IBM WebSphere Real Time for Linux
バージョン 3

ユーザー・ガイド

IBM WebSphere Real Time for Linux
バージョン 3

ユーザー・ガイド

お願い

本書および本書で紹介する製品をご使用になる前に、 85 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM WebSphere Real Time for Linux バージョン 3、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： IBM WebSphere Real Time for Linux
Version 3
User Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2014.2

目次

図

表

前書き

このユーザーズ・ガイドには、IBM® WebSphere® Real Time for Linux に関する一般情報が記載されています。

第 1 章 概要

この情報は IBM WebSphere Real Time for Linux に関する説明です。

このユーザー・ガイドに加えられた新たな変更個所は、左側の縦線で示されています。

このユーザー・ガイドに含めることができなかった、IBM WebSphere Real Time for Linux の最新情報は、次のサイトにあります。http://www.ibm.com/support/docview.wss?uid=swg21501145

- 『WebSphere Real Time for Linux の概要』
- 『新機能』
- 2 ページの『利点』

WebSphere Real Time for Linux の概要

WebSphere Real Time for Linux は、IBM J9 仮想マシン (JVM) にリアルタイム機能を組み込んだものです。

WebSphere Real Time for Linux は、リアルタイム機能によって IBM SDK for Java™ を拡張する、Software Development Kit を含む Java ランタイム環境です。正確な応答時間が要求されるアプリケーションでは、標準的な Java テクノロジーで、WebSphere Real Time for Linux に備えられたリアルタイム機能を活用できます。

特徴

リアルタイム・アプリケーションには、絶対的な速度よりも一貫性のある実行時間が必要です。

従来の JVM を使用してリアルタイム・アプリケーションをデプロイする際の主要な注意点は以下のとおりです。

- ガーベッジ・コレクション (GC) アクティビティーによる遅延が予測できません (長くなる可能性があります)。
- ジャストインタイム (JIT) コンパイルおよび再コンパイルが行われたときにメソッド・ランタイムが遅延し、実行時間が変動します。
- 任意のオペレーティング・システム・スケジューリング。

WebSphere Real Time for Linux は、以下の機能を提供してこれらの障害を除去します。

- Metronome ガーベッジ・コレクター (休止時間がきわめて短い、増分的な決定論的ガーベッジ・コレクター)。

新機能

このトピックでは、IBM WebSphere Real Time for Linux の変更内容を紹介します。

WebSphere Real Time for Linux V3

WebSphere Real Time for Linux V3 は、リアルタイム機能を組み込むためにこのリリースから使用可能になったフィーチャーおよび機能を基に構築された、IBM SDK for Java V7 に対する拡張です。以前のバージョンの WebSphere Real Time for Linux は、以前のリリースの IBM SDK for Java を基にしていました。

IBM SDK for Java V7 の新機能について詳しくは、IBM SDK for Java 7 インフォメーション・センターの新機能を参照してください。

このユーザー・ガイドに加えられた新たな変更個所は、左側の縦線で示されています。

Linux スケジューリング・ポリシーを使用した Java スレッドのスケジューリング

Service Refresh 1 から、リアルタイム・アプリケーションを微調整するために、**SCHED_RR** スケジューリング・ポリシーを使用して通常の Java スレッドをスケジュールできるようになりました。詳しくは、6 ページの『スレッド・スケジューリング』を参照してください。

Metronome ガーベッジ・コレクターの休止時間の制御

デフォルトでは、Metronome ガーベッジ・コレクターは、ガーベッジ・コレクション・サイクルの間に 3 ミリ秒間休止します。新しいコマンド行オプションを使用して、この値を変更して休止時間を制御することができます。このオプションについて詳しくは、26 ページの『休止時間の制御』を参照してください。

圧縮参照

Metronome ガーベッジ・コレクターは、64 ビット・プラットフォームで、圧縮参照と同様に非圧縮参照もサポートするようになりました。パフォーマンスへの影響については、35 ページの『第 7 章 パフォーマンス』を参照してください。

利点

リアルタイム環境の利点は、Java アプリケーションが標準 JVM の場合よりはるかに予測可能性が高い形で実行され、Java アプリケーションのために一貫性のあるタイミング動作が提供されることです。コンパイルやガーベッジ・コレクションなどのバックグラウンド・アクティビティーは、指定された時刻に行われるため、アプリケーションの実行中にバックグラウンド・アクティビティーが予期せずピークに達することはなくなります。

上記の利点は、Metronome リアルタイム・ガーベッジ・コレクション・テクノロジーを使用して JVM を拡張することにより、利用できます。

アクセシビリティー

アクセシビリティー機能とは、運動障害または視覚障害など身体に障害を持つユーザーが情報技術製品を快適に使用できるようにサポートする機能です。

IBM は、年齢や障害の有無にかかわらず誰もが使用できる製品の提供に努めています。

例えば、WebSphere Real Time for Linux は、マウスを使用せずにキーボードだけで操作できます。

基盤となる IBM SDK for Java V7 のアクセシビリティーに影響する問題については、IBM インフォメーション・センターを参照してください。WebSphere Real Time for Linux の固有の機能に影響するアクセシビリティーの問題はありません。

キーボード・ナビゲーション

この製品では、Microsoft Windows 標準のナビゲーション・キーを使用しています。

キーボード・ナビゲーションが必要なユーザーのために、Swing アプリケーション用の便利なキー・ストロークの説明が、Swing Key Bindings にあります。

IBM とアクセシビリティー

IBM のアクセシビリティーに対する取り組みについて詳しくは、IBM Human Ability and Accessibility Center を参照してください。

第 2 章 IBM WebSphere Real Time for Linux の理解

このセクションでは、IBM WebSphere Real Time for Linux に関する重要なコンポーネントについて説明します。

- 『Metronome ガーベッジ・コレクターの概要』

Metronome ガーベッジ・コレクターの概要

WebSphere Real Time for Linux では、標準のガーベッジ・コレクターの代わりに、Metronome ガーベッジ・コレクターを使用します。

Metronome ガーベッジ・コレクションと標準的なガーベッジ・コレクションの主な違いは、Metronome ガーベッジ・コレクションが細かく分けられた割り込み可能なステップで少しずつ実行されるのに対して、標準的なガーベッジ・コレクションでは、ガーベッジにマークを付けて収集する間はアプリケーションを停止することです。

例えば、次のようにします。

```
java -Xgcpolicy:metronome -Xgc:targetUtilization=80 yourApplication
```

この例では、60 ミリ秒ごとに 80% の時間がアプリケーションの実行に使用されるように指定しています。残りの 20% の時間は、収集すべきガーベッジが残っている場合、ガーベッジ・コレクションに使用される可能性があります。Metronome ガーベッジ・コレクターに十分なリソースが与えられている場合は、使用率レベルが保証されます。ガーベッジ・コレクションは、ヒープ内のフリー・スペース量が、動的に決定されたしきい値を下回ったときに開始されます。

Metronome ガーベッジ・コレクションとクラスのアンロード

Metronome では、標準的な Java Developer Kit と同じ方法でのクラスのアンロードをサポートします。ただし、クラスをアンロードしている間は、必要な作業のために、ガーベッジ・コレクション・アクティビティー時の休止時間が異常値になる可能性があります。

Metronome ガーベッジ・コレクターのスレッド

Metronome ガーベッジ・コレクターは、単一のアラーム・スレッドおよび複数のコレクション (GC) スレッドという 2 つのタイプのスレッドで構成されています。デフォルトでは、GC は、オペレーティング・システムに使用可能な論理アクティブ・プロセッサーごとに 1 つのスレッドを使用します。これにより、GC サイクル中に最も効率的な並列処理が可能になります。 GC サイクルは、GC が起動されてからガーベッジの解放が完了するまでの時間を意味します。 GC の全サイクルの経過時間は、Java ヒープのサイズに応じて、数秒になる場合があります。 GC サイクルには、通常、数百の GC クォンタが含まれます。これらのクォンタは、アプリケーション・コードに対する非常に短い休止であり、通常は、3 ミリ秒間続きます。サイクルおよびクォンタの要約報告書を表示するには、**-verbose:gc** を使用しま

す。詳しくは、 73 ページの『verbose:gc 情報の使用』を参照してください。 JVM の GC スレッド数は、**-Xgcthreads** オプションを使用して設定することができます。

-Xgcthreads をデフォルト値よりも大きい値にするメリットはありません。**-Xgcthreads** を減らすことで、GC サイクル中の全体的な CPU 負荷を減らすことができますが、GC サイクルは長くなります。

注: GC クォンタの目標とする時間は常に 3 ミリ秒のままです。

JVM のアラーム・スレッド数は変更できません。

Metronome ガーベッジ・コレクターは定期的に JVM を検査して、ヒープ・メモリーに十分なフリー・スペースがあるかどうかを確認します。フリー・スペース量が制限値を下回った場合、Metronome ガーベッジ・コレクターは JVM を起動してガーベッジ・コレクションを開始します。

アラーム・スレッド

単一のアラーム・スレッドでは、最小リソースの使用が保証されます。このスレッドは一定の間隔で「ウェイク」し、以下を確認します。

- ヒープ・メモリー内のフリー・スペース量
- ガーベッジ・コレクションが現在行われているかどうか

使用可能なフリー・スペースが不足しており、ガーベッジ・コレクションが行われていない場合は、アラーム・スレッドがコレクション・スレッドを起動してガーベッジ・コレクションを開始します。アラーム・スレッドは、次の JVM の検査のためにスケジュールされた時間になるまでは何も行いません。

コレクション・スレッド

コレクション・スレッドはガーベッジ・コレクションを実行します。

ガーベッジ・コレクション・サイクルが完了すると、Metronome ガーベッジ・コレクターはフリー・ヒープ・スペースの容量を確認します。フリー・ヒープ・スペースがまだ不足している場合は、別のガーベッジ・コレクション・サイクルが同じトリガー ID を使用して開始されます。十分なフリー・ヒープ・スペースがある場合は、トリガーが終了し、ガーベッジ・コレクション・スレッドは停止されます。アラーム・スレッドは引き続きフリー・ヒープ・スペースをモニターし、必要に応じて、別のガーベッジ・コレクション・サイクルを起動します。

Metronome ガーベッジ・コレクターの使い方について詳しくは、 25 ページの『Metronome ガーベッジ・コレクターの使用』 を参照してください。

スレッド・スケジューリング

Linux スケジューリング・ポリシーを通常の Java スレッドで使用して、リアルタイム・アプリケーションを調整することができます。

WebSphere Real Time for Linux では、SCHED_RR スケジューリング・ポリシーを使用して通常の Java スレッドを実行できます。 SCHED_RR ポリシーを使用すると、アプリケーションを細かく制御でき、Java スレッドのリアルタイム・パフォーマンスを向上させることができます。 SCHED_RR ポリシーを使用して Java が開始されると、メイン・スレッドの優先順位およびポリシーが JVM によって検出さ

れます。 JVM はそれに応じて、優先順位およびポリシーのマッピングを変更します。通常の Java スレッドの優先順位およびポリシーについて詳しくは、 21 ページの『スレッドのスケジューリングとディスパッチング』を参照してください。

Linux スケジューリング・ポリシーには、以下のようなものがあります。

SCHED_OTHER
ほとんどのスレッドで使用される、デフォルトの一般的なタイム・シェアリング・スケジューリング・ポリシー。これらのスレッドには、ゼロの優先順位を割り当てる必要があります。

SCHED_OTHER ではタイム・スライシングを使用します。これは、次のスレッドの実行が許可されるまでに、各スレッドが制限時間内に実行されることを意味します。

SCHED_FIFO
優先順位がゼロより高い場合にのみ使用できます。SCHED_FIFO スレッドが使用可能になると、そのスレッドは通常の SCHED_OTHER スレッドよりも優先されます。

より優先順位が高い SCHED_FIFO スレッドが使用可能なった場合、このスレッドは、より低い優先順位を持つ既存の SCHED_FIFO スレッドよりも優先されます。その後、このスレッドは、その優先順位のキューの先頭に保持されます。

タイム・スライシングは使用されません。

注: SCHED_FIFO は WebSphere Real Time for Linux では使用されません。

SCHED_RR
SCHED_FIFO を拡張したものです。各スレッドの実行が許可されるのが制限時間内のみであるという点が異なります。その制限時間を超えると、スレッドは優先順位のリストに戻されます。SCHED_RR は WebSphere Real Time for Linux V3 で使用可能です。

これらの Linux スケジューリング・ポリシーについて詳しくは、 **sched_setscheduler** の **man** ページを参照してください。

WebSphere Real Time for Linux での Linux スケジューリング・ポリシーの使用について詳しくは、 21 ページの『スレッドのスケジューリングとディスパッチング』を参照してください。

第 3 章 計画

WebSphere Real Time for Linux をインストールする前に、このセクションを読んでください。

•
• 『サポート対象の環境』
•
• 10 ページの『考慮事項』

マイグレーション

ご使用の標準的な Java アプリケーションを、変更せずに WebSphere Real Time for Linux 上で使用することも可能です。

サポート対象の環境

IBM WebSphere Real Time for Linux は、特定のハードウェア・プラットフォームとオペレーティング・システムでサポートされています。

IBM WebSphere Real Time for Linux

以下のプラットフォーム・アーキテクチャーがサポートされています。

- Intel アーキテクチャー、32 ビット (IA-32)
  - Pentium 4
  - Pentium Xeon
  - Pentium M
  - Pentium D および同等のもの
- AMD64/EM64T
- IBM POWER® 32
- IBM POWER 64

注: Pentium 3 ハードウェアは、現在はサポートされていません。

以下のオペレーティング・システムがサポートされています。

表 1. テスト済みの Linux 環境

ハードウェア	**IA-32 32 ビット**	**AMD64/EM64T 64 ビット**	
SDK アドレス・スペース	**32-bit**	**32-bit**	**64-bit**
RHEL 5 Update 7	あり	あり	あり
RHEL 6 Update 1	あり	あり	あり
SLES 11 Service Pack 2	あり	あり	あり
Ubuntu 8.04	あり	あり	あり

表 1. テスト済みの Linux 環境 (続き)

ハードウェア	IA-32 32 ビット	AMD64/EM64T 64 ビット	
Ubuntu 10.04	あり	あり	あり

注: SLES 9、SLES 10、および RHEL 4 はサポートされていません。

考慮事項

WebSphere Real Time for Linux の使用時には、いくつかの要因に配慮する必要があります。

- 可能な場合には、複数のリアルタイム JVM を同一のシステムで実行しないようにしてください。複数のリアルタイム JVM を実行すると、ガーベッジ・コレクターが複数実行されるためです。各 JVM は、他の JVM のメモリー域について認識しません。結果として、JVM 間で GC サイクルと休止時間を調整することはできません。つまり、ある JVM が別の JVM の GC パフォーマンスに悪影響を及ぼす可能性があります。 複数の JVM を使用する必要がある場合は、**taskset** コマンドを使用して、各 JVM が特定のプロセッサーのサブセットにバインドされるようにしてください。
- 以前のリリースの WebSphere Real Time for Linux でプリコンパイル済みコードとクラスの保管に使用された共有キャッシュは、このリリースの WebSphere Real Time for Linux で使用されるキャッシュとは互換性がありません。以前のキャッシュの内容を再生成する必要があります。
- 共有クラス・キャッシュを使用する際、そのキャッシュの名前は 53 文字以下にする必要があります。

第 4 章 WebSphere Real Time for Linux のインストール

この製品をインストールするには、以下の手順に従ってください。

- 『インストール・ファイル』
- 『InstallAnywhere パッケージからのインストール』
  - 13 ページの『手動インストールの実行』
  - 14 ページの『自動インストールの実行』
  - 15 ページの『既知の問題と制約』
- 16 ページの『パスの設定』
- 17 ページの『クラスパスの設定』
- 18 ページの『インストール済み環境のテスト』
- 19 ページの『WebSphere Real Time for Linux のアンインストール』

インストール・ファイル

以下のインストール・ファイルが必要です。

IBM WebSphere Real Time for Linux は 2 種類の InstallAnywhere パッケージで提供されます。

インストール可能なパッケージ

インストール可能なパッケージでは、システムが構成されます。例えば、プログラムが環境変数を設定する場合があります。

- `wrt-3.0-0.0-linux-<arch>-sdk.bin`
- `wrt-3.0-0.0-linux-<arch>-jre.bin`

アーカイブ・パッケージ

これらのパッケージでは、ファイルはシステムに解凍されますが、構成は実行されません。

- `wrt-3.0-0.0-linux-<arch>-sdk-archive.bin`
- `wrt-3.0-0.0-linux-<arch>-jre-archive.bin`

注: <arch> はプラットフォーム・アーキテクチャー (x86_32 または x86_64) です。

InstallAnywhere パッケージからのインストール

これらのパッケージには、インストール・オプションの設定をガイドする対話式プログラムが含まれています。このプログラムは、グラフィカル・ユーザー・インターフェースとして実行することも、システム・コンソールから実行することもできます。

始める前に

お使いのシステムに、次の両方の共有ライブラリーがインストールされている必要があります。

- GNU C ライブラリー V2.3 (glibc)
- `libstdc++.so.5`

`libstdc++.so.5` 共有ライブラリーがない場合、インストール時に次のエラーを含む Java コア・ダンプが生成される場合があります。

```
JVMJ9VM011W Unable to load j9dmp24: libstdc++.so.5: cannot open shared object file:
No such file or directory
JVMJ9VM011W Unable to load j9gc24: libstdc++.so.5: cannot open shared object file:
No such file or directory
JVMJ9VM011W Unable to load j9vrb24: libstdc++.so.5: cannot open shared object file:
No such file or directory
```

インストール可能パッケージをインストールする場合、システムに rpm-build ツールがインストールされている必要があります。インストールされていない場合は、インストール・プログラムで新しいパッケージを RPM データベースに登録することができません。 rpm-build ツールがインストールされているかどうかを調べるには、次のコマンドを入力します。

```
rpm -q rpm-build
```

このタスクについて

InstallAnywhere のパッケージは、ファイル拡張子が `.bin` になっています。

パッケージには次の 2 つのタイプがあります。

インストール可能
: これらのパッケージをインストールすると、環境変数の設定などの、システムの構成も行われます。

アーカイブ
: これらのパッケージをシステムにインストールすると、ファイルが抽出されますが構成は一切行われません。

手順

- パッケージを対話式にインストールするには、手動インストールを実行します。
- ユーザーとの追加的な対話を行わずにパッケージをインストールするには、自動インストールを実行します。 インストールするシステムの数が多い場合には、こちらのオプションを選ぶと便利な場合があります。
- インストール・プロセスが完了したら、このセクションで説明する構成手順に従い、パスや CLASSPATH 環境変数の設定などを行ってください。

タスクの結果

製品がインストールされます。

注: Ctrl+C を押すなどのインストール・プロセスを中断する操作は行わないようにしてください。プロセスが中断されると、製品の再インストールが必要になる場合があります。詳しくは、15 ページの『中断されたインストール』を参照してください。

インストール可能パッケージを使用している場合には、問題が発見されたことを示すメッセージが表示される可能性があります。アーカイブ・パッケージのインストール時には、メッセージは生成されません。インストール可能パッケージを使用している場合に表示されるメッセージの例を、次のリストに示します。

The installer cannot run on your configuration. It will now quit.
このエラー・メッセージは、ユーザー ID がインストール・プロセス実行の権限を持っていない場合に表示されます。処理を継続できないため、インストール・プログラムは終了します。この問題を修正するには、root 権限を持つユーザー ID を使用して、再びインストールを開始してください。

An RPM package is already installed. Uninstall the package before proceeding.
このメッセージは、RPM パッケージが既にインストールされていることを示します。処理を継続できないため、インストール・プログラムは終了します。この問題を修正するには、処理を先に進める前に RPM パッケージをアンインストールしてください。

手動インストールの実行

InstallAnywhere パッケージから、製品を対話式にインストールします。

始める前に

インストール・プロセスを始める前に、以下の条件を確認してください。

- 既に RPM パッケージから WebSphere Real Time for Linux をインストールしている場合には、手順を進める前にこのパッケージをアンインストールする必要があります。
- root 権限のあるユーザー ID を持っている必要があります。

手順

1. 一時ディレクトリーにインストール・パッケージ・ファイルをダウンロードします。
2. 一時ディレクトリーに移動します。
3. シェル・プロンプトに *./package* と入力して、インストール・プロセスを開始します。ここで、*package* はインストールするパッケージの名前です。
4. インストーラー・ウィンドウに表示されているリストから言語を選択し、「**次へ**」をクリックします。選択可能な言語のリストは、お使いのシステムのロケール設定に基づくものです。
5. 使用許諾契約書を読みます。スクロール・バーを使用して許諾契約書の文章の最後まで読んでください。インストールを先に進めるには、使用許諾契約書に同意する必要があります。条件に同意するには、ラジオ・ボタンを選択し、「**OK**」をクリックします。

   注: 使用許諾契約書の文章を最後まで読むと、使用許諾契約書に同意するラジオ・ボタンを選択することができるようになります。
6. インストールするターゲット・ディレクトリーを選択するよう求められます。デフォルトのディレクトリーにインストールしない場合は、ブラウザー・ウィンド

ウを使用して、「**選択 (Choose)**」をクリックし、別のディレクトリーを選択します。インストール・ディレクトリーを選択したら、「**次へ**」をクリックして手順を続行します。

7. これまでに選択した内容を再確認するよう求められます。選択内容を変更する場合は、「**前へ**」をクリックします。選択内容が正しければ、「**インストール**」をクリックしてインストールを進めます。
8. インストール・プロセスが完了したら、「**完了**」をクリックして終了します。

自動インストールの実行

インストールするシステムが複数あり、使用するインストール・オプションが決まっている場合には、自動インストール手順を使用することをお勧めします。一度手動インストール手順を使用してインストールし、その結果生成された応答ファイルを使用して、ユーザーとの追加的な対話を行わずにそれ以降のインストールを実行します。

手順

1. 手動インストールを完了させて、応答ファイルを作成します。以下の方法のいずれかを使用します。
   - GUI を使用して、インストール・プログラムで応答ファイルを作成するように指定します。応答ファイルは `installer.properties` という名前で、インストール・ディレクトリーに作成されます。
   - コマンド・ラインを使用して、手動インストールのコマンドに `-r` オプションを追加し、応答ファイルへの絶対パスを指定します。例えば、次のようになります。

     ```
     ./package -r /path/installer.properties
     ```

   応答ファイルの内容例:

   ```
   INSTALLER_UI=silent
   USER_INSTALL_DIR=/my_directory
   ```

   この例では、 */my_directory* がインストール時に選択したターゲット・インストール・ディレクトリーです。
2. オプション: 必要に応じて、応答ファイルを編集してオプションを変更します。

   注: アーカイブ・パッケージには、次のような既知の問題があります。応答ファイルを使用したインストールは、応答ファイル中のディレクトリーを変更した場合でも、デフォルト・ディレクトリーを使用します。デフォルト・ディレクトリーに以前のインストールが存在している場合、上書きされます。

   それぞれ異なるインストール・オプションの複数の応答ファイルを作成する場合、 *myfile*`.properties` という形式で、それぞれの応答ファイルに固有の名前を指定してください。
3. オプション: ログ・ファイルを生成します。 サイレント・インストールを実行しているため、インストール・プロセスの終了時に、状況メッセージが表示されることはありません。インストールの状況を記録したログ・ファイルを生成するには、以下の手順を実行します。
   a. 次のコマンドを使用して、必須のシステム・プロパティーを設定します。

```
export _JAVA_OPTIONS="-Dlax.debug.level=3 -Dlax.debug.all=true"
```

b. ログの出力をコンソールに送信するために、次の環境変数を設定します。

```
export LAX_DEBUG=1
```

4. パッケージ・インストーラーを実行する際に、**-i** silent オプションを指定し、さらに **-f** オプションで応答ファイルを指定して、自動インストールを開始します。 例えば、次のように入力します。

```
./package -i silent -f /path/installer.properties 1>console.txt 2>&1
./package -i silent -f /path/myfile.properties 1>console.txt 2>&1
```

