

IBM WebSphere Real Time for Linux
Version 3

Benutzerhandbuch



IBM WebSphere Real Time for Linux
Version 3

Benutzerhandbuch



Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter Kapitel 11, „Bemerkungen“, auf Seite 73 gelesen werden.

Erste Ausgabe (August 2011)

Diese Ausgabe des Benutzerhandbuchs bezieht sich auf IBM WebSphere Real Time for Linux, Version 3 und auf alle nachfolgenden Releases und Änderungen, bis in neuen Ausgaben etwas anderes angegeben wird.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
IBM WebSphere Real Time for Linux Version 3, User's Guide,
herausgegeben von International Business Machines Corporation, USA

© Copyright International Business Machines Corporation 2003, 2011
© Copyright IBM Deutschland GmbH 2011

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
TSC Germany
Kst. 2877
August 2011

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Tabellen	vii
Vorwort.	ix
Kapitel 1. Einführung	1
Übersicht über WebSphere Real Time for Linux.	1
Neuerungen	1
Vorteile	2
Kapitel 2. Einführung in IBM WebSphere Real Time for Linux.	3
Einführung in den Metronom-Garbage-Collector	3
Kapitel 3. Planung	5
Migration	5
Unterstützte Umgebungen.	5
Wichtige Faktoren	6
Kapitel 4. WebSphere Real Time for Linux installieren	7
Installationsdateien	7
Installation über ein InstallAnywhere-Paket durchführen	7
Beaufsichtigte Installation durchführen	9
Unbeaufsichtigte Installation durchführen	9
Unterbrochene Installation	11
Bekannte Probleme und Einschränkungen	11
Pfad festlegen	12
Klassenpfad festlegen	13
Installation testen	14
WebSphere Real Time for Linux deinstallieren.	14
Kapitel 5. IBM WebSphere Real Time for Linux-Anwendungen ausführen	17
Metronom-Garbage-Collector verwenden	17
Pausezeit steuern	17
Prozessorauslastung steuern.	22
Einschränkung für den Metronom-Garbage-Collector.	22
Kapitel 6. Anwendungen entwickeln	23
Echtzeitorientierte Musterhashzuordnung	23
Kapitel 7. Leistung.	25

Gemeinsame Nutzung von Klassendaten auf verschiedenen JVMs	26
----------------------------------------------------------------------	----

Kapitel 8. Sicherheit	27
Sicherheitsaspekte für den Cache für gemeinsam genutzte Klassen	27

Kapitel 9. Fehlerbehebung und Unterstützung	29
Allgemeine Problembestimmungsmethoden	29
Fehlerbestimmung unter Linux	29
Fehlerbestimmung für NLS	34
Fehlerbestimmung für ORB	35
Fehlerbehebung von OutOfMemory-Fehlern	36
Fehler aufgrund abnormaler Speicherbedingungen (OutOfMemoryErrors) diagnostizieren	36
Diagnosetools verwenden	39
Speicherauszugsagenten verwenden	40
Java-Speicherauszug verwenden	43
Heapspeicherauszug verwenden	48
Systemspeicherauszüge und die Anzeigefunktion für Speicherauszüge verwenden	51
Tracefunktion von Java-Anwendungen und der JVM	52
Fehlerbestimmung für JIT und AOT	53
Diagnostics-Collector	59
Garbage-Collector-Diagnose	60
Diagnose bei gemeinsam genutzten Klassen	65
Mit der JVMTI arbeiten	65
Diagnostic Tool Framework for Java verwenden	66
IBM Monitoring and Diagnostic Tools for Java - Health Center verwenden	66

Kapitel 10. Referenz	67
Befehlszeilenooptionen	67
Java-Optionen und Systemeigenschaften angeben	67
Systemeigenschaften	67
Standardoptionen	68
Vom Standard abweichende Optionen	69
Standard Einstellungen für die JVM	70

Kapitel 11. Bemerkungen	73
Marken.	74

Index	75
------------------------	-----------

Abbildungsverzeichnis

- | | | | |
|----|--------------------------------------------------------------------------------------------------------------------------------------|----|-------------------------------------------------------------------------------------------|
| 1. | Tatsächliche Garbage-Collection-Pausezeiten, wenn für die Zielpausezeit der Standardwert (3 Millisekunden) festgelegt wurde. | 3. | Tatsächliche Pausezeiten, wenn für die Zielpausezeit 10 Millisekunden festgelegt wurden . |
| 2. | Tatsächliche Pausezeiten, wenn für die Zielpausezeit 6 Millisekunden festgelegt wurden . | 4. | Tatsächliche Pausezeiten, wenn für die Zielpausezeit 15 Millisekunden festgelegt wurden . |

Tabellen

1. Getestete Linux-Umgebungen 5
2. Threadnamen in IBM WebSphere Real Time for
Linux 47

Vorwort

In diesem Handbuch bzw. Benutzerhandbuch erhalten Sie allgemeine Informationen zu IBM® WebSphere Real Time for Linux.

Kapitel 1. Einführung

Diese Informationen führen Sie in IBM WebSphere Real Time for Linux ein.

- „Übersicht über WebSphere Real Time for Linux“
- „Neuerungen“
- „Vorteile“ auf Seite 2

Übersicht über WebSphere Real Time for Linux

WebSphere Real Time for Linux bündelt Echtzeitfunktionalität mit der IBM J9 Virtual Machine (JVM).

WebSphere Real Time for Linux ist eine Java Runtime Environment mit einem Software-Development-Kit, das IBM SDK for Java durch Echtzeitfunktionalität erweitert. Anwendungen, die von präzisen Antwortzeiten abhängig sind, können die Echtzeitfunktionen von WebSphere Real Time for Linux über Standard-Java-Technologie nutzen.

Features

Echtzeitanwendungen benötigen eine konsistente Ausführung anstatt absoluter Geschwindigkeit.

Die Hauptproblemstellungen bei der Implementierung von Echtzeitanwendungen mit traditionellen JVMs lauten wie folgt:

- Unvorhersehbare (potenziell lange) Verzögerungen aufgrund der Garbage-Collection-Aktivität
- Verzögerungen für die Methodenlaufzeit bei JIT-Kompilierung und erneuter Kompilierung mit unterschiedlicher Ausführungszeit
- Willkürliche Betriebssystemzeitplanung

WebSphere Real Time for Linux entfernt diese Hindernisse durch die Bereitstellung folgender Komponenten:

- Metronom-Garbage-Collector, ein inkrementeller und deterministischer Garbage-Collector mit sehr kurzen Pausezeiten

Neuerungen

Dieser Abschnitt enthält eine Einführung in Änderungen für IBM WebSphere Real Time for Linux.

WebSphere Real Time for LinuxV3

WebSphere Real Time for Linux Version 3 ist eine Erweiterung von IBM SDK for Java 7, die auf den in diesem Release verfügbaren Features und Funktionen aufbaut, um Echtzeit-Funktionalität einzuschließen. Ältere Versionen von WebSphere Real Time for Linux basierten auf früheren Releases von IBM SDK for Java.

Informationen zu den Neuerungen finden Sie in Neuerungen im Information Center für IBM SDK for Java 7.

Pausezeiten für den Metronom-Garbage-Collector steuern

Der Metronom-Garbage-Collector pausiert standardmäßig 3 Millisekunden lang zwischen Garbage-Collection-Zyklen. Sie können diesen Wert über eine neue Befehlszeilenoption ändern, um die Pausezeit zu steuern. Weitere Informationen zu dieser Option finden Sie in „Pausezeit steuern“ auf Seite 17.

Komprimierte Verweise

Der Metronom-Garbage-Collector unterstützt nun nicht komprimierte Verweise sowie komprimierte Verweise auf 64-Bit-Plattformen. Informationen zur Auswirkung auf die Leistung finden Sie in Kapitel 7, „Leistung“, auf Seite 25.

Vorteile

Die Vorteile der Echtzeitumgebung bestehen darin, dass Java-Anwendungen mit einem höheren Grad an Vorhersagbarkeit ausgeführt werden als die Standard-JVM und konsistentes Ablaufsteuerungsverhalten für Ihre Java-Anwendung liefern. Hintergrundaktivitäten wie Kompilierung und Garbage-Collection finden zu angegebenen Zeiten statt. Dadurch entfallen nicht erwartete Hintergrundaktivitätsspitzen, wenn Ihre Anwendung ausgeführt wird.

Sie können von diesen Vorteilen profitieren, indem Sie die JVM durch die Metronom-Garbage-Collection-Echtzeittechnologie erweitern.

Kapitel 2. Einführung in IBM WebSphere Real Time for Linux

In diesem Abschnitt werden die Schlüsselkomponenten von IBM WebSphere Real Time for Linux eingeführt.

- „Einführung in den Metronom-Garbage-Collector“

Einführung in den Metronom-Garbage-Collector

Der Metronom-Garbage-Collector ersetzt den Standard-Garbage-Collector in WebSphere Real Time for Linux.

Der Hauptunterschied zwischen der Metronom-Garbage-Collection und der Standard-Garbage-Collection ist, dass die Metronom-Garbage-Collection in kleinen unterbrechbaren Schritten durchgeführt wird, wohingegen die Standard-Garbage-Collection die Anwendung stoppt, während sie Garbage markiert und erfasst.

Beispiel:

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

Das Beispiel gibt an, dass Ihre Anwendung alle 60 ms 80 % ausgeführt wird. Die verbleibenden 20 % der Zeit kann für die Garbage-Collection verwendet werden, sofern zu erfassender Garbage vorhanden ist. Der Metronom-Garbage-Collector garantiert bestimmte Auslastungen, vorausgesetzt ihm wurden ausreichende Ressourcen zugeteilt. Die Garbage-Collection fängt an, wenn der freie Speicherplatz im Heapspeicher unter einen dynamisch ermittelten Schwellenwert fällt.

Metronom-Garbage-Collection und Klassenentladung

Metronom unterstützt das Entladen von Klassen genau so wie ein Standard-Java-Entwicklerkit. Aufgrund der hiermit verbundenen Arbeit kann es beim Entladen von Klassen jedoch während der Garbage-Collection-Aktivitäten zu Pausenzeitaustrisern kommen.

Metronom-Garbage-Collector-Threads

Der Metronom-Garbage-Collector besteht aus zwei Typen von Threads: ein einzelner Alarmthread und eine Anzahl GC-Threads. Die GC verwendet standardmäßig einen Thread für jeden logischen aktiven Prozessor, der für das Betriebssystem verfügbar ist. Dies ermöglicht die effizienteste Parallelverarbeitung während der GC-Zyklen. Ein GC-Zyklus bedeutet die Zeit zwischen der GC-Auslösung und dem Abschluss der Garbagefreigabe. Je nach der Größe des Java-Heapspeichers kann die abgelaufene Zeit für einen vollständigen GC-Zyklus mehrere Sekunden betragen. Ein GC-Zyklus enthält gewöhnlich Hunderte von GC-Quanten. Diese Quanten sind die sehr kurzen Pausen für den Anwendungscode, die in der Regel 3 Millisekunden dauern. Mit **-verbose:gc** können Sie Ergebnisberichte von Zyklen und Quanten abrufen. Weitere Informationen finden Sie in „Informationen von 'verbose:gc' verwenden“ auf Seite 60. Sie können die Anzahl GC-Threads für die JVM mit der Option **-Xgcthreads** festlegen.

Die Erhöhung von **-Xgcthreads** über den Standardwert hinaus bringt keine Vorteile. Die Reduzierung von **-Xgcthreads** kann die CPU-Gesamtbelastung während der GC-Zyklen verkleinern, die GC-Zyklen werden jedoch verlängert.

Anmerkung: Die Ziele für die GC-Quantendauer bleiben konstant bei 3 Millisekunden.

Sie können die Anzahl Alarmthreads für die JVM nicht ändern.

Der Metronom-Garbage-Collector prüft die JVM periodisch daraufhin, ob der Heapspeicher über ausreichend freien Speicherplatz verfügt. Wenn die freie Speicherplatzkapazität unter den Grenzwert fällt, löst der Metronom-Garbage-Collector den Start der Garbage-Collection durch die JVM aus.

Alarmthread

Der einzelne Alarmthread garantiert die Verwendung minimaler Ressourcen. Er wird in regelmäßigen Intervallen aktiviert und prüft Folgendes:

- Freie Speicherplatzkapazität im Heapspeicher
- Ob die Garbage-Collection zurzeit durchgeführt wird

Wenn der freie Speicherplatz nicht ausreicht und keine Garbage-Collection durchgeführt wird, weist der Alarmthread die Collection-Threads an, die Garbage-Collection zu starten. Der Alarmthread führt keine Aktivitäten aus, bis er das nächste Mal die JVM prüft.

Collection-Threads

Die Collection-Threads führen die Garbage-Collection aus.

Nach dem Abschluss des Garbage-Collection-Zyklus prüft der Metronom-Garbage-Collector, wie viel Heapspeicher frei ist. Wenn weiterhin nicht genügend Heapspeicher frei ist, wird ein weiterer Garbage-Collection-Zyklus mit derselben Trigger-ID gestartet. Wenn ausreichend Heapspeicher frei ist, wird der Trigger beendet und werden die Garbage-Collection-Threads gestoppt. Der Alarmthread überwacht weiterhin den freien Heapspeicher und löst bei Bedarf einen weiteren Garbage-Collection-Zyklus aus.

Weitere Informationen zum Verwenden des Metronom-Garbage-Collectors finden Sie in „Metronom-Garbage-Collector verwenden“ auf Seite 17.

Kapitel 3. Planung

Lesen Sie diesen Abschnitt vor der Installation von WebSphere Real Time for Linux.

-
- „Unterstützte Umgebungen“
-
- „Wichtige Faktoren“ auf Seite 6

Migration

Sie können Ihre Standard-Java-Anwendungen unter WebSphere Real Time for Linux ohne Änderung ausführen.

Unterstützte Umgebungen

IBM WebSphere Real Time for Linux wird auf bestimmten Hardwareplattformen und unter bestimmten Betriebssystemen unterstützt.

IBM WebSphere Real Time for Linux

Die folgenden Plattformarchitekturen werden unterstützt:

- Intel-Architektur, 32 Bit (IA-32)
 - Pentium 4
 - Pentium Xeon
 - Pentium M
 - Pentium D und Äquivalente
- AMD64/EM64T
- IBM POWER 32
- IBM POWER 64

Anmerkung: Pentium 3-Hardware wird nicht mehr unterstützt.

Die folgenden Betriebssysteme werden unterstützt:

Tabelle 1. Getestete Linux-Umgebungen

Hardware	IA-32 32-Bit		AMD64/EM64T 64-Bit	
	32-Bit	64-Bit	32-Bit	64-Bit
SDK-Adressraum				
RHEL 5 Update 7	Ja		Ja	Ja
RHEL 6 Update 1	Ja		Ja	Ja
SLES 11 SP1	Ja		Ja	Ja

Anmerkung: SLES 9, SLES 10 und RHEL 4 werden nicht unterstützt.

Wichtige Faktoren

Sie müssen bei der Verwendung von WebSphere Real Time for Linux eine Anzahl Faktoren beachten.

- Sofern möglich, führen Sie nicht mehrere Echtzeit-JVMs auf demselben System aus. Der Grund hierfür ist, dass Sie dann über mehrere Garbage-Collector verfügen würden. Jede einzelne JVM kennt nicht die Hauptspeicherbereiche der anderen JVMs. Eine Auswirkung ist, dass GC-Zyklen und -Pausezeiten JVM-übergreifend nicht koordiniert werden können. Dies bedeutet, dass sich eine JVM negativ auf die GC-Leistung einer anderen JVM auswirken kann. Wenn Sie mehrere JVMs verwenden müssen, stellen Sie sicher, dass jede JVM mithilfe des Befehls **taskset** an eine bestimmte Untergruppe von Prozessoren gebunden wurde.
- Die gemeinsam genutzten Caches, die von früheren Releases von WebSphere Real Time for Linux zum Speichern von vorkompiliertem Code und Klassen verwendet wurden, sind nicht mit den Caches kompatibel, die von diesem Release von WebSphere Real Time for Linux verwendet werden. Sie müssen den Inhalt der früheren Caches neu generieren.
- Bei Verwendung von Caches für gemeinsam genutzte Klassen darf der Cachename 53 Zeichen nicht überschreiten.

Kapitel 4. WebSphere Real Time for Linux installieren

Führen Sie folgende Schritte aus, um das Produkt zu installieren.

- „Installationsdateien“
- „Installation über ein InstallAnywhere-Paket durchführen“
 - „Beaufsichtigte Installation durchführen“ auf Seite 9
 - „Unbeaufsichtigte Installation durchführen“ auf Seite 9
 - „Bekannte Probleme und Einschränkungen“ auf Seite 11
- „Pfad festlegen“ auf Seite 12
- „Klassenpfad festlegen“ auf Seite 13
- „Installation testen“ auf Seite 14
- „WebSphere Real Time for Linux deinstallieren“ auf Seite 14

Installationsdateien

Sie benötigen die folgenden Installationsdateien.

IBM WebSphere Real Time for Linux wird in zwei InstallAnywhere-Pakettypen geliefert.

Installierbare Pakete

Installierbare Pakete konfigurieren Ihr System. Beispielsweise können die Programme Umgebungsvariablen festlegen.

- wrt-3.0-0.0-linux-<Architektur>-sdk.bin
- wrt-3.0-0.0-linux-<Architektur>-jre.bin

Archivierungspakete

Diese Pakete extrahieren die Dateien auf Ihr System, führen jedoch keine Konfiguration aus.

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

Anmerkung: <Architektur> ist Ihre Plattformarchitektur: x86_32 oder x86_64.

Installation über ein InstallAnywhere-Paket durchführen

Diese Pakete stellen ein interaktives Programm zur Verfügung, das Sie durch die Installationsoptionen führt. Sie können das Programm über eine grafische Benutzerschnittstelle oder über eine Systemkonsole ausführen.

Vorbereitende Schritte

Die folgenden beiden gemeinsam genutzten Bibliotheken müssen auf Ihrem System vorhanden sein:

- GNU-C-Bibliothek Version 2.3 (glibc)
- libstdc++.so.5

Ist die gemeinsam genutzte Bibliothek libstdc++.so.5 nicht vorhanden, wird bei der Installation möglicherweise ein Java-Kernspeicherauszug mit den folgenden Fehlern ausgegeben:

```
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
```

Wenn Sie ein installierbares Paket installieren, muss das Tool **rpm-build** auf Ihrem System installiert sein. Andernfalls kann das Installationsprogramm das neue Paket nicht in der RPM-Datenbank registrieren. Durch Eingabe des folgenden Befehls können Sie ermitteln, ob das Tool **rpm-build** installiert ist:

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

Informationen zu diesem Vorgang

Die InstallAnywhere-Pakete haben die Dateierweiterung `.bin`.

Es gibt zwei Typen von Paketen:

Installierbares Paket

Im Rahmen der Installation dieser Pakete erfolgt auch eine Konfiguration Ihres Systems, zum Beispiel durch Festlegen von Umgebungsvariablen.

Archivierungspaket

Im Rahmen der Installation dieser Pakete werden die Dateien auf Ihr System extrahiert, es wird jedoch keine Konfiguration durchgeführt.

Vorgehensweise

- Soll das Paket interaktiv installiert werden, führen Sie eine beaufsichtigte Installation durch.
- Soll das Paket ohne weitere Benutzerinteraktion installiert werden, führen Sie eine unbeaufsichtigte Installation durch. Die Auswahl dieser Option kann sinnvoll sein, wenn mehrere Systeme installiert werden sollen.
- Wenn der Installationsprozess abgeschlossen ist, führen Sie die Konfigurationsschritte in diesem Abschnitt wie das Festlegen der Umgebungsvariablen für Pfad und Klassenpfad aus.

Ergebnisse

Das Produkt wird installiert.

Anmerkung: Unterbrechen Sie den Installationsprozess nicht (z. B. durch Drücken von Strg+C). Wenn Sie den Prozess unterbrechen, müssen Sie das Produkt möglicherweise erneut installieren. Weitere Informationen finden Sie in „Unterbrochene Installation“ auf Seite 11.

Wenn Sie ein installierbares Paket verwenden, werden möglicherweise Nachrichten ausgegeben, die darauf hinweisen, dass ein Problem aufgetreten ist. Bei der Installation der Archivierungspakete werden keine Nachrichten generiert. Einige der Nachrichten, die bei Verwendung eines installierbaren Pakets unter Umständen ausgegeben werden, sind in der folgenden Liste aufgeführt:

Das Installationsprogramm kann Ihre Konfiguration nicht ausführen und wird jetzt beendet.

Diese Fehlermeldung wird ausgegeben, wenn die Benutzer-ID nicht zur Ausführung des Installationsprozesses berechtigt ist. Da das Installationsprogramm nicht fortgesetzt werden kann, wird es beendet. Starten Sie die

Installation erneut und verwenden Sie dabei eine Benutzer-ID, die über Rootberechtigung verfügt, um dieses Problem zu beheben.

An RPM package is already installed. Uninstall the package before proceeding. Diese Nachricht weist darauf hin, dass bereits ein RPM-Paket installiert ist. Da das Installationsprogramm nicht fortgesetzt werden kann, wird es beendet. Deinstallieren Sie zur Behebung des Problems das RPM-Paket, bevor Sie fortfahren.

Beaufsichtigte Installation durchführen

Installieren Sie das Produkt interaktiv aus einem InstallAnywhere-Paket.

Vorbereitende Schritte

Überprüfen Sie vor Beginn des Installationsprozesses, ob die folgenden Bedingungen gegeben sind:

- Wurde WebSphere Real Time for Linux zuvor aus einem RPM-Paket installiert, müssen Sie dieses Paket erst deinstallieren, bevor sie fortfahren.
- Sie müssen über eine Benutzer-ID mit Rootberechtigung verfügen.

Vorgehensweise

1. Laden Sie die Installationspaketdatei in ein temporäres Verzeichnis herunter.
2. Wechseln Sie in das temporäre Verzeichnis.
3. Starten Sie den Installationsprozess, indem Sie `./package` an einer Shelleingabeaufforderung eingeben. Dabei steht *Paket* für den Namen des Pakets, das installiert wird.
4. Wählen Sie eine Sprache aus der im Fenster des Installationsprogramms angezeigten Liste aus und klicken Sie dann auf **Next**. Die Liste der verfügbaren Sprachen basiert auf der Ländereinstellung Ihres Systems.
5. Lesen Sie die Lizenzvereinbarung. Führen Sie dabei mithilfe der Bildlaufleiste einen Bildlauf bis zum Ende der Lizenzinformationen durch. Wenn Sie mit der Installation fortfahren wollen, müssen Sie die Bedingungen der Lizenzvereinbarung akzeptieren. Klicken Sie zum Akzeptieren der Bedingungen auf das Optionsfeld und klicken Sie dann auf **OK**.

Anmerkung: Das Optionsfeld zum Akzeptieren der Lizenzvereinbarung können Sie erst auswählen, wenn Sie bis zum Ende der Lizenzvereinbarung gelesen haben.

