IBM WebSphere Real Time for RT Linux
Versión 3

Guía del usuario

IBM WebSphere Real Time for RT Linux
Versión 3

Guía del usuario

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información del apartado “Avisos” en la página 165.

Quinta edición (febrero de 2014)

Esta edición de la guía del usuario se aplica a IBM WebSphere Real Time for RT Linux, versión 3, y a todos los releases y modificaciones posteriores, hasta que se indique otra cosa en ediciones nuevas.

Contenido

Figuras

Tablas

Prefacio

Esta guía del usuario ofrece información general sobre IBM® WebSphere Real Time for RT Linux.

Capítulo 1. Introducción

Aquí se proporciona información sobre IBM WebSphere Real Time for RT Linux.

Las nuevas modificaciones realizadas en esta guía del usuario se indican mediante barras verticales a la izquierda de los cambios.

Otra información más reciente sobre IBM WebSphere Real Time for RT Linux que no esté disponible en la guía de usuario puede consultarse aquí: http://www.ibm.com/support/docview.wss?uid=swg21501145

Visión general de WebSphere Real Time for RT Linux

Funciones en tiempo real de paquetes de WebSphere Real Time for RT Linux con IBM J9 virtual machine (JVM).

WebSphere Real Time for RT Linux es un entorno de ejecución Java™ con un kit de desarrollo de software que amplía el SDK de IBM para Java con prestaciones en tiempo real. Las aplicaciones que dependen de tiempos de respuesta precisos pueden aprovechar las funciones de tiempo real que se proporcionan con WebSphere Real Time for RT Linux en la tecnología Java estándar.

Características

Las aplicaciones de tiempo real necesitan tiempo de ejecución coherente en lugar de una velocidad absoluta.

Cuando la JVM se ejecuta en modalidad de tiempo real,habrá áreas de memoria adicionales disponibles junto con el almacenamiento dinámico con recogida de basura. Los programas pueden solicitar o especificar un número indefinido de áreas de memoria de ámbito reutilizables o áreas de memoria inmortal no reutilizables, que no pasan por la recogida de basura. Esta capacidad proporciona a la aplicación más control sobre el uso de la memoria. También utiliza Recopilador de basura Metronome para conseguir colecciones basadas en el tiempo. Cuando la JVM se ejecuta en una modalidad de rendimiento tradicional, se pueden utilizar varios recopiladores de basura basados en el trabajo que optimizan el rendimiento, pero que pueden pasar por mayores retardos individuales que Recopilador de basura Metronome.

Las principales preocupaciones a la hora de desplegar aplicaciones de tiempo real con las JVM tradicionales son las siguientes:
- Retardos imprevisible (potencialmente largos) por la actividad de recogida de basura.
- Retardos en el tiempo de ejecución del método mientras se producen la compilación JIT (Just-In-Time) y la recompilación, con variabilidad en el tiempo de ejecución.
- Planificación arbitraria del sistema operativo.

WebSphere Real Time for RT Linux elimina estos obstáculos proporcionando:

- Recopilador de basura Metronome, un recopilador de basura incremental y determinante, con tiempos de pausa muy breves.
- Compilación AOT (Ahead-Of-Time).
- Planificación FIFO basada en prioridad.

Además, WebSphere Real Time proporciona al programador de tiempo real los recursos de RTSJ; consulte el apartado "Soporte para RTSJ" en la página 12.

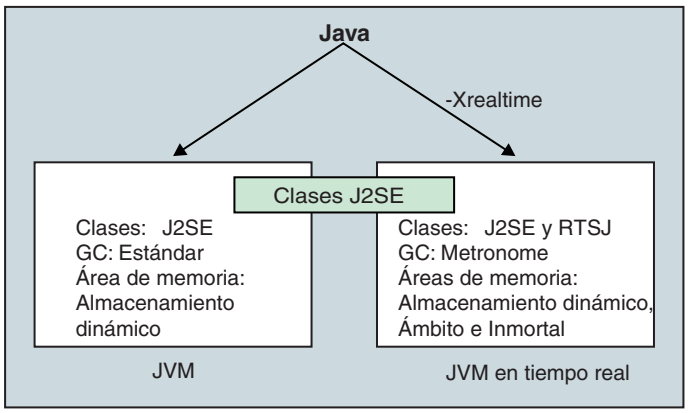

Figura 1. Visión general de WebSphere Real Time for RT Linux

Las prestaciones de tiempo real se habilitan utilizando la opción -Xrealtime al ejecutar la JVM o cualquiera de las herramientas proporcionadas. De forma predeterminada, la JVM y las herramientas proporcionadas se ejecutan sin las prestaciones de tiempo real habilitadas. Figura 1 muestra las relaciones de las dos JVM proporcionadas con WebSphere Real Time for RT Linux.

Los siguientes mandatos de Java reconocen la opción -Xrealtime:

Tabla 1. Mandatos Java utilizados en modalidad de tiempo real

Mandato	**Función**
java	Se ejecuta en la modalidad estándar de forma predeterminada, pero también se ejecuta en modalidad de tiempo real cuando se especifica la opción -Xrealtime. En modalidad de tiempo real, el programador acceder a las clases desde el paquete javax.realtime. Puede utilizar archivos jar precompilados y la tecnología de recogida de basura determinista de Metronome.
javac, javah, javap	Se ejecuta en modalidad estándar de forma predeterminada; pero, cuando se especifica la opción -Xrealtime, incluye las clases javax.realtime.* en la vía de acceso de clases.
admincache	Se puede ejecutar con o sin -Xrealtime, pero llenar una memoria caché con la herramienta **admincache** solo es posible en la modalidad de tiempo real. En la modalidad común, solo están disponibles los programas de utilidad de la memoria caché (como listAllCaches o printStats) Al igual que **jdmpview**, **admincache** se debe ejecutar con -Xrealtime para acceder a las memorias caché de la JVM de tiempo real, y se debe ejecutar sin -Xrealtime para acceder a las memorias caché de la JVM común.
jextract	jextract se ejecuta en modalidad estándar de forma predeterminada, pero se deben ejecutar con la opción -Xrealtime cuando procesa los volcados del sistema generados por la JVM en modalidad de tiempo real

Novedades

En este tema se presentan los cambios para IBM WebSphere Real Time for RT Linux.

WebSphere Real Time for RT Linux V3

WebSphere Real Time for RT Linux V3 es una ampliación al SDK para Java V7 de IBM, construcción sobre las características y funciones disponibles con este release para incluir prestaciones en tiempo real. Las versiones anteriores de WebSphere Real Time for RT Linux estaban basadas en releases anteriores del SDK para Java de IBM.

Para obtener más información sobre las novedades de SDK para Java 7 de IBM, consulte: Novedades en el Information Center del SDK para Java 7 de IBM.

Las nuevas modificaciones realizadas en esta guía del usuario se indican mediante barras verticales a la izquierda de los cambios.

Soporte para el sistema operativo Real Time Linux

A partir de ahora se incluye soporte para Red Hat Enterprise MRG 2.2, para Red Hat Enterprise Linux 6 Update 3. Para obtener más información sobre el hardware y el software admitidos, incluidas las actualizaciones o parches necesarios, consulte "Requisitos de hardware y software" en la página 27.

Uso de páginas grandes

En la renovación de servicio 4, el SDK de IBM para Java V7 incorpora el uso de páginas grandes de forma predeterminada para plataformas Linux x86 y AMD64/EM64T. Aunque WebSphere Real Time for RT Linux es una ampliación del SDK para Java V7 de IBM, las páginas grandes no están habilitadas de forma predeterminada debido a un problema conocido.

Resolución de símbolos

De forma predeterminada, la JVM resuelve inmediatamente la resolución de símbolos para cada función de una biblioteca nativa de usuario. Utilice la opción **-XX:+LazySymbolResolution** para forzar a la JVM a retrasar la resolución de símbolos de todas las funciones de una biblioteca nativa de usuario hasta que se llame a la función. Para obtener más información, consulte "Opciones no estándar" en la página 159.

Soporte para el sistema operativo Real Time Linux

A partir de ahora se incluye soporte para SUSE Linux Enterprise Real Time versión 11. Para obtener más información sobre el hardware y el software admitidos, incluidas las actualizaciones o parches necesarios, consulte "Requisitos de hardware y software" en la página 27.

jxeinajar

WebSphere Real Time for RT Linux V3 ya no tiene soporte para el uso de jxeinajar. Por motivos de referencia, la información anterior sobre jxeinajar y, concretamente, cómo migrar a admincache, la encontrará en la Documentación de WebSphere Real Time for RT Linux V2.

Ventajas

Las ventajas del entorno de tiempo real son que las aplicaciones Java se ejecutan con un mayor grado de predicción que con la JVM estándar y ofrecen un comportamiento de temporización coherente para su aplicación Java. Las actividades de fondo, como la compilación y la recogida de basura, se producen en momentos determinados y eliminan cualquier pico inesperado de actividad de fondo durante la ejecución de la aplicación.

Estas ventajas se obtienen al ampliar la JVM con las funciones siguientes:

- Tecnología de recogida de basura en tiempo real de Metronome
- Compilación AOT (Ahead-of-time)
- Soporte para Especificación de tiempo real para Java (RTSJ)

Todas las aplicaciones Java se pueden ejecutar en un entorno de tiempo real sin modificaciones, beneficiándose del recopilador de basura Metronome y de su recogida de basura determinante en intervalos regulares. Para conseguir el máximo provecho de WebSphere Real Time for RT Linux, puede escribir aplicaciones específicamente para el entorno de tiempo real utilizando hebras de tiempo real y hebras en tiempo real de almacenamiento no dinámico . El enfoque que tome dependerá de la especificación de temporización de su aplicación.

Muchas aplicaciones Java de tiempo real pueden utilizar los pequeños tiempos de pausa de Recopilador de basura Metronome y AOT para conseguir sus objetivos, conservando las ventajas de la portabilidad Java. Las aplicaciones con requisitos más estrictos deben utilizar los recursos RTSJ de hebras de tiempo real y hebras en tiempo real de almacenamiento no dinámico , con la memoria inmortal y de ámbito. Este enfoque limita su aplicación para que se ejecute solo en un entorno de tiempo real, perdiendo la ventaja de la portabilidad a JSE Java. También tiene que desarrollar un modelo de programación más complejo.

Accesibilidad

Las características de accesibilidad permiten que los usuarios con discapacidades, por ejemplo, con limitaciones en la movilidad o la vista, puedan utilizar productos de tecnologías de la información sin problemas.

IBM se esfuerza en proporcionar productos con acceso apto para todos los usuarios, independientemente de la edad o capacidad.

Por ejemplo, puede utilizar WebSphere Real Time for RT Linux sin ratón, utilizando solo el teclado.

Si desea leer acerca de estas cuestiones que afectan a la accesibilidad de IBM SDK para Java V7 subyacente, consulte IBM Information Center. No hay problemas de accesibilidad que afecten a una determinada prestación o función en WebSphere Real Time for RT Linux.

Navegación del teclado

Este producto utiliza las teclas de navegación estándar de Microsoft Windows.

Los usuarios que necesiten el desplazamiento mediante teclas pueden consultar la descripción de las pulsaciones más útiles para las aplicaciones Swing en Swing Key Bindings.

IBM y accesibilidad

Consulte el apartado IBM Human Ability and Accessibility Center para obtener más información sobre el compromiso que IBM tiene con la accesibilidad.

Capítulo 2. Información acerca de IBM WebSphere Real Time for RT Linux

En esta sección se presentan componentes clave de IBM WebSphere Real Time for RT Linux.

Introducción al recopilador de basura Metronome

El recopilador de basura Metronome sustituye al recopilador de basura estándar en WebSphere Real Time for RT Linux.

La diferencia principal entre la recogida de basura Metronome y la recogida de basura estándar es que la recogida de basura Metronome se produce en pequeños pasos que se pueden interrumpir, mientras que la recogida de basura estándar detiene la aplicación mientras marca y recoge la basura.

Por ejemplo:

```
java -Xrealtime -Xgc:targetUtilization=80 su_aplicación
```

El ejemplo especifica que su aplicación se ejecuta durante el 80% de cada 60 ms. El 20% del tiempo se utiliza para la recogida de basura, si hay basura que recoger. El recopilador de basura Metronome garantiza la utilización de los niveles siempre que se le hayan otorgado recursos suficientes. La recogida de basura empieza cuando el espacio libre del almacenamiento dinámico pasa a estar por debajo de un umbral determinado de forma dinámica.

Recogida de basura y prioridades

La hebra de recogida de basura se tiene que ejecutar con una prioridad superior a la hebra de prioridad más alta que genera basura en el almacenamiento dinámico; en caso contrario, es posible que no se ejecute según lo especificado por la utilización configurada. Las hebras de Java común y hebras de tiempo real pueden generar basura y, por lo tanto, la recogida de basura se debe ejecutar con una prioridad superior a las hebras comunes y hebras de tiempo real. Esta priorización la gestiona automáticamente la JVM y la recogida de basura se ejecuta con un prioridad de 0,5 por encima de la prioridad más alta de todas las hebras comunes y hebras de tiempo real. Sin embargo, es importante asegurarse de que hebras en tiempo real de almacenamiento no dinámico (NHRT) no se vean afectados por la recogida de basura. Ejecute todos los NHRT con una prioridad más alta que el hebras de tiempo real de prioridad más alta. Esto significa que los NHRT se ejecutarán con una prioridad más alta que la recogida de basura y que no se retrasarán.

Tabla 2 en la página 8 muestra un ejemplo típico de las prioridades que puede definir y las prioridades de recogida de basura derivadas de su elección

Para ver una comparación entre las prioridades de Java y las del sistema operativo, consulte el apartado “Correlación y herencia de prioridades” en la página 15.

Tabla 2. Ejemplo de recogida de basura y prioridades

Hebras	Prioridades (ejemplos)
Si la hebra en tiempo real de prioridad más alta es:	20 (prioridad 43 del sistema operativo)
El recopilador de basura es:	20,5 (prioridad del sistema operativo 44)
Para garantizar que un NHRT se ejecute de forma independiente con respecto al recopilador de basura, defina una prioridad más alta que la del recopilador de basura:	21 (prioridad del sistema operativo 45) o superior.
La hebra de alarma de Metronome es:	Prioridad 46 (prioridad del sistema operativo 89)

Nota: Incluso con esta configuración, los hebras en tiempo real de almacenamiento no dinámico se verán en parte afectados por la recogida de basura, ya que la hebra de alarma de metronome se ejecuta con la prioridad más alta del sistema para garantizar que se pueda activar con regularidad y trabajar aunque la recogida de basura tenga que realizar alguna acción. El trabajo que hacer es, por supuesto, muy pequeño, por lo que no es necesario tenerlo muy en cuenta.

Recogida de basura y descarga de clases Metronome

El recopilador de basura Metronome no descarga las clases en IBM WebSphere Real Time porque puede requerir una cantidad no determinante de trabajo que provocaría valores de tiempo de pausa poco habituales.

Hebras del recopilador de basura Metronome

El recopilador de basura Metronome consta de dos tipos de hebras: una hebra de alarma simple y varias hebras de la recogida de basura. De forma predeterminada, hay una hebra de recogida de basura. Puede definir la cantidad de hebras de recogida de basura de la JVM utilizando la opción **-Xgcthreads**.

No puede cambiar el número de hebras de alarma de la JVM.

El recopilador de basura Metronome comprueba periódicamente la JVM para comprobar si el almacenamiento dinámico tiene suficiente espacio libre. Cuando la cantidad de espacio libre pase a estar por debajo del límite, el recopilador de basura Metronome desencadenará la JVM para que se inicie la recogida de basura.

Hebra de alarma

La hebra de alarma sencilla garantiza la utilización de los mínimos recursos. Se “activa” en intervalos regulares y realiza estas comprobaciones:

- La cantidad de espacio libre de la memoria de almacenamiento dinámico
- Si la recogida de basura se está realizando actualmente

Si no hay suficiente espacio libre disponible y no se está realizando la recogida de basura, la hebra de alarma desencadena las hebras de recogida para iniciar la recogida de basura. La hebra de alarma no realiza ninguna acción hasta la próxima vez que compruebe la JVM, según lo planificado.

Hebras de recogida

Cada hebra de recogida comprueba Java y hebras de tiempo real en busca de objetos de almacenamiento dinámico. Comprueban las áreas de memoria con la secuencia siguiente:

1. Memoria de ámbito para identificar y marcar los objetos activos del almacenamiento dinámico que estén siendo utilizados por objetos de la memoria de ámbito.
2. Memoria inmortal para identificar y marcar los objetos activos del almacenamiento dinámico que estén siendo utilizados por objetos de la memoria inmortal.
3. Memoria de almacenamiento dinámico para identificar y marcar objetos activos.

Cuando los objetos activos se hayan marcado, los objetos no marcados estarán listos para la recogida.

Una vez completado el ciclo de recogida de basura, el recopilador de basura Metronome comprueba el espacio de almacenamiento dinámico liberado. Si sigue sin ser suficiente espacio de almacenamiento dinámico libre, se inicia otro ciclo de recogida de basura que utiliza el mismo ID de desencadenante. Si hay suficiente espacio de almacenamiento dinámico libre, el desencadenante finaliza y se detienen las hebras de recogida de basura. La hebra de alarma sigue supervisando el espacio de almacenamiento dinámico libre y desencadenará otro ciclo de recogida de basura cuando sea necesario.

Para obtener más información sobre el uso de la Recogida de basura metronome, consulte "Utilización del recopilador de basura Metronome" en la página 76.

Compiladores

IBM WebSphere Real Time for RT Linux admite varios modelos de compilación de código, que proporcionan varios niveles de determinación y rendimiento de código.

Las opciones disponibles para la compilación de código Java con IBM WebSphere Real Time for RT Linux incluyen:

Compilación Just-In-Time (JIT) de prioridad baja

El modelo de compilación predeterminado en WebSphere Real Time for RT Linux utiliza un compilador Just-In-Time para compilar los métodos importantes de una aplicación Java mientras se ejecuta la aplicación. En esta modalidad, el compilador JIT funciona de manera parecida a como funciona cuando se encuentra en una JVM no de tiempo real. La diferencia es que el compilador JIT de WebSphere Real Time for RT Linux se ejecuta con un nivel de prioridad más bajo que cualquier hebra de tiempo real. La prioridad inferior implica que el compilador JIT utiliza recursos del sistema cuando la aplicación no necesita realizar tareas en tiempo real. El efecto es que el compilador JIT no afecta significativamente al rendimiento de las tareas en tiempo real.

Código precompilado AOT (Ahead-Of-Time)

WebSphere Real Time for RT Linux compila los métodos Java en código nativo, en un paso de precompilación anterior a la ejecución de la aplicación. La utilización del código AOT precompilado proporciona el nivel más alto de determinación, con un buen rendimiento.

Modo mixto, que combina el código precompilado AOT y la compilación JIT de prioridad baja

Los códigos compilados AOT y JIT se pueden utilizar juntos mientras se ejecuta la aplicación. Esta modalidad de operación puede proporcionar una buena determinación y un buen rendimiento, además de un rendimiento muy alto para aquellos métodos que se ejecutan con frecuencia.

Operación interpretada

El intérprete ejecuta una aplicación Java, pero no utiliza la compilación de códigos para nada.

Para obtener más información sobre el uso de código compilado, consulte "Utilización del código compilado con WebSphere Real Time for RT Linux" en la página 45.

Comparación de las compilaciones JIT y AOT

La compilación AOT (Ahead-of-Time) le permite compilar métodos y clases Java antes de ejecutar su código. La compilación AOT evita el efecto de temporización imprevisible que puede tener el compilador JIT sobre las vías de acceso sensibles al rendimiento. Para asegurarse de que su código esté compilado antes de ejecutarlo y para obtener el nivel más alto de rendimiento determinante, puede precompilar su código en una memoria caché de clase compartida utilizando el compilador AOT.

Nota: El código compilado AOT no se suele ejecutar tan rápidamente como el código compilado JIT.

El compilador JIT (Just-In-Time) se ejecuta como hebra SCHED_OTHER de alta prioridad, y se ejecuta por encima de la prioridad de las hebras de Java estándar, pero por debajo de la prioridad de las hebras de tiempo real. La compilación JIT, por lo tanto, no provoca retardos no determinantes en el código en tiempo real. Como consecuencia, un trabajo en tiempo real importante se realiza de manera puntual porque no será anulado por el compilador JIT. Es posible que el código en tiempo real se ejecute como código interpretado, sin embargo, porque el JIT no ha tenido tiempo suficiente como para compilar los métodos activos de la cola. Se muestra una comparación, Figura 2 en la página 11.

En general, si su aplicación pasa por una fase de calentamiento, resulta más eficaz realizar la ejecución con JIT y, si es necesario, inhabilitar JIT una vez finalizada la fase de calentamiento. Este enfoque permite al compilador JIT generar código para el entorno en el que se ejecuta su aplicación.

Si la aplicación no tiene fase de calentamiento y no está claro si las vías de acceso clave de la ejecución se han compilado mediante un funcionamiento estándar de la aplicación, la compilación AOT se ejecutará correctamente en este entorno.

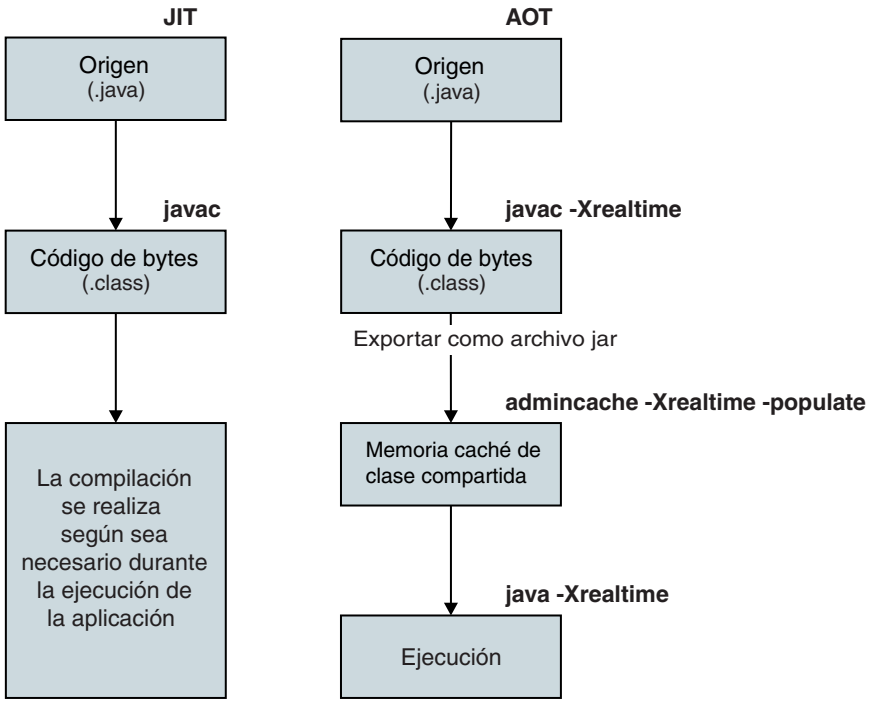

Figura 2. Comparación entre el compilador JIT y el compilador AOT.

Planificación de hebras

Las políticas de planificación de Linux se pueden utilizar con hebras regulares de Java y con hebras en tiempo real de Java para ajustar aplicaciones de tiempo real.

Con WebSphere Real Time for RT Linux, puede ejecutar las hebras regulares de Java con la política de programación de SCHED_RR o SCHED_FIFO. La utilización de la política SCHED_RR o SCHED_FIFO le ofrece un mayor control sobre su aplicación, lo que puede mejorar el rendimiento en tiempo real de las hebras de Java. La máquina virtual Java detecta la prioridad y la política de la hebra principal cuando Java se inicia con la política SCHED_RR o SCHED_FIFO. La JVM altera las correlaciones de política y prioridad en consecuencia. Para obtener más información sobre la modificación de políticas y prioridades de hebras regulares de Java, consulte "Asignación y planificación de hebras" en la página 41.

La asignación y planificación de hebras de Java de tiempo real forma parte de Real Time Specification for Java (RTSJ). Puede consultar este tema, incluidas las políticas de planificación y el manejo de prioridades de hebras Java de tiempo real, en "Soporte para RTSJ" en la página 12.

Entre las políticas de planificación de Linux se incluyen las siguientes:

SCHED_OTHER
: Política de planificación de compartición de tiempo universal predeterminada que utilizan la mayoría de las hebras. Estas hebras se deben asignar con una prioridad de cero.

SCHED_OTHER utiliza la distribución en porciones de tiempo, lo que significa que cada hebra se ejecuta durante un período de tiempo limitado, tras el cual se puede ejecutar la siguiente hebra.

SCHED_FIFO
Se puede utilizar solo con prioridades mayores que cero. Cuando una hebra SCHED_FIFO pasa a estar disponible, esta tiene prioridad sobre cualquier hebra SCHED_OTHER normal.

Si una hebra SCHED_FIFO con una prioridad más alta pasa a estar disponible, esta tiene prioridad sobre las demás hebras SCHED_FIFO existentes con prioridad más baja. Entonces, esta hebra se mantiene en lo alto de la cola según su prioridad.

No hay distribución en porciones de tiempo.

SCHED_RR
Es una mejora de SCHED_FIFO. La diferencia es que cada hebra se puede ejecutar solo durante un período de tiempo limitado. Si la hebra supera dicho tiempo, se devuelve a la lista según su prioridad.

Para obtener más detalles sobre estas políticas de planificación de Linux, consulte la página **man** para **sched_setscheduler**.

Para obtener más información sobre el uso de las políticas de planificación de Linux con WebSphere Real Time for RT Linux, consulte "Asignación y planificación de hebras" en la página 41.

Soporte para RTSJ

WebSphere Real Time for RT Linux implementa Especificación de tiempo real para Java (RTSJ).

WebSphere Real Time for RT Linux versión 3.0 se ha certificado como compatible con RTSJ 1.0.2 con respecto a RTSJ Technology Compatibility Kit versión 3.1.0 FCS y es también compatible con Java Compatibility Kit (JCK) para la versión 7.0.

Asignación y planificación de hebras de tiempo real

La asignación y planificación de hebras de Java de tiempo real forma parte de Real Time Specification for Java. La política de planificación SCHED_FIFO se utiliza para dar prioridad a las hebras de real-time Java de tiempo real que utilizan las prioridades 11 - 89 del sistema operativo Linux.

Encontrará información sobre las políticas de planificación de Linux en "Asignación y planificación de hebras" en la página 41.

Objetos programables y sus parámetros

Hay dos tipos principales de objetos programables en tiempo real: hebras de tiempo real y manejadores de sucesos asíncronos.

Estos objetos programables tienen los parámetros siguientes asociados:

SchedulingParameters
PriorityParameters planifica los objetos programables en tiempo real por prioridades.

ReleaseParameters

- **PeriodicParameters** describe la publicación periódica de los objetos programables en tiempo real. Una hebra en tiempo real periódica es aquella que se publica en intervalos periódicos.
- **AperiodicParameters** describe la publicación de objetos programables en tiempo real. Las hebras en tiempo real aperiódicas se publican en intervalos irregulares.

MemoryParameters
Describe las restricciones de asignación de memoria para objetos programables en tiempo real.

ProcessingGroupParameters
No admitido en WebSphere Real Time for RT Linux.

Planificador de prioridades

En WebSphere Real Time for RT Linux, el planificador es un planificador de prioridades. Como indica su nombre, gestiona la ejecución de objetos programables de acuerdo con sus prioridades activas.

El planificador mantiene la lista de objetos programables y determina cuándo se puede liberar cada objeto para ejecutarse en la CPU. El planificador debe cumplir con los diversos parámetros asociados a cada objeto programable. Con este fin, se proporcionan los métodos addToFeasibility, isFeasible y removeFromFeasibility.

Prioridades y políticas

Las hebras de Java común, es decir, las hebras asignadas como objetos `java.lang.Thread`, pueden utilizar políticas de planificación SCHED_OTHER, SCHED_RR o SCHED_FIFO. Las hebras de Real Time, es decir, las hebras asignadas como `java.lang.RealtimeThread`, y los manejadores de sucesos asíncronos utilizan la política de planificación SCHED_FIFO.

Las hebras de Java común utilizan la política de planificación predeterminada de SCHED_OTHER, a menos que la JVM se inicie mediante una hebra con la política SCHED_RR o SCHED_FIFO. Las hebras de Java común que utilizan la política SCHED_OTHER tienen la prioridad de hebra de sistema operativo definida en 0. Las hebras de Java común que utilizan la política SCHED_RR o SCHED_FIFO heredan la prioridad de la hebra que inicia la JVM. Para obtener más información sobre las prioridades y las políticas de las hebras regulares de Java, consulte “Políticas y prioridades de las hebras de Java común” en la página 42.

Para hebras de tiempo real, la política SCHED_FIFO no tiene distribución en porciones de tiempo y admite 99 prioridades, desde 1 (la más baja) hasta 99 (la más alta). Esta implementación de WebSphere Real Time for RT Linux admite 28 prioridades de usuario en el rango 11-38, ambas incluidas, de manera que:

```
javax.realtime.PriorityScheduler().getMinPriority()
```

devuelve 11, y:

```
javax.realtime.PriorityScheduler().getMaxPriority()
```

devuelve 38.

Las prioridades de sistema operativo 81 - 89 las utiliza la JVM de IBM para asignar las hebras Worker. Estas hebras se han diseñado para realizar una pequeña parte del trabajo antes de volver a la inactividad. Las hebras son las siguientes:

- La hebra de alarma Recopilador de basura Metronome se ejecuta con una prioridad de 89. Esta hebra se ejecuta con regularidad y asigna una unidad de trabajo del recopilador de basura.
- Dos hebras de señal asíncrona, que procesan las señales asíncronas, una de ellas una hebra en tiempo real de almacenamiento no dinámico (NHRT) con una prioridad 88 y la otra con prioridad 87.
- Dos hebras de temporizador, con sucesos del temporizador de asignación, una de ellas una hebra en tiempo real de almacenamiento no dinámico para temporizadores no de pila con una prioridad de 85 y la otra con prioridad 83.
- Las hebras del manejador de sucesos asíncrono, que se asignan para ejecutar manejadores de sucesos asíncronos y a las que, mientras ejecutan un manejador de sucesos asíncronos, se les asigna la prioridad de dicho manejador. El sistema se inicia con dos hebras de manejador de tiempo real no de pila con la prioridad 85 y otros 8 con la prioridad 83.
- La hebra en tiempo real de almacenamiento no dinámico de señal asíncrona con la prioridad 88 maneja las solicitudes de volcados de almacenamiento dinámico, volcados del núcleo y volcados Java core. Aumenta temporalmente su prioridad a 89 mientras crea los archivos de volcado.

La hebra de rastreo del recopilador de basura Metronome se ejecuta en la prioridad 12 del sistema operativo, y la hebra de muestreo del JIT, que toma como muestra los métodos de Java para la compilación, se ejecuta en una prioridad 13 del sistema operativo.

La hebra de compilación JIT (distinta de la hebra de muestreo del JIT) se ejecuta con una política SCHED_OTHER en la prioridad 0 del sistema operativo.

Las hebras de compilación JIT y de muestreo del JIT se encuentran inhabilitadas si se ha especificado **-Xnojit** o **-Xint**.

La prioridad del Recopilador de basura Metronome y el finalizador cambia constantemente (antes de cada rutina de recogida) para estar por encima de la hebra de asignación de almacenamiento dinámico de mayor prioridad. Debe asegurarse de que la prioridad de las hebras de asignación de almacenamiento dinámico se encuentre por debajo de la de NoHeapRealtimeThreads.

Una hebra de asignación de almacenamiento dinámico es cualquier hebra de usuario distinta a NHRT que no esté inactiva o bloqueada en un supervisor. No se considera que una hebra de usuario que ejecuta código nativo fuera de la interfaz JNI sea de asignación de almacenamiento dinámico. Si hay una recogida de basura en curso cuando una hebra de asignación de almacenamiento dinámico se activa, deja de estar bloqueada en el supervisor, o deja JNI, se ve forzada a esperar hasta que la recogida de basura haya terminando antes de continuar.

La prioridad de sistema operativo 81 se reserva para las hebras internas de la JVM que se asignan desde el almacenamiento dinámico. Si una hebra interna de la JVM se encuentra en la prioridad de sistema operativo 81, el recopilador de basura se ejecuta con la prioridad de sistema operativo 82. Cuando las únicas hebras de usuario asignadas al almacenamiento dinámico no son hebras de tiempo real, la prioridad del recopilador de basura se ejecuta en la prioridad 11 del sistema operativo. De lo contrario, el recopilador de basura se ejecuta en una prioridad que se encuentra una prioridad de sistema operativo por encima de la hebra de usuario de asignación de almacenamiento dinámico de máxima prioridad.

La prioridad del recopilador de basura se ajusta justo antes de una rutina de recogida.

Correlación y herencia de prioridades

Cada prioridad de Java está asociada a una prioridad base del sistema operativo, y cada prioridad del sistema operativo está asociada a una política de planificación. Las políticas de planificación del sistema operativo WebSphere Real Time for RT Linux Linux son SCHED_OTHER, SCHED_RR y SCHED_FIFO.

Las hebras de Java de tiempo real utilizan la política SCHED_FIFO, mientras que las hebras de Java común utilizan la política de la hebra que inicia la JVM. La política de planificación predeterminada para las hebras de Java común es SCHED_OTHER, pero puede utilizar un programa de utilidad como **chrt** para definir las políticas SCHED_RR o SCHED_FIFO. Para obtener más información sobre las políticas y prioridades de hebras, consulte el apartado "Asignación y planificación de hebras" en la página 41.

En la tabla siguiente se muestra cómo se correlacionan las prioridades de Java con las prioridades del sistema operativo nativo. Algunas prioridades de Java se reservan para ser utilizadas por la JVM, y otras prioridades nativas sin prioridades Java correspondientes son utilizadas también por la JVM.

Nota:

- Las prioridades 1-10 son utilizadas por hebras de Java común.
  - Para la política SCHED_OTHER, las prioridades de Java 1-10 se correlacionan con la prioridad 0 del sistema operativo.
  - Para las políticas SCHED_FIFO o SCHED_RR, las prioridades de Java 1-10 heredan la prioridad de la hebra que inicia la JVM.
- Las prioridades de la 11 en adelante son utilizadas por hebras de tiempo real y hebras en tiempo real de almacenamiento no dinámico
- Un objeto programable siempre se ejecuta con su prioridad activa. La prioridad activa es en un principio la prioridad básica del objeto programable, pero la prioridad activa se puede ver temporalmente elevada por la herencia de prioridades. La prioridad básica de un objeto programable se puede cambiar mientras se ejecuta.

Prioridades de la base de usuarios:

```
Java priorities 1-10: SCHED_OTHER, OS priority 0

Java priority 11: SCHED_FIFO, OS priority 25
Java priority 12: SCHED_FIFO, OS priority 27
Java priority 13: SCHED_FIFO, OS priority 29
Java priority 14: SCHED_FIFO, OS priority 31
Java priority 15: SCHED_FIFO, OS priority 33
Java priority 16: SCHED_FIFO, OS priority 35
Java priority 17: SCHED_FIFO, OS priority 37
Java priority 18: SCHED_FIFO, OS priority 39
Java priority 19: SCHED_FIFO, OS priority 41
Java priority 20: SCHED_FIFO, OS priority 43
Java priority 21: SCHED_FIFO, OS priority 45
Java priority 22: SCHED_FIFO, OS priority 47
Java priority 23: SCHED_FIFO, OS priority 49
Java priority 24: SCHED_FIFO, OS priority 51
Java priority 25: SCHED_FIFO, OS priority 53
Java priority 26: SCHED_FIFO, OS priority 55
Java priority 27: SCHED_FIFO, OS priority 57
Java priority 28: SCHED_FIFO, OS priority 59
Java priority 29: SCHED_FIFO, OS priority 61
Java priority 30: SCHED_FIFO, OS priority 63
```

```
Java priority 31: SCHED_FIFO, OS priority 65
Java priority 32: SCHED_FIFO, OS priority 67
Java priority 33: SCHED_FIFO, OS priority 69
Java priority 34: SCHED_FIFO, OS priority 71
Java priority 35: SCHED_FIFO, OS priority 73
Java priority 36: SCHED_FIFO, OS priority 75
Java priority 37: SCHED_FIFO, OS priority 77
Java priority 38: SCHED_FIFO, OS priority 79
```

Prioridades de base interna:

```
Internal Java priority 39: SCHED_FIFO, OS priority 81
Internal Java priority 40: SCHED_FIFO, OS priority 83
Internal Java priority 41: SCHED_FIFO, OS priority 84
Internal Java priority 42: SCHED_FIFO, OS priority 85
Internal Java priority 43: SCHED_FIFO, OS priority 86
Internal Java priority 44: SCHED_FIFO, OS priority 87
Internal Java priority 45: SCHED_FIFO, OS priority 88
OS priorities 11, 12, 13
OS priorities even numbers 26, 28, 30, ..., 82
OS priority 89
```

Consulte también la sección "Sincronización" en http://www.rtsj.org/specjavadoc/book_index.html.

Herencia de prioridades:

La prioridad activa de una hebra puede aumentar temporalmente porque contiene un bloqueo requerido por una hebra de prioridad más alta. Estos bloqueos pueden ser bloqueos internos de la JVM o bien supervisores de nivel de usuario asociados a métodos sincronizados o bloqueos sincronizados. La prioridad de una hebra de Java común, por lo tanto, puede tener temporalmente una prioridad de tiempo real, hasta el punto en el que la hebra libere el bloqueo.

Una consecuencia de la herencia de prioridades es que la política de hebras de una hebra SCHED_OTHER cambia temporalmente a SCHED_FIFO.

Para obtener más información sobre las prioridades activas y de base, consulte la sección "Sincronización" de la especificación de RTSJ.

Gestión de memoria

La recogida de basura de almacenamientos dinámicos de memoria siempre se ha visto como un obstáculo para la programación en tiempo real, debido al comportamiento imprevisible introducido por la recogida de basura. El recopilador de basura Metronome de IBM WebSphere Real Time for RT Linux puede proporcionar un gran rendimiento determinante en el recopilador de basura. Al mismo tiempo, Especificación de tiempo real para Java (RTSJ) proporciona varias ampliaciones sobre el modelo de memoria para los objetos que se encuentran fuera del almacenamiento dinámico con recogida de basura, de manera que un programador Java puede gestionar explícitamente tanto los objetos de larga duración como los de corta duración.

Áreas de memoria

RTSJ introduce el concepto de un área de memoria que se puede usar para la asignación de objetos. Algunas áreas de memoria existen fuera del almacenamiento dinámico y plantean restricciones sobre lo que pueden hacer con los objetos el sistema y el recopilador de basura. Por ejemplo, los objetos de algunas áreas de memoria nunca pasan por la recogida de basura, pero el recopilador de basura

puede analizar estas áreas de memoria en busca de referencias a cualquier objeto del almacenamiento dinámico para conservar la integridad del almacenamiento dinámico.

La gestión de memoria tiene tres tipos básicos:

- El almacenamiento dinámico es el almacenamiento dinámico tradicional de Java, pero lo gestiona Recopilador de basura Metronome.
- La memoria de ámbito debe ser específicamente solicitada por aplicaciones y solo la puede utilizar hebras de tiempo real, incluidas las hebras en tiempo real de almacenamiento no dinámico y los manejadores de sucesos asíncronos no de pila.
- La memoria inmortal representa un área de memoria que contiene objetos a los que puede hacer referencia cualquier objeto programable, incluidos específicamente los manejadores de eventos asíncronos de hebras en tiempo real de almacenamiento no dinámico no de pila. La usa la carga de clases y la inicialización estática, incluso aunque la aplicación no la use.

La memoria inmortal o de ámbito se puede designar para utilizar memoria física, que consta de regiones de memoria con características específicas, como un acceso bastante más rápido. En general, la memoria física no se usa con frecuencia y es poco probable que afecte a un usuario estándar de la máquina virtual Java.

Memoria de almacenamiento dinámico

El tamaño máximo viene controlado por **-Xmx**, pero recuerde *no* definir el tamaño de almacenamiento dinámico inicial (**-Xms**) o bien haga que sea igual al tamaño de almacenamiento dinámico máximo **-Xmx**, porque, en tiempo real, el almacenamiento dinámico nunca va del tamaño de almacenamiento dinámico inicial al tamaño de almacenamiento dinámico máximo. Si alcanza el tamaño de almacenamiento dinámico máximo y no hay espacio libre, se generará la excepción OutOfMemoryError. Por lo general, la JVM en tiempo real consume más memoria de almacenamiento dinámico que la JVM tradicional, porque dar soporte a la colección determinante requiere que los objetos se organicen de manera distinta, lo que resulta en una mayor fragmentación del almacenamiento dinámico. Además, las matrices se dividen en fragmentos, cada uno de los cuales tiene una cabecera. Depende de la proporción de objetos grandes y pequeños, y del uso de la matriz, pero es probable que una aplicación necesite un 20% más de espacio de almacenamiento dinámico.

Recopilador de basura Metronome es parecido al colector "casi simultáneo" que existe en la principal máquina virtual Java, en el sentido de que recoge la basura mientras se ejecuta la aplicación. En un mundo perfecto, el ciclo de colección se completaría antes de que la aplicación se quedase sin memoria, pero algunas aplicaciones con tasas de asignación muy altas pueden realizar asignaciones más rápido de lo que Recopilador de basura Metronome puede recopilar. Hay varios controles detallados que afectan a la tasa de colección, pero hay un control que fuerza a Metronome a volver a la tradicional operación STW (stop-the-world) del recopilador de basura antes de generar la excepción OutOfMemoryError. El parámetro de tiempo de ejecución es **-Xgc:synchronousGCOnOOM** y la contrapartida es **-Xgc:nosynchronousGCOnOOM**. El valor predeterminado es **-Xgc:synchronousGCOnOOM**.

Memoria de ámbito

RTSJ introduce el concepto de memoria de ámbito. Se puede utilizar con objetos que tienen un tiempo de vida bien definido. Un ámbito se puede especificar de

forma explícita, o bien se puede adjuntar a un objeto programable (una hebra en tiempo real o un manejador de eventos asíncrono) que se introduce de forma eficaz en el ámbito antes de que se ejecute el método run() del objeto. Cada ámbito tiene un recuento de referencias y, cuando llega a cero, los objetos que se encuentran en dicho ámbito se pueden cerrar (finalizar) y la memoria asociada a dicho ámbito se libera. La reutilización del ámbito se bloquea hasta que la finalización termina.

La memoria de ámbito se puede dividir en dos tipos: VTMemory y LTMemory. Estos tipos de memoria de ámbito varían según el tiempo requerido para asignar objetos desde el área. LTMemory garantiza la asignación de tiempo lineal cuando el consumo de memoria del área de memoria es inferior al tamaño inicial del área de memoria. VTMemory no ofrece estas garantías.

Los ámbitos se pueden anidar. Cuando se especifica un ámbito anidado, todas las asignaciones subsiguientes se toman de la memoria asociada con el nuevo ámbito. Cuando se completa el ámbito anidado, el ámbito previo se restaura y las asignaciones subsiguientes se vuelven a tomar de dicho ámbito.

Debido al tiempo de vida de los objetos del ámbito, es necesario limitar las referencias a los objetos del ámbito, por medio de un conjunto limitado de reglas de asignación. No se puede asignar una referencia a un objeto del ámbito a una variable desde un ámbito externo, o a un campo de un objeto en el área de almacenamiento dinámico o inmortal. La referencia a un objeto del ámbito solo se puede asignar en el mismo ámbito o en un ámbito interno. La máquina virtual detecta intentos de asignaciones incorrectos y genera una excepción IllegalAssignmentError cuando dichos intentos se producen. La flexibilidad proporcionada en la selección de tipos de memorias de ámbito permite a la aplicación utilizar un área de memoria que tiene características adecuadas a una región del código con particularidades sintácticas.

El tamaño del área se debe especificar durante la construcción del área, y un parámetro de la línea de mandatos, **-Xgc:scopedMemoryMaximumSize**, controla el valor máximo. El valor predeterminado es 8 MB y es adecuado para la mayoría de los objetivos.

Memoria inmortal

La memoria inmortal es un recurso de memoria compartida entre todos los objetos y las hebras programables de una aplicación. Los objetos asignados en la memoria inmortal siempre están a disposición de los manejadores de eventos asíncronos y las hebras no de pila, y no están sujetos a retardos causados por la recogida de basura. Los objetos son liberados por el sistema cuando el programa termina.

El tamaño lo controla **-Xgc:immortalMemorySize**; por ejemplo, **-Xgc:immortalMemorySize=20m** define 20 MB, el valor predeterminado es 16 MB, que suele ser el adecuado, a menos que esté realizando una gran carga de clases. La carga de clases es la causa más probable en la mayoría de las excepciones OutOfMemoryError.

Estimación de los requisitos de memoria

Cómo obtener la información necesaria para asignar la memoria suficiente

Un método razonable consiste en identificar la memoria necesaria para contener los objetos esperados, con un margen de seguridad razonable. El análisis de la aplicación ayuda a identificar la cantidad y la naturaleza de los objetos necesarios, aunque el tamaño actual necesario para un objeto puede variar entre los distintos

sistemas. Al utilizar la clase SizeEstimator, se tiene en cuenta el tamaño real del objeto, lo que proporciona información más fiable.

Clase SizeEstimator

La clase SizeEstimator proporciona información de orientación sobre la memoria necesaria para almacenar un objeto. La estimación es un indicativo del espacio de memoria mínimo que se debe asignar al propio objeto, y no tiene en cuenta los requisitos de memoria de la cuenta para ningún otro recurso que pueda necesitar el objeto, por ejemplo durante su construcción.

Para obtener detalles sobre esta clase, consulte la publicación http://www.rtsj.org/specjavadoc/book_index.html

Utilización de la memoria

Comparación de las hebras Java, hebras de tiempo real yhebras en tiempo real de almacenamiento no dinámico .

Especificación de tiempo real para Java (RTSJ) añade dos clases para admitir las hebras de tiempo real: clase RealtimeThread y clase NoHeapRealtimeThread.

- Tanto las hebras en tiempo real como hebras en tiempo real de almacenamiento no dinámico son objetos programables. Como tales, tienen los parámetros siguientes: release, planificación, memoria y grupo de procesos.
- Las hebras en tiempo real pueden acceder a objetos de la memoria de almacenamiento dinámico, así como a la memoria inmortal y de ámbito.
- Las hebras en tiempo real de almacenamiento no dinámico solo acceder a las áreas de memoria inmortal y de ámbito.
- Las hebras en tiempo real de almacenamiento no dinámico necesitan una prioridad más alta que otras hebras de tiempo real. Si su prioridad es inferior a la de otras hebras de tiempo real, pierden la ventaja de ejecutarse sin interferencias del recopilador de basura.

Nota: Una hebra en tiempo real de almacenamiento no dinámico con una prioridad mayor que la de otras hebras en tiempo real no será interrumpida por la recogida de basura.

Tabla 3. Acceso a la memoria por parte de hebras en tiempo real y hebras en tiempo real de almacenamiento no dinámico

Hebras	Memoria inmortal	Memoria de ámbito	Memoria de almacenamiento dinámico
Hebras normales	✓	✗	✓
Hebras de tiempo real	✓	✓	✓
hebras en tiempo real de almacenamiento no dinámico	✓	✓	✗

Tipos de áreas de memoria

Memoria inmortal

La memoria inmortal no está sujeta a la recogida de basura. Después de asignar espacio en la memoria inmortal, el espacio no se puede reclamar hasta que se sale de la aplicación.

- Debido a la naturaleza de la memoria inmortal, puede que desee encontrar otras formas de reutilizar la memoria. Una posibilidad es crear una agrupación de objetos reutilizables. La utilización de la memoria de ámbito es una alternativa.
- Los objetos de la memoria inmortal no pueden hacer referencia a ningún elemento de la memoria de ámbito. Si a un campo de un objeto de la memoria inmortal se le ha asignado un objeto de la memoria de ámbito, se genera una excepción de tipo IllegalAssignmentError.

Memoria de ámbito
La memoria de ámbito se puede utilizar como área de memoria inicial de un objeto programable o puede ser especificada por uno de ellos. Cuando se le deje de hacer referencia, se borrarán todos los objetos del área. Los objetos programables que se ejecutan en un área de memoria de ámbito realizan todas las asignaciones de objetos desde dicha área. Si un área de memoria de ámbito no se utiliza, los objetos que se encuentran en ella se finalizan y se reclama la memoria, a fin de preparar el ámbito para ser reutilizado. Cuando la memoria de ámbito deja de estar disponible para los objetos programables, la memoria se reclama para otros usos.

