Oracle9*i* Application Server

パフォーマンス・ガイド

リリース 2（9.0.2）

2002 年 7 月

部品番号：J05913-01

ORACLE®

Oracle9*i* Application Server パフォーマンス・ガイド , リリース 2（9.0.2）

部品番号：J05913-01

原本名：Oracle9*i* Application Server Performance Guide, Release 2 (9.0.2)

原本部品番号：A95102-02

原本著者：Thomas Van Raalte

原本協力者：Eric Belden, Paul Benninghoff, Alice Chan, Greg Cook, Marcelo Goncalves, Helen Grembowicz, Bruce Irvin, Pushkar Kapasi, Paul Lane, Sharon Malek, Valarie Moore, Carol Orange, Julia Pond, Leela Rao, Joan Silverman, Sanjay Singh, Cheryl Smith, Zhunquin Wang, Brian Wright

目次

2 Oracle9*i*AS の監視

3 Oracle HTTP Server の監視

4 OC4J の監視

5 Oracle HTTP Server の最適化

7 Web Cache の最適化

8 PL/SQL のパフォーマンスの最適化

A Oracle9*i*AS パフォーマンス・メトリック

索引

はじめに

このガイドでは、Oracle9*i* Application Server 環境におけるパフォーマンスの監視と最適化の方法、複数のコンポーネントを使用する際のパフォーマンス最適化の方法、およびパフォーマンスの高いアプリケーションの作成方法について説明します。

「はじめに」の項目は次のとおりです。

対象読者

『Oracle9*i* Application Server パフォーマンス・ガイド』では、対象読者としてインターネット・アプリケーション開発者、Oracle9*i* Application Server 管理者、データベース管理者および Web マスターを想定しています。

マニュアルの利用について

マニュアル内にある外部 Web サイトへのリンクの利用について　このマニュアルには、オラクル社が所有または管理していない他の企業や組織の Web サイトへのリンクが記載されている場合があります。オラクル社では、このような Web サイトの利用に関する評価も代弁も行いません。

このマニュアルの構成

このマニュアルは、次の章および付録から構成されています。

第 1 章「パフォーマンスの概要」

この章では、Oracle9*i*AS のパフォーマンスおよびチューニングの概要について説明します。

第 2 章「Oracle9iAS の監視」

この章では、Oracle Enterprise Manager などのパフォーマンス監視ツール、および Oracle9*i*AS 組込みの監視ツールを紹介します。

第 3 章「Oracle HTTP Server の監視」

この章では、Oracle Enterprise Manager および Oracle9*i*AS で使用可能な組込みパフォーマンス・ツールを使用した Oracle HTTP Server の監視について説明します。

第 4 章「OC4J の監視」

この章では、Oracle Enterprise Manager および Oracle9*i*AS で使用可能な組込みパフォーマンス・ツールを使用した Oracle9*i*AS Containers for J2EE（OC4J）の監視について説明します。

第 5 章「Oracle HTTP Server の最適化」

この章では、Oracle HTTP Server の最適化について説明します。

第 6 章「OC4J での J2EE アプリケーションの最適化」

この章では、Oracle9*i*AS Containers for J2EE 上で動作する J2EE アプリケーションの最適化について説明します。

第 7 章「Web Cache の最適化」
この章では、Web Cache の最適化について説明します。

第 8 章「PL/SQL のパフォーマンスの最適化」
この章では、`mod_plsql` を使用したコードの最適化について説明します。

付録 A「Oracle9iAS パフォーマンス・メトリック」
この付録には、Oracle9*i*AS コンポーネントのパフォーマンスの監視と分析に使用する統計およびメトリックを記載しています。

関連文書

詳細は、次に示す Oracle 製品の関連文書を参照してください。

- 『Oracle9*i* Application Server 概要』
- 『Oracle9*i* Application Server 管理者ガイド』
- 『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』
- 『Oracle9*i*AS Containers for J2EE ユーザーズ・ガイド』
- 『Oracle9*i* Application Server セキュリティ・ガイド』
- 『Oracle9*i*AS Web Cache 管理および配置ガイド』
- 『Oracle9*i*AS Containers for J2EE Enterprise JavaBeans 開発者ガイドおよびリファレンス』
- 『Oracle9*i*AS Containers for J2EE Servlet 開発者ガイド』
- 『Oracle9*i*AS Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』
- 『Oracle9*i* データベース・パフォーマンス・ガイドおよびリファレンス』
- 『Oracle9*i* Application Server PL/SQL Web Toolkit リファレンス』

リリース・ノート、インストレーション・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J（Oracle Technology Network Japan）の Web サイトから無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

```
http://otn.oracle.co.jp/membership
```

OTN-J のユーザー名とパスワードをすでに取得している場合は、次の OTN-J Web サイトの文書セクションに直接接続できます。

```
http://otn.oracle.co.jp/document
```

詳細は、次に示す Oracle 製品の関連文書を参照してください。

- このリリースについては、次の URL にある Oracle9*i*AS Portal パフォーマンスの情報を参照してください。

  ```
  http://otn.oracle.co.jp/
  ```

  Oracle Technology Network のメイン・ページから、次を選択します。

  - 「製品」リンクを選択
  - 「Oracle9*i* Application Server」の下にある「Oracle9*i*AS Portal」を選択

表記規則

この項では、このマニュアルの本文およびコード例に使用されている表記規則について説明します。ここで説明する内容は、次のとおりです。

- 本文の表記規則
- コード例の表記規則
- Windows オペレーティング・システムの表記規則

本文の表記規則

本文中には、特別な用語が一目でわかるように、様々な表記規則が使用されています。次の表に、本文の表記規則と使用例を示します。

表記規則	意味	例
太字	太字は、本文中で定義されている用語、用語集に記載されている用語、またはその両方に該当する用語を示します。	この句を指定することにより、**索引構成表**が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システムによって指定される要素を示します。この要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージとメソッド、システム指定の列名、データベース・オブジェクトおよび構造体、ユーザー名およびロールが含まれます。	この句は、NUMBER 列にのみ指定可能です。 データベースをバックアップするには、BACKUP コマンドを使用します。 USER_TABLES データのディクショナリ・ビューの TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザー指定要素のサンプルを示します。これらの要素には、コンピュータ名、データベース名、ネット・サービス名および接続識別子、さらにユーザー指定のデータベース・オブジェクトおよび構造体、列名、パッケージおよびクラス、ユーザー名およびロール、プログラム・ユニットおよびパラメータ値が含まれます。 **注意 :** プログラム要素の中には、大文字と小文字が混在して使用されているものもあります。これらの要素については、表示されているとおりに入力してください。	sqlplus と入力し、SQL*Plus を開きます。 パスワードは、orapwd ファイルに指定されています。 データファイルおよび制御ファイルを /disk1/oracle/dbs ディレクトリにバックアップします。 department_id、department_name および location_id 列は、hr.departments 表内に存在します。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーで接続します。 JRepUtil クラスによってこれらのメソッドが実装されます。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。*old_release* は、アップグレード前にインストール済のリリースを示します。

コード例の表記規則

コード例では、SQL、PL/SQL、SQL*Plus またはその他のコマンドライン構文が示されます。この中では、次の例のように、固定幅フォントが使用され、通常の本文とは区別して表示されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例に使用される表記規則と使用例を示します。

表記規則	意味	例
[ ]	大カッコはオプションの項目を示します。大カッコ自体は入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコは、カッコ内の項目のうちの 1 つを指定する必要があることを示します。中カッコ自体は入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の選択肢を示します。これらのオプションのうちの 1 つを入力します。縦線自体は入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平の省略記号は、次のどちらかを示します。	
	■ コード中で、例に直接関係のない部分が省略されていること。	CREATE TABLE ... AS *subquery*;
	■ コードの一部が繰返し可能であること。	SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	縦方向の省略記号は、コード中で、例に直接関係のない行が何行か省略されていることを示します。	
その他の表記	大カッコ、中カッコ、縦線および省略記号以外の記号は、表示されているとおりに入力してください。	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
イタリック体	イタリック体のテキストは、特定の値を指定する必要のあるプレースホルダまたは変数を示します。	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*

表記規則	意味	例

大文字	大文字で表記されている部分は、システムによって指定される要素を示します。ユーザーが定義する用語と区別するために、これらの用語は大文字で表記されます。用語が大カッコで囲まれている場合を除いて、表示されている順序およびスペルのとおりに入力します。ただし、これらの用語には大文字・小文字の区別がないため、小文字で入力しても構いません。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字で表記されている部分は、ユーザーが指定するプログラム要素を示します。たとえば、表、列またはファイルの名前を示します。
	注意 : プログラム要素の中には、大文字と小文字が混在して使用されているものもあります。これらの要素については、表示されているとおりに入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Windows オペレーティング・システムの表記規則

次の表に、Windows オペレーティング・システムに関する表記規則と使用例を示します。

表記規則	意味	例
「スタート」>	プログラムの起動方法を示します。	Oracle Database Configuration Assistant を起動するには、「スタート」>「プログラム」>「Oracle - *HOME_NAME*」>「Configuration and Migration Tools」>「Database Configuration Assistant」を選択します。
ファイル名およびディレクトリ名	ファイル名およびディレクトリ名では、大文字と小文字を区別しません。また、特殊文字のうち、左山カッコ（<）、右山カッコ（>）、コロン（:）、二重引用符（"）、スラッシュ（/）、パイプ（|）およびダッシュ（-）は使用できません。円記号（¥）は、引用符に囲まれている場合でも、要素の区切り文字として扱われます。ファイル名が¥¥で始まる場合、Windows では汎用命名規則に従った名前であると見なされます。	`c:¥winnt"¥"system32` と `C:¥WINNT¥SYSTEM32` は、同じように解釈されます。
`C:¥>`	現在のハード・ディスク・ドライブを表す Windows コマンド・プロンプトを示します。コマンド・プロンプトでのエスケープ文字はカレット（^）です。表示されるプロンプトは、作業中のサブディレクトリを表します。このマニュアルでは、コマンド・プロンプトと呼びます。	`C:¥oracle¥oradata>`

	Windows コマンド・プロンプトで二重引用符（"）を使用するときには、円記号（¥）をエスケープ文字として付けなければならない場合があります。カッコおよび一重引用符（'）の場合には、エスケープ文字を付ける必要はありません。エスケープ文字および特殊文字の詳細は、ご使用の Windows オペレーティング・システムのドキュメントを参照してください。	`C:¥>exp scott/tiger TABLES=emp QUERY=¥"WHERE job='SALESMAN' and sal<1600¥"`
		`C:¥>imp SYSTEM/`*`password`*
		`FROMUSER=scott TABLES=(emp, dept)`

HOME_NAME	Oracle ホーム名を示します。ホーム名は、最大 16 文字の英数字で指定できます。ホーム名に使用できる特殊文字はアンダースコアのみです。	`C:¥> net start Oracle`*`HOME_NAME`*`TNSListener`

表記規則	意味	例
ORACLE_HOME および *ORACLE_BASE*	Oracle8 リリース 8.0 以下の製品では、Oracle のコンポーネントをインストールすると、最上位の *ORACLE_HOME* ディレクトリの下に、すべてのサブディレクトリが配置されます。このディレクトリには、デフォルトで次のいずれかの名前が使用されます。 ■ C:¥orant（Windows NT の場合） ■ C:¥orawin98（Windows 98 の場合） このリリースは、Optimal Flexible Architecture（OFA）のガイドラインに準拠しています。すべてのサブディレクトリが最上位の *ORACLE_HOME* ディレクトリの下にあるとは限りません。*ORACLE_BASE* と呼ばれる最上位ディレクトリがあり、デフォルトでは C:¥oracle になります。Oracle ソフトウェアが 1 つもインストールされていないコンピュータに Oracle9*i* リリース 1（9.0.1）をインストールすると、初めて作成される Oracle ホーム・ディレクトリのデフォルト設定は C:¥oracle¥ora90 になります。Oracle ホーム・ディレクトリは、*ORACLE_BASE* の直下にあります。 このマニュアルの中で例として使用されているディレクトリ・パスは、OFA の規則に準拠しています。	%*ORACLE_HOME*%¥rdbms¥admin ディレクトリに移動します。

1

パフォーマンスの概要

この章では、Oracle9*i* Application Server のパフォーマンスとチューニングの概要を説明します。

この章には、次の項が含まれています。

- Oracle9iAS パフォーマンスの概要
- パフォーマンスのチューニングとは
- パフォーマンス・ターゲット
- パフォーマンス管理の方法

関連項目： Oracle9*i* Application Server の概要についての説明は、『Oracle9*i* Application Server 概要』を参照してください。

Oracle9*i*AS パフォーマンスの概要

Oracle9*i* Application Server のパフォーマンスを最大限に発揮するには、すべてのコンポーネントの監視、分析およびチューニングが必要です。この章では、パフォーマンス監視に使用するツールと、Oracle HTTP Server、Oracle9*i*AS Containers for J2EE（OC4J）などの Oracle9*i*AS コンポーネントのパフォーマンスを最適化するテクニックについて説明します。

パフォーマンスに関する用語

次に、このマニュアルで使用されているパフォーマンスに関する用語を示します。

同時実行性　複数のリクエストを同時に処理する能力。同時実行性メカニズムの例には、スレッドおよびプロセスがあります。

競合　リソースの競合。

ハッシュ　アルゴリズムを使用してテキスト文字列から生成された数値。通常、ハッシュ値はテキストに比べてかなり小さくなります。ハッシュ数値は、セキュリティ、およびデータへの高速なアクセスの目的で使用されます。

レイテンシ　全体のタスクを完了するために、あるシステム・コンポーネントが別のコンポーネントを待機している時間。レイテンシは、無駄な時間として定義できます。ネットワークのコンテキストでは、レイテンシはパケットのソースから宛先への移動時間として定義されます。

応答時間　リクエストの送信から応答の受信までの時間。

スケーラビリティ　使用可能なハードウェア・リソースに比例して、そして使用可能なハードウェア・リソースによってのみ制限された状態でシステムが**スループット**を提供できる能力。スケーラブルなシステムとは、応答時間および**スループット**に悪影響を与えずに、増加したリクエストの処理が可能なシステムです。

サービス時間　リクエストの受信からリクエストへの応答完了までの時間。

思考時間　ユーザーが実際にプロセッサを使用していない時間。

スループット　単位時間当たりに処理されるリクエスト数。

待機時間　リクエストの送信からリクエストの開始までの時間。

パフォーマンスのチューニングとは

パフォーマンスは、事前に設定しておく必要があります。アプリケーションの分析および設計中にパフォーマンス要件を予測し、最適なパフォーマンスのコストと利益を考慮する必要があります。この項では、次のような基本概念について説明します。

- 応答時間
- システム・スループット
- 待機時間
- 重要なリソース
- 過度の需要による影響
- 問題解決のための調整

> **関連項目：** パフォーマンス要件、およびシステムのどの部分をチューニングするかを判別する方法については、1-8 ページの「パフォーマンス・ターゲット」を参照してください。

応答時間

応答時間は**サービス時間**と**待機時間**の合計であるため、次の方法でパフォーマンスを向上できます。

- **待機時間**を削減する
- **サービス時間**を削減する。

図 1-1 に、1 つのリソースに対して 10 個の順次タスクが時間の経過に沿って競合している状態を示します。

図 1-1 個別タスクの順次処理

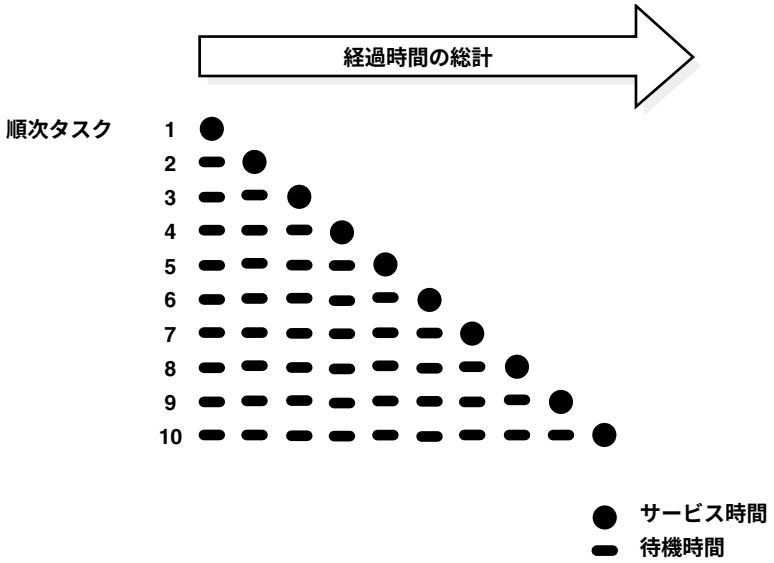


図 1-1 で示す例では、タスク 1 のみが待機時間なしで実行されます。タスク 2 はタスク 1 が完了するまで待機し、タスク 3 はタスク 1 と 2 が完了するまで待機する必要があります。他のタスクについても同様です。この図では各タスクの大きさは同じですが、実際のタスクのサイズはそれぞれ異なります。

複数のリソースを使用したパラレル処理の場合、より多くのリソースをタスクに割り当てることができます。各タスクは専用のリソースを使用してすぐに実行され、**待機時間**が発生しません。

Oracle HTTP Server では、このように、クライアントのリクエストを使用可能な `httpd` プロセスに割り当てて処理します。`MaxClients` パラメータにより、クライアントのリクエストを同時に処理可能な `httpd` プロセス数の上限を指定します。使用中のプロセス数が `MaxClients` 値に達すると、リクエストが完了して、プロセスが解放されるまで、サーバーは接続を拒否します。

関連項目： 第 5 章「Oracle HTTP Server の最適化」

システム・スループット

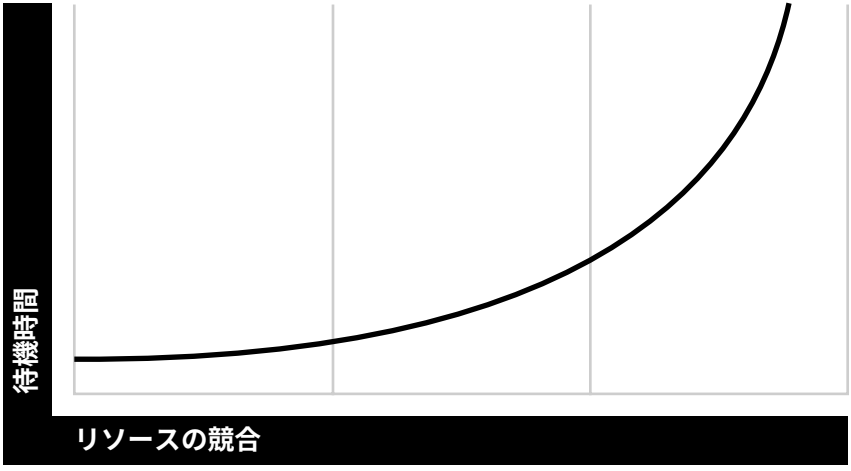
システム・**スループット**は、一定時間内に完了する処理量です。**スループット**は、次の方法で増加できます。

- **サービス時間を削減する。**
- 不足しているリソースを増加することにより、全体の**応答時間**を削減する。たとえば、システムが CPU の限界に達している場合、CPU リソースを追加することでパフォーマンスを改善できます。

待機時間

1 つのタスクの**サービス時間**が同じ場合でも、**競合**が増加すると**待機時間**は長くなります。1 秒を要するサービスを多数のユーザーが待っている場合、10 番目のユーザーは 9 秒間待機する必要があります。図 1-2 に、**待機時間**とリソースに対する**競合**の関係を示します。この図のグラフは、リソースの競合が増加するに連れて、待機時間が指数関数的に増加することを示しています。

図 1-2 リソースに対する競合の増加による待機時間の増加



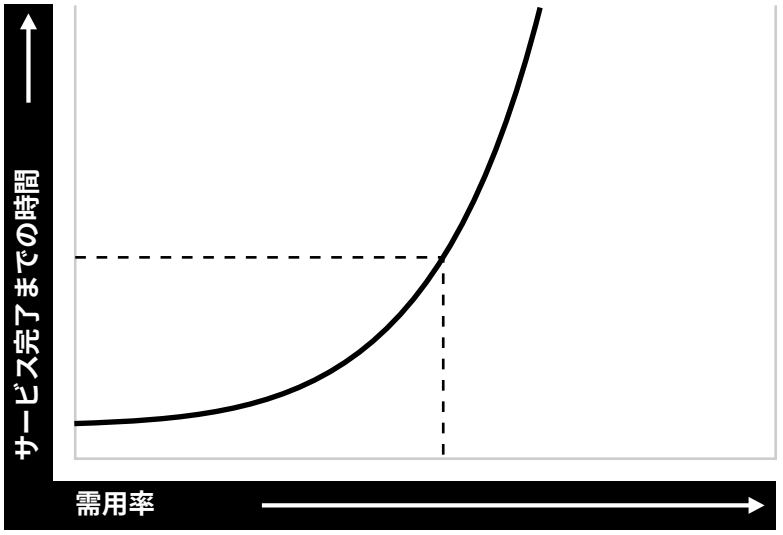
重要なリソース

CPU、メモリー、I/O 容量およびネットワーク帯域幅などのリソースは、**サービス時間**短縮の重要な要素となります。リソースを増加すると、**スループット**が増加し、**応答時間**も短縮できます。パフォーマンスは次の要因に依存します。

- 使用可能なリソースの量
- リソースを必要とするクライアントの数
- リソースに対するクライアントの待機時間
- クライアントがリソースを保持する時間

図 1-3 に、サービスの完了までにかかる時間と需用率との関係を示します。この図のグラフは、リクエストの単位数が増加するに連れて、サービスにかかる時間が増加していることが示しています。

図 1-3 サービス完了までの時間と需用率



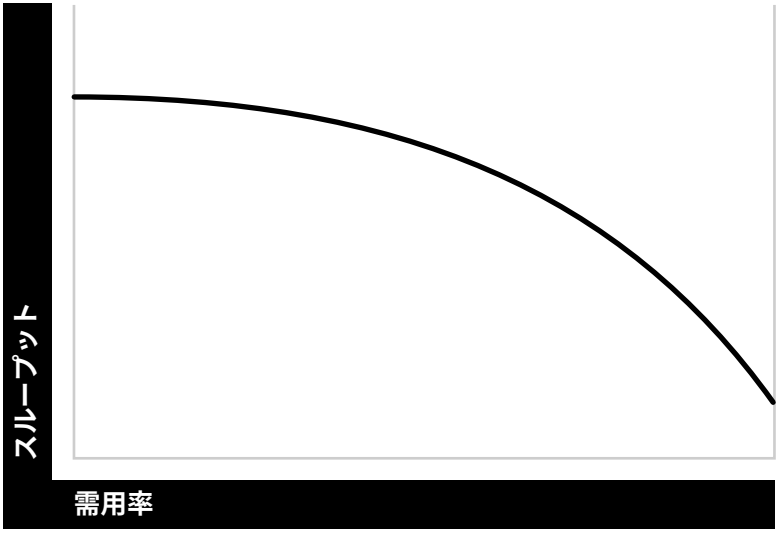
この状況を解消するには、2 つの方法があります。

- 許容範囲の応答時間を維持するために需用率を制限する
- リソースを追加する

過度の需要による影響

過度の需要により、**応答時間**が増加し、**スループット**が減少します。この様子を図 1-4 のグラフに示します。

図 1-4 需要の増加とスループットの減少



需用率が**スループット**の限界を越えた場合、監視によって、どのリソースが使い果たされてしまったかを判断し、可能であれば、そのリソースを増加します。

問題解決のための調整

パフォーマンスに関する問題は、次のような調整により解決できます。

- 単位消費

  リクエスト当たりのリソース（CPU、メモリー）の消費の削減により、パフォーマンスを改善できます。これは、プーリングおよびキャッシングにより実現できます。

- 機能面での需要

  問題によっては、処理のスケジュールを変更したり、処理を分散しなおすことによって解決できます。

- 容量

  リソース（CPU など）の増加や再割当てによって問題を解決できる場合があります。

パフォーマンス・ターゲット

システムを設計する場合もメンテナンスを行う場合も、最適化の手段および対象を判断できるよう、具体的なパフォーマンス目標を設定する必要があります。特定の目標を持たずにパラメータを変更すると、目立った効果もなくシステムのチューニングに余分な時間を費やすことになります。

具体的なパフォーマンス目標の例として、注文入力の**応答時間**を 3 秒以内にする、などがあります。アプリケーションがその目標を達成できない場合、原因（たとえばI/O **競合**など）を識別して対処します。開発中にアプリケーションをテストして、設計時に設定されたパフォーマンス目標を達成できるかどうかを調べます。

通常、チューニングには他の面とのトレード・オフが発生します。ボトルネックを判断できたら、目標を達成するために、他の部分のパフォーマンスを変更する必要がある場合もあります。たとえば、I/O が問題である場合、メモリーまたはディスクの購入が必要な場合があります。購入できない場合は、目標のパフォーマンスを得るためにシステムの**同時実行性**を制限する必要があります。ただし、パフォーマンスの目標が明確に定まっていれば、何が最も重要かがわかっているため、パフォーマンス向上のために何を犠牲にするかの判断が容易になります。

ユーザーの期待

アプリケーション開発者、データベース管理者およびシステム管理者は、ユーザーが期待しているパフォーマンスを、注意しながら適切に設定する必要があります。システムが特に複雑な処理を行っている場合は、単純な処理を行っている場合よりも**応答時間**が長くなる可能性があります。どの処理に時間がかかるかを明確にユーザーに知らせる必要があります。

パフォーマンス評価

パフォーマンス目標を明確に定めると、パフォーマンスのチューニングが成功したかどうか、容易に判断できます。チューニングの成功を左右するのは、ユーザー・コミュニティに対して設定した機能面での目標、基準が満たされたかどうかを判断する能力、そして例外事項を解決するための対策を講じる能力です。

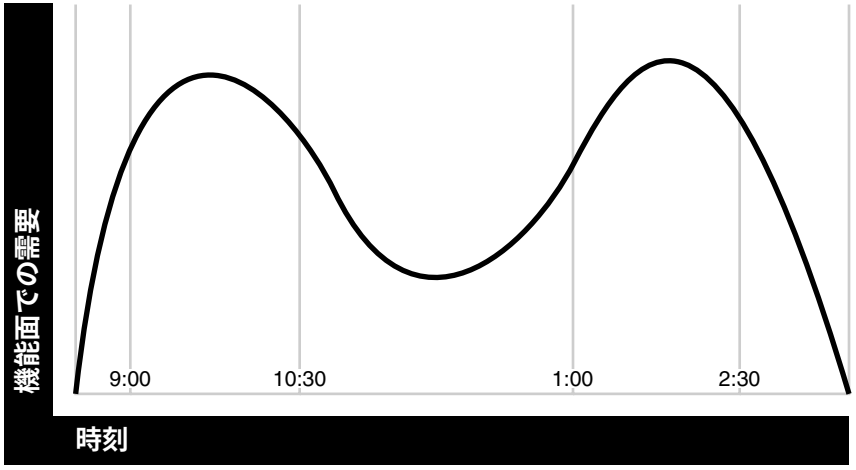
常にパフォーマンスを監視することにより、十分にチューニングされたシステムを維持できます。アプリケーションのパフォーマンスの履歴を記録することにより、有効な比較が可能となります。様々な大きさの負荷に関する実際のリソース消費データを使用して、客観的な**スケーラビリティ**の調査を行うことにより、予期される負荷のボリュームに合わせたリソース要件を予測できます。

パフォーマンス管理の方法

システムの最適効率を実現するためには、計画、監視および定期的な調整が必要です。パフォーマンス・チューニングの最初のステップは、目標を決定し、使用可能なテクノロジを効率的にアプリケーションに使用するよう設計することです。システムの実装後は、システムの監視と調整を定期的に行う必要があります。たとえば、ユーザーの 90% について**応答時間**が 5 秒以下であり、全ユーザーについても最大で 20 秒以下の**応答時間**であるといったことを確認したいとします。通常、これは簡単なことではありません。アプリケーションには、それぞれ特徴および許容できる応答時間が異なる様々な処理が含まれます。それぞれのアプリケーションに対し、適切な目標を設定する必要があります。

また、負荷の変動も判別する必要があります。たとえば、図 1-5 のグラフに示すように、ユーザーはシステムに午前 9 時から 10 時に集中的にアクセスし、再び午後 1 時から 2 時に集中的にアクセスする場合があります。たとえば、毎日あるいは毎週など、定期的に負荷のピークが発生する場合、一般的には負荷のピーク時の要件に合わせてシステムを設定し、チューニングします。ピーク時以外にアプリケーションにアクセスするユーザーに対する**応答時間**は、ピーク時のユーザーよりも短くなります。負荷のピークが頻繁に発生しない場合は、少ないハードウェア構成でコストを抑えるために、負荷のピーク時には**応答時間**が長くても我慢することも考えられます。

図 1-5 容量と機能面での需要の調整



パフォーマンス改善の要因

パフォーマンスは、様々な領域にまたがっています。

- サイズ設定と構成 : パフォーマンス目標をサポートするために必要なハードウェアのタイプの判断。
- パラメータのチューニング : アプリケーションの最高のパフォーマンスを実現するための、設定可能なパラメータの設定。
- パフォーマンスの監視 : アプリケーションが使用しているハードウェア・リソースおよびユーザーが費やしている**応答時間**の判断。
- トラブルシューティング : アプリケーションが過度にハードウェア・リソースを使用していたり、**応答時間**が目標よりも長い場合の理由の診断。

2

Oracle9*i*AS の監視

この章では、Oracle9*i*AS とそのコンポーネントのパフォーマンス監視方法について説明します。パフォーマンス・データを取得すると、Oracle9*i*AS のチューニング、およびパフォーマンスに問題のあるアプリケーションのチューニングとデバッグが容易になります。

この章には、次の項目が含まれています。

Oracle9*i*AS 監視の概要

この項では、パフォーマンスの監視に利用できるツールの使用方法について説明します。次の中の 1 つまたは複数を使用することで、Oracle9*i*AS とそのコンポーネントの監視が可能になります。

- Oracle Enterprise Manager
- Oracle9iAS の組込みパフォーマンス・メトリック
- オペレーティング・システム固有のパフォーマンス・コマンド
- ネットワーク・パフォーマンス監視ツール

Oracle Enterprise Manager

Oracle Enterprise Manager を使用すると、Oracle9*i*AS とそのコンポーネントの監視が可能になります。Oracle Enterprise Manager では、次のような Oracle9*i*AS コンポーネントのパフォーマンス・メトリックが表示されます。

- Oracle HTTP Server（OHS）
- Oracle9*i*AS Containers for J2EE （OC4J）および OC4J 下で稼動するアプリケーション
- Oracle9*i*AS Web Cache
- Oracle9*i*AS Portal

関連項目：

- 第 3 章「Oracle HTTP Server の監視」
- 第 4 章「OC4J の監視」
- 『Oracle9*i* Application Server 管理者ガイド』

Oracle9*i*AS の組込みパフォーマンス・メトリック

Oracle9*i*AS は自動的に実行時のパフォーマンスを測定し、子サーバーおよび Oracle9*i*AS Containers for J2EE（OC4J）サーバーも含めた Oracle HTTP Server のメトリックを収集します。Oracle9*i*AS のパフォーマンス・メトリックは、Oracle9*i*AS コンポーネントの実装に組み込まれたパフォーマンス測定処理を使用して、自動的かつ連続的に測定されます。パフォーマンス・メトリックは自動的に使用可能になります。パフォーマンス・メトリックを収集するためにオプションを設定したり、追加の構成を行う必要はありません。

Oracle HTTP Server のパフォーマンス・メトリックでは次のことが可能です。

- Oracle HTTP Server のリクエスト処理における重要なフェーズ期間中の監視。
- Oracle HTTP Server リクエストに関するステータス情報の収集。たとえば、任意の瞬間のリクエスト処理数を監視できます。

OC4J パフォーマンス・メトリックでは、J2EE コンテナのパフォーマンス監視の他、次のことを実行可能です。

- アクティブなサーブレット、JSP、EJB および EJB メソッドの数の監視。
- 各サーブレット、JSP、EJB または EJB メソッドの処理時間の監視。
- サーブレット、JSP、EJB または EJB メソッドに関連するセッションと JDBC 接続の監視。

Oracle9*i*AS コンポーネントのトラブルシューティングにパフォーマンス・メトリックを使用して、ボトルネックの発見、リソース可用性の問題特定、あるいはコンポーネントのスループットや応答時間の改善に役立てることができます。

注意： コマンドを使用して、スクリプトの組込みメトリックを利用したり、他の監視ツールと連動したりすることにより、パフォーマンス・データの収集またはアプリケーション・パフォーマンスのチェックを行うことができます。

オペレーティング・システム固有のパフォーマンス・コマンド

パフォーマンスの問題解決やシステム・アクティビティの監視に、オペレーティング・システム固有のコマンドを使用できます。オペレーティング・システム固有のコマンドを使用すると、CPU 使用率、ページング・アクティビティ、スワッピングやその他のシステム・アクティビティ情報の収集と監視が可能になります。

関連項目： オペレーティング・システム固有の監視用コマンドについての情報は、システム・レベルのドキュメントを参照してください。

ネットワーク・パフォーマンス監視ツール

ネットワーク監視ツールを使用して、Oracle9*i*AS コンポーネントにアクセスするリクエストのステータスを確認できます。ネットワークの通信量情報を調査および保存するツールを使用できます。このようなツールはパフォーマンスの問題の分析と解決に役立ちます。

Oracle9*i*AS の組込みパフォーマンス・メトリックの使用

Oracle Enterprise Manager を使用するか、Oracle9*i*AS の組込みパフォーマンス・メトリックを参照して、Oracle9*i*AS パフォーマンスの監視を実行できます。

この項では、`AggreSpy` サーブレットまたは `dmstool` コマンドを使用した、Oracle9*i*AS の組込みパフォーマンス・メトリックの表示方法について説明します。表 2-1 に、組込みパフォーマンス・メトリックの表示方法を簡単に説明します。

表 2-1　Oracle9*i*AS の組込み監視コマンド

コマンド	説明
`AggreSpy`	`AggreSpy` は、パフォーマンス・メトリックをレポートするパッケージ済のサーブレットです。`AggreSpy` は、Oracle9*i*AS インスタンスのパフォーマンス・データをレポートします。`AggreSpy` を実行できるのは、OC4J インスタンスがこれをサポートするように構成されている場合に限られます。OC4J ホーム・インスタンスは、デフォルトで `AggreSpy` をサポートしています。
`dmstool`	1 つのパフォーマンス・メトリック、すべてのパフォーマンス・メトリック、または任意の数のパフォーマンス・メトリックを監視できます。オプションを使用すると、リクエストしたメトリックを *t* 秒ごとにレポートするようにレポート間隔を指定できます。このコマンドでは、サイト上のすべての組込みパフォーマンス・メトリックもレポートできます。 `dmstool` は、`$ORACLE_HOME/bin` ディレクトリにあります。

関連項目： 付録 A「Oracle9iAS パフォーマンス・メトリック」

AggreSpy を使用したパフォーマンス・メトリックの表示

AggreSpy サーブレットは、Oracle HTTP Server および OC4J のパフォーマンス・メトリックを表示します。AggreSpy では HTML の表を使用して読みやすく出力され、複数の OC4J を実行している場合は、複数の OC4J インスタンスからの OC4J メトリックが出力されます。

Oracle9*i*AS の組込みメトリックを含む表は、例えば Oracle HTTP Server のメトリックの場合は ohs_server など、名前で識別されます。このガイドでは、組込みパフォーマンス・メトリックの表名を**メトリック表**と呼びます。

パフォーマンス・メトリックには、次の URL から AggreSpy を使用してアクセスできます。

```
http://myhost:myport/dmsoc4j/AggreSpy
```

注意： AggreSpy を実行できるのは、OC4J インスタンスがこれをサポートするように構成されており、このインスタンスが実行中の場合に限られます。OC4J home インスタンスはデフォルトで AggreSpy をサポートしています。

図 2-1 に、AggreSpy レスポンスのサンプルを示します。AggreSpy レスポンスは 2 つのフレームで表示されます。1 つはメトリック表リストを含む左側のフレームで、もう 1 つは選択されたパフォーマンス・メトリックの現在値を表示する右側のフレームです。

AggreSpy には、次のようなナビゲーションと表示オプションが用意されています。

- 他のメトリック表へのアクセスに使用する左フレーム内のリンク
- メトリック表の列ヘッダーのクリックよる行のソート
- 「Raw」または「XML」リンクのクリックによる RAW 形式、XML 形式での表示

AggreSpy はパフォーマンス・データのキャッシュを行うため、レポートされている AggreSpy のメトリック値は最新のデータを表示しているとは限りません。AggreSpy では、データ値のアクセス頻度に応じてメトリックがリフレッシュされます（値は最小で 15 秒おきにリフレッシュされます）。

注意： AggreSpy の起動後は、ブラウザをリフレッシュして組込みメトリック・データを表示してください。AggreSpy を最初に使用するときに、多くのフィールド、およびメトリック表の全リストにデータがなく、空白になっている場合があります。しばらく待った後で表示をリフレッシュすると、データが利用できるようになりデータ値が表示されます。

AggreSpy の URL とアクセス制御

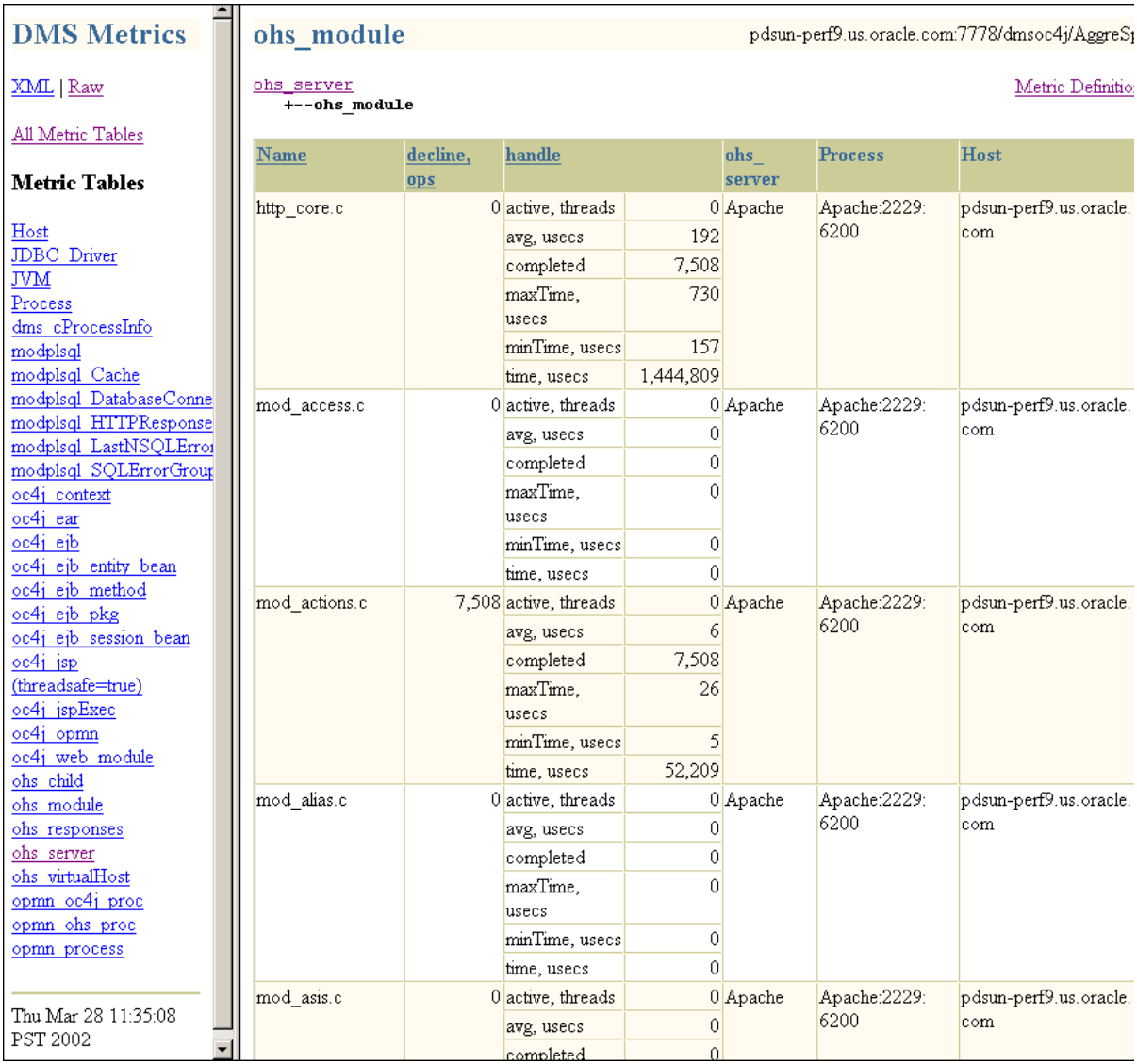
デフォルトで、URL dmsoc4j/AggreSpy は保護されており、ローカル・ホストからのみメトリックへアクセスできます。ローカル・ホスト以外のシステムからメトリックを表示したい場合には、Oracle HTTP Server の $ORACLE_HOME/Apache/Apache/conf/mod_oc4j.conf ファイルを編集して、dmsoc4j のアクセス制御を変更する必要があります。

AggreSpy サーブレットへのデフォルト・パスは dmsoc4j/AggreSpy です。dmsoc4j アプリケーションへの URL が変更されたり、デフォルト・アプリケーションが使用不可になっている場合、AggreSpy サーブレットへアクセスする有効なパスを決定するために、$ORACLE_HOME/Apache/Apache/conf/mod_oc4j.conf ファイルを更新する必要があります。

関連項目：

- mod_oc4j の構成についての情報は『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』を参照してください。
- Oracle HTTP Server のアクセス制御についての情報は、『Oracle9*i* Application Server セキュリティ・ガイド』を参照してください。

図 2-1 AggreSpy によるパフォーマンス・メトリックの表示



DMS Metrics

XML | Raw

All Metric Tables

Metric Tables

Host
JDBC_Driver
JVM
Process
dms_cProcessInfo
modplsql
modplsql_Cache
modplsql_DatabaseConne
modplsql_HTTPResponse
modplsql_LastNSQLError
modplsql_SQLErrorGroup
oc4j_context
oc4j_ear
oc4j_ejb
oc4j_ejb_entity_bean
oc4j_ejb_method
oc4j_ejb_pkg
oc4j_ejb_session_bean
oc4j_isp
(threadsafe=true)
oc4j_jspExec
oc4j_opmn
oc4j_web_module
ohs_child
ohs_module
ohs_responses
ohs_server
ohs_virtualHost
opmn_oc4j_proc
opmn_ohs_proc
opmn_process

Thu Mar 28 11:35:08 PST 2002

ohs_module

pdsun-perf9.us.oracle.com:7778/dmsoc4j/AggreSp

ohs_server
+--ohs_module

Metric Definitio

Name	decline, ops	handle		ohs_server	Process	Host
http_core.c	0	active, threads	0	Apache	Apache:2229:6200	pdsun-perf9.us.oracle.com
		avg, usecs	192			
		completed	7,508			
		maxTime, usecs	730			
		minTime, usecs	157			
		time, usecs	1,444,809			
mod_access.c	0	active, threads	0	Apache	Apache:2229:6200	pdsun-perf9.us.oracle.com
		avg, usecs	0			
		completed	0			
		maxTime, usecs	0			
		minTime, usecs	0			
		time, usecs	0			
mod_actions.c	7,508	active, threads	0	Apache	Apache:2229:6200	pdsun-perf9.us.oracle.com
		avg, usecs	6			
		completed	7,508			
		maxTime, usecs	26			
		minTime, usecs	5			
		time, usecs	52,209			
mod_alias.c	0	active, threads	0	Apache	Apache:2229:6200	pdsun-perf9.us.oracle.com
		avg, usecs	0			
		completed	0			
		maxTime, usecs	0			
		minTime, usecs	0			
		time, usecs	0			
mod_asis.c	0	active, threads	0	Apache	Apache:2229:6200	pdsun-perf9.us.oracle.com
		avg, usecs	0			
		completed	0			

dmstool を使用したパフォーマンス・メトリックの表示

dmstool コマンドを使用すると、1 つのパフォーマンス・メトリック、すべてのパフォーマンス・メトリック、または任意の数のパフォーマンス・メトリックを表示できます。dmstool コマンドでは、メトリックのレポートを *t* 秒ごとに更新するため、レポートの作成間隔を秒単位で指定するオプションもサポートされています。

たとえば、特定のサーブレット、JSP、EJB、EJB メソッドまたはデータベース接続のパフォーマンスを監視しながら、これらのコンポーネントに関するメトリックの定期的なスナップショットをリクエストすることができます。

