パフォーマンスおよびチューニングガイド

iPlanet™ Application Server

Version 6.5

816-5274-01
2002 年 2 月

目 次

はじめに

このマニュアルは iPlanet Application Server の上級管理者を対象としています。このマニュアルは、iPlanet Application Server をチューニングして最高のパフォーマンスと信頼性を得るのに役立ちます。iPlanet Application Server の設定を変更する前に、コンフィグレーションファイルのバックアップを行うことをお勧めします。

この章では『iPlanet Application Server インストールガイド』の内容について説明します。この章には次の節があります。

- マニュアルの使用法
- このマニュアルについて
- 前提事項
- このマニュアルの構成
- マニュアルの表記規則

マニュアルの使用法

次の表は、iPlanet Application Server のマニュアル、および『リリースノート』に記述されているタスクと概念を示しています。特定のタスクを行う場合や特定の概念について調べる場合は、該当するマニュアルを参照してください。

注 : 印刷版マニュアルは、http://docs.iplanet.com/docs/manuals/ias.html から入手できます。

情報の内容	参照するマニュアル	添付されている製品
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	http://docs.iplanet.com の iPlanet Web サイトから入手できます。

情報の内容	参照するマニュアル	添付されている製品
iPlanet Application Server およびそのコンポーネント (Web コネクタプラグイン、iPlanet Application Server Administrator) のインストールと、サンプルアプリケーションの設定	インストールガイド	iPlanet Application Server 6.5
次のタスクによる、オープン Java 標準モデル (Servlet、EJB、JSP、および JDBC) に準拠した iPlanet Application Server 6.5 アプリケーションの作成 • アプリケーションのプレゼンテーション層および実行層の作成 • EJB (Enterprise JavaBeans) コンポーネントへのビジネスロジックの個別部分およびエンティティの配置 • JDBC を使ったデータベースとの通信 • 反復テスト、デバッグなどアプリケーションの調整機能を使用した、正確かつ高速に動作するアプリケーションの生成	開発者ガイド	iPlanet Application Server 6.5

情報の内容	参照するマニュアル	添付されている製品
次のタスクを行うための、iPlanet Application Server Administrator Tool による 1 台または複数台のアプリケーションサーバの管理 • サーバの稼動状況の監視およびログ記録 • iPlanet Application Server へのセキュリティの実装 • サーバリソースの高利用度の実現 • Web コネクタプラグインの設定 • データベース接続の管理 • トランザクションの管理 • 複数のサーバの設定 • 複数のサーバでのアプリケーションの管理 • サーバのロードバランス • 分散データ同期の管理 • 開発用の iPlanet Application Server のセットアップ	管理者ガイド	iPlanet Application Server 6.5
iPlanet 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を含む、Netscape Application Server バージョン 2.1 から新しい iPlanet Application Server 6.5 プログラミングモデルへのアプリケーションの移行	移行ガイド	iPlanet Application Server 6.5
Java アプリケーションを作成する場合の iPlanet Application Server クラスライブラリの共有クラスとインターフェイス、およびそれらのメソッドの使用	Server Foundation Class Reference (Java)	iPlanet Application Server 6.5
C++ アプリケーションを作成する場合の iPlanet Application Server クラスライブラリの共有クラスとインターフェイス、およびそれらのメソッドの使用	Server Foundation Class Reference (C++)	別注文

このマニュアルについて

この『パフォーマンスおよびチューニングガイド』では、iPlanet Application Server の多様な機能と、最高のパフォーマンスおよび信頼性を得るための iPlanet Application Server のチューニング方法について説明します。

このマニュアルは、iPlanet Application Server のパフォーマンスをチューニングするために利用できるさまざまなタスクやツールについて理解したいシステム管理者、ネットワーク管理者、評価者、アプリケーションサーバ管理者、Web 開発者、およびソフトウェア開発者を対象にしています。

前提事項

インストールを始める前に、次のトピックについて熟知していることを前提とします。

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

このマニュアルの構成

このマニュアルの構成は次のとおりです。

第 1 章「iPlanet Application Server について」では、iPlanet Application Server の機能とコンポーネントの概要について説明します。

第 2 章「チューニングとサイジングについて」では、サイジング、操作要件の見積もり、およびチューニング可能なパラメータについて説明します。

第 3 章「アプリケーションのチューニング」では、最大のパフォーマンスを得るためにアプリケーションをチューニングする包括的な手順を示します。Java コーディングのガイドラインおよび J2EE プログラミングのガイドラインについてもこの章で説明します。

第 4 章「iPlanet Application Server のチューニング」では、最大のパフォーマンスを得るために iPlanet Application Server をチューニングする包括的な手順を示します。

第 5 章「Java 実行時システムのチューニング」では、メモリのチューニングおよびガベージコレクタの設定について説明します。

第 6 章「オペレーティングシステムのチューニング」では、Solaris チューニングパラメータについての情報を提供します。

第 7 章「データベースサーバのチューニング」では、さまざまなデータベースのチューニングパラメータについて説明します。

第 8 章「パフォーマンス向上のための一般的なガイドライン」には、アプリケーションのパフォーマンスを向上させるための、より良いコーディングについてのガイドラインが用意されています。

第 9 章「サーバパフォーマンスの確認」では、チューニング後のサーバのパフォーマンスを監視して検証することに関連する情報を提供します。

第 10 章「FAQ/ よくある質問」は、iPlanet Application Server のチューニングに関連する最も一般的な質問の一覧です。

マニュアルの表記規則

ファイルとディレクトリのパスは、Windows の形式で表記されます (ディレクトリ名を円記号で区切って表記)。UNIX バージョンでは、ディレクトリパスについては Windows と同じですが、ディレクトリの区切りには円記号ではなくスラッシュが使われます。

このマニュアルでは次のように URL 形式を使います。

`http://server.domain/path/file.html` のような URL 形式を使います。ここで、

- server は、アプリケーションを実行するサーバの名前です。
- domain は、インターネットのドメイン名です。
- path は、サーバ上のディレクトリ構造です。
- file は、個々のファイル名です。

次の表は、iPlanet マニュアルで採用しているフォントの規約を示します。

表 1　フォントの規約

書体	意味	例
モノスペース	ファイル名、ディレクトリ、サンプルコード、コードの一覧表示、および HTML タグ	`Hello.html` ファイルを開きます。 `<HEAD1>` は、最上位の見出しを作成します。
イタリック	変数、コードのプレースホルダ、強調する語句、およびリテラルに使われる語句	名前のフィールドに「*Login*」と入力します。
太字	テキストに初めて登場した用語	**テンプレート**は、ページのアウトラインです。

第 1 章

iPlanet Application Server について

iPlanet™ Application Server は、信頼性、可用性、およびスケーラビリティのある Web サービスを配置する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は、よく設計されたソフトウェアのコンポーネントを使用してビジネスロジックの実装に専念し、大規模な配置のためのサービスは、iPlanet Application Server に任せることができます。

この章には次の節があります。

- iPlanet Application Server のコンポーネント
- iPlanet Application Server のプロセスアーキテクチャ
- iPlanet Application Server 内の通信
- iPlanet Application Server のツール

iPlanet Application Server のコンポーネント

iPlanet Application Server には、円滑なパフォーマンスのために相互の対話を必要とするさまざまなコンポーネントが含まれています。これらのコンポーネントをチューニングして、運用環境と開発環境のパフォーマンスを最適化できます。

この節では、次のトピックについて説明します。

- Web コネクタプラグイン
- アプリケーションサーバプロセス
- Directory Server のコンポーネント
- データベース

Web コネクタプラグイン

Web サーバインスタンスに接続するために、ダイナミックに読み込まれるライブラリです。受信した HTTP リクエストを、アプリケーションサーバインスタンスの Executive Server (KXS) プロセスにリダイレクトします。

アプリケーションサーバプロセス

Executive Server (KXS): Web コネクタプラグインから受信した HTTP トラフィックを Java サーバにリダイレクトします。アプリケーションサーバインスタンス間の高可用性セッションやステートデータの複製を管理します。

Java サーバ (KJS): Web コンテナと EJB コンテナが含まれています。

RMI/IIOP ブリッジ (CXS): RMI/IIOP クライアントから受信したリクエストを、Java エンジン (KJS) に配置された EJB コンテナにリダイレクトします。

Administration Server (KAS): ほかのアプリケーションサーバプロセスを監視し、管理ツールや配置ツールのサーバとして機能します。

Directory Server のコンポーネント

Directory Server プロセス : 認証と承認に使用するユーザおよびグループについての情報が含まれています。アプリケーションサーバのレジストリ情報を格納する分散ストアとして機能します。

iPlanet Administration Server: iPlanet Console 管理ツールへの接続ポイントとして機能します。このサーバは、アプリケーションサーバが機能するためにアクティブである必要はありません。

データベース

iPlanet Application Server がサポートするデータベースには、Oracle、Sybase、Informix、DB2 などがあります。iPlanet Application Server を使って、データベース用のサードパーティ JDBC ドライバを設定することができます。また、データベースドライバのデータソースとトランザクションマネージャも設定できます。

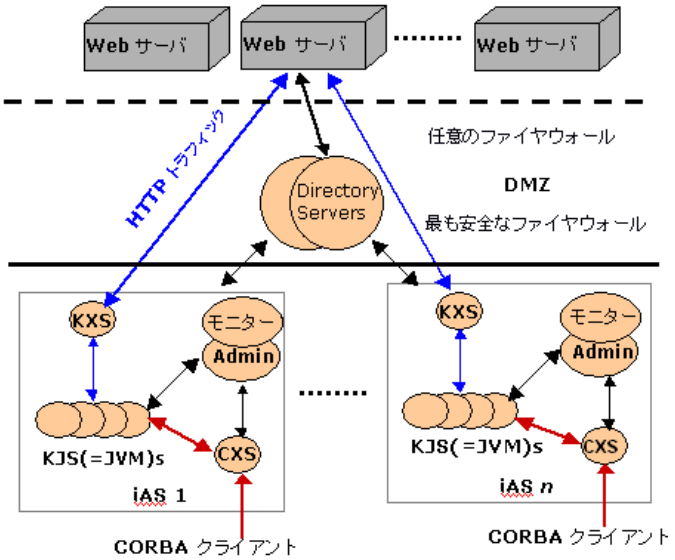
iPlanet Application Server のプロセスアーキテクチャ

パフォーマンスを最大にするには、iPlanet Application Server の基本プロセスアーキテクチャを理解しておく必要があります。iPlanet Application Server の一般的な配置シナリオは次の図に示すとおりです。この図では、わかりやすくするために 1 台の Web サーバインスタンスから iPlanet Application Server および Directory Server へのトラフィックだけを示しています。この図に示されているように、Web サーバ (厳密には Web コネクタプラグイン) は、iPlanet Application Server クラスタに設定されているすべてのノードに対するリクエストのロードバランスを管理します。

複数のインスタンスでクラスタを構成したアーキテクチャでは、ほかのアプリケーションサーバのインスタンスノードの状態を複製することにより、すぐれた水平スケーラビリティと高可用性を提供します。各インスタンスを複数の KJS プロセス (それぞれの KJS プロセスは JVM インスタンス) で設定して、十分なスケーラビリティを達成できます。それぞれの KJS プロセスに複数のスレッドを割り当てると、パフォーマンスが向上します。

次の図で、iPlanet Application Server のすべてのコンポーネントがどのように組み合わされているかを理解できます。この図は、iPlanet Application Server のプロセスアーキテクチャの各部がどのように相互に関連しているか、どのように Web サーバに接続されているか、最終的にユーザにどのように接続されるかを示しています。

図 1-1　iPlanet Application Server のプロセスアーキテクチャ



iPlanet Application Server 内の通信

iPlanet Application Servers は通常、Web サーバ、ディレクトリサーバ、およびクライアントデータベースアプリケーションとともに配置されます。このような場合、プロセス間通信の量が非常に多くなるため、データの統合性とセキュリティの問題が非常に重要になります。アプリケーションサーバ間、Web サーバ間、およびアプリケーションサーバで実行するクライアントアプリケーション間の円滑で安全な通信を確保するために、システムのセキュリティコンポーネントがファイヤウォールによって区切られる場合があります。

iPlanet Application Server の基本コンポーネントは Executive Server (KXS) です。Executive Server はアプリケーションのコンポーネントを作成し、セッションごとのデータ、負荷の監視機能、およびロードバランス機能を iPlanet Application Server のほかのインスタンスとともに管理します。

アプリケーションコードは KXS によって作成されるマルチスレッドプロセスで動作します。プロセスには、C++ Server (KCS) と Java Server (KJS) の 2 つの種類があります。

システムは Administration Server (KAS) によって管理されます。Web サーバは、iPlanet Application Server と同じマシンにインストールすることも、別のマシンにインストールすることもできます。一般的なネットワークインストールの場合、Web サーバは別のマシンにインストールされます。

Web サーバとアプリケーションサーバ間の通信には、Web サーバにあるプラグインを使います。アプリケーションサーバの複数のインスタンスがインストールされている場合、プラグインではロードバランスのメカニズムが選択したアプリケーションサーバのインスタンスと通信します。使用している Web サーバがプラグインを使うことができない場合、CGI (Common Gateway Interface) アプリケーションが通信モジュールとして呼び出されることがあります。この CGI アプリケーションが iPlanet Application Server とのコネクションを確立します。だだし CGI モデルは効率的でないので、通常は最後の手段として使われます。

また、OCL (Object Constraint Language) を使う方法もあります。OCL は CORBA を使って必要なサービスを検索し、アプリケーションサーバを介してそれらのサービスと通信します。この方法では、IIOP コネクションについて標準化されたセキュリティが利用できないため、イントラネット用だけにお勧めします。

サーバ、アプリケーション、およびセキュリティプロトコルの間のプロセス間通信は、さまざまな方法で行われます。iPlanet Application Server インスタンス (KXS/KJS/KCS プロセスグループであるアプリケーションサーバインスタンス) 間にファイヤウォールを置くことは可能ですが、お勧めしません。そのようなファイヤウォールを実装する場合は、ファイヤウォールを通過して IP (Internet Protocol) 通信ができる IGMP (Internet Group Management Protocol) を実装する必要があります。

iPlanet Application Server インスタンス間にファイヤウォールを置くことはお勧めしませんが、環境によっては iPlanet Application Server マシン間に IP マルチキャストトラフィックの高速専用ネットワークを実装することをお勧めします。

iPlanet Application Server システムは、LAN の速度で動作するネットワークを対象に設計されています。iPlanet Application Server インスタンスを WAN (Wide Area Network) に分散させると、パフォーマンスに問題が生じることがあります。iPlanet Application Server を使ったファイヤウォールの設定については、『iPlanet Application Server 管理者ガイド』の第 5 章「iPlanet Application Server の保護」を参照してください。

iPlanet Application Server のツール

iPlanet Application Server の管理インタフェースと配置インタフェースは、iPlanet Application Server Administration Tool (iASAT) と iPlanet Application Server Deployment Tool (iASDT) の 2 つのツールを使って管理します。これらのツールについては、次のトピックで説明します。

- iPlanet Application Server Administration Tool
- iPlanet Application Server Deployment Tool

iPlanet Application Server Administration Tool

iASAT はグラフィカルユーザインタフェースを持つスタンドアロン Java アプリケーションで、1 つまたは複数の iPlanet Application Server インスタンスを管理することができます。多くの場合、管理には、データベースコネクションスレッドの調整やロードバランスパラメータの変更など、パフォーマンスに関連するタスクが含まれます。Administration Tool は、iPlanet Application Server がサポートするどのプラットフォームでも実行することができ、また、ネットワークを介して 1 つまたは複数の iPlanet Application Server インスタンスへ接続することができます。

iPlanet Application Server Administration Tool を使って、サーバプロセス、EJB パラメータ、構成済みのデータソースなどをチューニングして、パフォーマンスを向上させることができます。このガイドでは、管理ツールを使用してアプリケーションサーバをチューニングする方法を説明します。

iPlanet Application Server Deployment Tool

iASDT は、グラフィカルユーザインタフェースを持つスタンドアロンの Java アプリケーションであり、次のようなことができます。

- J2EE アプリケーションコンポーネントをモジュールにパッケージ化する
- モジュールを配置可能なユニットにまとめる
- ユニットを 1 つまたは複数の iPlanet Application Server 実行環境に配置する

J2EE アプリケーションコンポーネントは、配置時にそれらのコンポーネントを受け取るコンテナに従って、モジュールにアーカイブされます。J2EE アプリケーションコンポーネントは、`.jar` 拡張子を持つ EJB JAR モジュールか、または `.war` 拡張子を持つ Web アプリケーションモジュールにアーカイブすることができます。それぞれのモジュールには J2EE 記述子と iPlanet Application Server 固有の配置記述子があり、XML ファイルに保存されます。

第 2 章

チューニングとサイジングについて

この章では、パフォーマンスのチューニングに関するヒントと技法について説明します。iPlanet™ Application Server のチューニング可能な一部のパラメータの簡易ガイドにもなっています。