プロパティー・ファイルの指定には、完全修飾パスと相対パスのどちらでも使用できます。これらの例では、1>console.txt 2>&1 の文字列で、標準エラー出力ストリームと標準出力ストリームからのインストール・プロセスに関する情報を、現行ディレクトリーの console.txt ログ・ファイルにリダイレクトしています。インストールに問題があると思われる場合には、このログ・ファイルを調べてください。

注: インストール・ディレクトリーに複数の応答ファイルがある場合、デフォルトの応答ファイルである installer.properties が使用されます。

中断されたインストール

パッケージ・インストーラーがインストール中に予期せず停止した場合 (例えば、ユーザーが Ctrl+C を押した場合)、そのインストールは壊れており、製品のアンインストールや再インストールはできません。アンインストールや再インストールを行おうとすると、「Fatal Application Error」というメッセージが表示される可能性があります。

このタスクについて

この問題を解決するには、以下の手順に従って、ファイルを削除して再インストールを行ってください。

手順

1. レジストリー・ファイル /var/.com.zerog.registry.xml を削除します。
2. インストールが置かれるディレクトリーが作成されている場合には、それを削除します。例: /opt/IBM/javawrt3[_64]/
3. インストール・プログラムを再び実行します。

既知の問題と制約

InstallAnywhere パッケージには、いくつかの既知の問題と制約があります。

- システムに共有ライブラリー libstdc++.so.5 がない場合、インストールは失敗し、Java コア・ダンプが生成されます。詳しくは、11 ページの『InstallAnywhere パッケージからのインストール』を参照してください。
- インストール・パッケージの GUI は、Orca 画面読み上げプログラムに対応していません。 GUI の代わりとして、自動インストール・モードを使用することができます。
- インストール終了後に、 *./package* と入力して再度プログラムを開始した場合、プログラムで以下のメッセージが表示されます。

```
ENTER THE NUMBER OF THE DESIRED CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:
```

Enter を押してデフォルトを選択すると、プログラムは応答しなくなります。数字を入力してから、Enter を押してください。

- パッケージをインストールしてから、異なるモードで再度インストールしようとした場合 (例えば、コンソール・モードやサイレント・モード)、以下のエラー・メッセージが表示される場合があります。

```
Invocation of this Java Application has caused an InvocationTargetException.
This application will now exit
```

GUI モードを使用してインストールを行った後、コンソール・モードで再びインストール・プログラムを実行した場合は、このメッセージは表示されません。アンインストール・オプションを選択するプログラムを実行していて、このエラーが表示された場合 (インストール可能パッケージのみ) は、代わりに、19 ページの『WebSphere Real Time for Linux のアンインストール』の説明に従って **./_uninstall/uninstall** コマンドを使用してください。

インストール可能パッケージのみに関する問題

- 既存のインストール済み環境を、InstallAnywhere パッケージを使用してアップグレードすることはできません。WebSphere Real Time for Linux をアップグレードするには、まず以前のバージョンをアンインストールする必要があります。
- 異なるインストール・ディレクトリーを使用する場合でも、同一システム上に WebSphere Real Time for Linux の同一バージョンの 2 つの異なるインスタンスを同時にインストールすることはできません。例えば、/previous ディレクトリーに WebSphere Real Time for Linux V3 を保持しながら、同時に /current ディレクトリーに WebSphere Real Time for Linux サービスを新たにインストールすることはできません。インストール・プログラムはバージョン番号をチェックします。プログラムが、同じバージョン番号の既存のパッケージを発見すると、ユーザーは既存のパッケージのアンインストールを求められます。
- パッケージがインストールされた後で、GUI を使用して再びパッケージ・インストーラーを実行した場合、パッケージのアンインストールを選択することができます。このアンインストール・オプションは、自動インストール・モードでは使用できません。自動インストール・モードで再度パッケージ・インストーラーを実行した場合、プログラムは一切動作を行いません。

アーカイブ・パッケージのみに関する問題

- 応答ファイルのインストール・ディレクトリーを変更し、その応答ファイルを使用して自動インストールを実行した場合、インストール・プログラムは新しいインストール・ディレクトリーを無視し、代わりにデフォルトのディレクトリーを使用します。デフォルト・ディレクトリーに以前のインストールが存在している場合、上書きされます。

パスの設定

PATH 環境変数を設定してある場合、シェル・プロンプトで名前を入力することにより、アプリケーションまたはプログラムを実行できます。

このタスクについて

注: このセクションの説明に従って **PATH** 環境変数を変更すると、ご使用のパスにある既存の Java 実行可能ファイルがすべてオーバーライドされます。

ツールへのパスは、その都度、ツール名の前にパスを入力して指定します。 例えば、SDK が /opt/IBM/javawrt3[_64]/ にインストールされている場合は、シェル・プロンプトで次のコマンドを入力すると、*myfile.java* という名前のファイルをコンパイルすることができます。

```
/opt/IBM/javawrt3[_64]/bin/javac myfile.java
```

毎回絶対パスを入力しないで済むようにするには、

1. ホーム・ディレクトリーにあるシェル始動ファイル (ご使用のシェルによりますが、通常は、**.bashrc**) を編集し、**PATH** 環境変数に絶対パスを追加してください。例えば、次のようにします。

   ```
   export PATH=/opt/IBM/javawrt3[_64]/bin:/opt/IBM/javawrt3[_64]/jre/bin:$PATH
   ```

2. 再びログオンするか、または、更新したシェル・スクリプトを実行して、新しい **PATH** 設定をアクティブにします。
3. ファイルを **javac** ツールでコンパイルします。 例えば、*myfile.java* というファイルをコンパイルするには、シェル・プロンプトで、次のように入力します。

   ```
   javac -Xgcpolicy:metronome myfile.java
   ```

   PATH 環境変数を指定することにより、Linux は、どの現行ディレクトリーからでも **javac** ツール、**java** ツール、および **javadoc** ツールなどの実行可能ファイルを見つけられるようになります。ご使用のパスの現行値を表示するには、コマンド・プロンプトで次のコマンドを入力します。

   ```
   echo $PATH
   ```

次のタスク

CLASSPATH 環境変数を設定する必要があるかどうか判別するためには、『クラスパスの設定』を参照してください。

クラスパスの設定

CLASSPATH 環境変数により、SDK ツール (**java**、**javac**、および **javadoc** ツールなど) に Java クラス・ライブラリーの場所を通知します。

このタスクについて

CLASSPATH 環境変数は、以下のいずれかの条件が該当する場合にのみ明示的に設定してください。

- 例えば、独自に作成したものなど異なるライブラリーまたはクラス・ファイルが必要であり、それが現行ディレクトリーにない場合。
- bin および lib ディレクトリーの位置を変更して、同じ親ディレクトリーが既にない場合。
- 同一システム上に、異なるランタイム環境を使用する複数のアプリケーションを作成し実行する予定の場合。

CLASSPATH の現行値を表示するには、シェル・プロンプトで次のコマンドを入力します。

```
echo $CLASSPATH
```

別にインストール済みの他のバージョンも含めて、異なるランタイム環境を使用する複数のアプリケーションを作成し実行する場合、**CLASSPATH** および **PATH** をアプリケーションごとに明示的に設定する必要があります。複数のアプリケーションを同時に実行し、異なるランタイム環境を使用する場合は、それぞれのアプリケーションを固有のシェルで実行する必要があります。

一度に 1 つのバージョンの Java のみを実行する場合は、シェル・スクリプトを使用して、異なるランタイム環境を切り替えることができます。

次のタスク

インストールが正常に行われたことを検証するには、『インストール済み環境のテスト』を参照してください。

インストール済み環境のテスト

インストールが正常に行われたかどうかを検査するには、**-version** オプションを使用します。

このタスクについて

Java のインストール済み環境は、リアルタイム JVM から構成されています。

手順

以下の手順に従って、インストール済み環境をテストします。

1. リアルタイム JVM のバージョン情報を表示するには、シェル・プロンプトで以下のコマンドを入力します。

   ```
   java -Xgcpolicy:metronome -version
   ```

   このコマンドが正常に実行された場合は、以下のようなメッセージが返されます。

   ```
   java version "1.7.0"
   WebSphere Real Time V3(build pxi3270-20110428_04)
   IBM J9 VM (build 2.6, JRE 1.7.0 Linux x86-32 20110427_81014 (JIT enabled, AOT
   enabled)
   J9VM - R26_head_20110426_2022_B81001
   JIT  - r11_20110426_19388
   GC   - R26_head_20110426_1548_B80973
   J9CL - 20110427_81014)
   JCL - 20110427_03 based on Oracle 7b145
   ```

   注: バージョン情報は正確ですが、プラットフォーム・アーキテクチャーと日付はこの例に示されているものとは異なっている可能性があります。この日付ストリングの形式は yyyymmdd です。後ろにそのコンポーネントに固有の追加情報が続く場合もあります。

WebSphere Real Time for Linux のアンインストール

WebSphere Real Time for Linux を削除するためのプロセスは、使用したインストールのタイプによって異なります。

始める前に

InstallAnywhere インストール可能パッケージの場合は、ルート権限を持つユーザー ID が必要です。

このタスクについて

InstallAnywhere アーカイブ・パッケージにアンインストール・プロセスはありません。アーカイブ・パッケージをシステムから削除するには、パッケージのインストール時に選択したターゲット・ディレクトリーを削除してください。
InstallAnywhere インストール可能パッケージで製品をアンインストールするには、以下のステップで説明するように、コマンドを使用するか、インストール・プログラムを再度実行します。

手順

- オプション: **uninstall** コマンドを使用して手動でアンインストールします。
  1. インストールがあるディレクトリーに移動します。 例えば、次のように入力します。

     ```
     cd /opt/IBM/javawrt3
     ```

  2. コマンド `./_uninstall/uninstall` を入力して、アンインストール・プロセスを開始します。
- オプション: アンインストール・プログラムが簡単に見つからない場合は、代わりに手動インストールを実行できます。インストール・プログラムでは、製品がすでにインストールされていることが検出され、その後で前のインストールをアンインストールすることができます。

第 5 章 IBM WebSphere Real Time for Linux アプリケーションの実行

リアルタイム・アプリケーションを実行する際に役立つ重要な情報を説明します。

- 『スレッドのスケジューリングとディスパッチング』
-
- 25 ページの『Metronome ガーベッジ・コレクターの使用』

スレッドのスケジューリングとディスパッチング

Linux オペレーティング・システムでは、さまざまなスケジューリング・ポリシーがサポートされています。デフォルトの一般的タイム・シェアリング・スケジューリング・ポリシーは SCHED_OTHER であり、ほとんどのスレッドで使用されます。SCHED_RR および SCHED_FIFO は、リアルタイム・アプリケーションのスレッドで使用できます。WebSphere Real Time for Linux で使用されるのは SCHED_OTHER および SCHED_RR のみです。

プロセッサーで次に実行される実行可能スレッドは、カーネルが決定します。カーネルでは、実行可能スレッドのリストが保持されています。このカーネルは優先順位が最も高いスレッドを検索し、そのスレッドを次に実行するスレッドとして選択します。

スレッドの優先順位とポリシーは、以下のコマンドを使用してリストすることができます。

```
ps -emo pid,ppid,policy,tid,comm,rtprio,cputime
```

policy には以下のいずれかを指定します。

- TS。SCHED_OTHER を表します。
- RR。SCHED_RR を表します。
- FF。SCHED_FIFO を表します。
- -。報告されるポリシーはありません。

出力は以下の例のようになります。

```
  PID  PPID POL   TID COMMAND         RTPRIO     TIME
31531 30800 -       - java                 - 00:00:13
    -     - RR  31531 -                    6 00:00:00
    -     - RR  31532 -                    6 00:00:13
    -     - RR  31533 -                    6 00:00:00
    -     - RR  31538 -                    6 00:00:00
    -     - RR  31539 -                    6 00:00:00
    -     - RR  31540 -                    6 00:00:00
    -     - RR  31541 -                    6 00:00:00
    -     - RR  31542 -                    6 00:00:00
    -     - RR  31543 -                    6 00:00:00
    -     - RR  31544 -                    6 00:00:00
    -     - RR  31545 -                    6 00:00:00
    -     - RR  31546 -                    6 00:00:00
```

この出力は、Java プロセス、およびポリシーが SCHED_RR で優先順位が 6 の多数のスレッドを示しています。

現行のスケジューリング・ポリシーを照会する場合は、**sched_getscheduler** を使用するか、前述の例に示されている **ps** コマンドを使用してください。

プロセスについて詳しくは、 40 ページの『一般的なデバッグ手法』を参照してください。

通常の Java スレッドの優先順位およびポリシー

通常の Java スレッド (つまり java.lang.Thread オブジェクトとして割り振られたスレッド) は、デフォルトのスケジューリング・ポリシー SCHED_OTHER を使用します。 WebSphere Real Time for Linux V3 Service Refresh 1 から、SCHED_RR スケジューリング・ポリシーを使用して通常の Java スレッドを実行することができます。

デフォルトでは、Java スレッドはデフォルトの SCHED_OTHER ポリシーを使用して実行されます。このポリシーは、Java スレッドをオペレーティング・システム優先順位 0 にマップします。

SCHED_RR ポリシーを使用すると、アプリケーションを細かく制御でき、Java スレッドのリアルタイム・パフォーマンスを向上させることができます。 SCHED_RR ポリシーを使用して Java が開始されると、メイン・スレッドの優先順位およびポリシーが JVM によって検出されます。 JVM はそれに応じて、優先順位およびポリシーのマッピングを変更します。 Java スレッドはすべて、メイン・スレッドと同じオペレーティング・システム優先順位で実行されます。 SCHED_RR には 1 から 99 の範囲で優先順位を割り当てることができますが、WebSphere Real Time for Linux V3 に適した SCHED_RR の優先順位は 1 から 10 です。優先順位を 10 より高く設定すると、メイン・スレッドの優先順位が 10 まで下げられ、マッピングも値 10 に基づいて適用されることになります。

コマンド行でプロセスのリアルタイム・スケジューリング・プロパティーを変更する方法の 1 つは、**chrt** コマンドを使用することです。以下の例では、オペレーティング・システム優先順位 6 で SCHED_RR スケジューリング・ポリシーを使用するように、メインの Java スレッドの優先順位が設定されています。

```
chrt -r 6 java
```

必要に応じて、優先順位を変更できるようにご使用のシステムを構成してください。詳しくは、 23 ページの『優先順位の変更を可能にするシステム構成』を参照してください。

表 2. Java 優先順位とオペレーティング・システム優先順位

Java 優先順位	スレッドの Java 優先順位値	オペレーティング・システム優先順位
1	MIN_PRIORITY	6
2		6
3		6
4		6

表 2. Java 優先順位とオペレーティング・システム優先順位 (続き)

Java 優先順位	スレッドの Java 優先順位値	オペレーティング・システム優先順位
5	NORM_PRIORITY (デフォルト)	6
6		6
7		6
8		6
9		6
10	MAX_PRIORITY	6

メイン Java スレッドに関連付けられたスレッドはすべて、同じオペレーティング・システム優先順位で実行されます。

`chrt -r 11 java` コマンドを実行した場合の結果は、`chrt -r 10 java` を実行した場合と同じです。これは、JVM スレッドが使用する優先順位マッピングに、10 を超える優先順位を適用できないためです。ただし、JVM を起動して JVM の終了を待機するスレッドの優先順位は 11 のままになります。

`chrt -f 6 java` コマンドを使用しようとすると、JVM からエラー・メッセージが出されます。これは、WebSphere Real Time for Linux V3 で SCHED_FIFO がサポートされていないためです。

chrt コマンドについて詳しくは、http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaai/realtime/liaairtchrt.htm を参照してください。

優先順位の変更を可能にするシステム構成

デフォルトでは、Linux の root 以外のユーザーがスレッドまたはプロセスの優先順位を上げることはできません。優先順位を変更できるようにシステム構成を変更するには、Pluggable Authentication Modules (PAM) for Linux の `pam_limits` モジュールを使用します。

chrt ユーティリティーを使用してスレッドまたはプロセスの優先順位を変更できない場合、通常は以下のメッセージが表示されます。

```
sched_setscheduler: Operation not permitted
```

最近の Linux カーネルでは、`pam_limits` モジュールを使用して、優先順位を変更できるようにシステム構成を変更することができます。このモジュールを使用すると、制限構成ファイルでシステム・リソースでの制限を構成できます。デフォルト・ファイルは `/etc/security/limits.conf` です。

`/etc/security/limits.conf` ファイルの項目の形式は以下のとおりです。

```
<domain> <type> <item> <value>
```

各部の説明:

<domain> は以下のいずれかです。

- リソースでの制限を変更できる、システム上のユーザー名。

- グループ名 (構文は `@group`)。このグループに属しているメンバーがリソースでの制限を変更できます。

- ワイルドカード「`*`」。すべてのユーザーまたはグループがリソースでの制限を変更できることを示します。

<type> は以下のいずれかです。

- `hard`。カーネルによりハード制限が強制されます。

- `soft`。ソフト制限が適用されます。ソフト制限は、ハード制限で指定された範囲内で変更が可能です。

- ダッシュ「`-`」。ハード制限およびソフト制限を示します。

<item> は以下のいずれかです。

- リソース。リアルタイム優先順位には `rtprio` を使用します。

<value> は以下のいずれかです。

- 制限値。リアルタイム優先順位設定の上限を示す値を 1 から 100 の範囲で指定します。

例えば、

```
* - rtprio 100
```

と指定すると、すべてのユーザーが **chrt** その他のメカニズムを使用してリアルタイム・プロセスの優先順位を変更できるようになります。

デフォルトでは、root ユーザーは制限なしでリアルタイム優先順位を上げることができます。 root に制限を適用するには、root ユーザーを明示的に指定する必要があります。構成ファイル内のグループ制限およびワイルドカード制限は、root ユーザーには適用されません。

このファイルで個々にユーザー制限を指定した場合、これらの制限はグループ制限よりも優先されます。

`limits.conf` を変更しても、直ちに有効にはなりません。構成変更を有効にするには、影響を受けるサービスを再始動するか、システムをリブートする必要があります。

Java 仮想マシン (JVM) のリアルタイム優先順位を上げる機能は、Linux カーネル 2.6.12 以前では使用できません。下の表に、いくつかの一般的な Linux ディストリビューションでこの機能がサポートされているかどうかを示します。

表 3. リアルタイム優先順位変更のサポート

Linux ディストリビューション	**Linux カーネル・バージョン**	**リアルタイム優先順位変更のサポート (あり/なし)**
Red Hat Enterprise Linux (RHEL) 4	2.6.9	なし
RHEL 5 以降	2.6.18 以降	あり
SUSE Linux Enterprise Server (SLES) 9	2.6.5-7	なし
SLES 10 以降	2.6.16 以降	あり

表 3. リアルタイム優先順位変更のサポート (続き)

Linux ディストリビューション	Linux カーネル・バージョン	リアルタイム優先順位変更のサポート (あり/なし)
Red Hat Enterprise MRG - 全バージョン	2.6.24 以降	あり
SUSE Linux Enterprise Real Time (SLERT) - 全バージョン	2.6.16 以降	あり
Ubuntu 5.10	2.6.12	なし
Ubuntu 6.06 以降	2.6.15 以降	あり

リアルタイム Linux システムで優先順位変更を有効にするには、`limits.conf` ファイルに示すように、`realtime` グループにユーザーを追加します。

2 次プロセスの起動

Java 仮想マシン (JVM) API の java.lang.Runtime.exec メソッドを使用すると、Java アプリケーションは別個のプロセスでコマンドを実行することが可能になります。

このメソッド呼び出しから、新しい java.lang.Process オブジェクトが作成されます。このオブジェクトを使用して新規プロセスを制御でき、またそのプロセスに関する情報を入手できます。

この目的のために、exec メソッドによっていくつかのスレッドが作成されます。IBM WebSphere Real Time for Linux では、手順の変更によって、リアルタイム環境でより決定論的な動作が可能になります。

Runtime.exec を呼び出すと、fork されるサブプロセスごとに「リーパー」スレッドが作成されます。リーパー・スレッドは、サブプロセスが終了するまで待機する唯一のスレッドです。サブプロセスが終了すると、リーパー・スレッドはそのサブプロセスの終了状況を記録します。リーパー・スレッドは新規プロセスを spawn し、Runtime.exec を最初に呼び出したスレッドと同じ優先順位を割り当てます。

spawn されたプロセスが別の WebSphere Real Time for Linux JVM であり、Linux リアルタイム・ポリシーおよび優先順位で実行されている別のメソッドによって Runtime.exec メソッドが呼び出された場合、新規仮想マシンのメイン・スレッドのポリシーおよび優先順位は、同じ Linux リアルタイム・ポリシーおよび優先順位にマップされます。この Java スレッド優先順位の範囲は 1 から 10 です。

また、このリーパー・スレッドにより、新規プロセスの `stdout` ストリームおよび `stderr` ストリームを listen する 2 つの新規スレッドが作成されます。 `stdout` と `stderr` のデータは、これらのスレッドで使用されるバッファーに保存されます。これらのバッファーは、spawn されたプロセスの存続期間を過ぎても維持されます。この持続性により、spawn されたプロセスで保持されていたリソースを、プロセスの終了直後にクリアすることが可能になります。

Metronome ガーベッジ・コレクターの使用

WebSphere Real Time for Linux では、標準的なガーベッジ・コレクターの代わりに、Metronome ガーベッジ・コレクターを使用します。

休止時間の制御

Metronome ガーベッジ・コレクター (GC) の休止時間は、各 Java プロセスごとに細かく調整できます。

デフォルトでは、Metronome GC は個々の休止ごとに 3 ミリ秒休止することになっており、これはクォンタと呼ばれます。完全なガーベッジ・コレクション・サイクルにはこの休止が多数必要で、これが分散して行われることで、アプリケーションを実行するのに必要な時間を確保します。この個々の休止時間の最大値は、**-Xgc:targetPauseTime** オプションで変更できます。例えば、**-Xgc:targetPauseTime=20** を指定して実行すると、GC は、個々の休止時間が 20 ミリ秒以下に設定されて動作するようになります。

IBM Monitoring and Diagnostics Tools for Java - Garbage Collection and Memory Visualizer (GCMV) を使用すると、アプリケーションの GC 休止時間をモニターすることができ、Java アプリケーションのパフォーマンス上の問題に関する診断と調整にも役立ちます。このツールは、以下のようなさまざまな種類のログのデータを解析し、プロットします。

- 詳細ガーベッジ・コレクション・ログ。
- **-Xtgc** パラメーターを使用して生成されたトレース・ガーベッジ・コレクション・ログ。
- システム・コマンド **ps**、**svmon**、または **perfmon** を使用して生成されたネイティブ・メモリー・ログ。

このセクションで示すグラフは GCMV で生成されたもので、ガーベッジ・コレクション・サイクルの目標休止時間を変更した場合の影響を示しています。それぞれのグラフは、アプリケーションの実行時間 (X 軸) に対する、Metronome ガーベッジ・コレクション・サイクル間の実際の休止時間 (Y 軸) をプロットしたものです。

