開発者ガイド

Sun™ ONE Application Server

Version 7

817-0602-10
2002 年 9 月

目 次

本書について

このマニュアルでは、Sun™ Open Net Environment (Sun ONE) Application Server 7 上での新しい Java オープンスタンダードモデルである サーブレット、Enterprise JavaBeans (EJB コンポーネント)、および JavaServer Pages (JSP) に準拠する Java 2 Platform Enterprise Edition (J2EE) アプリケーションの作成および実行方法について説明します。また、プログラミングの概念およびタスクを記述するほかに、サンプルコード、ヒント、および参考資料 (用語集など) を提供します。

この章には次の項目があります。

- 対象読者
- マニュアルの使用法
- マニュアルの構成
- 関連情報
- マニュアルの表記規則
- 製品サポート

対象読者

このマニュアルは、企業内で J2EE アプリケーションの開発、アセンブリ、および配備を担当する方々を対象としています。

このマニュアルでは、次の項目に精通していることを前提としています。

- J2EE 仕様
- HTML
- Java プログラミング
- サーブレット、JSP、EJB、および JDBC の仕様に定義されている Java API

- SQL などの構造化データベースクエリ言語
- リレーショナルデータベースの概念
- デバッグ、ソースコード制御を含むソフトウェア開発プロセス

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。次のサイトを参照してください。

`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	Sun ONE Application Server 7 プラットフォーム
アプリケーションサーバーの紹介。新機能、評価用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 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 Application Server 7 の エンタープライズ Beans 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
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構成 • 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構成 • Java Messaging Service の構成 • J2EE 機能の使用 • CORBA ベースのクライアント機能の構成 • データベース接続性の構成 • トランザクション管理の構成 • Web コンテナの構成 • 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
サーバー構成ファイルの編集	管理者用構成ファイルリファレンス

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 Application Server 7 運用環境のセキュリティの設定および管理。セキュリティに関する一般情報、証明書、SSL/TLS による暗号化など。HTTP サーバーベースのセキュリティについても解説	セキュリティ管理者ガイド
Sun ONE Application Server 7 で利用する 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 についての情報およびサンプル XML ファイルを提供	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 Message Queue の使用	次のサイトに用意されている Sun ONE Message Queue のマニュアルを参照 http://docs.sun.com

マニュアルの構成

このマニュアルでは、プログラムの設計に関する Sun ONE Application Server 環境の概要を次の項目に沿って説明します。

- 第 1 章「アプリケーションの設計」

  この章では、アプリケーションの設計プロセスの概要を説明し、Sun ONE Application Server の効果的な開発のためのガイドラインを提供します。

- 第 2 章「J2EE アプリケーションの開発」

  この章では、開発環境を設定する方法、およびアプリケーションコンポーネントの作成手順について説明します。

- 第 3 章「J2EE アプリケーションのセキュリティ」

  この章では、安全な J2EE アプリケーションを記述する方法について説明します。安全な J2EE アプリケーションには、サーブレットと EJB ビジネスロジックのユーザー認証とアクセス承認を実行するコンポーネントが含まれます。

- 第 4 章「J2EE アプリケーションのアセンブルと配備」

この章では、Sun ONE Application Server モジュールの内容と、これらのモジュールをアプリケーション内に個別に、または一括してアセンブルする方法について説明します。

- 第 5 章「J2EE アプリケーションのデバッグ」

  この章では、Sun ONE Application Server 7 でアプリケーションをデバッグするためのガイドラインを示します。

- 第 6 章「ライフサイクルリスナーの開発」

  この章では、ライフサイクルモジュールの作成方法と使用方法について説明します。ライフサイクルモジュールは、サーバーの起動時に自動的に開始され、サーバーがシャットダウンされると自動的に通知されます。

このマニュアルの最後には、用語集と索引があります。

関連情報

公式の仕様書の URL ディレクトリには、*install_dir*`/docs/index.htm` からアクセスできます。また、次の書籍や Web サイトも参考にしてください。

J2EE の一般情報:

『Core J2EE Patterns: Best Practices and Design Strategies』、Deepak Alur、John Crupi、Dan Malks 共著、Prentice Hall Publishing 発行

『Java Security』、Scott Oaks 著、O'Reilly Publishing 発行

サーブレット および JSP を使ったプログラミング

『Java Servlet Programming』、Jason Hunter 著、O'Reilly Publishing 発行

『Java Threads, 2nd Edition』、Scott Oaks、Henry Wong 共著、O'Reilly Publishing 発行

EJB コンポーネントを使ったプログラミング:

『Enterprise JavaBeans』、Richard Monson-Haefel 著、O'Reilly Publishing 発行

JDBC を使ったプログラミング

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

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

マニュアルの表記規則

ここでは、このマニュアル全体に適用される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

このマニュアルに適用される一般的な表記規則は次のとおりです。

- **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「/」記号ではなく「¥」記号を使用します。
- URL は次のように表記されます。

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

  これらの URL で、*server* はアプリケーションを実行するサーバー名で、*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。
- このマニュアルでは、**フォントについて次の規則**を採用しています。
  - 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。
  - また斜体文字は、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、段落の先頭文字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**は、*install_dir* として表記されます。例外については、15 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。

  ほとんどのプラットフォームのデフォルトの *install_dir* は次のとおりです。
  - Solaris 8 のパッケージベースでない評価 (Evaluation) バージョン

    user's home directory/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

```
/opt/SUNWappserver7
```

- Windows のすべてのインストール

```
C:¥Sun¥AppServer7
```

上記プラットフォームでは、*default_config_dir* と *install_config_dir* は *install_dir* と同じ意味です。これ以外の説明と例外については、15 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* と表記されます。これは、実際には次のディレクトリを示しています。

  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、**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。

注　Forte for Java 4.0 は、名称が変更されました。このマニュアルでは Sun ONE Studio 4 と表記されます。

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 との一括インストールでは、アプリケーションサーバーのファイルは通常は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 にバンドルのインストール**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、ライセンスやそ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など、インストール全体に適用される設定情報が保存されます。

- **Solaris 8 および 9 のパッケージベースの**評価用以外のアンバンドル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 /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、ライセンスやそ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など、インストール全体に適用される設定情報が保存されます。

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。

事前に次の情報を準備してください。テクニカルサポートスタッフが問題解決のお手伝いする上で、この情報が役立ちます。

- 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 章

アプリケーションの設計

この章では、アプリケーションの設計プロセスの概要を説明し、Sun ONE Application Server の効果的な開発のためのガイドラインを提供します。

この章には次の節があります。

- アプリケーションの要件
- J2EE プログラミングモデルについて
- J2EE アプリケーションの最適な設計方法

アプリケーションの要件

Sun ONE Application Server アプリケーションを開発するときは、まず、アプリケーションの要件を明確にします。通常、高速かつ安全であり、新規ユーザーによる追加要求に対して信頼性の高い処理が期待できる、広範囲に配備可能なアプリケーションを開発することを意味します。

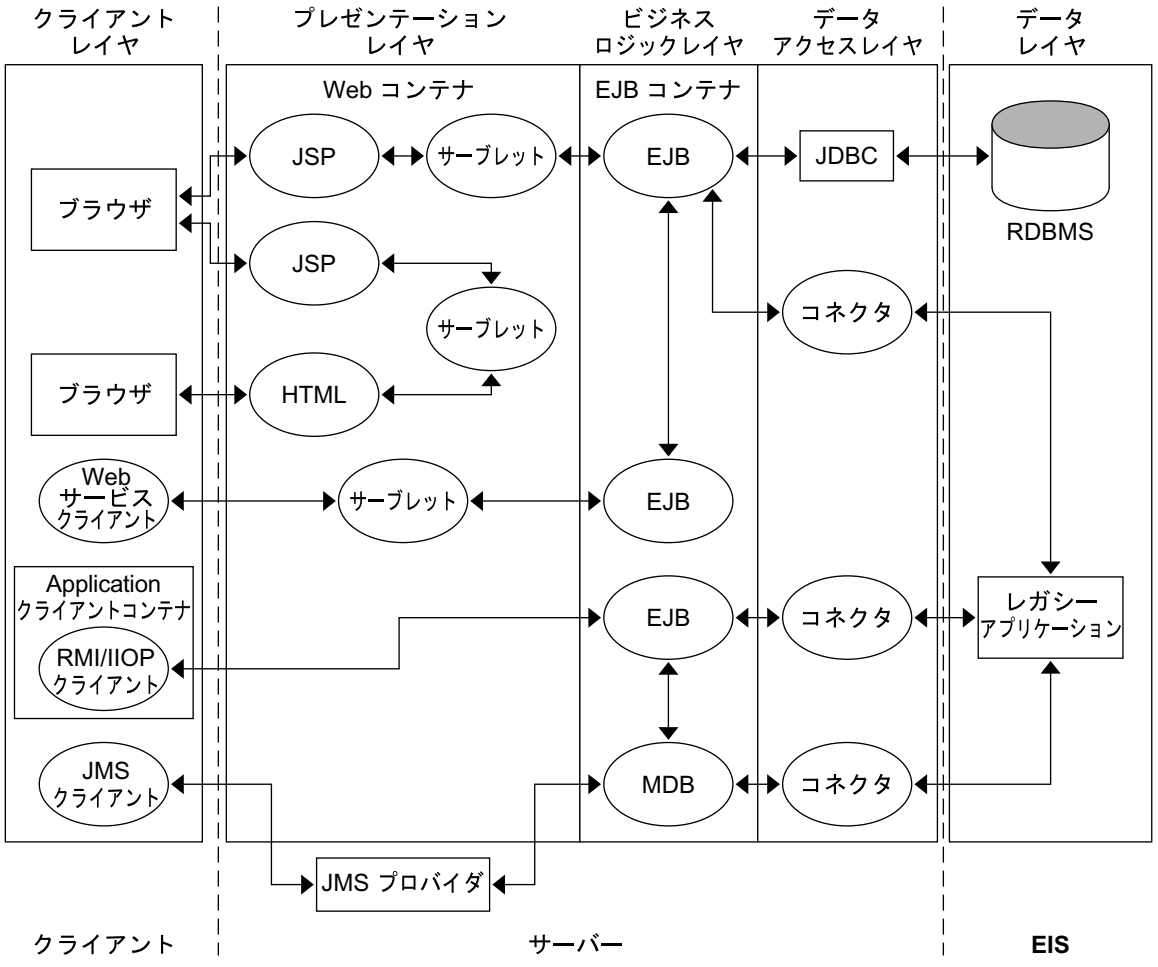
Sun ONE Application Server は、J2EE API だけでなく既存の高性能機能もサポートしており、これらの要件を満たしています。たとえば、オンラインバンキングアプリケーションでは、次の機能を提供することができます。

- セキュリティ
- 特定機能の導入時間の短縮。口座振替、会計報告、オンライン取引、特定の条件を満たしたお客様に対する特別サービスなど
- さまざまなタイプのエンドユーザーの管理。個人、法人、社内ユーザー (銀行の従業員) など
- 内部レポート
- EIS (Enterprise Information System : 企業情報システム) との接続性。従来のデータベースに格納されている情報へのアクセスを提供

J2EE プログラミングモデルについて

次の図では、クライアントマシンが Web ブラウザ、Web サービス、RMI/IIOP、または JMS クライアントを稼動し、J2EE サーバーマシンが Sun ONE Application Server を稼動しています。データベースとレガシーアプリケーションは、EIS サーバーマシンによって稼動されます。JSP とサーブレットはクライアントレイヤにインタフェースを提供し、EJB コンポーネントはビジネスレイヤに含まれます。レガシーアプリケーションへのインタフェースは、コネクタによって提供されます。

J2EE のアプリケーションレイヤ



分散アプリケーションモデルでは、個々の異なるアプリケーションレイヤを別々の機能要素に集中させることができるため、パフォーマンスが向上します。

これらのアプリケーションレイヤについては、次の節で説明します。

- クライアントレイヤ
- プレゼンテーションレイヤ
- ビジネスロジックレイヤ
- データアクセスレイヤ

クライアントレイヤ

クライアントレイヤは、ユーザーがアプリケーションにアクセスするレイヤです。アプリケーションには、次のいずれかのタイプのクライアントが必要な場合があります。

- ブラウザクライアント
- シンプルな CORBA クライアント
- ACC クライアント
- Web サービスクライアント
- JMS クライアント

クライアントレイヤ内のコンポーネントに関する詳細は、『Sun ONE Application Server Developer's Guide to Clients』を参照してください。

ブラウザクライアント

多くの場合、クライアントはブラウザです。

シンプルな CORBA クライアント

CORBA (Common Object Request Broker Architecture) と互換性のあるクライアントであれば、どれでも Sun ONE Application Server に配備された EJB コンポーネントにアクセスできます。クライアントは、Java、C、C++、Visual Basic など、CORBA に対応している言語で記述できます。CORBA クライアントは、スタンドアロンプログラムまたは別のアプリケーションサーバーを、Sun ONE Application Server に配備したアプリケーションのクライアントとして動作させる場合に使用します。

Sun ONE Application Server では、『Enterprise JavaBeans Specification, V2.0』および『Enterprise JavaBeans to CORBA Mapping』仕様書で指定されている IIOP プロトコルを使用した EJB コンポーネントへのアクセスをサポートしています。

ACC (Application Client Container) を使用しないシンプルな CORBA クライアントには、次の制限があります。

- JNDI がサポートされない。ただし、名前を変換し、標準の COSNaming バインドを使って検索できる
- RMI/IIOP を介した SSL がサポートされない
- `sun-application-client.xml` ファイルと `sun-acc.xml` ファイルで設定する機能を利用できない

ACC クライアント

Sun ONE Application Server では、Java で記述され、RMI/IIOP を介してサーバーと通信する ACC (Application Client Container) CORBA クライアントをサポートしています。

Sun ONE Application Server は、ACC クライアントプログラムを実行可能にするシステムサービスを提供します。ACC クライアントは、JNDI (Java Naming and Directory Interface) を使って EJB コンポーネント、JDBC リソース、JavaMail などのサービスを特定します。特別な機能の設定には、`application-client.xml` と `sun-application-client.xml` という配備記述子ファイルを使います。

このマニュアルで説明するすべての CORBA クライアントは、特に明記されていない限り ACC クライアントでもあります。

Web サービスクライアント

通常、ビジネスアプリケーションは、HTTP を介した SOAP プロトコルを使用して、所定の URL で Web サービスへリクエストを送信します。サービスはそのリクエストを受信し、処理した後、応答を返します。よく引用される Web サービスの例として、株式相場サービスがありますが、その場合、指定した株式の現在値をリクエストすると、その株価を応答するというサービスです。

Sun ONE Application Server では、Apache SOAP バージョン 2.2 および JAX RPC 1.1 をサポートしています。Apache SOAP Web サービスのサポートは、Sun ONE Studio 4 にも組み込まれています。

Web サービスの詳細は、『Sun ONE Application Server Developer's Guide to Web Services』を参照してください。

JMS クライアント

JMS (Java Message Service) を使用する Java アプリケーションは、JMS クライアントと呼ばれます。JMS クライアントは、メッセージの作成、送信、受信、および読み取りが可能です。メッセージを送信するクライアントはプロデューサと呼ばれ、メッセージを受信するクライアントはコンシューマと呼ばれます。JMS の詳細は、『Sun ONE Message Queue 開発者ガイド』を参照してください。

プレゼンテーションレイヤ

プレゼンテーションレイヤでは、ユーザーインタフェースが動的に生成されます。アプリケーションには、プレゼンテーションレイヤ内に、次の J2EE コンポーネントが必要です。

- サーブレット
- JSP
- 静的なコンテンツ

さらに、アプリケーションには、プレゼンテーションレイヤ内に、次の J2EE 以外の HTTP サーバーベースのコンポーネントが必要になる場合があります。

- SHTML
- CGI

プレゼンテーションレイヤ内のコンポーネントに関する詳細は、『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。

サーブレット

サーブレットは、アプリケーションのプレゼンテーションロジックを処理します。サーブレットは、ページ間移動ディスパッチャ、セッション管理、および簡単な入力確認を処理します。また、ビジネスロジックの要素を互いに関連付けます。

したがって、サーブレット開発者は、HTTP リクエスト、セキュリティ、国際化、および Web のステートレス (セッション、cookie、タイムアウトなど) に関するプログラミングの問題を理解する必要があります。Sun ONE Application Server アプリケーションでは、サーブレットは Java で記述する必要があります。サーブレットは、JSP、EJB コンポーネント、および JDBC オブジェクトを呼び出します。したがって、サーブレット開発者はこれらのアプリケーション要素の開発者と緊密に協力して作業する必要があります。

JSP

JSP は、ほとんどのアプリケーション表示タスクを処理し、サーブレットとともに動作することによって、アプリケーションの表示画面およびページ移動を定義します。JSP は、EJB コンポーネントおよび JDBC オブジェクトを呼び出します。EJB コンポーネントは、通常、ビジネスロジック機能をカプセル化します。EJB は、このようにして、計算やその他の繰り返し要求されるタスクを実行します。JDBC オブジェクトは、データベースに接続し、クエリを実行し、クエリ結果を返します。

静的なコンテンツ

画像や HTML ページなどの静的なコンテンツも利用できます。適切に設計された HTML ページでは、次の効果が得られます。

- ブラウザが異なっていても外観が統一される
- 低速モデムによる接続でも HTML が効果的に読み込まれる
- サーブレットまたは JSP がディスパッチした、動的に生成されたページが表示される

SHTML

SHTML (サーバー解析 HTML) ファイルとは、サーバー上で実行されるタグを含んだ HTML ファイルのことです。SSI などの標準サーバーサイドタグをサポートするほかに、Sun ONE Application Server 7 は、サーブレットを埋め込んだり、独自のサーバーサイドタグを定義することができます。

CGI

CGI (Common Gateway Interface) プログラムは、サーバー上で動作し、リクエストを送信したクライアントに返す応答を生成します。CGI プログラムは、シェルスクリプトとして、C、C++、Java、Perl などさまざまな言語で記述されます。CGI プログラムは、URL を呼び出すことにより、起動します。Sun ONE Application Server は、CGI バージョン 1.1 仕様に準拠しています。

ビジネスロジックレイヤ

ビジネスロジックレイヤには通常、ビジネスルールおよびその他のビジネス機能を、次のコンポーネントにカプセル化する EJB コンポーネントが配備されています。

- セッション Beans
- エンティティ Beans
- メッセージ駆動型 Beans

ビジネスロジックレイヤ内のコンポーネントに関する詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

セッション Beans

セッション Beans で、ビジネスプロセスおよび規則のロジックをカプセル化します。たとえば、セッション Beans では請求書の税金を計算できます。頻繁に変更が加えられる複雑なビジネスルールの場合 (新しい商慣習、政府規制など)、通常、アプリケーションはエンティティ Beans よりもセッション Beans を多く使うので、セッション Beans を絶えず改訂する必要があります。

セッション Beans は、ほかの EJB コンポーネントだけでなく、あらゆる JDBC インタフェースを呼び出します。アプリケーションは、本来ステートフルなセッション Beans がステートレスであっても、十分に動作するようになっています。ショッピングカートなど、ユーザー固有の状態をサーバー側で維持する必要がある場合は、ステートフルセッション Bean が必要です。

エンティティ Beans

エンティティ Beans は、データベース行などの持続的オブジェクトを表します。エンティティ Beans はあらゆる JDBC インタフェースを呼び出します。ただし、ほかの EJB コンポーネントを呼び出すことはありません。エンティティ Beans 開発者の役割は、組織のビジネスデータのオブジェクト指向ビューを設計することです。多くの場合、オブジェクト指向ビューを作成することは、エンティティ Beans へデータベーステーブルをマッピングすることです。たとえば、開発者は、顧客テーブル、インボイステーブル、および注文書テーブルを、対応する顧客オブジェクト、請求書オブジェクト、および注文書オブジェクトに変換します。

エンティティ Beans 開発者は、セッション Beans 開発者および サーブレット開発者と協力して、アプリケーションでの持続性ビジネスデータへの高速でスケーラブルなアクセスを実現します。

エンティティ Bean の持続性には次の 2 種類があります。

- コンテナ管理による持続性 (CMP) - ビジネスロジックとデータベースの間のやり取りは、EJB コンテナによって維持される
- Bean 管理による持続性 (BMP) - データベースとのやり取りを制御するコードの記述は開発者に任される

メッセージ駆動型 Beans

メッセージ駆動型 Beans は、エンティティ Beans と同様に、あらゆる JDBC インタフェースを呼び出す持続性のあるオブジェクトです。ただし、その他の EJB コンポーネントとは異なり、メッセージ駆動型 Beans はローカルインタフェースやリモートインタフェースを持ちません。また、Beans へのアクセス方法も他のエンティティ Beans とは異なります。

メッセージ駆動型 Bean は、待ち行列または永続的なサブスクリプションからのメッセージを確実にコンシュームできるメッセージリスナーです。アプリケーションクライアント、別の EJB コンポーネント、Web コンポーネントなどの J2EE コンポーネントから送信されるメッセージや、J2EE テクノロジを使用しないアプリケーションやシステムから送信されるメッセージを処理できます。

たとえば、在庫エンティティ Beans は、在庫数が設定された下限を下回ったら、在庫注文メッセージ駆動型 Beans へメッセージを送信します。

データアクセスレイヤ

データアクセスレイヤでは、JDBC (Java database connectivity) を使用して、データベースへの接続、クエリの実行、およびクエリ結果の返送を行います。また、IBM の CICS のようなレガシー EIS システムと Sun ONE Application Server の通信を可能にするために、カスタムコネクタが使われます。

開発者は、J2EE CA (コネクタアーキテクチャ) を使って、次のシステムへのアクセスを統合します。

- 企業リソース管理システム
- メインフレームシステム
- サードパーティ製のセキュリティシステム

JDBC の詳細については、『Sun ONE Application Server Developer's Guide to J2EE Features and Services』を参照してください。

コネクタの詳細については、『Sun ONE J2EE CA Service Provider Implementation 管理者ガイド』および該当するリリースノートを参照してください。

J2EE アプリケーションの最適な設計方法

この節では、Sun ONE Application Server アプリケーションの設計および開発時に考慮する必要があるガイドラインについて簡単に説明します。『Core J2EE Patterns: Best Practices and Design Strategies』(Deepak Alur、John Crupi、and Dan Malks 著) も参考になります。

このガイドラインは次の目的に分類されます。

- サーブレットおよび JSP を使ったデータ表示
- 再利用可能なアプリケーションコードの作成
- アプリケーションのモジュール化

サーブレットおよび JSP を使ったデータ表示

サーブレットはプレゼンテーションロジックによく使われ、ユーザー入力およびデータ表示のセントラルディスパッチャとして機能します。JSP は、プレゼンテーションレイアウトをダイナミックに生成します。サーブレットおよび JSP を使うと、条件に応じてさまざまなページを生成できます。

ページのレイアウトが主であり、ページを作成するための処理が最小限の場合は、対話に JSP を使った方が簡単な場合があります。

たとえば、オンライン書店のアプリケーションではユーザーの認証後に、ユーザーが、本の検索、選択したアイテムの購入などのいくつかのタスクを行うためのポータルフロントページを提供します。このポータルページは処理をほとんど行わないため、JSP で実装することができます。

JSP と サーブレットを 1 枚のコインの表と裏と考えてください。JSP のタスクはサーブレットでも実行でき、その逆の操作も可能です。しかし、JSP のタスクは JSP に最適化されており、サーブレットのタスクはサーブレットに最適化されています。サーブレットは処理能力と適応性に優れています。ただし、Java ファイルから HTML の出力を行うと、扱いにくい `println` 文が多量に発生します。JSP は HTML ファイルなので、複雑な計算や処理タスクの実行には不向きですが、HTML エディタで編集できるので、レイアウト作業に優れています。JSP とサーブレットを併用することで、両者の長所を生かすことができます。

サーブレットと JSP 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。

再利用可能なアプリケーションコードの作成

オブジェクト指向の最適な設計方針が使われている場合を除き、再利用性を最大限にするためには、アプリケーション開発時に次の点を考慮する必要があります。

- コードツリーを移動した場合でもリンクが無効にならないように、相対パスと相対 URL を使う
- JSP 内での Java の使用を最小限に抑え、サーブレットおよびヘルパークラス内で Java を使う。JSP 設計者は、Java に関する高度な知識がなくても JSP を修正できる
- データソース名、テーブル名、カラム名、JNDI オブジェクト名、ほかのアプリケーションプロパティ名などのハードコードされた文字列の保存には、プロパティファイルまたはグローバルクラスを使う
- ドメイン固有のビジネスルールや入力確認などの頻繁に変更されるビジネスルールの保存には、サーブレットや JSP でなく、セッション Beans を使う
- 持続オブジェクトにはエンティティ Beans を使う。エンティティ Beans を使うことで、各ユーザーが複数の Bean を使用することができる
- 柔軟性を最大限に高めるために、Java クラスでなく Java インタフェースを使う
- レガシーデータにアクセスするには、J2EE CA を使う

アプリケーションのモジュール化

J2EE アプリケーションを設計するときは、次の主要な要素を考慮する必要があります。

- 機能の分離
- 再利用可能なコード
- パッケージ済みコンポーネント
- 共有フレームワーククラス
- セッションとセキュリティの問題

モジュールおよびアプリケーションのアセンブル方法の詳細は、第 4 章「J2EE アプリケーションのアセンブルと配備」を参照してください。

機能の分離

各コンポーネントには、特定の処理だけを割り当てる必要があります。たとえば給与システムでは、401k アカウントにアクセスする EJB コンポーネントと給与データベースにアクセスする EJB コンポーネントとは、分離する必要があります。タスクを機能別に分離することによって、ビジネスロジックは物理的に 2 つの Bean に分離されます。異なる開発チームがこれらの Bean を作成する場合、各チームは EJB JAR パッケージを個別に開発する必要があります。

シナリオ 1

ユーザーインタフェース開発チームが、2 つの Bean 開発チームと協力して作業するとします。この場合、UI 開発チームは、サーブレット、JSP、および静的ファイルを 1 つの WAR ファイルにアセンブルする必要があります。次に例を示します。

給与システム EAR ファイル = 給与 EJB jar
+ 401k EJB JAR
+ UI チームの 1 つの共通 war

EAR ファイル内でこのように機能を分離しても、各コンポーネント間で対話することができます。これらの Bean は、個別の EJB JAR ファイルから相互にビジネスメソッドを呼び出すことができます。

シナリオ 2

Bean の各開発チームに、UI 開発チームが個別に存在するとします。この場合、各 Web 開発チームは、サーブレット、JSP、および静的ファイルを個別の WAR ファイルにアセンブルする必要があります。次に例を示します。