El área de memoria descrita por una instancia ScopedMemory no existe en el almacenamiento dinámico Java y no está sujeta a la recogida de basura. Es seguro utilizar un objeto ScopedMemory como área de memoria inicial asociada a NoHeapRealtimeThread o bien especificar el área de memoria utilizando el método ScopedMemory.enter dentro de NoHeapRealtimeThread.

Memoria física
Utilice la memoria física cuando las características de la propia memoria sean importantes; por ejemplo, no paginable o no volátil.

Esquema de asignación de tiempo lineal (LTMemory)
LTMemory representa un área de memoria a la que el sistema le garantiza tener asignación de tiempo lineal cuando el consumo de memoria del área de memoria es inferior al tamaño inicial del área de memoria. El tiempo de ejecución de la asignación puede variar si el consumo de memoria se encuentra entre el tamaño inicial y el tamaño máximo del área. Además, el sistema subyacente no tiene que garantizar que siempre haya disponible memoria entre la inicial y la máxima.

Esquema de asignación de tiempo variable (VTMemory)
VTMemory es similar a LTMemory, salvo por el hecho de que no es necesario que el tiempo de ejecución de una asignación desde el área **`VTMemory`** se complete en tiempo lineal.

Memoria de almacenamiento dinámico
Los objetos de la memoria de almacenamiento dinámico no pueden hacer referencia a ningún elemento de la memoria de ámbito. Si a un campo de un objeto de la memoria de almacenamiento dinámico se le ha asignado un objeto de la memoria de ámbito, se genera una excepción de tipo IllegalAssignmentError.

Sincronización y compartición de recursos

En un sistema de tiempo real, cuando hay tres o más hebras ejecutándose con diferentes prioridades y que se están sincronizando entre ellas, se puede general una condición denominada inversión de prioridades, en la que una hebra de prioridad más alta queda bloqueada en la ejecución por una hebra de prioridad más baja durante un período de tiempo prolongado. WebSphere Real Time for RT Linux utiliza un esquema denominado herencia de prioridades para evitar esta condición.

Cuando una tarea de prioridad más baja bloquea la ejecución de una tarea de prioridad superior, la prioridad de la tarea de prioridad más baja aumentará temporalmente para coincidir con la prioridad más alta hasta que la tarea de prioridad superior deje de estar bloqueada.

Parámetros periódicos y aperiódicos

Las hebras en tiempo real tienen varios parámetros de release, que determinan la frecuencia con la que se publica un objeto programable. Los parámetros periódicos y aperiódicos son ejemplos de parámetros de publicación.

Parámetros periódicos

Esta clase es para los objetos programables que se publican en intervalos regulares.

AbsoluteTime
Se expresa en milisegundos y nanosegundos.

RelativeTime
Es el período de tiempo de un suceso dado expresado en milisegundos y nanosegundos. Por ejemplo, puede medir el tiempo absoluto en el que se inicia y finaliza un suceso. A continuación, puede calcular el tiempo relativo como la diferencia entre las dos medidas.

Parámetros aperiódicos

Esta clase la usan los objetos programables publicados en intervalos irregulares. Dado que se puede producir un segundo suceso aperiódico después de que termine el primero, puede definir la longitud de la cola de solicitudes pendientes.

Manejo de sucesos asíncronos

Los manejadores de sucesos asíncronos reaccionan a los sucesos que se producen fuera de una hebra; por ejemplo, la entrada desde la interfaz de una aplicación. En sistemas de tiempo real, estos sucesos deben responder en las fechas límites definidas para su aplicación.

Los sucesos asíncronos se pueden asociar a las interrupciones del sistema y las señales POSIX, y los sucesos asíncronos se pueden vincular a un temporizador.

Al igual que hebras de tiempo real, manejadores de sucesos asíncronos tiene varios parámetros asociados. Para obtener una lista de dichos parámetros, consulte el apartado “Objetos programables y sus parámetros” en la página 12.

Manejadores de señal

POSIXSignalHandler admite las señales SIGQUIT, SIGTERM y SIGABRT. El comportamiento predeterminado de SIGQUIT hace que se genere un volcado Java.

La generación del volcado Java no interfiere con el funcionamiento de un programa activo, salvo por el tiempo de la CPU y por la lectura y escritura de archivos. La generación de un volcado Java interrumpe el programa hasta que el volcado Java finaliza; el rendimiento de la aplicación no será previsible mientras se generen volcados Java.

Para suprimir toda la generación de volcado Java y de núcleos en una anomalía, utilice **-Xdump:none**.

Para suprimir solo el volcado del sistema y la generación del volcado Jaca en una señal SIGQUIT, especifique **-Xdump:java:none -Xdump:java:events=gpf+abort**.

Las señales siguientes se pueden adjuntar a manejadores de sucesos asíncronos (AEH) mediante el mecanismo POSIXSignalHandler (descripciones de señales, según lo definido en `/usr/include/bits/signum.h`):

```
#define SIGQUIT         3       /* Quit (POSIX).  */
#define SIGABRT         6       /* Abort (ANSI).  */
#define SIGKILL         9       /* Kill, unblockable (POSIX).  */
```

No se admiten más señales actualmente. Todas las señales anteriores son señales asíncronas, y resulta imposible admitir los adjuntos en señales asíncronas (como SIGILL y SIGSEGV) porque indican una anomalía de la aplicación o del código JVM, no de un suceso generado de forma externa.

Nota: La señal SIGQUIT causa, de forma predeterminada, que la aplicación Java genere volcados (por ejemplo, un volcado Java) cuando la JVM la recibe. Aunque adicionalmente se entrega a cualquier AEH adjunta, esta entrega puede provocar confusión o un comportamiento no deseado, y puede inhabilitarla con la opción **-Xdump:none:events=user** de la línea de mandatos de Java.

Documentación necesaria

WebSphere Real Time for RT Linux implementa Especificación de tiempo real para Java (RTSJ).

WebSphere Real Time for RT Linux versión 2.0 se ha certificado como compatible con RTSJ 1.0.2 con respecto a RTSJ Technology Compatibility Kit versión 3.0.13 FCS y es compatible con Java Compatibility Kit (JCK) para la versión 6.0.

Recursos admitidos

Se admiten los recursos siguientes:

- La obligatoriedad de la tasa de asignación en el almacenamiento dinámico para limitar la velocidad con la que un objeto programable crea objetos en el almacenamiento dinámico.

Recursos no soportados

No se admiten los recursos siguientes:

- Emulación del protocolo de techo de prioridad. Por ejemplo, no permite que se utilice PriorityCeilingEmulation como política de control de supervisión.
- Soporte de acceso atómico, excepto cuando es necesario por razones de conformidad con la especificación.
- No hay más planificadores que el planificador de prioridad básica a disposición de las aplicaciones.

- Obligatoriedad de costes.

Documentación necesaria para Especificación de tiempo real para Java

La sección *Documentación necesaria* de Especificación de tiempo real para Java (RTSJ) se cita en esta sección. Se anota cualquier desviación de la implementación estándar de RTSJ.

1. **El algoritmo de pruebas de viabilidad es el predeterminado.**

   "Si el algoritmo de pruebas de viabilidad no es el predeterminado, documente el algoritmo de pruebas de viabilidad."
2. **Solo el planificador de prioridad básica está a disposición de las aplicaciones.**

   "Si hay planificadores distintos al planificador de prioridad básica a disposición de las aplicación, documente el comportamiento del planificador y su interacción con otro planificador, como se indica en el capítulo Planificación. Asimismo, documente la lista de clases que constituyen objetos programables para el planificador, a menos que dicha lista sea igual a la lista de objetos programables para el planificador básico."
3. **Un objeto programable anulado por un objeto programable de prioridad superior se colocará al principio de la cola por su prioridad.**

   "Un objeto programable anulado por un objeto programable de prioridad superior se coloca en la cola por su prioridad activa, en una posición determinada por la implementación. Si el objeto programable anulado no se coloca en la parte delantera de la cola correspondiente, la implementación debe documentar el algoritmo utilizado para dicha colocación. La colocación al principio de la cola puede ser necesaria en una futura versión de esta especificación."
4. **No se admite la obligatoriedad de costes.**

   "Si la implementación admite la obligatoriedad de costes, la implementación es necesaria para documentar la granularidad en la que se actualiza el consumo actual de la CPU. "
5. **Se admite la correlación secuencial simple.**

   "La correlación de memoria implementada por un filtro de tipo de memoria física se debe documentar, a menos que se trate de una correlación secuencial simple de bytes contiguos."
6. **No existen subclases para el recopilador de basura Metronome proporcionado con WebSphere Real Time for RT Linux.**

   "La implementación debe documentar en su totalidad el comportamiento de las subclases de GarbageCollector. "
7. **No se proporcionan subclases MonitorControl con WebSphere Real Time for RT Linux.**

   "Una implementación que proporciona subclases MonitorControl no detalladas en esta especificación debe documentar sus efectos, especialmente con respecto a un control de inversión de prioridades, y qué planificadores (en caso de que haya alguno) no admiten la nueva política. "
8. **A un objeto programable que contenga un supervisor requerido por un objeto programable de prioridad más alta, se le aumenta la prioridad a la prioridad superior hasta que publica el supervisor. Si, en ese momento, el objeto programable se va a hacer no ejecutable (es decir, hay un trabajo con más prioridad que hacer), se colocará al final de la cola por su prioridad original (sin aumentar) cuando se ejecute en kernels anteriores a SUSE**

Linux Enterprise Real Time 10 SP2 update kernel versión 2.6.22.19-0.16 y Red Hat Enterprise Linux 5.1 MRG 2.6.24.7-73 Errata 1. Los kernels de estos niveles o posteriores colocan el objeto programable al principio de la cola.

"Si al perder el "aumento" de prioridad, debido a un algoritmo de elusión de inversión de prioridades, el objeto programable no se coloca en el primer puesto de su nueva cola, la implementación debe documentar el comportamiento de la cola."

9. **El planificador básico es el único planificador que cuenta con WebSphere Real Time for RT Linux.**

   "En el caso de cualquier otro planificador disponible distinto al planificador básico, una implementación debe documentar cómo, si se da el caso, la semántica de la sincronización difiere de las reglas definidas para la instancia PriorityInheritance predeterminada. Debe proporcionar documentación para el comportamiento del nuevo planificador con la herencia de prioridades (y, si se admite, la emulación del protocolo de techo de prioridad) equivalente a la semántica del planificador de prioridad básica que se encuentra en el capítulo Sincronización. "

10. **El peor tiempo posible desde la activación de un suceso hasta la planificación de un manejador de sucesos asociado será de una media de 40 milisegundos y no superará los 100 milisegundos, siempre que no haya objetos programables opuestos ni una actividad del sistema de prioridad igual o superior, y siempre que la recogida de basura no interfiera. Si el objeto programable que controla el método de activación, el objeto AsyncEvent o el manejador hacen referencia al almacenamiento dinámico, la influencia potencial de la recogida de basura se ha documentado en (A). Se presupone que el código está siendo interpretado y que se ha configurado un manejador único (vinculado) en el suceso.**

    "El peor intervalo de respuesta posible entre la activación de un AsyncEvent debido a un hecho vinculado y la publicación de un AsyncEventHandler asociado (siempre que no haya objetos programables de mayor prioridad ejecutables) se debe documentar para la arquitectura de referencia."

11. **El peor intervalo entre la activación de una AsynchronouslyInterruptedException en una hebra habilitada para ATC y la primera entrega de dicha excepción será de una media de 35 milisegundos y no superará los 160 milisegundos, siempre que no haya objetos programables opuestos ni una actividad del sistema de prioridad igual o superior, y siempre que la recogida de basura no interfiera. La habilitación de ATC en este caso implica que la hebra se está ejecutando en un método habilitado para AI en una región que no se ha aplazado para ATC, y dichas condiciones siguen siendo verdaderas hasta que se entrega la excepción. La influencia potencial de la recogida de basura se ha documentado en (A). Si la hebra de destino está en código nativo, el retardo puede ser ilimitado. Se presupone que el código está siendo interpretado.**

    "El intervalo entre la activación de una AsynchronouslyInterruptedException en una hebra habilitada para ATC y la primera entrega de dicha excepción (siempre que no haya objetos programables de mayor prioridad ejecutables) se debe documentar para la arquitectura de referencia. "

12. **No aplicable, consulte la respuesta 4.**

    "Si se admite la obligatoriedad de costes y la implementación asigna el coste de ejecutar finalizadores para los objetos de la memoria de ámbito a cualquier objeto programable distinto del que ha causado que el recuento de referencias del ámbito bajase hasta cero al dejar el ámbito, las reglas para asignar el coste se deben documentar. "

13. **No se han producido cambios en la implementación estándar de RealtimeSecurity.**

    "Si la implementación de RealtimeSecurity es más restrictiva que la implementación necesaria, o tiene opciones de configuración de tiempo de ejecución, dichas características se deben documentar."

14. **Los finalizadores de objetos de un área de memoria de ámbito se ejecutarán según la última hebra para hacer referencia a dicha área, es decir, se ejecutarán cuando la hebra reduzca el recuento de referencia de uno a cero. Cualquier coste asociado a la ejecución de los finalizadores se asignará a dicha hebra.**

    "Una implementación puede ejecutar finalizadores para los objetos de la memoria de ámbito antes de que se vuelva a entrar en la memoria y antes de que devuelva una llamada a getReferenceCount() desde dicho ámbito. Sin embargo, debe documentarse la ejecución de dichos finalizadores. "

15. **La resolución no es definible.**

    "Para cada reloj admitido, la documentación debe especificar si la resolución es definible y, de serlo, la documentación debe indicar los valores admitidos."

16. **No hay más relojes que el reloj de tiempo real que se proporciona con WebSphere Real Time for RT Linux.**

    "Si una implementación incluye relojes distintos al reloj de tiempo real necesario, su documentación debe indicar en qué contextos se pueden utilizar dichos relojes. "

Nota:

A La arquitectura de referencia para las pruebas será LS20, de 4 vías, 2 GHz con 1 MB de memoria caché y 4 GB de memoria.

B La recogida de basura puede provocar un retardo en cualquier punto de la hebra asociada al almacenamiento dinámico. El recopilador puede operar en una de las dos modalidades básicas controlando el comportamiento cuando se agota la memoria del almacenamiento dinámico. Si el recopilador se define para generar inmediatamente OutOfMemoryError en estas circunstancias, el retardo de la recogida de basura, en el peor de los casos, estará por debajo de 1 ms. Actualmente, en algunas circunstancias, el retardo puede ser mayor; por ejemplo, si hay muchas hebras con pilas profundamente anidadas o una gran cantidad de ámbitos de gran tamaño. Si el recopilador se ha definido para realizar una recogida de basura síncrona antes de generar una excepción OutOfMemoryError, el retardo potencial de la recogida estará relacionado con el número de objetos activos del almacenamiento dinámico y el número de objetos de otras áreas de memoria. En estas circunstancias, se considera que el retardo no tiene límites porque puede ser de muchos segundos para los tamaños habituales del almacenamiento dinámico.

Capítulo 3. Planificación

Lea esta sección antes de instalar WebSphere Real Time for RT Linux.

- "Migración"
-
- "Requisitos de hardware y software"
- "Consideraciones" en la página 28

Migración

WebSphere Real Time for RT Linux se ejecuta en un entorno Linux modificado para aplicaciones en tiempo real. Puede utilizar las aplicaciones Java estándar en un entorno de tiempo real. También puede modificar sus aplicaciones para aprovechar las características de WebSphere Real Time.

Migración del sistema

Siga las instrucciones proporcionadas por el equipo de soporte de Linux.

Requisitos de hardware y software

Utilice esta lista para comprobar el hardware, el sistema operativo y el entorno de Java admitidos para WebSphere Real Time for RT Linux.

Hardware

Las configuraciones de hardware certificado de WebSphere Real Time for RT Linux son variantes de multiprocesador de los sistemas siguientes:

- IBM BladeCenter LS20 (tipos 8850-76U, 8850-55U, 7971, 7972)
- IBM eServer xSeries 326m (Tipos 7969-65U, 7969-85U, 7984-52U, 7984-6AU)
- IBM BladeCenter LS21 (Tipo 7971-6AU)
- IBM BladeCenter HS21 XM Dual Quad Core (Tipo 7995)

Para conservar la certificación para WebSphere Real Time for RT Linux, los sistemas de IBM con Hyper-Threading no deben tener la opción Hyper-Threading habilitada.

Además, WebSphere Real Time for RT Linux se admite en el hardware que ejecuta un sistema operativo soportado, y que cuenta con estas características:

- Un mínimo de 512 MB de memoria física.
- Mínimo Intel Pentium 4, AMD Opteron o Intel Atom Processor.

Para los sistemas que no sean configuraciones certificadas de hardware, IBM no realiza sentencias de rendimiento. Las consideraciones de rendimiento para las configuraciones de hardware certificadas se muestran a continuación: Capítulo 7, "Rendimiento", en la página 103

En los sistemas con soporte para Hyper-Threading, asegúrese de que dicha opción no esté habilitada para evitar efectos negativos en el rendimiento al utilizar WebSphere Real Time for RT Linux.

Sistema operativo

- Red Hat Enterprise Messaging, Realtime, Grid (MRG) 1.3 for Red Hat Enterprise Linux 5 Update 5.
- Red Hat Enterprise MRG 2.2 for Red Hat Enterprise Linux (RHEL) 6 Update 5. Consulte la nota.
- SUSE Linux Enterprise Real Time (SLERT) 10.
- SLERT 11 service pack 1, con nivel de parche 2.6.33.18.

Nota: En condiciones de carga de trabajo alta, se observa un problema intermitente al ejecutar las aplicaciones en IBM WebSphere Real Time V3 for Real Time Linux en MRG 2.2 con RHEL 6 Update 3. Este problema queda resuelto en RHEL 6 Update 5, que es el nivel base recomendado. Para obtener más información, consulte http://www.ibm.com/support/docview.wss?uid=swg21624408.

Consulte el apartado "Instalación de un entorno de Real Time Linux" en la página 31.

Consideraciones

Debe tener en cuenta algunos factores cuando utilice WebSphere Real Time for RT Linux.

- Cuando sea posible, no ejecute más de una JVM en tiempo real en el mismo sistema. La razón es que tendría varios recopiladores de basura. Una JVM no conoce las áreas de memoria de las demás. Un efecto es que los ciclos y tiempos de pausa de GC no se pueden coordinar entre las JVM, lo que significa que es posible que una JVM afecte negativamente al rendimiento de GC de otra JVM. Si tiene que utilizar varias JVM, asegúrese de que cada JVM esté vinculada a un subconjunto específico de procesadores mediante el mandato **taskset**.
- No puede utilizar la opción **-Xdebug** ni la opción **-Xnojit** con el código que se ha precompilado mediante el uso del compilador AOT (Ahead-of-Time). La razón es que **-Xdebug** compila el código de una forma distinta al compilador AOT (Ahead-of-Time) y no se admite.

  Para depurar su código, utilice el código interpretado o compilado por JIT.
- Si está utilizando la interfaz com.sun.tools.javac.Main para compilar el código de origen Java que utiliza el paquete javax.realtime, debe asegurarse de que se haya incluido `sdk/jre/lib/i386/realtime/jclSC170/realtime.jar` en la variable classpath. Un ejemplo habitual de este tipo de compilación es la compilación Ant.
- El paquete JavaComm opcional se puede instalar en WebSphere Real Time for RT Linux, y se puede acceder desde la JVM de tiempo real y no de tiempo real. Para obtener más información sobre la instalación y la configuración, consulte el apartado http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/user/jcommchapter.html. La JVM de tiempo real en WRT admite la API JavaComm para utilizarla con las hebras de Java común. Sin embargo, no se ofrece ninguna garantía con respecto a la determinación o el rendimiento en tiempo real cuando se accede a dispositivos externos con JavaComm. Así pues, no utilice JavaComm con hebras en tiempo real y de tiempo real no de pila, o cuando se requiera un comportamiento de tiempo real.
- Las memorias caché compartidas usadas por releases anteriores de WebSphere Real Time for RT Linux para almacenar clases y código precompilado no son compatibles con las memorias caché utilizadas por este release de WebSphere Real Time for RT Linux. Debe volver a generar el contenido de las memorias caché anteriores.

- Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.
- El mandato **ps** trunca los nombres de hebras de Java.

  El mandato **ps** está limitado a 15 caracteres. Si ha definido un nombre de hebra en más de 15 caracteres, el nombre quedará truncado por el mandato **ps**.
- WebSphere Real Time for RT Linux no admite la autenticación NTLoginModule (NTLM).

  NTLoginModule (NTLM) se utiliza para ayudar a autenticar el acceso a un servicio de Windows. La autenticación mediante el uso de NTLM se admite solo en la plataforma Windows. Esto significa que WebSphere Real Time for RT Linux no admite la autenticación NTLM.

Capítulo 4. Instalación de WebSphere Real Time for RT Linux

Siga estos pasos para instalar el producto.

Archivos de instalación

Necesita estos archivos de instalación

IBM WebSphere Real Time for RT Linux se proporciona en dos tipos de paquetes InstallAnywhere.

Paquetes instalables
Los Paquetes instalables configuran su sistema. Por ejemplo, los programas podrían establecer variables de entorno.

- `wrt-3.0-0.0-rtlinux-x86_32-sdk.bin`
- `wrt-3.0-0.0-rtlinux-x86_32-jre.bin`

Paquetes de archivo
Estos paquetes extraen los archivos en su sistema, pero no realizan configuración alguna.

- `wrt-3.0-0.0-rtlinux-x86_32-sdk.archive.bin`
- `wrt-3.0-0.0-rtlinux-x86_32-jre.archive.bin`

Instalación de un entorno de Real Time Linux

Antes de instalar WebSphere Real Time for RT Linux, debe instalar una versión de 64 bits de Real Time Linux.

Red Hat Enterprise Messaging, Realtime, Grid (MRG 1.3) for RHEL 5.5
Para obtener más información sobre cómo instalar el componente en tiempo real de Red Hat Enterprise Linux 5.5 MRG 1.3, consulte las instrucciones de instalación para RT-Linux RHEL 5.5 MRG 1.3: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/1.3/html/Realtime_Installation_Guide/index.html

Red Hat Enterprise MRG 2.2 for RHEL 6.3
Para obtener más información sobre cómo instalar el componente en tiempo real de Red Hat Enterprise MRG 2.2, consulte las instrucciones de

instalación: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/2/html/Realtime_Installation_Guide/index.html

SUSE Linux Enterprise Real Time (SLERT) 10
Para obtener más información sobre cómo instalar SLERT 10, consulte: http://www.novell.com/products/realtime/eval.html

SUSE Linux Enterprise Real Time (SLERT) 11 service pack 1
Puede obtener SLERT 11 de http://download.novell.com/. Service pack 1 es necesario para trabajar correctamente con WebSphere Real Time for RT Linux. También debe aplicar el nivel de parche 2.6.33.18, que puede descargar desde Real Time Linux Kernel 5075. Para obtener más información sobre SLERT 11, consulte: https://www.suse.com/products/realtime/.

Si utiliza un gran número de descriptores de archivo para cargar diferentes instancias de clases, es posible que vea un mensaje de error "`java.util.zip.ZipException: error al abrir el archivo ZIP`", o alguna otra forma de IOException advirtiéndole de que no se ha podido abrir un archivo. La solución es aumentar el suministro para los descriptores de archivo mediante el mandato **`ulimit`**. Para encontrar el límite actual de los archivos abiertos, utilice el mandato:

```
ulimit -a
```

Para permitir más archivos abiertos, utilice el mandato:

```
ulimit -n 8196
```

Instalación desde un paquete InstallAnywhere

Estos paquetes proporcionan un programa interactivo que le guía durante las opciones de instalación. Puede ejecutar el programa como una interfaz gráfica de usuario o desde una consola del sistema.

Antes de empezar

El sistema debe tener las siguientes bibliotecas compartidas:

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

Si no tiene la biblioteca compartida `libstdc++.so.5`, es posible que se produzca un volcado Java core al realizar la instalación, con los siguientes errores:

```
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
```

Si está instalando un paquete instalable, debe tener la herramienta rpm-build instalada en el sistema, de lo contrario el programa de instalación no podrá registrar el nuevo paquete en la base de datos de RPM. Para averiguar si la herramienta rpm-build está instalada, especifique el mandato siguiente:

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

Acerca de esta tarea

Los paquetes InstallAnywhere tienen una extensión de archivo `.bin`.

Existen dos tipos de paquetes:

Instalable
Al instalar estos paquetes también se configura el sistema, por ejemplo estableciendo las variables de entorno.

De archivo
Al instalar estos paquetes se extraen los archivos del sistema, pero no se realiza ninguna configuración.

Procedimiento

- Para instalar el paquete de una forma interactiva, complete una instalación atendida.
- Para instalar el paquete sin ninguna interacción adicional del usuario, complete una instalación desatendida. Puede seleccionar esta opción si desea instalar muchos sistemas.
- Cuando se completa el proceso de instalación, siga los pasos de configuración de esta sección, como el establecimiento de las variables de entorno path y classpath.

Resultados

El producto se instala.

Nota: No interrumpa el proceso de instalación, por ejemplo pulsando Ctrl+C. Si interrumpe el proceso, puede que tenga que reinstalar el producto. Para obtener más información, consulte "Instalación interrumpida" en la página 35.

Si utiliza un paquete instalable, podrá ver mensajes que le notificarán en caso de encontrar algún problema. La instalación de los paquetes de archivo no genera ningún mensaje. Algunos de los mensajes que podrá ver si utiliza un paquete instalable se muestran en la siguiente lista:

`El instalador no se puede ejecutar en la configuración. Va a abandonar.`
Este mensaje de error se genera cuando ni ID de usuario no está autorizado para ejecutar el proceso de instalación. Como no puede continuar, el programa de instalación finaliza. Para arreglar el problema, inicie de nuevo la instalación pero con un ID de usuario que tenga autorización de usuario root.

`Ya hay un paquete RPM instalado. Desinstale el paquete antes de continuar.`
Este mensaje indica que ya hay un paquete RPM instalado. Como no puede continuar, el programa de instalación finaliza. Para solucionar el problema, desinstale el paquete RPM antes de continuar.

Finalización de una instalación atendida

Instale el producto desde un paquete InstallAnywhere, de forma interactiva.

Antes de empezar

Compruebe las condiciones siguientes antes de empezar el proceso de instalación:

- Si ha instalado con anterioridad WebSphere Real Time for RT Linux desde un paquete RPM, deberá desinstalar este paquete antes de continuar.
- Deberá tener un ID de usuario con autorización de usuario root.

Procedimiento

1. Descargue el archivo del paquete de instalación en un directorio temporal.
2. Vaya al directorio temporal.
3. Inicie el proceso de instalación escribiendo `./package` en un indicador de shell, donde *paquete* es el nombre del paquete que está instalando.
4. Seleccione un idioma de la lista que aparece en la ventana del instalador y, a continuación, pulse **Siguiente**. La lista de idiomas disponibles se basa en el valor de entorno local de su sistema.
5. Lea el acuerdo de licencia, utilice la barra de desplazamiento para leer el texto de licencia hasta el final. Para proseguir con la instalación deberá aceptar los términos del acuerdo de licencia. Para aceptar los términos, seleccione el botón de selección y, a continuación, pulse **Aceptar**.

   Nota: No seleccione el botón de selección para aceptar el acuerdo de licencia hasta que haya leído todo el texto de licencia.
6. Se le solicitará que elija un directorio de destino para la instalación. Si no desea realizar la instalación en el directorio predeterminado, pulse **Elegir** para seleccionar un directorio alternativo utilizando la ventana de navegador. Cuando haya seleccionado el directorio de instalación, pulse **Siguiente** para continuar.
7. Se le solicitará que revise las opciones que hay realizado. Para cambiar su selección, pulse **Anterior**. Si sus opciones son correctas, pulse **Instalar** para continuar con la instalación.
8. Cuando el proceso de instalación haya finalizado, pulse **Hecho** para terminar.

Finalización de una instalación desatendida

Si tiene que instalar más de un sistema y ya sabe las opciones de instalación que desea utilizar, tal vez quiera utilizar un proceso de instalación desatendida. Instale uno mediante el proceso de instalación atendida y, a continuación, utilice el archivo de respuestas resultante para completar las demás instalaciones sin ninguna interacción adicional del usuario.

Procedimiento

1. Cree un archivo de respuestas completando una instalación atendida. Utilice una de las opciones siguientes:
   - Utilice la GUI y especifique que el programa de instalación cree un archivo de respuestas. El archivo de respuestas se denomina `installer.properties` y se crea en el directorio de instalación.
   - Utilice la línea de comandos y añada la opción -r al mandato de instalación atendida, especificando la vía de acceso completa al archivo de respuestas. Por ejemplo:

     `./paquete -r /vía de acceso/installer.properties`

   Contenido del archivo de respuestas de ejemplo:

   ```
   INSTALLER_UI=silent
   USER_INSTALL_DIR=/mi_directorio
   ```

   En este ejemplo, */mi_directorio* es el directorio de instalación de destino que seleccionó para la instalación.

2. Opcional: En caso necesario, edite el archivo de respuestas para cambiar opciones.

   Nota: Los paquetes de archivo presentan el siguiente problema conocido: las instalaciones que utilizan un archivo de respuestas usan el directorio predeterminado incluso aunque cambie el directorio en el archivo de respuestas. Si existe una instalación anterior en el directorio predeterminado, se sobrescribe.

   Si crea más de un archivo de respuestas, cada uno con diferentes opciones de instalación, especifique un nombre único para cada archivo de respuesta, con el formato *myfile*.properties.
3. Opcional: Genere un archivo de registro. Como está instalando en modalidad silenciosa, no se muestra ningún mensaje de estado al final del proceso de instalación. Para generar un archivo de registro que contenga el estado de la estado, complete los pasos siguientes:
   a. Defina las propiedades del sistema necesarias mediante el mandato siguiente.

      ```
      export _JAVA_OPTIONS="-Dlax.debug.level=3 -Dlax.debug.all=true"
      ```
   b. Defina la siguiente variable de entorno para enviar la salida de registro a la consola:

      ```
      export LAX_DEBUG=1
      ```
4. Inicie una instalación desatendida mediante la ejecución del instalador del paquete con la opción silenciosa **-i** y la opción **-f** para especificar el archivo de respuestas. Por ejemplo:

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

   Puede utilizar una vía de acceso completa o relativa para el archivo de propiedades. En estos ejemplos, la cadena `1>console.txt 2>&1` redirecciona la información del proceso de instalación desde las secuencias stderr y stdout al archivo de registro `console.txt` en el directorio actual. Revise este archivo de registro si cree que ha habido un problema con la instalación.

   Nota: Si el directorio de instalación contiene varios archivos de respuestas, se utiliza el archivo de respuestas predeterminado, `installer.properties`.

Instalación interrumpida

Si el instalador de paquete se detiene inesperadamente durante la instalación, por ejemplo si pulsa Ctrl+C, la instalación será errónea y no podrá desinstalar o reinstalar el producto. Si intenta desinstalar o volver a instalar es posible que vea el mensaje `Error de aplicación fatal`.

Acerca de esta tarea

Para resolver este problema, suprima los archivos y vuelva a instalarlos, tal como se describe en los pasos siguientes.

Procedimiento

1. Suprima el archivo de registro `/var/.com.zerog.registry.xml`.
2. Suprima el directorio que contiene la instalación, en el caso de que se haya creado. Por ejemplo, opt/IBM/javawrt3/.
3. Ejecute el programa de instalación de nuevo.

Problemas y limitaciones conocidos

Los paquetes InstallAnywhere tienen algunos problemas y limitaciones conocidas.

- Si no tiene la biblioteca compartida `libstdc++.so.5` en su sistema, la instalación fallará, produciendo un volcado Java core. Para obtener más información, consulte "Instalación desde un paquete InstallAnywhere" en la página 32.
- La GUI de paquete de instalación no da soporte al programa de lector de pantalla Orca. Puede utilizar la instalación desatendida como una alternativa a la GUI.
- Si, tras la instalación, especifica `./package` para iniciar de nuevo el programa, el programa muestra el siguiente mensaje:

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

  Si pulsa la tecla Intro para aceptar el valor predeterminado, el programa no responde. Escriba un número y, a continuación, pulse Intro.
- Si instala el paquete; a continuación, instálelo de nuevo en una modalidad diferente, por ejemplo, consola o silencio, es posible que vea el mensaje de error siguiente:

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

  No debería ver este mensaje si ha realizado la instalación con la modalidad GUI y está ejecutando el programa de instalación de nuevo en la modalidad de consola. Si ve este error y está ejecutando el programa para seleccionar la opción de desinstalación (solo paquetes instalables), utilice en su lugar el mandato **`./_uninstall/uninstall`**, tal como se describe en "Desinstalación de WebSphere Real Time for RT Linux" en la página 39.

Sólo paquetes instalables

- No puede actualizar una instalación existente mediante los paquetes InstallAnywhere. Para actualizar WebSphere Real Time for RT Linux, primero debe desinstalar cualquier versión anterior.
- No puede instalar dos instancias distintas de la misma versión de WebSphere Real Time for RT Linux en el mismo sistema y al mismo tiempo, incluso si utiliza directorios de instalación diferentes. Por ejemplo, no puede tener simultáneamente WebSphere Real Time for RT Linux V3 en el directorio `/previous`, y una instalación actualizada del servicio WebSphere Real Time for RT Linux en el directorio `/current`. El programa de instalación comprueba el número de versión. Si el programa encuentra un paquete existente con el mismo número de versión, se le solicita que desinstale el paquete existente.
- Si el paquete está instalado y ejecuta el instalador del paquete de nuevo mediante la GUI, puede seleccionar desinstalar el paquete. Esta opción de desinstalación no está disponible en la modalidad desatendida. Si ejecuta el instalador del paquete en modalidad desatendida, el programa se ejecuta pero no realiza ninguna acción.

Sólo paquetes de archivo

- Si cambia el directorio de instalación en un archivo de respuestas y, a continuación, ejecuta una instalación desatendida mediante ese archivo de respuesta, el programa de instalación ignora el nuevo directorio de instalación y utiliza el directorio predeterminado. Si existe una instalación anterior en el directorio predeterminado, se sobrescribe.

Configuración de la vía de acceso

Cuando haya definido la variable de entorno **PATH**, podrá ejecutar una aplicación o un programa escribiendo su nombre en un indicador de shell.

Acerca de esta tarea

Nota: Si altera la variable de entorno **PATH** como se ha descrito en esta sección, alterará temporalmente los archivos Java ejecutables existentes en su vía de acceso.

Puede especificar la vía de acceso a una herramienta escribiendo siempre la vía de acceso antes del nombre de la herramienta. Por ejemplo, si el SDK está instalado en `opt/IBM/javawrt3/`, puede compilar un archivo denominado *myfile.java* escribiendo el mandato siguiente en el indicador de shell:

```
opt/IBM/javawrt3/bin/javac myfile.java
```

Para no tener que escribir cada vez la vía de acceso completa:

1. Edite el archivo de arranque de shell en el directorio local (normalmente **.bashrc**, dependiendo del shell) y añada las vías de acceso absolutas a la variable de entorno **PATH**; por ejemplo:

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

2. Vuelva a iniciar una sesión o ejecute el script de shell actualizado para activar el nuevo valor de **PATH**.
3. Compile el archivo con la herramienta **javac**. Por ejemplo, para compilar el archivo *myfile.java*, en un indicador de shell, escriba:

   ```
   javac -Xrealtime myfile.java
   ```

   La variable de entorno **PATH** permite que Linux busque archivos ejecutables, como, por ejemplo, **javac**, **java** y la herramienta **javadoc**, desde cualquier directorio actual. Para visualizar el valor actual de la vía de acceso, escriba el mandato siguiente en un indicador de mandatos:

   ```
   echo $PATH
   ```

Qué hacer a continuación

Consulte el apartado "Establecimiento de classpath" para determinar si necesita definir su variable de entorno CLASSPATH.

Establecimiento de classpath

La variable de entorno **CLASSPATH** indica las herramientas de del SDK, como **java**, **javac** y **javadoc**, donde se encuentran las bibliotecas de clases Java.

Acerca de esta tarea

Defina la variable de entorno **CLASSPATH** explícitamente solo si se da una de estas condiciones:

- Si necesita un archivo de clase o de biblioteca diferente, por ejemplo, uno que haya desarrollado previamente, y no está en el directorio activo.
- Si cambia la ubicación de los directorios `bin` y `lib` de forma que ya no tienen el mismo directorio padre.
- Si tiene previsto desarrollar o ejecutar aplicaciones que utilicen entornos de ejecución diferentes en el mismo sistema.

Para visualizar el valor actual de **CLASSPATH**, escriba el mandato siguiente en un indicador del intérprete de mandatos:

```
echo $CLASSPATH
```

Si desarrolla y ejecuta aplicaciones que utilicen entornos de ejecución diferentes, incluyendo otras versiones que haya instalado por separado, debe establecer **CLASSPATH** y **PATH** de forma explícita para cada aplicación. Si ejecuta varias aplicaciones simultáneamente y utiliza entornos de ejecución diferentes, cada aplicación debe ejecutarse en su propio shell.

Si solo ejecuta una versión de Java en cada momento, puede utilizar un script de shell para pasar de un entorno de tiempo de ejecución a otro.

Qué hacer a continuación

Consulte el apartado "Prueba de la instalación" para verificar que ha realizado su instalación correctamente.

Prueba de la instalación

Utilice la opción **-version** para comprobar si su instalación es correcta.

Acerca de esta tarea

La instalación de Java consta de una JVM estándar y una JVM en tiempo real.

Procedimiento

Pruebe su instalación siguiendo estos pasos:

1. Para ver la información de la versión para la JVM estándar, escriba el mandato siguiente en un indicador de shell:

   ```
   java -version
   ```

   Este mandato devuelve los mensajes siguientes si es correcto:

   ```
   java versión "1.7.0"
   WebSphere Real Time V3 (build pxi3270rt-20110518_02)
   IBM J9 VM (build 2.6, JRE 1.7.0 Linux x86-32 20110516_82445 (JIT enabled,
   AOT enabled)
   J9VM - R26_head_20110515_0456_B82363
   JIT  - r11_20110510_19526
   GC   - R26_head_20110513_1009_B82250
   J9CL - 20110516_82445)
   JCL - 20110516_01 based on Oracle 7b145
   ```

   Si va a utilizar la JVM estándar y no la JVM en tiempo real, consulte la publicación IBM User Guides for Java v7 on Linux.

   Nota: La información de la versión es correcta, pero es posible que las fechas sean posteriores a las de este ejemplo. El formato de la cadena de fecha es el siguiente: `yyyymmdd`, posiblemente seguido de información adicional específica del componente.
2. Para ver la información de la versión para la JVM en tiempo real, especifique el mandato siguiente en un indicador de shell:

   ```
   java -Xrealtime -version
   ```

   Este mandato devuelve los mensajes siguientes si es correcto:

```
java versión "1.7.0"
WebSphere Real Time V3 (build pxi3270rt-20110518_02)
IBM J9 VM (build 2.6, JRE 1.7.0 real-time Linux x86-32 20110516_82445 (JIT
enabled, AOT enabled)
J9VM - R26_head_20110515_0456_B82363
JIT  - r11_20110510_19526
GC   - R26_head_20110513_1009_B82250
J9CL - 20110516_82445)
JCL - 20110516_01 based on Oracle 7b145
```

Nota: La información de la versión es correcta pero las fechas y arquitectura de la plataforma podrían ser distintas del ejemplo. El formato de la cadena de fecha es el siguiente: `yyyymmdd`, posiblemente seguido de información adicional específica del componente.

Desinstalación de WebSphere Real Time for RT Linux

El proceso que utiliza para eliminar WebSphere Real Time for RT Linux depende del tipo de instalación que haya empleado.

Antes de empezar

Para los paquetes instalables InstallAnywhere, deberá tener un ID de usuario con autorización de usuario root.

Acerca de esta tarea

No existe ningún proceso de desinstalación para los paquetes de archivo InstallAnywhere. Para eliminar un paquete de archivo desde el sistema, suprima el directorio de destino que seleccionó al instalar el paquete. Para los paquetes instalables InstallAnywhere, desinstale el producto mediante un mandato o bien ejecutando de nuevo el programa de instalación, como se describe en los pasos siguientes.

Procedimiento

- Opcional: Desinstálelo manualmente mediante el mandato **uninstall**.
  1. Acceda al directorio que contiene la instalación. Por ejemplo:
     ```
     cd /opt/IBM/javawrt3
     ```
  2. Inicie el proceso de desinstalación mediante el mandato siguiente:
     ```
     ./_uninstall/uninstall
     ```
- Opcional: Si no puede ubicar fácilmente el programa de desinstalación, puede ejecutar otra instalación atendida como alternativa. El programa de instalación detecta que el producto ya está instalado y, a continuación, le ofrece la oportunidad de desinstalar la instalación anterior.

Capítulo 5. Ejecución de aplicaciones de IBM WebSphere Real Time for RT Linux

Información importante para ayudarle en la ejecución de aplicaciones en tiempo real.

Asignación y planificación de hebras

El sistema operativo Linux admite varias políticas de planificación. La política de planificación de compartición de tiempo universal es SCHED_OTHER, que utilizan la mayoría de las hebras. Las hebras pueden utilizar SCHED_RR y SCHED_FIFO en aplicaciones en tiempo real. .

El kernel decide cuál es la siguiente hebra ejecutable que puede ejecutar el procesador. El kernel mantiene una lista de hebras ejecutables. Busca la hebra con la prioridad más alta y selecciona dicha hebra como la hebra que se va a ejecutar a continuación.

Las políticas y prioridades de hebras se pueden listar utilizando el mandato siguiente:

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

donde la política:

- `TS` es SCHED_OTHER
- `RR` es SCHED_RR
- `FF` es SCHED_FIFO
- `-` no tiene ninguna política notificada

La salida es similar a la de este ejemplo:

```
  PID  PPID POL   TID COMMAND         RTPRIO     TIME
18314 30285 -       - java                 - 00:01:40
    -     - RR  18314 -                    6 00:00:00
    -     - RR  18315 -                    6 00:01:40
    -     - FF  18318 -                   88 00:00:00
    -     - RR  18323 -                    6 00:00:00
    -     - FF  18324 -                   13 00:00:00
    -     - RR  18325 -                    6 00:00:00
    -     - RR  18326 -                    6 00:00:00
    -     - FF  18327 -                   11 00:00:00
    -     - FF  18328 -                   89 00:00:00
```

Esta salida muestra el proceso Java, la política de planificación que se está aplicando, la hebra principal con la prioridad "-" (otra) y hebras de tiempo real con prioridades entre 11 y 89.

Para consultar la política de planificación actual, utilice **sched_getscheduler**, o bien el mandato **ps** que se muestra en el ejemplo.

Para obtener más información sobre los procesos, consulte el apartado "Técnicas de depuración generales" en la página 108.

Políticas y prioridades de las hebras de Java común

Las hebras de Java común, es decir, las hebras asignadas como objetos `java.lang.Thread`, utilizan la política de programación predeterminada de SCHED_OTHER. Desde WebSphere Real Time for RT Linux V3 renovación de servicio 1, puede ejecutar las hebras regulares de Java con la política de programación de SCHED_RR o SCHED_FIFO.

De forma predeterminada, las hebras de Java se ejecutan utilizando la política SCHED_OTHER predeterminada. Esta política asigna las hebras de Java con la prioridad 0 del sistema operativo.

La utilización de la política SCHED_RR o SCHED_FIFO le ofrece un mayor control sobre su aplicación, lo que puede mejorar el rendimiento en tiempo real de las hebras de Java. La máquina virtual Java detecta la prioridad y la política de la hebra principal cuando Java se inicia con la política SCHED_RR o SCHED_FIFO. La JVM altera las correlaciones de política y prioridad en consecuencia. Todas las hebras de Java se ejecutan con la misma prioridad de sistema operativo que la hebra principal. Aunque a SCHED_RR o SCHED_FIFO se les pueden asignar las prioridades 1 - 99, las prioridades de SCHED_RR o SCHED_FIFO que se pueden utilizar para WebSphere Real Time for RT Linux V3 son las prioridades 1-10. Si la prioridad se define más alta que 10, la prioridad de la hebra principal baja a 10 y la correlación aplicada se basa en un valor de 10.

Una manera de modificar la propiedad de programación en tiempo real de un proceso en la línea de mandatos es utilizar el mandato **chrt**. En el ejemplo siguiente, la prioridad de la hebra principal de Java se ha definido para utilizar la política de programación SCHED_FIFO, con una prioridad de sistema operativo de 6.

```
chrt -f 6 java
```

Es posible que necesite configurar su sistema para permitir que se cambien las prioridades. Consulte el apartado "Configuración del sistema para permitir los cambios de prioridades" en la página 43 para obtener más información.

Tabla 4. Prioridades del sistema operativo y Java

Prioridad de Java	Valor de prioridad de Java para la hebra	Prioridad del sistema operativo
1	MIN_PRIORITY	6
2		6
3		6
4		6
5	NORM_PRIORITY (valor predeterminado)	6
6		6
7		6
8		6

Tabla 4. Prioridades del sistema operativo y Java (continuación)

Prioridad de Java	Valor de prioridad de Java para la hebra	Prioridad del sistema operativo
9		6
10	MAX_PRIORITY	6

Todas las hebras asociadas a la hebra principal de Java se ejecutan con la misma prioridad de sistema operativo. La hebra de alarma de Metronome, y otras hebras internas de la JVM con requisitos concretos de temporización en tiempo real, se pueden ejecutar con una prioridad más alta que las prioridades utilizadas por las hebras de Java común.

Si ejecuta el mandato `chrt -f 11 java`, el resultado es el mismo que si ejecuta `chrt -f 10 java`. Esto se debe a que no puede aplicar una prioridad por encima de 10 a la correlación de prioridades utilizada por las hebras de la JVM, aunque la hebra que inicia la JVM y espera la finalización de la JVM se mantenga con la prioridad 11.

Para obtener más información sobre el mandato **chrt**, consulte http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaai/realtime/liaairtchrt.htm.

Configuración del sistema para permitir los cambios de prioridades

De forma predeterminada, los usuarios no root en Linux no pueden elevar la prioridad de una hebra o un proceso. Puede cambiar la configuración del sistema para permitir los cambios de prioridades usando el módulo `pam_limits` de PAM (Pluggable Authentication Modules) para Linux.

Si no puede cambiar la prioridad de una hebra o un proceso usando el programa de utilidad **chrt**, verá este mensaje:

```
sched_setscheduler: Operation not permitted
```

En kernels Linux más recientes, puede cambiar la configuración del sistema para permitir cambios de prioridades usando el módulo `pam_limits`. Este módulo le permite configurar los límites de los recursos del sistema en el archivo de configuración de límites. El archivo predeterminado es `/etc/security/limits.conf`.

Una entrada en el archivo `/etc/security/limits.conf` tiene el formato siguiente:

```
<dominio> <tipo> <elemento> <valor>
```

donde:

<dominio> puede ser:

- un nombre de usuario del sistema que puede modificar los límites de un recurso.

- un nombre de grupo, con la sintaxis `@group`, cuyos miembros pueden modificar los límites de un recurso.

- un comodín "`*`", que indica que cualquier usuario o grupo puede modificar los límites de un recurso.

<tipo> puede ser:

- `hard`: el kernel impone límites estrictos.

- `soft`: se aplican límites intermedios, que se pueden modificar dentro del rango indicado por los límites estrictos.

- un guión "**-**": indica los límites estrictos e intermedios.

<elemento> es:
- un recurso. Utilice `rtprio` para las prioridades en tiempo real.

<valor> es:
- un límite. Utilice un valor del rango 1 - 100 para indicar el límite máximo para el valor de la prioridad en tiempo real.

Por ejemplo,

```
* - rtprio 100
```

permite a todos los usuarios cambiar la prioridad de los procesos en tiempo real, utilizando **chrt** u otros mecanismos.

De forma predeterminada, el usuario root puede aumentar las prioridades en tiempo real sin límites. Para aplicar un límite a una raíz, el usuario root se debe haber especificado explícitamente. Los límites de grupos y comodines del archivo de configuración no se aplican al usuario root.

Si especifica límites de usuario individuales en el archivo, estos límites tendrán prioridad sobre los límites de grupos.

Los cambios realizados en `limits.conf` no entran en vigor de inmediato. Debe reiniciar los servicios afectados o reiniciar el sistema para que el cambio realizado en la configuración entre en vigor.

