BEA WebLogic Server™

パフォーマンス チューニング ガイド

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

BEA WebLogic Server パフォーマンス チューニング ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 24 日	BEA WebLogic Server バージョン 6.1

目次

3. WebLogic Server のチューニング

4. WebLogic Server アプリケーションのチューニング

A. 関連情報

索引

このマニュアルの内容

WebLogic Server™ プラットフォームで最高のパフォーマンスを実現するには、WebLogic Server 環境を構成するコンポーネントのパフォーマンスを最適化する必要があります。このマニュアルでは、パフォーマンスに関する以下の情報を提供します。

- 第 1 章「ハードウェア、オペレーティング システム、およびネットワーク パフォーマンスのチューニング」では、ハードウェア、オペレーティング システム、およびネットワークのパフォーマンスの問題について説明します。
- 第 2 章「Java 仮想マシン（JVM）のチューニング」では、JVM のチューニングに関する考慮事項について説明します。
- 第 3 章「WebLogic Server のチューニング」では、WebLogic Server をアプリケーションのニーズに合わせてチューニングする方法について説明します。
- 第 4 章「WebLogic Server アプリケーションのチューニング」では、アプリケーションのチューニングに関する考慮事項について説明します。
- 付録 A「関連情報」では、広範囲にわたるパフォーマンス関連の参照リストを提供します。

このマニュアルには索引も含まれています。

対象読者

このマニュアルは、WebLogic Server プラットフォームに含まれるコンポーネントのチューニング担当者を対象としています。サーバの管理、パフォーマンス チューニングの基礎概念、WebLogic Server プラットフォーム、XML、および Java プログラミングに読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックします。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は Adobe の Web サイトで無料で入手できます。

サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@bea.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェアの名前とバージョン、およびドキュメントのタイトルと日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号、会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+ 〔Tab〕	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例 :
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

斜体の等幅テキスト	コード内の変数を示す。 例 : `String CustomerName;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。

[ ]	構文内の任意指定の項目を示す。
	例 :
	`java utils.MulticastTest -n name -a address`
	`[-p portnumber] [-t timeout] [-s send]`

表記法	適用

|	構文の中で相互に排他的な選択肢を区切る。
	例 :
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

. . .	コマンドラインで以下のいずれかを示す。 ♦ 引数を複数回繰り返すことができる。 ♦ 任意指定の引数が省略されている。 ♦ パラメータや値などの情報を追加入力できる。
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 ハードウェア、オペレーティング システム、およびネットワーク パフォーマンスのチューニング

以下の節では、ハードウェア、オペレーティング システム、およびネットワーク パフォーマンスの最適化に関する考慮事項について説明します。

ハードウェアのチューニング

パフォーマンスを調べる際は、まずハードウェアの容量とコンフィグレーションを考慮してください。所定のハードウェア コンフィグレーションで WebLogic Server と所定のアプリケーションをサポートするために必要になる容量には、いくつもの要因が影響します。

表 1-1 は、ハードウェアのチューニング、および標準ベンチマークとメトリックに関する情報へのリンクを示しています。

表 1-1 プラットフォーム別ハードウェアのチューニング情報

対象事項	詳細情報の参照先
「プラットフォーム サポート」ページ	「プラットフォーム サポート」ページは頻繁に更新されており、さまざまなプラットフォームに関する最新の動作確認情報が記載されている。
Solaris	BEA WebLogic Server および Solaris 固有の詳細情報については、「プラットフォーム サポート」ページの「SPARC 上の Sun Microsystems Solaris」を参照。 A-2 ページの「Sun Microsystems の情報」も参照。
Hewlett-Packard	BEA WebLogic Server および HP-UX 固有の詳細情報については、「プラットフォーム サポート」ページの「WebLogic HP-UX プラットフォーム サポート」を参照。 A-3 ページの「Hewlett-Packard Company の情報」も参照。
標準ベンチマークとメトリック	Standard Performance Evaluation Corporation は、コンピュータ システムのパフォーマンスを評価するための標準ベンチマークおよびメトリックを提供している。

オペレーティング システムのチューニング

オペレーティング システムのチューニングは、使用しているオペレーティング システムのマニュアルに従って行ってください。BEA では、複数のオペレーティング システム上で WebLogic Server の動作確認をしています。表 1-2 は、オペレーティング システムのチューニングに関する詳細情報へのリンクを示しています。

表 1-2 オペレーティング システムの考慮事項

対象事項	詳細情報の参照先
「プラットフォーム サポート」ページ	「プラットフォーム サポート」ページは頻繁に更新されており、さまざまなプラットフォームに関する最新の動作確認情報が記載されている。
ファイル記述子	UNIX プラットフォームでは、サーバへの各ソケット接続でファイル記述子が使用される。このため、オペレーティング システムをコンフィグレーションして、適切な数のファイル記述子を用意しておく必要がある。 「プラットフォーム サポート」ページの「SPARC 上の Sun Microsystems Solaris」の「Solaris ファイル ディスクリプタ制限のチューニング」を参照。
Solaris TCP チューニング パラメータ	「プラットフォーム サポート」ページの「SPARC 上の Sun Microsystems Solaris」の「チューニング可能な Solaris パラメータの設定」を参照。
ユーザ プロセス用の最大メモリ	ユーザ プロセス用に使用できる最大メモリについては、オペレーティング システムのマニュアルを確認すること。オペレーティング システムによっては、この値が 128 MB しかないものもある。詳細については、オペレーティング システムのマニュアルを参照。 メモリ管理の詳細については、第 2 章「Java 仮想マシン（JVM）のチューニング」を参照。

表 1-2 オペレーティング システムの考慮事項（続き）

対象事項	詳細情報の参照先
ネイティブ I/O を使用した静的ファイルの提供（Windows のみ）	Windows NT/2000 上で WebLogic Server を実行する場合、WebLogic Server で Java メソッドを使用する代わりにネイティブ オペレーティング システム呼び出し `TransmitFile` を使用して、HTML ファイル、テキスト ファイル、および画像ファイルなどの静的ファイルを提供することができる。ネイティブ I/O を使用すると、サイズの大きな静的ファイルを提供するときのパフォーマンスが向上する。 詳細については、「静的ファイルを提供するネイティブ I/O の使用」を参照。

ネットワークのパフォーマンス

ネットワークのパフォーマンスが低下するのは、ネットワーク リソースの供給が需要に追いつくことができない場合です。表 1-3 は、考慮すべき事項を示しています。

表 1-3 ネットワーク コンフィグレーションの考慮事項

対象事項	考慮事項
ネットワークのハードウェアおよびソフトウェア	1 つまたは複数のネットワーク コンポーネント（ハードウェアまたはソフトウェア）に問題がある場合は、ネットワーク管理者との共同作業でその問題を分離して取り除く必要がある。
ネットワークの帯域幅	WebLogic Server 用に、適切な広さのネットワーク帯域幅、およびアーキテクチャ内の他の層への適切な数の接続（クライアント接続やデータベース接続など）を用意する必要がある。 1-5 ページの「ネットワーク帯域幅の決定」を参照。

表 1-3 ネットワーク コンフィグレーションの考慮事項（続き）

対象事項	考慮事項
LAN インフラストラクチャ	ローカル エリア ネットワークには、アプリケーションのピーク容量を処理できるだけの速度が必要になる。 ネットワーク トラフィックが常に使用可能リソースの容量を上回っている場合（Web サーバのヒット率が最大値に達し、システムが 100% ビジーの場合など）、以下のいずれかの処理を行う必要がある。 ■ ネットワークを再設計し、負荷を再分散する。 ■ ネットワーク クライアントの数を減らす。 ■ ネットワークの負荷を処理するシステムの数を増やす。

ネットワーク帯域幅の決定

WebLogic Server マシンにすべてのクライアント接続を処理できるだけの十分な帯域幅を用意することは重要です。プログラムに基づくクライアントの場合、クライアント JVM ごとにサーバへの 1 つのソケットがあります。各ソケットには帯域幅が必要です。WebLogic Server でプログラムに基づくクライアントを処理するには、同様な Web サーバで処理する場合の 125 ～ 150% の帯域幅が必要になります。

Web サーバの実行に必要な帯域幅を決定するには、56k/s の帯域幅ごとに、配信するコンテンツにサイズに応じて 7 ～ 10 の要求を同時に処理できると仮定します。

HTTP クライアントのみを処理する場合は、静的ページを提供する Web サーバと同程度の帯域幅が必要になると考えてください。

所定のデプロイメントに十分な帯域幅があるかどうかを調べるには、使用しているネットワーク オペレーティング システムのベンダから提供されているネットワーク モニタ ツールを使用して、ネットワーク システム上の負荷を参照します。負荷が非常に高い場合は、帯域幅がシステムの問題点になっている可能性があります。

2 Java 仮想マシン（JVM）のチューニング

Java 仮想マシン（JVM）は、マイクロプロセッサ上で Java クラス ファイルのバイト コードを実行する「実行エンジン」です。JVM のチューニングは、WebLogic Server とアプリケーションのパフォーマンスに影響を与えます。