給与システム EAR ファイル = 給与 EJB jar
+ 401k EJB JAR
+ 1 つの給与 UI チームの war
+ 1 つの 401k UI チームの war

このように設定すると、各 WAR ファイルのコンポーネントからほかの WAR ファイルのコンポーネントにアクセスできます。

アセンブリ式

モジュールおよびアプリケーションをアセンブルするときは、いくつかの一般的なアセンブリ式に従う必要があります。

次の表は、アセンブリ式の概要を示しています。左の列は開発グループのタイプ、真中の列はグループ内のチーム、右の列はモジュール化の式を示しています。

アセンブリ式

開発グループのタイプ	グループ内のチーム	モジュール化の式
小規模なワークグループ	1 Web チーム + 1 EJB チーム	1 EAR = 1 EJB + 1 WAR
企業のワークグループ	2 EJB チーム + 1 Web チーム + 1 コンポーネント	1 EAR = 2 EJB + 1 WAR + 1 つの個別コンポーネント

再利用可能なコード

コンポーネントの再利用は、アプリケーションよりもモジュールをアセンブルしたり配備したりするときに、特に有効です。ある開発者チームが開発したコードが、複数のアプリケーション (EAR ファイルが異なる場合) からアクセスされる再利用可能なコンポーネントである場合、そのコードを個別モジュールとして配備する必要があります。詳細は、第 4 章「J2EE アプリケーションのアセンブルと配備」を参照してください。

パッケージ済みコンポーネント

アプリケーションを最初から作成したくない場合は、パッケージ済みコンポーネントを利用できます。J2EE コンポーネントの主なベンダーは、一連のサービスが用意されたさまざまなパッケージ済みコンポーネントを提供しています。パッケージ済みコンポーネントでは、アプリケーションに必要な標準コンポーネントの最大 60% を提供することを目標としています。Sun ONE Application Server では、これらのコンポーネントを利用して、アプリケーションを簡単にアセンブルできます。

共有フレームワーククラス

複数のアプリケーションが単一のモジュールライブラリにアクセスしなければならないこともあります。この場合、次の理由から、各 J2EE アプリケーションにライブラリを含めることはお勧めできません。

- **ライブラリサイズ** : ほとんどのフレームワークライブラリはサイズが大きいため、アプリケーションに含めると、アセンブルしたときのアプリケーションサイズが大きくなる
- **異なるバージョン** : 複数のクラスローダーが個別に各アプリケーションを読み込むため、実行時に複数のフレームワーククラスのコピーが生成される

複数のアプリケーションが共有できるようにライブラリを設定する方法については、80 ページの「クラスローダー分離の回避」を参照してください。

セッションとセキュリティの問題

セッションを共有する必要がある場合は、セッションにアクセスするすべてのコンポーネントを同じアプリケーションに含める必要があります。

注	複数のアプリケーションによるセッションの共有は、Sun ONE Application Server ではサポートしていません。また、J2EE 仕様書に違反しています。

HTTP セッションを EAR ファイル内の 2 つの WAR ファイル間で共有する場合は、そのセッションを分散させることを配備記述子に記述する必要があります。

クラス、EJB コンポーネントなどのリソースに対する認証されていない実行時アクセスは、禁止してください。コンポーネントは、そのコンポーネントに含まれるほかのリソースに対してアクセス権を持っているクラスだけで構成してください。また、重要なタスクには、J2EE 標準の宣言セキュリティ (第 3 章「J2EE アプリケーションのセキュリティ」を参照) を使用する必要があります。

第 2 章

J2EE アプリケーションの開発

この章では、Sun ONE Application Server 7 でのアプリケーション開発のガイドラインについて説明します。この章には次の節があります。

- 開発環境の設定
- コンポーネントの作成手順

開発環境の設定

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環境の設定は、Sun ONE Application Server のメインストリームバージョン (評価用でないバージョン) をインストールして、開発ツールを利用して行います。これらの作業について、次の各項で説明します。

- 開発用サーバーのインストールと準備
- 開発ツール

開発用サーバーのインストールと準備

Solaris 9 にバンドルの Sun ONE Application Server は、オペレーティングシステムのインストールの一環として同時にインストールされます。詳細については、『Solaris 9 インストールガイド』を参照してください。

それ以外のプラットフォームでは、Sun ONE Application Server のメインストリームバージョン (評価用でないバージョン) をインストールすることをお勧めします。

Solaris 9 にバンドルの製品およびメインストリームバージョンの製品では、デフォルトで次のコンポーネントがインストールされます。

- Sun ONE Application Server コア (次を含む)

    - Sun ONE Message Queue
    - 管理インタフェース
- JDK
- Sun ONE Studio 4
- サンプルアプリケーション

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。

Sun ONE Application Server をインストール後、サーバーを次の方法で開発用にさらに最適化することができます。

- 適切なクラスローダーによってアクセスされるように、ユーティリティクラスおよびライブラリを配備する。詳細は、76 ページの「クラスローダー」を参照
- 動的再読み込みを有効化する。詳細は、97 ページの「動的な再読み込み」を参照
- デバッグを設定する。詳細は、第 5 章「J2EE アプリケーションのデバッグ」を参照
- JVM を設定する。詳細については、『Sun ONE Application Server 管理者ガイド』を参照

開発ツール

次のような一般的なツールが Sun ONE Application Server に付属しています。

- asadmin コマンド
- 管理インタフェース

次のツールは Sun ONE Application Server に付属しているか、または Sun のサイトからダウンロードできます。

- Sun ONE Studio
- Apache Ant
- 移行ツール

次のサードパーティのツールも使用することができます。

- プロファイルツール
- ソースコード制御ツール
- Sun ONE Studio でサポートされるその他のツール

asadmin コマンド

asadmin コマンドを使用すると、ローカルサーバーまたはリモートサーバーの設定と、管理および開発タスクの両方をコマンド行で実行できます。asadmin を使用した配備の詳細は、99 ページの「asadmin コマンド」を参照してください。asadmin についての詳細は、『Sun ONE Application Server 管理者ガイド』を参照してください。

管理インタフェース

管理インタフェースを使用すると、サーバーの設定や、管理および開発タスクの両方を Web ブラウザで実行できます。管理インタフェースを使った配備の詳細は、100 ページの「管理インタフェース」を参照してください。管理インタフェースについての詳細は、『Sun ONE Application Server 管理者ガイド』を参照してください。

Sun ONE Studio

Sun ONE Studio 4 は、Sun ONE Application Server でのコードの作成、アセンブリ、配備およびデバッグを、単一の使いやすいインタフェースで行うことができる IDE (統合開発環境) です。Sun ONE Studio と Sun ONE Application Server は、プラグインにより統合されています。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。

Apache Ant

Ant の自動アセンブリ機能を使用することができます。Ant は、次の Apache Software Foundation のサイトから入手できる Java ベースのビルドツールです。

```
http://jakarta.apache.org/ant/
```

Ant は、Java クラスを使って拡張した Java ベースのビルドツールです。シェルコマンドを使う代わりに、XML ベースの設定ファイルによって、タスクを実行するターゲットツリーが呼び出されるようになっています。各タスクは、特定のタスクインタフェースを実装したオブジェクトによって実行されます。

Sun ONE Application Server (Solaris 9 にバンドルされている場合もバンドルされていない場合も) には Apache Ant 1.4.1 が付属しています。また、Sun ONE Application Server には、サンプルアプリケーションの配備および管理用に、サーバー固有の Ant タスクが複数用意されています。Sun ONE Application Server に付属の Ant の使用方法についての詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照してください。

移行ツール

次の自動移行ツールは、Sun のサイトからダウンロードできます。

- Sun ONE Migration Tool for Application Servers は、次のサーバー上で開発された J2EE アプリケーションおよびモジュールを再アセンブルします。
  - iPlanet Application Server 6.*x*
  - iPlanet Web Server 6.*x*
  - IBM の Websphere Application Server 4.0
  - BEA Systems の WebLogic Server 6.1
- Sun ONE Migration Toolbox を使用すると、NetDynamics および Netscape Application Server 上で開発されたアプリケーションの移行を簡単に行うことができます。

詳細は、『Sun ONE Application Server サーバーアプリケーションの移行および再配備』を参照してください。

プロファイルツール

Sun ONE Application Server では、HPROF、Optimizeit™、Wily Introscope、JProbe™ など、さまざまなプロファイラを使用できます。詳細については、144 ページの「プロファイル」を参照してください。

ソースコード制御ツール

Sun ONE Studio 4 では、次のソースコード制御ツールをサポートしています。

- Concurrent Versioning System (CVS) - Community Edition の組み込みサポート
- PVCS - 事前に定義されている構成
- Visual Source Safe - 事前に定義されている構成
- VCS - API を使って統合が可能
- RCS - API を使って統合が可能
- SCCS - API を使って統合が可能
- Clearcase - API を使って統合が可能

詳細は、次のサイトを参照してください。

`http://wwws.sun.com/software/sundev/jde/features/ce-features.html`

Sun ONE Studio でサポートされるその他のツール

Sun ONE Studio 4 でサポートされるその他のツールのリストについては、次のサイトを参照してください。

`http://forte.sun.com/ffj/partnerprograms/partnerlist.html`

コンポーネントの作成手順

J2EE アプリケーションおよびコンポーネントを作成する前に、25 ページの「J2EE アプリケーションの最適な設計方法」をご覧ください。

この節は、次の基本手順について説明します。

- Web アプリケーションの作成
- Enterprise JavaBeans の作成
- ACC クライアントの作成
- コネクタの作成
- アプリケーションの作成

Web アプリケーションの作成

Web アプリケーションを作成するには

1. Web アプリケーションファイル全体用のディレクトリを作成します。これは、Web アプリケーションのドキュメントルートになります。
2. HTML ファイル、画像ファイルなど、必要な静的コンテンツを作成します。これらのファイルをドキュメントルートディレクトリまたはサブディレクトリに置き、ほかのアプリケーション部分からアクセスできるようにします。
3. 必要な JSP ファイルを作成します。
4. 必要なサーブレットを作成します。
5. サーブレットをコンパイルします。JSP を事前にコンパイルすることもできます。
6. `WEB-INF` ディレクトリ、および Web アプリケーションに必要なその他のディレクトリを作成します。
7. `WEB-INF` ディレクトリに配備記述子ファイル `web.xml` を作成し、オプションで、`sun-web.xml` ファイルを作成します。86 ページの「配備記述子ベリファイア」で説明する方法でこれらのファイルの構造を確認することをお勧めします。
8. WAR ファイルに Web アプリケーションをパッケージ化します。ディレクトリ配備を利用する場合は、この操作はオプションとなります。
9. Web アプリケーションを単独で配備するか、または J2EE アプリケーション内に含めます。

これらの手順に関する詳細は、『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。

Enterprise JavaBeans の作成

EJB コンポーネントを作成するには

1. EJB コンポーネントファイル全体用のディレクトリを作成します。
2. 作成する EJB コンポーネントのタイプを決定します。
   - セッション
     - ステートフル
     - ステートレス
   - エンティティ
     - Bean 管理による持続性を使用
     - コンテナ管理による持続性を使用
   - メッセージ駆動
3. EJB 仕様に準拠して、次の EJB コンポーネントのコードを記述します。
   - ローカル、リモートのいずれか、または両方のホームインタフェース
   - ローカルインタフェース、リモートインタフェースのいずれか、または両方
   - 実装クラス (メッセージ駆動型 Bean 用のものだけ)
4. インタフェースおよびクラスをコンパイルします。
5. `META-INF` ディレクトリ、および EJB コンポーネントの構築に必要なその他のディレクトリを作成します。
6. `META-INF` ディレクトリに配備記述子ファイル `ejb-jar.xml` および `sun-ejb-jar.xml` を作成します。EJB コンポーネントがコンテナ管理による持続性を使用するエンティティ Bean である場合、`.dbschema` ファイルおよび `sun-cmp-mapping.xml` ファイルも作成する必要があります。86 ページの「配備記述子ベリファイア」で説明する方法でこれらのファイルの構造を確認することをお勧めします。
7. JAR ファイルに EJB コンポーネントをパッケージ化します。ディレクトリ配備を利用する場合は、この操作はオプションとなります。
8. EJB コンポーネントを単独で配備するか、または J2EE アプリケーション内に含めます。

これらの手順に関する詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

ACC クライアントの作成

ACC クライアントを作成するには、次の手順に従います。

1. クライアントファイル全体用のディレクトリを作成します。
2. Java 2 Platform Enterprise Edition 仕様に従って、クライアントクラス用のコードを作成します。
3. クライアントのインタフェースおよびクラスをコンパイルします。
4. `META-INF` ディレクトリ、および ACC クライアントの構築に必要なその他のディレクトリを作成します。
5. `META-INF` ディレクトリに配備記述子ファイル `application-client.xml` および `sun-application-client.xml` を作成します。86 ページの「配備記述子ベリファイア」で説明する方法でこれらのファイルの構造を確認することをお勧めします。
6. JAR ファイルにクライアントをパッケージ化します。ディレクトリ配備を利用し、クライアントが 1 つまたは複数の EJB コンポーネントと通信する場合は、この操作は必要に応じて行います。
7. クライアントが 1 つまたは複数の EJB コンポーネントと通信する場合は、クライアントと EJB コンポーネントをアプリケーションにパッケージ化し、そのアプリケーションを配備します。
8. クライアントマシンを準備します。
   a. ACC パッケージ JAR ファイルを作成する
   b. ACC パッケージ JAR ファイルをクライアントマシンにコピーし、`unjar` を実行してファイルを展開する
   c. `sun-acc.xml` ファイルと `asenv.conf` ファイル (Windows 環境では `asenv.bat` ファイル) を設定する
   d. クライアント JAR ファイルをクライアントマシンにコピーする
9. クライアントを実行します。

これらすべての手順に関する詳細は、『Sun ONE Application Server Developer's Guide to Clients』を参照してください。

コネクタの作成

コネクタを作成するには、次の手順に従います。

1. コネクタファイル全体用のディレクトリを作成します。
2. J2EE コネクタアーキテクチャ仕様に従って、リソースアダプタクラス (`ConnectionFactory`、`Connection` など) 用のコードを作成します。
3. コネクタのインタフェースおよびクラスをコンパイルします。
4. コネクタのすべてのクラスを含む 1 つまたは複数の JAR ファイルを作成します。
5. コネクタに必要なネイティブライブラリをディレクトリ構造に追加します。
6. `META-INF` ディレクトリ、およびコネクタに必要なその他のディレクトリを作成します。次に、ディレクトリ構造全体の例を示します。

```
+ MyConnector/
|--- readme.html
|--- ra.jar
|--- client.jar
|--- win.dll
|--- solaris.so
'--+ META-INF/
   |--- MANIFEST.MF
   |--- ra.xml
   '--- sun-ra.xml
```

   `ra.jar` ファイルと `client.jar` ファイルは 手順 4 で作成しました。`win.dll` ファイルと `solaris.so` ファイルは、手順 5 で追加したネイティブライブラリです。

7. `META-INF` ディレクトリに配備記述子ファイル `ra.xml` および `sun-ra.xml` ファイルを作成します。これらのファイルは両方とも必要です。86 ページの「配備記述子ベリファイア」で説明する方法でこれらのファイルの構造を確認することをお勧めします。
8. 手順 6 で説明したディレクトリ構造から RAR ファイルを作成します。ディレクトリ配備を利用する場合は、この操作はオプションとなります。
9. コネクタを単独で配備するか、または J2EE アプリケーション内に含めます。

これらすべての手順に関する詳細は、『Sun ONE Application Server J2EE CA Service Provider Implementation 管理者ガイド』を参照してください。

注 Sun ONE Connector Builder を使えば、コネクタを簡単に作成できます。詳細については、『Sun ONE Connector Builder 開発者ガイド』を参照してください。

アプリケーションの作成

完全な J2EE アプリケーションを作成するには、次の手順に従います。

1. アプリケーションを構成するコンポーネント (Web アプリケーション、EJB コンポーネント、およびコネクタ) を決定します。
2. アプリケーションファイル全体用のディレクトリを作成し、その下に `META-INF` ディレクトリを作成します。
3. アプリケーションのコンポーネントを作成し、それらをアプリケーションディレクトリにコピーします。`.jar`、`.war`、または `.rar` ファイルが、それぞれ `_jar`、`_war`、または `_rar` という名前の空きサブディレクトリに置かれる必要があります。
4. コンポーネントが互いに正しく呼び出し合うことを確認します。
5. アプリケーションディレクトリの下の `META-INF` ディレクトリに、配備記述子ファイル `application.xml` を作成し、オプションで、`sun-application.xml` ファイルを作成します。86 ページの「配備記述子ベリファイア」で説明する方法でこれらのファイルの構造を確認することをお勧めします。
6. EAR ファイルにアプリケーションをパッケージ化します。ディレクトリ配備を利用する場合は、この操作はオプションとなります。
7. アプリケーションを配備します。

これらすべての手順の詳細は、第 4 章「J2EE アプリケーションのアセンブルと配備」を参照してください。

第 3 章

J2EE アプリケーションのセキュリティ

この章では、安全な J2EE アプリケーションを記述する方法について説明します。安全な J2EE アプリケーションには、サーブレットと EJB ビジネスロジックのユーザー認証とアクセス承認を実行するコンポーネントが含まれます。

サーバーの管理セキュリティの詳細は、『Sun ONE Application Server Administrator's Guide to Security』を参照してください。

この章には次の節があります。

- Sun ONE Application Server のセキュリティの目標
- Sun ONE Application Server 固有のセキュリティ機能
- Sun ONE Application Server のセキュリティモデル
- セキュリティ責任の概要
- セキュリティの一般的な用語
- コンテナセキュリティ
- セキュリティ情報のガイド
- レルムの設定
- `server.policy` ファイル
- プログラムによるログイン

Sun ONE Application Server のセキュリティの目標

企業のコンピューティング環境には、多くのセキュリティ上のリスクがあります。Sun ONE Application Server の目標は、J2EE セキュリティモデルに基づいて、安全性の高い相互利用可能な分散コンポーネントコンピューティング環境を実現することです。Sun ONE Application Server のセキュリティの目標は次のとおりです。

- J2EE セキュリティモデルへの完全準拠 (詳細は、J2EE 仕様書バージョン 1.3 の第 3 章「Security」を参照)
- EJB v2.0 セキュリティモデルへの完全準拠 (詳細は、Enterprise JavaBeans 仕様書バージョン 2.0 の第 15 章「Security Management」を参照)。EJB ロールベースの承認も含まれる。
- Java Servlet v2.3 セキュリティモデルへの完全準拠 (詳細は、Java Servlet 仕様書バージョン 2.3 の第 11 章「Security」を参照)。サーブレットロールベースの承認も含まれる
- 1 つのセキュリティドメイン内にあるすべての Sun ONE Application Server アプリケーションへの単一サインオンをサポート
- ACC クライアントのセキュリティをサポート
- 簡易ファイル、LDAP など、基礎となっている複数の認証領域をサポート。SSL クライアント認証用に、証明書認証もサポートされている。Solaris では、これらに加えて OS プラットフォーム認証もサポートしている
- Sun ONE Application Server 固有の XML ベースのロールマッピングを利用して宣言型セキュリティをサポート

Sun ONE Application Server 固有のセキュリティ機能

Sun ONE Application Server は、J2EE v1.3 セキュリティモデルだけでなく、次の Sun ONE Application Server 固有の機能をサポートします。

- 1 つのセキュリティドメイン内にあるすべての Sun ONE Application Server アプリケーションへの単一サインオン
- プログラムによるログイン

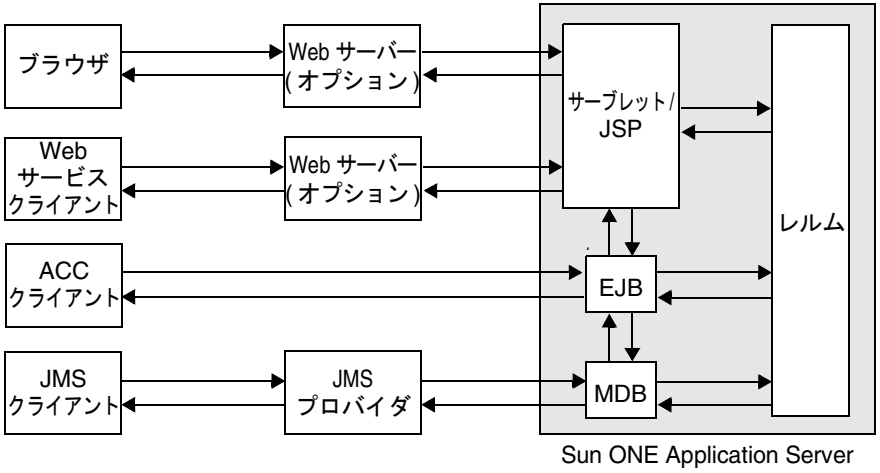
Sun ONE Application Server のセキュリティモデル

安全なアプリケーションでは、クライアントが、有効なアプリケーションユーザーとして認証されていて、サーブレット、JSP、および EJB ビジネスロジックにアクセスすることを承認されている必要があります。Sun ONE Application Server では、Web、ACC、Web サービス、および JMS クライアントのセキュリティをサポートしています。

安全な Web コンテナと安全な EJB コンテナを使用したアプリケーションは、クライアント用に次のセキュリティプロセスを実行できます。

- 呼び出し側を認証する
- 呼び出し側に EJB ビジネスメソッドへのアクセスを承認する

次の図に Sun ONE Application Server セキュリティモデルを示します。



Web アプリケーションと URL の承認

安全な Web アプリケーションは認証と承認のためのプロパティを持つことができます。Web コンテナは、3 種類の認証タイプ (基本、証明書、およびフォームベース) をサポートしています。ブラウザがメインアプリケーションの URL を要求すると、Web コンテナはユーザー認証情報 (たとえば、ユーザー名とパスワード) を収集し、それを認証用のセキュリティサービスに転送します。

Sun ONE Application Server は Web リソースに関連付けられたセキュリティポリシー (配備記述子から派生する) を調べ、リソースへのアクセスを許可するセキュリティロールを調べます。Web コンテナは各ロールに対してユーザーの証明書をテストし、ロールにユーザーを割り当てることができるかどうかを判断します。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。

エンタープライズ Bean メソッドの呼び出し

ブラウザが Web コンテナによって認証および承認され、JSP が EJB コンポーネントのリモートメソッド呼び出しを実行すると、認証プロセスで収集されたユーザーの証明書は、EJB コンテナへ伝達されます。安全な EJB コンテナには、Bean メソッドにアクセス制御を適用するために使用される承認プロパティを持つ、配備記述子が含まれています。EJB コンテナは、EJB JAR 配備記述子から取得したロール情報を使用して、呼び出し側をロールにマップすることができるかどうか、また、Bean メソッドへのアクセスを許可するかどうかを判断します。詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

ACC クライアントによるエンタープライズ Bean メソッドの呼び出し

ACC クライアントの場合、安全な EJB コンテナはセキュリティポリシー (配備記述子から取得) を調べて、呼び出し側が Bean メソッドへのアクセス権限を持っているかどうかを判断します。このプロセスは、ACC クライアントおよび Web クライアントで同じです。詳細は、『Sun ONE Application Server Developer's Guide to Clients』を参照してください。

セキュリティ責任の概要

J2EE プラットフォームの主な目標は、セキュリティメカニズムから開発者を解放し、安全なアプリケーションをさまざまな環境に容易に配備できるようにすることです。この目標を達成するには、アプリケーションセキュリティの仕様要件のメカニズムをアプリケーションの外側に明確に設定する必要があります。

アプリケーション開発者

アプリケーション開発者は次の項目に対して責任があります。

- アプリケーションロールを指定する
- アプリケーションコンポーネント (サーブレット /JSP および EJB コンポーネント) に対するロールベースのアクセス制限を定義する
- プログラムによるセキュリティを使用する場合は、ユーザーロールを確認し、これらのロールに基づいて機能へのアクセスを承認する (プログラムによるセキュリティ管理では、コンテナで管理する代わりに、アプリケーション内にセキュリティログインがハードコードされるため、お勧めできません)

アプリケーション編成者

アプリケーション編成者またはアプリケーションコンポーネントプロバイダは、コンポーネントに組み込まれた次のようなセキュリティ関連事項をすべて確認する必要があります。

- コンポーネントが `isCallerInRole` または `isUserInRole` を呼び出すときに使うすべてのロール名
- コンポーネントがアクセスするすべての外部リソースへの参照
- コンポーネントが行うすべての内部コンポーネント呼び出しへの参照

アプリケーション配備者

アプリケーション配備者は編成者が提供したすべてのコンポーネントのセキュリティビューを受け取り、それを使用して次のようにアプリケーションにおける特定の企業環境を保護します。

- ユーザーまたはグループ、あるいはその両方をセキュリティロールに割り当てる
- コンポーネントメソッドへのアクセスに必要な権限を修正して、特定の配備シナリオの要件に適合させる

セキュリティの一般的な用語

最も一般的なセキュリティプロセスは、認証、承認、レルムの割り当て、およびロールマッピングです。次の節でこれらの用語を定義します。

認証

認証とは、ユーザーを確認することです。たとえば、ユーザーが Web ブラウザ内でユーザー名とパスワードを入力し、その証明書がアクティブなレルム内に保存されているパーマネントプロファイルと一致したとき、ユーザーは認証されます。ユーザーは、それ以降のセッションで使われるセキュリティ ID に関連付けられます。

Sun ONE Application Server の認証に関する詳細は、『Sun ONE Application Server Web アプリケーション開発者ガイド』および『Sun ONE Application Server Developer's Guide to Clients』を参照してください。

承認

認証された後、承認によってユーザーは希望する操作を実行することができます。たとえば、人事管理アプリケーションでは、管理者には社員全員の個人情報を見ることを承認し、社員には自身の個人情報だけを見ることを承認します。

Sun ONE Application Server の認証に関する詳細は、『Sun ONE Application Server Web アプリケーション開発者ガイド』および『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

レルム

レルム (J2EE 仕様でセキュリティポリシードメインまたはセキュリティドメインとも呼ばれている) とは、共通のセキュリティポリシーが、セキュリティサービスのセキュリティ管理者によって定義および適用される範囲のことです。Sun ONE Application Server でサポートされているレルムは、`file`、`ldap`、`certificate`、および `solaris` です。レルムの設定方法については、51 ページの「レルムの設定」を参照してください。

ロールマッピング

クライアントはセキュリティロールによって定義できます。たとえば、会社が社員のデータベースを使用して、社内電話帳アプリケーションと支払給与情報の両方を生成するとします。電話番号と電子メールアドレスには、すべての社員がアクセスできますが、給与情報にアクセスできるのは一部の社員に限られます。給与を表示あるいは変更する権限を持つ社員は、特別なセキュリティロールを持つように定義できます。