Para habilitar los cambios de prioridad en un sistema Linux en tiempo real, puede añadir un usuario al grupo `realtime`, que aparece en el archivo `limits.conf`.

Inicio de procesos secundarios

Los métodos java.lang.Runtime.exec de la API de la máquina virtual Java dan a su aplicación Java la capacidad de ejecutar un mandato en un proceso independiente.

Desde dicha llamada a método, se crea un nuevo objeto java.lang.Process. El objeto se puede usar para controlar el nuevo proceso o para obtener información acerca de él.

Los métodos exec crean varias hebras con esta finalidad. En IBM WebSphere Real Time for RT Linux, las modificaciones del procedimiento habilitan un comportamiento más determinante en un entorno de tiempo real.

La llamada Runtime.exec crea una hebra “recolectora” para cada subproceso bifurcado. La hebra recolectora es la única que espera a que el subproceso termine. Cuando el subproceso termina, la hebra recolectora registra el estado de salida del subproceso. La hebra recolectora genera el nuevo proceso y le da los mismos parámetros de planificación de que a la hebra que llamó en un principio a Runtime.exec.

Si el proceso generado es otra máquina virtual Java de WebSphere Real Time for RT Linux y el método Runtime.exec ha sido llamado por otro método que ejecuta una prioridad y una política en tiempo real de Linux, la hebra principal de la nueva máquina virtual correlaciona su política y su prioridad con la misma política y prioridad en tiempo real de Linux. Esta prioridad de hebra Java está entre 1 y 10.

La hebra recolectora también crea dos nuevas hebras que escuchan las secuencias `stdout` y `stderr` del nuevo proceso. Estas nuevas hebras son hebras Java regulares,

no hebras Java en tiempo real. Los datos `stdout` y `stderr` se guardan en almacenamientos intermedios utilizados por estas hebras. Los almacenamientos intermedios persisten más allá del tiempo de vida del proceso generado. Esta persistencia permite que los recursos que mantiene el proceso generado se borren de inmediato cuando el proceso termina.

Si desea que las hebras lectoras `stdout` y `stderr` se ejecuten en una prioridad en tiempo real de Linux, inicie la máquina virtual Java original propietaria de estas hebras con una política y una prioridad SCHED_FIFO o SCHED_RR de Linux. El efecto es que todas las hebras comunes se correlacionan en una prioridad y una política en tiempo real de, como máximo, 10 en el sistema operativo Linux.

Políticas y prioridades de las hebras Java Real Time

Las hebras de Real Time, es decir, las hebras asignadas como `java.realtime.RealtimeThread`, y los manejadores de sucesos asíncronos utilizan la política de planificación SCHED_FIFO.

La asignación y planificación de hebras de Java de tiempo real forma parte de Real Time Specification for Java (RTSJ). Este tema, incluidas las políticas de planificación y el manejo de prioridades de hebras Java de tiempo real, se analiza en la sección "Soporte para RTSJ" en la página 12.

Utilización del código compilado con WebSphere Real Time for RT Linux

IBM WebSphere Real Time for RT Linux admite varios modelos de compilación de código, y proporciona varios niveles de determinación y rendimiento de código.

Operación interpretada

Es el modelo de compilación de código más sencillo. El intérprete ejecuta una aplicación Java, pero no utiliza la compilación de códigos para nada. El intérprete muestra una buena determinación, pero proporciona un rendimiento muy bajo, así que es mejor evitar el uso de esta modalidad de operación para los sistemas de producción.

Para utilizar la operación interpretada, especifique la opción **`-Xint`** en la línea de comandos Java.

Compilación Just-In-Time (JIT) de prioridad baja

El modelo de compilación predeterminado en WebSphere Real Time for RT Linux utiliza un compilador Just-In-Time para compilar los métodos importantes de una aplicación Java mientras se ejecuta la aplicación. En esta modalidad, el compilador JIT funciona de manera parecida a como funciona cuando se encuentra en una JVM no de tiempo real. La diferencia es que el compilador JIT de WebSphere Real Time for RT Linux se ejecuta con un nivel de prioridad más bajo que cualquier hebra de tiempo real. La prioridad inferior implica que el compilador JIT utiliza recursos del sistema cuando la aplicación no necesita realizar tareas en tiempo real. El efecto es que el compilador JIT no afecta significativamente al rendimiento de las tareas en tiempo real.

El compilador JIT utiliza dos hebras para las actividades relacionadas con la compilación: la hebra de compilación y la hebra de muestreo. Estas hebras se ejecutan con una prioridad más baja que las tareas en tiempo real. La hebra de compilación se ejecuta de manera asíncrona en la aplicación. Esto significa que una hebra de aplicación no espera a que la hebra de compilación termine de compilar un método en ningún momento.

La hebra de muestreo envía periódicamente un mensaje asíncrono a las hebras de aplicación para identificar el método que se está ejecutando actualmente en cada hebra. El procesamiento del mensaje lleva muy poco tiempo en la hebra de aplicación. No se envían mensajes si la hebra de muestreo no se puede ejecutar debido a las tareas en tiempo real de prioridad más alta. La utilización del compilador JIT tiene muy poco efecto sobre la determinación, pero este modo de compilación ofrece el mejor rendimiento a muchos usuarios.

Para ejecutar una aplicación con JIT en una prioridad más baja, consulte el apartado "Habilitación del compilador JIT" en la página 63.

Código precompilado AOT (Ahead-Of-Time)
WebSphere Real Time for RT Linux compila los métodos Java en código nativo, en un paso de precompilación anterior a la ejecución de la aplicación. Antes de WebSphere Real Time for RT Linux V2, el paso se precompilación ha utilizado la herramienta jxeinajar para compilar métodos utilizando un compilador AOT (Ahead-Of-Time) y ha almacenado los resultados en archivos ejecutables Java especiales. Estos archivos se pueden recopilar en archivos .jar vinculados. Al ejecutar una aplicación, los archivos .jar vinculados se añaden a la classpath de la aplicación, de manera que la JVM pueda cargar el código AOT cuando se carguen las clases de los métodos desde JXE. Con esta metodología, el compilador JIT pasa a estar totalmente no disponible, especificando la opción **-Xnojit** en la línea de mandatos. La aplicación puede utilizar el código AOT precompilado que se haya creado, y el intérprete para otros métodos. Esta modalidad de operación ofrece una gran determinación, porque el compilador JIT no está presente, de manera que no hay hebras de muestreo ni reducciones de rendimiento asociadas al conmutador de contexto. La dificultad de compilar el código Java de manera anticipada, cumpliendo la especificación Java, significa que el código compilado AOT suele tener un rendimiento más bajo que el código compilado JIT, aunque suele ser mucho más rápido que la interpretación.

WebSphere Real Time for RT Linux V2 almacena y las versiones posteriores almacenan el código AOT en una memoria caché de clase compartida, en lugar de en archivos JXE, utilizando la tecnología de memorias caché de clase compartida proporcionada en las JVM de IBM Java 6. La herramienta admincache le permite consultar el contenido de una memoria caché, listar todas las memorias caché existentes y llenar una memoria caché con clases y código AOT. Las ventajas de almacenar el código compilado AOT son que los archivos .jar de la aplicación no se modifican y no se necesitan cambios en la classpath cuando se ejecuta la aplicación.

Una memoria caché de clase compartida tiene un límite de tamaño práctico, basado en el espacio de direcciones virtuales disponible. Esto significa que la compilación AOT para todos los archivos .jar no resulta práctica. Se debe realizar una compilación AOT selectiva.

Cuando una aplicación se ejecuta con código AOT en una memoria caché de clase compartida, el código AOT de los métodos de una clase se carga automáticamente cuando la clase se carga en la JVM. El coste adicional al cargar una clase para instalar código AOT con sus métodos hace que resulte importante cargar previamente todas las clases que sea posible antes de que se ejecuten los componentes de la aplicación más importantes para el rendimiento.

La utilización del código AOT precompilado proporciona el nivel más alto de determinación, con un buen rendimiento. El código AOT se puede

utilizar cuando se ejecuta la aplicación especificando las opciones **-Xshareclasses** y **-Xaot**. La opción **-Xaot** está activada de forma predeterminada.

Para almacenar y utilizar el código AOT con una memoria caché de clase compartida utilizando la herramienta admincache, consulte el apartado "Utilización de la herramienta admincache" en la página 48. La información sobre la migración desde jxeinajar hasta admincache se puede hallar en la Documentación de WebSphere Real Time for RT Linux V2.

Para ver un ejemplo sobre cómo ejecutar una aplicación con código AOT compilado, consulte el apartado "Ejecución de la aplicación de muestra mientras se usa AOT" en la página 96.

Modo mixto, que combina el código precompilado AOT y la compilación JIT de prioridad baja

Los códigos compilados AOT y JIT se pueden utilizar juntos mientras se ejecuta la aplicación. Esta modalidad de operación puede proporcionar una buena determinación y un buen rendimiento, además de un rendimiento muy alto para aquellos métodos que se ejecutan con frecuencia. La ventaja principal de esta modalidad es que la precompilación AOT se utiliza para garantizar que los componentes más importantes de la aplicación no se ejecuten en el intérprete, que suele ser bastante más lento que el código compilado AOT o JIT. No tiene que precompilar todos los métodos, porque el compilador JIT puede identificar de forma dinámica todos los métodos interpretados que se ejecutan con frecuencia, sin alterar de manera significativa el rendimiento de la aplicación. El modo mixto es el modo predeterminado cuando la opción **-Xshareclasses** se añade a la línea de mandatos.

Para ejecutar su aplicación con compilación AOT y JIT mixta, consulte el apartado "Ejecución de la aplicación de muestra mientras se usa AOT" en la página 96

Gestión explícita de la compilación

En la modalidad de compilación con el compilador JIT habilitado, la API java.lang.Compiler se puede utilizar para controlar de forma explícita la operación del compilador JIT. El compilador JIT compila los métodos de la clase que se han pasado utilizando el método compileClass(). compileClass() es un método síncrono, por lo que no genera un retorno hasta que no se han compilado los métodos proporcionados. Una aplicación puede utilizar compileClass() en una fase de inicialización, iterando sobre las clases utilizadas por la fase principal del tiempo de ejecución de la aplicación. Cuando finalice la fase de inicialización, llame al método Compiler.disable() para inhabilitar la compilación y las hebras de muestreo en su totalidad. La principal dificultad con esta técnica radica en el problema de gestionar la lista de clases que cargar y compilar en la fase de inicialización de la aplicación, especialmente durante el desarrollo de la aplicación.

Para obtener más información sobre cómo gestionar la compilación en una aplicación, consulte la publicación IBM Real-Time Class Analysis Tool for Java.

Visión general de las opciones de la línea de comandos de compilación

Puede ejecutar una aplicación con JIT habilitado utilizando la opción **-Xjit**, o bien sin JIT con la opción **-Xnojit**. **-Xjit** es la modalidad predeterminada.

Puede ejecutar una aplicación con el código AOT habilitado utilizando las opciones **-Xshareclasses -Xaot**. Inhabilite el código AOT utilizando la opción **-Xnoaot**. **-Xaot** es la opción predeterminada, pero no tiene ningún efecto a menos que se haya especificado también la opción **-Xshareclasses**, porque el código AOT debe estar almacenado en una memoria caché de clase compartida.

Utilización del compilador AOT

Utilice estos pasos para precompilar su código Java. Este procedimiento describe el uso de la opción **-Xrealtime** en un mandato **javac**, la herramienta **admincache** y las opciones **-Xrealtime** y **-Xnojit** con el mandato **java**.

Acerca de esta tarea

Utilizar el compilador AOT (ahead-of-time) implica que la compilación se separa del tiempo de ejecución de la aplicación. Del mismo modo, puede compilar más métodos al mismo tiempo, en lugar de solo los métodos que se utilizan con más frecuencia. Puede compilar toda una aplicación o solo clases individuales, como se muestra en los pasos siguientes.

Nota: Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.

Procedimiento

1. Desde un indicador de shell, especifique:

   ```
   javac -Xrealtime origen
   ```

   Este mandato crea el código de bytes Java desde su origen para utilizarlo en el entorno en tiempo real. Consulte el apartado Figura 2 en la página 11.
2. Empaquete los archivos de clase generados en un archivo JAR. Por ejemplo, para crear el archivo test.jar:

   ```
   jar cvf test.jar source
   ```
3. Desde un indicador de shell, especifique:

   ```
   admincache -Xrealtime -populate -aot test.jar -cacheName myCache -cp test.jar
   ```

   Este mandato precompila el archivo test.jar y escribe la salida en el directorio de salida `./aot`.
4. Para ejecutar el archivo utilizando el código AOT en la memoria caché de clase compartida, en un indicador de shell, especifique:

   ```
   java -Xrealtime -Xshareclasses:name=myCache -cp test.jar -Xnojit MyTestClass
   ```

   Para ejecutar el archivo utilizado el código AOT en la memoria caché de clase compartida, vuelva a compilar los métodos que haya llamado frecuentemente. A continuación, ejecute el mandato siguiente sin crear un nuevo archivo JAR:

   ```
   java -Xrealtime -Xshareclasses:name=myCache -cp test.jar MyTestClass
   ```

   Estos mandatos utilizan los mismos archivos JAR precompilados en el paso 3.

Utilización de la herramienta admincache

La herramienta admincache se utiliza para gestionar las memorias caché de clase compartida en una estación de trabajo

En el producto IBM WebSphere Real Time for RT Linux, la herramienta admincache se puede utilizar para crear una memoria caché de clase compartida

que contenga clases o clases y código compilado AOT. Una vez creadas las memorias caché, esta herramienta también se puede usar para inspeccionar las memorias caché existentes.

La memoria caché de clase compartida se usa para reducir la ocupación de memoria en casos de varias JVM, así como para acelerar el inicio de la aplicación.

WebSphere Real Time for RT Linux puede utilizar memorias caché de clase compartida en las modalidades de tiempo real o no de tiempo real, pero las técnicas de llenado, creación y formato de la memoria caché difieren. Las memorias caché en modalidad de tiempo real no son compatibles con las memorias caché en modalidad no de tiempo real. En modalidad no de tiempo real, las memorias caché se crean y se llenan de la misma manera que una JVM estándar. Esto significa que la JVM llena y crea la memoria caché a medida que se ejecuta una aplicación, de forma transparente para el usuario. En modalidad de tiempo real, utilizando la opción **-Xrealtime**, las memorias caché de clase compartida se deben crear y llenar previamente con admincache, mediante la opción **-populate**. Las aplicaciones que se ejecutan en modalidad de tiempo real pueden leer contenido de la memoria caché rellenada previamente, pero no pueden modificar su contenido.

Las memorias caché de clase compartida creadas en modalidad de tiempo real solo se pueden utilizar cuando se ejecuta una aplicación en modalidad de tiempo real. Las memorias caché de clase compartida creadas en modalidad no de tiempo real solo se pueden utilizar cuando se ejecuta una aplicación en una modalidad no de tiempo real. Esto se aplica también a la herramienta admincache. Para gestionar las memorias caché creadas por la JVM en modalidad de tiempo real, utilice admincache con la opción **-Xrealtime**. Para gestionar las memorias caché creadas por la JVM en modalidad no de tiempo real, no utilice la opción**-Xrealtime**. Para conectarse a una memoria caché de clase compartida durante el tiempo de ejecución, añada la opción **-Xshareclasses** a la línea de mandatos.

En una estación de trabajo, se pueden crear varias memorias caché de clase compartida, cada una de ellas con un nombre específico y en un directorio específico. Cuando se cree una nueva memoria caché, el nombre de la memoria caché se podrá especificar con la opción **-cacheName** <*nombre*>. El nombre de la memoria caché no debe superar los 53 caracteres.

De forma predeterminada, las memorias caché de clase compartida se crean en el directorio `/tmp/javasharedresources`, pero esta ubicación puede ser temporalmente alterada especificando la opción **-cacheDir** <*directorio*>. El formato interno de una memoria caché de clase compartida depende de las características de la estación de trabajo donde se haya creado. Esto significa que las memorias caché de clase compartida no se pueden crear en unidades de red, como medida de seguridad. Otro motivo para esta restricción radica en los imprevisibles efectos de un rendimiento más lento al acceder a una memoria caché de clase compartida desde un sistema de archivos de red.

Si no se especifica ningún nombre de memoria caché en la línea de mandatos, el valor predeterminado será **sharedcc_<user_login>**

Para obtener más información sobre la operación de memoria caché compartida en la modalidad no de tiempo real, consulte el apartado “Datos de clases compartidos entre las JVM para tiempo no real” en la página 103.

Nota: Desde IBM WebSphere Real Time for RT Linux V2 SR1 y posteriores, debe utilizar la opción **-classpath** con la opción **-populate**.

Creación de una memoria caché de clase compartida en tiempo real:

La herramienta admincache se usa para crear memorias caché de clase compartida accesibles en modalidad de tiempo real.

Nota: Debe tener en cuenta las consideraciones de seguridad cuando cree archivos de memoria caché de clase compartida con valores predeterminados. Consulte el apartado "Consideraciones de seguridad para la memoria caché de clase compartida" en la página 105 para obtener más información sobre las consideraciones de seguridad para la memoria caché de clase compartida e información sobre cómo cambiar los permisos predeterminados.

La opción **-populate** de la herramienta admincache se utiliza para crear memorias caché de clase compartida. La opción se utiliza en combinación con una lista de archivos .jar, o un directorio o árbol de directorios, donde buscar archivos .jar. Para cada archivo .jar especificado o encontrado, admincache almacena cada clase del archivo .jar en la memoria caché de clase compartida. Los métodos de clase también se compilan con AOT y se almacenan en la memoria caché de clase compartida, a menos que especifique la opción **-noaot**.

Debe utilizar la opción **-classpath** con **-populate** o verá el siguiente mensaje de error:

```
-populate action requires -classpath <class path> option to be specified
```

La opción **-help** de admincache muestra las subopciones que se pueden usar para controlar cómo admincache llena la memoria caché.

```
$ admincache -Xrealtime -help
Usage: admincache [option]*
  where [option] can be:
     -help | -?                Action: show this help
     -Xrealtime                use in real time environment
     -cacheName <name>         specify name of shared cache (Use %%u to substitute username)
     -cacheDir <dir>           set the location of the JVM cache files
     -listAllCaches            Action: list all existing shared class caches
     -printStats               Action: print cache statistics
     -printAllStats            Action: print more verbose cache statistics
     -destroy                  Action: destroy the named (or default) cache
     -destroyAll               Action: destroy all caches
     -populate                 Action: Create a new cache and populate it
       -searchPath <path>      specify the directory in which to find files if no files specified
                               (default is .)
                               only one -searchPath option can be specified
            -classpath <class path> specify the classpath that will be used at runtime to access this cache
                               the -classpath option is required
       -[no]recurse            [do not] recurse into subdirectories to find files to convert
                               (default do not recurse)
       -[no]grow               if specified cache exists already, [do not] add to it (default no grow)
                               if -grow is not selected, specified cache will be removed if present
       -verbose                print out progress messages for each jar
       -noisy                  print out progress messages for each class in each jar
       -quiet                  suppress all output
       -[no]aot                also perform AOT compilation on methods after storing classes into cache
       -aotFilter <signature>  only matching methods will be AOT compiled and stored into cache
          e.g. -aotFilter {mypackage/myclass.mymethod(I)I} compiles only mymethod(I)I
          e.g. -aotFilter {mypackage/myclass.mymethod*} compiles any mymethod
          e.g. -aotFilter {mypackage/myclass.*} compiles all methods from myclass
       -aotFilterFile <file>   only methods matching those in file will be AOT compiled and stored into
```

```
                            cache (input file must have been created by -Xjit:verbose={precompile},
                            vlog=<file>)
       -printvmargs         print VM arguments needed to access populated cache at runtime
       [jar file]*.[jar][zip]  explicit list of jar files to populate into cache
                            if no files are specified, all files.[jar][zip] in the searchPath
                            will be converted.
Exactly one action option must be specified
```

Nota: Si utiliza memorias caché de clase compartida, el nombre especificado por la opción **-cacheName** no debe superar los 53 caracteres.

Se puede especificar una lista de archivos .jar, en cuyo caso solo las clases de dichos archivos .jar se añadirán a la memoria caché de clase compartida. Si no especifica una lista de archivos .jar, utilice la opción **-searchPath <path>** para especificar un árbol de directorios donde buscar archivos .jar o .zip. La opción **-recurse** es la predeterminada e implica que se realizan búsquedas recursivas de archivos .jar o .zip en el árbol de directorios. Las opciones **-norecurse** implican que solo se busca en el directorio especificado. Especifique la opción **-classpath <class path>** de manera que admincache pueda encontrar todas las clases necesarias para procesar los archivos .jar especificados. Las clases se cargan en la JVM como parte del llenado de la memoria caché de clase compartida, por lo que es importante que admincache pueda encontrar todas las clases y superclases a las que se hace referencia cuando intenta cargar una clase desde un archivo .jar.

La opción **-grow** especifica que se añade un nuevo archivo .jar al contenido de la memoria caché existente, si hay una memoria caché de clase compartida existente y con el mismo nombre en el directorio de la memoria caché. La opción **-nogrow** especifica que un nuevo archivo .jar sustituye al contenido anterior de la memoria caché, si hay una memoria caché de clase compartida existente con el mismo nombre en el anterior directorio de la memoria caché. La opción **-grow** se utiliza para añadir nuevos archivos .jar que actualmente no existen a la memoria caché de clase compartida, no para sustituir las clases que han cambiado. No utilice la opción **-grow** para actualizar las clases que ya se encuentran en la memoria caché, pero que han cambiado debido a modificaciones realizadas en la aplicación. Para actualizar las clases existentes, cree una memoria caché totalmente nueva con el contenido de la clase actual. Si no actualiza su memoria caché de clase compartida al cambiar una clase, su aplicación se ejecutará correctamente con el contenido de la clase nueva, pero no sacará todo el partido a la memoria caché de clase compartida. Esto se debe a que la clase cambiada se cargará desde el disco, y no desde la memoria caché de clase compartida. Al cargar la clase desde el disco, el código compilado mediante AOT no se podrá utilizar para dicha clase. Vuelva a generar su memoria caché de clase compartida cuando cambie una clase.

Utilice las opciones **-quiet**, **-verbose** y **-noisy** para controlar el nivel de detalles que ofrece admincache.

Para especificar la compilación previa AOT (de manera anticipada) para estos métodos en las clases que llenan la memoria caché de clase compartida, utilice la opción **-aot**. Para evitar la compilación previa AOT y sencillamente almacenar las clases en la memoria caché de clase compartida, utilice la opción **-noaot**. La opción **-aot** es el valor predeterminado.

Para precompilar algunos métodos de forma selectiva, utilice las opciones **-aotFilter** <*firma*> o **-aotFilterFile** <*archivo*>. La <*firma*> es una expresión regular simplificada para una firma de método, colocada entre llaves, donde '*'

puede sustituir a cualquier secuencia de caracteres. Puede que necesite colocar la <*firma*> entre comillas simples para que el shell no interprete ninguno de los caracteres de la firma de método.

Tabla 5 muestra algunos ejemplos de la opción <*firma*>.

Tabla 5. Ejemplos de la opción <firma>

Firma	Significado
`-aotFilter '{java/lang/*}'`	AOT compila los métodos en el paquete java/lang.
`-aotFilter '{*.sample*}'`	AOT compila los métodos que empiezan por "sample".
`-aotFilter '{mypackage/myclass.mymethod(I)I}'`	AOT compila el método con su firma exacta.

La opción **-aotFilterFile** <*archivo*> utiliza el contenido de <*archivo*> para seleccionar los métodos de la compilación AOT. Ningún otro método se compila mediante AOT. El contenido de <*archivo*> se genera durante una ejecución anterior de la aplicación utilizando la opción **-Xjit:verbose={precompile},vlog=**<*archivo*>. La salida detallada almacenada en <*archivo*> utiliza un formato interno. Este formato resulta necesario para la opción **-aotFilterFile**.

Nota: La opción **-vlog=**<*archivo*> no genera directamente un archivo denominado "archivo". Se añade una cadena con la fecha y el ID de proceso a la palabra "archivo" cuando se genera la salida detallada. Al especificar la opción **-Xjit:verbose={precompile},vlog=my_file**, el nombre del archivo generado es similar a `my_file.<fecha>.<#>.<ID_proceso>` Los campos adicionales facilitan que se generen archivos de registro detallados individuales en casos de varias JVM en los que podría resultar difícil proporcionar opciones de línea de mandatos a una JVM concreta, o bien utilizar opciones de línea de mandatos **-Xjit** distintas con distintas JVM. En un caso de JVM única, estos números de añaden al nombre del archivo proporcionado en la línea de mandatos.

Un archivo generado se puede utilizar con la opción **-aotFilterFile**, sin que sea necesario realizar ningún tipo de edición. Varios archivos de registro detallado generados por varias ejecuciones de la aplicación utilizando la opción **-Xjit:verbose={precompile},vlog=<archivo>** se pueden concatenar y proporcionar a admincache utilizando la opción **-aotFilterFile**.

La opción **-printvmargs** le ayuda a garantizar que se proporcionan los argumentos correctos en la línea de comandos cuando se ejecuta la aplicación.

```
$ admincache -Xrealtime -classpath myapp.jar -cacheDir myCacheDir -cacheName
myCache -populate myapp.jar -printvmargs

admincache 1.02
Converting files
Processing classes in /team/triage/180724/bin/myapp.jar into shared class cache
No errors while processing jar file /team/triage/180724/bin/myapp.jar

Processing complete

VM args needed at runtime: -Xshareclasses:name=myCache,cacheDir=/tmp/peter
-classpath myapp.jar -Xaot
```

En este ejemplo, la última línea de la salida muestra las opciones que se deben añadir a la línea de mandatos al ejecutar la aplicación, de manera que se utilicen

las clases y los métodos AOT almacenados en la memoria caché de clase compartida. Para utilizar las opciones de este ejemplo, especifique el mandato:

```
java -Xshareclasses:name=myCache,cacheDir=myCacheDir -classpath myapp.jar -Xaot
myMainClass <application arguments>
```

Gestión de memorias caché de clase compartida con admincache:

La herramienta admincache incluye varios programas de utilidad para gestionar las memorias caché de clase compartida en su sistema.

La herramienta admincache proporciona programas de utilidad para ayudarle con varias actividades.

- Listado de las memorias caché de clase compartida presentes en una memoria caché.
- Proporción de detalles sobre el contenido de una memoria caché de clase compartida.
- Eliminación de algunas o todas las memorias caché de un directorio específico de memoria caché.

Listado de memorias caché de clase compartida disponibles:

La herramienta admincache proporciona una lista de memorias caché de clase compartida presentes en una memoria caché.

Para obtener un listado de todas las memorias caché de clase compartida presentes en una memoria caché, utilice la opción **-listAllCaches** y especifique el directorio de la memoria caché utilizando la opción **-cacheDir**.

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

Listing all caches in cacheDir /tmp/javasharedresources/

Cache name              level         persistent  last detach time

Compatible shared caches
sharedcc_username       Java6 32-bit  yes         Thu Oct 16 17:02:39 2008
rtCache                 Java6 32-bit  yes         Thu Oct 16 17:03:12 2008

Incompatible shared caches
nonrtCache              Java6 32-bit  yes         Thu Oct 16 17:17:32 2008
```

En este ejemplo, hay dos memorias caché de clase compartida en el directorio de memoria caché predeterminado:

- La memoria caché nombrada predeterminada para un usuario con el inicio de sesión *nombre_usuario*
- Otra memoria caché denominada *rtCache*

El ejemplo muestra también una memoria caché incompatible denominada *nonrtCache*. La JVM creó *nonrtCache* cuando se ejecutaba en modalidad no de tiempo real. Esto significa que no se puede acceder a ella utilizando la opción **-Xrealtime**.

La JVM en modalidad de tiempo real puede ver las memorias caché creadas en modalidad no de tiempo real. La JVM en modalidad no de tiempo real puede ver las memorias caché creadas en modalidad de tiempo real.

```
$ admincache -listAllCaches
J9 Java(TM) admincache 1.0
Licensed Materials - Property of IBM

(c) Copyright IBM Corp. 1991, 2008  All Rights Reserved
IBM is a registered trademark of IBM Corp.
Java and all Java-based marks and logos are trademarks or registered
trademarks of Oracle Corporation


Listing all caches in cacheDir /tmp/javasharedresources/

Cache name              level         persistent  last detach time

Compatible shared caches
nonrtCache              Java6 32-bit  yes         Thu Oct 16 17:17:32 2008
```

En este ejemplo, *nonrtCache* aparece en la lista y se muestra como compatible porque no se ha especificado **-Xrealtime**.

Inspección de contenido de las memorias caché de clase compartida:

La herramienta admincache describe el contenido de una memoria caché de clase compartida.

Puede utilizar la opción **-printStats** de la herramienta admincache para obtener una visión general en la que se describa el contenido principal de una memoria caché de clase compartida. Para obtener información sobre una memoria caché específica, en un directorio de memoria caché específico, utilice las opciones **-cacheName** y **-cacheDir**. En el ejemplo siguiente se ofrece información sobre la memoria caché *nonrtCache* del directorio predeterminado de memoria caché.

```
$ admincache -cacheName nonrtCache -printStats

admincache 1.02

Current statistics for cache "nonrtCache":


base address       = 0xD5445000
end address        = 0xD6437000
allocation pointer = 0xD5529FA8

cache size         = 16776852
free bytes         = 14070360
ROMClass bytes     = 1166004
AOT bytes          = 1437412
Data bytes         = 57440
Metadata bytes     = 45636
Metadata % used  = 1%

# ROMClasses       = 372
# AOT Methods      = 981
# Classpaths                          = 1
# URLs           = 0
# Tokens         = 0
# Stale classes = 0
% Stale classes  = 0%

Cache is 16% full
```

Nota: Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.

Existen varios fragmentos de información útil sobre esta memoria caché:

- El tamaño de la memoria caché, que aparece como `cache size = 16776852`.
- El espacio disponible en la memoria caché, mostrado como `free bytes = 14070360`. Puede calcular que la memoria caché está llena en un 16%.
- La cantidad de clases almacenadas en la memoria caché, mostrada como `# ROMClasses = 372`.
- La cantidad de métodos AOT almacenados en la memoria caché, mostrada como `# AOT Methods = 981`.

Para obtener más detalles sobre la información proporcionada por la opción **-printStats** en la herramienta admincache, consulte el apartado Programa de utilidad printStats.

La opción **-printAllStats** proporciona una descripción más detallada del contenido de una memoria caché de clase compartida. Entre la información se incluye la lista de clases y métodos AOT almacenados en la memoria caché. La salida de la opción **-printAllStats** es detallada.

Las clases contenidas en la memoria caché se indican por líneas similares a la siguiente:

```
1: 0xD643B788 ROMCLASS: java/lang/ClassLoader at 0xD5469B88.
```

Esta línea indica que la clase java/lang/ClassLoader está contenida en la memoria caché. Las direcciones son internas de la memoria caché de clase compartida y pocas veces resultan útiles, excepto con fines de diagnóstico.

Los métodos AOT contenidos en la memoria caché se indican por líneas similares a la siguiente:

```
1: 0xD643B290 AOT: callerClassLoader
        for ROMClass java/lang/ClassLoader at 0xD5469B88.
```

Estas líneas indican que el método callerClassLoader de la clase java/lang/ClassLoader está contenido en la memoria caché. Las direcciones que aparecen son direcciones internas de memoria caché compartida. La salida de la opción **-printAllStats** no muestra la firma de cada uno de los métodos AOT de la memoria caché, donde la firma consta de los tipos de parámetros y el tipo de retorno.

Para obtener más detalles sobre la información proporcionada por la opción **-printAllStats** en la herramienta admincache, consulte Programa de utilidad printAllStats.

Destrucción de memorias caché de clase compartida:

La herramienta admincache tiene opciones para borrar una memoria caché concreta o todas las memorias caché de un directorio específico de la memoria caché.

La opción **-destroy** de la herramienta admincache se utiliza para borrar una memoria caché específica de un directorio específico de memoria caché, si el usuario tiene permiso para hacerlo. La opción **-destroyAll** se utiliza para borrar todas las memorias caché, si el usuario tiene permiso para hacerlo. Por ejemplo:

```
$ admincache -Xrealtime -destroy

admincache 1.02

JVMSHRC256I Persistent shared cache "sharedcc_username" has been destroyed
```

Después de borrar la memoria caché, una lista de las memorias caché de clase compartida disponibles en el directorio predeterminado de memoria caché muestra que la memoria caché borrada ya no existe:

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

Listing all caches in cacheDir /tmp/javasharedresources/

Cache name              level         persistent  last detach time

Compatible shared caches
rtCache                 Java6 32-bit  yes         Thu Oct 16 17:03:12 2008

Incompatible shared caches
nonrtCache              Java6 32-bit  yes         Thu Oct 16 17:17:32 2008
```

La opción **-destroyAll** elimina todas las memorias caché del directorio de memoria caché especificado, independientemente de si son compatibles o no con la JVM actual. La opción **-destroyAll** se debe utilizar con mucho cuidado:

```
$ admincache -Xrealtime -destroyAll

admincache 1.02

Attempting to destroy all caches in cacheDir /tmp/javasharedresources/

JVMSHRC256I Persistent shared cache "rtCache" has been destroyed
JVMSHRC256I Persistent shared cache "nonrtCache" has been destroyed
```

El resultado es que deja de haber memorias caché de clase compartida disponibles en la máquina:

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

JVMSHRC005I No shared class caches available
```

Si el usuario actual no tiene permiso para acceder a una memoria caché, la memoria caché no será destruida por las opciones **-destroy** ni **-destroyAll**.

Tamaños prácticos para memorias caché de clase compartida:

La herramienta admincache proporciona información para el dimensionamiento de memorias caché de clase compartida.

Para las aplicaciones más pequeñas, una memoria caché de clase compartida se puede llenar con todas las clases y los métodos sin que se genere un tamaño de memoria caché demasiado grande. Para aplicaciones más grandes, el tamaño de la memoria caché de clase compartida resultante podría ser demasiado grande, por cuestiones prácticas. Se debe a que un proceso de la máquina virtual Java debe tener suficiente espacio de direcciones virtuales como para ocuparse de todo el contenido de la memoria de clase compartida. Hay algunas consideraciones que puede aplicar si utiliza la tecnología de memoria caché de clase compartida.

La memoria caché de clase compartida debe ser virtualmente localizable en su totalidad, en cualquier JVM que se conecte a ella. Esto significa que se debe evitar el uso de memorias caché de clase compartida de más de 700 MB. La herramienta admincache puede predecir el tamaño de una memoria caché. Si la herramienta indica que la memoria caché superará el límite de 700 MB, aparecerá un mensaje avisándole de que debe almacenar un número de clases menor, o bien indicándole que sea más selectivo con los métodos AOT almacenados en la memoria caché.

```
$ admincache -Xrealtime -populate veryBigJar.jar -cp <Mi classpath>

admincache 1.02

WARNING: predicted cache size (15960MB) exceeds recommended maximum shared class cache size of 700MB
If your jar files contain primarily class files then you may not be able to create a cache of this size
        or you may not be able to connect to the created cache when you run your application.
Alternatively, you may want to more selectively compile AOT methods by using -aotFilterFile
To override this warning message, please directly specify -Xscmx15960M on your command-line
        but beware that the resulting failure may not occur until the very end of the population
        procedure.
```

La herramienta admincache predice un tamaño de memoria caché conservador, en función del tamaño total de los archivos .jar especificados o hallados para el llenado. Esto significa que la predicción podría no ser precisa si el archivo .jar contiene muchos archivos que no son archivos de clase. Para obtener una predicción más precisa del tamaño de memoria caché, cree versiones temporales del archivo .jar que contengan solo los archivos de clase. Si la herramienta admincache sigue produciendo un mensaje de aviso, podría optar porque AOT realice una compilación previa más selectiva de los métodos del archivo .jar, utilizando las opciones **-aotFilter** <*patrón*> o **-aotFilterFile** <*archivo*>. El mensaje de la herramienta admincache le recuerda que la predicción no tiene en cuenta los métodos AOT filtrados por estas opciones.

Para alterar temporalmente el mensaje de aviso y continuar con el paso de llenado de la memoria caché, añada la opción **-Xscmx** indicada a la línea de mandatos admincache. Si el tamaño predicho es muy grande, es posible que la herramienta admincache no pueda crear una memoria caché de clase compartida del tamaño requerido. Para solucionarlo, reduzca el tamaño de la memoria caché hasta que la herramienta admincache pueda continuar.

Cuando se escriba la memoria caché final en el disco, tendrá el tamaño necesario para contener las clases y los métodos AOT especificados. Esto significa que especificar un tamaño de memoria caché inicial más grande no supone un problema.

Almacenamiento de clases SDK en una memoria caché de clase compartida:

Puede que no sea necesario crear una memoria caché que contenga todos los archivos .jar del kit de desarrollo de software para todas las aplicaciones.

La cantidad y el tamaño de los archivos .jar del SDK implican que intentar crear una memoria caché que contenga todos estos archivos .jar genera un mensaje de aviso que indica que la memoria caché resultante será demasiado grande. En muchas aplicaciones, nunca se hará referencia a la mayoría de los archivos .jar del kit de desarrollo de software.

Los principales archivos .jar del kit de desarrollo de software se encuentran en el directorio `SDK/jre/lib`. Para la mayoría de las aplicaciones, el más importante de estos archivos .jar es `rt.jar`, una novedad de los releases de Java 6. `rt.jar` es una colección de clases previamente almacenadas en archivos .jar independientes antes

del release de Java 6. Al llenar una memoria caché de clase compartida con solo `rt.jar` y compilar todos sus métodos con el compilador AOT, se crea una memoria caché de aproximadamente 300 MB de tamaño. Una aplicación habitual no hará referencia a la mayoría de los métodos de las clases `rt.jar`. Para llenar una memoria caché de clase compartida con `rt.jar`:

1. Llene las clases solo desde `rt.jar` a la memoria caché de clase compartida. Esto consume aproximadamente unos 50 MB de espacio en la memoria caché.
2. Utilice la opción **-aotFilterFile** *<archivo>* para compilar solo los métodos que su programa pueda utilizar. Puede generar el *<archivo>* ejecutando la aplicación.

Existen otros archivos .jar importantes y que se utilizan con frecuencia en el SDK, como los siguientes:

- `sdk/jre/lib/i386/realtime/jclSC160/realtime.jar`
- `sdk/jre/lib/i386/realtime/jclSC160/vm.jar`
- `sdk/jre/lib/java.util.jar`

`realtime.jar` contiene la implementación de IBM de Real Time Specification for Java (RTSJ). Si su aplicación utiliza cualquiera de las características de RTSJ, almacene el archivo `realtime.jar` en la memoria caché de clase compartida para conseguir un mejor rendimiento determinante. `vm.jar` contiene varias clases JVM internas, utilizadas habitualmente en todas las aplicaciones. `java.util.jar` contiene varias clases de contenedor y se debe almacenar en la memoria caché de clase compartida de todas las aplicaciones para conseguir un mejor rendimiento determinante.

Otros archivos .jar de los directorios `sdk/jre/lib` y `sdk/jre/lib/ext` se pueden almacenar en una memoria caché de clase compartida si una aplicación utiliza dichas clases. La manera más sencilla de identificar si su aplicación utiliza estas clases es utilizar la opción **-verbose:dynload** al ejecutar el programa. La opción **-verbose:dynload** solo describe las clases cargadas por la ejecución actual de la aplicación. Por ejemplo:

```
<Loaded java/io/InputStreamReader from /myjdk/sdk/jre/lib/rt.jar>
<  Class size 2126; ROM size 2280; debug size 0>
<  Read time 54 usec; Load time 47 usec; Translate time 86 usec>
<Loaded java/util/LinkedHashSet from /myjdk/sdk/jre/lib/java.util.jar>
<  Class size 1218; ROM size 1136; debug size 0>
<  Read time 48 usec; Load time 31 usec; Translate time 55 usec>
<Loaded java/util/HashSet from /myjdk/sdk/jre/lib/java.util.jar>
<  Class size 3171; ROM size 2664; debug size 0>
<  Read time 71 usec; Load time 70 usec; Translate time 118 usec>
```

En esta salida de muestra se ven tres clases cargadas desde dos archivos .jar de SDK distintos. La clase java/io/InputStreamReader se ha cargado desde `rt.jar`. Las clases java/util/LinkedHashSet y java/util/HashSet se han cargado desde `java.util.jar`.

Otras consideraciones sobre admincache:

Información útil para trabajar con admincache.

Llenado de la memoria caché y dimensionamiento de la memoria inmortal

Cuando la herramienta **admincache** llena una memoria caché de clase compartida en modalidad de tiempo real, debe cargar cada clase a medida que va por el proceso de relleno. Cada clase consume parte de la memoria inmortal, aunque es

posible que el tamaño predeterminado de la memoria inmortal no sea lo suficientemente grande para todas las clases solicitadas. Si la herramienta **admincache** genera un error OutOfMemory mientras llena una memoria caché con varias clases, intente aumentar el tamaño de la memoria inmortal más allá de los 16 MB predeterminados, utilizando la opción **-Xgc:immortalMemorySize=32M**.

Si cambia las clases

Si un archivo de clase cambia en el disco, la tecnología de memoria caché de clase compartida detecta automáticamente que no se debe utilizar la versión almacenada en memoria caché de dicha clase en una memoria caché de clase compartida. Su programa funcionará correctamente, pero no podrá aprovechar al máximo la memoria caché de clase compartida, y no se utilizarán los métodos AOT para dicha clase. Si cambia una clase en su aplicación, vuelva a crear su memoria caché de clase compartida. No intente utilizar la opción **-grow** para volver a llenar solo el archivo .jar que contiene la clase modificada, porque esta opción no se ha diseñado para un caso en el que el archivo .jar siga existiendo en la memoria caché.

Gestión de memorias caché compartidas

Las memorias caché compartidas requieren un espacio de direcciones incluso aunque no se hayan cargado archivos. Consulte el apartado "Cómo gestiona la memoria la JVM de IBM" en la página 116 para obtener más información sobre cómo las memorias caché de clase compartida consumen memoria en el proceso JVM.

Almacenamiento de archivos .jar precompilados en una memoria caché de clase compartida

Puede almacenar todas o algunas de las clases Java proporcionadas por IBM en una memoria caché de clase compartida. Este proceso utiliza la opción **-Xrealtime** con **javac** y la herramienta **admincache** para almacenar las clases en una memoria caché de clase compartida.

Antes de empezar

Los archivos .jar almacenados de forma anticipada en una memoria caché de clase compartida solo se admiten con la opción **-Xrealtime** y cuando se ejecuta java con la opción **-Xrealtime**. Puede utilizar los mismos archivos .jar cuando realice la ejecución con o sin **-Xrealtime**, pero los archivos .jar almacenados en la memoria caché solo se pueden utilizar cuando se especifica **-Xrealtime**.

Nota: Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.

Acerca de esta tarea

Puede almacenar archivos jar en una memoria caché de clase compartida utilizando la herramienta **admincache**. **admincache** le permite crear su aplicación utilizando una de las tres opciones posibles.

Nota:
- Si ha establecido un tiempo de espera en su sistema Linux, es posible que tenga que sustituirlo al precompilar archivos .jar más grandes; de lo contrario, la compilación superará el tiempo de espera y no se creará el archivo .jar.

Precompilación de todas las clases y métodos en una aplicación:

Este procedimiento precompila todas las clases de una aplicación. Almacena un conjunto de archivos jar en una memoria caché de clase compartida. Todos los métodos de todas las clases de los archivos .jar se almacenan en la memoria caché. Los archivos .jar optimizados tienen todos los métodos compilados.

Acerca de esta tarea

En este ejemplo, la aplicación reside en el directorio especificado por la variable de entorno *$APP_HOME* y los archivos .jar se encuentran en el subdirectorio *$APP_HOME*/lib. La aplicación utiliza también algunas clases de las proporcionadas por IBM en core.jarrt.jar. En este caso, puede precompilar solo el código de aplicación, particularmente main.jar y util.jar.

De forma predeterminada, la memoria caché de clase compartida se encuentra en /tmp/javasharedresources. Utilice la opción **-cacheDir** para colocar la memoria caché en un directorio distinto. No puede crear una memoria caché en un sistema de archivos de red.

Procedimiento

1. Desde un indicador de shell, especifique: cd *$APP_HOME*
   donde *$APP_HOME* es el directorio de su aplicación.
2. Desde un indicador de shell, especifique: cd *$APP_HOME*/lib. *$APP_HOME*/lib es el directorio donde están almacenadosmain.jar y util.jar.
3. Desde un indicador de shell, especifique: admincache -Xrealtime -populate -aot -classpath *$APP_HOME*/lib -searchPath *$APP_HOME*/lib -norecurse . Este procedimiento optimiza todos los archivos .jar que se encuentran en *$APP_HOME*/lib, grabando la información de progreso en la pantalla y, a continuación, creando el nuevo archivo .jar en el directorio *$APP_HOME*/aot. Puede especificar un nombre de memoria caché con **-cacheName <nombre>**, pero se utiliza un nombre predeterminado basado en el inicio de sesión del usuario si no se ha especificado ninguno.

   Nota: El nombre especificado por la opción **-cacheName** no debe superar los 53 caracteres.
4. Desde un indicador de shell, si especifica: admincache -Xrealtime -listAllCaches se muestra la existencia de la memoria caché.

Qué hacer a continuación

Para ver más opciones, especifique: admincache -Xrealtime -help.

Métodos de precompilación usados con frecuencia:

Puede utilizar la compilación AOT controlada por perfiles para precompilar solo los métodos que la aplicación usa con frecuencia. La compilación AOT almacena un conjunto de archivos jar en una memoria caché de clase compartida utilizando un archivo de opciones generado mediante la ejecución de la aplicación con una opción especial **-Xjit:verbose={precompile},vlog=optFile**. Solo se precompilan los métodos que aparecen en el archivo de opciones.

Antes de empezar

Antes de empezar, cree una lista de los métodos que normalmente se compilan mediante un compilador JIT.

Acerca de esta tarea

Puede editar el archivo generado por la opción **-Xjit:verbose={precompile}**. El archivo es una especificación explícita de los métodos que se van a precompilar. Estos métodos son específicos, es decir, contienen la firma completa para compilar cada método, lo que le permite compilar `com/acme/sample.myMethod(J)V`, pero no `com/acme/sample.myMethod(I)V`.

Nota: Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.

Procedimiento

1. Desde un indicador de shell, especifique:

   ```
   cd $APP_HOME
   ```

   donde *$APP_HOME* es el directorio de su aplicación.
2. Desde un indicador de shell, especifique:

```
java -Xjit:verbose={precompile},vlog=$APP_HOME/app.precompileOpts \
     -cp $APP_HOME/lib/demo.jar nombre_aplicación
```

   donde:

   - *app.precompileOpts* es el nombre del archivo de registro que muestra los métodos compilados con JIT.
   - *nombre_aplicación* es el nombre de su aplicación.

   Este mandato crea una lista de métodos compilados utilizando JIT.
3. Desde un indicador de shell, especifique:

   ```
   cd $APP_HOME/lib
   ```

   $APP_HOME/lib es el directorio donde se almacenan los archivos .jar de la aplicación.
4. Para compilar todos los métodos de aplicaciones de muestra en la memoria caché, especifique:

   ```
   admincache -Xrealtime -populate -cacheName myCache \
              -aotFilterFile $APP_HOME/app.precompileOpts \
              -cp $APP_HOME/lib/demo.jar
   ```
5. Para compilar `realtime.jar` y `vm.jar` en la memoria caché, especifique:

```
admincache -Xrealtime -populate -grow -cacheName myCache \
           -aotFilterFile $APP_HOME/app.precompileOpts \
           -searchPath $JAVA_HOME/jre/bin/realtime/jclSC160
           -cp $APP_HOME/lib/demo.jar
```

6. Para compilar `rt.jar` en la memoria caché, especifique:

```
admincache -Xrealtime -populate -grow -cacheName myCache \
           -aotFilterFile $APP_HOME/app.precompileOpts \
           $JAVA_HOME/jre/lib/rt.jar
           -cp $APP_HOME/lib/demo.jar
```

7. Para probar este mandato, ejecute esta aplicación con la opción **-nojit**, que utiliza el código de la memoria caché. Desde el indicador de shell, especifique:

```
java -Xrealtime -Xshareclasses:name=myCache -Xnojit \
     -cp $APPHOME/aot/demo.jar nombre_aplicación
```

donde *nombre_aplicación* es el nombre de su aplicación.

Precompilación de archivos proporcionados por IBM:

Puede precompilar los archivos proporcionados por IBM, como por ejemplo `rt.jar`, para conseguir un equilibrio entre el rendimiento y la capacidad de predicción.

Acerca de esta tarea

La precompilación resulta parecida a la tarea de precompilar los archivos .jar de su aplicación, pero hay un requisito adicional que se aplica durante el tiempo de ejecución; debe asegurarse de que la classpath de arranque se haya especificado correctamente para utilizar estos archivos en lugar de los archivos de JRE. Puede hacerlo con la opción **-Xshareclasses**, que indica a la JVM que busque primero en la memoria caché de la clase especificada antes de buscar en las ubicaciones predeterminadas de vía de acceso de clases.

Nota: Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.

Precompilar `rt.jar` para utilizarlos con la aplicación:

Procedimiento

1. Desde un indicador de shell, especifique: `cd $JAVA_HOME/lib` donde *$JAVA_HOME* es su directorio de inicio de Java.
2. Ejecute la herramienta **admincache**. En un indicador del intérprete de mandatos, escriba:

   ```
   admincache -Xrealtime -populate -cacheName myCache
       -classpath <class path> rt.jar
   ```

   Este mandato llena la memoria caché denominada `myCache` con los resultados de la precompilación del archivo proporcionado por IBM denominado `rt.jar`.
3. Ejecute su aplicación especificando la opción **-Xshareclasses** para especificar el nombre de la memoria caché. Para ejecutar su aplicación, especifique:

   ```
   java -Xrealtime -Xnojit -Xshareclasses:name=myCache
            -classpath:$APP_HOME/main.jar:$APP_HOME/util.jar ...
   ```

Compilador Just-In-Time (JIT)

Puede controlar cuándo y cómo opera el compilador JIT utilizando la clase java.lang.Compiler proporcionada como parte de la biblioteca de clase SDK estándar. IBM da soporte total a los métodos Compile.compileClass(), Compiler.enable() y Compiler.disable().

Por ejemplo, si desea preparar su aplicación y sabe que se han compilado los métodos clave de su aplicación, puede llamar al método Compiler.disable() después de haber preparado su aplicación y confiar en que la compilación JIT no se producirá durante el tiempo restante de ejecución de la aplicación.

Puede controlar la compilación del método de dos formas:

- Especifique un conjunto de métodos que pueda compilar:

  ```
  Compiler.command("{<especificación_método>}(compile)");
  ```

donde <*especificación_método*> es una lista con todos los métodos que se han cargado hasta el momento y que se van a compilar. <*especificación_método*> describe un nombre de método completo. Un asterisco denota una coincidencia de comodín.

Por ejemplo, para compilar todos los métodos que empiezan por java.lang.String que ya se han cargado, especifique:

```
Compiler.command("{java.lang.String*}(compile)");
```

Nota: Este mandato no solo compila los métodos de la clase java.lang.String, sino también los de la clase java.lang.StringBuffer, que quizás no es lo que quiere el usuario. Para compilar solo los métodos de la clase java.lang.String, especifique:

```
Compiler.command("{java.lang.String.*}(compile)");
```

- Especifique que todos los métodos de la cola de compilación se compilarán antes de que esta hebra se ejecute y continúe:

  ```
  Compiler.command("waitOnCompilationQueue");
  ```

  Es posible que desee que la cola de compilación esté vacía antes de inhabilitar el compilador. Una técnica habitual para compilar un conjunto de métodos y clases puede ser:

```
Compiler.enable();                              // asegúrese de que el compilador esté activo
Compiler.command("{com.mycompany.*}(compile)"); // ponga en cola todos los métodos que desee compilar
Compiler.command("waitOnCompilationQueue");     // espere hasta que se hayan compilado todos esos métodos
Compiler.disable();                             // desactive el compilador
```

Determinación durante las transiciones de JNI

De forma predeterminada, JIT genera código optimizado para las transiciones JNI de alto rendimiento de código Java a nativo (J2N). La determinación reducida se puede producir cuando se vuelve a cargar una biblioteca nativa utilizando la siguiente secuencia de código:

```
RegisterNatives / UnregisterNatives / RegisterNatives
```

Para volver a un código más lento y determinante, utilice la opción de línea de mandatos **-Xjit:disableDirectToJNI**.

Habilitación del compilador JIT

Puede habilitar explícitamente el JIT de varias formas. Las dos opciones de línea de mandatos alteran temporalmente la variable de entorno **JAVA_COMPILER**.

Procedimiento

- Establezca la variable de entorno **JAVA_COMPILER** en "jitc" antes de ejecutar la aplicación Java. En un indicador del intérprete de mandatos, escriba:
  - **Para el shell Korn:** `export JAVA_COMPILER=jitc`

    Nota: En esta información se utilizan los mandatos del shell Korn a menos que se indique lo contrario.
  - **Para el shell Bourne:**

    ```
    JAVA_COMPILER=jitc
    export JAVA_COMPILER
    ```
  - **Para el C shell:** `setenv JAVA_COMPILER jitc`

  Si la variable de entorno **JAVA_COMPILER** es una serie vacía, el JIT continúa inhabilitado. Para inhabilitar la variable de entorno, en un indicador shell, escriba `unset` **JAVA_COMPILER**.

- Utilice la opción **-D** en la línea de mandatos de la JVM para establecer la propiedad java.compiler en "jitc". En un indicador de shell, especifique: `java -Djava.compiler=jitc <MyApp>`
- Utilice la opción **-Xjit** en la línea de mandatos de la JVM. *No* debe especificar la opción **-Xint** al mismo tiempo. En un indicador de shell especifique: `java -Xjit <MyApp>`

Inhabilitación de JIT

Puede inhabilitar el JIT de varias formas. Las dos opciones de línea de mandatos alteran temporalmente la variable de entorno **JAVA_COMPILER**.

Acerca de esta tarea

Procedimiento

- Establezca la variable de entorno **JAVA_COMPILER** en "NONE" o en la hebra vacía antes de ejecutar la aplicación Java. En un indicador de shell, escriba:
  - **Para el shell Korn:** `export JAVA_COMPILER=NONE`

    Nota: En el resto se esta información se utilizan los mandatos de shell Korn.
  - **Para el shell Bourne:**

    ```
    JAVA_COMPILER=NONE
    export JAVA_COMPILER
    ```
  - **Para el C shell:** `setenv JAVA_COMPILER NONE`
- Utilice la opción **-D** de la línea de mandatos de la JVM para establecer la propiedad java.compiler en "NONE" o en la serie vacía. En un indicador de shell, especifique: `java -Djava.compiler=NONE <MyApp>`
- Utilice la opción **-Xint** en la línea de mandatos de la JVM. En un indicador de shell, especifique: `java -Xint <MyApp>`

Cómo determinar si el JIT está habilitado

Puede determinar el estado del JIT utilizando la opción**-version**.

Procedimiento

Especifique el mandato siguiente en un indicador de shell:

```
java -version
```

Si el JIT no se está utilizando, aparece un mensaje con el texto siguiente:
`(JIT disabled)`
Si el JIT se está utilizando, aparece un mensaje con el texto siguiente:
`(JIT enabled)`

Utilización de hebras en tiempo real de almacenamiento no dinámico

La recogida de basura Metronome ofrece tiempos de respuesta más coherentes, pero en ocasiones resulta adecuado evitar por completo las interrupciones en la recogida de basura.

Las NoHeapRealtimeThreads (NHRT) son una extensión de RealtimeThreads. Difieren de las RealtimeThreads (hebras de tiempo real) en que no tienen acceso al almacenamiento dinámico. Sin acceso al almacenamiento dinámico, las NHRT se pueden continuar ejecutando incluso durante un ciclo de recogida de basura, con algunas restricciones. Si acceso al almacenamiento dinámico, el modelo de programación es distinto al de las hebras de tiempo real.

Consideraciones en la utilización de NHRT

Tenga estos tres puntos sobre NHRT en cuenta:

- La razón principal para utilizar NHRT es en el caso de una tarea que no tolera la recogida de basura. Por ejemplo, si el tiempo es muy importante para su aplicación y no puede tolerar interrupciones.
- Si el tiempo es tan importante que está utilizando NHRT, considere también utilizar el compilador AOT (ahead-of time); es decir, utilice la opción **-Xnojit**.
- Si utiliza la opción **-Xrealtime**, utilizará automáticamente Recopilador de basura Metronome. Las ventajas de Recopilador de basura Metronome podrían ser suficientes para su empresa y se reduciría la necesidad de codificar NHRT.
- Las hebras de NHRT se ejecutan de forma independiente con respecto al recopilador de basura porque tienen una prioridad mayor que la del recopilador de basura. Las hebras de Java pueden tener una prioridad en el rango 1 - 10. Si hay NHRT presentes, la prioridad de las hebras de Java se restablece a 0, independientemente de la prioridad definida en el programa. El recopilador de basura se define automáticamente medio paso más alto que la hebra en tiempo real más alta. Defina la prioridad de sus NHRT para que sean al menos un paso más altos que la hebra en tiempo real más alta. De este modo, los NHRT son independientes del recopilador de basura.

  Nota: Los NHRT no están totalmente libres de la recogida de basura porque el recopilador de basura de la hebra de alarma de Metronome se ejecuta con la prioridad más alta del sistema. Esta prioridad garantiza que la JVM se puede activar para comprobar si el recopilador de basura necesita realizar alguna acción. La tarea para ejecutar la hebra de alarma de Metronome es pequeña y no tiene un efecto significativo sobre el rendimiento. En un sistema multiprocesador, la hebra de alarma se puede ejecutar simultáneamente con otras hebras de NHRT, de forma que no se produzcan interrupciones en la recogida de basura.
- Dado que las hebras NHRT están restringidas a las áreas de memoria inmortal y de ámbito, los métodos de Java realizan comprobaciones para garantizar que no se asignen desde el almacenamiento dinámico. El método de inicio comprueba y devuelve una excepción (`MemoryAccessError`) si las hebras NHRT se asignan desde el almacenamiento dinámico. Las hebras NHRT solo pueden acceder a `ImmortalMemory` y `ScopedMemory`.
- La semántica del bloqueo no ha cambiado, por lo que las hebras NHRT pueden ser bloqueadas por hebras normales si existe un bloqueo compartido.
- Una hebra que utiliza el almacenamiento dinámico puede ver esta prioridad incrementada en un método sincronizado si una hebra NHRT trata de utilizar el mismo método.
- Utilice colas que no sean de bloqueo para las comunicaciones entre hebras NHRT y hebras de almacenamiento dinámico. De lo contrario, separe los dos tipos de hebras.

Excepciones

Al utilizar NHRT se pueden producir estas excepciones:

- `IllegalAssignmentError`. Como ejemplo, este error se puede producir cuando se realiza un intento de crear una referencia a la memoria de ámbito en la memoria inmortal.

- `MemoryAccessError`. Como ejemplo, este error se puede producir cuando una hebra NHRT trata de hacer referencia an la memoria de almacenamiento dinámico.

Restricciones en el manejo de sucesos asíncronos

Hay varios casos en los que las hebras NHRT se pueden bloquear durante la recogida de basura, incluyendo:

1. Cuando una NHRT invoca fire(), setHandler() o addHandler() en un AsyncEvent que ya está asociado con manejadores que están asignados desde la memoria de almacenamiento dinámico
2. Cuando una NHRT llama a destroy(), start() o stop() en un Timer que está asociado a los manejadores que se asignan desde la memoria de almacenamiento dinámico
3. Una NHRT es la última hebra que sale del ámbito y apaga los Timer o AsyncEvents del mismo. No obstante, los Timer o AsyncEvents tienen asociados manejadores que se asignan desde la memoria de almacenamiento dinámico

Para evitar estas situaciones con las hebras NHRT:

1. Evite añadir manejadores asignados desde el almacenamiento dinámico a AsyncEvents o Timers que pueden ser activados por una hebra NHRT.
2. Evite condiciones en las que una NHRT sale la última de un ámbito que tenga AsyncEvents o Timers con manejadores asignados desde la memoria de almacenamiento dinámico.

Restricciones de memoria y planificación

La JVM evita que el tiempo real no de pila cargue referencias a objetos que estén en el almacenamiento dinámico en la pila del operando. Para hacerlo, genera una excepción de tipo javax.realtime.MemoryAccessError.

La JVM también evita las referencias a objetos de la memoria de ámbito que se están almacenando en la memoria inmoral de almacenamiento dinámico. Aunque la memoria de ámbito no la utiliza en exclusiva NHRT, es probable que se utilice si la memorial immortal no es adecuada y se necesita asignación de memoria en un contexto NHRT.

Aunque NHRT se esté ejecutando, si llena un campo con una referencia a un objeto, puede sobrescribir correctamente cualquier referencia previamente existente a un objeto del almacenamiento dinámico en dicho campo. NHRT sobrescribirá correctamente la referencia previamente existente sin generar una excepción de tipo MemoryAccessError.

Restricciones en la carga de clases

Las clases se cargan en las mismas áreas de memoria que el cargador de clases. El área predeterminada para los cargadores de clases es la memoria inmortal.

Para que las aplicaciones ofrezcan los tiempos de respuesta esperados, deben estar "calientes". Las aplicaciones deben cargar sus clases antes, de manera que la carga de clases no interrumpa las hebras en tiempo real y los manejadores de sucesos asíncronos más adelante.

Restricciones en hebras Java al ejecutarse con NHRT

Dado que las propiedades del sistema se comparten en una JVM y que cualquier hebra puede acceder a las propiedades del sistema, es necesario tomar algunas precauciones cuando se utilizan los métodos getProperties y setProperties en las JVM en las que se ejecutan NHRT. Para que las hebras NHRT puedan acceder a las propiedades del sistema, se deben encontrar en la memoria inmortal.

La clase `java.lang.System` proporciona varios métodos que permiten a las hebras interaccionar con las propiedades del sistema; incluidos estos métodos:

```
String getProperty(String)
String getProperty(String,String)
Properties getProperties()

String setProperty(String,String)
void setProperties(Properties)
```

La JVM de tiempo real utiliza una instancia de com.ibm.realtime.ImmortalProperties específicamente creada para que el objeto de la JVM de tiempo real almacene todas las propiedades del sistema. El uso de esta instancia garantiza que todas las llamadas a System.setProperty() o System.getProperties.setProperty() provoquen que la propiedad se almacene en la memoria inmortal. No es necesario ningún código de usuario especial en este caso, pero es importante comprender que cada vez que se define una propiedad, se consume parte de la memoria inmortal.

Las llamadas al método setProperties() son un poco más difíciles porque el objeto Properties compartido se utiliza para almacenar las propiedades del sistema. Si una aplicación se ejecuta en una JVM en tiempo real que tiene NHRTs en ejecución, las llamadas al método setProperties deben pasar en una instancia de la clase com.ibm.realtime.ImmortalProperties, o subclase, que se ha creado en memoria inmortal. El uso de esta instancia garantiza que todas las propiedades que se han establecido usando el método setProperties se almacenan en memoria inmortal.

Nota: La invocación de setProperties(null) tiene como resultado la creación interna de un objeto ImmortalProperties nuevo con un conjunto predeterminado de propiedades, que consume memoria inmortal adicional.

Las llamadas al método getProperties() devuelven el objeto definido o bien el objeto de propiedades predeterminado, que es un objeto com.ibm.realtime.ImmortalProperties. Para maximizar la compatibilidad con el código existente que llama a getProperties(), el objeto ImmortalProperties serializa el objeto y, a continuación, lo deserializa en una JVM estándar. El comportamiento predeterminado para la serialización de ImmortalProperties consiste en serializar un objeto común de tipo Properties, porque la JVM estándar no tiene un objeto ImmortalProperties y la deserialización falla. Para cambiar este comportamiento predeterminado, la clase ImmortalProperties proporciona el método enabledReplacement(boolean) que, si se llama con false, inhabilita el comportamiento predeterminado. En este caso, la serialización serializa el objeto ImmortalPropeties y, a continuación, es posible deserializarlo y utilizar el objeto resultante en una llamada a System.setProperties en una JVM de tiempo real.

Nota: La deserialización se produce en la memoria inmortal, que puede consumir una cantidad demasiado grande de este recurso limitado.

Gestor de seguridad

El gestor de seguridad definido para el sistema lo utilizan todos los tipos de hebras de la JVM. Por este motivo, en una JVM de tiempo real en la que se ejecuten hebras NHRT, el gestor de seguridad se debe asignar en la memoria inmortal. La JVM de tiempo real garantiza que cualquier gestor de seguridad especificado en las opciones de línea de mandatos se asigne en la memoria inmortal. El gestor de seguridad también se puede definir mediante llamadas a System.setSecurityManager(SecurityManager). Si la aplicación define el gestor de seguridad de este modo, debe garantizar que el gestor de seguridad se ha asignado desde la memoria inmortal, a fin de que las hebras NHRT se puedan ejecutar correctamente.

Las excepciones generadas y los objetos devueltos por el gestor de seguridad se deben encontrar en la memoria inmortal, si están en la memoria caché, o asignarse en el contexto de asignación actual.

Sincronización

La clase MonitorControl y su subclase PriorityInheritance gestionan la sincronización, en concreto el control de la inversión de prioridades. Estas clases permiten la configuración de una política de control de inversión de prioridades, bien como predeterminada, bien para objetos específicos.

Las clases WaitFreeReadQueue, WaitFreeWriteQueue y WaitFreeDequeue permiten la comunicación sin esperas entre objetos programables (especialmente instancias de NoHeapRealtimeThread) y hebras de Java común.

Las clases WaitFree ofrecen un acceso seguro y simultáneo a los datos compartidos entre instancias de NoHeapRealtimeThread y objetos programables sujetos a retardos a causa de la recogida de basura.

Seguridad de clase de tiempo real no de pila

En determinadas circunstancias, hay partes de la API de JSE que no se pueden utilizar en un contexto no de pila. Se realizan restricciones en las clases compartidas entre hebras de pila y no de pila. Preste atención a las clases proporcionadas con la JVM que se pueden utilizar con seguridad.

Compartición de objetos

Los métodos que se ejecutan en hebras en tiempo real de almacenamiento no dinámico generan una excepción javax.realtime.MemoryAccessError siempre que intentan cargar una referencia en un objeto del almacenamiento dinámico.

Figura 3 en la página 69 es un ejemplo del tipo de código que se debe evitar:

```
/**
 * NHRTError1
 *
 * This example is a simple demonstration of an NHRT accessing
 * a heap object reference.
 *
 * The error generated is:
 *
 * Exception in thread "NoHeapRealtimeThread-0" javax.realtime.MemoryAccessError
 *      at NHRT.run(NHRTError1.java:56)
 *      at javax.realtime.RealtimeThread.runImpl(RealtimeThread.java:1754)

 */
import javax.realtime.*;

public class NHRTError1 {
    public static void main(String[] args) {
        NHRTError1 example = new NHRTError1();

        example.run();
    }

    public NHRTError1() {
        message = new String("This on the heap.");
    }

    static public String message; /* The NHRT can access static fields directly - they are always Immortal. */
    static public NHRT myNHRT = null;

    public void run() {
                ImmortalMemory.instance().executeInArea(new Runnable() {
            public void run() {
                NHRTError1.this.myNHRT = new NHRT();
            }
        });

        myNHRT.start();

        try {
            myNHRT.join();
        } catch (InterruptedException e) {
            // PENDIENTE Bloque catch generado automáticamente
            e.printStackTrace( );
        }

    }
```

Figura 3. Ejemplo de NHRT accediendo a una referencia a objeto de almacenamiento dinámico

```
    /* A NHRT class */
    class NHRT extends NoHeapRealtimeThread {
        public NHRT() {
            super(null, ImmortalMemory.instance());
        }

        /* Prints the String via the static reference in NHRTError1.message */
        public void run() {
            System.out.println("Message: " + NHRTError1.message);
        }
    }

}
```

Figura 4. Ejemplo de NHRT accediendo a una referencia a objeto de almacenamiento dinámico (continuación desde la figura 1)

Figura 3 genera una excepción **javax.realtime.MemoryAccessError**:

```
Exception in thread "NoHeapRealtimeThread-0" javax.realtime.MemoryAccessError
        at NHRTError1$NHRT.run(NHRTError1.java:56)
        at javax.realtime.RealtimeThread.runImpl(RealtimeThread.java:1754)
```

Si a un objeto podrán acceder las hebras en tiempo real de almacenamiento no dinámico y una hebra de Java estándar, el objeto se debe asignar en la memoria inmortal. Del mismo modo, si a un objeto podrán acceder la hebra en tiempo real de almacenamiento no dinámico y las hebras de tiempo real, el objeto se puede mantener también en un área de memoria de ámbito.

En Figura 3 en la página 69, la referencia a la cadena "This on the heap." se mantenía en una variable de clase. Esta variable resulta accesible para NHRT porque todas las clases están asignadas en la memoria inmortal. Del mismo modo, la cadena se podría haber pasado al constructor NHRT.

La mayoría de los objetos contienen referencias a otros objetos y, por lo tanto, se debe tener cuidado cuando estos objetos se comparten entre hebras ordinarias y NHRT. Un ejemplo común es el de una LinkedList asignada en la memoria inmortal, compartida entre una hebra ordinaria y una NHRT. Si no se tiene cuidado, la hebra estándar podría introducir objetos en la LinkedList que se encuentran en el almacenamiento dinámico. Resulta más preocupante que las estructuras de datos asignadas por la LinkedList para rastrear los objetos sean asignadas al almacenamiento dinámico por la hebra ordinaria, lo que puede provocar una excepción MemoryAccessError en el NHRT.

Algunas clases no se pueden compartir de forma segura entre NHRT y otras hebras, independientemente de donde puedan estar asignadas sus instancias individuales. Estas clases se basan en objetos almacenados en variables de clases, normalmente con fines de almacenamiento en caché. InetAddress es un ejemplo típico que almacena las direcciones en la memoria caché; si la primera hebra en llamar a determinados métodos en InetAddress se está ejecutando en el almacenamiento dinámico, los mismos métodos dejan de ser seguros para ser llamados por NHRT en el futuro.

Bloqueo en objetos con NHRT

Las hebras NHRT deben evitar la sincronización con otras hebras. Considere el caso de ejemplo siguiente:

- Una hebra en tiempo real con prioridad baja entra en un método o bloque sincronizado, realizando la sincronización en un objeto.
- Una hebra NHRT de prioridad alta se bloquea cuando trata de sincronizar en el mismo objeto.
- La herencia de prioridades provoca que la hebra en tiempo real asuma temporalmente la misma prioridad que la NHRT.
- La recogida de basura se ejecuta a continuación con una prioridad mayor que NHRT y puede, por lo tanto, interrumpir la NHRT. El motivo para usar la NHRT es evitar la interrupción por parte de la recogida de basura, por lo que este escenario niega el uso de NHRT.

En ocasiones, es inevitable que las hebras NHRT y otras hebras se sincronicen en el mismo objeto, pero debe minimizar esta posibilidad. Tenga cuidado de evitar las sincronizaciones innecesarias cuando comparta objetos.

Restricciones sobre las clases seguras

Es necesario aplicar algunas consideraciones cuando una aplicación contiene objetos de hebras en tiempo real y de hebras en tiempo real de almacenamiento no dinámico.

- La hebra en tiempo real de almacenamiento no dinámico puede sufrir excepciones de tipo MemoryAccessErrors provocadas por la interacción con las hebras de tiempo real.
- La hebra en tiempo real de almacenamiento no dinámico se puede ver accidentalmente retardada por una recogida de basura provocada por la hebra de tiempo real.

Excepciones MemoryAccessError provocadas en una hebra en tiempo real de almacenamiento no dinámico

Si los dos tipos de hebras llaman a métodos en la misma clase, la hebra en tiempo real puede "corromper" las variables estáticas de la clase con objetos asignados desde el almacenamiento dinámico. La hebra en tiempo real de almacenamiento no dinámico recibirá una excepción MemoryAccessError al tratar de acceder a dichos objetos de almacenamiento dinámico. La corrupción también se puede producir en instancias de la clase. Lamentablemente, ambos problemas son bastante habituales en patrones de codificación típicos y, por lo tanto, vale la pena analizar un par de casos.

Si una clase está realizando una operación que lleva mucho tiempo, suele optar por almacenar en memoria caché el resultado para mejorar el rendimiento de las operaciones siguientes. La memoria caché suele ser una recopilación, como una HashMap anclada en una variable estática de la clase. Una hebra en tiempo real que opera e n un contexto de almacenamiento dinámico puede almacenar un objeto de almacenamiento dinámico en esta recopilación, con lo que no solo se añade a la recopilación el propio objeto, sino también los objetos de la infraestructura; por ejemplo, partes del índice. Cuando una hebra en tiempo real de almacenamiento no dinámico trata de acceder más tarde a la recopilación, incluso aunque no esté tratando de acceder al objeto añadido por la otra hebra, trata de cargar los objetos de infraestructura y recibe una excepción MemoryAccessError. A medida que las bibliotecas de clases se desarrollan y se ajustan para el rendimiento, estas memorias caché se vuelven más comunes.

Una instancia de clase se puede corromper también con objetos de almacenamiento dinámico de distintas maneras. Piense en una instancia creada en la memoria inmortal y a la que, por lo tanto, pueden acceder los dos tipos de hebras. Si el primer uso del objeto viene de parte de una hebra en tiempo real en un contexto de almacenamiento dinámico, es posible que se encuentre un objeto secundario almacenado en un campo del objeto original. Si el objeto secundario se encuentra en un contexto de almacenamiento dinámico, el uso subsiguiente por parte de la hebra en tiempo real de almacenamiento no dinámico vuelve a mostrar una excepción MemoryAccessError. Es posible que estos objetos secundarios no se añadan siempre en un primer uso, sino tras un cierto número de usos, y puede que se hayan diseñado para mejorar el rendimiento de los métodos más utilizados.

Hebra no de pila retardada por la recogida de basura

A las hebras no de pila se les deben asignar prioridades más altas que a otras hebras, a fin de evitar un retardo provocado por la recogida de basura.

Además, si una clase contiene métodos sincronizados, es posible que una hebra en tiempo real de almacenamiento no dinámico que llame a dichos métodos se vea accidentalmente retardada por una recogida de basura. Este caso de ejemplo se describe en el apartado "Bloqueo en objetos con NHRT" en la página 70.

Si una clase contiene métodos sincronizados (ya sean métodos estáticos o de instancias), es posible que una hebra en tiempo real de almacenamiento no dinámico que llame a dichos métodos se vea accidentalmente retardada por una recogida de basura. El problema aparece si una hebra en tiempo real está accediendo a un método sincronizado (estático o de instancia) en el punto en el que una hebra en tiempo real de almacenamiento no dinámico trata de llamar a otro método sincronizado que se bloqueará a la espera de que la otra hebra finalice. Si la hebra en tiempo real de almacenamiento no dinámico tiene una prioridad mayor que la hebra en tiempo real, se aumenta la prioridad de ésta. Si dicha hebra se ve forzada a esperar a que se interrumpa la recogida de basura, es posible realizar una inversión de prioridades, ya que la hebra del recopilador de basura tiene una prioridad mayor que la hebra en tiempo real con la prioridad más alta, que quizás no sea tan alta como la hebra en tiempo real de almacenamiento no dinámico que está actualmente bloqueada esperando a entrar en el método sincronizado.

La única manera de solucionar estos problemas es garantizar que hebras en tiempo real de almacenamiento no dinámico nunca llame a métodos sincronizados en clases o instancias compartidas con otros tipos de hebras. Lamentablemente, no siempre está claro desde una firma de método si un método está sincronizado; podría, por ejemplo, contener un bloque sincronizado o llamar a un método sincronizado.

Resumen

La clase NoHeapRealtimeThread añade una dosis de complejidad importante al entorno en tiempo real y pueden aparecer bastantes problemas cuando hay varios tipos de hebras funcionando en un mismo entorno. Durante el desarrollo de una aplicación, debe diseñar con cuidado las áreas en las que ha compartido el uso de clases entre los distintos tipos de hebras. Resulta especialmente importante la utilización que dichas hebras hagan de las clases del SDK. Debido a la complejidad del análisis, es imposible ofrecer ninguna garantía de que todas las clases proporcionadas en el SDK sean seguras para este uso compartido. En lugar de ello, se ha verificado un pequeño subconjunto de clases. Inicialmente, la verificación se ha concentrado en el aspecto MemoryAccessError y el resultado es una lista de clases que se han analizado, comprobado y modificado según lo necesario para garantizar que pueden ser utilizadas por hebras no de pila y de otros tipos.

Clases seguras

En esta sección se muestra el conjunto de clases que NoHeapRealtimeThread y otros tipos de hebra deben utilizar de forma segura.

La principal preocupación se centra en el aspecto de la seguridad MemoryAccessError. En la lista siguiente se muestran las clases de detalles que pueden utilizar los tres tipos de hebras de la misma JVM.

Nota: Es posible que las instancias individuales de la clase no se compartan siempre de forma segura.

Siga estas reglas para asegurarse de que se utilcie una clase de forma segura en todos los tipos de hebras:

- La instancia se debe haber creado en un área de memoria que sea accesible para la hebra que intenta acceder a dicha instancia.
- Si la clase tiene campos estáticos públicos, evite almacenar objetos de almacenamiento dinámico en ellos.

- Si la clase tiene campos de instancia públicos, evite almacenar objetos de almacenamiento dinámico en ellos.

No todas las clases proporcionadas por IBM son seguras para NHRT. Los paquetes siguientes contienen clases que son seguras para NHRT:

- Paquete java.lang
- Paquete java.lang.reflect
- Paquete java.lang.ref (todas las clases)
- Paquete java.net
- Paquete java.io
- Paquete java.math

Estas tablas muestran clases dentro de estos paquetes que no son seguras para NHRT:

Tabla 6. Clases del paquete java.lang que no son seguras para NHRT

Clase	Método
java.lang.ProcessBuilder	*
java.lang.Thread	getAllStackTraces()Ljava.util.Map;
java.lang.ThreadGroup	*
java.lang.ThreadLocal	*
java.lang.InheritableThreadLocal	*

Tabla 7. Clases del paquete java.lang.reflect que no son seguras para NHRT

Clase	Método
java.lang.reflect.Proxy.*	*

Tabla 8. Clases del paquete java.net que no son seguras para NHRT

Clase	Método
java.net.SocketPermission.*	newPermissionCollection()Ljava.net.SocketPermissionCollection;

Tabla 9. Clases del paquete java.io que no son seguras para NHRT

Clase	Método
java.io.ExpiringCache	*
java.io.SequenceInputStream	*
java.io.FilePermission	newPermissionCollection()Ljava.io.FilePermissionCollection;
java.io.ObjectInputStream	*
java.io.ObjectOutputStream	*
java.io.ObjectStreamClass	*

Tabla 10. Clases del paquete java.math que no son seguras para NHRT

Clase	Método
java.math.BigInteger	*

Los paquetes podrían incluir subpaquetes que contengan clases que no sean seguras. Por ejemplo, las clases siguientes no son seguras para NHRT:

- java.lang.management.*

- java.lang.annotation.*
- java.lang.instrument.*

Incluso si se considera que una clase es segura para NHRT, dicha clase podría no ser apropiada para su uso en NHRT. Los desarrolladores de aplicaciones deben determinar los requisitos de tiempo real de las clases una a una, independientemente de si se trata o no de una clase segura para NHRT.

Compartimiento de datos de clases entre varias JVM

La máquina virtual Java (JVM) le permite compartir datos de clases entre varias JVM almacenándolos en un archivo de la memoria caché correlacionado con la memoria en el disco.

Compartir reduce el consumo de almacenamiento virtual global cuando más de una JVM comparte una memoria caché. También se disminuye el tiempo de arranque de una JVM después de que se ha creado la memoria caché. La memoria caché de clase compartida es independiente de cualquier JVM activa y se mantiene hasta que se destruye. Una memoria caché compartida puede contener:

- Clases de arranque
- Clases de aplicación
- Metadatos que describen las clases
- Código compilado de manera anticipada (AOT)

IBM WebSphere Real Time for RT Linux puede utilizar memorias caché de clase compartida en las modalidades de tiempo real o no de tiempo real, pero las técnicas de llenado, creación y formato de la memoria caché difieren. Las memorias caché en modalidad de tiempo real no son compatibles con las memorias caché en modalidad no de tiempo real. En modalidad no de tiempo real, las memorias caché se crean y se llenan de la misma manera que una JVM estándar. Esto significa que la JVM llena y crea la memoria caché a medida que se ejecuta una aplicación, de forma transparente para el usuario. En modalidad de tiempo real, utilizando la opción **-Xrealtime**, las memorias caché de clase compartida se deben crear y llenar previamente con **admincache**, mediante la opción **-populate**. Las aplicaciones que se ejecutan en modalidad de tiempo real pueden leer contenido de la memoria caché rellenada previamente, pero no pueden modificar su contenido.

Utilice la herramienta **admincache** para crear, llenar y destruir memorias caché.

Para habilitar una aplicación para que utilice una memoria caché de clase compartida, añada la opción **-Xshareclasses** a su línea de mandatos. Debido a que las memorias caché en modalidad de tiempo real son de solo lectura, algunas de las subopciones de la modalidad no de tiempo real de **-Xshareclasses** no están disponibles en modalidad de tiempo real.

Para obtener más información, consulte “Utilización de la herramienta admincache” en la página 48, “Datos de clases compartidos entre las JVM para tiempo no real” en la página 103 y “Datos de diagnóstico de clases compartidas” en la página 154.

Ejecución de aplicaciones con una memoria caché de clase compartida

Para ejecutar una aplicación con una memoria caché de clase compartida, utilice la opción **-Xshareclasses** en la línea de mandatos.

Tabla 11 muestra las subopciones disponibles cuando se ejecuta una aplicación en modalidad de tiempo real, utilizando la opción **-Xshareclasses**.

Tabla 11. Subopciones disponibles cuando se ejecuta una aplicación en modalidad de tiempo real

Opción	Significado
cacheDir=<*directorio*>	Establece el directorio en el que se leen y escriben los datos de la memoria caché de clase compartida. De forma predeterminada, el <*directorio*> es /tmp/javasharedresources. El nombre de directorio debe coincidir con el especificado en la opción **-cacheDir** utilizada en el mandato admincache para crear la memoria caché.
name=<*nombre*>	Nombre de la memoria caché de clase compartida que se va a utilizar. El nombre debe coincidir con el especificado en la opción **-cacheName** utilizada en el mandato admincache para crear la memoria caché. El nombre no debe superar los 53 caracteres.
none	Inhabilitación explícita de la compartición de clase. Se puede añadir al final de una línea de mandatos para inhabilitar la función de compartir datos de clases. Esta subopción altera temporalmente los argumentos para compartir clases que aparecen al principio en la línea de mandatos.
nonfatal	Inicie siempre la máquina virtual Java, independientemente de los errores o los avisos. Permite que la JVM se inicie incluso cuando se producen errores al compartir datos de clases. El comportamiento habitual de la JVM es negarse a reiniciarse si falla la función de compartir datos de clases. Si se selecciona **nonfatal** y la memoria caché de clases compartidas no consigue inicializarse, la memoria virtual Java intenta conectarse a la memoria caché en modalidad de sólo lectura. Si este intento falla, la máquina virtual Java se inicia sin el compartimiento de datos de clases.
silent	Suprima todos los mensajes de salida. Inactiva todos los mensajes de clases compartidos, incluidos los mensajes de error. Se visualizan mensajes de error irrecuperables que impiden que la memoria virtual Java se inicialice.
verbose	Habilita la salida detallada, que proporciona estado global sobre la memoria caché de la clase compartida y mensajes de error más detallados.

Tabla 11. Subopciones disponibles cuando se ejecuta una aplicación en modalidad de tiempo real (continuación)

Opción	Significado
`verboseAOT`	Habilita la salida detallada cuando se encuentra código compilado mediante AOT en la memoria caché, por ejemplo durante las solicitudes de carga del método AOT.
`verboseHelper`	Habilita la salida en modalidad detallada para la API del ayudante Java. Esta salida le muestra cómo usa el cargador de clases la API del ayudante.
`verboseIO`	Habilita la salida detallada de las solicitudes de carga de clase. Esta opción proporciona una salida detallada sobre la actividad de E/S, con información sobre las clases encontradas.

Para asegurarse de que estas opciones sean correctas, use la opción **-printvmargs** con admincache (consulte **-printvmargs** para obtener más información). La opción **nonfatal** no es adecuada para el uso general, porque fuerza a la JVM a omitir los avisos y errores sobre la memoria caché de clase compartida. La opción **none** inhabilita explícitamente la compartición de clase y es equivalente a omitir la opción **-Xshareclasses** en la línea de mandatos.

Para obtener más detalles sobre las subopciones de **-Xshareclasses** consulte Opciones de línea de mandatos de compartimiento de datos de clase.

Utilización del recopilador de basura Metronome

El recopilador de basura Metronome sustituye al recopilador de basura estándar en WebSphere Real Time for RT Linux.

Control de la utilización del procesador

Puede limitar la cantidad de energía de proceso disponible para el recopilador de basura Metronome.

Puede controlar la recogida de basura con el recopilador de basura Metronome utilizando la opción **-Xgc:targetUtilization=N** para limitar la cantidad de CPU utilizada por el recopilador de basura.

Por ejemplo:

```
java -Xrealtime -Xgc:targetUtilization=80 su_aplicación
```

El ejemplo especifica que la aplicación se ejecuta durante el 80% de cada 60 milisegundos. El 20% del tiempo se utiliza para la recogida de basura. El recopilador de basura Metronome garantiza la utilización de los niveles siempre que se le hayan otorgado recursos suficientes. La recogida de basura empieza cuando el espacio libre del almacenamiento dinámico pasa a estar por debajo de un umbral determinado de forma dinámica.

Ajuste de Recopilador de basura Metronome

Puede ajustar el entorno en tiempo real controlando la cantidad de memoria que utiliza la aplicación. Por ejemplo, utilice las opciones **-Xmx**, **-Xgc:immortalMemorySize=size**, **-Xgc:scopedMemoryMaximumSize=size** y **-Xgc:targetUtilization=N**.

- Utilice la opción **-Xmx** para limitar el tamaño del almacenamiento dinámico.

  El valor seleccionado se utiliza como límite superior del tamaño de almacenamiento dinámico y, por lo tanto, refleja el uso probable a lo largo del tiempo. Seleccionar un valor demasiado bajo aumenta la frecuencia de la recogida de basura y lleva a un menor rendimiento global, aunque reduce la ocupación de memoria. Para obtener un buen rendimiento en tiempo real, evite la paginación. Es normal asegurarse de que la ocupación de todos los procesos en ejecución de una máquina no supere el tamaño de la memoria física.
- Utilice la opción **-Xgc:immortalMemorySize=size** para controlar el tamaño del área de memoria inmortal.

  Debe analizar cuidadosamente el uso de la memoria inmortal. La aplicación "ideal" utiliza la memoria inmortal durante el inicio, pero luego deja de utilizarla. Si continúa la asignación de objetos inmortales, la aplicación se puede seguir ejecutando hasta que se agota la memoria inmortal. La utilización actual se puede obtener añadiendo:

  ```
  long used = ImmortalMemory.instance().memoryConsumed();
  ```

  a su código.
- Utilice la opción **-Xgc:scopedMemoryMaximumSize=size** para asegurarse de que las aplicaciones no necesiten una cantidad excesiva de memoria de ámbito. Utilice esta opción para el diagnóstico, no para el ajuste.
- Defina la opción **-Xgc:targetUtilization=N** para asegurarse de que incluso en las peores condiciones (tasa máxima de asignación de objetos de almacenamiento dinámico), el recopilador de basura pueda recoger basura más rápido de lo que la aplicación la genera.

  Normalmente, el valor predeterminado es suficiente, pero el rendimiento de la aplicación puede mejorar aumentando la utilización hasta el punto en el que el recopilador pueda recoger basura algo más rápido de lo que la crea la aplicación.
- Utilice la opción **-Xgcthreads <n>** para crear hebras adicionales para ejecutar la recogida de basura en paralelo.

  El valor predeterminado es utilizar una hebra. Si su carga de trabajo tiene una alta tasa de generación de basura, y se ejecuta en un multiprocesador simétrico con ciclos de CPU disponibles, el rendimiento podría beneficiarse de definir este parámetro en >1.

  Nota: Definir este parámetro en un valor demasiado alto puede tener un efecto negativo en el rendimiento.

Limitaciones del recopilador de basura Metronome

En este tema se describen las limitaciones o problemas conocidos que afectan a la política de recopilación de basura de Metronome.

Soporte de AESNI en plataformas x86

Actualmente, con la política de recopilación de basura de Metronome, no es posible el aprovechamiento del software de las instrucciones AESNI en arquitecturas de x86.

Largos tiempos de pausa durante la recogida de basura

Bajo excepcionales circunstancias, puede sufrir pausas más largas de lo esperado durante la recopilación de basura.Durante la recogida de basura, se utiliza un

proceso de exploración de la raíz. El recopilador de basura recorre el almacenamiento dinámico, empezando por las referencias activas conocidas. Entre estas referencias se incluyen las siguientes:

- Variables de referencias activas en las pilas de llamadas de hebra activas.
- Referencias estáticas.
- Todas las referencias a objetos de memoria inmortales y de ámbito.

Para encontrar todas las referencias a objetos activas en la pila de hebras de una aplicación, el recopilador de basura explora todos los marcos de la pila de esta pila de llamadas a la hebra. Cada pila de hebras activas se explora en un paso que no se puede interrumpir. Por tanto, la exploración debe realizarse en una pausa del recopilador de basura individual.

El efecto es que el rendimiento del sistema podría ser peor de lo esperado si tiene algunas hebras con pilas muy profundas, porque la recogida de basura ampliada se detiene al principio de cada ciclo de recogida.

La memoria inmortal se procesa de forma incremental. Todas las áreas de memoria de ámbito restantes se procesan en un paso atómico que no se puede interrumpir. Por lo tanto, una utilización significativa de las áreas de memoria de ámbito podría llevar a un empeoramiento del rendimiento del sistema con respecto a lo esperado, debido a que la recogida de basura ampliada se detiene cuando la exploración raíz está procesando la memoria de ámbito.

Capítulo 6. Desarrollo de aplicaciones

Información importante sobre la escritura de aplicaciones en tiempo real, incluyendo muestras de código.

Escritura de aplicaciones Java para aprovechar el tiempo real

En estos ejemplos se describe cómo aprovechar el entorno en tiempo real. Van desde el ejemplo más sencillo, la ejecución de una aplicación Java en tiempo real sin modificaciones en el código, hasta un proceso más complejo de planificación y escritura de hebras en tiempo real de almacenamiento no dinámico . Se ofrecen motivos para ayudarle a decidir qué metodología resulta más adecuada para sus aplicaciones.

Introducción a la escritura de aplicaciones en tiempo real

No es necesario que cree aplicaciones de tiempo real no de pila para aprovechar las características de la tecnología en tiempo real. Podrá utilizar algunas de sus ventajas con muy pocos cambios sobre el código existente.

Para los programadores de aplicaciones, estos son los pasos que puede seguir para utilizar WebSphere Real Time for RT Linux:

1. Puede ejecutar una aplicación Java estándar en una JVM en tiempo real para conseguir las ventajas de recogida de basura de Metronome y obtener una importante mejora en la capacidad de predicción del tiempo de ejecución de su aplicación.
2. Añada la opción **-Xnojit** después de precompilar su código para utilizar el compilador AOT (ahead-of-time). Consulte el apartado "Almacenamiento de archivos .jar precompilados en una memoria caché de clase compartida" en la página 59.
3. Sustituya java.lang.Thread por javax.realtime.RealtimeThread en su aplicación. Es posible que vea una pequeña mejora en comparación con la opción AOT.

   La ventaja principal de utilizar hebras de tiempo real es la capacidad de controlar la prioridad que se da a cada una de las hebras. Las hebras en tiempo real también pueden ser periódicas. Para aprovechar estas ventajas, debe estar preparado para realizar cambios en la propia aplicación.
4. Planifique y escriba una aplicación específica para utilizar hebras de tiempo real y manejadores de sucesos asíncronos para tratar con temporizadores o sucesos externos. Considere estos tres factores:
   - Planificación de la prioridad que se asigna a hebras de tiempo real
   - Decisión de las áreas de memoria que se utilizarán para contener los objetos
   - Comunicación con los manejadores de sucesos
5. Planifique y escriba una aplicación específica para utilizar hebras en tiempo real de almacenamiento no dinámico . Las hebras en tiempo real de almacenamiento no dinámico son ampliaciones de hebras de tiempo real y tiene que tener en cuenta la prioridad asignada al área de memoria. Por lo general,

solo debe realizar este paso si la aplicación debe gestionar los sucesos en un tiempo comparable al de la pausa del recopilador de basura (submilisegundo). Tenga en cuenta la complejidad del desarrollo con hebras en tiempo real de almacenamiento no dinámico .

En el apartado Figura 5 se muestran los pasos descritos anteriormente.

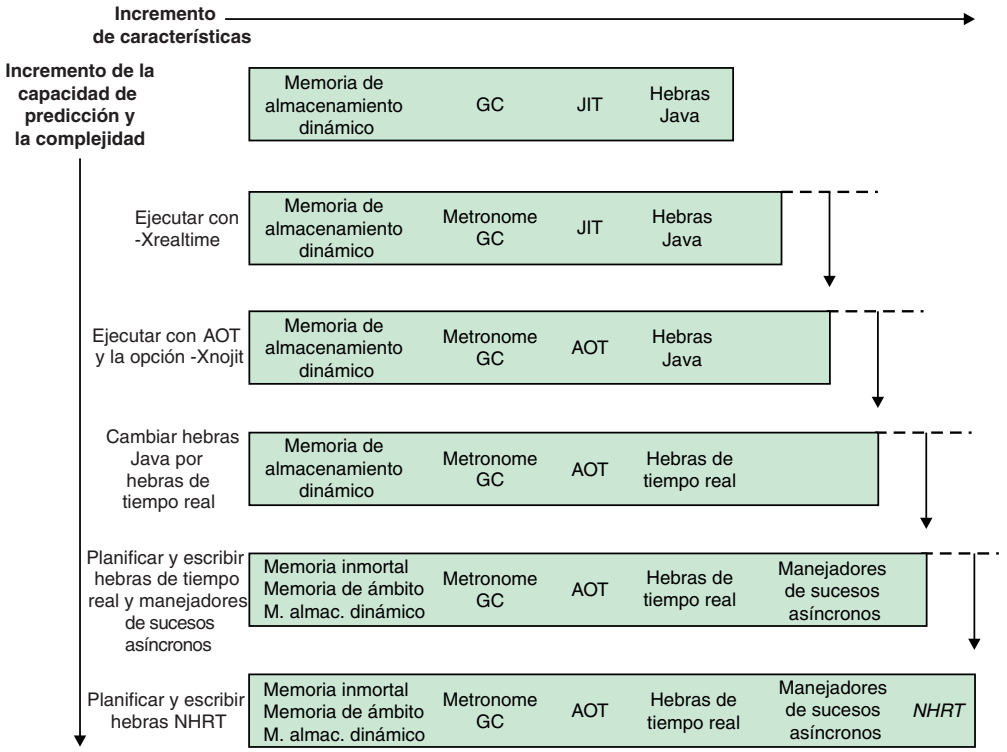

Figura 5. Comparación de las características de RTSJ con mayor capacidad de predicción.

Planificación de su aplicación WebSphere Real Time for RT Linux

Cuando se esté preparando para escribir aplicaciones en tiempo real de Java, debe pensar si utilizar hebras Java,hebras de tiempo real o hebras en tiempo real de almacenamiento no dinámico . Además, puede decidir qué área de memoria utilizarán sus hebras.

Acerca de esta tarea

Al planificar su aplicación, estos pasos describen las decisiones que debe tomar:

Procedimiento

1. Identifique sus tareas.
2. Decida los periodos de temporización:
   - Para respuestas de más de 10 ms, seleccione hebras de Java, aprovechando el recopilador de basura Metronome.

     Estas hebras utilizan solo la memoria de almacenamiento dinámico para el almacenamiento. Su desventaja es que la recogida de basura interrumpe su

aplicación, pero, dado que está controlada por Recopilador de basura Metronome, la longitud y la temporización de las interrupciones es predecible.

- Para respuestas de menos de 10 ms, seleccione hebras de tiempo real.

  Las hebras en tiempo real se pueden colocar en la memoria de almacenamiento dinámico, de ámbito o inmortal. Las ventajas de utilizar hebras de tiempo real son las siguientes:
  - Se pueden ejecutar con una prioridad más alta que las hebras de Java estándar.
  - La recogida de basura está bajo el control del recopilador de basura Metronome. Sin embargo, el recopilador de basura se ejecuta con una prioridad más alta que la prioridad más alta de una hebra de tiempo real, e interrumpe la ejecución del programa.
- Para respuestas de menos de un milisegundo, seleccione hebras en tiempo real de almacenamiento no dinámico .

  La prioridad de hebras en tiempo real de almacenamiento no dinámico se puede definir como más alta que la recogida de basura y, por lo tanto, no se ve significativamente interrumpida por el Metronome. Solo la hebra de alarma Metronome se ejecuta con la máxima prioridad y utiliza una parte muy pequeña de la CPU.

3. Determine si su aplicación necesita manejadores de sucesos asíncronos. Este requisito depende de la estructura de su programa.
   - Para un tiempo de respuesta de menos de 10 ms, seleccione hebras de tiempo real.
   - Para un tiempo de respuesta de menos de 1 milisegundo, seleccione hebras en tiempo real de almacenamiento no dinámico
4. Determine las prioridades de hebras. Por lo general, cuando más breve sea el período de tiempo, mayor será la prioridad.
5. Decida las características de la memoria.
   - Si una tarea tiene una tasa de asignación alta o variable, que puede bloquear al recopilador de basura, puede imponer un límite de tasa (utilizando MemoryParameters) o bien considerar la asignación en un área de memoria de ámbito.
   - Si una tarea genera muchos datos temporales durante un cálculo, considere utilizar un área de memoria de ámbito.
   - Si una tarea genera datos durante el inicio que son necesarios durante el tiempo de vida de la JVM, considere utilizar la memoria inmortal. Intente evitar utilizar la memoria inmortal en aquellos casos en los que se continuarán creando objetos durante el tiempo de vida de la JVM.
   - Si las tareas necesitan comunicarse, especialmente si hay una ejecutándose en una hebra en tiempo real de almacenamiento no dinámico, considere utilizar un área de memoria de ámbito para la comunicación.
   - Si una tarea se ejecuta bajo una hebra en tiempo real de almacenamiento no dinámico, puede crear un área de memoria de ámbito, como por ejemplo LTMemory, donde contener la hebra no de pila, los parámetros de tiempo de ejecución y, posiblemente, las colas sin tiempo de espera que se utilizan para la comunicación con la tarea. El objeto LTMemory se debe crear en memoria inmortal o en otro ámbito para evitar los errores cuando la hebra no de pila trate de hacer referencia a él.
6. Modifique las opciones de tiempo de ejecución para mejorar el rendimiento de su aplicación, cuando haya decidido la estructura y el contenido de su aplicación. En los pasos siguientes se describe cómo hacerlo:

a. Durante las pruebas iniciales de su aplicación, defina una generosa cantidad de espacio en la memoria de almacenamiento dinámico, de ámbito e inmortal utilizando las opciones **-Xmx**, **-Xgc:immortalMemorySize=size** y **-Xgc:scopedMemoryMaximumSize=size**.

   Nota: Con el recopilador de basura Metronome, los tamaños de almacenamiento dinámico iniciales y máximos deben ser iguales, porque el recopilador de basura Metronome no aumenta el tamaño del almacenamiento dinámico. Hacer crecer el almacenamiento dinámico es una operación no determinante.

b. Utilice la opción **-verbose:gc** para determinar la memoria utilizada.

c. Modifique la opción **-Xgc:targetUtilization** para permitir el tiempo suficiente como para que se produzca la recogida de basura. El valor predeterminado es 70% y este porcentaje suele ser el adecuado para la mayoría de las aplicaciones. Asegúrese de que la tasa de recogida de basura sea ligeramente superior que la tasa de asignación.

d. Defina un tamaño realista para la memoria de almacenamiento dinámico utilizando la opción **-Xmx**.

Modificación de aplicaciones Java

Para escribir código que utilice las características de tiempo real de Java, utilice `javax.realtime.RealtimeThread` para sustituir `java.lang.Thread` para las hebras.

Antes de empezar

Este ejemplo se basa en la clase JavaRadar.java que se encuentra en el archivo `demo/realtime/sample_application.zip`.

Acerca de esta tarea

El modelo de programación de las hebras en tiempo real es parecido al de las aplicaciones Java estándar. Sin embargo, esta manera, más bien improvisada, de añadir hebras de tiempo real a sus programas no saca el máximo partido de las características de WebSphere Real Time for RT Linux. Para hacerlo, debe modificar las hebras de manera que tengan una prioridad asociada y reflexionar también sobre las áreas de memoria que se utilizarán.

Si solo cambia las clases de las hebras, obtendrá un pequeño beneficio para su aplicación porque la prioridad predeterminada de hebras de tiempo real es mayor que la de las hebras estándar de Java.

Para cambiar JavaRadar a RealtimeThread, cambie la clase que amplía de `Thread` a `RealtimeThread`.

Sustitución de `java.lang.Thread` por `javax.realtime.RealtimeThread`

La clase JavaRadar de la aplicación de ejemplo amplía `java.lang.Thread`. Por ejemplo:

```
public class JavaRadar extends Thread implements Radar
```

Para convertir esta hebra de Java en una hebra de tiempo real, redefina esta definición de clase como se indica a continuación:

```
public class RTJavaRadar extends RealtimeThread implements Radar
```

Escritura de hebras de tiempo real

Hasta ahora, solo ha modificado una aplicación, ha llegado el momento de escribir código. Puede escribir aplicaciones que utilicen hebras de tiempo real para sacar provecho de las áreas de memoria y niveles de prioridad en tiempo real.

Antes de empezar

Este ejemplo se basa en las clases JavaRadar.java, RTJavaRadar.java y RTJavaControlLauncher.java que se encuentran en el archivo `demo/realtime/sample_application.zip`.

En este ejemplo se le muestra cómo utilizar la memoria inmortal con el mismo ejemplo que se describe en el apartado "Modificación de aplicaciones Java" en la página 82.

Acerca de esta tarea

El modelo de programación para hebras de tiempo real es parecido que el de las aplicaciones Java estándar.

Las ventajas de utilizar hebras de tiempo real son las siguientes:

- Soporte completo para las prioridades de hebras a nivel de sistema operativo en hebras de tiempo real.
- Utilización de áreas de memoria de ámbito o inmortal.
  - Con la memoria de ámbito, puede controlar de forma explícita la desasignación de memoria sin que ello afecte a la recogida de basura.
  - Con hebras en tiempo real de almacenamiento no dinámico , puede utilizar la memoria inmortal para evitar las pausas en la recogida de basura.
  - Los hebras de tiempo real que hacen referencia a objetos en el almacenamiento dinámico están sujetos a la recogida de basura, al igual que los hebras de tiempo real almacenados en la memoria de almacenamiento dinámico.
  - Las hebras en tiempo real de almacenamiento no dinámico no pueden hacer referencia a objetos de la memoria de almacenamiento dinámico y, por lo tanto, no se ven afectados por la recogida de basura.

En Tabla 12 en la página 84, las prioridades se asignan en base a que SimulationThread tiene la prioridad más alta porque representa sucesos externos y no se debe permitir que sea anulado por ningún elemento del programa. RadarThread tiene que responder rápidamente a los pings del controlador. Cuanto más rápida sea la respuesta, más precisa será la medida de la altura del dispositivo de aterrizaje lunar. ListenThread tiene que responder también rápidamente a los mandatos del controlador, pero toma el segundo lugar de RadarThread.

Estas tres hebras se encuentran en la memoria de ámbito porque la simulación se ejecuta como un servidor. Una vez que el servidor ha ejecutado una simulación, puede salir del área de la memoria de ámbito y, a continuación, volver a entrar en ella para esperar otra ejecución de la simulación. El servidor utiliza la memoria de ámbito para que se pueda restablecer automáticamente.

RTJavaRadarthread tiene la prioridad más alta de las hebras del controlador porque es más sensible a la temporización debido a que utiliza este tiempo para derivar la altura. Es inmortal porque se ejecuta como una NHRT, el controlador se ejecuta solo una vez y la memoria se publica cuando la JVM sale.

Para RTJavaControlThread y RTJavaEventThread, las restricciones temporales no son tan importantes, por lo que utilizar la memoria de almacenamiento dinámico es aceptable.

RTLoadThread no realiza ninguna función útil para el dispositivo de aterrizaje lunar. RTLoadThread muestra que se puede realizar una asignación y desasignación de memoria significativa con una prioridad más baja que la de otras hebras, sin afectar al rendimiento de las hebras de prioridad más alta.

Tabla 12. Relación de las áreas de hebras en la aplicación de ejemplo

Memoria	Hebra	Prioridad
Ámbito	demo.sim.SimulationThread	38
	demo.sim.RadarThread	37
	demo.sim.SimulationThread.ListenThread	36
Inmortal	demo.controller.RTJavaRadarThread	15
Almacenamiento dinámico	demo.controller.RTJavaControlThread	14
	demo.controller.RTJavaEventThread	13
Ámbito y almacenamiento dinámico	demo.controller.RTLoadThread	12

Ejemplos

Este código de demo.sim.SimulationThread muestra dónde se ha definido la prioridad de 38. **1** Esta línea de código recupera la prioridad máxima disponible en la JVM.

```
super(null, area);

// Defina la prioridad por separado, ya que se está utilizando "this".
// Tenga en cuenta que PriorityScheduler.MAX_PRIORITY está en desuso.
this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
        .getMaxPriority(this)));     1
```

Este código de demo.sim.SimLauncher muestra dónde se ha definido la memoria de ámbito. **2** muestra la asignación de LTMemory, que es una memoria de ámbito que asigna memoria en tiempo lineal.

```
final IndirectRef<MemoryArea> myMemRef = new IndirectRef<MemoryArea>();

/*
 * The LTMemory object has to be created in a memory area that the
 * NHRTs can access.
 */
ImmortalMemory.instance().enter(new Runnable() {
    public void run() {
        myMemRef.ref = new LTMemory(10000000);  2
    }
});

final MemoryArea simMemArea = myMemRef.ref;
```

El objeto ScopedMemoryArea al que hace referencia `simMemArea` se está asignando en la memoria inmortal, ya que NHRT debe ser capaz de hacer referencia al objeto que representa ScopedMemoryArea. La asignación en el almacenamiento dinámico

provoca que el constructor NHRT genere una excepción IllegalArgumentException, porque el argumento de su área de memoria estaba en el almacenamiento dinámico.

```
simMemArea.enter(new Runnable() {
    public void run() {
        try {
            CommsControl commsControl = new CommsControl();
```

Este código de demo.controller.RTJavaControlLauncher muestra dónde se ha definido y utilizado la memoria inmortal en RTJavaRadar. Dado que RTJavaRadar se ejecuta solo una vez durante toda la duración de la JVM del controlador, se ha diseñado para asignar la memoria solo durante el inicio, se puede ejecutar de forma segura en la memoria inmortal. El diseño de la aplicación es una ventaja porque el controlador puede acceder a los métodos RTJavaRadar sin tener que entrar primero en el área de memoria de ámbito. Entrar en el área de memoria de ámbito es difícil porque el Controlador se ha escrito en código Java ordinario, así como en Java de tiempo real.

```
final RadarPort radarPort = commsControl.getRadarPort();
EventPort eventPort = commsControl.getEventPort();

final IndirectRef<RTJavaRadar> radarRef = new IndirectRef<RTJavaRadar>();

// Crear RTJavaRadar en Inmortal, ya que es una NHRT.
// Si estuviese en el ámbito, su interacción con las otras hebras sería
// más compleja.
ImmortalMemory.instance().enter(new Runnable() {
    public void run() {
        // Versión del radar en tiempo real.
        radarRef.ref = new RTJavaRadar(radarPort, ImmortalMemory
                .instance());
    }
});

RTJavaRadar radarJava = radarRef.ref;
```

Escritura de manejadores de sucesos asíncronos

Los manejadores de sucesos asíncronos reaccionan ante sucesos de temporizador o sucesos que se producen fuera de una hebra; por ejemplo, la entrada desde la interfaz de una aplicación. En sistemas de tiempo real, estos sucesos deben responder en las fechas límites definidas para su aplicación.

Antes de empezar

Este ejemplo se basa en las clases RTJavaEventThread.java y RTJavaControlLauncher.java que se encuentran en el archivo `demo/realtime/sample_application.zip`.

Acerca de esta tarea

En la aplicación de muestra, la hebra de suceso espera los sucesos de la simulación que señalan un bloqueo o una finalización. En la versión en tiempo real de esta hebra, se utiliza el mecanismo AsyncEvent. Estos sucesos se utilizan para imprimir el mensaje de estado correspondiente y para que el controlador salga.

RTJavaEventThread tiene dos sucesos asíncronos definidos. Ninguno de ellos tiene parámetros.

```
public class RTJavaEventThread extends RealtimeThread {

    private AsyncEvent landEvent = new AsyncEvent(),   Land
            crashEvent = new AsyncEvent();  Crash
```

Estos sucesos crean y registran dos manejadores de sucesos asíncronos:

```
    /**
     * Pass a runnable object that will be fired when the land event occurs.
     *
     * @param runnable code to be executed when land event is triggered.
     */
    public void addLandHandler(Runnable runnable) {
        AsyncEventHandler handler = new AsyncEventHandler(runnable);
        this.landEvent.addHandler(handler);
    }

    /**
     * Pass a runnable object that will be run when the crash event occurs.
     *
     * @param runnable code to be executed when crash event is triggered.
     */
    public void addCrashHandler(Runnable runnable) {
        AsyncEventHandler handler = new AsyncEventHandler(runnable);
        this.crashEvent.addHandler(handler);
    }
```

Cuando se reciben mensajes de bloqueo o finalización, se activa el correspondiente manejador de sucesos asíncrono, lo que provoca que se liberen los objetos Runnable.

```
                tag = this.eventPort.receiveTag();

                switch (tag) {
                case EventPort.E_CRSH:
                    // Choque
                    this.crashEvent.fire();
                    this.running = false;
                    break;
                case EventPort.E_LAND:
                    // Aterrizaje
                    this.landEvent.fire();
                    this.running = false;
                    break;
                }
```

Resultados

RTJavaControlLauncher.java contiene invocaciones a los métodos addLandHandler y addCrashHandler. Los objetos Runnable pasados hacen que se imprima un mensaje en la consola y que se detenga la hebra de control cuando se activan sus manejadores de sucesos asíncronos asociados. Consulte RTJavaEventThread.java para ver en qué punto se activan.

```
            // AEH ejecutable para el manejador del aterrizaje.
            javaEventThread.addLandHandler(new Runnable() {
                public void run() {
                    System.out.println("LAND!");
                }
            });

            // AEH ejecutable para el manejador del choque.
            javaEventThread.addCrashHandler(new Runnable() {
```

```
            public void run() {
                System.out.println("CRASH!");
            }
        });
```

Escritura de hebras NHRT

Para añadir hebras de hebras en tiempo real de almacenamiento no dinámico (NHRT) a una aplicación Java, utilice esta guía de aprendizaje para desarrollar o modificar sus propios programas.

Antes de empezar

Este ejemplo se basa en las clases SimulationThread.java y SimLauncher.java que se encuentran en el archivo `demo/realtime/sample_application.zip`.

Acerca de esta tarea

La clase demo.sim.SimulationThread forma parte de la simulación en la aplicación de muestra. Pretende actuar como sustituta del mundo real y, por lo tanto, probablemente se ejecutará sin interrupción desde el resto del sistema. La hebra se crea como NoHeapRealtimeThread con la prioridad más alta disponible, a fin de garantizar que la hebra no se vea interrumpida por la recogida de basura o por otras hebras del sistema.

En SimulationThread, el constructor siguiente llama al superconstructor "NoHeapRealtimeThread(SchedulingParameters scheduling, MemoryArea area)", pero antes define sus SchedulingParameters y ReleaseParameters de forma independiente:

```
public SimulationThread(MemoryArea area, ControlPort controlPort,
            EventPort eventPort, RadarThread radarThread) {

        super(null, area);

        // Defina la prioridad por separado, ya que se está utilizando "this".
        // Tenga en cuenta que PriorityScheduler.MAX_PRIORITY está en desuso.
        this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
                .getMaxPriority(this)));

        ReleaseParameters releaseParms = new PeriodicParameters(null,
                new RelativeTime(period, 0)); // Ciclo de 20 ms (50 Hz)
        this.setReleaseParameters(releaseParms);

        // Se recomienda identificar todas las hebras
        this.setName("SimulationThread");

        this.controlPort = controlPort;
        this.eventPort = eventPort;
        this.radarThread = radarThread;
    }
```

Las otras hebras activas de la simulación se crean también como hebras en tiempo real de almacenamiento no dinámico (NHRT), pero con una prioridad algo más baja. Consulte "Escritura de hebras de tiempo real" en la página 83 para ver la organización de las prioridades.

La simulación tiene la opción de ejecutarse de forma indefinida, de manera que después de que una simulación termine, vuelva a empezar. Dado que la simulación se compone de NHRT, puede elegir ScopedMemory o ImmortalMemory. La aplicación de ejemplo utiliza ScopedMemory para la

simulación porque es adecuada para salir de la ScopeMemoryArea asignada cuando terminó la simulación y, a continuación, volver a entrar en ella para esperar la siguiente ejecución. En este caso, no se lleva ningún estado de una ejecución a la siguiente.

La mayoría de las clases son seguras para NHRT; sin embargo, la mayoría de las clases se pueden ejecutar de una manera que no es segura para NHRT. Por ejemplo, si los DatagramSockets se guardasen en la memoria inmortal, o en un área de memoria de ámbito externo, se podrían producir problemas porque no se han diseñado para abarcar áreas de memoria. La aplicación de muestra utiliza solo un área ScopedMemory para evitar estos problemas.

Asignación de memoria en RTSJ

En RTSJ, puede asignar un objeto en un área de memoria específica de varias formas distintas, y no siempre resulta evidente cuál de ellas elegir en un momento determinado.

Cada metodología tiene algunas características, que varían entre implementaciones de RTSJ, y marcan la diferencia en el rendimiento o en la posible ocupación de memoria. En esta sección se resumen las opciones disponibles y se sugieren aquellas ocasiones en las que puede existir una opción más adecuada para asignar un objeto.

Inicializador estático

La forma más simple de asignar un objeto en el área de memoria inmortal consiste en asignarlo en un inicializador estático. La ventaja es que no tiene que tratar con los problemas de cambiar el contexto de la memoria, pero las circunstancias en las que este es el patrón adecuado son bastante limitadas. Esta metodología resulta eficaz en el sentido de que la memoria inmortal consumida se limita a la necesaria para el propio objeto.

MemoryArea.newInstance(Class c)

Este método es el elegido si una hebra se encuentra en el contexto de la memoria y desea asignar un objeto en otra área, que ya se debe encontrar en la pila de ámbito de la hebra. La ventaja es que solo necesita acceder a la clase cuya instancia se va a crear, pero el método newInstance debe crear un constructor adecuado. Este patrón es el más adecuado si los objetos de una clase dada se deben asignar con poca frecuencia, pero en caso contrario tiende a mostrar un elevado uso de la memoria.

MemoryArea.newInstance(Constructor c, Object[] args)

De nuevo, un método sencillo si la hebra está en un contexto de memoria y desea asignar un objeto en otro contexto, que se debe encontrar ya en la pila de ámbito de la hebra. En este caso, debe pasar un constructor y algunos argumentos, y se presupone la responsabilidad de garantizar que el constructor es válido en el contexto de memoria actual. Dado que el método newInstance no tiene que crear un constructor, el uso de la memoria es más bajo que newInstance(Class c) y, por lo tanto, este patrón resulta más adecuado si los objetos se van a asignar con más frecuencia y desea pagar el precio de asignar el constructor por adelantado y almacenarlo en otro sitio, como ImmortalMemory.

MemoryArea.enter(Runnable r) seguido de la nueva <clase>()

Este método convierte el parámetro MemoryArea dado en predeterminado para las asignaciones, y elimina la necesidad de reflejo y los objetos de constructor complementarios. Por lo tanto, es el más adecuado si se van a crear muchos objetos porque no se produce ninguna utilización de la memoria en el propio objeto. Este método solo funciona si el área indicada no está ya activa en la pila de ámbito de alguna hebra. El requisito para crear un área de memoria Runnable hace que este método sea más complejo que utilizar newInstance, porque normalmente tendrá que pasar los parámetros en el área Runnable o mediante los campos estático o de instancia.

MemoryArea.executeInArea(Runnable r) seguido de la nueva <clase>()

De nuevo, este método convierte el parámetro MemoryArea dado en predeterminado para las asignaciones, y elimina la necesidad de reflejo y los objetos de constructor complementarios. Por lo tanto, es el más adecuado si se van a crear muchos objetos porque no se produce ninguna utilización de la memoria en el propio objeto. Puede utilizar esta metodología si el área indicada ya se encuentra en la pila de ámbito de la hebra actual y es, por lo tanto, más flexible que MemoryArea.enter. El requisito para crear un Ejecutable hace que este método sea más complejo que utilizar newInstance, porque normalmente tendrá que pasar los parámetros en el área Runnable o mediante los campos estático o de instancia.

Class.newInstance()

Esta metodología crea la nueva instancia en el área de memoria actual y, por lo tanto, se debe utilizar con MemoryArea.enter o executeInArea. No se produce ninguna utilización de memoria adicional aparte del objeto en sí mismo.

Utilización del temporizador de alta resolución

El reloj en tiempo real ofrece más precisión que los relojes asociados a la JVM estándar.

Antes de empezar

Este ejemplo se basa en la clase RTJavaRadar.java que se encuentra en el archivo `demo/realtime/sample_application.zip`.

Acerca de esta tarea

El Java común tiene una capacidad limitada para tratar con relojes y temporizadores. Especificación de tiempo real para Java permite que se especifiquen horas absolutas con precisión de nanosegundos y una magnitud suficiente para la hora real. javax.realtime.HighResolutionTime y sus subclases se utilizan para representar la hora con dos componentes, milisegundos y nanosegundos.

WebSphere Real Time for RT Linux utiliza el soporte del sistema operativo subyacente para proporcionar una hora de alta resolución. Los kernels de Linux actuales proporcionan un reloj con, como mucho, una precisión garantizada de 4 milisegundos. Los parches de Linux proporcionados con WebSphere Real Time for RT Linux ofrecen un reloj con una precisión de casi 1 microsegundo.

La clase RTJavaRadar muestra el uso del temporizador de alta resolución:

- 1 obtiene la hora real.
- 2 obtiene la hora absoluta actual.
- 3 obtiene el componente de nanosegundos de la hora. La precisión del reloj de tiempo real implica que utilizar nanosegundos es razonable.
- 4 obtiene la hora antes y después del ping.
- 5 devuelve la velocidad de descenso de aterrizaje.
- 6 hace que la hebra espere 5 milisegundos antes de realizar otra iteración.

```
public void run() {
       // Los objetos siguientes se crean por adelantado y se reutilizan en cada
       // iteración.
       Clock rtClock = Clock.getRealtimeClock();          1
       AbsoluteTime time = rtClock.getTime();             2

       try {
           double height = 0.0, lastheight;
           long millis = time.getMilliseconds(), lastmillis;
           long nanos = time.getNanoseconds(), lastnanos;     3

           while (this.running) {

               lastmillis = millis;
               lastnanos = nanos;
               lastheight = height;

               // En lugar de utilizar el formulario time = rtClock.getTime(), este
               // método
               // sustituye los valores de un objeto AbsoluteTime preexistente.
               rtClock.getTime(time);                  4
               millis = time.getMilliseconds();
               nanos = time.getNanoseconds();

               // Medimos el tiempo que se tarda en enviar el ping y recibir el
               // pong.
               this.radarPort.ping();

               rtClock.getTime(time);                  4

               height = (time.getMilliseconds() - millis)
                       / demo.sim.RadarThread.timeScale;
               height += ((time.getNanoseconds() - nanos) / 1.0e6)       5
                       / demo.sim.RadarThread.timeScale;

               double difference = ((double) (millis - lastmillis)) / 1.0e3
                       + ((double) (nanos - lastnanos)) / 1.0e9;
               double speed = (height - lastheight) / difference;

               this.myHeight = height;
               this.mySpeed = speed;

               try {
                   sleep(5);                      6
               } catch (InterruptedException e) {
                   // Esto no es importante.
               }
           }
```

El código precedente se puede comparar con el siguiente código JVM estándar en la clase JavaRadar:

```
public void run() {
       try {
           double height = 0.0, lastheight;
```

```
long nanos = System.nanoTime(), lastnanos;
while (this.running) {
    /* Set the height every x milliseconds */
    Thread.sleep(5);
    lastnanos = nanos;
    lastheight = height;

    nanos = System.nanoTime();

    this.radarPort.ping();

    // La escala de tiempo es de unidades de altura por milisegundo
    height = ((System.nanoTime() - nanos) / 1.0e6)
            / demo.sim.RadarThread.timeScale;

    double speed = (height - lastheight)
            / (((double) (nanos - lastnanos)) / 1.0e9);

    this.myHeight = height;
    this.mySpeed = speed;
}
```

Aplicación de muestra

La aplicación de muestra utiliza una serie de ejemplos para demostrar las características de WebSphere Real Time for RT Linux que se pueden utilizar para mejorar las características de tiempo real de los programas Java.

Los archivos de origen de la aplicación de muestra se encuentran en el archivo `demo/realtime/sample_application.zip`.

El ejemplo se compone de dos partes principales:

- **Una simulación**, un ejemplo sencillo de un dispositivo de aterrizaje lunar. La posición del dispositivo de aterrizaje viene definida por su altura sobre el suelo, derivada de impulsos cronometrados, y la distancia horizontal con respecto al área de aterrizaje. Consulte el apartado Figura 6 en la página 92.

  La clase de simulación se graba utilizando hebras en tiempo real de almacenamiento no dinámico (NHRT) y no se vuelve a modificar en esta documentación.
- **Un controlador** que envía mandatos a la simulación. El controlador envía pings de radar para juzgar la altura del dispositivo de aterrizaje y controlar la velocidad de descenso de dicho dispositivo según esta información. El controlador recibe también una secuencia de información del dispositivo de aterrizaje, por ejemplo, la distancia del dispositivo de aterrizaje con respecto al área de aterrizaje.

  El controlador se escribe inicialmente en Java estándar. En “Modificación de aplicaciones Java” en la página 82, se desarrolla en un programa Java de tiempo real

Según el resultado del aterrizaje, se envía al controlador uno de estos mensajes: choque o aterrizaje.

Con la aplicación de ejemplo, puede realizar estas operaciones:

- Ejecutar la simulación y el controlador juntos para mostrar una combinación de clases de Java estándar y en tiempo real ejecutándose al mismo tiempo. Para obtener información, consulte los apartados “Creación de la aplicación de

muestra" en la página 93 y "Ejecución de la aplicación de muestra" en la página 93, donde también verá la salida que puede esperar de la aplicación de muestra.

Nota: Puede iniciar tanto la simulación como el controlador al mismo tiempo utilizando la clase LaunchBoth.

- Comparar la diferencia cuando utilice Recopilador de basura Metronome y el recopilador de basura estándar. Para obtener información, consulte los apartados "Ejecución de la aplicación de muestra sin Real Time" en la página 94 y "Ejecución de la aplicación de muestra con el recopilador de basura Metronome" en la página 95.
- Ejecutar la aplicación utilizando el compilador AOT (ahead-of-time). Para obtener información, consulte el apartado "Ejecución de la aplicación de muestra mientras se usa AOT" en la página 96.

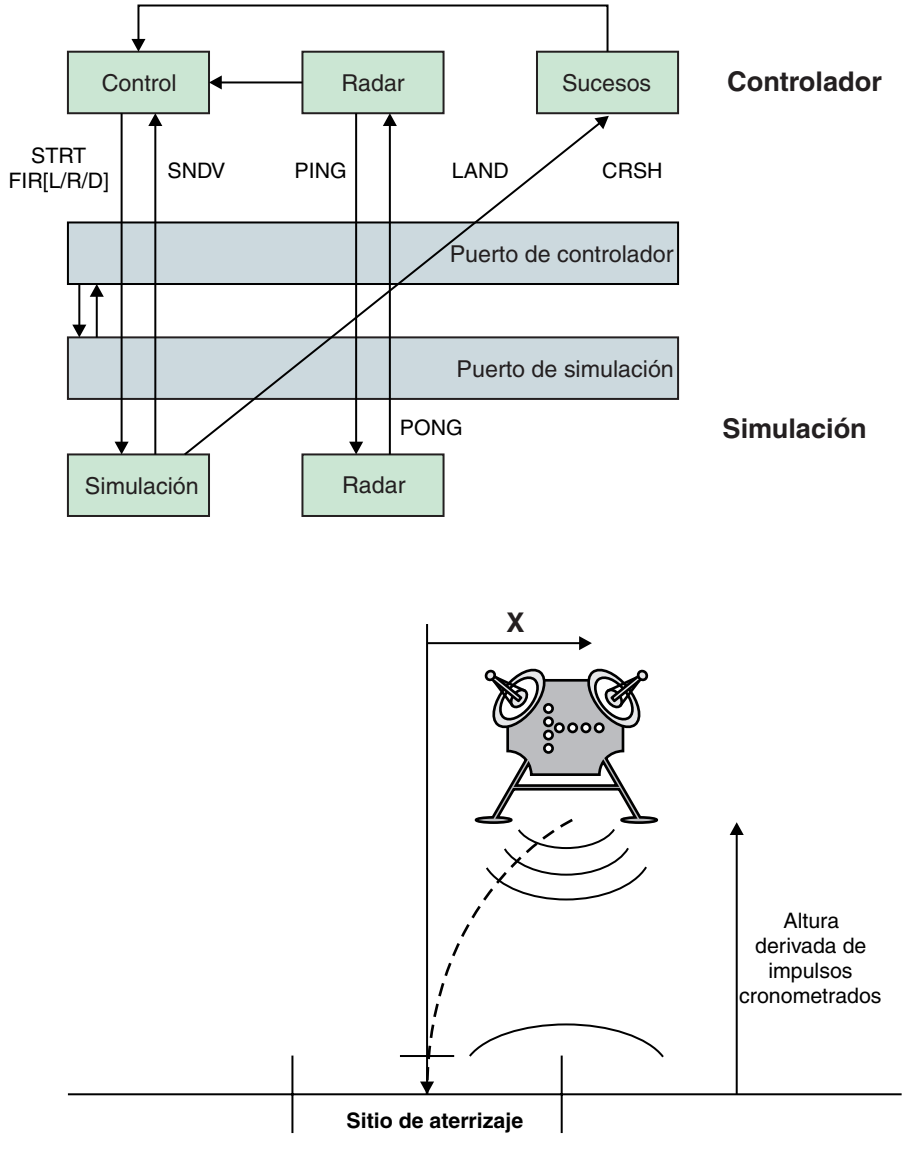

Figura 6. Diagrama del dispositivo de aterrizaje lunar

Este diagrama muestra la relación de los módulos proporcionados en la muestra. Los componentes del controlador y de simulación se comunican entre sí a través de los puertos de controlador y simulación. El componente del controlador tiene tres hebras: Control, Radar y Sucesos. El componente de simulación tiene dos hebras: Simulación y Radar. La hebra de control inicia la simulación y, a continuación, envía mensajes de activación al componente de simulación para controlar la dirección del dispositivo de aterrizaje. La hebra de simulación envía los valores que representan su estado. Las hebras de radar de cada componente se envían mensajes PING y PONG entre sí. La hebra de control utiliza el tiempo que transcurre entre el intercambio de estos mensajes para calcular la altura del dispositivo. La hebra de simulación también envía sucesos de finalización adecuados, ya sea de choque o aterrizaje, a la hebra de sucesos. La hebra de sucesos devuelve los sucesos de finalización a la hebra de control, con lo que finaliza la simulación.

Creación de la aplicación de muestra

El código fuente de la aplicación de muestra se proporciona como orientación. La preparación requiere el desempaquetado y la compilación del código fuente de Java para que se pueda ejecutar.

Procedimiento

1. Cree un directorio de trabajo.
2. Extraiga la aplicación de muestra en el directorio de trabajo:
   ```
   unzip sample_application.zip
   ```
3. Cree un nuevo directorio para la salida:
   ```
   mkdir classes
   ```
4. Compile el origen.
   a. Genere una lista de archivos:
      ```
      find -name "*.java" > source
      ```
   b. Compile el origen:
      ```
      javac -Xrealtime -Xlint:deprecated -g -d classes @source
      ```
   c. Cree un archivo .jar a partir de los archivos de clase:
      ```
      jar cf demo.jar -C classes/ .
      ```

Qué hacer a continuación

Ahora puede ejecutar la aplicación de muestra.

Ejecución de la aplicación de muestra

WebSphere Real Time proporciona una JVM estándar, así como una JVM de tiempo real, que se inicia con el argumento de línea de mandatos **-Xrealtime**.

La aplicación de muestra tiene dos componentes, diseñados para ejecutarse en JVM independientes:

- Simulación, que solo se ejecuta en Real-Time Java.
- Controlador, que se puede ejecutar en Real-Time o no Real-Time Java.

La ejecución del código de Controlador en varias modalidades muestra las ventajas de la tecnología IBM Real-Time Java.

Ejecución de la aplicación de muestra sin Real Time

En este procedimiento, la aplicación de muestra se ejecuta sin aprovechar IBM WebSphere Real Time.

Antes de empezar

Para ejecutar la aplicación de muestra, antes debe crear el código fuente de muestra. Consulte el apartado "Creación de la aplicación de muestra" en la página 93 para obtener más información.

Procedimiento

1. Inicie la simulación:

   ```
   java -Xrealtime -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m
   demo.sim.SimLauncher <puerto>
   ```

   En este mandato, *<port>* es un puerto desasignado para la estación de trabajo.
2. Inicie el controlador:

   ```
   java -classpath ./demo.jar -mx300m demo.controller.JavaControlLauncher <host>
   <puerto>
   ```

   En este mandato *<host>* es el nombre de host de la estación de trabajo que ejecuta la simulación, y *<puerto>* es el puerto especificado en el paso anterior.

Resultados

La aplicación produce un mensaje que muestra que se han iniciado la simulación y el controlador:

```
SimLauncher: Waiting for connections...
Starting control thread...
```

Algunas muestras puntuales de los valores del controlador se imprimen en la consola:

```
x=99.50, radar=199.11, y=198.34, vx=-0.71, vy=-0.43, timeSinceLast=0.19, targetVx=-6.01, targetVy=-9.00
x=95.50, radar=194.59, y=192.70, vx=-2.70, vy=-2.43, timeSinceLast=0.20, targetVx=-5.94, targetVy=-9.00
x=87.50, radar=186.57, y=183.06, vx=-4.70, vy=-4.40, timeSinceLast=0.20, targetVx=-5.77, targetVy=-9.00
x=76.46, radar=172.84, y=169.42, vx=-5.42, vy=-6.75, timeSinceLast=0.20, targetVx=-5.60, targetVy=-9.00
x=65.36, radar=155.58, y=151.84, vx=-5.50, vy=-9.19, timeSinceLast=0.20, targetVx=-5.57, targetVy=-9.00
x=54.36, radar=138.06, y=135.24, vx=-5.44, vy=-7.63, timeSinceLast=0.20, targetVx=-5.56, targetVy=-9.00
x=43.26, radar=120.57, y=117.22, vx=-5.67, vy=-9.62, timeSinceLast=0.20, targetVx=-5.52, targetVy=-9.00
x=32.36, radar=103.60, y=100.72, vx=-5.47, vy=-9.06, timeSinceLast=0.20, targetVx=-5.43, targetVy=-9.00
x=21.52, radar=84.60, y=82.86, vx=-5.32, vy=-9.09, timeSinceLast=0.20, targetVx=-5.60, targetVy=-9.00
x=10.72, radar=67.07, y=65.56, vx=-5.30, vy=-10.54, timeSinceLast=0.20, targetVx=-5.65, targetVy=-9.00
x=0.76, radar=51.08, y=49.78, vx=-4.30, vy=-7.52, timeSinceLast=0.20, targetVx=-0.50, targetVy=-9.00
x=-5.24, radar=37.07, y=35.94, vx=-2.30, vy=-8.26, timeSinceLast=0.20, targetVx=0.50, targetVy=-9.00
x=-7.24, radar=20.05, y=19.90, vx=-0.30, vy=-6.15, timeSinceLast=0.20, targetVx=0.50, targetVy=-9.00
x=-6.36, radar=2.68, y=2.80, vx=0.27, vy=-10.08, timeSinceLast=0.20, targetVx=0.50, targetVy=-9.00
```

Justo antes de que la simulación se detenga, se emitirá un mensaje de resumen de sucesos:

```
Fire down transitions 141, fire horizontally transitions 141
LAND!
```

Además de muestras puntuales y el mensaje de resumen de sucesos, el controlador produce un gráfico denominado `graph.svg` en el mismo directorio. El gráfico contiene un gráfico de las muestras puntuales. El gráfico muestra el efecto de las pausas de la recogida de basura en la hebra JavaRadar cuando se ejecuta la aplicación con una JVM estándar no de tiempo real. Los datos que representan la

Altura de radar tiene picos. Los picos están provocados por pausas en la recogida de basura estándar que afectan a la aplicación Controller. En algunas ejecuciones, las pausas en la recogida de basura son lo bastante largas como para provocar anomalías, lo que lleva a la emisión del mensaje:

```
CRASH!
```

Para ver los tiempos de pausa provocados por la recogida de basura, añada la opción **-verbose:gc** al mandato de lanzamiento del controlador:

```
java -classpath ./demo.jar -verbose:gc -mx300m demo.controller.JavaControlLauncher
<host> <port>
```

Ejecución de la aplicación de muestra con el recopilador de basura Metronome

Puede ejecutar una aplicación Java estándar en un entorno de tiempo real sin necesidad de volver a escribir el código, añadiendo la opción -Xrealtime. La opción permite tanto las características del lenguaje Java en tiempo real como la Recogida de basura Metronome.

Antes de empezar

Para ejecutar la aplicación de muestra, antes debe crear el código fuente de muestra. Consulte el apartado "Creación de la aplicación de muestra" en la página 93 para obtener más información.

Procedimiento

1. Inicie la simulación:

   ```
   java -Xrealtime -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m
   demo.sim.SimLauncher <puerto>
   ```

   En este mandato, <*port*> es un puerto desasignado en la estación de trabajo.
2. Inicie el controlador:

   ```
   java -Xrealtime -classpath ./demo.jar -mx300m
   demo.controller.JavaControlLauncher <host> <puerto>
   ```

   En este mandato <*host*> es el nombre de host de la estación de trabajo que ejecuta la simulación, y <*puerto*> es el puerto especificado en el paso anterior. Ejecutar ambas JVM en la misma estación de trabajo puede conducir a un comportamiento menos determinante. Consulte el apartado "Consideraciones" en la página 28 para obtener más información.

Resultados

La aplicación se ejecuta y genera varias salidas, incluyendo:

1. Mensajes que muestran que se han iniciado la simulación y el controlador.
2. Muestras puntuales de valores del controlador.
3. Una gráfica denominada `graph.svg` en el mismo directorio, y que incluye un gráfico de muestras puntuales.
4. Un mensaje de resumen de sucesos.

Cuando se ejecuta la aplicación con la recogida de basura metronome, las muestras puntuales y el gráfico correspondiente tienen a mostrar:

- Falta de picos en los datos de Altura del radar.
- Rastreo preciso de los datos de Altura real.

El motivo es que el código de Controller se ejecuta ahora con pausas más cortas para la recogida de basura.

Las pausas de la recogida de basura Metronome son frecuentes, pero suelen durar menos de 1 milisegundo. El número de pausas de la recogida de basura que no es en tiempo real es menos, pero suelen durar decenas o cientos de milisegundos. La diferencia entre las pausas se puede ver añadiendo la opción **-verbose:gc** al mandato de ejecución de Controller.

Para obtener más información sobre la salida de la recogida de basura detallada, consulte "Utilización de la información de verbose:gc" en la página 148.

Ejecución de la aplicación de muestra mientras se usa AOT

Este procedimiento ejecuta una aplicación Java estándar en un entorno de tiempo real mientras se utiliza el compilador AOT (ahead-of-time), sin necesidad de volver a escribir el código. Utilice esta muestra para comparar la ejecución de la misma aplicación cuando se usa el compilador JIT.

Consulte el apartado "Utilización del código compilado con WebSphere Real Time for RT Linux" en la página 45 para obtener más detalles sobre la compilación AOT.

Antes de empezar

Para ejecutar la aplicación de muestra, antes debe crear el código fuente de muestra. Consulte el apartado "Creación de la aplicación de muestra" en la página 93 para obtener más información.

Acerca de esta tarea

El compilador AOT (ahead-of-time) compila su aplicación de Java en el código nativo antes de ejecutarlo. Puede predecir cómo se ejecuta la aplicación con más precisión, porque no hay interrupciones provocadas por la compilación JIT (just-in-time).

Procedimiento

1. Convierta los códigos de bytes de la aplicación en código nativo.
   a. La conversión tiene lugar ejecutando en primer lugar la muestra con el compilador JIT normal.

      ```
      java -Xrealtime -Xjit:verbose={precompile},vlog=./sim.aotOpts \
           -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m
           demo.sim.SimLauncher <puerto>
      ```

      En este mandato, *<port>* es un puerto desasignado para la estación de trabajo.
   b. En una ventana distinta, ejecute la aplicación.

      ```
      java -Xrealtime -Xjit:verbose={precompile},vlog=./control.aotOpts \
           -classpath ./demo.jar -Xmx300m demo.controller.JavaControlLauncher
           localhost <port>
      ```

      En este mandato, *<port>* es el puerto especificado en el paso anterior. Los resultados de la salida de la aplicación son parecidos a los mensajes siguientes:

      ```
      Fire down transitions 141, fire horizontally transitions 141
      ```

      y:

```
Land!
```

c. Combine los archivos de opciones AOT creados en los pasos anteriores.

```
cat sim.aotOpts.20081014.234958.13205 control.aotOpts.20081014.234958.13205
> sample.aotOpts
```

Los nombres utilizados para los archivos de registro creados en los pasos anteriores tienen información de ID de proceso y fecha añadidos al nombre de archivo. El formato para el nombre de archivo se especifica mediante la opción **vlog=**. Por ejemplo, **vlog=sim.aotOpts** genera un nombre de archivo similar a **sim.aotOpts.20081014.234958.13205**:

d. Compile los archivos del archivo `sample.aotOpts` en `realtime.jar`,`vm.jar`, `rt.jar`, y la aplicación `demo.jar`. Si utiliza memorias caché de clase compartida, el nombre de la memoria caché no debe superar los 53 caracteres.

```
admincache -Xrealtime -populate -cacheName "sample" -aotFilterFile
sample.aotOpts -classpath ./demo.jar \
$JAVA_HOME/jre/lib/i386/realtime/jclSC160/vm.jar \
$JAVA_HOME/jre/lib/i386/realtime/jclSC160/realtime.jar \
$JAVA_HOME/jre/lib/rt.jar \
./demo.jar
```

Se informa del resultado de la compilación:

```
J9 Java(TM) admincache 1.0
Licensed Materials - Property of IBM

(c) Copyright IBM Corp. 1991, 2008 All Rights Reserved
IBM is a registered trademark of IBM Corp.
Java and all Java-based marks and logos are trademarks or registered
trademarks of Oracle Corporation

JVMSHRC256I Persistent shared cache "sample" has been destroyed
Converting files
Converting /team/mstoodle/demo/sdk/jre/lib/i386/realtime/jclSC160/vm.jar into shared class cache
Succeeded to convert jar file /team/mstoodle/demo/sdk/jre/lib/i386/realtime/jclSC160/vm.jar
Converting /team/mstoodle/demo/sdk/jre/lib/i386/realtime/jclSC160/realtime.jar into shared class cache
Succeeded to convert jar file /team/mstoodle/demo/sdk/jre/lib/i386/realtime/jclSC160/realtime.jar
Converting /team/mstoodle/demo/sdk/jre/lib/rt.jar into shared class cache
Succeeded to convert jar file /team/mstoodle/demo/sdk/jre/lib/rt.jar
Converting /team/mstoodle/demo/demo.jar into shared class cache
Succeeded to convert jar file /team/mstoodle/demo/demo.jar

Processing complete
```

Nota: La línea:

```
JVMSHRC256I Persistent shared cache "sample" has been destroyed
```

significa que una memoria caché existente denominada "sample" se destruye por este mandato, a fin de crear la memoria caché especificada.

e. Muestre el contenido de la memoria caché llena.