この章には次のトピックがあります。

- iPlanet Application Server のチューニングが必要な理由
- アプリケーションのサイジングについて
- サイジングに影響を与える要因
- 操作要件について
- パフォーマンスの予測
- 一般的なパフォーマンスのガイドライン
- パフォーマンスチューニングの手順

iPlanet Application Server のチューニングが必要な理由

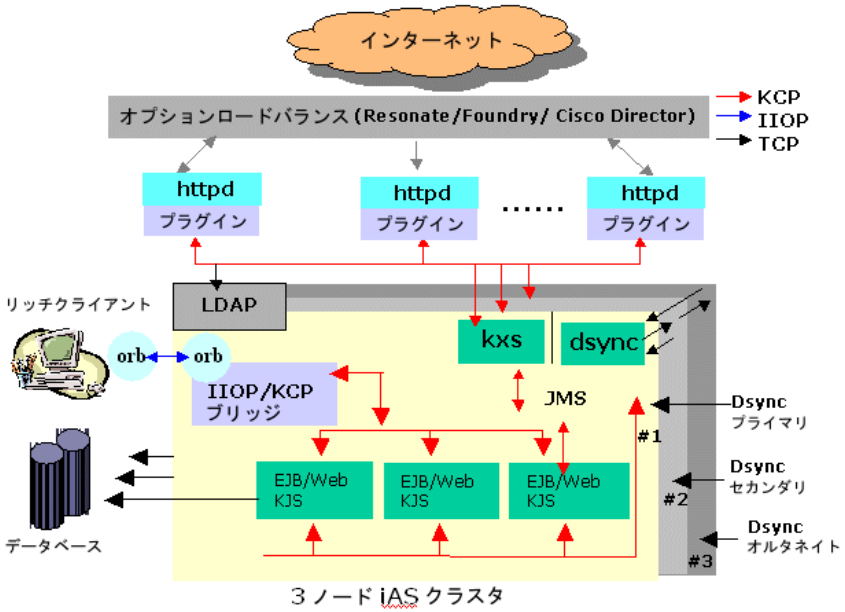
アプリケーションサーバのパフォーマンスに影響を与える変数は非常に多いため、すべてのサイトで最適なパフォーマンスを得られる設定というものはありません。システムのサイズが適切であっても、デフォルトの配置にほんのわずかな変更を加えただけで、パフォーマンスが大きく向上することがあります。

こうした変更によって、システム全体のパフォーマンスが向上することがあります。ある変更は開発環境に良い影響を与えても、運用環境には反対の場合があります。また、その逆のこともあります。

iPlanet Application Server のチューニングを始める前に、チューニングによって得られる結果を正確に評価しておく必要があります。この章では、iPlanet Application Server を最適化するための、さまざまなチューニングオプションやサイジングオプションを紹介します。

確認のために、iPlanet Application Server のプロセスアーキテクチャを次の図に示します。

図 2-1　iPlanet Application Server のプロセスアーキテクチャ



アプリケーションのサイジングについて

システムのサイジングをすると、特定のユーザ負荷をサポートするのに必要なハードウェアの数と種類を予測することができます。

システムを適切なサイズにするには、負荷、アプリケーション、およびプラットフォーム (オペレーティングシステムとハードウェア) の特徴を理解することが重要です。

この節では、負荷、アプリケーション、およびプラットフォームを測定し、アプリケーションサーバのシステム要件を見積もる方法について詳しく説明します。

アプリケーションのサイジング方法を説明する前に、サイジングに影響する各種要因について理解しておく必要があります。

サイジングに影響を与える要因

次の表に、アプリケーションのサイジングに影響を与える要因を示します。

表 2-1 サイジングに影響を与える要因

要因	説明
ユーザ負荷	負荷を処理するのに必要なハードウェアとシステム負荷が釣り合っている必要があります。
アプリケーションの設計と実装	仕事量が非常に少ないアプリケーションでは、同じハードウェアでも多くのユーザを扱うことができます。多くの場合、この種のアプリケーションは共有リソースを待機している時間の割合が大きいため、調整が難しくなります。その反対に大量の計算を実行するアプリケーションは、ユーザごとのハードウェアリソースがより多く必要になる傾向がありますが、この場合のほうがうまく調整されます。 スレッドが共有リソースを競合するアプリケーションでは、共有リソースを待機する時間が多いので、1 つのサーバ内でうまく調整できない傾向があります。このような状況には、多くの小規模サーバを配置するソリューションが最適です。 サーバ間で共有リソースを競合するアプリケーションでは、共有リソースを待機するために時間が費やされるので、クラスタ内ではうまく調整できない傾向があります。このような状況には、少数の大規模サーバを配置するソリューションが最適です。

ハードウェアプラットフォーム	必要なハードウェアの数を減らす上でプロセッサそのもののパフォーマンスが重要になります。通常、アプリケーションでは浮動小数点に集中した計算が含まれていないので、内部的なパフォーマンスが、通常もっとも重要な要因となります。応答時間は、プロセッサのパフォーマンスに大きく依存しています。
安全マージン	共有リソースの著しい競合が発生する可能性がある場合は、高速プロセッサを使用していても、サーバをうまく調整できません。通常、プロセッサをサーバに追加すると達成されるパフォーマンスを判断するうえで、キャッシュの設計とメモリの帯域幅が重要な役割を果たします。

操作要件について

単純な配置のチューニングを始める場合、実装チームで、iPlanet Application Server と使用するアプリケーションの要件に基づいた操作環境の概要レイアウトを作成することができます。この概要レイアウトに、システムの外部インタフェースの相対位置を示します。概要レイアウトからより具体的な設計に移行する前に、新しいシステムについて次の操作要件を考慮する必要があります。

- セキュリティ
- 可用性
- パフォーマンス

セキュリティ

ブラウザと Web サーバ間の通信を暗号化しようとする場合は、以下の要因を考慮する必要があります。

- アプリケーションのすべてまたは一部で、ブラウザと Web サーバ間の通信を暗号化する必要性
- Web サーバ層を、アプリケーションサーバ層およびバックエンドの企業システムから分離された非武装地帯 (DMZ) に配置するかどうか
- Web サーバとアプリケーションサーバ間での暗号化の必要性

Web サーバと iPlanet Application Server 間の通信を暗号化する方法については、『iPlanet Application Server 管理者ガイド』の第 6 章「サーバリソースの可用性の向上」を参照してください。

可用性

アプリケーションの可用性を高めるために、以下の要因を考慮する必要があります。

- アプリケーション可用性の要件。マシンが利用できなくなったときのサービスの損失を許容できるか
- ユーザのセッション情報の損失を許容できるか
- アプリケーションがほかの操作環境で対話処理する場合、弱点となる可能性のあるリンク
- これらの弱点を適度に強化できるか。それともそのようなリンクは不変なものか
- 多くの企業では、80% の CPU ビジーレートを最高点とみなしている。自分の会社の標準は何パーセントか

パフォーマンス

パフォーマンスに関連する以下の要因を考慮する必要があります。

- アプリケーションとのさまざまな対話処理で、エンドユーザに必要となる応答時間
- 認知可能な安定状態およびピーク時のユーザ負荷
- Web リクエストごとの平均およびピーク時のデータ転送量
- 次の 12 ヶ月のユーザ負荷の増加率

ピーク時のユーザ負荷については、アプリケーションサーバで管理される同時セッションの数に注意する必要があります。多くの組織で、ピーク時のユーザ負荷を、同時ユーザの平均数でなく最大ユーザ数とみなしていることがしばしば見受けられます。ユーザ負荷をより現実的な観点から見ると、理論上ピーク時のユーザ数は、数十万あるいは数百万の同時ユーザから数万の同時ユーザへと劇的に減少します。

これらの操作上の要件を定義したら、配置環境の理解を次の段階に進めることができます。ここで、操作上のセキュリティおよび可用性の要件として、ファイヤウォール群から分離された Web サーバとアプリケーションサーバの複数のインスタンスを環境の基礎とするとします。そのため、DMZ と保護されたバックエンドのビジネスシステムとを区分できるようにするために、独立したマシンの層が必要になります。また、アプリケーションの可用性を高めるため、各層の複数のマシンのインスタンスについても計画する必要があります。

このような前提から操作環境が次第に絞り込まれますが、まだシステムに必要なコンピュータの正確な数やサイズには至りません。次の段階では、パフォーマンスの予測方法とシステムサイズの指定方法についての基本的理解を深めます。

パフォーマンスの予測

サイジングのプロセスに影響を与える要因と操作環境の概要レイアウトから、既存のハードウェアの組み合わせで得られる能力または特定の能力を維持するために必要な最小のハードウェアを、どのようにして予測したらよいのでしょうか。その疑問に答えるには、前の節の説明にあるデータを収集し、iPlanet Application Server に含まれているサイジングのカリキュレータでそのデータを使用するのが最良の方法です。

iPlanet には、iPlanet Application Server に配置されるアプリケーションをサイジングする際に役立つ、2 つのカリキュレータが用意されています。最初のカリキュレータは、前の節で説明した要因に基づいて、CPU の数や各層のマシン数などのシステムのサイズを計算します。2 番目のカリキュレータは、与えられたハードウェア設定の最大能力を計算します。

一般的なアプリケーションワークフロー用も含め、複数のテストの組み合わせに基づいて iPlanet はこれらのカリキュレータを構築し、RDBMS とプロセッサ用に広く使用できるベンチマークの結果を利用しています。2 つのカリキュレータとも、完全にチューニングされたシステムを前提としています。

サイジングのカリキュレータを使用する以外に、以下の手順に基づいて、アプリケーションのサイジングを理解することをお勧めします。

1. 1 つの CPU のパフォーマンスを判断する

   最初に、既知の処理能力で維持できる最大の負荷を決定する必要があります。この数値は、ユニプロセッサのマシンでアプリケーションのパフォーマンスを測定して得ることができます。似た特性のある処理を行う既存のアプリケーションのパフォーマンス数値、または理想的には、開発中に行われた基本的なパフォーマンステストの結果を使用します。

   単独 CPU でのパフォーマンスを判断する一方で、基本的な環境のチューニングを開始する必要があります。どのようなパフォーマンステストを使用しても、ドライバマシン、Web サーバ、データベースマシンなどのテスト範囲外のシステムがテストを妨害しないようにする必要があります。テストに妨害があった場合は、パフォーマンスの数値が不自然に低くなり、サイジングの結果に悪影響を及ぼすことがあります。

2. 垂直スケーラビリティの判断

   プロセッサを追加すると、どのくらいの追加パフォーマンスを得ることができるかを、正確に知っておく必要があります。つまり、サーバ上で該当のワークフローによって、共有リソースの競合が発生する回数を間接的に測定します。この情報は、マルチプロセッサシステムでアプリケーションに追加の負荷テストを実

行して入手することができます。または、すでに負荷テストが実行された類似したアプリケーションの情報を使用することもできます。最大 で 4 つの CPU 上で一連のパフォーマンステストを実行すると、通常は、システムの垂直スケーラビリティの特性に関するまずまずの結果が提供されます。

サイジングの見積もりに基づき、目標とする設定が行われたシステムの負荷のもとで、アプリケーションを実行してみることが重要です。垂直スケーラビリティを判断する一方、可用性の要件を満たした設定がされていることも確認してください。たとえば、1 つの JVM で障害が起こっても、すべてのセッションが喪失されないように保証するには、最低 2 つの JVM で垂直スケーラビリティのテストを実行し、これら JVM 間でセッションの複製を設定してください。

3. 水平スケーラビリティの判断

   サーバを追加するとどのくらいの追加パフォーマンスを得ることができるかを、知っておく必要があります。類似したアプリケーションの情報がまだ利用できない場合は、アプリケーションサーバシステムのクラスタのベンチマークが再び必要になります。水平スケーラビリティテスト環境の概略を作成する際に、高度な可用性の要件に付随するセッション複製の設定を必ず考慮してください。

   この場合、アプリケーションサーバの各インスタンス内の JVM にセッションの複製が作成されるほか、複数のマシンに配置されたアプリケーションサーバのインスタンスにまたがってセッションが複製されます。この一連のテストを実行すると、アプリケーションサーバのパフォーマンスがはっきりと理解できます。この情報を使用して、独自のサイジング式を作成できます。

   最初に、既知の処理能力で維持できる負荷を判断する必要があります。この数値は、マルチプロセッサについて推定するためにユニプロセッサでアプリケーションのパフォーマンスを測定して得ることができます。

単一インスタンスのインストールという基本的な高可用性 (HA) サイトのサイジング式は、次のとおりです。

```
TotalProcessorCount =(P*(1.0-k) * (ks+P*k*K*(1.0-2.0 * K) +sqrt(4.0
*ks*K*(1.0-K)*(ks+P*k*(1.0-2.0*K)))) / ((ks-P*k*K) * (ks-P*K*K));
```

ここで、

- `P` = ピーク時の負荷に必要なパフォーマンス (スループット)。CPU 利用率の最大レベルが指定されている場合は、その値で `P` を除算します。たとえば、10,000 ユーザの 80% を超えてサーバがビジーになってはならない場合は、`P = 10,000/0.80 = 12,500` とします。
- `k= (1 - CPU スケーラビリティ ) / (1+ CPU スケーラビリティ )`。CPU スケーラビリティは、異なる CPU 数のサーバでアプリケーションを実行して測定します。この場合の CPU スケーラビリティは、2 番目のプロセッサを追加して獲得できる追加のパフォーマンスのことです。CPU スケーラビリティは、ワークフロー、アプリケーションサーバソフトウェア、オペレーティングシステム、およびサーバハードウェアに左右されます。

- ks= CPU 自体のパフォーマンス。これは、このハードウェア上の該当するアプリケーションで必要とされるスループット (P) です。ks は、少なくとも 1 台のプラットフォームで測定され、測定されたプラットフォームに対するものとして新しいプラットフォーム上の SPECint_base95 パフォーマンス率を使用して推定されます。
- K = (1 - サーバのスケーラビリティ)。「(1 + サーバのスケーラビリティ) - サーバのスケーラビリティ」は、異なるサーバ数またはアプリケーションサーバのインスタンス数を使用して、クラスタ上でアプリケーションを実行して測定します。この場合のサーバのスケーラビリティは、最初のサーバと同じ 2 番目のアプリケーションサーバを追加して獲得できる追加のパフォーマンスのことです。

この式は、必要な合計 (最低) プロセッサ数の見積もりです。またこの値は、設定が最適化されることを想定したものです。これらの値はアプリケーションによって異なることに注意してください。最適なクラスタ数、アプリケーションサーバのインスタンス数、およびサーバごとのプロセッサ数を識別するために、さまざまな式の値が使用されます。この値は、1 つのサーバがダウンしても必要なピーク時の負荷を満たす必要があることも想定されています。

単一サーバのサイジングのもう 1 つの方法は、サイトに必要なスループットのレートを確立して、ワークフローが以下の 3 つの例のどれかに適合するかどうかを判断することです。

- Servlet で実装されるオンラインストア。共通してアクセスされるページはキャッシュされる
- Servlet で実装されますが、共通してアクセスされるページがないオンラインストア。そのため、キャッシュは無効にされる
- JSP、Servlet、セッション、およびエンティティ Beans の混合で実装されるオンラインストア。キャッシュは無効にされる

一般的なパフォーマンスのガイドライン

次の表は、サイジングとアプリケーションのパフォーマンスに影響を与える要因を示しています。

表 2-2 サイジングに影響を与える要因 - 概念の適用

概念	概念の適用	測定単位	値の根拠
ユーザ負荷	最大同時セッション	トランザクション速度 (RPM、WIPS)	最大同時ユーザ数 / クリック間隔
			加入者のセッション時間数 / セッション間隔
アプリケーションの設計と実装	CPU パフォーマンス単位ごとのトランザクション速度	SPECint_base95 ごとの RPM	ベンチマークによる測定
	サーバ内のスケーラビリティ (追加 CPU のパフォーマンス)	%	ベンチマークの適合曲線に基づく %
	クラスタ内のスケーラビリティ (追加サーバのパフォーマンス)	%	ベンチマークの適合曲線に基づく %
ハードウェアプラットフォーム	プロセッサのパフォーマンス (通常は整数パフォーマンス)	SPARC®20@40Mhz のパフォーマンス率	SPECint_base95
	スケーラビリティ (多くの場合、キャッシュ設計とメモリ帯域)	%	ベンチマークの適合曲線に基づく %
安全マージン	高可用性要件	当てはまる場合は、サーバシステムが 1 つダウンしたという前提でシステムのサイズを指定	高可用性が要求される場合は別の式を使用
	% ビジー高水位	%	通常は 80%、許容可能なリスクのレベルを計算する必要がある

パフォーマンスチューニングの手順

iPlanet Application Server とその関連要素は、次の手順でチューニングすることをお勧めします。

- アプリケーションのチューニング
- iPlanet Application Server のチューニング
- Java 実行時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
- データベースサーバのチューニング

これらのトピックについては、次の章で詳しく説明します。

第 3 章

アプリケーションのチューニング

この章では、最大のパフォーマンスを得るためにアプリケーションをチューニングする包括的な手順を示します。この章には次のトピックがあります。

- Java コーディ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 コーディングのガイドライン

この節では、Java コーディングに関する事項とパフォーマンスに関連する事項を扱います。この節で扱うガイドラインは iPlanet™ Application Server だけに当てはまるものではなく、Java アプリケーションのコーディング全般に当てはまる規則です。

- 直列化と直列化解除の回避

  Java でオブジェクトを直列化したり直列化を解除したりすると、CPU に負担がかかり、アプリケーションの速度が低下しがちになります。

- アレイの回避

  実行するのは難しいですが、Java ではアレイの使用は最小限にとどめてください。Java の主要目的の 1 つは安全性であり、C や C++ のプログラマを悩ませる問題の多くは、Java で生じることはありません。Java のアレイは初期化されていることが保証され、その範囲外にアクセスすることはできません。

  範囲の検査には、実行時の索引検証と各アレイでわずかのメモリオーバーヘッドが必要です。最小限の時間しか使用しませんが、多数のアレイがコードで使用される場合はパフォーマンス低下の要因となります。

- `StringBuffer.append()` を使い、「+」演算子は使わない

  Java の文字列は不変です。つまり、一度作成すると変更できません。たとえば、次のシーケンスは、

  ```
  String str = "testing";
  ```

```
str = str + "abc";
```

