Oracle® Application Server

パフォーマンス・ガイド

10*g* リリース 3（10.1.3.1.0）

部品番号 : B31836-02

2008 年 5 月

ORACLE®

Oracle Application Server パフォーマンス・ガイド , 10*g* リリース 3（10.1.3.1.0）

部品番号 : B31836-02

原本名 : Oracle Application Server Performance Guide, 10*g* Release 3 (10.1.3.1.0)

原本部品番号 : B28942-02

原本著者 : Thomas Van Raalte

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

制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社に所有権のある情報が含まれています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションへの用途を目的としておりません。このプログラムをかかる目的で使用する際、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle、JD Edwards、PeopleSoft、Siebel は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称は、他社の商標の可能性があります。

このプログラムは、第三者の Web サイトへリンクし、第三者のコンテンツ、製品、サービスへアクセスすることがあります。オラクル社およびその関連会社は第三者の Web サイトで提供されるコンテンツについては、一切の責任を負いかねます。当該コンテンツの利用は、お客様の責任になります。第三者の製品またはサービスを購入する場合は、第三者と直接の取引となります。オラクル社およびその関連会社は、第三者の製品およびサービスの品質、契約の履行（製品またはサービスの提供、保証義務を含む）に関しては責任を負いかねます。また、第三者との取引により損失や損害が発生いたしましても、オラクル社およびその関連会社は一切の責任を負いかねます。

目次

B アプリケーションへの DMS のインストルメント

C パフォーマンス・メトリック

D OC4J パフォーマンス・メトリック

はじめに

このガイドでは、Oracle 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」には、次の項が含まれています。

対象読者

『Oracle 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、Oracle 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。

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle HTTP Server 管理者ガイド』
- 『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』
- 『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』
- 『Oracle Containers for J2EE サーブレット開発者ガイド』
- 『Oracle 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』
- 『Oracle Database パフォーマンス・チューニング・ガイド』
- 『Oracle Application Server PL/SQL Web Toolkit リファレンス』

表記規則

本文では、次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連するグラフィカル・ユーザー・インタフェース要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック	イタリックは、特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、パラグラフ内のコマンド、URL、例に記載されているコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

オラクル社カスタマ・サポート・センター

オラクル製品サポートの購入方法、およびオラクル社カスタマ・サポート・センターへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.co.jp/support/
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

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

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://www.oracle.co.jp/education/
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

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

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

1

パフォーマンスの概要

この章では、Oracle 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。

1.1 Oracle Application Server パフォーマンスの概要

Oracle 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。このドキュメントの各章では、パフォーマンス監視に使用するツールと、Oracle HTTP Server、Oracle Containers for J2EE（OC4J）などの Oracle Application Server 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。

1.1.1 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。

待ち時間　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。待ち時間は、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、待ち時間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。

レスポンス時間　リクエストの送信からレスポンスの受信までの時間。

スケーラビリティ　ハードウェア・リソースに十分な余裕さえあれば、これに応じた**スループット**を発揮するシステムの能力。スケーラブルなシステムとは、レスポンス時間および**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、そのリクエストに対するレスポンスの完了までの時間。

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。

1.2 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。

- レスポンス時間
- 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、およびチューニングするシステム部分の判断方法は、1-6 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。

1.2.1 レスポンス時間

レスポンス時間は、**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

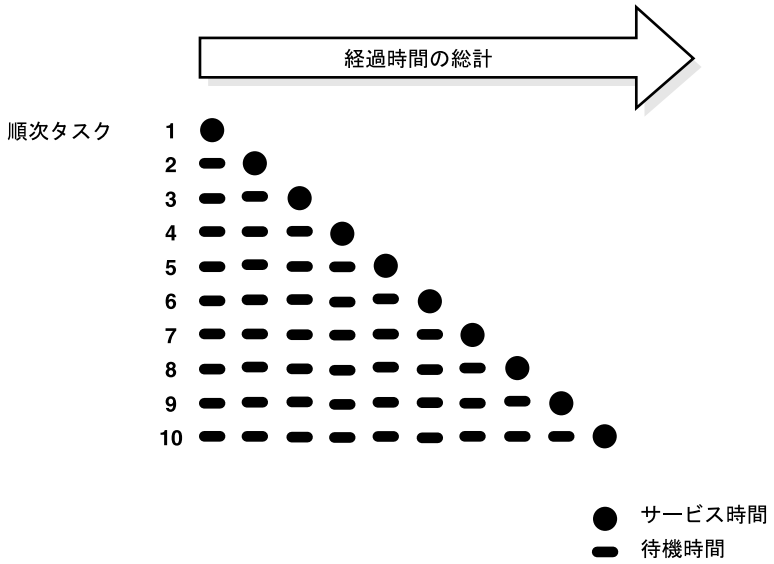


図 1-1 で示す例では、タスク 1 のみが待機時間なしで実行されます。タスク 2 はタスク 1 が完了するまで待機し、タスク 3 はタスク 1 と 2 が完了するまで待機する必要があります。他のタスクについても同様です。この図では各タスクの大きさは同じですが、実際のタスクのサイズはそれぞれ異なります。

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（またはスレッド）に割り当てて処理します。`MaxClients` ディレクティブにより、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、サーバーでは接続が拒否されます。

関連項目： 第 6 章「Oracle HTTP Server の最適化」

1.2.2 システム・スループット

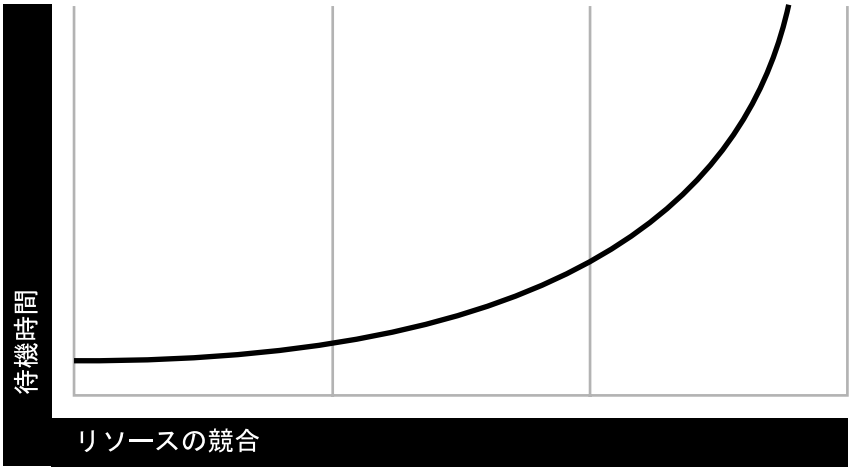
システム・**スループット**は、一定時間内に完了する処理量です。**スループット**は、次の方法で増加できます。

- **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**レスポンス時間**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。

1.2.3 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つれて、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



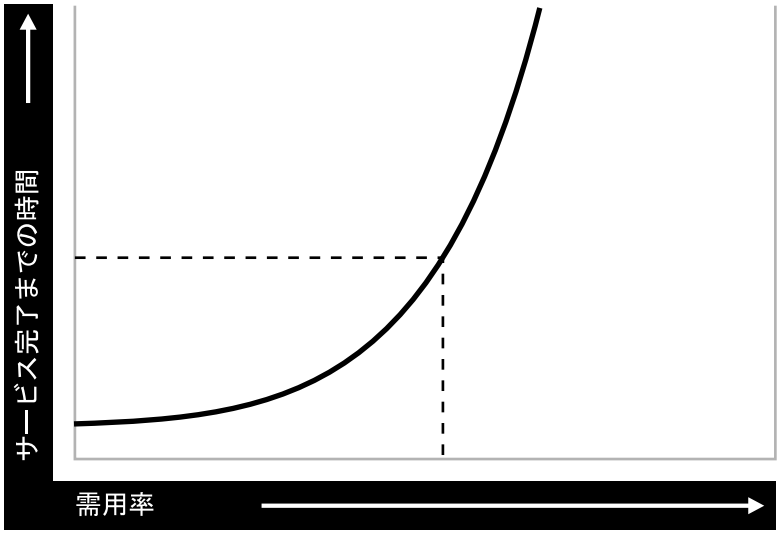
1.2.4 重要なリソース

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、**レスポンス時間**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つれて、サービスにかかる時間が増加していることを示しています。

図 1-3 サービス完了までの時間と需用率



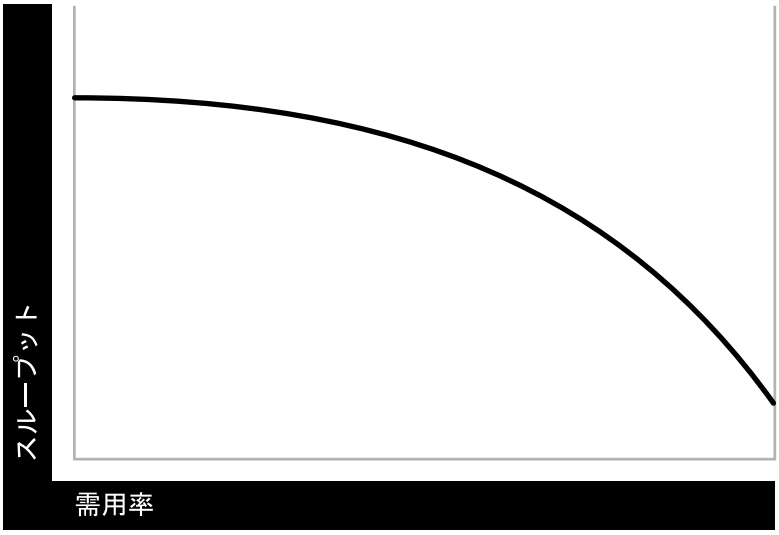
この状況を解消するには、2 つの方法があります。

- 許容範囲のレスポンス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。

1.2.5 過度の需要による影響

過度の需要により、**レスポンス時間**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**の限界を超えた場合、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。

1.2.6 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。

1.3 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**レスポンス時間**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。

1.3.1 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**レスポンス時間**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。

1.3.2 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。

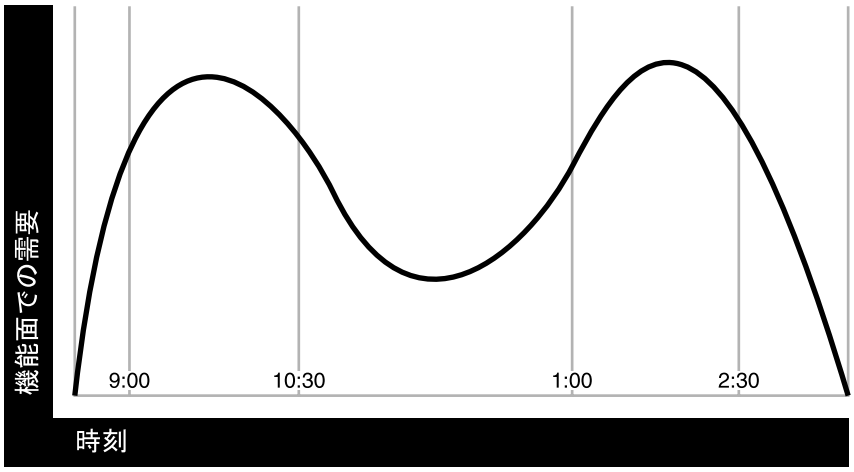
常にパフォーマンスを監視することにより、十分にチューニングされたシステムを維持できます。アプリケーションのパフォーマンスの履歴を記録することにより、有効な比較が可能となります。様々な大きさの負荷に関する実際のリソース消費データを使用して、客観的な**スケーラビリティ**の調査を行うことにより、予期される負荷のボリュームに合せたリソース要件を予測できます。

1.4 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**レスポンス時間**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**レスポンス時間**であるといったことを確認するとします。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レスポンス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。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せてシステムを構成し、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**レスポンス時間**は、ピーク時のユーザーの場合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**レスポンス時間**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



1.4.1 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、構成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**レスポンス時間**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、**レスポンス時間**が目標よりも長い場合の理由の診断

2

Oracle Application Server の監視

この章では、Oracle Application Server とそのコンポーネントの概要とその監視方法について説明します。Oracle Application Server を監視し、パフォーマンス・データを取得すると、システムのチューニング、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デバッグが容易になります。

この章には、次の項が含まれています。

2.1 Oracle Enterprise Manager 10*g* Application Server Control コンソール

Oracle Enterprise Manager 10*g* Application Server Control コンソール（Application Server Control コンソール）を使用して、Oracle Application Server とそのコンポーネントを監視できます。Application Server Control コンソールでは、次のような Oracle Application Server 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 Containers for J2EE（OC4J）および OC4J 下で稼動するアプリケーション

Application Server Control コンソールを使用すると、パフォーマンス・メトリックや他のステータス情報を表示することもできます。

関連項目：『Oracle Application Server 管理者ガイド』

2.2 Oracle Application Server の組込みパフォーマンス・メトリック

Oracle Application Server は自動的にランタイム・パフォーマンスを測定し、Oracle HTTP Server、Oracle Containers for J2EE（OC4J）サーバーおよびコンポーネントのメトリックを収集します。サーバーのパフォーマンス・メトリックは、Oracle Application Server コンポーネントの実装に組み込まれたパフォーマンス・インストルメントを使用して、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（パフォーマンス上の理由から、JDBC メトリックはオプションを設定して有効化します）。

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 コンテナおよびコンポーネントのパフォーマンス監視以外に、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
- OC4J JMS のイベントとステータスの監視

Oracle Application Server 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、リソースの可用性に関する問題特定、あるいはコンポーネントのスループットやレスポンス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て、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。

関連項目：

- 付録 A「組込みパフォーマンス・ツールを使用した監視」
- 付録 C「パフォーマンス・メトリック」

2.3 Oracle Application Server インスタンスの集中管理

Application Server Control コンソールでは、Application Server とそのコンポーネントをスタンドアロンで管理できますが、Oracle Enterprise Manager 10*g* Grid Control コンソールを使用すると、複数の Application Server Control コンソールではなく、1 つのツールですべての Application Server を集中管理できます。たとえば、10 個の Application Server が 5 つのホストに配置されているとします。各ホストに管理エージェントを配置すると、Enterprise Manager がそれらのホスト上にある Application Server を自動的に検出して、デフォルトの監視レベル、通知ルールを使用した監視を自動的に開始します。

Oracle Enterprise Manager 10*g* Grid Control コンソールの Application Server ホーム・ページでは、アプリケーション・サーバー管理者が必要とする次の重要な情報に簡単にアクセスできます。

- Oracle Application Server コンポーネントのホーム・ページへのリンク
- アプリケーション・サーバーのステータス、レスポンス速度およびパフォーマンス・データ
- 問題の特定と解決を迅速に行えるようにする、ドリルダウン可能なアラートや診断
- アプリケーション・サーバーとそのコンポーネントのリソース使用率
- すべての Java 2 Platform, Enterprise Edition（J2EE）アプリケーションおよび Web サービスを表示する単一のビュー
- コンポーネントの起動と停止、構成の変更、およびアプリケーションのデプロイなどの管理作業を行える Application Server Control コンソールへのリンク

関連項目：

Oracle Enterprise Manager 10*g* Grid Control コンソールの詳細は、『Oracle Enterprise Manager 概要』を参照してください。

『Oracle Application Server 管理者ガイド』

『Oracle Enterprise Manager Grid Control インストレーションおよび基本構成』

2.4 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の詳細は、システム・レベルのドキュメントを参照してください。

2.5 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、Oracle Application Server 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。

3

重要なパフォーマンス分野

この章では、Oracle Application Server の重要なチューニング分野について説明します。この章には、次の項が含まれています。

3.1 重要なパフォーマンス分野

この項では、Oracle Application Server の重要なパフォーマンス上の問題について説明するとともに、OC4J 上で動作する J2EE アプリケーションのチューニング方法について説明します。表 3-1 は、Oracle Application Server のパフォーマンスに関するクイック・ガイドです。

表 3-1 Oracle Application Server アプリケーションの重要な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
十分なハードウェア・リソースの確保	Oracle Application Server を使用するには、最小限のハードウェア要件を満たす必要があります。また、データベース層を含め、アプリケーション要件をサポートするハードウェア上で実行する必要があります。Oracle Application Server のインストール要件の詳細は、各プラットフォームのインストレーション・ガイドの「要件」の章を参照してください。 ハードウェア・リソースが十分であるかどうかを調べるのに役立つプラットフォーム固有ツールの詳細は、3-3 ページ の「十分なハードウェア・リソースの確保」を参照してください。
十分な Java ヒープ・サイズの確保	J2EE アプリケーションのパフォーマンスを向上させるには、アプリケーションが実行される JVM に十分なヒープ・サイズを設定する必要があります。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。 JVM ヒープ・サイズのオプションの設定方法の詳細は、3-4 ページの「OC4J 用の十分な Java ヒープの確保」を参照してください。
JVM のガベージ・コレクションのチューニング	J2EE アプリケーションのパフォーマンスを向上させ、同時に、アプリケーションのパフォーマンスが JVM のガベージ・コレクションによって低下しないように構成するには、ヒープ・サイズの管理が必要です。また、ガベージ・コレクションの頻度に影響する JVM オプションの設定が必要になる場合もあります。 ガベージ・コレクションの詳細は、3-4 ページの「JVM のガベージ・コレクション・オプションのチューニング」を参照してください。
データベース接続の再利用	OC4J データソースでは、デフォルトでデータベース接続のプールが有効になっています。そのため、OC4J は、新規接続の再確立が頻繁に行われないように、データベースの接続を管理します。Oracle Application Server のデータソース機能には、データベース接続の維持数や維持時間を制御できるオプションがあります。 3-6 ページの「データベース接続の再利用」を参照してください。
十分な HTTP 接続の指定	HTTP 接続数を指定して、同時実行性のレベルを設定して、Oracle HTTP Server ディレクティブをチューニングします。 3-6 ページの「十分な Oracle HTTP Server 接続の指定」を参照してください。
JDBC 文キャッシュ・オプションの有効化	カーソルを繰返し作成したり、文を繰り返し解析および作成することに起因するオーバーヘッドを軽減するために、文キャッシュを有効にすることで、データベース層を使用する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。 3-8 ページの「データソースに対する文キャッシュの有効化」を参照してください。
データベースのチューニングの適正な実行	アプリケーションがデータベースにアクセスする場合は、アプリケーション要件がサポートされるよう、データベースを適切に構成する必要があります。 3-8 ページの「データベース・チューニングの検証」を参照してください。
ロギング・レベルの検証	ロギング・レベルは、デフォルトのロギング・レベルである「INFO」よりも高く設定しないようにする必要があります。ロギングの設定がデフォルトと一致していない場合は、デフォルトにリセットし、最良のパフォーマンスが得られるようにします。 3-10 ページの「ロギング・レベルの検証」を参照してください。

表 3-1 Oracle Application Server アプリケーションの重要なパフォーマンス分野（続き）

パフォーマンス分野	説明およびリファレンス
EJB インスタンスの再利用	インスタンスの作成と再利用に関連する EJB オプションを設定し、EJB のパフォーマンスを向上させます。 3-10 ページの「EJB インスタンスの再利用」を参照してください。

3.1.1 十分なハードウェア・リソースの確保

ユーザー数とアプリケーション要件をサポートする十分な CPU、メモリーおよびネットワーク・リソースを Oracle Application Server インストール用に確保することは、最も重要なパフォーマンス分野です。リソースの利用状況を一定期間監視して、使用が一時的にピークに達することがあるのか、それともリソースが常に飽和状態にあるのかを調べる必要があります。また、サイトで実行されるアプリケーションに許容できるレスポンス時間とスループットを、ピーク時と平常時の両方で定義する必要があります。さらには、通常の負荷でアプリケーションが実行されているときにシステムをチェックして、CPU、メモリー、ディスク、ネットワーク・パフォーマンスなどのオペレーティング・システム統計を監視し、飽和状態のハードウェア・リソースがあるかどうかを調べます。

CPU、メモリーおよびディスク・パフォーマンスをチェックするには、次のコマンドを使用します。

Linux システムでは、`sar` または `mpstat` コマンドを使用します。

Windows システムでは、`perfmon` コマンドを使用します。

ネットワーク・パフォーマンスをチェックするには、次のコマンドを使用します。

Linux および Windows システム :

```
% netstat
```

Windows システムでは、Windows のタスク マネージャを使用して、ネットワーク・パフォーマンスをチェックすることもできます。

飽和状態になっているハードウェア・リソースが 1 つでもある場合は、次に示す原因の 1 つ以上が該当する可能性があります。

- アプリケーションの実行にハードウェア・リソースが不足している。
- システムが適切に構成されていない。
- アプリケーションまたはデータベースがチューニングされていない。

リソースが常に飽和状態の場合は、負荷を軽減するか、リソースの追加が必要です。通信のピーク時、レスポンス時間が許容範囲を超える場合は、やはりリソースを追加します。または、バッチ処理やバックグラウンド操作を非ピーク時にスケジュール変更するなど、ピーク時の負荷を軽減できるようなスケジュール変更が可能でないかどうかを調べます。Oracle Application Server には、リソースの同時使用を制御する各種メカニズムが用意されており、これによって通信の一時的な急増を制限できます。ただし、常に飽和状態にあるシステムでは、これらのメカニズムは一時的な解決策にしかなりません。

関連項目：

- 「十分な Oracle HTTP Server 接続の指定」（3-6 ページ）
- 「同時実行性の管理および接続の制限」（3-11 ページ）

3.1.2 OC4J 用の十分な Java ヒープの確保

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します。必要なヒープ量はアプリケーションおよび使用可能なメモリー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、メモリーが十分である場合、初期ヒープ・サイズを 512MB 以上に設定することをお薦めします。

初期ヒープ・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。

OC4J インスタンスのヒープ・サイズ値を変更する手順は次のとおりです。

1. OC4J インスタンスのホーム・ページにナビゲートします。
2. 「管理」をクリックします。
3. 必要な場合は、「開く」アイコンをクリックして表の「プロパティ」セクションを開きます。次に、「サーバー・プロパティ」行の「タスクに移動」アイコンをクリックします。
4. 「コマンドライン・オプション」領域にある「**最大ヒープ・サイズ**」および「**初期ヒープ・サイズ**」フィールドの値を変更します。
5. 「適用」をクリックします。
6. 「クラスタ・トポロジ」ページにナビゲートし、変更した OC4J インスタンスを選択して「再起動」をクリックします。「確認」ページで「はい」をクリックします。

これによって、次の項の JVM オプションが指定され、OC4J インスタンス内の OC4J プロセスに割り当てられるヒープ・サイズが変更されます。

Oracle Application Server トポロジ内の同じシステム上に複数の JVM がある場合、パフォーマンスを最大にするには、アプリケーションの要件を満たす最大ヒープ・サイズを設定し、システム上で動作するすべての JVM が消費する合計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します。

関連項目：

- Application Server Control コンソールの「JVM メトリック」ページ。このページにアクセスするには、OC4J ホーム・ページの「**パフォーマンス**」サブタブをクリックし、その「関連リンク」領域にある「**JVM メトリック**」をクリックします。
- JVM ベンダーの次の Web サイトでは、JVM オプションと、それらのパフォーマンスに対する影響の詳細を入手できます。

  ```
  http://java.sun.com/performance/reference/whitepapers/
  5.0_performance.html
  ```

3.1.3 JVM のガベージ・コレクション・オプションのチューニング

JVM のガベージ・コレクションは負荷のかかる操作で、アプリケーションのパフォーマンスに影響します。非効率的なガベージ・コレクションは、アプリケーションのパフォーマンスを大幅に低下させる可能性があります。そのため、アプリケーションでオブジェクトがどのように作成および破棄されるかを理解しておくことが重要です。

JVM のガベージ・コレクション・オプションをチューニングするには、ガベージ・コレクションのデータを解析し、ガベージ・コレクションの頻度とタイプ、メモリー・プールのサイズ、およびガベージ・コレクションに要した時間を確認する必要があります。

アプリケーションに必要なメモリーを調べるには、次のツールを使用して、JVM ガベージ・コレクションとメモリー・プール・サイズを監視します。

- JVM コマンドライン・オプション：

  ```
  -verbose:gc
  -XX:+PrintGCDetails
  ```

  「Full GC」行を調べ、負荷のかかるコレクションの発生頻度を確認します。

- jstat ツール
- visualgc ツール
- Application Server Control コンソールの「JVM メトリック」ページには、JVM メモリー・プールとガベージ・コレクタに関する情報が表示されます。このページにアクセスするには、OC4J ホーム・ページの「**パフォーマンス**」サブタブをクリックし、その「関連リンク」領域にある「**JVM メトリック**」をクリックします。

-XX:+AggressiveHeap JVM オプションを設定して、VM の内部パラメータをチューニングします。また、3-4 ページの「OC4J 用の十分な Java ヒープの確保」の説明に従って、ヒープの合計サイズを増やし、「Full GC」ガベージ・コレクションに起因するオーバーヘッドを軽減します。-XX:+AggressiveHeap オプションを使用すると、VM の内部パラメータが、メモリーを集中的に使用する長時間実行されるワークロードに対して最適化されます。このオプションは、コマンドラインのヒープ・サイズ設定オプションの -Xms および -Xmx の後に指定します。Oracle Application Server 管理環境での Java コマンドライン・オプションの設定方法の詳細は、3-4 ページの「OC4J 用の十分な Java ヒープの確保」を参照してください（-XX:+AggressiveHeap を使用しないよう説明している、Sun 社の古いバージョンのドキュメントは無視してください）。

> **注意：** JMV には、ヒープの管理とガベージ・コレクタの動作をより細かくチューニングできる各種パラメータが用意されています。これらのトピックの詳細は、この後のリンクを参照してください。

アプリケーションでガベージ・コレクション（パフォーマンスを低下させる原因となる）が明示的に使用されているかどうかを調べるには、-XX:+DisableExplicitGC オプションを設定します。このデバッグ・オプションを設定すると、ガベージ・コレクションは明示的に行われなくなり、アプリケーションで system.gc() コールが使用されなくなります。アプリケーションがガベージ・コレクションを明示的に起動していると思われる場合は、このパラメータを設定して、ガベージ・コレクションの動作の違いを観察します。アプリケーションで system.gc() が明示的にコールされていることが判明した場合は、それが起動される理由とコールを無効にした場合の影響についてアプリケーション開発者と検討します。アプリケーション開発者は、ファイナライザを起動する目的で system.gc() コールを使用している場合があります。この方法は推奨されておらず、想定外の動作の原因となります。

JVM コマンドライン・オプションを変更する手順は次のとおりです。

1. OC4J インスタンスのホーム・ページにナビゲートします。
2. 「管理」をクリックします。
3. 必要な場合は、「開く」アイコンをクリックして表の「プロパティ」セクションを開きます。次に、「サーバー・プロパティ」行の「タスクに移動」アイコンをクリックします。
4. 「コマンドライン・オプション」領域の「オプション」表で、該当するコマンドライン・オプションを変更します。
5. 「適用」をクリックします。
6. 「クラスタ・トポロジ」ページにナビゲートし、変更した OC4J インスタンスを選択して「再起動」をクリックします。「確認」ページで「はい」をクリックします。

関連項目：

- http://java.sun.com/j2se/1.5/pdf/jdk50_ts_guide.pdf
- http://java.sun.com/docs/hotspot/gc5.0/ergo5.html
- http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

3.1.4 データベース接続の再利用

データベース接続の作成と再作成のオーバーヘッドを軽減することでアプリケーションのパフォーマンスを改善するには、接続プールの min-connections 属性を指定して接続プールに維持する最小接続数を設定します。

デフォルトでは、min-connections の値は 0 です。最高のパフォーマンスを得るには、min-connections の値を 0 以外に設定します。min-connections が 0 以外の値に設定されている場合は、その数の接続が維持されます。接続は使用されていないときでも OC4J によって維持され、inactivity-timeout に設定されている値に達してもタイムアウトしません。min-connections 属性を設定しても、OC4J によって起動時に接続が事前作成されるわけではありません。接続プールの初期作成時または再初期化時に作成される接続数を設定するには、initial-limit 属性を指定します。initial-limit 属性は min-connections 属性と同じ値に設定することをお薦めします。

min-connections の設定値が max-connections よりも小さい場合は、inactivity-timeout を設定して、非アクティブな接続状態が妥当な時間継続した場合にのみ接続がタイムアウトするようにする必要があります。接続プールの inactivity-timeout には、接続が閉じられるまで未使用の接続をキャッシュする時間を秒単位で指定します。

パフォーマンスを向上させるには、接続プールの inactivity-timeout を、J2EE アプリケーションの実行中に接続の切断や再取得が生じないような値に設定します。inactivity-timeout のデフォルト値は 60 秒です。通常、この値は頻繁にアクセスされるアプリケーションには短すぎるため、リクエスト間に非アクティブな状態が生じる可能性があります。ほとんどのアプリケーションのパフォーマンス向上には、inactivity-timeout の値として、120 秒の設定をお薦めします。

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。

データベース接続の再利用に関する注意事項 :

- 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。次に説明する値を超える接続数をサポートするようにデータベースが構成されていることを、データベース管理者に確認してください。

  同時にアクティブになる可能性があるすべての接続プールの min-connections に対する設定値の合計、およびデータベースの全データソースに対する必要最大同時実行数。

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

  指定された接続が開かれている場合は、OC4J を停止するかリフレッシュ操作を実行して、接続を閉じる必要があります。リフレッシュ機能は、Application Server Control の「JDBC リソース」ページの「接続プール」領域にあります。リフレッシュ操作を実行するには、「**接続プールのリフレッシュ**」フィールドにあるアイコンをクリックします。

3.1.5 十分な Oracle HTTP Server 接続の指定

Oracle HTTP Server 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。

Windows では、類似パラメータとして ThreadsPerChild があります。Oc4jCacheSize ディレクティブは、mod_oc4j が OC4J JVM ごとに維持するアイドル接続の最大数を指定します（Windows にのみ該当）。

MaxClients、ThreadsPerChild および Oc4jCacheSize ディレクティブを使用して、Oracle HTTP Server から OC4J インスタンスへの着信接続を制限できます。この項には、次の項目が含まれています。

- MaxClients ディレクティブの構成（UNIX）
- ThreadsPerChild ディレクティブの構成（Windows）
- Oc4jCacheSize ディレクティブの構成

注意： この項の説明は、Oracle Application Server に付属するデフォルトの Oracle HTTP Server（Apache 1.3 準拠）にのみ該当します。Apache 2.0 ベースのスタンドアロン・バージョンの Oracle HTTP Server には該当しません。

3.1.5.1 MaxClients ディレクティブの構成（UNIX）

MaxClients ディレクティブは、httpd.conf ファイルで構成でき、最大値は 8K です（デフォルト値は 150）。システム・リソースが飽和状態でなく、HTTP の同時接続ユーザー数が 150 を超えている場合、MaxClients を増やしサーバーの同時実行性を向上させると、パフォーマンスを改善できます。システムの利用状況がフル（目安は 85%）になるまで、MaxClients を増やします。

システム・リソースが飽和状態のときに MaxClients を増やしても、パフォーマンスは改善されません。この場合、サーバーへの同時リクエスト数に対するスロットルとして、MaxClients 設定を小さくします。

サーバーで永続的な接続を処理している場合は、アクティブな接続とアイドルな接続の両方を処理する十分な同時 httpd サーバー・プロセスが必要になります。システムの同時実行性に対するスロットルとして MaxClients を指定するとき、アイドル状態の永続的な httpd 接続も httpd プロセスを消費することを考慮に入れる必要があります。つまり、接続数には、現在アクティブな永続的接続および非永続的接続と、アイドルな永続的接続が含まれます。永続的（KeepAlive）な HTTP 接続は、リクエストの処理中でないときでも、接続期間中は、httpd 子プロセス、つまりスレッドを消費します。

十分なリソースがある場合は、KeepAlive を有効にすることをお薦めします。永続的な接続を使用することで、パフォーマンスが向上し、HTTP 接続の再確立による CPU リソースの浪費が回避されます。通常の場合、KeepAlive パラメータの変更は不要です。

注意： 永続的な接続に対するデフォルトの最大リクエスト数は 100 で、これは httpd.conf の MaxKeepAliveRequests ディレクティブで指定されています。デフォルトでは、サーバーは、クライアントからのリクエストを 15 秒待機してから接続を閉じます。これは httpd.conf の KeepAliveTimeout ディレクティブで指定されています。

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。

3.1.5.2 ThreadsPerChild ディレクティブの構成（Windows）

ThreadsPerChild ディレクティブは、httpd.conf ファイルで構成でき、最大値は 8K です（デフォルト値は 150）。Windows システムの ThreadsPerChild パラメータは、UNIX システムの MaxClients パラメータと同様に動作します。

関連項目： 「MaxClients ディレクティブの構成（UNIX）」（3-7 ページ）

3.1.5.3 Oc4jCacheSize ディレクティブの構成

Oc4jCacheSize ディレクティブでは、mod_oc4j が OC4J JVM ごとに維持するアイドル接続の最大数を指定します。Windows でのみ、このディレクティブのデフォルト値を変更すると効果的な場合があります。

UNIX システムでは、Oracle HTTP Server の各プロセスはシングル・スレッドのため、意味のある値はデフォルト値の 1 とゼロ（0）のみです。この値がゼロ（0）の場合、Oracle HTTP Server は接続を維持する必要はなく、リクエストごとに新しい接続を開く必要があることを指定します。各プロセスはシングル・スレッドであるため、複数の接続は不要であり、値が 1 より大きくても UNIX システムに与える影響は値が 1 の場合と同じです。UNIX システムで最高のパフォーマンスを得るには、Oc4jCacheSize のデフォルト値を変更しないでください。

Windows システムでは、デフォルトの Oc4jCacheSize 値は ThreadsPerChild の値の 75% で、その接続キャッシュが子プロセス内のスレッド間で共有されます。Oracle HTTP Server に静的コンテンツと OC4J リクエストの両方の負荷がかかる場合は、このデフォルト値で十分です。ユーザーの負荷がすべての OC4J リクエストである場合、つまり Oracle HTTP Server がコンテンツをほとんど、あるいはまったく提供せず、OC4J のフロント・エンドとしての役割しか果たさない場合、ThreadsPerChild と等しい Oc4jCacheSize を設定することをお薦めします。この設定では、必要に応じてスレッドごとに専用の接続が用意され、パフォーマンスが最高になります。

3.1.6 データソースに対する文キャッシュの有効化

num-cached-statements 属性を 0 よりも大きな値に設定（デフォルト値は 0 で、文キャッシュは無効）することで、カーソルを繰返し作成したり、文を繰り返し解析および作成することに起因するオーバーヘッドを軽減する、文キャッシュを有効にできます。
num-cached-statements に設定する値は、アプリケーションで使用されている SQL 文の数とします。

関連項目：『Oracle Containers for J2EE サービス・ガイド』の「マネージド・データソースでの文キャッシュ」

3.1.7 データベース・チューニングの検証

Oracle Application Server で、データベースを使用するアプリケーションのパフォーマンスを最適にするには、アクセスするデータベース表を設計する際に、パフォーマンスを考慮します。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。

この項には、次の項目が含まれています。

- init.ora データベース・パラメータのチューニング
- REDO ログの配置場所のチューニングとサイズ設定
- 自動セグメント領域管理（ASSM）

関連項目：『Oracle Database パフォーマンス・チューニング・ガイド』

3.1.7.1 init.ora データベース・パラメータのチューニング

表 3-2 に、init.ora データベース初期化パラメータのチューニング情報を示します。

表 3-2 重要な init.ora データベース・チューニング・パラメータ

init.ora パラメータ	説明
DB_BLOCK_SIZE	デフォルトのブロック・サイズは 8K で、ほとんどのシステムに最適です。ただし、OLTP システムではこれよりも小さいブロック・サイズ、DSS システムではこれよりも大きいブロック・サイズのほうが効果的な場合があります。 関連項目：『Oracle Database パフォーマンス・チューニング・ガイド』の表 8-3「ブロック・サイズの長所と短所」
PGA_AGGREGATE_TARGET	インスタンスにアタッチされたすべてのサーバー・プロセスで使用可能な、ターゲットの総 PGA メモリー・サイズを指定します。 関連項目 : PGA メモリー管理の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』の「メモリーの構成と使用方法」を参照してください。
PROCESSES	Oracle に同時接続できるオペレーティング・システム・プロセスの最大数を設定します。このパラメータの値は 6 以上に設定する必要があります（バックグラウンド・プロセスに 5、それぞれのユーザー・プロセスに 1）。たとえば、50 の同時ユーザー数を予定している場合は、このパラメータを少なくとも 55 に設定します。この値から、他の多くの初期化パラメータの値が推測されます。

表 3-2 重要な init.ora データベース・チューニング・パラメータ（続き）

init.ora パラメータ	説明
SGA_MAX_SIZE	このパラメータは、実行されているインスタンスの SGA の最大サイズです。このパラメータを、SGA 専用のメモリー量に設定します。これには、次のメモリー・プールが含まれます。 ■ データベース・バッファ・キャッシュ ■ 共有プール ■ ラージ・プール ■ Java プール バッファ・キャッシュのヒット率を定期的に監視し、バッファ・キャッシュが負荷に対して適切なフレーム数を持つように SGA のサイズを調整してください。バッファ・キャッシュのヒット率は、ビュー V$SYSSTAT のデータから計算されます。また、ビュー V$DB_CACHE_ADVICE からは、バッファ・キャッシュのチューニングに役立つデータが得られます。 関連項目 : V$SYSSTAT および V$DB_CACHE_ADVICE ビューを使用してバッファ・キャッシュのヒット率を最適化する方法を含む、SGA_MAX_SIZE パラメータの設定方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』の「メモリーの構成と使用方法」を参照してください。
SGA_TARGET	このパラメータを 0 でない値に設定すると、自動共有メモリー管理が有効になります。構成の単純化とパフォーマンスの向上のため、自動メモリー管理の使用を強くお薦めします。自動共有メモリー管理は、Oracle Database 10*g* リリース 1（10.1）で導入されました。これ以前のバージョンでは、個々の SGA メモリー・プールを手動で構成する必要があります。 関連項目 : SGA_TARGET パラメータの値の選択の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』の「メモリーの構成と使用方法」、「自動共有メモリー管理」を参照してください。
UNDO_TABLESPACE UNDO_MANAGEMENT	自動 UNDO 管理（UNDO_MANAGEMENT = AUTO）および UNDO_TABLESPACE を使用した UNDO 領域の管理を強くお薦めします。下位互換性の理由から、UNDO_MANAGEMENT のデフォルト値は MANUAL に設定されています。 関連項目 : UNDO 領域管理の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

3.1.7.2 REDO ログの配置場所のチューニングとサイズ設定

データベース I/O のロード・バランシングの管理は重要なタスクです。しかし、REDO ログのオプションをチューニングすることによって Oracle Application Server 環境で実行されているアプリケーションのパフォーマンスを改善したり、場合によっては、REDO ログを別のディスクへ移動することで I/O スループットを大幅に向上させることができます。

データベース・ライター・プロセスとアーカイバ・プロセスの動作が REDO ログのサイズに依存するため、REDO ログ・ファイルのサイズはパフォーマンスにも影響を与える可能性があります。一般的に、REDO ログ・ファイルのサイズが大きいほどチェックポイント・アクティビティが減り、パフォーマンスは向上します。REDO ログ・ファイルについて推奨されるサイズを特定することはできませんが、REDO ログ・ファイルは、100MB から数 GB の範囲に設定することが適切であると考えられます。オンライン REDO ログ・ファイルのサイズは、システムで生成される REDO の大きさに従って設定します。おおよその目安としては、多くても 20 分に 1 回ログ・スイッチを実行します。初期化パラメータ LOG_CHECKPOINTS_TO_ALERT = true を設定して、アラート・ファイルにチェックポイントの時間を書き込みます。

必要な REDO ログ・ファイルの完全なセットは、データベース作成時に作成できます。作成後、REDO ログ・ファイルのサイズは変更できません。ただし、新たに大きなサイズのファイルを後から追加することはできます。その際に元の小さいファイルを削除できます。

> **関連項目：** 『Oracle Database パフォーマンス・チューニング・ガイド』の「パフォーマンスを考慮したデータベースの構成」および「I/O 構成および設計」

3.1.7.3 自動セグメント領域管理（ASSM）

永続表領域には、自動セグメント領域管理を使用することをお薦めします。こうした表領域は、ビットマップ表領域とも呼ばれ、ビットマップ・セグメント領域管理によってローカルで管理されます。

下位互換の理由から、ローカル表領域のデフォルトのセグメント領域管理モードは MANUAL です。

> **関連項目：** 空き領域管理の詳細は、『Oracle Database 概要』を参照してください。表領域に対する自動セグメント領域管理の作成と使用の詳細は、『Oracle Database 管理者ガイド』を参照してください。

3.1.8 ロギング・レベルの検証

アプリケーションとサーバーのロギング・レベルが適正であること、またデバッグ・プロパティや他のアプリケーション・レベルのデバッグ・フラグのレベルが適正、または無効に設定されていることを確認する必要があります。Oracle Application Server OC4J のログ出力レベルを、「INFO」レベルのログ・メッセージに設定します（詳細な診断メッセージが出力される「FINE」や「トレース」などのレベルには設定しないでください）。

Application Server Control コンソールを使用して OC4J コンポーネントのログ出力を構成するには、OC4J ホーム・ページから次の手順を実行します。

1. 「管理」リンクをクリックします。
2. 表の「**プロパティ**」で、「**ログ出力の構成**」のためのタスクをクリックします。
3. 表の「**ログ出力**」で、ルートの「ログ・レベル」を適切な値に設定するか、またはツリーを開いて目的のログ出力に対してロギング・レベルを個別に選択します。
4. 「**適用**」をクリックして、OC4J ランタイムに変更を適用します。

3.1.9 EJB インスタンスの再利用

この項では、インスタンスの作成と再利用によって EJB のパフォーマンスを向上させる EJB チューニング・オプションについて説明します。これらのオプションは OC4J 固有で、すべてのタイプの EJB（ステートフル・セッションの EJB を除く）に適用できます。これらのオプションを構成するには、orion-ejb-jar.xml ファイルで属性を設定します。

min-instances 属性には、インスタンス化またはプールされたまま保持される Bean 実装インスタンスの最小数を指定します。デフォルト値は 0 です。最高のパフォーマンスを得るには、min-instances の値を 0 以外に設定します。min-instances の値が 0 よりも大きい場合は、インスタンスが使用されていないときでも、OC4J によって min-instances の数のインスタンスがプールに保持されます。min-instances を超えるインスタンスは、pool-cache-timeout に指定されたタイムアウトの経過後、プールから削除されます。pool-cache-timeout はキャッシュの有効期限を示し、このタイムアウト・ウィンドウ期間内にアクセスされなかったすべての Bean インスタンスが削除されます。たとえば、デフォルト値の pool-cache-timeout では、60 秒間アクセスされなかったすべての Bean がプールから削除され、アクティブまたは最近使用された Bean がプールに維持されます。

pool-cache-timeout のデフォルト値は 60 秒で、通常、これは頻繁にアクセスされる EJB には低すぎます。pool-cache-timeout が 0 またはマイナスの場合、pool-cache-timeout は無効になり、Bean はプールから削除されません。

パフォーマンスをチューニングするには、pool-cache-timeout を大きな値に設定して、Bean がプールから削除される頻度を小さくします。pool-cache-timeout は、J2EE アプリケーションの実行中にインスタンスの破棄と再作成が生じないような、十分に大きな値に設定する必要があります。

3.2 高度なパフォーマンス分野

この項では、特定の使用方法および環境下で、パフォーマンスの向上に役立つパフォーマンス分野について説明します。

この項には、次の項目が含まれています。

- 同時実行性の管理および接続の制限
- ロード・バランシング
- -XX:AppendRatio オプションの使用（スタンドアロン OC4J）

3.2.1 同時実行性の管理および接続の制限

Oracle Application Server では、特定の使用要件に合せて、システムの複数の層で同時実行性を制限できます。HTTP 接続の制御以外に、製品の他のレベルでも同時実行性を制御でき、様々な使用要件に対応できます。

この項には、次の項目が含まれています。

- OC4J スレッド・プールによる同時実行性の制御
- データソースへの最大接続数の設定
- EJB 使用時の EJB インスタンス数の制御
- リモート EJB クライアント接続の制限

関連項目：「十分な Oracle HTTP Server 接続の指定」（3-6 ページ）

3.2.1.1 OC4J スレッド・プールによる同時実行性の制御

OC4J では、この後に説明するスレッド・プールがデフォルトで作成されます。新規スレッドは必要になった時点で作成され、プールに追加されます。

- **http** スレッド・プール : HTTP および AJP リクエストの処理に加えて、場合によっては RMI リクエスト（`rmi request` スレッド・プールが未構成の場合）および RMI 接続（`rmi connection` スレッド・プールが未構成の場合）を処理するスレッド・プール。デフォルトの動作では、スレッド数は最大 1024 で、要求に応じて作成されます。
- **jca** スレッド・プール : OC4J がデプロイされているリソース・アダプタでの専用使用を目的に予約しているスレッドのプールが含まれます。デフォルトの動作では、スレッド数は最大 1024 で、要求に応じて作成されます。
- **system** スレッド・プール : OC4J の内部スレッドが含まれます。デフォルトの動作では、スレッド数は最大 1024 で、要求に応じて作成されます。

OC4J プロセスが利用または再利用するスレッドは、スレッド・プールによって作成および保持されます。一般的な使用例では、デフォルト構成のスレッド・プール管理で十分です。スレッド・プールにある既存のスレッドを再利用することで、パフォーマンスが向上し、JVM およびオペレーティング・システムに対する負荷が軽減されます。

注意： スレッド・プール管理オプションを使用するには、専門知識が必要です。デフォルトのスレッド・プール構成を変更する場合は、以降の同時実行性が、OC4J プロセス用のすべてのスレッド・プールのスレッドの合計によって決まることに注意する必要があります。

一部のケースでは、スレッド・プールの管理オプションを指定し、デフォルトの動作を変更することでパフォーマンスが向上することがあります。次に、そうした状況を示します。

- Oracle Application Server 10*g* に対するハードウェア・リソースの利用率が最大に近づいている場合。たとえば、サイトのユーザー数は 1000 で、通常 20 の同時リクエストを処理しています。これでサイトのリソースの利用率がフルに近いとします。このサイトでは、ピーク時には 75 から 100 の同時リクエストを処理する必要があります。この場合は、スレッド・プールのスレッドの最大数値を 20 から 25 の範囲に指定すると、全体的な結果が改善される可能性があります（この値を 75 から 100 の範囲に設定しても、このレベルではスレッドの処理に使用可能な追加リソースがないため、ピーク時のパフォーマンスは改善されません）。
- データベースに対するハードウェア・リソースの利用率が最大に近づいているとき、またはデータベースに他の制限に起因するボトルネックがあるときに、OC4J によるスレッド制御を使用してデータベースへの接続を制限する場合（これ以外に、データソースの `max-connections` パラメータを使用してデータベース接続を制限できます）。

次の状況では、スレッド・プールをチューニングしても、パフォーマンスは向上しません。

- システムに最も負荷がかかっているときにシステム上に使用可能なハードウェア・リソースが十分にあり、データベースのロック問題などの既知のソフトウェア問題が他にない場合。これには、Oracle Application Server 10*g* 層とデータベース・システムの両方が含まれます。たとえば、サイトのユーザー数は 1000 で、通常同時リクエストは少数しか発生しないとします。このサイトでは、アプリケーション・サーバーのスレッド・プールをチューニングしてスレッドを制限しても効果は期待できません。スレッド数は同時リクエストの程度によって決められ、この場合は非常に低いためです。
- Oracle Application Server 10*g* またはデータベース・システムの他の領域で、十分な同時実行性の制限がすでに指定されている場合。たとえば、Oracle HTTP Server `MaxClients` ディレクティブによって HTTP サーバー・レベルで同時実行性が制御されている場合や、データソースの `max-connections` 属性によってデータベース接続の同時実行性が制御されている場合。

注意： 着信リクエストの数が物理ハードウェアがサポート可能なリクエスト数よりも常に多い場合は、サイトにある物理リソースを増やすことを検討する必要があります。同様に、ピーク時のレスポンス時間が許容外の場合は、ハードウェア構成を追加してこの問題を解決する必要があります。

関連項目：

- 「十分な Oracle HTTP Server 接続の指定」（3-6 ページ）
- 「データソースへの最大接続数の設定」（3-15 ページ）

3.2.1.1.1 アプリケーション・サーバーのスレッド・プールの制御と使用

この項では、スレッド・プールの構成オプションを管理および使用する方法について説明します。

注意： この項では、Oracle Application Server 10*g* リリース 3（10.1.3.1.0）の OC4J スレッド・プール構成オプションについて説明します。このリリースでは、以前の Oracle Application Server リリースで使用した OC4J スレッド・プール構成オプションもサポートされます。`server.xml` の `<global-thread-pool>` および `<work-manager-thread-pool>` 要素によるスレッド・プールの構成は、最新の方法ではなくなりました。これらの要素は、OC4J 10*g*（10.1.3.1.0）では非推奨です。詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

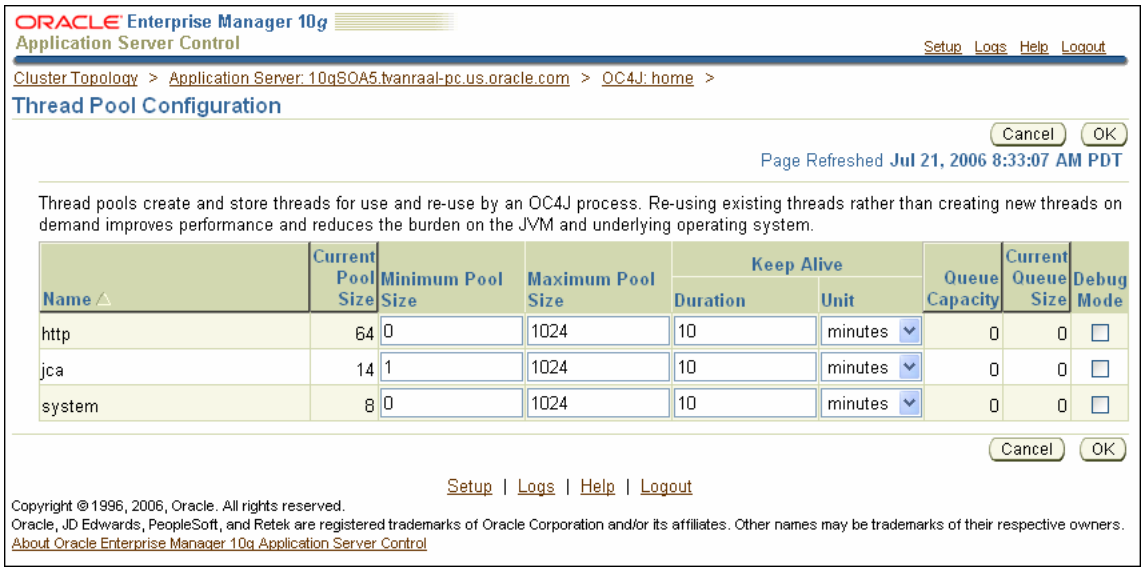
アプリケーション・サーバーのスレッド・プールは、次の方法で管理できます。

- `server.xml` の `<thread-pool>` 要素に、適切な属性を追加します。次に例を示します。

```
<thread-pool name="http" min="120" max="120" queue="240" ¥>
```

- Application Server Control コンソールのシステム MBean ブラウザからアクセスできる `ThreadPool` MBean の属性を更新します。
- Application Server Control コンソールの「スレッド・プール構成」ページを使用します。このページにアクセスするには、OC4J ホーム・ページで「管理」サブタブを選択し、「スレッド・プール構成」タスクをクリックします（図 3-1 を参照）。

図 3-1 Application Server Control コンソールの「スレッド・プール構成」



スレッド・プール・オプションの指定に関する注意事項：

- `min` 値（最小プール・サイズ）を `max` 値（最大プール・サイズ）と同じに設定することをお薦めします。`min` と `max` が同じとき、`queue` は、想定されるリクエストの最大同時実行数と等しい値に設定します。たとえば、使用する Oracle HTTP Server に OC4J インスタンスが 1 つあり直接的な RMI 接続を使用しない場合、`MaxClients` の値は最大同時実行数を表します。注意：`MaxClients` を非常に大きな値に設定する場合、`queue` のサイズは 300 以下程度で十分です。スレッド・プールおよびキューの監視方法は、3-15 ページの「スレッド・プールのパフォーマンスの検証と監視」を参照してください。

