BEA WebLogic Server™

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 7.0
マニュアルの改訂 :2002 年 8 月 30 日

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

パート番号	マニュアルの改訂	ソフトウェアのバージョン
なし	2002 年 8 月 30 日	BEA WebLogic Server バージョン 7.0

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：パフォーマンス ツールと情報

B. ベンチマークを利用した WebLogic Server 7.0 SP1 のチューニング例

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 : パフォーマンス ツールと情報」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。
- 付録 B「ベンチマークを利用した WebLogic Server 7.0 SP1 のチューニング例」では、ECPerf または SPECjAppServer 2001/2002 ベンチマークを実行する際のチューニングのヒントを示します。

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、パフォーマンスのモニタおよびチューニング担当者を対象としています。サーバの管理、ハードウェアのパフォーマンス 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 プログラミング言語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。または、WebLogic Server 製品ドキュメント ページ (http://edocs.beasys.co.jp/e-docs/index.html) を直接表示してください。

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、WebLogic Server の 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。

関連情報

WebLogic Server の管理およびチューニングについての詳細は、以下を参照してください。

- 『管理者ガイド』 (http://edocs.bea.co.jp/e-docs/wls/docs70/adminguide/index.html)
- BEA の dev2dev Web サイト
- BEA Newsgroup サーバで利用できる WebLogic Server パフォーマンスのニュースグループ「weblogic.developer.interest.performance」

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@beasys.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〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

表記法	適用

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、データ構造体とそのメンバー、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
{ }	構文の中で複数の選択肢を示す。

[ ]	構文の中で任意指定の項目を示す。
	例:
	`java utils.MulticastTest -n *name* -a *address*`
	`[-p *portnumber*] [-t *timeout*] [-s *send*]`

|	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}`

表記法	適用
. . .	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と、WebLogic Server および特定のアプリケーションをサポートするために特定のハードウェア コンフィグレーションが必要とするキャパシティが、さまざまな要因に左右されています。アプリケーションをサポートするために必要なハードウェア要件は、アプリケーションとコンフィグレーションの詳細によって異なります。これらの要因がそれぞれコンフィグレーションおよびアプリケーションにどのように影響するかを考慮する必要があります。

この節を読み進める前に、コンピュータのシステム パフォーマンスを評価する際の標準ベンチマークおよびメトリックを提供している「Standard Performance Evaluation Corporation」に目を通しておいてください。

サポートされるプラットフォーム

次の表では、「WebLogic Platform サポート対象のコンフィグレーション」ページにある情報への、選定されたリンクを示しています。このページには、WebLogic Server の各リリースについてサポートされている、ハードウェアやオペレーティング システムに関する最新の動作確認情報の完全なリストが記載されています。

表 1-1 プラットフォーム別チューニング情報

プラットフォーム	詳細情報の参照先
AIX 搭載の Bull/IBM pSeries	「WebLogic Platform サポート対象のコンフィグレーション」ページの「Bull/IBM」リンクを参照。 ■ AIX 4.3.3 搭載の Bull/IBM pSeries ■ AIX 5L v5.1 搭載の Bull/IBM pSeries ■ AIX 5L v5.2 搭載の Bull/IBM pSeries
HP-UX 搭載の Hewlett-Packard	「WebLogic Platform サポート対象のコンフィグレーション」ページの「HP-UX 11.0 および 11i 搭載の Hewlett-Packard HP/9000 」を参照。 A-4 ページの「Hewlett-Packard Company の情報」も参照。
Windows 対応の Intel Pentium 互換機	「WebLogic Platform サポート対象のコンフィグレーション」ページの「Intel/Windows」リンクを参照。 ■ IA-32 対応の Windows 2000 Server、Windows 2000 Advanced Server ■ IA-32 対応の Windows 2000 Professional ■ IA-32 対応の Windows NT 4.0 ■ IA-32 対応の Windows XP A-4 ページの「Microsoft の情報」も参照。

表 1-1 プラットフォーム別チューニング情報

プラットフォーム	詳細情報の参照先
Red Hat Advanced Server 搭載の Intel 32 ビット互換機	「WebLogic Platform サポート対象のコンフィグレーション」ページの「Red Hat」リンクを参照。 ■ IA-32 対応の Red Hat Enterprise Linux AS 2.1 および ES 2.1 ■ IA-32 対応の Red Hat Enterprise Linux WS 2.1 A-3 ページの「Linux OS の情報」も参照。
Solaris 搭載の Sun Microsystems SPARC	「WebLogic Platform サポート対象のコンフィグレーション」ページの「Sun Microsystems SPARC Solaris」リンクを参照。 ■ Solaris 2.7 搭載の Sun Microsystems/Fujitsu SPARC ■ Solaris 8 搭載の Sun Microsystems/Fujitsu SPARC ■ Solaris 9 搭載の Sun Microsystems/Fujitsu SPARC A-2 ページの「Sun Microsystems の情報」も参照。

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、「WebLogic Platform サポート対象のコンフィグレーション」ページに記載されている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。

Windows プラットフォームの場合、通常はデフォルト設定で十分です。しかし、Soloaris および Linux プラットフォームでは通常、適切なチューニングを行う必要があります。

ndd コマンドを使用した TCP パラメータの設定

次の例に従い、ndd コマンドを使用して以下の TCP 関連のチューニング パラメータを設定します。

```
ndd -set /dev/tcp tcp_conn_req_max_q 16384
```

表 1-2 TCP 関連のパラメータの推奨値

パラメータ	推奨値
/dev/tcp tcp_time_wait_interval	60000
/dev/tcp tcp_conn_req_max_q	16384
/dev/tcp tcp_conn_req_max_q0	16384
/dev/tcp tcp_ip_abort_interval	60000
/dev/tcp tcp_keepalive_interval	7200000
/dev/tcp tcp_rexmit_interval_initial	4000
/dev/tcp tcp_rexmit_interval_max	10000
/dev/tcp tcp_rexmit_interval_min	3000
/dev/tcp tcp_smallest_anon_port	32768
/dev/tcp tcp_xmit_hiwat	131072
/dev/tcp tcp_recv_hiwat	131072
/dev/tcp tcp_naglim_def	1
/dev/ce instance	0
/dev/ce rx_intr_time	32

注意： Solaris 2.7 以前では、tcp_time_wait_interval パラメータを tcp_close_wait_interval と呼んでいました。このパラメータには、close の呼び出しを発行した後に TCP ソケットを生存させる時間間隔を指定します。Solaris の場合、このパラメータのデフォルト値は 4 分となります。短時間に数多くのクライアントが接続する場合、これらのソ

ケット リソースを保持するとパフォーマンスに大きな悪影響を及ぼすことがあります。Solaris でのベンチマーク JSP テストでは、このパラメータの値を 60000 (60 秒) に設定するとスループットがかなり改善するという結果が出ています。オープン途中の接続のキューでサーバがバックアップされている場合は、この値をもっと減らすことも可能です。

ヒント : 使用できる TCP パラメータをすべて表示するには、`netstat -s -P tcp` コマンドを使用します。

/etc/system ファイル内のパラメータの設定

サーバへの各ソケット接続では、ファイル記述子を消費します。ソケット パフォーマンスを最適化するには、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ります。したがって、デフォルトのファイル記述子の制限、ハッシュ テーブルのサイズなど、`/etc/system` ファイル内にあるチューニング パラメータを次の表に示す推奨値に変更してください。