以下の節では、JVM のチューニングの考慮事項について説明します。

関連情報へのリンクについては、A-5 ページの「Java 仮想マシン（JVM）の情報」を参照してください。

JVM のチューニングの考慮事項

表 2-1 は、JVM のチューニングに関する一般的な考慮事項を示しています。

表 2-1 JVM のチューニングの一般的な考慮事項

対象事項	説明
JVM のベンダおよびバージョン	WebLogic Server の動作が保証されているプロダクション JVM だけを使用すること。WebLogic Server 6.x は、Java 1.3 準拠の JVM のみサポートする。 「プラットフォーム サポート」ページは頻繁に更新されており、さまざまなプラットフォームに関する最新の動作確認情報が記載されている。
ヒープ サイズおよびガベージ コレクションのチューニング	チューニングの詳細については、2-3 ページの「JVM ヒープ サイズについて」を参照。 ガベージ コレクションのチューニングの概要については、「Tuning Garbage Collection with the 1.3.1 Java Virtual Machine」を参照。
世代別ガベージ コレクション	2-4 ページの「世代別ガベージ コレクションについて」を参照。
クライアント / サーバ JVM の混在	WebLogic Server では、クライアントとサーバ用に異なる JVM バージョンを使用したデプロイメントがサポートされている。クライアント / サーバ JVM の混在のサポートについては、「クライアント / サーバ混在 JVM のサポート」を参照。
UNIX スレッディング モデル	UNIX には、グリーン スレッドとネイティブ スレッドという 2 つのスレッディング モデルがある。WebLogic Server で最高のパフォーマンスとスケーラビリティを得るには、ネイティブ スレッドを使用する JVM を選択する。 Solaris を使用する場合は、JavaSoft の Web サイトにある「Threading Models and Solaris Versions Supported」を参照。

表 2-1 JVM のチューニングの一般的な考慮事項（続き）

対象事項	説明
Just-In-Time（JIT）JVM	WebLogic Server を実行するときは、JIT コンパイラを使用する。Sun Microsystems や Symantec などから提供されるほとんどの JVM では、JIT コンパイラが使用される。 詳細については、JVM サプライヤのドキュメントを参照。 **注意：** Sun の JVM 1.3.x、JIT オプションは有効ではなくなった。A-5 ページの「Java 仮想マシン（JVM）の情報」を参照。

JVM ヒープ サイズについて

ガベージ コレクションは、Java ヒープ内の使用されていない Java オブジェクトを解放する VM プロセスです。Java ヒープは Java プログラムのオブジェクトが存在している場所であり、ライブ オブジェクト、デッド オブジェクト、およびフリー メモリのリポジトリです。実行中のプログラムでどのポインタからもアクセスされなくなると、オブジェクトはガベージ（廃棄物）になります。

JVM ヒープ サイズによって、ガベージ コレクションを行う頻度とその時間が決定されます。ガベージ コレクションの適切な実行頻度はアプリケーションによって異なるので、ガベージ コレクションの実際の時間と頻度を解析して調整する必要があります。

大きいヒープ サイズを設定した場合、ガベージ コレクション全体は低速化しますが、実行頻度が低くなります。メモリのニーズに合わせてヒープ サイズを設定した場合、ガベージ コレクション全体は高速化しますが、実行頻度が高くなります。

ヒープ サイズのチューニングの目標は、所定の時間に処理できるクライアントの数を最大限に増やしつつ、ガベージ コレクションの実行時間を最小限に抑えることです。

ベンチマーク時に最高のパフォーマンスを保証するには、大きいヒープ サイズ値を設定し、ベンチマークの実行中にガベージ コレクションが実行されないようにします。

ヒープ領域が不足している場合、次の Java エラーが表示されます。

```
java.lang.OutOfMemoryError <<no stack trace available>>
java.lang.OutOfMemoryError <<no stack trace available>>
Exception in thread "main"
```

ヒープ領域値を変更するには、2-8 ページの「ヒープ サイズ値の指定」を参照してください。

世代別ガベージ コレクションについて

Java HotSpot JVM 1.3 では、世代別ガベージ コレクションが使用されます。ネイティブ ガベージ コレクションではヒープ内のすべてのライブ オブジェクトが調べられますが、世代別ガベージ コレクションでは余分な作業を省くため、オブジェクトの有効期間が考慮されます。

ヒープは、New、Old という 2 つの世代領域に分割されます。New 世代領域は、さらに Eden と 2 つのサバイバル領域に細分化されます。Eden は、新しいオブジェクトが割り当てられる領域です。ガベージ コレクションが実行されると、Eden 内のライブ オブジェクトは隣のサバイバル領域にコピーされます。オブジェクトは最大しきい値を超過するまで、このようにサバイバル領域間でコピーされ、その後、New 領域から Old 領域に移動されます。New 世代領域および Old 世代領域のサイズと比率の指定については、2-8 ページの「ヒープ サイズ値の指定」を参照してください。

多くのオブジェクトは、割り当てられた直後にガベージになります。このようなオブジェクトは、「初期廃棄」オブジェクトと呼ばれます。オブジェクトの有効期間が長くなるほど、ガベージ コレクションの実行回数が増え、ガベージ コレクションは低速化します。アプリケーションがオブジェクトを作成および解放する頻度によって、ガベージ コレクションの実行頻度が決まります。可能であれば、新しいオブジェクトを作成するよりも、オブジェクトをキャッシュして再利用するようにしてください。

オブジェクトの大半の有効期間が短いことを理解すれば、ガベージ コレクションを効率化できるようチューニングできます。世代単位でメモリ管理を行う場合、複数のメモリ プールを作成して世代の異なるオブジェクトを保持します。

ガベージ コレクションは世代ごとにプールがいっぱいになると実行されます。オブジェクトの大半の有効期間を 1 回のコレクションより短くすると、ガベージ コレクションを効率化できます。世代のサイズが小さいと、ガベージ コレクションの実行頻度が増加し、パフォーマンスが低下します。

世代別ガベージ コレクションのチューニングの概要については、「Tuning Garbage Collection with the 1.3.1 Java Virtual Machine」を参照してください。

ヒープ サイズの決定

この節では、最も効率的なヒープ サイズを決定する基本手順について説明します。

1. アプリケーション実行中に、最大の負荷をかけた状態で WebLogic Server のパフォーマンスをモニタします。
2. `-verbosegc` オプションを使用して、ガベージ コレクションに投入される時間とリソースを正確に測定します。
3. Java VM の verbose ガベージ コレクション出力を有効にし、標準エラーおよび標準出力の両方をログ ファイルにリダイレクトします。

   2-7 ページの「verbose ガベージ コレクションの有効化と出力のリダイレクト」を参照してください。
4. 以下について解析します。
   a. ガベージ コレクションの実行頻度。`weblogic.log` ファイルで、ガベージ コレクション前後のタイム スタンプを比較します。
   b. ガベージ コレクションの実行時間。ガベージ コレクション全体の実行時間は 3 ～ 5 秒を上回ってはいけません。
   c. 平均メモリ占有量。つまり、各ガベージ コレクション実行後のヒープの状態です。ヒープの空きが常に 85% になる場合、ヒープ サイズを小さくしてもかまいません。
5. Java HotSpot JVM 1.3 を使用している場合は、世代のサイズを設定します。

   2-8 ページの「ヒープ サイズ値の指定」

6. ヒープ サイズがシステムの使用可能な空き RAM より大きくならないようにします。

   ページがディスクに「スワップ」されない範囲で、できるだけ大きいヒープ サイズを設定します。システムの空き RAM の容量は、ハードウェアのコンフィグレーションと、そのマシン上で実行されているプロセスのメモリ要件によって異なります。システムの空き RAM の容量の決定については、システム管理者に問い合わせてください。

7. システムがガベージ コレクションに費やす時間が長すぎる場合（割り当てられた「仮想」メモリを RAM が処理できない場合）、ヒープ サイズを小さくします。

   通常、使用可能な RAM（オペレーティング システムまたはその他のプロセスによって占有されない RAM）の 80% を JVM に割り当てます。

8. 残りの RAM の容量が大きい場合は、そのマシンでより多くの WebLogic Server を実行します。

2-8 ページの「ヒープ サイズ値の指定」も参照してください。

verbose ガベージ コレクションの有効化と出力のリダイレクト

この節では、診断のために verbose ガベージ コレクションを有効にし、出力をログ ファイルにリダイレクトする方法について説明します。

1. 次の手順の例で示すように、WebLogic Server を起動したときに、Java VM の verbose ガベージ コレクション出力を有効にします。
2. 標準エラーと標準出力の両方をログ ファイルにリダイレクトします。

   リダイレクトすることにより、スレッド ダンプ情報が WebLogic Server の情報メッセージとエラー メッセージに関連してログに記録されるので、診断する際にログ ファイルがさらに役立ちます。

   次に例を示します。

   ```
   % java -ms64m -mx64m -verbosegc -classpath $CLASSPATH
   -Dweblogic.domain=mydomain -Dweblogic.Name=clusterServer1
   -Djava.security.policy==/bea/weblogic6x/lib/weblogic.policy
   -Dweblogic.management.server=192.168.0.101:7001
   -Dweblogic.management.username=system
   -Dweblogic.management.password=systemPassword weblogic.Server
   >> logfile.txt
   ```

HPUX では、次のオプションを使用して `stderr` `stdout` を 1 つのファイルにリダイレクトします。

```
-Xverbosegc:file=/tmp/gc$$.out
```

ここで `$$` は、Java プロセスの PID にマップされます。出力にはガベージ コレクションの実行時間を示すタイムスタンプが含まれているので、ガベージ コレクションの実行頻度を推測できます。

Solaris では、次のコマンドを実行します。

```
weblogic.Server > server.out 2>&1
```

ヒープ サイズ値の指定

Java ヒープ サイズの値は、Java コマンドラインから WebLogic 管理サーバを起動するときに指定できます。次に例を示します。

```
$ java ... -XX:NewSize=128m -XX:MaxNewSize=128m -XX:SurvivorRatio=8
-Xms512m -Xmx512m
```

これらの値のデフォルト サイズはバイト単位で指定されます。KB（キロバイト）を示すには、値に「k」または「K」を付加します。同様に、MB（メガバイト）を示すには「m」または「M」を、GB（ギガバイト）を示すには「g」または「G」を付加します。

これらのパラメータは、WebLogic Server の起動スクリプトおよび環境スクリプトの影響を受けることに注意する必要があります。WebLogic 配布キットには、デフォルト サーバを起動するサンプル スクリプト、およびサーバを構築したり実行したりするための環境を設定するサンプル スクリプトが付属しています。