  スレッド数の設定は小さな値から始めることをお薦めします。たとえば、基本インストール・オプションの Oracle Application Server 10*g* リリース 3（10.1.3.1.0）では、最初は `min` と `max` を 80 から 100 の範囲の値に設定し、この値で問題がないかどうかパフォーマンスを監視します。スタンドアロンの OC4J 構成では、`min` と `max` を 15 から 40 の範囲の値に設定し、この値で問題がないかどうかパフォーマンスを監視します。サイトに使用可能な CPU およびメモリー・リソースが十分にあり、同時リクエスト数が現行のスレッド数設定よりも大きい場合は、スレッド数を増やします。これでシステムのリソースが飽和状態になる場合は、スレッド数を減らします。適切なスレッド設定は、アプリケーションの特性によって決まります。

- スレッド・プール・オプションの最初の設定では、min 値と max 値を同じにすることをお薦めします。ただし、max 値を大きくしてピーク時の一時的な通信に対応する場合は、最小数のスレッドが作成されるまで、リクエストごとに 1 つのスレッドが追加されます。最小数のスレッドが作成された後は、キューが一杯になるまで、新規スレッドは作成されません。OC4J では、キューが一杯にならないかぎり、スレッド数を最小かまたはそれに近い数で維持します。min 値の設定を max 値よりも小さくした場合は、キュー・サイズを小さくしておくことをお薦めします。queue のデフォルト値は 0 です。これは、リクエストはキューされず、スレッド数が指定された max 値未満の場合に新しいスレッドが作成されることを意味します。スレッドが最小値を超えないようであれば、キュー・サイズを減らす必要があります。

  関連項目： アプリケーション・サーバーのスレッド・プールの使用の詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

3.2.1.1.2 RMI 接続および RMI リクエスト・スレッド・プールの使用 OC4J では、RMI 接続スレッド用のスレッド・プールがサポートされます。デフォルトでは、RMI 接続スレッドは、http という名前のアプリケーション・サーバーのスレッド・プールから割り当てられます。RMI 接続スレッド・プールを使用すると、専用のスレッド・プールが作成され、そのスレッドによって RMI 接続に対するブロック読取りが実行されます。RMI 接続スレッド・プールを定義すると、RMI 接続に独自の制限が課され、http スレッド・プールから割り当てられなくなります。RMI 接続スレッド・プールを指定するには、server.xml の <thread-pool> 要素に rmi connection という名前を使用し、アプリケーション作業用のスレッドから長時間存続する可能性がある接続スレッドを分離します。この構成によって、残りの作業用スレッドが、長時間存続する RMI 接続に割り当てられることなく、作業用に確保されます。

RMI 接続スレッド・プールを定義すると、RMI 接続の作業は http スレッド・プールのスレッドから分離実行されます。また、RMI リクエスト・スレッド・プールも指定されている場合は、RMI 接続の作業は RMI リクエスト・スレッド・プールのスレッドから分離実行されます（RMI リクエスト・スレッド・プールは、server.xml の <thread-pool> 要素に名前 rmi request を指定して構成する）。RMI リクエスト・スレッド・プールは、RMI 接続に関連付けられている作業を、他の作業（HTTP リクエストなど）から分離して制御する必要がある場合に指定します。

注意： rmi connection 接続プールおよび rmi request 接続プールは、RMI の同時接続が多数発生し、その数が大きく変動すると想定される環境で、アクティブなワーカー・スレッド数を制限する場合にのみ指定してください。

3.2.1.1.3 作業マネージャ（JCA）のスレッド・プールの使用 10*g* リリース 3（10.1.3）から、MDB タイプの EJB では、JMS リソース・アダプタにレシーバ・スレッドが使用されるようになりました。OC4J は、これらのスレッドと他の JCA リソース・アダプタ用のスレッドを、作業マネージャのスレッド・プール（jca スレッド・プール）から割り当てます。この場合、レシーバ・スレッドによってシステム全体の同時実行性が制御されることを考慮に入れる必要があります。jca スレッド・プールはデフォルト設定のままにすることをお薦めします（デフォルトでは、JCA 作業マネージャ・スレッドは最大 2040 に制限される）。EJB MDB の同時実行性を制御する場合は、JMS ReceiverThreads に最大値を使用します。システム全体の同時実行性に対する制限は、OC4J にデプロイされたアプリケーションに構成されているすべての MDB receiverThreads の合計または jca の max スレッドの少ないほうを、http スレッド・プールに指定された max スレッドに加算したものになります（RMI 接続の max スレッドおよび RMI リクエストの max スレッドが構成されている場合は、それらも加算する）。

3.2.1.1.4 スレッド・プールのパフォーマンスの検証と監視 システムの OC4J スレッドの使用状況は、Application Server Control コンソールの「**現在のプールサイズ**」および「**現在のキュー・サイズ**」メトリックを使用して監視できます。これらのメトリックにアクセスするには、OC4J ホーム・ページの「管理」サブタブ→「スレッド・プール構成」タスクをクリックします（図 3-1 を参照）。「**現在のプールサイズ**」には、プール内にある現行のスレッド数が表示されます。「**現在のキュー・サイズ**」には、スレッドが使用可能になるのをキュー内で待機している現行のリクエスト数が表示されます。設定はデフォルトから開始して、システムの動作を監視することをお薦めします。また、スレッド・プールまたはキュー・サイズの制限をインスタンスに対して変更した場合も、これらのメトリックを監視してください。

3.2.1.2 データソースへの最大接続数の設定

データベースを使用するアプリケーションでは、データソースに関連付ける接続プールの接続数を制限することでパフォーマンスを改善できます。データベースが着信リクエストで飽和状態にならないように Oracle Application Server からデータベースへのリクエストを制限したり、データベース・アクセスが Oracle Application Server 層のリソースに負荷をかけすぎないようにデータベース・リクエストを制限するには、`max-connections` 属性を使用します。

接続プールの `max-connections` 属性は、接続プールに許可される最大接続数を指定します。デフォルトでは、`max-connections` の値は -1（無制限）に設定されています。最高のパフォーマンスを得るには、`max-connections` の値を、各自のデータベースのパフォーマンス特性に適した数に設定する必要があります。

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 `max-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、データベースが構成されているかどうかを確認する必要があります。

3.2.1.3 EJB 使用時の EJB インスタンス数の制御

EJB インスタンスの数を制限してメモリーの使用を軽減したり、同時実行性を制御して EJB が使用するリソース（データソースなど）に対する競合を減らすことが必要になる場合があります。

`max-instances` パラメータは、メモリー内で許容される、インスタンス化またはプールされた Bean インスタンスの数を指定します。

- ステートフル・セッション Bean 以外の全タイプの EJB では、`max-instances` 値に到達し新しい EJB がリクエストされると、コンテナは `call-timeout` 属性に設定されているミリ秒数だけ待機し、プール内の Bean インスタンスが使用可能になるかどうかを確認します。プール内の Bean インスタンスが使用可能にならない場合は、クライアントに `TimeoutExpiredException` がスローされます。
- ステートフル・セッション Bean では、`max-instances` 値に到達すると、コンテナはメモリーの最も古い Bean インスタンスの非アクティブ化を試みます。非アクティブ化に失敗した場合、`TimeoutExpiredException` がクライアントにスローされる前に、コンテナは、`call-timeout` 属性に設定されているミリ秒数だけ待機し、非アクティブ化、`remove()` メソッドの使用、または Bean の期限切れによって Bean インスタンスがメモリーから削除されるかどうかを調べます。

Bean インスタンスの数を無制限に許可するには、`max-instances = 0` を指定します（デフォルト値は 0）。

インスタンスのプーリングを無効にするには、`max-instances` を < 0（-1 など）に設定します。この場合は、EJB コールを開始すると、OC4J によって、新しい Bean インスタンスまたはコンテキストが作成されます。そしてコールの終了時に、コンテキストが解放され、存在しない状態にインスタンスがスローされます。

例外の `com.evermind.server.ejb.TimeoutExpiredException: timeout expired waiting for an instance` が、利用可能な EJB インスタンスがない場合に発生します。この問題を回避するには、`max-instances` および `call-timeout` パラメータを適切に設定します。

3.2.1.4 リモート EJB クライアント接続の制限

リモート EJB クライアント接続を制限するには、http スレッド・プールの max 値を変更し全スレッドに対して制限を指定するか、または着信 EJB クライアントにサービスを提供するスレッドの最大数を制御するスレッド・プール機能を使用できます。デフォルトでは、リモート EJB クライアント接続用のスレッドは http スレッド・プールから割り当てられます。RMI 接続スレッド・プールの使用が必要なときは、server.xml の <thread-pool> 要素に名前 rmi connection を構成します。RMI 接続スレッド・プールの max 値を設定することで、リモート EJB クライアント接続を制限できます。

関連項目： アプリケーション・サーバーのスレッド・プールの使用の詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

3.2.2 ロード・バランシング

Oracle Application Server には、J2EE アプリケーションに対する負荷と着信リクエストを複数のアプリケーション・サーバー・インスタンスに分散するロード・バランシング機能が組み込まれています。この機能によって、通常、スループットが向上し、レスポンス時間が短縮されます。アプリケーション・サーバー・インスタンスが複数あるとき、ロード・バランシング機能を使用すると、リクエストが複数のアプリケーション・サーバー・インスタンス間で振り分けられ、パフォーマンスが向上します。また、複数のホストで複数のアプリケーション・サーバー・インスタンスを実行することで、高可用性とフェイルオーバーの要件に対応できます。

この項には、次の項目が含まれています。

- 複数の Oracle Application Server インスタンスの構成
- Web アプリケーション・ロード・バランシング
- EJB アプリケーション・ロード・バランシング

注意： Web セッションおよびステートフル・セッション EJB に、フェイルオーバー時レプリケーションを提供する Oracle Application Server 機能には、パフォーマンスのオーバーヘッドがかかります。これらの機能はフェイルオーバー時、レプリケーションが必要な場合にのみ使用してください。

3.2.2.1 複数の Oracle Application Server インスタンスの構成

この項には、次の項目が含まれています。

- OC4J プロセス数の決定
- 異なる OC4J インスタンスへのアプリケーションのパーティション

3.2.2.1.1 OC4J プロセス数の決定 使用可能な CPU に対する OC4J プロセスの最適比率の決定は、実行するアプリケーションの特性、OC4J 構成、ハードウェア構成、予期される着信リクエストの種類と数によって異なります。CPU が少数のハードウェア構成では、1 つの OC4J インスタンスで十分です。

システム・リソースの許容範囲を超えて OC4J インスタンスを追加しても、パフォーマンスは向上しません。たとえば、1 つの OC4J インスタンスでシステムの CPU リソースが使い尽くされている場合、OC4J インスタンスを追加してもパフォーマンスが向上しないばかりか、逆にそれを低下させます。最初は OC4J インスタンスを 1 つのみ構成し、OC4J インスタンスを追加するたびにパフォーマンスが向上したか測定する方法を採用することをお薦めします。

関連項目：

- 『Oracle Application Server 高可用性ガイド』
- 『Oracle Containers for J2EE 構成および管理ガイド』

3.2.2.1.2 異なる OC4J インスタンスへのアプリケーションのパーティション 要件が異なる各アプリケーションを別々の OC4J インスタンスで動作するようにパーティション化すると、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。この場合は、特定の OC4J インスタンスが特定のアプリケーションにサービスを提供するように構成します。それぞれの OC4J インスタンスにアプリケーションをデプロイした後、パフォーマンスを監視して、全体的なスループットの増加およびレスポンス時間の短縮を確認します。

3.2.2.2 Web アプリケーション・ロード・バランシング

Oracle Application Server 環境では、Oracle HTTP Server は mod_oc4j を使用して使用可能な OC4J インスタンス間でリクエストをロード・バランシングします。この環境で mod_oc4j 構成オプションの適切なロード・バランシング・ポリシーを選択し、パフォーマンスを向上させます。デフォルトでは、リクエストはラウンド・ロビン・アルゴリズムを使用してルーティングされます。

多くのサイトで、Oracle Application Server では Oracle HTTP Server モジュールの mod_oc4j を使用して、着信したステートレス HTTP リクエストのロード・バランシングが実装されます。mod_oc4j の適切なロード・バランシング・ポリシーを選択することで、サイトのパフォーマンスを改善できます。

mod_oc4j モジュールでは、次のような構成可能なロード・バランシング・ポリシーがサポートされています。

- ラウンド・ロビン・ルーティング（mod_oc4j のデフォルトのロード・バランシング・ポリシー）
- ランダム・ルーティング
- local オプションを使用したローカル・アフィニティによるラウンド・ロビン・ルーティングまたはランダム・ルーティング
- weighted オプションを使用したホストレベルの重み付けによるラウンド・ロビン・ルーティングまたはランダム・ルーティング

注意： セッション・ベースのリクエストの場合、mod_oc4j によって、常にセッションを作成した元の OC4J プロセスへリクエストがルーティングされます。ただし、元の OC4J プロセスが使用可能な場合に限ります。これに失敗した場合、mod_oc4j では、元のリクエストと同じグループ内の別の OC4J（使用可能な場合は同じホスト内、そうでない場合は別のホスト上）にリクエストが送信されます。

---

mod_oc4j を使用したロード・バランシングにおける推奨事項 :

1. Oracle HTTP Server と OC4J の両方が 1 つのホスト上にある場合は、デフォルトのロード・バランシング・ポリシーであるラウンド・ロビン・ロード・バランシングが推奨されます。ランダム・ロード・バランシングでは、通常、同程度のパフォーマンスが提供されます。
2. Oracle HTTP Server と OC4J が別々のホスト上にある場合 :
   - OC4J ホストが 1 つのみの場合は、デフォルトのロード・バランシング・ポリシーであるラウンド・ロビン・ロード・バランシングが推奨されます。ランダム・ロード・バランシングでは、通常、同程度のパフォーマンスが提供されます。
   - 同程度の容量を持つ複数の OC4J ホストを使用する場合は、デフォルトのロード・バランシングが推奨されます。ホストに送信されるリクエスト数は、ランダムまたはラウンド・ロビンによるロード・バランシングであるかということに関係なく、ホスト上で起動される OC4J プロセス数によって暗黙的に重み付けされることに注意してください。OC4J プロセス数が 4 のホストは、OC4J プロセス数が 1 のホストの 4 倍のリクエストを受け取ります。

- ハードウェア・リソースや容量が異なる複数の OC4J ホストを使用する場合は、各ホストに送信されるリクエスト数をホストの容量に合せて明示的に重み付けします。この場合は、重み付けオプションがあるラウンド・ロビンまたは重み付けオプションがあるランダムを使用します。ホストの容量が同程度の場合は、単純なランダムまたはラウンド・ロビン・ロード・バランシングを使用します。

  たとえば、Oracle HTTP Server で mod_oc4j モジュールを構成して、Host_A に対するルーティングの重み付けが 3、Host_B に対するルーティングの重み付けが 1 のラウンド・ロビン・ポリシーを指定するには、mod_oc4j.conf に次のディレクティブを追加します。

```
Oc4jSelectMethod roundrobin:weighted
Oc4jRoutingWeight   Host_A 3
```

  この例では、ルーティングの重み付けのデフォルト値は 1 であるため、Host_B にルーティングの重み付けを指定する必要はありません。

3. Oracle HTTP Server と OC4J の両方が配置されたホストが複数ある場合 :

- Oracle HTTP Server と OC4J の両方が配置された複数のホストとハードウェア・ロード・バランサがある場合は、ローカル・アフィニティ・オプションを選択して、着信リクエストの処理に常にローカルの OC4J プロセスが選択されるように mod_oc4j を設定します。これによって、通常はパフォーマンスが向上します。ローカルの OC4J プロセスが使用できない場合、mod_oc4j は、使用可能なリモートの OC4J プロセスのリストからプロセスを選択します。

  たとえば、ローカル・アフィニティ・オプションを使用したラウンド・ロビン・ポリシーを選択するには、mod_oc4j.conf で次のディレクティブを指定します。

```
Oc4jSelectMethod roundrobin:local
```

3.2.2.3 EJB アプリケーション・ロード・バランシング

EJB アプリケーションを複数の OC4J インスタンスにデプロイすると、EJB のクライアント側アプリケーションのリクエストに対するロード・バランシングは、使用可能な OC4J インスタンス間でなされます。ロード・バランシングを使用するには、クライアント側のアプリケーションでロード・バランシングを使用するように JNDI プロパティを構成します。一部のクライアントのパフォーマンスの改善には、oracle.j2ee.rmi.loadBalance=context を設定する必要があります。これによって、クライアント全体に対して 1 度のみロード・バランシングが行われるのではなく、initialcontext コールのたびにロード・バランシングが行われるようになります。

関連項目 :

- 『Oracle Containers for J2EE サービス・ガイド』の「ORMI リクエストのロード・バランシングの構成」
- 『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』の「OC4J EJB アプリケーション・クラスタリング・サービスについて」

3.2.3 -XX:AppendRatio オプションの使用（スタンドアロン OC4J）

Sun 5.0 JVM では、負荷が非常にかかっている一部の状況で、アプリケーションの同期によってスレッド不足が生じる場合があります。これは、アプリケーションのリクエストが停止または長時間経過後にタイムアウトする原因となります。

10*g* リリース 3（10.1.3.1.0）では、管理 OC4J に対して、パラメータ -XX:AppendRatio=3 がデフォルトで指定されます。スタンドアロンの OC4J で、インストールにこの問題があると思われる場合は、JDK パラメータの -XX:AppendRatio=3 を設定し問題を回避することをお薦めします。

関連項目 : この問題の説明と回避策は、SUN バグ・データベースの http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4985566 を参照してください。

4

その他のパフォーマンス分野

この章には、Oracle Application Server の次の分野のパフォーマンス情報が含まれています。

- TopLink パフォーマンスの向上
- JTA パフォーマンスの向上
- EJB パフォーマンスの向上

4.1 TopLink パフォーマンスの向上

Oracle TopLink（TopLink）には、アプリケーションのパフォーマンスを最適化するための機能があります。そのうち、重要な分野は次のとおりです。

- **キャッシュ構成**: TopLink では、失効したデータが使用されることがないように、キャッシュを適切に構成する必要があります。また、この構成は、ロックおよび問合せのリフレッシュと組み合せて行います。また、アプリケーションのパフォーマンスとスケーラビリティを高めるために、キャッシュの仕組み、問合せおよびトランザクションの処理とキャッシュとの関係、およびキャッシュの構成オプションについて理解しておいてください。
- **効率的な問合せ**: TopLink では、SQL の実行回数を最少に抑えるために問合せのバッチ読取りオプションと結合読取りオプションについて理解すること、および必要なオブジェクトのグラフを取得することが重要です。ユースケースの検索と投影の使用に関しては、`ReportQueries` を使用して、読み込んでオブジェクトとしてキャッシュする必要があるのは、変更可能か、またはリクエスト間で共有可能なオブジェクトのみであることを確認します。
- **効率的なトランザクション**: TopLink では、`UnitOfWork` を使用してトランザクション・スコープを最小化し、それによってコミット・サイクルを最小化する方法を理解しておいてください。

TopLink のドキュメントには、これらの重要な TopLink のパフォーマンス分野に関する情報が含まれています。TopLink のパフォーマンスを最適化するためのアプリケーションをチューニングする方法の詳細は、ドキュメントの該当する章を参照してください。

関連項目：

- 『Oracle TopLink 開発者ガイド』の第 11 章「最適化」
- 『Oracle TopLink 開発者ガイド』の第 90 章「キャッシュの概要」
- 『Oracle TopLink 開発者ガイド』の第 96 章「TopLink 問合せの概要」

4.2 JTA パフォーマンスの向上

この項では、JTA のパフォーマンス・オプションについて説明します。この項には、次の項目が含まれています。

- パフォーマンス向上を目的とした 2 フェーズ・コミット・ロギングの構成
- パフォーマンス向上を目的とした JTA データソースの構成
- JTA リソースの監視

4.2.1 パフォーマンス向上を目的とした 2 フェーズ・コミット・ロギングの構成

構成オプションを使用することで、2 フェーズ・コミット・ロギングのタイプとレベルを制御できます。構成オプションを変更するには、`transaction-manager.xml` ファイルを変更するか、または JTA リソースの MBean を使用します。この MBean は、Application Server Control コンソールの「トランザクション・マネージャ」ページで利用できます。2 フェーズ・コミット・ロギングを構成する場合は、2 フェーズ・コミット・ロギングをオフにしたときのトランザクションへの波及効果を認識しておく必要があります。

注意： 2 フェーズ・コミット・ロギングは、デフォルトでオフに設定されています。デフォルトのロギング・レベルを使用すると、JTA リソースでは、リカバリおよび完全な ACID プロパティがサポートされません。

表 4-1 に、2 フェーズ・コミット・ロギングの構成オプションを示します。これらのオプションは、transaction-manager.xml で設定するか、または JTA リソースの MBean を使用して設定できます。

表 4-1 2 フェーズ・コミット・ロギングのログ・タイプ構成オプション

ログ・タイプ	説明	パフォーマンス上の注意
none	ロギング（またはリカバリ）なし。 これはデフォルト値です。	リカバリおよび ACID プロパティが不要の場合は、このオプションを指定することで最高のパフォーマンスが得られます。
file	ファイル・ロギングでは、トランザクションのリカバリ用に、ファイル・システムへのロギングを指定します。	通常、ファイル・ロギングはデータベース・ロギングより、オーバーヘッドが低くなるためパフォーマンスがよくなります。
database	データベース・ロギングでは、トランザクションのリカバリ用に、Oracle データベースへのロギングを指定します。	通常、ファイル・ロギングのほうがデータベース・ロギングよりパフォーマンスがよくなります。

関連項目： 2 フェーズ・コミット・ロギングの構成オプションの詳細、およびそれぞれのログ・タイプ構成オプションによるトランザクションおよびリカバリへの波及効果の詳細は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。

4.2.1.1 JTA ストアのファイル・ロギング・オプションの設定

表 4-2 で、ファイル・ストア・ロギングのパフォーマンス設定について説明します。これらの設定は transaction-manager.xml で指定するか、または JTA リソースの MBean を使用して指定できます。2 フェーズ・コミット・トランザクションの最大同時実行数が 256 より少ない場合は、デフォルト設定が適しています。

2 フェーズ・コミット・トランザクションの最大同時実行数を判定するには、JTAResource メトリック表の TwoPhaseCommitCompletion.maxActive メトリックを使用します。

関連項目： JTA リソースのメトリックの詳細は、D-2 ページの表 D-1 を参照してください。

表 4-2 JTA ファイル・ストア・ロギングのパラメータ

パラメータ	説明	パフォーマンス上の注意
maxOpenFiles	オープンまたはアクティブな状態を維持できるファイル記述子の最大数。この数を超えると、xid が再びリクエストされるまで、最も古いファイル記述子が解放されます。 maxOpenFiles の値は、できるだけ超えないようにしてください。 デフォルト値 : 256	最適な値は、2 フェーズ・コミット・トランザクションを使用する最大同時リクエスト数（プラス、リカバリ用に必要になる可能性がある少数の追加ファイル数）に対応できる大きさの数値です。 maxOpenFiles の値は、オペレーティング・システムのオープン・ファイル記述子の上限によって制限されます。
minPoolSize	起動時にプールに事前に割り当てられるファイル数。 デフォルト値 : 40	最適な値は、2 フェーズ・コミットの最大同時リクエスト数を処理できる大きさの数値です。 注意 : 最大同時実行数が大きい場合、起動時のコストが高くならないように、デフォルト値よりは大きくても maxOpenFiles の値よりは小さい値に設定します。
oldFileReleaseSize	maxOpenFiles を超えたときに閉じる、最も古いファイル・ハンドルの数。 デフォルト値 : 20	本来は好ましくないことですが、maxOpenFiles の値を繰返し超えることが予想される場合は、oldFileReleaseSize の値を大きくして、解放するファイル・ハンドルの数を増やすことで、maxOpenFiles を超える回数を減らすことができます。

注意： maxOpenFiles に指定した数値で、トランザクションの数が制限されることはありません。maxOpenFiles を超えると、古いファイル・ハンドルが解放されますが、新しいトランザクションは引き続き作成できます（oldFileReleaseSize パラメータを参照）。

4.2.2 パフォーマンス向上を目的とした JTA データソースの構成

この項には、次の項目が含まれています。

- データソースのタイプの指定
- 最終リソース・コミットの使用
- 単一のデータソースの使用（可能な場合）

4.2.2.1 データソースのタイプの指定

一般的に、XA に準拠していないデータソースは、XA データソースより高速です。XA に準拠した完全な 2 フェーズ・コミットが必要な場合は、XA データソースを使用する必要があります。

トランザクションのロギングが none に設定されている場合は、XA 準拠または非準拠のリソースを、グローバル・トランザクションでいくつでも確保できます。ただしこの場合、ACID の保証にもリカバリにも対応しません。

トランザクションのロギングが有効になっている場合は、グローバル・トランザクションに組み込むリソースは XA に準拠していることが必要です。最終リソース・コミット機能を使用すると、XA に準拠していない単一のリソースを、XA トランザクションに組み込むことができます。

関連項目：『Oracle Containers for J2EE サービス・ガイド』

4.2.2.2 最終リソース・コミットの使用

最終リソース・コミットは、XA に準拠していないリソースをグローバル・トランザクションに組み込めるだけでなく、パフォーマンスの最適化を図るために使用することもできます。XA 対応リソースを非 XA リソースとして確保して最終リソース・コミットを使用すると、そのリソースが XA ロギングを実行する必要がないため、パフォーマンスが向上します。また、そのリソースはインダウト状態、つまり準備状態には決してならないため、リソースのロックが発生しません。最終リソース・コミットは、パフォーマンスの最適化を図るために使用できますが、そのかわり正確性は保証されなくなります。

関連項目：『Oracle Containers for J2EE サービス・ガイド』

4.2.2.3 単一のデータソースの使用（可能な場合）

単一のリソースへのアクセスにアプリケーションが複数のデータソースを使用すると、（XA トランザクションを使った）2 フェーズ・コミット操作が、意図せずに使用される可能性があります。一部のケースでは、構成を変更することでパフォーマンスが向上することがあります。この構成変更により、OC4J は 2 フェーズ・コミットでなく 1 フェーズ・コミットを使用するようになります。

次のメトリックを使用すると、1 フェーズ・コミット数と 2 フェーズ・コミット数、およびリソースの確保を行わないグローバル・トランザクションの件数を確認できます。

```
/oc4j/JTA/SinglePhaseCommitCompletion.completed
/oc4j/JTA/TwoPhaseCommitCompletion.completed
/oc4j/JTA/AverageCommitTime.completed
```

注意： /oc4j/JTA/AverageCommitTime.completed メトリックでは、JTA 関連のトランザクションがすべて示されますが、ローカル・トランザクションは示されません。

同じグローバル・トランザクション内に複数のデータソースが存在する場合は必ず、それらのデータソースが実際には同じデータベースを指していたとしても、XA の 2 フェーズ・コミット・トランザクションが使用されます。アプリケーションで使用するデータソースが単一のデータベースおよびスキーマにデプロイされている場合は、単一のデータソースを使用するようにアプリケーションを再構成することができます。これにより、2 フェーズ・コミットが 1 フェーズ・コミットに変更され、パフォーマンスが向上します。

このように、表など従来型のデータベース・リソースを使用し、リソースが同じデータベースに存在している場合に OJMS を使用するトランザクション型アプリケーションは、リソースごとに単一のデータソースを指定することにより、一部の 2 フェーズ・コミットを回避できます。この変更によってパフォーマンスが向上しますが、この場合、OJMS データソースの構成が、表へのアクセスに対して指定された構成と一致している必要があります。

関連項目： ローカル・トランザクションとグローバル・トランザクションの詳細は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。

4.2.3 JTA リソースの監視

JTA リソースを監視する際には、エラーによってパフォーマンス上の問題が発生する可能性があることに注意してください。JTA エラーがあるかどうかを調べるには、JTAResource メトリック表のメトリックを使用して、ロールバックと例外の件数に 0 より大きい数値がないかを調べます。たとえば、`RollbackExceptionCount`、`RolledbackCount`、`SystemExceptionCount` の各メトリックの値を調べます。

また、一部のパフォーマンス上の問題から JTA エラーが発生することもあります。たとえば、パフォーマンスが悪い場合に、タイムアウト・エラーが発生することがあります。この場合は、メトリック `RolledbackDueToTimedOutCount` の値を調べます。

関連項目： 「JTA リソースのメトリック」の表 D-1（D-2 ページ）

4.3 EJB パフォーマンスの向上

この項では、次の項目について説明します。

- MDB パフォーマンスの向上
- EJB CMP 2.1 パフォーマンスの向上

4.3.1 MDB パフォーマンスの向上

この項では、`orion-ejb-jar.xml` 構成ファイルで指定され、メッセージドリブン Bean（MDB）に適用される、パフォーマンスに関連する重要な EJB 構成プロパティのいくつかについて説明します。内容は次のとおりです。

- JMS コネクタのレシーバ・スレッドの設定
- ejbCreate メソッドを使用した 1 回かぎりの初期化
- MDB リソースの監視

関連項目： MDB の使用と構成の詳細は、『Oracle Containers for J2EE サービス・ガイド』の「Oracle Enterprise Messaging Service（OEMS）」を参照してください。

4.3.1.1 JMS コネクタのレシーバ・スレッドの設定

MDB のキューにメッセージを送信する多数の同時ユーザーがいる場合や onMessage メソッドで大量の処理が発生する場合、MDB に対して JMS コネクタのレシーバ・スレッド数を設定すると、パフォーマンスを向上させることができます。たとえば、onMessage メソッドに別の EJB をコールするコードが含まれていて、他のメッセージの処理中に EJB 処理が同時に発生する場合、JMS コネクタのレシーバ・スレッドを 1 より大きい値に設定するとパフォーマンスが向上します。基礎となる JMS コネクタおよび特定の MDB によっては、JMS コネクタの ReceiverThreads 構成プロパティの値を増やすと、パフォーマンスが大幅に向上するアプリケーションもあります。

たとえば、キューに 100 個のメッセージが含まれていて、ReceiverThreads がデフォルト値の 1 に設定されている場合、メッセージは順番に 1 つの MDB のレシーバ・スレッド・プロセスによってのみ処理されます。ReceiverThreads を 5 に設定すると、最大で 5 個の MDB インスタンスがキューからメッセージを取得でき、メッセージをパラレル処理できることが指定されます。この例では、OC4J はメッセージのデキューと処理に最大 5 個の MDB スレッドを使用するので、100 個のメッセージの処理の完了に必要な合計時間は短くなります。

注意： JMS コネクタの ReceiverThreads 値では、スレッドの最大値が指定されます。そのため、負荷によっては、一部のスレッドが使用されないこともあります。

JMS コネクタの ReceiverThreads の値を 1 より大きい値に設定すると、MDB の複数のインスタンスによって、キューのメッセージを同時に処理できるようになります。ただし、この場合、パフォーマンスの向上は、アプリケーションおよび指定したスレッドの数によって異なります。指定した値が大きすぎると、リソースの競合によってパフォーマンスが低下することがあります。

注意： JMS トピックに関しては、JMS コネクタの ReceiverThreads 構成プロパティは常に、値 1 に設定してください（1 より大きい値が有効に機能するのは、キューのみです）。

4.3.1.1.1 MDB のメッセージ処理順序に関する要件の考慮

メッセージを順番に処理する必要がある場合は、値が 1 に設定された JMS コネクタの ReceiverThreads を使用します。1 より大きい値を設定した ReceiverThreads を使用した場合、メッセージはキューから逐次削除されますが、MDB では複数のスレッドでメッセージが処理されるのでメッセージを処理する順序は保証されません。

4.3.1.1.2 スレッド・プールの設定と Bean インスタンスの設定の調整

OC4J は、JMS コネクタのレシーバ・スレッドとして使用されるスレッドを作業マネージャのスレッド・プール（Application Server Control コンソールおよび server.xml で jca スレッド・プールとして示される）から割り当てます。JMS コネクタのレシーバ・スレッド数は、作業マネージャのスレッド・プールで、jca スレッド・プールの max パラメータを使用して制限できます。また、min の値を使用して、作業マネージャのスレッド・プールの利用可能なスレッドの初期数を設定することもできます。

パフォーマンス上の注意： jca 作業マネージャのスレッド・プールはデフォルト設定のままにすることをお薦めします。そのため、EJB MDB の同時実行数を制御する場合は、JMS コネクタの ReceiverThreads の値を使用します。

JMS コネクタの ReceiverThreads の値を設定するときには、次の構成オプションを調整する必要があります。

- 全体の同時実行性に対する制限 : これは、システム上のすべての thread-pool の max スレッドに、次のいずれかを加算したものになります。

  a. OC4J にデプロイされたアプリケーションに構成されている MDB JMS コネクタのすべての ReceiverThreads の合計（jca スレッド・プールとして示される作業マネージャのスレッド・プールの max がデフォルト値に指定されている場合、他の JCA アダプタによって使用されるすべてのスレッドの合計が加算される）。

  b. max パラメータで指定された、作業マネージャのスレッド・プール（jca として示される）の使用可能な最大スレッド数（この数が、JMS コネクタのすべての ReceiverThreads の合計に対して指定された最大数より小さい場合）。

- レシーバ・スレッドに適した最小 MDB インスタンスの設定 : 作成された JMS コネクタの ReceiverTheads とアクティブな MDB Bean インスタンスの数は、1 対 1 の関係になります。MDB の初期処理時間が長くなる可能性がある場合は、MDB の min-instances を、JMS コネクタの目的の ReceiverTheads と一致するように設定して、それらのインスタンスが起動時に初期化されるようにします。

  また、MDB 構成の min-instances の値は、MDB ごとに設定された JMS コネクタの ReceiverThreads より大きな値にしないように構成してください。

  目的のインスタンス数を維持するには、アイドル中に MDB インスタンスが削除されないように、pool-cache-timout を十分な長さの値に設定します。

  関連項目 : 「OC4J スレッド・プールによる同時実行性の制御」（3-11 ページ）

4.3.1.1.3 JMS コネクタのレシーバ・スレッド設定時におけるデータベース接続の考慮

MDB でデータベース接続が必要な場合は、JMS コネクタの ReceiverThreads の数によって、必要なデータベース接続の数も乗算されます。たとえば、ある MDB で 5 つのデータベース接続が同時に使用され、アクティブな MDB インスタンスが 5 つある場合、要求されるデータベース同時接続数は 25 になります。このように、データソース max-connections の件数の計算には、JMS コネクタの ReceiverThreads の数を組み込む必要があります。

関連項目 : 「データベース接続の再利用」（3-6 ページ）

4.3.1.2 ejbCreate メソッドを使用した 1 回かぎりの初期化

MDB は、ステートレスで、複数の起動にわたって特定のクライアントの状態を保持することはありません。ただし、クライアント関連以外の状態については、MDB インスタンスは、複数のクライアント・メッセージの処理の間、状態を保持できます。たとえば、ルックアップの状態を保持できます。さらに、EJB への参照など、onMessage の複数の起動にわたってキャッシュする必要があるその他の状態情報は、ejbCreate メソッドで初期化されてキャッシュされ、MDB のパフォーマンスが最適になります。

アイドル状態の MDB オブジェクトがプールから削除され、必要に応じて再割当てされる場合は、ejbRemove メソッドで、状態を忘れずに破棄してください。

4.3.1.3 MDB リソースの監視

MDB で OracleAS JMS がメッセージ・プロバイダとして使用される場合、Oracle Application Server パフォーマンス監視ツールから DMS メッセージ関連のメトリックを使用できます。

たとえば、OracleAS JMS JMSStoreStats メトリック表には、MDB によって使用されるキューに対応する宛先の情報が含まれています。

```
destination.value:        name
messageDequeued.count:    x ops
messageEnqueued.count:    x ops
messageCount.value:  n
```

これらのメトリックには、宛先名、エンキューされた合計メッセージ数、デキューされた合計メッセージ数、およびキュー内の現在の合計数が表示されます。

また、MDB の onMessage メトリックを調べて、onMessage の時間が予想どおりの長さになっているかを確認したり、maxActive メトリックを使用して、同時実行されるレシーバ・スレッドの合計数が予想どおりの数になっているかを確認することもできます。

```
client.active:   1   threads
client.avg:   112   msecs
client.completed:   4   ops
client.maxActive:        1       threads
client.maxTime:  70       msecs
client.minTime:  130      msecs
client.time:     121      msecs
```

注意： JMS 宛先を監視する場合、MDB 以外のアプリケーションもその宛先にアクセスできます。したがって、アプリケーションのパフォーマンスをテストする場合は、そのアプリケーションがメトリックでレポートされるメッセージ・アクティビティを行っているかどうかを確認してください。

Application Server Control コンソールには、すべての MDB と個々の MDB のパフォーマンスに関する情報が表示されます。

MDB のサマリー情報にアクセスするには、次の手順を実行します。

1. OC4J ホーム・ページから、「管理」サブタブを選択します。
2. 表の「サービス」で、「**JMS プロバイダ**」タスクを選択します。
3. Application Server Control コンソールの「パフォーマンス」領域に、次のサマリー情報が表示されます。

```
Active Connections
Messages Waiting for Read
Messages Waiting for Commit
Messages Enqueued per Second
Messages Dequeued per Second
Messages Paged In per Second
Messages Paged Out per Second
Messages Committed Since Startup
Messages Rolled Back Since Startup
Messages Expired Since Startup
```

個々の MDB の情報にアクセスにするには、Application Server Control コンソールのパフォーマンス領域を使用します。

1. OC4J ホーム・ページから、「アプリケーション」サブタブを選択します。
2. 監視するアプリケーションを選択します。
3. 「**モジュール**」表で、該当する EJB モジュールを選択します。
4. 「**メッセージドリブン Bean**」領域で、監視する MDB を選択します。次の情報が表示されます。

```
Messages Dequeued
Messages Rolled Back
Average Message Processing Time (seconds)
Number of Available Instances
Number of Used Instances
```

関連項目： 「OC4J JMS メトリック」（D-13 ページ）

4.3.2 EJB CMP 2.1 パフォーマンスの向上

この項では、CMP を使用するエンティティ Bean で利用できるいくつかのパフォーマンス・オプションについて説明します。次の項目が含まれます。

- 効率的な SQL 文と SQL 問合せの使用
- キャッシュ構成のパフォーマンスのチューニング
- CMP リソースの監視

注意： この項では、いくつかの EJB オプションについて説明します。オプションを使用することにより、アプリケーションのパフォーマンスを向上できます。オプションの設定方法については、この項では説明しません。各種オプションを使用するための EJB の構成の詳細は、『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』および『Oracle TopLink 開発者ガイド』を参照してください。

関連項目：

- 『Oracle TopLink 開発者ガイド』の「最適化」
- 『Oracle TopLink 開発者ガイド』の「キャッシュの概要」
- 『Oracle TopLink 開発者ガイド』の「TopLink 問合せの概要」
- 『Oracle TopLink 開発者ガイド』の「拡張作業ユニット API の使用」
- 『Oracle TopLink 開発者ガイド』のデータベース・トランザクションの分離レベルに関する項

4.3.2.1 効率的な SQL 文と SQL 問合せの使用

この項では、効率的な SQL 文と SQL 問合せの使用について説明します。表 4-3 と表 4-4 に、SQL 文および SQL 問合せに関連するチューニング・パラメータとパフォーマンス推奨事項を示します。

表 4-3 効率的な SQL 文と SQL 問合せを使用した CMP EJB

チューニング・パラメータ	説明	パフォーマンス上の注意
パラメータ化された SQL バインディング	パラメータ化された SQL と準備された文キャッシュを使用すると、頻繁にコールされる問合せに対してデータベースの SQL エンジンが SQL の解析と準備を行う回数が減るため、パフォーマンスが向上することがあります。データベースと JDBC ドライバのなかには、パラメータ化された SQL と準備された文キャッシュをサポートしていないものもあるため、TopLink と OC4J/CMP では、これらのオプションがデフォルトでは有効になっていません。OC4J にバンドルされている Oracle JDBC ドライバは、これらのオプションをサポートしていません。 デフォルト値 : オフ 関連項目 :『Oracle TopLink 開発者ガイド』の、名前を付けた問合せ、パラメータ化された SQL および文キャッシュのプロジェクト・レベルでの構成に関する項	すべての問合せに対して、(プロジェクト・レベルで) SQL パラメータのバインディングをオンにします。パラメータ化された SQL と準備された文キャッシュをサポートするデータベースと JDBC ドライバを選択し、それに対してこれらのオプションを有効化することをお薦めします。
JDBC 文キャッシュ	カーソルを繰返し作成したり、文を繰り返し解析および作成することに起因するオーバーヘッドを軽減するために、文キャッシュを使用することで、データベースを使用する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。 注意 : データソースの文キャッシュを使用してください (CMP には TopLink の文キャッシュは使用しないでください)。 デフォルト値 : オフ 関連項目 :『Oracle Containers for J2EE サービス・ガイド』の「マネージド・データソースでの文キャッシュ」	使用している JDBC ドライバが文キャッシュをサポートしている場合は、必ずそのオプションを有効化してください。Oracle JDBC ドライバは、このオプションをサポートしています。このオプションを設定するには、`data-sources.xml` で `num-cached-statements` を設定します。

表 4-3 効率的な SQL 文と SQL 問合せを使用した CMP EJB（続き）

チューニング・パラメータ	説明	パフォーマンス上の注意
フェッチ・サイズ	JDBC フェッチ・サイズは、追加の行が必要な場合にデータベースからフェッチする行数に関して、JDBC ドライバにヒントを提供します。多数のオブジェクトを返す大きな問合せに対しては、問合せに使用する行フェッチ・サイズを構成し、選択条件を満たすために必要なデータベースのヒット件数を減らして、パフォーマンスを高めることができます。 ほとんどの JDBC ドライバは、デフォルトのフェッチ・サイズである 10 を使用します。1000 個のオブジェクトを読み取る場合、フェッチ・サイズを 256 に増やすと、問合せ結果のフェッチ所要時間を大幅に短縮できます。 注意 : デフォルト値は、JDBC ドライバのデフォルト値を使用することを意味します。Oracle JDBC ドライバの場合、このデフォルト値は通常 10 行です。 デフォルト値 : 0 関連項目 :『Oracle TopLink 開発者ガイド』	最適のフェッチ・サイズは、常に明らかなわけではありません。通常、最適なフェッチ・サイズは、予想される合計結果サイズの 1/2 または 1/4 です。結果セットのサイズが不明な場合に、設定したフェッチ・サイズが大きすぎたり小さすぎたりすると、パフォーマンスが低下することがあるので注意してください。
バッチ書込み	バッチ書込みでは、`INSERT` 文、`UPDATE` 文および `DELETE` 文のグループが、個別にではなく単一のトランザクションで一括してデーベースに送られるため、データベースのパフォーマンスを向上させることができます。 デフォルト値 : オフ 関連項目 :『Oracle TopLink 開発者ガイド』の、データ・アクセスの最適化に関する項	すべての EJB に対して有効化します。

4.3.2.1.1 コンテナ管理の関連性の問合せのパフォーマンスのチューニング

表 4-4 に、パフォーマンスのチューニングのための CMR パラメータを示します。

表 4-4 CMP EJB および CMR の問合せのパフォーマンス・オプション

チューニング・パラメータ	説明	パフォーマンス上の注意
バッチ読取り	バッチ読取りでは、問合せの選択条件がオブジェクトの関連属性のマッピングによって伝播されます。また、複合的なオブジェクト・グラフによって、バッチ読取り操作をネストすることもできます。それにより、必要な SQL SELECT 文の数が大幅に減り、データベースへのアクセスの効率が向上します。 デフォルト値 : オフ 関連項目 :『Oracle TopLink 開発者ガイド』の、バッチ読取りに関する項	同じく取得が必要な表データに列マッピングされた表の問合せに使用します。 データのすべてにアクセスすることがわかっている場合は、バッチ読取りと結合のいずれか一方のみを使用してください。関係にアクセスしない場合は、インダイレクションによってロードが遅延する状態のままにしてください。 バッチ読取りは、1 対 m などの関係に対しては、データベースから読み取るデータが少なくなるため（`n*m` に対して n で済むため）、効率が上がります。また、m 対 1 の論理関係でも、データの読取りが減るため効率が上がります。バッチ読取りは、キャッシュを併用すると、パフォーマンスがさらによくなります。これは、元のオブジェクトとその関係がすでにキャッシュされている場合、バッチ問合せの実行が不要になるためです（この場合、結合ですべてのデータがすでに読み込まれていると想定されます）。 TopLink では、ほとんどのマッピング（1 対 1、1 対 m、m 対 m、dc、ac）に対して、問合せおよびマッピングのレベルでのバッチ読取りをサポートしています。

表 4-4 CMP EJB および CMR の問合せのパフォーマンス・オプション（続き）

チューニング・パラメータ	説明	パフォーマンス上の注意
結合	結合読取りは問合せ最適化機能の 1 つで、これを使用すると、1 つのクラスの単一問合せでデータを戻し、そのクラスのインスタンスとその関連オブジェクトを構築することができます。この機能を使用すると、データベースへのアクセスが減るため、問合せのパフォーマンスが向上します。デフォルトでは、関係は結合読取りされません。各関係は、インダイレクションを使用している場合はアクセス時に個別にフェッチされ、インダイレクションを使用していない場合は個別のデータベース問合せとしてフェッチされます。 デフォルト値 : 使用しない 関連項目 :『Oracle TopLink 開発者ガイド』の、結合読取りの使用に関する項	同じく取得が必要な表データに列マッピングされた表の問合せに使用します。 データのすべてにアクセスすることがわかっている場合は、バッチ読取りと結合のいずれか一方のみを使用してください。関係にアクセスしない場合は、インダイレクションによってロードが遅延する状態のままにしてください。 TopLink では、問合せレベルでは 1 対 1 と 1 対 m の結合のみ、マッピング・レベルでは 1 対 1（内部）の結合のみをサポートしています。パフォーマンスの面では、結合は 1 対 1 の論理関係の場合にのみお薦めします。 継承を使用し、複数の表にまたがるサブクラスを持つ関連クラスに対しては、結合はサポートされていません。
インダイレクション	インダイレクションがオンになっていないと、TopLink は、永続オブジェクトの取得時に、その永続オブジェクトが参照する依存オブジェクトをすべて取得します。関係マッピングによってマッピングされた属性に対してインダイレクション（遅延読取り、遅延ロード、Just-in-Time 読取りともいう）を構成すると、TopLink はインダイレクション・オブジェクトを参照オブジェクトのプレースホルダとして使用します。依存オブジェクトの読取りは、その属性自体にアクセスするまで延期されます。その結果、パフォーマンスが大幅に向上することがあります。特に、アプリケーションが必要としているものが取得したオブジェクトの内容のみに限られ、そのオブジェクトの関連オブジェクトが不要な場合に効果があります。 デフォルト値 : すべての CMR に対するバリュー・ホルダのインダイレクションがオン 関連項目 :『Oracle TopLink 開発者ガイド』のインダイレクションに関する項	インダイレクション・オプションは、デフォルト値のままにしておきます。つまり、あらゆる状況で CMP に対してインダイレクションを使用します。結合読取りやバッチ読取りを使用して参照オブジェクトの問合せを実行すると、さらに効率が上がります。

4.3.2.2 キャッシュ構成のパフォーマンスのチューニング

表 4-5 に、キャッシュ構成オプションを示します。

表 4-5 CMP EJB およびキャッシュ構成オプション

チューニング・パラメータ	説明	パフォーマンス上の注意
オブジェクト・キャッシュ	TopLink セッションには、オブジェクト・キャッシュが用意されています。TopLink 永続性マネージャを使用する CMP 2.1 アプリケーションが TopLink セッションを作成し、そのセッションはデフォルトでこのキャッシュを使用します。セッション・キャッシュと呼ばれるこのキャッシュには、データベースとの間で読み書きされたオブジェクトの情報が格納されます。セッション・キャッシュは TopLink 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重要な要素の 1 つです。通常、サーバー・セッションのオブジェクト・キャッシュは、そのセッションから取得されたすべてのクライアント・セッションで共有されます。分離セッションは、共有オブジェクト・キャッシュから分離された固有のセッション・キャッシュを備えています。 デフォルト値 : 有効 関連項目 :『Oracle TopLink 開発者ガイド』の、オブジェクト・キャッシュに関する項	即時問合せに対しては無効にします（分離キャッシュを使用）。

表 4-5 CMP EJB およびキャッシュ構成オプション（続き）

チューニング・パラメータ	説明	パフォーマンス上の注意
問合せ結果セット・キャッシュ	TopLink では、TopLink のオブジェクト・キャッシュに加えて、問合せキャッシュもサポートしています。 ■ オブジェクト・キャッシュでは、主キーによってオブジェクトの索引が作成されるため、主キーの問合せでキャッシュ・ヒットを取得できます。オブジェクト・キャッシュを使用することで、データソースにアクセスする問合せで、該当するオブジェクトがすでに存在する場合に、オブジェクトとその関係を構築するコストが不要になります。 ■ 問合せキャッシュは、オブジェクト・キャッシュとは区別されます。問合せキャッシュの索引は、オブジェクトの主キーではなく、問合せと問合せパラメータによって作成されます。そのため、同じパラメータで実行されたあらゆる問合せで、問合せキャッシュ・ヒットを取得し、同じ結果セットを戻すことができます。 デフォルト値 : 使用しない 関連項目 :『Oracle TopLink 開発者ガイド』の、問合せ結果のセッション・キャッシュへのキャッシングに関する項	主キー以外のキーで頻繁に実行される問合せのうち、結果セットがめったに変わらないものに使用します。 キャッシュ検証タイムアウトとともに使用し、必要に応じてリフレッシュします。
キャッシュ・サイズ	デフォルト値 : `SoftCacheWeakIdentityMap` サイズ 100（EJB あたり） 関連項目 :『Oracle TopLink 開発者ガイド』の、キャッシュおよびアイデンティティ・マップの構成のガイドラインに関する項	キャッシュ・サイズは、1 つのトランザクション内で参照される（同タイプの）オブジェクトの最大数を格納できる大きさに設定します。
ロック	表 4-6 に示すロック方針がサポートされています。 デフォルト値 : ロックなし 関連項目 :『Oracle TopLink 開発者ガイド』の、ロック方針の構成に関する項と記述子およびロックの概要に関する項	
キャッシュ使用	キャッシュ使用一致を使用すると、セッション・キャッシュはチェックされません。全読取りの場合、まずデータベースをチェックし、その後、作業ユニットの変更と新規オブジェクトおよび削除されたオブジェクトに結果を一致させます。オブジェクト読取りの場合、まず作業ユニットの変更と新規オブジェクトをチェックして一致するオブジェクトがないかどうかを確認し、その後、データベースをチェックして、変更と削除されたオブジェクトに結果を一致させます。 一致機能は、`CheckCacheThenDatabase` よりも POJO のデフォルト・オプションである `CheckCacheByPrimaryKey` に似ていますが、オブジェクト読取り問合せの場合は `CheckCacheThenDatabase` にも似ているところがあります。作業ユニットの変更、新規オブジェクトおよび削除されたオブジェクトに問合せ結果を一致させる必要があることから、一致を使用すると追加のオーバーヘッドが発生します。 `CMP` の場合、デフォルトは `ConformResultsInUnitofWork` です。一致処理を行わないようにするには、通常、このデフォルト値を `CheckCacheByPrimaryKey` に変更します。全読取り問合せの場合、これは基本的に `DoNotCheckCache` と同じ意味です。 デフォルト : `ConformResultsInUnitofWork`	コミットされていないデータをトランザクションに読み取らせる必要がない場合、特に読取り専用操作では、各記述子、および不要な場合は問合せレベルで、キャッシュ使用一致をオフにします。
分離	TopLink を使用する CMP アプリケーションには、特定のデータベース・トランザクションの分離レベルを単独で設定するチューニング・パラメータはありません。一般的な CMP アプリケーションでは、データベース・トランザクションの分離レベルが適用されるとき、また特定のデータベース・トランザクションを分離できる程度には、次のような様々な要素が影響します。 ■ ロック・モード ■ セッション・キャッシュの使用 ■ 外部アプリケーション ■ データベース・ログイン・メソッド setTransactionIsolation 関連項目 :『Oracle TopLink 開発者ガイド』の、データベース・トランザクションの分離レベルに関する項	