注意 : `/etc/system` パラメータを変更した場合は、マシンを再起動する必要があります。

表 1-3 /etc/system の推奨値

パラメータ	推奨値
`set rlim_fd_cur`	`8192`
`set rlim_fd_max`	`8192`
`set tcp:tcp_conn_hash_size`	`32768`
`set shmsys:shminfo_shmmax` **注意 :** これは 4GB 以上の RAM を搭載したマシンについてのみ設定できる。	`4294967295`
`set autoup`	`900`
`set tune_t_fsflushr`	`1`

CE Gigabit ネットワーク カードの設定

CE Gigabit カードを使用する場合は、以下の設定をお勧めします。

表 1-4 CE Gigabit カードを使用する場合の推奨値

パラメータ	推奨値
`set ce:ce_bcopy_thresh`	`256`
`set ce:ce_dvma_thresh`	`256`
`set ce:ce_taskq_disable`	`1`
`set ce:ce_ring_size`	`256`
`set ce:ce_comp_ring_size`	`1024`
`set ce:ce_tx_ring_size`	`4096`

Solaris のチューニング オプションの詳細については、以下を参照してください。

- 「Solaris Tunable Parameters Reference Manual」(Solaris 8)
- 「Solaris Tunable Parameters Reference Manual」(Solaris 9)

Linux チューニング パラメータ

Linux オペレーティング システムで最適なパフォーマンスを実現するには、以下のように設定することをお勧めします。

表 1-5 Linux の場合の推奨値

パラメータ	推奨値
`/sbin/ifconfig lo mtu`	`1500`
`kernel.msgmni`	`1024`
`kernel.sem`	`1000 32000 32 512`
`fs.file-max`	`65535`
`kernel.shmmax`	`2147483648`
`net.ipv4.tcp_max_syn_backlog`	`8192`

Linux のチューニングの詳細については、Linux ベンダのマニュアルを参照してください。また、「Ipsysctl Tutorial 1.0.4」では、Linux で提供されるすべての IP オプションを説明しています。

その他のオペレーティング システムのチューニング情報

Windows、HP-UX、および AIX のチューニング オプションの詳細については、以下の Web サイトを参照してください。

- Windows のチューニング情報については、「Microsoft Windows 2000 TCP/IP Implementation Details」を参照。
- HP-UX のチューニング情報については、「Tunable Kernel Parameters 」リファレンスを参照。
- AIX のチューニング情報については、「AIX 5L Version 5.2 Performance Management Guide」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で確認。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。また、オペレーティング システムのマニュアルも参照。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。

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、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。最近のエンタープライズ レベルのネットワークは非常に高速であるため、適切に設計されたアプリケーションであれば、パフォーマンス低下の直接原因になることは稀です。しかし、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ります。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) が用意されているかどうかを検証する必要もあります。したがって、パフォーマンスの潜在的なボトルネックを解消するには、ネットワークのパフォーマンスを継続的にモニタすることが重要です。

ネットワーク帯域幅の決定

帯域幅は、一般には「データ通信の転送レート、通常は通信を送受信するためのリンクの容量を bps (bits-per-second) で表したもの」と定義されます。WebLogic Server を実行するマシンには、そのすべての WebLogic Server クライアント接続を処理できるだけの十分なネットワーク帯域幅が必要です。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、ソケットごとに専用の帯域幅が必要になります。WebLogic Server インスタンス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。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できます。ネットワークの利用状況は、Solaris の `netstat` コマンド、Windows のシステム モニタ (`perfmon`) など、一般的なオペレーティング システム ツールを使用してモニタすることもできます。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。

また、アプリケーションとアプリケーション サーバの間およびアプリケーション サーバとデータベース サーバの間で転送されているデータをチェックして、ネットワーク上で転送されるデータの量をモニタします。このデータ量がネットワークの帯域幅を超えないようにします。帯域幅を超える場合、ネットワークがボトルネックとなります。これを確認するには、次のコマンドを使用して、パケットの再転送や重複に関するネットワーク統計値をモニタします。

```
netstat -s -P tcp
```

`netstat -s -P` コマンドを使用してその他の TCP パラメータを表示する手順については、1-4 ページの「ndd コマンドを使用した TCP パラメータの設定」を参照してください。

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ります。トラフィックの量が常に帯域幅の容量を超えて、ネットワークが最大限に活用されているのに、WebLogic Server マシンを有効に活用できていない場合は、以下のいずれかの対処をしてください。

- 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。

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 コードを実行する仮想の「実行エンジン」インスタンスです。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、WebLogic Server 用の JVM チューニング オプションについて説明します。

- 2-2 ページの「JVM のチューニングの考慮事項」
- 2-3 ページの「JVM のヒープ サイズとガベージ コレクション」
- 2-7 ページの「ヒープ サイズ値の指定」
- 2-10 ページの「自動的な低メモリ状態のロギング」
- 2-11 ページの「ガベージ コレクションの手動リクエスト」
- 2-12 ページの「Java HotSpot VM オプションの設定」

JVM チューニングの関連情報へのリンクについては、付録 A「関連情報 : パフォーマンス ツールと情報」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 7.0 は、Java 1.3 準拠の JVM のみサポートする。 「動作確認状況」ページは頻繁に更新されて、さまざまなプラットフォームの最新の動作確認情報が示される。
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	WebLogic Server のヒープ サイズ チューニングの詳細については、2-3 ページの「JVM のヒープ サイズとガベージ コレクション」を参照。 `java.sun.com` でのガベージ コレクションの概要については、「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」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。詳細については、クライアント / サーバ JVM の混在のサポート ページを参照。
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、Sun Microsystems 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 Microsystems の JVM 1.3.x、JIT オプションは有効ではなくなった。A-7 ページの「Java 仮想マシン (JVM) の情報」を参照。

JVM のヒープ サイズとガベージ コレクション

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 JVM のプロセスです。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)」と見なされ、コレクションの対象となります。

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、WebLogic Server で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、JVM によるガベージ 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、2-7 ページの「ヒープ サイズ値の指定」を参照してください。

ヒープ領域の不足を自動的に検出して、サーバの低メモリ状態を解決するよう WebLogic Server をコンフィグレーションするには、2-10 ページの「自動的な低メモリ状態のロギング」を参照してください。

世代別ガベージ コレクション

Java HotSpot JVM 1.3 では、割り当て速度と全体的なガベージ コレクションの効率を大幅に向上させる世代別コレクタが使用されます。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アクセス可能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。Hotspot JVM は、ほとんどのオブジェクトは有効期間が短く、コレクションを考慮する必要がないという、効率的なガベージ コレクションが行われる状態を前提として動作します。

世代別ガベージ コレクションを使用した場合、Java ヒープは Young、Old という 2 つの世代領域に分割されます。Young 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。オブジェクトはヒープ サイズの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、Young 領域から Old 領域に移動されます。Young 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、2-7 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、ヒープ サイズが影響を受け、それによってガベージ コレクションの実行頻度が決まります。したがって、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、それによって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。

verbose ガベージ コレクションを使用したヒープ サイズの決定

verbose ガベージ コレクション (`verbosegc`) を使用すると、ガベージ コレクションに費やされる時間とリソースを正確に測定できます。最も効率的なヒープ サイズを判断するには、診断を目的として 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します。

次に、この手順を簡単に示します。

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 JVM 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。

