IBM WebSphere Real Time for Linux

버전 3

사용자 안내서

IBM

IBM WebSphere Real Time for Linux
버전 3

사용자 안내서

IBM

참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에 89 페이지의 『주의사항』의 정보를 읽으십시오.

제 5 판(2014년 2월)

이 사용자 안내서 개정판은 새 개정판에 별도로 명시하지 않는 한 IBM WebSphere Real Time for Linux, 버전 3 및 모든 후속 릴리스와 수정사항에 적용됩니다.

목차

그림

표

서문

이 사용자 안내서는 IBM® WebSphere® Real Time for Linux에 대한 일반적인 정보를 제공합니다.

제 1 장 소개

이 정보는 IBM WebSphere Real Time for Linux에 대해 설명합니다.

이 사용자 안내서의 새 수정사항은 변경사항 왼쪽에 수직 막대로 표시되어 있습니다.

사용자 안내서에 없는 IBM WebSphere Real Time for Linux에 대한 최신 정보는 http://www.ibm.com/support/docview.wss?uid=swg21501145에서 찾을 수 있습니다.

- 『WebSphere Real Time for Linux 개요』
- 『새로운 기능』
- 2 페이지의 『장점』

WebSphere Real Time for Linux 개요

WebSphere Real Time for Linux는 IBM J9 가상 머신(JVM)과 실시간 기능을 번들로 제공합니다.

WebSphere Real Time for Linux는 실시간 기능이 있는 IBM SDK for Java™를 확장하는 SDK(Software Development Kit)가 있는 Java Runtime Environment입니다. 정확한 응답 시간에 따르는 애플리케이션은 표준 Java 기술의 WebSphere Real Time for Linux와 함께 제공되는 실시간 기능을 이용할 수 있습니다.

기능

Real-time 애플리케이션은 절대적 속도보다 일관된 런타임을 필요로 합니다.

기존의 JVM으로 실시간 애플리케이션을 배치할 때 주요 관심 사항은 다음과 같습니다.

- 가비지 콜렉션(GC) 활동에 따른 예측 불가능한 지연(장기화될 수 있음).
- JIT(Just-In-Time) 컴파일 및 재컴파일이 발생할 때 메소드 런타임이 지연되고 실행 시간이 변경될 수 있음.
- 임의의 운영 체제 스케줄링.

WebSphere Real Time for Linux는 다음을 제공하여 이 문제를 해결합니다.

- 일시정지 시간이 매우 적은 증분식 결정적 가비지 콜렉터인 메트로놈 가비지 콜렉터

새로운 기능

이 주제에서는 IBM WebSphere Real Time for Linux의 변경사항을 소개합니다.

WebSphere Real Time for Linux V3

WebSphere Real Time for Linux V3은 IBM SDK for Java V7의 확장으로 이 릴리스에서 사용 가능한 기능에 실시간 기능을 포함시키기 위해 빌드합니다. 이전 버전의 WebSphere Real Time for Linux는 IBM SDK for Java의 이전 릴리스에 기반합니다.

IBM SDK for Java V7의 새로운 기능에 대한 더 자세한 정보는 IBM SDK for Java 7 정보 센터에 있는 새로운 기능의 내용을 참조하십시오.

이 사용자 안내서의 새 수정사항은 변경사항 왼쪽에 수직 막대로 표시되어 있습니다.

Linux 스케줄링 정책을 사용하는 java 스레드 스케줄링

Service Refresh 1부터 **SCHED_RR** 스케줄링 정책으로 일반 java 스레드를 스케줄해서 실시간 애플리케이션을 자세히 조정할 수 있습니다. 자세한 정보는 웹 사이트 6 페이지의 『스레드 스케줄링』의 내용을 참조하십시오.

메트로놈 가비지 콜렉터의 일시정지 시간 제어

기본적으로 메트로놈 가비지 콜렉터는 가비지 콜렉션 주기 사이에 3밀리초 동안 일시정지합니다. 새 명령행 옵션을 사용하여 이 값을 변경하여 일시정지 시간을 제어할 수 있습니다. 이 옵션에 대한 자세한 정보는 26 페이지의 『일시정지 시간 제어』의 내용을 참조하십시오.

압축 참조

메트로놈 가비지 콜렉터가 이제 64비트 플랫폼에서 압축된 참조는 물론 압축되지 않은 참조도 지원합니다. 그 밖의 성능 구현에 대한 정보는 35 페이지의 제 7 장 『성능』의 내용을 참조하십시오.

장점

실시간 환경의 이점은 표준 JVM보다 높은 수준의 예측 가능성으로 Java 애플리케이션을 실행하고 사용자의 Java 애플리케이션에 일관된 시간의 동작을 제공하는 것입니다. 컴파일, 가비지 콜렉션과 같은 백그라운드 활동이 지정된 시간에 발생하므로 애플리케이션 실행 시 백그라운드 활동이 예기치 않게 많아지는 것을 방지합니다.

JVM을 메트로놈 실시간 가비지 콜렉션 기술로 확장하면 이러한 이점을 얻을 수 있습니다.

내게 필요한 옵션

내게 필요한 옵션 기능은 거동이 불편하거나 시각 장애 등의 신체적 장애가 있는 사용자가 IT 제품을 사용하는 데 도움을 줍니다.

IBM은 연령 또는 능력에 상관 없이 누구나 액세스하여 사용할 수 있는 제품을 제공하고자 노력하고 있습니다.

예를 들어, 마우스 없이 키보드만을 사용하여 WebSphere Real Time for Linux를 작동할 수 있습니다.

기본 IBM SDK for Java V7의 내게 필요한 옵션에 영향을 미치는 문제에 대해 읽으려면 IBM Information Center를 참조하십시오. WebSphere Real Time for Linux의 고유 기능에 영향을 미치는 내게 필요한 옵션 문제는 없습니다.

키보드 탐색

이 제품은 표준 Microsoft Windows 탐색 키를 사용합니다.

키보드 탐색이 필요한 사용자를 위해 Swing 키 바인딩에서 Swing 애플리케이션의 유용한 키스트로크에 대한 설명을 포함하고 있습니다.

IBM 및 내게 필요한 옵션

IBM의 내게 필요한 옵션 기능 이행에 대한 자세한 정보는 IBM Human Ability and Accessibility Center를 참조하십시오.

제 2 장 IBM WebSphere Real Time for Linux 이해

이 섹션에서는 IBM WebSphere Real Time for Linux의 주요 컴포넌트를 소개합니다.

- 『메트로놈 가비지 콜렉터 소개』

메트로놈 가비지 콜렉터 소개

WebSphere Real Time for Linux에서 표준 가비지 콜렉터 대신 메트로놈 가비지 콜렉터를 사용합니다.

메트로놈 가비지 콜렉션과 표준 가비지 콜렉션의 주요 차이점은 메트로놈 가비지 콜렉션은 인터럽트 가능한 작은 단계에서 발생하고 표준 가비지 콜렉션은 가비지를 표시하고 수집하는 동안 애플리케이션을 중지시킨다는 것입니다.

예를 들면 다음과 같습니다.

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

예제에서 애플리케이션이 60밀리초마다 80% 실행되도록 지정합니다. 수집할 가비지가 있는 경우 나머지 20% 시간은 가비지 콜렉션에 사용됩니다. 메트로놈 가비지 콜렉터는 충분한 자원이 제공될 경우에 한해 이용 수준을 보장합니다. 힙의 여유 공간 *크기*가 동적으로 판별되는 임계값보다 작아지면 가비지 콜렉션이 시작됩니다.

메트로놈 가비지 콜렉션 및 클래스 로드 해제

메트로놈은 표준 JDK(Java Developer Kit)와 동일한 방식으로 클래스 로드 해제를 지원합니다. 그러나, 관련된 작업 때문에 클래스 로드를 해제하면서 가비지 콜렉션 활동을 수행하는 동안에는 일시정지 시간 이탈이 있을 수 있습니다.

메트로놈 가비지 콜렉터 스레드

메트로놈 가비지 콜렉터는 단일 알람 스레드와 많은 콜렉션(GC) 스레드라는 두 가지 유형의 스레드로 구성됩니다. 기본적으로 GC는 운영 체제에서 사용 가능한 각 논리적 활성 프로세서마다 한 개의 스레드를 사용합니다. 따라서 GC 주기 동안 가장 효율적인 병렬 처리를 사용할 수 있습니다. GC 주기는 GC를 트리거한 후 가비지 해제를 완료할 때까지의 시간을 의미합니다. Java 힙 *크기*에 따라 전체 GC 주기의 경과 시간은 몇 초입니다. GC 주기는 일반적으로 수 백개의 GC 퀀텀을 포함합니다. 이 퀀텀은 애플리케이션 코드에 대한 짧은 일시정지로, 대개 3밀리초입니다. 주기 및 퀀텀의 요약 보고

서를 가져오려면 **-verbose:gc**를 사용하십시오. 자세한 정보는 73 페이지의 『verbose:gc 정보 사용』의 내용을 참조하십시오. **-Xgcthreads** 옵션을 사용하여 JVM의 GC 스레드 수를 설정할 수 있습니다.

-Xgcthreads를 기본값보다 높여도 아무런 이점이 없습니다. **-Xgcthreads**를 줄이면 GC 주기 동안 전체 CPU 로드를 줄일 수 있지만 GC 주기가 길어집니다.

참고: GC 퀀텀 지속 기간 대상은 3밀리초에 변함없이 유지됩니다.

JVM에 대한 알람 스레드 수는 변경할 수 없습니다.

메트로놈 가비지 콜렉터는 힙 메모리에 충분한 여유 공간이 있는지 확인하기 위해 JVM을 정기적으로 검사합니다. 여유 공간의 *크기*가 한계보다 적으면 메트로놈 가비지 콜렉터가 JVM을 트리거하여 가비지 콜렉션을 시작합니다.

알람 스레드

단일 알람 스레드는 최소 자원 사용을 보장합니다. 정기적인 간격으로 『활성화』되고 다음을 검사합니다.

- 힙 메모리의 여유 공간 *크기*
- 가비지 콜렉션이 현재 진행 중인지 여부

여유 공간이 충분하지 않고 가비지 콜렉션이 진행 중이지 않으면 알람 스레드가 가비지 콜렉션을 시작하도록 콜렉션 스레드를 트리거합니다. JVM을 검사하는 다음 스케줄 시간까지 알람 스레드는 아무런 작업을 수행하지 않습니다.

콜렉션 스레드

콜렉션 스레드는 가비지 콜렉션을 수행합니다.

가비지 콜렉션 주기가 완료된 후 메트로놈 가비지 콜렉터가 여유 힙 공간의 *크기*를 확인합니다. 여유 힙 공간이 계속 불충분한 경우 동일한 트리거 ID를 사용하는 다른 가비지 콜렉션 주기를 시작합니다. 여유 힙 공간이 충분하면 트리거가 종료하고 가비지 콜렉션 스레드가 중지됩니다. 알람 스레드가 여유 힙 공간을 계속 모니터링하고 필요하면 다른 가비지 콜렉션 주기를 트리거합니다.

메트로놈 가비지 콜렉터를 사용하는데 관한 자세한 정보는 25 페이지의 『메트로놈 가비지 콜렉터 사용』의 내용을 참조하십시오.

스레드 스케줄링

Linux 스케줄링 정책을 사용하여 및 일반 Java 스레드의 실시간 애플리케이션을 조정할 수 있습니다.

WebSphere Real Time for Linux에서는 SCHED_RR 스케줄링 정책으로 일반 Java 스레드를 실행할 수 있습니다. SCHED_RR 정책을 사용하면 애플리케이션을 보다 정교하여 제어하여 Java 스레드의 실시간 성능을 높일 수 있습니다. JVM은 SCHED_RR 정책으로 Java를 시작할 때 기본 스레드의 우선순위 및 정책을 발견합니다. JVM은 이

에 맞게 우선순위 및 정책 맵핑을 변경합니다. 일반 Java 스레드 우선순위 변경 및 정책에 대한 더 자세한 정보는 21 페이지의 『스레드 스케줄링 및 디스패치』의 내용을 참조하십시오.

Linux 스케줄링 정책은 다음 내용을 포함합니다.

SCHED_OTHER

대부분의 스레드에서 사용하는 기본 유니버셜 시간 공유 스케줄링 정책입니다. 이 스레드는 우선순위 0으로 지정되어야 합니다.

SCHED_OTHER는 시간 분할을 사용합니다. 즉, 제한된 기간 동안 각 스레드가 실행되고 이후 다음 스레드를 실행할 수 있습니다.

SCHED_FIFO

0보다 큰 우선순위로만 사용할 수 있습니다. SCHED_FIFO 스레드를 사용할 수 있는 경우 해당 스레드는 일반 SCHED_OTHER 스레드보다 높은 우선순위를 갖습니다.

더 높은 우선순위를 갖는 SCHED_FIFO 스레드를 사용할 수 있는 경우 해당 스레드는 더 낮은 우선순위를 갖는 기존 SCHED_FIFO 스레드보다 높은 우선순위를 갖습니다. 그러면 이 스레드가 해당 우선순위로 큐 맨 위에 보관됩니다.

분할할 시간이 없습니다.

참고: SCHED_FIFO는 WebSphere Real Time for Linux에서 사용되지 않습니다.

SCHED_RR

SCHED_FIFO의 개선된 기능입니다. 차이점은 각 스레드를 제한된 기간 동안에만 실행할 수 있다는 것입니다. 스레드가 해당 시간을 초과하면 우선순위를 위해 목록으로 돌아갑니다. WebSphere Real Time for Linux V3는 SCHED_RR을 사용할 수 있습니다.

이 Linux 스케줄링 정책에 대한 더 자세한 내용은 **sched_setscheduler**의 **man** 페이지를 참조하십시오.

WebSphere Real Time for Linux를 포함한 Linux 스케줄링 정책 사용에 대한 더 자세한 정보는 see 21 페이지의 『스레드 스케줄링 및 디스패치』의 내용을 참조하십시오.

제 3 장 계획

WebSphere Real Time for Linux 설치 전에 이 섹션을 읽으십시오.

•

• 『지원되는 환경』

•

• 10 페이지의 『고려사항』

마이그레이션

수정 없이 WebSphere Real Time for Linux의 표준 Java 애플리케이션을 실행할 수 있습니다.

지원되는 환경

IBM WebSphere Real Time for Linux는 특정 하드웨어 플랫폼 및 운영 체제에서 지원됩니다.

IBM WebSphere Real Time for Linux

지원되는 아키텍처는 다음과 같습니다.

- Intel 아키텍처, 32비트(IA-32)
  - Pentium 4
  - Pentium Xeon
  - Pentium M
  - Pentium D 및 동등 항목
- AMD64/EM64T
- IBM POWER® 32
- IBM POWER 64

참고: Pentium 3 하드웨어는 더 이상 지원되지 않습니다.

다음 운영 체제가 지원됩니다.

표 1. 테스트된 Linux 환경

하드웨어	**IA-32 32비트**	**AMD64/EM64T 64비트**	
SDK 주소 공간	**32비트**	**32비트**	**64비트**
RHEL 5 업데이트 7	예	예	예

표 1. 테스트된 Linux 환경 (계속)

하드웨어	IA-32 32비트	AMD64/EM64T 64비트	
RHEL 6 업데이트 1	예	예	예
SLES 11 서비스 팩 2	예	예	예
Ubuntu 8.04	예	예	예
Ubuntu 10.04	예	예	예

참고: SLES 9, SLES 10 및 RHEL 4는 지원되지 않습니다.

고려사항

WebSphere Real Time for Linux를 사용할 때 많은 요소를 알고 있어야 합니다.

- 가능하면, 동일한 시스템에서 둘 이상의 실시간 JVM을 실행하지 마십시오. 여러 개의 가비지 콜렉터가 있을 수 있기 때문입니다. 각 JVM은 서로의 메모리 영역을 알지 못합니다. 이 경우 JVM 전반에서 GC 사이클 및 일시정지 시간을 조정할 수 없으며 이는 한 개의 JVM이 또 다른 JVM의 GC 성능에 부정적인 영향을 줄 수 있음을 의미합니다. 여러 JVM을 사용해야 하는 경우, **taskset** 명령을 사용하여 각 JVM이 프로세서의 특정 서브세트로 바인드되는지 확인하십시오.
- 이전 WebSphere Real Time for Linux 릴리스에서 사전 컴파일된 코드 및 클래스를 저장하도록 사용한 공유 캐시는 이 WebSphere Real Time for Linux 릴리스에서 사용한 캐시와 호환되지 않습니다. 이전 캐시의 컨텐츠를 다시 생성해야 합니다.
- 공유 클래스 캐시를 사용할 때는 캐시 이름이 53자를 초과하지 않아야 합니다.

제 4 장 WebSphere Real Time for Linux 설치

제품을 설치하려면 다음 단계를 따르십시오.

- 『설치 파일』
- 『InstallAnywhere 패키지에서 설치』
  - 13 페이지의 『수동 설치 완료』
  - 14 페이지의 『자동 설치 완료』
  - 15 페이지의 『알려진 문제와 제한사항』
- 17 페이지의 『경로 설정』
- 17 페이지의 『클래스 경로 설정』
- 18 페이지의 『설치 테스트』
- 19 페이지의 『WebSphere Real Time for Linux 설치 제거』

설치 파일

이 설치 파일이 필요합니다.

IBM WebSphere Real Time for Linux는 두 가지 유형의 InstallAnywhere 패키지로 제공됩니다.

설치 가능 패키지

설치 가능 패키지는 시스템을 구성합니다. 예를 들어, 프로그램이 환경 변수를 설정할 수 있습니다.

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

아카이브 패키지

이 패키지는 파일을 시스템에 추출하지만 구성은 수행하지 않습니다.

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

참고: <arch>는 사용자의 플랫폼 아키텍처입니다. x86_32 또는 x86_64.

InstallAnywhere 패키지에서 설치

이 패키지는 설치 옵션을 안내하는 대화식 프로그램을 제공합니다. 그래픽 사용자 인터페이스 또는 시스템 콘솔에서 프로그램을 실행할 수 있습니다.

시작하기 전에

시스템에 다음 공유 라이브러리가 둘 다 있어야 합니다.

- GNU C 라이브러리 V2.3(glibc)
- `libstdc++.so.5`

`libstdc++.so.5` 공유 라이브러리가 없으면 설치할 때 다음 오류를 포함한 Java 코어 덤프가 표시될 수 있습니다.

```
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
```

설치 가능한 패키지를 설치하는 경우 rpm-build 도구가 시스템에 설치되어 있어야 하며, 그렇지 않으면 설치 프로그램이 새 패키지를 RPM 데이터베이스에서 등록할 수 없습니다. rpm-build 도구가 설치되어 있는지 알아보려면 다음 명령을 입력하십시오.

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

이 태스크 정보

InstallAnywhere 패키지는 `.bin` 파일 확장자를 가집니다.

두 가지 유형의 패키지가 있습니다.

설치 가능

이러한 패키지를 설치하면 예를 들면 환경 변수를 설정하여 시스템도 구성합니다.

아카이브

이러한 패키지를 설치하면 파일을 시스템에 추출하지만, 구성은 수행하지 않습니다.

프로시저

- 패키지를 대화식으로 설치하려면, 수동 설치를 완료하십시오.
- 추가 사용자 상호작용 없이 패키지를 설치하려면, 자동 설치를 완료하십시오. 많은 시스템을 설치하려는 경우 이 옵션을 선택할 수 있습니다.
- 설치 프로세스가 완료되면 경로 및 클래스 경로 환경 변수 설정과 같은 섹션에서 구성 단계를 따르십시오.

결과

제품이 설치됩니다.

참고: 예를 들면 Ctrl+C를 눌러 설치 프로세스를 인터럽트하지 마십시오. 프로세스를 인터럽트하면 제품을 다시 설치해야 할 수 있습니다. 자세한 정보는 웹 사이트 15 페이지의 『인터럽트된 설치』의 내용을 참조하십시오.

설치 가능한 메시지를 사용하는 경우 문제점이 발견되었음을 알리는 메시지가 표시될 수도 있습니다. 아카이브 패키지의 설치에서는 메시지가 생성되지 않습니다. 설치 가능한 패키지를 사용할 때 표시될 수 있는 메시지 중 일부를 다음 목록에서 보여줍니다.

The installer cannot run on your configuration. It will now quit.

이 오류 메시지는 설치 프로세스를 실행할 수 있는 권한이 사용자 ID에 부여되지 않은 경우 발생합니다. 계속할 수 없으므로 설치 프로그램이 종료됩니다. 문제점을 수정하려면 루트 권한이 있는 사용자 ID로 설치를 다시 시작하십시오.

An RPM package is already installed. Uninstall the package before proceeding.

이 메시지는 RPM 패키지가 이미 설치되어 있음을 표시합니다. 계속할 수 없으므로 설치 프로그램이 종료됩니다. 문제점을 수정하려면 진행하기 전에 RPM 패키지를 설치 제거하십시오.

수동 설치 완료

InstallAnywhere 패키지에서 대화식으로 제품을 설치합니다.

시작하기 전에

설치 프로세스를 시작하기 전에 다음 조건을 확인하십시오.

- 이전에 RPM 패키지에서 WebSphere Real Time for Linux를 설치한 경우에는 진행하기 전에 이 패키지를 설치 제거해야 합니다.
- 루트 권한을 가진 사용자 ID가 있어야 합니다.

프로시저

1. 설치 패키지 파일을 임시 디렉토리에 다운로드하십시오.
2. 임시 디렉토리로 변경하십시오.
3. 쉘 프롬프트에서 `./package`를 입력하여 설치 프로세스를 시작하십시오. 여기서 *package*는 설치하는 패키지의 이름입니다.
4. 설치 프로그램 창에 표시된 목록에서 언어를 선택한 후 **다음**을 클릭하십시오. 사용 가능한 언어의 목록은 시스템의 로케일 설정에 기반합니다.
5. 화면이동 막대를 사용하여 라이센스 텍스트의 끝까지 도달하면서 라이센스 계약을 읽으십시오. 설치를 진행하려면 라이센스 계약의 이용 약관에 동의해야 합니다. 이용 약관에 동의하려면 단일 선택 단추를 선택한 후 **확인**을 클릭하십시오.

참고: 라이센스 텍스트의 끝까지 읽어야 단일 선택 단추를 선택하여 라이센스 계약에 동의할 수 있습니다.

6. 설치의 대상 디렉토리 선택을 묻는 메시지가 나타납니다. 기본 디렉토리에 설치하지 않으려면, 브라우저 창을 사용하여 대체 디렉토리를 선택하도록 **선택**을 클릭하십시오. 설치 디렉토리를 선택했으면 **다음**을 클릭하여 계속하십시오.
7. 선택사항 검토를 묻는 메시지가 나타납니다. 선택사항을 변경하려면 **이전**을 클릭하십시오. 선택사항이 올바르면 **설치**를 클릭하여 설치를 진행하십시오.
8. 설치 프로세스가 완료되면 **완료**를 클릭하여 종료하십시오.

자동 설치 완료

설치할 시스템이 둘 이상이고 사용할 옵션을 이미 알고 있는 경우, 자동 설치 프로세스를 사용할 수 있습니다. 수동 설치를 사용하여 한 번 설치한 다음, 결과 응답 파일을 사용하여 추가적인 사용자 상호작용 없이 추가 설치를 완료합니다.

프로시저

1. 수동 설치를 완료하여 응답 파일을 작성하십시오. 다음 옵션 중 하나를 사용하십시오.
   - GUI를 사용하여 설치 프로그램에서 응답 파일을 작성함을 지정합니다. 응답 파일은 `installer.properties`라는 파일이며, 설치 디렉토리에 작성됩니다.
   - 명령행을 사용하고 수동 설치 명령에 -r 옵션을 추가하여 응답 파일에 대한 전체 경로를 지정합니다. 예를 들면 다음과 같습니다.