注: GCMV は古い形式の詳細ガーベッジ・コレクション・ログにも対応しています。詳細 GC 出力を GCMV で分析する場合は、 **-Xgc:verboseFormat=deprecated** オプションを指定して出力を生成してください。詳しくは、 GC コマンド行オプションを参照してください。

デフォルトの目標休止時間が設定されているケースでは、詳細 GC 休止時間グラフは休止時間がおよそ 3 ミリ秒、あるいは 3 ミリ秒未満になっていることを示しています。

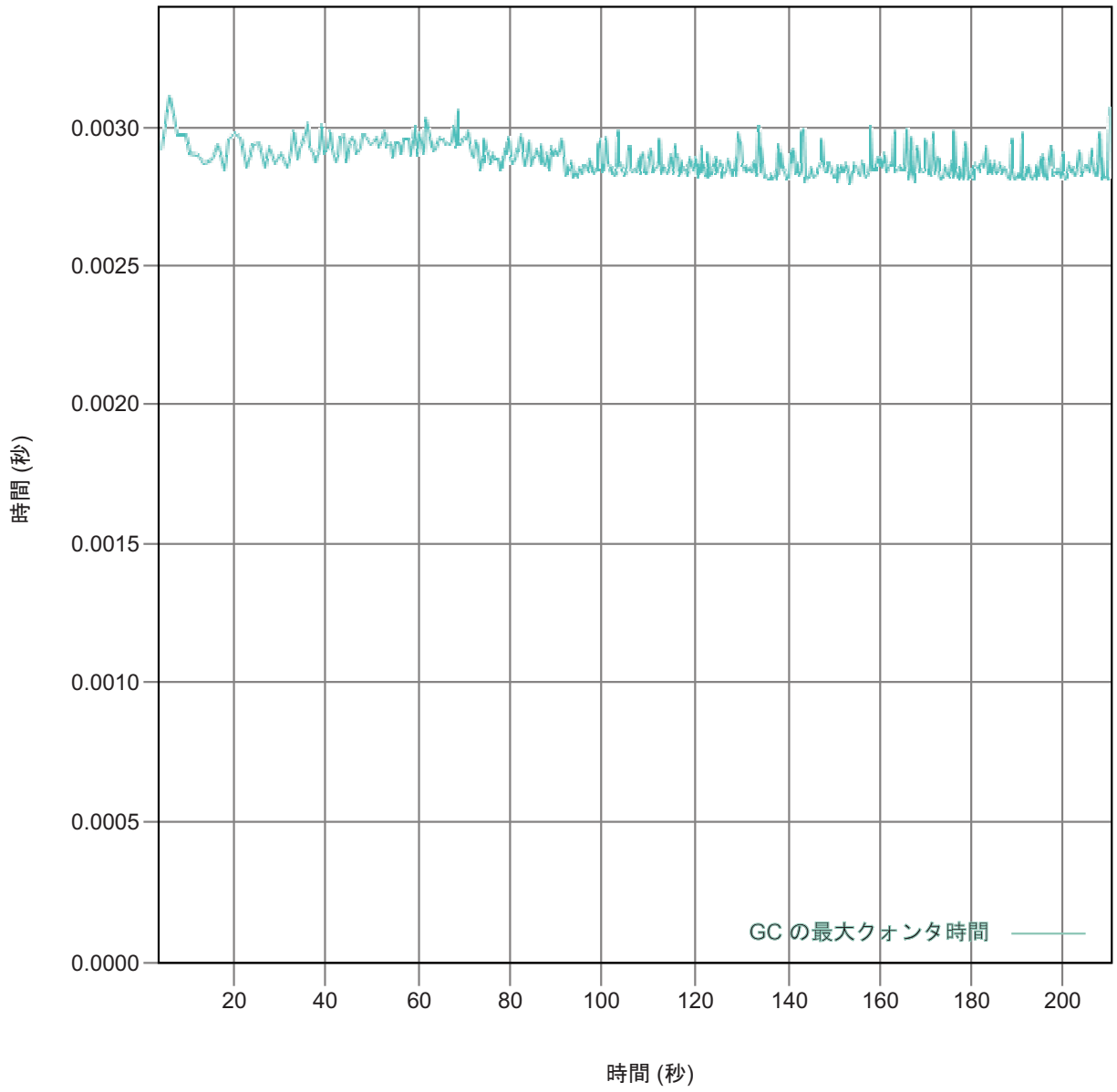

図 1. 目標休止時間がデフォルト (3 ミリ秒) に設定されている場合の実際のガーベッジ・コレクション休止時間

目標休止時間を 6 ミリ秒に設定したケースでは、詳細 GC 休止時間グラフは休止時間がおよそ 6 ミリ秒、あるいは 6 ミリ秒未満になっていることを示しています。

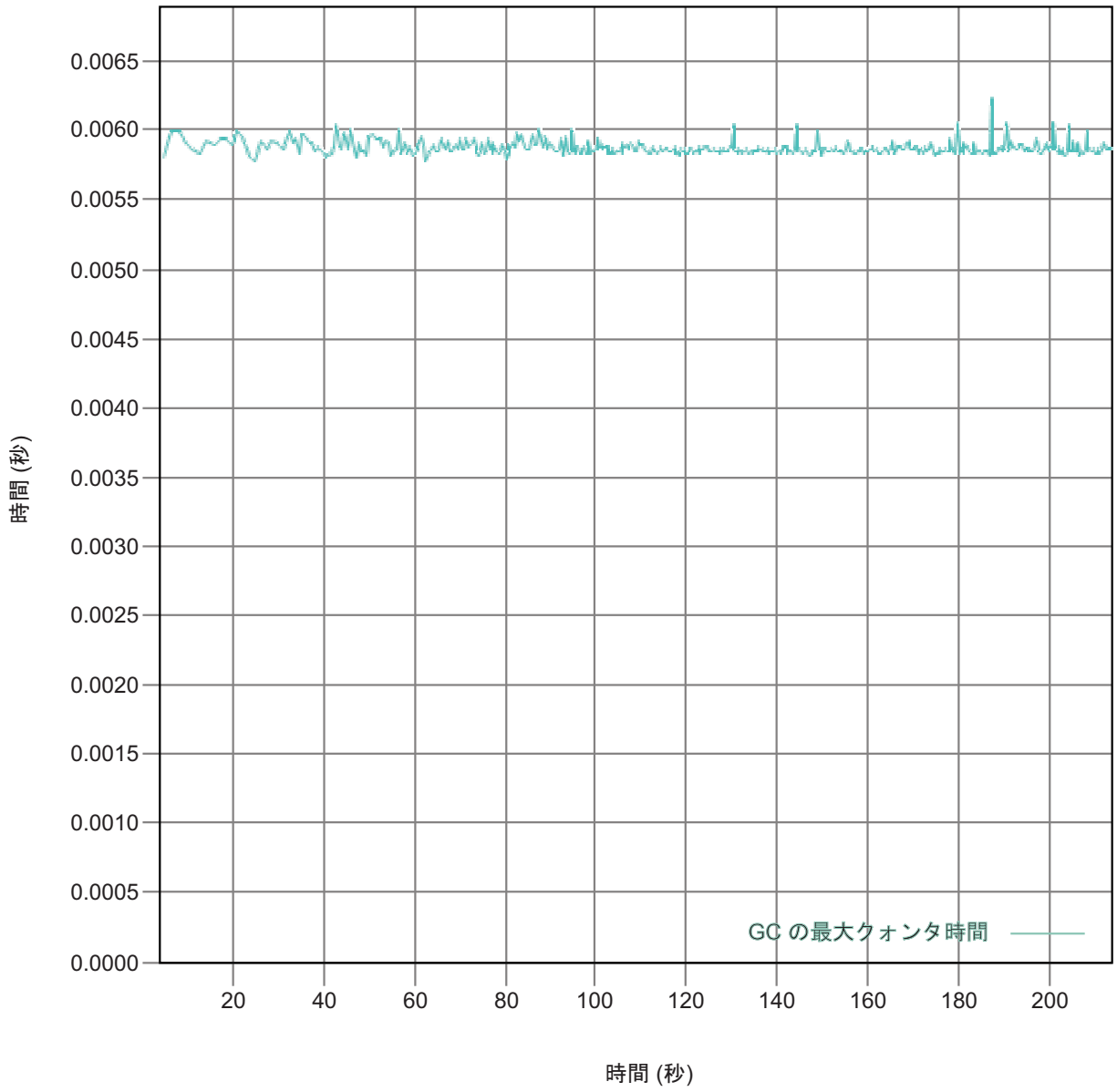

図 2. 目標休止時間が 6 ミリ秒に設定されている場合の実際の休止時間

目標休止時間を 10 ミリ秒に設定したケースでは、詳細 GC 休止時間グラフは休止時間がおよそ 10 ミリ秒、あるいは 10 ミリ秒未満になっていることを示しています。

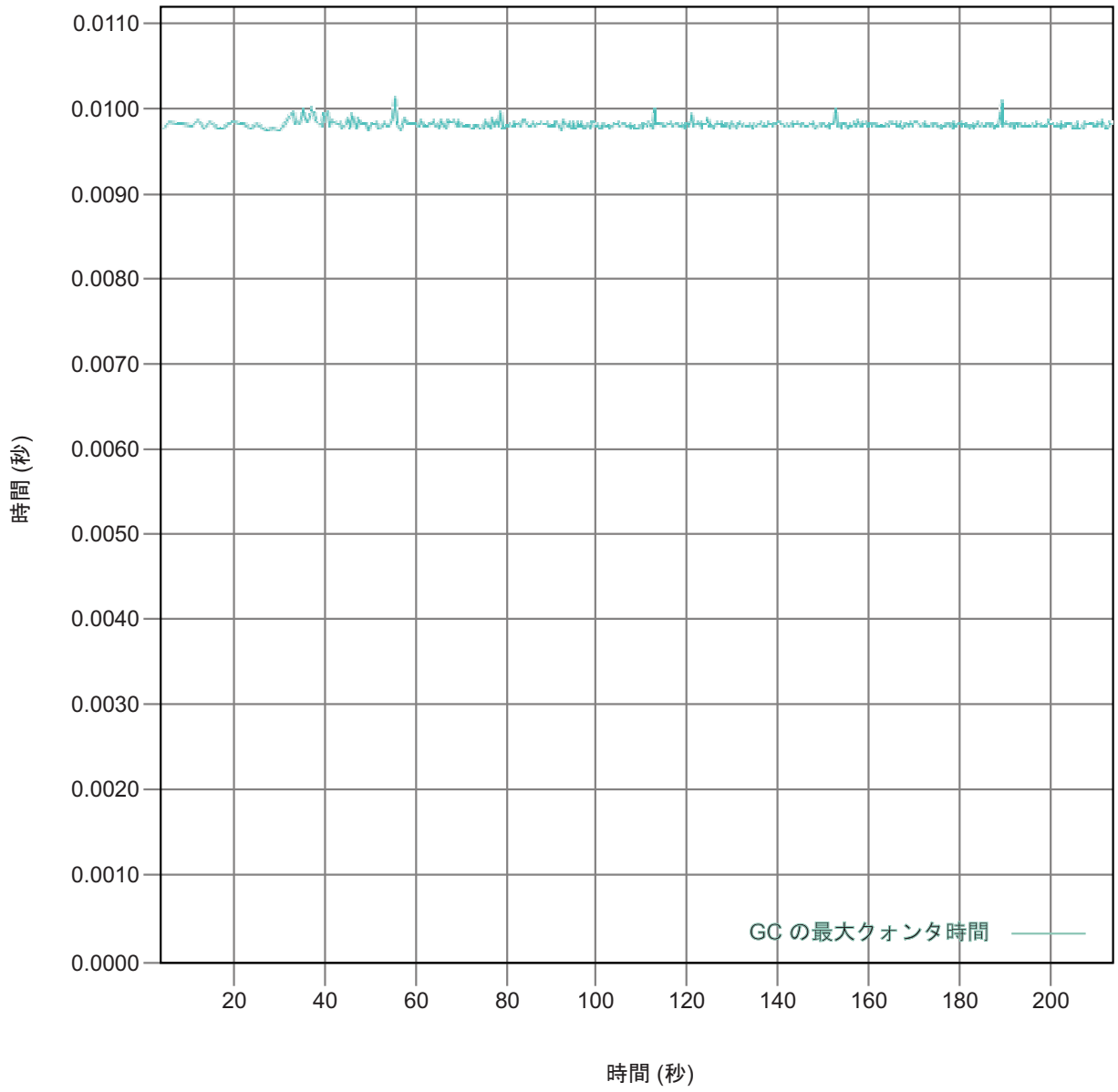

図 3. 目標休止時間が 10 ミリ秒に設定されている場合の実際の休止時間

目標休止時間を 15 ミリ秒に設定したケースでは、詳細 GC 休止時間グラフは休止時間がおよそ 15 ミリ秒、あるいは 15 ミリ秒未満になっていることを示しています。

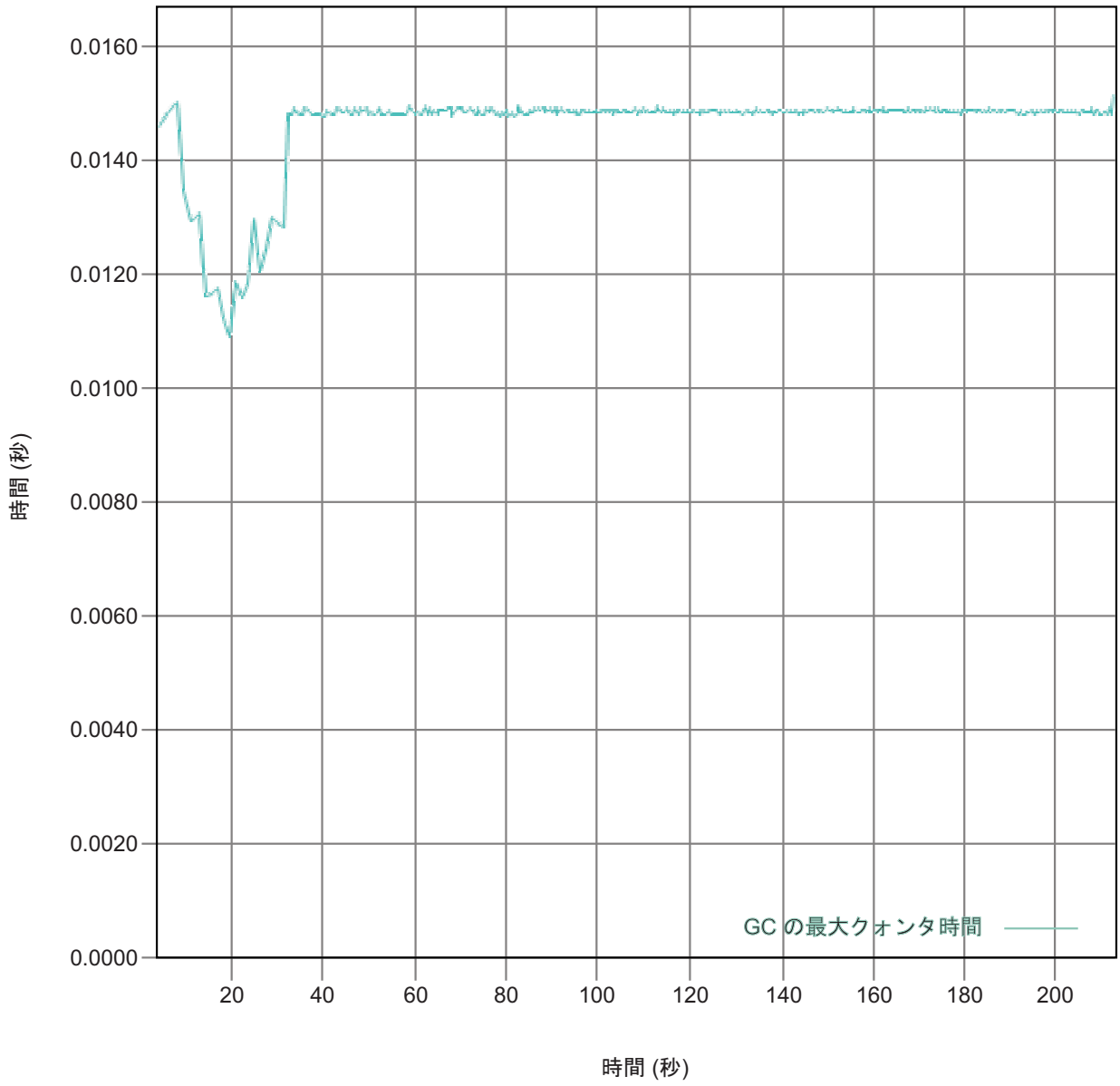

図 4. 目標休止時間が 15 ミリ秒に設定されている場合の実際の休止時間

プロセッサー使用率の制御

Metronome ガーベッジ・コレクターが使用可能な処理能力の量を制限することができます。

Metronome ガーベッジ・コレクターによるガーベッジ・コレクションを **-Xgc:targetUtilization=N** オプションを使用して制御することで、ガーベッジ・コレクターで使用される CPU 量を制限できます。

例えば、次のようにします。

```
java -Xgcpolicy:metronome -Xgc:targetUtilization=80 yourApplication
```

この例では、60 ミリ秒ごとに 80% の時間がアプリケーションの実行に使用されるように指定しています。残りの 20% の時間はガーベッジ・コレクションに使用されます。Metronome ガーベッジ・コレクターに十分なリソースが与えられている場合は、使用率レベルが保証されます。ガーベッジ・コレクションは、ヒープ内のフリー・スペース量が、動的に決定されたしきい値を下回ったときに開始されます。

Metronome ガーベッジ・コレクターの制限

このトピックでは、metronome GC ポリシーに影響を与える既知の問題や制限について説明します。

x86 プラットフォームでの AESNI サポート

x86 アーキテクチャーでの AESNI 命令のソフトウェア利用は、metronome GC ポリシーでは現在サポートされていません。

ガーベッジ・コレクション中の長い一時停止時間

まれなケースですが、ガーベッジ・コレクション時の休止が予想よりも長くなる場合があります。ガーベッジ・コレクション時は、ルート・スキャン・プロセスが使用されます。ガーベッジ・コレクターは、ヒープ内の既知のライブ参照から調べます。参照には以下が含まれます。

- アクティブ・スレッド呼び出しスタックのライブ参照変数。
- 静的参照。

アプリケーション・スレッド・スタックのすべてのライブ・オブジェクト参照を検索するために、ガーベッジ・コレクターはそのスレッドの呼び出しスタックにあるすべてのスタック・フレームをスキャンします。各アクティブ・スレッド・スタックは一度にスキャンされます。このため、スキャンは単一の GC 休止内で実行される必要があります。

その結果、スタックが非常に深いスレッドが複数ある場合は、コレクション・サイクル開始時のガーベッジ・コレクション休止時間が長くなるため、システム・パフォーマンスが予想よりも悪くなる可能性があります。

第 6 章 アプリケーションの開発

リアルタイム・アプリケーションの作成に関する重要な情報について、コードの例を示しながら説明します。

- 『サンプル・リアルタイム・ハッシュ・マップ』

サンプル・リアルタイム・ハッシュ・マップ

WebSphere Real Time for Linux には、IBM SDK for Java 7 の標準の HashMap よりも、put メソッドにおいてより一貫したパフォーマンスを提供する、HashMap と HashSet の実装が含まれています。

IBM 提供の標準 java.util.HashMap は、高スループット・アプリケーションに適しています。また、拡張する必要があるハッシュ・マップの最大サイズが分かっているアプリケーションにも役立ちます。可変サイズに拡張可能なハッシュ・マップが必要なアプリケーションの場合、使用に応じて、標準ハッシュ・マップではパフォーマンス上の問題が生じる可能性があります。標準ハッシュ・マップは、put メソッドを使用して新規項目をハッシュ・マップに追加する際の応答時間が優れています。ただし、ハッシュ・マップが満杯になると、より大きな補助ストレージを割り振る必要があります。このため、現在の補助ストレージ内の項目をマイグレーションする必要が生じます。ハッシュ・マップが大きい場合、put の実行時間も長くなる可能性があります。例えば、操作に数ミリ秒かかることがあります。

WebSphere Real Time for Linux には、サンプル・リアルタイム・ハッシュ・マップが含まれています。これは、標準 java.util.HashMap と同じ機能のインターフェースを提供しますが、はるかに一貫性のある put メソッドのパフォーマンスを可能にします。補助ストレージを作成して、ハッシュ・マップが満杯になったときにすべての項目をマイグレーションする代わりに、サンプル・ハッシュ・マップは追加の補助ストレージを作成します。この新しい補助ストレージは、ハッシュ・マップ内のその他の補助ストレージにチェーニングされます。チェーニングにより、初期段階では、空の補助ストレージが割り振られてその他の補助ストレージにチェーニングされる間、若干のパフォーマンスの低下が生じます。バッキング (補助ストレージへの保管) ハッシュ・マップが更新されると、すべての項目をマイグレーションする必要がある場合よりも高速になります。リアルタイム・ハッシュ・マップの欠点として、get、put、および remove の各操作が若干遅くなります。操作が遅くなるのは、ルックアップごとに、一回だけではなく、バッキング・ハッシュ・マップのセット全体を探索する必要があるためです。

リアルタイム・ハッシュ・マップを試行するには、`RTHashMap.jar` ファイルを、ブート・クラスパスの先頭に追加します。 WebSphere Real Time for Linux を `$WRT_ROOT` ディレクトリーにインストールしている場合は、以下のオプションを追加して、標準ハッシュ・マップの代わりにリアルタイム・ハッシュ・マップをアプリケーションで使用します。

```
-Xbootclasspath/p:$WRT_ROOT/demo/realtime/RTHashMap.jar
```

リアルタイム・ハッシュ・マップ実装のソース・ファイルおよびクラス・ファイルは、 `demo/realtime/RTHashMap.jar` ファイル内に含まれています。また、リアルタイム java.util.LinkedHashMap および java.util.HashSet 実装も提供されています。

第 7 章 パフォーマンス

WebSphere Real Time for Linux は、最高のスループット・パフォーマンスまたは最小のメモリー・フットプリントを実現させるためでなく、GC による休止を一貫して短くするために最適化されています。

認定されたハードウェア構成でのパフォーマンス

認定されたシステムは、WebSphere Real Time for Linux のパフォーマンス・ゴールを達成するために十分なクロック細分性とプロセッサー速度を備えています。例えば、過負荷状態でなくヒープ・サイズが十分なシステムで実行される、適切に作成されたアプリケーションでは通常、GC による休止が約 3 ミリ秒 (3.2 ミリ秒以下) になります。GC サイクル中、デフォルト環境設定が適用されるアプリケーションは、60 ミリ秒時間枠の移動中に経過する時間の 30% を超えて休止することはありません。どの 60 ミリ秒の期間内でも、GC による休止時間は、通常は合計で約 18 ミリ秒になります。

タイミングの変動性の軽減

標準 JVM での変動性の主要な原因は、ガーベッジ・コレクションによる休止です。WebSphere Real Time for Linux では、標準ガーベッジ・コレクター・モードで生じる可能性のある長時間の休止は、Metronome ガーベッジ・コレクターを使用することによって回避されます。 25 ページの『Metronome ガーベッジ・コレクターの使用』を参照してください。

JVM 間でのクラス・データの共用

クラス・データ共用は、メモリー・フットプリントを減らし、JVM 起動時間を改善するための、透過的な手法です。クラス・データ共用について詳しくは、 36 ページの『JVM 間でのクラス・データの共有』を参照してください。