組込みパフォーマンス・メトリックの表示に使用する dmstool の書式は次のとおりです。

```
% dmstool [options] metric metric ...
```

または

```
% dmstool [options] -list
```

または

```
% dmstool [options] -dump
```

表 2-2 に、dmstool のコマンドライン *options* を示します。表 2-2 の後のセクションでは、いくつかのパフォーマンス・メトリックに関する使用方法のサンプルを示します。dmstool は、$ORACLE_HOME/bin ディレクトリにあります。

注意： dmstool をスクリプト内で、または他の監視ツールと組み合せて使用して、パフォーマンス・データの収集やアプリケーション・パフォーマンスのチェックを行ったり、パフォーマンス・メトリックの値に基づいてシステムの修正ツールを作成することができます。

---

dmstool のアクセス制御

デフォルトでは、dmstool をローカル・ホストから実行している場合のみメトリックを取得できます。リモート・ホストで稼動している Oracle HTTP Server のメトリックを表示する場合は、-a オプションを使用して dmstool を実行し、Oracle HTTP Server の $ORACLE_HOME/Apache/Apache/conf/httpd.conf ファイルを編集して /dms0 のアクセス制御を変更する必要があります。

表 2-2 dmstool のコマンドライン・オプション

オプション	説明

–a[ddress] *host*[:*port*]	-a オプションがない場合、dmstool はデフォルトでローカル・ホストからメトリックを取得します。Oracle HTTP Server が dmstool と同じシステム上で稼動している場合、–a オプションは必要ありません。
	
	コマンドライン中で –a を複数指定して、クラスタを監視することもできます。
	
	host は、Oracle HTTP Server が稼動しているホストのドメイン名または IP アドレスです。
	
	port には、メトリックを提供する OPMN リクエストのポートを指定します。このリクエスト・ポートは、$ORACLE_HOME/opmn/conf/opmn.xml で次のように指定します。
	
	`<notification-server>`
	`<port local="6100" remote="6200" request="6003"/>`
	`<log-file path="/private/oracle/opmn/logs/ons.log" level="3"/>`
	`</notification-server>`

–c[ount] *num*	メトリックを監視する際に、値を取得する回数を指定します。指定しない場合、dmstool は、プロセスが停止されるまでメトリック値を取得し続けます。 –count オプションを、–list オプションと同時に使用することはできません。
–dump	-dump オプションを使用して dmstool を実行すると、Oracle9*i*AS インスタンスからのすべてのメトリックが標準出力にレポートされます。-dump オプションを使用したコマンドを 10 ～ 15 分などの間隔で定期的に実行して、使用している Oracle9*i*AS サーバーのパフォーマンス・データを取得し、記録を保存しておくことをお薦めします。
–i[nterval] *secs*	メトリックを取得してから、次に取得するまで待機する秒数を指定します。デフォルトは 1 秒です。interval 引数を、–list オプションと同時に使用することはできません。指定する時間間隔は近似値になります。 **注意 :** システムの負荷が高い場合、–interval オプションで指定した間隔は実際の間隔と異なることがあります。
–l[ist]	使用可能なすべてのメトリックのリストを生成します。–list オプションを使用して dmstool を実行すると、コマンドラインで指定しているメトリック名は無効になります。
–table *metric_table*	*metric_table* で指定した名前のメトリック表にある、すべてのパフォーマンス・メトリックを含みます。 メトリック表の名前のリストは、付録 A「Oracle9iAS パフォーマンス・メトリック」を参照するか、AggreSpy を実行してください。

すべてのメトリック名をリストするための dmstool の使用方法

Oracle9*i*AS の各パフォーマンス・メトリックには一意の名前があります。–list を使用して dmstool を実行すると、すべてのメトリック名のリストが作成されます。–list による出力には、1 つのメトリックまたは複数のメトリックの監視を要求するときに、dmstool で使用できるメトリック名が含まれています。

次のコマンドを使用すると、dmstool は Oracle HTTP Server で取得可能なすべてのメトリックのリストを表示します。

```
% dmstool –list
```

このコマンドにより、取得可能なメトリックのリストが表示されます。

関連項目： 付録 A「Oracle9iAS パフォーマンス・メトリック」

特定のパフォーマンス・メトリックをレポートするための dmstool の使用方法

1 つメトリックまたは複数のメトリックを監視するには、コマンドラインでメトリック名を指定して dmstool を使用します。たとえば、JVM の稼働時間を監視するには次の手順を実行します。

1. -list オプションを使用し、dmstool を次のように実行して、JVM のアップタイムを示すメトリック名を見つけます。

   ```
   % dmstool -list | grep JVM/upTime.value
   /myhost/OC4J:3000:6003/JVM/upTime.value
   ```

2. 引数として、このメトリックの完全なパス名を次のように与えて dmstool を実行し、メトリックの現在の値を表示します。

   ```
   % dmstool /myhost/OC4J:3000:6003/JVM/upTime.value
   Tue Apr 02 16:27:32 PST 2002
   /myhost/OC4J:3000:6003/JVM/upTime.value       23991009 msecs
   ```

ホストの 2 つの OC4J プロセス間で負荷を均衡化する場合、各 OC4J で処理されているリクエスト数の経過を監視することがあります。取得可能なメトリックのリストを dmstool –list コマンドを使用して生成し、このリストで OC4J の情報を検索すると、次のようなメトリック名が表示されます。

```
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojsp/JSPs/processRequest.completed
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojsp/processRequest.completed
/myhost/OC4J:3001:6003/oc4j/ojsp/WEBs/processRequest.completed
```

この dmstool –list による出力は、このサイトに 2 つの OC4J プロセスがあることを示しています。AJP ポートの 3000 でリスニングしている OC4J は、default というアプリケーションを実行しており、一方、AJP ポートの 3001 でリスニングしている OC4J は ojsp というアプリケーションを実行しています。ojsp アプリケーションは JSP にアクセスしていますが、default アプリケーションはアクセスしていません。

Interval および Count オプションを使用した dmstool の実行

場所が明らかな 2 つの OC4J プロセス間のロード・バランスを 2 時間監視するには、複数のメトリック名をコマンドラインで指定して、次のようにコマンドを実行します。

```
% dmstool -i 60 -c 120 \
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed \
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojsp/JSPs/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojsp/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/processRequest.completed
```

このコマンドでは、–c オプションで指定したように、コマンドラインでリストしたメトリックの出力が 120 回レポートされ、同時に 60 秒間隔でデータが収集されます。

```
                Mon Nov 19 17:13:01 PDT 2001
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed           437 ops
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed                   441 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed           432 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed                   436 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/JSPs/processRequest.completed  452 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/processRequest.completed       425 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/processRequest.completed                 455 ops

                Mon Nov 19 17:14:01 PDT 2001
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed           452 ops
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed                   470 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed           462 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed                   451 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/JSPs/processRequest.completed  469 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/processRequest.completed       452 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/processRequest.completed                 472 ops
                .
                .
                .
```

すべてのメトリックとそのメトリック値をレポートするための dmstool の使用方法

-dump オプションを使用して dmstool を実行すると、Oracle9*i*AS インスタンスからすべてのメトリックが標準出力にレポートされます。

次のコマンドにより、Oracle HTTP Server 上で取得可能なすべてのメトリックとメトリック値が表示されます。

```
% dmstool -dump
```

-dump オプションを使用した dmstool を 10 ～ 15 分の間隔などで定期的に実行して、使用している Oracle9*i*AS サーバーのパフォーマンス・データを取得し、記録を保存しておくことをお薦めします。一定の期間にわたってパフォーマンス・データを保存しておけば、パフォーマンス改善のためにシステムの動作を分析したり、問題が生じたときに役立つ場合があります。

リモートにある Oracle9*i*AS システムのメトリックを表示するための dmstool の使用方法

-a オプションを使用して dmstool を実行すると、リモートにある Oracle9*i*AS インスタンスのすべてのメトリックがレポートされます。

次のコマンドにより、-a オプションで指定したアドレスの Oracle HTTP Server で取得可能なすべてのメトリックおよびメトリック値が表示されます。

```
% dmstool -a system1:6003 -list
```

3

Oracle HTTP Server の監視

この章では、Oracle HTTP Server のパフォーマンスを監視する方法を説明します。パフォーマンス・データを取得すると、Oracle9*i*AS のチューニング、およびパフォーマンスに問題のあるアプリケーションのチューニングとデバッグが容易になります。

この章には、次の項目が含まれています。

Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server の監視

Oracle HTTP Server は、Oracle9*i*AS サイトで中心的な役割を担う重要な部分です。動的データのほとんどすべてが Oracle HTTP Server で処理される他、多くの静的データも処理されます。Oracle HTTP Server のパフォーマンスを監視することで、Oracle9*i*AS のパフォーマンスの問題を特定して解決することが可能になります。

この項には、次の項目が含まれています。

- Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server の負荷の見積り
- Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server エラーの調査
- Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server 問題の分類

Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server の負荷の見積り

Oracle HTTP Server のパフォーマンス監視の最初のステップは、ワークロード（負荷）を見積ります。

Oracle HTTP Server の負荷を見積る場合は、次の点に注意してください。

- 新規アプリケーションの開発とテストを行う場合、品質保証およびパフォーマンス・テストで Oracle HTTP Server にかけるべき負荷の量を判断する必要があります。
- Oracle HTTP Server のパフォーマンスを監視する場合、時間帯や曜日によってサイトの負荷が軽いときと重いときがあるという、利用率の変動がよく見られる点に注意します。パフォーマンス・テストの実施やパフォーマンス・ベースラインの設定時には、時間帯や曜日による影響を必ず考慮に入れます。Oracle9*i*AS サイトを開発している場合でも、管理している場合でも、予想される負荷の範囲を常に設定し、サイトの利用率とパフォーマンスが予想した範囲内に収まっていることを確認するために必ず監視を行ってください。
- Oracle HTTP Server のパフォーマンス情報では、サイト全体のパフォーマンス概要が示されますが、Oracle9*i*AS Web Cache や他のキャッシュ機構によってリクエストが Oracle HTTP Server に到達する前に処理される場合は、キャッシュの監視も必要になります。

関連項目： 1-9 ページの「パフォーマンス管理の方法」

Oracle Enterprise Manager が提供する Oracle HTTP Server のパフォーマンス・データのカテゴリには次のものがあります。

- ステータス・メトリック
- 応答およびロード・メトリック
- モジュール・メトリック

関連項目：

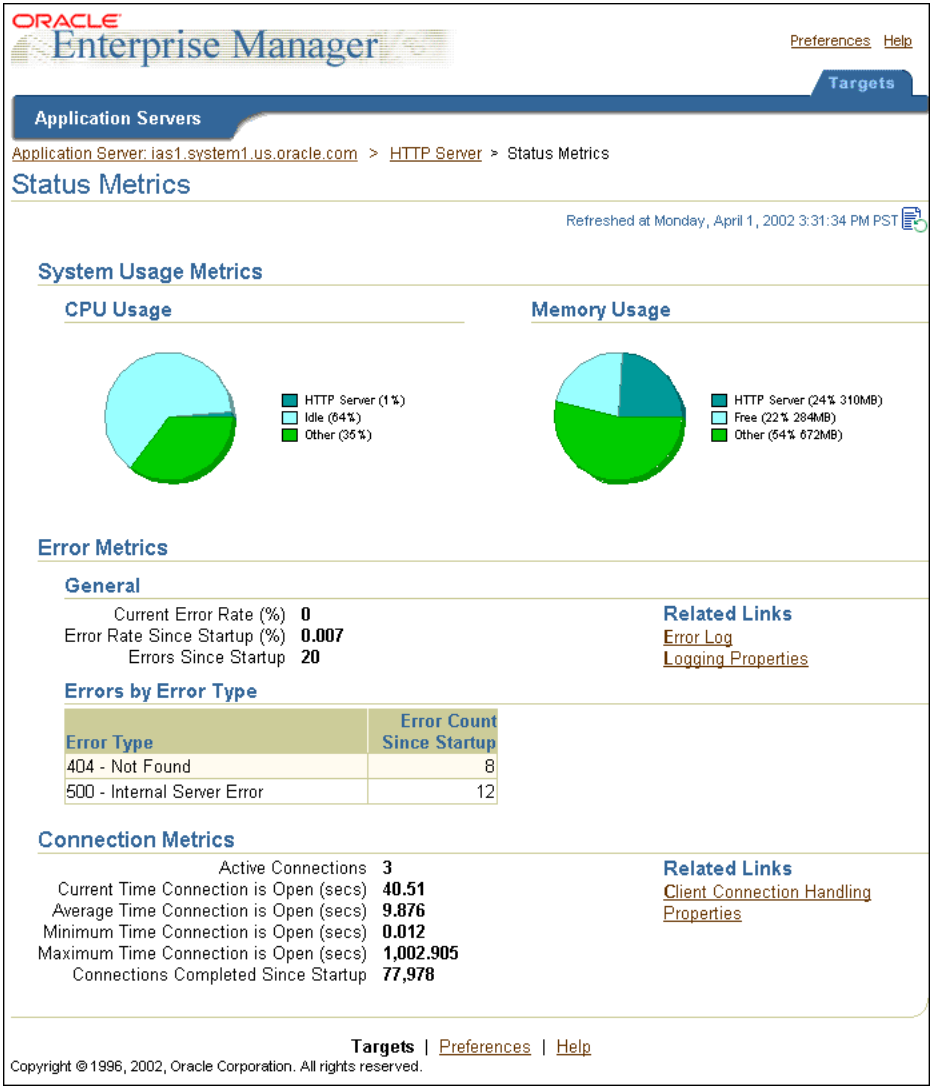
- Oracle9*i*AS における Oracle9*i*AS Web Cache の最適化に関する情報は、第 7 章「Web Cache の最適化」を参照してください。
- Oracle9*i*AS Web Cache についての詳細は、『Oracle9*i*AS Web Cache 管理および配置ガイド』を参照してください。
- Oracle9*i*AS を使用した Enterprise Manager の使用方法に関する情報は、『Oracle9*i* Application Server 管理者ガイド』を参照してください。

ステータス・メトリック

Oracle Enterprise Manager のステータス・メトリックでは、CPU 使用量、メモリー使用量、Oracle HTTP Server エラー、アクティブな接続の数に関する情報が提供されます。

図 3-1 に、Enterprise Manager の「HTTP サーバーのステータス・メトリック」ページを示します。

図 3-1 Oracle Enterprise Manager のステータス・メトリック・ページ



応答およびロード・メトリック

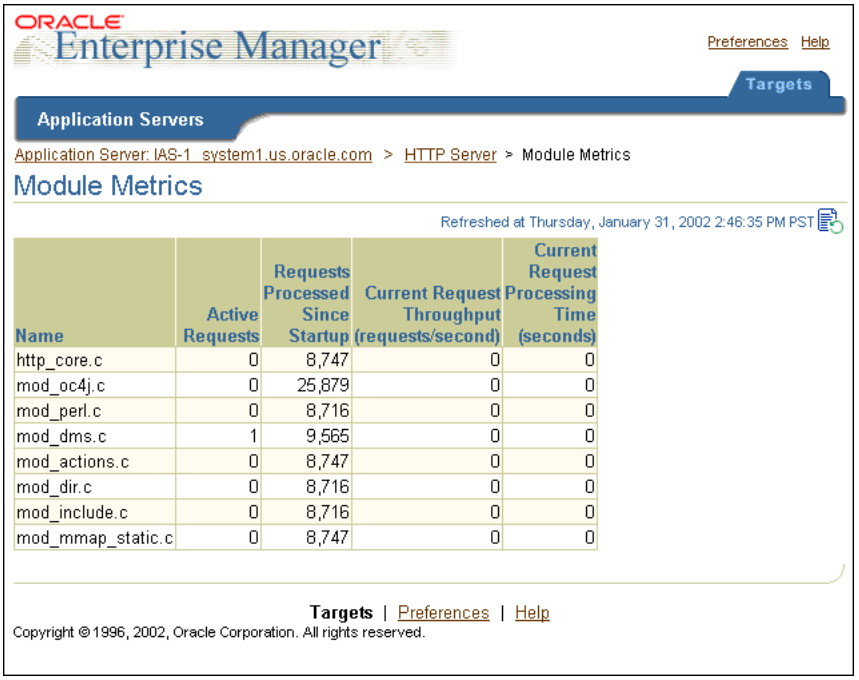
図 3-2 に Oracle Enterprise Manager の「応答およびロード・メトリック」ページを示します。このページには Oracle HTTP Server のアクティブなリクエストとリクエスト・スループットの値が表示され、リクエストの処理時間の平均、最小、最大値がレポートされます。「応答およびロード・メトリック」ページに表示される値は、システム負荷の調査に役立ちます。

図 3-2 Oracle Enterprise Manager 応答およびロード・メトリック

モジュール・メトリック

図 3-3 に Oracle Enterprise Manager の「モジュール・メトリック」ページを示します。「モジュール・メトリック」ページでは、Oracle HTTP Server のモジュールで処理されているアクティブなリクエストとリクエストの合計が表示されます。このページには起動時以降アクティブになったモジュールのみが、つまり 1 つ以上のリクエストを受け付けたモジュールが表示されます。

図 3-3 Oracle Enterprise Manager の「モジュール・メトリック」ページ



Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server エラーの調査

サイトで発生した Oracle HTTP Server のエラーは、徹底的に調査する必要があります。Oracle HTTP Server のエラーは容認できるアクティビティである場合もありますが、セキュリティ上の問題や、構成のエラー、アプリケーションのバグなどを示唆している場合もあります。エラーは、ほぼ例外なく Oracle9*i*AS のパフォーマンスに影響を与えます。エラー処理によって通常のリクエスト処理の速度が低下することがあり、逆にエラー処理によって正常なリクエスト処理に必要な処理が省略される場合には、パフォーマンスが改善されることもあります。

Oracle Enterprise Manager を使用すると、図 3-1 に示すような「HTTP ステータス」ページ上にエラー・メトリックを表示できます。エラー・メトリックには、現在のエラー率、つまり最近 5 分間の総リクエスト数に占めるエラーの発生割合や、起動時から現在までのエラー率、起動時から現在までのエラー総件数が含まれています。図 3-1 の「ステータス・メトリック」ページにある「エラー・タイプ別エラー」の表は、エラー・タイプとエラー件数を含む HTTP エラーのより詳細なリストです。この表で、各エラーは HTTP のエラー・レスポンス・タイプに基づいてカテゴリに分類されます。

図 3-1 の「エラー・タイプ別エラー」に示されているデータ値は、エラーの大半が不明な URI へのリクエスト（`404 - Not Found` エラー）によるものであることを示しています。多くの Oracle HTTP Server サイトで、Not Found エラーは比較的よく見られます。しかし、Not Found エラーの中で多数を占めるもの、たとえば全リクエスト中で 1% を越えるようなものに対しては、レポートを調査する必要があります。

レポートされた内部エラーのようなエラーをより詳細に調査するには、「関連リンク」見出しの下にある「エラー・ログ」リンクを選択してエラー・ログを調べます。エラー・ログを使用すると、特定のエラーを起こしている URI についてより詳細な情報を判断できます。

Oracle Enterprise Manager Web Site を使用した Oracle HTTP Server 問題の分類

Oracle HTTP Server でのパフォーマンス問題に気づいた場合、可能であればドリルダウンを行って問題をカテゴリに分類してください。パフォーマンス分析をより細かく行うことで、問題についてより深く理解し、問題の特定と解決に集中して作業を行うことができます。

Oracle Enterprise Manager は、パフォーマンス問題の分類に役立ちます。リクエストが処理されている場所を特定したり、リクエストの処理時間の多くが集中している箇所を特定することができます。Oracle Enterprise Manager を使用すると、パフォーマンス問題を次のように分類できます。

- Oracle HTTP Server の問題のモジュールによる分類
- Oracle HTTP Server の問題の仮想ホストによる分類
- Oracle HTTP Server の問題の子サーバーによる分類

Oracle HTTP Server の問題のモジュールによる分類

図 3-3 に、Oracle HTTP Server モジュールのモジュール・メトリックを示します（このレポートには起動後何らかのリクエストを受け取ったモジュールの情報が示されます）。モジュール・メトリックを使用すると、多数のリクエストを処理しているモジュールの名前や、個別のリクエストの処理にかかる時間が異常に長いモジュールを特定できます。モジュール・メトリック表のメトリック値を見れば、Oracle9*i*AS のパフォーマンスをモジュール別に分類できます。

モジュール・メトリックを調べる場合には、次の点に注意してください。

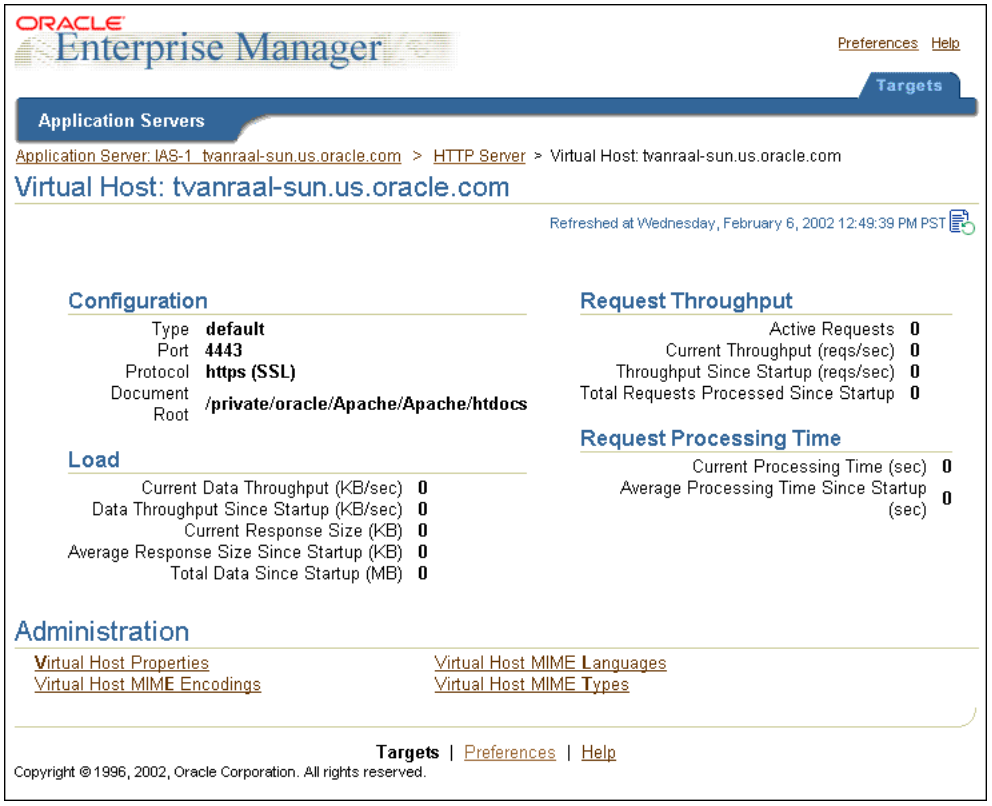
1. 静的ページのすべてのリクエストは、`http_core.c` モジュールによって処理されています。Oracle9*i*AS Web Cache が有効になっている場合は、`http_core.c` の使用は減少するはずです。Oracle9*i*AS Web Cache を使用している場合は、`http_core.c` で処理されるリクエストを監視して、Oracle HTTP Server の行う静的ページのアクティビティが Oracle9*i*AS Web Cache により減少していることを確認してください。
2. 「モジュール・メトリック」ページを調べると、リクエストの多くが `mod_oc4j.c` モジュールを通して OC4J に転送されていることを示している場合があります。転送先の OC4J について入手可能な情報を確認して、ドリルダウンする必要があります。Oracle Enterprise Manager では、OC4J インスタンスと J2EE アプリケーションのパフォーマンス測定方法が幅広く用意されています。

関連項目： 第 4 章「OC4J の監視」

Oracle HTTP Server の問題の仮想ホストによる分類

図 3-4 は、「仮想ホスト」ページが表示されているところを示しています。「仮想ホスト」ページを調べることで、仮想ホストによって処理されているリクエストの情報を入手できます。「要求スループット」、「ロード」、「要求処理時間」の情報によって、システム上で多数のリクエストを処理している仮想ホスト、または非常に多くの処理リソースを消費しているためシステムに負担を与えている可能性のある仮想ホストの識別が可能になります。この情報によって、仮想ホストごとに Oracle9*i*AS のパフォーマンス問題を分類できます。

図 3-4 Oracle Enterprise Manager の「仮想ホスト」ページ



Oracle HTTP Server の問題の子サーバーによる分類

Oracle HTTP Server の稼動中は通常、どの子サーバーがどのリクエストを処理しているのかを気にする必要はありません。使用可能な子サーバーのどれもが受信した任意のリクエストを処理できる、つまり各リクエストは空いている子サーバーによって処理されるためです。しかし、Oracle9*i*AS システムで遅延やデッドロックが生じている場合は、Oracle HTTP Server の子サーバー・プロセスの分析が必要なことがあります。「応答およびロード・メトリック」ページの「関連リンク」セクションから利用できる「プロセス詳細」ページには、Oracle HTTP Server のアクティブな各子サーバーのプロセス ID が表示されます。この情報を調べれば、リクエスト処理のデッドロックおよび非常に大きな遅延の原因となる、実行時の問題、構成のエラーまたはアプリケーションのバグを特定するために、子プロセスを監視できるようになります。このような状況では、「プロセス詳細」ページの分析によりデッドロックや遅延の発生箇所を判断できます。

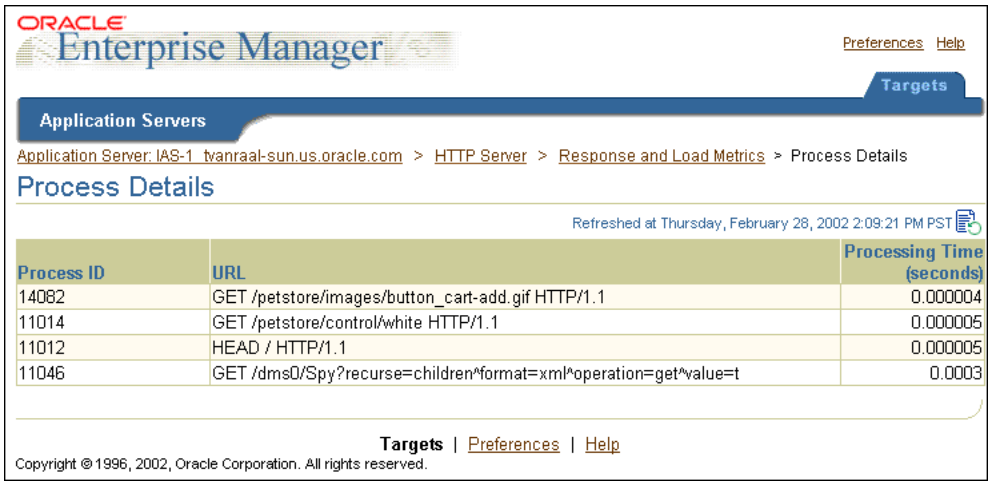
図 3-5 に、Oracle HTTP Server の子サーバー情報の「プロセス詳細」ページを示します。

Oracle HTTP Server「プロセス詳細」ページを調べる場合は、次の点に注意してください。

1. 必要であれば、プロセス ID 値を使用してデッドロックした Oracle HTTP Server の子サーバーを特定し、終了します。
2. Oracle HTTP Server は、`TimeOut` ディレクティブによって構成したタイムアウト設定の経過後、リクエストを終了します。

関連項目： `TimeOut` ディレクティブについての情報は、『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』の第 4 章、「ネットワーク接続の管理」を参照してください。

図 3-5 Oracle Enterprise Manager の HTTP サーバー「プロセス詳細」（子サーバー）ページ



組込みパフォーマンス・メトリックを使用した Oracle HTTP Server の監視

Oracle HTTP Server は、Oracle9*i*AS サイトで中心的な役割を担う重要な部分です。動的データのほとんどすべてが Oracle HTTP Server で処理される他、多くの静的データも処理されます。Oracle HTTP Server のパフォーマンスを監視することで、Oracle9*i*AS のパフォーマンスの問題を特定して解決することが可能になります。

この項には、次の項目が含まれています。

- 組込みメトリックを使用した Oracle HTTP Server の負荷の見積り
- 組込みメトリックを使用した Oracle HTTP Server エラーの調査
- 組込みメトリックを使用した Oracle HTTP Server パフォーマンス問題の分類

組込みメトリックを使用した Oracle HTTP Server の負荷の見積り

Oracle HTTP Server のパフォーマンス監視の最初のステップは、ワークロード（負荷）を見積ります。

Oracle HTTP Server のワークロードを見積る場合は、次の点に注意してください。

- 新規アプリケーションの開発とテストを行う場合、品質保証およびパフォーマンス・テストで Oracle HTTP Server にかけるべき負荷の量を判断する必要があります。
- Oracle HTTP Server のパフォーマンスを監視する場合、時間帯や曜日によってサイトの負荷が軽いときと重いときがあるという、利用率の変動がよく見られる点に注意します。パフォーマンス・テストの実施やパフォーマンス・ベースラインの設定時には、時間帯や曜日による影響を必ず考慮に入れます。Oracle9*i*AS サイトを開発している場合でも、管理している場合でも、予想される負荷の範囲を常に設定し、サイトの利用率とパフォーマンスが予想した範囲内に収まっていることを確認するために必ず監視を行ってください。
- Oracle HTTP Server のパフォーマンス・メトリックでは、サイト全体のパフォーマンス概要が示されますが、Oracle9*i*AS Web Cache や他のキャッシュ機構によってリクエストが Oracle HTTP Server に到達する前に処理される場合には、キャッシュの監視も必要になります。

関連項目： 1-9 ページの「パフォーマンス管理の方法」

Oracle HTTP Server ではパフォーマンス・メトリックが提供され、`AggreSpy` または `dmstool` を使用して表示できます。この組込みパフォーマンス・ツールを使用して `ohs_server` メトリックを表示することで、Oracle HTTP Server の負荷を見積ることができます。`AggreSpy` を使用すると、「`AggreSpy`」ウィンドウの左ペインにある `ohs_server` メトリック表を選択して、`ohs_server` メトリックを表示できます。

例 3-1 に、RAW フォーマットを使用した ohs_server メトリック表の AggreSpy 出力を示します。

例 3-1 HTTP サーバー・メトリック・レポートの全体

```
 <DMSDUMP version='2.0' timestamp='1017345371143 (Thu Mar 28 11:56:11 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
/pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    internalRedirect.count:7418 ops
    numMods.value:45
    handle.maxTime:22205524 usecs
    handle.minTime:2 usecs
    handle.avg:14274 usecs
    handle.active:2 threads
    handle.time:997159521 usecs
    handle.completed:69858
    request.maxTime:22206941 usecs
    request.minTime:602 usecs
    request.avg:31537 usecs
    request.active:1 threads
    request.time:1033442848 usecs
    request.completed:32769
    connection.maxTime:1002008298 usecs
    connection.minTime:7254 usecs
    connection.avg:258721053 usecs
    connection.active:3 threads
    connection.time:152386700540 usecs
    connection.completed:589
    childFinish.count:0 ops
    childStart.count:11 ops
    lastConfigChange.value:1017260765
    busyChildren.value:1
    readyChildren.value:10
    numChildren.value:11
    responseSize.value:903136783
    error.count:1 ops
    post.count:0 ops
    get.count:32769 ops
</statistics>
</DMSDUMP>
```

例 3-1 に示したメトリック表では、メトリックが、handle、request、connection というカテゴリでクループ化されています。各カテゴリに属する個別のメトリック名には、たとえば connection.time というように、*name.metric* という書式が使用されます。この 3 つのカテゴリに属するメトリックは、次のように説明されます。

- handle

  リクエストが HTTP サーバー・モジュールによって処理されるフェーズです。1 つのリクエストが複数の HTTP サーバー・モジュールによって処理されることもあるので注意してください。ohs_server メトリック表のトップ・レベルに表示される handle メトリックは、HTTP サーバーの全モジュールを要約したものです。

- request

  HTTP サーバー・デーモンがリクエストを受け取り、このリクエストに対するレスポンスを送信する間、つまり初めの 1 バイトを受信してから最後の 1 バイトを送信するまでのフェーズです。1 つの接続フェーズの間に、複数のリクエストがサービスを受ける場合があります。HTTP パラメータ KeepAlive が設定されていて、クライアントがこれを利用する場合は、このようなことが起こり得ます。

- connection

  HTTP 接続が確立されたときに開始し、接続がクローズされたときに終了する接続フェーズです。

現在の Oracle HTTP Server の負荷を測定するには、次の ohs_server メトリックを調べます。

- request.active
- busyChildren.value
- readyChildren.value
- numChildren.value.

これらのパフォーマンス・メトリックは、使用中の Oracle HTTP Server の子サーバーの数と、そのうちアクティブにリクエストを処理している子サーバーの数を示しています。例 3-1 に示すデータでは、11 の子サーバーが稼動中（numChildren.value）で、そのうち 1 つが現在リクエストを処理中（busyChildren.value）であることを示しています。

Oracle HTTP Server は、通常の負荷を処理するのに十分な数の子サーバーを確保する一方で、通常の負荷の変動にも対応できる必要があります。Oracle HTTP Server の子サーバーが同時に処理するリクエストの数は 1 つに限られるので、Oracle HTTP Server は同時に多数の子サーバーを実行する必要があります。デフォルト構成による処理能力では現在の負荷を処理できないことを Oracle HTTP Server が検出すると、新規の子サーバーを自動的に起動します。その後、負荷が減少すると、Oracle HTTP Server はシステム・リソースを節約するために子サーバーのいくつかを終了します。

現在の構成で、Oracle HTTP Server が頻繁に子サーバーの起動と停止を行う場合、システムのパフォーマンスが低下することがあり、システム構成の調整が必要であることを示しています。Oracle HTTP Server の子サーバーの起動が行われたか、またこれまでにいくつの子サーバーの終了が行われたかを知るには、次の ohs_server メトリックを調べます。

- childStart.count
- childFinish.count

これらのパフォーマンス・メトリックは、起動および終了した Oracle HTTP Server の子サーバーの数を示しているので、Oracle HTTP Server の負荷を示していると考えることもできます。例 3-1 で示した Oracle HTTP Server では、11 個より多くの子サーバーが起動され、終了したものはありません。

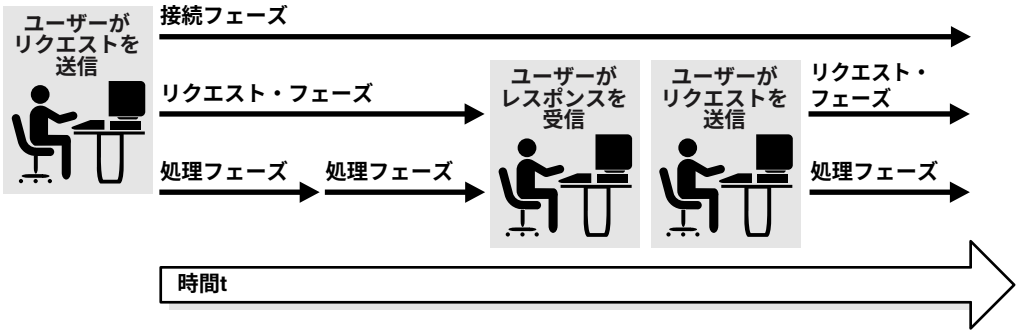
childStart.count および childFinish.count メトリック値は、ある瞬間における Oracle HTTP Server の負荷が現在の負荷を上回っていたことや、Oracle HTTP Server のデフォルトの構成パラメータで想定された範囲を超えたことを示している場合があります。起動した子サーバーおよび終了した子サーバーの数が両方とも多い場合には、次の構成パラメータの値をチューニングすることで、Oracle HTTP Server のパフォーマンスが向上する可能性があります。

- MinSpareServers
- MaxSpareServers
- StartServers

ohs_server メトリックでは、handle.avg、request.avg、connection.avg メトリック、および handle.time、request.time、connection.time の値がフェーズごとに増加します。処理フェーズの時間が最も短く、接続フェーズの時間が最も長くなります。図 3-6 に、ユーザー・リクエストを扱う場合の 3 つのフェーズの関係を示します。

KeepAlive が有効でクライアントがこれを使用する場合、図 3-6 に示すように、接続の継続時間はリクエストを実施しレスポンスを返すために必要な時間よりも大幅に長くなることがあります。これは 1 つのクライアントから複数のリクエストが送信される間、接続を維持しているからです。

図 3-6 Oracle HTTP Server の実行フェーズ



関連項目：

- 第 5 章「Oracle HTTP Server の最適化」
- Oracle9*i*AS における Oracle9*i*AS Web Cache の最適化に関する情報は、第 7 章「Web Cache の最適化」を参照してください。
- 付録 A「Oracle9iAS パフォーマンス・メトリック」
- Oracle9*i*AS Web Cache についての詳細は、『Oracle9*i*AS Web Cache 管理および配置ガイド』を参照してください。
- 子サーバーの起動と停止に関する Oracle HTTP Server の設定パラメータについての情報は、『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』を参照してください。

組込みメトリックを使用した Oracle HTTP Server エラーの調査

サイトで発生した Oracle HTTP Server のエラーは、徹底的に調査する必要があります。Oracle HTTP Server のエラーは容認できるアクティビティである場合もありますが、セキュリティ上の問題や、構成のエラー、アプリケーションのバグなどを示唆している場合もあります。エラーは、ほぼ例外なく Oracle9*i*AS のパフォーマンスに影響を与えます。エラー処理によって通常のリクエスト処理の速度が低下することがあり、逆にエラー処理によって正常なリクエスト処理に必要な処理が省略される場合には、パフォーマンスが改善されることもあります。

`dmstool` または `AggreSpy` を使用すると、`ohs_server` メトリックを調べることで Oracle HTTP Server のエラーを調査できます。例 3-1 の `ohs_server` メトリックには、エラー・アクティビティの概要が示されています。`error.count` メトリックは、Oracle HTTP Server へのリクエストの結果が HTTP エラー応答であった場合に必ず増加します。

ohs_responses メトリック表を使用して、エラー・タイプとエラー件数の詳細を調べます。この表では、すべての error.count 値が HTTP のレスポンス・タイプごとに分類されています。表では、成功した HTTP リクエストと HTTP リダイレクトの件数も集計されています。

例 3-2 に、RAW フォーマットを使用した ohs_responses メトリック表の AggreSpy レポートを示します。

例 3-2 HTTP サーバー応答メトリック（ohs_responses メトリック表）

```
<DMSDUMP version='2.0' timestamp='1017345294216 (Thu Mar 28 11:54:54 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
 /pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Responses [type=ohs_
responses]
  SvrErr_Not_Extended_510.count:        0 ops
  .
  .
  .
  CltErr_Method_Not_Allowed_405.count:  0 ops
  CltErr_Not_Found_404.count:   29 ops
  Redirect_NotModified_304.count:       23 ops
  Success_Created_201.count:    0 ops
  Success_OK_200.count: 10103 ops
  Info_Processing_102.count:    0 ops
 </statistics>
</DMSDUMP>
```

例 3-2 によると、エラーの大半が不明な URI へのリクエスト（404 - Not Found エラー）によるものであることが分かります。多くの Oracle HTTP Server サイトで、Not Found エラーは比較的よく見られます。しかし、Not Found エラーの中で多数を占めるもの、たとえば全リクエスト中で 1% を越えるようなものに対しては、レポートを調査する必要があります。

error_log および access_log ファイルを調べることにより、レポートされた内部エラー（SvrErr_InternalError_500.count）などのエラーの原因となっている URI を特定できます。

> **関連項目：** Oracle HTTP Server access_log および error_log ファイルについての情報は、『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』を参照してください。

組込みメトリックを使用した Oracle HTTP Server パフォーマンス問題の分類

Oracle HTTP Server のパフォーマンス問題に気づいた場合、可能であればドリルダウンを行って問題をカテゴリに分類してください。パフォーマンス問題についての検索を Oracle HTTP Server のサブセットに限定することで、問題についてより深く理解し、問題の特定と解決に集中することができます。組込みパフォーマンス・ツールを使用すれば、パフォーマンス問題を次の領域のどれかに分類できます。リクエストが処理されている場所を特定したり、リクエストの処理時間の多くが集中している箇所を特定できます。

この項では、パフォーマンス問題を次の各領域に分類する方法について説明します。

- Oracle HTTP Server のパフォーマンス問題のモジュールによる分類
- Oracle HTTP Server のパフォーマンス問題の仮想ホストによる分類
- Oracle HTTP Server のパフォーマンス問題の子サーバーによる分類

Oracle HTTP Server のパフォーマンス問題のモジュールによる分類

ohs_module メトリックを使用して、パフォーマンス問題を 1 つ以上のモジュールの分析に絞り込みます。モジュール・メトリックを表示すると、メトリック・データを使用してパフォーマンス問題の検索範囲を特定のモジュールに制限できます。

例 3-3 に、RAW フォーマットを使用した ohs_module メトリック表の AggreSpy 出力を示します。

例 3-3 モジュールごとの Oracle HTTP Server アクティビティ調査のためのドリルダウン

```
<DMSDUMP version='2.0' timestamp='1017345223482 (Thu Mar 28 11:53:43 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
/pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Modules [type=n/a]
     /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Modules/mod_mmap_static.c
[type=ohs_module]
      handle.maxTime:182 usecs
      handle.minTime:3 usecs
      handle.avg:5 usecs
      handle.active:0 threads
      handle.time:38942 usecs
      handle.completed:7562
      decline.count:7562 ops
        ...
       /Apache/Modules/mod_cgi.c [type=ohs_module]
        ...
        handle.avg:  199730 usecs
        ...
```

```
/Apache/Modules/mod_perl.c [type=ohs_module]
 handle.maxTime:      768041 usecs
 ...
/Apache/Modules/mod_fastcgi.c [type=ohs_module]
 ...
 handle.avg:  5866 usecs
 ...
/Apache/Modules/mod_oc4j.c [type=ohs_module]
 handle.maxTime:      33676386 usecs
 handle.minTime:      165 usecs
 handle.avg:  5488 usecs
 handle.active:       0 threads
 handle.time: 317776833 usecs
 handle.completed:    57902
 decline.count:       0 ops
 ...
/Apache/Modules/http_core.c [type=ohs_module]
 ...
 handle.completed:    93535
 ...
```

モジュール・メトリックを調べる場合は、次の点に注意してください。

1. 静的ページのすべてのリクエストは、`http_core.c` モジュールによって処理されています。Oracle9*i*AS Web Cache が有効になっている場合は、`http_core.c` の使用は減少するはずです。Oracle9*i*AS Web Cache が有効になっている場合は、`http_core.c` メトリックを監視して、静的ページのアクティビティが Oracle HTTP Server に到達しないよう Oracle9*i*AS Web Cache により効果的に防がれていることを確認してください。

2. 通常、レスポンスにはプロセスの初期化やクラスのロードなど 1 度限りの処理が必要であり、これによってリクエスト処理の平均時間のレポートに偏りが生じることがあります。パフォーマンスのレポートと分析では、総計値から最大値と最小値を除外して平均値を再計算することで、このような 1 回限りの動作による影響を排除しています。たとえば、例 3-3 に示した `mod_oc4j.c` メトリックの場合、次の数式を使用してリクエストの処理時間の平均を再計算すれば、再計算した平均値は典型的なレスポンスの処理時間として、より適切なものになります。

   ```
   new average = (time - min - max) / (completed - 2)
               = (317776833 - 165 - 33676386)/ (57902 - 2)
               = 4907 milliseconds
   ```

3. `ohs_module` メトリック表を調べると、多くのリクエストが `mod_oc4j.c` モジュールを通して OC4J に転送されていることを示している場合があります。Oracle9*i*AS では、OC4J と J2EE アプリケーションの幅広いパフォーマンス測定方法を用意しています。

   関連項目： 第 4 章「OC4J の監視」

Oracle HTTP Server のパフォーマンス問題の仮想ホストによる分類

ohs_virtualHost メトリックを使用して、パフォーマンス問題の分析を Oracle HTTP Server の仮想ホストごとに絞り込むことができます。仮想ホストを表示すると、メトリック・データを使用してパフォーマンス問題の検索範囲を Oracle HTTP Server のサブセットに限定できます。

例 3-4 に、RAW フォーマットを使用した ohs_virtualHost メトリック表の AggreSpy 出力を示します。

例 3-4 仮想ホストごとの Oracle HTTP Server アクティビティ調査のためのドリルダウン

```
<DMSDUMP version='2.0' timestamp='1017345119223 (Thu Mar 28 11:51:59 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
 /pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Virtual_Hosts [type=n/a]
     /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Virtual_
Hosts/pdsun-perf9.us.oracle.com [type=ohs_virtualHost]
      responseSize.value:       0
      request.maxTime:  0 usecs
      request.minTime:  0 usecs
      request.avg:      0 usecs
      request.active:   0 threads
      request.time:     0 usecs
      request.completed:        0
</statistics>
</DMSDUMP>
```