6. Sie werden aufgefordert, das Zielverzeichnis für die Installation auszuwählen. Wenn die Installation nicht im Standardverzeichnis durchgeführt werden soll, klicken Sie auf **Choose**, um über das Browserfenster ein alternatives Verzeichnis auszuwählen. Klicken Sie nach der Auswahl des Installationsverzeichnisses auf **Next**, um fortzufahren.
7. Sie werden aufgefordert, die von Ihnen gewählten Einstellungen zu prüfen. Wenn Sie die gewählten Einstellungen ändern wollen, klicken Sie auf **Previous**. Sind die von Ihnen gewählten Einstellungen richtig, klicken Sie auf **Install**, um mit der Installation fortzufahren.
8. Wenn der Installationsprozess abgeschlossen ist, klicken Sie auf **Done**, um den Vorgang zu beenden.

Unbeaufsichtigte Installation durchführen

Wenn Sie mehrere System installieren müssen und bereits wissen, welche Installationsoptionen Sie verwenden wollen, haben Sie auch die Option, einen unbeaufsich-

tigten Installationsprozess durchzuführen. Dazu führen Sie zunächst *eine* Installation über den beaufsichtigten Installationsprozess durch und verwenden anschließend die daraus resultierende Antwortdatei, um weitere Installationen ohne Benutzerinteraktion durchzuführen.

Vorgehensweise

1. Erstellen Sie eine Antwortdatei, indem Sie eine beaufsichtigte Installation durchführen. Verwenden Sie eine der folgenden Optionen:

- Verwenden Sie die grafische Benutzerschnittstelle (GUI) und geben Sie an, dass das Installationsprogramm eine Antwortdatei erstellen soll. Die Antwortdatei hat den Namen `installer.properties` und wird im Installationsverzeichnis erstellt.
- Hängen Sie unter Verwendung der Befehlszeile die Option `-r` an den Befehl für die beaufsichtigte Installation an und geben Sie dabei den vollständigen Pfad zur Antwortdatei an. Beispiel:

```
./Paket -r /Pfad/installer.properties
```

Beispielinhalt einer Antwortdatei:

```
INSTALLER_UI=silent  
USER_INSTÄLL_DIR=/Mein_Verzeichnis
```

In diesem Beispiel steht `/Mein_Verzeichnis` für das von Ihnen gewählte Zielinstallationsverzeichnis für die Installation.

2. Optional: Ändern Sie in der Antwortdatei nach Bedarf die Optionen.

Anmerkung: Bei Archivierungspaketen tritt das folgende bekannte Problem auf: Bei Installationen, die mithilfe einer Antwortdatei durchgeführt werden, wird auch nach einer Änderung des Verzeichnisses in der Antwortdatei weiterhin das Standardverzeichnis verwendet. Ist im Standardverzeichnis eine vorherige Installation vorhanden, wird sie überschrieben.

Wenn Sie mehrere Antwortdateien erstellen, jede mit jeweils anderen Installationsoptionen, geben Sie für jede Antwortdatei einen eindeutigen Namen im Format `MeineDatei.properties` an.

3. Optional: Generieren Sie eine Protokolldatei. Da Sie die Installation im Hintergrund durchführen, werden am Ende des Installationsprozesses keine Statusnachrichten angezeigt. Führen Sie die folgenden Schritte durch, damit eine Protokolldatei generiert wird, in der der Installationsstatus angegeben ist:

- a. Legen Sie mithilfe des folgenden Befehls die erforderlichen Systemeigenschaften fest.

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

- b. Legen Sie die folgende Umgebungsvariable fest, um die Protokollausgabe an die Konsole zu senden.

```
export LAX_DEBUG=1
```

4. Starten Sie eine unbeaufsichtigte Installation, indem Sie das Installationsprogramm für das Paket mit der Option `-i` (Installation im Hintergrund) und der Option `-f` (Angabe der Antwortdatei) festlegen. Beispiel:

```
./Paket -i silent -f /Pfad/installer.properties 1>Konsole.txt 2>&1
```

```
./Paket -i silent -f /Pfad/MeineDatei.properties 1>Konsole.txt 2>&1
```

Sie können einen vollständig qualifizierten Pfad oder einen relativen Pfad zu der Eigenschaftendatei verwenden. In diesen Beispielen leitet die Zeichenfolge `1>Konsole.txt 2>&1` die Informationen zum Installationsprozess von den Daten-

strömen stderr und stdout an die Protokolldatei `Konsole.txt` im aktuellen Verzeichnis um. Prüfen Sie diese Protokolldatei, wenn Sie denken, dass bei der Installation ein Problem aufgetreten ist.

Anmerkung: Wenn Ihr Installationsverzeichnis mehrere Antwortdateien enthält, wird die Standardantwortdatei `installer.properties` verwendet.

Unterbrochene Installation

Wird das Installationsprogramm für das Paket während der Installation unerwartet gestoppt (z. B. durch Drücken von Strg+C), wird die Installation beschädigt und das Produkt kann nicht deinstalliert oder neu installiert werden. Wenn Sie versuchen, eine Deinstallation oder Neuinstallation durchzuführen, wird möglicherweise eine Nachricht ausgegeben, dass ein schwerwiegender Anwendungsfehler aufgetreten ist.

Informationen zu diesem Vorgang

Dieses Problem können Sie beheben, indem Sie Dateien löschen und wie in den folgenden Schritten beschrieben eine Neuinstallation durchführen.

Vorgehensweise

1. Löschen Sie die Registry-Datei `/var/.com.zerog.registry.xml`.
2. Löschen Sie das Verzeichnis, das die Installation enthält, sofern eines erstellt wurde. Beispielsweise `/opt/IBM/javawrt3[_64]/`.
3. Führen Sie das Installationsprogramm erneut aus.

Bekannte Probleme und Einschränkungen

In Verbindung mit den InstallAnywhere-Paketen sind bekannte Probleme und Einschränkungen zu beachten.

- Ist die gemeinsam genutzte Bibliothek `libstdc++.so.5` nicht auf Ihrem System vorhanden, schlägt die Installation fehl und es wird ein Java-Kernspeicherauszug erstellt. Weitere Informationen finden Sie in „Installation über ein InstallAnywhere-Paket durchführen“ auf Seite 7.
- Die grafische Benutzerschnittstelle des Installationspakets unterstützt das Sprachausgabeprogramm Orca nicht. Alternativ zu der grafischen Benutzerschnittstelle können Sie den Modus für unbeaufsichtigte Installationen verwenden.
- Wenn Sie `./Paket` nach einer Installation eingeben, um das Programm erneut zu starten, wird vom Programm die folgende Nachricht angezeigt:

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

Wenn Sie die Eingabetaste drücken, um den Standardwert zu übernehmen, reagiert das Programm nicht. Geben Sie eine Zahl ein und drücken Sie dann die Eingabetaste.

- Wenn Sie das Paket installieren und dann versuchen, eine weitere Installation in einem anderen Modus durchzuführen (z. B. im Konsolen- oder Hintergrundsmodus), wird möglicherweise die folgende Fehlermeldung ausgegeben:

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

Diese Nachricht wird in der Regel nicht ausgegeben, wenn nach einer Installation im GUI-Modus beim erneuten Ausführen des Installationsprogramms der Konsolenmodus verwendet wird. Tritt dieser Fehler auf, wenn Sie das Programm lediglich ausführen, um die Deinstallationsoption auszuwählen (nur bei installierbaren Paketen), verwenden Sie stattdessen den Befehl `./_uninstall/uninstall` wie in „WebSphere Real Time for Linux deinstallieren“ auf Seite 14 beschrieben.

Nur installierbare Pakete

- Die InstallAnywhere-Pakete können nicht verwendet werden, um ein Upgrade für eine vorhandene Installation durchzuführen. Wenn Sie für WebSphere Real Time for Linux ein Upgrade durchführen wollen, müssen Sie zuerst alle vorherigen Versionen deinstallieren.
- Sie können nicht zwei verschiedene Instanzen derselben Version von WebSphere Real Time for Linux auf demselben System installieren, auch nicht, wenn Sie verschiedene Installationsverzeichnisse verwenden. Es kann z. B. nicht gleichzeitig WebSphere Real Time for Linux Version 3 im Verzeichnis `/previous` und eine Serviceaktualisierung von WebSphere Real Time for Linux im Verzeichnis `/current` installiert werden. Das Installationsprogramm überprüft die Versionsnummer. Findet das Programm ein vorhandenes Paket mit derselben Versionsnummer, werden Sie aufgefordert, das vorhandene Paket zu deinstallieren.
- Ist das Paket installiert und führen Sie das Installationsprogramm für das Paket erneut unter Verwendung der grafischen Benutzerschnittstelle aus, können Sie auswählen, dass das Paket deinstalliert werden soll. Diese Deinstallationsoption ist im nicht überwachten Modus nicht verfügbar. Wenn Sie das Installationsprogramm für das Paket erneut im nicht überwachten Modus ausführen, wird das Programm ausgeführt, führt jedoch keine Aktionen aus.

Nur Archivierungspakete

- Wenn Sie das Installationsverzeichnis in einer Antwortdatei ändern und dann eine unbeaufsichtigte Installation unter Verwendung dieser Antwortdatei ausführen, ignoriert das Installationsprogramm das neue Installationsverzeichnis und verwendet stattdessen das Standardverzeichnis. Ist im Standardverzeichnis eine vorherige Installation vorhanden, wird sie überschrieben.

Pfad festlegen

Wenn Sie die Umgebungsvariable `PATH` festgelegt haben, können Sie eine Anwendung oder ein Programm durch die Eingabe des entsprechenden Namens an einer Shelleingabeaufforderung ausführen.

Informationen zu diesem Vorgang

Anmerkung: Alle vorhandenen ausführbaren Java-Programme in dem von Ihnen verwendeten Pfad werden überschrieben, wenn Sie die Umgebungsvariable `PATH` so wie in diesem Abschnitt beschrieben ändern.

Sie können den Pfad für ein Tool angeben, indem Sie den Pfad jedes Mal vor dem Toolnamen eingeben. Wenn z. B. das SDK in `/opt/IBM/javawrt3[_64]/` installiert ist, können Sie die Datei `meineDatei.java` kompilieren, indem Sie Folgendes an einer Shelleingabeaufforderung eingeben:

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


Gehen Sie wie folgt vor, um die jeweilige Eingabe des vollständigen Pfads zu vermeiden:

1. Bearbeiten Sie die Shellstartdatei im Ausgangsverzeichnis (in der Regel **.bashrc**, je nach der von Ihnen verwendeten Shell), und fügen Sie der Umgebungsvariablen **PATH** die absoluten Pfade hinzu. Beispiel:

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

2. Melden Sie sich erneut an, oder führen Sie das aktualisierte Shell-Script aus, um die neue Einstellung der Umgebungsvariablen **PATH** zu aktivieren.

3. Kompilieren Sie die Datei mit dem Tool **javac**. Geben Sie beispielsweise zum Kompilieren der Datei *MeineDatei.java* an einer Shelleingabeaufforderung Folgendes ein:

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

Über die Umgebungsvariable **PATH** kann Linux ausführbare Dateien wie **javac** und **java** sowie das Tool **javadoc** im aktuellen Verzeichnis finden. Geben Sie zum Anzeigen des aktuellen Werts Ihres Pfads Folgendes an einer Eingabeaufforderung ein:

```
echo $PATH
```

Nächste Schritte

Mithilfe der Informationen in „Klassenpfad festlegen“ können Sie ermitteln, ob Sie die Umgebungsvariable **CLASSPATH** festlegen müssen.

Klassenpfad festlegen

Die Umgebungsvariable **CLASSPATH** teilt den SDK-Tools (z. B. **java**, **javac** und **javadoc**) die Speicherposition der Java-Klassenbibliotheken mit.

Informationen zu diesem Vorgang

Legen Sie die Umgebungsvariable **CLASSPATH** nur explizit fest, wenn eine der folgenden Bedingungen zutrifft:

- Sie benötigen eine andere Bibliothek oder Klassendatei (ähnlich der, die Sie entwickeln), und diese befindet sich nicht im aktuellen Verzeichnis.
- Sie ändern die Speicherposition der Verzeichnisse `bin` und `lib`, und sie befinden sich nicht mehr im selben übergeordneten Verzeichnis.
- Sie wollen Anwendungen entwickeln oder ausführen, die unterschiedliche Laufzeitumgebungen auf demselben System verwenden.

Geben Sie Folgendes an einer Shelleingabeaufforderung ein, um den aktuellen Wert der Umgebungsvariable **CLASSPATH** anzuzeigen:

```
echo $CLASSPATH
```

Wenn Sie Anwendungen entwickeln und ausführen wollen, die unterschiedliche Laufzeitumgebungen verwenden, wie z. B. andere Versionen, die separat installiert wurden, müssen Sie **CLASSPATH** und **PATH** für jede Anwendung explizit festlegen. Wenn Sie mehrere Anwendungen gleichzeitig ausführen und unterschiedliche Laufzeitumgebungen verwenden, muss jede Anwendung in einem separaten Befehlsfenster in einer separaten Shell ausgeführt werden.

Wenn Sie nur jeweils eine Java-Version ausführen, können Sie mit einem Shell-Script zwischen den verschiedenen Laufzeitumgebungen umschalten.

Nächste Schritte

Mithilfe der Informationen in „Installation testen“ können Sie prüfen, ob Ihre Installation erfolgreich ist.

Installation testen

Mit der Option **-version** können Sie prüfen, ob Ihre Installation erfolgreich ist.

Informationen zu diesem Vorgang

Die Java-Installation besteht aus einer Echtzeit-JVM.

Vorgehensweise

Führen Sie die folgenden Schritte aus, um Ihre Installation zu testen:

1. Geben Sie den folgenden Befehl an einer Shelleingabeaufforderung ein, um Versionsinformationen für die Echtzeit-JVM anzuzeigen:

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

Dieser Befehl gibt bei erfolgreicher Ausführung die folgende Nachricht zurück:

```
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
```

Anmerkung: Diese Versionsinformationen sind korrekt, allerdings können die Plattformarchitektur und Daten vom Beispiel abweichen. Das Datumsformat ist JJMMTT möglicherweise gefolgt von zusätzlichen Informationen zur jeweiligen Komponente.

WebSphere Real Time for Linux deinstallieren

Welchen Prozess Sie zum Entfernen von WebSphere Real Time for Linux verwenden müssen, hängt davon ab, welcher Installationstyp verwendet wurde.

Vorbereitende Schritte

Bei installierbaren InstallAnywhere-Paketen ist eine Benutzer-ID mit Rootberechtigung erforderlich.

Informationen zu diesem Vorgang

Bei InstallAnywhere-Archivierungspaketen gibt es keinen Deinstallationsprozess. Wenn Sie ein Archivierungspaket von Ihrem System entfernen wollen, löschen Sie das beim Installieren des Pakets gewählte Zielverzeichnis. Bei installierbaren InstallAnywhere-Paketen deinstallieren Sie das Produkt mithilfe eines Befehls oder durch erneutes Ausführen des Installationsprogramms wie in den folgenden Schritten beschrieben.

Vorgehensweise

- Optional: Führen Sie die Deinstallation manuell mithilfe des Befehls **uninstall** durch.
 1. Wechseln Sie in das Verzeichnis, in dem sich die Installation befindet. Beispiel:

```
cd /opt/IBM/javawrt3
```
 2. Starten Sie den Deinstallationsprozess durch Eingabe des folgenden Befehls:

```
./_uninstall/uninstall
```
- Optional: Ist das Deinstallationsprogramm nicht leicht auffindbar, können Sie alternativ auch eine weitere beaufsichtigte Installation ausführen. Das Installationsprogramm erkennt das bereits installierte Produkt und gibt Ihnen die Gelegenheit, die vorherige Installation zu deinstallieren.

Kapitel 5. IBM WebSphere Real Time for Linux-Anwendungen ausführen

Wichtige Informationen für die Ausführung von Echtzeitanwendungen.

-
- „Metronom-Garbage-Collector verwenden“

Metronom-Garbage-Collector verwenden

Der Metronom-Garbage-Collector ersetzt den Standard-Garbage-Collector in WebSphere Real Time for Linux.

Pausezeit steuern

Die Pausezeit des Metronom-Garbage-Collectors (GC - Garbage Collector) kann für jeden Java-Prozess einzeln fein abgestimmt werden.

Der Metronom-GC hält standardmäßig während jeder einzelnen Pause für eine Zeitdauer von 3 Millisekunden an. Dies wird als Quantum bezeichnet. Ein vollständiger Garbage-Collection-Zyklus erfordert eine ganze Reihe dieser Pausen, die über den Zyklus verteilt werden, damit genügend Zeit für die Ausführung der Anwendung zur Verfügung steht. Mit der Option **-Xgc:targetPauseTime** kann der maximale Zeitwert für diese einzelnen Pausen geändert werden. Bei Angabe von **-Xgc:targetPauseTime=20** haben die einzelnen Pausen innerhalb der GC eine Zeitdauer von nicht mehr als 20 Millisekunden.

IBM Monitoring and Diagnostics Tools for Java - Garbage Collection and Memory Visualizer (GCMV) ermöglicht es Ihnen, die GC-Pausezeiten für Ihre Anwendung zu überwachen sowie unterstützt Sie, sollten in Ihrer Java-Anwendung Leistungsprobleme auftreten, bei der Diagnose und Optimierung. Mit dem Tool können die Daten aus verschiedenen Protokolltypen syntaktisch analysiert und geplottet werden. Zu diesen Protokolltypen zählen:

- Ausführliche Garbage-Collection-Protokolle
- Trace-Garbage-Collection-Protokolle (generiert mithilfe des Parameters **-Xtgc**)
- Protokolle für native Speicher (generiert mithilfe der Systembefehle **ps**, **svmon** oder **perfmon**)

Die Diagramme in diesem Abschnitt wurden von GCMV erstellt. Sie veranschaulichen, wie sich eine Änderung der Zielpausezeit jeweils auf die Garbage-Collection-Zyklen auswirkt. In jedem der geplotteten Diagramme sind die tatsächlichen Pausenzeiten zwischen den Metronom-Garbage-Collection-Zyklen (Y-Achse) im Verhältnis zur Ausführungszeit einer Anwendung (X-Achse) dargestellt.

Anmerkung: GCMV unterstützt ein älteres ausführliches Garbage-Collection-Format. Wenn Sie die ausführliche GC-Ausgabe mit GCMV analysieren wollen, verwenden Sie beim Generieren der Ausgabe die Option **-Xgc:verboseFormat=deprecated**. Weitere Informationen finden Sie in GC-Befehlszeilenoptionen.

Ist die Zielpausezeit auf den Standardwert gesetzt, bewegen sich die Pausezeiten um die 3-Millisekunden-Marke bzw. bleiben darunter, wie im Diagramm mit den Pausezeiten einer ausführlichen GC zu erkennen ist:

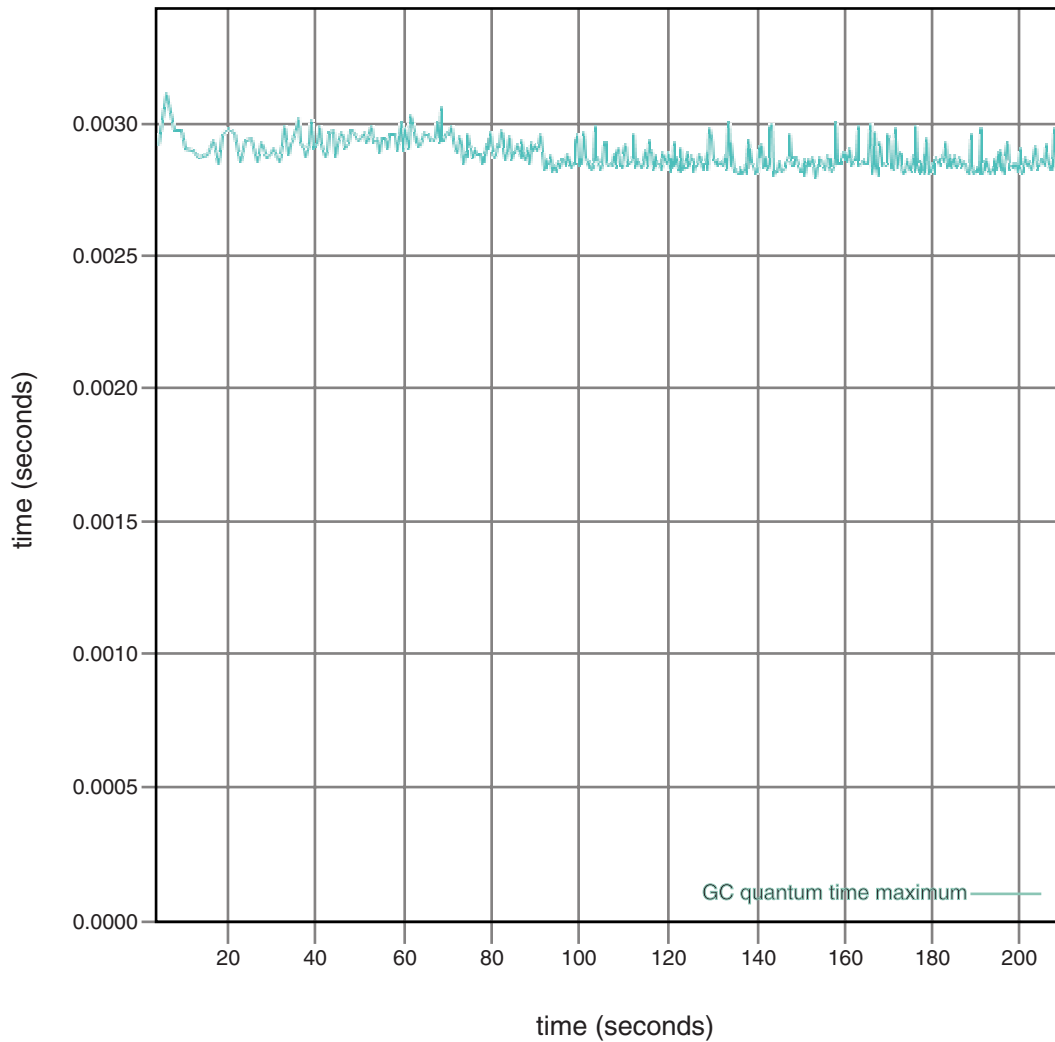


Abbildung 1. Tatsächliche Garbage-Collection-Pausezeiten, wenn für die Zielpausezeit der Standardwert (3 Millisekunden) festgelegt wurde

Wurde für die Zielpausezeit eine Zeitdauer von 6 Millisekunden festgelegt, bewegen sich die Pausezeiten um 6-Millisekunden-Marke bzw. bleiben darunter, wie im Diagramm mit den Pausezeiten für eine ausführliche GC zu erkennen ist:

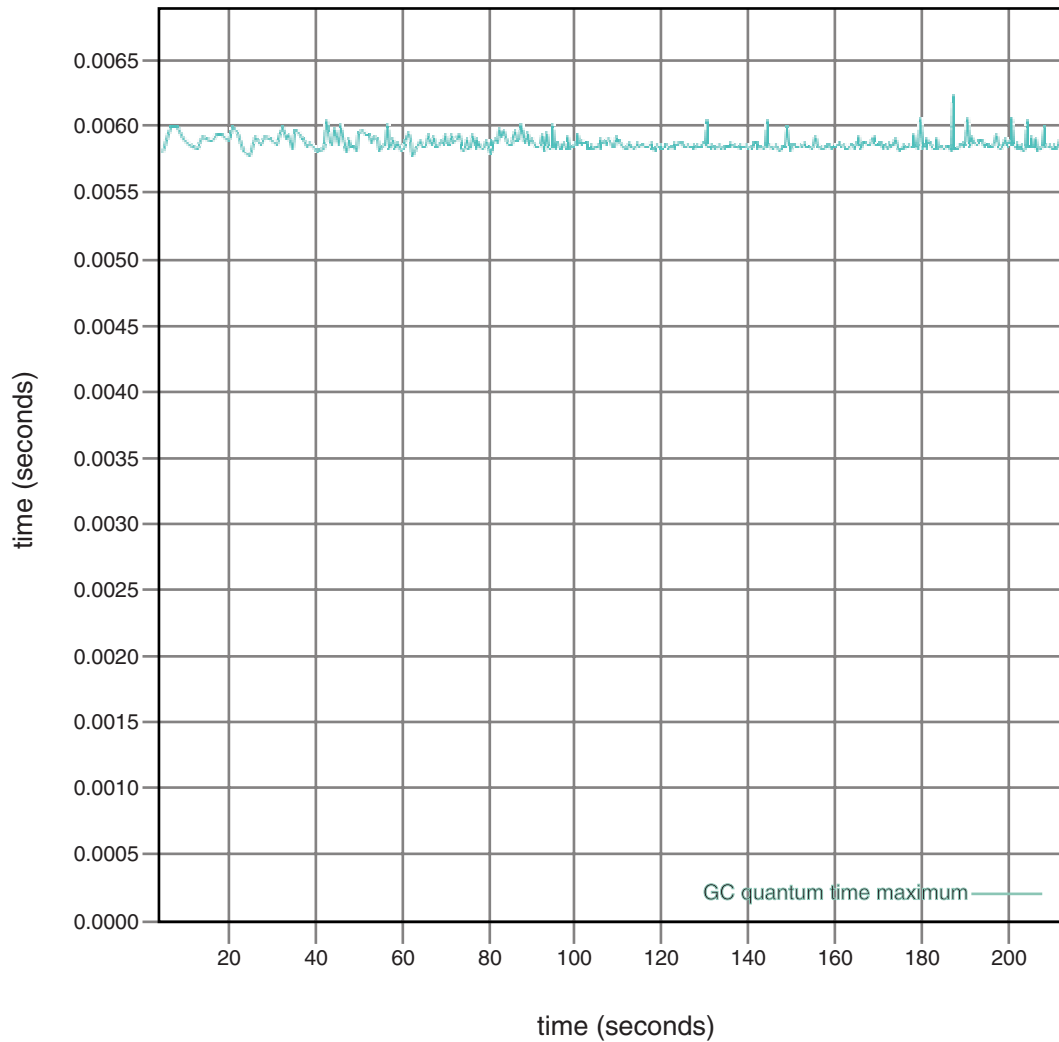


Abbildung 2. Tatsächliche Pausezeiten, wenn für die Zielpausezeit 6 Millisekunden festgelegt wurden

Wurde für die Zielpausezeit eine Zeitdauer von 10 Millisekunden festgelegt, bewegen sich die Pausezeiten um 10-Millisekunden-Marke bzw. bleiben darunter, wie im Diagramm mit den Pausezeiten für eine ausführliche GC zu erkennen ist:

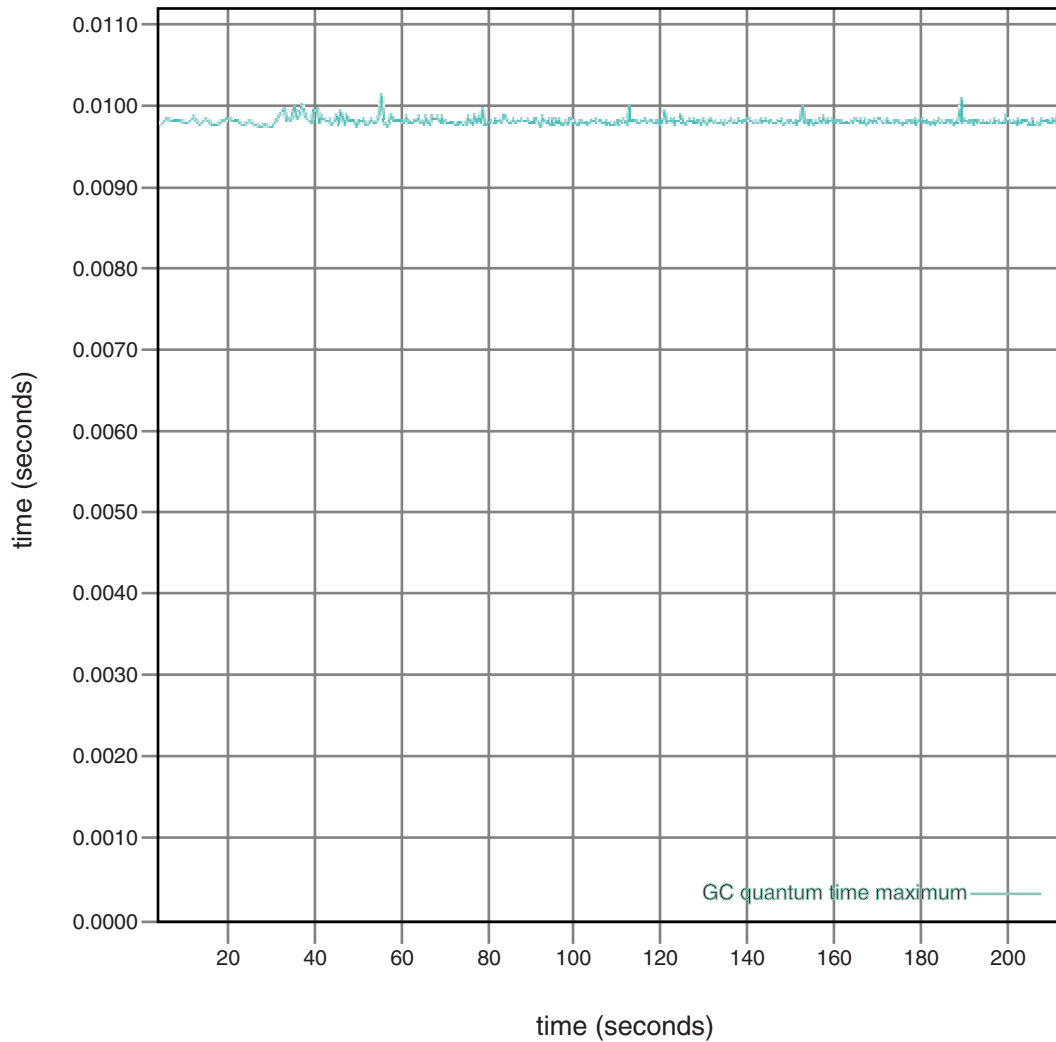


Abbildung 3. Tatsächliche Pausezeiten, wenn für die Zielpausezeit 10 Millisekunden festgelegt wurden

Wurde für die Zielpausezeit eine Zeitdauer von 15 Millisekunden festgelegt, bewegen sich die Pausezeiten um 15-Millisekunden-Marke bzw. bleiben darunter, wie im Diagramm mit den Pausezeiten für eine ausführliche GC zu erkennen ist:

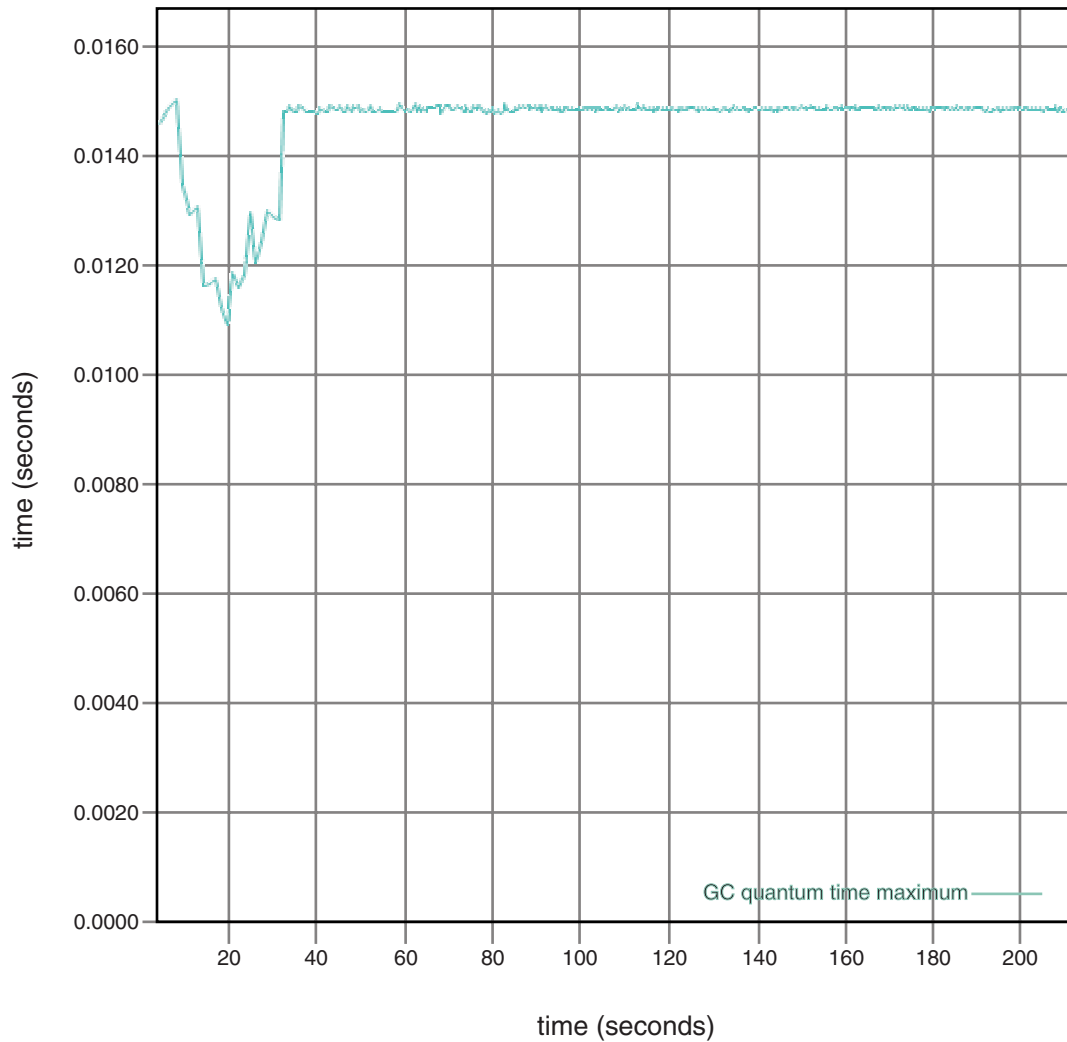


Abbildung 4. Tatsächliche Pausezeiten, wenn für die Zielpausezeit 15 Millisekunden festgelegt wurden

Prozessorauslastung steuern

Sie können die Menge der dem Metronom-Garbage-Collector zur Verfügung stehenden Verarbeitungskapazität begrenzen.

Sie können die Garbage-Collection mit dem Metronom-Garbage-Collector unter Verwendung der Option `-Xgc:targetUtilization=N` steuern. Mit dieser Option können Sie die vom Garbage-Collector verwendete CPU-Kapazität begrenzen.

Beispiel:

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

Das Beispiel gibt an, dass Ihre Anwendung in jedem 60-Millisekunden-Intervall 80 % der Zeit belegt. Die verbleibenden 20 % der Zeit werden für die Garbage-Collection verwendet. Der Metronom-Garbage-Collector garantiert bestimmte Auslastungen, vorausgesetzt ihm wurden ausreichende Ressourcen zugeteilt. Die Garbage-Collection fängt an, wenn der freie Speicherplatz im Heapspeicher unter einen dynamisch ermittelten Schwellenwert fällt.

Einschränkung für den Metronom-Garbage-Collector

In seltenen Fällen kann es während der Garbage-Collection zu längeren Pausen als erwartet kommen.

Während der Garbage-Collection wird ein Root-Scan-Vorgang verwendet. Der Garbage-Collector führt für den Heapspeicher eine Walk-Operation durch und startet mit bekannten zeitnahen Verweisen. Diese Verweise können folgender Art sein:

- Zeitnahe Verweisvariablen in den aktiven Thread-Aufruf-Stacks
- Statische Verweise

Der Garbage-Collector durchsucht alle Stack-Frames im Aufrufstack des Anwendungsthreads, um alle Verweise auf Liveobjekte im Stack dieses Threads zu finden. Jeder aktive Thread-Stack wird in einem unterbrechungsfreien Schritt durchsucht. Daher muss der Suchlauf innerhalb einer einzelnen GC-Pause stattfinden.

Die Systemleistung ist aufgrund von erweiterten Garbage-Collection-Pausen am Anfang eines Erfassungszyklus möglicherweise schlechter als erwartet, wenn einige Threads mit sehr tiefen Stacks vorliegen.

Kapitel 6. Anwendungen entwickeln

Wichtige Informationen zum Schreiben von Echtzeitanwendungen, einschließlich Codemuster.

- „Echtzeitorientierte Musterhashzuordnung“

Echtzeitorientierte Musterhashzuordnung

WebSphere Real Time for Linux enthält HashMap- und HashSet-Implementierungen, die eine konsistentere Leistung für die Methode put als die HashMap-Standardimplementierung in IBM SDK for Java 7 liefern.

Der von IBM gelieferte Standard `java.util.HashMap` eignet sich gut für Anwendungen mit hohem Durchsatz. Er eignet sich auch für Anwendungen, bei denen die maximale Größe der Hashzuordnung erhöht werden muss. Bei Anwendungen, die eine Hashzuordnung benötigen, die nutzungsabhängig verschiedentlich groß sein können, gibt es ein potenzielles Leistungsproblem mit der Standardhashzuordnung. Die Standardhashzuordnung bietet gute Antwortzeiten beim Hinzufügen von neuen Einträgen zur Hashzuordnung mit der Methode `put`. Wenn diese Hashzuordnung jedoch aufgefüllt ist, muss ein größerer Sicherungsspeicher zugeordnet werden. Dies bedeutet, dass die Einträge im aktuellen Sicherungsspeicher migriert werden müssen. Wenn die Hashzuordnung groß ist, kann die Ausführung einer `put`-Operation auch lange dauern. Beispielsweise kann die Operation mehrere Millisekunden dauern.

WebSphere Real Time for Linux enthält eine echtzeitorientierte Musterhashzuordnung. Sie bietet dieselbe Funktionsschnittstelle wie der Standard `java.util.HashMap`, ermöglicht jedoch konsistentere Leistung für die Methode `put`. Die Musterhashzuordnung erstellt einen zusätzlichen Sicherungsspeicher, anstatt nur einen Sicherungsspeicher zu erstellen und alle Einträge zu migrieren, wenn die Hashzuordnung aufgefüllt ist. Der neue Sicherungsspeicher ist mit den anderen Sicherungsspeichern in der Hashzuordnung verkettet. Die Verkettung verursacht anfänglich eine leichte Leistungssenkung, während der leere Sicherungsspeicher zugeordnet und mit den anderen Sicherungsspeichern verkettet wird. Nachdem die Sicherungshashzuordnung aktualisiert wurde, ist sie schneller als die Migration aller Einträge. Ein Nachteil der echtzeitorientierten Hashzuordnung ist, dass die `get`-, `put`- und `remove`-Operationen etwas langsamer ausgeführt werden. Die Operationen sind langsamer, weil jede Suche eine Gruppe von Sicherungshashzuordnungen anstatt nur einer Sicherungshashzuordnung durcharbeiten muss.

Fügen Sie am Anfang Ihres Bootklassenpfads die Datei `RTHashMap.jar` hinzu, um die echtzeitorientierte Hashzuordnung auszuprobieren. Wenn Sie WebSphere Real Time for Linux im Verzeichnis `$WRT_ROOT` installiert haben, fügen Sie die folgende Option hinzu, um die echtzeitorientierte Hashzuordnung anstatt der Standardhashzuordnung mit Ihrer Anwendung zu verwenden:

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

Die Quellen- und Klassendateien für die Implementierung der echtzeitorientierten Hashzuordnung sind in die Datei `demo/realtime/RTHashMap.jar` eingeschlossen. Außerdem werden die Echtzeitimplementierungen `java.util.LinkedHashMap` und `java.util.HashSet` bereitgestellt.

Kapitel 7. Leistung

WebSphere Real Time for Linux ist für konsistent kurze GC-Pausen anstatt für die höchste Durchsatzleistung oder den kleinsten Speicherbedarf optimiert.

Leistung bei zertifizierten Hardwarekonfigurationen

Zertifizierte Systeme haben eine ausreichende Taktgebergranularität und Prozessorgeschwindigkeit, um die WebSphere Real Time for Linux-Leistungsziele zu unterstützen. Beispielsweise würde es für eine gut geschriebene Anwendung, die auf einem nicht überladenen System und mit einer adäquaten Größe des Heapspeichers normalerweise GC-Pausezeiten geben, die ca. 3 Millisekunden, allerdings nicht mehr als 3,2 Millisekunden betragen. Im Verlauf von GC-Zyklen wird eine Anwendung mit Standardumgebungeinstellungen nicht länger als 30 % der abgelaufenen Zeit während eines 60 Millisekunden langen gleitenden Fensters angehalten. Die in GC-Pausen im Verlauf eines 60 Millisekunden langen Zeitraums verbrachte Gesamtzeit beträgt normalerweise ca. 18 Millisekunden.

Ablaufsteuerungsvariabilität reduzieren

Die Hauptquellen von Variabilität in einer Standard-JVM sind Garbage-Collection-Pausen. In WebSphere Real Time for Linux werden die möglicherweise langen Pausen im Vergleich zu den Garbage-Collector-Standardmodi durch die Verwendung des Metronom-Garbage-Collectors vermieden. Weitere Informationen hierzu finden Sie in „Metronom-Garbage-Collector verwenden“ auf Seite 17.

Gemeinsame Nutzung von Klassendaten zwischen JVMs

Die gemeinsame Nutzung von Klassendaten stellt eine transparente Methode zur Verringerung des Speicherbedarfs und zur Verbesserung der JVM-Startzeit bereit. Weitere Informationen zur gemeinsamen Nutzung von Klassendaten finden Sie in „Gemeinsame Nutzung von Klassendaten auf verschiedenen JVMs“ auf Seite 26.

Komprimierte Verweise

Die Metronom-Garbage-Collection unterstützt sowohl komprimierte als auch nicht komprimierte Verweise auf 64-Bit-Plattformen. Bei der Verwendung von komprimierten Verweisen speichert die JVM alle Verweise auf Objekte, Klassen und Threads und überwacht sie als 32-Bit-Werte. Die Verwendung von komprimierten Verweisen verbessert die Leistung vieler Anwendungen, da Objekte kleiner sind, was zu weniger häufigen Garbage-Collections und zu einer verbesserten Auslastung des Hauptspeichercaches führt.

Anmerkung: Die für komprimierte Verweise verfügbare Größe des Heapspeichers ist auf ca. 28 GB beschränkt.

Weitere Informationen zu komprimierten Verweisen finden Sie in http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.doc.70/diag/understanding/mm_compressed_references.html.

Gemeinsame Nutzung von Klassendaten auf verschiedenen JVMs

Für die Unterstützung gemeinsam genutzter Klassen macht es keinen Unterschied, ob die Option **-Xrealtime** verwendet wird oder nicht.

Sie können Klassendaten zwischen Java Virtual Machines (JVMs) gemeinsam nutzen, indem Sie sie in einer im Speicher abgelegten Cachedatei speichern. Durch die gemeinsame Nutzung verringert sich die gesamte virtuelle Speicherbelegung, wenn mehrere JVMs einen Cache gemeinsam nutzen. Außerdem verkürzt sich durch die gemeinsame Nutzung der Systemstart von JVM, nachdem der Cache erstellt wurde. Der Cache für gemeinsam genutzte Klassen ist unabhängig von aktiven JVMs und bleibt persistent, bis er gelöscht wird.

Ein gemeinsam genutzter Cache kann Folgendes enthalten:

- Bootprogrammklassen
- Anwendungsklassen
- Metadaten, die die Klassen beschreiben
- Kompilierter AOT-Code (Ahead-of-time)

Kapitel 8. Sicherheit

Dieser Abschnitt enthält wichtige Informationen zur Sicherheit.

Sicherheitsaspekte für den Cache für gemeinsam genutzte Klassen

Der Cache für gemeinsam genutzte Klassen wurde entwickelt, um die Cacheverwaltung und die Nutzbarkeit zu erleichtern. Die Standardsicherheitsrichtlinie ist hierbei jedoch möglicherweise nicht geeignet.

Wenn Sie den Cache für gemeinsam genutzte Klassen verwenden, müssen Sie die Standardberechtigungen für neue Dateien beachten, um die Sicherheit durch das Einschränken des Zugriffs verbessern zu können.

Datei	Standardberechtigungen
neue gemeinsam genutzte Caches	Leseberechtigungen für Gruppe und andere
Verzeichnis javasharedresources	Allgemeine Lese-, Schreib- und Ausführungsberechtigungen

Sie benötigen Schreibberechtigung für die Cachedatei und für das Cacheverzeichnis, um einen Cache zu löschen oder weiter zu füllen.

Dateiberechtigungen für die Cachedatei ändern

Sie können den Zugriff auf einen Cache für gemeinsam genutzte Klassen mit dem Befehl **chmod** einschränken.

Erforderliche Änderung	Befehl
Zugriff auf den Benutzer und die Gruppe einschränken	<code>chmod 770 /tmp/javasharedresources</code>
Zugriff auf den Benutzer einschränken	<code>chmod 700 /tmp/javasharedresources</code>
Benutzer auf Lese- und Schreibzugriff für einen bestimmten Cache einschränken	<code>chmod 600 /tmp/javasharedresources/ <Datei für gemeinsam genutzten Cache></code>
Benutzer und Gruppe auf Lese- und Schreibzugriff für einen bestimmten Cache einschränken	<code>chmod 660 /tmp/javasharedresources/ <Datei für gemeinsam genutzten Cache></code>

Verbindung zu einem Cache herstellen, für den Sie keine Zugriffsberechtigung haben

Wenn Sie versuchen, eine Verbindung zu einem Cache herzustellen, für den Sie nicht die entsprechenden Zugriffsberechtigungen haben, wird eine Fehlermeldung angezeigt:

```
JVMShrc226E Fehler beim Öffnen der Cachedatei für eine gemeinsam genutzte Klasse
JVMShrc220E Fehlercode für Portschnittstelle = -302
JVMShrc221E Fehlermeldung für Plattform: Zugriff verweigert
JVMJ9VM015W Initialisierungsfehler für Bibliothek j9shr25(11):
JVMJ9VM009E J9VMD11Main fehlgeschlagen
Java Virtual Machine konnte nicht erstellt werden
```

Kapitel 9. Fehlerbehebung und Unterstützung

Fehlerbehebung und Unterstützung für WebSphere Real Time for Linux

- „Allgemeine Problembestimmungsmethoden“
- „Fehlerbehebung von OutOfMemory-Fehlern“ auf Seite 36
- „Diagnosetools verwenden“ auf Seite 39

Allgemeine Problembestimmungsmethoden

Mithilfe der Fehlerbestimmung können Sie feststellen, welche Art von Fehler vorliegt und wie Sie am besten vorgehen.