圧縮参照

Metronome GC は、64 ビット・プラットフォーム上では圧縮参照と非圧縮参照の両方をサポートします。圧縮参照を使用する場合、JVM ではオブジェクト、クラス、スレッド、およびモニターへのすべての参照を 32 ビット値として格納します。圧縮参照を使用すると、多くのアプリケーションでパフォーマンスが向上します。これは、オブジェクトのサイズが小さくなり、ガーベッジ・コレクションの実行頻度が減少して、メモリー・キャッシュの使用効率が向上するためです。

注: 圧縮参照に使用できるヒープ・サイズは、約 28 GB に制限されています。圧縮参照について詳しくは、http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/understanding/mm_compressed_references.htmlを参照してください。

JVM 間でのクラス・データの共有

共有クラスのサポートは、**-Xrealtime** オプションを指定して実行している場合と、指定せずに実行している場合では同じです。

Java 仮想マシン (JVM) 間でのクラス・データの共有は、ディスク上のメモリー・マップ・キャッシュ・ファイルにデータを保管することによって行うことができます。複数の JVM がキャッシュを共有する場合、共有によって全体の仮想ストレージの使用量が削減されます。また、共有によって、キャッシュ作成後の JVM の起動時間も短縮されます。共有クラス・キャッシュは、実行中のどの JVM からも独立しており、削除されるまで存続します。

共有キャッシュには、以下を含むことができます。

- ブートストラップ・クラス
- アプリケーション・クラス
- クラスを記述するメタデータ
- Ahead-of-time (AOT) コンパイル済みコード

注: 非リアルタイム JVM によってリアルタイム共有クラス・キャッシュを削除することはできません。

第 8 章 セキュリティー

このセクションには、セキュリティーに関する重要な情報が含まれています。

共有クラス・キャッシュのセキュリティーの考慮事項

共有クラス・キャッシュは、キャッシュを管理しやすくし、ユーザビリティーを向上するように設計されていますが、デフォルトのセキュリティー・ポリシーは適切でない場合があります。

共有クラス・キャッシュを使用するときには、アクセスを制限してセキュリティーを強化できるようにするために、新規ファイルのデフォルト許可に配慮する必要があります。

ファイル	デフォルトの許可
新規共有キャッシュ	グループおよびその他の読み取り許可
javasharedresources ディレクトリー	全ユーザーに対する読み取り、書き込み、および実行許可

キャッシュを破棄または拡張するには、キャッシュ・ファイルとキャッシュ・ディレクトリーの両方に対する書き込み許可が必要です。

キャッシュ・ファイルに対するファイル許可の変更

共有クラス・キャッシュへのアクセスを制限するには、**chmod** コマンドを使用します。

必要な変更	コマンド
アクセスをユーザーおよびグループに制限	chmod 770 /tmp/javasharedresources
アクセスをユーザーに制限	chmod 700 /tmp/javasharedresources
ユーザーを特定のキャッシュのみの読み取りおよび書き込みアクセスに制限	chmod 600 /tmp/javasharedresources/<*file for shared cache*>
ユーザーおよびグループを特定のキャッシュのみの読み取りおよび書き込みアクセスに制限	chmod 660 /tmp/javasharedresources/<*file for shared cache*>

アクセス許可がないキャッシュへの接続

適切なアクセス許可がないキャッシュに接続しようとすると、以下のエラー・メッセージが表示されます。

```
JVMSHRC226E Error opening shared class cache file
JVMSHRC220E Port layer error code = -302
JVMSHRC221E Platform error message: Permission denied
JVMJ9VM015W Initialization error for library j9shr25(11): JVMJ9VM009E J9VMDllMain
failed
Could not create the Java virtual machine.
```

第 9 章 トラブルシューティングおよびサポート

WebSphere Real Time for Linux のトラブルシューティングおよびサポート

- 『一般的な問題判別方法』
- 46 ページの『OutOfMemory エラーのトラブルシューティング』
- 50 ページの『診断ツールの使用』

一般的な問題判別方法

問題判別は、発生した障害の種類や、適切な処置方針を理解する上で役に立ちます。

発生した問題の種類が分かれば、以下の 1 つ以上の作業を実行することができます。

- 問題を修正する。
- 適切な回避策を見つける。
- IBM に送るバグ・レポートを生成するために必要なデータを収集する。

Linux の問題判別

このセクションでは、Linux での問題判別について説明します。

IBM SDK for Java V7 ユーザー・ガイドには、Linux での問題の診断についての次のような有用なガイダンスが記載されています。

- Linux 環境のセットアップと確認
- 一般的なデバッグ手法
- クラッシュの診断
- ハングのデバッグ
- メモリー・リークのデバッグ
- パフォーマンス上の問題のデバッグ

この情報は、IBM SDK for Java 7 - Linux の問題判別で参照できます。

以下の情報は、IBM WebSphere Real Time for Linux についての補足情報です。

Linux 環境のセットアップと確認

IBM WebSphere Real Time for Linux 上で、JVM がシステム・ダンプを生成するように正しく構成されていることを確認します。

Linux システム・ダンプ (コア・ファイル)

クラッシュが発生した場合に取得すべき最も重要な診断データは、Linux システム・ダンプ (コア・ファイル) です。このファイルが確実に生成されるようにするには、IBM SDK for Java V7 ユーザー・ガイドの説明に従って、オペレーティング・システムの設定と使用可能なディスク・スペースを確認する必要があります。

Java 仮想マシンの設定

JVM は、クラッシュの発生時にコア・ファイルを生成するように設定されている必要があります。コマンド行で `java -Xdump:what` を実行します。このオプションの出力は以下のとおりです。

```
-Xdump:system:
    events=gpf+abort+traceassert+corruptcache,
    label=/mysdk/sdk/jre/bin/core.%Y%m%d.%H%M%S.%pid.dmp,
    range=1..0,
    priority=999,
    request=serial
```

示されている値はデフォルトの設定値です。クラッシュの発生時にコア・ファイルを生成するには、少なくとも `events=gpf` を設定する必要があります。オプションは、コマンド行オプション **`-Xdump:system[:name1=value1,name2=value2 ...]`** で変更および設定することができます。

一般的なデバッグ手法

Java スレッド名はオペレーティング・システムで表示できるため、ps コマンドを使用するとデバッグの際に便利です。トレース・ツールを使用する際は、IBM WebSphere Real Time for Linux の正しいコマンドを使用する必要があります。

プロセス情報の分析

IBM WebSphere Real Time for Linux で ps コマンドを実行すると、以下のような出力が表示されます。

```
ps -eLo pid,tid,rtprio,comm,cmd
13654 13654      - java            jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13655      - main            jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13656      - Signal Reporter jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13661      - JIT Compilation jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13662      - JIT Sampler     jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13666      - Signal Dispatch jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13667      - Finalizer maste jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13668      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13669      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13670      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13671      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13672      - Metronome GC Al jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13673      - Thread-2        jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13698      - process reaper  jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13700      - stdout reader j jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13701      - stderr reader j jre/bin/java -Xgcpolicy:metronome -jar example.jar
```

e すべてのプロセスを選択します。

L スレッドを表示します。

o 表示列の事前定義された形式を指定します。指定される列は、プロセス ID、スレッド ID、スケジューリング・ポリシー、リアルタイム・スレッド優先順位、およびプロセスに関連付けられたコマンドです。この情報は、ある時点において、アプリケーションおよび仮想マシン内のどのスレッドが実行されているのかを理解する上で役立ちます。

トレース・ツール

Linux には、**strace**、**ltrace**、および **mtrace** の 3 つのトレース・ツールがあります。コマンド `man strace` を実行すると、使用可能なすべてのオプションが表示されます。

strace

strace ツールは、システム呼び出しをトレースします。このツールは、既に使用可能なプロセスに対して使用するか、新規のプロセスで開始することができます。 strace は、プログラムが実行したシステム呼び出しおよびプロセスが受け取ったシグナルを記録します。システム呼び出しごとに、名前、引数、および戻り値が使用されます。 strace では、ソースを必要とせずにプログラムをトレースできます (再コンパイルは不要です)。 strace に **-f** オプションを指定すると、fork したシステム呼び出しの結果作成された子プロセスがトレースされます。 strace を使用して、プラグインの問題を調べたり、プログラムが正しく開始されない理由の解明を試みたりすることができます。

Java アプリケーションで strace を使用するには、`strace java -Xgcpolicy:metronome <class-name>` と入力します。

strace ツールのトレース出力は、`-o` オプションを使用してファイルに送ることができます。

ltrace

ltrace ツールは、ディストリビューションに依存します。これは strace に非常によく似ています。このツールは、実行中のプロセスによって呼び出された動的ライブラリー呼び出しをインターセプトして記録します。 strace は、実行中のプロセスが受け取ったシグナルに対しても同様の動作をします。

Java アプリケーションで ltrace を使用するには、`ltrace java -Xgcpolicy:metronome <class-name>` と入力します。

mtrace

mtrace は GNU ツール・セットに含まれています。これは、malloc、realloc、および free に対する特殊なハンドラーをインストールして、これらの関数の使用をすべてトレースしてファイルに記録できるようにします。このトレース処理はプログラムの効率を下げるため、通常使用時には使用可能にしないでください。mtrace を使用するには、**IBM_MALLOCTRACE** に 1 を設定し、**MALLOC_TRACE** にトレース情報の保管先となる有効なファイルを設定します。ユーザーに、このファイルに対する書き込み権限が必要です。

Java アプリケーションで mtrace を使用するには、以下のように入力します。

```
export IBM_MALLOCTRACE=1
export MALLOC_TRACE=/tmp/file
java -Xgcpolicy:metronome <class-name>
mtrace /tmp/file
```

クラッシュの診断

クラッシュする前の実行プロセスと Java 環境に関する情報を収集する場合は、以下のガイドラインに従ってください。

プロセス情報の収集

クラッシュが発生するまでに何が起きていたのかを調査する場合は、コア・ファイルを分析するのではなく、**gdb** コマンドと **bt** コマンドを使用して、障害が発生したスレッドのスタック・トレースを表示してください。

Java 環境の理解

Javadump を使用して、各スレッドが実行していた内容、およびどの Java メソッドが実行されていたのかを判別します。関数のアドレスとライブラリーのアドレスを突き合わせて、各時点で実行されていたコードのソースを判別します。

-verbose:gc オプションを使用して Java ヒープ・メモリー域の状態を確認します。以下について検討します。

- いずれかのメモリー域でメモリーが不足していて、これがクラッシュの原因になったかどうか。
- ガーベッジ・コレクション中にクラッシュが発生したかどうか (ガーベッジ・コレクションで障害が発生した可能性があります)。
- ガーベッジ・コレクション後にクラッシュが発生したかどうか (メモリーが破損した可能性があります)。

パフォーマンス上の問題のデバッグ

パフォーマンス上の問題をデバッグする際は、IBM SDK for Java V7 ユーザー・ガイドのトピックに加え、IBM WebSphere Real Time for Linux についての以下の特定の項目についても考慮してください。

メモリー域のサイズ変更

Java ヒープ・サイズは、JVM の最も重要なチューニング・パラメーターの 1 つです。 適切なサイズを選択して、パフォーマンスを最適化してください。適切なサイズを使用することによって、ガーベッジ・コレクターがより簡単に必要な使用効率を達成できるようになります。

メモリー域のサイズ変更について詳しくは、 72 ページの『Metronome ガーベッジ・コレクターのトラブルシューティング』を参照してください。

JIT コンパイルおよびパフォーマンス

JIT を使用する場合は、リアルタイム動作に対する影響を考慮する必要があります。

Linux における既知の制限

Linux は急速に発展しているため、JVM とオペレーティング・システムの相互作用、特にスレッドの分野に関するさまざまな問題が持ち上がっています。

Linux システムに影響を与える可能性がある以下の制限に注意してください。

プロセスとしてのスレッド

Java スレッドの数がプロセスの許容最大数を超えた場合、プログラムで以下のことが発生する可能性があります。

- エラー・メッセージが表示される
- **SIGSEGV** エラーが発生する
- 停止する

詳しくは、http://www.volano.com/report/index.html にある「*The Volano Report*」を参照してください。

フローティング・スタックの制限

フローティング・スタックなしで実行している場合は、**-Xss** の設定内容とは無関係に、スレッドごとに 256 KB の最小ネイティブ・スタック・サイズが指定されます。

フローティング・スタックの Linux システムでは、**-Xss** の値が使用されます。フローティング・スタック以外の Linux システムからマイグレーションする場合は、すべての **-Xss** 値を十分な大きさにして、256 KB という最小値を利用しないようにしてください

glibc の制限

シンボルが見つからないために `libjava.so` ライブラリーをロードできなかったということを示すメッセージ (__bzero など) を受け取った場合は、GNU C ランタイム・ライブラリー (glibc) の旧バージョンがインストールされている可能性があります。 SDK for Linux でのスレッド実装には、glibc バージョン 2.3.2 以上が必要です。

フォントの制限

Red Hat システムにインストールする場合、フォント・サーバーが Java TrueType フォントを検出できるようにするには、以下を実行してください (例えば、Linux IA32 で)。

```
/usr/sbin/chkfontpath --add /opt/IBM/javawrt3[_64]/jre/lib/fonts
```

これはインストール時に実行する必要があります。また、このコマンドを実行するには、「root」としてログオンしている必要があります。フォントの問題について詳しくは、「*Linux SDK and Runtime Environment User Guide*」を参照してください。

Linux Completely Fair Scheduler による Java パフォーマンスへの影響

同期を多く使用する Java アプリケーションでは、Completely Fair Scheduler を含む Linux 配布版でパフォーマンスが低下する場合があります。Completely Fair Scheduler (CFS) は、リリース 2.6.23 以降のメインライン Linux カーネルに導入されたスケジューラーです。CFS のアルゴリズムは、旧 Linux リリースのスケジューリング・アルゴリズムとは異なります。これにより、一部のアプリケーションのパフォーマンス・プロパティーが変更される場合があります。特に CFS では sched_yield() の実装が変更されており、処理を譲る側のスレッドには確実に CPU 時間が付加されます。

この問題が発生した場合、Java アプリケーションによる CPU の高使用量、および同期化ブロックの進行の遅れが生じることがあります。進行が遅いため、アプリケーションが停止したように見えることがあります。

これを回避するには、2 つの方法があります。

- 引数 **-Xthr:minimizeUserCPU** を追加して JVM を開始する。
- 旧バージョンとの互換性がより高い sched_yield() の実装を使用するように Linux カーネルを構成する。これは、調整可能なカーネル・プロパティー sched_compat_yield を **1** に設定することで行います。例えば、以下のようにします。

  ```
  echo "1" > /proc/sys/kernel/sched_compat_yield
  ```

パフォーマンスが低下していない場合は、この回避策を使用しないでください。

この問題は、Completely Fair Scheduler を含む Linux カーネルで実行されている IBM Developer Kit and Runtime Environment for Linux 5.0 (すべてのバージョン) および 6.0 (SR 4 以前のすべてのバージョン) に影響を及ぼすことがあります。IBM Developer Kit and Runtime Environment for Linux バージョン 6.0 (SR 4 より後) では、カーネルで使用されている CFS の検出とオプション **-Xthr:minimizeUserCPU** の使用可能化が自動的に行われます。 Completely Fair Scheduler を含む Linux 配布版には、Ubuntu 8.04 および SUSE Linux Enterprise Server 11 があります。

CFS について詳しくは、Completely Fair Scheduler によるマルチプロセッシングを参照してください。

Linux Red Hat MRG カーネルでのパフォーマンスの問題

詳細ガーベッジ・コレクションを使用可能にして WebSphere Real Time を開始した場合、Red Hat MRG カーネルの構成に関する問題が原因で、アプリケーション・スレッドが予期せずに一時停止することがあります。これらの一時停止は詳細 GC の出力には報告されませんが、ネットワーク構成によっては数ミリ秒間停止することがあります。リモート側で定義された LDAP ユーザーから開始された JVM が最も大きな影響を受けます。ネーム・サービス・キャッシュ・デーモン (nscd) が開始されず、ネットワーク遅延が発生するためです。この問題を解決するには、nscd を開始します。以下の手順に従って nscd サービスの状況を確認し、問題を修正してください。

1. 以下のコマンドを入力して、nscd デーモンが実行されているか確認します。

   ```
   /sbin/service nscd status
   ```

   デーモンが実行されていない場合、以下のメッセージが表示されます。

   ```
   nscd is stopped
   ```

2. root ユーザーとして以下のコマンドを実行し、nscd サービスを開始します。

   ```
   /sbin/service nscd start
   ```

3. root ユーザーとして以下のコマンドを実行し、nscd サービスの開始情報を変更します。

   ```
   /sbin/chkconfig nscd on
   ```

これで nscd プロセスは実行状態になりました。リブート後に自動的に開始されます。

NLS の問題判別

JVM には、さまざまなロケールに対応するサポートが組み込まれています。

IBM SDK for Java V7 ユーザー・ガイドには、NLS に関する問題の診断についての次のような有用なガイダンスが記載されています。

- フォントの概要
- フォント・ユーティリティー
- NLS に関する一般的な問題と、考えられる原因

この情報は、IBM SDK for Java 7 - NLS の問題判別で参照できます。

ORB の問題判別

ORB の問題をデバッグする際にまず実行すべきことは、問題が分散アプリケーションのサーバー・サイドにあるのか、クライアント・サイドにあるのかを判断することです。ここでは、標準的な RMI-IIOP セッションを、オブジェクトへのアクセスを要求するクライアントと、アクセスを提供するサーバーの間の簡単な同期通信と考えます。

IBM SDK for Java V7 ユーザー・ガイドには、ORB に関する問題の診断についての次のような有用なガイダンスが記載されています。

- ORB 問題の特定
- スタック・トレースの解釈
- ORB トレースの解釈
- 共通問題
- IBM ORB サービス: データ収集

この情報は、IBM SDK for Java 7 - ORB の問題判別で参照できます。

以下の情報は、IBM WebSphere Real Time for Linux についての補足情報です。

IBM ORB サービス: データ収集

サービス用の Java バージョンの出力を収集するには、次のコマンドを実行します。

```
java -Xgcpolicy:metronome -version
```

予備テスト

問題が発生すると、ORB は次のような `org.omg.CORBA.*` 例外を生成する場合があります。

- 原因を示すテキスト
- マイナー・コード
- 完了状況

ORB を問題の原因と考える前に、以下の項目を確認してください。

- 同じような構成でシナリオを再現できる。
- JIT が使用不可である。
- AOT コンパイル・コードが使用されていない。

その他の処置として以下のことがあります。

- その他のプロセッサーの電源を切ります。

- 可能な場合は、同時マルチスレッド (SMT) をオフにします。
- クライアントまたはサーバーとのメモリー依存関係を除去します。物理メモリー不足は、パフォーマンスの低下、明らかなハング、またはクラッシュの原因になる可能性があります。これらの問題を除去するために、メモリーの十分なヘッドルームを確保するようにしてください。
- 物理ネットワークの問題 (ファイアウォール、通信リンク、ルーター、DNS ネーム・サーバーなど) を確認します。これらは、CORBA COMM_FAILURE 例外の主な原因です。試験的に、ご使用のワークステーション名を ping してみてください。
- アプリケーションで DB2® などのデータベースを使用している場合は、最も信頼性のあるドライバーに切り替えます。例えば、DB2 AppDriver を切り離す場合は、低速で複数のソケットを使用するが、より信頼性のある Net Driver に切り替えます。

OutOfMemory エラーのトラブルシューティング

OutOfMemoryError 例外

Metronome ガーベッジ・コレクターの一般的なトラブルシューティング情報については、72 ページの『Metronome ガーベッジ・コレクターのトラブルシューティング』を参照してください。

OutOfMemoryError の診断

Metronome ガーベッジ・コレクターでの OutOfMemoryError 例外の診断は、このガーベッジ・コレクターが定期的な性質を持っているために、標準の JVM の場合よりも複雑になることがあります。

通常、リアルタイム・アプリケーションは、標準の Java アプリケーションよりも約 20% 多いヒープ・スペースを必要とします。

キャッチされていない OutOfMemoryError が発生した場合、JVM はデフォルトでは以下の診断出力を生成します。

- スナップ・ダンプ。52 ページの『ダンプ・エージェントの使用』を参照してください。
- Heapdump。60 ページの『Heapdump の使用』を参照してください。
- Javadump。55 ページの『Javadump の使用』を参照してください。
- システム・ダンプ。63 ページの『システム・ダンプおよびダンプ・ビューアーの使用』を参照してください。

以下のように、ダンプ・ファイル名はコンソール出力に示されます。

```
JVMDUMP006I Processing dump event "systhrow", detail "java/lang/OutOfMemoryError" - please wait.
JVMDUMP007I JVM Requesting Snap dump using 'Snap.20081017.104217.13161.0001.trc'
JVMDUMP010I Snap dump written to Snap.20081017.104217.13161.0001.trc
JVMDUMP007I JVM Requesting Heap dump using 'heapdump.20081017.104217.13161.0002.phd'
JVMDUMP010I Heap dump written to heapdump.20081017.104217.13161.0002.phd
JVMDUMP007I JVM Requesting Java dump using 'javacore.20081017.104217.13161.0003.txt'
JVMDUMP010I Java dump written to javacore.20081017.104217.13161.0003.txt
JVMDUMP013I Processed dump event "systhrow", detail "java/lang/OutOfMemoryError".
```

コンソール出力に示され、Javadump にも記録される Java バックトレースは、Java アプリケーションのどこで OutOfMemoryError が発生したのかを示します。 JVM メモリー管理コンポーネントは、障害が発生した割り振りのサイズ、クラス・ブロック・アドレス、およびメモリー・スペース名を示すトレース・ポイントを発行します。以下のように、このトレース・ポイントはスナップ・ダンプで見つかります。

```
<< 行省略... >>
09:42:17.563258000 *0xf2888e00       j9mm.101  Event       J9AllocateIndexableObject() returning NULL! 80
bytes requested for object of class 0xf1632d80 from memory space 'Metronome' id=0xf288b584
```

割り振られているオブジェクトのタイプによって、トレース・ポイント ID およびデータ・フィールドがここに示されているものと異なる場合があります。この例のトレース・ポイントは、アプリケーションが、Metronome ヒープのメモリー・セグメント id=0x809c5f0 で、class 0x81312d8 タイプの 33.6 MB のオブジェクトを割り振ろうとした際に障害が発生したことを示しています。

Javadump のメモリー管理情報を確認することによって、どの メモリー域が影響を受けるかを判別できます。

```
NULL           ------------------------------------------------------------------
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STMEMTYPE     Object Memory
NULL           region     start      end        size       name
1STHEAP        0xF288B584 0xF2A1C000 0xF6A1C000 0x04000000 Default
NULL
1STMEMUSAGE    Total memory available: 67108864 (0x04000000)
1STMEMUSAGE    Total memory in use:    66676824 (0x03F96858)
1STMEMUSAGE    Total memory free:      00432040 (0x000697A8)

<< lines removed for clarity >>
```

Javadump のクラス・セクションを確認することによって、割り振られているオブジェクトのタイプを判別できます。

```
NULL           ------------------------------------------------------------------
0SECTION       CLASSES subcomponent dump routine
NULL           ================================
<< 行省略... >>
1CLTEXTCLLOD    ClassLoader loaded classes
2CLTEXTCLLOAD    Loader *System*(0xF182BB80)
<< 行省略... >>
3CLTEXTCLASS      [C(0xF1632D80)
```

Javadump の情報により、割り振りが通常のヒープ (ID=0xF288B584) 内で文字配列に対して試行されたことや、ヒープの合計割り振りサイズが、該当する 1STHEAP 行が示すとおり 67108864 10 進バイトまたは 0x04000000 16 進バイト (つまり 64 MB) であることが分かります。