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

   예제 응답 파일 컨텐츠:

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

   이 예제에서 */my_directory*는 설치에 선택한 대상 설치 디렉토리입니다.
2. 옵션: 필요한 경우, 옵션을 변경하려면 응답 파일을 편집하십시오.

   참고: 아카이브 패키지에 다음의 알려진 문제가 있습니다. 응답 파일을 사용하는 설치에서 응답 파일에 있는 디렉토리를 변경해도 기본 디렉토리를 사용합니다. 이전 설치가 기본 디렉토리에 있으면 겹쳐씁니다.

   설치 옵션이 각각 다른 응답 파일을 두 개 이상 작성하는 경우, 각 응답 파일에 고유한 이름을 *myfile*`.properties` 형식으로 지정하십시오.
3. 옵션: 로그 파일을 생성하십시오. 자동으로 설치하는 중이므로, 설치 프로세스의 끝에 상태 메시지가 표시되지 않습니다. 설치 상태를 포함하는 로그 파일을 생성하려면 다음 단계를 완료하십시오.
   a. 다음 명령을 사용하여 필수 시스템 특성을 설정하십시오.

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

b. 로그 출력을 콘솔로 보내기 위해 다음 환경 변수를 설정하십시오.

```
export LAX_DEBUG=1
```

4. 패키지 설치 프로그램을 **-i** 자동 옵션 및 응답 파일을 지정하는 **-f** 옵션으로 실행하여 자동 설치를 시작하십시오. 예를 들면 다음과 같습니다.

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

특성 파일에 대한 완전한 경로 또는 상대 경로를 사용할 수 있습니다. 이러한 예제에서 `1>console.txt 2>&1` 문자열은 설치 프로세스 정보를 stderr 및 stdout 스트림에서 현재 디렉토리의 `console.txt` 로그 파일로 경로 재지정합니다. 설치에 문제점이 있다고 판단되면 이 로그 파일을 검토하십시오.

참고: 설치 디렉토리에 여러 응답 파일이 포함된 경우, 기본 응답 파일인 `installer.properties`가 사용됩니다.

인터럽트된 설치

패키지 설치 프로그램이 설치 중에 예상치 못하게 중지되는 경우(예: Ctrl+C를 누르는 경우), 설치가 손상되어 제품을 설치 제거하거나 다시 설치할 수 없습니다. 설치 제거하거나 다시 설치하려고 시도하면 심각한 애플리케이션 오류 메시지가 표시됩니다.

이 태스크 정보

이 문제점을 해결하려면 다음 단계에서 설명된 대로 파일을 삭제하고 다시 설치하십시오.

프로시저

1. `/var/.com.zerog.registry.xml` 레지스트리 파일을 삭제하십시오.
2. 설치가 들어 있는 디렉토리를 삭제(작성된 경우)하십시오. (예: /opt/IBM/javawrt3[_64]/)
3. 설치 프로그램을 다시 실행하십시오.

알려진 문제와 제한사항

InstallAnywhere 패키지에는 알려진 문제와 제한사항이 있습니다.

- `libstdc++.so.5` 공유 라이브러리가 시스템에 없으면 Java 코어 덤프가 생성되면서 설치에 실패합니다. 자세한 정보는 11 페이지의 『InstallAnywhere 패키지에서 설치』의 내용을 참조하십시오.
- 설치 패키지 GUI에서는 Orca 화면 판독 프로그램이 지원되지 않습니다. GUI에 대한 대체로서 자동 설치 모드를 사용할 수 있습니다.

- 설치 후 다음 작업을 수행하십시오. **./*package***를 입력하여 프로그램을 다시 시작하면 프로그램이 다음 메시지를 표시합니다.

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

  Enter를 눌러 기본값을 수락하는 경우 프로그램이 응답하지 않습니다. 숫자를 입력한 후 Enter를 누르십시오.
- 패키지를 설치한 후 다른 모드(예: 콘솔 또는 자동)에서 다시 설치하려고 시도하면, 다음 오류 메시지가 표시될 수 있습니다.

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

  GUI 모드를 사용하여 설치했고 콘솔 모드에서 설치 프로그램을 다시 실행하는 경우에는 이 메시지가 표시되지 않습니다. 이 오류가 표시되며 설치 제거 옵션(설치 가능 패키지만)을 선택하기 위해 프로그램을 실행하는 경우 19 페이지의 『WebSphere Real Time for Linux 설치 제거』에 설명된 대로 **./_uninstall/uninstall** 명령을 대신 사용하십시오.

설치 가능 패키지만

- InstallAnywhere 패키지를 사용하여 기존 설치를 업그레이드할 수 없습니다. WebSphere Real Time for Linux를 업그레이드하려면 우선 기존 버전을 설치 제거해야 합니다.
- 다른 설치 디렉토리를 사용하는 경우라도 동시에 동일한 시스템에서 동일한 버전의 WebSphere Real Time for Linux의 다른 두 인스턴스를 동시에 설치할 수 없습니다. 예를 들어, `/previous` 디렉토리에 있는 WebSphere Real Time for Linux V3와 `/current` 디렉토리에 있는 WebSphere Real Time for Linux Service Refresh 설치를 동시에 실행할 수 없습니다. 설치 프로그램이 버전 번호를 확인합니다. 프로그램이 동일한 버전 번호의 기존 패키지를 찾으면 기존 패키지를 설치 제거할지 여부를 선택해야 합니다.
- 패키지가 설치되어 있고 GUI를 사용하여 패키지 설치 프로그램을 다시 실행하는 경우, 패키지를 설치 제거하도록 선택할 수 있습니다. 이 설치 제거 옵션은 무인 모드에서는 사용할 수 없습니다. 패키지 설치 프로그램을 무인 모드에서 다시 실행하는 경우, 프로그램은 실행되지만 조치는 수행되지 않습니다.

아카이브 패키지만

- 응답 파일의 설치 디렉토리를 변경한 후 해당 응답 파일을 사용하여 자동 설치를 실행하면, 설치 프로그램은 새 설치 디렉토리를 무시하고 기본 디렉토리를 대신 사용합니다. 이전 설치가 기본 디렉토리에 있으면 겹쳐씁니다.

경로 설정

PATH 환경 변수를 설정한 경우, 쉘 프롬프트에 이름을 입력하여 애플리케이션 또는 프로그램을 실행할 수 있습니다.

이 태스크 정보

참고: 이 절에 설명된 대로 **PATH** 환경 변수를 변경하면 경로에 있는 기존 Java 실행 파일이 대체됩니다.

매번 도구 이름 앞에 경로를 입력하여 경로를 도구에 지정할 수 있습니다. 예를 들어 SDK가 /opt/IBM/javawrt3[_64]/에 설치된 경우 쉘 프롬프트에 다음 명령을 입력하여 *myfile.java* 파일을 컴파일할 수 있습니다.

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

전체 경로를 매번 입력하지 않으려면 다음을 수행하십시오.

1. 홈 디렉토리에 있는 쉘 시작 파일(쉘에 따라 다르며, 일반적으로 **.bashrc**)을 편집하고 절대 경로를 **PATH** 환경 변수에 추가하십시오. 예:

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

2. 다시 로그온하거나 업데이트된 쉘 스크립트를 실행하여 새 **PATH** 설정을 활성화하십시오.
3. 파일을 **javac** 도구와 컴파일하십시오. 예를 들어, *myfile.java* 파일을 컴파일하려면 쉘 프롬프트에서 다음을 입력하십시오.

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

   PATH 환경 변수를 사용하면 Linux에서 **javac**, **java** 및 **javadoc** 도구와 같은 실행 파일을 현재 디렉토리에서 찾을 수 있습니다. 경로의 현재 값을 표시하려면 명령 프롬프트에서 다음 명령을 입력하십시오.

   ```
   echo $PATH
   ```

다음에 수행할 작업

CLASSPATH 환경 변수를 설정해야 하는지 여부를 판별하려면 『클래스 경로 설정』의 내용을 참조하십시오.

클래스 경로 설정

CLASSPATH 환경 변수는 SDK 도구(예: **java**, **javac** 및 **javadoc** 도구)에 Java 클래스 라이브러리가 있는 위치를 알려줍니다.

이 태스크 정보

다음 중 한 가지 조건만 적용되는 경우 **CLASSPATH** 환경 변수를 명시적으로 설정하십시오.

- 직접 개발하거나 현재 디렉토리에 없는 새로운 라이브러리 또는 클래스 파일이 필요한 경우
- `bin` 및 `lib` 디렉토리의 위치를 변경하여 상위 디렉토리가 더 이상 동일하지 않은 경우.
- 동일한 시스템에서 다른 런타임 환경을 사용하는 애플리케이션을 개발하거나 실행할 계획인 경우.

CLASSPATH의 현재 값을 표시하려면 쉘 프롬프트에 다음 명령을 입력하십시오.

```
echo $CLASSPATH
```

다른 런타임 환경(별도로 설치한 다른 버전 포함)을 사용하는 애플리케이션을 개발하고 실행하는 경우, 각 애플리케이션에 대해 **CLASSPATH** 및 **PATH**를 명시적으로 설정해야 합니다. 여러 애플리케이션을 동시에 실행하며 다른 런타임 환경을 사용하는 경우, 각 애플리케이션을 자체 쉘에서 실행해야 합니다.

한 번에 한 개의 Java 버전만 실행하려면 쉘 스크립트를 사용하여 다른 런타임 환경 간에 전환할 수 있습니다.

다음에 수행할 작업

성공적으로 설치되었는지 확인하려면 『설치 테스트』의 내용을 참조하십시오.

설치 테스트

성공적으로 설치되었는지 확인하려면 **-version** 옵션을 사용하십시오.

이 태스크 정보

Java 설치는 실시간 JVM으로 구성됩니다.

프로시저

설치를 테스트하려면 다음 단계를 따르십시오.

1. 실시간 JVM의 버전 정보를 알려면 쉘 프롬프트에서 다음 명령을 입력하십시오.

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

   이 명령은 성공할 경우 다음 메시지를 리턴합니다.

```
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
```

참고: 버전 정보는 올바르지만 플랫폼 아키텍처 및 날짜가 예제와 다를 수 있습니다. 날짜 문자열의 형식은 yyyymmdd이며 뒤에 구성요소별 추가 정보가 나올 수도 있습니다.

WebSphere Real Time for Linux 설치 제거

WebSphere Real Time for Linux의 제거에 사용하는 프로세스는 사용한 설치 유형에 따라서 달라집니다.

시작하기 전에

InstallAnywhere 설치 가능 패키지의 경우 루트 권한을 가진 사용자 ID가 있어야 합니다.

이 태스크 정보

InstallAnywhere 아카이브 패키지의 경우 설치 제거 프로세스가 없습니다. 시스템에서 아카이브 패키지를 제거하려면, 패키지를 설치할 때 선택한 대상 디렉토리를 삭제하십시오. InstallAnywhere 설치 가능 패키지의 경우, 다음 단계에서 설명된 대로 명령을 사용하거나 설치 프로그램을 다시 실행하여 제품을 설치 제거합니다.

프로시저

- 옵션: **uninstall** 명령을 사용하여 수동으로 설치 제거하십시오.
  1. 설치가 포함된 디렉토리로 변경하십시오. 예를 들면 다음과 같습니다.

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

  2. `./_uninstall/uninstall` 명령을 입력하여 설치 제거 프로세스를 시작하십시오.
- 옵션: 설치 제거 프로그램을 쉽게 찾을 수 없는 경우, 대체하여 다른 수동 설치를 실행할 수 있습니다. 설치 프로그램에서 제품이 이미 설치되었음을 발견하면 이전 설치를 설치 제거하는 기회를 제공합니다.

제 5 장 IBM WebSphere Real Time for Linux 애플리케이션 실행

실시간 애플리케이션 실행 시 유용한 중요한 정보입니다.

- 『스레드 스케줄링 및 디스패치』
-
- 25 페이지의『메트로놈 가비지 콜렉터 사용』

스레드 스케줄링 및 디스패치

Linux 운영 체제는 다양한 스케줄링 정책을 지원합니다. 기본 유니버셜 시간 공유 스케줄링 정책은 대부분의 스레드에 사용되는 SCHED_OTHER입니다. SCHED_RR 및 SCHED_FIFO는 실시간 애플리케이션의 스레드가 사용할 수 있습니다. SCHED_OTHER 및 SCHED_RR만 WebSphere Real Time for Linux에서 사용합니다.

커널은 프로세서에서 실행할 다음 실행 가능 스레드를 결정합니다. 커널이 실행 가능 스레드 목록을 관리합니다. 우선순위가 가장 높은 스레드를 찾고 실행할 다음 스레드로 해당 스레드를 선택합니다.

다음 명령을 사용하여 스레드 우선순위 및 정책을 나열할 수 있습니다.

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

여기서 policy:

- TS는 SCHED_OTHER입니다.
- RR은 SCHED_RR입니다.
- FF는 SCHED_FIFO입니다.
- -는 보고된 정책이 없습니다.

출력은 다음 예제와 같습니다.

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

```
-     - RR  31544 -                    6 00:00:00
-     - RR  31545 -                    6 00:00:00
-     - RR  31546 -                    6 00:00:00
```

이 출력은 Java 프로세스, 정책 SCHED_RR 및 우선순위 6의 많은 스레드를 보여줍니다.

현재 스케줄링 정책을 조회하려면 예제에 표시된 **sched_getscheduler** 또는 **ps** 명령을 사용하십시오.

프로세스에 대한 자세한 정보는 40 페이지의 『일반 디버깅 기술』를 참조하십시오.

일반 Java 스레드 우선순위 및 정책

java.lang.Thread 오브젝트로 할당되는 스레드인 일반 Java 스레드는 기본 스케줄링 정책 SCHED_OTHER를 사용합니다. WebSphere Real Time for Linux V3 Service Refresh 1부터 SCHED_RR 스케줄링 정책을 사용하여 일반 Java 스레드를 실행할 수 있습니다.

기본적으로 Java 스레드는 기본 SCHED_OTHER 정책을 사용하여 실행합니다. 이 정책은 Java 스레드를 운영 체제 우선순위 0에 맵핑합니다.

SCHED_RR 정책을 사용하면 애플리케이션을 보다 정교하여 제어하여 Java 스레드의 실시간 성능을 높일 수 있습니다. JVM은 SCHED_RR 정책으로 Java를 시작할 때 기본 스레드의 우선순위 및 정책을 발견합니다. JVM은 이에 맞게 우선순위 및 정책 맵핑을 변경합니다. 모든 Java 스레드가 기본 스레드와 동일한 운영 체제 우선순위로 실행됩니다. SCHED_RR에 우선순위 1 - 99가 지정되더라도 WebSphere Real Time for Linux V3에 대한 사용 가능 SCHED_RR 우선순위는 1 - 10입니다. 우선순위가 10보다 높게 설정되면 기본 스레드의 우선순위가 10으로 낮아지고 10의 값에 따라 맵핑이 적용됩니다.

명령행에서 프로세스의 실시간 스케줄링 특성을 변경하는 한 가지 방법은 **chrt** 명령을 사용하는 것입니다. 다음 예제에서 기본 Java 스레드의 우선순위가 운영 체제 우선순위 6으로 SCHED_RR 스케줄링 정책을 사용하도록 설정됩니다.

```
chrt -r 6 java
```

우선순위 변경을 허용하도록 시스템을 구성해야 할 수도 있습니다. 자세한 정보는 웹 사이트 23 페이지의 『우선순위 변경을 허용하도록 시스템 구성』의 내용을 참조하십시오.

표 2. Java 및 운영 체제 우선순위

Java 우선순위	**스레드의 Java 우선순위 값**	**운영 체제 우선순위**
1	MIN_PRIORITY	6
2		6
3		6
4		6

표 2. Java 및 운영 체제 우선순위 (계속)

Java 우선순위	**스레드의 Java 우선순위 값**	**운영 체제 우선순위**
5	NORM_PRIORITY(기본값)	6
6		6
7		6
8		6
9		6
10	MAX_PRIORITY	6

기본 Java 스레드와 연관된 모든 스레드는 동일한 운영 체제 우선순위로 실행됩니다.

`chrt -r 11 java` 명령을 실행하면 `chrt -r 10 java`를 실행할 때와 결과가 동일합니다. 이는 JVM을 시작하고 JVM 종료를 대기하는 스레드가 우선순위 11로 유지되더라도 10 이상의 우선순위를 JVM 스레드가 사용하는 우선순위 맵핑에 적용할 수 없기 때문입니다.

SCHED_FIFO이 WebSphere Real Time for Linux V3에서 지원되지 않기 때문에 JVM은 `chrt -f 6 java` 명령을 사용하려고 할 때 오류 메시지를 생성합니다.

chrt 명령에 대한 자세한 정보는 http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaai/realtime/liaairtchrt.htm의 내용을 참조하십시오.

우선순위 변경을 허용하도록 시스템 구성

기본적으로, Linux의 루트가 아닌 사용자는 스레드 또는 프로세스의 우선순위를 높일 수 없습니다. Linux용 PAM(Pluggable Authentication Modules)의 `pam_limits` 모듈을 사용하여 우선순위 변경을 허용하도록 시스템 구성을 변경합니다.

chrt 유틸리티를 사용하여 스레드 또는 프로세스의 우선순위를 변경할 수 없는 경우 일반적으로 다음과 같은 메시지가 나타납니다.

```
sched_setscheduler: Operation not permitted
```

최근 Linux 커널에서 `pam_limits` 모듈을 사용하여 우선순위 변경을 허용하도록 시스템 구성을 변경할 수 있습니다. 이 모듈로 한계 구성 파일의 시스템 자원 한계를 구성할 수 있습니다. 기본 파일은 `/etc/security/limits.conf`입니다.

`/etc/security/limits.conf` 파일의 항목은 다음 양식을 가집니다.

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

여기서, 각 인수는 다음을 의미합니다.

<domain>은 다음 중 하나입니다.

- 시스템에서 자원의 한계를 변경할 수 있는 사용자 이름입니다.
- 구성원이 자원의 한계를 변경할 수 있는 `@group` 구문을 포함한 그룹 이름입니다.

- 사용자 또는 그룹이 자원의 한계를 변경할 수 있음을 나타내는 와일드카드 "*"입니다.

<type>은 다음 중 하나입니다.

- 커널에서 하드 한계를 적용하는 hard입니다.

- 소프트 한계가 적용되는 soft이며 하드 한계가 지정하는 범위 안에서 변경할 수 있습니다.

- 하드 한계 및 소프트 한계를 나타내는 대시 "-"입니다.

<item>은 다음입니다.

- 자원입니다. 실시간 우선순위에 대해서는 rtprio를 사용하십시오.

<value>는 다음입니다.

- 한계입니다. 실시간 우선순위 설정의 최대 한계를 나타내려면 1 - 100 범위에서 값을 사용하십시오.

예를 들어 다음과 같습니다.

```
* - rtprio 100
```

chrt 또는 기타 메커니즘을 사용하여 모든 사용자가 실시간 프로세스의 우선순위를 변경하도록 허용할 수 있습니다.

기본적으로, 루트 사용자가 한계 없이 실시간 우선순위를 높일 수 있습니다. 루트에 한계를 적용하려면 루트 사용자를 명시적으로 지정해야 합니다. 구성 파일의 그룹 및 와일드카드 한계는 루트 사용자에 적용되지 않습니다.

파일에 개별 사용자 한계를 지정할 경우, 이 한계가 그룹 한계보다 우선합니다.

limits.conf 변경사항은 즉시 적용되지 않습니다. 구성 변경사항을 적용하려면 관련 서비스를 다시 시작하거나 시스템을 재부팅해야 합니다.

JVM(Java Virtual Machine)의 실시간 우선순위를 높이는 기능을 Linux 커널 2.6.12 이상에서는 사용할 수 없습니다. 다음 표는 몇 가지 일반 Linux 배포판에서 이 기능의 지원 여부를 보여줍니다.

표 3. 실시간 우선순위 변경 지원

Linux 배포판	**Linux 커널 버전**	**실시간 우선순위 변경 지원(예/아니오)**
Red Hat Enterprise Linux(RHEL) 4	2.6.9	아니오
RHEL 5 이상	2.6.18 이상	예
SUSE Linux Enterprise Server(SLES) 9	2.6.5-7	아니오
SLES 10 이상	2.6.16 이상	예

표 3. 실시간 우선순위 변경 지원 (계속)

Linux 배포판	**Linux 커널 버전**	**실시간 우선순위 변경 지원(예/아니오)**
Red Hat Enterprise MRG - 모든 버전	2.6.24 이상	예
SUSE Linux Enterprise Real Time(SLERT) - 모든 버전	2.6.16 이상	예
Ubuntu 5.10	2.6.12	아니오
Ubuntu 6.06 이상	2.6.15 이상	예

`limits.conf` 파일에 표시되는 `realtime` 그룹에 사용자를 추가하여 실시간 Linux 시스템에서 우선순위 변경을 사용할 수 있습니다.

보조 프로세스 시작

JVM(Java Virtual Machine) API의 java.lang.Runtime.exec 메소드는 Java 애플리케이션에 별도의 프로세스에서 명령을 실행하는 기능을 제공합니다.

메소드 호출에서 새 java.lang.Process 오브젝트가 작성됩니다. 해당 오브젝트는 새 프로세스를 제어하거나 관련 정보를 확보하는 데 사용합니다.

이를 위해 exec 메소드에서 몇 가지 스레드를 작성합니다. IBM WebSphere Real Time for Linux에서 프로시저를 수정하면 실시간 환경에서 보다 결정적 동작이 가능합니다.

Runtime.exec 호출은 각 분기 서브프로세스에 대한『reaper』스레드를 작성합니다. reaper 스레드는 서브프로세스가 종료될 때까지 대기하는 유일한 스레드입니다. 서브프로세스가 종료될 때 reaper 스레드는 서브프로세스 종료 상태를 기록합니다. reaper 스레드는 새 프로세스를 생성하고 원래 Runtime.exec를 호출한 스레드와 동일한 우선순위를 제공합니다.

생성된 프로세스가 다른 WebSphere Real Time for Linux JVM이고 Runtime.exec 메소드를 Linux 실시간 정책 및 우선순위로 실행되는 다른 메소드에서 호출한 경우, 새 가상 머신의 기본 스레드가 해당 정책 및 우선순위를 동일한 Linux 실시간 정책 및 우선순위에 맵핑합니다. 이 Java 스레드 우선순위는 1 - 10 사이입니다.

또한 reaper 스레드가 새 프로세스의 `stdout` 및 `stderr` 스트림을 인식하는 두 개의 새 스레드를 작성합니다. `stdout` 및 `stderr` 데이터는 이 스레드가 사용하는 버퍼에 저장됩니다. 해당 버퍼는 생성된 프로세스 수명보다 오래 지속됩니다. 이 지속성 때문에 생성된 프로세스로 인해 보류된 자원을 프로세스 종료 시 바로 해제할 수 있습니다.

메트로놈 가비지 콜렉터 사용

WebSphere Real Time for Linux에서 표준 가비지 콜렉터 대신 메트로놈 가비지 콜렉터를 사용합니다.

일시정지 시간 제어

메트로놈 가비지 콜렉터(GC) 일시정지 시간은 각 Java 프로세스에 대해 더 자세하게 조정될 수 있습니다.

기본적으로 각각의 개별 일시정지에서 3밀리초 동안의 메트로놈 GC 일시정지를 할당량이라고 합니다. 전체 가비지 콜렉션 주기에는 이러한 일시정지가 많이 필요하며, 애플리케이션에 충분한 실행 시간을 제공하기 위해 전반에 걸쳐 있습니다. **-Xgc:targetPauseTime** 옵션으로 이 최대 개별 일시정지 시간을 변경할 수 있습니다. 예를 들어 **-Xgc:targetPauseTime=20**으로 실행하면 GC가 20밀리초보다 길지 않은 개별 일시정지로 작동합니다.