ロールはアプリケーション内での役割を定義するのに対し、グループはある方法で関連付けられているユーザーの集まりに過ぎません。この点で、ロールとユーザーグループは異なります。たとえば、*astronauts*、*scientists*、および場合によっては *politicians* というグループのメンバーはすべて、*SpaceShuttlePassenger* のロールに該当します。

EJB セキュリティモデルは、アプリケーション開発者の記述どおりに、特定のドメインとは関係なくロール (ユーザーグループとは区別された) を記述します。グループは配備ドメインに固有です。各アプリケーションまたはモジュール内での 1 つまたは複数のグループへのロールの割り当ては、配備者が決定します。

Sun ONE Application Server では、ロールは、アクティブなレルムに設定されているユーザーまたはグループ (あるいはその両方) に対応しています。

コンテナセキュリティ

コンポーネントコンテナは、J2EE アプリケーションのセキュリティを確保する役目を果たします。コンテナによって確保されるセキュリティ形式には、次の 2 つがあります。

- プログラムによるセキュリティ
- 宣言によるセキュリティ

プログラムによるセキュリティ

プログラムによるセキュリティとは、EJB コンポーネントまたはサーブレットが J2EE セキュリティモデルによって指定されたセキュリティ API へのメソッド呼び出しを使用して、呼び出し側またはリモートユーザーのセキュリティロールに基づいて、ビジネスロジックの決定を行うことです。プログラムによるセキュリティは、宣言によるセキュリティ単独ではアプリケーションのセキュリティモデルの要求を十分に満たすことができない場合に使います。

J2EE 仕様書バージョン 1.3 では、プログラムによるセキュリティは、EJB の `EJBContext` インタフェースの 2 つのメソッドおよびサーブレットの `HttpServletRequest` インタフェースの 2 つのメソッドで構成されるものと定義されています。Sun ONE Application Server では、この仕様書で示されているように、これらのインタフェースをサポートしています。

プログラムによるセキュリティの詳細は、次を参照してください。

- J2EE 仕様書バージョン 1.3 の第 3.3.6 節「Programmatic Security」
- 62 ページの「プログラムによるログイン」

宣言によるセキュリティ

宣言によるセキュリティとは、アプリケーションのセキュリティメカニズムが、そのアプリケーションの外部で宣言され、処理されることを意味します。配備記述子には、アプリケーションのセキュリティロール、アクセス制御、認証要件など、J2EE アプリケーションのセキュリティ構造が示されています。

Sun ONE Application Server は、J2EE v1.3 が指定する DTD をサポートしており、さらに別のセキュリティ要素が配備記述子に含まれています。宣言によるセキュリティの責任は、アプリケーション配備者にあります。詳細は、第 4 章「J2EE アプリケーションのアセンブルと配備」を参照してください。

宣言によるセキュリティには、次の 3 つのレベルがあります。

- アプリケーションレベルのセキュリティ
- Web コンポーネントレベルのセキュリティ
- EJB レベルのセキュリティ

アプリケーションレベルのセキュリティ

アプリケーションの XML 配備記述子 (`sun-application.xml`) には、アプリケーションのサーブレットおよび EJB コンポーネントにアクセスするときに使用される、すべてのユーザーロール用の承認記述子が含まれています。アプリケーションレベルでは、アプリケーションのコンテナが使用するすべてのロールがこのファイルの `role-name` 要素のリストに含まれている必要があります。ロール名は、EJB XML 配備記述子 (`ejb-jar.xml` および `sun-ejb-jar.xml` ファイル) およびサーブレット XML 配備記述子 (`web.xml` および `sun-web.xml` ファイル) の適用対象となります。`sun-application.xml` ファイルには、アプリケーションで使用される各 `role-name` にマッチングする `security-role-mapping` 要素も含まれている必要があります。

Web コンポーネントレベルのセキュリティ

安全な Web コンテナは、サーブレット XML 配備記述子 (`web.xml` および `sun-web.xml` ファイル) に記述されているセキュリティポリシーを使用して、ユーザーを認証し、サーブレットまたは JSP へのアクセスを認可します。ユーザーが認証され承認されると、サーブレットはユーザーの証明書を EJB コンポーネントに転送して Bean との安全な関連付けを確立します。

EJB レベルのセキュリティ

EJB コンテナは、EJB XML 配備記述子 (`ejb-jar.xml` および `sun-ejb-jar.xml` ファイル) に記述されているセキュリティポリシーを使用して、Bean メソッドへのアクセスを承認する役割を果たします。

セキュリティ情報のガイド

次の種類の各情報を、短い説明、情報の保存場所、情報の作成方法、情報へのアクセス方法、および詳しい情報を参照できる場所とともに示します。

- ユーザー情報
- セキュリティロール

ユーザー情報

ユーザー名、パスワードなどです。

場所

ユーザー情報の場所は、使用されているレルムによって異なります。

- `file` レルムの場合は、*instance_dir*`/config` ディレクトリ内にあるキーファイルにユーザーおよびグループが含まれている
- `ldap` レルムの場合は、外部の LDAP ディレクトリにユーザーおよびグループが格納される
- `certificate` レルムの場合は、暗号を使って確認されたクライアント証明書からユーザー ID が取得される
- `solaris` の場合は、ユーザーおよびグループは、システムの PAM 設定で指定されているとおりに、基礎となっている Solaris ユーザーデータベース内に格納される

これらの領域の詳細は、51 ページの「レルムの設定」を参照してください。

作成方法

ユーザーの作成方法およびグループの定義方法は、使用されているレルムに固有のものです。Sun ONE Application Server には、Solaris/PAM または LDAP などの外部レルムのための管理機能はありません。詳細は、Solaris または LDAP のマニュアルを参照してください。

セキュリティロール

アプリケーションの機能を定義するロールで、多数のユーザーまたはグループ、あるいはその両方から構成されます。ユーザーとグループの関係は、使用されている特定の領域の実装によって決定されます。

場所

ロールは、J2EE アプリケーション配備記述子内で定義されます。

作成方法

Sun ONE Application Server の管理インタフェース、または Sun ONE Studio 4 のアプリケーションアセンブリ用および配備用の開発ツールを使います。

アクセス方法

`isCallerInRole()` を使用してユーザーのロールメンバーシップをテストします。たとえば、次のコードで `Manager` ロールが `securedMethod()` にアクセスできる場合は、`sctx.isCallerInRole("Manager")` を呼び出すと `true` が返されます。

```
public class SecTestEJB implements SessionBean
{
   private SessionContext sctx = null;

   public void setSessionContext(SessionContext sc)
   {
      sctx = sc;
   }

   public void securedMethod( )
   {
      System.out.println(  sctx.isCallerInRole( "Manager" ) );
   }
}
```

レルムの設定

この節には次のトピックがあります。

- レルムの設定方法
- サポートされるレルム

レルムの設定方法

次の方法のいずれかで領域を設定できます。

- 管理インタフェースの使用法
- `asadmin` コマンドの使用
- `server.xml` ファイルの編集

管理インタフェースの使用法

管理インタフェースを使用して領域を設定するには

1. サーバーインスタンスの下の「セキュリティ」コンポーネントを開きます。
2. 「セキュリティ」コンポーネントの下の「レルム」コンポーネントを開きます。
3. 「レルム」ページに移動します。
4. 有効にしたいレルムのチェックボックスをクリックします。レルムの追加と削除は、「新規」および「削除」ボタンを使用しても行うことができます。レルムを編集するときは、そのレルムをクリックします。
5. レルムを追加または編集するときは、レルムの名前、クラス名、プロパティ、ユーザー (`file` レルムのみ) を入力し、「了解」または「保存」ボタンを選択します。
6. 「セキュリティ」ページに移動します。
7. デフォルトのレルムを選択し、「保存」ボタンを選択します。
8. 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。
9. サーバーを再起動します。

asadmin コマンドの使用

ローカルサーバー上にレルムを設定するときは、`asadmin` コマンドを使用することができます。

asadmin create-auth-realm

`asadmin create-auth-realm` コマンドは、レルムを設定します。構文は次のとおりです。

```
asadmin create-auth-realm --user admin_user [--password admin_password]
[--passwordfile password_file] [--host hostname] [--port adminport]
[--secure | -s] [--instance instance_name] --classname realm_class
[--property (name=value)[:name=value]*] realm_name
```

たとえば、次のコマンドは、`certificate` レルムを設定します。

```
asadmin create-auth-realm --user jadams --password secret --instance
server1 --classname
com.iplanet.ias.security.auth.realm.certificate.CertificateRealm
certificate
```

asadmin delete-auth-realm

`asadmin delete-auth-realm` コマンドは、レルムを無効にします。構文は次のとおりです。

```
asadmin delete-auth-realm --user admin_user [--password admin_password]
[--passwordfile password_file] [--host hostname] [--port adminport]
[--secure | -s] [--instance instance_name] realm_name
```

たとえば、次のコマンドは、`certificate` レルムを無効にします。

```
asadmin delete-auth-realm --user jadams --password secret --instance
server1 certificate
```

asadmin list-auth-realms

`asadmin list-auth-realms` コマンドは、すべてのレルムを一覧表示します。構文は次のとおりです。

```
asadmin list-auth-realms --user admin_user [--password admin_password]
[--passwordfile password_file] [--host hostname] [--port adminport]
[--secure | -s] instance_name
```

たとえば、次のコマンドは、`server1` インスタンス内のすべてのレルムを一覧表示します。

```
asadmin list-auth-realms --user jadams --password secret server1
```

server.xml ファイルの編集

デフォルトのレルムは、バックグラウンドで、`server.xml` 内の `security-service` 要素に設定されます。`security-service` の設定は、次のようになります。

```
<security-service default-realm="file" anonymous-role="ANYONE"
      audit-enabled="false">
   <auth-realm name="file"
         classname="com.iplanet.ias.security.auth.realm.file.FileRealm">
      <property name="file" value="instance_dir/config/keyfile"/>
      <property name="jaas-context" value="fileRealm"/>
   </auth-realm>
   ...
</security-service>
```

default-realm 属性は、サーバーが使用しているレルムを指します。この属性は、設定済みの auth-realm 名のいずれかを指す必要があります。デフォルトは、file レルムです。

audit フラグは、監査情報をログに記録するかどうかを決定します。true に設定すると、サーバーはすべての認証および承認イベントに対する監査メッセージをログに記録します。

レルムの設定を変更した場合は、変更を有効にするためにサーバーを再起動する必要があります。

server.xml ファイルの詳細は、『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。

サポートされるレルム

Sun ONE Application Server では、次のレルムがサポートされています。

- file
- ldap
- certificate
- solaris
- カスタムレルムの作成

file

file レルムは、Sun ONE Application Server を最初にインストールした場合のデフォルトのレルムです。このレルムは次のように設定されています。

- 名前 - file
- クラス名 - com.iplanet.ias.security.auth.realm.file.FileRealm

必須プロパティは次のとおりです。

- file - ユーザー情報を格納するファイルの名前。デフォルトでは *instance_dir*/config/keyfile
- jaas-context - 値は fileRealm にする必要がある

ユーザー情報ファイルは最初は空のため、ユーザーを追加しなければ file レルムを使用できません。次の方法のいずれかでユーザーを設定できます。

- 管理インタフェースの使用法
- asadmin コマンドの使用

管理インタフェースの使用法

管理インタフェースを使って file レルムのユーザーを設定するには、次の手順に従います。

1. サーバーインスタンスの下の「セキュリティ」コンポーネントを開きます。
2. 「セキュリティ」コンポーネントの下の「レルム」コンポーネントを開きます。
3. 「file」ページに移動します。
4. 「ユーザーを管理」ボタンをクリックします。
5. ユーザーを新たに追加するときは、「新規」ボタンをクリックします。ユーザーの情報を変更するときは、表示されているユーザー名をクリックします。どちらの場合も次の情報を入力します。
   - ユーザー ID (新規の場合に必要) - ユーザーの名前
   - パスワード (必須) - ユーザーのパスワード
   - パスワードの再入力 (必須) - ユーザーのパスワードの確認用再入力
   - グループリスト (オプション) - ユーザーが属するグループのコンマで区切られたリスト
6. 「了解」ボタンをクリックします。
7. 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。
8. サーバーを再起動します。

asadmin コマンドの使用

asadmin create-file-user コマンドは、file レルムにユーザーを 1 つ作成します。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 create-file-user --user admin_user [--password admin_password]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] [--instance instance_name] [--userpassword user_password]
[--groups user_group[:user_group]*] user_name
```

たとえば、次のコマンドは dsanchez というユーザーを file レルムに追加し、ユーザーのパスワードを secret に設定します。jadams と topsecret は、管理者の名前とパスワードです。

```
asadmin create-file-user --user jadams --password topsecret
--userpassword secret dsanchez
```

asadmin delete-file-user コマンドは、file レルムからユーザーを 1 つ削除します。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 delete-file-user --user admin_user [--password admin_password]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] [--instance instance_name] user_name
```

たとえば、次のコマンドは file レルムから dsanchez というユーザーを削除します。jadams と topsecret は、管理者の名前とパスワードです。

```
asadmin delete-file-user --user jadams --password topsecret dsanchez
```

asadmin update-file-user コマンドは、file レルムの 1 つのユーザーの情報を変更します。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 update-file-user --user admin_user [--password admin_password]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] [--instance instance_name] [--userpassword user_password]
[--groups user_group[:user_group]*] user_name
```

たとえば、次のコマンドはユーザーのパスワードを dsanchez から private に変更します。jadams と topsecret は、管理者の名前とパスワードです。

```
asadmin update-file-user --user jadams --password topsecret
--userpassword private dsanchez
```

asadmin list-file-users コマンドは、file レルムのユーザーをリスト表示します。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 list-file-users --user admin_user [--password admin_password]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] instance_name
```

たとえば、次のコマンドは server1 インスタンスの file レルムに含まれるユーザーをリスト表示します。jadams と topsecret は、管理者の名前とパスワードです。

```
asadmin list-file-users --user jadams --password topsecret server1
```

asadmin list-file-groups コマンドは、file レルムのグループをリスト表示します。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 list-file-groups --user admin_user [--password admin_password]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] [--name user_name] instance_name
```

たとえば、次のコマンドは dsanchez というユーザーの file レルムグループをリスト表示します。jadams と topsecret は、管理者の名前とパスワードです。

```
asadmin list-file-users --user jadams --password topsecret --name
dsanchez server1
```

ldap

ユーザーセキュリティ情報の LDAP データベースを使うときは、ldap レルムを使います。このレルムは次のように設定されています。

- 名前 - ldap
- クラス名 - com.iplanet.ias.security.auth.realm.ldap.LDAPRealm

必須プロパティは次のとおりです。

- directory - 使用しているサーバーの LDAP URL
- base-dn - ユーザーデータの場所のベース DN。ツリー範囲検索が実行されるため、このベース DN は、ユーザーデータのレベルより上に置かれる。検索ツリーが小さければ小さいほど、パフォーマンスは良くなる
- jaas-context - 値は ldapRealm にする必要がある

LDAP 領域の動作を調整するために、オプションで、次のプロパティを追加できます。

- search-filter - ユーザーを見つけるために使用する検索フィルタ。デフォルトは、uid=%s (%s はサブジェクト名に展開される)
- group-base-dn - グループデータの場所のベース DN。デフォルトでは、base-dn と同じ。ただし、必要に応じて変更できる
- group-search-filter - ユーザーのグループメンバーシップを検索するために使用する検索フィルタ。デフォルトは、uniquemember=%d (%d はユーザー要素 DN に展開される)
- group-target - グループ名エントリを含む LDAP の属性名。デフォルトは、CN
- search-bind-dn - search-filter 検索を実行するディレクトリの認証に使用するオプション DN。匿名検索を実行できないディレクトリだけで必要とされる
- search-bind-password - search-bind-dn で指定した DN の LDAP パスワード

必要なユーザーを LDAP ディレクトリに作成する必要があります。作成は、「Users & Groups」メインタブの Sun ONE Directory Server コンソールで行うか、LDAP とディレクトリスキーマをサポートするほかの管理ツールを使用して行うことができます。

配備記述子内で使用される principal-name は、LDAP ユーザー情報に対応している必要があります。

certificate

certificate レルムは、SSL 認証をサポートしています。certificate レルムでは、Sun ONE Application Server のセキュリティコンテキストにユーザー ID が設定され、クライアント証明書からユーザーデータがそのユーザー ID に入力されます。その後、J2EE コンテナは、証明書からの各ユーザーの DN に基づいて、承認処理を行います。このレルムは次のように設定されています。

- 名前 - certificate
- クラス名 - com.iplanet.ias.security.auth.realm.certificate.CertificateRealm

certificate レルムの動作を調整するために、オプションで、次のプロパティを追加できます。

- assign-groups - このプロパティを設定すると、値は各グループ名がコンマで区切られたリストとなる。有効な証明書を持つすべてのクライアントには、これらのグループのメンバーシップが割り当てられ、Web コンテナと EJB コンテナの承認に使用される

アプリケーションを配備するときは、次のように、web.xml 内に認証メカニズムとして CLIENT-CERT を指定する必要があります。

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

クライアント証明書を取得し、それをブラウザにインストールして、クライアント証明書の認証機能のセットアップを完了する必要があります。サーバーおよびクライアント証明書のセットアップ方法に関する詳細は、『Sun ONE Application Server Administrator's Guide to Security』を参照してください。

次のいずれかの方法で、SSL 認証用にサーバーインスタンスを設定できます。

- server.xml ファイルの ssl 要素を設定し、サーバーを再起動します。server.xml ファイルの詳細は、『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。
- 次のように管理インタフェースを使用します。
  a. サーバーインスタンスの下の「HTTP サーバー」コンポーネントを開きます。
  b. 「HTTP サーバー」コンポーネントの下の「HTTP リスナー」コンポーネントを開きます。
  c. SSL を設定するリスナーを選択します。

d. そのリスナー用の「SSL/TLS 設定」を編集します。

e. 「Save」ボタンを選択します。

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。

g. サーバーを再起動します。

詳細は、『Sun ONE Application Server Administrator's Guide to Security』を参照してください。

solaris

`solaris` レルムでは、Solaris の `username+password` データを使用して認証を行うことができます。このレルムは Solaris 9 だけでサポートされています。このレルムは次のように設定されています。

- 名前 - `solaris`
- クラス名 - `com.iplanet.ias.security.auth.realm.file.SolarisRealm`

必須プロパティは次のとおりです。

- `jaas-context` - 値は `solarisRealm` にする必要がある

注 solaris レルムは、認証のために基礎となる PAM インフラストラクチャを呼び出します。PAM モジュールの設定が root 権限を必要とする場合は、このレルムを使用するには root としてインスタンスを実行する必要があります。詳細は、『Solaris のシステム管理 (セキュリティサービス)』の「認証サービスの使用 (手順)」の章を参照してください。

Solaris は、リソースへのアクセスを制御する RBAC (Role Based Access Control) という拡張メソッドをサポートしています。Sun ONE Application Server の管理コマンドには RBAC が含まれます。詳細は、『Solaris のシステム管理 (セキュリティサービス)』の「役割によるアクセス制御 (概要)」の章を参照してください。

カスタムレルムの作成

JAAS (Java Authentication and Authorization Service) ログインモジュールとレルムの実装を用意することで、カスタムレルムを作成できます。クライアントサイドの JAAS ログインモジュールは、Sun ONE Application Server には適していません。JAAS に関する詳細は、次のサイトで Java 2 SDK, v 1.4 用の JAAS 仕様書を参照してください。

`http://java.sun.com/products/jaas/`

カスタムレルムを使用するサンプルアプリケーションは、Sun ONE Application Server の次のディレクトリにあります。

install_dir/samples/security/realms

server.policy ファイル

Sun ONE Application Server の各インスタンスは、*instance_dir*/config ディレクトリに、専用の J2SE 標準ポリシーファイルを持っています。ファイル名は、server.policy です。

Sun ONE Application Server 7 は J2EE 1.3 に準拠したアプリケーションサーバーです。J2EE 仕様の推奨や要件に従っているため、ポリシーを適用する Java コンポーネントであるセキュリティマネージャを装備し、J2EE アプリケーションコードへのアクセス権は限定されています。

この節には次のトピックがあります。

- デフォルトのアクセス権
- アプリケーションのアクセス権の変更
- セキュリティマネージャの無効化

デフォルトのアクセス権

内部サーバーコードにはすべてのアクセス権が付与されています。これは、サーバーインフラストラクチャコードの各種部分に AllPermission 付与ブロックを加えることで適用されています。このエントリを変更しないでください。

デフォルトの付与ブロックにはアプリケーションのアクセス権が付与されています。このアクセス権は、前述の内部サーバーコードの一部ではなく、すべてのコードに適用されます。Sun ONE Application Server 7 は、EJB モジュールと Web モジュールのアクセス権を区別しません。すべてのコードには、Web コンポーネントのアクセス権の最小セット (EJB の最小セットの上位集合) が付与されます。

server.policy ファイルには、最小セットのほかに少数のアクセス権が付与されています。これは、サーバー実装の各種内部依存に対応するために必要なアクセス権です。J2EE アプリケーションの開発者は、これらの追加アクセス権に依存した開発はするべきではありません。

コネクタを使用するためにアクセス権が 1 つ追加されています。特定のサーバーインスタンスでコネクタを使用しない場合は不要なので、このアクセス権を削除することをお勧めします。

アプリケーションのアクセス権の変更

各インスタンスのデフォルトポリシーは、配備された J2EE アプリケーションのアクセス権を、アプリケーションが正しく動作するのに最低限必要なアクセス権に制限しています。このデフォルトのアクセス権以上のアクセス権を必要とするアプリケーションを開発している場合は、server.policy ファイルを編集して、そのアプリケーションに必要なカスタムのアクセス権を追加します。

サーバーインスタンスに配備するすべてのアプリケーションではなく、追加が必要なアプリケーションだけに適切なアクセス権を追加します。デフォルトセット (すべてのコードに適用される、コードベースを持たない付与ブロック) にアクセス権を追加しないでください。その代わりに、アクセス権の追加が必要なアプリケーションには、コードベースを持つ新しい付与ブロックを追加します。この付与ブロックにも、必要最小限のアクセス権だけを追加します。

注 server.policy ファイルには、アプリケーションコード用に java.security.AllPermission を追加しないでください。追加すると、セキュリティマネージャの機能が完全に無効化され、パフォーマンスのオーバーヘッドも伴います。その代わりに、61 ページの「セキュリティマネージャの無効化」で説明する手順でセキュリティマネージャを無効化してください。

J2EE 仕様に規定されているように、追加したアクセス権に関する説明をアプリケーションに用意する必要があります。アクセス権の追加が必要なアプリケーションで、追加するアクセス権のセットが説明されていない場合は、アプリケーションの作成者に詳細を確認してください。

最後の手段として、サーバーログファイルを開いて AccessControlException を検索し、アプリケーションに必要なアクセス権を調べることもできます。この方法が適さない場合は、サーバーインスタンスに -Djava.security.debug=all JVM オプションを追加します。詳細は、『Sun ONE Application Server 管理者ガイド』または『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.policy ファイルを編集するには、J2SE 標準 policytool、またはテキストエディタを使用します。詳細は、次のサイトを参照してください。

http://java.sun.com/docs/books/tutorial/security1.2/tour2/index.html

server.policy ファイル内に設定するアクセス権の詳細は、次のサイトを参照してください。

http://java.sun.com/j2se/1.4/docs/guide/security/permissions.html

Permission クラスの Javadoc は、次のサイトにあります。

http://java.sun.com/j2se/1.4/docs/api/java/security/Permission.html

セキュリティマネージャの無効化

J2EE アプリケーションコンポーネントの開発者は、セキュリティマネージャを無効にするべきではありません。J2EE 仕様に規定されているように、J2EE アプリケーションはデフォルトセットで実行可能である必要があり、特別な要件についてはコンポーネントに説明を加える必要があります。

注 J2EE アプリケーションが J2EE のポリシーセットに確実に準拠するには、J2EE 開発者がセキュリティマネージャを無効にしないことが重要です。Sun ONE Application Server 7 にはセキュリティマネージャを無効にするオプションが用意されていますが、このオプションは将来廃止される可能性があります。J2EE のポリシーセットに違反する (および相違点が説明されていない) アプリケーションは、将来のリリースでは実行できないことがあります。

本稼動環境では、次の場合にセキュリティマネージャを安全に無効化することができます。

- パフォーマンスの影響を受けやすい
- 本稼動サーバーへの配備が慎重に制御されている
- 信頼できるアプリケーションだけが配備されている
- アプリケーションにポリシーを適用する必要がない

一部のアプリケーションでは、セキュリティマネージャを無効化するとパフォーマンスが大きく向上することがあります。セキュリティマネージャを無効にするには、`server.xml` ファイル内の次のエントリを削除するかコメントアウトします。

```
<jvm-options>-Djava.security.policy=instance_dir/config/server.policy</jvm-options>
```

`server.xml` ファイルの詳細は、『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。

プログラムによるログイン

プログラムによるログインでは、配備された J2EE アプリケーションによりログインメソッドが呼び出されます。ログインが成功すると、あたかもクライアントが従来のいずれかの J2EE メカニズムを使用して認証したかのように、`SecurityContext` が確立されます。

プログラムによるログインは、J2EE 標準認証メカニズムを適用できない特別なニーズを持つアプリケーションに有効です。

注	プログラムによるログインは、Sun ONE Application Server に固有の機能です。他のアプリケーションサーバーには移植できません。

この節には次の項目があります。

- 対策
- プログラムによるログインのアクセス権の付与
- ProgrammaticLogin クラス

対策

Sun ONE Application Server は、ログイン情報 (`user`、`password`) が配備アプリケーションによってどのように取得されるかということには関与していません。プログラムによるログインを使用する場合、セキュリティ要件を満たすシステムを作成するようアプリケーション開発者に責任が課せられます。アプリケーションコードがネットワーク上の認証情報を読み込む場合、ユーザーを信用するかどうかの判断はそのアプリケーションに委ねられます。

プログラムによるログインを使用する場合、アプリケーション開発者は、アプリケーションサーバーでサポートされている認証メカニズムを使用せずに、セキュリティサービスに直接認証データを渡すことができます。この機能は、柔軟性がありますが、セキュリティに関する事項を十分理解した上で使用してください。

このメカニズムは、コンテナ管理認証プロセスおよびシーケンスを迂回するため、アプリケーション開発者は、制限されたリソースまたはメソッドにアクセスするときはその前に必ず認証が確立されるようにする必要があります。ログイン試行の状態を確認し、それに応じてアプリケーションの動作を変更することも、アプリケーション開発者の責任です。

プログラムによるログインの状態は、必ずしも複数のセッションにわたって持続したり、シングルサインオンに関係しているわけではありません。

遅延認証は、プログラムによるログインではサポートしていません。アクセスチェックのときに、配備アプリケーションがプログラムによるログインメソッドを使用して正しく認証していないことが判明した場合は、アクセスはただちに拒否され、また、このような事態に備えてアプリケーションが適切にコーディングされていない場合は、アプリケーションが強制終了することがあります。

プログラムによるログインのアクセス権の付与

アプリケーションに対してプログラムによるログインメカニズムを呼び出すには、`ProgrammaticLoginPermission` 権が必要です。このアクセス権は標準 J2EE メカニズムではないため、デフォルトでは、配備されたアプリケーションには付与されていません。

必要なアクセス権をアプリケーションに付与するには、*instance_dir*`/config/server.policy` ファイルに次の行を追加します。