Wenn Sie wissen, welche Art von Fehler vorliegt, können Sie eine oder mehrere der folgenden Tasks ausführen:

- Fehler beheben
- Passende Fehlerumgehung finden
- Erforderliche Daten für die Generierung eines Fehlerberichts für IBM erfassen

Fehlerbestimmung unter Linux

In diesem Abschnitt wird die Fehlerbestimmung unter Linux beschrieben.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zum Diagnostizieren von Problemen unter Linux:

- Linux-Umgebung einrichten und prüfen
- Allgemeine Debugging-Verfahren
- Abstürze diagnostizieren
- Debugging von Blockierungen
- Debugging von Speicherlecks
- Debugging von Leistungsproblemen

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Linux-Fehlerbestimmung.

Die folgenden Informationen sind für IBM WebSphere Real Time for Linux ergänzend.

Linux-Umgebung einrichten und prüfen

Prüfen Sie in IBM WebSphere Real Time for Linux, ob die JVM ordnungsgemäß für das Generieren eines Systemspeicherauszugs konfiguriert ist.

Linux-Systemspeicherauszüge (Kerndateien)

Bei einem Absturz ist der Linux-Systemspeicherauszug (Kerndatei) die wichtigste Quelle für Diagnosedaten. Wenn Sie sicherstellen möchten, dass diese Datei generiert wird, müssen Sie Ihre Betriebssystemeinstellungen und den verfügbaren Plattenspeicher so wie im Benutzerhandbuch für IBM SDK for Java 7 beschrieben prüfen.

Java Virtual Machine-Einstellungen

Die JVM muss so konfiguriert sein, dass sie im Fall eines Absturzes Kern-

dateien generiert. Führen Sie `java -Xdump:what` in der Befehlszeile aus. Die Ausgabe dieser Option sieht wie folgt aus:

```
-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
```

Bei den gezeigten Werten handelt es sich um die Standardeinstellungen. Es muss mindestens die Option `events=gpf` eingestellt sein, damit bei einem Absturz eine Kerndatei generiert wird. Mit der Befehlszeilenoption `-Xdump:system[:name1=value1,name2=value2 ...]` können Sie Optionen ändern und festlegen.

Allgemeine Debugging-Verfahren

Da Java-Threadnamen im Betriebssystem angezeigt werden, können Sie den Befehl `ps` beim Debugging einsetzen. Wenn Sie Traceerstellungstools verwenden, müssen Sie die richtigen Befehle für IBM WebSphere Real Time for Linux verwenden.

Prozessinformationen prüfen

Bei der Ausführung des Befehls `ps` in IBM WebSphere Real Time for Linux wird die folgende Ausgabe generiert:

```
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
```

U Wählt alle Prozesse aus.

L Zeigt Threads an.

o Gibt ein vordefiniertes Format für Spalten an, die angezeigt werden sollen. Bei den angegebenen Spalten handelt es sich um die Prozess-ID, die Thread-ID, die Planungsrichtlinie, die Echtzeitthreadpriorität und den Befehl, der dem Prozess zugeordnet ist. Diese Informationen sind hilfreich, um festzustellen, welche virtuelle Maschine und welche Threads in Ihrer Anwendung zu einem bestimmten Zeitpunkt aktiv sind.

Traceerstellungstools

Unter Linux stehen die drei Traceerstellungstools `strace`, `ltrace` und `mtrace` zur Verfügung. Mit dem Befehl `man strace` werden eine ganze Reihe verfügbarer Optionen angezeigt.

strace

Das Tool 'strace' verfolgt Systemaufrufe. Sie können es für einen Prozess ver-

wenden, der bereits verfügbar ist, oder mit einem neuen Prozess starten. 'strace' erfasst die Systemaufrufe, die von einem Programm ausgeführt werden, und die Signale, die von einem Prozess empfangen werden. Für jeden Systemaufruf werden der Name, Argumente und der Rückgabewert verwendet. Mit 'strace' können Sie ein Programm ohne die Quelle verfolgen (es ist keine Neukompilierung erforderlich). Wenn Sie 'strace' mit der Option `-f` verwenden, werden untergeordnete Prozesse verfolgt, die als Ergebnis eines verzweigten Systemaufrufs erstellt wurden. Mit 'strace' können Sie Plug-in-Probleme untersuchen oder versuchen zu verstehen, warum Programme nicht ordnungsgemäß gestartet werden.

Um 'strace' mit einer Java-Anwendung zu verwenden, geben Sie `strace java -Xgcpolicy:metronome <class-name>` ein.

Sie können die Traceausgabe vom Tool 'strace' in eine Datei übertragen, indem Sie die Option `-o` verwenden.

ltrace

Das Tool 'ltrace' ist variantenabhängig. Es ist dem Tool 'strace' sehr ähnlich. Dieses Tool fängt die Aufrufe der dynamischen Bibliothek durch den ausführenden Prozess ab und zeichnet sie auf. 'strace' führt für die Signale, die vom ausführenden Prozess empfangen werden, dieselbe Aktion aus.

Um 'ltrace' mit einer Java-Anwendung zu verwenden, geben Sie `ltrace java -Xgcpolicy:metronome <class-name>` ein.

mtrace

'mtrace' ist im GNU-Toolset enthalten. Es installiert spezielle Steuerroutinen für 'malloc', 'realloc' und 'free' und macht es möglich, dass alle Verwendungen dieser Funktionen verfolgt und in einer Datei erfasst werden. Durch diese Traceerstellung wird die Programmeffizienz verringert. Sie sollte deshalb nicht während der normalen Verwendung aktiviert werden. Um 'mtrace' verwenden zu können, setzen Sie `IBM_MALLOCTRACE` auf 1 und legen Sie `MALLOC_TRACE` so fest, dass auf eine gültige Datei verwiesen wird, in der die Traceinformationen gespeichert werden. Sie müssen über Schreibzugriff auf diese Datei verfügen.

Um 'mtrace' mit einer Java-Anwendung zu verwenden, geben Sie Folgendes ein:

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

Abstürze diagnostizieren

Befolgen Sie beim Zusammenstellen von Informationen zur Ausführung von Prozessen und der Java-Umgebung vor einem Absturz diese Richtlinien.

Prozessinformationen zusammenstellen

Zeigen Sie beim Erforschen der Umstände vor dem Absturz mithilfe der Befehle `gdb` und `bt` den Stack-Trace des fehlgeschlagenen Threads an, anstatt die Kerndatei zu analysieren.

Informationen zur Java-Umgebung ermitteln

Mithilfe des Java-Speicherauszugs können Sie feststellen, welche Operationen die einzelnen Threads ausgeführt haben und welche Java-Methoden ausgeführt wur-

den. Gleichen Sie Funktionsadressen mit Bibliotheksadressen ab, um die Quelle des Codes zu ermitteln, der an verschiedenen Punkten ausgeführt wird.

Überprüfen Sie mithilfe der Option **-verbose:gc** den Status des Java-Heapspeichers. Stellen Sie die folgenden Fragen:

- Gab es einen Speicherengpass in einem der Hauptspeicherbereiche, der möglicherweise den Absturz verursacht hat?
- Ist der Absturz während der Garbage-Collection aufgetreten? Dies deutet auf einen möglichen Garbage-Collection-Fehler hin.
- Ist der Absturz nach der Garbage-Collection aufgetreten? Dies deutet auf einen möglichen Datenverlust im Hauptspeicher hin.

Debugging von Leistungsproblemen

Beachten Sie beim Debugging von Leistungsproblemen die folgenden Punkte für IBM WebSphere Real Time for Linux zusätzlich zu den Themen im Benutzerhandbuch für IBM SDK for Java 7.

Größe von Hauptspeicherbereichen ändern

Die Größe des Java-Heapspeichers ist einer der wichtigsten Optimierungsparameter der JVM. Wählen Sie die richtige Größe, um die Leistung zu optimieren. Wenn Sie die richtige Größe verwenden, kann der Garbage-Collector leichter die erforderliche Auslastung bereitstellen.

Weitere Informationen zum Ändern der Größe von Hauptspeicherbereichen finden Sie in „Fehlerbehebung für den Metronom-Garbage-Collector“ auf Seite 60.

JIT-Kompilierung und Leistung

Bei der Verwendung von JIT sollten Sie die Auswirkungen auf das Echtzeitverhalten berücksichtigen.

Bekannte Einschränkungen unter Linux

Bei Linux hat eine schnelle Entwicklung stattgefunden und es gab verschiedene Probleme bei der Interaktion der JVM und des Betriebssystems, insbesondere im Bereich der Threads.

Beachten Sie die folgenden Einschränkungen, die sich möglicherweise auf Ihr Linux-System auswirken.

Threads als Prozesse

Wenn die Anzahl der Java-Threads die maximal zulässige Anzahl an Prozessen überschreitet, hat dies möglicherweise die folgenden Folgen für Ihr Programm:

- Es erhält eine Fehlermeldung.
- Es erhält einen **SIGSEGV**-Fehler.
- Es wird gestoppt.

Weitere Informationen finden Sie in *The Volano Report* auf der Website <http://www.volano.com/report/index.html>.

Einschränkungen für variabel verknüpfte Stacks

Bei einer Ausführung ohne variabel verknüpfte Stacks wird unabhängig von der Einstellung für **-Xss** eine Minimalgröße von 256 KB für native Stacks für die einzelnen Threads bereitgestellt.

Auf einem Linux-System mit variabel verknüpften Stacks werden die Werte für **-Xss** verwendet. Wenn Sie ein Linux-System ohne variabel verknüpfte Stacks migrieren, müssen alle Werte für **-Xss** groß genug sein und es darf kein Minimum von 256 KB erforderlich sein.

glibc-Einschränkungen

Wenn Sie eine Nachricht erhalten, die angibt, dass die Bibliothek `libjava.so` nicht geladen werden konnte, da ein Symbol nicht gefunden wurde (z. B. `__bzero`), ist möglicherweise eine frühere Version der GNU C-Laufzeitbibliothek, `glibc`, installiert. Das SDK für die Linux-Threadimplementierung erfordert `glibc` Version 2.3.2 oder höher.

Einschränkungen für Schriftarten

Wenn Sie die Installation auf einem Red Hat-System durchführen, führen Sie (beispielsweise unter Linux IA32) Folgendes aus, damit der Schriftartenserver die Java-TrueType-Schriftarten finden kann:

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

Sie müssen dies während der Installation ausführen und Sie müssen als „Root“ angemeldet sein, um den Befehl ausführen zu können. Weitere Informationen zu Problemen mit Schriftarten finden Sie im SDK- und Runtime Environment-Benutzerhandbuch für Linux.

Beeinträchtigung der Leistung von Java durch Linux Completely Fair Scheduler

Die Leistung von Java-Anwendungen mit intensivem Verwenden von Synchronisierungsprozessen kann in Linux-Varianten, die das Tool Completely Fair Scheduler umfassen, stark beeinträchtigt sein. Completely Fair Scheduler (CFS) ist ein Scheduler, der ab Version 2.6.23 in den Haupt-Kernel von Linux aufgenommen wurde. CFS-Algorithmus unterscheidet sich von den Scheduler-Algorithmen früherer Linux-Versionen. Er kann die Leistung einiger Anwendungen beeinträchtigen. Insbesondere implementiert CFS die Methode `sched_yield()` auf andere Weise, wodurch die Wahrscheinlichkeit steigt, dass ein Yield-Thread, trotz des Verzichts auf sein Ausführungsvorrecht durch Aufruf der Yield-Methode, CPU-Zeit zugeteilt bekommt.

Wenn dieses Problem auftritt, ist es möglich, dass Sie zwar eine hohe CPU-Auslastung durch die Java-Anwendung, aber gleichzeitig langsamen Fortschritt bei synchronisierten Blöcken beobachten. Aufgrund der Langsamkeit könnten Sie sogar den Eindruck erhalten, dass die Anwendung gestoppt wurde.

Es gibt zwei mögliche Fehlerumgehungen:

- Starten Sie die JVM mit dem zusätzlichen Argument **-Xthr:minimizeUserCPU**.

- Konfigurieren Sie den Linux-Kernel so, dass er eine Implementierung von `sched_yield()` verwendet, die eine größere Kompatibilität mit früheren Versionen aufweist. Setzen Sie dazu die Eigenschaft `sched_compat_yield` des optimierbaren Kernels auf **1**. Beispiel:

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

Wenden Sie diese Fehlerumgehungen nur an, wenn Sie Leistungseinbußen feststellen.

Dieses Problem kann sich auf das IBM Developer Kit und die Runtime Environment für Linux 5.0 (alle Versionen) und 6.0 (alle Versionen bis einschließlich SR 4) auswirken, die in Linux-Kernels mit Completely Fair Scheduler ausgeführt werden. Für das IBM Developer Kit und die Runtime Environment für Linux Version 6.0 nach SR 4 wird das Verwenden von CFS im Kernel erkannt und die Option **-Xthr:minimizeUserCPU** automatisch aktiviert. Beispiele für Linux-Varianten, die das Tool Completely Fair Scheduler umfassen sind Ubuntu 8.04 und SUSE Linux Enterprise Server 11.

Weitere Informationen zu CFS finden Sie in *Multiprocessing with the Completely Fair Scheduler* (Ausführen mehrerer Prozesse mit Completely Fair Scheduler).

Leistungsprobleme bei Linux Red Hat MRG-Kernen

Ein Konfigurationsproblem mit Red Hat MRG-Kernen kann unerwartete Pausen bei Anwendungsthreads verursachen, wenn WebSphere Real Time mit aktivierter ausführlicher Garbage-Collection gestartet wird. Diese Pausen werden nicht in der Ausgabe der ausführlichen GC dokumentiert, Sie können jedoch je nach Netzkonfiguration mehrere Millisekunden dauern. JVMs, die von fern definierten LDAP-Benutzern gestartet werden, sind am meisten betroffen, da der Cachedämon des Namensservice (`nscd`) nicht gestartet wird, was zu Verzögerungen bei der Netzübertragung führt. Beheben Sie das Problem, indem Sie `nscd` starten. Führen Sie folgende Schritte aus, um den Status des Service `nscd` zu überprüfen und das Problem zu beheben:

1. Überprüfen Sie, ob der Dämon `nscd` aktiv ist, indem Sie den folgenden Befehl eingeben:

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

Wenn der Dämon nicht aktiv ist, wird folgende Nachricht angezeigt:

```
nscd is stopped
```

2. Starten Sie den Service `nscd` als Rootbenutzer mit dem folgenden Befehl:

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

3. Ändern Sie die Startinformationen für den Service `nscd` als Rootbenutzer mit dem folgenden Befehl:

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

Der Prozess `nscd` ist jetzt aktiv und wird nach einem Warmstart automatisch gestartet.

Fehlerbestimmung für NLS

Die JVM enthält integrierte Unterstützung für verschiedene Ländereinstellungen.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zum Diagnostizieren von NLS-Problemen:

- Schriftarten - Übersicht

- Dienstprogramme für Schriftarten
- Häufig auftretende NLS-Probleme und mögliche Ursachen

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - NLS-Problembestimmung.

Fehlerbestimmung für ORB

Eine der ersten Aufgaben beim Debugging eines ORB-Fehlers besteht darin zu bestimmen, ob der Fehler bei der verteilten Anwendung clientseitig oder serverseitig auftritt. Stellen Sie sich eine typische RMI-IIOP-Sitzung als eine einfache synchrone Übertragung zwischen einem Client, der den Zugriff auf ein Objekt anfordert, und einem Server, der ihn gewährt, vor.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zum Diagnostizieren von ORB-Problemen:

- ORB-Fehler bestimmen
- Stack-Trace interpretieren
- ORB-Traces interpretieren
- Häufig auftretende Probleme
- IBM ORB-Service: Daten erfassen

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - ORB-Problembestimmung.

Die folgenden Informationen sind für IBM WebSphere Real Time for Linux ergänzend.

IBM ORB-Service: Daten erfassen

Führen Sie beim Erfassen der Java-Versionsausgabe für den Service den folgenden Befehl aus:

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

Vortests

Wenn ein Problem auftritt, generiert ORB möglicherweise eine Ausnahmebedingung `org.omg.CORBA.*`, die Folgendes einschließt:

- Text zur Angabe der Ursache
- Nebencode
- Fertigstellungsstatus

Bevor Sie davon ausgehen, dass die Fehlerursache bei ORB liegt, prüfen Sie Folgendes:

- Das Szenario kann in ähnlicher Konfiguration reproduziert werden.
- Der JIT-Compiler ist inaktiviert.
- Es wird kein AOT-kompilierter Code verwendet

Weitere Aktionen:

- Inaktivieren Sie zusätzliche Prozessoren.
- Inaktivieren Sie simultanes Multithreading (SMT), wo dies möglich ist.
- Beseitigen Sie Speicherabhängigkeiten bei Client oder Server. Der Mangel an physischem Hauptspeicher kann die Ursache langsamer Verarbeitung, scheinba-

rer Blockierungen oder von Abstürzen sein. Stellen Sie sicher, dass Sie über ein angemessenes Speichervolumen verfügen, um diese Probleme zu lösen.

- Überprüfen Sie Fehler im physischen Netz wie Firewalls, Übertragungsleitungen, Router und DNS-Namensserver. Dies sind die Hauptursachen für die Ausnahmebedingung `COMM_FAILURE` in CORBA. Überprüfen Sie als Test den Namen Ihrer eigenen Workstation mit Ping.
- Wenn die Anwendung eine Datenbank wie DB2 verwendet, wechseln Sie zum zuverlässigsten Treiber. Um zum Beispiel den DB2 AppDriver zu isolieren, wechseln Sie zu 'Net Driver', der zwar langsamer ist und Sockets verwendet, dafür aber zuverlässiger ist.

Fehlerbehebung von OutOfMemory-Fehlern

Handhabung von Ausnahmebedingungen `OutOfMemoryError`.

Allgemeine Fehlerbehebungsinformationen zum Metronom-Garbage-Collector finden Sie in „Fehlerbehebung für den Metronom-Garbage-Collector“ auf Seite 60.

Fehler aufgrund abnormaler Speicherbedingungen (`OutOfMemoryErrors`) diagnostizieren

Das Diagnostizieren von Ausnahmebedingungen `OutOfMemoryError` im Metronom-Garbage-Collector kann aufgrund des periodischen Charakters des Garbage-Collectors komplexer sein als in einer Standard-JVM.

Im Allgemeinen benötigt eine Echtzeitanwendung ungefähr 20 % mehr Heapspeicher als eine Standard-Java-Anwendung.

Die JVM erzeugt standardmäßig die folgende Diagnoseausgabe, wenn ein nicht abgefangener `OutOfMemoryError` auftritt:

- Kurzspeicherauszug; siehe „Speicherauszugsagenten verwenden“ auf Seite 40.
- Heapspeicherauszug; siehe „Heapspeicherauszug verwenden“ auf Seite 48.
- Java-Speicherauszug; siehe „Java-Speicherauszug verwenden“ auf Seite 43.
- Systemspeicherauszug; siehe „Systemspeicherauszüge und die Anzeigefunktion für Speicherauszüge verwenden“ auf Seite 51.

Die Speicherauszugsdateinamen werden in der Konsolenausgabe angegeben:

```
JVMDUMP006I Speicherauszugsereignis "systhrow", Detail "java/lang/OutOfMemoryError" wird verarbeitet
- bitte warten.
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 Speicherauszugsereignis "systhrow", Detail "java/lang/OutOfMemoryError" wurde verarbeitet.
```

Der in der Konsolenausgabe gezeigte und im Java-Speicherauszug enthaltene Java-Backtrace gibt an, wo der `OutOfMemoryError` in der Java-Anwendung aufgetreten ist. Die JVM-Speicherverwaltungskomponente gibt einen Tracepunkt aus, der die Größe, die Klassenblockadresse und den Speicherbereichsnamen der fehlgeschlagenen Zuordnung angibt. Diesen Tracepunkt finden Sie im Kurzspeicherauszug:


```
<< Zeilen ausgelassen... >>
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
```

Die Tracepunkt-ID und Datenfelder weichen je nach dem zugeordneten Objekttyp möglicherweise von den angezeigten Elementen ab. In diesem Beispiel zeigt der Tracepunkt, dass der Zuordnungsfehler auftrat, als die Anwendung versuchte, ein 33,6 MB großes Objekt des Typs class 0x81312d8 im Speichersegment id=0x809c5f0 des Metronomheapspeichers zuzuordnen.

Sie können anhand der Speicherverwaltungsinformationen im Java-Speicherauszug ermitteln, welcher Hauptspeicherbereich betroffen ist:

```
NULL      -----
0SECTION  MEMINFO subcomponent dump routine
NULL      =====
NULL
1STMENTYPE  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)
```

<< Zeilen aus Gründen der Übersichtlichkeit entfernt >>

Sie können den Objekttyp ermitteln, der zugeordnet wird, indem Sie den Klassenabschnitt im Java-Speicherauszug prüfen:

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

Die Informationen im Java-Speicherauszug bestätigen, dass die versuchte Zuordnung für ein Zeichenarray im normalen Heapspeicher (ID=0xF288B584) vorgenommen werden sollte und dass die in der entsprechenden Zeile 1STHEAP angegebene Gesamtgröße zugeordneten Heapspeichers 67108864 Dezimalbyte oder 0x04000000 Hexadezimalbyte ist, d. h., sie beträgt 64 MB.

In diesem Beispiel ist die fehlgeschlagene Zuordnung in Beziehung zur Gesamtgröße des Heapspeichers groß. Wenn Ihre Anwendung 33 MB große Objekte erstellen soll, ist der nächste Schritt die Erhöhung des Heapspeichers mit der Option **-Xmx**.

Gewöhnlich ist die fehlgeschlagene Zuordnung in Beziehung zur Gesamtgröße des Heapspeichers klein. Dies liegt daran, dass vorherige Zuordnungen den Heapspeicher auffüllen. In diesen Fällen ist der nächste Schritt die Verwendung des Heapspeicherauszugs, um die Speicherkapazität zu prüfen, die vorhandenen Objekten zugeordnet ist.

Der Heapspeicherauszug ist eine komprimierte Binärdatei, die eine Liste aller Objekte mit der zugehörigen Objektklasse, der Größe und Verweisen enthält. Analysieren Sie den Heapspeicherauszug mit dem Tool 'Memory Dump Diagnostics for Java' (MDD4J), das über IBM Support Assistant (ISA) heruntergeladen werden kann.

Mit MDD4J können Sie einen Heapspeicherauszug laden und Baumstrukturen nach Objekten durchsuchen, von denen Sie vermuten, dass sie viel Heapspeicher belegen. Das Tool bietet verschiedene Ansichten für Objekte im Heapspeicher. Beispielsweise kann MDD4J eine Ansicht anzeigen, die wahrscheinliche Leckkandidaten aufführt und die fünf Objekte und Pakete angibt, die am meisten zur Größe des Heapspeichers beitragen. Durch die Auswahl der Baumstrukturansicht erhalten Sie weitere Informationen zur Art des Containerobjektlecks.