Oracle HTTP Server のパフォーマンス問題の子サーバーによる分類

Oracle HTTP Server の稼動中には通常、どの子サーバーがどのリクエストを処理しているのかを気にする必要はありません。使用可能な子サーバーのどれもが受信した任意のリクエストを処理できる、つまり各リクエストは空いている子サーバーによって処理されるためです。しかし、Oracle9*i*AS システムで遅延やデッドロックが生じている場合には、Oracle HTTP Server の子サーバーのメトリックを分析する必要があります。これらのメトリックを調べることで、リクエスト処理のデッドロックまたは非常に大きな遅延の原因となる、実行時の問題や構成のエラー、またはアプリケーションのバグを特定するために、子プロセスを監視できるようになります。このような状況では、Oracle HTTP Server の子サーバーのメトリックの分析によりデッドロックや遅延の発生箇所を判断することができます。

ohs_child メトリック表を使用して、パフォーマンス問題を 1 つ以上の Oracle HTTP Server の子サーバーの分析に絞り込みます。

例 3-5 に、RAW フォーマットを使用した ohs_child メトリック表の AggreSpy 出力を示します。

ohs_child メトリック表は、現在のリクエストに消費された時間順に上位 10 個の Oracle HTTP Server の子サーバーを示します。例 3-5 に示されているメトリックの場合、最上位のエントリは実行に 1 億 1700 万マイクロ秒、つまり約 2 分かかっています。ohs_child メトリックには、リスト中の各 Oracle HTTP Server の子サーバーの、リクエストに関係する URL とプロセス識別子が含まれています。

例 3-5 子サーバーごとのアクティビティ調査のためのドリルダウン

```
/Apache [type=ohs_server]
      /Apache/Children [type=n/a]
       /Apache/Children/Child00 [type=ohs_child]
        time.value:  117045690 usecs
        pid.value:   2466
        status.value:        writing
        url.value:   GET /cgi-bin/deadlock HTTP/1.1
        slot.value:  2
       /Apache/Children/Child01 [type=ohs_child]
        time.value:  5 usecs
        pid.value:   2469
        status.value:        writing
        url.value:   GET /dms0/Spy?name=/Apache/Children HTTP/1.1
        slot.value:  5
       /Apache/Children/Child02 [type=ohs_child]
        time.value:  4 usecs
        pid.value:   2465
        status.value:        keepalive
        url.value:   HEAD / HTTP/1.1
        slot.value:  1
       /Apache/Children/Child03 [type=ohs_child]
        time.value:  2 usecs
        pid.value:   7591
        status.value:        writing
        url.value:   GET /fcgi-bin/echo HTTP/1.0
        slot.value:  8
```

Oracle HTTP Server の子サーバーのメトリックを調べる場合は、次の点に注意してください。

1. 必要であれば ohs_child のメトリック値 pid.value を使用して、デッドロックした Oracle HTTP Server の子サーバーを特定し、終了します。
2. Oracle HTTP Server は、TimeOut ディレクティブによって構成したタイムアウト設定の経過後、リクエストを終了します。

関連項目： TimeOut ディレクティブについての情報は、『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』の第 4 章、「ネットワーク接続の管理」を参照してください。

4

OC4J の監視

この章では、Oracle9*i*AS Containers for J2EE（OC4J）のパフォーマンス監視方法について説明します。パフォーマンス・データを取得すると、Oracle9*i*AS のチューニング、およびパフォーマンスに問題のあるアプリケーションのチューニングとデバッグが容易になります。

この章には、次の項目が含まれています。

Oracle Enterprise Manager Web Site を使用した OC4J の監視

Oracle Enterprise Manager を使用すると、OC4J で稼動している OC4J インスタンスおよび J2EE アプリケーションのパフォーマンス特性に関する情報を表示できます。この項には、次の項目が含まれています。

- Oracle Enterprise Manager Web Site を使用した OC4J インスタンスの監視
- Oracle Enterprise Manager Web Site を使用した J2EE アプリケーションの監視

Oracle Enterprise Manager Web Site を使用した OC4J インスタンスの監視

OC4J のパフォーマンス分析を開始する前に、OC4J インスタンスが実行中であることを確認してください。図 4-1 は、Oracle Enterprise Manager に表示される選択した OC4J インスタンスのステータスで、見出し「一般」の下にある「稼動中」はこの OC4J インスタンスが稼動していることを示しています。

図 4-1 に、Oracle Enterprise Manager の OC4J インスタンスのページを示します。Oracle Enterprise Manager は、アクティブな OC4J インスタンスについての全般的な OC4J パフォーマンス・データを提供します。この全般的なパフォーマンス・データには、次のカテゴリごとに収集された OC4J パフォーマンス・データが含まれています。

- 一般
- ステータス
- OC4J の監視
- 応答 - EJB
- JDBC 使用

図 4-1 Oracle Enterprise Manager の OC4J インスタンスの表示

一般

Oracle Enterprise Manager の OC4J「一般」情報では、OC4J インスタンスの稼動および停止ステータス、起動時刻、OC4J インスタンスが稼動している仮想マシンについての情報が表示されます。この部分には、OC4J インスタンスの「停止」および「再起動」ボタンも表示されます。

ステータス

Oracle Enterprise Manager の OC4J 「ステータス」情報には、OC4J インスタンスの CPU 使用量、メモリー使用量、ヒープ使用量が表示されます。

応答 - サーブレットおよび JSP

Oracle Enterprise Manager の OC4J「応答 - サーブレットおよび JSP」情報では、アクティブなセッションの数、アクティブなリクエストの数、リクエスト処理時間の平均、アクティブなリクエストに対する秒あたりのリクエスト処理数が表示されます。

応答 - EJB

Oracle Enterprise Manager の OC4J「応答 - EJB」情報では、アクティブな EJB メソッドの数と EJB メソッドの実行率が表示されます。

JDBC 使用

Oracle Enterprise Manager の OC4J「JDBC 使用」情報では、OC4J インスタンスで開いている JDBC 接続数、JDBC 接続数の合計、アクティブなトランザクション数、コミット・トランザクションおよびロールバック・トランザクションの総数が表示されます。

Oracle Enterprise Manager Web Site を使用した J2EE アプリケーションの監視

J2EE アプリケーションを含む OC4J インスタンスが実行中であることを確認した後、アプリケーションのステータスをチェックします。J2EE アプリケーションがロードされていない場合は、デプロイしてからアプリケーションにアクセスし、正常に動作していることを確認します。

Oracle9*i*AS にデプロイした各 J2EE アプリケーションについて、カテゴリ別にパフォーマンス情報を表示できます。

図 4-2 は、Oracle Enterprise Manager に表示されている petstore アプリケーションのサンプルです。

図 4-2 Oracle Enterprise Manager の J2EE アプリケーション・メトリック

図 4-2 では、Oracle Enterprise Manager で取得可能な J2EE アプリケーション・レベルのパフォーマンス・データが、次のカテゴリ別に収集されています。

- 一般
- 応答 - サーブレットおよび JSP
- 応答 - EJB
- Web モジュール表
- EJB モジュール表

一般

Oracle Enterprise Manager の J2EE アプリケーション「一般」情報では、アプリケーションがロードされているかどうかが「ステータス」フィールドに表示され、「自動開始」のステータスが、TRUE または FALSE で表示されます。さらに、「親アプリケーション」フィールドには、親アプリケーションへのリンクが表示されます。

この部分には、アプリケーションの「Redeploy」および「Undeploy」ボタンも表示されます。

応答 - サーブレットおよび JSP

Oracle Enterprise Manager の J2EE アプリケーション「応答 - サーブレットおよび JSP」情報では、アクティブなセッションの数、アクティブなリクエストの数、リクエスト処理時間の平均、アクティブなリクエストに対する秒あたりのアプリケーションのリクエスト処理数が表示されます。

これらの情報に関する詳細を参照したり、特定のサーブレットや JSP のドリルダウンを行うには、「Web モジュール」表にあるリンクを使用してください。

応答 - EJB

Oracle Enterprise Manager の J2EE アプリケーション「応答 - EJB」情報では、アクティブな EJB メソッドの数と EJB メソッドの実行率が表示されます。

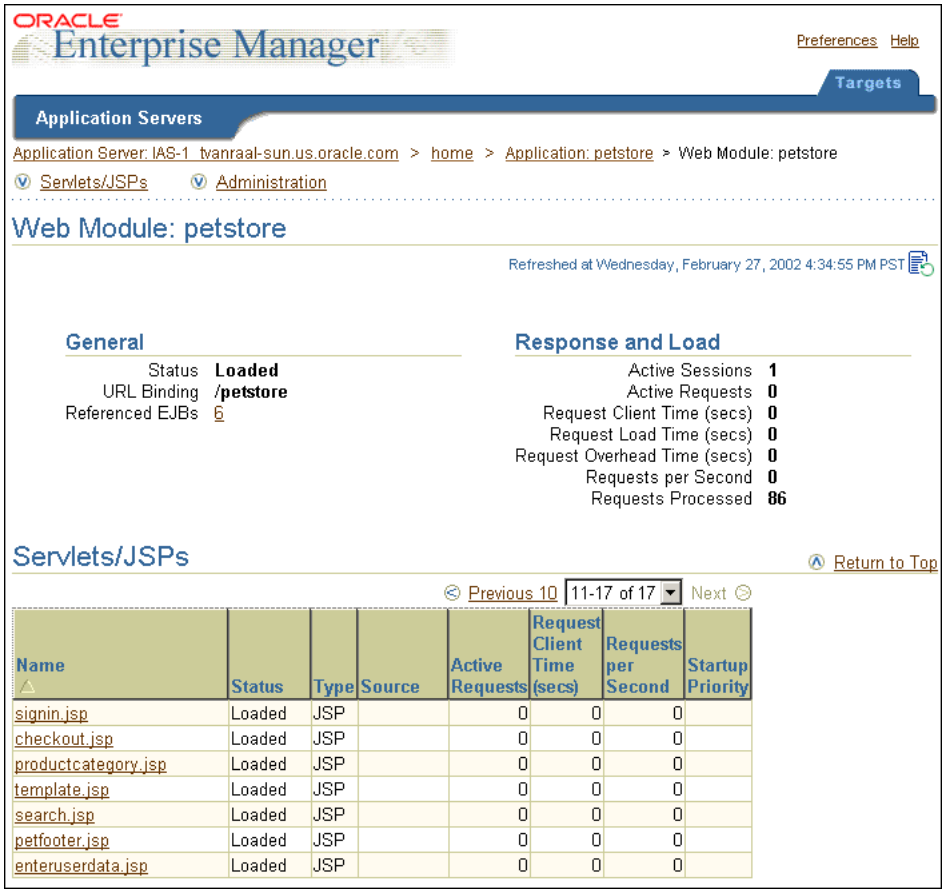
これらの情報に関する詳細を参照したり、特定のサーブレットや JSP のドリルダウンを行うには、「EJB モジュール」表にあるリンクを使用してください。

Web モジュール表

「Web モジュール」表では、J2EE アプリケーション内にあるサーブレットや JSP についての詳細情報を取得できます。

図 4-3 は、petstore アプリケーションの Web モジュールの詳細ページで、「一般」情報、「応答とロード」情報、およびアプリケーションの一部である各サーブレットと JSP のデータ値を示す表が含まれています。

図 4-3 Oracle Enterprise Manager の J2EE アプリケーションの Web モジュール・メトリック

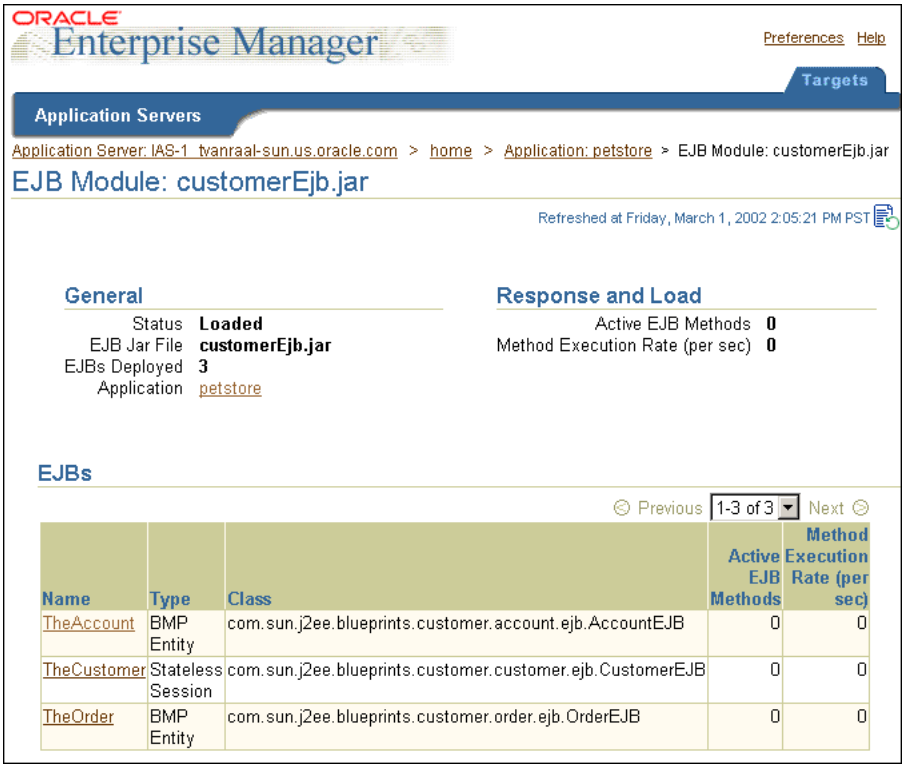


EJB モジュール表

EJB モジュール表では、EJB モジュールおよび J2EE アプリケーション内の EJB に関するより詳細な情報を取得できます。

図 4-4 に、「EJB モジュール」ページのサンプルを示します。

図 4-4 Oracle Enterprise Manager の EJB モジュール・ページ



組込みパフォーマンス・メトリックを使用した OC4J の監視

OC4J および J2EE アプリケーションのパフォーマンス分析に、Oracle9*i*AS の組込みパフォーマンス・メトリックを使用できます。OC4J のパフォーマンスを監視する前に、Oracle9*i*AS とともにデフォルトでインストールされる、OC4J ホーム・インスタンスが実行中であることを、次の URL にアクセスして確認します。

```
http://myhost:port/j2ee/
```

myhost の値は、OC4J がインストールされているホストに置き換えます。さらに、*port* には、Oracle HTTP Server の `httpd.conf` ファイルで指定されている、Oracle HTTP Server のリスニング・ポート番号を指定してください。

URL の末尾がスラッシュ（/）で終わっていることを確認してください。スラッシュがないと、ページをシステム上で見つけることができません。Web サイトがデフォルトの位置である `/j2ee/` 以外にマップされている場合、システムで構成されている位置を指定する必要があります。

OC4J がデフォルト設定で実行中の場合、この URL にアクセスすると Oracle9*i*AS Containers for J2EE の「ようこそ」ページが表示されます。

OC4J の「ようこそ」ページから、JSP やサーブレットのサンプルにアクセスできます。OC4J インスタンス上でリクエストを生成するアクティブな J2EE アプリケーションがない場合、ブラウザを使用してサンプルのサーブレットや JSP が生成した Web ページにリクエストを送信できます。

たとえば、次の URL を使用します。

```
http://myhost:myport/j2ee/servlet/SnoopServlet
http://myhost:myport/j2ee/servlet/HelloWorldServlet
```

次に、`AggreSpy` または `dmstool` を使用して、組込みパフォーマンス・メトリックを表示します。

たとえば、`AggreSpy` を使用するには、Web ブラウザに次の URL を入力します。

```
http://myhost:myport/dmsoc4j/AggreSpy
```

`AggreSpy` が出力する結果表示では、OC4J および Oracle9*i*AS コンポーネントのパフォーマンス・メトリックを表示するために選択可能なメトリック表のリストが左側のペインに表示されます。あるいは、コマンドラインまたはスクリプト内で `dmstool` を使用してパフォーマンス・メトリックを表示することもできます。

OC4J の組込みメトリックを監視する場合は、次の点に注意してください。

- 作成した各アプリケーションのサーブレット、JSP、EJP およびその他のコンポーネントの使用件数とサービス時間をそれぞれ監視し、収集したメトリックを設計およびデプロイの想定値と比較することをお薦めします。これらの想定値は、本番でのマルチユーザーのワークロードをシミュレートした状態で、単一のブラウザ・クライアントを使用してテストする必要があります。

- パフォーマンス低下のトラブルシューティングを行う場合、AggreSpy のメトリック表、または dmstool の収集するメトリックを使用して、最も頻繁に使用されているサーブレット、JSP、EJB、EJB メソッドを特定できます。頻繁に使用されているアプリケーション・コンポーネントは、システム・リソースの消費の主な原因である場合が多いので、トラブルシューティングの作業は、まず最も頻繁に使用されているコンポーネントに集中して行います。
- OC4J インスタンスの処理における JVM パフォーマンスの全体的な分析を行うには、JVM メトリック表を選択します。JVM メトリック表には、スレッドとヒープ・メモリーの割当てに関する有用な情報が表示されます。これらの値をチェックして、JVM リソースが予想の範囲内で使用されていることを確認します。

関連項目：

- 2-5 ページの「AggreSpy を使用したパフォーマンス・メトリックの表示」
- 2-8 ページの「dmstool を使用したパフォーマンス・メトリックの表示」
- 第 6 章「OC4J での J2EE アプリケーションの最適化」
- 組込みパフォーマンス・メトリックについての説明は、付録 A「Oracle9iAS パフォーマンス・メトリック」を参照してください。

5

Oracle HTTP Server の最適化

この章では、Oracle9*i* Application Server における Oracle HTTP Server の最適化のテクニックについて説明します。

この章には、次の項目が含まれています。

- TCP チューニング・パラメータ（UNIX）
- ネットワークのチューニング（Windows）
- Oracle HTTP Server ディレクティブの構成
- ロギング
- Secure Sockets Layer
- Oracle HTTP Server のパフォーマンスのヒント

TCP チューニング・パラメータ（UNIX）

TCP パラメータを正しくチューニングすると、パフォーマンスを大幅に改善できます。この項では、TCP チューニングの推奨事項と、各パラメータの簡単な説明を示します。

表 5-1 に、推奨される TCP パラメータ設定および各パラメータについての参照先を示します。

表 5-1 推奨 TCP パラメータ設定値（Solaris）

パラメータ	設定	コメント
tcp_conn_hash_size	32768	5-6 ページの「TCP 接続テーブルのアクセス速度の増加」を参照してください。
tcp_conn_req_max_q	1024	5-8 ページの「ハンドシェイクのキューの長さの増加」を参照してください。
tcp_conn_req_max_q0	1024	5-8 ページの「ハンドシェイクのキューの長さの増加」を参照してください。
tcp_recv_hiwat	32768	5-9 ページの「データ転送ウィンドウ・サイズの変更」を参照してください。
tcp_slow_start_initial	2	5-8 ページの「データ転送率の変更」を参照してください。
tcp_close_wait_interval	60000	Solaris リリース 2.6 のパラメータ名。
tcp_time_wait_interval	60000	Solaris リリース 2.7 以降のパラメータ名。 5-7 ページの「接続テーブルのエントリ保有時間の指定」を参照してください。
tcp_xmit_hiwat	32768	5-9 ページの「データ転送ウィンドウ・サイズの変更」を参照してください。

表 5-2 TCP パラメータ設定値（HP-UX）

パラメータ	スコープ	デフォルト値	チューン値	コメント
tcp_time_wait_interval	ndd/dev/tcp	60,000	60,000	5-7 ページの「接続テーブルのエントリ保有時間の指定」を参照してください。
tcp_conn_req_max	ndd/dev/tcp	20	1,024	5-8 ページの「ハンドシェイクのキューの長さの増加」を参照してください。
tcp_ip_abort_interval	ndd/dev/tcp	600,000	60,000	
tcp_keepalive_interval	ndd/dev/tcp	7,20,00,000	900,000	

表 5-2 TCP パラメータ設定値（HP-UX）（続き）

パラメータ	スコープ	デフォルト値	チューン値	コメント
tcp_rexmit_interval_initial	ndd/dev/tcp	1,500	1,500	
tcp_rexmit_interval_max	ndd/dev/tcp	60,000	60,000	
tcp_rexmit_interval_min	ndd/dev/tcp	500	500	
tcp_xmit_hiwater_def	ndd/dev/tcp	32,768	32,768	5-9 ページの「データ転送ウィンドウ・サイズの変更」を参照してください。
tcp_recv_hiwater_def	ndd/dev/tcp	32,768	32,768	5-9 ページの「データ転送ウィンドウ・サイズの変更」を参照してください。

表 5-3 TCP パラメータ設定値（Tru64）

パラメータ	モジュール	デフォルト値	チューン値	コメント
tcbhashsize	sysconfig -r inet	512	16,384	5-6 ページの「TCP 接続テーブルのアクセス速度の増加」を参照してください。
tcbhashnum	sysconfig -r inet	1	16（バージョン 5.0）	
tcp_keepalive_default	sysconfig -r inet	0	1	
tcp_sendspace	sysconfig -r inet	16,384	65,535	
tcp_recvspace	sysconfig -r inet	16,384	65,535	
somaxconn	sysconfig -r socket	1,024	65,535	
sominconn	sysconfig -r socket	0	65,535	
sbcompress_threshold	sysconfig -r socket	0	600	

表 5-4 TCP パラメータ設定値（AIX）

パラメータ	モデル	デフォルト値	推奨値	コメント
rfc1323	/etc/rc.net	0	1	
sb_max	/etc/rc.net	65,536	1,31,072	
tcp_mssdflt	/etc/rc.net	512	1,024	
ipqmaxlen	/etc/rc.net	50	100	
tcp_sendspace	/etc/rc.net	16,384	65,536	
tcp_recvspace	/etc/rc.net	16,384	65,536	
xmt_que_size	/etc/rc.net	30	150	

Linux のチューニング

Linux システム 2.1.100 以降における新たなネットワーク制限

Linux では、TCP ウィンドウ・フィールドで、15 ビットのみしか使用できません。このことは、すべての値を 2 倍するか、この制限を外してカーネルを再コンパイルする必要があることを意味します。

関連項目：「コンパイル時のチューニング」

システム稼動時のチューニング

カーネル値を変更するための sysctl アプリケーションは準備されていません。カーネル値の変更には、vi などのエディタを使用してください。

デフォルトおよび最大サイズのチューニング

カーネル値を変更するには、次のファイルを編集します。

表 5-5 Linux の TCP パラメータ

ファイル名	詳細
/proc/sys/net/core/rmem_default	受信ウィンドウのデフォルト値
/proc/sys/net/core/rmem_max	受信ウィンドウの最大値
/proc/sys/net/core/wmem_default	送信ウィンドウのデフォルト値
/proc/sys/net/core/wmem_max	送信ウィンドウの最大値

この他にも、/proc/sys/net/ipv4/ で次の TCP パラメータをチューニングできます。

- tcp_timestamps
- tcp_windowscaling
- tcp_sack

TCP パラメータについての簡単な説明は、/LINUX_SOURCE_DIR/Documentation/networking/ip-sysctl.txt を参照してください。

コンパイル時のチューニング

これまでに述べたすべての TCP パラメータのデフォルト値は、Linux カーネルのソース・ディレクトリにあるヘッダー・ファイル /LINUX_SOURCE_DIR/include/linux/skbuff.h で設定されています。

デフォルト値は次のとおりです。これは稼動時に構成可能です。

```
# ifdef CONFIG_SKB_LARGE
#define SK_WMEM_MAX 65535
#define SK_RMEM_MAX 65535
# else
#define SK_WMEM_MAX 32767
#define SK_RMEM_MAX 32767
#endif
```

MAX-WINDOW 値は、Linux のカーネル・ソース・ディレクトリにある /LINUX_SOURCE_DIR/include/net/tcp.h で変更できます。

```
#define MAX_WINDOW 32767
#define MIN_WINDOW 2048
```

注意： ウィンドウ・スケーリングを使用していない場合、ウィンドウには 32767 よりも大きな値を割り当てないでください。

TCP パケット・ヘッダーのウィンドウ・フィールドに 15 ビットしか使用できないのは、MIN_WINDOW の定義によって制限されているためです。

たとえば、40KB のウィンドウを使用するには、rmem_default を 40KB に設定します。スタックはこの値が 64KB よりも小さいことを認識し、winshift とのネゴシエーションを行いません。このような確認を行うため、取得できるのは 32KB のみになります。したがって、強制的に winshift=1 とするために、rmem_default の値を 64KB よりも大きく設定する必要があります。これにより、必要な 40KB を 15 ビットのみで表現できるようになります。

チューニングした TCP スタックを使用して、2Mbit の衛星リンクを介して netperf による測定を行ったところ、最大で 1.5 ～ 1.8Mbit のスループットを得ることができました。

TCP パラメータの設定

接続テーブルの**ハッシュ**・パラメータを Solaris で設定するには、次の行を /etc/system ファイルに追加し、その後システムを再起動します。

```
set tcp:tcp_conn_hash_size=32768
```

Tru64 では、/etc/sysconfigtab ファイルにある tcbhashsize を設定します。

TCP パラメータをこれらの推奨値に変更するサンプル・スクリプト tcpset.sh が、$ORACLE_HOME/Apache/Apache/bin/ ディレクトリにあります。

注意： スクリプトの実行後にシステムが再起動された場合、デフォルトの設定が復元され、もう一度スクリプトを実行する必要があります。設定を永続的にするには、ご使用のシステムの起動ファイルに設定を入力します。

TCP 接続テーブルのアクセス速度の増加

ユーザー数が多い場合、TCP 接続テーブルの**ハッシュ**・サイズを増加する必要があります。**ハッシュ**・サイズは、接続データの格納に使用される**ハッシュ**・バケットの数です。バケットがいっぱいになると、接続が見つかるまでに時間がかかります。**ハッシュ**・サイズを増加すると接続を探す時間は減少しますが、消費メモリーが増加します。

ご使用のシステムで 1 秒当たり 100 の接続を実行するとします。tcp_close_wait_interval を 60000 に設定すると、常に TCP 接続テーブルに約 6000 のエントリが存在します。**ハッシュ**・サイズを 2048 または 4096 に増やすと、パフォーマンスが著しく改善されます。

1 秒当たり 300 の接続を処理するシステムの場合、**ハッシュ**・サイズをデフォルトの 256 から接続テーブルのエントリ数に近い数値に変更すると、往復に要する平均時間が最大で 3 から 4 秒減少します。**ハッシュ**・サイズの最大値は 262144 です。必要に応じてメモリーを必ず増加してください。

Solaris 上で tcp_conn_hash_size を設定するには、次の行を /etc/system ファイルに追加します。パラメータは、システムの再起動後に有効になります。

```
set tcp:tcp_conn_hash_size=32768
```

Tru64 では、/etc/sysconfigtab ファイルにある tcbhashsize を設定します。

接続テーブルのエントリ保有時間の指定

前の項で説明したように、接続が確立されると、その接続に関連のあるデータは TCP 接続テーブルで管理されます。ビジーなシステムでは、TCP パフォーマンスの大半（および拡張 Web サーバーのパフォーマンス）は、接続テーブル内で特定の TCP 接続へのエントリにアクセスできるスピードに依存します。そしてこのアクセス・スピードは、テーブルにあるエントリの数と、テーブルの構成方法（たとえばハッシュ・サイズなど）に左右されます。テーブル内のエントリ数は、着信リクエストの割合と、各接続の存続期間の両方によって異なります。

サーバーは各接続のクローズ後も、それ以降にクライアントから着信したパケットを識別し、正しく破棄できるよう、TCP 接続テーブルのエントリを一定期間維持します。TCP 接続テーブルのエントリが接続のクローズ後に維持される期間は、`tcp_close_wait_interval` パラメータ（Solaris 2.7 では `tcp_time_wait_interval` に変更）で制御できます。Solaris 2.x でのこのパラメータのデフォルトは、TCP の標準に準拠した 240,000ms になっています。4 分間というこのパラメータ値は、無視されるはずのパケットへの応答として送信されてきたエラー・パケットによる、インターネットの輻輳を避けるために設定されています。実際には 60,000ms で十分であり、許容範囲であると考えられます。このように設定することで、TCP 接続表のエントリ数が大幅に削減される一方、このエントリに関して余ったパケットのすべてではないにしても、大半を破棄するのに十分な期間にわたって接続が保持されることになります。そのため、次のように設定することをお薦めします。

Solaris 2.6 での設定

```
/usr/sbin/ndd -set /dev/tcp tcp_close_wait_interval 60000
```

HP-UX および Solaris 2.7 以降での設定

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

注意： インターネットのトポロジによってユーザー数が大きく変化する場合、このパラメータをより大きな値にします。TCP 接続テーブルへのアクセス時間は、`tcp_conn_hash_size` パラメータによって改善できます。

ハンドシェイクのキューの長さの増加

TCP 接続のハンドシェイク中、サーバーは、クライアントからリクエストを受信した後、応答を送信し、クライアントからの応答を待機します。クライアントがサーバーのメッセージに応答すると、ハンドシェイクは完了します。クライアントから最初のリクエストを受信すると、サーバーはリスニング・キューにエントリを作成します。クライアントがサーバーのメッセージに応答すると、完了したハンドシェイクのメッセージのキューに移動されます。リクエストは、サーバーがサービスを行うリソースを確保するまでの間、ここで待機します。

完了していないハンドシェイクのキューの最大長は、tcp_conn_req_max_q0 によって決定されます。デフォルトは 1024 です。完了したハンドシェイクのリクエストのキューの最大長は、tcp_conn_req_max_q で定義されます。デフォルトは 128 です。

ほとんどの Web サーバーではデフォルトで十分ですが、同時ユーザーが数百人以上いる場合、これらの設定では低すぎる可能性があります。その場合、キューがいっぱいであるため、接続はハンドシェイクの段階で排除されます。この問題がご使用のシステムによるものであるかどうかを判断するには、netstat -s を使用して、tcpListenDrop、tcpListenDropQ0 および tcpHalfOpenDrop の値を調べます。最初の 2 つの値がゼロではない場合、最大値を増加する必要があります。

おそらくデフォルト値で十分ですが、オラクル社では tcp_conn_req_max_q の値を 1024 に増やすことをお薦めします。これらのパラメータを設定するには、次のコマンドを使用します。

Solaris での設定

```
% /usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 1024
% /usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q0 1024
```

HP-UX での設定

```
prompt>/usr/sbin/ndd-set /dev/tcp tcp_conn_req_max 1024
```

データ転送率の変更

TCP では、インターネットの混雑したセグメントに負荷がかかり過ぎないよう、低速でデータ転送を開始します。低速で開始する場合、1 つのパケットが送信され、確認が受信された後に 2 つのパケットが送信されます。サーバーに送信される数は、確認を受信するたびに倍になり、TCP 転送ウィンドウの制限に達するまで増加します。

ところが、オペレーティング・システムの中には、接続開始時の 1 パケットの受信をすぐに確認しないものがあります。Solaris は TCP の標準に準拠しており、デフォルトで接続の開始時に 1 パケットのみを送信します。これは、接続の開始に要する時間が大幅に増大する原因となります。そのため、データ転送開始の際の初期パケット数を 2 に増やすことをお薦めします。これは、次のコマンドで設定できます。

```
% /usr/sbin/ndd -set /dev/tcp tcp_slow_start_initial 2
```

データ転送ウィンドウ・サイズの変更

データの送受信に使用する TCP 転送ウィンドウのサイズにより、確認を待たずに送信可能なデータ量が決まります。デフォルトのウィンドウ・サイズは 8192 バイトです。ご使用のシステムのメモリーに制約がある場合以外は、これらのウィンドウの値を最大値の 32768 に増やします。これにより、大きなデータの転送速度が著しく向上します。ウィンドウを大きくするには、次のコマンドを使用します。

Solaris での設定

```
% /usr/sbin/ndd -set /dev/tcp tcp_xmit_hiwat 32768
% /usr/sbin/ndd -set /dev/tcp tcp_recv_hiwat 32768
```

HP-UX での設定

```
prompt>/usr/sbin/ndd -set /dev/tcp tcp_xmit_hiwater_def 32768
prompt>/usr/sbin/ndd -set /dev/tcp tcp_recv_hiwater_def 32768
```

通常クライアントは大量のデータを受信するため、エンド・ユーザーのシステムの TCP 受信ウィンドウも同様に大きくすると効果的です。

ネットワークのチューニング（Windows）

Windows システムで Oracle HTTP Server を稼動する場合は、多くの注意点があります。

1. メモリーが十分にあることを確認します。システムのメモリー使用量は、タスクマネージャの「パフォーマンス」タブを監視することで監視できます。
2. **TPC/IP プロトコル**・スタックのみが実行されていることを確認します。他のプロトコルが実行されている場合、コントロール・パネルの「ネットワーク」ダイアログ・ボックスの「プロトコル」タブで、プロトコルの一覧を表示します。不要なプロトコルを削除するには、マウスで選択してから「削除」をクリックします。「ネットワーク」ダイアログ・ボックスを閉じると、システムを再起動を促すメッセージが表示されます。ただし、その前に手順 3 を実行することをお薦めします。
3. ネットワークの最適化スキームとして「ファイル共有のデータ スループットを最大にする」を選択します。コントロール パネルの「ネットワーク」ダイアログ・ボックスの「サービス」タブで、サーバーのプロパティを調べることができます。一覧から「サーバー」を選択して、「プロパティ」をクリックします。すると、TCP の最適化の基準を選択するダイアログ・ボックスが表示されます。デフォルトの設定は、「ファイル共有のデータ スループットを最大にする」になっています。この設定を使用することをお薦めします。設定が変更されている場合は、デフォルトの設定にリセットしてから「OK」をクリックします。次に、コントロール パネルの「ネットワーク」ダイアログ・ボックスを閉じます。この設定を変更した場合、システムを再起動を促すメッセージが表示されます。このダイアログ・ボックスや手順 1 で変更を行った場合には、システムの再起動を行ってください。

注意：「ファイル共有のデータ スループットを最大にする」または「ネットワーク アプリケーションのデータ スループットを最大にする」のどちらかのオプションが選択されている方が、これ以外のオプションが選択されている場合よりもパフォーマンスが向上します。さらに、ネットワーク・アプリケーションのスループットではなく、ファイル共有のスループットを最大にすると、負荷がある場合での応答時間が半分になります。

これまでの説明に加え、レジストリを使用して個別の TCP/IP パラメータを調整することもできます。しかし、レジストリの調整方法は複雑であるためお薦めしません。ご使用の環境に与える影響をテストする十分な時間がある場合を除いて、TCP/IP のチューニングをこの項の手順で説明した範囲に限定することをお薦めします。

Oracle HTTP Server ディレクティブの構成

アプリケーション・サーバーを構成するために、Oracle HTTP Server では `httpd.conf` のディレクティブを使用します。この構成ファイルでは、同時に処理できる HTTP リクエストの最大数、ロギングの詳細、タイムアウトを指定します。

表 5-6 に、パフォーマンスに大きな影響を与えるディレクティブをまとめます。

表 5-6 Oracle HTTP Server の構成プロパティ

ディレクティブ	説明
`MaxClients`	実行可能なサーバー・プロセスの総数、すなわち同時に接続可能なクライアントの数を制限します。この制限数に達するとクライアントはロック・アウトされるため、あまり小さな値を設定しないでください。このディレクティブは、アクセスが過度に集中したサーバーのシステムがダウンするのを防ぐために設けられています。
`MaxRequestsPerChild`	各子プロセスが終了するまでに処理可能なリクエストの数です。Apache（および Apache が使用するライブラリ）がメモリーやその他のリソースを浪費する場合に、長期間使用した結果生じる問題を避けるために、子プロセスを終了します。大部分のシステムではほとんど必要ありませんが、Solaris などのいくつかのシステムでは無視できないリークがライブラリで発生します。このようなプラットフォームの場合は、10000 程度の値を設定してください。0 を設定すると無制限になります。 この値には、各接続の初期リクエストに続く `KeepAlive` リクエストは含まれません。たとえば、子プロセスが初期リクエストと、それに続く 10 個の KeepAlive リクエストを処理した場合、この制限に関しては 1 個のリクエストとしてのみカウントされます。

表 5-6 Oracle HTTP Server の構成プロパティ（続き）

ディレクティブ	説明
MaxSpareServers MinSpareServers	サーバーのプール・サイズの規定値です。Oracle HTTP Server は、検出した負荷に動的に適応するため、必要なサーバーのプロセス数を予想する必要はありません。つまり、現在の負荷の処理に十分なサーバー・プロセス数に加えて、一時的な負荷の増加（たとえば、1 つの Netscape ブラウザから複数のリクエストが同時に行われる場合）に備えていくつかの予備サーバーを維持します。 これは、いくつのサーバーがリクエスト待ちの状態にあるかを定期的にチェックすることで実現されています。待機中のサーバー数が MinSpareServers よりも少ない場合、新規の予備サーバーを作成します。待機サーバー数が MaxSpareServers よりも多い場合は、予備サーバーのいくつかを終了させます。 ほとんどのサイトでは、次のデフォルト値のままで問題ありません。 デフォルト値 : MaxSpareServers: 10 MinSpareServers: 5
StartServers	最初に起動するサーバーの数です。適切な概算値を設定します。起動後に急激な負荷が予想される場合は、そのときに必要な子サーバーの数を基準にしてこの値を設定してください。 デフォルト値 : 5
Timeout	受信および送信がタイムアウトするまでの秒数です。 デフォルト値 : 300
KeepAlive	永続的な接続（1 回の接続について複数のリクエストを送信する接続）を許可するかどうかを設定します。解除するには Off に設定します。 デフォルト値 : On
MaxKeepAliveRequests	永続的な接続の間に許可するリクエストの最大数です。0 に設定すると無制限になります。 クライアントとのセッションが長い場合、この値を増やすことができます。 デフォルト値 : 100
KeepAliveTimeout	1 つの接続において同じクライアントからの次のリクエストを待機する秒数です。 デフォルト値 : 15 秒

MaxClients ディレクティブの設定

MaxClients ディレクティブは、Web サーバーに同時に接続できるクライアント数、ひいては httpd プロセスの数を制限します。このパラメータは、httpd.conf ファイルで設定でき、最大値は 8000 です。MaxClients 設定が低すぎ、限界に達すると、クライアントは接続できません。

以前のリリースで、2 プロセッサの Sun Ultra SPARC 168MHz および 100Mbps のネットワークという条件で、静的ページのリクエスト（平均サイズ 20KB）をテストしたところ、次の結果が得られました。

- MaxClients のデフォルト設定 150 では、ネットワークが飽和に達した。
- 300 ユーザーの同時サポートには、約 60 個の httpd プロセスが必要だった（思考時間なし）。

前述のシステム、さらに 4 および 6 プロセッサの 336MHz システムでは、MaxClients 設定を 150 から 256 に増加しても、顕著なパフォーマンスの向上はみられませんでした。これは、最高 1000 ユーザーで静的ページとサーブレットを使用したテストの結果です。

システム・リソースが飽和状態のときに MaxClients を増加しても、パフォーマンスは改善されません。使用可能な httpd プロセスがない場合、接続リクエストはプロセスが使用可能になるまで TCP/IP システムのキューに入れられ、しばらくするとクライアントにより接続が終了されます。

永続的な接続を使用する場合、さらに多くの同時 httpd サーバー・プロセスが必要な場合があります。

動的リクエストの場合、システムの負荷が大きければ、リクエストをネットワークでキューに入れる方がよいことがあります（これにより、システムの負荷が適度な状態になります）。システム管理者は、長い応答時間よりも、タイムアウト・エラーと再試行の方が許容できるのではないか、という点を検討してみてください。この場合、MaxClients 設定を小さくし、サーバー上の同時リクエスト数のスロットルの役割を担うようにすることが可能です。

永続的な接続による httpd プロセスの可用性の低下

Oracle HTTP Server で永続的な接続を使用する場合、いくつか重大な欠点があります。特に、httpd プロセスはシングル・スレッドであるため、1 つのクライアントによりプロセスが一定の期間拘束されることがあります（期間の長さは、KeepAlive 設定によって異なります）。ユーザー数が多く、KeepAlive の制限を高くしすぎると、httpd デーモンの不足により、クライアントが拒否される可能性があります。

KeepAlive ディレクティブのデフォルトの設定は、次のとおりです。

```
KeepAlive on
MaxKeepAliveRequests 100
KeepAliveTimeOut 15
```

これらの設定により、永続的な接続の欠点を最小限に抑えながら、利点を活用できるように、接続当たりのリクエストおよびリクエスト間の時間が設定されています。ご使用のシステムでこれらの値を設定する際は、ご使用のシステムのユーザー数と作業内容を考慮する必要があります。たとえば、ユーザー数は多いが、ユーザーが送信するリクエストが小さく頻度が低い場合は、前述の設定の値を小さくするか、または KeepAlive を off に設定します。ユーザー数が少なく、サイトに頻繁にアクセスする場合は、設定値を大きくします。

ThreadsPerChild パラメータの構成（Windows）

httpd.conf にある ThreadsPerChild パラメータでは、HTTP サーバーが同時に処理できるリクエストの数を指定します。ThreadsPerChild パラメータの値を超える分のリクエストは、TCP/IP キューで待機します。TCP/IP キューでリクエストの待機を許可すると、**応答時間**と**スループット**がベストな状態になることがよくあります。

静的ページのリクエストに対する ThreadsPerChild の構成

リクエストを処理する同時スレッドの数を増やすと、サーバーが処理できるリクエストの数も増えます。しかし、高負荷の下でスレッドの数が多すぎると、リクエストの処理速度が低下し、サーバーのシステム・リソースの消費量が増加するので注意してください。

オラクル社における静的ページのリクエストのテストでは、CPU ごとの ThreadsPerChild を 20 と設定すると、良好な**応答時間**と**スループット**が得られました。たとえば、4 つの CPU を使用する場合は、ThreadsPerChild を 80 に設定します。この設定を行ったときに CPU 利用率が 85% を超えなければ、ThreadsPerChild の値を大きくすることができますが、設定したスレッド数がすべて使用されていることを確認してください。

ロギング

この項では、ロギングのタイプ、ログ・レベル、さらにロギングの使用によるパフォーマンスへの影響について説明します。

アクセス・ロギング

静的ページのリクエストの場合、デフォルト・フィールドのアクセス・ロギングにより、パフォーマンス・コストが 2 ～ 3% 増加します。

HostNameLookups ディレクティブの設定

デフォルトでは、HostNameLookups ディレクティブは Off に設定されています。サーバーにより、着信リクエストの IP アドレスがログ・ファイルに書き込まれます。HostNameLookups が On に設定されると、サーバーは各リクエストの IP アドレスに関連付けられたホスト名をインターネット上の DNS システムに問い合せ、ホスト名をログに書き込みます。

オラクル社のテストで、HostNameLookups を On に設定したところ、パフォーマンスは約 3%（最良の場合）低下しました。サーバーの負荷とご使用の DNS サーバーへのネットワーク接続状況により、DNS 検索のパフォーマンス・コストが高くなる可能性があります。ログ中にリアル・タイムでホスト名が必要な場合以外は、IP アドレスでログを行うことをお薦めします。

UNIX システムでは、$ORACLE_HOME/Apache/Apache/bin/ ディレクトリにある logresolve ユーティリティを使用して、オフラインで IP アドレスをホスト名に解決できます。

エラーのロギング

サーバーにより、エラー・ログに異常なアクティビティが記録されます。ErrorLog および LogLevel ディレクティブにより、ログ・ファイルと、記録されるメッセージの詳細レベルを指定します。デフォルトのレベルは warn です。負荷のかかったシステムにおいて、静的ページのパフォーマンスは、warn、info および debug レベルで違いはありませんでした。

Secure Sockets Layer

Oracle HTTP Server では、デフォルトでクライアントの SSL（Secure Sockets Layer）セッション情報をキャッシュします。セッション・キャッシュを使用すると、**待機時間**が長いのはサーバーへの最初の接続時のみになります。たとえば、SSL 対応のサーバーで接続と切断の簡単なテストを行ったところ、SSL セッション・キャッシュを使用しない場合、5 回の接続に要した時間は 11.4 秒でした。SSL セッション・キャッシュを使用可能にした場合、5 回の往復に要した時間は 1.9 秒でした。