表 4-5 CMP EJB およびキャッシュ構成オプション（続き）

チューニング・パラメータ	説明	パフォーマンス上の注意
キャッシュのリフレッシュ	デフォルトでは、TopLink はデータソースから読み取ったオブジェクトをキャッシュします。これらのオブジェクトに対する後続の問合せでは、キャッシュへのアクセスが行われます。その結果、データソースへのアクセスが減り、オブジェクトとその関係を再構築するコストが不要になるため、パフォーマンスが向上します。全読取り問合せなどの問合せでデータソースにアクセスした場合も、返されたレコードに対応するオブジェクトがキャッシュに存在すれば、キャッシュ内のオブジェクトが使用されます。このデフォルトのキャッシング方針は、失効したデータをアプリケーションにもたらす場合があります。 リフレッシュは、記述子レベル（`alwaysRefreshCache`）と問合せレベル（`refreshIdentityMapResult`）で有効化できます。記述子レベルで設定した場合、（`disableCacheHits`）も設定しないと、プライマリ・オブジェクト読取り問合せではキャッシュ・ヒットが取得されます。 失効したデータや競合するデータがデータベースにコミットされないようにするには、適切なロック方針を使用するしか方法がありません。キャッシュでのデータのリフレッシュに関して、考慮すべきケースがいくつかあります。これらのケースはいずれも、パフォーマンスに影響を及ぼします。 ■ キャッシュされたデータではなく最新データが常に必要な場合は、分離キャッシュを使用することをお薦めします。これは、アプリケーション内の特定のデータが頻繁に変更されるため、競合が検出されたときにだけデータをリフレッシュするのではなく、常にデータをリフレッシュすることが好ましいというような場合です。 ■ 失効したデータは使用したくないが、失効データの取得が大した問題でない場合は、解決策として `cache-invalidation` 方針を使用することをお薦めします。この場合は、コミット時ロックも使用する必要があります。また通常は、ロック・エラーの発生時に失効オブジェクトをリフレッシュするメソッドが必要になります。この場合、`findAndRefreshByPrimaryKey` など、オブジェクトを強制リフレッシュするように（`refreshIdentityMapResult`）を設定した finder を定義する必要があります。コミット時ロックを使用する場合は、さらに `alwaysRefreshCache` と `onlyRefreshCacheIfNewerVersion` を有効にすると、データベースにアクセスする問合せで、返された失効データをリフレッシュできるとともに、無効オブジェクトが変更されていない場合はリフレッシュしないようにできます。また、リフレッシュされたデータが必要であることがわかっている場合に、特定の finder 操作に対してリフレッシュを有効にしたり、クライアントから取得したデータをリフレッシュするオプション（リフレッシュを実行する finder をコールするもの）を提供したりすることもできます。 ■ データが失効していても問題ない場合。 この場合は、コミット時ロックを使用する必要があります。また、ロック・エラーの発生時に失効オブジェクトをリフレッシュする操作をコミットするエラー・ハンドラを記述することも必要になります。 デフォルト：キャッシュのリフレッシュなし 関連項目：『Oracle TopLink 開発者ガイド』の、キャッシュのリフレッシュの構成に関する項	記述子レベルでのキャッシュのリフレッシュはできるだけ避け、かわりに次の構成を検討してください。 ■ 問合せごとのキャッシュのリフレッシュ ■ キャッシュの有効期限 ■ 分離キャッシュ

注意： デフォルトでは、TopLink は、アプリケーションがそのアプリケーションで使用しているデータに排他アクセスしている（つまり、TopLink 以外にデータを変更する外部アプリケーションは存在しない）と想定しています。アプリケーションがデータに排他アクセスしていない場合は、表 4-5 のデフォルト設定を一部変更する必要があります。

TopLink 永続性マネージャおよびキャッシュとともに使用される CMP2.1 のデフォルト設定は、ロックなし、キャッシュのリフレッシュなし、キャッシュ使用一致なしです。排他アクセスができないときに、アプリケーションがキャッシュから失効データを読み取らないようにし、必要なデータ整合性レベルを維持するには、分離レベルに関連するこれらの設定や他の設定を適切に構成する必要があります。

表 4-6 に示すロック・モードを、TopLink のキャッシュ使用オプションおよび問合せリフレッシュ・オプションとともに使用すると、CMP を使用する EJB エンティティ Bean のデータ整合性が保証されます。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最新データとデータ整合性のための機能上の要件を優先して、これらのオプションの設定を決定します。

表 4-6 ロック方針

ロック・オプション	説明	パフォーマンス上の注意
ロックなし	ユーザーは、別のユーザーの変更内容を上書きできます。これはデフォルトのロック・モードです。Bean が同時更新されることがない場合や、read-committed セマンティクスによる、同じ行に対する同時読取りおよび更新が十分である場合は、このモードを使用します。 関連項目：『Oracle TopLink 開発者ガイド』の、ロック方針の構成に関する項と記述子およびロックの概要に関する項	通常、ロックなしのほうが高速ですが、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。
コミット時	すべてのユーザーがデータに読取りアクセスします。ユーザーがデータを変更しようとすると、そのユーザーがデータを読み取った後にデータが変更されていないか、アプリケーションによってチェックされます。 関連項目：『Oracle TopLink 開発者ガイド』の、ロック方針の構成に関する項と記述子およびロックの概要に関する項	
即時	更新目的でデータにアクセスした最初のユーザーによって、更新処理が完了するまでデータがロックされます。 関連項目：『Oracle TopLink 開発者ガイド』の、ロック方針の構成に関する項と記述子およびロックの概要に関する項	同じ行に対して頻繁に同時更新を実行することが予想される場合、同時アクセスの例外と再試行が多数発生するコミット時ロックより、即時ロックのほうが高速になることがあります。Bean レベルで即時ロックを使用する場合、最高のパフォーマンスを実現するには、分離キャッシュとともに使用することをお薦めします。
読取り専用	CMP Bean の記述子を読取り専用に設定すると、エンティティ Bean が変更不能になり、TopLink で作業ユニットのパフォーマンスを最適化することができます。 関連項目：『Oracle TopLink 開発者ガイド』の、ロック方針の構成に関する項と記述子およびロックの概要に関する項	Bean を読取り専用として定義すると、TopLink によって作業ユニットのパフォーマンスが最適化されるため、読取り専用として定義されておらず、挿入、更新、削除を行わない Bean よりパフォーマンスが向上します。

4.3.2.3 CMP リソースの監視

EJB メソッドの実行に著しく時間がかかっていないかを確認するには、`oc4j_ejb_method` という DMS メトリック表のタイプとメトリック `wrapper.avg` または `client.avg` をチェックします（メソッドのなかには、完了までに長時間かかることが予想されるものもあるので、異常に大きな値がないかを調べます）。たとえば、`ejbCreate`、`create`、`findAll` の各メソッドや、アプリケーション固有の EJB メソッドのメトリック値をチェックします。

また次の手順に従って、Application Server Control コンソールに表示されるメトリックをチェックすることで、レスポンス時間やトランザクション / 秒が時間の経過とともにどのように推移しているかを確認できます。

1. OC4J ホーム・ページから、「アプリケーション」サブタブを選択します。
2. 表の「すべてのアプリケーション」で、監視するアプリケーションを選択します。
3. 「**モジュール**」表で、監視する EJB モジュールを選択します。

また、CMP TopLink のメトリックを参照して、CMP のパフォーマンスに関する追加情報を確認することもできます。

注意： TopLink 関連の DMS メトリックの収集をオンにするには、DMS 構成の OC4J コマンドライン・プロパティ `-Doracle.dms.sensors` の値を、`Heavy` または `All` に設定する必要があります。

関連項目：

- 「OC4J の J2EE アプリケーション・メトリック」の表 D-11（D-6 ページ）
- 使用可能なメトリックの詳細は、『Oracle TopLink 開発者ガイド』の表 11-1「TopLink DMS メトリック」を参照してください。

5

PL/SQL のパフォーマンスの最適化

この章では、Web アプリケーションの PL/SQL パフォーマンスの向上に関する情報の参照先について説明します。大部分の情報は、『Oracle Application Server mod_plsql ユーザーズ・ガイド』に記載されています。

関連項目：

- 『Oracle Application Server mod_plsql ユーザーズ・ガイド』（PL/SQL パフォーマンスの最適化について）
- 付録 C「パフォーマンス・メトリック」（`mod_plsql` メトリックの詳細について）
- 『Oracle HTTP Server 管理者ガイド』（DAD パラメータの詳細について）
- 『Oracle Application Server PL/SQL Web Toolkit リファレンス』（Oracle データベース・サーバーに格納されている PL/SQL プロシージャとして、Web アプリケーションの開発を可能にする PL/SQL Web Toolkit について）

6

Oracle HTTP Server の最適化

この章では、Oracle 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、次の項が含まれています。

6.1 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、制限およびタイムアウトを指定します。

表 6-1 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 6-1 Oracle HTTP Server の構成プロパティ

ディレクティブ	説明
ListenBackLog	保留中の接続のキューの最大長を指定します。通常、チューニングする必要はありません。一部のオペレーティング・システムでは、ListenBacklog で指定した数値そのものではなく、設定に基づく数値（通常はそれより大きな数値）が使用されます。 デフォルト値 : 511
MaxClients	実行可能なサーバーの総数、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。クライアントの接続数がこの制限に達した場合、後続のリクエストは、ListenBackLog ディレクティブで指定したキューの最大長に達するまで、TCP/IP システムのキューに格納されます（保留中の接続のキューが一杯になると、プロセスが使用可能になるまで新しいリクエストは接続エラーになります）。 MaxClients に使用できる最大数は、8192（8K）です。 デフォルト値 : 150
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、UNIX 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、MaxRequestsPerChild に 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 keep alive 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。 **注意** : Windows システムでは、MaxRequestsPerChild は常に 0（無制限）に設定する必要があります。Windows 上に存在するサーバー・プロセスは 1 つのみなので、このプロセスを制限しないでください。
MaxSpareServers MinSpareServers	サーバーのプール・サイズの規定値です。Oracle HTTP Server では、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測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が維持されます。 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。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

表 6-1 Oracle HTTP Server の構成プロパティ（続き）

ディレクティブ	説明
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

6.1.1 永続的な接続による httpd プロセスの可用性の低下

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、KeepAlive ディレクティブのデフォルト設定値を小さくするか、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。

6.2 Oracle HTTP Server のロギング・オプション

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。

6.2.1 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。

6.2.2 HostNameLookups ディレクティブの構成

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。

6.2.3 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ける静的ページのパフォーマンスでは、warn、info および debug のレベルで違いはありませんでした。

動的リソースを使用するリクエストの場合（たとえば mod_osso、mod_plsql、または mod_oc4j を使用するリクエスト）、debug レベルなどのより高位のデバッグ・レベルの設定によりパフォーマンス・コストが発生します。

6.3 Oracle HTTP Server のセキュリティ・パフォーマンスの考慮事項

この項には、次の項目が含まれています。

- Oracle HTTP Server における Secure Sockets Layer（SSL）のパフォーマンスに関する問題
- Oracle HTTP Server ポート・トンネリングのパフォーマンスに関する問題

6.3.1 Oracle HTTP Server における Secure Sockets Layer（SSL）のパフォーマンスに関する問題

Secure Sockets Layer（SSL）は、Netscape 社によって開発されたプロトコルで、インターネット上での認証と暗号化による通信を提供します。概念的には、SSL はプロトコル・スタックのアプリケーション層とトランスポート層の中間に存在します。SSL は技術的にはアプリケーションに依存しないプロトコルですが、HTTP を介したセキュリティの提供が標準となった現在では、すべての主要な Web ブラウザで SSL がサポートされています。

SSL は、Web ベースのアプリケーションのレスポンス能力とスケーラビリティの両方でボトルネックとなる可能性があります。SSL が要求される場合、プロトコルのパフォーマンスに関する問題を慎重に考慮する必要があります。SSL プロトコルを使用する場合、特にセッションの作成および始動などのセッション管理が、最もパフォーマンス・コストのかかる部分です。

この項には、次の SSL パフォーマンス関連の情報が含まれています。

- Oracle HTTP Server の SSL キャッシュ
- SSL によるアプリケーション・レベルのデータ暗号化
- SSL パフォーマンスの推奨事項

関連項目：『Oracle Application Server セキュリティ・ガイド』

6.3.1.1 Oracle HTTP Server の SSL キャッシュ

SSL 接続が開始されると、クライアントとサーバー間のハンドシェイクに基づくセッションが発生します。これには暗号スイートのネゴシエーション、データ暗号化用の秘密鍵の交換、およびデジタル署名証明書を介したサーバーおよび任意のクライアントの認証が含まれます。

SSL のセッション状態がクライアントとサーバー間で開始された後は、サーバーによってセッション状態を保存および再利用できるので、後続の SSL リクエストでセッションを確立しハンドシェイクを作成する必要がなくなります。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、待ち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。

httpd.conf の SSLSessionCacheTimeout ディレクティブにより、サーバーで保存された SSL セッションを保持する期間を指定します（デフォルトは 300 秒です）。セッション状態は、指定した時間の経過後に使用されない場合は廃棄されます。そして、後続の SSL リクエストでは新しい SSL セッションを確立し、もう一度ハンドシェイクを開始する必要があります。SSLSessionCache ディレクティブでは、SSL セッション情報の保存場所を指定します。デフォルトの保存場所は、UNIX システムでは $ORACLE_HOME/Apache/Apache/logs/ ディレクトリで、Windows システムでは %ORACLE_HOME%¥Apache¥Apache¥logs¥ です。保存されたセッション・キャッシュ・ファイルは、複数の Oracle HTTP Server プロセスで使用可能です。

SSL のセッション状態を保存すると、SSL を使用したアプリケーションのパフォーマンスが大幅に改善します。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。

保存された SSL のセッション状態を再利用する場合は、いくつかのパフォーマンス・コストが発生します。SSL のセッション状態をディスクに保存した場合、保存されたセッション状態を再利用するには、通常ディスクから関連するセッション状態を検出し、取得する必要があります。永続的な HTTP 接続を使用すれば、このパフォーマンス・コストを削減できます。永続的な HTTP 接続がクライアント側でサポートされている場合、Oracle HTTP Server では、この接続がデフォルトで使用されます。Oracle HTTP Server によって実装される SSL を使用した HTTP 接続では、SSL のセッション状態は、関連する HTTP 接続が維持されている間メモリーに保存されます。このプロセスにより、SSL セッションの再利用によるオーバーヘッドが実質的に排除されます（概念的には、SSL 接続は HTTP 接続とともに維持されます）。

6.3.1.2 SSL によるアプリケーション・レベルのデータ暗号化

SSL を使用するほとんどのアプリケーションでは、データ暗号化のコストは、SSL セッション管理と比較して小さくなります。暗号化コストが著しく発生するのは、暗号化されたデータのボリュームが大きい場合です。そのような場合、データ暗号化アルゴリズムと SSL セッションに選択した鍵のサイズはきわめて大きくなります。

通常、セキュリティ・レベルとパフォーマンス間にはトレードオフが発生します。たとえば、最新のプロセッサでは、RSA は RC4 暗号化が 1 出力バイト当たり 8 ～ 16 の処理に対応すると見積っています。通常の DES 暗号化では RC4 のおよそ 8 倍のオーバーヘッドが発生し、Triple-DES では DES のおよそ 25 倍のオーバーヘッドが発生します。ただし、Triple-DES を使用している場合、暗号化コストはほとんどのアプリケーションで顕著ではありません。Oracle HTTP Server では、これら 3 つの暗号スイートおよびその他の暗号スイートがサポートされています。

Oracle HTTP Server では、`httpd.conf` で指定されている `SSLCipherSuite` 属性に基づき、クライアントと暗号スイートをネゴシエートします。

> **関連項目：** サポートされている暗号スイートの使用方法の詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

6.3.1.3 SSL パフォーマンスの推奨事項

次の推奨事項は、Oracle HTTP Server を SSL とともに使用する場合のパフォーマンス要件を決定する際に役立ちます。

1. SSL ハンドシェイクは、CPU 使用量およびレスポンス時間の両方の点から本質的にコストのかかる処理です。したがって、必要な場合のみ SSL を使用してください。セキュリティを必要とするアプリケーションの部分と要求されるセキュリティ・レベルを決定し、必要なセキュリティ・レベルでこれらの部分のみを保護します。SSL の使用を控え、可能なかぎりセッション状態を再利用して、SSL ハンドシェイクの必要性を最小限にします。たとえば、ページ上に少量の機密データと多量の機密ではないグラフィック画像がある場合、SSL を使用して機密データのみを転送し、通常の HTTP を使用して画像を転送します。アプリケーションでサーバー認証のみを要求する場合、クライアント認証は使用しません。アプリケーションのパフォーマンス目標がこの方法だけでは実現できない場合、追加のハードウェアが必要な場合もあります。
2. SSL を効果的に使用できるようにアプリケーションを設計します。セキュアな操作をグループ化して、SSL セッションおよび SSL 接続を再利用します。
3. 可能な場合は永続的な接続を使用して、SSL セッションの再利用のコストを最小限にします。

4. セッション・キャッシュのタイムアウト値をチューニングします（httpd.conf の SSLSessionCacheTimeout 属性）。SSL セッション・キャッシュを維持するコストと、新しい SSL セッションを確立するコストの間に、トレードオフが存在します。一般に、保護されたビジネス・プロセスまたは SSL 交換の概念的なグループ化は、複数のセッションを作成せずに完了する必要があります。SSLSessionCacheTimeout 属性のデフォルト値は、300 秒です。アプリケーションの活用性をテストして、この設定のチューニングに役立ててください。
5. 大量のデータが SSL を介して保護されている場合、使用されている暗号スイートに細心の注意を払ってください。httpd.conf で指定されている SSLCipherSuite ディレクティブによって、暗号スイートを制御します。より低いレベルのセキュリティが許容される場合、より小さいサイズの鍵を使用する暗号強度の低いプロトコルを使用します（これによってパフォーマンスが大幅に改善します）。最後に、目的のセキュリティ・レベルを確保するために使用可能な暗号スイートをそれぞれ使用してアプリケーションをテストし、最もパフォーマンスの高い暗号スイートを検証します。
6. 前述の考慮事項に配慮しても、SSL が引き続きアプリケーションのパフォーマンスとスケーラビリティのボトルネックである場合、ハードウェア・クラスタに複数の Oracle HTTP Server インスタンスを配置するか、SSL アクセラレータ・カードの使用を検討してください。

6.3.2 Oracle HTTP Server ポート・トンネリングのパフォーマンスに関する問題

OracleAS Port Tunneling が構成されている場合、すべてのリクエストは OracleAS Port Tunneling インフラストラクチャを経由して処理されます。つまり OracleAS Port Tunneling は、Oracle HTTP Server のリクエスト処理に関わるパフォーマンスおよびスケーラビリティ全体に大きく影響します。

実行される OracleAS Port Tunneling プロセスの数を除き、OracleAS Port Tunneling のパフォーマンスは自動的に調整されます。唯一可能なパフォーマンス制御は、より多くの OracleAS Port Tunneling プロセスを開始することです。これにより、使用可能な接続の数が増え、システムのスケーラビリティが増大します。

OracleAS Port Tunneling プロセスの数は、プロセスが必要とされる度合いや期待される接続数に基づきます。プロセスを追加するたびに、DMZ とイントラネット間のファイアウォールを介して新しいポートをオープンする必要があるので、この数を自動的に決定することはできません。オープンしたポートよりも多くのプロセスは開始できません。また、オープンしたポートよりも少ないプロセスも好ましくありません。この場合、プロセスが割り当てられないポートが存在することになります。

OracleAS Port Tunneling のパフォーマンスを測定するには、OracleAS Port Tunneling インフラストラクチャを経由するサーブレット・リクエストのリクエスト時間を判断します。OracleAS Port Tunneling を使用した場合とそうでない場合の Oracle Application Server インスタンスのレスポンス時間を比較して、OracleAS Port Tunneling によってパフォーマンス要件が満たされているかどうかを確認する必要があります。

> **関連項目：** OracleAS Port Tunneling の構成の詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

6.4 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

6.4.1 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、着信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。

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。

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

状況によっては、Oracle Containers for J2EE（OC4J）のリクエストの平均処理時間と、ユーザーが経験する平均レスポンス時間に大きな開きがあるかもしれません。OC4J での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。

OC4J のリクエスト処理時間と平均レスポンス時間の差が大きい場合は、6-2 ページの「Oracle HTTP Server ディレクティブの構成」に記述されている Oracle HTTP Server ディレクティブをチューニングしてください。

6.4.3 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を受信したときに、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。この場合、より正確な PL/SQL モジュールの平均サービス時間を取得するには、`handle.maxTime` 値を増加させていたデータベース接続のオープンにかかる時間を除外して、次のように平均を再計算します。

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

たとえば、この場合は次のようになります。

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

7

Oracle BPEL Process Manager のパフォーマンス・チューニング

Oracle BPEL Process Manager には多数のプロパティ設定が用意されています。これらのプロパティを構成して、プロセス、ドメイン、アプリケーション・サーバー、Java 仮想マシン(JVM)、デハイドレーション・ストア・データベースなどのレベルでパフォーマンスを最適化できます。この章では、これらのプロパティ設定および推奨される使用方法について説明しています。

この章には、次の項が含まれています。

7.1 パフォーマンス・チューニングの概要

この項では、主要な Oracle BPEL Process Manager チューニングの概要について説明します。なんらかのプロパティ設定を行う前に、この項の内容を確認してください。

この項には、次の項目が含まれています。

7.1.1 ドメインおよびプロセス構成プロパティの設定

Oracle BPEL Process Manager では、ドメインおよびプロセスの構成プロパティを次の 2 つのレベルで設定できます。

- ドメイン・レベル : 特定のドメインにデプロイされたすべてのプロセスを構成できます。
- プロセス・レベル : 特定のドメインで、構成するプロセスと構成しないプロセスを指定できます。ドメイン・レベルの設定が、プロセス・レベルの同じ設定と競合する場合、プロセス・レベルの設定が優先的に使用されます。

7.1.2 永続プロセスおよび一時プロセス

Oracle BPEL Process Manager は、非同期コールバックを待機しながら、デハイドレーション・ストア・データベースを使用して、長期の非同期プロセスおよびそれらの現在の状態情報を管理します。データベースにプロセスを保存することでプロセスが維持され、システムの停止やネットワーク障害が発生した場合に、状態や信頼性の損失を回避できます。Oracle BPEL Process Manager には、2 種類のプロセスがあります。これらのプロセスは、デハイドレーション・ストア・データベースに、それぞれ異なる影響を与えます。

- 一時プロセス : このタイプでは、プロセスの実行中に中間デハイドレーション・ポイントは作成されません。プロセスの実行中に未対応の障害またはシステム停止時間が発生した場合、一時プロセスのインスタンスではシステムにトレースが残りません。一時プロセスのインスタンスは、(正常終了、異常終了にかかわらず) 進行中に保存できません。一時プロセスは通常、短期間で終了する、リクエスト / レスポンス形式のプロセスです。一時プロセスの例には、Oracle JDeveloper で設計した同期プロセスがあります。
- 永続プロセス : このタイプでは、次に示すアクティビティにより、実行中に 1 つ以上のデハイドレーション・ポイントがデータベースに作成されます。
  - receive アクティビティ
  - pick アクティビティの onMessage ブランチ
  - pick アクティビティの onAlarm ブランチ
  - wait アクティビティ

  永続プロセスのインスタンスは、(正常終了、異常終了にかかわらず) 進行中に保存できます。永続プロセスは通常、長時間稼動し、一方向呼出しにより開始されます。メモリー不足やシステム停止時間の問題が発生するため、永続プロセスに対するメモリー最適化はできません。

7.1.3 一方向呼出しおよび双方向呼出し

BPEL プロセスのインスタンスの呼出しには次の 2 種類があります。

- 一方向呼出し : リクエストのみの操作。インバウンド・メッセージのみ含まれます。
- 双方向呼出し : リクエスト / レスポンス操作。コール元スレッドは、レスポンスの準備ができるまでブロックされます。

表 7-1 は、一方向呼出しと双方向呼出しの用途について説明しています。

表 7-1 一方向呼出しおよび双方向呼出し

用途	一方向呼出し	双方向呼出し

WSDL ファイルの定義	`<operation name="oneway">`
	`<input message="in"/>`
	`</operation>`
		`<operation name="twoway">`
		`<input message="in"/>`
		`<output message="out"/>`
		`</operation>`

BPEL アクティビティでの変数定義	`<receive operation="oneway"`
	`variable="in"/>`
		`<receive operation="twoway"`
		`variable="in"/>`
		`...`
		`<reply operation="twoway"`
		`variable="out"/>`

配信サービスを利用するかどうか	リクエストは配信サービスに保存されます。メッセージが目的のインスタンスに配信されるまで、コール元スレッドはブロックされません。	リクエストは、Oracle BPEL Server、および目的の BPEL インスタンスに配信されます。コール元スレッドは、レスポンスの準備ができるまでブロックされます。

7.1.4 多重呼出し不変なアクティビティ

多重呼出し不変なアクティビティとは、再試行が可能なアクティビティ（たとえば、assign アクティビティや invoke アクティビティなど）です。Oracle BPEL Server では、多重呼出し不変でないアクティビティの実行後にインスタンスが保存されます。

> **関連項目：** 詳細は、7-7 ページの「idempotent BPEL プロパティ」を参照してください。

7.1.5 進行中のデータベース記憶域

BPEL インスタンスのライフ・サイクル全体にわたり、実行中の各時点の状態で BPEL インスタンスをデハイドレーション・ストア・データベースに複数回保存できます。これは次の 2 つの場合に実行されます。

- インスタンスがイベントを待機しているとき。これはアラーム、または呼出しメッセージのどちらかです。次に示す BPEL アクティビティのいずれかが実行中の場合に発生します。
  - wait アクティビティ
  - pick アクティビティの onAlarm ブランチ
  - receive アクティビティ
  - pick アクティビティの onMessage ブランチ

  BPEL インスタンスがデハイドレーション・ストア・データベースに保存されると、このインスタンスはデハイドレーションされたとみなされます。この後にイベントが発生した場合（アラーム時刻がきた、またはメッセージが到着したなど）、インスタンスはデータベースから読み込まれ、実行が再開されます。
- 多重呼出し不変でないアクティビティの後。手順を再試行する場合には、インスタンス記憶域が必要です。多重呼出し不変でないアクティビティ後の手順から再試行されます。

7.1.6 双方向呼出しの JTA トランザクション

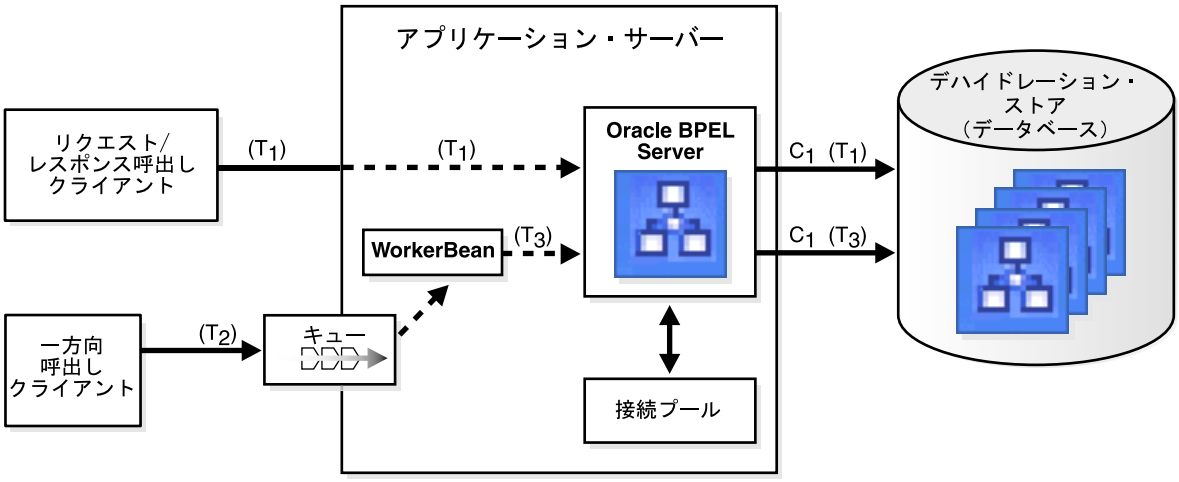
双方向呼出しでは、呼び出されるプロセスが一時プロセスである場合、Oracle BPEL Server ではコール元の Java Transaction API（JTA）トランザクションが実行されます。呼び出されるプロセスが永続プロセスである、つまり進行中にデータベースの保存が行われる場合、Oracle BPEL Server により新しいトランザクションが作成されます。

7.1.7 BPEL スレッド・モデル

なんらかのプロパティ設定を行う前に、この項で BPEL スレッド・モデルの詳細を確認してください。

図 7-1 は、リクエスト / レスポンスおよび一方向のプロセス・インスタンスの呼出しにおけるスレッドの処理を示しています。

図 7-1 スレッドの処理



7.1.7.1 リクエスト / レスポンス呼出し

図 7-1 のクライアントは、スレッド T1 を実行しています。コール元がプロセス・インスタンスを開始すると、同じスレッドが処理に使用されます。最終的に、データベース操作が必要になったときに、このスレッドは接続プールからデータベース接続（図 7-1 の C1）を取得します。

7.1.7.2 一方向呼出し

図 7-1 の一方向呼出しクライアントは、スレッド T2 を実行しています。クライアントがプロセス・インスタンスを開始すると、呼出しリクエストがキューに置かれます。この時点で、スレッド T2 は Oracle BPEL Server から解放され、コール元は自身の処理を継続できるようになります。Oracle BPEL Server の内部では、メッセージドリブン Bean（MDB）の WorkerBean がキューの呼出しリクエストを監視しています。メッセージがデキューされると、Oracle BPEL Server はメッセージを処理するために、別のスレッド（T3）を割り当てます。このスレッドは、インスタンスを処理するために Oracle BPEL Process Manager によって使用されます。データベース操作が必要な場合に、このスレッドは接続プールからデータベース接続を取得します。

関連項目： WorkerBean の詳細は、7-23 ページの「Oracle BPEL Server の EJB 構成」を参照してください。

7.1.7.3 スレッドと接続プールの関係

図 7-1 と前述の項から、スレッド・パラメータおよび接続プール・パラメータの適切な設定について、重要な関係を導き出すことができます。

処理中の同時インスタンスの数は、リクエスト / レスポンス・クライアントのリクエスト数と、割り当てられた WorkerBean スレッドの数によって決まります。この場合、次のような関係が成り立ちます。

DB 接続数の最大値 >= (WorkerBean リスナー・スレッド) + (同時に呼び出せるリクエスト・レスポンスの最大値)

dspMaxThreads プロパティにより、各ドメインに WorkerBean スレッドが割り当てられます。これにより、次の関係が成り立ちます。

$\Sigma_{domains}$ dspMaxThreads = (WorkerBean リスナー・スレッド)

DB 接続数の最大値 >= ($\Sigma_{domains}$ dspMaxThreads) + (同時に呼び出せるリクエスト・レスポンスの最大値)

存在するドメインが 1 つのみの場合、前述の式は次のように簡略化できます。

dspMaxThreads = (WorkerBean リスナー・スレッド)
DB 接続数の最大値 >= (dspMaxThreads) + (同時に呼び出せるリクエスト・レスポンスの最大値)

7.2 プロセス・レベルのパフォーマンス設定

この項では、プロセス・レベルのパフォーマンス・チューニング・プロパティについて説明します。

プロセス・レベルのパフォーマンス・プロパティは、目的の BPEL プロセスで使用される bpel.xml ファイルで設定します。このファイルは、プロセスの .bpel ファイルと同じディレクトリにあります。新しい設定を有効にするには、*JDev_Oracle_Home*¥jdev¥mywork¥*workspace_name*¥*process_name*¥bpel¥bpel.xml ファイルで設定を変更した後、プロセスを再デプロイする必要があります。

注意： これらのプロパティは、Oracle JDeveloper のデプロイメント・ディスクリプタのプロパティ・ウィンドウでも設定できます。

関連項目： 「ドメインおよびプロセス構成プロパティの設定」(7-2 ページ)

7.2.1 completionPersistLevel BPEL プロパティ

このプロパティは、インスタンスの完了後に保存されるデータの型（および量）を制御します。

プロセス・インスタンスが完了すると、デフォルトでは、Oracle BPEL Server によりプロセスの最終状態（変数値など）が保存されます。完了後にその値を保存する必要がない場合、インスタンスのメタデータ（完了状態、開始日、終了日など）のみが保存されるように、このプロパティを設定できます。このプロパティは、一時 BPEL プロセスに適用されます。

このプロパティは、inMemoryOptimization パフォーマンス・プロパティが true に設定されている場合のみ使用されます。completionPersistLevel プロパティは、completionPersistPolicy プロパティとともに使用します。

このプロパティの設定は、データベース（特に、cube_instance 表、cube_scope 表および work_item 表）の増大に大きな影響を与える場合があります。これはスループットにも影響します（I/O が低下するため）。

関連項目：

- 「completionPersistPolicy BPEL プロパティ」（7-6 ページ）
- 「inMemoryOptimization BPEL プロパティ」（7-8 ページ）
- cube_instance 表、cube_scope 表および work_item 表の詳細は、7-10 ページの表 7-2 を参照してください。

値

このプロパティの値を次に示します。

- all（デフォルト）：Oracle BPEL Server により、最終的な変数値、作業アイテム・データおよび監査データを含む、終了インスタンスが保存されます。この値を設定すると、データベースのサイズは増大します。
- instanceHeader：Oracle BPEL Process Manager により、インスタンス・メタデータのみが保存されます。

例

次の例では、失敗したインスタンスのみが永続化されます（completionPersistPolicy=faulted）。失敗したインスタンスについては、インスタンスに関連する変数値がすべて保存されます（competionPersistLevel=All）。

```
<BPELSuitcase>
  <BPELProcess src="HelloWorld.bpel" id="HelloWorld">
    ...
    <configurations>
      <property name="inMemoryOptimization">true</property>
      <property name="completionPersistPolicy">faulted</property>
      <property name="completionPersistLevel">All</property>
    </configurations>
  </BPELProcess>
</BPELSuitcase>
```

7.2.2 completionPersistPolicy BPEL プロパティ

このプロパティは、インスタンスを永続化するかどうか、また、どのような場合に永続化するかを制御します。保存されないインスタンスは、Oracle BPEL Control に表示されません。このプロパティは、一時 BPEL プロセスに適用されます。

このプロパティは、inMemoryOptimization が true に設定されている場合のみ使用されます。completionPersistPolicy に off 以外の値を設定した場合、completionPersistLevel を設定し、保存する永続データをより厳密にチューニングできます。

このパラメータは、データベース（特に、cube_instance 表、cube_scope 表および work_item 表）に格納されるデータの量に大きな影響を与えます。また、スループットにも影響します。

関連項目：

- 「completionPersistLevel BPEL プロパティ」（7-5 ページ）
- 「inMemoryOptimization BPEL プロパティ」（7-8 ページ）

値

このプロパティの値を次に示します。

- on（デフォルト）: 終了したインスタンスは通常に保存されます。
- deferred: 終了したインスタンスは、異なるスレッドで別のトランザクション内で保存されます。サーバーに障害が発生すると、一部のインスタンスは保存されない可能性があります。
- faulted: 失敗したインスタンスのみ保存されます。
- off: インスタンス（およびそのデータ）は保存されません。

例

次の例では、completionPersistPolicy が deferred に設定されています。

```
<BPEL Suitcase>
   <BPELProcess src="HelloWorld.bpel" id="HelloWorld">
      . . .
      <configurations>
         <partnerLinkBinding name="PartnerService">
            <property name="inMemoryOptimization">true</property>
            <property name="completionPersistPolicy">deferred</property>
         </partnerLinkBinding>
      </configurations>
    </BPELProcess>
 </BPEL Suitcase>
```

7.2.3 idempotent BPEL プロパティ

BPEL の invoke アクティビティは、デフォルトでは多重呼出し不変なアクティビティです。つまり、BPEL プロセスでは、invoke アクティビティの呼出し直後に、インスタンスのデハイドレーションを行いません。したがって、idempotent が true に設定されている状態で、invoke アクティビティの実行直後に Oracle BPEL Server に障害が発生した場合、再起動後にもう一度この invoke が実行されます。これは、invoke アクティビティの実行が記録されていないためです。このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

idempotent が false に設定されている場合、invoke アクティビティは実行直後にデハイドレーションされ、デハイドレーション・ストアに記録されます。その後、Oracle BPEL Server に障害が発生して再起動されても、invoke アクティビティは繰り返されません。Oracle BPEL Process Manager では、この invoke が実行済であると認識されるためです。

idempotent を false に設定すると、フェイルオーバー時の保護機能は向上しますが、BPEL プロセスがデハイドレーション・ストアにアクセスする頻度が高まるため、パフォーマンスがある程度犠牲になります。この設定は、bpel.xml ファイル内の各パートナ・リンクに対して構成可能です。

このパラメータを true に設定すると、スループットは大幅に向上する可能性があります。しかし、前述のように、サーバーに障害が発生した場合にパートナのサービスが安全に再試行できることを確認する必要があります。このプロパティを true に設定できる例としては、読取り専用サービス（たとえば、CreditRatingService）やインスタンスのトランザクションを共有するローカルの EJB/WSIF 起動などがあります。

値

このプロパティの値を次に示します。

- false: アクティビティは実行直後にデハイドレーションされ、デハイドレーション・ストアに記録されます
- true（デフォルト）: Oracle BPEL Server に障害が発生した場合、再起動後に、このアクティビティがもう一度実行されます。これは、アクティビティの呼出し直後にサーバーがデハイドレーションされず、アクティビティの実行が記録されていないためです。

例

次の bpel.xml ファイルの例に、idempotent プロパティを示します。この例は、デハイドレーション・ストア・データベースに保存される一方向呼出しメッセージを示しています。このプロパティは、各パートナ・リンクに設定できます。

```
<BPELSuitcase>
  <BPELProcess src="Invoke.bpel" id="Invoke">
    <partnerLinkBindings>
      . . .
      <partnerLinkBinding name="PartnerService">
        <property name="wsdlLocation">
          partner-wsdl
        </property>
        <property name="idempotent">false</property>
      </partnerLinkBinding>
    </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>
```

7.2.4 inMemoryOptimization BPEL プロパティ

このプロパティにより、処理対象のプロセスが一時プロセスであり、インスタンスのデハイドレーションが必要ないことを示します。true に設定されている場合、Oracle BPEL Server では、そのプロセスのインスタンスが実行中のみメモリーに保持されます。このプロパティは、一時プロセス（つまり、中間プロセスの receive アクティビティ、pick アクティビティまたは wait アクティビティを一切含まないもの）に対してのみ、true に設定できます。

このプロパティのデフォルトは false です。同期 BPEL プロセスのインスタンスは完全に永続化され、デハイドレーション・ストア・データベースに記録されます。

inMemoryOptimization を true に設定すると、デハイドレーションは解除され、インスタンスはメモリー内にのみ保持されます。永続性動作を判断するために、completionPersistPolicy プロパティと completionPersistLevel プロパティの設定も調査されます。inMemoryOptimization プロパティを true に設定すると、スループットを向上させることができます。さらに、このプロパティを前述の 2 つのプロパティとともに使用した場合、データベースの増大を最小限に抑えることができます。

関連項目：

- 「completionPersistLevel BPEL プロパティ」（7-5 ページ）
- 「completionPersistPolicy BPEL プロパティ」（7-6 ページ）

値

このプロパティの値を次に示します。

- false（デフォルト）: 同期 BPEL プロセスのインスタンスは完全に永続化され、デハイドレーション・ストア・データベースに記録されます。
- true: インスタンスはメモリー内にのみ保持されます。

例

次の bpel.xml ファイルの例に、同期 Hello World BPEL プロセスの inMemoryOptimization プロパティを示します。

```
<BPEL Suitcase>
   <BPELProcess src="HelloWorld.bpel" id="HelloWorld">
      . . .
      <configurations>
            <property name="inMemoryOptimization">true</property>
      </configurations>
    </BPELProcess>
 </BPEL Suitcase>
```

7.2.5 nonBlockingInvoke BPEL プロパティ

このプロパティを使用すると、flow アクティビティまたは flowN アクティビティで複数のブランチを実行するときのパフォーマンスを向上することができます。デフォルトでは、Oracle BPEL Process Manager は単一のスレッドで実行されます。つまり、複数のブランチはパラレルではなく、順番に実行されます。このプロパティを `true` に設定すると、各ブランチの invoke アクティビティをパラレルで実行するための新しいスレッドが作成されます。この設定は、`bpel.xml` ファイル内の各パートナ・リンクに対して構成可能です。このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

複数の flow ブランチや flowN ブランチに invoke アクティビティが存在する場合は、このパラメータを `true` に設定することを検討してください。これは、パラレルの invoke アクティビティが双方向の際には特に有効ですが、パラレルの一方向呼出しに対しても効果的な場合があります。

値

このプロパティの値を次に示します。

- `true`：呼出しを実行するための新しいスレッドが生成されます。このスレッドは、基本的には `InvokerBean` メッセージドリブン Bean のスレッドです。このプロセスに、ブロックしていない他の invoke アクティビティが存在する場合は、`InvokerBean` スレッドの値を増加します。また、次のように接続プールの最大サイズも増加する必要がある場合があります。

  接続プールのサイズ >= (InvokerBean リスナー・スレッド + WorkerBean リスナー・スレッド + 同時に呼び出せるリクエスト・レスポンスの最大値)

- `false`（デフォルト）：invoke アクティビティは単一のプロセス・スレッドで実行されます。

  関連項目： `InvokerBean` の構成手順については、7-23 ページの「InvokerBean」を参照してください。

例

次の `bpel.xml` ファイルの例では、`nonBlockingInvoke` プロパティを有効化しています。

```
<BPELSuitcase>
  <BPELProcess src="Invoke.bpel" id="Invoke">
    <partnerLinkBindings>
      . . .
      <partnerLinkBinding name="PartnerService">
        <property name="wsdlLocation">
          partner-wsdl
        </property>
        <property name="nonBlockingInvoke">true</property>
      </partnerLinkBinding>
    </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>
```

7.3 インスタンス・データの増大に影響される表

インスタンス・データは、Oracle BPEL Process Manager スキーマ表の領域を占有します。表 7-2 は、インスタンス・データの増大による影響を受ける表の一覧です。それぞれの表について、簡単に説明しています。これらの表は、7-11 ページの「ドメイン・レベルのパフォーマンス・チューニング」で説明されているドメイン・レベルのパフォーマンス・プロパティに設定した値による影響を受けて増大します。

表 7-2 インスタンス・データの増大に影響される Oracle BPEL Process Manager の表

表の名前	表の説明
audit_details	API を通じてログに記録される監査詳細を格納します。assign アクティビティなどでは、デフォルトで、監査詳細として変数がログに記録されます。この動作の設定には、「**BPEL ドメインの管理**」→「**構成**」にある **auditLevel** プロパティを使用します。 監査詳細はサイズが大きいため、audit_trail 表から分離されています。詳細を表示するには、Oracle BPEL Control で目的のインスタンスの「**監査**」タブにあるリンクをクリックし、別途、詳細をロードします。この表では、Oracle BPEL Control の「**BPEL ドメインの管理**」→「**構成**」にある **auditDetailThreshold** プロパティが使用されます。詳細のサイズがこのプロパティに指定された値より大きな場合、この表に保存されます。それ以外の場合は、audit_trail 表に保存されます。 **関連項目：**「auditDetailThreshold BPEL プロパティ」（7-11 ページ）および「auditLevel BPEL プロパティ」（7-12 ページ）
audit_trail	インスタンスの監査証跡を格納します。Oracle BPEL Control に表示される監査証跡は、XML 文書から作成されます。インスタンスを 1 つ処理するたびに、各アクティビティはイベントを XML として監査証跡に書き込みます。
cube_instance	プロセス・インスタンス・メタデータ（インスタンスの作成日、現在の状態、タイトル、プロセス識別子など）を格納します。
cube_scope	インスタンスのスコープ・データ（BPEL フローで宣言されている変数すべてや、フロー全体のルーティング・ロジックを支援する内部オブジェクトなど）を格納します。
dlv_message	受信時のコールバック・メッセージを格納します。この表には、メッセージのメタデータ（現在の状態、プロセス識別子、受信日など）のみが格納されます。
dlv_subscription	インスタンスの配信サブスクリプションを格納します。receive アクティビティや onMessage アクティビティのように、インスタンスがパートナからのメッセージを待っているときはいつでも、その特定の receive アクティビティに対してサブスクリプションが書き込まれます。
document_ci_ref	xml_document 表に格納されているデータへのキューブ・インスタンスの参照を格納します。
document_dlv_msg_ref	xml_document 表に格納されている invoke_message および dlv_message への参照を格納します。
invoke_message	受信（呼出し）メッセージ（インスタンスの作成を生じさせるメッセージ）を格納します。この表には、メッセージのメタデータ（現在の状態、プロセス識別子、受信日など）のみが格納されます。
schema_md	Oracle BPEL Process Manager スキーマ（orabpel）で定義された列に関するメタデータを格納します。
task	インスタンス用に作成されたタスクを格納します。TaskManager プロセスは、この表に現在の状態を保存します。
work_item	インスタンスにより作成されたアクティビティを格納します。BPEL フローのすべてのアクティビティに work_item 表が存在します。この表には、アクティビティのメタデータ（現在の状態、ラベルおよび（wait アクティビティで使用される）有効期限）が含まれます。
xml_document	システム内のすべてのラージ・オブジェクト（invoke_message ドキュメント、dlv_message ドキュメントなど）を格納します。この表はデータをバイナリ・ラージ・オブジェクト（BLOB）として保存します。メタデータとドキュメントの記憶域を分けることで、ドキュメントのサイズに関係なく柔軟にメタデータを変更できます。

7.4 ドメイン・レベルのパフォーマンス・チューニング

この項では、ドメイン・レベルのパフォーマンス・チューニング・プロパティについて説明します。

これらの設定は、Oracle BPEL Control の「**BPEL ドメインの管理**」→「**構成**」で変更することをお薦めします。Oracle BPEL Control では、既存の設定と新たに入力された設定が確認され、再起動せずにこれらの検証が行われます。ドメイン・レベルのパフォーマンス設定は、*SOA_Oracle_Home*¥bpel¥domains¥*domain_name*¥config¥domain.xml ファイルにあります。domain.xml ファイルを直接編集した場合、変更後の設定を有効にするには、Oracle BPEL Server を再起動する必要があります。

関連項目：「ドメインおよびプロセス構成プロパティの設定」（7-2 ページ）

7.4.1 編集できない Oracle BPEL Console のプロパティ

次のプロパティは、Oracle BPEL Control の「**BPEL ドメインの管理**」→「**構成**」に表示されます。「構成」タブでは、これらのプロパティの「**名前**」列と「**コメント**」列は空です。これらのプロパティはシステムのパフォーマンス・チューニングには影響がないため、編集しないでください。

- **cbCacheHighWatermark**
- **cbCacheLowWatermark**
- **cbCachePolicy**
- **cbCacheUnits**
- **instCacheUnits**
- **invCacheHighWatermark**
- **invCacheLowWatermark**
- **invCachePolicy**
- **invCacheUnits**
- **subCacheHighWatermark**
- **subCacheLowWatermark**
- **subCachePolicy**
- **subCacheUnits**

パフォーマンスを最適化するために設定可能なプロパティの詳細は、これ以降の項目を参照してください。

7.4.2 auditDetailThreshold BPEL プロパティ

このプロパティは、監査証跡詳細の文字列の最大サイズ（KB 単位）を設定します。この値を超えた文字列は、監査証跡とは別に保存されます。監査証跡詳細の文字列のサイズがここで設定されたしきい値よりも大きい場合、監査証跡を取得しても、すぐにはロードされません。詳細の文字列のサイズとともにリンクが表示されます。このしきい値設定よりもサイズの大きな文字列は、audit_trail 表ではなく、audit_details 表に格納されます。

このプロパティは、永続プロセスに適用されます。

詳細の文字列には、通常、BPEL 変数の内容が含まれます。変数が非常に大きい場合、これを監査証跡のログに記録すると、パフォーマンスに深刻な影響を及ぼす可能性があります。

値

デフォルト値は 50KB です。

関連項目： audit_trail 表および audit_details 表の詳細は、7-10 ページの表 7-2 を参照してください。

7.4.3 auditLevel BPEL プロパティ

このプロパティは、監査証跡のロギング・レベルを設定します。このプロセスは、永続プロセスおよび一時プロセスの両方に適用されます。

このプロパティは、プロセスによりログに記録される監査イベント数を制御します。監査イベントの数が多いほど、audit_trail 表に記録される件数も増えるため、この設定はパフォーマンスに大きな影響を与えます。この監査情報は、Oracle BPEL Control でプロセスの状態を表示するためにのみ使用されます。

一部の監査情報のみを保存する場合は、このプロパティを使用します。業務の要件に応じたレベルを選択してください。監査情報は、データベースの増大およびスループットに大きな影響を与えます。パフォーマンスを最適化するには、このプロパティにできるだけ低いレベルを設定してください。

関連項目： 監査レベルと audit_trail 表の詳細は、7-10 ページの表 7-2 を参照してください。

値

このプロパティの値を次に示します。

- off: 監査イベント（アクティビティの実行情報）は一切永続化されず、ログも記録されません。これにより、インスタンス処理に対するパフォーマンスがわずかに向上します。
- minimal: すべてのイベントがログに記録されますが、監査詳細（変数の内容）はログに記録されません。比較的ペイロードの大きいプロセスには、この設定をお薦めします。
- production: すべてのイベントがログに記録されます。assign アクティビティの監査詳細を除き、他のすべてのアクティビティの詳細はログに記録されます。比較的ペイロードの小さいプロセスには、この設定をお薦めします。
- development（デフォルト）: すべてのイベントがログに記録されます。すべてのアクティビティの監査詳細もログに記録されます。

7.4.4 bpelcClasspath BPEL プロパティ

このプロパティは、BPEL プロセス・コンパイラのクラスパスを設定します。

このプロパティは、サーバー側の BPEL プロセス・コンパイラのクラスパスを設定します。BPEL の Java 実行アクティビティにより使用される（BPEL アーカイブにパッケージされていない）ユーザー固有のクラスおよびライブラリは、必ずこのクラスパスに指定する必要があります。これにより、サーバー側の BPEL プロセス・コンパイラで、BPEL プロセスが正常にコンパイルされます。

このプロセスは、永続プロセスおよび一時プロセスの両方に適用されます。

値

デフォルト値は次のとおりです。

Oracle_Home¥bpel¥system¥classes;
Oracle_Home¥bpel¥lib¥j2ee_1.3.01.jar

7.4.5 datasourceJndi BPEL プロパティ

このプロパティはドメイン・データソースの JNDI 名を設定します。

このデータソースは、任意のデータソースを参照できます（JTA は必要ありません）。

このプロセスは、永続プロセスおよび一時プロセスの両方に適用されます。

値

デフォルト値は、jdbc/BPELServerDataSourceWorkflow です。

7.4.6 deliveryPersistPolicy BPEL プロパティ

警告： このプロパティの設定は、デフォルト値の on のままにしておくことをお薦めします。このプロパティを off に設定した場合、システムに障害が発生するとメッセージが失われます。このプロパティ設定をデフォルト以外の値に変更する場合は、十分注意してください。

このプロパティにより、Oracle BPEL Server が受信するメッセージのデータベース永続性を有効または無効にします。デフォルトでは、受信リクエストは次に示す配信サービスのデータベース表に保存されます。

- dlv_message
- invoke_message

これらのリクエストは、後に Oracle BPEL Server のワーカー・スレッドにより取得され、目的の BPEL プロセスに配信されます。信頼性よりもパフォーマンスが重要な場合は、受信メッセージをデータベースに保存する必要はありません。このプロパティにより、配信メッセージが永続化されます。このプロパティは、永続プロセスに適用されます。