   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。

たとえば、Windows および Solaris では次のように入力します。

```
% java -ms32m -mx200m -verbosegc -classpath $CLASSPATH
-Dweblogic.Name=%SERVER_NAME% -Dbea.home="C:\bea"
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.management.server=%ADMIN_URL%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"
weblogic.Server
>> logfile.txt 2>&1
```

`logfile.txt 2>&1` コマンドは、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。

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
```

`$$` は、Java プロセスのプロセス ID (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。

3. 以下のデータ ポイントを解析します。

   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。

4. Java HotSpot JVM 1.3 を使用している場合は、New 世代のヒープ サイズを設定します。

   2-7 ページの「ヒープ サイズ値の指定」および 2-9 ページの表 2-2「Java ヒープ サイズのオプション」を参照してください。

5. 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。

6. 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。

7. 使用可能な空き RAM が大量に残っている場合は、そのマシンでより多くの WebLogic Server インスタンスを実行します。

   もう一度確認しますが、ヒープ サイズのチューニングの目標は、WebLogic Server で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、JVM によるガベージ コレクションの実行時間を最小限に抑えることです。

ヒープ サイズ値の指定

Java ヒープ サイズの値は、WebLogic Server を起動するたびに指定する必要があります。ヒープ サイズ値は、Java コマンドラインから指定するか、WebLogic Server 起動用に WebLogic 配布キットに付属しているサンプル起動スクリプトのデフォルト値を変更して指定します。

たとえば、Java コマンドラインから WebLogic Server を起動する場合、ヒープ サイズ値は次のように指定できます。

