パフォーマンスチューニングガイド

Sun ONE™ Application Server

Version 7

816-6485-10
2002 年 12 月

目 次

本書について

概要

Sun ONE Application Server 7 は J2EE 1.3 仕様に準拠したアプリケーションサーバーで、Java Web サービス標準や、標準の HTTP サーバープログラミング機能に対応しています。本稼動環境と開発環境の両方に幅広く対応するため、次の 3 種類のアプリケーションサーバーが用意されています。

- Platform Edition
- Standard Edition
- Enterprise Edition

Platform Edition

Platform Edition は、Sun ONE Application Server 7 製品ラインの中核を成す製品です。J2EE 1.3 仕様に準拠した高性能かつ小型のランタイム環境を提供し、基本的な配置作業や、サードパーティアプリケーションの埋め込みに適しています。また、運用用途に限り、無料で利用できます。Platform Edition には Sun ONE Web Server、Sun ONE Message Queue、J2EE リファレンス実装で培われたテクノロジが搭載されており、Web サービスに直ちに利用できます。

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

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

Standard Edition

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

Enterprise Edition

Enterprise Edition では、最も要求の厳しい J2EE ベースアプリケーションにも適応できるようにコアアプリケーションサーバープラットフォームの可用性を向上、ロードバランス機能、およびクラスタリング機能をさらに高めています。Standard Edition より高度な管理機能により、複数のインスタンスの配備、複数のマシンへの配備にも対応しています。

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

製品の詳細については、
http://wwws.sun.com/software/products/appsrvr/home_appsrvr.html にある Sun ONE Application Server のページを参照してください。

マニュアルの使用法

Sun ONE Application Server のマニュアルは、PDF 形式または HTML 形式でも入手できます。

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

次の表は、Sun ONE Application Server のマニュアルに記述されているタスクと概念を示しています。左の列はタスクと概念、右の列は対応するマニュアル名を示します。

表 : Sun ONE Application Server マニュアルの概要

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

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

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

このマニュアルは、次の表記規則に従っています。

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

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

  server はアプリケーションを実行するサーバー名、domain はユーザーのインターネットドメイン名、path はサーバー上のディレクトリの構造、file は個別のファイル名を示します。URL の斜体文字の部分は可変部分です。
- **フォント**は、次のように使い分けます。
  - モノスペースフォントは、サンプルコード、コードの一覧表示、API および言語要素 (関数名、クラス名など)、ファイル名、パス名、ディレクトリ名、および HTML タグに使います。
  - 斜体文字はコード変数に使います。
  - 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、段落の先頭またはリテラルに使われる文字の強調に使います。
- このマニュアルでは、ほとんどのプラットフォームの**インストールルートディレクトリ**を *install_dir* と記述します。例外については、12 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。

  配布される各種製品のデフォルトの *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バージョンインストール :

    ユーザーのホームディレクトリ `/usn/appserver7`
  - Solaris にバンドルされていない非評価バージョンインストール :

    `/opt/SUNWappserver7`

- Windows のインストール :

  ```
  C:¥Sun¥AppServer7
  ```

  上記のプラットフォームで *default_config_dir* および *install_config_dir* は、*install_dir* と同義です。これ以外の説明と例外については、12 ページの「ディレクトリ名の表記規則」を参照してください。

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

  default_config_dir/domains/*domain*/*instance*

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

ディレクトリ名の表記規則

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

- **Solaris 9 バンドル版のインストール**では、デフォルトのインストールディレクトリは次のように表記されます。
  - *install_dir* は /usr/appserver/ を示します。このディレクトリにはインストールイメージの静的な要素が保存されます。ユーティリティ、実行可能ファイル、およびアプリケーションサーバーを構成するライブラリは、すべてここに保存されます。
  - *default_config_dir* は /var/appserver/domains を示します。このディレクトリは、作成したドメインのデフォルトの保存場所です。
  - *install_config_dir* は /etc/appserver/config を示します。このディレクトリには、ライセンスなどのインストール全体に適用される設定情報や、このインストール用に設定した管理ドメインのマスターリストが保存されます。
- **Solaris 8 および 9 パッケージベースのアンバンドルのインストール (評価バージョン以外)** では、デフォルトのインストールディレクトリは次のように表記されます。
  - *install_dir* は /opt/SUNWappserver7 を示します。このディレクトリにはインストールイメージの静的な要素が保存されます。ユーティリティ、実行可能ファイル、およびアプリケーションサーバーを構成するライブラリは、すべてここに保存されます。

- *default_config_dir* は `/var/opt/SUNWappserver7/domains` を示します。このディレクトリは、作成したドメインのデフォルトの保存場所です。
- *install_config_dir* は `/etc/opt/SUNWappserver7/config` を示します。このディレクトリには、ライセンスなどのインストール全体に適用される設定情報や、このインストール用に設定した管理ドメインのマスターリストが保存されます。

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。

製品サポート

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

- 次のオンラインサポート Web サイトをご利用ください。

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

- 保守契約を結んでいるお客様の場合は、専用ダイヤルをご利用ください。

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

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

このマニュアルは、Sun ONE Application Server の上級管理者を対象としています。ここでは、Sun ONE Application Server のパフォーマンスと信頼性を最大化するためのチューニングの方法が示されます。Sun ONE Application Server の設定を変更する前に、設定ファイルのバックアップをとっておくことをお勧めします。

マニュアルの内容

このマニュアルでは、Sun ONE Application Server に含まれるサブシステム、機能、ツールについて説明し、パフォーマンスと信頼性を最大化するために、これをチューニングする方法を紹介します。このマニュアルは、サーバー管理者、J2EE 開発者、ネットワーク管理者、評価担当者を対象としています。

前提事項

このマニュアルでは、次の項目について熟知していることを前提とします。

- インターネットおよび WWW (World Wide Web)
- Java プログラミング
- J2EE アプリケーションモデル
- アプリケーションサーバー
- Solaris™ または Windows NT/2000 オペレーティングシステム

マニュアルの構成

このマニュアルの構成は次のとおりです。

「Sun ONE Application Server について」では、Sun ONE Application Server の機能とコンポーネントについて概要を示します。

「Sun ONE Application Server のパフォーマンスについて」では、Sun ONE Application Server のチューニングに関連する技法とプロセスについて説明します。

「アプリケーションのチューニング」では、パフォーマンスを最大化するために、アプリケーションで利用できる設定と設定方法について説明します。

「Sun ONE Application Server のチューニング」では、必要に合わせてアプリケーションサーバーを設定する方法について説明します。

「Java 実行システムのチューニング」では、Sun ONE Application Server のパフォーマンスを最適化するために Java 仮想マシンを設定する方法について説明します。

「オペレーティングシステムのチューニング」では、Sun ONE Application Server のパフォーマンスを最適化するためにオペレーティングシステムを設定する方法について説明します。

「パフォーマンスに関する一般的な問題」では、Sun ONE Application Server を従来の Web サーバーとして利用する場合に生じるパフォーマンス上の問題について説明します。

パフォーマンスに関する主要事項を簡単に調べられるように、索引が用意されています。

第 1 章

Sun ONE Application Server について

Sun ONE Application Server は、信頼性が高くスケーラブルな Web サービス配備プラットフォームを提供します。アプリケーションプログラマーは、本稼働レベルの配備を Sun ONE Application Server が提供するサービスに任せ、優れたソフトウェアコンポーネントを使ってビジネスロジックの実装に専念することができます。

この章では、次の項目について説明します。

- サーバーコンポーネント
- サーバーのアーキテクチャ
- サーバーツール

サーバーコンポーネント

Sun ONE Application Server には、相互の対話が必要なさまざまなコンポーネントが用意されています。これらのコンポーネントは、本稼働環境と開発環境の両方で最適なパフォーマンスにチューニングできます。

この節では、次の項目について説明します。

- アプリケーションサーバーのプロセス
- データベース

アプリケーションサーバーのプロセス

Sun ONE Application Server は、appservd という 1 つのプロセス内で実行されます。

Sun ONE Application Server のプロセスを呼び出すときは、startserv スクリプトを実行します。このスクリプトは、server.xml に記録されている設定情報を読み込み、サーバーインスタンスを起動して、次の処理を行います。

- 内蔵 ORB の起動
- 内蔵 HTTP サーバーの起動
- 配備されているアプリケーションとコンポーネントの各対応コンテナへのロード
- ロギングサービスの開始

データベース

Sun ONE Application Server は、Oracle、Sybase、Informix、DB2 などのデータベースをサポートしています。また、Sun ONE Application Server では、データベース用のサードパーティ製 JDBC ドライバも設定できます。データソースと接続プールの設定については、『Sun ONE Application Server 管理者ガイド』を参照してください。

サーバーのアーキテクチャ

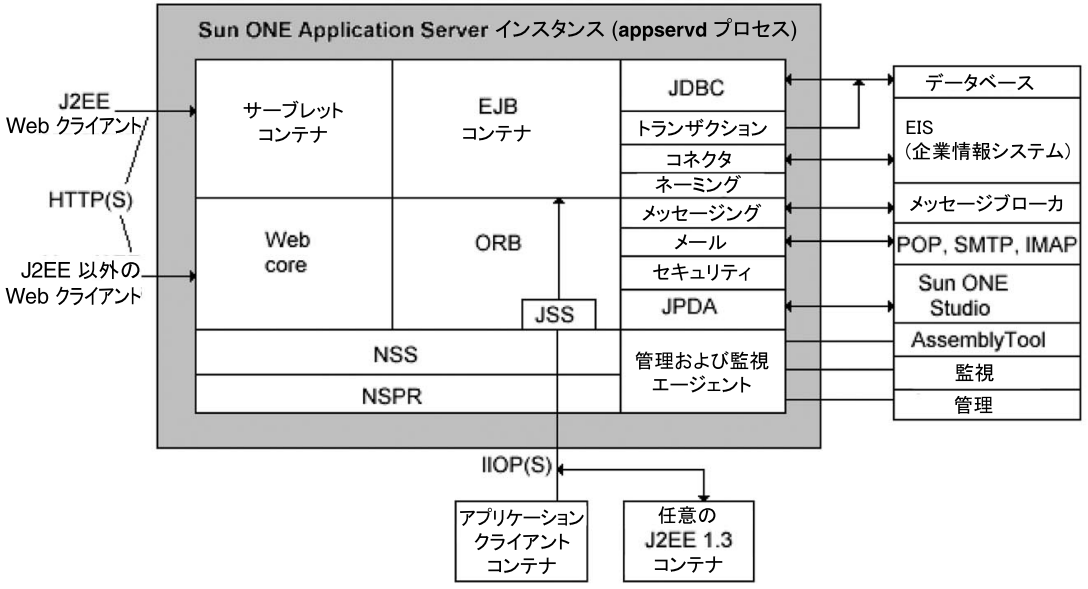


図 : Sun ONE Application Server コンポーネントのアーキテクチャ

Sun ONE Application Server は、高度な Web 要求の処理、セキュリティ、管理の基盤を提供します。上の図は、アプリケーションサーバーの主要コンポーネントを示しています。

アプリケーションサーバーインスタンスは、配備するアプリケーションサーバーの基本となります。各アプリケーションサーバーインスタンスには、J2EE 1.3 の Web コンテナと EJB コンテナが含まれています。定評のある高性能 HTTP サーバーは Web コンテナの前段に配備され、EJB コンテナは内蔵の ORB に支えられています。バックエンドシステムにアクセスできるので、アプリケーションは J2EE コネクタアーキテクチャとサードパティのリソースアダプタ、JMS と内蔵の JMS プロバイダーまたはサードパーティプロバイダ、および一般的なサードパーティ JDBC ドライバの自由な組み合わせを利用できます。分散トランザクションの範囲内であれば、バックエンドシステムへのアクセスは、すべてが Java で記述された内蔵のトランザクションマネージャを使って管理できます。

管理サーバーには、コア管理アプリケーションと SNMP エージェントが格納されています。リモート管理は、すべて管理サーバー経由で行われます。コマンド行ベースと Web ブラウザベースの管理クライアントは、どちらも HTTP または HTTPS を介して管理サーバーに直接アクセスします。HTTPS を使った場合はセキュアなアクセスが可能です。

1 つ以上のファイアウォール層によって保護された非武装地帯 (DMZ) には、1 つまたは複数の Web サーバーが置かれています。Web サーバープロキシプラグインを使って、この背後にアプリケーションサーバーを配備できます。プラグインは、フロントエンドの Web サーバーが、インターネットから受信した HTTP または HTTPS トラフィックを、バックエンドのアプリケーションサーバー層に置かれている 1 つまたは複数のアプリケーションサーバーに配信しするときに使われます。

さまざまなクライアントアプリケーションが、アプリケーションサーバーに配備されたビジネスサービスにアクセスできます。Web サービスとブラウザベースのクライアントは、HTTP または HTTPS のいずれかを使って Java Web サービスと J2EE Web アプリケーションにアクセスできます。Java アプリケーションクライアントは、スタンドアロンモードで配備するか、または標準のアプリケーションクライアントコンテナ内に配備できます。このクライアントは、Java RMI-IIOP テクノロジ (Java Remote Method Invocation over Internet Inter-ORB Protocol Technology) を使って、アプリケーションサーバーに配備された EJB にアクセスできます。C++ 言語クライアントは、Java IDL/IIOP を使って EJB にアクセスできます。

Sun ONE Application Server の基本コンポーネントは、ウォッチドッグプロセスによって管理される `appservd` プロセスです。アプリケーションコードは、`appservd` プロセスが作成するマルチスレッドプロセスで実行されます。Java 仮想マシン (JVM) も `appservd` 内で起動されます。

システムは、管理サーバーを使って管理されます。また、別のシステムにインストールされている Web サーバーをプロキシとして使い、要求をアプリケーションサーバーインスタンスに転送するように設定する機能も用意されています。

サーバーツール

Sun ONE Application Server には、管理ツールと、アプリケーションをアプリケーションサーバーに配備するためのツールが用意されています。管理インタフェースは、Web ベースのツールです。また、コマンド行インタフェース (CLI) を使って、コマンド行からサーバーを管理することもできます。Web ベースと CLI ベースのどちらのツールでも、すべての管理機能を利用できます。

Sun ONE Application Server と共に Sun ONE Studio 4 を設定した場合は、アプリケーションのアセンブリ機能と配備機能を利用できます。

サーバー管理インタフェース

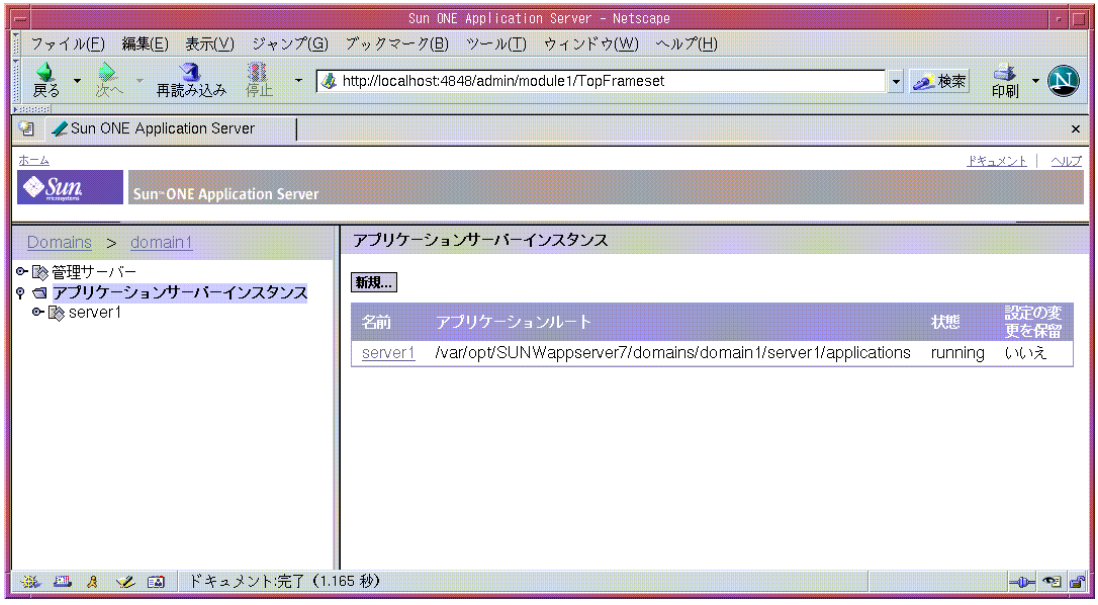


図 : Sun ONE Application Server の管理インタフェース

上の図に示す Sun ONE Application Server の管理インタフェースは、管理サーバーインスタンスに配備された Web アプリケーションとして実行され、ブラウザを使ってどこからでもアクセスできます。ユーザーは、これを使ってアプリケーションサーバーインスタンスを設定します。

次の図に示すように、コマンド行インタフェース (CLI) から管理サーバーにアクセスすることもできます。CLI では、Web ベースインタフェースの代わりにコマンド行を使ってアプリケーションサーバーインスタンスを設定できます。

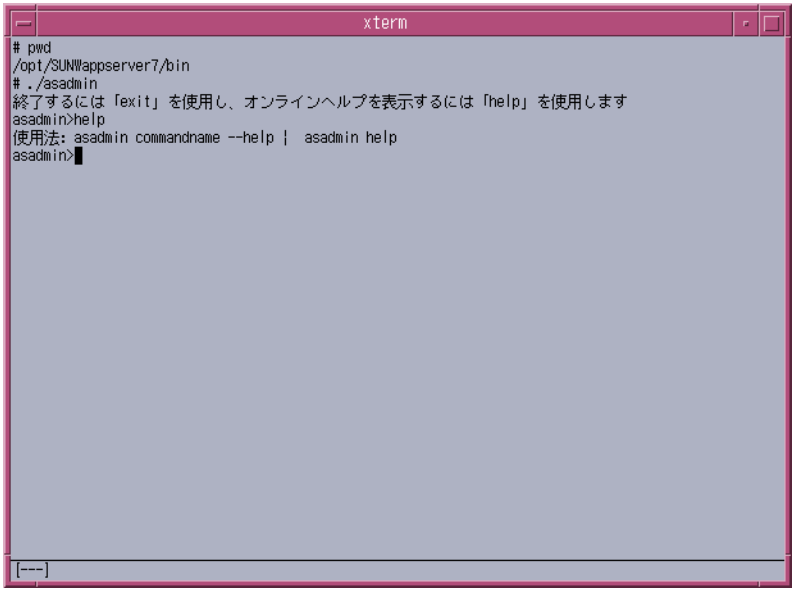


図 : Sun ONE Application Server のコマンド行インタフェース

このマニュアルでは、パフォーマンスの向上のために、Sun ONE Application Server の管理インタフェースを使ってサーバープロセスとアプリケーション配備記述子をチューニングする方法について説明します。管理インタフェースの詳細については、『Sun ONE Application Server 7 管理者ガイド』を参照してください。

Sun ONE Studio 4

Sun ONE Studio 4 統合開発環境 (IDE) は、Sun ONE Application Server に配備する J2EE (Java 2 Enterprise Edition) の開発、アセンブル、配備に使用します。Sun ONE Studio では、J2EE アプリケーションコンポーネントは、配備時の受取側となるコンテナに基づいてモジュール化されます。これらの J2EE コンポーネントは、EJB JAR モジュール (拡張子は .JAR) または Web アプリケーションモジュール (拡張子は .WAR) としてアーカイブできます。各モジュールには、J2EE 記述子と、XML ファイルとして保存された Sun ONE Application Server 固有の配備記述子が含まれます。詳細については、『Sun ONE Studio 4 開発者ガイド』を参照してください。

Sun ONE Studio と Sun ONE Application Server を併用することで、IDE とアプリケーションサーバーをシームレスに統合する数多くの機能を利用できるようになります。

表 : Sun ONE Studio と Sun ONE Application Server のシームレスな統合

Studio の機能	Sun ONE Application Server による拡張
CMP マッピング	開発者は、データベーステーブルを検索して関連するテーブルを選び、コンテナ管理による持続性 (CMP) EJB を自動的に生成できます。
サーバーランタイムの制御	開発者は、簡単な手順でローカルとリモートのアプリケーションサーバーを登録し、アプリケーションサーバーインスタンスを起動または停止できます。
リソースの設定	開発者は、アプリケーションを配備する前に、登録済みアプリケーションサーバーに J2EE リソースを登録できます。Sun ONE Studio では、JDBC リソースと接続プール、JMS リソース、その他の各種リソースを設定できます。
アプリケーションの配備	開発者は、登録済みアプリケーションサーバーのリストから項目を選択して、Sun ONE Application Server 7 の動的 (ホット) 配備および再配備機能を利用できます。
デバッグとログの表示	ローカルとリモートのアプリケーションサーバーインスタンスに配備されたアプリケーションは、簡単にデバッグできます。アプリケーションサーバーを手動で構成する必要はありません。開発者は、アプリケーションをデバッグしている間、Sun ONE Studio でサーバーイベントログファイルを表示することもできます。

第 2 章

Sun ONE Application Server のパフォーマンスについて

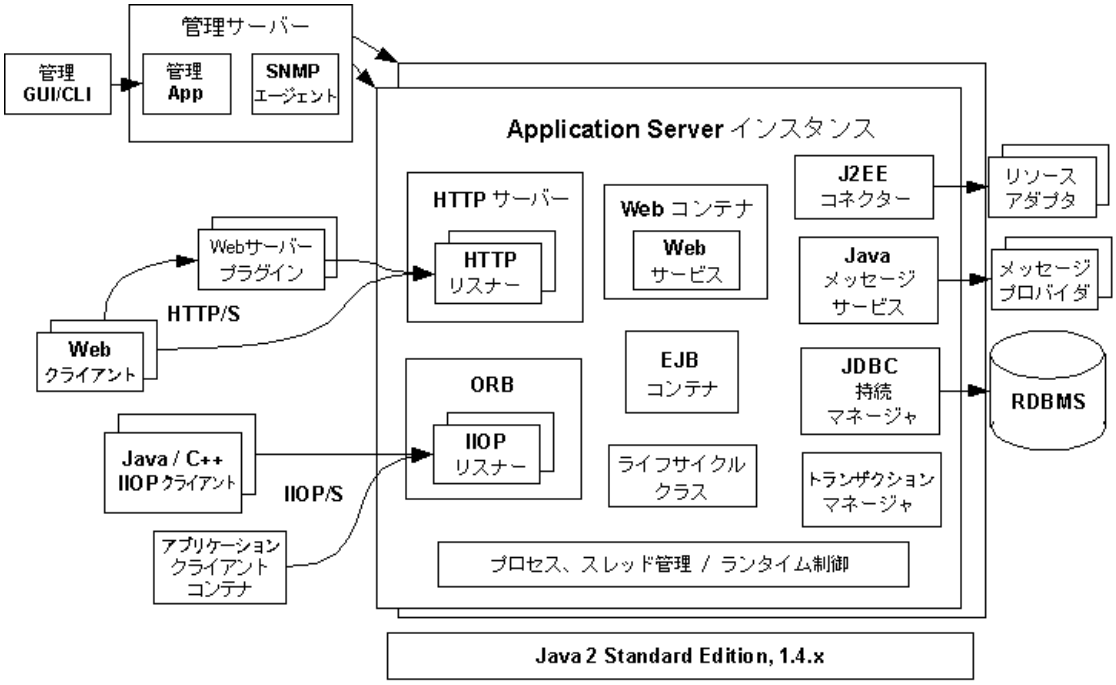
この章では、次の項目について説明します。

- アプリケーションサーバーをチューニングする目的
- 動作要件について
- 処理能力の計画
- パフォーマンスチューニングの流れ
- 設定ファイル

アプリケーションサーバーをチューニングする目的

配備記述子の設定をいくつか調整したり、サーバー設定ファイルに変更を加えるだけでパフォーマンスは大きく向上します。しかし、環境とパフォーマンスの最終的な目標を把握することが重要です。本稼働環境で最適な設定が必ずしも開発環境に適しているとは限りません。このマニュアルでは、Sun ONE Application Server のパフォーマンスを最大限に引き出せるように、用意されている調節オプションとサイズ設定のオプションについて説明します。

次の図は、Sun ONE Application Server のプロセスアーキテクチャを示しています。



図：シングルドメインの Sun ONE Application Server プロセスアーキテクチャ

動作要件について

Sun ONE Application Server にアプリケーションを配備し、チューニングを始める前に、動作環境を明確にしておくことが重要です。動作環境は、次のような高レベルの制約と要件から決定されます。

- アプリケーションアーキテクチャ
- セキュリティ要件
- アプリケーションの用途
- ハードウェアリソース
- 管理

アプリケーションアーキテクチャ

次の図に示すように、J2EE アプリケーションモデルはとても柔軟性があり、アプリケーション開発者はロジックを機能別に複数の層に分割できます。プレゼンテーション層は、通常はサーブレットと JSP を使って実装され、Web コンテナ内で実行されます。

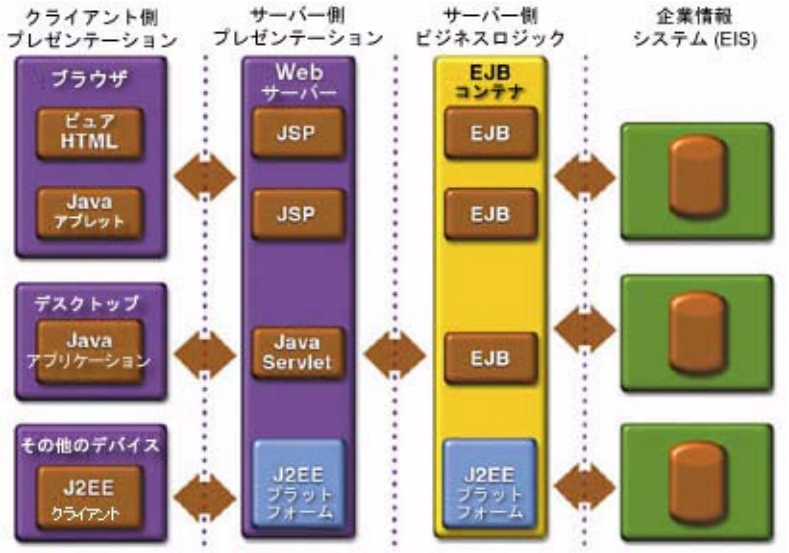


図 : J2EE アプリケーションモデル

ある程度の複雑さが必要となるエンタープライズアプリケーション全体をサーブレットと JSP だけで開発することは一般的ではありません。より複雑なビジネスアプリケーションの多くは、EJB を使って実装されます。Sun ONE Application Server は、Web コンテナと EJB コンテナを 1 つのプロセスに統合します。サーブレットから EJB へのローカルアクセスはとても効率的です。しかし、アプリケーションの配備によっては、EJB を別のプロセスで実行したり、スタンドアロンのアプリケーションやサーブレットから EJB にアクセスすることが必要です。アプリケーションアーキテクチャに基づけば、サーバー管理者は Sun ONE Application Server を複数の層に分割したり、プレゼンテーションとビジネスロジックを 1 つの層にまとめることができます。

新しいアプリケーションサーバーの配備を設計したり、既存のアプリケーションサーバーの配備に新しいビジネスアプリケーションを配備するときは、サーバー管理者は事前にアプリケーションアーキテクチャを理解することが重要です。

セキュリティ要件

ビジネスアプリケーションのほとんどはセキュリティを必要とします。ここでは、セキュリティに関するさまざまな考慮事項や、利用できる選択肢について説明します。

ユーザーの認証と承認

アプリケーションのユーザーを認証する必要があります。Sun ONE Application Server には、3 種類のユーザー認証方法が用意されています。

新しいアプリケーションを開発、テストする開発環境には、デフォルトのファイルベースのセキュリティレルムが適しています。開発時に、サーバー管理者は LDAP レルムと UNIX レルムのいずれかを選択できます。

LDAP は、Lightweight Directory Access Protocol の略です。大企業の多くは、従業員プロフィールや顧客プロフィールの管理に LDAP ベースのディレクトリサーバーを使っています。

ディレクトリサーバーを使っていない中小企業にとっても、Solaris のセキュリティインフラストラクチャの活用は魅力的です。

さまざまなセキュリティレルムを統合する方法については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

選択した認証メカニズムによっては、配備に追加ハードウェアが必要となります。一般に、ディレクトリサーバーは別のサーバーで実行され、レプリケーションや可用性の確保に備える場合はバックアップも必要になります。配備、サイズ設定、可用性のガイドラインについては、Sun ONE Directory Server のマニュアルを参照してください。

認証されたユーザーによる、アプリケーションのさまざまな機能へのアクセスには、さらに承認チェックが必要です。アプリケーションがロールベースの J2EE 承認チェックを採用した場合は、アプリケーションサーバーによって追加チェックも行われます。これによってオーバーヘッドも増えるため、処理能力を計画するときには注意が必要です。

暗号化

セキュリティ上の理由から、機密性のあるユーザー入力やアプリケーション出力の転送は、暗号化した上で行う必要があります。ビジネスに関連する Web アプリケーションのほとんどは、ブラウザとアプリケーションサーバーの間の通信内容の一部またはすべてを必要に応じて暗号化しています。オンラインショッピングのアプリケーションでは、通常のトラフィックは暗号化されていませんが、ユーザーが購入を完了する場合や、個人データを入力する場合は暗号化されます。ニュースなどのポータルアプリケーションでは、通常は暗号化は行われません。インターネットで最も一般的なセキュリティフレームワークは SSL で、多くのブラウザとアプリケーションサーバーがこれに対応しています。

Sun ONE Application Server は SSL 2.0 と 3.0 に対応し、さまざまな暗号方式をサポートするソフトウェアも用意されています。また、より高性能なハードウェア暗号化カードの統合にも対応しています。セキュリティに関する考慮事項、特に暗号化ソフトウェアの統合は、ハードウェアのサイズ設定と処理能力の計画に影響します。配備するアプリケーションの暗号化について検討する場合、管理者は次の点に注意する必要があります。

- セキュリティを考慮した場合、アプリケーションの性質はどのようなものか。アプリケーションの入出力のすべてを暗号化するか、一部だけを暗号化するか。セキュアな転送が必要な情報は、全体の何パーセント程度か。
- インターネットに直接接続されたアプリケーションサーバーにアプリケーションを配備するのか。アプリケーションサーバー層とバックエンドエンタープライズシステムとは別に、DMZ (非武装ゾーン) に Web サーバーは置かれているか。高度なセキュリティが必要な配備では、DMZ を使った配備をお勧めします。また、アプリケーションに静的なテキストや画像が大量に含まれ (ほとんどの場合は暗号化の必要がなく、DMZ に配備した Web サーバーからの配信が可能)、比較的小規模なビジネスロジックを強力なファイアウォールの奥のアプリケーションサーバーで実行する場合にも便利です。一般的な Web サーバー が Sun ONE Application Server を統合できるように、Sun ONE Application Server にはセキュアなリバースプロキシプラグインが用意されています。Sun ONE Application Server は本格的な Web サーバーでもあるので、DMZ 内で Web サーバーとして利用することもできます。
- DMZ 内の Web サーバーと次の層で稼働するアプリケーションサーバーの間で暗号化は必要か。Sun ONE Application Server のリバースプロキシプラグインは、Web サーバー層とアプリケーションサーバー層の間の SSL 暗号化をサポートしています。これを有効にする場合、管理者は暗号化のポリシーとメカニズムを考慮してハードウェアの処理能力を計画する必要があります。