この例では、障害が発生した割り振りが、合計ヒープ・サイズとの関係で大きくなっています。アプリケーションが 33 MB の複数のオブジェクトを作成すると予想される場合、次のステップは、**-Xmx** オプションを使用してヒープのサイズを大きくすることです。

障害が発生したアプリケーションは、合計ヒープ・サイズとの関係で小さくなる方が一般的です。これは、以前の割り振りによってヒープが埋まっているためです。その場合、次のステップは、Heapdump を使用して、既存のオブジェクトに割り振られているメモリーの量を調べることです。

Heapdump は、すべてのオブジェクトのリストと、それらのオブジェクト・クラス、サイズ、および参照が含まれている圧縮バイナリー・ファイルです。Heapdump の分析には、IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer ツールを使用します。このツールは IBM Support Assistant (ISA) からダウンロードできます。

MDD4J を使用すると、Heapdump をロードして、ヒープ・スペースを大量に消費している疑いのあるオブジェクトのツリー構造を特定することができます。このツールには、ヒープ上のオブジェクトに関するさまざまなビューが用意されています。例えば、MDD4J は、リークが疑われるものの詳細を示すビューを表示し、ヒープ・サイズの消費量が多い上位 5 つのオブジェクトおよびパッケージを示すこともできます。ツリー・ビューを選択すると、リークが発生しているコンテナー・オブジェクトの性質に関するさらに詳しい情報が表示されます。

IBM JVM によるメモリーの管理方法

IBM JVM は、クラス、コンパイル済みコード、Java オブジェクト、Java スタック、JNI スタック用のメモリー領域など、各種コンポーネント用のメモリーを必要とします。これらのメモリー領域のいくつかは、連続したメモリー内にある必要があります。その他のメモリー領域は、複数の小さなメモリー領域にセグメント化して、相互にリンクすることができます。

動的にロードされるクラスおよびコンパイル済みコードは、動的にロードされるクラス用のセグメント化メモリー領域に格納されます。クラスはさらに細かく分割され、書き込み可能メモリー領域 (RAM クラス) および読み取り専用メモリー領域 (ROM クラス) に格納されます。 実行時、アプリケーションを開始する際にクラス・キャッシュ内の ROM クラスおよび AOT コードは連続したメモリー領域にメモリー・マップされますが、ロードはされません。クラス・キャッシュ内のクラスおよびコンパイル済みコードがストレージにマップされるのは、クラスがアプリケーションによって参照されるためです。クラスの ROM コンポーネントは、このクラスを参照する複数のプロセス間で共有されます。クラスの RAM コンポーネントは、そのクラスが最初に JVM によって参照された際に動的にロードされるクラス用のセグメント化メモリー領域に作成されます。クラス・キャッシュにある、クラスのメソッドの AOT コンパイル済みコードは、実行可能な動的コード・メモリー領域にコピーされます。このコードはプロセス間で共有されないためです。クラス・キャッシュからロードされないクラスは、キャッシュに入れられるクラスに似ていますが、ROM クラスの情報が、動的にロードされるクラス用のセグメント化メモリー領域に作成される点が異なります。動的に生成されるコードは、キャッシュに入れられるクラスの AOT コードを保持する場合と同じ動的コード・メモリー領域に格納されます。

各 Java スレッドのスタックは、セグメント化メモリー領域をまたぐことがあります。各スレッドの JNI スタックは、連続したメモリー領域を使用します。

JVM をどのように構成すべきか判別するには、**-verbose:sizes** オプションを指定して実行します。このオプションにより、サイズ管理可能なメモリー領域に関する情報が出力されます。連続していないメモリー領域の場合は、その領域を大きくする必要があるたびに獲得されるメモリーの量を示す増分が出力されます。

-Xrealtime -verbose:sizes オプションを使用した出力の例を以下に示します。

```
-Xmca32K                    RAM class segment increment
-Xmco128K                   ROM class segment increment
-Xms64M                     initial memory size

-Xmx64M                     memory maximum
-Xmso256K                   operating system thread stack size
-Xiss2K                     java thread stack initial size
-Xssi16K                    java thread stack increment
-Xss256K                    java thread stack maximum size
```

この例では、RAM クラス・セグメントは最初は 0 ですが、必要に応じて 32 KB のブロック単位で大きくなることが示されています。 ROM クラス・セグメントは最初は 0 で、必要に応じて 128 KB のブロック単位で大きくなります。これらのサイズは、**-Xmca** オプションおよび **-Xmco** オプションを使用して制御できます。RAM クラスおよび ROM クラスのセグメントは必要に応じて大きくなるため、通常、これらのオプションを変更する必要はありません。

クラス・キャッシュを使用する場合にメモリー・マップ領域がどの程度の大きさになるかを判別するには、**-Xshareclasses** オプションを使用します。コマンド `java -Xgcpolicy:metronome -Xshareclasses:printStats` の出力例を以下に示します。

```
Current statistics for cache "sharedcc_chamlain":


base address = 0xF1BBD000
end address = 0xF2BAF000
allocation pointer = 0xF1CA95A0

cache size = 16776852
free bytes = 15499564
ROMClass bytes = 1198572
AOT bytes = 0
Data bytes = 57300
Metadata bytes = 21416
Metadata % used = 1%

# ROMClasses = 368
# AOT Methods = 0
# Classpaths = 1
# URLs = 0
# Tokens = 0
# Stale classes = 0
% Stale classes = 0%

Cache is 7% full
```

実行時に、クラスが参照されると、約 3 MB の AOT バイトおよびメタデータ・バイトが動的コード・セグメント化領域にコピーされます。クラスが参照された際に、データ・バイトが RAM クラス・セグメント化領域にコピーされます。

診断ツールの使用

IBM WebSphere Real Time for Linux JVM に関連した問題の診断に利用できる診断ツールは、数多くあります。

IBM SDK for Java 7 には、IBM WebSphere Real Time for Linux JVM に関連する問題の診断に利用できる診断ツールが数多く用意されています。このセクションでは使用可能なツールを紹介し、それらのツールの使い方に関する詳細情報へのリンクを記載しています。

SDK 診断ツールを使う際には、覚えておく必要のある重要な点があります。リアルタイム JVM を呼び出す際には、以下のオプションを使用してください。

```
java  -Xgcpolicy:metronome
```

リアルタイム JVM の診断ツールを実行する際には、このオプションを指定する必要があります。例えば、IBM WebSphere Real Time for Linux JVM に登録されているダンプ・エージェントを表示するには、次のように入力します。

```
java -Xgcpolicy:metronome -Xdump:what
```

IBM WebSphere Real Time for Linux に付属するこれらのツールを使う際の詳しい相違点については、以下のセクションで補足情報として説明し、同時に診断に役立つサンプル出力も提供します。

IBM SDK for Java 7 が生成する診断情報のまとめについては、 診断情報のまとめ (Summary of diagnostic information) を参照してください。

IBM Monitoring and Diagnostic Tools for Java の使用

IBM では、IBM JRE を使用するアプリケーションに関する問題を把握、モニター、および診断するために役立つツールおよび資料を提供しています。

以下のツールを使用できます。

- Health Center
- Garbage Collection and Memory Visualizer
- Interactive Diagnostic Data Explorer
- Memory Analyzer

Garbage Collection and Memory Visualizer

Garbage Collection and Memory Visualizer (GCMV) では、Java アプリケーションのメモリー使用、ガーベッジ・コレクションの動作、およびパフォーマンスを把握することができます。

GCMV は、以下に挙げるようなさまざまなタイプのログから得られたデータの解析および作図を行います。

- 詳細ガーベッジ・コレクション・ログ。
- -Xtgc パラメーターを使用して生成されたトレース・ガーベッジ・コレクション・ログ。
- ps、**svmon**、または **perfmon** システム・コマンドを使用して生成されるネイティブ・メモリーのログ。

このツールは、メモリー・リークなどの問題の診断や、さまざまな表示形式でのデータの分析に役立てることができ、推奨される調整を把握することができます。

GCMV は、IBM Support Assistant (ISA) のアドオンとして提供されます。アドオンのインストールおよび開始について詳しくは、http://www.ibm.com/developerworks/java/jdk/tools/gcmv/ を参照してください。

GCMV について詳しくは、IBM インフォメーション・センターを参照してください。

Health Center

Health Center は、実行中の Java 仮想マシン (JVM) の状況をモニターするための診断ツールです。

このツールには以下の 2 つのパーツがあります。

- 実行中のアプリケーションからデータを収集する Health Center エージェント。
- エージェントに接続する Eclipse ベースのクライアント。このクライアントは、データを解釈し、モニター対象アプリケーションのパフォーマンスを向上させるための推奨事項を提示します。

Health Center は、IBM Support Assistant (ISA) のアドオンとして提供されます。アドオンのインストールおよび概要については、http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/ を参照してください。

Health Center について詳しくは、IBM インフォメーション・センターを参照してください。

Interactive Diagnostic Data Explorer

Interactive Diagnostic Data Explorer (IDDE) は、ダンプ・ビューアー (**jdmpview** コマンド) に代わる GUI ベースのツールです。IDDE にはダンプ・ビューアーと同じ機能に加えて、コマンド出力を保存する機能などが用意されています。

IDDE を使用すれば、JVM で生成されたダンプ・ファイルを簡単に探索したり調べたりできます。IDDE 内で、調査ログにコマンドを入力して、ダンプ・ファイルを探索します。調査ログでサポートされている項目は次のとおりです。

- コマンド・アシスタンス
- テキストのオートコンプリート、およびクラス名などの一部のパラメーター
- コマンドおよび出力の保存機能 (後で他のユーザーにコマンドおよび出力を送信できます)
- テキストの強調表示、および問題にフラグを立てる
- 独自のコマンドを追加する機能
- IDDE 内からの Memory Analyzer の使用のサポート

IDDE は、IBM Support Assistant (ISA) のアドオンとして提供されます。アドオンのインストール方法および開始方法については、developerWorks® の「IBM Monitoring And Diagnostic Tools for Java: Interactive Diagnostic Data Explorer」の『Overview』を参照してください。

IDDE の詳細情報は、IBM インフォメーション・センターから入手できます。

Memory Analyzer

Memory Analyzer は、オペレーティング・システム・レベルのダンプおよび Portable Heap Dump (PHD) を使用して Java ヒープを分析するために役立ちます。

このツールは、きわめて多くのオブジェクトを含むダンプを分析することができ、以下の情報を提供します。

- オブジェクトの保持サイズ。
- ガーベッジ・コレクターによるオブジェクトの収集を妨げているプロセス。
- リークの疑いがあるものを自動的に抜き出すレポート。

このツールは Eclipse Memory Analyzer (MAT) プロジェクトに基づいており、IBM Diagnostic Tool Framework for Java (DTFJ) 機能を使用して、IBM JVM からのダンプの処理を可能にします。

Memory Analyzer は、IBM Support Assistant (ISA) のアドオンとして提供されます。アドオンのインストールおよび概要については、http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/ を参照してください。

Memory Analyzer について詳しくは、IBM インフォメーション・センターを参照してください。

ダンプ・エージェントの使用

ダンプ・エージェントは、JVM の初期化時にセットアップされます。ダンプ・エージェントを使用すると、JVM で発生しているガーベッジ・コレクション、スレッド開始、JVM 終了などのイベントに応じて、ダンプを開始したり外部ツールを起動したりできます。

IBM SDK for Java V7 ユーザー・ガイドには、ダンプ・エージェントについての次のような有用なガイダンスが記載されています。

- **-Xdump** オプションの使用
- ダンプ・エージェント
- ダンプ・イベント
- ダンプ・エージェントの拡張制御
- ダンプ・エージェントのトークン
- デフォルトのダンプ・エージェント
- ダンプ・エージェントの削除
- ダンプ・エージェントの環境変数
- シグナルのマッピング
- ダンプ・エージェントのデフォルトの場所

この情報は、IBM SDK for Java 7 - ダンプ・エージェントの使用で参照できます。

IBM WebSphere Real Time for Linux の補足情報は以下に記載されています。

ダンプ・イベント

ダンプ・エージェントは、JVM の実行中に発生したイベントによって起動されます。 IBM WebSphere Real Time for Linux の場合、slow イベントのデフォルト値は 5 ミリ秒です。

一部のイベントをフィルターに掛けることで、出力の妥当性を向上させることができます。詳しくは、54 ページの『filter オプション』を参照してください。

注: 現在のところ、WebSphere Real Time で unload イベントおよび expand イベントは発生しません。クラスは永久メモリー内にあるため、アンロードできません。

注: gpf イベントおよび abort イベントによって、Heapdump の起動、ヒープの準備 (request=prepwalk)、またはヒープの圧縮 (request=compact) を実行することはできません。

以下の表に、ダンプ・エージェントのトリガーとして使用可能なイベントを示します。

イベント	起動条件	フィルター操作
gpf	一般保護違反 (GPF) が発生したとき。	
user	JVM がオペレーティング・システムから SIGQUIT シグナルを受け取ったとき。	
abort	JVM がオペレーティング・システムから SIGABRT シグナルを受け取ったとき。	
vmstart	仮想マシンが開始されたとき。	
vmstop	仮想マシンが停止されたとき。	終了コードに対するフィルター (例えば、**filter=#129..#192#-42#255**)
load	クラスがロードされたとき。	クラス名に対するフィルター (例えば、**filter=java/lang/String**)
unload	クラスがアンロードされたとき。	
throw	例外がスローされたとき。	例外クラス名に対するフィルター (例えば、**filter=java/lang/OutOfMem***)
catch	例外が catch されたとき。	例外クラス名に対するフィルター (例えば、**filter=*Memory***)
uncaught	Java 例外がアプリケーションによって catch されなかったとき。	例外クラス名に対するフィルター (例えば、**filter=*MemoryError**)
systhrow	JVM が Java 例外をスローしようとしているとき。これは「throw」イベントとは異なり、JVM の内部で検出されたエラー条件に対してのみ起動されます。	例外クラス名に対するフィルター (例えば、**filter=java/lang/OutOfMem***)
thrstart	新規スレッドが開始されたとき。	
blocked	スレッドがブロック状態になったとき。	
thrstop	スレッドが停止されたとき。	
fullgc	ガーベッジ・コレクション・サイクルが開始されたとき。	

イベント	起動条件	フィルター操作
slow	内部 JVM 要求に対してスレッドの応答時間が 5ms を超えたとき。	低速と判断されるイベントについては所要時間を変更してください。例えば、**filter=#300ms** にした場合は、内部 JVM 要求に対してスレッドの応答時間が 300ms を超えたときに起動します。
allocation	所定のフィルター指定と一致するサイズの Java オブジェクトが割り振られたとき。	オブジェクト・サイズに対するフィルター。フィルターの指定は必須です。例えば、**filter=#5m** の場合は、オブジェクトのサイズが 5 Mb より大きいときに起動します。範囲もサポートしています。例えば、**filter=#256k..512k** の場合は、オブジェクトのサイズが 256 Kb から 512 Kb までの間であるときに起動します。
traceassert	JVM で内部エラーが発生したとき。	適用されません。
corruptcache	JVM が共有クラス・キャッシュの破損を検出したとき。	適用されません。

filter オプション

一部の JVM イベントは、アプリケーションの存続期間中に数千回発生します。ダンプ・エージェントでは、フィルターおよび範囲を使用して、必要以上のダンプが生成されないようにすることができます。

ワイルドカード

例外イベント・フィルターには、ワイルドカードとしてアスタリスクを使用できます。使用できるのは、フィルターの先頭または末尾のみです。以下のコマンドは、2 つ目のアスタリスクが末尾以外の場所に置かれているため、機能しません。

```
-Xdump:java:events=vmstop,filter=*InvalidArgumentException#*.myVirtualMethod
```

このフィルターを機能させるためには、以下のように変更する必要があります。

```
-Xdump:java:events=vmstop,filter=*InvalidArgumentException#MyApplication.*
```

クラス・ロード・イベントおよび例外イベント

クラス・ロード (load) イベントおよび例外 (throw、catch、uncaught、systhrow) イベントは、以下のように、Java クラス名を基準にしてフィルターに掛けることができます。

```
-Xdump:java:events=throw,filter=java/lang/OutOfMem*
-Xdump:java:events=throw,filter=*MemoryError
-Xdump:java:events=throw,filter=*Memory*
```

throw、uncaught、および systhrow の各例外イベントは、Java メソッド名を基準にしてフィルターに掛けることができます。

```
-Xdump:java:events=throw,filter=ExceptionClassName[#ThrowingClassName.
throwingMethodName[#stackFrameOffset]]
```

任意指定部分は大括弧で囲んで示しています。

catch 例外イベントは、Java メソッド名を基準にしてフィルターに掛けることができます。

```
-Xdump:java:events=catch,filter=ExceptionClassName[#CatchingClassName.
catchingMethodName]
```

任意指定部分は大括弧で囲んで示しています。

vmstop イベント

JVM のシャットダウン・イベントは、以下のように、1 つ以上の終了コードを使用してフィルターに掛けることができます。

```
-Xdump:java:events=vmstop,filter=#129..192#-42#255
```

slow イベント

slow イベントは、フィルターに掛けることで時間しきい値をデフォルトの 5 ミリ秒から変更することができます。

```
-Xdump:java:events=slow,filter=#300ms
```

デフォルトの時間よりも短い時間にフィルターを設定することはできません。

allocation イベント

allocation イベントは、フィルターに掛けて、起動条件となるオブジェクトのサイズを指定する必要があります。フィルターのサイズは、32 ビット・プラットフォームの場合にはゼロから 32 ビット・ポインターの最大値まで、64 ビット・プラットフォームの場合にはゼロから 64 ビット・ポインターの最大値までの間に設定できます。フィルターの下限値をゼロに設定すると、すべての割り振りでダンプが起動されます。

例えば、割り振りのサイズが 5 Mb より大きい場合にダンプを起動するには、以下の設定を使用します。

```
-Xdump:stack:events=allocation,filter=#5m
```

割り振りのサイズが 256Kb から 512Kb までの間である場合にダンプを起動するには、以下の設定を使用します。

```
-Xdump:stack:events=allocation,filter=#256k..512k
```

その他のイベント

フィルターをサポートしていないイベントにフィルターを適用した場合、そのフィルターは無視されます。

Javadump の使用

Javadump は、実行中のある一時点で JVM および Java アプリケーションに関して収集した診断情報を記録したファイルを生成します。例えば、オペレーティング・システム、アプリケーション環境、スレッド、スタック、ロック、およびメモリーなどに関する情報です。

IBM SDK for Java V7 ユーザー・ガイドには、Javadump についての次のような有用なガイダンスが記載されています。

- Javadump の有効化
- Javadump の起動
- Javadump の解釈
- 環境変数と Javadump

この情報は、IBM SDK for Java 7 - Javadump の使用で参照できます。

IBM WebSphere Real Time for Linux の補足情報と出力例は、以下のトピックに記載されています。

ストレージ管理 (MEMINFO)

MEMINFO セクションには、ヒープ・メモリー域、永久メモリー域、スコープ・メモリー域を含む、メモリー・マネージャーに関する情報が示されます。

Javadump の MEMINFO セクションには、メモリー・マネージャーに関する情報が示されます。メモリー・マネージャー・コンポーネントの動作について詳しくは、Metronome ガーベッジ・コレクターの使用 (Using the Metronome Garbage Collector) を参照してください。

Javadump のこの部分には、次のようなさまざまなストレージ管理の値が示されます。

- 空きメモリーの量
- 使用メモリーの量
- ヒープの現在のサイズ
- 永久メモリー域の現在のサイズ
- スコープ・メモリー域の現在のサイズ

このセクションには、ガーベッジ・コレクションの履歴データも含まれています。このデータは、タイム・スタンプが記された一連のトレース・ポイントとして表されます。最新のトレース・ポイントが先頭になります。

標準の JVM が生成した Javadump には、「GC History」セクションがあります。この情報は、リアルタイム JVM 使用時に生成された Javadump には含まれていません。 GC の動作に関する情報を取得するには、**-verbose:gc** オプションまたは JVM スナップ・トレースを使用してください。詳しくは、73 ページの『verbose:gc 情報の使用』、および IBM SDK for Java V7 ユーザー・ガイドのダンプ・エージェントのセクションを参照してください。

Javadump で、セグメントは Java ランタイムによって、大量のメモリーを使用するタスクに割り振られているメモリーのブロックです。例として次のようなタスクがあります。

- JIT キャッシュの保守
- Java クラスの保管

Java ランタイム環境では、MEMINFO セクションにリストされていない、その他のネイティブ・メモリーも割り振られます。 Java ランタイム・セグメントによって使用される合計メモリーが、必ずしも Java ランタイムの完全なメモリー占有スペースを表しているとは限りません。Java ランタイム・セグメントは、セグメント・データ構造および関連するネイティブ・メモリーのブロックで構成されています。

典型的な出力の例を以下に示します。値はすべて 16 進値として示されています。MEMINFO セクションの列見出しには、以下の意味があります。

- オブジェクト・メモリーのセクション (HEAPTYPE):

  id スペースまたは領域の ID。

  start ヒープのこの領域の開始アドレス。

  end ヒープのこの領域の終了アドレス。

  size ヒープのこの領域のサイズ。

  space/region
  id と名前のみの行の場合は、この列にメモリー・スペースの名前が示されます。それ以外の場合、この列には、メモリー・スペースの名前に続けて、そのメモリー・スペース内に含まれている特定の領域の名前が示されます。