- `startWebLogic.cmd` および `setEnv.cmd`（Windows システム用）
- `startWebLogic.sh` および `setEnv.sh`（UNIX システム用）

それらのスクリプトは、実際の環境やアプリケーションに合わせて修正する必要があります。「WebLogic Server の起動と停止」を参照してください。

Java ヒープ サイズのオプション

最高のパフォーマンスを得るには、アプリケーションを個別にチューニングします。また、表 2-2 に示されている JVM ヒープ サイズのオプションをコンフィグレーションすると、ほとんどのアプリケーションでパフォーマンスが向上します。

表 2-2 に示されているオプションは、使用するアーキテクチャおよびオペレーティング システムによって異なります。プラットフォーム別の JVM チューニング オプションについては、ベンダのマニュアルを参照してください。

表 2-2 Java ヒープ サイズのオプション

タスク	オプション	説明
New 世代領域のヒープ サイズを設定する。	`-XX:NewSize`	このオプションを使用すると、New 世代領域の Java ヒープ サイズを設定できる。この値は、1MB より大きい 1024 の倍数に設定する。一般に `-XX:NewSize` は、最大ヒープ サイズの 4 分の 1 のサイズになるように設定する。有効期間の短いオブジェクトが多い場合ほど、このオプションの値を大きくする。 プロセッサ数が増加するほど、New 世代領域を大きく設定する必要がある。メモリの割り当ては並列処理できるが、ガベージ コレクションは並列処理されない。
New 世代領域の最大ヒープ サイズを設定する。	`-XX:MaxNewSize`	このオプションを使用すると、New 世代領域の最大 Java ヒープ サイズを設定できる。この値は、1MB より大きい 1024 の倍数に設定する。
New 世代領域のヒープ サイズ比率を設定する。	`-XX:SurvivorRatio`	New 世代領域は、3 つのサブ領域、つまり Eden と、同じサイズの 2 つのサバイバル領域に分割される。 `-XX:SurvivorRatio=X` オプションを使用すると、Eden とサバイバル領域のサイズ比率をコンフィグレーションできる。この値は 8 に設定し、その後、ガベージ コレクションをモニタするようにする。

表 2-2 Java ヒープ サイズのオプション（続き）

タスク	オプション	説明
最小ヒープ サイズを設定する。	-Xms	このオプションを使用すると、メモリ割り当てプールの最小サイズを設定できる。この値は、1MB より大きい 1024 の倍数に設定する。一般に、最小ヒープ サイズ（-Xms）は最大ヒープ サイズ（-Xmx）と同じ大きさに設定する。
最大ヒープ サイズを設定する。	-Xmx	このオプションを使用すると、最大 Java ヒープ サイズを設定できる。この値は、1MB より大きい 1024 の倍数に設定する。

ガベージ コレクションの強制

サーバでガベージ コレクションを強制する前に、完全なガベージ コレクションが必要であることを確認してください。ガベージ コレクションを強制すると、JVM ではヒープ内のすべてのライブ オブジェクトを頻繁に調べます。

Administration Console を使用して、指定したサーバでガベージ コレクションを強制するには、次の手順に従います。

1. Administration Console の左ペインで、メモリ使用状況を表示させるサーバのサーバ インスタンス ノードをクリックします。このインスタンスと関連付けられているタブを示すダイアログが右ペインに表示されます。
2. [モニタ] タブをクリックします。
3. [パフォーマンス] タブをクリックします。
4. [メモリ使用状況] グラフで使用率が高くなっているかどうかを確認します。

   [メモリ使用状況] グラフには、実行中のサーバの使用状況のみが表示されます。
5. ガベージ コレクションを強制するには、[ガベージコレクションを強制する] をクリックします。

   コレクション処理が正常に終了したことを示すメッセージが表示されます。

Java HotSpot VM オプションの設定

標準の Java オプションを使用して、パフォーマンスを向上させることができます。これらのオプションの使い方は、使用するアプリケーションのコーディングによって異なります。コマンドライン オプションはプラットフォーム間で一貫性がありますが、プラットフォームによってはデフォルトが異なる場合もあります。

クライアント JVM とサーバ JVM の両方をテストして、特定のアプリケーションに対して、よりパフォーマンスの優れた JVM を調べる必要があります。

パフォーマンスを向上させる、これ以外の VM オプションについては、2-12 ページの「非標準 Java コマンド ライン オプションの設定」を参照してください。

NT 用の標準オプション

NT の場合、WebLogic Server では Java コマンドを使用して JVM を起動できます。表 2-3 に示されているオプションを使用します。

表 2-3 NT 上の HotSpot VM 用の標準オプション

オプション	説明
`-hotspot`	HotSpot クライアント VM を選択する。
`-server`	HotSpot サーバ VM を選択する。
`-classic`	クラシック VM を選択する。

UNIX 用の標準オプション

UNIX の場合、WebLogic Server では Java コマンドを使用して JVM を起動できます。表 2-4 に示されているオプションを使用します。

表 2-4 UNIX 上の HotSpot VM 用の標準オプション

オプション	説明
-client または -hotspot	HotSpot クライアント VM を選択する。
-server	HotSpot サーバ VM を選択する。

非標準 Java コマンド ライン オプションの設定

非標準の Java オプションを使用して、パフォーマンスを向上させることができます。これらのオプションの使い方は、使用するアプリケーションのコーディングによって異なります。コマンドライン オプションはプラットフォーム間で一貫性がありますが、プラットフォームによってはデフォルトが異なる場合もあります。

NT 用の非標準オプション

表 2-5 に、NT 上の HotSpot VM でパフォーマンスを向上させる非標準オプションの例をいくつか示します。

表 2-5 NT 上の HotSpot VM 用の非標準オプション

オプション	説明
`-Xnoclassgc`	このオプションは、クラスのガベージ コレクションを無効にして、そのクラスへのすべての参照が失われた後にそのクラスが参照されたときに、そのクラスの再ロードを防ぐ。このオプションでは、ヒープ サイズを大きくする必要がある。
`-oss`	このオプションは、Java スレッドのスタック サイズを設定する。大きすぎる値（>2MB）を設定すると、パフォーマンスが大幅に低下する。
`-ss`	このオプションは、ネイティブ スレッドのスタック サイズを設定する。大きすぎる値（>2MB）を設定すると、パフォーマンスが大幅に低下する。
`-Xverbosegc:file=/tmp/gc$$.out`	このオプションは、`-verbosegc` メッセージをファイルにリダイレクトして、ガベージ コレクション メッセージを `stderr` の残りのメッセージから分離する。 また、このオプションを使用すると、パフォーマンスが向上する。これは、ファイルへの書き込みのバッファ処理が、`stderr` などの文字ストリームへの書き込みのバッファ処理より高速であるため。 2-7 ページの「verbose ガベージ コレクションの有効化と出力のリダイレクト」も参照。

Solaris 用の非標準オプション

「Solaris VM 用の非標準オプション」を参照してください。

3 WebLogic Server のチューニング

以下の節では、WebLogic Server をアプリケーションのニーズに合わせてチューニングする方法について説明します。

config.xml ファイルの要素のチューニング

表 3-1 は、サーバのパフォーマンスに影響を与える `config.xml` ファイルの要素を示しています。

表 3-1 パフォーマンス関連の config.xml 要素

要素	属性	詳細情報の参照先
`Server`	`NativeIOEnabled`	3-2 ページの「WebLogic Server パフォーマンス パックの使い方」を参照。
`ExecuteQueue`	`ThreadCount`	3-3 ページの「スレッド数の設定」を参照。
`Server`	`ThreadPoolPercentSocketReaders`	3-6 ページの「ソケットリーダーとしてのスレッドの割り当て」を参照。

表 3-1 パフォーマンス関連の config.xml 要素（続き）

要素	属性	詳細情報の参照先
Server	AcceptBacklog	3-8 ページの「接続バックログのバッファリングのチューニング」を参照。
JDBCConnectionPool	InitialCapacity MaxCapacity	3-7 ページの「JDBC 接続プール サイズのチューニング」を参照。

WebLogic Server パフォーマンス パックの使い方

ベンチマークの結果、プラットフォームに対応したパフォーマンス パックを使用すると、WebLogic Server のパフォーマンスが大幅に向上することが実証されています。パフォーマンス パックは、プラットフォーム用に最適化された（ネイティブ）ソケット マルチプレクサを使用してサーバのパフォーマンスを向上させます。

パフォーマンス パックを使用するには、config.xml ファイルの Server 要素の NativeIOEnabled 属性を定義する必要があります。配布キットに付属のデフォルト config.xml ファイルでは、この属性はデフォルトで有効（NativeIOEnabled=true）になっています。

パフォーマンス パックを使用できるプラットフォーム

現時点でパフォーマンス パックを使用できるプラットフォームを確認するには、次の手順を行います。

1. 「プラットフォーム サポート」ページにアクセスします。
2. [編集 | このページの検索（ページ内を検索）] を選択し、「パフォーマンス パックを含む」という文字列があるすべてのプラットフォームを確認します。

パフォーマンス パックの有効化

Administration Console を使用してパフォーマンス パックが有効化されていることを確認するには、次の手順を行います。

1. WebLogic Server Console を起動します。
2. ナビゲーション ツリーで [サーバ] フォルダを開きます。
3. [サーバ] フォルダ内のサーバ（デフォルト インストールの `myserver`）を選択します。
4. [コンフィグレーション] タブを選択します。
5. [チューニング] タブを選択します。
6. [ネイティブ IO を有効化] チェックボックスがチェックされていない場合はチェックします。
7. [適用] をクリックします。
8. サーバを再起動します。