- ソフトウェアによる暗号化を利用する場合、システムの各層で予想されるセキュリティ要件関連のパフォーマンスオーバーヘッドは何か。
- ハードウェアによる暗号化を利用する場合、各製品のパフォーマンスとスループットの関係はどのようなものか。

注 Web サーバーと Sun ONE Application Server の間の通信を暗号化する方法については、『Sun ONE Application Server 管理者ガイド』を参照してください。

アプリケーションの用途

アプリケーションのユーザーは、誰もがアプリケーションのパフォーマンスについて何らかの期待を持っています。これは、通常は数値化が可能です。サーバー管理者はこの期待を明確に把握し、完成したアプリケーションが顧客のニーズを満たせるように処理能力を計画する必要があります。

パフォーマンスについては、次の点に注意する必要があります。

- アプリケーションとの間でさまざまなやりとりを行うエンドユーザーが必要とする平均的な応答時間はどの程度か。最も頻繁に行われるやりとりは何か。応答時間の重要性が極端に高いやりとりがあるか。思考時間も含め、各トランザクションの長さはどの程度か。多くの場合、役に立つ予測を立てるには、ユーザーによる実際の操作テストを行う必要があります。
- 安定時とピーク時のユーザー負荷はどの程度か。負荷がピークとなる特定の時間帯、曜日、月はあるか。オンラインビジネスでは数百万人の登録顧客も考えられますが、ある一定の時間にログインし、ビジネストランザクションを行うのは、通常はそのごく一部に限られます。処理能力の計画で起きやすい間違いは、並行してアクセスする顧客の平均人数とピーク時の人数を基準とせずに、顧客の総数を基準とすることです。並行アクセスするユーザーの数は、時間帯によっても異なることがあります。
- 1 つの要求で転送されるデータの平均量と最大量はどの程度か。これもアプリケーションによって大きく異なります。コンテンツのサイズを正しく見積もることは、その他の用途パターンと合わせて、管理者によるネットワーク容量の計画に役立ちます。
- 今後 12 か月に予想されるユーザー負荷の成長はどの程度か。将来に備えた計画により、危機的な状況や、アップグレードのためのダウン時間を回避できます。

ハードウェアリソース

管理者が自由に使えるハードウェアリソースの種類と数は、パフォーマンスのチューニングとサイトの計画に大きく影響します。

Sun ONE Application Server は、優れた垂直スケーラビリティを提供します。1 つのアプリケーションサーバープロセスで、最大で 12 の高性能 CPU を利用できます。アプリケーションサーバーインスタンスの数は、少ないほど管理の手間とコストが少なくなります。また、少数のアプリケーションサーバーに複数の関連アプリケーションを配備することで、データのローカリティの向上や、アプリケーション間でのキャッシュデータの再利用が見込めるため、パフォーマンスの向上を期待できます。このようなサーバーでは負荷も増大するため、大容量のメモリ、ディスクスペース、ネットワーク容量も必要となります。

Sun ONE Application Server は、小規模なハードウェア製品のグループに配備することもできます。ビジネスアプリケーションは、さまざまなサーバーインスタンスに区切ることができます。1 台または複数の外部ロードバランサーを導入することで、ユーザーからのアクセスは、効率的にすべてのアプリケーションサーバーインスタンスに分散されます。水平スケーリングのアプローチは、ハードウェアコストを抑えると同時に、可用性も向上させることができるため、このアプローチが適するアプリケーションもあります。ただし、管理が必要なアプリケーションサーバーインスタンスとハードウェアノードの数も多くなります。

管理

サーバーにインストールした 1 つの Sun ONE Application Server を使って、複数のインスタンスを作成できます。1 つの管理サーバーは 1 つまたは複数のインスタンスを管理でき、この管理サーバーと管理対象インスタンスのグループを「ドメイン」と呼びます。複数の管理ドメインを作成することで、アプリケーションサーバーインスタンスの複数グループを異なる担当者が個別に管理できます。

開発環境の開発者用に、1 つのインスタンスだけのドメインを作成し、これを独立した作業領域として使うこともできます。この場合、各開発者はその他のアプリケーションサーバードメインに影響することなく、専用のアプリケーションサーバーを管理できます。小規模な開発チームであれば、共同開発用に複数のインスタンスを持つ共用の管理ドメインも作成できます。

開発環境では、サーバー管理者はアプリケーションとビジネス機能に基づいて管理ドメインを作成できます。たとえば、社内用の人材アプリケーションを 1 つの管理ドメインの 1 つまたは複数のサーバーでホストし、外部の顧客用アプリケーションを複数のハードウェアにまたがる複数の管理ドメインでホストすることができます。

Sun ONE Application Server は、Web アプリケーションの仮想サーバー機能をサポートしています。Web アプリケーションをホストするサービスプロバイダでは、管理を容易にするために、1 つの Sun ONE Application Server で複数の URL ドメインをホストしたいと考えるかもしれません。サーバー管理者は、この機能の必要性を検討する必要があります。

この時点でサーバー管理者は、すべてのアプリケーション、パフォーマンス特性の概要、セキュリティ要件を把握し、かなり具体的な配備環境のイメージを持っているはずです。次に、パフォーマンスを見積もり、処理能力を計画する方法について説明します。

処理能力の計画

これまでに、理想的な配備アーキテクチャについて説明してきました。しかし、実際の配備サイズは、処理能力の計画によって決定されます。

しかし、特定のハードウェアの処理能力、または特定のアプリケーション負荷や顧客基準に見合ったハードウェアリソースをどのように見積もればよいのでしょう。これは、実際のアプリケーション、および現実的なデータと作業負荷のシミュレーションを使った慎重なパフォーマンスベンチマークプロセスによって行われます。次に、基本的な手順について簡単に説明します。

1. 1 つの CPU のパフォーマンスを求める

   まず、わかっている処理能力から得られる最大の負荷耐性を求めます。この値を求めるには、1 つのプロセッサだけを搭載したマシンでアプリケーションのパフォーマンスを測定します。テスト環境で、同じような処理特性を必要とする既存のアプリケーションを組み合わせるか、理想的には実際のアプリケーションと作業負荷を使って測定します。アプリケーションとデータリソースは、実際の配備と同じ層構造に設定しておく必要があります。

2. 垂直スケーラビリティを求める

   プロセッサを追加したときに、パフォーマンスがどれだけ向上するかを求める必要があります。これは、特定の作業負荷の下で生じる共有リソースの競合を間接的に測定するということです。これは、マルチプロセッサシステムでアプリケーションの追加負荷を測定して求めるか、すでにテストが終わっている同様のアプリケーションに関する既存の情報から求められます。CPU の数を 1 から 8 まで順に増やしてテストすることで、システムの垂直スケーラビリティの概要を把握することができます。正しい調査結果を得るには、アプリケーション、アプリケーションサーバーとバックエンドデータベースのリソース、オペレーティングシステムなどを適切に調節しておく必要があります。

3. 水平スケーラビリティを求める

   十分に強力なハードウェアリソースを利用できる場合は、単独のハードウェアノードだけでもパフォーマンス要件を満たすことができます。しかし、サービスの可用性を向上するために、複数のシステムによるクラスタ化が必要になることもあります。外部のロードバランサーを導入した場合の作業負荷をシミュレートし、手順 2 で求めた 1 つのチューニング済みアプリケーションサーバーノードのパフォーマンスと比較して、どれだけの向上を見込めるかを調べます。

次の表は、処理能力の計画手順をまとめたものです。

表 : パフォーマンスに影響する要因 - コンセプトの対象

コンセプト	コンセプトの対象	測定	計算方法
ユーザー負荷	負荷ピーク時の並行セッション	1 分あたりのトランザクション数 (TPM)	((負荷ピーク時の並行アクセスユーザー数) × (見込まれる応答時間))/ (クリックのインターバル)
		1 秒あたりの Web 対話 (WIPS)	たとえば、(並行アクセスユーザー数 100 ×応答時間 2 秒) / (クリックインターバル 10 秒) = 20

表 : パフォーマンスに影響する要因 - コンセプトの対象 (続き)

アプリケーションのスケーラビリティ	1 CPU あたりのトランザクションレート	**TPM または WIPS**	作業負荷ベンチマークから求める。各層で実行する必要がある
	サーバー内のスケーラビリティ (CPU の追加によるパフォーマンスの向上)	CPU の追加による向上割合 (%)	ベンチマーク曲線の当てはめから求める。CPU の数を徐々に増やしながらテストを行う。CPU を追加してもパフォーマンスが向上しなくなる位置を曲線上で特定する。このマニュアルの後の章で説明するチューニングが必要。重層構造の場合は各層でテストを繰り返す。パフォーマンス要件を満たせるようであれば、ここで終了
	クラスタ内のスケーラビリティ (サーバーの追加によるパフォーマンスの向上)	サーバープロセス、ハードウェアノード、または両方の追加による向上割合 (%)	前の手順と同様に、チューニング済みアプリケーションサーバーインスタンス 1 つを使用する。サーバーインスタンス、ハードウェアノード、または両方を徐々に追加しながらパフォーマンスの向上を測定する
安全対策	高可用性要件	障害にも対応できるシステムが必要な場合は、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が機能しなくなった場合を前提にパフォーマンス要件を満たせるサイズまでシステムを拡張する	高可用性が必要な場合は、別の式を用いる
	予定外のピークに対する冗長性	安全マージンを確保できるように、サーバーの運用ピークをベンチマークより下に設定する	ピーク負荷をシステム処理能力の 80% に設定すれば、通常は問題ない。実際の、またはシミュレートしたピーク負荷の下で配備環境を測定する

パフォーマンスチューニングの流れ

配備環境のチューニングは、次の順序で行います。

- アプリケーションのチューニング
- Sun ONE Application Server のチューニング
- Java 実行システムのチューニング
- オペレーティングシステムのチューニング

設定ファイル

Sun ONE Application Server の設定ファイル `init.conf`、`obj.conf`、`server.xml` には、変更することでパフォーマンスを向上できる数多くの属性が設定されています。このマニュアルでも随所でとりあげられるこれらのファイルは、次のディレクトリに保存されています。

```
<APPSERVER_HOME>/appserv/domains/<DOMAIN_NAME>/<SERVER_NAME>/config/
```

`APPSERVER_HOME` は、Sun ONE Application Server のインストールディレクトリです。`DOMAIN_NAME` と `SERVER_NAME` は、設定するサーバーインスタンスのドメイン名とサーバー名です。

次の図は、あるインスタンスの設定ファイルの内容を示しています。

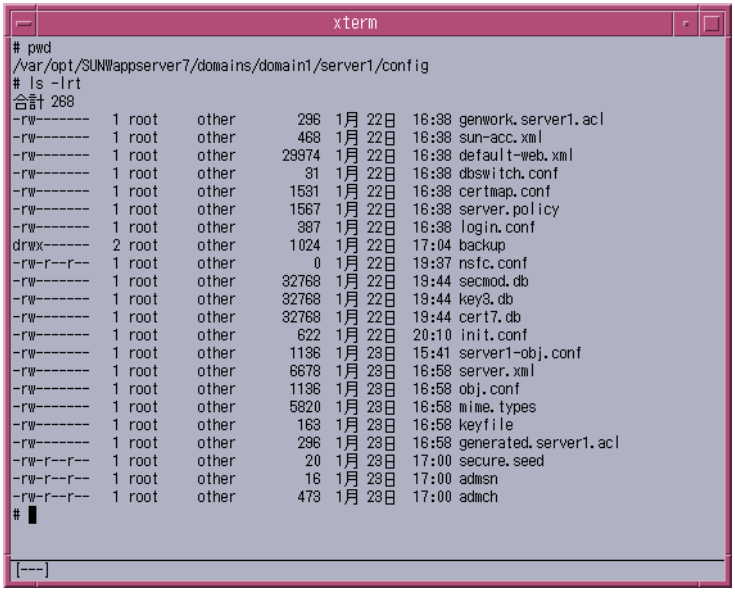


図 : Sun ONE Application Server の設定ファイル

config/backup ディレクトリには、サーバー設定ファイルのコピーが保存されています。これらのファイルは、管理サーバーインスタンスによって自動的に作成されます。一般に、ユーザーがファイルを手動で変更することはお勧めできません。config ファイルを手動で変更した場合は、backup ディレクトリに保存されているファイルのコピーも置き換えてください。また、サーバーインスタンスの再起動も必要となります。

ログの記録とパフォーマンスへの影響

Sun ONE Application Server が生成するログメッセージと例外のスタックトレース出力は、ログファイルに書き込まれます。ログメッセージと例外スタックは、各インスタンスの logs ディレクトリに保存されます。ログの記録はサーバーのパフォーマンスに影響するので、特にベンチマーク測定時には注意を払う必要があります。

デフォルトでは、ログレベルは INFO に設定されています。log_service 要素の属性レベルを変更することで、サーバーのすべてのサブシステムのログレベルを変更できます。特定のサブシステムに設定したログレベルは、これに優先して適用されます。たとえば、mdb_container 要素の log_level 属性を変更することで、mdb_container はサーバーのデフォルトとは異なるレベルでログメッセージを生成するようになります。ログレベルを FINE、FINER、または FINEST に設定することで、より多くのデバッグメッセージを記録できます。ベンチマーク測定時のログレベルは、SEVERE が適当でしょう。

第 3 章

アプリケーションのチューニング

次に、最大のパフォーマンスを得るためのアプリケーションチューニングについて詳しく説明します。優れた性能を持つ Java アプリケーションおよび J2EE アプリケーションの記述方法についての完全な説明は、このマニュアルでは取り扱いません。

この章では、次の項目について説明します。

- Java プログラミングのガイドライン
- J2EE プログラミングのガイドライン

Java プログラミングのガイドライン

ここでは、Java プログラミングとパフォーマンスに関係する問題について取り上げます。ここに示すガイドラインは Sun ONE Application Server に固有のものではなく、多くの状況に適用できる一般的なルールです。Java の最適な記述方法については、http://java.sun.com/blueprints/performance/index.html に用意されている Java BluePrint を参照してください。

- 直列化と直列化復元をできるだけ避ける

  Java では、オブジェクトの直列化と直列化復元は CPU を多用するため、アプリケーションの速度低下を招きがちです。直列化されたデータの量を減らすには、`transient` キーワードを使います。`readObject()` メソッドと `writeObject()` メソッドのカスタマイズが役立つことがあります。

- 「`+`」演算子の代わりに `StringBuffer.append()` を使う

  Java では、String は不変であり、作成後に変更されることはありません。たとえば、次のようなコードがあります。

```
String str = "testing";
str = str + "abc";
```

これは、コンピュータによって次のように解釈されます。

```
String str = "testing";
StringBuffer tmp = new StringBuffer(str);
tmp.append("abc");
str = tmp.toString();
```

このため、コピーは本質的にはリソースの消費につながり、頻用した場合はパフォーマンス低下の大きな要因となります。代わりに `StringBuffer.append()` を使うことをお勧めします。

- 不要になった変数の値には null を明示的に割り当てる

  こうすることで、安全に回収できるメモリ領域をガベージコレクターが簡単に識別できます。Java ではメモリ管理が自動化されているため、メモリ消費とメモリリークの超過を防ぐことができません。アプリケーションが参照を解放せずにオブジェクトを保持し続けると、メモリリークが生じることがあります。この場合、Java のガベージコレクターはこれらのオブジェクトを回収できなくなり、結果としてメモリの消費量が増えます。トランザクションごとに不要な変数への参照を無効化することで、ガベージコレクターはメモリを回収できます。メモリリークを検出するには、プロファイリングツールを使ってトランザクションが終わるごとにメモリの消費状況を調べる方法があります。メモリリークのない健全なアプリケーションでは、ガベージコレクション後に一定したヒープメモリが記録されます。

- メソッドを不必要に `final` と宣言しないようにしてください。最近の動的な最適化コンパイラであれば、Java メソッドが `final` でない場合でも、自動インライン展開による最適化を実行できます。このキーワードは、プログラムのアーキテクチャとして必要な場合や、メンテナンス上の理由など、本来の目的だけに使用します。メソッドのオーバーライドを避ける必要があると確信を持てる場合にだけ `final` キーワードを使ってください。

- 定数を宣言するときは、`static final` を使います。動的なコンパイラは、ヒントを与えるだけで式を評価し、定数畳み込み最適化を実行できます。

- コードにファイナライザを含めると、ガベージコレクターによるリソースの消費が増え、動作を予測しにくくなります。仮想マシンでは、ファイナライザがいつ実行されるかを事前に知ることはできません。プログラムが終了するまでに、ファイナライザが実行されないことも考えられます。`finalize()` メソッドを使って重要なリソースを解放すると、アプリケーションの動作を予想しにくくなることがあります。

- メソッド内で引数が変更されないときは、そのメソッド引数に `final` を宣言します。一般に、初期化後、または値の設定後に変更されないすべての変数には、`final` を宣言します。

- 同期が必要ない場合は、コードブロックやメソッドを同期させないでください。スケーラビリティのボトルネックとならないよう、ブロックやメソッドの同期は最小にとどめます。非同期のデータ構造では、`java.util.HashTable` などのリソース消費の多い方法の代わりに、Java コレクションフレームワークを使います。

J2EE プログラミングのガイドライン

J2EE モデルは、エンタープライズアプリケーション開発のフレームワークを定義します。これは、ソフトウェアの基本コンポーネント (JSP、サーブレット、EJB) のコンテナと、コンテナサービス (JAAS、JDBC、JNDI、JTA など) を定義します。J2EE モデルのすべての構成要素にはそれぞれの用途があります。次の各項では、アプリケーションアーキテクチャの設計時に考慮すべき事項について説明します。

サーブレットと JSP プログラミングのガイドライン

Sun ONE Application Server 上で稼働する多くのアプリケーションは、プレゼンテーション層の JSP またはサーブレットによって処理されます。サーブレットと JSP は、より複雑なトランザクションビジネスロジックが実装された EJB のエントリポイントとなります。サーブレットとそれ以外の J2EE API を使って、ある程度複雑なビジネスアプリケーションを作成することも珍しくありません。

- デフォルトのマルチスレッドモデルのサーブレットでは、アプリケーションサーバーのインスタンスごとにサーブレットのインスタンスが一つずつ作成されます。あるアプリケーションインスタンスのサーブレットに対するすべての要求は、同じサーブレットインスタンスで処理されます。このため、サーブレットのコードに同期されたブロックが含まれていると、スレッドの競合が生じます。修正されたクラス変数の共有は、同期の原因となるので避けてください。
- セッションの作成によってもリソースは消費されます。セッションは、必要な場合にだけ作成します。また、不要になったセッションは無効化してください。
- JSP で必要のない場合に HTTP セッションの自動作成を停止するには、`<%page session="false"%>` 指令を使います。
- 規模の大きなオブジェクトグラフを `HttpSession` に格納しないでください。これにより、Java の直列化が生じ、オーバーヘッドの原因となります。
- `HttpSession` アクセスはトランザクションではありません。トランザクションデータのキャッシュとして使わないでください。トランザクションデータは、通常はデータベースに格納され、エンティティ Bean を使ってアクセスされます。障害が発生すると、トランザクションは元の状態にロールバックします。しかし、

HttpSession オブジェクトには古く不正確なデータが残ることがあります。Sun ONE Application Server では、キャッシュされた読み取り専用データへのアクセスを増やせるように、Bean 管理による「読み取り専用」エンティティ Bean を提供しています。

EJB プログラミングのガイドライン

次に、J2EE アプリケーションの EJB コンポーネントのパフォーマンスを向上するためのガイドラインを示します。詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

- 要求ごとの JNDI ルックアップを避けるため、EJB 参照はサーブレットにキャッシュします。
- EJBHome は、サーブレットの `init()` メソッドにキャッシュします。ルックアップの繰り返しはホームインタフェースを頻繁に使用するため、リソースの消費も多くなります。
- Bean 固有のリソースをキャッシュするときは、`setSessionContext()` メソッドまたは `ejbCreate()` メソッドを使います。これも、Bean のライフサイクルメソッドを使ってアプリケーションを動作させる例です。場合によっては、1 サイクルだけでリソースを解放できます。`ejbRemove()` メソッドを使って、取得したリソースを必ず解放してください。
- 可能であれば、データベーストランザクションの分離レベルを下げます。分離レベルを下げることで、データベース層での処理が削減され、アプリケーションパフォーマンスの改善につながります。ただしこれを行うには、事前にデータベーステーブルの利用パターンを十分に分析する必要があります。Sun ONE Application Server では、サーバー設定ファイルの `<jdbc-connection-pool>` でデータベースの分離レベルを設定できます。サーバー設定の詳細については、『Sun ONE Application Server 管理者ガイド』および『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 Application Server ORB には、サーバーと同じ Java 仮想マシンにホストされているクライアントからの呼び出しを最適化するメカニズムが用意されています。たとえば、サーブレットのコードが Enterprise JavaBeans を呼び出したり、Enterprise JavaBeans が同じサーバーインスタンス上の別の Enterprise JavaBeans を呼び出す場合が対象となります。サーブレットと EJB が同じ Java 仮想マシンで稼働している場合は、`-nolocalstubs` フラグを指定せずに `rmic` コンパイラを実行します。これはデフォルトの設定であり、サーバー設定ファイルでは `-nolocalstubs` は指定されていません。

EJB がリモートアプリケーションサーバーでホストされるアプリケーションアーキテクチャでは、デフォルトの動作を変更する必要があります。この変更は、管理用のコマンドインタフェースである asadmin を使って行うか、アプリケーションが配備されるサーバーインスタンス上で直接行います。ブラウザベースの管理インタフェースでは、rmic オプションは「JVM 設定」タブに表示されます。ローカルスタブを使用することで、パフォーマンスは大きく向上します。これは、スタブジェネレータのデフォルトの動作です。

- 対象データにアクセスするパスをアプリケーション開発者が確実に知っているのであれば、参照による受け渡しを行うように Bean を設定できます。これにより、メソッドを呼び出したときに引数をコピーしたり、メソッドからの返される結果をコピーしたりする必要がなくなります。ただし、呼び出し時に別のソースがデータを変更してしまった場合は問題が生じます。この値は、次の方法で sun-ejb-jar.xml 配備記述子に設定できます。
  `<pass-by-reference>true</pass-by-reference>` (各 EJB 単位で設定)
- 不要になったステートフルセッション Bean は削除します。これにより、ステートフルセッション Bean の非活性化と、ディスク入出力を削減できます。

次に、パフォーマンスへの影響が少ない順に EJB の種類を示します。

- ステートレスセッション Bean とメッセージ駆動型 Bean
- ステートフルセッション Bean
- Bean 管理による持続性を持つエンティティ Bean (読み取り専用に設定)
- コンテナ管理による持続性を持つエンティティ Bean (CMP)
- Bean 管理による持続性を持つエンティティ Bean

EJB のプールとキャッシュ

ステートレスセッション Bean とエンティティ Bean は、どちらもプールすることでサーバーのパフォーマンスを向上できます。また、ステートレスセッション Bean とエンティティ Bean は、キャッシュによってもサーバーのパフォーマンスを向上できます。

表 : Bean の種類、およびプールとキャッシュへの対応

Bean の種類	プール	キャッシュ
ステートレスセッション	Yes	No
ステートフルセッション	No	Yes
エンティティ	Yes	Yes

プールされた Bean とキャッシュされた Bean の違いは、プールされた Bean はすべてが等しく、それぞれを区別できないことです。反対に、キャッシュされた Bean の場合、ステートフルセッション Bean は対話状態を含み、エンティティ Bean は主キーと関連します。エンティティ Bean は、`ejbActivate()` でプールから削除されてキャッシュに追加され、`ejbPassivate()` でキャッシュから削除されてプールに追加されます。必要なエンティティ Bean がキャッシュに見つからないと、コンテナは `ejbActivate()` を呼び出します。キャッシュのサイズが設定されている限界値を超えると、コンテナは `ejbPassivate()` を呼び出します。

次に、EJB のプールとキャッシュの設定をチューニングする方法について説明します。

- EJB プールを使えるのは、ステートレスセッション EJB とエンティティ EJB です。ステートレスセッション EJB の使い道と、サーバーが対応できるトラフィック量に注意して、作成と削除の回数が多くなりすぎないようにプールサイズをチューニングする必要があります。`sun-ejb-jar.xml` 配備記述子の *bean-pool* 要素を参照してください。
- キャッシュを使えるのは、ステートフルセッション EJB です。ステートフルセッション EJB の使い道と、サーバーが対応できるトラフィック量に注意して、活性化と非活性化を最小に押さえられるように EJB のキャッシュサイズとタイムアウト設定をチューニングする必要があります。`sun-ejb-jar.xml` 配備記述子の *bean-cache* 要素を参照してください。
- クライアント側で `remove()` メソッドを明示的に呼び出して、コンテナキャッシュからステートフルセッション EJB を削除できるようにします。
- エンティティ Bean には、EJB プールと EJB キャッシュの両方の設定が適用されます。Bean の作成と削除を最小化できるように、エンティティ EJB のプールサイズをチューニングします。0 以外でサイズが一定の内容をプールに設定しておくことで、最初の要求に対する応答が向上します。
- Bean 固有のリソースをキャッシュするときは `setEntityContext()` メソッドを使い、解放するときは `unSetEntityContext()` メソッドを使います。
- 遅延ロードを使って、不必要な子データを最初から取り込まないようにします。
- 読み取り専用の操作には、読み取り専用のエンティティ Bean を設定します。

トランザクション

トランザクションを使用する場合は、次のような処理を行います。

- リソースを不必要なほど長く保持されないように、トランザクションがユーザーの入力時間や思考時間を大きく超えないようにします。
- 整合性を得るには、コンテナ管理トランザクションのほうが適しています。また、パフォーマンスにも優れています。

- セッション EJB のトランザクション以外のメソッドを宣言するときに、トランザクション属性 `NotSupported` または `Never` を使わないでください。これらの属性は、`ejb-jar.xml` 配備記述子ファイルにあります。トランザクションはデータベースの行をロックするので、使用時間をできるだけ小さくします。
- 長いトランザクションチェーンでは、トランザクション属性 `TX_REQUIRED` を使います。EJB メソッドが確実に呼び出しチェーンに含まれるようにするには、同じトランザクションを使います。
- すべてのトランザクションで利用できる限り、データベースの負担ができるだけ少ないロックを使います。各メソッドの呼び出し後ではなく、トランザクションが完了してからデータをコミットします。
- 1 つのトランザクションに複数のデータベースリソース、コネクタリソース、JMS リソース、またはその組み合わせが関連するときは、分散トランザクションまたはグローバルトランザクションを実行する必要があります。これには、XA に対応したリソースマネージャとデータソースが必要です。XA 対応データソースは、1 つのトランザクションに複数のデータソースが関連する場合にだけ使います。ほとんどの場合に 1 つのまたはローカルのデータベーストランザクションに使われるデータベースが、いくつかの分散トランザクションにも関わる場合は、サーバー設定ファイルに 2 つの `<jdbc-resource>` 要素を登録し、アプリケーションに適したリソースを使うようにします。

JDBC とデータベースアクセス

次に、JDBC 接続プールをチューニングする方法について説明します。

- 大規模なデータベースの検索など、大容量のデータを扱うときは、エンティティ EJB ではなく、JDBC を直接使います。
- ビジネスロジックと、処理するロジックに必要なデータを保持するエンティティ EJB を組み合わせます。
- 接続が確実にプールに戻るように、使用後は常に接続を閉じます。

JMS

JMS を使用する場合は、次のような処理を行います。

- メッセージ駆動型 EJB のプールサイズは、メッセージの並行処理を最適化できるようにチューニングします。
- Bean 固有のリソースをキャッシュするときは `setMesssageDrivenContext()` メソッドまたは `ejbCreate()` メソッドを使い、解放するときは `ejbRemove()` メソッドを使います。

注	アプリケーションをある程度以上の数の EJB に分割すると、アプリケーションのパフォーマンスが低下し、オーバーヘッドも増加します。JavaBeans とは異なり、EJB は単なる Java オブジェクトではありません。EJB は Java オブジェクトより高いレベルのエンティティです。EJB は、リモート呼び出しインタフェースセマンティクス、セキュリティセマンティック、トランザクションセマンティクス、およびプロパティから構成されるコンポーネントです。

参考資料

- Java のパフォーマンスに関する詳細なガイドラインについては、次を参照してください。

  http://java.sun.com/blueprints/performance/index.html

- EJB の最適化については、次を参照してください。

  http://developer.java.sun.com/developer/technicalArticles/ebeans/sevenrules/

第 4 章

Sun ONE Application Server のチューニング

この章では、最適なパフォーマンスを得るために Sun ONE Application Server をチューニングする方法について説明します。この章の項目は次のとおりです。

- HTTP サーバーのチューニング
- 接続プールのチューニング
- JSP とサーブレットのチューニング
- EJB のパフォーマンスチューニング
- 各種 EJB のパフォーマンスに関する注意点
- ORB のチューニング
- 関連する注意点
- トランザクションマネージャのチューニング

HTTP サーバーのチューニング

Sun ONE Application Server のパフォーマンスを最大限に引き出すには、クライアントからの要求を処理する HTTP サーバーの監視とチューニングが重要です。ここでは、HTTP サーバーのチューニングに関する次の項目について説明します。

- stats-xml による統計の有効化
- perfdump ユーティリティによる現在のアクティビティの監視
- 統計に基づくサーバーのチューニング
- ビジー機能

- パフォーマンスバケットの使用
- ファイルキャッシュの設定
- ACL ユーザーキャッシュのチューニング
- サービス品質機能の使用
- スレッド、プロセス、および接続
- Java パフォーマンスの改善
- init.conf のその他の指令
- obj.conf のその他のパラメータ
- サーバーのスケーリング

stats-xml による統計の有効化

perfdump などの既存の監視ツールや同様のカスタムツールを利用するには、stats-xml を使って統計を有効化する必要があります。

stats-xml を使って統計を有効化する手順は、次のとおりです。

1. obj.conf のデフォルトオブジェクトの下に次の行を追加します。

   ```
   NameTrans fn="assign-name" from="/stats-xml/*" name="stats-xml"
   ```

2. obj.conf に次の Service 機能を追加します。

   ```
   <Object name="stats-xml">
   Service fn="stats-xml"
   </Object>
   ```

   次の図は、stats-init というサーバーアプリケーション機能 (SAF) を組み込んだ <instancename>-obj.conf の例です。

xterm

```
<Object name="default">
AuthTrans fn="match-browser" browser="*MSIE*" ssl-unclean-shutdown="true"
NameTrans fn="ntrans-j2ee" name="j2ee"
NameTrans fn=pfx2dir from=/mc-icons dir="/opt/SUNWappserver7/lib/icons" name="es-internal"
NameTrans fn=document-root root="$docroot"
# Line added below for Enabling stats-xml
NameTrans fn="assign-name" from="/stats-xml/*" name="stats-xml"
PathCheck fn=unix-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
Service method=(GET|HEAD) type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD|POST) type=*~magnus-internal/* fn=send-file
Error fn="error-j2ee"
AddLog fn=flex-log name="access"
</Object>

<Object name="j2ee">
ObjectType fn=force-type type=text/html
Service fn="service-j2ee" method="*"
</Object>

<Object name="cgi">
ObjectType fn=force-type type=magnus-internal/cgi
Service fn=send-cgi user="$user" group="$group" chroot="$chroot" dir="$dir" nice="$nice"
</Object>

<Object name="es-internal">
PathCheck fn="check-acl" acl="es-internal"
</Object>

# Service Function Added for enabling stats-xml
<Object name="stats-xml">
Service fn="stats-xml"
</Object>
~
[---]
```