IBM Monitoring and Diagnostics Tools for Java - GCMV(Garbage Collection and Memory Visualizer)를 사용하면 애플리케이션의 GC 일시정지 시간을 모니터할 수 있고, Java 애플리케이션의 성능 문제점을 진단하고 조정하는 데 도움이 됩니다. 도구에서는 다음을 포함하여 다양한 유형의 로그를 구문 분석하고 구상합니다.

- 상세 가비지 콜렉션 로그.
- **-Xtgc** 매개변수를 사용하여 생성되는 추적 가비지 콜렉션 로그.
- **ps**, **svmon**, 또는 **perfmon** 시스템 명령을 사용하여 생성되는 기본 메모리 로그.

이 절의 그래프는 GCMV에서 생성되었으며, 가비지 콜렉션 사이클에 대한 목표 일시정지 시간 변경의 영향을 보여줍니다. 각 그래프는 애플리케이션의 실행 시간(X축)에 대한 메트로놈 가비지 콜렉션 사이클(Y축) 사이의 실제 일시정지 시간을 구상합니다.

참고: GCMV는 기존 상세 가비지 콜렉션 형식을 지원합니다. GCMV로 상세 GC 출력을 분석하려면 **-Xgc:verboseFormat=deprecated** 옵션으로 출력을 생성하십시오. 자세한 정보는 GC 명령행 옵션을 참조하십시오.

기본 목표 일시정지 시간 설정을 사용하면, 상세 GC 일시정지 시간 그래프에는 3밀리초 정도이거나 아래의 표시가 보유되는 일시정지 시간이 표시됩니다.

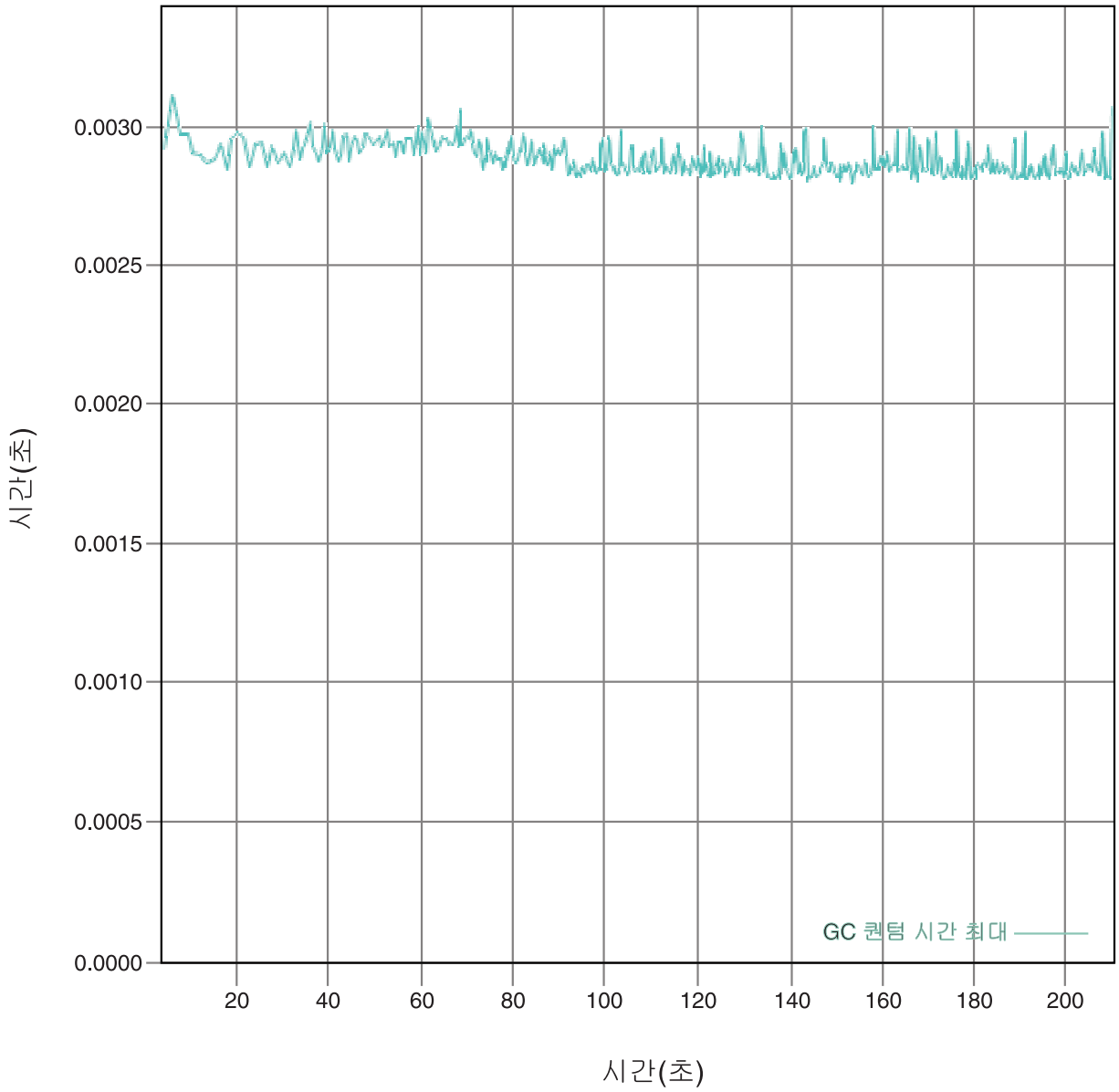

그림 1. 목표 일시정지 시간이 기본값(3밀리초)으로 설정된 경우 실제 가비지 콜렉션 일시정지 시간

목표 일시정지 시간 설정인 6밀리초를 사용하면, 상세 GC 일시정지 시간 그래프에는 6밀리초 정도이거나 아래의 표시가 보유되는 일시정지 시간이 표시됩니다.

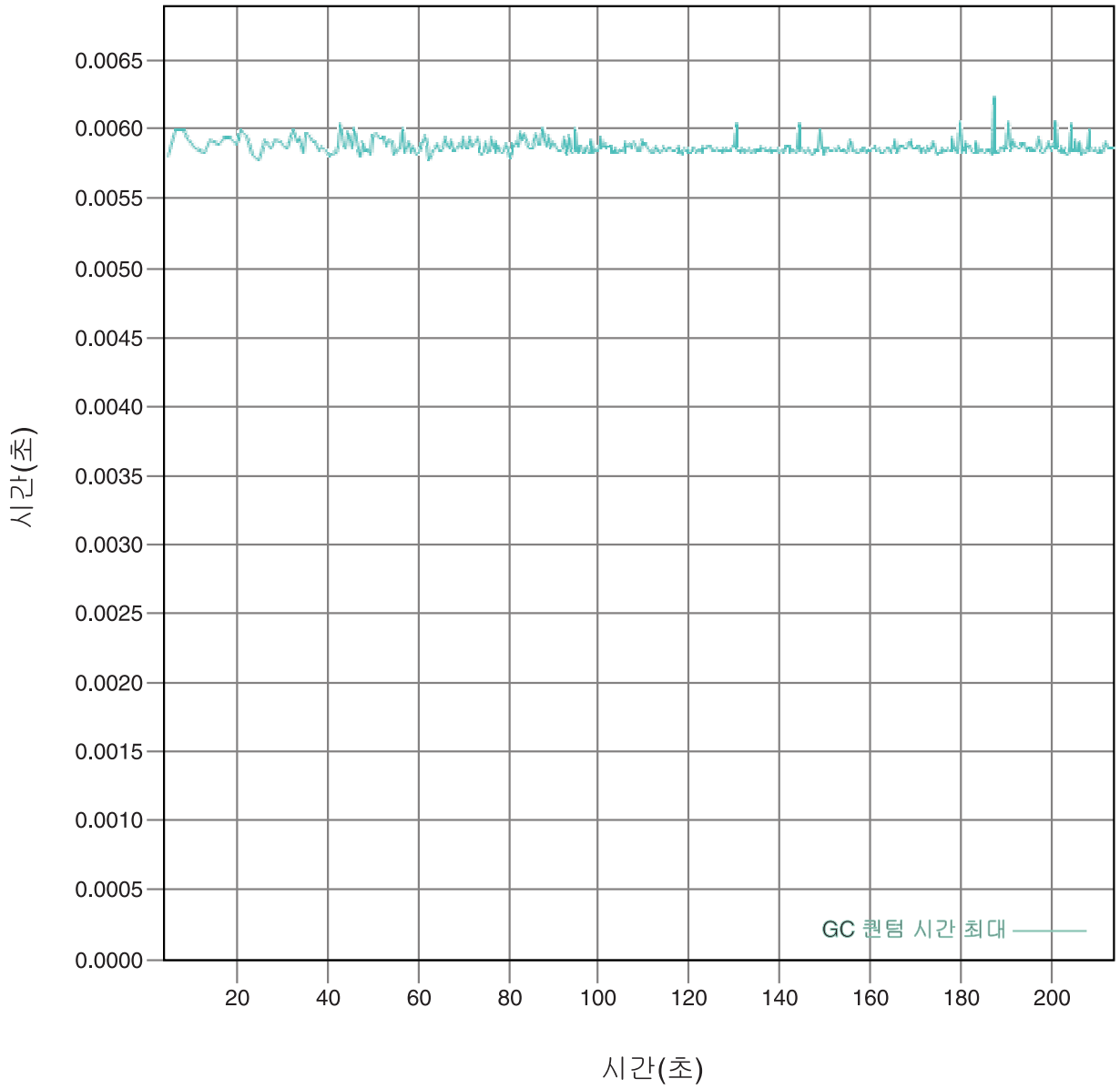

그림 2. 목표 일시정지 시간이 6밀리초로 설정된 경우 실제 일시정지 시간

목표 일시정지 시간 설정인 10밀리초를 사용하면, 상세 GC 일시정지 시간 그래프에는 10밀리초 정도이거나 아래의 표시가 보유되는 일시정지 시간이 표시됩니다.

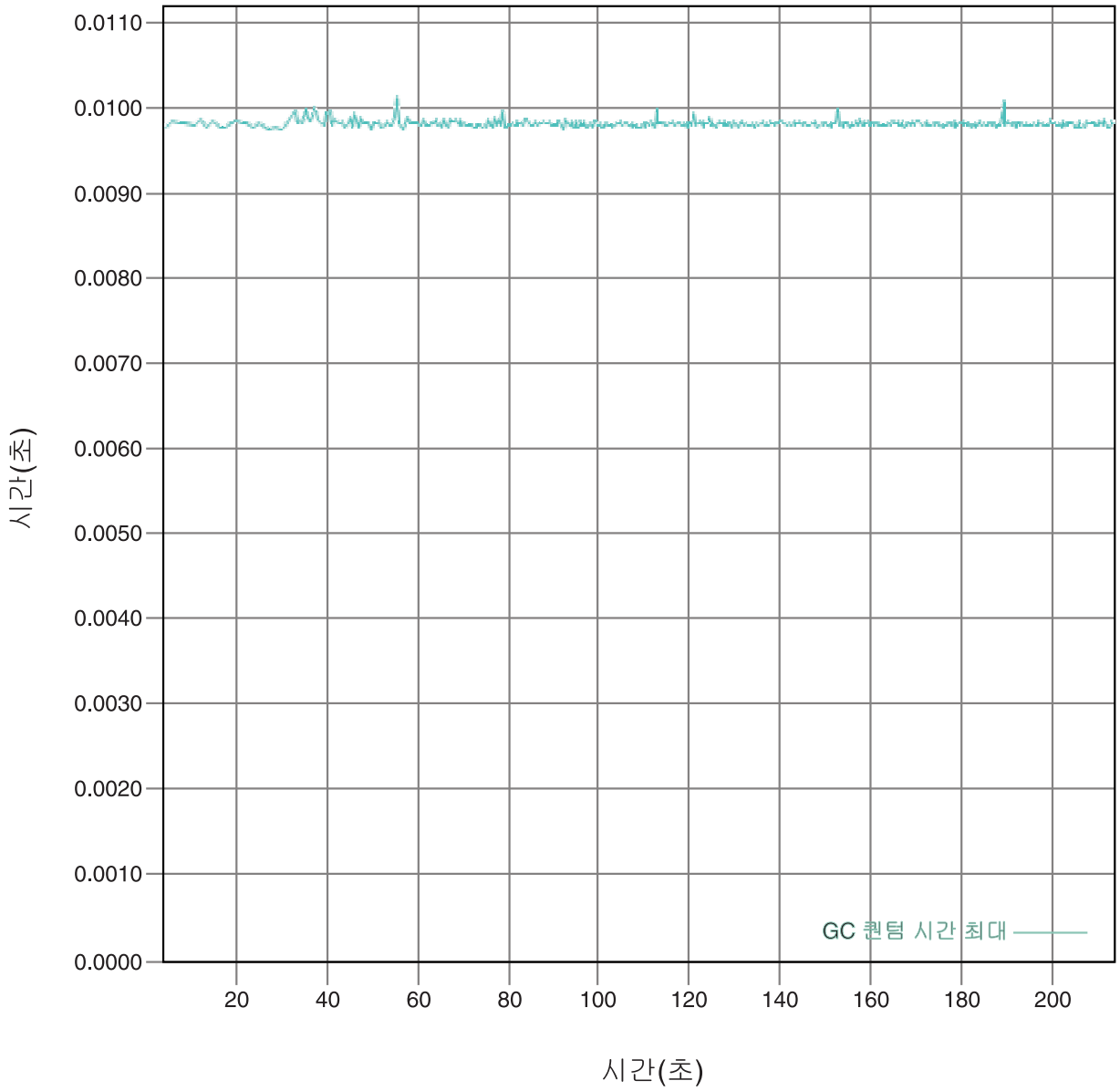

그림 3. 목표 일시정지 시간이 10밀리초로 설정된 경우 실제 일시정지 시간

목표 일시정지 시간 설정인 15밀리초를 사용하면, 상세 GC 일시정지 시간 그래프에는 15밀리초 정도이거나 아래의 표시가 보유되는 일시정지 시간이 표시됩니다.

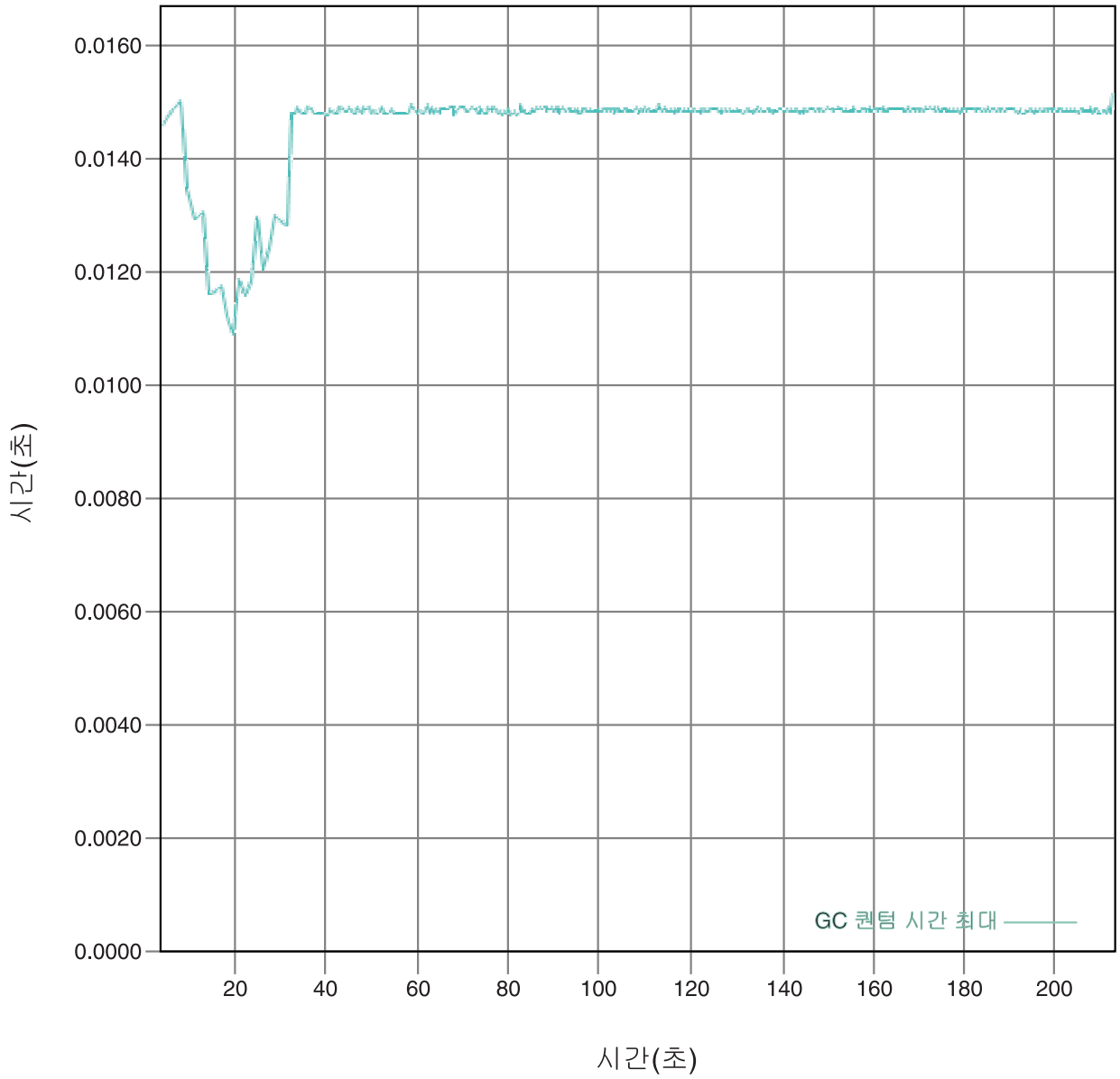

그림 4. 목표 일시정지 시간이 15밀리초로 설정된 경우 실제 일시정지 시간

프로세서 사용 제어

메트로놈 가비지 콜렉터에 사용 가능한 처리 성능 크기를 제한할 수 있습니다.

가비지 콜렉터에서 사용되는 CPU 시간을 한정하도록 **-Xgc:targetUtilization=N** 옵션을 사용하여 메트로놈 가비지 콜렉터로 가비지 콜렉션을 제어할 수 있습니다.

예를 들면 다음과 같습니다.

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

예제에서는 애플리케이션이 60밀리초마다 80%에 대해 실행됨을 지정합니다. 나머지 20% 시간은 가비지 콜렉션에 사용됩니다. 메트로놈 가비지 콜렉터는 충분한 자원이 제공될 경우에 한해 이용 수준을 보장합니다. 힙의 여유 공간 *크기*가 동적으로 판별되는 임계값보다 작아지면 가비지 콜렉션이 시작됩니다.

메트로놈 가비지 콜렉터 제한사항

이 주제에서는 메트로놈 GC 정책에 영향을 미치는 알려진 문제 또는 제한사항에 대해 설명합니다.

x86 플랫폼에서 AESNI 지원

x86 아키텍처에서 AESNI 지시사항의 소프트웨어 개발은 현재 메트로놈 GC 정책에 지원되지 않습니다.

가비지 콜렉션 중의 긴 일시정지 시간

가비지 콜렉션 중에 일시정지 시간이 예상보다 더 길어지는 경우가 드물게 나타날 수 있습니다. 가비지 콜렉션 동안 루트 스캔 프로세스가 사용됩니다. 가비지 콜렉터가 알려진 라이브 참조에서 시작하여 힙을 지나갑니다. 이 참조는 다음과 같습니다.

- 활성 스레드 호출 스택의 라이브 참조 변수
- Static 참조

애플리케이션 스레드 스택에서 모든 라이브 오브젝트 참조를 찾기 위해 가비지 콜렉터가 스레드 호출 스택에서 모든 스택 프레임을 스캔합니다. 각 활성 스레드 스택을 인터럽트 불가능한 단계에서 스캔합니다. 따라서 스캔은 개별 GC 일시정지 내에서 발생해야 합니다.

그 영향으로, 매우 깊은 스택을 포함한 스레드가 몇 개 있는 경우 시스템 성능이 예상보다 저하질 수 있는데 이유는 콜렉션 주기를 시작할 때 가비지 콜렉션 일시정지가 길어지기 때문입니다.

제 6 장 애플리케이션 개발

코드 샘플을 포함한 실시간 애플리케이션 작성에 대한 중요 정보입니다.

- 『샘플 실시간 해시 맵』

샘플 실시간 해시 맵

WebSphere Real Time for Linux에는 IBM SDK for Java 7에서 표준 HashMap보다 put 메소드에 대해 더 일정한 성능을 제공하는 HashMap 및 HashSet 구현이 포함되어 있습니다.

IBM이 제공하는 표준 java.util.HashMap은 처리량이 높은 애플리케이션에서 잘 작동합니다. 또한 해시 맵이 확장되어야 하는 최대 *크기*를 애플리케이션이 알 수 있도록 지원합니다. *가변 크기*로 확장할 수 있는 해시 맵이 필요한 애플리케이션의 경우, 사용량에 따라 표준 해시 맵에서 성능 문제점이 발생할 수 있습니다. 표준 해시 맵은 put 메소드를 사용하여 새 항목을 해시 맵에 추가하기 위한 좋은 응답 시간을 제공합니다. 그러나, 해시 맵이 꽉 차면 더 큰 지원 저장소를 할당해야 합니다. 즉, 현재 지원 저장소에 있는 항목을 마이그레이션해야 합니다. 해시 맵이 크면 put 수행 시간이 길어질 수 있습니다. 예를 들어, 이 조작은 몇 밀리초가 걸릴 수 있습니다.

WebSphere Real Time for Linux에 샘플 실시간 해시 맵이 포함되어 있습니다. 표준 java.util.HashMap과 동일한 기능 인터페이스를 제공하지만 put 메소드에 대해 보다 일관된 성능을 허용합니다. 지원 저장소를 작성하고 해시 맵이 꽉 차면 모든 항목을 마이그레이션하는 대신, 샘플 해시 맵이 추가 지원 저장소를 작성합니다. 새 지원 저장소는 해시 맵의 다른 지원 저장소에 체인 형식으로 연결됩니다. 체인 형식의 연결로 인해 빈 지원 저장소를 할당하고 다른 지원 저장소에 체인 형식으로 연결하는 동안 초기에 성능이 약간 저하됩니다. 지원 해시 맵을 업데이트하고 나면 모든 항목을 마이그레이션하는 것보다 빨라집니다. 실시간 해시 맵의 단점은 get, put 및 remove 조작이 다소 느려질 수 있다는 점입니다. 각 검색을 단일 항목이 아닌 지원 해시 맵 세트에서 진행해야 하기 때문에 조작이 느려집니다.

실시간 해시 맵을 사용해 보려면 `RTHashMap.jar` 파일을 부트클래스 경로의 시작 부분에 추가하십시오. WebSphere Real Time for Linux를 디렉토리 `$WRT_ROOT`에 설치한 경우 표준 해시 맵 대신 다음 옵션을 추가하여 애플리케이션에서 실시간 해시 맵을 사용하도록 하십시오.

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

실시간 해시 맵 구현의 소스 및 클래스 파일이 demo/realtime/RTHashMap.jar 파일에 포함되어 있습니다. 또한, 실시간 java.util.LinkedHashMap 및 java.util.HashSet 구현도 제공됩니다.

제 7 장 성능

WebSphere Real Time for Linux는 가장 높은 처리량 성능 또는 가장 작은 메모리 풋프린트보다 일관되게 짧은 GC 일시정지를 위해 최적화되었습니다.