一方向呼出しメッセージは、配信されるまで配信キャッシュに格納されます。一方向メッセージが到着する割合が、Oracle BPEL Server によりメッセージが配信される割合を大幅に上回る場合、またはサーバーに障害が発生した場合に、一部のメッセージが失われる場合があります。Oracle BPEL Control（「**BPEL ドメインの管理**」→「**スレッド**」）では、「**保留中のリクエスト**」セクションの「**新規インスタンス・リクエスト**」と「**コールバック・リクエスト**」の統計を確認することで、配信キャッシュのサイズを監視できます。「**スケジュール済**」列には、キャッシュされたメッセージの数が表示されます。

関連項目： 配信サービス・データベース表の詳細は、7-10 ページの表 7-2 を参照してください。

値

このプロパティの値を次に示します。

- on（デフォルト）: 配信メッセージはデータベースに保存されます。
- off: 受信した配信メッセージは、メモリー内キャッシュにのみ格納されます。配信されるメッセージの数が増えると、システムはオーバーロード状態（メッセージはスケジュール済キューにバックログされる）になり、メモリー不足エラーが表示されます。受信メッセージの数に対応できるように、WorkerBean スレッドの数をチューニングしてください。
- off.immediate: 呼出しキューのメッセージのスケジューリングを実行しないよう Oracle BPEL Server に指示し、BPEL インスタンスを同期的に呼び出します。

関連項目： 「WorkerBean」（7-23 ページ）

7.4.7 dspAgentDelay BPEL プロパティ

このプロパティは、ディスパッチャ・エージェントのトリガー間隔を秒単位で設定します。このエージェントは、JMS レイヤーの障害のために処理されていないディスパッチャ・レイヤー内のメッセージをすべてクリーンアップします。

このプロセスは、永続プロセスに適用されます。

値

デフォルト値は 120 秒です。

7.4.8 dspInvokeAllocFactor BPEL プロパティ

このプロパティは、受信した呼出しメッセージを処理するためにタスク化されるアクティブ・スレッドの割合を設定します。現在のスレッド割当て状態によっては、メッセージの処理を終了したスレッドをもう一度タスク化して、Oracle BPEL Server または呼出しメッセージの処理に使用することができます。

このプロセスは、永続プロセスに適用されます。

値

デフォルト値は `0.4`（40%）です。

7.4.9 dspMaxRequestDepth BPEL プロパティ

このプロパティは、同一のリクエスト内で処理されるメモリー内アクティビティの最大数を設定します。アクティビティ・リクエストの処理後、Oracle BPEL Process Manager は、リクエストのトランザクション性を保護したまま、できるだけ多くの後続アクティビティの処理を試行します。アクティビティの処理チェーンがこの限界に達すると、そのインスタンスはデハイドレーションされ、次のアクティビティは別のトランザクションで実行されます。

リクエストの上限値が大きすぎる場合、リクエスト時間の合計はアプリケーション・サーバーのトランザクションのタイムアウト制限を上回る可能性があります。

このプロセスは、永続プロセスに適用されます。

値

デフォルト値は `600` アクティビティです。

7.4.10 dspMaxThreads BPEL プロパティ

このプロパティは、負荷がピークに達している期間中にメッセージを処理するアクティブ・ディスパッチャ・スレッドの最大数を設定します。このプロパティは永続プロセスに適用され、アプリケーション・サーバーの構成に依存しています。

ドメインのパフォーマンスおよびスケーラビリティを向上させるには、これが最も簡単な方法です。Oracle BPEL Server では、MDB スレッドを使用して Oracle BPEL Server メッセージが処理されます。このプロパティの最大値は、Oracle BPEL Server の MDB J2EE リスナー・スレッドの設定によって変化します。Oracle Application Server では、この数は `orion-ejb-jar.xml` デプロイメント・ディスクリプタ・ファイルで設定します。

たとえば、MDB J2EE リスナー・スレッドの総数が `120` である場合、**dspMaxThreads** の値は `120` 以下に設定できます。複数のドメインを構成している場合、すべてのドメインに対する **dspMaxThreads** 設定の合計が、MDB J2EE リスナー・スレッドの設定を超えないようにしてください。

アプリケーション・サーバーおよびデータベース・ホストの CPU 使用率が処理能力を大きく下回っている場合、必要に応じて、この値と MDB J2EE リスナー・スレッドの設定を大きくしてください。それでも CPU が十分に使用されない場合は、複数の Oracle BPEL Server インスタンスの実行を検討します。

注意： MDB J2EE リスナー・スレッドの構成は、次のファイルで指定します。

- Oracle BPEL Process Manager for Developers インストール・タイプの場合、このファイルは `WorkerBean` の下の *`SOA_Oracle_Home`*`¥j2ee¥home¥application-deployments¥orabpel¥ejb_ob_engine¥orion-ejb-jar.xml` にあります。
- Oracle BPEL Process Manager for OracleAS Middle Tier インストール・タイプの場合、このファイルは `WorkerBean` の下の *`SOA_Oracle_Home`*`¥j2ee¥home¥application-deployments¥orabpel¥ejb_ob_engine¥orion-ejb-jar.xml` にあります。

関連項目：

- 詳細は、7-4 ページの「BPEL スレッド・モデル」を参照してください。
- MDB J2EE リスナー・スレッドの詳細は、7-23 ページの「Oracle BPEL Server の EJB 構成」を参照してください。

値

デフォルト値は 100 スレッドです。

7.4.11 dspMinThreads BPEL プロパティ

このプロパティは、負荷がピークに達している期間中にメッセージを処理するアクティブ・ディスパッチャ・スレッドの最小数を設定します。

現在のアクティブ・スレッドの数がこの値を下回る場合、新しいスレッドを割り当てるかどうかを決定するときに負荷率は考慮されません。

このプロセスは、永続プロセスに適用されます。

値

デフォルト値は 5 スレッドです。

7.4.12 expirationMaxRetry BPEL プロパティ

このプロパティは、wait アクティビティ、または pick アクティビティの onAlarm ブランチにおける期限切れコールの最大数を設定します。期限切れコールの再試行がこの回数を超えると、失敗となります。

期限切れコールのターゲットであるアクティビティやインスタンスが見つからない場合、コールは再スケジュールされます。

期限切れコールによる最初（オリジナル）の試行は、再試行の回数には含まれません。

このプロセスは、永続プロセスに適用されます。

値

デフォルト値は 5 です。

7.4.13 idempotentThreshold BPEL プロパティ

このプロパティは、多重呼出し不変なサービスによりアクティビティが正常終了されるまでの、最大時間（秒単位）を設定します。この時間が経過しても多重呼出し不変なサービスが完了しない場合、このサービスは多重呼出し不変でないとみなされ、現在のトランザクションがデータベースにコミットされます。これにより、多重呼出し不変チェーン内で別のサービスが失敗した場合に、時間のかかるサービスが再実行されるのを防ぐことができます。

値

デフォルト値は 30 秒です。

関連項目：「多重呼出し不変なアクティビティ」（7-3 ページ）

7.4.14 instanceKeyBlockSize BPEL プロパティ

このプロパティは、インスタンス ID の範囲のサイズを制御します。Oracle BPEL Server では、この値を使用して、インスタンス・キー（プロセス・インスタンス ID の範囲）がバッチで作成されます。このメモリー内 ID の範囲を作成すると、その次の範囲は更新され、`ci_id_range` 表に保存されます。たとえば、**instanceKeyBlockSize** を `100` に設定すると、Oracle BPEL Server はメモリー内にインスタンス・キーの範囲を作成します（100 個のキー。これは後に、`cikey` として `cube_instance` 表に挿入される）。ブロック・サイズが `ci_id_range` 表に対する更新の数よりも小さい場合、パフォーマンスの問題が発生する可能性があります。

> **関連項目：** `cube_instance` 表の詳細は、7-10 ページの表 7-2 を参照してください。

値

デフォルト値は `10000` です。

7.4.15 instCacheHighWatermark BPEL プロパティ

> **注意：** このパラメータは変更しないことをお薦めします。このパラメータは、JVM の問題を十分に理解している場合にのみ変更してください。

このプロパティは、プルーニング前にキャッシュ可能な進行中インスタンスの最大数を設定します。最大値に到達すると、最小値（**instCacheLowWatermark** プロパティで設定）になるまでキャッシュから古いインスタンスが削除（プルーニング）されます。プルーニングされたインスタンスは、必要に応じて、デハイドレーション・ストアから取得できます。このプロパティは、永続プロセスに適用されます。

この値は、**instCachePolicy** プロパティが `lru` または `hybrid` に設定されている場合のみ使用されます。このプロパティを設定する場合は、次の点を考慮してください。

- いずれかの時点で、Oracle BPEL Process Manager による処理が見込まれている進行中のインスタンスの数。
- 各プロセス・インスタンスにより使用されるメモリーの量。メモリー使用量を判断するには Java プロファイラを使用します。

システムを通じてインスタンスを 1 つ実行し、この実行に応じて増加したメモリー使用率を測定できます。

このプロパティの設定値が大きすぎる場合、`OutOfMemoryException` エラー・メッセージが表示されます。また、この値が大きすぎると、ガベージ・コレクタがより頻繁に実行されるようになるため、実際にシステムは遅くなります。ガベージ・コレクタを監視するには、Sun 社のビジュアル・ガベージ・コレクション（GC）ツール（`http://java.sun.com/performance/jvmstat`）を使用します。

> **関連項目：**
>
> - 「instCacheLowWatermark BPEL プロパティ」（7-17 ページ）
> - 「instCachePolicy BPEL プロパティ」（7-17 ページ）
> - 「optCacheOn BPEL プロパティ」（7-19 ページ）

値

デフォルト値は `3000` です。0 は制限がないことを表します。

7.4.16 instCacheLowWatermark BPEL プロパティ

> **注意：** このパラメータは変更しないことをお薦めします。このパラメータは、JVM の問題を十分に理解している場合にのみ変更してください。

このプロパティは、プルーニングの際にキャッシュに残す進行中インスタンスの数を設定します。このプロパティは、永続プロセスに適用されます。

キャッシュ内でインスタンスの数が最大値に達すると、このレベルになるまでキャッシュ内のインスタンスが削除されます。

キャッシュのプルーニングは、キャッシュ・サイズが最大値（**instCacheHighWatermark** プロパティで設定）に達したときに行われます。この **instCacheLowWatermark** プロパティは、プルーニングの実行率を制御します。デフォルトは最大値の 75% です。これは、プルーニング時に、キャッシュが最大値の 75% に削減されることを表します。この値は、**instCachePolicy** プロパティが `lru` または `hybrid` に設定されている場合のみ使用されます。

インスタンス・キャッシュの統計を監視するには、Oracle BPEL Control の「**BPEL ドメインの管理**」→「**スレッド**」に移動します。このページの下部にある「**サーバー・キャッシュ統計**」セクションには、「**インスタンス・キャッシュ**」というエントリがあります。ここにはキャッシュ・サイズとヒット率が表示されます。ヒット率がきわめて低い場合は、キャッシュ・サイズ、または最小値を大きくすることを検討してください。

このプロパティの設定値が大きすぎる場合、`OutOfMemoryException` エラーが表示されます。また、この値が大きすぎると、実際にシステムは遅くなります。これは、ガベージ・コレクタをより頻繁に実行しなければならないためです。ガベージ・コレクタを監視するには、Sun 社のビジュアル GC ツール（`http://java.sun.com/performance/jvmstat`）を使用します。

関連項目：

- 「instCacheHighWatermark BPEL プロパティ」（7-16 ページ）
- 「instCachePolicy BPEL プロパティ」（7-17 ページ）
- 「optCacheOn BPEL プロパティ」（7-19 ページ）

値

デフォルト値は `2250`（75%）です。

7.4.17 instCachePolicy BPEL プロパティ

このプロパティは、キャッシュから進行中のインスタンスを削除するときに使用される追出しポリシーを設定します。このプロパティは、永続プロセスに適用されます。

このプロパティは、**optCacheOn** プロパティが `true` に設定されている場合のみ有効です。

キャッシュ管理を微調整する必要がある場合は、このプロパティを使用します。メモリーに保存する必要のあるプロセス・インスタンスの数がわかっている場合には、`lru` 設定の使用をお薦めします。`lru` 設定を使用する場合、**instCacheHighWatermark** プロパティおよび **instCacheLowWatermark** プロパティも設定する必要があります。

> **注意：** キャッシュ値が **auto** に設定されている場合、JVM 実装で `OutOfMemoryException` エラー・メッセージが表示されることがあります。これは、自動キャッシュ設定が JVM ソフト参照に依存しているためです。このエラーが発生した場合は、キャッシュの値を `lru` に設定してください。

関連項目：

- 「instCacheHighWatermark BPEL プロパティ」（7-16 ページ）
- 「instCacheLowWatermark BPEL プロパティ」（7-17 ページ）
- 「optCacheOn BPEL プロパティ」（7-19 ページ）

値

このプロパティの値を次に示します。

- `lru`: 最低使用頻度。この設定では、最も長期間アクセスされていないインスタンスから順に削除されます。これは推奨される設定です。
- `auto`（デフォルト）: 削除の判断を JVM に委任します。インスタンスは、ガベージ・コレクタがソフト参照を取得したときに削除されます。
- `soft-lru`: `lru` と `auto` との組合せです。

7.4.18 invokerQueueConnectionPoolMinSize BPEL プロパティ

このプロパティは、インボーカ・キューの接続プールの最小サイズを設定します。この値は、インボーカ・スレッド数と同じである必要があります。インボーカ・スレッドが `200` に設定されている場合、このプロパティの値を `200` に設定することで JMS のウォーミング・アップを回避できます。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

値

デフォルト値は `25` です。

関連項目： インボーカ・スレッドの詳細は、7-23 ページの「InvokerBean」を参照してください。

7.4.19 largeDocumentThreshold BPEL プロパティ

このプロパティは、大きな XML 文書の永続性しきい値を設定します。これは BPEL 変数の最大サイズ（KB 単位）です。この値を超えた場合、残りのインスタンスのスコープ・データとは別の場所に保存されます。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

インスタンスの処理が実行されるたびに XML 文書を読み書きしている場合、大きな XML 文書は Oracle BPEL Server 全体のパフォーマンスに影響を与えます。

値

デフォルト値は `50KB` です。

7.4.20 minBPELWait BPEL プロパティ

このプロパティは、BPEL アクティビティの最小待機時間を設定します。

wait アクティビティの待機時間、または pick アクティビティの onAlarm ブランチの待機時間がここで定義された値よりも短い場合、待機は無視されます。

このプロパティは、永続プロセスに適用されます。

値

デフォルト値は `2` 秒です。

7.4.21 optCacheOn BPEL プロパティ

このプロパティは、進行中インスタンスのメモリー内キャッシュを設定します。このプロパティは、永続プロセスに適用されます。

true に設定した場合、Oracle BPEL Process Manager ではデータベースは検索されず、メモリー内のキャッシュからアクティブ・インスタンスのロードが試行されます。最適化を無効にするには、true 以外の値を指定します。

プロセスの実行時間が長く、サブプロセスはすぐにコール・バックしない場合、このプロパティを false に設定します。多くのコール・バックが予想される、比較的短めのプロセスを処理している場合には、こちらを検討してください。

このプロパティを true に設定した場合、次のようなキャッシュ関連の設定が必要です。

- **instCacheHighWatermark**
- **instCacheLowWatermark**
- **instCachePolicy**

キャッシュを使用しなくても、パフォーマンスの目標を達成できる場合は、管理やチューニングを簡略化するために、この設定を false のままにしておくことをお薦めします。

注意： キャッシュの有効化により、パフォーマンスが影響を受けることがあります。これは、キャッシュの設定値が大きすぎる場合に発生し、JVM ガベージ・コレクタが頻繁に実行されるようになります。ガベージ・コレクタを監視するには、Sun 社のビジュアル GC ツール（http://java.sun.com/performance/jvmstat）を使用します。

値

このプロパティの値を次に示します。

- true: Oracle BPEL Server ではデータベースは検索されず、メモリー内のキャッシュからアクティブ・インスタンスのロードが試行されます。
- false（デフォルト）: Oracle BPEL Server では毎回、データベースからインスタンスがロードされます。

関連項目：

- 「instCacheHighWatermark BPEL プロパティ」（7-16 ページ）
- 「instCacheLowWatermark BPEL プロパティ」（7-17 ページ）
- 「instCachePolicy BPEL プロパティ」（7-17 ページ）

7.4.22 optIdempotentRouting BPEL プロパティ

注意： このパラメータは変更しないことをお薦めします。

このプロパティは、多重呼出し不変なサービスのルーティング・ショートカットを設定します。

true に設定すると、アクティビティ・サービスが多重呼出し不変である場合に、Oracle BPEL Server では同一のトランザクション内でできるだけ多くのアクティビティの処理が試行されます。

このプロパティは、永続プロセスに適用されます。

デフォルト値は true です。最適化を無効にするには、true 以外の値を指定します。

値

デフォルト値は true です。

7.4.23 optSoapShortcut BPEL プロパティ

注意： このパラメータは変更しないことをお薦めします。

このプロパティは、ローカル SOAP リクエストのショートカットを設定します。

ローカル SOAP コールは、通常、SOAP スタック経由でメッセージを送信するのではなく、内部コールを使用して実行されます。

デフォルトでは、Oracle BPEL Process Manager は、SOAP スタックを無視してすべてを最適化します。最適化を無効にするには、`true` 以外の値を指定します。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

値

- `true`（デフォルト）: ローカル SOAP コールは、SOAP スタックを無視します。
- `false`: ローカル SOAP コールは、SOAP スタックを経由します。

7.4.24 processCheckSecs BPEL プロパティ

このプロパティは、Oracle BPEL Server で最後に BPEL アーカイブがチェックされてから、次にチェックされるまでの待ち時間を秒数で指定します。チェックするというのは、BPEL アーカイブで、特定プロセスの最終変更時間のタイムスタンプを確認するという意味です。最後のチェックの後、指定された秒数が経過し、BPEL アーカイブ・ファイルが変更されている場合、プロセスは新しいアーカイブからリフレッシュされます。失効チェックが最後に実行されてから十分な時間が経過していない場合は、現在ロードされているプロセス・クラスが使用されます。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

プロセスのチェックを無効化するには、値に `-1` を指定します。この場合、Oracle BPEL Server では、プロセスがロードされても、同じプロセスの新しいバージョンがデプロイされているかどうかのチェックは行われません。

値

デフォルト値は `1` 秒です。

7.4.25 relaxBpelAssignRules BPEL プロパティ

注意： このプロパティは Oracle BPEL Control に表示されますが、使用しないことをお薦めします。このプロパティは廃止されています。

このプロパティを使用すると、『Business Process Execution Language for Web Services Specification Version 1.1』における割当てルールの施行を緩和できます。`true` に設定されている場合、Oracle BPEL Process Manager では、BPEL 変数の割当て時にルールが適用されません。たとえば、BPEL 仕様で許可されていない NULL 割当てに関するエラーが表示されません。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

値

このプロパティの値を次に示します。

- `false`（デフォルト）: 割当てルールを緩和しません。
- `true`: 割当てルールを緩和します。

7.4.26 slowPerfThreshold BPEL プロパティ

このプロパティは、アクティビティを正常終了するためにサービスが使用できる最大時間（秒単位）を設定します。この時間が経過してもサービスが完了しない場合、このサービスは遅いとみなされます。Oracle BPEL Process Manager では、遅いサービスに関する統計が収集されます。

このプロパティは、永続プロセスに適用されます。

値

デフォルト値は 1 秒です。

7.4.27 statsLastN BPEL プロパティ

このプロパティは、最後に処理されたリクエストのリストのサイズを設定します。リクエストが終了するごとに、このリクエストに関する統計がリストに保存されます。値が 0 以下の場合、統計の収集は行われません。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

統計は、Oracle BPEL Control の「**BPEL ドメインの管理**」→「**統計**」に表示されます。

値

デフォルト値は 1000 です。

7.4.28 syncMaxWaitTime BPEL プロパティ

このプロパティは、プロセス結果の受信側が結果が戻るまで待機する、最大時間を設定します。非同期 BPEL プロセスの結果は、Oracle BPEL Server の結果を待機する受信側により、同期的に取得されます。

このプロパティは、一時プロセスに適用されます。

値

デフォルト値は 45 秒です。

7.4.29 txDatasourceJndi BPEL プロパティ

このプロパティはドメインのトランザクション・データソースの JNDI 名を設定します。JTA のサポートには、このデータソースの構成が必要です。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

値

デフォルト値は jdbc/BPELServerDataSource です。

7.4.30 uddiLocation BPEL プロパティ

このプロパティは、Universal Description, Discovery, and Integration（UDDI）バージョン 3 に準拠するレジストリの問合せ URL を指定します。

仮想の場所を使用している場合、つまりパートナ・リンクで抽象 WSDL のみを参照し、パートナ・リンクのバインディング・レベルでデプロイメント・ディスクリプタに registryServiceKey という名前のプロパティを設定している場合は、このプロパティを使用して UDDI レジストリに接続し、情報を取得します。

値

デフォルト値はありません。

7.4.31 validateXML BPEL プロパティ

このプロパティは、受信 XML 文書、および送信 XML 文書を検証します。

`true` に設定されている場合、Oracle BPEL Process Manager では、受信および送信 XML 文書に対してスキーマ検証が行われます。

このプロパティは、永続プロセスおよび一時プロセスの両方に適用されます。

値

デフォルト値は `false` です。

7.4.32 workerQueueConnectionPoolMinSize BPEL プロパティ

このプロパティは、ワーカー・キューの接続プールの最小サイズを設定します。この値は、ワーカー・スレッド数と同じである必要があります。ワーカー・スレッド数が `200` に設定されている場合、この値を `200` に設定することで JMS のウォーミング・アップを回避できます。

このプロパティは、永続プロセスに適用されます。

値

デフォルト値は `25` です。

関連項目： 「WorkerBean」（7-23 ページ）

7.5 Oracle BPEL に対する OC4J チューニング

この項で説明するパラメータは、Oracle Application Server レベルで設定されます。これらのパラメータを有効にするには、OC4J インスタンスを再起動する必要があります。

この項には、次の項目が含まれています。

- Oracle BPEL Process Manager の JTA トランザクション・タイムアウトのチューニング
- Oracle BPEL Server の EJB 構成
- Oracle BPEL のデータソース構成

関連項目： Oracle Application Server の起動と停止の手順については、『Oracle Application Server 管理者ガイド』を参照してください。

7.5.1 Oracle BPEL Process Manager の JTA トランザクション・タイムアウトのチューニング

Oracle BPEL Server では、原子性を達成するために JTA が使用されます。トランザクション・タイムアウトの値は、`transaction-manager.xml` ファイルにデフォルトで `60000` ミリ秒と設定されています。このファイルの場所は、Oracle BPEL Process Manager をインストールした方法により異なります。

- Oracle Application Server SOA インストールの場合、このファイルは *`SOA_Oracle_Home`*`¥j2ee¥home¥config` にあります。
- Oracle BPEL Process Manager インストールの場合、このファイルは *`SOA_Oracle_Home`*`¥bpel¥system¥appserver¥oc4j¥j2ee¥home¥config` にあります。

特に Oracle BPEL Server に負荷がかかっている場合は、タイムアウトにより、トランザクション・ロールバック・エラーが発生することがあります。タイムアウトの原因には、次のようなものが考えられます。

- リソースが不足している（たとえば、接続プールに十分なデータベース接続がない、サーバー・スレッドが 60 秒間待機してタイムアウト・エラーを表示したなど）。
- 大きな文書の操作が行われている（たとえば、文書のサイズが非常に大きく、データベースへの書込みに 60 秒を超える時間がかかっている）。

プロセスに応じて、この値を変更します。次の例では、タイムアウトを 120 秒に設定しています。

```
<transaction-config timeout="120000" />
```

プロセスによるパートナの呼出しに、指定したタイムアウトしきい値よりも長い時間がかかる場合、一方向リクエストを使用して呼び出すか、bpel.xml デプロイメント・ディスクリプタ・ファイルで nonBlockingInvoke パートナ・リンク・プロパティを true に設定します。

関連項目： 「nonBlockingInvoke BPEL プロパティ」（7-9 ページ）

7.5.2 Oracle BPEL Server の EJB 構成

パフォーマンスを改善するには、すべての Oracle BPEL Server の EJB に対する max-instances 属性を orion-ejb-jar.xml ファイルから削除することをお薦めします。Oracle BPEL Process Manager for OracleAS Middle Tier インストール・タイプの場合、このファイルは *SOA_Oracle_Home*¥j2ee¥home¥application-deployments¥orabpel¥ejb_ob_engine にあります。

これにより、アプリケーション・サーバーでは、さらに多くのリソースを利用率の高い Bean に割り当てられるようになります。

7.5.2.1 WorkerBean

Oracle BPEL Server では、WorkerBean と呼ばれる MDB を使用して処理が行われます。したがって、この MDB に十分なスレッドを割り当てることが重要です。不足している場合、リソースの使用率は最適化されません。orion-ejb-jar.xml ファイルの次のコードは、70 スレッドを割り当てています。

```
<message-driven-deployment name="WorkerBean"
 destination-location="jms/collaxa/BPELWorkerQueue"
 connection-factory-location="jms/collaxa/BPELWorkerQueueFactory"
 listener-threads="70" min-instances="100">
     <ejb-ref-mapping name="ejb/local/DispatcherLocalBean" />
      ..
      ..
</message-driven-deployment>
```

7.5.2.2 InvokerBean

インボーカ Bean は、ブロックしていない invoke アクティビティでのみ使用されます。一部の呼出しをブロックせずに行う場合、InvokerBean に割り当てられるスレッドの数を増やします。orion-ejb-jar.xml にある次のコードは、30 スレッドを割り当てています。

```
<message-driven-deployment name="InvokerBean"
 destination-location="jms/collaxa/BPELInvokerQueue"
 connection-factory-location="jms/collaxa/BPELInvokerQueueFactory"
 listener-threads="30" min-instances="100">
<ejb-ref-mapping name="ejb/local/ProcessManagerLocalBean" />
   …
   …
</message-driven-deployment>
```

注意： InvokerBean スレッド数と WorkerBean スレッド数の合計は、Oracle BPEL Control の「**BPEL ドメインの管理**」→「**構成**」の **dspMaxThreads** ドメイン・プロパティに指定された値以上である必要があります。複数のドメインを構成した場合、すべてのドメインに対する **dspMaxThreads** プロパティを加算し、この合計値を MDB の総スレッド数と比較します。

関連項目：

- 「nonBlockingInvoke BPEL プロパティ」（7-9 ページ）
- 「dspMaxThreads BPEL プロパティ」（7-14 ページ）

7.5.3 Oracle BPEL のデータソース構成

Oracle BPEL Server では、アプリケーション・サーバーの JTA データソースを使用して、データベース接続が取得されます。デフォルトで、Oracle BPEL Server は Oracle Database Lite のデハイドレーション・ストアを使用するよう構成されています。ストレス・テストおよび本番環境については、Oracle Database 10*g* の使用をお薦めします。Oracle Database Lite は、開発初心者の負担を緩和するためにデフォルトのインストールにパッケージされています。

Oracle BPEL Server のデータソース・エントリを構成する場合は、次の点に注意してください。Oracle BPEL Process Manager for OracleAS Middle Tier インストール・タイプの場合、このデータソース・エントリは *SOA_Oracle_Home*¥j2ee¥home¥config¥data-sources.xml ファイルにあります。

- データソースを構成する場合、Oracle BPEL Server にサービスを提供するための十分な空き接続が接続プールに存在することを確認してください。
- 接続プールのサイズは、Oracle BPEL Control にある **dspMaxThreads** プロパティ値の合計以上であることが必要です。複数のドメインを構成している場合は、**dspMaxThreads** プロパティの値をすべて合計し、この値をデータソースの `max-connections` の値と比較します。デフォルトの `max-connections` 値は無制限です。
- Oracle Database 10*g* では、データソースはシン・ドライバを使用する必要もあります。Oracle9*i* Database では、Oracle Call Interface（OCI）のパフォーマンスがわずかに優れています。

データベース永続性が有効な場合、`num-cached-statements` 属性を使用して JDBC 文キャッシュを有効にすると、通常は Oracle BPEL Server のパフォーマンスが向上します。文キャッシュにより、カーソルを繰返し作成したり、文を繰り返し解析および作成することに起因するオーバーヘッドが軽減します。また文キャッシュにより、アプリケーション・サーバーとデータベース・サーバー間の通信オーバーヘッドも低下します。

関連項目：

- 「dspMaxThreads BPEL プロパティ」（7-14 ページ）
- Oracle BPEL Process Manager デハイドレーション・ストアとして Oracle Database を構成する手順については、『Oracle BPEL Process Manager インストレーション・ガイド』を参照してください。
- JDBC 文キャッシュの詳細は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。
- 「データソースに対する文キャッシュの有効化」（3-8 ページ）

7.6 Oracle BPEL Server に対する Java 仮想マシンのパフォーマンス・チューニング

JVM パラメータは、Oracle BPEL Server のパフォーマンスに影響を与える場合があります。パフォーマンスに影響を与える主な要因は、ヒープ・サイズに関係します。ヒープ・サイズは JVM で使用するメモリー量を制御します。BPEL プロセス・インスタンスが専用のホストで稼動している場合、ヒープ・サイズはなるべく大きな値に設定します。

もう 1 つの重要なヒープ構成は、ガベージ・コレクタの世代の設定です。ガベージ・コレクタは、オブジェクトを存在期間の長さによって分類することにより、コレクションを最適化します。Oracle BPEL Server オブジェクトのほとんどは短期的なものであり、Eden スペースで存続します。Eden スペースのサイズは総ヒープ・サイズの 60 ～ 70% にすることをお薦めします。

JVM の -Xmn 起動オプションは、Eden スペースのサイズを、値を明示して設定します。このオプションを設定するには、次の手順を実行します。

1. 指定した最大ヒープ・サイズを基に、60 ～ 70% の値を計算します。
2. 計算した値を使用して、JVM -Xmn コマンドライン・パラメータを設定します。

JVM コマンドライン・オプションを変更する手順は次のとおりです。

1. OC4J インスタンスのホーム・ページにナビゲートします。
2. 「管理」をクリックします。
3. 必要な場合は、「開く」アイコンをクリックして表の「プロパティ」セクションを開きます。次に、「サーバー・プロパティ」行の「タスクに移動」アイコンをクリックします。
4. 「コマンドライン・オプション」領域の「オプション」表で、該当するコマンドライン・オプションを変更します。
5. 「適用」をクリックします。
6. 「クラスタ・トポロジ」ページにナビゲートし、変更した OC4J インスタンスを選択して「再起動」をクリックします。「確認」ページで「はい」をクリックします。

関連項目：

- 「OC4J 用の十分な Java ヒープの確保」（3-4 ページ）
- 「JVM のガベージ・コレクション・オプションのチューニング」（3-4 ページ）

7.7 デハイドレーション・ストア・データベースのパフォーマンス・チューニング

Oracle BPEL Server のパフォーマンスは、デハイドレーション・ストアの容量と関係しています。次の処理が推奨されます。

- REDO ログを独立した RAID 1+0 ディスクに移動する。
- 各 REDO ログ・ファイルのサイズを大きな値（たとえば、1GB）に増やす。
- Oracle BPEL Server のために独立したデータベース表領域を作成する。

表 7-3 のデータベース・パラメータは、Oracle BPEL Process Manager のパフォーマンスに影響を与えます。使用する値は、ハードウェア構成により異なります。

表 7-3 Oracle BPEL Process Manager のパフォーマンスに影響を与えるデータベース・パラメータ

パラメータ名	値の例
LOG_BUFFER	1048576
SHARED_POOL_SIZE	400M
JOB_QUEUE_PROCESSES	1
DB_CACHE_SIZE	1000M
DB_FILE_MULTIBLOCK_READ_COUNT	8
UNDO_RETENTION	0
PROCESSES	300
SESSION_CACHED_CURSORS	100

関連項目： 使用している Oracle Database リリースの『Oracle Database パフォーマンス・チューニング・ガイド』と『Oracle Database リファレンス』

- Oracle Database 10*g* リリース 2（10.2）の場合

  `http://www.oracle.com/technology/documentation/database10gr2.html`

- Oracle Database 10*g* リリース 1（10.1）の場合

  `http://www.oracle.com/technology/documentation/database10g.html`

- Oracle Database 9*i* リリース 2（9.2）の場合

  `http://www.oracle.com/technology/documentation/oracle9i.html`

7.8 まとめ

この章では、Oracle BPEL Process Manager のプロパティを設定し、プロセス、ドメイン、アプリケーション・サーバー、Java 仮想マシン（JVM）、デハイドレーション・ストア・データベースなどのレベルでパフォーマンスを最適化する方法について説明しました。この章では、これらのプロパティ設定および推奨される使用方法について説明しています。

8

Oracle Business Activity Monitoring のパフォーマンス

Oracle Business Activity Monitoring のパフォーマンスを最大化するには、Oracle Business Activity Monitoring システム専用のハードウェアを用意し、それにデータベースを配置します。また、この章で説明するデータベース管理を実践することで、Oracle Business Activity Monitoring の着信トランザクションのスループットを最大化できます。これらのデータベース管理は、Oracle Business Activity Monitoring システムに特化された検証テストと監視情報に基づいています。Database パフォーマンス・チューニング・ガイドに説明されている一般的なデータベース管理方法も、Oracle Business Activity Monitoring 専用のデータベースに適用されます。

この章には、次の項が含まれています。

8.1 REDO ログ・ファイルの管理

Oracle Business Activity Monitoring による入力データの受信率が高くなると、Oracle Business Activity Monitoring Active Data Cache（ADC）が受信したデータをデータベースに送信し、使用可能な入力帯域幅がほとんどあるいはまったくなくなる状態が ADC サーバーで生じます。そのため、データベースが ADC サーバーからの受信データをどれくらいの率で取得できるかが、入力データのスループットを制限する要因となります。データの取得率が高くなると、ログ・ファイル同期とも呼ばれる、データベースが REDO ログ・レコードをディスクに書き込む機能によって、データベースのスループットが制限されます。アプリケーション、この場合は ADC サーバーが、REDO ログ・データがディスクに書き込まれる速度よりも速くデータをコミットすると、以降のトランザクション・コミット・リクエストまたはロールバック・リクエストは待機が必要になります。

REDO ログ・ファイルのチューニングの最終的な目標は、挿入スループットの制限要因となる、ログ・ファイルの同期待ちを軽減または除去することです。Oracle データベースでログ・ファイルの同期待ち数を減らすには、次の 2 つの戦略が有効です。

- REDO ログの同期アクティビティに使用可能な I/O 帯域幅を増やします。そのためには、REDO ログ・ファイルを専用の物理ディスクに移動し、システム内のその他の I/O アクティビティと競合しないようにします。
- データベースへの挿入ワークロードを変更し、トランザクションの 1 回のコミットでデータベースに挿入される行数を増やします。この戦略では、コミットの発行回数を減らすようにアプリケーションを変更する必要があります。より多くの操作を 1 つのトランザクションにバッチ化することで、コミット処理の必要性が軽減されます。ただし、このアプローチには、システムに障害が発生したときに、前回のコミット以降のすべての入力が失われるというマイナス面があります。コミットの回数が多いと、システム障害時に失われるデータ量は少なくなります。一方、コミットの回数が少ないと、それまでに取得したより多くのデータが失われます。パフォーマンスと信頼性の間のこのトレードオフは、各トランザクションの入力操作数を増やす選択をする場合、アプリケーションの信頼性要件と一貫性を保つ必要があることを意味します。Oracle BPEL Process Manager から Oracle Business Activity Monitoring へのデータ送信では、バッチ化は有効なオプションです。Oracle Business Activity Monitoring に障害が発生すると Oracle BPEL Process Manager によって操作が再試行されるため、多数の操作をバッチ化してもデータは失われません。

REDO ログ同期用の帯域幅は、RAID またはオペレーティング・システム・ベースのストライプ化メカニズムを使用して、REDO ログ・ファイルを複数のディスクにストライプ化する方法でも増やせます。ログ同期アクティビティでは順次書込みが実行されるため、32K や 64K などのようにストライプ・サイズを大きくすると最高のパフォーマンスを得られます。

注意： REDO ログには RAID-5 を使用しないでください。RAID-5 ストレージ・システムに REDO ログを格納すると、ログ同期アクティビティのパフォーマンスが低下することがわかっています。RAID-5 を使用する必要がある場合は、REDO ログ・グループを格納する 1 つ以上のディスクを RAID-5 構成から除外することを検討してください。

関連項目：

- REDO ログの管理の詳細は、『Oracle Database 管理者ガイド』の第 6 章「REDO ログの管理」を参照してください。
- 高パフォーマンスを実現するための REDO ログ管理を含む、I/O アーキテクチャの高パフォーマンス化の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』の第 8 章「I/O 構成および設計」を参照してください。

8.2 頻繁なログ・スイッチとチェックポイントの回避

Oracle データベースには、少なくとも 2 つの REDO ログ・ファイルが必要です。これによって、1 つのファイルがフルになると別のファイルにスイッチし、そのファイルの領域を非同期式に解放できます。頻繁なログ・スイッチはデータベースのパフォーマンスを大きく低下させるため、スイッチの頻度が許容可能なレベルまで下がるように、REDO ログ・ファイルのサイズを調整する必要があります。原則として、ログ・スイッチは 20 分に 1 回を超えないようにします。

ログ・スイッチの頻度を調べるには、データベースの初期化パラメータ LOG_CHECKPOINTS_TO_ALERT を TRUE に設定します。これによって、ログ・スイッチ・イベントがデータベース・アラート・ログにタイムスタンプとともに書き込まれ、ログ・スイッチが過度に生じていないかどうかをアラート・ログで監視できます。

同様に、増分チェックポイントもパフォーマンス低下の要因になります。データベースのクラッシュ・リカバリ時に処理が必要となる REDO ログの量を制限するために（つまり、システム障害後のデータベースの起動時間を短縮するために）、データベースは一定の間隔で REDO ログにチェックポイントを設定します。LOG_CHECKPOINTS_TO_ALERT パラメータを TRUE に設定すると、これらのチェックポイント・イベントがアラート・ログに書き込まれます。

通常、増分チェックポイントの頻度は暗黙的に設定されます。FAST_START_MTTR_TARGET 初期化パラメータを設定すると、ターゲット・インスタンスに対して、クラッシュ後のクラッシュ・リカバリ時の起動時間を指定できます。FAST_START_MTTR_TARGET を大きな値に設定すると、増分チェックポイントの頻度が減少し、ランタイム・パフォーマンスが向上します（その一方で、インスタンス起動時のクラッシュ・リカバリ時間が犠牲になる）。FAST_START_MTTR_TARGET を小さな値に設定すると、増分チェックポイントの頻度が増加し、インスタンス起動時に必要なクラッシュ・リカバリ時間が短縮されますが、ランタイム・パフォーマンスが犠牲となります。アプリケーションの要件に基づいて、このトレードオフを分析する必要があります。ランタイム・パフォーマンスが最重要要件の場合は、初期化パラメータ FAST_START_MTTR_TARGET を十分大きな値に設定し、増分チェックポイントが発生しないようにします。この場合、チェックポイントはログ・スイッチ時に作成されます。

関連項目：

- REDO ログ・オプションのチューニングの詳細は、3-9 ページの「REDO ログの配置場所のチューニングとサイズ設定」を参照してください。
- 『Oracle Database パフォーマンス・チューニング・ガイド』の「パフォーマンスを考慮したデータベースの構成」および「I/O 構成および設計」

8.3 システム・グローバル領域のチューニング

Oracle Business Activity Monitoring データベースのパフォーマンスは、データベースのシステム・グローバル領域（SGA）の db_cache および log_buffer に対して指定されるサイズと、SGA 全体のサイズ（このサイズは、SGA 内で管理されるオブジェクトのサイズに基づいて指定される）に大きく影響されます。

注意： SGA の最適なサイズは、データベース全体の負荷によって決まります。そのため、ここで説明する Oracle Business Activity Monitoring データベース用の SGA チューニングのガイドラインは、Oracle Business Activity Monitoring データ専用のデータベースに対してのみ適用されます。

Oracle Business Activity Monitoring 専用データベースの SGA は、最小サイズの 1024MB に構成します。SGA サイズは、SGA_MAX_SIZE 初期化パラメータを使用して設定します。SGA 内のメモリー・プールの値は、SGA 自動チューニングを使用して設定するか、または固定サイズのプールを使用します。自動チューニングを指定するには、SGA_TARGET 初期化パラメータを SGA_MAX_SIZE と同じ値に設定します。

SGA 内部のメモリー・プールは手動設定したほうが、よいパフォーマンスが得られる可能性があります。SGA の自動チューニングを無効化するには、SGA_TARGET 初期化パラメータを 0 に設定します。SGA の最大サイズが 1024MB に設定されているインストールで自動チューニングを無効化したときは、初期設定として例 8-1 に示す設定をお薦めします。

例 8-1 自動チューニングを無効化したときの SGA の初期設定例

```
db_cache_size = 820M
java_pool_size = 16M
large_pool_size = 16M
streams_pool_size = 16M
shared_pool_size = 144M
log_buffer = 10485760
```

データベースのキャッシュまたはバッファ・プールには、データベース内の最近アクセスされたデータが保持され、これによって同じデータ・ページの再度の読取りが不要になります。Oracle Business Activity Monitoring データベースの場合、データベースへの挿入パフォーマンスは、主キーの制約チェックの速度に依存しますが、これはデータベース・キャッシュに保持されている各主キーに作成されている索引を介して実行されます。バッファ・プールのヒット率は、データベース・キャッシュのサイズが小さすぎないかどうかを調べる際のよい指標になります。通常のデータベースでは、ヒット率は 95% を超えるようにします。Oracle Business Activity Monitoring データベースでは、ヒット率を 99% 以上に維持することをお薦めします。受信データのデータベースへの挿入がワークロードの大半となるデータベースでは、一般的に、これくらいの値が必要です。主要なワークロードがレポート問合せの場合は、バッファ・プールのヒット率は 95% が有効な目安です。バッファ・プールのヒット率は、次の問合せを使用して取得できます。

```
SELECT NAME, PHYSICAL_READS, DB_BLOCK_GETS, CONSISTENT_GETS,
   1 - (PHYSICAL_READS / (DB_BLOCK_GETS + CONSISTENT_GETS))
   "Hit Ratio" FROM V$BUFFER_POOL_STATISTICS;
```

Oracle Business Activity Monitoring は、ログのバッファ・サイズにも大きく影響されます。データベースにログ・ファイルの同期待ちがあるときは、コミット処理を保留中のすべてのリクエストがログ・バッファに格納されます。ログ・ファイルの同期待ちが多数になると、ログ・バッファがフルになることがあります。これが原因でログ・バッファへのリクエストが実行不能になると、リクエスト元は、ログ・バッファの再試行と呼ばれる機能によって、しばらくしてから再度リクエストを実行します。

ログ・バッファの再試行は、パフォーマンスを大きく低下させる原因となります。データ入力のバッチ・サイズが大きいワークロードや、ログ・バッファの再試行数が多いワークロードがあるときは、ログ・バッファ・サイズの増加が必要となる場合があります。

データベースでバッファの再試行を監視するには、次の問合せを使用します。

```
SELECT NAME, VALUE FROM V$SYSSTAT
  WHERE NAME = 'redo buffer allocation retries';
```

通常、このメトリックは、ワークロードの開始時の値とワークロードの終了時の値を収集することで測定され、これらの 2 つの値の差異がワークロードの実行に必要なログ・バッファの再試行数を示します。テスト・ワークロードの開始時にデータベースを再起動した場合は、開始値が 0 となるため、終了値のみが必要です。10 分程度の間隔では、ログ・バッファの再試行数が 100 未満である必要があります（理想的なケースでは、値は 0 になる）。この値が 100 から 300 の範囲であればパフォーマンスへの影響は許容範囲ですが、これよりも大きいと無視できなくなります。

ログ・バッファやデータベース・キャッシュなどの SGA コンポーネントのサイズを増やすことも重要ですが、SGA の全体サイズを同じだけ増やすことも重要です（コンポーネントのサイズを増加した分、全体サイズも増加する）。そうしないと、サイズを増加したコンポーネントによって、SGA 内の他のオブジェクト用のメモリーが占有されます。

注意： SGA のサイズは、データベースの起動時に、次に大きな 4MB の倍数に切り上げられます。

8.4 削除アクティビティ後のデータベースの再編成

Oracle Business Activity Monitoring をインストールすると、Oracle Business Activity Monitoring 表領域の ORACLEBAM が作成され、データセットとメタデータが保持されます。Oracle Business Activity Monitoring ユーザーがデータセットを作成すると、Oracle Database 表が作成され、データセット内のオブジェクトが格納されます。この表には、SysIterID という名前のシステム生成の主キーがあります。Oracle Business Activity Monitoring ユーザーが Oracle Business Activity Monitoring データセット内の列に Oracle Business Activity Monitoring 索引を作成すると、データベース内の対応する表に、対応する索引が作成されます。

Oracle Business Activity Monitoring 表領域は、自動セグメント領域管理機能が有効化されて作成されます。これによって、Oracle Business Activity Monitoring 表にマップされているセグメント内の空き領域管理が支援され、さらにはデータベースの再編成時にシステムの停止が不要になります。ただし、データベースの手動再編成の必要性が完全になくなるわけではなく、特に、削除アクティビティが相当な回数実行された後などは注意が必要です。

削除アクティビティが相当量実行されると、Oracle Business Activity Monitoring データセット表にマップされているセグメントのデータ密度が散在的になる場合があります。この状況は一般的に内部フラグメンテーションと呼ばれ、表操作の実行時に必要な I/O 回数が急激に増加する原因となります。内部フラグメンテーションがあると、主キーに対する索引を含め、表に対して作成された索引の構造が、多数の削除操作の実行後に、最適なバランス構成でなくなります。そのため、多数の削除操作の実行後は、データベースの再編成を行い、索引の構造を改善して、より少ないデータ・ページに表の行が効率的に収まるようにする必要があります。または、不要になった Oracle Business Activity Monitoring データを削除するスクリプトを記述し、その削除ジョブの一部として再編成を行うこともできます。

Oracle Business Activity Monitoring データセット表のセグメントを再編成するプロシージャは、表の主キーの削除、表の索引の削除、表の行移動の有効化、表領域の縮小、表内の未使用領域の割当て解除、表の主キーの再作成、表の索引の再作成、および表の行移動の無効化の各手順で構成されます。

例 8-2 に、データベース表の _Call_Detail および _Call_Agg に格納されている 2 つの Oracle Business Activity Monitoring データセット Call_Detail および Call_Agg のセグメントを再編成するサンプル・スクリプトを示します。

注意： Oracle Business Activity Monitoring データセット表の再編成プロシージャは、週に 1 度や月に 1 度などの定期間隔でスケジュールするか、相当量の削除アクティビティの実行後や、前述の削除スクリプトとともにスケジュールしてください。

再編成プロシージャを実行する前に、必ずバックアップを実行してください。

例 8-2 Oracle Business Activity Monitoring の表を再編成するサンプル・スクリプト

```
alter table "ORABAM"."_Call_Detail" drop primary key;
alter table "ORABAM"."_Call_Agg" drop primary key;
<drop indexes on "ORABAM"."_Call_Detail">
<drop indexes on "ORABAM"."_Call_Agg">
alter table "ORABAM"."_Call_Detail" enable row movement;
alter table "ORABAM"."_Call_Agg" enable row movement;

< Note: include deletion activity here, if it is added as part of the script>

alter table "ORABAM"."_Call_Detail" shrink space;
alter table "ORABAM"."_Call_Agg" shrink space;
alter table "ORABAM"."_Call_Detail" deallocate unused;
alter table "ORABAM"."_Call_Agg" deallocate unused;
alter table "ORABAM"."_Call_Detail" disable row movement;
alter table "ORABAM"."_Call_Agg" disable row movement;
alter table "ORABAM"."_Call_Detail" add primary key ("SysIterID");
alter table "ORABAM"."_Call_Agg" add primary key ("SysIterID");
```

```
<rebuild indexes on "ORABAM"."_Call_Detail">
<rebuild indexes on "ORABAM"."_Call_Detail">
```

注意： 行移動を有効化して Oracle Business Activity Monitoring 表を実行するよう構成する場合は、例 8-2 に示すスクリプトを変更し、行移動の無効化コマンドを削除する必要があります。

8.5 複数のプラン・モニター・サービスおよびエンタープライズ・リンクの構成

Oracle Business Activity Monitoring ユーザーは、複数のプラン・モニター・サービスと複数のエンタープライズ・リンクを構成することで、パフォーマンスを改善できます。

関連項目：『Oracle Business Activity Monitoring インストレーション・ガイド』

9

Oracle Application Server Wireless メッセージ・サーバーのパフォーマンス・チューニング

この章では、パフォーマンスを向上するために DBA が実行する必要がある Oracle Application Server Wireless メッセージ・サーバー用構成手順について説明します。

パフォーマンスの向上には、RAC 環境と RAC でない環境の両方において、メッセージ・サーバーの一方向処理と双方向処理のパフォーマンスを向上させる SQL 最適化も含まれます。この章で説明する内容は次のとおりです。

9.1 高いパフォーマンスを実現するための Oracle Application Server Wireless メッセージ・サーバーの構成

RAC 環境で様々なベンチマーク・テストを行った結果、高度に協調的に動作するようにアプリケーションを注意深く設計し、マルチノード RAC 環境で実行する高パフォーマンス型 AQ を活用できるようにアプリケーションを適切にチューニングする必要があるということが判明しました。

マルチノード RAC 環境のスループットは、複数のキューを次のように作成すると、大幅に向上する場合があります。すなわち複数のキューはそれぞれ、(a) 各キューと特定の RAC ノード間にアフィニティが設定されるようにし、(b) RAC ノード上のエンキューやデキューのリクエストが、アフィニティが設定されているキューからのみメッセージを取得するように作成します。これによって、共有キューにアクセスする際に複数の RAC ノードからのリクエストで発生するキャッシュ・バッファ待ち状態が防止されます。

パフォーマンスに関する推奨事項の一部は、RAC でない環境にも適用されます。

次の各項では、メッセージ・サーバーのパフォーマンスを向上するために必要な構成手順について説明します。

メッセージ・サーバーのパフォーマンスに影響を与える要因の詳細は、「メッセージ・サーバーのパフォーマンスに影響を与える要因」を参照してください。

9.1.1 概要

この項では、Oracle Application Server 10.1.2.0.2 Wireless メッセージ・サーバーに個別パッチを適用する手順について説明します。

注意： 単一 RAC ノード（または単一データベース・マシン）の環境では、チューニングの推奨事項によってはパフォーマンスに影響しない場合があります。

9.1.2 データベースと OS のチューニング

トランザクション数の多い環境で最適なパフォーマンスを実現するようにデータベースをチューニングする必要があります。データベースをチューニングする際の推奨手順は次のとおりです。

1. アプリケーション・サーバーのインスタンスを停止します。
2. チェックポイントの待機状態を解消します。

   REDO ログのサイズを 500MB 以上に設定し、各グループにメンバーを 3 つ以上作成します（またはグループを 3 つ作成します）。
3. チェックポイントの実行頻度を減らします。

   log_checkpoint_timeout を 0 に設定すると、自動チェックポイントを無効にできます。

   これには、次のコマンドを使用します。

   alter system set log_checkpoint_timeout=0 scope=both;
4. システム・グローバル領域（SGA）のサイズを 1GB 以上にします。
5. 共有プール・サイズを、推奨最小サイズの 300MB 以上にします。
6. RHEL 4 では LMS（ロック・マネージャ・サーバー）プロセスの数を変更します。これは、グローバル・キャッシュ・サービスの主要コンポーネントです。LMS プロセスでは、グローバル・キャッシュの一貫性を維持し、キャッシュ・フュージョンのリクエストに応じて複数のインスタンス間におけるブロック転送を処理します。

   パフォーマンスを向上するには、次のコマンドを使用して、LMS を 1 に設定します。

   alter system set gcs_server_processes = 1 scope=spfile sid='*'

