix
#
# prefix プロパティは、プラグインの DDL ファイル（sun-passthrough.dll）を含む IIS 仮想
# ディレクトリを指定します。
```

```
#
prefix=/sun-passthrough
# error-url
#
# error-url プロパティは、アプリケーションサーバーを利用できない場合にクライアントをリダイレク
# トするページの URL を指定します。
#
#error-url=/badgateway.htm
# プール名にピリオド (.) を続け、server プロパティと passthrough プロパティの名前を指定する
# ことで、複数のサーバープールを設定できます。プールには、アルファベットと数字を使って自由に名前
# をつけられます。
#
# たとえば、次の例は 2 つのサーバープールを定義しています。一方のサーバープールは「/app1」に
# ある Web アプリケーションに対応し、他方は「/app2」と「/app3」にある Web アプリケーションに
# 対応します。
#
#serverpool1.server=http://server-a
#serverpool1.passthrough=/app1
#
#serverpool2.server=http://server-b
#serverpool2.passthrough=/app2
#serverpool2.passthrough=/app3
```

Apache サーバー用に Web サーバープラグインを設定

ここでは、Apache のソースコードをコンパイルし、Apache Web サーバーのインストールを設定する方法について説明します。

1. 最新の Apache ソースコードを `www.apache.org` からダウンロードします。

   ソースコードを解凍します。ソースコードは圧縮されたアーカイブとして提供されます。Apache 1.3.22 をインストールする場合、ソースコードのアーカイブ名は `apache_1.3.22.tar.gz` です。

2. 次のコマンドを使ってアーカイブを解凍します。

   ```
   $ tar -zxvf apache_1.3.26.tar.gz
   ```

   このコマンドにより、`apache_1.3.26` という名前のディレクトリが、現在の作業ディレクトリ内に作成されます。

3. ここで、Apache のソースコードをコンパイルできるように環境を設定する必要があります。このソースコードには、`configure` という名前のスクリプトが付属しています。このスクリプトは、Apache を正しくコンパイルするのに必要なサポートファイル (ヘッダー、共有ライブラリ、ユーティリティプログラムなど) がユーザーの環境に含まれているかどうかをチェックするためのものです。

   環境を設定するには、Apache ソースディレクトリに移動し、次の手順を実行します。

   a. Apache を Solaris または Linux にインストールする場合、次のパスが存在していることを確認します。

      - `CC=/opt/SUNWspro/bin/cc`
      - ar が PATH に含まれている (このコマンドは `/usr/ccs/bin` 内にある)
      - Makefile (このコマンドは `/usr/ccs/bin/make` 内にある)

   b. 次のコマンドを実行します。

      ```
      $ ./configure -enable-module=proxy --prefix=/usr/local/apache
      ```

   上記のコマンドで指定したディレクトリは任意の場所を指定できます。Apache のインストール先のパスを指定できます。prefix 引数は Apache のインストール先を示します。このコマンドを実行すると、画面上にメッセージが数行出力されます。本質的には、このコマンドはユーザーのシステム設定に応じたビルド用 Makefile を生成します。`configure` の実行時にエラーが発生した場合、ヘッダーファイルやユーティリティプログラムが不足している可能性があります。インストールしてから続行してください。

4. configure スクリプトの実行が正常終了したら、Apache をコンパイルできます。それには、次のように make コマンドを実行します。

   $ make

   このコマンドを実行すると、Apache ソースコードのコンパイルと Apache のリンクを実行中であることを示すメッセージが画面上に数行出力されます。通常、この処理でエラーは発生しません。エラーが発生した場合は、Apache のすべてのライブラリファイルとユーティリティプログラムが正しくダウンロードされたかどうかを確認してください。

5. ここで、Apache をインストールします。Apache のインストール先は、/usr/local/apache ディレクトリ (別のディレクトリを指定した場合はそのディレクトリ) になります。Apache をインストールするには、次のコマンドを実行します。

   ```
   $ make install
   ```

   このコマンドが正常に実行されると、システムに Apache がインストールされます。Apache のインストールファイルが次のディレクトリに格納されます。

   /usr/local/apache

   /usr/local/apache/ ディレクトリには、httpd.conf という名前の主要な設定ファイルがインストールされます。

6. 次のコマンドを実行して Apache を設定します。

   ```
   $ Configure Apache
   ```

   Apache の設定は、httpd.conf ファイルで行われます。このファイルには、Apache サーバーの各種操作パラメータを決定するいくつかの Apache 指令が記述されています。単純な Apache インストールの場合、変更する必要があるのは、次に示すわずかな指令だけです。

   - Server Root “ (Server Root は、Server Root &/usr/local/apache、/space/apache など、Apache のインストール先のパス)
   - Port “5000” (ユーザーの選択)”

これで、Apache はデフォルトの動作をし、Web サーバーとして機能するように設定されました。その他のパラメータを設定する必要がある場合は、設定ファイルを参照してください。

7. Apache には apachectl という名前のスクリプトが付属しています。このスクリプトを使うと、Apache の起動、停止、および再起動が簡単に行えます。Apache を起動するには、次のコマンドを実行します。

   ```
   $ /usr/local/apache/bin/apachectl start
   ```

   Apache を停止するには、次のコマンドを実行します。

   ```
   use /usr/local/apache/bin/apachectl stop
   ```

   Apache の起動後に、Apache が正しくインストールされたかどうかをテストできます。Apache が起動したら、Web ブラウザでアドレス「http://localhost/」を入力します。Apache が正しくインストールされ、実行されている場合は、そのことを示すメッセージを含むテストページが表示されます。

第 8 章

J2EE コンテナの設定

Sun ONE Application Server は、J2EE 1.3 仕様に準拠したさまざまな J2EE コンテナを提供します。コンテナは、EJB (Enterprise Java Beans) や MDB (メッセージ駆動型 Beans) などの J2EE アプリケーションコンポーネントの実行時サポートを提供します。MDB および EJB が、他の J2EE アプリケーションコンポーネントと直接対話することはありません。これらのアプリケーションコンポーネントは、EJB コンテナのプロトコルとメソッドを使って、その他のアプリケーションコンポーネントや、Java トランザクションサービスなどのプラットフォームサービスと対話します。コンテナは、アプリケーションコンポーネントと J2EE サービスの間に介在します。このため、コンテナは、コンポーネントの配備記述子によって定義されたサービス (宣言型トランザクション管理、セキュリティチェック、リソースプーリング、状態管理など) を透過的に組み込むことができます。

Sun ONE Application Server には、Web コンテナおよび EJB コンテナが組み込まれています。

この章では次のトピックについて説明します。

- Web コンテナについて
- EJB コンテナについて

Web コンテナについて

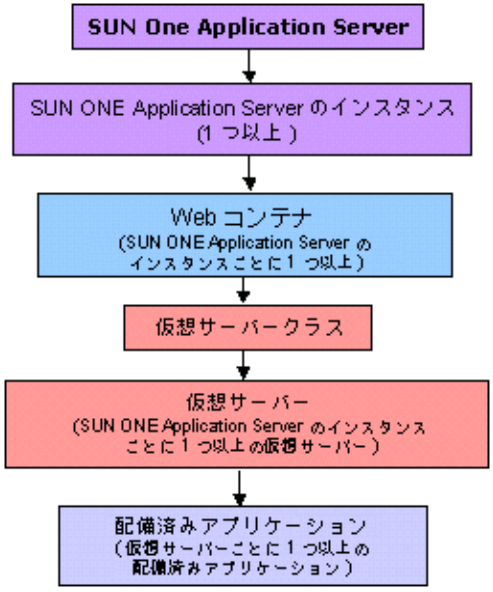
Web コンテナは、Web アプリケーションをホストする J2EE コンテナです。Web コンテナは、サーブレットと JSP (Java Server Pages) の実行環境を開発者に提供することにより、Web サーバーの機能を拡張します。サーブレットでは、コンポーネントをベースにし、プラットフォームに依存しない方法で Web ベースのアプリケーションを構築できます。CGI (Common Gateway Interface) プログラムのパフォーマンス制限はありません。JSP テクノロジは、サーブレットテクノロジの拡張であり、HTML およ

び XML ページのオーサリング機能を提供します。Web コンテナに含まれるサーブレットや JSP は、EJB (Enterprise Java Beans) コンテナ内の Bean メソッドを呼び出すことができます。Bean メソッドは、ORB (Object Request Broker) によってローカルまたはリモートで呼び出されます。

また、Web コンテナは、JNDI (Java Naming Directory Interface) で検索されたローカル EJB に対して、Web アプリケーションアクセスを提供します。

次の図では、Sun ONE Application Server アーキテクチャにおける Web コンテナの役割と位置付けを示します。

Sun ONE Application Server アーキテクチャ内の Web コンテナ



この節には次の項目があります。

- Web コンテナの役割
- Web アプリケーションの設定
- Web アプリケーションの配備
- シングルサインオン機能
- Web コンテナのロギング

Web コンテナの役割

Web コンテナの第 1 の役割は、Web アプリケーションにランタイム環境を提供し、コンテナ内でホストされる Web アプリケーションにサービス (データベースアクセス、セキュリティ、マルチスレッドなど) を提供することです。Web アプリケーションは、Sun ONE Application Server 上で完全なアプリケーションを構成するサーブレット、HTML ページ、クラス、およびその他のリソースの集合です。

Web アプリケーションの要素を次に示します。

- サーブレット
- JSP ページ
- ユーティリティクラス
- 静的ドキュメント (HTML、イメージ、音声ファイルなど)
- クライアントサイドの Java アプレット、Beans、およびクラス
- 上記のすべての要素を 1 つにまとめる記述的なメタ情報

Web アプリケーションは、Sun ONE Application Server で実行中の Web コンテナに配備できます。

Sun ONE Application Server での Web サーバープラグインの詳しい設定方法および使用方法については、179 ページの「Web サーバープラグインの設定」を参照してください。

Web アプリケーションの設定

Web コンテナの設定により、仮想サーバー内に Web アプリケーションを配備することもできます。Web コンテナに複数の仮想サーバーを設置する設定も可能です。各仮想サーバーには、任意の数の Web アプリケーションをホストさせることができます。Web アプリケーションの使用範囲は、仮想サーバーのコンテキスト内です。仮想サーバーの詳細については、第 15 章「仮想サーバーの使用」を参照してください。

この節では次の項目について説明します。

- 仮想サーバー属性
- Web モジュール属性

仮想サーバー属性

仮想サーバーには、設定可能な一定の属性値を指定できます。仮想サーバーには、複数の Web アプリケーションを関連付けることができます。ユーザーは Web アプリケーションに「サインオン」する必要があります。

server.xml ファイルでシングルサインオンの属性である sso-enabled をデフォルト値 true に設定した場合、ユーザーは特定の仮想サーバーに関連付けられたいずれか 1 つの Web アプリケーションにサインオンできます。この場合、同じ仮想サーバーで実行中のその他すべての Web アプリケーションが、ユーザーの識別情報を認識します。sso-enabled 属性の値を false に設定した場合、この仮想サーバーのすべてのアプリケーションで、シングルサインオンが無効になります。

sso-enabled 属性は動的に設定できます。つまり、サーバーを再起動しなくても設定内容が有効になります。

シングルサインオンの詳細については、202 ページの「シングルサインオン機能」を参照してください。

Web モジュール属性

Web アプリケーションの WEB-INF ディレクトリにある sun-web.xml ファイルには、Sun ONE Application Server 固有の配備記述子が指定されます。

通常、sun-web.xml ファイルは Web アプリケーションごとに 1 つずつ設定されます。ただし、Web コンテナは、sun-web.xml ファイルを Web アプリケーションごとに 1 つずつ設定しなくても動作します。sun-web.xml ファイルが存在しない場合、Web コンテナは、Sun ONE Application Server 固有のすべての属性についてデフォルト値を使用します。

context-root 属性

この属性は、Web アプリケーションのインストール先となるコンテキストルートを定義します。属性値として空文字列を指定した場合、この Web アプリケーションが仮想サーバーのデフォルトの Web アプリケーションになります。仮想サーバーのデフォルトの Web アプリケーションは、仮想サーバー上に配備されたその他の Web アプリケーションに解決されない要求すべてに応答します。各仮想サーバーには、デフォルトの Web アプリケーションが 1 つずつ割り当てられます。

デフォルトの Web アプリケーションの場合、このフィールドの値は空文字列 "" にします。

location 属性

この属性には、デフォルトの Web アプリケーションの位置を表す有効なディレクトリパスを指定します。インストール時に、デフォルトの Web アプリケーションの位置は modules/*default-web-app*/ ディレクトリに設定されます。

location 属性は必須であり、その値は WAR (Web ARchive) ファイルのコンテンツが抽出されるディレクトリの絶対パスまたは相対パスになります。指定されたパスが相対パスの場合、そのパスは、仮想サーバーレベルで定義されたアプリケーションルートディレクトリと関連を持っている必要があります。

次に例を示します。

location="*applications/<ear name>/<war-module name>/*"

location="*modules/<war-module name>*"

location="*/u/myapps/<war-module name>*"

location="*/u/myapps/<ear-name>/<war-module name>*"

enabled 属性

この属性のデフォルト値は true です。これは、Web アプリケーションがサービス要求に対して有効であることを示します。enabled 属性の値を false にすると、Web アプリケーションをサービス要求に対して一時的に無効にできます。ただし、Web アプリケーションのコンテンツは、ハードディスクに格納されているため、削除されません。

Web アプリケーションの配備

Web コンテナは、WAR (Web ARchive) ファイルまたは WAR ファイルの分割ビュー (WEB-INF/lib クラス、WEB-INF/ クラスなど) を含むディレクトリから Web アプリケーションを配備します。アプリケーションを配備するために、サーバーを再起動する必要はありません。

Web コンテナは、各仮想サーバーに「デフォルト」の Web アプリケーションを配備します。デフォルトの位置 (ディレクトリ) は、仮想サーバーの app root ディレクトリのサブディレクトリ、modules/*default-web-app*/ です。このデフォルトの Web アプリケーションは、仮想サーバー上に配備されたその他の Web アプリケーションに解決されない要求すべてに応答します。この Web アプリケーションは、/servlet/* への要求を処理する呼び出し元サーブレットと、JSP ページを提供する JSP サーブレットで構成されます。デフォルトの Web アプリケーションから EJB にアクセスするためには、web.xml ファイルおよび sun-web.xml ファイル内の EJB 参照を示す必要があります。

デフォルトの Web アプリケーションは、次のように、仮想サーバーの server.xml に定義されます。

```
<web-module context-root="" location="modules/default-web-app/">
```

動的再配備とホット配備機能

動的再配備により、サーバーを再起動することなく既存のアプリケーションを配備し直すことができます。動的再配備は、アプリケーションの設定 (xml ファイルのコンテンツ) や特定のクラスが変更されたときに行われます。動的再配備を行うと、アプリケーションクラス全体の動的再読み込みをした場合と同じ結果になります。さらに、

新しいアプリケーションコンテキスト (web および ejb) が作成され、古いアプリケーションコンテキストが削除されます。このように、動的再配備では、まったく新しいアプリケーションインスタンス (既存のセッションデータを除く) が作成されることになります。動的再配備をサポートするには、配備モードにする必要があります。この機能の使用時に発行される例外は、動的再読み込みで発行される例外とほぼ同じです。また、サーバーの再起動が必要になる変更を設定に加えた場合、変更内容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。動的再読み込み機能は、それを指定する中央の設定ファイルを持つアプリケーションおよび非共有のスタンドアロンモジュールだけで有効です。

Web アプリケーションの再読み込みを行うと、セッションマネージャの持続性の設定とは関係なく、既存のセッション情報がすべて自動的に保存および復元されます。

ホット配備は、サーバーを再起動することなく、サーバーの実行時にアプリケーションを配備する機能です。この機能では、動的再配備と同じインフラストラクチャを使用します。ただし、以前の状態は引き継がれないため、この機能は本番稼働時にサポートされます。

シングルサインオン機能

ユーザーが特定の仮想サーバー上で、任意の Web アプリケーションの保護されていないリソースだけにアクセスする場合、自分自身を認証する必要はありません。

特定の仮想サーバー上で、任意の Web アプリケーションの保護されているリソースにアクセスする場合、ユーザーはアクセス対象の Web アプリケーションに定義されているログインメソッドを使って、自分自身を認証します。

認証が完了すると、関連するすべての Web アプリケーションで、このユーザーに割り当てられたロールによるアクセス制御が行われます。ユーザーは、各 Web アプリケーションで個別に認証を行う必要はありません。

ユーザーが Web アプリケーションからログアウトすると、このユーザーのセッションは、すべての Web アプリケーションで無効になります。それ以降、アプリケーション内の保護されているリソースにアクセスするには、再度ユーザー認証を行う必要があります。

シングルサインオン機能は、HTTP Cookie を使ってトークンを転送し、個々の要求と保存されているユーザーの識別情報を関連付けます。したがって、この機能は、Cookie をサポートするクライアント環境以外ではサポートされません。

Web コンテナのロギング

さまざまなログレベルを設定することにより、Web コンテナのデフォルトのロギング動作と、仮想サーバー内でホストされている任意のアプリケーションのデフォルトのロギング動作を制御できます。このロギング動作は、アプリケーション独自のロギングには影響しません。

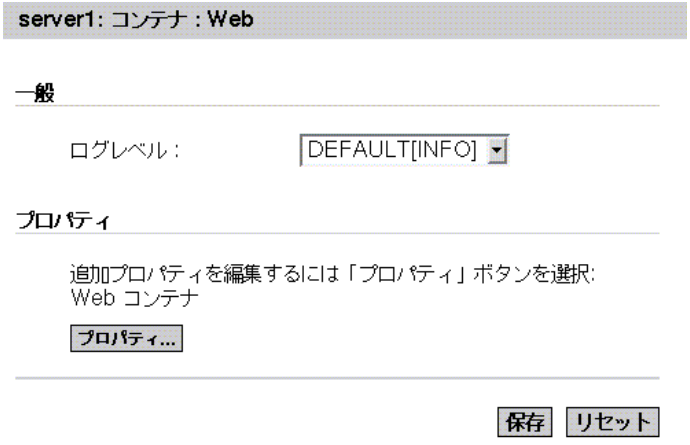
ログレベルを指定して、ログに記録するメッセージの種類を制御します。たとえば、ログレベル FATAL のメッセージだけを記録するように指定した場合、このレベルより上のレベルのメッセージは無視されます。明示的に指定されたログレベルで記録されるメッセージだけが、この値と比較されます。

明示的に指定されてないログレベルで記録されたメッセージは、無条件に記録されます。デフォルトでは、すべての警告、エラー、致命的なメッセージが記録されます。

Web コンテナのログレベルを設定するには、次の手順に従ってください。

1. 管理インタフェースの左側のペインで、Sun ONE Application Server インスタンスツリーを展開して、変更する Web コンテナ設定を探します。
2. 表示された J2EE コンテナの一覧から「Web コンテナ」を選択します。管理インタフェースの右側のペインに、Web コンテナのロギングのページが表示されます。

Web コンテナのロギング



3. 「ログレベル」ドロップダウンリストからログレベルを選択します。すべてのログレベルとその定義については、第 5 章「ログの使用」を参照してください。
4. 「保存」をクリックして設定を保存します。

Web コンテナの追加のプロパティを作成するには、「プロパティ」ボタンをクリックしてください。

EJB コンテナについて

Enterprise JavaBean コンテナは、Enterprise JavaBean を制御し、システムレベルの重要なサービスを提供する実行時環境です。EJB は、EJB コンテナ内で実行されるコンポーネントです。これらは、EJB サーバー内で順番に実行されます。Beans には、次のようなシステムレベルのサービスが提供されます。

- トランザクション管理
- セキュリティ
- ライフサイクル管理
- リモート接続
- データベースコネクションプーリング
- ネーミングサービス

Enterprise JavaBean は、ビジネスロジックを含む Java で記述されたサーバーコンポーネントです。EJB コンテナは、Bean へのリモートアクセス機能を提供します。EJB は常にコンテナのコンテキスト内で動作します。コンテナは、EJB とそれらを管理するサーバー間のリンクとして機能します。EJB コンテナによって、独自のコンポーネントや他の供給元に提供されたコンポーネントを使った分散アプリケーションを構築できます。

Sun ONE Application Server では、EJB コンテナを通して高レベルのトランザクション、状態管理、マルチスレッド、およびリソースプールラッパーを提供するので、管理者が低レベル API の詳細を理解する必要はありません。このコンテナは、EJB 2.0 仕様に規定されているすべての標準コンテナサービスに加えて、Sun ONE Application Server に固有のサービスも提供します。

コンテナは、非活性化および活性化プロセスを使って、Bean のアクティビティを管理し、スケーラビリティを確保します。

この節には次の項目があります。

- EJB コンテナの役割
- EJB コンテナの設定

EJB コンテナの役割

EJB コンテナは、次の標準サービスを提供します。

- 非活性化

  メモリから二次ストレージへ EJB を転送するプロセス。非活性化では、Bean を削除することなく Bean のリソースを解放できます。これによって、Bean は持続的になり、インスタンス化せずに再び呼び出すことができます。

- 活性化

  EJB を二次ストレージからメモリに転送するプロセス。コンテナ規約は EJB とそのコンテナ間の関係を規定しており、クライアントに対して完全に透過的です。この関係を次に示します。

  - ライフサイクル

    セッション Beans の場合は、`javax.ejb.SessionBean` インタフェースおよび `javax.ejb.SessionSynchronization` インタフェースが実装されています。エンティティ Beans の場合は、`javax.ejb.EntityBean` インタフェースが実装されています。メッセージ駆動型 Beans の場合は、`javax.ejb.MessageDriven` インタフェースが実装されています。

  - セッションコンテキスト

    コンテナは `javax.ejb.SessionContext` インタフェースを実装して、Bean インスタンス作成時にセッション Bean インスタンスにサービスおよび情報を渡します。

  - エンティティコンテキスト

    コンテナは `javax.ejb.EntityContext` インタフェースを実装して、Bean インスタンス作成時にエンティティ Bean にサービスおよび情報を渡します。

  - メッセージコンテキスト

    コンテナは `javax.ejb.MDBContext` インタフェースを実装して、Bean インスタンス作成時にメッセージ駆動型 Bean にサービスおよび情報を渡します。

  - 環境

    コンテナは `java.util.Properties` を実装し、これらのプロパティをそのコンテナの EJB で利用できるようにします。

  - サービス情報

    コンテナは、そのサービスを EJB で利用できるようにします。

Sun ONE Application Server サービスには、リモートアクセス、ネーミング、セキュリティ、並行処理、トランザクション制御、データベースアクセスなどがあります。

EJB コンテナの機能は次のとおりです。

- リモート接続を許可する実装オブジェクト (EJBObject) を作成する
- EJBObject の作成を許可するホーム実装オブジェクトを作成する
- クライアントがホームオブジェクトを検索できるように、ホーム実装オブジェクトをネーミングサービスにバインドする
- 認証されたクライアントのみが Bean メソッドを (EJBObject を介して) 呼び出すことができるようにする
- ビジネスメソッドが適切なトランザクションから呼び出されるようにする
- Beans のライフサイクルを管理する。次の方法で Beans のライフサイクルを管理する
  - Beans をプールする
  - 適切なコールバックメソッド (ejbActivate/ejbPassivate など) を呼び出す
  - アプリケーションによる接続の利用または再利用の効率化を図るため、データベース接続プールを管理する

実際の実装の詳細は、コンテナとその EJB 間の指定された標準インタフェースに基づいて、コンテナの一部となります。プラットフォーム固有の実装の詳細を把握したり、それを直接扱ったりする必要はありません。その代わりに、EJB 標準をサポートする任意のベンダーの製品とともに使用できる、一般的なタスク限定の EJB を作成できます。

Sun ONE Application Server で使用される EJB の型を理解しておくことをお勧めします。

Enterprise JavaBeans の種類

EJB は、次のいずれかを表すオブジェクトです。

- 特定のクライアントとのセッション。クライアントによって起動された複数のメソッドの状態を自動的に管理する
- 持続的なエンティティオブジェクト。複数のクライアントによって共有可能
- ステートレスサービス。メッセージ処理など

エンティティ Beans は、おもに、JDBC (Java Database Connectivity) API を使ったデータアクセス処理に使用されます。一方、セッション Beans は、一時的なアプリケーションオブジェクトを提供し、個々のビジネスタスクを実行します。EJB は 3 種類あります。次の各項目で説明します。

- セッション Beans について
- エンティティ Beans について
- メッセージ駆動型 Beans について

セッション Beans について

セッション Bean は、ビジネス規則または特定のクライアント要求のロジックを実装します。

セッション Beans は、一時的なオブジェクトおよびプロセス (単一のデータベースレコードの更新、これから編集する文書コピー、個々のクライアントに固有のビジネスオブジェクトなど) を表します。つまり、セッション Beans は、そのセッション Beans を作成するクライアントだけが使用するプライベートリソースです。これらのオブジェクトは単一のクライアントだけが使用できるため、セッション Beans は会話型ステートと呼ばれるクライアント固有のセッション情報を維持できます。

たとえば、EJB を作成して電子ショッピングカートをシミュレートするとします。ユーザーがアプリケーションにログインするたびに、アプリケーションはショッピングカートのセッション Bean を作成して、そのユーザーが購入したアイテムを保持します。ユーザーがログアウトするか、ショッピングを終了すると、セッション Bean は解放されます。

セッション Beans には次の特性があります。

- セッション Beans は単一のクライアントとの関連で実行される
- セッション Beans は比較的短命である
- セッション Beans ではサーバークラッシュに対する耐久性は保証されない
- セッション Beans は EJB コンテナのクラッシュ時には削除される
- セッション Beans はプロパティの設定値に従ってトランザクション管理を行う (オプション)
- セッション Beans は基になるデータベースで共有データを更新する (オプション)
- セッション Beans はステートレスとステートフルのいずれかになる

ステートレスのセッション Beans。ステートレスのセッション Beans は、限られた時間内で、特定のクライアントに必要なビジネスロジックの一時的な部分をカプセル化します。ステートレスのセッション Beans は、会話型ステートを維持しません。

ステートフルのセッション Beans。ステートフルのセッション Beans も一時的ですが、会話型ステートを使って、次のクライアント呼び出しまでコンテンツおよび値に関する情報を保持します。会話型ステートにより、Beans のコンテナはセッション Beans の状態情報を保持し、プログラム実行中に必要に応じてその状態を再現できます。

エンティティ Beans について

一般に、エンティティ Beans は、データベース内で直接保持される持続データ、または EIS (Enterprise Information System) アプリケーションを介してオブジェクトとしてアクセスされる持続データを表します。EJB および EJB コンテナをホストするサーバーは、同時にアクティブになっているエンティティ EJB にスケーラブルな実行時環境を提供します。

簡単なエンティティ Bean の例としては、データベーステーブルの 1 行を表すように定義され、各 Bean インスタンスが特定の行を表すものがあります。より複雑な例としては、データベース内の結合されたテーブルの複雑なビューを表すように設計されたものがあります。たとえば、各 Bean インスタンスが 1 つのショッピングカートの内容を表すエンティティ Bean などです。

エンティティ Beans には次の特性があります。

- エンティティ Beans は EIS リソース (通常はデータベース) 内のデータのオブジェクトビューを提供する
- エンティティ Beans にはすべてのユーザーがアクセス可能である
- エンティティ Beans はサーバークラッシュ後も透過的に存続する
- エンティティ Beans はコンテナ管理または Beans 管理のトランザクションを使用する

エンティティ Beans は、持続データをコンテナ管理または Bean 管理の持続性として表します。エンティティ Beans の持続性は、Bean またはコンテナによって管理できます。

Bean 管理の持続性。エンティティ Bean が独自の持続性を管理します。Bean 開発者が EJB クラスメソッドに持続コード (JDBC 呼び出しなど) を直接実装します。専用インタフェースを使うと、ダウンサイドでは移植性が失われる可能性があり、Bean が特定のデータベースに関連付けられるというリスクがあります。

コンテナ管理の持続性。エンティティ Bean の持続性がコンテナによって管理されます。コンテナは持続的状態を透過的に管理するので、Bean メソッドにデータアクセスコードを実装する必要はありません。この方法によって、簡単に実装できるだけでなく、特定のデータベースに関連付けずに完全に Bean を移植できます。

コンテナ管理の持続性を使用するエンティティ Bean は、Bean 管理の持続性を使用するエンティティ Bean のうち、コンテナによって自動生成されるものです。

エンティティ Beans の構築と使用に関する詳細については、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

メッセージ駆動型 Beans について

メッセージ駆動型 Beans は、J2EE アプリケーションでメッセージを非同期的に処理できる EJB です。メッセージ駆動型 Beans は、Java Message Service メッセージの着信によって動作します。

メッセージ駆動型 Bean インスタンスは、作成されてから削除されるまで、メッセージ駆動型 Bean コンテナ内にあります。コンテナは、メッセージ駆動型 Bean インスタンスのセキュリティ、トランザクション、メッセージの並行処理、ライフサイクル管理など、メッセージ駆動型 Bean に関するさまざまなサービスを提供します。EJB および EJB コンテナを管理するサーバーは、同時にアクティブになっているメッセージ駆動型 Beans にスケーラブルな実行時環境を提供します。

J2EE 1.3 プラットフォームの Java Message Service API には、次のことが指定されています。

- アプリケーションクライアント、EJB コンポーネント、Web コンポーネントは、Java Message Service メッセージを送信または同期受信できる。さらに、アプリケーションクライアントは、Java Message Service メッセージを非同期で使用できる
- メッセージ駆動型 Beans は、メッセージの非同期の消費を可能にする。Java Message Service プロバイダは、オプションで、メッセージ駆動型 Beans によるメッセージの並行処理を実装できる

メッセージ駆動型 Bean はステートレスのサービスを表します。これは、完全に匿名で、クライアントに識別情報を公開しない非同期メッセージコンシューマです。メッセージ駆動型 Bean は、ホームインタフェースもコンポーネントインタフェースも持ちません。クライアントは、メッセージ駆動型 Bean クラスが MessageListener である Java Message Service の送信先 (キューまたはトピック) にメッセージを送信することにより、Java Message Service を介してメッセージ駆動型 Beans にアクセスします。

メッセージ駆動型 Beans のみが、非同期でメッセージを受け取ることができます。セッション Beans やエンティティ Beans は、Java Message Service の `MessageListener` にはなれません。

メッセージ駆動型 Beans には次の特性があります。

- 1 つのクライアントメッセージを受信すると実行される
- 非同期的に呼び出される
- 比較的短命である
- 直接データベース内の共有データを表すわけではないが、このデータのアクセスおよび更新は可能である
- EJB サーバーのクラッシュ時に削除される
- ステートレスである

- トランザクション対応 (オプション)

EJB コンテナの設定

EJB コンテナのログレベルを設定できます。また、監視を有効にすることも可能です。EJB コンテナは、EJB と MDB の両方を処理します。コンテナが管理する EJB と MDB については、管理インタフェースを使って設定できます。この節には次の項目があります。

- 一般設定
- EJB 設定
- MDB プールの設定

一般設定

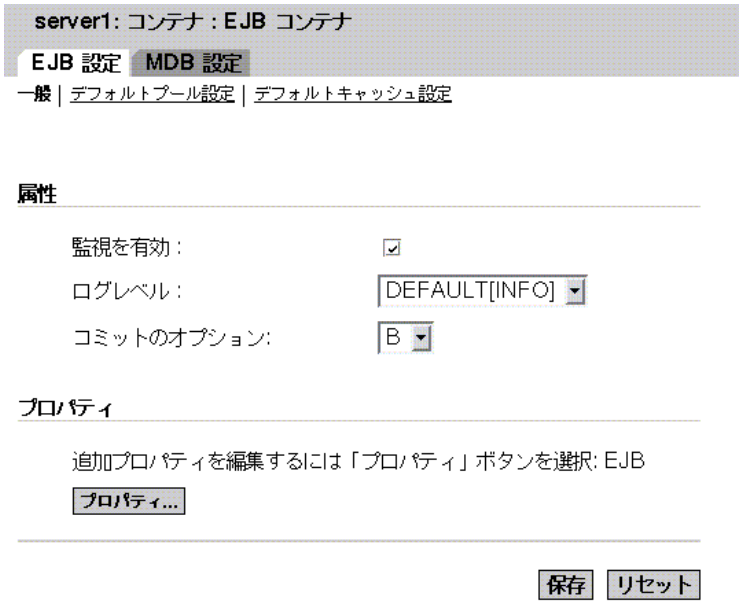
EJB コンテナに関して次の項目を設定します。

- ログ
- 監視
- トランザクション属性

EJB コンテナのログレベルの設定、監視の有効化、およびトランザクション属性の設定を行うには、次の手順に従います。

1. Sun ONE Application Server の左側のペインで、アプリケーションサーバーインスタンスツリーを展開して、変更する EJB コンテナ設定を探します。
2. コンテナのリストを表示し、ここから「EJB コンテナ」を選択します。管理インタフェースの右側のペインに、次のような EJB コンテナの一般設定用のウィンドウが表示されます。

EJB コンテナの一般設定



3. 「監視を有効」のチェックボックスにチェックマークをつけて、EJB コンテナの監視を有効にします。これで、指定した Sun ONE Application Server インスタンスの EJB コンテナの監視が有効になりました。EJB コンテナの監視可能な項目については、「EJB コンテナ統計情報の監視」の表を参照してください。

4. 「ログレベル」ドロップダウンリストからログレベルを選択します。すべてのログレベルとその定義については、第 5 章「ログの使用」を参照してください。ログレベルを指定して、ログに記録するメッセージの種類を制御します。たとえば、ログレベル `FATAL` のメッセージだけを記録するように指定した場合、このレベルより上のレベルのメッセージは無視されます。明示的に指定されたログレベルで記録されるメッセージだけが、この値と比較されます。

   明示的に指定されてないログレベルで記録されたメッセージは、無条件に記録されます。デフォルトでは、すべての警告、エラー、致命的なメッセージが記録されます。

5. 「コミットのオプション」ドロップダウンリストで、EJB コンテナに使用する「コミットのオプション」を選択します。

   トランザクションの終了方法には、コミットまたはロールバックによる 2 種類があります。トランザクションがコミットすると、ステートメントによって変更されたデータは保存されます。Enterprise JavaBean の設計時には、コミットがコンテナ管理のトランザクションと Bean 管理のトランザクションのどちらであるかを決めます。ドロップダウンリストの選択肢では、「B」は Bean 管理のコミットを、「C」はコンテナ管理のコミットを示しています。

6. 「プロパティ」ボタンをクリックして、EJB コンテナの新しいプロパティを作成します。

7. 「了解」をクリックして、設定を保存します。

監視可能な EJB コンテナの属性を次の表に示します。

EJB コンテナ統計情報の監視

統計情報の名前	データ型、単位	値の範囲	コメント
minBeansInPool	整数	0 ～ MAXINT	プール内の Beans の最小数 (ステートレスのセッション Beans に適用)
initialBeansInPool	整数	0 ～ MAXINT	プール内の Beans の初期数 (ステートレスのセッション Beans に適用)
maxBeansInPool	整数	0 ～ MAXINT	プール内の Beans の最大数 (ステートレスのセッション Beans に適用)
beanIdleTimeoutInSeconds	整数	0-MAXLONG	Bean が削除されるまでのアイドルタイムアウト (秒数)。
numBeansCreated	整数	0 ～ MAXINT	これまでに作成された Beans 数
numBeansDestroyed	整数	0 ～ MAXINT	これまでに削除された Beans 数
numThreadsWaitaing	整数	0 ～ MAXINT	利用可能な Beans を待機しているスレッド数
numBeansInPool	整数	0 ～ MAXINT	プール内の利用可能な Beans 数 (この値が 0 より大きい場合、numThreadsWaitaing は必ず 0)
maxBeansInCache	整数	0 ～ MAXINT	キャッシュ内の Beans の最大数 (エンティティ Beans およびステートフル Bean に適用)
minBeansInCache	整数	0 ～ MAXINT	キャッシュ内の Beans の最小数 (エンティティ Beans およびステートフル Beans に適用)

EJB コンテナ統計情報の監視 (続き)

統計情報の名前	データ型、単位	値の範囲	コメント
cacheFaultsPercentage	倍精度浮動小数点数		バックアップストアからのアクティブ化によるキャッシュミスの回数

EJB 設定

管理インタフェースを使うと、EJB コンテナに管理される EJB のデフォルトのプールと Bean キャッシュに関する設定を行うことができます。次の各項目で説明します。

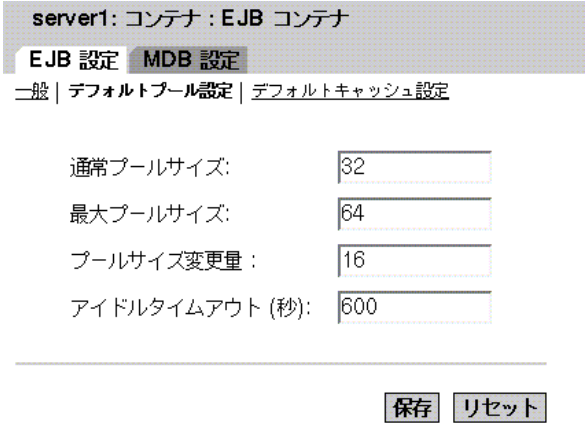
- EJB プールを設定するには
- EJB キャッシュを設定するには

EJB プールを設定するには

EJB プール設定を行うには、次の手順に従います。

1. Sun ONE Application Server の左側のペインで、アプリケーションサーバーインスタンスツリーを展開して、変更する EJB 設定を探します。
2. コンテナのリストを表示し、ここから「EJB コンテナ」を選択します。管理インタフェースの右側のペインに、次のような EJB プールの設定用のウィンドウが表示されます。

EJB プールの設定



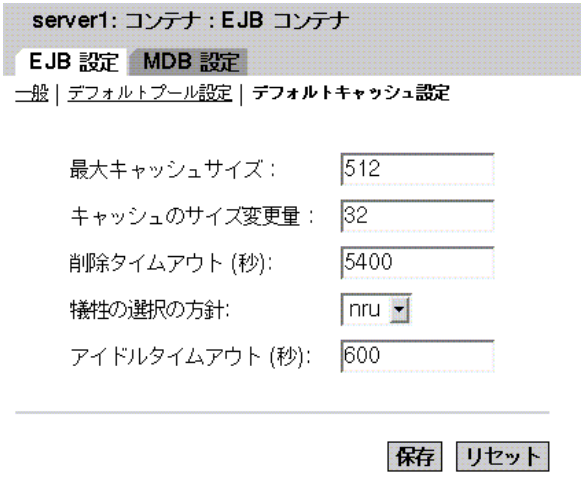
3. 「通常プールサイズ」フィールドに、プール内の Beans の最小数を指定します。これは、ステートレスのセッション Beans に適用されます。
4. 「最大プールサイズ」ドロップダウンリストに、任意の時点でプール内に格納できる Beans の最大数を指定します。この設定は、ステートレスのセッション Beans に適用されます。
5. 「プールサイズ変更量」フィールドに、Beans が idle-timeout-in-seconds タグで指定された時間を超えてアイドル状態であった場合にプールから削除される Beans 数を指定します。
6. 「アイドルタイムアウト (秒)」フィールドに、Bean がアイドル状態で残ることができる期間を秒単位で指定します。アイドルタイムアウトの期間が経過してもアイドル状態のままである Bean は削除されます。
7. 「保存」をクリックして変更を保存します。

EJB キャッシュを設定するには

EJB キャッシュ設定を行うには、次の手順に従います。

1. Sun ONE Application Server の左側のペインで、EJB 設定を変更するアプリケーションサーバーインスタンスツリーを展開します。
2. コンテナのリストを表示し、ここから「EJB コンテナ」を選択します。管理インタフェースの右側のペインに、次のような EJB キャッシュの設定用のウィンドウが表示されます。

EJB キャッシュの設定



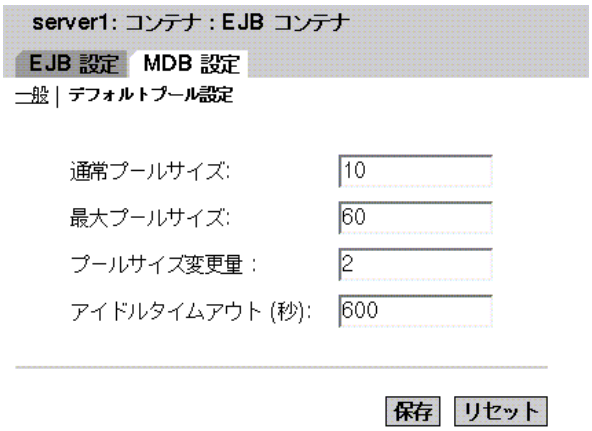
3. 「最大キャッシュサイズ」フィールドに、キャッシュに保持される Beans の最大数を指定します。この属性のデフォルト値は、`idle-timeout-in-seconds` 属性に指定されています。
4. 「キャッシュサイズ変更量」フィールドに、プール内の Beans 数が `Max Cache Size` 属性に指定された量を超えた場合に削除する Beans 数を指定します。
5. 「削除タイムアウト (秒)」フィールドに、バックアップストア内でアイドル状態の Bean が非活性化されたままでいられる時間を指定します。Bean がクライアントからアクセスされない期間が `removal-timeout-in-seconds` 属性に指定された値を経過すると、Bean はバックアップストアから削除されるため、それ以降クライアントはアクセスできなくなります。
6. 「犠牲の選択の方針」ドロップダウンリストで、プールからの削除時に犠牲 (削除対象) となる Beans を選択するためのアルゴリズムを指定します。
7. 「アイドルタイムアウト (秒)」フィールドに、Bean がキャッシュ内でアイドル状態でいられる期間を指定します。この期間を過ぎると Bean は削除されます。Bean をアイドルバックアップストアに非活性化の状態で残す期間は、`removal-timeout-in-seconds` パラメータによって制御されます。
8. 「保存」をクリックして変更を保存します。

MDB プールの設定

管理インタフェースを使うと、EJB コンテナが管理する MDB のデフォルトのプール設定を行うことができます。MDB のデフォルトのプール設定を行うには、次の手順に従います。

1. 管理インタフェースの左側のペインで、変更する MDB コンテナ設定を含む Sun ONE Application Server インスタンスツリーを開きます。
2. コンテナのリストを表示し、ここから「EJB コンテナ」を選択します。管理インタフェースの右側のペインに、次のような MDB プールの設定用のウィンドウが表示されます。

MDB プールの設定



3. 「MDB 設定」をクリックします。「通常プールサイズ」テキストフィールドに、プール内の Beans の最小数を指定します。これは、ステートレスのセッション Beans に適用されます。
4. 「最大プールサイズ」フィールドに、任意の時点でプール内に格納できる Beans の最大数を指定します。
5. 「プールサイズ変更量」フィールドに、Beans が `idle-timeout-in-seconds` タグで指定された時間を超えてアイドル状態であった場合にプールから削除される Beans 数を指定します。
6. 「アイドルタイムアウト (秒)」フィールドに、Bean がアイドル状態で残ることができる期間を秒単位で指定します。アイドルタイムアウトの期間が経過してもアイドル状態のままである Bean は削除されます。
7. 「保存」をクリックして設定を保存します。

第 9 章

トランザクションサービスの使用

トランザクションはビジネスにとって不可欠な部分です。一般的なビジネストランザクションには、2 つ以上の関連する組織の間での資産譲渡などがあります。通常、厳密な記録が 1 つ以上のデータベースに保存されます。これらの情報はビジネスオペレーションにとって重要な意味を持つため、正確さ、即時性と信頼性が必要となります。トランザクション処理の開発は、初心者のプログラマには困難です。J2EE プラットフォームは、ある程度の抽象化によって、信頼できるトランザクション処理アプリケーションの開発を容易にしています。この章では、J2EE トランザクションと、Sun ONE Application Server のトランザクションサポートについて説明します。

この章では、一般的な Java トランザクションと、Sun ONE Application Server に組み込まれている固有のトランザクションサポートについて説明します。

この章では次のトピックについて説明します。

- トランザクションとは
- J2EE のトランザクション
- トランザクションリソースマネージャ
- ローカルトランザクションと分散トランザクション
- コンテナ管理トランザクション
- Bean 管理トランザクション
- トランザクションサービスの管理

トランザクションとは

ビジネストランザクションをエミュレートするには、プログラムによって複数のステップを実行する必要があります。たとえば、金融処理のプログラムでは、次の擬似コードで示されるような複数のステップを実行して、当座預金から普通預金に資金を移動することがあります。

```
begin transaction
debit checking account
credit savings account
update history log
commit transaction
```

上記の擬似コードでは、begin ステートメントと commit ステートメントによって、トランザクションの境界をマーク付けしています。このトランザクションを完了するには、3 つのステップがすべて成功することが必要です。3 つのステップのすべてが成功しなかった場合は、データの整合性が失われる可能性があります。

このような保証を原子性 (atomicity) と呼びます。トランザクションの終了方法には、コミット (commit) またはロールバック (rollback) による 2 種類があります。トランザクションのコミット時には、トランザクション境界内のステートメントによる変更は永久に保存されます。変更内容は「永続的」です。つまり、その後にシステム障害が発生した場合でも存続します。トランザクション内のいずれかのステートメントが失敗した場合、トランザクションはロールバックします。このとき、その時点までにトランザクションで実行されたすべてのステートメントによる効果を元に戻します。たとえば、上記の擬似コードで、ディスクドライブが credit のステップでクラッシュした場合、トランザクションはロールバックし、debit ステートメントで変更されたデータを元に戻します。

トランザクションが失敗した場合でも、トランザクションアカウントが保たれているため、データの整合性は保たれます このようなトランザクションの動作を、トランザクションの「整合性」と呼びます。

トランザクションサービスは、「遮断」も提供します。つまり、トランザクションがコミットまたはロールバックされるまで、他のアプリケーションやスレッドから、トランザクションのフェーズを調べることができません。トランザクションがコミットされた後は、アプリケーションやスレッドはコミットされたトランザクションを安全に調べることができます。

J2EE のトランザクション

J2EE のトランザクション処理には、トランザクションマネージャ、アプリケーションサーバー、リソースマネージャ、リソースアダプタ、およびユーザーアプリケーションの 5 つの関係要素が含まれます。これらの各エンティティは、次に説明する API や機能を実装することにより、信頼性のあるトランザクション処理を実現しています。

- トランザクションマネージャは、トランザクション境界、トランザクションリソース管理、同期化、およびトランザクションコンテキスト伝達のサポートに必要なサービスと管理機能を提供する
- アプリケーションサーバーは、トランザクション状態管理を含むアプリケーションランタイム環境のサポートに必要なインフラストラクチャを提供する
- リソースマネージャは、リソースアダプタを通じて、リソースへのアプリケーションアクセスを提供する。トランザクションマネージャがトランザクションの関連付け、完了およびリカバリ作業を行う際に使用する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によって、リソースマネージャは分散トランザクション処理を実行する。このようなリソースマネージャの例としては、リレーショナルデータベースサーバーがある
- リソースアダプタはシステムレベルのソフトウェアライブラリで、リソースマネージャへ接続するためにアプリケーションサーバーまたはクライアントが使用する。通常、リソースアダプタはリソースマネージャに固有となる。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る。このようなリソースアダプタの例としては、JDBC ドライバがある
- J2EE 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従来のユーザーアプリケーションは、JNDI を使って、トランザクションデータソースおよびトランザクションマネージャ (オプション) を検索する。宣言による EJB のトランザクション属性、または明示的なプログラムによるトランザクション境界を使用する

リソースマネージャとリソースアダプタには密接な関係があるため、これらの用語は同じ意味で使われることがあります。

トランザクションリソースマネージャ

J2EE トランザクションでは、次のトランザクションリソースマネージャがサポートされています。

- データベース
- JMS プロバイダ
- J2EE コネクタ

データベース

J2EE アプリケーションでもっとも多く使われるトランザクションリソースマネージャはデータベースです。JDBC は、J2EE コンポーネントがデータベースへのアクセスで使う API です。データベースリソースは、JDBC リソースとして設定されます。JDBC リソースは、リソースマネージャである JDBC ドライバによって管理されます。JDBC ドライバは、ローカルトランザクションまたはグローバルトランザクションのサポートを提供します。場合によっては、これら両方のサポートを提供します。

Sun ONE Application Server は、さまざまな J2EE コンポーネントから JDBC およびトランザクションの使用をサポートしています。JDBC リソースの登録方法と設定方法の詳細については、263 ページの「JDBC リソースについて」を参照してください。アプリケーションサーバーは、トランザクションの継続 (トランザクションの起動と、複数のアプリケーションコンポーネントからのデータベースアクセス) に対して責任を持ちます。たとえば、サーブレットはトランザクションを起動し、データベースにアクセスし、同じトランザクションの一部として同じデータベースにアクセスするエンタープライズ Bean を呼び出し、最後にトランザクションをコミットします。

JMS プロバイダ

JMS は Java Message Service の略語です。JMS プロバイダは、メッセージブローカサービスに対する J2EE 用語です。JMS API は、トランザクションによるアプリケーション間の信頼性できるメッセージ交換を提供します。J2EE では、トランザクションの JMS データソースをサポートする機能が必要とされています。JMS リソースと JDBC リソースは、同じトランザクションに使用できます。

Sun ONE Application Server は、Sun ONE メッセージキュー、完全機能の JMS プロバイダ、および対応するトランザクションリソースマネージャと統合されています。そのため、Sun ONE Application Server では、サーブレット、JSP ページ、およびエンタープライズ Beans からのトランザクションによる JMS アクセスが可能です。また、Sun ONE Application Server では、サードパーティの JMS プロバイダを使うこともできます。詳細については、第 11 章「JMS サービスの使用」を参照してください。

J2EE コネクタ

Sun ONE Application Server は、XATransaction モードをトランザクションリソースマネージャとして使うリソースアダプタをサポートしています。プラットフォームが、サーブレット、JSP ページ、およびエンタープライズ Beans からのトランザクションによるリソースアダプタへのアクセスを有効にしていることが必要です。単一のトランザクション内で、複数のアプリケーションコンポーネントからリソースアダプタへアクセスすることができます。たとえば、サーブレットはトランザクションを起動し、リソースアダプタにアクセスし、同じトランザクションの一部としてリソースアダプタにアクセスするエンタープライズ Bean を呼び出し、最後にトランザクションをコミットします。

ローカルトランザクションと分散トランザクション

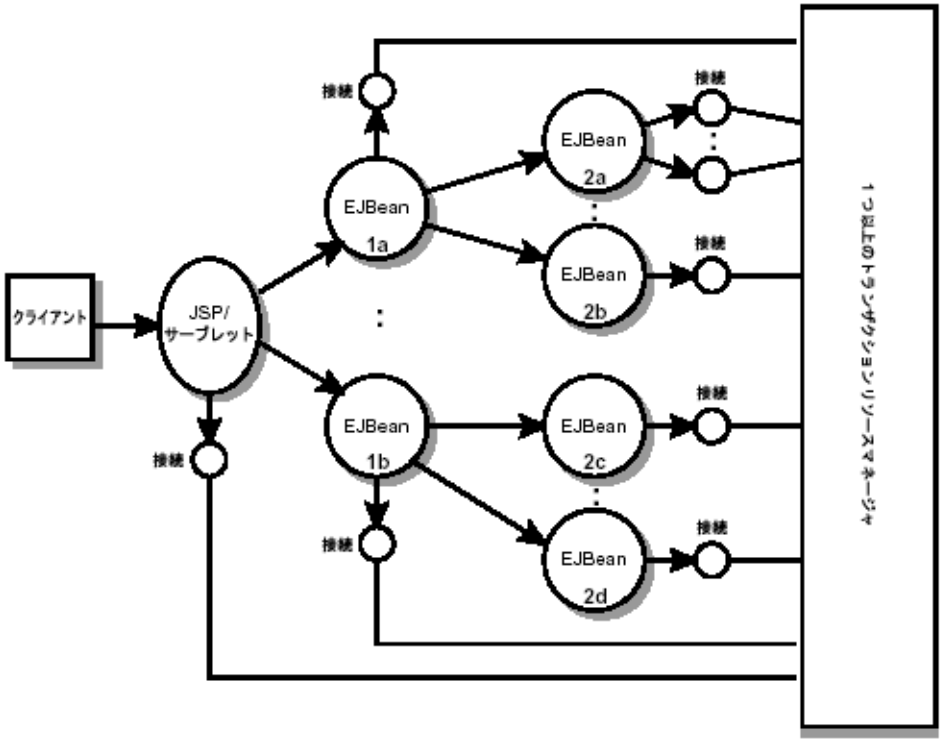
単一のリソースだけを対象とする場合は、ローカルトランザクションを使って完了できます。ローカルトランザクションでは、関係するすべてのアプリケーションコンポーネントが 1 つのプロセスで実行することも必要です。複数のリソースを対象とするトランザクション、つまり複数の関連プロセスを含むトランザクションは、分散トランザクションまたはグローバルトランザクションとなります。ローカルトランザクションの最適化では、最適化に固有のリソースマネージャが使われ、J2EE アプリケーションに透過であることが重要となります。

トランザクションのタイプは、主に関連するリソースマネージャに実装されるインタフェースにより決まります。たとえば、`javax.sql.DataSource` インタフェースを実装する JDBC データソースは、ローカルトランザクションに属します。`javax.sql.XADataSource` を実装するデータソースは、グローバルトランザクションに関係します。JDBC リソースには両方のインタフェースを実装するものもあります。そのような JDBC リソースを Sun ONE Application Server で実装するときには、Sun ONE Application Server に追加の設定を行い、このリソースで優先する機能を指定する必要があります。

ローカルトランザクションは、グローバルトランザクションに比べて単純で効率的です。ただし、変換対象のデータが複数のデータソースに分散している場合、ローカルトランザクションは不適切です。トランザクションで使用されるデータソースの数を予測することが不可能な場合もあります。そのため、実際にはグローバルトランザクションがよく使用されます。グローバルトランザクションでは、パフォーマンスを向上させるための最適化を行うことができます。

J2EE は、トランザクション内の複数のエンタープライズ Beans にアクセスするサーブレットや JSP の任意の組み合わせを構成するようなトランザクションアプリケーションをサポートしています。各コンポーネントは、1 つ以上の接続を確立して、1 つ以上のトランザクションリソースマネージャにアクセスします。次の図では、呼び出しツリーの開始点は、複数のエンタープライズ Beans にアクセスするサーブレットまたは JSP です。これらのエンタープライズ Beans は、さらに他のエンタープライズ Beans にアクセスします。コンポーネントは、接続を介してリソースマネージャにアクセスします。

トランザクションでリソースにアクセスする J2EE コンポーネント



たとえば、アプリケーションでは、単一のトランザクションで上の図のすべてのコンポーネントがリソースにアクセスすることが必要な場合があります。アプリケーションサーバーのプロバイダは、そのようなシナリオをサポートするトランザクションの機能を提供する必要があります。

J2EE のトランザクション管理は、フラットなトランザクションをサポートしています。フラットなトランザクションでは、子の (入れ子になっている) トランザクションを持つことはできません。

分散トランザクションでは、トランザクションの回復が大切です。重要なポイントでリソースにアクセスできなくなった場合、またはその他の回復不可能なエラーが発生した場合、分散トランザクションの状態が不明になることがあります。未完了のトランザクションの自動または手動による回復は、Sun ONE Application Server の重要な機能です。管理インタフェースを使うと、自動トランザクション回復を有効にできます。トランザクション回復の制御方法については、233 ページの「トランザクションサービスの管理」を参照してください。

接続 (ここではリソースと同義語) は、共有可能または共有不可能のいずれかとしてマーク付けできます。接続を非共有として使用する J2EE アプリケーションコンポーネントは、有効な配備情報を提供して、接続がコンテナに共有されないようにする必要があります。これが必要になる例としては、セキュリティ属性、遮断レベル、文字設定、およびローカライズ設定の変更時などがあります。

コンテナは、共有不可能とマーク付けされた接続を共有できません。接続が共有不可能とマーク付けされていない場合は、接続は実際に共有されているかどうかに関わらず、アプリケーションに透過であることが必要です。

J2EE アプリケーションコンポーネントは、オプションの配備記述子要素 `res-sharing-scope` を使って、リソースマネージャへの接続が共有可能であるかどうかを示す場合があります。配備情報が提供されていない場合、コンテナは接続が共有可能であると仮定します。J2EE アプリケーションコンポーネントは、接続オブジェクトをキャッシュし、複数のトランザクションで再利用することがあります。接続の共有を提供するコンテナは、ディスパッチ時にキャッシュされた接続オブジェクトを透過的に切り替えて、正しいトランザクション範囲で適切な共有接続を指す必要があります。

エンタープライズ Bean アプリケーションの設計時には、開発者は境界をどのように指定するかを決める必要があります。

コンテナ管理トランザクション

コンテナ管理トランザクションを使用するエンタープライズ Bean では、EJB コンテナがトランザクションの境界を設定します。セッション、エンティティ、またはメッセージ駆動型の任意のタイプのエンタープライズ Bean を使用するコンテナ管理トランザクションを使うことができます。コンテナ管理トランザクションを使用すると、エンタープライズ Bean コードでトランザクションの境界を明示的に設定しないので、開発作業が簡素化されます。コードにはトランザクションを開始および終了するステートメントを記述しません。

通常、エンタープライズ Bean メソッドが起動する直前に、コンテナはトランザクションを開始します。メソッドが終了する直前に、コンテナはトランザクションをコミットします。各メソッドを、1 つのトランザクションに関連付けることができます。ネストされたトランザクションまたは複数のトランザクションを 1 つのメソッド内に含めることはできません。

コンテナ管理トランザクションでは、すべてのメソッドをトランザクションに関連付ける必要はありません。Bean の配備時に、トランザクション属性を設定することによって、Bean のどのメソッドをトランザクションと関連付けるかを指定することができます。

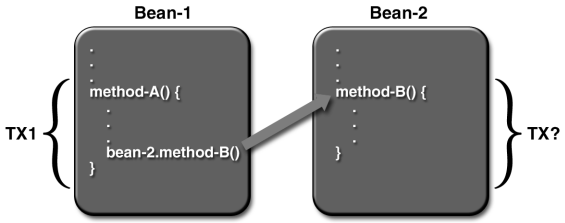
この節には次の項目があります。

- トランザクション属性
- トランザクション属性の設定
- コンテナ管理トランザクションのロールバック
- セッション Beans のインスタンス変数の同期化
- コンテナ管理トランザクションで使用できないメソッド

トランザクション属性

トランザクション属性は、トランザクションの範囲を制御します。次の図で、範囲の制御が重要である理由を示します。method-A はトランザクションを開始し、Bean-2 の method-B を呼び出します。このとき、method-B が、method-A の起動したトランザクションの範囲内で動作するか、新しいトランザクションで動作するかは、method-B のトランザクション属性によって決まります。

トランザクション属性



トランザクション属性には、次の値のいずれかを設定できます。

- Required
- RequiresNew
- Mandatory
- NotSupported
- Supports
- Never

Required

クライアントがトランザクション内で動作中にエンタープライズ Bean のメソッドを呼び出した場合、そのメソッドはクライアントのトランザクション内で動作します。クライアントがトランザクションに関連付けられていない場合、コンテナは、新しいトランザクションを開始してからメソッドを実行します。

`Required` 属性は、ほとんどのトランザクションで使用できます。したがって、デフォルト値として、少なくとも開発の初期段階ではこの値を使用するとよいでしょう。トランザクション属性は宣言型であるため、あとで容易に変更できます。

RequiresNew

クライアントがトランザクション内で動作中にエンタープライズ Bean のメソッドを呼び出した場合、コンテナは、次の処理を実行します。

- クライアントのトランザクションを中断する
- 新しいトランザクションを開始する
- 呼び出しをメソッドに委任する
- メソッドの完了後、クライアントのトランザクションを再開する

クライアントがトランザクションに関連付けられていない場合、コンテナは、新しいトランザクションを開始してからメソッドを実行します。

メソッドを常に新しいトランザクション内で動作させる必要がある場合は、RequiresNew 属性を使用します。

Mandatory

クライアントがトランザクション内で動作中にエンタープライズ Bean のメソッドを呼び出した場合、そのメソッドはクライアントのトランザクション内で動作します。クライアントがトランザクションに関連付けられていない場合、コンテナは TransactionRequiredException をスローします。

エンタープライズ Bean のメソッドでクライアントのトランザクションを使用する必要がある場合は、Mandatory 属性を使用します。

NotSupported

クライアントがトランザクション内で動作中にエンタープライズ Bean のメソッドを呼び出した場合、コンテナは、クライアントのトランザクションを中断してからメソッドを呼び出します。メソッドの完了後、コンテナはクライアントのトランザクションを再開します。

クライアントがトランザクションに関連付けられていない場合、コンテナは新しいトランザクションを開始せずに、メソッドを実行します。

トランザクションを必要としないメソッドでは NotSupported 属性を使用してください。トランザクションはオーバーヘッドを伴うので、この属性によりパフォーマンスを高めることができます。

Supports

クライアントがトランザクション内で動作中にエンタープライズ Bean のメソッドを呼び出した場合、そのメソッドはクライアントのトランザクション内で動作します。クライアントがトランザクションに関連付けられていない場合、コンテナは新しいトランザクションを開始せずに、メソッドを実行します。

メソッドのトランザクション動作は大きく異なるので、Supports 属性を使用するには注意が必要です。

Never

クライアントがトランザクション内で動作中にエンタープライズ Bean のメソッドを呼び出した場合、コンテナは RemoteException をスローします。クライアントがトランザクションに関連付けられていない場合、コンテナは新しいトランザクションを開始せずに、メソッドを実行します。

属性のまとめ

次の表は、各トランザクション属性の効果をまとめたものです。T1 および T2 のトランザクションはコンテナによって制御されます。T1 トランザクションは、エンタープライズ Bean 内でメソッドを呼び出すクライアントに関連付けられます。ほとんどの場合、クライアントは別のエンタープライズ Bean になります。T2 トランザクションは、メソッドの実行直前に、コンテナによって開始されます。

最後の列 (ビジネスメソッドのトランザクション) の「なし」は、そのビジネスメソッドが、コンテナによって制御されるトランザクション内で実行されないことを意味しています。ただし、ビジネスメソッドなどでのデータベース呼び出しは、DBMS のトランザクションマネージャによって制御される場合があります。

トランザクション属性

トランザクション属性	クライアントのトランザクション	ビジネスメソッドのトランザクション
Required	なし	T2
	T1	T1
RequiresNew	なし	T2
	T1	T2
Mandatory	なし	エラー
	T1	T1
NotSupported	なし	なし
	T1	なし
Supports	なし	なし
	T1	T1

トランザクション属性の設定

トランザクション属性は配備記述子内に保存されるため、エンタープライズ Bean 作成時、アプリケーションのアセンブリ時、配備時など、J2EE アプリケーション開発中のいくつかの段階で変更が可能です。ただし、開発者は、Bean の作成時にトランザクション属性を指定する必要があります。アプリケーション開発者がコンポーネントを大規模なアプリケーションにアセンブリするときだけ、この属性を変更する必要があります。J2EE アプリケーションを配備する個々の担当者が、トランザクション属性を指定する必要はありません。

エンタープライズ Bean 全体、または個々のメソッドのトランザクション属性を指定できます。メソッドと Bean に別々の属性を指定した場合は、メソッドの属性が優先されます。個々のメソッドの属性を指定する場合は、Bean のタイプによって必要条件が異なります。セッション Beans では、ビジネスメソッドの属性を定義する必要がありますが、create メソッドの属性を指定できません。エンティティ Beans では、ビジネスメソッド、create メソッド、remove メソッド、および検索メソッドのトランザクション属性が必要です。メッセージ駆動型 Beans では、`onMessage` メソッドのトランザクション属性 (`Required` か `NotSupported` のどちらか) が必要となります。

コンテナ管理トランザクションのロールバック

コンテナ管理トランザクションのロールバックには、次の 2 つの方法があります。1 つ目の方法は、システム例外がスローされた場合に、コンテナが自動的にトランザクションをロールバックする方法です。2 番目の方法は、`EJBContext` インタフェースの `setRollbackOnly` メソッドを呼び出すことによって、Bean メソッドがコンテナにトランザクションのロールバックを指示する方法です。Bean がアプリケーション例外をスローした場合は、自動的なロールバックは行われませんが、`setRollbackOnly` の呼び出しによってロールバックを起動できます。

次の例では、`BankEJB` の `transferToSaving` メソッドが `setRollbackOnly` メソッドを呼びだしています。バランスチェックが負の場合、`transferToSaving` は `setRollBackOnly` を呼び出し、アプリケーション例外 (`InsufficientBalanceException`) をスローします。`updateChecking` メソッドと `updateSaving` メソッドはデータベーステーブルを更新します。更新に失敗すると、これらのメソッドは `SQLException` をスローし、`transferToSaving` メソッドは `EJBException` をスローします。`EJBException` はシステム例外であるため、コンテナが自動的にトランザクションをロールバックします。`transferToSaving` メソッドのコードを次に示します。