スレッド数の設定

`config.xml` ファイルの `ExecuteQueue` 要素の `ThreadCount` 属性の値は、実行キューを使用するアプリケーションで実行可能な同時処理の数と同じです。処理が WebLogic Server インスタンスに入ると、その処理は実行キューに置かれます。次に、この処理は 1 つのスレッドに割り当てられて実行されます。スレッドはリソースを消費するため、この属性は注意して取り扱う必要があります。値を不必要に大きくすると、パフォーマンスが低下する可能性があります。

デフォルトでは、新しい WebLogic Server インスタンスは、「default」という名前のデフォルト実行キューにコンフィグレーションされています。この実行キューのスレッド数は 15 です。また、WebLogic Server には、`__weblogic_admin_html_queue` と `__weblogic_admin_rmi_queue` という 2 つの組み込み実行キューがあります。ただし、これらのキューは、WebLogic Administration Console との通信用に予約されています。追加の実行キューをコンフィグレーションしない場合、デフォルト キューは、すべての Web アプリケーションおよび RMI オブジェクトによって使用されます。

注意： ほとんどのアプリケーションでは、デフォルト値を変更する必要はありません。

スレッド数の変更

デフォルト実行キューにスレッドを増やしたからといって、必ずしもより多くの作業を処理できるわけではありません。スレッドを増やした場合でも、依然としてプロセッサの処理能力による制約を受けます。`ThreadCount` 属性の値を不必要に大きくすると、パフォーマンスが低下する可能性があります。また、スレッドはメモリを消費するため、実行スレッドを大きくすると、メモリの使用量が大きくなり、コンテキストの切り替えが増加し、パフォーマンスが低下する可能性があります。

`ThreadCount` 属性の値は、アプリケーションが実行する処理のタイプによって大きく異なります。たとえば、使用しているクライアント アプリケーションが軽量で、その多くの処理をリモート呼び出しを介して行う場合を考えます。そのクライアント アプリケーションが接続に費やす時間は、多くの処理をクライアントサイドで行うクライアント アプリケーションより長くなります。

処理に対してスレッドの追加が不要な場合は、この属性の値を変更しないようにします。クライアント アプリケーションでは、スレッドは保持されません。

アプリケーションが応答に時間のかかるデータベース呼び出しを行う場合、応答が早く時間のかからない呼び出しを行うアプリケーションより多くの実行スレッドが必要となります。後者の場合は、少ない実行スレッドを使用してパフォーマンスを向上させることができます。

スレッド数のシナリオ

表 3-2 に、デフォルト実行キューで使用可能なスレッドを調整するシナリオをいくつか示します。これらのシナリオでは、すべてのスレッド要求がデフォルト実行キューを使用して満たされていることを前提としています。追加の実行キューをコンフィグレーションして、特定のキューにアプリケーションを割り当てる場合は、結果をプールごとにモニタする必要があります。

表 3-2 スレッド数のシナリオ

条件	結果	対策
スレッド数 < CPU の数	CPU の使用率が低下する。	スレッド数を増加させる。

表 3-2 スレッド数のシナリオ（続き）

条件	結果	対策
（スレッド数 == CPU の数）	理論的には理想的だが、CPU の使用率が低い。	スレッド数を増加させる。
（スレッド数 > CPU の数）で、スレッド数が適度に多い場合	実用上は理想的であり、適度なコンテキストの切り替え数と高い CPU 使用率が実現される。	スレッド数をさらにチューニングして、パフォーマンスの結果を比較する。
（スレッド数 > CPU の数）で、スレッド数が多すぎる場合	コンテキストの切り替えが増えすぎて、パフォーマンスが大幅に低下する。	スレッド数を減らす。 たとえば、プロセッサが 4 個の場合、4 つのスレッドを同時に実行できる。このため、実行スレッド数を、4 +（ブロックされるスレッドの数）にすることができる。 スレッド数は、アプリケーションによって大きく異なる。たとえば、アプリケーションがスレッドをブロックする時間によっては、上の公式が当てはまらない場合もある。

症状 : スレッド数が少なすぎる

スレッド数が少なすぎる場合、WebLogic Server が最大負荷で動作しているときに以下の症状が現れます。

- 実行できる処理があるのに、CPU が処理を待機している。
- CPU の使用率が 100% には決してならない。

症状 : スレッド数が多すぎる

WebLogic Server が最大負荷で動作しているときにスレッド数が多すぎる場合は、スレッド数を減らすにつれてパフォーマンスが向上します。

実行キューへのアプリケーションの割り当て

デフォルト実行キューをコンフィグレーションして、すべての WebLogic Server アプリケーションに対して最適な数のスレッドを提供することができます。また、複数の実行キューをコンフィグレーションすると、重要なアプリケーションをより詳細に制御することができます。複数の実行キューを使用することにより、WebLogic Server の負荷に関係なく、選択したアプリケーションが固定数の実行スレッドに確実にアクセスできるようになります。コンフィグレーションされた実行キューへのアプリケーションの割り当てに関する詳細については、4-7 ページの「実行キューによるスレッド使用の制御」を参照してください。

ソケット リーダーとしてのスレッドの割り当て

ソケットからメッセージを読み込む実行スレッドの最高割合を設定するには、`config.xml` ファイルの `ThreadPoolPercentSocketReaders` 属性を使用します。この属性の最適値は、アプリケーションによって異なります。デフォルト値は 33、有効範囲は 1 ～ 99 です。

`ThreadPoolPercentSocketReaders` 属性は、ソケットからメッセージを読み込む実行スレッドの最高割合を設定します。実行スレッドをソケット リーダー スレッドとして割り当てると、サーバがクライアント要求を受け入れる速度と能力が向上します。重要なのは、ソケットからメッセージを読み込む実行スレッドの数と、サーバでタスクを実際に実行するスレッド数のバランスを取ることです。

ネイティブ パフォーマンス パックを使用していない場合は、実行スレッドをソケット リーダー スレッドとして割り当てる必要があります。可能な場合は、使用しているプラットフォーム用のパフォーマンス パックを使用します。3-2 ページの「WebLogic Server パフォーマンス パックの使い方」を参照してください。

接続プールによるパフォーマンスの向上

DBMS への JDBC 接続の確立には非常に時間がかかる場合があります。JDBC アプリケーションでデータベース接続のオープンとクローズを繰り返す必要がある場合、これは重大なパフォーマンスの問題となります。WebLogic 接続プールは、こうした問題を効率的に解決します。

WebLogic Server を起動すると、接続プール内の接続が開き、すべてのクライアントが使用できるようになります。クライアントが接続プールの接続をクローズすると、その接続はプールに戻され、他のクライアントが使用できる状態になります。つまり、接続そのものはクローズされません。プール接続のオープンとクローズには、ほとんど負荷がかかりません。

どのくらいの数の接続をプールに作成すればよいでしょうか。接続プールの数は、コンフィグレーションされたパラメータに従って最大数と最小数の間で増減させることができます。最高のパフォーマンスが得られるのは、同時ユーザと同じくらいの数の接続が接続プールに存在する場合です。

JDBC 接続プール サイズのチューニング

`JDBCConnectionPool` 要素の `InitialCapacity` 属性を使用すると、プールのコンフィグレーション時に作成する物理的なデータベース接続の数を設定できます。ここで指定した数の接続をサーバで作成できない場合、この接続プールの作成は失敗します。3-7 ページの「JDBC 接続プールの初期サイズのチューニング」を参照してください。

`JDBCConnectionPool` 要素の `MaxCapacity` 属性を使用すると、接続プールの物理的なデータベース接続の最大数を設定できます。3-8 ページの「JDBC 接続プールの最大サイズのチューニング」を参照してください。

JDBC チューニングの詳細については、『WebLogic JDBC プログラミング ガイド』の「JDBC アプリケーションのパフォーマンス チューニング」を参照してください。

JDBC 接続プールの初期サイズのチューニング

開発時は、`InitialCapacity` 属性の値を小さく設定すると便利です。これにより、サーバの起動が速くなります。

プロダクション システムでは、サーバの起動時にすべてのデータベース接続を取得できるよう、`InitialCapacity` の値は `MaxCapacity` の値と同じに設定するようにしてください。

InitialCapacity の値が MaxCapacity の値より小さい場合は、負荷が増加すると、サーバで追加のデータベース接続を作成しなくてはならなくなります。サーバに負荷がかかると、できるだけ速く要求を完了するためにすべてのリソースが処理に費やされることになり、新しいデータベース接続を作成できなくなります。

JDBC 接続プールの最大サイズのチューニング

JDBCConnectionPool 要素の MaxCapacity 属性を使用すると、接続プールが保持できる物理的なデータベース接続の最大数を設定できます。JDBC ドライバおよびデータベース サーバによっては、可能な物理的接続の数が制限されている場合もあります。

プロダクション システムでは、プール内の接続数を、JDBC 接続を必要とする並行クライアント セッションの数と同じにすることをお勧めします。プールのサイズは、サーバ内の実行スレッドの数とは無関係です。実行中のユーザ セッションが実行スレッドより多くなる場合もあります。

接続バックログのバッファリングのチューニング

config.xml ファイルの Server 要素の AcceptBacklog 属性を使用すると、サーバが受け入れる接続要求の数を設定できます（これ以上の要求は拒否されます）。AcceptBacklog 属性は、待機キューに格納できる TCP 接続の数を指定します。この固定サイズのキューには、TCP スタックでは受信されたが、アプリケーションにはまだ受け入れられていない接続要求が格納されます。デフォルト値は 50 で、最大値はオペレーティング システムによって異なります。

Administration Console で、[サーバ｜コンフィグレーション｜チューニング] を選択し、[バックログを受け入れ] 属性の値を入力します。

処理中に、クライアントで多くの接続が削除または拒否され、サーバに他のエラー メッセージが存在しない場合は、AcceptBacklog 属性の値が小さすぎる可能性があります。