Speicherverwaltung durch die IBM JVM

Die IBM JVM benötigt Speicher für mehrere verschiedene Komponenten, einschließlich Speicherbereiche für Klassen, kompilierten Code, Java-Objekte, Java-Stacks und JNI-Stacks. Einige dieser Speicherbereiche müssen in zusammenhängendem Speicher angeordnet sein. Andere Speicherbereiche können in kleinere Speicherbereiche segmentiert und verknüpft werden.

Dynamisch geladene Klassen und kompilierter Code werden in segmentierten Speicherbereichen für dynamisch geladene Klassen gespeichert. Klassen sind in beschreibbare Speicherbereiche (RAM-Klassen) und Nur-Lese-Speicherbereiche (ROM-Klassen) unterteilt. Während der Ausführung wird ROM-Klassen und AOT-Code aus dem Klassencache Speicher zugeordnet, sie werden beim Anwendungsstart jedoch nicht in einen zusammenhängenden Speicherbereich geladen. Beim Verweisen auf Klassen durch die Anwendung werden Klassen und kompilierter Code im Klassencache dem Speicher zugeordnet. Die ROM-Komponente der Klasse wird von mehreren auf diese Klasse verweisenden Prozessen gemeinsam genutzt. Die RAM-Komponente der Klasse wird in den segmentierten Speicherbereichen für dynamisch geladene Klassen erstellt, wenn die JVM zum ersten Mal auf die Klasse verweist. Der mit AOT kompilierte Code für die Methoden einer Klasse im Klassencache wird in einen Speicherbereich für ausführbaren dynamischen Code kopiert, weil dieser Code von Prozessen nicht gemeinsam genutzt wird. Klassen, die nicht aus dem Klassencache geladen werden, ähneln zwischengespeicherten Klassen, außer dass die ROM-Klasseninformationen in segmentierten Speicherbereichen für dynamisch geladene Klassen erstellt werden. Dynamisch generierter Code wird in denselben Speicherbereichen für dynamischen Code gespeichert, die den AOT-Code für zwischengespeicherte Klassen enthalten.

Der Stack für jeden Java-Thread kann ein segmentierter Speicherbereich sein. Der JNI-Stack für jeden Thread belegt einen zusammenhängenden Speicherbereich.

Führen Sie Ihre JVM mit der Option **-verbose:sizes** aus, um zu ermitteln, wie die JVM konfiguriert ist. Diese Option gibt Informationen zu Speicherbereichen aus, deren Größe Sie verwalten können. Für nicht zusammenhängende Speicherbereiche wird ein Inkrement ausgegeben, das beschreibt, wie viel Speicher angefordert wird, wenn der Bereich vergrößert werden muss.

Es folgt eine Beispielausgabe, die die Optionen **-Xrealtime -verbose:sizes** verwendet:

<code>-Xmca32K</code>	RAM class segment increment
<code>-Xmco128K</code>	ROM class segment increment
<code>-Xms64M</code>	initial memory size
<code>-Xmx64M</code>	memory maximum
<code>-Xmso256K</code>	operating system thread stack size
<code>-Xiss2K</code>	java thread stack initial size
<code>-Xssi16K</code>	java thread stack increment
<code>-Xss256K</code>	java thread stack maximum size

Dieses Beispiel gibt an, dass das RAM-Klassensegment anfänglich 0 ist, jedoch bei Bedarf um 32-KB-Blöcke vergrößert wird.

Das ROM-Klassensegment ist anfänglich 0 und wird bei Bedarf um 128-KB-Blöcke vergrößert. Sie können diese Größen mit den Optionen **-Xmca** und **-Xmco** steuern. RAM- und ROM-Klassensegmente werden bei Bedarf vergrößert. Daher brauchen Sie diese Optionen in der Regel nicht zu ändern.

Mit der Option **-Xshareclasses** können Sie ermitteln, wie groß Ihr dem Speicher zugeordneter Bereich ist, wenn Sie den Klassencache verwenden. Es folgt ein Beispiel für die Ausgabe vom Befehl `java -Xgcpolicy:metronome -Xshareclasses:printStats`.

```
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
```

Während der Ausführung werden ungefähr 3 MB von AOT-Byte und Metadatenbyte beim Verweis auf die Klassen in den segmentierten Bereich für dynamischen Code kopiert. Die Datenbyte werden beim Verweis auf die Klassen in den segmentierten Bereich für die RAM-Klasse kopiert.

Diagnosetools verwenden

Ihnen stehen zahlreiche Diagnosetools für das Diagnostizieren von Problemen mit IBM WebSphere Real Time for Linux JVM zur Verfügung.

IBM SDK for Java 7 bietet zahlreiche Diagnosetools für das Diagnostizieren von Problemen mit IBM WebSphere Real Time for Linux JVM. In diesem Abschnitt werden die verfügbaren Tools eingeführt und Links zu weiteren Informationen zur Verwendung der Tools bereitgestellt.

Bei der Verwendung der SDK-Diagnosetools müssen Sie einen wichtigen Punkt beachten. Wenn Sie die Echtzeit-JVM aufrufen, verwenden Sie die folgende Option:

```
java -Xgcpolicy:metronome
```

Diese Option muss verwendet werden, wenn Sie Diagnosetools für die Echtzeit-JVM ausführen. Sollen z. B. die registrierten Speicherauszugsagenten für IBM WebSphere Real Time for Linux JVM angezeigt werden, geben Sie Folgendes ein:

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

Weitere Unterschiede bei der Verwendung dieser Tools mit IBM WebSphere Real Time for Linux werden als ergänzende Informationen geliefert. Außerdem wird eine Musterausgabe bereitgestellt, um Sie bei der Diagnose zu unterstützen.

Eine Zusammenfassung der von IBM SDK for Java 7 generierten Diagnoseinformationen finden Sie in Zusammenfassung der Diagnoseinformationen.

Speicherauszugsagenten verwenden

Speicherauszugsagenten werden während der JVM-Initialisierung eingerichtet. Mit ihrer Hilfe können Sie anhand von Ereignissen, die in der JVM auftreten (z. B. Garbage-Collection, Threadstart oder JVM-Beendigung), Speicherauszüge auslösen oder ein externes Tool starten.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zu Speicherauszugsagenten:

- Option **-Xdump** verwenden
- Speicherauszugsagenten
- Speicherauszugsereignisse
- Erweiterte Steuerung von Speicherauszugsagenten
- Tokens für Speicherauszugsagenten
- Standardspeicherauszugsagenten
- Speicherauszugsagenten entfernen
- Umgebungsvariablen für Speicherauszugsagenten
- Signalzuordnungen
- Standardpositionen von Speicherauszugsagenten

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Speicherauszugsagenten verwenden.

Ergänzende Informationen für IBM WebSphere Real Time for Linux finden Sie an folgender Stelle:

Speicherauszugsereignisse

Speicherauszugsagenten werden durch Ereignisse ausgelöst, die während des JVM-Betriebs auftreten. Bei IBM WebSphere Real Time for Linux beträgt der Standardwert für das langsame Ereignis 5 Millisekunden.

Einige Ereignisse können gefiltert werden, um die Relevanz der Ausgabe zu verbessern. Weitere Informationen finden Sie in „Option 'filter'“ auf Seite 42.

Anmerkung: Die Ereignisse 'unload' und 'expand' treten in WebSphere Real Time zurzeit nicht auf. Klassen befinden sich im Immortal Memory und können nicht entladen werden.

Anmerkung: Die Ereignisse 'gpf' und 'abort' können keinen Heapspeicherauszug auslösen und den Heapspeicher nicht vorbereiten (request=prewalk) oder komprimieren (request=compact).

In der folgenden Tabelle sind Ereignisse aufgeführt, die als Auslöser für Speicherauszugsagenten verfügbar sind:

Event	Auslöser	Filteroperation
gpf	Es gibt ein Problem beim allgemeinen Zugriffsschutz (General Protection Fault, GPF).	
Benutzer	Die JVM empfängt das Signal SIGQUIT vom Betriebssystem.	
abort	Die JVM empfängt das Signal SIGABRT vom Betriebssystem.	
vmstart	Die virtuelle Maschine wird gestartet.	
vmstop	Die virtuelle Maschine wird gestoppt.	Filterung nach Exit-Code, z. B. filter=#129..#192#-42#255
load	Eine Klasse wird geladen.	Filterung nach Klassenname, z. B. filter=java/lang/String
unload	Eine Klasse wird entladen.	
throw	Es wird eine Ausnahmebedingung ausgelöst.	Filterung nach Ausnahmeklassenname, z. B. filter=java/lang/OutOfMem*
catch	Es wird eine Ausnahmebedingung abgefangen.	Filterung nach Ausnahmeklassenname, z. B. filter=*Memory*
uncaught	Eine Java-Ausnahmebedingung wird nicht von der Anwendung abgefangen.	Filterung nach Ausnahmeklassenname, z. B. filter=*MemoryError
systhrow	Eine Java-Ausnahmebedingung steht kurz davor, von der JVM ausgelöst zu werden. Dies unterscheidet sich vom Ereignis 'throw', weil es nur für Fehlerbedingungen ausgelöst wird, die intern in der JVM erkannt werden.	Filterung nach Ausnahmeklassenname, z. B. filter=java/lang/OutOfMem*
thrstart	Ein neuer Thread wird gestartet.	
blocked	Ein Thread wird geblockt.	
thrstop	Ein Thread wird gestoppt.	
fullgc	Es wird ein Garbage-Collection-Zyklus gestartet.	
slow	Ein Thread benötigt mehr als 5 ms, um auf eine interne JVM-Anforderung zu antworten.	Ändert die für ein Ereignis benötigte Zeit, ab der es als langsam gilt; Beispiel: filter=#300ms führt zu einer Auslösung, wenn ein Thread mehr als 300 ms benötigt, um auf eine interne JVM-Anforderung zu antworten.
allocation	Es wird ein Java-Objekt mit einer Größe zugeordnet, die der angegebenen Filterspezifikation entspricht.	Filterung nach Objektgröße; es muss ein Filter angegeben werden. Beispiel: filter=#5m führt bei Objekten mit einer Größe von mehr als 5 MB zu einer Auslösung. Es werden auch Bereichsangaben unterstützt; beispielsweise führt filter=#256k..512k bei Objekten mit einer Größe zwischen 256 KB und 512 KB zu einer Auslösung.
traceassert	In der JVM ist ein interner Fehler aufgetreten.	Nicht zutreffend.
corruptcache	Die JVM stellt fest, dass der Cache für gemeinsam genutzte Klassen beschädigt ist.	Nicht zutreffend.

Option 'filter'

Einige JVM-Ereignisse treten während der Lebensdauer einer Anwendung unzählige Male auf. Speicherauszugsagenten können mithilfe von Filtern und Bereichen die übermäßige Erstellung von Speicherauszügen verhindern.

Platzhalterzeichen

Sie können in einem Ausnahmebedingungsereignisfilter ein Platzhalterzeichen verwenden, indem Sie nur am Anfang oder Ende des Filters einen Stern angeben. Der folgende Befehl funktioniert nicht, weil der zweite Stern nicht am Ende des Filters steht:

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

Damit dieser Filter funktioniert, muss er wie folgt geändert werden:

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

Klassenlade- und Ausnahmebedingungsereignisse

Sie können Klassenladeereignisse (load) und Ausnahmebedingungsereignisse (throw, catch, uncaught, systhrow) nach dem Java-Klassennamen filtern:

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

Sie können die Ausnahmebedingungsereignisse throw, uncaught und systhrow nach dem Java-Methodennamen filtern:

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

Optionale Angaben stehen in eckigen Klammern.

Sie können 'catch'-Ausnahmebedingungsereignisse nach dem Java-Methodennamen filtern:

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

Optionale Angaben stehen in eckigen Klammern.

Ereignis vmstop

Sie können das JVM-Systemabschlussereignis mithilfe eines oder mehrerer Exit-Codes filtern:

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

Ereignis 'slow'

Sie können das Ereignis 'slow' filtern, um den Standardschwellenwert von 5 ms für die Zeit zu ändern:

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

Sie können den Filter auf eine Zeit setzen, die unter der Standardzeit liegt.

Ereignis 'allocation'

Sie müssen das Ereignis 'allocation' filtern, um die Größe von Objekten anzugeben, die einen Auslöser verursachen. Sie können die Filtergröße auf einen Wert von null bis zum Maximalwert eines 32-Bit-Zeigers auf 32-Bit-Plattformen bzw. bis zum Maximalwert eines 64-Bit-Zeigers auf 64-Bit-Plattformen setzen. Wird der niedrigere Filterwert auf null gesetzt, löst dies einen Speicherauszug für alle Zuordnungen aus.

Geben Sie beispielsweise Folgendes an, um Speicherauszüge für Zuordnungen mit einer Größe von mehr als 5 MB auszulösen:

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

Geben Sie Folgendes an, um Speicherauszüge für Zuordnungen mit einer Größe zwischen 256 KB und 512 KB auszulösen:

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

Sonstige Ereignisse

Wenn Sie einen Filter auf ein Ereignis anwenden, das keine Filterung unterstützt, wird der Filter ignoriert.

Java-Speicherauszug verwenden

Bei einem Java-Speicherauszug werden Dateien erstellt, die Diagnoseinformationen zur JVM und zu einer Java-Anwendung enthalten, die zu einem Zeitpunkt während der Ausführung erfasst wurden. Dies können beispielsweise Informationen über das Betriebssystem, die Anwendungsumgebung, Threads, Stacks, Sperren und den Speicher sein.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zu Java-Speicherauszügen:

- Java-Speicherauszug aktivieren
- Java-Speicherauszüge auslösen
- Java-Speicherauszug interpretieren
- Umgebungsvariablen und Java-Speicherauszug

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Java-Speicherauszug verwenden.

Ergänzende Informationen und eine Musterausgabe für IBM WebSphere Real Time for Linux finden Sie in den folgenden Themen.

Speichermanagement (MEMINFO)

Der Abschnitt MEMINFO enthält Informationen zum Memory Manager (Speichermanager), einschließlich Speicherbereiche für den Heapspeicher sowie Speicher für Objekte mit unbeschränkter und beschränkter Lebensdauer.

Der Abschnitt MEMINFO eines Java-Speicherauszugs enthält Informationen zum Memory Manager (Speichermanager). Informationen zur Funktionsweise der Speichermanagerkomponente finden Sie in Using the Metronome Garbage Collector.

Dieser Teil des Java-Speicherauszugs enthält verschiedene Werte zum Speicherma-
nagement:

- Freie Speicherkapazität
- Belegte Speicherkapazität
- Aktuelle Größe des Heapspeichers
- Aktuelle Größe der Speicherbereiche für Objekte mit unbeschränkter Lebens-
dauer
- Aktuelle Größe der Speicherbereiche für Objekte mit beschränkter Lebensdauer

Darüber hinaus enthält dieser Abschnitt Protokolldaten zur Garbage-Collection.
Die Daten werden in Form von Tracepunkten angezeigt, die mit einer Zeitmarke
versehen sind. Dabei steht der aktuellste Tracepunkt an erster Stelle.

Java-Speicherauszüge, die mit der Standard-JVM erstellt wurden, enthalten einen
Abschnitt mit der Bezeichnung „GC History“ (Protokolldaten zur Garbage-Collec-
tion). Diese Informationen sind nicht in Java-Speicherauszügen enthalten, die mit
der Echtzeit-JVM erstellt wurden. Informationen zum Verhalten der Garbage-Col-
lection erhalten Sie durch Angabe der Option **-verbose:gc** oder über den JVMn-
Snap-Trace. Weitere Details finden Sie in „Informationen von 'verbose:gc'
verwenden“ auf Seite 60 und im Abschnitt zu Speicherauszugsagenten des Benut-
zerhandbuchs für IBM SDK for Java 7.

In einem Java-Speicherauszug sind Segmente jeweils Blöcke von Speicher, denen
die Java-Laufzeitumgebung Tasks mit großem Speicherbedarf zuordnet. Beispiel-
tasks:

- JIT-Caches verwalten
- Java-Klassen speichern

Java Runtime ordnet auch anderen nativen Speicher zu, der im Abschnitt MEMINFO
nicht aufgelistet ist. Der von den Segmenten der Java-Laufzeitumgebung verwen-
dete Gesamtspeicher entspricht nicht zwangsläufig dem vollständigen Speicherbe-
darf der Java-Laufzeitumgebung. Ein Java Runtime-Segment besteht aus der Seg-
mentdatenstruktur und einem zugeordneten Block von nativem Speicher.

Bei dem folgenden Beispiel handelt es sich um eine typische Ausgabe. Alle Werte
werden als Hexadezimalwerte bereitgestellt. Die Spaltenüberschriften im Abschnitt
MEMINFO haben die folgenden Bedeutungen:

```
| 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
|              (Zeilen zur Verdeutlichung entfernt)
| 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
| (Zeilen zur Verdeutlichung entfernt)
| 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
| (Zeilen zur Verdeutlichung entfernt)
| NULL
| NULL -----

```

Threads und Stack-Trace (THREADS)

Der Abschnitt THREADS ist für Anwendungsprogrammierer einer der hilfreichsten Abschnitte in einem Java-Speicherauszug. Er enthält eine Liste der Java-Threads, der nativen Threads und der Stack-Traces.

Java-Threads werden durch native Threads des Betriebssystems implementiert. Jeder Thread wird durch eine Gruppe von Zeilen wie die folgenden dargestellt:

```

"main" JVMThread: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)

```

Die Namen der Java-Threads können über den Befehl **ps** im Betriebssystem angezeigt werden. Weitere Informationen zur Verwendung des Befehls **ps** finden Sie in „Allgemeine Debugging-Verfahren“ auf Seite 30.

In einem Java-Speicherauszug, der mit einem No-Heap-Echtzeitthread erstellt wird, fehlen unter Umständen einige Informationen. Ist das Threadnamensobjekt aus dem No-Heap-Echtzeitthread nicht ersichtlich, wird anstelle des tatsächlichen Threadnamens der Text „(access error)“ (Zugriffsfehler) angezeigt.

Bei den Eigenschaften in der ersten Zeile handelt es sich um den Threadnamen, die Adressen der JVM-Threadstrukturen und das Java-Threadobjekt, den Threadstatus und die Priorität des Java-Threads. Bei den Eigenschaften in der zweiten Zeile handelt es sich um die ID und die Priorität des nativen Betriebssystemthreads sowie um die native Planungsrichtlinie des Betriebssystems.

Es gibt drei Möglichkeiten, die Threadnamen zu finden:

- In Javacore-Dateien. Allerdings sind nicht alle Threads in den Javacore-Dateien enthalten.
- Beim Auflisten der Threads über das Betriebssystem mit dem Befehl **ps**.
- Bei Verwendung der Methode `java.lang.Thread.getName()`.

In der folgenden Tabelle werden Informationen zu Threadnamen in IBM WebSphere Real Time for Linux aufgelistet.

Tabelle 2. Threadnamen in IBM WebSphere Real Time for Linux

Informationen zum Thread	Threadname
Ein interner JVM-Thread, mit dem das Garbage-Collection-Modul die Endbearbeitung von Objekten durch sekundäre Threads zuteilt.	Finalizer master
Der vom Garbage-Collector verwendete Alarmthread.	GC Alarm
Die für die Garbage-Collection verwendeten Slave-Threads.	GC Slave
Ein interner JVM-Thread, mit dem das JIT-Compilermodul (Just-in-time) Stichproben zur Nutzung von Methoden in der Anwendung nimmt.	JIT Sampler
Ein Thread, mit dem die virtuelle Maschine die von der Anwendung empfangenen Signale verwaltet; dabei spielt es keine Rolle, ob die Signale extern oder intern generiert wurden.	Signal Reporter

Die Priorität der Java-Threads wird plattformabhängig einem Prioritätswert des Betriebssystems zugeordnet. Eine Java-Threadpriorität mit einem großen Wert besagt, dass es sich um einen Thread mit hoher Priorität handelt. Dieser Thread wird häufiger ausgeführt als Threads mit niedrigerer Priorität.

Folgende Statuswerte sind möglich:

- R (Runnable): Der Thread kann bei Bedarf ausgeführt werden.
- CW (Condition Wait): Der Thread befindet sich im Wartestatus, beispielsweise aus den folgenden Gründen:
 - Ein sleep()-Aufruf wurde ausgegeben.
 - Der Thread wurde für eine Ein-/Ausgabe geblockt.
 - Eine wait()-Methode wurde aufgerufen, um auf die Benachrichtigung eines Monitors zu warten.
 - Der Thread wird über einen join()-Aufruf mit einem anderen Thread synchronisiert.
- S (Suspended): Der Thread wurde von einem anderen Thread ausgesetzt.
- Z (Zombie): Der Thread wurde abgebrochen.
- P (Parked): Der Thread wurde von der neuen Concurrency-API von Java (java.util.concurrent) vorgehalten bzw. "geparkt".
- B (Blocked): Der Thread wartet auf die Übernahme einer Sperre, die gerade anderweitig vergeben ist.

Ist ein Thread geparkt oder blockiert, enthält die Ausgabe eine mit 3XMTHREADBLOCK beginnende Zeile für diesen Thread, in der die Ressource, auf die der Thread wartet und (sofern möglich) der Thread aufgelistet ist, der zurzeit Eigner dieser Ressource ist. Weitere Informationen finden Sie im Thema zu geblockten Threads im Benutzerhandbuch für IBM SDK for Java 7.

Wenn Sie einen Java-Speicherauszug zum Abrufen von Diagnoseinformationen initialisieren, versetzt die JVM Java-Threads vor der Erzeugung des Java-Kernspeichers in den Wartemodus. In der Zeile 1TIPREPSTATE des Abschnitts TITLE wird der Vorbereitungsstatus `exclusive_vm_access` angezeigt.

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

Threads, die beim Auslösen des Java-Kernspeichers Java-Code ausführten, befinden sich im Status CW (Condition Wait).

```
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)
```

Der Abschnitt LOCKS des Java-Kernspeichers zeigt, dass diese Threads auf eine interne JVM-Sperre warten.

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

Heapspeicherauszug verwenden

Der Begriff Heapspeicherauszug beschreibt den IBM Virtual Machine for Java-Mechanismus, bei dem ein Auszug von allen Liveobjekten, die sich im Java-Heapspeicher befinden, erstellt wird, d. h., von den Objekten, die gerade von der aktiven Java-Anwendung verwendet werden.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zu Heapspeicherauszügen:

- Heapspeicherauszüge abrufen
- Tools für die Verarbeitung von Heapspeicherauszügen
- **-Xverbose:gc** zum Abrufen von Heapspeicherinformationen verwenden
- Umgebungsvariablen und Heapspeicherauszug
- Textformat (klassisches Format) der Heapspeicherauszugsdatei
- PHD-Dateiformat

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Heapspeicherauszug verwenden.

Ergänzende Informationen für IBM WebSphere Real Time for Linux:

Textformat (klassisches Format) der Heapspeicherauszugsdatei

Der Heapspeicherauszug im Textformat oder klassischen Format ist eine Liste aller Objektinstanzen im Heapspeicher (einschließlich Objekttyp, Größe und Verweise zwischen Objekten).