```
public void transferToSaving(double amount) throws
   InsufficientBalanceException  {

   checkingBalance -= amount;
   savingBalance += amount;

if (checkingBalance < 0.00) {
context.setRollbackOnly();

throw new InsufficientBalanceException();
}
   try {
      updateChecking(checkingBalance);

   updateSaving(savingBalance);
   } catch (SQLException ex) {
      throw new EJBException
         ("Transaction failed due to SQLException: "
         + ex.getMessage());
   }
}
```

コンテナは、トランザクションのロールバック時に、トランザクション内の SQL 呼び出しによって加えられたデータ変更を元に戻します。ただし、エンティティ Beans の場合のみ、コンテナはインスタンス変数に加えられた変更を元に戻します。その場合は、エンティティ Beans の `ejbLoad` メソッドを自動的に呼び出して、データベースからインスタンス変数を読み込みます。セッション Beans では、ロールバックが発生したときに、トランザクション内で変更されたすべてのインスタンス変数を明示的にリセットする必要があります。セッション Beans のインスタンスをリセットするもっとも簡単な方法は、`SessionSynchronization` インタフェースを実装することです。

コマンド行インタフェースを通じてトランザクション ID を渡すことにより、トランザクションをロールバックすることもできます。詳細については、236 ページの「コマンド行インタフェースを使用したトランザクションの管理」を参照してください。

セッション Beans のインスタンス変数の同期化

オプションで設定できる SessionSynchronization インタフェースでは、インスタンス変数とデータベース内の対応する値の同期をとることができます。コンテナは、トランザクションの主要ステージごとに SessionSynchronization のメソッド (afterBegin、beforeCompletion、および afterCompletion) を呼び出します。

afterBegin メソッドは、新しいトランザクションが開始されたことをインスタンスに知らせます。コンテナは、トランザクション内で最初のビジネスメソッドを呼び出す前に afterBegin を呼び出します。afterBegin メソッドは、データベースからインスタンスを読み込むのに適しています。たとえば、BankBean クラスは、次のように afterBegin メソッドで checkingBalance 変数と savingBalance 変数を読み込みます。

```
public void afterBegin() {

   System.out.println("afterBegin()");
   try {
      checkingBalance = selectChecking();
      savingBalance = selectSaving();
   } catch (SQLException ex) {
      throw new EJBException("afterBegin Exception: " +
         ex.getMessage());
   }
}
```

コンテナは、ビジネスメソッドの完了後、トランザクションをコミットする直前に beforeCompletion メソッドを起動します。beforeCompletion メソッドは、セッション Bean が (setRollbackOnly を呼び出して) トランザクションをロールバックする最後の機会です。インスタンス変数の値によってデータベースがまだ更新されていない場合、セッション Bean は beforeCompletion メソッドでその処理を実行できます。

afterCompletion メソッドは、トランザクションが完了したことを示します。このメソッドは 1 つのブール型パラメータを持ち、その値は、トランザクションがコミットされた場合は true、ロールバックされた場合は false となります。ロールバックが発生した場合、セッション Bean は、afterCompletion メソッドでデータベースに基づいてインスタンス変数を更新できます。

```
public void afterCompletion(boolean committed) {

   System.out.println("afterCompletion:" + committed);
   if (committed == false) {
      try {
          checkingBalance = selectChecking();
          savingBalance = selectSaving();
```

```
        } catch (SQLException ex) {
            throw new EJBException("afterCompletion SQLException:
            " + ex.getMessage());
}
}
}
```

コンテナ管理トランザクションで使用できないメソッド

コンテナが設定するトランザクション境界を侵害する可能性のあるメソッドを呼び出すことはできません。禁止されているメソッドは、次のとおりです。

- `java.sql.Connection` のコミットメソッド、`setAutoCommit` メソッド、およびロールバックメソッド
- `javax.ejb.EJBContext` の `getUserTransaction` メソッド
- `javax.transaction.UserTransaction` のすべてのメソッド

ただし、Bean 管理トランザクションで境界を設定する場合は、これらのメソッドを使用できます。

Bean 管理トランザクション

Bean 管理トランザクションでは、セッション Bean またはメッセージ駆動型 Bean のコードでトランザクションの境界を明示的に指定します。エンティティ Bean では、Bean 管理トランザクションではなく、コンテナ管理トランザクションを使用する必要があります。コンテナ管理トランザクションを使用する Bean では、コーディングが少なくてすみますが、次の制限があります。メソッドの実行中は、1 つのトランザクションだけに関連付けるか、トランザクションには一切関連付けないかのどちらかしかありません。この制限によって Bean のコーディングが難しくなる場合は、Bean 管理のトランザクションを使用することを検討してください。

次の擬似コードでは、Bean 管理トランザクションの使用により得られる細かい制御を示しています。この擬似コードでは、さまざまな条件をチェックすることにより、ビジネスメソッドで異なるトランザクションを起動または停止します。

```
   begin transaction
...
update table-a
...
if (condition-x)
   commit transaction
else if (condition-y)
   update table-b
   commit transaction
else
   rollback transaction
   begin transaction
   update table-c
   commit transaction
```

トランザクションサービスの管理

トランザクションの管理には、管理インタフェースまたはコマンド行インタフェースを使用できます。

この節には次の項目があります。

- 管理インタフェースを使用したトランザクションの管理
- コマンド行インタフェースを使用したトランザクションの管理

管理インタフェースを使用したトランザクションの管理

管理インタフェースを使用すると、監視の有効化、ログレベルの設定、およびトランザクションに関する詳細なオプションの設定を行うことができます。

リカバリポリシーやタイムアウトなど、インスタンス全体のトランザクションサービス属性を制御できます。管理インタフェースで指定したプロパティおよび設定は、`server.xml` ファイルに保存されます。

トランザクションサービスオプションの設定を行うには、次の手順に従います。

1. 管理インタフェースの左側のペインで、変更するトランザクション設定の Sun ONE Application Server インスタンスツリーを展開します。
2. 表示された J2EE サービスの一覧から「トランザクションサービス」を選択します。管理インタフェースの右側のペインに、トランザクションサービスのオプションの設定用のウィンドウが表示されます。

トランザクションサービスのオプションの設定

3. トランザクションの監視を有効にするには、「監視を有効」チェックボックスにチェックマークをつけます。次の表では、監視できる Java トランザクションサービスの機能を示します。

監視可能な Java トランザクションサービスの属性

プロパティ	型	説明
transactionsCompleted	int	監視を有効にしてからの完了済みトランザクション数
transactionsRolledBack	int	監視を有効にしてからロールバックされたトランザクション数
transactionsRecovered	int	監視を有効にしてからリカバリされたトランザクション数
transactionsInFlight	int	現在処理中のトランザクション数
timeStamp	long	統計情報が作成された時刻 (ミリ秒単位)。`System.getCurrentTimeInMillis()` によって記録されたものすべて

4. 「ログレベル」ドロップダウンリストから、トランザクションに設定するログレベルを選択します。ログレベルとその組み込み方法に関する詳細については、第 5 章「ログの使用」を参照してください。

5. 「再起動で回復」のチェックボックスにチェックマークをつけて、失敗したトランザクションをサーバーの再起動時に復旧するよう設定します。トランザクションのコミットプロトコル中の重要なポイントでリソースにアクセスできなくなった場合、トランザクションは完了せず、トランザクションログファイルに残ることがあります。このチェックボックスにチェックマークをつけた場合、サーバーは再起動時に未完了のトランザクションの復旧を試みます。関係するリソースにアクセスできない状態が続いていると、サーバーの再起動に時間がかかることがあります。デフォルトでは、このチェックボックスにはチェックマークがつけられていません。

6. コンテナ管理トランザクションを使用するエンタープライズ Beans では、「トランザクションタイムアウト」の値を設定することにより、トランザクションのタイムアウト間隔を制御できます。

   このプロパティ値を 0 に設定すると、トランザクションはタイムアウトしません。

   「トランザクションタイムアウト」フィールドに、トランザクションのタイムアウト間隔を指定します。指定された時間内にトランザクションが完了しないと、トランザクションはロールバックされます。この属性の設定値が 0 の場合、トランザクションはタイムアウトしません。

7. 「トランザクションログの位置」フィールドに、ログファイルを保存するディレクトリの絶対パスを指定します。トランザクションログの新しいディレクトリを有効にするには、サーバーを再起動する必要があります。

8. 「特殊な結果判別」ドロップダウンボックスで、トランザクションに適用する特殊な結果判別を選択します。表示されたオプションから「Commit」または「Rollback」を選択して、アプリケーションサーバーが未確定トランザクションの復旧中に結果を判別できない場合に行う処理を指定します。「特殊な結果判別」を「Rollback」に設定すると、アプリケーションサーバーはトランザクションをロールバックします。場合によっては、未確定トランザクションをコミットしたほうが良い場合もあります。

9. 「キーポイント間隔 (トランザクション数)」フィールドに、ログ内でのキーポイント間のトランザクション数を指定します。キーポイント処理によって、完了トランザクションのエントリが削除され、トランザクションログファイルが圧縮されるため、トランザクションログファイルのサイズが小さくなります。この属性の値を大きくすると、トランザクションログファイルのサイズが大きくなりますが、キーポイント処理が少なくなるため、パフォーマンスが向上する可能性があります。小さい値 (たとえば 100) にすると、ログファイルのサイズは小さくなりますが、キーポイント処理の頻度が増えるため、パフォーマンスがわずかに低下します。

コマンド行インタフェースを使用したトランザクションの管理

次の各項目で説明するように、コマンド行インタフェース (CLI) を使用してデータベーストランザクションの管理と監視を行うことができます。

- 実行中トランザクションの一覧表示
- トランザクションの管理
- トランザクションサービスの凍結
- トランザクションの監視

各項目では、コマンド行インタフェースを使用してトランザクションを管理する方法と監視する方法について説明します。

実行中トランザクションの一覧表示

マルチモードでユーザー名とパスワードを設定済みの場合、次のコマンドを使用して、実行中のトランザクションに関するデータを取得できます。

```
- asadmin> get --monitor
<instanceName>.transaction-service.inflight-tx
```

次のような複数行の結果が表示されます。

```
Transaction Id State Elapsed Time(ms)

txnid1 Prepared 20

txnid2 Active 100

txnid3 Active 120

  ... ... ...
```

トランザクションの管理

実行中トランザクションの一覧表示の例で、`txn-ids`、`txnid2`、および `txnid3` の各トランザクション ID を使用して、トランザクションをロールバックすることを考えます。選択したトランザクションをロールバックするためのコマンドは、次の例のようになります。

```
asadmin> set --monitor
<instanceName>.transaction-service.rollback-list=txnid2,txnid3
```

トランザクションサービスの凍結

トランザクションサービスを凍結するには、次のコマンドを実行します。

```
asadmin> set --monitor
<instanceName>.transaction-service.freeze=true
```

トランザクションサービスが凍結されると、アプリケーションサーバーのトランザクションマネージャは実行中のトランザクションを中断します。本稼働のシステムでは、凍結処理はお勧めできません。

トランザクションサービスの凍結を解除するには、次のコマンドを実行します。

```
asadmin> set --monitor
<instanceName>.transaction-service.freeze=false
```

トランザクションサービスが再び動作すると、システムは停止していた状態から処理を継続します。システムが長時間凍結状態にあると、データベース接続がタイムアウトして、トランザクションがロールバックする可能性があります。

トランザクションの監視

実行中のトランザクションなど、トランザクションに関する監視データを取得するには、次のコマンドを実行します。

```
asadmin> get --monitor <instanceName>.transaction-service.*
```

コマンドの実行時にアクティブなトランザクションがなかった場合は、次のような結果が得られます。

```
total-tx-completed = 5

total-tx-rolledback = 2

total-tx-inflight = 0

isFrozen = false

tx-inflight = No active transactions found.
```

コマンドの実行時にアクティブなトランザクションが見つかった場合は、次のような結果が得られます。

```
total-tx-completed = 5

total-tx-rolledback = 2

total-tx-inflight = 2

isFrozen = false

tx-inflight =

Transaction Id State Elapsed Time(ms)

txnid1 Prepared 500

txnid2 Active 360
```

第 10 章

ネーミングとリソースの設定

この章では、Sun ONE Application Server が使用する J2EE リソースについて、およびリソースの作成と管理に使われるメソッドについて説明します。

この章では次のトピックについて説明します。

- J2EE ネーミングサービスとリソースについて
- JNDI (Java Naming and Directory Interface) について
- 持続マネージャリソースについて
- JDBC リソースについて
- JavaMail リソースについて

J2EE ネーミングサービスとリソースについて

Enterprise JavaBean、Web アプリケーションコンポーネント、アプリケーションクライアントなどの J2EE アプリケーションは、リソースマネージャ、データソース (SQL データソースなど)、接続ファクトリ、メールセッション、Java Message Service 送信先オブジェクト、URL 接続ファクトリなど、多様なリソースにアクセスします。J2EE プラットフォームは、JNDI (Java Naming and Directory Interface) ネーミングサービスを通じてアプリケーションにリソースを渡します。

Sun ONE Application Server では、次の J2EE リソースを作成、管理できます。

- JDBC データソース
- Java Mail セッション
- JMS 送信先

JDBC データソース

JDBC データソースは、Sun ONE Application Server を使って作成、管理できる J2EE リソースです。

JDBC API は、リレーショナルデータベースシステムとの接続に使われる API です。JDBC API は、次の 2 つから構成されます。

- アプリケーションコンポーネントがデータベースへのアクセスに使うアプリケーションレベルのインタフェース
- JDBC ドライバを J2EE プラットフォームに接続するためのサービスプロバイダインタフェース

JDBC `DataSource` オブジェクトは、Java プログラミング言語で記述されたデータソースを意味します。データソースは、基本的にはデータを格納するための機能です。大企業の複雑なデータベースのように洗練されている場合もあれば、行と列だけを含む簡単なファイルである場合もあります。JDBC データソースは、Sun ONE Application Server を使って作成、管理できる J2EE リソースです。

JDBC データソースの詳細については、263 ページの「JDBC リソースについて」を参照してください。

Java Mail セッション

JMS 送信先は、Sun ONE Application Server を使って作成、管理できる J2EE リソースです。

多くのインターネットアプリケーションでは、電子メール通知を送信する機能を必要とするため、J2EE プラットフォームには、JavaMail API と、アプリケーションコンポーネントがインターネットメールを送信するための JavaMail サービスプロバイダが含まれています。JavaMail API は、次の 2 つから構成されます。

- アプリケーションコンポーネントがメールの送信に使うアプリケーションレベルのインタフェース
- J2EE SPI レベルで使われるサービスプロバイダインタフェース

Java Mail セッションは、Sun ONE Application Server を使って作成、管理できる J2EE リソースです。Java Mail セッションの詳細については、283 ページの「JavaMail リソースについて」を参照してください。

JMS 送信先

JMS (Java Messaging Service) は、信頼性の高いポイントツーポイントのメッセージングとパブリッシュ - サブスクライブモデルをサポートする標準 API です。この仕様では、ポイントツーポイントのメッセージングとパブリッシュ - サブスクライブのメッセージングの両方を実装した JMS プロバイダが必要です。

JMS は、接続ファクトリオブジェクトと送信先オブジェクトという 2 つの一般的な管理対象オブジェクトを提供します。どちらのオブジェクトもプロバイダ固有の情報をカプセル化しますが、JMS クライアント内での使用方法はまったく異なっています。接続ファクトリオブジェクトは、メッセージサーバーへの接続の確立に使用されます。一方、送信先オブジェクトは、JMS メッセージングサービスによって使用される物理的な送信先の識別に使用されます。

JNDI (Java Naming and Directory Interface) について

この節では、JNDI (Java Naming and Directory Interface) について説明します。JNDI は、多様なネーミングサービスとディレクトリサービスにアクセスするための API (アプリケーションプログラミングインタフェース) です。J2EE コンポーネントは、JNDI ルックアップメソッドを呼び出してオブジェクトの場所を特定します。

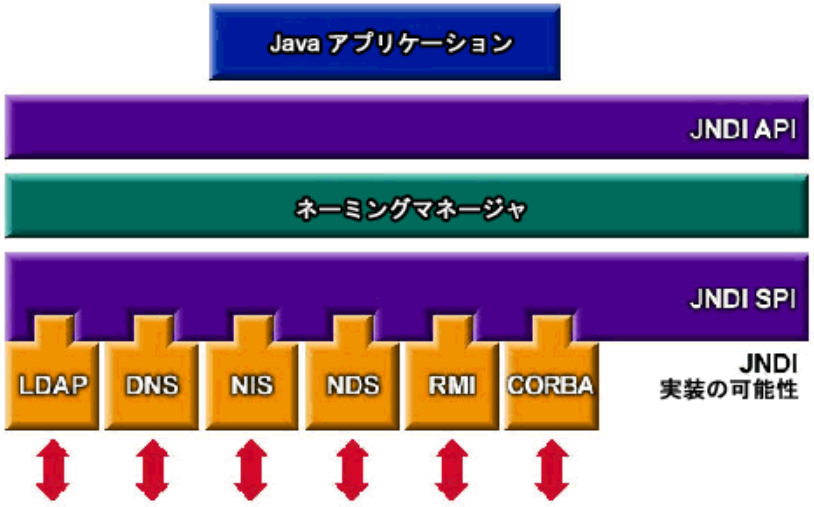
この節には次の項目があります。

- JNDI アーキテクチャ
- J2EE ネーミングサービス
- ネーミング参照とバインド情報
- J2EE 標準配備記述子でのネーミング参照
- JNDI 接続ファクトリ

JNDI アーキテクチャ

JNDI アーキテクチャは、API と SPI (Service Provider Interface) から構成されます。Java アプリケーションは JNDI API を使って各種ネーミングサービスとディレクトリサービスにアクセスします。SPI を使うことで、さまざまなネーミングサービスとディレクトリサービスを透過的にプラグインできるので、JNDI API を使う Java アプリケーションはこれらのサービスにアクセスできます。次の図「JNDI アーキテクチャの概要」は、JNDI API を通じてアクセスできるサービスを示しています。

JNDI アーキテクチャの概要



J2EE ネーミングサービス

JNDI 名は、人間が理解しやすいオブジェクト名です。この名前は、J2EE サーバーのネーミング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結びつけられます。J2EE コンポーネントは JNDI API を介してこのサービスにアクセスするため、人間が理解しやすいオブジェクト名を通常は JNDI 名と呼んでいます。Pointbase データベースの JNDI 名は、`jdbc/Pointbase` です。Sun ONE Application Server を起動すると、設定ファイルの情報が読み込まれ、名前スペースには JNDI データベース名が自動的に追加されます。

J2EE アプリケーションクライアント、Enterprise JavaBean、Web コンポーネントは、JNDI ネーミング環境へのアクセスが必要です。

アプリケーションコンポーネントのネーミング環境は、配備時またはアセンブリ時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をカスタマイズするためのメカニズムです。アプリケーションコンポーネントの環境を利用することで、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、それを変更したりすることなく、アプリケーションコンポーネントをカスタマイズできます。

J2EE コンテナはアプリケーションコンポーネントの環境を実装し、それを JNDI ネーミングコンテキストとしてアプリケーションコンポーネントに提供します。アプリケーションコンポーネントの環境は、次のように使用されます。

- アプリケーションコンポーネントのビジネスメソッドが JNDI インタフェースを使って環境にアクセスする。アプリケーションコンポーネントプロバイダは、実行時のアプリケーションコンポーネントの環境に提供されるすべての環境エントリを配備記述子に宣言する
- 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した JNDI ネーミングコンテキストの実装を提供する。また、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、管理できるように、コンテナはツールも提供する
- 配備担当者は、コンテナから提供されるツールを使ってアプリケーションコンポーネントの配備記述子に宣言された環境エントリを初期化する。配備担当者は、環境エントリの値を設定、変更できる
- コンテナは、実行時にアプリケーションコンポーネントインスタンスが利用できる環境ネーミングコンテキストを作成する。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って環境エントリの値を取得する

各アプリケーションコンポーネントは、それぞれに固有の環境エントリセットを定義します。同じコンテナに含まれるアプリケーションコンポーネントのすべてのインスタンスは、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インスタンスが実行時に環境を変更することはできません。Web コンテナや Enterprise JavaBean コンテナなどの J2EE コンテナが JNDI ネーミングサービスを使ってオブジェクトをルックアップする仕組みについては、197 ページの「J2EE コンテナの設定」を参照してください。

ネーミング参照とバインド情報

リソース参照は、配備記述子の要素の 1 つであり、そのリソースのコンポーネントのコード名を識別します。具体的には、コード化された名前がリソースの接続ファクトリを参照します。次の項の例では、リソースの参照名は `jdbc/SavingsAccountDB` です。

リソースの JNDI 名とリソース参照の名前は同じではありません。この命名方法では、配備前に 2 つの名前をマッピングする必要がありますが、さらに、リソースからコンポーネントを切り離すことも必要です。コンポーネントを切り離しておくと、あとでそのコンポーネントが別のリソースにアクセスする必要が生じた場合に、コード内の名前を変更する必要がありません。また、既存のコンポーネントから J2EE アプリケーションを編成する作業も容易になります。

次の表の「JNDI ルックアップとそれに対応する参照」は、JNDI ルックアップと、Sun ONE Application Server が使用する J2EE リソースの参照の対応を示しています。

JNDI ルックアップとそれに対応する参照

JNDI ルックアップ名	対応する参照
`java:comp/env`	アプリケーション環境エントリ
`java:comp/env/jdbc`	JDBC DataSource リソースマネージャ接続ファクトリ
`java:comp/env/ejb`	EJB 参照
`java:comp/UserTransaction`	UserTransaction 参照
`java:comp/env/mail`	JavaMail セッション接続ファクトリ
`java:comp/env/url`	URL 接続ファクトリ
`java:comp/env/jms`	JMS 接続ファクトリと JMS 送信先
`java:comp/ORB`	アプリケーションコンポーネントが共有する ORB インスタンス

J2EE 標準配備記述子でのネーミング参照

ネーミング参照は、指定されたネーミングコンテキストからアプリケーションがオブジェクトをルックアップするための文字列です。各 J2EE アプリケーションの標準のコンポーネント配備記述子には、ネーミングコンテキストと参照が設定されています。この項では、Sun ONE Application Server で使われる標準の配備記述子の機能について説明します。この節には次の項目があります。

- アプリケーション環境エントリ
- EJB 参照
- リソースマネージャ接続ファクトリへの参照
- リソース環境参照
- UserTransaction 参照
- COSNaming サービス

アプリケーション環境エントリ

<env-entry> を使って定義される環境エントリは、配備時のパラメータを J2EE アプリケーションに指定します。Web アプリケーションでは、<context-param> を使ってサーブレットコンテキストの初期化パラメータを定義できますが、アプリケーションの配備担当者が名前、タイプ、値を指定して明示的にアプリケーションパラメータを設定できるので、<env-entry> を使うことをお勧めします。

次の例は、J2EE 標準配備記述子に指定する <env-entry> の構文を示しています。

```
<env-entry>
<description> Send pincode by mail </description>
<env-entry-name> mailPincode </env-entry-name>
<env-entry-value> false </env-entry-value>
<env-entry-type> java.lang.Boolean </env-entry-type>
</env-entry>
```

<env-entry-type> タグは、エントリの完全修飾クラス名を指定しています。アプリケーションコンポーネントから JNDI (サーブレット / JSP または エンティティ Bean を参照する、または IIOP アプリケーションクライアントを参照する) を使って <env-entry> をルックアップするコード例は次のとおりです。

```
Context initContext = new InitialContext();
Boolean mailPincode = (Boolean)
initContext.lookup("java:comp/env/mailPincode");

// サブコンテキストでは相対名を使用できる
Context envContext = initContext.lookup("java:comp/env");
Boolean mailPincode = (Boolean)
envContext.lookup("mailPincode");
```

EJB 参照

JNDI ネーミングサービスにより、配備記述子のサポートとは別に、アプリケーションは「論理」名 (EJB 参照) を使って Enterprise JavaBean のホームインタフェースにマップすることができます。次に例を示します。

```
<ejb-ref>
<ejb-ref-name> ejb/EmplRecord </ejb-ref-name>
<ejb-ref-type> Entity </ejb-ref-type>
<home> com.wombat.empl.EmployeeRecordHome </home>
<remote> com.wombat.empl.EmployeeRecord </remote>
<ejb-link> EmployeeEJB </ejb-link>
</ejb-ref>
```

次の例に示すように、JSP などのアプリケーションコンポーネントは、JNDI を使って Enterprise JavaBean ホームオブジェクトにアクセスできます。

```
Context initContext = new InitialContext();
Context envContext = initContext.lookup("java:comp/env");
Object result = envContext.lookup("ejb/EmplRecord");
EmployeeRecordHome emplRecordHome = (EmployeeRecordHome)
javax.rmi.PortableRemoteObject.narrow(result,
EmployeeRecordHome.class);
```

`ejb-ref-name` 要素は、アプリケーションコードで使われる文字列を定義します (上記の例を参照)。`ejb-link` 要素は、`ejb-jar.xml` に定義されているエンティティ Bean の `ejb-name` 要素を使って定義されるターゲット Enterprise JavaBean にこの参照をリンクします。また、アプリケーション配備記述子または Enterprise JavaBean 配備記述子を変更せずにリンクすることもできます。

リソースマネージャ接続ファクトリへの参照

ファクトリは、他のオブジェクトを随時作成するオブジェクトです。リソースファクトリは、データベース接続やメッセージサービス接続などのリソースオブジェクトを作成します。これは、標準配備記述子の `<resource-ref>` 要素を使って設定されます。

次の例では、ファクトリの使用について説明しています。

例 A

タイプが `javax.sql.DataSource` のオブジェクトを返す JDBC 接続ファクトリへの参照の宣言は次のとおりです。

```
<resource-ref>
<description> Primary database </description>
<res-ref-name> jdbc/primaryDB </res-ref-name>
<res-type> javax.sql.DataSource </res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

例 B

JavaMail Session リソースファクトリへの参照の例は次のとおりです。

```
<resource-ref>
<description> mail Session </description>
<res-ref-name> mail/Session </res-ref-name>
<res-type> javax.mail.Session </res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

<res-type> は、リソースファクトリの完全修飾クラス名です。<res-auth> 変数には、コンテナまたはアプリケーションを値として指定できます。Java Mail セッションリソースファクトリの詳細については、283 ページの「JavaMail リソースについて」を参照してください。

コンテナを指定すると、リソースファクトリを JNDI ルックアップレジストリに結びつける前に、Web コンテナが認証を処理します。アプリケーションを指定した場合は、サーブレットが認証をプログラム的に処理する必要があります。次のように、リソースファクトリが異なると、リソースタイプを説明する異なるサブコンテキストがルックアップ対象となります。

- JDBC の javax.sql.DataSource ファクトリの場合は、jdbc/
- JMS の javax.jms.QueueConnectionFactory ファクトリまたは javax.jms.TopicConnectionFactory ファクトリの場合は、jms/
- JavaMail の JavaMail javax.mail.Session factory ファクトリの場合は、mail/
- java.net.URL factory ファクトリの場合は、url/

コンテナが認証を処理するアプリケーションコンポーネントからの JDBC 接続を取得するコード例は、次のとおりです。

```
InitialContext initContext = new InitialContext();
DataSource source =
(DataSource) initContext.lookup("java:comp/env/jdbc/primaryDB");
Connection conn = source.getConnection();
```

これらのリソース参照が正しく機能するには、実行時に res-ref-name を有効なリソースファクトリにマップする必要があります。

リソース環境参照

リソース環境参照を使うことで、リソースに関連づけられた管理対象オブジェクトに JNDI ルックアップを通じてアクセスできます。たとえば、アプリケーションが JMS 送信先オブジェクトにアクセスする場合などです。アプリケーションは、標準の配備記述子に定義される <resource-env-ref> 要素を使ってリソースの要件を宣言できます。

<resource-env-ref> 要素と <resource-ref> 要素の主な違いは、特定のリソース認証要件の有無です。どちらの要素もリソースファクトリ記述子によるバックアップが必要です。

例

```
<resource-env-ref>
<description> My Topic </description>
<res-env-ref-name> jms/MyTopic </res-ref-name>
<res-env-ref-type> javax.jms.Topic </res-type>
</resource-env-ref>
```

次のコード例は、JMS Topic オブジェクトへのアクセスを許可します。

```
InitialContext initContext = new InitialContext();
javax.jms.Topic myTopic =
(javax.jms.Topic) initContext.lookup("java:comp/env/jms/MyTopic");
```

resource-env-ref 変数が正しく機能するためには、管理者は実行時にターゲットリソースファクトリを利用できるように設定する必要があります。JMS Topic と Queue 送信先へのアクセスについては、第 11 章「JMS サービスの使用」を参照してください。

UserTransaction 参照

J2EE では、コンテナが JNDI 名 java:comp/UserTransaction に UserTransaction オブジェクトの実装を提供する必要があります。アプリケーションは、UserTransaction オブジェクトを使ってトランザクションを開始、コミット、中止します。

トランザクションをプログラム的に初期化、実行する場合、コンポーネントは java:comp/UserTransaction の JNDI ルックアップを実行してコンテナのデフォルトトランザクションコーディネータへの参照を取得します。返されるオブジェクトは、javax.transaction.UserTransaction インタフェースを実装しており、トランザクションを開始、コミット、ロールバックし、トランザクションの状態を問い合わせるプログラムで使用できます。Sun ONE Application Server に実装される JNDI は、このようなトランザクションコーディネータのルックアップをサポートしています。javax.transaction.UserTransaction インタフェースの詳細については、217 ページの「トランザクションサービスの使用」を参照してください。

初期ネーミングコンテキスト

Sun ONE Application Server がサポートするネーミングは、J2EE 1.3 を基本として、いくつかの拡張が追加されています。アプリケーションコンポーネントが `InitialContext()` を使って初期コンテキストを作成すると、Sun ONE Application Server はアプリケーションのネーミング環境への参照として機能するオブジェクトを返します。次に、このオブジェクトは `java:comp/env namespace` のサブコンテキストを返します。各アプリケーションには専用の名前スペースがあります。つまり、`java:comp/env name` スペースはアプリケーションごとに存在し、あるアプリケーションの名前スペースに結びつけられるオブジェクトは、別のアプリケーションに結びつけられるオブジェクトと競合しません。

COSNaming サービス

Enterprise JavaBean 相互運用プロトコルでは、JNDI API を使った Enterprise JavaBean オブジェクトのルックアップに COSNaming プロトコルを使う必要があります。

Enterprise JavaBean コンテナは、CORBA `CosNaming` サービスに `EJBHome` オブジェクト参照をパブリッシュできる必要があります。`CosNaming` サービスは、定義されている `CosNaming` モジュールに IDL インタフェースを実装し、IIOP を通じたオペレーションをクライアントが解決および一覧できるようにする必要があります。

`CosNaming` サービスは、ルート `NamingContext` オブジェクトのホスト、ポート、およびオブジェクトキーを提供する上で、CORBA Interoperable Name Service 仕様の要件に従う必要があります。`CosNaming` サービスは、指定されたホスト、ポート、およびオブジェクトキーのルート `NamingContext` で IIOP を呼び出す必要があります。

クライアントコンテナ (Enterprise JavaBean、Web、またはアプリケーションクライアントコンテナ) には、サーバーの `CosNaming` サービスにアクセスし、標準の `CosNaming` API を使って `EJBHome` オブジェクトを解決するために、Interoperable Name Service 仕様に定義されているメカニズムを使う JNDI `CosNaming` サービスプロバイダが含まれている必要があります。JNDI `CosNaming` サービスプロバイダが、JNDI SPI アーキテクチャを使っているとは限りません。JNDI `CosNaming` サービスプロバイダは、次の URL からオブジェクト参照を作成し、サーバーの `CosNaming` サービスのルート `NamingContext` にアクセスする必要があります。

`corbaloc:iiop:1.2@<host>:<port>/<objectkey>` の `<host>`、`<port>`、および `<objectkey>` は、サーバーの `CosNaming` サービスによって指定されるルート `NamingContext` に対応する値です。この URL または同等のメカニズムを使用する必要があります。

配備時に、クライアントコンテナの開発者は、サーバーの CosNaming サービスのホスト、ポート、およびオブジェクトキー、およびクライアントコンポーネントの配備記述子に含まれる各 ejb-ref 要素について、サーバーの名前スペースを参照するなどして、サーバーの EJBHome オブジェクトの CosNaming 名を取得する必要があります。次に、ejb-ref-name (これは JNDI ルックアップの呼び出しでクライアントコードによって使用される) を EJBHome オブジェクトの CosNaming 名にリンクします。実行時は、クライアントコンポーネントの JNDI ルックアップ呼び出しは、サーバーの CosNaming サービスにアクセスする CosNaming サービスプロバイダを使って CosNaming 名を解決し、クライアントコンポーネントに EJBHome オブジェクト参照を返します。

EJBHome オブジェクトの名前は、指定されたホストとポートでアクセスが可能な CosNaming サービスの名前スペースに確実に存在するため、クライアントコンテナとサーバーコンテナの名前スペースを組み合わせる必要はありません。

CosNaming を使う利点は、非 J2EE CORBA クライアントおよびサーバーとの相互運用で必要となる IIOP インフラストラクチャとの統合性が高いことです。CosNaming は CORBA オブジェクトだけを格納するため、多くのベンダーは、その他のリソースを格納するために別のエンタープライズディレクトリサービスも使用します。

Sun ONE Application Server は、J2EE 1.3 仕様に基づいて、JNDI のすべてのネーミングリソースを統合します。

CosNaming プロバイダ : グローバルな JNDI 名スペースをサポートする (IIOP アプリケーションクライアントにアクセスできる) ために、Sun ONE Application Server には J2EE ベースの CosNaming プロバイダが含まれます。このプロバイダは、CORBA 参照 (リモート EJB 参照) のバインドをサポートします。IIOP クライアントに返される InitialContext は CosNaming プロバイダです。Sun ONE Application Server のインスタンスは、IIOP クライアントがルックアップし、バインドするエンティティ Beans を登録します。

Sun ONE Application Server は、CosNaming およびローカル JNDI ネーミング環境に格納されるオブジェクトを一時的なものとして扱います。つまり、サーバーの起動時またはアプリケーションの再読み込み時に、すべての関連オブジェクトは元の名前スペースに戻されます。CORBA/IIOP クライアントの設定については、327 ページの「Corba/IIOP クライアント用のサーバーの設定」を参照してください。

JNDI 接続ファクトリ

J2EE Web アプリケーションでは、web.xml ファイル内の配備記述子を使って、アプリケーション環境エントリ、リソースマネージャ (SQL データソースなど) 接続ファクトリ、または Enterprise JavaBean への参照を定義します。アプリケーションは、J2EE コンテナから提供される JNDI InitialNamingContext を使ってこれらの参照をルックアップします。これにより、アプリケーションのソースコードにアクセスしたり、変更したりせずに、配備記述子に変更を加えるだけで別のアプリケーションサーバー環境にアプリケーションを移植できます。同様に、J2EE では、これらの JNDI ネーミング参照の主要な連絡媒体として、エンティティ Beans の配備記述子 (ejb-jar.xml) や IIOP アプリケーションクライアントの配備記述子 (application-client.xml) が必要となります。

コネクションファクトリは、J2EE コンポーネントによるリソースへのアクセスを可能にするコネクションオブジェクトを作成するオブジェクトです。データベースの接続ファクトリは javax.sql.DataSource オブジェクトで、このオブジェクトは java.sql.Connection オブジェクトを作成します。

Sun ONE Application Server では、次のリソースとリソースファクトリにアクセスする方法を設定できます。

- JDBC 接続ファクトリ
- MQ に基づく JMS 接続ファクトリ
- JavaMail セッション接続ファクトリ
- JCA 接続ファクトリ
- ユーザーが記述する汎用のカスタムリソースオブジェクトファクトリ
- LDAP などの外部リソースリポジトリのサポート

すべての Sun ONE Application Server リソースファクトリは、server.xml ファイルの <resources> </resources> タグ内に指定され、jndi-name 属性を使って指定される JNDI 名を持ちます。この属性は、ファクトリをサーバー全体の名前スペースに登録するときに使われます。開発者は、ユーザーが指定したアプリケーション固有のリソース参照名 (resource-ref 要素または resource-env-ref 要素内で宣言される) を、resource-ref-mapping 要素を使ってサーバー全体のリソースファクトリにマッピングできます。これにより、特定のアプリケーションにどの JDBC ドライバ (およびその他のリソースファクトリ) を使うかを配備時に決定できます。

カスタムリソースはローカル 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。どちらのリソースも、ユーザーが指定したファクトリクラス要素、JNDI 名、属性などを必要とします。ここでは、JNDI 接続ファクトリリソースを J2EE リソース用に設定する方法と、これらのリソースにアクセスする方法について説明します。

この節では次の項目について説明します。

- カスタムリソースの作成
- 外部 JNDI リソースの作成
- 外部 JNDI リポジトリへのアクセス
- アプリケーションリソース参照のマッピング
- URL 接続ファクトリリソースについて
- アプリケーションリソース環境参照のマッピング
- EJB 参照のマッピング

カスタムリソースの作成

サーバー全体のカスタムリソースオブジェクトファクトリを指定するときは、`server.xml` に定義されている `custom-resource` 要素を使います。これらのオブジェクトファクトリは、`javax.naming.spi.ObjectFactory` インタフェースを実装しています。この要素は、サーバー全体の名前スペースで使われる JNDI 名 (その他の Sun ONE Application Server リソースのように、jndi-name サブ要素を使って指定される) と、タイプ、リソースファクトリクラスの名前、インスタンス化で使われる標準プロパティのセットを関連づけます。

次の例は、`javax.naming.spi.ObjectFactory` インタフェースの実装を示しています。

```
<resources> <custom-resource jndi-name="test/myBean"
res-type="test.MyBean"factory-class="test.MyBeanFactory"
enabled="true">

<property name="foo" value="test custom bean prop" />
</custom-resource>
</resources>
```

リソース参照の環境参照と EJB 参照は、`server.xml` 内の `custom-resource` タグと `external-jndi-resource` タグを使って定義されるサーバー全体の設定済みリソースにリンクされている必要があります。アプリケーションコンポーネントの動的な再配備は、JNDI ネーミング環境の問題です。Sun ONE Application Server は、アプリケーション固有のすべての参照を解放し、すべての新しい参照を新たにインストールされたアプリケーションのネーミングコンテキストに結びつけます。

管理インタフェースを使ってカスタムリソースを作成するには、次の手順を実行します。

1. Sun ONE Application Server の左側のペインで、変更したい JNDI 設定を含むアプリケーションサーバーインスタンスを開きます。

2. 「JNDI」を展開し、「カスタムリソース」をクリックします。すでにカスタムリソースが作成されている場合は、右側のペインにそれがリスト表示されます。新しいカスタムリソースを作成するには、「新規」をクリックします。管理インタフェースの右側のペインに「JNDI のカスタムリソースページ」が表示されます。

JNDI のカスタムリソースページ

3. リソースへのアクセスに使う名前を「JNDI 名」フィールドに入力します。この名前は JNDI ネーミングサービスに登録されます。
4. 上記の例のように、タイプの完全修飾定義を「リソースタイプ」に入力します。リソースタイプの定義は、次の形式で指定する必要があります。`xxx.xxx`.
5. 作成しているカスタムリソースのファクトリクラス名を「ファクトリクラス」フィールドに入力します。この名前は、ユーザーが指定するファクトリクラス名です。このクラスは、`javax.naming.spi.ObjectFactory` インタフェースを実装します。
6. 作成しているリソースの説明を「説明」フィールドに入力します。これは文字列のフィールドです。250 文字以内で入力します。
7. 「カスタムリソースを有効」ボックスにチェックマークをつけて、カスタムリソースを有効にします。
8. 「了解」をクリックして、外部リソースを保存します。

外部 JNDI リソースの作成

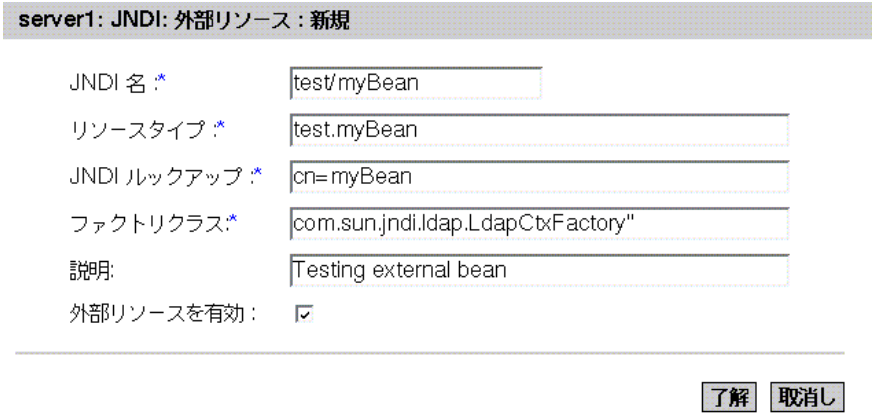
管理インタフェースを使って外部リソースを作成するには、次の手順を実行します。

1. Sun ONE Application Server の左側のペインで、変更したい JNDI 設定を含む Sun ONE Application Server インスタンスを開きます。

2. 「JNDI」を開き、「外部リソース」を選択します。すでに外部リソースが作成されている場合は、右側のペインにそれがリスト表示されます。新しい外部リソースを作成するには、「新規」をクリックします。

   管理インタフェースの右側のペインに次のような「JNDI 外部ソースページ」のウィンドウが表示されます。

JNDI 外部ソースページ



3. リソースへのアクセスに使う名前を「JNDI 名」フィールドに入力します。この名前は JNDI ネーミングサービスに登録されます。
4. 上記の例のように、タイプの完全修飾定義を「リソースタイプ」に入力します。リソースタイプの定義は、次の形式で指定する必要があります。`xxx.xxx`.
5. 外部リポジトリでルックアップする JNDI 値を「JNDI ルックアップ」に入力します。たとえば、Bean クラスをテストするために外部リポジトリに接続する外部リソースを作成する場合は、JNDI 値は `cn=`*`testmybean`* となります。
6. たとえば `com.sun.jndi.ldap` のような JNDI ファクトリクラス外部リポジトリを「ファクトリクラス」フィールドに入力します。このクラスは、`javax.naming.spi.ObjectFactory` インタフェースを実装します。
7. 作成しているリソースの説明を「説明」フィールドに入力します。これは文字列のフィールドです。250 文字以内で入力します。
8. 「外部リソースを有効」ボックスにチェックマークをつけて、外部リソースを有効にします。
9. 「了解」をクリックして、外部リソースを保存します。

外部 JNDI リポジトリへのアクセス

Sun ONE Application Server で稼働するアプリケーションの多くは、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します。たとえば、汎用の Java オブジェクトは、Java スキーマごとに LDAP に格納されます。外部 JNDI リソース要素を使うことで、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。

例

```
<resources>
<!-- external-jndi-resource 要素は、外部 JNDI リポジトリに格納されて
-- いる J2EE リソースへのアクセス方法を指定します。次の例は、LDAP に
-- 格納されている Java オブジェクトへのアクセス方法を示しています。
-- factory-class 要素は、リソースファクトリへのアクセスに必要な JNDI
-- InitialContext ファクトリを指定します。外部 JNDI コンテキストおよび
-- jndi-lookup-name に適用される環境に対応するプロパティ要素は、JNDI
-- 名を参照してルックアップし、ターゲットオブジェクト ( この場合は Java)
-- を取得します。
-->
<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>
```

アプリケーションリソース参照のマッピング

アプリケーション固有のリソース参照は、事前に定義されたサーバー全体のリソースファクトリにマップする必要があります。このマップには、Sun ONE Application Server 固有のリソース参照をマップする要素が使われます。

次の例では、リソース参照が JDBC DataSource に指定されている Web アプリケーションの配備記述子 `web.xml` を紹介します。

```
<resource-ref>
<res-ref-name> jdbc/EstoreDataSource </res-ref-name>
<res-type> javax.sql.DataSource </res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

次のように、目的の res-ref-name をコンテナ全体の Oracle JDBC 接続リソースファクトリにマップすることもできます。

```
<resource-ref>
<res-ref-name> jdbc/EstoreDataSource </resource-ref-name>
<jndi-name> jdbc/estore/InventoryDB </jndi-name>
</resource-ref>
```

URL 接続ファクトリリソースについて

URL 接続ファクトリでは、server.xml にリソースを定義する必要はありません。Sun ONE Application Server アプリケーションの Web または Enterprise JavaBean 配備記述子に対応する jndi-name 要素がターゲット URL を決定します。

たとえば、Web アプリケーションの配備記述子 web.xml がリソース参照 java.net.URL を指定し、これが sun-web.xml 内の URL http://www.*sun.com/index.html* にマップされていると仮定します。

マッピングは次のようになります。

```
<resource-ref>
<res-ref-name>myURL</res-ref-name>
<res-type>java.net.URL</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

```
<sun-web-app>
<resource-ref>
<res-ref-name>myURL</res-ref-name>
<jndi-name> http://www.sun.com/index.html </jndi-name>
</resource-ref>
</sun-web-app>
```

アプリケーションリソース環境参照のマッピング

アプリケーション固有のリソース環境参照宣言は、アプリケーションサーバーの実行時環境で利用できるターゲットリソースオブジェクトにマップする必要があります。配備担当者は、次のように Sun ONE Application Server 固有の設定ファイルに定義されるリソース環境マッピング要素を使うことで、このマッピングを行えます。

例

```
<resource-env-ref>
<description> My Topic </description>
<res-env-ref-name> jms/MyTopic </res-ref-name>
<res-env-ref-type> javax.jms.Topic </res-type>
</resource-env-ref>
```

この参照は、server.xml に定義されている jms/iMQ/Topics/Stocks/SUNW トピックにマップされます。詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

```
<resource-env-ref-mapping>
<res-env-ref-name> jms/MyTopic </res-ref-name>
<jndi-name> jms/iMQ/Topics/Stocks/SUNW </jndi-name>
</resource-env-ref-mapping>
```

EJB 参照のマッピング

アプリケーションコードで実際に使われている ejb-name をターゲット Enterprise JavaBean で使われている ejb-name から切り離すこともできます。これは、Web アプリケーションの配備記述子 web.xml を変更せずに、Enterprise JavaBean の配備記述子の ejb-name を使う場合に特に便利です。固有の設定を使うことで、Sun ONE Application Server 固有の配備記述子に含まれる ejb-ref-mapping 要素を使わずに、ejb-ref-name 要素をターゲット Bean の ejb-name にマップできます。

例

```
<ejb-ref>
<ejb-ref-name> ejb/EmplRecord </ejb-ref-name>
<ejb-ref-type> Entity </ejb-ref-type>
<home> com.wombat.empl.EmployeeRecordHome </home>
<remote> com.wombat.empl.EmployeeRecord </remote>
</ejb-ref>
```

```
<ejb-ref>
<ejb-ref-name> ejb/EmplRecord </ejb-ref-name>
<jndi-name> AccountEJB </jndi-name>
</ejb-ref-mapping>
```

持続マネージャリソースについて

この節では、持続性について、および Sun ONE Application Server がサポートしているプラグイン可能な持続マネージャの概要について説明します。

この節では、次の項目について説明します。

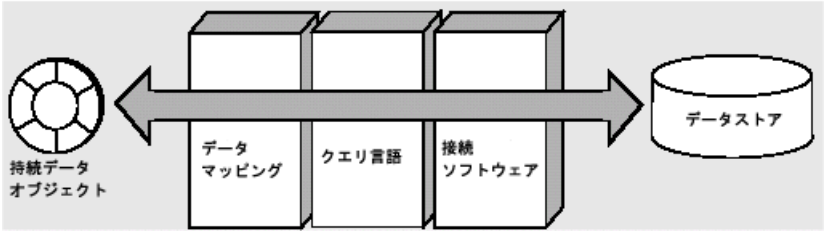
- 持続性について
- 持続マネージャの役割
- 配備前の Bean の設定
- 持続マネージャの新規作成

持続性について

ほとんどのビジネスアプリケーションで重要とされるのは、アプリケーション外に長期間格納される持続性データのプログラム処理です。使用や変更が必要な場合は、持続性データは一時メモリに読み込まれますが、長期的に保管する場合は、データはリレーショナルデータベースや単層ファイルシステムに書き込まれます。

オブジェクト指向のプログラミングシステムでは、アプリケーションコードが 1 つまたは複数のオブジェクトを操作すると、持続性データはメモリに読み込まれます。次の図「基本的な持続性スキーム」に示すように、一般に、データストア内の持続性データとメモリ内の持続性データオブジェクトは、何層ものソフトウェア層を介して対応しています。

基本的な持続性スキーム



各データストアには、データストアとアプリケーションの接続を設定、維持するドライバソフトウェアを介して外界に通じるインタフェースがあります。この接続が確立されている状態では、データストアからの情報の取得、アプリケーションへの情報の読み込み、あるいはその反対にアプリケーションからデータストアへのデータの書き込みにはクエリ言語が使われます。もう一方の層は、メモリ内のデータオブジェクトとデータストア上の情報とのマップを行います。

この一般スキームでは、プログラマはアプリケーションが使用、操作する実行時オブジェクトとして持続性データを扱えます。このスキームは、基本的なすべての持続操作 (CRUD と呼ばれます) をサポートしています。

- C (Creating) - 持続性データの作成 (データストアへの挿入)
- R (Retrieving) - 持続性データの検索 (データストアからの選択)
- U (Updating) - 持続性データの更新
- D (Deleting) - 持続性データの削除

持続マネージャの役割

PM (持続マネージャ) は、Enterprise JavaBean コンテナ内で持続性がコンテナ管理のエンティティ Beans の持続性を維持します。エンティティ Bean のプロバイダは、エンティティ Bean のクラスを抽象クラスとして提供する必要があります。PM プロバイダのツールは、具体的な実装を提供する必要があります。これらは、抽象エンティティ Bean と関連クラスをサブクラスに分け、具体的な実装を提供したり、カプセル化と委譲を使って、持続性を維持します。

PM のツールで提供されるクラスは、エンティティ Beans 間の関係および持続状態へのアクセスを管理します。また、PM ツールは、コンテナ管理による関係 (CMR) の管理に使用される `java.util.Collection` クラスの実装を提供する必要があります。

配備前の Bean の設定

Enterprise JavaBean の標準には、Enterprise JavaBean の 2 種類の持続性が定義されています。コンテナ管理による持続性 (CMP) と Bean 管理による持続性 (BMP) がそれにあたります。Enterprise JavaBean 2.0 仕様には、Enterprise JavaBean サーバーと持続マネージャを結ぶ標準の API は定義されていません。

ここでは、配備とコード生成で必要になる統合要件について説明します。配備には複数の意味があります。一般に、配備プロセスは設定、コード生成、インストールという 3 段階の手順から構成されます。

使用する持続メカニズム、持続性ベンダー、使用中のバージョン、持続メカニズムが必要とする追加情報など、Bean には多数のプロパティを設定する必要があります。ほとんどの持続性ベンダーには、関連するすべての Bean およびその依存クラスを表し、1 つの単位として配備することができるプロジェクトという概念が適用されます。プロジェクトごとにベンダー固有の xml ファイルを利用できます。

配備に使用する標準ファイルには、ejb-jar.xml、sun-ejb-jar.xml、sun-cmp-mappings.xml の 3 種類があります。CMP Beans を持つ Enterprise JavaBean モジュールは、sun-ejb-jar.xml 内に <pm-descriptors> を持ち、ここに少なくとも 1 つの <pm-descriptor> 要素があります。さらに、この要素は 5 つの属性を指定します。この 5 つの属性は、pm-identifier、pm-version、pm-config、pm-class-generator、pm-mapping-factory です。

sunEjb_jar_2_0.DTD 内の記述子のような Sun ONE Application Server 固有の記述子は持続マネージャに関するタグを定義します。CMP 記述子の例として、Sun ONE Application Server DTD に定義されている次のようなコードを示します。

PM 記述子には 1 つまたは複数の pm 記述子を含めることができます。ただし一度に使える記述子は 1 つだけです。

```
-->
<!ELEMENT pm-descriptors ( pm-descriptor+, pm-inuse)>
<!--
pm-descriptor は、エンティティ Bean に関連付けられた持続マネージャの
プロパティを示します。
-->
<!ELEMENT pm-descriptor ( pm-identifier, pm-version, pm-config?,
pm-class-generator?,
pm-mapping-factory?)>
<!--
この要素は、PM の実装を提供するベンダーを説明しています。この例では、Sun ONE Application Server Transparent Persistence、TopLink、Versant、または CocoBase となります
-->
<!ELEMENT pm-identifier (#PCDATA)>
<!--
pm-version は、使用する PM ベンダー製品のバージョンを指定します
-->
<!ELEMENT pm-version (#PCDATA)>
<!--
pm-config は、使用するベンダー固有の設定ファイルを指定します
-->
```

```
<!ELEMENT pm-config (#PCDATA)>
<!--
pm-class-generator は、ベンダー固有の具体的なクラスのジェネレータを指定します
これは、そのベンダーに固有のクラスの名前です
-->
<!ELEMENT pm-class-generator (#PCDATA)>
<!--
pm-mapping-factory は、ベンダー固有のマッピングファクトリを指定します
これは、そのベンダーに固有のクラスの名前です
-->
<!ELEMENT pm-mapping-factory (#PCDATA)>
```

持続マネージャの新規作成

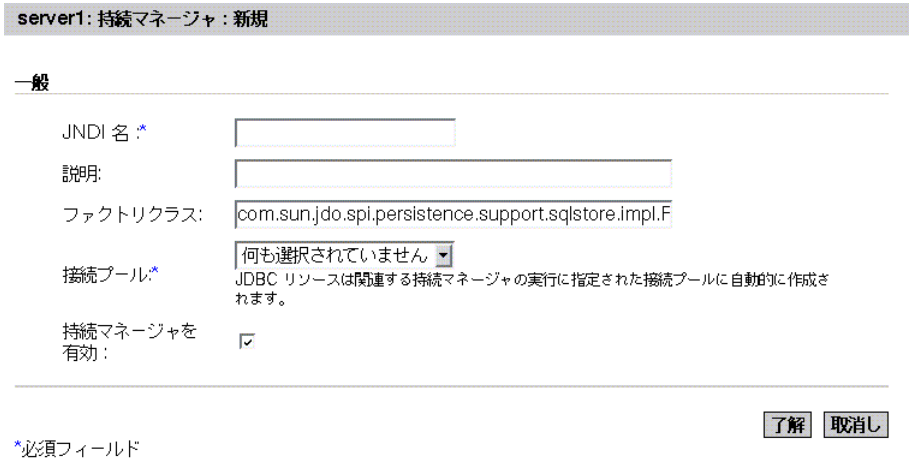
管理インタフェースを使って新しい持続マネージャを作成できます。持続マネージャを新規作成するには、次の手順を実行します。

1. 管理インタフェースの左側のペインで、新たに持続マネージャを作成する Sun ONE Application Server インスタンスを開きます。表示されるサーバーコンポーネントのリストから「持続マネージャ」を選択します。

   その Sun ONE Application Server インスタンスに固有の持続マネージャがすでに作成されている場合は、管理インタフェースの右側のペインにリスト表示されます。

2. 新しい持続マネージャを作成するには、「新規」をクリックします。以下のような「持続マネージャの新規作成」用のウィンドウが表示されます。

持続マネージャの新規作成



3. アプリケーションに代わる持続マネージャを特定するためにアプリケーションサーバーランタイムが使用する JNDI 名を「JNDI 名」に入力します。この名前は、Sun ONE Application Server 固有の配備記述子に含まれるエンティティ Bean の cmp-resource 要素に定義されている名前と同じである必要があります。
4. 新しい持続マネージャの説明を「説明」フィールドに入力します。これは文字列のフィールドです。250 文字以内で入力します。
5. 持続マネージャのファクトリクラス接続を「ファクトリクラス」フィールドに入力します。`setEntityContext` はこの接続ファクトリから JNDI 名をルックアップします。ファクトリクラス名は、持続マネージャインスタンスを作成する持続マネージャファクトリのクラス名です。標準の設定では、これは Sun ONE Application Server の内部持続マネージャファクトリクラスに設定されます。別の実装を使う場合は、サーバークラスパスからそのクラスにアクセスできる必要があります。
6. 新しい持続マネージャがプールされるデータベース接続プールを「接続プール」ドロップダウンリストから選択します。接続プールでは、エンティティ Bean は 1 つの接続を要求し、それを使って複数のクライアントスレッドのステートメントを同時に実行できます。その他のデータベースアクセスと同様に、持続マネージャは接続プールを使ってパフォーマンスとスケーラビリティを向上します。既存の接続プールを選択するか、プールを作成していない場合は「何も選択されていません (何も選択されていません)」を選択します。

   注 : PM ランタイムが JNDI を使って接続プールにバインドできるように、JDBC リソースが自動的に作成されます。JDBC リソースの JNDI 名は、プレフィックス「PM」をつけた PM JNDI 名と同じになります。持続マネージャを削除すると、関連する JDBC リソースも削除されます。

7. 「持続マネージャを有効」ボックスにチェックマークをつけて、持続マネージャを有効にします。これで、指定した接続ファクトリの持続マネージャが有効になります。
8. 「了解」をクリックして、変更内容を保存します。

JDBC リソースについて

この節では、JDBC API の概要を説明し、JDBC リソースについて、および Sun ONE Application Server での JDBC リソースの実装と使用について詳しく説明します。

この節には次の項目があります。

- JDBC API について
- データベースアクセスモデルについて
- JDBC データソースについて
- JDBC 接続について
- JDBC トランザクションについて

JDBC API について

JDBC API は、仮想的にあらゆる表形式データにアクセスするための Java API です。JDBC は「Java Database Connectivity」の略号であると考えられがちですが、これは商標名であって略号ではありません。JDBC API は Java プログラミング言語で記述されたクラスとインタフェースのセットです。これは、ツールやデータベースの開発者に標準 API を提供し、全体を Java で記述した API によるデータベースアプリケーションの作成を可能にします。

JDBC API を使うことで、リレーショナルデータベースシステムへの SQL ステートメントの送信や、あらゆる種類の SQL のサポートが簡単になります。ただし、JDBC 3.0 API はデータベース外のファイルなど、その他の種類のデータソースの利用にも対応しており、SQL だけを対象とした API ではありません。

JDBC API の利点は、アプリケーションが仮想的にあらゆるデータソースにアクセスし、Java 仮想マシンを実装したあらゆるプラットフォームで実行できることにあります。言い換えれば、JDBC API を使うことで、Sybase データベースにアクセスするプログラムを記述すると、Oracle データベースや IBM DB2 データベースなどにアクセスする別のプログラムを個別に記述する必要がなくなります。JDBC API を使って 1 つのプログラムを記述すれば、そのプログラムは SQL などのステートメントを適切な

データソースに送信できます。また、Java プログラミング言語で記述したアプリケーションを使えば、プラットフォームごとに異なるアプリケーションを記述する必要もなくなります。Java プラットフォームと JDBC API を組み合わせることで、プログラマは 1 回記述したコードをどこででも実行できます。

JDBC API の機能

JDBC テクノロジベースのドライバ (JDBC ドライバ) は、次の 3 つの処理を実行できます。

- データソースとの接続を確立する
- クエリステートメントとアップデートステートメントをデータソースに送信する
- 結果を処理する

次のコードは、この 3 段階の処理を示す簡単な例です。

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/AcmeDB");
Connection con = ds.getConnection("myLogin", "myPassword");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
          int x = rs.getInt("a");
          String s = rs.getString("b");
          float f = rs.getFloat("c");
   }
```

データベースアクセスモデルについて

JDBC API は、2 層と 3 層の両方のデータベースアクセスモデルをサポートしています。Sun ONE Application Server には、より一般的な 2 層のデータベースアクセスモデルが統合されています。

この節には次の項目があります。

- 2 層のデータベースアクセスモデル
- 3 層のデータベースアクセスモデル

2 層のデータベースアクセスモデル

2 層のデータベースアクセスモデルでは、Java アプレットまたはアプリケーションは、DBMS 専用プロトコルを使ってデータソースに直接アクセスします。このアクセスモデルでは、アクセス先の特定のデータソースと対話できる JDBC ドライバが必要です。ユーザーのコマンドはデータベースまたはその他のデータソースに送信され、ステートメントの結果がユーザーに返されます。データソースは、ユーザーがネットワークを介して接続できる別のマシンにあってもかまいません。この設定をクライアント / サーバー設定と呼び、ユーザーのマシンをクライアント、データソースを格納するマシンをサーバーと呼びます。ネットワークは、企業の従業員が接続するようなイントラネットでも、インターネットでもかまいません。

3 層のデータベースアクセスモデル

3 層のデータベースアクセスモデルでは、Java アプレットまたはアプリケーションはサービスの「中間層」にコマンドを送信し、コマンドはそこからデータソースに送信されます。クライアントアプリケーションは HTTP、RM、CORBA などの呼び出しを使って中間層と対話します。中間層は、DBMS 専用プロトコルを使ってデータソースと対話します。データソースはコマンドを処理し、結果を中間層に返し、中間層は結果をユーザーに返します。中間層を設けることで、企業データへのアクセスと実行できるアップデートの種類を管理できるため、MIS のディレクターには 3 層のアクセスモデルは魅力的かもしれません。また、アプリケーションの配備が簡単になるという利点もあります。さらに、多くの場合、3 層のアーキテクチャにはパフォーマンス上の利点もあります。

JDBC データソースについて

`DataSource` オブジェクトは、Java プログラミング言語で記述されたデータソースです。データソースは、基本的にはデータを格納するための機能です。大企業の複雑なデータベースのように洗練されている場合もあれば、行と列だけを含む簡単なファイルである場合もあります。データソースは、リモートサーバーに置くことも、ローカルマシンに置くこともできます。アプリケーションは接続を使ってデータソースにアクセスし、`DataSource` インスタンスが指定する特定データソースへの接続のファクトリとして `DataSource` オブジェクトを使います。DataSource インタフェースには、データソースとの接続を確立する 2 種類のメソッドが用意されています。

`DataSource` オブジェクトには、特定のデータソースを識別および説明するプロパティがあります。また、`DataSource` オブジェクトと JNDI ネーミングサービスを併用することで、そのオブジェクトを使うアプリケーションとは別に作成、配備、管理することができます。ドライバベンダーは、`DataSource` インタフェースの基本実装となるクラスを JDBC 2.0 または 3.0 ドライバ製品の一部として提供します。

この節には次の項目があります。

- DataSource オブジェクトのプロパティ

- JDBC リソースの登録

DataSource オブジェクトのプロパティ

DataSource オブジェクトには、実際のデータソースを識別するプロパティセットがあります。これらのプロパティには、データベースサーバーの場所、データベースの名前、サーバーとの通信に使用するネットワークプロトコルなどの情報が含まれます。DataSource のプロパティは、JavaBeans の設計パターンを踏襲しているので、通常は DataSource オブジェクトの配備時に設定されます。