図：<instancename>-obj.conf に組み込まれた statsxml-obj

3. init.conf に stats-init SAF を追加します。

   init.conf に stats-init を組み込む例を示します。

   ```
   Init fn="stats-init" update-interval="5" virtual-servers="2000"
   profiling="yes"
   ```

   次の図は、stats-xml が有効化された init.conf ファイルを示しています。

図：stats-xml による統計の有効化

上の例は、次の設定にも適用できます。

- update-interval: 統計を更新する間隔を秒単位で指定します。値を大きく設定するほど更新頻度が下がり、パフォーマンスが向上します。最小値は 1 で、デフォルト値は 5 です。
- virtual-servers: 統計の対象となる仮想サーバーの最大数です。これは、設定する仮想サーバーと同数以上に設定する必要があります。値が小さいほどメモリの消費は少なくなります。最小値は 1 で、デフォルト値は 1000 です。
- profiling: NSAPI パフォーマンスプロファイリングを有効化します。デフォルトは「no」で、サーバーのパフォーマンスに若干有利です。

設定ファイルの編集については、『Sun ONE Application Server NSAPI Programmer's Guide』(英語) を参照してください。

perfdump ユーティリティによる現在のアクティビティの監視

perfdump ユーティリティは、Sun ONE Application Server に内蔵されている SAF です。これは、アプリケーションサーバーの内部統計からパフォーマンスに関するさまざまな情報を収集し、ASCII テキストとして出力します。perfdump ユーティリティを使うことで、より多くの統計を監視できます。

perfdump ユーティリティのインストール

次の図は、perfdump ユーティリティが設定された <instancename>-obj.conf ファイルの例を示しています。

図：perfdump が設定された <instance-name>-obj.conf ファイルの例

perfdump をインストールするには、<instancename>-obj.conf ファイルを次のように変更します。

1. <instancename>-obj.conf ファイルのデフォルトオブジェクトの後に、次のオブジェクトを追加します。

   ```
   <Object name="perf">
   Service fn="service-dump"
   </Object>
   ```

2. デフォルトオブジェクトに次の行を追加します。

   ```
   NameTrans fn=assign-name from="/.perf" name="perf"
   ```

3. 有効化されていない場合は stats-xml を有効化します。

   stats-xml を有効化する方法については、「stats-xml による統計の有効化」を参照してください。

4. サーバーソフトウェアを再起動します。

5. 次の URL を使って perfdump にアクセスします。

   ```
   http://yourhost/.perf
   ```

6. perfdump に統計を要求し、ブラウザの自動更新頻度 (秒単位) を設定できます。次の例では、5 秒ごとの更新が設定されています。

   ```
   http://yourhost/.perf?refresh=5
   ```

次の図は、perfdump の出力例を示しています。

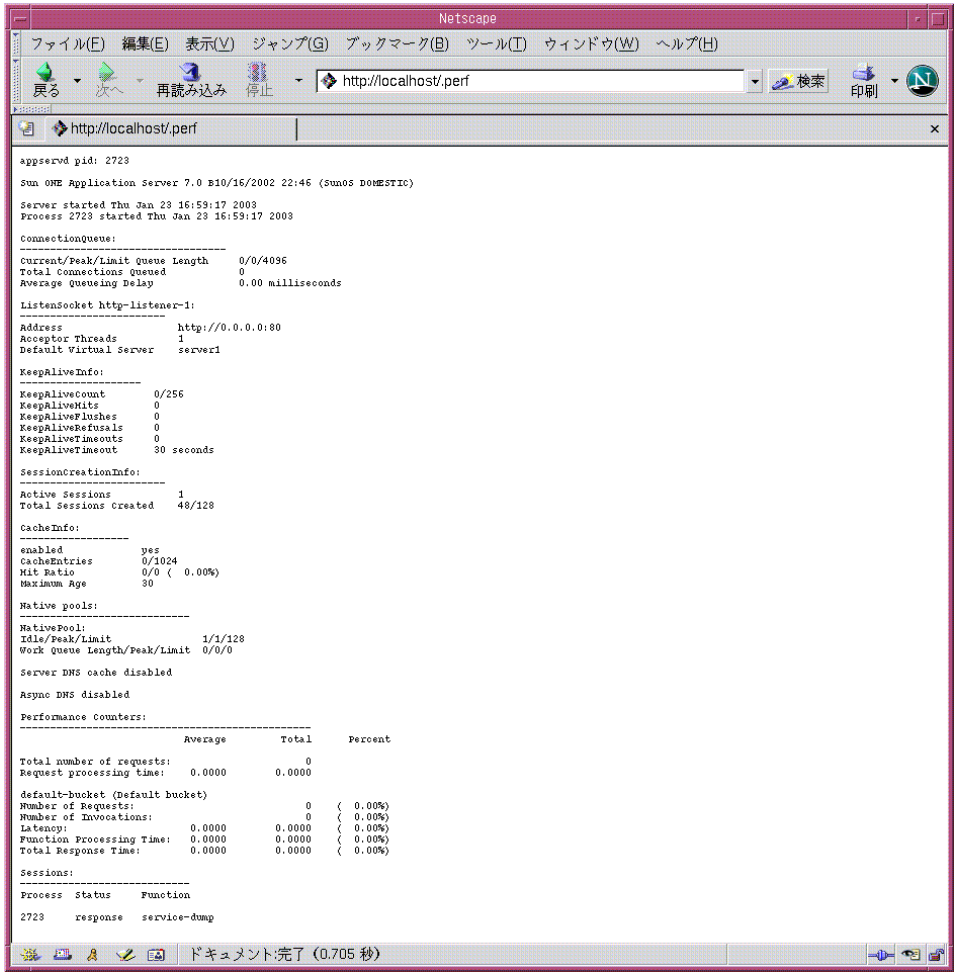


図 : perfdump の出力例

ファイルの編集については、『Sun ONE Application Server Developer's Guide to NSAPI』(英語) を参照してください。

統計に基づくサーバーのチューニング

ここでは、perfdump ユーティリティから得られる情報と、この情報に基づいて一部のパラメータをチューニングし、サーバーのパフォーマンスを向上する方法について説明します。デフォルトのチューニングパラメータは、非常に大規模な場合を除いてすべてのサイトに適用できます。大規模なサイトで定期的な変更が必要なパラメータは、RqThrottle、MaxKeepAliveConnections、KeepAliveTimeout だけです。これらのパラメータをチューニングするには、Web ベースの管理インタフェースを使うか、<instancename>-obj.conf ファイルを直接編集します。次の図は、管理インタフェースの HTTP サーバーチューニング画面を示しています。

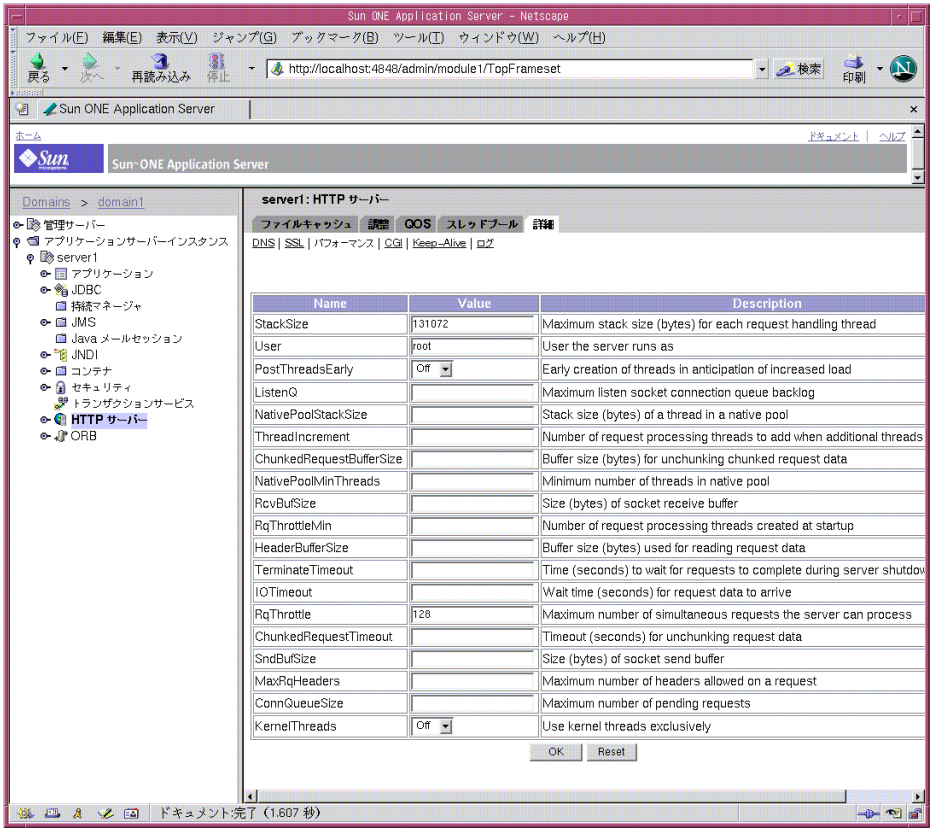


図 : 管理インタフェースによるサーバーパフォーマンスのチューニング

perfdump ユーティリティが監視する統計は、次のカテゴリに分かれています。

- 接続キュー情報
- HTTP リスナーの情報
- キープアライブ (持続的) 接続の情報
- セッション作成情報
- キャッシュ情報
- スレッドプール
- DNS キャッシュ情報

接続キュー情報

接続キュー情報は、キューに含まれるセッション数と、接続受け入れまでの平均所要時間を示します。

perfdump では、この統計は次のように表示されます。

```
ConnectionQueue:
---------------------------------
Current/peak/limit queue length 0/48/5000
Total connections queued 3753
Average queueing delay 0.0013 seconds
```

Current /peak /limit

Current/peak/limit queue length は、順に次の値を示します。

- 現時点でキューに含まれる接続の数
- キューに入れることができる接続の最大数
- 接続キューの最大サイズ

チューニング

キューに含まれる接続の数が上限に近づいた場合、負荷が大きくなる状況ではキューの最大サイズを増やして接続を維持することが必要になります。

キューの最大サイズを変更するには、次の操作を行います。

- Web ベースの管理インタフェースで、ConnQueueSize の値を設定または変更します。「HTTP サーバー」、「詳細」、「パフォーマンス」の順に移動してください。
- init.conf の ConnQueueSize 指示を編集します。

警告 接続キューのサイズを大きく設定すると、サーバーのパフォーマンスが低下することがあります。サイズを設定する目的は、サーバーが処理しきれないほどの接続によってオーバーロードが生じることを防ぐことにあります。オーバーロードが生じた場合に接続キューのサイズを大きくすると、要求処理の待ち時間はさらに長くなり、接続キューは再びいっぱいになってしまいます。

Total Connections Queued

Total Connections Queued は、キューに入れられた接続の総数です。これには、新たに受け入れられた接続と、キープアライブシステムからの接続が含まれます。

これは収集された統計情報であり、チューニングできません。

Average Queuing Delay

Average Queuing Delay は、接続が接続キューに入れられていた平均時間を示します。これは、要求された接続をサーバーが受け入れてから、要求処理スレッド (つまりセッション) が要求の処理を開始するまでの時間を意味します。

これは収集された統計情報であり、チューニングできません。

HTTP リスナーの情報

HTTP リスナーの情報には、そのリスナーの IP アドレス、ポート番号、アクセプタスレッドの数、デフォルト仮想サーバーが含まれます。HTTP リスナーの情報の中で、チューニングに最も重要な情報はアクセプタスレッドの数です。

仮想サーバーでは、いくつでもリスナーを待機させることができますが、デフォルトサーバーインスタンス用に少なくとも 1 つを用意する必要があります (通常は `http://0.0.0.0:80`)。

```
Http listeners1:
------------------------
Address http://0.0.0.0:1890
Acceptor threads 1
Default virtual server test
```

チューニング

HTTP リスナーは、Web ベースの管理インタフェースを使って作成および設定できます。詳細は、『Sun ONE Application Server 管理者ガイド』を参照してください。

複数の HTTP リスナーを作成した場合、`perfdump` はすべてのリスナーに関する情報を出力します。

すべての HTTP リスナーの TCP/IP 待機キューのサイズを設定する方法は、次のとおりです。

- `init.conf` の `ListenQ` パラメータを編集します。
- Web ベースの管理インタフェースで「パフォーマンス」ページにアクセスし、「ListenQ」フィールドに値を入力します。

Address

リスナーの待機アドレスです。IP アドレスとポート番号が含まれます。

HTTP リスナーがマシンのすべての IP アドレスで待機している場合は、アドレスの IP 部分は 0.0.0.0 となります。

チューニング

これを設定するときは、待機ソケットを編集します。0.0.0.0 以外の IP アドレスを指定することで、接続ごとにサーバーのシステム呼び出しが 1 つ少なくなります。最大のパフォーマンスを得るには、0.0.0.0 以外の IP アドレスを指定します。

次の図は、管理インタフェースの HTTP リスナーチューニング画面を示しています。

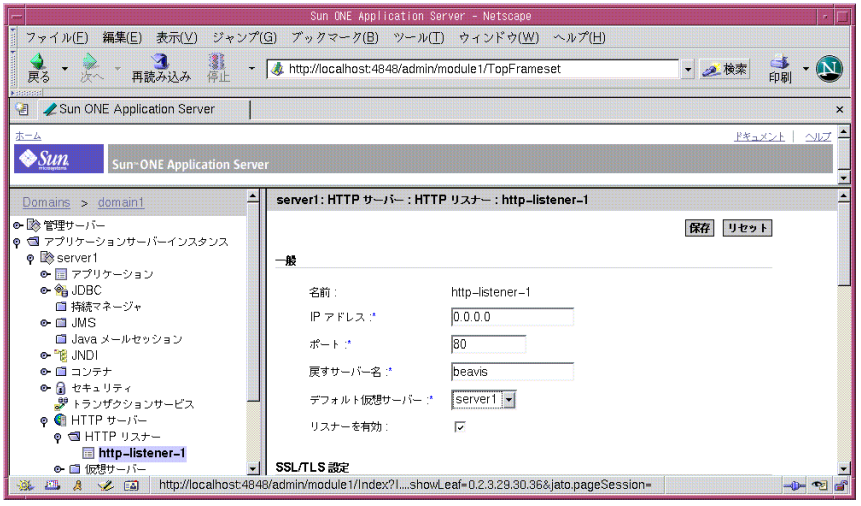


図 : 管理インタフェースによる HTTP リスナーのチューニング

Acceptor Threads

Acceptor Threads は、接続を待機するアクセプタスレッドの数を示します。アクセプタスレッドによって受け入れられ、キューに入れられた接続は、ワーカースレッドによって取り出されます。ユーザーからの要求にいつでも対応できるように、常に十分な数のアクセプタスレッドを確保しておくことが理想的ですが、システムに負荷がかかり過ぎない数に抑える必要があります。システムの CPU ごとに 1 つのアクセプタスレッドを用意することをお勧めします。これで TCP/IP 待機キューがいっぱいになるようであれば、CPU の数の 2 倍程度まで値を増やすことができます。

チューニング

アクセプタスレッドの数を変更するときは、「HTTP リスナー」ノードを選択し、右ペインの「詳細」カテゴリでスレッド数を変更します。

次の図は、管理インタフェースのアクセプタスレッドチューニング画面を示しています。

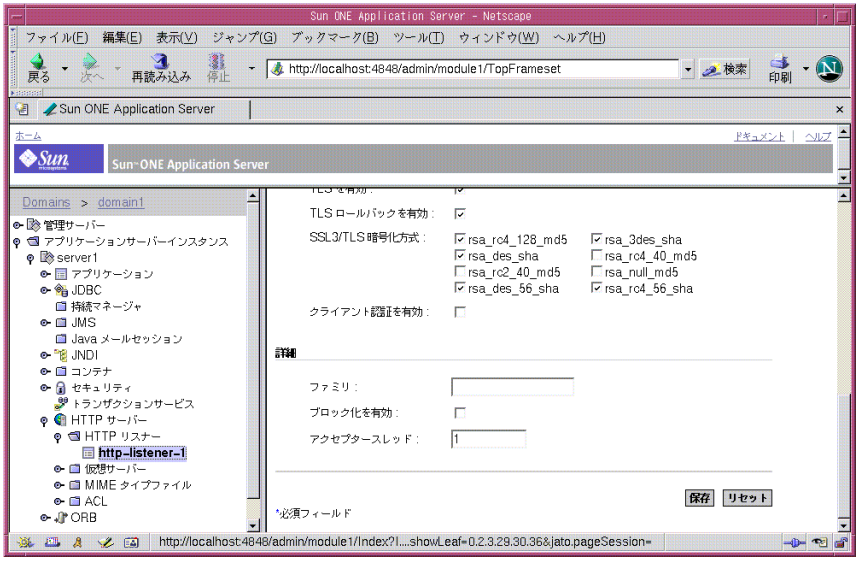


図 : 管理インタフェースによるアクセプタスレッドのチューニング

Default Virtual Server

ソフトウェアの仮想サーバーは、HTTP 1.1 Host ヘッダーを使います。エンドユーザーのブラウザがホストヘッダーを送信しない、または指定された仮想サーバーをサーバーが見つけられないときは、Sun ONE Application Server はデフォルトの仮想サーバーを使って要求を処理します。また、ハードウェアの仮想サーバーも、IP アドレスに対応する仮想サーバーをアプリケーションサーバーが見つけられない場合にデフォルトの仮想サーバーを表示します。デフォルトの仮想サーバーを設定して、特定のドキュメントルートからエラーメッセージまたはサーバーページを送信させることができます。

チューニング

待機ソケットおよびサーバーインスタンスのデフォルトの仮想サーバーを指定できます。HTTP リスナーにデフォルトの仮想サーバーが設定されていない場合は、サーバーインスタンスのデフォルト仮想サーバーが使われます。

待機ソケットのデフォルト仮想サーバーは、Web ベースの管理インタフェースを使って設定できます。Web ベースの管理インタフェースで HTTP サーバーの「プリファレンス」タブで「HTTP リスナー」ページにアクセスし、デフォルト仮想サーバーの情報を設定または変更します。「仮想サーバー」をクリックすると、デフォルト仮想サーバーの設定が表示されます。

キープアライブ (持続的) 接続の情報

ここでは、サーバーの HTTP レベルのキープアライブシステムに関する統計について説明します。

次に、`perfdump` が出力するキープアライブ統計の例を示します。

```
KeepAliveInfo:
--------------------
KeepAliveCount 1/256
KeepAliveHits 4
KeepAliveFlushes 1
KeepAliveTimeout 30 seconds
```

注　この「キープアライブ」と、TCP の「キープアライブ」を混同しないでください。「キープアライブ」と言う表現は HTTP/1.1 で「持続的接続 (Persistent Connection)」に改められましたが、`.perf` では現在も「KeepAlive」という表現が使われています。

HTTP 1.0 と HTTP 1.1 は、どちらも 1 つの HTTP セッションによる複数の要求の送信をサポートしています。Web サーバーは、数百もの新規 HTTP 要求を受信できます。すべての要求が接続を開き続けていれば、サーバーは接続でオーバーフローしてしまいます。UNIX または Linux システムでは、これはファイルテーブルのオーバーフローに直結します。

これに対処するために、サーバーは「待ち」状態にあるキープアライブ接続の最大数を一定に維持します。待ち状態のキープアライブ接続は、前回の要求の処理を完了し、同じ接続からの新しい要求を待っています。新しい接続がキープアライブ要求を待つ場合に、開いている接続数がサーバーの最大待ち接続数を超えているときは、サーバーは最も古い接続を閉じます。サーバーが維持する待ち状態のキープアライブ接続の上限は、このアルゴリズムによって一定に保たれます。

Sun ONE Application Server は、クライアントからのキープアライブ要求に常に応じるとは限りません。次の状況では、クライアントがキープアライブ接続を要求する場合にもサーバーは接続を閉じます。

- `KeepAliveTimeout` が 0 に設定されている。
- `MaxKeepAliveConnections` を超える数の接続が開かれている
- CGI などの動的なコンテンツが HTTP ヘッダーに `content-length` が設定されていない。これは HTTP 1.0 要求だけに適用されます。HTTP 1.1 要求であれば、`content-length` が設定されていない場合でもサーバーはキープアライブ要求に対応します。クライアント側でチャンク形式エンコーディングに対応している (要求ヘッダーの `transfer-encoding:chunked` で識別されます) 場合は、このような要求について、サーバーはこの形式を使います。チャンク形式エンコーディングの詳細は、『Sun ONE Application Server Developer's Guide to NSAPI』(英語) を参照してください。
- 要求が HTTP GET または HTTP HEAD でない
- 要求が不正であると識別される。たとえば、クライアントがコンテンツを含まないヘッダーだけを送信した場合などです。

KeepAliveThreads

キープアライブシステムが使うスレッドの数を設定する方法は、次のとおりです。

- `init.conf` の KeepAliveThreads パラメータを編集します。
- Web ベースの管理インタフェースで、KeepAliveThreads の値を設定または変更します。「HTTP サーバー」、「詳細」タブ、「Keep-Alive」サブメニューの順に移動してください。

KeepAliveCount

ここには次の 2 つの値が出力されます。

- キープアライブモードの接続数

- キープアライブモードの最大同時接続数

チューニング

最も古い接続をサーバーが閉じる前に同時に開くことができる接続の上限値を設定する方法は、次のとおりです。

- init.conf の MaxKeepAliveConnections パラメータを編集します。
- Web ベースの管理インタフェースで、MaxKeepAliveConnections の値を設定または変更します。

注 MaxKeepAliveConnections が指定する接続数は、キープアライブスレッドの間で等分されます。MaxKeeepAliveConnections の値を KeepAliveThreads の値で割り切れないときは、サーバーは実際の MaxKeepAliveConnections の値より若干多めに同時キープアライブ接続を許可します。

KeepAliveHits

キープアライブ状態の接続から正しく受信できた要求の回数です。

これはチューニングできません。

KeepAliveFlushes

KeepAliveCount が MaxKeepAliveConnections を超えているためにサーバーが閉じた接続の数です。

これはチューニングできません。

KeepAliveTimeout

応答のないクライアントとの接続をサーバーが開いた状態で残しておく秒数を表します。Web クライアントは、サーバーへの接続を開いたままにしておくことで、1 つのサーバーに対する複数の要求を 1 つのネットワーク接続だけで処理できます。サーバーが処理できる接続の数には限りがあるため、多数の接続を開いた状態で残しておくと、新しいクライアント接続を開けなくなります。

チューニング

KeepAliveTimeout の設定を変更する方法は、次のとおりです。

- init.conf の KeepAliveTimeout パラメータを編集します。
- Web ベースの管理インタフェースで、KeepAliveTimeout の値を設定または変更します。
- Web ベースの管理インタフェースで「調整」ページにアクセスし、「HTTP 持続的接続のタイムアウト」フィールドに値を入力します。

KeepAliveQueryMeanTime

これは、KeepAlive サブシステムが処理している接続のポーリング間隔を示します。これを N ミリ秒に設定すると、持続的接続を要求したクライアント側から見て、応答時間のオーバーヘッドは 0 ～ N ミリ秒となります。init.conf ファイルを編集しない場合、この値は 1 ミリ秒に設定されます。並行して行われる KeepAlive 接続の数が 300 程度を上回らない限り、この値で問題ありません。並行負荷がこれを超える場合は、デフォルト値ではスケーラビリティに重大な影響が生じます。このような場合は、適切な値に増やすことをお勧めします。

チューニング

KeepAliveQueryMeanTime の値を変更するときは、init.conf の KeepAliveQueryMeanTime パラメータを編集します。

UseNativePoll

UNIX または Linux システムで最大のパフォーマンスを得るには、このパラメータを有効化する必要があります。

キープアライブシステムのネイティブポールを有効化するときは、Web ベースの管理インタフェースを開き、次の手順を実行します。

1. このオプションを有効化するサーバーインスタンスの「HTTP サーバー」ノードを開きます。
2. 右ペインの「詳細」タブをクリックします。
3. 「Keep-Alive」タブをクリックします。
4. UseNativePoll のドロップダウンリストから「ON」を選択します。
5. 「OK」をクリックします。
6. 左ペインのツリーでサーバーインスタンスを選択します。
7. 「変更の適用」をクリックします。
8. インスタンスを再起動して変更を反映させます。

次の図は、キープアライブシステムの設定方法を示しています。

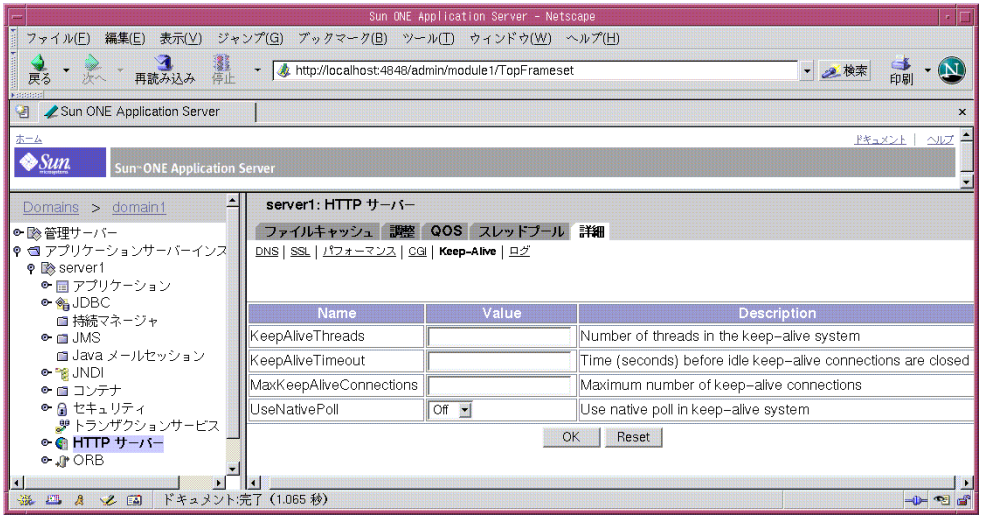


図 : 管理インタフェースによるキープアライブ (持続的) 接続のチューニング

セッション作成情報

`perfdump` では、セッションの作成に関する統計は表示されるだけです。次に、`perfdump` が表示する `SessionCreationInfo` の例を示します。

```
SessionCreationInfo:
------------------------
Active Sessions 1
Total Sessions Created 48/512
```

Active Sessions は、現在要求を処理しているセッション (要求処理スレッド) の数を示します。

Total Sessions Created は、作成されたセッションの数と、作成可能なセッションの最大数を示します。

設定されているセッション数の最大値に近づくことは、必ずしも問題であるとは言えません。サーバーのセッション数を直ちに増やすような対応は必要ありません。この限界値に達することは、ピーク時にサーバーがそれだけのスレッド数を必要とすることを意味します。要求の処理に遅れがない限り、サーバーは適切にチューニングされ

ていると言えます。ただし、制限値を超えた時点で、接続は接続キューに入れられるため、キューがオーバーフローする可能性は否定できません。perfdump の出力を定期的に確認し、作成されるセッションの総数が頻繁に RqThrottle の値に近づく場合は、この制限値を引き上げることを検討してください。

チューニング

スレッド数の制限値を引き上げる方法は、次のとおりです。

- init.conf の RqThrottle パラメータを編集します。
- Web ベースの管理インタフェースで、RqThrottle の値を設定または変更します。
- Web ベースの管理インタフェースで「調整」ページにアクセスし、「最大同時接続」フィールドに値を入力します。

キャッシュ情報

キャッシュ情報のセクションには、ファイルキャッシュがどのように使われているかが示されます。ファイルキャッシュは、サーバーが静的なコンテンツに対する要求を迅速に処理できるように、このようなコンテンツをキャッシュに取り込みます。

perfdump では、キャッシュ統計は次のように表示されます。

```
CacheInfo:
------------------
enabled yes
CacheEntries 5/1024
Hit Ratio 93/190 ( 48.95%)
Maximum age 30
```

enabled

キャッシュを無効にすると、このセクションの残りの部分は出力されなくなります。

チューニング

キャッシュは、デフォルトで有効にされています。無効にする方法は次のとおりです。

- Web ベースの管理インタフェースで HTTP サーバーのインスタンスにアクセスし、「ファイルキャッシュ」タブの「ファイルキャッシュ設定」ページで選択を解除します。
- nsfc.conf ファイルの FileCacheEnable パラメータを編集します。詳細は、『Sun ONE Application Server Developer's Guide to NSAPI』(英語) を参照してください。

CacheEntries

現在キャッシュされているエントリの数と、キャッシュできるエントリの最大数を示します。1 つのキャッシュエントリは、1 つの URI を表します。

チューニング

キャッシュできるエントリの最大数を設定する方法は、次のとおりです。

- Web ベースの管理インタフェースで HTTP サーバーのインスタンスにアクセスし、「ファイルキャッシュ」タブの「ファイルキャッシュ設定」ページで「最大ファイル数」フィールドに値を入力します。
- `nsfc.conf` ファイルの `MaxFiles` パラメータを作成または編集します。詳細は、『Sun ONE Application Server Developer's Guide to NSAPI』(英語) を参照してください。

Hit Ratio (キャッシュのヒット回数 / キャッシュのルックアップ回数)

Hit Ratio は、キャッシュのルックアップ回数に対するヒット回数の割合を示します。値が 100% に近づくほどファイルキャッシュが効率的に行われています。反対に、0% に近づくほどファイルキャッシュが処理している要求が少ないことを意味します。

これはチューニングできません。

Maximum Age

Maximum Age は、有効なキャッシュエントリの最大生存期間を示します。このパラメータは、ファイルをキャッシュした後に、その情報をいつまで使用できるかを制御します。Maximum Age より古いエントリは、同じファイルの新しいエントリに置き換えられます。

チューニング

Web サイトのコンテンツが頻繁に更新されない場合は、大きな値を設定することでパフォーマンスを向上できます。Maximum Age を設定する方法は、次のとおりです。

- Web ベースの管理コンソールで HTTP サーバーのノードにアクセスし、「ファイルキャッシュ」タブの「ファイルキャッシュ設定」ページで「最長有効期間」フィールドに値を入力するか、値を変更します。
- `nsfc.conf` ファイルの `MaxAge` パラメータを編集します。詳細は、『Sun ONE Application Server Developer's Guide to NSAPI』(英語) を参照してください。