Kopfsatz

Der Kopfsatz ist ein einzelner Datensatz, der eine Zeichenfolge aus Versionsinformationen enthält.

```
// Version: <Versionszeichenfolge mit SDK-Version, Plattform und  
JVM-Buildstufe>
```

Beispiel:

```
// Version: J2RE 7.0 IBM J9 2.6 Linux x86-32 build 20101016_024574_1HdRSr
```

Objektdatensätze

Objektdatensätze bestehen aus mehreren Datensätzen (einer für jede Objektinstanz im Heapspeicher), die Informationen wie Objektadresse, Größe, Typ und Verweise aus dem Objekt bereitstellen.

```
<Objektadresse, hexadezimal> [<Länge der Objektinstanz in Bytes, dezimal>]  
OBJ <Objekttyp> <Klassenblockverweise, hexadezimal>  
<Heapspeicherverweis, hexadezimal <Heapspeicherverweis,  
hexadezimal> ...
```

Die Objektadresse und Heapspeicherverweise befinden sich im Heapspeicher, die Klassenblockadresse jedoch außerhalb des Heapspeichers. Alle in der Objektinstanz gefundenen Verweise werden aufgelistet, einschließlich der Verweise mit Nullwerten. Der Objekttyp ist entweder ein Klassenname (einschließlich Paket) oder ein primitives Array oder Klassen-Array, dessen JVM-Standardtypkennung angezeigt wird (siehe „Java VM-Typkennungen“ auf Seite 51). Objektdatensätze können außerdem zusätzliche Klassenblockverweise enthalten, zum Beispiel bei Reflexionsklasseninstanzen.

Beispiele:

Objektinstanz mit einer Länge von 28 Bytes und dem Typ 'java/lang/String':

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

Klassenblockadresse des Typs 'java/lang/String', gefolgt von einem Verweis auf eine Array-Instanz des Typs 'char':

```
0x415319D8 0x00436EB0
```

Objektinstanz mit einer Länge von 44 Bytes und dem Array-Typ 'char':

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

Klassenblockadresse von char-Array:

```
0x41530F20
```

Objekt des Typs Array der untergeordneten Klasse 'java/util/Hashtable Entry':

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

Objekt des Typs untergeordnete Klasse 'java/util/Hashtable Entry':

```
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
```

Klassenblockadresse und Heapspeicherverweise, einschließlich Verweise mit Nullwerten:

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

Klassendatensätze

Klassendatensätze bestehen aus mehreren Datensätzen (einer für jede geladene Klasse), die Informationen wie Klassenblockadresse, Größe, Typ und Verweise aus der Klasse bereitstellen.

```
<Klassenblockadresse,
hexadezimal> [<Länge des Klassenblocks in Bytes, dezimal>]
CLS <Klassentyp>
<Klassenblockverweis, hexadezimal> <Klassenblockverweis, hexadezimal> ...
<Heapspeicherverweis, hexadezimal> <Heapspeicherverweis, hexadezimal>...
```

Die Klassenblockadresse und Klassenblockverweise befinden sich außerhalb des Heapspeichers, aber der Klassendatensatz kann auch Verweise auf den Heapspeicher enthalten, zum Beispiel bei statischen Klassendatenelementen. Alle im Klassenblock gefundenen Verweise werden aufgelistet, einschließlich der Verweise mit Nullwerten. Der Klassentyp ist entweder ein Klassenname (einschließlich Paket) oder ein primitives Array oder Klassen-Array, dessen JVM-Standardtypkennung angezeigt wird (siehe „Java VM-Typkennungen“ auf Seite 51).

Beispiele:

Klassenblock mit einer Länge von 32 Byte, für Klasse 'java/lang/Runnable':

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

Verweise auf andere Klassenblöcke und Heapspeicherverweise, einschließlich Verweise mit Nullwerten:

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

Klassenblock mit einer Länge von 168 Bytes, für Klasse 'java/lang/Math':

```
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
```

Trailerdatensatz 1

Trailerdatensatz 1 ist ein einzelner Datensatz, der Satzzählungen enthält.

```
// Breakdown - Classes: <Klassendatensatzzählung, dezimal>,
Objects: <Objektdatensatzzählung, dezimal>,
ObjectArrays: <Objekt-Array-Datensatzzählung, dezimal>,
PrimitiveArrays: <Primitiv-Array-Datensatzzählung, dezimal>
```

Beispiel:

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

Trailerdatensatz 2

Trailerdatensatz 2 ist ein einzelner Datensatz, der Gesamtzahlen enthält.

```
// EOF: Total 'Objects',Refs(null) :  
<Gesamtzahl Objektzählung, dezimal>,  
<Gesamtzahl Verweiszählung, dezimal>  
(,Gesamtzahl Nullverweiszählung, dezimal>)
```

Beispiel:

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

Java VM-Typkennungen

Die Java VM-Typkennungen sind Abkürzungen für die Java-Typen. Sie werden in der folgenden Tabelle aufgeführt:

Java VM-Typkennung	Java-Typ
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L <vollständig qualifizierte Klasse> ;	<vollständig qualifizierte Klasse>
[<Typ>	<Typ>[] (Array von <Typ>)
(<arg-Typen>) <ret-Typ>	method

Systemspeicherauszüge und die Anzeigefunktion für Speicherauszüge verwenden

Die JVM kann unter konfigurierbaren Bedingungen native Systemspeicherauszüge, die auch als Kernspeicherauszüge bekannt sind, generieren. Systemspeicherauszüge sind in der Regel umfangreich. Die meisten Tools zum Analysieren von Systemspeicherausügen sind außerdem plattformspezifisch. Verwenden Sie das Tool **gdb** zum Analysieren eines Systemspeicherauszugs unter Linux.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zur Verwendung von Speicherauszugsagenten und der Anzeigefunktion für Speicherauszüge:

- Übersicht über Systemspeicherauszüge
- Standardeinstellungen für Systemspeicherauszüge
- Anzeigefunktion für Speicherauszüge verwenden
 - **jextract** verwenden
 - Mit der Anzeigefunktion für Speicherauszüge zu lösende Probleme
 - In **jdmview** verfügbare Befehle
 - Beispielsitzung
 - **jdmview**/jdmview-Befehle - Kurzübersicht

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Speicherauszüge und Anzeigefunktion für Speicherauszüge verwenden.

Ergänzende Informationen für IBM WebSphere Real Time for Linux:

In `jdmview` verfügbare Befehle

`jdmview` ist ein interaktives Befehlszeilentool zur Untersuchung der Informationen aus einem JVM-Systemspeicherauszug und zur Ausführung verschiedener Analysefunktionen.

info `jitm`

Listet mit AOT und JIT kompilierte Methoden und deren Adressen auf:

- Methodename und Signatur
- Startadresse der Methode
- Endadresse der Methode

Alle anderen Befehlsoptionen finden Sie im Benutzerhandbuch für IBM SDK for Java 7.

Tracefunktion von Java-Anwendungen und der JVM

'JVM trace' ist eine in IBM WebSphere Real Time for Linux bereitgestellte Tracefunktion, die die Leistung nur unwesentlich beeinträchtigt. In den meisten Fällen werden die Tracedaten in einem kompakten Binärformat gespeichert, das mit dem mitgelieferten Java-Formatierungsprogramm formatiert werden kann.

Die Tracefunktion ist standardmäßig aktiviert, zusammen mit einer kleinen Anzahl an Tracepunkten, die auf Hauptspeicherpuffer verweisen. Sie können Tracepunkte während der Ausführung aktivieren, indem Sie Stufen, Komponenten, Gruppennamen oder einzelne Tracepunktkennungen verwenden.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält ausführliche Informationen zur Erstellung von Anwendungstraces:

- Traceziele
- Tracepunkt-Arten
- Standardtrace
- Tracedaten erfassen
- Tracesteuerung
- Traceerstellung für Java-Anwendungen
- Traceerstellung für Java-Methoden

Bei der IBM WebSphere Real Time for Linux-Traceerstellung müssen Sie die Echtzeit-JVM ordnungsgemäß aufrufen, wenn Sie Traceoptionen einschließen. Geben Sie z. B. beim Angeben von Traceoptionen Folgendes ein:

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

Sie finden Informationen zu IBM SDK for Java 7 an folgender Stelle: Traceerstellung für Java-Anwendungen und JVM.

Fehlerbestimmung für JIT und AOT

Mithilfe von Befehlszeilenoptionen können Sie Probleme und Fehler in Zusammenhang mit dem JIT-Compiler und dem AOT-Compiler diagnostizieren und die Leistung der Compiler optimieren.

IBM WebSphere Real Time for Linux nutzt zwar einige allgemeine Komponenten gemeinsam mit IBM SDK for Java 7, das Verhalten von JIT und AOT ist jedoch unterschiedlich. In diesem Abschnitt wird die Fehlerbehebung für JIT- und AOT-Probleme unter IBM WebSphere Real Time for Linux behandelt.

JIT- oder AOT-Probleme diagnostizieren

Hin und wieder kann es vorkommen, dass gültiger Bytecode bei der Kompilierung zu ungültigem nativen Code wird, sodass das Java-Programm abstürzt. Sie können dem Java-Service-Team wertvolle Hinweise geben, mit deren Hilfe es feststellen kann, ob der Fehler beim JIT- oder AOT-Compiler liegt und wenn ja, *wo* genau die Fehlerursache liegt.

Informationen zu diesem Vorgang

Mit der Option `-Xaot:verbose` in der `admincache`-Befehlszeile können Sie die Methoden ermitteln, die beim Auffüllen des Cache für gemeinsam genutzte Klassen kompiliert werden. Beispiel:

```
admincache -Xrealtime -Xaot:verbose -populate -aot my.jar -cp <Ihr_Klassenpfad>
```

In diesem Abschnitt wird beschrieben, wie Sie feststellen können, ob ein Problem durch den Compiler verursacht wurde. Darüber hinaus werden mögliche Strategien zur Fehlerbehebung sowie Debugverfahren aufgeführt, mit denen Probleme in Zusammenhang mit dem Compiler behoben werden können.

JIT- oder AOT-Compiler inaktivieren:

Wenn Sie vermuten, dass ein Problem beim JIT- oder AOT-Compiler liegt, sollten Sie den Kompilervorgang inaktivieren, um festzustellen, ob das Problem weiterhin auftritt. Ist dies der Fall, liegt das Problem nicht beim Compiler.

Informationen zu diesem Vorgang

Der JIT-Compiler ist standardmäßig aktiviert. Der AOT-Compiler ist ebenfalls aktiviert, jedoch nur aktiv, wenn die gemeinsam genutzten Klassen aktiviert wurden. Aus Gründen der Effizienz werden nicht alle Methoden in einer Java-Anwendung kompiliert. Die Java Virtual Machine (JVM) zeichnet auf, wie oft die einzelnen Methoden in der Anwendung aufgerufen werden; bei jedem Aufruf und bei jeder Interpretation einer Methode wird der Aufrufzähler der betreffenden Methode entsprechend erhöht. Erreicht der Zähler den Grenzwert für die Kompilierung, wird die Methode kompiliert und nativ ausgeführt.

Durch die Aufrufzähler wird die Kompilierung von Methoden über die gesamte Lebensdauer der Anwendung verteilt, wobei Methoden, die häufiger aufgerufen werden, eine höhere Priorität erhalten. Methoden dagegen, die nur selten aufgerufen werden, werden möglicherweise nie kompiliert. Wenn ein Java-Programm daher fehlschlägt, kann das Problem zwar beim JIT- oder AOT-Compiler liegen, es kann aber auch an anderer Stelle in der JVM verursacht werden.

Bei der Fehlerdiagnose muss in einem ersten Schritt festgestellt werden, *wo* das Problem liegt. Dazu müssen Sie das Java-Programm zunächst im reinen Interpretationsmodus ausführen (d. h. mit inaktiviertem JIT- und AOT-Compiler).

Vorgehensweise

1. Entfernen Sie alle **-Xjit-** und **-Xaot-**Optionen (sowie alle zugehörigen Parameter) aus der Befehlszeile.
2. Inaktivieren Sie mit der Befehlszeilenoption **-Xint** den JIT- und den AOT-Compiler. Aus leistungstechnischen Gründen sollte die Option **-Xint** in Produktionsumgebungen nicht verwendet werden.

Nächste Schritte

Die Ausführung des Java-Programms mit inaktivierter Kompilierung kann folgendes Ergebnis haben:

- Das Problem besteht weiterhin, d. h., es wird nicht durch den JIT- oder AOT-Compiler verursacht. In einigen Fällen wird das Programm möglicherweise auf andere Weise abgebrochen, trotzdem liegt der Fehler nicht beim Compiler.
- Das Problem tritt nicht mehr auf. In diesem Fall wird es höchstwahrscheinlich durch den JIT- oder AOT-Compiler verursacht.

Wenn Sie keine gemeinsam verwendeten Klassen verwenden, ist der JIT-Compiler fehlerhaft. Bei Verwendung gemeinsam genutzter Klassen müssen Sie feststellen, welcher Compiler den Fehler verursacht; dazu müssen Sie die Anwendung nur mit aktivierter JIT-Kompilierung ausführen. Führen Sie die Anwendung mit der Option **-Xnoaot** anstelle der Option **-Xint** aus. Das kann folgendes Ergebnis haben:

- Das Problem besteht weiterhin, d. h., es wird durch den JIT-Compiler verursacht. Sie können auch mithilfe der Option **-Xnojit** anstelle der Option **-Xnoaot** feststellen, ob das Problem allein beim JIT-Compiler liegt.
- Das Problem tritt nicht mehr auf, d. h., es wird durch den AOT-Compiler verursacht.

JIT-Compiler gezielt inaktivieren:

Wenn der Fehler Ihres Java-Programms auf ein Problem mit dem JIT-Compiler hinweist, können Sie versuchen, den Fehler weiter einzugrenzen.

Informationen zu diesem Vorgang

Standardmäßig optimiert der JIT-Compiler Methoden auf unterschiedlichen Optimierungsstufen. Auf verschiedene Methoden werden je nach der entsprechenden Aufrufanzahl verschiedene Optimierungsoptionen angewendet. Methoden, die häufiger aufgerufen werden, werden mit einer höheren Stufe optimiert. Indem Sie Parameter des JIT-Compilers ändern, können Sie die Optimierungsstufe der Methoden steuern. Sie können ermitteln, ob das Optimierungsprogramm fehlerhaft ist, und wenn ja, welche Optimierung problematisch ist.

JIT-Parameter werden in Form einer durch Kommas getrennten Liste an die Option **-Xjit** angehängt: Die Syntax lautet wie folgt:

-Xjit:<Parameter1>,<Parameter2>=<Wert>. Beispiel:

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

Es wird das Programm HelloWorld ausgeführt, die ausführliche Ausgabe vom JIT-Compiler aktiviert und der native Code vom JIT-Compiler generiert, ohne dass Optimierungen ausgeführt werden.

So können Sie feststellen, wo im Compiler das Problem verursacht wird:

Vorgehensweise

1. Setzen Sie Grenzwert für die Kompilierung über den JIT-Parameter **count=0** auf null. Dieser Parameter bewirkt, dass jede Java-Methode vor ihrer Ausführung kompiliert wird. Verwenden Sie **count=0** nur zur Diagnose von Fehlern, weil viele weitere Methoden kompiliert werden, einschließlich Methoden, die nur selten verwendet werden. Die zusätzliche Kompilierung verbraucht mehr IT-Ressourcen und verlangsamt Ihre Anwendung. Mit **count=0** schlägt Ihre Anwendung sofort fehl, wenn der Problembereich erreicht wird. In einigen Fällen kann ein Fehlschlagen zuverlässiger mit **count=1** erreicht werden.
2. Fügen Sie den JIT-Compilerparametern **disableInlining** hinzu. Mit **disableInlining** wird die Generierung von umfangreicherem und komplexerem Code inaktiviert. Tritt das Problem nicht mehr auf, können Sie **disableInlining** als Fehlerumgehung einsetzen, bis das Compilerproblem vom Java-Service-Team analysiert und behoben wurde.
3. Verringern Sie die Optimierungsstufen, indem Sie den Parameter **optLevel** hinzufügen und das Programm erneut ausführen, bis der Fehler nicht mehr auftritt oder Sie die Stufe „noOpt“ erreichen. Starten Sie bei einem Problem mit dem JIT-Compiler mit der Optimierungsstufe „scorching“ und fahren Sie anschließend mit der nächstniedrigeren Stufe usw. fort. Hier die Optimierungsstufen in absteigender Reihenfolge:
 - a. scorching
 - b. veryHot
 - c. hot
 - d. warm
 - e. cold
 - f. noOpt

Nächste Schritte

Wenn mithilfe einer dieser Einstellungen der Fehler behoben wird, verfügen Sie über eine Fehlerumgehung, die Sie verwenden können. Diese Fehlerumgehung ist vorläufig einsetzbar, während das Compilerproblem vom Java-Service-Team analysiert und behoben wird. Wenn das Entfernen von **disableInlining** aus der JIT-Parameterliste nicht dazu führt, dass das Problem erneut auftritt, sollten Sie 'disableInlining' entfernen, um die Leistung zu erhöhen. Gehen Sie entsprechend den Anweisungen im Abschnitt „Methode ermitteln, bei der der Fehler auftritt“ vor, um die Leistung der Fehlerumgehung zu erhöhen.

Tritt das Problem auch bei der Optimierungsstufe „noOpt“ auf, müssen Sie den JIT-Compiler inaktivieren, um den Fehler zu umgehen.

Methode ermitteln, bei der der Fehler auftritt:

Wenn Sie die niedrigste Optimierungsstufe ermittelt haben, bei der der JIT- oder AOT-Compiler bei der Kompilierung von Methoden ein Problem auslöst, können Sie herausfinden, welcher Teil des Java-Programms dieses Problem bei der Kompilierung verursacht. Anschließend können Sie den Compiler anweisen, die Fehlerumgehung auf eine bestimmte Methode oder Klasse oder auf ein bestimmtes Paket zu beschränken, sodass der Compiler den Rest des Programms normal kompilieren kann. Wenn ein JIT-Compilerproblem bei **-Xjit:optLevel=noOpt** auftritt, können Sie den Compiler auch anweisen, die Methoden, die der Auslöser sind, gar nicht zu kompilieren.

Vorbereitende Schritte

Mithilfe einer Fehlernachricht wie der folgenden können Sie die Methode ermitteln, die das Problem verursacht:

```
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;
```

Folgende Zeilen sind von Bedeutung:

vmState=0x00000000

Gibt an, dass es sich bei dem Code, der fehlerhaft ist, nicht um JVM-Laufzeitcode handelt.

Module= oder Module_base_address=

Nicht in der Ausgabe (kann leer oder null sein), da der Code von JIT außerhalb einer DLL oder Bibliothek kompiliert wurde.

Compiled_method=

Gibt die Java-Methode an, für die der kompilierte Code erstellt wurde.

Informationen zu diesem Vorgang

Wird die Methode, bei der der Fehler aufgetreten ist, nicht in der Ausgabe angezeigt, können Sie sie wie folgt ermitteln:

Vorgehensweise

1. Führen Sie das Java-Programm unter Angabe der JIT-Parameter **verbose** und **vlog=<Dateiname>** mit der Option **-Xjit** oder **-Xaot** aus. Bei Angabe dieser Parameter listet der Compiler die kompilierten Methoden in einer Protokolldatei mit dem Namen *<Dateiname>.<Datum>.<Uhrzeit>.<PID>* (auch als *Begrenzungsdatei* (LimitFile) bezeichnet) auf. Eine Begrenzungsdatei enthält typischerweise Zeilen, die kompilierten Methoden entsprechen; Beispiel:

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

Zeilen, die nicht mit einem Pluszeichen beginnen, werden in den anschließenden Schritten vom Compiler ignoriert und können aus der Datei entfernt werden. Methoden, die mit dem AOT-Compiler kompiliert werden, beginnen mit **+(AOT cold)**. Methoden, für die AOT-Code aus dem Cache für gemeinsam genutzte Klassen geladen wird, beginnen mit **+(AOT load)**.

2. Führen Sie das Programm erneut mit dem JIT- oder AOT-Parameter **limitFile=(<Dateiname>,<m>,<n>)** aus; dabei ist *<Dateiname>* der Pfad zur Begrenzungsdatei und *<m>* und *<n>* sind die Zeilennummern, die die erste und die letzte Methode in der Begrenzungsdatei angeben, die kompiliert werden sollen. Der Compiler kompiliert nur die in der Begrenzungsdatei zwischen den Zeilen *<m>* und *<n>* aufgelisteten Methoden. Methoden, die nicht in der Begrenzungsdatei aufgeführt sind und nicht innerhalb des angegebenen Bereichs liegen, werden nicht kompiliert und es wird auch kein AOT-Code im Cache für gemeinsam genutzte Daten für diese Methoden geladen. Schlägt das Programm

nicht mehr fehl, müssen eine oder mehrere der in der letzten Iteration entfernten Methoden das Problem verursacht haben.

3. Optional: Wenn Sie ein AOT-Problem diagnostizieren, müssen Sie das Programm ein zweites Mal mit denselben Optionen ausführen, damit die kompilierten Methoden aus dem Cache für gemeinsam genutzte Daten geladen werden können. Sie können auch die Option `-Xaot:scout=0` angeben, um sicherzustellen, dass beim ersten Aufruf der Methode die mit dem AOT-Compiler kompilierten und im Cache für gemeinsam genutzte Daten gespeicherten Methoden verwendet werden. Einige AOT-Kompilierungsfehler treten nur auf, wenn der mit AOT kompilierte Code aus dem Cache für gemeinsam genutzte Daten geladen wird. Diese Probleme können Sie mithilfe der Option `-Xaot:scout=0` diagnostizieren, um sicherzustellen, dass beim ersten Aufruf der Methode die mit dem AOT-Compiler kompilierten und im Cache für gemeinsam genutzte Daten gespeicherten Methoden verwendet werden; unter Umständen lässt sich das Problem auf diese Weise einfacher reproduzieren. Wenn Sie die Option `scout` auf '0' setzen, wird das Laden von AOT-Code erzwungen; außerdem werden alle Anwendungsthreads angehalten, die auf die Ausführung dieser Methode warten. Daher sollte diese Einstellung nur zu Diagnosezwecken genutzt werden. Durch Angabe der Option `-Xaot:scout=0` können erheblich längere Pausezeiten entstehen.
4. Wiederholen Sie diesen Vorgang unter Angabe verschiedener Werte für `<m>` und `<n>` so oft, bis Sie die Mindestanzahl an Methoden ermittelt haben, bei deren Kompilierung das Problem auftritt. Sie können eine Binärsuche nach der Methode durchführen, die das Problem verursacht, indem Sie die Anzahl der ausgewählten Zeilen jedesmal halbieren. Häufig kann die Datei auf eine einzige Zeile reduziert werden.

Nächste Schritte

Wenn Sie die betreffende Methode ermittelt haben, können Sie den JIT- oder AOT-Compiler gezielt nur für diese Methode inaktivieren. Schlägt das Programm bei einer JIT-Kompilierung mit Optimierungsstufe `optLevel=hot` beispielsweise aufgrund der Methode `java/lang/Math.max(II)I` fehl, führen Sie das Programm wie folgt aus:

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

Damit wird nur die Methode, die das Problem verursacht, unter Verwendung der Optimierungsstufe „warm“ kompiliert; alle anderen Methoden dagegen werden ganz normal kompiliert.