ベンダー間での DataSource 実装の均一性を確保するために、JDBC 2.0 API ではプロパティの標準セット、および各プロパティの標準名を規定しています。

DataSource インタフェースを実装するクラスのインスタンスは、1 つの特定のデータソースを表します。そのインスタンスが作成するすべての接続は、同じデータソースを参照します。DataSource の基本的な実装では、DriverManager によって返される接続オブジェクトと同様に、DataSource.getConnection メソッドの呼び出しによって、データソースとの物理的な接続を提供する接続オブジェクトが返されます。

アプリケーションがネットワーク上のリモートサービスを探してアクセスする方法は、JNDI によって提供されます。リモートサービスは、メッセージングサービスやアプリケーション固有のサービスなど、どのようなエンタープライズサービスでもかまいませんが、JDBC アプリケーションの主な目的は、もちろんデータベースサービスにあります。DataSource オブジェクトを作成して JNDI ネーミングサービスに登録すると、アプリケーションは JNDI API を使ってその DataSource オブジェクトにアクセスできるようになり、そのオブジェクトが表すデータソースへの接続に利用されます。

同様に、接続プールを実装する DataSource オブジェクトは、DataSource クラスによって表される特定のデータソースへの接続を作成します。ただし、DataSource.getConnection メソッドが返す接続オブジェクトは、物理的な接続ではなく、PooledConnection オブジェクトへの参照です。アプリケーションは、通常どおりに接続オブジェクトを使うため、違いを意識することはほとんどありません。すべての接続と同様に、プールされた接続を常に明示的に閉じることが必要であることを除き、接続プールはアプリケーションコードにまったく影響しません。プールされている接続をアプリケーションが閉じると、接続は再利用可能な接続のプールに加えられます。次に DataSource.getConnection を呼び出したときに、接続が残っていれば、プールされているいずれかの接続への参照が返されます。接続プールを利用することで、要求のたびに物理的な接続を作成する必要がなくなるため、アプリケーションを高速に実行するのに役立ちます。

同様に、分散トランザクション環境で機能するように DataSource クラスを実装することもできます。たとえば、Enterprise JavaBean サーバーは分散トランザクションをサポートしており、対話相手となる DataSource クラスの実装を必要とします。この場合、DataSource.getConnection メソッドは分散トランザクションで利用できる Connection オブジェクトを返します。規定により、Enterprise JavaBean サーバーは接

続プールと分散トランザクションの両方をサポートします。トランザクション管理も、接続プールのように内部処理されるため、分散環境の利用は簡単です。ただし、トランザクションを分散するときに、アプリケーションがトランザクションメソッドとして commit または rollback を呼び出せないという制約があります。また、接続を auto-commit モードでプットすることもできません。これらの制約は、トランザクションマネージャが分散トランザクションを自動的に開始、終了するため、トランザクションの開始時または終了時に影響する処理をアプリケーションが実行できないことに起因しています。Java トランザクションの詳細については、第 9 章「トランザクションサービスの使用」を参照してください。

JDBC リソースの登録

管理インタフェースまたはコマンド行インタフェースを使って、JDBC リソースを Sun ONE Application Server に登録できます。

この節には次の項目があります。

- コマンド行によるリソースの登録
- 管理インタフェースによるリソースの登録

コマンド行によるリソースの登録

コマンド行インタフェースを使って JDBC リソースを登録するには、次のコマンドを実行します。

```
./asadmin create-jdbc-resource
```

次に示すように、JDBC リソースを登録する XML コードにはいくつかの属性を指定する必要があります (sun-server_7_0.dtd から抜粋)。

```
<!-- JDBC javax.sql.DataSource resource definition -->

<!ELEMENT jdbc-resource (description?, property*)>

<<!ATTLIST jdbc-resource  jndi-name CDATA     #REQUIRED

pool-name CDATA     #REQUIRED

enabled %boolean; 'true'>
```

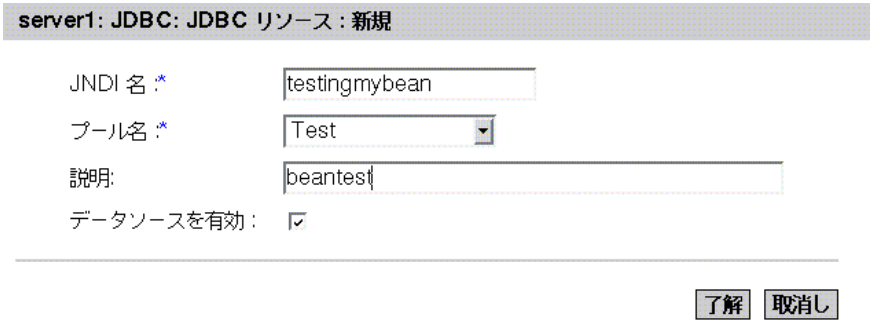
すべての指定は、J2EE アプリケーションの内部からアプリケーションがこのデータソースを参照するときに使われる象徴的な名前です。pool-name 属性は、名前がつけられたプールの定義を参照し、データベースとの接続に必要なすべての設定はここで指定されます。有効な属性は、管理者が一部のリソースをオフにする場合に利用できます。

管理インタフェースによるリソースの登録

管理インタフェースを使ってデータソースを登録するには、次の手順を実行します。

1. 管理インタフェースの左側のペインで、JDBC リソースを登録するアプリケーションサーバーインスタンスを開きます。
2. 「JDBC」を展開します。
3. 「JDBC」の下の「JDBC リソース」をクリックします。
4. 右側のペインの「新規」をクリックします。次のような JDBC リソースを新規作成するためのページが右側のペインに表示されます。

JDBC リソースの新規作成



5. 作成するリソースの JNDI 名を「JNDI 名」フィールドに入力します。

   JDBC リソースは JNDI リポジトリに格納され、アクセスには JNDI 名を使います。JNDI 名の明示的なルートは `Java:comp:env/` なので、名前にこの部分を含める必要はありません。指定する JNDI 名が `jdbc/EmployeeDB_DS` に近くなるように、`jdbc` サブコンテキストの下に JDBC リソースを格納することをお勧めします。
6. 新しいデータソースのプール名を「プール名」ドロップダウンリストから選択します。このリストには、登録されているすべての接続プールが表示されます。選択したプール名は、名前がつけられたプールの定義を参照し、データベースとの接続に必要なすべての設定が指定されます。1 つのプール定義を複数の JDBC が利用することができます。JDBC 接続プールの設定については、271 ページの「管理インタフェースによる JDBC 接続プールの新規作成」を参照してください。
7. データソースの目的を「説明」フィールドに簡単に入力します。250 文字以内で入力する必要があります。
8. 「データソースを有効」ボックスにチェックマークをつけて、データソースを有効にします。無効にするときは、マークを外します。これが有効でない限り、データソースを使ってデータベースに接続することはできません。
9. 「了解」をクリックして新しいデータソースを登録するか、「キャンセル」をクリックして新しいデータソースをキャンセルします。「キャンセル」をクリックすると、JDBC リソースのメインページに戻ります。新しいデータソースの作成は、このページから再開できます。

JDBC 接続について

接続オブジェクトはデータベースとの接続を表します。接続セッションには、実行される SQL ステートメント、およびその接続を介して返される結果が含まれます。1 つのアプリケーションは、1 つのデータベースとの間に 1 つまたは複数の接続を持つことも、複数の異なるデータベースとの間に複数の接続を持つこともできます。

ユーザーは `Connection.getMetaData` メソッドを呼び出して、接続オブジェクトのデータベースの情報を取得できます。このメソッドは、データベースの表、サポートしている SQL 文法、ストアドプロシージャ、接続の機能などの情報を含む `DatabaseMetaData` オブジェクトを返します。

アプリケーションは、DataSource オブジェクトが生成する接続オブジェクトを使います。例外がスローされた場合でも接続が確実に閉じるように、アプリケーションには常に「finally (最終)」ブロックが含まれている必要があります。プールされた接続が接続オブジェクトである場合は、有効な接続は常に利用可能な接続のプールに戻されるため、これは特に重要になります。次のコードは、接続が有効な場合に接続を閉じる最終ブロックの例です。con が接続オブジェクトです。

```
finally{
        if (con != null) con.close();
}
```

次の例に示すように、`finally` ブロックは try ブロックと catch ブロックの後に記述されます。`ds` は DataSource オブジェクトです。

```
try {
        Connection con = ds.getConnection("user", "secret");
        // . . . アプリケーションの処理を実行するコード
} catch {
        // . . . SQLException を処理するコード
} finally {
        if (con != null) con.close();
}
```

この節には次の項目があります。

- JDBC URL について
- JDBC 接続プールの設定
- 接続プールについて
- JDBC 接続プールの監視

- 接続の共有について

JDBC URL について

URL (Uniform Resource Locator) は、インターネット上のリソースを特定するための情報を提供します。これをアドレスと見なすこともできます。

JDBC URL は、適切なドライバがデータソースを認識し、接続を確立できるように、データソースを識別します。ドライバの開発者は、特定のドライバを識別する JDBC URL を実際に決定します。ユーザーは、JDBC URL の形式を気にする必要はなく、使用するドライバと共に供給された URL を使用するだけです。JDBC の役割は、JDBC URL の構造に適用される規約をドライバ開発者に伝えることです。

JDBC URL はさまざまなドライバで使われるため、構造に関する規約もとても柔軟です。まず、各種のドライバが異なるスキームを使ってデータベースに名前をつけることができます。たとえば、ODBC サブプロトコルでは、属性値を含む URL を作成できます (必須ではありません)。

次に、JDBC URL では、ドライバの開発者は必要なすべての接続情報をドライバにエンコードすることができます。たとえば、ユーザーがシステム管理タスクを実行することなく、指定のデータベースと会話するアプレットがデータベース接続を開くようにすることができます。

第三に、JDBC URL は間接レベルに対応しています。つまり JDBC URL は、ネットワークネーミングシステムによって実際の名前に動的に変換される論理ホスト名またはデータベース名を参照できます。これにより、システム管理者は JDBC 名の一部に特定のホストを指定する必要がなくなります。ネットワークネーミングシステムは多様であり、どれを使用するかについて制約はありません。

JDBC URL の標準的な構文は次のとおりです。3 つの部分から構成され、それぞれはコロンで区切られています。

`jdbc:<`*subprotocol*`>:<`*subname*`>`

JDBC URL の 3 つの部分の内容は、次のとおりです。

- jdbc プロトコル

  JDBC URL のプロトコルは、常に jdbc です。

- `<subprotocol>`

  1 つまたは複数のドライバがサポートするデータベース接続メカニズムのドライバ名またはデータベース名です。代表的なサブプロトコルに ODBC があります。これは、ODBC スタイルのデータソース名を指定する URL 用に予約されています。たとえば、JDBC-ODBC ブリッジを経由してデータベースにアクセスするには、`jdbc:odbc:fred.` のような URL を使用します。

  この例では、サブプロトコルは ODBC で、ローカル ODBC データソース (サブネーム) は fred です。

ネットワークネーミングサービスを利用して、JDBC URL に実際のデータベース名を指定しない場合は、ネーミングサービスがサブプロトコルとなります。たとえば、次のような URL が例としてあげられます。

```
jdbc:dcenaming:accounts-payable
```

この例では、URL はローカル DCE ネーミングサービスが指定されており、このサービスは、実際のデータベースとの接続に利用できるように、accounts-payable というデータベース名をより具体的な名前に解決します。

- `<subname>`:

  データソースを識別します。サブネームはサブプロトコルによって異なり、ドライバ開発者が選ぶ任意の内部構文を持つことができます。これには `sub-subname` も含まれます。`subname` で重要なことは、データソースを特定するのに十分な情報を持たせることです。前述の例では、残りの情報が ODBC から提供されるため、fred で十分です。ただし、リモートサーバー上のデータソースを特定するには、より多くの情報が必要となります。たとえば、インターネットを介してデータソースにアクセスする場合、次の標準 URL 命名規約に準拠して、`subname` の一部としてネットワークアドレスを JDBC URL に指定する必要があります。

  //hostname:port/subsubname

  インターネット上のホストに接続するためのプロトコルを `dbnet` とした場合、JDBC URL は次のようになります。

  `jdbc:`*dbnet*`://`*wombat:356/fred*

JDBC 接続プールの設定

Sun ONE Application Server では、名前をつけた JDBC 接続プールを作成できます。JDBC 接続プールは、接続プールの作成に適用されるプロパティを定義します。プールの定義には名前がつけられ、複数の JDBC リソースの設定にこの定義を何度も利用できます。名前をつけたプール定義は、それぞれがサーバー起動時に物理的なプールをインスタンス化します。複数の JDBC リソースが同じプール定義を参照する場合、それぞれが実行時に同じ接続プールを利用します。

次の各項で説明するように、管理インタフェースまたはコマンド行インタフェースを使って JDBC 接続プールを作成、設定できます。

- 管理インタフェースによる JDBC 接続プールの新規作成
- コマンド行インタフェースによる JDBC 接続プールの新規作成
- コマンド行インタフェースによる JDBC 接続プールの管理

管理インタフェースによる JDBC 接続プールの新規作成

管理インタフェースを使って新しい JDBC 接続プールを作成するには、次の手順を実行します。

1. 管理インタフェースの左側のペインで、新たに JDBC 接続プールを作成する Sun ONE Application Server インスタンスを開きます。
2. Sun ONE Application Server の下にリスト表示される J2EE サービスから JDBC を選択し、その下の「接続プール」タブを開きます。管理インタフェースの右側のペインに「JDBC 接続プールの新規作成」のページが表示されます。

JDBC 接続プールの新規作成

3. 作成する接続プールの JNDI 名を「名前」フィールドに入力します。
4. 「グローバルトランザクションのサポート」ボックスにチェックマークをつけて、新しい接続プールのグローバルトランザクションサポートを有効にします。グローバルトランザクションに関与できる接続プールを XA 対応接続プールと呼びます。
5. 「データベースベンダー」ドロップダウンリストからデータベースベンダーを選択し、「新規」をクリックします。表示される次の画面で接続プールを設定する必要があります。

接続プールの設定

接続プールを設定するには、271 ページの「管理インタフェースによる JDBC 接続プールの新規作成」の手順 1 ～手順 5 までを実行します。手順 5 で「新規 (New)」をクリックすると、管理インタフェースの右側のペインに新しいページが表示されます。このページには次の項目があります。

- 一般
- プロパティ
- プール設定
- 接続検証

- トランザクション遮断

このページの「一般」セクションでは、次の表に示すガイドラインに基づいてパラメータの値を指定します。

一般設定

パラメータ	説明
属性名	接続プールの名前
データソースクラス名	DataSource API、XADataSource API、あるいはその両方を実装するベンダー固有のクラス名
説明	接続プールの説明

このページの「プロパティ」セクションでは、標準または固有の JDBC 接続プロパティを指定します。多くのプロパティは指定が必須ではありません。デフォルトでは、すべての標準プロパティの名称が表示されます。どの標準プロパティ、およびどのベンダー固有プロパティの指定が必要であるかを確認するには、データベースベンダーが提供するマニュアル等の資料を参照してください。

このページの「プール設定」セクションでは、次の表に示すガイドラインに基づいてパラメータの値を指定します。

接続プールの設定

パラメータ	説明
通常プールサイズ	プールで維持する接続の最小数を指定する。要求スレッドに接続が渡されると、その接続はプールから除去され、プールサイズは小さくなる。このプールサイズは、サーバー起動時にプールに追加するエントリ数も参照する
最大プールサイズ	一度にプールで維持できる接続の最大数を指定する
プールサイズ変更量	プールのサイズが通常プールサイズに近づくと、プールサイズが一括処理で変更される。この値は、一括処理のサイズを決定する。過大な値を設定すると接続の再利用が遅れ、過小な値を設定すると効率が落ちる。プールの容量は一度に 1 つの接続だけが増分されるため、このフィールドの設定はプール容量の増大には影響しない
アイドルタイムアウト (秒)	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する。この時間を過ぎると、プールの実装はこの接続を閉じることができる

接続プールの設定 (続き)

パラメータ	説明
最大待ち時間	接続がタイムアウトになる前に、呼び出し側が待つ時間を指定する。デフォルト値は long で、呼び出し側の応答待ち時間は長く設定されている

このページの「接続検証」セクションと「トランザクション遮断」セクションでは、次の表に示すガイドラインに基づいて接続プールの検証方法とトランザクション遮断方法を指定します。

接続検証とトランザクション遮断

パラメータ	説明
接続検証が必要	このフィールドにチェックマークをつけると、アプリケーションに渡される前に接続が検証される。これにより、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にアクセス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る。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る
検証方法	アプリケーションサーバーには、データベース接続の検証方法が 3 種類用意されているので、データベースの機能を理解した上で適切な方法を選択する必要がある。検証方法は、次の 3 種類である • `auto-commit`、`meta-data` - con.getAutoCommit() メソッドと con.getMetaData() メソッドは、接続の検証に広く利用されているが、JDBC ドライバの多くは呼び出しの結果をキャッシュするため、検証結果を常に信頼できるとは限らない。呼び出しがキャッシュされるかどうかについて、ベンダーに問い合わせる必要がある • `table` : この方法では、ユーザーが指定した表に対してアプリケーションサーバーがクエリを実行する必要がある。実際のクエリは「select (count *) from <table-name>」である。表は実在し、アクセス可能である必要があるが、行は必要ない。多くの行を持つ既存の表や、頻繁にアクセスされる表を使用するべきではない
表名	「検証方法」ドロップダウンリストで `table` オプションを選択した場合は、表の名前をここに指定する
すべての接続を再確立	1 つの接続が失敗した場合に、プールのすべての接続を終了し、再確立するときは、このボックスにチェックマークをつける。チェックマークを外した場合は、接続の使用時に個別に再確立される

接続検証とトランザクション遮断 (続き)

パラメータ	説明
トランザクション遮断	接続のトランザクション遮断レベルを選択できる。指定しない場合は、JDBC ドライバによって設定されるデフォルトの分離レベルがプールに適用される
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る。これにより、プールから取得されるすべての接続に同じ遮断レベルが適用される。たとえば、最後の使用時に `con.setTransactionIsolation` など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遮断レベルは指定レベルに戻される

コマンド行インタフェースによる JDBC 接続プールの新規作成

ここでは、コマンド行インタフェースによる JDBC 接続プールの新規作成について、例を使って説明します。

次の表は、サーバー名やパスワードなど、接続プールの作成に必要なすべてのオプションを示しています。また、値の例も示しています。この項で説明するコマンドを実行する前に、Sun ONE Application Server のインストールに固有のパラメータを手元に準備しておくことをお勧めします。

コマンド行インタフェースによる JDBC 接続プールの新規作成に必要なオプション

必要オプションの説明	値の例
管理サーバーの管理ユーザー名	*admin*
管理サーバーの管理パスワード	*adminadmin*
アプリケーションサーバーの管理ポート	*8888*
アプリケーションサーバーのマシン名	*sas.sun.com*
アプリケーションサーバーのインスタンス名	*server1*
接続プールのデータソースクラス名	*oracle.jdbc.xa.client.OracleXADataSource* 注 : 接続プールを作成するデータベースのデータソースクラス名を指定する。この例で使われているデータベースは Oracle である
JDBC リソースの説明	*Jdbc Resource*
接続プールの説明	*Jdbc Connection Pool*
JDBC リソースの名前	*jdbc/SampleJdbcResource*

コマンド行インタフェースによる JDBC 接続プールの新規作成に必要なオプション (続き)

必要オプションの説明	値の例
管理サーバーの管理ユーザー名	*admin*
接続プールの名前	*SampleJdbcConnectionPool*
データベースユーザーの名前	*oracle*
データベースのパスワード	*oracle*
JDBC 接続 URL	*jdbc:oracle:thin:@oracleserver.sun.com:1521:ORA*

次の例は、表「コマンド行インタフェースによる JDBC 接続プールの新規作成に必要なオプション」に示した変数の用例です。

例 1

この例は、`SampleJdbcConnectionPool` という JDBC 接続プールを作成します。次のように、この例では 2 段階の処理で JDBC 接続プールを作成しています。

- 第 1 段階 - 接続プールの作成
- 第 2 段階 - インスタンスへの変更の適用

第 1 段階 - 接続プールの作成

JDBC 接続プールを作成するためのコマンド行インタフェースの構文は、次のとおりです。

```
asadmin create-jdbc-connection-pool --user admin_user [--password
admin_password] [--host localhost] [--port 4848] [--secure | -s]
[--instance instancename] --datasourceclassname classname [--restype
res_type] [--steadypoolsize 8] [--maxpoolsize 32] [--maxwait 60000]
[--poolresize 2] [--idletimeout 300] [--isolationlevel isolation_level]
[--isisolationguaranteed] [--isconnectvalidatereq=false]
[--validationmethod auto-commit] [--validationtable tablename]
[--failconnection=false] [--description text] [--property
(name=value)[:name=value]*] connectionpool_id
```

たとえば、次のコマンドは `SampleJdbcConnectionPool` という接続プールを作成します。

```
asadmin create-jdbc-connection-pool --user admin --password
adminadmin --host sas.sun.com --port 8888 --instance server1
--restype javax.sql.XADataSource --datasourceclassname
oracle.jdbc.xa.client.OracleXADataSource --description "Sample Jdbc
Connection Pool" --property
User="oracle":Password="oracle":URL="jdbc¥:oracle¥:thin¥:@oracleserv
er.sun.com¥:1521¥:ORA" SampleJdbcConnectionPool
```

注 新しい接続プールの「グローバルトランザクションサポート」を有効にするときは、`--restype javax.sql.XADataSource` を設定します。URL プロパティのコロン (`:`) を (`¥:`) に置き換えます。

JDBC 接続プールの作成が完了すると、次のメッセージが表示されます。

```
Created the JDBC connection pool resource with id =
SampleJdbcConnectionPool
```

第 2 段階 - インスタンスへの変更の適用

これで JDBC 接続プールを作成できました。次に、変更を Sun ONE Application Server の現在のインスタンスに適用する必要があります。

Sun ONE Application Server のインスタンスに変更を適用する構文は、次のとおりです。

```
asadmin reconfig --user admin_user [--password admin_password] [--host
localhost] [--port adminport] [--secure | -s]
[--discardmanualchanges=false|--keepmanualchanges=false] instancename
```

たとえば、次のコマンドは Sun ONE Application Server のインスタンス *server1* に変更を適用します。

```
asadmin reconfig --user admin --password adminadmin --host sas.sun.com
--port 8888 server1
```

Sun ONE Application Server のインスタンスに変更が適用されると、次のメッセージが表示されます。

```
Successfully reconfigured
```

コマンド行インタフェースによる JDBC 接続プールの管理

この項で説明するように、コマンド行インタフェースを使って JDBC 接続プールとそのプロパティを管理できます。

接続プールのリスト表示 : 次のコマンドは、第 2 段階で使用した Sun ONE Application Server のインスタンス *server1* に作成されているすべての接続プールをリスト表示します。

```
asadmin list-jdbc-connection-pools --user admin --password adminadmin
--host sas.sun.com --port 8888 server1
```

JDBC 接続プールのプロパティの変更 : `maxPoolSize` など、JDBC 接続プールのプロパティを次のような手順で変更できます。

1. 次のコマンドを実行し、JDBC 接続プールの属性 `maxPoolSize` に指定されている値を取得します。

   ```
   asadmin get -u admin -w adminadmin -H sas.sun.com -p 8888
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.maxPoolSize
   ```

   このコマンドを実行すると、次の結果が表示されます。

   ```
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.maxPoolSiz
   e = 32
   ```

   次のコマンドを実行し、MaxPoolSize の値を 80 に変更します。

   ```
   asadmin set -u admin -w adminadmin -H sas.sun.com -p 8888
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.maxPoolSize="80"
   ```

   値の設定が完了すると、次のメッセージが表示されます。

   ```
   Attribute maxPoolSize set to 80
   ```

2. 次のコマンドを実行して、Sun ONE Application Server のインスタンスに変更を適用します。

   ```
   asadmin reconfig --user admin --password adminadmin --host sas.sun.com
   --port 8888 server1
   ```

User プロパティの変更 : 次のコード例は、`User` プロパティの値を `oracle` から `System` に変更します。

```
asadmin create-jdbc-connection-pool --user admin --password adminadmin
--host sas.sun.com --port 8888 --instance server1 --restype
javax.sql.XADataSource --datasourceclassname oracle.jdbc.xa.client.OracleXADataSource
--description "Sample Jdbc Connection Pool" --property
User="oracle":Password="oracle":URL="jdbc¥:oracle¥:thin¥:@oracleserver.sun.com¥:1521¥:
ORA" SampleJdbcConnectionPool
```

1. 次のコマンドを実行して、User プロパティを変更します。

   ```
   asadmin set -u admin -w adminadmin -H sas.sun.com -p 8888
   server1.jdbc-connection-pool.SampleJdbcConnectionPool.property.User="System"
   ```

   ユーザーの名前が *Oracle* から *System* に変更されます。

2. ユーザー名を変更したら、次のコマンドを実行して変更を適用します。

   ```
   asadmin reconfig --user admin --password adminadmin --host
   sas.sun.com --port 8888 server1
   ```

SampleJdbcResource という JDBC リソースの作成 : 次の方法で、JDBC リソースを作成できます。JDBC リソースを作成する構文は次のとおりです。

```
asadmin create-jdbc-resource --user admin_user [--password
admin_password] [--host localhost] [--port 4848] [--secure | -s]
[--instance instancename] --connectionpoolid id [--enabled=true]
[--description text] [--property (name=value)[:name=value]*] jndiname
```

1. 次のコマンドを実行して、SampleJdbcResource という JDBC リソースを作成します。

   ```
   asadmin create-jdbc-resource --user admin --password adminadmin
   --host sas.sun.com --port 8888 --instance server1 --description "Sample
   Jdbc Resource" --connectionpoolid SampleJdbcConnectionPool
   jdbc/SampleJdbcResource
   ```

   このコマンドを実行すると、JDBC リソースが作成され、次のメッセージが表示されます。

   ```
   Created the external JDBC resource with jndiname =
   jdbc/SampleJdbcResource
   ```

2. 次のコマンドを実行して、Sun ONE Application Server のインスタンスに変更を適用します。

   ```
   asadmin reconfig --user admin --password adminadmin --host sas.sun.com
   --port 8888 server1
   ```

3. 次のコマンドを実行して、*server1* インスタンスのすべての JDBC リソースをリスト表示します。

   ```
   asadmin list-jdbc-resources --user admin --password adminadmin
   --host sas.sun.com --port 8888 server1
   ```

接続プールについて

接続を取得するときに、アプリケーションはまず JNDI を使って DataSource をルックアップします。この場合のコード例は、次のようになります。

```
InitialContext ctx  = new InitialContext();

DataSource ds = (DataSource)
ctx.lookup("java:comp/env/jdbc/employee_ds");
```

DataSource を取得すると、アプリケーションコンポーネントは J2EE 配備記述子の <res-auth> 要素に設定されている値に応じて 2 つの方法で接続を取得できるようになります。この要素の値が Container であれば、アプリケーションは ds.getConnection() メソッドを使って接続を取得できます。この場合、サインオン情報は必要ありません。それ以外の値が設定されているときは、ds.getConnecion などのリソースマネージャから接続を取得するために、アプリケーションはサインオン情報 (*userName, password*) を指定する必要があります。

getConnection() へのすべての要求は、プールから供給されます。JDBC 接続プールは、server.xml に指定されているパラメータセットに基づいて作成されます。作成されたプールには、すぐに利用できる多数の接続が含まれます。このため、プールで現在利用できる接続によって ds.getConnection() 要求は満たされます。それ以前の接続がプールに返されなかった場合は、プールが空であるため、次の要求では増分の接続が作成されます。接続の作成は、プールに設定されている最大接続数によって制限されます。プールの実装は、作成された接続の数を追跡しています。getConnection() 要求に対して、プールが空であると見なされた場合、または作成した接続の数がプールの最大接続数と等しい場合は、その要求はブロックされます。これは、プールの共有が無効に設定されている場合にだけ生じる現象で、接続がプールに返されるまで継続します。

サーバーが稼働している限り、データベースがクラッシュしてから復旧した場合でも、接続プールは正常に機能し続けます。これは、272 ページの「接続プールの設定」で説明した接続検証を有効にした場合にだけ利用できる機能です。

「検証方法」ドロップダウンリストで選択した値に応じて、プールの実装プログラムは次のパラメータを実行します。

- 接続検証タイプを auto-commit に設定した場合、システムは conn.getAutoCommit() メソッドを実行して接続が有効であるかどうかを確認する。メソッドが SQLException をスローしない場合は、接続は有効であると見なされる。auto-commit は、このパラメータのデフォルトオプションである
- 接続検証タイプを meta-data に設定した場合、接続のメタデータを調べるために conn.getMetaData() メソッドが実行される。SQLException がスローされない場合は、接続は有効であると見なされる
- 接続検証タイプを table に設定した場合、クエリ「Select * From <table-name>」が実行される。SQLException がスローされない場合は、接続は有効であると見なされる。

fail-all-connections (すべての接続を再確立) プロパティを有効にしたときは、プール内のいずれかの接続が無効な場合にすべての接続が閉じられ、再確立されます。それ以外の場合は、個々の接続の利用時に接続の中止と再確立が行われます。

プールの実装には、プールで利用できるすべての接続を再利用する機能もあります。指定したアイドル期間を過ぎると、アイドル状態の接続は閉じられ、プールのサイズは通常サイズに戻ります。プールのアイドル状態が長く続いた場合、プール内に通常数の利用可能な接続を維持するために、コンテナは古い接続を再確立する必要があります。プールの通常サイズと最大サイズを決定するときは、この点に注意する必要があります。

JDBC 接続プールの監視

プールサイズの設定が適切であるかを判断するには、プールの動作を定期的に監視します。次の表は、監視できる JDBC 接続プールパラメータの一覧です。

ただし、監視を有効にするメカニズムや監視可能な属性は、将来のリリースで向上する可能性があります。

監視可能な JDBC 接続プールのパラメータ

属性名	データ型	説明
total-threads-waiting	整数	JDBC 接続を待機するスレッド総数
total-outbound-connections	整数	JDBC 接続検証の失敗総数
total-connections-timed-out	整数	タイムアウトになった接続要求の総数

接続の共有について

同じリソースマネージャを使う J2EE アプリケーションが複数の接続を必要とする場合、同じトランザクションの範囲内で接続を共有させることができます。トランザクションの範囲内という言葉の意味については、次の例を参考にしてください。

Bean_A がトランザクション (Tx1) を開始し、接続を取得します。次に、Bean_A は同じトランザクション (Tx1) で Bean_B 内のメソッドを呼び出します。Bean_B が同じ `DataSource` からの接続を必要とし、同じサインオン情報が必要となる場合は、同じ接続が共有され、Bean_A だけがトランザクションを完了することは明白です。また、接続の共有は、J2EE 配備記述子でリソースの共有が `Shareable` に設定されている場合にだけ行われます。接続の共有が適さない場合は、配備記述子でリソースの共有を `Unshareable` に設定します。Sun ONE Application Server では、パフォーマンス向上のために接続の共有をサポートしています。

JDBC トランザクションについて

トランザクションは、実行、完了され、さらにコミットまたはロールバックされた 1 つまたは複数のステートメントから構成されます。`commit` メソッドまたは `rollback` メソッドが呼び出されると、現在のトランザクションが終了され、新しいトランザクションが開始されます。

一般に、新しい接続オブジェクトはデフォルトでは auto-commit モードに設定されているため、ステートメントが完了すると、そのステートメントで `commit` メソッドが自動的に呼び出されます。この場合、各ステートメントは個別にコミットされるため、トランザクションは 1 つのステートメントだけから構成されます。auto-commit モードを無効にすると、`commit` メソッドまたは `rollback` メソッドが明示的に呼び出さ

れるまでトランザクションは終了しません。このため、いずれかのメソッドを最後に呼び出してから実行されたすべてのステートメントがトランザクションに含まれます。この場合、トランザクションのすべてのステートメントはグループとしてコミットまたはロールバックされます。

commit メソッドは、SQL ステートメントがデータベースに加えたすべての変更を永続化し、そのトランザクションが保持するすべてのロックを解放します。rollback メソッドは、このような変更を破棄します。

1 つのトランザクションに 2 つのアップデートが含まれる場合、一方のアップデートが反映されなければ、もう一方のアップデートも反映させたくないことがあります。これは、auto-commit を無効にして、2 つのアップデートを 1 つのトランザクションにグループ化することで可能になります。両方のアップデートが成功した場合は、commit メソッドが呼び出され、両方のアップデートが永続化されます。一方または両方のアップデートが失敗した場合は、rollback メソッドが呼び出され、値はどちらのアップデートも実行される前の状態に戻されます。ほとんどの JDBC ドライバはトランザクションをサポートしています。

javax.sql パッケージに含まれるクラスとインタフェースは、接続オブジェクトを複数の DBMS サーバーに接続する分散トランザクションの一部として利用できます。分散トランザクションで接続オブジェクトを利用するには、中間層サーバーの分散トランザクションインフラストラクチャに対して働きかけるように実装された DataSource オブジェクトが接続オブジェクトを生成する必要があります。DriverManager によって生成される接続オブジェクトとは異なり、このような DataSource オブジェクトが生成する接続オブジェクトの auto-commit モードはデフォルトで無効に設定されます。一方、DataSource オブジェクトの標準的な実装は、DriverManager クラスによって生成される接続オブジェクトとまったく同じオブジェクトを生成します。

分散トランザクションの一部に接続オブジェクトが使われている場合、commit メソッドまたは rollback メソッドの実行タイミングはトランザクションマネージャによって決定されます。このため、接続オブジェクトが分散トランザクションに参加している場合は、Connection.commit メソッドや Connection.rollback メソッドの呼び出し、接続の auto-commit モードの有効化など、接続の開始と終了に影響する処理をアプリケーションは一切実行できません。このような処理は、トランザクションマネージャによる分散トランザクションの処理を妨害します。

JavaMail リソースについて

JavaMail API は、メッセージストアに格納されている電子メールメッセージにアクセスしたり、メッセージトランスポートを使って電子メールメッセージを作成および送信したりするための API です。インターネット標準 MIME メッセージに固有のサポートも含まれます。メッセージストアとトランスポートへのアクセスには、ストアとトランスポートに固有のプロトコルをサポートするプロトコルプロバイダを使って行われます。JavaMail API 仕様は特定のプロトコルプロバイダを必要としませんが、JavaMail には IMAP メッセージストアプロバイダと SMTP メッセージトランスポートプロバイダが含まれます。

JavaMail API は、メールシステムを構成するオブジェクトを定義する抽象クラスのセットを提供します。この API は、Message、Store、Transport などのクラスを定義します。API を拡張したり、サブクラスに分割することで、必要に応じて新しいプロトコルや機能を追加できます。さらに、この API は抽象クラスの具体的なサブクラスも提供します。これらのサブクラスには `MimeMessage` や `MimeBodyPart` が含まれ、一般に広く利用されているインターネットメールプロトコルを実装しています。

JavaMail API は、IMAP、MAPI、CMC、c-client、およびその他の電子メールメッセージングシステム API から多くを得ています。JavaMail API は、各種メッセージングストア、各種メッセージ形式、各種メッセージトランスポートなど、さまざまなメッセージングシステムの実装をサポートしています。JavaMail API は、クライアントアプリケーションの API を定義する基本的なクラスとインタフェースのセットを提供します。開発者は、JavaMail クラスをサブクラスに分割し、IMAP、POP3、SMTP など、特定のメッセージングシステムの実装を提供することができます。

この節では次の項目について説明します。

- JavaMail によるメッセージ処理のプロセスについて
- JavaMail のアーキテクチャコンポーネントについて
- JAF (JavaBeans Activation Framework) について
- JavaMail の設定パラメータについて
- JavaMail セッション参照の J2EE 配備記述子
- Sun ONE Application Server 配備記述子のエントリ
- JavaMail セッションの新規作成
- リソースの詳細プロパティの設定

JavaMail によるメッセージ処理のプロセスについて

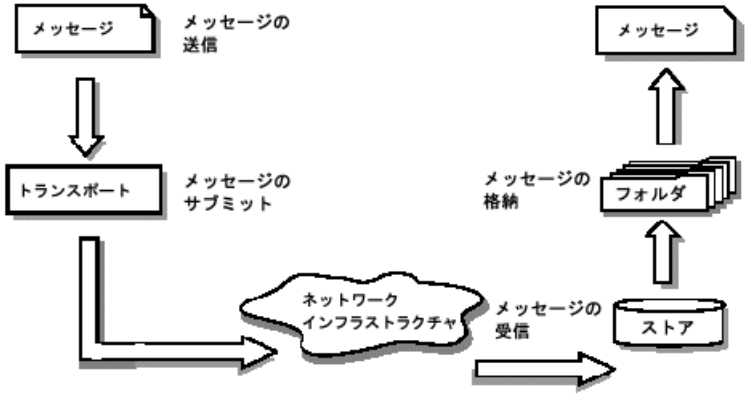
JavaMail API は、一般的なクライアントアプリケーションの標準的なメール処理プロセスを構成する、次の機能を実行します。

- ヘッダー属性の集合、および Content-Type ヘッダーフィールドに指定されたデータタイプのデータブロックから構成されたメールメッセージを作成する。JavaMail は、Part インタフェースと Message クラスを使ってメールメッセージを定義する。メッセージにデータを含めるときは、JAF 定義による DataHandler オブジェクトを使用する
- ユーザーを認証し、メッセージストアとメッセージトランスポートへのアクセスを制御するセッションオブジェクトを作成する
- 受信者リスト宛てにメッセージを送信する
- メッセージストアからメッセージを取得する
- 取得したメッセージに対して高レベルのコマンドを実行する。表示や印刷などの高レベルコマンドは、JAF が認識する JavaBeans による実装を前提とする

注 現時点では、JavaMail のフレームワークは、メッセージ配信、セキュリティ、接続解除状態での処理、ディレクトリサービス、フィルタ機能をサポートするメカニズムを定義していません。

次の図は、JavaMail API によるメッセージ処理プロセスを示しています。

この図は、JavaMail API のメッセージ処理プロセスを示しています。



JavaMail API の設定は、静的なファクトリメソッドを使って javax.mail.Session を作成することで行われます。Sun ONE Application Server は JNDI を使ってセッションオブジェクトを要求し、セッションオブジェクトが必要であることを配備記述子の resource-ref 要素に記録します。JavaMail API セッションオブジェクトは、リソースファクトリと見なされます。

javax.mail.internet.InternetAddress タイプのアドレスと、javax.mail.internet.MimeMessage タイプのメッセージを処理できるメッセージトランスポートが提供されます。デフォルトのメッセージトランスポートは、javax.mail.Transport クラスの送信メソッドを使ってこのようなメッセージを送信できるように適切に設定する必要があります。

JavaMail API の抽象層は、すべてのメールシステムがサポートするメール処理に対応したクラス、インタフェース、および抽象メソッドを宣言します。抽象層を構成する API 要素は、標準のデータタイプをサポートするために、必要に応じてサブクラスに分割および拡張されます。また、必要に応じて、メッセージアクセスプロトコルとメッセージトランスポートプロトコルのインタフェースとして機能します。

インターネット実装層は、インターネット標準の RFC822 と MIME を使って抽象層の一部を実装します。

JavaMail のアーキテクチャコンポーネントについて

ここでは、JavaMail のアーキテクチャを構成する次の主要コンポーネントについて説明します。

- Message クラス
- メッセージの格納と取得
- メッセージの構成とトランスポート

Message クラス

Message クラスは、メールメッセージの属性セットとコンテンツを定義する抽象クラスです。Message クラスの属性は、アドレス情報を指定し、コンテンツの構造を定義します (コンテンツタイプを含む)。コンテンツは、実際のデータを内包した DataHandler オブジェクトとして表されます。

Message クラスは Part インタフェースを実装します。Part インタフェースは、Message オブジェクトによって運ばれるデータコンテンツの定義と書式設定に必要な属性、およびメールシステムとのインタフェースの成功に必要な属性を定義します。Message クラスは、メッセージトランスポートシステムを経由したメッセージのルー

ティングに必要な From、To、Subject、Reply-To などの属性を追加します。フォルダに含まれる Message オブジェクトには、関連するフラグのセットがあります。JavaMail は、特定のメッセージング実装をサポートする Message サブクラスを提供します。

メッセージのコンテンツはバイトの集合、またはバイトの集合に対する参照で、Message オブジェクト内にカプセル化されます。JavaMail は、メッセージコンテンツのデータタイプや形式を認識できません。Message オブジェクトは、JAF (JavaBeans Activation Framework) という中間層を通じてコンテンツと対話します。この分離によって、Message オブジェクトはあらゆる種類のコンテンツを処理できます。また、同じ API メソッドを呼び出すことで任意の適切な転送プロトコルを使ってコンテンツを転送できます。メッセージの受信側は、通常はコンテンツのデータタイプと形式を認識し、コンテンツの処理方法を理解できます。

JavaMail API は、各 Bodypart がそれぞれの属性とコンテンツを定義する複数パートの Message オブジェクトもサポートしています。

メッセージの格納と取得

メッセージは Folder オブジェクトに格納されます。Folder オブジェクトにはサブフォルダだけでなくメッセージも格納できるので、フォルダ階層のようなツリー構造になります。Folder クラスは、メッセージをフェッチ、修正、コピー、および削除するメソッドを宣言します。Folder オブジェクトは、イベントリスナーとして登録されているコンポーネントにイベントを送信することもできます。

Store クラス

Store クラスは、フォルダ階層とメッセージを格納するデータベースを定義します。また、Store クラスは、フォルダにアクセスして格納されているメッセージを取得するためのアクセスプロトコルも指定します。Store クラスは、データベースへの接続の確立、フォルダのフェッチ、および接続を閉じるために適用されるメソッドも提供します。メッセージアクセスプロトコル (IMAP、POP3 など) を実装するサービスプロバイダは、Store クラスをサブクラスに分割するところから処理を開始します。通常、ユーザーは特定の Store 実装に接続することでメールシステムとのセッションを開始します。

メッセージの構成とトランスポート

クライアントは、適切な Message サブクラスをインスタンス化して新しいメッセージを作成します。これにより、受信側のアドレスや件名などの属性が設定され、Message オブジェクトにコンテンツが挿入されます。最後に、Transport 送信メソッドが呼び出され、メッセージが送信されます。Transport クラスは、メッセージを送

信先アドレスにルーティングするトランスポートエージェントを作成します。このクラスは、受信者リストにメッセージを送信するメソッドを提供します。Message オブジェクトを指定して Transport 送信メソッドを呼び出すと、送信先アドレスに基づいて適切なトランスポートが識別されます。

Session クラス

Session クラスは、メールを利用できるクライアントとネットワークの間のインタフェースを定義する、グローバルおよびユーザー単位のメール関連プロパティを定義します。

JavaMail システムコンポーネントは、セッションオブジェクトを使って特定のプロパティを設定、取得します。また、Session クラスは、デスクトップアプリケーションによる共有が可能で、デフォルトで認証されるセッションオブジェクトを提供します。Session クラスは、最終の具体的なクラスです。これをサブクラスに分割することはできません。また、Session クラスは、特定のアクセスプロトコルとトランスポートプロトコルを実装する Store オブジェクトと Transport オブジェクトのファクトリとしても機能します。セッションオブジェクトで、適切なファクトリメソッドを呼び出すことで、クライアントは特定のプロトコルをサポートする Store オブジェクトと Transport オブジェクトを取得できます。

JAF (JavaBeans Activation Framework) について

JavaMail は、メッセージデータのカプセル化、およびデータと対話するコマンドの処理に JAF (JavaBeans Activation Framework) を使います。メッセージデータとの対話は JAF が認識する JavaBeans を介して行う必要があり、これは JavaMail API によって提供されません。

JAF の標準拡張を利用することで Java テクノロジを使う開発者は、データの任意の部分のタイプの特定、そのデータへのアクセスのカプセル化、そのデータで利用できる機能の確認、指定処理を実行する適切な Bean のインスタンス化といった標準サービスの利点を活用することができます。たとえば、ブラウザが JPEG 画像にアクセスした場合、このフレームワークによって、ブラウザはデータのストリームを JPEG 画像として認識できます。さらに、特定したタイプに基づいて、その画像を操作または表示できるオブジェクトを探し、インスタンス化することができます。

JAF API は、さまざまな MIME データタイプをサポートしています。Java Mail API には、次の表に示す Java プログラミング言語のタイプに対応する MIME データタイプをサポートするために、`javax.activation.DataContentHandlers` を含める必要があります。

JavaMail API の MIME データタイプと Java のタイプのマッピング

MIME タイプ	Java のタイプ
Text/Plain	java.lang.String
Multipart/	javax.mail.internet.MIME.Multipart
Message/rfc822	javax.mail.internet.MIME.Message

JAF は、MIME データタイプのサポートを Java プラットフォームに統合します。MIME バイトストリームと Java プログラミング言語オブジェクトは、avax.activation.DataContentHandlerobjects を使って相互に変換できます。データの表示や編集など、MIME データを処理する JavaBeans コンポーネントを指定できます。また、JAF にはファイル名の拡張子を MIME タイプにマップするメカニズムも用意されています。JavaMail API は、メッセージに含まれるデータの処理に JAF を使用します。通常の J2EE アプリケーションは JAF を直接使う必要はありませんが、電子メールを利用するアプリケーションを洗練させる場合には必要になることがあります。

JavaMail の設定パラメータについて

Sun ONE Application Server の JavaMail リソースは、次の設定パラメータを使用します。これらの設定パラメータは、`server.xml` ファイルの `mail-resource` 要素から読み込まれる名前と値のペアです。

- JNDI Name

  JNDI 名は、J2EE アプリケーションが参照するこのメールリソースの名前

- Enabled

  `enabled` 設定パラメータは、このメールリソースを JNDI ツリーにパブリッシュし、参照可能にするかどうかを指定する。無効なリソースを参照した J2EE アプリケーションは `NameNotFoundException` 例外を受け取る

- store-protocol

  デフォルトのメッセージアクセスプロトコルを指定する。`Session.getStore()` メソッドは、このプロトコルを実装する Store オブジェクトを返す。クライアントは、`Session.getStore(String protocol)` メソッドを使ってこのプロパティをオーバーライドし、別のプロトコルを明示的に指定できる

- store-protocol class

  上で指定したストアプロトコルを実装するクラスの名前を指定する。このクラスのデフォルト名は `com.sun.mail.imap.IMAPStore`

- transport-protocol

  デフォルトのトランスポートプロトコルを指定する。`Session.getTransport()` メソッドは、このプロトコルを実装する Transport オブジェクトを返す。クライアントは、`Session.getTransport(String protocol)` メソッドを使ってこのプロパティをオーバーライドし、別のプロトコルを明示的に指定できる

- transport-protocol class

  上で指定したトランスポートプロトコルを実装するクラスの名前を指定する。このクラスのデフォルト名は `com.sun.mail.smtp.SMTPTransport`

- host

  デフォルトのメールサーバーを指定する。Store オブジェクトと Transport オブジェクトの接続メソッドは、プロトコル固有の host プロパティが見つからない場合にこのプロパティを使ってターゲットホストを特定する

- user

  メールサーバーへの接続時に渡すユーザー名を指定する。Store オブジェクトと Transport オブジェクトの接続メソッドは、プロトコル固有の username プロパティが見つからない場合にこのプロパティを使ってユーザー名を取得する

- from

  現在のユーザーの返信先アドレスを指定する。`InternetAddress.getLocalAddress` メソッドが現在のユーザーの電子メールアドレスを指定するときに使われる

- debug

  初期デバッグモードを指定する。このプロパティを true に設定すると、デバッグモードがオンになり、false に設定するとオフになる

- mail-<protocol>-host

  プロトコル固有のデフォルトメールサーバーを指定する。これは、`mail.host` プロパティに優先して適用される。このプロパティは、store-protocol 属性の値に応じて設定できる。store-protocol の値が IMAP または POP の場合、プロパティにそれぞれ `mail.imap.host` または `mail.pop3.host` という名前を追加する必要がある。特定のプロパティの値は、メールシステムの設定に合わせて設定する必要がある。たとえば、store-protocol を IMAP に設定した場合、`mail-imap-host` というプロパティ名には `spaceduck.acme.com` という値が追加される

- mail-<protocol>-user

  メールサーバーへの接続に適用される、プロトコル固有のデフォルトユーザー名を指定する。これは、mail.user プロパティに優先して適用される。store-protocol 属性の設定に応じて、このプロパティの値は mail.imap.user または mail.pop3.user となる。たとえば、store-protocol を IMAP に設定した場合、mail-imap-user というプロパティ名には fredbloggs という値が追加される

JavaMail セッション参照の J2EE 配備記述子

JavaMail リソースをサーバーに登録すると、JNDI ルックアップを使って J2EE アプリケーションコンポーネントがこれを参照できるようになります。リソースマネージャ接続ファクトリを参照するアプリケーションを配備するには、コンポーネントプロバイダは、標準の J2EE 1.3 配備記述子にすべてのリソースマネージャ接続ファクトリ参照を宣言する必要があります。

J2EE 1.3 記述子の JavaMail 参照の要素は次のとおりです。

```
<resource-ref>
    <description>
        メールの送信に利用される JavaMail リソース
     </description>
     <res-ref-name>mail/MyMailSession</res-ref-name>
     <res-type>javax.mail.Session</res-type>
     <res-auth>Container</res-auth>
     <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Sun ONE Application Server 配備記述子のエントリ

配備担当者は、メールリソースを参照する配備コンポーネントごとに、コンポーネントで使われるリソースの名前と、ネーミングサービスに DataSource が登録されている実際の JNDI 名をマップする必要があります。配備ツールを使えば、このマッピングは簡単に行えます。このマッピングは、Sun ONE Application Server 固有の xml ファイルに登録されます。次に、このマッピングを含む Sun ONE Application Server 固有の XML の一部を示します。

```
<resource-ref>
   <res-ref-name>mail/MyMailSession</res-ref-name>
   <jndi-name>mail/Session</jndi-name>
</resource-ref>
```

JavaMail セッションの新規作成

管理インタフェースを使って JavaMail セッションを設定できます。新しい JavaMail セッションを作成、設定するには、次の手順を実行します。

1. 管理インタフェースの左側のペインで、新たに JavaMail セッションを作成する Sun ONE Application Server インスタンスを展開します。
2. 「Java メールセッション」をクリックします。管理インタフェースの右側のペインに次のような「JavaMail セッションの設定」のウィンドウが表示されます。

JavaMail セッションの設定

3. 作成している JavaMail セッションの名前を「JNDI 名」テキストフィールドに入力します。JavaMail リソースをサーバーに登録すると、JNDI ルックアップを使って J2EE アプリケーションコンポーネントがこれを参照できるようになります。
4. デフォルトメールサーバーの DNS 名を「メールホスト」テキストフィールドに入力します。Store オブジェクトと Transport オブジェクトの接続メソッドは、プロトコル固有の host プロパティが見つからない場合にこのプロパティを使ってターゲットホストを特定します。

5. メールサーバーへの接続時に渡すユーザー名を「デフォルトユーザー」テキストフィールドに入力します。Store オブジェクトと Transport オブジェクトの接続メソッドは、プロトコル固有の username プロパティが見つからない場合にこのプロパティを使ってユーザー名を取得します。
6. 現在のユーザーのデフォルトの返信先アドレスを「デフォルトの返信用アドレス」フィールドに入力します。デフォルトのアドレスは、「username@host」の形式で指定する必要があります。
7. この JavaMail セッションの説明を「説明」フィールドに入力します。
8. 「Java メールセッションを有効」ボックスにチェックマークをつけて、作成した JavaMail セッションを有効にします。
9. 「了解」をクリックして、新たに設定した JavaMail セッションを保存します。

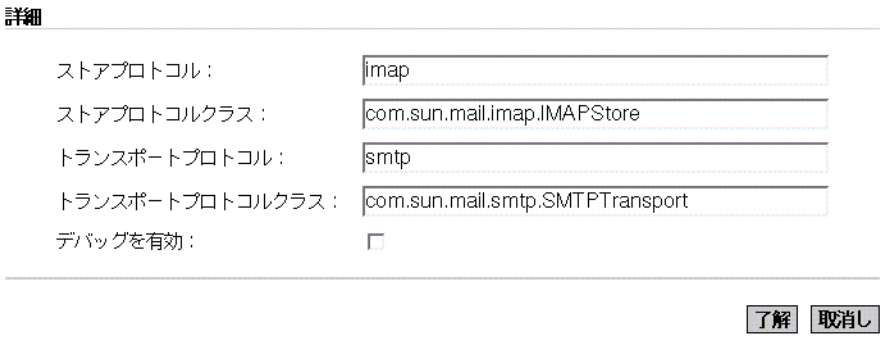
リソースの詳細プロパティの設定

管理インタフェースを使って、新しい JavaMail セッションにいくつかの追加プロパティを設定できます。プロパティの名前と値のペアは、使用するメイルプロトコルによって異なります。また、これらのプロパティを server.xml ファイルに直接設定することもできます。

追加プロパティを設定するには、次の手順を実行します。

1. 管理インタフェースの左側のペインで、設定を変更する JavaMail セッションを含む Sun ONE Application Server インスタンスを展開します。
2. 「Java メールセッション」をクリックします。管理インタフェースの右側のペインの「JavaMail セッションの新規作成」で説明したメイン設定セクションの下に、次のような「JavaMail セッションリソースの追加設定」のウィンドウが表示されます。

JavaMail セッションリソースの追加設定



3. POP3 や IMAP など、この JavaMail セッションに適用するストアプロトコルを「ストアプロトコル」テキストフィールドに入力します。
4. 例に示されるように、指定したストアプロトコルのクラス名を「ストアプロトコルクラス」テキストフィールドに入力します。
5. たとえば SMTP など、JavaMail セッションに適用するトランスポートプロトコルを「トランスポートプロトコル」テキストフィールドに入力します。
6. 例に示されるように、このセッションに指定したトランスポートプロトコルのクラス名を「トランスポートプロトコルクラス」テキストフィールドに入力します。
7. この JavaMail セッションのデバッグを有効にするときは、「デバッグを有効」ボックスにチェックマークをつけます。このボックスにチェックマークをつけると、デバッグモードが有効になります。
8. 「了解」をクリックして、追加プロパティの設定を保存します。

次に、メールリソースのすべての設定の例を示します。

```
<mail-resource
   jndi-name = "mail/Session"
   enabled = "true"
   store-protocol = "imap"
   store-protocol-class = "com.sun.mail.imap.IMAPStore"
   transport-protocol = "smtp"
   transport-protocol-class = "com.sun.mail.smtp.SMTPTransport"
   host = "gopostal.acme.com"
   user = "kingkong"
   from = "kingkong@acme.com"
```

```
    debug = "false">
    <property name = "mail-imap-host" value = "spaceduck.acme.com"/>
    <property name = "mail-imap-user" value = "fredbloggs"/>
</mail-resource>
```

第 11 章

JMS サービスの使用

Sun ONE Application Server は、JMS (Java Message Service) API を使ってメッセージング処理を行うアプリケーションをサポートしています。JMS は、Java アプリケーションが分散環境でメッセージの作成、送受信、および読み取りを行うための共通の方法を提供するプログラミングインタフェースです。

J2EE (Java 2 Enterprise Edition) アプリケーションは、非同期メッセージングの標準ベースの手段として JMS を使用します。このため、J2EE コンポーネント (Web コンポーネントか Enterprise JavaBeans (EJB) コンポーネント) は、JMS API を使って、メッセージ駆動型 Beans (MDB) と呼ばれる特殊な EJB によって非同期で消費されるメッセージを送信できます。

一般に、Sun ONE Application Server では JMS メッセージングをサポートします。特に MDB をサポートするには、JMS 仕様を実装するメッセージングミドルウェア、すなわち JMS プロバイダが必要です。Sun ONE Application Server のネイティブ JMS プロバイダは、Sun ONE Message Queue (MQ) バージョン 3.01 です。

MQ は Sun ONE Application Server に緊密に統合されています。このため、JMS メッセージングは透過的にサポートされます。このサポートにより、必要な管理作業を最小限に抑えることができます。Sun ONE Application Server では、これを「JMS サービス」と呼びます。

この章では、Sun ONE Message Queue に組み込まれた JMS サービスについて説明します。さらに、このサービスを管理するために必要な情報を提供します。この節では次の項目について説明します。

- JMS について
- 組み込み JMS サービス
- 組み込み JMS サービスの管理

JMS について

JMS 仕様には、分散エンタープライズメッセージングをサポートするプログラミングインタフェースが規定されています。エンタープライズメッセージングシステムにより、個々の分散型コンポーネントやアプリケーションは、メッセージを使って対話することができます。これらのコンポーネントは、同一システム上にある場合、同一ネットワーク上にある場合、インターネットを介して自由な状態で接続している場合がありますが、メッセージングを使ってデータの受け渡しを行い、相互の機能を調整する点で共通しています。

エンタープライズ規模のメッセージングをサポートするため、JMS は、信頼性の高い非同期メッセージ配信を行います。

高信頼性配信：ネットワークやシステムの障害が発生しても、一方のコンポーネントからもう一方のコンポーネントへのメッセージが失われることはありません。つまり、システムがメッセージを確実に配信できます。

非同期配信：多数のコンポーネントが同時にメッセージを交換し、高密度のスループットをサポートするために、メッセージの送信はコンシューマ側で受信の準備ができているかどうかに左右されません。コンシューマがビジー状態またはオフラインになっている場合も、システムはメッセージを送信します。このメッセージは、コンシューマの準備ができた時点で受信されます。この方式を非同期メッセージ配信、または蓄積型 (store-and-forward) メッセージングと呼びます。

ここでは、JMS の概念と用語について簡単に説明します。

- メッセージングシステムの基本概念
- JMS 仕様
- メッセージ駆動型 Beans

JMS の詳しい解説が必要な場合は JMS 1.0.2 仕様を参照してください。次の URL から参照できます。