次の図は、管理インタフェースのファイルキャッシュシステム設定画面を示しています。

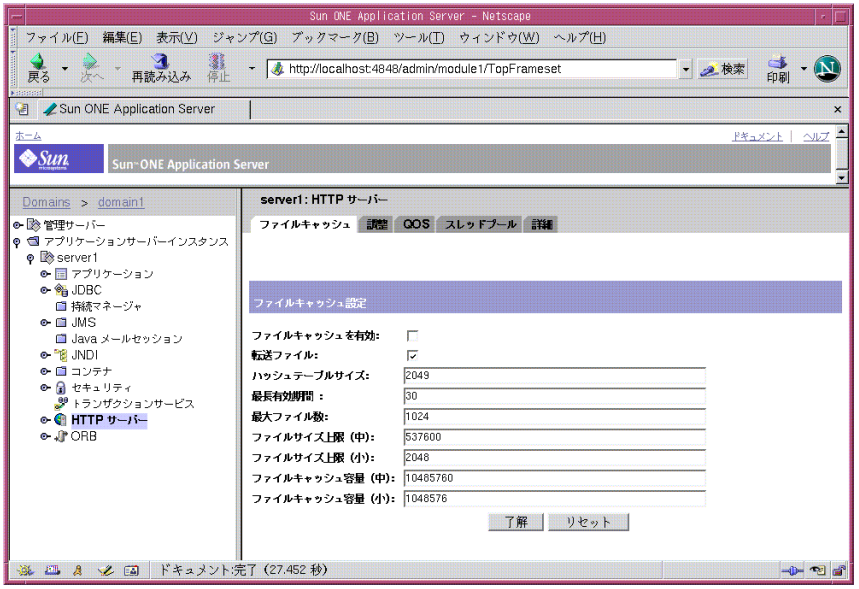


図 : 管理インタフェースによるファイルキャッシュ情報のチューニング

スレッドプール

次の図は、管理インタフェースのスレッドプール設定画面を示しています。

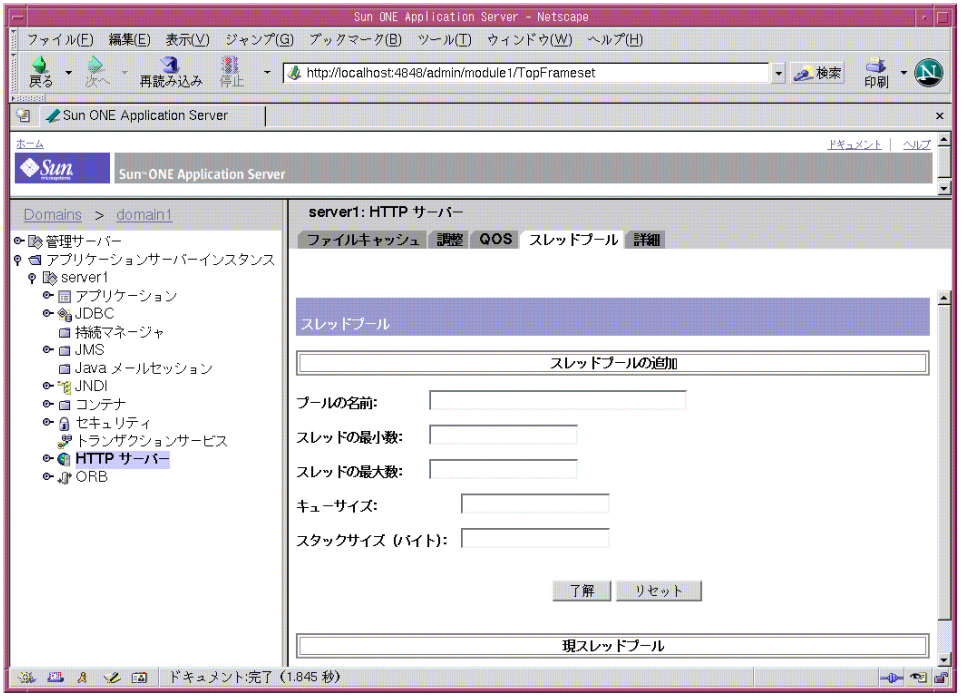


図 : 管理インタフェースによるスレッドプールのチューニング

Web ベースの管理インタフェースでは、3 種類のスレッドプールを設定できます。

- スレッドプール (UNIX/Linux)
- ネイティブスレッドプール (NT)
- 汎用スレッドプール (NT)

スレッドプール (UNIX/Linux のみ)

UNIX または Linux 環境では、スレッドは常にオペレーティングシステム (OS) によってスケジュールされます。ユーザーによるスケジュールは行われないため、この環境ではネイティブスレッドプールを使う必要がありません。従って、UNIX または Linux のユーザーインタフェースでは、このオプションは提供されません。しかし、Web ベースの管理インタフェースを使うことで、必要に応じて OS がスケジュールしたスレッドプールを編集したり、新しいスレッドプールを追加することができます。

ネイティブスレッドプール (NT のみ)

NT 環境では、ネイティブスレッドを必要とする NSAPI 機能を実行するときに、サーバーは内部的にネイティブスレッドプール (NativePool) を使います。

```
Native pools:
----------------------------
NativePool:
Idle/Peak/Limit 1/1/128
Work queue length/Peak/Limit 0/0/0
```

Windows NT ユーザーは、Web ベースの管理インタフェースを使ってネイティブスレッドプールの設定を変更できます。

Sun ONE Application Server は、ホスト OS のサービにアクセスするために、基本的な移植層として NSPR を使います。この層では、OS から提供されるスレッドとは異なる可能性があるスレッドが抽象化されます。これらの非ネイティブスレッドではスケジュールのオーバーヘッドが低いため、これを利用することでパフォーマンスを向上できます。ただしこれらのスレッドは、I/O 呼び出しなどの OS のブロック呼び出しに反応します。ブロック呼び出しを利用できる NSAPI 拡張機能を簡単に記述できるように、サーバーはブロック呼び出しを安全にサポートするスレッドのプールを維持しています。これは、通常は OS スレッドを意味します。要求の処理時に、非ネイティブスレッドでの実行の安全性が印されていないすべての NSAPI 機能は、ネイティブスレッドプールに含まれるいずれかのスレッドで実行するようにスケジュールされます。

NameTrans、Service、PathCheck 機能などの NSAPI プラグインを独自に作成したときは、デフォルトではこれらのプラグインはネイティブスレッドプールに含まれるスレッドで実行されます。作成したプラグインが I/O に NSAPI 機能を明示的に使用する場合、または NSAPI の I/O 機能をまったく使用しない場合は、非ネイティブスレッドで実行することができます。この場合は、ネイティブスレッドが不要であることを示すために、NativeThread="no" オプションを使って機能をロードする必要があります。

この処理を行うには、init.conf ファイルの "load-modules" Init 行に次の行を追加します。

```
Init funcs="pcheck_uri_clean_fixed_init"
shlib="C:/Netscape/p186244/P186244.dll" fn="load-modules"
NativeThread="no"
```

NativeThread フラグは funcslist のすべての機能に影響するため、ライブラリに含まれる複数の機能のうち一部だけがネイティブスレッドを使う場合は、別の Init 行を使います。

汎用スレッドプール (NT のみ)

NT 環境では、Web ベースの管理コンソールを使って追加のスレッドプールを設定できます。スレッドプールを使うことで、サービス機能が一度に処理できる要求の最大数に制限を設けることができます。追加のスレッドプールは、thread-unsafe プラグインの実行に使われます。プールの最大スレッド数を 1 に設定すると、指定されたサービス機能で許可される要求の数が 1 つに制限されます。

Idle/Peak/Limit

Idle は、現在アイドル状態にあるスレッドの数を示します。Peak は、実際にプールに入れられたスレッドの最大数を示します。Limit は、スレッドプールに入れることができるネイティブスレッドの最大値を示します。この値は、`NativePoolMaxThreads` の設定によって決定されます。

チューニング

`NativePoolMaxThreads` を変更する方法は、次のとおりです。

- `init.conf` の `NativePoolMaxThreads` パラメータを編集します。
- 管理インタフェースで HTTP サーバーのノードにアクセスし、「スレッドプール」タブの「Native スレッドプール」ページにある「スレッドの最大数」フィールドに値を入力するか、値を変更します。

Work Queue Length /Peak /Limit

これらの値は、プールに含まれるネイティブスレッドの使用を待つサーバー要求の数を示します。Work Queue Length は、ネイティブスレッドを待つ要求の現在の数を示します。

Peak は、サーバーの起動後にネイティブスレッドの使用を待つために同時にキューに入れられていた要求の最大数を示します。この値は、ネイティブスレッドを必要とする要求の最大並行性とみなすことができます。

Limit は、ネイティブスレッドを待つ要求を一度にキューにいれておける最大数を示します。この値は、`NativePoolQueueSize` の設定によって決定されます。

チューニング

`NativePoolQueueSize` を変更する方法は、次のとおりです。

- `init.conf` の `NativePoolQueueSize` パラメータを編集します。
- 管理インタフェースで HTTP サーバーのノードにアクセスし、「スレッドプール」タブの「Native スレッドプール」ページにある「キューサイズ」フィールドに値を入力するか、値を変更します。

NativePoolStackSize (NT のみ)

NativePoolStackSize は、ネイティブ (カーネル) スレッドプールのスレッドごとのスタックサイズを指定します。

チューニング

NativePoolStackSize を変更する方法は、次のとおりです。

- init.conf の NativePoolStackSize パラメータを編集します。
- Web ベースの管理コンソールで HTTP サーバーのノードにアクセスし、「詳細」タブの「パフォーマンス」サブメニューで NativePoolStackSize の値を設定するか、値を変更します。

NativePoolQueueSize (NT のみ)

NativePoolQueueSize は、キューでスレッドプールを待機するスレッドの数を指定します。プールのスレッドがすべてビジー状態の場合、次の要求処理スレッドはネイティブプールのスレッドが空くまでキューで待機します。キューが満杯の場合は次の要求処理スレッドがキューに入ろうとしても拒否され、クライアントにはビジー応答が返されます。その後、スレッドは解放され、キューで待機状態になっていた要求を処理します。

NativePoolQueueSize の値を RqThrottle より小さく設定すると、プールスレッドからのサービスを待つ要求の数がこの値に達したときに、サーバーは本来の NSAPI 機能を実行せずに、ビジー機能を実行します。デフォルトでは「503 Service Unavailable」が返され、LogVerbose が有効であればメッセージが記録されます。NativePoolQueueSize の値を RqThrottle より大きく設定すると、ビジー機能を実行する前にサーバーは接続を拒否します。

この値は、ネイティブスレッドを必要とするサービスの最大同時要求数を表します。負荷によってシステムが要求を処理できない場合にキューに入れられる要求の数を増やすと、要求の処理を開始するまでの時間が長くなり、要求の処理に利用できるすべてのスレッドがネイティブスレッドを待つことになりかねません。ネイティブスレッドが必要な要求を実行するユーザーの並行最大数を設定することで、この値には、通常は要求が拒否されないように十分な値を設定します。

この値と RqThrottle では、静的な HTML ファイルや画像ファイルなど、非ネイティブスレッド要求のために予約されている要求の数が異なります。予約を維持して要求を拒否することで、サーバーは静的なファイルに対する要求にも対応し続けることができます。こうすることで、動的なコンテンツのロードによって負荷が極端に大きくなる状況でも、サーバーが応答不能になることはありません。サーバーが接続拒否を続けがちなときは、この値が小さすぎるか、サーバーハードウェアがオーバーロードしています。

チューニング

NativePoolQueueSize を変更する方法は、次のとおりです。

- init.conf の NativePoolQueueSize パラメータを編集します。

NativePoolMaxThreads (NT のみ)

NativePoolMaxThreads は、ネイティブ (カーネル) スレッドプールの最大スレッド数を指定します。

大きな値を設定すると、より多くの要求を同時に処理できますが、コンテキストの切り替えが必要となるため、オーバーヘッドは増えます。通常は、この値を変更する必要はありませんが、CPU の利用率が飽和状態ではなく、要求がキューに入れられる状況ではこの値を大きめに設定します。

チューニング

NativePoolMaxThreads の値を変更するときは、init.conf の NativePoolMaxThreads パラメータを編集します。

NativePoolMinThreads (NT のみ)

ネイティブ (カーネル) スレッドプール内のスレッドの最小数を指定します。

チューニング

NativePoolMinThreads を変更する方法は、次のとおりです。

- init.conf の NativePoolMinThreads パラメータを編集します。
- Web ベースの管理コンソールで HTTP サーバーのノードにアクセスし、「詳細」タブの「パフォーマンス」サブメニューで NativePoolMinThreads の値を設定するか、値を変更します。

DNS キャッシュ情報

DNS キャッシュは、IP アドレスと DNS 名をキャッシュに取り込みます。デフォルトでは、サーバーの DNS キャッシュは無効に設定されています。Web ベースの管理インタフェースの「調整」にある「性能の調整」には、次の統計が表示されます。

enabled

DNS キャッシュが無効な状態では、このセクションの残りの部分は出力されなくなります。

チューニング

デフォルトでは、DNS キャッシュは無効に設定されています。DNS キャッシュを有効化する方法は、次のとおりです。

- init.conf に次の行を追加します。

```
Init fn=dns-cache-init
```

- Web ベースの管理インタフェースを使って、「サーバーへアクセスするクライアントを DNS 検索」に値を設定します。

CacheEntries (現在のキャッシュエントリ / キャッシュエントリの最大値)

現在キャッシュされているエントリの数と、キャッシュできるエントリの最大数を示します。1 つのキャッシュエントリは、IP アドレスまたは DNS 名の 1 回のルックアップを意味します。キャッシュは、Web サイトに同時にアクセスするクライアントの最大数に対応できるサイズに設定します。キャッシュに大きすぎる値を設定すると、メモリが無駄になり、パフォーマンスが低下します。

チューニング

DNS キャッシュの最大サイズを設定する方法は、次のとおりです。

- `init.conf` に次の行を追加します。

  ```
  Init fn=dns-cache-init cache-size=1024
  ```

  デフォルトのキャッシュサイズは 1024 です。

- Web ベースの管理インタフェースで「調整」ページにアクセスし、「DNS キャッシュサイズ」フィールドに値を入力するか、値を変更します。

HitRatio (キャッシュのヒット数 / キャッシュのルックアップ回数)

HitRatio は、キャッシュのルックアップ回数に対するキャッシュのヒット回数の割合を示します。

これはチューニングできません。

注　サーバーの DNS ルックアップを無効にすると、ホスト名の制限が機能しなくなり、ログファイルにホスト名が記録されなくなります。代わりに、IP アドレスが記録されます。

DNS エントリのキャッシング

DNS エントリをキャッシュするかどうかも指定できます。DNS キャッシュを有効にすると、ホストは一度受信したホスト名を格納できるようになります。その後、サーバーがそのクライアントに関する情報を必要とする場合、クエリーを実行することなく、キャッシュされている情報を利用できます。DNS キャッシュのサイズ、および DNS キャッシュエントリの有効期限を設定することができます。DNS キャッシュには 32 ～ 32768 のエントリを保存できます。デフォルト値は 1024 エントリです。
キャッシュエントリの有効期限は 1 秒から 1 年の範囲で秒単位で指定できます。デフォルト値は 1200 秒 (20 分) です。

DNS ルックアップの非同期への限定

サーバープロセスでの DNS ルックアップはリソースを多く消費するので、使用しないことをお勧めします。DNS ルックアップを含める必要があるときは、非同期にしてください。

enabled

非同期 DNS を無効にすると、このセクションの残りの部分は出力されなくなります。

チューニング

非同期 DNS を有効化する方法は、次のとおりです。

- `init.conf` ファイルに `AsyncDNS` ON というエントリを追加します。
- Web ベースの管理インタフェースで、`AsyncDNS` の値を ON に設定します。
- サーバーマネージャの「プリファレンス」で「調整」の「非同期 DNS を有効」を選択します。

NameLookups

サーバーを起動してから名前ルックアップ (DNS 名による IP アドレスの検索) を実行した回数を示します。

これはチューニングできません。

AddrLookups

サーバーを起動してからアドレスルックアップ (IP アドレスによる DNS 名の検索) を実行した回数を示します。

これはチューニングできません。

LookupsInProgress

現在実行中のルックアップの数を示します。

これはチューニングできません。

次の図は、管理インタフェースの DNS キャッシュ情報設定画面を示しています。

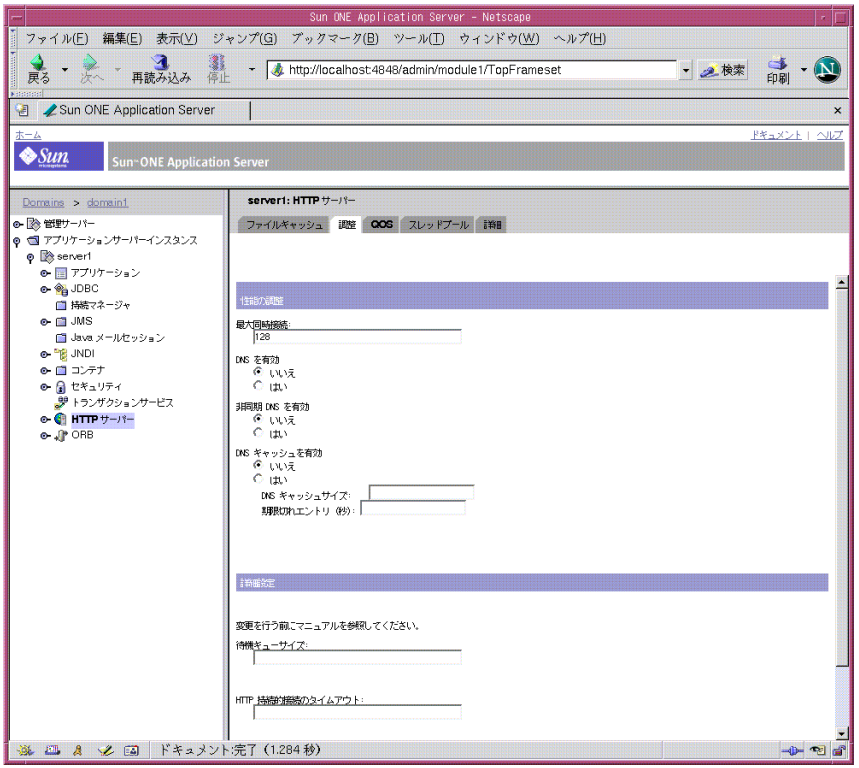


図 : 管理インタフェースによる DNS キャッシュ情報のチューニング

ビジー機能

ビジー機能は、デフォルトでは「503 Service Unavailable」を返し、`LogVerbose` が有効であればメッセージを記録します。アプリケーションによっては、この動作を変更する必要があります。パフォーマンスに関するトラブルシューティングを効率的に行うには、NSAPI 機能について独自のビジー機能を `<instancename>-obj.conf` ファイルに設定すると便利です。この場合は、次の形式で設定ファイルにサービス機能を追加します。

```
busy="<my-busy-function>"
```

たとえば、次のようにサービス機能を設定します。

```
Service fn="send-cgi" busy="service-toobusy"
```

これにより、サーバーが要求の処理時に過度なビジー状態になった場合は、異なる応答 (Service、AddLog、PathCheck など) を返すことができます。独自のビジー機能は、デフォルトのスレッドタイプが非ネイティブな場合に、ネイティブスレッドを必要とするすべての機能に適用されます。

デフォルトのビジー機能ではない独自のビジー機能をサーバー全体で使用するときは、次のように、func_insert 呼び出しを含む NSAPI の init 機能を記述します。

```
extern "C" NSAPI_PUBLIC int my_custom_busy_function(pblock *pb,
Session *sn, Request *rq);

my_init(pblock *pb, Session *, Request *)

{

func_insert("service-toobusy", my_custom_busy_function);

}
```

ビジー機能はプールスレッドでは実行されないため、スレッドのブロックを生じる機能呼び出しを行わないように注意してください。

パフォーマンスバケットの使用

パフォーマンスバケットを使うことで、バケットを定義し、それをさまざまなサーバー機能と関連づけることができます。いずれかの機能が呼び出されると、サーバーは統計データを収集し、それをバケットに追加します。たとえば、send-cgi と NSServletService は、それぞれ CGI 要求と Java サーブレット要求を処理します。CGI 要求とサーブレット要求のそれぞれにバケットを定義してカウンタを独立させることも、1 つのバケットを作成して動的なコンテンツである両方の要求に対応することもできます。この情報収集に要するリソースの消費は少ないので、通常はサーバーのパフォーマンスに与える影響を無視できます。バケットには、次の情報が記録されます。

- **バケット名** : バケットと機能の関連づけにはこの名前が使われます。
- **記述** : バケットに関連づけられている機能の説明です。
- **この機能の要求数** : この機能が呼び出された合計回数です。
- **この機能の実行回数** : 一部の機能は 1 回の要求で複数回実行されるため、この値は機能の要求回数と一致しないことがあります。
- **機能の応答時間またはディスパッチ時間** : サーバーが機能を実行するまでにかかった時間です。
- **機能に要する時間** : 機能の完了までにかかった時間です。

default-bucket は、事前にサーバーに定義されています。これは、ユーザー定義のバケットに関連づけられていない機能の統計を記録します。

設定

パフォーマンスバケットのすべての設定情報は、init.conf ファイルと <instancename>-obj.conf ファイルに設定します。自動的に有効化されるのはデフォルトバケットだけです。

次の例は、init.conf に新しいバケットを定義する方法を示しています。

```
Init fn="define-perf-bucket" name="acl-bucket" description="ACL
bucket"

Init fn="define-perf-bucket" name="file-bucket"
description="Non-cached responses"

Init fn="define-perf-bucket" name="cgi-bucket" description="CGI
Stats"
```

上の例では、acl-bucket、file-bucket、cgi-bucket という 3 つのバケットが作成されます。これらのバケットを機能に関連づけるときは、obj.conf ファイルを開き、パフォーマンスを測定する機能 bucket=*bucket-name* を追加します。次に例を示します。

```
PathCheck fn="check-acl" acl="default" bucket="acl-bucket"
...
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file" bucket="file-bucket"
...
<Object name="cgi">
ObjectType fn="force-type" type="magnus-internal/cgi"
Service fn="send-cgi" bucket="cgi-bucket"
</Object>
```

パフォーマンスレポート

パフォーマンスバケットに関する情報は、perfdump が出力するレポートの最後に含まれます。

詳細は、stats-xml による統計の有効化および「パフォーマンスバケットの使用」を参照してください。

レポートには、次の情報が含まれます。

- 「Avarage」、「Total」、「Percent」列は、各要求の統計データを示します。
- 「Request Processing Time」は、これまでにサーバーが受信したすべての要求の処理にかかった合計時間を示します。
- 「Number of Requests」は、その機能が要求された合計回数を示します。

- 「Number of Invocations」は、その機能が実際に実行された合計回数を示します。一部の機能では、1 つの要求の処理で複数回実行されるため、この値は機能の要求回数と異なることがあります。この行のパーセント値を示す列は、すべてのバケットに記録されている実行回数の合計に対する、その機能の実行回数の割合を示します。
- 「Latency」は、Sun ONE Application Server がその機能を呼び出すまでにかかった時間を秒単位で示します。
- 「Function Processing Time」は、Sun ONE Application Server がその機能の実行に要した時間を秒単位で示します。「Function Processing Time」と「Total Response Time」のパーセントの値は、「Request Processing Time」に対するそれぞれの割合を示します。
- 「Total Response Time」は、「Function Processing Time」と「Latency」の合計で、単位は秒です。

次に、perfdump から出力されるパフォーマンスバケット情報の例を示します。

```
Performance Counters:

------------------------------------------------

                            Average         Total         Percent


Total number of requests:                  474851

Request processing time:    0.0010      485.3198


Default Bucket (default-bucket)

Number of Requests:                           597        ( 0.13%)

Number of Invocations:                       9554        ( 1.97%)

Latency:                    0.0000        0.1526        ( 0.03%)

Function Processing Time:   0.0256      245.0459       ( 50.49%)

Total Response Time:        0.0257      245.1985       ( 50.52%)
```

ファイルキャッシュの設定

Sun ONE Application Server は、静的な情報を迅速に処理するため、ファイルキャッシュを使用します。ファイルキャッシュには、ファイルに関する情報と静的なファイルコンテンツが含まれます。ファイルキャッシュには、サーバー解析 HTML の処理を高速化するために使用する情報も格納されます。

ファイルキャッシュはデフォルトで有効になっています。ファイルキャッシュの設定情報は、`nsfc.conf` ファイルに格納されています。Web ベースの管理インタフェースを使って、ファイルキャッシュの設定を変更することができます。

次の図は、管理インタフェースのファイルキャッシュ設定画面を示しています。

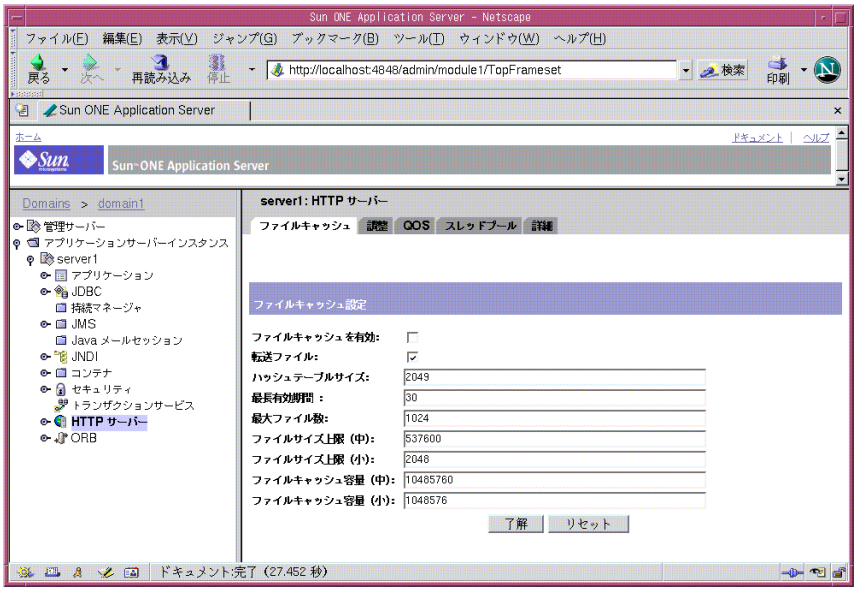


図 : 管理インタフェースによるファイルキャッシュのチューニング

ファイルキャッシュを設定するには、次の手順を実行します。

1. HTTP サーバーの「ファイルキャッシュ」タブを選択します。
2. 「ファイルキャッシュを有効」にチェックマークがつけられていない場合は、これを選択します。

3. ファイルを転送するかどうかを選択します。

   「転送ファイル」を有効にすると、サーバーはファイルの内容ではなく、開いているファイル記述子をファイルキャッシュに取り込みます。クライアントへのファイルコンテンツの送信には、`PR_TransmitFile` が使われます。「転送ファイル」が有効な場合、開いているファイル記述子だけがキャッシュされるので、通常は区別されるファイルのサイズの違いがファイルキャッシュに適用されなくなります。デフォルトでは、「ファイル転送」は NT では有効、UNIX では無効に設定されています。UNIX では、OS が `PR_TransmitFile` をネイティブサポートしているプラットフォームでは「ファイル転送」を有効化します。現時点では、HP-UX と AIX がこれに含まれます。その他の UNIX または Linux プラットフォームで有効にすることはお勧めできません。

4. ハッシュテーブルのサイズを入力します。

   デフォルトのサイズは、ファイルの最大数の 2 倍に 1 を加えた数です。たとえば、ファイルの最大数が 1024 の場合、デフォルトのハッシュテーブルサイズは 2049 となります。

5. キャッシュエントリの最長有効期間を秒単位で入力します。

   デフォルトでは、30 分に設定されます。

   この設定により、ファイルがキャッシュされてから、キャッシュされた情報がどれだけ長く使用されるかが制御されます。`MaxAge` より古いエントリは、同じファイルがキャッシュから参照されるときに、そのファイルの新しいエントリに置き換えられます。

   最長期間を設定するときは、ファイル内容の更新 (既存ファイルの修正) が規則的なスケジュールで行われるかどうかを基準にします。たとえば、コンテンツを 1 日に 4 回等間隔で更新する場合は、最長期間を 21600 (6 時間) に設定します。それ以外の場合は、更新してからその内容を最も長く使用するコンテンツファイルに合わせて最長期間を設定します。

6. キャッシュするファイルの最大数を「最大ファイル数」に入力します。

   デフォルトでは、1024 分に設定されます。

7. 中と小のファイルサイズ上限をバイト単位で指定します (UNIX/Linux のみ)。

   デフォルトでは、「ファイルサイズ上限 (中)」は 525000 (525 キロバイト) に設定されています。

   デフォルトでは、「ファイルサイズ上限 (小)」は 2048 に設定されています。

   キャッシュの動作は、ファイルサイズの大、中、小に応じて異なります。「中規模」のファイルの内容は、ファイルを仮想メモリにマッピングするでキャッシュされます (UNIX/Linux プラットフォーム)。「小さい」ファイルの内容は、ヒープ領域を割り当て、ファイルをそのヒープ領域に読み込むことでキャッシュされます。「大きな」ファイル (「中」より大きなファイル) の場合、ファイルに関する情報はキャッシュされますが、その内容はキャッシュされません。

   小規模なファイルと中規模なファイルを区別することで、小規模なファイルが多い場合に仮想メモリのページを無駄にすることがなくなります。このため、通常は VM ページのサイズより若干少ない容量を「ファイルサイズ上限 (小)」に設定します。

8. 中規模と小規模のファイルキャッシュ容量を設定します (UNIX/Linux のみ)。

   中規模ファイルの領域は、すべての中規模サイズのファイルをマップするために使用する仮想メモリのサイズ (バイト単位) となります。デフォルトでは、10000000 (10 メガバイト) に設定されます。

   小規模ファイルの領域は、小さいファイルをキャッシュするために使用するヒープ領域を含めて、キャッシュに使用するヒープ領域のサイズ (バイト単位) となります。UNIX または Linux 環境では、この値はデフォルトで 1 メガバイトに設定されます。

9. 「了解」をクリックします。

10. 「適用」をクリックします。

11. 「変更の適用」をクリックしてサーバーを再起動します。

nocache パラメータの使用

send-file サービス機能の nocache パラメータを使って、特定のディレクトリに含まれるファイルをキャッシュしないように設定できます。たとえば、キャッシュの有用性を欠くほど頻繁に更新するファイルがある場合は、そのファイルを特定のディレクトリに保存し、`<instancename>-obj.conf` ファイルを編集してサーバーがそのディレクトリ内のファイルをキャッシュしないように設定します。

次に例を示します。

```
<Object name=default>

...

NameTrans fn="pfx2dir" from="/myurl" dir="/export/mydir"
name="myname"
```

```
...
Service method=(GET|HEAD|POST) type=*~magnus-internal/*
fn=send-file
...
</Object>
<Object name="myname">
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
nocache=""
</Object>
```

上の例では、/myurl というプレフィックスを持つ URL から要求された場合に、サーバーは /export/mydir/ ディレクトリに含まれる静的なファイルをキャッシュしません。

ファイルキャッシュの動的な制御と監視

次のように、<instancename>-obj.conf ファイルにオブジェクトを追加することで、サーバーの稼働中に nsfc.conf ファイルキャッシュを動的に監視および制御することができます。

デフォルトオブジェクトに NameTrans 指示を追加します。

```
NameTrans fn="assign-name" from="/nsfc" name="nsfc"
```