인증된 하드웨어 구성에서의 성능

인증된 시스템에서 WebSphere Real Time for Linux 성능 목표를 지원하기에 충분한 클럭 단위 및 프로세서 속도가 제공됩니다. 예를 들어, 적합한 힙 *크기*를 사용하고 오버로드되지 않은 시스템에서 실행 중인 잘 작성된 애플리케이션에서는 일반적으로 약 3밀리초 또는 3.2 밀리초의 GC 일시정지가 발생합니다. GC 주기 동안 기본 환경 설정의 애플리케이션은 슬라이딩 60 밀리초 시간대 동안 경과 시간의 30% 이상 일시정지되지 않습니다. 60밀리초를 초과하는 GC 일시정지의 집합 시간은 일반적으로 총 약 18밀리초입니다.

시간 변동 줄이기

표준 JVM에서 주요 변동 원인은 가비지 콜렉션 일시정지입니다. WebSphere Real Time for Linux에서 표준 가비지 콜렉터 모드에서의 장시간 일시정지는 메트로놈 가비지 콜렉터를 사용하면 방지할 수 있습니다. 25 페이지의 『메트로놈 가비지 콜렉터 사용』의 내용을 참조하십시오.

JVM 간 클래스 데이터 공유

클래스 데이터 공유는 메모리 풋프린트를 줄이고 JVM 시작 시간을 향상시키는 명백한 메소드를 제공합니다. 클래스 데이터 공유에 대해 자세히 알려면 36 페이지의 『JVM 사이의 클래스 데이터 공유』의 내용을 참조하십시오.

압축 참조

메트로놈 GC는 64비트 플랫폼에서 압축 및 압축 해제된 참조를 둘 다 지원합니다. 압축 참조를 사용할 때 JVM은 오브젝트, 클래스, 스레드 및 모니터에 대한 모든 참조를 32비트 값으로 저장합니다. 압축 참조를 사용하면 오브젝트의 수가 적어지므로 가비지 콜렉션 주기가 감소하고 메모리 캐시 활용이 향상되어 많은 애플리케이션의 성능이 향상됩니다.

참고: 압축 참조에 사용 가능한 힙 *크기*는 약 28GB로 제한됩니다.
압축 참조에 대한 자세한 정보는 http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/understanding/mm_compressed_references.html의 내용을 참조하십시오.

JVM 사이의 클래스 데이터 공유

공유 클래스 지원은 **-Xrealtime** 옵션을 사용하여 실행하는 경우나 사용하지 않고 실행하는 경우 모두 동일합니다.

공유 클래스 데이터를 디스크의 메모리 맵핑된 캐시 파일에 저장하여 JVM(Java Virtual Machine)사이에서 해당 데이터를 공유할 수 있습니다. 둘 이상의 JVM이 캐시를 공유하는 경우 데이터를 공유하면 전체 가상 스토리지 이용이 줄어듭니다. 또한 데이터를 공유하면 캐시를 작성한 후 JVM의 시작 시간도 단축됩니다. 공유 클래스 캐시는 실행 중인 JVM에 종속되지 않으며 삭제할 때까지 지속됩니다.

공유 캐시에는 다음이 포함될 수 있습니다.

- 부트스트랩 클래스
- 애플리케이션 클래스
- 클래스를 설명하는 메타데이터
- AOT(Ahead-of-time) 컴파일된 코드

참고: 실시간 이외 JVM으로 실시간 공유 클래스 캐시를 제거할 수 없습니다.

제 8 장 보안

이 섹션에는 보안에 대한 중요한 정보가 있습니다.

공유 클래스 캐시의 보안 고려사항

공유 클래스 캐시는 각 캐시 관리 및 사용을 위해 설계되었지만 기본 보안 정책이 적합하지 않을 수 있습니다.

공유 클래스 캐시를 사용할 때는 액세스 제한으로 보안이 강화되도록 사용자가 새 파일에 대한 기본 권한을 알고 있어야 합니다.

파일	기본 권한
새 공유 캐시	그룹 및 other의 읽기 권한
javasharedresources 디렉토리	읽기, 쓰기 및 실행 권한

캐시를 제거하거나 확장하려면 캐시 파일 및 캐시 디렉토리 둘 다에 대한 쓰기 권한이 필요합니다.

캐시 파일의 파일 권한 변경

공유 클래스 캐시에 대한 액세스를 제한하려면 **chmod** 명령을 사용합니다.

필요한 변경	명령
사용자 및 그룹으로 액세스 제한	chmod 770 /tmp/javasharedresources
사용자로 액세스 제한	chmod 700 /tmp/javasharedresources
사용자를 특정 캐시 전용의 읽기 및 쓰기 액세스로 제한	chmod 600 /tmp/javasharedresources/<*file for shared cache*>
사용자 및 그룹을 특정 캐시 전용의 읽기 및 쓰기 액세스로 제한	chmod 660 /tmp/javasharedresources/<*file for shared cache*>

액세스 권한이 없는 캐시에 연결

올바른 액세스 권한이 없는 캐시에 연결하려고 하면 오류 메시지가 표시됩니다.

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

제 9 장 문제점 해결 및 지원

WebSphere Real Time for Linux에 대한 문제점 해결 및 지원 방법을 설명합니다.

- 『일반 문제점 판별 메소드』
- 46 페이지의『OutOfMemory 오류 문제점 해결』
- 50 페이지의『진단 도구 사용』

일반 문제점 판별 메소드

문제점 판별은 사용자가 결함의 종류와 적절한 조치 순서에 대해 이해할 수 있도록 도와줍니다.

문제점의 종류가 무엇인지 알고 있으면 다음 태스크를 수행할 수 있습니다.

- 문제점을 수정하십시오.
- 유용한 해결 방법을 찾으십시오.
- IBM으로 버그 보고서를 생성하는 데 필요한 데이터를 콜렉트하십시오.

Linux 문제점 판별

이 섹션에서는 Linux에서의 문제저 판별에 대해 설명합니다.

IBM SDK for Java V7 사용자 안내서에는 Linux에서 발생하는 문제점 진단에 필요한 정보가 있으며 다음 내용을 다루고 있습니다.

- Linux 환경 설정 및 확인
- 일반 디버깅 기술
- 크래쉬 진단
- 정지 디버깅
- 메모리 누수 디버깅
- 성능 문제 디버깅

이 정보는 IBM SDK for Java 7 - Linux 문제 판별에서 찾을 수 있습니다.

다음 정보는 IBM WebSphere Real Time for Linux에 대한 보충 정보입니다.

Linux 환경 설정 및 확인

IBM WebSphere Real Time for Linux에서 JVM이 시스템 덤프를 생성하도록 올바르게 구성되어 있는지 확인합니다.

Linux 시스템 덤프(코어 파일)

클래쉬가 발생하는 경우, 획득해야 하는 가장 중요한 진단 데이터는 Linux 시스템 덤프(코어 파일)입니다. 이 파일이 생성되도록 하려면 IBM SDK for Java V7 사용자 안내서에 설명된 대로 운영 체제 설정과 사용 가능한 디스크 공간을 확인해야 합니다.

JVM(Java Virtual Machine) 설정

크래쉬가 발생할 때 코어 파일을 생성할 수 있도록 JVM을 구성해야 합니다. 명령행에서 `java -Xdump:what`을 실행하십시오. 이 옵션의 출력은 다음과 같습니다.

```
-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
```

표시된 값이 기본 설정입니다. 최소한 `events=gpf`가 설정되어야 시스템 크래쉬 발생 시에 코어 파일을 생성할 수 있습니다. 명령행 옵션 **`-Xdump:system[:name1=value1,name2=value2 ...]`**를 사용하여 옵션을 변경하고 설정할 수 있습니다.

일반 디버깅 기술

운영 체제에서 Java 스레드 이름이 보이므로 ps 명령을 사용하여 디버깅을 지원할 수 있습니다. 추적 도구를 사용할 때 IBM WebSphere Real Time for Linux에 대해 올바른 명령을 사용해야 합니다.

프로세스 정보 조사

IBM WebSphere Real Time for Linux에서 ps 명령을 실행할 때 표시될 수 있는 예상 출력은 다음과 같습니다.

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

e 모든 프로세스를 선택합니다.

L 스레드를 표시합니다.

o 표시할 열의 사전 정의된 형식을 제공합니다. 지정된 열은 프로세스 ID, 스레드 ID, 스케줄 정책, 실시간 스레드 우선순위 및 프로세스와 연관된 명령입니다. 이 정보는 가상 시스템뿐 아니라 사용자 애플리케이션에서 주어진 시간에 어떤 스레드가 실행 중인지를 이해하는 데 유용합니다.

추적 도구

Linux의 세 가지 추적 도구는 **strace**, **ltrace** 및 **mtrace**입니다. 명령 `man strace`는 사용 가능한 전체 옵션 세트를 표시합니다.

strace

strace 도구는 시스템 호출을 추적합니다. 사용 가능한 프로세스에서 이를 사용하거나 새 프로세스로 이를 시작할 수 있습니다. strace는 프로그램에서 작성된 시스템 호출 및 프로세스에서 수신한 신호를 기록합니다. 각 시스템 호출에 대해 이름, 인수 및 리턴 값이 사용됩니다. strace를 사용하면 소스 없이도 프로그램을 추적할 수 있습니다(다시 컴파일할 필요 없음). strace를 **-f** 옵션과 함께 사용할 경우 포크된 시스템 호출의 결과로 작성된 하위 프로세스를 추적합니다. Plug-in 문제점을 조사하거나 프로그램이 제대로 시작되지 않은 이유를 파악할 때 strace를 사용합니다.

Java 애플리케이션과 함께 strace를 사용하려면 `strace java -Xgcpolicy:metronome <class-name>`을 입력하십시오.

`-o` 옵션을 사용하여 strace 도구를 통한 추적 출력을 파일에 넣을 수 있습니다.

ltrace

ltrace 도구는 배포 의존적으로, strace와 매우 유사합니다. 이 도구는 실행 프로세스에 의해 호출된 대로 동적 라이브러리 호출을 가로채어 기록합니다. strace는 실행 프로세스에 의해 수신되는 신호에 대해 동일합니다.

Java 애플리케이션과 함께 Itrace를 사용하려면 `ltrace java -Xgcpolicy:metronome <class-name>`를 입력하십시오.

mtrace

mtrace는 GNU 도구 세트에 포함되어 있습니다. 이는 malloc, realloc 및 free용 특수 핸들러를 설치하며, 이러한 기능의 모든 사용을 추적하고 파일에 기록하도록 해줍니다. 이 추적은 프로그램 효율성을 떨어뜨리으로 일반 사용 중에는 사용하지 않도록 하십시오. mtrace를 사용하려면 **IBM_MALLOCTRACE**를 1로 설정하고 **MALLOC_TRACE**를 추적 정보가 저장되는 올바른 파일을 가리키도록 설정하십시오. 이 파일에 대한 쓰기 액세스가 있어야 합니다.

Java 애플리케이션과 함께 mtrace를 사용하려면 다음을 입력하십시오.

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

크래쉬 진단

클래쉬 이전의 Java 환경 및 실행 프로세스에 대한 정보를 수집하려면 다음 가이드라인을 수행하십시오.

프로세스 정보 수집

크래쉬가 발생하기 전의 상태를 수집할 때, 코어 파일을 분석하는 대신 **gdb** 및 **bt** 명령을 사용하여 실패한 스레드의 스택 추적을 표시하십시오.

Java 환경 정보 찾기

각 스레드가 어떤 작업을 수행 중이었고 어떤 Java 메소드가 실행 중이었는지 판별하려면 Javadump를 사용하십시오. 다양한 지점에서 실행 중인 코드의 소스를 판별하려면 라이브러리 주소에 대해 함수 주소를 일치시키십시오.

-verbose:gc 옵션을 사용하여 Java 힙의 상태를 확인하십시오. 다음 질문을 고려하십시오.

- 메모리 영역 중 한 곳에 메모리가 부족한지, 그리고 이것이 크래쉬의 원인이 되었는지 여부.
- 가비지 콜렉션 동안 가능한 가비지 콜렉션 결함을 나타내는 크래쉬가 발생했는지 여부.
- 가비지 콜렉션 후에 가능한 메모리 충돌을 나타내는 크래쉬가 발생했는지 여부.

성능 문제 디버깅

성능 문제를 디버깅할 때, IBM SDK for Java V7 사용자 안내서 외에도 이러한 IBM WebSphere Real Time for Linux의 특정 항목을 고려하십시오.

메모리 영역 크기 조정

Java 힙 *크기*는 JVM의 가장 중요한 조정 매개변수 중 하나입니다. 성능을 최적화하려면 올바른 *크기*를 선택하십시오. 올바른 *크기*를 사용하면 가비지 콜렉터가 필요한 용도를 제공하기가 더 쉬워집니다.

메모리 영역의 *크기* 다양화에 대한 자세한 정보는 73 페이지의 『메트로놈 가비지 콜렉터 문제점 해결』의 내용을 참조하십시오.

JIT 컴파일 및 성능

JIT를 사용할 경우, 실시간 동작을 위한 구현을 고려해야 합니다.

Linux의 알려진 제한사항

Linux는 빠르게 진화해 왔으며 그에 따라 특히 스레드 영역에서 JVM과 운영 체제의 상호작용과 관련하여 다양한 문제가 생겨났습니다.

Linux 시스템에 영향을 줄 수 있는 다음 제한사항에 주의하십시오.

프로세스로서의 스레드

Java 스레드의 수가 허용되는 최대 프로세스 수를 초과하는 경우 프로그램이 다음과 같이 될 수 있습니다.

- 오류 메시지를 수신합니다.
- **SIGSEGV** 오류가 발생합니다.
- 중지됩니다.

자세한 정보는 http://www.volano.com/report/index.html의 *Volano 보고서*를 참조하십시오.

플로팅 스택 제한사항

플로팅 스택 없이 실행 중인 경우 **-Xss**에 설정된 값에 관계없이 각 스레드에 대해 최저 256KB의 원본 스택 크기가 제공됩니다.

플로팅 스택 Linux 시스템에서는 **-Xss** 값이 주어집니다. 플로팅이 아닌 Linux 시스템에서 마이그레이션하는 경우, **-Xss** 값이 충분히 큰지 그리고 최소 256KB에 의존하고 있지 않은지 확인하십시오.

glibc 제한사항

__bzero 같은 기호를 찾을 수 없으므로 `libjava.so` 라이브러리를 로드할 수 없다는 메시지가 표시된다는 것은 이전 버전의 GNU C 런타임 라이브러리인 glibc가 설치되어 있다는 의미입니다. Linux 스레드 구현용 SDK를 사용하려면 glibc 버전 2.3.2 이상이 있어야 합니다.

글꼴 제한사항

Red Hat 시스템에 설치할 경우 글꼴 서버로 Java 트루타입 글꼴을 찾고 다음을 실행할 수 있도록 하십시오(예: Linux IA32).

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

이는 설치 시 수행해야 하며 명령을 실행하려면 『루트』로 로그인되어 있어야 합니다. 글꼴 문제에 대한 자세한 정보는 *Linux SDK and Runtime User Guide*를 참조하십시오.

Linux Completely Fair Scheduler가 Java 성능에 미치는 영향

동기화를 광범위하게 사용하는 Java 애플리케이션은 Completely Fair Scheduler가 포함된 Linux 분배에서 성능이 저하될 수 있습니다. CFS(Completely Fair Scheduler)는 릴리스 2.6.23 이상의 주요 Linux 커널에 채택된 스케줄러입니다. CFS 알고리즘은 이전 Linux 릴리스의 예약 알고리즘과 다릅니다. 일부 애플리케이션의 성능 특성을 변경할 수 있습니다. 특히 CFS는 결과 스레드가 CPU 시간과 상관없이 제공되기 쉽도록 sched_yield()를 다르게 구현합니다.

이러한 문제점이 발생하는 경우 CPU 사용도가 높은 Java 애플리케이션을 관찰하여 동기화된 블록을 통해 진행 속도를 낮출 수 있습니다. 진행 속도가 낮아 해당 애플리케이션이 중지된 것처럼 보일 수도 있습니다.

가능한 해결책은 다음 두 가지입니다.

- 추가 인수 **-Xthr:minimizeUserCPU**를 사용하여 JVM을 시작하십시오.
- 이전 버전과 호환성이 더 높은 sched_yield()의 구현을 사용하도록 Linux 커널을 구성하십시오. 다음과 같이 sched_compat_yield 조정 가능 커널 특성을 **1**로 설정하여 이를 수행하십시오.

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

성능이 낮아진 경우 이러한 해결책을 사용하지 마십시오.

이 문제점은 Completely Fair Scheduler가 포함된 Linux 커널에서 실행되는 IBM Developer Kit and Runtime Environment for Linux 5.0(모든 버전) 및 6.0(SR 4 이하의 모든 버전)에 영향을 줄 수 있습니다. Linux 버전 6.0 SR 4 이후의 IBM Developer Kit and Runtime Environment는 커널의 CFS 사용을 감지하고 **-Xthr:minimizeUserCPU** 옵션을 자동으로 사용 가능하게 합니다. Completely Fair Scheduler가 포함된 Linux 분배로는 Ubuntu 8.04 및 SUSE Linux Enterprise Server 11이 있습니다.

CFS에 대한 자세한 정보는 Completely Fair Scheduler를 사용하는 멀티프로세싱에 있습니다.

Linux Red Hat MRG 커널의 성능 문제

Red Hat MRG 커널과 관련된 구성 문제가 발생하면 WebSphere Real Time이 사용 가능한 상세한 가비지 콜렉션을 시작할 때 애플리케이션 스레드가 예기치 않게 일시정지될 수 있습니다. 이러한 일시정지는 상세한 GC 출력에 기록되지는 않지만 네트워크 구성에 따라 수 밀리초 동안 지속될 수 있습니다. 원격으로 정의된 LDAP 사용자로부터 시작된 JVM은 이름 서비스 캐시 디먼(nscd)이 지원되지 않으므로 네트워크 지연에 영향을 줄 수 있습니다. nscd를 시작하여 문제점을 해결하십시오. nscd 서비스의 상태를 확인하고 문제점을 정정하려면 다음 단계를 따르십시오.

1. 다음 명령을 입력하여 nscd 디먼이 실행 중인지 검사하십시오.

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

디먼이 실행되고 있지 않으면 다음 메시지가 표시됩니다.

```
nscd is stopped
```

2. 루트 사용자로서 다음 명령으로 nscd 서비스를 시작하십시오.

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

3. 루트 사용자로서 다음 명령으로 nscd 서비스에 대한 시작 정보를 변경하십시오.

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

nscd 프로세스는 현재 실행 중이며 다시 부팅한 후에 자동으로 시작됩니다.

NLS 문제점 판별

JVM에는 다른 로케일에 대한 기본 제공 지원이 포함되어 있습니다.

IBM SDK for Java V7 사용자 안내서에는 NLS 문제점 진단에 필요한 정보가 있으며 다음 내용을 다루고 있습니다.

- 글꼴 개요
- 글꼴 유틸리티
- 공통 NLS 문제점 및 가능한 원인

이 정보는 IBM SDK for Java 7 - NLS 문제점 판별에서 찾을 수 있습니다.

ORB 문제점 판별

ORB 문제점을 디버깅하는 경우 첫 번째 태스크 중 하나는 문제점이 분산 애플리케이션의 클라이언트 측에 있는지 또는 서버 측에 있는지 여부를 판별하는 것입니다. 일반 RMI-IIOP 세션을 오브젝트 액세스를 요청하는 클라이언트와 액세스를 제공하는 서버 사이의 단순 동기 통신으로 간주하십시오.

IBM SDK for Java V7 사용자 안내서에는 ORB 문제점 진단에 필요한 정보가 있으며 다음 내용을 다루고 있습니다.

- ORB 문제 식별
- 스택 추적 해석
- ORB 추적 해석
- 공통 문제점
- IBM ORB 서비스: 데이터 콜렉트

이 정보는 IBM SDK for Java 7 - ORB 문제점 판별에서 찾을 수 있습니다.

다음 정보는 IBM WebSphere Real Time for Linux에 대한 보충 정보입니다..

IBM ORB 서비스: 데이터 콜렉트

서비스의 Java 버전 출력을 수집할 때 다음 명령을 실행하십시오.

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

사전 테스트

문제가 발생하는 경우 ORB가 다음 사항을 포함하는 org.omg.CORBA.* 예외를 생성할 수 있습니다.

- 원인을 표시하는 텍스트
- 보조 코드
- 완료 상태

ORB가 문제점의 원인이라고 가정하기 전에 다음을 먼저 확인하십시오.

- 유사한 구성으로 시나리오를 재생성할 수 있어야 합니다.
- JIT가 사용 안함으로 설정되어 있어야 합니다.
- AOT 컴파일 코드를 사용하지 않아야 합니다.

기타 조치에는 다음이 포함됩니다.

- 추가 프로세서를 끄십시오.
- 가능한 경우 SMT(Simultaneous Multithreading)를 끄십시오.
- 클라이언트 또는 서버와의 메모리 종속성을 제거하십시오. 실제 메모리 부족으로 인해 성능 저하, 명백한 정지 또는 크래시가 발생할 수 있습니다. 이러한 문제점을 제거하려면 메모리에 적절한 여유 공간이 있는지 확인하십시오.
- 실제 네트워크 문제점(예: 방화벽, 통신 링크, 라우터, DNS 이름 서버 등)을 확인하십시오. 이러한 문제점은 CORBA COMM_FAILURE 예외의 주요 원인입니다. 테스트로 자체 워크스테이션 이름에 대해 Ping을 실행하십시오.
- 애플리케이션이 DB2®와 같은 데이터베이스를 사용하는 경우 가장 신뢰할 수 있는 드라이버로 전환하십시오. 예를 들어, DB2 AppDriver를 분리하려면 속도가 더 느리고 소켓을 사용하지만 더 신뢰할 수 있는 네트 드라이버로 전환하십시오.

OutOfMemory 오류 문제점 해결

OutOfMemoryError 예외을 처리합니다.

메트로놈 가비지 콜렉터를 사용하는데 관한 일반적인 문제점 해결 정보는 73 페이지의 『메트로놈 가비지 콜렉터 문제점 해결』의 내용을 참조하십시오.

OutOfMemoryError 진단

메트로놈 가비지 콜렉터에서 OutOfMemoryError 예외를 진단하는 것은 가비지 콜렉터의 정기적인 특성 때문에 표준 JVM보다 더 복잡할 수 있습니다.

일반적으로 실시간 애플리케이션은 표준 Java 애플리케이션보다 대략 20% 추가 힙 공간을 필요로 합니다.

기본적으로 JVM은 발견되지 않은 OutOfMemoryError 발생 시 다음과 같은 진단 출력을 생성합니다.

- 스냅 덤프: 53 페이지의 『덤프 에이전트 사용』 참조
- 힙 덤프: 61 페이지의 『힙 덤프 사용』 참조
- Java 덤프: 56 페이지의 『Java 덤프 사용』 참조
- 시스템 덤프: 64 페이지의 『시스템 덤프 및 덤프 뷰어 사용』 참조

덤프 파일 이름이 콘솔 출력에 제공됩니다.

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

콘솔 출력에 표시되고 Java 덤프에서도 사용 가능한 Java 백추적은 Java 애플리케이션에서 OutOfMemoryError가 발생한 위치를 나타냅니다. JVM 메모리 관리 구성요소가 할당 실패한 메모리 공간 이름, 크기, 클래스 블록 주소를 제공하는 추적포인트를 발행합니다. 이 추적포인트는 스냅 덤프에 있습니다.

```
<< lines omitted... >>
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
```

추적포인트 ID 및 데이터 필드는 할당하는 오브젝트의 유형에 따라 다를 수 있습니다. 이 예제에서 추적포인트는 애플리케이션이 `Metronome heap`, 메모리 세그먼트 `id=0x809c5f0`에 `class 0x81312d8` 유형의 33.6MB 오브젝트를 할당하려고 할 때 할당 장애가 발생했음을 보여줍니다.