```
http://java.sun.com/products/jms/docs.html
```

メッセージングシステムの基本概念

ここでは、エンタープライズメッセージングシステムの一般的な概念、および JMS に固有の概念について説明します。

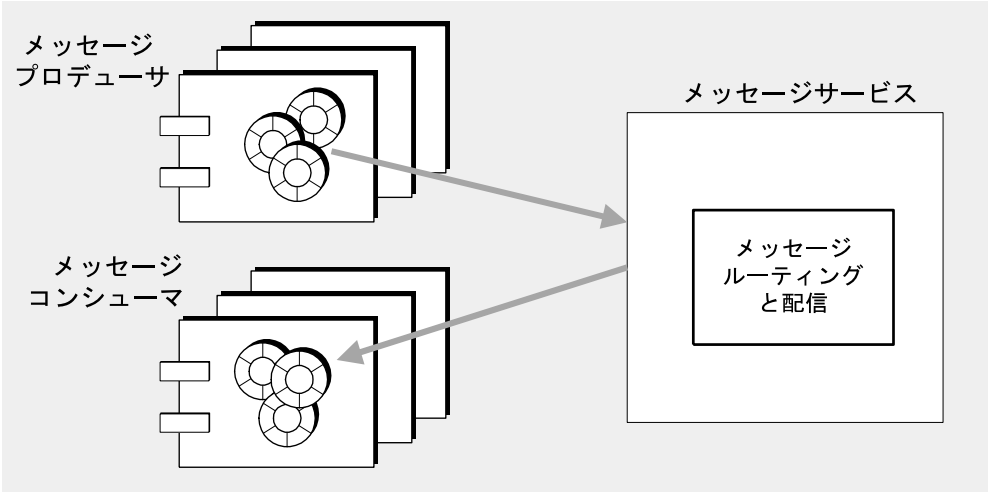
メッセージ

メッセージは、何らかの形式のデータ (メッセージ本文) と、メッセージの特性またはプロパティ (たとえば、そのメッセージの送信先、寿命など、メッセージングシステムによって規定される特性) を説明するメタデータ (メッセージヘッダー) で設定されます。

メッセージサービスアーキテクチャ

次の図「メッセージサービスアーキテクチャ」は、メッセージングシステムの基本アーキテクチャを示しています。メッセージングシステムは、共通のメッセージングサービスを利用してメッセージを交換するメッセージプロデューサとメッセージコンシューマで構成されます。一般に、単一のメッセージコンポーネント内に存在できるメッセージプロデューサとメッセージコンシューマの数に制限はありません。メッセージプロデューサは、メッセージサービスにメッセージを送信します。すると、メッセージサービスは、メッセージルーティングコンポーネントとメッセージ配信コンポーネントを使って、配信対象として登録されている 1 個以上のメッセージコンシューマにメッセージを配信します。メッセージルーティングコンポーネントとメッセージ配信コンポーネントは、適切な全コンシューマに確実にメッセージを配信することになっています。

メッセージサービスアーキテクチャ



メッセージ配信モデル

プロデューサとコンシューマの間には、1 対 1、1 対多、多対多の関係があります。たとえば、次のようなメッセージ配信が可能です。

- 単一のプロデューサから単一のコンシューマへ
- 単一のプロデューサから複数のコンシューマへ
- 複数のプロデューサから単一のコンシューマへ
- 複数のプロデューサから複数のコンシューマへ

これらの関係は、ポイントツーポイントとパブリッシュ / サブスクライブという 2 種類のメッセージ配信モデルで表されます。ポイントツーポイント配信モデルは、主に、特定のプロデューサから発信され特定のコンシューマによって受信されるメッセージを取り扱います。パブリッシュ / サブスクライブ配信モデルは、主に、任意の数のプロデューサから発信され任意の数のコンシューマによって受信されるメッセージを取り扱います。2 つのメッセージ配信モデルには共通点があります。

これまで、メッセージングシステムは、この 2 つの配信モデルの多種多様な組み合わせをサポートしてきました。JMS API は、ポイントツーポイントモデルとパブリッシュ / サブスクライブモデルの両方をサポートする共通のプログラミングアプローチとして開発されたものです。

JMS 仕様

JMS 仕様には、メッセージ構造、プログラミングモデル、およびメッセージング処理の方法や意味を規定する一連の規則が指定されています。

JMS メッセージ構造

JMS 仕様では、メッセージは、ヘッダー、プロパティ (ヘッダーの拡張と考えられる)、および本文の 3 つの部分で設定されています。

ヘッダー : ヘッダーは、メッセージの JMS 特性、すなわち、メッセージの送信先、持続性があるかどうか、寿命、優先度を指定します。メッセージングシステムによるメッセージ配信の方法は、これらの特性によって決まります。

プロパティ : アプリケーションは、プロパティによって提供される値を元に、さまざまな選択基準に従ってメッセージをフィルタリングできます。プロパティはオプションです。

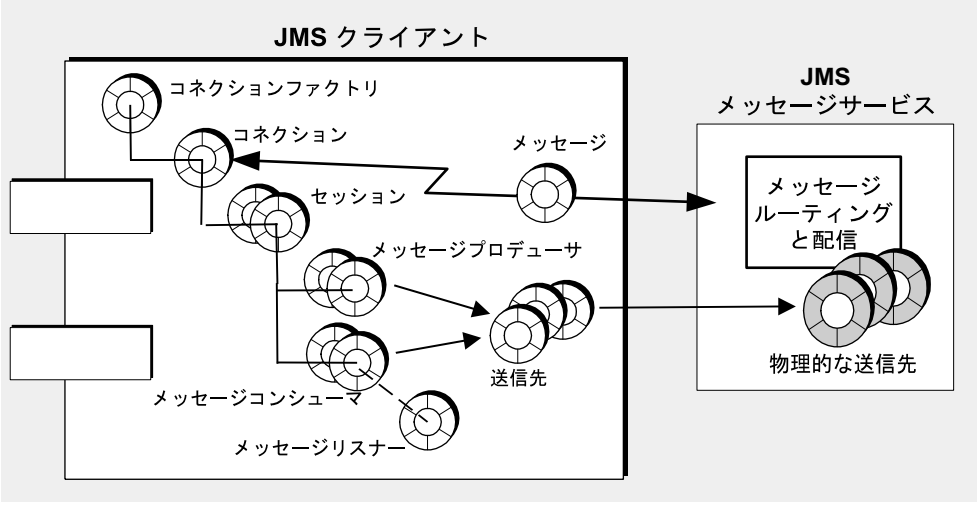
メッセージ本文 : メッセージ本文には、実際に交換されるデータが含まれます。JMS は、6 種類のメッセージ本文をサポートします。

JMS プログラミングモデル

JMS プログラミングモデルでは、JMS クライアント (コンポーネントまたはアプリケーション) は JMS メッセージサービスを利用してメッセージを交換します。メッセージプロデューサはメッセージサービスにメッセージを送信します。メッセージコンシューマは、このメッセージサービスからメッセージを受信します。このようなメッセージング処理は、JMS API を実装する一連のオブジェクト (JMS プロバイダによって提供される) によって行われます。図「JMS プログラミングオブジェクト」は、メッセージ配信のプログラミングに使用される JMS オブジェクトを示しています。

JMS プログラミングモデルでは、JMS クライアントは、接続ファクトリオブジェクトを使って接続を確立し、この接続を介して JMS メッセージングサービスとメッセージをやりとりします。接続は、JMS クライアントのメッセージサービスとのアクティブな接続です。通信リソースの割り当てとクライアントの認証は、接続の確立時に行われます。

JMS プログラミングオブジェクト



接続を使って、セッションを作成できます。セッションは、メッセージを生成および消費するためのシングルスレッドコンテキストです。これを使って、メッセージを送受信するメッセージプロデューサとメッセージコンシューマを作成できます。セッションは、複数の承認オプションまたは分散トランザクションマネージャで管理できるトランザクションにより、高信頼性配信をサポートします。

JMS クライアントは、メッセージプロデューサを使って、API では送信先オブジェクトとして表される指定された物理的な送信先へ、メッセージを送信します。メッセージプロデューサは、物理的な送信先に送信するすべてのメッセージに適用される、デフォルトの配信モード (持続または非持続メッセージ)、優先度、および生存期間を指定できます。

同様に、JMS クライアントは、メッセージコンシューマを使って、API では送信先オブジェクトとして表される指定された物理的な送信先から、メッセージを受信します。メッセージコンシューマは、同期または非同期のメッセージコンシュームをサポートします。非同期消費をサポートするには、コンシューマにメッセージリスナーを登録する必要があります。クライアントは、セッションスレッドがメッセージリスナーオブジェクトの `onMessage()` メソッドを呼び出した時点でメッセージを消費します。

管理対象オブジェクト：プロバイダ非依存

299 ページの「JMS プログラミングモデル」の図の 2 つのオブジェクトは、JMS プロバイダによる JMS メッセージサービスの実装方法に依存します。接続ファクトリオブジェクトは、プロバイダがメッセージの配信に使用する配下のプロトコルとメカニズムに依存します。送信先オブジェクトは特定の命名規則と、プロバイダによって使用される物理的な送信先の機能に依存します。

通常、こうしたプロバイダ固有の特性には、JMS クライアントコードを JMS API 実装の詳細から独立させる働きがあります。JMS 仕様によると、JMS クライアントコードをプロバイダ非依存にするには、プロバイダ固有のオブジェクト (管理対象オブジェクト) をクライアントコード内で直接インスタンス化するのではなく、標準化された方法でアクセスする必要があります。

管理対象オブジェクトは、プロバイダ固有の実装および設定情報をカプセル化します。これらは、管理者によって作成および設定され、ネームサービスに格納され、クライアントアプリケーションから JNDI ルックアップコードを介してアクセスされます。このような方法で管理対象オブジェクトを使用すれば、JMS クライアントコードをプロバイダから独立させることができます。

JMS は、接続ファクトリオブジェクトと送信先オブジェクトという 2 つの一般的な管理対象オブジェクトを提供します。どちらのオブジェクトもプロバイダ固有の情報をカプセル化しますが、JMS クライアント内での使用方法はまったく異なっています。接続ファクトリオブジェクトは、メッセージサーバーへの接続の確立に使用されます。一方、送信先オブジェクトは、JMS メッセージサービスによって使用される物理的な送信先の識別に使用されます。

注　Sun ONE Application Server のコンテキストでは、JMS 管理オブジェクトはその他の Application Server リソースと同様に JMS リソースとして扱われます。

メッセージ駆動型 Beans

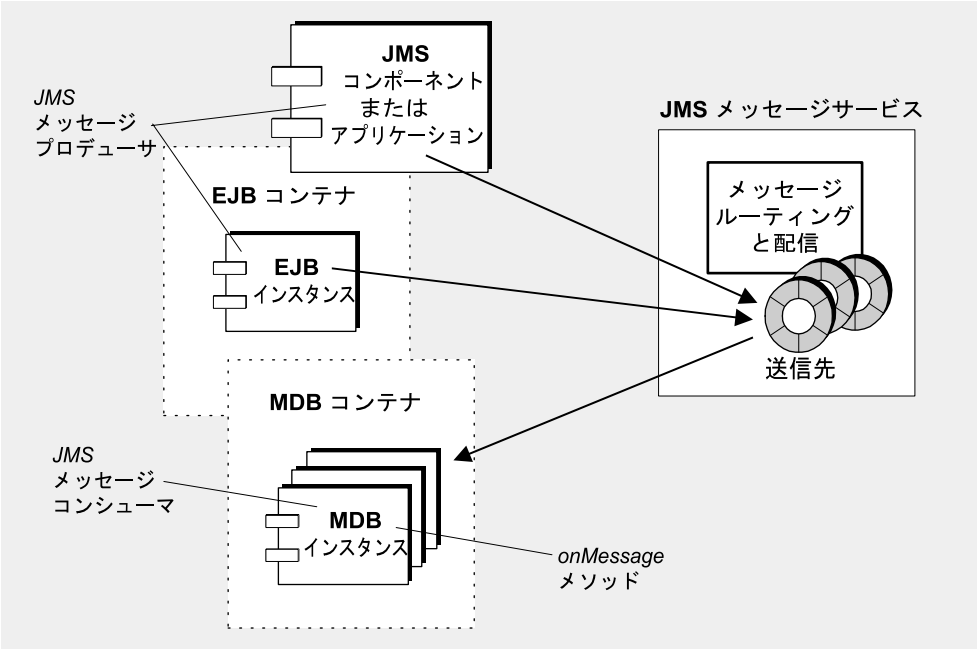
JMS クライアントプログラミングモデル (図 299 ページの「JMS プログラミングモデル」) のほかに、J2EE アプリケーションのコンテキスト内で使用される、JMS API の特殊な JMS クライアントがあります。この特殊な JMS クライアントをメッセージ駆動型 Beans と呼びます。メッセージ駆動型 Beans は、EJB 2.0 仕様 (`http://java.sun.com/products/ejb/docs.html`) で規定されている EJB コンポーネントのファミリです。

メッセージ駆動型 Beans が必要なのは、その他の EJB コンポーネント (セッション Beans およびエンティティ Beans) が同期呼び出ししかサポートしないからです。したがって、これらの Bean のメソッドを呼び出そうとしても、リソースはメソッドが完了するまでブロックされます。これらの EJB コンポーネントは、標準 EJB インタフェース経由でしかアクセスできないので、メッセージを非同期で受信する手段を持ちません。

しかし、多くのエンタープライズアプリケーションは、非同期メッセージングのニーズを抱えています。そこで、メッセージのプロデューサに密結合することなくメッセージを受信し、処理できる EJB コンポーネントが必要になります。

MDB は、特殊な EJB コンテナ (サポートするコンポーネントに対して分散サービスを提供するソフトウェア環境) によってサポートされる特殊な EJB コンポーネントです。

MDB メッセージコンシューマ



メッセージ駆動型 Bean: MDB は、JMS メッセージリスナーインタフェースを実装する JMS メッセージコンシューマです。その `onMessage` メソッド (MDB 開発者が作成する) は、MDB コンテナによってメッセージが受信された時点で呼び出されます。`onMessage` メソッドは、JMS MessageListerner オブジェクトの `onMessage` と同様に、メッセージを消費します。MDB は、単一の送信先からのメッセージを消費できます。メッセージのプロデューサは、スタンドアロンの JMS クライアントアプリケーション、Web コンポーネント、その他の EJB コンポーネントなどです。「MDB メッセージコンシューマ」の図を参照してください。

MDB コンテナ : 特別な EJB コンテナによってサポートされる MDB は、MDB インスタンスを作成し、メッセージを非同期で消費できるように設定します。これには、メッセージサービス (認証を含む) との接続設定、指定された送信先のセッションプールの作成、セッションプールおよび関連 MDB インスタンスによって受信されるメッセージの配信管理が関連します。コンテナは MDB インスタンスのライフサイクルを制御するので、着信メッセージの負荷に応じて MDB インスタンスプールを調整します。

MDB には、メッセージ消費の設定時にコンテナによって使用される管理対象オブジェクト (接続ファクトリオブジェクトと送信先オブジェクト) の JNDI ルックアップ名を指定する配備記述子が 1 つずつ割り当てられます。配備記述子には、配備ツールがコンテナの設定に使用するその他の情報も含まれています。これらのコンテナは、複数の MDB のインスタンスをサポートできません。

EJB コンテナの MDB を Sun ONE Application Server 用に設定する方法については、209 ページの「メッセージ駆動型 Beans について」を参照してください。

組み込み JMS サービス

Sun ONE Application Server には、JMS メッセージングのサポート、特に MDB のサポートが組み込まれています。このサポートは、Sun ONE Message Queue と Sun ONE Application Server の密接な統合によって実現されています。これにより、ネイティブの組み込み JMS サービスが提供されます。

この節では、この組み込み JMS サービスを理解する上で重要な項目を取り上げます。

- Sun ONE Message Queue (MQ) について
- MQ と Sun ONE Application Server の統合

組み込み JMS サービスの管理方法については、312 ページの「組み込み JMS サービスの管理」を参照してください。

Sun ONE Message Queue (MQ) について

Sun ONE Message Queue (MQ) は、JMS オープン標準を実装するエンタープライズメッセージングシステムです。MQ は、JMS プロバイダの 1 つでもあります。

MQ 製品は、信頼性の高い非同期メッセージングを行うために JMS 仕様に規定されている最小限の条件を上回る機能を備えています。これらの機能の一部 (集中管理、パフォーマンス調整、複数のメッセージングトランスポートのサポート、ユーザーの認証および承認) は、Sun ONE Application Server に統合された MQ Platform Edition でも使用できます。その他の機能 (スケーラブルなメッセージサーバーと安全なメッセージング) を利用したい場合は、MQ Enterprise Edition にアップグレードする必要があります。

305 ページの「MQ システムアーキテクチャ」の図に示すように、MQ メッセージングシステムは複数の部分で構成されています。個々の部分が、信頼性の高いメッセージ配信を行うために協調して動作します。

MQ メッセージングシステムの主要部分は次のとおりです。

- MQ メッセージサーバー
- MQ クライアントランタイム
- MQ 管理対象オブジェクト
- MQ 管理ツール

これらについては、次のトピックで簡単に説明します。MQ メッセージングシステムの詳細については、MQ の『管理者ガイド』を参照してください。次に URL を示します。

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

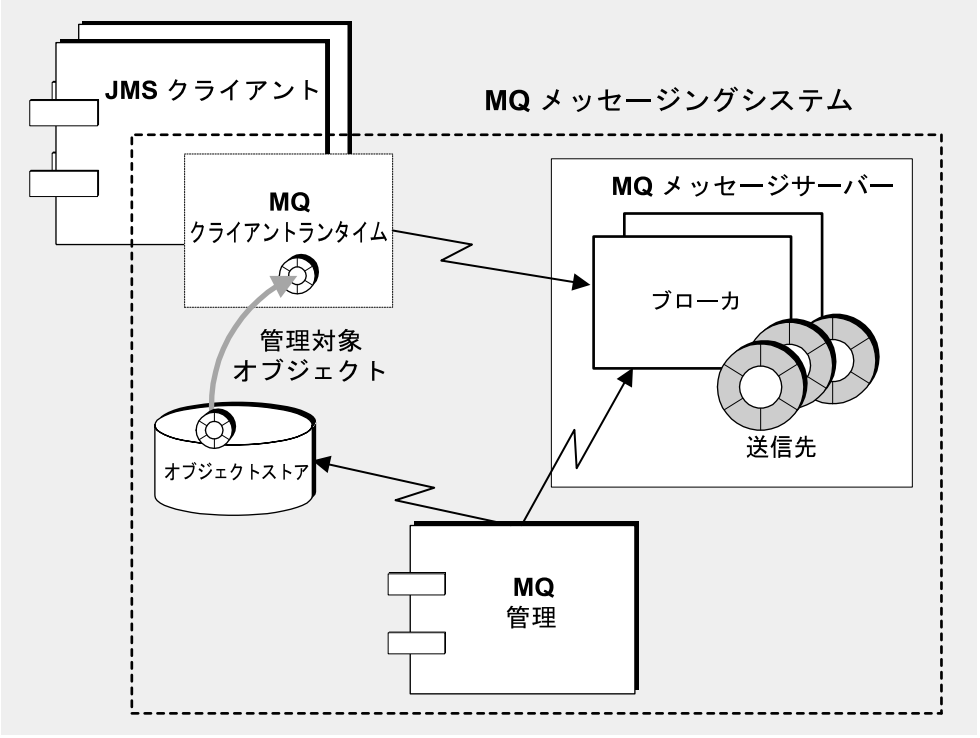
MQ メッセージサーバー

305 ページの「MQ システムアーキテクチャ」の図のように、MQ メッセージサーバーの主要部分は、ブローカと物理的な送信先です。

ブローカ : ブローカは、MQ メッセージングシステムの配信サービスを提供します。メッセージ配信は、接続サービス、メッセージルーティング、メッセージ配信、持続性、セキュリティ、およびログを処理する多数のコンポーネントに依存しています。メッセージサーバーのスケーラビリティは、1 個以上のブローカによって実現されます。

物理的な送信先 : メッセージ配信は、プロデューサクライアントからブローカによって管理されている物理的な送信先への配信と、この送信先から 1 つ以上のコンシューマクライアントへの配信の 2 段階で行われます。物理的な送信先は、ブローカの物理メモリーや固定記憶領域内の場所を表します。詳細については、306 ページの「物理的な送信先」を参照してください。

MQ システムアーキテクチャ



ブローカ

MQ メッセージングシステムにおけるメッセージ配信 (プロデューサクライアントから送信先への配信と、この送信先から 1 つ以上のコンシューマクライアントへの配信) は、ブローカ (MQ 3.01 Enterprise Edition では連携して機能するブローカクラスタ) によって行われます。ブローカがメッセージ配信を行うために必要なことは、通信チャネルとクライアントの設定、認証と承認、メッセージの適切な送信先への配信、配信の信頼性の確保、およびシステムパフォーマンスの監視用データの提供です。

この複雑な一連の機能を実行するために、ブローカは、複数のコンポーネントを使用します。これらのコンポーネントは、それぞれが、配信プロセスにおいて特別な役割を果たします。これらの内部コンポーネントを設定することにより、負荷の条件やアプリケーションの複雑さに応じて、ブローカのパフォーマンスを最適化できます。詳細については、MQ の『管理者ガイド』を参照してください。

物理的な送信先

MQ メッセージングは、2 段階のメッセージ配信に基づいています。最初のメッセージ配信は、プロデューサクライアントからブローカ上の送信先への配信です。2 番目のメッセージ配信は、ブローカ上の送信先から 1 つ以上のコンシューマクライアントへの配信です。送信先は、キュー (ポイントツーポイント配信モデル) かトピック (パブリッシュ / サブスクライブ配信モデル) のどちらかになります。これらの送信先は、物理メモリー上にあります。着信メッセージは、ここで整列化されたあと、コンシューマクライアントへ配信されます。

しかし、このような送信先は、メッセージの受信時にブローカによって自動的に作成されることもあります。

送信先キュー : 送信先キューは、ポイントツーポイントメッセージングで使用されます。この場合、メッセージは、最終的に、送信先に配信対象として登録されている複数のコンシューマのうち 1 つだけに配信されます。プロデューサクライアントから着信したメッセージは、待ち行列に入ったあと、コンシューマクライアントへ配信されます。

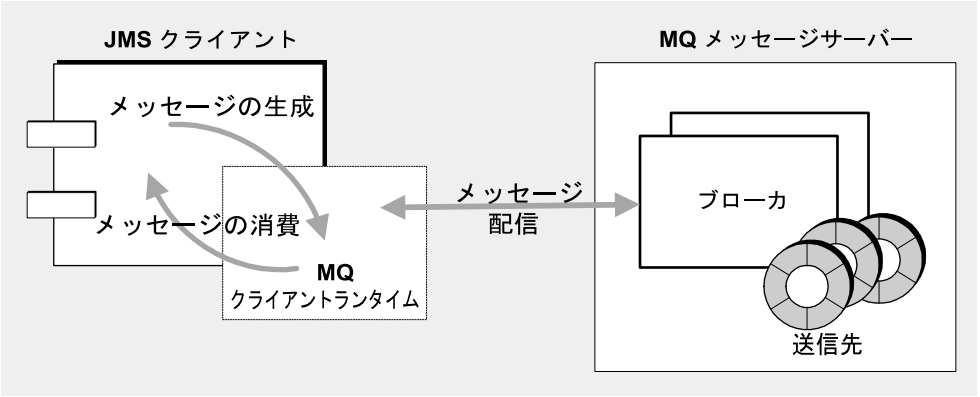
送信先トピック : 送信先トピックは、パブリッシュ / サブスクライブメッセージングで使用されます。この場合、メッセージは、最終的に、送信先に配信対象として登録されているすべてのコンシューマに配信されます。プロデューサから着信したメッセージは、トピックに登録されているすべてのコンシューマへ配信されます。トピックに永続的に登録されているコンシューマは、トピックにメッセージが配信された時点でアクティブになっていなくてもかまいません。メッセージはブローカが格納し、コンシューマがアクティブになった時点で配信されます。

MQ クライアントランタイム

MQ クライアントランタイムは、JMS クライアント (スタンドアロンアプリケーション、Web コンポーネント、EJB コンポーネントなど) に対して、MQ メッセージサーバーへのインタフェースを提供します。つまり、送信先にメッセージを送信したり、これらの送信先からメッセージを受信したりするクライアントに必要な、すべてのプログラミングインタフェースの実装を提供します。

307 ページの「メッセージング処理」の図は、メッセージの生成時および消費時の JMS クライアントと MQ クライアントランタイム間の対話、およびメッセージの配信時の MQ クライアントランタイムと MQ メッセージサーバーの対話を表しています。

メッセージング処理



MQ 管理対象オブジェクト

MQ 管理対象オブジェクトは、プロバイダ固有の実装および設定情報をオブジェクト内にカプセル化することにより、JMS クライアントコードをプロバイダから独立させます (300 ページの「管理対象オブジェクト : プロバイダ非依存」を参照)。クライアントアプリケーションは、カプセル化された情報を含むオブジェクトを、プロバイダに依存しない方法で使用します。MQ 管理対象オブジェクトは、管理者によって作成および設定され、ネームサービスに格納され、JMS クライアントから JNDI ルックアップコードを介してアクセスされます。

管理対象の接続ファクトリオブジェクト : 接続ファクトリオブジェクトは、JMS クライアント (スタンドアロンアプリケーション、Web コンポーネント、EJB コンポーネントなど) と MQ メッセージサーバー間の物理接続を作成するときに使用されます。接続ファクトリオブジェクトは、ブローカ内で物理的に表されることはありません。接続ファクトリオブジェクトは、JMS クライアントとブローカを接続するために使用されるだけです。また、接続の動作や、ブローカにアクセスするために接続を使用するクライアントランタイムの動作を指定するためにも使用されます。そのため、MQ 接続ファクトリは、MQ システムのパフォーマンス調整に使用する多数の設定可能な属性を備えています。

管理対象の送信先オブジェクト : 管理対象の送信先オブジェクト (キューまたはトピック) は、ブローカ内の物理的な送信先 (物理的なキューまたはトピック) を表します。公開指定された管理対象オブジェクトは、この送信先に対応付けられます。管理対象の送信先オブジェクトを作成することにより、JMS クライアント (メッセージコンシューマやメッセージプロデューサ) は対応する物理的な送信先にアクセスできるようになります。

MQ 管理ツール

MQ 管理ツールには、コマンド行ユーティリティと、グラフィカルユーザーインタフェース (GUI) である管理コンソールの 2 種類があります。

管理コンソール : 管理コンソールを使って、ブローカに接続して管理したり、ブローカ上に物理的な送信先を作成したりできます。また、オブジェクトストアに接続して、その管理対象オブジェクトを追加、更新、または削除することができます。管理コンソールでは実行できないタスクもあります。該当する主なタスクは、ブローカの起動、ブローカクラスタの作成、専門性の高い一部のブローカプロパティの設定、ユーザーデータベースの管理です。

コマンド行ユーティリティ : MQ ユーティリティでは、管理コンソールで実行できるすべてのタスクを実行できます。さらに、ブローカの起動および管理、専門性の高い一部のブローカのプロパティの設定、MQ ユーザーデータベースの管理も可能です。

MQ と Sun ONE Application Server の統合

MQ Platform Edition は、Sun ONE Application Server のインストール時に自動的にインストールされます。詳細は、『Sun ONE Application Server インストールガイド』を参照してください。

MQ をインストールすると、Sun ONE Application Server に、任意の数の Sun ONE Application Server インスタンスをサポートする JMS メッセージングシステムが提供されます。各サーバーインスタンスには、デフォルトで、そのインスタンスで実行中のすべての JMS クライアントをサポートする組み込み JMS サービスが割り当てられています。

ここでは次の項目について説明します。

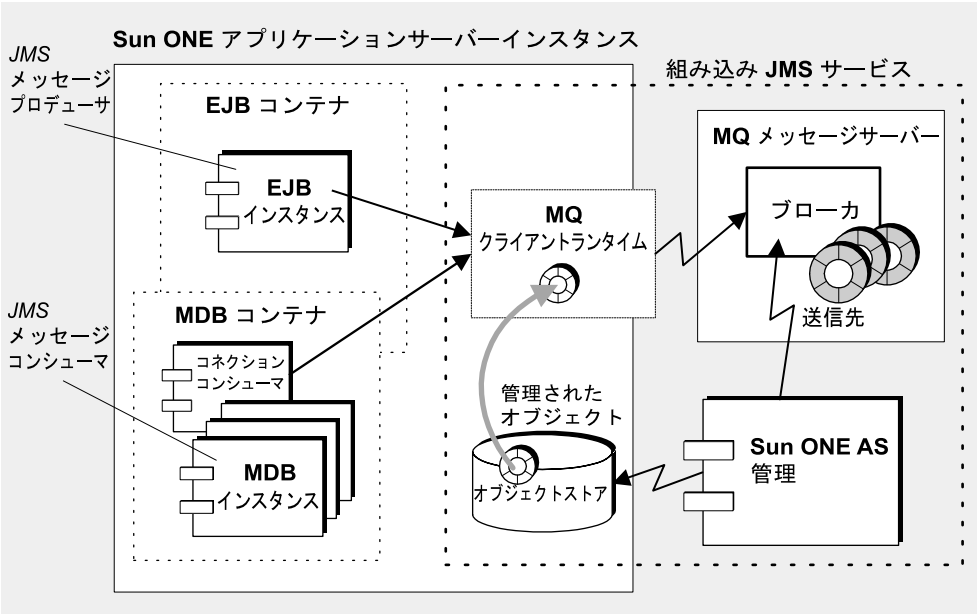
- 組み込み JMS サービスのアーキテクチャ
- 組み込み JMS サービスの無効化

組み込み JMS サービスは、Sun ONE Application Server 管理ツールで管理できます (312 ページの「組み込み JMS サービスの管理」を参照)。

組み込み JMS サービスのアーキテクチャ

組み込み JMS サービス (309 ページの「組み込み MQ メッセージングシステム」の図を参照) は、次の点を除いて、通常の MQ メッセージングシステム (305 ページの「MQ システムアーキテクチャ」の図を参照) と同じです。

組み込み MQ メッセージングシステム



MQ メッセージサーバー : Sun ONE Application Server インスタンスは、それぞれ独自の組み込み JMS サービスに関連付けられています。組み込み JMS サービスは、1 つのブローカメッセージサーバーを利用します。上の図の「組み込み MQ メッセージングシステム」のように、ブローカは Sun ONE Application Server インスタンスの外部の独立したプロセス内で実行されます。デフォルトでは、ブローカインスタンス (組み込み JMS サービス) は、関連サーバーインスタンスの起動時に起動し、関連サーバーインスタンスの停止時に停止します。サーバーインスタンスの組み込み JMS サービスの設定情報は、Sun ONE Application Server 設定ストア (`server.xml` ファイル) に記録されます。この情報の変更方法については、313 ページの「JMS サービスの設定」を参照してください。

MQ クライアントランタイム : JMS サービスのクライアントランタイムは、JMS API をサポートするライブラリのセットです。サーバーインスタンス内で実行されるすべての JMS クライアント (MDB を含む JMS クライアントコンポーネント) が、このライブラリセットにアクセスできます。

MQ 管理対象オブジェクト : 組み込み JMS サービスは、Sun ONE Application Server が提供するオブジェクトストアを使用します。それぞれのサーバーインスタンスが独自のオブジェクトストアを持ちます。JMS サービスは、このオブジェクトストア内に管理対象オブジェクト (接続ファクトリオブジェクトおよび送信先オブジェクト) を

格納します。これらの管理対象オブジェクトリソースの作成方法については、318 ページの「管理対象オブジェクトリソースの管理」を参照してください。JMS クライアントは、JNDI ルックアップコードを使ってこれらのオブジェクトにアクセスします。

Sun ONE Application Server の管理 : Sun ONE Application Server の管理インタフェースとコマンド行ユーティリティは、MQ 管理機能の限定されたサブセットを実装します。管理インタフェースとコマンド行を使って、組み込み JMS サービスを設定できます。物理的な送信先の作成および削除、JMS クライアントが JMS メッセージング処理に使用する管理対象オブジェクトリソースの作成および削除も可能です。しかし、ブローカプロパティの設定、MQ クライアントラインタイムの調整、MQ ユーザーリポジトリの変更、MQ セキュリティ管理といった高度な管理タスクは、これらの管理ツールでは実行できません (または処理が複雑)。組み込み JMS サービスの高度な管理タスクを実行したい場合は、MQ のインストール時にインストールされた管理ツールを使用し、MQ の『管理者ガイド』の説明に従ってください。MQ と Sun ONE Application Server の管理機能については、312 ページの「Sun ONE Message Queue と Sun ONE Application Server の管理機能の比較」の対照表を参照してください。

組み込み JMS サービスの無効化

デフォルトでは、組み込み JMS サービス (MQ ブローカ) は、Sun ONE Application Server インスタンスの起動時に起動します。しかし、サーバーインスタンスで JMS メッセージングをサポートする必要がない場合や、サーバーインスタンスが外部 JMS サービスを使用する場合など、サーバーインスタンスの起動時に JMS サービスを自動的に起動したくない場合もあります。この場合は、313 ページの「JMS サービスの設定」の説明に従って、組み込み JMS サービスを無効にします。

外部 JMS サービスは、Sun ONE Application Server 内で制御されないメッセージングシステムです。MQ、すなわちネイティブ JMS プロバイダの場合は、MQ 管理ツールを使って MQ メッセージサーバーを個別に起動し、管理します。サーバーインスタンス上で稼働している JMS クライアントも、MQ 管理対象オブジェクトを使って MQ メッセージサーバーにアクセスできます。これらの管理対象オブジェクトは、各アプリケーションサーバーインスタンスに関連づけられたオブジェクトストアに格納されるか、MQ 管理ツールによって管理される独立したオブジェクトストア (必要に応じて複数のサーバーインスタンスが共有します) に格納されます。

サーバーインスタンスは、さまざまな方法で外部 JMS サービスを使用します。もっとも典型的な例は、別々のサーバーインスタンス内の JMS クライアントから同一の物理的な送信先にアクセスする必要がある場合です。この場合、すべてのサーバーインスタンスが同一のメッセージサーバーにアクセスする必要があります。このためには、すべてのサーバーインスタンスの組み込み JMS サービスを無効にします。さらに、す

べての JMS クライアントが適切な JNDI ルックアップを実行して外部 JMS サービスにアクセスするように設定します。なお、外部 JMS サービス (メッセージサーバーの管理、物理的な送信先の作成、必要なすべての管理対象オブジェクトの作成) は、外部 JMS サービスプロバイダの管理ツールを使って個別に管理されます。

複数のアプリケーションサーバーインスタンスが 1 つの MQ ブローカインスタンスを共有するように設定する方法は、次のとおりです。

1. すべてのサーバーインスタンス上の JMS サービスを無効化します。
2. すべてのサーバーインスタンスで共有 MQ ブローカを個別に管理します。つまり、外部サービスを管理する管理ツールを使ってブローカを起動、停止します。また、物理的な送信先を Sun ONE Application Server から独立して管理する必要があります。
3. 各サーバーインスタンスの接続ファクトリ JMS リソースが外部 MQ ブローカを参照するように設定します (`imqBrokerHostName` プロパティと `imqBrokerHostPort` プロパティを適切に設定する)。
4. JMS アプリケーションを Sun ONE Application Server に配備するときは、この接続ファクトリリソースを使います。

外部 JMS サービスと組み込み JMS サービスを同時に実行できます。サーバーインスタンス内の JMS クライアントは、必要な JMS サービスにアクセスできます。

複数のサーバーインスタンスに同一の組み込み JMS サービスを共有させる、すなわち、JMS サービスを 1 つだけ有効にして残りは無効にするという方法はお勧めしません。有効になっているこの JMS サービスは、関連サーバーインスタンスが実行されているときしか実行されないので、状況に対処するのが非常に困難になります。

組み込み JMS サービスを無効にしたら、その JMS サービスに関連付けられている管理タスクも、実行できないように無効にします。また、外部 JMS サービスをサポートするために必要なすべての管理タスクは、その外部サービスの管理用ツールで実行する必要があります。

組み込み JMS サービスの管理

この節では、組み込み JMS サービスの管理について取り上げます。組み込み JMS サービスの管理は、サーバーインスタンス単位で行います。

組み込み JMS サービスの管理タスクは次のとおりです。

- JMS サービスの設定
- 物理的な送信先の管理
- 管理対象オブジェクトリソースの管理
- コマンド行インタフェースによる組み込み JMS サービスの管理

管理には、Sun ONE Application Server の管理インタフェースまたはコマンド行ユーティリティを使用します。これらの管理ツールと MQ 管理ツールについては、「Sun ONE Message Queue と Sun ONE Application Server の管理機能の比較」の表を参照してください。

Sun ONE Message Queue と Sun ONE Application Server の管理機能の比較

機能	Sun ONE MQ 管理ツール	Sun ONE AS 管理インタフェース	Sun ONE AS 管理コマンド行
MQ ブローカの状態管理	可	起動 / 停止	起動 / 停止
MQ ブローカの設定	可	不可	不可
MQ ブローカユーザーリポジトリの管理	可	不可	不可
マルチブローカクラスタ	可	不可	不可
セキュリティ管理	可	不可	不可
物理的な送信先の管理	可	作成 / 削除	作成 / 削除
永続的な登録およびトランザクションの管理	可	不可	不可
管理対象オブジェクトリソースの管理	可	作成 / 削除 / 設定	可

次に、Sun ONE Application Server の管理インタフェースを使用して、JMS サービスの管理タスクを実行する方法について説明します。

JMS サービスの設定

組み込み JMS サービスには、インストール時に多数の JMS サービスプロパティが設定されます。JMS サービスを設定すると、これらのプロパティのデフォルト値を変更できます。

JMS サービスプロパティについては、「JMS サービスプロパティ」の表を参照してください。

JMS サービスプロパティ

プロパティ	説明	デフォルト値
ログレベル	Sun ONE Application Server ログファイルに書き込むログ情報のレベル。詳細については、第 5 章「ログの使用」を参照してください。	DEBUG_HIGH
ポート	組み込み JMS サービスを提供するブローカインスタンスのプライマリポート番号。デフォルトでは、組み込み JMS サービスはデフォルトのプライマリポート番号を使用する。ただし、このポートがほかのソフトウェアと競合する場合や、複数の Sun ONE Application Server インスタンスを起動する場合は、それぞれに固有のプライマリポート番号を指定することが必要 JMS サービスのインストール時に割り当てられたポート番号を後から別のサービスが使用すると、ポートが競合する可能性がある。この場合は、JMS サービスに別のポート番号を割り当てる必要がある	7676
管理者のユーザー名 / パスワード	物理的な送信先の管理など、ブローカ管理タスクの実行に必要なユーザー名とパスワード (315 ページの「物理的な送信先の管理」を参照)。セキュリティ上の理由からブローカインスタンスへの管理者としてのアクセスを制限したい場合 (デフォルトではすべてのユーザーがアクセス可能) は、MQ の『管理者ガイド』の説明に従って、ブローカのユーザーリポジトリ内に適切なエントリを最初に作成する。さらに、このプロパティに管理者のユーザー名とパスワードに対応する値を指定する	admin/admin
起動タイムアウト	サーバーインスタンスが JMS サービスの起動を待機する時間を秒単位で指定する。このタイムアウトが経過すると、サーバーインスタンスの起動は中断される	60

JMS サービスプロパティ (続き)

プロパティ	説明	デフォルト値
起動引数	JMS サービスの起動時に使用される引数を指定する。imqbroker コマンドの起動引数とその指定方法については、MQ の『管理者ガイド』を参照。なお、-name 引数と -port 引数は指定しても無視される	
起動的に有効	サーバーインスタンスの起動時に、組み込み JMS サービスを起動するかどうかを指定する。JMS メッセージングをサポートしない場合や、外部 JMS メッセージサービスを使用する場合は、このプロパティの値を FALSE にする	TRUE

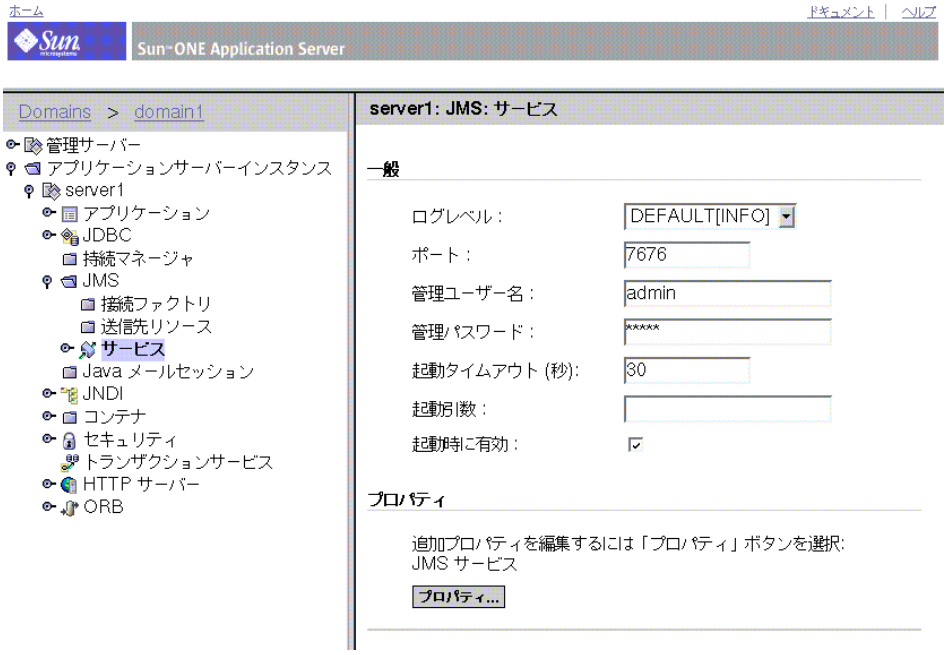
組み込み JMS サービスは、対応するサーバーインスタンスを起動する前に設定できます。サーバーインスタンスの実行中に、組み込み JMS サービスの設定に変更を加えた場合、変更内容を有効にするには、サーバーインスタンスを停止して再起動する必要があります。

組み込み JMS サービスを設定するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「サービス」リンクを選択します。

   右側のペインに JMS サービスの設定画面が表示されます。

JMS サービスの設定画面



5. 313 ページの「JMS サービスプロパティ」の表を参照して、目的のプロパティの値を変更します。
6. 「保存」ボタンをクリックします。

   JMS サービス画面が更新されます。

物理的な送信先の管理

JMS メッセージングでは、JMS プロデューサはメッセージサービス上の物理的な送信先にメッセージを送信します。その後、この送信先から JMS コンシューマにメッセージが振り分けられます。

組み込み型の JMS サービスでは、こうした物理的な送信先は明示的に作成できますが、メッセージの受信時に JMS サービス (MQ ブローカ) によって自動的に作成することもできます。通常は、メッセージングアプリケーションに必要な物理的な送信先を明示的に作成したほうが、メッセージングシステムとそのリソースを制御しやすくなります。これらの送信先は、不要になったら削除できます。

組み込み JMS サービスの物理的な送信先を作成または削除するためには、JMS サービスが実行されていて、(組み込み JMS サービスを設定する時に指定した) 管理者のユーザー名とパスワードがブローカのユーザーリポジトリ内の有効なエントリと一致している必要があります。313 ページの「JMS サービスプロパティ」の表を参照してください。

管理インタフェースでは、組み込み JMS サービス上の物理的な送信先に対して、次の管理タスクを実行できます。

- 送信先キューまたは送信先トピックの作成
- 物理的な送信先の管理
- 物理的な送信先の削除

送信先キューまたは送信先トピックの作成

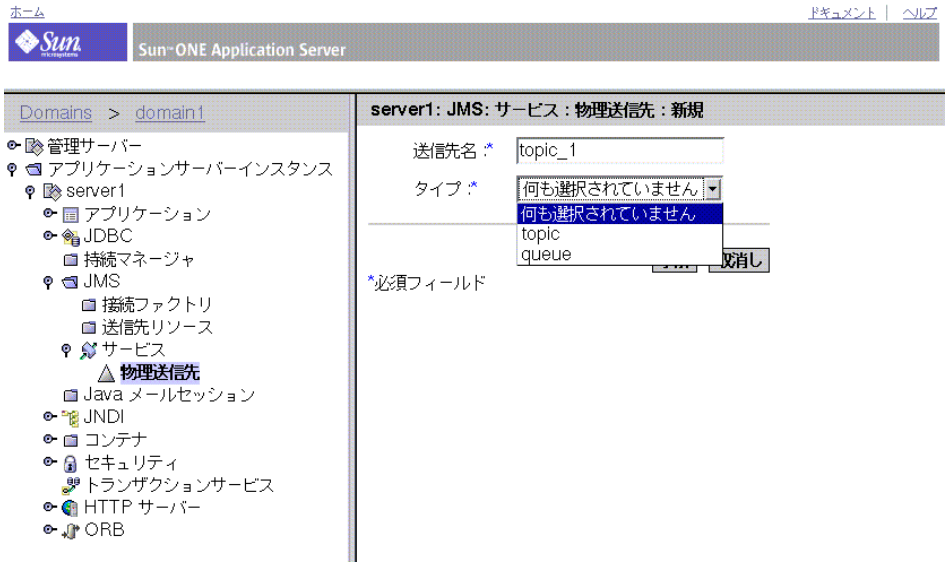
送信先キューまたは送信先トピックを作成するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「サービス」リンクを選択します。
5. 「物理送信先」リンクを選択します。

   右側のペインに「物理送信先」画面が表示されます。
6. 「新規」ボタンをクリックします。

   右側のペインに「物理送信先 : 新規」画面が表示されます。

「物理送信先 : 新規」画面



7. 物理的な送信先の名前を入力します。
8. 「タイプ」プルダウンから queue または topic を選択します。
9. 「了解」ボタンをクリックします。

   右側のペインが更新され、既存の送信先キューと送信先トピックのリストに新しいキューまたはトピックが表示されます。

物理的な送信先の管理

既存の送信先キューや送信先トピックを一覧表示するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「サービス」リンクを選択します。
5. 「物理送信先」リンクを選択します。

   右側のペインに現在の物理送信先が表示されます。

物理的な送信先の削除

必要に応じて、送信先キューまたは送信先トピックを削除できます。

物理的な送信先を削除するには、次の手順に従います。

1. 317 ページの「物理的な送信先の管理」の手順に従って、既存の送信先を一覧表示します。
2. 削除したい送信先の選択ボックスをクリックします。

JMS の「物理送信先 (Physical Destinations)」画面

3. 「削除」ボタンをクリックすると、選択された送信先が削除されます。

   リストが更新され、残りの送信先が表示されます。

管理対象オブジェクトリソースの管理

Sun ONE Application Server は、MQ 管理対象オブジェクトを JMS リソースと見なします。JMS クライアントは、これらのオブジェクトを使って JMS サービス (組み込み型サービスまたは外部サービス) にアクセスします。

J2EE コンポーネントは、管理対象の接続ファクトリオブジェクトリソースと送信先オブジェクトリソースを使って JMS サービスへの接続を確立し、サービス上の物理的な送信先とメッセージをやりとりします (307 ページの「MQ 管理対象オブジェクト」を参照)。

管理対象オブジェクトリソースの作成には、JMS サービスは直接関わらないため、JMS サービスを有効にする必要はありません。また、サーバーインスタンスの管理対象オブジェクトリソースを作成するために、有効なユーザー名とパスワード (313 ページの「JMS サービスプロパティ」の表を参照) を入力する必要もありません。

管理対象オブジェクトの属性

JMS メッセージングをサポートするには、サーバーインスタンスで実行中のすべての JMS クライアントに必要な管理対象オブジェクトリソースを作成します。少なくとも、各管理対象オブジェクトリソースの JNDI ルックアップ名、型 (接続ファクトリ、キュー、またはトピック)、説明 (オプション)、リソースが有効であるかどうかを指定する必要があります。その他の属性については、次の項で説明します。

送信先 (キューまたはトピック)

管理対象のキューオブジェクトまたはトピックオブジェクトの場合は、対応する物理的な送信先の名前も指定する必要があります。

接続ファクトリ

接続ファクトリの管理対象オブジェクトの場合、管理インタフェースは、組み込み JMS サービスを使用する接続ファクトリを作成します。この組み込み JMS サービスは、ホスト名がローカルホストで、JMS サービスの設定時にポート番号が設定されるブローカインスタンスです (313 ページの「JMS サービスプロパティ」の表を参照)。

ただし、特定のサーバーインスタンスで JMS サービスが無効になっている場合、このサーバーインスタンスでサポートされるすべての JMS クライアントは、外部 JMS サービスを使用しなければなりません。この外部 JMS サービスへの接続の確立に使われる接続ファクトリを作成するときは、適切なブローカインスタンスのホスト名とポート番号を示す属性を設定する必要があります。

接続ファクトリの管理対象オブジェクトには、サーバーインスタンスの MQ クライアントランタイムの調整に使われる追加属性があります。これについては、MQ の『開発者ガイド』を参照してください。

管理インタフェースによって作成された管理対象の接続ファクトリオブジェクトは、分散トランザクションマネージャをサポートします。

管理対象オブジェクトリソースの管理タスク

管理インタフェースでは、管理対象オブジェクトリソースに対して次の管理タスクを実行できます。

- キューオブジェクトまたはトピックオブジェクトの作成 (送信先リソース)
- 管理対象の ConnectionFactory オブジェクトの作成
- 管理対象オブジェクトリソースの一覧表示
- 管理対象オブジェクトリソースの削除

キューオブジェクトまたはトピックオブジェクトの作成 (送信先リソース)

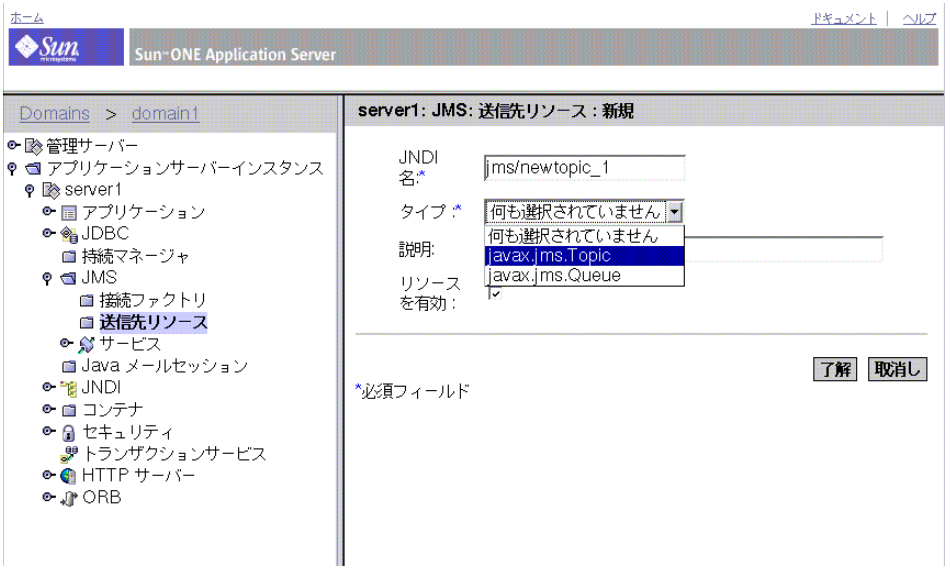
管理対象のキューオブジェクトまたはトピックオブジェクトを作成するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「送信先リソース」リンクを選択します。

   右側のペインに「送信先リソース」画面が表示されます。
5. 「新規」ボタンをクリックします。

   「送信先リソース : 新規」画面が表示されます。

「送信先リソース : 新規」画面



6. 管理対象の送信先オブジェクトの JNDI ルックアップ名を入力します。
7. プルダウンリストからオブジェクトタイプとして「Queue」または「Topic」を選択します。
8. 「了解」ボタンをクリックします。

   右側のペインに「送信先リソース : 新規」画面が再表示されます。

また、オブジェクトの送信先名を `imqDestinationName` プロパティに指定する必要があります。このプロパティの値は、物理的な送信先名と一致している必要があります。

このプロパティの値を変更するには、次の手順に従います。

1. 管理インタフェースを開きます。

2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「送信先リソース」フォルダを開きます。
5. 編集する送信先リソースを選択します。

   右側のペインに「送信先リソース」画面が表示されます。
6. 右側のペインの「プロパティ」をクリックします。

   「プロパティを編集」画面が表示されます。
7. 「名前」フィールドに imqDestinationName と入力します。
8. 「値」フィールドに物理的な送信先名を入力します。
9. 「了解」をクリックします。

   右側のペインに「送信先リソース」画面が再表示されます。

管理対象の ConnectionFactory オブジェクトの作成

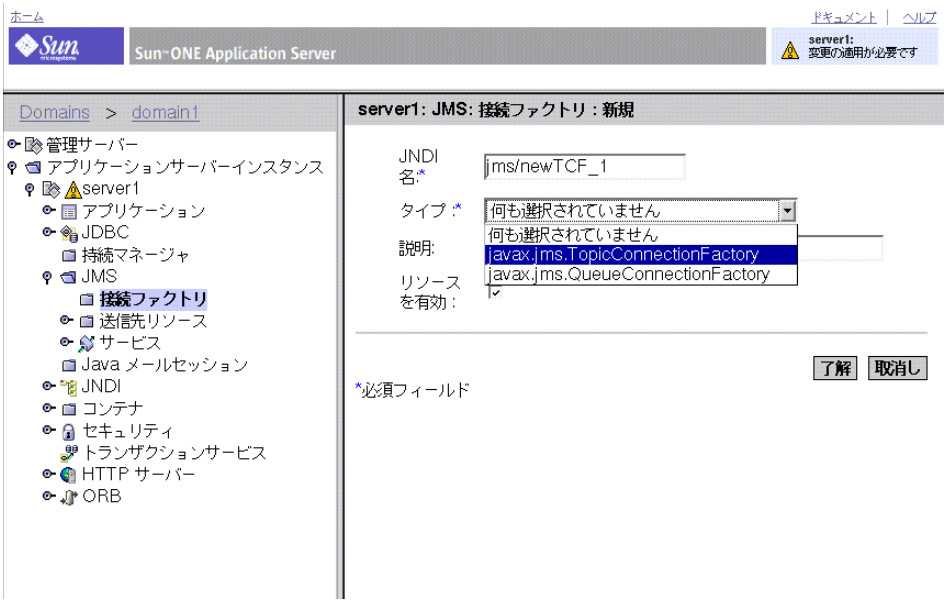
管理対象のキュー接続ファクトリオブジェクトまたはトピック接続ファクトリオブジェクトを作成するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「接続ファクトリ」リンクを選択します。

   右側のペインに「接続ファクトリ」画面が表示されます。
5. 「新規」ボタンをクリックします。

   「接続ファクトリ : 新規」画面が表示されます。

「接続ファクトリ：新規」画面



6. 管理対象の接続ファクトリオブジェクトの JNDI ルックアップ名を入力します。
7. プルダウンリストから接続ファクトリオブジェクトタイプを選択します。
8. 「了解」ボタンをクリックします。

   右側のペインの「接続ファクトリ」画面のリストに、新しく作成された接続ファクトリオブジェクトが再表示されます。

組み込み JMS サービス以外のブローカへの接続を確立する接続ファクトリの場合は、`imqBrokerHostName` プロパティと `imqBrokerHostPort` プロパティに適切な値を設定する必要があります。

これらのプロパティの値を変更するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「接続ファクトリ」フォルダを開きます。
5. 編集する接続ファクトリリソースを選択します。

   右側のペインに「接続ファクトリ」画面が表示されます。
6. 右側のペインの「プロパティ」をクリックします。

   「プロパティを編集」画面が表示されます。

7. 「名前」フィールドに imqBrokerHostName と入力します。
8. 「値」フィールドにプロパティの値を入力します。
9. 「名前」フィールドに imqBrokerHostPort と入力します。
10. 「値」フィールドにプロパティの値を入力します。
11. 「了解」をクリックします。

    右側のペインに「接続ファクトリ」画面が再表示されます。

管理対象オブジェクトリソースの一覧表示

既存の管理対象オブジェクトを一覧表示するには、次の手順に従います。

1. 管理インタフェースを開きます。
2. 左側のペインのサーバーインスタンスを開きます。
3. JMS フォルダを開きます。
4. 「送信先リソース」リンクまたは「接続ファクトリ」リンクを選択します。

   現在の管理対象送信先オブジェクトまたは管理対象接続ファクトリオブジェクトが表示されます。

管理対象オブジェクトリソースの削除

管理対象オブジェクトリソースを削除するには、次の手順に従います。

1. 323 ページの「管理対象オブジェクトリソースの一覧表示」の手順に従って、既存の管理対象オブジェクトリソースを一覧表示します。

   右側のペインに現在の管理対象オブジェクトリソースが表示されます。
2. 削除したいオブジェクトの「選択」ボックスをクリックします。

「JMS 接続ファクトリ」画面が更新されます。

3. 「削除」ボタンをクリックすると、選択されたオブジェクトが削除されます。

   リストが更新され、残りの管理対象オブジェクトリソースが表示されます。

コマンド行インタフェースによる組み込み JMS サービスの管理

Sun ONE Application Server には、コマンド行ユーティリティ `asadmin` が用意されています。このユーティリティを使って、管理インタフェースと同じタスクを実行できます。

組み込み JMS サービスを設定、管理するには、次の `asadmin` コマンドを使います。

組み込み JMS サービスの管理に使う `asadmin` コマンド

コマンド	用途
`add-resources`	型が jdbc、jms、または javamail の 1 つまたは複数のリソースを追加する
`create-jmsdest`	JMS 物理送信先を作成する
`create-jms-resource`	JMS リソースを作成する

組み込み JMS サービスの管理に使う asadmin コマンド (続き)

コマンド	用途
delete-jmsdest	JMS 物理送信先を削除する
delete-jms-resource	JMS リソースを削除します。
jms-ping	JMS プロバイダが稼働しているかどうかを ping で確認する
list-jmsdest	サーバーインスタンスの JMS 物理送信先を一覧表示する
list-jms-resources	サーバーインスタンスの JMS リソースを一覧表示する
get/set jms-service	JMS サービスの属性を取得 / 設定する
get/set jms-resource	JMS リソースの属性を取得 / 設定する

これらのコマンドの構文については、asadmin のオンラインヘルプを参照してください。asadmin の詳細と、jms-service および jms-resource の属性リストについては、付録 A「コマンド行インタフェースの使用」を参照してください。

第 12 章

Corba/IIOP クライアント用のサーバーの設定

この章では、Sun ONE Application Server 環境で、RMI/IIOP プロトコルを使って CORBA/IIOP クライアントのサポートを設定する方法を説明します。

この章では次のトピックについて説明します。

- CORBA/IIOP クライアントのサポートについて
- ORB の設定

CORBA/IIOP クライアントのサポートについて

J2EE プラットフォームは、相互運用性の要件により、多種多様なクライアント、ハードウェアプラットフォーム、およびソフトウェアアプリケーションを間接的にサポートします。J2EE に準拠した Sun ONE Application Server は、相互運用性の保証された標準のプロトコルおよび形式をサポートします。

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、実行する必要が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、存在する場合は実際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さまざまな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。

この節では次の項目について説明します。

- 相互運用性について
- ORB について
- RMI/IIOP の機能について

- 認証プロセスについて

相互運用性について

エンタープライズ環境で、さまざまな言語で記述された複数のアプリケーションを使用することができれば、相互運用性があることになります。こうした既存のアプリケーションは、パーソナルコンピュータのプラットフォームで実行されている場合もあれば、UNIX で実行されている場合もあります。相互運用性のあるエンタープライズ環境では、J2EE プラットフォームで直接サポートされないアプリケーションを使用するスタンドアロン Java テクノロジもサポートされます。

J2EE は、CORBA IIOP (Internet Inter-Orb Protocol) プロトコルのサポートを提供します。CORBA は、ユーザーに意識されることなく、ネットワーク上の分散オブジェクト間の相互運用性を指定するモデルを定義します。これは、外部から参照できる分散オブジェクトの特性を、実装に依存することなく指定する方法を定義することによって、実現されます。

ORB について

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

個々の CORBA オブジェクトが相互に対話することはありません。その代わりに、リモートスタブを介して、ローカルマシンで実行されている ORB に要求を送ります。次に、ローカルの ORB が、IIOP (Internet Inter-Orb Protocol) を使ってその他のマシン上の 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 (サーバント) を検出し、要求を処理して、結果を返します。Java アプリケーションや Java オブジェクトでは、RMI-IIOP テクノロジにより、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。

RMI/IIOP の機能について

CORBA は、アプリケーションが場所に関係なく相互に通信するための ORB を指定する。これはイントラネットの設定によく見られる相互運用性であり、IIOP によって提供されます。次に、RMI over IIOP によって提供される機能の一部を紹介します。

- さまざまな言語で記述されたオブジェクトとの相互運用性
- トランザクションおよびセキュリティコンテキストを伝達する機能
- ORB サービスのプラグアンドプレイ環境

- EJB との相互運用性
- IIOP ベースのネーミングサービス、COSN の ming サービスの使用。EJB 相互運用プロトコルは、JNDI (Java Naming Directory Interface) API を使用する EJB オブジェクトを検索するため、COSNaming を使用することが必要

Sun ONE Application Server にバンドルされている JAVA ORB は、次の機能をサポートします。

- CSIv2 (Common Secure Interoperabillity バージョン 2) の適合性レベル 0
- 完全準拠の COSNaming サービス。IDL インタフェースを実装し、EJB コンテナによる EJBHome 参照の発行を支援する
- IIOP/GIOP バージョン 1.2。CORBA は、アプリケーションが場所に関係なく相互に通信するための ORB を指定する。この相互運用性は、IIOP によって実現する

認証プロセスについて

認証とは、同一性 (ID) を確認するためのプロセスのことです。ネットワークを利用した対話の中で、認証によって各グループは他のグループとの同一性を識別します。証明書は、認証をサポートする方法の 1 つです。

次の 2 種類の認証を適用できます。

サーバー認証 : サーバー認証とは、クライアントによるサーバーの確実な認識を意味します。つまり、特定のネットワークアドレスにあるサーバーに対して責任を持つとされている組織を認識するということです。

クライアント認証 : クライアント認証とは、サーバーによるクライアントの確実な認識を意味します。つまり、クライアントソフトウェアを使用していると見なされる人を認識するということです。

クライアントは、複数の証明書を所有できます。これは、1 人が数種類の ID を所有しているのと同じことです。

ORB の設定

Sun ONE Application Server の各インスタンスには、複数の IIOP リスナーを設定できます。デフォルトでは、IIOP リスナーは 1 つだけ設定されます。ORB の IIOP リスナープロパティを設定すると、別のリスナーを追加できます。

また、ORB 監視の有効化、ログメッセージを記録するログレベルの指定、スレッドプールの設定、IIOP リスナーポートの設定と IIOP パスの SSL 設定を行うことができます。この節では、Sun ONE Application Server インスタンスの ORB サポートを設定する方法を説明します。

この節では次の項目について説明します。

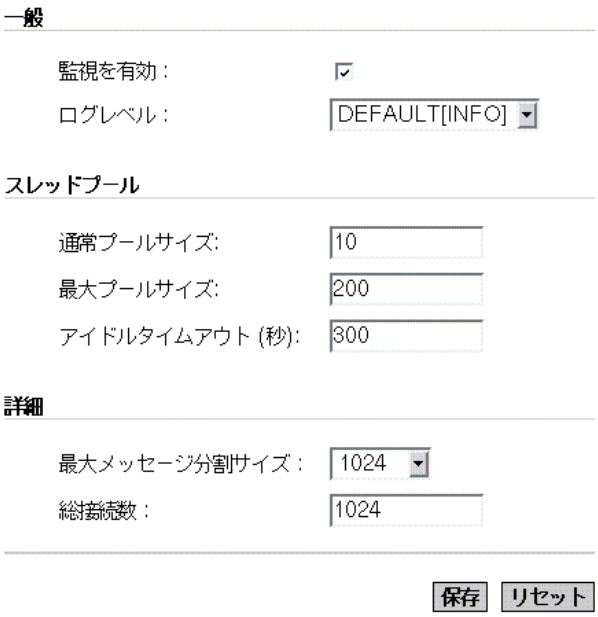
- 一般的な ORB 設定
- ORB の IIOP リスナーの設定

一般的な ORB 設定

管理インタフェースを使用して、監視の有効化、ログレベルの設定、およびスレッドプールのプール設定を行うことができます。一般的な ORB 設定を行うには、次の手順に従います。

1. 管理インタフェースの左側のペインで、ORB 設定を行う Sun ONE Application Server インスタンスを展開します。
2. 「ORB」タブをクリックします。管理インタフェースの右側のペインに、次の「一般的な ORB 設定」の内容が表示されます。

一般的な ORB 設定



3. このウィンドウの「一般」セクションでは、ORB に関する監視の有効化とログレベルの設定を行うことができます。

   a. ORB の監視を有効にするには、「監視を有効」チェックボックスにチェックマークをつけます。

   b. 「ログレベル」ドロップダウンリストから「ログレベル」を選択します。通常、サーバーのデフォルトのログレベルは INFO に設定されています。ORB のデフォルトのログレベルは、サーバーのデフォルトを使用します。ログレベルのドロップダウンリストには、「デフォルト」と表示されます。

      ログレベルによって、重要度が FINEST から FATAL までのメッセージを記録できます。ログレベルを設定することで、ログに表示されるメッセージの細分レベルを選択できます。細分レベルの WARNING では、WARNING、ALERT、SEVERE、および FATAL の各メッセージが表示されます。通常、サーバー全体に対する細分レベルを設定する必要がありますが、この設定を使って Sun ONE Application Server ORB から表示されるメッセージを制御することも可能です。

4. このウィンドウの「スレッドプール」セクションでは、ORB が使う要求スレッドのプール設定を指定できます。

   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。Sun ONE Application Server 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の 2 つの値の間で動的に調整されます。ORB は、ユーザーが指定し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、ユーザーが指定し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。

   a. 「通常プールサイズ」フィールドに、プール内のスレッドの最小数を指定します。「アイドルタイムアウト (秒)」フィールドで指定された期間を超えて、スレッドがアイドル状態にあると、プールはこの指定値に縮小されます。

   b. 「最大プールサイズ」フィールドに、スレッドプールが増大可能なスレッドの最大数を指定します。

   c. 「アイドルタイムアウト (秒)」フィールドに、スレッドプール内のアイドルスレッドが削除されるまでのタイムアウトを指定します。

5. このウィンドウの「詳細」セクションでは、ORB に関する高度なオプションを、次のように設定できます。

   a. 「最大メッセージ分割サイズ」フィールドに、メッセージ分割をサポートできる最大の GIOP 1.2 メッセージサイズを指定します。デフォルトのフラグメントサイズは 1024 です。

   b. 「総接続数」フィールドに、ORB サーバープロセスに許可される受信リモート IIOP 接続の最大数を指定します。

6. 「保存 」をクリックして設定を保存します。変更内容を保存しないで以前の設定に戻したい場合は、「リセット」をクリックします。

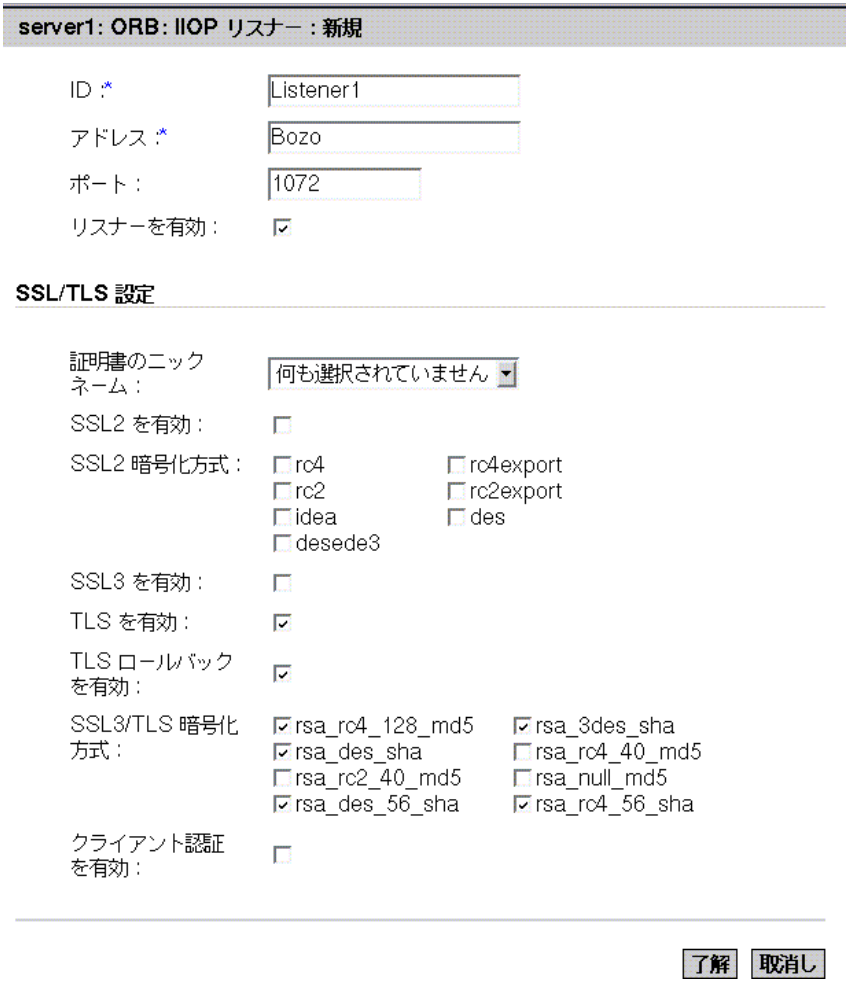
ORB の IIOP リスナーの設定

新しい Sun ONE Application Server インスタンスには、設定済み IIOP リスナーなど、それぞれデフォルトの ORB 設定があります。IIOP リスナーは、特定のポートで待機して、CORBA ベースのクライアントアプリケーションから送信される接続を受け付ける待機ソケットです。1 つのアプリケーションサーバーインスタンスに対して構成できる IIOP リスナーの数に制限はありません。

新しい IIOP リスナーを作成するには、あるいは IIOP リスナーのプロパティを設定するには、次の手順に従います。

1. 管理インタフェースの左側のペインで、ORB プロパティを設定する Sun ONE Application Server インスタンスを展開します。
2. ORB をクリックし、下の「IIOP リスナー (IIOP Listener)」タブを開きます。選択した Sun ONE Application Server インスタンスに設定されている IIOP リスナーが一覧表示されます。
3. 新しい IIOP リスナーを作成するには、「新規 (New)」をクリックします。既存の IIOP リスナーを編集している場合は、そのリスナーを開き、以下の手順に従ってください。「新規」をクリックすると、あるいは既存の IIOP リスナーを開くと、次の「新しい IIOP リスナーの作成」の内容が表示されます。

新しい IIOP リスナーの作成



4. IIOP リスナーの一般的なパラメータを、次のように設定できます。

   a. 「ID」テキストフィールドに、リスナーを識別する名前を入力します。*ORB_Listener1* や *ORB_Listener2* のように、任意の ID を使用できます。

   b. 「アドレス」テキストフィールドに、Sun ONE Application Server のインストールマシンのアドレスを入力します。例のように `machinename.domainname` の形式でマシンアドレスを指定するか、マシンの IP アドレスを入力します。

c. 「ポート」テキストフィールドに、新しい IIOP リスナー固有のポート番号を入力します。デフォルトの IIOP リスナーには、デフォルトのポート番号が設定されています。このポート番号は、変更可能です。ただし、ポート番号を変更する前に、新しいポート番号が他のソフトウェアアプリケーションやプロセスによって使用されていないことを確認してください。

d. 「リスナーを有効」チェックボックスにチェックマークをつけて、リスナーを有効にします。

5. このページの「SSL/TLS 設定」セクションでは、IIOP リスナーのセキュリティを設定できます。すべての暗号化方式を含めて、SSL (Secure Sockets Layer) と TLS (Transport Layer Security) に関連する適切なチェックボックスにチェックマークをつけてください。SSL2 または SSL3/TLS のいずれかのソケットを選択できます。次のように、リスナーに対する SSL/TLS 設定を指定できます。

a. 「証明書のニックネーム」フィールドに、SSL ハンドシェーク時にサーバーがクライアントに渡す証明書のニックネームを入力します。この一覧に表示される証明書は、既にインストールされているものに限られます。

b. 「SSL2 を有効」フィールドにチェックマークをつけて、リスナーパスに対する SSL2 セキュリティオプションを有効にします。

c. SSL2 セキュリティに使用する SSL2 暗号化方式を選択します。必要な暗号化方式に対するチェックボックスにチェックマークをつけます。やむを得ない理由で特定の暗号化方式セットを使わない場合を除き、すべてのセットを利用可能にすることをお勧めします。

d. 「SSL3 を有効」フィールドにチェックマークをつけて、リスナーパスに対する SSL3 セキュリティオプションを有効にします。

e. 「TLS を有効」フィールドにチェックマークをつけて、TLS を有効にします。サーバーへのアクセスを検索するために、ブラウザ側でも TLS を有効にする必要があります。Netscape Navigator 6.0 の TLS と SSL3 の両方にチェックマークをつけます。

f. 「TLS ロールバックを有効」フィールドにチェックマークをつけます。TLS ロールバックを有効にするには、先に TLS を有効にしておく必要があります。また、このオプションを有効にする場合は、SSL3 と SSL2 を無効にする必要があります。Microsoft Internet Explorer 5.0 および 5.5 には、TLS ロールバックオプションを使用します。

g. SSL3 および TLS に使用する SSL3/TLS 暗号化方式を選択します。SSL3 または TLS を有効にした場合にだけ、これらを選択してください。やむを得ない理由で特定の暗号化方式セットを使わない場合を除き、すべてのセットを利用可能にすることをお勧めします。

h. 「クライアント認証を有効」チェックボックスにチェックマークをつけて、クライアント認証と SSL IIOP 接続の ORB リスナーポートを有効にするかどうかを指定します。クライアント認証は、クライアントの証明書を認証するプロセスです。証明書の署名と、信頼できる CA (証明機関) リストに記録された CA につながっている証明書のチェーンを、暗号を使って検証します。

6. 「了解」をクリックして IIOP リスナーの設定を保存します。

注

- Sun ONE Application Server のインストール時には、デフォルトのサーバーインスタンスとして IIOP リスナーが 1 つ作成されます。デフォルト IIOP リスナーポートのポート番号のデフォルト値は 3700 です。
- 個々の IIOP リスナーには一意のポート番号を割り当てる必要があります。また、「アドレス」テキストフィールドには、Sun ONE Application Server がインストールされているマシンのアドレスを指定する必要があります。
- リスナーパスの SSL 設定に関する詳細、およびその他の Sun ONE Application Server のセキュリティに関する詳細については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

第 13 章

アプリケーションの配備

この章では、Sun ONE Application Server のさまざまなモジュールとアプリケーションを配備する方法について説明します。

Sun ONE Application Server のモジュールとアプリケーションには、J2EE 標準の要素と Sun ONE Application Server に固有の要素が含まれます。この章では、Sun ONE Application Server に固有の要素についてだけ詳しく説明しています。

配備するモジュールとアプリケーションのパッケージ化とアセンブルについては、『Sun ONE Application Server 開発者ガイド』を参照してください。

この章では次のトピックについて説明します。

- J2EE モジュールについて
- J2EE アプリケーションについて
- J2EE 標準記述子
- Sun ONE Application Server 記述子
- 命名規則
- 配備ディレクトリの構造
- 実行時環境
- クラスローダについて
- モジュールおよびアプリケーションの配備
- アプリケーション配備記述子ファイル

J2EE モジュールについて

J2EE モジュールは、J2EE コンポーネントの集合で、同一コンテナタイプの 2 つの配備記述子を持ちます。1 つは J2EE 標準の配備記述子で、もう 1 つは Sun ONE Application Server に固有の配備記述子です。J2EE モジュールの種類は次のとおりです。

- **Web アプリケーションアーカイブ (WAR)** : Web アプリケーションは、サーブレット、HTML ページ、クラスなどのリソースの集合で、いくつかの J2EE アプリケーションサーバーにバンドルして配備することができます。WAR ファイルは、Servlet、JSP、JSP タグライブラリ、ユーティリティクラス、静的ページ、クライアントサイドアプレット、Beans、Bean クラス、および配備記述子 (`web.xml` およびオプションで `sun-web.xml`) から構成されます。
- **EJB JAR ファイル** : EJB JAR ファイルは、Enterprise JavaBean をアセンブルするときに使われる標準フォーマットです。このファイルには、Bean クラス (ホーム、リモート、ローカル、および実装)、すべてのユーティリティクラス、および配備記述子 (`ejb-jar.xml`、およびオプションで `sun-ejb-jar.xml`) が含まれています。EJB がコンテナ管理パーシスタンス付きのエンティティ Beans である場合、CMP 配備記述子である `sun-cmp-mapping.xml` ファイルも同様に含まれる場合があります。
- **アプリケーション (RMI/IIOP) クライアント JAR ファイル** : RMI/IIOP クライアントは、Sun ONE Application Server に固有の J2EE クライアントです。RMI/IIOP クライアントでは、J2EE 標準のアプリケーションクライアント仕様がサポートされているだけでなく、Sun ONE Application Server に直接アクセスすることができます。RMI/IIOP クライアントの配備記述子は、`application-client.xml`、およびオプションで `sun-application-client.xml` です。
- **リソース RAR ファイル** : RAR ファイルは、J2EE CA コネクタに適用されます。コネクタモジュールは、デバイスドライバのような働きをします。この移植性のある方法を使用すると、EJB は外部の企業システムにアクセスできます。Sun ONE Application Server の各コネクタには `ra.xml` という J2EE XML ファイルがあります。コネクタには、`sun-ra.xml` という Sun ONE Application Server 配備記述子も必要です。

モジュールを配備した後にクラスローダが正しいクラスを検索できるように、すべてのモジュールのソースコードでパッケージ定義を使う必要があります。

配備記述子内の情報は宣言型であるため、ソースコードを変更しなくても変更できます。J2EE サーバーは、実行時に読み込んだ配備記述子内の情報に従って動作します。

また、EJB JAR および Web モジュールは、次の図に示すように、`.jar` ファイルまたは `.war` ファイルとして個別にアセンブルされ、アプリケーションの外部に個別に配備することもできます。

J2EE アプリケーションについて

J2EE アプリケーションは、1 つまたは複数の J2EE モジュールの論理集合で、アプリケーション配備記述子によって関連付けられています。コンポーネントは、モジュールレベルまたはアプリケーションレベルでアセンブルできます。また、モジュールレベルまたはアプリケーションレベルで配備することもできます。

コンポーネントはモジュールとしてアセンブルされ、その後、配備可能な Sun ONE Application Server アプリケーション `.ear` ファイルにアセンブルされます。

各モジュールには、Sun ONE Application Server 配備記述子と J2EE 配備記述子があります。Sun ONE Application Server の管理インタフェースは、アプリケーションコンポーネントの配備、および Sun ONE Application Server へのリソースの登録に配備記述子を使います。

アプリケーションは、1 つまたは複数のモジュール、オプションの Sun ONE Application Server 配備記述子、および必要な J2EE アプリケーション配備記述子で設定されています。これらのすべてのアイテムが、Java ARchive (`.jar`) ファイル形式で、拡張子 `.ear` を持つ 1 つのファイルにアセンブルされます。

J2EE 標準記述子

J2EE プラットフォームでは、アセンブリおよび配備機能が提供されます。これらの機能では、コンポーネントおよびアプリケーションの標準パッケージとして JAR ファイルが使われ、パラメータのカスタマイズには XML ベースの配備記述子が使われます。J2EE アセンブリおよび配備プロセスの詳細は、『Developing Enterprise Applications with the J2EE, v 1.0』の第 7 章を参照してください。

J2EE 標準配備記述子については、J2EE 仕様書 バージョン 1.3 に説明があります。

配備前に配備記述子の正確さを確認する方法については、『Sun ONE Application Server 開発者ガイド』の配備記述子ベリファイアに関する情報を参照してください。

次の表「J2EE 標準記述子」は、J2EE 標準配備記述子に関する詳細情報の参照先を示しています。左の列は配備記述子、右の列はそれらの記述子に関する詳細情報の参照先を示しています。

J2EE 標準記述子

配備記述子	詳細情報の参照先
`application.xml`	『Java 2 Platform Enterprise Edition Specification, v1.3』の第 8 章「Application Assembly and Deployment - J2EE:application XML DTD」

J2EE 標準記述子 (続き)

配備記述子	詳細情報の参照先
web.xml	『Java Servlet Specification, v2.3』の第 13 章「Deployment Descriptor」および『JavaServer Pages Specification, v1.2』の第 7 章「JSP Pages as XML Documents」および第 5 章「Tag Extensions」
ejb-jar.xml	『Enterprise JavaBeans Specification, v2.0』の第 16 章「Deployment Descriptor」
application-client.xml	『Java 2 Platform Enterprise Edition Specification, v1.3』の第 9 章「Application Clients - J2EE:application-client XML DTD」
ra.xml	『Java 2 Enterprise Edition, J2EE Connector Architecture Specification, v1.0』の第 10 章「Packaging and Deployment」

仕様書は、次の場所にあります。

http://java.sun.com/products/

Sun ONE Application Server 記述子

Sun ONE Application Server には、Sun ONE Application Server に固有の機能を設定するための追加配備記述子があります。これらの記述子は、コネクタモジュールに必要な sun-ra.xml ファイルを除いて、任意で使用できます。

配備前に配備記述子の正確さを確認する方法については、『Sun ONE Application Server 開発者ガイド』の配備記述子ベリファイアに関する情報を参照してください。

次の表「Sun ONE Application Server 記述子」は、Sun ONE Application Server 配備記述子に関する詳細情報の参照先を示しています。左の列は配備記述子、右の列はそれらの記述子に関する詳細情報の参照先を示しています。

Sun ONE Application Server 記述子

配備記述子	詳細情報の参照先
sun-application.xml	356 ページの「アプリケーション配備記述子ファイル」
sun-web.xml	Sun ONE Application Server Web アプリケーション開発者ガイド

Sun ONE Application Server 記述子 (続き)

配備記述子	詳細情報の参照先
sun-ejb-jar.xml および sun-cmp-mapping.xml	Sun ONE Application Server Enterprise Java Beans 開発者ガイド
sun-application-client.xml および sun-acc.xml	Sun ONE Application Server Developer's Guide to Clients
sun-ra.xml	『Sun ONE J2EE CA Service Provider Implementation 管理者ガイド』

注 Sun ONE Application Server 配備記述子は、UNIX システム上でレベル 600 のアクセス権限を持っている必要があります。

すべての Sun ONE Application Server 配備記述子の DTD スキーマファイルは、*install_dir*/appserv/lib/dtds ディレクトリにあります。

命名規則

アプリケーション名および個別に配備された EJBJAR、WAR、およびコネクタ RAR モジュールの名前 (server.xml ファイル内の name 属性によって指定される) は、Sun ONE Application Server 内で一意である必要があります。名前を指定しない場合、ファイル名の最初の部分がデフォルト名となります (.war または .jar 拡張子は含まない)。server.xml の詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

さまざまなタイプのモジュールが、1 つのアプリケーション内で同じ名前を持つ可能性があります。なぜなら、アプリケーションが配備されると、それぞれのモジュールを持つディレクトリ名には、_jar、_war、_rar などのサフィックスが付けられるためです。1 つのアプリケーション内にある同じタイプのモジュールには、一意の名前を付ける必要があります。さらに、データベーススキーマのファイル名も 1 つのアプリケーション内で一意である必要があります。

ejb-jar.xml ファイルの <module-name> に指定するモジュールファイル名や、ejb-jar.xml ファイルの <ejb-name> に指定する EAR ファイル名には、Java パッケージ方式の命名規則を使用することをお勧めします。Java パッケージ方式の命名規則を使えば、名前の衝突は発生しません。この命名規則を、Sun ONE Application Server だけでなく、ほかの J2EE アプリケーションサーバーでも適用することをお勧めします。

EJB の JNDI ルックアップ名も一意でなければなりません。この場合も、一貫した命名規則を作成すると有効です。たとえば、EJB 名にアプリケーション名とモジュール名を追加すると、確実に一意な名前になります。この場合、モジュール `pkgingEJB.jar` 内の EJB の JNDI 名は、アプリケーション `pkging.ear` にパッケージ化されているため、`mycompany.pkging.pkgingEJB.MyEJB` になります。

パッケージとファイル名に、スペースや使用しているオペレーティングシステムで不正となる文字が含まれていないことを確認してください。

配備ディレクトリの構造

アプリケーションを配備すると、個別のモジュールを持つディレクトリ名には、`_jar`、`_war`、`_rar` などのサフィックスが付きます。EAR ファイルの代わりに、`asadmin deploydir` コマンドを使用してディレクトリを配備した場合、ディレクトリ構造はこの規則に従っています。

モジュールおよびアプリケーションのディレクトリ構造は、J2EE 仕様書に示されている構造に準拠します。

次に、Web モジュール、EJB モジュール、クライアントモジュールを含む簡単なアプリケーションのディレクトリ構造の一例を示します。