- クラス・メモリー、JIT コード・キャッシュ、および JIT データ・キャッシュを含む、内部メモリーのセクション (SEGTYPE):

  segment
  セグメント制御データ構造のアドレス。

  start ネイティブ・メモリー・セグメントの開始アドレス。

  alloc ネイティブ・メモリー・セグメント内の現行割り振りアドレス。

  end ネイティブ・メモリー・セグメントの終了アドレス。

  type ネイティブ・メモリー・セグメントの特性が記述された内部ビット・フィールド。

  size ネイティブ・メモリー・セグメントのサイズ。

```
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STHEAPTYPE    Object Memory
NULL           id         start      end        size       space/region
1STHEAPSPACE   0x00497030     --         --         --     Generational
1STHEAPREGION  0x004A24F0 0x02850000 0x05850000 0x03000000 Generational/Tenured Region
1STHEAPREGION  0x004A2468 0x05850000 0x06050000 0x00800000 Generational/Nursery Region
1STHEAPREGION  0x004A23E0 0x06050000 0x06850000 0x00800000 Generational/Nursery Region
NULL
1STHEAPTOTAL   Total memory:           67108864 (0x04000000)
1STHEAPINUSE   Total memory in use:    33973024 (0x02066320)
1STHEAPFREE    Total memory free:      33135840 (0x01F99CE0)
NULL
1STSEGTYPE     Internal Memory
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x073DFC9C 0x0761B090 0x0761B090 0x0762B090 0x01000040 0x00010000
       (lines removed for clarity)
1STSEGMENT     0x00497238 0x004FA220 0x004FA220 0x0050A220 0x00800040 0x00010000
NULL
1STSEGTOTAL    Total memory:           873412 (0x000D53C4)
1STSEGINUSE    Total memory in use:         0 (0x00000000)
1STSEGFREE     Total memory free:      873412 (0x000D53C4)
NULL
1STSEGTYPE     Class Memory
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x0731C858 0x0745C098 0x07464098 0x07464098 0x00010040 0x00008000
        (lines removed for clarity)
1STSEGMENT     0x00498470 0x070079C8 0x07026DC0 0x070279C8 0x00020040 0x00020000
NULL
1STSEGTOTAL    Total memory:           2067100 (0x001F8A9C)
1STSEGINUSE    Total memory in use:    1839596 (0x001C11EC)
1STSEGFREE     Total memory free:       227504 (0x000378B0)
NULL
1STSEGTYPE     JIT Code Cache
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x004F9168 0x06960000 0x069E0000 0x069E0000 0x00000068 0x00080000
NULL
1STSEGTOTAL    Total memory:           524288 (0x00080000)
1STSEGINUSE    Total memory in use:    524288 (0x00080000)
```

```
1STSEGFREE     Total memory free:           0 (0x00000000)
NULL
1STSEGTYPE     JIT Data Cache
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x004F92E0 0x06A60038 0x06A6839C 0x06AE0038 0x00000048 0x00080000
NULL
1STSEGTOTAL    Total memory:           524288 (0x00080000)
1STSEGINUSE    Total memory in use:     33636 (0x00008364)
1STSEGFREE     Total memory free:      490652 (0x00077C9C)
NULL
1STGCHTYPE     GC History
3STHSTTYPE     15:18:14:901108829 GMT j9mm.134 -   Allocation failure end: newspace=7356368/8388608
oldspace=32038168/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:901104380 GMT j9mm.470 -   Allocation failure cycle end: newspace=7356416/8388608
oldspace=32038168/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:901097193 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=0 failedflipbytes=0 failedtenurecount=0
failedtenurebytes=0 flipcount=11454 flipbytes=991056 newspace=7356416/8388608 oldspace=32038168/50331648
loa=3523072/3523072 tenureage=1
3STHSTTYPE     15:18:14:901081108 GMT j9mm.140 -   Tilt ratio: 50
3STHSTTYPE     15:18:14:893358658 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=24 weakrefs=0
soft=0 phantom=0 finalizers=0
3STHSTTYPE     15:18:14:893354551 GMT j9mm.63 -   Set scavenger backout flag=false
3STHSTTYPE     15:18:14:893348733 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.002
meanexclusiveaccessms=0.002 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
3STHSTTYPE     15:18:14:893348391 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/8388608
oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
3STHSTTYPE     15:18:14:893347364 GMT j9mm.133 -   Allocation failure start: newspace=0/8388608
oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
3STHSTTYPE     15:18:14:866523613 GMT j9mm.134 -   Allocation failure end: newspace=2359064/8388608
oldspace=38199368/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:866519507 GMT j9mm.470 -   Allocation failure cycle end: newspace=2359296/8388608
oldspace=38199368/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:866513004 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=5056 failedflipbytes=445632
failedtenurecount=0 failedtenurebytes=0 flipcount=9212 flipbytes=6017148 newspace=2359296/8388608
oldspace=38199368/50331648 loa=3523072/3523072 tenureage=1
3STHSTTYPE     15:18:14:866493839 GMT j9mm.140 -   Tilt ratio: 64
3STHSTTYPE     15:18:14:859814852 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=23 weakrefs=0
soft=0 phantom=0 finalizers=0
3STHSTTYPE     15:18:14:859808692 GMT j9mm.63 -   Set scavenger backout flag=false
3STHSTTYPE     15:18:14:859801848 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.004
meanexclusiveaccessms=0.004 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
3STHSTTYPE     15:18:14:859801163 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/10747904
oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
3STHSTTYPE     15:18:14:859800479 GMT j9mm.133 -   Allocation failure start: newspace=0/10747904
oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
3STHSTTYPE     15:18:14:652219028 GMT j9mm.134 -   Allocation failure end: newspace=2868224/10747904
oldspace=38985800/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:650796714 GMT j9mm.470 -   Allocation failure cycle end: newspace=2868224/10747904
oldspace=38985800/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:650792607 GMT j9mm.475 -   GlobalGC end: workstackoverflow=0 overflowcount=0
memory=41854024/61079552
3STHSTTYPE     15:18:14:650784052 GMT j9mm.90 -   GlobalGC collect complete
3STHSTTYPE     15:18:14:650780971 GMT j9mm.57 -   Sweep end
3STHSTTYPE     15:18:14:650611567 GMT j9mm.56 -   Sweep start
3STHSTTYPE     15:18:14:650610540 GMT j9mm.55 -   Mark end
3STHSTTYPE     15:18:14:645222792 GMT j9mm.54 -   Mark start
3STHSTTYPE     15:18:14:645216632 GMT j9mm.474 -   GlobalGC start: globalcount=2
           (lines removed for clarity)
NULL
NULL           ------------------------------------------------------------------------
```

スレッドおよびスタック・トレース (THREADS)

アプリケーション・プログラマーにとって、Java ダンプの中で最も有用な部分の 1 つが THREADS セクションです。このセクションには、Java スレッド、ネイティブ・スレッド、およびスタック・トレースのリストが示されます。

Java スレッドは、オペレーティング・システムのネイティブ・スレッドによって実装されます。各スレッドは以下のような一連の行で表されます。

```
"main" J9VMThread:0x41D11D00, j9thread_t:0x003C65D8, java/lang/Thread:0x40BD6070, state:CW, prio=5
 (native thread ID:0xA98, native priority:0x5, native policy:UNKNOWN)
 Java callstack:
 at java/lang/Thread.sleep(Native Method)
 at java/lang/Thread.sleep(Thread.java:862)
 at mySleep.main(mySleep.java:31)
```

ps コマンドを使用する際に、オペレーティング・システムで Java スレッド名を表示できます。 **ps** コマンドの使用について詳しくは、 40 ページの『一般的なデバッグ手法』を参照してください。

1 行目のプロパティーは、スレッド名、JVM スレッド構造のアドレスと Java スレッド・オブジェクトのアドレス、スレッドの状態、および Java スレッドの優先順位です。2 行目のプロパティーは、ネイティブ・オペレーティング・システムのスレッド ID、ネイティブ・オペレーティング・システムのスレッド優先順位、およびネイティブ・オペレーティング・システムのスケジューリング・ポリシーです。

スレッド名は 3 とおりの方法で表示されます。

- javacore ファイル内にリストされる。すべてのスレッドが javacore ファイルにリストされるわけではありません。
- **ps** コマンドを使用して、オペレーティング・システムからスレッドをリストする場合。
- java.lang.Thread.getName() メソッドを使用している場合

以下の表に、IBM WebSphere Real Time for Linux のスレッド名に関する情報を示します。

表 4. IBM WebSphere Real Time for Linux のスレッド名

スレッドの詳細	スレッド名
2 次スレッドによるオブジェクトのファイナライズをディスパッチするために、ガーベッジ・コレクション・モジュールが使用する内部 JVM スレッド。	`Finalizer master`
ガーベッジ・コレクターが使用するアラーム・スレッド。	`GC Alarm`
ガーベッジ・コレクション用に使用されるスレーブ・スレッド。	`GC Slave`
アプリケーションにおけるメソッドの使用を抽出するために、JIT (Just-In-Time) コンパイラー・モジュールが使用する内部 JVM スレッド。	`IProfiler`
外部で生成されたか内部で生成されたかに関係なく、アプリケーションが受信したシグナルを管理するために VM が使用するスレッド。	`Signal Reporter`
Java コードをコンパイルするために使用される内部 JVM スレッド。	`JIT Compilation Thread`
実行中の JVM に JVMTI エージェントが接続できるようにするために使用される内部 JVM スレッド。	`Attach API wait loop`

Java のスレッド優先順位は、プラットフォームに依存しない方法でオペレーティング・システムの優先順位値にマップされます。 Java のスレッド優先順位の値が大きいほど、そのスレッドの優先順位が高いことを意味します。つまり、優先順位の高いスレッドほど実行頻度が上がるということです。

使用される状態値は以下のとおりです。

- R - 実行可能 (Runnable) - スレッドは必要に応じて実行可能です。
- CW - 待機状態 (Condition Wait) - スレッドは待機中です。以下のような理由が考えられます。
  - sleep() 呼び出しが実行された
  - スレッドで入出力がブロックされている
  - モニターに通知があるまで待機する wait() メソッドが呼び出された
  - スレッドが join() 呼び出しによって他のスレッドと同期中である
- S – 中断状態 (Suspended) - スレッドは他のスレッドによって中断されています。
- Z – ゾンビ (Zombie) – スレッドは強制終了されました。
- P – 保留状態 (Parked) – スレッドは新規の並行性 API (java.util.concurrent) によって保留されています。
- B – ブロック状態 (Blocked) – スレッドは現在他のものが所有しているロックの取得を待機しています。

スレッドが保留状態またはブロック状態の場合は、出力にそのスレッドについての行が含まれます。この行は、3XMTHREADBLOCK で始まり、スレッドが待機しているリソースと、そのリソースが現在所有しているスレッド (ある場合) がリストされます。詳しくは、IBM SDK for Java V7 ユーザー・ガイドのブロック状態のスレッドに関するトピックを参照してください。

Javadump を開始して診断情報を取得すると、JVM は javacore を生成する前に Java スレッドを休止します。exclusive_vm_access の準備状態が、TITLE セクションの 1TIPREPSTATE 行に表示されます。

```
1TIPREPSTATE Prep State: 0x4 (exclusive_vm_access)
```

javacore がトリガーされたときに Java コードを実行していたスレッドは、CW (待機状態) にあります。

```
3XMTHREADINFO      "main" J9VMThread:0x41481900, j9thread_t:0x002A54A4, java/lang/Thread:0x004316B8,
state:CW, prio=5
3XMTHREADINFO1           (native thread ID:0x904, native priority:0x5, native policy:UNKNOWN)
3XMTHREADINFO3          Java callstack:
4XESTACKTRACE               at java/lang/String.getChars(String.java:667)
4XESTACKTRACE               at java/lang/StringBuilder.append(StringBuilder.java:207)
```

javacore の LOCKS セクションには、これらのスレッドが内部の JVM ロックを待機していることが示されます。

```
2LKREGMON          Thread public flags mutex lock (0x002A5234): <unowned>
3LKNOTIFYQ            Waiting to be notified:
3LKWAITNOTIFY            "main" (0x41481900)
```

Heapdump の使用

Heapdump とは、Java ヒープ上にあるすべてのライブ・オブジェクトのダンプを生成する IBM Virtual Machine for Java のメカニズムを指します。ライブ・オブジェクトとは、実行中の Java アプリケーションによって使用されているオブジェクトのことです。

IBM SDK for Java V7 ユーザー・ガイドには、Heapdumps についての次のような有用なガイダンスが記載されています。

- Heapdump の取得
- Heapdump を処理するためのツール
- **-Xverbose:gc** を使用したヒープ情報の取得
- 環境変数と Heapdump
- テキスト (標準型) の Heapdump ファイル・フォーマット
- ポータブル Heapdump (PHD) ファイル・フォーマット

この情報は、IBM SDK for Java 7 - Heapdump の使用で参照できます。

IBM WebSphere Real Time for Linux の補足情報:

テキスト (標準型) の Heapdump ファイル・フォーマット

テキスト (標準型) の Heapdump では、ヒープ内のすべてのオブジェクト・インスタンスが、オブジェクトの型とサイズ、およびオブジェクト間の参照を含めてリストされます。

ヘッダー・レコード

ヘッダー・レコードは、一連のバージョン情報が格納された単一のレコードです。

```
// Version:
<SDK レベル、プラットフォーム、および JVM ビルド・レベルが含まれたバージョン文字列>
```

例:

```
// Version: J2RE 7.0 IBM J9 2.6 Linux x86-32 build 20101016_024574_lHdRSr
```

オブジェクト・レコード

オブジェクト・レコードは複数のレコード (ヒープ上のオブジェクト・インスタンスごとに 1 つのレコード) からなり、オブジェクトのアドレス、サイズ、型、およびそのオブジェクトからの参照を表します。

```
<16 進値のオブジェクト・アドレス> [<10 進値のオブジェクト・インスタンスの長さ
(バイト数)>]
OBJ <オブジェクト型> <16 進値のクラス・ブロック参照>
<16 進値のヒープ参照 <16 進値のヒープ参照> ...
```

オブジェクト・アドレスとヒープ参照はヒープ内にありますが、クラス・ブロック・アドレスはヒープ外にあります。オブジェクト・インスタンス内で見つかったすべての参照が (NULL 値の参照も含めて) リストされます。オブジェクト型は、パッケージを含むクラス名か、プリミティブ配列型またはクラス配列型であり、その標準 JVM 型シグニチャーによって示されます (63 ページの『Java VM の型シグニチャー』を参照)。オブジェクト・レコードには、追加のクラス・ブロック参照も格納できます (通常はリフレクション・クラス・インスタンスの場合)。

例:

長さが 28 バイトで java/lang/String 型のオブジェクト・インスタンス:

```
0x00436E90 [28] OBJ java/lang/String
```

java/lang/String のクラス・ブロック・アドレスと、char 配列インスタンスへの参照:

```
0x415319D8 0x00436EB0
```

長さが 44 バイトで char 配列型のオブジェクト・インスタンス:

```
0x00436EB0 [44] OBJ [C
```

char 配列のクラス・ブロック・アドレス:

```
0x41530F20
```

java/util/Hashtable Entry 内部クラスの配列型のオブジェクト:

```
0x004380C0 [108] OBJ [Ljava/util/Hashtable$Entry;
```

java/util/Hashtable Entry 内部クラス型のオブジェクト:

```
0x4158CD80 0x00000000 0x00000000 0x00000000 0x00000000 0x00421660 0x004381C0
0x00438130 0x00438160 0x00421618 0x00421690 0x00000000 0x00000000 0x00000000
0x00438178 0x004381A8 0x004381F0 0x00000000 0x004381D8 0x00000000 0x00438190
0x00000000 0x004216A8 0x00000000 0x00438130 [24] OBJ java/util/Hashtable$Entry
```

クラス・ブロック・アドレスとヒープ参照 (NULL 参照を含む):

```
0x4158CB88 0x004219B8 0x004341F0 0x00000000
```

クラス・レコード

クラス・レコードは複数のレコード (読み込まれたクラスごとに 1 つのレコード) からなり、クラス・ブロック・アドレス、サイズ、型、およびそのクラスからの参照を表します。

<16 進値のクラス・ブロック・アドレス> [*<10 進値のクラス・ブロックの長さ (バイト数)>*]
CLS *<クラス型>*
<16 進値のクラス・ブロック参照> *<16 進値のクラス・ブロック参照>* ...
<16 進値のヒープ参照> *<16 進値のヒープ参照>* ...

クラス・ブロック・アドレスとクラス・ブロック参照はヒープ外にありますが、クラス・レコードにはヒープ内への参照も格納できます (通常は静的クラス・データ・メンバーについて)。クラス・ブロック内で見つかったすべての参照が (NULL 値のものも含めて) リストされます。クラス型は、パッケージを含むクラス名か、プリミティブ配列型またはクラス配列型であり、その標準 JVM 型シグニチャーによって示されます (63 ページの『Java VM の型シグニチャー』を参照)。

例:

java/lang/Runnable クラスの長さが 32 バイトのクラス・ブロック:

```
0x41532E68 [32] CLS java/lang/Runnable
```

他のクラス・ブロックへの参照とヒープ参照 (NULL 参照を含む):

```
0x4152F018 0x41532E68 0x00000000 0x00000000 0x00499790
```

java/lang/Math クラスの長さが 168 バイトのクラス・ブロック:

```
0x00000000 0x004206A8 0x00420720 0x00420740 0x00420760 0x00420780 0x004207B0
0x00421208 0x00421270 0x00421290 0x004212B0 0x004213C8 0x00421458 0x00421478
0x00000000 0x41589DE0 0x00000000 0x4158B340 0x00000000 0x00000000 0x00000000
0x4158ACE8 0x00000000 0x4152F018 0x00000000 0x00000000 0x00000000
```

トレーラー・レコード 1

トレーラー・レコード 1 は、レコード数が格納された単一のレコードです。

```
// Breakdown - Classes: <10 進値のクラス・レコード数>,
Objects: <10 進値のオブジェクト・レコード数>,
ObjectArrays: <10 進値のオブジェクト配列レコード数>,
PrimitiveArrays: <10 進値のプリミティブ配列レコード数>
```

例:

```
// Breakdown - Classes: 321, Objects: 3718, ObjectArrays: 169,
PrimitiveArrays: 2141
```

トレーラー・レコード 2

トレーラー・レコード 2 は、合計数が格納された単一のレコードです。

```
// EOF:  Total 'Objects',Refs(null) :
<10 進値の合計オブジェクト数>,
<10 進値の合計参照数>
(,10 進値の合計 NULL 参照数>)
```

例:

```
// EOF:  Total 'Objects',Refs(null) : 6349,23240(7282)
```

Java VM の型シグニチャー

Java VM の型シグニチャーは、次の表に示す Java 型の略記です。

Java VM の型シグニチャー	Java 型
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L <完全修飾クラス> ;	<完全修飾クラス>
[<型>	<型>[ ] (<型> の配列)
(<引数の型>) <戻り値の型>	メソッド

システム・ダンプおよびダンプ・ビューアーの使用

JVM は、ユーザーによって構成可能な状況下でネイティブ・システム・ダンプ (コア・ダンプとも呼ばれます) を生成できます。システム・ダンプは一般的に大きいサイズになります。また、システム・ダンプを分析するために使用されるほとんどのツールは、プラットフォーム固有です。Linux 上でシステム・ダンプを分析するには、**gdb** ツールを使用します。

IBM SDK for Java V7 ユーザー・ガイドには、システム・ダンプとダンプ・ビューアーの使用についての、次のような有用なガイドラインが記載されています。

- システム・ダンプの概要
- システム・ダンプのデフォルト
- ダンプ・ビューアーの使用
  - **jextract** の使用
  - ダンプ・ビューアーを使用して対処すべき問題
  - **jdmpview** で使用可能なコマンド
  - セッション例
  - **jdmpview** コマンドのクイック・リファレンス

この情報は、IBM SDK for Java 7 - システム・ダンプおよびダンプ・ビューアーの使用で参照できます。

IBM WebSphere Real Time for Linux の補足情報:

jdmpview で使用可能なコマンド

jdmpview は対話型のコマンド行ツールであり、JVM システム・ダンプから得られる情報を調べて、各種の分析機能を実行できます。

info jitm
AOT および JIT コンパイル済みメソッドとそのアドレスの一覧を表示します。
- メソッドの名前とシグニチャー
- メソッドの開始アドレス
- メソッドの終了アドレス

その他のコマンド・オプションについて詳しくは、IBM SDK for Java V7 ユーザー・ガイドを参照してください。

Java アプリケーションと JVM のトレース

JVM トレースとは、IBM WebSphere Real Time for Linux で提供されるトレース機能であり、パフォーマンスに及ぼす影響を最小限に抑えることができます。大部分の場合、トレース・データは圧縮されたバイナリー・フォーマットで保持されます。これは、提供されている Java フォーマッターでフォーマット設定できます。

デフォルトで、トレースは使用可能に設定されており、トレース・ポイントの小規模なセットがメモリー・バッファーに入れられます。レベル、コンポーネント、グループ名、または個々のトレース・ポイント ID を使用して、実行時にトレース・ポイントを使用可能に設定することができます。

IBM SDK for Java V7 ユーザー・ガイドには、アプリケーションのトレースに関する以下のような詳細情報が記載されています。

- トレース可能なもの
- トレース・ポイントのタイプ
- デフォルト・トレース
- トレース・データの記録
- トレースの制御
- Java アプリケーションのトレース

- Java メソッドのトレース

IBM WebSphere Real Time for Linux をトレースする際は、トレース・オプションを指定するときにリアルタイム JVM を正しく起動する必要があります。例えば、トレース・オプションを指定するときは、次のように入力します。

```
java -Xgcpolicy:metronome -Xtrace:<options>
```

IBM SDK for Java V7 の情報については、Java アプリケーションと JVM のトレースで参照できます。

JIT および AOT の問題判別

コマンド行オプションは、JIT コンパイラーおよび AOT コンパイラーの問題診断に役立つほか、パフォーマンスの調整も行うことができます。

IBM WebSphere Real Time for Linux は一部の共通コンポーネントを IBM SDK for Java V7 と共有しますが、JIT と AOT の動作は異なります。このセクションでは、IBM WebSphere Real Time for Linux での JIT と AOT の問題のトラブルシューティングについて説明します。

JIT または AOT の問題の診断

場合により、有効なバイトコードをコンパイルした結果、無効なネイティブ・コードが生成され、Java プログラムで障害が発生することがあります。 JIT コンパイラーまたは AOT コンパイラーに欠陥があるか、また欠陥がある場合はどこに 欠陥があるかを判別することによって、Java サービス・チームに有益な情報を提供することができます。

このタスクについて

共有クラス・キャッシュへのデータの追加時にどのメソッドがコンパイルされていたのかを判別するには、admincache コマンド行で **-Xaot:verbose** オプションを使用します。例えば、次のようになります。

```
admincache -Xrealtime -Xaot:verbose -populate -aot my.jar -cp <My Class Path>
```

このセクションでは、問題がコンパイラーに関連したものであるかどうかを判別する方法について説明します。このセクションではまた、コンパイラーに関連した問題を解決する上で考えられる回避策およびデバッグ技法を提案します。

JIT コンパイラーまたは AOT コンパイラーの無効化:

問題が JIT コンパイラーまたは AOT コンパイラーで発生していると疑われる場合は、コンパイルを無効にして、引き続き問題が発生するかどうかを確認します。引き続き問題が発生する場合は、コンパイラーが原因ではないと分かります。

このタスクについて

JIT コンパイラーはデフォルトで有効になっています。 AOT コンパイラーも有効になっていますが、共有クラスが有効にされていない限り、アクティブにはなりません。効率性の点から、Java アプリケーションのすべてのメソッドがコンパイルされるわけではありません。 JVM はアプリケーションの各メソッドの呼び出し回数を保持しています。メソッドの呼び出し回数は、そのメソッドが呼び出されるか解

釈されるたびに増えていきます。回数がコンパイルのしきい値に達すると、メソッドがコンパイルされ、ネイティブに実行されます。

呼び出し回数メカニズムにより、メソッドのコンパイルはアプリケーションの存続時間全体にわたって分散されます。使用頻度の高いメソッドほど優先順位が高くなります。使用頻度の低い一部のメソッドは、一度もコンパイルされない可能性があります。結果として、Java プログラムで障害が発生した場合、それは JIT または AOT コンパイラーの問題である可能性も、JVM の他の場所における問題である可能性もあります。

障害を診断する最初のステップは、問題がどこで 発生しているかを判別することです。そのためには、まず純粋に解釈を行うモードで (つまり、JIT コンパイラーおよび AOT コンパイラーを無効にして) Java プログラムを実行する必要があります。

手順

1. コマンド行からすべての **-Xjit** オプションと **-Xaot** オプション (およびそのパラメーター) を除去します。
2. **-Xint** コマンド行オプションを使用して、JIT コンパイラーおよび AOT コンパイラーを無効にします。 パフォーマンス上の理由から、実稼働環境では **-Xint** オプションを使用しないようにしてください。

次のタスク

コンパイルを無効にして Java プログラムを実行すると、以下のいずれかの状態になります。

- 引き続き障害が発生する。この問題は、JIT コンパイラーまたは AOT コンパイラーが原因ではありません。場合によっては、プログラムでの問題の発生の仕方が変わることがありますが、その場合でも、この問題とコンパイラーは関係ありません。
- 障害が発生しなくなる。この問題は、JIT コンパイラーまたは AOT コンパイラーが原因と思われます。