Schlägt eine Methode bei einer JIT-Kompilierung mit Optimierungsstufe „noOpt“ fehl, können Sie sie unter Angabe des Parameters `exclude={<Methode>}` ganz von der Kompilierung ausschließen:

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

Verursacht eine Methode einen Programmfehler, wenn AOT-Code kompiliert oder aus dem Cache für gemeinsam genutzte Daten geladen wird, können Sie die Methode mit dem Parameter `exclude={<Methode>}` von der AOT-Kompilierung und dem AOT-Ladevorgang ausschließen:

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

AOT-Methoden werden nur mit der Optimierungsstufe „cold“ kompiliert. Daher ist es die beste Lösung für diese Methoden, die AOT-Kompilierung oder den AOT-Ladevorgang zu verhindern.

JIT-Kompilierfehler identifizieren:

Untersuchen Sie bei JIT-Compilerfehlern zunächst die Fehlernachricht, um festzustellen, ob ein Fehler bei dem Versuch des JIT-Compilers auftritt, eine Methode zu kompilieren.

Wenn die Java Virtual Machine (JVM) abstürzt und Sie erkennen können, dass der Fehler in der JIT-Bibliothek (`libj9jit26.so`) aufgetreten ist, ist der JIT-Compiler möglicherweise bei dem Versuch ausgefallen, eine Methode zu kompilieren.

Mithilfe einer Fehlernachricht wie der folgenden können Sie die betreffende Methode ermitteln:

```
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$JCMMethodDecl;)
```

Folgende Zeilen sind von Bedeutung:

vmState=0x00050000

Gibt an, dass der JIT-Compiler Code kompiliert. Eine Liste der vmState-Code-Nummern finden Sie in der Tabelle der Java-Speicherauszugstags im Benutzerhandbuch für IBM SDK for Java 7, http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html.

Module=/home/test/sdk/jre/bin/libj9jit26.so

Gibt an, dass der Fehler in `libj9jit26.so`, dem JIT-Compilermodul, aufgetreten ist.

Method_being_compiled=

Gibt die Java-Methode an, die kompiliert wird.

Wird die Methode, bei der der Fehler aufgetreten ist, nicht angezeigt, geben Sie die Option **verbose** mit den folgenden zusätzlichen Einstellungen an:

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

Mit diesen **verbose**-Einstellungen wird dokumentiert, wann der JIT-Compiler mit dem Kompilieren einer Methode beginnt und wann er den Vorgang beendet. Schlägt der JIT-Compiler bei einer bestimmten Methode fehl (d. h., der Kompilierungsvorgang wird zwar gestartet, aber vorzeitig beendet), können Sie diese Methode über die Angabe des Parameters **exclude** von der Kompilierung ausschließen (Informationen hierzu finden Sie in „Methode ermitteln, bei der der Fehler auftritt“ auf Seite 55). Kann durch den Ausschluss der Methode ein vorzeitiger Absturz vermieden werden, können Sie mit dieser Fehlerumgehung arbeiten, bis das Problem vom Service-Team behoben wurde.

Leistung von Anwendungen mit kurzer Laufzeit

Der JIT-Compiler von IBM ist für Anwendungen mit langer Laufzeit optimiert, wie sie in der Regel auf einem Server eingesetzt werden. Mit der **-Xquickstart**-Befehlszeilenoption können Sie die Leistung von Anwendungen mit kurzer Laufzeit erhöhen; dies gilt besonders für Anwendungen, bei denen die Verarbeitung nicht auf eine geringe Anzahl von Methoden konzentriert ist.

Bei Verwendung der Option **-Xquickstart** verwendet der JIT-Compiler standardmäßig eine niedrigere Optimierungsstufe und kompiliert weniger Methoden. Werden weniger Methoden schneller kompiliert, kann dies den Anwendungsstart beschleunigen. Ist der AOT-Compiler aktiv (gemeinsam genutzte Klassen und AOT-Kompilierung sind aktiviert), werden bei Angabe der Option **-Xquickstart** alle für eine Kompilierung ausgewählten Methoden mit dem AOT-Compiler kompiliert, wodurch der Start nachfolgender Ausführungen verkürzt wird. **-Xquickstart** kann bei der Verwendung für Anwendungen mit langer Laufzeit, die Methoden mit vielen Verarbeitungsressourcen enthalten, die Leistung vermindern. Änderungen an der Implementierung der Option **-Xquickstart** in künftigen Releases sind vorbehalten.

Sie können auch versuchen, die Startzeiten durch Anpassung des JIT-Schwellenwerts zu verbessern. Weitere Informationen finden Sie in „JIT-Compiler gezielt inaktivieren“ auf Seite 54.

Verhalten der Java Virtual Machine bei Inaktivität

Sie können die CPU-Zyklen einer inaktiven Java Virtual Machine (JVM) über die Option **-XsamplingExpirationTime** reduzieren, mit der der JIT-Sampling-Thread inaktiviert wird.

Über den JIT-Sampling-Thread wird die aktive Java-Anwendung auf häufig verwendete Methoden überprüft. Die Hauptspeicherbelegung und die Prozessorauslastung des Sampling-Threads sind geringfügig und die durchgeführten Stichproben werden automatisch reduziert, wenn die JVM inaktiv ist.

In einigen Fällen ist es aber vielleicht wünschenswert, dass keine CPU-Zyklen durch eine inaktive JVM verbraucht werden. Dazu müssen Sie die Option **-XsamplingExpirationTime<Zeitraum>** angeben. Dabei wird **<Zeitraum>** auf die Anzahl an Sekunden gesetzt, die der Sampling-Thread ausgeführt werden soll. Allerdings sollte diese Option mit Vorsicht gehandhabt werden, da ein inaktivierter Sampling-Thread nicht wieder aktiviert werden kann. Der Sampling-Thread sollte zumindest lange genug aktiv sein, um wichtige Optimierungen zu erkennen.

Diagnostics-Collector

Der Diagnostics-Collector erfasst die Java-Diagnosedateien zu einem Problemereignis.

Das Zusammenstellen der vom IBM Service benötigten Dateien kann die Zeit reduzieren, die für das Lösen der gemeldeten Probleme aufgewendet wird. Das Benutzerhandbuch für IBM SDK for Java 7 enthält ausführliche Informationen zur Diagnostics-Collector-Verwendung.

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Diagnostics-Collector.

Garbage-Collector-Diagnose

In diesem Abschnitt wird die Vorgehensweise zur Diagnose von Problemen bei der Garbage-Collection beschrieben.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zum Diagnostizieren von Garbage-Collector-Problemen:

- Ausführliche Protokollierung bei Garbage-Collection
- Garbage-Collection-Tracing mit **-Xtgc**

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Garbage-Collector-Diagnose.

Ergänzende Informationen zum Metronom-Garbage-Collector von IBM WebSphere Real Time for Linux finden Sie in den folgenden Abschnitten.

Fehlerbehebung für den Metronom-Garbage-Collector

Mit den Befehlszeilenoptionen können Sie die Häufigkeit der Metronom-Garbage-Collection, Ausnahmebedingungen aufgrund abnormaler Speicherbedingungen und das Metronomverhalten bei expliziten Systemaufrufen steuern.

Informationen von 'verbose:gc' verwenden:

Sie können die Option **-verbose:gc** mit der Option **-Xgc:verboseGCCycleTime=N** verwenden, um Informationen zur Metronom-Garbage-Collector-Aktivität auf der Konsole auszugeben. Nicht alle XML-Eigenschaften in der **-verbose:gc**-Ausgabe von der Standard-JVM werden erstellt oder gelten für die Ausgabe des Metronom-Garbage-Collectors.

Mit der Option **-verbose:gc** können Sie den minimalen, maximalen und durchschnittlichen freien Speicherplatz im Heapspeicher anzeigen. Dadurch können Sie den Grad der Aktivität sowie die Verwendung des Heapspeichers prüfen und anschließend die Werte bei Bedarf anpassen. Die Option **-verbose:gc** gibt Metronomstatistikdaten auf der Konsole aus.

Die Option **-Xgc:verboseGCCycleTime=N** steuert die Informationsabrufhäufigkeit. Sie ermittelt die Zeit in Millisekunden, die bis zum Erstellen eines Speicherauszugs der Zusammenfassungen verstreicht. Der Standardwert für N ist 1000 Millisekunden. Die Zykluszeit bedeutet nicht, dass ein Speicherauszug der Zusammenfassung genau zu dieser Zeit erstellt wird, sondern wenn das letzte Garbage-Collection-Ereignis stattfindet, das dieses Zeitkriterium erfüllt. Die Erfassung und Anzeige dieser Statistikdaten kann die Pausezeiten des Metronom-Garbage-Collectors erhöhen. Je kleiner N wird, desto größer können die Pausezeiten werden.

Ein Quantum ist ein einzelner Zeitraum von Metronom-Garbage-Collector-Aktivität, der für eine Anwendung eine Unterbrechung, d. h. Pausezeit, verursacht.

Beispiel für die Ausgabe von 'verbose:gc'

Geben Sie Folgendes ein:

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

Wird eine Garbage-Collection ausgelöst, tritt ein Ereignis `trigger start` gefolgt von einer Anzahl `heartbeat`-Ereignissen und einem Ereignis `trigger end` auf, wenn der Trigger abgeschlossen ist. Das folgende Beispiel zeigt einen ausgelösten Garbage-Collection-Zyklus als Ausgabe von 'verbose:gc' an:


```

| <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" intervals="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" />

```

Die folgenden Ereignistypen können auftreten:

<trigger-start ...>

Der Anfang eines Garbage-Collection-Zyklus, als die belegte Speicherkapazität den Triggerschwellenwert überschritt. Der Standardschwellenwert ist 50 % des Heapspeichers. Das Attribut `intervals` ist das Intervall zwischen dem vorherigen Ereignis `trigger end` (mit der ID -1) und diesem Ereignis `trigger start`.

<trigger-end ...>

Ein Garbage-Collection-Zyklus senkte den verwendeten Speicher erfolgreich unter den Triggerschwellenwert. Wenn ein Garbage-Collection-Zyklus beendet wird, der verwendete Speicher den Triggerschwellenwert jedoch nicht unterschreitet, wird ein neuer Garbage-Collection-Zyklus mit derselben Kontext-ID gestartet. Für jedes Ereignis `trigger start` gibt es ein entsprechendes Ereignis `trigger end` mit derselben Kontext-ID. Das Attribut `intervals` ist das Intervall zwischen dem vorherigen Ereignis `trigger start` und dem aktuellen Ereignis `trigger end`. Während dieser Zeit wird mindestens ein Garbage-Collection-Zyklus beendet, bis der verwendete Speicher den Triggerschwellenwert unterschreitet.

<gc-op id="28" type="heartbeat" ...>

Ein periodisches Ereignis, das Speicher- und Zeitinformationen zu allen Garbage-Collection-Quanten für den abgedeckten Zeitraum zusammenstellt. Ein Ereignis `heartbeat` kann nur zwischen einem übereinstimmenden Paar von `trigger start`- und `trigger end`-Ereignissen auftreten, d. h., während ein Garbage-Collection-Zyklus aktiv ist. Das Attribut `intervals` ist das Intervall zwischen dem vorherigen Ereignis `heartbeat` (mit id -1) und diesem Ereignis `heartbeat`.

| **<gc-op id="29" type="syncgc"...>**

Ein synchrones (nicht deterministisches) Garbage-Collection-Ereignis. Weitere Informationen hierzu finden Sie in „Synchrone Garbage-Collections“ auf Seite 63.

Die XML-Tags in diesem Beispiel haben die folgenden Bedeutungen:

| **<quanta ...>**

Eine Zusammenfassung der Quantumpausezeiten während des Heartbeatintervalls, einschließlich die Länge der Pausen in Millisekunden.

| **<free-mem type="heap" ...>**

Eine Zusammenfassung der freien Heapspeicherkapazität während des Heartbeatintervalls, die am Ende jeden Garbage-Collection-Quantums stichprobenmäßig ermittelt wird.

| **<classunload-info classloadersunloaded=9 classesunloaded=156 />**

Die Anzahl der Klassenladeprogramme und der Klassen, die während des Heartbeatintervalls entladen wurden.

| **<references type="weak" cleared="660 />**

Die Anzahl und der Typ der Java-Verweisobjekte, die während des Heartbeatintervalls gelöscht wurden.

Anmerkung:

- Wenn nur ein Garbage-Collection-Quantum im Intervall zwischen zwei Heartbeats aufgetreten ist, wird der freie Speicher nur am Ende dieses einen Quantums stichprobenmäßig ermittelt. Daher sind die in der Heartbeatzusammenfassung angegebenen minimalen, maximalen und durchschnittlichen Beträge gleich.
- Das Intervall zwischen zwei heartbeat-Ereignissen ist möglicherweise wesentlich größer ist als die angegebene Zykluszeit, wenn der Heapspeicher nicht voll genug ist, um eine Garbage-Collection-Aktivität auszulösen. Wenn z. B. Ihr Programm eine Garbage-Collection-Aktivität nur alle paar Sekunden erfordert, wird ein Heartbeat wahrscheinlich nur alle paar Sekunden angezeigt.
- Es ist möglich, dass das Intervall wesentlich größer ist als die angegebene Zykluszeit, weil die Garbage-Collection für einen Heapspeicher, der nicht voll genug ist, keine Arbeit ausführen muss. Wenn z. B. Ihr Programm eine Garbage-Collection-Aktivität nur alle paar Sekunden erfordert, wird ein Heartbeat wahrscheinlich nur alle paar Sekunden angezeigt.

Wenn ein Ereignis wie eine synchrone Garbage-Collection oder eine Änderung der Priorität auftritt, werden die Ereignisdetails und anstehende Ereignisse wie Heartbeats unverzüglich als Ausgabe erzeugt.

- Wenn das maximale Garbage-Collection-Quantum für einen angegebenen Zeitraum zu groß ist, können Sie die Zielauslastung mit der Option **-Xgc:targetUtilization** senken. Diese Aktion gibt dem Garbage-Collector mehr Arbeitszeit. Alternativ können Sie die Größe des Heapspeichers mit der Option **-Xmx** erhöhen. Wenn Ihre Anwendung längere Verzögerungen tolerieren kann als zurzeit aufgelistet, können Sie die Zielauslastung erhöhen bzw. die Größe des Heapspeichers senken.
- Die Ausgabe kann mit der Option **-Xverbosegclog:<Datei>** von der Konsole in eine Protokolldatei umgeleitet werden. Beispielsweise schreibt **-Xverbosegclog:out** die Ausgabe von **-verbose:gc** in die Datei *out*.
- Die in thread-priority aufgelistete Priorität ist die Priorität des zugrunde liegenden Betriebssystemthreads, keine Java-Threadpriorität.

Synchrone Garbage-Collections

Ein Eintrag wird auch in das Protokoll von **-verbose:gc** geschrieben, wenn eine synchrone (nicht deterministische) Garbage-Collection auftritt. Für dieses Ereignis gibt es drei mögliche Ursachen:

- Ein expliziter Aufruf von `System.gc()` im Code.
- Für die JVM ist kein Speicher mehr verfügbar und es wird eine synchrone Garbage-Collection durchgeführt, um eine Bedingung `OutOfMemoryError` zu vermeiden.
- Die JVM wird während einer fortlaufenden Garbage-Collection beendet. Die JVM kann die Collection nicht abbrechen. Daher schließt sie die Collection synchron ab und wird dann beendet.

Es folgt ein Beispiel für einen Eintrag `System.gc()`:

```
| <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>
```

Es folgt ein Beispiel für einen Eintrag einer synchronen Garbage-Collection als Ergebnis der Beendigung der JVM:

```
| <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>
```

Die XML-Tags und Attribute in diesem Beispiel haben die folgenden Bedeutungen:

<gc-op id="9" type="syncgc" timems="6.439" ...

Diese Zeile gibt an, dass der Ereignistyp eine synchrone Garbage-Collection ist. Das Attribut `timems` ist die Dauer der synchronen Garbage-Collection in Millisekunden.

<syncgc-info reason="..."/>

Die Ursache der synchronen Garbage-Collection.

<free-mem-delta.../>

Der freie Java-Heapspeicher vor und nach der synchronen Garbage-Collection in Byte.

<finalization .../>

Die Anzahl Objekte, die auf die Endbearbeitung warten.

<classunload-info .../>

Die Anzahl der Klassenladeprogramme und der Klassen, die während des Heartbeatintervalls entladen wurden.

<references type="weak" cleared="53" .../>

Die Anzahl und der Typ der Java-Verweisobjekte, die während des Heartbeatintervalls gelöscht wurden.

Es kann nur zu einer synchronen Garbage-Collection aufgrund von abnormalen Speicherbedingungen oder einer VM-Beendigung kommen, wenn der Garbage-Collector aktiv ist. Hierbei muss ein Ereignis trigger start vorangehen, obwohl nicht unbedingt unmittelbar. Einige heartbeat-Ereignisse treten wahrscheinlich zwischen einem Ereignis trigger start und dem Ereignis synchgc auf. Eine durch System.gc() verursachte synchrone Garbage-Collection kann jederzeit auftreten.

Alle GC-Quanten protokollieren

Einzelne GC-Quanten können durch die Aktivierung der Tracepunkte GlobalGCStart und GlobalGCEnd protokolliert werden. Diese Tracepunkte werden am Anfang und Ende der Metronom-Garbage-Collector-Aktivität einschließlich synchroner Garbage-Collections erzeugt. Die Ausgabe für diese Tracepunkte ähnelt der folgenden Ausgabe:

```
| 03:44:35.281 0x833cd00 j9mm.52 - GlobalGC start: globalcount=3
|
| 03:44:35.284 0x833cd00 j9mm.91 - GlobalGC end: workstackoverflow=0 overflowcount=0
```

Einträge wegen Fehlern aufgrund abnormaler Speicherbedingungen

Wenn für den Heapspeicher kein Speicherplatz mehr verfügbar ist, wird ein Eintrag in das Protokoll von **-verbose:gc** geschrieben, bevor die Ausnahmebedingung OutOfMemoryError ausgelöst wird. Es folgt ein Beispiel für diese Ausgabe:

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

Standardmäßig wird ein Java-Speicherauszug als Ergebnis einer Ausnahmebedingung OutOfMemoryError erzeugt. Der folgende Speicherauszug enthält Informationen zum von Ihrem Programm verwendeten Speicher .

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

Metronom-Garbage-Collector-Verhalten bei abnormalen Speicherbedingungen:

Der Metronom-Garbage-Collector löst eine uneingeschränkte, nicht deterministische Garbage-Collection aus, wenn für die JVM kein Speicher mehr verfügbar ist. Sie können das nicht deterministische Verhalten mit der Option **-Xgc:noSynchronousGCOnOOM** verhindern, durch die ein OutOfMemoryError ausgelöst wird, wenn für die JVM kein Speicher mehr verfügbar ist.

Die uneingeschränkte Standard-Garbage-Collection wird ausgeführt, bis der gesamte Garbage in einer einzelnen Operation erfasst wurde. Die erforderliche Pausezeit ist gewöhnlich viele Millisekunden länger als ein normales inkrementelles Metronomquantum.

Zugehörige Informationen

Mit '-Xverbose:gc' synchrone Garbage-Collections analysieren

Metronom-Garbage-Collector-Verhalten bei expliziten System.gc()-Aufrufen:

Läuft ein Garbage-Collection-Zyklus, schließt der Metronom-Garbage-Collector den Zyklus auf synchrone Art ab, wenn System.gc() aufgerufen wird. Läuft kein Garbage-Collection-Zyklus, wird ein voller synchroner Zyklus ausgeführt, wenn System.gc() aufgerufen wird. Mit System.gc() können Sie den Heapspeicher auf

kontrollierte Weise bereinigen. Hierbei handelt es sich um eine nicht deterministische Operation, weil sie vor ihrer Rückgabe eine vollständige Garbage-Collection ausführt.

Einige Anwendungen rufen die Software anderer Anbieter auf, die über System.gc()-Aufrufe verfügt. Für sie ist es jedoch nicht zulässig, diese nicht deterministischen Verzögerungen zu erstellen. Mit der Option **-Xdisableexplicitgc** können Sie alle System.gc()-Aufrufe inaktivieren.

Die ausführliche Garbage-Collection-Ausgabe für einen System.gc()-Aufruf weist als Ursache „system garbage collect“ auf und hat wahrscheinlich eine lange Dauer:

```
| <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>
```

Diagnose bei gemeinsam genutzten Klassen

Für die Verwendung des Modus gemeinsam genutzter Klassen ist ein Verständnis der Vorgehensweise bei der Diagnose von eventuell auftretenden Problemen hilfreich.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält nützliche Informationen zum Diagnostizieren von Problemen mit gemeinsam genutzten Klassen:

- Gemeinsam genutzte Klassen implementieren
- Umgang mit Änderungen am Bytecode zur Ausführungszeit
- Dynamische Aktualisierungen - Grundlagen
- Java-Helper-API verwenden
- Diagnosenachrichten für gemeinsam genutzte Klassen - Grundlagen
- Probleme in Zusammenhang mit gemeinsam genutzten Klassen beheben

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Diagnose bei gemeinsam genutzten Klassen.

Mit der JVMTI arbeiten

JVMTI ist eine bidirektionale Schnittstelle, die die Kommunikation zwischen der Java Virtual Machine (JVM) und einem nativen Agenten ermöglicht. Sie ersetzt die JVMDI- und die JVMPI-Schnittstelle.

Mithilfe der JVMTI können andere Anbieter Tools zur Fehlerermittlung, Profilerstellung und Überwachung für die Java Virtual Machine (JVM) entwickeln. Die Schnittstelle enthält Mechanismen, mit der der Agent der JVM mitteilen kann, welche Informationen benötigt werden. Darüber hinaus ermöglicht die Schnittstelle auch den Empfang von Benachrichtigungen. Es können jederzeit mehrere Agenten mit der JVM verbunden werden.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält ausführliche Informationen zur Verwendung von JVMTI, einschließlich einen API-Referenzabschnitt zu IBM Erweiterungen für JVMTI.

Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Mit JVM-TI arbeiten.

Diagnostic Tool Framework for Java verwenden

Das Diagnostic Tool Framework for Java (DTFJ) ist eine Java-Anwendungsprogrammierschnittstelle (API) von IBM zur Unterstützung der Erstellung von Java-Diagnosetools. DTFJ arbeitet mit Daten aus einem Systemspeicherauszug oder einem Java-Speicherauszug.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält ausführliche Informationen zur DTFJ-Verwendung. Folgen Sie diesem Link: [Diagnostic Tool Framework for Java verwenden](#)

IBM Monitoring and Diagnostic Tools for Java - Health Center verwenden

IBM Monitoring and Diagnostic Tools for Java - Health Center ist ein Diagnosetool zur Überwachung des Status einer aktiven Java Virtual Machine (JVM).

Informationen zu IBM Monitoring and Diagnostic Tools for Java - Health Center sind in developerWorks und in einem Infocenter verfügbar.