```
+ converter_1/
|--- converterClient.jar
|--+ META-INF/
|  |--- MANIFEST.MF
|  |--- application.xml
|  '--- sun-application.xml
|--+ war-ic_war/
|  |--- index.jsp
|  |--+ META-INF/
|  |  |--- MANIFEST.MF
|  '--+ WEB-INF/
|     |--- web.xml
|     '--- sun-web.xml
|--+ ejb-jar-ic_jar/
|  |--- Converter.class
|  |--- ConverterBean.class
|  |--- ConverterHome.class
|  '--+ META-INF/
|     |--- MANIFEST.MF
|     |--- ejb-jar.xml
|     '--- sun-ejb-jar.xml
'--+ app-client-ic_jar/
   |--- ConverterClient.class
   '--+ META-INF/
      |--- MANIFEST.MF
      |--- application-client.xml
      '--- sun-application-client.xml
```

次に、個別に配備したコネクタモジュールのディレクトリ構造の例を示します。

```
+ MyConnector/
|--- readme.html
|--- ra.jar
|--- client.jar
|--- win.dll
|--- solaris.so
'--+ META-INF/
   |--- MANIFEST.MF
   |--- ra.xml
   '--- sun-ra.xml
```

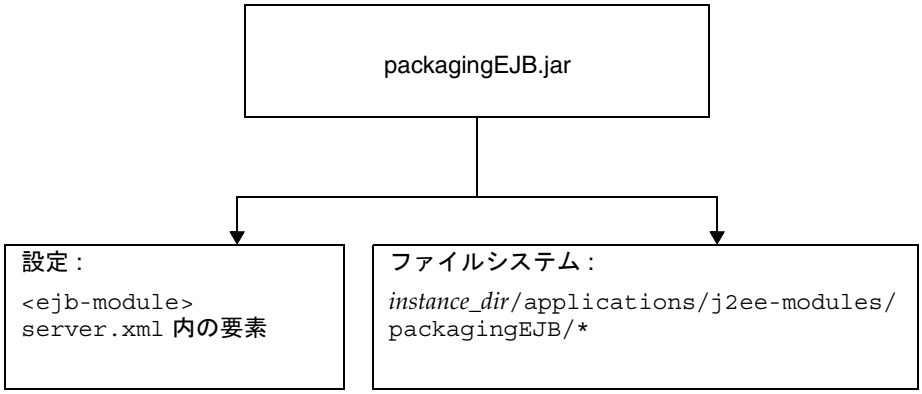
実行時環境

コンポーネントを個別に配備するモジュールとして配備する場合も、アプリケーションとして配備する場合も、配備は、ファイルシステムやサーバー設定に影響を及ぼします。次に示す図「モジュールの実行時環境」と「アプリケーションの実行時環境」を参照してください。

モジュールの実行時環境

次の図「モジュールの実行時環境」は、モジュールベースで個別に配備した場合の実行時環境を示しています。

モジュールの実行時環境



ファイルシステムのエントリとして、モジュールは次のように抽出されます。

instance_dir/applications/j2ee-modules/*module_name*
instance_dir/generated/ejb/j2ee-modules/*module_name*
instance_dir/generated/jsp/j2ee-modules/*module_name*

generated/ejb ディレクトリには、スタブとタイがあり、generated/jsp ディレクトリには、コンパイル済みの JSP があります。

ライフサイクルモジュールは、次の手順で抽出されます。

instance_dir/applications/lifecycle-modules/*module_name*

設定エントリは、server.xml 内に次のように追加されます。

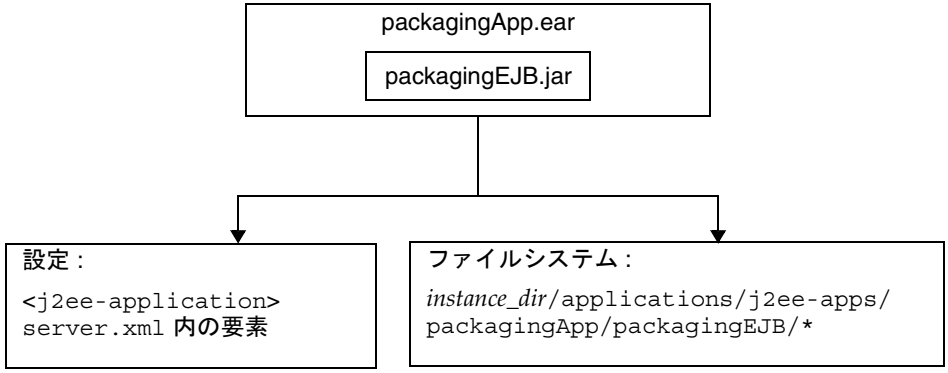
```
<server>
   <applications>
      <type-module>...module configuration...
      </type-module>
   </applications>
</server>
```

server.xml 内のモジュールの *type* は、lifecycle、ejb、web、または connector のいずれかになります。server.xml の詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

アプリケーションの実行時環境

次の図「アプリケーションの実行時環境」は、アプリケーションベースで配備した場合の実行時環境を示しています。

アプリケーションの実行時環境



ファイルシステムのエントリとして、アプリケーションは次のように抽出されます。

instance_dir/applications/j2ee-apps/*app_name*
instance_dir/generated/ejb/j2ee-apps/*app_name*
instance_dir/generated/jsp/j2ee-apps/*app_name*

generated/ejb ディレクトリには、スタブとタイがあり、generated/jsp ディレクトリには、コンパイル済みの JSP があります。

設定エントリは、server.xml 内に次のように追加されます。

```
<server>
   <applications>
      <j2ee-application>...application configuration...
      </j2ee-application>
   </applications>
</server>
```

server.xml の詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

FastJavac コンパイラを使用するための server.xml の設定

デフォルトでは、Sun ONE Application Server は組み込みの JDK コンパイラを使って配備時にアプリケーションをコンパイルします。また、より高速のコンパイルが可能な Sun One Studio の FastJavac コンパイラを配備時に使うこともできます。

Solaris バンドル版インストールの場合、FastJavac コンパイラの場所は透過的ではありません。FastJavac コンパイラを使用するためには、次のように、管理サーバーの server.xml にコンパイラのパスを設定する必要があります。

次のように、server.xml の java-config 要素に jvm-option を追加します。

```
<java-config java-home="/<install-dir>/jdk" server-classpath="....." >
jvm-options>-Dcom.sun.aas.deployment.java.compiler=/<install-dir>/studio
4/bin/fastjavac/fastjavac.sun</jvm-options>
<property name="com.sun.aas.deployment.java.compiler.options"
value="-jdk /<install-dir>/jdk" />
</java-config>
```

クラスローダについて

Sun ONE Application Server クラスローダについて理解すると、サポートしている JAR ファイルとリソースファイルをモジュールおよびアプリケーションのどこに配備するかや、その方法を決定しやすくなります。

Java 仮想マシン (JVM) のクラスローダは、依存関係の解決に必要な Java クラスファイルを動的に読み込みます。たとえば、java.util.Enumeration のインスタンスを作成する場合は、クラスローダの 1 つが関連するクラスを実行時環境に読み込みます。クラスローダの詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

モジュールおよびアプリケーションの配備

この節では、J2EE のアプリケーションおよびモジュールを Sun ONE Application Server に配備する方法について説明します。この節には、次の項目があります。

- 配備名とエラー
- 配備のライフサイクル
- モジュールまたはアプリケーションの配備
- WAR モジュールの配備
- EJB JAR モジュールの配備
- ライフサイクルモジュールの配備
- RMI/IIOP クライアントの配備
- J2EE CA リソースアダプタの配備
- 静的コンテンツの配備
- 共有フレームワークへのアクセス

配備名とエラー

アプリケーションまたはモジュールを配備すると、一意の名前が `server.xml` ファイルに作成されます。この名前を変更しないでください。配備時に、サーバーは名前の重複を検出し、一意の名前を持たないアプリケーションまたはモジュールをロードしません。この場合、サーバーログにメッセージが送信されます。詳細は、341 ページの「命名規則」を参照してください。

配備中にエラーが発生すると、アプリケーションまたはモジュールは配備されません。アプリケーション内のモジュールにエラーが含まれる場合、そのアプリケーション全体が配備されません。

`server.xml` の詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

配備のライフサイクル

アプリケーションが初めて配備された後、修正、再読み込み、再配備、無効化、再有効化、および配備取消し (サーバーから削除) されることがあります。この節には、配備のライフサイクルに関連する次の項目があります。

- 動的配備

- 配備されたアプリケーションまたはモジュールの無効化
- 動的再読み込み

動的配備

サーバーを再起動せずにアプリケーションまたはモジュールを配備、再配備、および配備取消しすることができます。これを動的配備と呼びます。

動的な配備は、主にサーバーを再起動せずに新しいアプリケーションおよびモジュールを運用環境でオンラインにするために使用されます。ただし、再配備を行うと、再配備中に実行されていたセッションが無効になります。クライアントはセッションを実行し直す必要があります。

配備されたアプリケーションまたはモジュールの無効化

配備されたアプリケーションまたはモジュールをサーバーから削除しないで無効にすることができます。各アプリケーションまたはモジュールは `server.xml` ファイルに `enabled` 属性があり、対応するオプションが管理インタフェースにあります。`server.xml` の詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

動的再読み込み

動的再読み込みを有効にすると、コードを変更したときにアプリケーションまたはモジュールを再配備する必要がありません。必要となるのは、変更した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ことだけです。サーバーは、定期的に変更を確認して、アプリケーションを変更に合わせて自動的かつ動的に再配備します。

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。動的な再読み込みは、パフォーマンスが低下することがあるので運用環境にはお勧めしません。また、再読み込みを行うと、再読み込み中に実行されていたセッションが無効になります。クライアントはセッションを実行し直す必要があります。

動的再読み込みを有効にするには、次のいずれかを行います。

- 管理インタフェースを使用する :
  a. サーバーインスタンスの下にある「アプリケーション」コンポーネントを開きます。
  b. 「アプリケーション」ページに移動します。
  c. 「再読み込みを有効」ボックスをオンにして動的再読み込みを有効にします。
  d. 「再読込のポーリング間隔」フィールドに秒数を入力して、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確認して動的に再読み込みする間隔を設定します。

e. 「保存」ボタンをクリックします。

f. サーバーインスタンスのページを表示し、「変更の適用」ボタンを選択します。

- `server.xml` ファイルの `applications` 要素の次の属性を編集します。
  - `dynamic-reload-enabled="true"` に設定して、動的再読み込みを有効にします。
  - `dynamic-reload-poll-interval-in-seconds` で、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確認して動的に再読み込みする間隔を設定します。

  `server.xml` の詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

さらに、新しいサーブレットファイルの読み込み、変更に関連する EJB の再読み込み、または配備記述子の変更の再読み込みを行うには、次の操作を行う必要があります。

1. 配備されたアプリケーションのルートに `.reload` という名前の空のファイルを作成します。

   instance_dir`/applications/j2ee-apps/`*app_name*`/.reload`

   または個別に配備されたモジュールに作成します。

   instance_dir`/applications/j2ee-modules/`*module_name*`/.reload`

2. 上記の変更を行うたびに、`.reload` ファイルのタイムスタンプ (UNIX では `touch.reload`) を明示的に更新します。

JSP では、`sun-web.xml` ファイルの `jsp-config` 要素にある `reload-interval` プロパティで設定した頻度で、変更が自動的に再読み込みされます。JSP の動的再読み込みを無効にするには、`reload-interval="-1"` に設定します。

配備ツール

ここでは、モジュールとアプリケーションを配備するときに使用するツールについて説明します。次の配備ツールがあります。

- asadmin ユーティリティ
- 管理インタフェース
- Sun ONE Studio

asadmin ユーティリティ

asadmin ユーティリティを使用すると、アプリケーションおよび個別に配備されたモジュールをローカルサーバー上に配備および配備取消しができます。複数マシンへの同時配備はサポートされていません。ここでは、asadmin ユーティリティについて簡単に解説します。

ライフサイクルモジュールを配備する場合は、353 ページの「ライフサイクルモジュールの配備」を参照してください。

asadmin deploy

asadmin deploy コマンドを使用すると、WAR、JAR、RAR、または EAR ファイルを配備できます。アプリケーションを配備するには、コマンドに --type application を指定します。個別のモジュールを配備するには、--type ejb、web、または connector を指定します。構文は次のとおりです。オプションパラメータにデフォルト値がある場合は、その値を表示しています。

asadmin deploy --user *admin_user* [--password *admin_password*] [--host localhost] [-port 4848] [--secure | -s] [--virtualservers *virtual_servers*] [--type application|ejb|web|connector] [--contextroot *contextroot*] [--force=true] [--precompilejsp=false] [--name *component_name*] [--upload=true] [--retrieve *local_dirpath*] [--instance *instance_name*] *filepath*

たとえば、次のコマンドは、個別の EJB モジュールを配備します。

```
asadmin deploy --user jadams --password secret --host localhost
--port 4848 --type ejb --instance server1 packagingEJB.jar
```

asadmin deploydir

asadmin deploydir コマンドを使用すると、オープンディレクトリ構造内のアプリケーションまたはモジュールを配備できます。ディレクトリ構造は、342 ページの「配備ディレクトリの構造」に指定されているとおりにする必要があります。*dirpath* の場所が *instance_dir*/applications/j2ee-apps の下または *instance_dir*/applications/j2ee-modules の下のどちらにあるかによって、それがアプリケーションか個別に配備されたモジュールかが決まります。構文は次のとおりです。オプションパラメータにデフォルト値がある場合は、その値を表示しています。

asadmin deploydir --user *admin_user* [--password *admin_password*] [--host localhost] [-port 4848] [--secure | -s] [--virtualservers *virtual_servers*] [--type application|ejb|web|connector] [--contextroot *contextroot*] [--force=true] [--precompilejsp=false] [--name *component_name*] [--instance *instance_name*] *dirpath*

たとえば、次のコマンドは、個別の EJB モジュールを配備します。

```
asadmin deploydir --user jadams --password secret --host localhost
--port 4848 --type ejb --instance server1 packagingEJB
```

asadmin undeploy

`asadmin undeploy` コマンドを使用すると、アプリケーションまたはモジュールを配備取消しができます。アプリケーションを配備を取消すには、コマンド内に `--type app` を指定します。個別のモジュールを配備を取消すには、`--type ejb`、`web`、または `connector` を指定します。構文は次のとおりです。オプションパラメータにデフォルト値がある場合は、その値を表示しています。

```
asadmin undeploy --user admin_user [--password admin_password] [--host
localhost] [-port 4848] [--secure | -s] [--type
application|ejb|web|connector] [--instance instance_name] component_name
```

たとえば、次のコマンドは、個別の EJB モジュールの配備を取消します。

```
asadmin undeploy --user jadams --password secret --host localhost
--port 4848 --type ejb --instance server1 packagingEJB
```

管理インタフェース

管理インタフェースを使用すると、モジュールとアプリケーションをローカルおよびリモートの Sun ONE Application Server サイトに配備できます。このツールを使うには、次の手順で行います。

1. サーバーインスタンスの下にある「アプリケーション」コンポーネントを開きます。
2. 「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」、「コネクタモジュール」、「EJB モジュール」のいずれかのページに移動します。
3. 「配備」ボタンをクリックします。
4. モジュールまたはアプリケーションへのフルパスを入力し (または「ブラウズ」をクリックして指定)、「了解」ボタンをクリックします。
5. モジュールまたはアプリケーションの名前を入力します。

   モジュールまたはアプリケーションがすでに配備されていれば、適切なボックスをチェックして、それを再配備することもできます。これはオプションです。
6. 仮想サーバー名の隣のボックスにチェックマークをつけて、1 つまたは複数の仮想サーバーにアプリケーションまたはモジュールを割り当てます。
7. 「了解」ボタンをクリックします。

ライフサイクルモジュールを配備する場合は、353 ページの「ライフサイクルモジュールの配備」を参照してください。

Sun ONE Studio

J2EE アプリケーションとモジュールの配備には Sun ONE Studio 4 を利用できます。Sun ONE Studio に関する詳細は、『Sun ONE Studio 4, Enterprise Edition のチュートリアル』を参照してください。

注 Sun ONE Studio では、モジュールまたはアプリケーションの配備を「実行」と呼びます。「実行」には、サーバーが稼動していることの確認、およびモジュールまたはアプリケーションをアクティブにする正しい URL の表示も含まれます。

モジュールまたはアプリケーションの配備

アプリケーションまたはアプリケーションから独立した個別のモジュールを配備することができます。アプリケーションベースまたは個別のモジュールベースで配備したときの実行時環境およびファイルシステムについては、344 ページの「実行時環境」を参照してください。

次のものがコンポーネントにアクセスする場合は、個別のモジュールベースで配備することをお勧めします。

- ほかのモジュール
- J2EE アプリケーション
- RMI/IIOP クライアント (モジュールベースで配備すると、RMI/IIOP クライアント、サーブレット、または EJB から Bean に共有アクセスできる)

複数のモジュールを 1 つの EAR ファイルに結合すると、1 つのモジュールとして配備できるようになります。これは、EAR のモジュールを個別に配備するのと似ています。

WAR モジュールの配備

WAR モジュールの配備は、349 ページの「配備ツール」で説明されている方法で行うことができます。

JSP 用の生成されたソースは、`-keepgenerated` プロパティを `sun-web.xml` 内の `jsp-config` 要素に追加することによって保持できます。WAR モジュールを配備するときにこのプロパティを追加すると、生成されたソースが保存されます。保存先は、アプリケーションの場合は

instance_dir/generated/jsp/j2ee-apps/*app_name*/*module_name*、個別に配備された Web モジュールの場合は *instance_dir*/generated/jsp/j2ee-modules/*module_name* です。-keepgenerated プロパティの詳細については、『Sun ONE Application Server Web アプリケーション開発者ガイド』を参照してください。

EJB JAR モジュールの配備

EJB JAR モジュールの配備は、349 ページの「配備ツール」に記載されている方法で行うことができます。

スタブとタイ用の生成されたソースは、-keepgenerated フラグを server.xml 内の java-config 要素の rmic-options 属性に追加することによって保持できます。EJB JAR モジュールを配備するときにこのフラグを追加すると、生成されたソースが保存されます。保存先は、アプリケーションの場合は *instance_dir*/generated/ejb/j2ee-apps/*app_name*/*module_name*、個別に配備された Web モジュールの場合は *instance_dir*/generated/ejb/j2ee-modules/*module_name* です。-keepgenerated フラグの詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

ライフサイクルモジュールの配備

ライフサイクルモジュールに関する一般的な情報については、『Sun ONE Application Server 開発者ガイド』を参照してください。

ライフサイクルモジュールを配備するには、次のツールを使います。

- asadmin ユーティリティ
- 管理インタフェース

asadmin ユーティリティ

ライフサイクルモジュールを配備するには、asadmin create-lifecycle-module コマンドを使います。構文は次のとおりです。オプションパラメータにデフォルト値がある場合は、その値を表示しています。

```
asadmin create-lifecycle-module --user admin_user [--password
admin_password] [--host localhost] [-port 4848] [--secure | -s]
[--instance instance_name] --classname classname [--classpath classpath]
[--loadorder load_order_number] [--failurefatal=false] [--enabled=true]
[--description text_description] [--property (name=value)[:name=value]*]
modulename
```

次に例を示します。

```
asadmin create-lifecycle-module --user jadams --password secret
--host localhost --port 4848 --instance server1 --classname
RMIServer MyRMIServer
```

ライフサイクルモジュールの配備を取消すには、asadmin delete-lifecycle-module コマンドを使います。構文は次のとおりです。オプションパラメータにデフォルト値がある場合は、その値を表示しています。

```
asadmin delete-lifecycle-module --user admin_user [--password
admin_password] [--host localhost] [-port 4848] [--secure | -s]
[--instance instance_name] module_name
```

次に例を示します。

```
asadmin delete-lifecycle-module --user jadams --password secret
--host localhost --port 4848 --instance server1 MyRMIServer
```

サーバーインスタンス上に配備されたライフサイクルモジュールをリスト表示するときは、asadmin list-lifecycle-modules コマンドを使います。構文は次のとおりです。オプションパラメータにデフォルト値がある場合は、その値を表示しています。

```
asadmin list-lifecycle-modules --user admin_user [--password
admin_password] [--host localhost] [-port 4848] instance_name
```

次に例を示します。

```
asadmin list-lifecycle-module --user jadams --password secret --host
localhost --port 4848 server1
```

管理インタフェース

管理インタフェースを使ってライフサイクルモジュールを配備することもできます。次の手順に従います。

1. サーバーインスタンスの下にある「アプリケーション」コンポーネントを開きます。
2. 「ライフサイクルモジュール」ページに移動します。
3. 「配備」ボタンをクリックします。
4. 次の情報を入力します。
   - 「名前」(必須) - ライフサイクルモジュールの名前
   - 「クラス名」(必須) - ライフサイクルモジュールのクラスファイルの完全修飾された名前
   - 「クラスパス」(オプション) - ライフサイクルモジュールのクラスパス。モジュールの場所を指定する。デフォルトの場所は、アプリケーションのルートディレクトリの下

- 「読み込み順序」(オプション) - 起動時にライフサイクルモジュールが読み込まれる順序を決定する。モジュールに指定された数値が小さいほど、早く読み込まれる。値の範囲は、101 から オペレーティングシステムの MAXINT まで。～ 100 までの値は予約されている
- 「致命的な障害」(オプション) - ライフサイクルモジュールが失敗した場合にサーバーをシャットダウンするかどうかを決定する。デフォルトは false
- 「ライフサイクルを有効」(オプション) - ライフサイクルモジュールを有効にするかどうかを決定する。デフォルトは true

5. 「了解」ボタンをクリックします。

RMI/IIOP クライアントの配備

EJB とやりとりするクライアントの場合のみ、配備が必要です。RMI/IIOP クライアントは、3 つの手順で配備します。

1. RMI/IIOP クライアントがアクセスする EAR または EJB JAR を配備します。
2. 必要なクライアントファイルをアセンブルし、クライアントを配備します。
3. 配備後、クライアント JAR ファイルは、アプリケーション用の次の場所に作成されます。

   instance_dir/applications/j2ee-apps/*app_name*/*app_name*Client.jar

   または、個別に配備されたモジュール用の次の場所に作成されます。

   instance_dir/applications/j2ee-modules/*module_name*/*module_name*Client.jar

   クライアント JAR には、RMI/IIOP クライアント用のタイおよび必要なクラスが含まれています。このファイルをクライアントマシンにコピーし、クライアント上の APPCPATH 環境変数がこの JAR を示すように設定します。

これで、クライアントを実行する準備ができました。詳細は、『Sun ONE Application Server Developer's Guide to Clients』を参照してください。

J2EE CA リソースアダプタの配備

コネクタモジュールの配備は、349 ページの「配備ツール」で説明されている方法で行うことができます。

静的コンテンツの配備

静的コンテンツ (HTML、画像など) は、Web サーバー上および Sun ONE Application Server 上で管理できます。ただし、WAR が登録されているときは、静的コンテンツはアプリケーションサーバーに配備されます。Sun ONE Application Server に付属するすべてのサンプルは、アプリケーションサーバー上で静的コンテンツを管理します。

たとえば、アプリケーションサーバー上の静的ファイル `index.html` にアクセスするには、次のパスを使います。

`http://`*server*`:`*port*`/NASApp/`*context_root*`/index.html`

共有フレームワークへのアクセス

J2EE のアプリケーションとモジュールで共有フレームワーククラス (コンポーネント、ライブラリなど) を使用する場合、それらのクラスはアプリケーションやモジュールではなくシステムクラスローダまたは共有クラスローダのパスに配備できます。サイズが大きい共有ライブラリを、そのライブラリを使用するすべてのモジュールにアセンブルする場合、サーバーへの登録に多くの時間がかかります。また、同一クラスの複数のインスタンスが独自のクラスローダを使用すると、リソースの浪費になります。

システムクラスローダについては、346 ページの「クラスローダについて」を参照してください。

アプリケーション配備記述子ファイル

Sun ONE Application Server には、次の 2 種類の配備記述子ファイルがあります。

- 『Java Servlet Specification, v2.3』の第 13 章「Deployment Descriptors」で説明している J2EE 標準ファイル (`application.xml`)
- この章で説明した Sun ONE Application Server に固有のオプションファイル (`sun-application.xml`)

アプリケーション配備記述子ファイルの詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

第 3 部

HTTP サーバーの機能と仮想サーバーの管理

第 14 章

HTTP 機能の設定

この章では、Sun ONE Application Server の HTTP 関連機能の設定方法を説明します。仮想サーバーと HTTP リスナーに関連する詳細設定については、第 15 章「仮想サーバーの使用」を参照してください。

この章では次のトピックについて説明します。

- HTTP 機能について
- ファイルキャッシュの設定
- サーバーのパフォーマンスの調整
- HTTP のサービス品質の設定
- スレッドプールの追加と使用
- 詳細設定の編集
- MIME タイプの設定

HTTP 機能について

Sun ONE Application Server の HTTP 機能には、アプリケーションサーバーインスタンスのパフォーマンスレベルの設定、パフォーマンスチューニング関連パラメータの設定、パフォーマンスを改善するためのファイルキャッシュの使用などがあります。これらの設定情報は、`init.conf` と `server.xml` という 2 つの設定ファイルに格納されています。`init.conf` ファイルの設定は、「詳細設定」ページで編集します。詳細については、363 ページの「詳細設定の編集」を参照してください。

その他の編集可能なプロパティは、`server.xml` ファイルの `http-service` 要素内に格納されています。`init.conf` ファイルおよび `server.xml` ファイルの詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

ファイルキャッシュの設定

Sun ONE Application Server は、静的な情報を迅速に処理するため、ファイルキャッシュを使用します。ファイルキャッシュには、ファイルに関する情報と静的なファイルコンテンツが含まれます。ファイルキャッシュには、サーバー解析 HTML の処理を高速化するために使用する情報も格納されます。

ファイルキャッシュはデフォルトで有効になっています。ファイルキャッシュの設定情報は、`nsfc.conf` ファイルに格納されています。このファイルは、ファイルのキャッシュパラメータがデフォルトから変更されている場合にのみ存在します。`nsfc.conf` の詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

ファイルキャッシュを設定するには、次の手順を実行します。

1. 左側のペインの「HTTP サーバー」をクリックします。
2. 「ファイルキャッシュ」タブをクリックします。
3. フィールドに適切な値を入力します。
4. 「了解」をクリックします。

ファイルキャッシュを使ってパフォーマンスを改善する方法については、『Sun ONE Application Server パフォーマンスおよびチューニングガイド』を参照してください。

サーバーのパフォーマンスの調整

「性能の調整」ページでは、処理できる要求の数、稼動していない状態になってからタイムアウトになるまでの要求の存続時間、および DNS によるクライアント IP の逆方向検索を行うかどうかなどを指定して、Sun ONE Application Server のパフォーマンスを制御する設定情報を設定できます。DNS を使用する場合は、パフォーマンス関連機能 (たとえば非同期 DNS を使用するかどうか) と、DNS キャッシュ設定を設定できます。

詳細は、『Sun ONE Application Server パフォーマンスおよびチューニングガイド』を参照してください。

パフォーマンスを調整するには、次の手順に従います。

1. 左側のペインの「HTTP サーバー」をクリックします。
2. 「調整」タブをクリックします。
3. フィールドに適切な値を入力します。
4. 「了解」をクリックします。

このほか、管理インタフェースを使って調整できる設定の詳細については、オンラインヘルプを参照してください。

HTTP のサービス品質の設定

サービス品質とは、ユーザーがサーバーに設定するパフォーマンス制限のことです。たとえば、ISP は、許可する帯域幅に応じて仮想サーバーの課金額を変えたいことがあります。

特定の仮想サーバーのサービス品質を使用するには、サーバーインスタンスのサービス品質を事前に有効化し、値を設定しておく必要があります。

サーバーインスタンスのサービス品質を設定するには、次の手順に従います。

1. 左側のペインの「HTTP サーバー」をクリックします。
2. 「QOS」タブをクリックします。
3. サービス品質を全体で使用可能にするには、「有効」をクリックします。

   デフォルトでは、サービス品質は無効になっています。サービス品質を有効にすると、サーバーのオーバーヘッドがわずかに増えます。
4. 「QOS 再計算時間間隔」を選択します。

   再計算時間の間隔は、帯域幅の計算間隔をミリ秒単位で表しています。デフォルトは 100 ミリ秒です。
5. 「QOS メトリック間隔」を選択します。

   測定時間は、トラフィックの測定間隔を秒単位で示しています。デフォルトは 30 秒。この時間に測定されたすべての帯域幅を平均して、1 秒あたりのバイト数が得られます。

   サイズの大きいファイルを転送することが多い場合は、このフィールドの値を大きくします (数分またはそれ以上)。サイズの大きいファイルを転送する際、メトリック間隔が短いと、許容帯域幅がすべて占有される可能性があります。この場合、最大帯域幅の設定が有効になっていると接続が拒否されます。帯域幅はメトリック間隔によって平均化されるため、間隔を長くすれば、サイズの大きいファイルによるトラフィックスパイクを防ぐことができます。

   帯域幅の制限値が使用可能な帯域幅よりもはるかに小さい場合 (たとえば、帯域幅の制限値が 1M バイト / 秒で、バックボーンとの接続が 1G バイト / 秒の場合) は、メトリック間隔を短くする必要があります。

   転送する静的ファイルのサイズが大きいという問題の解決策と、帯域幅の制限値が使用可能な帯域幅よりもはるかに小さいという問題の解決策は相反しています。ユーザーは、どちらの問題を調整するかを決定する必要があります。
6. サーバーに対して、帯域幅の制限をバイト / 秒単位で設定します。

7. 帯域幅の制限設定を実施するかどうかを選択します。

   帯域幅の制限設定を実施すると、帯域幅の制限値に達した場合にそれ以上の接続が拒否されます。

   帯域幅の制限設定を実施しないときは、制限値を超えた場合にサーバーのエラーログにメッセージが記録されます。

8. このサーバーの最大接続許可数を選択します。

   この数は、同時に処理する要求の数です。

9. 接続制限の設定を強制するかどうかを選択します。

   接続数の制限設定を実施すると、接続数が制限値に達した場合にそれ以上の接続が拒否されます。

   接続数の制限設定を実施しないときは、制限値を超えた場合にサーバーのエラーログにメッセージが記録されます。

10. 別の名前 / 値ペアを指定するときは、「プロパティ」ボタンをクリックします。

11. 「了解」をクリックします。

コマンド行インタフェースの `asadmin` ユーティリティを使ってサービス品質を設定するときは、次のコマンドを使います。

- `create-http-qos`
- `delete-http-qos`

これらのコマンドの構文は次のとおりです。

`asadmin create-http-qos --user` *admin_user* `[--password` *password*`] [--host` *hostname*`] [--port` *admin_port*`] [--secure | -s] [--passwordfile` *file_name*`] [--virtualserver` *virtual_server_id*`] [--bwlimit` *bandwidth_limit*`] [--enforcebwlimit` *enforce_bandwidth_limit*`] [--connlimit` *connection_limit*`] [--enforceconnlimit` *enforce_connection_limit*`]` *instancename*

`asadmin delete-http-qos --user` *admin_user* `[--password` *password*`] [--host` *hostname*`] [--port` *admin_port*`] [--secure | -s] [--passwordfile` *file_name*`] [--virtualserver` *virtual_server_id*`]` *instancename*

仮想サーバーを指定してこれらのコマンドを実行すると、その仮想サーバーのサービス品質情報が作成または削除されます。仮想サーバーを指定しない場合は、コマンドはサーバーインスタンスに適用されます。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

サービス品質機能の制限については、151 ページの「CLI によるトランザクションサービスの管理」を参照してください。

スレッドプールの追加と使用

スレッドプールを使って、特定のサービスに一定数のスレッドを割り当て、それ以上のスレッドを使用できないようにすることができます。スレッドプールは、スレッドに対して安全でないプラグインを実行するときにも使用できます。プールの最大スレッド数を 1 に設定すると、指定されたサービス機能で許可される要求数が 1 つに制限されます。

スレッドプールを追加するときは、スレッドの最大数および最小数、スタックサイズ、および待ち行列サイズも指定します。

スレッドプールを追加するには、次の手順に従います。

1. 左側のペインの「HTTP サーバー」をクリックします。
2. 「スレッドプール」をクリックします。
3. フィールドに適切な値を入力します。
4. 「了解」をクリックします。

ページの下部にスレッドプールが表示されます。スレッドプールを編集または削除するには、その横の「編集」または「削除」ボタンをクリックします。

スレッドプールの設定が完了したら、それを特定のサービス用に使用するように指定します。

スレッドプールを使ってパフォーマンスを改善する方法については、『パフォーマンスおよびチューニングガイド』を参照してください。

詳細設定の編集

Sun ONE Application Server は起動時に、*instance_dir*/config/ ディレクトリの init.conf というファイルを検索して、サーバーインスタンスの動作と設定に関係がするグローバル変数セットを設定します。Sun ONE Application Server は init.conf で定義した指令をすべて実行します。

これらの設定は、「詳細設定」ページに表示されます。init.conf ファイルの特定の設定を変更すると、次の項目に影響が出ます。

- DNS
- SSL
- パフォーマンス
- CGI
- Keep-Alive

- ログ

init.conf の指令の一覧とその説明は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』に記載されています。

詳細設定を編集するには、次の手順に従います。

1. 左側のペインの「HTTP サーバー」をクリックします。
2. 「詳細」タブを選択します。
3. 変更する設定の種類 (DNS、SSL、など) をクリックします。
4. 設定内容に必要な変更を加えて、「了解」をクリックします。

各種類の設定の詳細については、オンラインヘルプを参照してください。

MIME タイプの設定

「MIM タイプ (Mime Types)」ページでは、サーバーの MIME ファイルを編集できます。MIME (Multi-purpose Internet Mail Extension) タイプを使って、ユーザーのシステムでサポートされるマルチメディアファイルのタイプを制御できます。また、特定のサーバーファイルタイプに属するファイル拡張子も指定できます。たとえば、CGI プログラムになるファイルを指定できます。

必要な数の MIME タイプファイルを作成し、アプリケーションサーバーインスタンスや仮想サーバーに関連付けることができます。サーバー上にデフォルトで存在する MIME タイプファイル mime.types は削除できません。

新しい MIME タイプファイルを作成するには、次の手順に従います。

1. 左側のペインの「HTTP サーバー」の下の「MIME タイプファイル」をクリックします。
2. 右側のペインの「新規」をクリックします。
3. MIME ファイルの識別子とファイル名を入力します。
4. 「了解」をクリックします。

MIME ファイル内の定義を編集するには、次の手順に従います。

1. 左側のペインの「HTTP サーバー」の下の「MIME タイプファイル」の横にあるアイコンをクリックし、ビューを展開します。
2. 編集したい MIME ファイルの ID をクリックします。
3. このページで、指定した ID に関連する MIME ファイルの名前を編集します。
4. MIME ファイルの拡張子を編集するときは、「MIME ファイルを編集」をクリックします。

5. 既存のエントリを編集するときは、そのエントリのとなりの「編集」をクリックします。
6. 表示されるページで適切な変更を加え、「MIME タイプを変更」をクリックします。
7. MIME タイプを削除するときは、そのタイプのとなりの「削除」をクリックします。
8. 新しい MIME タイプを追加するときは、各フィールドにカテゴリ、コンテンツタイプ、ファイルサフィックスを入力し、「新規タイプ」をクリックします。

コマンド行インタフェースの `asadmin` ユーティリティを使って MIME タイプ を設定するときは、次のコマンドを使います。

- `create-mime`
- `delete-mime`
- `list-mimes`

これらのコマンドの構文は次のとおりです。

`asadmin create-mime --user` *admin_user* `[--password` *password*`] [--host` *hostname*`] [--port` *admin_port*`] [--secure | -s] [--passwordfile` *file_name*`] [--instance` *instancename*`] --mimefile` *filename mime_id*

`asadmin delete-mime --user` *admin_user* `[--password` *password*`] [--host` *hostname*`] [--port` *admin_port*`] [--secure | -s] [--passwordfile` *file_name*`] [--instance` *instancename*`]` *mime_id*

`asadmin list-mimes --user` *admin_user* `[--password` *password*`] [--host` *hostname*`] [--port` *admin_port*`] [--secure | -s] [--passwordfile` *file_name*`]` *instancename*

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

仮想サーバーで MIME タイプを使用する方法の詳細については、オンラインヘルプと第 15 章「仮想サーバーの使用」を参照してください。

第 15 章

仮想サーバーの使用

この章では、Sun ONE Application Server による仮想サーバーのセットアップおよび管理方法を説明します。仮想サーバーのコンテンツ管理設定の設定方法については、第 16 章「仮想サーバーコンテンツの管理」を参照してください。

この章では次のトピックについて説明します。

- 仮想サーバーの概要
- 仮想サーバーでの Sun ONE Application Server の機能の使用
- HTTP リスナーの作成と設定
- 仮想サーバーの作成と設定
- 仮想サーバーの配備

仮想サーバーの概要

仮想サーバーを使用すると、インストール済みの単一のサーバーで、複数の企業または個人のドメイン名、IP アドレス、およびいくつかのサーバー監視機能を提供することが可能になります。これにより、ユーザーは、あたかも独自の Web サーバーを持っているかのように操作できますが、実際には、管理者がハードウェアを提供し、仮想サーバーを管理しています。

アンバンドル版の Sun ONE Application Server をインストールすると、アプリケーションサーバーインスタンスのデフォルトの仮想サーバーが作成されます。デフォルトのアプリケーションサーバーインスタンス `server1` には、`server1` という名前の仮想サーバーが作成されます。Solaris 9 バンドル版を使う場合は、サーバーインスタンスを作成する必要があります。インスタンスを作成すると、同じ名前の仮想サーバーも同時に作成されます。仮想サーバーは、アプリケーションサーバーインスタンスを作成するたびに作成されます。仮想サーバーの作成方法と設定方法については、379 ページの「仮想サーバーの作成と設定」を参照してください。仮想サーバーの配備方法については、「仮想サーバーの配備」を参照してください。

仮想サーバーは、仮想サーバーごとに使用可能な Sun ONE Application Server の HTTP 機能を制御します。複数の仮想サーバーを使用する必要はありませんが、アプリケーションサーバーインスタンスとともに作成されるデフォルトの仮想サーバーを設定して、このアプリケーションサーバーインスタンスの特定のプロパティを設定する必要はあります。

仮想サーバーの設定は、*instance_dir*/config ディレクトリ内の server.xml ファイルの virtual-server 要素に格納されます。このファイルの詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

仮想サーバーに関連する一部の情報は、obj.conf ファイルに格納されます。obj.conf ファイルは、仮想サーバーごとに用意されます。

この節には次の項目があります。

- HTTP リスナー
- 仮想サーバー
- obj.conf ファイル
- 要求を処理する仮想サーバーの選択
- ドキュメントルート
- アクセスログファイルとサーバーログファイルの使用

HTTP リスナー

サーバーとクライアント間の接続は、HTTP リスナー (待機ソケット) 上で行われます。各 HTTP リスナーには、IP アドレス、ポート番号、返されるサーバー名、およびデフォルトの仮想サーバーが割り当てられます。HTTP リスナーが、マシン上の指定されたポートですべての設定済み IP アドレスを待機するように設定したい場合は、IP アドレスとして 0.0.0.0、any、ANY、INADDR_ANY のいずれかを指定します。HTTP リスナーの作成方法と設定方法については、376 ページの「HTTP リスナーの作成と設定」を参照してください。

アンバンドル版の Sun ONE Application Server をインストールすると、http-listener-1 という HTTP リスナーが自動的に作成されます。この HTTP リスナーは、IP アドレス 0.0.0.0 と、インストール時に HTTP サーバーポート番号として指定したポート番号を使用します。なお、デフォルトのポート番号は 80 です。UNIX 環境で、スーパーユーザー以外のユーザーがインストールした場合は、1024 になります。デフォルトの HTTP リスナーは削除できません。複数の仮想サーバーを使う場合は、そのすべてでデフォルトの HTTP リスナーを使うか、複数の HTTP リスナーを使うかを選択できます。

Solaris 9 バンドル版の Sun ONE Application Server では、サーバーインスタンスの作成時に HTTP リスナーが作成されます。IP アドレスは 0.0.0.0 で、ポート番号はインスタンスの作成時に指定した番号になります。

HTTP リスナーは IP アドレスとポート番号の組み合わせです。このため、IP アドレスが同じでポート番号の異なる HTTP リスナーや、IP アドレスは異なるがポート番号は同じの HTTP リスナーを使用できます。マシンが各アドレスに応答するように設定されていれば、たとえば、1.1.1.1:81 と 1.1.1.1:82、1.1.1.1:81 と 1.2.3.4:81 のような IP アドレスを共存させることができます。しかし、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 IP アドレスを HTTP リスナーに設定することはできません。たとえば、HTTP リスナー 0.0.0.0:80 (ポート 80 上のすべての IP アドレスを待機) を設定すると、1.2.3.4:80 を使用する HTTP リスナーを作成することはできません。

それぞれの HTTP リスナーには、要求に指定されている仮想サーバーに接続できない場合に、要求のリダイレクト先となるデフォルトの仮想サーバーがあります。

さらに、HTTP リスナー内のアクセプタスレッド (受け入れスレッド) の数を指定します。受け入れスレッドは、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入れられ、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、常に十分な数の受け入れスレッドを確保しておくのが理想的ですが、システムに負荷がかかり過ぎない数に抑える必要があります。デフォルトの受け入れスレッド数は 1 です。システム上の CPU ごとに受け入れスレッドを 1 つずつ確保するのが適切です。パフォーマンスに問題があるときは、この値を調整できます。

HTTP リスナーのセキュリティを有効にするかどうか、また、セキュリティの種類 (SSL の種類や暗号化方式の種類など) も指定します。

仮想サーバー

仮想サーバーを作成するときは、まず、その仮想サーバーの種類を決定する必要があります。IP アドレスベースの仮想サーバーか、URL ホストベースの仮想サーバーを作成できます。作成するには、仮想サーバー ID、1 つ以上の HTTP リスナー、および 1 つ以上の URL ホストを指定するだけで済みます。

この節には次の項目があります。

- 仮想サーバーの種類
- IP アドレスベースの仮想サーバー
- URL ホストベースの仮想サーバー
- デフォルトの仮想サーバー

仮想サーバーの種類

すべての仮想サーバーには URL ホストが指定されます。同時に、その仮想サーバーを HTTP リスナーに基づいた IP アドレスに関連づけます。仮想サーバーの HTTP リスナーが特定の IP アドレスを待機する場合、この仮想サーバーは IP アドレスベースの仮想サーバーと呼ばれます。

複数の仮想サーバーが同じ IP アドレスを待機する場合、この仮想サーバーは URL ホストとして扱われ、URL ホストベースの仮想サーバーと呼ばれます。

新しい要求を着信すると、サーバーは、IP アドレスまたは Host ヘッダーの値から、この要求の送信先となる仮想サーバーを決定します。サーバーは、最初に IP アドレスを評価します。詳細については、372 ページの「要求を処理する仮想サーバーの選択」を参照してください。

IP アドレスベースの仮想サーバー

単一のコンピュータに複数の IP アドレスを設定する場合は、オペレーティングシステムでマッピングするか、カードを追加する必要があります。オペレーティングシステムで複数の IP アドレスを設定する際、Windows 環境ではネットワークコントロールパネルを使用します。UNIX 環境では `ifconfig` ユーティリティを使用します。`ifconfig` の使用方法はプラットフォームによって異なります。詳細については、オペレーティングシステム付属のマニュアルを参照してください。

IP アドレスベースの仮想サーバーを作成するときは、特定の IP アドレスを待機する HTTP リスナーを作成します。次に、その HTTP リスナーのデフォルトの仮想サーバーとして関連づけます。仮想サーバーの詳しい配備方法については、386 ページの「仮想サーバーの配備」を参照してください。

URL ホストベースの仮想サーバー

URL ホストベースの仮想サーバーをセットアップする場合は、各仮想サーバーに固有の URL ホストを割り当てます。サーバーは、要求をその Host 要求ヘッダーの内容によって、正しい仮想サーバーに転送します。

たとえば、*aaa*、*bbb*、*ccc* という顧客の仮想サーバーをセットアップし、それぞれの顧客に個別のドメイン名を割り当てるには、まず、各顧客の URL (`www.aaa.com`,、`www.bbb.com`、`www.ccc.com`) が HTTP リスナーの IP アドレスに解釈処理されるように、DNS を設定します。次に、各仮想サーバーの URL ホストを正しく設定します (`www.aaa.com` など)。`/etc/hosts` ファイルでホストと IP アドレスをマップします。

単一の HTTP リスナーに関連付けることができる URL ホストベースの仮想サーバー数に制限はありません。

URL ホストベースの仮想サーバーは、Host 要求ヘッダーを使ってユーザーに正しいページを表示するので、クライアントソフトウェアによってはこの種類の仮想サーバーを使用できない場合もあります。HTTP Host ヘッダーをサポートしない古いクライアントソフトウェアがこれに該当します。これらのクライアントは、HTTP リスナーのデフォルトの仮想サーバーを使用します。

デフォルトの仮想サーバー

URL ホストベースの仮想サーバーは、要求の Host ヘッダーを使って選択されます。エンドユーザーのブラウザが Host ヘッダーを送信しない場合、またはサーバーが指定された Host ヘッダーを検出できない場合、HTTP リスナーのデフォルトの仮想サーバーが要求に対処します。

IP アドレスベースの仮想サーバーのときも、Sun ONE Application Server が指定された IP アドレスを検出できない場合、HTTP リスナーのデフォルトの仮想サーバーが要求に対処します。デフォルトの仮想サーバーを設定して、特定のドキュメントルートからエラーメッセージまたはサーバーページを送信させることができます。

注	HTTP リスナーのデフォルトの仮想サーバーと、サーバーのインストール時に作成されたデフォルトの仮想サーバーは別のものです。デフォルトの仮想サーバーは、デフォルトのアプリケーションサーバーインスタンス用の仮想サーバーです。HTTP リスナーのデフォルトの仮想サーバーは、ユーザーがデフォルトとして指定した任意の仮想サーバーです。

HTTP リスナーを作成したら、デフォルトの仮想サーバーを指定します。デフォルトの仮想サーバーはいつでも変更可能です。

obj.conf ファイル

デフォルト設定では、仮想サーバーごとに `obj.conf` ファイルが作成され、仮想サーバーの設定が保存されます。管理インタフェースまたはコマンド行インタフェースを使って設定を変更すると、この変更は仮想サーバーの `obj.conf` ファイルを含む設定ファイルに自動的に反映されます。すべての `obj.conf` ファイルは、*instance_dir*/config ディレクトリにあります。このマニュアルで示す「`obj.conf` ファイル」とは常に、すべての `obj.conf` ファイル、または説明中の仮想サーバーの `obj.conf` ファイルのことを指します。

プレフィックスを持たない obj.conf ファイルは、各仮想サーバーの obj.conf ファイルを作成するときに Sun ONE Application Server が使用するテンプレートです。このファイルを編集しても既存の仮想サーバーには影響しません。ただし、それ以後に作成する仮想サーバーには影響します。obj.conf ファイルを直接編集する方法については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

デフォルト設定では、アクティブな obj.conf ファイルには *virtual_server_name*-obj.conf という名前がつけられます。サーバーインスタンスのデフォルトの仮想サーバーにはインスタンスに基づいた名前がつけらるので、サーバーインスタンスの作成時に obj.conf ファイルには *instance_name*-obj.conf という名前がつけられます。これらのファイルの 1 つを直接、または管理インタフェースを使って編集すると、仮想サーバーの設定が変更されます。

要求を処理する仮想サーバーの選択

サーバーが要求を処理するには、HTTP リスナーから要求を受け取り、その要求を適切な仮想サーバーに転送する必要があります。ここでは、仮想サーバーの決定方法について説明します。

- HTTP リスナーがデフォルトの仮想サーバーのみに対して設定されている場合、この仮想サーバーが選択されます。
- HTTP リスナーが複数の仮想サーバーに対して設定されている場合、Host ヘッダーと仮想サーバーの hosts 属性が照合されます。Host ヘッダーが存在しない場合、または hosts 属性と一致しない場合は、その HTTP リスナーのデフォルトの仮想サーバーが選択されます。

仮想サーバーが SSL HTTP リスナーに対して設定されている場合、サーバーの起動時に、仮想サーバーの hosts 属性と証明書のサブジェクトパターンが照合されます。これらが一致しない場合は警告が生成され、サーバーログに書き込まれます。

仮想サーバーが決定すると、Sun ONE Application Server は仮想サーバーの obj.conf ファイルを実行します。obj.conf ファイルのどの指令を実行するかを決定する仕組みについては、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

ドキュメントルート

ドキュメントルート (一次ドキュメントディレクトリ) は、仮想サーバーの全ファイルを格納してリモートクライアントに提供するための中心的なディレクトリです。

ドキュメントルートを使用すると、仮想サーバー上のファイルへのアクセスを簡単に制限できます。また、URL に指定されたパスは一次ドキュメントディレクトリへの相対パスであるため、URL を変更せずに、簡単にドキュメントを新しいディレクトリ (別のディスク上の場合もある) に移動することができます。

たとえば、*install_dir*/docs というドキュメントディレクトリでは、http://www.sun.com/products/info.html などの要求によって *install_dir*/docs/info.html 内のファイルを検索します。ドキュメントルートを変更する (つまり、すべてのファイルおよびサブディレクトリを移動する) 場合も、仮想サーバーが使用するドキュメントルートを変更するだけなので、すべての URL を新しいディレクトリにマッピングする必要はありません。また、クライアントに新しいディレクトリ内を検索させる必要もありません。

Sun ONE Application Server のデフォルトのインスタンス (server1) のドキュメントルートが、server1 アプリケーションサーバーインスタンス内に作成された仮想サーバーのドキュメントルートになります。作成された各仮想サーバーのこのディレクトリは、オーバーライドすることができます。

仮想サーバーでの Sun ONE Application Server の機能の使用

Sun ONE Application Server は、SSL やアクセス制御など、仮想サーバーで使用できる機能を多数提供します。次の節では、これらの機能について説明します。また、詳しい情報の参照先も紹介します。

この節には次の項目があります。

- 仮想サーバーでの SSL の使用
- アクセスログファイルとサーバーログファイルの使用
- 仮想サーバーでのアクセス制御機能の使用
- 仮想サーバーでの CGI の使用

仮想サーバーでの SSL の使用

仮想サーバーで SSL を使用するときは、ほとんどの場合、IP アドレスベースの仮想サーバーを使用します。通常、ポートは 443 を使用します。Sun ONE Application Server は、要求の送信先 URL ホストを決定する前に、その要求を読み取る必要があるため、URL ホストベースの仮想サーバーで SSL を使用するのは困難です。サーバーが要求を読み取ると、セキュリティ情報をやりとりする最初のハンドシェークが発生したことになります。

唯一の例外は、URL ホストベースの全仮想サーバーが同一の SSL 設定を持っている場合です。たとえば、「ワイルドカード証明書」を使用して、同一のサーバー証明書を持っている場合です。詳細は、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

仮想サーバーに SSL を実装する方法として、仮想サーバーに 2 つの HTTP リスナーを設定する方法があります。一方の HTTP リスナーは、SSL を使ってポート 443 で待機する設定にします。もう一方は SSL を使用しない設定にします。通常、ユーザーは SSL を使用しない HTTP リスナーから仮想サーバーにアクセスします。トランザクションをセキュリティ保護する必要が生じた場合、ユーザーは、Web ページ上のボタンをクリックして、トランザクションのセキュリティ保護を開始します。その後、要求にはセキュリティ保護された HTTP リスナーが使用されます。

SSL トランザクションは、SSL を使用しないトランザクションよりもかなり時間がかかるため、必要な場合以外は使用されません。通常は、SSL を使用しないより高速な接続が使用されます。

Sun ONE Application Server と仮想サーバーのセキュリティの設定および使用方法については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。仮想サーバーに SSL を設定した例については、388 ページの「例 2: セキュリティの保護されたサーバー」を参照してください。

アクセスログファイルとサーバーログファイルの使用

アクセスログファイルには、仮想サーバーへの HTTP アクセスが記録されます。標準の設定では、新しい仮想サーバーを作成すると、アプリケーションサーバーインスタンスのログファイルがアクセスログファイルとして使用されます。多くの場合、仮想サーバーごとに専用のログファイルを用意する必要があります。このためには、各仮想サーバーのログのパスを変更します。仮想サーバーへのすべてのアクセスを同じアクセスログファイルに記録したい場合は、仮想サーバーのログ設定を変更して、ログファイルに仮想サーバーの ID を記述します。アプリケーションサーバーインスタンスのログ設定の変更については、第 5 章「ログの使用」を参照してください。

サーバーログファイルには、情報メッセージとエラーが記録されます。標準の設定では、新しい仮想サーバーを作成すると、アプリケーションサーバーインスタンスのログファイルがログファイルとして使用されます。各仮想サーバーのログファイルは変更が可能です。

仮想サーバーでのアクセス制御機能の使用

各仮想サーバーには、個別にアクセス制御機能を設定できます。LDAP データベースを使ってユーザーやグループを認証させるように、各仮想サーバーを設定することもできます。詳細は、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

仮想サーバーでの CGI の使用

仮想サーバーで CGI を使用できます。仮想サーバーごとに CGI を格納するディレクトリを設定し、CGI のファイルタイプを指定します。詳細は、『Sun ONE Application Server Web アプリケーション開発者ガイド』を参照してください。

HTTP リスナーの作成と設定

サーバーが要求を処理するには、HTTP リスナーから要求を受け取り、その要求を適切な仮想サーバーに転送する必要があります。サーバーのインストール時またはそれ以後にサーバーインスタンスを作成すると、`http-listener-1` という HTTP リスナーが自動的に作成されます。この HTTP リスナーは、IP アドレス 0.0.0.0 を使用します。また、アプリケーションサーバーのポートとして指定されたポートを使用します。デフォルトの HTTP リスナーは削除できません。

この節には次の項目があります。

- HTTP リスナーの作成
- HTTP リスナー設定の編集
- HTTP リスナーの削除

HTTP リスナーの作成

管理インタフェースを使って HTTP リスナーを作成するには、次の手順を実行します。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「HTTP リスナー」をクリックします。
3. 「新規」をクリックします。
4. フィールドに必要な情報を入力します。

   HTTP リスナーは、ポート番号と IP アドレスの一意な組み合わせになります。IPv4 アドレスまたは IPv6 アドレスを使用できます。IP アドレスベースの仮想サーバー用の HTTP リスナーを作成したい場合は、この HTTP リスナーに特定の IP アドレスを指定します。

   「戻すサーバー名」フィールドには、サーバーがクライアントに送信する URL に含まれるホスト名を指定します。これは、サーバーが自動的に生成する URL には影響するが、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、この名前はエイリアス名である必要があります。

   デフォルトの仮想サーバーは、その他の仮想サーバーが見つからない場合に HTTP リスナーの要求に応答する仮想サーバーです。詳細については、372 ページの「要求を処理する仮想サーバーの選択」を参照してください。

   HTTP リスナーに要求を受け付けさせるためには、この HTTP リスナーを有効にする必要があります。

さらに、この HTTP リスナー用に、セキュリティ機能を有効にしたり、詳細なプロパティを設定することもできます。IPV6 を指定するときは、「ファミリ」フィールドの inet6 の値を使います。この値が inet6 の場合、サーバーログでは、IPv4 アドレスに ::ffff: プレフィックスが付けられます。

5. 「了解」をクリックします。

HTTP リスナーを作成するとき、デフォルトの仮想サーバーのフィールドには、必ず既存の仮想サーバーを入力します。最初はサーバーインスタンスの作成時に作成された仮想サーバーを使用します。必要に応じて、後から新しい仮想サーバーに切り替えることができます。

コマンド行インタフェースを使って HTTP リスナーを作成する場合は、`asadmin` ユーティリティの `create-http-listener` コマンドを使用します。作成した HTTP リスナーを一覧表示するには、`list-http-listeners` コマンドを使用します。

HTTP リスナーを作成するコマンドの構文は次のとおりです。

`asadmin create-http-listener --user` *username* `[--password` *password*`]` `[--host` *hostname*`]` `[--port` *adminport*`]` `[--secure | -s]` `[--passwordfile` *file_name*`]` `--address` *address* `[--instance` *instancename*`]` `--listenerport` *listener_port* `--defaultvs` *virtual_server* `--servername` *server_name* `[--family` *family*`]` `[--acceptorthreads` *acceptor_threads*`]` `[--blockingenabled` *blocking_enabled*`]` `[--securityenabled` *security_enabled*`]` `[--enabled` *enabled*`]` *listener_id*

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

HTTP リスナー設定の編集

管理インタフェースを使って HTTP リスナーの設定を編集するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「HTTP リスナー」を開きます。
3. 編集する HTTP リスナーをクリックします。
4. 設定内容に必要な変更を加え、「保存」をクリックします。

詳細については、オンラインヘルプを参照してください。

コマンド行インタフェースで `asadmin` ユーティリティを使って HTTP リスナーを編集することもできます。現在の設定を取得するときは `get` コマンド、新しい値を設定するときは `set` コマンドを使用します。

HTTP リスナーのすべての属性の値を取得するには、次のコマンドを実行します。