```
grant codeBase "file:jar_file_path" {
   permission com.sun.appserv.security.ProgrammaticLoginPermission
   "login";
};
```

jar_file_path は、アプリケーションの JAR ファイルへのパスです。

`server.policy` ファイルの詳細は、59 ページの「server.policy ファイル」を参照してください。

ProgrammaticLogin クラス

`com.sun.appserv.security.ProgrammaticLogin` クラスを使用すると、ユーザーがプログラムによるログインを実行できます。このクラスには、`login` メソッドが 2 種類あります。1 つは、サーブレットまたは JSP 用で、もう 1 つは EJB コンポーネント用です。

サーブレットまたは JSP 用のログインメソッドは、次の署名を持ちます。

```
public Boolean login (String user, String password,
javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
```

EJB 用のログインメソッドは、次の署名を持ちます。

```
public Boolean login (String user, String password)
```

どちらの `login` メソッドも、次の処理を行います。

- 認証を実行する

- ログインが成功したら `true` を返し、ログインが失敗したら `false` を返す

第 4 章

J2EE アプリケーションのアセンブルと配備

この章では、Sun ONE Application Server モジュールと、これらのモジュールをアプリケーション内に個別に、または一括してアセンブルする方法について説明します。アセンブリに影響を与える設計上の考慮事項については、26 ページの「アプリケーションのモジュール化」を参照してください。

Sun ONE Application Server のモジュールおよびアプリケーションには、J2EE 標準の要素と Sun ONE Application Server 固有の要素が組み込まれています。この章では、Sun ONE Application Server 固有の要素についてのみ詳細に説明します。

この章には次の節があります。

- アセンブリと配備の概要
- モジュールとアプリケーションのアセンブル
- モジュールおよびアプリケーションの配備
- Apache Ant のアセンブリツールおよび配備ツール
- アプリケーション配備記述子ファイル

アセンブリと配備の概要

アプリケーションアセンブリ (パッケージ化とも呼ばれる) は、アプリケーションの個別のコンポーネントを、J2EE に準拠するアプリケーションサーバーに配備できる単位に結合するプロセスです。パッケージは、モジュールまたは独立したアプリケーションとして利用できます。この節には次のトピックがあります。

- モジュール
- アプリケーション
- J2EE 標準記述子
- Sun ONE Application Server 記述子

- 命名規則
- JNDI ネーミングサービス
- ディレクトリ構造
- 実行時環境
- クラスローダー
- サンプルアプリケーション

モジュール

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 (たとえば Web、EJB など) の 2 つの配備記述子を持ちます。一方の配備記述子は J2EE 標準で、もう一方の記述子は Sun ONE Application Server 固有です。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る。WAR ファイルは、サーブレット、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`) から構成される
- **EJB JAR ファイル** : EJB JAR ファイルは、エンタープライズ Bean をアセンブルするときに使われる標準フォーマットである。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 `sun-ejb-jar.xml`) が格納される。EJB コンポーネントがコンテナ管理による持続性を使用するエンティティ Bean である場合、`.dbschema` ファイルと CMP マッピング記述子ファイル (`sun-cmp-mapping.xml`) ファイルも同様に含まれる場合がある
- **Application Client Container JAR ファイル** ACC クライアントは、Sun ONE Application Server 固有の J2EE クライアントである。ACC 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る。このクライアントの配備記述子は、`application-client.xml` と `sun-application-client.xml` である
- **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る。コネクタモジュールは、デバイスドライバのように機能する。この移植性のある方法を使用すると、EJB コンポーネントは外部の企業システムにアクセスできるようになる。Sun ONE Application Server の各コネクタには、J2EE XML ファイルである `ra.xml` がある。また、Sun ONE Application Server 配備記述子である `sun-ra.xml` も必要となる

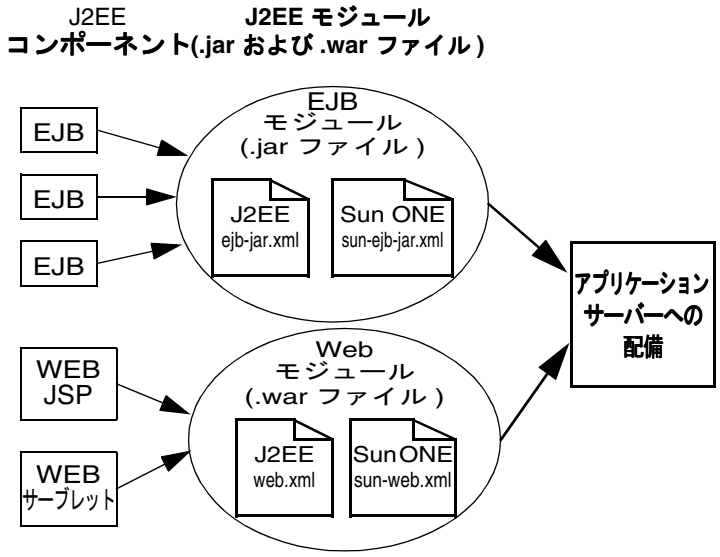
モジュールを配備した後にクラスローダーが正しいクラスを検索できるように、すべてのモジュールのソースコードでパッケージ定義を使う必要があります。

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。

Sun ONE Application Server は、ライフサイクルモジュールもサポートしています。詳細は、第 6 章「ライフサイクルリスナーの開発」を参照してください。

次の図に示すように、EJB JAR モジュールと Web モジュールは 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。

モジュールのアセンブリと配備

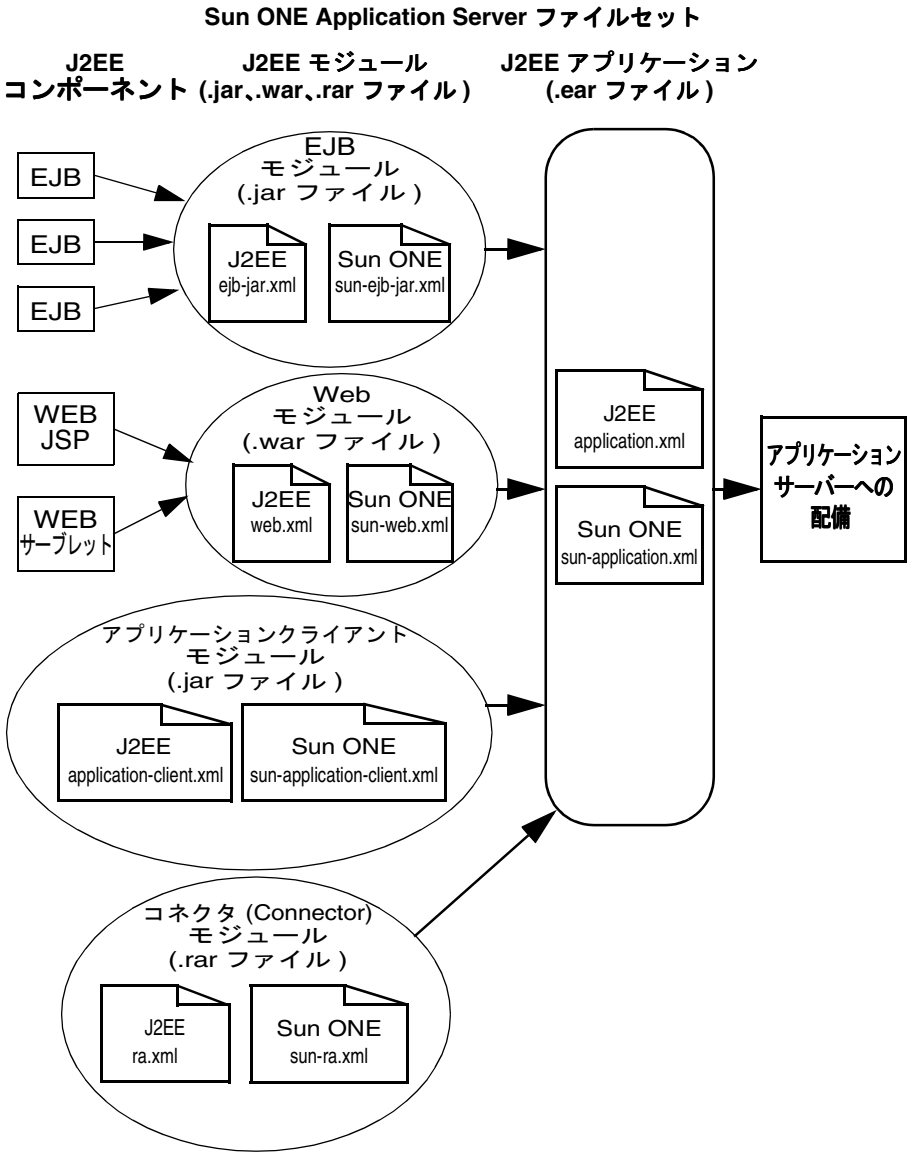


アプリケーション

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。

次の図は、配備する準備として、モジュールにコンポーネントをアセンブルしてから、Sun ONE Application Server アプリケーションの EAR ファイルにアセンブルする方法を示しています。

アプリケーションのアセンブリと配備



各モジュールには、Sun ONE Application Server 配備記述子および J2EE 配備記述子が定義されています。Sun ONE Application Server 管理インタフェースは、配備記述子を使って、アプリケーションコンポーネントを配備し、Sun ONE Application Server にリソースを登録します。

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構成されています。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 WAR、JAR、EAR 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 の標準配備記述子の詳細は、次のサイトでバージョン 1.3 の J2EE 仕様書を参照してください。

`http://java.sun.com/products/`

配備前に配備記述子の正しさを確認する方法については、86 ページの「配備記述子ベリファイア」を参照してください。

次の表は、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」
`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」

Sun ONE Application Server 記述子

Sun ONE Application Server では、Sun ONE Application Server 固有の機能を設定するために追加の配備記述子を使用します。`sun-application.xml` ファイルと `sun-web.xml` ファイルはオプションで、それ以外のすべての配備記述子は必須です。

すべての Sun ONE Application Server 配備記述子用の DTD スキーマファイルは、*install_dir*`/lib/dtds` ディレクトリにあります。

配備前に配備記述子の正しさを確認する方法については、86 ページの「配備記述子ベリファイア」を参照してください。

次の表は、Sun ONE Application Server 記述子についての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`	132 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-ejb-jar.xml` および `sun-cmp-mapping.xml`	『Sun ONE Application Server Enterprise JavaBeans 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 600 のアクセス権限を持っている必要があります。

命名規則

アプリケーション名および個別に配備された EJB、JAR、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。また、CMP を使用するエンティティ Beans では、同じアプリケーション内で `.dbschema` ファイルに一意の名前を付ける必要があります。

パッケージとファイル名に、スペースやお使いの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。

JNDI ネーミングサービス

クライアントまたは Web アプリケーションが EJB コンポーネントと通信する場合、または Web アプリケーションまたは EJB コンポーネントが JDBC や別のリソースからのサービスを必要とする場合は、ネーミングサービスによって相互のコンポーネントが特定され、対話が行われます。ネーミングサービスは、名前とオブジェクトを関連付けるバインドのセットを維持します。J2EE のネーミングサービスは JNDI (Java Naming and Directory Interface) です。

Sun ONE Application Server では、コンテナがコンポーネントに命名環境 (コンテキスト) を提供しています。これにより、コンポーネントは分散しているその他のコンポーネントやリソースを検索できます。`Context` オブジェクトは、名前とオブジェクトのバインド、名前とオブジェクトのバインド解除、オブジェクト名の変更、バインドリストの表示を実行するメソッドを提供します。

JNDI はサブコンテキスト機能にも対応しています。ファイルシステムのディレクトリのように、サブコンテキストはコンテキストに含まれるコンテキストです。この階層構造により、情報を効率的に組織化できます。サブコンテキストをサポートするネーミングサービスでは、`Context` クラスはサブコンテキストの生成と削除を実行するメソッドも提供します。

EJB コンポーネントの JNDI 名は一意である必要があります。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。

注 JNDI で他の企業リソースと名前が重複することを避けるため、また、移植性の問題を回避するために、Sun ONE Application Server アプリケーションのすべての名前は、`java:comp/env` という文字列から始める必要があります。

次の表は、Sun ONE Application Server の接続ファクトリの JNDI サブコンテキストを示しています。左の列にリソースマネージャのタイプ、中央の列に接続ファクトリのタイプ、右の列に JNDI サブコンテキストを示します。

接続ファクトリの JNDI サブコンテキスト

リソースマネージャのタイプ	コネクションファクトリのタイプ	JNDI サブコンテキスト
JDBC	`javax.sql.DataSource`	`java:comp/env/jdbc`
JMS	`javax.jms.TopicConnectionFactory`	`java:comp/env/jms`
	`javax.jms.QueueConnectionFactory`	
JavaMail	`javax.mail.Session`	`java:comp/env/mail`
URL	`java.net.URL`	`java:comp/env/url`
コネクタ	`javax.resource.cci.ConnectionFactory`	`java:comp/env/eis`

ディレクトリ構造

アプリケーションを配備すると、アプリケーションは空のディレクトリ構造に展開され、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けられます。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構成した場合、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
```

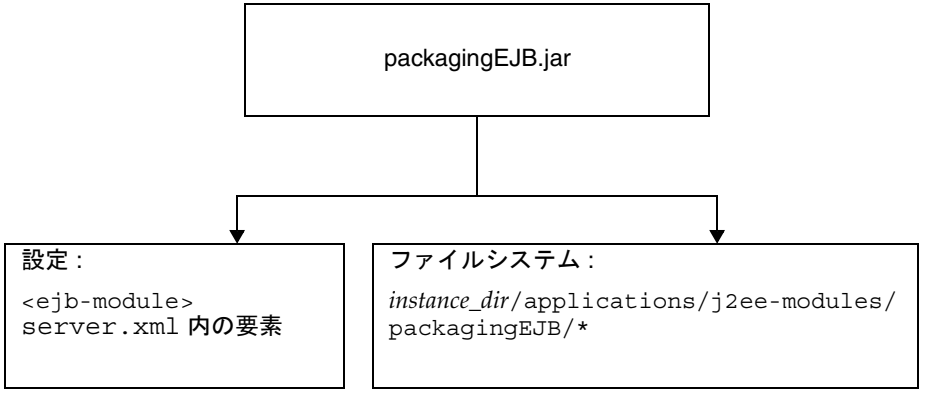
実行時環境

個々のモジュールまたはアプリケーションを配備すると、ファイルシステムとサーバー設定の両方が配備の影響を受けます。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です。

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*

applications ディレクトリには、73 ページの「ディレクトリ構造」で説明したディレクトリ構造が含まれます。generated/ejb ディレクトリには、ACC クライアントがモジュールにアクセスするときに必要となるスタブとタイが保存され、generated/jsp ディレクトリにはコンパイルされた JSP が保存されます。

ライフサイクルモジュール (第 6 章「ライフサイクルリスナーの開発」を参照) 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。

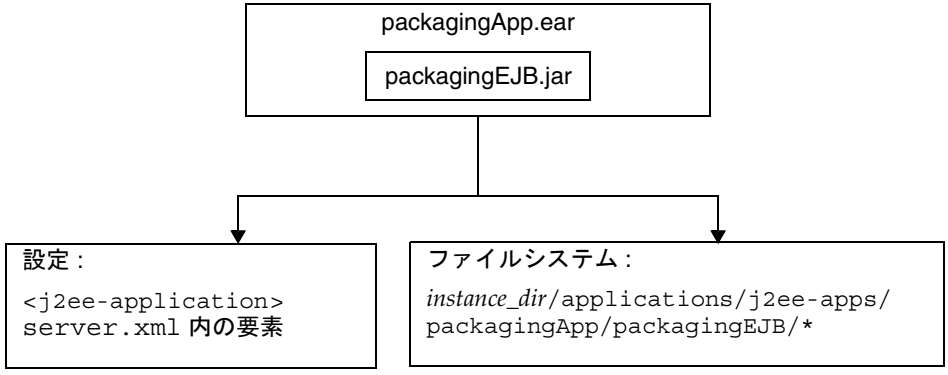
```
<server>
   <applications>
      <type-module>
         ... モジュールの設定 ...
      </type-module>
   </applications>
</server>
```

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*

applications ディレクトリには、73 ページの「ディレクトリ構造」で説明したディレクトリ構造が含まれます。generated/ejb ディレクトリには、ACC クライアントがモジュールにアクセスするときに必要となるスタブとタイが保存され、generated/jsp ディレクトリにはコンパイルされた JSP が保存されます。

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>
         ... アプリケーションの設定 ...
      </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。

クラスローダー

Sun ONE Application Server クラスローダー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。

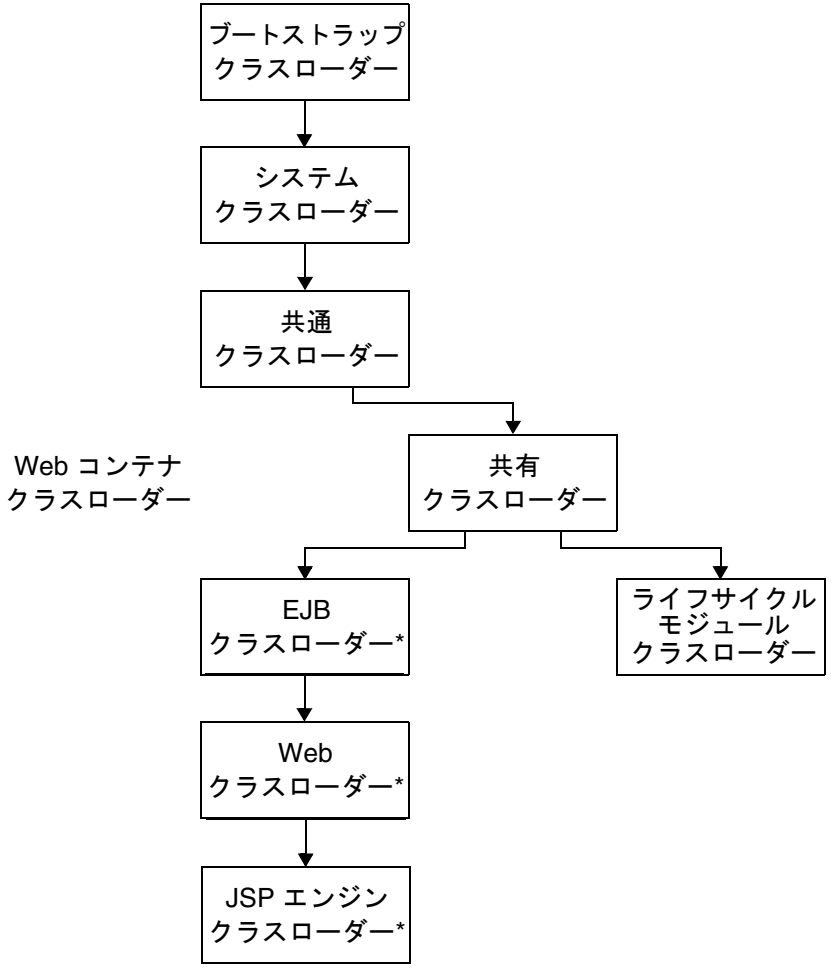
Java 仮想マシン (JVM) のクラスローダーは、依存関係の解決に必要な Java クラスファイルを動的に読み込みます。たとえば、java.util.Enumeration のインスタンスを作成する場合は、クラスローダーの 1 つが関連するクラスを実行時環境に読み込みます。この節には次のトピックがあります。

- クラスローダーの階層
- クラスローダー領域
- クラスローダー分離の回避

クラスローダーの階層

Sun ONE Application Server 実行時環境のクラスローダーは、次に示す階層に構成されています。

クラスローダーの実行時階層



* 各アプリケーション用に別々のクラスローダーインスタンスがあります
(これらのクラスローダーの 1 つは、各アプリケーションのクラスローダー領域にあります)。

これは、Java 継承階層ではなく、委譲階層です。委譲方式のクラスローダーは、自分でクラスを読み込まずに、親へクラスの読み込み作業を委託します。クラスローダーの親は、システムクラスローダーか、または別のカスタムクラスローダーのいずれかです。親クラスローダーがクラスを読み込めない場合、クラスローダーサブクラスで、`findClass()` メソッドが呼び出されます。つまり、クラスローダーは、親が読み込めないクラスだけを読み込みます。

Web クラスローダーは例外で、サーブレット仕様の委譲モデルに従います。Web クラスローダーは、親に委譲する前にローカルクラスローダーを調べます。Web クラスローダーが最初に親に委譲するようにするには、`sun-web.xml` ファイルの `class-loader` 要素を `delegate="true"` に設定します。詳細は、『Web アプリケーション開発者ガイド』を参照してください。

注 Web サービスの Web コンポーネントで使われる Web クラスローダーは、親クラスローダーに委譲する必要があります。この場合は、`sun-web.xml` ファイルの `class-loader` 要素を `delegate="true"` に設定します。

次の表は、Sun ONE Application Server のクラスローダーを示しています。左の列はクラスローダー、右の列はそのクラスローダーの説明と参照するファイルを示しています。

Sun ONE Application Server のクラスローダー

クラスローダー	説明
ブートストラップ	ブートストラップクラスローダーは、すべての JDK クラスを読み込む
システム	システムクラスローダーは、コアの Sun ONE Application Server クラスのほとんどを読み込む。このクラスローダーは、`server.xml` ファイル内の `java-config` 要素の `classpath-prefix`、`server-classpath`、および `classpath-suffix` 属性に基づいて作成される。`env-classpath-ignored="false"` が `java-config` 要素内に設定される場合は、環境クラスパスが含まれる
共通	共通クラスローダーは、*instance_dir*`/lib/classes` ディレクトリ内にクラスを読み込み、続いて *instance_dir*`/lib` ディレクトリ内に JAR および ZIP ファイルを読み込む。特別なクラスパス設定は必要ない。これらのディレクトリは必ずしも存在しているとは限らない。存在しない場合、共通クラスローダーは作成されない
共有	共有クラスローダーは、すべてのアプリケーションで共有されるクラス (個別に配備したコネクタモジュールなど) を読み込む単一のインスタンスである
ライフサイクルモジュール	LifeCycleModule クラスローダーは、ライフサイクルモジュールの親クラスローダーである。各ライフサイクルモジュールのクラスパスは、それぞれの専用のクラスローダーを構築するときに使用される

Sun ONE Application Server のクラスローダー (続き)

クラスローダー	説明
EJB	EJB クラスローダーは、特定の有効な EJB モジュールまたは J2EE アプリケーション内にある有効な EJB クラスを読み込む。このクラスローダーのインスタンスの 1 つは、各クラスローダーの領域にある。EJB クラスローダーは、読み込む必要のあるクラスの場所を示す URL のリストを使って作成される
Web	Web クラスローダーは、特定の有効な Web モジュールまたは J2EE アプリケーション内にあるサーブレットおよびその他のクラスを読み込む。このクラスローダーのインスタンスの 1 つは、各クラスローダーの領域にある。Web クラスローダーは、読み込む必要のあるクラスの場所を示す URL のリストを使って作成される
JSP エンジン	JSP エンジンクラスローダーは、有効な JSP のコンパイルされた JSP クラスを読み込む。このクラスローダーのインスタンスの 1 つは、各クラスローダーの領域にある。JSP エンジンクラスローダーは、読み込む必要のあるクラスの場所を示す URL のリストを使って作成される

クラスローダー領域

サーバー上にインストールしたアプリケーションやモジュール内のコンポーネントへのアクセスは、分離されたクラスローダー領域のコンテキスト内で発生し、各領域は、専用の EJB、Web、および JSP エンジンクラスローダーを持ちます。

- **アプリケーション領域** : 各 J2EE アプリケーションは、アプリケーションのすべてのモジュール内にあるクラスを読み込む、専用のクラスローダー領域を持っている
- **個別に配備されたモジュール領域** : 個別に配備された各 EJB JAR、web WAR、またはライフサイクルモジュールは、モジュール内のクラスを読み込む、専用のクラスローダー領域を持っている

注 iPlanet Application Server 6.*x* では、個別に配備されたモジュールは同じクラスローダーを共有していました。Sun ONE Application Server 7 では、個別に配備されたモジュールは、それぞれ専用のクラスローダー領域を持っています。

注 サーブレット、JSP、または EJB コンポーネントがアクセスするファイルなどのリソースは、クラスローダーのクラスパスが指定するディレクトリにある必要があります。たとえば、Web クラスローダーのクラスパスには次のようなディレクトリがあります。

module_name/WEB-INF/classes
module_name/WEB-INF/lib

サーブレットがリソースにアクセスする場合、これらのディレクトリにリソースがないと読み込まれません。

クラスローダー分離の回避

各アプリケーションまたは個別に配備されたモジュールのクラスローダー領域は分離されているため、アプリケーション (またはモジュール) は、別のアプリケーション (またはモジュール) からクラスを読み込むことができません。このため、異なるアプリケーションに同じ名前の 2 つのクラスがあったとしても、互いに問題になることはありません。

複数のアプリケーションによってアクセスされるライブラリ、ユーティリティクラス、または個別に配備されたモジュールに対して、この制限が適用されないようにするには、次のいずれかの方法で、必要なクラスへの相対パスを追加します。

- システムクラスローダーの使用
- 共通クラスローダーの使用
- アプリケーション内の別アプリケーション用クライアント JAR のパッケージ化

システムクラスローダーの使用

システムクラスローダーを使用するには、次のいずれかを実行した後、サーバーを再起動します。

- 管理インタフェースからサーバーインスタンスページを表示し、「JVM 設定」タブを選択し、「パス設定」オプションを選択してから、「クラスパスのサフィックス」フィールドを編集し、「保存」を選択する
- `server.xml` ファイル内の `java-config` 要素の `classpath-sufix` 属性を編集する。`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

システムクラスローダーを使用すると、アプリケーションまたはモジュールが、サーバーインスタンス内のほかのアプリケーションまたはモジュールにアクセスできるようになります。

共通クラスローダーの使用

共通クラスローダーを使用するには、JAR および ZIP ファイルを *instance_dir*/lib ディレクトリにコピーするか、または .class ファイルを *instance_dir*/lib/classes ディレクトリにコピーした後、サーバーを再起動します。

共通クラスローダーを使用すると、アプリケーションまたはモジュールが、サーバーインスタンス内のほかのアプリケーションまたはモジュールにアクセスできるようになります。

アプリケーション内の別アプリケーション用クライアント JAR のパッケージ化

アプリケーション A に含まれるアプリケーション B 用にクライアント JAR をパッケージ化することで、アプリケーション B の EJB コンポーネントまたは Web コンポーネントが、アプリケーション A (従属アプリケーション) の EJB コンポーネントを呼び出せるようになります。このとき、どちらのアプリケーションのコンポーネントも、別のアプリケーションやモジュールにアクセスできるようにする必要はありません。

アプリケーションに含まれる別アプリケーション用にクライアント JAR をパッケージ化することには、短所もあります。-nolocalstubs オプションを有効化すると、他のアプリケーションのクライアント JAR を含む複数のアプリケーションを、サーバーを再起動せずに配備できるようになります。ただし、-nolocalstubs オプションの使用はサーバーのパフォーマンスを低下させることがあります。

その代わりに、81 ページの「共通クラスローダーの使用」で説明する方法で共通クラスローダーに従属アプリケーションのクライアント JAR をロードさせることができます。サーバーのパフォーマンスは良くなりますが、従属アプリケーションにアクセスするにはサーバーの再起動が必要です。またサーバーインスタンス内でのアクセスが可能になります。本稼動環境ではこの方法をお勧めします。

アプリケーション内の別アプリケーション用にクライアント JAR をパッケージ化するには、次の手順に従います。

1. 次のいずれかの方法でサーバーインスタンスの rmic オプションに -nolocalstubs オプションを追加し、サーバーを再起動します。
   - 管理インタフェースでサーバーインスタンスページを表示し、「JVM 設定」タブを選択し、「一般」オプションを選択してから、「rmic オプション」フィールドに -nolocalstubs を追加し、「保存」を選択する
   - server.xml ファイル内の java-config 要素の rmic-options 属性に -nolocalstubs を追加する。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

   この変更を加えると、それ以後に配備するすべての EJB コンポーネントには、リモートアクセスだけが可能となります。

注 -nolocalstubs オプションの使用はサーバーのパフォーマンスを低下させることがあります。

2. 従属アプリケーションを配備します。
3. 従属アプリケーションのクライアント JAR ファイルを呼び出し側アプリケーションに追加します。
   - EJB コンポーネントの呼び出しでは、EJB コンポーネントと同じレベルにクライアント JAR ファイルを追加する。次に、呼び出し側 EJB コンポーネントの MANIFEST.MF ファイルに Class-Path エントリを追加する。Class-Path エントリの構文は、次のとおり

     Class-Path: *filepath1*.jar *filepath2*.jar ...