オブジェクト定義 nsfc を追加します。

```
<Object name="nsfc">
Service fn=service-nsfc-dump
</Object>
```

これにより、/nsfc という URI からアクセスされるファイルについて、ファイルキャッシュの制御と監視の機能 (nsfc-dump) が有効化されます。NameTrans 指示の from パラメータの値を変更することで、別の URI も指定できます。

次の例は、この URI にアクセスした場合に表示される情報を示しています。

```
Sun ONE Application Server File Cache Status (pid 7960)
The file cache is enabled.
Cache resource utilization


Number of cached file entries = 1039 (112 bytes each, 116368
total bytes)
```

```
Heap space used for cache = 237641/1204228 bytes

Mapped memory used for medium file contents = 5742797/10485760
bytes

Number of cache lookup hits = 435877/720427 ( 60.50 %)

Number of hits/misses on cached file info = 212125/128556

Number of hits/misses on cached file content = 19426/502284

Number of outdated cache entries deleted = 0

Number of cache entry replacements = 127405

Total number of cache entries deleted = 127407

Number of busy deleted cache entries = 17


Parameter settings


HitOrder:false

CacheFileInfo:true

CacheFileContent:true

TransmitFile:false

MaxAge:30 seconds

MaxFiles: 1024 files

SmallFileSizeLimit: 2048 bytes

MediumFileSizeLimit: 537600 bytes

CopyFiles:false

Directory for temporary files:
/tmp/netscape/https-axilla.mcom.com

Hash table size: 2049 buckets
```

`/nsfc` という URI にアクセスするときに、クエリ文字列を含めることができます。認識される値は次のとおりです。

- `?list` - キャッシュに含まれるファイルをリスト表示します。
- `?refresh=n` - クライアントに n 秒ごとにページを再読み込みさせます。
- `?restart` - キャッシュを終了し、開始し直します。

- ?start - キャッシュを開始します。
- ?stop - キャッシュを終了します。

?list オプションを設定したときに表示される情報は、ファイル名、フラグの設定、キャッシュエントリに対する現在の参照回数、ファイルサイズ、内部ファイル ID の値です。フラグには次の種類があります。

- C - ファイルのコンテンツがキャッシュされています。
- D - キャッシュエントリに削除の印がつけられています。
- E - このファイルについて PR_GetFileInfo() がエラーを返しています。
- I - ファイルに関する情報 (サイズ、更新日時など) がキャッシュされています。
- M - ファイルのコンテンツが仮想メモリにマッピングされています。
- O - ファイル記述子がキャッシュされています (TransmitFile が true に設定されている場合)。
- P - ファイルがプライベートデータに関連づけられています (shtml ファイルで表示されます)。
- T - キャッシュエントリに一時ファイルがあります。
- W - キャッシュエントリへの書き込みアクセスが禁止されています。

コンテンツの更新がスケジュールされているサイトでは、コンテンツの交信中はキャッシュを停止し、更新完了後に開始し直します。パフォーマンスは低下しますが、キャッシュがオフでもサーバーは通常どおりに機能します。

ACL ユーザーキャッシュのチューニング

デフォルトでは、ACL ユーザーキャッシュは ON に設定されています。キャッシュのデフォルトサイズ (200 エントリ) により、ACL ユーザーキャッシュがボトルネックになったり、トラフィックの多いサイトではキャッシュが機能しないことがあります。アクセスの多いサイトでは、キャッシュエントリのライフタイムが終わるまでに ACL 保護されたリソースに対して 200 以上のユーザーがアクセスすることもあります。このような状況では、Sun ONE Application Server はユーザーを検証するために LDAP サーバーに頻繁にクエリを送信しなければならず、これがパフォーマンスに影響します。

このボトルネックを回避するには、`init.conf` の `ACLUserCacheSize` 指示を使って ACL キャッシュのサイズを増やします。キャッシュサイズを大きくすると、その分だけキャッシュを保持するための RAM も必要になることに注意してください。

また、キャッシュエントリに格納できるグループの数 (デフォルトでは 4) がボトルネックになることもあります (ただし、可能性は高くありません)。ユーザーが 5 つのグループに所属し、ACL キャッシュのライフタイム中に 5 つの異なるグループに対して確認が必要となる 5 つの ACL にアクセスした場合は、追加のグループエントリ用にキャッシュエントリを作成する必要が生じます。2 つのキャッシュエントリがある場合、元のグループ情報を格納したエントリは無視されます。

パフォーマンスに関するこのような問題が生じることは極めてまれですが、1 つの ACL キャッシュエントリに格納できるグループ数を変更するときは、`ACLGroupCacheSize` 指示を使います。

ACL ユーザーキャッシュ指示

ACL ユーザーキャッシュの値を変更するときは、通常は `init.conf` ファイルに次の指示を手動で追加します。

- `ACLCacheLifetime`
- `ACLUserCacheSize`
- `ACLGroupCacheSize`

ACLCacheLifetime

この指示には、キャッシュエントリの有効期間を秒単位で設定します。キャッシュのエントリが参照されるたびに、経過時間が計算され `ACLCacheLifetime` と照合されます。経過時間が `ACLCacheLifetime` 以上の場合、このエントリは使用されません。デフォルト値は 120 秒です。この値を 0 にすると、キャッシュが無効になります。こ

の値を大きくすると、LDAP エントリを変更した場合に Sun ONE Application Server の再起動が必要になることがあります。たとえば 120 秒に設定すると、Sun ONE Application Server は 2 分間 LDAP サーバーと同期が取れなくなる可能性があります。LDAP が頻繁に変更されない場合は、大きな値を設定できます。

ACLUserCacheSize

この指示には、ユーザーキャッシュのサイズをエントリ数単位で設定します (デフォルトは 200)。

ACLGroupCacheSize

この指示には、1 つの UID またはキャッシュエントリにキャッシュできるグループ ID の数を設定します (デフォルトは 4)。

ACL ユーザーキャッシュ設定の確認

`LogVerbose` を使うことで、使用中の ACL ユーザーキャッシュの設定を確認できます。`LogVerbose` を実行している場合は、サーバー起動時のエラーログに次のようなメッセージが記録されます。

```
User authentication cache entries expire in ### seconds.

User authentication cache holds ### users.

Up to ### groups are cached for each cached user.
```

チューニング

`LogVerbose` を ON にするときは、`init.conf` の `LogVerbose` パラメータを編集します。

警告 本稼働環境で `LogVerbose` を ON にしないでください。パフォーマンスが低下し、エラーログのサイズが大きくなります。

サービス品質機能の使用

サービス品質機能を使うことで、サーバーインスタンス、仮想サーバーのクラス、または個々の仮想サーバーの帯域幅と接続数を制限できます。パフォーマンスに関するこれらの制限を設定し、結果を追跡してからオプションとして実行することができます。

次の図は、管理インタフェースのサービス品質設定画面を示しています。

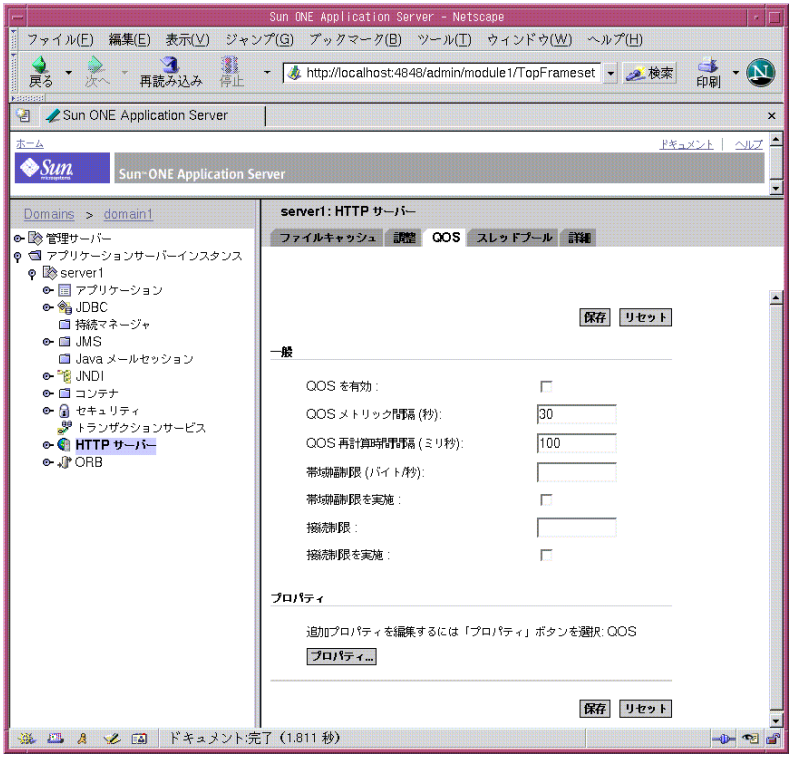


図 : 管理インタフェースによるサービス品質のチューニング

詳細は、『Sun ONE Application Server 管理者ガイド』の「サービス品質の使用」を参照してください。

スレッド、プロセス、および接続

Sun ONE Application Server では、HTTP リスナーのアクセプタスレッドは、受け入れた接続を接続キューで待機させます。次にセッションスレッドがキューから接続を受け取って要求を処理します。要求が必要とする場合は、より多くのセッションスレッドが生成されます。新しいスレッドは、接続キューが次のような状態の場合に追加されます。

- 新しい接続が返されるたびに、キューで待機している接続数 (接続のバックログ) と作成済みセッションスレッドの数が比較されます。待機している接続の数が作成済みスレッドの数よりも多ければ、次の要求完了時にスレッドがさらに追加されます。
- 前述のバックログが追跡され、次の場合に `ThreadIncrement` と同数のスレッドの追加がスケジュールされます。
  - 時間の経過とともにスレッド数が増加している
  - 増加数が `ThreadIncrement` の値より大きい
  - セッションスレッドの数からバックログの値を差し引いた結果が `ThreadIncrement` の値より小さい
- 新しいセッションスレッドの追加が `RqThrottle` によって厳密に制限されている場合に追加されます。
- ベンチマーク負荷の開始などによってバックログが急に増えても、作成されるスレッドが多くならないようにするために、16 回または 32 回ごとの接続時にスレッドが必要かどうかが決定されます。この回数は既に存在するセッションスレッドの数によって決まります。

次の指示は、スレッド、プロセス、接続の数に影響します。チューニングには、Web ベースの管理コンソールまたは `init.conf` を使います。

- `ConnQueueSize`
- `HeaderBufferSize`
- `AcceptTimeOut`
- `KeepAliveThreads`
- `KeepAliveTimeout`
- `KernelThreads`
- `ListenQ`
- `MaxKeepAliveConnections`
- `MaxProcs` (UNIX のみ)
- `PostThreadsEarly`

- RcvBufSize
- RqThrottle
- RqThrottleMin
- SndBufSize
- StackSize
- TerminateTimeout
- ThreadIncrement
- UseNativePoll (UNIX のみ)

これらの指示の詳細については、『Sun ONE Application Server Developer's Guide to NSAPI』(英語) を参照してください。

HTTP リスナーのアクセプタスレッド

待機ソケットで受け入れを待機するスレッドの数を指定することができます。この値は、システムで使用する CPU の数と同じ、またはそれより小さく設定することをお勧めします。

チューニング

HTTP リスナーのアクセプタスレッドの数を設定する方法は、次のとおりです。

- server.xml ファイルを編集します。
- Web ベースの管理インタフェースで HTTP リスナーのノードを選択します。

最大同時要求数

Web サーバーが処理できる同時トランザクションの最大数は、init.conf ファイルの RqThrottle パラメータによって決定されます。デフォルト値は 128 です。この値を変更してサーバー側の処理可能数を絞り込み、実行するトランザクションの待ち時間を最小化することができます。RqThrottle の値は複数の仮想サーバーにまたがって適用されますが、負荷の分散 (ロードバランス) は行われません。

同時要求の数を計算するために、有効な要求の数を追跡し、新しい要求を受信するたびに 1 を加え、要求を完了するたびに 1 を差し引きます。新しい要求を受信すると、サーバーは処理中の要求数が最大値に到達していないかどうかをチェックします。上限に到達している場合、有効な要求の数が最大値を下回るまで、サーバーは新しい要求を保留します。

理論的には、最大同時要求数を 1 に設定してもサーバーは機能します。この値を 1 に設定すると、サーバーが同時に処理できる要求の数は 1 つに限定されますが、静的なファイルに対する HTTP 要求は一般に存続時間が短く (応答時間が 5 ミリ秒程度のこともあります)、一度に処理できる要求が 1 つだけでも 1 秒間に 200 程度までの要求に対応できます。

しかし実際は、インターネットクライアントはサーバーに接続し、要求を完了しないことがよくあります。この場合、サーバーはタイムアウトとなるまで 30 秒以上も待つことになります。このタイムアウトまでの時間は、`init.conf` の `AcceptTimeOut` 指示を使って設定できます。デフォルト値は 30 秒です。また、完了までに数分を要する大規模なトランザクションが必要なサイトもあります。これらの要因は、どちらも最大同時要求数に影響します。処理の完了までに何秒もかかる要求を数多く処理するサイトでは、最大同時要求数を高めに設定する必要があります。`AcceptTimeOut` の詳細については、「AcceptTimeOut 情報」を参照してください。

`RqThrottle` の値としては、負荷に応じて 100 ～ 500 が適しています。

`RqThrottleMin` は、起動時にサーバーが初期化するスレッドの最小数です。デフォルト値は 48 です。`RqThrottle` は同時に実行できるアクティブスレッドの最大数に対する物理的な制限であり、パフォーマンス上のボトルネックになることがあります。デフォルト値は 128 です。

注 再入に対応していない古い NSAPI プラグインは、ここで説明したマルチスレッドモデルをサポートしていることがあります。このようなプラグインの使用を継続するときは、再入に対応できるように新しいバージョンに変更します。それが不可能な場合は、`RqThrottle` を 1 に設定し、`MaxProcs` にたとえば 48 以上の大きな値を設定することで、サーバー側で対応します。これは、サーバーのパフォーマンスに悪影響を与えます。

チューニング

同時要求の数をチューニングする方法は、次のとおりです。

- `init.conf` ファイルの `RqThrottleMin` と `RqThrottle` を編集します。
- Web ベースの管理インタフェースで「プリファレンス」タブの「調整」ページにアクセスし、「最大同時接続」フィールドに値を入力します。

Java パフォーマンスの改善

Sun ONE Application Server では、さまざまな方法で Java パフォーマンスを向上できます。次のような方法があります。

- 代替スレッドライブラリの使用
- コンパイル済み JSP の使用
- クラスの再読み込み設定

代替スレッドライブラリの使用

Solaris 8 以降では、`libthread` または `/usr/lib/lwp` などの代替スレッドライブラリを使ってパフォーマンスを最適化できます。`libthread` は、デフォルトで有効にされています。

コンパイル済み JSP の使用

JSP のコンパイルはリソースを消費し、時間もかかります。サーバーにインストールする前に JSP をコンパイルしておくことで、パフォーマンスを向上できます。

クラスの再読み込み設定

パフォーマンスを向上するには、動的な再読み込みのフラグは無効に設定しておく必要があります。このように設定するには、`server.xml` を編集して `dynamic-reload-enabled="false"` に設定します。

init.conf のその他の指令

次に、サーバーの効率化に役立つ `init.conf` のその他の指示について説明します。

- AcceptTimeOut 情報
- CGIStub プロセス (UNIX/Linux)
- バッファサイズ

AcceptTimeOut 情報

クライアントとの接続を受け入れてから、情報を受信するまでのサーバーの待ち時間は、`AcceptTimeOut` を使って秒単位で設定します。デフォルトの設定は 30 秒です。ほとんどの状況では、この設定を変更する必要はありません。デフォルトの 30 秒より短く設定すると、スレッドを早く解放できます。ただし、接続速度が遅いユーザーの接続も切断してしまうことになります。

チューニング

AcceptTimeOut を設定する方法は、次のとおりです。

- init.conf の AcceptTimeOut パラメータを編集します。
- Web ベースの管理インタフェースのパフォーマンスサブメニューで値を変更します。

CGIStub プロセス (UNIX/Linux)

UNIX または Linux システムでは、CGIStub パラメータを調整できます。Sun ONE Application Server では、必要に応じて CGI エンジンが CGIStub を生成します。大きな負荷を処理し、CGI が生成するコンテンツに大きく依存するシステムでは、CGIStub プロセスがシステムのすべてのリソースを消費してしまうことも考えられます。このような状況が生じるサーバーでは CGIStub プロセスをチューニングして、新規生成される CGIStub プロセスの数、タイムアウト値、一度に実行できる CGIStub プロセスの最小値を設定できます。

注 init.conf ファイルに init-cgi 機能が設定され、マルチプロセスモードで実行している場合は、init-cgi 行に LateInit = yes を追加する必要があります。

CGI スタブの制御をチューニングする 4 種類の指示とそれぞれのデフォルト値は、次のとおりです。

- MinCGIStubs
- MaxCGIStubs
- CGIStubIdleTimeout
- CGIExpirationTimeout

MinCGIStubs は、デフォルトで起動するプロセス数を制御します。最初の CGIStub プロセスは CGI プログラムが呼び出されると起動します。デフォルト値は 2 です。init.conf ファイルに init-cgi 指令があれば、最小数の CGIStub プロセスが起動時に生成されます。

MaxCGIStubs は、サーバーが生成する CGIStub プロセスの最大値を制御します。これは CGIStub 実行時の最大同時プロセス数であり、保留された要求の最大数ではありません。デフォルト値はほとんどのシステムに適合します。この値を大きくしすぎると実際のスループットが減少します。デフォルト値は 10 です。

CGIStubIdleTimeout に設定された秒数の間アイドル状態にある CGIStub プロセスは、サーバーによって強制終了されます。プロセスの数が MinCGIStubs に達すると、サーバーはそれ以上のプロセスを終了しなくなります。デフォルト値は 45 です。

CGIExpirationTimeout は、CGI プロセスを実行できる最大時間を秒単位で制限します。

チューニング

CGIStub プロセスのすべての指令を設定する方法は、次のとおりです。

- init.conf ファイルを編集します。
- Web ベースの管理インタフェースの「詳細」タブにある CGI サブメニューで値を変更します。

バッファサイズ

サーバーソケットの送信バッファ (SndBufSize) と受信バッファ (RcvBufSize) のサイズを設定できます。これらのバッファについては、UNIX または Linux のマニュアルを参照してください。

チューニング

バッファサイズを設定する方法は、次のとおりです。

- init.conf の SndBufSize パラメータと RcvBufSize パラメータを編集します。
- Web ベースの管理インタフェースの「詳細」タブにある「パフォーマンス」サブメニューで SndBufSize と RcvBufSize の値を設定または変更します。

obj.conf のその他のパラメータ

obj.conf ファイルの一部のパラメータを使ってサーバーのパフォーマンスを向上できます。次に示す情報のほかに、「nocache パラメータの使用」も参照してください。

obj.conf の詳細は、『Sun ONE Application Server Developer's Guide to NSAPI』を参照してください。

find-pathinfo-forward

PathCheck 機能の find-pathinfo、および NameTrans 機能の pfx2dir と assign-name では、find-pathinfo-forward パラメータを設定してパフォーマンスを向上できます。このパラメータを設定すると、サーバーはサーバー機能 find-pathinfo のパスを最後から逆方向に検索せずに、パスの ntrans-base の後の PATH_INFO を順方向検索します。

注 サーバー機能 find-pathinfo を呼び出したときに、rq->vars に ntrans-base パラメータが設定されていない場合、サーバーは find-pathinfo-forward パラメータを無視します。デフォルトでは、ntrans-base は設定されています。

次に例を示します。

```
NameTrans fn="pfx2dir" find-pathinfo-forward="" from="/cgi-bin"
dir="/export/home/cgi-bin" name="cgi"

NameTrans fn="assign-name" from="/perf" find-pathinfo-forward=""
name="perf"
```

この機能を利用することで、サーバー機能 find-pathinfo で実行する stat が少なくなるため、特定の URL についてパフォーマンスを向上できます。Windows NT ではこの機能を使って、PathCheck サーバー機能 find-pathinfo を利用するときに、サーバーが「¥」を「/」に変更することを防ぐこともできます。

nostat

NameTrans 機能の assign-name に nostat パラメータを指定することで、指定した URL についてサーバーが stat を実行しないように設定できます。構文は次のとおりです。

```
nostat=virtual-path
```

次に例を示します。

```
<Object name=default>

NameTrans fn="assign-name" from="/nsfc" nostat="/nsfc"
name="nsfc"

</Object>

<Object name=nsfc>

Service fn=service-nsfc-dump

</Object>
```

上の例では、*ntrans-base* が設定されている場合に、パス /*ntrans-base*/nsfc と /*ntrans-base*/nsfc/* についてサーバーは stat を実行しません。*ntrans-base* が設定されていない場合、サーバーは URL /nsfc と /nsfc/* について stat を実行しません。デフォルトでは、*ntrans-base* は設定されています。この例は、デフォルトの PathCheck サーバー機能の利用を前提としています。

`assign-nameNameTrans` で `nostat=` *virtual-path* を指定すると、サーバーは指定の *virtual-path* で stat が失敗するものと判断します。このため、NSAPI プラグインの URL など、*virtual-path* のパスがシステムに存在しない場合にだけ `nostat` を使ってください。これらの URL に `nostat` を適用することで、不要な stat を回避できるため、パフォーマンスを向上できます。

サーバーのスケーリング

ここでは、サーバーのサブシステムと最適なパフォーマンスについて説明します。

- プロセッサ
- メモリ
- ディスク容量
- ネットワーク

プロセッサ

Solaris と Windows NT では、Sun ONE Application Server は複数 CPU の利点を透過的に活用できます。一般に、複数 CPU の効果はオペレーティングシステムの種類と作業負荷に応じて異なります。システムにプロセッサを追加することで、動的なコンテンツの処理パフォーマンスは向上します。静的なコンテンツは主に I/O に関連しており、CPU よりも I/O でより多くの時間が使われます。一次メモリを増やすほど多くのコンテンツがキャッシュされます (サーバーがメモリの有効利用に合わせてチューニングされている場合)。当社の測定では、4 CPU のマシンで CPU の数を 2 倍に増やすと、NSAPI で 40 〜 60% の改善、サーブレットで 50 〜 80% の改善が見られました。

メモリ

Solaris と Windows では、Sun ONE Application Server は 256 メガバイトの RAM を必要とします。これは、Sun ONE Studio を実行していないサーバーで稼働するアプリケーションサーバーに適用される値です。Sun Microsystems の Web サイトで、『Sun ONE Application Server インストールガイド』を参照してください。

ディスク容量

OS、ドキュメントツリー、ログファイルに適した十分なディスク容量が必要です。ほとんどの場合は、合計で 2 ギガバイトが必要となります。

OS、スワップファイルとページングファイル、Sun ONE Application Server のログファイル、ドキュメントツリーは、それぞれを別のハードディスクドライブに保存します。これにより、ログドライブでログファイルが一般になっても、OS に影響が及びません。また、たとえば、OS のページングファイルにドライブアクティビティを実行させるかどうかも指定できます。

スワップとページングにどれだけの容量を割り当てるべきか、OS のベンダーが推奨していることもあります。テスト結果によれば、RAM と同容量をスワップファイルの容量に割り当てると Sun ONE Application Server のパフォーマンスが最適化され、ドキュメントツリーのマッピングにも十分に対応できます。

ネットワーク

インターネットサイトでは、サーバーがピーク時に処理する同時アクセスユーザーの数を求め、その値にサイトの平均要求サイズを掛け合わせます。平均的な要求には、複数のドキュメントが関わることがあります。よくわからないときは、ホームページ、および関連するすべてのサブフレームと画像を適用してください。

次に、ピーク時のユーザーがドキュメントを待つ平均時間を秒単位で求めます。これでサーバーに必要な WAN 帯域幅を求めることができます。

たとえば、ピーク時のユーザー数が 50 で、平均ドキュメントサイズが 24 キロバイト、各ドキュメントの転送時間が 5 秒であれば、240 キロバイト (毎秒 1920 キロビット) が必要となります。この場合、2 本の T1 回線 (それぞれは毎秒 1544 キロビット) が必要となり、将来の成長に備えてオーバーヘッドも確保できます。

サーバーのネットワークインタフェースカードは、接続する WAN より大きな値に対応している必要があります。たとえば、最大で 3 本の T1 回線を利用するには、10BaseT インタフェースが必要です。1 本の T3 回線 (毎秒 45 メガビット) を利用するには、100BaseT が必要です。しかし、WAN の帯域幅が毎秒 50 メガビットを超える場合は、複数の 100BaseT インタフェースを利用するか、ギガビットイーサネットの採用を検討してください。

イントラネットでは、ネットワークがボトルネックとなることはほとんどありません。ただし、必要帯域幅の決定には前述と同じ計算方法を適用できます。

接続プールのチューニング

ここでは、JDBC 接続プールのチューニング方法について説明します。

データベースを多用するアプリケーションでは、Sun ONE Application Server が管理する JDBC 接続プールを最適なパフォーマンスに合わせてチューニングすることができます。この接続プールは、データベースとの有効な物理接続を数多く維持します。この接続を再利用することで、データベースとの接続を開閉するオーバーヘッドを減らすことができます。

JDBC リソースは、Sun ONE Application Server の設定ファイル `server.xml` の `<jdbc-resource>` 要素に定義され、`<jdbc-connection-pool>` を参照するように設定されます。J2EE アプリケーションは、JDBC リソースを使って JDBC 接続プールが維持する接続を取得します。複数の JDBC リソースが 1 つの JDBC 接続プールを参照することもできます。この場合、物理的な接続プールはすべてのリソースによって共有されます。

JDBC 接続プールは、Web ベースの管理コンソールを使うか、`server.xml` ファイルの `jdbc-connection-pool` 要素を編集することで定義および設定できます。定義した各プールはサーバーの起動時にインスタンス化されますが、物理的な接続が設定されるのは、最初にアクセスされたときです。

JDBC 接続プールの設定に使う属性は、次のとおりです。

表 : JDBC 接続プールの属性

属性名	説明
`name`	プール定義の一意の名前
`datasource-classname`	ベンダーから提供される JDBC データソースリソースマネージャの名前。XA またはグローバルトランザクションに対応したデータソースでは、`javax.sql.XADatasource` インタフェースが実装される。XA に対応していない、またはローカルトランザクション専用のデータソースでは、`javax.sql.Datasource` インタフェースが実装される
`res-type`	データソース実装クラスは、`javax.sql.DataSource` インタフェース、`javax.sql.XADataSource` インタフェース、または両方を実装できる。データソースクラスが両方のインタフェースを実装するときは、このオプション属性を指定してインタフェースを特定する必要がある。この属性に有効な値が指定されていても、指定したインタフェースがデータソースクラスによって実装されていない場合は、エラーが発生する。この属性にはデフォルト値はない
`steady-pool-size`	作成する接続の最小数 (初期作成数)
`max-pool-size`	作成できる接続の最大数

表 : JDBC 接続プールの属性 (続き)

max-wait-time-in-millis	接続がタイムアウトになる前に、呼び出し側が待つ時間。デフォルトは 60 秒。0 に設定すると、呼び出し側の待ち時間は無制限になる
pool-resize-quantity	idle-timeout-in-seconds timer が経過したときに削除する接続の数。タイムアウト値より長くアイドル状態であった接続が削除の対象となる。プールのサイズが steady-pool-size に達すると、接続削除プロセスは停止する
idle-timeout-in-seconds	プールで接続がアイドル状態のままでいられる最長時間。この時間を過ぎると、プールの実装はこの接続を非活性化できる。これは、データベースサーバー側で実行される接続タイムアウトを制御する属性ではない 管理者は、利用できない接続がアプリケーションサーバーに蓄積されないように、データベースサーバー側のタイムアウト (特定ベンダーのデータベースにこのタイムアウト値が設定されている場合) より短い時間をこのタイムアウト時間に設定する
transaction-isolation-level	プールされているデータベース接続のトランザクション分離レベルを指定する。この設定はオプションであり、デフォルト値はない。 指定しない場合は、JDBC ドライバによって設定されるデフォルトの分離レベルがプールに適用される 分離レベルを設定するときは、標準のトランザクション分離レベルである read-uncommitted、read-committed、repeatable-read、または serializable を指定する
is-isolation-level-guaranteed	transaction-isolation-level に特定の分離レベルを指定した場合にだけ適用される。デフォルト値は true これにより、プールから接続を取得するたびに、指定した分離レベルが設定される 一部の JDBC ドライバでは、この設定はパフォーマンスに影響する。接続を返す前にアプリケーションが分離レベルを変更しない場合、管理者はこの値を false に設定できる
is-connection-validation-required	true に設定すると、アプリケーションに渡す前に接続が評価され、その接続が利用可能であることが確認される。また、connection-validation-type は実行する検証の種類も指定する。デフォルトは false。サポートしている評価は次のとおり 1) connection.autoCommit() の使用 2) connection.getMetaData() の使用 3) ユーザーが指定したテーブル (validation-table-name を参照) へのクエリの実行 auto-commit または meta-data のいずれかを値として指定できる クエリによる接続の評価では、テーブルの属性 validation-table-name を使ってテーブル名を指定する。テーブルに connection-validation-type が設定されている場合、このパラメータの指定は必須である。特に setAutoCommit() および getMetaData() に対する呼び出しをデータベースドライバがキャッシュする場合は、接続を評価するためにユーザー指定のテーブルにアクセスする必要がある

表 : JDBC 接続プールの属性 (続き)

fail-all-connections:	1 回の評価チェックが失敗した場合に、プールされているすべての接続を閉じるかどうかを指定する。デフォルトは false。失敗した接続を確立し直すために、1 回の接続が試みられる

JDBC 接続プールのチューニング

JDBC 接続プールのチューニングでは、次の処理をお勧めします。

- 別の分離レベルを指定してもアプリケーションが正常に動作し、その分離レベルのほうがパフォーマンスの向上に適していることが確実でない限り、`setTransactionIsolationLevel()` を呼び出さずにドライバのデフォルトの分離レベルを使います。
- アイドル状態によるタイムアウトの値を 0 秒に設定します。この指示によって、アイドル状態の接続が一切削除されなくなります。これにより、通常は接続を新規作成するときにリソースが消費されなくなり、アイドル監視スレッドが無効化されます。ただし、長期間使われなかった接続は、データベースサーバーによってリセットされる可能性があります。
- 接続プールのサイズを変更するときは、次の長所と短所を検討する必要があります。

表 : 接続プールのサイズ変更に関する長所と短所

接続プール	長所	短所
小規模な接続プール	• 接続テーブルで高速にアクセスできる	• 要求に見合うだけの接続数を確保できない • ほとんどの接続で、キューに入れられる時間が長くなる
大規模な接続プール	• 要求に備えてより多くの接続を確保できる • キューに入れられる時間が少なくなる (またはなくなる)	• 接続テーブルでのアクセス速度が低下する

- `max-wait-time-in-millis` を 0 に設定します。これにより、接続を利用できるようになるまでクライアントスレッドがブロックされます。また、要求ごとに待ち時間の経過を追跡するタスクが軽減されるので、パフォーマンスが向上します。

- すでに説明したように、transaction-isolation-level の設定を変更しないようにします。変更が必要なときは、is-isolation-level-guaranteed フラグを false に設定し、アプリケーションがプログラムによって接続の分離レベルを変更しないようにします。