WebLogic Server にアクセスしようとしたときに「connection refused（接続が拒否されました）」というメッセージを受け取った場合は、AcceptBacklog 属性の値をデフォルトから 25% 大きくします。メッセージが表示されなくなるまで、この属性の値を 25% ずつ大きくしていきます。

weblogic-ejb-jar.xml ファイルの要素のチューニング

表 3-3 は、パフォーマンスに影響を与える weblogic-ejb-jar.xml ファイルの要素を示しています。

表 3-3 パフォーマンス関連の weblogic-ejb-jar.xml 要素

要素	詳細情報の参照先
max-beans-in-free-pool	3-9 ページの「EJB プール サイズの設定」を参照。
initial-beans-in-free-pool	3-11 ページの「フリー プール内の初期 Bean のチューニング」を参照。
max-beans-in-cache	3-12 ページの「EJB キャッシュ サイズの設定」を参照。
concurrency-strategy	3-14 ページの「データベース ロックの委任」を参照。
isolation-level	3-14 ページの「トランザクションのアイソレーション レベルの設定」を参照。

以降の節では、これらの要素について説明します。

EJB プール サイズの設定

WebLogic Server では、すべてのステートレス セッション Bean クラスに対して EJB のフリー プールが保持されます。weblogic-ejb-jar.xml ファイルの max-beans-in-free-pool 要素は、このフリー プールのサイズを定義します。デフォルトでは、max-beans-in-free-pool は無制限です。フリー プール内の Bean の最大数はメモリによってのみ制限されます。

この節では次の内容について説明します。

- 3-10 ページの「セッション Bean およびメッセージ Bean に対するプール サイズの割り当て」
- 3-11 ページの「エンティティ Bean に対するプール サイズの割り当て」
- 3-11 ページの「プール サイズのチューニング」

関連項目

- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「max-beans-in-free-pool」
- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「max-beans-in-free-pool の使用」

セッション Bean およびメッセージ Bean に対するプール サイズの割り当て

EJB が作成されると、セッション Bean インスタンスが作成され、ID が与えられます。クライアントが Bean を削除すると、その Bean インスタンスはフリー プールに置かれます。その後 Bean を作成する場合、フリー プールに置かれている直前のインスタンスを再利用することで、オブジェクトの割り当てを回避できます。`max-beans-in-free-pool` 要素を使用すると、EJB が頻繁に作成および削除される場合のパフォーマンスを向上させることができます。

メッセージの並行処理での必要に応じて、EJB コンテナではメッセージ Bean の新しいインスタンスが作成されます。`max-beans-in-pool` 要素によって、作成されるインスタンス数に対して絶対的な制限が設定されます。使用可能な実行時リソースに応じて、コンテナでこの設定値がオーバーライドされる場合もあります。

ステートレス セッション Bean およびメッセージ Bean の最高のパフォーマンスを引き出すには、`max-beans-in-free-pool` 要素のデフォルト値を使用します。デフォルト値を使用すると、できる限り多くのスレッドを使用して Bean を並行して実行できます。並行して実行される Bean の数を制限する場合を除き、この値を変更しないでください。

エンティティ Bean に対するプール サイズの割り当て

ファインダやホーム メソッドを呼び出す場合や、エンティティ Bean を作成する場合に使用される匿名エンティティ Bean のプールがあります。`max-beans-in-free-pool` 要素は、このフリー プールのサイズを指定します。

数多くのファインダやホーム メソッドを実行している場合や、多くの Bean を作成している場合は、プール内で使用可能な Bean を確保できるよう、`max-beans-in-free-pool` 要素をチューニングすることもできます。

プール サイズのチューニング

セッション Bean を頻繁に作成し、処理を迅速に行って Bean を解放する場合を除き、`max-beans-in-free-pool` パラメータの値は変更しないでください。変更する場合は、フリー プールを 25 ～ 50% 拡張して、パフォーマンスが改善されるかどうかを確認します。オブジェクトの作成が負荷全体の中でわずかな割合しか占めない場合、このパラメータを大きくしてもパフォーマンスは大幅には改善されません。EJB がデータベースを頻繁に使用するアプリケーションの場合は、このパラメータの値を変更しないでください。

警告 : このパラメータを大きくしすぎると、余計なメモリが消費されます。小さくしすぎると、不必要にオブジェクトが作成されます。このパラメータの変更について疑問がある場合は、値をそのままにしておいてください。

フリー プール内の初期 Bean のチューニング

`weblogic-ejb-jar.xml` ファイルの `initial-beans-in-free-pool` 要素を使用すると、起動時のフリー プール内のステートレス セッション Bean インスタンスの数を指定できます。

`initial-bean-in-free-pool` の値を指定すると、WebLogic Server では、起動時に、指定した数の Bean インスタンスがフリー プールに生成されます。この方法で Bean インスタンスをフリー プールに格納しておくと、要求が来てから新しいインスタンスを生成せずに Bean に対する初期要求が可能になるため、EJB の初期応答時間が短縮されます。

initial-bean-in-free-pool が定義されていない場合のデフォルト値は 0 です。

『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』にある、initial-beans-in-free-pool 要素の説明を参照してください。

EJB キャッシュ サイズの設定

WebLogic Server では、EJB キャッシュ（Bean が存在するインメモリ スペース）に存在するアクティブな Bean の数をコンフィグレーションできます。

weblogic-ejb-jar.xml ファイルの max-beans-in-cache 要素は、メモリに保持可能なこのクラスのオブジェクトの最大数を指定します。max-bean-in-cache の値に達すると、WebLogic Server では、最近クライアントに使用されていない EJB の一部に対してパッシベーションが行われます。また、max-beans-in-cache 要素の値は、EJB を WebLogic Server のキャッシュからいつ削除するかにも影響を与えます。

この要素を使用すれば、ステートフル セッション Bean とエンティティ Bean のキャッシュ サイズを同じように設定できます。

詳細については、「エンティティ EJB のロック サービス」を参照してください。

『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「max-beans-in-cache」を参照してください。

ステートフル セッション EJB のアクティベーションとパッシベーション

過度のパッシベーションとアクティベーションを回避するには、max-beans-in-cache 要素を使用してキャッシュを適切なサイズに設定します。アクティベーションとは、2 次ストレージからメモリに EJB インスタンスを転送することです。パッシベーションとは、メモリから 2 次ストレージに EJB インスタンスを転送することです。max-beans-in-cache の値を大きくしすぎると、メモリが不必要に消費されます。

EJB コンテナでは、ejbPassivate() メソッドを呼び出すことでパッシベーションが実行されます。EJB セッション オブジェクトが再び必要になると、ejbActivate() メソッドによって、そのオブジェクトが呼び出されます。ejbPassivate() メソッドが呼び出されると、EJB オブジェクトは Java シリアライゼーション API または類似のメソッドによってシリアライズされ、2 次メモリ（ディスク）に格納されます。ejbActivate() メソッドが呼び出されると、これとは反対の処理が実行されます。

コンテナは、クライアントまたはサーバの直接の関与を必要とせずに、EJB キャッシュ内の一連のセッション オブジェクトを自動的に管理します。各 EJB の特定のコールバック メソッドは、これらのオブジェクトのパッシベーション（キャッシュへの格納）またはアクティベーション（キャッシュからの取り出し）の方法を定義します。アクティベーションとパッシベーションの回数が多すぎると、EJB キャッシュ内に一連のセッション オブジェクトをキャッシュするというパフォーマンス上のメリットが消えてしまいます（特にアプリケーションで多くのセッション オブジェクトを処理する必要がある場合）。

データベース ロックの委任

WebLogic Server では、データベース ロックと排他的ロックのメカニズムがサポートされています。デフォルトであり、EJB 1.1 および EJB 2.0 に対する推奨メカニズムは、データベース ロックです。

データベース ロックによって、エンティティ EJB の同時アクセスの処理速度が向上します。WebLogic Server コンテナでは、ロック サービスを基盤となるデータベースに委ねることにより、エンティティ EJB の同時アクセスの改善が行われます。排他的ロックとは異なり、基盤データ ストアはより高い粒度で EJB データをロックでき、ほとんどの場合では、さらにデッドロックの検出もできます。

データベース ロックの詳細については、「エンティティ EJB のロック サービス」を参照してください。

`weblogic-ejb-jar.xml` の `concurrency-strategy` デプロイメント パラメータを設定することによって、EJB 用に使用するロック メカニズムを指定します。`http://edocs.beasys.co.jp/e-docs/wls61/ejb/reference.html#concurrency_strategy_60` を参照してください。

トランザクションのアイソレーション レベルの設定

データへのアクセスは、トランザクションのアイソレーション レベル メカニズムを介して制御されます。トランザクション アイソレーション レベルは、マルチユーザ データベース システムにおいて、複数のインターリーブされるトランザクションが互いを干渉しないようにするレベルを決定します。トランザクションのアイソレーションは、トランザクション データの読み書きを管理するロック プロトコルによって実現されます。トランザクション データは、「シリアライゼーション」というプロセスでディスクに書き込まれます。アイソレーション レベルを低くすると、トランザクションのアイソレーションは低くなりますが、データベースの同時接続性を高めることができます。

詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』にある、`weblogic-ejb-jar.xml` ファイルの isolation-level 要素の説明を参照してください。

異なるアイソレーション レベルの関係と各アイソレーション レベルのサポートの詳細については、各データベースのマニュアルを参照してください。

WebLogic Server 起動用パラメータのチューニング

WebLogic 配布キットには、WebLogic Server の起動に使用できるサンプル スクリプトが付属しています（「WebLogic Server の起動と停止」を参照）。