```
$ java -XX:NewSize=128m -XX:MaxNewSize=128m -XX:SurvivorRatio=8
-Xms512m -Xmx512m
-Dweblogic.Name=%SERVER_NAME% -Dbea.home="C:\bea"
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.management.server=%ADMIN_URL%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"
 weblogic.Server
```

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。ヒープ サイズ オプションの詳細については、2-8 ページの「Java ヒープ サイズのオプション」を参照してください。

WebLogic 起動スクリプトを使用したヒープ サイズの設定

WebLogic Server 配布キットには、サーバを起動するサンプル起動スクリプト、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起動スクリプトが付属しています。

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

これらのスクリプトは、*`WL_HOME`*`\server\bin` にあります (*`WL_HOME`* は WebLogic Server のインストール位置)。これらの起動スクリプトは、Java に渡されるデフォルト メモリ引数 (つまり、ヒープ サイズ) や JDK の位置などの環境変数を設定し、その後に WebLogic Server の引数を使用して JVM を起動します。

WebLogic Server 起動スクリプトでは、デフォルト ヒープ サイズのパラメータが指定されます。したがって、実際の環境やアプリケーションに合わせてそれらのパラメータを変更する必要があります。「スクリプトを使用した管理サーバの起動」(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は、各アプリケーションを個別にチューニングすることで実現されます。ただし、WebLogic Server の起動時に表 2-2 の JVM ヒープ サイズ 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を向上させることができます。

これらのオプションは、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。
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して、ガベージ コレクションを最小限に抑える。

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。

自動的な低メモリ状態のロギング

WebLogic Server では、サーバが観測した低メモリ状態を自動的にロギングできます。WebLogic Server では、ある特定の時間間隔で設定された回数だけ利用可能な空きメモリをサンプリングして低メモリが検出されます。各間隔の最後に、空きメモリの平均が記録され、それが次の間隔で得られた平均と比較されます。サンプル間隔の後に、ユーザがコンフィグレーションした量だけ平均が落ち込んだ場合、サーバは低メモリの警告メッセージをログ ファイルに記録し、サーバの状態を「警告」に設定します。

平均空きメモリが、サーバの起動直後に記録された初期空きメモリの 5% を下回ると、WebLogic Server はメッセージをログ ファイルに記録します。

低メモリ検出プロセスの各側面は、次のように Administration Console を使用してコンフィグレーションします。

1. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. ナビゲーション ツリーの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. コンフィグレーションするサーバ インスタンスの名前をクリックします。低メモリの検出は、サーバごとにコンフィグレーションします。
5. 右ペインの [コンフィグレーション | メモリ] タブを選択します。
6. 次の属性を必要に応じて変更し、選択したサーバ インスタンスに対する低メモリの検出をチューニングします。
   - [低メモリ GC しきい値] WebLogic Server が低メモリ警告をロギングし、状態を「警告」に変更した後のしきい値を表すパーセント値 (0 ～ 99%)

を入力します。デフォルトでは、[低メモリ GC しきい値] は 5% に設定されます。つまり、平均空きメモリがサーバの起動時に測定された初期空きメモリの 5% に達すると、低メモリ警告がロギングされます。

- [低メモリ粒度レベル]: 低メモリ状態をログに記録し、サーバの状態を「警告」に変更するために使用するパーセント値 (1 ～ 99%) を入力します。デフォルトでは、この値は 5% に設定されます。つまり、2 つの間隔の間で平均空きメモリが 5% 以上落ち込むと、サーバがログ ファイルに低メモリの警告を記録し、サーバの状態を「警告」に変更します。
- [低メモリ サンプル サイズ]: 決められた期間にサーバが空きメモリをサンプリングする回数を入力します。デフォルトでは、各間隔で 10 回空きメモリをサンプリングして平均空きメモリが算出されます。サンプル サイズを増やすと、数値がより正確になります。
- [低メモリ時間間隔]: 平均空きメモリ値を測定する間隔を定義する時間を秒単位で入力します。デフォルトでは、3600 秒ごとに平均空きメモリ値が取得されます。

7. [適用] をクリックして変更を反映させます。
8. サーバを再起動して、低メモリ検出の新しい属性を使用します。

ガベージ コレクションの手動リクエスト

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 コレクションを実行すると、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 インスタンスでガベージ コレクションを要求するには、次の手順に従います。

1. Administration Console のナビゲーション ツリーで、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。右ペインのダイアログ ボックスに、このインスタンスと関連付けられているタブが表示されます。
2. [モニタ | パフォーマンス] タブを選択します。

3. [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、実行中のサーバの情報のみが表示されます。
4. [ガベージコレクションを強制する] ボタンをクリックして、ガベージ コレクションを要求します。[ガベージコレクションを強制する] ボタンにより、JVM の `System.gc()` メソッドが呼び出され、ガベージ コレクションが実行されます。ただし、このリクエストが実際にガベージ コレクションをトリガするかどうかは、JVM の実装そのもので決定されます。

Java HotSpot VM オプションの設定

標準の `java` コマンドライン オプションを使用すると、JVM の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 の両方をテストすると、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オプションを調べることができます。Sun Microsystems の「Java HotSpot VM Options」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。

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、2-14 ページの「非標準の Java オプション (Windows および UNIX 用)」を参照してください。

標準の Java オプション (Windows および UNIX 用)

Windows では、WebLogic Server は java コマンドを通じ、表 2-3 のオプションのいずれかを指定して特定バージョンの JVM を起動します。

表 2-3 Windows 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。HotSpot クライアント VM では、Sun Microsystems によれば、クラシック VM よりも優れたパフォーマンスが提供される。
-classic	クラシック VM を選択する。クラシック VM は、本質的には Java 2 SDK のバージョン 1.2 と同じ仮想マシンの実装である。 **注意：** Java 2 クラシック VM は、Java 2 SDK にのみ含まれる。Java 2 Runtime Environment には含まれていない。-classic オプションは、Java 2 Runtime Environment では機能しない。

UNIX では、WebLogic Server は java コマンドを通じ、表 2-4 のオプションのいずれかを指定して特定バージョンの JVM を起動します。

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。

Sun Microsystems の「The Java HotSpot Client and Server Virtual Machines」では、J2SE 1.3 で利用可能な Java 仮想マシンの 2 つの実装が説明されています。

非標準の Java オプション (Windows および UNIX 用)

非標準の Java オプションを使用しても、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。非標準のコマンドライン オプションは、将来のリリースにおいて変更される可能性があります。

表 2-5 に、Windows 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 2 つ示します。

表 2-5 Windows 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、指定されたクラスのガベージ 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。
`-Xrs`	JVM によるオペレーティング システム シグナルの使用を減らす。

非標準 Windows オプションの他の例については、「Non-Standard Options (Windows VM 用)」(を参照してください。

表 2-6 に、UNIX Solaris 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 2 つ示します。

表 2-6 Solaris 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、指定されたクラスのガベージ 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。
`-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。

Solaris 用の非標準オプションの他の例については、「Non-Standard Options (Solaris VM 用)」(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。

パフォーマンス関連の config.xml 要素の設定

WebLogic Server のコンフィグレーション ファイル (`config.xml`) には、実際の環境やアプリケーションに合わせて細かくチューニングできるパフォーマンス関連の多くの要素があります。`config.xml` ファイル (管理サーバのホスト マシン上にある) は、MBean 属性値の永続ストレージを提供します。システム管理ツール (コマンドライン インタフェースまたは Administration Console) を使用して属性値を変更するたびに、その値は適切な管理 MBean に格納され、`config.xml` ファイルに書き込まれます。各 WebLogic Server ドメインには、それ専用の `config.xml` ファイルがあります。

WebLogic Server システム管理ツールの使い方の詳細については、『管理者ガイド』の「システム管理ツール」(を参照してください。

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
Server	NativeIOEnabled	3-3 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	3-4 ページの「スレッド数の設定」を参照。
ExecuteQueue	QueueLength QueueLengthThresholdPercent ThreadsIncrease ThreadsMaximum ThreadsMinimum	3-10 ページの「オーバーフロー条件に対する実行キューのチューニング」を参照。
Server	StuckThreadMaxTime StuckThreadTimerInterval	3-12 ページの「「スタック」スレッドの検出」を参照。
Server	ThreadPoolPercentSocketReaders	3-8 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。
Server	AcceptBacklog	3-13 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	3-14 ページの「JDBC 接続プールによるパフォーマンスの向上」を参照。
JDBCConnectionPool	PreparedStatementCacheSize	3-16 ページの「Prepared Statement のキャッシング」を参照。

WebLogic Server パフォーマンス パックの使い方

ベンチマークは、WebLogic Server インスタンスのホスト マシンでネイティブ パフォーマンス パックを使用したときにパフォーマンスの大きな向上を示します。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のソケット 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。ただし、ホスト マシンで pure-Java ソケット リーダー実装を使用しなければならない場合でも、サーバ インスタンスおよびクライアント マシンごとにソケット リーダー スレッドの数を適切にコンフィグレーションすることによって、ソケット通信のパフォーマンスを向上させることができます。

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、サポート対象のプラットフォームを確認するには、次の手順に従います。

1. 「WebLogic Platform サポート対象のコンフィグレーション」を表示します。
2. サポート対象のプラットフォームのリストから、必要なプラットフォームのリンクをクリックします。

   次のページには、サポート対象の WebLogic Server の各リリース (サービス パックを含む) に関する情報の表があります。各リリースの表には、そのリリースにパフォーマンス パックが「付属」しているかどうかを示す「パフォーマンス パック」欄があります。
3. パフォーマンス パックの情報を確認するには、ページ上部にある WebLogic Server の特定のリリースをクリックしてから、該当する表を探します。または、ブラウザの [編集 | このページの検索] 機能を使用して、ページ内の「パフォーマンス パック」という文字をすべて検索します。

パフォーマンス パックの有効化

`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。

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. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. コンフィグレーションするサーバ インスタンスの名前をクリックします。
5. [コンフィグレーション | 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 の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、サーバ インスタンスで動作するすべてアプリケーションによって使用されます。また、WebLogic Server のインスタンスには、`__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、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： ネイティブ パフォーマンス パックがプラットフォームで使用されていない場合は、パフォーマンスを最適化するために、実行キューのスレッドのデフォルト数とソケット リーダーとして機能するスレッドの割合をチューニングする必要があります。詳細については、3-8 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」を参照してください。

デフォルト 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。スレッドはメモリを消費するので、`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。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、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、その結果としてスレッド数の値をより大きくしなければならなくなります。

デフォルトの 15 を超えるスレッドを使用する必要がない場合は、この属性の値を変更しないようにします。一般に、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は、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。後者のケースでは、実行スレッドの数を減らすとパフォーマンスが向上します。

デフォルト スレッド数のシナリオ

実行キューの理想的なスレッド数を決めるには、キュー内のすべてのアプリケーションが最大負荷で動作しているときにキューのスループットをモニタします。キューのスレッド数を増やし、キューのスループットが最適になるまで負荷テストを繰り返します。あるポイントまでスレッドの数を増やしていくと、コンテキストの切り替えが増えすぎて、キューのスループットが低下し始めるので注意してください。

注意： WebLogic Server Administration Console には、サーバのすべての実行キューの累積スループットが表示されます。このスループット値を表示するには、3-7 ページの「デフォルト実行キューのスレッド数の変更」の手順 1 ～ 6 を行ってください。

表 3-2 に、WebLogic Server システムの CPU 数との関係で使用可能なスレッドを調整するためのデフォルトのシナリオを示します。これらのシナリオも、WebLogic Server が最大負荷で動作し、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します。

表 3-2 デフォルト スレッド数のシナリオ

条件	結果	対策
スレッド数 < CPU の数	以下の場合はスレッド数が少なすぎる。 ■ 実行できる処理があるのに、CPU が処理を待機している。 ■ 100% の CPU 使用率を実現できない。	スレッド数を増加させる。
スレッド数 = CPU の数	理論的には理想的だが、まだ 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 (スレッド数が適度に多い)	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。

表 3-2 デフォルト 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 (スレッド数が過度に多い)	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。 スレッド数を減らせばパフォーマンスは向上する。	CPU と同じ数にスレッド数を減らしてから、確認された「スタック」スレッドの数だけ追加する。 たとえば、プロセッサが 4 つの場合は、スタック スレッドの数に加えて 4 つのスレッドを同時に実行できる。このため、実行スレッド数は、4 とスタック スレッド数を足した数が望ましい。 スタック スレッドの数を確認するには、3-12 ページの「「スタック」スレッドの検出」を参照。 **注意：** 以上のことは、アプリケーションに大きく依存する。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。