- is-connection-validation-required パラメータを true に設定すると、プールから接続が返されるたびに接続評価アルゴリズムの適用が実施されます。これにより、getConnection() の反応時間にオーバーヘッドが追加されます。データベースとの接続が信頼できるものであれば、評価を行う必要はありません。

次の図は、管理インタフェースの接続プール設定画面を示しています。

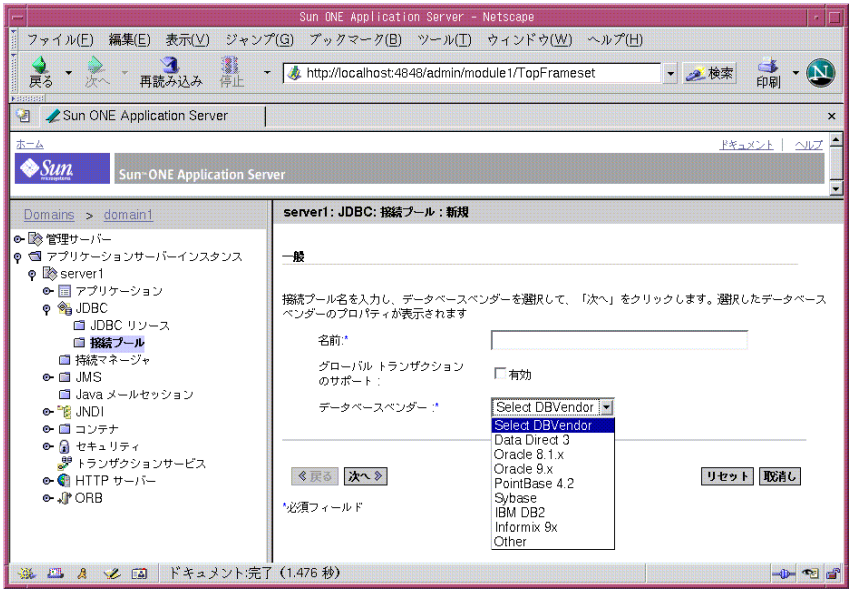


図 : 管理インタフェースによる JDBC 接続プールのチューニング

JSP とサーブレットのチューニング

ここでは、コード作成に関する注意点と、Sun ONE Application Server の設定に関する注意点を示し、JSP アプリケーションとサーブレットアプリケーションのチューニング方法について説明します。

JSP とサーブレットのコード作成に関する注意点

JSP アプリケーションおよびサーブレットアプリケーションのコードを記述するときは、次の点に注意することをお勧めします。

1. 規模の大きなオブジェクトを `HttpSession` 変数として格納しません。
2. 不要になった HTTP セッションを解放するときは、`javax.servlet.http.HttpSession.invalidate()` を使います。
3. HTTP セッションが必要なくなった場合に自動作成を停止するには、`<%page session="false"%>` 指示を使います。
4. サーブレットでの Java 同期を最小化します。
5. サーブレットではシングルスレッドモデルを使いません。
6. 多くのリソースを消費する 1 回の初期化では、サーブレットの `init()` メソッドを使います。
7. `System.out.println()` 呼び出しを使わないようにします。

JSP とサーブレットのパフォーマンスに影響する設定

次の設定によってパフォーマンスが向上します。これらの設定が本稼働環境を前提としていることに注意してください。一部の設定は開発中の JSP やサーブレットには効果がありません。

8. サーバーの `CLASSPATH` 設定から余分なディレクトリを取り除くことで、クラスのロード時間が短縮されます。関連するクラスは、JAR ファイルにまとめてください。
9. HTTP の設定 (接続とキープアライブサブシステムの設定) では、キープアライブサブシステムと HTTP サーバーの一般的なチューニングによって応答時間が変化します。詳細については、「HTTP サーバーのチューニング」を参照してください。
10. 再コンパイルと再読み込みの間隔を -1 に設定して、JSP が再コンパイルされないようにします。

11. SSL には mtmalloc を使います。このライブラリに含まれる機能には、ヒープ領域への同時アクセスを提供する malloc ルーチンのセットが用意されています。libmtmalloc 用のパッチは、http://www.sunsolve で入手できます。該当するサーバーインスタンスの /bin/startserv にある startserv スクリプトを編集し、LD_LIBRARY_PATH に so ファイルの場所の指定を追加します。
12. JSP とサーブレットのキャッシングを設定します。詳細は、『Sun ONE Application Server 7 Web アプリケーション開発者ガイド』の「サーブレットの使用」という章にある「キャッシュ機能」を参照してください。
13. EJB を含まないアプリケーションは、EAR ファイルではなく、WAR ファイルとして配備します。
14. セキュリティマネージャは多くのリソースを消費します。これは、必要なリソースへのすべての呼び出しが doPrivileged() メソッドを呼び出すためです。また、問題となったリソースは server.policy ファイルと照合されます。アプリケーションに server.policy を適用する意味がなく、サーバーで不正なコードが実行されないことが確実であれば、server.xml の該当行をコメントアウトして server.policy を無効化することがあります。

    たとえば、次のように記述することで server.policy をコメントアウトできます。

    ```
    <!-- jvm-options>
    -Djava.security.policy=/export/home/software/ias70_gold3/domains
    /domain1/server1/config/server.policy

    </jvm-options -->
    ```

EJB のパフォーマンスチューニング

Sun ONE Application Server の高性能な EJB コンテナには、パフォーマンスをチューニングするための多数の設定可能要素 (それぞれにデフォルト値があります) が用意されており、`server.xml` 設定ファイルと各 Bean の配備記述子で変更できます。`server.xml` に設定されている値は、個々の Bean の配備記述子に設定されている場合を除き、すべての EJB に適用されます。Bean の配備記述子に設定したプロパティは、常に `server.xml` の設定に優先して適用されます。`server.xml` ファイルの `<ejb-container>` 要素の詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

EJB 2.0 配備記述子に含まれる一部のプロパティは、チューニングにも適しています。`server.xml` ファイルの `<ejb-container>` 要素のデフォルト設定は、シングルプロセッサのコンピュータシステムに合わせて設定されています。コンテナから最大の性能を引き出すには、デフォルト設定を変更する必要があります。チューニングによって、次の効果を得られます。

- スピード : できるだけ多くの Bean を EJB キャッシュにキャッシュすることで、応答時間を簡単に短縮できます。これにより、CPU を多用するいくつかの処理を回避できます (後述します)。ただし、キャッシュを大きくしすぎると、キャッシュを管理するタスク (ガベージコレクションを含む) に時間がかかるようになるため、リソースとして利用できるメモリには限りがあります。
- メモリの消費 : プール内またはキャッシュされている Bean は、Java 仮想マシンのヒープのメモリを消費します。プールやキャッシュのサイズを大きくしすぎると、ガベージコレクションに要する時間と頻度が増えるため、パフォーマンスは低下します。
- 機能のプロパティ (ユーザータイムアウト、コミットオプション、セキュリティオプション、トランザクションオプションなど): これらのプロパティのほとんどは、J2EE サーバー内のアプリケーションの機能と設定に関連します。パフォーマンスのために機能面で妥協することはお勧めできませんが、このような状況が生じた場合の対応方法について選択肢を提供することはできます。

EJB コンテナのパフォーマンスチューニング

各種 Bean のライフサイクルは、EJB 仕様に正式に定義されています。このマニュアルでは、読者が Bean のライフサイクルイベントに習熟していることを前提とします。コンテナ内のアクティブ Bean は、要求を処理し、パフォーマンスを向上するためにキャッシュまたはプールに格納されます。パフォーマンスのチューニングでは、キャッシュとプールのプロパティをチューニングすることが重要です。

Bean の種類によっては、推奨されるチューニング方法を特定のコンテナに適用できないことがあります。

設定可能要素の使用について

次の表は、キャッシュとプールの設定に使われる設定可能要素を EJB の種類別に示しています。

表 : EJB のキャッシュとプールの設定に使う設定可能要素

	キャッシュの設定可能要素						プールの設定可能要素			
Bean の種類	cache-resize-quantity	max-cache-size	cache-idle-timeout-in-seconds	removal-timeout-in-seconds	victim-selection-policy	refresh-period-in-seconds	steady-pool-size	pool-resize-quantity	max-pool-size	pool-idle-timeout-in-seconds
ステートフルセッション	X	X	X	X	X					
ステートレスセッション							X	X	X	X
エンティティ (BMP/CMP)	X	X	X	X	X		X	X	X	X
エンティティ (BMP) 読み取り専用	X	X	X	X	X	X	X	X	X	X
メッセージ駆動型 Bean								X	X	X

EJB 記述子のプロパティ

EJB コンテナに含まれる各 Bean の設定には、次のプロパティを利用できます。

- steady-pool-size: Bean の初期数と最小数を指定します。steady-pool-size は、プールで保持する必要があります。指定できる値は、0 から MAX_INTEGER の範囲です。この値は、サーバーインスタンスレベルで指定します。Bean 固有の設定にも同じ名前のプロパティを使います。この変数も含め、次のすべての変数では、Bean レベルで値が設定されている (sun-ejb-jar.xml に指定) 場合は、Bean レベルで指定した値が適用されます。
- pool-resize-quantity: resize-quantity は、サーバーがプールを処理するときに、作成または削除する Bean の数を指定します。有効な値は 0 から MAX_INTEGER の範囲内で、上限値の適用対象となります。デフォルト値は 16 です。sun-ejb-jar.xml で対応する属性は resize-quantity です。
- max-pool-size: max-pool-size は、プールの最大サイズを指定します。指定できる値は、0 から MAX_INTEGER の範囲です。デフォルト値は 64 です。0 を指定すると、プールサイズは無制限になります。この場合、JVM ヒープがプールに格納するオブジェクトでいっぱいになることが考えられます。sun-ejb-jar.xml で対応する属性は、<bean-pool> 要素の max-pool-size です。
- max-wait-time-in-millis (現在は使われません)
- pool-idle-timeout-in-seconds: pool-idle-timeout-in-seconds は、ステートレスセッション Bean やメッセージ駆動型 Bean がアイドル状態でプール内に存在できる最大時間を指定します。この時間を過ぎると、ステートレスセッション Bean または メッセージ駆動型 Bean は削除されます。この時間は、サーバーが参考にします。pool-idle-timeout-in-seconds のデフォルト値は 600 秒です。sun-ejb-jar.xml で対応する属性は、<bean-pool> 要素の pool-idle-timeout-in-seconds です。
- cache-resize-quantity: cache-resize-quantity は、サーバーがキャッシュを処理するときに、作成または削除する Bean の数を指定します。有効な値は 0 から MAX_INTEGER の範囲内で、上限値の適用対象となります。sun-ejb-jar.xml で対応する属性は、<bean-cache> 要素の resize-quantity です。
- max-cache-size: max-cache-size は、キャッシュに入れることができる Bean の最大数を指定します。1 より大きな値を設定する必要があります。デフォルト値は 512 です。0 を設定すると、キャッシュできる Bean の数は無制限になります。この場合、キャッシュサイズは cache-idle-timeout-in-seconds と cache-resize-quantity によって制限されます。sun-ejb-jar.xml で対応する属性は、<bean-cache> 要素の max-cache-size です。

- cache-idle-timeout-in-seconds: cache-idle-timeout-in-seconds は、ステートフルセッション Bean やエンティティ Bean がアイドル状態でキャッシュ内に存在できる最大時間を指定します。この時間を過ぎると、Bean はバックアップストアで非活性化されます。cache-idle-timeout-in-seconds のデフォルト値は 600 秒です。sun-ejb-jar.xml で対応する属性は、<bean-cache> 要素の cache-idle-timeout-in-seconds です。
- is-cache-overflow-allowed (現在は使われません)
- removal-timeout-in-seconds: ステートフル Bean が非活性化される (バックアップストアでアイドル状態になっている) 時間は、removal-timeout-in-seconds パラメータによって制御されます。removal-timeout-in-seconds の時間を過ぎてもアクセスされなかった場合、Bean はバックアップストアから削除され、クライアントからアクセスできなくなります。removal-timeout-in-seconds のデフォルト値は 60 分です。sun-ejb-jar.xml で対応する属性は、<bean-cache> 要素の removal-timeout-in-seconds です。
- victim-selection-policy: victim-selection-policy は、ステートフルセッション Bean のキャッシュから削除対象を特定するアルゴリズムを指定します。設定できる値は、FIFO | LRU | NRU です。デフォルトは NRU (pseudo-random) です。sun-ejb-jar.xml で対応する属性は、<bean-cache> 要素の victim-selection-policy です。
- commit-option: 有効な値は B または C です。デフォルトは B です。この設定は、EJB 固有のトランザクションの commit-option に反映されます。
- refresh-period-in-seconds: (BMP の読み取り専用 Bean のみ) refresh-period-in-seconds は、読み取り専用 Bean をデータソースに基づいて更新する頻度を指定します。0 (更新しない) または正数 (指定した間隔で更新する) を指定できます。デフォルトは 600 秒です。

EJB プールのチューニング

プール内の Bean の状態は、EJB のライフサイクルでは「プールされた状態」となります。つまり、この状態の Bean には固有の情報が含まれません。Bean をプールに保持する利点は、要求を受信したときに Bean を最初から作成する時間を節約できることです。コンテナには、プールオブジェクトをバックグラウンドで作成し、要求されたパスに Bean を作成するための時間を節約するメカニズムが用意されています。

システムの通常の負荷に合わせて steady-pool-size に値を設定します。steady-pool-size には 0 より大きい値を設定することをお勧めします。これにより、受信した要求を処理するインスタンスが常にプール内に保持されます。

max-pool-size には、システムで予想される高負荷に合わせて値を設定します。プールサイズを大きくしすぎると、メモリが無駄になり、システムの処理速度が低下することがあります。接続の効率を考えると、プールサイズを小さくしすぎることも問題です。

<max-pool-size> の値を変更するときは、<pool-resize-quantity> の設定を再確認することをお勧めします。この値は、定期的なクリーニングで回収される Bean の数に対応します。最大サイズを増やす場合は、それに応じて回収する Bean の数も増やす必要があります。

その他の設定可能要素では、<pool-idle-timeout-in-seconds> の値が重要です。プール内に <steady-pool-size> を超える数の Bean が存在する場合、Bean の数が <steady-pool-size> に戻るまで、<pool-idle-timeout-in-seconds> 秒ごとに <pool-resize-quantity> の数だけ Bean が削除されます。
pool-idle-timeout-in-seconds の値が大きく、pool-resize-quantity の値が小さすぎる場合、Bean の数が pool-resize-quantity に戻るまでの時間が長くかかることになります。これを事前に理解しておくか、設定を修正する必要があります。

次の図は、サーバーインスタンスの EJB プールをチューニングするための管理インタフェースの画面を示しています。

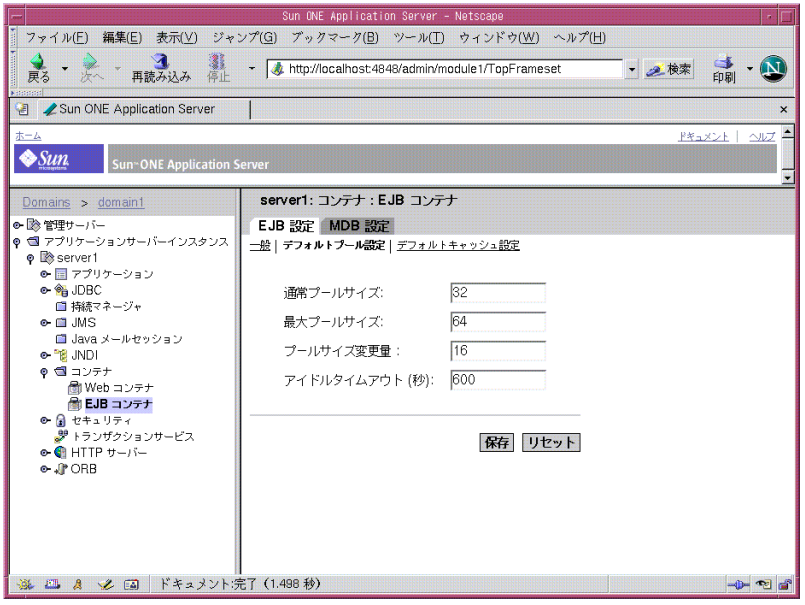


図 : 管理インタフェースによる EJB プールのチューニング

EJB キャッシュのチューニング

キャッシュ内の Bean の状態は、EJB のライフサイクルでは「使用可能状態」となります。つまり、この状態の Bean には関連する固有の情報 (主キー、セッション ID など) が含まれます。キャッシュから取り出した Bean は、EJB のライフサイクルに従って非活性化または削除する必要があります。非活性化された Bean をキャッシュに戻すには、活性化し直す必要があります。通常、エンティティ Bean はデータベースに格納され、なんらかのクエリ言語セマンティクスを使ってデータをロードまたは格納します。セッション Bean を非活性化してディスクまたはデータベースに格納するときは直列化する必要があり、活性化するときは直列化を復元する必要があります。

受信した要求が、キャッシュに入れられた「使用可能状態」の Bean を使用する場合、Bean の作成、固有情報の設定、活性化のオーバーヘッドが回避されます。このため、理論上はできるだけ多くの Bean をキャッシュするべきです。しかし、大規模なキャッシュには次のような問題があります。

- すべての Bean が消費するメモリは、仮想マシンが利用できるヒープのサイズに影響します。
- キャッシュに必要なオブジェクトとメモリが増えると、ガベージコレクションにかかる時間が長くなり、頻度も増す可能性があります。
- アプリケーションサーバーがメモリ不足になる可能性があります (ピーク負荷に合わせてヒープが慎重にチューニングされている場合を除く)

`<cache-idle-timeout-in-seconds>` を過ぎてもキャッシュに残っているすべての Bean は、定期的なクリーニングで削除されます。

次の図は、コンテナ全体の EJB キャッシュをチューニングするための管理インタフェースの画面を示しています。

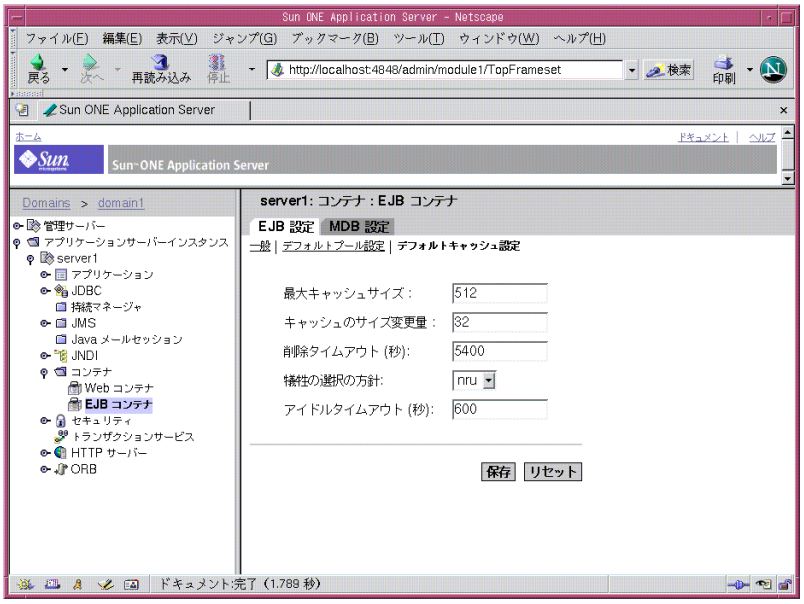


図 : 管理インタフェースによる EJB キャッシュのチューニング

各種 EJB のパフォーマンスに関する注意点

次の図は、プールとキャッシュを個々の Bean に設定する Bean 記述子の例を示しています。

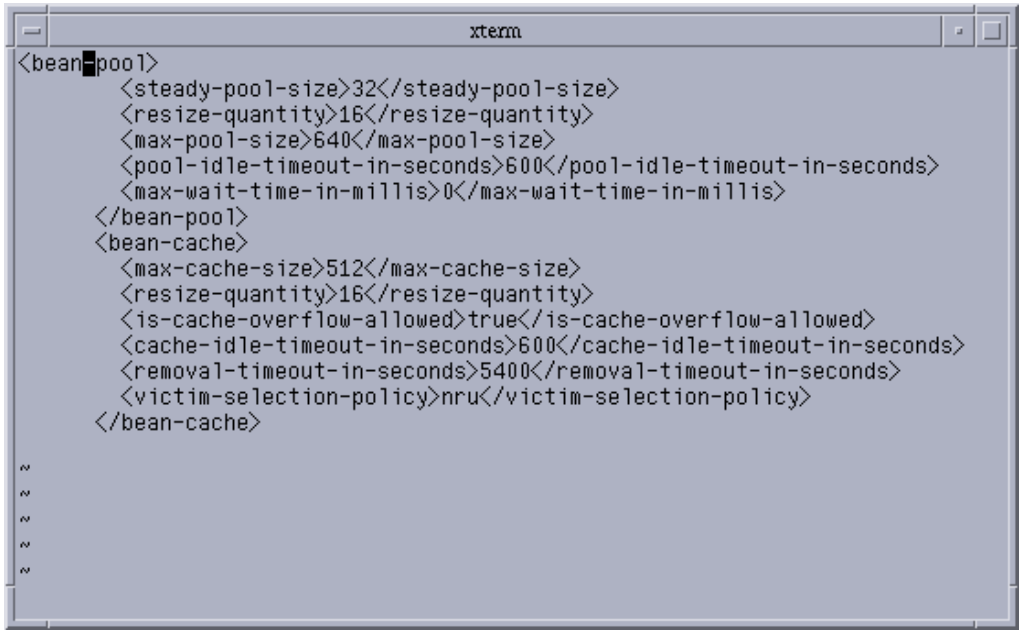


図 : 個々の Bean の Bean 記述子

次に、パフォーマンスに関する注意点を Bean の種類別に示します。

- **エンティティ Bean**: 特定のエンティティ Bean の利用状況によっては、あまり使われない Bean (作成後、トランザクションが終了すると使われなくなる注文など) がキャッシュされる回数を減らし、頻繁に使われる Bean (頻繁に参照される在庫項目など) がキャッシュされる回数を増やせるように、max-cache-size を設定する必要があります。エンティティコンテナをチューニングするその他の方法については、「コミットオプション」を参照してください。
- **ステートフルセッション Bean**: ステートフル Bean がユーザーを表す場合は、アプリケーションサーバープロセスの予想並行ユーザー数に合わせて Bean の `max-cache-size` を設定します。通常のユーザー負荷から見て小さすぎる値を設定すると、Bean の非活性化と活性化が頻繁に行われ、CPU を多用する直列化と直列化復元およびディスク I/O が増えるため、応答時間に悪影響を生じます。もう一つの重要な設定可能要素は `cache-idle-timeout-in-seconds` です。`cache-idle-timeout-in-seconds` を過ぎてもアクセスされなかったキャッ

シュ内のすべての Bean は、cache-idle-timeout-in-seconds の間隔で非活性化されます。HTTP セッションの非活性化とアイドルのタイムアウトと同様に、removal-timeout-in-seconds を過ぎてもアクセスされなかった Bean は削除されます。非活性化された Bean は直列化され、ディスクに格納されます。多数の Bean を非活性化することは、ディスクシステムのファイル数が増えるだけでなく、呼び出す前にセッションの状態を直列化復元しなければならないため、応答時間が遅くなります。

- **ステートレスセッション Bean**: ステートレスセッション Bean には、関連する状態がなく、エンティティ Bean やステートフルセッション Bean より利用しやすい状態でプールされます。この Bean を最適にチューニングするには、設定可能要素 steady-pool-size、pool-resize-quantity、max-pool-size に適切な値を設定します。事前の設定をプールに含めておく場合は、steady-pool-size に 0 より大きな値を設定します。これにより、コンテナが起動されると、steady-pool-size で指定した数の Bean を持つプールが作成されます。プールに事前に Bean を用意しておくことで、メソッドを呼び出すときにオブジェクトを作成する時間を節約できます。steady-pool size に大きすぎる値を設定すると、メモリの消費が必要以上に多くなり、GC にかかる時間が長くなります。pool-resize-quantity は、プールの増大だけでなく縮小にも影響します。縮小は指数関数的に行われるため、小さな値を設定することをお勧めします。max-pool-size に小さな値を設定すると、現在のプールサイズが max-pool-size を超える場合にプールからインスタンスが削除されるため、余分なオブジェクト削除が行われることになります (これは、余分なオブジェクト作成を行った結果です)。
- **読み取り専用エンティティ Bean**: データベースを更新することのない通常の BMP Bean を使うときは、代わりに読み取り専用 Bean を使ってみてください。デフォルトでは、その Bean が示すデータベース行がバックグラウンドで変更されない場合にだけ読み取り専用 Bean は正常に機能します。Sun ONE Application Server では、読み取り専用 Bean は Bean 管理持続 (BMP) だけでサポートされています。読み取り専用 Bean がデータベースを更新することはありません (ejbStore が呼び出されることはありません)。このため、BMP Bean と読み取り専用 Bean がパフォーマンスに与える影響には大きな違いがあります。トランザクションを使って読み取り専用 Bean のメソッドにアクセスすると (ejb-jar.xml に指定されたメソッド記述子が TX_REQUIRED または TX_REQUIRES_NEW であるため)、ejbLoad() が常に呼び出されます。キャッシュされたデータをデータベースと常に同期させるには、トランザクションを使って読み取り専用 Bean のメソッドにアクセスします。読み取り専用 Bean は ejbLoad() によるオーバーヘッドを回避しようとするため、これはトランザクションアクセスとはなりません。この場合に重要な設定可能要素は、refresh-period-in-seconds です。これらの Bean のコンテナの現在の実装はプルキャッシュです。つまり、どの Bean インスタンスでも refresh-period-in-seconds を経過した場合、この Bean インスタンスにアクセスするユーザーはビジネスメソッドを実行する前に Bean で ejbLoad() を呼び出す必要があります。データが変更される時間を想定し、これに適切な値を入力します。通常は数分間です。たとえば、株価情報を 5 分間保持するときは、refresh-period-in-seconds を 300 秒に設定します。

注 読み取り専用 Bean がキャッシュしたデータを更新する通常の BMP Bean があるときは、プログラムによる更新機能の利用を検討します。詳細は、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

注 Sun ONE Studio を使って Bean を開発し、アプリケーションサーバーに配備したときは、Bean プールと Bean キャッシュについて、個々の Bean の記述子を編集する必要があります。この設定は、本稼働環境レベルの配備には適さないことがあります。

- **メッセージ駆動 Bean**: これらの Bean のコンテナ、エンティティ Bean やセッション Bean のコンテナとは若干異なります。現在のバージョンのアプリケーションサーバーの MDB コンテナでは、MDB プール内の Bean にセッションとスレッドが関連づけられています。これは、コンテナ内のメッセージ駆動型の要求を実行するスレッドをプールする場合に適しています。このため、Bean プールには、他のアプリケーションも含めてサーバーのすべてのパラメータを考慮した値を設定する必要があります。たとえば、500 を超える値は適していません。

次の図は、メッセージ駆動型 Bean のプールをチューニングするための管理インタフェースの画面を示しています。

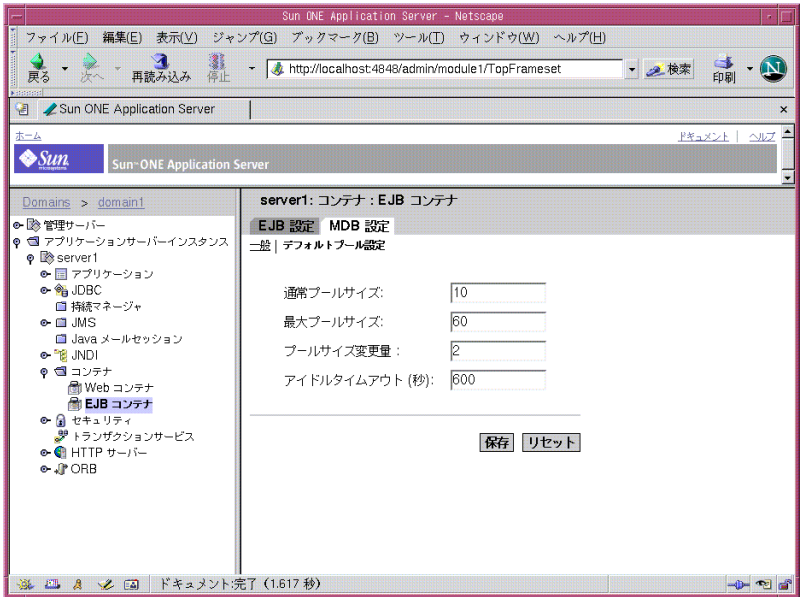


図 : 管理インタフェースによるメッセージ駆動型 Bean コンテナの設定

関連する注意点

次に、EJB の使用に関連する注意点を示します。

- リモートインタフェースとローカルインタフェース

  EJB は、リモートインタフェースとローカルインタフェースを持つことができます。アプリケーションサーバーインスタンスと異なる場所にあるリモートクライアントは、Bean へのアクセスにリモートインタフェースを使います。リモートインタフェースへの呼び出しは、引数の操作、ネットワークによるデータの転送、受信側でのデータの操作と処理を必要とするため、より多くのリソースが必要となります。アプリケーションサーバーインスタンスと同じ場所にあるローカルクライアントは、Bean のローカルインタフェースがあれば、これを利用できます。引数の操作や転送が必要ないため、ローカルインタフェースを使った方が効率的です。引数は参照を使って渡されるので、渡されたオブジェクトを呼び出す側と呼び出される側の両方が共有できます (このためには、共有できるように適切に実装する必要があります)。同じ場所にあるクライアントだけが使用できるように Bean が記述されている場合、その Bean にローカルインタフェースを持たせ、クライアントにローカルインタフェースを利用させることができます。一方、場所

に関係なく利用できるように Bean が記述されている場合は、リモートクライアントはリモートインタフェースを使用し、ローカルクライアントはローカルインタフェースを使用するように、リモートインタフェースとローカルインタフェースの両方を持たせることができます。

- pass-by-value セマンティクスと pass-by-reference セマンティクス

  リモートインタフェースとローカルインタフェースを適切に使うことで、クライアントがアクセスする Bean を効率的に記述るようになります。しかし、次のような場合はローカルインタフェースを利用できません。

1. アプリケーションの仕様が EJB2.0 より古く、ローカルインタフェースの利用が記述されていない
2. Bean から Bean の呼び出しが含まれ、呼び出される Bean が同じ場所にあるという前提がクライアント Bean に記述されていない

   このような状況に対応できるように、Sun ONE Application Server 7.0 には、Bean のリモートインタフェースに呼び出しを行うときに、同じ場所にあるクライアントが引数を参照を使って渡せる pass-by-reference オプションが用意されています。デフォルトでは、Sun ONE Application Server は同じ場所にある Bean のリモートインタフェースを呼び出すときに pass-by-value セマンティクスを使います。pass-by-value セマンティクスでは、渡す前に引数のコピーが作成されるため、リソースの消費が多くなります。pass-by-reference オプションは、アプリケーション全体または Bean 単位で適用できます。アプリケーションレベルで適用すると、アプリケーションのすべての Bean のリモートインタフェースに引数を渡すときに、pass-by-value セマンティクスが利用されます。Bean レベルで適用した場合は、その Bean のリモートインタフェースへの呼び出しだけで pass-by-value セマンティクスが利用されます。pass-by-value フラグの詳細については、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』および『Sun ONE Application Server 開発者ガイド』を参照してください。