     それぞれの *filepath* は、MANIFEST.MF ファイルを含むディレクトリまたは JAR ファイルに対して相対的なものです。詳細は、J2EE 仕様書の第 8.1.1.2 項「Dependencies」を参照
   - 呼び出し側の Web コンポーネントでは、WEB-INF/lib ディレクトリの下にクライアント JAR ファイルを追加する
4. ほとんどのアプリケーションでは、クライアント JAR ファイルと呼び出し側 EJB コンポーネントをパッケージ化するだけで十分です。Web コンポーネントが従属アプリケーションの EJB コンポーネントを直接呼び出す場合を除き、EJB コンポーネントと Web コンポーネントの両方をクライアント JAR ファイルとパッケージ化する必要はありません。EJB コンポーネントと Web コンポーネントの両方をクライアント JAR とパッケージ化するときは、sun-web.xml ファイルの class-loader 要素に delegate="true" を設定します。この設定により、クラスローダーの標準の委譲モデルが適用され、Web クラスローダーはクラス自体をロードする前に親に委譲します。
5. 呼び出し側アプリケーションを配備します。

注 呼び出し側の EJB コンポーネントまたは Web コンポーネントは、従属アプリケーション側の EJB コンポーネントと同じ JNDI 名を使う必要があります。ejb-ref マッピングは機能しません。

サンプルアプリケーション

Sun ONE Application Server には、参照したり配備したりできるサンプルアプリケーションがあり、*install_dir*/samples ディレクトリに含まれています。サンプルアプリケーションは、ejb、jdbc、connectors、i18n などのカテゴリに分かれています。サンプルアプリケーションの各カテゴリは、さらにサブカテゴリに分かれています。たとえば ejb カテゴリには、stateless、stateful、security、mdb、bmp、cmp のサブカテゴリがあります。

Sun ONE Application Server のほとんどのサンプルアプリケーションは、次のディレクトリ構造を持ちます。

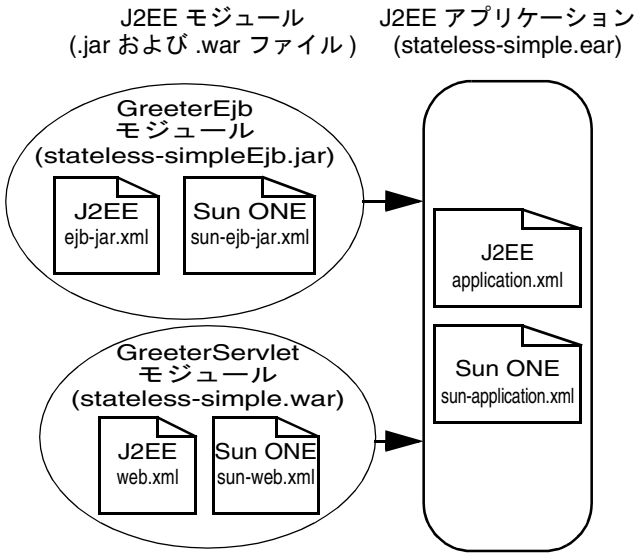
- docs ディレクトリには、サンプルアプリケーションの使用方法に関するドキュメントが保存されている
- src ディレクトリには、次の内容が保存されている
  - ソースコード
  - サンプルの asant ターゲットを定義した build.xml ファイル (106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照)
  - 配備記述子
- build ディレクトリ、assemble ディレクトリ、および javadocs ディレクトリは、build.xml ファイルに指定されているターゲットの結果として生成される

install_dir/samples/common.xml ファイルには、すべてのサンプルアプリケーションに共通するプロパティが定義されており、サンプルアプリケーションのコンパイル、アセンブル、配備、再配備に必要なターゲットを実装します。ほとんどのサンプルアプリケーションでは、build.xml ファイルが common.xml ファイルを含みます。

注 *install_dir*/samples/webservices の下にあるサンプルアプリケーションを使用する前に、jre/lib/endorsed ディレクトリに Java XML Pack JAR ファイルをコピーし、JDK に付属する JAR ファイルを上書きします。

次の図は、helloworld というサンプルアプリケーションの構造を示しています。

サンプルアプリケーション helloworld



Sun ONE Application Server に配備したサンプルアプリケーションは、次の URL を使って起動できます。

`http://`*server*`:`*port*`/helloworld`

サンプルアプリケーションの詳細、および配備と実行の方法については、次のディレクトリにある関連ドキュメントを参照してください。

install_dir`/samples/ejb/stateless/simple/docs/index.html`

モジュールとアプリケーションのアセンブル

Sun ONE Application Server 内でモジュールおよびアプリケーションをアセンブリ (パッケージ化) するときは、従来のすべての J2EE 仕様書に準拠する必要があります。ただし、Sun ONE Application Server 内でアセンブルするときは、Sun ONE Application Server 固有の配備記述子 (`sun-web.xml`、`sun-ejb-jar.xml` など) を含めて、アプリケーションサーバーの機能を拡張する必要があります。

この節には次のトピックがあります。

- アセンブリ用ツール
- WAR モジュールのアセンブル
- EJB JAR モジュールのアセンブル
- ライフサイクルモジュールのアセンブル
- アプリケーションのアセンブル
- ACC クライアントのアセンブル
- J2EE CA リソースアダプタのアセンブル

アセンブリ用ツール

Sun ONE Application Server では、モジュールまたはアプリケーションのアセンブルを次の方法で行うことができます。

- Apache Ant
- Sun ONE Studio
- 配備記述子ベリファイア

Apache Ant

Ant は、モジュールとアプリケーションのアセンブルおよび配備に役立ちます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」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アセンブルには 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。

配備記述子ベリファイア

ベリファイアツールは、J2EE 配備記述子と Sun ONE Application Server 固有の配備記述子の両方を、対応する DTD ファイルと比較し、モジュールまたはアプリケーションが J2EE および Sun ONE Application Server に準拠していない場合にエラーや警告を返します。EAR、WAR、RAR、JAR ファイルの配備記述子を検証できます。

ベリファイアツールは、単なる XML 構文ベリファイアではありません。配備記述子に含まれるさまざまな要素間で、ルールと相互依存関係が検証されます。必要に応じて、ユーザーアプリケーションクラスを調べて検証ルールを適用します。

ベリファイアは、`sun-appserv-deploy` Ant タスクにも統合されています。

ヒント ベリファイアツールを使うことで、デバッグが難しい実行時エラーを回避できます。

この節には次のトピックがあります。

- コマンド行ユーティリティ
- Ant の統合
- 結果ファイルの例

コマンド行ユーティリティ

ベリファイアツールの構文は次のとおりです。

`verifier [`*options*`] `*file*

file には、EAR、WAR、RAR、または JAR ファイルを指定できます。

次の表は、`verifier` コマンドの *options* (省略可能) を示しています。左の列はオプション、右の列は各オプションの説明です。

verifier コマンドのオプション

オプション	説明
`-v`	詳細デバッグモードを有効化する
`-d` *output_dir*	テスト結果を既存のディレクトリ *output_dir* に書き出す。デフォルトでは、結果ファイルはシステム定義のディレクトリ `tmp` に書き出される
`-ra`	すべての結果が表示されるように、レポート出力のレベルを設定する。これは、詳細モード、非詳細モードの両方のデフォルトレベルである
`-rw`	警告と失敗の結果だけが表示されるように、レポート出力のレベルを設定する
`-rf`	失敗の結果だけが表示されるように、レポート出力のレベルを設定する

たとえば、次のコマンドを実行するとベリファイアが詳細モードで実行され、ejb.jar ファイルの静的検証のすべての結果が ResultsDir ディレクトリに書き出されます。

```
verifier -v -ra -d ResultsDir ejb.jar
```

結果ファイルは、ejb.jar_verifier.txt と ejb.jar_verifier.xml になります。

ベリファイアが問題なく実行されると、結果コード 0 が返されます。これは、検証エラーがなかったことを意味するわけではありません。ベリファイアの実行に失敗した場合は、ゼロ以外の値が返されます。

Ant の統合

Ant の build ファイルにターゲットとしてベリファイアを統合すると、アプリケーションまたはモジュールをアセンブルするたびに、Ant の呼び出し機能を使ってターゲットを呼び出せます。これは、com.sun.enterprise.tools.verifier.Verifier の main メソッドをユーザー Ant スクリプトから呼び出せるためです。main メソッドは、「verifier コマンドのオプション」の表で説明した引数を受け入れます。

次に、Ant の verify ターゲットのコード例を示します。

```
<target name="verify">
   <echo message="Verification Process for ${testfile}"/>
   <java classname="com.sun.enterprise.tools.verifier.Verifier"
          fork="yes">
       <sysproperty key="com.sunone.enterprise.home"
          value="${appserv.home}"/>
       <sysproperty key="verifier.xsl"
          value="${appserv.home}/verifier/config" />
       <!-- uncomment the following for verbose output -->
       <!--<arg value="-v"/>-->
       <arg value="${assemble}/${ejbjar}" />
       <classpath path="${appserv.cpath}:${java.class.path}"/>
   </java>
</target>
```

結果ファイルの例

次に、結果ファイル (XML ファイル) の例を示します。

```
<static-verification>
   <ejb>
       <failed>
          <test>
             <test-name>
tests.ejb.session.TransactionTypeNullForContainerTX
             </test-name>
```

```
            <test-assertion>
Session bean with bean managed transaction demarcation test
            </test-assertion>
            <test-description>
For [ TheGreeter ] Error: Session Beans [ TheGreeter ] with [ Bean ]
managed transaction demarcation should not have container
transactions defined.
            </test-description>
         </test>
      </failed>
   </ejb>
...
</static-verification>
```

次に、結果ファイル (TXT ファイル) の例を示します。

```
---------------------------
STATIC VERIFICATION RESULTS
---------------------------
---------------------------------
NUMBER OF FAILURES/WARNINGS/ERRORS
---------------------------------

# of Failures : 3
# of Warnings : 6
# of Errors : 0

-----------------------------
RESULTS FOR EJB-RELATED TESTS
-----------------------------
--------------
FAILED TESTS :
--------------

Test Name : tests.ejb.session.TransactionTypeNullForContainerTX
Test Assertion : Session bean with bean managed transaction
demarcation test
Test Description : For [ TheGreeter ]
Error: Session Beans [ TheGreeter ] with [ Bean ] managed
transaction demarcation should not have container transactions
defined.

...

---------------
PASSED TESTS :
---------------
```

```
Test Name : tests.ejb.session.ejbcreatemethod.EjbCreateMethodStatic
Test Assertion : Each session Bean must have at least one non-static
ejbCreate method test
Test Description : For [ TheGreeter ] For EJB Class [
samples.helloworld.ejb.GreeterEJB ] method [ ejbCreate ] [
samples.helloworld.ejb.GreeterEJB ] properly declares non-static
ejbCreate(...) method.

...

-----------
WARNINGS :
-----------

Test Name : tests.ejb.businessmethod.BusinessMethodException
Test Assertion : Enterprise bean business method throws
RemoteException test
Test Description :

Test Name : tests.ejb.ias.beanpool.IASEjbBeanPool
Test Assertion :
Test Description : WARNING [IAS-EJB ejb] : bean-pool should be
defined for Stateless Session and Message Driven Beans

...

----------------------
NOTAPPLICABLE TESTS :
----------------------

Test Name :
tests.ejb.entity.pkmultiplefield.PrimaryKeyClassFieldsCmp


Test Assertion : Ejb primary key class properly declares all class
fields within subset of the names of the container-managed fields
test.
Test Description : For [ TheGreeter ] class
com.sun.enterprise.tools.verifier.tests.ejb.entity.pkmultiplefield.
PrimaryKeyClassFieldsCmp expected Entity bean, but called with
Session.

Test Name : tests.ejb.entity.ejbcreatemethod.EjbCreateMethodReturn
Test Assertion : Each entity Bean may have zero or more ejbCreate
methods which return primary key type test
Test Description : For [ TheGreeter ] class
com.sun.enterprise.tools.verifier.tests.ejb.entity.ejbcreatemethod.
EjbCreateMethodReturn expected Entity bean, but called with Session
```

```
bean.

...

-----------------------------------
RESULTS FOR OTHER XML-RELATED TESTS
-----------------------------------
---------------
PASSED TESTS :
---------------

Test Name : tests.dd.ParseDD
Test Assertion : Test parses the deployment descriptor using a SAX
parser to avoid the dependency on the DOL
Test Description : PASSED [EJB] : [ remote ] and [ home ] tags
present.
PASSED [EJB]: session-type is Stateless.
PASSED [EJB]: trans-attribute is NotSupported.
PASSED [EJB]: transaction-type is Bean.

...
```

WAR モジュールのアセンブル

WAR モジュールをアセンブルするには、次の手順で行います。

1. 作業ディレクトリを作成し、Web モジュールのコンテンツをその中にコピーします。ディレクトリの構造については、『Sun ONE Application Server Web アプリケーション開発者ガイド』の説明を確認してください。
2. `web.xml` (必須) および `sun-web.xml` (省略可能) という名前の 2 つの配備記述子ファイルを作成します。これらのファイルの詳細は、『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。

ヒント 最初は、WAR モジュールのアセンブルと配備記述子の作成に Sun ONE Studio を使えます。生成された WAR ファイルから配備記述子を抽出することもできます。

---

3. 次のコマンドを実行して、WAR ファイルを作成します。

   ```
   jar -cvf module_name.war *
   ```

ヒント アセンブルプロセスは、Ant ツールを使って自動化できます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照してください。

EJB JAR モジュールのアセンブル

EJB JAR モジュールをアセンブルするには、次の手順で行います。

1. 作業ディレクトリを作成し、モジュールの内容をコピーします。ディレクトリの構造については、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』の説明を確認してください。
2. `ejb-jar.xml` および `sun-ejb-jar.xml` という名前の 2 つの配備記述子ファイル (どちらも必須) を作成します。これらのファイルの詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。EJB コンポーネントがコンテナ管理による持続性を使用するエンティティ Bean である場合、`.dbschema` ファイルおよび `sun-cmp-mapping.xml` ファイルを作成する必要があります。

ヒント 最初は、EJB WAR モジュールのアセンブルと配備記述子の作成に Sun ONE Studio を使えます。生成された EJB JAR ファイルから配備記述子を抽出することもできます。

3. 次のコマンドを実行して、JAR ファイルを作成します。

   ```
   jar -cvf module_name.jar *
   ```

ヒント アセンブルプロセスは、Ant ツールを使って自動化できます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照してください。

注 J2EE 仕様の第 8.1.1.2 項「Dependencies」には、個別に配備した EJB モジュールのユーティリティクラスをパッケージ化できないことが規定されています。その代わりに、JAR Extension Mechanism Architecture を使ってアプリケーション内の EJB モジュールとユーティリティ JAR をパッケージ化します。それ以外の方法については、80 ページの「クラスローダー分離の回避」を参照してください。

ライフサイクルモジュールのアセンブル

ライフサイクルモジュールをアセンブルするには、次の手順で行います。

1. 作業ディレクトリを作成し、モジュールの内容をコピーします。
2. 次のコマンドを実行して、JAR ファイルを作成します。

   ```
   jar -cvf module_name.jar *
   ```

ヒント アセンブルプロセスは、Ant ツールを使って自動化できます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照してください。

ライフサイクルモジュールの全般的な情報については、第 6 章「ライフサイクルリスナーの開発」を参照してください。

アプリケーションのアセンブル

アプリケーションをアセンブルするには、次の手順で行います。

1. 作業ディレクトリを作成し、すべてのモジュールを含むアプリケーションのコンテンツをその中にコピーします。ディレクトリの構造については、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』の説明を確認してください。
2. `application.xml` (必須) および `sun-application.xml` (省略可能) という名前の 2 つの配備記述子ファイルを作成します。これらのファイルの詳細は、138 ページの「サンプルアプリケーション XML ファイル」を参照してください。

ヒント 最初は、アプリケーションのアセンブルと配備記述子の作成に Sun ONE Studio を使えます。生成された EAR ファイルから配備記述子を抽出することもできます。

3. 次のコマンドを実行して、J2EE アプリケーションの EAR ファイルを作成します。

   ```
   jar -cvf app_name.ear *
   ```

ヒント アセンブルプロセスは、Ant ツールを使って自動化できます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照してください。

ACC クライアントのアセンブル

この節では、ACC クライアントのアセンブルについて簡単に説明します。ただし、『Sun ONE Application Server Developer's Guide to Clients』の内容を理解していることを前提としています。

ACC クライアントの JAR モジュールをアセンブルするには、次の手順で行います。

1. 作業ディレクトリを作成し、モジュールの内容をコピーします。ディレクトリの構造については、『Sun ONE Application Server Developer's Guide to Clients』の説明を確認してください。
2. `application-client.xml` および `sun-application-client.xml` という名前の 2 つの配備記述子ファイル (どちらも必須) を作成します。これらのファイルの詳細は、『Sun ONE Application Server Developer's Guide to Clients』を参照してください。

ヒント 最初は、クライアント JAR モジュールのアセンブルと配備記述子の作成に Sun ONE Studio を使えます。生成されたクライアント JAR ファイルから配備記述子を抽出することもできます。

3. 次のコマンドを実行して、クライアント JAR ファイルを作成します。

```
jar -cvfm module_name.jar META-INF/MANIFEST.MF *
```

ヒント アセンブルプロセスは、Ant ツールを使って自動化できます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」を参照してください。

ACC クライアントの配備とクライアントマシンの準備については、104 ページの「ACC クライアントの配備」で簡単に説明しています。

J2EE CA リソースアダプタのアセンブル

この節では、J2EE CA リソースアダプタのアセンブルについて簡単に説明します。ただし、『Sun ONE J2EE CA Service Provider Implementation 管理者ガイド』の内容を理解していることを前提としています。

次の XML コネクタファイルは、コネクタをアプリケーションサーバーに配備するのに必要です。

- `ra.xml`
- `sun-ra.xml` (セキュリティマップを含む)

`ra.xml` ファイルは J2EE CA 仕様に基づいており、コネクタにパッケージ化されています。`sun-ra.xml` ファイルは Sun ONE Application Server 固有の情報を含んでいます。

RAR モジュールをアセンブルするには、次の手順で行います。

1. 作業ディレクトリを作成し、モジュールの内容をコピーします。ディレクトリの構造については、『Sun ONE J2EE CA Service Provider Implementation 管理者ガイド』の説明を確認してください。
2. `ra.xml` および `sun-ra.xml` という名前の 2 つの配備記述子を作成します。これらのファイルに関する詳細は、『Sun ONE J2EE CA Service Provider Implementation 管理者ガイド』を参照してください。

ヒント 最初は、RAR モジュールのアセンブルと配備記述子の作成に Sun ONE Studio を使えます。生成された RAR ファイルから配備記述子を抽出することもできます。

3. 次のコマンドを実行して、RAR ファイルを作成します。

   ```
   jar -cvf module_name.rar *
   ```

ヒント アセンブルプロセスは、Ant ツールを使って自動化できます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」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、次のトピックがあります。

- 配備名とエラー
- 配備のライフサイクル
- 配備ツール
- モジュールまたはアプリケーションベースでの配備
- 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
- ACC クライアントの配備
- J2EE CA リソースアダプタの配備
- 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、Sun ONE Application Server の配備記述子ファイルに一意の名前が生成されます。アプリケーションを再配備すると、この名前は変更されます。この名前を手作業で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、名前の重複したアプリケーションまたはモジュールのロードは行いません。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、71 ページの「命名規則」を参照してください。

配備時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。これは、部分的な配備によってサーバーの状態に整合性がなくなることを避けるためです。

配備のライフサイクル

アプリケーションははじめに配備されて、修正および再読み込み、再配備、無効化、再有効化され、最後に配備解除されてサーバーから削除されます。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。

- 動的な配備
- 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

- 動的な再読み込み

動的な配備

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、および配備解除することができます。これを動的な配備と呼びます。

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。

注　`asadmin deploy` に `--force` オプションを指定するか、配備時に管理インタフェースの適切なチェックボックスにチェックマークをつけることで、以前に配備したアプリケーションを上書きすることができます。ただし、設定済みのリソースを更新するときは、事前にそのリソースを削除しておく必要があります。

アプリケーションを再配備すると、自動的に生成された名前が変更されます。

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。無効化したアプリケーションからクライアントにアクセスすることはできません。

アプリケーションまたはモジュールを無効化するには、次のいずれかの手順を実行します。

- `server.xml` ファイルで、アプリケーションまたはモジュールを `enabled="false"` に設定する。`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
- 管理インタフェースを使って次の手順を実行します。
  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. アプリケーションまたはモジュールのタイプのページに移動します。たとえば、Web アプリケーションであれば「Web アプリケーション」ページに移動します。
  c. 無効にするアプリケーションまたはモジュールのチェックボックスをクリックします。
  d. 「無効」ボタンを選択します。このページに表示されるアプリケーションまたはモジュールの状態が無効に変更されます。

動的な再読み込み

動的再読み込みを有効にすると、コードまたは配備記述子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。変更した JSP ファイルまたはクラスファイルをアプリケーションまたはモジュールの配備ディレクトリにコピーするだけで再配備が完了します。サーバーは定期的に変更を確認し、変更に合わせてアプリケーションを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本稼動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。

動的な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して、次の手順を実行する
  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く
  b. 「アプリケーション」ページに移動する
  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する
  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のコード変更を確認して動的に再読み込みする間隔を設定する
  e. 「保存」ボタンをクリックする
  f. サーバーインスタンスのページを表示し、「変更の適用」ボタンを選択する

  詳細は、『Sun ONE Application Server 管理者ガイド』を参照
- `server.xml` ファイルの `applications` 要素の次の属性を編集する
  - `dynamic-reload-enabled="true"` に設定して、動的な再読み込みを有効にする
  - `dynamic-reload-poll-interval-in-seconds` で、アプリケーションとモジュールのコード変更を確認して動的に再読み込みする間隔を設定する

  `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

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 プロパティに設定した頻度で、変更が自動的に再読み込みされます。JSP の動的な再読み込みを無効にするには、reload-interval プロパティを -1 に設定します。

配備ツール

この節では、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。

- Apache Ant
- Sun ONE Studio
- asadmin コマンド
- 管理インタフェース

Apache Ant

Ant は、モジュールとアプリケーションのアセンブルおよび配備に役立ちます。詳細は、106 ページの「Apache Ant のアセンブリツールおよび配備ツール」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。

asadmin コマンド

asadmin コマンドを使用すると、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解除できます。複数マシンまたは複数インスタンスへの同時配備はサポートされていません。この節では、asadmin コマンドについて簡単に解説します。詳細は、『Sun ONE Application Server 管理者ガイド』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場合は、103 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、connector、または client 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]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] [--virtualservers virtual_servers] [--type
application|ejb|web|connector] [--contextroot contextroot]
[--force=true] [--precompilejsp=false] [--verify=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
```

upload を false に設定するときは、*filepath* にサーバーマシンの絶対パスを指定する必要があります。

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に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、73 ページの「ディレクトリ構造」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]
[--passwordfile password_file] [--host localhost] [--port 4848]
[--secure | -s] [--virtualservers virtual_servers] [--type
application|ejb|web|connector] [--contextroot contextroot]
[--force=true] [--precompilejsp=false] [--verify=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
```

upload を false に設定するときは、*filepath* にサーバーマシンの絶対パスを指定する必要があります。

注 Windows 環境でマッピングされたドライブにディレクトリを配備するときは、マッピングされたドライブが割り当てられているユーザーとして Sun ONE Application Server を実行する必要があります。それ以外のユーザーとして実行していると、Sun ONE Application Server はそのディレクトリを認識できません。