デフォルト実行キューのスレッド数の変更

Administration Console でデフォルト実行キューのスレッド数を変更するには、次の手順を行います。

1. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。

4. コンフィグレーションする実行キューを含むサーバ インスタンスの名前をクリックします。変更できるのは、サーバのデフォルト実行キューまたはユーザ定義の実行キューだけです。
5. 右ペインの [モニタ | 一般] タブを選択します。
6. [すべてのアクティブなキューのモニタ] テキスト リンクをクリックして、選択したサーバが使用する実行キューを表示します。
7. [Execute Queue のコンフィグレーション] テキスト リンクをクリックして、変更可能な実行キューを表示します。
8. コンフィグレーションされている実行キューの表で、デフォルト実行キューの名前をクリックして、実行キューの [コンフィグレーション] タブを表示します。
9. スレッド数のデフォルト値を必要に応じて増減します。
10. [適用] をクリックして変更を反映させます。
11. 選択したサーバを再起動して、新しい実行キュー設定を有効にします。

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、4-4 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

ソケットの最適なパフォーマンスを実現するには、WebLogic Server インスタンスのホスト マシン上では、pure-Java 実装ではなくネイティブのソケット リーダー実装を使用することをお勧めします (3-3 ページの「WebLogic Server パフォーマンス パックの使い方」を参照)。ただし、ホスト マシンで pure-Java ソケット リーダー実装を使用しなければならない場合でも、サーバ インスタンス

およびクライアント マシンごとにソケット リーダー スレッドとして機能する実行スレッドの数を適切にコンフィグレーションすることによって、ソケット通信のパフォーマンスを向上させることができます。

`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。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。

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。

WebLogic Server でのソケット リーダー スレッド数の設定

Administration Console を使用し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、次の手順を行います。

1. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. コンフィグレーションするサーバの名前をクリックします。
5. [コンフィグレーション | チューニング] タブを選択します。
6. [ソケット リーダー] 属性フィールドで Java リーダー スレッドの割合を編集します。Java ソケット リーダーの数が、合計実行スレッド数 ([実行スレッド] 属性フィールドに表示されます) の割合として計算されます。
7. 変更を適用します。

クライアント マシンでのソケット リーダー スレッド数の設定

クライアント マシン上では、クライアントを実行する Java 仮想マシン (JVM) 内でソケット リーダーの数をコンフィグレーションできます。java コマンドラインで -Dweblogic.ThreadPoolSize=*value* オプションおよび -Dweblogic.ThreadPoolPercentSocketReaders=*value* オプションを定義してクライアントのソケット リーダー数を指定します。

オーバーフロー条件に対する実行キューのチューニング

デフォルトの実行キューやユーザ定義の実行キューでオーバーフローになりそうな条件を検出し、必要に応じて対応するようにサーバをコンフィグレーションできます。WebLogic Server は、キューのサイズがユーザ定義の最大サイズにおけるパーセンテージに達した場合、キューにオーバーフロー条件の可能性があることを認識します。しきい値に達すると、サーバは自己の状態を「警告」に変更し、必要に応じて、キュー内の未処理の作業を追加のスレッドに割り当ててそのサイズを軽減します。

キューのオーバーフロー条件を自動的に検出したり、対処したりするには、以下の項目をコンフィグレーションします。

- サーバがオーバーフロー条件を示すしきい値。この値は、コンフィグレーションされた実行キューのサイズのパーセンテージとして設定します (QueueLength 値)。
- オーバーフロー条件が検出されたときに実行キューに追加されるスレッドの数。これらの追加スレッドは、キューのサイズを減らし、通常の動作サイズまで軽減するよう機能します。
- キューに対して使用できる最小および最大スレッド数。特に、スレッドの最大数を設定することで、サーバがオーバーロード条件への対応のために過剰なスレッド数を割り当てることを防ぎます。

WebLogic Server Administration Console を使用して実行キューをチューニングするには、次の手順に従います。

1. 管理サーバが起動していない場合は、起動します。

2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. コンフィグレーションする実行キューを含むサーバ インスタンスの名前をクリックします。変更できるのは、サーバのデフォルト実行キューまたはユーザ定義の実行キューだけです。
5. 右ペインの [モニタ | 一般] タブを選択します。
6. [すべてのアクティブなキューのモニタ] テキスト リンクをクリックして、選択したサーバが使用する実行キューを表示します。
7. 変更可能な実行キューを表示するには、[Execute Queue のコンフィグレーション] テキスト リンクをクリックします。
8. コンフィグレーションするデフォルトの実行キューまたはユーザ定義の実行キューの名前をクリックして、実行キューの [コンフィグレーション] タブを表示します。
9. このサーバが、選択したキューに対するオーバーフロー条件をどのように検出するかを指定するには、次の属性を変更します。
   - [キューの長さ]: [キューの長さ] 属性が、実行キューが達する最大の長さを示すようにします。この値は、キューの通常の動作の長さよりも高い値にしてください。デフォルトでは、[キューの長さ] は 65536 エントリに設定されています。
   - [キューの長さのしきい値比率]: サーバがキューのオーバーフロー条件を示す前に到達する [キューの長さ] サイズのパーセンテージ (1 ～ 99) を入力します。しきい値のパーセンテージ以下の実際のキューの長さはすべて通常とみなされ、しきい値のパーセンテージを超えるサイズはオーバーフローを示します。デフォルトでは、WebLogic Server の [キューの長さのしきい値比率] は 90% に設定されています。
10. このサーバが、選択したキューに対するオーバーフロー条件にどのように対応するかを指定するには、次の属性を変更します。
    - [スレッド数の増分]: WebLogic Server がオーバーフロー条件を検出したときにこの実行キューに追加するスレッドの数を入力します。ゼロ スレッド (デフォルト) を指定した場合、サーバはスレッドのオーバーフロー条件に対して自己の状態を「警告」に変更しますが、追加スレッドを割り当てて負荷を軽減することはありません。
11. この実行キューの変数スレッドを微調整するには、次の属性を変更します。

- [最小スレッド数]: 不要なオーバーフロー条件を回避するために WebLogic Server がこの実行キューで維持する必要のある最小スレッド数を指定します。デフォルトでは、[最小スレッド数] は 5 に設定されます。
- [最大スレッド数]: この実行キューが持つことのできる最大スレッド数を指定します。この値によって、WebLogic Server が継続的なオーバーフロー条件に対してキュー内に過剰な数のスレッドを作成することを防ぎます。デフォルトでは、[最大スレッド数] は 400 に設定されます。

12. [適用] をクリックして変更を反映させます。
13. 選択したサーバを再起動して、新しい実行キュー設定を有効にします。

「スタック」スレッドの検出

WebLogic Server は、デフォルトの実行キューのスレッドが「スタック」状態になるとこれを自動的に検出します。スタック スレッドは現在の作業を終了したり新しい作業を承認したりできないため、サーバはスタック スレッドを診断するたびに、メッセージのロギングを行います。実行キューのすべてのスレッドがスタック状態になった場合、サーバは実行キューに基づいて自己の状態を「警告」または「危険」に変更します。

- デフォルト キューのすべてのスレッドがスタック状態になった場合、サーバは自己の状態を「危険」に変更します (状態が「危険」の場合にサーバを自動的に停止し、再起動するようノード マネージャ アプリケーションを設定できます。詳細については、『WebLogic Server ドメイン管理』の「ノード マネージャによるサーバの可用性の管理」を参照してください。
- `__weblogic_admin_html_queue` または `__weblogic_admin_rmi_queue` のすべてのスレッドがスタック状態になった場合、サーバは自己の状態を「警告」に変更します。

WebLogic Server は、設定した期間、作業状態が継続した (アイドルでない) 場合、スレッドをスタック状態と診断します。サーバのスレッド検出動作は、スレッドがスタック状態と診断されるまでの時間や、サーバがスタック スレッドをチェックする頻度を変更することでチューニングできます。

注意： WebLogic Server がスレッドがスタック状態かどうかを判断するために使用する基準は変更できますが、デフォルト実行キューのすべてのスレッドがスタック状態になった場合に「警告」および「危険」状態を設定するデフォルト動作を変更することはできません。

WebLogic Server のスレッド検出動作をコンフィグレーションするには、次の手順に従います。

1. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. [サーバ] ノードを展開して、ドメインでコンフィグレーションされたサーバを表示します。
4. スレッド検出動作をコンフィグレーションするサーバ インスタンスの名前をクリックします。スタック スレッド検出パラメータは、実行キューごとではなく、サーバごとにコンフィグレーションします。
5. [コンフィグレーション | チューニング] タブを選択します。
6. 次の属性を必要に応じて変更し、サーバに対するスレッド検出動作をチューニングします。
   - [スタック スレッド最大時間]: このサーバがスレッドをスタック状態であると診断するまでの、スレッドの継続動作時間を秒単位で入力します。デフォルトでは、WebLogic Server は 600 秒の継続使用後にスレッドをスタック状態であるとみなします。
   - [スタック スレッド タイマ間隔]: [スタック スレッド最大時間] で指定した期間スレッドが継続動作しているかどうかを WebLogic Server がスキャンする間隔を秒単位で入力します。デフォルトでは、WebLogic Server はこの間隔を 600 秒に設定しています。
7. [適用] をクリックして変更を反映させます。
8. サーバを再起動して新しい設定内容を有効にします。

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、WebLogic Server インスタンス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

Transmission Control Protocol (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 から [バックログを受け入れ] 値をチューニングするには、次の手順を行います。

1. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. コンフィグレーションするサーバ インスタンスの名前をクリックします。
5. [接続 | チューニング] タブを選択します。
6. デフォルトの [バックログを受け入れ] 値を必要に応じて変更し、待機キューに格納できる TCP 接続の数をチューニングします。
   - 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、[バックログを受け入れ] 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、[バックログを受け入れ] 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、値を 25% ずつ大きくしていきます。
7. 変更を適用します。

JDBC 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、同時クライアント セッション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。

以降の節以外にも、『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

`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。

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。

Prepared Statement のキャッシング

WebLogic Server 上で作成する各接続プールに対し、Prepared Statement キャッシュ サイズを指定できます。Prepared Statement のキャッシュ サイズを設定するとき、WebLogic Server は、指定した Prepared Statement 数に達するまでアプリケーションおよび EJB で使用される各 Prepared Statement を保存します。たとえば、Prepared Statement キャッシュ サイズを 10 に設定した場合、WebLogic Server はアプリケーションまたは EJB に呼び出される始めの 10 の Prepared Statement を保存します。

Prepared Statement キャッシュを使用することでパフォーマンスを大幅に向上させることができますが、制限があることも考慮に入れる必要があります。詳細については、『管理者ガイド』の「Prepared Statement キャッシュのパフォーマンスの向上」(を参照してください。

パフォーマンス関連の weblogic-ejb-jar.xml 要素の設定

`weblogic-ejb-jar.xml` デプロイメント ファイルには、EJB の同時実行、キャッシング、クラスタ化、および動作を定義する WebLogic Server 固有の EJB DTD が格納されます。このファイルには、利用可能な WebLogic Server リソースを EJB にマップする記述子も格納されます。WebLogic Server リソースには、セキュリティ ロール名、データ ソース (JDBC プールや JMS 接続ファクトリなど)、およびデプロイ済みの他の EJB があります。

`weblogic-ejb-jar.xml` デプロイメント ファイルの修正方法については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「EJB デプロイメント記述子の指定と編集」(を参照してください。

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
max-beans-in-free-pool	3-17 ページの「EJB プール サイズの設定」を参照。
initial-beans-in-free-pool	3-19 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	3-19 ページの「EJB キャッシュ サイズの設定」を参照。
concurrency-strategy	3-21 ページの「データベース ロックの委任」を参照。
isolation-level	3-21 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。

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。

以降の節以外に、『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」を参照してください。

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 (主キーの割り当てられていない Bean) のプールがあります。`max-beans-in-free-pool` 要素は、このプールのサイズも指定します。

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。

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-beans-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-beans-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。

initial-beans-in-free-pool 要素については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』(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-beans-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。

この要素の値は、ステートフル セッション Bean とエンティティ Bean の両方のキャッシュ サイズを設定します。

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』の「EJB の同時方式」(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。EJB 1.1 および EJB 2.0 でのデフォルトかつ推奨のロック メカニズムは、委任データベース 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、同時アクセスの改善が行われます。委任データベース ロックの場合、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。

`weblogic-ejb-jar.xml` ファイルの `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。

データベース ロックの詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「Database 同時方式」(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 を起動するための Java パラメータの設定

Java パラメータの値は、WebLogic Server を起動するたびに指定する必要があります。`weblogic.Server` コマンドを使用してコマンドラインから行うと、この作業を単純化できます。ただし、コマンドラインから WebLogic Server を起動するために必要な引数は長くなる場合があり、エラーが発生しやすくなるので、コマンドをスクリプトに組み込むことをお勧めします。そのプロセスを単純化するため、WebLogic 配布キットに付属するサンプル スクリプトのデフォルト値を変更して WebLogic Server を起動することもできます。詳細については、「スクリプトを使用した管理サーバの起動」(を参照してください。

管理サーバを起動するためのスクリプトは、`startWLS.sh` (UNIX) と `startWLS.cmd` (Windows) です。これらのスクリプトは、*`WL_HOME`*`\server\bin` ディレクトリにあります (*`WL_HOME`* は WebLogic Server のインストール位置)。

それらのスクリプトのいくつかのデフォルト Java 値は、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。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_03
  ```