コンパイラで解釈すると次のようになります。

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

このように、コピーは本質的に時間のかかる作業であり、多用するとパフォーマンスに影響を与える原因になります。`StringBuffer.append()` の使用をお勧めします。

- 遅延処理変数に `null` 値を明示的に割り当てる

  これは、ガベージコレクタが安全に再生できるメモリ部分を簡単に識別するのに役立ちます。Java では、メモリを多量に使ったり、大量のメモリを循環させる (たとえば、多数のオブジェクトを作成して遅延処理をする) ことができます。参照を解放しないで、オブジェクトを保持したままでメモリリークさせることができます。これにより、ガベージコレクタはオブジェクトの再要求を中止し、その結果、使用されるメモリ量が増加します。このように、変数に `null` を設定して明示的に遅延処理することでパフォーマンスが向上します。

J2EE プログラミングのガイドライン

J2EE モデルにより、アプリケーション開発のフレームワークが定義されます。定義されるのは、アプリケーションアーキテクチャにおける JSP、Servlet、および EJB (JNDI、JMS、および JTS を含む) の使い方です。J2EE モデルの全部品にはそれぞれの使い方がありますが、アーキテクチャの設計にあたり、次の点に留意しておく必要があります。

- Servlet プログラミングのガイドライン

  iPlanet Application Server のアプリケーションはすべて JSP または Servlet (EJB へのエントリポイントでもある) によって提供されます。デフォルトモデルである Servlet マルチスレッドモデルの場合は、JVM ごとに Servlet のインスタンスが 1 つ作成されます。この JVM にある Servlet のリクエストはすべて同じ Servlet インスタンスを使います。これにより、スレッドの競合が発生することがあります。このため、クラス変数を使うと同期上の問題が起こるので、使わないようにします。

  また、コードやメソッドに対して同期句を使うとコードに危険な個所が作成されるので、使わないようにします。同期ブロックでは一度に 1 つのスレッドしか実行できません。残りのスレッドはブロックされ、アクセスできるようになるまで待機します。この待機キューは、高パフォーマンスの Web サイトにとってパフォーマンスを低下させる重大な要因となります。

- EJB 使用の回避

  EJB はサービスの再利用という点では非常に便利です。しかし、この柔軟性にはコストがかかります。これは、コンテナの実装における特性によるものではなく、EJB の動作を設計しているメソッドのためです。ここでは iPlanet Application Server がコンテナになります。J2EE アプリケーションでは、EJB への全リクエストがしばしば Servlet を経由します。

  Servlet は JNDI 検索を行い、Beans 参照を取得し、この参照を使って Beans メソッドを呼び出します。通常、参照はキャッシュされ、以後ヒットしたすべての場合に使われます。EJB にアクセスするには事前に実行する必要のある指示が多くあるので、EJB は同じタスクを実行する Servlet よりも本質的に時間がかかることがわかります。

  EJB を使う必要がある場合は、次の手順に従って応答時間を短くしてください。

  - EJB 参照を Servlet でキャッシュします。こうすると、リクエストごとに JNDI 検索を行う必要がなくなります。
  - 次に、EJB の種類をパフォーマンス順に示します。パフォーマンスがもっとも高い Beans タイプが最初になっています。
    - 状態のないセッション Beans
    - 状態のあるセッション Beans
    - CMP (コンテナ管理パーシスタンス) を持つエンティティ Beans
    - BMP (Beans 管理パーシスタンス) を持つエンティティ Beans

    EJB の中では状態のないセッション Beans がもっとも速く、パフォーマンスでは Servlet に匹敵します。
  - 状態のあるセッション Beans ではスティッキーロードバランスを使用します。スティッキーロードバランスを使用していない場合、セッションに格納されている Beans 参照がほかの iPlanet Application Server に転送されると、コンテナ間検索を行ってリクエストを処理する必要があります。これは非常に時間がかかります。
  - パフォーマンステストの結果に基づいてコンテナ (iPlanet Application Server) をサイジングします。プロセスのスレッドを作成し、iPlanet Application Server のタイムアウト値を設定します。EJB キャッシュを設定するとパフォーマンスが向上します。
  - Servlet と JSP を iPlanet Web Server ではなく iPlanet Application Server に配置します。次の場合はアプリケーションを iPlanet Application Server に配置します。
    - アプリケーションに高度なトランザクション機能がある
    - セッションデータを保護するためのフェールオーバーサポートが必要である
    - レガシーデータにアクセスする

- アプリケーションの状態がほとんどなく、そのアプリケーションが読み取り専用であり、トランザクション型でない場合は、アプリケーションを iPlanet Web Server に配置します。
- EJB をプレゼンテーションロジック (Servlet および JSP) とともに同じサーバ上に配置するようにサーバ管理者に依頼して、アプリケーション実行時のリモートプロシージャコール (RPC) の数を減らします。

注　アプリケーションを中程度から多数の EJB に分割すると、アプリケーションパフォーマンスが著しく劣化し、オーバーヘッドが増加します。EJB は JavaBeans と同様、単なる Java オブジェクトではありません。EJB は Java オブジェクトよりも高レベルです。これらの EJB は、リモート呼び出しインターフェイスセマンティック、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ック、およびプロパティから構成されるコンポーネントです。

第 4 章

iPlanet Application Server のチューニング

この章では、最大のパフォーマンスを得るために iPlanet™ Application Server をチューニングする包括的な手順を示します。この節では次のトピックについて説明します。

- サーバプロセスのパフォーマンスの最適化
- 分散セッションと Lite HTTP セッションの比較
- 高可用セッションの単一バックアップ設定
- Dsync セッション管理スレッドの設定
- ロードバランスのオプション
- データベースコネクションプールの設定
- 実行時の EJB パラメータの設定
- JSP と Servlet のキャッシュ

サーバプロセスのパフォーマンスの最適化

Executive Server (KXS)、Java エンジン (KJS)、C++ エンジン (KCS)、および RMI/IIOP ブリッジプロセス (CXS) が iPlanet Application Server の核を形成しています。この節では、こうしたプロセスをチューニングしてパフォーマンスとスケーラビリティを最大にする方法について説明します。

この節では、次のトピックについて説明します。

- iPlanet Application Server プロセスのチューニング
- RMI/IIOP のパフォーマンスチューニング

iPlanet Application Server プロセスのチューニング

Executive Server (KXS)、Java エンジン (KJS)、および C++ エンジン (KCS) は、ワーカースレッドのプールを使ってリクエストを非同期的に処理します。これらのスレッドは、アプリケーションコンポーネントへのユーザのリクエストを処理します。iPlanet Application Server はリクエストを受け取ると、利用可能なスレッドにそのリクエストを割り当てます。スレッドはリクエストに対するシステムのニーズを管理します。たとえば、現在ビジー状態のシステムリソースが必要な場合、スレッドはそのリソースが解放されるのを待ってからリクエストにそのリソースの使用を許可します。

リクエストのスレッド数は、iPlanet Application Server のそのインスタンスが使用するすべてのプロセスについて、グローバルに調整できます。これは、プロセスレベルでも調整できます。

注	プロセスレベルの設定値はサーバレベルの設定値より優先されることに注意してください。設定のチューニングは、iPlanet Application Server Administration Tool を使って行います。

この節では次のトピックについて説明します。

- KXS パフォーマンスの最適化
- KJS パフォーマンスの最適化
- リクエストスレッド数の調整
- サーバおよびエンジンの最大シャットダウン時間の指定

KXS パフォーマンスの最適化

Web コネクタプラグインは、iPlanet Application Server アプリケーションへのユーザリクエストを実行プロセス (KXS) に送ります。これらのリクエストは実行プロセスのリクエストキューに記録されます。

次のタスクを実行して、KXS パフォーマンスを最適化できます。

- Web コネクタプラグインがプロセスリクエストに対して使う最大スレッド数の制御。これにより、リクエストキューが処理能力以上のリクエストを受け付けないようにします。iPlanet Application Server のインストールでは、KXS のスレッド数はデフォルトで 32 に設定されます。スレッドは 128 まで増やせますが、64 あれば十分です。スレッドを増やす場合は KXS を少なくとも 1 つのプロセッサにバインドし、mutex ロックを取得するスレッドで時間を浪費しないようにすることをお勧めします。

- リクエストのフローを制御するための、リクエストキューに記録する最大リクエスト数の設定。最大数を「高水位」と呼びます。
- ログが再開されるキューのリクエスト数の設定。この数を「低水位」と呼びます。
- 1 つのプロセッサまたはプロセッサセットへの KXS のバインド。これは、負荷テスト中にリクエストを KXS でキューに入れた場合にだけ実行します。これは KJS には実行しないでください。iPlanet Application Server では、JDK はマルチプロセッサ用に最適化されており、1 つのプロセッサにバインドしてもパフォーマンス上の利点がないためです。また、1 つのプロセッサにバインドしても KXS のパフォーマンスが向上しない場合は (KXS プロセスの CPU 使用率が高い場合など)、2 つのプロセッサを使って 1 つのプロセッサセットにすることができます。こうすると、KXS はこのプロセッサセットにバインドされます。

Executive Server (KXS)、Java サーバ (KJS)、C++ サーバ (KCS)、CORBA Executive Server (CXS) などのサーバプロセスが失敗すると、Administration Server によって再起動されます。再起動オプションを使うと、プロセスが再起動される回数を増減できます。フォールトトレランスとアプリケーションの可用性は、すべてのプロセスがスムーズに実行されているときに向上します。

KJS パフォーマンスの最適化

iPlanet Application Server のインストール時に、KJS スレッド数は 32 に設定されます。スレッドは 48 まで増やすことができます。KJS のスレッドは 48 が最適な設定数です。

リクエストスレッド数の調整

デフォルトのスレッドプールは、プロセスあたり 8 スレッドです。最大 32 まで設定できます。アプリケーションからのリクエスト用に確保するスレッドの最小数と最大数を指定することができます。スレッドプールはこれらの 2 つの値の間で動的に調整されます。アプリケーションリクエストに備えて、ユーザが指定した最小のスレッド数が維持されます。スレッドは、ユーザが指定した最大値まで増加します。

プロセスで使用可能なスレッドの数を増やすと、そのプロセスは同時により多くのアプリケーションリクエストに応答できるようになります。サーバレベルで、各プロセスのスレッドの追加や調整を行ったり、そのサーバのすべてのプロセスのスレッド数を定義したりできます。

これらのパラメータの最適な設定は、アプリケーションによって異なります。たとえば、リクエストに大量のデータベース処理が含まれており、データベースサーバのハードウェアへの負荷が軽くて同時処理が増えても処理できる場合は、プールサイズを多少大きくして大量のリクエストをデータベースに送り、スループットを増加させるようにします。一般に、スレッドプールのサイズは、KXS プロセスでも KJS プロセ

スでも 32 ～ 48 スレッドまでは、大きいほどよいと考えられます。プールサイズの最小値と最大値もこの数に合わせて最初は 32 に設定し、必要に応じて 48 で試してみることをお勧めします。必要なパラメータは、iASAT を使って一度にすべて設定することもできます。

デフォルトでは、各プロセスは iPlanet Application Server に割り当てられているスレッドを使います。たとえば、iPlanet Application Server が最小 8、最大 64 のスレッドを使う場合、個々のプロセスは最小 8、最大 64 のスレッドを使います。

サーバおよびエンジンの最大シャットダウン時間の指定

iPlanet Application Server とエンジンプロセスの両方に、Administration Server の最大エンジン再起動回数を設定できます。たとえば、エンジンシャットダウン時間を 60 秒に設定すると、処理中のアプリケーションタスクは処理を完了するのに 60 秒まで認められます。この期間の経過後は、新しいリクエストを受け付けません。シャットダウン値を指定することによって、クライアントにエラーを返す「ハード」シャットダウンを回避します。値の設定は、iPlanet Application Server Administration Tool を使って行います。

最大サーバシャットダウン時間 :「最大サーバシャットダウン時間」は、iPlanet Application Server をシャットダウンするまでの最大時間です。この時間を過ぎると、実行中のすべてのエンジンが強制終了されます。通常、負荷が大きくないかぎり、サーバはすぐにシャットダウンされます。

最大エンジンシャットダウン時間 :「最大エンジンシャットダウン時間」は、iPlanet Application Server がエンジンのシャットダウンを待つ最大時間です。この時間を過ぎると、エンジンは強制終了され、次のエンジンがシャットダウンされます。

すべてのログのスイッチオフ

入力と出力を繰り返すことによるシステムへの負担を軽減するために、KJS ログに記録されるアプリケーションログをすべてスイッチオフします。これにより、著しくパフォーマンスが向上します。

MaxBackups = 1 の設定

通常のアプリケーションサーバアーキテクチャでは、Sync Backup サーバが 1 台あるとクラスタ内通信の数が減少します。1 つのクラスタの `Maxbackups` (最大バックアップ数) を 1 に設定します。0 に設定すると、プライマリが利用不能になったときのセッションのバックアップはありません。2 に設定するとクラスタ内通信が増加し、サーバの負荷が大きくなります。そのため、このパラメータの設定は 1 が最適です。

RMI/IIOP のパフォーマンスチューニング

RMI/IIOP パスで多数の同時ユーザに対応しなければならない配置環境の場合は、この節で説明するチューニングを実行してください。RMI/IIOP を使う場合、JVM のデフォルト設定とその基本 OS だけでは最適なパフォーマンスおよび容量を達成できません。

この節には次のトピックがあります。

- パフォーマンス問題の認識
- 基本的なチューニング方法
- スケーラビリティの向上
- RMI/IIOP のファイヤウォールの設定

パフォーマンス問題の認識

RMI/IIOP クライアントアプリケーションを負荷状態で実行する前に、基本構造のテストが成功していることを確認してください。

負荷状態のクライアントアプリケーションの実行を開始する際、RMI/IIOP クライアントに次の例外が発生する場合があります。

```
org.omg.CORBA.COMM_FAILURE
```

```
java.lang.OutOfMemoryError
```

```
java.rmi.UnmarshalException
```

アプリケーションの基本構造は正しく動作することが確認されている場合に、アプリケーションの負荷テスト時にこれらの例外が発生した場合は、次の節で説明する RMI/IIOP 環境のチューニング方法を参照してください。

基本的なチューニング方法

次に説明するチューニング方法を試し、ユーザの環境に最適なバランスを見つけてください。

Solaris ファイル記述子の設定 : Solaris の場合、ulimit を使って、開いているファイル数のプロパティを最大に設定すると、サポートできる RMI/IIOP クライアントの最大数に影響を与えます。このプロパティのデフォルト値は、Solaris 2.6 または Solaris 8 のどちらを実行しているかによって、64 または 1024 となります。制限数を増やすには、次のコマンドを /etc/system に追加し、再起動します。

```
set rlim_fd_max = 8192
```

次のコマンドを使うと、この使用制限を確認できます。

```
ulimit -a -H
```

上記の使用制限を設定後、次のコマンドを使うと、このプロパティの値をこの制限まで明示的に増やすことができます。

```
ulimit -n 8192
```

次のコマンドを使うと、この制限を確認できます。

```
ulimit -a
```

たとえば、ulimit がデフォルトの 64 の場合、1 つのテストドライバがサポートできる同時クライアントは 25 ですが、ulimit を 8192 に設定すると、同じテストドライバで 120 の同時クライアントをサポートできます。テストドライバは複数のスレッドを生成します。これらの各スレッドは JNDI 検索を実行し、ビジネスメソッド呼び出し間の思考 (遅延) 時間が 500 ミリ秒の同じビジネスメソッドを繰り返し呼び出し、約 100 KB のデータを送受信します。

これらの設定値は RMI/IIOP クライアント (Solaris)、および Solaris システムにインストールされた RMI/IIOP ブリッジに適用されます。ファイル記述子の制限の設定については、Solaris のマニュアルを参照してください。

Java ヒープ設定値 : ファイル記述子の容量をチューニングするだけでなく、クライアントおよびブリッジ JVM の両方について異なるヒープ値も設定できます。詳細については、第 5 章「Java 実行時システムのチューニング」を参照してください。

スケーラビリティの向上

1 つのブリッジプロセスおよびクライアントシステムの容量をチューニングするだけでなく、複数の RMI/IIOP ブリッジプロセスを使うことによって、RMI/IIOP 環境のスケーラビリティを向上させることができます。同じアプリケーションサーバインスタンス上で複数のブリッジプロセスを設定すると、アプリケーション配置のスケーラビリティが向上します。場合によっては、それぞれ 1 つまたは複数のブリッジプロセスを使って設定した多数のアプリケーションサーバインスタンスを使うこともできます。

複数のブリッジプロセスがアクティブな設定では、一連のクライアントをさまざまなブリッジにスタティックにマッピングするか、またはクライアントサイドに独自のロジックを実装して既知のブリッジプロセスと照らし合わせてロードバランスを行うことによって、クライアント負荷を分割できます。

RMI/IIOP のファイヤウォールの設定

RMI/IIOP クライアントがファイヤウォールを通過して iPlanet Application Server と通信する場合は、RMI/IIOP ブリッジプロセスが使うクライアントシステムから IIOP ポートへのアクセスを有効にする必要があります。クライアントのポート番号は動的に割り当てられるため、RMI/IIOP トラフィックがクライアントシステムからファイヤウォールを通過してアプリケーションサーバのインスタンスに達するように、ソースポートの範囲を広げ、1 つのデスティネーションポートを開く必要があります。