---

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解除できます。アプリケーションを配備解除するには、コマンドに --type app を指定します。個別のモジュールを配備解除するには、--type ejb、web、connector、または client 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]
[--passwordfile password_file] [--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 モジュールを配備解除します。

```
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。このページでは、アプリケーションまたはモジュールの再配備と無効化も実行できます。
4. モジュールまたはアプリケーションのディレクトリまたはアーカイブファイルの完全パスを入力し (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。
6. Web モジュールでは、コンテキストルートを入力します。
7. 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 Web モジュールを割り当てます。
8. 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 (これを強制配備と呼びます)。この操作はオプションです。
9. ベリファイアを実行して配備記述子ファイルを検証します。これの操作は省略可能です。ベリファイアの詳細は、86 ページの「配備記述子ベリファイア」を参照してください。
10. モジュールのタイプによっては、その他のフィールドが表示されます。適切なボックスをチェックし、適切な値を入力してください。必須フィールドはアスタリスク (*) で示されます。
11. 「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、103 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。

モジュールまたはアプリケーションベースでの配備

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、74 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

- ACC クライアント (モジュールベースで配備すると、ACC クライアント、サーブレット、または EJB コンポーネントから 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

98 ページの「配備ツール」で説明されているとおりに、WAR モジュールを配備します。

配備時に管理インタフェースの適切なチェックボックスにチェックマークをつけるか、`asadmin deploy` コマンドまたは `asadmin deploydir` コマンドに `--precompilejsp` オプションを指定して、JSP を事前コンパイルすることができます。Ant タスク `sun-appserv-deploy` および `sun-appserv-jspc` を使って JSP を事前コンパイルすることもできます。

JSP 用の生成されたソースは、`-keepgenerated` フラグを `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。

JSP の事前コンパイルと `-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

98 ページの「配備ツール」で説明されているとおりに、EJB JAR モジュールを配備します。

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の全般的な情報については、第 6 章「ライフサイクルリスナーの開発」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 コマンド
- 管理インタフェース

asadmin コマンド

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] [--passwordfile password_file] [--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を配備解除するには、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] [--passwordfile password_file] [--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] [--passwordfile password_file] [--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」(省略可能) - 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」(省略可能) - 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。`1` 〜 `100` までの値は予約されている
   - 「致命的な障害」(省略可能) - 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」(省略可能) - 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。

ACC クライアントの配備

クライアントが EJB コンポーネントとデータをやり取りする場合にだけ配備が必要です。ACC クライアントを配備するには、次の手順に従います。

1. 必要なクライアントファイルをアセンブルします (93 ページの「ACC クライアントのアセンブル」を参照)。
2. クライアントがアクセスする EJB コンポーネントをアセンブルします。

3. クライアントと EJB コンポーネントをアプリケーションにパッケージ化します。
4. アプリケーションを配備します。
5. 配備が完了すると、クライアント JAR ファイルは次の場所に作成されます。

   instance_dir/applications/j2ee-apps/*app_name*/*app_name*Client.jar

   クライアント JAR には、ACC 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。

Sun ONE Application Server マシンでクライアントを実行テストするときは、*install_dir*/bin ディレクトリにある appclient スクリプトを使用できます。デフォルトのサーバーインスタンスを使っている場合に必要なオプションは、-client だけです。次に例を示します。

```
appclient -client converterClient.jar
```

デフォルトインスタンスを使っていない場合は、sun-acc.xml ファイルの場所を指定する -xml パラメータも指定する必要があります。

別のマシンで ACC クライアントを実行するには、事前にクライアントマシンを準備する必要があります。

1. *install_dir*/bin ディレクトリにある package-appclient スクリプトを使って ACC パッケージ JAR ファイルを作成します。JAR ファイルは *install_dir*/lib/appclient ディレクトリに作成されます。
2. ACC パッケージ JAR ファイルをクライアントマシンにコピーし、unjar を実行してファイルを展開します。これにより、appclient ディレクトリの下にディレクトリ構造が作成されます。
3. appclient/appserv/lib/appclient ディレクトリにある sun-acc.xml ファイルを設定します。
4. appclient/appserv/bin ディレクトリにある asenv.conf ファイル (Windows 環境では asenv.bat ファイル) を設定します。
5. クライアント JAR ファイルをクライアントマシンにコピーします。

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

98 ページの「配備ツール」で説明されているとおりに、コネクタモジュールを配備します。

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 (ユーティリティクラス、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ーまたは共有クラスローダー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ーを使用すると、リソースの浪費になります。

詳細は、80 ページの「クラスローダー分離の回避」を参照してください。

Apache Ant のアセンブリツールおよび配備ツール

Ant の自動アセンブリ機能を使用することができます。Ant は、次の Apache Software Foundation のサイトから入手できる Java ベースのビルドツールです。

`http://jakarta.apache.org/ant/`

Ant は、Java クラスを使って拡張した Java ベースのビルドツールです。シェルコマンドを使う代わりに、XML ドキュメントを使ってアセンブリ手順を宣言します。各タスクは、特定のタスクインタフェースを実装したオブジェクトによって実行されます。

Sun ONE Application Server (Solaris 9 にバンドルされている場合もバンドルされていない場合も) には Apache Ant 1.4.1 が付属します。Sun ONE Application Server に付属するサンプルアプリケーションには、Ant の `build.xml` ファイルが用意されています。詳しくは、83 ページの「サンプルアプリケーション」を参照してください。

Ant を使用する前に、次の操作を実行してください。

- PATH 環境変数に *install_dir*`/bin` を含める (Solaris 9 のバンドル製品では `/usr/sfw/bin`)。Sun ONE Application Server に付属する Ant スクリプト `asant` は、このディレクトリにある。`asant` の詳細な使用方法は、*install_dir*`/samples/docs/ant.html` ファイル内のサンプルアプリケーションマニュアルを参照
- `exec` タスクや `cvs` タスクのようなプラットフォームに固有のアプリケーションを実行している場合は、ANT_HOME 環境変数を Ant のインストールディレクトリに設定する必要がある
    - Solaris 9 のバンドル製品では、ANT_HOME 環境変数に次のディレクトリを含める必要がある
        - `/usr/sfw/bin` - Ant のバイナリ (シェルスクリプト)
        - `/usr/sfw/doc/ant` - HTML ドキュメント

- `/usr/sfw/lib/ant` - Ant を実装する Java クラス

- その他すべてのプラットフォームの ANT_HOME 環境変数は *install_dir*`/lib` とする

この節には Ant に関連する次の項目があります。

- Sun ONE Application Server 7 の Ant タスク
- 再利用可能なサブ要素

標準の Ant タスクに関する詳細は、Ant のマニュアルを参照してください。

`http://jakarta.apache.org/ant/manual/index.html`

Sun ONE Application Server 7 の Ant タスク

Sun ONE Application Server の Ant タスクを使って、モジュールとアプリケーションをアセンブル、配備、および配備解除したり、サーバーインスタンスを構成したりすることができます。タスクには、次のものがあります。

- sun-appserv-deploy
- sun-appserv-undeploy
- sun-appserv-instance
- sun-appserv-component
- sun-appserv-admin
- sun-appserv-jspc

sun-appserv-deploy

次のいずれかをローカルまたはリモートの Sun ONE Application Server インスタンスに配備します。

- エンタープライズアプリケーション (EAR ファイル)
- Web アプリケーション (WAR ファイル)
- Enterprise Java Beans (EJB-JAR ファイル)
- エンタープライズコネクタ (RAR ファイル)
- アプリケーションクライアント

サブ要素

次の表は、sun-appserv-deploy タスクのサブ要素について説明しています。これは、タスクの実行対象となるオブジェクトです。左側の列にはサブ要素名、右側の列には要素の説明を示しています。

sun-appserv-deploy のサブ要素

要素	説明
server	Sun ONE Application Server インスタンス
component	配備するコンポーネント
fileset	指定したパラメータと一致するコンポーネントファイルのセット

属性

次の表は、sun-appserv-deploy タスクの属性を説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

sun-appserv-deploy の属性

属性	デフォルト値	説明
file	なし	component または fileset サブ要素が存在する場合は省略可能、それ以外は必須) 配備するコンポーネント。この属性がファイルを参照する場合、そのファイルは有効なアーカイブである必要がある。この属性がディレクトリを参照する場合、そのディレクトリはすべてのコンポーネントが展開された有効なアーカイブを含んでいる必要がある。upload を false に設定するときは、サーバーマシンの絶対パスを指定する必要がある
name	拡張子を含まないファイル名	(省略可能) 配備するコンポーネントの表示名
type	ファイル名またはディレクトリ名の拡張子で決定される	(省略可能) 配備するコンポーネントのタイプ。有効なタイプは、application、ejb、web、connector。指定しない場合、コンポーネントタイプの決定にはファイル (またはディレクトリ) の拡張子が適用される。application は .ear、ejb は .jar、web は .war、connector は .rar がそれぞれ該当する。ファイル拡張子でコンポーネントを決定できない場合、デフォルト値は application となる

sun-appserv-deploy の属性 (続き)

属性	デフォルト値	説明
force	true	(省略可能) true の場合、コンポーネントがすでにサーバー上にあるときは上書きされる。false の場合、コンポーネントが存在するときは sun-appserv-deploy は失敗する
retrievestubs	クライアントスタブは保存されない	(省略可能) クライアントスタブを保存するディレクトリ。この属性は、入れ子の component 要素によって継承される
precompilejsp	false	(省略可能) true の場合、エンタープライズアプリケーション (.ear) または Web アプリケーション (.war) に含まれるすべての JSP が事前コンパイルされる。その他のコンポーネントタイプでは、この属性は無視される。この属性は、入れ子の component 要素によって継承される
verify	false	(省略可能) true の場合、すべての配備記述子の構文とセマンティックの正確さが自動的に検証される。この属性は、入れ子の component 要素によって継承される
contextroot	拡張子を含まないファイル名	(省略可能) Web モジュール (WAR ファイル) のコンテキストルート。コンポーネントが WAR ファイルでない場合、この属性は無視される
upload	true	(省略可能) true の場合、コンポーネントは配備先のサーバーに転送される。コンポーネントがローカルマシンに配備されている場合、upload を false に設定すると配備にかかる時間を短縮できる
virtualservers	デフォルトの仮想サーバーのみ	(省略可能) 配備の対象となる仮想サーバーをコンマで区切ったリスト。アプリケーションコンポーネント (.ear) または Web コンポーネント (.war) だけに適用され、その他のコンポーネントタイプでは無視される。この属性は、入れ子の server 要素によって継承される
user	admin	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するユーザー名。この属性は、入れ子の server 要素によって継承される
password	なし	アプリケーションサーバーの管理インスタンスにログインするときに使われるパスワード。この属性は、入れ子の server 要素によって継承される

sun-appserv-deploy の属性 (続き)

属性	デフォルト値	説明
host	localhost	(省略可能) ターゲットサーバー。リモートサーバーに配備するときは、完全修飾されたホスト名を使う。この属性は、入れ子の server 要素によって継承される
port	4848	(省略可能) ターゲットサーバーの管理ポート。この属性は、入れ子の server 要素によって継承される
instance	デフォルトインスタンスの名前	(省略可能) ターゲットアプリケーションサーバーインスタンス。この属性は、入れ子の server 要素によって継承される
sunonehome	説明を参照	(省略可能) ローカルに Sun ONE Application Server 7 をインストールするときのインストールディレクトリ。管理クラスの検索に使われる。指定しない場合、コマンドは sunone.home パラメータが設定されているかどうかを確認する。指定する場合、システムクラスパスに管理クラスが含まれている必要がある

例

多くの属性が指定されていない簡単なアプリケーション配備スクリプトを示します。

```
<sun-appserv-deploy
   file="${assemble}/simpleapp.ear"
   password="${password}" />
```

属性がすべて指定された同等のスクリプトを示します。

```
<sun-appserv-deploy
   file="${assemble}/simpleapp.ear"
   name="simpleapp"
   type="application"
   force="true"
   precompilejsp="false"
   verify="false"
   upload="true"
   user="admin"
   password="${password}"
   host="localhost"
   port="4848"
   instance="${default-instance-name}"
   sunonehome="${sunone.home}" />
```

次の例では、リモートサーバーで実行している 1 つの Sun ONE Application Server インスタンスに複数のコンポーネントを配備します。

```
<sun-appserv-deploy password="${password}" host="greg.sun.com"
          sunonehome="/opt/sunone" >
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"
      contextroot="test"/>
   <component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

次の例では、リモートサーバーで実行している 2 つの Sun ONE Application Server インスタンスに複数のコンポーネントを配備します。この例では、どちらのサーバーも同じ admin パスワードを使っています。同じパスワードを使わない場合は、各パスワードをサーバー要素に指定することができます。

```
<sun-appserv-deploy password="${password}" sunonehome="/opt/sunone" >
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"
      contextroot="test"/>
   <component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

次の例では、3 つのコンポーネントが `fileset` 条件と一致します。前の例と同じコンポーネントを配備しますが、コンポーネント固有の属性をいくつか設定できないことに注意してください。コンポーネント固有のすべての属性 (`name`、`type`、および `contextroot`) はデフォルト値を使います。

```
<sun-appserv-deploy password="${password}" host="greg.sun.com"
          sunonehome="/opt/sunone" >
   <fileset dir="${assemble}" includes="**/*.?ar" />
</sun-appserv-deploy>
```

sun-appserv-undeploy

次のいずれかをローカルまたはリモートの Sun ONE Application Server インスタンスから配備解除します。

- エンタープライズアプリケーション (EAR ファイル)
- Web アプリケーション (WAR ファイル)
- Enterprise Java Beans (EJB-JAR ファイル)
- エンタープライズコネクタ (RAR ファイル)
- アプリケーションクライアント

サブ要素

次の表は、sun-appserv-undeploy タスクのサブ要素について説明しています。これは、タスクの実行対象となるオブジェクトです。左側の列にはサブ要素名、右側の列には要素の説明を示しています。

sun-appserv-undeploy のサブ要素

要素	説明
server	Sun ONE Application Server インスタンス
component	配備するコンポーネント
fileset	指定したパラメータと一致するコンポーネントファイルのセット

属性

次の表は、sun-appserv-undeploy タスクの属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

sun-appserv-undeploy の属性

属性	デフォルト値	説明
name	拡張子を含まないファイル名	(component または fileset サブ要素が存在するか、file 属性が指定されている場合は省略可能、それ以外は必須) 配備解除するコンポーネントの表示名
file	なし	(省略可能) 配備解除するコンポーネント。この属性がファイルを参照する場合、そのファイルは有効なアーカイブである必要がある。この属性がディレクトリを参照する場合、そのディレクトリはすべてのコンポーネントが展開された有効なアーカイブを含んでいる必要がある。
type	ファイル名またはディレクトリ名の拡張子で決定される	(省略可能) 配備解除するコンポーネントのタイプ。有効なタイプは、application、ejb、web、connector。指定しない場合、コンポーネントタイプの決定にはファイル (またはディレクトリ) の拡張子が適用される。application は .ear、ejb は .jar、web は .war、connector は .rar がそれぞれ該当する。ファイル拡張子でコンポーネントを決定できない場合、デフォルト値は application となる
user	admin	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するユーザー名。この属性は、入れ子の server 要素によって継承される

sun-appserv-undeploy の属性

属性	デフォルト値	説明
password	なし	アプリケーションサーバーの管理インスタンスにログインするときに使われるパスワード。この属性は、入れ子の server 要素によって継承される
host	localhost	(省略可能) ターゲットサーバー。リモートサーバーに配備するときは、完全修飾されたホスト名を使う。この属性は、入れ子の server 要素によって継承される
port	4848	(省略可能) ターゲットサーバーの管理ポート。この属性は、入れ子の server 要素によって継承される
instance	デフォルトインスタンスの名前	(省略可能) ターゲットアプリケーションサーバーインスタンス。この属性は、入れ子の server 要素によって継承される
sunonehome	説明を参照	(省略可能) ローカルに Sun ONE Application Server 7 をインストールするときのインストールディレクトリ。管理クラスの検索に使われる。指定しない場合、コマンドは sunone.home パラメータが設定されているかどうかを確認する。指定する場合、システムクラスパスに管理クラスが含まれている必要がある

例

多くの属性が指定されていない簡単なアプリケーション配備解除スクリプトを示します。

```
<sun-appserv-undeploy name="simpleapp" password="${password}" />
```

属性がすべて指定された同等のスクリプトを示します。

```
<sun-appserv-undeploy
   name="simpleapp"
   type="application"
   user="admin"
   password="${password}"
   host="localhost"
   port="4848"
   instance="${default-instance-name}"
   sunonehome="${sunone.home}" />
```

次の例は、アーカイブファイル (この場合は EAR および WAR) を使用した配備解除、コンポーネントの名前とタイプ (この例では EJB コンポーネントの配備解除) の使用、および複数のコンポーネントの配備解除を示しています。

```
<sun-appserv-undeploy password="${password}">
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-undeploy>
```

配備プロセスと同様に、1 つのコマンドで複数のサーバーからコンポーネントを配備解除できます。次の例では、Sun ONE Application Server 7 の 2 つの異なるインスタンスから 3 つの同じコンポーネントが削除されています。この例では、どちらのインスタンスも同じパスワードを使っています。

```
<sun-appserv-undeploy password="${password}">
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-undeploy>
```

sun-appserv-instance

1 つまたは複数のアプリケーションサーバーインスタンスの起動、停止、再起動、作成、または削除を行います。

サブ要素

次の表は、sun-appserv-instance タスクのサブ要素について説明しています。これは、タスクの実行対象となるオブジェクトです。左側の列にはサブ要素名、右側の列には要素の説明を示しています。

sun-appserv-instance のサブ要素

要素	説明
server	Sun ONE Application Server インスタンス。

属性

次の表は、sun-appserv-instance タスクの属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

sun-appserv-instance の属性

属性	デフォルト値	説明
action	なし	ターゲットアプリケーションサーバー用の制御コマンド。有効な値は、start、stop、restart、create、および delete。再起動 (restart) すると、停止 (stop) コマンドに続いて起動 (start) コマンドが送信される。Windows 環境では restart コマンドはサポートされない
debug	false	(省略可能) start または restart に action を設定すると、サーバーがデバッグモードで起動するかどうかを指定できる。action にその他の値を指定した場合は、この属性は無視される。true の場合、そのインスタンスのライフタイムを通じて追加のデバッグ出力が生成される。この属性は、入れ子の server 要素によって継承される
instanceport	なし	action が create である場合を除いて省略可能) 新しいインスタンスが作成される場合にポート番号を指定する。それ以外の場合、この属性は無視される。この属性は、入れ子の server 要素によって継承される
local	false	(省略可能) true の場合、action のターゲットはローカルマシン上のインスタンス (localhost) となるため、管理サーバーが稼動している必要はなく、host、port、user、および password 属性は無視される。false の場合は管理サーバーが稼動している必要があり、host、port、user、および password 属性にも適切な値を設定する必要がある。この属性は、入れ子の server 要素によって継承される
domain		(local="true" と設定し、複数のローカルドメインが存在する場合を除いて省略可能) ローカル action のターゲットドメイン。local="false" の場合は、この属性は無視される。この属性は、入れ子の server 要素によって継承される
user	admin	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するユーザー名。この属性は、入れ子の server 要素によって継承される
password	なし	(local が true である場合を除いて必須) アプリケーションサーバーの管理インスタンスにログインするときに使用するパスワード。この属性は、入れ子の server 要素によって継承される

sun-appserv-instance の属性 (続き)

属性	デフォルト値	説明
host	localhost	(省略可能) ターゲットサーバー。リモートサーバーの場合は、完全修飾されたホスト名を使う。この属性は、入れ子の server 要素によって継承される
port	4848	(省略可能) ターゲットサーバーの管理ポート。この属性は、入れ子の server 要素によって継承される
instance	デフォルトインスタンスの名前	ターゲットアプリケーションサーバーインスタンス。この属性は、入れ子の server 要素によって継承される
sunonehome	説明を参照	(省略可能) ローカルに Sun ONE Application Server 7 をインストールするときのインストールディレクトリ。管理クラスの検索に使われる。指定しない場合、コマンドは sunone.home パラメータが設定されているかどうかを確認する。指定する場合、システムクラスパスに管理クラスが含まれている必要がある

例

次の例では、local Sun ONE Application Server 7 インスタンスを起動します。

```
<sun-appserv-instance action="start" password="${password}"
   instance="${default-instance-name}"/>
```

属性がすべて指定された同等のスクリプトを示します。

```
<sun-appserv-instance
   action="start"
   user="admin"
   password="${password}"
   host="localhost"
   port="4848"
   instance="${default-instance-name}"
   sunonehome="${sunone.home}" />
```

単一のコマンドで複数のサーバーを制御することができます。次の例では、2 つのサーバーを再起動します。この場合、サーバーはそれぞれ別のパスワードを使用します。

```
<sun-appserv-instance action="restart"
       instance="${default-instance-name}"/>
   <server host="greg.sun.com" password="${password.greg}"/>
   <server host="joe.sun.com" password="${password.joe}"/>
</sun-appserv-instance>
```

次の例では、Sun ONE Application Server 7 の新しいインスタンスを作成します。

```
<sun-appserv-instance
   action="create" instanceport="8080"
   password="${password}"
   instance="development" />
```

属性がすべて指定された同等のスクリプトを示します。

```
<sun-appserv-instance
   action="create"
   instanceport="8080"
   user="admin"
   password="${password}"
   host="localhost"
   port="4848"
   instance="development"
   sunonehome="${sunone.home}" />
```

単一のコマンドで複数のサーバー上にインスタンスを作成できます。次の例では、2 つの異なるサーバー上に qa という名前の新しいインスタンスを作成しています。この場合は、両方のサーバーが同じパスワードを使用しています。

```
<sun-appserv-instance
       action="create"
       instanceport="8080"
       instance="qa"
       password="${password}>
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
</sun-appserv-instance>
```

次のように、各サーバーからこれらのインスタンスを削除することもできます。

```
<sun-appserv-instance
       action="delete"
       instance="qa"
       password="${password}>
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
</sun-appserv-instance>
```

次のように、server サブ要素の属性を使って異なるインスタンス名およびインスタンスポートを指定することもできます。

```
<sun-appserv-instance action="create" password="${password}>
   <server host="greg.sun.com" instanceport="8080" instance="qa"/>
   <server host="joe.sun.com" instanceport="9090"
       instance="integration-test"/>
</sun-appserv-instance>
```

sun-appserv-component

Sun ONE Application Server 7 に配備されている 次の J2EE コンポーネントタイプを有効または無効にします。

- エンタープライズアプリケーション (EAR ファイル)
- Web アプリケーション (WAR ファイル)
- Enterprise Java Beans (EJB-JAR ファイル)
- エンタープライズコネクタ (RAR ファイル)
- アプリケーションクライアント

コンポーネントを有効または無効にするのにアーカイブを指定する必要はありません。必要となるのはコンポーネント名だけです。ただし、コンポーネントアーカイブは、コンポーネント名を示しているため、使用することもできます。

サブ要素

次の表は、sun-appserv-component タスクのサブ要素について説明しています。これは、タスクの実行対象となるオブジェクトです。左側の列にはサブ要素名、右側の列には要素の説明を示しています。

sun-appserv-component のサブ要素

要素	説明
server	Sun ONE Application Server インスタンス。
component	配備するコンポーネント
fileset	指定したパラメータと一致するコンポーネントファイルのセット

属性

次の表は、sun-appserv-component タスクの属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

sun-appserv-component の属性

属性	デフォルト値	説明
action	なし	ターゲットアプリケーションサーバー用の制御コマンド。有効な値は、enable および disable
name	拡張子を含まないファイル名	component または fileset サブ要素が存在するか、または file 属性が指定されている場合は省略可能、それ以外は必須) 有効または無効にするコンポーネントの表示名

sun-appserv-component の属性 (続き)

属性	デフォルト値	説明
file	なし	(省略可能) 有効または無効にするコンポーネント。この属性がファイルを参照する場合、そのファイルは有効なアーカイブである必要がある。この属性がディレクトリを参照する場合、そのディレクトリはすべてのコンポーネントが展開された有効なアーカイブを含んでいる必要がある。
type	ファイル名またはディレクトリ名の拡張子で決定される	(省略可能) 有効または無効にするコンポーネントのタイプ。有効なタイプは、application、ejb、web、connector。指定しない場合、コンポーネントタイプの決定にはファイル (またはディレクトリ) の拡張子が適用される。application は .ear、ejb は .jar、web は .war、connector は .rar がそれぞれ該当する。ファイル拡張子でコンポーネントを決定できない場合、デフォルト値は application となる
user	admin	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するユーザー名。この属性は、入れ子の server 要素によって継承される
password	なし	アプリケーションサーバーの管理インスタンスにログインするときに使われるパスワード。この属性は、入れ子の server 要素によって継承される
host	localhost	(省略可能) ターゲットサーバー。リモートサーバーを有効または無効にするときは、完全修飾されたホスト名を使う。この属性は、入れ子の server 要素によって継承される
port	4848	(省略可能) ターゲットサーバーの管理ポート。この属性は、入れ子の server 要素によって継承される
instance	デフォルトインスタンスの名前	(省略可能) ターゲットアプリケーションサーバーインスタンス。この属性は、入れ子の server 要素によって継承される
sunonehome	説明を参照	(省略可能) ローカルに Sun ONE Application Server 7 をインストールするときのインストールディレクトリ。管理クラスの検索に使われる。指定しない場合、コマンドは sunone.home パラメータが設定されているかどうかを確認する。指定する場合、システムクラスパスに管理クラスが含まれている必要がある

例

コンポーネントを無効にする簡単な例を示します。

```
<sun-appserv-component
   action="disable"
   name="simpleapp"
   password="${password}" />
```

コンポーネントを有効にする簡単な例を示します。

```
<sun-appserv-component
   action="enable"
   name="simpleapp"
   password="${password}" />
```

属性がすべて指定された同等のスクリプトを示します。

```
<sun-appserv-component
   action="enable"
   name="simpleapp"
   type="application"
   user="admin"
   password="${password}"
   host="localhost"
   port="4848"
   instance="${default-instance-name}"
   sunonehome="${sunone.home}" />
```

次の例は、アーカイブファイル (この例では EAR および WAR) またはコンポーネント名およびタイプ (この例では、EJB コンポーネント) を使って複数のコンポーネントを無効にする方法を示しています。

```
<sun-appserv-component action="disable" password="${password}">
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-component>
```

複数のサーバー上にあるコンポーネントを、単一のタスクで有効または無効にすることができます。次の例は、Sun ONE Application Server 7 の 2 つの異なるインスタンス上で 3 つの同じコンポーネントを有効にする方法を示しています。この例では、両インスタンスのパスワードは同じです。