```
admincache -Xrealtime -cacheName "sample" -printStats
```

2. Inicie la simulación:

```
java -Xrealtime -Xnojit -Xmx300m -Xshareclasses:name="sample" \
     -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m \
     demo.sim.SimLauncher <puerto>
```

En este mandato, <*port*> es un puerto desasignado para esta estación de trabajo.

3. Inicie el controlador:

```
java -Xrealtime -Xnojit -Xmx300m -Xshareclasses:name="sample" \
     -classpath ./demo.jar \
     demo.controller.JavaControlLauncher <host> <puerto>
```

En este mandato <*host*> es el nombre de host de la estación de trabajo que ejecuta la simulación, y <*puerto*> es el puerto especificado en el paso anterior. Ejecutar ambas JVM en la misma estación de trabajo puede conducir a un comportamiento menos determinante. Consulte el apartado "Consideraciones" en la página 28 para obtener más información.

Resultados

La aplicación se ejecuta y genera varias salidas, incluyendo:

1. Mensajes que muestran que se han iniciado la simulación y el controlador.
2. Muestras puntuales de valores del controlador.
3. Una gráfica denominada `graph.svg` en el mismo directorio, y que incluye un gráfico de muestras puntuales.
4. Un mensaje de resumen de sucesos.

Cuando se ejecuta la aplicación con compilación AOT, las muestras puntuales y el gráfico correspondiente tienen a mostrar:

- Falta de picos en los datos de Altura del radar.
- Rastreo preciso de los datos de Altura real.

El motivo es que el código de Controller se ejecuta ahora con pausas más cortas para la recogida de basura y sin interrupciones de compilación just-in-time.

Una ventaja de utilizar la memoria caché de clase compartida para ejecutar esta aplicación es que las JVM del controlador y la simulación comparten parte de la memoria utilizada por las clases cargadas por ambas JVM.

Correlación hash en tiempo real de muestra

WebSphere Real Time for RT Linux incluye implementaciones HashMap y HashSet que proporcionan rendimiento más coherente para el método put que el HashMap estándar en el SDK de IBM para Java 7.

La java.util.HashMap estándar que proporciona IBM funciona correctamente con aplicaciones de alto rendimiento. También ayuda con las aplicaciones que conocen el tamaño máximo hasta el que necesita crecer su correlación hash. Para las aplicaciones que necesiten una correlación hash que pueda crecer hasta distintos tamaños, según el uso, hay un problema potencial de rendimiento con la correlación hash estándar. La correlación hash estándar proporciona buenos tiempos de respuesta para añadir nuevas entradas a la correlación hash utilizando el método put. Sin embargo, cuando se completa la primera correlación hash, se debe asignar un almacenamiento de copia de seguridad más grande. Esto significa que las entradas del almacenamiento de copia de seguridad actual se deben migrar. Si la correlación hash es grande, el período de tiempo necesario para realizar un método put también podría ser grande. Por ejemplo, la operación podría llevar varios milisegundos.

WebSphere Real Time for RT Linux incluye una correlación hash de tiempo real de muestra. Proporciona la misma interfaz funcional que la java.util.HashMap estándar, pero habilita un rendimiento mucho más sólido para el método put. En lugar de crear un almacenamiento de copia de seguridad y de migrar todas las

entradas cuando la correlación hash se completa, la correlación hash de muestra crea un almacenamiento de copia de seguridad adicional. El nuevo almacenamiento de copia de seguridad se encadena a los otros almacenamientos de copia de seguridad en la correlación hash. El encadenamiento, provoca inicialmente una leve reducción en el rendimiento, mientras que el almacenamiento de copia de seguridad vacío se asigna y se encadena a otros almacenamientos de copia de seguridad. Una vez actualizada la correlación hash de copia de seguridad, resulta más rápido que migrar todas las entradas. Una desventaja de la correlación hash en tiempo real es que las operaciones get, put y remove son algo más lentas. Las operaciones son más lentas porque cada búsqueda se debe realizar mediante un conjunto de correlaciones hash de copia de seguridad, en lugar de solo una.

Para probar la correlación hash de tiempo real, añada el archivo `RTHashMap.jar` al inicio de su classpath de arranque. Si ha instalado WebSphere Real Time for RT Linux en el directorio`$WRT_ROOT`, añada la opción siguiente para utilizar la correlación hash de tiempo real con su aplicación, en lugar de la correlación hash estándar:

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

Los archivos de origen y de clase de la implementación de correlación hash de tiempo real están incluidos en el archivo `demo/realtime/RTHashMap.jar`. Además, se proporcionan también implementaciones java.util.LinkedHashMap y java.util.HashSet en tiempo real.

Desarrollo de aplicaciones WebSphere Real Time for RT Linux utilizando Eclipse

La utilización de Eclipse le proporciona un entorno de desarrollo integrado con todas las características para el desarrollo de sus aplicaciones en tiempo real.

Antes de empezar

Si es la primera vez que utiliza el entorno de desarrollo de aplicaciones de Eclipse para desarrollar aplicaciones en tiempo real, utilice este procedimiento para configurar su entorno.

WebSphere Real Time for RT Linux proporciona el compilador estándar **`javac`** de Oracle. No hay restricciones sobre el compilador que se debe utilizar, pero debe producir archivos de clase Java 5.0 válidos. Sin embargo, las clases javax.realtime.* de Java tienen que estar en la vía de acceso de creación.

Acerca de esta tarea

Para desarrollar sus aplicaciones en Eclipse, siga estas instrucciones:

Procedimiento

1. Descargue Eclipse desde http://www.eclipse.org/downloads/. Se recomienda utilizar Eclipse 3.1.2 para una correcta compilación de Java 5.0.
2. Descargue IBM SDK y Runtime Environment para plataformas Linux, Java 2 Technology Edition, Versión 5.0 con JVM compatible para ejecutar Eclipse.
3. Extraiga este archivo, `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/realtime.jar`, desde el paquete WebSphere Real Time for RT Linux.
4. Abra Eclipse y cree un proyecto. Pulse **Archivo** > **Nuevo**. Seleccione **Proyecto Java** en el panel **Nuevo proyecto**.

5. Pulse **Siguiente** para mostrar el panel **Nuevo proyecto Java**.
   a. Especifique un nombre de proyecto, por ejemplo, RTSJ-Tests.
   b. Compruebe que el compilador JDK se haya definido en 5.0.
6. Pulse **Finalizar**.
7. Cree un directorio de trabajo e importe el archivo `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/realtime.jar`.
8. Pulse **Archivo** > **Nuevo** > **Carpeta** para abrir el panel **Nueva carpeta**. Especifique un nombre para la nueva carpeta, por ejemplo, *deplib*.
9. Pulse **Finalizar**.
10. Para importar su archivo `realtime.jar`, pulse **Archivo** > **Importar** para abrir el panel **Importar**.
11. Pulse **Sistema de archivos** y, a continuación, **Siguiente**.
12. Abra el directorio `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/` del sistema de archivos donde se ha desempaquetado la JVM.
13. Marque el recuadro de selección junto al archivo realtime.jar, especifique una carpeta en la que importar, por ejemplo `RTSJ-Tests/deplib`, y asegúrese de que la opción **Crear sólo carpetas seleccionadas** esté seleccionada.
14. Pulse **Finalizar**.
15. Añada el archivo .jar a la vía de acceso a biblioteca. Pulse con el botón derecho del ratón en el proyecto y, a continuación, en **Propiedades** para abrir el panel **Propiedades**.
16. Pulse **Vía de acceso de creación de Java** y el separador **Bibliotecas**. Pulse **Añadir archivos .jar**.
17. Pulse **realtime.jar** en su directorio de proyecto. Pulse **Aceptar**.

Resultados

Si este procedimiento es correcto, el archivo realtime.jar aparece en la lista de archivos .jar en la pestaña **Bibliotecas**.

Ejemplo

Eclipse puede utilizar `realtime.src.jar` para presentar información adicional sobre las clases RTSJ. Para hacerlo, abra la ventana de propiedades del archivo `realtime.jar` importado, pulse **Adjunto de origen Java** y especifique en la **Vía de acceso de ubicación:** la ubicación del archivo `realtime.src.jar`.

Qué hacer a continuación

Si quiere construir aplicaciones utilizando Apache Ant con Eclipse, añada el archivo `realtime.jar` a la vía de acceso de clases en su script de construcción Ant. Por ejemplo:

```
<property name="rtsj.src" location="." />
<property name="rtsj.deplib" location="deplib" />
<property name="rtsj.jar.dir" location="build/rtsj-jar.dir" />

<!-- Generate .class files for this package -->
<target name="compile" depends="init">
<javac destdir="${rtsj.jar.dir}"
srcdir="${rtsj.src}"
target="1.5"
classpath="${rtsj.deplib}/realtime.jar:${rtsj.src}"
debug="true"/>
</target>
```

Esto es solo una parte de un script de construcción ant.

Depuración de sus aplicaciones

Con el desarrollador de aplicaciones de Eclipse, puede depurar sus aplicaciones de manera local o remota.

Acerca de esta tarea

Para depurar sus aplicaciones de tiempo real de forma remota, la JVM que se está depurando necesita la opción siguiente.
`-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=10100`

Procedimiento

1. En el entorno Linux donde se ejecute su aplicación, especifique:

   ```
   java -Xrealtime -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=10100
   ```

   donde:
   - `server=y` indica que la JVM acepta conexiones de los depuradores.
   - `suspend=y` hace que la JVM espere a que se adjunte un depurador antes de la ejecución.
   - `address=10100` es el número de puerto en el que el depurador se debe adjuntar a la JVM. Este número debería estar normalmente por encima de 1024.

   La JVM muestra el mensaje siguiente:

   ```
   Listening for transport dt_socket at address: 10100
   ```
2. Abra su aplicación en Eclipse y seleccione **Depurar**.
3. Se debe crear una nueva configuración para depurar aplicaciones remotas. Solo debe crear una si se ejecuta una aplicación en el mismo puerto para cada ejecución.
4. Cuando haya creado la configuración, complete el nombre de las Configuraciones, el nombre del proyecto que contiene la aplicación que está depurando, el *nombre_host* de la estación de trabajo donde se ejecuta la aplicación y el número de puerto pasado en las opciones **`-agentlib`**.
5. Pulse **Depurar** para iniciar la sesión de depuración. La perspectiva **Depurar** debe estar abierta para que pueda ver el estado de la JVM depurada de forma remota.

Ejecución de Eclipse con la JVM

En esta sección se explica cómo ejecutar Eclipse con la JVM de WebSphere Real Time for RT Linux.

Para ejecutar Eclipse con la JVM, especifique los elementos siguientes con el mandato **`eclipse`**:

- El directorio completo al archivo ejecutable de Java de la JVM de WebSphere Real Time for RT Linux que desee utilizar
- La opción **`-Xrealtime`** de JVM
- El tamaño de la memoria inmortal que desea que utilice Eclipse. El tamaño debe ser 128 M como mínimo.

Por ejemplo:

```
eclipse -vm $JAVA_HOME/jre/bin/java -vmargs -Xrealtime -Xgc:immortalMemorySize=128M
```

Nota: Eclipse SDK no aprovecha las distintas opciones de memoria de tiempo real disponibles en las aplicaciones de WebSphere Real Time for RT Linux. Una consecuencia de este comportamiento es que la memoria inmortal se agota, especialmente en aquellos casos en los que Eclipse se utiliza durante muchas horas o días sin reiniciarse en ningún momento. Si se produce un error **OutOfMemory**, puede aumentar el valor de la opción **-Xgc:immortalMemorySize** para aumentar la memoria inmortal que desea que utilice Eclipse.

Capítulo 7. Rendimiento

WebSphere Real Time for RT Linux se ha optimizado para pausas breves en la recogida de basura, en lugar de para el rendimiento más alto en la salida o la ocupación de memoria más baja.

En aquellos sistemas en los que se admite Hyper-Threading, debe asegurarse de que no esté habilitado. El motivo es para evitar los efectos adversos del rendimiento cuando utilice WebSphere Real Time for RT Linux.

La reducción de la variabilidad de temporización y el soporte para Especificación de tiempo real para Java (RTSJ) han requerido que se inhabilitasen algunas optimizaciones de tiempo de ejecución estándar de IBM Java. Por lo tanto, es probable que se produzca una reducción en el rendimiento global cuando una aplicación Java estándar se ejecute con el parámetro **-Xrealtime**.

Rendimiento en configuraciones de hardware certificadas

Los sistemas certificados tienen suficiente granularidad de reloj y velocidad de procesador como para admitir los objetivos de rendimiento de WebSphere Real Time for RT Linux. Por ejemplo, una aplicación bien escrita que se ejecute en un sistema no sobrecargado, y con un tamaño de almacenamiento dinámico adecuado, experimentará normalmente tiempos de pausa por recogida de basura inferiores a 1 milisegundo, normalmente unos 500 microsegundos. Durante los ciclos de recogida de basura, una aplicación con la configuración de entorno predeterminada no se pausa durante más del 30% del tiempo transcurrido durante cualquier ventana deslizante de 10 milisegundos. El tiempo colectivo empleado en la recogida de basura se pausa por encima de cualquier período de 10 milisegundos suele ser en total de menos de 3 milisegundos.

Reducción de la variabilidad de temporización

Los dos principales orígenes de variabilidad en una JVM estándar se gestionan en WebSphere Real Time for RT Linux, como se indica a continuación:

- Preparación del código Java: la carga y la compilación JIT (Just-In-Time) se tratan con compilación AOT (Ahead-Of-Time). Consulte el apartado "Utilización del compilador AOT" en la página 48.
- Pausas en la recogida de basura: las pausas, potencialmente grandes, de las modalidades estándar del recopilador de basura se pueden evitar utilizando Recopilador de basura Metronome. Consulte el apartado "Utilización del recopilador de basura Metronome" en la página 76.

Datos de clases compartidos entre las JVM para tiempo no real

Se da soporte al compartimiento de datos en modalidad de tiempo no real, pero funciona de forma distinta que en la modalidad de tiempo real.

Puede compartir datos de clases entre máquinas virtuales Java (JVM) si los almacena en un archivo en disco de caché correlacionada con memoria. Compartir reduce el consumo de almacenamiento virtual global cuando más de una JVM comparte una memoria caché. También se disminuye el tiempo de arranque de una

JVM después de que se ha creado la memoria caché. La memoria caché de clase compartida es independiente de cualquier JVM en ejecución y se mantiene hasta que se suprime.

Una memoria caché compartida puede contener:

- Clases de arranque
- Clases de aplicación
- Metadatos que describen las clases
- Código compilado de manera anticipada (AOT)

Nota: Una JVM no de tiempo real no puede eliminar una memoria caché de clases compartidas en tiempo real.

Capítulo 8. Seguridad

Esta sección contiene información importante sobre seguridad.

Consideraciones de seguridad para la memoria caché de clase compartida

La memoria caché de clase compartida se ha diseñado para facilitar el uso y la gestión de la memoria caché, pero la política de seguridad predeterminada puede no ser la adecuada.

Cuando utilice la memoria caché de clase compartida, debe tener en cuenta los permisos predeterminados de los nuevos archivos, a fin de poder mejorar la seguridad restringiendo el acceso.

Archivo	**Permisos predeterminados**
nuevas memorias caché compartidas	permisos de lectura de grupos y otros
directorio `javasharedresources`	permiso universal de ejecución, escritura y lectura

Se requiere permiso de escritura en el archivo de memoria caché y el directorio de memoria caché, para destruir o desarrollar una memoria caché.

Cambio de los permisos de archivos en el archivo de memoria caché

Para limitar el acceso a una memoria caché de clase compartida, puede utilizar el mandato **`chmod`**.

Cambio necesario	**Mandato**
Limitar acceso al usuario y al grupo	`chmod 770 /tmp/javasharedresources`
Limitar acceso al usuario	`chmod 700 /tmp/javasharedresources`
Limitar el acceso de lectura y escritura del usuario a solo una memoria caché concreta	`chmod 600 /tmp/javasharedresources/<`*`archivo para memoria caché compartida`*`>`
Limitar el acceso de lectura y escritura del usuario y el grupo a solo una memoria caché concreta	`chmod 660 /tmp/javasharedresources/<`*`archivo para memoria caché compartida`*`>`

Consulte el apartado “Creación de una memoria caché de clase compartida en tiempo real” en la página 50 para obtener más información sobre cómo crear una memoria caché de clase compartida.

Cómo conectarse a una memoria caché a la que no tiene permiso para acceder

Si intenta conectarse a una memoria caché para la que no tiene los permisos de acceso adecuados, verá un mensaje de error:

```
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.
```

Capítulo 9. Resolución de problemas y soporte

Resolución de problemas y soporte para WebSphere Real Time for RT Linux

Métodos para determinación de problemas generales

La determinación de problemas le ayuda a comprender el tipo de error que tiene y las acciones que debe realizar.

Cuando sepa que tipo de problema tiene, puede realizar una o más de las siguientes tareas:

- Arreglar el problema.
- Buscar un buen método alternativo.
- Recopilar los datos necesarios con los que generar un informe de errores para IBM.

Determinación de problemas en Linux

Este apartado describe la determinación de problemas en Linux.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre el diagnóstico de problema en Linux, y cubre:

- Configuración y comprobación del entorno de Linux
- Técnicas de depuración generales
- Diagnóstico de bloqueos
- Depuración de cuelgues
- Depuración de fugas de memoria
- Depuración de problemas de rendimiento

Puede encontrar esta información en: IBM SDK para Java 7 - Determinación de problemas en Linux .

La información siguiente es adicional para IBM WebSphere Real Time for RT Linux

Configuración y comprobación del entorno de Linux

En IBM WebSphere Real Time for RT Linux, compruebe que la JVM está configurada correctamente para generar un volcado del sistema.

volcados del sistema Linux (archivos clase Core)

Cuando se produce un bloqueo, los datos de diagnóstico más importantes que deben obtenerse son el volcado del sistema de Linux (archivo core). Para asegurarse de que se genera este archivo, debe comprobar los valores del sistema operativo y su espacio de disco disponible, según se muestra en la Guía del usuario de IBM SDK para Java V7.

Valores de la máquina virtual Java

La JVM se debe configurar para generar archivos core cuando se produzca un bloqueo. Ejecute `java -Xrealtime -Xdump:what` en la línea de mandatos. La salida de esta opción es:

```
-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
```

Los valores mostrados son los valores predeterminados. Al menos debe establecerse `events=gpf` para general un archivo clase Core cuando se produzca un bloqueo. Puede cambiar y establecer las opciones con la opción de línea de mandatos **-Xdump:system[:name1=value1,name2=value2 ...]**

Técnicas de depuración generales

Como los nombres de hebra de Java son visibles en el sistema operativo, puede utilizar el mandato ps como ayuda para la depuración. Cuando utilice herramientas de rastreo, debe utilizar los mandatos correctos para IBM WebSphere Real Time for RT Linux.

Examinar la información sobre procesos

Cuando ejecute el mandato ps en IBM WebSphere Real Time for RT Linux, la salida será algo parecido a lo siguiente:

```
ps -eLo pid,tid,rtprio,comm,cmd
29286 29286      - java            jre/bin/java -Xrealtime -jar example.jar
29286 29287      - main            jre/bin/java -Xrealtime -jar example.jar
29286 29290     88 Signal Reporter jre/bin/java -Xrealtime -jar example.jar
29286 29295      - JIT Compilation jre/bin/java -Xrealtime -jar example.jar
29286 29296     13 JIT Sampler     jre/bin/java -Xrealtime -jar example.jar
29286 29297      - Signal Dispatch jre/bin/java -Xrealtime -jar example.jar
29286 29298      - Finalizer maste jre/bin/java -Xrealtime -jar example.jar
29286 29299     11 Gc Slave Thread jre/bin/java -Xrealtime -jar example.jar
29286 29300     89 Metronome GC Al jre/bin/java -Xrealtime -jar example.jar
29286 29301      - Thread-2        jre/bin/java -Xrealtime -jar example.jar
29286 29302     43 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29303     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29304     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29305     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29306     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29307     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29311     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29312     83 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29313     85 Realtime AEH No jre/bin/java -Xrealtime -jar example.jar
29286 29314     85 Realtime AEH No jre/bin/java -Xrealtime -jar example.jar
29286 29315     87 Realtime Schedu jre/bin/java -Xrealtime -jar example.jar
29286 29316     79 Realtime AEH Se jre/bin/java -Xrealtime -jar example.jar
29286 29317     85 Realtime Non-he jre/bin/java -Xrealtime -jar example.jar
29286 29318     83 Realtime Heap T jre/bin/java -Xrealtime -jar example.jar
29286 29319     83 Realtime Heap T jre/bin/java -Xrealtime -jar example.jar
29286 29321     45 RealtimeThread- jre/bin/java -Xrealtime -jar example.jar
29286 29343     43 RealtimeThread- jre/bin/java -Xrealtime -jar example.jar
29286 29345      - stdout reader j jre/bin/java -Xrealtime -jar example.jar
29286 29346      - stderr reader j jre/bin/java -Xrealtime -jar example.jar
```

e Selecciona todos los procesos.

L Muestra las hebras.

o Proporciona un formato predefinido de las columnas que se deben

visualizar. Las columnas especificadas son el ID de proceso, el ID de hebra, la política de planificación, la prioridad de hebra en tiempo real y el mandato asociado al proceso. Esta información es útil para entender qué hebras de la aplicación así como de la máquina virtual se están ejecutando en un momento dado.

Herramientas de rastreo

Tres herramientas de rastreo de Linux son **strace**, **ltrace** y **mtrace**. El mandato `man strace` muestra un conjunto completo de opciones disponibles.

strace
La herramienta strace rastrea las llamadas al sistema. Puede utilizarla en un proceso que ya está disponible, o bien iniciarla con un nuevo proceso. strace registra las llamadas al sistema que realiza un programa y las señales que recibe un proceso. Para cada llamada al sistema, se utiliza el nombre, los argumentos y el valor de retorno. strace le permite rastrear un programa sin requerir el código fuente (no es necesario volver a compilar). Si utiliza strace con la opción **-f**, se rastrearán los procesos hijo que se hayan creado como resultado de una llamada al sistema bifurcada. Puede utilizar strace para investigar problemas de plug-in o intentar entender por qué los programas no se inician correctamente.

Para utilizar strace con una aplicación Java, escriba `strace java -Xrealtime <class-name>`.

Puede dirigir la salida de rastreo desde la herramienta strace a un archivo utilizando la opción `-o`.

ltrace
La herramienta ltrace depende de la distribución. Es muy parecida a strace. Esta herramienta intercepta y registra las llamadas dinámicas a la biblioteca que realiza el proceso en ejecución. strace hace lo mismo para las señales que recibe el proceso en ejecución.

Para utilizar ltrace con una aplicación Java, escriba `ltrace java -Xrealtime <class-name>`

mtrace
mtrace se incluye en el conjunto de herramientas GNU. Instala manejadores especiales para malloc, realloc y free, y permite que todos los usos de estas funciones se rastreen y se registren en un archivo. Este rastreo disminuye la eficiencia del programa y no debe habilitarse durante el uso normal. Para utilizar mtrace, establezca **IBM_MALLOCTRACE** en 1, y establezca **MALLOC_TRACE** para que apunte a un archivo válido donde se almacenará la información de rastreo. Debe tener acceso de escritura a este archivo.

Para utilizar mtrace con una aplicación Java, escriba:

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

Diagnóstico de bloqueos

Cuando recopile información sobre la ejecución de procesos y el entorno de Java antes de un bloqueo, siga estas directrices.

Recopilación de información sobre procesos

Cuando investigue qué ha pasado antes de que se produjera el bloqueo, use los mandatos **gdb** y **bt** para mostrar el seguimiento de la pila de la hebra errónea, en vez de analizar el archivo principal (core).

Obtención de información sobre el entorno Java

Utilice el volcado Java para determinar qué estaba haciendo cada hebra y qué métodos Java se estaban ejecutando. Relacione las direcciones de función con las direcciones de biblioteca para determinar el origen del código que se ejecuta en diversos puntos.

Use la opción **-verbose:gc** para ver el estado del almacenamiento dinámico de Java y las áreas de memoria inmortal y con ámbito. Hágase estas preguntas:

- ¿Ha habido una falta de memoria en una de las áreas de memoria que haya podido causar el bloqueo?
- ¿Se ha producido el bloqueo durante la recopilación de basura, lo que apuntaría a un posible error en la recopilación de basura?
- ¿Se ha producido el bloqueo después de la recopilación de basura, lo que indicaría posibles daños en la memoria?

Depuración de problemas de rendimiento

Cuando depure problemas de rendimiento, tenga en cuenta estos elementos específicos para IBM WebSphere Real Time for RT Linux además de los temas en la Guía del usuario de IBM SDK para Java V7.

Asignación de tamaño de áreas de memoria

La JVM puede ajustarse variando los tamaños del almacenamiento dinámico, la memoria inmortal y memoria de ámbito. Elija el tamaño adecuado para optimizar el rendimiento. Al utilizar el tamaño adecuado el colector de basura puede proporcionar más fácilmente la utilización necesaria.

Para obtener más información sobre cómo cambiar el tamaño de las áreas de memoria, consulte “Resolución de problemas del recopilador de basura Metronome” en la página 148.

Compilación JIT y rendimiento

Al usar el JIT, debe considerar las implicaciones para el comportamiento en tiempo real.

Si necesita un comportamiento predecible pero también un mejor rendimiento, debe considerar utilizar la compilación anticipada (AOT). Para obtener más información, consulte “Utilización del código compilado con WebSphere Real Time for RT Linux” en la página 45.

Limitaciones conocidas en Linux

Linux ha tenido un desarrollo rápido y se han observado varios problemas con la interacción de la JVM y el sistema operativo, especialmente en el área de las hebras.

Tenga en cuenta las siguientes limitaciones que podrían afectar a su sistema Linux.

Hebras como procesos

Si el número de hebras Java sobrepasa el número máximo de procesos permitidos, el programa podría:

- Obtener un mensaje de error
- Obtener un error **SIGSEGV**
- Detenerse

Para obtener más información, consulte *El informe Volano* en http://www.volano.com/report/index.html.

Limitaciones de las pilas flotantes

Si está ejecutando sin pilas flotantes, independientemente de lo que se establezca para **-Xss**, se proporciona un tamaño mínimo de pilas nativas de 256 KB para cada hebra.

En un sistema Linux de pilas flotantes, se utilizan los valores **-Xss**. Si realiza la migración desde un sistema Linux sin pilas flotantes, asegúrese de que los valores **-Xss** son suficientemente grandes y que no se basan en el mínimo de 256 KB.

limitaciones de glibc

Si recibe un mensaje que indica que la biblioteca `libjava.so` no se ha podido cargar porque no se ha encontrado un símbolo (como por ejemplo __bzero), es posible que tenga instalada una versión anterior de la biblioteca GNU C Runtime Library, glibc. El SDK para la implementación de la hebra de Linux requiere glibc versión 2.3.2 o superior.

Limitaciones de font

Cuando se instala en un sistema Red Hat, para permitir que el servidor encuentre los fonts TrueType de Java, ejecute (en Linux IA32, por ejemplo):

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

Debe hacerlo durante la instalación y debe haber iniciado sesión como usuario "root" para ejecutar el mandato. Para ver problemas de font más detallados, consulte el *SDK Linux y la Guía del usuario del entorno de tiempo de ejecución*..

Problemas de rendimiento en los kernels de Linux Red Hat MRG

Un problema de configuración con los kernels de Red Hat MRG puede producir pausas inesperadas en la hebras de aplicación cuando WebSphere Real Time se inicia con la recopilación de basura detallada habilitada. Estas pausas no se notifican en la salida de la recopilación de basura (GC) detallada, pero pueden durar varios milisegundos, en función de la configuración de red. Las JVM iniciadas desde usuarios LDAP definidos remotamente son las que se ven más afectadas, ya que el daemon de la memoria caché de servicio de nombres (`nscd`) no inicia y produce retardos en la red. Solucione el problema iniciando `nscd`. Siga los pasos que se indican a continuación para comprobar el estado del servicio `nscd` y corrija el problema:

1. Para comprobar que el daemon `nscd` se está ejecutando, escriba el siguiente mandato:

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

Si el daemon no se está ejecutando, verá el siguiente mensaje:

```
nscd is stopped
```

2. Como usuario root, inicie el servicio `nscd` con el siguiente mandato:

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

3. Como usuario root, cambie la información de inicio para el servicio `nscd` con el siguiente mandato:

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

Ahora el proceso `nscd` se está ejecutando y se inicia automáticamente después de rearrancar.

Determinación de problemas de NLS

La JVM contiene soporte integrado para diferentes entornos locales.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre el diagnóstico de problema con NLS, y cubre:

- Visión general de fonts
- Programas de utilidad de fonts
- Problemas comunes con NLS y causas posibles

Esta información está en: IBM SDK para Java 7 - Determinación de problemas de NLS.

Determinación de problemas de ORB

Una de las primeras tareas al depurar un problema de ORB es determinar si el problema está en el lado del cliente o en el lado del servidor de la aplicación distribuida. Piense en una sesión de RMI-IIOP normal como una sencilla comunicación síncrona entre un cliente que solicita acceso a un objeto y un servidor que lo suministra.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre el diagnóstico de problema con ORB, y cubre:

- Identificación de un problema de ORB
- Interpretación del rastreo de la pila
- Interpretación de rastreos de ORB
- Problemas comunes
- Servicio ORB de IBM: recopilación de datos

Esta información está en: IBM SDK para Java 7 - Determinación de problemas de ORB.

La información siguiente es adicional para IBM WebSphere Real Time for RT Linux.

Servicio ORB de IBM: recopilación de datos

Cuando recopile la salida de la versión de Java para el servicio, ejecute el mandato siguiente:

```
java -Xrealtime -version
```

Pruebas preliminares

Cuando se produzca un problema, el ORB podría generar una excepción `org.omg.CORBA.*` que incluya:

- texto para indicar la causa
- un código menor
- Un estado de terminación

Antes de dar por supuesto que el ORB es la causa del problema, compruebe estos elementos:

- El escenario se puede reproducir en una configuración similar.
- JIT está inhabilitado.
- NO se está utilizando ningún código compilado por AOT

Otras acciones serían:

- Apagar los procesadores adicionales.
- Apagar la multihebra simultánea (SMT) cuando sea posible.
- Elimine dependencias de memoria con el cliente o el servidor. La falta de memoria física puede ser la causa de un rendimiento lento, de bloqueos o cuelgues aparentes. Para eliminar estos problemas, asegúrese de disponer de suficiente espacio de memoria.
- Compruebe los problemas de red física, como cortafuegos, enlaces de comunicaciones, direccionadores y servidores de nombres DNS. Éstas son las causas principales de las excepciones de COMM_FAILURE de CORBA. Como prueba, haga ping en el nombre de su propia estación de trabajo.
- Si la aplicación utiliza una base de datos como, por ejemplo, DB2, cambie al controlador más fiable. Por ejemplo, para aislar DB2 AppDriver, cambie a Net Driver, que es más lento y utiliza sockets, pero es más fiable.

Resolución de problemas de tipo OutOfMemory

Tratamiento de excepciones OutOfMemoryError, fugas de memoria y asignaciones de memoria oculta.

Para información de resolución de problemas general sobre el uso de la Recogida de basura metronome, consulte “Resolución de problemas del recopilador de basura Metronome” en la página 148.

Diagnóstico de excepciones OutOfMemoryError

Diagnosticar excepciones de tipo OutOfMemoryError en el recopilador de basura Metronome puede resultar más complejo que en una JVM estándar, debido a la naturaleza periódica del recopilador de basura.

En el apartado “Gestión de memoria” en la página 16 encontrará las características de los distintos tipos de almacenamiento dinámico. Por lo general, una aplicación RTSJ requiere aproximadamente un 20% más de espacio de almacenamiento dinámico que una aplicación Java estándar.

De forma predeterminada, la JVM produce la siguiente salida de diagnóstico cuando se genera una excepción de tipo OutOfMemoryError no detectada:

- Un volcado breve; consulte el apartado “Uso de agentes de volcado” en la página 126.

- Un volcado de almacenamiento dinámico; consulte el apartado “Utilizar el vuelco de almacenamiento dinámico” en la página 135.
- Un volcado Java; consulte el apartado “Uso del volcado Java” en la página 129
- Un volcado del sistema; consulte “Uso de volcados del sistema y el visor de volcados” en la página 139.

Los nombres de los archivos de volcado se otorgan en la salida de la consola:

```
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".
```

El rastreo inverso de Java, mostrado en la salida de la consola, también disponible en el volcado Java, indica en qué parte de la aplicación Java se ha producido la excepción OutOfMemoryError. El paso siguiente es averiguar qué área de memoria RTSJ está llena. El componente de gestión de memoria de la JVM emite un punto de rastreo con el tamaño, la dirección de bloque de clases y el nombre del espacio de memoria de la asignación que falla. Encontrará este punto de rastreo en el volcado breve:

```
<< 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
```

Los campos de datos e ID de punto de rastreo pueden variar con respecto a los mostrados, en función del tipo de objeto que se esté asignando. En este ejemplo, el punto de rastreo muestra que el error en la asignación se ha producido cuando la asignación trató de asignar un objeto de 33,6 MB de tipo `class 0x81312d8` en `Metronome heap`, en el segmento de memoria `id=0x809c5f0`.

Puede determinar qué área de memoria de RTSJ está afectada observando la información de gestión de memoria del volcado Java:

```
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)
NULL
NULL           region     start      end        size       name
1STHEAP        0xF288B5A4 0xF17FF008 0xF27FF008 0x01000000 Immortal
NULL
1STMEMUSAGE Total memory available: 16777216 (0x01000000)
1STMEMUSAGE    Total memory in use:    00450816 (0x0006E100)
1STMEMUSAGE    Total memory free:      16326400 (0x00F91F00)
NULL
1STSEGTYPE     Internal Memory
```

```
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x0808DA48 0x0814A0A8 0x0814A0A8 0x0815A0A8 0x01000040 0x00010000
1STSEGMENT     0x0808DB50 0x08131EB8 0x08131EB8 0x08141EB8 0x01000040 0x00010000
<< lines removed for clarity >>
```

Puede determinar el tipo de objeto que se está asignando examinando la sección de clases del volcado Java:

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

La información del volcado Java confirma que se ha intentado una asignación para una matriz de caracteres, en el almacenamiento dinámico normal (`ID=0xF288B584`) y que el tamaño total asignado del almacenamiento dinámico, indicado por la línea `1STHEAP` correspondiente, es de 67108864 bytes decimales o 0x04000000 bytes hexadecimales, o bien 64 MB.

En este ejemplo, la asignación que falla es más grande en relación con el tamaño total del almacenamiento dinámico. Si se espera que su aplicación cree objetos de 33 MB, el paso siguiente será aumentar el tamaño del almacenamiento dinámico mediante la opción **-Xmx**.

Es más habitual que la asignación que falla sea pequeña en relación con el tamaño total del almacenamiento dinámico. Esto se debe a que las asignaciones anteriores llenan el almacenamiento dinámico. En estos casos, el paso siguiente pasa por utilizar el volcado de almacenamiento dinámico para investigar la memoria asignada a los objetos existentes.

El volcado de almacenamiento dinámico es un archivo binario comprimido que contiene una lista con todos los objetos con su clase de objeto, su tamaño y sus referencias. Analice el volcado de almacenamiento dinámico utilizando la herramienta IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer, disponible para su descarga desde IBM Support Assistant (ISA).

Mediante MDD4J, puede cargar un volcado de almacenamiento dinámico y buscar tres estructuras de objetos sospechosos de consumir una gran cantidad de espacio en el almacenamiento dinámico. La herramienta proporciona varias vistas para los objetos del almacenamiento dinámico. Por ejemplo, MDD4J puede mostrar una vista que detalla los sospechosos probables de las fugas de memoria, además de mostrarse los cinco principales objetos y paquetes que contribuyen al tamaño del almacenamiento dinámico. Al seleccionar la vista de árbol, obtendrá más información sobre la naturaleza del objeto contenedor con la fuga de memoria.

De forma predeterminada, se produce un único archivo de volcado de almacenamiento dinámico que contiene todos los objetos de todos los espacios de memoria RTSJ. Utilice la opción de línea de mandatos **-Xdump:heap:request=multiple** para solicitar un volcado de almacenamiento dinámico individual para cada espacio de memoria. Con varios volcados, puede examinar el conjunto de objetos asignados en un área de memoria específica. Los volcados de almacenamiento dinámico se identifican mediante el nombre de archivo otorgado en la salida de la consola:

```
JVMDUMP006I Processing Dump Event "uncaught", detail "java/lang/OutOfMemoryError" - Please Wait.
<< lines omitted... >>
JVMDUMP007I JVM Requesting Heap Dump using '/home/test/heapdump-Default0809DCD8-0002.phd'
JVMDUMP010I Heap Dump written to /home/test/heapdump-Default0809DCD8-0002.phd
JVMDUMP007I JVM Requesting Heap Dump using '/home/test/heapdump-Immortal0809DCF4-0002.phd'
JVMDUMP010I Heap Dump written to /home/test/heapdump-Immortal0809DCF4-0002.phd
JVMDUMP007I JVM Requesting Heap Dump using '/home/test/heapdump-Scope0809DD10-0002.phd'
JVMDUMP010I Heap Dump written to /home/test/heapdump-Scope0809DD10-0002.phd
<< lines omitted... >>
JVMDUMP013I Processed Dump Event "uncaught", detail "java/lang/OutOfMemoryError".
Exception in thread "RTJ Memory Consumer (thread_type=Realtime)" java.lang.OutOfMemoryError
 at tests.com.ibm.jtc.ras.runnable.DepleteMemory.depleteMemory(DepleteMemory.java:57)
<< lines omitted... >>
```

Cómo gestiona la memoria la JVM de IBM

La JVM de IBM requiere memoria para varios componentes distintos, incluidas regiones de memoria para clases, código compilado, objetos Java, pilas Java y pilas JNI. Algunas de estas regiones de memoria deben estar en la memoria contigua. Otras regiones de memoria se pueden segmentar en regiones de memoria más pequeñas y enlazarse juntas.

Las clases cargadas de forma dinámica y el código compilado se almacenan en regiones de memoria segmentadas para clases cargadas dinámicamente. Las clases se subdividen a continuación en regiones de memoria de escritura (clases RAM) y regiones de memoria de solo lectura (clases ROM). Durante el tiempo de ejecución, la memoria caché de clase se correlaciona mediante la memoria, pero no es inevitable que se cargue, en una región de memoria contigua del inicio de la aplicación. Dado que la aplicación hace referencia a las clases, estas clases y el código compilado en la memoria caché de clase se correlacionan en el almacenamiento. Varios procesos que hacen referencia a esta clase comparten el componente ROM de la clase. La primera vez que la JVM hace referencia a la clase, se crea el componente RAM de la clase en las regiones de memoria segmentadas para las clases cargadas dinámicamente. El código compilado mediante AOT para los métodos de una clase de la memoria caché de clase se copian en una región de memoria de código dinámico ejecutable, porque los procesos no comparten este código. Las clases que no se cargan desde la memoria caché de clase son similares a las clases almacenadas en memoria caché, salvo que la información de la clase ROM se crea en regiones de memoria segmentadas para las clases cargadas de forma dinámica. El código generado de forma dinámica se almacena en las mismas regiones de memoria de código dinámico que contienen el código AOT para clases almacenadas en memoria caché.

Todos los objetos de Java se almacenan en la memoria de almacenamiento dinámico estándar cuando la JVM se ejecuta sin la opción **-Xrealtime**. Si se utiliza la opción **-Xrealtime**, los objetos también se pueden asignar fuera de las dos regiones de memoria adicionales, denominadas memoria inmortal y memoria de ámbito.

La pila para cada hebra de Java puede abarcar una región de memoria segmentada. La pila JNI de cada hebra ocupa una región de memoria contigua.

Para determinar la configuración de la JVM, ejecútela con la opción **-verbose:sizes**. Esta opción imprime información sobre las regiones de memoria en las que puede gestionar el tamaño. Para las regiones de memoria que no son contiguas, se imprime un incremento que describe cuánta memoria se obtiene cada vez que la región necesita crecer.

Esta es una salida de ejemplo que utiliza las opciones **-Xrealtime -verbose:sizes**:

```
-Xmca32K                        RAM class segment increment
-Xmco128K                       ROM class segment increment
-Xms64M                         initial memory size
-Xgc:immortalMemorySize=16M     immortal memory space size
-Xgc:scopedMemoryMaximumSize=8M scoped memory space maximum 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
```

En este ejemplo se indica que el segmento de la clase RAM es inicialmente 0, pero crece en bloques de 32 KB cuando es necesario. El segmento de la clase ROM es inicialmente 0, y crece en bloques de 128 KB cuando es necesario. Puede utilizar las opciones **-Xmca** y **-Xmco** para controlar estos tamaños. Los segmentos de la clase RAM y la clase ROM crecen cuando es necesario, por lo que normalmente no será necesario que cambie estas opciones.

La memoria inmortal es una región contigua y es posible que sea preciso asignarle previamente un espacio mayor. En este ejemplo, a la región de la memoria inmortal se le han preasignado 16 MB. Si intenta grabar más de 16 MB de objetos en esta región de memoria inmortal, recibirá una excepción de tipo OutOfMemory porque, por definición, no se recopila la basura de esta área de memoria.

La región de memoria de ámbito es contigua y, en este ejemplo, se le han preasignado 8 MB. Si tiene muchas áreas de memoria de ámbito activas mientras el programa se ejecuta, es posible que tenga que especificar una región de memoria de ámbito mayor.

Utilice el programa de utilidad admincache para determinar el tamaño que tendrá su región de memoria correlacionada si utiliza la memoria caché de clase. Esta es una muestra de la salida del mandato `admincache -Xrealtime -printStats -nologo`:

```
J9 Java(TM) admincache 1.0

Current statistics for cache "sharedcc_localuser":


base address       = 0xA52B4000
end address        = 0xA59B7000
allocation pointer = 0xA59B4000

cache size         = 7356040
free bytes         = 330604
ROMClass bytes     = 3798460
AOT bytes          = 3101560
Data bytes         = 3812
Metadata bytes     = 121604
Metadata % used  = 1%

# ROMClasses       = 1044
# AOT Methods      = 1652
# Classpaths       = 2
# URLs             = 1
# Tokens         = 0
# Stale classes = 0
% Stale classes  = 0%

Cache is 95% full
```

El tamaño de memoria caché indica que la región de memoria correlacionada tendrá poco más de 7 MB de espacio. La clase ROM y los bytes AOT ocupan la mayor parte de este espacio, utilizando poco más de 3 MB respectivamente.

Ejemplo de OutOfMemoryError en espacio de memoria inmortal

Este ejemplo muestra cómo identificar una excepción de tipo OutOfMemoryError en el espacio de memoria inmortal y describe los pasos para evitar el problema.

El volcado breve muestra que han fallado dos solicitudes de asignación pequeña en el área de memoria inmortal id=0x809dd1c:

```
16:08:04.876087000 083d4000       j9mm.100  Event       J9AllocateObject() returning NULL!
  16 bytes requested for object of class 0x8110e60 from memory space 'Immortal' id=0x809dd1c
16:08:04.876171000 083d4000       j9mm.100  Event       J9AllocateObject() returning NULL!
  32 bytes requested for object of class 0x81180f0 from memory space 'Immortal' id=0x809dd1c
```

El volcado Java muestra que el espacio de la memoria inmortal está lleno:

```
NULL           ------------------------------------------------------------------------
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
1STHEAPFREE    Bytes of Heap Space Free: 3f0c000
1STHEAPALLOC  Bytes of Heap Space Allocated: 4000000
1STHEAPFREE    Bytes of Immortal Space Free: 0
1STHEAPALLOC  Bytes of Immortal Space Allocated: 1000000
<< lines omitted... >>
1STSEGTYPE    Object Memory
NULL          segment  start    alloc    end      type     bytes
1STSEGSUBTYPE  Immortal Segment  ID=0809DD1C
1STSEGMENT     0809D510 B279D008 B379D008 B379D008  00001008 1000000
```

Un análisis de MDD4J muestra que se ha asigando un LinkedList muy grande, que consume una gran proporción de la memoria disponible.

Se recomienda minimizar el número de objetos asignados en el área de memoria inmortal, porque no se recoge la basura de los objetos del área inmortal. La utilización más común de memoria inmortal es la carga de clases, una actividad finita que se produce sobre todo durante la inicialización de la aplicación y la JVM. Las aplicaciones con gran cantidad de clases cargadas (u otra utilización de memoria inmortal) puede aumentar el tamaño del área de memoria inmortal utilizando la opción **-Xgc:immortalMemorySize=<tamaño>**. El tamaño predeterminado del área de memoria inmortal es de 16 MB.

Si al aumentar el tamaño del área de memoria inmortal solo se retrasa la excepción OutOfMemoryError de la memoria inmortal, investigue el patrón de asignación continuada de datos inmortales, ya sea relacionados con la carga de clases o con otros objetos de la aplicación.

Ejemplo de OutOfMemoryError en espacio de memoria de ámbito

Este ejemplo muestra cómo identificar una excepción de tipo OutOfMemoryError en el espacio de memoria de ámbito y describe los pasos para evitar el problema.

Utilice la opción de línea de mandatos **-Xdump:heap:request=multiple** para producir volcados independientes para cada espacio de memoria:

```
VMDUMP006I Processing Dump Event "uncaught", detail "java/lang/OutOfMemoryError" - Please Wait.
JVMDUMP007I JVM Requesting Snap Dump using '/home/test/snap-0001.trc'
JVMDUMP010I Snap Dump written to /home/test/snap-0001.trc
JVMDUMP007I JVM Requesting Heap Dump using '/home/test/heapdump-Default0809DCD8-0002.phd'
JVMDUMP010I Heap Dump written to /home/test/heapdump-Default0809DCD8-0002.phd
JVMDUMP007I JVM Requesting Heap Dump using '/home/test/heapdump-Immortal0809DCF4-0002.phd'
JVMDUMP010I Heap Dump written to /home/test/heapdump-Immortal0809DCF4-0002.phd
JVMDUMP007I JVM Requesting Heap Dump using '/home/test/heapdump-Scope0809DD10-0002.phd'
JVMDUMP010I Heap Dump written to /home/test/heapdump-Scope0809DD10-0002.phd
JVMDUMP007I JVM Requesting Java Dump using '/home/test/javacore-0003.txt'
JVMDUMP010I Java Dump written to /home/test/javacore-0003.txt
JVMDUMP013I Processed Dump Event "uncaught", detail "java/lang/OutOfMemoryError".
Exception in thread "RTJ Memory Consumer (thread_type=Realtime)" java.lang.OutOfMemoryError
   at tests.com.ibm.jtc.ras.runnable.DepleteMemory.depleteMemory(DepleteMemory.java:57)
   at tests.com.ibm.jtc.ras.runnable.DepleteMemory.run(DepleteMemory.java:26)
<< lines omitted... >>
```

El volcado breve muestra que han fallado dos solicitudes de asignación en el área de memoria de ámbito id=0x809dd10:

```
16:14:45.887176823 08480900       j9mm.100  Event       J9AllocateObject() returning NULL!
  16 bytes requested for object of class 0x8110e38 from memory space 'Scoped' id=0x809dd10
16:14:45.887252747 08480900       j9mm.100  Event       J9AllocateObject() returning NULL!
  32 bytes requested for object of class 0x81180c8 from memory space 'Scoped' id=0x809dd10
```

El volcado Java muestra que, para el área de memoria de ámbito con id=0x809dd10, el tamaño asignado de la memoria de ámbito es bastante pequeño, solo 60 KB; en este caso, aumente el tamaño del área de memoria de ámbito en el código de la aplicación.