7. RHEL 4 を使用している場合は、Metalink Note 363147.1 に従ってネットワーク・インタフェースをチューニングすることを検討します。
8. アプリケーション・サーバーのインスタンスを起動します。

表 9-1 は、データベース・システム・パラメータと推奨値の一覧です。

表 9-1 データベース・システム・パラメータと推奨値

データベース・システム・パラメータ	推奨値	備考
REDO ログ	500MB 以上 グループごとのメンバーの数は 3	この設定によりチェックポイントの待機状態を解消します。
Log_checkpoint_timeout	0	この設定により自動チェックポイントを無効にします。
システム・グローバル領域	1GB 以上	
共有プール・サイズ	300MB 以上	
LMS プロセス	1	

データベースのチューニングの詳細や、メッセージ・サーバーのパフォーマンスに影響を与える OS 要因の詳細は、「データベースのチューニング」を参照してください。

9.1.3 マルチ RAC 環境のパフォーマンスの最適化

マルチノード RAC 環境のパフォーマンスを最適化する場合のみ、次の手順を実行します。

> **注意：** この手順は、RAC でない環境の設定には使用しないでください。

1. sysdba としてデータベースに接続し、次のスクリプトを実行します。

   SQL> @@trans_tbs_create.sql

   > **注意：** trans_tbs_create.sql により、ASSM（自動セグメント領域管理）機能を持つ TRANS 表領域を新規に作成します。ASSM は、メッセージ・サーバーのパフォーマンスを最適化するために RAC 環境で必要な機能です。

2. プロンプトが表示されたら、このデータベースのデータファイルに使用するベース・ディレクトリのパスを入力します。

   たとえば、「/product/10.1.0/oradata/orcl」と入力します。

   この作成スクリプトはまず、データファイルへのパスを指定せずに表領域の作成を試みます。ASM 機能のある一般的な RAC 環境では、このスクリプトは問題なく実行されます。ただし、RAC でない一般的な環境では、データファイルのパスが指定されていないことを示すエラーが発生し、コマンドは失敗します。スクリプトはこのエラーを認識すると、ユーザーが指定したデータファイル用ベース・ディレクトリとファイル名（trans.dbf）を使用して、表領域の作成を試みます。

   SQL> @@trans_tbs_migrate.sql

   > **注意：** trans_tbs_migrate.sql により、すべてのトランスポート（つまりメッセージ・サーバー）表と索引を、新しい表領域の TRANS に移行するか移動します。

3. すべての Wireless 中間層（OC4J_Wireless コンポーネントと Wireless コンポーネントを含む）を再起動します。

> **注意：** Windows では、Enterprise Manager も起動してください。

9.1.4 メッセージ・サーバー構成

この項では、次のメッセージ・サーバー構成について説明します。

- デキュー・ナビゲーション・モードの更新
- 中間層インスタンスにおけるノード固有 DB 接続文字列の追加

9.1.4.1 デキュー・ナビゲーション・モードの更新

デキュー・ナビゲーション・モードを first から next のメッセージに更新すると、メッセージ優先順位のセマンティックは失われますが、スループットはさらに向上します。これには、次の手順を実行します。

ワイヤレス・ユーザーとしてデータベースにログインし、次のコマンドを実行します。

SQL> @@trans_config_update.sql dequeue_navigation_mode next

> **注意：** メッセージ・サーバーで使用される属性を構成できるよう、属性の追加や更新を行うためのスクリプトである trans_config_update.sql が用意されています。

使用方法 : SQL> @@trans_config_update.sql <attr_name> <attr_val>

デフォルトのデキュー・ナビゲーション・モードに戻すには、次のコマンドを実行します。

SQL> @@trans_config_update.sql dequeue_navigation_mode first

9.1.4.2 中間層インスタンスにおけるノード固有 DB 接続文字列の追加

高いメッセージング・パフォーマンスをマルチノード RAC 環境において実現するには、中間層インスタンス上のメッセージ・サーバー・プロセスと OC4J_Wireless プロセスが、必ず 1 つの RAC ノードにのみ接続する必要があります。このように構成すると、RAC ノードの数が中間層インスタンスの数より多くても、メッセージ・サーバーが別の RAC ノードを使用することはありません。図 9-1 と図 9-2 は、有効な構成を示しています。

図 9-1 3 つの中間層と 3 つの RAC ノード : それぞれが特定の RAC に接続している構成

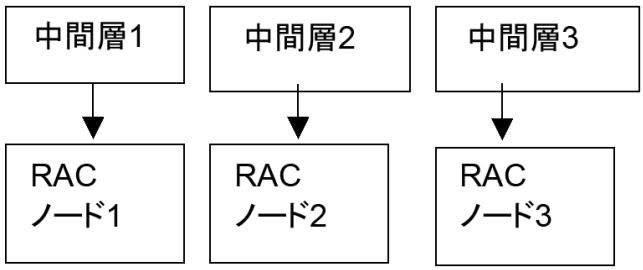
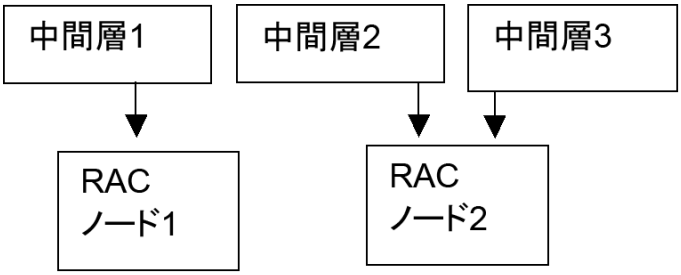


図 9-2 3 つの中間層と 2 つの RAC ノード



中間層インスタンスを前述のように構成するには、これらのプロセスに対して、カスタマイズした接続文字列を JVM パラメータとして指定します。手順は次のとおりです。

1. 各中間層インスタンスに Enterprise Manager としてログインし、ホーム・ページにナビゲートします。
2. 「関連リンク」セクションで、「プロセス管理」リンクをクリックします。
3. "messaging_server" プロセス・タイプの XML セクションを探します。

   たとえば、<process-type id="messaging_server" module-id="messaging"> の XML を探します。
4. そのセクションで、次の XML を追加または変更します。

```
<module-data>
  <category id="start-parameters">
    <data id="java-parameters"
      value="-Xms64M -Xmx256M -Dwireless.db.instance.string=
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<%hostname%>)(PORT=<%port
%>))(CONNECT_DATA=(SID=<%sid%>)))"/>
  </category>
</module-data>
```

   <%hostname%>、<%port%> および <%sid%> のトークンを、各 RAC ノードで適切な値に置換します。
5. "OC4J_Wireless" プロセス・タイプの XML セクションを探します。

   たとえば、<process-type id="OC4J_Wireless" module-id="OC4J"> の XML を探します。
6. そのセクションで、"java-options" データ・ノードを探します（検索対象 XML の例 : <data id="java-options" value="…"/>）。

   次の JVM パラメータを value 属性に追加します。

   -Dwireless.db.instance.string=
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<%hostname%>)(PORT=<%port
%>))(CONNECT_DATA=(SID=<%sid%>)))

   <%hostname%>、<%port%> および <%sid%> のトークンを、各 RAC ノードで適切な値に置換します。

   > **注意 :** 有効であれば、どの DB 接続文字列でも指定できます（"username/password@" の接頭辞は不要）。たとえば、フェイルオーバー・インスタンスを接続文字列に追加することもできます。ただし、SERVICE_NAME ではなく SID を使用することが重要です。ここでの目的は、中間層のプロセスを特定の RAC インスタンスに強制的に接続することであり、リスナーにおいて複数の RAC インスタンス間におけるロード・バランシングを試みることではないからです。
7. すべての中間層マシンの OC4J_Wireless コンポーネントと Wireless コンポーネントを再起動し、変更を有効にします。

9.2 メッセージ・サーバーのパフォーマンスに影響を与える要因

パフォーマンスを向上させるためにメッセージ・サーバーをチューニングするには、次の手順を実行します。

9.2.1 trans_mid と trand_did の索引におけるキャッシュの順序付け

メッセージ・サーバーでは、ステータス ID、メッセージ ID、ストア ID などの様々な ID を生成するために、trans_mid と trans_did の順序が広範囲に使用されます。

推奨事項: アクセス速度を高速にするために、インスタンスの SGA には、50,000 個の順序番号が事前に割り当てられたキャッシュが用意されています。

順序番号の数を 100,000 個に増やすことで、ディスク I/O の削減と CR 待機時間の短縮もできます。ただし、キャッシュのサイズを大きくすることで、他のアプリケーションのメモリー・フットプリントが減少しないように注意する必要があります。

警告: 順序キャッシュのサイズを大きくするとメッセージ ID が不規則になったり、データベースの再起動時に ID に欠番が生じる場合があることを、DBA は認識しておく必要があります。

9.2.2 trans_ids 表の索引の強制使用

trans_ids 表における object_id 列へのアクセス頻度は非常に高いです。

推奨事項: ソフトウェアにより、trans_ids 表の object_id 列に作成された索引が強制的に使用されるようになっています。

テストではメッセージング・パフォーマンスが大幅に向上しました。

9.2.3 ナビゲーション・モードの変更

デキュー操作のデフォルト実装では、first メッセージがナビゲーション・モードとして使用されます（dequeue_option.navigation = DBMS_AQ.FIRST_MESSAGE）。

推奨事項: ナビゲーション・モードを next メッセージに変更してください（dequeue_option.navigation = DBMS_AQ.NEXT_MESSAGE）。詳細は、「デキュー・ナビゲーション・モードの更新」を参照してください。

テストではメッセージング・パフォーマンスが若干向上しました。しかし、このパフォーマンス向上方法には制限事項があります。next メッセージをナビゲーション・モードに設定すると、キューからメッセージを選択するために使用されるカーソルがキャッシュされ、デキュー操作によってキューのスナップショットが取得されます。したがって、キューにエンキューされた新しいメッセージは参照できなくなります。エンキューされた新しいメッセージの優先順位が高く、スナップショット内のメッセージよりも先に処理する必要がある場合、この方法は問題になります。したがって、この変更は、受信するメッセージがすべて同じ優先順位であることが判明している場合にのみ適用してください。

9.2.4 データベースのチューニング

大量のトランザクションを処理するシステムでは、多数の REDO が生成されます。そのため、ログ・スイッチにおいてチェックポイントが頻繁に発生する場合があります。チェックポイントは、データベースの alert.log ファイルに記録されます。

推奨事項: チェックポイントとログ・スイッチの実行頻度を減らしてください。

チェックポイントの待機状態を解消するには、REDO ログのサイズを 500MB 以上に設定し、各グループにメンバーを 3 つ以上作成します（またはグループを 3 つ作成します）。さらに、log_checkpoint_timeout パラメータを 0 に設定すると、自動チェックポイントを無効にできます。

高いスループットを実現するには、リソース共有に関連するデータベース・パラメータを細かくチューニングします。

推奨事項: SGA および共有プールのサイズは、十分なサイズに設定してください。

SGA は 1GB 以上に設定します。

共有プールは 300MB 以上に設定します。

テストではメッセージング・パフォーマンスが大幅に向上しました。

9.2.5 ノード・アフィニティを持つ複数のキュー

2 つの RAC ノードでベンチマーク・テストを行った結果、高度に協調的に動作するようにアプリケーションを注意深く設計し、マルチノード RAC 環境で実行する高パフォーマンス型 AQ を最大限に活用できるようにアプリケーションを適切にチューニングする必要があるということが判明しました。

推奨事項: ソフトウェアにより、複数のキューが作成されています。すなわち複数のキューはそれぞれ、(a) 各キューと特定の RAC ノード間にアフィニティが設定されるようにし、(b) RAC ノード上のエンキューやデキューのリクエストが、アフィニティが設定されているキューからのみメッセージを取得するように作成されています。これによって、共有キューにアクセスする際に複数の RAC ノードからのリクエストで発生するキャッシュ・バッファ待ち状態が防止されます。

テストではメッセージング・パフォーマンスが大幅に向上しました。

9.2.6 ASSM 表領域

複数のノードからメッセージ・サーバーの表に対して頻繁にデータが挿入される場合は、データ・ブロック、表セグメント・ヘッダーおよび他のグローバル・リソース要求に対する同時アクセスにより、パフォーマンス問題が発生していました。

ASSM を使用すると、表の空き領域の管理に関連付けられたデータ構造が、個々のインスタンスに使用できる非結合セットに分割されます。ASSM により、挿入用に十分な空き領域のあるデータ・ブロックが、各インスタンスで個別に管理されるため、個々のインスタンスで動作するプロセス間のパフォーマンス問題が改善されます。

推奨事項: メッセージ・サーバー（トランスポート）の表と索引では、ASSM 表領域を使用してください。「マルチ RAC 環境のパフォーマンスの最適化」を参照してください。

9.2.7 ロード・バランシング

複数のキューが必要になったことで、中間層インスタンスのバランシングも必要になります。

ある中間層インスタンス上のクライアントやメッセージ・サーバーが、使用可能な RAC ノードに不均等に分散して接続している場合は、キューが順番に処理されない場合があります。さらに、あるキューのエンキュー元やデキュー元が多すぎると、その RAC ノードのスループット・パフォーマンスが低下する場合があります。

推奨事項: このような不均衡を防止するには、中間層インスタンスの messaging_server プロセスと OC4J_Wireless プロセスを、1 つの特定 RAC ノードに関連付けてください。詳細は、「中間層インスタンスにおけるノード固有 DB 接続文字列の追加」を参照してください。

9.2.8 エンキューとデキューにおけるスレッドの数

エンキューとデキューにおけるスレッドの数は、全体的なスループットに大きく影響します。そこで、実際のドライバとクライアント・アプリケーションを使用してから、最高の結果になるようにスレッド数を再度チューニングする必要があることに注意してください。RAC ノード上のキューにエンキュー元やデキュー元が多すぎると、そのキュー（および同じ RAC ノード上の他のキュー）のスループットが低下する場合があります。エンキュー元やデキュー元（これらは、ドライバ送信処理スレッドとドライバ受信処理スレッド、またはクライアント送信処理スレッドとクライアント受信処理スレッドで制御される）の最適数は、RAC ノード、中間層マシン、ディスク・ストレージなどの物理的なハードウェア特性にも左右されます。

推奨事項: 各エンキュー処理スレッド（送信の場合はクライアント送信処理スレッド、受信の場合はドライバ受信処理スレッド）で生成される負荷が均等であると仮定すると、設定するスレッド数は、負荷が数分間続いた後のキュー内のメッセージ数によって異なります。キューがほとんど空の状態で、エンキュー・スレッド（送信の場合はクライアント送信処理スレッド）の数の値を大きくすると、エンキュー率が上昇する場合があります。キューのサイズが一定範囲を超えて増加し始めた場合（クライアント側でメッセージのエンキューを行う処理によって増加し続けた場合）は、デキュー・スレッド（送信の場合はドライバ送信処理スレッド）の数の値を大きくします。各 RAC ノード上のキューのサイズが一定の値（100,000 メッセージのテストで 2,000 メッセージ未満）になったとき、最高の結果になります。その差は微妙であるため、最高の結果を実現するには、繰り返してテストを実行する必要があります。ドライバとクライアントの両方においてスレッドの数が少なすぎても多すぎても、実現可能な最大スループットの値は小さくなります。

9.2.9 DB パラメータの aq_tm_processes

Oracle Streams AQ のタイム・マネージャ・プロセスは、init.ora パラメータである AQ_TM_PROCESSES で制御できます。このパラメータをゼロ以外の値に設定することで、キュー・メッセージの時間を監視したり、遅延や有効期限のプロパティが指定されているメッセージを処理できます。テスト環境においてこのパラメータを 10 に設定してテストを行いました。そうすると、AQ のキュー・モニター・プロセスで問題が発生し、あるプロセスでは短いループの中でそのプロセスの CPU 使用率が 100% になり、他のプロセスに使用できる CPU 時間が減少しました。この問題は、対応するプロセスを強制終了しても、そのプロセスのかわりに別のプロセスが作成されるため、解決しませんでした。オラクル社の AQ チームでは、この問題に対処するためにこのパラメータ値を 9 に設定することをお薦めします。

AQ チームによると、Oracle Streams AQ リリース 10.1 ではこれがコーディネータ・スレーブ型アーキテクチャに変更されているため、実際に Oracle Streams AQ や Streams がシステムで使用されている場合、コーディネータのプロセスが自動的に作成されます。このプロセスは QMNC と呼ばれ、システムの負荷に応じてスレーブを動的に作成します。スレーブの名前は qXXX で、Oracle Streams AQ や Streams のために様々なバックグラウンド・タスクを実行します。プロセス数は自動的に決定されて絶えず調整されるため、AQ_TM_PROCESSES を設定する必要はありません。しかし、テスト環境でこのパラメータを削除すると、メッセージ・サーバーによって必要以上のキューが作成されることが判明しました。値を 9 に設定した場合は、メッセージ・サーバーは適切に動作し、必要な数だけキューが作成されました。

そこで、次の現象が発生するかどうかを調べてください。

- メッセージ・サーバーがチャネル用に作成したドライバ・キューの数が、RAC ノードの数より多くなる（サービス・キューの場合も同様）。
- メッセージ・サーバーにおいてキューを作成するのに要する時間が数分になる（これは、「select * from wireless.trans_queue」、「select * from wireless.trans_driver_queue」および「select * from wireless.trans_service_queue」を使用して調べます）。
- メッセージ・サーバーを停止した後も、Infra ノードの DB プロセスの CPU 使用率が 100% になり、数分経過してもプロセスが停止しない。

前述の現象のいずれかが発生した場合は、次の作業を実行します。

- すべての中間層を停止します。
- aq_tm_processes パラメータの値を確認します。

  SQL> show parameter aq_tm_processes;

- 値が 0 または 10 の場合は、9 に設定します。sysdba としてこの値を変更する例を次に示します。

  SQL> alter system set aq_tm_processes=9 scope=memory sid='*';

注意： 値を永続的に変更するには、SPFile から PFILE を作成して値を更新し、更新された PFILE を使用して SPFile を作成する必要があります。

- ワイヤレス・ユーザーとしてデータベースに接続し、次のコマンドを実行します。

  SQL> execute transport.drop_all_queues;

- 中間層の 1 つを再起動し、前述の問題が発生しないことを確認します。
- 残りの中間層を再起動します。

9.2.10 RHEL4 での RAC インターコネクト

次の情報は、http://metalink.oracle.com の Metalink Note 363147.1 の抜粋です。

9.2.10.1 目的

RHEL3 から RHEL4 にアップグレードすると、RAC のインターコネクト環境のシステム構成によっては、パフォーマンスが低下することがあります。このノートでは、問題を特定して解決するための手順について説明します。

9.2.10.2 適用範囲とアプリケーション

この項の内容は、次のユーザーを対象としています。

- RHEL3 から RHEL4 にアップグレードしたユーザー
- RHEL4 を使用している新規ユーザー

グローバル・キャッシュ・ブロック損失と IP フラグメンテーション障害によって、RAC インターコネクトのパフォーマンスが低下します。RAC クラスタの Linux OS を RHEL3（2.4.21）から RHEL4（2.6.9）にアップグレードすると、グローバル・キャッシュ・ブロック損失率が 1.5/ 秒まで上昇し、GC CR ブロック損失が上位 5 位の待機イベントになることが判明しました。また、スループット・パフォーマンスは最大 30% 低下しました。GC ブロック損失数は、パケット再組立てエラーの数に関連しています。これらの現象は、リリース番号によって異なりません。RHEL4 を使用するシステムで発生する可能性があります。

さらなる分析により、RHEL3 と RHEL4 との間で、Intel 社製 e1000 ドライバのイーサネット・フロー制御設定が変更されたことが、この現象の原因であることが判明しました。

この問題は、次の社内システム構成とお客様のシステム構成で確認されています。

- Intel Xeon ベースのサーバー（32 ビット版と 64 ビット版 EM64T の両方）
- Intel GigE イーサネット・カードと e1000 ドライバ
- Redhat Enterprise Linux 4 upgrade 2（32 ビット版と 64 ビット版のカーネル）
- クラスタ環境で Oracle RAC データベース（リリース番号を問わず）を実行している場合

同様の問題が発生しているかどうかを確認する方法は次のとおりです。

- Oracle AWR または Statspack レポートを実行し、次のことを確認します。
  - 「global cache cr block lost」または「global cache current block lost」が上位 5 位の待機イベントに出力されている。
  - 「global cache block lost」の統計値がゼロ以外の値になっており、ドロップ率が毎秒 0.4 ブロックを超えている。
- OS で netstat -s コマンドを実行し、IP 統計出力から次のことを確認します。
  - 「packet reassembles failed」の値がゼロではなく、その値が「global cache block lost」に連動している。

9.2.10.3 問題の解決方法

RHEL4（2.6.9）では、e1000 ネットワーク・アダプタの RX フロー制御がデフォルトで無効になりますが、RHEL3 ではこれが有効になります。このアダプタの RX フロー制御を有効にすると、ブロック損失とパケット再組立て障害が発生しなくなります。

フロー制御を設定するには、次の構文を使用します（ここでは、例として eth1 を使用します）。

- eth1 のフロー制御設定を確認する構文 : ethtool –a eth1
- eth1 の RX フロー制御設定を有効にする構文 : ethtool –A eth1 rx on
- 再起動後もフロー制御を保持する手順は次のとおりです。

  /etc/modprobe.conf を編集して、modprobe.conf に次の行を追加します。

  options e1000 FlowControl=1,1

  再起動します。設定が保存されます。

完全な構文は、FlowControl value: 0-3（0= なし、1=RX のみ、2=TX のみ、3=Rx と Tx）です。各アダプタの値はカンマで区切って指定します。

9.2.10.4 問題の解説

e1000 ネットワーク・アダプタのある Intel Xeon サーバーの場合、RHEL3（2.4）では、ネットワーク・アダプタの RX（スイッチから送信されたフレームに対する e1000 レスポンス）フロー制御がデフォルトで有効になります。しかし、RHEL4（2.6.9）にアップグレードすると、RX フロー制御はデフォルトで無効になるため、ブロック損失が発生します。この設定に変更される理由は現在も調査中です。フロー制御のデフォルト設定は、カーネルの e1000 ドライバのバージョンと e1000 カード本体のバージョンの両方によって異なることに注意してください。様々なチップセットと様々な e1000 ドライバをベースとする Intel GigE NIC の可用性も、この問題を複雑にしています。前述の問題の組合せを調べ、推奨される解決策を使用する必要があります。この動作は、RHEL4 やすべての Intel GigE NIC で発生するわけではありません。Linux 2.4 と Linux 2.6.9 では、どちらの TX フロー制御もデフォルトでは無効にされます。TX フロー制御を有効にすることはお薦めしません。

9.3 RAC インスタンス障害の処理

この項では、RAC インスタンスのいずれかに障害が発生した場合（またはオフラインになった場合）に、メッセージ・サーバーが実行する処理について説明します。

ドライバ・キューとサービス・キューが作成されると、メッセージ・サーバーは 1 次ノードと 2 次ノードをキューに割り当てます。デフォルトでは、1 次ノードがアクティブ・ノード（現在キューにアフィニティが設定されているノード）になります。1 次ノードに障害が発生すると、AQ は 2 次ノードをアクティブにします。つまり、アフィニティを 2 次ノードに割り当てます。

メッセージ・サーバーはこのような変更を追跡でき、かつ変更を追跡する必要があります。ノードに接続する際に、エンキュー用とデキュー用の適切なキューを選択する必要があるためです。

現在の実装では、軽量の DBMS ジョブが一定の間隔（2 分ごと）でキューのアフィニティを監視し、キューと RAC ノードとのマッピング（メッセージ・サーバーで使用）を自動更新します。

したがって、ノードに障害が発生した場合、メッセージ・サーバーではアフィニティが変更されたキューからメッセージの処理を続行します。障害のあったノードがオンラインに復旧すると、マッピングは DBMS ジョブによって自動的に再び割り当てられます。DBA が何もしなくても、単一インスタンスの障害は自動的に処理されます。

RAC インスタンスを停止する予定があり、その停止時間中に DBMS ジョブを待たずに、キューと RAC ノードとのマッピングを更新する場合は、RAC ノードを停止した後、次の手順を使用して更新を強制適用することができます。

1. Sqlplus で wireless/<%wireless_pwd%>@<%db-connect-string%> として接続します。

2. 次の SQL コマンドを実行し、キューと RAC ノードとのマッピング（メッセージ・サーバーで維持されているマッピング）を更新します。

   SQL> execute transport.monitor_queues;

9.4 RAC の再構成

この項では、RAC ノードと中間層が永続的に追加されたり削除された場合に必要となるメッセージ・サーバー再構成手順について説明します。

9.4.1 RAC ノードの追加または削除

RAC ノードを永続的に追加したり削除する場合は、メッセージ・サーバーを再構成して、キューを適切に作成できるようにする必要があります。そのためには、次の手順を実行します。

1. すべての中間層インスタンスで OPMN を停止します。
2. $ORACLE_HOME/wireless/repository/sql ディレクトリに移動します。
3. **sqlplus <wireless/<%wireless_pwd%>@<%db_conn_string%>** を実行します。
4. 既存のキューを削除します。

   execute transport.drop_all_queues;

5. すべての中間層インスタンスで OPMN を起動します。

   キューは起動時に自動的に作成されます。

9.4.2 中間層の追加または削除

最適なスループットを実現するにはロード・バランシングが不可欠であるため、メッセージ・サーバー用に構成されている RAC ノードの数と同数以上の中間層インスタンスを用意することが重要です。

RAC ノードの数より中間層インスタンスの数が少ない場合、メッセージ・サーバーは、中間層インスタンスが接続されている RAC ノードのみを使用します。

逆に、RAC ノードの数より中間層インスタンスの数が多い場合は、第 1.4.1 項の説明に従って、中間層インスタンスの DB 接続文字列を構成できます。ただし、RAC ノード上のキューにエンキュー元やデキュー元が多すぎると、そのキュー（および同じ RAC ノード上の他のキュー）のスループットが低下する場合があることに注意してください。エンキュー元やデキュー元（これらは、ドライバ送信処理スレッドとドライバ受信処理スレッド、またはクライアント送信処理スレッドとクライアント受信処理スレッドで制御される）の最適数は、RAC ノード、中間層マシン、ディスク・ストレージなどの物理的なハードウェア特性に左右されます。

9.5 テスト・シナリオと結果

この項では、テスト・シナリオの設定とテスト結果について説明します。

9.5.1 設定の詳細

この項では、マシン、ソフトウェア、ハードウェアおよび RAC の設定について説明します。

9.5.1.1 マシンの設定の詳細

テストを実行する際、6 台の同一マシンを使用しました。

3 台のマシンに中間層をインストールし、3 台のノードで構成される RAC にデータベースを配置しました。IM は別のマシンにインストールしました。

9.5.1.2 マシンのハードウェアの詳細

6 台のマシン（3 台は中間層、残り 3 台は RAC ノード）はいずれも、4GB のメモリー、3.0GHz の Intel Xeon CPU が 2 基（OS からは 4 基に見える）搭載されています。

9.5.1.3 ソフトウェアの詳細

OS: Red Hat Enterprise Linux AS R4（Nahant Update 3）

Oracle Application Server 10.1.2.0.2

9.5.1.4 RAC の設定の詳細

DB および CRS は 10.2.0.2 です。ASM は 2.0 です。

データベース・サーバーは RAC 内に構成しました。DB サーバーが同じデータベースにアクセスできるようにするには、3 台のサーバー間で共有できるネットワーク・ストレージ・ユニットが必要です。これは、ファイバ・チャネルでサーバーに接続される SAN ディスク（表 9-2 を参照）を使用することで解決しました。SAN を使用すると、データベース・サーバーからのディスク・アクセスが高速にでき容易になります。また、Web ベースのクライアント・ツールを使用して構成も簡単にできます。

表 9-2 SAN ディスク

ベンダー	EMC
モデルとバージョン	EMC Clarion CX 300
ディスク	33GB のディスクが 3 台、RAID 0 構成のファイバ・チャネル、disk0、disk1、disk2。33GB のディスクが 2 台、ファイバ・チャネル、RAID 1 構成、disk3。
補足情報	disk0、disk1 および disk2 は、DATADG ASM ディスク・グループに追加され、データベースのデータファイルと SPFile の格納に使用されます。 これらは、db01、db02 および db03 から、/dev/sdc、/dev/sdd および /dev/sdedisk3 として 3 つのパーティションに分割されて参照できます。最初の 2 台は、投票ディスク（/dev/sdb1）とクラスタ・リポジトリ（/dev/sdb2）の RAW デバイスとして使用されます。3 台目のディスク（/dev/sdb3）は、LOGDG Asm ディスク・グループに追加され、データベースの REDO ログの格納に使用されます。

9.5.2 一方向テスト・シナリオ

一方向テストは、パフォーマンス向上用コードの実装前と実装後に実行しました。テスト・シナリオの詳細と結果は次のとおりです。

9.5.2.1 テスト・シナリオ

テストの実行に使用したメッセージの合計数 : 各中間層で 100,000 メッセージ

実行したクライアント・プログラムの数 : 各中間層マシンで 1 つ

実行したメッセージ・サーバーの台数 : 各中間層マシンで 1 台

9.5.2.2 クライアント構成

各中間層からメッセージを送信するクライアント・スレッドの数 : 10 個

各中間層から各クライアント・スレッドが送信するメッセージの数 : 10,000 個

送信処理間の間隔 : 0

9.5.2.3 ドライバ構成

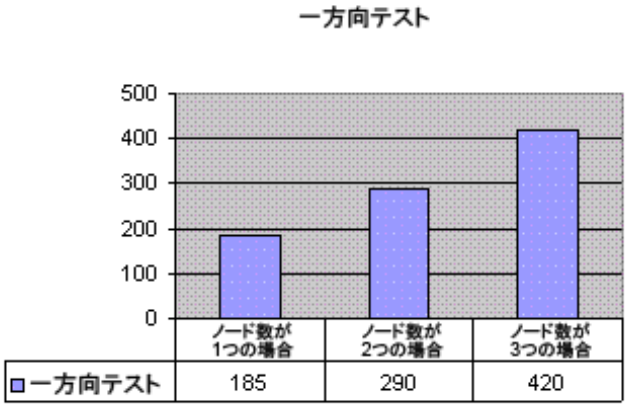
各中間層メッセージ・サーバー・インスタンスにあるドライバ送信スレッドの数 : 8 個

> **注意 :** 使用したドライバは、実際の SMSC や SMSC シミュレータには接続しないため、ダミーです。実際の動作環境シナリオで実際にテストすると、テスト結果が記載と若干異なる場合があります。

9.5.2.4 テスト結果

パフォーマンス向上用パッチを適用した後、一方向スループットの結果は図 9-3 のようになりました。

図 9-3 一方向スループット・テストの結果



9.5.2.5 テスト・データ

表 9-3 は、ダミー・ドライバを使用して生成された一方向テスト・データを示しています。

表 9-3 一方向テストの詳細

テストの詳細	送信スループット（メッセージ数 / 秒）
テスト 1: 1 つの DB ノードの構成によるパフォーマンス向上措置実行後（メッセージ・サーバーとデータベースのチューニング後）の一方向（送信）テストの結果	185
テスト 2: 2 つの DB ノードの構成によるパフォーマンス向上措置実行後（メッセージ・サーバーとデータベースのチューニング後）の一方向（送信）テストの結果	290
テスト 3: 3 つの DB ノードの構成によるパフォーマンス向上措置実行後（メッセージ・サーバーとデータベースのチューニング後）の一方向（送信）テストの結果	420

> **注意 :** 記載されたスループットは、安定状態時におけるスループットです（システムとテストにはウォーミングアップ時間がありました）。

9.5.3 双方向テスト・シナリオ

双方向テストは、パフォーマンス向上措置の適用前と適用後に実行しました。テスト・シナリオの詳細と結果は次のとおりです。

9.5.3.1 テスト・シナリオ

テストの実行に使用したメッセージの合計数 : 各中間層で 20,000 メッセージ

実行したクライアント・プログラムの数 : 各中間層マシンで 1 つ

実行したメッセージ・サーバーの台数 : 各中間層マシンで 1 台

クライアント構成 :

各中間層から実行するクライアントの受信スレッド・セットの数 : 4 個

ドライバ構成 :

各中間層メッセージ・サーバー・インスタンスにあるドライバ送信スレッドの数 : 3 個

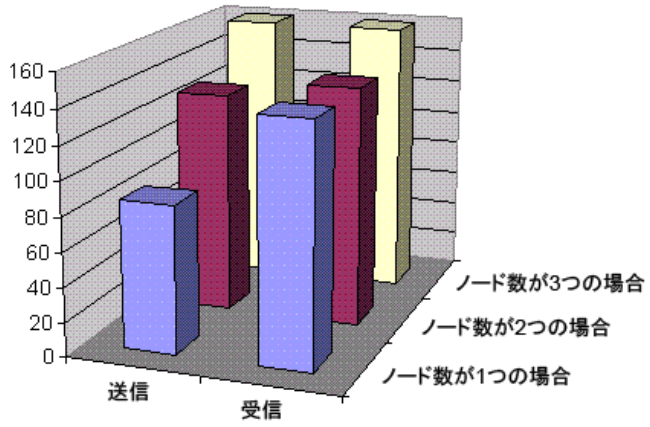
各中間層メッセージ・サーバー・インスタンスにあるドライバ受信スレッドの数 : 2 個

> **注意 :** 使用したドライバはダミーで、実際の SMSC や SMSC シミュレータには接続しません。実際の動作環境シナリオで実際にテストすると、テスト結果が記載と若干異なる場合があります。

9.5.3.2 テスト結果

双方向テスト・シナリオの結果は、次のとおりです。

図 9-4 双方向スループット・テストの結果



	送信	受信
ノード数が1つの場合	86	140
ノード数が2つの場合	130	140
ノード数が3つの場合	160	160

9.5.3.3 テスト・データ

表 9-4 は、ダミー・ドライバを使用して生成された双方向テスト・データを示しています。

表 9-4 双方向テスト・データ

テストの詳細	送信スループット（メッセージ数 / 秒）	受信スループット（メッセージ数 / 秒）
テスト 1: 1 つの DB ノードの構成によるパフォーマンス向上措置実行後（メッセージ・サーバーとデータベースのチューニング後）の双方向（送受信）テストの結果	86	140
テスト 2: 2 つの DB ノードの構成によるパフォーマンス向上措置実行後（メッセージ・サーバーとデータベースのチューニング後）の双方向（送受信）テストの結果	130	140
テスト 3: 3 つの DB ノードの構成によるパフォーマンス向上措置実行後（メッセージ・サーバーとデータベースのチューニング後）の双方向（送受信）テストの結果	160	160

注意： 記載されたスループットは、安定状態時におけるスループットです（システムとテストにはウォーミングアップ時間がありました）。

A

組込みパフォーマンス・ツールを使用した監視

この付録には、次の項が含まれています。

A.1 Oracle Application Server の組込みパフォーマンス・メトリックの要約

パフォーマンスの監視には、Application Server Control コンソールの「**パフォーマンス**」サブタブを使用するか、「**管理**」サブタブの「JMX」領域からシステム MBean ブラウザを使用できます。または、Oracle Application Server の組込みパフォーマンス・メトリックを表示します。

この付録では、Oracle Application Server `AggreSpy` サーブレットまたは `dmstool` コマンドを使用した、組込みパフォーマンス・メトリックの参照方法について説明します。

表 A-1 は、パフォーマンス・メトリックの参照に使用できる組込みツールを要約したものです。

表 A-1 Oracle Application Server の組込み監視コマンド

コマンド	説明
`AggreSpy`	`AggreSpy` は、Oracle Application Server インスタンスのパフォーマンス・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` を実行できるのは、OC4J のホーム・インスタンスが実行されている場合に限ります。ホームという名前の OC4J インスタンスは、デフォルトで `AggreSpy` をサポートしています。 **注意**: 場合によっては、`AggreSpy` の使用には、ホーム・インスタンスの起動が必要なことがあります。
`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レポートするレポート間隔を指定できます。さらに、このコマンドでは、サイト上で使用可能なすべての組込みパフォーマンス・メトリックのリストをテキスト形式のレポートで表示します。`dmstool` コマンドは、`$ORACLE_HOME/bin` ディレクトリ（UNIX システムの場合）または `%ORACLE_HOME%¥bin` ディレクトリ（Windows システムの場合）にあります。

関連項目： 付録 C「パフォーマンス・メトリック」

A.2 基本インストールでの 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、OC4J、Oracle Process Manager and Notification Server およびその他の Oracle Application Server コンポーネント・プロセスを含む、Oracle Application Server プロセスのメトリックを表示します。

A.2.1 AggreSpy 表示の使用方法

`AggreSpy` は、DMS Spies およびメトリック表の 2 つの領域にメトリックを編成します。

- DMS Spies は、親プロセスのタイプと親プロセスの番号によって、使用可能なメトリックを表示します。個々の DMS Spies を選択すると、関連するプロセスに対して収集されたすべてのメトリックを、テキスト形式で表示できます。
- メトリック表は、メトリック表のタイプ別に使用可能なメトリックを表示します。複数の OC4J が実行されている場合は、複数の OC4J インスタンスからの OC4J メトリックが含まれます。個々のメトリック表を選択すると、指定したタイプのすべてのメトリックを、表形式で表示できます。たとえば、メトリック表では OC4J サーブレットおよび Oracle Process Manager and Notification Server プロセスに関連するメトリックを表示できます。

DMS メトリック表は、`oc4j_servlet` や `opmn_process` などの名前で識別されます。`AggreSpy` では、メトリック表という用語は、組込みパフォーマンス・メトリックの表名を示します。

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://`*`host`*`:`*`port`*`/dmsoc4j/AggreSpy`

ここで、

host は、HTTP リスナーを提供する OC4J のホストを示します。たとえば、`tv.us.oracle.com` です。

port は、OC4J の HTTP リスナー・ポートを示します。たとえば、`8888` です。

注意： `AggreSpy` を実行できるのは、OC4J のホーム・インスタンスが実行されている場合に限ります。ホームという名前の OC4J 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 A-1 に、サンプルの `AggreSpy` の表示を示します。ここには、2 つのフレームが表示されています。1 つには DMS Spies と DMS メトリック表のリストが表示され、もう 1 つには DMS Spies またはメトリック表の選択した値が表示されます。

`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。

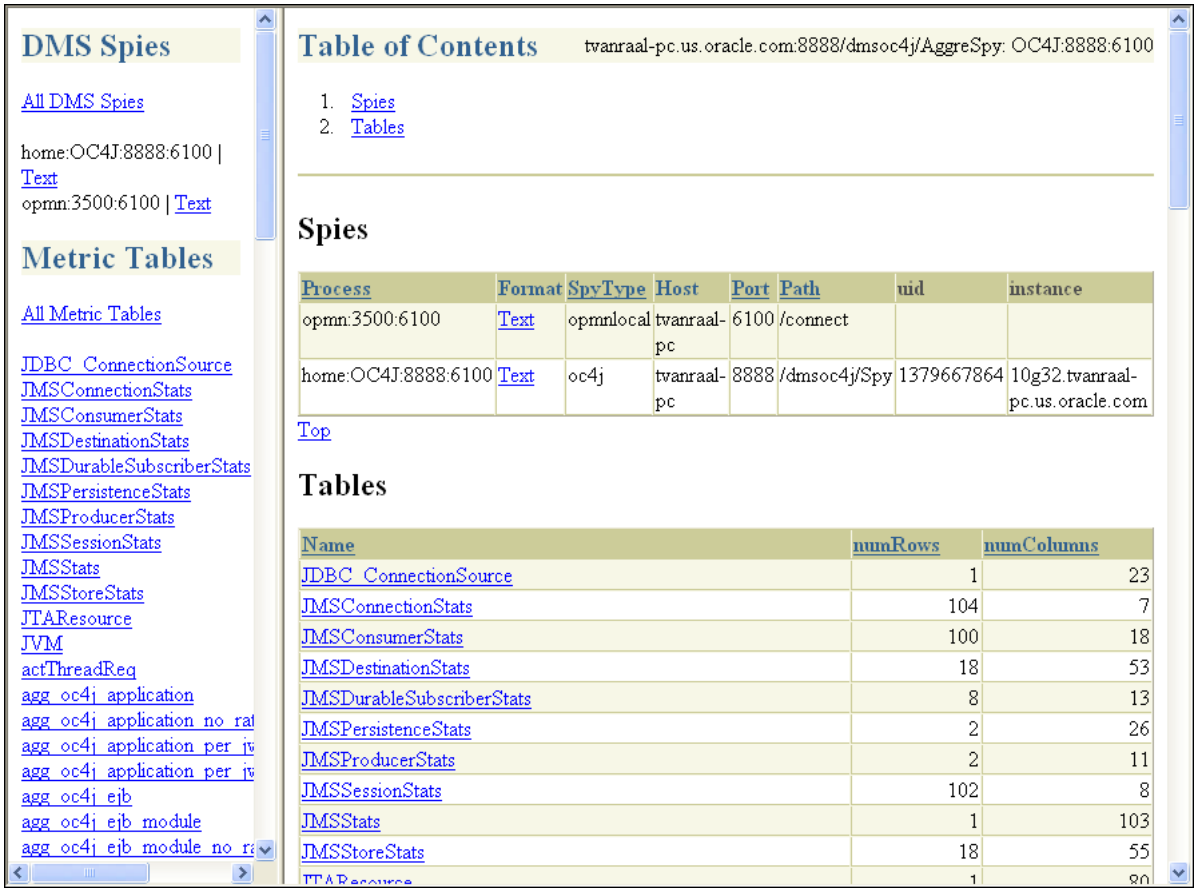
- 左フレーム内のリンクを使用して、DMS Spies およびメトリック表へアクセスする。
- 列ヘッダーをクリックして、メトリック表の行をソートする。
- 各メトリック表に表示されるメトリック定義リンクをクリックして、メトリック表のメトリックを説明する表を表示する。

`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・データを表示する必要があります。`AggreSpy` を最初に使用する場合、多くのフィールドや DMS Spies のリストに、現在のメトリック表が完全に含まれていない場合があります。しばらくして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、`AggreSpy` にメトリック表の完全なリストが表示されます。

注意： `AggreSpy` を使用するには、OC4J のホーム・インスタンスが実行されている必要があります。

基本インストールでは、ホーム・インスタンスは `opmnctl startall` コマンドで、または Application Server Control コンソールを使用して「**起動**」をクリックすることで起動されます。

図 A-1 AggreSpy によるパフォーマンス・メトリックの表示



A.3 Web サーバーでの 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、OC4J、Oracle Process Manager and Notification Server、およびその他の Oracle Application Server コンポーネント・プロセスを含む、Oracle Application Server プロセスのメトリックを表示します。

注意： この項では、AggreSpy によるパフォーマンス・メトリックを Oracle HTTP Server を使用して表示する方法について説明します。Oracle HTTP Server は、選択した拡張インストールのタイプに基づいてシステムにインストールされます。Oracle HTTP Server がシステムにインストールされていない場合は、この項で説明するコマンドは機能しません。

この項には、次の項目が含まれています。

- AggreSpy 表示の使用方法
- Web サーバーでの AggreSpy の URL とアクセス制御
- Web サーバーでの AggreSpy の URL とアクセス制御
- 複数のインスタンスでロード・バランシングを行う際の AggreSpy の制限

A.3.1 Web サーバーでの AggreSpy 表示の使用

AggreSpy は、DMS Spies およびメトリック表の 2 つの領域にメトリックを編成します。

- DMS Spies は、親プロセスのタイプと親プロセスの番号によって、使用可能なメトリックを表示します。個々の DMS Spies を選択すると、関連するプロセスに対して収集されたすべてのメトリックを、テキスト形式で表示できます。
- メトリック表は、メトリック表のタイプ別に使用可能なメトリックを表示します。複数の OC4J が実行されている場合は、複数の OC4J インスタンスからの OC4J メトリックが含まれます。個々のメトリック表を選択すると、指定したタイプのすべてのメトリックを、表形式で表示できます。たとえば、メトリック表では OC4J サーブレット、Oracle HTTP Server モジュール、および Oracle Process Manager and Notification Server プロセスに関連するメトリックを表示できます。

注意： AggreSpy を使用して DMS メトリックを表示するには、システムでプロキシを使用するように構成されている場合、ブラウザを構成し、ローカルホストに対するプロキシの使用を無効化する必要がある場合があります。Oracle Application Server では、デフォルトでローカルホストへのアクセスのみ許可されています。詳細は、A-6 ページの「Web サーバーでの AggreSpy の URL（プロキシ・サーバーが設定されている場合）」を参照してください。

DMS メトリック表は、Oracle HTTP Server メトリックに対する ohs_server のような名前で識別されます。AggreSpy では、メトリック表という用語は、組込みパフォーマンス・メトリックの表名を示します。

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://host:port/dms0/AggreSpy
```

ここで、

host は、Oracle HTTP Server ホストを示します。たとえば、tv.us.oracle.com です。

port は、Oracle HTTP Server のリスナー・ポートを示します。たとえば、7777 です。

注意： AggreSpy を実行できるのは、OC4J のホーム・インスタンスが実行されている場合に限ります。ホームという名前の OC4J インスタンスは、デフォルトで AggreSpy をサポートしています。OracleAS Infrastructure を使用する場合は、ホーム・インスタンスを起動しないと AggreSpy を使用できません。ホーム・インスタンスはデフォルトで OracleAS Infrastructure とともにインストールされますが、起動はされないためです。

図 A-1 に、サンプルの AggreSpy の表示を示します。ここには、2 つのフレームが表示されています。1 つには DMS Spies と DMS メトリック表のリストが表示され、もう 1 つには DMS Spies またはメトリック表の選択した値が表示されます。

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。

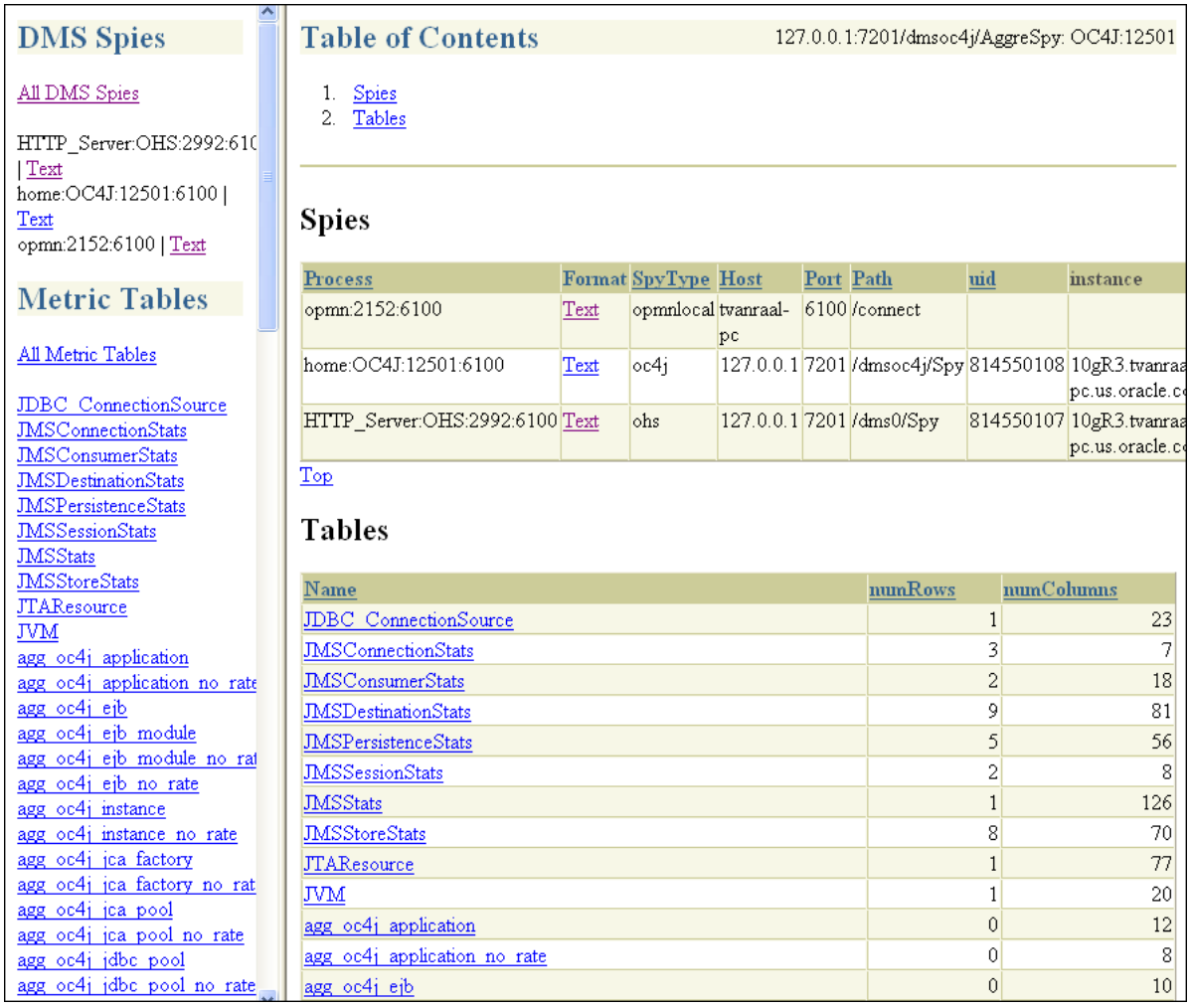
- 左フレーム内のリンクを使用して、DMS Spies およびメトリック表へアクセスする。
- 列ヘッダーをクリックして、メトリック表の行をソートする。
- 各メトリック表に表示されるメトリック定義リンクをクリックして、メトリック表のメトリックを説明する表を表示する。

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・データを表示する必要があります。AggreSpy を最初に使用する場合、多くのフィールドや DMS Spies のリストに、現在のメトリック表が完全に含まれていない場合があります。しばらくして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、AggreSpy にメトリック表の完全なリストが表示されます。

注意： AggreSpy を使用するには、OC4J のホーム・インスタンスが実行されている必要があります。ホーム・インスタンスが停止しているときに、http://<*host*>:<*port*>:/dms0/AggreSpy から AggreSpy をリクエストすると、HTTP 500 内部サーバー・エラーが返されます。

J2EE インストールでは、ホーム・インスタンスは opmnctl startall コマンドで、または Application Server Control コンソールを使用して「**起動**」をクリックすることで起動されます。

図 A-2 AggreSpy によるパフォーマンス・メトリックの表示



A.3.2 Web サーバーでの AggreSpy の URL（プロキシ・サーバーが設定されている場合）

ブラウザがプロキシ・サーバーを使用するように構成されている場合、ローカルホストの AggreSpy にアクセスするには、ローカルホストでのプロキシの使用を無効化するようにブラウザを構成する必要があります。ローカルホストでのプロキシ・サーバーの使用を無効化する手順は、使用しているブラウザによって異なります。

A.3.3 Web サーバーでの AggreSpy の URL とアクセス制御

デフォルトでは、dms0/AggreSpy URL がリダイレクトされ、このリダイレクト先は保護されます。そして、ローカルホスト（127.0.0.1）からのみ、AggreSpy サーブレット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するには、UNIX システム上では $ORACLE_HOME/Apache/Apache/conf/dms.conf ファイルを、Windows システム上では %ORACLE_HOME%¥Apache¥Apache¥conf¥dms.conf ファイルを変更して、監視する Oracle Application Server を実行するシステムの DMS 構成を変更する必要があります。

例 A-1 に、dms.conf のサンプルのデフォルト構成を示します。この構成では、AggreSpy は、ローカルホスト（127.0.0.1）からメトリックにアクセスするように制限されます。表示されているポート 7200 は、ご使用のインストールによって異なる場合があります。

例 A-1 DMS メトリックへのローカルホスト・アクセスが設定されたサンプルの dms.conf ファイル