`httpd.conf` の `SSLSessionCacheTimeout` ディレクティブにより、サーバーがセッションを維持する期間が決まります（デフォルトは `300` 秒です）。セッション情報はファイルに保管されます。`SSLSessionCache` ディレクティブを使用してセッション情報の保管場所を指定できます。デフォルトは、`$ORACLE_HOME/Apache/Apache/logs/` ディレクトリで、Windows システムの場合は `%ORACLE_HOME%¥Apache¥Apache¥logs¥` です。このファイルは、複数の Oracle HTTP Server プロセスで使用可能です。

SSL セッションの存続期間は、HTTP の永続的な接続の使用とは関係ありません。

Oracle HTTP Server のパフォーマンスのヒント

次のヒントを利用すると、Oracle HTTP Server（OHS）のパフォーマンスの潜在的な問題の回避やデバッグが可能になります。

- 静的リクエストと動的リクエストの比較
- Oracle HTTP Server と OC4J サーバー間の時間差の分析
- 不正確な結果の要因となる 1 つのデータに対する注意

静的リクエストと動的リクエストの比較

サーバーがどこでリソースを消費しているのかを把握すれば、チューニングに対する労力を問題の主な原因となっている箇所に集中できます。システムを構成する際は、リクエストの何パーセントが静的なもので何パーセントが動的なものかを知ることが役に立つ場合があります。静的なページは、WebCache によってキャッシュされることがあるためです。一般的に、生成のコストが余計にかかることの多い動的なページのチューニングに時間をかけることが多いようです。アプリケーションの監視とチューニングを行うことで、カタログ・データなどの動的に生成されたコンテンツの多くがキャッシュ可能であり、リソースの使用量を大幅に節減できることも明らかになります。

関連項目：

- Web Cache についての詳細は、第 7 章「Web Cache の最適化」を参照してください。
- 第 3 章「Oracle HTTP Server の監視」

Oracle HTTP Server と OC4J サーバー間の時間差の分析

状況によっては、Oracle9*i*AS Containers for J2EE（OC4J）のリクエストの平均処理時間と、ユーザーが感じる応答時間の平均に大きな開きがあるかもしれません。実際の処理に時間が費やされているのでなければ、おそらく転送に時間がかかっています。この時間差が大きいことに気付いた場合は、5-10 ページの「Oracle HTTP Server ディレクティブの構成」で説明されているパフォーマンス・ガイドラインを考慮してください。

不正確な結果の要因となる 1 つのデータに対する注意

データに外れ値があると、状態を正確に反映していない結果を得る場合があります。外れ値は、起動時に発生することがよくあります。簡単な例をシミュレートするため、PL/SQL の "Hello, World" アプリケーションを約 30 秒間実行したとします。結果を調べてみると、処理はすべて `mod_plsql.c` 内で次のように行われていました。

```
/ohs_server/ohs_module/mod_plsql.c
  handle.maxTime:     859330
  handle.minTime:      17099
  handle.avg:          19531
  handle.active:           0
  handle.time:      24023499
  handle.completed:     1230
```

ここで、`handle.maxTime` の値が `handle.avg` よりもはるかに大きい点に注意してください。これは、おそらく最初のリクエストに関する値で、データベースとの接続をオープンする必要があったためです。それ以降のリクエストは、すでに確立した接続を利用できます。PL/SQL モジュールのサービス時間の平均値を、より正確な推定値にするには、次のように平均値を再計算します。

```
(time - maxTime)/(completed -1)
```

値は次のようになります。

```
(24023499 - 859330)/(1230 -1) = 18847.98
```

6

OC4J での J2EE アプリケーションの最適化

この章では、Oracle9*i* Application Server で Oracle9*i*AS Containers for J2EE（OC4J）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ガイドラインを示します。

この章には、次の項目が含まれています。

注意： この章では、OC4J およびアプリケーション構成オプションを設定するための Oracle Enterprise Manager の使用方法について説明します。また、DCM（Distributed Configuration Management）ユーティリティである `dcmctl` を使用しても構成オプションを設定できます。このユーティリティでは、Oracle Enterprise Manager のかわりにコマンドラインを使用して、Oracle9*i*AS の構成および管理タスクを実行できます。

OC4J J2EE アプリケーション・パフォーマンスのクイック・スタート

この項では、OC4J で動作する J2EE アプリケーションのチューニング方法を簡単に説明し、重要なパフォーマンス情報へのリンクを提供します。

表 6-1 は、J2EE アプリケーションのパフォーマンスに関するクイック・ガイドです。

表 6-1 J2EE アプリケーションの考慮すべきパフォーマンス分野

パフォーマンス分野	説明およびリファレンス
十分なメモリー・リソースの準備	J2EE アプリケーションの性能を向上させるためには、十分なメモリー・リソースを用意します。J2EE アプリケーションを実行する OC4J に十分なメモリーがない場合、限られたメモリーの管理に必要なオーバーヘッドによりパフォーマンスが低下します。 6-3 ページの「OC4J プロセスの JVM ヒープ・サイズの設定」を参照してください。
データベース接続のキャッシュおよび再利用	データベース接続プーリングを適切に設定すると、多くの場合、データベースにアクセスする J2EE アプリケーションのパフォーマンスに非常に良い影響を与えます。データ・ソースでは、プールされたデータベース接続の使用および設定を可能にする構成オプションを使用できます。 6-9 ページの「データ・ソースの設定 – パフォーマンス問題」を参照してください。
同時実行性の管理および接続の制限	6-36 ページの「HTTP 接続の制限」を参照してください。
ロード・バランシング	6-37 ページの「複数の OC4J プロセスの構成」を参照してください。
アプリケーションの均衡化	6-38 ページの「OC4J インスタンスでのアプリケーション均衡化」を参照してください。
データベースの監視およびチューニング	6-39 ページの「データベースの監視およびチューニング」を参照してください。

OC4J インスタンスの設定による J2EE アプリケーションのパフォーマンスの向上

OC4J の構成オプションをチューニングすると、OC4J インスタンスで動作する J2EE アプリケーションのパフォーマンスを向上させることができます。構成を変更するには、アプリケーションのパフォーマンス要件に従ってシステムのリソースを調整する必要があります。

この項では、J2EE アプリケーションのパフォーマンスに影響を与える構成の変更について説明します。次の項目が含まれます。

- OC4J プロセスの Java オプションの設定
- データ・ソースの設定 – パフォーマンス問題

関連項目：

第 3 章「Oracle HTTP Server の監視」

第 4 章「OC4J の監視」

OC4J プロセスの Java オプションの設定

Oracle9*i*AS を実行する際、`mod_oc4j` モジュールは、Oracle HTTP Server と 1 つまたは複数の OC4J インスタンスをつなぐためのコネクタの役割を果たします。OC4J インスタンス内の各 OC4J プロセスは、それぞれの JVM（Java Virtual Machine）で実行され、J2EE リクエストの解析とレスポンスの生成を行います。リクエストが Oracle HTTP Server に到達すると、`mod_oc4j` は OC4J プロセスを取得し、リクエストを選択した OC4J プロセスにルーティングします。各 OC4J インスタンス内では、すべての OC4J JVM プロセスが同じ構成を使用し、同じ Java オプションで開始します。同様に、プロセスが終了するかその他の問題が発生しないかぎり、OC4J インスタンスの一部である各 OC4J プロセスは、プロセスにデプロイされている同じ J2EE アプリケーションを使用します。

J2EE アプリケーションによっては、アプリケーションがデプロイされた OC4J を実行している JVM の Java オプションを設定することにより、アプリケーションのパフォーマンスが向上する場合があります。

OC4J プロセスの JVM ヒープ・サイズの設定

システムに十分なメモリーがあり、アプリケーションがメモリーを集中的に使用する場合、JVM ヒープ・サイズのデフォルト値を大きくすることによってアプリケーションのパフォーマンスを向上させることができます。ヒープ・サイズに必要な量は、アプリケーションおよび使用可能なメモリーの量によって異なりますが、通常の OC4J サーバー・アプリケーションでは、最低でも 128MB のヒープ・サイズを設定することをお薦めします。十分なメモリーがある場合、256MB 以上のヒープ・サイズの使用をお薦めします。

OC4J インスタンスの OC4J プロセスに割り当てられているヒープ・サイズを変更するには、6-7 ページの「Oracle Enterprise Manager を使用した OC4J JVM コマンドライン・オプションの変更」で説明されている手順に従って、次の Java オプションを指定します。

```
-Xmssizem -Xmxsizem
```

size には、Java ヒープ・サイズを MB で指定します。

アプリケーションが常に大容量のヒープ・サイズを必要とする場合、JVM の `-Xms` サイズを `-Xmx` サイズと同じに設定し、最小のヒープ・サイズを最大のヒープ・サイズと等しくすることにより、パフォーマンスを向上させることができます。

たとえば、ヒープ・サイズを 128MB に設定するには、次のように指定します。

```
-Xms128m -Xmx128m
```

システム上で動作するすべての JVM が消費する全メモリー量が、システムのメモリー容量を超えないように最大の Java ヒープ・サイズを設定します。ハードウェア構成に対して大きすぎる Java ヒープ・サイズの値を設定すると、OC4J インスタンス内の 1 つまたは複数の OC4J プロセスが起動せず、Oracle Enterprise Manager がエラーをレポートする場合があります。エラー・レポートを確認するには、`$ORACLE_HOME/opmn/logs` ディレクトリ内の OC4J インスタントのログ・ファイルを開きます。

```
Could not reserve enough space for object heap
Error occurred during initialization of VM
```

JVM ヒープ・サイズに小さすぎる値を設定すると、OC4J プロセスをまったく開始できず、Oracle Enterprise Manager はエラーをレポートします。`$ORACLE_HOME/opmn/logs` ディレクトリ内の OC4J インスタンスのログ・ファイルを確認すると、次のようなエラーが表示されます。

```
java.lang.OutOfMemoryError
```

注意： 別の理由により、`java.lang.OutOfMemoryError` エラーが起こる場合もあります。たとえば、アプリケーションにメモリー・リークがある場合です。

システム・メモリーが不足していると、OC4J は停止します。これは、オブジェクトへのリファレンスがリリースされていないために起こります。たとえば、ハッシュ表またはベクトルにオブジェクトが保存されていて、削除されていない場合です。

これは、プロセスが大量のメモリーを必要とする原因になり得ます。この場合、頻繁なガベージ・コレクションを避けるために、プロセスの最大ヒープ・サイズを大きくします。

パフォーマンスを最大にするには、アプリケーション要件を満たす最大ヒープ・サイズを設定します。必要な Java ヒープ・サイズを判断するには、`java.lang` パッケージの `Runtime.getRuntime().totalMemory()` および `Runtime.getRuntime().freeMemory` メソッドのプログラムにコールを挿入します。合計メモリーからメモリーの空き容量を引きます。その差がアプリケーションが消費したヒープの量です。

> **関連項目：** JVM ベンダーの Web サイトで、JVM オプションおよびパフォーマンス対するその影響についての詳細な情報を入手できます。

OC4J プロセスのサーバー・オプションの設定（UNIX）

J2EE アプリケーションによっては、OC4J を実行する JVM にコマンドライン・オプション `-server` を設定すると、パフォーマンスが向上する場合があります（JVM は、2 つの関連オプション `-client` および `-server` で設定される 2 つのモードのうちのどちらかで動作します。デフォルト値は `-client` です）。このオプションを設定するには、6-7 ページの「Oracle Enterprise Manager を使用した OC4J JVM コマンドライン・オプションの変更」で説明されている手順に従って `-server` Java オプションを指定します。

> **注意：** Windows システムの Oracle9*i* Application Server で OC4J が動作している場合は、`-server` オプションを使用しないでください。オラクル社のテストでは、Windows システムで OC4J を実行する JVM で `-server` オプションを使用してもパフォーマンスは向上せず、パフォーマンスが低下する場合があることが分かりました。

`-server` オプションでは、デフォルトのクライアント VM を使用するかわりに、サーバー VM を選択します。クライアント VM とサーバー VM はほぼ同じで、その違いは特にサーバー VM がピークの処理速度を最大化するようチューニングすることのみです。これは、長時間動作するサーバー・アプリケーションを実行するために行われます。このようなアプリケーションでは、スタート・アップ時間の高速化またはランタイムのメモリー・フットプリントの低減よりも、高速のオペレーション・スピードが重視されます。デフォルトで `-server` オプションを使用しないクライアント VM は、サーバー VM よりも素早く起動し、メモリー・フットプリントも少なくてすみます。

> **注意：** `-server` オプションは、他のすべての Java オプションより前に指定する必要があります。

OC4J プロセスのスタック・サイズ・オプションの設定

J2EE アプリケーションによっては、OC4J を実行する JVM の -Xss コマンドライン・オプションの設定を変更すると、パフォーマンスが向上する場合があります。このオプションを設定するには、6-7 ページの「Oracle Enterprise Manager を使用した OC4J JVM コマンドライン・オプションの変更」で説明されている手順に従って -Xss Java オプションを指定します。

このオプションでは、スレッド内にある C コードの最大スタック・サイズを *n* に設定します。Java に渡されるプログラムの実行時に生成された各スレッドは、C コードのスタック・サイズとして *n* を持っています。デフォルトの C コードのスタック・サイズは 512KB（-Xss512k）です。各スレッドに許容される、C コードのスタック・スペースの最小値は 64KB です。

オラクル社では、J2EE アプリケーションのパフォーマンス向上のために、次の値を使用することをお薦めします。

```
-Xss128k
```

OC4J プロセスの Concurrentio オプションの設定

J2EE アプリケーションおよび JDK バージョンによっては、OC4J を実行する JVM の -Xconcurrentio コマンドライン・オプションの設定を変更すると、パフォーマンスが向上する場合があります。このオプションを設定するには、6-7 ページの「Oracle Enterprise Manager を使用した OC4J JVM コマンドライン・オプションの変更」で説明されている手順に従って -Xconcurrentio Java オプションを指定します。

このオプションは、多くのスレッドを持つプログラムに役立ちます。-Xconcurrentio でオンになるメイン機能では、スレッド・ベースの同期化ではなく、LWP ベースの同期化が使用されます。JDK 1.3.1 を使用する特定のアプリケーションでは、このオプションによってスピードが 40% 以上向上します。詳細については、次のサイトを参照してください。

```
http://java.sun.com/docs/hotspot/threads/threads.html
```

-Xconcurrentio オプションを使用する場合、必ずこのオプションを使用しない場合のアプリケーションの結果と比較してください。テストの結果はまちまちで、オプションを使用するとスピードが向上するアプリケーションもありますが、パフォーマンスが低下するアプリケーションおよび JDK バージョンもあります。次のサイトを参照してください。

```
http://java.sun.com/docs/hotspot/PerformanceFAQ.html
```

注意： JDK 1.4 では、LWP ベースの同期化がデフォルトです。ただし、JDK 1.4 でも -Xconcurrentio を設定すると、他の内部オプションがオンになるためパフォーマンスが向上する可能性があります。

Oracle Enterprise Manager を使用した OC4J JVM コマンドライン・オプションの変更

OC4J インスタンスの Java コマンドライン・オプションを変更するには、OC4J インスタンスのホーム・ページに移動して、次の手順を実行します。

1. OC4J インスタンスを停止します。
2. 「サーバー・プロパティ」ページにドリル・ダウンします。
3. 「サーバー・プロパティ」ページの「複数 VM 構成」ヘッダーの下にある「コマンドライン・オプション」領域で、「Java オプション」を設定します。

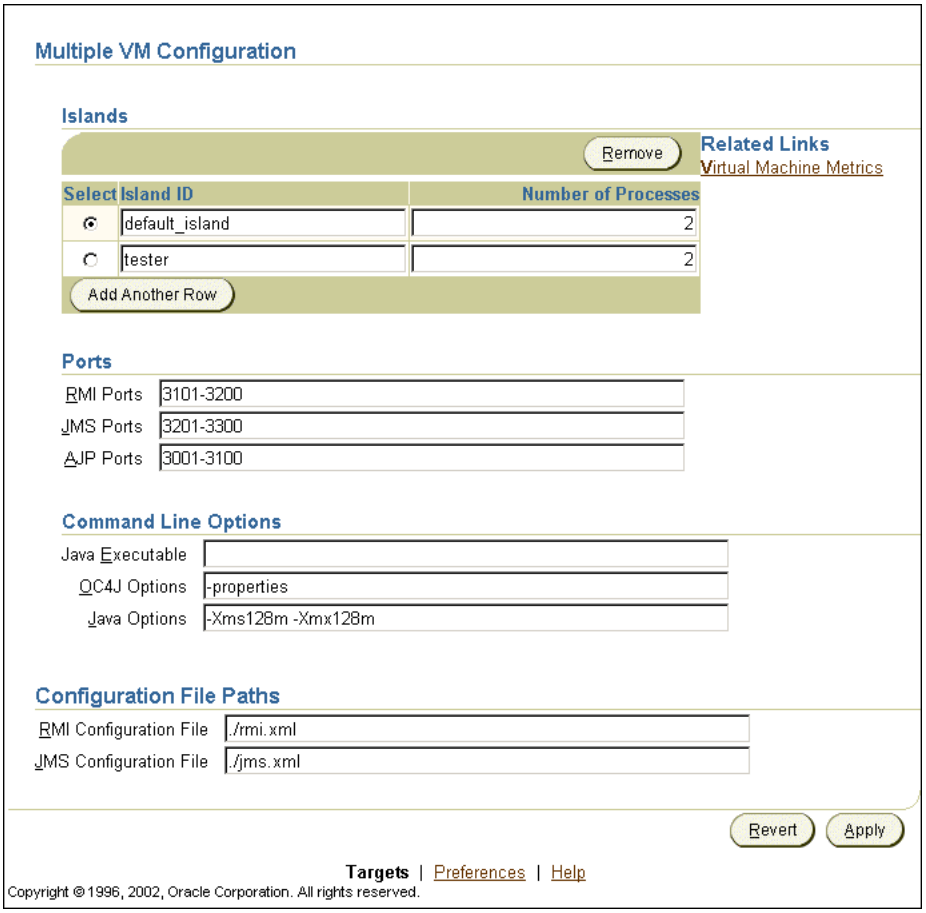
   たとえば、Java ヒープ・サイズを 128MB に設定するには、次のように入力します。

   ```
   -Xmx128m
   ```

4. 「適用」ボタンを押し、変更を適用します。
5. OC4J インスタンスを開始します。

図 6-1 は、「サーバー・プロパティ」ページとその「Java オプション」の例です。

図 6-1 Oracle Enterprise Manager を使用した OC4J インスタンスの Java ヒープ・サイズの設定



データ・ソースの設定 – パフォーマンス問題

データ・ソース（javax.sql.DataSource インタフェースを実装するオブジェクトのインスタンス化）を使用すると、データベース・サーバーへの接続を取得できます。この項では、グローバル・データ・ソースのデータ・ソース構成オプションについて説明します。グローバル・データ・ソースは、OC4J インスタンスにデプロイされているすべてのアプリケーションに対して有効です。

この項では、次の項目について説明します。

- エミュレート化および非エミュレート化データ・ソース
- エミュレート化データ・ソースで指定された EJB 対応バージョンの場所の使用
- データ・ソースでの最大オープン接続数の設定
- データ・ソースでの最小オープン接続数の設定
- キャッシュされた接続の非アクティブ・タイムアウトをデータ・ソースで設定する
- データ・ソースでの空き接続待ちタイムアウトの設定
- データ・ソースでの接続再試行間隔の設定
- データ・ソースでの最大接続試行回数の設定
- Oracle Enterprise Manager を使用したデータ・ソース構成オプションの変更

注意： サード・パーティのデータ・ソースを使用する場合、専用のプロパティを設定する必要があります。それらのプロパティについては、サード・パーティのドキュメントを参照してください。

関連項目：

- 6-28 ページの「Oracle9iAS での EJB パフォーマンスの向上」
- 『Oracle9*i*AS Containers for J2EE ユーザーズ・ガイド』の第 4 章「データ・ソース入門」
- 『Oracle9*i*AS Containers for J2EE サービス・ガイド』の第 15 章「データ・ソース」
- 『Oracle9*i*AS Containers for J2EE Enterprise JavaBeans 開発者ガイドおよびリファレンス』

エミュレート化および非エミュレート化データ・ソース

パフォーマンス関連の構成オプションの中には複数の影響を及ぼすものがあり、その影響はデータ・ソースのタイプによって異なります。OC4J では、2 種類のデータ・ソース（エミュレート化および非エミュレート化）をサポートします。

事前にインストールされているデフォルトのデータ・ソースは、エミュレート化データ・ソースです。エミュレート化データ・ソースは、Oracle データ・ソースのラッパーです。データ・ソースを使用すると、XA または JTA のグローバル・トランザクションをフル・サポートしないため、接続は非常に高速になります。ローカル・トランザクションを行う場合、または 1 つのデータベースに対してアクセスまたは更新を行う場合は、データ・ソースを使用することをお薦めします。エミュレート化データ・ソースは、Oracle データベースにも、Oracle 以外のデータベースにも使用できます。

url および connection-driver のパラメータを変更することにより、エミュレート化データ・ソースを使用して異なるデータベースへの接続を取得できます。

エミュレート化データ・ソースの定義を次に示します。

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="scott"
  password="tiger"
  url="jdbc:oracle:thin:@localhost:5521:oracle"
  inactivity-timeout="30"
/>
```

非エミュレート化データ・ソースは、Oracle データ・ソースにのみ使用できます。これらのデータ・ソースは、同じデータベース内の複数のセッションまたはグローバル・トランザクション内の複数のデータベースへの、アクセス調整を行うアプリケーションによって使用されます。

エミュレート化データ・ソースで指定された EJB 対応バージョンの場所の使用

各データ・ソースは、1 つまたは複数の論理名によって構成され、その名前により J2EE アプリケーション内のデータ・ソースを識別することができます。ejb-location は、EJB データ・ソースの論理名です。さらに、大部分の J2EE アプリケーションのデータ・ソースを識別するために、EJB を使用しない場合でも可能な限り ejb-location 名を使用します。ejb-location は、エミュレート化データ・ソースのみに適用されます。このオプションは、1 フェーズ・コミット・トランザクションまたはエミュレート化データ・ソースに使用できます。

ejb-location を使用すると、データ・ソースは接続プールのオープンを管理し、そのプールを管理します。データベースへの接続のオープンは時間のかかる処理であるため、データそのものを取得するよりも多くの時間を要する場合があります。接続プーリングによって、アプリケーションはデータベース接続が確立されるまで待機する必要がないため、クライアント・リクエストへの応答時間は短くなります。そのために、アプリケーションは接続プーリングで利用可能な接続を再利用します。

注意： データ・ソースを取得するには、エミュレート化データ・ソース定義で ejb-location JNDI 名のみを使用することをお薦めします。非エミュレート化データ・ソースには、location JNDI 名を使用する必要があります。

データ・ソースでの最大オープン接続数の設定

max-connections オプションでは、プールされたデータ・ソースに対する最大オープン接続数を指定します。システムのパフォーマンスを向上させるために、データベース・サーバーのサイズおよび構成、アプリケーションが実行する SQL のタイプなど、いくつかの要因を考慮に入れて max-connections の数を指定します。

max-connections のデフォルト値および最大数の処理は、データ・ソースのタイプ（エミュレート化または非エミュレート化）によって異なります。

エミュレート化データ・ソースに対する max-connections のデフォルト値はありませんが、接続数に影響を与えるデータベース構成の制限は適用されます。max-connections で指定した接続の最大数がすべてアクティブである場合、新しいリクエストは接続が使用可能になるまで待機する必要があります。待機の最大時間は、wait-timeout で指定します。

非エミュレート化データ・ソースには、最大接続の解析方法を定義する cacheScheme プロパティがあります。表 6-2 に、cacheScheme プロパティの値を示します（DYNAMIC_SCHEME は cacheScheme のデフォルト値）。

関連項目：

- 6-14 ページの「データ・ソースでの空き接続待ちタイムアウトの設定」
- 『Oracle9*i*AS Containers for J2EE サービス・ガイド』の第 15 章「データ・ソース」

表 6-2 非エミュレート化データ・ソースの cacheScheme 値

値	説明
FIXED_WAIT_SCHEME	この方法では、最大数に達すると、新しい接続のリクエストは他のクライアントが接続を開放するまで待機します。
FIXED_RETURN_NULL_SCHEME	この方法では、最大数の制限を越えることはできません。接続がすでに最大数に達している場合、接続リクエストは NULL を返します。
DYNAMIC_SCHEME	この方法では、新しくプールされた接続を最大数の制限を越えて作成できます。ただし、使用されていた論理接続インスタンスが終了すると、プールされた接続は自動的にクローズおよび解放され、利用可能なキャッシュに返されます。 DYNAMIC_SCHEME は cacheScheme のデフォルト値です。

max-connections の値を変更するには、次の条件を満たす必要があります。

- アプリケーションによっては、データベースへの接続数を制限するとパフォーマンスが向上します（これにより、システムはリクエストを中間層でキューに入れます）。たとえば、更新や複雑なパラレル問合せを、まとめて同じデータベース表に対して実行するアプリケーションの場合、max-connections 値を制限してデータベースの最大オープン接続数を減らすことにより、パフォーマンスが 35% 以上向上しました。

> **注意：** 少なくとも、すべての J2EE アプリケーションのデータ・ソース max-connections オプションで指定している、オープン接続の合計数が許可されるようにデータベースが設定されているかどうかを確認する必要があります。

データ・ソースでの最小オープン接続数の設定

min-connections オプションでは、プールされたデータ・ソースに対する最小オープン接続数を指定します。

データベースを使用するアプリケーションでは、データ・ソースで接続プールのオープンを管理し、そのプールを管理することによりパフォーマンスが向上します。これは、受信リクエストがデータベース接続を確立するまで待機する必要がないためで、利用可能な接続が 1 つ与えられます。これにより、接続のクローズおよび再オープンの時間を節約できます。

デフォルトでは、min-connections の値は 0 に設定されています。プールの接続を維持するために接続プーリングを使用するには、min-connections に 0 以外の値を指定します。

エミュレート化データ・ソースと非エミュレート化データ・ソースとでは、min-connections オプションは別なものとして扱われます。

エミュレート化データ・ソースでは、最初の min-connections 接続を開始する際は必要に応じて接続をオープンし、一度 min-connections の数の接続が確立されると、その数は維持されます。

非エミュレート化データ・ソースでは、データ・ソースに最初のアクセスを行った後、OC4J が min-connections で設定されている数の接続を開始し、この接続数を維持します。

開いているデータベース接続の合計数を、データベースが処理できる数に制限することは、チューニングの際の重要なポイントです。少なくとも、データベースにアクセスするすべてのアプリケーションで指定されている、すべてのデータ・ソース min-connections オプションの値の合計と同じ数のオープン接続を許可するように、データベースが設定されているかどうかを確認する必要があります。

注意： min-connections に 0 以外の値が設定されている場合、その数の接続が維持されます。接続が使用されていないときでも OC4J によって維持され、inactivity-timeout で設定されている値に達してもタイムアウトしません。

指定されている数の接続がオープンになると、OC4J を停止する以外にその接続をクローズする方法はありません。

キャッシュされた接続の非アクティブ・タイムアウトをデータ・ソースで設定する

inactivity-timeout では、未使用の接続をクローズするまでキャッシュしておく時間を秒数で指定します。

パフォーマンスを向上させるには、J2EE アプリケーションの実行中における接続の切断および再取得を行うことがないように、データ・ソースの inactivity-timeout に値を設定します。

inactivity-timeout のデフォルト値は 60 秒です。一般に、この値はある程度の間隔でアクセスが行われるアプリケーションには短すぎるため、リクエストとリクエストの間に非アクティブ状態が生じる可能性があります。パフォーマンス向上のために、ほとんどのアプリケーションで inactivity-timeout の値を 120 に設定することをお薦めします。

デフォルトの inactivity-timeout 値が低すぎるかどうかを判断するには、システムを監視します。データベース接続数が増加した後アイドル時間中に減少し、その後再び増加する場合は、inactivity-timeout または min-connections のいずれかの値を大きくします。

関連項目：

- 6-12 ページの「データ・ソースでの最小オープン接続数の設定」

データ・ソースでの空き接続待ちタイムアウトの設定

wait-timeout では、接続プールに利用可能な接続がない場合に、空き接続を待機する時間を秒数で指定します（すなわち、max-connections で指定されている接続数の制限に達していて、すべてが使用中の場合）。

アプリケーションでタイムアウト・エラーが発生する場合、wait-timeout の値を大きくすることによりエラーを回避できます。デフォルトの wait-timeout は 60 秒です。

メモリーおよび CPU などのデータベース・リソースの空きがあるにもかかわらず、オープン・データベース接続数が max-connections に近づいている場合、max-connections の設定値が低すぎる可能性があります。max-connections の値を大きくして、パフォーマンスへの影響を監視します。マシン・リソースの空きがない場合は、max-connections の値を大きくしてもパフォーマンスの向上は期待できません。

システムが飽和状態の場合、次の作業を行うことができます。

- wait-timeout の値を大きくする。
- 潜在的なパフォーマンスを向上させるために、アプリケーションの設計を評価する。
- システム・リソースを増やし、そのパラメータを調整する。

データ・ソースでの接続再試行間隔の設定

connection-retry-interval では、接続に失敗した場合に再試行するまでの秒数を指定します。

connection-retry-interval が小さい値に設定されていたり、max-connect-attempts で大きい値の接続試行数が設定されている場合、接続に失敗すると再試行が繰り返し行われ、パフォーマンスが低下する場合があります。

connection-retry-interval のデフォルト値は 1 秒です。

データ・ソースでの最大接続試行回数の設定

max-connect-attempts オプションでは、接続を再試行する最大回数を指定します。このオプションは、なんらかの理由によりネットワークや環境が不安定で、接続に失敗しやすい状況で使用すると役立ちます。

connection-retry-interval オプションが小さい値に設定されていたり、max-connect-attempts で大きい値の接続試行数が設定されている場合、接続に失敗すると再試行が繰り返し行われ、パフォーマンスが低下する場合があります。

max-connect-attempts のデフォルト値は 3 です。

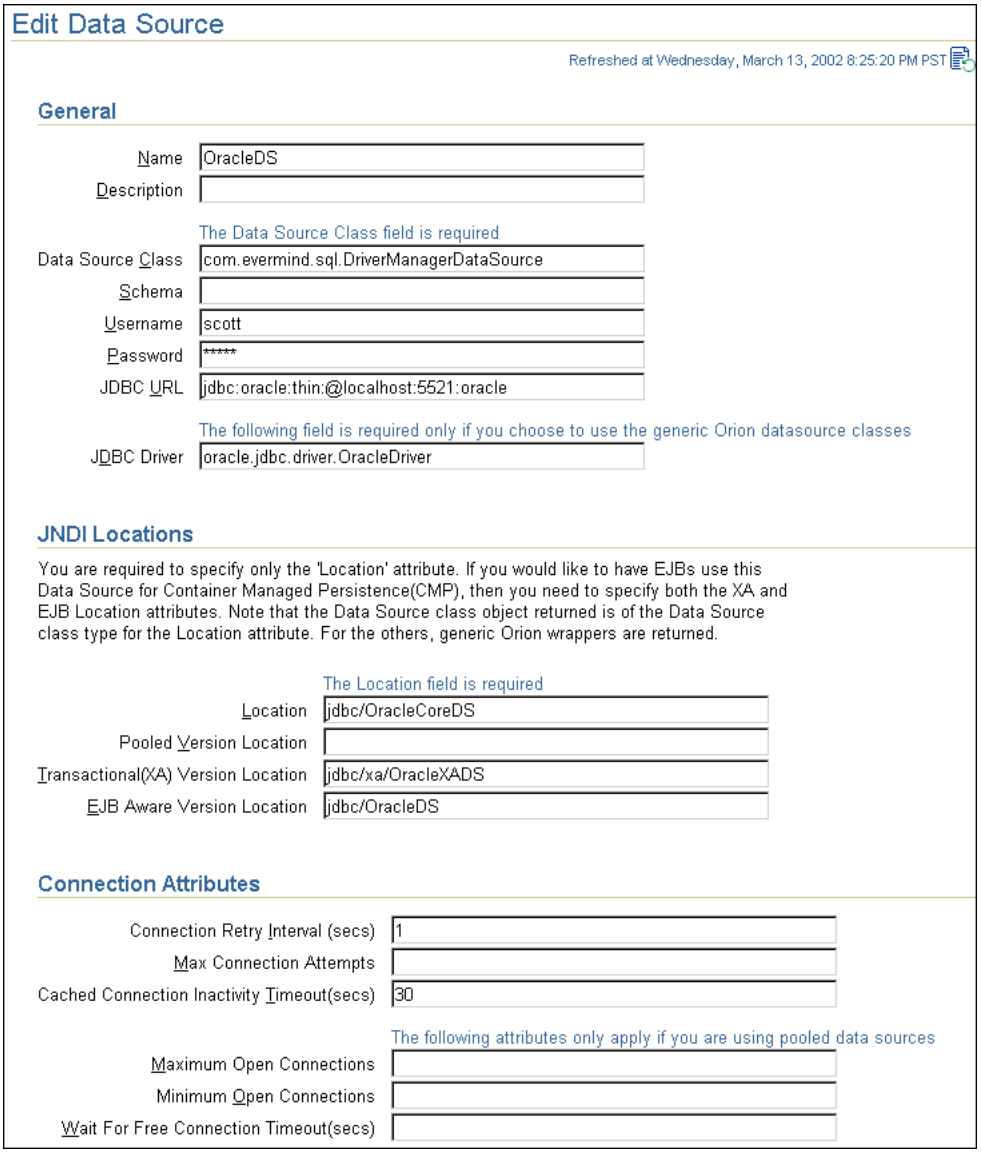
Oracle Enterprise Manager を使用したデータ・ソース構成オプションの変更

図 6-2 は、データ・ソースの表示および変更を行う Oracle Enterprise Manager 構成ページの例です。OC4J インスタンスのアプリケーション・デフォルト・ページから「データ・ソース」ページに移動し、データ・ソースを選択して「編集」ボタンをクリックすると、Oracle Enterprise Manager のこのページを利用できます。または、アプリケーションが独自のローカル・データ・ソースを所有している場合は、配布済アプリケーション説明ページの管理セクションでデータ・ソースを選択しても利用できます。

Oracle Enterprise Manager には、XML ファイルで追加または変更するデータ・ソース要素が保存されています。このファイルは、デフォルトでは `data-sources.xml` という名前で、`/j2ee/home/config` ディレクトリにあります。アプリケーションにローカル・データ・ソースを指定している場合に、このファイルの名前または場所を変更するには、デフォルト・アプリケーション画面または特定のアプリケーションのページから「一般プロパティ」ページに移動して実行できます。

注意： Oracle Enterprise Manager の「拡張プロパティ」リンクを使用して、データ・ソースを作成または編集することもできます。これにより、XML が提供されている場合に、XML 定義を使用してデータ・ソースを追加できます。

図 6-2 Oracle Enterprise Manager のデータ・ソースの構成ページ



Oracle9*i*AS でのサーブレット・パフォーマンスの向上

この項では、サーブレットの構成オプションおよびパフォーマンス・ヒントについて説明し、OC4J パフォーマンスの最適化方法を示します。

この項では、次の項目について説明します。

サーブレットの構成パラメータ変更によるパフォーマンスの向上

この項には、次の項目が含まれています。

起動時のサーブレット・クラスのロード

デフォルトでは、OC4J は最初のリクエストが行われた際にサーブレットをロードします。さらに、OC4J を使用すると、サーブレットを実行する JVM が起動した際にサーブレット・クラスをロードできます。これを行うには、`<load-on-startup>` サブ要素をアプリケーションの `web.xml` 構成ファイル内の `<servlet>` 要素に追加します。

たとえば、`<load-on-startup>` を次のように追加します。

```
<servlet>
      <servlet-name>viewsrc</servlet-name>
      <servlet-class>ViewSrc</servlet-class>
      <load-on-startup></load-on-startup>
</servlet>
```

この load-on-startup 機能を使用すると、OC4J プロセスの起動時間は長くなりますが、サーブレットの最初のリクエストの**レイテンシ**が改善されます。

Oracle Enterprise Manager を使用すると、OC4J が起動時にすべての Web モジュールをロードするよう指定できます。起動時の Web モジュールのロードを指定するには、OC4J インスタンスの「Web サイト・プロパティ」ページを選択して、「起動時にロード」チェックボックスをオンにします。

サーブレットのパフォーマンスのヒント

次のヒントを利用すると、パフォーマンスの潜在的な問題の回避やデバッグが可能になります。

- サーブレットの期間の分析
- サーバー・リクエストのロードについての理解
- ロードに長時間を要する大きなサーブレットの検出
- 未使用セッションの監視
- 異常なセッション使用の監視
- 起動時のサーブレット・セッション・セキュリティ・ルーチンのロード

サーブレットの期間の分析

J2EE エンタープライズ・アプリケーションのサーブレット（および JSP）リクエストの平均期間を知っておくと役立ちます。システムがロードされていないときのサーブレットの期間を知ることにより、システム・ロード時のパフォーマンス問題の原因を特定することが容易になります。サーブレットの平均期間は、そのサーブレットのメトリック `service.avg` でレポートされます。クラスのロードやデータベース接続の確立など、起動時のオーバーヘッドの影響を避けるため、サーブレットに多くのコールを行った後にこの値を確認します。

たとえば、`service.avg` が 32 ミリ秒であるサーブレットが存在すると仮定します。さらに、システムのロード時に応答時間が増加しますが、CPU の限界には達していないとします。`service.avg` の値を調べると、値が 32 ミリ秒に近いとします。この場合、パフォーマンスの低下がアプリケーションによるものではなく、システムまたはアプリケーションのサーバー構成によるものであるということが推測できます。反対に、`service.avg` が大幅に増加していると、アプリケーションの問題を調べる必要があります。たとえば、アプリケーションの複数のユーザーが同じリソース（データベース接続および他の要因を含む）を競合している場合がそれに当てはまります。

関連項目： 付録 A「Oracle9iAS パフォーマンス・メトリック」の表 A-11

サーバー・リクエストのロードについての理解

サーブレットおよび JSP の問題をデバッグする際、OC4J プロセスが対応しているリクエストの数を知っていると役立つ場合があります。問題がパフォーマンスに影響を与えている場合、リクエストの大量のロードがその問題をさらに悪化させてないかどうかを知る必要があります。Oracle Enterprise Manager を使用するか、アプリケーションの Web モジュール・メトリックを確認して、特定の OC4J インスタンスに対するリクエストを追跡できます。

関連項目： 付録 A「Oracle9iAS パフォーマンス・メトリック」の表 A-11

ロードに長時間を要する大きなサーブレットの検出

サーブレット・アプリケーションが、サーバーの起動後に初めて使用する際に特に遅かったり、断続的に遅くなることがあります。この場合、サーバーに大量の負荷がかかっているゆえに、応答時間が長くなっている可能性があります。ただし、アクセス・ログや CPU 使用量を定期的に監視したり、HTTP サーバーや OC4J にアクティブなリクエストを出しているユーザーの数を追跡しても、大量の負荷がかかっている形跡がない場合、単にロードに時間を要する大型のサーブレットが存在しているだけの可能性があります。

`service.maxTime`、`service.minTime`、および `service.avg` を確認することによって、ロードに時間がかかるサーブレットが存在するかどうかを知ることができます。サービスの時間よりもサーブレットをロードする時間が非常に長い場合、システム起動後に最初にサーブレットにアクセスするユーザーがその影響を受け、`service.maxTime` が大きくなります。システム起動時にサーブレットを初期化するようにシステムを構成することにより、この問題を回避できます。

関連項目： 6-17 ページの「起動時のサーブレット・クラスのロード」

未使用セッションの監視

未使用のセッションがあるかどうかを確認して、アプリケーションを定期的に監視する必要があります。セッションが無効にされていないサーブレットに、誤って書込みが行われることがよくあります。アプリケーション・ソフトウェアのソース・コードがないと、これがホストの問題の原因となり得ることに気づかないかもしれませんが、遅かれ早かれメモリーの消費量が普通よりも多いことが明らかになります。セッション・メトリック（`sessionActivation.time`、`sessionActivation.completed` および `sessionActivation.active`）によって、使用されていないセッションが存在するか、使用された後に適切に無効化されていないセッションが存在するかどうかを確認できます。

関連項目： 付録 A「Oracle9iAS パフォーマンス・メトリック」の表 A-12

異常なセッション使用の監視

この例では、セッションを作成しても 1 度も使用しないアプリケーションについて説明します。

この例を説明するにあたって、/oc4j/*application*/WEBs/*context* にある JSP のメトリックを次に示します。

```
session.Activation.active:       500 ops
session.Activation.completed:      0 ops
```

このアプリケーションは 500 のセッションを作成し、現在でもすべてアクティブな状態です。これは、アプリケーションがセッションを必要以上に作成したことを示していて、メモリーおよび CPU の消費において問題が発生するのは時間の問題です。

適切にチューニングされたアプリケーションでは、sessionActivation.active は、セッションがタイムアウトしていない sessionActivation.completed の値よりも小さい値を示します。これは、おそらくセッションが使用中およびクリーン・アップ中であることを示しています。

次に、セッションを有効に使用していて、それらを適切に無効化したと仮定します。すると、/oc4j/<application>/WEBs/<context> に、次のようなメトリックのセットが見られます。

```
session.Activation.active:         2 ops
session.Activation.completed:    500 ops
```

500 以上のセッションが作成され完了していると同時に、2 つのセッションがアクティブであるということは、セッションは使用後に無効化されているということを示しています。

起動時のサーブレット・セッション・セキュリティ・ルーチンのロード

OC4J は、セキュア・シードの生成に java.security.SecureRandom クラスを使用します。このメソッドの最初のコールには時間がかかります。システムのセキュリティ構成によっては、Oracle9*i* Application Server がセッション・ベース・サーブレットの最初のリクエストを受信するまで、このメソッドがコールされない場合があります。別な方法としては、アプリケーションの web.xml 設定ファイルでアプリケーションを起動時にロードするよう設定し、アプリケーションのクラス初期化時に SecureRandom のインスタンスを作成します。この結果、最初のリクエスト処理が短縮されるかわりに、起動時間が長くなります。

関連項目： 6-17 ページの「起動時のサーブレット・クラスのロード」

Oracle9*i*AS での JSP パフォーマンスの向上

Oracle JSP は、サン社の JavaServer Pages 仕様の Oracle による実装です。その他の機能として、Oracle データベースにアクセスするためのカスタム JavaBeans、SQL サポートおよび拡張データ型などがあります。

この項では、Oracle JSP のパフォーマンス改善方法について説明します。次の項目が含まれます。

- JSP の構成パラメータの変更によるパフォーマンスの向上
- JSP コードのチューニングによるパフォーマンスの向上

注意： JSP は実行される前に Java サーブレットに変換されるため、サーブレットのパフォーマンス問題は JSP にも適用されます。

Oracle9*i*AS では、J2EE アプリケーションのパフォーマンスを向上させる機能を持つ、JSP タグ・ライブラリが提供されます。たとえば、タグ・ライブラリで使用可能な JSP キャッシュ機能を使用して、アプリケーションのスピードおよび拡張性を向上できます。

- JESI タブ・ライブラリでは、Oracle9*i*AS Web Cache の使用がサポートされます。これは、アプリケーションの外側で行われる HTTP レベル・キャッシュをサポートし、非常に高速なキャッシュ処理を実現します。Oracle9*i*AS Web Cache は HTML、GIF、JPEG ファイルなどの静的データや、サーブレットや JSP 結果などの動的データをキャッシュできます。
- Web Object Cache タグ・ライブラリでは、JSP およびサーブレットの実行の中間結果を取得して、後でこれらのキャッシュされた結果を Java アプリケーション・ロジックの他の部分で再利用できます。

関連項目：

- 6-21 ページの「Oracle9iAS での JSP パフォーマンスの向上」
- 『Oracle9*i*AS Containers for J2EE Servlet 開発者ガイド』
- 『Oracle9*i*AS Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』

JSP の構成パラメータの変更によるパフォーマンスの向上

この項では、JSP の構成パラメータについて説明します。このパラメータを変更することにより、JSP 処理の向上および制御が可能になります。global-web-application.xml ファイルを変更することにより、このパラメータを各 OC4J インスタンスに対して設定します。

関連項目：

- JSP の構成パラメータについての詳細は、『Oracle9*i*AS Containers for J2EE JavaServer Pages サポート・リファレンス』を参照してください。
- global-web-application.xml についての詳細は、『Oracle9*i*AS Containers for J2EE Servlet 開発者ガイド』を参照してください。

main_mode パラメータの使用

main_mode パラメータでは、変更が行われた場合にクラスを自動で再ロードするか、JSP を自動で再コンパイルするかを決定します。

表 6-3 に、main_mode でサポートされている設定を示します。

表 6-3 JSP の main_mode パラメータ値

オプション	説明
justrun	ランタイム・ディスパッチャはタイムスタンプのチェックをまったく行わないため、JSP の再コンパイルや Java クラスの再ロードは行われません。これは、コードの変更がないデプロイ環境にとって最も効果的なモードです。 ソース・コードの変更がない典型的な製品デプロイ環境のように、タイムスタンプを比較する必要がない場合は、main_mode パラメータの値を justrun に設定することにより、すべてのタイムスタンプ比較、およびすべての再変換や再ロードを省略できます。 この値を使用すると、JSP アプリケーションのパフォーマンスが向上します。 **注意：** main_mode の値を justrun に設定する前に、JSP が最低 1 回はコンパイルされていることを確認してください。ブラウザから JSP を起動するか、main_mode、recompile にデフォルト値を使用してアプリケーションを起動することにより、JSP をコンパイルできます。これにより、justrun フラグを設定する前に、JSP が確実にコンパイルされます。
reload	このディスパッチャは、変換された JSP を含め、ロード後に変更されたクラスがあるかどうかをチェックします。ページおよび他の依存クラスから JavaBeans が起動します。
recompile	これは main_mode のデフォルト値です。 このディスパッチャは、JSP のタイムスタンプをチェックし、ロード後に変更があった場合はそれを再変換して、reload のすべての機能も同様に実行します。

main_mode パラメータを使用する場合は、次のことに注意してください。

- クラス・ファイルの最終変更時間にメモリー内の値を使用しているため、ファイル・システムからページ実装クラス・ファイルを削除しても、関連 JSP ソースの再変換は自動的に行われません。
- メモリー・キャッシュが失われると、ページ実装クラス・ファイルは再生成されます。これは、サーバーを再起動した後か、このアプリケーションの他のページを再変換した後に、リクエストがこのページに送信されるたびに行われます。
- 静的にインクルードされるファイルが変更されただけでは、ページは再ロードされません。静的にインクルードされるファイル（<jsp:include ... /> 構文と反対の <%@ include ... %> 構文によってインクルードされる）は、変換時にインクルードされます。

注意： main_mode の値を justrun に設定する前に、JSP が最低 1 回はコンパイルされていることを確認してください。ブラウザから JSP を起動するか、アプリケーションを実行することにより、JSP をコンパイルできます。

JSP コードのチューニングによるパフォーマンスの向上

この項では、JSP コードを変更してパフォーマンスを向上させる方法について説明します。

この項では、次の項目について説明します。

- セッション管理のパフォーマンスへの影響
- テキスト出力を行う out.print の代用としての静的テンプレート・テキストの使用
- JSP のバッファに関するパフォーマンスの問題
- 静的インクルードの使用と動的インクルードの使用

セッション管理のパフォーマンスへの影響

通常、セッションは Web アプリケーションにパフォーマンスのオーバーヘッドを与えます。各セッションは、javax.servlet.http.HttpSession クラスのインスタンスです。セッションごとのメモリーの量は、作成されたセッション・オブジェクトのサイズによって異なります。リクエストごとに新しいセッションを作成しない場合は、JSP のセッションをオフにできます。Oracle JSP セッションでは、デフォルトで有効になっています。JSP でセッションを使用する必要がない場合、JSP のトップに次の行を含めることによってセッションをオフにします。