さらに、Converter サンプルアプリケーションを一度実行すると、2 つのシステム間の IIOP トラフィックが巡回ベースで追跡されます。ホスト `swatch` は RMI/IIOP クライアント、ホスト `mamba` は、デスティネーションシステムまたはアプリケーションサーバシステムです。RMI/IIOP ブリッジプロセスに割り当てられているポート番号は 9010 です。動的に割り当てられた 2 つのポート (`33046` および `33048`) は RMI/IIOP クライアントで使われます。一方、ブリッジプロセスとの通信にはポート `9010` が使われます。

```
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Syn
Seq=140303570 Len=0 Win=24820

Options=<nop,nop,sackOK,mss 1460>

mamba.red.iplanet.com -> swatch TCP D=33046 S=9010 Syn
Ack=140303571 Seq=1229729413 Len=0 Win=8760

Options=<mss 1460>

swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729414
Seq=140303571 Len=0 Win=24820

swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729414
Seq=140303571 Len=236 Win=24820

mamba.red.iplanet.com -> swatch TCP D=33046 S=9010 Ack=140303807
Seq=1229729414 Len=168 Win=8524

swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729582
Seq=140303807 Len=0 Win=24820

swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Syn
Seq=140990388 Len=0 Win=24820

Options=<nop,nop,sackOK,mss 1460>

mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Syn
Ack=140990389 Seq=1229731472 Len=0 Win=8760

Options=<mss 1460>

swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731473
Seq=140990389 Len=0 Win=24820

swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731473
Seq=140990389 Len=285 Win=24820

mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140990674
Seq=1229731473 Len=184 Win=8475

swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731657
Seq=140990674 Len=0 Win=24820

swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731657
Seq=140990674 Len=132 Win=24820
```

```
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140990806
Seq=1229731657 Len=25 Win=8343
```

分散セッションと Lite HTTP セッションの比較

分散セッションではセッションデータが iPlanet Application Server のバックアップノードに複製されるので、可用性が向上します。そのためには、分散セッションに配置されるオブジェクトが `java.lang.Serializable` インタフェースを実装している必要があります。

ただし、セッションサイズが大きく、頻繁に書き込まれる場合は、ネットワークのトラフィックが増大します。キューが使用されていることで分散セッションにアクセスする JSP と Servlet のパフォーマンスも低下しますが、その規模は使用する `HttpSession` の数とサイズによって異なります。Lite セッションの可用性とパフォーマンスはトレードオフの関係にあります。セッションオブジェクトは KJS プロセスにローカルにキャッシュされます。KJS プロセスはそのセッションの全リクエストを保存する場所として機能します。

Java オブジェクトが `HttpSession` に格納される場合は、Lite セッションにも利点があります。Lite セッションのオブジェクトはそれぞれの KJS プロセスにキャッシュされるため、ネットワークの Java オブジェクトを直列化する必要がありません。このため、HTTPSession に Java オブジェクトが格納されると、このオブジェクトは `java.lang.Serializable` インタフェースを実装する必要がなくなります。

必要なセッションタイプは `ias-web.xml` の `<session-info>/<impl>` プロパティを編集して指定します。

注　Lite セッションに切り換えることによるパフォーマンスの向上は、8 ～ 40% と報告されています。

高可用セッションの単一バックアップ設定

これは、高可用セッション (dsync 分散) を使い、HttpSession フェールオーバーにプライマリとセカンダリを設定している場合だけに当てはまります。クラスタに対するレジスタ設定の Maxbackups は、1 にする必要があります。0 に設定すると、プライマリが利用不能になったときのセッションのバックアップはありません。2 に設定すると、クラスタの内部通信が増加します。このパラメータの最適な設定値は 1 です (デフォルト)。

このパラメータは、次の iPlanet レジストリキー内に設定できます。

```
Software¥iPlanet¥Application
Server¥6.0¥Clusters¥<cluster-name>¥MaxBackups
```

Dsync セッション管理スレッドの設定

Dsync は、分散ステート同期サービスです。指定した Dsync プライマリとバックアップにはセッションノードのメモリデータベースが含まれています。HTTP セッションがタイムアウトもしくは無効になった場合、メモリ内のデータベースからすぐに削除する必要があります。専用スレッド数を増やすと、削除と記録をほとんど同時に行うことができます。

レジストリで次のプロパティを設定してください。メモリが一時的に増加する問題を解決し、セッションノードをより効率的に無効化できます。次のように設定すると、KXS 削除スレッドが iPlanet レジストリ内で 20 になります。

```
Software¥iPlanet¥Application
Server¥6.0¥CCSO¥ENG¥0¥SyncTimeoutThreadCount=20
```

KJS エンジン #1 にも同じことをする場合は、iPlanet レジストリ内で次のキーを変更します。

```
Software¥iPlanet¥Application
Server¥6.0¥CCSO¥ENG¥1¥SyncTimeoutThreadCount=20
```

スレッドがアクティブな場合は次の方法で間隔を設定します。

```
Software¥iPlanet¥Application
Server¥6.0¥CCSO¥ENG¥<n>¥SyncTimerInterval
```

デフォルトは 60 です。この設定で十分です。

注 この方法で直接パフォーマンスが向上するわけではありませんが、小さなセッションオブジェクトを保存しておくと必ず役に立ちます。また、すぐに効率よく削除すると、KXS プロセスと KJS プロセスでのメモリリークを防ぐこともできます。

セッションを削除するために HttpSession タイムアウトに頼らないことをお勧めします。削除には HttpSession.invalidate() メソッドを使います。このメソッドが使えない場合は、配置環境でのデフォルトの HttpSession タイムアウトをできるだけ小さく設定してください。

ロードバランスのオプション

この節では、次のトピックについて説明します。

- クラスタ設定のロードバランス
- 情報のブロードキャストと更新
- ロードバランス情報の監視
- 奨励するクラスタのロードバランス設定
- クラスタのセッションサイズの最適化
- 個々の JSP のロードバランス
- スティッキーセッションロードバランスの使用

クラスタ設定のロードバランス

iPlanet Application Server 環境では、クラスタはステートとセッションの情報を共有して保存するサーバコレクションとして定義され、データ同期サービス (Dsync) によって実行されます。

したがって、Dsync は指定したクラスタのサイズを制約する共有リソースです。一般に、iPlanet Application Server の各クラスタは最大 4 個のインスタンスで実行されます。それ以上のサーバが必要な場合は、Web サーバ層のスティッキーロードバランスを使ってクラスタを複数にします。ステート情報とセッション情報が Dsync に格納されていない場合は、並行して実行できるサーバ数に制限はありません。最適な設定は、iPlanet Application Server Sizing Tool を使って見つけます。

クラスタ設定のロードバランスには次の 2 つのシナリオが考えられます。

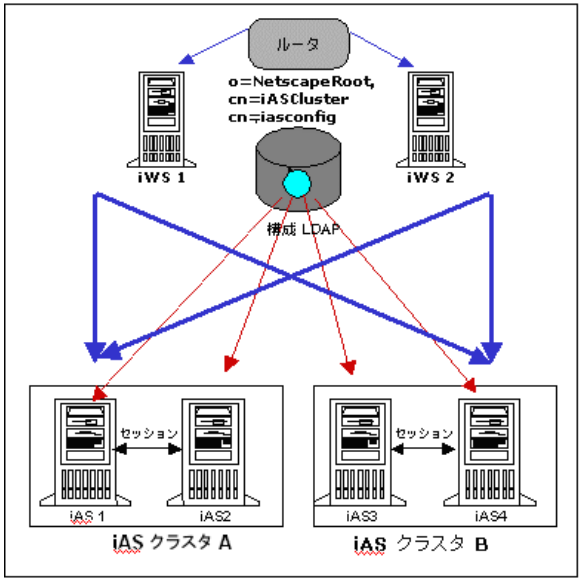
シナリオ 1:

2 つの iPlanet Application Server クラスタが 1 つの LDAP 設定ツリーを共有する。

このシナリオでは、すべての iPlanet Application Server が同じ LDAP 構成ツリーを共有しています。スティッキーロードバランスをサポートするために、スティッキーロードバランスメソッドをルータで作動させる必要はありません。

次の図にこの設定を示します。この図では、受信したリクエストは青い矢印間で設定 LDAP を共有することで示され、LDAP のアクセスは赤い矢印で示されています。

図 4-1　1 つの LDAP 設定ツリーを共有する 2 つのアプリケーションサーバ



奨励する使用法 :

この設定は各アプリケーションが 1 つのクラスタだけに存在する場合に有効です。このシナリオには、アプリケーションを分離し、Web 層を単純化する利点があります。

設定 :

iPlanet Application Server のインストール時に、すべてのクラスタのすべての iPlanet Application Server に同じ LDAP と同じ設定ルートを指定します。

シナリオ 2 :

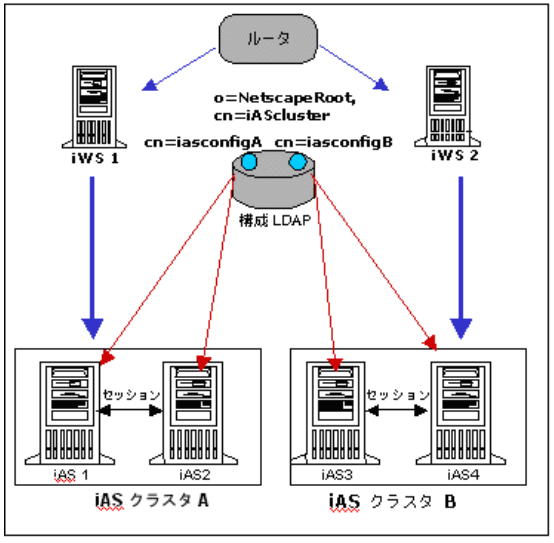
iPlanet Application Server クラスタを別々の LDAP 設定ツリーに割り当てる。

このシナリオでは、各 iPlanet Application Server クラスタの LDAP 設定ツリーに枝があります。各 iPlanet Application Server クラスタには、専用の iPlanet Web Server が少なくとも 1 つあります。リクエストが iPlanet Web Server に到着すると、iPlanet Web Server は 1 つの iPlanet Application Server クラスタしか知らないため、リクエストは常にクラスタ内の iPlanet Application Server に送信されます。ルータの役目は、個々の iPlanet Web Server 間の負荷を分散することです。

スティッキーロードバランスのサポートを有効にするには、ルータのスティッキーオプションをオンにして、後続のリクエストも同じ iPlanet Web Server に送信されるようにする必要があります。

次の図にこの設定を示します。この図では、受信したリクエストは青い矢印間で設定 LDAP を共有することで示され、LDAP のアクセスは赤い矢印で示されています。

図 4-2　別々の LDAP ツリーに割り当てられた 2 つのアプリケーションサーバクラスタ



奨励する使用法：

この設定は、アプリケーションがクラスタ間配置を必要とする場合に有効です。このシナリオの利点は、1 つのクラスタで新しいバージョンのアプリケーションを使用できることと、1 つのクラスタで提供できる以上の処理能力を必要とするアプリケーションに対応できることです。

情報のブロードキャストと更新

ロードバランスが有効に機能するために、プロセスに関連する各サーバには、ほかのすべてのサーバについての最新情報がなければなりません。つまり、ロードバランスに影響を与える要因についての情報がすべての iPlanet Application Server マシンにブロードキャストされる必要があり、また、各 iPlanet Application Server マシンでは、ロードバランスの決定を行うためにこの情報を監視して更新する必要があります。情報をブロードキャストする頻度が多すぎると、ネットワークトラフィックが増え、応

答時間が遅くなる可能性があります。ただし、ロードバランス情報が頻繁に計算および更新されない場合は、ロードバランスを決定するために iPlanet Application Server が使用する情報が古くなるため、アプリケーションコンポーネントの負荷のリスクが適切に分散されなくなります。

ロードバランスについての意思決定を行うときに、2 つの主要なジレンマに直面します。

- どのくらいの頻度で iPlanet Application Server サーバがロードバランス情報を更新したらよいか
- どのくらいの頻度で各 iPlanet Application Server インストールがロードバランス情報をブロードキャストしたらよいか

更新の間隔 : ほとんどの場合、最低 5 秒間、最高 10 秒間が適切です。通常、安定した状態で各サーバでもっともよく使用されるアプリケーションコンポーネントの応答時間の 2 倍を、更新間隔の基準にします。たとえば、もっともよく使用されるアプリケーションコンポーネントが 5 秒間でリクエストに応答するシステムでは、更新間隔を 10 秒間に設定します。更新の頻度を多くするとサーバの作業量も増え、ロードバランスの特性さえも変えてしまうことがあります。この計算には注意を払ってください。非常によく使用されるアプリケーションコンポーネントの応答時間が 1.5 秒だったとしても、更新間隔を 3 秒に設定しないでください。

注	非常によく使用されるアプリケーションコンポーネントの応答時間が 1.5 秒だったとしても、更新間隔を 3 秒に設定しないでください。

ブロードキャスト間隔 : 上記で説明したように、ロードバランス情報のブロードキャストの頻度が多すぎると、ネットワークトラフィックが増えるだけでなく、iPlanet Application Server の作業負荷も増大します。これは、すべてのサーバが情報を記録して収集するためです。通常、更新間隔の値の 2 倍にブロードキャスト間隔の基準を設定します。

iPlanet Application Server Administration Tool 内のロードバランスのツールを使用して、更新間隔とブロードキャスト間隔を設定します。

ロードバランス情報の監視

ロードバランスの基準を設定するときは、精細なチューニングプロセスに忍耐強く臨んでください。ロードバランスの基準を最適な組み合わせにするには、一定期間、iPlanet Application Server の設定を注意深く監視することが必要です。その期間中に、ピーク時の負荷、リクエストの各タイプの割合、平均応答時間、ボトルネックなどの統計データを収集する必要があります。各システムが異なるパラメータと基準で配置

されているため、すべての iPlanet Application Server ユーザに対して、1 つの同じソリューションはあり得ません。iPlanet Application Server 配置のすべての側面と同様、サイトに配置された iPlanet Application Server システムの長期にわたるパフォーマンスを向上させる最適な基準を決定できるのは、管理者だけです。

ロードバランスについて、および iASAT を使用してロードバランスの基準を設定する方法については、『iPlanet Application Server 管理者ガイド』にある「ユーザリクエストのロードバランス」を参照してください。

奨励するクラスタのロードバランス設定

iPlanet Application Server のインストール時に、すべてのクラスタ内のすべての iPlanet Application Server に同じ LDAP を指定します。クラスタが 1 つの場合はすべての iPlanet Application Server に同じ設定ルートを指定し、クラスタが異なる場合は異なる設定ルートを指定します。

クラスタのセッションサイズの最適化

セッションサイズは、iPlanet Application Server クラスタのパフォーマンスにもっとも影響を与えます。iPlanet Application Server を数多くインストールした結果、パフォーマンスを最大にするにはセッションサイズを 4K 以下にすべきことがわかりました。セッションサイズが大きくてもシステムは機能しますが、パフォーマンスが低下します。パフォーマンスが低下する主な原因は、システムのプライマリとホットバックアップがセッションデータを同期させるために常に通信していることです。

次のいずれかの方法を使って、セッションのパフォーマンスを向上させることができます。

- セッションで最も重要な要素だけを格納する

  アプリケーション作成者が重要なデータを指定し、セッションにこのデータだけを格納します。分散する必要がないデータはセッションから除外されます。

- セッションにスティッキーロードバランスを使用する

  スティッキーロードバランスとセッション分散は同じ数式の別々の値ですが、関連があり、同時に有効にすることができます。スティッキーロードバランスでは、1 つのクライアントは常に同じ KJS に転送されます。このため、セッションではなく JVM メモリへのデータストレージが可能になります。セッションに格納する必要があるのはキー (ハッシュテーブルやアレイなどのキー) だけです。セッション内のデータ量が著しく減るため、パフォーマンスが向上します。

この設定では、クライアントのリクエストを扱う iPlanet Application Server が使えなくなると、リクエストはクラスタ内の別の iPlanet Application Server インスタンスに転送されます。キーはこのセッションにあるので、データを再作成してメモリに格納することができます。これにより、リクエストは新しい KJS に転送されます。

このオプションは、セッションのデータがアクセスされ LDAP サーバなどの二次ソースに格納される場合に有効です。

- 別のデータストアを使って大量のセッションデータを直列化する

  この概念では、システムに新しい変数、つまりデータベースが導入されます。ここで考えられているのは、セッションの大部分をデータベースに格納してセッションにはプライマリキーだけを格納するというものです。そうすればセッションは小さくなり、パフォーマンスが向上します。これには、データベーススキーマを注意深く計画し、データベースアクセスの速度を上げる適切な検索インデックスを作成することが必要です。

個々の JSP のロードバランス

iPlanet Application Server では、JSP のロードバランスを個別に行うことができます。これは XML 記述子で JSP に GUID を割り当てることによって実行され、Servlet に GUID を割り当てる方法に似ています (JSP の登録に関する節を参照)。JSP に GUID を割り当てると、Servlet と同じように iPlanet Application Server Administration Tool を使って JSP にロードバランスを行うことができるようになります。

スティッキーセッションロードバランスの使用

Web アプリケーションの Servlet にスティッキーロードバランスが設定されていると、Web パフォーマンスが最大になります。この設定では、アプリケーションサーバや Web コネクタプラグインのユーザセッションのロードバランスが取られます。セッションの最初のユーザリクエストは、最適な候補である KJS プロセスまたはサーバにロードバランスされます。これ以降、同じユーザのセッションは同じプロセスに送信されます。このため、追加された KJS プロセスはセッションオブジェクトをローカルにキャッシュし、パフォーマンスが向上します。スティッキーロードバランスは分散セッションでも使用できます。

Servlet や JSP をスティッキーにするには、ias-web.xml の `<servlet>/<servlet-info>/<sticky>` プロパティをオンに設定します。配置ツールとパッケージツールにより、自動的にスティッキーがオンになります。