```
# proxy to DMS AggreSpy
Redirect /dms0/AggreSpy http://localhost:7200/dmsoc4j/AggreSpy
#DMS VirtualHost for access and logging control
Listen 127.0.0.1:7200
OpmnHostPort http://127.0.0.1:7200
<VirtualHost 127.0.0.1:7200>
    ServerName 127.0.0.1
```

dms.conf の構成を変更して、DMS メトリックを提供または表示するホストを指定すると、ローカルホスト以外のシステム上のユーザーが、http://*host*:*port*/dms0/AggreSpy から DMS メトリックにアクセスできるようになります。

> **注意：** dms.conf を変更すると、セキュリティに影響します。サイトに対するセキュリティの影響を理解している場合のみ、このファイルを変更してください。ローカルホスト以外のシステムにメトリックを公開した場合、重要な Oracle Application Server 内部のステータスおよびランタイム情報が、他のサイトに表示される可能性があります。

ローカルホスト（127.0.0.1）以外のシステムからメトリックを表示するには、次の手順を実行します。

1. 例 A-1 に示すローカルホスト 127.0.0.1 の値を、メトリックを提供するサーバーの名前に変更して、dms.conf を変更します。サーバー名は httpd.conf ファイルの ServerName ディレクティブから取得します（たとえば、tv.us.oracle.com です）。
2. 例 A-2 に、ローカルホスト（127.0.0.1）以外のシステムからアクセスできるように設定が更新された dms.conf のサンプルを示します。

例 A-2 DMS メトリックへのリモートホスト・アクセスが設定されたサンプルの dms.conf ファイル

```
# proxy to DMS AggreSpy
Redirect /dms0/AggreSpy http://tv.us.oracle.com:7200/dmsoc4j/AggreSpy
#DMS VirtualHost for access and logging control
Listen tv.us.oracle.com:7200
OpmnHostPort http://tv.us.oracle.com:7200
<VirtualHost tv.us.oracle.com:7200>
ServerName tv.us.oracle.com
```

3. Application Server Control コンソールまたは opmnctl コマンドを使用して、Oracle HTTP Server を再起動するか、一度停止してから起動します。次に例を示します。

```
%opmnctl restartproc process-type=HTTP_Server
```

または

```
%opmnctl stopproc process-type=HTTP_Server
%opmnctl startproc process-type=HTTP_Server
```

関連項目： Oracle HTTP Server のアクセス制御の詳細は、『Oracle Application Server セキュリティ・ガイド』を参照してください。

A.3.4 複数のインスタンスでロード・バランシングを行う際の AggreSpy の制限

Oracle Application Server で複数のインスタンスが実行されている場合、AggreSpy は期待どおりに動作しません。Oracle HTTP Server の mod_oc4j コンポーネントが、OC4J に対するリクエストのロードを複数のインスタンス間で分散するとき、AggreSpy は、ローカルホスト（127.0.0.1）ではないシステムの結果をレポートすることがあります。

注意： この場合は、AggreSpy ではなく、dmstool を使用することをお薦めします。

A.4 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、Oracle Application Server インスタンスの 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 A-2 に、dmstool のコマンドライン・オプションを示します。表 A-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 コマンドは、$ORACLE_HOME/bin ディレクトリ（UNIX の場合）または %ORACLE_HOME%¥bin ディレクトリ（Windows の場合）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。

関連項目：

「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」（A-10 ページ）

DMS メトリックのリストとその説明は、付録 C「パフォーマンス・メトリック」を参照してください。

A.4.1 dmstool のアクセス制御

デフォルトでは、dmstool をローカルホスト（127.0.0.1）から実行している場合のみ、メトリックが表示されます。リモートホストで実行されている Oracle Application Server からメトリックを表示する場合、ローカルホストで -a オプションを使用して dmstool を実行し、リモート Oracle Application Server インスタンスの dms.conf ファイルを更新する必要があります。このファイルは、$ORACLE_HOME/Apache/Apache/conf/ ディレクトリ（UNIX の場合）または %ORACLE_HOME%¥Apache¥Apache¥conf¥ ディレクトリ（Windows の場合）にあります。

dmstool を使用したメトリックへのアクセス制御に必要な構成の変更は、dms0/AggreSpy へのアクセスの場合と同様です。

関連項目： 「Web サーバーでの AggreSpy の URL とアクセス制御」（A-6 ページ）

表 A-2 dmstool のコマンドライン・オプション

オプション	説明
-a[ddress] opmn://*host*[:*port*]	デフォルトでは、-a オプションがない場合、dmstool は、同じ $ORACLE_HOME を持つ Oracle Application Server インスタンスからメトリックを取得します。dmstool が Oracle Process Manager and Notification Server（OPMN）と同じ $ORACLE_HOME で実行されている場合、-a オプションは必要ありません。 左記のように opmn:// 接頭辞と引数を使用して -a を指定すると、DMS メトリックを作成する OPMN 制御下にある Oracle Application Server プロセスを監視できます（たとえば、OPMN により制御される一部のプロセス、Oracle Web Cache では DMS メトリックは公開されていません）。 ここで、 *host* は、OPMN プロセスが稼動しているホストのドメイン名または IP アドレスです。 *port* には、メトリックを提供する OPMN リクエストのポートを指定します。このリクエスト・ポートは、$ORACLE_HOME/opmn/conf/opmn.xml で指定します。 たとえば、次のように、opmn.xml でリクエスト・ポート（request="6003"）を指定します。 <notification-server> <port local="6100" remote="6200" request="6003"/> . . </notification-server> dmstool -a を使用してリモート・システムからメトリックをリクエストする場合、システムがメトリックを提供するように構成されている必要があります（デフォルトでは、ローカルホストから DMS メトリックにアクセスできます）。 関連項目：「Web サーバーでの AggreSpy の URL とアクセス制御」（A-7 ページ）
-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 オプションは、-list オプションとともに使用できません。
-dump [format=xml]	-dump オプションを使用して dmstool を実行すると、使用可能な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 15 ～ 20 分などの間隔で定期的に実行して、Oracle Application Server 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。 -dump オプションでは、format=xml 問合せもサポートしています。この問合せをコマンドラインの最後で使用すると、XML 形式でメトリックが出力されます。
-help	dmstool のコマンドライン・オプションをリストします。

表 A-2 dmstool のコマンドライン・オプション（続き）

オプション	説明
`-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 5 秒です。`interval` 引数は、`-list` オプションとともに使用できません。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` オプションを使用して指定した間隔が実際の間隔と異なることがあります。
`-l[ist] [-table]`	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 `-table` オプションとともに使用して、すべてのメトリック表の名前をリスト表示します。 `-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。
`-reset [-table metric_table]`	指定したメトリックをリセットします。-table オプションを使用して実行すると、指定したメトリック表にあるすべてのメトリックがリセットされます。 Event および phaseEvent メトリックは 0 にリセットされます（更新されなかったことになる）。State メトリックは現在の値にリセットされます（現在の値で開始されたことになる）。 **注意** : `reset` オプションを使用すると、Application Server Control コンソールが値の計算とレポートに使用する情報がリセットされる場合があります。
`-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 C「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。

A.4.2 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

Oracle Application Server 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` オプションを使用して `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リクエストするときに、`dmstool` で使用可能な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` では、サーバー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 C「パフォーマンス・メトリック」

A.4.3 特定のパフォーマンス・メトリックの値をレポートするための dmstool の使用方法

1 つの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 の稼動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検出します。

```
% dmstool -list | grep JVM/upTime.value
/system1/OC4J:12502:6100/JVM/upTime.value
```

2. 次のように、引数としてこのメトリックの完全なパス名を指定して、dmstool を実行し、メトリックの値を表示します。

```
% dmstool /system1/OC4J:12502:6100/JVM/upTime.value
  Wed Dec 21 15:37:08 PST 2005
/system1/OC4J:12502:6100/JVM/upTime.value   159312  msecs
```

dmstool を使用して、更新されたメトリックを表示します。デフォルトの繰返し間隔は 5 秒なので、次のコマンドでは 5 秒ごとに更新されたメトリック値が表示されます。-count オプションを使用して、dmstool で値をレポートする回数を制限します。

次に例を示します。

```
% dmstool /system1/OC4J:12502:6100/JVM/upTime.value -count 2
Wed Dec 21 15:39:38 PST 2005
/system1/OC4J:12502:6100/JVM/upTime.value   310031  msecs
Wed Dec 21 15:39:43 PST 2005
/system1/OC4J:12502:6100/JVM/upTime.value   314516  msecs
```

注意： 場合によっては、メトリック名のフルパスにスペースが含まれることがあります。パスにスペースが含まれる場合は、シェルがそのメトリック名を 1 つの引数として dmstool に渡せるように、dmstool コマンドラインでパスを引用符で囲む必要があります。

A.4.4 Interval および Count オプションを使用した dmstool の実行

アプリケーションで完了したリクエストを 1 分間隔で監視するには、次の dmstool コマンドを使用して、コマンドラインでメトリック名を指定します。

```
% dmstool -i 60 -c 120 ¥
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed
```

このコマンドでは、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。

```
Tue Oct 12 14:43:43 PDT 2004
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed    8576 ops

Tue Oct 12 14:44:43 PDT 2004
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed    8581 ops

Tue Oct 12 14:45:43 PDT 2004
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed    8588 ops
.
.
.
```

A.4.5 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、Oracle Application Server インスタンスからすべてのメトリックが標準出力に表示されます。

次のコマンドは使用可能なすべてのメトリックを表示します。

```
% dmstool –dump
```

-dump オプションを使用して dmstool コマンドを 15 ～ 20 分などの間隔で定期的に実行して、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、問題が発生した場合に役立てることができます。

A.4.6 すべてのメトリックとそのメトリック値を XML 形式でレポートするための dmstool の使用方法

メトリック・データを処理する必要がある場合、dmstool コマンドラインで format=xml 問合せを使用して、すべてのメトリック・データを XML 形式でレポートします。

次のコマンドは、使用可能なすべてのメトリックを XML 形式で表示します。

```
% dmstool -dump format=xml
```

A.4.7 メトリック値をリセットするための dmstool の使用方法

メトリック値をリセットする場合は、dmstool コマンドラインで reset オプションを使用して、複数のメトリックの値、または指定されたメトリック表のすべてのメトリックの値をリセットします。

reset オプションを使用すると、Event および phaseEvent メトリックは 0 にリセットされます（更新されなかったことになる）。State メトリックは現在の値にリセットされます（現在の値で開始されたことになる）。

次のコマンドは、指定したメトリックをリセットします。

```
% dmstool -reset /system1/OC4J:3000:6004/JVM/upTime.value
```

次のコマンドは、指定したメトリック表をリセットします。

```
% dmstool -reset /system1/OC4J:3000:6004/JVM/upTime.value
```

注意： reset オプションによって、Application Server Control コンソールが値の計算とレポートに使用する情報がリセットされる場合があります。

A.4.8 リモートにある Oracle Application Server 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 Oracle Application Server インスタンスからメトリックがレポートされます。

注意： Oracle Application Server では、デフォルトで、dmstool はローカルホストからのみメトリックにアクセスできます。ローカルホスト以外のシステムからのアクセスをサポートするには、dms.conf を編集する必要があります。DMS のアクセス制御の詳細は、A-7 ページの「Web サーバーでの AggreSpy の URL とアクセス制御」を参照してください。

次のコマンドにより、-a オプションで指定したように、Oracle Application Server インスタンス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 opmn://system1:6003 -list
```

dmstool の -a オプションを使用する場合、接頭辞 opmn:// とともに引数を指定し、メトリックを取得するホスト名と OPMN リクエストのポート番号も含めます。ポートは、Oracle Application Server のメトリックを提供する OPMN リクエスト・ポートを指定します。これは、$ORACLE_HOME/opmn/conf/opmn.xml（UNIX の場合）および %ORACLE_HOME%¥opmn¥conf¥opmn.xml（Windows の場合）内の <notification-server> 要素の request 属性で指定します。

関連項目： 「Web サーバーでの AggreSpy の URL とアクセス制御」（A-7 ページ）

A.5 AggreSpy を使用したパフォーマンス・メトリックの表示（スタンドアロン OC4J の場合）

Oracle Application Server を使用せずに、スタンドアロン・モードで OC4J を使用している場合は、AggreSpy サーブレットを使用して OC4J メトリックにアクセスできます。

OC4J をスタンドアロンで実行している場合、次の URL から AggreSpy を使用してパフォーマンス・メトリックにアクセスします。

```
http://myhost:myport/dms0/AggreSpy
```

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

表 A-3 に、スタンドアロン・モードの OC4J にのみ適用する dmstool オプションを示します。さらに、表 A-2 に示されるオプションも、dmstool に適用されます（ただし、opmn:// 接頭辞を持つ -a オプションを除く）。

表 A-3 dmstool コマンドライン・オプション（スタンドアロン OC4J のみ）

オプション	説明
-a[ddress] *host*[:*port*][*path*],...	スタンドアロン OC4J システムでは、-a オプションを使用します。これは http:// プロトコルを指定します。次のようになります。 *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* は、関連するポートを指定します。

A.6 Windows システムにおける組込みパフォーマンス・メトリックの使用方法

Windows システムで Oracle Application Server を使用する場合、特定の DMS メトリックを表示するには統計の収集を有効にする必要があります。0 以外の値が予測されるときに DMS メトリックが 0 をレポートした場合、システムで統計の収集が無効になっていることがあります。統計の収集が無効になっている Windows システムで統計の収集を有効にするには、次のレジストリ・エントリの値を 0 に設定します。

```
HKEY_LOCAL MACHINE¥SYSTEM¥CurrentControlSet¥Services¥PerfProc¥Performance¥Disable
Performance Counters
```

注意： レジストリの編集に誤りがあると、システムに深刻な問題が発生する場合があります。レジストリを変更する前に、コンピュータの重要なデータをバックアップしておいてください。

B

アプリケーションへの DMS のインストルメント

Oracle Dynamic Monitoring Service（DMS）を使用すると、アプリケーション開発者、サポート・アナリスト、システム管理者およびその他のユーザーは、アプリケーション固有のパフォーマンス情報を測定できます。この章では DMS について説明し、DMS を使用した Oracle Application Server Java アプリケーションへのインストルメント方法について、サンプルのアプリケーションを使用して示します。

注意： Oracle Application Server には、組込みメトリックがいくつか用意されています。DMS を使用してアプリケーションのインストルメントを行うと、この組込みメトリックのセットに新しいメトリックが追加されます。

この章には、次の項が含まれています。

関連項目： 付録 C「パフォーマンス・メトリック」

B.1 DMS パフォーマンス・メトリックについて

ダイナミック・モニタリング・サービス（DMS）API を使用すると、Oracle Application Server アプリケーションにパフォーマンス・インストルメントを追加できます。実行時、OC4J は DMS メトリックと呼ばれるパフォーマンス情報を収集します。開発者、システム管理者およびサポート・アナリストは、この情報を利用してシステム・パフォーマンスを分析したり、システムの状態を監視できます。

この項には、次の項目が含まれています。

- アプリケーションへの DMS メトリックのインストルメント
- DMS メトリックの監視
- DMS 用語の説明（Noun および Sensor）
- DMS ネーミング規則

> **注意：** OC4J を含む Oracle Application Server コンポーネントには、いくつかの定義済メトリックが用意されています。定義済メトリックの一覧は、付録 C「パフォーマンス・メトリック」を参照してください。

B.1.1 アプリケーションへの DMS メトリックのインストルメント

DMS **インストルメント**とは、アプリケーション・コードに DMS コールを挿入するときに行う処理のことです。DMS API を使用することで、アプリケーションのパフォーマンス情報を測定、収集および保存できます。

DMS メトリックを作成する場合、アプリケーション開発者は、イベントが発生するタイミング、重要な時間隔の開始と終了、および事前に処理済の値が変更されるタイミングなどを DMS に通知する DMS API コールを追加します。実行時、DMS はメトリックをメモリーに格納し、ユーザーがそのメトリックを保存または表示できるようにします。

Oracle Application Server には、組込み DMS メトリックが含まれています。アプリケーションに DMS コールを追加することで、この組込みメトリックのセットは拡張されます。DMS コールをアプリケーションにインストルメントする場合は、組込みメトリックで使用されている API と同じ API を使用します。また、収集したメトリックを保存して表示するには、組込みメトリックで使用されているものと同じ監視ツールを使用します。

> **ヒント：** 「DMS インストルメントの Java アプリケーションへの追加」（B-8 ページ）

B.1.2 DMS メトリックの監視

DMS メトリックの監視とは、パフォーマンス・メトリックを取得する処理のことです。アプリケーションの実行時、DMS はメトリックをメモリーに格納します。これにより、ユーザーはコンソールでメトリックを表示したり、Web ブラウザを使用してメトリックを表示できます。

Oracle Application Server には、`dmstool` や `AggreSpy` サーブレットなどの、DMS メトリックを表示したり保存するためのランタイム・ツールが用意されています。

例 B-1 は、`dmstool` を使用して出力されるメトリックのセットを示しています。

例 B-1 dmstool を使用した dmsDemo のサンプル・メトリックのセット

```
/dmsDemo [type=n/a]
 /dmsDemo/BasicBinomial [type=MathSeries]
  computeSeries.active:        0       threads
  computeSeries.avg:   21.181818181818183      msecs
  computeSeries.completed:     11      ops
  computeSeries.maxActive:     1       threads
  computeSeries.maxTime:       93      msecs
  computeSeries.minTime:       0       msecs
  computeSeries.time:  233     msecs
  lastComputed.value:  184756
  loops.count: 604     ops
```

関連項目： 付録 A「組込みパフォーマンス・ツールを使用した監視」

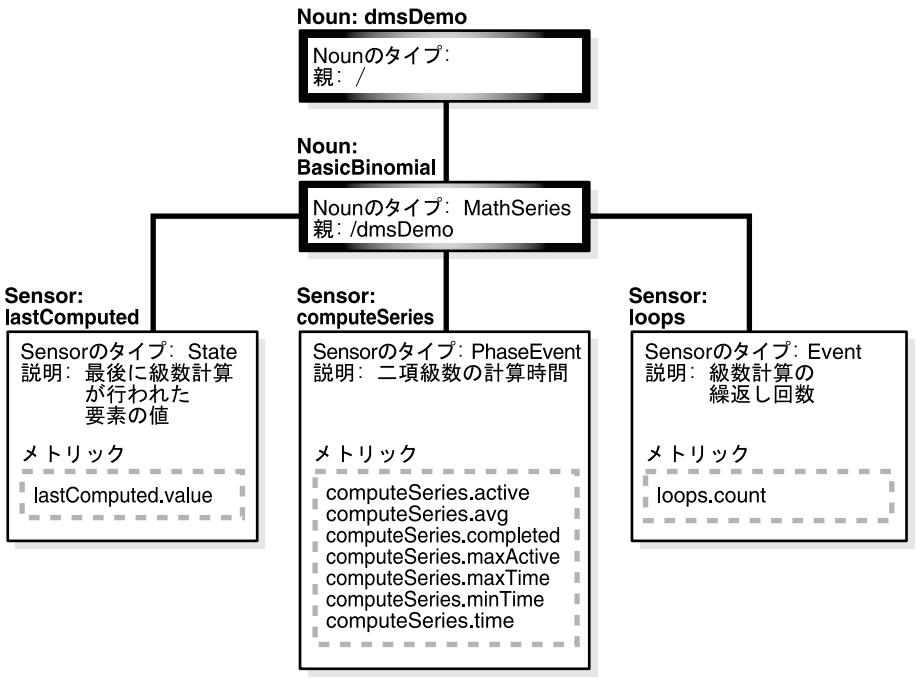
B.1.3 DMS 用語の説明（Noun および Sensor）

この項では、DMS を使用するために理解する必要がある用語について説明します。図 B-1 は、この章で説明するデモ・アプリケーションのメトリックと例 B-1 に示すメトリックに対応する、DMS メトリックのセットの構成を示したものです。

この項には、次の項目が含まれています。

- DMS メトリック
- DMS Sensor
- DMS Noun
- DMS ロールアップ Noun
- DMS オブジェクトの関係

図 B-1 dmsDemo アプリケーションのサンプル・メトリックの構成



B.1.3.1 DMS メトリック

DMS メトリックは、開発者、システム管理者およびサポート・アナリストが、システム・パフォーマンスを分析したりシステムの状態を監視するために使用するパフォーマンス情報を追跡します。

B.1.3.2 DMS Sensor

DMS Sensor は、パフォーマンス・データを測定します。DMS は Sensor によってメトリックのセットを定義および収集します。メトリックには、常に Sensor に含まれるものと任意で含まれるものがあります。

B.1.3.2.1 DMS PhaseEvent Sensor DMS **PhaseEvent Sensor** は、コード内の特定セクションの開始から終了までにかかる時間を測定します。PhaseEvent Sensor を使用すると、あるメソッドまたはコード・ブロックの処理にかかる時間を追跡することができます。

DMS では、PhaseEvent Sensor の処理にかかる平均時間、最大時間および最小時間など、PhaseEvent に関連するオプションのメトリックを計算できます。

表 B-1 は、PhaseEvent Sensor で使用可能なメトリックを説明したものです。

表 B-1 DMS PhaseEvent Sensor のメトリック

メトリック	説明
sensor_name.time	フェーズ *sensor_name* の処理にかかった合計時間を示します。 デフォルトのメトリック : time は、PhaseEvent Sensor のデフォルトのメトリックです。
sensor_name.completed	プロセス開始以降、完了したフェーズ *sensor_name* の回数を示します。 オプションのメトリック
sensor_name.minTime	completed の回数分繰り返されたフェーズ *sensor_name* の中で、最短処理時間を示します。 オプションのメトリック
sensor_name.maxTime	completed の回数分繰り返されたフェーズ *sensor_name* の中で、最長処理時間を示します。 オプションのメトリック
sensor_name.avg	フェーズ *sensor_name* の平均処理時間を、（合計時間）/（フェーズが完了した回数）として計算します。 オプションのメトリック
sensor_name.active	DMS 統計の収集時、フェーズ *sensor_name* 内にあったスレッド数を示します（この値は時間の経過によって異なります）。 オプションのメトリック
sensor_name.maxActive	プロセス開始以降、フェーズ *sensor_name* で処理された同時スレッドの最大数を示します。 オプションのメトリック

B.1.3.2.2 DMS Event Sensor DMS **Event Sensor** は、システム・イベントをカウントします。継続時間の短いシステム・イベントや、継続時間よりも発生ポイントが重要なシステム・イベントを追跡する場合は、DMS Event Sensor を使用します。

表 B-2 は、Event Sensor に関連するメトリックを説明したものです。

表 B-2 DMS Event Sensor のメトリック

メトリック	説明
sensor_name.count	プロセス開始以降、イベントが発生した回数を示します。*sensor_name* は、DMS インストルメント API で指定されている Event Sensor の名前です。 デフォルト : count は、Event Sensor のデフォルトのメトリックです。Event Sensor では、これ以外のメトリックは使用できません。

B.1.3.2.3 DMS State Sensor DMS **State Sensor** には、事前に処理済の値を割り当てます。State Sensor は、Java プリミティブの値または Java オブジェクトのコンテンツを追跡します。integer、double、long および object などの型がサポートされています。State Sensor は、システムの状態情報を追跡したり、イベントに関係しないパフォーマンス・メトリックを収集する場合に使用します。たとえば、State Sensor を使用して、キューの長さ、プール・サイズ、バッファ・サイズまたはホスト名などを示すことができます。

表 B-3 は、State Sensor のメトリックを説明したものです。State Sensor は、デフォルトのメトリック value およびオプションのメトリックをサポートします。minValue および maxValue のオプション・メトリックは、State Sensor が（integer、double および long などの）数値の Java プリミティブを表す場合にのみ、State Sensor に適用されます。

表 B-3 DMS State Sensor のメトリック

メトリック	説明
sensor_name.value	*sensor_name* の作成時に割り当てられた型を使用して、*sensor_name* のメトリック値を示します。 デフォルト : value は、State のデフォルトのメトリックです。
sensor_name.count	*sensor_name* が更新された回数を示します。 オプションのメトリック
sensor_name.minValue	起動後、*sensor_name* に割り当てられた最小値を示します。 オプションのメトリック
sensor_name.maxValue	起動後、*sensor_name* に割り当てられた最大値を示します。 オプションのメトリック

B.1.3.3 DMS Noun

DMS Noun（Noun）はパフォーマンス・データを編成します。各 Sensor は、関連付けられたメトリックとともに階層構造で Noun に付加されます。Noun を使用すると、ファイル・システムのディレクトリ構造と同様の方法で、DMS メトリックを編成できます。たとえば、Noun は、クラス、メソッド、オブジェクト、キュー、接続、アプリケーション、データベースまたはその他のオブジェクトなど、測定の様々な対象を示すことができます。

Noun のタイプには、収集対象となるメトリックのセットを表す名前が使用されます。たとえば、組込みメトリックの Noun のタイプ oc4j_servlet は、各 J2EE アプリケーション内の各 Web モジュールのサーブレットそれぞれについて収集されるメトリックを表します。また Noun のタイプ JVM は、サイトで現在実行されている各 Java プロセス（OC4J）に関するメトリックのセットを表します。

注意 : 付録 C「パフォーマンス・メトリック」では、この Noun のタイプはメトリック表の名前として示されます。

Noun のネーミング計画では、階層のルートに「/」を使用し、各 Noun はルート Noun または親 Noun の下にあるコンテナとして動作します。

関連項目 : 付録 C「パフォーマンス・メトリック」

B.1.3.4 DMS ロールアップ Noun

DMS ロールアップ Noun は、集計 Noun のセットをリクエストするインストルメントを含めるときに、DMS によって生成される Noun です。ロールアップ Noun には、指定された Noun タイプの子孫 Noun 内の Sensor のセットからのメトリックが含まれます。ロールアップ Noun には、サマリー情報も含まれます。

関連項目： 「子孫 Noun の DMS データのロールアップ」（B-21 ページ）

B.1.3.5 DMS オブジェクトの関係

この項では、DMS メトリック、Sensor および Noun それぞれのオブジェクトの関係と属性について説明します。

表 B-4 は、DMS オブジェクト間の関係を説明したものです。図 B-1 は、サンプルのメトリックのセットを使用して表 B-4 に示した関係を図示したものです。

表 B-4 DMS オブジェクトの関係と属性

オブジェクト	含まれる対象	属性
Noun	Sensor またはその他の Noun	名前、Noun のタイプ、親
Sensor	メトリック	名前、説明、Sensor のタイプ、親 Sensor のタイプには、PhaseEvent、Event および State があります。
メトリック	値	名前、単位の指定

B.1.4 DMS ネーミング規則

DMS の名前の定義には、特定のガイドラインが適用されます。このガイドラインに従うことにより、DMS メトリック・レポートの参照ユーザーは、アプリケーション全体および Oracle Application Server コンポーネント全体のメトリックを簡単に理解することができます。

注意： このネーミング規則はガイドラインとして使用してください。各ルールには、例外がある可能性があります。名前は、できるかぎり明確に定義してください。矛盾が存在する場合は、例外を作成する必要があります。

この項には、次の項目が含まれています。

- 一般的な DMS のネーミング規則
- 一般的な DMS のネーミング規則とキャラクタ・セット
- Noun および Noun のタイプのネーミング規則
- Sensor のネーミング規則

B.1.4.1 一般的な DMS のネーミング規則

DMS メトリックの名前は、Sensor 名、「.」およびメトリックで構成されます。たとえば、`computeSeries.time`、`loops.count` および `lastComputed.value` は有効な DMS メトリックの名前です。

Sensor 名は、「.」や派生を含まない単純な文字列です。たとえば、`computeSeries`、`loops` および `lastComputed` などです。Sensor のフルネームは、その Sensor が関連付けられた Noun の名前、区切り文字、Sensor 名で構成されます。たとえば、`/dmsDemo/BasicBinomial/computeSeries`、`/dmsDemo/BasicBinomial/loops` および `/dmsDemo/BasicBinomial/lastComputed` などです。

Noun 名は、区切り文字を含まない単純な文字列です。たとえば、`BasicBinomial` は Noun 名です。Noun のフルネームは、その親のフルネーム、区切り文字、Noun 名で構成されます。たとえば、`/dmsDemo/BasicBinomial` などです。

B.1.4.2 一般的な DMS のネーミング規則とキャラクタ・セット

DMS の名前はできるかぎり簡潔にしてください。また、Noun や Sensor の名前を定義する場合は、可能であれば特殊文字（空白、スラッシュ、ピリオド、括弧、カンマ、制御文字など）を使用しないでください。

表 B-5 は、DMS において置換される、名前内の特殊文字を示します。

表 B-5 置換される DMS ネーミングにおける特殊文字

文字	DMS における置換文字
スペース「 」またはピリオド「.」	アンダースコア「_」
制御文字	アンダースコア「_」
「<」	「(」
「>」	「)」
「&」	「^」
「”」（二重引用符）	「‘」（バッククウォート）。二重引用符がバッククウォートに置き換えられます。
「’」（一重引用符）	「‘」（バッククウォート）。一重引用符がバッククウォートに置き換えられます。

注意： Oracle Application Server には、組込みメトリックがいくつか含まれています。Oracle Application Server の組込みメトリックが、この DMS のネーミング規則に常に従っているとは限りません。

B.1.4.3 Noun および Noun のタイプのネーミング規則

Noun には、特定のエンティティを識別する名前を付けてください。

Noun のタイプには、収集対象となるメトリックのセットを明確に表す名前を使用する必要があります。たとえば、Servlet という Noun のタイプは、その Noun で、特定のサーブレットに固有のメトリックを収集することを示します。

Noun のタイプの名前は、他の DMS の名前と区別するために大文字で始めてください。同じタイプのすべての Noun は、同じ Sensor のセットを持ちます。

B.1.4.4 Sensor のネーミング規則

次のリストは、DMS Sensor のネーミング規則の概要を示します。

1. Sensor の名前は、簡潔で説明的なものにしてください。冗長にならないように、Sensor の名前には、Noun の名前の一部やタイプを含めないようにします。
2. また Sensor の名前には、各メトリックの単位の指定を含めないでください。
3. Sensor を説明するために複数の単語を組み合せる必要がある場合は、最初の単語は小文字で、続く単語は大文字で始めます。たとえば、`computeSeries` のようになります。
4. 一般的に Sensor 名で、「/」を使用することは避けることをお薦めします。ただし、名前に「/」を使用することが道理にかなっている場合があります。Noun または Sensor 名に「/」を使用した場合、DMS メソッドの文字列で Sensor を使用するとき、区切り文字として、パスにまったく存在しない「,」または「_」などを「/」のかわりに使用する必要があります。これによって、「/」は区切り文字ではなく、Noun または Sensor 名の一部であることが正しく認識されるようになります。

たとえば、次のような名前の子 Noun があるとします。

```
examples/jsp/num/numguess.jsp
```

そして次の文字列を使用し検索することができます。

```
,oc4j,default,WEBs,defaultWebApp,JSPs,example/jsp/num/numguess.jsp,service
```

ここで、区切り文字は「,」です。

5. Event Sensor や PhaseEvent Sensor の名前は、英単語を動詞 + 名詞という形式で組み合せて定義します。たとえば、`activateInstance` や `runMethod` のようになります。PhaseEvent によって、関数、メソッドまたはコード・ブロックを監視する場合は、実行されるタスクを可能なかぎり明確に反映した名前を使用します。
6. State Sensor の名前には名詞を使用します。場合によっては、この State で追跡した値の内容を説明する形容詞が前に付きます。たとえば、`lastComputed`、`totalMemory`、`port`、`availableThreads`、`activeInstances` のようになります。
7. 混乱を避けるため、Sensor の名前として、「.time」、「.value」または「.avg」を使用しないでください。理由は、表 B-1、表 B-2、表 B-3 で示しているとおり、これらが Sensor のデフォルトまたはオプションのメトリックだからです。

B.2 DMS インストルメントの Java アプリケーションへの追加

Java アプリケーションのパフォーマンス情報を収集するには、DMS インストルメントを既存のアプリケーションに追加するか、DMS インストルメントを含む新しいアプリケーションを作成します。

この章に示す DMS のサンプルは、次の Oracle Technology Network の Web サイトから入手できます。

```
http://www.oracle.com/technology/tech/java/oc4j/demos/index.html
```

DMS の `demo.zip` ファイルには、すぐにデプロイ可能な .ear ファイルと、ビルド手順付きのソース・コードが含まれています。このデモには、`BasicBinomial.java` および `ImprovedBinomial.java` の 2 つのサーブレットが含まれています。

BasicBinomial サーブレットは、DMS API を使用して DMS Sensor を追加する方法を示します。

ImprovedBinomial サーブレットは BasicBinomial サーブレットを拡張したもので、改善されたコードを BasicBinomial と比較して示します。ImprovedBinomial サーブレットは、さらに詳細な情報を収集するための、より負荷のかかるメトリックの追加方法についても示しています。

この章に示す例の完全なコードは、サンプル・コードを参照してください。

DMS インストルメントを使用するには、次の手順を実行して DMS コールを追加します。

- DMS import のインクルード
- パフォーマンス・データの編成
- タイミングを計測するメトリックの定義と使用
- カウントを計測するメトリックの定義と使用
- 状態情報を記録するメトリックの定義と使用（State Sensor）

B.2.1 DMS import のインクルード

DMS を使用するには、DMS import を追加する必要があります。次の例は、サンプル・アプリケーション BasicBinomial.java で必要な import を示しています。

```
import oracle.dms.instrument.DMSConsole;
import oracle.dms.instrument.Event;
import oracle.dms.instrument.Noun;
import oracle.dms.instrument.PhaseEvent;
import oracle.dms.instrument.State;
import oracle.dms.instrument.Sensor;
```

B.2.2 パフォーマンス・データの編成

Sensor および各 Sensor に関連付けるメトリックを編成するには、最初に DMS Noun を定義します。DMS Noun では、ファイル・システムのディレクトリ構造と同様に、最上位のルート Noun の下にツリー階層で Sensor が編成されます。

例 B-2 は、BasicBinomial.java の Noun.create() を使用するコードのセクションを示したものです。

例 B-2 の MathSeries は、この Noun のタイプを示します。Noun のタイプには、収集対象となるメトリックのセットを表す名前を使用します。たとえば、MathSeries は、二項級数の計算を含むサンプル・アプリケーションについてメトリックを収集することを示します。AggreSpy では、同じ Noun タイプの Sensor が一緒に表示されます。

Sensor を直接含む Noun の Noun タイプのみを使用するようにしてください。Noun dmsDemo のように、Noun のみ含まれて Sensor が直接含まれない Noun の Noun タイプは、AggreSpy では、メトリックを含まないメトリック表として表示されます。例 B-2 は、Noun BasicBinomial を 1 つ含んで Sensor は含まない dmsDemo Noun を示しています。この Noun に対して Noun タイプが指定されない場合、AggreSpy では、この Noun に関連付けられたメトリック表が表示されません。

注意： Noun のタイプの名前は、他の DMS の名前と区別するために大文字で始めてください。

例 B-2 Noun.create を使用した Sensor の編成

```
private Noun binRoot;       // Container for Binomial series DMS metrics.
Noun base = Noun.create("/dmsDemo");
binRoot = Noun.create(base, "BasicBinomial", "MathSeries");
```

関連項目： 「DMS ネーミング規則」（B-6 ページ）

B.2.2.1 Noun のタイプの選択

通常、Noun のタイプは、その祖先 Noun または子孫 Noun とは異なるものにする必要があります。一般的に、これのコード化は容易で、さらには同じタイプの Noun が同じレベルにある論理階層を実現します。たとえば、dmsDemo アプリケーションには、BasicBinomial サーブレットと、それを拡張した 2 番目のサーブレットの ImprovedBinomial があります。この場合、インストルメントでは、両方に対して MathSeries タイプの Noun を使用します。この Noun は、両方のサーブレットにとって同じ階層レベルとなる /dmsDemo の下に作成されます。この規則を順守することで、生成されるメトリック表の理解がより容易になります。また、レポート処理時の情報の漏れが最小になります。

B.2.3 タイミングを計測するメトリックの定義と使用

コードのセグメントの期間を測定するメトリックを作成するには、次の手順で、PhaseEvent Sensor を定義および使用します。

- PhaseEvent Sensor の定義
- PhaseEvent Sensor の使用

B.2.3.1 PhaseEvent Sensor の定義

例 B-3 は、computeSeries という PhaseEvent Sensor を宣言して作成する DMS コールを示しています。このコードは、/dmsDemo/BasicBinomial/computeSeries.time という名前の DMS メトリックを定義します。

PhaseEvent Sensor は、デフォルトのメトリック .time（PhaseEvent start() コールから PhaseEvent stop() コールまでの合計時間を示す）とともに、オプションのメトリックのセットをサポートします。PhaseEvent Sensor のオプションのメトリックは、個別に、あるいは完全なメトリックのセットとして得ることができます。表 B-1 は、PhaseEvent Sensor で使用可能なメトリックを示しています。例 B-3 の binComp.deriveMetric(Sensor.all) コールによって、サポートされるすべてのオプションのメトリックが計算されレポートされるようになります。

注意： オプションのメトリックを追加する場合は、メソッド deriveMetric(Sensor.all) を使用することをお薦めします。Sensor.all とともにこのメソッドを使用すればすべてのメトリックを追加できます。オプションのメトリックの一覧は、今後リリースされる Oracle Application Server で変更される可能性があるので、これはよい方法です。また、このメトリックは、計算が効率的で、パフォーマンスの評価に役立ちます。

例 B-3 PhaseEvent Sensor の定義

```
private PhaseEvent binComp; // Time to compute Binomial series.
.
.
.
binComp = PhaseEvent.create(binRoot, "computeSeries",
                                     "Time to compute a Binomial series");
binComp.deriveMetric(Sensor.all);
```

B.2.3.2 PhaseEvent Sensor の使用

PhaseEvent Sensor を使用するには、アプリケーションで、start() メソッドをコールしてフェーズの開始を示し、続いて stop() メソッドをコールしてフェーズの完了を示します。

例 B-4 は、start() メソッドおよび stop() メソッドを使用して dmsDemo/BasicBinomial/computeSeries.time メトリックを計測する、BasicBinomial.java のコードのセグメントを示しています。PhaseEvent start() メソッドから返される token という名前の long 値は、対応する PhaseEvent stop() メソッドに渡す必要があります。この値は、開始時刻を表すタイム・スタンプです。この値を stop() メソッドに渡すことにより、DMS で PhaseEvent の継続時間を計算できます。

注意： PhaseEvent が停止したことを確認するには、例 B-4 に示すように、各 PhaseEvent start() メソッドを測定コードとともに try ブロックに置き、対応する finally ブロックに PhaseEvent stop() メソッドを置く必要があります。

例 B-4 PhaseEvent Sensor での start() および stop() の使用

```
long token = 0; // DMS
try {
    token = binComp.start(); // DMS
    BigInteger bins[] = bin(length);
    out.println("<H2>Binomial series for " +  length + "</H2>");
    for (int i = 0; i < length; i++)
        out.println("<br>" + bins[i]);
    }
    finally {
        binComp.stop(token); // DMS
        out.close();
    }
```

例 B-4 は、Phase が始まるたびに停止される、インストルメントされたコードを示します (finally 句に stop メソッドがあるため)。これは Phase Sensor の暴走を防止します。ただし、これによって、例外をスローする時間が必要となり、Phase 統計に影響する場合があります。例外処理が PhaseEvent に影響を与えるのを避けるには、例 B-5 に示すように、`abort()` メソッドを使用します。

例 B-5 は、停止されない Phase は中断されるサンプル・コードを示します。中断のコールが、該当の start と対応する統計を削除します。そしてこれらの統計は、メトリックの計算に影響しません。

例 B-5 PhaseEvent Sensor での abort() の使用

```
PhaseEvent pe = heavyPhase(param);
    long token1 = 0;
    long token2 = 0;
    boolean stopped = false;
    try {
      token1 = binComp.start();
      if (pe != null) token2 = pe.start();
      BigInteger bins[] = bin(length);
      out.println("<H2>ImprovedBinomial series for " +  length + "</H2>");
      for (int i = 0; i < length; i++)
        out.println("<br>" + bins[i]);
      if (pe != null) pe.stop(token2);
      binComp.stop(token1);
      stopped = true;
    }
    finally {
      if (!stopped) {
        if (pe != null) pe.abort(token2);
        binComp.abort(token1);
      }
```

B.2.4 カウントを計測するメトリックの定義と使用

イベントの発生をカウントするメトリックを作成するには、Event Sensor を次のように定義して使用します。

- Event Sensor の定義
- Event Sensor の使用

B.2.4.1 Event Sensor の定義

例 B-6 は、Event Sensor を定義する DMS コールを示しています。このコードは、カウンタを割り当て、`/dmsDemo/BasicBinomial/loops.count` という名前の DMS メトリックを定義します。

例 B-6 Event Sensor の定義

```
private Event binLoop;   // Loops needed for Binomial series.
.
.
.

binLoop = Event.create(binRoot, "loops", "Iterations to compute series");
```

B.2.4.2 Event Sensor の使用

アプリケーションで Event Sensor の `occurred()` メソッドがコールされるたびに、DMS はカウンタを増加させます。例 B-7 は、`/dmsDemo/BasicBinomial/loops.count` メトリックを増加する、Event Sensor の `occurred()` コールを示しています。

例 B-7 Event Sensor での occurred() の使用

```
binLoop.occurred();
```

B.2.5 状態情報を記録するメトリックの定義と使用（State Sensor）

DMS は、State Sensor を使用して状態情報を取得します。State Sensor は、Java プリミティブの値または Java オブジェクトのコンテンツを追跡します。`create()` メソッドの 3 番目の引数として指定しているように、integer、double、long および object などの型がサポートされています。Java プリミティブの State Sensor が不正な型で更新された場合、DMS は指定された値を正しい型に変換しようとします。オブジェクト型の State Sensor の場合、DMS はオブジェクトへの参照を格納し、DMS 値をサンプリングする際に、そのオブジェクトに対して `toString()` をコールします。

状態情報を記録するメトリックを作成するには、State Sensor を次のように定義して使用します。

- State Sensor の定義
- State Sensor の使用

B.2.5.1 State Sensor の定義

State Sensor は、デフォルトのメトリック `value` およびオプションのメトリックをサポートします。`minValue` および `maxValue` のオプション・メトリックは、State Sensor に対し、State Sensor が（integer、double、および long などの）数値の Java プリミティブを表す場合にのみ定義できます。表 B-3 は、State Sensor で使用可能なメトリックを示しています。例 B-3 は、オプションのメトリックを有効にする方法を示しています。

例 B-8 は、State Sensor を宣言して作成する DMS コールを示しています。このコードは、`/dmsDemo/BasicBinomial/lastComputed.value` という名前の DMS メトリックを定義します。

例 B-8 State Sensor の定義

```
private State binLast;   // Value of the last computed element in series.
.
.
.
binLast = State.create(binRoot, "lastComputed", State.OBJECT, "",
                           "Value of last computed series element");
```

State Sensor を定義するとき、State Sensor に単位が関連付けられていない場合、`create()` メソッドの 4 番目の引数に空の文字列を使用します。または、適切な単位の文字列を使用します（例 B-8 を参照）。State Sensor は、初期値なしで作成されます。State Sensor が初期化されたかどうかを確認したい場合は、`isInitialized()` メソッドを使用します。

オブジェクトへの参照ではなく、オブジェクトの文字列の値を State Sensor が格納するようにするには、値を TRUE とし、setCopy() メソッドを使用します。これによって、State Sensor は、メトリック値としてオブジェクトへの参照を使用するのではなく、toString() をコールした結果をオブジェクトに格納します。

B.2.5.2 State Sensor の使用

アプリケーションで State Sensor の update() メソッドがコールされると、DMS は State Sensor の値を更新します。例 B-9 は、/dmsDemo/BasicBinomial/lastComputed.value メトリックを更新する、State Sensor の update() コールを示しています。

例 B-9 State Sensor での update() の使用

```
binLast.update(bins[k-1].toString());
```

B.3 DMS メトリックを使用したアプリケーションの検証とテスト

Java アプリケーションに追加するメトリックは、その精度をテストして検証する必要があります。

この項には、次の項目が含まれています。

- DMS メトリックの検証
- DMS メトリックの効率性のテスト

B.3.1 DMS メトリックの検証

新しいメトリックを検証してテストするには、dmstool およびその他の使用可能な DMS 監視ツールを使用します。

新しいメトリックについて、次のことを検証してください。

- 予期されるメトリックがディスプレイに表示されるかどうか。これをテストするには、コードを調べて、DMS インストルメントによって追加されたすべてのメトリックの名前が、ディスプレイあるいは保存されたメトリックのセットに表示されていることを確認します。
- 予期しないメトリックがディスプレイに表示されないかどうか。追加対象のメトリックのみが追加されていることを検証します。
- 表示されるメトリック値が有効な範囲内にあるかどうか。通常、メトリックには上限および下限を設定できます。メトリックの範囲を設定した場合は、レポートされるメトリック値をテストして、その範囲を超えないことを確認します。

  たとえば、プール・サイズを計測するメトリックでは、負の値がレポートされることはありません。
- 新しいメトリックが必要であることを確認します。たとえば、非常に短い期間のイベントを常に測定する PhaseEvent を追加する場合は、メトリックの Event メトリックへの変更、またはメトリックの削除を検討します。
- 新しいメトリックが正確であることを確認します。DMS メトリックを使用するほとんどのアプリケーションでは、DMS インストルメントを追加するパフォーマンス・コストよりも正確さが重要です。新しい DMS メトリックは、信頼できる有用な情報を提供する必要があります。

  正確性のテストは簡単ではありません。ただし、特定のメトリックを測定する別の方法がある場合は、それを使用してメトリック値を検証します。たとえば、既知の数のリクエストをサーバーへ送信して処理にかかる合計時間を測定する場合、関連するメトリックの適切な値を予測し、それらを実際に監視された値と比較します。また、ログ・ファイルまたはコンソールに書き込まれたレコードを調べれば、Event Sensor の count メトリックを検証できます。

メトリックに適用されるタイミングの不正確さをチェックします。タイミングの不正確さは、低分解能のクロックを使用して、継続期間の短い時間隔のメトリックを計測した場合に発生する可能性があります。たとえば、Windows システムでは、デフォルトの Java のクロックは 15 ミリ秒ごとに 1 回早まります。これらのシステムで短時間のイベントについてレポートされる DMS メトリックは、注意深く分析する必要があります。この問題を解決するには、高分解能のクロックの使用を検討してください。

関連項目： 「DMS の精度を上げるための高分解能クロックの使用」（B-17 ページ）

B.3.2 DMS メトリックの効率性のテスト

DMS メトリックの使用は、アプリケーションのパフォーマンスに少なからず影響を与えます。メトリックを追加する場合は、次のことに注意してください。

- メトリックを計算して格納するために必要な処理は、アプリケーションの実行速度を低下させます。DMS は高速ですが、かなりのオーバーヘッド・コストを必要とします。さらに、DMS では、開発者による DMS API の非効率な使用を回避できません。そのため、DMS インストルメントを追加する場合は、事前に適切な予測を立てる必要があります。実装の完了後、実際のコストを測定して予測と比較します。実際の測定値が予測と一致するまで、インストルメントを変更してオーバーヘッド・コストを軽減してください。
- DMS には、使用するメトリックを制御できる `DMSConsole.getSensorWeight()` メソッドが用意されています。このメソッドの中心的な設定は推奨される測定レベルであり、DMS で規定されるものではありません。含めるメトリックを制御するには、実行時、コード内で `SensorWeight` の値をテストして、DMS コールを実行するかどうかを判断する必要があります。
- DMS インストルメントを既存のパッケージに統合したり新しい機能を実装する場合は、以前に動作していたシステムを切り離すことを検討してください。たとえば、新しい DMS メトリックを有効あるいは無効にするオプションが含まれる可能性があります。
- パフォーマンスのみに気をとられると、設計や実装のエラーに膨大なコストがかかる場合があります。Donald Knuth によれば、「早まった最適化はすべての悪の根源である」そうです。
- DMS を有効あるいは無効にしてパフォーマンス・テストを実行します。DMS を有効にして行ったテスト結果が受け入れられないものであれば、メトリックをもう一度設計して、実装しなおしてください。

B.4 DMS のセキュリティの考慮事項

DMS メトリックは、DMS レポートへのユーザー・ベースのアクセスをサポートしません。DMS メトリックを定義して使用する場合、DMS メトリックにアクセスできる管理者であれば誰でもそのメトリックを使用できます。DMS メトリックを追加する場合は、顧客の機密情報をメトリックに配置しないよう注意してください。

DMS インストルメントを追加する場合、その DMS メトリックにアクセスできるユーザーは次のとおりです。

- 同じ OC4J インスタンスで実行されているアプリケーション。
- `dmstool` コマンドまたは `AggreSpy` サーブレットにアクセスできるすべてのユーザーは、メトリックにアクセスできます（デフォルトではこれは管理者に限定されています）。

関連項目：

- 「Web サーバーでの AggreSpy の URL とアクセス制御」（A-7 ページ）
- 「dmstool のアクセス制御」（A-9 ページ）

B.5 DMS Sensor Weight を使用した条件付きインストルメント

条件付きでインストルメントを制限するには、DMS Sensor Weight 機能を使用します。Sensor Weight 機能を使用すると、Sensor Weight に特定の値が設定されている場合のみ、負荷のかかるインストルメントを実行するようにアプリケーションを指定できます。この機能を使用すれば、デバッグのみに必要とされるような、負荷のかかるメトリックの収集を制御することができます。

例 B-10 は、`DMSConsole.getSensorWeight()` を使用して Sensor Weight の値をテストし、その結果によってメトリックを定義および使用する方法を示しています。

Sensor Weight は、コマンドラインで `oracle.dms.sensors` プロパティを使用してグローバルに設定します。このプロパティは、OC4J スタートアップ・オプションを使用して設定します。このプロパティでサポートされる値には、`none`、`normal`、`heavy` および `all` があります。

例 B-10 Sensor Weight を使用した条件付きインストルメント

```
 /* DMS Method
    *
    * If the SensorWeight is high enough, return a phase with the
    * parameter in the name. Otherwise, return null.
    */
PhaseEvent heavyPhase(String param) {
PhaseEvent pe = null;
if (DMSConsole.getSensorWeight() > DMSConsole.NORMAL) {
   Noun base = Noun.create(binRoot, param, "MathSeries");
   pe = PhaseEvent.create(base, "computeSeries",
                                  "Time to compute a Binomial series");
   pe.deriveMetric(Sensor.all);
   }
return pe;
```

B.6 DMS メトリックのファイルへのダンプ

Java アプリケーションでは、次のメソッドを使用して DMS メトリックをファイルへダンプします。

次のコードを使用すると、指定したファイルに現行のメトリックを追加したり、そのファイルの内容を現行のメトリックで置き換えることができます。

```
DMSConsole cons2 = new DMSConsole();
DMSConsole.dump("dmsmathseries.log", true, true);
```

最初の引数にはファイルのパス名、2 番目の引数には出力形式を指定します。3 番目の引数には、この出力をファイルに追加するか、あるいは出力によってファイルの内容を置き換えるかを指定します。

B.7 Sensor のリセットと破棄

Sensor の抽象クラスは、PhaseEvent、Event および State Sensor を制御するためのメソッドを提供します。`reset()` メソッドは、Sensor のメトリックを初期値にリセットします。`getResetTime()` メソッドは、Sensor がリセットされたかどうか判断します。`destroy()` メソッドは、Sensor を DMS から削除して、その基礎となるリソースへの参照を解放します。

注意： このメソッドを、組込みメトリックをリセットまたは破棄するために使用しないでください。`reset()` および `destroy()` メソッドは、作成したメトリックで使用するためにあります。これらのメソッドを、内部の組込みメトリックで使用すると、Application Server Control コンソールおよび他の Oracle Application Server 管理機能が、予期しない値をレポートしたり、予期しない動作をすることがあります。

B.8 DMS コーディングの推奨事項

DMS でのコーディングの推奨事項を次に示します。

1. DMS メトリックにはグローバルな名前領域があります。新しい Noun Sensor（PhaseEvent、Event あるいは State）を作成する場合は、そのフルネームが、Oracle の組込みメトリックまたはその他のアプリケーションで使用される名前と競合しないようにします。そのため、アプリケーションのフルネームを含むアプリケーションのルート Noun を用意することをお薦めします。これによって、名前領域の競合を回避できます。

   関連項目： 「一般的な DMS のネーミング規則」（B-6 ページ）

2. すべての PhaseEvent が停止していることを確認します。測定対象のコード・ブロックが `try` ブロック内にない場合は、これを PhaseEvent の `start()` を含む `try` ブロックに置きます。また、PhaseEvent の `stop()` は `finally` ブロックに置きます。または、例 B-5 のように、`finally` ブロックの `abort()` メソッドを使用します。