Java 덤프에서 메모리 관리 정보를 확인하여 영향을 받는 메모리 영역을 판별할 수 있습니다.

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

```
1STMEMUSAGE    Total memory in use:    66676824 (0x03F96858)
1STMEMUSAGE    Total memory free:      00432040 (0x000697A8)

<< lines removed for clarity >>
```

Java 덤프의 클래스 섹션을 확인하여 할당되는 오브젝트의 유형을 판별할 수 있습니다.

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

Java 덤프의 정보는 시도된 할당이 정상 힙 (ID=0xF288B584)에서 문자 배열에 대한 것이 해당 1STHEAP 행에 표시된 힙의 총 할당된 *크기*가 67108864 10진수 바이트 또는 0x04000000 16진수 바이트 또는 64MB임을 확인합니다.

이 예제에서 총 힙 *크기*와 비교하여 실패한 할당이 큽니다. 애플리케이션이 33MB 오브젝트를 작성할 것으로 예상되면 다음 단계는 **-Xmx** 옵션을 사용하여 힙의 *크기*를 늘리는 것입니다.

총 힙 *크기*와 비교하여 실패한 할당이 작은 것이 일반적입니다. 힙을 채우는 이전 할당 때문입니다. 이 경우, 다음 단계는 힙 덤프를 사용하여 기존 오브젝트에 할당되는 메모리의 양을 조사하는 것입니다.

힙 덤프는 오브젝트 클래스, *크기* 및 참조를 포함한 모든 오브젝트 목록이 있는 압축된 2진 파일입니다. Java용 IBM 모니터링 및 진단 도구 (메모리 분석기 도구로서 ISA(IBM Support Assistant)에서 다운로드 가능) 를 사용하여 힙 덤프를 분석하십시오.

MDD4J를 사용하여 대량의 힙 공간 이용이 의심되는 오브젝트의 트리 구조를 찾고 힙 덤프를 로드할 수 있습니다. 도구에서 힙의 오브젝트에 대한 다양한 보기를 제공합니다. 예를 들어, MDD4J는 누수 의심에 대해 설명하는 보기를 표시할 수 있으며 힙 크기에 기여한 상위 5개 오브젝트 및 패키지를 알려줍니다. 트리 보기를 선택하면 누수 컨테이너 오브젝트의 특성에 대한 자세한 정보가 제공됩니다.

IBM JVM의 메모리 관리 방식

IBM JVM은 클래스, 컴파일된 코드, Java 오브젝트, Java 스택, JNI 스택용 메모리 영역을 포함하여 다양한 구성요소에 대한 메모리가 필요합니다. 이 메모리 영역 중 일부는 인접 메모리에 있어야 합니다. 기타 메모리 영역은 더 작은 메모리 영역으로 세그먼트화하고 서로 링크할 수 있습니다.

동적으로 로드된 클래스 및 컴파일된 코드는 동적으로 로드된 클래스용으로 세그먼트화된 메모리 영역에 저장됩니다. 클래스는 쓰기 가능 메모리 영역(RAM 클래스) 및 읽기 전용 메모리 영역(ROM 클래스)로 다시 세분화됩니다. 런타임에 클래스 캐시의 ROM

클래스 및 AOT 코드가 맵핑된 메모리이지만 애플리케이션 시작 시 인접 메모리 영역으로 로드되지 않습니다. 애플리케이션이 클래스를 참조하므로 클래스 및 클래스 캐시의 컴파일된 코드가 저장 공간에 맵핑됩니다. 클래스의 ROM 구성요소는 이 클래스를 참조하는 여러 프로세스 간에 공유됩니다. JVM이 클래스를 처음 참조할 때 클래스의 RAM 구성요소가 동적으로 로드된 클래스의 세그먼트화된 메모리 영역에 작성됩니다. 클래스 캐시의 클래스 메소드에 대한 AOT 컴파일된 코드는 프로세스에서 공유하지 않으므로 실행 가능한 동적 코드 메모리 영역에 복사됩니다. 클래스 캐시에서 로드되지 않은 클래스는 ROM 클래스 정보가 동적으로 로드된 클래스의 세그먼트화된 메모리 영역에 작성되는 점을 제외하고 캐싱된 클래스와 비슷합니다. 동적으로 생성된 코드는 캐싱된 클래스의 AOT 코드가 저장된 동일한 동적 코드 메모리 영역에 저장됩니다.

각 Java 스레드의 스택이 세그먼트화된 메모리 영역에 걸쳐 있습니다. 각 스레드의 JNI 스택이 인접 메모리 영역을 차지합니다.

JVM의 구성 방식을 판별하려면 **-verbose:sizes** 옵션을 사용하여 실행하십시오. 이 옵션은 *크기*를 관리할 수 있는 메모리 영역에 대한 정보를 인쇄합니다. 인접하지 않은 메모리 영역의 경우, 영역을 확장해야 될 때마다 획득되는 메모리의 *크기*를 설명하는 증분 정보가 인쇄됩니다.

다음은 **-Xrealtime -verbose:sizes** 옵션을 사용한 예제 출력입니다.

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

이 예제는 RAM 클래스 세그먼트가 초기에 0이고 필요에 따라 32KB 블록씩 확장되는 것을 보여줍니다. ROM 클래스 세그먼트가 초기에 0이고 필요에 따라 128KB 블록씩 확장됩니다. **-Xmca** 및 **-Xmco** 옵션을 사용하면 이 *크기*를 제어할 수 있습니다. 대개 이 옵션을 변경할 필요가 없도록 RAM 클래스 및 ROM 클래스 세그먼트가 필요에 따라 확장됩니다.

클래스 캐시를 사용할 경우 메모리 맵핑된 영역의 *크기*를 판별하려면 **-Xshareclasses** 옵션을 사용하십시오. 다음은 `java -Xgcpolicy:metronome -Xshareclasses:printStats` 명령의 출력 샘플입니다.

```
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
```

런타임에 클래스를 참조할 경우 약 3MB의 AOT 바이트 및 메타데이터 바이트가 동적 코드 세그먼트화된 영역에 복사됩니다. 데이터 바이트는 클래스를 참조할 때 RAM 클래스 세그먼트화된 영역에 복사됩니다.

진단 도구 사용

IBM WebSphere Real Time for Linux JVM 문제점 진단에 사용할 수 있는 진단 도구가 있습니다.

IBM SDK for Java 7은 IBM WebSphere Real Time for Linux JVM 문제점 진단에 사용할 수 있는 진단 도구를 제공합니다. 이 섹션에서는 사용 가능한 도구를 소개하고 해당 도구를 사용하는 데 필요한 추가 정보에 대한 링크를 제공합니다.

SDK 진단 도구 사용 시 기억해야 할 중요한 사항이 있습니다. 실시간 JVM을 호출할 때 다음 옵션을 사용합니다.

```
java  -Xgcpolicy:metronome
```

이 옵션은 실시간 JVM에 대한 진단 도구를 실행할 때 사용해야 합니다. 예를 들어, IBM WebSphere Real Time for Linux의 등록된 덤프 에이전트를 표시하려면 다음을 입력합니다.

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

이 도구를 IBM WebSphere Real Time for Linux와 함께 사용하는 데 관한 차이점은 여기에서 진단을 돕는 샘플 출력과 함께 보충 정보로 제공됩니다.

IBM SDK for Java 7에서 생성된 진단 정보의 요약은 진단 정보 요약을 참조하십시오.

IBM Monitoring and Diagnostic Tools for Java 사용

IBM은 사용자가 IBM JRE를 사용하는 애플리케이션에서 발생할 수 있는 문제점을 이해하고 모니터하며 진단할 수 있도록 도구 및 문서를 제공합니다.

다음 도구를 사용할 수 있습니다.

- Health Center
- GCMV(Garbage Collection and Memory Visualizer)
- IDDE(Interactive Diagnostic Data Explorer)
- Memory Analyzer

GCMV(Garbage Collection and Memory Visualizer)

GCMV(Garbage Collection and Memory Visualizer)를 통해 메모리 사용, 가비지 콜렉션 동작 및 Java 애플리케이션의 성능을 이해할 수 있습니다.

GCMV는 다음을 포함한 다양한 유형의 로그 데이터를 구문 분석하고 구상합니다.

- 상세 가비지 콜렉션 로그.
- -Xtgc 매개변수를 사용하여 생성되는 추적 가비지 콜렉션 로그
- ps, **svmon** 또는 **perfmon** 시스템 명령을 사용하여 생성되는 기본 메모리 로그

이 도구는 메모리 누수와 같은 문제점을 진단하고 다양한 비주얼 형식의 데이터를 분석하며 조정 권장사항을 제공합니다.

GCMV는 ISA(IBM Support Assistant) 추가 기능으로 제공됩니다. 추가 기능의 설치 및 시작에 대한 정보는 http://www.ibm.com/developerworks/java/jdk/tools/gcmv/의 내용을 참조하십시오.

GCMV에 대한 더 자세한 정보는 IBM Information Center에서 확인할 수 있습니다.

Health Center

Health Center는 실행 중인 JVM(Java Virtual Machine)의 상태를 모니터링하는 진단 도구입니다.

이 도구는 다음 두 가지 파트로 제공됩니다.

- 실행 중인 애플리케이션에서 데이터를 수집하는 Health Center 에이전트
- 에이전트에 연결하는 Eclipse 기반 클라이언트. 이 클라이언트는 모니터되는 애플리케이션의 성능을 향상시키도록 데이터를 해석하고 권장사항을 제공합니다.

Health Center는 ISA(IBM Support Assistant) 추가 기능으로 제공됩니다. 추가 기능의 설치 및 시작에 대한 정보는 http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/의 내용을 참조하십시오.

Health Center에 대한 더 자세한 정보는 IBM Information Center에서 확인할 수 있습니다.

IDDE(Interactive Diagnostic Data Explorer)

IDDE(Interactive Diagnostic Data Explorer)는 덤프 뷰어(**jdmpview** 명령)의 GUI 기반 대안입니다. IDDE는 덤프 뷰어와 동일한 기능을 제공하지만 명령 출력을 저장하는 기능과 같은 추가 지원을 포함합니다.

JVM에서 생성되는 덤프 파일을 보다 쉽게 탐색하고 검사하려면 IDDE를 사용하십시오. IDDE 내에서 조사 로그에 명령을 입력하여 덤프 파일을 탐색합니다. 조사 로그에서 제공되는 지원에는 다음 항목이 포함됩니다.

- 명령 지원
- 텍스트 및 일부 매개변수(예: 클래스 이름)의 자동 완성
- 이후 다른 사용자에게 전송할 수 있도록 명령 및 출력을 저장하는 기능
- 문제의 플래그 지정 및 강조표시된 텍스트
- 자체 주석을 추가할 수 있는 기능
- IDDE 내에서 Memory Analyzer 사용 지원

IDDE는 ISA(IBM Support Assistant) 추가 기능으로 제공됩니다. 추가 기능 설치 및 시작에 대한 정보는 developerWorks®의 IDDE 개요를 참조하십시오.

IDDE에 대한 자세한 정보는 IBM Information Center에서 확인할 수 있습니다.

Memory Analyzer

Memory Analyzer를 사용하여 운영 체제 레벨 덤프 및 이동식 힙 덤프(PHD, Portable Heap Dumps)를 사용하는 Java 힙을 분석할 수 있습니다.

이 도구는 수백만 개의 오브젝트가 포함된 덤프를 분석할 수 있으며, 다음 정보를 제공합니다.

- 보유한 오브젝트의 크기
- 가비지 콜렉터의 오브젝트 수집을 방지하는 프로세스
- 누수 의심을 자동으로 추출하는 보고서

이 도구는 Eclipse Memory Analyzer(MAT) 프로젝트 기반이며 IBM의 JAVA 진단 도구 프레임워크(DTFJ, Diagnostic Tool Framework for Java) 기능을 사용하여 IBM JVM에서 덤프를 처리합니다.

Memory Analyzer는 ISA(IBM Support Assistant)의 추가 기능입니다. 추가 기능의 설치 및 시작에 대한 정보는 http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/의 내용을 참조하십시오.

Memory Analyzer에 대한 더 자세한 정보는 IBM Information Center에서 확인할 수 있습니다.

덤프 에이전트 사용

JVM 초기화 중에 덤프 에이전트를 설정합니다. 덤프 에이전트를 통해 JVM에서 발생하는 이벤트(예: 가비지 콜렉션, 스레드 시작 또는 JVM 종료)를 사용하고 덤프를 시작하거나 외부 도구를 시작할 수 있습니다.

IBM SDK for Java V7 사용자 안내서에는 덤프 에이전트에 대한 유용한 정보를 포함하고 있으며 다음 내용을 다루고 있습니다.

- **-Xdump** 옵션 사용
- 덤프 에이전트
- 덤프 에이전트
- 덤프 에이전트 고급 제어
- 덤프 에이전트 토큰
- 기본 덤프 에이전트
- 덤프 에이전트 제거
- 덤프 에이전트 환경 변수
- 신호 맵핑
- 덤프 에이전트 기본 위치

이 정보는 IBM SDK for Java 7 - 덤프 에이전트 사용에서 찾을 수 있습니다.

IBM WebSphere Real Time for Linux의 보충 정보는 다음 위치에 있습니다.

덤프 에이전트

덤프 에이전트는 JVM 운영 중에 발생하는 이벤트에 의해 트리거됩니다. IBM WebSphere Real Time for Linux의 경우, 느린 이벤트의 기본값은 5밀리초입니다.

일부 이벤트를 필터하여 출력의 연관성을 향상시킬 수 있습니다. 자세한 정보는 웹 사이트 54 페이지의 『필터 옵션』의 내용을 참조하십시오.

참고: 로드 해제 및 확장 이벤트는 현재 WebSphere Real Time에서 발생하지 않습니다. 클래스는 영구 메모리 상태이며 로드 해제될 수 없습니다.

참고: gpf 및 중단 이벤트는 힙 덤프를 트리거하지 않거나, 힙을 준비(request=prepwalk) 또는 압축(request=compact)하지 않습니다.

다음 테이블은 덤프 에이전트 트리거로 사용할 수 있는 이벤트를 표시합니다.

이벤트	트리거하는 경우	필터 작업
gpf	GPF(General Protection Fault)가 발생함	
user	JVM이 운영 체제에서 SIGQUIT 신호를 수신합니다.	
abort	JVM이 운영 체제에서 IGABRT 신호를 수신함	
vmstart	가상 머신이 시작됨	
vmstop	가상 머신이 중지됨	종료 코드에 대해 필터링(예: **filter=#129..#192#-42#255**)
load	클래스가 로드됨	클래스 이름에 대해 필터링(예:**filter=java/lang/String**)
unload	클래스가 언로드됨	
throw	예외가 발생함	예외 클래스 이름에 대해 필터링(예: **filter=java/lang/OutOfMem***)
catch	예외를 발견함	예외 클래스 이름에 대해 필터링(예: **filter=*Memory***)
uncaught	애플리케이션에서 Java 예외를 발견하지 못함	예외 클래스 이름에 대해 필터링(예: **filter=*MemoryError**)
systhrow	JVM에서 Java 예외가 발생함. 이 이벤트는 JVM 내부에서 발견한 오류 조건에 대해서만 트리거되므로 'throw' 이벤트와는 다름니다.	예외 클래스 이름에 대해 필터링(예: **filter=java/lang/OutOfMem***)
thrstart	새로운 스레드가 시작됨	
blocked	스레드가 블록됨	
thrstop	스레드가 중지됨	
fullgc	가비지 콜렉션 순환이 시작됨	
slow	스레드가 내부 JVM 요청에 대한 응답에 5ms 이상을 소요합니다.	'slow'로 간주하는 이벤트 소요 시간을 변경할 수 있습니다. 예를 들어 **filter=#300ms**는 내부 JVM 요청에 대한 응답에 스레드가 300ms 시간 이상 소요하면 트리거합니다.
allocation	Java 오브젝트가 주어진 필터 스펙과 일치하는 *크기*로 할당됨	오브젝트 *크기*에 대해 필터링하고 필터를 제공해야 합니다. 예를 들어 **filter=#5m**는 5MB 이상인 오브젝트를 트리거합니다. **filter=#256k..512k**는 *크기*가 256KB에서 512KB 사이인 오브젝트에 대해 트리거합니다.
traceassert	내부 오류가 JVM에서 발생합니다.	적용할 수 없습니다.
corruptcache	JVM에서 공유 클래스 캐시의 손상을 발견합니다.	적용할 수 없습니다.

필터 옵션

일부 JVM 이벤트는 애플리케이션 실행 시간 중 여러 번 발생합니다. 덤프 에이전트는 필터 및 범위를 사용하여 초과 덤프가 생성되지 않도록 방지합니다.

와일드 카드

필터의 시작 또는 끝에만 별표를 지정하여 예외 이벤트 필터에서 와일드 카드를 사용할 수 있습니다. 다음 명령은 두 번째 별표가 끝에 있지 않으므로 작동하지 않습니다.

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

이 필터를 작동하게 하려면 다음과 같이 변경되어야 합니다.

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

클래스 로딩 및 예외 이벤트

클래스 로딩(load) 및 예외(throw, catch, uncaught, systhrow) 이벤트를 Java 클래스 이름에 따라 필터할 수 있습니다.

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

throw, uncaught 및 systhrow 예외 이벤트를 Java 메소드 이름에 따라 필터링할 수 있습니다.

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

선택적 부분은 대괄호로 표시되어 있습니다.

발견 예외 이벤트를 Java 메소드 이름에 따라 필터링할 수 있습니다.

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

선택적 부분은 대괄호로 표시되어 있습니다.

vmstop 이벤트

하나 이상의 종료 코드를 사용하여 JVM 시스템 종료 이벤트를 필터할 수 있습니다.

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

느린 이벤트

느린 이벤트를 필터링하여 기본 5ms에서 시간 임계값을 변경할 수 있습니다.

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

사용자는 필터를 기본 시간보다 낮은 시간으로 설정할 수 없습니다.

할당 이벤트

트리거를 발생시키는 오브젝트의 *크기*를 지정하려면 할당 이벤트를 필터해야 합니다. 필터 *크기*는 32비트 플랫폼의 경우 0에서 최대값인 32비트 포인터까지 64비트 플랫폼의 경우 최대값인 64비트 포인터까지 설정할 수 있습니다. 필터 값을 0으로 낮게 설정하면 모든 할당에 대해 덤프가 트리거됩니다.

예를 들어, *크기*가 5Mb를 초과하는 할당에 대해 덤프를 트리거하려면 다음을 사용하십시오.

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

크기가 256Kb - 512Kb 사이인 할당에 대해 덤프를 트리거하려면 다음을 사용하십시오.

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

기타 이벤트

필터링을 지원하지 않는 이벤트에 필터를 적용하면 해당 필터가 무시됩니다.

Java 덤프 사용

Java 덤프는 JVM 및 Java 애플리케이션과 관련하여 실행 중 한 지점에서 수집한 진단 정보를 포함하는 파일을 생성합니다. 예를 들어 운영 체제, 애플리케이션 환경, 스레드, 스택, 잠금 및 메모리에 관한 정보가 포함될 수 있습니다.

IBM SDK for Java V7 사용자 안내서에는 Java 덤프에 대한 유용한 정보를 포함하고 있으며 다음 내용을 다루고 있습니다.

- Java 사용
- Java 덤프 트리거
- Javadump 해석
- 환경 변수 및 Java 덤프

이 정보는 IBM SDK for Java 7 - Java 덤프 사용에서 찾을 수 있습니다.

다음 주제에 IBM WebSphere Real Time for Linux의 보충 정보 및 샘플 출력이 있습니다.

스토리지 관리(MEMINFO)

MEMINFO 섹션에서는 힙, 영구 및 범위 메모리 영역을 포함한 Memory Manager 관련 정보를 제공합니다.

Java 덤프의 MEMINFO 섹션은 Memory Manager에 대한 정보를 표시합니다. Memory Manager 컴포넌트 작동 방식에 대한 세부사항은 메트로놈 가비지 콜렉터 사용의 내용을 참조하십시오.

Java 덤프의 이 파트는 다음을 포함한 다양한 스토리지 관리 값을 제공합니다.

- 사용 가능한 메모리 용량
- 사용된 메모리 용량
- 현재 힙 크기
- 현재 영구 메모리 영역 크기
- 현재 범위 메모리 영역 크기

이 섹션에는 가비지 콜렉션 히스토리 데이터도 있습니다. 가비지 콜렉션 히스토리 데이터는 추적점 시퀀스로 표시되며 각 추적점에는 시간소인이 있고 가장 최근 추적점이 맨 처음에 오도록 정렬되어 있습니다.

표준 JVM에서 생성된 Javadump에는『GC 히스토리』섹션이 있습니다. 이러한 정보는 실시간 JVM 사용 시 생성되는 Javadump에는 포함되지 않습니다. **-verbose:gc** 옵션 또는 JVM 스냅 추적을 사용하여 GC 동작에 대한 정보를 얻으십시오. 자세한 내용은 IBM SDK for Java V7 사용자 안내서의 73 페이지의『verbose:gc 정보 사용』및 덤프 에이전트 섹션을 참조하십시오.

Java 덤프에서 세그먼트는 대량의 메모리를 사용하는 태스크를 위해 Java Runtime이 할당한 메모리 블록입니다. 예제 태스크는 다음과 같습니다.

- JIT 캐시 유지보수
- Java 클래스 저장

또한 Java Runtime Environment는 MEMINFO 섹션에 나열되지 않은 다른 원시 메모리를 할당합니다. Java Runtime 세그먼트에 사용되는 총 메모리가 Java Runtime의 전체 메모리 풋프린트를 반드시 나타내는 것은 아닙니다. Java Runtime 세그먼트는 세그먼트 데이터 구조 및 원시 메모리의 연관된 블록으로 구성됩니다.

다음은 일반 출력의 예입니다. 모든 값이 16진 값으로 제공됩니다. MEMINFO 섹션의 열 표제는 다음 의미를 가집니다.

- 오브젝트 메모리 섹션(HEAPTYPE):

  id 공간 또는 영역ID입니다.

  start 힙에 있는 이 영역의 시작 주소입니다.

  end 힙에 있는 이 영역의 끝 주소입니다.

  size 힙에 있는 이 영역의 크기입니다.

  space/region
  id 및 이름만 포함하는 행의 경우 이 열에는 메모리 공간 이름이 표시됩니다. 그렇지 않으면 열에 메모리 공간 내에 포함된 특정 영역의 이름이 뒤에 있는 메모리 공간 이름이 표시됩니다.