注 パフォーマンスが 10% 前後向上したというテスト結果が報告されています。

セッションデータの単純化

次のような理由から、オブジェクトはセッションに格納した後で直列化します。

- iPlanet Application Server の分散セッションは単純なデータ要素で動作するように調整されています。規模の大きい直列化されたオブジェクトは体裁も悪く、パフォーマンスに著しい影響があります。セッションを最大限に活用するには、データは個別の単純なデータ要素として格納する必要があります。

Java コーディングの観点からは、直列化と直列化解除は時間がかかる操作なので避けた方が賢明です。

データベースコネクションプールの設定

iPlanet Application Server には、コネクションをプールする機能があります。この機能で、わずかなデータベースコネクションが多くのスレッドに多重送信されます。最初に使うときはコネクションは試験的に確立されますが、閉じられることはありません。すでに開かれているコネクションは、アプリケーションサーバのプロセスが存在しているかぎり再利用できます。

コネクションプールは、JDBC データソースとユーザの一意の組み合わせごとに作成されます。アプリケーションでコネクションを得るために、呼び出すごとにユーザとパスワードを変える `DataSource.getConnection (user,password)` を使用すると、同じデータソースに対して複数のプールを作成することができます。

このプールのサイズが、KJS に設定されているワーカースレッドの数よりもわずかに多いことを確認します。必要なデータベースについては、レジストリの `CacheInitSlots` プロパティと `CacheMaxConn` プロパティを設定します。

たとえば、iPlanet Application Server に含まれている Oracle JDBC ドライバを使う場合、iPlanet レジストリの次のキーでプロパティを変更します。

```
Software¥iPlanet¥Application Server¥6.0¥CCS0¥DAE2¥ORACLE_OCI
```

注 すべてのスレッドでユーザリクエストを同時に処理し、データベースにアクセスする必要がある場合は、プールのコネクション数と KJS プロセスのワーカースレッド数が同じである必要があります。データベースへのアクセスが集中するアプリケーションでは、明らかにパフォーマンスが向上します。

この節には次のトピックがあります。

- データベースコネクションプール設定のガイドライン
- コネクションプールを設定するための統計の使用

データベースコネクションプール設定のガイドライン

iPlanet Application Server 内のコネクションプールは、次のようなさまざまなレベルで設定できます。

- データソース XML ファイル内
- レジストリの変更
- Administration Tool による動的な設定

  設定可能なコネクションプールのパラメータについては、『iPlanet Application Server 管理者ガイド』を参照してください。

コネクションプールを設定する際は、次の一般的なガイドラインに従ってください。

- データベースのバックエンドごとに論理データソースを使用します。複数の論理データソースが同じバックエンドをポイントしている場合は、リソースが最適に利用されていない可能性があります。
- 回復時間は、主として、コネクションを解放しないアプリケーションがコネクションを回復するための時間なので、できるだけ大きく保ちます。回復時間の後に使用されたとしても、コネクションの回復で生じる副作用があるためです。
- `maxPoolSize` の値は、このデータソースから行おうとした物理的な接続数に合わせます。
- `minPoolSize` の値は、このデータソースにアクセスするデータベースに関連した同時接続ユーザの平均リクエスト数に合わせます。
- コネクションがプールからアプリケーションに与えられる前に、コネクションの妥当性がチェックされます。最初のチェックは `setAutoCommit` に基づく単純な妥当性に関してであり、次のチェックはテーブルに基づく妥当性に関してです。

  ほとんどの場合は、単純な妥当性のチェックで十分です。ただしある種の JDBC ドライバでは、単純な妥当性で不備なコネクションを認識することができません。

  そのため、データベースドライバが単純な妥当性チェックをサポートしているときにだけそれを使用し、データベースのバックエンドが適度にフェールセーフである場合は、`isSanityRequired` を `false` に設定します。
- アプリケーションで `DataSource.getConnection` (username,password) を呼び出し、ユーザ名とパスワードが毎回異なる場合は、コネクションプールの設定を非常に小さく保ちます。

コネクションプールを設定するための統計の使用

iPlanet Application Server では、コネクションプールを設定するために使用できる豊富なコネクションプールの統計がサポートされます。

統計の設定と収集については、『iPlanet Application Server 管理者ガイド』を参照してください。

一般的な提案には、次のものがあります。

- 「ドロップしたコネクション総数」がゼロでなく、「プール内のコネクション総数のピーク値」が `MaxPoolSize` に達していなかった場合は、データベースのバックエンドに余分なコネクションを与える余裕はない

  この問題を避けるために、データベースのバックエンドのコネクション数を増やして設定するか、`MaxPoolSize` の値を増やすことができます。

- 「キューサイズのピーク値」がゼロでない場合は、コネクションのリクエスト数が `maxPoolSize` より多く、コネクションのリクエストはキューに入れられる。「キューサイズのピーク値」が 5 より大きい場合は、maxPoolSize を増やすことが望ましい

- 「キャッシュのミス数」の数値がゼロ以上で、「プール内のコネクション総数のピーク値」が `maxPoolSize` に達していなかった場合は、`minPoolSize` をさらに大きな値に増やすことができる

実行時の EJB パラメータの設定

iPlanet Application Server では、EJB コンテナによって、ユーザ独自の EJB コンポーネントやほかのベンダコンポーネントを使って、分散アプリケーションを構築できます。企業向けに iPlanet Application Server を設定する場合は、EJB コンテナの宣言パラメータを設定する必要があります。たとえば、これらのパラメータで、EJB が指定された時間 (秒単位) アクティブでなかった場合に削除されるセッションタイムアウトを決定します。iPlanet Application Server Administration Tool を使ってこれらのパラメータを設定します。

設定可能な値は次のとおりです。

- デフォルトのセッションタイムアウト

  デフォルトのセッションタイムアウトは `14400` 秒です。これは、`HttpSession` オブジェクトが使われないために削除されるまでの間、オブジェクトが存在できる時間です。できるだけ小さい有効な値を設定します。状態のあるセッション EJB に適用されます。

- デフォルトの不活性化タイムアウト

デフォルトの不活性化タイムアウトは 60 秒です。Beans の作成率が非常に低く、しかも Beans サイズが大きい場合は、この時間を増やす必要がある場合があります。ただし、この値を増やしても、ほとんどの事例でパフォーマンスに影響がありません。

Beans はファイルシステムに対して不活性化されるので、ファイルシステムが極端に活発な場合は、不活性化も極端になり、このパラメータを調整すると効果がある可能性があります。この値はセッションタイムアウトの値より小さくする必要があります。

- メタデータのキャッシュサイズ

  メタデータのキャッシュサイズは 10 Beans です。これはホーム Beans のキャッシュが扱うサイズです。この値は、アプリケーションに存在する Beans の種類の数まで設定することができます。50 から 60 までであればほとんどのユーザアプリケーションを扱えます。`EJBHome` インスタンスがキャッシュされるので、同じホームインタフェースのその後の検索では、キャッシュの内容が使われます。

- 実装キャッシュサイズ

  実装キャッシュサイズは 10 のインスタンスに設定されています。N 個の同時ユーザセッションが状態のあるセッション Beans にアクセスする場合、この値が N 以上に設定されていることを確認してください。状態のないセッション Beans では、この値を KJS のスレッド数以上に設定しても効果がないことがあります。同じことがエンティティ Beans にも言えます。この iPlanet Application Server Administration Tool の設定は、配置されているすべての Beans に適用されるため、制御が大まかになります。

  最大キャッシュサイズは EJB の数以内です。

- タイマー間隔

  「タイマー間隔」で間隔を設定し、Beans 実装プールをスキャンして不活性化の候補を検索します。

  この間隔は iPlanet レジストリキーの `CCSO¥EB¥EbInterval` で指定することもできます。

  このパラメータで、エンティティおよび状態のあるセッション Beans の削除間隔を指定します。デフォルト値は 10 秒です。実験的な条件下では、タイマー間隔の設定値を小さくすると EJB コンテナが頻繁に停止して応答時間に影響が出ることが確認されました。設定値を極端に大きくすると、不活性化の時間が長くなります。

- フェールオーバー保存間隔

「フェールオーバー保存間隔」で、フェールオーバー用に設定されているすべてのアクティブな状態付きセッション Beans が状態を直列化し、Dsync メモリ内データベースに不活性化される間隔を指定します。これは時間がかかる操作で、Beans のサイズが大きすぎたり保存間隔が短すぎるとパフォーマンスに影響を与えます。

状態のあるセッション Beans のフェールオーバーサポートの設定と使い方については、『iPlanet Application Server 開発者ガイド』を参照してください。

サーバが失敗した場合、最後に保存された EJB のステートが復元されます。保存されたデータはクラスタ内にあるすべてのエンジンからアクセスできます。この値はサーバごとに設定され、Deployment Tool EJB 記述子エディタの「一般」タブで有効にしたフェールオーバーオプションで配置された EJB に適用されます。

JSP と Servlet のキャッシュ

各 iPlanet Application Server インスタンスの各 KJS エンジンによってキャッシュされる JSP ページの数を指定できます。JSP をキャッシュすることにより、アプリケーションの応答時間が最適化されます。

キャッシュサイズはページごとに設定されます。JSP キャッシュは、構造的な JSP の開発を目的にしています。Java エンジン内に JSP をキャッシュして、複数の JSP が含まれるマスター JSP を作成できます。各 JSP は、異なるキャッシュ基準を使ってキャッシュできます。たとえば、株式相場を表示するウィンドウや気象情報を表示するウィンドウなどを含んでいるポータルページでは、株式相場のウィンドウを 10 分間キャッシュし、気象通報のウィンドウを 30 分間キャッシュするというように設定できます。

JSP のキャッシュは結果キャッシュの追加機能なので、複数の JSP を取り込んである JSP を合成し、それぞれに異なるキャッシュ基準を持たせることができます。JSP には GUID があるため、使用できるようになった結果キャッシュを使って合成 JSP 自体を KXS にキャッシュすることができます (JSP の登録に関する節を参照)。

JSP のキャッシュには JSP 1.1 で提供されているカスタムタグライブラリのサポートを使います。キャッシュ可能な一般的な JSP ページは次のとおりです。

```
<%@ taglib prefix="ias" uri="CacheLib.tld"%>

<ias:cache>

<ias:criteria timeout="30">

<ias:check class="com.iplanet.server.servlet.test.Checker"/>

<ias:param name="y" value="*" scope="request"/>
```

```
</ias:criteria>
</ias:cache>
<%! int i=0; %>
<html>
<body>
<h2>Hello there</h2>
I should be cached.
No? <b><%= i++ %></b>
</body>
</html>
```

<ias:cache> および </ias:cache> はキャッシュの各制約条件を区切ります。<ias:criteria> タグはタイムアウト値を指定し、別のキャッシュ基準を囲みます。キャッシュ基準は <ias:check> と <ias:param> の両方のタグを使って表現できます。これらのタグの構文は次のとおりです。

<ias:criteria timeout="val" > は、キャッシュされた要素のタイムアウトを秒単位で指定します。キャッシュ基準はこれと終了用の </ias:criteria> <ias:check class="classname" /> の範囲内で指定されます。これはキャッシュ基準の指定メカニズムの 1 つです。classname は、「check」と呼ばれるメソッドを持つクラスを参照します。これは次のシグネチャを持ちます。

```
public Boolean check(HttpServletRequest req)
```

これは、要素がキャッシュされるかどうかを示すブール値を返します。

<ias:param name="paramName" value="paramValue" scope="request" /> :

このメカニズムでも各基準を指定することができます。

paramName は属性の名前です。この名前は setAttribute を使ってリクエストオブジェクトまたは URI に渡されます。これはキャッシュ基準として使用されるパラメータです。paramValue はパラメータの値で、キャッシュを実行するかどうかを決定します。これには次のような種類があります。

```
Constraint
Meaning
x = ""
```

x はパラメータまたは属性として存在している必要があります。

x = "v1|...|vk"、ここで vi は "*" にすることができます。

x のリクエストパラメータが、キャッシュされたバッファを格納するときに使われた値と同じ値を持つとき、現在のリクエストの制約は true になります。

`x = "1-u"`、ここで `1` および `u` は整数です。

`x` は、`[1,u]` の範囲内の値にマッピングされます。

この範囲は、チェックする属性のソースを指定します。ページ、リクエスト (デフォルト)、セッション、またはアプリケーションです。

キャッシュされる JSP の例は次のとおりです。

```
<%@ taglib prefix="ias" uri="CacheLib.tld"%>
<ias:cache>
<ias:criteria timeout="30">
<ias:check class="com.iplanet.server.servlet.test.Checker"/>
<ias:param name="y" value="*" scope="request"/>
</ias:criteria>
</ias:cache>
<%! int i=0; %>
<html>
<body>
<h2>Hello there</h2>
I should be cached.
No? <b><%= i++ %></b>
</body>
</html>
```

Checker は次のように定義されます。

```
package com.iplanet.server.servlet.test;
import javax.servlet.*;
import javax.servlet.http.*;
public class Checker {
String chk = "42";
public Checker()
{
}
```

```
public Boolean check(ServletRequest _req)
HttpServletRequest req = (HttpServletRequest)_req;
String par = req.getParameter("x");
return new Boolean(par == null ? false :par.equals(chk));
}
}
```

この例でキャッシュされた要素は、リクエストのパラメータが x=42 で、y がその要素の保存に使われた値と等しい場合に有効です。<ias:criteria> ブロック内に <ias:param> および <ias:check> の複数の組み合わせを持たせることができます。また、JSP 内に複数の <ias:criteria> ブロックを持たせることもできます。

注

* cache-criteria = "*" は使用できません。

* キャッシュ基準が正しく確立されると (arg="*")、動作が遅くなります。つまり、キャッシュできなかったときの更新、キャッシュヒット、次に更新、次にキャッシュヒットという動作です。

第 5 章

Java 実行時システムのチューニング

アプリケーションのスレッドを Solaris® ユーザレベルのスレッドにバインドすると、Java 実行時システムをチューニングできます。Solaris のオペレーティング環境は、デフォルトでは、2 つのレベルのスレッドモデルをサポートします。アプリケーションレベルの Java スレッドはユーザレベルの 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、ユーザレベルスレッドが何百もある場合に役立ちます。

この章では、次のトピックについて説明します。

- バインドされたスレッドの使い方
- メモリと割り当ての管理

バインドされたスレッドの使い方

アプリケーションスレッドと Solaris `lwps` を 1 対 1 でバインドすることもできます。スレッドがわずかしかなく、作成される `lwps` も少ない場合、アプリケーションによっては、デフォルト以外のモデルを使うとパフォーマンスが向上することがあります。JVM の設定によって Java スレッドを Solaris スレッドにバインドした後に、KJS シェルスクリプト内で KJS 実行可能コマンドを呼び出すことができます。

```
_JVM_ARGS="bound_threads"
export _JVM_ARGS
```

メモリと割り当ての管理

ツールを効率的に実行するには、メモリとガベージコレクションの管理がきちんと行われている必要があります。この節で扱うトピックでは、メモリと割り当て機能の最適化に必要な情報について説明します。

この節では、次のトピック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設定値の指定
- ガベージコレクションの追跡
- Java ヒープのチューニング
- ダイナミックコンパイラのチューニング

ガベージコレクタのチューニング

新しい Java ランタイム環境 (JRE) には、世代別オブジェクトメモリシステムと高性能のガベージコレクションアルゴリズムが備わっています。

世代別メモリシステムでは、ヒープを適切なサイズのパーティション数個に分割します。これを世代と呼びます。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寿命が短いことに基づいています。新しく割り当てられたオブジェクトは、`Eden` とも呼ばれる若い世代に割り当てられます。新しく割り当てられたオブジェクトの死亡率は高いので、若い世代の掃除つまりガベージコレクションは生産的であることが多く、割り当てスペースの回転率が上がります。

ガベージコレクタをコンパクトにする場合は、Eden で 2 つのセミスペースを使い、存続するオブジェクトを一方の若いスペースから次のスペースにコピーします。複数の若いスペースコレクションで生き延びるオブジェクトは終身権を得ます。つまり、終身世代にコピーされます。終身世代は規模が大きく、すぐにいっぱいになることはありません。そのため、終身世代ではガベージコレクションがそれほど頻繁に行われず、各コレクションには若いスペースだけのコレクションよりも長い時間がかかります。終身スペースのコレクションはフル GC コレクションとも呼ばれます。

頻繁に行われる若いスペースのコレクションは短時間 (数ミリ秒) で終了し、たまに行われるフル GC は比較的時間がかかります (数十ミリ秒から数秒間、ヒープサイズによって異なる)。

Train アルゴリズムなど、ほかのガベージコレクションアルゴリズムは増分的で、フル GC がいくつかの増分区分に区切られます。このため、フル GC が始まると小さなガベージコレクションが停止する可能性が高くなります。オーバーヘッドを伴うので、通常、企業向けの Web アプリケーションには使われません。

通常、永続世代と呼ばれる第三世代も JVM により作成され、読み込まれた Java クラスなどの内部オブジェクトに格納されます。

HotSpot と Solaris JDK のどちらにも世代別ガベージコレクションシステムがあります。増分性のある Train ガベージコレクタは、HotSpot だけに同梱されています。HotSpot は、Solaris および NT の両方のプラットフォームのデフォルトです。将来、JDK 1.4 には新しい並列で同時的なコレクタが導入されます。

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、ロックしない、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。ユーザアプリケーションレベルのオブジェクトプールは、以前の世代の Java 仮想マシンで実行すると実際には利点があることもあります。しかし、JDK 1.2 以後で使用できる新しい世代の仮想マシンでは、実際はアプリケーションの速度が低下することもあります。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。