```
<sun-appserv-component action="enable" password="${password}">
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-component>
```

sun-appserv-admin

Sun ONE Application Server 7 上で実行される任意の管理コマンドおよびスクリプトを有効にします。これは、特定の Ant タスクが開発されていないか、または関連コマンド一式が単一のスクリプト内にある場合に有効です。

サブ要素

次の表は、sun-appserv-admin タスクのサブ要素について説明しています。これは、タスクの実行対象となるオブジェクトです。左側の列にはサブ要素名、右側の列には要素の説明を示しています。

sun-appserv-admin のサブ要素

要素	説明
server	Sun ONE Application Server インスタンス。

属性

次の表は、sun-appserv-admin タスクの属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

sun-appserv-admin の属性

属性	デフォルト値	説明
command	なし	(command、commandfile、または explicitcommand のうちいずれか 1 つが必要) 実行するコマンド。user、password、host、port、または instance 属性も指定されている場合、実行前にその値も自動的にコマンドに挿入される。コマンド文字列にこれらのオプションが指定されている場合、対応する属性の値は無視される
commandfile	なし	(command、commandfile、または explicitcommand のうちいずれか 1 つが必要) 実行するコマンドスクリプト。commandfile を使用する場合、その他すべての属性の値は無視される。commandfile が参照するスクリプトの最後に exit コマンドを挿入する必要がある。exit を省略すると、コマンドスクリプトを呼び出した後に、Ant タスクが停止したように見えることがある

sun-appserv-admin の属性 (続き)

属性	デフォルト値	説明
explicitcommand	なし	(command、commandfile、または explicitcommand のうちいずれか 1 つが必要) 実行するコマンド。コマンドプロセスは実行されず、ほかのすべての属性は無視される
user	admin	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するユーザー名。この属性は、入れ子の server 要素によって継承される
password	なし	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するパスワード。この属性は、入れ子の server 要素によって継承される
host	localhost	(省略可能) ターゲットサーバー。リモートサーバーの場合は、完全修飾されたホスト名を使う。この属性は、入れ子の server 要素によって継承される
port	4848	(省略可能) ターゲットサーバーの管理ポート。この属性は、入れ子の server 要素によって継承される
instance	デフォルトインスタンスの名前	(省略可能) ターゲットアプリケーションサーバーインスタンス。この属性は、入れ子の server 要素によって継承される
sunonehome	説明を参照	(省略可能) ローカルに Sun ONE Application Server 7 をインストールするときのインストールディレクトリ。管理クラスの検索に使われる。指定しない場合、コマンドは sunone.home パラメータが設定されているかどうかを確認する。指定する場合、システムクラスパスに管理クラスが含まれている必要がある

sun-appserv-jspc

Sun ONE Application Server による初回の呼び出し用に、JSP ソースコードを Sun ONE Application Server 互換の Java コードに事前コンパイルします。このタスクを使うことで、JSP ページへのアクセスや、JSP ソースコードの構文チェックを迅速に行えます。生成される Java コードを javac タスクに送り、JSP のクラスファイルを生成することもできます。

サブ要素

なし

属性

次の表は、sun-appserv-jspc タスクの属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

sun-appserv-jspc の属性

属性	デフォルト値	説明
destdir		Java ソースファイルの生成先ディレクトリ
srcdir		(srcdir または webapp が必要) JSP ファイルが保存されているソースディレクトリ
webapp		(srcdir または webapp が必要) Web アプリケーションが保存されているソースディレクトリ。ディレクトリに含まれるすべての JSP ページが再帰的に解析される。ベースディレクトリの下に WEB-INF サブディレクトリが必要である。webapp を使用した場合、sun-appserv-jspc はすべての依存性チェックをコンパイラに渡す
verbose	2	(省略可能) コンパイラに渡される冗長性レベル
classpath		(省略可能) JSP コンパイラ実行用のクラスパス
classpathref		(省略可能) JSP コンパイラのクラスパスへの参照
uribase	/	(省略可能) JSP ページに含まれる関連 URI 参照の URI コンテキスト。このコンテキストが存在しない場合、uriroot の宣言値または派生値に関連する JSP ファイルの場所が参照される。明示的に宣言された JSP ファイルから変換したページだけが対象となる
uriroot	説明を参照	(省略可能) Web アプリケーションのルートディレクトリ。URI ファイルはここで解決される。このディレクトリを指定しない場合、最初の JSP ページのそれぞれの親ディレクトリで WEB-INF ディレクトリが検索され、このディレクトリを含むディレクトリのうち、最初の JSP ページに最も近いものが使用される。WEB-INF ディレクトリが見つからない場合、sun-appserv-jspc の呼び出し側ディレクトリが使用される。明示的に宣言された JSP ファイル (タグライブラリを含む) から変換したページだけが対象となる
package		(省略可能) Java クラスの生成先パッケージ

sun-appserv-jspc の属性 (続き)

属性	デフォルト値	説明
failonerror	true	(省略可能) true の場合、エラーを検出すると JSP のコンパイルは失敗する
sunonehome	説明を参照	(省略可能) ローカルに Sun ONE Application Server 7 をインストールするときのインストールディレクトリ。管理クラスの検索に使われる。指定しない場合、コマンドは sunone.home パラメータが設定されているかどうかを確認する。指定する場合、システムクラスパスに管理クラスが含まれている必要がある

例

次の例では、webapp 属性を使って JSP ファイルから Java ソースファイルを生成します。sun-appserv-jspc タスクが終了すると、生成された Java ファイルをクラスファイルにコンパイルする javac タスクが直ちに実行されます。javac タスクの classpath の値は、空白文字を挿入せずにすべてを 1 行で記述する必要があります。

```
<sun-appserv-jspc
   destdir="${assemble.war}/generated"
   webapp="${assemble.war}"
   classpath="${assemble.war}/WEB-INF/classes"
   sunonehome="${sunone.home}" />
<javac
   srcdir="${assemble.war}/WEB-INF/generated"
   destdir="${assemble.war}/WEB-INF/generated"
   debug="on"
   classpath="${assemble.war}/WEB-INF/classes:${sunone.home}/lib/
      appserv-rt.jar:${sunone.home}/lib/appserv-ext.jar">
   <include name="**/*.java"/>
</javac>
```

再利用可能なサブ要素

Sun ONE Application Server 7 の Ant タスクの再利用可能なサブ要素は次のとおりです。これは、タスクの実行対象となるオブジェクトです。

- server
- component
- fileset

server

Sun ONE Application Server インスタンスを指定します。1 つのタスクが複数のサーバーインスタンスで動作できます。server 属性は、親タスクの対応属性をオーバーライドするため、親タスクの属性にはデフォルト値が適用されます。

サブ要素

なし

属性

次の表は、server 要素の属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

server の属性

属性	デフォルト値	説明
user	admin	(省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するユーザー名
password	なし	(親タスクで指定されている場合は省略可能) アプリケーションサーバーの管理インスタンスにログインするときに使用するパスワード
host	localhost	(省略可能) ターゲットサーバー。リモートサーバーをターゲット指定するときは、完全修飾されたホスト名を使う
port	4848	(省略可能) ターゲットサーバーの管理ポート
instance	デフォルトインスタンスの名前	(省略可能) ターゲットアプリケーションサーバーインスタンス
domain		(local="true" と設定し、複数のローカルドメインが存在する場合を除いて sun-appserv-instance にだけ適用される) ローカル action のターゲットドメイン。local="false" の場合は、この属性は無視される
instanceport	なし	action が create である場合を除いて sun-appserv-instance に適用される) 新しいインスタンスが作成される場合にポート番号を指定する。それ以外の場合、この属性は無視される。

server の属性 (続き)

属性	デフォルト値	説明
debug	false	(省略可能。sun-appserv-instance だけに適用される) start に action を設定すると、サーバーがデバッグモードで起動するかどうかを指定できる。action にその他の値を指定した場合は、この属性は無視される。true の場合、そのインスタンスのライフタイムを通じて追加のデバッグ出力が生成される。
local	false	(省略可能。sun-appserv-instance だけに適用される) true の場合、action のターゲットはローカルマシン上のインスタンス (localhost) となるため、管理サーバーが稼動している必要はなく、host、port、user、および password 属性は無視される。false の場合は管理サーバーが稼動している必要があり、host、port、user、および password 属性にも適切な値を設定する必要がある
upload	true	(省略可能。sun-appserv-deploy だけに適用される) true の場合、コンポーネントが配備先のサーバーに転送される。コンポーネントをローカルマシンに配備する場合、upload を false に設定すると、配備にかかる時間を短縮できる
virtualservers	デフォルトの仮想サーバーのみ	(省略可能。sun-appserv-deploy だけに適用される) 配備の対象となる仮想サーバーをコンマで区切ったリスト。アプリケーションコンポーネント (.ear) または Web コンポーネント (.war) だけに適用され、その他のコンポーネントタイプでは無視される

例

1 つのタスクで複数のサーバーを制御できます。この例では、異なるパスワードを使って 2 つのサーバーを起動します。一方のサーバーだけをデバッグモードで起動します。

```
<sun-appserv-instance action="start">
   <server host="greg.sun.com" password="${password.greg}"/>
   <server host="joe.sun.com" password="${password.joe}"
      debug="true"/>
</sun-appserv-instance>
```

1 つのタスクで複数のサーバーにインスタンスを作成できます。次の例では、2 つの異なるサーバー上に qa という名前の新しいインスタンスを作成しています。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-instance action="create" instanceport="8080"
       instance="qa" password="${password}>
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
</sun-appserv-instance>
```

次のように、各サーバーからこれらのインスタンスを削除することもできます。

```
<sun-appserv-instance action="delete" instance="qa"
       password="${password}>
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
</sun-appserv-instance>
```

入れ子の server 要素の属性を使って、別のインスタンス名とポート番号を指定することもできます。

```
<sun-appserv-instance action="create" password="${password}>
   <server host="greg.sun.com" instanceport="8080" instance="qa"/>
   <server host="joe.sun.com" instanceport="9090"
      instance="integration-test"/>
</sun-appserv-instance>
```

複数のサーバーに複数のコンポーネントを配備できます (入れ子の component 要素を参照)。次の例では、リモートサーバーで実行している 2 つの Sun ONE Application Server インスタンスにコンポーネントを 1 つずつ配備します。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-deploy password="${password}" sunonehome="/opt/s1as7" >
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"
      contextroot="test"/>
   <component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

複数のサーバーの複数のコンポーネントを配備解除することもできます。次の例では、2 つの異なるインスタンスから 3 つの同じコンポーネントを削除します。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-undeploy password="${password}">
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-undeploy>
```

複数のサーバーにあるコンポーネントを有効化または無効化することができます。次の例では、2 つの異なるインスタンスで 3 つの同じコンポーネントを有効化します。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-component action="enable" password="${password}">
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-component>
```

component

J2EE コンポーネントを指定します。1 つのタスクが複数のコンポーネントで動作するようになります。component 属性は、親タスクの対応属性をオーバーライドするため、親タスクの属性にはデフォルト値が適用されます。

サブ要素

なし

属性

次の表は、component 要素の属性について説明しています。左側の列には属性名、中央の列にはデフォルト値、右側の列には属性の説明を示しています。

component の属性

属性	デフォルト値	説明
file	なし	(親タスクが sun-appserv-undeploy または sun-appserv-component の場合は省略可能) ターゲットコンポーネント。この属性がファイルを参照する場合、そのファイルは有効なアーカイブである必要がある。この属性がディレクトリを参照する場合、そのディレクトリはすべてのコンポーネントが展開された有効なアーカイブを含んでいる必要がある。upload を false に設定するときは、サーバーマシンの絶対パスを指定する必要がある
name	拡張子を含まないファイル名	(省略可能) コンポーネントの表示名。

component の属性 (続き)

属性	デフォルト値	説明
type	ファイル名またはディレクトリ名の拡張子で決定される	(省略可能) コンポーネントのタイプ。有効なタイプは、application、ejb、web、connector。指定しない場合、コンポーネントタイプの決定にはファイル (またはディレクトリ) の拡張子が適用される。application は .ear、ejb は .jar、web は .war、connector は .rar がそれぞれ該当する。ファイル拡張子でコンポーネントを決定できない場合、デフォルト値は application となる
force	true	(省略可能。sun-appserv-deploy だけに適用される) true の場合、コンポーネントがすでにサーバー上にあるときは上書きされる。コンポーネントが存在する状態で false に設定すると、含んでいる要素の処理は失敗する
precompilejsp	false	(省略可能。sun-appserv-deploy だけに適用される) true の場合、エンタープライズアプリケーション (.ear) または Web アプリケーション (.war) に含まれるすべての JSP が事前コンパイルされる。その他のコンポーネントタイプでは、この属性は無視される
retrievestubs	クライアントスタブは保存されない	(省略可能。sun-appserv-deploy だけに適用される) クライアントスタブを保存するディレクトリ
contextroot	拡張子を含まないファイル名	(省略可能。sun-appserv-deploy だけに適用される) Web モジュール (WAR ファイル) のコンテキストルート。コンポーネントが WAR ファイルでない場合、この属性は無視される
verify	false	(省略可能。sun-appserv-deploy だけに適用される) true の場合、すべての配備記述子の構文とセマンティックの正確さが自動的に検証される。

例

1 つのタスクで複数のコンポーネントを配備できます。次の例では、リモートサーバーで実行している 1 つの Sun ONE Application Server インスタンスに各コンポーネントを配備します。

```
<sun-appserv-deploy password="${password}" host="greg.sun.com"
       sunonehome="/opt/s1as7" >
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"
      contextroot="test"/>
   <component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

1 つのタスクで複数のコンポーネントを配備解除することもできます。次の例では、アーカイブファイル (この例では EAR と WAR) および EJB コンポーネントの名前とタイプを使っています。

```
<sun-appserv-undeploy password="${password}">
   <component file="${assemble}/simpleapp.ear"/
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-undeploy>
```

複数のサーバーに複数のコンポーネントを配備できます。次の例では、リモートサーバーで稼動している 2 つのインスタンスにコンポーネントを 1 つずつ配備します。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-deploy password="${password}" sunonehome="/opt/s1as7" >
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"
       contextroot="test"/>
   <component file="${assemble}/simplebean.jar"/>
</sun-appserv-deploy>
```

複数のサーバーの複数のコンポーネントを配備解除することもできます。次の例では、2 つの異なるインスタンスから 3 つの同じコンポーネントを削除します。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-undeploy password="${password}">
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-undeploy>
```

複数のコンポーネントを有効化または無効化することができます。次の例では、アーカイブファイル (この例では EAR と WAR) および EJB コンポーネントの名前とタイプを使って複数のコンポーネントを無効化しています。

```
<sun-appserv-component action="disable" password="${password}">
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-component>
```

複数のサーバーにある複数のコンポーネントを有効化または無効化することもできます。次の例では、2 つの異なるインスタンスで 3 つの同じコンポーネントを有効化します。どちらのサーバーでも同じパスワードを使用します。

```
<sun-appserv-component action="enable" password="${password}">
   <server host="greg.sun.com"/>
   <server host="joe.sun.com"/>
   <component file="${assemble}/simpleapp.ear"/>
   <component file="${assemble}/simpleservlet.war"/>
   <component name="simplebean" type="ejb"/>
</sun-appserv-component>
```

fileset

指定したパラメータと一致するコンポーネントファイルを選択します。`fileset` がサブ要素として含まれている場合、`fileset` の各ファイルに対して、含んでいる要素の `name` および `contextroot` 属性にデフォルト値を使う必要があります。詳細は、次のサイトを参照してください。

`http://jakarta.apache.org/ant/manual/CoreTypes/fileset.html`

アプリケーション配備記述子ファイル

Sun ONE Application Server アプリケーションには、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`)
- この節で説明しているオプションの Sun ONE Application Server 固有のファイル (`sun-application.xml`)

この節には次のトピックがあります。

- sun-application_1_3-0.dtd ファイル
- sun-application.xml ファイル内の要素
- サンプルアプリケーション XML ファイル

sun-application_1_3-0.dtd ファイル

`sun-application_1_3-0.dtd` ファイルでは、`sun-application.xml` ファイルの構造、および記述できる要素とそのサブ要素および属性が定義されています。`sun-application_1_3-0.dtd` ファイルは、*install_dir*`/lib/dtds` ディレクトリにあります。

注	`sun-application_1_3-0.dtd` ファイルを編集しないでください。このファイルの内容は、Sun ONE Application Server の新しいバージョンだけで変更されます。

DTD ファイルおよび XML の全般的な情報については、次のサイトにある XML 仕様書を参照してください。

`http://www.w3.org/TR/REC-xml`

DTD ファイルに定義される各要素 (対応する XML ファイルに含まれていることもあります) には次の情報が含まれます。

- サブ要素
- データ
- 属性

サブ要素

要素にはサブ要素を含めることができます。たとえば、次のコードは、sun-application の要素を定義しています。

```
<!ELEMENT sun-application (web*, pass-by-reference?, unique-id?,
security-role-mapping*)>
```

この ELEMENT タグは、sun-application 要素に web、pass-by-reference、unique-id、および security-role-mapping 要素を含めることができることを示しています。

次の表は、サブ要素の末尾に追加したオプション文字によって決定されるサブ要素の必要規則 (指定可能回数) について説明しています。左側の列にはサブ要素の終了文字、右側の列には対応する必要指定数を示しています。

要件規則とサブ要素のサフィックス

サブ要素のサフィックス	必要指定数
*element**	このサブ要素を含まないか、1 個以上含めることができる
element?	このサブ要素を含まないか、1 個含めることができる
element+	このサブ要素を 1 個以上含まなければならない
element (サフィックスなし)	このサブ要素を 1 個だけ含まなければならない

要素にほかの要素を含めることができない場合は、カッコで囲まれた要素名のリストの代わりに、EMPTY または (#PCDATA) が表示されます。

データ

要素の中には、サブ要素の代わりに文字データを含むものもあります。これらの要素は、次の形式で定義されます。

```
<!ELEMENT element-name (#PCDATA)>
```

次に例を示します。

```
<!ELEMENT role-name (#PCDATA)>
```

sun-application.xml ファイル内では、データ要素内の空白スペースはデータの一部として扱われます。そのため、データ要素で区切られたデータの前後には余分な空白がないようにする必要があります。次に例を示します。

```
<role-name>manager</role-name>
```

属性

ATTLIST タグを持つ要素には属性が含まれています。sun-application.xml ファイル内の要素には、属性は含まれていません。

sun-application.xml ファイル内の要素

この節では、sun-application.xml ファイルの次の要素について説明します。

- sun-application
- web
- web-uri
- context-root
- pass-by-reference
- unique-id
- security-role-mapping
- role-name
- principal-name
- group-name

sun-application

アプリケーション用の Sun ONE Application Server 固有の設定を定義します。これはルート要素であり、sun-application.xml ファイル内には sun-application 要素が 1 つだけ存在します。

サブ要素

次の表では、sun-application 要素のサブ要素について説明しています。左側の列にはサブ要素名、中央の列には必要指定数、右側の列には要素の説明を示しています。

sun-application のサブ要素

要素	必要指定数	説明
web	0 または 1 個以上	アプリケーションの Web 層の設定を指定する

sun-application のサブ要素 (続き)

要素	必要指定数	説明
pass-by-reference	0 または 1 個	EJB モジュールが pass-by-value セマンティクスまたは pass-by-reference セマンティクスのどちらを使用するのかを決定する
unique-id	0 または 1 個	アプリケーション用の固有 ID を含む
security-role-mapping	0 または 1 個以上	対応する J2EE XML ファイル内のロールを、ユーザーまたはグループに割り当てる

web

アプリケーションの Web 層の設定を指定します。

サブ要素

次の表では、Web 要素のサブ要素について説明しています。左側の列にはサブ要素名、中央の列には必要指定数、右側の列には要素の説明を示しています。

web のサブ要素

要素	必要指定数	説明
web-uri	1 個のみ	アプリケーション用の Web URI を含む
context-root	1 個のみ	アプリケーション用の Web コンテキストルートを含む

web-uri

アプリケーション用の Web URI を含みます。application.xml ファイル内の対応する要素と一致する必要があります。

サブ要素

なし

context-root

アプリケーション用の Web コンテキストルートを含みます。application.xml ファイル内の対応する要素よりも優先されます。

サブ要素
なし

pass-by-reference

`false` (この要素が存在しない場合はこれがデフォルト値) の場合、このアプリケーションでは、EJB 仕様で要求される pass-by-value セマンティクスが使用されます。`true` の場合、このアプリケーションでは、pass-by-reference セマンティクスが使用されます。`sun-application.xml` ファイル内のこの要素の設定は、アプリケーション内のすべての EJB モジュールに適用されます。

個別に配備された EJB モジュール用に、`sun-ejb-jar.xml` ファイル内に同じ要素を設定することもできます。Bean レベルとアプリケーションレベルの両方で pass-by-reference を使用する場合は、アプリケーションレベルより Bean レベルが優先されます。詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

サブ要素
なし

unique-id

アプリケーション用の固有 ID を含みます。この値は、アプリケーションが配備または配備解除されるたびに自動的に更新されます。この値を手動で変更しないでください。

サブ要素
なし

security-role-mapping

ユーザーおよびグループにロールを割り当てます。最低 1 つの主体名またはグループ名が必要ですが、必ずしも両方は必要ありません。

サブ要素
次の表では、`security-role-mapping` 要素のサブ要素について説明しています。左側の列にはサブ要素名、中央の列には必要指定数、右側の列には要素の説明を示しています。

security-role-mapping のサブ要素

要素	必要指定数	説明
role-name	1 個のみ	application.xml ファイルの security-role 要素内に role-name を含む
principal-name	group-name がない場合は 1 個以上、ある場合は 0 または 1 個以上	主体 (ユーザー) 名を含む
group-name	principal-name がない場合は 1 個以上、ある場合は 0 または 1 個以上	グループ名を含む

role-name

application.xml ファイルの security-role 要素内に role-name を含みます。

サブ要素
なし

principal-name

主体 (ユーザー) 名を含みます。

サブ要素
なし

group-name

グループ名を含みます。

サブ要素
なし

サンプルアプリケーション XML ファイル

この節には次の項目があります。

- サンプル application.xml ファイル
- サンプル sun-application.xml ファイル

サンプル application.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE
Application 1.3//EN' 'http://java.sun.com/dtd/application_1_3.dtd'>

<application>

   <display-name>app_stateless-simple</display-name>

   <description>Application description</description>

   <module>

       <ejb>stateless-simpleEjb.jar</ejb>

   </module>

   <module>

       <web>

          <web-uri>stateless-simple.war</web-uri>

          <context-root>helloworld</context-root>

      </web>

   </module>

</application>
```

サンプル sun-application.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sun-application PUBLIC '-//Sun Microsystems, Inc.//DTD Sun
ONE Application Server 7.0 J2EE Application 1.3//EN'
'http://www.sun.com/software/sunone/appserver/dtds/sun-application_
1_3-0.dtd'>

<sun-application>

  <unique-id>67488732739338240</unique-id>

</sun-application>
```

第 5 章

J2EE アプリケーションのデバッグ

この章では、Sun ONE Application Server 7 でアプリケーションをデバッグするためのガイドラインについて説明します。この章には次の節があります。

- デバッグの有効化
- JPDA オプション
- Sun ONE Studio を利用したデバッグ
- JSP のデバッグ
- デバッグ用スタックトレースの生成
- Sun ONE Message Queue のデバッグ
- ログ
- プロファイル

アプリケーションをデバッグするときは、この章で説明しているとおりに、`server.xml` ファイルを編集する必要があります。このファイルの全般的な説明については、『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。

デバッグの有効化

デバッグを有効化するときは、ローカルとリモートの両方でデバッグを有効化します。

次の方法のいずれかでデバッグを有効化できます。

- 管理インタフェースの使用 (推奨)
- `server.xml` ファイルの編集

Sun ONE Application Server のデバッグは、JPDA (Java Platform Debugger Architecture) に基づいています。詳細は、141 ページの「JPDA オプション」を参照してください。

管理インタフェースの使用

デバッグを有効にするには、次の手順に従います。

1. サーバーインスタンスのページに移動します。
2. 「一般」タブを選択します。
3. 「デバッグモードで起動」ボックスにチェックマークをつけます。
4. 「変更の適用」ボタンを選択します。
5. サーバーを再起動します。
6. 「一般」オプションの「JVM 設定」タブを選択します。
7. 「デバッグオプション」フィールドの「address=*port_number*」に表示されるポート番号を書き留めます。デバッガを使用するときに、この番号が必要になります。
8. JPDA オプションを追加するときは、次のオプション手順を実行します。
   a. 「デバッグオプション」に任意の JPDA デバッグオプションを追加します。141 ページの「JPDA オプション」を参照してください。
   b. 「保存」ボタンを選択します。
   c. 手順 2 〜手順 5 を繰り返します。

server.xml ファイルの編集

デバッグを有効にするには、`server.xml` ファイル内の `java-config` 要素の次の属性を設定します。

- `debug-enabled="true"` に設定してデバッグを有効にする
- `debug-options` 属性に任意の JPDA デバッグオプションを追加する。141 ページの「JPDA オプション」を参照
- JVM とデバッガの接続に使用するポートを指定するときは、`debug-options` 属性の `address=`*port_number* に番号を指定する

`server.xml` ファイルの詳細は、『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。

JPDA オプション

Sun ONE Application Server に用意されているデフォルトの JPDA オプションは次のとおりです。

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n
```

`suspend=y` に設定すると、JVM は、中断モードで起動し、デバッガが接続されるまで中断状態のままになります。これは、JVM が起動すると同時にデバッグを開始したい場合に便利です。

JVM とデバッガの接続に使用するポートを指定するには、`address=`*port_number* に番号を指定します。

追加のオプションを含めることもできます。JPDA デバッグオプションのリストは次のサイトで入手できます。

`http://java.sun.com/products/jpda/doc/conninv.html#Invocation`

Sun ONE Studio を利用したデバッグ

Sun ONE Application Server で Sun ONE Studio 4 デバッガを使用するには、次の手順に従います。

1. Sun ONE Studio を起動し、デバッグするアプリケーションソースコードを含んでいるディレクトリをマウントします。
2. 「実行時」タブを選択し、デバッグモード (ローカルまたはリモート) で起動する Sun ONE Application Server インスタンスを表示します。
3. サーバーインスタンスを右クリックし、表示されるメニューから「状態」を選択します。「状態」ウィンドウが表示されます。
4. サーバーインスタンスがデバッグモードで稼動していないときは、「サーバーインスタンスを停止」を選択してから「デバッグモードで起動」を選択します。
5. サーバーインスタンスがデバッグモードで稼動しているときは、「状態」ウィンドウと状態行にポート番号 (*port_number*) が表示されます。このポート番号を書き留めます。
6. 「デバッグ」メニューの「接続」オプションを選択します。
7. 必要に応じて「コネクタ」テキストフィールドを「SocketAttach (ソケットでその他の VM に接続します。)」に変更します。
8. 「ホスト」テキストボックスにアプリケーションサーバーのホスト名を入力します。