Kapitel 10. Referenz

In diesen Themen werden die Optionen und Klassenbibliotheken aufgelistet, die mit WebSphere Real Time for Linux verwendet werden können.

Befehlszeilenoptionen

Sie können beim Starten von Java Optionen in der Befehlszeile angeben. Standardoptionen wurden für die beste allgemeine Verwendung ausgewählt.

Java-Optionen und Systemeigenschaften angeben

Sie können Java-Eigenschaften und Systemeigenschaften auf drei verschiedene Arten angeben.

Informationen zu diesem Vorgang

Sie können Java-Optionen und Systemeigenschaften auf folgende Arten angeben. Sie lauten nach Vorrang wie folgt:

1. Durch Angeben der Option oder der Eigenschaft in der Befehlszeile. Beispiel:
`java -Dmysysprop1=tcPIP -Dmysysprop2=wait -XdisableJavadump MeineJavaKlasse`
2. Durch Erstellen einer Datei, die die Optionen enthält, und Angeben dieser Datei in der Befehlszeile mit der Option **-Xoptionsfile=<Dateiname>**.

Geben Sie in der Optionsdatei jede Option in einer neuen Zeile an. Sie können das Zeichen '\' als Fortsetzungszeichen verwenden, wenn sich eine Option über mehrere Zeilen erstrecken soll. Verwenden Sie das Zeichen '#', um Kommentarzeilen zu definieren. Sie können die Option **-classpath** nicht in einer Optionsdatei angeben. Im Folgenden finden Sie ein Beispiel für eine Optionsdatei:

```
#My options file
-X<Option1>
-X<Option2>=\
<Wert1>,\
<Wert2>
-D<sysprop1>=<Wert1>
```

3. Durch Erstellen einer Umgebungsvariable mit der Bezeichnung **IBM_JAVA_OPTIONS**, die die Optionen enthält. Beispiel:

```
export IBM_JAVA_OPTIONS="-Dmysysprop1=tcPIP -Dmysysprop2=wait -XdisableJavadump"
```

Die letzte in der Befehlszeile angegebene Option hat Vorrang vor der ersten Option. Wenn Sie z. B. die Optionen **-Xint -Xjit meineKlasse** angeben, hat die Option **-Xjit** Vorrang vor **-Xint**.

Systemeigenschaften

Systemeigenschaften sind für Anwendungen verfügbar und liefern Informationen zur Laufzeitumgebung.

com.ibm.jvm.realtime

Mit dieser Eigenschaft können Java-Anwendungen ermitteln, ob Sie in einer WebSphere Real Time for Linux-Umgebung ausgeführt werden.

Wenn Ihre Anwendung in der IBM WebSphere Real Time for RT Linux-Laufzeitumgebung ausgeführt wird und mit der Option **-Xrealtime** gestartet wurde, hat die Eigenschaft **com.ibm.jvm.realtime** den Wert „hard“.

Wenn Ihre Anwendung in der IBM WebSphere Real Time for RT Linux-Laufzeitumgebung ausgeführt wird, allerdings nicht mit der Option **-Xrealttime** gestartet wurde, ist die Eigenschaft **com.ibm.jvm.realttime** nicht festgelegt.

Wenn Ihre Anwendung in der IBM WebSphere Real Time-Laufzeitumgebung ausgeführt wird, hat die Eigenschaft **com.ibm.jvm.realttime** den Wert „soft“.

Standardoptionen

Die Definitionen für die Standardoptionen.

-agentlib:*<Bibliotheksname>*[=*<Optionen>*]

Lädt die native Agentenbibliothek *<Bibliotheksname>*; z. B.

-agentlib:hprof. Geben Sie **-agentlib:jwp=help** und

-agentlib:hprof=help in der Befehlszeile an, um weitere Informationen zu erhalten.

-agentpath:*Bibliotheksname*[=*<Optionen>*]

Lädt die native Agentenbibliothek mit dem vollständigen Pfadnamen.

-assert Zeigt Hilfe für Optionen für 'assert' an.

-cp oder **-classpath** *<Verzeichnisse und ZIP- oder JAR-Dateien getrennt durch :>*

Legt den Suchpfad für Anwendungsklassen und -ressourcen fest. Wenn **-classpath** und **-cp** nicht verwendet werden und **CLASSPATH** nicht definiert wurde, ist der Benutzerklassenpfad standardmäßig das aktuelle Verzeichnis (.).

-D*<property_name>* [= *<Wert>*]

Definiert eine Systemeigenschaft.

-help oder **-?**

Gibt einen Verwendungshinweis aus.

-javaagent:*<JAR-Pfad>*[=*<Optionen>*]

Lädt einen Agenten der Programmiersprache Java. Weitere Informationen finden Sie in der Dokumentation zur API `java.lang.instrument`.

-jre-restrict-search

Schließt private Benutzer-JREs in die Versionssuche ein.

-no-jre-restrict-search

Nimmt private Benutzer-JREs in die Versionssuche auf.

-showversion

Zeigt die Produktversion an und setzt den Vorgang fort.

-verbose:*[class,gc,dynload,sizes,stack,jni]*

Aktiviert die ausführliche Ausgabe.

-verbose:class

Schreibt für jede geladene Klasse einen Eintrag in 'stderr'.

-verbose:gc

Siehe „Informationen von 'verbose:gc' verwenden“ auf Seite 60.

-verbose:dynload

Stellt ausführliche Informationen bereit, während die einzelnen Klassen von der JVM geladen werden:

- Klassename und -paket
- Bei Klassendateien, die sich in einer .jar-Datei befanden, der Name und der Verzeichnispfad der .jar-Datei

- Details zur Klassengröße und der zum Laden der Klasse benötigten Zeit

Die Daten werden in 'stderr' ausgegeben. Im Folgenden finden Sie eine Beispielausgabe:

```
<Loaded java/lang/String from /myjdk/sdk/jre/lib/i386/softrealtime/jc1SC160/vm.jar>
<Class size 17258; ROM size 21080; debug size 0>
<Read time 27368 usec; Load time 782 usec; Translate time 927 usec>
```

Anmerkung: Aus dem Cache für gemeinsam genutzte Klassen geladene Klassen werden in der Ausgabe von **-verbose:dynload** nicht angezeigt. Verwenden Sie **-verbose:class**, um Informationen zu diesen Klassen zu erhalten.

-verbose:sizes

Schreibt Informationen in 'stderr', die die für die Stacks und Heapspeicher in der JVM verwendete Speicherkapazität beschreiben.

-verbose:stack

Schreibt Informationen in 'stderr', die die Java- und C-Stack-Belegung beschreiben.

-verbose:jni

Schreibt Informationen in 'stderr', die die von der Anwendung und der JVM aufgerufenen JNI-Services beschreiben.

-version

Gibt Versionsinformationen für den Nicht-Echtzeitmodus aus.

-version:<Wert>

Erfordert, dass die angegebene Version ausgeführt wird.

-X Zeigt Hilfe für vom Standard abweichende Optionen an.

Vom Standard abweichende Optionen

Optionen mit dem Präfix **-X** weichen vom Standard ab und können jederzeit geändert werden.

Das Benutzerhandbuch für IBM SDK for Java 7 enthält ausführliche Informationen zu vom Standard abweichenden Optionen. Sie finden diese Informationen an folgender Stelle: IBM SDK for Java 7 - Befehlszeilenoptionen.

Ergänzende Informationen für IBM WebSphere Real Time for Linux finden Sie in den folgenden Abschnitten.

Optionen für den Metronom-Garbage-Collector

Die Definitionen der Optionen für den Metronom-Garbage-Collector.

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM

Die Garbage-Collection tritt auf, wenn im Heapspeicher kein Speicherplatz mehr verfügbar ist. Ist im Heapspeicher kein Speicherplatz mehr verfügbar, stoppt die Verwendung von **-Xgc:synchronousGCOnOOM** Ihre Anwendung, während die Garbage-Collection nicht verwendete Objekte entfernt. Ist erneut kein Speicherplatz mehr verfügbar, reduzieren Sie die Zielauslastung, um der Garbage-Collection mehr Zeit zum Abschließen zu geben. Wird **-Xgc:nosynchronousGCOnOOM** angegeben und ist der Heapspeicher voll, wird Ihre Anwendung gestoppt und setzt eine Nachricht zu einem Fehler aufgrund abnormaler Speicherbedingungen ab. Der Standardparameter ist **-Xgc:synchronousGCOnOOM**.

-Xnoclassgc

Inaktiviert die Garbage-Collection für Klassen. Diese Option inaktiviert die Garbage-Collection für Speicher, der Java-Klassen zugeordnet ist, die nicht mehr von der JVM verwendet werden sollen. Das Standardverhalten ist **-Xnoclassgc**.

-Xgc:targetPauseTime=N

Legt die Garbage-Collection-Pausezeit fest. Dabei ist *N* die Zeit in Millisekunden. Wird diese Option angegeben, wird die GC mit Pausen ausgeführt, die den angegebenen Wert nicht überschreiten. Wird diese Option nicht angegeben, wird die Standardpausezeit auf 3 Millisekunden gesetzt. Beispielsweise pausiert die GC bei der Ausführung mit **-Xgc:targetPauseTime=20** im Verlauf von GC-Operationen nicht länger als 20 Millisekunden.

-Xgc:targetUtilization=N

Setzt die Anwendungsauslastung auf *N* %. Der Garbage-Collector versucht, höchstens (100-*N*) % jeden Zeitintervalls zu verwenden. Angemessene Werte liegen im Bereich zwischen 50-80 %. Anwendungen mit niedrigen Zuordnungsraten können mit 90 % ausgeführt werden. Der Standardwert beträgt 70 %.

Das folgende Beispiel zeigt, dass die maximale Größe des Heapspeichers 30 MB beträgt. Der Garbage-Collector versucht, 25 % jeden Zeitintervalls zu verwenden, weil die Zielauslastung für die Anwendung 75 % ist.

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

-Xgc:threads=N

Gibt die Anzahl auszuführender GC-Threads an. Der Standardwert ist die Anzahl Prozessorkerne, die für den Prozess verfügbar sind. Der Maximalwert, den Sie angeben können, ist die Anzahl Prozessoren, die dem Betriebssystem zur Verfügung stehen.

-Xgc:verboseGCCycleTime=N

N ist die Zeit in Millisekunden, die bis zum Erstellen eines Speicherauszugs der Übersichtsdaten verstreichen soll.

Anmerkung: Die Zykluszeit bedeutet nicht, dass ein Speicherauszug der Übersichtsdaten genau zu dieser Zeit erstellt wird, sondern wenn das letzte Garbage-Collection-Ereignis stattfindet, das dieses Zeitkriterium erfüllt.

-Xmx<Größe>

Gibt die Java-Heapspeichergröße an. Im Gegensatz zu anderen Garbage-Collection-Strategien unterstützt die Echtzeit-Metronom-Garbage-Collection nicht die Heapspeichererweiterung. Sie können nicht zwischen einer Anfangsgröße des Heapspeichers und einer Maximalgröße des Heapspeichers wählen. Sie können nur die Maximalgröße des Heapspeichers angeben.

Standardeinstellungen für die JVM

Die Standardeinstellungen gelten für die Echtzeit-JVM, wenn an der Umgebung, in der die JVM ausgeführt wird, keine Änderungen vorgenommen werden. Allgemeine Einstellungen werden zu Referenzzwecken angezeigt.

Die Standardeinstellungen können über Umgebungsvariablen oder Befehlszeilenparameter beim JVM-Start geändert werden. Die Tabelle zeigt einige der allgemeinen JVM-Einstellungen an. Die letzte Spalte gibt an, wie Sie das Verhalten ändern können. Dabei gelten die folgenden Schlüssel:

- **U**: nur von der Umgebungsvariable gesteuerte Einstellung
- **B**: nur vom Befehlszeilenparameter gesteuerte Einstellung
- **UB**: von der Umgebungsvariable und vom Befehlszeilenparameter gesteuerte Einstellung, wobei der Befehlszeilenparameter Vorrang hat

Die Informationen diesen als Kurzübersicht und sind nicht umfassend.

JVM-Einstellung	Standard-einstellung	Einstellung beeinflusst durch:
Java-Speicherauszüge	aktiviert	UB
Java-Speicherauszüge bei nicht genügend Speicherkapazität	aktiviert	UB
Heapspeicherauszüge	inaktiviert	UB
Heapspeicherauszüge bei nicht genügend Speicherkapazität	aktiviert	UB
Systemspeicherauszüge	aktiviert	UB
Wo Speicherauszugsdateien erstellt werden	aktuelles Verzeichnis	UB
Ausgabe über 'verbose'	inaktiviert	B
Suche im Bootklassenpfad	inaktiviert	B
JNI-Prüfungen	inaktiviert	B
Fernes Debugging	inaktiviert	B
Genauere Übereinstimmungsprüfungen	inaktiviert	B
Schnellstart	inaktiviert	B
Ferner Debuginformationsserver	inaktiviert	B
Reduzierte Signalisierung	inaktiviert	B
Verkettung von Signalhandlern	aktiviert	B
Klassenpfad	nicht festgelegt	UB
Gemeinsame Nutzung von Klassendaten	inaktiviert	B
Unterstützung von Eingabehilfen	aktiviert	U
JIT-Compiler	aktiviert	UB
AOT-Compiler (AOT wird von JVM nur verwendet, wenn auch gemeinsam genutzte Klassen aktiviert sind)	aktiviert	B
JIT-Debugoptionen	inaktiviert	B
Java2D-Schriftarten mit algorithmischer Fettschrift in maximaler Größe	14 Punkte	U
Verwendung von wiedergegebenen Bitmapdateien in skalierbaren Java2D-Schriftarten	aktiviert	U
Rastern von Java2D-FreeType-Schriftarten	aktiviert	U
Verwendung von Java2D-AWT-Schriftarten	inaktiviert	U
Länderspezifische Standardeinstellung	keine	U
Wartezeit vor Start des Plug-in	null	U
Temporäres Verzeichnis	/tmp	U
Umleitung des Plug-ins	keine	U
IM-Switching	inaktiviert	U

JVM-Einstellung	Standard-einstellung	Einstellung beeinflusst durch:
IM-Modifikatoren	inaktiviert	U
Threading-Modell	nicht zutreffend	U
Anfängliche Stackgröße für 32-Bit-Java-Threads. Verwenden Sie: -Xiss<Größe>	2 KB	B
Maximale Stackgröße für 32-Bit-Java-Threads. Verwenden Sie: -Xss<Größe>	256 KB	B
Stackgröße für 32-Bit-Betriebssystemthreads. Verwenden Sie -Xmso<Größe>	256 KB	B
Anfangsgröße des Heapspeichers. Verwenden Sie -Xms<Größe>	64 MB	B
Maximale Größe des Java-Heapspeichers. Verwenden Sie -Xmx<Größe>	Die Hälfte der verfügbaren Hauptspeichergöße, Minimum 16 MB, Maximum 512 MB	B
Nutzung des Sollzeitintervalls für eine Anwendung. Der Garbage-Collector versucht, den Rest zu verwenden. Verwenden Sie -Xgc:targetUtilization=<Prozentsatz> .	70 %	B
Die Anzahl der auszuführenden Garbage-Collector-Threads. Verwenden Sie -Xgc:threads=<Wert>	Die Anzahl Prozessorkerne, die für den Prozess verfügbar sind.	B
Maximale Speicherkapazität, die Bereichsspeichern im Modus -Xrealtime zugeordnet werden kann. Verwenden Sie -Xgc:scopedMemoryMaximumSize=<Größe> .	8 MB	B
Größe des Speicherbereichs für Objekte mit unbeschränkter Lebensdauer im Modus -Xrealtime . Verwenden Sie -Xgc:immortalMemorySize=<Größe>	16 MB	B

Anmerkung: Der „verfügbare Hauptspeicher“ ist entweder die Kapazität des Real-speichers (physischen Speichers) oder der Wert von **RLIMIT_AS**, wobei der niedrigere Wert ausschlaggebend ist.

Kapitel 11. Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden. Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in dieser Dokumentation beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieser Dokumentation ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

- JIMMAIL@uk.ibm.com [Ansprechpartner im Hursley Java Technology Center (JTC)]

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesem Dokument beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Marken

IBM, das IBM Logo und [ibm.com](http://www.ibm.com) sind Marken oder eingetragene Marken der IBM Corporation in den USA und/oder anderen Ländern. Sind diese und weitere Markennamen von IBM bei ihrem ersten Vorkommen in diesen Informationen mit einem Markensymbol (® oder ™) gekennzeichnet, bedeutet dies, dass IBM zum Zeitpunkt der Veröffentlichung dieser Informationen Inhaber der eingetragenen Marken oder der Common-Law-Marken (common law trademarks) in den USA war. Diese Marken können auch eingetragene Marken oder Common-Law-Marken in anderen Ländern sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information" unter <http://www.ibm.com/legal/copytrade.shtml>.

Adobe, das Adobe-Logo, PostScript und das PostScript-Logo sind Marken oder eingetragene Marken der Adobe Systems Incorporated in den USA und/oder anderen Ländern.

Intel und Itanium sind Marken der Intel Corporation oder ihrer Tochtergesellschaften in den USA oder anderen Ländern.

Linux ist eine Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Weitere Unternehmens-, Produkt- oder Servicenamen können Marken anderer Hersteller sein.

Index

Sonderzeichen

-? 68
-agentlib: 68
-agentpath: 68
-assert 68
-classpath 68
-cp 68
-D 68
-help 68
-javaagent: 68
-jre-restrict-search 68
-no-jre-restrict-search 68
-showversion 68
-verbose: 68
-verbose:gc, Option 60
-version: 68
-X 68
-Xdebug 6
-Xgc:immortalMemorySize 69
-Xgc:nosynchronousGConOOM 69
-Xgc:noSynchronousGConOOM, Option 64
-Xgc:scopedMemoryMaximumSize 69
-Xgc:synchronousGConOOM 69
-Xgc:synchronousGConOOM, Option 64
-Xgc:targetUtilization 69
-Xgc:threads 69
-Xgc:verboseGCCycleTime=N 69
-Xgc:verboseGCCycleTime=N, Option 60
-Xmx 36, 69
-Xnojit 6
-Xshareclasses 6
-XsynchronousGConOOM 36

A

Abstürze
 Linux 31
Alarmthread
 Metronom-Garbage-Collector 3
Anwendungen ausführen 17
Anwendungen entwickeln 23
Anwendungen mit kurzer Laufzeit
 JIT 59
Anzeigefunktion für Speicherauszüge 51
 Diagnosetools verwenden 51
AOT
 inaktivieren 53
Arbeitsbasierte Erfassung 3

B

Bekannte Einschränkungen 32

C

CLASSPATH
 festlegen 13

Collection-Threads
 Metronom-Garbage-Collector 3

D

Debugging von Leistungsproblemen 32
Deinstallation 14
 InstallAnywhere 14
Diagnosetools verwenden 39
 Diagnostics-Collector 59
 DTFJ 66
Diagnostics-Collector 59
DTFJ 66

E

Echtzeit-Garbage-Collection 3, 17
Einführung 1
Einschränkungen
 Metronom 22
Einstellungen, Standardwerte (JVM) 70
Ereignisse
 Speicherauszugsagenten 40
Ermitteln der Methode, die das Problem verursacht (JIT) 56

F

Fehlerbehebung
 Metronom 60
Fehlerbehebung und Unterstützung 29
Fehlerbestimmung 29

G

Garbage-Collection
 Echtzeit 3, 17
 Metronom 3, 17
Garbage-Collector-Diagnose 60
 Diagnosetools verwenden 60
Gemeinsam genutzte Klassen
 Diagnose 65
Gemeinsame Nutzung von Klassendaten 26
Gezieltes Inaktivieren des JIT-Compilers 54

H

Health Center 66
 Diagnosetools verwenden 66
Heapspeicherauszug 48
 Diagnosetools verwenden 48
 Textformat (klassisches Format) der
 Heapspeicherauszugsdatei 49

I

Inaktivieren des AOT-Compilers 53
Inaktivieren des JIT-Compilers 53
InstallAnywhere 14
Installation 7

J

Java-Speicherauszug 43
 Diagnosetools verwenden 43
 Speichermanagement 43
 Threads und Stack-Trace (THREADS) 46
JIT 53
 Anwendungen mit kurzer Laufzeit 59
 Diagnosetools verwenden 53
 gezielt inaktivieren 54
 inaktiv 59
 inaktivieren 53
 Kompilierfehler identifizieren 58
 Methode ermitteln, die das Problem verursacht 56
JVMTI 65
 Diagnosetools verwenden 65

K

Kerndateien 29
Klassendatensätze in einem Heapspeicherauszug 50
Klassentladung
 Metronom 3
Klassisches Format (Textformat) der Heapspeicherauszugsdatei
 Heapspeicherauszüge 49
Kompilierfehler, JIT 58
Konzepte 3
Kopfsatz in einem Heapspeicherauszug 49

L

Linux
 Abstürze diagnostizieren 31
 bekannte Einschränkungen 32
 Debugging-Verfahren 30
 Fehlerbestimmung 29
 Debugging von Leistungsproblemen 32
 Umgebung einrichten und prüfen
 Kerndateien 29

M

Methode, die das Problem verursacht (JIT) 56
Metronom
 Einschränkungen 22

Metronom (*Forts.*)
 Prozessorauslastung steuern 17, 22
 zeitbasierte Erfassung 3
Metronom-Garbage-Collection 3, 17
Metronom-Garbage-Collector
 Alarmthread 3
 Collection-Threads 3
Metronomklassenentladung 3
Musteranwendung 23

N

NLS
 Fehlerbestimmung 34

O

Objektdatensätze in einem Heapspeicher-
auszug 49
Optionen
 -verbose:gc 60
 -Xgc:immortalMemorySize 69
 -Xgc:nosynchronousGConOOM 69
 -Xgc:noSynchronousGConOOM 64
 -Xgc:scopedMemoryMaximumSi-
 ze 69
 -Xgc:synchronousGConOOM 64, 69
 -Xgc:targetUtilization 69
 -Xgc:threads 69
 -Xgc:verboseGCCycleTime=N 60, 69
 -Xmx 69
ORB
 Debugging 35
OutOfMemoryError 36, 64

P

Paketierung 7
PATH
 festlegen 12
Planung 5
Prozessorauslastung steuern 17, 22

R

Referenz 67

S

Sicherheit 27
Speicher für Objekte mit beschränkter Le-
bensdauer 3
Speicher für Objekte mit unbeschränkter
Lebensdauer 3
Speicherauszugsagenten
 Ereignisse 40
 Filter 42
 verwenden 40
Speichermanagement, Java-Speicheraus-
zug 43
Speicherverwaltung verstehen 38
Standardeinstellungen, JVM 70

T

Textformat (klassisches Format) der He-
apspeicherauszugsdatei
 Heapspeicherauszüge 49
Threads und Stack-Trace (THREADS) 46
Traceerstellung 52
 Diagnosetools verwenden 52
Trailerdatensatz 1 in einem Heapspeicher-
auszug 50
Trailerdatensatz 2 in einem Heapspeicher-
auszug 51
Typkennungen 51

U

Unterstützte Umgebungen 5

V

Verwenden, Speicherauszugsagenten 40

Z

Zeitbasierte Erfassung
 Metronom 3