```
<%@ page session="false" %>
```

セッションを使用する場合、セッションを明示的に取り消すようにしてください。セッションを取り消さない場合、タイム・アウトするまでアクティブのままになります。セッションを取り消すには、invalidate() メソッドを起動します。

OC4J のデフォルトのセッション・タイムアウトは 30 分です。特定のアプリケーションに対するこの値を変更するには、web.xml の <session-config> 要素にある <session-timeout> パラメータを設定します。

たとえば、次のコードは、セッションの使用後にセッションを取り消す方法を示しています。

```
HtttpSession session;
session = httpRequest.getSession(true);
.
.
.
session.invalidate();
```

OC4J は、セキュア・シードの生成に java.security.SecureRandom クラスを使用します。このメソッドの最初のコールには時間がかかります。システムのセキュリティ構成によっては、Oracle9*i* Application Server がセッション・ベース JSP の最初のリクエストを受信するまで、このメソッドがコールされない場合があります。別な方法としては、起動時にロードするアプリケーションのクラス初期化にコールを含めることによって、起動時にこのコールが強制的に実行されるようにします。この結果、最初のリクエスト処理が短縮されるかわりに、起動時間が長くなります。

注意： サーブレットはデフォルトでセッションを使用しませんが、JSP ページはデフォルトでセッションを使用します。

関連項目：

- セッションの詳細については、『Oracle9*i*AS Containers for J2EE JavaServer Pages サポート・リファレンス』を参照してください。
- セッションの詳細については、『Oracle9*i*AS Containers for J2EE Servlet 開発者ガイド』を参照してください。

テキスト出力を行う out.print の代用としての静的テンプレート・テキストの使用

JSP コード `out.print("<html>")` を使用するには、静的テンプレート・テキストを使用するよりも多くのリソースが必要です。パフォーマンスの面では、動的に生成されるテキストに対して `out.print()` コマンドの使用を予約しておくことが最もよい方法です。

例 6-1 および例 6-2 は、HTML コードの例です。これらの JSP サンプルについては、例 6-2 の方が効率的で高いパフォーマンスを発揮します。

例 6-1 out.print の使用

```
<%
  out.print("<HTML> <HEAD> <TITLE>Bookstore Home Page</TITLE></HEAD>\n");
  out.print("<BODY BGCOLOR=\"#ffffff\">\n");
  out.print("<H1 ALIGN=\"center\">Book Store Web Commerce Test</H1>\n");
  out.print("<P ALIGN=\"CENTER\">\n");
  out.print("<IMG SRC=\"../bookstore/Images/booklogo.gif\" ALIGN=\"BOTTOM\""+
              "BORDER=\"0\" WIDTH=\"288\" HEIGHT=\"67\"></P>\n");
  out.print("<H2 ALIGN=\"center\">Home Page</H2>\n");
%>
<jsp:useBean  id="randomid"  class="bookstore.BOOKS_Util" scope="request" >
<%
  random_id = randomid.getRandomI_ID();
%>
```

例 6-2 静的テキストの使用

```
<HTML> <HEAD> <TITLE>Bookstore Home Page</TITLE></HEAD>
<BODY BGCOLOR=\"#ffffff\">
<H1 ALIGN=\"center\">Bookstore Web Commerce Test </H1>
<P ALIGN=\"CENTER\">
<IMG SRC=\"../bookstore/Images/booklogo.gif\" ALIGN=\"BOTTOM\""+
          "BORDER=\"0\" WIDTH=\"288\" HEIGHT=\"67\"></P>
<H2 ALIGN=\"center\">Home Page</H2>
<jsp:useBean  id="randomid"  class="bookstore.BOOKS_Util" scope="request" >
<%
  random_id = randomid.getRandomI_ID();
%>
```

JSP のバッファに関するパフォーマンスの問題

デフォルトでは、JSP はページ・バッファと呼ばれるメモリー領域を使用します。JSP が動的グローバリゼーション、contextType 設定、またはエラー・ページなどを使用する場合、ページ・バッファ（デフォルト設定は 8KB）が必要になります。ページがこれらの機能を使用しない場合、次のコマンドを使用してバッファを無効にできます。

```
<%@ page buffer="none" %>
```

バッファの値を none に設定してバッファを無効化すると、メモリの使用が減少し、バッファのコピー処理が省略されるため、パフォーマンスが向上します。

バッファが必要な場合、バッファに適切なサイズを設定することが重要です。デフォルトの 8KB のバッファよりも大きいページを作成する場合、およびバッファ・サイズをリセットしていない場合、JSP の autoflush がアクティブ化され、パフォーマンスに影響を与える可能性があります。そのため、JSP のバッファが必要な場合、ページ・バッファに適切な値が設定されていることを確認してください。たとえば、バッファ・サイズを 24KB に設定するには、次のコマンドを使用します。

```
<%@ page buffer="24KB" %>
```

静的インクルードの使用と動的インクルードの使用

include ディレクティブは、インクルードされるページを変換時に JSP（including page）にコピーします。これは**静的インクルード**（または**変換時インクルード**）と呼ばれ、次の構文を使用します。

```
<%@ include file="/jsp/userinfopage.jsp" %>
```

別の方法として、jsp:include タグでは、インクルードするページの出力にあるインクルードされるページからの出力をランタイム時に動的にインクルードします。これは**動的インクルード**（または**実行時インクルード**）と呼ばれ、次の構文を使用します。

```
<jsp:include page="/jsp/userinfopage.jsp" flush="true" />
```

パフォーマンスの観点から見て、大きすぎない静的テキストの場合、動的インクルードよりも静的インクルードを使用するほうが有効的です。

通常、インクルードを使用する場合は次の点に注意してください。

- 静的インクルードはページ・サイズに影響します。静的インクルードは、動的インクルードが必要とするリクエスト・ディスパッチャのオーバーヘッドを回避できますが、大きなファイルに使用すると問題が発生することがあります。静的インクルードは、通常、複数の JSP で繰り返し使用されるコンテンツを含む小さいファイルをインクルードするために使用します。たとえば、次のような場合です。
  - アプリケーションの各ページの上部または下部に、ロゴや著作権情報などを静的にインクルードする場合

  - 複数ページで必要とされる宣言やディレクティブ（Java クラスのインポートなど）を含むページを静的にインクルードする場合
  - アプリケーションの各ページからの中央 `status checker` ページを静的にインクルードする場合

- 動的インクルードは、処理上のオーバーヘッドやパフォーマンスに影響を与えます。動的インクルードは、モジュール化されたプログラミングに使用すると便利です。あるときにはそのページのコンテンツを実行し、あるときには他のページの出力生成に使用するページを作成できます。動的にインクルードされるページは、インクルードするページのサイズを大きくすることなく、複数のインクルードするページで再利用できます。

注意： 静的インクルードおよび動的インクルードは、どちらも同じサーブレット・コンテキスト内のページ間でのみ使用できます。

関連項目：『Oracle9*i*AS Containers for J2EE JavaServer Pages サポート・リファレンス』

静的コンテンツをインクルードする際のパフォーマンスの問題

実行時に変化しない大量の HTML コードなど、大量の静的コンテンツを含む JSP では、変換および実行が遅くなることがあります。

この問題を回避して、パフォーマンスを向上させるための 2 つの方法があります。

- 静的 HTML を別のファイルに入力し、動的 `include` コマンド（`jsp:include`）を使用して、実行時にその出力を JSP 出力にインクルードします。

注意： 静的 `<%@ include... %>` コマンドは動作しません。これは、インクルードされるファイルを変換時にインクルードすると同時に、そのコードをインクルードするページに実質上にコピーし戻すことになります。これでは問題を解決できません。

- 静的 HTML の Java リソース・ファイルへの入力

  `external_resource` 構成パラメータを有効にしている場合、JSP トランスレータは静的 HTML を Java リソース・ファイルに入力します。

  事前変換を行う場合、`ojspc` ツールの `-extres` オプションを使用しても実行できます。

注意： 静的 HTML をリソース・ファイルに入力すると、上で説明した `jsp:include` による解決法よりも多くのメモリー・フットプリントを必要とします。これは、ページ実装クラスがクラスのロード時に常にリソース・ファイルのロードを必要とするためです。

Oracle9*i*AS での EJB パフォーマンスの向上

この項では、OC4J の EJB 処理方法を設定する構成パラメータについて説明します。このオプションをチューニングすると、OC4J で動作する EJB のパフォーマンスを向上させることができます。

この項では、次の項目について説明します。

- EJB の server.xml 構成パラメータの設定
- OC4J 固有の EJB 構成パラメータの設定

EJB の server.xml 構成パラメータの設定

この項では、OC4J インスタンスの `server.xml` ファイル内の EJB のパフォーマンスをチューニングするパラメータについて説明します。

トランザクション構成タイムアウトの設定

OC4J インスタンスの `server.xml` ファイル内にある `transaction-config` 要素で、トランザクション構成タイムアウトのデフォルト値を変更できます。`transaction-config` タイムアウトは、EJB 固有のタイムアウトではありませんが、EJB を使用するすべてのトランザクションに影響します。この構成パラメータでは、タイムアウトによってロール・バックする前のトランザクションの許容最大時間を指定します。このパラメータは、OC4J インスタンス上のすべてのトランザクションに適用されます。このパラメータのデフォルト値は、60000 ミリ秒（60 秒）です。

たとえば、次の `server.xml` では `transaction-config` タイムアウト・パラメータを 30 秒に設定しています。

```
<transaction-config timeout="30000"/>
```

トランザクションを行うアプリケーションでトランザクション・タイムアウト・エラーが発生する場合、またはトランザクションが 60 秒を超える場合（datasources.xml の wait-timeout で設定した接続待機時間を含む）は、大きい値を指定します。

transaction-config タイムアウトは、OC4J で動作するすべてのトランザクションに適用されるため、1 番長いトランザクションを考慮に入れる必要があります。transaction-config timeout は EJB レベルのすべてのトランザクションに適用されるため、transaction-config に短いタイムアウト値を指定すると、各 EJB のタイムアウト値にそれより大きな値を設定できなくなります。このように、タイムアウト値には、トランザクション内で使用するタイムアウトよりも長いか、それと等しい値を設定する必要があります（たとえば、データ・ソースの wait-timeout および EJB の call-timeout）。

関連項目：

- 6-14 ページの「データ・ソースでの空き接続待ちタイムアウトの設定」
- 6-30 ページの「すべての EJB に適用される構成パラメータ」

OC4J 固有の EJB 構成パラメータの設定

この項では、OC4J 固有の EJB パフォーマンスをチューニングするパラメータについて説明します。このパラメータは、orion-ejb-jar.xml ファイルで設定します。

この項では、次の項目について説明します。

すべての EJB に適用される構成パラメータ

表 6-4 では、OC4J 固有の EJB パフォーマンスをチューニングするパラメータについて説明しています。これらのパラメータは、Session Bean および Entity Bean を含むすべての EJB のタイプに適用されます。表 6-4 のパラメータは、`orion-ejb-jar.xml` で指定します。

表 6-4 EJB のすべてのタイプに適用される EJB パラメータ

パラメータ	説明
`call-timeout`	Session Bean および Entity Bean に適用されます。このパラメータでは、コンテナが EJB メソッドをコールする前に、EJB コンテナが必要とするすべてのリソースに対する最大待機時間（データベース接続を除く）を指定します。 デフォルト値 : Entity Bean は 90000 ミリ秒、Session Bean は 0（無制限）。 **関連項目 :** 6-28 ページの「トランザクション構成タイムアウトの設定」
`max-tx-retries`	Session Bean および Entity Bean に適用されます。このパラメータでは、システム・レベルの障害によってロール・バックされるトランザクションの再試行回数を指定します。 通常、開始時には `max-tx-retries` を 0 に設定し、再試行によって解決する見込みのあるエラーが発生した場合のみ再試行を追加することをお薦めします。たとえば、シリアル化可能な分離を使用していて、競合時にトランザクションを自動的に再試行したい場合には、再試行を使用できます。ただし、競合時に Bean が通知を受け取るようにする場合には、`max-tx-retries=0` を設定します。 デフォルト値 : 3（Session Bean および Entity Bean の両方） **関連項目 :** 6-28 ページの「トランザクション構成タイムアウトの設定」 **関連項目 :** 6-14 ページの「データ・ソースでの接続再試行間隔の設定」

CMP Entity Bean の構成パラメータ

この項では、CMP を使用する Entity Bean のパラメータについて説明します。このパラメータは、orion-ejb-jar.xml 構成ファイルで設定し、パフォーマンスに影響を与えます。

表 6-5 では、Entity Bean の CMP 固有のパラメータについて説明しています。

表 6-6 では、サポートされている locking-mode パラメータの値を説明しています。

表 6-5 CMP Entity Bean のパフォーマンス・パラメータおよび説明

パラメータ	説明
call-timeout	詳細は、表 6-4 を参照してください。
do-select-before-insert	値を false に設定して、挿入前の余分な選択（エンティティがすでに存在しているかどうかの判断）を無効にすることをお薦めします。これにより、重複がある場合挿入時に検出できます。 デフォルト値 : true
exclusive-write-access	locking-mode=optimistic または pessimistic の Bean のデフォルト値は false、locking-mode=read-only の Bean のデフォルト値は true。 このパラメータは、使用しているコミット・オプションに対応します（EJB 仕様における A、B または C）。 exclusive-write-access = true の場合のコミット・オプションは A。 ロックが即時またはコミット時の場合、およびロックが EJB クラスタリングで使用されていない場合、exclusive-write-access は強制的に false に設定されます。読取り専用ロックで exclusive-write-access が false の場合がありますが、exclusive-write-access=false であるとフィールドに変更がない場合に ejbStore はすでにスキップされているため、読取り専用ロックがパフォーマンスに影響することはありません。パフォーマンスを向上させ、読取り専用 Bean での ejbLoad の実行を回避するには、exclusive-write-access=true も設定する必要があります。
isolation	データベースがすでに分離モードで設定されていて、これをトランザクションに使用したい場合、orion-ejb-jar.xml file で分離モード属性を明示的に設定しないようにすると、パフォーマンスが向上します。分離設定を回避することにより、データベースをデフォルト設定で使用して、トランザクションで分離レベルを明確に設定するための余分なプロセスを省くことができます。 isolation オプションの説明およびロック・モードとの関連については、表 6-7 を参照してください。 デフォルト値 : optimistic
locking-mode	locking-mode パラメータで指定されるロック・モードでは、同時実行性を管理し、リソースの競合の管理のブロックおよびパラレル実行のタイミングを設定します。 locking-mode の説明については、表 6-6 を参照してください。 isolation オプションの説明およびロック・モードとの関連については、表 6-7 を参照してください。

表 6-5 CMP Entity Bean のパフォーマンス・パラメータおよび説明（続き）

パラメータ	説明
max-tx-retries	詳細は、表 6-4 を参照してください。
update-changed-fields-only	コンテナによって、ejbStore の起動時に CMP Entity Bean の永続記憶領域に対するすべてのフィールドを更新するか、または変更されたフィールドのみを更新するかを指定します。 デフォルト値：true
validity-timeout	validity-timeout は、exclusive-write-access=true および locking-mode=read-only の場合のみ使用されます。 有効性タイムアウトとは、エンティティがキャッシュ内で有効な最大時間（ミリ秒）です（再ロード前）。外部的にデータが変更されていない（つまり exclusive-write-access=true に設定している）場合、この値を 0 または -1 に設定して、このオプションを無効にすることをお薦めします。これは、キャッシュ内のデータが、外部的に変更されていない読取り専用 EJB に対して常に有効になるためです。 EJB が外部的に変更されていない場合は、通常 exclusive-write-access=true を使用しますが、表が時折更新されるため、キャッシュを更新する必要があります。さらに、データが外部的に変更される間隔を予想して値を設定します。

注意： 次の CMP Entity Bean パラメータは、このリリースではサポートされていません。

max-instances、min-instances、pool-cache-timeout

表 6-6 CMP Entity Bean の Locking-Mode 値

ロック・モード値	説明
optimistic	複数のユーザーが Entity Bean をパラレル実行可能です。コミット時ロック・モードはリソースの競合を監視しないため、データ整合性の負担はデータベースの分離モードにかかります。 locking-mode のデフォルト値。
pessimistic	リソースの競合を管理し、パラレル実行はできません。1 度に 1 人のユーザーのみが Entity Bean を実行できます。即時ロックでは、データベースのシリアル化アクセスに "SELECT....FOR UPDATE" を使用します。
read-only	複数のユーザーが Entity Bean をパラレル実行可能です。コンテナは、Bean の状態の更新を許可しません。

locking-mode は isolation とともに、CMP を使用する EJB Entity Bean のデータベース整合性を確保します。表 6-7 に、一般的な locking-mode および isolation の組合せを示します。組合せは機能およびパフォーマンスの両方に関連していますが、通常はパフォーマンスよりもデータ整合性のための機能上の要件を優先して、選択するモードを決定します。

表 6-7 CMP Entity Bean の Locking-Mode と Isolation の関係

Locking-mode	Isolation	使用する状況
pessimistic	committed	データ整合性を保証する必要がある場合、および同じ行に対して頻繁に同時アップデートを実行することが予想される場合。
pessimistic	serializable	この組合せは、使用しないことをお薦めします。
optimistic	committed	read-commited セマンティクスによる、同じ行に対する同時読込みおよび更新が十分である場合。
optimistic	serializable	データ整合性を保証する必要がある場合。ただし同じ行に対して頻繁ではないが同時アップデートを実行することが予想される場合。
read-only	committed	反復可能読取りが必要でない場合。
read-only	serializable	反復可能読取りが必要な場合。

表 6-7 では、isolation 設定は、トランザクション isolation 属性の設定（明確に指定されている場合）、またはデータベースの分離レベル（トランザクションの isolation 属性が設定されていない場合）のいずれかを参照します。さらに、locking-mode およびトランザクションの分離レベルは CMP Bean の属性として設定されますが、トランザクションに対して有効になる分離レベルは、トランザクションで使用された最初の Entity Bean の分離レベルです。そのため、同じトランザクション内のすべての Bean を同じ分離レベルにすることが最も理想的です。

通常、コミット済分離レベルのコミット時ロックの方が多くの同時実行を行うことができるため高速ですが、データ整合性が必要とされる状況には適合しない場合があります。コミット済分離レベルの即時ロック、およびシリアル化可能分離レベルのコミット時ロックの方がスピードは遅くなりますが、更新時のデータ整合性を確実に保持できます。

Bean を読取り専用に定義すると、Bean への更新は行われません。読取り専用として定義されておらず、挿入、更新、削除に使用されない Bean（すなわち、読取りがコールされるメソッドのみ）と、パフォーマンスの面で同じようになります。これは、読取り専用ロックで定義されていない Bean において変更されたフィールドがない場合、すでに ejbStore を行わないように最適化されているためです。パフォーマンスを向上させ、読取り専用 Bean での ejbLoad の実行を回避するには、exclusive-write-access=true も設定する必要があります。

BMP Entity Bean の構成パラメータ

この項では、BMP を使用する Entity Bean に適用するパラメータについて説明します。このパラメータは、orion-ejb-jar.xml 構成ファイルで指定します。

表 6-8 では、Entity Bean の BMP 固有のパラメータについて説明しています。

表 6-8 BMP Entity Bean のパフォーマンス・パラメータおよび説明

パラメータ	説明
call-timeout	詳細は、表 6-4 を参照してください。
locking-mode	locking-mode パラメータで指定されるロック・モードでは、同時実行性を管理し、リソースの競合の管理のブロックおよびパラレル実行のタイミングを設定します。 BMP Bean は、Bean への同時アクセスを可能にするコミット時ロックを使用する必要があります。さらに、BMP Bean ではデータベースのアクセスおよびデータ整合性の管理を行う必要があります。分離の管理を行うかどうかは BMP Bean によって異なるため、分離の設定は BMP には適用されません。 デフォルト値: optimistic
max-tx-retries	詳細は、表 6-4 を参照してください。

注意: 次の BMP Entity Bean パラメータは、このリリースではサポートされていません。

max-instances、min-instances、pool-cache-timeout

Session Bean の構成パラメータ

この項では、orion-ejb-jar.xml 構成ファイルで指定されている、Session Bean に適用されるパラメータについて説明します。

表 6-9 では、ステートレス Session Bean 固有のパラメータを説明しています。

表 6-10 では、ステートフル Session Bean 固有のパラメータを説明しています。

表 6-9 ステートレス Session Bean のパラメータ

パラメータ	説明
call-timeout	詳細は、表 6-4 を参照してください。
cache-timeout	cache-timeout はステートレス Session EJB に適用されます。このパラメータでは、プールにキャッシュされたステートレス・セッションを保持する期間を指定します。 ステートレス Session Bean では、cache-timeout を指定すると、cache-timeout の間隔ごとに、プール内にある対応する Bean タイプのすべての Bean が削除されます。指定されている値が 0 またはマイナスの場合、cache-timeout は無効になり、Bean はプールから削除されません。 デフォルト値 : 60（秒）
max-tx-retries	詳細は、表 6-4 を参照してください。

表 6-10 ステートフル Session Bean のパラメータ

パラメータ	説明
call-timeout	詳細は、表 6-4 を参照してください。
timeout	timeout はステートフル Session EJB に適用されます。値が 0 またはマイナスの場合、すべてのタイムアウトは無効になります。 timeout パラメータは、ステートフル Session Bean の非アクティブ状態のタイムアウトです。30 秒ごとにプールをクリーン・アップするロジックが起動します。プールをクリーン・アップするロジック内では、timeout 値を経過してタイムアウトしたセッションのみが削除されます。 アプリケーションのステートフル Session Bean の使用状況に従って、timeout を調整します。たとえば、ステートフル Session Bean がアプリケーションによって明示的に削除されず、アプリケーションが多くのステートフル Session Bean を作成する場合、timeout 値を小さくする必要があります。 アプリケーションが 30 秒以上有効なステートフル Session Bean を必要とする場合、timeout 値をそれに従って調整します。 デフォルト値 : 30（秒）
max-tx-retries	詳細は、表 6-4 を参照してください。

複数の OC4J の使用および接続の制限

この項では、パフォーマンスの向上方法について説明し、OC4J インスタンス内のアクティブ・プロセス数の設定、別の OC4J インスタンスへのアプリケーションの送信、および OC4J インスタンスに送信するリクエストの数の制限を行うことによって実現します。

この項では、次の項目について説明します。

- HTTP 接続の制限
- 複数の OC4J プロセスの構成
- OC4J インスタンスでのアプリケーション均衡化

HTTP 接続の制限

特定のサイトで受け入れるアクティブな HTTP の同時接続数を制限することにより、J2EE アプリケーションのパフォーマンスを向上させることができます。Oracle HTTP Server で `mod_oc4j` を使用して、`httpd.conf` の `MaxClients` パラメータを設定することにより、受信するリクエスト数を制限できます。

関連項目： 5-10 ページの「Oracle HTTP Server ディレクティブの構成」

スタンドアロン OC4J の HTTP 接続の制限

スタンドアロン OC4J を使用している場合、HTTP 接続の最大数を制限することにより、OC4J サイトが同時に受け入れるアクティブ Web ユーザーの数を制限できます。システムが十分に処理できないほどの多くの同時ユーザーが存在するか、または限られたリソースしかなく処理能力が十分ではない場合、スタンドアロン OC4J のパラメータをチューニングすることによってパフォーマンスを向上させることができます。

HTTP 接続を制限するには、`server.xml` の `max-http-connections` 構成要素を使用して、`value`、`max-connections-queue-timeout` および `socket-backlog` 属性を指定します。

次のコードは、最大接続数を設定する `server.xml` の行の例です。

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100"/>
```

表 6-11 では、`max-http-connections` 属性について説明しています。

最大接続数の制限に達した際に、別の URL にメッセージをリダイレクトする場合は、HTTP のリダイレクト URL も記述します。

たとえば、http://optional.redirect.url/page.jspにリダイレクトするには、server.xml に次の行を追加します。

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100">http://optional.redirect.url/page.jsp</max-http-connections>
```

表 6-11 max-http-connections 属性の設定

属性	説明
max-connections-queue-timeout	最大接続数に達している場合、サーバー・ビジーまたはリダイレクト・メッセージを返す前に、接続の空きを待機する秒数を指定します。 デフォルト値 : 10（秒）
socket-backlog	ソケット・レベルで接続を拒否する前に、キューで待機する接続の数を指定します。 デフォルト値 : 30
value	最大接続数を指定します。 デフォルト値 : 100000

関連項目： OC4J の server.xml ファイルについての詳細は、『Oracle9*i*AS Containers for J2EE ユーザーズ・ガイド』の付録 B「追加情報」を参照してください。

複数の OC4J プロセスの構成

Oracle9*i*AS では、OC4J インスタンスの複数の OC4J プロセスを構成できます。システムに十分なリソースがある場合、J2EE アプリケーションで使用される OC4J インスタンスの複数の OC4J プロセスを構成することにより、パフォーマンスが向上する可能性があります。OC4J プロセスの複数使用は、Oracle9*i*AS クラスタリングを使用しない、ロード・バランシングの最も簡単な方法です。複数の OC4J プロセスでリクエストを処理するように OC4J インスタンスを構成すると、競合を回避し、拡張性を向上させることができます。さらに、Oracle9*i*AS では、複数ノードのクラスタリングによるロード・バランシングもサポートされています。

CPU に対する OC4J プロセスの最適な比率は、どのようなアプリケーション、ハードウェア構成、および JDK を使用しているかによって異なります。ただし、2 プロセッサ以上のマルチ CPU 構成を使用している場合は、複数の OC4J プロセスの構成を考慮に入れることをお薦めします。たとえば、最近行われた J2EE アプリケーションのテストでは、4 プロセッサ・システムの場合、1 つの OC4J プロセスで使用されるリソースはリソース全体の 70% でした。この結果から、OC4J プロセスの追加が効果的であることと、さらに多くの受信負荷を処理してマシンのリソースを有効利用できることが分かります。受信負荷を監視すると、前者の方法が効果的であり、2 つめのプロセスを追加するとパフォーマンスが向上することがわかりました。

システム・リソースの許容範囲を超えてプロセスを追加しても、パフォーマンスは向上しません。たとえば、1 つの OC4J プロセスでシステムの CPU リソースが十分に使用されている場合、4 つのプロセスを追加してもパフォーマンスが向上しないばかりか、逆に低下してしまうことがあります。最初の段階では、2 ～ 3 の CPU に対して 1 つの OC4J プロセスを構成し、さらにプロセスの追加によるパフォーマンスの向上を測定することをお薦めします。

Oracle Enterprise Manager を使用した複数の OC4J プロセスの構成

Oracle Enterprise Manager を使用すると、OC4J インスタンスのホーム・ページの OC4J インスタンス・レベル構成で、OC4J インスタンスのプロセス数を指定できます。

OC4J を構成して、同じアイランドで動作する複数 OC4J プロセスでロードバランシングを利用し、かつセッションをレプリケートしない場合、アプリケーションの `web.xml` ファイルで `<distributable/>` プロパティを設定しないようにしてください。

注意： `web.xml` で `<distributable/>` プロパティを設定すると、セッションを使用するアプリケーションに対して大量のパフォーマンス・オーバーヘッドをもたらします。これは、複数の OC4J プロセスが同じアイランドで動作している場合、アプリケーションがフェイルオーバー時にセッション・レプリケーションを使用するように構成されるためです。

関連項目：『Oracle9*i*AS Containers for J2EE ユーザーズ・ガイド』の第 3 章「高度な構成、開発およびデプロイ」

OC4J インスタンスでのアプリケーション均衡化

特に、マルチプロセッサ・システムで動作するアプリケーションの場合、J2EE アプリケーションの負荷を複数の OC4J インスタンスに分散すると効果的です。OC4J インスタンスを実行すると、単一・プロセッサ・ホスト上であっても、通常より高い**スループット**と短い**応答時間**が実現されます。

Web サイトにさまざまなアプリケーションが存在し、アプリケーションごとに異なる要件やニーズを持つ場合、複数の OC4J インスタンスを構成して、それぞれのアプリケーションに対して適切な数の OC4J プロセスを割り当てることができます。

アプリケーションを複数の OC4J インスタンスに配置するには、次の手順を実行します。

1. 複数の OC4J インスタンスを作成します。
2. アプリケーションの配布ウィザードを使用して、各インスタンスに適切なアプリケーションを配置し、各アプリケーションにマッピングされた固有の URL を指定します。

それぞれの OC4J インスタンスにアプリケーションを配置した後、パフォーマンスを監視して、スループットの増加および応答時間の短縮を確認できます。

データベースの監視およびチューニング

データベースを使用する Oracle9*i*AS OC4J の J2EE アプリケーションで高いパフォーマンスを実現するには、アクセスするデータベース表をパフォーマンスを考慮して設計する必要があります。さらに、システムを効果的に利用していることを確認するためにデータベース・サーバーを監視し、チューニングする必要があります。

関連項目：『Oracle9*i* データベース・パフォーマンス・ガイドおよびリファレンス』

Oracle9*i*AS での BC4J パフォーマンスの向上

この項には、Oracle Business Components for Java（BC4J）アプリケーションのメンテナンス性、拡張性、およびパフォーマンスを向上させるためのヒントが含まれています。

関連項目：「Oracle9*i*AS Business Components for Java」という見出しの下の「ビジネス・コンポーネントの開発」

適切な配置構成の選択

クライアントの Web モジュールにビジネス・コンポーネントを配置すると、アプリケーションの高いパフォーマンスと拡張性を得ることができます。分散トランザクションの使用または EJB のセキュリティ上の理由など、よほどの理由がない限り EJB 配置よりも Web モジュールにビジネス・コンポーネントを配置することをお薦めします。

Web モジュール配置および EJB 配置はどちらも J2EE に完全に準拠しており、BC4J フレームワークではこれらを簡単に切り替えることができます。両方のモードでアプリケーションをテストして、どちらでより高いパフォーマンスを得られるかを確認できます。

拡張性のためのアプリケーション・モジュール・プーリングの使用

クライアントでは、アプリケーション・モジュール・プーリングと呼ばれるアプリケーション・モジュール・インスタンスを使用できます。これには、次の利点があります。

- サーバー側リソースの取得時間を短縮できる
- 少数のインスタンスで、より多くのリクエストに対応できる
- 何千もの受信リクエストを処理する Web アプリケーションの要件に対応できる
- セッション状態の保存およびフェイルオーバーのサポートが可能になる

たとえば、ある Web アプリケーションの場合、1000 人のユーザーが存在しますが、100 人のみが特定のアプリケーション・モジュールを同時に使用しています。この場合、アプリケーション・モジュール・プールを使用します。クライアントがアプリケーション・モジュール・インスタンスを必要とする場合、プールから空いている 1 つを取り出して、コミット後またはトランザクションのロール・バック後にそれをプールに解放します。インス

タンスは事前に作成されているため、エンド・ユーザーはタスクを実行する際に、アプリケーション・モジュールをインスタンス化する時間を省くことができます。通常は、Web ベースの JSP クライアントがプールを使用します。アプリケーション・モジュール・プールに最大数である 100 のアプリケーション・モジュール・インスタンスが確実にあるようにしたい場合、デフォルトのアプリケーション・モジュール・プールをカスタマイズできます。

クライアントでアプリケーション・モジュールの状態を追跡する場合、ステートフル・モードを使用することをお薦めします。ステートフル JSP アプリケーションでは、アプリケーション・モジュールの数がリサイクルのしきい値を超えると、クライアントはアプリケーション・モード・インスタンスを予約せず、他のクライアントが使用できるようにします。そのかわり、状態が次のいずれかの方法で保存されます: アプリケーション・モジュールがリサイクルされていない場合、アプリケーション・モジュール・プールはクライアントの元のアプリケーション・モジュールを返し、データベース内のリサイクルされたアプリケーション・モジュールの状態を維持して、後でそれらをリクエストするクライアントが利用できるようにします。

ユーザー・セッションの最後にアプリケーション・モジュールを解放する場合、ステートフルまたは予約済モードではなく、必ずステートレス・リリース・モードを使用してください。これによりデータベースの空きを確保し、プールがアプリケーション・モジュールを直ちにリサイクルできるようになります。

カスタム・サブクラスを使用したグローバル・フレームワーク・コンポーネントのカスタマイズの実行

とりわけ大規模な組織では、特定の機能を特定のタイプの全コンポーネントで共有したい場合があります（たとえば、すべてのビュー・オブジェクトによる共有）。設計者は、任意の動作を実装する `MyOrgViewObjectImpl` などの Thin レイヤー・クラスを作成できます。各開発者は、`ViewObjectImpl` のかわりに `MyOrgViewObjectImpl` を拡張でき、「代替」機能を使用してレガシー・コードの `MyOrgViewObjectImpl` を拡張できます。

SQL-Only および Forward-Only ビュー・オブジェクトの使用（可能な場合）

エンティティ・オブジェクトのビュー・オブジェクトをベースにすると、ビュー・オブジェクトを使用してデータを挿入、更新、削除でき、同期された同一データに基づいてビュー・オブジェクトを維持するのに役立ちます。ただし、ビュー・オブジェクトが読取り専用問合せにのみ使用され、このビュー・オブジェクトで問合せされるデータが、同じアプリケーション・モジュールの他のビュー・オブジェクトから実行した保留変更を行う機会がない場合、基礎となるエンティティを持たない SQL のみのビュー・オブジェクトを使用します。これにより、行をエンティティ・キャッシュに追加する必要がないため、パフォーマンスが向上します。

Web ページや順方向に進行するバッチ処理のフォーマット・データのように一方向にデータをスクロールする場合、Forward-only ビュー・オブジェクトを使用できます。Forward-only モードでは、データがビュー・キャッシュに入ることはありません。Foward-only モードを使用すると、1 度に 1 つのビュー行のみがメモリーに入るため、メモリー・リソースおよび

時間を節約できます。ビュー・オブジェクトが 1 つまたは複数のエンティティ・オブジェクトに基づいている場合、データは通常の方法でエンティティ・キャッシュに渡されますが、行はビュー・キャッシュに追加されません。

アプリケーション・モジュールの肥大化防止

ルート・アプリケーション・モジュールは、1 つのタスク（1 つのデータベース・トランザクションに含まれるものすべて）に対応します。1 つのアプリケーション・モジュールのタスクに必要な数以上のビュー・オブジェクトまたはビュー・リンクを配置しないでください。

加えて、`createViewLink()` によるビュー・リンクの動的な作成を延期することも検討してください。すべてのビュー・リンクを設計時に含めると、クライアントがマスターのビュー・オブジェクトをナビゲートする際に、ビジネス・ロジック層は自動的にすべての詳細ビュー・オブジェクトに対する問合せを実行します。ビュー・リンクの作成を延期すると、まだ必要のない詳細ビュー・オブジェクトに対する問合せを、ビジネス・ロジック層が実行することはありません。

たとえば、リクエスト時にのみ詳細行が表示されるフォームに対して、設計時にビュー・リンクを含めると、必要のない問合せを自動的に実行することをビジネス・ロジック層に強制することになります。これを防ぐには、詳細行がリクエストされたときにビュー・リンクを動的に作成します。これとは対照的に、マスターが選択されると詳細行が表示されるフォームでは、設計時にビュー・リンクを作成して、`createViewLink()` と呼ばれるランタイム・オーバーヘッドを回避します。

適切なフェイルオーバー・モードの使用

デフォルトでは、アプリケーション・モジュール・プールではフェイルオーバーがサポートされます。これは、アプリケーション・モジュールがプールにチェックされるとアプリケーション・モジュールの状態を直ちにデータベースに保存します。ビジネス・ロジック層またはデータベースが中間トランザクションで動作不能になった場合（たとえば、電源障害やシステム・クラッシュ）、クライアントは消失したものと同じ状態をもつアプリケーション・モジュールを新たにインスタンス化できるため、作業が消えることはありません。

ただし、このような高レベルの信頼性を必要としないアプリケーションもあります。サーバーの障害による作業の消失を容認できる場合、フェイルオーバーを無効化できます。フェイルオーバーを無効にすると、アプリケーション・モジュールの状態は、データベースにコミットされる（この時点でアプリケーション・モジュールの状態は破棄される）か、リサイクルされる（この時点で、クライアントが取得できるように状態を保存する）までの間だけメモリー内に存在します。アプリケーション・モジュールのチェック毎にアプリケーション・モジュールを保存しないので、フェイルオーバー無効モードではパフォーマンスが向上する可能性があります。

メモリー量を抑えて多数の行をキャッシュするビュー行スピルオーバーの使用

ビジネス・ロジック層が動作している間、ビュー行はメモリーのキャッシュに保存されます(Java ヒープ)。ビジネス・ロジック層が 1 度に多くの行を保存する必要がある場合、メモリーが不足していないことを確認してください。これを確認するには、行数が一定のサイズに達する時期を指定します（ディスクに保存されるデータベースの、行による「オーバーフロー」)。この機能は、ビュー行スピルオーバーと呼ばれます。アプリケーションが大規模な問合せ結果を抱える場合、ビュー行スピルオーバーにより効率的にキャッシュが行われます。

バインド・パラメータの適切なスタイルの選択

Oracle スタイル・バインド・パラメータ（:1、:2、など）は、JDBC スタイル・バインド・パラメータよりも高いパフォーマンスを発揮します。

JDBC スタイル・バインド・パラメータを使用する理由は、次の 2 つだけです。

- Oracle JDBC ドライバ以外のドライバを使用している場合。
- `WHERE` 句に同じパラメータが複数回存在する場合。

設計時における問合せ条件の実装（可能な場合）

ビュー・オブジェクト・ウィザードの `WHERE` 句フィールドに、あらかじめ決定している問合せ条件を含めることができます。完全に動的な問合せ条件には、`setWhereClause()` のみを使用します。

問合せ条件が完全に動的な場合でも、`setWhereClause()` のかわりにパラメータ付きの問合せを使用できる場合があります。たとえば、`WHERE` 句に `EMPLOYEE_ID=<x>`（x は任意の値）を指定してビュー・オブジェクトを実行する場合、`EMPLOYEE_ID=:1` のようにパラメータ付きの `WHERE` 句を使用します。これは、`setWhereClause()` を繰り返しコールするよりも効果的です。

適切な JDBC フェッチ・サイズの使用

デフォルトの JDBC フェッチ・サイズは、すべてのアプリケーションにとってメモリ使用量とネットワーク使用量のバランスが最適になるように調整されています。ただし、メモリー使用量よりもネットワーク・パフォーマンスの方が重要である場合、JDBC フェッチ・サイズの値を上げることを検討してください。

バッチ処理で使用されるビュー・オブジェクトのイベント・リスニングをオフにする

非対話形式のバッチ処理では、ビュー・オブジェクトがエンティティ・オブジェクト・イベントをリスニングする必要はありません。このようなビュー・オブジェクトでは、`ViewObject.setListenToEntityEvents(false)` を使用してイベント・リスニングによるパフォーマンスのオーバーヘッドを取り除きます。

7

Web Cache の最適化

この章では、Oracle9*i*AS Web Cache 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。この章では、CPU の数、メモリーの量、ネットワークの帯域幅、ネットワーク接続の数などを含む、キャッシュ・サイズの決定について重点を置いて説明します。

この章には、次の項目が含まれています。

- Oracle9iAS Web Cache での 2CPU の使用
- Oracle9iAS Web Cache の十分なメモリーの設定
- 十分なネットワーク帯域幅の確保
- 適切なネットワーク接続数の設定

注意： 詳細については、『Oracle9*i*AS Web Cache 管理および配置ガイド』を参照してください。

Oracle9*i*AS Web Cache での 2CPU の使用

Oracle9*i*AS Web Cache は 1 または複数の CPU を使用するように設計されています。Oracle9*i*AS Web Cache はメモリ内キャッシュであるため、CPU サイクルによって制限されることはほとんどありません。CPU を追加しても飛躍的なパフォーマンスの向上は期待できません。ただし、プロセッサのスピードは非常に重要であるため、できるだけ高速な CPU を使用してください。

Oracle9*i*AS Web Cache は割り当て可能なメモリーの量によって制限を受けることに注意してください。メモリーを追加することにより、パフォーマンスおよび拡張性を向上できます。必要なメモリーの量についての詳細は、7-3 ページの「Oracle9iAS Web Cache の十分なメモリーの設定」を参照してください。

Oracle9*i*AS Web Cache には 3 つのプロセス（管理サーバー、auto-restart およびキャッシュ・サーバー）があります。

- 管理サーバーのプロセスは、Oracle9*i*AS Web Cache の設定および監視に使用されます。プロセスのアイドル時に、「Health Monitor」の設定に従って CPU のサイクルを使用します。たとえば、「Health Monitor」ページでリフレッシュ・レートに「Every 15 Seconds」を選択した場合、管理プロセスは「Every Minute」を選択した場合よりも多くの CPU を使用します。
- auto-restart プロセスでは、キャッシュが起動しているかどうかを確認します。起動していない場合は、キャッシュを再起動します。auto-restart では、最小量の CPU を使用します。
- キャッシュ・サーバーは、着信接続要求管理用とリクエスト処理用の 2 つのスレッドを使用します。これにより、2CPU は Oracle9*i*AS Web Cache の最適化を専用に行います。

Oracle9*i*AS Web Cache をコスト効率よく実行するには、大量のメモリーを持つ高速な 2CPU のコンピュータ上で実行します。さまざまな配置のシナリオについては、『Oracle9*i*AS Web Cache 管理および配置ガイド』を参照してください。

複数の Oracle9*i*AS Web Cache インスタンスを持つ Web サイトでは、キャッシュ・クラスタの一部、またはスタンドアロン・インスタンスとして別の 2CPU のノードにインストールすることを検討してください。Oracle9*i*AS Web Cache インスタンスが別のノードにある場合、オペレーティング・システムの制限（特にネットワーク・スループット）を受けることが少なくなります。たとえば、2 つの別々の 2CPU のノード上に 2 つのキャッシュがある場合、1 つの同じ 4CPU ノード上に 2 つのキャッシュがある場合よりもオペレーティング・システムの制限を受けにくくなります。

他のリソースが Oracle9*i*AS Web Cache と CPU の使用を競合している場合、必要な CPU の数を決める際には当然これらのリソースの要件を考慮します。Web Cache には別々のノードが最適ですが、アプリケーション・サーバーの残りの部分として同じノード上で実行している Web Cache からも、パフォーマンスのメリットを得ることができます。

Oracle9*i*AS Web Cache の十分なメモリーの設定

スワッピングによってドキュメントをキャッシュからの出し入れすることを回避するために、キャッシュに十分なメモリーを設定することが非常に重要です。一般的に、Oracle9*i*AS Web Cache のメモリー量（最大キャッシュ・サイズ）は 256MB 以上に設定する必要があります。デフォルトでは、最大キャッシュ・サイズは 50MB に設定されており、これは初期のポスト・インストール・テストのみに十分な量です。

必要なメモリーの量を正確に決定するために、次の手順を実行します。

1. キャッシュするドキュメントを決定し、4KB より大きいドキュメントと 4KB より小さいドキュメントの数を決定します。4KB より大きいドキュメントの平均のサイズを決定します。予期されるピーク・ロード（同時に処理されるドキュメントの最大数）を決定します。

   これを行う方法の 1 つは、既存の Web サーバーの 1 日のログを監視して、どのドキュメントが頻繁に使用されるかを知ることです。ログの URL のリストから、キャッシュするドキュメントを決定します。ドキュメントを取得して、ドキュメントの各サイズを取得します。

2. 必要なメモリーの量を計算します。計算方法は、Oracle9*i*AS Web Cache のバージョンによって異なります。

   Oracle9*i*AS Web Cache がドキュメントを保存するのに必要なメモリーのサイズは、ドキュメントのサイズによって異なります。

   - ドキュメントが 4KB より小さい場合、Oracle9*i*AS Web Cache は HTTP 本体を保存するために 4KB のバッファを使用します。
   - ドキュメントが 4KB より大きい場合、Oracle9*i*AS Web Cache は HTTP 本体を保存するために 32KB のバッファを使用します。たとえば、ドキュメントが 40KB の場合、Oracle9*i*AS Web Cache は HTTP 本体を保存するために 2 つの 32KB のバッファを使用します。
   - 本体のサイズにかかわらず、Oracle9*i*AS Web Cache は HTTP レスポンス・ヘッダーを保存するのに 4KB を使用します。

   必要な最大のメモリーを予測するには、次の式を使用します。

   ```
   ( X * ( 4KB + 4KB ) ) + ( Y * (( [m/32] * 32KB ) + 4KB )) + basemem
   ```

   式は次のようになっています。

   - *X* は、4KB より小さいドキュメントの数です。
   - 4KB は、4KB より小さいドキュメントの HTTP 本体のためのバッファ・サイズです。
   - 4KB は、HTTP レスポンス・ヘッダーのバッファ・サイズです。
   - *Y* は、4KB 以上のドキュメントの数です。

- [m/32] は、m（4KB 以上のドキュメントの平均サイズの KB）の上限を 32 で割ったものです。**上限**は、その数以上で、その数に最も近い整数です。
- 32KB は、4KB 以上のドキュメントの HTTP 本体のためのバッファ・サイズです。
- *basemem* は、リクエストを処理するために Oracle9*i*AS Web Cache が必要とする基本のメモリー量です。この量は、検索キーおよびタイムスタンプなどの内部ファンクションのメモリーを含みます。必要な量は、同時リクエストの数、およびリクエストが Edge Side Includes（ESI）を含むかどうかによって異なります。ESI は、HTML フラグメントの部分的なページのキャッシングを可能にするマークアップ言語です。

  非 ESI リクエストでは、各同時リクエストに約 6 ～ 25KB のメモリーが必要です。たとえば、1000 の同時リクエストをサポートするには、6 ～ 25MB のメモリーが必要です。

  ESI リクエストには、各同時リクエストはおよそ次の量のメモリーが必要です。