```
asadmin> get server_instance.http-listener.http_listener_name.*
```

たとえば、デフォルト HTTP リスナーの値を取得するときは、次のコマンドを実行します。

```
asadmin> get server1.http-listener.http-listener-1.*
```

属性に値を設定するには、次のコマンドを実行します。

```
asadmin> set server_instance.http-listener.http_listener_name.attribute_name=value
```

たとえば、http-listener-1 の属性 defaultVirtualServer に server2 という値を設定するときは、次のコマンドを実行します。

```
asadmin> set
server1.http-listener.http-listener-1.defaultVirtualServer=server2
```

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。asadmin の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

HTTP リスナーの削除

管理インタフェースを使って HTTP リスナーを削除するには、次の手順を実行します。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「HTTP リスナー」をクリックします。
3. 削除する HTTP リスナーの横のチェックボックスをクリックします。
4. 「削除」をクリックします。

コマンド行インタフェースを使って HTTP リスナーを削除する場合は、asadmin ユーティリティの delete-http-listener コマンドを使用します。構文は次のとおりです。

```
asadmin delete-http-listener ---user username [--password password]
[--host hostname] [--port adminport] [--secure | -s] [--passwordfile
file_name] --instance instance httplistener_id
```

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。asadmin の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

仮想サーバーの作成と設定

HTTP リスナーの設定が完了すると、仮想サーバーを作成して使用できるようになります。

この節には次の項目があります。

- 仮想サーバーの作成
- 仮想サーバーの設定の編集
- 仮想サーバーの削除

仮想サーバーの作成

管理インタフェースを使って仮想サーバーを作成するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」をクリックします。
3. 「新規」をクリックします。
4. 必須フィールドとオプションフィールドに必要な情報を入力し、
5. 「保存」をクリックします。

コマンド行インタフェースを使って仮想サーバーを作成する場合は、`asadmin` ユーティリティの `create-virtual-server` コマンドを使用します。構文は次のとおりです。

`asadmin create-virtual-server --user` *username* `---user` *username* `[--password` *password*`] [--host` *hostname*`] [--port` *adminport*`] [--secure | -s] [--passwordfile` *file_name*`] [--instance` *instancename*`] --hosts` *hosts* `--mime` *mime_types_file* `[--httplisteners` *http-listeners*`] [--defaultwebmodule` *default_web_module*`] [--configfile` *config_file*`] [--defaultobj` *default_object*`] [--state` *state*`] [--acls` *acls*`] [--acceptlang` *accept_language*`] [--logfile` *logfile*`] [--property (`*name=value*`)[:`*name=value*`]*]` *virtual_server_id*

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

仮想サーバーを作成すると、次の設定情報を入力できるようになります。

- 必須設定
- オプションの一般設定
- Web アプリケーションの設定

- CGI の設定
- HTTP のサービス品質の設定

必須設定

仮想サーバーの必須設定には、名前 (ID) と URL ホストの設定が含まれます。

また、MIME タイプのファイルも指定する必要があります。MIME タイプファイルには、ファイル拡張子とファイルタイプのマッピング情報が含まれています。たとえば、.cgi という拡張子を持つファイルを CGI ファイルとして処理するように指定できます。

仮想サーバーごとに個別の MIME タイプファイルを作成する必要はありません。必要な数の MIME タイプファイルを作成し、仮想サーバーに関連付けます。デフォルトの MIME タイプファイルは `mime1`、そのファイル名は `mime.types` になります。

MIME タイプファイルの詳細については、364 ページの「MIME タイプの設定」を参照してください。

オプションの一般設定

必須フィールドのほかに、オプションフィールドも設定できます。

HTTP リスナー

HTTP リスナーは、仮想サーバーへの接続を処理します。リモートクライアントが仮想サーバーにアクセスできるようにするには、これを指定する必要があります。

ACL

仮想サーバーに適用される ACL (アクセス制御リスト) です。詳細は、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

使用可能な言語

クライアントが HTTP 1.1 を使ってサーバーにアクセスするときに、そのクライアントが使用可能な言語に関する情報が渡されることがあります。この言語情報を解析するように、サーバーを設定できます。

たとえば、日本語と英語でドキュメントを保存している場合に、使用可能な言語のヘッダー情報を解析するように設定できます。言語ヘッダーに日本語の使用が設定されているクライアントがサーバーにアクセスすると、そのクライアントは日本語バージョンのページを受信します。言語ヘッダーに英語の使用が設定されているクライアントがサーバーにアクセスすると、そのクライアントは英語バージョンのページを受信します。

複数の言語をサポートしない場合は、使用可能な言語のヘッダー情報を解析するべきではありません。

状態

ここで設定した状態は、仮想サーバーの状態です。仮想サーバーの状態は、アプリケーションサーバーインスタンスが有効であるかどうかに左右されません。このページで仮想サーバーの状態が「オン (On)」になっている場合、仮想サーバーは、アプリケーションサーバーインスタンスが「オン (On)」である場合に限り、要求を受け付けることができます。

これは、デフォルトのアプリケーションサーバーインスタンスのデフォルトの仮想サーバーの場合も同様です。アプリケーションサーバーインスタンスを無効にしても、デフォルトの仮想サーバーは「オン (On)」のままです。ただし、接続は受け付けられません。

有効な状態は、「オン (On)」、「オフ (Off)」、または「無効 (Disable)」です。「オン (On)」に設定した場合、仮想サーバーは接続を受け入れることができます。

アプリケーションサーバーインスタンスのデフォルトの仮想サーバーを無効にすることはできません。

ログファイル

ログファイル (サーバーログファイル) には、情報メッセージとエラーが記録されます。アクセスログファイルには、仮想サーバーへの HTTP アクセスが記録されます。

ドキュメントルート

ドキュメントルート (一次ドキュメントディレクトリ) は、仮想サーバーの全ファイルを格納してリモートクライアントに提供するための中心的なディレクトリです。詳細については、373 ページの「ドキュメントルート」を参照してください。

Web アプリケーションの設定

Web アプリケーションは、サーブレット、JavaServer Pages、HTML ドキュメント、および、イメージファイルや圧縮アーカイブなどのデータを含むその他の Web リソースの集まりです。Web アプリケーションは、アーカイブ (WAR ファイル) にパッケージされている場合や、オープンディレクトリ構造に配備されている場合があります。

Sun ONE Application Server 7 は Servlet 2.3 API 仕様をサポートしています。この仕様では、サーブレットや JSP を Web アプリケーションに組み込むことができます。また、Sun ONE Application Server 7 は、J2EE アプリケーションコンポーネントではない SHTML と CGI もサポートしています。

仮想サーバーを作成するときに、仮想サーバーのデフォルトの Web モジュールを指定します。このデフォルトの Web モジュールは、仮想サーバー上に配備されたその他の Web モジュールが解決できないすべての要求に応答します。デフォルトの Web モジュールを指定しない場合は、コンテキストルートが空の Web モジュールが使われます。コンテキストルートが空の Web モジュールが存在しない場合は、システムのデフォルトの Web モジュールが作成され、これが使用されます。

Web アプリケーションを配備するときに、仮想サーバーを指定します。Web アプリケーションを配備すると、利用可能な Web モジュールのリストにこのアプリケーションが表示され、仮想サーバーのデフォルトの Web モジュールとして選択できるようになります。Web モジュールを仮想サーバーのデフォルト Web モジュールとして指定すると、その Web アプリケーションの仮想サーバーのリストにこの仮想サーバーが追加されます。

CGI の設定

仮想サーバーの作成時に設定した CGI 設定は、CGI がどのユーザーまたはグループとして実行されるか、CGI の実行前にどのディレクトリに変更するか (chroot)、および chroot の後にどのディレクトリに変更するかに影響します。

UNIX 環境では、nice も設定できます。nice は、サーバーを基準として CGI プログラムの優先度を決定する増分値です。通常、サーバーは nice 値 0 で動作しており、CGI の nice 増分値は 0 から 19 の範囲で指定します。0 を指定すると CGI プログラムはサーバーと同じ優先度で動作し、19 を指定すると CGI プログラムはサーバーよりもかなり低い優先度で動作します。

HTTP のサービス品質の設定

サービス品質とは、ユーザーが仮想サーバーに設定するパフォーマンス制限のことです。たとえば、ISP は、許可する帯域幅に応じて仮想サーバーの課金額を変えたいことがあります。これらの設定は、強制することも (つまり、特定の帯域幅に許可される最大接続数を指定)、強制しないこともできます。強制しない設定では、制限を超えるとログファイルにメッセージが記録されます。詳細については、151 ページの「CLI によるトランザクションサービスの管理」を参照してください。

管理インタフェースを使ってこれらの設定を変更できるだけでなく、`asadmin` ユーティリティを使ってコマンド行ユーティリティから変更することもできます。コマンド行インタフェースの `asadmin` ユーティリティを使ってサービス品質を設定するときは、次のコマンドを使います。

- `create-http-qos`
- `delete-http-qos`

これらのコマンドの構文は次のとおりです。

`asadmin create-http-qos --user` *username* [`--password` *password*] [`--host` *hostname*] [`--port` *adminport*] [`--secure | -s`] [`--passwordfile` *file_name*] [`--virtualserver` *virtual_server_id*] [`--bwlimit` *bandwidth_limit*] [`--enforcebwlimit` *enforce_bandwidth_limit*] [`--connlimit` *connection_limit*] [`--enforceconnlimit` *enforce_connection_limit*] *instance_name*

`asadmin delete-http-qos --user` *username* [`--password` *password*] [`--host` *hostname*] [`--port` *adminport*] [`--secure | -s`] [`--passwordfile` *file_name*] [`--virtualserver` *virtual_server_id*] *instance_name*

仮想サーバーを指定してこれらのコマンドを実行すると、その仮想サーバーのサービス品質情報が作成または削除されます。仮想サーバーを指定しない場合は、コマンドはサーバーインスタンスに適用されます。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

仮想サーバーの設定の編集

仮想サーバーのセットアップ内容を編集できます。仮想サーバーの設定の編集方法については、次の項目を参照してください。

- 管理インタフェースによる一般設定の編集
- コマンド行インタフェースによる一般設定の編集
- CGI 設定の編集
- ドキュメント処理の設定、ドキュメントディレクトリの設定、および HTTP/HTML 設定の編集

管理インタフェースによる一般設定の編集

仮想サーバーの基本設定は、仮想サーバーの作成時に行います。変更を加えるには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーをクリックします。
4. 必要な変更を加えます。

   サービス品質の設定、ACL の追加、コンテンツ関連の設定 (ドキュメントルートの設定、言語ヘッダーの受け入れなど)、CGI 関連の設定 (user、group、nice、chroot の設定など)、およびデフォルトの Web モジュールなどの変更が可能です。

5. 「保存」をクリックします。

これらの設定の詳細については、379 ページの「仮想サーバーの作成と設定」を参照してください。オンラインヘルプも参考になります。

コマンド行インタフェースによる一般設定の編集

コマンド行インタフェースで asadmin ユーティリティを使って これらの設定を編集することもできます。現在の設定を取得するときは get コマンド、新しい値を設定するときは set コマンドを使用します。

仮想サーバーのすべての属性を取得するには、次の構文を使います。

```
asadmin> get instance_name.virtual-server.vserver_id.*
```

次に例を示します。

```
asadmin> get server1.virtual-server.vs1.*
```

アプリケーションサーバーインスタンス server1 のすべての属性を取得するときは、次の構文を使います。

```
asadmin> get server1.virtual-server.server1.*
```

たとえば、受け入れる言語ヘッダーの属性に値を設定するときは、次の構文を使います。

```
asadmin> set
server1.virtual-server.server1.virtualserver.acceptLanguage=false
```

注 「一般」ページのすべてのフィールドの値は、コマンド行インタフェースを使って設定できます。ただし、「CGI」タブのように、このページのその他のタブのフィールド値をコマンド行インタフェースから設定することはできません。

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。asadmin の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

CGI 設定の編集

CGI 設定の編集については、『Sun ONE Application Server Web アプリケーション開発者ガイド』を参照してください。

ドキュメント処理の設定、ドキュメントディレクトリの設定、および HTTP/HTML 設定の編集

これらの設定の変更については、第 16 章「仮想サーバーコンテンツの管理」を参照してください。

仮想サーバーの削除

仮想サーバーを削除するには、次の手順に従います。

1. 管理インタフェースの左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」をクリックします。
3. 削除する仮想サーバーの横のチェックボックスをクリックします。
4. 「削除」をクリックします。

   管理インタフェースを使ってすべての仮想サーバーを削除することはできません。

コマンド行インタフェースを使って仮想サーバーを削除する場合は、`asadmin` ユーティリティの `delete-virtual-server` コマンドを使用します。

構文は次のとおりです。

`asadmin delete-virtual-server --user` *username* `[--password` *password*`] [--host` *hostname*`] [--port` *adminport*`] [--secure | -s] [--passwordfile` *file_name*`] --instance` *instance virtualserver_id*

コマンド構文の詳細については、コマンド行インタフェースのヘルプを参照してください。`asadmin` の使い方の詳細については、付録 A「コマンド行インタフェースの使用」を参照してください。

仮想サーバーの配備

Sun ONE Application Server の仮想サーバーのアーキテクチャはとても柔軟です。アプリケーションサーバーインスタンスには、安全である、または安全でない HTTP リスナーをいくつでも持たせることができます。また、これらの HTTP リスナーには、仮想サーバーをいくつでも関連づけることができます。さらに、IP アドレスベースの仮想サーバーと URL ホストベースの仮想サーバーの両方を利用できます。

各仮想サーバーには、専用の ACL、専用の `mime.types` ファイル、専用の Java Web アプリケーションを持たせることができます (必須ではありません)。

この設計により、さまざまな種類のアプリケーションに合わせてサーバーを柔軟に設定できます。次の例は、Sun ONE Application Server で利用できる設定の一部を示しています。

- 例 1: デフォルト設定
- 例 2: セキュリティの保護されたサーバー
- 例 3: イントラネットのホスティング
- 例 4: マスホスティング

例 1: デフォルト設定

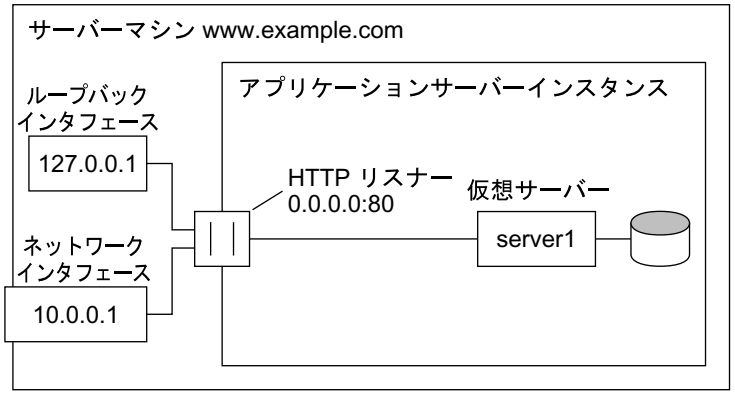
デフォルト設定では、アプリケーションサーバーの数は 1 つです。このアプリケーションサーバーインスタンスには、マシンが設定されている IP アドレスのポート 80、1024、または指定したポートで待機する 1 つの HTTP リスナーがあります。

ローカルネットワーク側のメカニズムによって、マシンが設定されている各アドレスと名前のマッピングが行われます。次の例では、アドレス 127.0.0.1 のループバックインタフェース (ネットワークカードなしでも存在できるインタフェース)、およびアドレス 10.0.0.1 のイーサネットインタフェースという 2 つのインタフェースがあります。

example.com という名前は DNS を経由して 10.0.0.1 にマップされます。HTTP リスナーは、マシンに設定されているすべてのアドレスについて、ポート 80 で待機するように設定されます (0.0.0:80)。

デフォルトの設定では IP アドレスベースの仮想サーバーは使われないため、デフォルトの HTTP リスナーだけが使われます。すべての接続は、仮想サーバー server1 を経由します。

デフォルト設定



DNS

www.example.com	10.0.0.1

この設定では、次の各要素への接続は仮想サーバー VS1 によって処理されます。

- http://127.0.0.1/ (example.com から呼び出し)
- http://localhost/ (example.com から呼び出し)
- http://example.com/
- http://10.0.0.1/

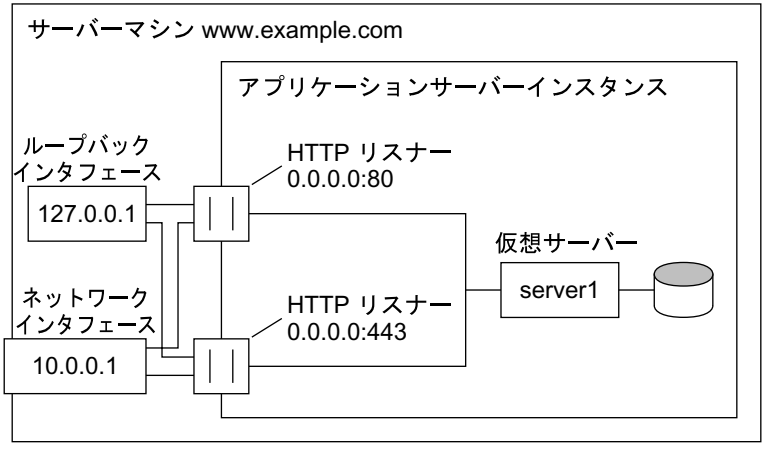
旧来の HTTP サーバーを利用する場合は、この設定を利用します。仮想サーバーや HTTP リスナーを追加する必要はありません。サーバーの設定を変更するときは、server1 の設定を変更します。

例 2: セキュリティの保護されたサーバー

デフォルトの設定で SSL を使うには、HTTP リスナーをセキュリティの保護されたモードに変更するだけです。

また、0.0.0.0:443 に安全な HTTP リスナーを追加し、server1 を新しい HTTP リスナーに関連づけます。これにより、仮想サーバーにはセキュリティの保護された HTTP リスナーと、セキュリティ保護されていない HTTP リスナーが用意されます。サーバーは、同じコンテンツを SSL を使って、または使わずに提供できるようになります。つまり、http://example.com/ と https://example.com/ は同じコンテンツを配信します。

安全なサーバー



DNS

www.example.com	10.0.0.1

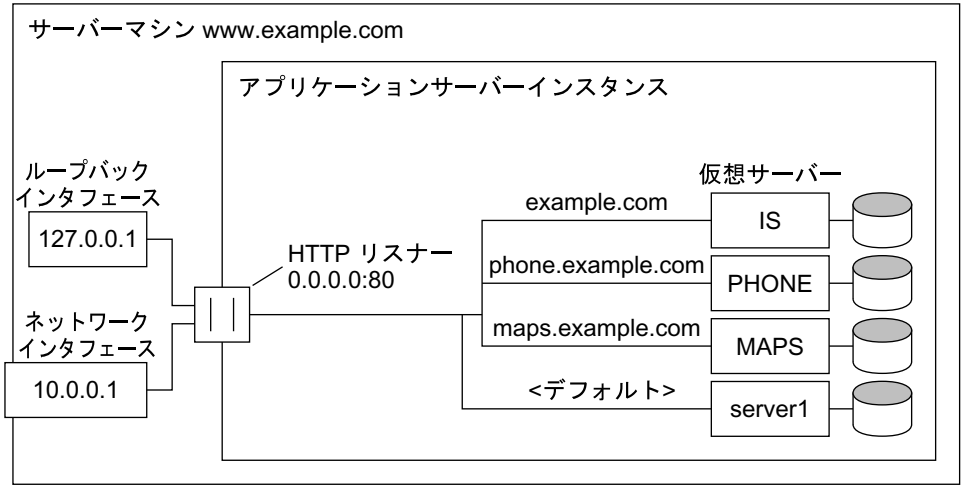
HTTP リスナーには SSL パラメータが設定されています。

例 3: イントラネットのホスティング

Sun ONE Application Server のより複雑な設定には、イントラネットに配備するために複数の仮想サーバーをサーバーがホスティングする設定があります。たとえば、従業員が別の従業員の電話番号を検索するサイト、社内の地図を参照するサイト、情報サービス部に送った要求の状態を追跡するサイトという 3 つの社内サイトがあると仮定します。この例では、従来は phone.example.com、maps.example.com、is.example.com という名前がマップされた 3 つの異なるマシンがそれぞれのサイトをホスティングしていました。

ハードウェアと管理のオーバーヘッドを削減するために、machine example.com というマシンで稼働する 1 つのアプリケーションサーバーにすべてのサイトを統合します。これは、URL ホストベースまたは IP アドレスベースの仮想サーバーを使って設定できます。どちらを利用した場合にも、利点と欠点があります。

URL ホストベースの仮想サーバーを使ったイントラネットのホスティング



DNS

www.example.com	10.0.0.1
is.example.com	10.0.0.1
phone.example.com	10.0.0.1
maps.example.com	10.0.0.1

URL ホストベースの仮想サーバーは設定が容易ですが、次のような欠点があります。

- この設定で SSL をサポートするには、ワイルドカード証明書を使った標準以外の設定が必要となる。詳細は、『Sun ONE Application Server セキュリティ管理者ガイド』を参照
- URL ホストベースの仮想サーバーは、従来の HTTP クライアントでは機能しない

IP アドレスベースの仮想サーバーには、次のような利点があります。

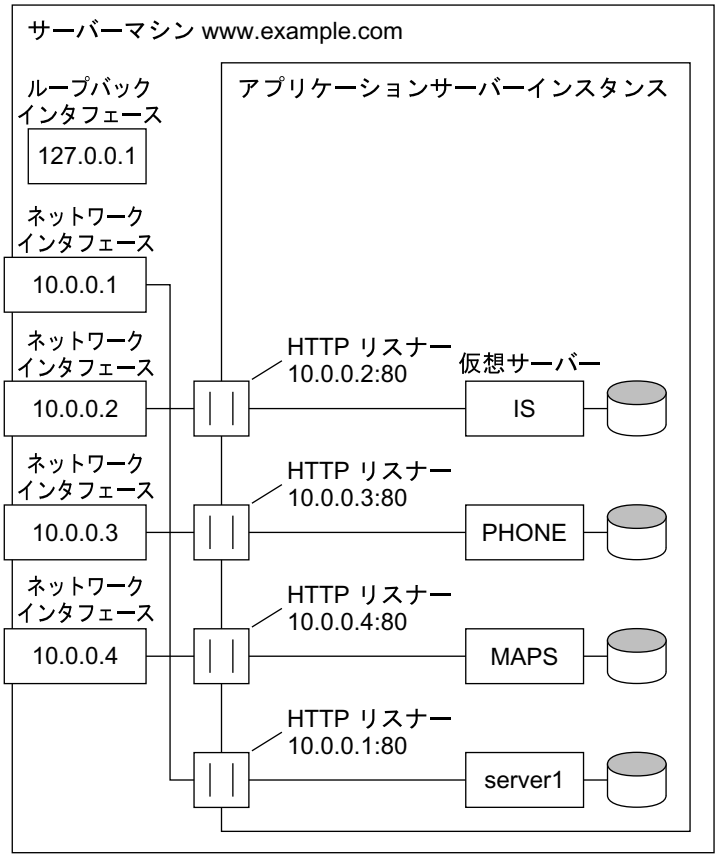
- HTTP/1.1 Host ヘッダーをサポートしていない従来のクライアントでも機能する
- 簡単に SSL をサポートできる

欠点は次のとおりです。

- ホストコンピュータの設定 (現実のまたは仮想のネットワークインタフェースの設定) を変更する必要がある
- 数千の仮想サーバーを使った設定に対応するだけのスケーラビリティがない

どちらの設定でも、3 つの名前をアドレスにマップする設定が必要です。IP アドレスベースの設定では、それぞれの名前は異なるアドレスにマップされます。ホストマシンでは、すべてのアドレスとの接続を受信するように設定する必要があります。URL ホストベースの設定では、ホストマシンに固有の 1 つのアドレスにすべての名前をマップできます。

IP アドレスベースの仮想サーバーを使ったイントラネットのホスティング



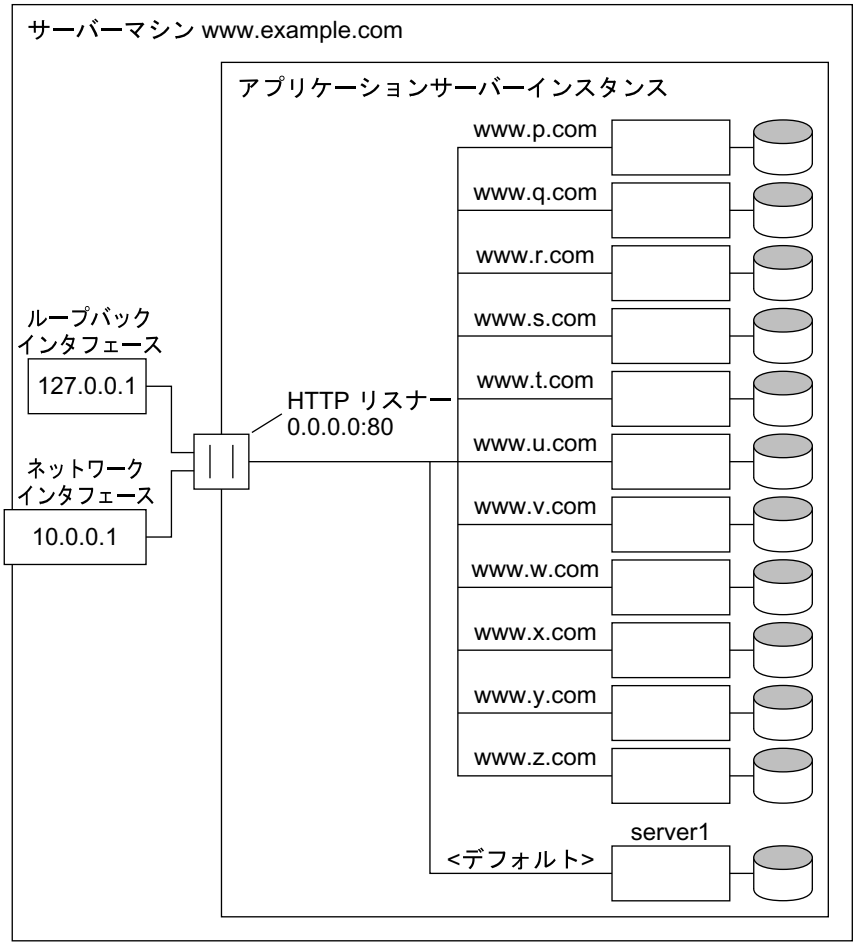
DNS

www.example.com	10.0.0.1
is.example.com	10.0.0.2
phone.example.com	10.0.0.3
maps.example.com	10.0.0.4

例 4: マスホスティング

マスホスティングは、多数の低トラフィック仮想サーバーが有効な設定です。たとえば、低トラフィックの多数の個人ホームページをホスティングする ISP などがこれに該当します。この仮想サーバーは、通常は URL ホストベースです。

マスホスティング



DNS

www.example.com	10.0.0.1
www.p.com	10.0.0.1
www.q.com	10.0.0.1
www.r.com	10.0.0.1
. . .	
www.z.com	10.0.0.1

デフォルトの仮想サーバー `server1` は依然として存在します。

第 16 章

仮想サーバーコンテンツの管理

この章では、仮想サーバーが処理するファイルの設定方法と管理方法について説明します。

この章では次のトピックについて説明します。

- ドキュメントルートの変更
- 追加ドキュメントディレクトリの設定
- リモートファイル操作の有効化
- htaccess の使用
- シンボリックリンクの制限 (UNIX)
- ユーザーの公開情報ディレクトリのカスタマイズ (UNIX)
- ドキュメントの環境設定
- エラー応答のカスタマイズ
- 国際文字セットの変更
- ドキュメントフッターの設定
- URL 転送の設定
- サーバーで解析される HTML の設定
- キャッシュ制御指令の設定
- より強力な暗号化方式の使用

ドキュメントルートの変更

ドキュメントルートは中央のディレクトリで、リモートクライアントから利用できるファイルのすべてを格納します。

仮想サーバーを追加するときには、絶対パスでドキュメントルートを指定します。ドキュメントルートとその使用方法の詳細については、373 ページの「ドキュメントルート」を参照してください。

管理インタフェースで、ドキュメントルートを別のパスに変更するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「一般」タブをクリックします。
5. 「ドキュメントルート」フィールドにディレクトリの絶対パスを入力します。

   このディレクトリを手動で作成する必要があります。
6. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

注	通常、個々の仮想サーバーには独自のドキュメントルートが設定されます。

追加ドキュメントディレクトリの設定

ほとんどの場合、仮想サーバーインスタンスおよびサーバーインスタンスのドキュメントは、ドキュメントルートに格納されます。ただし、ドキュメントルートの外部のディレクトリからドキュメントを配信することもできます。そのためには、追加ドキュメントディレクトリを設定します。ドキュメントルートの外部のディレクトリから配信することにより、プライマリドキュメントルートへのアクセスを許可せずに、グループ化したドキュメントの管理を、他者に任せることができます。

管理インタフェースで、追加ドキュメントディレクトリを設定するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。

3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメントディレクトリ」タブをクリックします。
5. 「追加ドキュメントディレクトリ」をクリックします。
6. マッピングする URL プレフィックスを選択します。

   クライアントは、ドキュメントの要求時に、この URL をサーバーへ送ります。
7. これらの URL のマッピング先となるディレクトリを指定します。
8. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

ユーザーによるディレクトリへの書き込みを不可能にするために、追加ドキュメントディレクトリへのアクセスに制限を設定する必要があります。

リモートファイル操作の有効化

リモートファイル操作を有効にすると、サーバーに対するファイルのアップロード、ファイルの削除、ディレクトリの作成、ディレクトリの削除、ディレクトリ内容の一覧表示、およびファイル名の変更をクライアントが実行できるようになります。仮想サーバーの設定ファイル `obj.conf` には、リモートファイル操作を有効にした場合にアクティブになるコマンドが含まれています。これらのコマンドをアクティブにすると、リモートのブラウザからサーバー上のドキュメントを変更できます。アクセス制御を使ってこれらのリソースへの書き込みアクセスを制限することによって、認証されていない変更を阻止する必要があります。

リモートファイル操作を有効にしても、Microsoft Frontpage などのコンテンツ管理システムの使用に影響がないことが必要となります。

UNIX 環境では : この設定を行う管理者がファイルに対する適切なアクセス権を持たない場合、この操作は機能しません。つまり、ドキュメントルートユーザーはサーバーユーザーと同等であることが必要です。

管理インタフェースで、リモートファイル操作を有効にするには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメントディレクトリ」タブをクリックします。
5. 「リモートファイル操作」をクリックします。

6. 仮想サーバー全体に変更を適用するために、リソースの選択肢から「サーバー全体」を選びます。あるいは、仮想サーバー内の特定のディレクトリに移動します。
7. アクティブにするリモートファイル操作を選択します。
8. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

htaccess の使用

htaccess ファイルは、設定オプションのサブセットを格納した動的な設定ファイルです。Sun ONE Application Server の標準のアクセス制御と htaccess ファイルを組み合わせて使用できます。標準のアクセス制御は、常に htaccess によるアクセス制御の前に適用されます。

htaccess の使用方法については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

シンボリックリンクの制限 (UNIX)

サーバーでのファイルシステムリンクの使用を制限することができます。ファイルシステムリンクは、別のディレクトリやファイルシステムに格納されているファイルへの参照です。参照によって、現在のディレクトリにあるファイルと同様に、リモートファイルへのアクセスが可能となります。ファイルシステムリンクには次の 2 種類があります。

- ハードリンク : 同じデータブロックの一式を指す 2 つのファイル名。元のファイルとリンクは同じ。このため、異なるファイルシステム間ではハードリンクを使用できない
- シンボリック (ソフト) リンク : データを含む元のファイルと、元のファイルを指すファイルで構成される。シンボリックリンクは、ハードリンクよりも柔軟性がある。異なるファイルシステム間でも使用でき、ディレクトリへもリンクできる

ハードリンクとシンボリックリンクの詳細については、使用している UNIX システムのマニュアルを参照してください。

ファイルシステムリンクを使用すると、プライマリディレクトリの外部に格納されているドキュメントへのポインタを誰でも簡単に作成できます。このため、重要なファイル (機密文書、システムのパスワードファイルなど) へのポインタを作成されないように気を付ける必要があります。

管理インタフェースで、シンボリックリンクを制限するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメントディレクトリ」タブをクリックします。
5. 「シンボリックリンク」をクリックします。
6. 仮想サーバー全体に変更を適用するために、リソースの選択肢から「サーバー全体」を選びます。あるいは、仮想サーバー内の特定のディレクトリに移動します。
7. ソフトリンクまたはハードリンク、あるいはその両方を有効にするかどうか、および元のディレクトリを選択します。
8. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

ユーザーの公開情報ディレクトリのカスタマイズ (UNIX)

ユーザーが独自の Web ページを運用したい場合もあります。公開情報ディレクトリを設定することにより、サーバー上のすべてのユーザーがホームページなどのドキュメントを自由に作成できるようになります。

注 Windows システムでは管理インタフェースで「ユーザードキュメントディレクトリ」ページが表示されますが、この機能は使用できません。

このシステムによって、サーバーが公開情報ディレクトリとして認識する URL を使って、クライアントはサーバーにアクセスできます。たとえば、プレフィックス ~ とディレクトリ `public_html` を選択するとします。
`http://www.sun.com/~jdoe/aboutjane.html` という要求が到着すると、サーバーは `~jdoe` をユーザーの公開情報ディレクトリと認識します。サーバーは、システムのユーザーデータベースで `jdoe` を検索し、Jane のホームディレクトリを見つけ出します。その結果、`~/jdoe/public_html/aboutjane.html` が参照されます。

この節には次の項目があります。

- 公開情報ディレクトリの設定
- コンテンツ公開の制限
- 起動時のパスワードファイル全体の読み込み

公開情報ディレクトリの設定

管理インタフェースで、公開ディレクトリを使うために仮想サーバーを設定するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメント処理」タブをクリックします。
5. 「ユーザードキュメントディレクトリ」をクリックします。
6. ユーザー URL プレフィックスを選択します。

   通常、プレフィックスは ~ です。UNIX では、ティルダが、ユーザーのホームディレクトリへアクセスする場合の標準プレフィックスであるためです。
7. ユーザーのホームディレクトリで、サーバーが HTML ファイルを検索するサブディレクトリを選択します。

   通常は `public_html` です。
8. パスワードファイルを指定します。

   サーバー上のユーザーリストが格納されるファイルを検索する場所を、サーバーに認識させる必要があります。サーバーはこのファイルを使って、有効なユーザー名を確認し、ユーザーのホームディレクトリを見つけます。このとき、システムのパスワードファイルを指定すると、サーバーはユーザーの検索に標準のライブラリ呼び出しを使います。一方、別のユーザーファイルを作成して、ユーザーの検索に使うこともできます。絶対パスを使って、ユーザーファイルを指定できます。

   ファイルの各行を、次の構造にする必要があります。`/etc/passwd` ファイル内の不要な要素は、`*` で表されています。

   ```
   username:*:*:groupid:*:homedir:*
   ```
9. 起動時にパスワードデータベースを読み込むかどうかを選択します。

   詳細については、399 ページの「起動時のパスワードファイル全体の読み込み」を参照してください。
10. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

すべてのユーザーが変更可能な中央ディレクトリへの URL マッピングを作成して、ユーザーに個別のディレクトリを与える方法もあります。

コンテンツ公開の制限

システム管理者としては、ユーザードキュメントディレクトリを介してコンテンツを公開できるユーザーアカウントに、制限を加えたい場合もあります。ユーザーによる公開を制限するには、/etc/passwd ファイルでユーザーのホームディレクトリの後ろにスラッシュを追加します。

```
jdoe::1234:1234:John Doe:/home/jdoe:/bin/sh
```

上記を次のように変更します。

```
jdoe::1234:1234:John Doe:/home/jdoe/:/bin/sh
```

このように変更すると、Sun ONE Application Server はこのユーザーのディレクトリからページを配信しません。この URI を要求したブラウザは、"404 File Not Found" エラーを受け取ります。また、404 エラーはアクセスログに記録されます。

その後、このユーザーにコンテンツの公開を許可する場合は、/etc/passwd エントリから後続のスラッシュを削除し、アプリケーションサーバーインスタンスを再起動します。

起動時のパスワードファイル全体の読み込み

起動時に、パスワードファイル全体を読み込むオプションを設定することもできます。このオプションを選択すると、サーバーは起動時にパスワードファイルをメモリに読み込むため、ユーザーの検索が速くなります。ただし、パスワードファイルが大きい場合は、メモリが大量に消費されます。

ドキュメントの環境設定

この節には次の項目があります。

- インデックスファイル名の入力
- ディレクトリの索引化の選択
- サーバーホームページの指定
- デフォルト MIME タイプの指定

管理インタフェースで、ドキュメントの環境設定を行うには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメント処理」タブをクリックします。
5. 「ドキュメントプリファレンス」をクリックします。
6. 次の各節で説明する適切なフィールド値を選択します。
7. 「了解」をクリックします。

設定できる環境設定については、後続の各節で説明します。詳細については、オンラインヘルプを参照してください。

インデックスファイル名の入力

ドキュメント名が URL で指定されていない場合、サーバーはインデックスファイルを自動的に表示します。デフォルトのインデックスファイルは、`index.html` および `home.html` です。複数のインデックスファイルが指定されている場合、サーバーはこのフィールドに表示されている順序でファイルを検索します。たとえば、インデックスファイル名が `index.html` と `home.html` の場合、サーバーは先に `index.html` を検索し、見つからないと次に `home.html` を検索します。

ディレクトリの索引化の選択

通常、ドキュメントディレクトリには複数のサブディレクトリを作成します。たとえば、`products`、`people` などです。クライアントがこれらのディレクトリの概要 (索引) にアクセスできると便利です。

サーバーは、各ディレクトリで index.html または home.html と呼ばれるインデックスファイルを検索して、ディレクトリの索引化を行います。これらのファイルは、管理者がディレクトリのコンテンツの概要として作成および管理するファイルです。詳細については、400 ページの「インデックスファイル名の入力」を参照してください。ファイル名をデフォルトの名前にすることによって、任意のファイルをディレクトリのインデックスファイルとして指定できます。したがって、CGI プログラムをインデックスとすることも可能です。

インデックスファイルが見つからない場合、サーバーはドキュメントルート内のすべてのファイルを一覧表示するインデックスファイルを生成します。

警告 サーバーをファイアウォールの外側に設置している場合は、ディレクトリの索引化を無効にして、ディレクトリ構造とファイル名にアクセスできないようにする必要があります。

サーバーホームページの指定

通常、エンドユーザーがサーバーにアクセスすると、ホームページと呼ばれるファイルが最初に表示されます。このファイルには、サーバーに関する情報や、その他のドキュメントへのリンクを設定します。

デフォルトでは、サーバーは「ドキュメントプリファレンス」ページの「インデックスファイル名」フィールドで指定されたインデックスファイルを見つけ出し、このファイルをホームページとして使います。ただし、ホームページとして使うファイルを別に指定することもできます。

デフォルト MIME タイプの指定

サーバーは、ドキュメントをクライアントに送信するときに、ドキュメントタイプを示すセクションを挿入するため、クライアントはドキュメントを適切に表現できます。ただし、ドキュメントの拡張子がサーバーで定義されていないと、サーバーは適切なドキュメントタイプを判別できません。このような場合は、デフォルトのタイプが送信されます。

通常、デフォルトは text/plain ですが、サーバーに格納されているファイルのうち最も多いタイプを設定することをお勧めします。一般的な MIME タイプには、次のようなものがあります。

- text/plain
- text/richtext
- image/jpeg
- application/x-tar
- application/x-gzip
- text/html
- image/tiff
- image/gif
- application/postscript
- audio/basic

エラー応答のカスタマイズ

カスタムエラー応答を指定することにより、仮想サーバーでエラーが発生したときにクライアントへ詳しいメッセージを送ることができます。送信するファイルまたは実行する CGI プログラムを指定できます。

たとえば、特定のディレクトリに関するエラーが発生した場合、サーバーの動作を変更することができます。サーバー内のアクセス制御で保護された部分にクライアントが接続を試みた場合に、アカウントの取得方法を説明するエラーファイルを返すこともできます。

カスタムエラー応答を有効にする前に、エラーに応答して送信される HTML ファイルまたは実行される CGI プログラムを作成しておく必要があります。作成後、管理インタフェースでエラー応答を有効にします。

管理インタフェースで、カスタマイズしたエラー応答を有効にするには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメント処理」タブをクリックします。
5. 「エラー応答」をクリックします。
6. 仮想サーバー全体に変更を適用するために、リソースの選択肢から「サーバー全体」を選びます。あるいは、仮想サーバー内の特定のディレクトリに移動します。
7. 変更するエラーコードごとに、エラー応答を含むファイルまたは CGI への絶対パスを指定します。
8. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

国際文字セットの変更

ドキュメントの文字セットは、ドキュメントが記述されている言語によって部分的に決められます。リソースを選択し、そのリソースに対する文字セットを設定することにより、ドキュメント、ドキュメントセット、またはディレクトリに対してデフォルトで設定されているクライアントの文字セット設定をオーバーライドできます。

HTTP で MIME タイプの charset パラメータを使うと、ブラウザは文字セットを変更できます。サーバーが応答にこのパラメータを挿入すると、それに基づいてブラウザは文字セットを変更します。たとえば、次のような指定があります。

- `Content-Type:text/html;charset=iso-8859-1`
- `Content-Type:text/html;charset=iso-2022-jp`

次の charset 名が RFC 1700 で定められています (x- で始まる名前を除く)。

- us-ascii
- iso-8859-1
- iso-2022-jp
- x-sjis
- x-euc-jp
- x-mac-roman

us-ascii には、次のエイリアスが認識されます。

- ansi_x3.4-1968
- iso-ir-6
- ansi_x3.4-1986
- iso_646.irv:1991
- ascii
- iso646-us
- us
- ibm367
- cp367

iso_8859-1 には、次のエイリアスが認識されます。

- latin1
- iso_8859-1
- iso_8859-1:1987
- iso-ir-100
- ibm819
- cp819

管理インタフェースで、文字セットを変更するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメント処理」タブをクリックします。
5. 「国際文字セット」をクリックします。
6. 仮想サーバー全体に変更を適用するために、リソースの選択肢から「サーバー全体」を選びます。あるいは、仮想サーバー内の特定のディレクトリに移動します。
7. サーバーの全体または一部に文字セットを設定します。

   このフィールドに何も入力しないと、文字セットは NONE に設定されます。
8. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

ドキュメントフッターの設定

サーバーの特定のセクションの全ドキュメントに適用するドキュメントフッターを指定することができます。ドキュメントフッターには最終更新時刻が含まれます。このフッターは、CGI スクリプトの出力と解析される HTML ファイル (.shtml) の出力を除くすべてのファイルに適用されます。ドキュメントフッターを CGI スクリプトの出力や解析された HTML ファイルに表示させるには、フッターテキストを個別のファイルに入力しておき、このファイルをページの出力に加えるために、コード行などのサーバー側インクルードを追加します。

管理インタフェースで、ドキュメントフッターを設定するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「ドキュメント処理」タブをクリックします。
5. 「ドキュメントフッター」をクリックします。

6. 仮想サーバー全体に変更を適用するために、リソースの選択肢から「サーバー全体」を選びます。あるいは、仮想サーバー内の特定のディレクトリに移動します。

   ディレクトリを選択した場合、そのディレクトリかそのディレクトリ内のファイルを表す URL をサーバーが受信したときだけ、ドキュメントフッターが適用されます。

7. フッターを含めるファイルのタイプを指定します。
8. 日時形式を指定します。
9. フッターに表示するテキストを入力します。

   ドキュメントフッターの最大文字数は 765 です。ドキュメントの最終更新日時を含める場合は、文字列 :LASTMOD: を入力します。

詳細については、オンラインヘルプを参照してください。

URL 転送の設定

URL 転送によって、ドキュメント要求を別のサーバーにリダイレクトできます。URL の転送 (リダイレクト) は、URL が変更されたことをサーバーからユーザーに通知する手段です (ファイルを別のディレクトリやサーバーへ移動した場合など)。リダイレクトを使うことによって、あるサーバー上のドキュメントを要求したユーザーを、本人が意識することなく、別のサーバーの上のドキュメントへリダイレクトすることもできます。

たとえば、http://www.sun.com/info/movies をプレフィックス film.sun.com に転送する場合、http://www.sun.com/info/movies の URL は http://film.sun.com/info/movies にリダイレクトされます。

サブディレクトリ内のすべてのドキュメントに対する要求を、特定の URL へリダイレクトしたい場合も考えられます。たとえば、あるディレクトリへのトラフィックが集中していることや、なんらかの理由によりドキュメントを配信できなくなったことにより、そのディレクトリを削除する必要がある場合、ディレクトリ内のドキュメントへの要求を、ドキュメントが利用できなくなった理由を説明するページへ転送することができます。/info/movies のプレフィックスを http://www.sun.com/explain.html へリダイレクトすることなどが可能です。

管理インタフェースで、URL 転送を設定するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。

4. 「HTTP/HTML」タブをクリックします。
5. 「URL 転送」をクリックします。
6. リダイレクトする URL プレフィックスを入力し、そのプレフィックスを別のプレフィックスや静的 URL にリダイレクトするかどうかを指定します。
7. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

サーバーで解析される HTML の設定

通常、HTML がクライアントに送信されるときには、サーバーによる介入を受けずに、ディスクに存在するまま送られます。ただし、サーバーはドキュメントを送信する前に、特別なコマンドについて HTML ファイルを解析 (パース) することができます。HTML ファイルを解析し、要求に固有の情報やファイルをドキュメントに挿入するようサーバーを設定するには、HTML の解析を有効にしておく必要があります。

管理インタフェースで、HTML の解析を設定するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「HTTP/HTML」タブをクリックします。
5. 「HTML を解析」をクリックします。
6. 仮想サーバー全体に変更を適用するために、リソースの選択肢から「サーバー全体」を選びます。あるいは、仮想サーバー内の特定のディレクトリに移動します。

   ディレクトリを選択した場合、そのディレクトリかそのディレクトリ内のファイルを表す URL を受信したときだけ、サーバーは HTML を解析します。
7. サーバーで解析される HTML を有効にするかどうかを選択します。

   HTML ファイルに対してだけ有効にし、exec タグに対しては無効にすることができます。あるいは、HTML ファイルと exec タグの両方に対して有効にすることもでき、この場合は HTML ファイルでサーバー上のほかのプログラムを実行できます。

8. どのファイルを解析するかを選択します。

   .shtml という拡張子を持つファイルだけを解析するか、すべての HTML ファイルを解析するかを選択できます。後者の場合はパフォーマンスが低下します。UNIX を使用している場合は、実行権限がある UNIX ファイルを選択することもできますが、信頼性は保証されません。

9. 「了解」をクリックします。

解析された HTML を受け入れるサーバーの詳しい設定方法については、オンラインヘルプを参照してください。

サーバーで解析された HTML の使用方法に関する詳細は、『Sun ONE Application Server Web アプリケーション開発者ガイド』を参照してください。

キャッシュ制御指令の設定

キャッシュ制御指令は、プロキシサーバーにどの情報をキャッシュさせるかを制御する Sun ONE Application Server の機能です。キャッシュ制御指令を使うことで、プロキシによるデフォルトのキャッシングがオーバーライドされ、機密情報をキャッシュせずに後から検索することができます。この指令を利用するには、プロキシサーバーが HTTP 1.1 に準拠している必要があります。

HTTP 1.1 の詳細については、『Hypertext Transfer Protocol--HTTP/1.1 specification (RFC 2068)』を参照してください。URL は次のとおりです。

http://www.ietf.org/

管理インタフェースで、キャッシュ制御指令を設定するには、次の手順に従います。

1. 左側のペインでアプリケーションサーバーインスタンスを選択し、HTTP サーバーを開きます。
2. 「仮想サーバー」を開きます。
3. 編集する仮想サーバーの名前をクリックします。
4. 「HTTP/HTML」タブをクリックします。
5. 「キャッシュ制御指令」をクリックします。
6. フィールドに必要な情報を入力します。応答指令に有効な値は、次のとおりです。
   - **公開 (Public):** どのキャッシュでも、応答をキャッシュできる。これはデフォルトのオプションである
   - **非公開 (Private):** プライベート (非共有) キャッシュだけで、応答をキャッシュできる
   - **キャッシュ無し (No Cache):** どこにも、応答をキャッシュできない

- **ストアなし (No Store):** 不揮発性ストレージのどこにも、要求や応答のキャッシュを保存できない
- **再検証が必要 (Must Revalidate):** キャッシュエントリは発信元サーバーに再検証されることが必要
- **最長有効期間 (秒) (Maximum Age):** クライアントは、この期間を超過している応答を受け入れない

7. 「了解」をクリックします。

詳細については、オンラインヘルプを参照してください。

より強力な暗号化方式の使用

より強力な暗号化方式の設定方法については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

第 4 部

付録

付録 A 「コマンド行インタフェースの使用」

付録 B 「サードパーティ製品の著作権について」

付録 A

コマンド行インタフェースの使用

この付録では、コマンド行インタフェース (asadmin ユーティリティ) の使用方法について説明します。システムプロンプトでのシングルモード (コマンドプロンプトで一度に 1 つのコマンドを実行) とマルチモード (環境レベルの情報の再入力を必要とせずに複数のコマンドを実行)、およびスクリプト内やプログラム内で実行する方法があります。コマンド行インタフェースは、管理インタフェース画面の代わりに使用できます。

この付録には次の節があります。

- コマンド行インタフェースについて
- asadmin の使用
- セキュリティに関する注意事項
- 同時アクセスに関する注意事項
- コマンドリファレンス

コマンド行インタフェースについて

この節には次の項目があります。

- asadmin ユーティリティについて
- Ant タスクについて
- その他のコマンド行ユーティリティについて

asadmin ユーティリティについて

asadmin ユーティリティを使うと、すべての設定タスクおよび管理タスクを実行できます。このユーティリティは、管理インタフェースの代わりに使用できます。

Ant タスクについて

多くの開発者は、Ant を使って、J2EE アプリケーションの開発プロセスにかかる時間を短縮しています。Ant スクリプトは、いくつかのタスクで asadmin ユーティリティを利用します。開発者は Ant タスクを使って、アプリケーションの構築、モジュールやアプリケーションの配備または配備の取消し、および Sun ONE Application Server の制御を行います。

Ant タスクの詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

Ant の詳細については、Jakarta プロジェクトのサイト (http://jakarta.apache.org/ant/) を参照してください。

その他のコマンド行ユーティリティについて

Sun ONE Application Server には、追加のコマンド行ユーティリティが付属しています。ユーティリティの一覧とそれぞれの簡単な説明を次の表に示します。

その他のコマンド行ユーティリティ

ユーティリティ	定義
appclient	Application Client Container を起動し、アプリケーション JAR ファイルにパッケージ化されているクライアントアプリケーションを呼び出す
capture-schema	データベースのスキーマとマッピング情報を取得する
flexanlg	サーバーに関する統計情報を生成する
htpasswd	ユーザー認証ファイルを作成する
package-appclient	アプリケーションクライアントコンテナのライブラリと JAR ファイルをパックする。詳細は、『Sun ONE Application Server Developer's Guide to Clients』を参照してください。
verifier	DTD によって J2EE 配備記述子を検証する。詳細については、『Sun ONE Application Server 開発者ガイド』を参照

その他のコマンド行ユーティリティ (続き)

ユーティリティ	定義
wscompile	サービス定義インタフェースを使って、クライアントスタブまたはサーバー側スケルトンを生成する。つまり、該当のインタフェースに対応する一連の WSDL (Web サービス記述言語) を生成する
wsdeploy	配備可能な WAR ファイルを生成する

これらのユーティリティの詳細については、それぞれのオンラインヘルプを参照してください。

asadmin の使用

asadmin ユーティリティは、管理タスクを実行するためのコマンドセットを備えています。管理インタフェースで実行できるほとんどのタスクが、これらのコマンドで実行できます。asadmin ユーティリティは *install_dir*/bin に格納されているため、その場所から実行できます。Windows では、asadmin.bat ファイルをダブルクリックすると、コマンドウィンドウでマルチモードの asadmin ユーティリティが起動します。

HTTP サーバー関連のプロパティと管理サーバーのプロパティには、コマンド行で設定できないものがあります。これらの設定には、管理インタフェースを使用してください。server.xml 設定ファイルに保存されているすべてのプロパティを設定できますが、init.conf と obj.conf に保存されているプロパティを設定することはできません。設定ファイルの詳細については、『Sun ONE Application Server 管理者用構成ファイルリファレンス』を参照してください。

各コマンドの詳細については、431 ページの「コマンドリファレンス」とコマンドのヘルプを参照してください。

この節には次の項目があります。

- コマンド構文について
- シングルモードとマルチモードの使用
- 対話型オプションと非対話型オプションの使用
- 環境コマンドの使用
- パスワードファイルオプションの使用
- ローカルまたはリモートでの asadmin の実行
- コマンド行呼び出しの使用

- エスケープ文字の使用
- get コマンドと set コマンドの使用
- ヘルプの使用
- 出力とエラーの表示

コマンド構文について

`asadmin` ユーティリティの構文は次のとおりです。

asadmin command -short-option argument --long-option argument operand

コマンド

コマンドとは、実行される操作またはタスクのことです。コマンドには、大文字と小文字の区別があります。

オプション

オプションによって、ユーティリティによるコマンドの実行方法を変更できます。アルファベットの大文字と小文字は区別されます。短形式のオプションの前にはダッシュを 1 つ付けます (-)。長形式のオプションの前にはダッシュを 2 つ付けます (--)。多くのオプションは、短形式でも長形式でも使用できます。たとえば、`--user` と `-u` のどちらを使用してもかまいません。オプションには、必須オプションと省略できるオプションがあります。コマンド構文では、省略できるオプションをかっこで囲んで表しています。コマンドの実行時にはすべての必須オプションを指定する必要があります。指定しないと、エラーメッセージが返され、コマンドは実行されません。

使用可能な長形式および短形式のオプション名については、「コマンドリファレンス」の項目の中の 464 ページの表「各オプションに対応する短形式、長形式、デフォルト値、および環境変数」を参照してください。

ほとんどのオプションには引数が必要です。たとえば --port には引数 *port_number* を指定します。ただし、ブール型のオプションは、機能のオンまたはオフを切り替えるために使用されるため、引数を必要としません。

オプションを環境変数に保存することもできます。詳細については、418 ページの「環境コマンドの使用」を参照してください。オプションに対応する環境変数については、464 ページの「各オプションに対応する長形式、短形式、デフォルト値、および環境変数」を参照してください。

ブール型のオプション

ブール型のオプションでは、オンまたはオフを切り替えます。たとえば、`--interactive` を指定すると対話モードに切り替わります。`--no-interactive` を指定すると対話モードがオフになります。対話モードでは、オプションに関してプロンプトが表示されます。長形式のオプションの前に --no- を指定すると、そのオプションがオフになります。短形式のオプション名を指定すると、常にデフォルト値の逆の設定になります。

短形式のブール型オプションはまとめて指定できます。たとえば、対話モード (短形式のオプション名 `-I`) とエコー (短形式のオプション名 `-e`) を指定したい場合、`-Ie` と指定することができます。

オペランド

オペランドは、空白文字またはタブで区切って指定します。コマンド構文内にどの順番で指定してもかまいません。オペランドに続いてオプションを指定せずに -- を記述すると、オプションとオペランドを区別できます。その後に続く引数は、ダッシュ (-) で始まるものも含め、すべてオペランドとして扱われます。例を示します。