- トランザクション分離レベル

  トランザクションに関するアプリケーションセマンティックを明確にすることで、パフォーマンスを最適にする分離レベルを決定できるようになります。次に、パフォーマンスの向上に適しているものから順にトランザクション分離レベルを示します。

  a. `READ_UNCOMMITTED`

  b. `READ_COMMITTED`

  c. `REPEATABLE_READ`

  d. `SERIALIZABLE`

  これらの値は、データベース接続プールの属性 (`jdbc-connection-pool`) として設定できます。

3. EJB コンテナから見た場合、次のトランザクション属性を指定することができます。パフォーマンスの向上に適しているものから順にオプションを示します。
   a. NEVER
   b. TX_NOTSUPPORTED
   c. TX_MANDATORY
   d. TX_SUPPORTS
   e. TX_REQUIRED
   f. TX_REQUIRESNEW

コミットオプション

コミットオプションは、Bean が関与するトランザクションが完了したときに、コンテナがその Bean に対して実行する処理を制御します。コミットオプションが Bean のコードを変更することはありません (Bean 開発者は、コミットオプションを考慮する必要はありません)。ただし、コミットオプションはパフォーマンスに大きく影響します。

Sun ONE Application Server は、コミットオプション B、C をサポートしています。

どのコミットオプションをいつ使うかについて説明する前に、各コミットオプションを使った場合のコンテナの動作について説明します。

コミットオプション B では、トランザクションが完了すると Bean はキャッシュに保持され、固有の情報も残されます。つまり、同じ主キーが次に呼び出されたときに、キャッシュに保持されているインスタンスをそのまま使えます。もちろん、データベースとの同期をとるメソッド呼び出しの前に、Bean の `ejbLoad` が呼び出されます。

コミットオプション C では、トランザクションが完了すると、Bean の `ejbPassivate()` メソッドが呼び出され、Bean と主キーとの関連づけを解除した上で、Bean は空きプールに戻されます。つまり、同じ主キーが次に呼び出されたときは、プールから空き Bean を取り出し、このインスタンスに `PrimaryKey` を設定した上で、そのインスタンスが `ejbActivate` を呼び出す必要があります。この場合も、データベースとの同期をとるメソッド呼び出しの前に、Bean の `ejbLoad` が呼び出されます。

コミットオプション B では、`ejbAcivate` と `ejbPassivate` は呼び出されません。`ejbActivate` と `ejbPassivate` の呼び出し、およびプールからオブジェクトの取得とプールへのオブジェクトの解放がないため、ほとんどの場合はコミットオプション C よりもコミットオプション B のほうがパフォーマンスには有利です。ただし、コミットオプション C のほうが有効な場合もあります。キャッシュに入れられた Bean の再利用がまれで、Bean がコンスタントにキャッシュに追加される場合は、Bean をキャッシュする意味がありません。

これはコミットオプション C の動作とまったく同じです。コミットオプション C を利用すると、メソッドの呼び出し後、またはトランザクションの完了後にコンテナは Bean をキャッシュせずに、プールに戻します。これにより、インスタンスはより効率的に再利用され、VM に生存するオブジェクトの数が減るために GC のサイクルも短くなります。

コミットオプション B と C のどちらを利用するかをどのように決定すべきでしょうか。まず、Bean の監視コマンドを使ってキャッシュのヒット率を確認します。キャッシュに残されていない回数より、キャッシュに残されている (ヒットする) 回数のほうが多い場合は、コミットオプション B のほうが適していると言えます。結果を最適化するには、更に `max-cache-size` と `cache-resize-quantity` の調整が必要です。キャッシュのヒット回数が少なく、キャッシュに残されていない回数が極端に多い場合は、アプリケーションは Bean インスタンスを効率よく再利用していないことになるので、キャッシュサイズを大きくしても (`max-cache-size` を使います) 効率の改善は望めません (アクセスパターンが変化しない場合)。この場合は、コミットオプション C を指定します。キャッシュに残されている回数と残されていない回数に大きな違いがない場合は、`max-cache-size` および `cache-idle-timeout-in-seconds` をチューニングする必要があります。

次の図は、コミットオプションの設定を示しています。

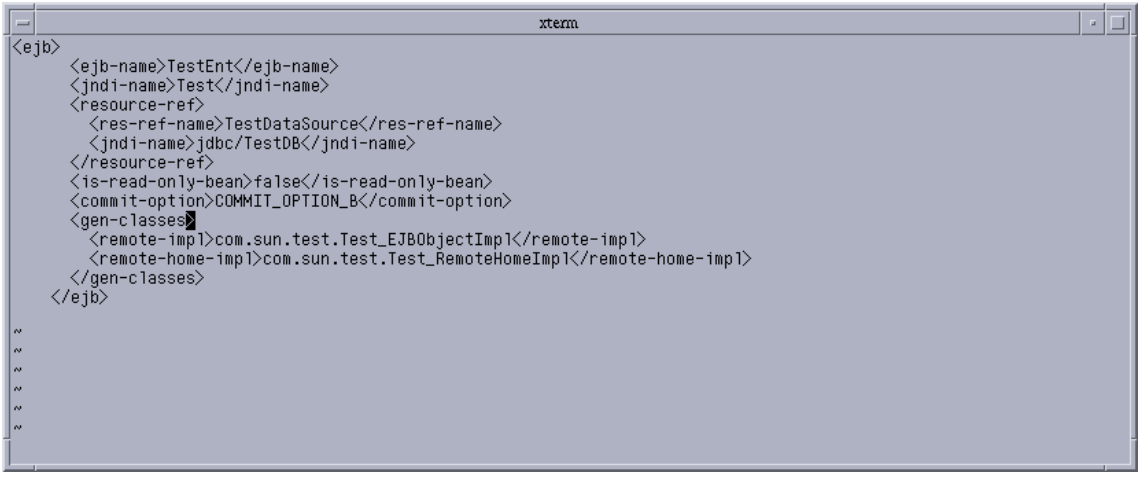


図 : コミットオプションの設定

EJB コンテナの監視を有効にしておくことで、いつでも個々の Bean の統計を調査、分析し、プールとキャッシュのチューニングに利用することができます。プールの設定は、ステートレスセッション Bean とエンティティ Bean に対して有効です。キャッシュの設定は、ステートフルセッション Bean とエンティティ Bean に対して有効で

す。server.xml ファイルのプロパティを設定することで、コンテナの設定をサーバーインスタンスレベルで変更できます。この設定は、sun-ejb-jar.xml に指定した各 Bean の設定によってオーバーライドされます。次の設定可能要素の説明は、「EJB 記述子のプロパティ」を参照してください。

サーバーインスタンスレベルで指定できる設定可能要素は、次のとおりです。

- steady-pool-size
- pool-resize-quantity
- max-pool-size
- cache-resize-quantity
- max-cache-size
- pool-idle-timeout-in-seconds
- cache-idle-timeout-in-seconds
- removal-timeout-in-seconds
- victim-selection-policy
- commit-option
- log-level
- monitoring-enabled

Bean レベルで指定できるプールの設定可能要素は、次のとおりです。

- steady-pool-size
- resize-quantity
- max-pool-size
- pool-idle-timeout-in-seconds

Bean レベルで指定できるキャッシュの設定可能要素は、次のとおりです。

- max-cache-size
- resize-quantity
- is-cache-overflow-allowed
- cache-idle-timeout-in-seconds
- removal-timeout-in-seconds
- victim-selection-policy

次の監視コマンドを実行すると、ステートフルセッション Bean のキャッシュ統計が出力されます。監視コマンドの出力例も合わせて示します。

```
$./asadmin get --user admin --password netscape --host e4800-241-a
--port 4848 -m
specjcmp.application.SPECjAppServer.ejb-module.supplier_jar.statefu
l-session-bean.BuyerSes.bean-cache.*

resize-quantity = -1

cache-misses = 0

idle-timeout-in-seconds = 0

num-passivations = 0

cache-hits = 59

num-passivation-errors = 0

total-beans-in-cache = 59

num-expired-sessions-removed = 0

max-beans-in-cache = 4096

num-passivation-success = 0
```

次の監視コマンドを実行すると、エンティティ Bean のプール統計が出力されます。

```
$./asadmin get --user admin --password netscape --host e4800-241-a
--port 4848 -m
specjcmp.application.SPECjAppServer.ejb-module.supplier_jar.statefu
l-entity-bean.ItemEnt.bean-pool.*

idle-timeout-in-seconds = 0

steady-pool-size = 0

total-beans-destroyed = 0

num-threads-waiting = 0

num-beans-in-pool = 54

max-pool-size = 2147483647

pool-resize-quantity = 0

total-beans-created = 255
```

次の監視コマンドを実行すると、ステートレス Bean のプール統計が出力されます。

```
$./asadmin get --user admin --password netscape --host e4800-241-a
--port 4848 -m
test.application.testEjbMon.ejb-module.slsb.stateless-session-bean.
slsb.bean-pool.*
```

```
idle-timeout-in-seconds = 200

steady-pool-size = 32

total-beans-destroyed = 12

num-threads-waiting = 0

num-beans-in-pool = 4

max-pool-size = 1024

pool-resize-quantity = 12

total-beans-created = 42
```

Bean をチューニングするには、対象となる Bean のキャッシュとプールの動作を記録し、時間とともにどのように変化するかを調べます。たとえば、次のような動作を観察します。

- 非活性化が頻繁に行われ、VM ヒープのサイズが小さく維持されている場合は、`max-cache-size` または `cache-idle-timeout-in-seconds` の値を大きくします。

GC が頻繁に行われ、プールサイズが大きくなり、それでもキャッシュのヒット率が上がらない場合は、`pool-idle-timeout-in-seconds` の値を小さくして削除されるインスタンスを増やします。

注　`max-pool-size` に 0 を指定すると、プールサイズに制限がなくなります。プールに入れられた Bean は、`pool-idle-timeout-in-seconds` に短い間隔を指定しない限り削除されません。本稼働環境のシステムでは、プールのサイズを無制限にすることはお勧めできません。

ORB のチューニング

Sun ONE Application Server には、高性能でスケーラブルな CORBA ORB (Object Request Broker) が用意されています。ORB は、サーバー上の EJB コンテナの基礎となります。ORB のほとんどの機能は、次の方法で Enterprise JavaBeans を実行したときに使用されます。

1. アプリケーションクライアントコンテナを使ったアプリケーションクライアント (またはリッチクライアント) からの RMI/IIOP パス
2. 別の Sun ONE Application Server インスタンス ORB からの RMI/IIOP パス
3. 別のベンダーの ORB からの RMI/IIOP パス
4. Web/MDB (メッセージ駆動型 Bean) コンテナからのプロセス内パス

サーバーインスタンスから別のサーバーインスタンス ORB への接続が作成されると、最初のインスタンスがクライアント側 ORB として動作します。IIOP を介した SSL では、最適化された (最も高速な) トランスポートを使い、暗号化アルゴリズムのネイティブ実装を利用することで優れた性能を維持します。

クライアントから ORB への接続方法

リッチクライアント Java プログラムは、クライアント側の ORB インスタンスを呼び出す `initialContext()` を新たに呼び出します。これは、Sun ONE Application Server の IIOP ポートへのソケット接続を作成します。このクライアントからの IIOP 要求を処理するサーバー ORB で、読み取りスレッドが開始されます。
`initialContext` を使った場合、クライアントコードはサーバーに配備された EJB をルックアップします。クライアントには、サーバーに配備された EJB へのリモート参照となる IOR が戻されます。このオブジェクト参照を使って、クライアントコードは EJB でリモートメソッドを呼び出します。

Bean の `InitialContext` ルックアップとメソッド呼び出しにより、Java で記述されたアプリケーション要求データの配列は IIOP メッセージに変換され、サーバー ORB が作成したソケット接続に送信されます。次に、サーバーは応答データを作成し、それを同じ接続に戻します。このデータの配列はクライアント ORB によって解除され、処理のためにクライアントコードに戻されます。リッチクライアントアプリケーションが終了すると、クライアント ORB は接続を終了し、閉じます。

ORB のパフォーマンスチューニング

高いパフォーマンスやスケーラビリティを得るために、デフォルト設定や、非標準オプションの追加が必要になることがあります。ORB のチューニングに利用できる主要コンポーネントは、次のとおりです。

- ORB 間通信インフラストラクチャ
- サーバー ORB スレッドプール

負荷のバランス、複数の共有接続、サーバースレッドプール、メッセージのフラグメントサイズをチューニングすることで、応答時間を短くすることができます。スケーラビリティは、複数の ORB サーバーを使ってクライアントからの負荷を均等に分散し、クライアントとサーバーの間の接続数をチューニングすることで得られます。

ORB の設定可能要素

ORB には、次の設定可能要素があります。

1. **ORB 間通信インフラストラクチャ** : このインフラストラクチャを利用することで、メッセージサイズ、ロードバランス (負荷が大きい場合)、高スループット、高パフォーマンスに適したチューニングを行えます。
2. **サーバー ORB スレッドプール** : ORB スレッドプールでは、値を設定して制御できるマルチスレッドを使って、必要な作業を迅速かつ同時に実行できます。スレッドをプールすることで、スレッドの作成、スレッドスタックの割り当て、スレッドに関連する GC などのオーバーヘッドを回避できます。場合によっては、スレッドの作成と削除が過剰に行われ、`OutOfMemoryError`, が生じることがあります。スレッドプールにしきい値を設定することで、これを回避できます。

ORB スレッドプールには、タスクキューのスレッドとプールのスレッドがあります。タスクはタスクキューに入れられ、空きスレッドはキューからタスクを取り出して実行します。タスクキューが常に空になるほどスレッドプールを大きくすることはお勧めできません。アプリケーションインスタンスの「現在のタスクキューのサイズ」と `max-thread-pool-size` の比率は、常に 1:10 程度となるのが普通です。スレッドプールには、`max-thread-pool-size` を `steady-thread-pool-size` (通常サイズ) より大きく設定していると、現在のサイズが通常サイズより大きくなったときに通常のサイズに戻す機能があります。`steady-thread-pool-size` には、標準的な (RMI/IIOP) 負荷に必要な平均スレッド数を設定します。

アプリケーションサーバーの現在のバージョンでは、ORB スレッドプールは主に次の 2 つの目的で使用されます。

1. すべての ORB 要求の実行
2. EJB プールおよびキャッシュの縮小

このため、リモート呼び出しに ORB を使わない (RMI/IIOP を経由しない) 場合でも、EJB プールとキャッシュのクリーニング機能を利用できるように、スレッドプールのサイズを設定しておく必要があります。

ORB のプロパティ

ORB をチューニングするプロパティは、管理インタフェースを使って管理できます。

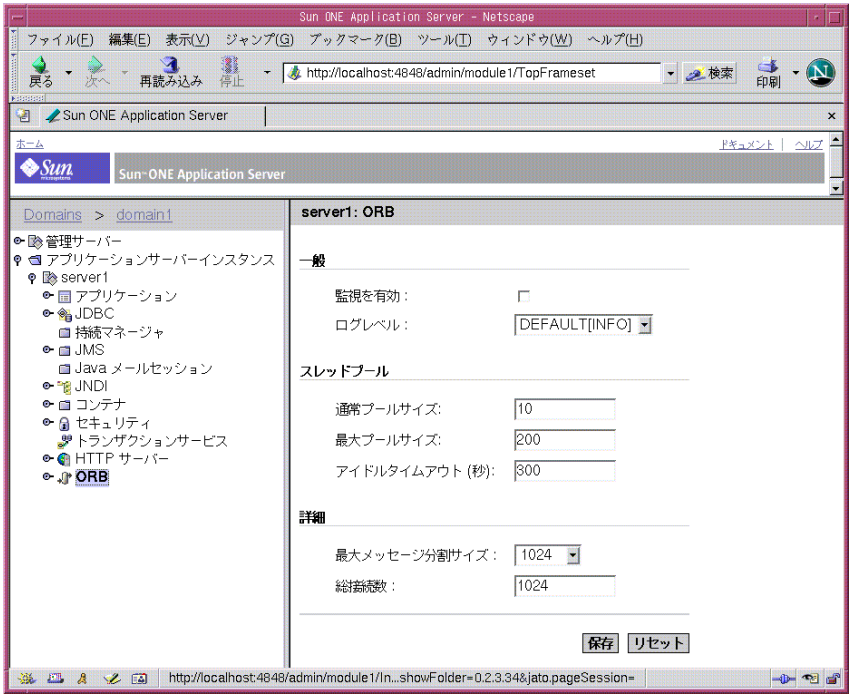


図 : 管理インタフェースによる ORB プロパティのチューニング

ORB のチューニングには、次の標準プロパティを利用できます。

- `message-fragment-size`: このサイズ (バイト単位) を超える CORBA GIOPv1.2 メッセージは分割されます。すべてのサイズは、8 の倍数で指定する必要があります。複数に分割できる GIOP v1.2 メッセージは、Request、Reply、LocateRequest、LocateReply です。最初のメッセージは通常の Request または Reply メッセージで、分割の可否を指定するフィールドは true に設定されています。ORB 間メッセージのほとんどがデフォルトサイズ (1024 バイト) より大きい場合は、このサイズを大きくすることで、分割によるネットワーク待ち時間を短縮できます。

- `steady-thread-pool-size`: ORB スレッドプールの最小スレッド数を指定します。サーバーが安定した状態 (通常の負荷状態) で要求 (および EJB のクリーニング) の処理に必要なアクティブスレッドの数に近い値を設定する必要があります。
- `max-thread-pool-size`: ORB スレッドプールの最大スレッド数を指定します。
- `idle-thread-timeout-in-seconds`: アイドル状態のスレッドがプールから削除されるまでのタイムアウト値を指定します。スレッドプールのサイズを縮小することができます。
- `max-connections`: すべてのリスナーで一度に受け付けることができる接続の最大数を指定します。接続数の上限を設けることで、サーバーの状態を保護します。この値は、接続からアクティブにデータを読み取るスレッドの最大数とも一致します。
- `iiop-listener`: このプロパティは、ORB にリスナー (SSL または HTTP) を追加し、リスナーのホストとポートを指定します。有効化された ORB リスナーは、サーバーソケットで受信する接続要求をアクティブに待機するスレッドに変換します。

次の図は、管理インタフェースの IIOP リスナーチューニング画面を示しています。

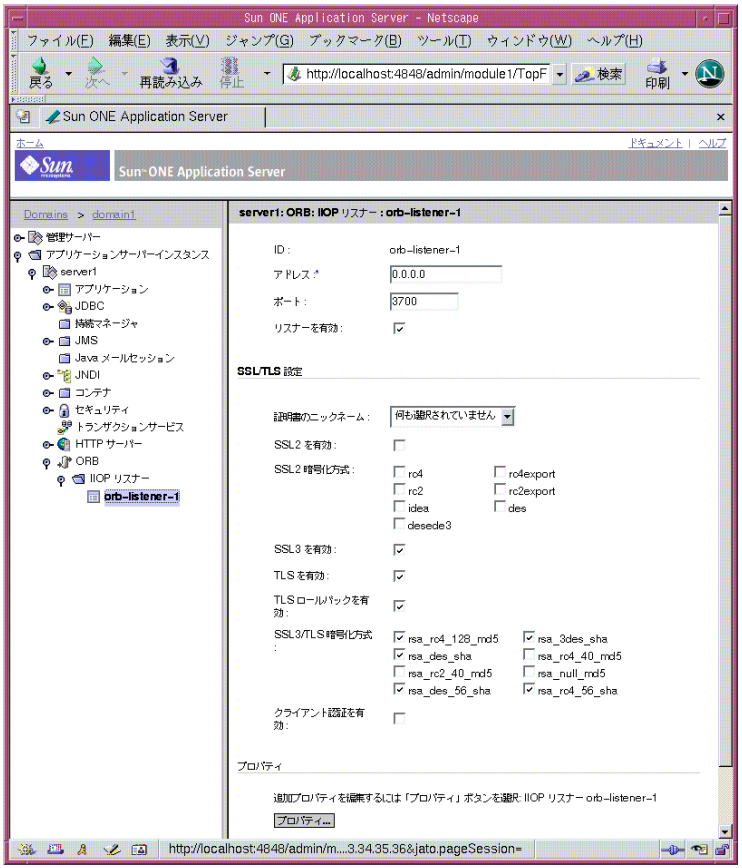


図 : 管理インタフェースによる HTTP リスナーのチューニング

標準外の ORB プロパティとその機能

次の値を指定するときは、クライアントプログラムを起動するときに引数 -D を指定します。

クライアントとサーバー ORB の間の通信の制御

クライアントでデフォルトの JDK ORB を使うときは、最初のコンテキストを作成するたびにクライアント ORB からアプリケーションサーバー ORB への接続が作成されます。これらの接続が同じプロセスで作成された場合に、これををプールする、または共有するときは、クライアント ORB の設定に次の行を追加します。

```
-Djava.naming.factory.initial=com.sun.appserv.naming.S1ASCtxFactory
```

複数接続によるスループットの改善

Sun ONE コンテキストファクトリ (`com.sun.appserv.naming.S1ASCtxFactory`) を使うときは、クライアント ORB からサーバーへの接続数を指定する設定可能要素も重要になります (デフォルト値は 1)。この機能を利用することで、ネットワーク上のアプリケーションインスタンスで、サーバーとの通信スループットが改善されます。クライアント ORB の設定を変更するときは、次の jvm オプションを追加します。

```
-Djava.naming.factory.initial=com.sun.appserv.naming.S1ASCtxFactory

-Dcom.sun.appserv.iiop.orbconnections=[number]
```

DNS の設定によるサーバー側の負荷のバランス

特別に設定した DNS を使うことで、1 つまたは複数のクライアント ORB が負荷を均等に分散できます。`nslookup` を呼び出すたびに、`nslookup` が特定ホストの IP アドレスのリストを利用して本来の負荷バランス機能を利用できるように、DNS にはホスト名のリストが設定されています。セクション 2.3.2 に指定されている接続プールを使って、利用する接続の数を指定することもできます。クライアント ORB の設定を変更するときは、次の jvm オプションを追加します。

```
-Djava.naming.factory.initial=com.sun.appserv.naming.S1ASCtxFactory

-Djava.naming.provider.url.pkgs=com.sun.enterprise.naming

-Djava.naming.provider.url=iiop://${SERVER_HOST}:${ORB_PORT}
```

クライアント側でのサーバーインスタンスの設定によるサーバー側のロードバランス

複数の ORB リスナー (または複数の独立した ORB プロセス) で単純なラウンドロビンスキームを利用することで、1 つまたは複数の ORB が負荷を均等に分散できます。制御された数のクライアントが RMI/IIOP パスを使ってサーバーに負荷を加える B2B 環境では、この設定をお勧めします。セクション 2.3.2 に指定されている接続プールを使って、利用する接続の数を指定することもできます。クライアント ORB の設定を変更するときは、次の jvm オプションを追加します。

```
-Djava.naming.factory.initial=
com.sun.appserv.naming.S1ASCtxFactory

-Djava.naming.provider.url.pkgs=com.sun.enterprise.naming 5.

-Dcom.sun.appserv.iiop.loadbalancingpolicy=roundrobin,host1:port1,h
ost2:port2,... ,host[n]:port[n]
```

優れたパフォーマンスの CORBA Util Delegate クラス

JDK にバンドルされた ORB または Sun ONE Application Server ORB を使うときは、CORBA Util Delegate の優れたパフォーマンスを利用できます。これを利用するには、設定 (`server.xml`) に次の行を追加します。

```
<jvm-options>-Djavax.rmi.CORBA.UtilClass=com.iplanet.ias.util.orbutil.IasUtilDelegate</jvm-options>
```

ORB の負荷のバランスと接続をチューニングするときは、サーバー ORB に作成できる接続の数に注意する必要があります。常に少数の接続から始めて徐々に数を増やし、パフォーマンスの改善を調べることをお勧めします。サーバーへの接続は、その接続からアクティブにデータを読み取る ORB スレッドに変換されます。これらのスレッドはプールされませんが、接続が閉じられるまでは存在し続けます。

設定可能要素の使用について

次の表は、アプリケーションのチューニングに関連する ORB モジュールと、サーバーの設定可能要素を示しています。

表 : 設定可能要素の使用

パス	関連する ORB モジュール	関連するサーバーの設定可能要素
アプリケーションクライアントからアプリケーションサーバーへの RMI/IIOP	通信インフラストラクチャ、スレッドプール	steady-thread-pool-size、max-thread-pool-size、idle-thread-timeout-in-seconds
Sun ONE (サーバー) ORB から Sun ONE Application Server への RMI/IIOP	通信インフラストラクチャ、スレッドプール	steady-thread-pool-size、max-thread-pool-size、idle-thread-timeout-in-seconds
ベンダー ORB からの RMI/IIOP	通信インフラストラクチャの一部、スレッドプール	steady-thread-pool-size、max-thread-pool-size、idle-thread-timeout-in-seconds
プロセス内	スレッドプール	steady-thread-pool-size、max-thread-pool-size、idle-thread-timeout-in-seconds

スレッドプールのサイズ設定

これまでに説明した方法でインバウンドとアウトバウンドの接続の数を調べたら、スレッドプールのサイズをチューニングします。これは、サーバーのパフォーマンスと応答時間に大きく影響します。

このサイズを計算するときは、並行して処理する要求の数、マシンで利用できるリソース (CPU の数、メモリの量など)、クライアント要求の処理に必要な応答時間を考慮する必要があります。小さすぎる値を設定すると、サーバーが並行して処理できる要求の数に影響します。ワーカースレッドによる処理が開始されるまでタスクがタスクキューに入れられるため、要求への応答時間が長くなります。一方、要求を処理するワーカースレッドの数を増やすと、スレッド数の増加によってより多くの並行処理が行われるようになるため、より多くのシステムリソースが消費され、システム全体のパフォーマンスが低下することがあります。つまり、スレッドが EJB コンテナ内の共有ストラクチャを取得するまでにかかる時間が長くなり、応答時間に影響します。ワーカースレッドプールは、プールやキャッシュの縮小など、EJB コンテナのクリーニング用にも使用されます。サイズを決定するときは、この処理も考慮する必要があります。

サーバーはすべてのスレッドを維持する必要があるため、ORB ワーカースレッドの数を過剰に増やすことも問題です。アイドル状態のスレッドは、`idle-thread-time-out-in-seconds` を過ぎると削除されます。次の例は、`server.xml` の一部のコードを示しています。これには、`iiop-service` のセクションが含まれます。

```
<iiop-service>

    <orb message-fragment-size=1024

    steady-thread-pool-size=10

    max-thread-pool-size=200

    idle-thread-timeout-in-seconds=300

    max-connections=1024

    monitoring-enabled=false />

    <iiop-listener id=orb-listener-1 address=0.0.0.0 port=3700
    enabled=true>

</ioop-listener>
```

関連する注意点

pass-by-value セマンティクスと pass-by-reference セマンティクスに関する注意点については、「EJB コンテナのパフォーマンスチューニング」を参照してください。

IIOP メッセージの分析

Sun ONE Application Server から渡される IIOP メッセージの内容の分析が役立つことがあります。メッセージのダンプを取得するには、`-Dcom.sun.CORBA.ORBDebug=giop` を jvm オプションとして `server.xml` に渡します。ダンプは server.log ファイルに生成されます。クライアント ORB でも同じオプションを利用できます。

次に、出力例を示します。

```
[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout:
++++++++++++++++++++++++++++++

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout:
Message(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]):

createFromStream: type is 4 <

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout:
MessageBase(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): Message GIOP version: 1.2

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout:
MessageBase(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): ORB Max GIOP Version: 1.2

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout:
Message(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): createFromStream: message construction
complete.

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout:
com.sun.corba.ee.internal.iiop.MessageMediator(Thread[ORB
Client-side Reader, conn to 192.18.80.118:1050,5,main]): Received
message:

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout: ----- Input
Buffer -----

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout: Current index:
0

[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout: Total length :
340
```

```
[29/Aug/2002:22:41:43] INFO (27179):CORE3282:stdout: 47 49 4f 50 01
02 00 04 0 0 00 01 48 00 00 00 05 GIOP.......H....
```

注 -Dcom.sun.CORBA.ORBdebug=giop フラッグを指定すると、ログファイルに多数のデバッグメッセージが生成されます。メッセージの分割が想定される場合にだけこのオプションを利用してください。

分割メッセージ

上の例では、「createFromStream」の種類は 4 と出力されています。これは、このメッセージが、容量の大きなメッセージを分割した一部であることを示しています。分割サイズを変更することで、メッセージが分割されないようにすることができます。これは、メッセージが複数に分割されずに単体として送信されるため、分割した複数のメッセージ (部分) を送信するオーバーヘッドと、受信側でメッセージを元の状態に戻すオーバーヘッドを回避できることを意味します。分割サイズの指定が小さいために、アプリケーションに送信するほとんどのメッセージが分割される場合は、分割サイズを大きくすると効率が向上する可能性があります。一方、ごく少数のメッセージしか分割されない場合は、分割されるメッセージの数が増えない程度に分割サイズを小さくすることで、メッセージの書き込みに割り当てるバッファを小さくすることができます。

EJB のローカルインタフェース

EJB のローカルインタフェースを使う場合は、ORB が使用されないことに注意してください。この場合、すべての引数は参照によって渡され、オブジェクトのコピーは行われません。

トランザクションマネージャのチューニング

分散トランザクションシステムでは、後からトランザクションを復元できるように、トランザクションアクティビティがトランザクションログに書き込まれます。しかし、トランザクションログへの書き込みはパフォーマンスに影響します。トランザクションの復元よりもパフォーマンスの方が重要な場合は、`disable-distributed-transaction-logging` プロパティを使ってログの記録を無効化します。このプロパティは、デフォルトでは `server.xml` に設定されていません。

トランザクションマネージャを使う場合は、`automatic-recovery` 属性と `key-point-interval` 属性もパフォーマンスに影響します。`automatic-recovery` を true に設定すると、`disable-distributed-transaction-logging` が無視され、トランザクションログが常に記録されます。`automatic-recovery` を false に設定した場合は、`disable-distributed-transaction-logging` に従ってトランザクションログを記録するかどうかが決定されます。

automatic-recovery

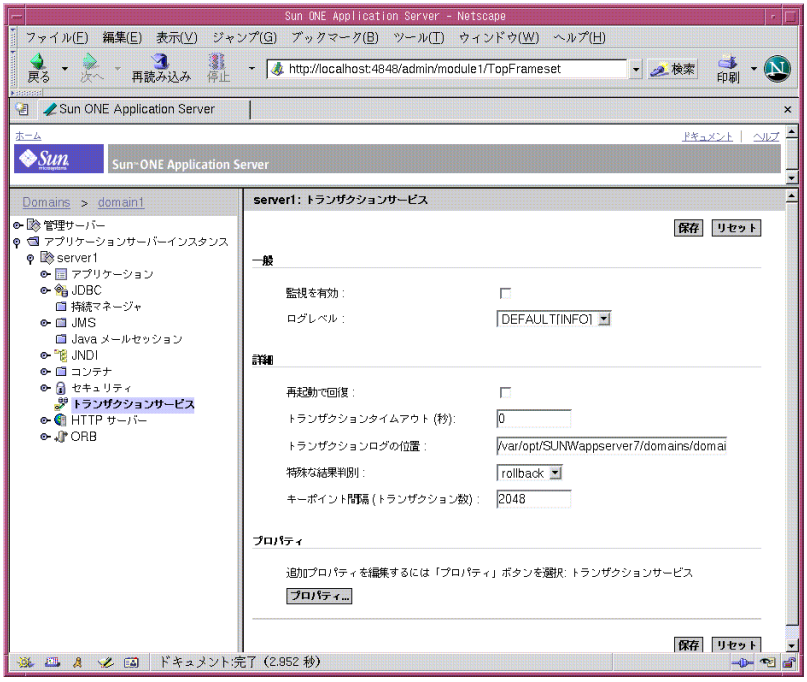
この値は、`disable-distributed-transaction-logging` 属性とともにパフォーマンスに影響します。これは次のように機能します。