ガベージコレクタの設定値の指定

ガベージコレクタによってメモリ使用率を向上させるために、次の設定値を使用できます。

- 明示ガベージコレクタ
- 遅延ガベージコレクタ

明示ガベージコレクタ

明示ガベージコレクタの動作を変更することで、アプリケーションサーバによるメモリ使用率を制御できるようになります。Solaris の `iasenv.ksh` ファイル内にある `JAVA_GX_ARGS=-DGX.cleaner.enabled` キー、また Windows のレジストリの設定を変更することで、クリーナを有効にしたり無効にしたりできます。

Solaris では、リクエストの応答時間を速くするために、`JAVA_GX_ARGS=-DGX.cleaner.enabled` エントリがデフォルトで「no」に設定されています。iPlanet Application Server のメモリ使用率が異常に高いことが判明した場合は、この値を「yes」に変更してクリーナを有効にするか、または `JAVA_GX_ARGS` ラインをコメント出力します。

```
JAVA_GX_ARGS=-DGX.cleaner.enabled=yes
```

クリーナは、いったん有効にされると 10 秒ごとに呼び出されます。クリーナの呼び出し間隔は、次のラインを `iasenv.ksh` ファイルに追加して制御できます。

```
JAVA_GX_ARGS=-DGX.cleaner.interval=N
```

この `N` は、ミリ秒数です。

Windows では、クリーナはデフォルトで有効にされています。パフォーマンスを向上させるために、クリーナを無効にしたり、クリーナの呼び出し間隔を長く設定することができます。

クリーナのデフォルトの動作を変更する場合は、SOFTWARE¥iPlanet¥Application Server¥6.0¥Java にあるレジストリ内の JavaArgs に、次のフラグを追加します。

```
-DGX.cleaner.doGC=yes -DGX.cleaner.interval=N
```

上記の N は、ミリ秒数です。

遅延ガベージコレクタ

メモリ使用率の監視をするアプリケーションに基づいた AppLogic 用の遅延ガベージコレクタを有効にするために、新しいスイッチが導入されました。この導入によって、Java システムの useDeferredGC プロパティを設定してチューニングすることができます。

Solaris では、iasenv.ksh ファイル内にある次の JAVA_ARGS を追加することで、遅延ガベージコレクタを有効にできます。

```
-DuseDeferredGC=true
```

Windows では、SOFTWARE¥iPlanet¥Application Server¥6.0¥Java にあるレジストリ内の次の JavaArgs エントリを追加します。

```
-DuseDeferredGC=true
```

AppLogic ベースのアプリケーションではメモリ使用率が急に増加する場合が少ないことが報告されているため、このプロパティのデフォルト値は「false」に設定されています。遅延ガベージコレクションでは、ガベージがコレクションされないように、新しく作成されたオブジェクトの参照がリクエストの実行の最後まで一時的に格納されます。したがって、リクエストの最後ですべてのメモリが一度に解放されますが、過渡的にメモリ使用率が上がることがあります。

ガベージコレクションの追跡

JVM に -verbose:gc フラグ (-verbosegc の場合もある) を立てると、コレクションのたびに 1 行の通知メッセージが出力されます。いくつかの理由から、このフラグを立てると便利です。

- ガベージコレクションの停止と停止時間が必ずログに記録され、異常に長く停止しているかどうかが検出されます。
- また、JVM の生存を表す「心臓の鼓動」のような役目を果たします。

アプリケーションロジックがデッドロックされている場合でも、通常のガベージコレクションが行われる可能性があるので注意が必要です。

- アプリケーションが Java オブジェクトをリークしているかどうかが非常に簡単にわかります。多くのフルガベージコレクションが行われた後でも非ガベージオブジェクトの数が増える場合は、メモリリークの疑いがあります。

Java ヒープのチューニング

世代別ガベージコレクションについて理解できたので、ヒープ設定がパフォーマンスに影響を与える理由はすぐにわかります。

この節には次のトピックがあります。

- Java ヒープのサイジングのガイドライン
- HotSpot サーバ VM のチューニングオプション
- Solaris でのヒープ設定の例
- Windows でのヒープ設定の例

Java ヒープのサイジングのガイドライン

Java ヒープのサイジングにとって重要なガイドラインがあります。

- 各 JVM プロセスに付与する Java ヒープを決定します。

  まず、アプリケーションサーバノードで使用するシステムメモリの量を決定します。次に、KJS プロセス数を設定し、システムメモリのサイズをこの数で割ります。それぞれの KJS プロセスが JVM プロセスです。

  KJS プロセスと CPU 数との比率は、およそ 1 対 1 です。比率を多少変えて試してみることができます。

- Java ヒープの開始サイズと最大サイズを、上で決定したサイズに設定します。JVM フラグの `-Xms<size>` および `-Xmx<size>` は最小ヒープサイズと最大ヒープサイズです。詳細については、JVM のマニュアルを参照してください。

たとえば、-Xms64m -Xmx64m のヒープサイズは 64 MB です。開始ヒープサイズ (`-Xms`) と最大許容ヒープサイズ (`-Xmx`) を同じ値にすると利点があります。JVM がデフォルトの開始ヒープサイズで始動するように指定された場合は、ヒープサイズが自動的に拡張します。ただし、この拡張は低速のプロセスであり、このヒープ拡張段階の間に頻繁にガベージがコレクションされてパフォーマンスが低下します。

規模の大きい Eden や若い世代スペースであれば、フルガベージコレクション間にスペースが増加します。しかし、若いスペースのコレクションには比較的時間がかかります。一般に、Eden のサイズは最大ヒープサイズの 1/4 から 1/2 にしておくことができます。

HotSpot サーバ VM のチューニングオプション

特に指定しないかぎり、iPlanet Application Server 6.5 では 1.3 Hotspot Server VM が読み込まれます。サーバ側のアプリケーションにとって、サーバモードの VM がより適しています。iPlanet Application Server は、次の引数でチューニングされた VM で始動します。

```
-server -Xss512k -Xms128m -Xmx1024m -XX:NewSize=42m
-XX:MaxNewSize=342m

-Xconcurrentio -XX:+DisableExplicitGC
```

利用可能な物理的なメモリサイズに基づいて、ヒープサイズを増やしたり減らしたりすることができます。

これらのオプションはアプリケーションによってはよく機能しますが、あまりよく機能しないアプリケーションもあります。つまり、I/O バインド型、計算集約型、メモリ集約型などのアプリケーションのタイプに左右されます。最適なオプションを決定する前に、調整可能なパラメータを試してみる必要があります。

表 5-1 に、VM のオプションと説明を示します。

表 5-1 HotSpot サーバ JVM のチューニングオプション

VM オプション	説明
-XX:NewSize=<n>	新世代の最初のサイズ (バイト数)
-XX:MaxNewSize=<n>	新世代の最大サイズ (バイト数)
-XX:+DisableExplicitGC	GC の全体的な制御下で GC、VM の明示した呼び出しを無効にする
-Xconcurrentio	スレッドベースの同期化の代わりに LWP ベースの同期化を行う
-XX:CompileThreshold=<n>	「n」は Hotspot コンパイラによって将来、最適化が実行された後のメソッドの呼び出し数。サーバモードの場合のデフォルト値は 10,000、クライアントモードの場合は 1,500
-Xbatch	バックグランドのコンパイルを無効にする
-Xincgc	増分性のあるガベージコレクション

若い世代のサイズを定数に設定すると、サイズの変更ができません。当社のテストでは、増分性のある GC を使用するとスループットが低下することが判明しました。

JVM のチューニングについては、http://java.sun.com/docs/hotspot/index.htm を参照してください。

Solaris でのヒープ設定の例

KJS シェルスクリプトで次の引数を JAVA_ARGS 環境変数に追加します。

```
-Xgenconfig:64m,64m,semispaces:64m,512m,markcompact
```

これにより、若い世代用の 64 MB セミスペースが 2 つある 512 MB の Java ヒープが作成されます。記号と簡易アルゴリズムを使うよう指定します。若い世代のサイズもヒープ全体のサイズも大きくすることができます。Solaris JDK では、オブジェクトヘッダやほかのスペースオーバーヘッドを占めることを正当化して、各セミスペースに指定したサイズの 2 倍のサイズを割り当てています。

`-Xgenconfig` は正しく理解しにくい複雑なフラグです。JVM の内部的な記事以外はあまり公表されず、文書化されていません。

`-Xgenconfig, -Xms, -Xmx` フラグは明らかに相互的であり、`genconfig` とともに指定されると、ほかの設定値に優先します。ただし、ある種の範囲のチェックでは、`genconfig` に指定された値と最小値および最大値が確実に一貫するように、JVM によって実行されるように見えます。

Windows でのヒープ設定の例

次のプロパティを iPlanet のレジストリに設定して、引数を JVM に渡すことができます。

`HKEY_LOCAL_MACHINE¥SOFTWARE¥iPlanet¥Application Server¥6.0¥Java¥JavaArgs` を希望する文字列に設定します。これがヒープサイジングパラメータを設定する場所です。

`-Xms` フラグと `-Xmx` フラグは、Solaris JDK の場合と同様に設定する必要があります。

`-XX:NewSize=<size>` で、新しいオブジェクトを割り当てる若いオブジェクトスペースの初期サイズをバイト数で指定します。若いスペースの初期サイズはデフォルトで 2 MB です。初期サイズは、`1024` の倍数にする必要があります。キロバイトの場合は k または K を、メガバイトの場合は m または M を付けます。`-XX:NewSize=64m` の場合、若いスペースの初期サイズが 64 MB に設定されます。若いスペースのサイズが大きいと、ガベージコレクションで停止する時間が長くなります。

`-XX:MaxNewSize=<size>` は、新しいオブジェクトを割り当てる若いオブジェクトスペースの最大サイズをバイト数で指定します。若いスペースの初期サイズは 2 MB です。`MaxNewSize` は `1024` の倍数で、`2MB` 以上であることが必要です。キロバイトの場合は k または K を、メガバイトの場合は m または M を付けます。`MaxNewSize` のデフォルト値は 64 MB です。`-XX:MaxNewSize=128m` を使って、若いスペースを必要に応じて 128 MB まで拡張することができます。

-XX:SurvivorRatio=k では、Eden のサイズ比率を生存スペースのサイズに設定します。たとえば、Windows でのデフォルト比率が 8 であれば、NewSize=64m の場合は 4 MB のセミスペースが 2 つになります。NewSize と SurvivorRatio を使うと、セミスペースを必要なサイズにすることができます。

このフラグは、内部的なサイジングの計算をかなり混乱させるので、最後の手段として使用することをお勧めします。

注 デフォルトの設定に比べて、-Xms256m -Xmx256m -XX:NewSize=128m -XX:MaxNewSize=128m などの設定ではパフォーマンスが 30% 向上したという報告があります。NewSize と MaxNewSize を正しく設定するだけでも、パフォーマンスに大きな影響を与えることができます。

このような HotSpot フラグは、HotSpot ベースのすべての Java ランタイム環境で使用することができます。HotSpot ベースの JDK 1.3 は、Solaris、Windows、および Linux でデフォルトとして提供されています。HotSpot のすべてのパフォーマンスフラグが、http://java.sun.com/docs/hotspot/VMOptions.html に一覧表示されています。

ダイナミックコンパイラのチューニング

Windows と Solaris JDK 1.3.1 の Java HotSpot 1.3 はどちらも適応型ダイナミックコンパイラを実装しており、プログラムのホットスポットを検出してピークパフォーマンス時のホットなプログラムセグメントだけをコンパイルします。このプロファイルによるコンパイルを行う場合、アプリケーション起動時に短時間のパフォーマンスの増加が見られることがあります。このため、ウォームアップの済んだ iPlanet Application Server でベンチマーク測定を行う場合は注意してください。

JIT をオフにすると時間がかかります。JIT をオフにして、よくチューニングされたシステム設定でアプリケーションを使った場合、3 倍のパフォーマンスが測定されたことがあります。これは、アプリケーションの性質やハードウェアとデータベース設定のシステムボトルネックによって異なります。

次の場合は、ダイナミックコンパイラをオフにすることができます。

- デバッグ中に、ソース行番号がある Java 例外スタック追跡を出力して確認する場合
- ダイナミックコンパイラでまれに発生するバグに対処する場合

  この場合は、JVM の隠し引数があり、クラスでコンパイルするメソッドの一部または全部を選択的に無効にすることができます。詳細は、Sun Java サポートにお問い合わせください。

Windows でコンパイラをオフにする必要がある場合は、レジストリの JavaArgs プロパティに -Xint を追加します。Solaris では、KJS シェルスクリプトの JAVA_ARGS に -Djava.compiler=none を追加します。

第 6 章

オペレーティングシステムのチューニング

Solaris® ネットワークには、iPlanet Application Server に直接役立つわけではありませんが、データセンタのアプリケーション群で実行するほかのソケット集約型カスタマアプリケーションに役立つパフォーマンスチューニングのヒントがいくつかあります。

Solaris の TCP/IP 設定をチューニングすると、多数のソケットを開閉するプログラムにとっての利点があります。iPlanet Application Server は小規模の固定されたコネクションセットで動作し、パフォーマンスの向上は Application Server ノード上でのように顕著ではありません。iPlanet Web Server と、iPlanet Application Server の Web フロントエンドとして設定されている iPlanet Web Servers には、多くの有利な点があります。

再起動後にこのような変化を維持するために、次のファイルに ndd 変数を設定します。
`/etc/rc2.d/S69inet`

この章には次のトピックがあります。

- 待機間隔時間の設定
- TCP コネクションハッシュテーブルサイズの設定
- プロセスのバインド

待機間隔時間の設定

クライアントとサーバの両方で接続が閉じられると、しばらくポートが利用できなくなるので、新しいプログラムは以前のプログラム用のパケットを受け取ることができません。Solaris をインストールしたマシンでは、`tcp_time_wait_interval` のデフォルト値は `240,000` ミリ秒 (4 分) です。この値を `60000` ミリ秒 (1 分) か `30000` ミリ秒 (30 秒) に設定し、ソケット通信集約型プログラムのパフォーマンスを上げることをお勧めします。値は、動作中のシステムで変更し、確認できます。

```
/usr/sbin/ndd -set /dev/tcp tcp_time_wait_interval 60000
/usr/sbin/ndd -get /dev/tcp tcp_time_wait_interval
```

TCP コネクションハッシュテーブルサイズの設定

コネクションハッシュテーブルはアクティブな TCP コネクション (`ndd -get /dev/tcp tcp_conn_hash`) のすべての情報を記録します。この値によってコネクションの数が制限されることはありませんが、コネクションハッシュに時間がかかることがあります。検索効率をよくするために、サーバで予想される同時 TCP コネクション数の半分に設定します （`netstat -nP tcp|wc -l` でコネクションの数がわかります)。デフォルトは `512` です。設定は `/etc/system` だけで行うことができ、起動時に有効になります。

```
set tcp:tcp_conn_hash_size=8192
```

プロセスのバインド

Solaris で動作する iPlanet Application Server の以前のバージョンでは、アプリケーションサーバのプロセスを 1 つまたは複数のプロセッサにバインドすると、パフォーマンスが大幅に向上しました (`20 ～ 30%`)。ただし、Solaris 8 の SP2 以後のリリースではパフォーマンスの大きな向上は見られません。そのため、この設定はお勧めしません。

KJS ごとに少なくとも 1 つの CPU をバインドすることをお勧めします。最近の JVM は並行アプリケーション上で複数のプロセッサ (最大 32 個の CPU) に拡張するので、使用可能なプロセッサを小さなセットに分けて KJS プロセスにそれぞれバインドすることもできます。

`psrinfo` コマンドは、オンライン状態のプロセッサの一覧を出力します。

`pbind -b <processor_id> <process_id>` を使って、`<process_id>` で識別されるプロセスを `<processor_id>` で識別されるプロセッサにバインドすることができます。たとえば、KJS プロセス ID が `6543` の場合、`$ pbind -b 0 6543` はプロセス `(id=6543)` を CPU `0` にバインドします。

`$ psrset -c 1 9` は、CPU `1` と `9` から成る新しいプロセッサを作成します。新しいセットの ID が出力されます。

この例のプロセス 6543 をここで新しく作成したプロセッサのセットに `$ psrset -b <processor_set_id> 6543` としてバインドできます。

詳細については、Man ページを参照してください。参照するのは KJS プロセスの `pid` であり、KJS プロセスを生成するシェルスクリプトの `pid` ではないことに注意してください。

KXS でも同じような方法が使えます。KXS は CPU 2 個まで適切に拡張し、KJS プロセス当たり最大 8 つの JVM をビジー状態に保持することがわかっています。

`pbind -q` は、現在のプロセッサとプロセスバインディングを表示します。

第 7 章

データベースサーバのチューニング

この章では、Oracle サーバをチューニングしてパフォーマンスを最大にする方法について説明します。Solaris をチューニングして Oracle とともに動作させる方法についても説明します。

Oracle サーバのチューニング

Oracle のチューニング自体が広範なトピックですが、後に説明するパラメータを正しく設定すれば十分です。チューニング可能な各設定についての詳細は、Oracle のマニュアルを参照してください。このマニュアルは特に Solaris プラットフォームで Oracle の初期パラメータを設定するためのものです。すべて Oracle 8.1.6 以降でテスト済みです。

はじめに、`/etc/system` ファイルで特定のシステム共有のメモリプールパラメータを設定します。Oracle アーキテクチャは、複数プロセスのデータを共有する共有メモリセグメントと、ロックを扱うセマフォを幅広く活用します。多くの場合、デフォルトのカーネル値では足りません。`/etc/system` ファイルを変更して、マシンを再起動する必要があります。通常はこれで十分ですが、システムやリソースによっては調整が必要になることもあります。

`dbassist` コマンドを使って、データベースを作成しチューニングすることができます。ただし、データベースインスタンスがすでに作成されている場合は、パラメータを手動でチューニングする必要があります。