それらのスクリプトは、実際の環境やアプリケーションに合わせて修正する必要があります。管理サーバの起動用と管理対象サーバの起動用に、別々のサンプル スクリプトが用意されています。管理サーバを起動するためのスクリプトは、`startWebLogic.sh`（UNIX）と `startWeblogic.cmd`（Windows）です。これらのスクリプトは、ドメインのコンフィグレーション サブディレクトリに配置されています。

これらのファイル内で重要なパフォーマンス チューニング パラメータは、`JAVA_HOME` パラメータと Java ヒープ サイズ パラメータです。

- 変数 `JAVA_HOME` の値を `JDK` の位置に変更します。次に例を示します。

  ```
  set JAVA_HOME=C:\bea\jdk131
  ```

- パフォーマンスとスループットをより高めるには、最小 Java ヒープ サイズを最大ヒープ サイズと同じ大きさに設定します。次に例を示します。

  ```
  "%JAVA_HOME%\bin\java" -hotspot –ms512m –mx512m -classpath
  %CLASSPATH% -
  ```

  ヒープ サイズ オプションの設定の詳細については、2-8 ページの「ヒープ サイズ値の指定」を参照してください。

Java コンパイラの設定

JSP サーブレットをコンパイルするための標準 Java コンパイラは javac です。サーバの Java コンパイラを javac ではなく sj または jikes に設定することにより、パフォーマンスを大幅に向上させることができます。以降の節では、この手順とコンパイラに関するその他の考慮事項について説明します。

Administration Console でのコンパイラの変更

コンパイラを Administration Console で変更するには、次の手順を行います。

1. WebLogic Server Console を起動します。
2. ナビゲーション ツリーで [サーバ] フォルダを開きます。
3. [サーバ] フォルダ内のサーバ（デフォルト インストールの myserver）を選択します。
4. [コンフィグレーション] タブを選択します。
5. [コンパイラ] タブを選択し、[Java コンパイラ] テキスト ボックスにコンパイラの絶対パスを入力します。次に例を示します。

   ```
   c:\visualcafe31\bin\sj.exe
   ```

6. [クラスパスの後ろに追加] テキスト ボックスに JRE rt.jar ライブラリの絶対パスを入力します。次に例を示します。

   ```
   weblogic_home\jdk131\jre\lib\rt.jar
   ```

7. [適用] をクリックします。
8. サーバを再起動して、[Java コンパイラ] および [クラスパスの後ろに追加] テキスト ボックスを有効にします。

weblogic.xml でのコンパイラの設定

weblogic.xml ファイルでは、jsp-descriptor 要素を使用してサーブレット JSP のパラメータの名前と値を定義します。

compileCommand パラメータでは、生成される JSP サーブレットをコンパイルするための Java コンパイラを指定します。

precompile パラメータを使用すると、WebLogic Server の起動時に WebLogic Server で JSP をあらかじめコンパイルするようコンフィグレーションできます。

サーバの Java コンパイラを weblogic.xml ファイルで設定する作業の詳細については、jsp-descriptor 要素を参照してください。

EJB コンテナ クラスのコンパイル

WebLogic Server には、EJB 2.0 および 1.1 のコンテナ クラスをコンパイルするための weblogic.ejbc ユーティリティが付属しています。EJB コンテナにデプロイするために .jar ファイルをコンパイルする場合は、weblogic.ejbc を使用して、コンテナ クラスを生成する必要があります。ejbc は、デフォルトでは javac をコンパイラとして使用します。パフォーマンスを向上させるには、-compiler フラグを使用して別のコンパイラ（Symantec の sj など）を指定します。

詳細については、「WebLogic Server EJB のユーティリティ」を参照してください。

UNIX でのコンパイル

UNIX マシンで JSP ファイルをコンパイルしている際に、以下のようなエラー メッセージを受信することがあります。

```
failed: java.io.IOException: Not enough space
```

その場合には、次の解決策のいずれか、またはすべてを試みてください。

- RAM が 256 MB しかない場合には、追加します。

- ファイル記述子の限度を、たとえば次のように引き上げます。

```
set rlim_fd_max = 4096
set rlim_fd_cur = 1024
```

- `-native` フラグを使って、JVM の起動時にネイティブ スレッドを使用します。

WebLogic Server クラスタとスケーラビリティ

WebLogic Server クラスタは WebLogic Server のグループで、互いに連携してフェイルオーバおよびレプリケーション サービスを提供することにより、クライアントに対してスケーラブルで可用性の高い運用をサポートします。クラスタはそのクライアントにとって単一のサーバに見えますが、実際には、一体で機能するサーバ群です。

スケーラビリティとは、リソースの追加に伴ってシステムが 1 つまたは複数の次元で拡張していく能力です。通常、これらの次元には（数ある中で）サポート可能な同時接続ユーザの数や、一定時間に処理可能なトランザクションの数などがあります。

優れたアプリケーションであれば、単にリソースを追加することによってパフォーマンスが向上します。WebLogic Server の負荷処理の機能を強化するには、新しい WebLogic Server をクラスタに追加するだけで済みます。その際、アプリケーションを変更する必要はありません。クラスタは、単一のサーバでは提供できない、スケーラビリティと可用性という 2 つの大きなメリットをもたらします。

WebLogic Server クラスタは、アプリケーション開発者からは見えないように、J2EE アプリケーションにスケーラビリティと高可用性を提供します。スケーラビリティにより、中間層の能力が単一の WebLogic Server またはコンピュータの能力を超えたところまで拡張されます。クラスタ メンバーシップの唯一の制限は、すべての WebLogic Server が IP マルチキャストで通信できなければならないということです。新しい WebLogic Server をクラスタに動的に追加して能力を増大させることができます。

WebLogic Server クラスタは、複数サーバの冗長性を利用してクライアントを障害から保護することで高可用性を保証します。クラスタ内の複数のサーバで、同じサービスを提供できます。1 つのサーバで障害が発生しても、別のサーバが引き継ぎます。障害が発生したサーバから機能しているサーバへのフェイルオーバ機能によって、クライアントに対するアプリケーションの可用性が増大します。

クラスタの詳細については、『WebLogic Server Clusters ユーザーズ ガイド』を参照してください。

警告 : すべてのアプリケーションおよび環境上のボトルネックを解決した場合、新しいサーバをクラスタに追加すると、直線的なスケーラビリティが実現されます。ベンチマークまたは初期コンフィグレーション テストを実行するときには、単一サーバ環境における問題を分離してから、クラスタ化環境に移行してください。

マルチ CPU マシンのパフォーマンスに関する考慮事項

マルチプロセッサ マシンを使用する場合、クラスタ化された WebLogic Server インスタンスと使用可能な CPU の数との比率も考慮する必要があります。WebLogic Server には、クラスタ内のサーバ インスタンス数に関する制限はありません。したがって、Sun Microsystems, Inc. の Sun Enterprise 10000 などの大規模マルチプロセッサ サーバは、非常に大規模なクラスタまたは複数のクラスタのホストとなることができます。

サーバと CPU の最適比率を決定する前に、以下の事項についてアプリケーションを徹底的にテストしてください。

- ネットワークの要件 ― Web アプリケーションが主にネットワーク I/O にバインドされている場合、使用可能な CPU の数を増やす前にネットワーク スループットを高める方策を検討する必要があります。アプリケーションが完全にネットワーク I/O にバインドされている場合は、CPU を追加するより高速の NIC を取り付ける方がパフォーマンスは向上します。これは、ほとんどの CPU が、使用可能なソケットの読み込み待機中もアイドル状態のままになるためです。
- ディスク I/O の要件 ― Web アプリケーションが主にディスク I/O にバインドされている場合、CPU を追加する前に、ディスク スピンドル数または個別のディスクとコントローラの数を増やすことを検討する必要があります。

つまり、CPU を増やす場合は、事前に Web アプリケーションがネットワーク I/O またはディスク I/O ではなく完全に CPU にバインドされていることを確認しておく必要があります。

CPU にバインドされたアプリケーションの場合、まず、すべての CPU につき 1 つの WebLogic Server インスタンスの比率でパフォーマンスをテストします。CPU 使用率がほぼ 100% を常に維持していれば、サーバに対する CPU の比率を高くします（たとえば、1 つの WebLogic Server インスタンスに 2 つの CPU を割り当てる）。プロダクション システムの場合、管理タスクを実行できるよう常に利用可能な CPU サイクルを予備として残しておく必要があることに注意してください。

Web アプリケーションの処理のニーズによって異なりますが、BEA では、WebLogic Server インスタンスと CPU の比率が 1 対 2 の場合に、一般的に最適の結果が得られることを確認しています。

WebLogic Server ドメインのモニタ

WebLogic Server ドメインの状態とパフォーマンスをモニタするためのツールは Administration Console です。Administration Console では、サーバ、HTTP、JTA サブシステム、JNDI、セキュリティ、CORBA 接続プール、EJB、JDBC、JMS といった WebLogic Server リソースのステータスと統計を表示できます。

詳細については、「WebLogic Server ドメインのモニタ」を参照してください。

4 WebLogic Server アプリケーションのチューニング

WebLogic Server のパフォーマンスは、その上で動作するアプリケーションによって決まります。以下の節では、パフォーマンスを低下させるボトルネックの解決について説明します。

パフォーマンス解析ツールの使い方

この節では、WebLogic Server での OptimizeIt および JProbe の各プロファイラの使用方法について説明します。

プロファイラは、高い CPU 使用率または共有リソース競合率を引き起こすアプリケーション内のホット スポットを発見するためのパフォーマンス解析ツールです。一般的なプロファイラについては、A-4 ページの「パフォーマンス解析ツール」を参照してください。

JProbe Profiler API の使い方

JProbe Profiler with Memory Debugger は、パフォーマンス ボトルネックの検出、コード カバレッジの実行、およびその他のメトリックの実行機能を備えた製品ファミリです。