- 클래스 메모리, JIT 코드 캐시 및 JIT 데이터 캐시를 포함한 내부 메모리 섹션 (SEGTYPE):

  segment
  세그먼트 제어 데이터 구조의 주소입니다.

  start 원시 메모리 세그먼트의 시작 주소입니다.

  alloc 원시 메모리 세그먼트 내의 현재 할당 주소입니다.

  end 원시 메모리 세그먼트의 끝 주소입니다.

type 원시 메모리 세그먼트의 특성을 설명하는 내부 비트 필드입니다.

size 원시 메모리의 크기입니다.

```
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STHEAPTYPE    Object Memory
NULL           id         start      end        size       space/region
1STHEAPSPACE   0x00497030     --         --         --     Generational
1STHEAPREGION  0x004A24F0 0x02850000 0x05850000 0x03000000 Generational/Tenured Region
1STHEAPREGION  0x004A2468 0x05850000 0x06050000 0x00800000 Generational/Nursery Region
1STHEAPREGION  0x004A23E0 0x06050000 0x06850000 0x00800000 Generational/Nursery Region
NULL
1STHEAPTOTAL   Total memory:           67108864 (0x04000000)
1STHEAPINUSE   Total memory in use:    33973024 (0x02066320)
1STHEAPFREE    Total memory free:      33135840 (0x01F99CE0)
NULL
1STSEGTYPE     Internal Memory
NULL segment start alloc end type size
1STSEGMENT     0x073DFC9C 0x0761B090 0x0761B090 0x0762B090 0x01000040 0x00010000
       (lines removed for clarity)
1STSEGMENT     0x00497238 0x004FA220 0x004FA220 0x0050A220 0x00800040 0x00010000
NULL
1STSEGTOTAL    Total memory:           873412 (0x000D53C4)
1STSEGINUSE    Total memory in use:         0 (0x00000000)
1STSEGFREE     Total memory free:      873412 (0x000D53C4)
NULL
1STSEGTYPE     Class Memory
NULL segment start alloc end type size
1STSEGMENT     0x0731C858 0x0745C098 0x07464098 0x07464098 0x00010040 0x00008000
        (lines removed for clarity)
1STSEGMENT     0x00498470 0x070079C8 0x07026DC0 0x070279C8 0x00020040 0x00020000
NULL
1STSEGTOTAL    Total memory:          2067100 (0x001F8A9C)
1STSEGINUSE    Total memory in use:   1839596 (0x001C11EC)
1STSEGFREE     Total memory free:     227504 (0x000378B0)
NULL
1STSEGTYPE     JIT Code Cache
NULL segment start alloc end type size
1STSEGMENT     0x004F9168 0x06960000 0x069E0000 0x069E0000 0x00000068 0x00080000
NULL
1STSEGTOTAL    Total memory:           524288 (0x00080000)
1STSEGINUSE    Total memory in use:    524288 (0x00080000)
1STSEGFREE     Total memory free:           0 (0x00000000)
NULL
1STSEGTYPE     JIT Data Cache
NULL segment start alloc end type size
1STSEGMENT     0x004F92E0 0x06A60038 0x06A6839C 0x06AE0038 0x00000048 0x00080000
NULL
1STSEGTOTAL    Total memory:           524288 (0x00080000)
1STSEGINUSE    Total memory in use:     33636 (0x00008364)
1STSEGFREE     Total memory free:      490652 (0x00077C9C)
NULL
1STGCHTYPE     GC History
3STHSTTYPE     15:18:14:901108829 GMT j9mm.134 -   Allocation failure end: newspace=7356368/8388608
oldspace=32038168/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:901104380 GMT j9mm.470 -   Allocation failure cycle end: newspace=7356416/8388608
oldspace=32038168/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:901097193 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=0 failedflipbytes=0 failedtenurecount=0
failedtenurebytes=0 flipcount=11454 flipbytes=991056 newspace=7356416/8388608 oldspace=32038168/50331648
loa=3523072/3523072 tenureage=1
3STHSTTYPE     15:18:14:901081108 GMT j9mm.140 -   Tilt ratio: 50
3STHSTTYPE     15:18:14:893358658 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=24 weakrefs=0
soft=0 phantom=0 finalizers=0
3STHSTTYPE     15:18:14:893354551 GMT j9mm.63 -   Set scavenger backout flag=false
3STHSTTYPE     15:18:14:893348733 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.002
meanexclusiveaccessms=0.002 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
3STHSTTYPE     15:18:14:893348391 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/8388608
oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
3STHSTTYPE     15:18:14:893347364 GMT j9mm.133 -   Allocation failure start: newspace=0/8388608
oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
3STHSTTYPE     15:18:14:866523613 GMT j9mm.134 -   Allocation failure end: newspace=2359064/8388608
oldspace=38199368/50331648 loa=3523072/3523072
```

```
3STHSTTYPE     15:18:14:866519507 GMT j9mm.470 -   Allocation failure cycle end: newspace=2359296/8388608
oldspace=38199368/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:866513004 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=5056 failedflipbytes=445632
failedtenurecount=0 failedtenurebytes=0 flipcount=9212 flipbytes=6017148 newspace=2359296/8388608
oldspace=38199368/50331648 loa=3523072/3523072 tenureage=1
3STHSTTYPE     15:18:14:866493839 GMT j9mm.140 -   Tilt ratio: 64
3STHSTTYPE     15:18:14:859814852 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=23 weakrefs=0
soft=0 phantom=0 finalizers=0
3STHSTTYPE     15:18:14:859808692 GMT j9mm.63 -   Set scavenger backout flag=false
3STHSTTYPE     15:18:14:859801848 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.004
meanexclusiveaccessms=0.004 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
3STHSTTYPE     15:18:14:859801163 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/10747904
oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
3STHSTTYPE     15:18:14:859800479 GMT j9mm.133 -   Allocation failure start: newspace=0/10747904
oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
3STHSTTYPE     15:18:14:652219028 GMT j9mm.134 -   Allocation failure end: newspace=2868224/10747904
oldspace=38985800/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:650796714 GMT j9mm.470 -   Allocation failure cycle end: newspace=2868224/10747904
oldspace=38985800/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:650792607 GMT j9mm.475 -   GlobalGC end: workstackoverflow=0 overflowcount=0
memory=41854024/61079552
3STHSTTYPE     15:18:14:650784052 GMT j9mm.90 -   GlobalGC collect complete
3STHSTTYPE     15:18:14:650780971 GMT j9mm.57 -   Sweep end
3STHSTTYPE     15:18:14:650611567 GMT j9mm.56 -   Sweep start
3STHSTTYPE     15:18:14:650610540 GMT j9mm.55 -   Mark end
3STHSTTYPE     15:18:14:645222792 GMT j9mm.54 -   Mark start
3STHSTTYPE     15:18:14:645216632 GMT j9mm.474 -   GlobalGC start: globalcount=2
           (lines removed for clarity)
NULL
NULL           ------------------------------------------------------------------------
```

스레드 및 스택 추적(THREADS)

애플리케이션 프로그래머에게 가장 유용한 Java 덤프 항목 중 하나는 THREADS 섹션입니다. 이 섹션에는 Java 스레드, 원시 스레드 및 스택 추적에 대한 목록이 있습니다.

Java 스레드는 운영 체제의 원시 스레드가 구현합니다. 각 스레드는 다음과 같이 일련의 행으로 표시됩니다.

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

ps 명령을 사용할 때 Java 스레드 이름이 운영 체제에 표시됩니다. **ps** 명령 사용에 대한 추가 정보는 40 페이지의 『일반 디버깅 기술』의 내용을 참조하십시오.

첫 번째 행의 특성은 스레드 이름, JVM 스레드 구조 및 Java 스레드 오브젝트의 주소, 스레드 상태, Java 스레드 우선순위입니다. 두 번째 행의 특성은 원시 운영 체제 스레드 ID, 원시 운영 체제 스레드 우선순위 및 원시 운영 체제 스케줄링 정책입니다.

스레드 이름은 세 가지 방법으로 표시됩니다.

- javacore 파일에 표시됩니다. 모든 파일이 javacore 파일에 나타나는 것은 아닙니다.
- **ps** 명령을 사용하는 운영 체제에서 스레드를 나열하는 경우
- java.lang.Thread.getName() 메소드를 사용하는 경우

다음 테이블에서 IBM WebSphere Real Time for Linux 스레드 이름에 대한 정보가 표시됩니다.

표 4. IBM WebSphere Real Time for Linux의 스레드 이름

스레드 세부사항	스레드 이름
2차 스레드에 의한 오브젝트 완료를 디스패치하기 위해 가비지 콜렉션이 사용하는 내부 JVM 스레드	`Finalizer master`
가비지 콜렉터가 사용하는 알람 스레드	`GC Alarm`
가비지 콜렉션을 위해 사용되는 Slave 스레드	`GC Slave`
애플리케이션에서 메소드의 사용을 샘플링하기 위해 JIT(just-in-time) 컴파일러 모듈이 사용하는 내부 JVM 스레드	`IProfiler`
내부 또는 외부적으로 생성되는 애플리케이션에 의해 수신된 신호를 관리하기 위해 VM이 사용하는 스레드	`Signal Reporter`
Java 코드를 컴파일하는 데 사용되는 내부 JVM 스레드입니다.	`JIT 컴파일 스레드`
JVMTI 에이전트가 실행 중인 JVM에 접속하도록 허용하는 데 사용되는 내부 JVM 스레드입니다.	`접속 API 대기 루프`

Java 스레드 우선순위는 플랫폼에 기반을 둔 방식으로 운영 체제 우선순위 값에 맵핑됩니다. Java 스레드 우선순위에서 큰 값은 스레드의 우선순위가 높음을 의미합니다. 즉, 이러한 스레드는 우선순위가 낮은 스레드보다 더 자주 실행됩니다.

상태 값이 될 수 있는 것은 다음과 같습니다.

- R - Runnable - 기회가 되면 스레드를 실행할 수 있습니다.
- CW - Condition Wait - 스레드가 대기 중입니다. 대기 원인의 예를 들면 다음과 같습니다.
  - sleep() 호출이 작성됨
  - 스레드가 I/O에 대해 블록됨
  - wait() 메소드가 호출되어 알림을 받는 모니터에서 대기함
  - 스레드가 join() 호출을 사용하여 다른 스레드와 동기화 중임
- S – Suspended – 스레드가 다른 스레드에 의해 일시중단되었습니다.
- Z – Zombie – 스레드가 강제 종료되었습니다.
- P – Parked – 새 동시성 API(java.util.concurrent)에 의해 스레드가 중지되었습니다.
- B – Blocked – 스레드가 다른 항목이 현재 소유하고 있는 잠금을 얻기 위해 대기 중입니다.

스레드가 대기 또는 차단 상태인 경우, 출력에서 3XMTHREADBLOCK으로 시작하는 해당 스레드 행에는 스레드가 기다리는 자원 및 가능한 경우 해당 자원을 현재 소유하는 스레드가 나열됩니다. 자세한 정보는 IBM SDK for Java V7 사용자 안내서에 있는 차단된 스레드 관련 주제를 참조하십시오.

진단 정보를 얻기 위해 java 덤프를 시작하면 JVM이 javacore를 생성하기 전에 Java 스레드를 중지합니다. exclusive_vm_access의 준비 상태는 TITLE 섹션의 1TIPREPSTATE 행에 표시됩니다.

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

javacore가 트리거되었을 때 Java 코드를 실행하던 스레드는 CW(조건 대기) 상태입니다.

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

javacore LOCKS 섹션은 이 스레드가 내부 JVM 잠금을 대기 중임을 표시합니다.

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

힙 덤프 사용

힙 덤프는 IBM Virtual Machine for Java 메커니즘을 설명하는 용어로, Java 힙에 있는 모든 활성 오브젝트 즉, 실행 중인 Java 애플리케이션에 사용되는 오브젝트의 덤프를 생성하는 메커니즘을 설명합니다.

IBM SDK for Java V7 사용자 안내서에는 힙 덤프에 대한 유용한 정보를 포함하고 있으며 다음 내용을 다루고 있습니다.

- 힙 덤프 가져오기
- 힙 덤프 처리에 사용되는 도구
- 힙 정보를 얻기 위해 **-Xverbose:gc** 사용
- 환경 변수 및 힙 덤프
- 텍스트(classic) 힙 덤프 파일 형식
- PHD(Portable Heap Dump) 파일 형식

이 정보는 IBM SDK for Java 7 - 힙 덤프 사용하기에서 찾을 수 있습니다.

IBM WebSphere Real Time for Linux 보충 정보:

텍스트(classic) 힙 덤프 파일 형식

텍스트 또는 classic 힙 덤프는 오브젝트 유형, 크기 및 오브젝트 간 참조를 포함한 힙에 있는 모든 오브젝트 인스턴스의 목록입니다.

헤더 레코드

헤더 레코드는 버전 정보 문자열이 포함된 단일 레코드입니다.

```
// Version: <version string containing SDK level, platform and JVM build level>
```

예:

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

오브젝트 레코드

오브젝트 레코드는 힙의 각 오브젝트 인스턴스마다 하나씩 있는 다중 레코드입니다. 오브젝트 주소, 크기, 유형 및 참조를 오브젝트에서 제공합니다.

```
<object address, in hexadecimal> [<length in bytes of object instance, in decimal>]
OBJ <object type> <class block reference, in hexadecimal>
<heap reference, in hexadecimal <heap reference, in hexadecimal> ...
```

오브젝트 주소 및 힙 참조는 힙에 있으나 클래스 블록 주소는 힙 외부에 있습니다. 널 값인 참조를 포함하여 오브젝트 인스턴스에서 발견된 모든 참조가 나열됩니다. 오브젝트 유형은 패키지가 포함된 클래스 이름이거나 표준 JVM 유형 서명으로 표시되는 클래스 배열 유형 또는 기본 배열입니다. 64 페이지의 『Java VM 유형 서명』의 내용을 참조하십시오. 리플렉션 클래스 인스턴스의 경우 일반적으로 오브젝트 레코드에 추가 클래스 블록 참조도 포함될 수 있습니다.

예:

다음은 길이가 28바이트인 java/lang/String 유형의 오브젝트 인스턴스입니다.

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

java/lang/String의 클래스 블록 주소 뒤에는 문자 배열 인스턴스에 대한 참조가 옵니다.

```
0x415319D8 0x00436EB0
```

다음은 길이가 44바이트인 문자 배열 유형의 오브젝트 인스턴스입니다.

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

다음은 문자 배열의 클래스 블록 주소입니다.

```
0x41530F20
```

다음은 java/util/Hashtable 항목 내부 클래스 배열 유형의 오브젝트입니다.

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

다음은 java/util/Hashtable 항목 내부 클래스 유형의 오브젝트입니다.

```
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
```

다음은 클래스 블록 주소 및 힙 참조(널 참조 포함)입니다.

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

클래스 레코드

클래스 레코드는 로드된 클래스마다 하나씩 있는 다중 레코드입니다. 클래스 블록 주소, 크기, 유형 및 참조를 클래스에서 제공합니다.

```
<class block address, in hexadecimal> [<length in bytes of class block, in decimal>]
CLS <class type>
<class block reference, in hexadecimal> <class block reference, in hexadecimal> ...
<heap reference, in hexadecimal> <heap reference, in hexadecimal>...
```

클래스 블록 주소 및 클래스 블록 참조는 힙 외부에 있으나 일반적으로 static 클래스 데이터 멤버인 경우 클래스 레코드에도 힙에 대한 참조가 포함될 수 있습니다. 널값인 참조를 포함하여 클래스 블록에서 발견된 모든 참조가 나열됩니다. 클래스 유형은 패키지가 포함된 클래스 이름이거나 표준 JVM 유형 서명으로 표시되는 클래스 배열 유형 또는 기본 배열입니다. 64 페이지의 『Java VM 유형 서명』의 내용을 참조하십시오.

예:

다음은 길이가 32바이트인 java/lang/Runnable 클래스의 클래스 블록입니다.

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

다음은 다른 클래스 블록에 대한 참조 및 힙 참조(널 참조 포함)입니다.

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

다음은 길이가 168바이트인 java/lang/Math 클래스의 클래스 블록입니다.

```
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
```

트레일러 레코드 1

트레일러 레코드 1은 레코드 계수가 포함된 단일 레코드입니다.

```
// Breakdown - Classes: <class record count, in decimal>,
Objects: <object record count, in decimal>,
ObjectArrays: <object array record count, in decimal>,
PrimitiveArrays: <primitive array record count, in decimal>
```

예:

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

트레일러 레코드 2

트레일러 레코드 2는 총계가 포함된 단일 레코드입니다.

```
// EOF:  Total 'Objects',Refs(null) :
<total object count, in decimal>,
<total reference count, in decimal>
(,total null reference count, in decimal>)
```

예:

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

Java VM 유형 서명

Java VM 유형 서명은 Java 유형의 약어이며 아래 테이블에 표시되어 있습니다.

Java VM 유형 서명	**Java 유형**
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L <*fully qualified-class*> ;	<*fully qualified-class*>
[<*type*>	<*type*>[ ] (array of <*type*>)
(<*arg-types*>) <*ret-type*>	메소드

시스템 덤프 및 덤프 뷰어 사용

JVM은 구성 가능한 조건에서 원시 시스템 덤프(코어 덤프)를 생성할 수 있습니다. 시스템 덤프는 일반적으로 상당히 큽니다. 시스템 덤프를 분석하는 데 사용하는 대부분의 도구는 또한 특정 플랫폼을 사용합니다.Linux에서 시스템 덤프를 분석하려면 **gdb** 도구를 사용하십시오.

IBM SDK for Java V7 사용자 안내서에는 시스템 덤프 및 덤프 뷰어 사용에 대한 유용한 정보가 있으며 다음 내용을 다루고 있습니다.

- 시스템 덤프 개요
- 시스템 덤프 기본값
- 덤프 뷰어 사용
  - **jextract** 사용

- 덤프 뷰어를 사용하여 다룰 문제점
- **jdmpview**에서 사용 가능한 명령
- 세션 예
- **jdmpview** 명령 빠른 참조

이 정보는 IBM SDK for Java 7 - 시스템 덤프 및 덤프 뷰어 사용에서 찾을 수 있습니다.

IBM WebSphere Real Time for Linux 보충 정보:

jdmpview에서 사용 가능한 명령

jdmpview는 JVM 시스템 덤프에서 정보를 탐색하고 다양한 분석 기능을 수행하는 대화식 명령행 도구입니다.

info jitm

AOT 및 JIT 컴파일 메소드와 해당 주소를 표시합니다.

- 메소드 이름 및 서명
- 메소드 시작 주소
- 메소드 끝 주소

기타 모든 명령 옵션은 IBM SDK for Java V7 사용자 안내서를 참조하십시오.

Java 애플리케이션 및 JVM 추적

JVM 추적은 IBM WebSphere Real Time for Linux에서 제공하는 추적 기능으로 성능에 최소한의 영향을 줍니다. 대부분 경우, 압축 데이터는 압축 2진 형식으로 보관되며 제공된 Java 포맷터로 포맷할 수 있습니다.

추적은 기본적으로 사용되며 메모리 버퍼로 이동하는 소량의 추적점이 함께 사용됩니다. 런타임 시 레벨, 컴포넌트, 그룹 이름 또는 개별 추적점 ID를 사용하여 추적점을 사용할 수 있습니다.

IBM SDK for Java V7 사용자 안내서에는 애플리케이션 추적에 대한 자세한 정보가 있으며 다음 내용을 다루고 있습니다.

- 추적 대상
- 추적 지점 유형
- 기본 추적
- 추적 데이터 레코딩
- 추적 제어
- Java 애플리케이션 추적
- Java 메소드 추적

IBM WebSphere Real Time for Linux 추적 시, 추적 옵션을 포함하는 경우 실시간 JVM을 올바르게 호출해야 합니다. 예를 들어, 추적 옵션을 지정하는 경우 다음을 입력해야 합니다.

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

IBM SDK for Java V7 정보는 Java 애플리케이션 및 JVM 추적에서 찾을 수 있습니다.

JIT 및 AOT 문제점 판별

명령행 옵션을 사용하여 JIT 및 AOT 컴파일러의 문제점을 진단하고 성능을 조정할 수 있습니다.

IBM WebSphere Real Time for Linux가 일부 공통 컴포넌트를 IBM SDK for Java V7과 공유해도 JIT와 AOT의 동작은 서로 다릅니다. 이 섹션에서는 IBM WebSphere Real Time for Linux의 JIT 및 AOT 문제점 해결에 대해 설명합니다.

JIT 또는 AOT 문제점 진단

때때로 올바른 바이트 코드를 올바르지 않은 원시 코드로 컴파일한 경우 Java 프로그램이 실패할 수 있습니다. JIT 또는 AOT 컴파일러의 장애 유무를 판별하여 해당 장애가 있는 *대상*에 Java 서비스 팀 지원을 제공할 수 있습니다.

이 태스크 정보

공유 클래스 캐시가 채워졌을 때 컴파일된 메소드를 판별하려면 관리 캐시 명령행에서 **-Xaot:verbose** 옵션을 사용하십시오. 예를 들면 다음과 같습니다.

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

본 절에서는 컴파일러 관련 문제일 경우 판별하는 방법을 설명합니다. 이 절에서는 또한 가능한 몇몇 해결 방법 및 컴파일러 관련 문제 해결을 위한 디버깅 기술을 제안합니다.

JIT 또는 AOT 컴파일러 사용 안함:

JIT 또는 AOT 컴파일러에서 문제점이 발생하는 것으로 의심되면 컴파일을 사용하지 않아도 문제가 지속되는지 확인하십시오. 문제가 여전히 발생하면 컴파일러가 원인이 아님을 알 수 있습니다.

이 태스크 정보

기본적으로 JIT 컴파일러는 사용 가능합니다. AOT 컴파일러도 사용 가능하지만 공유 클래스가 사용 가능해야 활성화됩니다. Java 애플리케이션에 있는 모든 메소드가 컴파일되는 것은 아니며 이는 효율성을 위해서입니다. JVM은 애플리케이션의각 메소드에

대한 호출 계수를 유지합니다. 메소드가 호출되고 해석될 때마다 해당 메소드에 대한 호출 계수가 증가됩니다. 계수가 컴파일 임계값에 도달하면 메소드가 컴파일되고 원래대로 실행됩니다.

호출 계수 메커니즘은 가장 자주 사용되는 메소드에 우선순위를 주어 애플리케이션의 실행 주기 동안 메소드 컴파일을 전개합니다. 일부 자주 사용되지 않는 메소드는 전혀 컴파일되지 않을 수도 있습니다. 결과적으로 Java 프로그램이 실패하면 문제가 JIT 또는 AOT 컴파일러에 있거나 JVM의 다른 부분에 있는 것입니다.

실패를 진단하는 첫 번째 단계는 문제가 *어디에* 있는지 판별하는 것입니다. 이렇게 하려면 먼저 단순히 해석된 모드(즉, JIT 및 AOT 컴파일러를 사용하지 않는 상태)에서 Java 프로그램을 실행해야 합니다.

프로시저

1. 명령행에서 **-Xjit**및 **-Xaot** 옵션(그리고 수반하는 매개변수)을 제거하십시오.
2. **-Xint** 명령행 옵션을 사용하여 JIT 및 AOT 컴파일러를 사용하지 않도록 설정하십시오. 성능을 고려하여 프로덕션 환경에서는 **-Xint** 옵션을 사용하지 마십시오.

다음에 수행할 작업

컴파일을 사용하지 않고 Java 프로그램을 실행하면 다음 경우 중 하나가 나타납니다.

- 실패가 지속됩니다. JIT 또는 AOT 컴파일러의 문제가 아닙니다. 일부 경우, 프로그램이 다른 방법으로 시작하지 못할 수 있지만 컴파일러 관련 문제는 아닙니다.
- 실패가 없어졌습니다. JIT 또는 AOT 컴파일러에 문제가 있을 가능성이 높습니다.

  공유 클래스를 사용하지 않는 경우라면 JIT 컴파일러에 문제가 있는 것입니다. 공유 클래스를 사용하는 경우라면 JIT 컴파일만 사용하여 애플리케이션을 실행해서 어떤 컴파일러에 문제가 있는지 판단할 수 있습니다. 애플리케이션을 **-Xint** 옵션이 아닌 **-Xnoaot** 옵션을 사용하여 실행하십시오. 이렇게 하면 다음 경우 중 하나가 나타납니다.
  - 실패가 지속됩니다. 이 경우 문제가 JIT 컴파일러에 있습니다. JIT 컴파일러에만 문제가 있는지 확인하려면 **-Xnoaot** 옵션 대신에 **-Xnojit** 옵션을 사용하십시오.
  - 실패가 없어졌습니다. 이 경우 문제가 AOT 컴파일러에 있습니다.