```
asadmin> create-jvm-options --instance server1 -- -Xmx1500m
```

`-XMx1500m` はダッシュで始まっていますが、オペランドとして扱われます。

構文例

`asadmin create-instance [--user` *admin_user*`] [--password` *admin_password*`] [-H` *host_name*`] [--port` *port_number*`] [--sysuser` *sys_user*`] [--domain` *domain_name*`] [--local=`*true/false*`] [--passwordfile` *file_name*`] [--secure | -s] --instanceport` *instance_port* *instance_name*

この構文例では、`-H` はホスト名の短形式のオプション、`--user` は *admin_user* を引数とする長形式のオプション、*instance_name* はオペランドです。省略できるオプションは、かっこで囲まれています。

次に、構文に実際の値を指定した例を示します。この例では、省略できるオプションの一部が指定されていません。

`asadmin create-instance --user admin --password` *password* `-H austen --port 4848 --instanceport 1024 server2`

シングルモードとマルチモードの使用

asadmin は、シングルモードまたはマルチモードで実行できます。シングルモードでは、コマンドプロンプトからコマンドを 1 つずつ実行します。マルチモードでは、環境レベルの情報を繰り返し入力することなく、複数のコマンドを実行できます。

シングルモードでファイルからの入力を使用している場合、コマンドの実行に失敗するとプログラムは終了します。マルチモードでコマンドの実行に失敗すると、asadmin のプロンプトが再度表示されます。

シングルモード

シングルモードでは、コマンド行インタフェースを使ってコマンドプロンプトから単一のコマンドを呼び出します。コマンド行インタフェースによってコマンドが実行され、再びコマンドプロンプトが表示されます。コマンドプロンプトからコマンド行インタフェースを実行するには、*install_dir*/appserv/bin ディレクトリに移動し、次のようにコマンドを入力します。

```
> asadmin command options arguments
```

次に例を示します。

```
> asadmin create-instance --user admin --password password -H austen
--port 4848 --instanceport 1024 server2
```

マルチモード

マルチモードでは、最初に環境設定を行うことにより、サーバー名、ポート、パスワードなどの環境レベルの情報を再入力することなく、複数のコマンドを実行できます。マルチモードを使用する利点は、asadmin がメモリにとどまるため、コマンドの入力と実行が非常に速くなる点です。環境変数がオペレーティングシステムのレベルで設定されている場合、マルチモードではそれらの設定が取り込まれます。それらの設定は、変更されないかぎり asadmin ユーティリティによって引き続き使用されます。

Windows では、asadmin.bat ファイルを実行すると、自動的にマルチモードになります。

UNIX では、asadmin ユーティリティをコマンド行からマルチモードで実行するには、次のコマンドを入力します。

```
> asadmin multimode
```

マルチモードの場合は、コマンドプロンプトが asadmin に変わります。次に、asadmin プロンプトにコマンドを入力します。ユーティリティ名を入力する必要はありません。次に例を示します。

```
asadmin> create-instance --user admin --password password -H austen
--port 4848 --instanceport 1024 server2
```

exit または quit と入力すると、マルチモードが終了します。コマンドプロンプトに戻ります。

複数のマルチモード

マルチモードセッション内で次のコマンドを入力すると、さらにマルチモードを呼び出すこともできます。

```
asadmin> multimode
```

2 番目のマルチモード環境を終了すると、最初のマルチモード環境に戻ります。

たとえば、server1 をマルチモードで管理しているとき、server2 を使って両者を比較したい場合は、server1 のマルチモードで server2 のマルチモードを呼び出します。現在のマルチモードセッションを終了する必要がないので、環境設定をそのまま保持できます。server2 のマルチモードセッションを終了すると、server1 のマルチモード環境に戻ります。

対話型オプションと非対話型オプションの使用

コマンド行インタフェースは、対話型モードまたは非対話型モードで使用できます。対話型モードでは、パスワードが指定されていない場合、パスワードの入力を求めるメッセージが表示されます。対話型モードはデフォルトで有効になっています。

export コマンドを使って対話型環境変数を設定することにより、対話型モードを有効および無効に切り替えることができます。詳細については、「export コマンドに指定する環境変数」の表を参照してください。

どのような場合でも、シングルモードで対話型オプションを使用できます。マルチモードで対話型オプションを使用できるのは、コマンドプロンプトからコマンドを 1 つずつ実行する場合と、ファイルからマルチモードで実行する場合です。ただし、マルチモードでは、入力ストリームからパイプされたコマンドや、別のプログラムから呼び出されたコマンドを対話型モードで実行できません。

環境コマンドの使用

asadmin ユーティリティには、環境コマンドを使用して設定できる一連の環境変数が含まれます。マルチモードでは、これらの変数を設定したあと、マルチモードを終了するまで環境を設定し直す必要はありません。環境変数をオペレーティングシステムのレベルで設定することもできます。その場合、マルチモードに入ると、それらの環境変数は自動的に読み込まれ、マルチモードを終了するまで保持されます。

環境変数は名前と値の組み合わせであり、いつでも値を割り当てて設定できます。環境変数には、AS_ADMIN_ というプレフィックスが付けられて、大文字を使用したオプション名となります。たとえば、管理サーバーのユーザーを設定する場合は、次のように入力します。

```
export AS_ADMIN_USER=administrator
```

administrator は管理者のユーザー名です。

これによって、次のように AS_ADMIN_USER の値を asadmin コマンドにも使用できます。

```
asadmin multimode
asadmin> export AS_ADMIN_HOST=austen
```

管理サーバーのホスト名は、新たに割り当てないかぎり、マルチモードセッションを終了するまで austen になります。

次の例のように、複数の環境変数をまとめて設定し、エクスポートすることもできます。

```
asadmin> export AS_ADMIN_PORT=4848 AS_ADMIN_USER=admin
```

現在の環境変数の設定を確認するには、引数を指定しないで export コマンドを実行します。

```
asadmin> export
AS_ADMIN_HOST=austen
AS_ADMIN_PORT=4848
AS_ADMIN_USER=admin
```

変数とその値を環境から削除するには、unset コマンドを使用します。次に例を示します。

```
asadmin> unset AS_ADMIN_HOST
```

環境変数の値は、変数を設定し直すか、asadmin コマンドの一部として別の値を設定することにより、オーバーライドできます。次に例を示します。

```
asadmin> export AS_ADMIN_HOST=dickens
asadmin> show-instance-status --host austen instance-name
```

この例では、管理サーバーホスト austen のインスタンスの状態が示されます。この値によって、以前のホストの値 dickens がオーバーライドされているためです。

エクスポートされた変数を使用しない場合は、ほとんどのコマンドで、次に示すオプションを指定するか、デフォルト値を使用する必要があります (デフォルト値の一覧については、464 ページの「各オプションに対応する長形式、短形式、デフォルト値、および環境変数」を参照)。

- --host
- --port
- --user
- --password または --passwordfile
- --secure=true (セキュリティ保護されている場合)
- --instance (必要に応じて指定)

次の表「export コマンドに指定する環境変数」では、export コマンドに指定できる環境変数について説明します。これらの変数は、環境設定用としてもっとも一般的に使用される変数です。第 1 列は環境変数名を、第 2 列は用途と、値が設定されていない場合のデフォルト値を示します。環境変数については、464 ページの「各オプションに対応する長形式、短形式、デフォルト値、および環境変数」を参照してください。

export コマンドに指定する環境変数

環境変数	用途
AS_ADMIN_HOST	管理サーバーのホスト名。値を指定しない場合は、localhost が使用される
AS_ADMIN_PORT	管理サーバーのポート番号。値を指定しない場合は、4848 が使用される
AS_ADMIN_USER	コマンドを実行するユーザーの名前
AS_ADMIN_PASSWORD	コマンドを実行するユーザーのパスワード。ユーザー名とパスワードは、ユーザーを認証するため、つまりユーザーにサーバーの管理が許可されているかどうかを確認するために使用される。これは、管理インタフェースから管理サーバーにアクセスするときに行われる認証と同じ
AS_ADMIN_SECURE	セキュリティ保護される場合は true
AS_ADMIN_INSTANCE	Sun ONE Application Server のインスタンスを設定する。インスタンス名をオペランドではなく引数として使用する、後続のすべてのコマンドは、この変数に指定されたインスタンスを使用する

パスワードファイルオプションの使用

コマンド行でパスワードを入力したくない場合やパスワードの環境変数を設定したくない場合は、パスワードファイルを作成しておき、そのファイルをコマンド行のオプションとして使用できます。

password オプションを指定できるコマンドには、passwordfile オプションを代わりに指定できます。パスワードファイルには、次の行を含めます。

AS_ADMIN_PASSWORD=*value*

AS_ADMIN_ADMINPASSWORD=*value*

AS_ADMIN_USERPASSWORD=*value*

passwordfile オプションを使用すると、ファイル内に記述したパスワードはマルチモード環境にエクスポートされるため、後続の password オプションを指定していないコマンドでも、これらの値が使用されます。

コマンド行に password オプションと passwordfile オプションを同時に指定した場合は、パスワードファイルに記述された値がマルチモード環境にエクスポートされますが、そのコマンドでは password オプションに指定されているパスワードが使用されます。password オプションが passwordfile オプションよりも優先されるためです。

ローカルまたはリモートでの asadmin の実行

通常、asadmin ユーティリティは、管理サーバーを介してコマンドを受け渡します。このため、Sun ONE Application Server がインストールされているシステムで asadmin を実行する必要はありません。ただし、ほとんどの asadmin コマンドが動作するには、管理サーバーが実行中であることが必要です。

create-instance など、一部のコマンドには、ローカルで実行するためのオプションを指定できます。--local=true オプションを指定して create-instance コマンドを使用する場合は、管理サーバーがインストールされているマシン上で実行する必要があります。ただし、管理サーバーを実行して、インスタンスを作成する必要はありません。

一部のコマンドは、ローカルで実行する必要があります。たとえば、管理サーバーを起動し、すべてのインスタンスを作成する start-appserv をリモートで実行することはできません。これは、このコマンドによって起動するまで、管理サーバーは実行されていないためです。

管理サーバーの詳細については、第 2 章「管理サーバーの設定」を参照してください。

次のコマンドは、ローカルとリモートの両方で実行できます。

- `create-instance`
- `delete-instance`
- `list-instances`
- `start-instance`
- `stop-instance`
- `display-license`
- `version`
- `stop-domain`
- `restart-instance`
- `list-domains`

これらのコマンドは、local オプションを指定しなくても、ローカルで実行できます。デフォルトでは、コマンド構文でユーザー、パスワード、ホスト、またはポートの値を指定すると、コマンドはリモートコマンドとして扱われます。ただし、これらのオプションにローカルの値を指定することも可能です。デフォルトでは、これらのオプションに値を指定しない場合、コマンドはローカルで実行されます。

`domain` オプションを指定できるコマンドをローカルで実行するときには、ドメインが 1 つだけの場合でも `domain` オプションを指定する必要があります。コマンドをリモートで実行するときには、`domain` オプションを指定しても無視されます。

コマンド行呼び出しの使用

コマンド行の呼び出しには、さまざまな方法があります。次の各項目で説明します。

- コマンド行からの asadmin の使用
- ファイルからの入力 (スクリプト) での asadmin の使用
- 標準入力 (パイプ) での asadmin の使用

コマンド行からの asadmin の使用

もっとも単純なコマンドの使用方法は、コマンド行から 1 つずつ実行する方法です。ユーティリティ名に続けて、コマンドとそのオプションおよび引数を指定します。マルチモードでは、ユーティリティ名と環境オプションを繰り返し入力することなく、複数のコマンドを実行できます (環境変数を設定済みの場合)。シングルモードのコマンドもマルチモードのコマンドも、対話型形式 (パスワードなどの追加入力を求めるプロンプトを表示する) または非対話型形式で実行できます。

シングルモードとマルチモードの詳細については、416 ページの「シングルモードとマルチモードの使用」を参照してください。

コマンドを対話型形式で使用する方法については、417 ページの「対話型オプションと非対話型オプションの使用」を参照してください。

コマンド行の使用例

```
> asadmin create-instance --user admin --password password --host
austen --port 4848 --instanceport 1024 server2
```

コマンドの実行が完了すると、オペレーティングシステムのプロンプトに戻ります。

ファイルからの入力 (スクリプト) での asadmin の使用

複数の asadmin コマンドを含むスクリプトを作成できます。スクリプトを使うと、バッチモードでのコマンド処理、ジョブの実行回数の設定、管理タスクの単純化および自動化を行うことができます。

ファイル内のスクリプトを呼び出すには、次の構文を使用します。

```
> asadmin multimode --file filename
```

次に、この方法で呼び出せる、ファイル内の単純なスクリプトの例を示します。

```
# Create new instance and start it.
export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=mypassword
AS_ADMIN_HOST=austen AS_ADMIN_PORT=4848
create-instance --instanceport 9000 austen3
start-instance austen3
```

このスクリプトは、環境設定を行い、austen3 というインスタンスを作成し、新しいインスタンスを起動します。ハッシュ記号 (#) で始まる行はコメントと見なされ、無視されます。

標準入力 (パイプ) での asadmin の使用

入力を asadmin ユーティリティにパイプすることができます。次の構文を使用します。

```
cat filename | asadmin multimode
```

この構文は、Windows では動作しません。

エスケープ文字の使用

エスケープ文字で区切らずにコロン (:)、アスタリスク (*)、およびバックスラッシュ (\) などの文字をコマンド構文で使用すると、エラーが発生します。エスケープ文字が必要となる場面は、プラットフォームやシングルモードとマルチモードの違いによって変わります。

注	get コマンドと set コマンドでは、コロンに対してエスケープ文字を使用する必要はありません。

この節には次の項目があります。

- UNIX のシングルモードでのエスケープ文字
- Windows のシングルモードでのエスケープ文字
- プラットフォームを問わないシングルモードでのエスケープ文字
- プラットフォームを問わないマルチモードでのエスケープ文字

UNIX のシングルモードでのエスケープ文字

Solaris では、二重のバックスラッシュ (\\) または二重引用符 (" ") を使用して、予約されている文字をエスケープします。

バックスラッシュ (\\) によるエスケープ

たとえば、値にコロンを含むオプションを指定して JDBC 接続プールを作成するときに、バックスラッシュを使用できます (一部のプロパティに関して環境変数が設定されていることが必要)。

```
asadmin create-jdbc-connection-pool --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url=jdbc\\:oracle\\:thin\\:@asperfsol8\\:1521\\:V8i:user=staging_lo
okup_app:password=staging_lookup_app OraclePoollookup
```

引用符によるエスケープ

上記の例と同じ内容に対して引用符を使用するには、値を引用符 (") で囲み、さらにこれらの引用符をバックスラッシュでエスケープします。

```
asadmin create-jdbc-connection-pool --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url=\"jdbc:oracle:thin:@asperfsol8:1521:V8i\":user=staging_lookup_a
pp:password=staging_lookup_app OraclePoollookup
```

424 ページの「プラットフォームを問わないシングルモードでのエスケープ文字」で説明している方法を使用することもできます。

Windows のシングルモードでのエスケープ文字

Windows では、バックスラッシュを使ってエスケープできます。たとえば、値にコロンを含むオプションを指定して JDBC 接続プールを作成するときに、バックスラッシュを使用できます (一部のプロパティに関して環境変数が設定されていることが必要)。

```
asadmin create-jdbc-connection-pool --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url=jdbc\:oracle\:thin\:@asperfsol8\:1521\:V8i:user=staging_lookup_
app:password=staging_lookup_app OraclePoollookup
```

424 ページの「プラットフォームを問わないシングルモードでのエスケープ文字」で説明している方法を使用することもできます。

プラットフォームを問わないシングルモードでのエスケープ文字

どのプラットフォームでも、バックスラッシュを文字の前に使い、その文字を含む値を二重引用符で囲むことによって、エスケープできます。たとえば、値にコロンを含むオプションを指定して JDBC 接続プールを作成するときに、次のようにエスケープ文字を使用できます (一部のプロパティに関して環境変数が設定されていることが必要)。

```
asadmin create-jdbc-connection-pool --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url="jdbc\:oracle\:thin\:@iasperfsol8\:1521\:V8i":user=staging_look
up_app:password=staging_lookup_app OraclePoollookup
```

プラットフォームを問わないマルチモードでのエスケープ文字

マルチモードでは、スラッシュやバックスラッシュなどを必要とせずに、引用符だけの次のような構文を使うことができます。

```
asadmin> create-jdbc-connection-pool --instance server1
--datasourceclassname oracle.jdbc.pool.OracleDataSource
--failconnection=true --isconnectvalidatereq=true --property
url="jdbc:oracle:thin:@asperfsol8:1521:V8i":user=staging_lookup_app
:password=staging_lookup_app OraclePoollookup
```

get コマンドと set コマンドの使用

get コマンドと set コマンドを使うと、Sun ONE Application Server の設定情報にアクセスおよび変更できます。ほとんどの場合、asadmin コマンドは必須プロパティだけを設定します。オプションのプロパティの値を変更するには、set コマンドを使用します。

get コマンドと set コマンド

コマンド	引数	用途
get	(scope) scope は属性を表す有効な名前	属性の値を取得する
set	(scope=*value*) scope は属性を表す有効な名前。*value* はその属性に設定する値	属性の値を設定する
reconfig	instance-name	設定ファイルの内容を変更するコマンドを実行したあと、サーバーに変更を適用するには、reconfig を実行することが必要。サーバーの変更および再設定の適用については、78 ページの「アプリケーションサーバーインスタンスの変更の適用」を参照

1 つのコマンドで、属性と属性の間に空白文字を使うことにより、複数の属性値を取得または設定できます。次に例を示します。

```
set server1.appReloadPollInterval=20
server1.mime.mime1.file=mime.types
```

また、AS_ADMIN_PREFIX 環境変数を使って、後続の get コマンドや set コマンドが使うプレフィックスを設定することもできます。プレフィックス文字列と、get コマンドまたは set コマンドのオペランドとの間にピリオド (".") が挿入されます。次に例を示します。

```
asadmin>export AS_ADMIN_PREFIX=server1
asadmin>get *
server1.locale = en_US
server1.appReloadPollInterval = 2
server1.name = server1
...
```

get コマンドと set コマンドは区切り文字としてピリオドを必要とするため、アイテム名にピリオドが含まれる場合は、含まれるピリオドの前にエスケープ文字のバックスラッシュ (\) を記述する必要があります。サーバーインスタンス名 server2.sun.com のピリオドの前にバックスラッシュを記述した例を、次に示します。

```
get server2\.sun\.com.*
```

バックスラッシュを挿入しないと、エラーメッセージが表示されます。

get コマンドと set コマンドの例

次に、get コマンドを使って属性値を取得する例と、set コマンドを使って値を設定する例を示します。

MDB コンテナサービスの例

アプリケーションサーバーインスタンスが server1 の場合、環境を設定し、次のコマンドをマルチモードで実行することにより、すべての mdb-container 属性の値を取得できます。

```
asadmin> get server1.mdb-container.*
```

次に、このコマンドの出力例を示します。出力には現在の属性値が示されます。

```
server1.mdb-container.logLevel = null
server1.mdb-container.steadyPoolSize = 10
server1.mdb-container.idleInPoolTimeoutInSeconds = 600
server1.mdb-container.maxPoolSize = 60
server1.mdb-container.monitoringEnabled = false
server1.mdb-container.poolResizeQuantity = 2
```

MDB コンテナ属性の monitoringEnabled の値だけを取得するには、次のコマンドを使用します。

```
asadmin> get server1.mdb-container.montioringEnabled
```

monitoringEnabled 属性の値を true に設定するには、次のコマンドを使用します。

```
asadmin> set server1.mdb-container.montioringEnabled=true
```

JMS リソースの例

リソースを設定するには、次のように属性を指定します。

instancename.resource.primary_key_value.attribute_name

次に例を示します。

```
asadmin> get server1.jms-resource.myjms.*
```

これにより、JMS 送信先リソース myjms のすべての属性を取得できます。次に例を示します。

```
server1.jms-resource.myjms.resType = javax.jms.Topic
server1.jms-resource.myjms.enabled = true
server1.jms-resource.myjms.name = myjms
server1.jms-resource.myjms.description = null
```

resType など、単一の属性の値を取得するには、次のように入力します。

```
asadmin> get server1.jms-resource.myjms.resType
```

description 属性を設定するには、次のように入力します。

```
asadmin> set server1.jms-resource.myjms.description=mydescription
```

この例では、mydescription に説明が設定されます。

複数の値の取得例と設定例

1 つのコマンドで複数の値を取得および設定することができます。同時に 2 つの属性を設定するには、属性の間を空白文字で区切ります。次に例を示します。

```
set server1.appReloadPollInterval=20
server1.mime.mime1.file=mime.types
```

また、AS_ADMIN_PREFIX 環境変数を使って、複数の get コマンドや set コマンドが使うプレフィックスを設定することもできます。

get コマンドと set コマンドによる監視

get コマンドおよび set コマンドを使って、実行中のサーバーを監視することもできます。list コマンドで、監視することもできます。monitor オプションを true または false に設定します。true に設定した場合は、指定された属性を監視できます。コマンド行インタフェースを使って Sun ONE Application Server を監視する方法の詳細については、130 ページの「CLI を使用した監視データの抽出」を参照してください。

ヘルプの使用

個々の asadmin コマンドのヘルプを表示するには、コマンドプロンプトで -h または --help と入力します。たとえば、asadmin のヘルプを表示するには、次のコマンドを入力します。

```
asadmin --help
```

すべての asadmin コマンドが一覧表示されます。

特定の asadmin コマンドのヘルプを表示するには、次のように入力します。

asadmin *command* -h

または

asadmin *command* --help

ヘルプには、構文、コマンドの説明、構文の説明、使用例、および関連コマンドが表示されます。

コマンド内の任意の位置に -h または --help を指定すると、そのコマンドのヘルプが表示されます。コマンドは実行されません。

UNIX 環境では、マニュアルページとしてコマンド行のヘルプページにアクセスすることもできます。アンバンドル版では、*install_dir*/man を MANPATH 環境変数に追加してください。一度追加すると、Sun ONE Application Server ユーティリティのマニュアルページにアクセスできるようになります。たとえば、コマンドプロンプトで man asadmin と入力するとアクセスできます。

出力とエラーの表示

コマンドが正常に実行されると、実行内容を知らせるメッセージが表示されます。コマンドの実行に失敗すると、エラーメッセージが表示されます。

この節には次の項目があります。

- 終了状態の表示
- 使用法の表示

終了状態の表示

エラーメッセージに加えて、asadmin コマンドの終了時には、常に終了状態が返されます。終了状態は、コマンドの実行が成功した場合は 0、失敗した場合は 1 になります。

UNIX での終了状態

コマンドプロンプトで `echo $?` と入力することにより、終了状態をチェックできます。

スクリプトでも終了コードを使用できます。たとえば、次の Korn シェルスクリプトは、終了状態を使用して `list-instances` コマンドが成功であるか失敗であるかを示します。

```
#!/bin/ksh
asadmin list-instances
if [[ $? = 0 ]]
   then
      echo "success"
   else
      echo "error"
fi
```

Windows での終了状態

Windows では、独自の .bat スクリプトを使って終了状態をチェックできます。成功するスクリプトとその出力、および失敗するスクリプトとその出力を、次に示します。

成功するスクリプト

```
myscript.bat
------------
echo off
echo Processing Command
call asadmin list-instances --domain domain1
if not %errorlevel% EQU 0 goto end
echo Command Succesful
goto program-end
:end
echo Command Failed
:program-end
```

出力

```
Processing Command
admin-server <not running>
server1 <not running>
Command Succesful
```

失敗するスクリプト

```
myscript.bat
-------------
echo off
echo Processing Command
call asadmin list-instances
```

```
if not %errorlevel% EQU 0 goto end
echo Command Succesful
goto program-end
:end
echo Command Failed
:program-end
```

出力

```
Processing Command
No default domain. Need to enter a domain.
Command Failed
```

使用法の表示

引数を指定せずにコマンドを実行すると、コマンドの構文を示すエラーメッセージが表示されます。次に例を示します。

```
asadmin> create-instance

Invalid number of operands received

USAGE:create-instance [--user admin_user] [--password
admin_password] [--host localhost] [--port 4848] [--sysuser
sys_user] [--domain domain_name] [--local=false] [--passwordfile
file_name] [secure | -s] --instanceport instanceport instancename
```

セキュリティに関する注意事項

コマンド行からコマンド行インタフェースを実行する場合は、すべてのコマンドにパスワードが必要になります。マルチモードで実行する場合は、最初の環境設定時にパスワードを入力します。マルチモードをいったん終了して、もう一度開始するときは、再び環境設定を行い、パスワードを入力します。パスワードの設定には、環境コマンドを使います。詳細については、418 ページの「環境コマンドの使用」を参照してください。

コマンド行でパスワードを入力しなくていいように、パスワードファイルを設定しておくこともできます。詳細については、420 ページの「パスワードファイルオプションの使用」を参照してください。

有効なユーザー名およびパスワードの認証情報がなければ、コマンドは実行されません。

コマンド行インタフェースは、使用している Sun ONE Application Server 用に設定したセキュリティ基準に従います。Sun ONE Application Server のセキュリティ関連情報については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

同時アクセスに関する注意事項

コマンド行インタフェースや管理インタフェースを使って、複数のユーザーが同時にサーバーの設定を行う場合があります。この場合、2 番目の設定要求は、最初の要求が完了するまでキューに入ります。キュー内での待ち時間が長くなると、タイムアウトになります。

一部のコマンドについては、変更内容は reconfig コマンドを実行するまで適用されません。したがって、変更内容がサーバーに適用される前に、複数のユーザーが同じ属性を編集する可能性があります。reconfig の詳細については、78 ページの「アプリケーションサーバーインスタンスの変更の適用」を参照してください。

コマンドリファレンス

この節には次の項目があります。

- コマンドの一覧
- ドット表記名と属性の一覧
- 各オプションに対応する長形式、短形式、デフォルト値、および環境変数

コマンドの一覧

次の表に、すべての asadmin コマンドとその用途を示します。コマンドの構文と使用方法については、オンラインヘルプを参照してください。

左側の列にはコマンド名、右側の列には用途が記述されています。

asadmin コマンド

コマンド	用途
add-resources	型が jdbc、jms、または javamail の 1 つまたは複数のリソースを追加する
create-acl	ACL (アクセス制御リスト) を作成する
create-authdb	認証データベースを作成する
create-auth-realm	認証レルムを作成する
create-custom-resource	カスタムリソースを作成する
create-domain	ドメインを作成する

asadmin コマンド (続き)

コマンド	用途
create-file-user	キーファイルにファイルレルムユーザーを作成する
create-http-listener	HTTP リスナーを作成する
create-http-qos	アプリケーションサーバーインスタンスまたは仮想サーバーの HTTP サービス品質の設定を作成する
create-iiop-listener	IIOP リスナーを作成する
create-instance	アプリケーションサーバーインスタンスを作成する
create-javamail-resource	Java メールリソースを作成する
create-jdbc-connection-pool	JDBC 接続プールを作成する
create-jdbc-resource	JDBC リソースを作成する
create-jmsdest	JMS (Java Message Service) 送信先を作成する
create-jms-resource	JMS リソースを作成する
create-jndi-resource	JNDI リソースを作成する
create-jvm-options	java-config 要素または profiler 要素の JVM オプションを作成する
create-lifecycle-module	ライフサイクルモジュールを作成する
create-mime	MIME タイプファイルを作成する
create-persistence-resource	持続マネージャファクトリリソースを作成する
create-profiler	JVM のプロファイラを作成する
create-ssl	HTTP リスナー、IIOP リスナー、または IIOP サービスの SSL 設定を作成する
create-virtual-server	仮想サーバーを作成する
delete-acl	ACL を削除する
delete-authdb	認証データベースを削除する
delete-auth-realm	認証レルムを削除する
delete-custom-resource	カスタムリソースを削除する
delete-domain	ドメインを削除する。このコマンドは、ローカルでのみ実行可能

asadmin コマンド (続き)

コマンド	用途
delete-file-user	キーファイルからファイルレルムユーザーを削除する
delete-http-listener	HTTP リスナーを削除する
delete-http-qos	アプリケーションサーバーインスタンスまたは仮想サーバーの HTTP サービス品質の設定を削除する
delete-iiop-listener	IIOP リスナーを削除する
delete-instance	アプリケーションサーバーインスタンスを削除する
delete-javamail-resource	Java メールリソースを削除する
delete-jdbc-connection-pool	JDBC 接続プールを削除する
delete-jdbc-resource	JDBC リソースを削除する
delete-jmsdest	JMS 送信先を削除する
delete-jms-resource	JMS リソースを削除する
delete-jndi-resource	JNDI リソースを削除する
delete-jvm-options	java-config 要素または profiler 要素の JVM オプションを削除する
delete-lifecycle-module	ライフサイクルモジュールを削除する
delete-mime	MIME タイプファイルを削除する
delete-persistence-resource	持続マネージャファクトリリソースを削除する
delete-profiler	JVM プロファイラを削除する
delete-ssl	HTTP リスナー、IIOP リスナー、または IIOP サービスの SSL 設定を削除する
delete-virtual-server	仮想サーバーを削除する
deploy	EJB、WEB、コネクタ、appclient、またはアプリケーションコンポーネントをアプリケーションサーバーインスタンスに配備する
deploydir	指定したディレクトリ内の EJB、WEB、コネクタ、appclient、またはアプリケーションコンポーネントをアプリケーションサーバーインスタンスに配備する
disable	アプリケーションサーバーインスタンスに配備されたコンポーネントを無効にする

asadmin コマンド (続き)

コマンド	用途
display-license	ライセンス情報を表示する。このコマンドは、ローカルでのみ実行可能
enable	アプリケーションサーバーインスタンスに配備されたコンポーネントを有効 (実行可能) にする
export	後続の asadmin コマンドで使用するために、asadmin 環境変数の値をエクスポートする
get	属性の値を取得する
help	指定されたコマンドのヘルプ (説明、使用法、構文、使用例)、または asadmin の一般的なヘルプを表示する
install-license	ライセンスファイルをインストールする。このコマンドは、ローカルでのみ実行可能
jms-ping	JMS プロバイダが稼働しているかどうかを ping で確認する
list	設定可能な要素を一覧表示する
list-acls	アプリケーションサーバーインスタンスの ACL を一覧表示する
list-authdbs	認証データベースを一覧表示する
list-auth-realms	認証レルムを一覧表示する
list-components	サーバーインスタンスに配備されたコンポーネントを一覧表示する
list-custom-resources	サーバーインスタンスのカスタムリソースを一覧表示する
list-domains	ドメインを一覧表示する
list-file-users	サーバーインスタンスのすべてのファイルレルムユーザーを一覧表示する
list-file-groups	指定されたファイルレルムユーザーのすべてのグループを一覧表示する。ユーザーを指定しない場合は、サーバーインスタンスのすべてのグループが一覧表示される
list-http-listeners	サーバーインスタンスの HTTP リスナーを一覧表示する
list-instances	ドメイン内のアプリケーションサーバーインスタンスを一覧表示する

asadmin コマンド (続き)

コマンド	用途
list-iiop-listeners	サーバーインスタンスの IIOP リスナーを一覧表示する
list-javamail-resources	サーバーインスタンスの JavaMail リソースを一覧表示する
list-jdbc-connection-pools	サーバーインスタンスの JDBC 接続プールを一覧表示する
list-jdbc-resources	サーバーインスタンスの JDBC リソースを一覧表示する
list-jmsdest	サーバーインスタンスの JMS 送信先を一覧表示する
list-jms-resources	サーバーインスタンスの JMS リソースを一覧表示する
list-jndi-resources	サーバーインスタンスの JNDI リソースを一覧表示する
list-lifecycle-modules	サーバーインスタンスのライフサイクルモジュールを一覧表示する
list-mimes	サーバーインスタンスの MIME タイプファイルを一覧表示する
list-persistence-resources	サーバーインスタンスの持続マネージャファクトリリソースを一覧表示する
list-profilers	サーバーインスタンスの JVM プロファイラを一覧表示する
list-sub-components	配備されたモジュール内、または配備されたアプリケーションのモジュール内の 1 つ以上の EJB またはサーブレットを一覧表示する
list-virtual-servers	サーバーインスタンスの仮想サーバーを一覧表示する
multimode コマンド	環境設定を保持しながら asadmin 内で複数コマンドの実行を可能とする
reconfig	サーバーに変更を適用する。ほとんどの変更は、適用されるまで有効にならない
restart-instance	サーバインスタンスを再起動する
set	属性の値を設定する
show-component-status	配備されたコンポーネントの状態を表示する

asadmin コマンド (続き)

コマンド	用途
show-instance-status	サーバーインスタンスの状態を表示する (実行中かどうか)
shutdown	管理サーバーを停止する
start-appserv	管理サーバーとすべてのサーバーインスタンスを起動する。このコマンドは、ローカルでのみ実行可能
start-domain	ドメイン内のすべてのインスタンスを起動する。このコマンドは、ローカルでのみ実行可能
start-instance	サーバーインスタンスを起動する
stop-appserv	管理サーバーとすべてのサーバーインスタンスを停止する。このコマンドは、ローカルでのみ実行可能
stop-domain	ドメイン内のすべてのインスタンスを停止する
stop-instance	サーバーインスタンスを停止する
undeploy	配備されたコンポーネントをサーバーインスタンスから削除する
unset	asadmin にエクスポートされた環境変数の設定を解除する
update-file-user	既存のファイルレルムユーザーを更新する
version	Sun ONE Application Server のバージョン情報を表示する

ドット表記名と属性の一覧

get コマンドおよび set コマンドを使用して属性を取得および設定するときには、該当のオブジェクトの属性を取得するために、asadmin がサービスやリソースなどに使用する名前を知っておく必要があります。

これらの名前を使用する構文はピリオドとピリオドの間に区切る名前を含むため、これらの名前をドット表記名と呼びます。

asadmin で使用されるドット表記名

asadmin でアイテムを設定するために使用する名前の一覧を、以下の各表で示します。これらの名前は、次の各カテゴリに分類されます。

- サービス名
- リソース名
- アプリケーション名
- その他の名前

サービス名

次の表では、サービスの属性を取得および設定するために使用するサービス名を示します。

コマンド行インタフェースのサービス名

サービス	ドット表記名
JMS サービス設定	jms-service
トランザクションサービス設定	transaction-service
MDB コンテナ設定	mdb-container
EJB コンテナ設定	ejb-container
Web コンテナ設定	web-container
JVM 設定	java-config
ORB 設定	orb または iiop-service
ORB リスナー設定	orblistener または iiop-listener orblistener と iiop-listener は、そのままでは有効な名前ではありません。どちらもリスナー名を続けて記述する必要があります。次に例を示します。 ORB リスナー設定 orblistener.<listener name> または iiop-listener.<listener name>
ログ設定	log-service
セキュリティ設定	security-service
HTTP 設定	http-service

リソース名

次の表では、リソースの属性を取得および設定するために使用するリソース名を示します。これらの名前は、そのままでは有効ではありません。リソース名を続けて記述する必要があります。

コマンド行インタフェースのリソース名

リソース	ドット表記名
JDBC リソース設定	jdbc-resource
JNDI リソース設定	jndi-resource
JDBC 接続プールリソース設定	jdbc-connection-pool
カスタムリソース設定	custom-resource
JMS リソース設定	jms-resource
持続マネージャファクトリリソース設定	persistence-manager-factory-resource
Java メールサービスリソース設定	mail-resource

アプリケーション名

次の表では、アプリケーション関連設定の属性を取得および設定するために使用するドット表記名を示します。これらの名前は、そのままでは有効ではありません。アプリケーション名を続けて記述する必要があります。

コマンド行インタフェースのアプリケーション名

アプリケーションコンポーネント	ドット表記名
アプリケーション設定	application
EJB モジュール設定	ejb-module
Web モジュール設定	web-module
コネクタモジュール設定	connector-module

その他の名前

次の表では、`get` および `set` を使って設定できるその他のアイテムのドット表記名を示します。これらの名前は、そのままでは有効ではありません。アプリケーション名を続けて記述する必要があります。たとえば、`http-listener.`*`listener_name`*、`lifecycle-module.`*`module-name`* などです。

その他のコマンド行インタフェースのアイテム名

アイテム	ドット表記名
HTTP リスナー	http-listener または http-server.http-listener
MIME タイプファイル	mime
ACL	acl
仮想サーバー	virtual-server
認証データベース	auth-db
セキュリティレルム	authrealm
ライフサイクルモジュール	lifecycle-module
プロファイラ設定	profiler
サーバーインスタンス設定	サーバー設定 (サーバーインスタンス名)

属性

ここでは、前述の各名前付きアイテムの属性と、その使用例を示します。一部の属性は読み取り専用です。つまり、`get` コマンドだけで使用でき、`set` コマンドでは使用できない属性があります。

注 ここの例では、ユーザー、パスワード、ホスト、およびポートが環境変数によって定義されていることを前提としているため、それらのオプションは構文中に記述されていません。

jms-service

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

JMS サービスの属性

server.xml 名	asadmin 名
port	port
admin-username	adminUserName
admin-password	adminPassword
log-level	logLevel

JMS サービスの属性 (続き)

server.xml 名	asadmin 名
enabled	enabled
init-timeout-in-seconds	initTimeoutInSeconds
start-args	startArgs

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.jms-service.*
```

adminPassword という属性を取得するには、次のように入力します。

```
asadmin> get server1.jms-service.adminPassword
```

adminPassword という属性に値 admin を設定するには、次のように入力します。

```
asadmin> set server1.jms-service.adminPassword=admin
```

transaction-service

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

トランザクションサービスの属性

server.xml 名	asadmin 名
automatic-recovery	automaticTransactionRecovery
timeout-in-seconds	transactionRecoveryTimeout
tx-log-dir	transactionLogFile
heuristic-decision	heuristicDecision
keypoint-interval	keypointInterval
log-level	logLevel
monitoring-enabled	monitoringEnabled

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.transaction-service.*
```

transactionRecoveryTimeout という属性を取得するには、次のように入力します。

```
asadmin> get server1.transaction-service.transactionRecoveryTimeout
```

transactionRecoveryTimeout という属性に値 49 を設定するには、次のように入力します。

```
asadmin> set
server1.transaction-service.transactionRecoveryTimeout=49
```

mdb-container

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

MDB コンテナの属性

server.xml 名	asadmin 名
steady-pool-size	steadyPoolSize
pool-resize-quantity	poolResizeQuantity
max-pool-size	maxPoolSize
idle-timeout-in-seconds	idleInPoolTimeoutInSeconds
log-level	logLevel
monitoring-enabled	monitoringEnabled

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.mdb-container.*
```

steadyPoolSize という属性を取得するには、次のように入力します。

```
asadmin> get server1.mdb-container.steadyPoolSize
```

steadyPoolSize という属性に値 10 を設定するには、次のように入力します。

```
asadmin> set server1.mdb-container.steadyPoolSize=10
```

ejb-container

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

EJB コンテナの属性

server.xml 名	asadmin 名
steady-pool-size	steadyPoolSize
pool-resize-quantity	poolResizeQuantity
max-pool-size	maxPoolSize
cache-resize-quantity	cacheResizeQuantity
max-cache-size	maxCacheSize
pool-idle-timeout-in-seconds	idleInPoolTimeoutInSeconds
cache-idle-timeout-in-seconds	idleInCacheTimeoutInSeconds
removal-timeout-in-seconds	removalTimeoutInSeconds
victim-selection-policy	victimSelectionPolicy
commit-option	commitOption
log-level	logLevel
monitoring-enabled	monitoringEnabled

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.ejb-container.*
```

`maxPoolSize` という属性を取得するには、次のように入力します。

```
asadmin> get server1.ejb-container.maxPoolSize
```

`maxPoolSize` という属性に値 12 を設定するには、次のように入力します。

```
asadmin> set server1.ejb-container.maxPoolSize=12
```

web-container

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

Web コンテナの属性

server.xml 名	asadmin 名
log-level	logLevel
monitoring-enabled	monitoringEnabled (使用されない)

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.web-container.*
```

logLevel という属性を取得するには、次のように入力します。

```
asadmin> get server1.web-container.logLevel
```

monitoringEnabled という属性に WARNING を設定するには、次のように入力します。

```
asadmin> set server1.web-container.logLevel=WARNING
```

java-config

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

JVM の属性

server.xml 名	asadmin 名
java-home	javahome
debug-enabled	debugEnabled
debug-options	debugOptions
javac-options	javacoptions
rmic-options	rmicoptions
classpath-prefix	classpathprefix
server-classpath	serverClasspath
classpath-suffix	classpathsuffix
native-library-path-prefix	libpathprefix
native-library-path-suffix	libpathsuffix
env-classpath-ignored	envpathignore

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.java-config.*
```

classpathprefix という属性を取得するには、次のように入力します。

```
asadmin> get server1.java-config.classpathprefix
```

classpathprefix という属性に値 com.sun を設定するには、次のように入力します。

```
asadmin> set server1.java-config.classpathprefix=com.sun
```

orb または iiop-service

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

ORB/IIOP サービスの属性

server.xml 名	asadmin 名
message-fragment-size	msgSize
steady-thread-pool-size	minThreads
max-thread-pool-size	maxThreads
max-connections	maxConnections
idle-thread-timeout-in-seconds	idleThreadTimeout
log-level	log
monitoring-enabled	monitor
cert-nickname	cert
ssl2-enabled	ssl2
ssl2-ciphers	ssl2Ciphers
ssl3-enabled	ssl3
ssl3-tls-ciphers	ssl3Ciphers
tls-enabled	tls
tls-rollback-enabled	tlsRollback
client-auth-enabled	clientAuth

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.orb.*
```

または

```
asadmin> get server1.iiop-service.*
```

msgSize という属性を取得するには、次のように入力します。

```
asadmin> get server1.orb.msgSize
```

または

```
asadmin> get server1.iiop-service.msgSize
```

idleThreadTimeout という属性に値 300 を設定するには、次のように入力します。

```
asadmin> set server1.orb.idleThreadTimeout=300
```

または

```
asadmin> set server1.iiop-service.idleThreadTimeout=300
```

orblistener または iiop-listener

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

IIOP リスナーの属性

server.xml 名	asadmin 名
id	id
address	address
port	port
enabled	enabled
cert-nickname	cert
ssl2-enabled	ssl2
ssl2-ciphers	ssl2Ciphers
ssl3-enabled	ssl3
ssl3-tls-ciphers	ssl3Ciphers
tls-enabled	tls
tls-rollback-enabled	tlsRollback
client-auth-enabled	clientAuth

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.orblistener.orb_listener_id.*
```

または

```
asadmin> get server1.iiop-listener.orb_listener_id.*
```

port という属性を取得するには、次のように入力します。

```
asadmin> get server1.orblistener.orb_listener_id.port
```

または

```
asadmin> get server1.iiop-listener.orb_listener_id.port
```

address という属性に bluestar を設定するには、次のように入力します。

```
asadmin> set server1.orblistener.orb_listener_id.address=bluestar
```

または

```
asadmin> set server1.iiop-listener.orb_listener_id.address=bluestar
```

log-service

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

ログサービスの属性

server.xml 名	asadmin 名
file	file
level	level
log-stdout	stdout
log-stderr	stderr
echo-log-messages-to-stderr	echoToStderr
create-console	createConsole
log-virtual-server-id	LogVirtualServerId
use-system-logging	useSystemLogging

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.log-service.*
```

level という属性を取得するには、次のように入力します。

```
asadmin> get server1.log-service.level
```

echoToStderr という属性に true を設定するには、次のように入力します。

```
asadmin> set server1.log-service.echoToStderr=true
```

security-service

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

セキュリティレルム設定の属性

server.xml 名	asadmin 名
default-realm	defaultRealm
default-principal	defaultPrinicpal
default-principal-password	defaultPrinicpalPassword
anonymous-role	anonymousRole
audit-enabled	auditEnabled
log-level	logLevel

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.security-service.*
```

`anonymousRole` という属性を取得するには、次のように入力します。

```
asadmin> get server1.security-service.anonymousRole
```

`encryptPasswords` という属性に true を設定するには、次のように入力します。

```
asadmin> set server1.security-service.auditEnabled=true
```

http-service

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

HTTP サービスの属性

server.xml 名	asadmin 名
qos-metrics-interval-in-seconds	qos-metrics-interval-in-seconds
qos-recompute-time-interval-in-millis	qos-recompute-time-interval-in-millis
qos-enabled	qos-enabled
bandwidth-limit	bandwidthLimit
enforce-bandwidth-limit	enforceBandwidthLimit

HTTP サービスの属性 (続き)

server.xml 名	asadmin 名
connection-limit	connectionLimit
enforce-connection-limit	enforceConnectionLimit

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.http-service.*
```

bandwidthLimit という属性を取得するには、次のように入力します。

```
asadmin> get server1.http-service.bandwidthLimit
```

qos-enabled という属性に true を設定するには、次のように入力します。

```
asadmin> set server1.http-service.qos-enabled=true
```

jdbc-resource

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

JDBC リソースの属性

server.xml 名	asadmin 名
jndi-name	name
pool-name	pool
enabled	enabled
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.jdbc-resource.jdbc_resource_name.*
```

pool という属性を取得するには、次のように入力します。

```
asadmin> get server1.jdbc-resource.jdbc_resource_name.pool
```

enabled という属性に true を設定するには、次のように入力します。

```
asadmin> set server1.jdbc-resource.jdbc_resource_name.enabled=true
```

jndi-resource

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

JNDI リソースの属性

server.xml 名	asadmin 名
jndi-name	name
jndi-lookup-name	LookupName
res-type	resType
factory-class	factory
enabled	enabled
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.jndi-resource.jndi_name.*
```

factory という属性を取得するには、次のように入力します。

```
asadmin> get server1.jndi-resource.jndi_name.factory
```

factory という属性に com.sun を設定するには、次のように入力します。

```
asadmin> set server1.jndi-resource.jndi_name.factory=com.sun
```

jdbc-connection-pool

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

JDBC 接続プールの属性

server.xml 名	asadmin 名
name	name
datasource-classname	dsClassName
res-type	resType
description	description
steady-pool-size	steadyPoolSize

JDBC 接続プールの属性 (続き)

server.xml 名	asadmin 名
max-pool-size	maxPoolSize
max-wait-time-in-millis	maxWaitTime
pool-resize-quantity	resizeValue
idle-timeout-in-seconds	idleTimeout
transaction-isolation-level	transactionIsolationLevel
is-isolation-level-guaranteed	isIsolationLevelGuaranteed
connection-validation-method	validationMethod
is-connection-validation-required	isValidationRequired
fail-all-connections	failAll
validation-table-name	validationTable

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.jdbc-connection-pool.pool_name.*
```

`dsClassName` という属性を取得するには、次のように入力します。

```
asadmin> get server1.jdbc-connection-pool.pool_name.dsClassName
```

`resizeValue` という属性に値 2 を設定するには、次のように入力します。

```
asadmin> set server1.jdbc-connection-pool.pool_name.resizeValue=2
```

custom-resource

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

カスタムリソースの属性

server.xml 名	asadmin 名
jndi-name	name
res-type	resType
factory-class	factory
enabled	enabled
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.custom-resource.jndi_name.*
```

`factory` という属性を取得するには、次のように入力します。

```
asadmin> get server1.custom-resource.jndi_name.factory
```

`factory` という属性を設定するには、次のように入力します。

```
asadmin> set server1.custom-resource.jndi_name.factory=myclass
```

jms-resource

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

JMS リソースの属性

server.xml 名	asadmin 名
jndi-name	name
res-type	resType
enabled	enabled
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.jms-resource.jms_resource_name.*
```

`res-type` という属性を取得するには、次のように入力します。

```
asadmin> get server1.jms-resource.jms_resource_name.resType
```

`enabled` という属性に true を設定するには、次のように入力します。

```
asadmin> set server1.jms-resource.jms_resource_name.enabled=true
```

persistence-manager-factory-resource

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

持続マネージャファクトリリソースの属性

server.xml 名	asadmin 名
jndi-name	jndiName

持続マネージャファクトリリソースの属性

server.xml 名	asadmin 名
jdbc-resource-jndi-name	JdbcResourceJndiName
factory-class	factoryClass
enabled	enabled
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.persistence-manager-factory-resource.jndi_name
```

`factoryClass` という属性を取得するには、次のように入力します。

```
asadmin> get
server1.persistence-manager-factory-resource.jndi_name.factoryClass
```

`enabled` という属性に true を設定するには、次のように入力します。

```
asadmin> set
server1.persistence-manager-factory-resource.jndi_name.enabled=true
```

mail-resource

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

Java メールリソースの属性

server.xml 名	asadmin 名
jndi-name	name
enabled	enabled
store-protocol	storeProtocol
store-protocol-class	storeProtocolClass
transport-protocol	transportProtocol
transport-protocol-class	transportProtocolClass
host	host
user	user
from	from
debug	debug

Java メールリソースの属性 (続き)

server.xml 名	asadmin 名
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.mail-resource.jndi_name.*
```

host という属性を取得するには、次のように入力します。

```
asadmin> get server1.mail-resource.jndi_name.host
```

enabled という属性に true を設定するには、次のように入力します。

```
asadmin> set server1.mail-resource.jndi_name.enabled=true
```

application

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

アプリケーションの属性

server.xml 名	asadmin 名
name	name
location	location
virtual-servers	virtualServers
description	description
enabled	enabled

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.application.application_name.*
```

アプリケーションの location という属性を取得するには、次のように入力します。

```
asadmin> get server1.application.application_name.location
```

location という属性を設定するには、次のように入力します。

```
asadmin> set server1.application.application_name.location=
"/export/home/as7se/as1/repository/applications/ASConverter"
```

ejb-module

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

EJB モジュールの属性

server.xml 名	asadmin 名
name	name
location	location
description	description
enabled	enabled

インスタンス (server1) 内にあるスタンドアロン EJB モジュールのすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.ejb-module.ejb_jar_name.*
```

インスタンス (server1) に関する、アプリケーション内にある EJB モジュールのすべての属性を取得するには、次のように入力します。

```
asadmin> get
server1.j2ee-application.application_name.ejb-module.ejb_jar_name.*
```

または

```
asadmin>get server1.application.application_name.ejb-module.ejb_jar_name.*
```

スタンドアロン EJB モジュールから location という属性を取得するには、次のように入力します。

```
asadmin> get server1.ejb-module.ejb_jar_name.location
```

アプリケーションの EJB モジュールから location という属性を取得するには、次のように入力します。

```
asadmin> get
server1.j2ee-application.application_name.ejb-module.ejb_jar_name.
location
```

または

```
asadmin> get
server1.application.application_name.ejb-module.ejb_jar_name.location
```

スタンドアロン EJB モジュールの location という属性を設定するには、次のように入力します。

```
asadmin> set
server1.ejb-module.ejb_jar_name.location="/export/home/as7se/as1/repos
itory/modules/ejb_jar_name"
```

アプリケーションにバンドルされている EJB モジュールの location という属性を設定するには、次のように入力します。

```
asadmin> set
server1.j2ee-application.application_name.ejb-module.ejb_jar_name.
location="/export/home/as7se/as1/repository/modules/ejb_jar_name"
```

または

```
asadmin>set
server1.application.application_name.ejb-module.ejb_jar_name.location="/ex
port/home/as7se/as1/repository/modules/ejb_jar_name"
```

web-module

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

Web モジュールの属性

server.xml 名	asadmin 名
name	name
location	location
context-root	contextRoot
virtual-servers	virtualServers
description	description
enabled	enabled

インスタンス (server1) 内にあるスタンドアロン Web モジュールのすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.web-module.web_war_name.*
```

インスタンス (server1) に関する、アプリケーション内にある Web モジュールのすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.web-module.application_name.web_war_name.*
```

スタンドアロン Web モジュールから location という属性を取得するには、次のように入力します。

```
asadmin> get server1.web-module.web_war_name.location
```

アプリケーションの Web モジュールから location という属性を取得するには、次のように入力します。

```
asadmin> get server1.web-module.application_name.web_war_name.location
```

スタンドアロン Web モジュールの location という属性を設定するには、次のように入力します。

```
asadmin> set server1.web-module.war-ic.location=
"/export/home/as7se/as1/repository/modules/web_war_name"
```

アプリケーションにバンドルされている Web モジュールの location という属性を設定するには、次のように入力します。

```
asadmin> set server1.web-module.application_name.web_war_name.location=
"/export/home/as7se/as1/repository/modules/web_war_name"
```

connector-module

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

コネクタモジュールの属性

server.xml 名	asadmin 名
name	name
location	location
description	description
enabled	enabled

インスタンス (server1) 内にあるスタンドアロンコネクタモジュールのすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.connector-module.connector_rar_name.*
```

スタンドアロンコネクタモジュールから location という属性を取得するには、次のように入力します。

```
asadmin> get server1.connector-module.connector_rar_name.location
```

スタンドアロンコネクタモジュールの location という属性を設定するには、次のように入力します。

```
asadmin> set server1.connector-module.connector_rar_name.location=
"/export/home/as7se/as1/repository/modules/connector_rar_name"
```

http-listener または http-server.http-listener

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

HTTP リスナーの属性

server.xml 名	asadmin 名
id	id
address	address
port	port
family	family
acceptor-threads	acceptorThreads
blocking-enabled	blockingEnabled
security-enabled	securityEnabled
default-virtual-server	defaultVirtualServer
server-name	serverName
enabled	enabled
cert-nickname	cert
ssl2-enabled	ssl2
ssl2-ciphers	ssl2Ciphers
ssl3-enabled	ssl3
ssl3-tls-ciphers	ssl3Ciphers
tls-enabled	tls
tls-rollback-enabled	tlsRollback
client-auth-enabled	clientAuth

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.http-listener.http_listener_name.*
```

または

```
asadmin> get server1.http-server.http-listener.http_listener_name.*
```

`factory` という属性を取得するには、次のように入力します。

```
asadmin> get server1.httpl-listener.http_listener_name.address
```

または

```
asadmin> get server1.http-server.http-listener.http_listener_name.address
```

address という属性に IP アドレス 0.0.0.0 を設定するには、次のように入力します。

```
asadmin> set server1.http-listener.http_listener_name.address=0.0.0.0
```

または

```
asadmin> set
server1.http-server.http-listener.http_listener_name.address=0.0.0.0
```

mime

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

MIME タイプの属性

server.xml 名	asadmin 名
id	id
file	file

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.mime.mime_name.*
```

file という属性を取得するには、次のように入力します。

```
asadmin> get server1.mime.mime_name.file
```

file という属性に mime.types を設定するには、次のように入力します。

```
asadmin> set server1.mime.mime_name.file=mime.types
```

acl

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

ACL の属性

server.xml 名	asadmin 名
id	id

ACL の属性 (続き)

server.xml 名	asadmin 名
file	file

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.acl.acl_name.*
```

`file` という属性を取得するには、次のように入力します。

```
asadmin> get server1.acl.acl_name.file
```

`file` という属性を設定するには、次のように入力します。

```
asadmin> set server1.acl.acl_name.file=com/as1.acl
```

virtual-server

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

仮想サーバーの属性

server.xml 名	asadmin 名
id	id
http-listeners	httpListeners
config-file	configFile
default-object	defaultObject
accept-language	acceptLanguage
log-file	logFile
default-web-module	defaultWebModule
hosts	hosts
mime	mime
state	state
acls	acls
bandwidth-limit	bandwidthLimit
enforce-bandwidth-limit	enforceBandwidthLimit
connection-limit	connectionLimit

仮想サーバーの属性 (続き)

server.xml 名	asadmin 名
enforce-connection-limit	enforceConnectionLimit
property name="dir" value=	property.dir
property name="nice" value=	property.nice
property name="user" value=	property.user
property name="group" value=	property.group
property name="chroot" value=	property.chroot
property name="docroot" value=	property.docroot
property name="accesslog" value=	property.accesslog

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get instance_name.virtual-server.vserver_id.*
```

次に例を示します。

```
asadmin> get server1.virtual-server.server1.*
```

仮想サーバー server1 の `httpListeners` という属性を取得するには、次のように入力します。

```
asadmin> get server1.virtual-server.server1.httpListeners
```

`acceptLanguage` という属性に false を設定するには、次のように入力します。

```
asadmin> set server1.virtual-acceptLanguage=false
```

auth-db

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

認証データベースの属性

server.xml 名	asadmin 名
id	id
database	database
basedn	basedn
certmaps	certmaps

インスタンスのすべての属性を確認するときは、次のコマンドを使います。

```
asadmin> get instancename.virtual-server.vserver_id.auth-db.authdb_id.*
```

たとえば、インスタンス server1 の仮想サーバー server1 では、次のように入力します。

```
asadmin> get server1.virtual-server.server1.auth-db.authdb_id.*
```

database という属性を取得するには、次のように入力します。

```
asadmin> get server1.virtual-server.server1.auth-db.authdb_id.database
```

database という属性を設定するには、次のように入力します。

```
asadmin> set
server1.virtual-server.server1.auth-db.authdb_id.database=Oracle
```

authrealm

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

承認レルムの属性

server.xml 名	asadmin 名
name	name
classname	classname

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.authrealm.authrealm_id.*
```

classname という属性を取得するには、次のように入力します。

```
asadmin> get server1.authrealm.authrealm_id.classname
```

classname という属性を設定するには、次のように入力します。

```
asadmin> set
server1.authrealm.authrealm_id.classname=com.sun.as.security.auth.real
m.sharedpassword.SharedPasswordRealm
```

lifecycle-module

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

ライフサイクルモジュールの属性

server.xml 名	asadmin 名
name	name
enabled	enabled
class-name	className
classpath	classPath
load-order	loadOrder
is-failure-fatal	isFailureFatal
description	description

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.lifecycle-module.lifecycle_module_id.*
```

ライフサイクルモジュールの `className` という属性を取得するには、次のように入力します。

```
asadmin> get server1.lifecycle-module.lifecycle_module_id.className
```

`className` という属性を設定するには、次のように入力します。

```
asadmin> set
server1.lifecycle-module.lifecycle_module_id.className=com.lifecycle_module_id.
lifecycle
```

profiler

次の表では、属性の `server.xml` 名を左側の列に、`asadmin` で使用する名前を右側の列に示しています。

JVM プロファイラ設定の属性

server.xml 名	asadmin 名
name	name
classpath	classPath
native-library-path	nativeLibraryPath
enabled	enabled

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.profiler.*
```

enabled という属性を取得するには、次のように入力します。

```
asadmin> get server1.profiler.enabled
```

enabled という属性に false を設定するには、次のように入力します。

```
asadmin> set server1.profiler.enabled=false
```

サーバー設定 (サーバーインスタンス名)

次の表では、属性の server.xml 名を左側の列に、asadmin で使用する名前を右側の列に示しています。

サーバー設定の属性

server.xml 名	asadmin 名
instance-name	name
locale	locale
log-root	logRoot
session-store	sessionStore
application-root	applicationRoot
dynamic-reload-enabled	appDynamicReloadEnabled
dynamic-reload-poll-interval-in-seconds	appReloadPollInterval

インスタンス (server1) からすべての属性を取得するには、次のように入力します。