JProbe Profiler API を使用するには、以降の節での説明に従って、環境、コード、およびプログラムの起動を変更します。

環境の変更

環境に以下のような変更を行います。

1. 次のディレクトリを PATH 変数に追加します。

   ```
   .../jprobe../profiler
   ```

2. 次の .jar を CLASSPATH 変数に追加します。

   ```
   .../jprobe.../profiler/profilerAPI.jar
   ```

コードの変更

```
import com.klg.jprobe.profiler.api.*;

....

// 記録を開始する
JPPerformanceAPI.getInstance().pauseRecording();
JPPerformanceAPI.getInstance().clear();
JPPerformanceAPI.getInstance().resumeRecording();

// これらの呼び出しの戻り値は標準出力に出力できる。
// 戻り値はブール値でしかなく、問題が発生しても
// エラーの表示や、例外の送出は行われない
```

ここにコードが入る ...

```
// JProbe データをスナップショットに保存する
JPPerformanceAPI.getInstance().pauseRecording();
if (!JPPerformanceAPI.getInstance().save("myprofile")) {
System.out.println ("COULD NOT SAVE PERF DATA");
System.exit(0);
}
```

プログラムの起動

次の例のように、プログラム（この場合は WebLogic Server）を起動します。

```
jplauncher -classic -jp_snapshot_dir=D:\temp \
-jp_function=performance \
-jp_java_exe=C:\java\java131\bin\java.exe \
-jp_java_home=C:\java\java131 \
-jp_vm=java2 \
```

```
-jp_measurement=elapsed \
-ms128m \
-mx128m \
-Dweblogic.management.username=system \
-Dweblogic.management.password=gumby1234 \
-Djava.security.policy==java.policy \
-Dweblogic.ConsoleInputEnabled=true \
-jp_record_from_start=performance \
weblogic.Server
```

OptimizeIt Profiler API の使い方

OptimizeIt Java Performance Profiler は、Solaris および NT 用のパフォーマンス デバッグ ツールです。OptimizeIt Profiler API を使用するには、以降の節での説明に従って、環境、コード、およびプログラムの起動を変更します。

環境の変更

OptimizeIt Profiler API を使用するには、環境に以下のような変更を行います。

1. OptimizeIt40 ライブラリを PATH 変数に追加します。

   ```
   .../OptimizeIt40/lib
   ```

2. optit.jar を CLASSPATH 変数に追加します。

   ```
   .../OptimizeIt40/optit.jar
   ```

コードの変更

```
// Security.Audit クラスもあるので、場所によっては
// 完全パッケージ名を使用する
intuitive.audit.Audit.start(
intuitive.audit.Audit.DEFAULT_PORT_NUMBER,
intuitive.audit.Audit.PROFILERS_ALWAYS_ENABLED,
intuitive.audit.Audit.SYSTEM_EXIT_DISABLED);

intuitive.audit.Audit.enableCPUProfiler();
```

ここにコードが入る ...

```
intuitive.audit.Audit.disableCPUProfiler();

intuitive.audit.Audit.generateSnapshot("myprofile",
intuitive.audit.Audit.INCLUDE_CPU); // myprofile.snp を生成する
```

プログラムの起動

次の例のように、プログラム（この場合は WebLogic Server）を起動します。

```
java -classic -Xrunoii -Xnoclassgc \
-ms128m mx128m\
-Dweblogic.management.username=system\
-Dweblogic.management.password=gumby1234 \
-Djava.security.policy==java.policy \
-Dweblogic.ConsoleInputEnabled=true \
intuitive.audit.Audit -startCPUprofiler:type=cpu weblogic.Server
```

JDBC アプリケーションのチューニング

BEA では、WebLogic Server ソフトウェアで使用する 3 つの WebLogic jDriver を提供します。

- 分散トランザクション機能を含む Oracle 用 Type 2 ネイティブ JDBC ドライバ

  『WebLogic jDriver for Oracle のインストールと使い方』を参照してください。

- Informix 用と Microsoft SQL Server 用の Type 4 JDBC ドライバ

『WebLogic jDriver for Informix のインストールと使い方』、および『WebLogic jDriver for Microsoft SQL Server のインストールと使い方』を参照してください。

Type 2 ドライバでは、データベース ベンダが提供するクライアント ライブラリを使用します。一方、Type 4 ドライバは pure-Java であり、通信レベルでデータベース サーバに接続するので、ベンダ固有のクライアント ライブラリは不要です。

『WebLogic JDBC プログラミング ガイド』の「JDBC アプリケーションのパフォーマンス チューニング」を参照してください。

Type 4 MS SQL ドライバ向けの JDBC の最適化

Type 4 MS SQL ドライバを使用すると、SQL 文の作成および実行速度が大幅に向上する場合があります。その場合は、一連の長い `setXXX()` 呼び出しの後に `execute()` を実行するのではなく、パラメータを指定しないか、またはパラメータ値を対応する文字列に変換し、その文字列に追加します。

セッションの永続性の管理

セッションの永続性を処理する場合、アプリケーションの作業ができるだけ少なくなるようアプリケーションを最適化します。WebLogic Server では、以下のオプションを使用してセッションの永続性を管理します。

- 4-6 ページの「インメモリ レプリケーション」
- 4-6 ページの「JDBC ベースの永続性」

詳細については、『WebLogic Server 管理者ガイド』の「セッションの永続性のコンフィグレーション」を参照してください。

インメモリ レプリケーション

インメモリ レプリケーションは、セッション ステートの JDBC ベースの永続性に比べて最大で 10 倍高速です。可能であれば、インメモリ レプリケーションを使用してください。

詳細については、『WebLogic Server クラスタ ユーザーズ ガイド』の「HTTP セッション ステートのレプリケーションについて」を参照してください。

また、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「ステートフル セッション EJB のインメモリ レプリケーション」も参照してください。

JDBC ベースの永続性

JDBC ベースの永続性を使用する場合、コードを最適化して、セッション ステートの永続性の粒度をできるだけ高くします。JDBC ベースの永続性の場合、セッション「格納」を実行するたびに、オブジェクト全体のデータベースへの書き込みが行われます。

HTTP セッション中に情報が永続化される頻度を最小限に抑える必要があります。実行される「格納」を調べ、それらを 1 つの大きい「格納」に統合します。

詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』の「データベースの永続ストレージとしての使い方（JDBC 永続性）」を参照してください。

セッションの最小化

アプリケーションをチューニングして最高のパフォーマンスを引き出すには、WebLogic Server のセッション管理の方法をコンフィグレーションすることが重要になります。以下のことを考慮してください。

- セッションの使用にはスケーラビリティのトレードオフが伴います。
- セッションの使用を抑えます。

ステートをクライアント上で現実的に保持できない場合か、または URL 書き換えのサポートが必要な場合にのみ、セッションを使用します。ユーザ名などの単純なステートを直接クッキーに保持します。また、ラッパー クラスを記述して、これらのクッキーの取得および設定を行うこともできます。これにより、同じプロジェクトに参加しているサーブレット開発者の作業が簡素化されます。

- 頻繁に使用する値をローカル変数に格納します。
- 可能な限り、複数の単一オブジェクトではなく集合オブジェクトをセッションに配置します。

『Web アプリケーションのアセンブルとコンフィグレーション』の「セッション管理の設定」を参照してください。

実行キューによるスレッド使用の制御

アプリケーションの実行スレッドへのアクセスをきめ細かくチューニングし、パフォーマンスを最適化または CPU 使用率を抑制するには、WebLogic Server で複数の実行キューを設定します。実行キューは、1 つまたは複数の指定されたサーブレット、JSP、EJB、または RMI オブジェクトで利用可能な実行スレッドの名前付きコレクションを表します。

WebLogic Server をデフォルト インストールした場合、実行キューは `default` にコンフィグレーションされます。この実行キューは、サーバ インスタンス上で実行中のすべてのアプリケーションで使用されます。キューは、以下の目的のためにコンフィグレーションして追加できます。

- **重要度の高いアプリケーションのパフォーマンスの最適化。**たとえば、単一のミッションクリティカルなアプリケーションを特定の実行キューに割り当てると、固定数の実行スレッドを確保することができます。サーバの負荷のピーク時には、重要度の低いアプリケーションがデフォルト実行キューと競合することもありますが、ミッションクリティカルなアプリケーションは常に同数のスレッドにアクセスできます。
- **重要度の低いアプリケーションのパフォーマンスの抑制。**大量のメモリを消費する可能性があるアプリケーションがある場合、指定した実行キューにそのアプリケーションを割り当てると、効果的にメモリの消費量を制限することができます。割り当てた実行キューで使用可能なすべてのスレッドをアプ

リケーションで使用できますが、他のキューのスレッドの使用には影響しません。

- **スレッド使用でのデッドロックの回避。**アプリケーションの設計によっては、スレッドがすべて使用中になるとデッドロックが発生することがあります。たとえば、指定した JMS キューからメッセージを読み込むサーブレットがあると仮定します。サーバのすべての実行スレッドがサーブレット リクエストの処理に使用されると、JMS キューからメッセージを配信するのに使用できるスレッドはなくなってしまいます。この状況のときに、デッドロック状態が生じ、処理が続行できなくなります。この場合、別々の実行キューにサーブレットを割り当てると、サーブレットと JMS キューにスレッド リソースの競合が生じないので、デッドロックを回避することができます。

実行キューの欠点

実行キューを使用すると、アプリケーションのパフォーマンスをきめ細かくチューニングできます。ただし、未使用のスレッドは、Weblogic Server システムのリソースをかなり消費する点に注意してください。すべての実行キューのスレッド使用について慎重に考慮しないと、他のキューのアプリケーションがアイドル状態でスレッドが使用可能になるのを待機している間に、コンフィグレーションされた実行キューの使用可能なスレッドが未使用になってしまう可能性があります。この場合、スレッドをキューに分割したことが原因で、1 つのデフォルト実行キューの場合よりも全体のパフォーマンスが低下する可能性があります。