  共有クラスを使用していない場合は、JIT コンパイラーに障害の原因があります。共有クラスを使用している場合は、JIT コンパイルのみを有効にしてアプリケーションを実行し、どのコンパイラーに障害の原因があるのかを判別する必要があります。 **-Xint** オプションの代わりに **-Xnoaot** オプションを使用してアプリケーションを実行します。これにより、以下のいずれかの状態になります。
  - 引き続き障害が発生する。この問題は JIT コンパイラーが原因です。 **-Xnoaot** オプションの代わりに **-Xnojit** オプションを使用して、JIT コンパイラーのみが障害の原因であるかを確認することもできます。
  - 障害が発生しなくなる。この問題は AOT コンパイラーが原因です。

JIT (Just-In-Time) コンパイラーの選択的な無効化:

Java プログラム障害の原因が JIT (Just-In-Time) コンパイラーの問題にあると思われる場合は、さらに問題を絞り込んでみてください。

このタスクについて

デフォルトでは、JIT コンパイラーはさまざまな最適化レベルでメソッドを最適化します。それぞれの呼び出し回数に基づいて、各メソッドに異なる最適化が選択されて適用されています。呼び出し頻度の高いメソッドほど、より高いレベルで最適化されます。 JIT コンパイラーのパラメーターを変更することで、メソッドの最適化レベルを制御できます。最適化プログラムが障害の原因であるのか、また、最適化プログラムが原因である場合には、どの最適化に問題があるのかを判別できます。

-Xjit オプションに付加する JIT パラメーターを、コンマ区切りのリストで指定します。構文は、**-Xjit**:<*param1*>,<*param2*>=<*value*> です。例えば、次のようにします。

```
java -Xjit:verbose,optLevel=noOpt HelloWorld
```

この場合は、 HelloWorld プログラムが実行され、JIT からの詳細出力が有効にされて、最適化は一切実行せずに JIT にネイティブ・コードを生成させます。

コンパイラーのどの部分が障害の原因となっているのかを判別するには、以下のステップに従います。

手順

1. JIT のパラメーター **count=0** を設定して、コンパイルのしきい値をゼロに変更します。このパラメーターにより、各 Java メソッドが実行前にコンパイルされるようになります。 **count=0** は、問題を診断する場合にのみ使用してください。これは、使用頻度が低いメソッドを含め、さらに多くのメソッドがコンパイルされるためです。追加のコンパイルにより、より多くの計算リソースが使用され、アプリケーションの処理速度が低下します。 **count=0** の場合、問題の領域に達すると、アプリケーションで即時に障害が発生します。場合によっては、**count=1** を使用することで、より確実に障害を再現できることがあります。
2. **disableInlining** を、JIT (Just-In-Time) コンパイラーのパラメーターに追加します。 **disableInlining** は、比較的大きく複雑なコードの生成を無効にします。問題が発生しなくなった場合は、Java サービス・チームがコンパイラーの問題を分析して修正するまでの間、回避策として **disableInlining** を使用してください。
3. **optLevel** パラメーターを追加することによって最適化レベルを下げ、障害が発生しなくなるか、「noOpt」レベルに達するまでプログラムを再実行します。JIT コンパイラーの問題の場合は、「scorching」から始めて、リストの降順に作業を進めてください。 最適化レベルを降順に示すと、以下のとおりになります。
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold
   f. noOpt

次のタスク

これらの設定のいずれかで障害が発生しなくなった場合は、それが使用可能な回避策になります。これは、Java サービス・チームがコンパイラーの問題を分析して修正する間の一時的な回避策です。JIT のパラメーター・リストから **disableInlining** を除去しても障害が再発しなければ、パフォーマンスを改善するために除去してください。 『障害が発生したメソッドの特定』の指示に従って、回避策のパフォーマンスを改善してください。

「noOpt」最適化レベルでも引き続き障害が発生する場合は、回避策として JIT (Just-In-Time) コンパイラーを無効にする必要があります。

障害が発生したメソッドの特定:

障害を引き起こすメソッドを JIT コンパイラーまたは AOT コンパイラーがコンパイルしなければならない最も低い最適化レベルを判別したら、コンパイル時に Java プログラムのどの部分が障害を引き起こしているのかを突き止めることができます。その後は、コンパイラーに命令して特定のメソッド、クラス、またはパッケージに回避策を限定し、コンパイラーがプログラムの残りの部分を通常どおりにコンパイルできるようにします。 JIT コンパイラーの障害で、**-Xjit:optLevel=noOpt** を使用しても障害が発生する場合は、その障害を引き起こしているメソッドを決してコンパイルしないようにコンパイラーに命令することもできます。

始める前に

以下の例のようなエラー出力がある場合は、これを使用して障害が発生しているメソッドを特定できます。

```
Unhandled exception
Type=Segmentation error vmState=0x00000000
Target=2_30_20050520_01866_BHdSMr (Linux 2.4.21-27.0.2.EL)
CPU=s390x (2 logical CPUs) (0x7b6a8000 RAM)
J9Generic_Signal_Number=00000004 Signal_Number=0000000b Error_Value=4148bf20 Signal_Code=00000001
Handler1=00000100002ADB14 Handler2=00000100002F480C InaccessibleAddress=0000000000000000
gpr0=0000000000000006 gpr1=0000000000000006 gpr2=0000000000000000 gpr3=0000000000000006
gpr4=0000000000000001 gpr5=0000000080056808 gpr6=0000010002BCCA20 gpr7=0000000000000000
......
Compiled_method=java/security/AccessController.toArrayOfProtectionDomains([Ljava/lang/Object;
Ljava/security/AccessControlContext;)[Ljava/security/ProtectionDomain;
```

重要な行は以下のとおりです。

vmState=0x00000000
障害が発生したコードが、JVM ランタイム・コードではないことを示します。

Module= or Module_base_address=
このコードは JIT によって、DLL またはライブラリーの外側でコンパイルされたため、この出力にはありません (ブランクやゼロである場合もあります)。

Compiled_method=
コンパイル済みコードの生成の対象となる Java メソッドを示します。

このタスクについて

障害が発生しているメソッドが出力に示されていない場合は、以下のステップに従って、障害が発生しているメソッドを特定してください。

手順

1. JIT のパラメーター **verbose** および **vlog=**<*filename*> を **-Xjit** または **-Xaot** オプションに追加して、Java プログラムを実行します。 これらのパラメーターが使用されている場合、コンパイラーは、コンパイルしたメソッドを <*filename*>.<*date*>.<*time*>.<*pid*> という名前のログ・ファイルにリストします。このファイルは、*しきい値ファイル* とも呼ばれます。典型的なしきい値ファイルには、コンパイルされたメソッドに対応する以下のような行が含まれています。

   ```
   + (hot) java/lang/Math.max(II)I @ 0x10C11DA4-0x10C11DDD
   ```

   先頭に正符号がない行は、以降のステップでコンパイラーによって無視されるため、ファイルから削除して構いません。 AOT コンパイラーによってコンパイルされたメソッドの先頭には、`+ (AOT cold)` が付きます。AOT コードが共有クラス・キャッシュからロードされたメソッドの先頭には、`+ (AOT load)` が付きます。

2. JIT パラメーターまたは AOT パラメーター **limitFile=**(<*filename*>,<*m*>,<*n*>) を使用してプログラムを再度実行します。 <*filename*> はしきい値ファイルのパス、<*m*> および <*n*> はしきい値ファイル内の最初と最後のコンパイル対象メソッドを示す行番号です。 コンパイラーは、しきい値ファイル内の <*m*> から <*n*> までの行にリストされたメソッドのみをコンパイルします。しきい値ファイルにリストされていないメソッド、およびこの範囲外の行にリストされているメソッドはコンパイルされません。また、これらのメソッドの共有データ・キャッシュにある AOT コードはロードされません。 プログラムで障害が発生しなくなった場合、恐らくは最後の反復で削除した 1 つ以上のメソッドが障害の原因であると考えられます。

3. オプション: AOT の問題を診断している場合は、同じオプションを使用してプログラムをもう一度実行し、コンパイルされたメソッドを共有データ・キャッシュからロードできるようにします。 **–Xaot:scount=0** オプションを追加して、最初にメソッドが呼び出されたときに、共有データ・キャッシュに格納された AOT コンパイル済みメソッドが使用されるようにすることもできます。 AOT のコンパイルに関する一部の障害は、共有データ・キャッシュから AOT コンパイル済みコードをロードする際にのみ発生します。これらの問題の診断を行いやすくするには、**–Xaot:scount=0** オプションを使用して、最初にメソッドが呼び出されたときに、共有データ・キャッシュに格納された AOT コンパイル済みメソッドが使用されるようにします。それにより、より簡単に問題を再現できることがあります。 **scount** オプションを 0 に設定すると、AOT コードが強制的にロードされるため、そのメソッドの実行を待機しているアプリケーション・スレッドが休止されることに注意してください。そのため、この設定は診断目的の場合にのみ使用するようにしてください。 **–Xaot:scount=0** オプションを使用すると、休止時間がかなり長くなる可能性があります。

4. <*m*> および <*n*> に異なる値を使用して必要な回数だけこのプロセスを繰り返し、コンパイル時に障害を引き起こす最小限のメソッド・セットを見つけてください。 毎回、選択する行の数を半数にしていくことで、障害が発生しているメソッドの二分探索を実行できます。 多くの場合、ファイルを 1 行になるまで減らしていくことができます。

次のタスク

障害が発生しているメソッドが見つかった場合は、そのメソッドに対してのみ、JIT コンパイラーまたは AOT コンパイラーを無効にすることができます。例えば、**optLevel=hot** を使用して JIT コンパイルを実行する際にメソッド java/lang/Math.max(II)I がプログラムの障害を引き起こしている場合は、以下を使用してプログラムを実行します。

```
-Xjit:{java/lang/Math.max(II)I}(optLevel=warm,count=0)
```

これにより、障害が発生しているメソッドだけ「warm」最適化レベルでコンパイルされ、他のすべてのメソッドは通常どおりにコンパイルされます。

「noOpt」で JIT コンパイルした際に障害が発生するメソッドについては、**exclude**={<*method*>} パラメーターを使用して、コンパイルの対象から完全に除外できます。

```
-Xjit:exclude={java/lang/Math.max(II)I}
```

共有データ・キャッシュから AOT コードをコンパイルまたはロードした際にメソッドがプログラムの障害を引き起こす場合は、**exclude**={<*method*>} パラメーターを使用して、そのメソッドを AOT コンパイルおよび AOT ロードから除外します。

```
-Xaot:exclude={java/lang/Math.max(II)I}
```

AOT メソッドは、「cold」最適化レベルでのみコンパイルされます。これらのメソッドに関しては、AOT コンパイルまたは AOT ロードを行わないことが最善の方法です。

JIT コンパイルの障害の特定:

JIT コンパイラーの障害が発生した場合は、エラー出力を分析して、JIT コンパイラーがメソッドのコンパイルを試行した際に障害が発生したのかどうかを判別してください。

JVM が異常終了し、その障害が JIT ライブラリー (`libj9jit26.so`) で起こったことが分かる場合は、JIT (Just-In-Time) コンパイラーがメソッドをコンパイルしようとしている間に障害が発生した可能性があります。

以下の例のようなエラー出力がある場合は、これを使用して障害が発生しているメソッドを特定できます。

```
Unhandled exception
Type=Segmentation error vmState=0x00050000
Target=2_30_20051215_04381_BHdSMr (Linux 2.4.21-32.0.1.EL)
CPU=ppc64 (4 logical CPUs) (0xebf4e000 RAM)
J9Generic_Signal_Number=00000004 Signal_Number=0000000b Error_Value=00000000 Signal_Code=00000001
Handler1=0000007FE05645B8 Handler2=0000007FE0615C20
R0=E8D4001870C00001 R1=0000007FF49181E0 R2=0000007FE2FBCEE0 R3=0000007FF4E60D70
R4=E8D4001870C00000 R5=0000007FE2E02D30 R6=0000007FF4C0F188 R7=0000007FE2F8C290
......
Module=/home/test/sdk/jre/bin/libj9jit26.so
Module_base_address=0000007FE29A6000
......
Method_being_compiled=com/sun/tools/javac/comp/Attr.visitMethodDef(Lcom/sun/tools/javac/tree/
JCTree$JCMethodDecl;)
```

重要な行は以下のとおりです。

vmState=0x00050000
JIT コンパイラーがコードをコンパイルしていることを示します。 vmState のコード番号のリストについては、「IBM SDK for Java V7 ユーザー・ガイド (IBM SDK for Java 7 User guide)」(http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html) に記載された Javadump タグの表を参照してください。

Module=/home/test/sdk/jre/bin/libj9jit26.so
JIT (Just-In-Time) コンパイラーのモジュール `libj9jit26.so` でエラーが発生したことを示します。

Method_being_compiled=
コンパイル対象の Java メソッドを示します。

障害が発生しているメソッドが出力に示されていない場合は、以下の追加設定とともに **verbose** オプションを使用してください。

```
-Xjit:verbose={compileStart|compileEnd}
```

これらの **verbose** 設定により、JIT がメソッドのコンパイルを開始した時刻および終了した時刻が報告されるようになります。 JIT コンパイラーが特定のメソッドで障害を起こしている場合 (つまり、コンパイルは開始されるが、終了する前に異常終了する場合) は、**exclude** パラメーターを使用して、そのメソッドをコンパイルから除外してください (68 ページの『障害が発生したメソッドの特定』を参照)。メソッドを除外することで異常終了を防げる場合には、それがサービス・チームが問題を修正するまでの回避策になります。

短期実行アプリケーションのパフォーマンス

IBM JIT コンパイラーは、通常はサーバーで使用される長期実行アプリケーションに合わせて調整されています。 **-Xquickstart** コマンド行オプションを使用すると、短期実行アプリケーション、特に処理が少数のメソッドに集中していないアプリケーションのパフォーマンスを向上させることができます。

-Xquickstart を使用すると、JIT コンパイラーがデフォルトでより低い最適化レベルを使用するようになり、コンパイルされるメソッドの数が少なくなります。より少ない回数のコンパイルをより短い時間で実行することにより、アプリケーションの開始時間を短縮できます。 AOT コンパイラーがアクティブになっている (共有クラスと AOT コンパイルの両方が有効になっている) 場合に **-Xquickstart** を使用すると、コンパイルの対象として選択されているすべてのメソッドが AOT コンパイルされ、以後実行した場合の起動時間が短縮されます。大量の処理リソースを使用するメソッドが含まれた長期実行アプリケーションで **-Xquickstart** を使用した場合、パフォーマンスが低下する可能性があります。**-Xquickstart** の実装は、今後のリリースで変更されることがあります。

開始時間は、JIT のしきい値を (試行錯誤して) 調整することによって短縮することもできます。詳しくは、 66 ページの『JIT (Just-In-Time) コンパイラーの選択的な無効化』を参照してください。

アイドル期間中の JVM の動作

-XsamplingExpirationTime オプションを使用して JIT のサンプリング・スレッドをオフにすることで、アイドル状態の JVM が消費する CPU サイクルを削減することができます。

JIT のサンプリング・スレッドは、実行中の Java アプリケーションに関するプロファイルを作成して、頻繁に使用されているメソッドをディスカバーします。サンプリング・スレッドによるメモリーおよびプロセッサーの使用量はごくわずかです。また、プロファイルの作成頻度は、JVM がアイドル状態の場合には自動的に引き下げられます。

アイドル状態の JVM に CPU サイクルを一切消費させないようにする必要がある場合もあります。その場合には、**-XsamplingExpirationTime**<*time*> オプションを指定します。 <*time*> には、サンプリング・スレッドの実行時間を秒数で設定します。このオプションは慎重に使用してください。このオプションをオフにした後でサンプリング・スレッドを再度アクティブにすることはできません。サンプリング・スレッドの実行には、肝心の最適化を識別できるだけの十分な時間を与えてください。

Diagnostics Collector

Diagnostics Collector は、問題のあるイベントの Java 診断ファイルを収集します。

IBM サービスから要求されるファイルを収集することにより、報告された問題の解決にかかる時間を短縮できます。IBM SDK for Java V7 ユーザー・ガイドには、Diagnostics Collector の使用に関する詳細情報が記載されています。

この情報は、「IBM SDK for Java 7 - Diagnostics Collector」に記載されています。

ガーベッジ・コレクター診断データ

このセクションでは、ガーベッジ・コレクションの問題を診断する方法を説明します。

IBM SDK for Java V7 ユーザー・ガイドには、ガーベッジ・コレクターの問題の診断についての次のような有用なガイダンスが記載されています。

- 詳細ガーベッジ・コレクションのロギング
- **-Xtgc** を使用したガーベッジ・コレクションのトレース

この情報は、『IBM SDK for Java 7 - ガーベッジ・コレクターの診断』で参照できます。

IBM WebSphere Real Time for Linux の Metronome ガーベッジ・コレクターに関する補足情報は、以下のセクションに記載されています。

Metronome ガーベッジ・コレクターのトラブルシューティング

コマンド行オプションを使用することで、Metronome ガーベッジ・コレクションの頻度、メモリー不足例外、および明示的なシステム呼び出しでの Metronome の動作を制御できます。

verbose:gc 情報の使用:

-verbose:gc オプションを **-Xgc:verboseGCCycleTime=N** オプションとともに使用して、Metronome ガーベッジ・コレクター・アクティビティーに関する情報をコンソールに書き込むことができます。標準の JVM からの **-verbose:gc** 出力ですべての XML プロパティーが作成されるわけではなく、Metronome ガーベッジ・コレクターの出力に適用されるわけでもありません。

-verbose:gc オプションを使用して、ヒープ内の最小、最大、および平均フリー・スペースを表示します。これにより、ヒープのアクティビティーや使用状況のレベルを確認し、必要に応じて値を調整することができます。この **-verbose:gc** オプションを使用すると、Metronome の統計がコンソールに書き込まれます。

-Xgc:verboseGCCycleTime=N オプションでは、情報の検索頻度を制御します。このオプションにより、ミリ秒単位の要約のダンプ時間が決まります。N のデフォルト値は 1000 ミリ秒です。要約がダンプされるのは、サイクル・タイムに指定された時点ちょうどではなく、この時間の基準を満たす最後のガーベッジ・コレクション・イベント時です。これらの統計の収集および表示により、Metronome ガーベッジ・コレクターによる休止時間が増加する可能性があります。また、N の値が小さくなるほど、休止時間が長くなる可能性があります。

クォンタは、アプリケーションの中断時間または休止時間を生じさせる、Metronome ガーベッジ・コレクター・アクティビティーの単一期間です。

verbose:gc 出力の例

以下のように入力します。

```
java -Xgcpolicy:metronome -verbose:gc -Xgc:verboseGCCycleTime=N myApplication
```

ガーベッジ・コレクションが起動されると、trigger start イベントが発生し、その後に任意の数の heartbeat イベントが続き、起動の完了時に trigger end イベントが発生します。この例では、起動されたガーベッジ・コレクション・サイクルが verbose:gc 出力として示されています。

```
<trigger-start id="25" timestamp="2011-07-12T09:32:04.503" />

<cycle-start id="26" type="global" contextid="26" timestamp="2011-07-12T09:32:04.503" intervalms="984.285" />

<gc-op id="27" type="heartbeat" contextid="26" timestamp="2011-07-12T09:32:05.209">
  <quanta quantumCount="321" quantumType="mark" minTimeMs="0.367" meanTimeMs="0.524" maxTimeMs="1.878"
    maxTimestampMs="598704.070" />
  <exclusiveaccess-info minTimeMs="0.006" meanTimeMs="0.062" maxTimeMs="0.147" />
  <free-mem type="heap" minBytes="99143592" meanBytes="114374153" maxBytes="134182032" />
  <thread-priority maxPriority="11" minPriority="11" />
</gc-op>

<gc-op id="28" type="heartbeat" contextid="26" timestamp="2011-07-12T09:32:05.458">
  <quanta quantumCount="115" quantumType="sweep" minTimeMs="0.430" meanTimeMs="0.471" maxTimeMs="0.511"
    maxTimestampMs="599475.654" />
  <exclusiveaccess-info minTimeMs="0.007" meanTimeMs="0.067" maxTimeMs="0.173" />
  <classunload-info classloadersunloaded=9 classesunloaded=156  />
  <references type="weak" cleared="660" />
  <free-mem type="heap" minBytes="24281568" meanBytes="55456028" maxBytes="87231320" />
  <thread-priority maxPriority="11" minPriority="11" />
</gc-op>

<gc-op id="29" type="syncgc" timems="136.945" contextid="26" timestamp="2011-07-12T09:32:06.046">
  <syncgc-info reason="out of memory" exclusiveaccessTimeMs="0.006" threadPriority="11" />
```

```
  <free-mem-delta type="heap" bytesBefore="21290752" bytesAfter="171963656" />
</gc-op>

<cycle-end id="30" type="global" contextid="26" timestamp="2011-07-12T09:32:06.046" />

<trigger-end id="31" timestamp="2011-07-12T09:32:06.046" />
```

発生する可能性のあるイベントのタイプは以下のとおりです。

<trigger-start ...>
使用メモリー量がトリガーしきい値を上回ったことによる、ガーベッジ・コレクション・サイクルの開始。デフォルトのしきい値はヒープの 50% です。intervalms 属性は、直前の trigger end イベント (ID は -1) と現行の trigger start イベントとの間隔を表します。

<trigger-end ...>
ガーベッジ・コレクション・サイクルにより、使用済みメモリー量がトリガーしきい値未満まで下がりました。ガーベッジ・コレクション・サイクルが終了しても、使用済みメモリーがトリガーしきい値を下回らなかった場合は、新しいガーベッジ・コレクション・サイクルが、同じコンテキスト ID で開始されます。trigger start イベントごとに、対応する同じコンテキスト ID の trigger end イベントがあります。intervalms 属性は、直前の trigger start イベントと現行の trigger end イベントとの間隔を表します。この間に、使用済みメモリーがトリガーしきい値を下回るまで、1 つ以上のガーベッジ・コレクション・サイクルが完了します。

<gc-op id="28" type="heartbeat"...>
対象となる期間中のすべてのガーベッジ・コレクション・クォンタに関する (メモリーや時間などの) 情報を収集する定期的なイベント。 heartbeat イベントは、対応する trigger start イベントと trigger end イベントの組の間 (つまり、アクティブ・ガーベッジ・コレクション・サイクルが進行中である間) にのみ発生する可能性があります。intervalms 属性は、直前の heartbeat イベント (ID は -1) と現行の heartbeat イベントとの間隔を示します。

<gc-op id="29" type="syncgc"...>
同期 (非決定論的な) ガーベッジ・コレクション・イベント。 75 ページの『同期ガーベッジ・コレクション』を参照してください。

この例の各 XML タグの意味は以下のとおりです。

<quanta ...>
ハートビート間隔内のクォンタ休止時間の長さの要約。休止時間の長さがミリ秒で示されます。

<free-mem type="heap" ...>
ハートビート間隔でのフリー・ヒープ・スペース量 (各ガーベッジ・コレクション・クォンタの終了時にサンプリングされます) の要約。

<classunload-info classloadersunloaded=9 classesunloaded=156 />
ハートビート間隔においてアンロードされたクラスおよびクラス・ローダーの数。

<references type="weak" cleared="660 />
ハートビート間隔内にクリアされた Java 参照オブジェクトの数とタイプ。

注:

- 2 つのハートビートの間にガーベッジ・コレクション・クォンタが 1 回しか生じなかった場合、空きメモリーのサンプリングは、この 1 回のクォンタの終了時にのみ行われます。このため、ハートビートの要約で示される最小量、最大量、および平均量はすべて同じになります。
- ガーベッジ・コレクション・アクティビティーが必要になるほどヒープが埋まっていない場合、2 つのハートビート・イベントの間隔は、指定されたサイクル・タイムよりも大幅に長くなる可能性があります。例えば、ご使用のプログラムでガーベッジ・コレクション・アクティビティーを数秒ごとに一度だけ行う必要がある場合、ハートビートは数秒ごとに一度だけになると考えられます。
- ガーベッジ・コレクション・アクティビティーに必要なヒープが十分でない場合、2 つのハートビート・イベントの間隔が指定されたサイクル・タイムより大幅に長くなる可能性があります。例えば、ご使用のプログラムでガーベッジ・コレクション・アクティビティーを数秒ごとに一度だけ行う必要がある場合、ハートビートは数秒ごとに一度だけになると考えられます。

  同期ガーベッジ・コレクションや優先順位変更などのイベントが発生した場合、そのイベントおよび保留中の (ハートビートなどの) すべてのイベントの詳細は、直ちに出力として生成されます。
- 一定期間の最大ガーベッジ・コレクション・クォンタが大きすぎる場合は、**-Xgc:targetUtilization** オプションを使用してターゲットの使用率を減らした方がよい場合があります。このアクションは、ガーベッジ・コレクターの処理時間を増やします。あるいは、**-Xmx** オプションでヒープ・サイズを増やす方法もあります。同様に、ご使用のアプリケーションで、現在報告されている時間よりも長い遅延時間が許容されている場合は、ターゲットの使用率を増やすか、ヒープ・サイズを減らすことができます。
- **-Xverbosegclog:<file>** オプションで、出力をコンソールではなく、ログ・ファイルにリダイレクトできます。例えば、**-Xverbosegclog:out** を使用した場合は、**-verbose:gc** の出力が *out* ファイルに書き込まれます。
- `thread-priority` にリストされている優先順位は、Java スレッド優先の順位ではなく、基盤となっているオペレーティング・システムのスレッド優先順位です。

同期ガーベッジ・コレクション

同期 (非決定論的な) ガーベッジ・コレクションが発生した場合、項目は **-verbose:gc** ログにも書き込まれます。このイベントの原因として、以下の 3 つが考えられます。

- コードでの明示的な `System.gc()` 呼び出し。
- JVM でメモリー不足が発生し、OutOfMemoryError 状態を回避するために、同期ガーベッジ・コレクションが実行された。
- JVM が継続的なガーベッジ・コレクション中にシャットダウンした。JVM はガーベッジ・コレクションを取り消すことができないため、ガーベッジ・コレクションを同期的に完了し、終了します。

`System.gc()` 項目は以下の例のようになります。

```
<gc-op id="9" type="syncgc" timems="12.92" contextid="8" timestamp=
"2011-07-12T09:41:40.808">
  <syncgc-info reason="system GC" totalBytesRequested="260" exclusiveaccessTimeMs="0.009"
```

```
  threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="22085440" bytesAfter="136023450" />
  <classunload-info classloadersunloaded="54" classesunloaded="234" />
  <references type="soft" cleared="21" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="523" />
  <finalization enqueued="124" />
</gc-op>
```

JVM のシャットダウンの結果として行われた同期ガーベッジ・コレクションの項目は、以下の例のようになります。

```
<gc-op id="24" type="syncgc" timems="6.439" contextid="19" timestamp="2011-07-12T09:43:14.524">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="56182430" bytesAfter="151356238" />
  <classunload-info classloadersunloaded="14" classesunloaded="276" />
  <references type="soft" cleared="154" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="53" />
  <finalization enqueued="34" />
</gc-op>
```

この例の XML タグおよび属性の意味は以下のとおりです。

<gc-op id="9" type="syncgc" timems="6.439" ...
この行は、イベント・タイプが同期ガーベッジ・コレクションであることを示しています。 timems 属性は、同期ガーベッジ・コレクションの継続時間です (ミリ秒単位)。

<syncgc-info reason="..."/>
同期ガーベッジ・コレクションの原因を示しています。

<free-mem-delta.../>
同期ガーベッジ・コレクションの実行前と実行後のフリー Java ヒープ・メモリー (バイト単位)。

<finalization .../>
ファイナライズを待っているオブジェクトの数。

<classunload-info .../>
ハートビート間隔においてアンロードされたクラスおよびクラス・ローダーの数。

<references type="weak" cleared="53" .../>
ハートビート間隔内にクリアされた Java 参照オブジェクトの数とタイプ。

メモリー不足状態または VM のシャットダウンが原因の同期ガーベッジ・コレクションは、ガーベッジ・コレクターがアクティブである場合にのみ発生する可能性があります。即時というわけではありませんが、trigger start イベントが先行する必要があります。場合によっては、複数の heartbeat イベントが、trigger start イベントと synchgc イベントの間に発生する可能性があります。 System.gc() が原因の同期ガーベッジ・コレクションはどの時点でも発生する可能性があります。

すべての GC クォンタのトラッキング

個々の GC クォンタのトラッキングは、GlobalGCStart および GlobalGCEnd のトレース・ポイントを有効にすることで可能になります。これらのトレース・ポイントは、同期ガーベッジ・コレクションを含む、すべての Metronome ガーベッジ・コレクター・アクティビティーの開始時と終了時に生成されます。これらのトレース・ポイントの出力は以下のようになります。

```
03:44:35.281 0x833cd00 j9mm.52 - GlobalGC start: globalcount=3

03:44:35.284 0x833cd00 j9mm.91 - GlobalGC end: workstackoverflow=0 overflowcount=0
```

メモリー不足に関する項目

ヒープでフリー・スペース不足が発生すると、OutOfMemoryError 例外がスローされる前に、**-verbose:gc** ログに項目が書き込まれます。この出力は以下の例のようになります。

```
<out-of-memory id="71" timestamp="2011-07-12T10:21:50.135" memorySpaceName="Metronome"
  memorySpaceAddress="0806DFDC"/>
```

デフォルトでは、OutOfMemoryError 例外の結果として、Javadump が生成されます。このダンプには、プログラムによって使用されたメモリーに関する情報が含まれます。

```
NULL
1STSEGTOTAL    Total memory:           4066080 (0x003E0B20)
1STSEGINUSE    Total memory in use:    3919440 (0x003BCE50)
1STSEGFREE     Total memory free:       146640 (0x00023CD0)
```

メモリー不足状態での Metronome ガーベッジ・コレクターの動作:

JVM でメモリー不足が発生した場合、デフォルトでは、Metronome ガーベッジ・コレクターは無制限の非決定論的なガーベッジ・コレクションを起動します。非決定論的な動作を回避するには、**-Xgc:noSynchronousGCOnOOM** オプションを使用して、JVM でメモリー不足が発生したときに OutOfMemoryError をスローするようにします。

デフォルトの無制限コレクションは、考えられるすべてのガーベッジが 1 回の操作で収集されるまで実行されます。必要な休止時間は、通常、Metronome の標準的な増分クォンタよりも長いミリ秒になります。

関連情報:

-Xverbose:gc を使用した同期ガーベッジ・コレクションの分析

明示的な System.gc() 呼び出しでの Metronome ガーベッジ・コレクターの動作:

ガーベッジ・コレクション・サイクルが進行中の場合は、System.gc() が呼び出されると、Metronome ガーベッジ・コレクターがサイクルを完了させるまでこのメソッドは待機します。進行中のガーベッジ・コレクション・サイクルがない場合は、System.gc() が呼び出されたときに完全なサイクルが実行され、このメソッドはサイクルが完了するまで待機します。制御された方法でヒープをクリーンアップするには、System.gc() を使用します。これは、ガーベッジ・コレクションが完全に実行されてからコントロールが戻されるため、非決定論的な操作になります。

一部のアプリケーションは、これらの非決定論的な遅延の発生が受け入れられない System.gc() を呼び出すベンダー・ソフトウェアを呼び出します。すべての System.gc() 呼び出しを無効にするには、**-Xdisableexplicitgc** オプションを使用します。

以下のように、System.gc() 呼び出しに対する詳細なガーベッジ・コレクション出力には、『system garbage collect』 という理由が含まれており、duration の時間が長くなる可能性があります。

```
<gc-op id="9" type="syncgc" timems="6.439" contextid="8" timestamp="2011-07-12T09:41:40.808">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11"/>
  <free-mem-delta type="heap" bytesBefore="126082300" bytesAfter="156085440"/>
  <classunload-info classloadersunloaded="14" classesunloaded="276"/>
  <references type="soft" cleared="154" dynamicThreshold="29" maxThreshold="32"/>
  <references type="weak" cleared="53"/>
  <finalization enqueued="34"/>
</gc-op>
```

共有クラス診断データ

発生した問題をどのように診断するのかを理解しておくと、共有クラス・モードを使用するときに役立ちます。

IBM SDK for Java V7 ユーザー・ガイドには、共有クラスに関する問題の診断についての次のような有用なガイダンスが記載されています。

- 共有クラスの配備
- 実行時バイトコード変更の取り扱い
- 動的更新について
- Java ヘルパー API の使用
- 共有クラス診断出力について
- 共有クラスに関する問題のデバッグ

この情報は、『IBM SDK for Java 7 - 共有クラス診断』で参照できます。

JVMTI の使用

JVMTI は、JVM とネイティブ・エージェント間の通信を可能にする、双方向インターフェースです。これにより、JVMDI および JVMPI のインターフェースが置き換えられます。

JVMTI により、JVM 用のデバッグ、プロファイル作成、およびモニター・ツールの作成をサード・パーティーが行うことができます。 必要とされる種類の情報についてエージェントが JVM に通知するメカニズムが、インターフェースに搭載されています。このインターフェースでは、関連した通知を受け取る手段も提供されています。複数のエージェントを、いつでも JVM に接続することができます。

IBM SDK for Java V7 ユーザー・ガイドには、JVMTI に対する IBM 拡張に関する API リファレンスのセクションを含む、JVMTI の使用に関する詳細情報が記載されています。

この情報は、IBM SDK for Java 7 - JVMTI の使用で参照できます。

Diagnostic Tool Framework for Java の使用

Diagnostic Tool Framework for Java (DTFJ) は、IBM が提供する Java アプリケーション・プログラミング・インターフェース (API) であり、Java 診断ツールの作成をサポートするために使用されます。DTFJ は、システム・ダンプまたは Javadump のデータを扱います。

IBM SDK for Java V7 ユーザー・ガイドには、DTFJ に関する詳細情報が記載されています。Diagnostic Tool Framework for Java の使用を参照してください。

第 10 章 参照

この一連のトピックでは、WebSphere Real Time for Linux で使用可能なオプションおよびクラス・ライブラリーのリストを示します。

コマンド行オプション

Java の開始時にコマンド行でオプションを指定できます。デフォルト・オプションは、最も一般的な使用法に応じて選択されています。

Java オプションとシステム・プロパティーの指定

Java プロパティーおよびシステム・プロパティーは、3 とおりの方法で指定できます。

このタスクについて

Java オプションおよびシステム・プロパティーは、以下の方法で指定できます。それらは、優先順に以下のものです。

1. コマンド行でオプションまたはプロパティーを指定します。例えば、次のようにします。

   ```
   java -Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump MyJavaClass
   ```

2. 該当のオプションを含むファイルを作成し、**-Xoptionsfile=**<*filename*> オプションを使用してこのファイルをコマンド行で指定します。

   オプション・ファイルでは、各オプションを改行して指定します。単一オプションを複数行にわたって記述する場合は、継続文字として「¥」文字を使用できます。コメント行を定義するには、「#」文字を使用します。オプション・ファイルで **-classpath** を指定することはできません。オプション・ファイルの例を以下に示します。

   ```
   #My options file
   -X<option1>
   -X<option2>=¥
   <value1>,¥
   <value2>
   -D<sysprop1>=<value1>
   ```

3. オプションを含む **IBM_JAVA_OPTIONS** という環境変数を作成します。例えば、次のようになります。

   ```
   export IBM_JAVA_OPTIONS="-Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump"
   ```

コマンド行では、最後に指定したオプションが最初のオプションより優位になります。例えば、「**-Xint -Xjit myClass**」というオプションを指定した場合、**-Xjit** というオプションは **-Xint** より優位になります。

システム・プロパティー

アプリケーションではシステム・プロパティーを使用することができ、これにより、ランタイム環境の情報を提供できます。

com.ibm.jvm.realtime
このプロパティーにより、Java アプリケーションは、WebSphere Real Time for Linux 環境内で実行中であるかどうかを判別できます。

アプリケーションが IBM WebSphere Real Time for RT Linux ランタイム環境内で実行中であり、**-Xrealtime** オプションを指定して開始されている場合、**com.ibm.jvm.realtime** プロパティーの値は「hard」になります。

アプリケーションが IBM WebSphere Real Time for RT Linux ランタイム環境内で実行中であり、**-Xrealtime** オプションを指定して開始されていない場合、**com.ibm.jvm.realtime** プロパティーは設定されません。

アプリケーションが IBM WebSphere Real Time ランタイム環境内で実行中の場合、**com.ibm.jvm.realtime** プロパティーの値は「soft」になります。

標準オプション

標準オプションの定義

-agentlib:<*libname*>**[=**<*options*>**]**
ネイティブ・エージェント・ライブラリー <*libname*> をロードします。例えば、**-agentlib:hprof** のように指定します。詳しくは、コマンド行で **-agentlib:jdwp=help** と **-agentlib:hprof=help** を指定してください。

-agentpath:*libname***[=**<*options*>**]**
絶対パス名でネイティブ・エージェント・ライブラリーをロードします。

-assert assert 関連オプションのヘルプを表示します。

-cp または -classpath <: で区切られたディレクトリーおよび *.zip* ファイルまたは *.jar* ファイル>
アプリケーション・クラスおよびリソースの検索パスを設定します。**-classpath** も **-cp** も使用しないで、**CLASSPATH** を設定しない場合、ユーザー・クラスパスは、デフォルトでは現行ディレクトリー (.) になります。

-D<property_name>=<*value*>
システム・プロパティーを設定します。

-help または -?
使用法メッセージを表示します。

-javaagent:<*jarpath*>**[=**<*options*>**]**
Java プログラミング言語エージェントをロードします。詳しくは、java.lang.instrument API の資料を参照してください。

-jre-restrict-search
バージョン検索の対象にユーザーのプライベート JRE を含めます。

-no-jre-restrict-search
バージョン検索の対象からユーザーのプライベート JRE を除外します。

-showversion
製品のバージョンを表示して継続します。

-verbose:[*class***,***gc***,***dynload***,***sizes***,***stack***,***jni***]**
詳細出力を使用可能にします。

-verbose:class
ロードされるクラスごとに、項目を stderr に書き込みます。

-verbose:gc
73 ページの『verbose:gc 情報の使用』を参照してください。

-verbose:dynload
各クラスが JVM によってロードされるときに、以下のような詳細情報を提供します。
- クラス名およびパッケージ
- .jar ファイル内にあったクラス・ファイルの場合、.jar の名前およびディレクトリー・パス
- クラスのサイズ、およびクラスのロードにかかった時間などの詳細

データは stderr に書き込まれます。出力の例を以下に示します。

```
<Loaded java/lang/String from /myjdk/sdk/jre/lib/i386/
softrealtime/jclSC160/vm.jar>
<Class size 17258; ROM size 21080; debug size 0>
<Read time 27368 usec; Load time 782 usec; Translate time 927 usec>
```

注: 共有クラス・キャッシュからロードされたクラスは **-verbose:dynload** 出力に表示されません。これらのクラスの情報を出力するには、**-verbose:class** を使用してください。

-verbose:sizes
JVM 内のスタックおよびヒープのために使用されたメモリーの量を示す情報を stderr に書き込みます。

-verbose:stack
Java および C のスタック使用量について説明する情報を stderr に書き込みます。

-verbose:jni
アプリケーションと JVM によって呼び出される JNI サービスについて説明する情報を stderr に書き込みます。

-version
非リアルタイム・モードのバージョン情報を表示します。

-version:<*value*>
指定したバージョンの実行を要求します。

-X 非標準オプションのヘルプを表示します。

非標準オプション

接頭部に **-X** が付いたオプションは非標準であり、予告なしに変更されることがあります。

IBM SDK for Java V7 ユーザー・ガイドに、非標準オプションに関する詳細情報が記載されています。この情報は、IBM SDK for Java 7 - コマンド行オプションで参照できます。

IBM WebSphere Real Time for Linux の補足情報は、以下のセクションに記載されています。

Metronome ガーベッジ・コレクターのオプション

Metronome ガーベッジ・コレクター・オプションの定義。

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM

ガーベッジ・コレクションが行われる原因の 1 つとして、ヒープでのメモリー不足があります。ヒープに空き領域がなくなった場合、**-Xgc:synchronousGCOnOOM** を使用すると、アプリケーションが停止し、その間にガーベッジ・コレクションによって未使用オブジェクトが削除されます。それでも空き領域が足りなくなる場合は、目標使用率を低くして、より長い時間をかけてガーベッジ・コレクションを完了させることを考慮してください。 **-Xgc:nosynchronousGCOnOOM** を設定すると、ヒープ・メモリーがいっぱいになったときにアプリケーションが停止してメモリー不足メッセージを出すようになります。デフォルトは **-Xgc:synchronousGCOnOOM** です。

-Xnoclassgc

クラス・ガーベッジ・コレクションを使用不可にします。このオプションは、JVM で使用されていない Java クラスに関連するストレージのガーベッジ・コレクションをオフに切り替えます。デフォルトの動作は **-Xnoclassgc** です。

-Xgc:targetPauseTime=*N*

ガーベッジ・コレクション休止時間を設定します。ここで、*N* はミリ秒単位の時間です。このオプションを指定すると、GC は指定した値を超えない休止時間で動作します。このオプションが指定されていない場合、デフォルトの休止時間は 3 ミリ秒です。例えば、**-Xgc:targetPauseTime=20** を指定して実行すると、GC 動作中の休止時間は 20 ミリ秒未満になります。

-Xgc:targetUtilization=*N*

アプリケーション使用率を N% に設定すると、ガーベッジ・コレクターは最大で各時間間隔の (100-N)% まで使用しようと試みます。妥当な値は 50% から 80% までの間です。割り振り速度の低いアプリケーションは、90% で実行できる可能性があります。デフォルトは 70% です。

次の例では、ヒープ・メモリーの最大サイズが 30 MB に設定されています。アプリケーションの目標使用率が 75% に設定されているため、ガーベッジ・コレクターは最大で各時間間隔の 25% まで使用するように試みます。

```
java -Xgcpolicy:metronome -Xmx30m -Xgc:targetUtilization=75 Test
```

-Xgc:threads=*N*

実行する GC スレッドの数を指定します。デフォルトは、そのプロセスで使用できるプロセッサー・コアの数です。指定可能な最大値は、オペレーティング・システムで使用できるプロセッサーの数です。

-Xgc:verboseGCCycleTime=N

N は、要約情報をダンプする時間 (ミリ秒単位) です。

注: 要約情報は、サイクル・タイムに指定された時間ちょうどにダンプされるのではなく、この時間の基準を満たす最後のガーベッジ・コレクション・イベント時にダンプされます。

-Xmx<*size*>

Java ヒープ・サイズを指定します。他のガーベッジ・コレクション戦略とは異なり、リアルタイム Metronome GC はヒープ拡張をサポートしません。初期または最大ヒープ・サイズを指定するオプションはありません。最大ヒープ・サイズのみを指定できます。

JVM のデフォルト設定

Real Time JVM の実行環境に変更が行われなかった場合、この JVM にはデフォルト設定が適用されます。一般的な設定をリファレンスとして示します。

デフォルト設定は、環境変数を使用するか、JVM の開始時にコマンド行パラメーターを使用することで変更できます。一般的な JVM 設定の一部を以下の表に示します。最後の列は動作の変更方法を示します。この列には以下のキーが適用されます。

- **e** - 環境変数のみが設定を制御します。
- **c** - コマンド行パラメーターのみが設定を制御します。
- **ec** - 環境変数とコマンド行パラメーターの両方が設定を制御しますが、コマンド行パラメーターが優先します。

この情報はクイック・リファレンスとして提供されるもので、包括的なものではありません。

JVM の設定	デフォルト	影響する設定
Javadump	使用可能	ec
メモリー不足時の Javadump	使用可能	ec
Heapdump	使用不可	ec
メモリー不足時の Heapdump	使用可能	ec
Sysdump	使用可能	ec
ダンプ・ファイルの生成場所	現行ディレクトリー	ec
verbose 出力	使用不可	c
ブート・クラスパス検索	使用不可	c
JNI チェック	使用不可	c
リモート・デバッグ	使用不可	c
厳密な適合チェック	使用不可	c
クイック・スタート	使用不可	c
リモート・デバッグ情報サーバー	使用不可	c
シグナル通知の削減	使用不可	c
シグナル・ハンドラーのチェーニング	使用可能	c
クラスパス	未設定	ec
クラス・データの共有	使用不可	c
アクセシビリティー・サポート	使用可能	e
JIT コンパイラー	使用可能	ec
AOT コンパイラー (AOT は、共有クラスも使用可能でない限り、JVM では使用されません)	使用可能	c

JVM の設定	デフォルト	影響する設定
JIT デバッグ・オプション	使用不可	c
アルゴリズムで太字に指定したフォントの Java2D での最大サイズ	14 ポイント	e
Java2D でスケーラブル・フォントのレンダー・ビットマップを使用	使用可能	e
Java2D でのフリータイプ・フォントのラスタライズ	使用可能	e
Java2D で AWT フォントを使用	使用不可	e
デフォルト・ロケール	なし	e
プラグイン開始までの待ち時間	ゼロ	e
一時ディレクトリー	/tmp	e
プラグインのリダイレクト	なし	e
IM 切り替え	使用不可	e
IM 修飾子	使用不可	e
スレッド・モデル	該当なし	e
Java スレッド 32 ビット用の初期スタック・サイズ。 使用法: **-Xiss<サイズ>**	2 KB	c
Java スレッド 32 ビット用の最大スタック・サイズ。 使用法: **-Xss<サイズ>**	256 KB	c
OS スレッド 32 ビット用のスタック・サイズ。 使用法: **-Xmso<サイズ>**	256 KB	c
初期ヒープ・サイズ。 使用法: **-Xms<サイズ>**	64 MB	c
最大 Java ヒープ・サイズ。 使用法: **-Xmx<サイズ>**	使用可能メモリーの半分のサイズ (最小 16 MB、最大 512 MB)	c
アプリケーションのターゲット時間間隔の利用。ガーベッジ・コレクターはその余剰時間を使用しようとします。使用法: **-Xgc:targetUtilization=<percentage>**	70%	c
実行するガーベッジ・コレクター・スレッドの数。使用法: **-Xgc:threads=<value>**	プロセスで使用可能なプロセッサー・コアの数。	c
-Xrealtime モードでスコープ・メモリーに割り振り可能なメモリーの最大量。使用法: **-Xgc:scopedMemoryMaximumSize=<size>**	8 MB	c
-Xrealtime モードでの永久メモリー域のサイズの設定。使用法: **-Xgc:immortalMemorySize=<size>**	16 MB	c

注: 「使用可能メモリー」とは、実際の (物理) メモリーまたは **RLIMIT_AS** 値のいずれかの最小値です。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

- JIMMAIL@uk.ibm.com (Hursley Java Technology Center (JTC) 連絡先)

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

プライバシー・ポリシーに関する考慮事項

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、「ソフトウェア・オファリング」により個人情報が収集されることはありません。 IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらのCookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項を確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、 お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、IBM の『IBM オンラインでのプライバシー・ステートメント』

(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および『IBM Software Products and Software-as-a-Service Privacy Statement』(http://www.ibm.com/software/info/product-privacy) を参照してください。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Intel および Itanium は、Intel Corporation の米国およびその他の国における商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21