- 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 –Xms512m –Xmx512m -classpath
  %CLASSPATH% -
  ```

  ヒープ サイズ オプションの設定の詳細については、2-7 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. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. コンフィグレーションするサーバ インスタンスの名前をクリックします。
5. [コンフィグレーション | コンパイラ] タブを選択し、[Java コンパイラ] フィールド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] フィールドに 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。

   ```
   BEA_HOME\jdk131_03\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 パラメータでは、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します。

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.ejbc ユーティリティを使用すると、EJB 2.0 および 1.1 のコンテナ クラスをコンパイルできます。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 コンパイラを使用します。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 マシン上で JSP ファイルをコンパイルしているときに次のエラー メッセージが表示された場合、

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

以下のいずれかまたはすべての処理を試みます。

- 256MB しかない場合は RAM を増設する。
- ファイル記述子の制限を上げる。設定例を示します。

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

- JVM の起動時に -native フラグを使用してネイティブ スレッドを使用する。

WebLogic Server クラスタの使用

WebLogic Server クラスタは WebLogic Server インスタンスのグループで、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高可用性

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 インスタンスをクラスタに追加します。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 クラスタ 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 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は、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します。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ネットワーク インタフェース カード (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は、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します。

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-6 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 の使い方

JProbe Suite は、パフォーマンス ボトルネックの検出、メモリ リークの検出と修正、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。製品の詳細については、`http://www.quest.com/jprobe/` を参照してください。

JProbe の Web サイトでは、テクニカル ホワイト ペーパー「Using Sitraka JProbe and BEA WebLogic Server」を提供しています。BEA WebLogic Server 内で動作する JProbe Suite 製品を使用して開発者がコードを分析する方法について説明しています。