```
asadmin> get server1.*
```

logRoot という属性を取得するには、次のように入力します。

```
asadmin> get server1.logRoot
```

logRoot という属性を設定するには、次のように入力します。

```
asadmin> set server1.logRoot="/space/log"
```

各オプションに対応する長形式、短形式、デフォルト値、および環境変数

次の表に、コマンド行オプションの長形式と短形式を示します。短形式が記載されていない場合、短形式のオプションは使用できません。

各オプションに対応する短形式、長形式、デフォルト値、および環境変数

オプション名	長形式	短形式	デフォルト値	環境変数
acceptlang	--acceptlang			AS_ADMIN_ACCEPT_
acceptorthreads	--acceptorthreads			AS_ADMIN_ACCEPTOR_THREADS
acls	--acls			AS_ADMIN_ACLS
address	--address			AS_ADMIN_ADDRESS
adminpassword	--adminpassword			AS_ADMIN_ADMINPASSWD
adminport	--adminport		4848	AS_ADMIN_ADMINPORT
adminuser	--adminuser			AS_ADMIN_ADMINUSER
basedn	--basedn			AS_ADMIN_BASEDN
blockingenabled	--blockingenabled			AS_ADMIN_BLOCKINGENABLED
bwlimit	--bwlimit			AS_ADMIN_BWLIMIT
certmaps	--certmaps			AS_ADMIN_CERTMAPS
certname	--certname			AS_ADMIN_CERTNAME
classname	--classname			AS_ADMIN_CLASSNAME
classpath	--classpath			AS_ADMIN_CLASSPATH
clientauthenabled	--clientauthenabled			AS_ADMIN_CLIENTAUTHENABLED
configfile	--configfile			AS_ADMIN_CONFIGFILE
connectionpoolid	--connectionpoolid			AS_ADMIN_CONNECTIONPOOLID
connlimit	--connlimit			AS_ADMIN_CONNLIMIT
contextroot	--contextroot			AS_ADMIN_CONTEXTROOT
database	--database			AS_ADMIN_DATABASE
debug	--debug		false	AS_ADMIN_DEBUG

各オプションに対応する短形式、長形式、デフォルト値、および環境変数 (続き)

オプション名	長形式	短形式	デフォルト値	環境変数
defaultobj	--defaultobj			AS_ADMIN_DEFAULTOBJ
defaultwebmodule	--defaultwebmodule			AS_ADMIN_DEFAULTWEBMODULE
description	--description			AS_ADMIN_DESCRIPTION
discardmanualchanges	--discardmanualchanges	-d	false	AS_ADMIN_DISCARDMANUALCHANGES
echo	--echo	-e	false	AS_ADMIN_ECHO
enabled	--enabled			AS_ADMIN_ENABLED
enforcebwlimit	--enforcebwlimit			AS_ADMIN_ENFORCEBWLIMIT
enforceconnlimit	--enforceconnlimit			AS_ADMIN_ENFORCECONNLIMIT
failconnection	--failconnection		false	AS_ADMIN_FAILCONNECTION
failurefatal	--failurefatal		false	AS_ADMIN_FAILUREFATAL
family	--family			AS_ADMIN_FAMILY
file	--file	-f		AS_ADMIN_FILE
force	--force	-F	true	AS_ADMIN_FORCE
help	--help	-h		AS_ADMIN_HELP
host	--host	-H		AS_ADMIN_HOST
hosts	--hosts			AS_ADMIN_HOSTS
httplistenerid	--httplistenerid			AS_ADMIN_HTTPLISTENERID
httplisteners	--httplisteners			AS_HTTP_LISTENERS
idletimeout	--idletimeout		300	AS_ADMIN_IDLETIMEOUT
instance	--instance	-i	server1	AS_ADMIN_INSTANCE
instanceport	--instanceport			AS_ADMIN_INSTANCEPORT
interactive	--interactive	-I	true	AS_AMDIN_INTERACTIVE
isconnectvalidaterequired	--isconnectvalidaterequired		false	AS_ADMIN_ISCONNECTVALIDATEREQUIRED
jdbcjndiname	--jdbcjndiname	-a		AS_ADMIN_JDBCJNDINAME
jndilookupname	--jndilookupname	-l		AS_ADMIN_JNDILOOKUPNAME

各オプションに対応する短形式、長形式、デフォルト値、および環境変数 (続き)

オプション名	長形式	短形式	デフォルト値	環境変数
keepmanualchanges	--keepmanualchanges	-k	false	AS_ADMIN_KEEPMANUALCHANGES
loadorder	--loadorder			AS_ADMIN_LOADORDER
local	--local	-l	false	
logfile	--logfile			AS_ADMIN_LOGFILE
maxpoolsize	--maxpoolsize		32	AS_ADMIN_MAXPOOLSIZE
maxwait	--maxwait		6000	AS_ADMIN_MAXWAIT
mime	--mime			AS_ADMIN_MIME
mimefile	--mimefile			AS_ADMIN_MIMEFILE
monitor	--monitor	-m	false	AS_ADMIN_MONITOR
name	--name	-n		AS_ADMIN_NAME
nativelibpath	--nativelibpath			AS_ADMIN_NATIVELIBPATH
objtype	--objtype	-o		AS_ADMIN_OBJTYPE
password	--password	-w		AS_ADMIN_PASSWORD
poolresize	--poolresize		2	AS_ADMIN_POOLRESIZE
port	--port	-p	8000	AS_ADMIN_PORT
prefix	--prefix	-x		AS_ADMIN_PREFIX
printprompt	--printprompt	-P	true	AS_ADMIN_PROMPT
property	--property			AS_ADMIN_PROPERTY
securityenabled	--securityenabled			AS_ADMIN_SECURITYENABLED
servername	--servername			AS_ADMIN_SERVERNAME
ssl2ciphers	--ssl2ciphers			AS_ADMIN_SSL2CIPHERS
ssl2enabled	--ssl2enabled			AS_ADMIN_SSL2ENABLED
ssl3enabled	--ssl3enabled			AS_ADMIN_SSL3ENABLED
ssl3tlsciphers	--ssl3tlsciphers			AS_ADMIN_SSL3TLSCIPHERS
state	--state			AS_ADMIN_STATE
steadypoolsize	--steadypoolsize	8		AS_ADMIN_STEADYPOOLSIZE
storeprotocol	--storeprotocol			AS_ADMIN_STOREPROTOCOL

各オプションに対応する短形式、長形式、デフォルト値、および環境変数 (続き)

オプション名	長形式	短形式	デフォルト値	環境変数
storeprotocolclass	--storeprotocolclass			AS_ADMIN_STOREPROTOCOLCLASS
tlsenabled	--tlsenabled			AS_ADMIN_TLSENABLED
tlsrollbackenabled	--tlsrollbackenabled			AS_ADMIN_TLSROLLBACKENABLED
transprotocol	--transprotocol		smtp	AS_ADMIN_TRANSPROTOCOL
type	--type			S_ADMIN_TRANSPROTOCOLCLASS
upload	--upload	-U	true	AS_ADMIN_TYPE
url	--url			AS_ADMIN_URL
user	--user	-u		AS_ADMIN_USER
validationmethod	--validationmethod		auto-commit	AS_ADMIN_VALIDATIONMETHOD
validationtable	--validationtable			AS_ADMIN_VALIDATIONTABLE
version	--version	-v		AS_AMDIN_VERSION
virtualserver	--virtualserver			AS_ADMIN_VIRTUALSERVER

サードパーティ製品の著作権について

この製品には、RSA Security, Inc. からライセンスされたコードが含まれます。

この製品の一部は ANTLR を使って開発されました。ANTLR 1989-2000 は jGuru.com (http://www.ANTLR.org および http://www.jGuru.com) によって開発されました。

この製品には、Sun Public License のもとで Netbeans Project (http://www.netbeans.org) で開発されたソフトウェアが含まれます。これらのソフトウェアは、www.netbeans.org に公開されている可能性があります。

この製品には Perl が含まれます。Perl は、http://public.ActiveState.com/gsar/APC/ に公開されている可能性があります。

この製品には、Exolab Project (http://www.exolab.org) で開発されたソフトウェアが含まれます。

この製品には、DOM4J Project (http://dom4j.org/) で開発されたソフトウェアが含まれます。

この製品には、Apache Foundation が開発したソフトウェアが含まれます。Copyright (c) 1999-2001 The Apache Software Foundation. All rights reserved.

この製品には、The Regents of University of California が開発したソフトウェアが含まれます。Copyright (c) 1991, 1993 The Regents of University of California. All rights reserved.

この製品には、International Business Machines Corporation が開発したソフトウェアが含まれます。Copyright (c) 1995-2001 International Business Machines Corporation and others. All rights reserved. IBM コードは、ICU License に基づいて入手しました。以下を参照してください。

ICU License - ICU 1.8.1 以降

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2001 International Business Machines Corporation and others All rights reserved.

用語集

この用語集では、Sun ONE Application Server の配備および開発環境を説明するために使われる一般的な用語を定義します。標準 J2EE の用語については、次のサイトにある用語集を参照してください。

```
http://java.sun.com/j2ee/glossary.html
```

ACL　アクセス制御リスト (Access Control List)。Sun ONE Application Server に格納されているリソースにアクセスできるユーザーの ID リストを記録したテキストファイル。「汎用 ACL (general ACL)」も参照。

API　Application Program Interface の略。コンピュータプログラムが、API を解釈するために設計されたほかのソフトウェアまたはハードウェアと通信するために使われる命令の集まり。

Bean 管理による持続性 (bean-managed persistence)　エンティティ Bean の変数とデータストアの間で行われるデータ転送。通常、データアクセスロジックは、JDBC (Java Database Connectivity) またはそれ以外のデータアクセステクノロジを使って、開発者によって決定される。「コンテナ管理による持続性 (container-managed persistence)」も参照。

Bean 管理によるトランザクション (bean-managed transaction)　エンタープライズ Bean のトランザクション境界設定を開発者が記述したプログラムで制御する。「コンテナ管理によるトランザクション (container-managed transaction)」も参照。

BLOB　Binary Large OBject の略。複合オブジェクトフィールドの格納と取り出しに使うデータ型。BLOB は、画像などのバイナリまたは直列化可能なオブジェクトで、大きなバイト配列に変換された後、コンテナ管理による持続性フィールドに直列化される。

BMP　「Bean 管理による持続性 (bean-managed persistence)」を参照。

BMT　「Bean 管理によるトランザクション (bean-managed transaction)」を参照。

CA　「証明書発行局 (certificate authority)」または「コネクタアーキテクチャ (connector architecture)」を参照。

CKL　Compromised Key List の略。証明書発行局が発行するリスト。クライアントユーザーまたはサーバーユーザーが信頼しなくなった証明書を示す。この場合、鍵は信頼性がなくなっている。「CRL」も参照。

CLI　コマンド行インタフェース (Command-line interface)。ユーザープロンプトで実行型の命令を入力できるインタフェース。「管理インタフェース (Administration interface)」も参照。

CMP　「コンテナ管理による持続性 (container-managed persistence)」を参照。

CMR　「コンテナ管理による関係 (container-managed relationship)」を参照。

CMT　「コンテナ管理によるトランザクション (container-managed transaction)」を参照。

cookie　呼び出し側である Web ブラウザに対して送信され、その後、そのブラウザから呼び出しが行われるたびにブラウザ側に記録される情報の小さなコレクション。サーバーは、cookie によって、同じクライアントからの呼び出しであるかどうかを認識できる。cookie はドメイン特有である。cookie は、アプリケーションとサーバー間の、ほかのデータ交換の場合と同じ Web サーバーセキュリティ機能を利用できる。

CORBA　Common Object Request Broker Architecture の略。オブジェクト指向型分散コンピューティングでの標準的なアーキテクチャ定義。

COSNaming サービス (COSNaming Service)　IIOP ベースのネーミングサービス。

CosNaming プロバイダ (CosNaming provider)　グローバルな JNDI ネームスペースをサポートする (IIOP アプリケーションクライアントにアクセスできる) ために、Sun ONE Application Server には J2EE ベースの CosNaming プロバイダが含まれる。このプロバイダは、CORBA 参照 (リモート EJB 参照) のバインドをサポートする。

CRL　Certificate Revocation List の略。証明書発行局が発行するリスト。クライアントユーザーまたはサーバーユーザーが信頼しなくなった証明書を示す。この場合、証明書は無効になっている。「CKL」も参照。

DataSource オブジェクト (DataSource Object)　実際のデータソースを識別する一連のプロパティを持ったオブジェクト。

DN　識別名 (Distinguished Name)。ディレクトリサーバーのエントリ名を表す文字列。

DN 属性 (DN attribute)　識別名の属性。関連するユーザー、グループ、オブジェクトの識別情報を含むテキスト文字列。

DTD　ドキュメントタイプ定義 (Document Type Definition)。XML ファイルのクラスの構造とプロパティを記述したもの。

EAR ファイル (EAR file)　Enterprise ARchive ファイル。J2EE アプリケーションを含むアーカイブファイル。EAR ファイルの拡張子は `.ear`。「JAR ファイル (JAR file)」も参照。

EIS　Enterprise Information System の略。EIS は、パッケージ化された企業アプリケーション、トランザクションシステム、またはユーザーアプリケーションと言い換えることができる。通常は、EIS と呼ばれている。EIS の例には、R/3、PeopleSoft、Tuxedo、CICS などがある。

EJB QL　EJB クエリ言語 (EJB Query Language)。コンテナ管理の関係によって定義されるエンティティ Bean のネットワーク上を移動するためのクエリ言語。

EJB コンテナ (EJB container)　「コンテナ (container)」を参照。

EJB テクノロジ (EJB technology)　エンタープライズ Bean は、アプリケーションのビジネスロジックをカプセル化したサーバーサイドコンポーネントである。ビジネスロジックは、アプリケーションの目的をすべて含むコードである。たとえば、在庫管理アプリケーションでは、エンタープライズ Bean はビジネスロジックを `checkInventoryLevel` や `orderProduct` などのメソッドに実装する。これらのメソッドを呼び出すことで、クライアントはアプリケーションが提供する在庫サービスにアクセスできる。「コンテナ (container)」、「エンティティ Bean (entity bean)」、「メッセージ駆動型 Beans (message-driven bean)」、「セッション Bean (session bean)」も参照。

ejbc ユーティリティ (ejbc utility)　エンタープライズ Bean のコンパイラ。すべての EJB クラスとインタフェースが EJB 仕様に合っているかどうかを調べ、スタブとスケルトンを作成する。

ERP　Enterprise Resource Planning の略。企業のリソースの計画をサポートするマルチモジュールのソフトウェアシステム。通常、ERP システムには、購買、在庫、人事、顧客サービス、出荷、資金計画などのビジネスの重要な面を管理するためのリレーショナルデータベースおよびアプリケーションが含まれている。

finder メソッド (finder method)　クライアントがグローバルに利用可能なディレクトリで、Bean または Bean のコレクションを調べることができるようにするメソッド。

FQDN　完全指定のドメイン名 (Fully Qualified Domain Name)。システムの完全指定された名前で、ホスト名とドメイン名の両方を含む。

HTML　Hypertext Markup Language の略。Web ブラウザに表示できるドキュメントを記述するためのマークアップ言語。テキストの各ブロックは、テキストの種類を指定したコードで囲む。

HTML ページ (HTML page)　HTML でコード化され、Web ブラウザで表示することを目的としたページ。

HTTP　HyperText Transfer Protocol の略。リモートホストからハイパーテキストオブジェクトをフェッチするインターネットプロトコル。TCP/IP を基本としている。

HTTP サーブレット (HTTP servlet)　`javax.servlet.HttpServlet` を拡張するサーブレット。HTTP サーブレットには、HTTP プロトコルのサポートが組み込まれている。「汎用サーブレット (generic servlet)」と対照的。

HTTPS　HyperText Transmission Protocol, Secure の略。安全なトランザクション用の HTTP。

IDE　統合開発環境 (Integrated Development Environment)。1 つの使いやすいインタフェースでコードを作成、アセンブル、配備、およびデバッグするためのソフトウェア。

IIOP　Internet Inter-ORB Protocol の略。IIOP 経由の RMI (Remote Method Invocation) と CORBA (Common Object Request Broker Architecture) の両方で使用されるトランスポートレベルプロトコル。

IIOP リスナー (IIOP Listener)　特定のポートで待機して、CORBA ベースのクライアントアプリケーションから送信される接続を受け付ける待機ソケット。

IMAP　インターネットメッセージアクセスプロトコル (Internet Message Access Protocol)。

IP アドレス (IP address)　TCP/IP ネットワーク上のコンピュータまたはその他のデバイスを識別する構造化された数値 ID。IP アドレスの形式は、4 つの数値をピリオドで区切って記述される 32 ビットの数値アドレスである。各数値は 0 ～ 255 の範囲で指定できる。たとえば、123.231.32.2 は IP アドレスにできる。

J2EE　Java 2 Enterprise Edition の略。多層 Web ベースエンタープライズアプリケーションを開発し、配備するための環境。J2EE プラットフォームは、一連のサービス、アプリケーションプログラミングインタフェース (API)、およびこれらのアプリケーションを開発する機能を提供するプロトコルから構成されている。

JAF　JavaBeans Activation Framework の略。MIME データタイプのサポートを Java プラットフォームに統合する。「Mime タイプ」を参照。

JAR ファイル (JAR file)　Java ARchive ファイル。多数のファイルを 1 つのファイルに統合するためのファイル。JAR ファイルの拡張子は `.jar`。

JAR ファイル形式 (JAR file format)　Java ARchive ファイル形式。多数のファイルを 1 つのファイルに統合できるファイル形式で、プラットフォームに依存しない。複数のアプレットと必要なコンポーネント (クラスファイル、イメージ、サウンド、その他のリソースファイル) を JAR ファイルにまとめて、1 回の HTTP トランザクションでブラウザにダウンロードできる。JAR ファイル形式はファイルの圧縮とデジタルシグネチャもサポートしている。

JAR ファイルの規約 (JAR file contract)　エンタープライズ Bean パッケージに含める情報を指定する Java ARchive の規約。

Java IDL　Java インタフェース定義言語 (Java Interface Definition Language)。Java プログラミング言語で記述した API で、Common Object Request Broker Architecture (CORBA) との標準ベースの互換性と接続性を提供する。

JavaBean　移植可能でプラットフォームに依存しない、再利用できるコンポーネントモデル。

JavaMail セッション (JavaMail session)　メールストアとの通信でアプリケーションが使用するオブジェクト。アプリケーションコードは、JNDI 名を使う JavaMail セッションリソースを JNDI サービスを使って特定する。

JAX-RPC　XML ベースのリモートプロシージャ呼び出し用 Java API (Java API for XML-based Remote Procedure Calls)。開発者が、XML ベースの RPC プロトコルに基づいた相互利用可能な Web アプリケーションや Web サービスを作成できるようにする。

JAXM　Java API for XML Messaging の略。アプリケーションが、SOAP 標準を使って、ドキュメント指向の XML メッセージを送受信できるようにする。これらのメッセージにファイルが添付されていても構わない。

JAXP　Java API for XML Processing の略。DOM、SAX、および XSLT を使った XML ドキュメントの処理をサポートしている Java API。アプリケーションが、特定の XML 処理実装に依存せずに、XML ドキュメントを解析および変換できるようにする。

JAXR　Java API for XML Registry の略。さまざまな種類の XML レジストリにアクセスするための、統一された標準の Java API を提供する。ユーザーが、Web サービスを作成、配備、および検索できるようにする。

JDBC　Java Database Connectivity の略。開発者がデータ認識コンポーネントを作成するときに使う、標準ベースの一連のクラスおよびインタフェース。JDBC は、プラットフォームやベンダーとは無関係にデータソースと接続して対話するためのメソッドを実装する。

JDBC 接続プール (JDBC connection pool)　データベースへの接続を指定するための JDBC データソースのプロパティと接続プールのプロパティを組み合わせたプール。

JDBC リソース (JDBC resource)　アプリケーションサーバー上で稼動しているアプリケーションとデータベースを接続するリソースで、既存の JDBC 接続プールを使用する。JNDI 名 (アプリケーション側で使用) と既存の JDBC 接続プールの名前から構成される。

JDK　Java Development Kit の略。Java 2 より前のバージョンの Java プラットフォームに対応したアプリケーションの開発に必要な API やツールを含むソフトウェア。

JMS　Java Message Service の略。JMS クライアントが JMS メッセージサービスの機能にアクセスする方法を定義するインタフェースとセマンティックの標準セット。これらのインタフェースは、Java プログラムによるメッセージの作成、送信、受信、読み込みの標準の方法を提供する。

JMS 管理オブジェクト (JMS-administered object)　1 つまたは複数の JMS クライアントを使用できるように、管理者が作成した設定済みの JMS オブジェクト (接続ファクトリまたは送信先)。管理オブジェクトを使うことで、プロバイダごとに別の JMS クライアントを使用せずに、同じクライアントを使用できるようになる。管理者はこれらのオブジェクトを JNDI ネームスペースに保存し、JMS クライアントは JNDI ルックアップによってこれらのオブジェクトにアクセスする。

JMS クライアント (JMS client)　JMS メッセージサービスを使ってメッセージを交換する別の JMS クライアントと通信するアプリケーションまたはソフトウェアコンポーネント。

JMS サービス (JMS Service)　JMS クライアントとの接続、メッセージのルーティングと配信、持続性、セキュリティ、ログなど、JMS メッセージシステムの配信サービスを提供するソフトウェア。メッセージサービスは、JMS クライアントのメッセージ送信先、およびメッセージをコンシュームするクライアントに配信されるメッセージの送信元である物理的送信先を維持する。

JMS 接続ファクトリ (JMS connection factory)　JMS クライアントが JMS メッセージサービスとの接続に使用する JMS 管理オブジェクト。

JMS 送信先 (JMS destination)　JMS メッセージに含まれる物理的送信先。生成されたメッセージの、ルーティング先またはコンシューマへの配信先。この物理的送信先は、JMS 管理オブジェクトによって識別され、カプセル化される。JMS クライアントは、プロデュースするメッセージの配信先、コンシュームするメッセージの送信元、またはその両方を決定するときに、この JMS 管理オブジェクトを使用する。

JMS プロバイダ (JMS provider)　メッセージシステム用の JMS インタフェースを実装した製品。

JMS メッセージ (JMS messages)　JMS クライアントがコンシュームする非同期の要求、報告、またはイベント。メッセージにはヘッダーとボディがある (ヘッダーにはフィールドを追加できる)。メッセージヘッダーは、標準フィールドとオプションプロパティを指定する。メッセージボディには、転送するデータが含まれる。

JNDI　Java Naming and Directory Interface の略。企業の複数のネーミングサービスやディレクトリサービスに対する統一インタフェースを Java 技術が使用可能なアプリケーションに提供する、Java プラットフォームの標準拡張。Java Enterprise API セットの一部として、JNDI は、企業の異種ネーミングサービスおよび異種ディレクトリサービスへのシームレスな接続を可能にする。

JNDI 名 (JNDI name)　JNDI ネーミングサービスに登録されているリソースへのアクセスに使用する名前。

JRE　Java 実行時環境 (Java Runtime Environment)。Java 仮想マシンと Java コアクラスに加え、Java プログラミング言語で書かれたアプリケーションの実行時サポートを提供するファイルから構成された、Java Development Kit (JDK) のサブセット。「JDK」も参照。

JSP　JavaServer Pages の略。HTML または XML タグ、JSP タグ、および Java コードを組み合わせて記述したテキストページ。JSP はプログラミング言語の能力と標準ブラウザページのレイアウト機能をあわせ持つ。

jspc ユーティリティ (jspc utility)　JSP のコンパイラ。JSP 仕様に準拠しているかすべての JSP をチェックする。

JTA　Java Transaction API の略。アプリケーションおよび J2EE サーバーによるトランザクションへのアクセスを可能にする API。

JTS　Java Transaction Service の略。トランザクションを処理する Java サービス。

LDAP　Lightweight Directory Access Protocol の略。LDAP は、TCP/IP 上で実行するオープンディレクトリアクセスプロトコルである。グローバルなサイズおよび多数のエントリに拡張できる。アプリケーションサーバーにバンドルされている LDAP サーバーである、Sun ONE Directory Server を使うと、アプリケーションサーバーがネットワーク経由でアクセスできる 1 つの一元化されたディレクトリ情報リポジトリに社内情報をすべて保存できる。

LDIF　LDAP Data Interchange Format の略。Sun ONE Directory Server エントリをテキスト形式で表す形式。

MDB　「メッセージ駆動型 Beans (message-driven bean)」を参照。

MIME データタイプ (MIME Data Type)　MIME (Multi-purpose Internet Mail Extension) タイプを使って、ユーザーのシステムでサポートされるマルチメディアファイルのタイプを制御できる。

NTV　名前 (Name)、タイプ (Type)、値 (Value)。

O/R マッピングツール (O/R mapping tool)　Object-to-relational mapping tool の略。Sun ONE Application Server 管理インタフェースのマッピングツールで、Entity Beans の XML 配備記述子を作成する。

POP3　Post Office Protocol の略。

QOS　QOS (Quality of Service、サービス品質) は、サーバーインスタンス、または仮想サーバーなどに対して設定するパフォーマンスの制限である。たとえば、ISP は、許可する帯域幅に応じて仮想サーバーの課金額を変えたいことがある。この場合、帯域幅の量と接続数に制限を課すことができる。

RAR ファイル (RAR file)　Resource ARchive の略。リソースアダプタを持つ JAR アーカイブ。

RDB　リレーショナルデータベース。

RDBMS　リレーショナルデータベース管理システム。

ResultSet　`java.sql.ResultSet` インタフェースを実装するオブジェクト。`ResultSets` は、データベースまたはほかのソースの表形式データから取得した一連の行のカプセル化に使われる。

RMI　Remote Method Invocation の略。オブジェクトをリモートプロセスに渡せるようにリモートインタフェースを記述するための一連の Java 標準 API。

RMIC　Remote Method Invocation Compiler の略。

RowSet　データベースまたはほかのソースの表形式データから取得した一連の行をカプセル化するオブジェクト。`RowSet` は、`java.sql.ResultSet` インタフェースを拡張して、`ResultSet` が JavaBeans コンポーネントとして機能できるようにする。

RPC　Remote Procedure Call の略。リモートオブジェクトまたはサービスにアクセスするメカニズム。

SAF　Server Application Function の略。要求の処理やその他のサーバーアクティビティに関与する機能。

Secure Socket Layer　「SSL」を参照。

SMTP　Simple Mail Transport Protocol の略。

SNMP　Simple Network Management Protocol の略。ネットワークの稼動状況に関するデータを交換するために使用されるプロトコル。管理対象デバイスとネットワーク管理ステーション (NMS) 間のデータのやりとりは、SNMP によって行われる。SNMP を使用するすべてのデバイス (ネットワーク上のホスト、ルーター、Web サーバー、その他のサーバーなど) が管理の対象となる。NMS は、そのネットワークのリモート管理を行うマシンである。

SOAP　Simple Object Access Protoco の略。XML ベースのデータ構築と HTTP (Hyper Text Transfer Protocol) の組み合わせを使って、インターネットを介して多様なオペレーション環境に配布されたオブジェクト内のメソッドを呼び出すための標準的な方法を定義している。

SQL　Structured Query Language の略。リレーショナルデータベースアプリケーションで一般的に使用される言語。SQL2 および SQL3 は、この言語のバージョンを表す。

SSL　Secure Sockets Layer の略。インターネットで安全に通信できるようにするためのプロトコル。

Sun ONE Directory Server　Lightweight Directory Access Protocol (LDAP) の Sun ONE バージョン。Sun ONE Application Server の各インスタンスは、Sun ONE Directory Server を使ってユーザーおよびグループに関する情報などの共有サーバー情報を保存する。「LDAP」も参照。

Sun ONE Message Queue　JMS (Java Message Service) オープン標準を実装する Sun ONE エンタープライズメッセージングシステム。MQ は JMS プロバイダの 1 つである。

TLS　Transport Layer Security の略。トランスポート層で暗号化と証明書を提供するプロトコル。クライアントおよびサーバーアプリケーションに対して大きな変更を加える必要なく、データをセキュリティの保護されたチャンネル経由で送受信することができる。

UDDI　Universal Description, Discovery, and Integration の略。検索および統合用に、Web サービスのワールドワイドなレジストリを提供する。

URI　Uniform Resource Identifier の略。ドメインの固有リソースを記述する。ローカルではベースディレクトリのサブセットとして記述され、`/ham/burger` はベースディレクトリになり、URI は `toppings/cheese.html` を指定する。対応する URL は、`http://domain:port/toppings/cheese.html` となる。

URL　Uniform Resource Locator の略。HTML ページまたはほかのリソースを一意に指定するアドレス。Web ブラウザは URL を使って、表示するページを指定する。URL では、転送プロトコル (HTTP、FTP など)、ドメイン (`www.my-domain.com` など)、URI (オプション) などを記述する。

WAR ファイル (WAR file)　Web ARchive の略。Web モジュールを含む Java アーカイブ。WAR ファイルの拡張子は `.war`。

Web アプリケーション (web application)　サーブレット、JavaServer Pages、HTML ドキュメント、およびその他の Web リソース (イメージファイル、圧縮アーカイブなどのデータを含む) の集まり。Web アプリケーションは、アーカイブ (WAR ファイル) にパッケージされている場合や、オープンディレクトリ構造に配備されている場合がある。Sun ONE Application Server では、SHTML や CGI など、Java 以外の Web アプリケーションテクノロジもサポートしている。

Web キャッシュ (web cache)　Sun ONE Application Server の機能の 1 つ。パフォーマンスの向上のため、サーブレットまたは JSP がその結果を指定した一定の時間キャッシュすることを可能にする。その時間内にサーブレットまたは JSP を呼び出すと、キャッシュに保存された結果が返されるので、サーブレットまたは JSP を実行し直す必要がない。

Web コネクタプラグイン (web connector plug-in)　Sun ONE Application Server との通信を可能にする Web サーバーの拡張機能。

Web コンテナ (web container)　「コンテナ (container)」を参照。

Web サーバー (web server)　HTML ページと Web アプリケーションを格納、管理するホスト。完全な J2EE アプリケーションではない。Web サーバーは、Web ブラウザからのユーザーリクエストに応答する。

Web サーバープラグイン (Web Server Plugin)　HTTP リバースプロキシプラグイン。これを使って、ユーザーから Sun ONE Web Server または Sun ONE Application Server に指示を送り、特定の HTTP 要求を別のサーバーへ転送することができる。

Web サービス (web service)　Web 経由で提供されるサービス。インターネットまたはイントラネットを経由してシステムからの要求を受け入れ、それを処理し、応答を返す、完全な自己記述式のモジュラーアプリケーション。

Web モジュール (web module)　個別に配備された Web アプリケーション。「Web アプリケーション (web application)」を参照。

WSDL　Web Service Description Language の略。標準化された方法で Web サービスを定義するために使用される、XML ベースの言語。主に、Web サービスの 3 つの 基本的なプロパティ (Web サービスの定義、Web サービスにアクセスする方法、および Web サービスの場所) を記述する。

XA プロトコル (XA protocol)　分散トランザクション対応のデータベース業界標準プロトコル。

XML　Extensible Markup Language の略。HTML スタイルタグを使って、ドキュメントをフォーマットするだけでなく、ドキュメントで使われるさまざまな種類の情報を識別する。

アクセス制御 (access control)　誰が、どんなアクセス権を持つかを制御することによって、Sun ONE Application Server 製品の安全を確保する方法。

アセンブリ (assembly)　アプリケーションの個別コンポーネントを配備可能な単位に結合するプロセス。「配備 (deployment)」も参照。

アプリケーション (application)　`.ear` ファイルにパッケージ化されたコンポーネント群。J2EE アプリケーション配備記述子を伴う。「コンポーネント (component)」、「モジュール (module)」も参照。

アプリケーションクライアントコンテナ (application client container)　「コンテナ (container)」を参照。

アプリケーションサーバー (application server)　ビジネスアプリケーションを実行する、信頼性が高く、安全で、スケーラブルなソフトウェアプラットフォーム。通常、アプリケーションサーバーは、コンポーネントのライフサイクル、場所、リソースの分配とトランザクションアクセスなど、高レベルのサービスをアプリケーションに提供する。

アプリケーション層 (application tier)　J2EE アプリケーションの概念的な分割。
クライアント層 : ユーザーインタフェース (UI)。エンドユーザーは、クライアントソフトウェア (Web ブラウザなど) と対話してアプリケーションを使う。
サーバー層 : アプリケーションを構成し、アプリケーションのコンポーネント内で定義されているビジネスロジックおよびプレゼンテーションロジック。
データ層 : アプリケーションがデータソースと対話できるようにするデータアクセスロジック。

アプレット (applet)　Web ブラウザで実行する、Java で書かれた小さなアプリケーション。通常、アプレットは、特別な機能を提供する Web ページに呼び出されたり、埋め込まれたりする。これに対し、サーブレットは、サーバーで実行される小さなアプリケーション。

暗号化 (encryption)　目的の受信者以外が認識できないように情報を変換するプロセス。

委譲 (delegation)　オブジェクトの構成を実装方法として使うオブジェクト指向技術の 1 つ。ある処理の結果に責任を持つオブジェクトが、委譲相手となる別のオブジェクトに実装を任せる。たとえば、クラスローダーは一部のクラスのロードを親に委譲することが多い。

イベント (event)　モジュールまたはアプリケーションからの応答をトリガする名前付きのアクション。

エンティティ Bean (entity bean)　エンタープライズ Bean は、データベースの行などの物理的なデータに関連している。エンティティ Beans は、持続データに結び付けられるので生存期間が長い。エンティティ Beans は、常にトランザクションおよびマルチユーザーを認識する。「メッセージ駆動型 Beans (message-driven bean)」、「読み込み専用 Bean (read-only bean)」、「セッション Bean (session bean)」を参照。

オブジェクトの持続性 (object persistance)　「持続 (persistence)」を参照。

外見 (facade)　アプリケーション固有の、ステートフルセッション Bean を使用してさまざまな Enterprise JavaBeans (EJBs) を管理する状態。

外部 JDNI リソース (external JDNI resource)　JNDI サービスをリモート JNDI サーバーへの橋渡しとして機能させるリソース。

会話型状態 (conversational state)　同一のクライアントと何度も対話した結果、オブジェクトの状態が変更される状態。「持続状態 (persistent state)」も参照。

仮想サーバー (virtual server)　指定した URL のターゲットとなるコンテンツを処理する仮想 Web サーバー。複数の仮想サーバーが、同一または異なったホスト名、ポート番号、IP アドレスなどを使ってコンテンツを提供できる。HTTP サービスは、URL に従って、受信する Web 要求を異なった仮想サーバーに送信できる。仮想ホストとも呼ばれる。特定の仮想サーバーに Web アプリケーションを割り当てることができる。サーバーインスタンスには、複数の仮想サーバーを持たせることができる。「サーバーインスタンス (server instance)」も参照。

活性化 (activation)　エンタープライズ Bean の状態を補助記憶装置からメモリに転送するプロセス。

カプセル化 (encapsulate) 情報をモジュール内に局所化すること。オブジェクトはデータと実装をカプセル化するので、サービスを提供するブラックボックスと見なせる。インスタンスの変数とメソッドを追加、削除、変更できるが、オブジェクトの提供するサービスが同じであれば、オブジェクトの使用するコードを書き換えずに使い続けることができる。

カラム (column) データベーステーブル内のフィールド。

監査 (auditing) エラーやセキュリティ違反などの重大なイベントが発生した場合に、それを後から調べることができるようにイベントを記録するメソッド。

管理インタフェース (Administration interface) Sun ONE Application Server の設定と管理に使用するブラウザベースのフォームの集まり。「CLI」も参照。

管理サーバー (administration server) Sun ONE Application Server の管理機能を担う専用のアプリケーションサーバーインスタンス。管理機能には、配備、ブラウザベースの管理、コマンド行インタフェース (CLI) と統合開発環境 (IDE) からのアクセスなどがある。

管理情報ベース (management information base : MIB) マスター SNMP エージェントからアクセス可能な変数を定義するツリー構造。MIB によって、HTTP サーバーのネットワーク設定、状態、および統計情報へアクセスできる。SNMP を使用すると、これらの情報を NMS (ネットワーク管理ステーション) から確認できる。「ネットワーク管理ステーション (network management station : NMS)」、「SNMP」も参照。

管理ドメイン (administrative domain) Sun ONE Application Server の機能の 1 つ。複数の管理ドメインに対応することで、複数の管理ユーザーのそれぞれが専用のドメインを作成、管理できる。ドメインはインスタンスのセットで、1 つのシステムにインストールされたバイナリの共通セットから作成される。

キーペアファイル (key-pair file) 「信頼データベース (trust database)」を参照。

キャッシュされた行セット (cached rowset) `CachedRowSet` オブジェクトを使うと、データソースからデータを取り込み、そのデータを確認したり変更したりしながらデータソースから切り離すことができる。キャッシュされた行セットには、取得した元のデータ、およびアプリケーションによるデータの変更の両方が記録される。アプリケーションが元のデータソースを更新しようとすると、行セットはデータソースに再び接続され、変更された行だけがデータベースにマージされる。

キャッシュ制御指令 (Cache Control Directives) プロキシサーバーにどの情報をキャッシュさせるかを制御する Sun ONE Application Server の機能。キャッシュ制御指令を使うことで、プロキシによるデフォルトのキャッシングがオーバーライドされ、機密情報をキャッシュせずに後から検索することができる。この指令を利用するには、プロキシサーバーが HTTP 1.1 に準拠している必要がある。

キュー (queue)　管理者が作成するオブジェクトで、ポイントツーポイント配信モデルが実装される。キューは、メッセージをコンシュームするクライアントが非活性化されている状態でも、メッセージを保持する。キューは、プロデューサとコンシューマの間のメッセージの保管場所として機能する。

行 (row)　テーブル内の各列の値を格納する 1 つのデータレコード。

クライアント規約 (client contract)　クライアントと EJB コンテナ間の通信ルールを決め、Enterprise Bean を使うアプリケーションのために均一な開発モデルを設定し、クライアントとの関係を統一することによって Bean を効率よく再利用できるように保証する規約。

クライアント認証 (client authentication)　クライアントの証明書を認証するプロセス。このプロセスでは暗号を使用して、証明書の署名と証明書チェーンが信頼できる CA のリストに載っている CA からのものであることを検証する。「認証 (authentication)」、「証明書発行局 (certificate authority)」も参照。

クラスパス (classpath)　Java クラスが格納されるディレクトリと JAR ファイルを識別するパス。「クラスローダー (classloader)」も参照。

クラスローダー (classloader)　特定のルールに従って Java クラスを読み込む機能を果たす Java コンポーネント。「クラスパス (classpath)」も参照。

グループ (group)　何らかの関連があるユーザーの集まり。通常、グループのメンバーシップはローカルシステム管理者が管理する。「ユーザー (user)」、「ロール (role)」を参照。

グローバルデータベースコネクション (global database connection)　複数のコンポーネントに対して利用可能なデータベースコネクション。データベースコネクションにはリソースマネージャが必要。

グローバルトランザクション (global transaction)　トランザクションマネージャによって管理および調整され、1 つのデータベースおよびプロセスに制限されないトランザクション。トランザクションマネージャは通常、XA プロトコルを使ってデータベースのバックエンドと対話する。「ローカルトランザクション (local transaction)」を参照。

公開鍵暗号法 (public key cryptography)　各ユーザーが公開鍵と秘密鍵を持つ暗号法。メッセージは受信者の公開鍵を使って暗号化され、受信者は秘密鍵を使ってメッセージを復号化する。この方法では、秘密鍵はユーザー以外に秘密鍵を知らせる必要がない。

コネクタ (connector)　EIS への接続を提供するコンテナ用の標準拡張メカニズム。コネクタは、EIS に固有のもので、EIS 接続用のリソースアダプタおよびアプリケーション開発ツールから構成されている。リソースアダプタは、コネクタアーキテクチャに定義されたシステムレベル規約を使ってコンテナへ接続される。

コネクタアーキテクチャ (connector architecture)　J2EE アプリケーションと EIS を統合するためのアーキテクチャ。このアーキテクチャには、EIS ベンダー提供のリソースアダプタと、このリソースアダプタの接続を許可する J2EE サーバーという 2 つの部分がある。このアーキテクチャは、トランザクション、セキュリティ、リソース管理など、リソースアダプタが J2EE サーバーに接続するために必要な規約を定義している。

コミットする (commit)　必要なコマンドをデータベースに送信することによって、トランザクションを実行すること。「ロールバック (rollback)」、「トランザクション (transaction)」を参照。

コンテナ (container)　特定のタイプの J2EE コンポーネントにライフサイクル管理、セキュリティ、配備、実行時サービスを提供するエンティティ。Sun ONE Application Server には Web コンテナと EJB コンテナがあり、アプリケーションクライアントコンテナをサポートしている。「コンポーネント (component)」も参照。

コンテナ管理による関係 (container-managed relationship)　クラスペアで表される、一方の動作が他方の動作に影響を与えるようなフィールドの関係。

コンテナ管理による持続性 (container-managed persistence)　EJB コンテナがエンティティ Bean の持続性を管理している状態。エンティティ Bean の変数とデータストアの間のデータ転送で、データアクセスロジックが Sun ONE Application Server によって決定される。「Bean 管理による持続性 (bean-managed persistence)」も参照。

コンテナ管理によるトランザクション (container-managed transaction)　Enterprise JavaBean のトランザクション境界設定を EJB コンテナが自動的に宣言して制御する。「Bean 管理によるトランザクション (bean-managed transaction)」も参照。

コントロール記述子 (control descriptor)　Enterprise Bean トランザクションおよびセキュリティプロパティだけでなく、Bean メソッドの個々のプロパティオーバーライド (オプション) を指定できるようにする一連の Enterprise Bean 設定エントリ。

コンパイル済みコマンド (prepared command)　実行の繰り返しを効率よくするために、SQL で書かれた、あらかじめコンパイルされているデータベースコマンド。コンパイル済みコマンドにはパラメータを入れることができる。コンパイル済みステートメントには、1 つまたは複数のコンパイル済みコマンドが含まれている。

コンパイル済みステートメント (prepared statement)　`QUERY`、`UPDATE`、または `INSERT` ステートメントをカプセル化したクラスで、データをフェッチするために繰り返し使用される。コンパイル済みステートメントには、1 つまたは複数のコンパイル済みコマンドが含まれている。

コンポーネント (component)　Web アプリケーション、Enterprise JavaBean、メッセージ駆動型 Bean、アプリケーションクライアント、またはコネクタ。「アプリケーション (application)」、「モジュール (module)」も参照。

コンポーネント規約 (component contract)　Enterprise JavaBean とそのコンテナ間の関係を確立する規約。

サーバーインスタンス (server instance)　Sun ONE Application Server では、同じマシンの同じインストールに複数のインスタンスを持つことができる。各インスタンスには、それぞれに専用のディレクトリ構造、設定、配備アプリケーションがある。各インスタンスに複数の仮想サーバーを持たせることもできる。「仮想サーバー (virtual server)」も参照。

サーブレット (servlet)　`Servlet` クラスのインスタンス。サーブレットは、サーバーで実行する再利用可能なアプリケーションである。Sun ONE Application Server では、サーブレットは、プレゼンテーションロジックの実行、ビジネスロジックの起動、およびプレゼンテーションレイアウトの起動または実行によって、アプリケーションでの対話ごとにセントラルディスパッチャとしての役割を果たす。

サーブレットエンジン (servlet engine)　すべてのサーブレットメタファンクションを処理する内部オブジェクト。インスタンス化および実行などのサービスをサーブレットに提供する一連のプロセス。

サーブレットランナー (servlet runner)　リクエストオブジェクトおよびレスポンスオブジェクトを持つサーブレットを起動するサーブレットエンジンの一部。「サーブレットエンジン (servlet engine)」を参照。

細分レベル (granularity level)　アプリケーションを細分化するアプローチ。細分度が高いとは、アプリケーションが細かく定義された多数の Enterprise JavaBeans (EJBs) に分割されていることを示す。細分度が低いとは、アプリケーションの分割数が少なく、大きなプログラムが生成されていることを示す。

再利用可能なコンポーネント (reusable component)　複数の容量、たとえば複数のリソースまたはアプリケーションが使えるように作成されたコンポーネント。

識別名 (Distinguished Name)　「DN」、「DN 属性 (DN attribute)」を参照。

システム管理者 (system administrator)　Sun ONE Application Server ソフトウェアを管理し、Sun ONE Application Server アプリケーションを配備する人。

持続 (persistence)　エンタープライズ Bean で、インスタンス変数と基礎となるデータベースとの間でエンティティ Beans の状態を転送するプロトコル。「トランジエンス (transience)」とは反対の概念。セッションでは、セッションのストレージメカニズムを意味する。

持続状態 (persistent state)　オブジェクトの状態が持続ストレージ (通常はデータベース) に保存されている状態。

持続マネージャ (persistence manager)　コンテナにインストールされたエンティティ Bean の持続性に対する責任を持っているエンティティ。

実行時システム (runtime system)　プログラムを実行するソフトウェア環境。実行時システムには、Java プログラミング言語で記述したプログラムのロード、ネイティブメソッドへの動的リンク、メモリー管理、例外処理に必要なコードがすべて含まれている。Java 仮想マシンの実装も含まれており、Java インタプリタになることもある。

主キー (primary key)　クライアントを特定のエンティティ Bean に配備する一意の識別子。

主キークラス名 (primary key class name)　Bean の主キーの完全修飾クラス名を指定する変数。JNDI 検索に使われる。

主体 (principal)　認証の結果として、エンティティに割り当てられる ID。

状態 (state)　1. 指定された時間におけるエンティティの環境または状態。2.Sun ONE Application Server 機能インタフェース `IState2` を使って、アプリケーションの状態を保存する分散データ保存メカニズム。「会話型状態 (conversational state)」、「持続状態 (persistent state)」も参照。

承認 (authorization)　メソッドまたはリソースへのアクセスを決定するプロセス。J2EE プラットフォームでの承認では、承認を必要とする要求に関連するユーザーが、そのセキュリティロールに含まれているかどうかが検証される。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する。

証明書 (certificate)　個人や企業などのエンティティの名前を指定するデジタルデータ。証明書に含まれる公開鍵がそのエンティティのものであることを証明する。クライアントとサーバーの両方が証明書を持つことができる。

証明書発行局 (certificate authority)　インターネットを通じて証明書を発行する企業。または、企業のイントラネットまたはエクストラネットの証明書の発行を担当する部門。

シングルサインオン (single sign-on)　1 つの仮想サーバーインスタンスの複数の J2EE アプリケーションでユーザーの認証状態を共有している状態。

信頼データベース (trust database)　公開鍵と秘密鍵を含むセキュリティファイル。「キーペアファイル (key-pair file)」とも呼ばれる。

スキーマ (schema)　基礎となるデータベースの構造で、テーブル名、カラムの種類、索引情報、主キーと外部キーの関係情報が含まれる。

ステートフルセッション Bean (stateful session bean)　特定のクライアントとのセッションを表すセッション Beans で、複数のクライアント起動メソッドのステートを自動的に管理する。

ステートレスセッション Bean (stateless session bean)　状態のないサービスを表すセッション Bean。状態のないセッション Bean は、完全にトランジェントであり、特定のクライアントが限られた時間必要とするビジネスロジックの一時的な部分がカプセル化される。

スティッキー cookie (sticky cookie)　常に同じサーバープロセスにクライアントを強制的に接続させるためにクライアントに返される cookie。「セッション cookie (session cookie)」も参照。

ストアドプロシージャ (stored procedure)　SQL で書かれ、データベースに保存されるステートメントのブロック。ストアドプロシージャを使って、レコードの変更、挿入、または削除などのすべてのタイプのデータベースオペレーションを実行できる。ストアドプロシージャを使うと、ネットワークを介して送信される情報量が減るのでデータベースのパフォーマンスが向上する。

ストリーミング (streaming)　HTTP によるデータの通信方法を管理するための技術。結果がストリーミングされると、そのデータの最初の部分をすぐに利用できる。結果がストリーミングされないと、結果全体が取得されるまで利用できない。ストリーミングを使うと、大量のデータを効率よく返すことができるため、アプリケーションの体感的なパフォーマンスが向上する。

スレッド (thread)　プロセス内部の実行シーケンス。プロセスで複数のスレッドが同時に実行される場合はマルチスレッド。各スレッドが逐次実行される場合はシングルスレッド。

生成メソッド (create method)　Enterprise Bean を作成時にカスタマイズするメソッド。

セキュリティ (security)　認証されたクライアントだけがアプリケーションリソースにアクセスできるようにしたスクリーニングメカニズム。

セッション Bean (session bean)　クライアントによって作成されるエンタープライズ Bean。通常は、1 回のクライアントサーバーセッションの間だけ存在する。セッション Bean は、クライアントのために計算や他の EJB へのアクセスなどを実行する。セッション Bean はトランザクションで使用されることもあるが、システムがクラッシュした場合に復元できない。セッション Bean オブジェクトにはステートレス (特定のクライアントに関連付けられない)、およびステートフル (特定のクライアントと関連付けられる) があり、メソッドやトランザクションの間で対話状態を保持できる。「ステートフルセッション Bean (stateful session bean)」、「ステートレスセッション Bean (stateless session bean)」も参照。

セッション cookie (session cookie)　ユーザーセッション識別子が含まれているクライアントに返される cookie。「スティッキー cookie (sticky cookie)」も参照。

セッション (session)　サーブレットが複数の HTTP リクエストでのユーザーと Web アプリケーションとの対話を追跡するために使用するオブジェクト。

セッションタイムアウト (session timeout)　ユーザーセッションの有効期限。この特定の時間を超えると、Sun ONE Application Server によってユーザーセッションが無効になる。「セッション (session)」を参照。

接続プール (Connection Pool)　物理的な接続をキャッシュおよび再利用することで、データベースへのアクセスを効率的にする方法。接続によるオーバーヘッドを回避し、多数のスレッド間で共有する接続を少数に抑えることができる。「JDBC 接続プール (JDBC connection pool)」も参照。

接続ファクトリ (connection factory)　J2EE コンポーネントがリソースにアクセスできるよう、接続オブジェクトを生成するオブジェクト。提供された JMS 実装をアプリケーションコードが使えるようにする JMS 接続 (TopicConnection または QueueConnection) の作成に使用される。アプリケーションコードは、JNDI 名を使う接続ファクトリを JNDI サービスを使って特定する。

設定 (configuration)　サーバーを調整する、またはコンポーネントのメタデータを提供するプロセス。通常、コンポーネントの設定はコンポーネントの配備記述子ファイルに保存されている。「管理サーバー (administration server)」、「配備記述子 (deployment descriptor)」も参照。

宣言によるセキュリティ (declarative security)　セキュリティプロパティをコンポーネントのコンフィグレーションファイル内で宣言し、コンポーネントのコンテナ (例 : Bean のコンテナやサーブレットエンジン) にセキュリティを暗黙的に管理させること。このタイプのセキュリティには、プログラムの制御は必要ない。「プログラムセキュリティ (programmatic security)」とは反対の概念。「コンテナ管理による持続性 (container-managed persistence)」を参照。

宣言によるトランザクション (declarative transaction)　「コンテナ管理によるトランザクション (container-managed transaction)」を参照。

送信先リソース (destination resource)　Topic 送信先または Queue 送信先を表すオブジェクト。キューの読み出しと書き込み、トピックのパブリッシュとサブスクライブを行うときにアプリケーションが使用する。アプリケーションコードは、JNDI 名を使う JMS リソースを JNDI サービスを使って特定する。

属性 (attribute)　サーブレットによって設定可能な、リクエストオブジェクト内の Name-value ペア。XML ファイル内の要素を修正する Name-value ペアでもある。「パラメータ」と対照的。一般的には、属性はメタデータの単位。

ダイジェスト認証 (digest authentication)　ユーザー名とパスワードをクリアテキストとして送信することなく、ユーザー名とパスワードに基づいてユーザーを認証する認証形態。

直列化可能オブジェクト (serializable object)　解体および再構築できるオブジェクト。複数のサーバーに保存したり分散したりできる。

データアクセスロジック (data access logic)　データソースとの対話を伴うビジネスロジック。

データソース (data source)　データベースなどの、データのソースへのハンドル。データソースは、iPlanet Application Server で登録された後、コネクションを確立してデータソースと対話できるようにするために、プログラムによって取得される。データソース定義により、データのソースへの接続方法を指定する。

データベース (database)　リレーショナルデータベース管理システム (RDBMS) の一般名。関連する組織化された大量のデータの作成および操作が可能なソフトウェアパッケージ。

データベース接続 (database connection)　データベースまたはほかのデータソースとの通信リンク。コンポーネントは、複数のデータベースコネクションを同時に作成および操作して、データにアクセスできる。

テーブル (table)　データベースの行および列内に保存されている関連データの特定のグループ。

ディレクトリサーバー (directory server)　「Sun ONE Directory Server」を参照。

デジタル署名 (digital signature)　メッセージと署名者の両方の認証に使用される電子的なセキュリティメカニズム。

電子商取引 (e-commerce)　電子商取引。インターネットで行うビジネス。

同一場所に置く (co-locate)　関連するコンポーネントと同じメモリ空間にコンポーネントを配備することによってリモートプロシージャコールを避け、パフォーマンスを向上させること。

動的再配備 (dynamic redeployment)　サーバーを再起動せずにコンポーネントを再配備するプロセス。

動的再読み込み (dynamic reloading)　サーバーを再起動せずにコンポーネントを更新して再読み込みするプロセス。デフォルトでは、サーブレット、JavaServer Page (JSP)、およびエンタープライズ Bean コンポーネントをダイナミックに再読み込みできる。バージョン付けとも呼ぶ。

ドキュメントルート (Document Root)　一次ドキュメントディレクトリ。仮想サーバーの全ファイルを格納してリモートクライアントに提供するための中心的なディレクトリ。

特殊な結果判別 (Heuristic Decision)　特定のトランザクションが使用するトランザクションモデル。トランザクションは、コミットまたはロールバックする必要がある。

トピック (topic)　管理者が作成するオブジェクトで、パブリッシュ / サブスクライブ配信モデルが実装される。送られてきたメッセージの収集と分配を担当するコンテンツ階層に含まれるノードと考えることもできる。トピックを中間媒体として使うことで、メッセージのパブリッシャとサブスクライバを分離できる。

ドメインレジストリ (Domain Registry)　Sun ONE Application Server のインストールで作成、および設定されるすべてのドメインについて、ドメイン固有の情報 (ドメインの名前、場所、ポート、ホストなど) を含む 1 つのデータ構造。

トランザクション (transaction)　グループとして成功または失敗する一連のデータベースコマンド。トランザクション全体が成功するには、そのトランザクションに関連するすべてのコマンドが成功する必要がある。

トランザクションコンテキスト (transaction context)　ローカルまたはグローバルなトランザクションの範囲。「ローカルトランザクション (local transaction)」、「グローバルトランザクション (global transaction)」を参照。

トランザクション遮断レベル (transaction isolation level)　データベース上で同時に実行されている複数のトランザクションをそれぞれに認識できる度合いを決定する。

トランザクション属性 (Transaction Attribute)　トランザクションの範囲を制御する。

トランザクションマネージャ (transaction manager)　通常 XA プロトコルを使ってグローバルトランザクションを制御するオブジェクト。「グローバルトランザクション (global transaction)」を参照。

トランザクションリカバリ (Transaction Recovery)　自動または手動による分散トランザクションのリカバリ。

トランジエンス (transience)　使われていないときにリソースを解放するプロトコル。「持続 (persistence)」とは反対の概念。

認証 (authentication)　ユーザーなどのエンティティが、別のエンティティ (アプリケーションなど) に対し特定の識別情報 (ユーザーのセキュリティ識別情報) を提供して認証されていることを証明するプロセス。Sun ONE Application Server は、基本的な認証のほかにフォームベースと SSL 相互認証もサポートしている。「クライアント認証 (client authentication)」、「ダイジェスト認証 (digest authentication)」、「ホスト -IP 認証 (host-IP authentication)」、「プラグイン対応認証 (pluggable authentication)」も参照。

ネットワーク管理ステーション (network management station : NMS)　特定のネットワークをリモート管理するためのマシン。通常、NMS ソフトウェアには、収集されたデータをグラフに表示する機能や、そのデータを使ってサーバーが特定の許容範囲内で動作していることを確認する機能がある。「SNMP」も参照。

バージョン付け (versioning)　「動的再読み込み (dynamic reloading)」を参照。

パーミッション (permission)　ユーザーまたはグループに対して付与または拒否する一連の権限。「ACL」も参照。

配備 (deployment) アプリケーションが必要とするファイルをアプリケーションサーバーに配布し、アプリケーションサーバー上でアプリケーションを実行できるようにするプロセス。「アセンブリ (assembly)」も参照。

配備記述子 (deployment descriptor) 配備方法を記述した XML ファイル。各モジュールおよびアプリケーションに備わっている。配備記述子は、配備ツールに、特定のコンテナオプションでモジュールまたはアプリケーションの配備を指示し、配備ツールが解決する必要のある特定の設定要件を示している。

バックアップストア (backup store) データのリポジトリ。一般的にはファイルシステムやデータベース。バックアップストアをバックグラウンドスレッド (スィーパスレッド) で監視して、不要なエントリを削除することができる。

パッケージ (package) 共通ディレクトリ内に保存されている、関連するクラスのコレクション。クラスのコレクションは、頻繁に、Java アーカイブ JAR ファイルにパッケージ化される。「アセンブリ (assembly)」、「配備 (deployment)」も参照。

パブリッシュ / サブスクライブ配信モデル (publish/subscribe delivery model) 一般に、パブリッシャとサブスクライバは匿名で、トピックに対して動的にパブリッシュまたはサブスクライブできる。このシステムでは、トピックの複数のパブリッシャから受信したメッセージを複数のサブスクライバに配信できる。

パラメータ (parameter) フォームフィールドデータや HTTP ヘッダー情報など、クライアントから送信される名前 - 値ペアであり、リクエストオブジェクト内にカプセル化されている。「属性」と対照的。一般的には、Java メソッドまたはデータベースコンパイル済みコマンドに渡される引数を指す。

ハンドル (handle) Enterprise Java Beans を識別するオブジェクト。クライアントはハンドルを直列化した後で直列化を解除し、Beans への参照を取得する。

汎用 ACL (general ACL) ユーザーまたはグループを 1 つまたは複数の権限に関連付ける、Sun ONE Directory Server 内の特定のリスト。一連の権限を記録するようにこのリストを定義し、自由にアクセスできる。

汎用サーブレット (generic servlet) `javax.servlet.GenericServlet` を拡張するサーブレット。汎用サーブレットはプロトコルに依存しない。これは、汎用サーブレットは本来、HTTP やその他の転送プロトコルをサポートしていないことを意味する。「HTTP サーブレット (HTTP servlet)」と対照的。

非活性化 (passivation) Bean を破棄せずに Bean のリソースを解放するメソッド。これによって、Bean は持続的になり、インスタンス化せずに再び呼び出すことができる。

ビジネスロジック (business logic) データ統合ロジックやプレゼンテーションロジックではなく、不可欠なビジネスルールを含むアプリケーションコード。

非同期通信 (asynchronous communication) メッセージの送信側が、他の処理を継続する前に送信メソッドの返りを待つ必要のない通信モード。

秘密鍵 (private key) 「公開鍵暗号法 (public key cryptography)」を参照。

プール (pooling) 設定済みのリソースを増やしてパフォーマンスを向上させるプロセス。リソースがプールされていると、コンポーネントは新しくインスタンス化しなくても、プールから既存のインスタンスを使用できる。Sun ONE Application Server では、データベースコネクション、サーブレットインスタンス、および エンタープライズ Bean インスタンスをすべてプールできる。

ファイアウォール (firewall) セキュリティを強化するために、管理者がネットワーク上の情報フローを制限するときに使用する電子的な境界。

ファイルキャッシュ (File Cache) ファイルキャッシュには、ファイルに関する情報と静的なファイルコンテンツが含まれる。ファイルキャッシュはデフォルトで有効になっている。

ファクトリクラス (factory class) 持続性マネージャを作成するクラス。「接続ファクトリ (connection factory)」も参照。

フェィルオーバー (failover) Bean がサーバークラッシュに透過的に耐えられるようにするリカバリプロセス。

フォームアクションハンドラ (form action handler) フォーム上の特定のボタンに基づいてアクションを実行する、サーブレットまたはアプリケーションロジック内で特別に定義されているメソッド。

復号化 (decryption) 暗号化された情報を認識可能な状態に戻すプロセス。

符号化方式 (cipher) 暗号化と復号化に使用される暗号化アルゴリズム (関数)。

プラグイン対応認証 (pluggable authentication) J2EE アプリケーションが J2SE プラットフォームから JAAS (Java Authentication and Authorization Service) を利用できるようにするメカニズム。開発者は、独自の認証メカニズムをプラグインできる。

プレゼンテーションレイアウト (presentation layout) Web ページコンテンツの形式。

プレゼンテーションロジック (presentation logic) アプリケーションでページを作成するアクティビティ。リクエストの処理、レスポンスコンテンツの生成、クライアントに返すページのフォーマット化など。通常は、Web アプリケーションによって処理される。

ブローカ (broker) JMS メッセージのルーティング、配信、持続性、セキュリティ、ログを管理する Sun ONE Message Queue のエンティティ。管理者がパフォーマンスとリソース使用率の監視と調整に使うインタフェースを提供する。

プログラマによる境界設定トランザクション (programmer-demarcated transaction)

「Bean 管理によるトランザクション (bean-managed transaction)」を参照。

プログラムセキュリティ (programmatic security) コンポーネントのコンテナ (Bean のコンテナやサーブレットエンジンなど) による処理ではなく、コードを記述して明示的にセキュリティを制御するプロセス。「宣言によるセキュリティ (declarative security)」とは反対の概念。

プロセス (process) アクティブなプログラムの実行シーケンス。プロセスは、1 つまたは複数のスレッドから構成される。

プロパティ (property) アプリケーションコンポーネントの動作を定義する 1 つの属性。`server.xml` ファイルでは、プロパティは名前と値のペアを含む要素である。

分散可能セッション (distributable session) クラスタ内のすべてのサーバー間に分散できるユーザーセッション。

分散トランザクション (distributed transaction) 別個のサーバー上に配備されている複数の異種データベースに適用可能な 1 つのトランザクション。

分離レベル (isolation level) 「トランザクション遮断レベル (transaction isolation level)」を参照。

ホームインタフェース (home interface) クライアントによる Enterprise Bean の作成や削除を可能にするメソッドを定義するメカニズム。

ポイントツーポイント配信モデル (point-to-point delivery model) プロデューサはメッセージを特定のキューに送り、コンシューマは、そのメッセージを保持するために確立されたキューからメッセージを抽出する。1 つのメッセージは 1 つのメッセージコンシューマだけに配信される。

ホスト -IP 認証 (host-IP authentication) 特定のコンピュータを使うクライアントだけにアクセスを限定することによって、管理サーバー、または Web サイト上のファイルやディレクトリへのアクセスを制限するセキュリティメカニズム。

マッピング (mapping) オブジェクト指向モデルを、データのリレーショナルモデル (通常はリレーショナルデータベースのスキーマ) に結びつける機能。スキーマを別の構造に変換するプロセス。ユーザーとセキュリティロールとの関連付けも意味する。

メタデータ (metadata) コンポーネントの名前やその動作の仕様などの、コンポーネントに関する情報。

メッセージ駆動型 Beans (message-driven bean)　非同期メッセージコンシューマの Enterprise JavaBean。メッセージ駆動型 Beans は特定のクライアントのステートを持っていないが、インスタンス変数はクライアントメッセージの処理に関するステート (オープンデータベースコネクションや EJB オブジェクトへのオブジェクト参照など) を持っていることがある。クライアントは、メッセージ駆動型 Bean がメッセージリスナとなっている宛先にメッセージを送信して、メッセージ駆動型 Bean にアクセスする。

メッセージング (messaging)　エンタープライズ Bean が使用する非同期の要求、報告、またはイベントのシステム。緩く結合されたアプリケーション間で確実かつ安全に情報をやり取りするときに利用される。

モジュール (module)　アプリケーションの外部に個別に配備された Web アプリケーション、Enterprise Bean、メッセージ駆動型 Bean、アプリケーションクライアント、またはコネクタ。「アプリケーション (application)」、「コンポーネント (component)」、「ライフサイクルモジュール (lifecycle module)」も参照。

ユーザー (user)　アプリケーションを使う人。プログラム的には、アプリケーションがクライアントを認識する際の手掛かりとなるユーザー名、パスワード、および一連の属性で構成される。「グループ (group)」、「ロール (role)」も参照。

ユーザーセッション (user session)　サーバーによって記録される、ユーザーとアプリケーション間の一連の対話。セッションでは、ユーザーステート、持続オブジェクト、および ID 認証が管理される。

要素 (element)　より大きなセットの一部分。たとえば、配列内のデータ単位や論理要素など。XML ファイルでは、これが基本構造単位となる。XML 要素は、サブ要素またはデータを含み、属性を含むこともある。

呼び出し可能なステートメント (callable statement)　ストアドプロシージャからのリザルトセットの戻しをサポートしているデータベースのデータベースプロシージャまたは関数呼び出しがカプセル化されているクラス。

読み込み専用 Bean (read-only bean)　EJB クライアントで修正されることがないエンティティ Bean。「エンティティ Bean (entity bean)」も参照。

ライフサイクルイベント (lifecycle event)　起動や停止など、サーバーのライフサイクルの各段階。

ライフサイクルモジュール (lifecycle module)　サーバーライフサイクルのイベントに応じて、タスクを待機し、実行するモジュール。

リクエストオブジェクト (request object)　クライアントによって生成されたページおよびセッションデータが含まれているオブジェクトであり、入力パラメータとしてサーブレットまたは JavaServer Page (JSP) に渡される。

リスナー (Listener)　ポストするオブジェクトに登録され、イベント発生時の処理を指示するクラス。

リソース参照 (resource reference)　配備記述子の要素で、リソースのコード化されたコンポーネント名を識別する。

リソースマネージャ (resource manager)　リソース (たとえばデータベースやメッセージブローカ) とクライアント (たとえば Sun ONE Application Server プロセス) のまとめ役となるオブジェクト。グローバルに利用可能なデータソースを制御する。

リモートインタフェース (remote interface)　Enterprise JavaBean の 2 つのインタフェースのうちの 1 つ。リモートインタフェースでは、クライアントから呼び出すビジネスメソッドを定義する。

レスポンスオブジェクト (response object)　呼び出しているクライアントを参照して、そのクライアントへの出力を生成するメソッドを提供するオブジェクト。

レルム (realm)　共通セキュリティポリシーが定義され、セキュリティサービスのセキュリティ管理者によって適用されている領域。J2EE 仕様では、セキュリティポリシードメインまたはセキュリティドメインとも呼ばれる。

ローカルインタフェース (local interface)　同じ Java 仮想マシン (JVM) にあるクライアントのメカニズムに、Bean にアクセスするためのセッションやエンティティ Bean を提供するインタフェース。

ローカルセッション (local session)　1 つのサーバーだけに見えるユーザーセッション。

ローカルデータベース接続 (local database connection)　ローカルコネクションのトランザクションコンテキストは現在のプロセスおよびデータソースに対してローカルであり、複数のプロセスまたはデータソース全体に分散できない。

ローカルトランザクション (local transaction)　1 つのデータベースに固有で、1 つのプロセス内に制限されるトランザクション。ローカルトランザクションは、1 つのバックエンドでのみ動作する。ローカルトランザクションは通常、JDBC API を使って区別される。「グローバルトランザクション (global transaction)」も参照。

ロール (role)　アプリケーションにおいてサブジェクトを機能別にグループ分けしたもの。配備環境では 1 つまたは複数のグループによって表される。「ユーザー (user)」、「グループ (group)」も参照。

ロールバック (rollback)　トランザクションの取り消し。

索引

数字

A

B

C

D

E

F

G

H

I

J

N

O

P

Q

R

S

T

U

V

W

え

お

か

き

く

け

こ

さ

し

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

む

め

も

ゆ

よ

ら

り

る

れ

ろ