  ```
  60KB + (number of ESI fragments * [6KB to 25KB])
  ```

  たとえば、10 の ESI フラグメントを持つドキュメントでは、次の計算を使用します。

  ```
  60KB + (10 * [6KB to 25KB]) = 120KB to 330KB
  ```

  すなわち、10 フラグメントからなる 1 つのドキュメントに対して 120 ～ 330KB のメモリーが必要です。1000 の同時リクエストをサポートするには、およそ 120 ～ 330MB のメモリーが必要です。

たとえば、5000 の 4KB より小さいドキュメント、および 2000 の 4KB 以上のドキュメント（平均サイズが 54KB）をキャッシュすると仮定します。これらのドキュメントは ESI を使用しません。500 ドキュメントを同時に処理すると予測します。最大メモリーの計算には次の式を使用します。

```
(5000 * (4KB + 4KB) ) + ( 2000 * (( [54/32] * 32KB ) + 4KB )) + (500 *[6KB to
25KB])
```

式を使用するには、次の条件が必要です。

- 小さいドキュメントに 40,000KB。
- 大きいドキュメントに 136,000 KB。HTTP 本体には、平均サイズ 54KB の各ドキュメントに対して 64KB（2 つの 32KB バッファ）が必要です。HTTP レスポンス・ヘッダーでは、各ドキュメントに 4KB 必要です。
- 500 の同時リクエストを処理するには 3,000 ～ 12,500KB の基本のメモリー量を必要とします。

これにより、179,000 ～ 188,500 KB のメモリーが必要であることが予測されます。

注意： 特定のドキュメントをキャッシュするように指定しても、すべてのドキュメントが同時にキャッシュされるわけではありません。リクエスト済みで有効のドキュメントのみがキャッシュされます。その結果、特定のパーセンテージのドキュメントのみが常にキャッシュに保存されています。これは、前出の式から導き出される最大メモリーの量を必要としていない場合があることを意味します。

3. Oracle9*i*AS Web Cache を設定し、最大キャッシュ・サイズとして式の結果を指定します。結果はあくまでも予測であることに注意してください。

   最大キャッシュ・サイズを指定するには、次の手順に従います。

   a. 「ナビゲータ」画面で、**「Cache-Specific Configuration」-> 「Resource Limits」**を選択します。

   b. 「Resource Limits」ページでキャッシュを選択し、**「Edit」**をクリックします。

      「Edit Resource Limits」ダイアログ・ボックスが表示されます。

   c. **「Maximum Cache Size」**フィールドで、式の結果を入力します。

   d. **「Submit」**をクリックします。

   e. Oracle9*i*AS Web Cache Manager のメイン・ウィンドウで、**「Apply Changes」**をクリックします。

4. Oracle9*i*AS Web Cache を再起動します。

5. シミュレーションのロードまたは実際のロードを行うには、キャッシュを監視して、実際にどの程度のメモリーが使用されるかを確認します。

   Oracle9*i*AS Web Cache が起動する際にはキャッシュは空であることに注意してください。有効であることを監視するために、キャッシュがすべて移入されていることを確認してください。すなわち、指定されている数のドキュメントをキャッシュするために、キャッシュが十分なリクエストを受け取っているかを確認します。

   「Web Cache Statistics」ページでは、現在のメモリー使用量および最大のメモリー使用量についての情報を提供します。

   「Web Cache Statistics」ページにアクセスするには、「ナビゲータ」画面から、**「Administration」->「Monitoring」->「Web Cache Statistics」**を選択します。Cache Overview 表で、次のメトリックスに注意してください。

   - **「Size of Documents in Cache」**では、現在のキャッシュの論理サイズを示します。キャッシュの論理サイズは、キャッシュ内の有効なドキュメントのサイズです。たとえば、キャッシュが 2 つのドキュメント（3KB と 50KB）を含んでいる場合、**「Size of Documents in Cache」**は 2 つのファイルの合計である 53KB になります。

- 「**Configured Maximum Cache Size**」では、「Resource Limits」ページに指定されている最大のキャッシュ・サイズが示されます。
- 「**Current Allocated Memory**」では、キャッシュの物理サイズが表示されます。キャッシュの物理サイズは、キャッシュの保存および処理用に Oracle9*i*AS Web Cache によって割り当てられたデータ・メモリーの量です。この数字は、常にオペレーティング・システムの統計によって示される処理サイズよりも小さくなります。これは、他のユーザー・プロセスと同様に、Oracle9*i*AS Web Cache プロセスもメモリーを指示記憶域、スタック・データ、スレッド、ライブラリ・データのような他の方法で消費するためです。
- 「**Current Action Limit**」は、「**Configured Maximum Cache Size**」の 90% です。この数字は通常「**Current Allocated Memory**」よりも大きくなります。

「**Current Allocated Memory**」が「**Current Action Limit**」よりも大きい場合、Oracle9*i*AS Web Cache はガベージ・コレクションを開始します。すなわち、Oracle9*i*AS Web Cache は、最も頻繁に使用される有効なドキュメントが最大のキャッシュ・サイズを超えずに新しい HTTP レスポンスのためのスペースを確保できるように、使用頻度が低く、有効性の低いドキュメントを削除します。

「**Current Allocated Memory**」が「**Current Action Limit**」に近いかまたはそれより大きい場合、最大キャッシュ・サイズを増加してスワップによるドキュメントのキャッシュからの出し入れを回避します。「**Cache-Specific Configuration**」->「**Resource Limits**」ページを使用して、最大キャッシュ・サイズを増加します。

十分なネットワーク帯域幅の確保

Oracle9*i*AS Web Cache を使用している場合、各ノードがスループットのロードに適合するだけのネットワーク帯域幅を持っていることを確認してください。そうでない場合、Oracle9*i*AS Web Cache に追加容量があるにも関わらず、ネットワークが飽和状態になります。たとえば、アプリケーションが 1 秒間に 100 メガビット以上のデータを生成する場合、10/100 Megabit Ethernet は飽和状態になることがあります。

ネットワークが飽和状態になった場合、10/100 Megabit Ethernet ではなく、Gigabit Ethernet の使用を検討してください。Gigabit Ethernet では、ネットワークの競合、再送信、帯域幅の不足を回避するための最も効率的な配置シナリオを提供します。さらに、2 つの異なるネットワーク・カードを使用することを検討してください。1 つはクライアント・リクエストの受信用、もう 1 つはキャッシュからアプリケーション Web サーバーへのリクエスト用です。

システムの監視によって、スループットが予想よりも低く、ネットワークが有効活用されていないことがわかった場合、CPU が飽和状態に達していないかを確認してください。

適切なネットワーク接続数の設定

Oracle9*i*AS Web Cache サーバーに対して、適切な数の最大接続数の制限を指定することが重要です。この数が大きすぎると、パフォーマンスに影響し、その結果応答時間が長くなります。低すぎる値を設定すると、処理されるリクエストの数が減ります。そのため、応答時間と同時に処理するリクエストの数のバランスを取ることが必要です。

適切な数を決定するには、次の要因について考慮してください。

- 同時にサービスするクライアントの最大数
- ページの平均サイズおよびページごとの平均リクエスト数
- ネットワーク帯域幅。1 度に送信できるデータの量は、ネットワーク帯域幅によって制限されます。詳細は、7-6 ページの「十分なネットワーク帯域幅の確保」を参照してください。
- キャッシュ・ミスの割合 リクエストが高い割合でキャッシュ・ミスする場合、リクエストはアプリケーションの Web サーバーに転送されます。これらのリクエストは追加のネットワーク帯域幅を消費するため、結果的に応答時間が長くなります。
- ページの処理速度。`ttcp` などのネットワーク監視ユーティリティを使用して、システムがページを処理する速度を決定します。
- キャッシュ・クラスタ環境を持っている場合、キャッシュ・クラスタ・メンバーの容量。容量は、他のキャッシュ・クラスタ・メンバーからの着信接続要求を反映します。クラスタ・メンバー容量の設定は、Oracle9*i*AS Web Cache Manager の**「Administration」->「Cluster Configuration」**ページを使用して行います。

オペレーティング・システムおよび Oracle9*i*AS Web Cache によって提供されるツールなどのさまざまなツールを使用して、最大接続数を決定します。たとえば、`netstat -a` コマンドを使用すると、確立されている接続数を決定できます。また、`ttcp` ユーティリティでは、ページが処理される速度を決定できます。Oracle9*i*AS Web Cache Manager では、ヒットおよびミスの統計を提供します。

着信接続要求の最大数を設定するには、次の手順に従います。

1. Oracle9*i*AS Web Cache Manager の「ナビゲータ」画面で、**「Cache-Specific Configuration」-> 「Resource Limits」**を選択します。
2. 「Resource Limits」ページでキャッシュを選択し、**「Edit」**をクリックします。

   「Edit Resource Limits」ダイアログ・ボックスが表示されます。
3. **「Maximum Incoming Connections」**フィールドで、新しい値を入力します。
4. **「Submit」**をクリックします。
5. Oracle9*i*AS Web Cache Manager のメイン・ウィンドウで、**「Apply Changes」**をクリックします。

無理に高い値を設定しないようにしてください。Oracle9*i*AS Web Cache がそれらの接続用に一部のリソースを確保するため、パフォーマンスに悪影響を及ぼします。UNIX システムでは、通常 5000 の接続数が適切な数値です。

UNIX プラットフォームでの接続

ほとんどの UNIX プラットフォームでは、各クライアント接続は異なるファイル記述子を必要とします。Oracle9*i*AS Web Cache サーバーは、起動時にファイル記述子の最大数の確保を試みます。`webcachectl` ユーティリティを `root` として実行できる場合、この値を増加できます。たとえば Sun Solaris では、`rlim_fd_max` パラメータを設定することにより、ファイル記述子の最大数を増加できます。`webcachectl` を `root` として実行不可能な場合、Oracle9*i*AS Web Cache サーバーはエラー・メッセージをログし、起動に失敗します。

`webcachectl` ユーティリティをルート・ユーザーとして実行するには、インストール時に `root.sh` が起動していることを確認してください。`root.sh` がインストール時に起動していない場合、`$ORACLE_HOME` ディレクトリからこれを起動します。Solaris コマンドおよび `webcachectl` ユーティリティについての情報は、『Oracle9*i*AS Web Cache 管理および配置ガイド』を参照してください。

Oracle9*i*AS Web Cache は、次の式を使用して、使用されるファイル記述子の最大数を計算します。

```
Max_File_Desc = Curr_Max_Conn + Total_WS_Capacity + Outgoing_Cluster_Conn + 100
```

式は次のようになっています。

- `Max_File_Desc` は、使用されるファイル記述子の最大数です。
- `Curr_Max_Conn` は、Oracle9*i*AS Web Cache の最大着信接続要求の制限です。最大着信接続要求数を設定するには、Oracle9*i*AS Web Cache Manager の**「Cache-Specific Configuration」>「Resource Limits」**ページを使用します。

  キャッシュ・クラスタ環境では、`Curr_Max_Conn` はピア・キャッシュからの着信接続要求であるクラスタ・メンバー容量を含みます。容量の設定は、Oracle9*i*AS Web Cache Manager の**「Administration」->「Cluster Configuration」**ページを使用して行います。
- `Total_WS_Capacity` は、すべての設定されたアプリケーション Web Server の容量の合計です。容量の設定は、Oracle9*i*AS Web Cache Manager の**「General Configuration」->「Application Web Servers」**ページを使用して行います。

  キャッシュ・クラスタ環境では、次の式を使用して容量をキャッシュ・クラスタ・メンバーの間で分けます。

  ```
  Total_WS_Capacity = Sum_Web_Server_Capacity / n
  ```

この式では、Sum_Web_Server_Capacity は設定されたアプリケーション Web サーバーの容量の合計で、n はキャッシュ・クラスタ・メンバーの数です。たとえば、2 つの設定済みのアプリケーション Web サーバーがあると仮定します。Web_Server_A の容量は 200 で、Web_Server_B の容量は 250 です。また、3 つのキャッシュを持つクラスタがあると仮定します。Total_WS_Capacity は 150 で、次は式の例です。

```
Total_WS_Capacity = (200 + 250) / 3
```

- Outgoing_Cluster_Conn は、キャッシュ・クラスタのピア・キャッシュへの送信接続の合計です。この値は、キャッシュ・クラスタがない場合は 0 です。この値を計算するには、次の式を使用します。

  ```
  Outgoing_Cluster_Conn = Sum_Cluster_Capacity / (n-1)
  ```

  この式では、Sum_Cluster_Capacity はクラスタ内のほかのすべての Web キャッシュの容量の合計で、n はキャッシュ・クラスタ・メンバーの数です。たとえば、3 つのキャッシュを持つクラスタがあると仮定します。Cache_A の容量は 100、Cache_B は 150、Cache_C は 200 です。Cache_A の Outgoing_Cluster_Conn は 175 で、次のように計算されます。

  ```
  Outgoing_Cluster_Conn = (150 + 200) / (3-1)
  ```

  クラスタにキャッシュの容量を設定する場合、Oracle9*i*AS Web Cache Manager の「ナビゲータ」画面から、**「Administration」->「Cluster Configuration」**を選択します。

- 100 は Oracle9*i*AS Web Cache の内部使用のために予約された接続の数です。

Windows NT および Windows 2000 での接続

Windows NT および Windows 2000 では、ファイル・ハンドルおよびソケット・ハンドルは、利用可能なカーネル・リソースのみ、すなわちページおよび非ページのプール・サイズのみによって限定されています。ただし、アクティブな TCP/IP 接続の数は、システムがオープンできる TCP ポートの数によって制限されます。

TCP ポートのデフォルトの最大数は、オペレーティング・システムによって 5000 に設定されています。これらのうち 1024 はカーネルによって予約されています。Windows のレジストリを編集することにより、ポートの最大数を変更できます。Windows NT および Windows 2000 では、65534 までのポートを使用できます。

デフォルトを変更するには、次のレジストリ・キーに新しい値を追加します。

```
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥Tcpip¥Parameters
```

新しい値を追加するには、次のように指定します。

- 値名：MaxUserPort
- データ・タイプ：REG_DWORD
- データ：1024 から 65354 までの整数

クラスタ・メンバー容量および着信接続要求の最大数の合計は、TCP ポートの数から 1024 を引いた値よりも大きい値を設定してはいけません。（最大着信接続要求数を設定するには、Oracle9*i*AS Web Cache Manager の**「Cache-Specific Configuration」->「Resource Limits」**ページを使用します。クラスタ・メンバー容量の設定は、**「Administration」->「Cluster Configuration」**ページを使用して行います。）

Windows NT および Windows 2000 では、Oracle9*i*AS Web Cache はファイル・ハンドルを予約したり、着信接続要求の現在の最大数が TCP ポートの数よりも小さいかどうかのチェックを行いません。

8

PL/SQL のパフォーマンスの最適化

Oracle9*i* Application Server での PL/SQL のパフォーマンスを最適化する方法について説明します。

このドキュメントには、次の内容が含まれています。

Oracle9*i*AS での PL/SQL のパフォーマンス - 概要

この章では、Oracle9*i* Application Server（Oracle9*i*AS）で PL/SQL アプリケーションのパフォーマンスを向上するためのいくつかの方法について説明します。

表 8-1 では、Database Access Descriptor（DAD）のパラメータおよび設定の推奨事項についてリストしています。デフォルトでは、これらの DAD パラメータは `$ORACLE_HOME/Apache/modplsql/conf` ディレクトリの `plsql.conf` ファイルで指定されています。

表 8-2 では、キャッシング・オプションをリストしています。

表 8-1 Database Access Descriptor（DAD）のパラメータ

パラメータ	設定
`PlsqlAlwaysDescribeProcedure`	最高のパフォーマンスを得るには、この値を O f f に設定する。
`PlsqlDatabaseConnectString`	TNS エントリのかわりに host:port:sid フォーマットを使用する。
`PlsqlFetchBufferSize`	デフォルト =128 日本語や中国語などのマルチバイト・キャラクタ・セットでは、この値を 256 に設定するとパフォーマンスが向上する。
`PlsqlIdleSessionCleanupInterval`	デフォルト =15（分） このパラメータを増加すると、プールされたデータベース接続をさらに長時間継続する。
`PlsqlLogEnable`	デフォルト =off このパラメータは、デバッグの目的で Oracle サポートに推奨されない限り Off のままにする。
`PlsqlMaxRequestsPerSession`	デフォルト =1000 PL/SQL アプリケーションがリソースおよびメモリーをリークしていない場合は、この値をさらに高い値に設定可能（たとえば、5000）。
`PlsqlNLSLanguage`	このパラメータをデータベースの NLS 言語に適合するように設定すると、Oracle Net Services でのキャラクタ・セット変換のオーバーヘッドが無効になる。
`PlsqlSessionStateManagement`	データベースが 8.1.7.2 以上の場合は、このパラメータを "StatelessWithFastResetPackageState" に設定する。 Oracle9*i*AS Portal では、`StatelessWithFastResetPackageState` モードはまだ対応していない。Oracle9*i*AS Portal では、このパラメータの値を `StatelessWithResetPackageState` に設定する。

表 8-2 キャッシング・オプション

期限方式	変更が予想可能なコンテンツに対して最高のパフォーマンスを提供する。 **関連項目：**8-20 ページの「期限方式の使用」
検証方式	変更が予想不可能なコンテンツに対してよいパフォーマンスを提供する。 **関連項目：**8-15 ページの「検証方式の使用」
システム・レベル・キャッシング	システム上のおのおのに対して 1 つのコピーをキャッシュすることにより、パフォーマンスを向上する。 **関連項目：**8-23 ページの「PL/SQL Web アプリケーションでのシステム / ユーザーレベルのキャッシング」

関連項目：

- mod_plsql メトリックスの詳細については、付録 A「Oracle9iAS パフォーマンス・メトリック」を参照。
- DAD パラメータの情報については、『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』の第 6 章「Oracle HTTP Server モジュール」を参照。
- 『Oracle9*i* Application Server mod_plsql ユーザーズ・ガイド』
- 『Oracle9*i* Application Server PL/SQL Web Toolkit リファレンス』

mod_plsql のパフォーマンス・チューニングの問題

Web アプリケーション上で PL/SQL のパフォーマンスを向上するために mod_plsql をチューニングする場合、mod_plsql の内容について知っておくことが重要です。このセクションでは、いくつかの mod_plsql 機能についての概要を説明します。

このセクションでは、次の項目について説明します。

- mod_plsql の接続のプーリング
- プールされたデータベース・セッションのクローズ
- データベースの再起動時の mod_plsql 接続プールにおける動作

mod_plsql の接続のプーリング

mod_plsql での接続プーリングの論理は、次の例で最もよく説明されています。次の典型的なシナリオを見てください。

1. Oracle9*i* Application Server リスナーが起動します。mod_plsql によって維持されている接続プールにデータベース接続はありません。
2. ブラウザは、Database Access Descriptor（DAD）D1 に対して mod_plsql リクエスト（R1）を作成します。
3. Oracle HTTP Server プロセスの 1 つ（httpd プロセス P1）が起動して、リクエスト R1 を処理します。
4. プロセス P1 の mod_plsql は、接続プールをチェックしますが、このユーザー・リクエストに対するデータベース接続がプールに存在ません。
5. DAD D1 の情報に基づいて、プロセス P1 の mod_plsql が新しいデータベース接続をオープンして、PL/SQL のリクエストを処理し、プールにデータベース接続を追加します。
6. この時点で、DAD D1 のプロセス P1 に後続するすべてのリクエストは、mod_plsql によってプールされたデータベース接続を利用できます。
7. DAD D1 に対するリクエストが他のプロセス（プロセス P2）によってピックされた場合、プロセス P2 の mod_plsql は別のデータベース接続を開き、リクエストを処理し、プールにデータベース接続を追加します。
8. この時点で、DAD D1 のプロセス P2 に後続するすべてのリクエストは、mod_plsql によってプールされたデータベース接続を利用できます。
9. ここで、リクエスト R2 が DAD D2 に対して作成され、このリクエストがプロセス P1 にルートされると仮定します。
10. プロセス P1 の mod_plsql は、DAD D2 用にプールされたデータベース接続を持たず、DAD D2 に対する新しいデータベース・セッションが作成され、リクエストを処理した後にプールされます。プロセス P1 は 2 つのプール済みのデータベース接続を持ち、1 つは DAD D1 用、もう 1 つは DAD D2 用です。

この例での重要な詳細は次のようになります。

- 各 Oracle HTTP Server プロセスは、統計ファイル・リクエスト、サーブレット・リクエスト、および mod_plsql リクエストなどのすべてのタイプのリクエストにサービスします。その次のリクエストにサービスする Oracle HTTP Server プロセスを制御できません。
- 1 つの Oracle HTTP Server プロセスは、他のプロセスで作成された接続プールを使用したり、共有したりすることはできません。
- 各 Oracle HTTP Server プロセスは、できるだけ各 DAD に 1 つのデータベース接続をプールします。

- ユーザー・セッションは、DAD のプールされたデータベース接続内で切り替えられます。Oracle9*i*AS Single Sign-On（SSO）に基づく DAD では、ユーザー・セッションの切り替えにプロキシ認証を使用します。非 SSO ユーザーは、DAD ではなくユーザー名およびパスワードによる HTTP の BASIC 認証を使用して、同じ接続上で再認証を行います。
- 複数の DAD が同じデータベース・インスタンスをポイントしている場合がありますが、データベース接続は同じプロセス内であっても異なる DAD で共有されません。
- 未使用の DAD はデータベース接続を使用しません。

最悪のシナリオでは、mod_plsql によってプールされるデータベース接続の合計数は、1 つの Oracle9*i* Application Server インスタンスに対して同時に起動している Oracle HTTP Server（httpd）プロセス数を掛けたアクティブ DAD の合計数の係数になります。Oracle HTTP Server プロセスをより大きい数に設定した場合、対応する量のデータベース・セッションを処理できるようにバックエンド・データベースを設定する必要があります。

たとえば、3 つの Oracle9*i*AS インスタンスがあり、それらがおのおの最大 50 の httpd プロセスおよび 2 つのアクティブな DAD を生成するように設定されている場合、300（3*50*2）セッションを実行できるようにデータベースを設定する必要があります。この数は、他のアプリケーションが接続を行うために必要なセッションの数を含んでいません。

異なる httpd プロセスでデータベース接続を共有できないため、プロセス・ベースのプラットフォームは接続プーリングよりも多くの接続の再使用機能を持ちます。これは、Oracle HTTP Server でのプロセスモデルの構造であることに注意してください。Oracle HTTP Server が将来スレッドされた場合、mod_plsql は本物の接続プーリングを使用します。データベース・セッションの数が問題である場合、これを解決するための方法についての詳細は 8-11 ページの「2 リスナー計画」を参照してください。

プールされたデータベース・セッションのクローズ

プールされたデータベース・セッションは、次の状況でクローズします。

- 設定された数のリクエストを処理するために、プールされた接続が使用されている場合

  デフォルトでは、mod_plsql によってプールされた各接続は最大 1000 のリクエストにサービスし、その後データベース接続を停止して再確立します。これは、PL/SQL アプリケーションまたは Oracle のクライアント / サーバー側でのリソースのリークがシステムに影響しないようにするために行われます。デフォルトの 1000 は、DAD の構成パラメータである PlsqlMaxRequestsPerSession をチューニングすることにより変更できます。

- プールされた接続が長時間アイドル状態である場合

  デフォルトでは、プールされた各接続はアイドル時間が 15 分に達すると自動的にクリーンアップされます。この処理では、mod_plsql のクリーンアップ・スレッドによって行われます。ロードの多いサイトでは、各接続は少なくとも 15 分に 1 度は使用されるため、接続のクリーンアップは長期間行われない場合があります。このような場合、接続は PlsqlMaxRequestsPerSession の設定に基づいてクリーンアップされます。

デフォルトの 15 分は、mod_plsql の構成パラメータである PlsqlMaxRequestsPerSession をチューニングすることにより変更できます。サイトのロードが多くない場合、パフォーマンスの向上のためにこのデフォルト値を増加することを検討してください。

- Oracle HTTP Server プロセスがダウンする場合

  Oracle HTTP Server プロセスが停止すると、Oracle HTTP Server 構成パラメータ MaxRequestsPerChild が管理を行います。たとえば、このパラメータが 5000 に設定されている場合、各 Oracle HTTP Server プロセスは停止する前に正確に 5000 のリクエストを処理します。また Oracle HTTP Server プロセスは、構成パラメータ MinSpareServers、MaxSpareServers および MaxClients に基づいて、Oracle HTTP Server メンテナンスの一部として起動および停止します。mod_plsql 接続プーリングを有効にするには、各 Oracle HTTP Server プロセスが一定の時間アクティブになるように Oracle9*i*AS の Oracle HTTP Server を設定することが非常に重要です。Oracle HTTP Server が正しく設定されていないと、Oracle HTTP Server プロセスが頻繁に起動および停止するようになります。このような設定では、新しい各 Oracle HTTP Server プロセスが接続プールを満たしてからでないと、後続のリクエストがプーリングを有効利用できません。

  関連項目：『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』の第 6 章「Oracle HTTP Server モジュール」を参照。

データベースの再起動時の mod_plsql 接続プールにおける動作

これは第一に、データベースが停止している時間によって異なります。データベースが停止してから 15 分以上して再起動した場合、ユーザーが Oracle9*i*AS リスナーを使用する際に何も問題は起きません。これは、mod_plsql クリーンアップ・スレッドが 15 分以上使用されていないセッションをクリーン・アップするためです。アイドル・セッションをクリーン・アップするために指定された値は、PlsqlIdleSessionCleanupInterval 構成パラメータを使用してチューニングできます（デフォルト値は 15 分）。

データベースが 15 分以内に再起動した場合、初期リクエストのいくつかはエラーとして返されますが、システムは迅速に使用できる状態に復帰します。エラーが返されるリクエストの数は、mod_plsql にプールされた接続の数と同じです。

関連項目： 表 8-1「Database Access Descriptor（DAD）のパラメータ」

mod_plsql のパフォーマンス・チューニングの領域

mod_plsql を使用する際、パフォーマンスおよびスケーラビリティに影響を及ぼす 3 つの領域があります。

- PL/SQL アプリケーション
- 接続プーリングおよび Oracle HTTP Server 構成
- データベース・セッションの数のチューニング

PL/SQL アプリケーション

PL/SQL ゲートウェイ・ユーザーは、PL/SQL アプリケーションを開発する際に次の項目について考慮する必要があります。

- Database Access Descriptors（DAD）

  各 Oracle9*i*AS ノードで使用される DAD の数を制限する必要があります。使用されていない追加の DAD があると、パフォーマンスに影響することに注意してください。詳細については、『Oracle9*i* Application Server PL/SQL Web Toolkit リファレンス』を参照してください。

- ネストされた表

  PL/SQL では表を作成できます。PL/SQL 表を作成するには、索引およびデータタイプを持つ表を作成します。表の索引は、-2147483647 ～ +2147483647 のバイナリ整数です。この表索引オプションでは Sparsity と呼ばれ、顧客番号、従業員番号またはその他の索引キーなどの重要な索引番号を使用できます。PL/SQL 表は、大量のデータを処理するために使用します。

  PL/SQL では TABLE および VARRAY（変数サイズ配列）コレクション・タイプを提供します。TABLE コレクション・タイプはネストされた表と呼ばれます。ネストされた表は無制限のサイズを持つことができ、疎密にできます。これは、ネストされた表内のエレメントを DELETE プロシージャを使用して削除可能であることを意味します。変数サイズ配列のサイズには制限があり、データベースに保存される際に順序および添字を維持します。ネストされた表のデータは、その表に関連付けられたシステムの表内に保存されます。変数サイズ配列は、アプリケーションがバッチ配列スタイルでデータを処理するバッチ処理に適しています。ネストされた表は、ネストされた表をストレージ表に保存することにより、効率的な問合せに役立ちます。そのストレージ表では、各エレメントがストレージ表の行にマップしています。

- PL/SQL アプリケーションでは、プロシージャ名のオーバーロード機能を使用する際には注意してください。複数のプロシージャに異なる名前を使用して、プロシージャ名のオーバーロードを回避することが最良の方法です。

- URL で、スカラーまたは配列などのパラメータ・タイプについて十分な詳細を提供していないプロシージャを実行する場合、PL/SQL アプリケーションでのオーバーヘッドについて注意してください。このような場合、プロシージャを実行するための最初の試みに失敗し、mod_plsql によって実行される前にプロシージャ署名を記述する必要があります。
- プロシージャには、4 パラメータ・スタイルのパラメータ渡しではなく、高パフォーマンスの 2 パラメータ・スタイルのフレキシブル・パラメータ渡しを使用します。

接続プーリングおよび Oracle HTTP Server 構成

- 新しいデータベース接続の作成は大変な作業であり、各リクエストがデータベース接続をオープンおよびクローズをする必要がないことが最も効率的です。最適な方法は、あるリクエストでオープンされたデータベース接続が後続のリクエストで再利用されるようにします。まれなケースですが、データベースのアクセスが非常に少なく高パフォーマンスが重要な問題ではない場合、接続のプーリングを無効化できます。たとえば、管理者が管理タスクを行うためにサイトにまれにアクセスする場合、管理アプリケーションへのアクセスに使用する DAD で接続プーリングを無効化できます。これによりは、パフォーマンスは低下しますが、データベース・セッションの数を減らすことができます。
- プロセスが一度起動したら、そのプロセスを一定の時間維持するように、Oracle HTTP Server 構成を正しく設定する必要があります。そうしないと、mod_plsql の接続プーリングを有効利用できません。Oracle9*i*AS リスナーは、プロセスの起動および停止を頻繁に行わないようにする必要があります。サイトのロードを正しく分析して、Web サイトのロードの平均を決定します。httpd プロセスがシステムの平均のロードを処理できるように、Oracle HTTP Server 構成を正しくチューニングする必要があります。さらに、httpd.conf ファイルの MaxClients 構成パラメータは、ランダムなロードの上昇を処理できる必要があります。
- Oracle HTTP Server プロセスは、プロセスが最終的に停止して再起動できるように設定します。これは、Oracle HTTP Server からアクセスするさまざまなコンポーネントにおいて起こり得るメモリー・リークを管理するために必要です。これは特に、データベース・セッションのリソース・リークが問題を引き起こさないようにするために mod_plsql で必要になります。MaxRequestsPerChild 構成パラメータが高い数値に設定されていることを確認してください。mod_plsql アプリケーションでは、これを 0 に設定しないでください。
- ロードの多いサイトでは、Oracle HTTP Server の KeepAlive 構成パラメータを無効に設定する必要があります。これにより、プロセスが現在のリクエストの処理を終了すると、各プロセスは直ちに他のクライアントからのリクエストを処理することができるようになります。Oracle9*i*AS リスナーへの同時リクエストの数よりも Oracle HTTP Server プロセスの数が常に多い、ロードの少ないサイトで KeepAlive パラメータを有効にすると、パフォーマンスが向上します。このような場合、KeepAliveTimeout パラメータ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。

- Oracle HTTP Server 構成で Timeout の値を低く設定することもできます。これにより、クライアントがすぐに応答しない場合、Oracle HTTP Server プロセスが事前にフリーになります。この値を低すぎる値に設定しないでください。値が低すぎると、クライアントの応答が遅くなるため、タイム・アウトが起こる場合があります。
- ほとんどの Web サイトは、一定のユーザー・インタフェースを保つために各画面に表示される静的イメージ・ファイルを持っています。このようなファイルはほとんど変更がないため、Oracle9*i*AS リスナーでサーブされる各イメージを mod_expires でタグ付けすることによって、システム上でのロードを大幅に減少できます。mod_expires の使用方法の詳細については、『Oracle9*i* Application Server mod_plsql ユーザーズ・ガイド』を参照してください。Web サイトを Oracle9*i*AS Web Cache でフロントエンドにすることも検討してください。
  - Web サイトでの mod_expires の使用が有効であるかどうかの確認
    - Netscape を使用して、サイト上のアクセスの多い Web ページをいくつか開きます。各ページでマウスを右クリックし、ポップ・アップ・メニューから「View Info」を選択します。ページの情報ウィンドウのトップ・パネルが多くの異なるイメージおよび静的コンテンツをリストしている場合、そのサイトでは mod_expires の使用が有効です。
    - また、Oracle HTTP Server アクセス・ログをチェックして、リクエストの何パーセントが HTTP 304（変更されませんでした）ステータスになるかを確認します。grep ユーティリティを使用して access_log の 304 を探し、この結果の行数を access_log の合計の行数で割ります。このパーセンテージが高い場合、mod_expires を使用すると有効です。
  - 静的ファイルを Expires ヘッダーでタグ付けする方法
    - 静的イメージ・ファイルに使用する Location ディレクティブを探します。ExpiresActive および ExpiresDefault ディレクティブをそれに追加します。

      ```
      Alias /images/ "/u01/app/oracle/myimages/"
      <Directory "/u01/app/oracle/myimages/">
           AllowOverride None
           Order allow, deny
           Allow from all
           ExpiresActive On
           ExpiresDefault A2592000
      </Directory>
      ```

      ブラウザは、現在から 30 日以内の日付を持つ /images パス内にあるすべてのサーブされる静的ファイルをキャッシュします。詳細は、『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』を参照してください。
    - Oracle9*i*AS Web Cache を使用している場合、Surrogate-Control ヘッダーを使用してこれらのファイルをメモリーにキャッシュできます。たとえば、次のようなものです。

```
Alias /images/ "/u01/app/oracle/myimages/"
<Directory "/u01/app/oracle/myimages/">
     AllowOverride None
     Order allow, deny
     Allow from all
     ExpiresActive On
     ExpiresDefault A2592000
     <Files *>
        Header set Surrogate-Control 'max-age=259200'
     </Files>
</Directory>
```

Surrogate-Control ヘッダーについての詳細は、『Oracle9*i*AS Web Cache 管理および配置ガイド』を参照してください。

- Expires ヘッダーでタグ付けされた静的ファイルを見分ける方法
  - Netscape を使用して、ブラウザ内にキャッシュされたすべてのファイルをクリーンアップします。
  - Expires ヘッダーでタグ付けされたイメージを持つ Web ページを開きます。各ページでマウスを右クリックし、ポップ・アップ・メニューから「View Info」を選択します。
  - ページ情報のトップ・パネルで、Expires ヘッダーでタグ付けされているイメージを選択します。
  - 下部のパネルに表示されている情報を確認します。Expires ヘッダーが有効な日付に設定されています。このエントリは、No date given である場合、ファイルは Expires ヘッダーでタグ付けされていません。

データベース・セッションの数のチューニング

- Oracle init$SID.ora 構成ファイルの processes パラメータを設定して、Oracle がデータベース・セッションの最大数を処理できるようにする必要があります。この数は、DAD の数、Oracle HTTP Server プロセスの最大数、および Oracle9*i*AS インスタンスの数に比例します。
- 2 リスナー計画または Multi Threaded Server（MTS）を使用すると、データベース・セッションの数を減らすことができます。8-11 ページの「2 リスナー計画」を参照してください。
- Unix プラットフォームでは、Oracle HTTP Server プロセスで接続プールを共有できません。この理由により、このアプリケーションでは DAD をできるだけ少なく使用することをお薦めします。
- Oracle HTTP Server を Oracle9*i*AS Web Cache でフロント・エンドすると、HTTP 構成に対して多くのプロセスが必要なくなり、その結果データベース・セッションの数が少なくなります。

2 リスナー計画

Unix ベースのプラットフォームのように、Oracle HTTP Server がプロセス・ベースであるプラットフォームでは、各プロセスは、サーブレット、PLSQL、静的ファイルおよび CGI を含む、HTTP リクエストのすべてのタイプに使用できます。1 つの Oracle9*i* Application Server リスナー・セットアップでは、各 `httpd` プロセスはデータベースへの独自の接続プールを維持します。データベース・セッションの最大数は、`StartServers`、`MinSpareServers` および `MaxSpareServers` を `httpd.conf` 構成ファイルに設定することで管理され、さらにシステム上でロードされます。この構造では、`mod_plsql` リクエストの数に基づいてデータベース・セッションの数をチューニングできません。`mod_plsql` リクエストの数に基づいてデータベース・セッションの数をチューニングする場合、`mod_plsql` リクエスト専用の独立した HTTP リスナーをインストールします。これにより、`mod_plsql` リクエストの処理に必要なデータベース・セッションの数を大幅に減らすことができます。

たとえば、メインの Oracle9*i*AS リスナーが mylsnr1.mycompany.com のポート 7777 で起動していると仮定します。始めに、別のリスナーを mylsnr2.mycompany.com のポート 8888 にインストールします。次に、mylsnr1.mycompany.com:7777 に対して行われたすべての `mod_plsql` リクエストを mylsnr2.mycompany.com:8888 の 2 番目のリスナーにリダイレクトします。次のステップを確認します。

1. mycompany.com:7777 へのすべての PL/SQL リクエストを mylsnr2.mycompany.com:8888 にリダイレクトするには、次のように構成を変更します。

   a. ポート 7777 で起動している Oracle9*i*AS リスナーに対して、`ORACLE9IAS_HOME/Apache/modplsql/conf/plsql.conf` ファイルを編集します。行の最初に # を付けて、次の行をコメントアウトします。

      ```
      #LoadModule plsql_module...
      ```

   b. `mylsnr1.mycompany.com` の PL/SQL リクエストの処理に使用される DAD の場所を、`mylsnr2.mycompany.com` の `$ORACLE_HOME/Apache/modplsql/conf/dads.conf` 構成ファイルにコピーします。

      行の最初に # を付けて、`mylsnr1.mycompany.com` の DAD 場所の構成パラメータをコメント・アウトします。

      ```
      #<Location /pls/portal>
      #...
      #</Location>
      ```

   c. `dads.conf` に次の行を追加して、この DAD の場所に対するすべての `mod_plsql` リクエストを 2 番目のリスナーに転送します。

      ```
      ProxyPass /pls/portal http://mylsnr2.mycompany.com:8888/pls/portal
      ```

   すべての DAD の場所に対してこの設定の手順を繰り返します。

2. PL/SQL プロシージャはブラウザに表示される URL を生成するため、すべての URL が mylsnr2.mycompany.com:8888 の内部 mod_plsql リスナーを参照せずに構成されていることが重要です。PL/SQL で URL がどのように生成されているかにより、3 つのオプションを選択します。

   - URL がアプリケーションにハードコーディングされている場合、常にそのハードコーディングの値（HOST=mylsnr1.mycompany.com および PORT=7777）を使用して生成されることに注意してください。このシナリオでは変更は必要ありません。
   - PL/SQL アプリケーションが常に CGI 環境変数 SERVER_NAME および SERVER_PORT を使用する場合、mylsnr2.mycompany.com のリスナーの構成を変更する方が簡単です。ファイルを編集し、2 番目のリスナーの ORACLE9IAS_HOME/Apache/Apache/conf/httpd.conf ファイルの ServerName および Port 行を、次のように変更します。

     ```
     ServerName mylsnr1.mycompany.com (was mylsnr2.mycompany.com)
     Port 7777                        (was 8888)
     ```

   - URL が CGI 環境変数 HTTP_HOST を使用して生成されている場合、ポート 8888 で起動する Oracle9*i*AS リスナーの CGI 環境変数をオーバーライドします。各 DAD の ORACLE9IAS_HOME/Apache/modplsql/conf/dads.conf ファイルに次の行を追加し、デフォルトの CGI 環境変数である HOST、SERVER_NAME および SERVER_PORT をオーバーライドします。

     ```
     PlsqlCGIEnvironmentList  SERVER_NAME  mylsnr1.mycompany.com
     PlsqlCGIEnvironmentList  SERVER_PORT  7777
     PlsqlCGIEnvironmentList  HOST mylsnr1.us.oracle.com:7777
     ```

   すべての場合において、アプリケーションに 2 番目のリスナーが存在しないように見せかけて、URL を生成することが目的です。

3. 設定をテストし、すべての DAD に問題なくアクセスできることを確認してください。
4. この設定では、Oracle9*i*AS リスナーの合計のロードに基づいてメインのリスナー mylsnr1.mycompany.com を設定できます。実行される mod_plsql リクエストのみに基づいて mylsnr2.mycompany.com の 2 番目のリスナーをチューニングできます。

オーバーヘッド問題

Portal のストアド・プロシージャの実行時、mod_plsql に Describe オーバーヘッドが発生することがあり、この場合、実行を完了するためにはデータベースに 2 つの追加のラウンドトリップが必要になります。これは、パフォーマンスに影響します。

Describe のオーバーヘッド

PL/SQL プロシージャを実行するために、mod_plsql は渡されるパラメータのデータタイプを知っている必要があります。mod_plsql はこの情報に基づいて各パラメータを配列またはスカラーとしてバインドします。プロシージャ署名を理解するための 1 つの方法は、実行前にそのプロシージャについて記述することです。ただし、このアプローチは、各プロシージャの実行前に記述する必要があるため、効率的ではありません。この記述によるオーバーヘッドを回避するには、mod_plsql は各パラメータ名に対して渡されるパラメータ数を見るようにします。この情報を利用して、各変数のデータタイプを推測します。この論理では単に、1 つの値が渡される際にパラメータをスカラーとし、それ以外の場合は配列とします。この動作はほとんど場合において有効ですが、配列パラメータに 1 つの値を渡す場合、またはスカラーに複数の値を渡す場合には失敗します。このような場合、PL/SQL の実行時の最初の試行に失敗します。mod_plsql は Describe コールを発行して PL/SQL プロシージャの署名を取得し、Describe 操作で取得した情報に基づいて各パラメータをバインドします。プロシージャは再実行され、結果が戻されます。

この Describe コールはプロシージャに対して透過的に発生しますが、mod_plsql は内部的に 2 つの余分なラウンドトリップを実行します。1 つは実行に失敗したコールに対して、もう 1 つは Describe コールに対してです。

Describe のオーバーヘッドの回避

次の方法でパフォーマンスの問題を回避できます。

- フレキシブル・パラメータ渡しを使用します。
- 配列に対して常に複数の値を渡します。1 つの値の場合、プロシージャで無視されるダミー値を渡すことができます。
- 未使用の値をデフォルトとする 2 パラメータ・スタイル・プロシージャを定義している、次の解決策を実行します。

  1. 元のプロシージャを内部的に呼び出すプロシージャに等しいスカラーを定義します。たとえば、元のパッケージは次の例に類似している場合があります。