선택적으로 JIT 컴파일러 사용 안함:

Java 프로그램 실패가 JIT 컴파일러의 문제를 지정하는 경우 문제의 범위를 더욱 축소할 수 있습니다.

이 태스크 정보

기본적으로 JIT 컴파일러는 다양한 최적화 레벨에서 메소드를 최적화합니다. 다양한 최적화 선택사항이 호출 계수에 기반하여 다양한 메소드에 적용됩니다. 가장 자주 호출되

는 메소드가 상위 레벨로 최적화됩니다. JIT 컴파일러 매개변수를 변경하여 메소드를 최적화하는 레벨을 제어할 수 있습니다. 최적화 프로그램에 결함이 있는지 여부와 결함이 있으면 문제가 되는 최적화를 판별할 수 있습니다.

JIT 매개변수를 쉼표로 구분된 목록으로 **-Xjit** 옵션에 추가하여 지정할 수 있습니다. 구문은 **-Xjit**:<*param1*>,<*param2*>=<*value*>입니다. 예를 들면 다음과 같습니다.

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

HelloWorld 프로그램을 실행하고 JIT에서 상세 출력을 활성화하여 JIT에서 최적화 수행 없이 원시 코드를 생성하게 합니다.

컴파일러의 어느 부분에 문제가 있는지 판단하려면 다음 단계를 따르십시오.

프로시저

1. JIT 매개변수 **count=0**을 설정하여 컴파일 임계값을 영(0)으로 변경하십시오. 이 매개변수를 사용하면 각 Java 메소드가 실행되기 전에 컴파일됩니다. **count=0**은 문제점 진단 시에만 사용하십시오. 자주 사용하지 않는 메소드를 포함하여 추가로 여러 메소드가 컴파일됩니다. 추가 컴파일에서는 컴퓨팅 자원을 많이 사용하므로 애플리케이션의 속도가 저하됩니다. **count=0**을 사용하면 문제 영역에 도달했을 때 애플리케이션이 즉각 실패합니다. 일부 경우 **count=1**을 사용하여 확증을 위해 실패를 재현할 수 있습니다.
2. **disableInlining**을 JIT 컴파일러 매개변수에 추가하십시오. **disableInlining**은 더 크고 복잡한 코드 생성을 불가능하게 합니다. 더 이상 문제가 발생하지 않는다면 Java 서비스 팀에서 컴파일러 문제를 분석하고 정정하는 동안 **disableInlining**을 임시 해결 방법으로 사용하십시오.
3. **optLevel** 매개변수를 추가하여 최적화 레벨을 줄이고 실패가 더 이상 발생하지 않을 때까지 또는 『noOpt』 레벨에 도달할 때까지 프로그램을 다시 실행하십시오. JIT 컴파일러 문제일 경우 『scorching』으로 시작하고 목록에 대해 작업하십시오. 최적화 레벨은 다음과 같으며 내림차순입니다.
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold
   f. noOpt

다음에 수행할 작업

이러한 설정 중 하나를 사용하여 실패가 발생하지 않았으면, 이 방법은 사용할 수 있는 임시 해결 방법이 됩니다. 이 임시 해결 방법은 Java 서비스 킴이 컴파일러 문제를 분

석 및 수정하는 동안 임시로 사용됩니다. JIT 매개변수 목록에서 **disableInlining**을 제거해도 실패가 다시 발생하지 않으면 성능 향상을 위해 그렇게 하십시오. 임시 해결 방법의 성능을 향상시키려면 『실패 메소드 찾기』에 있는 지시사항을 따르십시오.

『noOpt』 최적화 레벨에서 실패가 계속해서 발생한다면 임시 해결 방법으로 JIT 컴파일러를 끄십시오.

실패 메소드 찾기:

가장 낮은 최적화 레벨로 JIT 또는 AOT 컴파일러가 메소드를 컴파일해서 실패를 트리거할 수 있다고 판단하는 경우 컴파일된 Java 프로그램의 어느 부분이 실패의 원인이 되는지 찾을 수 있습니다. 컴파일러를 특정 메소드, 클래스 또는 패키지에 대한 임시 해결 방법으로 한정하도록 지시하여 컴파일이 프로그램의 다른 부분은 정상적으로 컴파일하도록 할 수 있습니다. JIT 컴파일러 실패의 경우, 실패가 **-Xjit:optLevel=noOpt**에서 발생한다면, 컴파일러에 실패를 유발하는 메소드 모두를 컴파일하지 않도록 지시할 수 있습니다.

시작하기 전에

이 예와 유사한 오류 출력을 보고 실패한 메소드를 식별할 수 있습니다.

```
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;
```

중요한 행은 다음과 같습니다.

vmState=0x00000000

실패한 코드가 JVM Runtime 코드가 아님을 표시합니다.

Module= or Module_base_address=

코드가 JIT에서 컴파일되고 DLL 또는 라이브러리 외부에 있어 출력에 표시되지 않습니다(공백 또는 영(0)).

Compiled_method=

컴파일된 코드가 생성된 Java 메소드를 표시합니다.

이 태스크 정보

출력에서 실패한 메소드를 표시하지 않으면 다음 단계를 따라 추가 메소드를 식별하십시오.

프로시저

1. Java 프로그램을 **-Xjit**또는 **-Xaot** 옵션에 추가된 JIT 매개변수 **verbose** 및 **vlog=**<*filename*>을 적용하여 실행하십시오. 이 매개변수를 사용하면 컴파일러가 <*filename*>.<*date*>.<*time*>.<*pid*> 또는 *한계 파일* 로그 파일에 컴파일된 메소드를 나열합니다. 다음과 같이 일반적 한계 파일에는 컴파일된 메소드에 해당하는 행이 포함됩니다.

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

   더하기 부호로 시작하지 않는 행은 다음 단계에서 컴파일러가 무시하며 파일에서 이들을 제거할 수 있습니다. AOT에서 컴파일한 메소드는 +(AOT cold)로 시작합니다. 공유 클래스 캐시에서 로드된 AOT 코드에 대한 메소드는 +(AOT load)로 시작합니다.

2. JIT 또는 AOT 매개변수 **limitFile=**(<*filename*>,<*m*>,<*n*>)와 함께 프로그램을 다시 실행하십시오. 여기서 <*filename*>은 한계 파일에 대한 경로이며 <*m*> 및 <*n*>은 컴파일할 한계 파일에 있는 첫 번째 및 마지막 메소드를 표시하는 행 번호입니다. 컴파일러는 한계 파일의 <*m*>행부터 <*n*>행까지 나열된 메소드만 컴파일합니다. 한계 파일에 없는 메소드와 범위 밖의 행에 나열된 메소드는 컴파일되지 않으며 이러한 메소드에 대해 공유 데이터 캐시에 있는 AOT 코드는 로드되지 않습니다. 프로그램이 더 이상 실패하지 않으면 마지막 반복에서 제거한 하나 이상의 메소드가 실패의 원인입니다.

3. 옵션: AOT 문제를 진단하는 경우 동일한 옵션으로 프로그램을 두 번 실행하여 컴파일된 메소드를 공유 데이터 캐시에서 로드하십시오. **–Xaot:scount=0** 옵션을 추가하여 메소드가 처음 호출될 때 공유 데이터 캐시에 저장된 AOT 컴파일된 메소드가 사용되도록 하십시오. 일부 AOT 컴파일 실패는 AOT 컴파일된 코드가 공유 데이터 캐시에서 로드되는 경우에만 발생합니다. 이러한 문제를 진단하려면 **–Xaot:scount=0** 옵션을 사용하여 공유 데이터 캐시에 저장된 AOT 컴파일된 메소드가 메소드가 처음 호출될 때 사용되도록 하십시오. 문제를 쉽게 재현할 수 있습니다. **scount** 옵션을 0으로 설정하면 AOT 코드 로딩을 강제 실행하여 메소드 실행을 대기하는 애플리케이션 스레드를 일시정지시킵니다. 따라서 진단 목적으로만 사용하십시오. 일지정지 시간을 더 늘리려면 **–Xaot:scount=0** 옵션을 사용하십시오.

4. 필요한 횟수만큼 <*m*>과 <*n*>에 대해 다른 값을 적용하고 이 프로세스를 반복하여, 실패를 트리거하기 위해 컴파일해야 하는 최소 세트의 메소드를 찾으십시오. 매번 선택한 행의 수를 반으로 줄여서 실패한 메소드에 대해 2진 검색을 수행하십시오. 종종 파일을 한 행으로 줄일 수 있습니다.

다음에 수행할 작업

실패한 메소드의 위치를 찾으면 실패한 메소드에 한해 JIT 또는 AOT 컴파일러가 사용되지 않도록 하십시오. 예를 들어, 메소드 java/lang/Math.max(II)I가 **optLevel=hot**과 함께 JIT 컴파일되었을 때, 프로그램을 실패하게 한다면 다음을 적용하여 프로그램을 실행할 수 있습니다.

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

실패 메소드만 최적화 레벨『warm』으로 컴파일하고 다른 메소드들은 일반적으로 컴파일합니다.

메소드가『noOpt』에서 JIT 컴파일되었을 때 실패하는 경우 **exclude**={<*method*>} 매개변수를 사용하여 컴파일에서 이를 함께 제거할 수 있습니다.

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

메소드가 AOT 코드 컴파일되거나 공유 데이터 캐시에서 로드되었을 때 프로그램을 실패하게 하는 경우, AOT 컴파일 및 AOT 로딩에서 **exclude**={<*method*>} 매개변수를 사용하여 메소드를 제외하십시오.

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

AOT 메소드는『cold』최적화 레벨에서만 컴파일됩니다. AOT 컴파일 또는 AOT 로딩을 방지하는 것이 이러한 메소드를 접근하는 가장 좋은 방법입니다.

JIT 컴파일 장애 식별:

JIT 컴파일러 장애의 경우 JIT 컴파일러가 메소드 컴파일을 시도할 때 장애가 발생했는지 여부를 판별하기 위해 오류 출력을 분석하십시오.

JVM 크래쉬의 경우 JIT 라이브러리(libj9jit26.so)에서 장애가 발생하였다면 JIT 컴파일러가 메소드 컴파일을 시도하는 중에 실패했을 수 있습니다.

이 예와 유사한 오류 출력을 보고 실패한 메소드를 식별할 수 있습니다.

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

중요한 행은 다음과 같습니다.

vmState=0x00050000

JIT 컴파일러가 코드를 컴파일하고 있음을 표시합니다. vmState 코드 번호 목록의 경우 IBM SDK for Java V7사용자 안내서(http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html)의 Java 덤프 태그 테이블을 참조하십시오.

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

JIT 컴파일러 모듈인 libj9jit26.so에서 발생한 오류를 표시합니다.

Method_being_compiled=

Java 메소드가 컴파일되고 있음을 표시합니다.

출력에서 실패한 메소드를 표시하지 않으면 다음 추가 설정과 함께 **verbose** 옵션을 사용하십시오.

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

이러한 **verbose** 설정은 JIT가 메소드 컴파일을 시작 및 종료한 시기를 보고합니다. IT 특정 메소드에서 실패하는 경우(즉 컴파일을 시작했으나 종료 전에 크래쉬됨) **-Xjit** 또는 **-Xaot** 명령행 옵션에서 **exclude** 매개변수 컴파일에서 이를 제외시키십시오 (69 페이지의 『실패 메소드 찾기』 참조). 제외 메소드가 크래쉬를 방지한다면 서비스 팀에서 사용자 문제를 정정하는 동안 사용할 수 있는 임시 해결 방법으로 사용하십시오.

단기 실행 애플리케이션 성능

IBM JIT 컴파일러는 주로 서버에서 사용되는 장기 실행 애플리케이션을 위해 조정됩니다. 에서 **-Xquickstart** 명령행 옵션을 사용해서 적은 수의 메소드로 처리를 집중할 수 없는 애플리케이션의 경우에 단기 실행 어플리케이션의 성능을 향상하도록 사용할 수 있습니다.

-Xquickstart는 JIT 컴파일러가 기본적으로 낮은 최적화 레벨을 사용하도록 하여 적은 수의 메소드를 컴파일하게 합니다. 적은 수의 컴파일을 좀 더 빠르게 수행하면 애플리케이션 시작 시간을 단축할 수 있습니다. AOT 컴파일러가 활성(공유 클래스와 AOT 컴파일이 모두 사용 가능)이면 **-Xquickstart**는 컴파일을 위해 선택된 모든 메소드가 AOT 컴파일되도록 하여 후속 실행의 시작 시간을 단축합니다. **-Xquickstart**는 대량의 처리 자원을 사용하는 메소드가 포함된 장기 실행 애플리케이션에 사용될 경우 성능을 저하시킬 수 있습니다. **-Xquickstart** 구현은 향후 릴리스에서 변경될 수 있습니다.

JIT 임계값을 조정하여(시행 착오를 거쳐) 시작 시간을 향상시킬 수 있습니다. 자세한 정보는 67 페이지의 『선택적으로 JIT 컴파일러 사용 안함』의 내용을 참조하십시오.

대기 기간 동안 JVM 동작

JIT 샘플링 스레드를 끄기 위해 **-XsamplingExpirationTime** 옵션을 사용하여 대기 JVM에서 소비하는 CPU 주기를 줄일 수 있습니다.

JIT 샘플링 스레드는 공통적으로 사용되는 메소드를 감지하기 위해, 실행 중인 Java 애플리케이션을 프로파일합니다. 샘플링 스레드의 메모리와 프로세서 사용은 미미하며 프로파일링 빈도는 JVM이 대기 상태일 경우 자동으로 감축됩니다.

일부 환경에서 사용자는 대기 JVM이 CPU 주기를 소비하지 않기를 원할 수 있습니다. 그렇게 하려면 **-XsamplingExpirationTime**<*time*> 옵션을 지정하십시오. <*time*>을 샘플링 스레드를 실행하려는 시간(초)으로 설정하십시오. 이 옵션은 주의해서 사용해야 합니다. 샘플링 스레드를 *끄*게 되면 다시 활성화할 수 없습니다. 샘플링 스레드가 충분한 시간 동안 실행되어 중요한 최적화를 식별하도록 하십시오.

진단 콜렉터

진단 콜렉터는 문제점 이벤트에 사용할 Java 진단 파일을 수집합니다.

IBM 서비스에 필요한 파일을 수집하면 보고된 문제점을 해결하는 데 드는 시간을 줄일 수 있습니다. IBM SDK for Java V7 사용자 안내서에는 진단 콜렉터의 사용에 대한 자세한 정보가 포함되어 있습니다.

이 정보는 IBM SDK for Java 7 - 진단 콜렉터에서 찾을 수 있습니다.

가비지 콜렉터 진단 데이터

이 섹션은 가비지 콜렉션 문제점의 진단 방법을 설명합니다.

IBM SDK for Java V7 사용자 안내서에서는 가비지 콜렉터 문제점의 진단에 필요한 정보가 있으며 다음 내용을 다루고 있습니다.

- 상세 가비지 콜렉션 로깅
- **-Xtgc**를 사용하여 가비지 콜렉션 추적

이 정보는 IBM SDK for Java 7 - 가비지 콜렉터 진단 데이터에서 찾을 수 있습니다.

IBM WebSphere Real Time for Linux 메트로놈 가비지 콜렉터에 대한 보충 정보는 다음 섹션에 있습니다.

메트로놈 가비지 콜렉터 문제점 해결

명령행 옵션을 사용하여 메트로놈 가비지 콜렉션 빈도, 메모리 부족 예외 및 명시적 시스템 호출에서 메트로놈 동작을 제어할 수 있습니다.

verbose:gc 정보 사용:

-verbose:gc 옵션을 **-Xgc:verboseGCCycleTime=N** 옵션과 함께 사용하여 메트로놈 가비지 콜렉터 활동에 대한 정보를 콘솔에 작성할 수 있습니다. 표준 JVM의 **-verbose:gc** 출력에 있는 모든 XML 특성이 작성되거나 메트로놈 가비지 콜렉터 출력에 적용되는 것은 아닙니다.

힙에서 최소, 최대 및 중간 여유 공간을 보려면 **-verbose:gc** 옵션을 사용하십시오. 이 방식으로 활동 레벨과 힙 사용을 검사하고 필요하면 값을 조정할 수 있습니다. **-verbose:gc** 옵션은 메트로놈 통계를 콘솔에 작성합니다.

-Xgc:verboseGCCycleTime=N 옵션은 정보 검색 빈도를 제어하고, 요약을 덤프하는 시간(밀리초)를 판별합니다. N의 기본값은 1000밀리초입니다. 주기 시간이 해당 시간에 정확하게 요약이 덤프됨을 의미하지는 않지만 이 시간 기준을 충족하는 마지막 가비지 콜렉션 이벤트가 전달됩니다. 이 통계의 수집 및 표시로 메트로놈 가비지 콜렉터 일시정지 시간이 증가되고 N이 작아질수록 일시정지 시간이 길어질 수 있습니다.

퀀텀은 단일 메트로놈 가비지 콜렉터 활동 기간으로, 애플리케이션의 인터럽트 또는 일시정지 시간을 일으킵니다.

verbose:gc 출력 예제

입력:

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

가비지 콜렉션을 트리거하면 `trigger start` 이벤트와 많은 `heartbeat` 이벤트가 차례로 발생하고, 트리거를 충족하면 `trigger end` 이벤트가 발생합니다. 이 예제는 트리거된 가비지 콜렉션 주기를 verbose:gc 출력으로 보여줍니다.

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

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

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

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

<gc-op id="29" type="syncgc" timems="136.945" contextid="26" timestamp="2011-07-12T09:32:06.046">
  <syncgc-info reason="out of memory" exclusiveaccessTimeMs="0.006" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="21290752" bytesAfter="171963656" />
</gc-op>

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

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

다음과 같은 이벤트 유형이 발생할 수 있습니다.

<trigger-start ...>

가비지 콜렉션 주기의 시작이며 사용된 메모리 양이 트리거 임계값보다 커질 때입니다. 기본 임계값은 힙의 50%입니다. intervalms 속성은 이전 trigger end 이벤트(id-1)와 이 trigger start 이벤트 사이의 간격입니다.

<trigger-end ...>

가비지 콜렉션 주기를 사용된 메모리 *크기*를 트리거 임계값 아래로 낮추었습니다. 가비지 콜렉션 주기가 종료되었지만 사용된 메모리가 트리거 임계값 아래로 낮아지지 않으면 새 가비지 콜렉션 주기가 동일한 컨텍스트 ID로 시작됩니다. 각 trigger start 이벤트에 대해 동일한 컨텍스트 ID의 일치하는 trigger end 이벤트가 있습니다. intervalms 속성은 이전 trigger start 이벤트와 현재 trigger end 이벤트 사이의 간격입니다. 이 기간 동안 사용된 메모리가 트리거 임계값 아래로 낮아질 때까지 하나 이상의 가비지 콜렉션 주기가 완료됩니다.

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

관련 기간 동안 모든 가비지 콜렉션 퀀텀에 대한 정보(메모리 및 시간)를 수집하는 정기적 이벤트입니다. 일치하는 trigger start 및 trigger end 이벤트 쌍 사이(즉, 활성 가비지 콜렉션 주기가 진행 중일 때)에서만 하트비트 이벤트가 발생할 수 있습니다. intervalms 속성은 이전 하트비트 이벤트(id -1)와 이 하트비트 이벤트 사이의 간격입니다.

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

동기 (비결정적) 가비지 콜렉션 이벤트입니다. 76 페이지의 『동기 가비지 콜렉션』의 내용을 참조하십시오.

이 예제에서 XML 태그는 다음과 같은 의미를 가집니다.

<quanta ...>

일시정지 기간(밀리초)을 포함한 하트비트 간격 동안 퀀텀 일시정지 시간에 대한 요약입니다.

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

하트비트 간격 동안 여유 힙 공간 *크기*에 대한 요약으로, 각 가비지 콜렉션 퀀텀이 끝날 때 샘플링됩니다.

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

하트비트 간격 동안 로드 해제된 클래스 로더 및 클래스의 수입니다.

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

하트비트 간격 동안 해제된 Java 참조 오브젝트의 수 및 유형입니다.

참고:

- 두 하트비트 사이의 간격에 한 개의 가비지 콜렉션 퀀텀만 발생한 경우, 이 퀀텀이 끝날 때에만 여유 메모리가 샘플링됩니다. 따라서 하트비트 요약에 제공된 최소값, 최대값 및 중간값이 모두 동일합니다.
- 힙이 가바지 콜렉션 활동이 필요할 만큼 충분히 차지 않은 경우에는 두 하트비트 사이의 간격이 지정된 주기 시간보다 훨씬 길 수 있습니다. 예를 들어, 프로그램에서 몇 초마다 한 번만 가비지 콜렉션 활동이 필요한 경우 몇 초마다 한 번만 하트비트가 표시될 가능성이 높습니다.
- 가비지 콜렉션 활동이 일어날 만큼 충분히 차지 않은 힙에서는 가비지 콜렉션이 작동하지 않기 때문에 가비지 콜렉션 간격이 지정된 주기 시간보다 더 길어질 수 있습니다. 예를 들어, 프로그램에서 몇 초마다 한 번만 가비지 콜렉션 활동이 필요한 경우 몇 초마다 한 번만 하트비트가 표시될 가능성이 높습니다.

  동기 가비지 콜렉션이나 우선순위 변경 같은 이벤트가 발생하면 이벤트 세부사항과 하트비트 같은 보류 중인 이벤트가 출력으로 즉시 생성됩니다.
- 특정 기간 동안 최대 가비지 콜렉션 퀀텀이 너무 크면 **-Xgc:targetUtilization** 옵션을 사용하여 대상 이용률을 줄일 수 있습니다. 이 조치로 가비지 콜렉터에 작업 시간을 더 부여할 수 있습니다. 또는 **-Xmx** 옵션으로 힙 *크기*를 늘일 수도 있습니다. 이와 비슷하게, 애플리케이션이 현재 보고되는 것보다 긴 지연을 허용할 경우 대상 이용률을 높이거나 힙 *크기*를 줄일 수 있습니다.
- **-Xverbosegclog:<file>** 옵션을 사용하면 출력이 콘솔 대신 로그 파일로 경로 재지정됩니다. 예를 들어, **-Xverbosegclog:out**은 **-verbose:gc** 출력을 *out* 파일에 작성합니다.
- thread-priority에 나열된 우선순위는 Java 스레드 우선순위가 아닌 기본 운영 체제 스레드 우선순위입니다.

동기 가비지 콜렉션

동기 (비결정적) 가비지 콜렉션이 발생하면 **-verbose:gc** 로그에도 항목이 작성됩니다. 이 이벤트의 원인은 다음 3가지입니다.

- 코드에 명시적 System.gc() 호출.
- JVM에서 메모리가 부족하여 OutOfMemoryError 조건을 피하기 위해 동기 가비지 콜렉션을 수행합니다.
- JVM이 가비지 콜렉션이 지속되는 동안 종료됩니다. JVM은 콜렉션을 취소할 수 없으므로 콜렉션을 동기적으로 완료한 다음 존재합니다.

System.gc() 항목의 예제입니다.

```
<gc-op id="9" type="syncgc" timems="12.92" contextid="8" timestamp=
"2011-07-12T09:41:40.808">
  <syncgc-info reason="system GC" totalBytesRequested="260" exclusiveaccessTimeMs="0.009"
  threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="22085440" bytesAfter="136023450" />
  <classunload-info classloadersunloaded="54" classesunloaded="234" />
```

```
  <references type="soft" cleared="21" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="523" />
  <finalization enqueued="124" />
</gc-op>
```

JVM 종료의 결과로 나타나는 동기 가비지 콜렉션 항목의 예제입니다.

```
<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>
```

이 예제에서 XML 태그 및 속성은 다음과 같은 의미를 가집니다.