Oracle サーバは、専用モードと共有サーバモードの 2 つのモードで実行することができます。共有サーバつまりマルチスレッドモードでは、多数のクライアントユーザプロセスでわずかなサーバプロセスを共有できるようになります。

Oracle の初期パラメータやチューニング可能パラメータはすべて次のファイルに格納されています。

```
$ORACLE_HOME/dbs/init<SID>.ora
```

このファイルは実際には $ORACLE_HOME/admin/<SID>/pfile/init<SID>.ora にリンクされています。

多くの場合、インストーラによってデータベースインスタンスのために作成された値は十分ではありません。

警告 必ずこのファイルのバックアップを取ってから、エントリを編集てください。

Solaris カーネルパラメータのチューニング

次に、Solaris カーネルパラメータを Oracle 用にチューニングする方法について説明します。

- Solaris カーネルには Oracle に合うように設定したパラメータがあります。Oracle アーキテクチャは、複数プロセスのデータを共有する共有メモリセグメントと、ロックを扱うセマフォを幅広く活用します。Solaris を始めとする多くのオペレーティングシステムには、Oracle データベースを維持できるだけの共有メモリやセマフォがデフォルトで備わっているわけではありません。ただし、/etc/system ファイルを編集してサーバを再起動させるだけで Solaris のカーネルパラメータを変更することができます。

表 7-1 Oracle 用の Solaris カーネルパラメータ

カーネルパラメータ	初期設定	目的
SHMMAX	4294967295	共有メモリ 1 セグメントの最大サイズ
SHMMIN	1	共有メモリ 1 セグメントの最小サイズ
SHMMNI	100	システム全体の共有メモリセグメントの最大数
SHMSEG	10	1 つのプロセスが扱える共有メモリセグメントの最大数
SEMMNS	2000	システム全体のセマフォの最大数
SEMMSL	1000	セット当たりのセマフォの最大数
SEMMNI	100	システム全体のセマフォの最大セット数
SEMOPM	100	semop 呼び出し当たりの最大オペレーション数
SEMVMX	32767	セマフォの最大値

最初の 4 つのカーネルパラメータで共有メモリセグメントを設定します。ここで推奨されている設定は、ほとんどの Oracle データベース実装に適しています。SHMMAX 設定は大きすぎるように見えますが、SHMMAX 設定を実際に必要とする値より大きくしてもペナルティはありません。

最後の 5 つのカーネルパラメータでセマフォを設定します。各 Oracle インスタンスには、プロセスごとに 1 つのセマフォとそれ以外に 10 個のセマフォが必要です。さらに、最大のインスタンスにはプロセスごとに 2 番目のセマフォが必要です。サーバにデータベースを 1 つだけ設定する場合は、最終的には各プロセスに 2 個のセマフォとそれ以外に 10 個のセマフォが必要になります。

最初の 2 つのセマフォカーネルパラメータ SEMMNS と SEMMSL に推奨する設定は、ほとんどの Oracle 実装に適しています。多数の同時データベースコネクションを持つシステムでは、この値を大きくする必要があります。最後の 3 つのセマファカーネルパラメータに関してここで推奨されている設定は、ほとんどの Oracle データベース実装に適しています。

一般に、使用する Solaris カーネルにここで推奨するよりも大きいパラメータが設定されている場合は、その設定を変更しないようにしてください。/etc/system のカーネルパラメータ設定を変更した場合は、サーバを再起動して新しい設定を有効にします。

/etc/system ファイルの最後に次の行を追加します。

```
set shmsys:shminfo_shmmax=4294967295
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmseg=10
set semsys:seminfo_semmns=2000
set semsys:seminfo_semmsl=1000
set semsys:seminfo_semmni=100
set semsys:seminfo_semopm=100
set semsys:seminfo_semvmx=32767
```

第 8 章

パフォーマンス向上のための一般的なガイドライン

パフォーマンスを向上させるために、次の一般的なガイドラインがあります。

- クラスタが 1 つではないネットワークで複数の iPlanet™ Application Server がインストールされている場合、ネットワークのマルチキャストサーバホストアドレスとポート番号を変更すると、更新の受信を減らし、レジストリへの書き込みを避けることができます。
- Oracle 固有の JDBC ドライバを使用している場合は、`OPARSE` の呼び出し中に SQL ステートメントの解析ができるように、`iasenv.ksh` ファイル内に環境変数の `IAS_OPARSE_NODEFER` を設定します。この変数が設定されていないと、ステートメントの解析が `oexec` の実行まで行われません。

  違法な SQL ステートメントが渡されると、Oravle 8.1.6 とともに使用されている iPlanet Application Server のコアダンプが発生します。この変数を設定すると、コアダンプの発生を防ぐことができます。この設定は開発環境に対してだけ奨励するもので、運用環境に対しては奨励しません。この機能は、iPlanet Application Server, Enterprise Edition 6.0, SP3 から利用できます。

EJB のパフォーマンス向上のためのガイドライン

EJB のパフォーマンスの向上を促進するため、次のガイドラインがあります。

- セッションフェールオーバーが必要なければ、アプリケーションの DSync をオフにします。つまり、可能であれば分散セッションではなくライトセッションを使い、DSync を使ってほかのマシンにセッションの変更を伝えなくても済むようにします。
- セッションフェールオーバーは必要でも、マシン全体の障害に対する保護は必要ない場合は、複数の KJS プロセスがあることを確認して `Dsync-Local` セッションを使います。その後、セッションフェールオーバーは、iPlanet Application Server インスタンス内部のすべての KJS プロセスで利用できるようになります。ただし、マシンの境界を越えるセッションフェールオーバーは提供されません。KJS がクラッシュしても、少なくともセッション情報は残ります。
- セッションを小さくします。さまざまな検証の結果、ユーザセッションあたり 4 KB になりました。ただし、iPlanet Application Server では、パフォーマンスに影響を与えるのは DSynch プロセスであり、同期チャンクのサイズではないようです。このため、ユーザセッションサイズを小さくするよりも、クラスタを小さくして 4 インスタンス以下にする方が重要である可能性があります。
- 分散セッションを使う場合は、セッションに格納するものを単純化します。iPlanet Application Server セッションサービスは、簡単な名前と値の組み合わせを使うように設計されています。
- 分散セッション環境では、セッション情報を移動するために直列化や直列化解除を行う必要があります。このため、大きくて複雑なオブジェクトを作ると移動に時間がかかります。
- セッションを削除するために `HttpSession` タイムアウトに頼らないでください。できるだけ `HttpSession.invalidate()` メソッドを使い、手動で削除します。
- `HttpSession` タイムアウトをできるだけ小さく設定し、必要のないコンテナのセッションを早く削除できるようにします。
- 状態のある セッション EJB へのハンドルが Servelet によって `HttpSessions` に格納されていたり、JSP がハンドルにアクセスしている場合は、EJB のセッションタイムアウトを `HtttpSession` タイムアウトに近い値に設定する必要があります。

  状態のあるセッション EJB に対するデフォルトのセッションタイムアウトは、`14400` 秒 (4 時間) です。これを適切な値に変更します。これにより、KJS メモリが増えるのを抑え、不要になった EJB インスタンスが不必要に不活性化されるのを避けることができます。
- JSP ではセッションオブジェクトが自動的に作成されるので、HttpSession を明示的に使っていない場合は不要な `HttpSession` オブジェクトを作成していることになります。これを防ぐには、JSP ページディレクティブ `<%@session=false %>` を JSP に追加します。

第 9 章

サーバパフォーマンスの確認

サーバのパフォーマンスを定期的に監視および確認して、高パフォーマンス用に設定したメソッドが実行されるようにします。次のトピックは、iPlanet™ Application Server のパフォーマンスを確認するのに役立ちます。

この章では、次のトピックについて説明します。

- iPlanet Application Server の監視
- パフォーマンスチューニングツールの使用
- SNMP 監視の設定
- パフォーマンスデータの取得

iPlanet Application Server の監視

監視は RAS (信頼性、可用性、サービス性) のうち、サービス性に当てはまる部分です。動的なサービスの動作がわからなければ、サービスを規定することはできません。

サービスプロバイダは、外部に公開されている Web サービスにアクセスするユーザが経験するアプリケーションパフォーマンスをリアルタイムで監視しようとします。データセンタには通常、サービスレベル契約を維持するためのグラフィカルなディスプレイと警報システムを持つ監視コンソールがあります。

そのため、アプリケーションと Web サービスコンテナは、それらのツールに正確で簡単にアクセスできるパフォーマンス情報を提供する必要があります。このような監視にはさまざまなレベルがありますが、少なくともサーバで処理されているリクエストの割合は示すことができます。

監視には柔軟性が必要です。つまり、再起動せずに、実行中のサーバで監視のオンとオフを切り換えることができる必要があります。その他の高レベルの監視ツールに対応できるよう、データは正しくフォーマットされている必要があります。

うまくチューニングされたシステムのパフォーマンスには、基本的に次のような特徴があります。

- 全サーバの CPU 時間がすべて平均的に使用されている
- 各サーバの全プロセッサの CPU 時間がすべて平均的に使用されている
- 全 KJS プロセスおよび KCS プロセスの CPU 時間がすべて平均的に使用されている
- システムタイムがかなり低く抑えられている (0 - 25%)。ワークフローが計算集約型である場合は特に低く抑えられている
- KXS プロセスに割り当てられたプロセッサがすべてフルに活用されている

iPlanet Application Server Administration Tool (iASAT) を使用して、KXS と KJS/KCS の統計を監視します。プロセッサが完全にはビジーでないときでも待機しているリクエストを探します。これは、サーバが最適にチューニングされていないことを示します。DB コネクションプールに適切なコネクション数を設定できるように、KJS/KCS エンジン内のアクティブなデータベースコネクションをチェックします。

Solaris の場合

標準的なインストールの場合、各プロセスに使用される処理時間を調べるのは困難です。ただし、各 iPlanet Application Server プロセスに別の CPU に対する `pbind` を実行した場合は、`mpstat` によって各 CPU が消費した時間の割合が表示されます。さらに、プロセッサごとにシステム時間に対するユーザ時間の比率が表示されます。`http://www.sunfreeware.com` から入手できる `proctool` は、プロセスごとのリソース使用率を示す非常に使いやすいツールです。さらにこのツールには、プロセスのプロセッサへのバインドとプロセス優先順位の変更のための GUI が用意されています。

この節には次のトピックがあります。

- iASAT によるプロットの追加

iASAT によるプロットの追加

プロセスの属性を監視し、プロセスのパフォーマンスを確認するために、iASAT を使ってプロットを追加できます。これらのプロットは、KJS、KCS、および KXS プロセスの属性を一覧にするのに役立ちます。プロットやプロセス属性データの設定方法については、『iPlanet Application Server 管理者ガイド』の第 2 章「サーバの稼動状況の監視」を参照してください。

パフォーマンスチューニングツールの使用

市販されている多くのツールを使って、iPlanet Application Server で稼働する J2EE アプリケーションの動作のプロファイルを収集できます。これらのツールのほとんどは JVM プロファイルインタフェース (JVMPI) に依存しており、稼働中の Java 仮想マシンプロセスから動的に情報を取得します。

このようなツールの例としては、Machine Process、Jprobe、および OptimizeIt があります。これらのツールは開発環境で使用して、CPU とメモリの利用状況のプロファイルの収集、オブジェクトの調査、アプリケーションのメモリリークとデッドロックの検出、コードの補償、およびその他のトラブルシューティングを行います。JVMPI を使うとパフォーマンスオーバーヘッドが大幅に増加するため、配置されたアプリケーションの監視やプロファイル収集には適していません。

選択的バイトコード計測を使って、対象を絞った効率的なパフォーマンス監視を行うことができます。情報量や性能では劣りますが、より現実的なデータが得られます。Wily Solution 社の Introscope はこのようなテクノロジの一例であり、iPlanet Application Server の Version 6.0, SP3 以後に統合されています。

この節では、iPlanet Application Server での Jprobe、OptimizeIt、および Introscope の使い方について説明します。ここでは Windows 版のアプリケーションサーバだけを扱いますが、Solaris 版でも同様に動作します。

この節には次のトピックがあります。

- OptimizeIt を使ったパフォーマンスのチューニング
- Jprobe を使ったパフォーマンスのチューニング
- IntroScope を使ったパフォーマンスのチューニング

OptimizeIt を使ったパフォーマンスのチューニング

OptimizeIt は iPlanet Application Server のパフォーマンスのチューニングに使える製品です。この製品は *http://www.optimizeit.com* からダウンロードできます。このツールを使うために、次の作業を実行してください。

- iPlanet Application Server に同梱されている JDK (HotSpot を含む JDK 1.3.1_02) が使えるように JVM を設定します。
- 呼び出されるクラスファイル名として、`com.kivasoft.engine.Engine` を追加します。

- iPlanet レジストリのクラスパスエントリにあるセミコロン (;) で区切られたエントリをすべて OptimizeIt のクラスパスに追加します。これで、プロファイル収集を開始する準備ができました。

Solaris での手順も Windows NT の場合とほぼ同じです。ただし、Solaris の場合、OptimizeIt は Reference JDK 1.2 とだけ機能することが保証されており、iPlanet Application Server 6.0 には含まれていません。

Jprobe を使ったパフォーマンスのチューニング

JProbe は iPlanet Application Server のパフォーマンスのチューニングに使えるサードパーティツールです。iPlanet Application Server に Jprobe をインストールして使う方法については、
`http://www.jprobe.com/software/support/jprobe/j2ee/iplanet.html` を参照してください。

IntroScope を使ったパフォーマンスのチューニング

Introscope では、指定したアプリケーションクラスファイルをすべて計測してプロファイル情報を収集する必要があります。計測は手動でスタティックに行うことも、iPlanet Application Server の内部クラスローダと対話して動的に行うこともできます。

シームレスであり、ディスクに配置したバイトコードを変更せずに動的に制御できるので、後者の方法をお勧めします。Wily Tech 社の Introscope については、
`http://www.wilytech.com` を参照してください。

Introscope の統合サポートと動的なバイトコード計測は、iPlanet Application Server, Enterprise Edition 6.0, SP3 以後で利用できます。

SNMP 監視の設定

SNMP は、ネットワークの稼動状況に関するデータを交換するために使われるプロトコルです。SNMP によって、データはアプリケーションサーバと、ネットワーク管理ソフトウェアがインストールされているワークステーションの間で移動します。ワークステーションからリモートでネットワークを監視して、サーバ間のネットワークの稼動状況に関する情報を交換できます。たとえば、HP OpenView などのアプリケーションを使って、アプリケーションサーバが受信するエラーメッセージの数やタイプのほかに、どの iPlanet Application Server マシンが実行中かを監視できます。

ネットワーク管理ワークステーションは、サブエージェントおよびマスターエージェントの 2 つのタイプのエージェントを介して、企業のアプリケーションサーバを使って情報を交換します。サブエージェントは、アプリケーションサーバに関する情報を収集してマスターエージェントにその情報を渡します。マスターエージェントは、さまざまなサブエージェントとネットワーク管理ワークステーションとの間で情報を交換します。マスターエージェントは、通信するサブエージェントと同じホストマシンで動作します。

SNMP 監視の設定方法については、『iPlanet Application Server 管理者ガイド』の第 3 章「iPlanet Application Server をサードパーティツールを使って 監視するよう SNMP を設定する」を参照してください。

パフォーマンスデータの取得

Web リクエストは、ロードジェネレータから Web サーバのフロントエンドに渡され、iPlanet Application Server プラグインを経由して KXS プロセスに渡り、最終的に KJS プロセスに渡されて処理されます。応答は同じ流れを逆にたどります。4 か所で時間を測定することができます。

ロードジェネレータで : SilkRunner などのカスタムロード生成ツールを使用している場合は、ツールに測定メソッドとグラフ作成メソッドが含まれています。このようなツールでは、ユーザが実際に経験する応答時間を測定します。

Web サーバで : 応答時間のプロファイルを収集できるように iPlanet Web Server を設定することができます。『Web サーバのパフォーマンスチューニングと分析ガイド』を参照してください。

iPlanet Application Server Web プラグインで : iPlanet Application Server がリクエストに応答する時間を測定します。これは Web コネクタの観点から測定されます。つまり、リクエストが iPlanet Application Server に送信されてから、Web サーバで実行する Web コネクタプラグインに応答が戻ってくるまでの時間です。

ログを記録するには、iPlanet レジストリの iPlanet Web Server サーバで次のキーを有効にします。

```
SOFTWARE¥iPlanet¥Application Server¥6.0¥CCS0¥HTTPAPI¥iASRespTime=1
```

これによって、Web サーバのログファイルにタイミング統計情報がダンプされます。タイミング情報を抽出するには、次のような合成シェルコマンドを実行します。

```
grep "plugin reports" errors| grep -v Registry| cut -c2-21,64- | cut
-d " " -f1,2,4
```

次に出力例を示します。時間の単位はミリ秒です。

```
    22/Sep/2000:19:36:09 </iASApp/tmf/TMFServlet> 420
```

```
22/Sep/2000:19:36:10 </iASApp/tmf/TMFServlet> 600
22/Sep/2000:19:36:16 </iASApp/tmf/TMFServlet> 392
22/Sep/2000:19:36:16 </iASApp/tmf/TMFServlet> 220
22/Sep/2000:19:36:16 </iASApp/tmf/TMFServlet> 428
```

iPlanet Application Server サーバの KXS ログは <IAS_HOME>/ias/logs/KXS にあり、reqstart 時刻と reqexit 時刻がリクエストごとに記録されています。reqexit 値は各リクエストの処理時間を提供します。この時間から、iPlanet Application Server で Servlet や JSP リクエストを実行するのに必要な時間がわかります。

iPlanet レジストリで次のキーを 1 に設定すると、監視に便利です。

```
Software¥iPlanet¥Application Server¥6.0¥CCS0¥REQ¥debug=1
```