システム全体でスレッドを適切に使用するには、必ず各実行キューをモニタしてください。スレッド数の最適化に関する概要については、3-3 ページの「スレッド数の設定」を参照してください。

実行キューの作成

実行キューは、`Server` 要素の一部としてドメイン `config.xml` ファイル内で指定します。たとえば、名前が `CriticalAppQueue` でスレッドが 4 つある実行キューの場合、`config.xml` ファイルで次のように指定します。

```
...
<Server
```

```
 Name="examplesServer"
 ListenPort="7001"
 NativeIOEnabled="true"/>
 <ExecuteQueue Name="default"
  ThreadCount="15"/>
 <ExecuteQueue Name="CriticalAppQueue"
  ThreadCount="4"/>
 ...
</Server>
```

WebLogic Administration Console で新しい実行キューを作成するには、次の手順に従います。

1. Administration Console を起動して、実行キューを追加するサーバの名前をクリックします。
2. [モニタ] タブをクリックします。
3. [一般] タブで [すべてのアクティブなキューのモニタ] をクリックします。
4. [Execute Queue のコンフィグレーション] をクリックします。
5. [新しい Execute Queue のコンフィグレーション] をクリックします。
6. 名前、スレッドの優先度、および新しいキューのスレッド数を入力します。
7. [作成] をクリックすると、`config.xml` に新しいキューが作成されます。

サーブレットおよび JSP の実行キューへの割り当て

初期化パラメータの実行キュー名を識別することにより、コンフィグレーションされた実行キューにサーブレットまたは JSP を割り当てることができます。初期化パラメータは、サーブレットの `init-param` 要素内、または JSP のデプロイメント記述子ファイル `web.xml` 内で指定されています。実行キューを割り当てるには、次に示すように `wl-dispatch-policy` パラメータの値としてキュー名を入力します。

```
<servlet>
   <servlet-name>MainServlet</servlet-name>
   <jsp-file>/myapplication/critical.jsp</jsp-file>
   <init-param>
      <param-name>wl-dispatch-policy</param-name>
      <param-value>CriticalAppQueue</param-value>
   </init-param>
</servlet>
```

web.xml での初期化パラメータの指定に関する詳細については、『WebLogic HTTP サーブレット プログラマーズ ガイド』の「サーブレットの初期化」を参照してください。

EJB オブジェクトおよび RMI オブジェクトの実行キューへの割り当て

コンフィグレーションされた実行キューに RMI オブジェクトを割り当てるには、rmic コンパイラで -dispatchPolicy オプションを使用します。次に例を示します。

```
java weblogic.rmic -dispatchPolicy CriticalAppQueue ...
```

コンフィグレーションされた実行キューに EJB オブジェクトを割り当てるには、ejbc で -dispatchPolicy オプションを使用します。EJB のコンパイル時に ejbc コンパイラによって、このオプションと引数が rmic に渡されます。

A 関連情報

この章では、以下のような、パフォーマンスに関連する参照リストを示します。

BEA Systems, Inc. の情報

- BEA Systems の一般的な情報については、BEA の Web サイトを参照してください。
- BEA WebLogic Server 製品ドキュメントページ
- BEA WebLogic Server ホワイト ペーパー
- 「J2EE Design Considerations for WebLogic Server」、BEA ホワイト ペーパー、2000 年

- 『J2EE Applications and BEA WebLogic Server』、Michael Girdley、Rob Woollen、Sandra Emerson 著、2001 年
- 『Professional J2EE Programming with BEA WebLogic Server』、Paco Gomez、Peter Zadrozny 著、2000 年
- 『J2EE Performance Testing with BEA WebLogic Server』、Peter Zadrozny、Philip Aston、Ted Osborne 著、2002 年

Sun Microsystems の情報

- Sun Microsystems の一般的な情報については、Sun の Web サイトを参照してください。
- Sun Microsystems のパフォーマンス情報
- Java Standard Edition プラットフォーム ドキュメント
- Java 2 SDK、Standard Edition ドキュメント
- 「Solaris Tunable Parameters Reference Manual」

- BEA WebLogic Server および Solaris 固有の詳細情報については、BEA の「プラットフォーム サポート」ページの「SPARC 上の Sun Microsystems Solaris」を参照してください。
- Solaris コンフィグレーションの詳細については、Solaris FAQ をチェックしてください。

- 『Sun Performance and Tuning Java and the Internet』、Adrian Cockcroft 他著、1997 年
- 『Solaris 7 Performance Administration Tools』、Frank Cervone 著、2000 年

Hewlett-Packard Company の情報

- Hewlett-Packard Company の一般情報
- HP-UX のチューニングと BEA WebLogic Server
- HP-UX 上での Java パフォーマンス チューニング
- glance/jpm パフォーマンス診断ツール
- HPjconfig Java コンフィグレーション ツール

Microsoft の情報

- Microsoft の一般情報
- 「Windows 2000 Performance Tuning」、ホワイト ペーパー
- SQL-Server-Performance.Com、Microsoft SQL Server のパフォーマンス チューニングと最適化に関する情報
- 『Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook』、Ken England 著、2001 年、Digital Press

Web パフォーマンス チューニングの情報

- 「Apache Performance Notes」
- 「iPlanet Web Server 4.0 Performance Tuning, Sizing, and Scaling」
- 「The Art and Science of Web Server Tuning with Internet Information Services 5.0」
- 『Web Performance Tuning: Speeding Up the Web』、Patrick Killelea 著、Linda Mui 編、O'Reilly Nutshell、1998 年

- 『Capacity Planning for Web Performance: Metrics, Models, and Methods』、Daniel A. Menasce、Virgilio A. F. Almeida 著、Prentice Hall PTR、1998 年

ネットワーク パフォーマンス ツール

- Systems, Software, Technology (SST) Incorporated の TracePlus/Ethernet

  TracePlus/Ethernet は、Windows 95/98/ME、NT 4.x、Windows 2000/XP 用のネットワーク パケット解析ツールです。

パフォーマンス解析ツール

プロファイラは、高い CPU 使用率または共有リソース競合率を引き起こすアプリケーション内のホット スポットを発見するためのパフォーマンス解析ツールです。以下に、一般的なプロファイラを示します。

- OptimizeIt Java Performance Profiler（Solaris および NT 用のパフォーマンス デバッグ ツール）
- JProbe Profiler with Memory Debugger（パフォーマンス ボトルネックの検出、コード カバレッジの実行、およびその他のメトリックの実行機能を備えた製品ファミリ）
- 「Product Review: OptimizeIt vs. JProbe」、Journal of Object Oriented Programming、June 2001 号

- Hewlett Packard JMeter（Hewlett Packard のプロファイリング情報解析ツール）
- Topaz、Mercury Interactive のアプリケーション パフォーマンス管理ソリューション
- SE Toolkit（パフォーマンス解析ツール キット）

ベンチマーク情報

- SPECjbb2000

  SPECjbb2000 は、Standard Performance Evaluation Corporation（SPEC）によって開発されたソフトウェア ベンチマーク製品です。この製品は、システムの Java サーバ アプリケーションの実行能力を測定します。

- ECPerf Benchmark 仕様

- eTesting Labs Inc.

  Ziff Davis Media 社の eTesting Labs Inc. は、WebBench 4.0 などのベンチマーク ソフトウェアの独立開発ベンダです。

Java 仮想マシン（JVM）の情報

- artima.com の JVM コーナー
- HP-UX 上でのチューニングの規定手順、ガベージ コレクション、およびヒープ サイズ
- 「Frequently asked questions about the Java HotSpot virtual machine」

  この Sun Microsystems FAQ では、Java HotSpot テクノロジとパフォーマンス全般に関する一般的な質問への回答を参照できます。

- 「Tuning Garbage Collection with the 1.3.1 Java Virtual Machine」

  この Sun Microsystems ドキュメントには、ガベージ コレクションのチューニングの概要が記述されています。

- 「Java HotSpot VM Options」

  この Sun Microsystems ドキュメントには、Java HotSpot 仮想マシンのパフォーマンス特性に影響を与えるコマンドライン オプションと環境変数に関する情報が記述されています。

- 「The Java HotSpot Client and Server Virtual Machines」

この Sun Microsystems ドキュメントでは、J2SE 1.3 用の Java 仮想マシンの 2 つの実装が説明されています。

- 「Which Java VM scales best?」

  「JavaWorld」の記事です。VolanoMark 2.0 サーバ ベンチマークの結果では、12 台の仮想マシンがどのように積み重ねられるかが示されています。

- 『Garbage Collection : Algorithms for Automatic Dynamic Memory Management』、Richard Jones、Rafael D Lins 著、John Wiley & Sons、1999 年

エンタープライズ JavaBean の情報

- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』
- 『Enterprise JavaBeans, Second Edition』、Richard Monson-Haefel 著、Mike Loukides 編、2000 年
- 『Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition』、Ed Roman 著、1999 年
- TheServerSide.com

  エンタープライズ JavaBean（EJB）および J2EE 専門の無料のオンライン コミュニティです。

- 『Seven Rules for Optimizing Entity Beans』、Akara Sucharitakul 著、Java Developer Connection、2001 年

Java Message Service（JMS）の情報

- 『WebLogic JMS プログラマーズ ガイド』

一般的なパフォーマンス情報

- Jack Shirazi の Java パフォーマンス チューニング Web サイト
- 「The Software Testing and Quality Engineering Magazine」、Web Application Scalability、「Avoiding Scalability Shock」、Bill Shea 著、May/June 2000 号
- 『High-Performance Java Platform Computing™』、Thomas W. Christopher、George K. Thiruvathukal 著、2000 年
- 『Performance and Idiom Guide』、Craig Larman、Rhett Guthrie 著、1999 年

索引

ふ

へ

ま

め

れ