1. `automatic-recovery` を true に設定すると、トランザクションログが常に記録されます。
2. automatic-recovery が false で、`disable-distributed-transaction-logging` がオフ (デフォルトの設定) の場合は、ログは記録されます。
3. automatic-recovery が false で、`disable-distributed-transaction-logging` がオンの場合は、トランザクションログは記録されません。これにより、パフォーマンスを約 20% 改善できますが、トランザクションログが残らないため、復元は不可能となります。言い換えれば、トランザクションログを記録する 1 と 2 では、パフォーマンスが約 20% 低下します。これらの結果は、グローバルトランザクションの集中テストによって得られた値です。実際のアプリケーションでは、影響が小さくなることがあります。

keypoint-interval

このプロパティのデフォルト値は 2048 です。キーポイントを設定することで、完了したトランザクションのエントリを削除してログファイルをクリーニングする頻度が設定され、プロセスの物理的なログサイズが大きくなり過ぎないように制限できます。頻繁にチェックするポイントを設定すると、パフォーマンスに影響します。ほとんどの場合は、デフォルトの設定で問題ありません。

次の図は、管理インタフェースのトランザクションマネージャ設定画面を示しています。



図：管理インタフェースによるトランザクションサービスのチューニング

トランザクションマネージャの監視

トランザクションマネージャを監視して、パフォーマンスに関する統計を得ることができます。この統計を出力するには、asadmin ユーティリティを使って次のコマンドを実行します。

```
asadmin>export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=password
AS_ADMIN_HOST=localhost
asadmin>get -m server1.transaction-service.*
```

次の、上記コマンドの出力例を示します。

```
********** Stats for JTS ************

total-tx-completed = 244283

total-tx-rolled-back = 2640

total-tx-inflight = 702

isFrozen = False

inflight-tx =
Transaction Id , Status, ElapsedTime(msec)

000000000003C95A_00, Active, 999
```

参考資料

- プロファイルの詳細については、『Sun ONE Application Server 開発者ガイド』の「J2EE アプリケーションの開発」の章にある「プロファイルツール」を参照してください。
- SNMP 監視の詳細については、『Sun ONE Application Server 管理者ガイド』の「Sun ONE Application Server の監視と管理」の章を参照してください。
- 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。

第 5 章

Java 実行システムのチューニング

Solaris 環境は、デフォルトで 2 レベルのスレッドモデルをサポートしています (Solaris 8 まで)。アプリケーションレベルの Java スレッドは、ユーザーレベルの Solaris スレッドにマッピングされ、制限のあるライトウェイトプロセス (LWP) プール上で多重化されます。システムのプロセッサと同じ数の LWP があるだけで、カーネルリソースの保存とシステム効率の向上が可能になることがよくあります。これは、ユーザレベルのスレッドが何百もある場合に有効です。幸いにも (あるいは不幸にも)、複数のスレッドモデル、複数のモデル内同期メソッドから選択することができますが、これは VM ごとに異なります。さらに、スレッドライブラリが Solaris 8 から 9 に移行したことで、多くの選択肢が消滅したことも問題を複雑にしています。1.4 VM には 2 レベルのモデルがありますが、デフォルトでは VM は LWP ベースの同期を使うので、効率的な 1 対 1 のスレッド / LWP モデルを利用できます。

この章では、次のトピックについて説明します。

- 代替スレッドの使用
- メモリの管理と割り当て

代替スレッドの使用

Solaris 8 では、LD_LIBRARY_PATH の /usr/lib の前に /usr/lib/lwp を挿入することで、/usr/lib/lwp/ に保存されている代替スレッド libthread.so をロードできます。一部のアプリケーションでは、特に、使っているスレッドが少ない場合は、スループットとシステム利用効率が向上します。

デフォルトでは、Sun ONE Application Server は /usr/lib/lwp を使用します。デフォルトの設定を変更して LWP を使わないようにするには、startserv スクリプトの LD_LIBRARY_PATH から /usr/lib/lwp を削除します。ただし、特に必要のない場合は変更しないでください。

多数のスレッドを使うアプリケーションでは、/usr/lib/libthread.so ライブラリが適しています。これによって 1.4 のデフォルトである LWP ベースの同期が有効になるだけでなく、TLABS (スレッドローカル割り当てバッファ) が無効化され、ヒープがくり返し消費されるために GC が頻繁に行われることになるため、多数のスレッドを利用するアプリケーションでは -Xconcurrentio によるチェックが必要です。

Java を使った Solaris のスレッド設定について詳細は、http://java.sun.com/docs/hotspot/threads/threads.html を参照してください。(英語のみ)

メモリの管理と割り当て

アプリケーションの効率的な実行は、メモリとガベージコレクションをいかに効率的に管理するかにかかっています。次に、メモリ管理機能と割り当て機能の最適化について説明します。

- ガベージコレクタのチューニング
- ガベージコレクタのその他の設定
- ガベージコレクションの追跡
- Java ヒープのチューニング
- HotSpot 仮想マシンのチューニングオプション

ガベージコレクタのチューニング

ガベージコレクションは、オブジェクトに割り当てられ、不要になったヒープ領域を回収します。不要オブジェクトを特定し、削除するプロセスはすべてのアプリケーションに影響し、25% ものスループットが消費されます。

ほとんどすべての Java 実行時環境には、世代別メモリシステムと洗練されたガベージコレクションメカニズムが用意されています。世代別メモリシステムでは、ヒープが複数の「世代」に区切られます。各世代のサイズは、慎重に決定されます。集積したオブジェクトによってメモリ容量が少なくなると、ガベージコレクションが強制的に実行されます。世代別メモリシステムの効率は、ほとんどのオブジェクトの生存期間が短いという事実に基づいています。ヒープ領域は、Old 世代と New 世代に分けられます。

New 世代には、新しいオブジェクトの領域 (Eden) と 2 つの Survivor 領域があります。新しいオブジェクトは、Eden に割り当てられます。長く生存するオブジェクトは、New 世代から Old 世代に移動されます。

New 世代では、2 つの下位領域 (2 つの Survivor 領域) と Eden を使った高速コピーによるガベージコレクションが行われ、生存し続けているオブジェクトは、一方の Survivor 領域からもう一方の Survivor 領域にコピーされます。New 世代で何回ものガベージコレクションを経験しても生存し続けたオブジェクトは、古い世代に移動されます。古い世代はサイズが大きく、簡単にいっぱいになることはありません。このため、ガベージコレクションの頻度は低く、1 回のガベージコレクションには、New 世代の領域だけで行われるコレクションより長い時間が必要となります。Old 世代の領域で行われるガベージコレクションは、フルガベージコレクション (フル GC) とも呼ばれます。

頻繁に行われる New 世代領域でのガベージコレクションは高速で (数ミリ秒)、たまに行われる フル GC は比較的時間がかかります (ヒープサイズに応じて数 10 ミリ秒から数秒)。

トレインアルゴリズムなど、その他のガベージコレクションアルゴリズムはインクリメンタル (増分的) です。フル GC はいくつかの増分区分に分断されます。このため、フル GC の実行中も、高速ガベージコレクションによる短時間の中断が生じる可能性が高くなります。これは、オーバーヘッドを伴うため、企業向けの Web アプリケーションでは使われません。

New 世代がいっぱいになると、Survivor 領域のオブジェクトを Old 世代に移動するための小規模なコレクションが行われます。Old 世代がいっぱいになると、オブジェクトヒープ全体を対象とした大規模なコレクションが行われます。

HotSpot と Solaris JDK は、どちらもスレッドローカルなオブジェクト割り当てプールを使って、ロックされず、高速でスケーラブルなオブジェクト割り当てを行います。旧世代の Java 仮想マシンでは、ユーザーアプリケーションレベルのオブジェクトプーリングがより効率的でした。オブジェクトの構築に非常に手間がかかり、しかも重要と考えられる場合にだけ、実行プロファイルでのプールを検討します。

ガベージコレクションのチューニングについて詳細は、
http://java.sun.com/docs/hotspot/gc/index.html を参照してください。

ガベージコレクションの追跡

ガベージコレクションのパフォーマンスは、主にスループットとポーズによって測定されます。スループットは、ガベージコレクション以外のアクティビティが行われた合計時間の割合です。

ポーズは、ガベージコレクションによってアプリケーションが応答不能のようになる状態です。ガベージコレクションの要件は、ユーザーごと異なります。サーバーを多用するアプリケーションでは、スループットを測定対象とするかもしれませんが、グラフィカルプログラムは短いポーズでも機能しなくなることがあります。これ以外にも、フットプリントと即応性が重要な指標となります。

フットプリント

フットプリントはプロセスの作業セットで、ページとキャッシュ行単位で測定されます。即応性は、オブジェクトが不要になってから、メモリが解放されるまでの時間です。分散システムでは、これが重要になります。

各世代のサイズを決定するには、これらの要素を組み合わせて考える必要があります。New 世代を大きくすれば、スループットを最大化できるかもしれません。しかし、フットプリントと即応性は犠牲にされます。New 世代を小さくすることで、ポーズを最小化できますが、コレクションの頻度は高まります。

世代のコレクションによるポーズは、Java 仮想マシンの診断出力で確認できます。コマンド行に引数 `-verbose:gc` を指定すると、コレクションのたびに情報が出力されます。次に、このフラグを Java 仮想マシンに渡したときに出力される情報の例を示します。

```
[GC 50650K->21808K(76868K), 0.0478645 secs]
[GC 51197K->22305K(76868K), 0.0478645 secs]
[GC 52293K->23867K(76868K), 0.0478645 secs]
[Full GC 52970K->1690K(76868K), 0.54789968 secs]
```

矢印の前後の数字は、コレクションの前後に生存していたオブジェクトの合計サイズを示します。カッコ内の数字は合計空き容量を示します。これは、ヒープの合計容量からいずれかの Survivor 領域を差し引いたものです。この例では、3 回の小規模なコレクションと 1 回の大規模なコレクションが行われました。最初の GC では、コレク

ション前に生存していたオブジェクトは 50650 キロバイトで、コレクション後は 21808 キロバイトになりました。つまり、28842 キロバイトの不要オブジェクトが回収されたことになります。ヒープの合計サイズは 76868 キロバイトです。コレクションの処理には、0.0478645 秒かかりました。

ガベージコレクタのその他の設定

クラスを動的に生成し、ロードするアプリケーションでは、Old 世代は GC のパフォーマンスに関連しません。クラスを動的に生成し、ロードするアプリケーション (JSP) では、Old 世代がいっぱいになることでフル GC が行われる場合に、Old 世代が GC のパフォーマンスに影響を及ぼします。`-XX:MaxPermSize` オプションを使って、Old 世代を最大化します。

アプリケーションがガベージコレクタと対話するには、`System.gc()` 呼び出しを使ってコレクションを明示的に実行します。ただし、大規模なコレクションが強制されたり、大規模システムのスケーラビリティが損なわれるため、リソースの管理をアプリケーションに委ねることはお勧めできません。これを無効化するには、`-XX:+DisableExplicitGC` フラグを使います。

Sun ONE Application Server は、管理モジュールの RMI を使って監視を行います。RMI ベースの分散アプリケーションでは、たまに行われるローカルコレクション以外のガベージコレクションは行われません。このため、RMI は定期的なフル GC を強制的に実行します。このコレクションの頻度は、`-sun.rmi.dgc.client.gcInterval` プロパティを使って制御できます。たとえば、`- java -Dsun.rmi.dgc.client.gcInterval=3600000` と指定すると、デフォルトの 1 分おきではなく、1 時間おきに明示的なコレクションが行われます。

Java 仮想マシンの属性を指定する方法は、次のとおりです。

- `server.xml` を編集して `<jvm-config>vm tunable</jvm-config>` を追加します。この `vm tunable` は、適用する属性を示します。
- 管理インタフェースの JVM 設定で JVM オプションを設定します。

Java ヒープのチューニング

ここでは、パフォーマンスを向上するための Java ヒープのチューニングについて説明します。

- Java ヒープのサイズ設定について
- Solaris でのヒープ設定の例
- Windows でのヒープ設定の例

Java ヒープのサイズ設定について

ヒープのサイズは、さまざまなパラメータを使って制御できます。

`-Xms` パラメータと `-Xmx` パラメータは、ヒープの最大サイズと最小サイズを指定します。各世代の領域がいっぱいになるとコレクションが行われるため、スループットは使用可能メモリの量に反比例します。デフォルトでは、JVM はコレクションのたびにヒープサイズを調節し、空き容量と生存オブジェクト容量の比率を特定の範囲内に収めようとします。この範囲は、`-XX:MinHeapFreeRatio=<minimum>` パラメータと `-XX:MaxHeapFreeRatio=<maximum>` パラメータを使ってパーセント単位で設定されます。合計サイズは、`-Xms` と `-Xmx` によって制限されます。

サーバー側のアプリケーションでは、`-Xms` と `-Xmx` の値に同じ固定のヒープサイズを指定します。ヒープのサイズが変動すると、JVM は事前に定義されている `NewRatio` を維持するために、New 世代と Old 世代のサイズを計算しなおします。

`NewSize` パラメータと `MaxNewSize` パラメータは、New 世代の最小サイズと最大サイズを制御します。2 つのパラメータに同じ値を設定することで、New 世代のサイズを固定できます。New 世代のサイズを大きくするほど、小規模なコレクションの実行回数は少なくなります。デフォルトでは、New 世代は `NewRatio` によって制御されます。たとえば `-XX:NewRatio=3` と設定すると、New 世代と Old 世代の比率が 1:3 になり、Eden 領域と Survivor 領域の合計サイズがヒープ全体の 4 分の 1 になります。安全のため、`NewSize` と `MaxNewSize` には同じ値を設定します。

デフォルトでは、Sun ONE Application Server は Java HotSpot Server JVM を使って起動されます。Server JVM のデフォルトの `NewRatio` は 2 で、New 世代がヒープ全体の 3 分の 1、Old 世代が 3 分の 2 を占めます。New 世代を大きくするほど存在期間の短い多数のオブジェクトに対応でき、処理に時間のかかる大規模なコレクションの回数を減らせます。同時に、Old 世代のサイズも、存在期間の長いオブジェクトを保持できる程度の大きさがあります。

Java ヒープのサイズを決定するときは、次の点に注意してください。

- JVM に割り当てることができるメモリの総量を決定します。New 世代のサイズを決定するための独自のパフォーマンス測定対象をグラフにして、最適な設定を求めます。

- New 世代には十分なメモリを割り当てます。1.4 のデフォルト値は、NewRatio と -Xmx の設定から計算されます。
- New 世代の Eden 領域を大きくするほど、フルガベージコレクションの実行間隔は長くなります。ただし、それに比例して New 世代でのコレクションに時間がかかるようになります。一般に、Eden のサイズは最大ヒープサイズの 4 分の 1 から 3 分の 1 に設定します。

  通常は、New 世代より Old 世代を大きく設定する必要があります。

Survivor 領域の比率の設定

SurvivorRatio パラメータは、2 つの Survivor 領域のサイズを制御します。たとえば -XX:SurvivorRatio=6 と設定すると、各 Survivor 領域と Eden 領域の比率は 1:6 になり、それぞれの Survivor 領域のサイズは New 世代全体の 8 分の 1 となります。JDK 1.4 では、Solaris のデフォルトは 32 です。Survivor 領域が小さすぎると、高速コピーによるコレクションがオーバーフローし、その分が Old 世代に送られてしまいます。Survivor 領域が大きすぎると、この領域は空になってしまいます。ガベージコレクションのたびに、JVM は Old 世代に移動するまでにオブジェクトをコピーする回数のしきい値を選択します。

このしきい値は、Survivor 領域の使用容量が半分になる値に設定されます。

このしきい値と、New 世代のオブジェクトのコピー回数を表示するときは、-XX:+PrintTenuringDistribution オプションを使います。アプリケーションのライフタイムの分布を把握しておくと役立ちます。

最新のデフォルト設定を確認するときは、http://java.sun.com/docs/hotspot/VMOptions.html を参照してください。(英語のみ)

Solaris でのヒープ設定の例

次に、Solaris 環境での負荷の大きいサーバー中心型のアプリケーション用の Sun ONE Application Server のヒープ設定例を示します。これは、server.xml ファイルに設定されます。

```
<jvm-options> -Xms3584m </jvm-options>

<jvm-options> -Xmx3584m </jvm-options>

<jvm-options> -verbose:gc </jvm-options>

<jvm-options> -Dsun.rmi.dgc.client.gcInterval=3600000
</jvm-options>
```

Windows でのヒープ設定の例

次に、Windows 環境での負荷の大きいサーバー中心型のアプリケーション用の Sun ONE Application Server のヒープ設定例を示します。これは、`server.xml` ファイルに設定されます。

```
<jvm-options> -Xms1400m </jvm-options>

<jvm-options> -Xmx1400m </jvm-options>
```

HotSpot 仮想マシンのチューニングオプション

HotSpot は、Java アプリケーションのパフォーマンスを向上するための、「その場限りの」バイトコードコンパイラです。これを細かくチューニングすることで、パフォーマンスを向上できます。Sun ONE Application Server を使うときは、HotSpot のマニュアルを参照し、JVM が適切にチューニングされていることを確認してください。

次の Web ページを参照することをお勧めします。

- Java HotSpot VM オプションについては、
  http://java.sun.com/docs/hotspot/VMOptions.html (英語のみ)
- Java HotSpot 仮想マシンに関するよくある質問については、
  http://java.sun.com/docs/hotspot/PerformanceFAQ.html (英語のみ)

第 6 章

オペレーティングシステムのチューニング

Solaris の TCP/IP 設定のチューニングは、多数のソケットを開閉するプログラムで役立ちます。Sun ONE Application Server は、少数の固定された接続のセットで動作し、パフォーマンスの向上はアプリケーションサーバーノードでの向上ほど顕著ではありません。Sun ONE Application Server の Web フロントエンドとして設定されている Web サーバーでは、大きな改善を見込めます。ここでは、次の項目について説明します。

- チューニングパラメータ
- Solaris ファイル記述子の設定
- Linux の設定

チューニングパラメータ

次の表は、パフォーマンスとスケーラビリティのベンチマークに使われる、Solaris オペレーティングシステムのチューニングパラメータを示しています。これらの値は、最適な結果を得られるシステムのチューニング例を示しています。

表 : Soraris オペレーティングシステムのチューニング

パラメータ	適用範囲	デフォルト値	調整値	コメント
rlim_fd_max	/etc/system	1024	8192	オープンなファイル記述子の限界を処理する。予想される負荷 (該当する場合は関連づけられたソケット、ファイル、パイプ) を考慮する必要がある

表 : Soraris オペレーティングシステムのチューニング (続き)

rlim_fd_cur	/etc/system	1024	8192	
sq_max_size	/etc/system	2	0	ストリームドライバのキューサイズを制御する。0 に設定すると無限になり、パフォーマンスはバッファの容量不足による影響を受けなくなる。クライアントにも設定される
tcp_close_wait_interval	ndd /dev/tcp	240000	60000	クライアントにも設定される
tcp_time_wait_interval	ndd /dev/tcp	240000	60000	
tcp_conn_req_max_q	ndd /dev/tcp	128	1024	
tcp_conn_req_max_q0	ndd /dev/tcp	1024	4096	
tcp_ip_abort_interval	ndd /dev/tcp	480000	60000	
tcp_keepalive_interval	ndd /dev/tcp	7200000	900000	トラフィックの多い Web サイトでは、これより小さな値を設定する
tcp_rexmit_interval_initial	ndd /dev/tcp	3000	3000	再転送率が 30 ～ 40% を超える場合は、この値を大きくする
tcp_rexmit_interval_max	ndd /dev/tcp	240000	10000	
tcp_rexmit_interval_min	ndd /dev/tcp	200	3000	
tcp_smallest_anon_port	ndd /dev/tcp	32768	1024	クライアントにも設定される
tcp_slow_start_initial	ndd /dev/tcp	1	2	少量のデータを少し高速で転送する
tcp_xmit_hiwat	ndd /dev/tcp	8129	32768	送信バッファを大きくする
tcp_recv_hiwat	ndd /dev/tcp	8129	32768	送信バッファを大きくする
tcp_conn_hash_size	ndd /dev/tcp	512	8192	接続ハッシュテーブルは、アクティブな TCP 接続に関するすべての情報を保持する (ndd -get /dev/tcp tcp_conn_hash)。この値は接続数を制限するものではないが、接続のハッシュにかかる時間が長くなることがある。ルックアップを効率化するには、サーバーで予想される並行 TCP 接続数の半分の値を設定する (netstat -nP tcp | wc -l で求めることができる)。デフォルトは 512。これは /etc/system だけに設定され、起動時に適用される

Solaris ファイル記述子の設定

Solaris では、ulimit プロパティによるオープンファイルの最大数の設定は、サポートできる RMI/IIOP クライアントの最大数に大きく影響します。

ハードリミットを引き上げるには、/etc/system に次のコマンドを追加して 1 回再起動します。

```
set rlim_fd_max = 8192
```

このハードリミットの設定は、次のコマンドを使って確認できます。

```
ulimit -a -H
```

ハードリミットを設定すると、次のコマンドを使ってこの値を明示的に増やすことができます (設定した限界まで)。

```
ulimit -n 8192
```

この制限の設定は、次のコマンドを使って確認できます。

```
ulimit -a
```

たとえば、ulimit のデフォルト値である 64 では、簡単なテストドライバがサポートする並行クライアントの数は 25 に過ぎませんが、ulimit を 8192 に設定すると、同じテストドライバを使って 120 までの並行クライアントをサポートできます。このテストドライバは複数のスレッドを生成し、各スレッドが JNDI ルックアップと同じビジネスメソッドの呼び出しを行います。ビジネスメソッドの呼び出し間隔 (遅延) は 500 ミリ秒で、約 100 キロバイトのデータがやりとりされます。

これらの設定は、Solaris 上の RMI/IIOP クライアントに適用されます。ファイル記述子の限界設定については、Sun Microsystems のマニュアル Web サイト (www.docs.sun.com) にある Solaris のマニュアルを参照してください。

Linux の設定

システムの起動時に実行される次のパラメータを /etc/rc.d/rc.local ファイルに追加する必要があります。

```
<-- begin

# 最大ファイル数の上限は 4 メガバイトごとに最大で 256 記述子が追加されます。システムの RAM の容量に応じてファイル記述子の数を指定します。
echo ″65536″ >

# i ノードの上限は、ファイル数の上限の 3 〜 4 倍です。
# ファイルがありません。
#echo ″262144″ > /proc/sys/fs/inode-max

# 利用できるローカルポートを増やします。
echo 1024 25000 > /proc/sys/net/ipv4/ip_local_port_range

# ソケットバッファで利用できるメモリを増やします。
echo 2621143 > /proc/sys/net/core/rmem_max
echo 262143 > /proc/sys/net/core/rmem_default

# 2.4.X のカーネルでは次のように設定
echo 4096 131072 262143 > /proc/sys/net/ipv4/tcp_rmem
echo 4096 13107262143 > /proc/sys/net/ipv4/tcp_wmem

# RFC2018 ″TCP 選択確認応答″ と ″RFC1323 TCP タイムスタンプ″ を無効化します。
echo 0 > /proc/sys/net/ipv4/tcp_sack
echo 0 > /proc/sys/net/ipv4/tcp_timestamps

# 実行時に shm に割り当てられるメモリの最大量を 2 倍にします。
echo ″67108864″ > /proc/sys/kernel/shmmax

# Linux の仮想メモリ VM サブシステムを改善します。
echo ″100 1200 128 512 15 5000 500 1884 2″> /proc/sys/vm/bdflush

# sysctl も実行します。
sysctl -p /etc/sysctl.conf

-- end -->
```

更に、/etc/sysctl.conf ファイルを作成し、次の値を追加します。

```
<-- begin
# パケット転送を無効化します。
net.ipv4.ip_forward = 0
# ソースルートの検証を有効化します。
net.ipv4.conf.default.rp_filter = 1
# Magic Sysrq Key を無効化します。
kernel.sysrq = 0
fs.file-max=65536
```

```
vm.bdflush = 100 1200 128 512 15 5000 500 1884 2

net.ipv4.ip_local_port_range = 1024 65000
net.core.rmem_max= 262143
net.core.rmem_default = 262143

net.ipv4.tcp_rmem = 4096 131072 262143
net.ipv4.tcp_wmem = 4096 131072 262143
net.ipv4.tcp_sack = 0
net.ipv4.tcp_timestamps = 0

kernel.shmmax = 67108864
```

第 7 章

パフォーマンスに関する一般的な問題

ここでは、Web サイトのパフォーマンスに関する次の一般的な問題について説明します。

- check-acl サーバーアプリケーション機能
- 低メモリの状況
- 利用可能スレッドの不足
- キャッシュの未使用
- キープアライブ接続のフラッシュ
- ログファイルのモード

check-acl サーバーアプリケーション機能

サーバーのパフォーマンスを最適化するには、ACL は必要な場合にだけ使用します。

デフォルトサーバーは、デフォルト ACL を含む ACL ファイルで設定されています。デフォルト ACL では、サーバーへの書き込みアクセスは「all」だけに許可され、es-internal ACL の書き込みアクセスは「anybody」に限定されます。後者によって、サーバーのマニュアル、アイコン、検索 UI ファイルが保護されます。

デフォルトの `obj.conf` ファイルには、`NameTrans` という行があります。これは、読み取り専用に設定する必要のあるディレクトリを es-internal オブジェクトにマッピングし、このオブジェクトには es-internal ACL の check-acl SAF が含まれます。

デフォルトオブジェクトには、デフォルト ACL の check-acl SAF も含まれます。

サーバーのパフォーマンスを向上するには、server.xml の仮想サーバータグから aclis プロパティを削除します。これにより、ACL の処理が停止されます。

ACL によって保護されない URI のデフォルトオブジェクトから check-acl SAF を削除して、パフォーマンスを向上することもできます。

低メモリの状況

低メモリの状況で Sun ONE Application Server を実行するには、RqThrottle の値を下げて、スレッドの上限を最小限に抑えます。また、MaxProcs の値を下げて、Sun ONE Application Server が生成するプロセスの最大数を減らすことも有効です。

利用可能スレッドの不足

サーバーでは、スレッド数の上限を超えてアクティブスレッドの数が増えることはありません。同時に処理される要求の数がこの上限に達すると、サーバーは古い接続が解放されるまで新しい接続の処理を停止します。これにより、応答時間が長くなることがあります。

Sun ONE Application Server では、サーバーの RqThrottle のデフォルト値は 128 です。サーバーが同時に処理できる要求の数を増やすときは、RqThrottle の値を増やします。

サーバーで利用できるスレッドが不足している場合、その兆候は応答時間の長さに表れます。ブラウザから要求を送信した場合、通常はサーバーとの接続は迅速に確立されますが、利用可能スレッドが不足しているサーバーでは、クライアントに応答が戻るまでに長い時間が必要となります。

サーバーのスレッド不足を確認する一番の方法は、アクティブセッションの数が RqThrottle によって設定される最大数にどれだけ近づいているかを確認することです。手順については、「最大同時要求数」を参照してください。

キャッシュの未使用

キャッシュが使われないと、サーバーのパフォーマンスを最適化できません。ほとんどのサイトには、常にキャッシュ可能な GIF ファイルや JPEG ファイルが多数含まれているため、キャッシュを効率的に利用する必要があります。

ただし、一部のサイトでは CGI や SHTML などの動的なソースを使ってほとんどの処理を行っています。一般に、動的なコンテンツをキャッシュすることはできません。また、キャッシュの利用率も低くなります。キャッシュの利用率が低くても、過剰な心配は必要ありません。最も重要なことは、応答時間を低く抑えることです。キャッシュの利用率が低くても、良好な応答時間を得ることはできます。応答時間が良好であれば、キャッシュの利用率を気にする必要はありません。

キャッシュの利用率は、`perfdump` が出力する統計、または Web ベースの管理コンソールの「監視」ページを使って調べることができます。利用率は、キャッシュのルックアップ回数に対するキャッシュのヒット回数の割合を示します。50% 以上であれば、利用率は良好であると言えます。サイトによっては、98% 以上に達することもあります。

CGI や NSAPI の呼び出しが多いサイトでは、キャッシュの利用率が低くなることがあります。カスタム NSAPI 機能を利用している場合も、キャッシュの利用率は低くなります。

キープアライブ接続のフラッシュ

キープアライブ接続なしで 1 秒間に 75 の要求を処理できる Web サイトであれば、キープアライブ接続を有効にすると 1 秒間に 200 ～ 300 の要求を処理できます。クライアントは 1 つのページに対しても多数の項目を要求するため、キープアライブ接続を効率的に利用することが重要です。`KeepAliveCount` が `MaxKeepAliveConnections` を超えると、接続がキープアライブの対象となっており、実際に使われていても、その接続は閉じられます (または「フラッシュ」されます)。

`KeepAliveFlushes` と `KeepAliveHits` の値は、`perfdump` が出力する統計、または Web ベースの管理コンソールの「監視」ページを使って調べることができます。キープアライブ接続が効率的に利用されているサイトでは、`KeepAliveHits` に対する `KeepAliveFlushes` の割合は低く抑えられています。この比率が 1:1 以上に高くなるときは、サイトでキープアライブ接続を効率的に利用できていない可能性があります。

キープアライブ接続のフラッシュを減らすには、init.conf ファイルを編集するか、Web ベースの管理コンソールを使って `MaxKeepAliveConnections` の値を増やします。デフォルト値は 200 です。この値を大きくすることで、より多くのキープアライブ接続を待ち状態で開いたまま維持できます。

警告 UNIX または Linux システムでは、`MaxKeepAliveConnections` の値を大きくしすぎると、サーバーが開いておけるファイル記述子が不足することがあります。UNIX または Linux 環境では、通常、開いておけるファイルの上限は 1024 なので、500 以上の値を設定することはお勧めできません。

ログファイルのモード

ログファイルの設定を詳細モードのままにしておくと、パフォーマンスに大きな影響を生じます。Web ベースの管理コンソールを使うことで、`LogVerbose` には `FINE` 以上の任意のレベルを設定できます。

索引

I

J

K

L

M

N

X

あ

え

か

き

こ

さ

す

せ

た

り

ろ