OptimizeIt Profiler の使い方

Borland の Optimizeit Profiler は、Solaris および Windows プラットフォーム用のパフォーマンス デバッグ ツールです。

Borland は、WebLogic Server で機能するバージョンの Optimizeit Profiler について詳しい J2EE 統合チュートリアルを提供しています。

JDBC アプリケーションのチューニング

データベース アプリケーションのパフォーマンスの良し悪しはほとんどの場合、アプリケーションがどのように設計されているかによって決定まります。クライアントの数と場所、DBMS テーブルおよびインデックスのサイズと構造、およびクエリの数とタイプは、すべてアプリケーションのパフォーマンスに影響を与えます。

JDBC に合わせてアプリケーションを最適化する方法の詳細については、『WebLogic JDBC プログラマーズ 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。

詳細については、『WebLogic jDriver for Microsoft SQL Server のコンフィグレーションと使い方』(を参照してください。

セッションの管理

セッションの永続性およびセッションを処理する場合、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。

セッションの永続性の管理

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。

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」処理を実行するたびに、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、可能な場合は、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、以下のドキュメントを参照してください。

- 『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「セッションの永続性のコンフィグレーション」(
- 『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」(
- 『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」(
- 『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)」(

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および抑制するには、WebLogic Server で複数の実行キューを使用します。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。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ることがあります。この場合、スレッドを複数の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。

WebLogic Server をデフォルト インストールした場合、実行キューはデフォルト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については、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。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。