   関連項目： 「PhaseEvent Sensor の使用」（B-10 ページ）

3. DMS ネーミング規則を使用します。

   関連項目： 「DMS ネーミング規則」（B-6 ページ）

4. DMS Sensor や Noun は 2 つ以上作成しないでください。DMS API を複数作成して、オブジェクトは複数作成しないようにしてください。DMS では、後にこれらを作成しようとすると、参照が実行されます。つまり、可能なかぎり、静的初期化によって Sensor と Noun を定義するか、サーブレットの場合は `init()` メソッドで定義する必要があります。

5. Sensor を含むそれぞれの Noun にタイプを割り当てます。タイプが割り当てられない場合は、値「n/a」（使用不可）が割り当てられます。タイプに「n/a」が指定された Noun は、`AggreSpy` では表示されません。

6. PhaseEvent は、特定の状況において実行にかなりの時間がかかり、負荷のかかるコードのセクションを測定する場合にのみ使用します。コードの実行に著しく時間がかかることがない場合、Event メトリックを使用するか、PhaseEvent を削除します。

7. DMS API のコールはスレッドセーフであり、競合やアクセスの不具合を回避するための十分な同期を提供します。

B.8.1 PhaseEvent メトリックを使用した、過負荷な時間隔の分離

DMS インストルメントを追加する場合、新しいメトリックの要件は慎重に考慮してください。作成したコードが予測したとおりに動作していることを検証するには、十分な数のメトリックを追加することが重要です。

DMS メトリックを追加する場合、次のガイドラインに注意してください。

1. 目的のコード・ブロックまたはモジュールについて、システムでの処理に要した時間の概要のみを提供する PhaseEvent Sensor を追加します。すべてのメソッド・コール、または検証の対象となるコードやモジュールのすべての個別フェーズについてパフォーマンス・データを収集する必要はありません。

2. 目的のコード内から制御対象外の外部コードをコールし、その外部コードの処理に長時間かかることが予測される場合は、PhaseEvent Sensor を追加して、その外部コードの開始と完了を追跡します。

これらのガイドラインに従って、PhaseEvent メトリックを追加すると、次のメリットがあります。

- DMS で収集する情報量を制限できます。
- システムの分析担当者は、モジュールが期待されるランタイム・パフォーマンスを実現しているかどうかを検証できます。
- DMS メトリックの参照ユーザーは、膨大な量のデータを表示せずにランタイム・パフォーマンスを検証できます。
- システム・パフォーマンスの分析担当者は、目的のモジュールを追跡して、負荷の大きい、あるいは失敗回数の多い他のシステム・モジュールから分離することができます。

B.9 DMS の精度を上げるための高分解能クロックの使用

PhaseEvent の間、時間間隔を測定するために、DMS はデフォルトでシステム・クロックを使用します。デフォルトのクロックは、Apache などの C プロセスではマイクロ秒の単位で、OC4J などの Java プロセスではミリ秒の単位で精度をレポートします。パフォーマンス測定の精度向上のために、DMS は、オプションで高分解能クロックをサポートしています。これにより、レポートする時間間隔の単位を選択できます。デフォルトのクロックを使用する場合よりも正確に PhaseEvent を測る必要があるとき、またはシステムのデフォルトのクロックが分解能の要件に満たないとき、高分解能クロックを使用できます。

注意： デフォルトのクロックおよび高分解能クロックの分解能は、システムに依存します。システムによっては、デフォルトのクロックが提供する分解能では、タイミングの要件を満たさない場合があります。特に Windows プラットフォームでは、デフォルトのクロックでは 15 ミリ秒ごとに 1 度しか進みません。そのため、それよりも高い精度を要求するユーザーが多くいます。これらのシステムで短時間のイベントについてレポートされる DMS メトリックは、注意深く分析する必要があります。この問題を解決するには、高分解能のクロックの使用を検討してください。

この項には、次の項目が含まれています。

- OC4J（Java）での時間のレポートのための DMS クロックの構成
- Oracle HTTP Server での時間のレポートのための DMS クロックの構成

B.9.1 OC4J（Java）での時間のレポートのための DMS クロックの構成

Java プロセスでは、デフォルトのクロックは、`java.lang.System.currentTimeMillis()` を使用します。高分解能クロックを選択すると、クロックが変更されたプロセスで実行されているすべてのアプリケーションに対する、このコールが変更されます。プロセスのスタートアップ・オプションを制御する `oracle.dms.clock` および `oracle.dms.clock.units` プロパティを使用し、グローバルに DMS クロックとレポートの単位を設定します。

たとえば、デフォルトの単位で高分解能クロックを使用するには、OC4J の Java コマンドラインで、次のプロパティを設定します。

```
-Doracle.dms.clock=highres
```

注意： 高分解能クロックを使用しているとき、Application Server Control コンソールが予期している値（ミリ秒）とデフォルトの単位は異なります。高分解能クロックを使用しているときに Application Server Control コンソールの表示を適切にする必要がある場合は、単位のプロパティを次のように設定します。

```
-Doracle.dms.clock.units=msecs
```

表 B-6 は、`oracle.dms.clock` プロパティでサポートされる値を示しています。

表 B-7 は、`oracle.dms.clock.units` プロパティでサポートされる値を示しています。

表 B-6 oracle.dms.clock のプロパティ値

値	説明
DEFAULT	DMS でデフォルトのクロックを使用することを指定します。デフォルトのクロックでは、DMS は PhaseEvent で時間を取得するために、Java コール `java.lang.System.currentTimeMillis()` を使用します。 デフォルトのクロックの単位のデフォルト値は、MSECS です。
HIGHRES	DMS で高分解能クロックを使用することを指定します。DMS は JNI（JNI コールは、オペレーティング・システムで使用可能なクロックに依存します）を使用し、高分解能クロックにアクセスします。 HIGHRES クロックの単位のデフォルト値は、NSECS です。

注意： Pentium プロセッサを搭載した Windows プラットフォームでは、高分解能クロック（HIGHRES）にタイミングを提供するために、DMS は `QueryPerformanceCounter` 関数を使用します。Pentium プロセッサを搭載していない Windows プラットフォームで実行する場合は、高分解能クロックにタイミングを提供するために、DMS は DMS C クロックを使用します。DMS C クロックにはマイクロ秒の精度があり、`System.currentTimeMillis()` で使用可能なデフォルトのクロックより、精度はかなり向上します。

表 B-7 oracle.dms.clock.units のプロパティ値

値	説明
MSECS	時間がミリ秒に変換され、「msecs」でレポートされることを指定します。 **注意**: これがデフォルトのクロックのデフォルト値です。
NSECS	時間がナノ秒に変換され、「nsecs」でレポートされることを指定します。 **注意**: これが高分解能クロックのデフォルト値です。
USECS	時間がマイクロ秒に変換され、「usecs」でレポートされることを指定します。

高分解能 DMS クロックを使用するときは、次に注意してください。

- `oracle.dms.clock` および `oracle.dms.clock.units` プロパティを設定する場合、選択した値に大文字と小文字の任意の組合せを使用できます（大小文字の区別は重要ではありません）。たとえば、高分解能クロックを選択する場合、highres でも、HIGHRES でも、HighRes でも有効です。
- DMS はスタートアップ時にプロパティ値をチェックします。クロックに対し、表 B-6 にある値と一致しないものを設定した場合、DMS はデフォルトのクロックを使用します。DMS は、`oracle.dms.clock` プロパティが設定されていないときも、デフォルトのクロックを使用します。

- 指定したクロック単位のプロパティ値が表 B-7 にある値と一致しない場合、指定したクロックに対し、DMS はデフォルトの単位を使用します。DMS は、oracle.dms.clock.units プロパティが設定されていないとき、指定したクロックに対しデフォルトの単位を使用します。

表 B-8 は、サポートされている各プラットフォームに固有の環境変数の設定を示します。高分解能 DMS クロックを使用するには、環境変数を適切に設定する必要があります。高分解能クロックは、DMS C ライブラリを使用します。UNIX システムでは、指定する環境変数のパスに libdms2.so を含める必要があります。Windows システムでは、PATH 環境変数に yod.dll を含める必要があります。ナノ秒のクロックが使用不可能な場合には、高分解能のタイミングでミリ秒のクロックを使用します。

表 B-8 サポートされているプラットフォームのライブラリ・パスの環境変数

プラットフォーム	環境変数
AIX	LIBPATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
	LD_LIBRARY_PATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
HP-UX	SHLIB_PATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
	LD_LIBRARY_PATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
Linux	LD_LIBRARY_PATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
Tru64 UNIX	LD_LIBRARY_PATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
Solaris	LD_LIBRARY_PATH
	パスに $ORACLE_HOME/lib/libdms2.so を含める必要があります。
Windows 2000	パスに %ORACLE_HOME%¥Apache¥Apache¥yod.dll を含める必要があります。
Windows 2003	パスに %ORACLE_HOME%¥Apache¥Apache¥yod.dll を含める必要があります。
Windows XP	パスに %ORACLE_HOME%¥Apache¥Apache¥yod.dll を含める必要があります。

B.9.2 Oracle HTTP Server での時間のレポートのための DMS クロックの構成

Oracle HTTP Server のパフォーマンスを測定するデフォルトのクロックの分解能はマイクロ秒（usecs）です。Oracle HTTP Server で実行されている C プロセスの監視には、オプションで、より高分解能のクロックを選択できます。Oracle HTTP Server で高分解能クロックを使用するには、httpd.conf で構成オプションを設定するか、コマンドラインで環境変数を指定する必要があります。

表 B-9 は、Oracle HTTP Server の DMS クロックを制御する環境変数を示します。表 B-10 では、Oracle HTTP Server の DMS クロックを制御する httpd.conf の構成オプションを示します。コマンドライン・オプションと httpd.conf 構成オプションの両方を設定した場合、コマンドラインよりも構成オプションで設定した値が優先されます。

表 B-9 OHS の DMS クロック環境変数

環境変数	説明
DMS_CLOCK	DMS タイミングに使用するクロックを指定します。値は oracle.dms.clock と同じように解釈されます。 有効値 : DEFAULT、HIGHRES
DMS_CLOCK_UNITS	DMS タイミング値をレポートする単位を指定します。値は oracle.dms.clock.units と同じように解釈されます。 有効値 : MSECS、NSECS、USECS デフォルト値 : USECS

表 B-10 OHS の DMS クロック構成パラメータ

パラメータ	説明
`DmsClock`	Java プロセスに対する oracle.dms.clock プロパティのように、OHS によって開始される HTTP リスナー・プロセスに対するクロックを指定します。 有効値 : DEFAULT、HIGHRES
`DmsClockUnits`	Java プロセスに対する oracle.dms.clock.units プロパティとまったく同様に、OHS によって開始される HTTP リスナー・プロセスに対する時間単位を指定します。 有効値 : MSECS、NSECS、USECS デフォルト値 : USECS

注意 : Pentium プロセッサを搭載した Windows プラットフォームでは、高分解能クロック（HIGHRES）にタイミングを提供するために、DMS は `QueryPerformanceCounter` 関数を使用します。Pentium プロセッサを搭載していない Windows プラットフォームで実行する場合は、高分解能クロックにタイミングを提供するために、DMS は DMS C クロックを使用します。DMS C クロックにはマイクロ秒の精度があり、`System.currentTimeMillis()` で使用可能なデフォルトのクロックより、精度はかなり向上します。

たとえば、高分解能クロックを使用して、OC4J で実行されている Java プロセスと Oracle HTTP Server で実行されている mod_oc4j の時間を表示するために同じ単位を使用する場合、次のパラメータおよび値が含まれるよう、Oracle HTTP Server の httpd.conf ファイルを更新します。

```
DmsClock=HIGHRES
DmsClockUnits=MSECS
```

また、OC4J プロセスのスタートアップ・オプションとして次の値も含めます。

```
-Doracle.dms.clock=HIGHRES
-Doracle.dms.clock.units=MSECS
```

これらのオプションを使用して、DMS は、監視するすべての Oracle HTTP Server プロセスおよび Java OC4J プロセスに対して高分解能クロックを使用します。そして、DMS はミリ秒（msecs）単位で値をレポートします。

注意： Oracle HTTP Server で高分解能クロックを使用するとき、多くのプラットフォームで、デフォルトの単位は NSECS です。Application Server Control コンソールを使用する必要がある場合、単位は USECS が要求されます。高分解能クロックを使用しているときに Application Server Control コンソールの表示を適切にする必要がある場合は、単位のプロパティを次のように設定します。

```
DmsClock=HIGHRES
DmsClockUnits=USECS
```

B.10 子孫 Noun の DMS データのロールアップ

Oracle Application Server 10*g* リリース 3（10.1.3.1.0）には、メトリックの集計を指定できる DMS ロールアップ機能が用意されています。このロールアップ機能を使用することで、DMS のインストルメント時にメトリックの集計を指定できます。ロールアップの適用は、特定の Noun タイプの子孫に対して指定します。ロールアップは、直接の子孫のみに適用することも、すべての子孫に適用することもできます。例 B-11 は、図 B-2 に示す DMS ツリーを生成するコードを示しています。`myContainer` タイプの各 Noun には、`percentageFull`、`close` および open Sensor が含まれます（図 B-3 を参照）。

注意： 例 B-11 のコードでは、B-9 ページの「Noun のタイプの選択」に説明されている推奨事項に違反する Noun のツリー階層が生成されます。この例では、一部の Noun が同じタイプの子孫および祖先 Noun を持つことが道理にかなっています。この項で説明するロールアップ機能では、他の方法では失われる可能性があるデータを収集できます。

例 B-11 メトリックの Noun 階層を作成する DMS サンプル・コード

```
// Create DMS Noun hierarchy for metrics.
Noun home = Noun.create(Noun.getRoot(), "Home", "myContainer");
Noun containers = Noun.create(home, "Containers", "myContainer");
Noun closets = Noun.create(containers, "Closets", "myContainer");
Noun bedrooms = Noun.create(closets, "Bedrooms", "myContainer");
Noun br1 = Noun.create(bedrooms, "BR1", "myContainer");

// Create a closet Noun and create Sensors for it.
Noun c1 = Noun.create(br1, "C1", "myContainer");
State percent = State.create(br1, "percentageFull", State.INTEGER, "percent",
"percentage full");
Event close = Event.create(br1, "close", "container closed");
PhaseEvent open = PhaseEvent.create(br1, "open", "open container");

// Derive metrics for State and PhaseEvent Sensors
percent.deriveMetric(Sensor.all);
open.deriveMetric(Sensor.all);
```

図 B-2 ツリー内にメトリックがある DMS コンテナ階層

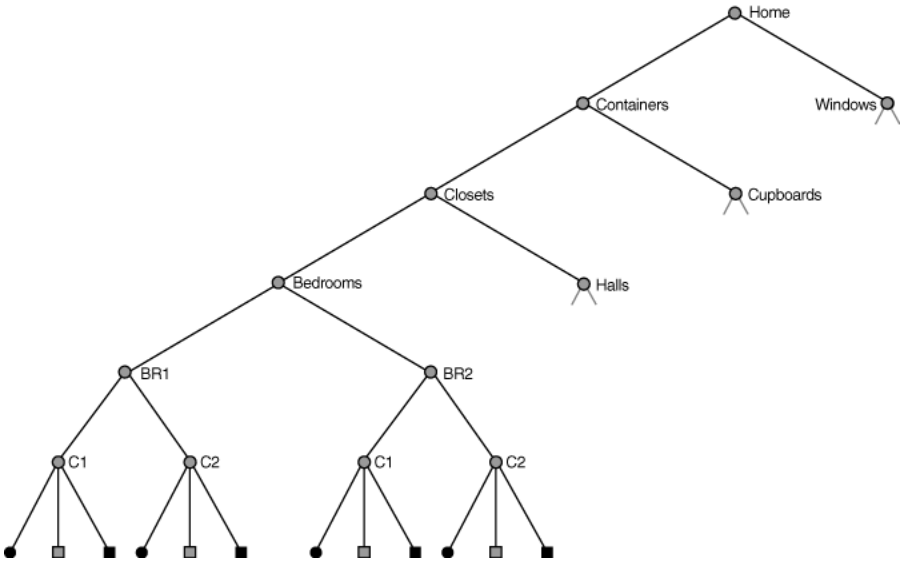
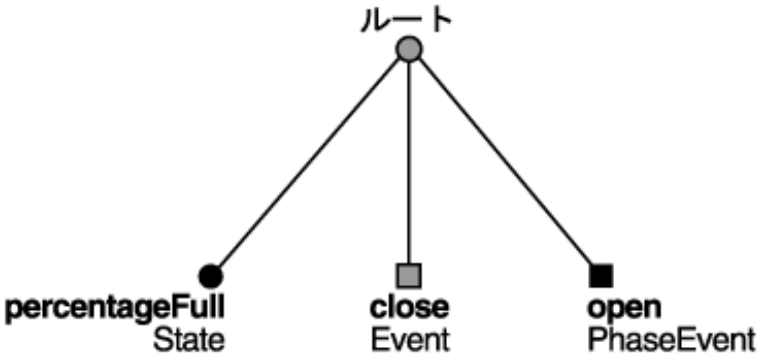


図 B-2 は、子孫コンテナを持つツリーを示しています。Bedrooms BR1 と BR2 の下にある C1 および C2 Noun は、myContainer タイプです（myContainer メトリックの説明は図 B-3 を参照）。

図 B-3 サンプル Sensor を示す Noun myContainer



DMS では、ロールアップ機能を使用して、子孫 Noun のサマリーを集計できます。たとえば、Bedrooms Noun に対して、例 B-11 に示されているロールアップ・コールを追加できます。BR1 下にある myContainer タイプ・メトリックを集計するには、次のコールを使用します。

```
br1.rollup("myContainer", Noun.DIRECT);
```

このコールによって、/Home/Containers/Closets/Bedrooms/BR1 の下に myContainer_rollup という名前のロールアップ Noun が作成されます。ロールアップ Noun には、percentageFull、close、open などの、関連付けられている Noun と同じ Sensor が含まれます。

DMS ロールアップ・メトリックでは、特定タイプのすべての子孫 Noun 内にある Sensor をロールアップすることも、直接の子孫 Noun 内にある Sensor のみをロールアップすることもできます。ロールアップ・コールに Noun.DIRECT を指定すると、指定されたタイプの直接の子孫 Noun のみが集計されます。myContainer タイプのすべての子孫 Noun からのメトリックを集計するには、次に示すように Noun.ALL を使用してコールします。

```
closets.rollup("myContainer", Noun.ALL);
```

ロールアップ・メトリックには、その内容を集計するサマリー情報があります。表 B-11 に、各 Sensor のタイプで使用できる派生のロールアップ・メトリックを示します。

表 B-11 ロールアップ・メトリックとその派生メトリック

メトリック	説明
PhaseEvent	PhaseEvent ロールアップ・メトリックの派生メトリックには、次のものがあります。 ■ time: time メトリックの合計値 ■ completed: completed メトリックの合計値 ■ maxTime: maxTime メトリックの最大値 ■ minTime: minTime メトリックの最小値 ■ avg: すべての Sensor に対して計算された平均時間 ■ active: active メトリックの合計値
Event	Event ロールアップ・メトリックの派生メトリックには、次のものがあります。 ■ sum: すべての count メトリックの合計値 ■ avg: すべての count メトリックの平均値
State	State ロールアップ・メトリックの派生メトリックには、次のものがあります。 ■ sum: すべての value メトリックの合計値 ■ avg: すべての value メトリックの平均値 ■ maxValue: maxValue メトリックの最大値 ■ minValue: minValue メトリックの最小値
descendents	ロールアップ Noun には、ロールアップの対象が直接の子孫のみかすべての子孫かを報告する descendents State Sensor があります。
rolled	ロールアップ Noun には、ロールアップされている Noun の数を報告する rolled State Sensor があります。
refresh	ロールアップ Noun には、このロールアップ Noun のメトリックの集計に要した時間を報告する refresh PhaseEvent があります。

例 B-12 に、/Home/Containers/Closets 下の myContainer ロールアップ Noun に対して作成されたサンプル・メトリックを示します。

例 B-12 テスト

```
myContainer_rollup
  descendent.value: all
  percentageFull.sum 40 percent
  percentageFull.avg 10.0 percent
  percentageFull.min 1 percent
  percentageFull.max 29 percent
  close.sum: 3
  close.avg: 0.75
  open.time: 871 msecs
  open.completed: 4 ops
  open.maxTime: 722 msecs
```

```
open.minTime: 23 msecs
open.avg: 217.7 msecs
open.active: 0
rolled.value: 4 nouns
refresh.maxActive:   1 threads
refresh.active:      0 threads
refresh.avg: 0.2857142857142857 msecs
refresh.maxTime:     1 msecs
refresh.minTime:     0 msecs
refresh.completed:   7 ops
refresh.time:        2 msecs
```

このメトリックは myContainer メトリックと類似していることに注意してください。ロールアップ・メトリックの場合、主に次のような違いがあります。

1. ロールアップ Noun には、descendent、rolled および refresh メトリックがあります(詳細は表 B-11 を参照)。
2. percentageFull State には、value メトリックではなく、sum および avg メトリックがあります。各メトリックの名前は、その内容を示しています。
3. close Event には、count メトリックではなく、sum および avg メトリックが含まれます。各メトリックの名前は、その内容を示しています。
4. open PhaseEvent には、maxActive メトリックは含まれません。このコンテキストでは意味を持たないためです。

関連項目：『Oracle Application Server DMS API Reference』の Javadoc

C

パフォーマンス・メトリック

この付録には、Oracle Application Server のパフォーマンスを分析するときに役立つ組込みメトリックの一覧が記載されています。これらのメトリックは、Oracle HTTP Server や Oracle Containers for J2EE（OC4J）などのいくつかの領域に明確に分類されています。この章の各表には、対応するダイナミック・モニタリング・サービス（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。

C.1 Oracle HTTP Server メトリック

表 C-1 から表 C-5 までの各表は、Oracle HTTP Server メトリックを説明したものです。

表 C-1 は、HTTP サーバーの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 C-1 HTTP Server メトリック（ohs_server）

メトリック	説明	単位
busyChildren.value	ビジーな子プロセスの数	プロセス
childFinish.count	終了した子プロセスの数	プロセス
childStart.count	開始した子プロセスの数	プロセス
connection.active	現在開いている接続の数	スレッド
connection.avg	HTTP 接続のサービスに費やされた平均時間	usecs
connection.completed	HTTP 接続が確立された回数	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.time	HTTP 接続のサービスに費やされた合計時間	usecs
error.count	HTTP エラーの数	件数
get.count	GET リクエストの数	件数
handle.active	現在ハンドル処理フェーズにある子サーバー	スレッド
handle.avg	モジュール・ハンドラで費やされた平均時間	usecs
handle.completed	ハンドル処理フェーズが完了した回数	ops
handle.maxTime	モジュール・ハンドラで費やされた最大時間	usecs
handle.minTime	モジュール・ハンドラで費やされた最小時間	usecs
handle.time	モジュール・ハンドラで費やされた合計時間	usecs
internalRedirect.count	モジュールが新しい内部 URI にリクエストをリダイレクトした回数	ops
lastConfigChange.value	構成が最後に変更された日時	時間
numChildren.value	リクエスト処理のプロセスの合計数	プロセス
numMods.value	ロードされたモジュールの数	ops
post.count	POST リクエストの数	ops
readyChildren.value	リクエストの処理準備ができたプロセスの数	プロセス
request.active	現在リクエスト処理フェーズにある子サーバー	スレッド
request.avg	HTTP リクエストのサービスに必要な平均時間	usecs
request.completed	完了した HTTP リクエストの数	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.time	HTTP リクエストのサービスに必要な合計時間	usecs
responseSize.value	レスポンスのサイズ	バイト

C.1.1 Oracle HTTP Server 子サーバー・メトリック

表 C-2 は、子サーバーの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_child です。

表 C-2 Oracle HTTP Server 子サーバー・メトリック（ohs_child）

メトリック	説明	単位
pid.value	子のプロセス識別子	
slot.value	子のスロット識別子	
status.value	子の現行のステータス	
time.value	この子が最新リクエストの処理に要した時間	
url.value	最新リクエストの URL	

C.1.2 Oracle HTTP Server レスポンス・メトリック

Oracle HTTP Server レスポンス・メトリックは、ohs_responses というタイプのメトリック表に含まれます。このメトリック表には、各 HTTP レスポンス・タイプについて、件数、つまりレスポンスが生成された回数を示すメトリックが 1 つ含まれます。

例：Success_OK_200.count: 28 ops

C.1.3 Oracle HTTP Server 仮想ホスト・メトリック

表 C-3 は、Oracle HTTP Server 仮想ホスト・メトリックを示します。

メトリック表のタイプは ohs_vhostSet です。

メトリック表のタイプ ohs_virtualHost には、仮想ホストの名前と位置、およびリクエストやレスポンスのメトリックに関する情報が含まれます。

表 C-3 Oracle HTTP Server 仮想ホスト・メトリック（ohs_virtualHost）

メトリック	説明	単位
request.active	このホストによって現在処理されているリクエストの数	スレッド
request.avg	この仮想ホストに対するリクエストの処理に要した平均時間	usecs
request.completed	この仮想ホストによって処理されたリクエスト数	ops
request.maxTime	この仮想ホストに対する 1 つの任意のリクエストの処理に要した最大時間	usecs
request.minTime	この仮想ホストに対する 1 つの任意のリクエストの処理に要した最小時間	usecs
request.time	この仮想ホストに対するリクエストの処理に要した合計時間	usecs
responseSize.value	レスポンスのサイズ	バイト
vhostType.value	仮想ホストのタイプ	

C.1.4 集計モジュール・メトリック

表 C-4 は、Oracle HTTP Server のモジュール・メトリックを示しています。

メトリック表のタイプは ohs_module です。

表 C-4 Oracle HTTP Server モジュール・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	

C.1.5 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 C-5 Oracle HTTP Server 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.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	リクエスト
handle.avg	このモジュールに必要な平均時間	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
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.time	このモジュールに必要な合計時間	usecs

C.1.6 Oracle HTTP Server mod_oc4j メトリック

表 C-6 は、mod_oc4j 障害原因メトリックを示しています。この表は、クライアントに INTERNAL_SERVER_ERROR を返すエラーのカテゴリを示します。

メトリック表の名前は mod_oc4j_request_failure_causes です。

表 C-6 HTTP Server mod_oc4j リクエスト障害原因メトリック

メトリック	説明	単位
IncorrectReqInit.count	内部エラーが発生した合計回数。mod_oc4j が接続エンドポイントを見つけられない、構成のエラーなど、様々な理由が考えられます。	ops
Oc4jUnavailable.count	リクエストのサービスを行う oc4j JVM が見つからなかった合計回数	ops
UnableToHandleReq.count	mod_oc4j がリクエストの処理を拒否した合計回数	ops

表 C-7 は、mod_oc4j マウント・ポイント・メトリックを示しています。mod_oc4j.conf で指定されているマウント・ポイントごとに、マウント・ポイント・メトリック表が 1 つあります。この表には、指定された各マウント・ポイントのメトリックのセットが含まれます。各セットは、mntPt*id* でグループ化されています。*id* は、モジュールの初期化時に自動的に生成される整数です。

メトリック表の名前は mod_oc4j_mount_pt_metrics です。

表 C-7 HTTP Server mod_oc4j マウント・ポイント・メトリック

メトリック	説明	単位
Destination.value	宛先の名前。たとえば、次のようなディレクティブがあります。 Oc4jMount /j2ee/* home この場合は、home が Destination.value になります。	文字列

表 C-7 HTTP Server mod_oc4j マウント・ポイント・メトリック（続き）

メトリック	説明	単位
ErrReq.count	mod_oc4j が OC4J へのルーティングに失敗したリクエスト（セッションおよび非セッション）の合計数	ops
ErrReqNonSess.count	mod_oc4j が OC4J プロセスへのルーティングに失敗した非セッション・リクエストの合計数	ops
ErrReqSess.count	mod_oc4j が OC4J プロセスへのルーティングに失敗したセッション・リクエストの合計数	ops
Failover.count	フェイルオーバーが発生した（JVM へのアクセス中にエラーが発生して別の JVM に切り替わった）リクエストの合計数	ops
Name.value	mod_oc4j.conf で Oc4jMount ディレクティブのパスとして指定された値のエコー。DMS は、「/」や「*」などの特定の文字を「_」に変換します。指定された実際のパス名を保持するため、mntPt*id* と実際のパス名との間のマッピングを含む内部表が、mod_oc4j の初期化時に作成されます。たとえば Oc4jMount /j2ee/* home というディレクティブがある場合、/j2ee/* が Name.value になります。	文字列
NonSessFailover.count	非セッション・リクエストに対するフェイルオーバーの合計数。フェイルオーバーが発生した（JVM へのアクセス中にエラーが発生して別の JVM に切り替わった）リクエストの数	ops
SessFailover.count	セッション・リクエストに対するフェイルオーバーの合計数。フェイルオーバーが発生した（JVM へのアクセス中にエラーが発生して別の JVM に切り替わった）リクエストの数	ops
SucReq.count	mod_oc4j が OC4J インスタンスへのルーティングに成功したリクエスト（セッションおよび非セッション）の合計数	ops
SucReqNonSess.count	mod_oc4j が OC4J プロセスへのルーティングに成功した非セッション・リクエストの合計数	ops
SucReqSess.count	mod_oc4j が OC4J プロセスへのルーティングに成功したセッション・リクエストの合計数	ops

表 C-8 は、mod_oc4j 宛先メトリックを示しています。この表には、特定の宛先のメトリックのセットが含まれます。それぞれの宛先には、複数のマウント・ポイントを含めることができます。mod_oc4j.conf で指定されているマウント・ポイントごとに、mntPts サブツリーが 1 つあります。

メトリック表の名前は mod_oc4j_destination_metrics です。

表 C-8 HTTP Server mod_oc4j 宛先メトリック

メトリック	説明	単位
ErrReq.count	mod_oc4j が OC4J へのルーティングに失敗したリクエスト（セッションおよび非セッション）の合計数	ops
ErrReqNonSess.count	mod_oc4j が OC4J プロセスへのルーティングに失敗した非セッション・リクエストの合計数	ops
ErrReqSess.count	mod_oc4j が OC4J プロセスへのルーティングに失敗したセッション・リクエストの合計数	ops
Failover.count	フェイルオーバーが発生した（JVM へのアクセス中にエラーが発生して別の JVM に切り替わった）リクエストの合計数	ops
JVMCnt.value	この宛先に属するルーティング可能な OC4J JVM の合計数	JVM の数
Name.value	mod_oc4j.conf で Oc4jMount ディレクティブの宛先として指定された値のエコー。mod_oc4j.conf では、1 つの宛先が複数回指定されている可能性があります。 例 : Oc4jMount /j2ee/* home,oc4jinstance2 この場合は、home,oc4jinstance2 が Name.value になります。	文字列
NonSessFailover.count	非セッション・リクエストに対するフェイルオーバーの合計数	ops
SessFailover.count	フェイルオーバーの合計数	ops

表 C-8 HTTP Server mod_oc4j 宛先メトリック（続き）

メトリック	説明	単位
SucReq.count	mod_oc4j が OC4J へのルーティングに成功したリクエスト（セッションおよび非セッション）の合計数	ops
SucReqNonSess.count	mod_oc4j が OC4J プロセスへのルーティングに成功した非セッション・リクエストの合計数	ops
SucReqSess.count	mod_oc4j が OC4J プロセスへのルーティングに成功したセッション・リクエストの合計数	ops

C.1.7 Oracle HTTP Server SSL メトリック

表 C-9 は、OSSL メトリックを示しています。

メトリック表のタイプは ohs_ossl です。

表 C-9 OHS_OSSL メトリック

メトリック	説明	単位
checkcrl.time	SSL checkcrl の起動	時間
closessl.time	SSL 接続のクローズ	時間
connectssl.time	SSL 接続の確立	時間
dataReceive.value	OSSL データの受信	KB
dataSent.value	OSSL データの送信	KB
entercache.time	SSL entercache の起動	時間
getcache.time	SSL getcache の起動	時間
handshake.time	SSL ハンドシェイクの起動	時間
receive.time	暗号化されたメッセージの受信	時間
receiveErrors.count	受信におけるエラー発生	ops
send.time	暗号化されたメッセージの送信	時間
sendErrors.count	送信におけるエラー発生	ops
setfixup.time	SSL setfixup の起動	時間

C.2 JVM メトリック

表 C-10 は、JVM メトリックを示しています。サイトで実行中の各 Java プロセス（OC4J）に対して 1 セットのメトリックがあります。

メトリック表のタイプは JVM です。

表 C-10 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	スレッド
activeThreads.minValue	JVM 内のアクティブなスレッドの最小数	スレッド
activeThreads.maxValue	JVM 内のアクティブなスレッドの最大数	スレッド
upTime.value	JVM の稼動時間	msecs
freeMemory.value	JVM 内のヒープの空き領域の量	KB

表 C-10 JVM メトリック（JVM）（続き）

メトリック	説明	単位
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
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

C.2.1 JVM プロパティ・メトリック

Oracle Application Server は、各 Java プロパティの値を追跡するメトリックを作成します。Java プロパティは、任意の Java プロセスに対して System.getProperties() をコールすることで得られます。各 Java プロパティのメトリックは、/JVM/Properties Noun の下に作成されます。

たとえば、各プロセスには、/JVM/Properties/java_version.value という名前の java.version システム・プロパティ値を含むメトリックが必要です。システムではプロパティ名コンポーネントのピリオド「.」が「_」に変換されます。

プロセスが生きている間、JVM システム・プロパティからプロパティが削除されると、それに対応するメトリックも削除されます。値が変更されると、次回のアクセス時にメトリック値に変更が反映されます。システム・プロパティに新しくプロパティが追加されると、新規メトリックが作成されます。

注意： JVM プロパティ・メトリックは、AggreSpy の Spies text リンクを使用するか、メトリックを表示するための dmstool コマンドを使用しないと表示できません。

表 C-11 JVM/ プロパティ - JVM システム・プロパティ・メトリック

メトリック	説明	単位
各システム・プロパティに対し、1 つのメトリックが作成されます。各プロパティ名に「.」の文字がある場合、「_」に置き換えられます。	Java システム・プロパティの値が含まれます。	文字列

C.3 JDBC メトリック

次の各表は、Oracle Application Server の JDBC メトリックを一覧にしたものです。

C.3.1 JDBC ドライバ・メトリック

表 C-12 は、JDBC ドライバ・メトリックを示しています。JDBC ドライバ・メトリックは、各 JVM に 1 セットあります。

メトリック表のタイプは JDBC_Driver です。

表 C-12 /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

表 C-12 /JDBC/Driver - JDBC_Driver メトリック（続き）

メトリック	説明	単位
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

C.3.2 JDBC データソース・メトリック

表 C-13 は、JDBC データソース・メトリックを示しています。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。

メトリック表のタイプは JDBC_DataSource です。

表 C-13 /JDBC/*data-source-name* - JDBC_Data Source 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

C.3.3 JDBC ドライバ固有の接続メトリック

表 C-14 は、JDBC ドライバ接続メトリックを示しています。JDBC 接続メトリックは、各接続に 1 セットあります。

メトリック表のタイプは JDBC_Connection です。

表 C-14 /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（文の解析に要した時間は含まれません。文の解析時間を含むメトリックは、表 C-17 の Execute.Time を参照）	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
JDBC_Connection_URL	この接続に指定された URL	
JDBC_Connection_Username	この接続に使用されるユーザー名	
LogicalConnection.value	物理接続の場合は、その論理接続を意味します（存在する場合）。	

表 C-14 /JDBC/Driver/*CONNECTION* - JDBC ドライバ接続メトリック（続き）

メトリック	説明	単位
StatementCacheHit.count	キャッシュにある文	ops
StatementCacheMiss.count	キャッシュにない文	ops

C.3.4 JDBC データソースに固有の接続メトリック

表 C-15 は、JDBC データソース・メトリックを示しています。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。

メトリック表のタイプは JDBC_Connection です。

表 C-15 /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
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（文の解析に要した時間は含まれません。文の解析時間を含むメトリックは、表 C-18 の Execute.Time を参照）	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
JDBC_Connection_Url	この接続に指定された URL	
JDBC_Connection_Username	この接続に使用されるユーザー名	
LogicalConnection.value	物理接続の場合は、その論理接続を意味します（存在する場合）。	
StatementCacheHit.count	キャッシュにある文	
StatementCacheMiss.count	キャッシュにない文	

C.3.5 JDBC 接続ソース・メトリック

表 C-16 は、JDBC 接続ソース・メトリックを示しています。

メトリック表のタイプは JDBC_ConnectionSource です。

表 C-16 JDBC 接続ソース・メトリック

メトリック	説明	単位
CacheFreeSize.count	接続キャッシュ内の空きスロットの数	ops
CacheFreeSize.maxValue	接続キャッシュ内の空きスロットの最大数	connections
CacheFreeSize.minValue	接続キャッシュ内の空きスロットの最小数	connections
CacheFreeSize.value	接続キャッシュ内の空きスロットの数	connections
CacheGetConnection.active		スレッド
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

表 C-16 JDBC 接続ソース・メトリック（続き）

メトリック	説明	単位
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	ops
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	ops
CacheSize.value	キャッシュ内の物理接続の数	ops

C.3.6 JDBC ドライバ文メトリック

表 C-17 は、JDBC 文メトリックを示しています。JDBC 文メトリックは、各文の各接続に 1 セットあります。

メトリック表のタイプは JDBC_Statement です。

注意： JDBC 文メトリックは、JDBC 文キャッシュが有効になっており、プロパティ oracle.jdbc.DMSStatementCachingMetrics の値が true に設定されている JDBC 接続に対してのみ使用できます。JDBC 文キャッシュが無効になっている場合は、プロパティ oracle.jdbc.DMSStatementMetrics を true に設定することによって JDBC 文メトリックを使用可能にします。これらのプロパティを両方ともデフォルトで false に設定しておくと、パフォーマンスを向上させ、負荷のかかるメトリックの収集を回避することができます。

表 C-17 /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と、この文の解析に要した時間	msecs
Fetch.time	この文によって他のフェッチに費やされた時間	msecs
SQLText.value	実行中の SQL	

C.3.7 JDBC データソース文メトリック

表 C-18 は、JDBC 文メトリックを示しています。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。

メトリック表のタイプは JDBC_Statement です。

注意： JDBC 文メトリックは、JDBC 文キャッシュが有効になっており、プロパティ oracle.jdbc.DMSStatementCachingMetrics の値が true に設定されている JDBC 接続に対してのみ使用できます。JDBC 文キャッシュが無効になっている場合は、プロパティ oracle.jdbc.DMSStatementMetrics を true に設定することによって JDBC 文メトリックを使用可能にします。これらのプロパティをデフォルトで false に設定しておくと、パフォーマンスを向上させ、負荷のかかるメトリックの収集を回避することができます。

表 C-18 /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と、この文の解析に要した時間	msecs
Fetch.time	この文によって他のフェッチに費やされた時間	msecs
SQLText.value	実行中の SQL	

C.3.8 JDBC 接続プールの統計メトリック

表 C-19 は、JDBC 接続プールの統計メトリックを示しています。

メトリック表のタイプは jdbc_connection_pool_stats です。

表 C-19 JDBC 接続プールの統計メトリック

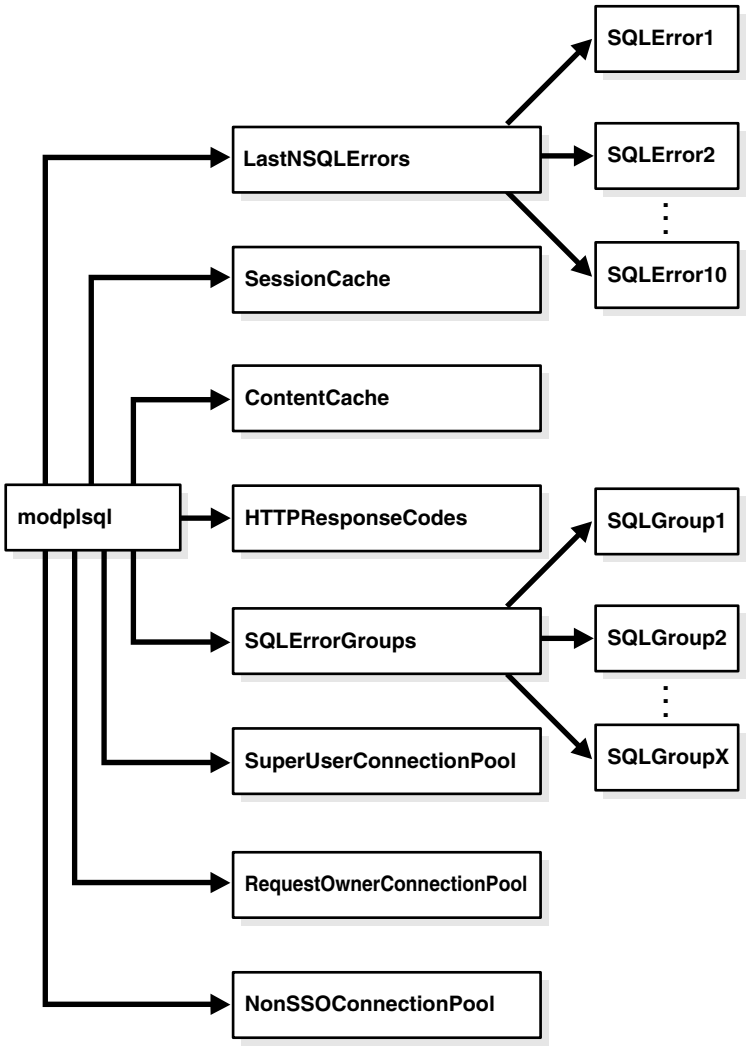
メトリック	説明	単位
CloseConnectionCount.value	閉じられた接続の数	connections
CreateConnectionCount.value	作成された接続の数	connections
FreePoolSize.maxValue	プール内で使用可能な接続の数	connections
FreePoolSize.minValue	プール内で使用可能な接続の数の上限	connections
FreePoolSize.value	プール内で使用可能な接続の数の上限	connections
FreePoolSizeUpperBound.value	プール内で使用可能な接続の数の上限	connections
PoolSize.maxValue	プール内の接続（使用されているものと使用可能なもの）の合計数	connections
PoolSize.minValue	プール内の接続（使用されているものと使用可能なもの）の合計数	connections
PoolSize.value	プール内の接続（使用されているものと使用可能なもの）の合計数	connections
PoolSizeLowerBound.value	プール内の接続の合計数の下限	connections
PoolSizeUpperBound.value	プール内の接続の合計数の上限	connections
UseTime.time	接続の使用に費やされた時間	時間
WaitTime.time	接続が使用可能になるまで待機に費やされた時間	時間
WaitingThreadCount.maxValue	接続待ち状態のスレッドの数	件数
WaitingThreadCount.minValue	接続待ち状態のスレッドの数	件数
WaitingThreadCount.value	接続待ち状態のスレッドの数	件数

C.4 mod_plsql メトリック

この項では、Oracle Application Server mod_plsql メトリックについて説明します。

図 C-1「mod_plsql メトリック・ツリー」は、mod_plsql 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 C-1 mod_plsql メトリック・ツリー



/modplsql/HTTPResponseCodes メトリックでは、mod_plsql によって戻されるレスポンス・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 レスポンス・タイプについて、件数、つまりレスポンス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、レスポンス・コード「HTTP 404: Not found」の件数が保持されます。

表 C-20 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 C-20 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 C-21 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 C-21 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 C-22 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 Database エラー・メッセージ』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 Database エラー・メッセージ』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 C-22 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 C-23 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、『Oracle Database エラー・メッセージ』を参照してください。

表 C-23 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 C-24 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 C-24 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	usecs
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	usecs
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	usecs
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	usecs
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 C-25 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 C-25 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	usecs
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	usecs
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	usecs
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	usecs
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 C-26 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 C-26 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	usecs
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	usecs
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	usecs
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	usecs
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

C.5 Oracle Process Manager and Notification Server（OPMN）メトリック

この項では、Oracle Process Manager and Notification Server（OPMN）メトリックについて説明します。

この項には、次の項目が含まれています。

- OPMN_PM メトリック表
- OPMN_OC4J_PROC 表
- OPMN_HOST_STATISTICS メトリック表
- OPMN_IAS_INSTANCE メトリック表
- OPMN_IAS_COMPONENT 表
- OPMN ONS メトリック
- OPMN_APPCTX 表

C.5.1 OPMN_PM メトリック表

opmn_pm メトリック表は、OPMN DMS メトリックのプロセス・マネージャ・サブツリーのルートになります。このメトリック表のメトリックには、OPMN リクエストに関する統計が含まれます。OPMN リクエストは、クライアントから OPMN に対して発行されるコマンドです。たとえば、DCM は 1 つ以上の OPMN 管理対象プロセスで操作を実行します。

リクエストの処理結果は、次のいずれかになります。

- 成功 : OPMN がリクエストを正常に処理したことを示します。
- 部分的に成功 : OPMN がリクエストの一部のみを正常に処理したことを示します。たとえば、クライアントが OPMN に対して OC4J プロセスを 3 つ開始するようリクエストしても、2 つしか開始されない場合、このリクエストの結果は部分的な成功となります。
- 失敗 : リクエストの処理に失敗したことを示します。

表 C-27 は、メトリック表のタイプ opmn_pm を示しています。

表 C-27 OPMN_PM メトリック

メトリック	説明	単位
jobWorkerQueue.value	OPMN ワーカー・キューにあるジョブの数	ops
lReq.count	OPMN が処理したローカル HTTP リクエストの数	ops
procDeath.count	プロセス・マネージャによって起動され、その後終了したプロセスの数	ops
procDeathReplace.count	プロセス・マネージャによって終了が検出され、その後再起動されたプロセスの数	ops
reqFail.count	失敗した HTTP リクエストの数	ops
reqPartialSucc.count	部分的に成功した HTTP リクエストの数	ops
reqSucc.count	成功した HTTP リクエストの数	ops
rReq.count	OPMN が処理したリモート HTTP リクエストの数	ops
workerThread.value	ワーカー・スレッドの数	スレッド

C.5.2 OPMN_OC4J_PROC 表

表 C-28 は OPMN OC4J プロセス・メトリック表で、OC4J プロセスに関する情報を提供します。

メトリック表のタイプは opmn_oc4j_proc です。

表 C-28 OPMN_OC4J_PROC メトリック

メトリック	説明	単位
oc4jinstance.value		
oc4jIsland.value	これは下位互換性のメトリックです。	

C.5.3 OPMN_HOST_STATISTICS メトリック表

OPMN ホスト統計メトリック表は、OPMN プロセスを実行するホストに関する情報を提供します。

表 C-29 は、メトリック表のタイプ opmn_host_statistics を示しています。

表 C-29 OPMN_HOST_STATISTICS メトリック

メトリック	説明	単位
cpuIdle.value	過去の指定しない時間から、CPU がアイドル状態であった時間（ミリ秒）	ミリ秒
freePhysicalMem.value	ホスト・マシンの物理メモリーの空き容量	KB
numProcessors.value	ホスト・マシンで使用可能なプロセッサの数	整数
timestamp.value	ホストの統計が収集された時刻。このタイムスタンプは、過去の指定しない時間からのミリ秒です。	指定しない時間からのミリ秒
totalPhysicalMem.value	ホスト・マシンで使用可能な総物理メモリー	KB

C.5.4 OPMN_IAS_INSTANCE メトリック表

OPMN IAS インスタンスのサブツリーには、Oracle Application Server インスタンスのノード名が示されます。

表 C-30 は、メトリック表のタイプ opmn_ias_instance を示しています。

表 C-30 OPMN_IAS_INSTANCE メトリック

メトリック	説明	単位
iasCluster.value	Oracle Application Server インスタンスの Oracle Application Server クラスタ名	文字列

C.5.5 OPMN_IAS_COMPONENT 表

OPMN IAS コンポーネントのサブツリーは、Oracle Application Server コンポーネントを示します。OPMN IAS コンポーネントのサブツリーには、コンポーネント情報を含む複数のメトリック表が含まれます。

表 C-31 は、メトリック表のタイプ opmn_process_type を示しています。

表 C-31 OPMN_PROCESS_TYPE メトリック

メトリック	説明	単位
moduleId.value	属性 module-id の値。opmn.xml 構成ファイルの process-type タグに指定されています。	文字列

表 C-32 は、メトリック表のタイプ opmn_process_set を示しています。

表 C-32 OPMN_PROCESS_SET メトリック

メトリック	説明	単位
numProcConf.value	このプロセス・セットに構成されているプロセスの数または最大数	文字列（整数）
numProcs.value	このプロセス・セットに存在するプロセスの数	
IsService.value	プロセス・セットのサービスとしての構成	文字列
reqFail.count	このプロセス・セットで失敗した HTTP リクエストの数	ops
reqPartialSucc.count	このプロセス・セットで部分的に成功した HTTP リクエストの数	ops
reqSucc.count	このプロセス・セットで成功した HTTP リクエストの数	ops
restartOnDeath.value	プロセスの終了時に OPMN がそのプロセスを再起動するかどうか	文字列（ブール型）

表 C-33 は、メトリック表のタイプ opmn_process を示しています。

表 C-33 OPMN_PROCESS メトリック

メトリック	説明	単位
cpuTime.value	このプロセスによって使用された CPU タイム	CPU msecs
heapSize.value	このプロセスのヒープ・サイズ	KB
iasCluster.value	このプロセスの Oracle Application Server クラスタ名	文字列
iasInstance.value	このプロセスの Oracle Application Server インスタンス名	文字列
indexInSet.value	このプロセス・セットのプロセス索引。この値は OPMN 管理対象プロセスにのみ有効です。それ以外のプロセスについては常に 0 となり、この値は意味を持ちません。	文字列（整数）

表 C-33 OPMN_PROCESS メトリック（続き）

メトリック	説明	単位
memoryUsed.value	このプロセスによって使用されたメモリー容量 このメトリックはオペレーティング・システム固有の方法で計算されます。 UNIX の場合、この値はプロセス・イメージのメモリー使用量を示します。これが、このプロセスで使用中のすべてのメモリーになります。 Windows の場合、この値はワーキング・セットのメモリー使用量を示します。これは、タスク マネージャによって「メモリ使用量」列にレポートされる値と同じです。ワーキング・セットは、プロセス内のスレッドによって最近使用されたメモリー・ページのセットです。システムの空きメモリーがしきい値を超える場合、ページは使用されていなくても、プロセスのワーキング・セット内に残ります。空きメモリーがしきい値を下回ると、ページはワーキング・セットから削除されます。これらのページが必要になると、ソフトフォールトとなり、メイン・メモリーから削除される前にワーキング・セットへ戻されます。	
pid.value	このプロセスのプロセス ID	
privateMemory.value	このプロセスのプライベート・メモリー	KB
sharedMemory.value	このプロセスの共有メモリー	KB
startTime.value	このプロセスの開始時間	msecs
status.value	このプロセスのステータス。有効なステータスの値は次のとおりです。 ■ NONE: 新しいプロセス・スロットです。操作はまだ何も行われていません（ステータスなし）。 ■ Init: プロセスは開始されました。OPMN は初期化の完了を待機しています。 ■ Alive: プロセスは完全に開始されました。 ■ Stop: プロセスの停止操作を実行しています。 ■ Stopped: プロセスは完全に停止しています。 ■ Bounce: 終了しないプロセスの再起動を実行しています。 ■ Restart: 新しい起動操作を発行する前に、プロセス停止操作を実行しています。 ■ InitFail: 初期化がタイムアウトする前に失敗しました。再試行制限内で停止と起動が試行されます。 ■ BounceFail: 終了しないプロセスの再起動に失敗しました。再試行制限内で停止と起動が試行されます。	文字列
type.value	このプロセスのタイプ。プロセス・タイプの詳細は、表 C-31 を参照してください。	
uid.value	OPMN によってこのプロセスに割り当てられた ID	
upTime.value	このプロセスの稼動時間	msecs

表 C-34 は、メトリック表のタイプ opmn_connect を示しています。

表 C-34 OPMN_CONNECT メトリック

メトリック	説明	単位
desc.value	ポートの説明（利用可能な場合）	文字列
host.value	ホスト名	文字列（ホスト名）
protocol.value		
port.value	ポート番号	文字列（ポート番号）