次の例のように、KXS ログのエントリの確認を開始します。

```
[26/Apr/2001 11:48:05:7] info:NSAPICLI-012:plugin reqstart, tickct:
988310885s 763786us
[04/26/01 11:48:05:768] Request 00 Starting AppLogic
{1A488137-7510-1941-BAE5-080020B90F48} on Engine 0
[04/26/01 11:51:07:504] Request 00 Completing AppLogic
{1A488137-7510-1941-BAE5-080020B90F48} Execution
[26/Apr/2001 11:51:07:5] info:NSAPICLI-009:plugin reqexit:181s
741781us
Request # starts at 00 and increments
```

これらのプローブポイントで収集したデータを使って、各リクエストの処理にかかった時間を正確に判断することができます。パスのキューにあるスレッド処理リクエストの数が正しく設定されていない場合は、キューによる遅延が応答時間を左右します。iPlanet Application Server と iPlanet Web Server をこのマニュアルのとおりに設定し、アプリケーションのパフォーマンスチューニングを行ってください。

第 10 章

FAQ/ よくある質問

iPlanet™ のシステムエンジニア、プロフェッショナルサービスコンサルタント、およびお客様は、長年にわたってさまざまな方法で iPlanet Application Server の運用環境を最適化してきました。この章はその結果をまとめて、複製したものです。

次の節では、パフォーマンス関連してよくある質問 (FAQ) を次のように分類しています。

- 環境設定
- システムのチューニング
- アプリケーションのチューニング

今後もこの FAQ を充実させていくために、利用者からのフィードバックをお待ちしています。

環境設定

この節では、iPlanet Application Server 環境の設定に関する事項を扱います。

1. 通常の運用環境では、iPlanet Application Server の CPU 1 個につき必要な RAM サイズはどのくらいですか。

   iPlanet Application Server の CPU 1 個につき 1 GB の RAM が必要です。

2. Solaris での一般的な開発者サンドボックスインストールの場合は、1 つの開発チームに CPU は何個必要ですか。

   iPlanet Application Server では、CPU 1 個につき 3 ～ 5 名の開発者をサポートできます。条件の厳しい使用法の場合は、CPU 1 個につき 2 ～ 3 名になります。

3. iPlanet Application Server をインストールするには、どのくらいのディスク容量が必要ですか。

   配布プログラムは約 150 MB ですが、インストールして操作するにはこの 3 倍の容量が必要です。最初は約 450 MB の未使用ディスク領域が必要です。インストール後の iPlanet Application Server の操作には約 256 MB のディスク容量が必要です。

4. 1 つの iPlanet Application Server インスタンスを有効に活用するには何個のプロセッサが必要ですか。

   一般的に、iAS は 8 〜 12 個以内のプロセッサであれば実行することができます。1 つの iPlanet Application Server インスタンスには KXS が 1 つしかありませんが、KJS プロセスは多数存在させることができます。アプリケーションが KXS ではなく KJS で制約を受けているような場合は、12 プロセッサの範囲内で拡張することができます。ただし、分散セッション管理 (DSync) や Servlet の結果セットキャッシングのように、KXS ベースのサービスを頻繁に使う場合は、KXS のスケーラビリティはプロセッサ 8 個の範囲に制限されます。8 個から 12 個のプロセッサ範囲を超えて拡張するには、該当のマシンに複数の iPlanet Application Server インスタンスをインストールすることを検討してください。

5. iPlanet Application Server の分散セッション管理 (DSync) 機能を使うと、オーバーヘッドが高くなる可能性があります。KXS プロセスで分散セッション管理 (DSync) の負荷を減らすには、インストール時にどのような点に注意すればよいでしょうか。

   DSync のバックアップを 1 つだけ行うようにします。こうするとマスタと同期されたバックアップの内部メモリセッションストアを保持するのに必要な DSync の作業量が減り、プライマリに障害が発生した場合に交代することができます。進行中のセッションの可用性を保証するには、DSync バックアップは 1 つで十分です。

6. Java Servlet/JSP のプログラマですが、iPlanet Application Server のセッション管理機能をできるだけ効率的にするにはどうすればよいでしょうか。

   セッションの使い方を検討してください。KXS プロセスにある DSync 機能を使用すると、お使いの環境で負荷をかなり高めることになります。次の点を検討してみてください。

   - クラスタサイズを 4 インスタンスまでに制限します。このサイズを超えると、分散セッションストアを同期させるためのオーバーヘッドによってパフォーマンスが制限されることが多くなります。
   - スティッキーロードバランスを使います。セッション中、ユーザの後続の処理は必ず同じ KJS に戻り、セッション情報がローカルに利用できます。これは状態のあるセッション EJB では特に重要です。

7. 管理者ですが、KXS ハンドルセッション管理の維持管理作業を効率的にするにはどうしたらよいでしょうか。

   セッションがタイムアウトや無効になった場合、メモリ内のセッションストアからすぐに削除する必要があります。セッション管理専用のスレッド数を増やすと、ほぼ同時に削除することができます。iPlanet Registry 内の次のプロパティを調整して、セッションノードの管理をより効率的に行います。

```
SOFTWARE/iPlanet/Application
Server/6.0/Clusters/<machine-name>-NoDsync/SyncTimeoutThreadCount
```

システムのチューニング

1. 上手にチューニングされた iPlanet Application Server システムの構成について教えてください。

   上手にチューニングされたシステムには次のような特徴があります。

   - すべてのサーバの CPU 時間が均一である
   - 各サーバのプロセッサの CPU 時間はすべて平均的に使用されている
   - すべての KJS プロセスで CPU 時間が同程度である
   - ワークフローが計算に集中するのはシステム時間の 0 ～ 25% である
   - KXS プロセスに割り当てられたプロセッサがすべて活用されている

2. マルチ CPU マシンに iPlanet Application Server を複数インストールしようと思います。これは最善の拡張方法でしょうか。

   1 台のマシンに複数の iPlanet Application Server をインストールするのは、ほかに方法がない場合だけにしてください。このようにすると、通常は運用管理が混乱します。代わりに、KJS (Java VM) プロセスを追加したり、iPlanet Application Server プロセスのスレッドプールで使用するスレッド数を変更したりして、まず iPlanet Application Server インスタンスを 1 つチューニングしてみてください。

3. KXS プロセスに必要なスレッドはいくつですか。

   KXS ごとに 32 個のスレッドがデフォルトの設定です。ただし、このスレッド数を増やすことができます。しかし KXS のスレッド数を増やしても、パフォーマンスが著しく向上するとは限りません。

   上手にチューニングされたシステムには次のような特徴があります。

   - すべてのサーバの CPU 時間が均一である
   - 各サーバのプロセッサの CPU 時間はすべて平均的に使用されている
   - すべての KJS プロセスで CPU 時間が同程度である

- ワークフローが計算に集中するのはシステム時間の 0 ～ 25% である

KXS プロセスプールに割り当てられたすべてのプロセッサが利用されることが重要です。KJS スレッドと比べ、KXS スレッドではあまり作業が行われません。このため、KXS スレッドは KJS ほど多く必要ありません。KXS のパフォーマンスを向上させたい場合は、プロセスまたはプロセッサのセットにバインド (pbind) してみてください。

4. KJS とプロセッサのバインドは、いつ検討すればよいでしょうか。

   iPlanet Application Server のプロセスを特定のプロセッサやプロセッサセットとバインドすることは、特に Solaris 環境では大きなメリットがあります。マルチプロセッサマシンでは、常に KXS をプロセッサにバインドしてください。こうすると、マルチプロセッサの mutex ロックを使って関連付けたオーバーヘッドにより、スループットが大幅に向上します。

5. KJS とプロセッサセット （複数プロセス）とのバインドは、いつ検討すればよいでしょうか。

   KXS がリクエストをキューに入れている場合は、プロセッサが 2 つあるプロセッサセットを作成します。KXS をこのプロセッサセットにバインドしてください。プロセッサが 3 つ以上あるプロセッサセットを KXS にバインドしても、通常、スループットは実質的に向上しません。

6. KXS にバインドされているプロセスセットからプロセッサを外すのはどのような場合ですか。

   KXS がプロセッサを十分に活用しておらず、パワーが足りない KJS プロセスがある場合 (スレッドがキューに入っている場合)、KXS からプロセッサを外して KJS プロセスで新しく利用し、このプロセスを iPlanet Application Server に追加します。

7. KJS に必要なスレッドはいくつですか。

   32 から始めて、プロセッサの負荷が高くなったら増やしてください。KJS に設定するスレッドプールは 48 までで十分です。これ以上になると、単にコンテキストの切り替えが増えるだけで、サイクルの無駄遣いになってしまいます。

8. KJS とプロセッサのバインドは、いつ検討すればよいでしょうか。

   一般には、KJS とプロセッサまたはプロセッサセットはバインドしません。バインドしてもパフォーマンスは 5% も増えません。JDK 1.2.2 以降では、VM はマルチプロセッサ環境で十分動作するように最適化されています。

9. KJS プロセスは Java VM の「ホーム」にあたりますが、KJS が管理する JVM のヒープサイズなどはどのように変更すればよいですか。

   Solaris JDK で提供される引数は、`iasenv.ksh` シェルスクリプトの `JAVA_ARGS` シェル変数を介して設定することができます。JVM フラグ、特に -Xms と -Xmx フラグでは、各 KJS エンジンで使われる開始ヒープサイズと最大ヒープサイズを指定します。ヒープサイズは、システムで使用できるメモリに基づいて決定されます。できるだけ大きく設定して、同じサーバで動作しているほかのアプリケーションに不足が生じないようにします。開始ヒープサイズのデフォルトは 8 MB です。

   ヒープは必要に応じて自動的に大きくなります。開始時にヒープサイズを大きくしておくと、拡張中に頻繁にガベージ収集が行われることがありません。ヒープサイズの拡張に上限を設ける場合は、`-Xmx` フラグを設定します。フラグについての詳細は、JDK のマニュアルを参照してください。

10. アプリケーションの動きが悪く、送信ボタンを押してからブラウザに結果が表示されるまで時間がかかります。どこで時間がかかるのかを知りたいのですが。

    リクエストプロセスサイクルのキーポイントに「クロック」を使ってみてください。

    - クロック 1 はクライアント (ブラウザや、`LoadRunner` などのロード生成ツール) でのタイムスタンプです。
    - クロック 2 は iWS Web サーバでのタイムスタンプで、フロントエンドスレッドがリクエストを受信する時刻です。
    - クロック 3 は iWS Web サーバでのタイムスタンプで、バックエンドのワーカースレッドがリクエストを受信する時刻です。
    - クロック 4 のタイムスタンプは、iPlanet Application Server Web コネクタがバックエンドワーカースレッドのリクエストに実際に応答する時刻です。
    - クロック 5 のタイムスタンプは、KXS がリクエストを受信する時刻です。リクエストは KJS に送信され、処理されます。
    - クロック 6 のタイムスタンプは、KJS から戻されたリクエストを KXS が受信する時刻です。

11. クロック 1 とクロック 2 の間で時間がかかっている場合、原因は何でしょうか。

    この場合の遅れには、次のいずれかの理由が考えられます。

    - ネットワークが混んでいる
    - クライアント側の CPU や NIC がビジーである
    - Web サーバで TCP スタックがキューに入っている
    - ロードバランサやファイヤウォールなどの中間部でリクエストがキューに入っている
    - ロード生成クライアントの設定が間違っている

12. クロック 2 とクロック 3 の間で 2 秒以上の遅延がある場合はどうでしょうか。

    バックエンドワーカースレッドがビジーで、リクエストがキューに入っていると考えられます。バックエンドのスレッド数を増やすか、フロントエンドのスレッド数を減らすことができます。

13. クロック 3 とクロック 4 の間で 30 ミリ秒以上の遅延がある場合はどうでしょうか。

    あまりよい対策はありません。Web コネクタが予想した速度のパフォーマンスを出していないということです。残念ながら、この場合チューニングできることはありません。

14. クロック 4 とクロック 5 の間で 3 秒以上の遅延がある場合はどうでしょうか。

    Web サーバの NIC でネットワークバッファリングが発生しているか、iPlanet Application Server の NIC でキューに入っている可能性があります。リソース不足のファイヤウォールも調べてみてください。通常のネットワークの混雑が原因の場合もあります。

15. クロック 5 とクロック 6 のタイムスタンプが予想よりも遅い場合は何が原因ですか。

    KXS や KJS プロセスをチューニングする必要があります。KXS と KJS のスレッドプールを調べてください。また、KJS プロセスを追加することも検討してください。それ以外にも、iPlanet Application Server プロセスをプロセッサや、場合によってはプロセッサセットにバインドすることを考慮する必要があります。

アプリケーションのチューニング

1. EJB プログラマですが、iPlanet Application Server の EJB コンテナのパフォーマンスを向上させるために何か調整することはありますか。

   - デフォルトの不活性化タイムアウトは 60 秒です。Beans インスタンスの作成速度が非常に遅く、Beans サイズが大きい場合は、この値を大きくします。不活性化プロセスが減少すると、パフォーマンスがやや向上することがあります。
   - メタデータキャッシュサイズを、アプリケーションに存在する Beans の種類の数にすることができます。デフォルトは 30 ですが、ホームが扱うすべてをキャッシュするには十分とは言えません。
   - `Implementation Cache Size` を、予想する同時ユーザセッション数に基づいて設定します。たとえば、200 の同時ユーザセッションがそれぞれ 1 つの状態を持つセッション EJB を必要とする場合、`Implementation Cache Size` を 200 以上に設定します。

2. iPlanet Application Server のパフォーマンスを向上させるには、Java コード内で何をチェックしたらよいでしょうか。

   iPlanet Application Server の KJS プロセスには、Java コードを実行する J2EE コンテナが含まれています。KJS プロセスは独立した Java VM を管理します。そのため、VM ができるだけパフォーマンスを発揮できるようにするには、正しい Java ガイドラインに従う必要があります。次の点を調べてみてください。

   - 直列化や直列化解除を避ける。これは時間がかかる操作です。
   - アレイの多用を避ける。アレイを初期化したり、範囲外のアクセスを防ぐなど、Java がアレイに行う操作にはオーバーヘッドが発生するためです。
   - 変数に null を設定して明示的に参照を解除し、ガベージコレクションの効率を上げます。
   - Servlet クラスでクラス変数 (スタティックメンバ) を使わない。サーバでの同期化が必要になるためです。デフォルトでは、ユーザ全員が Web コンテナ (JVM) ごとに Servlet コードのコピーを 1 つ共有しています。
   - 同期されたメソッドや同期されたブロックをコードで使わない。
   - EJB の使用は慎重に行う。EJB はサーバに負荷をかけます。

3. パフォーマンス上の理由から、アプリケーション内で EJB を慎重に使う必要がありますが、EJB の何が問題なのですか。

   EJB はすばらしいコンポーネントですが、サーバサイド Java のコンポーネントモデルです。また、コストがかかります。これはどのベンダの EJB サーバにも当てはまります。EJB にアクセスするには大量のオーバーヘッドが必要で、セキュリティやトランザクション管理など EJB コンテナが提供するサービスを管理するにもオーバーヘッドが必要です。このため、EJB を使う場合にはパフォーマンスの低下を最小限にするために、以下の点に注意してください。

   - Servlet で EJB 参照をキャッシュし、リクエストごとに JNDI 検索をする必要がないようにします。
   - スティッキーロードバランスを使い、セッション中の後続のリクエストが常に KJS (VM) と同じプロセスに戻るようにします。このようにすると EJB リソースはローカルになり、時間のかかる呼び出しをマシンを越えて行って EJB にアクセスすることが避けられます。
   - 状態のないセッション EJB を使います。パフォーマンスでは Servlet に匹敵します。
   - 状態を持つセッション Beans とエンティティ Beans は、状態のないセッション EJB Beans より時間がかかります。エンティティ EJB のパーシスタンス管理では BMP が CMP よりも時間がかかるために、エンティティ EJB のパフォーマンス集約度が最も高くなります。

4. アプリケーションで Servlet HTML 結果キャッシュを使うと、パフォーマンスはどのくらい向上しますか。

   CPU 1 個の場合、Servlet キャッシュによりパフォーマンスはおよそ 2 倍になりますが、CPU が 2 個の iPlanet Application Server インスタンスでのテストではわずか 4% しか向上しませんでした。CPU が 4 個以上の場合もキャッシュによるパフォーマンスの向上はなかったので、Servlet 結果キャッシュの使用はお勧めしません。代わりに JSP 結果キャッシュを使ってください。

   JSP 結果キャッシュは KXS ベースではなく KJS ベースです。この方法でうまく拡張できるようにするために、iPlanet Application Server インスタンスごとに複数の KJS プロセスを設定することができます。

5. どうすれば、もっとも効果的に Servlet HTML 結果キャッシュが行えますか。

   キャッシュを格納できるメモリを増やせば、キャッシュヒットの可能性を高めることができます。これは、Servlet キャッシュの配置記述子と JSP の Administration Tool で行います。

   HTML 結果キャッシュは Servlet の KXS で処理され、これはすでに過負荷になっていると思われる KXS にもう 1 つ作業をさせます。HTML 結果キャッシュを使うと確かにスループットが向上します。

6. JSP によってキャッシュが起きます。

   このキャッシュは KJS で処理されます。KXS プロセスとは違い、iPlanet Application Server インスタンスごとに複数の KJS プロセスを設定することができます。したがって、KXS を使わないようにして JSP 結果キャッシュを使うことにより、結果キャッシュを効果的に稼動させることができます。

索引

K

L

N

O

P

R

S

T

U

そ

た

て

は

ふ

ま

め

ゆ

り

ろ