```
CREATE OR REPLACE PACKAGE testpkg AS
 TYPE myArrayType is TABLE of VARCHAR2(32767) INDEX BY binary_ integer;
 PROCEDURE arrayproc (arr myArrayType);
END testpkg;
/
```

2. /pls/.../testpkg.arrayproc? arr= 1 のような URL コールを作成する場合、仕様を次のように変更します。

```
CREATE OR REPLACE PACKAGE testpkg AS
 TYPE myArrayType is TABLE of VARCHAR2( 32767) INDEX BY binary_integer;
  PROCEDURE arrayproc (arr varchar2);
  PROCEDURE arrayproc (arr myArrayType);
END testpkg;
/
```

3. プロシージャ arrayproc は、次に類似します。

```
CREATE OR REPLACE PACKAGE BODY testpkg AS
PROCEDURE arrayproc (arr varchar2) IS
  localArr myArrayType;
BEGIN
  localArr( 1) := arr;
  arrayproc (localArr);
END arrayproc;
```

フレキシブル・パラメータ渡し（4 パラメータ）のオーバーヘッド

ラウンドトリップ・オーバーヘッドは、PL/SQL プロシージャが古いスタイルの 4 パラメータ・インタフェースを使用している場合に起こります。PL/SQL ゲートウェイは、最初に 2 パラメータ・インタフェースを使用してプロシージャの実行を試みます。これに失敗すると、PL/SQL ゲートウェイは 4 パラメータ・インタフェースを使用します。これは、4 パラメータ・インタフェース・プロシージャが 1 度の余分なラウンドトリップを実行することになります。

- フレキシブル・パラメータ渡しのオーバーヘッドの回避

  このオーバーヘッドを回避するには、対応ラッパーを記述して、2 パラメータ・インタフェースを使用しながら内部的に 4 パラメータ・インタフェース・プロシージャを呼び出すようにすることをお薦めします。その他のオプションでは、元のプロシージャの仕様を変更して、デフォルトとして 2 パラメータ・インタフェースで渡されないパラメータを設定します。4 パラメータ・インタフェースは後方互換性のみを提供しているため、後で問題が発生します。

- フレキシブル・パラメータおよび感嘆符の使用

  Oracle9*i* Application Server のフレキシブル・パラメータ渡しモードでは、PL/SQL プロシージャがプロシージャ名の前に感嘆符を持っていると仮定しています。Oracle9*i*AS で使用されている自動検出メソッドはパフォーマンスに影響するため、現在 Oracle9*i* Application Server でのフレキシブル・パラメータ渡しには感嘆符が必要です。Oracle9*i*AS では、各プロシージャは実行前に完全に記述されます。プロシージャ Describe コールはプロシージャの署名を判断し、データベースへのラウンドトリップを必要とします。エンドユーザーがプロシージャの前に感嘆符を追加して明確にフレキ

シブル・パラメータ渡しの規則を指定することにより、Oracle9*i* Application Server の PL/SQL ゲートウェイではこのラウンドトリップを回避できます。

PL/SQL Web アプリケーションでのキャッシングの使用

キャッシングによって PL/SQL Web アプリケーションのパフォーマンスを向上できます。中間層の PL/SQL プロシージャによって生成された Web コンテンツをキャッシュし、データベースのワークロードを低減できます。

このセクションでは、次のようなキャッシングの方式について説明します。

- 検証方式の使用 - アプリケーションは、ページが最後に提示されてから変更があったかどうかをサーバーに確認します。
- 期限方式の使用 - PL/SQL アプリケーションは特定の時間間隔に基づいてページをキャッシュするか、または再度生成するかを決定します。
- PL/SQL Web アプリケーションでのシステム / ユーザーレベルのキャッシング - 検証方式または期限方式を使用している場合に有効です。キャッシングのレベルは、ページが特定のユーザー対してキャッシュされているか、またはシステム上のすべてのユーザーに対してキャッシュされているかによって決定されます。

これらの方式およびレベル決定は、PL/SQL Web Toolkit 内の `ows_cache` パッケージを使用して行います。

関連項目：『Oracle9*i* Application Server PL/SQL Web Toolkit リファレンス』

検証方式の使用

通常、検証方式では、ページが最後に表示されてから変更があったかどうかをサーバーに確認します。変更されていない場合、キャッシュされたページはユーザーに表示されます。ページが変更されている場合、新しいコピーを取得してユーザーに表示し、それをキャッシュします。

検証方式を使用するための 2 つのメソッド（Last-Modified メソッドおよび Entity Tag メソッド）が存在します。次の 2 つのセクションでは、これらの方式が HTTP プロトコルで使用される方法について示します。PL/SQL ゲートウェイは HTTP プロトコルを使用しませんが、多くの同じ原則が使用されています。

Last-Modified

Web ページが HTTP プロトコルを使用して生成されると、そのページには **Last-Modified** レスポンス・ヘッダーが含まれます。このヘッダーは、リクエストされたコンテンツの日付（サーバーに関連する）を示しています。ブラウザはコンテンツとともにこの日付を保存します。後続のリクエストが Web ページの URL に対して行われる場合、ブラウザは次のことを行います。

1. キャッシュされているバージョンがあるかどうかの確認
2. 日付情報の取得
3. リクエスト・ヘッダー **If-Modified-Since** の生成
4. リクエストのサーバーへの送信

キャッシュが有効化されているサーバーは **If-Modified-Since** ヘッダーを探し、それをコンテンツの日付と比較します。2 つが適合すると、「HTTP/1.1 304 Not Modified」などの HTTP レスポンス・ステータス・ヘッダーが生成され、コンテンツはストリームされません。このステータス・コードを受け取った後、キャッシュ・エントリが有効化されているため、ブラウザはキャッシュ・エントリを再利用できます。

2 つが適合しない場合、「HTTP/1.1 200 OK」などの HTTP レスポンス・ヘッダーが生成され、新しい **Last-Modified Response** ヘッダーと共に新しいコンテンツがストリームされます。このステータス・コードを受け取ると、ブラウザはキャッシュ・エントリを新しいコンテンツおよび日付情報で置き換える必要があります。

Entity Tag メソッド

HTTP プロトコルで提供されているもう 1 つの検証メソッドは、**ETag**（Entity Tag）レスポンスおよびリクエスト・ヘッダーです。このヘッダーの値はブラウザに対してあいまいな文字列です。サーバーはこの文字列をアプリケーションのタイプに基づいて生成します。これは、日付値のみを含めることができる **If-Modified-Since** ヘッダーよりもさらに一般的な検証メソッドです。

ETag メソッドは Last-Modified メソッドの動作と非常に類似しています。サーバーは ETag をレスポンス・ヘッダーの一部として生成します。ブラウザはこのあいまいなヘッダー値を、ストリーム・バックされたコンテンツと共に保存します。このコンテンツに対する次のリクエストが行われると、ブラウザはサーバーに保存するあいまいな値とともに **If-Match** ヘッダーを渡します。サーバーがこのあいまい値を生成することにより、何をブラウザに戻すかを決定できます。それ以外は、上で説明されている **Last-Modified** 検証メソッドとまったく同じです。

mod_plsql の検証方式の使用

HTTP 検証キャッシングをフレームワークとして使用する場合の mod_plsql の検証モデルは次のとおりです。

サービスされるコンテンツを制御する PL/SQL アプリケーションは、このタイプのキャッシングを使用します。この方式では、いくつかの適度なパフォーマンスのメリットがあります。これについての 1 つの例は、常時変更可能な動的コンテンツに使用されるアプリケーションです。この場合、アプリケーションはサービス対象のものに対する完全な制御が必要です。検証キャッシングでは、常にアプリケーションに対して、キャッシュされたコンテンツが古いものか、またはブラウザに戻された後のものであるかを尋ねます。

図 8-1 では、mod_plsql に対する検証方式の使用について図示しています。

1. Oracle HTTP Server は PL/SQL プロシージャのリクエストをクライアント・サーバーから受信します。Oracle HTTP Server はそのリクエストを mod_plsql にルーティングします。
2. mod_plsql はリクエストを準備します。
3. mod_plsql は PL/SQL プロシージャをアプリケーションで呼び出します。mod_plsql は通常の Common Gateway Interface（CGI）環境変数をアプリケーションに渡します。
4. PL/SQL プロシージャは送り戻すためにコンテンツを生成します。PL/SQL プロシージャが生成されたコンテンツをキャッシュ可能であると判断した場合、PL/SQL Web Toolkit から owa_cache を呼び出して、タグおよびキャッシュ・レベルを設定します。

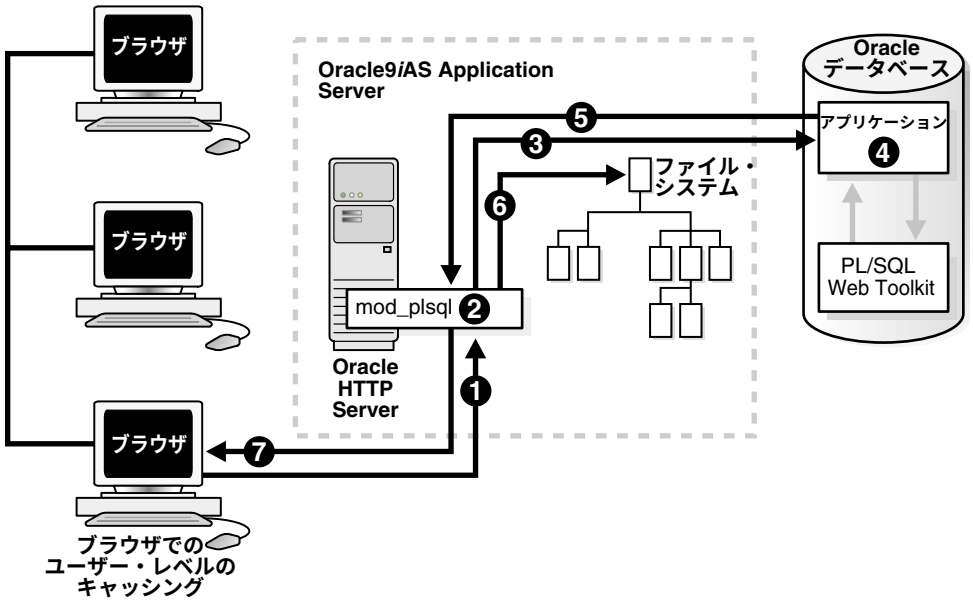
   ```
   owa_cache.set_cache(p_etag, p_level);
   ```

表 8-3 検証モデル・パラメータ

パラメータ	説明
set_cache プロシージャ	ストリーム・バックされたコンテンツがキャッシュ可能であることを mod_plsql に通知するためのヘッダーを設定する。その後、コンテンツがブラウザにストリーム・バックされる際、mod_plsql は、タグおよびキャッシング・レベル情報とともにコンテンツをローカル・ファイル・システムにキャッシュする。
p_etag	プロシージャがコンテンツタグをタグ付けするために生成する文字列。
p_level	キャッシング・レベル：システム・レベルには SYSTEM、ユーザー・レベルには USER。

5. HTML は mod_plsql に戻されます。
6. mod_plsql は、次のリクエストに備えてキャッシュ可能なコンテンツをファイル・システムに保存します。
7. Oracle HTTP Server はクライアントのブラウザにレスポンスを送信します。

図 8-1 検証方式



検証方式を使用した 2 番目のリクエスト

mod_plsql に対して検証方式を使用すると、同じ PL/SQL プロシージャに対する 2 番目のリクエストがクライアントのブラウザによって行われます。

図 8-2 では、検証方式を使用した 2 つめのリクエストを図示しています。

1. mod_plsql は、リクエストに対してキャッシュされたコンテンツがあるかどうかを検出します。
2. mod_plsql は、同じタグおよびキャッシング・レベル情報（最初のリクエストからの）を、CGI 環境変数の一部として PL/SQL プロシージャに送信します。
3. PL/SQL プロシージャはこれらのキャッシング CGI 環境変数を使用して、コンテンツが変更されているかどうかを確認します。PL/SQL Web Toolkit からの次の owa_cache ファンクションを呼び出して、これを行います。

   ```
   owa_cache.get_etag;
   owa_cache.get_level;
   ```

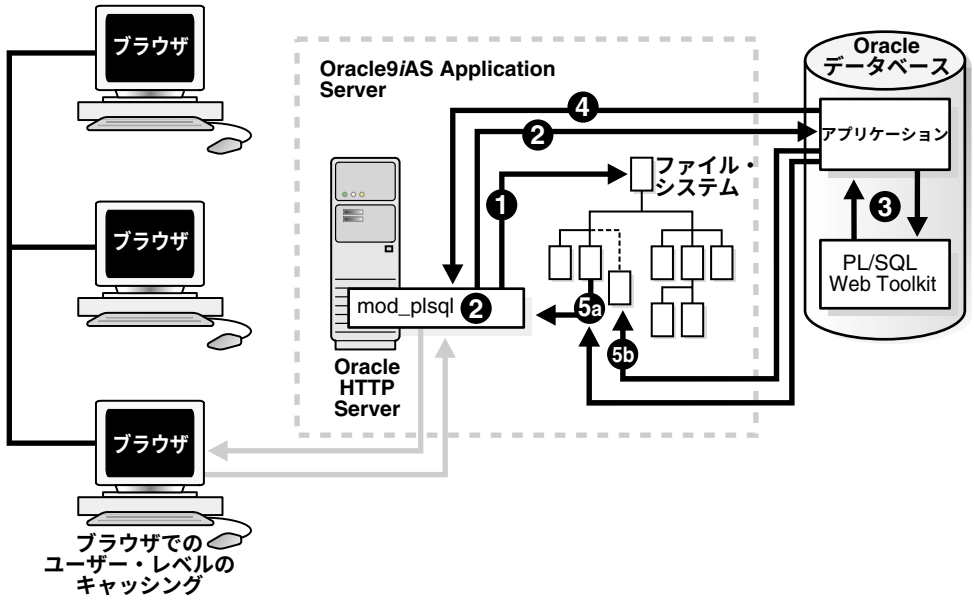
   これらの owa ファンクションは、タグおよびキャッシング・レベルを取得します。
4. アプリケーションはこのキャッシング情報を mod_plsql に送信します。

5. この情報に基づいて、コンテンツの再生成が必要か、またはキャッシュから使用可能かを決定します。

   a. コンテンツがまだ同じであれば、プロシージャは owa_cache.set_not_modified プロシージャを呼び出し、コンテンツを生成しません。これにより、mod_plsql はキャッシュされたコンテンツを使用します。このキャッシュされたコンテンツはブラウザに直接ストリーム・バックされます。

   b. コンテンツが変更されている場合、新しいタグおよびキャッシング・レベルとともに新しいコンテンツを生成します。mod_plsql は古いキャッシュのコピーを新しいものと置き換え、タグおよびキャッシング・レベル情報を更新します。新しく生成されたコンテンツはブラウザに直接ストリーム・バックされます。

図 8-2 検証方式 -2 番目のリクエスト



期限方式の使用

検証モデルでは、mod_plsql は、キャッシュからコンテンツに提供できるかどうかを常に PL/SQL プロシージャに確認します。期限モデルでは、プロシージャはコンテンツの有効期限を事前に設定します。そのため、mod_plsql は、プロシージャに確認することなくキャッシュからコンテンツに提供します。この機能により、データベースとの通信の必要がなくなるため、パフォーマンスが大幅に向上します。

このキャッシング方式では、最良のパフォーマンスを提供します。PL/SQL アプリケーションにおいて古いコンテンツの使用が問題にならない場合に使用します。この一例は、ニュースを日ごとに生成するアプリケーションです。ニュースは 24 時間有効に設定できます。24 時間以内には、キャッシュされたコンテンツはアプリケーションに確認せずに戻されます。これは、根本的にファイルへのサービスと同じです。24 時間後には、mod_plsql は再び新しいコンテンツをアプリケーションから取得します。

キャッシングに期限モデルを使用すること以外、検証モデルで説明したシナリオと同じであると仮定します。

図 8-3 では、mod_plsql に対する期限方式の使用について図示しています。

1. Oracle HTTP Server は PL/SQL サーバー・ページのリクエストをクライアント・サーバーから受信します。Oracle HTTP Server は、そのリクエストを mod_plsql にルーティングします。
2. リクエストは mod_plsql によって Oracle データベースに転送されます。
3. mod_plsql はアプリケーションで PL/SQL プロシージャを呼び出し、通常の Common Gateway Interface（CGI）環境変数をアプリケーションに渡します。
4. PL/SQL プロシージャはコンテンツを送り戻すために生成します。PL/SQL プロシージャが生成されたコンテンツをキャッシュ可能であると判断した場合、PL/SQL Web Toolkit から owa_cache を呼び出して、有効期限およびキャッシュ・レベルを設定します。

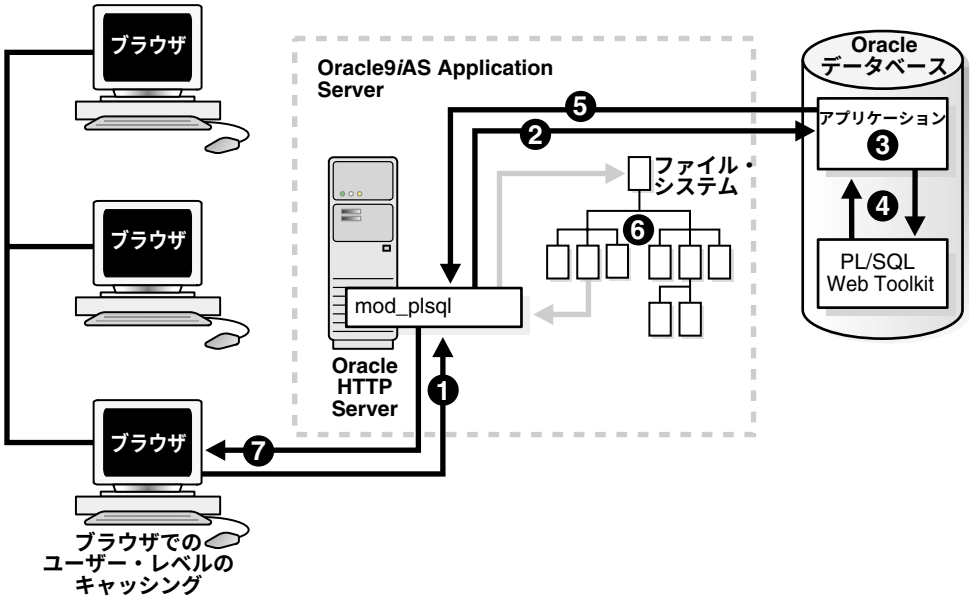
   ```
   owa_cache.set_expires(p_expires, p_level);
   ```

表 8-4　期限モデル・パラメータ

パラメータ	説明
set_expires プロシージャ	期限キャッシングを使用していることを mod_plsql に通知するためのヘッダーを設定する。その後 mod_plsql は、有効期限およびキャッシング・レベル情報とともにコンテンツをファイル・システムにキャッシュする。
p_expires	コンテンツが有効な時間数（分）。
p_level	キャッシング・レベル

5. HTML は mod_plsql に戻されます。
6. mod_plsql は、次のリクエストに備えてキャッシュ可能なコンテンツをファイル・システムに保存します。
7. Oracle HTTP Server はクライアントのブラウザにレスポンスを送信します。

図 8-3 期限方式



期限方式を使用する 2 つめのリクエスト

上で説明されているものと同じ期限モデルを使用すると、同じ PL/SQL プロシージャに対する 2 番目のリクエストがクライアントのブラウザによって行われます。

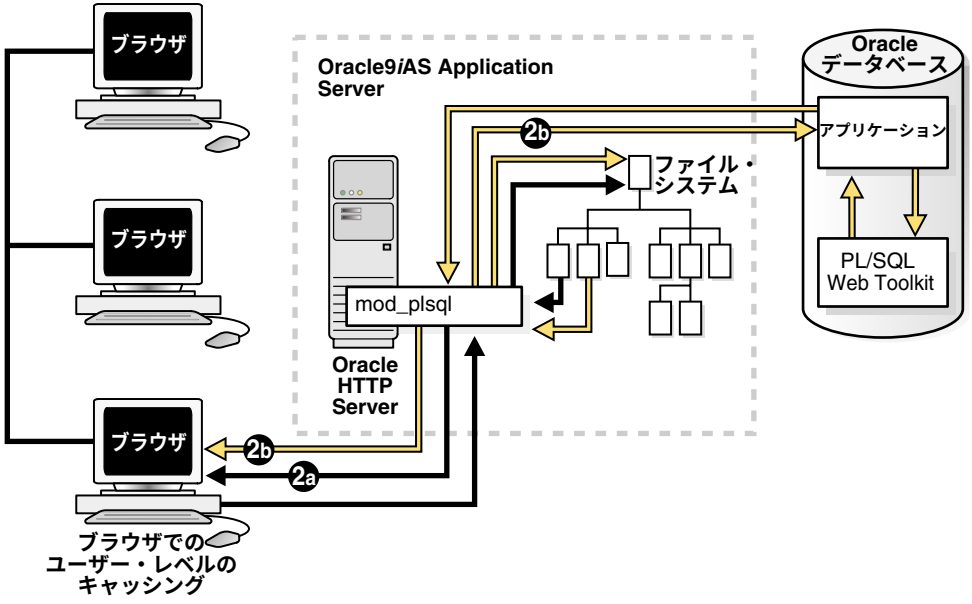
図 8-4 では、期限方式を使用した 2 つめのリクエストを図示しています。

1. mod_plsql は、期限ベースの、キャッシュされたコンテンツのコピーがあるかどうかを検出します。
2. mod_plsql では、現在の時間とキャッシュが作成された時間の差を取得してコンテンツの有効性をチェックします。

a. この差が有効期限内であれば、キャッシュされたコピーはまだ新しいものであるため、データベースとの通信なしに使用されます。このキャッシュされたコンテンツはブラウザに直接ストリーム・バックされます。

b. この差が有効期限内でない場合、キャッシュされたコピーは古くなっています。`mod_plsql` は PL/SQL プロシージャを呼び出して新しいコンテンツを生成します。さらに、このプロシージャは期限ベースのキャッシングを再び使用するかどうかを決定します。これを使用する場合、この新しいコンテンツの有効期限を決定します。新しく生成されたコンテンツはブラウザに直接ストリーム・バックされます。

図 8-4 期限方式 -2 番目のリクエスト



PL/SQL Webアプリケーションでのシステム/ユーザーレベルのキャッシング

PL/SQL プロシージャは、生成されたコンテンツがシステムレベルかユーザーレベルかを判断します。これにより、複数のユーザーが同じコンテンツを参照している場合、PL/SQL ゲートウェイ・キャッシュによる重複ファイルの保存をできるだけ少なくできます。これは次によって決定されます。

- **システムレベル**・コンテンツでは、プロシージャは文字列 SYSTEM をキャッシング・レベル・パラメータとして owa_cache ファンクションに渡します（検証モデルには set_cache、期限モデルには set_expires）。これはキャッシュを共有するすべてのユーザーに対して行われます。

  システムレベル・キャッシングを使用すと、ファイル・システム上のスペースとシステム上のすべてのユーザーが費やす時間の両方をセーブできます。これについての一例は、アプリケーションを使用するすべてのユーザーを対象にしたコンテンツを生成するアプリケーションです。システムレベル設定でコンテンツをキャッシングすることにより、コンテンツの 1 つのコピーがファイル・システムにキャッシュされます。さらに、コンテンツがキャッシュからのディレクトリにサーブされているため、そのシステムの各ユーザーは利益を得ることができます。

- **ユーザーレベル**・コンテンツに対しては、文字列 USER をキャッシング・レベルのパラメータとして渡します。ログインしている特定のユーザーに対して行われます。保存されたキャッシュはそのユーザー独自のものです。そのユーザーのみがキャッシュを使用できます。ユーザーのタイプは認証モードにより決定されます。異なるユーザーのタイプについては、次の表を参照してください。

表 8-5 認証モードによって決定されるユーザーのタイプ

認証モード	ユーザーのタイプ
Single Sign On（SSO）	軽量ユーザー
Basic	データベース・ユーザー
Custom	リモート・ユーザー

たとえば、PL/SQL Web アプリケーションをカスタマイズしているユーザーが存在しない場合、出力をシステムレベルのキャッシュに保存できます。システムの各ユーザーに 1 つのコピーのみが存在します。キャッシュは複数のユーザーによって使用されるため、ユーザー情報は使用されません。

ただし、ユーザーがアプリケーションをカスタマイズしている場合、ユーザーレベル・キャッシングはそのユーザー専用に保存されます。すべての他のユーザーはシステム・レベル・キャッシュを使用します。ユーザーレベル・キャッシュ・ヒットおいては、ユーザー情報が基準になります。ユーザーレベルのキャッシュは、常にシステム・レベルのキャッシュをオーバーライドします。

PL/SQL Web Toolkit ファンクション（owa_cache パッケージ）

検証方式または期限方式のどちらを使用するかによって、呼び出す owa_cache のファンクションが決定します。

owa_cache パッケージは、特別なキャッシング・ヘッダーおよび環境変数を設定し、取得するためのプロシージャを含んでいます。これを使用すると、開発者は PL/SQL Gateway キャッシュをより簡単に行うことができます。このパッケージは常にデータベースにインストールされます。

次のファンクションは、呼び出されるプライマリ・ファンクションです。

表 8-6 プライマリ owa_cache ファンクション

owa ファンクション	目的
owa_cache.set_cache (p_etag IN varchar2, p_level IN varchar2)	検証モデル - ヘッダーを設定。 ■ p_etag パラメータは生成されたコンテンツをタグ付け ■ p_level パラメータは使用するキャッシングのレベル
owa_cache.set_not_modified	検証モデル - キャッシュされたコンテンツを使用するように mod_plsql に通知するためのヘッダーを設定する。検証ベースのキャッシュがヒットする場合のみ使用。
owa_cache.get_level	検証モデル - キャッシング・レベル USER または SYSTEM を取得する。キャッシュがヒットしない場合は NULL を返す。
owa_cache.get_etag	検証モデル - キャッシュされたコンテンツに関連するタグを取得する。キャッシュがヒットしない場合は NULL を返す。
owa_cache.set_expires(p_expires IN number, p_level IN varchar2)	期限モデル - ヘッダーを設定。 ■ p_expires パラメータはコンテンツが有効な時間数（分）。 ■ p_level パラメータは使用するキャッシングのレベル。

その他の Oracle HTTP Server ディレクティブ

表 8-7 では、設定に対して適切にチューニングされる必要のある Oracle HTTP Server ディレクティブをリストしています。表 8-7 にリストされているディレクティブの設定を、システムに適した値に調整します。

表 8-7 デフォルト設定

ディレクティブ	デフォルト値
KeepAlive	On
KeepAliveTimeout	15（秒）
MaxClients	150
MaxKeepAliveRequests	100
MaxRequestsPerChild	10
MaxSpareServers	10
MinSpareServers	5
StartServers	5

関連項目：

- 第 5 章「Oracle HTTP Server の最適化」
- 『Oracle9*i* Application Server Oracle HTTP Server 管理ガイド』の第 3 章「サーバー・プロセスの管理」および第 4 章「ネットワーク接続の管理」

A

Oracle9*i*AS パフォーマンス・メトリック

この付録には、Oracle9*i*AS のパフォーマンスを分析するときに役立つメトリックの一覧が記載されています。メトリックは、Oracle HTTP Server、Oracle9*i*AS Containers for J2EE（OC4J）および Portal など、いくつかの領域に明確に分かれています。この章の各表には、対応する DMS メトリック表に含まれるメトリックが一覧表示されています。

この付録の内容は次のとおりです。

Oracle HTTP Server メトリック

表 A-1、表 A-2、表 A-3 の各表は、Oracle HTTP Server メトリックを説明したものです。メトリック表の名前は ohs_server です。

表 A-1　HTTP Server メトリック（ohs_server）

メトリック	説明	単位
handle.maxTime	モジュール・ハンドラで費やされた最大時間	usec
handle.minTime	モジュール・ハンドラで費やされた最小時間	usecs
handle.avg	モジュール・ハンドラで費やされた平均時間	usecs
handle.active	現在ハンドル処理フェーズにある子サーバー	threads
handle.time	モジュール・ハンドラで費やされた合計時間	usecs
handle.completed	ハンドル処理フェーズが完了した回数	ops
request.maxTime	HTTP リクエストのサービスに必要な最大時間	usecs
request.minTime	HTTP リクエストのサービスに必要な最小時間	usecs
request.avg	HTTP リクエストのサービスに必要な平均時間	usecs
request.active	現在リクエスト処理フェーズにある子サーバー	threads
request.time	HTTP リクエストのサービスに必要な合計時間	usecs
request.completed	完了した HTTP リクエストの数	ops
connection.maxTime	任意の HTTP 接続のサービスに費やされた最大時間	usecs
connection.minTime	任意の HTTP 接続のサービスに費やされた最小時間	usecs
connection.avg	HTTP 接続のサービスに費やされた平均時間	usecs
connection.active	現在開いている接続の数	threads
connection.time	HTTP 接続のサービスに費やされた合計時間	usecs

集計モジュール・メトリック

表 A-2　Apache/Modules メトリック

メトリック	説明	単位
numMods.value	ロードされたモジュールの数	

HTTP サーバー・モジュール・メトリック

サーバーにロードされた各モジュールに対して 1 セットのメトリックがあります。

メトリック表の名前は ohs_module です。

表 A-3 Apache/Modules/mod_*.c メトリック（ohs_module）

メトリック	説明	単位
decline.count	拒否されたリクエストの数	ops
handle.maxTime	このモジュールに必要な最大時間	usecs
handle.minTime	このモジュールに必要な最小時間	usecs
handle.avg	このモジュールに必要な平均時間	usecs
handle.active	このモジュールによって現在処理されているリクエストの数	requests
handle.time	このモジュールに必要な合計時間	usecs
handle.completed	このモジュールによって処理されるリクエストの数	ops

JVM メトリック

サイトで現在実行中の各 Java プロセス（OC4J または Jserv）に対して 1 セットのメトリックがあります。メトリック表の名前は JVM です。

表 A-4 JVM メトリック（JVM）

メトリック	説明	単位
activeThreadGroups.value	JVM 内のアクティブなスレッド・グループの数	整数
activeThreadGroups.minValue	JVM 内のアクティブなスレッド・グループの最小数	整数
activeThreadGroups.maxValue	JVM 内のアクティブなスレッド・グループの最大数	整数
activeThreads.value	JVM 内のアクティブなスレッドの数	threads
activeThreads.minValue	JVM 内のアクティブなスレッドの最小数	threads
activeThreads.maxValue	JVM 内のアクティブなスレッドの最大数	threads
upTime.value	JVM の稼働時間	msecs
freeMemory.value	JVM 内のヒープの空き領域の量	KB
freeMemory.minValue	JVM 内のヒープの空き領域の最小量	KB
freeMemory.maxValue	JVM 内のヒープの空き領域の最大量	KB

表 A-4 JVM メトリック（JVM）（続き）

メトリック	説明	単位
totalMemory.value	JVM 内のヒープ領域の合計量	KB
totalMemory.minValue	JVM 内のヒープの合計領域の最小量	KB
totalMemory.maxValue	JVM 内のヒープの合計領域の最大量	KB

JDBC メトリック

次の各表は、Oracle9*i*AS で収集された JDBC メトリックを一覧にしたものです。

JDBC ドライバ・メトリック

JDBC ドライバ・メトリックは、各 JVM に 1 セットあります。メトリック表の名前は JDBC_Driver です。

表 A-5 /JDBC/Driver - JDBC_Driver メトリック

メトリック	説明	単位
ConnectionCloseCount.count	閉じられている接続の合計数	ops
ConnectionCreate.active	接続を作成するスレッドの現在の数	ops
ConnectionCreate.avg	接続の作成に費やされた平均時間	msecs
ConnectionCreate.completed	この PhaseEvent が開始して終了した回数	ops
ConnectionCreate.maxTime	接続の作成に費やされた最大時間	msecs
ConnectionCreate.minTime	接続の作成に費やされた最小時間	msecs
ConnectionCreate.time	接続の作成に費やされた時間	msecs
ConnectionOpenCount.count	開いている接続の合計数	ops

JDBC データ・ソース・メトリック

メトリック表の名前は JDBC_DataSource です。

データ・ソース・メトリックは、各データ・ソースに 1 セットあります。

表 A-6 /JDBC/*data-source-name* - JDBC_Data Source メトリック

メトリック	説明	単位
CacheFreeSize.value	接続キャッシュ内の空きスロットの数	
CacheGetConnection.avg	キャッシュから接続を得るために費やされた平均時間	msecs
CacheGetConnection.completed	この PhaseEvent が開始して終了した回数	ops
CacheGetConnection.maxTime	キャッシュから接続を得るために費やされた最大時間	msecs
CacheGetConnection.minTime	キャッシュから接続を得るために費やされた最小時間	msecs
CacheGetConnection.time	キャッシュから接続を得るとき、あるいは得なかったときに費やされた時間	msecs
CacheHit.count	接続に対するリクエストがキャッシュから満たされた回数	
CacheMiss.count	接続に対するリクエストがキャッシュから満たされなかった回数	
CacheSize.value	接続キャッシュの合計サイズ	

JDBC ドライバ固有の接続メトリック

メトリック表の名前は JDBC_Connection です。JDBC 接続メトリックは、各接続に 1 セットあります。

表 A-7 /JDBC/Driver/*CONNECTION* - JDBC ドライバ接続メトリック

メトリック	説明	単位
CreateNewStatement.avg	新規文の作成に費やされた平均時間。	msecs
CreateNewStatement.completed	文に対するリクエストがキャッシュから満たされなかった回数。	ops
CreateNewStatement.maxTime	新規文の作成に費やされた最大時間。	msecs
CreateNewStatement.minTime	新規文の作成に費やされた最小時間。	msecs
CreateNewStatement.time	新規文の作成に費やされた時間。	msecs
CreateStatement.avg	文キャッシュから文を得るために費やされた平均時間。	msecs
CreateStatement.completed	文に対するリクエストがキャッシュから満たされた回数。	ops
CreateStatement.maxTime	文キャッシュから文を得るために費やされた最大時間。	msecs
CreateStatement.minTime	文キャッシュから文を得るために費やされた最小時間。	msecs
CreateStatement.time	文キャッシュから文を得るために費やされた時間。	msecs
LogicalConnection.value	物理接続の場合は、その論理接続を意味する（存在する場合）。	

JDBC データ・ソースに固有の接続メトリック

メトリック表の名前は JDBC_Connection です。JDBC データ・ソースに固有の接続メトリックは、各接続の各データ・ソースに 1 セットあります。

表 A-8 /JDBC/*data-source-name*/*CONNECTION* - JDBC データソース接続メトリック

メトリック	説明	単位
表記 CreateNewStatement.avg	新規文の作成に費やされた平均時間。	msecs
CreateNewStatement.completed	文に対するリクエストがキャッシュから満たされなかった回数。	ops
CreateNewStatement.maxTime	新規文の作成に費やされた最大時間。	msecs
CreateNewStatement.minTime	新規文の作成に費やされた最小時間。	msec
CreateNewStatement.time	新規文の作成に費やされた時間。	msec
CreateStatement.avg	文キャッシュから文を得るために費やされた平均時間。	msec
CreateStatement.completed	文に対するリクエストがキャッシュから満たされた回数。	ops
CreateStatement.maxTime	文キャッシュから文を得るために費やされた最大時間。	msec
CreateStatement.minTime	文キャッシュから文を得るために費やされた最小時間。	msec
CreateStatement.time	文キャッシュから文を得るために費やされた時間。	msec
LogicalConnection.value	物理接続の場合は、その論理接続を意味する（存在する場合）。	

JDBC ドライバ文メトリック

メトリック表の名前は JDBC_Statement です。

文メトリックは、各文の各接続に 1 セットあります。

注意： JDBC 文メトリックは、JDBC 文のキャッシングが有効になっている場合のみ使用できます。

表 A-9 /JDBC/Driver/*CONNECTION*/*STATEMENT* JDBC 文メトリック

メトリック	説明	単位
Execute.time	この文によって、最初のフェッチを含む SQL の実行に費やされた時間	msec
Fetch.time	この文によって他のフェッチに費やされた時間	msec
SQLText.value	実行中の SQL	

JDBC データ・ソース文メトリック

メトリック表の名前は JDBC_Statement です。

文メトリックは、各文の各接続の各データ・ソースに 1 セットあります。

注意： JDBC 文メトリックは、JDBC 文のキャッシングが有効になっている場合のみ使用できます。

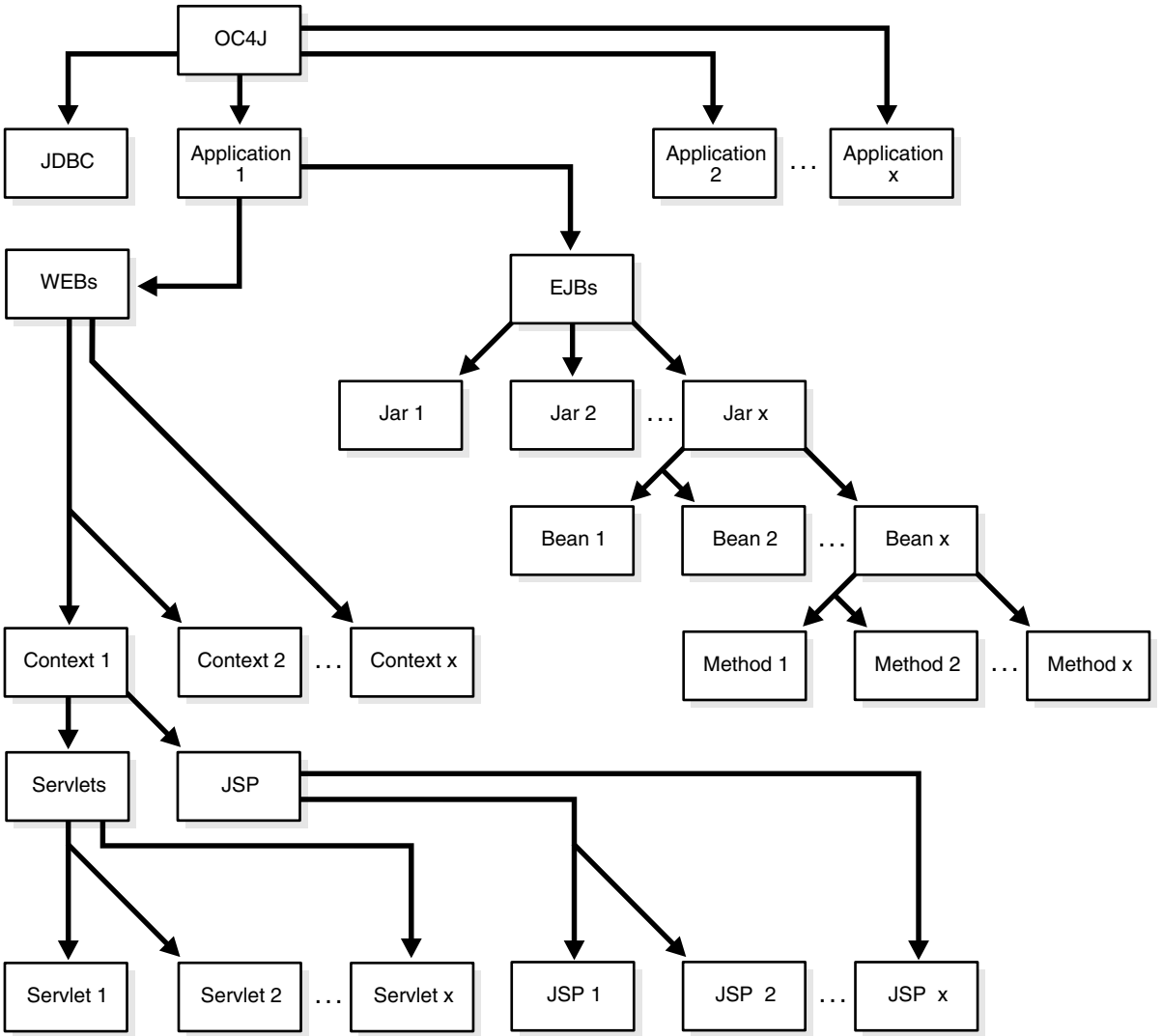
表 A-10 /JDBC/*data-source-name*/*CONNECTION*/*STATEMENT* JDBC 文メトリック

メトリック	説明	単位
Execute.time	この文によって、最初のフェッチを含む SQL の実行に費やされた時間	msec
Fetch.time	この文によって他のフェッチに費やされた時間	msec
SQLText.value	実行中の SQL	

J2EE アプリケーション・メトリック - OC4J メトリック

図 A-1 は、J2EE アプリケーションにおけるメトリックを図示したものです。

図 A-1 J2EE アプリケーションのパフォーマンス・メトリックの構造



Web モジュール・メトリック

各 J2EE アプリケーション内の各 Web モジュールに対して 1 セットのメトリックがあります。

メトリック表の名前は oc4j_web_module です。

表 A-11 OC4J/*application*/WEBs メトリック

メトリック	説明	単位
processRequest.time	このアプリケーションの Web リクエストのサービスに費やされた合計時間。	msec
processRequest.completed	このアプリケーションによって処理された Web リクエストの数。	ops
processRequest.minTime	1 つの Web リクエストのサービスに費やされた最小時間。	msec
processRequest.maxTime	1 つの Web リクエストのサービスに費やされた最大時間。	msec
processRequest.avg	Web リクエストのサービスに費やされた平均時間。	msec
processRequest.active	Web リクエストのサービスを行うスレッドの現在の数。	
resolveContext.time	サーブレット・コンテキストを作成または検索するために費やされた合計時間。各 Web モジュール（WAR）が 1 つのサーブレット・コンテキストにマッピングされます。	msec
resolveContext.completed	完了したコンテキスト解決のカウント。	ops
resolveContext.minTime	サーブレット・コンテキストを作成または検索するために費やされた最小時間。	msec
resolveContext.maxTime	サーブレット・コンテキストを作成または検索するために費やされた最大時間。	msec
resolveContext.avg	サーブレット・コンテキストを作成または検索するために費やされた平均時間。	msec
resolveContext.active	サーブレット・コンテキストの作成または検索を試みているスレッドの現在の数。	
parseRequest.time	ソケットからリクエストを読込みまたは解析するために費やされた合計時間。	msec
parseRequest.completed	解析の済んだ Web リクエストの数。	ops
parseRequest.minTime	リクエストの読込みまたは解析に費やされた最小時間。	msec
parseRequest.maxTime	リクエストの読込みまたは解析に費やされた最大時間。	msec
parseRequest.avg	リクエストの読込みまたは解析に費やされた平均時間。	msec
parseRequest.active	AJP リクエストまたは HTTP リクエストの読込みまたは解析を試みているスレッドの現在の数。	

Web コンテキスト・メトリック

各 J2EE アプリケーション内の各 Web コンテキスト・モジュールに対して 1 セットのメトリックがあります。

メトリック表の名前は oc4j_context です。

表 A-12 OC4J/*application*/WEBs/*context* メトリック

メトリック	説明	単位
resolveServlet.time	サーブレットのインスタンスを（サーブレット・コンテキスト内で）作成または検索するために費やされた合計時間	msec
resolveServlet.completed	OC4J によるサーブレットの参照の合計数	ops
resolveServlet.minTime	サーブレットのインスタンスを（サーブレット・コンテキスト内で）作成または検索するために費やされた最小時間	msec
resolveServlet.maxTime	サーブレットのインスタンスを（サーブレット・コンテキスト内で）作成または検索するために費やされた最大時間	msec
resolveServlet.avg	サーブレットのインスタンスを（サーブレット・コンテキスト内で）作成または検索するために費やされた平均時間	msec
sessionActivation.active	アクティブなセッションの数	ops
sessionActivation.time	セッションがアクティブであった合計時間	msec
sessionActivation.completed	セッションがアクティブになった数	ops
sessionActivation.minTime	セッションがアクティブであった最小時間	ops
sessionActivation.maxTime	セッションがアクティブであった最大時間	msec
sessionActivation.avg	セッション存続時間の平均値	msec
service.time	リクエストのサービスに費やされた合計時間	msec
service.completed	サービスされたリクエストの合計数	ops
service.minTime	リクエストのサービスに費やされた最小時間	msec
service.maxTime	リクエストのサービスに費やされた最大時間	ops
service.avg	サーブレットのサービスに費やされた平均時間	msec
service.active	アクティブなリクエストの現在の数	msec

サーブレット・メトリック

各 J2EE アプリケーション内の各 Web モジュールの各サーブレットに対して 1 セットのメトリックがあります。

メトリック表の名前は oc4j_servlet です。

表 A-13 OC4J/*application*/WEBs/*context* /SERVLETS/*servlet* メトリック

メトリック	説明	単位
service.time	サーブレットの service() コールに費やされた合計時間	msec
service.completed	service() に対するコールの合計数	
service.minTime	サーブレットの service() コールに費やされた最小時間	msec
service.maxTime	サーブレットの service() コールに費やされた最大時間	ops
service.avg	サーブレットのサービスに費やされた平均時間	msec
service.active	このサーブレットのサービスを行うスレッドの現在の数	msec

JSP メトリック

JSP 実行時メトリック

各 J2EE アプリケーションの各 Web コンテキストに対して 1 セットのメトリックがあります。

メトリック表の名前は oc4j_jspExec です。

表 A-14 OC4J/*application*/WEBs/*context* /JSP メトリック

メトリック	説明	単位
processRequest.time	JSP に対するリクエストの処理に費やされた時間。 コンテキストまたはアプリケーションの名前にのみ使用されます。	msec
processRequest.completed	このアプリケーションによって処理された、JSP に対するリクエストの数。	ops
processRequest.minTime	JSP に対するリクエストの処理に費やされた最小時間。	msec
processRequest.maxTime	JSP に対するリクエストの処理に費やされた最大時間。	msec
processRequest.avg	JSP に対するリクエストの処理に費やされた平均時間。	msec
processRequest.active	JSP に対するアクティブなリクエストの現在の数。	

JSP メトリック

各 Web モジュール内の各 JSP に対して 1 セットのメトリックがあります。availableInstance.* に対するデータは、ThreadSafe 以外の JSP に対してのみ表示されます。

メトリック表の名前は oc4j_jsp です。

表 A-15 OC4J/*application*/WEBs/*context* /JSP*jsp_name* メトリック

メトリック	説明	単位
service.time	JSP のサービスを行う時間（つまり、JSP の実際の実行時間）	msec
service.completed	この JSP によって処理された、JSP に対するリクエストの数	ops
service.minTime	JSP のサービスに費やされた最小時間	msec
service.maxTime	JSP のサービスに費やされた最大時間	msec
service.avg	JSP のサービスに費やされた平均時間	msec
service.active	JSP に対するアクティブなリクエストの現在の数。	
availableInstances.value	使用可能な（つまり作成済みの）インスタンスの数。threadsafe=false の場合のみ使用されます。	インスタンス
activeInstances.value	アクティブなインスタンスの数。threadsafe=false の場合のみ使用されます。	インスタンス

EJB メトリック

EJB Bean メトリック

Oracle9*i*AS では、各 J2EE アプリケーション内の各 EJB JAR ファイル内の Bean の各タイプに対し、次のような一連のメトリックが提供されます。

メトリック表の名前は oc4j_ejb_entity_bean です。

表 A-16 OC4J/*application*/EJBs/*ejb-jar-module*/*ejb-name* メトリック

メトリック	説明	単位
transaction-type.value	トランザクションのタイプ。 可能な値 : container または bean。	
session-type.value	セッションのタイプ。 可能な値 : stateful または stateless。	
bean-type.value	Bean のタイプ。 可能な値 : session または entity bean。	

表 A-16 OC4J/*application*/EJBs/*ejb-jar-module*/*ejb-name* メトリック（続き）

メトリック	説明	単位
exclusive-write-access.value	排他的書込みアクセスの値。 可能な値 : true または false。	
isolation.value	独立性の値。 可能な値 : serializable, uncommitted, committed, repeatable_read, none。	
persistence-type.value	永続性のタイプ。 可能な値 : bean または entity bean。	

EJB メソッド・メトリック

EJB Bean の各タイプ内の各メソッドに対し、1 セットのメトリックがあります。

メトリック表の名前は oc4j_ejb_method です。

client.* メトリックでは、メソッドの実際の実装に対する値が表示されます。wrapper.* メトリックでは、メソッドに対して自動的に生成されたラッパーに対する値が表示されます。

> **関連項目：** 自動的に生成されたラッパーの詳細は、『Oracle9*i*AS Containers for J2EE Enterprise JavaBeans 開発者ガイドおよびリファレンス』の第 6 章「高度な EJB のトピック」を参照してください。

表 A-17 OC4J/*application*/EJBs/*ejb-jar-module*/*ejb-name*/*method-name* メトリック

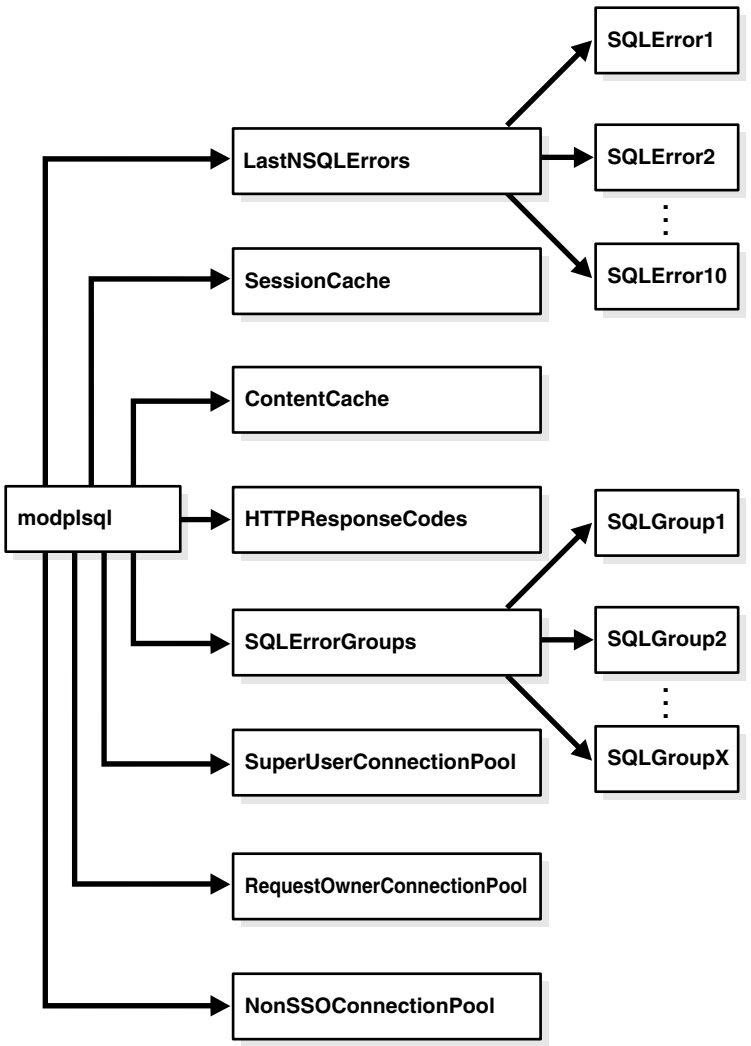
メトリック	説明	単位
client.time	このメソッドの実際の実装の中で費やされた時間。	msec
client.completed	このアプリケーションによって処理された、Beans に対するリクエストの数。	ops
client.minTime	このメソッドの実際の実装の中で費やされた最小時間。	msec
client.maxTime	このメソッドの実際の実装の中で費やされた最大時間。	msec
client.avg	このメソッドの実際の実装の中で費やされた平均時間。	msec
client.active	このメソッドの実際の実装にアクセスするスレッドの現在の数。	
wrapper.time	自動的に生成されたラッパー・メソッドの中で費やされた時間。 **注意 :** すべてのラッパー・メソッドが、実行時に実際の Bean の実装を実行する（たとえば、ステートレス Bean 内でメソッドを作成するなど）とは限りません。そのため、ラッパー・コードに費やされた時間は、Bean の実装に費やされた時間よりも少なくなる可能性があります。	msec
wrapper.completed	このアプリケーションによって処理された、Bean に対するリクエストの数。	ops
wrapper.minTime	自動的に生成されたラッパー・メソッドの中で費やされた最小時間。	msec
wrapper.maxTime	自動的に生成されたラッパー・メソッドの中で費やされた最大時間。	msec
wrapper.avg	自動的に生成されたラッパー・メソッドの中で費やされた平均時間。	msec
wrapper.active	自動的に生成されたラッパー・メソッドにアクセスするスレッドの現在の数。	
trans-attribute.value	トランザクションの属性。可能な値 : NotSupported, Supports, RequiresNew, Mandatory および Never。	
ejbPostCreate.time	ejbPostCreate メソッド（エンティティ Bean）で費やされた時間。	msec
ejbPostCreate.completed	この ejbPostCreate がコールされた回数。	ops
ejbPostCreate.minTime	ejbPostCreate で費やされた最小時間。	msec
ejbPostCreate.maxTime	ejbPostCreate で費やされた最大時間。	msec
ejbPostCreate.avg	ejbPostCreate で費やされた平均時間。	msec
ejbPostCreate.active	ejbPostCreate を実行するスレッドの現在の数。	

Portal メトリック

この項では、Oracle9*i*AS Portal メトリックについて説明します。

図 A-2「mod_plsql メトリック・ツリー」は、mod_plsql メトリックの構造を図示したものです。この項の各表では、関連するメトリックを説明しています。

図 A-2 mod_plsql メトリック・ツリー



/modplsql/HTTPResponseCodes メトリックでは、mod_plsql によって戻される応答コードの一覧が表示されます。

メトリック表の名前は modplsql_HTTPResponseCodes です。このメトリック表には、各 HTTP 応答タイプについて、件数つまり応答が生成された回数を示すメトリックが 1 つ含まれます。