C.5.6 OPMN ONS メトリック

Oracle Process Manager and Notification Server ONS サブツリーには、Oracle Notification System（ONS）情報が含まれます。

表 C-35 は、メトリック表のタイプ opmn_ons を示しています。

表 C-35 OPMN_ONS メトリック

メトリック	説明	単位
notifProcessed.value	ONS によって処理された通知の数	ops
notifProcessQueue.value	プロセス・キューにある通知の数	ops
notifReceived.value	ONS によって受信された通知の数	ops
notifReceiveQueue.value	受信キューにある通知の数	ops
workerThread.value	ワーカー・スレッドの数	文字列（スレッド）

表 C-36 は、local_port メトリックを示しています。../ons/local_port サブツリーには、ONS ローカル・ポートに関する情報が表示されます。

メトリック表のタイプは opmn_connect です。

表 C-36 OPMN ONS LOCAL_PORT メトリック

メトリック	説明	単位
desc.value	ポートの説明	文字列
host.value	ホスト名	文字列
port.value	ポート番号	文字列

表 C-37 は、remote_port メトリックを示しています。../ons/remote_port サブツリーには、ONS リモート・ポートに関する情報が表示されます。

メトリック表のタイプは opmn_connect です。

表 C-37 OPMN ONS REMOTE_PORT メトリック

メトリック	説明	単位
desc.value	ポートの説明	文字列
host.value	ホスト名	文字列
port.value	ポート番号	文字列

表 C-38 は、request_port メトリックを示しています。../ons/request_port サブツリーには、ONS リクエスト・ポートに関する情報が表示されます。

メトリック表のタイプは opmn_connect です。

表 C-38 OPMN ONS REQUEST_PORT メトリック

メトリック	説明	単位
desc.value	ポートの説明	文字列
host.value	ホスト名	文字列
port.value	ポート番号	文字列

表 C-39 は、opmn_ons_topo_entry メトリックを示しています。

表 C-39 OPMN ONS TOPO エントリ・メトリック

メトリック	説明	単位
protocol.value	ONS プロトコル・バージョン	
port.value	ポート値	
ip.value	IP アドレス	

C.5.7 OPMN_APPCTX 表

表 C-40 は、opmn_appctx メトリックを示しています。

表 C-40 OPMN APPCTX メトリック

メトリック	説明	単位
rtid.value		
routable.value		
state.value		

C.6 DMS 内部メトリック

表 C-41 は、DMS 内部クロック・メトリックを示しています。

表 C-41 DMS 内部クロック・メトリック

メトリック	説明	単位
logicalTime.value	DMS クロックで測定した現在の時刻	ティック
measuredFrequency.value	測定された毎秒のクロック刻み数	ティック
measuredResolution.value	このクロックで測定されたティック間の時間	
name.value		
overheadPerCall.value	このクロックで時刻を得るために必要な平均コール期間	
reportedFrequency.value	クロック時間がレポートされる毎秒のティック数	ティック
requestedUnits.value	時間がレポートされる単位の文字列による説明	

表 C-42 は、DMS 内部ログ・メトリックを示しています。

表 C-42 DMS 内部ログ・メトリック

メトリック	説明	単位
initLogging.count		ops
messagesLogged.count		ops
status.value		

表 C-43 は、DMS 内部測定メトリックを示しています。

表 C-43 DMS 内部測定メトリック

メトリック	説明	単位
createNoun.count		ops
createSensor.count		ops
destroyNoun.count		ops

表 C-43 DMS 内部測定メトリック（続き）

メトリック	説明	単位
destroySensor.count		ops
lastTreeNodeID.value		
sampleMetric.count		ops
sensorWeight.value		
treeNodes.maxValue		
treeNodes.value		

表 C-44 は、DMS 内部コレクタ・メトリックを示しています。

表 C-44 DMS 内部コレクタ・メトリック

メトリック	説明	単位
logger.count		ops
logger.logged		ops
responseGenerateTime.active		スレッド
responseGenerateTime.avg		
responseGenerateTime.completed		
responseGenerateTime.maxActive		
responseGenerateTime.maxTime		
responseGenerateTime.minTime		
responseGenerateTime.time		

表 C-45 は、DMS 内部トランストレース・メトリックを示しています。

表 C-45 DMS 内部トランストレース・メトリック

メトリック	説明	単位
expireMessages.avg		
expireMessages.completed		
expireMessages.maxActive		
expireMessages.maxTime		
expireMessages.minTime		
expireMessages.time		
messageCount.value		
pendingMessageCount.value		
s_debugEnabled.value		
s_dumpEnabled.value		
s_ecidEnabled.value		
s_transTraceEnabled.value		
storeSize.value		

D

OC4J パフォーマンス・メトリック

この付録には、次のメトリックが含まれています。

- JTA リソースのメトリック
- JCA メトリック
- OC4J の J2EE アプリケーション・メトリック
- OC4J JMS メトリック
- OC4J タスク・マネージャのメトリック
- Java Object Cache（JOC）メトリック

D.1 JTA リソースのメトリック

表 D-1 は、JTA リソースのメトリックを示しています。

メトリック表のタイプは JTAResource です。

表 D-1 /oc4j/JTAResource メトリック

JSR-77 JTA リソースのメトリック	説明	単位
ActiveCount jtaresource_active	アクティブなトランザクションの合計件数。この値が常時大きいと、サーバーの負荷が高いことを示す場合があります。	ops
AverageCommitTime jtaresource_averageCommit	すべてのトランザクションの平均コミット時間。これは jtaresource_performTransaction の平均値ですが、システムで他の問題が発生すると値が増加する場合があります。	msecs
CommittedCount jtaresource_committed	コミットしたトランザクションの合計件数。	ops
HeuristicCommittedCount jtaresource_heuristicCommitted	経験則的にコミットしたトランザクションの合計件数。 この値が大きいと、システムやアプリケーションの自動化が不十分であることを示します。たとえば、全般的にシステム管理が過大、トランザクション・アーキテクチャの扱いが不適切、大がかりな管理を要する具体的な問題が発生した、などが原因として考えられます。これは、従属 TransactionManager が原因であり、準備状態にあるときにリソース・マネージャがロールバックされることが原因ではありません。	ops
HeuristicCount jtaresource_heuristic	経験則的にロールバックしてコミットしたすべてのトランザクションの合計件数。heuristicCommittedCount および heuristicRolledbackCount のコメントを参照してください。	ops
HeuristicMixedExceptionCount jtaresource_heuristicMixedException	発生した HeuristicMixedExceptions の合計件数。 この値が大きいと、管理上の介入が一貫性や適切性を欠くために、潜在的に ACID でない結果が多数発生することを示す場合があります。	ops
HeuristicRollbackExceptionCount jtaresource_heuristicRollbackException	発生した HeuristicRollbackExceptions の合計件数。 この値が大きいと、システムやアプリケーションの自動化が不十分であることを示します。たとえば、全般的にシステム管理が過大、トランザクション・アーキテクチャの扱いが不適切、大がかりな管理を要する具体的な問題が発生した、などが原因として考えられます。トランザクションがアクティブなとき、管理ロールバックがトランザクション・マネージャのルート・レベルであることを示す rolledbackDueToAdminCount メトリックとは異なり、この原因は、従属 TransactionManager と、準備状態にあるときにリソース・マネージャがロールバックされることのいずれかです。	ops
HeuristicRolledbackCount jtaresource_heuristicRolledback	経験則的にロールバックしたトランザクションの合計件数。 この値が大きいと、システムやアプリケーションの自動化が不十分であることを示します。たとえば、全般的にシステム管理が過大、トランザクション・アーキテクチャの扱いが不適切、大がかりな管理を要する具体的な問題が発生した、などが原因として考えられます。これは、従属 TransactionManager が原因であり、準備状態にあるときにリソース・マネージャがコミットされることが原因ではありません。	ops
IllegalStateExceptionCount jtaresource_illegalStateException	発生した IllegalStateExceptions の合計件数。 この値が大きくなることはまれで、管理上の介入が行われた場合にのみ発生します。	ops
PerformTransaction jtaresource_performTransaction	トランザクションが開始してから終了するまでの時間。 これはパフォーマンスの問題における高レベルのインジケータとして有用ですが、トランザクションが開始してから終了するまでの時間のみを測定したものであるため、その時間内に発生したことがすべて原因となる場合があります。たとえば、アプリケーション・ロジック、データベース・アクティビティ、JMS アクティビティ、またはトランザクション処理などです。	msecs

表 D-1 /oc4j/JTAResource メトリック（続き）

JSR-77 JTA リソースのメトリック	説明	単位
RollbackCompletion	ロールバックの完了に必要な時間。	
jtaresource_rollbackCompletion	この値が大きいと、リソース・マネージャでロールバック・コールに遅延が生じていることを示しています。これは、ネットワーク待機時間またはリソース・マネージャの問題が原因で発生する場合があります。	
RollbackExceptionCount	発生した RollbackExceptions の合計件数。	ops
jtaresource_rollbackException	この値が大きいと、システムに問題（データベースのダウンなど）が発生してパフォーマンスが低下したことを示す場合があります。直接的な内部システム障害が原因で発生する場合も、なんらかの理由でアプリケーションが setRollbackOnly をコールしていることが原因で発生する場合もあります。 細かなロールバック原因件数、ロールバックの原因調査用のログ、および setRollbackOnly をコールするアプリケーション・コードを調べることをお薦めします。	
RolledbackCount	ロールバックしたトランザクションの合計件数。	ops
jtaresource_rolledback	この値が大きいと、システムに問題（データベースのダウンなど）が発生してパフォーマンスが低下したことを示す場合があります。	
RolledbackDueToAdminCount	管理アクションが原因でロールバックしたトランザクションの合計件数。	ops
jtaresource_rolledbackDueToAdmin	この値が大きいと、システムやアプリケーションの自動化が不十分であることを示します。たとえば、全般的にシステム管理が過大、トランザクション・アーキテクチャの扱いが不適切、大がかりな管理を要する具体的な問題が発生した、などが原因として考えられます。	
RolledbackDueToAppCount	setRollbackOnly または rollback を、アプリケーションが明示的にコールしたことが原因でロールバックしたトランザクションの合計件数。	ops
jtaresource_rolledbackDueToApp	この値が大きくなる原因はいろいろ考えられますが、アプリケーション内で例外を処理した（データベース更新中の SQLException 処理など）ために発生するケースが多く見られます。 setRollbackOnly または rollback をコールするアプリケーション・コードを調べて、コールする理由を究明することをお薦めします。	
RolledbackDueToResourceCount	確保したリソースにエラーが発生したことが原因でロールバックしたトランザクションの合計件数。	ops
jtaresource_rolledbackDueToResource	この値が大きいと、1 つ以上のリソース・マネージャ（データベース、または OC4J とそれらのリソースとの間のネットワーク接続など）に問題があることを示す場合があります。	
RolledbackDueToTimedOutCount	タイムアウトが原因でロールバックしたトランザクションの合計件数。	ops
jtaresource_rolledbackDueToTimedOut	この値が大きいと、トランザクション、またはトランザクション境界内におけるアクティビティに時間がかかりすぎる原因となる問題があることや、指定されたタイムアウト値が十分に柔軟でないことを示す場合があります。 関連するトランザクション内で、どのアクティビティに時間がかかっているかを調べたり（アクティビティは、ある種のアプリケーションの場合あり）、transaction-manager.xml 構成ファイルの transaction-timeout 値を調整することをお薦めします。	
SecurityExceptionCount	発生した SecurityExceptions の合計件数。	ops
jtaresource_securityException	この値が大きい、つまり 0 よりも大きくなると、これを実行するスレッドの ID に問題があることを示す場合があります。	

表 D-1 /oc4j/JTAResource メトリック（続き）

JSR-77 JTA リソースのメトリック	説明	単位
SinglePhaseCommitCompletion	1 フェーズ・コミットの完了に必要な時間。	
jtaresource_singlePhaseCommitCompletion	1 フェーズ・コミットでは、1 つのリソースのみコミットされるため、PC を 2 台用意するコスト（ロギングなど）が不要になります。この値が大きいと一般的に、コミット中のリソース（データベースへのネットワーク待機時間など）に問題があることを示します。コミットに関連するリソースのメトリックを詳細に調べることをお薦めします。	
SystemExceptionCount	発生した SystemExceptions の合計件数。	
jtaresource_systemException	この値が大きくなることはありませんが、もし大きくなったら、システムに重大な障害が発生したことを示します。OC4J とリソース・マネージャのログを分析することをお薦めします。	
TransactionSuspended	トランザクションが一時停止している時間。	
jtaresource_transactionSuspended	この値が大きいと、トランザクションが一時停止状態になり、異なるトランザクション・コンテキストやトランザクション・コンテキストのない状態で、またはトランザクション・コンテキストの伝播中に、一般的に EJB メソッド・コールからのコールの戻りを待っていることを示します。 アプリケーションを分析して、一時停止中にどのアクティビティが発生したか、また伝播中にネットワーク待機時間があるかどうかを判断することをお薦めします。	
TwoPhaseCommitCompletion	2 フェーズ・コミットの完了に必要な時間。	
jtaresource_twoPhaseCommitCompletion	この値が大きいと、リソース・マネージャで準備とコミットのコールに遅延が生じていること、または OC4J でトランザクション・レコードのロギングに遅延が生じていることを示しています。 準備とコミットに関連するリソースのメトリックと、transaction-manager.xml 構成ファイルにある OC4J でのトランザクション・レコードのロギングのパフォーマンス設定を、詳細に調べることをお薦めします。	

D.2 JCA メトリック

表 D-2 は、JCA メトリックを示しています。

メトリック表のタイプは jca_connection_stats です。

表 D-2 oc4j/*application*/OracleASjms/JCAmetrics メトリック

メトリック	説明	単位
closeCount.count	閉じられた接続ハンドルの件数	ops
createCount.count	作成された接続ハンドルの件数	ops
poolName.value	接続プールの値の名前	プール名
useTime.avg	接続の使用に費やされた時間	時間
useTime.completed	接続の使用に費やされた時間	ops
useTime.maxTime	接続の使用に費やされた時間	時間
useTime.minTime	接続の使用に費やされた時間	時間
useTime.time	接続の使用に費やされた時間	時間
waitTime.avg	接続が使用可能になるまで待機に費やされた時間	時間
waitTime.completed	接続が使用可能になるまで待機に費やされた時間	ops
waitTime.maxTime	接続が使用可能になるまで待機に費やされた時間	時間
waitTime.minTime	接続が使用可能になるまで待機に費やされた時間	時間
waitTime.time	接続が使用可能になるまで待機に費やされた時間	時間

表 D-3 は、JCA 接続プールの統計メトリックを示しています。

メトリック表のタイプは jca_connection_pool_stats です。

表 D-3 /oc4j/jca_connection_pool_stats メトリック

メトリック	説明	単位
closeCount.count	閉じられた ManagedConnections の数	ops
createCount.count	作成された ManagedConnections の数	ops
errorCount.count	接続エラー・イベントの数	ops
expiredCount.count	プールから削除された期限切れ接続の数	ops
freePoolSize.maxValue	プール内の空き接続の数	connections
freePoolSize.minValue	プール内の空き接続の数	connections
freePoolSize.value	プール内の空き接続の数	connections
inactivityTimeout.value	構成パラメータ : 未使用の接続のタイムアウト	時間
inactivityTimeoutCheck.value	構成パラメータ : 未使用の接続をチェックする状況	
initial-capacity.value	構成パラメータ : プールによって事前作成される接続の数	ops
invalidCount.count	プールから削除された無効な接続の数	ops
maxPoolSize.value	構成パラメータ : 最大接続数	connections
minPoolSize.value	構成パラメータ : 最小接続数	connections
poolSize.maxValue	接続プールのサイズ	connections
poolSize.minValue	接続プールのサイズ	connections
poolSize.value	接続プールのサイズ	connections
requestTimeoutCount.count	タイムアウトが原因で失敗した接続リクエストの数	ops
scheme.value	スキーム構成パラメータ : 接続プーリング・スキーム	
useTime.avg	接続の使用に費やされた時間	時間
useTime.completed	接続の使用に費やされた時間	ops
useTime.maxTime	接続の使用に費やされた時間	時間
useTime.minTime	接続の使用に費やされた時間	時間
useTime.time	接続の使用に費やされた時間	時間
waitTime.avg	接続が使用可能になるまで待機に費やされた時間	時間
waitTime.completed	接続が使用可能になるまで待機に費やされた時間	ops
waitTime.maxTime	接続が使用可能になるまで待機に費やされた時間	時間
waitTime.minTime	接続が使用可能になるまで待機に費やされた時間	時間
waitTime.time	接続が使用可能になるまで待機に費やされた時間	時間
waitTimeout.value	構成パラメータ : fixed_wait スキームにおける接続待機のタイムアウト	
waitingThreadCount.active	接続待ち状態のスレッドの数	ops
waitingThreadCount.avg	接続がアクティブになるまで待機しているスレッドの数	
waitingThreadCount.completed	接続がアクティブになるまで待機しているスレッドの数	ops
waitingThreadCount.maxActive	接続がアクティブになるまで待機しているスレッドの数	スレッド
waitingThreadCount.maxTime	接続がアクティブになるまで待機しているスレッドの数	時間
waitingThreadCount.minTime	接続がアクティブになるまで待機しているスレッドの数	時間
waitingThreadCount.time	接続がアクティブになるまで待機しているスレッドの数	時間

D.3 OC4J の J2EE アプリケーション・メトリック

この項では、OC4J J2EE アプリケーションに関連するメトリックを一覧表示します。

この項には、次のメトリックが含まれています。

- Web モジュール・メトリック
- Web コンテキスト・メトリック
- OC4J サーブレット・メトリック
- OC4J JSP メトリック
- OC4J EJB メトリック
- OC4J OPMN 情報メトリック
- OC4J ワーク管理プール・メトリック

D.3.1 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。

表 D-4 は、Web モジュール・メトリックを示しています。

メトリック表のタイプは `oc4j_web_module` です。

表 D-4 OC4J/*application*/WEBs メトリック

メトリック	説明	単位
`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	
`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	msecs
`parseRequest.completed`	解析の済んだ Web リクエストの数	ops
`parseRequest.maxActive`	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最大数	スレッド
`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	msecs
`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	msecs
`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	msecs
`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	
`processRequest.avg`	Web リクエストのサービスに費やされた平均時間	msecs
`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.maxActive`	Web リクエストのサービスを行うスレッドの最大数	スレッド
`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	msecs
`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	msecs
`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	msecs
`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	
`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	msecs
`resolveContext.completed`	完了したコンテキスト解決のカウント	ops
`resolveContext.maxActive`	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最大数	スレッド
`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	msecs

表 D-4 OC4J/*application*/WEBs メトリック（続き）

メトリック	説明	単位
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	msecs
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。	msecs

D.3.2 Web コンテキスト・メトリック

表 D-5 は、Web コンテキスト・メトリックを示しています。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 Web コンテキスト・メトリックがあります。

メトリック表のタイプは oc4j_context です。

表 D-5 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。これには、必要とされる認証の時間も含まれます。	msecs
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	msecs
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	msecs
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	msecs
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	msecs
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	msecs
sessionActivation.maxTime	セッションがアクティブであった最大時間	msecs
sessionActivation.avg	セッション存続時間の平均値	msecs
service.time	リクエストのサービスに費やされた合計時間。サーブレットのサービス・メトリックには、データベースへのコールに費やされたすべての時間が含まれます。oc4j のサービス時間のみを調べる必要がある場合は、実行時間を適宜減算します。	msecs
service.completed	サービスされたリクエストの合計数	ops

D.3.3 OC4J サーブレット・メトリック

表 D-6 は、サーブレット・メトリックを示しています。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 セットのサーブレット・メトリックがあります。

メトリック表のタイプは oc4j_servlet です。

表 D-6 OC4J/*application*/WEBs/*context* /SERVLETS/*servlet* メトリック

メトリック	説明	単位
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	スレッド
service.avg	サーブレットのサービスに費やされた平均時間	msecs
service.completed	service() に対するコールの合計数	
service.maxActive	このサーブレットのサービスを行うスレッドの最大数	スレッド

表 D-6 OC4J/*application*/WEBs/*context* /SERVLETS/*servlet* メトリック（続き）

メトリック	説明	単位
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.minTime	サーブレットの service() コールに費やされた最小時間	msecs
service.time	サーブレットの service() コールに費やされた合計時間	msecs

D.3.4 OC4J JSP メトリック

D.3.4.1 JSP 実行時メトリック

表 D-7 は、JSP メトリックを示しています。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 JSP メトリックがあります。

メトリック表のタイプは oc4j_jspExec です。

表 D-7 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。	msecs
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	msecs
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	msecs
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	msecs
processRequest.active	JSP に対するアクティブなリクエストの現在の数	ops

D.3.4.2 JSP メトリック

表 D-8 は、JSP メトリックを示しています。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。

メトリック表のタイプは oc4j_jsp(threadsafe=true) および oc4j_jsp(threadsafe=false) です。

dmstool を使用してこれらのメトリックを一覧表示するには、メトリック表のタイプを引用符で囲みます。

次に例を示します。

```
dmstool -table "oc4j_jsp(threadsafe=true)"
```

表 D-8 OC4J/*application*/WEBs/*context* /JSP*jsp_name* メトリック

メトリック	説明	単位
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
availableInstances.value	使用可能な（つまり作成済の）インスタンスの数 この値は threadsafe=false の場合のみ示されます。	インスタンス
service.active	JSP に対するアクティブなリクエストの現在の数	
service.avg	JSP のサービスに費やされた平均時間	msecs
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.maxTime	JSP のサービスに費やされた最大時間	msecs

表 D-8 OC4J/*application*/WEBs/*context* /JSP*jsp_name* メトリック（続き）

メトリック	説明	単位
service.minTime	JSP のサービスに費やされた最小時間	msecs
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）	msecs

D.3.5 OC4J EJB メトリック

D.3.5.1 OC4J EJB セッション Bean メトリック

表 D-9 は、各セッション Bean に関する情報を示した EJB セッション Bean メトリックを示しています。

メトリック表のタイプは oc4j_ejb_session_bean です。

表 D-9 OC4J EJB セッション Bean メトリック

メトリック	説明	単位
session-type.value	セッションのタイプ（Stateless または Stateful）	文字列
transaction-type.value	トランザクションのタイプ（Container または Bean）	文字列

D.3.5.2 EJB エンティティ Bean メトリック

表 D-10 は、エンティティ Bean メトリックを示しています。Oracle Application Server 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。

メトリック表のタイプは oc4j_ejb_entity_bean です。

表 D-10 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	可能な値：container または bean	
session-type.value	可能な値：stateful または stateless	
bean-type.value	可能な値：session または entity bean	
exclusive-write-access.value	可能な値：true または false	
isolation.value	可能な値：serializable、uncommitted、committed、repeatable_read、none、DB-determined 省略された場合、この属性の値は DB-determined になります。	
persistence-type.value	可能な値：container または bean	

D.3.5.3 EJB メソッド・メトリック

表 D-11 は、EJB メソッド・メトリックを示しています。EJB Bean の各タイプ内の各メソッドに対し、1 セットの EJB メソッド・メトリックがあります。

メトリック表のタイプは 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。

表 D-11 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.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	ops
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	msecs
client.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

表 D-11 OC4J/*application*/EJBs/*ejb-jar-module*/*ejb-name*/*method-name* メトリック（続き）

メトリック	説明	単位
client.maxActive	このメソッドの実際の実装にアクセスするスレッドの最大数	ops
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	msecs
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	msecs
client.time	このメソッドの実際の実装の中で費やされた時間	msecs
ejbPostCreate.active	ejbPostCreate を実行するスレッドの現在の数	ops
ejbPostCreate.avg	ejbPostCreate で費やされた平均時間	msecs
ejbPostCreate.completed	この ejbPostCreate がコールされた回数	ops
ejbPostCreate.maxTime	ejbPostCreate で費やされた最大時間	msecs
ejbPostCreate.minTime	ejbPostCreate で費やされた最小時間	msecs
ejbPostCreate.time	ejbPostCreate メソッド（エンティティ Bean）で費やされた時間	msecs
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	
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	
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	msecs
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.maxActive	ラッパーにアクセスするスレッドの最大数	ops
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	msecs
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	msecs
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。	msecs

D.3.5.4 EJB ステートレス Bean メトリック

表 D-12 は、EJB ステートレス Bean メトリックを示しています。

メトリック表のタイプは oc4j_ejb_stateless_bean です。

表 D-12 OC4J EJB ステートレス Bean メトリック

メトリック	説明	単位
pooled.count	プールされたインスタンスの件数	件数
pooled.maxValue	プールされたインスタンスの数	ops
pooled.minValue	プールされたインスタンスの数	ops
pooled.value	プールされたインスタンスの数	ops
ready.count	準備ができたインスタンスの数	件数
ready.maxValue	準備ができたインスタンスの数	ops
ready.minValue	準備ができたインスタンスの数	ops
ready.value	準備ができたインスタンスの数	
session-type.value	セッションのタイプ	

D.3.5.5 EJB ステートフル Bean メトリック

表 D-13 は、EJB ステートフル Bean メトリックを示しています。

メトリック表のタイプは oc4j_ejb_stateful_bean です。

表 D-13 OC4J EJB ステートフル Bean メトリック

メトリック	説明	
passive.count	非アクティブ化されたインスタンスの数	件数
passive.maxValue	非アクティブ化されたインスタンスの数	ops
passive.minValue	非アクティブ化されたインスタンスの数	ops
passive.value	非アクティブ化されたインスタンスの数	ops
ready.count	準備ができたインスタンスの数	件数
ready.maxValue	準備ができたインスタンスの数	ops
ready.minValue	準備ができたインスタンスの数	ops
ready.value	準備ができたインスタンスの数	ops
session-type.value	セッションのタイプ	
transaction-type.value	トランザクション	

D.3.5.6 EJB メッセージドリブン Bean メトリック

表 D-14 は、メッセージドリブン Bean メトリックを示しています。

メトリック表のタイプは oc4j_ejb_message-driven_bean です。

表 D-14 OC4J EJB メッセージドリブン Bean メトリック

メトリック	説明	単位
applicationExceptionCount.count	スローされたアプリケーション例外の数	件数
failedMessageDeliveryCount.count	失敗したメッセージ配信の数	件数
messageDelivery.avg	メッセージの配信試行	時間
messageDelivery.completed	メッセージの配信試行	ops
messageDelivery.maxTime	メッセージの配信試行	時間
messageDelivery.minTime	メッセージの配信試行	時間
messageDelivery.time	メッセージの配信試行	時間
messageEndpointCount.value	メッセージ・エンドポイントの数	ops
messageEndpointType.value	メッセージ・エンドポイントのタイプ	クラス名
pooled.count	プールされたインスタンスの数	件数
pooled.maxValue	プールされたインスタンスの数	ops
pooled.minValue	プールされたインスタンスの数	ops
pooled.value	プールされたインスタンスの数	ops
ready.count	準備ができたインスタンスの数	件数
ready.maxValue	準備ができたインスタンスの数	ops
ready.minValue	準備ができたインスタンスの数	ops
ready.value	準備ができたインスタンスの数	ops
startTime.value	MDB サービス利用可能時間	時間
successfulMessageDeliveryCount.count	成功したメッセージ配信の数	件数

表 D-14 OC4J EJB メッセージドリブン Bean メトリック（続き）

メトリック	説明	単位
systemExceptionCount.count	スローされた SystemExceptions の数	件数
transaction-type.value	トランザクションの値	

D.3.6 OC4J OPMN 情報メトリック

表 D-15 は、OC4J OPMN 情報メトリックを示しています。

メトリック表のタイプは oc4j_opmn です。

表 D-15 OC4J OPMN 情報メトリック

メトリック	説明	単位
default_application_log.value	アプリケーション・ログ・ファイルのデフォルトのパス	
ias_cluster.value	Oracle Application Server のクラスタ名	文字列
ias_instance.value	Oracle Application Server のインスタンス名	文字列
jms_log.value	JMS ログ・ファイルのパス	文字列
oc4j_instance.value	OC4J インスタンス ID	文字列
oc4j_island.value	OC4J アイランド ID	文字列
opmn_group.value	OPMN グループ ID	文字列
opmn_sequence.value	OPMN 順序番号	文字列
rmi_log.value	RMI ログ・ファイルのパス名	文字列
server_log.value	アプリケーション・サーバーのログ・ファイルのパス	文字列

D.3.7 OC4J ワーク管理プール・メトリック

表 D-16 は、OC4J ワーク管理プール・メトリックを示しています。

メトリック表のタイプは oc4j_workManagementPool です。

表 D-16 OC4J ワーク管理プール・メトリック

メトリック	説明	単位
idleThreadCount	プール内のアイドルなスレッドの数。これは、スレッド・プールの現行の状態のメトリックです。	スレッド
keepAlive	アイドル状態のスレッドが使用可能なスレッドのプールから削除されるまでの時間。これは構成値のメトリックです。	ミリ秒
maxPoolSize	プール内のスレッドの最大数。これは構成値のメトリックです。	スレッド
maxQueueSize	キューの最大サイズ。これは構成値のメトリックです。	work_requests
minPoolSize	プール内のスレッドの最小数。これは構成値のメトリックです。	スレッド
queueFullEvent	キューの満杯が原因で作業の送信に失敗した数。これは、スレッド・プールの現行の状態のメトリックです。	ops
queueSize	現在のキュー・サイズ。これは、スレッド・プールの現行の状態のメトリックです。	work_requests
totalThreadCount	プール内のスレッドの総数。これは、スレッド・プールの現行の状態のメトリックです。	スレッド

表 D-16 OC4J ワーク管理プール・メトリック（続き）

メトリック	説明	単位
workStartDuration	作業が受け入れられてから開始されるまでの時間。これは、スレッド・プールの現行の状態のメトリックです。待機時間は、作業の送信が受け入れられてから、作業の実行が開始されるまでに経過した時間と定義されます。このメトリックは、作業リクエストの送信がプールに受け入れられてから、その作業を実行するスレッドがスレッド・プールから割り当てられるまでの時間を測定します。スレッドがただちに使用可能な場合は、使用可能なスレッドを見つけて、作業の処理に適切なコンテキストを設定するのに要したスレッド・プールの処理のオーバーヘッドが測定されます。使用可能なすべてのスレッドが他の作業リクエストの処理に当てられている場合は、この時間にキュー時間が加算されます。	ops

D.4 OC4J JMS メトリック

OC4J JMS メトリックは、次の 2 つのカテゴリに分類される、いくつかのメトリック表に編成されます。

- JMS API レベルのメトリック : JMS API で参照可能なオブジェクト（接続、セッション、プロデューサ、コンシューマ、ブラウザなど）について収集されるメトリック。JMS API レベルのメトリックは、Web クライアントおよび EJB クライアントに対してのみ収集され維持されます。アプリケーション・クライアントでも API レベルのメトリックが収集されますが、自身の JVM 内で収集を行うため、これらのメトリックを OC4J JMS サーバーで使用することはできません。
- JMS サーバーレベルのメトリック : OC4J JMS サーバーによって収集され、クライアントの状態に関係なく維持されるメトリック。JMS サーバーレベルのメトリックは、アプリケーション、Web および EJB のすべてのタイプのクライアントに対して収集され維持されます。

OC4J JMS の各メトリック表（メトリック表のタイプ）には、同じタイプのインスタンスのメトリックが含まれます（各インスタンスは一意の名前を持ちます）。メトリック表内の各インスタンスについて、メトリックのセットが収集されます。各インスタンスのメトリックの名前は、OC4J JMS によって生成される一意の ID です。

インスタンスには、別のメトリック・インスタンスの名前を値に持つ、1 つ以上のメトリックが含まれる場合があります。たとえば、JMS セッション・インスタンスには、JMS コネクション・インスタンスを含む親を指すメトリックが含まれます。このポインタを使用して、メトリック間を移動することができます。

通常、親メトリックのインスタンスには、作成された特定のタイプの子メトリックの数を示すカウンタ・メトリックが含まれます。子メトリックのインスタンスは、基礎となるオブジェクトが作成または削除されるに従って、表示または非表示されます。このカウンタでは、親の存続期間に作成されたインスタンスなどの合計数を追跡します。

注意： Oracle Application Server JMS メトリックは OC4J JMS に対してのみ使用できます（したがって、メトリックは OJMS に対しては使用できません）。

関連項目： OC4J JMS の詳細は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。

D.4.1 JMS メトリック表

OC4J JMS メトリックは、その更新方法によって次の 3 つのタイプに分類されます。

1. **CTOR** メトリック : 関連する JMS オブジェクトのコンストラクタまたは初期化ルーチンで設定されるメトリック。オブジェクトの存続期間中は変更されません。
2. **Normal** メトリック : 関連する JMS オブジェクトの状態が変更されるとすぐに更新される、オブジェクト・レベルの状態メトリック。
3. **Lazy** メトリック : 基礎となるメトリック値が変更されてもすぐには更新されずに、周期的に更新される状態メトリック。通常、これらはサーバー保存のメトリックで、期限切れのメッセージがクリーンアップされるタイミングで更新されます。

表 D-17 は、OC4J JMS メトリック表の構成の概要を示しています。

表 D-17 OC4J JMS メトリック表

JMS メトリック表のタイプ	親の表のタイプ	インスタンスの数	説明
JMSConnectionStats	JMSStats	JMS コネクションごとに 1 つ	このサーバーでアクティブな JMS コネクションの統計
JMSDestinationStats	JMSStats	永続 JMS 宛先ごとに 1 つ	OC4J JMS サーバーで既知の永続 JMS 宛先ごとの統計
JMSDurableSubscriberStats	JMSStats	JMS 永続サブスクライバごとに 1 つ	このサーバーで既知の JMS 永続サブスクリプションごとの統計
JMSMessageBrowserStats	JMSSessionStats	JMS キュー・ブラウザごとに 1 つ	このサーバーの JMS キュー・ブラウザの統計
JMSMessageConsumerStats	JMSSessionStats	JMS メッセージ・コンシューマごとに 1 つ	このサーバーでアクティブな JMS コンシューマの統計
JMSMessageProducerStats	JMSSessionStats	JMS メッセージ・プロデューサごとに 1 つ	このサーバーでアクティブな JMS プロデューサの統計
JMSPersistenceStats	JMSDestinationStats	サーバー・サイドの永続宛先ごとに 1 つ	永続宛先それぞれの永続性ファイルに対する操作の統計
JMSRequestHandlerStats	JMSStats	リモート JMS コネクションごとに 1 つ	リモート JMS コネクションのサービスを行うリクエスト・ハンドラ・スレッドの統計
JMSSessionStats	JMSConnectionStats	JMS セッションごとに 1 つ	このサーバーでアクティブな JMS セッションの統計
JMSStats	なし	1	OC4J JMS サーバーの統計
JMSStoreStats	JMSDestinationStats JMSTemporaryDestinationStats	サーバー・サイド・メッセージ・ストアごとに 1 つ	OC4J JMS サーバーのメッセージ・ストアごとの統計（1 つはキュー単位、もう 1 つは各トピックのサブスクリプション単位）
JMSTemporaryDestinationStats	JMSStats	一時 JMS 宛先ごとに 1 つ	OC4J JMS サーバーで既知の一時 JMS 宛先ごとの統計

D.4.2 JMS 統計メトリック表

表 D-18 は、JMS 統計メトリックを示しています。

メトリック表のタイプは `JMSStats` です。

表 D-18 JMSStats メトリック表

メトリック	説明	更新	単位
`activeConnections`			
`activeHandlers`			
`address`	JMS サーバーがリモート接続を受け入れるホスト名	ctor	文字列

表 D-18 JMSStats メトリック表（続き）

メトリック	説明	更新	単位
closeConnection			
closeConsumer			
commit			
connections			
createConsumer			
deqMessage			
enqMessage			
host.value	OC4J JMS サーバーが実行されている明示的なホスト名	ctor	文字列
listMessages			
messageCommitted			
messageCount			
messageDequeued			
messageDiscarded			
messageEnqueued			
messageExpired			
messagePagedIn			
messagePagedOut			
messageRecovered			
messageRolledback			
oc4j.jms.checkPermissions			
oc4j.jms.debug	oc4j.jms.debug OC4J JMS 制御ノブの値	ctor	ブール型
oc4j.jms.forceRecovery	oc4j.jms.forceRecovery OC4J JMS 制御ノブの値	ctor	ブール型
oc4j.jms.j2ee14			
oc4j.jms.listenerAttempts	oc4j.jms.listenerAttempts OC4J JMS 制御ノブの値	ctor	int 型
oc4j.jms.maxOpenFiles	oc4j.jms.maxOpenFiles OC4J JMS 制御ノブの値	ctor	int 型
oc4j.jms.messagePoll	oc4j.jms.messagePoll OC4J JMS 制御ノブの値	ctor	msecs
oc4j.jms.noDms	oc4j.jms.noDms OC4J JMS 制御ノブの値	ctor	ブール型
oc4j.jms.noJmx			
oc4j.jms.pagingThreshold			
oc4j.jms.printStackTrace			
oc4j.jms.reconnectAttempts			
oc4j.jms.reconnectWait			
oc4j.jms.rememberALLXids			
oc4j.jms.saveAllExpired	oc4j.jms.saveAllExpired OC4J JMS 制御ノブの値	ctor	ブール型
oc4j.jms.serverPoll	oc4j.jms.serverPoll OC4J JMS 制御ノブの値	ctor	msecs
oc4j.jms.socketBufsize	oc4j.jms.socketBufsize OC4J JMS 制御ノブの値	ctor	int 型
peekMessage			
pendingMessageCount			
port.value	JMS サーバーが着信接続をリスニングする TCP/IP ポート	ctor	int 型

表 D-18 JMSStats メトリック表（続き）

メトリック	説明	更新	単位
registerConnection			
rollback			
startTime	OC4J JMS サーバー起動時の System.currentTimeMillis()	ctor	msecs
stats			
storeSize			
taskManagerInterval	OC4J タスク・マネージャがスケジューリングを実行する時間隔（および OC4J JMS の期限切れタスクのスケジューリングを実行する時間隔）	ctor	msecs

D.4.3 JMS リクエスト・ハンドラの統計

表 D-19 は、JMS リクエスト・ハンドラの統計を示しています。

メトリック表のタイプは JMSRequestHandlerStats です。

表 D-19 JMSRequestHandlerStats メトリック

メトリック	説明	更新	単位
address.value	リモート接続の開始元のホスト名（暗黙的な特殊アドレス）	ctor	文字列
connectionID.value	JMSConnectionStats インスタンスの ID	ctor	文字列
host.value	リモート接続の開始元の明示的なホスト名	ctor	文字列
port.value	リモート接続の開始元の TCP/IP ポート	ctor	int 型
startTime.value	リクエスト・ハンドラ起動時の System.currentTimeMillis()	ctor	文字列

D.4.4 JMS コネクションの統計

表 D-20 は、JMS コネクションの統計を示しています。

メトリック表のタイプは JMSConnectionStats です。

表 D-20 JMSConnectionStats メトリック

メトリック	説明	更新	単位
address.value	コネクション・ファクトリに指定された、この接続のリモート JMS サーバー・ホストの暗黙的なホスト名。非ローカル接続の場合のみ表示されます。	ctor	文字列
clientID.value	この接続に対して管理的に構成（ctor）またはプログラム的に設定（normal）されたクライアント ID	ctor/normal	文字列
domain.value	この接続の JMS ドメイン（queue、topic または unified）	ctor	文字列
exceptionListener.value	この接続の現在の例外リスナーの文字列化名	normal	文字列
host.value	この接続のリモート JMS サーバー・ホストの明示的なホスト名。非ローカル接続の場合のみ表示されます。	ctor	文字列
isLocal.value	JMS コネクションが、同じ JVM の OC4J JMS サーバーに対してローカルである場合のみ true	ctor	ブール型
isXA.value	接続が XA モードである場合のみ true	ctor	ブール型
port.value	この接続のリモート JMS サーバーのポート。非ローカル接続の場合のみ表示されます。	ctor	int 型
startTime.value	この接続の作成時の System.currentTimeMillis()	ctor	msecs

表 D-20 JMSConnectionStats メトリック（続き）

メトリック	説明	更新	単位
user.value	この接続のユーザー ID	ctor	文字列
メソッド名	この接続オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

D.4.5 JMS セッションの統計

表 D-21 は、JMS セッションの統計を示しています。

メトリック表のタイプは JMSSessionStats です。

表 D-21 JMSSessionStats メトリック

メトリック	説明	更新	単位
acknowledgeMode.value	このセッションの確認モード。有効なモードは、AUTO_ACKNOWLEDGE、CLIENT_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE および SESSION_TRANSACTED です。	ctor	文字列
domain.value	このセッションの JMS ドメイン（queue、topic または unified）	ctor	文字列
isXA.value	セッションが XA モードである場合のみ true	ctor	ブール型
sessionListener.value	このセッションの現在の識別リスナーの文字列化名	normal	文字列
startTime.value	このセッション作成時の System.currentTimeMillis()	ctor	msecs
transacted.value	このセッションが処理された場合のみ true	ctor	ブール型
txid.value	このセッションに関連付けられている現在のローカル・トランザクションのカウント（整数）。カウンタは、ローカル・トランザクションがコミットまたはロールバックされるタイミングで増加し、処理されないセッションについては設定されません。	normal	int 型
xid.value	このセッションに関連付けられている現在の分散トランザクションの Xid。ローカル・トランザクション・モードの場合は NULL または空の文字列に設定され、このセッションがグローバル・トランザクションに参加しない場合は設定されません。	normal	文字列
メソッド名	このセッション・オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

D.4.6 JMS メッセージ・プロデューサの統計

表 D-22 は、JMS プロデューサの統計を示しています。

メトリック表のタイプは JMSProducerStats です。

表 D-22 JMSProducerStats メトリック

メトリック	説明	更新	単位
deliveryMode.value	このプロデューサの現在の配信モード。有効な配信モードは、PERSISTENT および NON_PERSISTENT です。	normal	文字列
destination.value	このプロデューサの識別された宛先の名前。宛先が不明の場合は NULL または空の文字列になります。	ctor	文字列
disableMessageID.value	メッセージ ID がこのプロデューサに対して無効な場合は true	normal	ブール型
disableMessageTimestamp.value	メッセージのタイムスタンプがこのプロデューサに対して無効な場合は true	normal	ブール型
domain.value	このプロデューサの JMS ドメイン（queue、topic または unified）	ctor	文字列
priority.value	このプロデューサの現在の優先順位	normal	int 型

表 D-22 JMSProducerStats メトリック（続き）

メトリック	説明	更新	単位
startTime.value	このプロデューサ作成時の System.currentTimeMillis()	ctor	msecs
timeToLive.value	このプロデューサの現在の TTL	normal	msecs
メソッド名	このプロデューサ・オブジェクトでの、すべての主要なメソッド・コールに対するフェーズ・タイマー・メトリック（PhaseEvent Sensor）	normal	

D.4.7 JMS メッセージ・ブラウザの統計

表 D-23 は、JMS ブラウザの統計を示しています。

メトリック表のタイプは JMSBrowserStats です。

表 D-23 JMSBrowserStats メトリック

メトリック	説明	更新	単位
destination.value	このブラウザの宛先の名前	ctor	文字列
selector.value	このブラウザのメッセージ・セレクタ。指定されていない場合は NULL または空の文字列になります。	ctor	文字列
startTime.value	このブラウザ作成時の System.currentTimeMillis()	ctor	msecs
メソッド名	このブラウザ・オブジェクトでの、すべての主要なメソッド・コールの時間隔タイマー・メトリック（PhaseEvent Sensor）。個々の列挙オブジェクトで hasMoreElements および nextElement へのコールが作成されますが、ブラウザ・オブジェクトではデータの収集を簡単にするために PhaseEvent としてカウントされ、複数の列挙を同じブラウザでアクティブにできます。	normal	

D.4.8 JMS メッセージ・コンシューマの統計

表 D-24 は、JMS メッセージ・コンシューマの統計を示しています。

メトリック表のタイプは JMSMessageConsumerStats です。

表 D-24 JMSMessageConsumerStats

メトリック	説明	更新	単位
destination.value	このコンシューマの宛先の名前	ctor	文字列
domain.value	このコンシューマの JMS ドメイン（queue、topic または unified）	ctor	文字列
messageListener.value	このコンシューマの現在のメッセージ・リスナーの文字列化名	normal	文字列
name.value	このコンシューマの永続サブスクライバの名前。永続トピック・サブスクリプションの場合のみ表示されます。	ctor	文字列
noLocal.value	サブスクリプションの非ローカル設定。トピック・コンシューマの場合のみ表示されます。	ctor	ブール型
selector.value	このコンシューマのメッセージ・セレクタ。指定されていない場合は NULL または空の文字列になります。	ctor	文字列
startTime.value	このコンシューマ作成時の System.currentTimeMillis()	ctor	msecs
メソッド名	このコンシューマ・オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

D.4.9 JMS 永続サブスクリプションの統計

表 D-25 は、JMS 永続サブスクリプションの統計を示しています。

メトリック表のタイプは JMSDurableSubscriptionStats です。

表 D-25 JMSDurableSubscriptionStats メトリック

メトリック	説明	更新	単位
clientID.value	この永続サブスクリプションに関連付けられているクライアント ID	ctor	文字列
destination.value	この永続サブスクリプションのトピックの名前	ctor	文字列
isActive.value	この永続サブスクリプションが、コンシューマによって使用され現在アクティブである場合のみ true	normal	ブール型
name.value	ユーザー提供の永続サブスクリプションの名前	ctor	文字列
noLocal.value	この永続サブスクリプションの非ローカル・フラグ	ctor	ブール型
selector.value	この永続サブスクリプションの JMS メッセージ・セレクタ	ctor	文字列

D.4.10 JMS 宛先の統計

表 D-26 は、JMS 宛先の統計メトリックを示しています。

メトリック表のタイプは JMSDestinationStats です。

表 D-26 JMSDestinationStats メトリック

メトリック	説明	更新	単位
domain.value	宛先の JMS ドメイン（queue または topic）	ctor	文字列
name.value	宛先の構成名。jms.xml に定義されています。	ctor	文字列
locations.value	この宛先への JNDI 名のカンマ区切りリスト。jms.xml に定義されています。	ctor	文字列
メソッド名	この宛先オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

D.4.11 一時 JMS 宛先の統計

表 D-27 は、一時 JMS 宛先の統計を示しています。

メトリック表のタイプは JMSTempoaryDestinationStats です。

表 D-27 JMSTemporaryDestinationStats メトリック

メトリック	説明	更新	単位
connectionID.value	この一時宛先を作成した JMSConnectionStats インスタンスの ID	ctor	文字列
domain.value	宛先の JMS ドメイン（queue または topic など）	ctor	文字列
メソッド名	この宛先オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

D.4.12 JMS ストアの統計

表 D-28 は、JMS StoreStats メトリック表を示しています。

メトリック表のタイプは JMSStoreStats です。

表 D-28 JMSStoreStats メトリック

メトリック	説明	更新	単位
destination.value	このメッセージ・ストアに関連付けられた JMS 宛先の名前（書式を整えて出力される）	ctor	文字列
messageCount.value	このストアに含まれるメッセージの合計数	lazy	int 型
messageDequeued.count	メッセージ・デキューの合計数（処理済またはそれ以外）	normal	ops
messageDiscarded.count	エンキューのロールバック後に破棄されたメッセージの合計数	normal	ops
messageEnqueued.count	メッセージ・エンキューの合計数（処理済またはそれ以外）	normal	ops
messageExpired.count	期限切れメッセージの合計数	normal	ops
messagePagedIn.count	ページ・インされたメッセージ本体の合計数	normal	ops
messagePagedOut.count	ページ・アウトされたメッセージ本体の合計数	normal	ops
messageRecovered.count	永続性ファイルから、またはデキューのロールバック後に復元されたメッセージの合計数	normal	ops
pendingMessageCount.value	アクティブなトランザクションのエンキューまたはデキューのメッセージ部分の合計数	lazy	int 型
storeSize.value	メッセージ・ストアのバイト単位の合計サイズ	lazy	バイト
メソッド名	このメッセージ・ストア・オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

次の識別式が維持されます。

```
messageCount = messageRecovered + messageEnqueued - messageDequeued -
messageDiscarded - messageExpired
```

1 つのメッセージが同じトランザクション内でエンキューおよびデキューされる場合、messageEnqueued および messageDequeued イベントは発生しますが、messageRecovered および messageDiscarded イベントは発生しません。

D.4.13 JMS の永続性の統計

表 D-29 は、JMS の永続性の統計を示しています。

メトリック表のタイプは JMSPersistenceStats です。

表 D-29 JMSPersistenceStats メトリック

メトリック	説明	更新	単位
destination.value	この持続性ファイルに関連付けられた JMS 宛先の名前（書式を整えて出力される）	ctor	文字列
holePageCount.value	現在このファイル内で空いている 512 バイト・ページの数	normal	int 型
isOpen.value	永続性ファイル・ディスクリプタが現在オープンされている場合のみ true（LRU キャッシングの場合）	normal	ブール型
lastUsed.value	この永続性ファイルが最後に使用された時の System.currentTimeMillis()（LRU キャッシングの場合）	normal	msecs
persistenceFile.value	この永続宛先に使用される永続性ファイルの絶対パス名。この値は、OC4J が実行されているオペレーティング・システムによって異なります。	ctor	文字列

表 D-29 JMSPersistenceStats メトリック（続き）

メトリック	説明	更新	単位
usedPageCount.value	現在このファイル内で使用されている 512 バイト・ページの数	normal	int 型
メソッド名	この持続性ファイル・オブジェクトでの、すべての主要なメソッド・コールに対する時間隔タイマー・メトリック（PhaseEvent Sensor）	normal	

D.5 OC4J タスク・マネージャのメトリック

表 D-30 は、OC4J タスク・マネージャのメトリックを示しています。

メトリック表のタイプは oc4j_task です。

表 D-30 OC4J_taskManager メトリック

メトリック	説明	単位
interval.value	タスクの実行頻度。タスク・マネージャは、ラウンドロビン法ですべてのタスクを実行します。この時間隔が 0 の場合、タスク・マネージャは、タスクの選択時にラウンドロビン法でこのタスクを実行します。	msecs（ミリ秒）
run().active	アクティブなスレッドの数	スレッド
run().avg	タスク・マネージャがタスクを実行する平均時間	msecs
run().completed	タスク・マネージャがタスクを実行した回数	ops
run().maxActive	アクティブなタスクの最大数	スレッド
run().maxTime	タスクの実行に費やされた最大時間	msecs
run().minTime	タスクの実行に費やされた最小時間	msecs
run().time	タスク・マネージャの実行に費やされた合計時間	msecs

D.6 Java Object Cache（JOC）メトリック

表 D-31 は、トップレベルの Java Object Cache メトリックを示しています。

メトリック表のタイプは joc です。

注意： javacache.xml 構成ファイルの DMS 要素が true に設定されている場合のみ、JOC メトリックは参照可能です。

表 D-31 Java Object Cache（JOC）メトリック

メトリック	説明	単位
disk_Size.value	キャッシュ内でオブジェクトが消費するディスクの合計バイト数	バイト
memory_object_count.value	キャッシュ内のオブジェクトの合計数	バイト
memory_size.value	キャッシュ内でオブジェクトが消費するメモリーの合計バイト数	バイト
response_q_size.value	レスポンス・キューのサイズ	ops
task_count.value	非同期タスクの合計数	ops
time_q_size.value	時間キューのサイズ	ops
worker_thread_count.value	ワーカー・スレッドの合計数	スレッド

表 D-32 は、Java Object Cache のリージョン・メトリックを示しています。

メトリック表のタイプは java_cache_region です。

表 D-32 Java Cache のリージョン・メトリック

メトリック	説明	単位
disk_Count.value	ディスク上のリージョン内のオブジェクトの合計数	
disk_Size.value	リージョン内でオブジェクトが消費するディスクの合計バイト数	
disk_average_load_time.value	リージョン内のオブジェクトの平均ロード時間	
memory_average_load_time.value	リージョン内のオブジェクトの平均ロード時間	
memory_object_access_count.value	リージョン内のオブジェクトの合計アクセス回数	
memory_object_count.value	リージョン内のオブジェクトの合計数	
memory_size.value	リージョン内でオブジェクトが消費するメモリーの合計バイト数	

索引

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

あ

か

さ

た

な

は

ま

や

ら