9. 「ポート」テキストボックスにポート番号 (*port_number*) を入力し、「了解」を選択します。

これで、Sun ONE Studio を使って Java クラスをデバッグできるようになりました。

Sun ONE Studio を使ったアプリケーションのデバッグについてヘルプを参照するには、「ヘルプ」から「内容」を選択し、「Java プログラムのデバッグ」を選択します。Sun ONE Studio 4, Enterprise Edition for Java のチュートリアルも参考になります。

JSP のデバッグ

JSP のデバッグに Sun ONE Studio 4 を使う場合、JSP コードと生成されたサーブレットコードの両方にブレークポイントを設定し、これらの間で表示を切り替えて両方の同一ブレークポイントを見ることができます。

Sun ONE Studio でデバッグを設定するには、前の節を参照してください。詳細は、Sun ONE Studio 4, Enterprise Edition for Java のチュートリアルを参照してください。

デバッグ用スタックトレースの生成

次のサイトで説明されているように、Java のデバッグ用スタックトレースを生成できます。

`http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/`

`-Xrs` フラグが (`<jvm-options>` の下にある) `server.xml` ファイル内に (信号の使用を減らすために) 設定されている場合、スタックトレースを生成する前にコメントアウトします。`-Xrs` フラグを使用すると、トレースを生成するための信号を送信するときに、サーバーでコアダンプが発生して再起動される場合があります。

スタックトレースは、`server.xml` 内の `log-service` 属性に基づいて、システムログファイルまたは `stderr` に移動します。

`server.xml` ファイルの詳細は、『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。

Sun ONE Message Queue のデバッグ

Sun ONE Message Queue は、ブローカロガーを持っているため、JMS アプリケーション (メッセージ駆動型 Bean を含む) のデバッグに役立ちます。ロガーの冗長性を調節したり、ブローカの `-tty` オプションを使ってロガー出力をコンソールに送信することができます。詳細は、『Sun ONE Message Queue 管理者ガイド』を参照してください。

ログ

Sun ONE Application Server のログファイルを使うとアプリケーションのデバッグに役立つことがあります。ログについての全般的な情報については、『Sun ONE Application Server 管理者ガイド』を参照してください。`server.xml` ファイルでのログ設定の詳細は、『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。

次の方法のいずれかでログの設定を変更できます。

- 管理インタフェースの使用
- `server.xml` ファイルの編集

管理インタフェースの使用

ログの設定を変更するには、次の手順に従います。

1. サーバーインスタンスのページに移動します。
2. 「一般」オプションの「ログ」タブを選択します。
3. ログファイルだけでなく、クライアントにも例外を送るには、「標準エラー出力にエコー」ボックスにチェックマークをつけます。
4. コンソールを表示するときは、「コンソールを作成」ボックスにチェックマークをつけます (これは Windows 環境専用のオプションです)。
5. 「保存」ボタンを選択します。
6. サーバーを再起動します。

server.xml ファイルの編集

ログの設定を変更するには、server.xml ファイル内の log-service 要素の属性を設定します。

ログファイルだけでなく、クライアントにも例外を送ることができます。server.xml 内に次のパラメータを設定します。クライアントがブラウザの場合、例外はブラウザに表示されます。

```
<log-service ... echo-log-messages-to-stderr=true ... />
```

Windows 環境では、server.xml ファイルに次の行を追加して、コンソールを表示できます。

```
<log-service ... create-console=true ... />
```

server.xml ファイルの詳細は、『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。

プロファイル

プロファイラを使って Sun ONE Application Server 上でリモートプロファイルを実行すると、サーバー側のパフォーマンスのボトルネックを検出できます。この節では、Sun ONE Application Server で使用できるように、これらのプロファイラを設定する方法について説明します。

- HPROF プロファイラ
- Optimizeit プロファイラ
- Wily Introscope プロファイラ
- Jprobe プロファイラ

HPROF プロファイラ

HPROF は、Java 2 SDK に付属している簡単なプロファイラエージェントです。これは、JVMPI と対話して、ファイルまたはソケットに、ASCII またはバイナリ形式でプロファイル情報を書き出すダイナミックリンクライブラリです。この情報は、HAT などのプロファイラフロントエンドツールを使ってさらに処理することができます。

HPROF は、CPU の使用状況、ヒープ割り当て統計、およびモニター競合プロファイルを表示できます。さらに、Java 仮想マシン内のヒープダンプ、およびすべてのモニターとスレッドの状態もレポートします。HPROF プロファイラの詳細は、次のサイトにある JDK のマニュアルを参照してください。

http://java.sun.com/j2se/1.4/docs/guide/jvmpi/jvmpi.html#hprof

次の手順に従って HPROF がインストールされると、ライブラリがサーバープロセス内に読み込まれます。

UNIX 上で HPROF プロファイルを使うには、次の手順で行います。

1. 次の方法のいずれかで Sun ONE Application Server を設定します。
   - 管理インタフェースのサーバーインスタンスページに行き、「JVM 設定」タブを選択し、「プロファイラ」オプションを選択してから次のフィールドを編集し、「保存」を選択する
     - 「名前」: `hprof`
     - 「プロファイラを有効」: `true`
     - 「クラスパス」:(空白のまま)
     - 「ネイティブライブラリパス」:(空白のまま)
     - 「JVM オプション」: 次の各オプションを使用するには、「JVM オプション」フィールドにオプションを入力し、「追加」を選択してから、「JVM オプション」リストで該当するボックスにチェックマークをつける

       `-Xrunhprof:file=log.txt,`*options*
   - 次のように `server.xml` ファイルを編集する

```
<!-- hprof オプション -->
<profiler name="hprof" enabled="true">
   <jvm-options>
      -Xrunhprof:file=log.txt,options
   </jvm-options>
</profiler>
```

注 `-Xrs` フラグは使用しないでください。

次に、使用できる *options* の例を示します。

```
-Xrunhprof:file=log.txt,thread=y,depth=3
```

`file` オプションは、手順 6 のスタックダンプの書き出し先を決定するので重要です。

HPROF オプションの構文は次のとおりです。

```
-Xrunhprof[:help]|[:option=value,option2=value2, ...]
```

help を使うと、HPROF に渡すことができるオプションが表示されます。出力は次のとおりです。

```
Hprof usage:-Xrunhprof[:help]|[:<option>=<value>, ...]

Option Name and Value   Description             Default
---------------------   -----------             -------
heap=dump|sites|all     heap profiling          all
cpu=samples|old         CPU usage               off
format=a|b              ascii or binary output  a
file=<file>             write data to file      java.hprof
                                                (.txt for ascii)
net=<host>:<port>       send data over a socket write to file
depth=<size>            stack trace depth       4
cutoff=<value>          output cutoff point     0.0001
lineno=y|n              line number in traces?  y
thread=y|n              thread in traces?       n
doe=y|n                 dump on exit?           y
```

注 JDK 1.4 では、cpu オプションと monitor オプションは機能しません。

2. Sun ONE Application Server の起動スクリプトの行を変更する必要もあります。起動スクリプトファイルは、*instance_dir*/startserv にあります。次の行を変更します。

   ```
   PRODUCT_BIN=appservd-wdog
   ```

   これを次のように変更します。

   ```
   PRODUCT_BIN=appservd
   ```

3. 起動スクリプトを実行してサーバーを起動します。サーバーは (手順 2 で変更したとおり) フォアグラウンドで稼動しているため、コマンドプロンプトは、サーバーが停止した後にのみ戻ります。

4. 別のウィンドウまたは端末で、サーバープロセスのプロセス ID を見つけます。

   ```
   % ps -ef | grep appservd
   ```

   このコマンドは、2 つ以上の appservd プロセスをリストします。PPID (親プロセス ID) 列を見て、2 つのプロセスのうちどちらが親でどちらが子かを確認します。子のプロセス ID は PID (プロセス ID) であることに注意してください。

5. SIGQUIT シグナル (シグナル 3) を子のプロセスに送信します。

   ```
   % kill -QUIT child_PID
   ```

6. Application Server を停止するには、別のウィンドウから停止スクリプトを実行します。

   ```
   % ./stopserv
   ```

   これにより、手順 1 で `file` HPROF オプションを使って指定したファイルに HPROF スタックダンプが書き出されます。スタックダンプの使い方については、142 ページの「デバッグ用スタックトレースの生成」を参照してください。

7. Application Server を元の設定に戻すには、手順 1 と 2 の変更を元に戻します。

Optimizeit プロファイラ

Intuitive Systems 社の Optimizeit™ 4.2 は、次のサイトで購入できます。

`http://www.optimizeit.com/index.html`

次の手順に従って Optimizeit をインストールすると、そのライブラリがサーバープロセス内に読み込まれます。

Optimizeit を使用したリモートプロファイルを有効にするには、次のいずれかを実行します。

- 管理インタフェースのサーバーインスタンスページに行き、「JVM 設定」タブを選択し、「プロファイラ」オプションを選択してから次のフィールドを編集し、「保存」を選択する
  - 「名前」: `optimizeit`
  - 「プロファイラを有効」: `true`
  - 「クラスパス」: *Optimizeit_dir*`/lib/optit.jar`
  - 「ネイティブライブラリパス」: *Optimizeit_dir*`/lib`
  - 「JVM オプション」: 次の各オプションを使用するには、「JVM オプション」フィールドにオプションを入力し、「追加」を選択してから、「JVM オプション」リストで該当するボックスにチェックマークをつける
    - `-DOPTITHOME=`*Optimizeit_dir*
    - `-Xrunoii`
    - `-Xbootclasspath/a:`*Optimizeit_dir*`/lib/oibcp.jar`
- 次のように `server.xml` ファイルを編集する

```
<!-- Optimizeit options -->
<profiler name="optimizeit" classpath="Optimizeit_dir/lib/optit.jar"
        native-library-path="Optimizeit_dir/lib" enabled="true">
   <jvm-options>
```

```
            -DOPTIT_HOME=Optimizeit_dir -Xrunoii
            -Xbootclasspath/a:Optimizeit_dir/lib/oibcp.jar
        </jvm-options>
    </profiler>
```

さらに、server.policy ファイルを次のように設定する必要があります。

```
grant codeBase "file:Optimizeit_dir/lib/optit.jar" {
   permission java.security.AllPermission;
};
```

server.policy ファイルの詳細は、59 ページの「server.policy ファイル」を参照してください。

この設定でサーバーが起動したら、プロファイラを使用できます。詳細は、Optimizeit のマニュアルを参照してください。

注	設定オプションのいずれかが欠けていたり正しくない場合は、プロファイラに問題が生じ、Sun ONE Application Server の性能に影響が出ます。

Wily Introscope プロファイラ

Wily Technology 社の Introscope に関する情報は、次のサイトから入手できます。

```
http://www.wilytech.com/solutions_introscope.html
```

次の手順に従って Introscope をインストールすると、そのライブラリがサーバープロセス内に読み込まれます。

Introscope を使ったリモートプロファイルを有効にするには、server.xml ファイルを次のように編集します。

```
<!-- Introscope options. For Win2K, use ; in classpath -->
<java-config ... bytecode-preprocessors" value="S1ASAutoProbe" ... >
   <profiler name="wily" enabled="true"
          classpath="Wily_dir/ProbeBuilder.jar:Wily_dir/Agent.jar" >
   </profiler>
</java-config>
```

この設定でサーバーを起動すると、プロファイラを使用できます。詳細は、Introscope のマニュアルを参照してください。

注	設定オプションのいずれかが欠けていたり正しくない場合は、プロファイラに問題が生じ、Sun ONE Application Server の性能に影響が出ます。

Jprobe プロファイラ

Sitraka 社の JProbe™ に関する情報は、次のサイトから入手できます。

`http://www.klgroup.com/software/jprobe/`

次の手順に従って JProbe をインストールすると、そのライブラリがサーバープロセス内に読み込まれます。

JProbe を使ったリモートプロファイルを有効にするには、次の手順に従います。

1. JProbe 3.0.1.1 をインストールします。このバージョンは、JDK 1.4 をサポートしています。詳細は、JProbe のマニュアルを参照してください。
2. 次の方法のいずれかで Sun ONE Application Server を設定します。
   - 管理インタフェースでサーバーインスタンスのページを表示して「JVM 設定」タブを選択し、「Java ホーム」フィールドに JDK 1.4.0 または 1.4.0_01 へのパスを入力して「保存」を選択する

     「プロファイラ」オプションを選択して次のフィールドを編集し、「保存」を選択してからサーバーを再起動します。

     - 「名前」: `Jprobe`
     - 「プロファイラを有効」: `true`
     - 「クラスパス」:(空白のまま)
     - 「ネイティブライブラリパス」: *JProbe_dir*`/profiler`
     - 「JVM オプション」: 次の各オプションを使用するには、「JVM オプション」フィールドにオプションを入力し、「追加」を選択してから、「JVM オプション」リストで該当するボックスにチェックマークをつける

       `-Xbootclasspath/p:`*JProbe_dir*`/profiler/jpagent.jar`

       `-Xrunjprobeagent`

       `-Xnoclassgc`

   - `server.xml` ファイルを必要に応じて変更し、サーバーを再起動する

```
<java-config java-home="JDK_path" ...>
   <profiler name="jprobe" enabled="true"
          native-library-path="JProbe_dir/profiler" >
       <jvm-options>
          -Xbootclasspath/p:JProbe_dir/profiler/jpagent.jar
```

```
            -Xrunjprobeagent -Xnoclassgc
        </jvm-options>
    </profiler>
</java-config>
```

JDK_path は、JDK 1.4.0 または JDK 1.4.0_01 を指すように指定する必要があります。

注 Sun ONE Application Server に付属する JDK 1.4.0_02 では JProbe は機能しません。

注 設定オプションのいずれかが欠けていたり正しくない場合は、プロファイラに問題が生じ、Sun ONE Application Server の性能に影響が出ます。

この設定でサーバーが起動したら、プロファイラを使用できます。

3. 次の環境変数を設定します。

   `JPROBE_ARGS_0=-jp_input=`*JPL_file_path*

   JPL ファイルの作成方法についての説明は、手順 6 を参照してください。

4. サーバーインスタンスを起動します。
5. `jpprofiler` を起動して、リモートセッションに接続します。デフォルトのポートは `4444` です。
6. JProbe Launch Pad を使って JPL ファイルを作成します。必要な設定は次のとおりです。
   a. アプリケーションタイプとして「Server Side」を選択します。
   b. 「Program」タブで次の詳細情報を設定します。
      - 「Target Server」- *other_server*
      - 「Server home Directory」- *install_dir*
      - 「Server class File」- `com.iplanet.ias.server.J2EERunner`
      - 「Working Directory」- *install_dir*
      - 「Classpath」- *install_dir*`/lib/appserv-rt.jar`
      - 「Source File Path」- *source_code_dir* (行レベルの詳細を知る必要がある場合)
      - 「Server class arguments」- (オプション)

- 「Main Package」- `com.iplanet.ias.server`

「VM」タブ、「Attach」タブ、および「Coverage」タブも適切に設定する必要があります。詳細は、JProbe のマニュアルを参照してください。JPL ファイルを作成した後は、これを `JPROBE_ARGS_0` への入力として使います。

第 6 章

ライフサイクルリスナーの開発

ライフサイクルリスナーモジュールは、単体または RMI サーバーのインスタンス化など、アプリケーションサーバー環境内で短時間 (または長時間) の Java ベースタスクを実行するための手段を提供します。これらのモジュールは、サーバーの起動時に自動的に起動され、サーバーのライフサイクルの各種段階で通知を受け取ります。

次の各節では、ライフサイクルモジュールを作成および使用する方法について説明します。

- サーバーライフサイクルのイベント
- LifecycleListener インタフェース
- LifecycleEvent クラス
- サーバーライフサイクルイベントコンテキスト
- ライフサイクルモジュールのアセンブルと配備
- ライフサイクルモジュールに関する注意事項

サーバーライフサイクルのイベント

ライフサイクルモジュールは、サーバーライフサイクル内の次のイベントに応じて、タスクを待機し、実行します。

- `INIT_EVENT` では、サーバーは設定の読み込み、組み込みサブシステム (セキュリティおよびログサービスなど) の初期化、およびコンテナの作成を行う
- `STARTUP_EVENT` では、サーバーは配備アプリケーションのロードと初期化を行う
- `READY_EVENT` では、サーバーは要求を処理する準備ができた状態で待機している
- `SHUTDOWN_EVENT` では、サーバーはロードしたアプリケーションを破棄して停止する

- TERMINATION_EVENT では、サーバーはコンテナ、組み込みサブシステム、サーバーの実行時環境を終了する

これらのイベントは、LifecycleEvent クラスに定義されています。

これらのイベントからの通知を受け取るライフサイクルモジュールは、LifecycleListener インタフェースを実装し、server.xml ファイルに設定されます。

LifecycleListener インタフェース

ライフサイクルモジュールを作成するということは、com.sun.appserv.server.LifecycleListener インタフェースを実装するようカスタマイズしたクラスを設定するということです。複数のライフサイクルモジュールを作成し、同時に実行できます。

LifecycleListener インタフェースは、次のメソッドを定義します。

- public void handleEvent(com.sun.appserv.server.LifecycleEvent event) throws ServerLifecycleException

  このメソッドは、ライフサイクルイベントに応答し、エラーが発生した場合に com.sun.appserv.server.ServerLifecycleException をスローします。

LifecycleListener インタフェースの実装サンプルは、次の LifecycleListenerImpl.java ファイルです。ライフサイクルイベントのテストには、このファイルを利用できます。

```
package com.sun.appserv.server;

import java.util.Properties;

/**
 *  LifecycleListenerImpl は、LifecycleListener インタフェースのダミー実装です。
 * この実装は、さまざまなライフサイクルインタフェースメソッドを取り除きます。
 */

public class LifecycleListenerImpl implements LifecycleListener {

   /** receive a server lifecycle event
    *  @param event associated event
    *  @throws <code>ServerLifecycleException</code> for exceptional condition.
    *
    *  Configure this module as a lifecycle-module in server.xml:
    *
    *  <applications>
    *     <lifecycle-module name="test"
```

```
 *               class-name="com.sun.appserv.server.LifecycleListenerImpl"
 *               is-failure-fatal="false">
 *        <property name="foo" value="fooval"/>
 *     </lifecycle-module>
 *  </applications>
 *
 *  Set<code>is-failure-fatal</code>in server.xml to <code>true</code> for
 *  fatal conditions.
 */
public void handleEvent(LifecycleEvent event) throws ServerLifecycleException
{
   LifecycleEventContext context = event.getLifecycleEventContext();

   context.log("got event" + event.getEventType() + " event data: "
      + event.getData());

   Properties props;

   if (LifecycleEvent.INIT_EVENT == event.getEventType()) {
      context.log("LifecycleListener:INIT_EVENT");

      props = (Properties) event.getData();

      // handle INIT_EVENT
      return;
   }

   if (LifecycleEvent.STARTUP_EVENT == event.getEventType()) {
      context.log("LifecycleListener:STARTUP_EVENT");

      // handle STARTUP_EVENT
      return;
   }

   if (LifecycleEvent.READY_EVENT == event.getEventType()) {

      context.log("LifecycleListener:READY_EVENT");

      // handle READY_EVENT
      return;
   }

   if (LifecycleEvent.SHUTDOWN_EVENT== event.getEventType()) {
      context.log("LifecycleListener:SHUTDOWN_EVENT");

      // handle SHUTDOWN_EVENT
      return;
```

```
        }

        if (LifecycleEvent.TERMINATION_EVENT == event.getEventType()) {

            context.log("LifecycleListener:TERMINATE_EVENT");

            // handle TERMINATION_EVENT
            return;
        }
    }
}
```

LifecycleEvent クラス

サーバーライフサイクルのイベントは、com.sun.appserv.server.LifecycleEvent クラスに定義されます。イベントには次のメソッドが関連付けられています。

- public java.lang.Object getData()

  このメソッドは、イベントに関連付けられたデータを返す

- public int getEventType()

  このメソッドは、INIT_EVENT、STARTUP_EVENT、READY_EVENT、SHUTDOWN_EVENT、または TERMINATION_EVENT のいずれかのイベント型を返す

- public com.sun.appserv.server.LifecycleEventContext getLifecycleEventContext()

  このメソッドは、次に説明するライフサイクルイベントコンテキストを返す

LifecycleEvent インスタンスは、LifecycleListener.handleEvent メソッドに渡されます。

サーバーライフサイクルイベントコンテキスト

com.sun.appserv.server.LifecycleEventContext インタフェースは、サーバーに関する実行時の情報を表示します。ライフサイクルイベントコンテキストは、サーバーの初期化時に LifecycleEvent クラスがインスタンス化されるときに作成されます。LifecycleEventContext インタフェースは、次のメソッドを定義します。

- public java.lang.String[] getCmdLineArgs()

  このメソッドは、サーバー起動時のコマンド行引数を返す

- public java.lang.String getInstallRoot()

  このメソッドは、サーバーのインストールルートディレクトリを返す

- public java.lang.String getInstanceName()

  このメソッドは、サーバーのインスタンス名を返す

- public javax.naming.InitialContext getInitialContext()

  このメソッドは、初期 JNDI 命名コンテキストを返す。ライフサイクルモジュールの命名環境は、STARTUP_EVENT の間にインストールされる。ライフサイクルモジュールは、STARTUP_EVENT が完了した後、jndi-name 属性を使って server.xml ファイルに定義されたリソースを検索できる

注 JNDI で他の企業リソースと名前が重複することを避けるため、また、移植性の問題を回避するために、Sun ONE Application Server ライフサイクルモジュールのすべての名前は、java:comp/env という文字列から始める必要があります。

  ライフサイクルモジュールが Beans を検索する必要がある場合は、READY_EVENT 内で実行できる。また、getInitialContext() メソッドを使って、すべてのリソースが結合した初期コンテキストを取得することができる

- public void log(java.lang.String message)

  このメソッドは、サーバーログファイルに、指定されたメッセージを書き込む。message パラメータは、ログファイルに書き込むテキストを示す文字列である

- public void log(java.lang.String message, java.lang.Throwable throwable)

  このメソッドは、サーバーログファイルに、Throwable 例外の説明的なメッセージおよびスタックトレースを書き込む。message パラメータは、エラーまたは例外を説明する文字列である。throwable パラメータは、Throwable エラーまたは例外を示す

ライフサイクルモジュールのアセンブルと配備

92 ページの「ライフサイクルモジュールのアセンブル」で説明されているとおりに、ライフサイクルモジュールをアセンブルします。また、103 ページの「ライフサイクルモジュールの配備」で説明されているとおりに、ライフサイクルモジュールを配備します。

ライフサイクルモジュールを配備時に、`server.xml` ファイル内に `lifecycle-module` 要素が作成されます。このファイルを編集して、設定を変更できます。`property` サブ要素を使って、入力パラメータを指定できます。次に例を示します。

```
<lifecycle-module   name="customStartup"
                    enabled="true"
                    class-name="com.acme.CustomStartup"
                    classpath="/apps/customStartup"
                    load-order="200"
                    is-failure-fatal="true">
   <description>custom startup module to do my tasks</description>
   <property name="rmiServer" value="acme1:7070" />
   <property name="timeout" value="30" />
</lifecycle-module>
```

`is-failure-fatal` が `true` (デフォルトでは、`false`) に設定されている場合、ライフサイクルモジュールにエラーが発生すると、サーバーの初期化または起動は行われません。ただし、シャットダウンまたは強制終了は行われます。

`server.xml` ファイルの詳細は、『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を配備したら、サーバーを再起動してモジュールを有効にする必要があります。サーバーは設定をインスタンス化し、それをサーバー初期化時にライフサイクルイベントリスナーとして登録します。

ライフサイクルモジュールに関する注意事項

初期化中または起動中に割り当てられたリソースは、シャットダウン中または終了中に解放される必要があります。ライフサイクルモジュールのクラスは、メインサーバーのスレッドから同時に呼び出されるため、これらのクラスがサーバーをブロックしないようにすることが重要です。ライフサイクルモジュールは、必要に応じてスレッドを作成することがありますが、これらのスレッドはシャットダウンおよび終了時に停止される必要があります。

ライフサイクルモジュールクラスローダは、ライフサイクルモジュールの親クラスローダです。`server.xml` 内の各ライフサイクルモジュールの `classpath` は、クラスローダーの構築に使用されます。ライフサイクルモジュールが必要とするすべてのサポートクラスは、LifeCycleModule クラスローダーまたはその親である共有クラスローダーで使用可能である必要があります。(共有クラスローダーは、サーバー全体のリソースも同様に読み込みます。

`server.policy` ファイルが正しく設定されているかどうか、また、`System.exec()` を実行しようとしているライフサイクルモジュールがセキュリティアクセス違反を引き起こす可能性がないことを必ず確認してください。詳細は、59 ページの「server.policy ファイル」を参照してください。

ライフサイクルモジュールに設定したプロパティは、プロパティとして `INIT_EVENT` に渡されます。JNDI 命名コンテキストは、`INIT_EVENT` では使用できません。ライフサイクルモジュールが命名コンテキストを必要とする場合、`STARTUP_EVENT`、`READY_EVENT`、または `SHUTDOWN_EVENT` 内で取得することができます。

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 管理によるトランザクション (bean-managed transaction)　エンタープライズ Bean が、開発者が記述したプログラムで制御されるトランザクション境界設定。「コンテナ管理によるトランザクション (container-managed transaction)」も参照。

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。

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　`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。完全な製品用に必要な管理機能と制御機能を追加する。

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 Application Server RowSet　Sun ONE Application Server の拡張機能を実装する `RowSet` オブジェクト。

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。JMS プロバイダでもある。

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。

アクティベーション (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。

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 (EJB) を管理する状態。

外部 JDNI リソース (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。同じまたは異なるホスト名、ポート番号、または IP アドレスを使って、複数の仮想サーバーがコンテンツを処理できる。HTTP サービスは、受け取った Web 要求を URL に基づいて別の仮想サーバーに送ることができる。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。

カプセル化 (encapsulate) モジュールの知識をローカライズすること。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して表示することができる。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書式の集り。「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、1 つのシステムにインストールされた共通バイナリファイルセットから作成されるインスタンスの集り。

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っ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　サーブレット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 (EJB) 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。

持続性マネージャ (persistance 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。

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。

電子署名 (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。

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。

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。

トランザクション分離レベル (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。

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。

発見的決定 (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。

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」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である。

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) 「トランザクション分離レベル (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集まり。たとえば、配列のデータ単位や、論理要素。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

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

あ

い

え

か

き

く

こ

さ

し

す

せ

そ

た

つ

て

と

に

ね

ら

り

れ

ろ