```
[type=modplsql_HTTPResponseCodes]
```

たとえば、http404.count メトリックには、応答コード "HTTP 404: Not found" の件数が保持されます。

表 A-18 は、mod_plsql セッション・キャッシュの一連のメトリックを一覧にしたものです。

メトリック表の名前は modplsql_Cache です。

表 A-18 mod_plsql/SessionCache メトリック

メトリック	説明	単位
cacheStatus.value	キャッシュのステータス。enabled または disabled のどちらかになります。	ステータス
newMisses.count	セッション・キャッシュのミス（新規）の数。	ops
staleMisses.count	セッション・キャッシュのミス（失効）の数。	ops
hits.count	セッション・キャッシュのヒット数。	ops
requests.count	セッション・キャッシュに対するリクエストの数。	ops

表 A-19 は、mod_plsql コンテンツ・キャッシュの一連のメトリックを一覧にしたものです。

メトリック表の名前は modplsql_ContentCache です。

表 A-19 mod_plsql/ContentCache メトリック

メトリック	説明	単位
cacheStatus.value	キャッシュのステータス。enabled または disabled のどちらかになります。	
newMisses.count	コンテンツ・キャッシュのミス（新規）の数。	ops
staleMisses.count	コンテンツ・キャッシュのミス（失効）の数。	ops
hits.count	コンテンツ・キャッシュのヒット数。	ops
requests.count	コンテンツ・キャッシュに対するリクエストの数。	ops

SQLErrorGroups メトリックでは、SQL エラーの定義済のグループ化が表示されます。各グループに対し、表 A-20 のメトリックが記録されます。

メトリック表の名前は modplsql_SQLErrorGroup です。

/modplsql/SQLErrorGroups/*group* [type=modplsql_SQLErrorGroup]

グループは、Oracle SQL エラー・ドキュメントにおけるグループ化に基づきます。たとえばメトリック名 Ora24280Ora29249 は、Ora-24280 ～ Ora-29249 のグループ化を表します。リクエストを実行した結果として発生する各 SQL エラーは、それぞれのエラー・コードに基づいて適切なグループに入ります。同じエラーが頻繁に発生する場合は、Oracle SQL エラー・メッセージ・ドキュメントを使用してエラー・メッセージの詳細を確認し、問題の原因を調査する必要があります。

表 A-20 mod_plsql/SQLErrorGroups メトリック

メトリック	説明	単位
lastErrorDate.value	SQL エラーの原因となった最後のリクエストの日付	日付
lastErrorRequest.value	SQL エラーの原因となった最後のリクエスト	URL
lastErrorText.value	最後のエラーの SQL エラー・テキスト	エラー
error.count	グループ内で発生したエラーの数	ops

LastNSQLErrors 統計では、リクエストの実行中に発生した過去 10 件の SQL エラーが表示されます。エラーはラウンドロビン法で更新されます。各エラーに対して、表 A-21 に示されたメトリックが記録されます。

メトリック表の名前は modplsql_LastNSQLError です。

/modplsql/LastNSQLErrors/<*SQL Error Slot*> [type=modplsql_LastNSQLError]

同じエラーが頻繁に発生する場合は、問題の原因を調査する必要があります。errorText.value メトリックによって示されたエラーの詳細は、『Oracle9*i* データベース・エラー・メッセージ』を参照してください。

表 A-21 mod_plsql/LastNSQLErrors メトリック

メトリック	説明	単位
errorDate.value	リクエストによって SQL エラーが発生した日付	日付
errorRequest.value	SQL エラーの原因となったリクエスト	URL
errorText.value	SQL エラーのテキスト	エラー

表 A-22 は、非 SSO 接続プールの一連のメトリックを一覧にしたものです。

メトリック表の名前は modplsql_DatabaseConnectionPool です。

/modplsql/NonSSOConnectionPool [type=modplsql_DatabaseConnectionPool]

表 A-22 mod_plsql/NonSSOConnectionPool メトリック

メトリック	説明	単位
connFetch.maxTime	プールから接続をフェッチするためにかかる最大時間	usec
connFetch.minTime	プールから接続をフェッチするためにかかる最小時間	usec
connFetch.avg	プールから接続をフェッチするためにかかる平均時間	usec
connFetch.active	現在プール・フェッチ・フェーズにある子サーバー	スレッド
connFetch.time	プールから接続をフェッチするために費やされた合計時間	usec
connFetch.completed	プールから接続がリクエストされた回数	ops
newMisses.count	接続プールのミス（新規）の数	ops
staleMisses.count	接続プールのミス（失効）の数	ops
hits.count	接続プールのヒット数	ops

表 A-23 は、リクエスト所有者接続プールの一連のメトリックを一覧にしたものです。

メトリック表の名前は modplsql_DatabaseConnectionPool です。

/modplsql/RequestOwnerConnectionPool [type=modplsql_DatabaseConnectionPool]

表 A-23 mod_plsql/RequestOwnerConnectionPool メトリック

メトリック	説明	単位
connFetch.maxTime	プールから接続をフェッチするためにかかる最大時間	usec
connFetch.minTime	プールから接続をフェッチするためにかかる最小時間	usec
connFetch.avg	プールから接続をフェッチするためにかかる平均時間	usec
connFetch.active	現在プール・フェッチ・フェーズにある子サーバー	スレッド
connFetch.time	プールから接続をフェッチするために費やされた合計時間	usec
connFetch.completed	プールから接続がリクエストされた回数	ops
newMisses.count	接続プールのミス（新規）の数	ops
staleMisses.count	接続プールのミス（失効）の数	ops
hits.count	接続プールのヒット数	ops

表 A-24 は、スーパー・ユーザー接続プールの一連のメトリックを一覧にしたものです。メトリック表の名前は modplsql_DatabaseConnectionPool です。

/modplsql/SuperUserConnectionPool [type=modplsql_DatabaseConnectionPool]

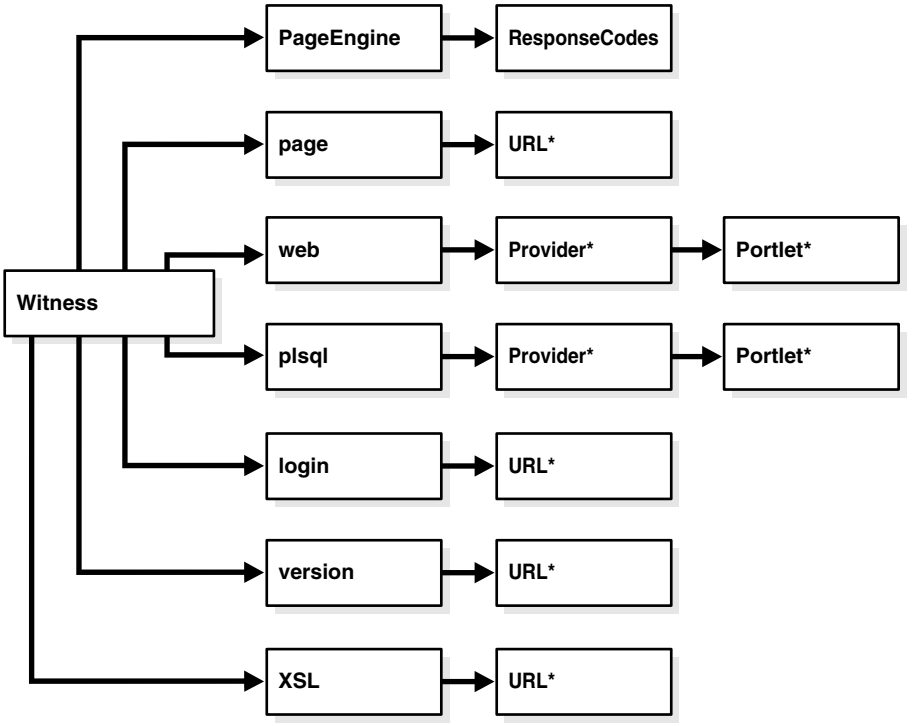
表 A-24 mod_plsql/SuperUserConnectionPool メトリック

メトリック	説明	単位
connFetch.maxTime	プールから接続をフェッチするためにかかる最大時間	usec
connFetch.minTime	プールから接続をフェッチするためにかかる最小時間	usec
connFetch.avg	プールから接続をフェッチするためにかかる平均時間	usec
connFetch.active	現在プール・フェッチ・フェーズにあるスレッド	スレッド
connFetch.time	プールから接続をフェッチするために費やされた合計時間	usec
connFetch.completed	プールから接続がリクエストされた回数	ops
newMisses.count	接続プールのミス（新規）の数	ops
staleMisses.count	接続プールのミス（失効）の数	ops
hits.count	接続プールのヒット数	ops

Parallel Page Engine メトリック

図 A-3「Parallel Page Engine メトリック・ツリー」は、Parallel Page Engine メトリックの構造を示したものです。この項の各表では、関連するメトリックを説明しています。

図 A-3 Parallel Page Engine メトリック・ツリー



これら一連のメトリックは、静的タイプと動的タイプに分けられます。静的タイプとは常に使用可能なメトリックであり、動的タイプとは、特定のポートレットがリクエストされたときのように、特定のイベントが発生したときのみ表示されるメトリックです。PageEngine メトリックおよび ResponseCodes メトリックはすべて静的であり、これ以外のメトリックは動的です。

表 A-25 は、Parallel Page Engine の一連のメトリックを一覧にしたものです。メトリック表の名前は modplsql_PageEngine です。これらのメトリックは、Parallel Page Engine の一般パフォーマンスを表します。キャッシュを使用する場合は、cacheEnabled.value メトリックが 1 に設定されていることを確認する必要があります。このキャッシュを有効にするには、mod_plsql キャッシュおよび Parallel Page Engine 構成のドキュメントを参照してください。

表 A-25 Witness/PageEngine メトリック

メトリック	説明	単位
pageRequests.value	Portal ページに対するリクエストの合計数。	件数
cacheEnabled.value	PPE は mod_plsql によって制御されるときに中間層キャッシュを利用し、JNI レイヤーを介してアクセスされます。このフラグは、この JNI キャッシュが PPE からアクセスされるときに使用可能かどうかを示します。このフラグがゼロになるのは、キャッシュがオフになるように構成されている場合、あるいは JNI 層 DLL のロードに問題があった場合です。	ステータス
cachePageHits.value	キャッシュ可能な完全にアセンブルされたページの中で、キャッシュ・ヒットとなったページに対するリクエストの数。	件数
cachePageRequests.value	キャッシュ可能な完全にアセンブルされたページに対するリクエストの数。	件数
pageMetadataWaitTimeAvg.value	すべてのリクエストに対し、ページ・メタデータを待機する PPE 内部リクエスト・キューに費やされた平均時間。平均を得るには、値メトリックを件数メトリックで割る必要があります。値はすべてのリクエストに対する累積時間、件数は行われたリクエストの数です。	msec
pageMetadataWaitTimeAvg.count	ページ・メタデータに対して行われたリクエストの数。このメトリックは、PPE 内部リクエスト・キューに費やされた平均時間を計算する pageMetadataWaitTimeAvg.value とともに使用する必要があります。	ops
pageMetadataWaitTime.value	PPE 内部リクエスト・キューで、最後のページ・メタデータ・リクエストに費やされた時間。	msec
pageMetadataWaitTime.count	ページ・メタデータに対して行われたリクエストの数。	ops
pageMetadataWaitTime.minValue	ページ・メタデータがリクエストされるまで待機する PPE 内部リクエスト・キューに費やされた最小時間。	msec
pageMetadataWaitTime.maxValue	ページ・メタデータがリクエストされるまで待機する PPE 内部リクエスト・キューに費やされた最大時間。	msec
pageElapsedTimeAvg.value	ページを生成するためにかかる平均時間（ページ・メタデータのフェッチを含む）。平均を得るには、値メトリックを件数メトリックで割る必要があります。値はすべてのリクエストに対する累積時間、件数は行われたリクエストの数です。	msec
pageElapsedTimeAvg.count	生成する必要があった（つまりキャッシュされなかった）ページの数。このメトリックは、ページを生成する平均時間（ページ・メタデータのフェッチを含む）を計算する pageElapsedTimeAvg.value とともに使用する必要があります。	ops
pageElapsedTime.value	リクエストされた最後のページを生成するためにかかる時間（ページ・メタデータのフェッチを含む）。	msec
pageElapsedTime.count	生成する必要があった（つまりキャッシュされなかった）ページの数。	ops
pageElapsedTime.minValue	ページを生成するためにかかる最小時間（ページ・メタデータのフェッチを含む）。	msec

表 A-25 Witness/PageEngine メトリック（続き）

メトリック	説明	単位
pageElapsedTime.maxValue	ページを生成するためにかかる最大時間（ページ・メタデータのフェッチを含む）。	msec
pageMetadataFetchTimeAvg.value	すべてのリクエストに対し、ページ・メタデータをフェッチするためにかかる平均時間。平均を得るには、値メトリックを件数メトリックで割る必要があります。値はすべてのリクエストに対する累積時間、件数は行われたリクエストの数です。	msec
pageMetadataFetchTimeAvg.count	ページ・メタデータに対して行われたリクエストの数。このメトリックは、ページ・メタデータをフェッチする平均時間を計算する pageMetadataFetchTimeAvg.value とともに使用する必要があります。	ops
pageMetadataFetchTime.value	最後のリクエストに対し、ページ・メタデータをフェッチするためにかかる時間。	msec
pageMetadataFetchTime.count	ページ・メタデータに対して行われたリクエストの数。	ops
pageMetadataFetchTime.minValue	ページ・メタデータをフェッチするためにかかる最小時間。	msec
pageMetadataFetchTime.maxValue	ページ・メタデータをフェッチするためにかかる最大時間。	msec
queueTimeout.value	PPE 内部リクエスト・キューでタイムアウトになった Portal データに対するリクエストの数。	msec
queueStayAvg.value	すべての PPE 内部リクエストによって、PPE 内部リクエスト・キューで費やされた平均時間。平均を得るには、値メトリックを件数メトリックで割る必要があります。値はすべてのリクエストに対する累積時間、件数は行われたリクエストの数です。	msec
queueStayAvg.count	PPE 内部リクエスト・キューに追加されたリクエストの数。このメトリックは、PPE 内部リクエスト・キューに費やされた平均時間を計算する queueStayAvg.value とともに使用する必要があります。	ops
queueStay.value	PPE 内部リクエスト・キューで、最後の PPE 内部リクエストに費やされた時間。	msec
queueStay.count	PPE 内部リクエスト・キューに追加されたリクエストの数。	ops
queueStay.minValue	PPE 内部リクエスト・キューで 1 つのリクエストに費やされる最小時間。	msec
queueStay.maxValue	PPE 内部リクエスト・キューで 1 つのリクエストに費やされる最大時間。	msec
queueLengthAvg.value	PPE 内部リクエスト・キューの平均の長さ。平均を得るには、値メトリックを件数メトリックで割る必要があります。	msec
queueLengthAvg.count	PPE 内部リクエストキューに追加されたリクエストの数。このメトリックは、PPE 内部リクエスト・キューの平均の長さを計算する queueLengthAvg.value とともに使用する必要があります。	ops
queueLength.value	PPE 内部リクエスト・キューの現在の長さ。	msec

表 A-25 Witness/PageEngine メトリック（続き）

メトリック	説明	単位
queueLength.count	PPE 内部リクエストキューに追加されたリクエストの数。	ops
queueLength.minValue	PPE 内部リクエスト・キュー内のリクエストの最小数。	msec
queueLength.maxValue	PPE 内部リクエスト・キュー内のリクエストの最大数。	msec

Parallel Page Engine によって作成され、ポートレット、ページ、メタデータなどの内部リクエストによって戻される応答コードに対する一連のメトリックは、メトリック表 modplsql_PageEngine_ResponseCodes に示されています。

この表には、各 HTTP 応答タイプに対する件数が含まれます。

たとえば http100.count には、応答コード HTTP:100 Continue の件数が含まれます。

さらに、メトリック httpUnresolvedRedirect.value にはリダイレクト HTTP 応答コードが戻された後に解決されなかったリクエストの件数が含まれ、httpTimeout.value には PPE 内部リクエスト・キュー内でタイムアウトになったリクエストの件数が含まれます。

表 A-26 は、内部 Parallel Page Engine ページ・メタデータ・リクエストに対する一連のメトリックを一覧にしたものです。メトリック表の名前は動的で、ページ・メタデータのリクエストに使用する URL が含まれます。頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-26 Witness/page/*url* メトリック

メトリック	説明	単位
lastResponseDate.value	レスポンスが最後に行われた日付	日付
lastResponseCode.value	このリクエストに対して戻された最後の応答コード	HTTP 応答コード
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在処理中のスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-27 は、内部 Parallel Page Engine ログイン・メタデータ・リクエストに対する一連のメトリックを一覧にしたものです。メトリック表の名前は動的で、ログイン・メタデータのリクエストに使用する URL が含まれます。頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-27 Witness/login/*url* メトリック

メトリック	説明	単位
lastResponseDate.value	リクエストが最後に行われた日付	日付
lastResponseCode.value	このリクエストに対して戻された最後の応答コード	HTTP 応答コード
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-28 は、内部 Parallel Page Engine Portal バージョン・リクエストに対する一連のメトリックを一覧にしたものです。メトリック表の名前は動的で、Portal レジストリのバージョン・リクエストに使用する URL が含まれます。頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-28 Witness/version/*url* メトリック

メトリック	説明	単位
lastResponseDate.value	リクエストが最後に行われた日付	日付
lastResponseCode.value	このリクエストに対して戻された最後の応答コード	HTTP 応答コード
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec

表 A-28 Witness/version/*url* メトリック（続き）

メトリック	説明	単位
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-29 は、内部 Parallel Page Engine Portal XSL リクエストに対する一連のメトリックを一覧にしたものです。メトリック表の名前は動的で、XSL ドキュメントのリクエストに使用する URL が含まれます。頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-29 Witness/XSL/*url* メトリック

メトリック	説明	単位
lastResponseDate.value	リクエストが最後に行われた日付	日付
lastResponseCode.value	このリクエストに対して戻された最後の応答コード	HTTP 応答コード
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-30 は、内部 Parallel Page Engine PL/SQL プロバイダ・リクエストの一連のメトリックを一覧にしたものです。特定のプロバイダが所有するすべてのリクエスト済ポートレットのメトリック・サマリーが含まれています。メトリック表の名前は動的で、プロバイダ名が含まれます。dad-provider は、指定されたプロバイダの登録およびアクセスに使用する DAD の名前を示します。頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-30 Witness/plsql/*dad-provider* メトリック

メトリック	説明	単位
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
offline.value	プロバイダがオフラインかどうかを示すフラグ。値 1 はプロバイダがオフラインであることを、値 0 はプロバイダがオンラインであることを示します。	状態
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-31 は、内部 Parallel Page Engine Portal PLS/SQL ポートレット・リクエストに対する一連のメトリックを一覧にしたものです。メトリック表の名前は動的で、プロバイダとポートレットの両方の名前が含まれます。表 A-30 には、特定の PL/SQL プロバイダが所有するすべてのリクエスト済ポートレットのメトリック・サマリーが含まれています。

頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-31 Witness/plsql/*dad-provider*/*portlet* メトリック

メトリック	説明	単位
lastResponseDate.value	リクエストが最後に行われた日付	日付
lastResponseCode.value	このリクエストに対して戻された最後の応答コード	HTTP 応答コード
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-32 は、内部 Parallel Page Engine Web プロバイダ・リクエストの一連のメトリックを一覧にしたものです。特定のプロバイダが所有するすべてのリクエスト済ポートレットのメトリック・サマリーが含まれています。メトリック表の名前は動的で、プロバイダ名が含まれます。頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。

表 A-32 Witness/Web/*dad-provider* メトリック

メトリック	説明	単位
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
offline.value	プロバイダがオフラインかどうかを示すフラグ。値 1 はプロバイダがオフラインであることを、値 0 はプロバイダがオンラインであることを示します。	状態
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

表 A-33 は、内部 Parallel Page Engine Portal Web ポートレット・リクエストに対する一連のメトリックを一覧にしたものです。メトリック名は動的で、プロバイダとポートレットの両方の名前が含まれます。表 A-32 には、特定の Web プロバイダが所有するすべてのリクエスト済ポートレットのメトリック・サマリーが含まれています。

頻繁にリクエストが失敗する場合は、HTTPD error_log をチェックし、リクエストが失敗する原因の詳細を確認してください。mod_plsql メトリックでは、その他の詳細が示される場合もあります。HTTP のリダイレクト（302）が頻繁に発生する場合は、リダイレクトを行わないようにポートレットをコーディングすることができます。これによってパフォーマンスが向上します。ポートレットをキャッシュ可能にコーディングしてもキャッシュのヒット数が低い場合は、mod_plsql のキャッシュ設定をチェックし、システムに適切なレベルに設定されていることを確認してください。

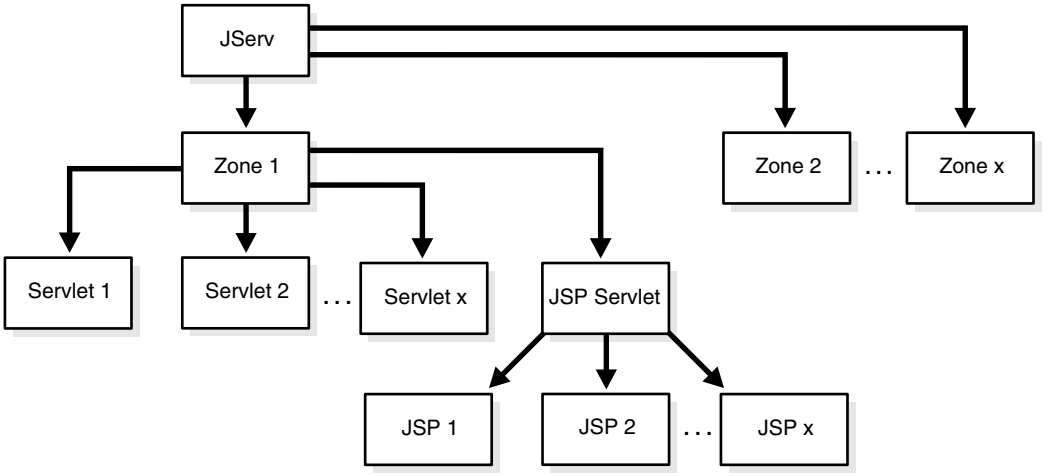
表 A-33 Witness/Web/*dad-provider*/*portlet* メトリック

メトリック	説明	単位
lastResponseDate.value	リクエストが最後に行われた日付	日付
lastResponseCode.value	このリクエストに対して戻された最後の応答コード	HTTP 応答コード
cacheHits.value	このリクエストに対するキャッシュのヒット数	ops
httpXXX.value	このリクエストに対する特定の HTTP 応答コードの件数	ops
executeTime.maxTime	リクエストを行うためにかかる最大時間	usec
executeTime.minTime	リクエストを行うためにかかる最小時間	usec
executeTime.avg	リクエストを行うためにかかる平均時間	usec
executeTime.active	現在リクエストを行うフェーズにあるスレッド	スレッド
executeTime.time	リクエストを行うために費やされた合計時間	usec
connFetch.completed	行われたリクエストの数	ops

JServ メトリック

図 A-4 は、JServ メトリックの構造を図示したものです。次の各表は、関連するメトリックを説明しています。

図 A-4 JServ メトリック・ツリー



Jserv 全体のメトリック

各 JServ サーバー・プロセスに対して 1 セットのメトリックがあります。

メトリック表の名前は jserv_server です。

表 A-34 JServ メトリック・ツリー

メトリック	説明	単位
port.value	この JServ がリスニングする TCP ポートの ID	
readRequest.active	現在 readRequest 処理フェーズにあるスレッド	
readRequest.avg	リクエストの読込みおよび解析にかかる平均時間	msec
readRequest.maxTime	リクエストの読込みおよび解析にかかる最大時間	msec
readRequest.minTime	リクエストの読込みおよび解析にかかる最小時間	msec
readRequest.completed	readRequest 処理フェーズが完了した回数	ops
readRequest.time	リクエストの読込みおよび解析にかかる合計時間	msec
maxConnections.value	JServ プロセス内で同時に処理できるリクエストの数	スレッド
activeConnections.maxValue	同時に処理されるリクエストの最大数	スレッド
activeConnections.value	同時に処理されるリクエストの数	スレッド
idlePeriod.maxTime	プロセスがリクエストを処理しなかった最大時間	msec
idlePeriod.minTime	プロセスがリクエストを処理しなかった最小時間	msec
idlePeriod.completed	リクエストが 1 つもサービスされなかった回数	ops
idlePeriod.time	プロセスがリクエストを処理しなかった合計時間	msec
host.value	この JServ プロセスがバインドされているホスト名および IP アドレス	
maxBacklog.value	この JServ を待機中の OS 内でキューに入っている可能性がある未処理分リクエストの最大数	整数

JServ ゾーン・メトリック

メトリックは各 JServ ゾーンに対して 1 セットあります。

メトリック表の名前は jserv_zone です。

表 A-35 jserv/*zone* メトリック

メトリック	説明	単位
checkReload.active	現在 checkReload 処理フェーズにあるスレッド	整数
checkReload.avg	ゾーンの再ロードが必要かどうかのチェックにかかる平均時間	msec
checkReload.maxTime	ゾーンの再ロードが必要かどうかのチェックにかかる最大時間	msec
checkReload.minTime	ゾーンの再ロードが必要かどうかのチェックにかかる最小時間	msec
checkReload.completed	checkReload 処理フェーズが完了した回数	ops
checkReload.time	ゾーンの再ロードが必要かどうかのチェックにかかる合計時間	msec
activeSessions.value	このゾーン内に存在するセッションの数	セッション
readSession.count	このゾーン内で HttpSession.getValue によってセッション・データが読込まれた回数	ops
writeSession.count	このゾーン内で HttpSession.putValue によってセッション・データが書き込まれた回数	ops
loadFailed.count	リクエストされたアプリケーションのロードに失敗した回数（OJSP に対して機能しない）	ops

JServ サーブレット・メトリック

メトリックは各サーブレットに 1 セットあります。JSP サーブレットは、ゾーン内のすべてのサーブレットおよび JSP に対するすべての集計ロード・メトリックを保持します。

メトリック表の名前は jserv_servlet です。

表 A-36 /jserv/*zone*/*servlet* メトリック

メトリック	説明	単位
processRequest.active	現在 processRequest 処理フェーズにあるスレッド	整数
processRequest.avg	サーブレットを完全に処理するためにかかる平均時間（JServ エンジンのオーバーヘッドを含む）	msec
processRequest.maxTime	サーブレットを完全に処理するためにかかる最大時間（JServ エンジンのオーバーヘッドを含む）	msec
processRequest.minTime	サーブレットを完全に処理するためにかかる最小時間（JServ エンジンのオーバーヘッドを含む）	msec
processRequest.completed	processRequest 処理フェーズが完了した回数	ops
processRequest.time	サーブレットを完全に処理するためにかかる合計時間（JServ エンジンのオーバーヘッドを含む）	msec
serviceRequest.active	現在 serviceRequest 処理フェーズにあるスレッド	整数
serviceRequest.avg	このアプリケーションをインプリメントするサービス・メソッドの平均時間（JServ エンジンのオーバーヘッドを除く）	msec
serviceRequest.maxTime	このアプリケーションを実装するサービス・メソッドの最大時間（JServ エンジンのオーバーヘッドを除く）	msec
serviceRequest.minTime	このアプリケーションを実装するサービス・メソッドの最小時間（JServ エンジンのオーバーヘッドを除く）	msec
serviceRequest.completed	serviceRequest 処理フェーズが完了した回数	ops
serviceRequest.time	このアプリケーションを実装するサービス・メソッドの合計時間（JServ エンジンのオーバーヘッドを除く）	msec
loadServlet.avg	サーブレットのロード（キャッシュまたはファイルから）にかかる平均時間	msec
loadServlet.maxTime	サーブレットのロード（キャッシュまたはファイルから）にかかる最大時間	msec
loadServlet.minTime	サーブレットのロード（キャッシュまたはファイルから）にかかる最小時間	msec
loadServlet.completed	loadServlet 処理フェーズが完了した回数	ops
loadServlet.time	サーブレットのロード（キャッシュまたはファイルから）にかかる合計時間	msec
loadServletClasses.active	現在 loadServletClasses 処理フェーズにあるスレッド	整数

表 A-36 /jserv/*zone*/*servlet* メトリック（続き）

メトリック	説明	単位
loadServletClasses.avg	サーブレット・クラスをファイルからロードするためにかかる平均時間	msec
loadServletClasses.maxTime	サーブレット・クラスをファイルからロードするためにかかる最大時間	msec
loadServletClasses.minTime	サーブレット・クラスをファイルからロードするためにかかる最小時間	msec
loadServletClasses.completed	loadServletClasses 処理フェーズが完了した回数。ほとんどのクラスでは、通常この値は 1 です。	ops
loadServletClasses.time	サーブレット・クラスをファイルからロードするためにかかる合計時間	msec
loadServlet.avg	サーブレットのロード（キャッシュまたはファイルから）にかかる平均時間	msec
createSession.active	現在 createSession 処理フェーズにあるスレッド	
createSession.avg	1 つのセッションを作成するためにかかる平均時間	msec
createSession.maxTime	1 つのセッションを作成するためにかかる最大時間	msec
createSession.minTime	1 つのセッションを作成するためにかかる最小時間	msec
createSession.completed	このアプリケーションに対して作成されたセッションの数が、createSession 処理フェーズによって完了した回数	ops
createSession.time	1 つのセッションを作成するためにかかる合計時間	msec
maxSTMInstances.value	この SingleThreadModel サーブレットに対して使用可能なインスタンスの合計数	整数
activeSTMInstances.maxValue	この SingleThreadModel に対するリクエストを同時にサービスするインスタンスの最大数	整数
activeSTMInstances.value	この SingleThreadModel サーブレットに対して使用可能なインスタンスの合計数	インスタンス

JServ JSP メトリック

メトリックは各 JSP に 1 セットあります。JSP サーブレットは、ゾーン内のすべてのサーブレットおよび JSP に対するすべての集計ロード・メトリックを保持します。

メトリック表の名前は jserv_jsp です。

表 A-37 /jserv/*zone*/*servlet* メトリック

メトリック	説明	単位
processRequest.active	現在 processRequest 処理フェーズにあるスレッド	整数
processRequest.avg	サーブレットを完全に処理するためにかかる平均時間（JServ エンジンのオーバーヘッドを含む）	msec
processRequest.maxTime	サーブレットを完全に処理するためにかかる最大時間（JServ エンジンのオーバーヘッドを含む）	msec
processRequest.minTime	サーブレットを完全に処理するためにかかる最小時間（JServ エンジンのオーバーヘッドを含む）	msec
processRequest.completed	processRequest 処理フェーズが完了した回数	ops
processRequest.time	サーブレットを完全に処理するためにかかる合計時間（JServ エンジンのオーバーヘッドを含む）	msec
serviceRequest.active	現在 serviceRequest 処理フェーズにあるスレッド	整数
serviceRequest.avg	このアプリケーションを実装するサービス・メソッドの平均時間（JServ エンジンのオーバーヘッドを除く）	msec
serviceRequest.maxTime	このアプリケーションを実装するサービス・メソッドの最大時間（JServ エンジンのオーバーヘッドを除く）	msec
serviceRequest.minTime	このアプリケーションを実装するサービス・メソッドの最小時間（JServ エンジンのオーバーヘッドを除く）	msec
serviceRequest.completed	serviceRequest 処理フェーズが完了した回数	ops
serviceRequest.time	このアプリケーションを実装するサービス・メソッドの合計時間（JServ エンジンのオーバーヘッドを除く）	msec
loadServlet.avg	サーブレットのロード（キャッシュまたはファイルから）にかかる平均時間	msec
loadServlet.maxTime	サーブレットのロード（キャッシュまたはファイルから）にかかる最大時間	msec
loadServlet.minTime	サーブレットのロード（キャッシュまたはファイルから）にかかる最小時間	msec
loadServlet.completed	loadServlet 処理フェーズが完了した回数	ops
loadServlet.time	サーブレットのロード（キャッシュまたはファイルから）にかかる合計時間	msec
loadServletClasses.active	現在 loadServletClasses 処理フェーズにあるスレッド	

表 A-37 /jserv/*zone*/*servlet* メトリック（続き）

メトリック	説明	単位
loadServletClasses.avg	サーブレット・クラスをファイルからロードするためにかかる平均時間	msec
loadServletClasses.maxTime	サーブレット・クラスをファイルからロードするためにかかる最大時間	msec
loadServletClasses.minTime	サーブレット・クラスをファイルからロードするためにかかる最小時間	msec
loadServletClasses.completed	loadServletClasses 処理フェーズが完了した回数。ほとんどのクラスでは、通常この値は 1 です。	ops
loadServletClasses.time	サーブレット・クラスをファイルからロードするためにかかる合計時間	msec
loadServlet.avg	サーブレットのロード（キャッシュまたはファイルから）にかかる平均時間	msec
createSession.active	現在 createSession 処理フェーズにあるスレッド	
createSession.avg	セッションを作成するためにかかる平均時間	msec
createSession.maxTime	セッションを作成するためにかかる最大時間	msec
createSession.minTime	セッションを作成するためにかかる最小時間	msec
createSession.completed	このアプリケーションに対して作成されたセッションの数が、createSession 処理フェーズによって完了した回数	ops
createSession.time	セッションを作成するためにかかる合計時間	msec
maxSTMInstances.value	この SingleThreadModel サーブレットに対して使用可能なインスタンスの合計数	
activeSTMInstances.maxValue	この SingleThreadModel に対するリクエストを同時にサービスするインスタンスの最大数	
activeSTMInstances.value	この SingleThreadModel サーブレットに対して使用可能なインスタンスの合計数	インスタンス

索引

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

U

V

W

あ

え

た

ち

て

と

ね

は

ひ

ふ

ほ

み

め

ゆ

よ

れ

ろ