<gc-op id="9" type="syncgc" timems="6.439" ...
: 이 행은 이벤트 유형이 동기 가비지 콜렉션임을 나타냅니다. timems 속성은 동기 가비지 콜렉션 기간(밀리초)입니다.

<syncgc-info reason="..."/>
: 동기 가비지 콜렉션의 원인입니다.

<free-mem-delta.../>
: 동기 가비지 콜렉션 전후에 여유 Java 힙 메모리(바이트)입니다.

<finalization .../>
: 완료 대기 중인 오브젝트의 수입니다.

<classunload-info .../>
: 하트비트 간격 동안 로드 해제된 클래스 로더 및 클래스의 수입니다.

<references type="weak" cleared="53" .../>
: 하트비트 간격 동안 해제된 Java 참조 오브젝트의 수 및 유형입니다.

메모리 부족 조건이나 VM 종료로 인한 동기 가비지 콜렉션은 가비지 콜렉터가 활성인 경우에만 발생할 수 있습니다. 바로 이어질 필요는 없지만 이전에 trigger start 이벤트가 선행해야 합니다. 일부 하트비트 이벤트는 trigger start 이벤트와 synchgc 이벤트 사이에 발생합니다. System.gc()에 의해 발생한 동기 가비지 콜렉션은 언제든지 발생할 수 있습니다.

모든 GC 퀀텀 추적

GlobalGCStart 및 GlobalGCEnd 추적포인트를 사용하여 개별 GC 퀀텀을 추적할 수 있습니다. 이 추적포인트는 동기 가비지 콜렉션을 포함한 모든 메트로놈 가비지 콜렉터 활동이 시작하고 끝날 때 생성됩니다. 이 추적포인트의 출력은 다음과 유사합니다.

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

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

메모리 부족 항목

힙에서 여유 공간이 부족하면 OutOfMemoryError 예외가 발생하기 전에 **-verbose:gc** 로그에 항목이 기록됩니다. 이 출력의 예제는 다음과 같습니다.

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

기본적으로 Java 덤프가 OutOfMemoryError 예외의 결과로 생성됩니다. 이 덤프는 프로그램에 사용된 메모리 에 대한 정보를 포함하고 있습니다.

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

메모리 부족 조건에서 메트로놈 가비지 콜렉터 동작:

기본적으로 메트로놈 가비지 콜렉터는 JVM에서 메모리가 부족해지면 무제한 비결정적 가비지 콜렉션을 트리거합니다. 비결정적 동작을 방지하려면 **-Xgc:noSynchronousGCOnOOM** 옵션을 사용하여 JVM에서 메모리가 부족해질 때 OutOfMemoryError를 발생시키십시오.

단일 조작에서 모든 가능한 가비지를 수집할 때까지 기본 무제한 콜렉션이 실행됩니다. 대개는 일시정지 시간이 정상 메트로놈 증분 퀀텀보다 몇 밀리초 큽니다.

관련 정보:

동기 가비지 콜렉션을 분석하기 위해 -Xverbose:gc 사용

명시적 System.gc() 호출에서 메트로놈 가비지 콜렉터 동작:

가비지 콜렉션 주기가 진행 중이면 System.gc()를 호출할 때 메트로놈 가비지 콜렉터가 동기적으로 주기를 완료합니다. 진행 중인 가비지 콜렉션 주기가 없으면 System.gc()를 호출할 때 전체 동기 주기가 수행됩니다. 제어 상태로 힙을 정리하려면 System.gc()를 사용하십시오. 이 조작은 리턴하기 전에 전체 가비지 콜렉션을 완료하기 때문에 비결정적입니다.

이 비결정적 지연이 용인되지 않는 일부 애플리케이션은 System.gc() 호출을 포함한 벤더 소프트웨어를 호출합니다. 모든 System.gc() 호출을 사용 불가능하게 하려면 **-Xdisableexplicitgc** 옵션을 사용하십시오.

System.gc() 호출에 대한 상세 가비지 콜렉션 출력은 『시스템 가비지 콜렉션』 이유를 포함하고 있으며 지속 기간이 깁니다.

```
<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>
```

공유 클래스 진단 데이터

발생 가능한 문제점을 진단하는 방법을 알면 공유 클래스 모드를 사용하는 데 도움이 됩니다.

IBM SDK for Java V7 사용자 안내서에는 공유 클래스 문제점 진단에 필요한 정보가 있으며 다음 내용을 다루고 있습니다.

- 공유 클래스 배치
- 런타임 바이트 코드 수정 처리
- 동적 업데이트 이해
- Java Helper API 사용
- 공유 클래스 진단 출력 이해
- 공유 클래스로 문제점 디버깅

이 정보는 IBM SDK for Java 7 - 공유 클래스 진단 데이터에서 찾을 수 있습니다.

JVMTI 사용

JVMTI는 JVM과 원시 에이전트 간의 통신이 가능한 양방향 인터페이스입니다. 이는 JVMDI와 JVMPI 인터페이스를 대체합니다.

JVMT를 사용하여 써드파티에서 JVM에 대한 모니터링, 프로파일링 및 디버깅 도구를 개발할 수 있습니다. 인터페이스에는 에이전트가 JVM에 필요한 정보의 종류를 통지하는 메커니즘이 포함되어 있습니다. 인터페이스는 또한 관련 통지를 수신하는 수단도 제공합니다. 몇몇 에이전트를 언제든 JVM에 부착할 수 있습니다.

IBM SDK for Java V7 사용자 안내서에는 JVMTI에 대한 IBM 확장기능에 관한 API 참조 섹션을 포함한 자세한 JVMTI 사용 정보가 포함되어 있습니다.

이 정보는 IBM SDK for Java 7 - JVMTI 사용에서 찾을 수 있습니다.

DTFJ(Diagnostic Tool Framework for Java) 사용

DTFJ(Diagnostic Tool Framework for Java)는 IBM의 Java API(Application Programming Interface)로 Java 진단 도구 빌드를 지원하는 데 사용됩니다. DTFJ는 시스템 덤프 또는 Java 덤프의 데이터로 작업합니다.

IBM SDK for Java V7 사용자 안내서에는 자세한 DTFJ 정보가 포함되어 있습니다. 다음 링크를 사용하십시오. Java용 진단 도구 프레임워크 사용

제 10 장 참조

이 주제에서는 WebSphere Real Time for Linux와 함께 사용할 수 있는 옵션 및 클래스 라이브러리를 보여줍니다.

명령행 옵션

Java를 시작할 때 명령행에서 옵션을 지정할 수 있습니다. 기본 옵션은 가장 적합한 일반 용도에 맞게 선택되었습니다.

Java 옵션 및 시스템 특성 지정

Java 옵션 및 시스템 특성을 지정하는 방법은 3가지가 있습니다.

이 태스크 정보

다음 방법으로 Java 옵션 및 시스템 특성을 지정할 수 있습니다. 우선 순위에 따라 다음 방법을 사용합니다.

1. 명령행에서 옵션 또는 특성 지정. 예를 들면 다음과 같습니다.

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

2. 옵션이 있는 파일을 작성하고 **-Xoptionsfile=**<*filename*> 옵션을 사용하여 명령행에서이 파일을 지정합니다.

   옵션 파일에서, 새 행에 각 옵션을 지정하십시오. 백슬래시('\') 문자를 연속 문자로 사용하는 경우 하나의 옵션을 여러 행에 걸쳐 지정할 수 있습니다. 명령행을 정의하려면 '#' 문자를 사용하십시오. 옵션 파일에서 **-classpath**를 지정할 수 없습니다. 옵션 파일의 예:

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

3. 옵션을 포함한 **IBM_JAVA_OPTIONS**라는 환경 변수 작성. 예를 들면 다음과 같습니다.

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

명령행에서 지정하는 마지막 옵션은 첫 번째 옵션보다 우선순위가 높습니다. 예를 들어, **-Xint -Xjit myClass** 옵션을 지정하는 경우 **-Xjit** 옵션이 **-Xint**보다 우선하여 사용됩니다.

시스템 특성

애플리케이션에서 시스템 특성을 사용할 수 있으며, 런타임 환경에 대한 정보를 제공합니다.

com.ibm.jvm.realtime

이 특성은 Java 애플리케이션이 WebSphere Real Time for Linux 환경에서 실행되는지 여부를 판별할 수 있도록 지원합니다.

애플리케이션이 IBM WebSphere– Real Time for RT Linux 런타임 환경에서 실행되고 **-Xrealtime** 옵션으로 시작한 경우 **com.ibm.jvm.realtime** 특성의 값은 『hard』입니다.

애플리케이션이 IBM WebSphere– Real Time for RT Linux 런타임 환경에서 실행되지만 **-Xrealtime** 옵션으로 시작하지 않은 경우 **com.ibm.jvm.realtime** 특성이 설정되지 않습니다.

애플리케이션이 IBM WebSphere Real Time 런타임 환경에서 실행되는 경우에 **com.ibm.jvm.realtime** 특성의 값은 『soft』입니다.

표준 옵션

표준 옵션에 대한 정의입니다.

-agentlib:<*libname*>**[=**<*options*>**]**

원시 에이전트 라이브러리 <*libname*>을 로드합니다(예: **-agentlib:hprof**). 자세한 정보를 보려면 명령행에 **-agentlib:jdwp=help** 및 **-agentlib:hprof=help**를 지정하십시오.

-agentpath:*libname***[=**<*options*>**]**

전체 경로 이름별로 원시 에이전트 라이브러리를 로드합니다.

-assert

assert 관련 옵션에 대한 도움말을 인쇄합니다.

-cp or -classpath <*:으로 구분된 디렉토리 및 .zip 또는 .jar 파일*>

애플리케이션 클래스 및 자원의 검색 경로를 설정합니다. **-classpath** 및 **-cp**가 사용되지 않고 **CLASSPATH**가 설정되지 않은 경우, 기본적으로 사용자 클래스 경로는 현재 디렉토리(.)입니다..

-D<property_name>=<*value*>

시스템 특성을 설정합니다.

-help 또는 **-?**

사용법 메시지를 인쇄합니다.

-javaagent:<*jarpath*>**[=**<*options*>**]**

Java 프로그래밍 언어 에이전트를 로드합니다. 자세한 정보는 java.lang.instrument API 문서를 참조하십시오.

-jre-restrict-search

버전 검색에 사용자 개인용 JRE를 포함합니다.

-no-jre-restrict-search

버전 검색에서 사용자 개인용 JRE를 제외합니다.

-showversion

제품 버전을 인쇄하고 계속 진행합니다.

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

상세 출력을 사용합니다.

-verbose:class

로드한 각 클래스의 stderr에 항목을 기록합니다.

-verbose:gc

73 페이지의 『verbose:gc 정보 사용』의 내용을 참조하십시오.

-verbose:dynload

다음을 포함하여 JVM에서 각 클래스를 로드할 때 상세한 정보를 제공합니다.

- 클래스 이름과 패키지
- `.jar` 파일에 있던 클래스 파일의 경우, `.jar`의 이름과 디렉토리 경로
- 클래스를 로드할 때 소요된 시간과 클래스 사이즈의 세부사항

데이터가 stderr에 작성됩니다. 출력 예는 다음과 같습니다.

```
<Loaded java/lang/String from /myjdk/sdk/jre/lib/i386/softrealtime/jclSC16
<Class size 17258; ROM size 21080; debug size 0>
<Read time 27368 usec; Load time 782 usec; Translate time 927 usec>
```

참고: 공유 클래스 캐시에서 로드되는 클래스는 **`-verbose:dynload`** 출력에 나타나지 않습니다. 그러한 클래스에 대한 정보는 **`-verbose:class`** 옵션을 사용하십시오.

-verbose:sizes

JVM에서 스택 및 힙에 사용된 메모리 양을 설명하는 정보를 stderr에 기록합니다.

-verbose:stack

Java 및 C 스택 사용량을 설명하는 정보를 stderr에 기록합니다.

-verbose:jni

애플리케이션 및 JVM에서 호출된 JNI 서비스를 설명하는 정보를 stderr에 기록합니다.

-version

실시간 이외 모드의 버전 정보를 인쇄합니다.

-version:<*value*>

지정된 버전을 실행해야 합니다.

-X 비표준 옵션에 대한 도움말을 인쇄합니다.

비표준 옵션

접두부에 **-X**가 있는 옵션은 비표준 옵션이며 별도의 통지없이 변경될 수 있습니다.

IBM SDK for Java V7 사용자 안내서에는 비표준 옵션에 대한 자세한 정보가 포함되어 있습니다. 이 정보는 IBM SDK for Java 7 - 명령행 옵션에서 찾을 수 있습니다.

다음 섹션에 IBM WebSphere Real Time for Linux에 대한 보충 정보가 있습니다.

메트로놈 가비지 콜렉터 옵션

메트로놈 가비지 콜렉터 옵션의 정의입니다.

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM

힙 메모리가 부족할 때 가비지 콜렉션이 발생하는 한 경우입니다. 힙에 여유 공간이 없을 때 **-Xgc:synchronousGCOnOOM**을 사용하면 가비지 콜렉션에서 사용되지 않는 오브젝트를 제거하는 동안 애플리케이션이 중지됩니다. 여유 공간이 다시 부족해지면 가비지 콜렉션이 완료될 때까지 추가 시간을 허용하도록 대상 이용률을 낮추는 것을 고려하십시오. **-Xgc:nosynchronousGCOnOOM** 설정은 힙 메모리가 꽉 차면 애플리케이션이 중지되고 메모리 부족 메시지가 발행됨을 의미합니다. 기본값은 **-Xgc:synchronousGCOnOOM**입니다.

-Xnoclassgc

클래스 가비지 콜렉션을 사용 불가능하게 합니다. 이 옵션은 JVM에서 더 이상 사용되지 않는 Java 클래스와 연관된 스토리지의 가비지 콜렉션이 사용되지 않도록 합니다. 기본 동작은 **-Xnoclassgc**입니다.

-Xgc:targetPauseTime=*N*

가비지 콜렉션 일시정지 시간을 설정합니다. 여기서 *N*은 시간(밀리초)입니다. 이 옵션을 지정하면 GC가 지정된 값을 초과하지 않는 일시정지와 함께 작동합니다. 이 옵션이 지정되지 않으면 기본 일시정지 시간이 3밀리초로 설정됩니다. 예를 들어, **-Xgc:targetPauseTime=20**으로 실행하면 GC가 GC 조작 중에 20밀리초보다 길지 않게 일시정지합니다.

-Xgc:targetUtilization=*N*

애플리케이션 이용률을 N%로 설정합니다. 가비지 콜렉터가 각 간격의 최대 (100-N)%를 사용하려고 합니다. 합리적인 값은 50-80%의 범위에 있습니다. 할당률이 낮은 애플리케이션은 90% 실행될 수 있습니다. 기본값은 70%입니다.

이 예제는 힙 메모리의 최대 *크기*가 30MB임을 보여줍니다. 애플리케이션의 대상 이용률이 75%이기 때문에 가비지 콜렉터가 각 간격의 25%를 사용하려고 합니다.

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

-Xgc:threads=*N*

실행할 GC 스레드의 수를 지정합니다. 기본값은 프로세스에 사용 가능한 프로세서 코어의 수입니다. 지정할 수 있는 최대값은 운영 체제에서 사용 가능한 프로세서의 수입니다.

-Xgc:verboseGCCycleTime=N

N은 요약 정보를 덤프하는 시간(밀리초)입니다.

참고: 주기 시간이 해당 시간에 정확하게 요약 정보가 덤프됨을 의미하지는 않지만 이 시간 기준을 충족하는 마지막 가비지 콜렉션 이벤트가 전달됩니다.

-Xmx<*size***>**

Java 힙 *크기*를 지정합니다. 다른 가비지 콜렉션 전략과 달리, 실시간 메트로놈 GC는 힙 확장을 지원하지 않습니다. 초기 또는 최대 힙 *크기* 옵션이 없습니다. 최대 힙 *크기*만 지정할 수 있습니다.

JVM 기본 설정

JVM이 실행되는 환경에 대한 변경사항이 없으면 기본 설정이 실시간 JVM에 적용됩니다. 참조용으로 공통 설정이 표시됩니다.

JVM 시작 시 환경 변수 또는 명령행 매개변수를 사용하여 기본 설정을 변경할 수 있습니다. 다음 표는 몇 가지 공통 JVM 설정을 보여줍니다. 마지막 열은 다음 키가 적용될 경우 동작을 어떻게 변경되는지 알려줍니다.

- **e** - 환경 변수로만 제어하는 설정
- **c** - 명령행 매개변수로만 제어하는 설정
- **ec** - 환경 변수와 명령행 매개변수 둘 다로 제어하는 설정으로, 명령행 매개변수가 우선 적용됩니다.

이 정보는 빠른 참조로 제공되므로, 모든 내용을 포함하고 있지는 않습니다.

JVM 설정	기본	설정에 영향을 주는 주체
Java 덤프	사용	ec
메모리 부족 시 Java 덤프	사용	ec
힙 덤프	사용 안함	ec
메모리 부족 시 힙 덤프	사용	ec
시스템 덤프	사용	ec
덤프 파일이 생성되는 위치	Current® 디렉토리	ec

JVM 설정	기본	설정에 영향을 주는 주체
상세 출력	사용 안함	c
부트 클래스 경로 검색	사용 안함	c
JNI 검사	사용 안함	c
원격 디버깅	사용 안함	c
엄격한 적합성 검사	사용 안함	c
빠른 시작	사용 안함	c
원격 디버그 정보 서버	사용 안함	c
신호 감소	사용 안함	c
신호 처리기 체인	사용	c
클래스 경로	설정 없음	ec
클래스 데이터 공유	사용 안함	c
내게 필요한 옵션 지원	사용	e
JIT 컴파일러	사용	ec
AOT 컴파일러(공유 클래스가 함께 사용 가능해야 JVM에서 AOT를 사용합니다.)	사용	c
JIT 디버그 옵션	사용 안함	c
알고리즘 볼드체의 Java2D 최대 사이즈 글꼴	14 포인트	e
Java2D가 확장 가능한 글꼴에서 렌더링된 비트맵 사용	사용	e
Java2D 프리타입 글꼴 래스터라이징	사용	e
Java2D가 AWT 글꼴 사용	사용 안함	e
기본 로케일	없음	e
플러그인 시작 전 대기 시간	0	e
임시 디렉토리	/tmp	e
플러그인 방향 재지정	없음	e
IM 교환	사용 안함	e
IM 수정자	사용 안함	e
스레드 모델	N/A	e
Java 스레드 32비트에 대한 초기 스택 *크기*. **-Xiss<size>** 사용	2KB	c
Java 스레드 32비트에 대한 최대 스택 *크기*. **-Xss<size>** 사용	256KB	c
OS 스레드 32비트에 대한 스택 *크기*. **-Xmso<size>** 사용	256KB	c
초기 힙 *크기*. **-Xms<size>** 사용	64MB	c
최대 Java 힙 *크기*. **-Xmx<size>** 사용	최소 16MB와 최대 512MB의 사용 가능 메모리 절반	c
애플리케이션의 대상 시간 간격 이용률. 가비지 콜렉터가 남은 항목 사용 시도. **-Xgc:targetUtilization=<percentage>** 사용	70%	c
실행할 가비지 콜렉터 스레드의 수. **-Xgc:threads=<value>** 사용	프로세스에 사용 가능한 프로세서 코어 수.	c

JVM 설정	기본	설정에 영향을 주는 주체
-Xrealtime 모드에서 범위 메모리에 할당할 수 있는 최대 메모리 *크기*. **-Xgc:scopedMemoryMaximumSize=<size>** 사용	8MB	c
-Xrealtime 모드에서 영구 메모리 영역의 *크기* 설정. **-Xgc:immortalMemorySize=<size>** 사용	16MB	c

참고: 『사용 가능 메모리』는 실제(물리적) 메모리이거나 **RLIMIT_AS** 값이며 가장 작은 값입니다.

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다.

IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함)간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이센스 사용자는 다음 주소로 문의하십시오.

- JIMMAIL@uk.ibm.com [한국 아이.비.엠 주식회사 고객만족센터]

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

본 문서에 기술된 라이센스가 있는 프로그램 및 사용 가능한 모든 라이센스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 단계의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한 일부 성능은 추정을 통해 추측되었을 수도 있으므로 실제 결과는 다를 수 있습니다. 이 책의 사용자는 해당 데이터를 본인의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

개인정보 보호정책 고려사항

SaaS(Software as a Service) 솔루션을 포함한 IBM 소프트웨어 제품(이하 "소프트웨어 오퍼링")은 제품 사용 정보를 수집하거나 일반 사용자의 사용 경험을 개선하거나 일반 사용자와의 상호 작용을 조정하거나 그 외의 용도로 쿠키나 기타 다른 기술을 사용할 수 있습니다. 대부분의 경우 소프트웨어 오퍼링에서 개인적으로 식별 가능한 정보는 수집하지 않습니다. 일부 소프트웨어 오퍼링은 사용자가 개인적으로 식별 가능한 정보

를 수집할 수 있도록 합니다. 이 소프트웨어 오퍼링이 쿠키를 사용하여 개인적으로 식별 가능한 정보를 수집하는 경우 이 오퍼링의 쿠키 사용에 대한 특정 정보가 아래에 나와 있습니다.

본 소프트웨어 오퍼링은 쿠키 또는 기타 기술을 사용하여 개인 식별 정보를 수집하지 않습니다.

이 소프트웨어 오퍼링을 위해 배치된 구성에서 고객이 쿠키 및 다른 기술을 통해서 일반 사용자의 개인 식별 정보를 수집할 수 있는 기능을 제공하는 경우, 해당 데이터 수집(고지 및 동의 요구사항 포함)에 적용되는 모든 법률에 대해서 자체적으로 법률 자문을 구하는 것이 좋습니다.

해당 용도의 쿠키를 포함하여 다양한 기술의 사용에 대한 자세한 정보는 IBM 개인정보 보호정책(http://www.ibm.com/privacy/kr/ko/), IBM 온라인 개인정보 보호정책(http://www.ibm.com/privacy/details/kr/ko/, 특히 "쿠키, 웹 비콘 및 기타 기술" 섹션) 및 "IBM 소프트웨어 제품 및 SaaS(Software-as-a Service) 개인정보 보호정책"(http://www.ibm.com/software/info/product-privacy)을 참조하십시오.

상표

IBM, IBM 로고 및 ibm.com은 미국 또는 기타 국가에서 사용되는 International Business Machines Corporation의 상표 또는 등록상표입니다. 이와 함께 기타 IBM 상표가 기재된 용어가 상표 기호(® 또는 ™)와 함께 이 정보에 처음 표시된 경우, 이와 같은 기호는 이 정보를 발행할 때 미국에서 IBM이 소유한 등록상표 또는 일반 법적 상표입니다. 또한 이러한 상표는 기타 국가에서 등록상표 또는 일반 법적 상표입니다. 현재 IBM 상표 목록은 웹 "저작권 및 상표 정보"(http://www.ibm.com/legal/copytrade.shtml)에 있습니다.

Adobe, Adobe 로고, PostScript 및 PostScript 로고는 미국 또는 기타 국가에서 사용되는 Adobe Systems Incorporated의 상표 또는 등록상표입니다.

Intel 및 Itanium은 미국 및 기타 국가에서 사용되는 Intel Corporation 또는 그 자회사의 상표입니다.

Linux는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 상표입니다.

Java 및 모든 Java 기반 상표와 로고는 Oracle 및/또는 그 계열사의 상표 또는 등록상표입니다.

기타 회사, 제품 또는 서비스 이름은 타사의 상표 또는 서비스표입니다.

색인

Printed in Korea