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、3-4 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、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。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>
```

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. 管理サーバが起動していない場合は、起動します。
2. ドメインの Administration Console にアクセスします。
3. 左ペインの [サーバ] ノードをクリックして、ドメインでコンフィグレーションされたサーバを表示します。
4. 実行キューを追加するサーバ インスタンスの名前をクリックします。
5. [モニタ | 一般] タブを選択します。
6. [すべてのアクティブなキューのモニタ] テキスト リンクをクリックして、選択したサーバが使用する実行キューを表示します。
7. [Execute Queue のコンフィグレーション] テキスト リンクをクリックして、変更可能な実行キューを表示します。
8. [新しい Execute Queue のコンフィグレーション] リンクをクリックします。
9. 実行キューの [コンフィグレーション] タブタブで、以下の属性の値を変更するか、システムのデフォルトをそのまま使用します。
   - [キューの長さ]: [キューの長さ] は常にデフォルトの 65536 エントリのままにします。[キューの長さ] では、サーバがキューに保持できる同時要求の最大数を指定します。デフォルトの 65536 は非常に大きな数です。キューの未処理の要求がこの最大値に達することはほとんどありません。

     [キューの長さ] の値に達すると、超過分に対応するためにキューのサイズが自動的に 2 倍になります。ただし、キューの要求が 65536 を超えた場合、それはキューの長さではなくキューのスレッドに問題があること示します。実行キューでスタック スレッドがないか、またはスレッド数が不十分ではないかを確認してください。

- [キューの長さのしきい値比率]: サーバがキューのオーバーフロー条件を示す前に到達する [キューの長さ] サイズのパーセンテージ (1 ～ 99) を入力します。しきい値のパーセンテージ以下の実際のキューの長さはすべて通常とみなされ、しきい値のパーセンテージを超えるサイズはオーバーフローを示します。オーバーフロー条件に達すると、WebLogic Server はエラー メッセージをログに記録し、[スレッド数の増分] 属性の値分だけキューのスレッド数を増やして負荷を軽減します。

  デフォルトでは、[キューの長さのしきい値比率] の値は 90% です。ほとんどの場合、この値は 90% のままにするか、その前後の値に設定します。処理要求の突然の増加に対処するために追加スレッドが必要になる可能性のある、現実に起こりそうなあらゆる状況に対応するためです。[キューの長さのしきい値比率] は、自動チューニング パラメータとしては使用しないでください。通常の操作状況では、そのしきい値によってスレッド数の増加が誘発されることはありません。

- [スレッド数]: このキューに割り当てられるスレッド数を指定します。デフォルトの 15 を超えるスレッドを使用する必要がない場合は、この属性の値を変更しないようにします。詳細については、3-5 ページの「デフォルト スレッド数の変更」を参照してください。

- [スレッド数の増分]: WebLogic Server がオーバーフロー条件を検出したときにこの実行キューに追加するスレッドの数を入力します。ゼロ スレッド (デフォルト) を指定した場合、サーバはスレッドのオーバーフロー条件に対して自己の状態を「警告」に変更しますが、追加スレッドを割り当てて負荷を軽減することはありません。

  WebLogic Server がオーバーフロー条件に反応してスレッド数を増やすと、その追加のスレッドはサーバが再起動されるまで実行キューにとどまります。通常は、エラー ログをモニタしてオーバーフロー条件の原因を判別し、同様の状況が今後起こらないように必要に応じてスレッド数を再コンフィグレーションします。[スレッド数の増分] と [キューの長さのしきい値比率] の組み合わせを自動チューニング ツールとして使用することはしないでください。そのようにすると、実行キューで必要以上のスレッドが割り当てられ、コンテキストの切り替えが原因でパフォーマンスが低下することになります。

- [最小スレッド数]: 不要なオーバーフロー条件を回避するために WebLogic Server がこの実行キューで維持する必要のある最小スレッド数を指定します。デフォルトでは、[最小スレッド数] は 5 に設定されています。

- [最大スレッド数]: この実行キューが持つことのできる最大スレッド数を指定します。この値によって、WebLogic Server が継続的なオーバーフロー条件に対してキュー内に過剰な数のスレッドを作成することを防ぎます。デフォルトでは、[最大スレッド数] は 400 に設定されています。
- [スレッド優先順位]: このキューに関連付けられたスレッドの優先順位を指定します。デフォルトでは、Thread Priority は 5 に設定されています。

10. [作成] をクリックして、新しい実行キューを作成します。
11. サーバを再起動して新しい設定内容を有効にします。

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、サーブレットまたは JSP のデプロイメント記述子ファイル `web.xml` の `init-param` 要素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 ユーティリティで -dispatchPolicy オプションを使用します。EJB のコンパイル時に ejbc 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：パフォーマンス ツールと情報

以下の節では、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。

- BEA Systems, Inc. の情報
- Sun Microsystems の情報
- Linux OS の情報
- Hewlett-Packard Company の情報
- Microsoft の情報
- Web パフォーマンス チューニングの情報
- ネットワーク パフォーマンス ツール
- パフォーマンス解析ツール
- ベンチマーク情報
- Java 仮想マシン (JVM) の情報
- エンタープライズ JavaBean の情報
- Java Message Service (JMS) の情報
- 一般的なパフォーマンス情報

BEA Systems, Inc. の情報

- BEA Systems の一般的な情報については、BEA Web サイトを参照 `http://www.beasys.co.jp/index.html` を参照してください。

- BEA WebLogic Server ドキュメント ページ
- BEA WebLogic Platform ドキュメント ページ
- BEA の dev2dev Web サイト
- BEA WebLogic Server 評価用ホワイト ペーパー

  http://dev2dev.bea.com/products/wlserver/resources.jsp
- 「Large-Scale Financial Applications & Service-Oriented Architectures」、Anwar Ludin 著、2002 年
- 『Professional J2EE Programming with BEA WebLogic Server』、Paco Gomez、Peter Zadrozny 著、2000 年
- 『BEA WebLogic Server Bible』、Joe Zuffoletto 他著、2002 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
- 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、「WebLogic Platform サポート対象のコンフィグレーション」で Fujitsu SPARC Solaris のリンクを参照してください。
  - Solaris 2.7 搭載の Fujitsu SPARC
  - Solaris 8 搭載の Fujitsu SPARC
  - Solaris 9 搭載の Fujitsu SPARC
- Solaris コンフィグレーションの詳細については、Solaris FAQ を確認

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

Linux OS の情報

- Linux オペレーティング システムの一般的な情報については、「Linux Online」を参照してください。
- Linux Documentation Project については、「LDP」を参照してください。
- Redhat Enterprise Linux については、「Redhat」を参照してください。
- SuSE Linux Enterprise Server については、「SuSE Linux」を参照してください。
- 『Linux Performance Tuning and Capacity Planning』、Jason R. Find 他著、2001 年、Sams
- 「Ipsysctl Tutorial 1.0.4」では、Linux で提供される IP オプションを説明しています。
- 「The Linux Cookbook: Tips and Techniques for Everyday Use」、Michael Stutz 著

Hewlett-Packard Company の情報

- Hewlett-Packard Company の一般情報
- BEA WebLogic Server と HP-UX に固有の詳細情報については、BEA の「動作確認状況」ページの「HP-UX 11.0 および 11i 搭載の Hewlett-Packard HP/9000」を参照
- HP-UX 上での Java パフォーマンス チューニング
- Hewlett Packard JMeter (Hewlett Packard のプロファイリング情報解析ツール)
- GlancePlus システムのパフォーマンス診断ツール
- HPjconfig Java システム 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

- TracePlus/Ethernet (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)

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。

- Borland の OptimizeIt Java Performance Profiler（Solaris および Windows 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、April 2004 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 (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)。SPECjbb2000 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 Kit (パフォーマンスとスケーラビリティを測定し、J2EE ユーザによる J2EE のスケーラビリティおよびチューニングの理解を助ける、Java Community Process℠ プログラムの下に開発されたソフトウェア ベンチマーク製品)

Java 仮想マシン (JVM) の情報

- artima.com の JVM コーナー
- Java HotSpot テクノロジおよびパフォーマンス全般に関する Sun Microsystems FAQ
- 「Tuning Garbage Collection with the 1.3.1 Java Virtual Machine」
- 「Java HotSpot VM Options」(HotSpot JVM 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を提供する Sun Microsystems のドキュメント)
- 「The Java HotSpot Client and Server Virtual Machines」(J2SE 1.3)
- 「Which Java VM scales best?」(「JavaWorld」の記事。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る)
- 『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)
- 『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』
- 『WebLogic Server 管理者ガイド』の「JMS の管理」
- 『WebLogic Server 管理者ガイド』の「WebLogic メッセージング ブリッジの使い方」
- BEA dev2dev Web サイトのホワイト ペーパー「WebLogic JMS Performance Guide」
- Sun Microsystems の JMS 仕様

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
- 『Java 2 Performance and Idiom Guide』、Craig Larman、Rhett Guthrie 著、1999 年

B ベンチマークを利用した WebLogic Server 7.0 SP1 のチューニング例

以下の節では、ECPerf または SPECjAppServer 2001/2002 ベンチマークの実行時に WebLogic Server のデフォルトのパフォーマンスを向上させる推奨事項を示します。最適な WebLogic Server のプロダクション チューニング値は、環境やアプリケーションによって異なります。

Intel Xeon システムのチューニング

Intel の Xeon (Pentium 4) プロセッサを使用したシステムでは、以下に示すチューニング上の推奨事項に従うと、WebLogic Server のデフォルトのチューニング コンフィグレーションと比べて最大 700% のパフォーマンス向上が望めます。それらの推奨事項は、以下のハードウェアおよびオペレーティング システム コンフィグレーションを前提としています。

- 4 プロセッサ、1.6 GHz Intel Xeon (Pentium 4)
- ハイパースレッディング テクノロジが有効化されている
- 4GB のメモリ
- Windows 2000 Advanced Server

JVM チューニングのヒント

WebLogic Server に付属の Sun JVM ではなく BEA JRockit JVM を使用します。JRockit のチューニングの詳細については、「BEA JRockit Performance Tuning Guide」(を参照してください。

- 以下の JRockit ガベージ コレクションおよびメモリ管理オプションを使用する。
  - `-Xnativethreads` (JRockit ネイティブのスレッド システム)
  - `-Xgc:parallel` (並列ガベージ コレクタ)
  - `-Xallocationtype:local` (ローカル スレッド割り当て)
- 最小および最大のヒープ サイズを 1536MB に引き上げる (`-Xms1536m -Xmx1536m`)。
- 若い世代領域 (nursery) のヒープ サイズを 512MB に指定する (`-Xns512m`)。

WebLogic Server チューニングのヒント

WebLogic Server インスタンスでチューニングに関する以下の推奨事項を行います。WebLogic Server パラメータのチューニングの詳細については、第 3 章「WebLogic Server のチューニング」を参照してください。

- Intel Pentium マシンでは 1 つの WebLogic Server インスタンスを実行する。
- デフォルトの `ExecuteQueue` パラメータを 27 スレッドに増やす。
- JDBC 接続プールの `InitialCapacity` および `MaxCapacity` データベース接続パラメータを 40 に増やす。
- JDBC 接続プールの `PreparedStatementCacheSize` パラメータを 300 に増やす。

Sun UltraSparc III システムのチューニング

Sun Microsystems の UltraSparc III プロセッサを使用したシステムでは、以下に示すチューニング上の推奨事項に従うと、WebLogic Server のデフォルトのチューニング コンフィグレーションと比べて最大 560% のパフォーマンス向上が望めます。それらの推奨事項は、以下のハードウェアおよびオペレーティング システム コンフィグレーションを前提としています。

- 4 プロセッサ、900 MHz Sun UltraSparc III
- 4GB のメモリ
- Solaris 8

JVM チューニングのヒント

以下のチューニングの推奨事項は、WebLogic Server に付属の Sun Hotspot JVM に対するものです。Hotspot JVM のチューニングの詳細については、第 2 章「Java 仮想マシン (JVM) のチューニング」を参照してください。

- HotSpot Client VM (`-client`) の代わりに HotSpot Server VM オプション (`-server`) を使用する。
- 最小および最大のヒープ サイズを 1536MB に引き上げる (`-Xms1536m -Xmx1536m`)。
- New 世代の最小および最大ヒープ サイズを 350MB に指定する (`-XX:NewSize=350m -XX:MaxNewSize=350m`)。
- New 世代のサバイバル比率を 10 に指定する (`-XX:SurvivorRatio=10`)。

WebLogic Server チューニングのヒント

WebLogic Server インスタンスでチューニングに関する以下の推奨事項を行います。WebLogic Server パラメータのチューニングの詳細については、第 3 章「WebLogic Server のチューニング」を参照してください。

- UltraSparc III マシンでは 2 つの WebLogic Server インスタンスを実行する。
- WebLogic Server インスタンスの起動時にコマンドラインで `-Dweblogic.PosixSocketReaders=1` を指定する。
- JDBC 接続プールの `InitialCapacity` および `MaxCapacity` データベース接続パラメータを 25 に増やす。
- JDBC 接続プールの `PreparedStatementCacheSize` パラメータを 300 に増やす。

索引

L

M

N

O

S

T

U

V

W

X

ふ

へ

ま

め

れ