```
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
1STHEAPFREE    Bytes of Heap Space Free: 3eb0000
1STHEAPALLOC  Bytes of Heap Space Allocated: 4000000
1STHEAPFREE    Bytes of Immortal Space Free: f47474
1STHEAPALLOC  Bytes of Immortal Space Allocated: 1000000
1STHEAPFREE    Bytes of Scoped Space ID=0809DD10 Free: eb00
1STHEAPALLOC   Bytes of Scoped Space Allocated: eb00
.......
1STSEGTYPE    Object Memory
NULL          segment  start    alloc    end      type     bytes
1STSEGSUBTYPE  Scoped Segment  ID=0809DD10
1STSEGMENT     0809D560 08416350 08424E50 08424E50  00002008 eb00
1STSEGSUBTYPE  Immortal Segment  ID=0809DCF4
1STSEGMENT     0809D4E8 B2857008 B3857008 B3857008  00001008 1000000
```

En el volcado Java de ejemplo, el área de memoria de ámbito parece estar vacía. Parece vacía porque el volcado Java se produce cuando la excepción de tipo OutOfMemoryError alcanza la JVM, momento en el que se ha salido del ámbito y se ha limpiado. Puede producir un volcado Java en el punto de la anomalía utilizando la opción de línea de mandatos **-Xdump:java:events=throw,filter=java/lang/OutOfMemoryError**. Al utilizar esta opción, se informa correctamente sobre el espacio libre del área de memoria de ámbito.

También es posible agotar el espacio total disponible de la memoria de ámbito; en este caso, aumente el tamaño del área de la memoria de ámbito utilizando la opción de línea de mandatos **-Xgc:scopedMemoryMaximumSize=<tamaño>**. El tamaño predeterminado para el área de memoria de ámbito es de 8 MB. Si el espacio total disponible para la memoria de ámbito se agota, verá distintos mensajes en la consola, como por ejemplo:

```
Exception in thread "main" java.lang.OutOfMemoryError: Creating (LTMemory) Scoped memory # 0 size=16777216
        at javax.realtime.MemoryArea.create(MemoryArea.java:808)
        at javax.realtime.MemoryArea.create(MemoryArea.java:798)
        at javax.realtime.ScopedMemory.create(ScopedMemory.java:1359)
        at javax.realtime.ScopedMemory.create(ScopedMemory.java:1351)
        at javax.realtime.ScopedMemory.initialize(ScopedMemory.java:1705)
        at javax.realtime.ScopedMemory.<init>(ScopedMemory.java:216)
        at javax.realtime.ScopedMemory.<init>(ScopedMemory.java:164)
```

Problemas de diagnóstico en varios almacenamientos dinámicos

Puede utilizar el rango de direcciones proporcionado en el volcado de almacenamiento dinámico con la información de ocupación del volcado de almacenamiento dinámico para ayudar a analizar las excepciones de tipo OutOfMemoryError en varias áreas de memoria de RTSJ.

En este volcado Java, el segmento inmortal oscila entre 0xB281C008 y 0xB381C008, y el segmento de almacenamiento dinámico normal oscila entre 0xB381D008 y 0xB781D008:

```
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
1STHEAPFREE    Bytes of Heap Space Free: 58000
1STHEAPALLOC  Bytes of Heap Space Allocated: 4000000
1STHEAPFREE    Bytes of Immortal Space Free: b319d8
1STHEAPALLOC  Bytes of Immortal Space Allocated: 1000000
NULL
1STSEGTYPE     Internal Memory
<< lines omitted... >>
1STSEGTYPE    Object Memory
NULL          segment  start    alloc    end      type     bytes
1STSEGSUBTYPE  Immortal Segment  ID=0809C68C
1STSEGMENT     0809BE80 B281C008 B381C008 B381C008  00001008 1000000
1STSEGSUBTYPE  Heap Segment  ID=0809C670
1STSEGMENT     0809BE08 B381D008 B781D008 B781D008  00000009 4000000
NULL
1STSEGTYPE     Class Memory
NULL          segment  start    alloc    end      type     bytes
1STSEGMENT     08158154 083FFD68 083FFEF0 08407D68  00010040 8004
```

El volcado de almacenamiento dinámico es un archivo binario comprimido que contiene una lista con todos los objetos con su clase de objeto, su tamaño y sus referencias. Analice el volcado de almacenamiento dinámico utilizando la herramienta IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer, disponible para su descarga desde IBM Support Assistant (ISA).

Puede utilizar las ubicaciones de memoria de objetos que muestra MDD4J para determinar el espacio de memoria en el que se encuentra un objeto. Direcciones en el rango 0xB28*nnnnn* están en el área de memoria inmortal. Las direcciones del rango 0xB61*nnnnn* están en el almacenamiento dinámico normal.

Cómo evitar fugas de memoria

El colector de basura no procesa áreas de memoria inmortales o con ámbito. En el caso de la memoria inmortal, la memoria solo se libera cuando la máquina virtual Java se libera. Las áreas de la memoria de ámbito se liberan solo si sus recuentos de referencias se ponen a cero. Las tareas de larga duración que se ejecutan en estos contextos se deben escribir de tal manera que, cuando la tarea esté preparada, no se asigne memoria adicional del área de memoria inmortal.

La carga de clases utiliza una pequeña cantidad de memoria inmortal. Estas clases no pasan por la recogida de basura en el entorno de tiempo real. Por lo tanto, la carga de clases que la aplicación no necesita puede provocar que la aplicación utilice más memoria inmortal de la necesaria.

Si su aplicación contiene clases que implementan la interfaz Serializable, ajuste el tamaño inicial de la memoria inmortal a la cuenta de ocupación de las clases generadas. Cada constructor tiene un objeto generado por clase, con el formato "GeneratedSerializationConstructorAccessorXXX" (donde XXX es un número) que se carga en la memoria inmortal la primera vez que se serializa el objeto.

Evite el uso de memoria inmortal, ya que los objetos asignados desde memoria inmortal no los puede recopilar la recogida de basura. Considere la agrupación de objetos en la memoria inmortal si el área de memoria inmortal se está usando con frecuencia.

Asignación de memoria oculta mediante características de lenguaje

En un contexto de memoria inmortal o de ámbito, evite la característica de lenguaje de los argumentos de variable, ya que estos métodos asignan memoria oculta.

Argumentos de variable (vararg)

El lenguaje Java implementa argumentos de variable pasándolos al método como una matriz. El compilador hace que llamar a los métodos de argumentos de variable resulte fácil al crear e inicializar la matriz por el usuario.

La memoria se puede perder si se llama a un método de argumento de variable en un contexto de memoria inmortal o de ámbito. No utilice argumentos de variable en contextos de memoria inmortal o de ámbito. En su lugar, cree explícitamente una matriz y úsela en lugar de los argumentos de variable.

A continuación se muestran dos ejemplos que muestran dos formas equivalentes de llamar a un método de argumento de variable:

```
public class VarargEx {

    public static void main(String[] args) {
        System.out.println("Sum: "+ sum(1.0, 2.0 , 3.0, 4.0));

     }
    static double sum(double... params) {
        double total=0.0;

        for(double num : params) {
            total += num;
        }

        return total;
    }
}
public class VarargEx {

    public static void main(String[] args) {
        double array[] = new double[4];

        array[0] = 1.0; array[1] = 2.0; array[2] = 3.0; array[3] = 4.0;
        System.out.println("Sum: " + sum(array));
    }
```

```
    static double sum(double... params) {
        double total=0.0;

        for(double num : params) {
            total += num;
        }

        return total;
    }
}
```

El segundo ejemplo es el recomendado. Como la asignación de matriz doble pasa a estar visible en el código, la asignación se puede dirigir a un área de memoria concreta.

Concatenación de series

La adición a una serie para producir una serie más larga se implementa utilizando objetos java.lang.StringBuilder, que requieren asignaciones de memoria.

Autoboxing

La función Autoboxing implica la creación de un objeto que contenga un tipo básico, que requiere asignaciones de memoria.

Utilización del reflejo en contextos de memoria

Si un objeto constructor se ha creado en un área de memoria de ámbito, solo se puede utilizar en el mismo ámbito o en un ámbito interno. Todo intento de utilizar dicho objeto constructor en un contexto de memoria inmortal, de almacenamiento dinámico o de ámbito externo fallará.

La excepción generada cuando se produce un reflejo en contextos de memoria es similar a la salida siguiente:

```
Exception in thread "NoHeapRealtimeThread-14" javax.realtime.IllegalAssignmentError
   at java.lang.reflect.Constructor$1.<init>(Constructor.java:570)
   at java.lang.reflect.Constructor.acquireConstructorAccessor(Constructor.java:568)
   at java.lang.reflect.Constructor.newInstance(Constructor.java:521)
   at testMain$TestRunnable$1.run(testMain.java:40)
   at javax.realtime.MemoryArea.activateNewArea(MemoryArea.java:597)
   at javax.realtime.MemoryArea.doExecuteInArea(MemoryArea.java:612)
   at javax.realtime.ImmortalMemory.executeInArea(ImmortalMemory.java:77)
   at testMain$TestRunnable.allocate(testMain.java:36)
   at testMain$TestRunnable.run(testMain.java:12)
   at java.lang.Thread.run(Thread.java:875)
   at javax.realtime.ScopedMemory.runEnterLogic(ScopedMemory.java:280)
   at javax.realtime.MemoryArea.enter(MemoryArea.java:159)
   at javax.realtime.ScopedMemory.enterAreaWithCleanup(ScopedMemory.java:194)
   at javax.realtime.ScopedMemory.enter(ScopedMemory.java:186)
   at javax.realtime.RealtimeThread.runImpl(RealtimeThread.java:1824)
```

Es posible solucionar temporalmente esta restricción utilizando el constructor en el mismo ámbito en el que se ha asignado.

Utilización de clases internas con áreas de memoria de ámbito

Si utiliza las clases internas en el contexto de áreas de memoria de ámbito, debe tener cuidado al crear instancias de objetos de clase internos si los objetos internos y externos se encuentran en distintas áreas de memoria. Se generará una excepción

IllegalAssignmentError del código generado por el compilador que no resulta visible en el código fuente original, si el objeto interno no es capaz de almacenar una referencia al objeto externo.

Un objeto de clase interna debe ser capaz de almacenar una referencia implícita a su objeto de clase externa. Si la referencia infringe las reglas de referencia de la memoria RTSJ, se genera una excepción IllegalAssignmentError.

La mayoría de las clases internas (como las clases internas locales y anónimas) contienen un campo no estático generado por el compilador (sintético) para la instancia de la clase externa léxicamente incluida. La única excepción se produce cuando una instancia de clase interna no tiene un objeto externo incluido, como un objeto de clase anónima con una instancia en un bloque inicializador estático. El campo sintético del objeto interno contendrá una referencia al objeto externo. Este se implementa mediante el compilador por comodidad para el programador de Java. Este campo no resultará visible en el código fuente original, aunque es posible escribir código parecido utilizando las clases anidadas estáticas con una referencia visible. Si la referencia implícita infringe las reglas de área de memoria RTSJ, se generará la excepción IllegalAssignmentError, cuando se cree el objeto interno, a medida que trata de almacenar la referencia al objeto externo.

Por lo general, no puede infringir las reglas de la referencia de memoria RTSJ mientras utiliza clases internas. No puede crear un objeto interno si una referencia al objeto externo asociado infringe las reglas de la referencia de memoria RTSJ. Esta regla implica que un objeto interno asignado en la memoria inmortal o el almacenamiento dinámico no puede tener una referencia a un objeto externo desde la memoria de ámbito. Un objeto interno de la memoria de ámbito puede tener una referencia a un objeto externo desde la memoria de ámbito, pero el objeto externo se debe haber asignado desde la misma área de memoria de ámbito o desde un área de memoria de ámbito externa.

Hay soluciones temporales, como las siguientes:

- utilizar clases anidadas estáticas para eliminar la referencia implícita
- seleccionar áreas de memoria para garantizar que las relaciones de objetos externos e internos no infrinjan las restricciones de la referencia al área de memoria

Utilización de las herramientas de diagnóstico

Hay una serie de herramientas de diagnóstico disponibles que ayudan a diagnosticar problemas con la JVM de IBM WebSphere Real Time for RT Linux.

El SDK de IBM para Java 7 proporciona una serie de herramientas de diagnóstico que se pueden usar para diagnosticar problemas con la JVM de IBM WebSphere Real Time for RT Linux. En esta sección se presentan las herramientas que hay disponibles, y se proporcionan enlaces a más información sobre el uso de las herramientas.

Hay un punto importante que recordar cuando se utilizan las herramientas de diagnóstico de SDK. Cuando invoque la JVM en tiempo real, utilice la opción siguiente:

```
java -Xrealtime
```

Esta opción se debe utilizar cuando se ejecutan herramientas de diagnóstico para la JVM en tiempo real. Por ejemplo, para mostrar los agentes de volcado registrados para la JVM de IBM WebSphere Real Time for RT Linux, escriba:

```
java -Xrealtime -Xdump:what
```

El resto de diferencias en el uso de estas herramientas con IBM WebSphere Real Time for RT Linux se proporciona aquí como información suplementaria, junto con salida de ejemplo, como ayuda al diagnóstico.

Para un resumen de la información de diagnóstico generada por el SDK de IBM para Java 7, consulte Resumen de información de diagnóstico.

Uso de IBM Monitoring and Diagnostic Tools for Java

IBM proporciona herramientas y documentación como ayuda para comprender, supervisar y diagnosticar problemas con aplicaciones que utilizan IBM JRE.

Están disponibles las herramientas siguientes:

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

Garbage Collection and Memory Visualizer

Garbage Collection and Memory Visualizer (GCMV) le ayuda a conocer el uso de memoria, el comportamiento de la recogida de basura y el rendimiento de las aplicaciones Java.

GCMV analiza y traza datos desde distintos tipos de registro, entre los que se incluyen:

- Registros cronológicos detallados de la recopilación de basura
- Registros de recopilación de basura de rastreo, generados mediante el parámetro -Xtgc.
- Registros de memoria nativa, generados mediante los mandatos del sistema ps, **svmon** o **perfmon**.

La herramienta ayuda a diagnosticar problemas como fugas de memoria y a analizar los datos en distintos formatos visuales, y proporciona recomendaciones de ajuste.

GCMV se proporciona como un complemento de IBM Support Assistant (ISA). Para obtener información sobre cómo instalar y cómo empezar con el complemento, consulte: http://www.ibm.com/developerworks/java/jdk/tools/gcmv/.

En el IBM Information Center hay disponible más información sobre GCMV.

Health Center

Health Center es una herramienta de diagnóstico para supervisar el estado de una Java Virtual Machine (JVM) en ejecución.

La herramienta contiene dos partes:

- El agente de Health Center que recopila datos de una aplicación en ejecución.

- Un cliente basado en la plataforma Eclipse que se conecta al agente. El cliente interpreta los datos y realiza recomendaciones para mejorar el rendimiento de la aplicación supervisada.

Health Center se proporciona como un complemento de IBM Support Assistant (ISA). Para obtener información sobre cómo instalar y cómo empezar con el complemento, consulte: http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/.

En el IBM Information Center hay disponible más información sobre Health Center.

Interactive Diagnostic Data Explorer

Interactive Diagnostic Data Explorer (IDDE) es una alternativa basada en GUI al visor de volcado (mandato **jdmpview**). IDDE proporciona la misma funcionalidad que el visor de volcado, pero con soporte adicional, como la posibilidad de guardar la salida del mandato.

Utilice IDDE para explorar y examinar más fácilmente archivos de volcado generados por la JVM. En IDDE, se especifican mandatos en un registro de investigación para explorar el archivo de volcado. El soporte que se proporciona con el registro de investigación incluye los elementos siguientes:

- Asistencia de mandatos
- Cumplimentación automática del texto y algunos parámetros como nombres de clase
- La posibilidad de guardar mandatos y salida, que podrá enviar entonces a otras personas.
- Texto resaltado y señalización de problemas
- La posibilidad de añadir sus propios comentarios
- Soporte para utilizar el analizador de memoria desde IDDE

IDDE se proporciona como un complemento de IBM Support Assistant (ISA). Para obtener más información sobre cómo instalar el complemento e iniciarse en él, consulte la visión general de IDDE en developerWorks.

Hay disponible más información sobre IDDE en el Information Center de IBM.

Memory Analyzer

Memory Analyzer le ayuda a analizar el almacenamiento dinámico de Java mediante volcados de almacenamiento dinámico portátil (PHD, Portable Heap Dump) y volcados de nivel de sistema operativo.

Esta herramienta puede analizar volcados que contienen millones de objetos y proporciona la siguiente información:

- Los tamaños retenidos de los objetos.
- Los procesos que están impidiendo que el recopilador de basura recopile los objetos.
- Un informe para extraer automáticamente los sucesos que pueden estar causando fugas de memoria.

Esta herramienta se basa en el proyecto Eclipse Memory Analyzer (MAT) y utiliza la característica IBM Diagnostic Tool Framework for Java (DTFJ) para permitir el proceso de volcados de las JVM de IBM.

Memory Analyzer se proporciona como un complemento de IBM Support Assistant (ISA). Para obtener información sobre cómo instalar y cómo empezar con el complemento, consulte: http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/.

En el IBM Information Center hay disponible más información sobre Memory Analyzer.

Uso de agentes de volcado

Los agentes de volcado se configuran durante la inicialización de la JVM. Le permiten utilizar sucesos que se producen en la JVM como, por ejemplo, la recopilación de basura, el inicio de hebras o la terminación de la JVM, para iniciar volcados o iniciar una herramienta externa.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre agentes de volcado, y cubre:

- Uso de la opción **-Xdump**
- Agentes de volcado
- Sucesos de volcado
- Control avanzado de agentes de volcado
- Símbolos de agente de volcado
- Agentes de volcado predeterminados
- Eliminación de agentes de volcado
- Variables de entorno de agente de volcado
- Correlaciones de señales
- Ubicaciones predeterminadas del agente de volcado

Puede encontrar esta información aquí: IBM SDK for Java 7 - Using dump agents.

Aquí se proporciona información adicional para IBM WebSphere Real Time for RT Linux:

Sucesos de volcado

Los agentes de volcado son desencadenados por sucesos que se producen durante el funcionamiento de la JVM. Para IBM WebSphere Real Time for RT Linux, el valor predeterminado del suceso lento es de 5 milisegundos.

Algunos sucesos se pueden filtrar para mejorar la relevancia de la salida. Consulte “Opción filter” en la página 127 para obtener más información.

Nota: Los sucesos unload y expand actualmente no se producen en WebSphere Real Time. Las clases están en memoria inmortal y no se pueden descargar.

Nota: Los sucesos gpf y abort no pueden desencadenar un volcado de almacenamiento dinámico, prepare el almacenamiento dinámico (request=prepwalk), o compáctelo (request=compact).

La siguiente tabla muestra sucesos disponibles cuando el agente de volcado se desencadena:

Suceso	Desencadenado cuando...	Operación de filtro
gpf	Se produce un error de protección general.	

Suceso	Desencadenado cuando...	Operación de filtro
user	La JVM recibe la señal SIGQUIT del sistema operativo.	
abort	La JVM recibe la señal SIGABRT del sistema operativo.	
vmstart	La máquina virtual se inicia.	
vmstop	La máquina virtual se detiene.	Filtra código de salida; por ejemplo, **filter=#129..#192#-42#255**
load	Se carga una clase.	Filtra nombre de clase; por ejemplo, **filter=java/lang/String**
unload	Se descarga una clase.	
throw	Se emite una excepción.	Filtra nombre de clase de excepción; por ejemplo, **filter=java/lang/OutOfMem***
catch	Se detecta una excepción.	Filtra nombre de clase de excepción; por ejemplo, **filter=*Memory***
uncaught	La aplicación no detecta una excepción de Java.	Filtra nombre de clase de excepción; por ejemplo, **filter=*MemoryError**
systhrow	La JVM está a punto de emitir una excepción de Java. Es diferente del suceso "throw" porque solo se desencadena para condiciones de error detectadas internamente en la JVM.	Filtra nombre de clase de excepción; por ejemplo, **filter=java/lang/OutOfMem***
thrstart	Se inicia una nueva hebra.	
blocked	Se bloquea una hebra.	
thrstop	Se detiene una hebra.	
fullgc	Se inicia un ciclo de recopilación de basura.	
slow	Una hebra tarda más de 5 ms en responder a una solicitud interna de la JVM.	Cambia el tiempo que tarda un suceso en considerarse lento; por ejemplo, **filter=#300ms** se desencadenará cuando una hebra tarde más de 300 ms en responder a una solicitud interna de la JVM.
allocation	Un objeto Java se asigna con un tamaño que coincide con una especificación de filtro determinada.	Filtra tamaño de objeto; debe suministrarse un filtro. Por ejemplo, se desencadenará **filter=#5m** en objetos superiores a 5 Mb. También se da soporte a rangos; por ejemplo, se desencadenará **filter=#256k..512k** en objetos entre 256 Kb y 512 Kb en tamaño.
traceassert	Se ha producido un error interno en la JVM	No aplicable.
corruptcache	La JVM encuentra que la memoria caché de clases compartidas está dañada.	No aplicable.

Opción filter

Algunos sucesos de la JVM se producen miles de veces durante el tiempo de vida de una aplicación. Los agentes de volcado pueden utilizar filtros y rangos para evitar que se produzcan excesivos volcados.

Comodines

Puede utilizar un comodín en su filtro de sucesos de excepción colocando un asterisco sólo al principio o al final del filtro. El siguiente mandato no funciona porque el segundo asterisco no está al final:

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

Para que funcione este filtro, debe cambiarlo a:

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

Sucesos de carga de clases y excepción

Puede filtrar los sucesos de carga de clase (load) y excepción (throw, catch, uncaught, systhrow) por nombre de clase Java:

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

Puede filtrar los sucesos de excepción throw, uncaught y systhrow por nombre de método Java:

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

Las partes opcionales aparecen entre paréntesis.

Puede filtrar los sucesos de excepción catch por nombre de método Java:

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

Las partes opcionales aparecen entre corchetes.

Suceso vmstop

Puede filtrar el suceso JVM shut down utilizando uno o varios códigos de salida:

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

Suceso slow

Puede filtrar el suceso slow para cambiar el umbral de tiempo del valor predeterminado de 5 ms:

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

No puede establecer el filtro en un tiempo inferior al tiempo predeterminado.

Suceso allocation

Debe filtrar el suceso allocation para especificar el tamaño de los objetos que causan un desencadenante. Puede establecer el tamaño de filtro de cero al máximo valor de un puntero de 32 bits en plataformas de 32 bits, o al valor máximo de un puntero de 64 bits en plataformas de 64 bits. Si establece el valor de filtro inferior en cero desencadenará un volcado en todas las ubicaciones.

Por ejemplo, para desencadenar volcados en asignaciones superiores a 5 Mb de tamaño, utilice:

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

Para desencadenar volcados en asignaciones entre 256 Kb y 512 Kb en tamaño, utilice:

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

Otros sucesos

Si aplica un filtro a un suceso que no soporta filtrado, el filtro se ignorará.

Opción request

Utilice la opción request para pedir a la JVM que prepare el estado antes de iniciar el agente de volcado. Para IBM WebSphere Real Time for RT Linux hay una opción de solicitud adicional; **multiple**.

Las opciones disponibles se muestran en la siguiente tabla:

Valor de opción	Descripción
exclusive	Solicitar acceso exclusivo a la JVM.
compact	Ejecutar la recopilación de basura. Esta opción elimina todos los objetos inaccesibles del almacenamiento dinámico antes de que se genere el volcado.
prepwalk	Preparar el almacenamiento dinámico para examen. Debe también especificar **exclusive** al utilizar esta opción.
serial	Suspender otros volcados hasta que éste se haya terminado.
multiple	Producir volcados de almacenamiento dinámico independientes para cada área de memoria RTSJ.
preempt	Se aplica al agente de volcado de Java y controla si las hebras nativas del proceso se vacían o no previamente para recopilar rastreos de pila. Si esta opción no está especificada, solo se recopilarán rastreos de pila de Java en el volcado Java.

Por ejemplo, la configuración predeterminada para la opción solicitada para Javadump es `request=exclusive+preempt`. Para cambiar la configuración de forma que los Javadumps se produzcan sin vaciar las hebras previamente para recopilar los rastreos de pilas nativas, utilice la opción siguiente:

```
-Xdump:java:request=exclusive
```

En general, las opciones request predeterminadas son suficientes.

Puede especificar más de una opción request utilizando +. Por ejemplo:

```
-Xdump:heap:request=exclusive+compact+prepwalk
```

Uso del volcado Java

El volcado Java produce archivos que contienen información de diagnóstico relacionada con la JVM y una aplicación Java capturada en un punto durante la ejecución. Por ejemplo, la información puede ser sobre el sistema operativo, el entorno de aplicación, hebras, pilas, bloqueos y memoria.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre volcados Java, y cubre:

- Habilitación de un volcado Java
- Cómo se desencadena un volcado Java
- Interpretación de un volcado Java
- Variables de entorno y volcado Java

Puede encontrar dicha información aquí: IBM SDK for Java 7 - Using Javadump.

En los temas siguientes se proporciona información suplementaria y salida de ejemplo para IBM WebSphere Real Time for RT Linux.

Gestión del almacenamiento (MEMINFO)

La sección MEMINFO proporciona información sobre el Gestor de memoria, incluyendo las áreas de memoria de almacenamiento dinámico, inmortal y con ámbito.

La sección MEMINFO de un volcado Java proporciona información sobre el gestor de memoria. Consulte Uso del recopilador de basura Metronome para obtener detalles sobre cómo funciona el componente gestor de memoria.

Esta parte del volcados Java proporciona varios valores de gestión de almacenamiento, incluyendo:

- cantidad de memoria libre
- cantidad de memoria usada
- tamaño actual del almacenamiento dinámico
- tamaño actual de las áreas de memoria inmortal
- tamaño actual de las áreas de memoria con ámbito

Esta sección también contiene datos históricos sobre la recogida de basura. Los datos aparecen como secuencia de puntos de rastreo, cada uno con indicación de fecha y hora, ordenados con el punto de rastreo más reciente primero.

Los volcados Java producidos por la JVM estándar contienen una sección "GC History". Esta información no está contenida en los volcados Java producidos cuando se utiliza la JVM de tiempo real. Utilice la opción **-verbose:gc** o el rastreo breve de la JVM para obtener información sobre el comportamiento de la recopilación de basura (GC). Para obtener más detalles, consulte "Utilización de la información de verbose:gc" en la página 148 y la sección de agentes de volcado de la Guía de usuario de IBM SDK para Java V7.

Si ejecuta un programa que utiliza memoria de ámbito, y se emite un error OutOfMemoryError, algunas de las áreas de memoria listadas en el volcado Java podrían parecer vacías. Cuando un ámbito anidado dentro de otro se queda sin memoria, el ámbito interno se podría eliminar para cuando se haya generado el volcado Java. Para obtener información relacionada con el estado de las áreas de memoria durante la emisión del error OutOfMemoryError, ejecute el programa con la siguiente opción de línea de mandatos:

```
-Xdump:java:events=throw,filter=java/lang/OutOfMemoryError,range=1..1
```

Este mandato genera un volcado Java adicional al emitirse un error OutOfMemoryError, en lugar de cuando se detecta una excepción no detectada, cosa que ocurre poco después. En este volcado Java puede ver todas las áreas de memoria que estaban activas en el momento en que se emitió el error OutOfMemoryError, incluidos los ámbitos internos. Para obtener más información sobre cómo utilizar la opción **-Xdump**, consulte la Guía de usuario de IBM SDK para Java V7.

En un volcado Java, los segmentos son bloques de memoria asignados por el tiempo de ejecución de Java para que las tareas utilicen más cantidad de memoria. Estas son algunas tareas de ejemplo:

- mantenimiento de memorias caché JIT
- almacenamiento de clases Java

El entorno de tiempo de ejecución Java también asigna otra memoria nativa que no aparece en la lista de la sección MEMINFO. El total de la memoria utilizada por los segmentos de Java Runtime no representa necesariamente toda la ocupación de memoria de Java Runtime. Un segmento de Java Runtime consiste en la estructura de datos del segmento y un bloque de memoria nativa asociado.

El siguiente ejemplo muestra una salida normal. Todos los valores se proporcionan como valores hexadecimales. Las cabeceras de columna en la sección MEMINFO significan lo siguiente:

- Sección de memoria de objeto (HEAPTYPE):

  id El ID del espacio o región.

  start La dirección inicial de esta región del almacenamiento dinámico.

  end La dirección final de esta región del almacenamiento dinámico.

  size El tamaño de esta región del almacenamiento dinámico.

  space/region
  Para una línea que contiene solo un id y un nombre, esta columna muestra el nombre del espacio de memoria. De lo contrario, la columna muestra el nombre del espacio de memoria seguido del nombre de una región determinada incluida dentro de ese espacio de memoria.

- Sección de memoria interna (SEGTYPE), incluidas memoria de clase, memoria caché de código JIT y memoria caché de datos JIT:

  segment
  La dirección de la estructura de datos de control del segmento.

  start La dirección inicial del segmento de memoria nativa.

  alloc La dirección de asignación actual dentro del segmento de memoria nativa.

  end La dirección final del segmento de memoria nativa.

  type Un campo de bits interno que describe las características del segmento de memoria nativa.

  size El tamaño del segmento de memoria nativa.

```
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
       (líneas eliminadas para facilitar la lectura)
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
        (líneas eliminadas para facilitar la lectura)
```

```
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
           (líneas eliminadas para facilitar la lectura)
NULL
NULL           ------------------------------------------------------------------------
```

Hebras y rastreo de la pila (THREADS)

Para el programador de aplicaciones, una de las partes más útiles de un volcado Java es la sección THREADS. En esta sección se muestra una lista de hebras de Java, hebras nativas y seguimientos de la pila. Para IBM WebSphere Real Time for RT Linux, también se muestran hebras en tiempo real y hebras que no son en tiempo real.

Una hebra Java es implementada por una hebra nativa del sistema operativo. Cada hebra se representa por un conjunto de líneas como las siguientes:

```
"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)
```

Los nombres de hebras de Java se pueden ver en el sistema operativo cuando se utiliza el mandato **ps**. Para obtener más información sobre cómo utilizar el mandato **ps**, consulte “Técnicas de depuración generales” en la página 108.

Las propiedades de la primera línea son el nombre de hebra, la dirección de las estructuras de la JVM y del objeto, el estado y la prioridad de la hebra de Java thread object, thread state, and Java. Las propiedades de la segunda línea son el ID de hebra de sistema operativo nativo, la prioridad de hebra de sistema operativo nativo y la política de planificación del sistema operativo nativo.

Los nombres de hebra son visibles de tres maneras:

- Listados en archivos javacore. No todas las hebras aparecen en archivos javacore.
- Al listar hebras del sistema operativo utilizando el mandato **ps**.
- Al utilizar el método java.lang.Thread.getName().

La tabla siguiente proporciona información sobre los nombres de hebras de IBM WebSphere Real Time for RT Linux.

Tabla 13. Nombres de hebra en IBM WebSphere Real Time for RT Linux

Detalle de la hebra	Nombre de hebra
Una hebra interna de la JVM que utiliza la recopilación de basura para asignar la finalización de objetos por hebras secundarias.	`Maestro finalizador`
La hebra de alarma que utiliza el recopilador de basura (GC).	`Alarma del GC`
Las hebras esclavas que se utilizan para la recopilación de basura.	`Esclava del GC`
Una hebra interna de la JVM que utiliza el módulo del compilador JIT (just-in-time) para obtener muestras del uso de los métodos en la aplicación.	`IProfiler`
Una hebra que utiliza la máquina virtual (VM) para gestionar las señales que recibe la aplicación, tanto si se generan interna como externamente.	`Informador de señales`
Una hebra de la JVM interna que se utiliza para compilar código Java.	`JIT Compilation Thread`

Tabla 13. Nombres de hebra en IBM WebSphere Real Time for RT Linux (continuación)

Detalle de la hebra	Nombre de hebra
Una hebra de la JVM interna que se utiliza para permitir que los agentes JVMTI se conecten a una JVM en ejecución.	`Attach API wait loop`

Los nombres predeterminados de las hebras en tiempo real (javax.realtime.RealtimeThread) creadas en código Java son `RTThread-x`, donde "`x`" es el número de la hebra.

Los nombres predeterminados de las hebras en tiempo real de almacenamiento no dinámico son `NHRTThread-x`, donde "`x`" es el número de hebra.

La prioridad de hebra Java se correlaciona con un valor de prioridad de sistema operativo en función de la plataforma. Un valor grande para la prioridad de hebra Java significa que la hebra tiene una prioridad alta. Es decir, la hebra se ejecuta con más frecuencia que las hebras de prioridad inferior. Para obtener más detalles sobre cómo funciona esto para hebras Java, hebras en tiempo real y hebras en tiempo real que no son del almacenamiento dinámico, consulte "Correlación y herencia de prioridades" en la página 15.

Los valores de estado pueden ser:

- R (Runnable): Ejecutable; la hebra puede ejecutarse cuando tenga oportunidad.
- CW (Condition Wait): Condición de espera; la hebra está esperando. Por ejemplo, porque:
  - Se ha realizado una llamada sleep()
  - Se ha bloqueado la hebra para E/S
  - Se ha llamado al método wait() para que espere en un supervisor que se está notificando
  - La hebra está sincronizándose con otra hebra con una llamada join()
- S (Suspended): Suspendida; la hebra ha sido suspendida por otra hebra.
- Z (Zombie): Inerte; la hebra se ha detenido.
- P (Parked): Aparcada; la hebra ha sido aparcada por la nueva API de simultaneidad (java.util.concurrent).
- B (Blocked): Bloqueada; la hebra está esperando para obtener un bloqueo que posee actualmente otra cosa.

Si una hebra está aparcada o bloquead, la información de salida contendrá una línea correspondiente a dicha hebra, comenzando por `3XMTHREADBLOCK`, indicando el recurso que esa hebra está esperando y, cuando sea posible, indicando la hebra que actualmente posee dicho recurso. Para obtener más información, consulte el tema sobre hebras bloqueadas en la Guía de usuario IBM SDK para Java V7.

Cuando se inicia un volcado de Java para obtener información sobre un diagnóstico, la JVM desactiva temporalmente las hebras de Java antes de generar el archivo javacore. Un estado de preparación de `exclusive_vm_access` se muestra en la línea `1TIPREPSTATE` del apartado `TITLE`.

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

Las hebras que están ejecutando código Java cuando se desencadenó el javacore están es estado CW (Condición de espera).

```
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)
```

El apartado The LOCKS del javacore muestra que están hebras están esperando en un bloqueo interno de la JVM.

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

Utilizar el vuelco de almacenamiento dinámico

El término Heapdump describe el mecanismo de máquina virtual de IBM para Java que genera un volcado de todos los objetos activos que están en el almacenamiento dinámico de Java; es decir, los que se utilizan ejecutando la aplicación Java.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre volcados de almacenamiento dinámico, y cubre:

- Obtención de volcados de almacenamiento dinámico
- Herramientas para procesar volcados de almacenamiento dinámico
- Uso de **-Xverbose:gc** para obtener información de almacenamiento dinámico
- Variables de entorno y Heapdump
- Formato de archivo Heapdump de texto (clásico)
- Formato de archivo de vuelco de almacenamiento dinámico portátil (PHD)

Puede encontrar esta información en: IBM SDK for Java 7 - Using Heapdump.

Información suplementaria para IBM WebSphere Real Time for RT Linux:

Habilitación de varios volcados de almacenamiento dinámico para JVM en tiempo real

El volcado de almacenamiento dinámico generado es, de forma predeterminada, un único archivo que contiene información sobre todos los objetos Java de todas las áreas de memoria, la memoria de almacenamiento dinámico, la memoria inmortal y la memoria de ámbito. El principal motivo para producir varios volcados es que cada área de almacenamiento dinámico individual se puede analizar usando las herramientas de volcado de almacenamiento dinámico tradicionales sin modificaciones.

Acerca de esta tarea

De forma predeterminada, los volcados de almacenamiento dinámico contienen información sobre todos los objetos de las áreas de memoria de la JVM, el almacenamiento dinámico, la memoria inmortal y de ámbito. Puede obtener distintos volcados de almacenamiento dinámico con información sobre los objetos Java de cada área de memoria utilizando la opción **request=multiple** con **-Xdump:heap**. Tenga en cuenta que debe repetir también la configuración predeterminada de la opción de solicitud, por lo que debe especificar `request=multiple+exclusive+prepwalk+compact`. Esto produce un conjunto de volcados de almacenamiento dinámico con un campo adicional en el nombre que indica el área de memoria específica:

```
heapdump.%id.%Y%m%d.%H%M%S.%pid.phd
```

donde *%id* identifica el archivo de volcado de almacenamiento dinámico que contiene objetos en la memoria de almacenamiento dinámico, la memoria inmortal o un área específica de memoria de ámbito.

Hay 4 tipos de almacenamientos dinámicos representados por los nombres siguientes: “Predeterminado”, “Inmortal”, “Ámbito” y “Otros”. El código del volcado de almacenamiento dinámico sustituye el valor %id de la etiqueta del almacenamiento dinámico por uno de estos nombres concatenados con un identificador (normalmente numérico), por ejemplo: `heapdump.Inmortal12994208.20060807.093653.7684.txt`.

Ejemplo

```
java -Xrealtime -Xdump:heap:defaults:request=multiple+exclusive+compact+prepwalk
     <java program>
```

Mediante esta opción adicional se producen varios volcados de almacenamiento dinámico en el formato de almacenamiento dinámico portátil (phd).

```
java -Xrealtime -Xdump:heap:defaults:request=multiple+exclusive+compact+prepwalk,
     opts=CLASSIC <java program>
```

Mediante esta opción adicional, se producen varios volcados de almacenamiento dinámico en el formato de texto CLASSIC.

Mediante la opción -Xdump:what se muestran los agentes de volcado en el inicio de la máquina virtual Java, y resulta útil para comprobar las opciones de volcado existentes.

Formato de archivo Heapdump de texto (clásico)

El Heapdump de texto o clásico es una lista de todas las instancias de objeto del almacenamiento dinámico, que incluyen el tipo de objeto, el tamaño y las referencias entre objetos.

Registro de cabecera

El registro de cabecera es un único registro que contiene una serie de información de la versión.

```
// Version: <serie de versión que contiene el nivel del SDK, nivel de build de la JVM y la plataforma
```

Ejemplo:

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

Registros de objetos

Los registros de objetos son varios registros, uno para cada instancia de objeto del almacenamiento dinámico, que proporcionan la dirección de objeto, el tamaño, el tipo y las referencias del objeto.

```
<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> ...
```

Las referencias de dirección de objeto y almacenamiento dinámico están en el almacenamiento dinámico pero la dirección de bloque de clases está fuera del almacenamiento dinámico. Todas las referencias encontradas en la instancia de objeto aparecen, incluidas las referencias de valores nulos. El tipo de objeto es un nombre de clase que incluye paquete o una matriz primitiva o tipo de matriz de clase, mostrada mediante su firma de tipo de JVM estándar; consulte “Firmas de

tipo de la máquina virtual Java" en la página 138. Los registros de objetos también pueden contener referencias de bloque de clases adicionales, normalmente en el caso de las instancias de clase de reflejo.

Ejemplos:

Una instancia de objeto, longitud de 28 bytes, de tipo java/lang/String:

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

Una dirección de bloque de clases de java/lang/String, seguida de una referencia a la instancia de matriz de caracteres:

```
0x415319D8 0x00436EB0
```

Una instancia de objeto, longitud de 44 bytes, de tipo matriz de caracteres:

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

Una dirección de bloque de clases de matriz de caracteres:

```
0x41530F20
```

Un objeto de tipo matriz de java/util/Hashtable Entry inner class:

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

Un objeto de tipo java/util/Hashtable Entry inner class:

```
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
```

Una dirección de bloque de clases y referencias de almacenamiento dinámico, incluidas las referencias nulas:

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

Registros de clases

Los registros de clases son varios registros, uno de cada clase cargada, que proporcionan la dirección de bloque de clases, el tamaño, el tipo y las referencias de la clase.

```
<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>...
```

La dirección de bloque de clases y las referencias de bloque de clases están fuera del almacenamiento dinámico, pero el registro de clases también puede contener referencias en el almacenamiento dinámico, normalmente para miembros de datos de clase estática. Todas las referencias encontradas en el bloque de clases aparecen, incluidas las de valores nulos. El tipo de clase es un nombre de clase que incluye paquete o una matriz primitiva o tipo de matriz de clase, mostrada mediante su firma de tipo de JVM estándar; consulte "Firmas de tipo de la máquina virtual Java" en la página 138.

Ejemplos:

Un bloque de clases, longitud de 32 bytes, para clase java/lang/Runnable:

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

Referencias a otros bloques de clases y referencias de almacenamiento dinámico, incluidas las referencias nulas:

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

Un bloque de clases, longitud de 168 bytes, para clase java/lang/Math:

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

Registro de cola 1

El registro de cola 1 es un registro único que contiene recuentos de registros.

```
// 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>
```

Ejemplo:

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

Registro de cola 2

El registro de cola 2 es un registro único que contiene totales.

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

Ejemplo:

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

Firmas de tipo de la máquina virtual Java

Las firmas de tipo de la máquina virtual Java son abreviaciones de los tipos de Java que aparecen en la siguiente tabla:

Firmas de tipo de la máquina virtual Java	Tipo de Java
Z	booleano
B	byte
C	carácter
S	corto
I	int
J	largo
F	flotante
D	doble
L *<clase completa>* ;	*<clase completa>*
[*<tipo>*	*<tipo>*[ ] (matriz de *<tipo>*)
(*<arg-tipos>*) *<ret-tipo>*	método

Uso de volcados del sistema y el visor de volcados

La JVM puede generar volcados del sistema nativos, también conocidos como volcados del núcleo, en condiciones configurables. Los volcados del sistema suelen ser grandes. La mayoría de las herramientas usadas para analizar los volcados del sistema también son específicas para cada plataforma.Use la herramienta **gdb** para analizar un volcado del sistema en Linux.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre el uso del visor de volcados y los volcados del sistema, y cubre:

- Visión general de volcados del sistema
- Valores predeterminados de volcado del sistema
- Uso del visor de volcados
  - Uso de **jextract**
  - Problemas con los que se enfrenta el visor de volcados
  - Mandatos disponibles en **jdmpview**
  - Sesión de ejemplo
  - Referencia rápida de mandatos **jdmpview**

Puede encontrar esta información aquí:IBM SDK para Java 7 - Uso del visor de volcados y volcados del sistema.

Información suplementaria para IBM WebSphere Real Time for RT Linux:

Uso de jextract

Cuando se procesa un volcado del sistema de una JVM en tiempo real, debe incluir la opción **-Xrealtime**. Por ejemplo:

```
jextract -Xrealtime <core file name> [<zip_file>]
```

Cuando ejecuta **jextract** en una JVM que sea diferente de aquella en la que se produjo el volcado, aparecen los siguientes mensajes de error:

```
J9RAS.buildID es incorrecto (se ha encontrado e8801ed67d21c6be, se espera eb4173107d21c673).
Esta versión de jextract es incompatible con este volcado.
Se ha detectado anomalía durante jextract, consulte el mensaje o los mensajes anteriores.
```

De la misma forma, este mensaje también se produce si estaba ejecutando Java con la JVM estándar, pero ha usado la opción **-Xrealtime** cuando procesaba el volcado con **jextract**.

Mandatos disponibles en jdmpview

jdmpview es una herramienta de línea de mandatos interactiva para explorar la información desde un volcado del sistema JVM y realizar varias funciones de análisis.

info jitm
Lista los métodos AOT y JIT compilados y sus direcciones:
- Nombre de método y signatura
- Dirección inicial de método
- Dirección final de método

Para ver el resto de opciones de mandato, consulte la Guía de usuario de IBM SDK para Java V7.

Rastreo de aplicaciones Java y la JVM

El rastreo de la máquina virtual Java (JVM) es un recurso de rastreo que se facilita en IBM WebSphere Real Time for RT Linux con una impacto mínimo en el rendimiento. En la mayoría de los casos, los datos de rastreo se conservan en un formato binario compacto, que se puede formatear con el formateador Java que se suministra.

El rastreo está habilitado de forma predeterminada, junto con un pequeño conjunto de puntos de rastreo que se dirigen a las memorias intermedias. Puede habilitar los puntos de rastreo en tiempo de ejecución utilizando niveles, componentes, nombres de grupos o identificadores de puntos de rastreo individuales.

La Guía del usuario de IBM SDK para Java V7 contiene información detallada sobre aplicaciones de rastreo, cubriendo:

- Qué se puede rastrear
- Tipos de punto de rastreo
- Rastreo predeterminado
- Registro de datos de rastreo
- Control del rastreo
- Rastreo de aplicaciones Java
- Rastreo de métodos Java

Cuando rastrea IBM WebSphere Real Time for RT Linux, debe invocar correctamente la JVM en tiempo real cuando incluya las opciones de rastreo. Por ejemplo, cuando especifique opciones de rastreo, escriba:

```
java -Xrealtime -Xtrace:<options>
```

Puede encontrar información de IBM SDK para Java V7 aquí: Rastreo de aplicaciones Java y de la JVM.

Determinación de problemas de JIT y AOT

Puede utilizar opciones de línea de mandatos para ayudar a diagnosticar problemas del compilador JIT y AOT y para ajustar el rendimiento.

Aunque IBM WebSphere Real Time for RT Linux comparte algunos componentes comunes con el IBM SDK para Java V7, el comportamiento de JIT y AOT es distinto. En esta sección se trata la resolución de problemas para incidencias de JIT y AOT en IBM WebSphere Real Time for RT Linux.

Diagnóstico de un problema de JIT o AOT

Ocasionalmente, los códigos de bytes válidos podrían compilarse en código nativo no válido, haciendo que el programa Java falle. Al determinar si el compilador JIT o AOT está defectuoso y, si es así, *dónde* está defectuoso, podrá proporcionar ayuda válida al equipo de servicio de Java.

Acerca de esta tarea

Para determinar qué métodos están compilados al rellenarse la memoria caché de clase compartida, utilice la opción **-Xaot:verbose** de la línea de mandatos de admincache. Por ejemplo:

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

Este apartado describe cómo puede determinar si el problema está relacionado con el compilador. Este apartado también sugiere algunos métodos alternativos posibles y técnicas de depuración para resolver problemas relacionados con el compilador.

Inhabilitación del compilador JIT o AOT:

Si sospecha que se está produciendo un problema en el compilador JIT o AOT, inhabilite la compilación para ver si el problema continúa. Si el problema sigue produciéndose, sabrá que el compilador no es su causa.

Acerca de esta tarea

El compilador JIT está habilitado de forma predeterminada. El compilador AOT también está habilitado, pero, no está activo a menos que estén habilitadas las clases compartidas. Por motivos de eficacia, no todos los métodos de una aplicación Java están compilados. La JVM mantiene un recuento de llamadas para cada método de la aplicación; cada vez que se llama y se interpreta un método, el recuento de llamadas de dicho método se incrementa. Cuando el recuento alcanza el umbral de compilación, el método se compila y se ejecuta de forma nativa.

El mecanismo de recuento de llamadas extiende la compilación de métodos durante toda la vida de la aplicación, dando mayor prioridad a los métodos que se utilizan con más frecuencia. Algunos métodos utilizados con menos frecuencia podrían no compilarse nunca. Como resultado, cuando falla un programa Java, el problema podría estar en el compilador JIT o AOT o podría estar en otro lugar de la JVM.

El primer paso para diagnosticar la anomalía es determinar *dónde* está el problema. Para ello, debe primero ejecutar el programa Java en modalidad puramente interpretada (es decir, con los compiladores JIT y AOT inhabilitados).

Procedimiento

1. Elimine las opciones **-Xjit**y **-Xaot** (y los parámetros que las acompañan) de la línea de mandatos.
2. Utilice la opción de línea de mandatos **-Xint** para inhabilitar los compiladores JIT y AOT. Por motivos de rendimiento, no utilice la opción **-Xint** en un entorno de producción.

Qué hacer a continuación

Si ejecuta el programa Java con la compilación inhabilitada se producirá una de las siguientes situaciones:

- La anomalía continúa. El problema no está en el compilador JIT ni AOT. En algunos casos, el programa podría empezar a fallar de otra manera; sin embargo, el problema no estará relacionado con el compilador.
- La anomalía desaparece. Lo más probable es que el problema esté en el compilador JIT o AOT.

  Si no utiliza clases compartidas, el compilador JIT estará defectuoso. Si utiliza clases compartidas, deberá determinar qué compilador está defectuoso ejecutando su aplicación sólo con la compilación JIT habilitada. Ejecute la aplicación con la opción **-Xnoaot** en lugar de con la opción **-Xint**. Esto le llevará a una de las situaciones siguientes:

– La anomalía continúa. El problema estará en el compilador JIT. También puede utilizar **-Xnojit** en lugar de la opción **-Xnoaot** para asegurarse de que sólo el compilador JIT esté defectuoso.
– La anomalía desaparece. El problema estará en el compilador AOT.

Inhabilitación selectiva del compilador JIT:

Si la anomalía del programa Java apunta a un problema con el compilador JIT, podrá intentar de esta forma reducir más el ámbito del problema.

Acerca de esta tarea

De forma predeterminada, el compilador JIT optimiza métodos en diversos niveles de optimización. Diferentes selecciones de optimizaciones se aplican a distintos métodos, en función de sus recuentos de llamadas. Los métodos a los que se llama con más frecuencia se optimizan en niveles superiores. Al cambiar los parámetros del compilador JIT, puede controlar el nivel de optimización al que se optimizan los métodos. Puede determinar si el optimizador está defectuoso y, si lo está, qué optimización es problemática.

Los parámetros de JIT se especifican como lista separada por comas, añadidos a la opción **-Xjit**. La sintaxis es **-Xjit**:<*param1*>,<*param2*>=<*valor*>. Por ejemplo:

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

lanza el programa HelloWorld, habilita verbose output desde JIT y hace que JIT genere el código nativo sin realizar ninguna optimización.

Siga estos pasos para determinar qué parte del compilador está causando la anomalía:

Procedimiento

1. Establezca el parámetro de JIT **count=0** para cambiar el umbral de compilación a cero. Este parámetro hará que cada método Java se compile antes de ejecutarse. Utilice **count=0** únicamente cuando diagnostique problemas porque se compilen muchos métodos, incluidos métodos que se utilizan con poca frecuencia. La compilación adicional utiliza más recursos informáticos para la compilación y ralentiza la aplicación. Con **count=0**, la aplicación falla inmediatamente cuando se alcance el área del problema. En algunos casos, el uso de **count=1** puede reproducir la anomalía de forma más fiable.
2. Añada **disableInlining** a los parámetros del compilador JIT. **disableInlining** inhabilita la generación de código más grande y más complejo. Si el problema ya no se produce, utilice **disableInlining** como método alternativo mientras el equipo de servicio de Java analiza y soluciona el problema del compilador.
3. Disminuya los niveles de optimización añadiendo el parámetro **optLevel** y vuelva a ejecutar el programa hasta que ya no se produzca la anomalía o se alcance el nivel “noOpt”. En caso de problema con el compilador JIT, empiece con “scorching” y vaya bajando por la lista. Los niveles de optimización son, en orden decreciente:
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold

f. noOpt

Qué hacer a continuación

Si uno de estos valores hace que desaparezca la anomalía, tendrá un método alternativo que podrá utilizar. Este método alternativo es temporal mientras el equipo de servicio de Java analiza y soluciona el problema del compilador. Si eliminar **disableInlining** de la lista de parámetros de JIT hace que no reaparezca la anomalía, hágalo así para mejorar el rendimiento. Siga las instrucciones de "Ubicación del método erróneo" para mejorar el rendimiento del método alternativo.

Si sigue produciéndose la anomalía en el nivel de optimización "noOpt", deberá inhabilitar el compilador JIT como método alternativo.

Ubicación del método erróneo:

Cuando haya determinado el nivel de optimización más bajo al que el compilador JIT o AOT debe compilar métodos para desencadenar la anomalía, podrá descubrir qué parte del programa Java, cuando está compilado, causa la anomalía. A continuación, puede indicar al compilador que limite el método alternativo a un método, una clase o un paquete específicos, permitiendo al compilador compilar el resto del programa como lo hace normalmente. En caso de anomalías del compilador JIT, si se produce la anomalía con **-Xjit:optLevel=noOpt**, también puede indicar al compilador que no compile el método o los métodos que causan la anomalía.

Antes de empezar

Si ve una salida de error como la de este ejemplo, puede utilizarla para identificar el método erróneo:

```
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;
```

Las líneas importantes son:

vmState=0x00000000
Indica que el código que ha fallado no era código de tiempo de ejecución de la JVM.

Module= o Module_base_address=
No en la salida (podría estar vacía o ser cero) porque el código fue compilado por el JIT, y fuera de cualquier DLL o biblioteca.

Compiled_method=
Indica el método Java para el que se produjo el código compilado.

Acerca de esta tarea

Si la salida no indica el método erróneo, siga estos pasos para identificar el método erróneo:

Procedimiento

1. Ejecute el programa Java con los parámetros de JIT **verbose** y **vlog=**<*nombre_archivo*> añadido a la opción **-Xjit**o **-Xaot**. Con estos parámetros, el compilador lista métodos compilados en un archivo de registro llamado <*nombre_archivo*>.<*fecha*>.<*hora*>.<*PID*>, también denominado *archivo de límite*. Un archivo de límite normal contiene líneas que corresponden a los métodos compilados, como:

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

   Las líneas que no empiezan por el signo más son ignoradas por el compilador en los siguientes pasos y puede eliminarlas del archivo. Los métodos para los que se carga el código AOT desde la memoria caché de clase compartida empiezan por `+ (AOT load)`.
2. Ejecute de nuevo el programa con el parámetro de JIT o AOT **limitFile=**(<*nombre_archivo*>,<*m*>,<*n*>), donde <*nombre_archivo*> es la vía de acceso al archivo de límite, y <*m*> y <*n*> son los números de línea que indican el primer y el último método del archivo de límite que deben compilarse. El compilador solo compila los métodos mostrados en las líneas de <*m*> a <*n*> en el archivo de límite. Los métodos no mostrados en el archivo de límite y los métodos mostrados en líneas fuera del rango no se compilan; no se cargará ningún código AOT de la memoria caché de datos compartidos para dichos métodos. Si el programa ya no falla, uno o varios de los métodos que ha eliminado en la última iteración habrán sido la causa de la anomalía.
3. Repita este proceso utilizando valores diferentes para <*m*> y <*n*>, tantas veces como sea necesario, para encontrar el conjunto mínimo de métodos que deben compilarse para desencadenar la anomalía. Dividiendo por dos el número de líneas seleccionadas cada vez, puede realizar una búsqueda binaria del método erróneo. A menudo, puede reducir el archivo a una sola línea.

Qué hacer a continuación

Cuando haya ubicado el método erróneo, podrá inhabilitar el compilador JIT o AOT únicamente para el método erróneo. Por ejemplo, si el método java/lang/Math.max(II)I hace que falle el programa cuando se realiza la compilación JIT con **optLevel=hot**, podrá ejecutar el programa con:

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

para compilar únicamente el método erróneo a un nivel de optimización de "warm", aunque compilando los otros métodos como de costumbre.

Si un método falla cuando se compila mediante JIT en "noOpt", puede excluirlo totalmente de la compilación, utilizando el parámetro **exclude**={<*método*>}:

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

Si un método hace que falle el programa cuando el código AOT se se carga desde la memoria caché de datos compartidos, excluya el método de la la carga AOT utilizando el parámetro **exclude**={<*método*>}:

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

Los métodos AOT solo se compilan en la memoria caché de clase compartida durante el paso de relleno de **admincache**. Evitar la carga AOT es el mejor enfoque de diagnóstico para problemas con estos métodos.

Identificación de anomalías de compilación JIT y AOT:

Para anomalías del compilador JIT, analice la salida de errores para determinar si se ha producido una anomalía cuando el compilador JIT intentaba compilar un método.

Si la JVM se bloquea y puede ver que la anomalía se ha producido en la biblioteca JIT (`libj9jit26.so`), puede que el compilador JIT haya fallado durante un intento de compilación de un método.

Si ve una salida de error como la de este ejemplo, puede utilizarla para identificar el método erróneo:

```
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;)
```

Las líneas importantes son:

vmState=0x00050000
: Indica que el compilador JIT está compilando código. Para ver una lista de los números de código vmState, consulte la tabla de códigos del volcado Java en la Guía de usuario de IBM SDK para Java V7, http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html.

Module=/home/test/sdk/jre/bin/libj9jit26.so
: Indica que el error se ha producido en `libj9jit26.so`, el módulo del compilador JIT.

Method_being_compiled=
: Indica que el método de Java se está compilando.

Si la salida no indica el método erróneo, utilice la opción **verbose** con los siguientes valores adicionales:

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

Estos valores de **verbose** informan cuándo el compilador JIT o AOT empieza a compilar un método, y cuándo lo finaliza. Si JIT o el compilador AOT falla en un método concreto (esto es, inicia la compilación pero falla antes de poder finalizar), utilice el parámetro **exclude** en la opción de línea de mandatos **-Xjit** o **-Xaot** para excluirla de la compilación JIT o AOT (consulte "Ubicación del método erróneo" en la página 143). En caso de problemas con la compilación AOT, destruya su memoria caché de clase compartida antes de utilizar la opción **exclude**. Si la

exclusión del método impide el bloqueo, tendrá un método alternativo que podrá utilizar mientras el equipo de servicio corrige el problema.

Identificación de anomalías de compilación de AOT en modalidad no de tiempo real:

La determinación de problemas de AOT en modalidad no de tiempo real es muy parecida a la determinación de problemas de JIT.

Acerca de esta tarea

Como ocurre con JIT, ejecute primero la aplicación con **-Xnoaot**, que garantiza que el código de AOT no se utilice al ejecutar la aplicación.

Si esto soluciona el problema, reconstruya los archivos jar de AOT utilizando la misma técnica que se describe en "Ubicación del método erróneo" en la página 143, proporcionando la opción **-Xaot** a la hora del build de AOT, en lugar del tiempo de ejecución de aplicación.

Identificación de anomalías de compilación de AOT en modalidad de tiempo real:

La determinación de problemas de AOT utiliza la herramienta admincache para localizar el problema.

Acerca de esta tarea

En contraste con las anomalías de compilación de JIT, que se producen durante el tiempo de ejecución de la aplicación, las anomalías de compilación de AOT se producen durante el paso de llenado de admincache.

Para averiguar donde se produce el problema, ejecute la herramienta admincache con la opción **-Xnoaot**. Esto garantiza que la aplicación no se ejecute con código compilado de manera anticipada.

Si utilizar la opción **-Xnoaot** soluciona el problema, examine la salida del bloqueo original. La salida proporciona información que identifica qué método es la causa del problema. Busque una línea parecida a esta:

```
Method_being_compiled=myAppClass.main(Ljava/lang/String;)V
```

Para evitar el problema, este método se debe excluir de la compilación anticipada. Para hacerlo, añada una opción a la línea de mandatos admincache, similar al ejemplo siguiente:

```
-Xaot:exclude={myAppClass.main(Ljava/lang/String;)V}
```

Esta exclusión evita la compilación AOT del método del problema.

Rendimiento de aplicaciones de ejecución breve

El compilador JIT de IBM está ajustado para las aplicaciones de larga ejecución que normalmente se utilizan en un servidor. Puede usar la opción de línea de mandatos **-Xquickstart** en modalidad no de tiempo real para mejorar el rendimiento de las aplicaciones de ejecución breve, en especial, de las aplicaciones en las que el proceso no está concentrado en pocos métodos.

-Xquickstart hace que el compilador JIT utilice un nivel de optimización inferior de forma predeterminada y que compile menos métodos. Realizar menos

compilaciones más rápidamente puede mejorar el tiempo de inicio de las aplicaciones. Cuando el compilador AOT está activo (tanto las clases compartidas como la compilación AOT habilitadas), **-Xquickstart** hará que todos los métodos seleccionados para compilación sean compilados por AOT, lo que mejora el tiempo de inicio de las ejecuciones posteriores. **-Xquickstart** puede degradar el rendimiento si se utiliza con aplicaciones que tardan en ejecutarse y que contengan métodos que usan una gran cantidad de recursos de proceso. La implementación de **-Xquickstart** está sujeta a cambios en futuros releases.

También puede intentar mejorar los tiempos de inicio ajustando el umbral de JIT (utilizando prueba y error). Para obtener más información, consulte "Inhabilitación selectiva del compilador JIT" en la página 142.

-Xquickstart no tiene efecto sobre el uso del código AOT con **-Xrealtime**.

Comportamiento de la JVM durante períodos desocupados

Puede reducir los ciclos de la CPU consumidos por una JVM desocupada utilizando la opción **-XsamplingExpirationTime** para desactivar la hebra de muestreo de JIT.

La hebra de muestreo de JIT perfila la aplicación Java en ejecución para descubrir métodos utilizados comúnmente. El uso de la memoria y el procesador de la hebra de muestreo es insignificante y la frecuencia de definición de perfiles se reduce automáticamente cuando la JVM está desocupada.

En algunas circunstancias, puede que no desee ciclos de la CPU consumidos por una JVM desocupada. Para ello, especifique la opción **-XsamplingExpirationTime**<*tiempo*>. Establezca la <*tiempo*> en el número de segundos durante los cuales desea que se ejecute la hebra de muestreo. Utilice esta opción con precaución; una vez desactivada, no podrá reactivar la hebra de muestreo. Permita que la hebra de muestreo se ejecute durante el suficiente tiempo para identificar optimizaciones importantes.

El recopilador de diagnósticos

El recopilador de diagnósticos recopila los archivos de diagnóstico Java para un suceso de un problema.

La recopilación de los archivos necesarios para el servicio de IBM puede acelerar el el proceso de resolución de problemas. La Guía de usuario de IBM SDK para Java V7 contiene información detallada sobre el uso del Recopilador de diagnósticos.

Aquí puede encontrar esta información: IBM SDK para Java 7 - El Recopilador de diagnósticos.

Datos de diagnóstico del recopilador de basura

Este apartado describe cómo diagnosticar problemas en la recopilación de basura.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre el diagnóstico de problemas con la recopilación de basura, y cubre:

- registro cronológico detallado de la recopilación de basura
- rastreo de la recopilación de basura usando **-Xtgc**

Puede encontrar la información aquí: IBM SDK para Java 7 - Datos de diagnóstico del recopilador de basura.

En las secciones siguientes se proporciona información adicional sobre la Recogida de basura metronome de IBM WebSphere Real Time for RT Linux.

Resolución de problemas del recopilador de basura Metronome

Con las opciones de línea de mandatos, puede controlar la frecuencia de la recogida de basura Metronome, las excepciones de falta de memoria y el comportamiento de Metronome en llamadas explícitas al sistema.

Utilización de la información de verbose:gc:

Puede utilizar la opción **-verbose:gc** con la opción **-Xgc:verboseGCCycleTime=N** para escribir información en la consola sobre la actividad del sistema de recogida de basura de Metronome. No todas las propiedades XML de la salida **-verbose:gc** de la JVM estándar se crean o se aplican a la salida de Recopilador de basura Metronome.

Utilice la opción **-verbose:gc** para ver el espacio libre mínimo, máximo y medio del almacenamiento dinámico. De este modo, podrá comprobar el nivel de actividad y uso del almacenamiento dinámico y luego ajustar los valores, si fuese necesario. La opción **-verbose:gc** escribe las estadísticas de Metronome en la consola.

La opción **-Xgc:verboseGCCycleTime=N** controla la frecuencia de recuperación de la información. Determina el tiempo en milisegundos en el que se vuelcan los resúmenes. El valor predeterminado para N es 1000 milisegundos. El tiempo de ciclo no implica que el resumen se vuelque exactamente a dicha hora, sino cuando pasa el último suceso de recogida de basura que satisface este criterio temporal. La recogida y visualización de estas estadísticas pueden aumentar los objetivos de tiempo de pausa del recopilador de basura Metronome y, a medida que N se reduce, los tiempos de las pausas pueden ser grandes.

Una cantidad es un único periodo de actividad de Recopilador de basura Metronome, que provoca una interrupción o una pausa en una aplicación.

Ejemplo de salida de verbose:gc

Especifique:

```
java -Xrealtime -verbose:gc -Xgc:verboseGCCycleTime=N mi_aplicación
```

En este ejemplo se muestra la salida inicial de verbose:gc, que contiene la versión y los valores de la recogida de basura:

```
<verbosegc
xmlns="http://www.ibm.com/j9/verbosegc" version="R26_Java726_GA_20110716_0946_B87065">

<initialized id="1" timestamp="2011-07-27T14:17:52.277">
  <attribute name="gcPolicy" value="-Xgcpolicy:metronome" />
  <attribute name="maxHeapSize" value="0x5800000"/>
  <attribute name="initialHeapSize" value="0x4000000"/>
  <attribute name="compressedRefs" value="false" />
  <attribute name="pageSize" value="0x1000" />
  <attribute name="requestedPageSize" value="0x1000" />
  <attribute name="gcthreads" value="1" />
  <region>
    <attribute name="regionSize" value="16384"/>
    <attribute name="regionCount" value="4096"/>
    <attribute name="arrayletLeafSize" value="2048"/>
  </region>
  <metronome>
    <attribute name="beatsPerMeasure" value="500" />
```

```
    <attribute name="timeInterval" value="10000" />
    <attribute name="targetUtilization" value="70"/>
    <attribute name="trigger" value="0x2000000"/>
    <attribute name="headRoom" value="0x100000" />
  </metronome>
  <system>
    <attribute name="physicalMemory" value="12507463680"/>
    <attribute name="numCPUs" value="8"/>
    <attribute name="architecture" value="x86" />
    <attribute name="os" value="Linux"/>
    <attribute name="osVersion" value="2.6.24.7-75ibmrt2.18"/>
  </system>
  <vmargs>
    <vmarg
name="-Xoptionsfile=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime/options.default"/>
    <vmarg name="-Xjcl:jclse7b_26"/>
    <vmarg
name="-Dcom.ibm.oti.vm.bootstrap.library.path=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime:/
my_dir/pxi3270hrt-2011071..."/>
    <vmarg
name="-Dsun.boot.library.path=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime:/my_dir/
pxi3270hrt-20110719_02/sdk/jre/lib..."/>
    <vmarg
name="-Djava.library.path=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime:/my_dir/
pxi3270hrt-20110719_02/sdk/jre/lib/i38..."/>
    <vmarg name="-Djava.home=/my_dir/pxi3270hrt-20110719_02/sdk/jre"/>
    <vmarg name="-Djava.ext.dirs=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/ext"/>
    <vmarg name="-Duser.dir=/my_dir/pxi3270hrt-20110719_02/sdk/jre/bin"/>
    <vmarg name="_j2se_j9=1120000"
value="F76FF700"/>
    <vmarg name="-Djava.runtime.version=pxi3270hrt-20110719_02"/>
    <vmarg name="-Djava.class.path=."/>
    <vmarg name="-Xrealtime"/>
    <vmarg name="-verbose:gc" />
    <vmarg name="-Dsun.java.launcher=SUN_STANDARD" />
    <vmarg name="-Dsun.java.launcher.pid=5543"/>
    <vmarg name="_port_library" value="F7701B80"/>
    <vmarg name="_bfu_java" value="F77029A8"/>
    <vmarg name="_org.apache.harmony.vmi.portlib" value="08051DA0"/>
  </vmargs>
</initialized>
```

Cuando se desencadena la recogida de basura, se produce un suceso `trigger start`, seguido de varios sucesos `heartbeat` y, por último, un suceso `trigger end` cuando se satisface el desencadenante. En este ejemplo se muestra el ciclo de recogida de basura desencadenado como salida verbose:gc:

```
<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" />
  <free-mem type="immortal" minBytes="44234538" meanBytes="60342344" maxBytes="61219900"/>
  <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" />
  <free-mem type="immortal" minBytes="38234500" meanBytes="41736440" maxBytes="42233458"/>
```

```
  <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" />
  <free-mem-delta type="immortal" bytesBefore="35735400" bytesAfter="35735400"/>
</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" />
```

Se pueden producir los siguientes tipos de evento:

<trigger-start ...>
El inicio de un ciclo de recogida de basura, cuando la cantidad de memoria usada llega a ser mayor que el umbral del desencadenante. El umbral predeterminado es el 50% del almacenamiento dinámico. El atributo `intervalms` es el intervalo entre el suceso anterior `trigger end` (con id-1) y este suceso `trigger start`.

<trigger-end ...>
Un ciclo de recogida de basura ha reducido correctamente la cantidad de memoria utilizada por debajo del umbral desencadenante. Si un ciclo de recogida de basura ha terminado, pero la memoria utilizada no ha caído por debajo del umbral desencadenante, se iniciará un nuevo ciclo de recogida de basura con el mismo ID de contexto. Para cada suceso de tipo `trigger start`, existe un suceso `trigger end` coincidente con el mismo ID de contexto. El atributo `intervalms` es el intervalo entre el suceso `trigger start` anterior y el suceso `trigger end` actual. Durante este tiempo, se habrán completado una o varias recogidas de basura hasta que la memoria utilizada esté por debajo del umbral desencadenante.

<gc-op id="28" type="heartbeat"...>
Evento periódico que recopila información (sobre la memoria y el tiempo) acerca de todas las cantidades de la recogida de basura del tiempo que cubre. Un suceso de pulsación solo se puede producir entre un par coincidente de sucesos `trigger start` y `trigger end`; es decir, mientras un ciclo activo de recogida de basura está en curso. El atributo `intervalms` es el intervalo entre el suceso de pulsación anterior (con `id` -1) y este suceso de pulsación.

<gc-op id="29" type="syncgc"...>
Evento de recogida de basura síncrono (no determinante). Consulte el apartado “Recogidas de basura síncronas” en la página 151

Las etiquetas XML de este ejemplo tienen los significados siguientes:

<quanta ...>
Resumen de las pausas de cantidad durante el intervalo de pulsaciones, incluyendo la longitud de las pausas, en milisegundos.

<free-mem type="heap" ...>
Resumen de la cantidad de espacio libre de almacenamiento dinámico durante el intervalo de pulsaciones, con la muestra tomada al final de cada cantidad de recogida de basura.

<classunload-info classloadersunloaded=9 classesunloaded=156 />
Número de cargadores de clase y clases descargadas durante el intervalo de pulsaciones.

<references type="weak" cleared="660 />
Número y tipo de objetos de referencia de Java borrados durante el intervalo de pulsaciones.

Nota:

- Si solo se ha producido una cantidad de recogida de basura en el intervalo entre dos pulsaciones, se toma una muestra de la memoria libre solo al final de esta cantidad. Por lo que las cantidades mínima, máxima y media que se ofrecen en el resumen de la pulsación son todas iguales.
- El intervalo entre dos sucesos de pulsación podría ser significativamente mayor que el tiempo de ciclo especificado sei el almacenamiento dinámico no está suficientemente completo como para precisar de la actividad de recogida de basura. Por ejemplo, si su programa necesita una actividad de recogida de basura solo una vez cada pocos segundos, es probable que vea una pulsación solo una vez cada pocos segundos.
- Es posible que el intervalo sea significativamente mayor que el tiempo del ciclo especificado porque la recogida de basura no tiene trabajo en un almacenamiento dinámico que no está lo suficientemente lleno como para garantizar la actividad de la recogida de basura. Por ejemplo, si su programa necesita una actividad de recogida de basura solo una vez cada pocos segundos, es probable que vea una pulsación solo una vez cada pocos segundos.

  Si se produce un suceso, como la recogida de basura síncrona o un cambio en la prioridad, los detalles del suceso y todos los sucesos pendientes, como pulsaciones, se generan como salida de inmediato.
- Si la cantidad máxima de la recogida de basura para un periodo determinado es demasiado grande, es posible que desee reducir la utilización objetivo utilizando la opción **-Xgc:targetUtilization**. Esta acción proporciona a la recogida de basura más tiemp para trabajar. Como alternativa, es posible que quiera aumentar el tamaño de almacenamiento dinámico con la opción **-Xmx**. Del mismo modo, si su aplicación puede tolerar retardos más largos de los que se están notificando actualmente, puede aumentar la utilización objetivo o reducir el tamaño de almacenamiento dinámico.
- La salida se puede redirigir a un archivo de registro en lugar de a la consola con la opción **-Xverbosegclog:<archivo>**; por ejemplo, **-Xverbosegclog:out** escribe la salida **-verbose:gc** en el archivo *out*.
- La prioridad de la lista de `thread-priority` es la prioridad de la hebra del sistema operativo subyacente, no una prioridad de hebra Java.

Recogidas de basura síncronas

También se graba una entrada en el registro **-verbose:gc** cuando se produce una recogida de basura síncrona (no determinante). Este suceso tiene tres causas posibles:

- Una llamada `System.gc()` explícita en el código.
- La JVM se queda sin memoria y ejecuta una recogida de basura síncrona para evitar una condición de tipo OutOfMemoryError.
- La JVM se apaga durante una recogida de basura continua. La JVM no puede cancelar la recogida, por lo que completa la recogida de forma síncrona y luego sale.

Este es un ejemplo de una entrada `System.gc()`:

```
<gc-op id="9" type="syncgc" timems="12.92" contextid="8" timestamp="2011-07-12T09:41:40.808">
  <syncgc-info reason="system GC" totalBytesRequested="260" exclusiveaccessTimeMs="0.009"
    threadPriority="11" />
```

```
  <free-mem-delta type="heap" bytesBefore="22085440" bytesAfter="136023450" />
  <free-mem-delta type="immortal" bytesBefore="62324800" bytesAfter="62324800"/>
  <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>
```

Este es un ejemplo de una entrada de recogida de basura síncrona como resultado del cierre de la JVM:

```
<gc-op id="24" type="syncgc" timems="6.439" contextid="19" timestamp="2011-07-12T09:43:14.524">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="56182430" bytesAfter="151356238" />
  <free-mem-delta type="immortal" bytesBefore="23659200" bytesAfter="23659200"/>
  <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>
```

Los atributos y etiquetas XML de este ejemplo tienen los significados siguientes:

<gc-op id="9" type="syncgc" timems="6.439" ...
: La línea indica que el tipo de evento es una recogida de basura síncrona. El atributo `timems` es la duración de la recogida de basura síncrona, en milisegundos.

<syncgc-info reason="..."/>
: El motivo de la recogida de basura síncrona.

<free-mem-delta.../>
: Memoria de almacenamiento dinámico de Java libre antes y después de la recogida de basura síncrona en bytes.

<finalization .../>
: Número de objetos pendientes de finalización.

<classunload-info .../>
: Número de cargadores de clase y clases descargadas durante el intervalo de pulsaciones.

<references type="weak" cleared="53" .../>
: Número y tipo de objetos de referencia de Java borrados durante el intervalo de pulsaciones.

La recogida de basura síncrona debida a condiciones de falta de memoria o al cierre de la máquina virtual solo se puede producir si el recopilador de basura está activo. Tiene que ir precedida por un suceso `trigger start`, aunque no es necesario que sea de inmediato. Algunos sucesos de pulsación pueden ocurrir entre un suceso `trigger start` y el suceso `synchgc`. La recogida de basura síncrona causada por `System.gc()` se puede producir en cualquier momento.

Rastreo de todas las cantidades del recopilador de basura

Las cantidades individuales del recopilador de basura se pueden rastrear si se habilitan los puntos de rastreo `GlobalGCStart` y `GlobalGCEnd`. Estos puntos de rastreo se producen al principio y al final de toda la actividad del recopilador de basura Metronome, incluidas las recogidas de basura síncronas. La salida de estos puntos de rastreo tendrá un aspecto similar al siguiente:

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

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

Cambios en la prioridad

Además de los resúmenes, se escribe una entrada en el registro **-verbose:gc** cuando la cambia la prioridad de hebras del recopilador de basura (porque la aplicación ha cambiado prioridades de hebras o porque una o más hebras de una aplicación han finalizado). La prioridad de la lista es la prioridad de la hebra del sistema operativo subyacente, no una prioridad de hebra Java. Este es un ejemplo de una entrada de cambio de prioridades de hebras del recopilador de basura:

```
<gc type="heartbeat" id="73" timestamp="Feb 26 13:11:35 2007" intervalms"1001.754">
  <summary quantumcount="240">
    <quantum minms="0.022" meanms="0.984" maxms="1.011" />
    <classunloading classloaders="11" classes="17" />
    <heap minfree="202833920" meanfree="214184823" maxfree="221102080" />
    <thread-priority maxPriority="11" minPriority="11" />
  </summary>
</gc>
```

Los cambios de prioridades se pueden rastrear en Real Time generando la información del punto de rastreo relacionado con las prioridades de hebras del recopilador de basura. Esta salida tiene un aspecto parecido a este:

```
15:58:25.493*0x8286e00    j9mm.102       - setGCThreadPriority() called with
newGCThreadPriority = 11
```

Esta salida se puede habilitar utilizando el ID, como se indica a continuación: **-Xtrace:iprint=tpnid{j9mm.102}**

Entradas de falta de memoria

Cuando falta espacio libre en una de las áreas de memoria, se graba una entrada en el registro **-verbose:gc** antes de que se genere la excepción OutOfMemoryError. Este es un ejemplo de esta salida:

```
<out-of-memory id="71" timestamp="2011-07-23T08:32:51.435" memorySpaceName="Scoped"
  memorySpaceAddress="080EED9C"/>
```

De forma predeterminada, el volcado Java se produce como resultado de una excepción OutOfMemoryError. Este volcado contiene información sobre las áreas de memoria que utiliza su programa. Junto con el valor `J9MemorySpace` dado en la salida **-verbose:gc**, puede utilizar esta información del volcado para identificar el área de memoria concreta que se ha quedado sin espacio:

```
NULL           id         start      end        size       space/region
1STHEAPSPACE   0x080EED9C     --         --         --     Scoped
1STHEAPREGION  0x0810C570 0xF1B09028 0xF2B09028 0x01000000 Scoped/Region
NULL
1STHEAPTOTAL   Total memory:           16777216 (0x01000000)
1STHEAPINUSE   Total memory in use:      625952 (0x00098D20)
1STHEAPFREE    Total memory free:      16151264 (0x00F672E0)
```

En el ejemplo anterior, el ID de espacio de memoria proporcionado en la salida de -verbose:gc (0x080EED9C) se puede hacer coincidir con el ID del área de memoria con ámbito del volcado de Java. Esta coincidencia puede ser útil si tiene varios ámbitos y necesita identificar cuál se ha quedado sin memoria, ya que la salida de -verbose:gc sólo indica si el error OutOfMemoryError se ha producido en memoria inmortal, con ámbito o de almacenamiento dinámico.

Comportamiento del recopilador de basura Metronome en condiciones de falta de memoria:

De forma predeterminada, el recopilador de basura Metronome desencadena una recogida de basura ilimitada y no determinante cuando la JVM se queda sin memoria. Para evitar un comportamiento no determinante, utilice la opción **-Xgc:noSynchronousGCOnOOM** para generar una excepción OutOfMemoryError cuando la JVM se quede sin memoria.

La recogida ilimitada predeterminada se ejecuta hasta que se recoge toda la basura posible en una única operación. El tiempo de pausa necesario suele ser mucho mayor que en una cantidad incremental normal de Metronome.

Información relacionada:

Utilización de -Xverbose:gc para analizar las recogidas de basura síncronas

Comportamiento del recopilador de basura Metronome en llamadas System.gc() explícitas:

Si hay un ciclo de recogida de basura en curso, el recopilador de basura Metronome completa el ciclo de manera síncrona cuando se llama a `System.gc()`. Si no hay ningún ciclo de recogida de basura en curso, se realiza un ciclo síncrono completo cuando se llama a `System.gc()`. Utilice `System.gc()` para limpiar el almacenamiento dinámico de forma controlada. Se trata de una operación no determinante porque realiza una recogida de basura completa antes del retorno.

Algunas aplicaciones llaman al software del proveedor que tiene llamadas System.gc() cuando no es aceptable crear estos retardos no determinantes. Para inhabilitar todas las llamadas System.gc(), utilice la opción **-Xdisableexplicitgc**.

La salida detallada de la recogida de basura para una llamada System.gc() tiene una razón de "recogida de basura del sistema" y es probable que su duración sea larga:

```
<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"/>
  <free-mem-delta type="immortal" bytesBefore="5129096" bytesAfter="5129096"/>
  <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>
```

Datos de diagnóstico de clases compartidas

Entender cómo diagnosticar problemas que puedan producirse le ayudará a utilizar la modalidad de clases compartidas.

Para ver una introducción a las clases compartidas, consulte Compartimiento de datos de clase entre JVM.

La Guía de usuario de IBM SDK para Java V7 contiene instrucciones útiles sobre el diagnóstico de problema con clases compartidas, y cubre:

- Despliegue de clases compartidas
- Gestión de la modificación del código de bytes en tiempo de ejecución
- Información sobre actualizaciones dinámicas
- Utilización de la API del ayudante Java

- Descripción de la salida de diagnóstico de clases compartidas
- Depuración de problemas con clases compartidas

Puede encontrar la información aquí: IBM SDK para Java 7 - Datos de diagnóstico de clases compartidas .

Parte del material en la Guía del usuario de IBM SDK para Java V7 podría no ser de aplicación a IBM WebSphere Real Time for RT Linux. Concretamente:

- En la modalidad de tiempo real, las aplicaciones sólo tienen acceso de lectura a memorias caché de clases compartidas, no acceso lectura/escritura.
- Las memorias caché se pueden modificar exclusivamente utilizando la herramienta **admincache**.
- Las memorias caché no persistentes no están disponibles en modalidad de tiempo real.

Uso de JVMTI

JVMTI es una interfaz bidireccional que permite la comunicación entre la JVM y un agente nativo. Sustituye a las interfaces JVMDI y JVMPI.

JVMTI permite a terceros desarrollar herramientas de depuración, definición de perfiles y supervisión para la JVM. La interfaz contiene mecanismos para que el agente notifique a la JVM acerca del tipo de información que requiere. La interfaz también proporciona un medio para recibir las notificaciones importantes. Se pueden conectar varios agentes a una JVM en cualquier momento.

La Guía de usuario de IBM SDK para Java V7 contiene información detallada sobre el uso de JVMTI, incluyendo una sección de referencia de API en ampliaciones de IBM para JVMTI.

Puede encontrar dicha información aquí: IBM SDK for Java 7 - Using JVMTI.

Uso de la Diagnostic Tool Framework for Java

La Diagnostic Tool Framework for Java (DTFJ) es una interfaz de programación de aplicación (API) Java de IBM que se utiliza para dar soporte a la creación de herramientas de diagnósticos de Java. DTFJ funciona con datos de un volcado del sistema o un volcado Java.

La Guía del usuario de IBM SDK para Java V7 contiene información detallada sobre DTFJ. Siga este enlace:Uso de Diagnostic Tool Framework para Java

Capítulo 10. Referencia

Este conjunto de temas muestra las opciones y las bibliotecas de clases que se pueden utilizar con WebSphere Real Time for RT Linux

Opciones de línea de mandatos

Puede especificar opciones en la línea de mandatos mientras inicia Java. Las opciones por omisión se han seleccionado para el mejor uso general.

Especificación de opciones Java y propiedades del sistema

Existen tres maneras de especificar las propiedades Java y las propiedades del sistema.

Acerca de esta tarea

Puede especificar las opciones Java y las propiedades del sistema de las siguientes formas. En orden de prioridad, éstos modos son:

1. Especificando la opción o propiedad en la línea de mandatos. Por ejemplo:

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

2. Al crear un archivo que contenga las opciones y especificar dicho archivo en la línea de mandatos con la opción **-Xoptionsfile=**<*nombre_archivo*>.

   En el archivo de opciones, especifique cada opción en una nueva línea; puede utilizar el carácter '\' como carácter de continuación si desea que una única opción abarque varias líneas. Utilice el carácter "#" para definir líneas de comentarios. No puede especificar **-classpath** en un archivo de opciones. He aquí un ejemplo de archivo de opciones:

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

3. Creando una variable de entorno denominada **IBM_JAVA_OPTIONS** que contenga las opciones. Por ejemplo:

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

La última opción que especifique en la línea de mandatos tiene preferencia sobre la primera. Por ejemplo, si especifica las opciones **-Xint -Xjit myClass**, la opción **-Xjit** prevalece sobre **-Xint**.

Propiedades del sistema

Las propiedades del sistema están a disposición de las aplicaciones y ayudan a proporcionar información sobre el entorno de ejecución.

com.ibm.jvm.realtime
: Esta propiedad habilita a las aplicaciones de Java para determinar si se están ejecutando en un entorno de WebSphere Real Time for RT Linux.

  Si su aplicación se está ejecutando en el entorno de ejecución de IBM WebSphere Real Time for RT Linux y se ha iniciado con la opción **-Xrealtime**, la propiedad **com.ibm.jvm.realtime** tendrá el valor “hard”.

Si su aplicación se ejecuta en el entorno de ejecución de IBM WebSphere Real Time for RT Linux, pero no se ha iniciado con la opción **-Xrealtime**, no se define la propiedad **com.ibm.jvm.realtime**.

Si su aplicación se está ejecutando en el entorno de ejecución de IBM WebSphere Real Time, la propiedad **com.ibm.jvm.realtime** tendrá el valor "soft".

Opciones estándar

Las definiciones de las opciones estándar.

-agentlib:<*nombre biblioteca*>**[=**<*opciones*>**]**
Carga la biblioteca nativa del agente <*nombre biblioteca*>; por ejemplo **-agentlib:hprof**. Para obtener más información, especifique **-agentlib:jdwp=help** y **-agentlib:hprof=help** en la línea de mandatos.

-agentpath:*nombre de biblioteca***[=**<*opciones*>**]**
Carga la biblioteca nativa del agente mediante el nombre de vía de acceso completo.

-assert Imprime ayuda sobre opciones relacionadas con aserciones.

-cp or -classpath <*directorios y archivos .zip or .jar separados por* **:**>
Establece la vía de acceso de búsqueda de clases y recursos de aplicaciones. Si no se utilizan **-classpath** y **-cp** y no está establecida **CLASSPATH**, la classpath del usuario es, de forma predeterminada, el directorio activo (.).

-D<nombre_propiedad>=<*valor*>
Establece una propiedad de sistema.

-help o -?
Imprime un mensaje de uso.

-javaagent:<*vía de acceso de jar*>**[=**<*opciones*>**]**
Carga un agente de lenguaje de programación Java. Para obtener más información, consulte la documentación sobre la API java.lang.instrument.

-jre-restrict-search
Incluye los JRE privados en la búsqueda de versión.

-no-jre-restrict-search
Excluye los JRE privados en la búsqueda de versión.

-showversion
Imprime la versión del producto y prosigue.

-verbose:[*clase***,***gc***,***dynload***,***tamaños***,***pila***,***jni***]**
Habilita la salida verbosa.

-verbose:clase
Graba una entrada en stderr para cada clase cargada.

-verbose:gc
Consulte "Utilización de la información de verbose:gc" en la página 148.

-verbose:dynload
Proporciona información detallada a medida que la JVM carga cada clase, entre ella:
- El nombre de clase y paquete
- Para archivos de clase que estén en un archivo `.jar`, el nombre y la vía de acceso del directorio del `.jar`

- Detalles del tamaño de la clase y el tiempo que tarda en cargarse la clase

Los datos se escriben fuera de stderr. Un ejemplo de la salida es el siguiente:

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

Nota: Las clases cargadas desde la memoria caché de clase compartida no aparecen en la salida de **-verbose:dynload**. Utilice **-verbose:class** para obtener información sobre estas clases.

-verbose:tamaños
Escribe información en stderr que describe la cantidad de memoria utilizada para pilas y volcados en la JVM

-verbose:pila
Escribe información en stderr que describe el uso de la pila C y Java.

-verbose:jni
Graba información en stderr que describe los servicios JNI que invocan la aplicación y la JVM.

-version
Imprime la información de la versión para la modalidad que no es de tiempo real. Cuando se utiliza con la opción -Xrealtime, imprime la información de la versión para la modalidad en tiempo real.

-version:*<valor>*
Requiere que se ejecute la versión especificada.

-X Imprime ayuda sobre las opciones no estándar.

Opciones no estándar

Las opciones con el prefijo **-X** no son estándar, y están sujetas a cambios sin aviso.

La Guía del usuario de IBM SDK para Java V7 contiene información detallada sobre opciones que no son estándar. Puede encontrar la información aquí: IBM SDK para Java 7 - Opciones de la línea de mandatos.

Los cambios de la guía del usuario de IBM SDK para Java V7 se describen en la siguiente lista:

- **-XX:-LazySymbolResolution** es la opción predeterminada en el sistema operativo RT Linux, y no **-XX:+LazySymbolResolution**.

En las secciones siguientes se proporciona información adicional para IBM WebSphere Real Time for RT Linux.

Opciones de tiempo real

La definición de la opción **-Xrealtime** utilizada en WebSphere Real Time for RT Linux.

Las opciones **-X** siguientes se aplican en el entorno WebSphere Real Time for RT Linux.

-Xrealtime
Inicia la modalidad en tiempo real. Es necesaria si desea ejecutar

Recopilador de basura Metronome y utilizar los servicios de Especificación de tiempo real para Java (RTSJ). Si no especifica esta opción, la JVM se inicia en modalidad no de tiempo real equivalente a SDK de IBM y Runtime Environment para plataformas Linux, Java 2 Technology, versión 7.

La opción **-Xrealtime** es intercambiable con **-Xgcpolicy:metronome**. Puede especificar cualquiera de ellas para obtener la modalidad en tiempo real.

Opciones AOT (Ahead-of-time)

Definiciones de las opciones AOT (ahead-of-time).

Finalidad

Ninguna opción especificada:
Se ejecuta con el intérprete y el código compilado de forma dinámica. Si se descubre código AOT, no se utiliza. En su lugar, se realiza la compilación dinámica necesaria. Esto resulta especialmente útil para las aplicaciones no de tiempo real y para algunas de tiempo real. Esta opción proporciona un rendimiento óptimo, pero puede sufrir retardos no determinantes durante el tiempo de ejecución, cuando se produce la compilación.

-Xjit: Esta opción es la misma que la predeterminada.

-Xint: Ejecuta solo el intérprete, omite el código escrito para AOT que se puede encontrar en un archivo .jar precompilado y no ejecuta el compilador dinámico. Esta modalidad no suele ser necesaria, más que para los problemas de depuración que crea que están relacionados con la compilación o en aplicaciones por lotes muy breves que no obtengan ventajas de la compilación.

-Xnojit:
Ejecuta el intérprete y utiliza código escrito para AOT si se encuentra en un archivo .jar precompilado. No ejecuta el compilador dinámico. Esta modalidad funciona bien para algunas aplicaciones de tiempo real si desea garantizar que no se produzcan retardos no determinantes durante el tiempo de ejecución debidos a la compilación. El código escrito para AOT solo se puede utilizar al realizar la ejecución con la opción **-Xrealtime**. No se admite cuando la ejecución se realiza en una JVM estándar, es decir, cuando no se ha especificado **-Xrealtime**.

Ejemplo

```
java -Xrealtime -Xnojit prueba_salida.jar.
```

Opciones del recopilador de basura Metronome

Definiciones de las opciones del recopilador de basura Metronome.

-Xgc:immortalMemorySize=*tamaño*
Especifica el tamaño de su área de almacenamiento dinámico inmortal. El valor predeterminado es 16 MB.

-Xgc:scopedMemoryMaximumSize=*tamaño*
Especifica el tamaño de su área de almacenamiento dinámico de memoria de ámbito. El valor predeterminado es 8 MB.

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM
Una de las ocasiones en las que se produce la recogida de basura es cuando el almacenamiento dinámico se queda sin memoria. Si no hay más espacio libre en el almacenamiento dinámico, utilizar **-Xgc:synchronousGCOnOOM** detiene su aplicación mientras la recogida de basura elimina los objetos no utilizados. Si el espacio libre se agota de

nuevo, puede reducir la utilización objetivo para así dejar más tiempo a la recogida de basura para finalizar. Definir **-Xgc:nosynchronousGCOnOOM** implica que cuando la memoria de almacenamiento dinámico está llena, su aplicación se detiene y emite un mensaje de falta de memoria. El valor predeterminado es **-Xgc:synchronousGCOnOOM**.

-Xnoclassgc
Inhabilita la recogida de basura de clases. Esta opción desactiva la recogida de basura del almacenamiento asociado con las clases Java que ya no utiliza la JVM. El comportamiento predeterminado es **-Xnoclassgc**.

-Xgc:targetUtilization=*N*
Define la utilización de la aplicación en N%; el recopilador de basura intenta utilizar como máximo (100-N)% de cada intervalo de tiempo. Los valores razonables se encuentran en el rango 50-80%. Las aplicaciones con una tasa de asignación baja se pueden ejecutar al 90%. El valor predeterminado es 70%.

En este ejemplo se muestra que el tamaño máximo de la memoria de almacenamiento dinámico es 30 MB. El recopilador de basura trata de utilizar el 25% de cada intervalo de tiempo porque la utilización objetivo de la aplicación es del 75%.

```
java -Xrealtime -Xmx30m -Xgc:targetUtilization=75 Test
```

-Xgc:threads=*N*
Especifica el número de hebras del recopilador de basura que se van a ejecutar. El valor predeterminado es 1.

-Xgc:verboseGCCycleTime=N
N es el tiempo en milisegundos durante el que se debe volcar la información de resumen.

Nota: El tiempo de ciclo no implica que la información de resumen se vuelque exactamente a dicha hora, sino cuando pasa el último suceso de recogida de basura que satisface este criterio temporal.

-Xmx<*tamaño*>
Especifica el tamaño del almacenamiento dinámico de Java. Al contrario que otras estrategias de recogida de basura, la recogida de basura Metronome en tiempo real no admite la expansión de almacenamiento dinámico. No hay una opción de tamaño de almacenamiento dinámico máximo o inicial. Solo puede especificar el tamaño de almacenamiento dinámico máximo.

-Xthr:metronomeAlarm=os*xx*
Controla la prioridad con la que se ejecuta la hebra de alarma de Recopilador de basura Metronome.

Donde *xx* es un número de 11 al 89 que especifica la prioridad a la que se debería ejecutar la hebra de alarma metronome. Se debe tener cuidado al modificar la prioridad del sistema operativo con la que se ejecuta la hebra de alarma. Si especifica una prioridad de sistema operativo inferior a la de la hebra en tiempo real, se generarán errores de tipo OutOfMemory porque el recopilador de basura acabará ejecutándose con una prioridad más baja que la de las hebras en tiempo real que asignan la basura. La hebra de alarma Recopilador de basura Metronome predeterminada se ejecuta con una prioridad de sistema operativo de 89.

Valores predeterminados para la JVM

Se aplican los valores predeterminados a JVM de tiempo real si no se realizan cambios en el entorno en el que se ejecuta la JVM. Se muestran los valores comunes para utilizar cómo referencia.

Los valores predeterminados pueden cambiarse mediante variables de entorno o parámetros de línea de mandatos en el inicio de la JVM. La tabla muestra algunos de los valores comunes de la JVM. La última columna indica cómo cambiar el comportamiento cuando son aplicables las siguientes claves:

- **e**: valor controlado solo por variable de entorno
- **c**: valor controlado solo por parámetro de línea de mandatos
- **ec** - valor controlado por variable de entorno y por parámetro de línea de mandatos, pero tiene prioridad este último.

La información se proporciona como una referencia rápida y no está detallada.

Valor de JVM	Valor predeterminado	Valor afectado por
Volcados Java	Habilitado	ec
Volcados Java en falta de memoria	Habilitado	ec
Volcados de almacenamiento dinámico	Inhabilitado	ec
Volcados de almacenamiento dinámico en falta de memoria	Habilitado	ec
Sysdumps	Habilitado	ec
Donde se producen los archivos de volcado	Directorio actual	ec
Salida detallada	Inhabilitado	c
Búsqueda de classpath de arranque	Inhabilitado	c
Comprobaciones JNI	Inhabilitado	c
Depuración remota	Inhabilitado	c
Comprobaciones de cumplimiento estricta	Inhabilitado	c
Inicio rápido	Inhabilitado	c
Servidor de información de depuración remoto	Inhabilitado	c
Señalización reducida	Inhabilitado	c
Encadenamiento de manejador de señal	Habilitado	c
Classpath	No establecida	ec
Compartimiento de datos de clases	Inhabilitado	c
Soporte de accesibilidad	Habilitado	e
compilador JIT	Habilitado	ec
Compilador AOT (la JVM no utiliza AOT a menos que estén también habilitadas las clases compartidas)	Habilitado	c
Opciones de depuración de JIT	Inhabilitado	c
Tamaño máximo Java2D de fonts con negrita algorítmica	14 puntos	e
Mapas de bits representados de uso de Java2D en fonts escalables	Habilitado	e
Trazado de font de tipo libre Java2D	Habilitado	e
Fonts AWT de uso de Java2D	Inhabilitado	e

Valor de JVM	Valor predeterminado	Valor afectado por
Entorno local predeterminado	Ninguno	e
Tiempo de espera antes de iniciar plugin	cero	e
Directorio temporal	/tmp	e
Redirección de plugin	Ninguno	e
Conmutación IM	Inhabilitado	e
Modificadores IM	Inhabilitado	e
Modelo de hebra	N/D	e
Tamaño de pila inicial para hebras Java de 32 bits. Utilice: **-Xiss<tamaño>**	2 KB	c
Tamaño de pila máximo para hebras Java de 32 bits. Utilice: **-Xss<tamaño>**	256 KB	c
Tamaño de pila para hebras de sistema operativo de 32 bits. Utilice **-Xmso<tamaño>**	256 KB	c
Tamaño de almacenamiento dinámico inicial. Utilice **-Xms<tamaño>**	64 MB	c
Tamaño de almacenamiento dinámico Java máximo. Utilice **-Xmx<tamaño>**	Mitad de la memoria disponible con un mínimo de 16 MB y un máximo de 512 MB	c
Uso del intervalo de tiempo de destino para una aplicación. El colector de basura intenta utilizar el resto. Utilice **-Xgc:targetUtilization=<porcentaje>**	70%	c
El número de hebras de colector de basura para ejecutar. Utilice **-Xgc:threads=<valor>**	1	c
Cantidad máxima de memoria que puede asignarse a memorias de ámbito en la modalidad **-Xrealtime**. Utilice **-Xgc:scopedMemoryMaximumSize=<tamaño>**.	8 MB	c
Establece el tamaño de un área de memoria inmortal en la modalidad **-Xrealtime**. Utilice **-Xgc:immortalMemorySize=<tamaño>**	16 MB	c

Nota: "tamaño disponible" es la cantidad real (física) de memoria o el valor de **RLIMIT_AS**, el que sea más pequeño.

Bibliotecas de clases WebSphere Real Time for RT Linux

Referencia a las bibliotecas de clases Java utilizadas por WebSphere Real Time for RT Linux.

Las bibliotecas de clase Java que utiliza WebSphere Real Time for RT Linux se describen en http://www.rtsj.org/specjavadoc/book_index.html.

Ejecución con TCK

Si está ejecutando Especificación de tiempo real para Java (RTSJ) Technology Compatibility Kit (TCK) con WebSphere Real Time for RT Linux, debe incluir `demo/realtime/TCKibm.jar` en la classpath para que las pruebas se realicen correctamente.

`TCKibm.jar` incluye la clase **`VibmcorProcessorLock`**, ampliación de IBM para la clase TCK.ProcessorLock. Esta clase proporciona un comportamiento de procesador único necesario en un pequeño conjunto de pruebas de TCK. Para obtener más información sobre la clase TCK.ProcessorLock y las ampliaciones específicas de esta clase, consulte el archivo readme incluido con la distribución TCK.

Avisos

Esta información se ha desarrollado para productos y servicios que se ofrecen en los EE.UU. IBM puede que no ofrezca los productos, servicios o características que se discuten en este documento en otros países. Consulte a su representante local de IBM para obtener información acerca de los productos y servicios actualmente disponibles en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implica que solo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar, puede utilizarse cualquier producto, programa o servicio funcionalmente equivalente que no infrinja ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, la evaluación y la verificación del funcionamiento conjuntamente con otros productos, programas o servicios que no son de IBM son responsabilidad del usuario.

IBM puede tener aplicaciones patentadas o pendientes de patente que cubran el tema tratado en este documento. La posesión de este documento no le otorga ninguna licencia sobre estas patentes. Puede enviar consultas de licencias por escrito a la dirección siguiente:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para consultas de licencias relacionadas con la información de doble byte (DBCS), póngase en contacto con el Departamento de la propiedad intelectual de IBM de su país o envíe las consultas por escrito a la dirección siguiente:

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

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país en el que tales disposiciones entren en contradicción con la legislación nacional:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN GARANTÍA DE NINGUNA CLASE, YA SEA EXPLICITA O IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN, COMERCIALIZACIÓN O IDONEIDAD PARA UN PROPÓSITO DETERMINADO. Algunos estados no contemplan la limitación de responsabilidades, ni implícitas ni explícitas, en determinadas transacciones, por lo que cabe la posibilidad de que esta declaración no sea aplicable en su caso.

Esta información puede contener imprecisiones técnicas o errores tipográficos. Periódicamente se realizarán modificaciones en la información aquí contenida; dichos cambios se incorporarán en nuevas ediciones de la publicación. IBM puede efectuar mejoras y/o cambios en los productos y/o programas descritos en esta información en cualquier momento y sin previo aviso.

Cualquier referencia hecha en esta información a sitios web que no son de IBM se proporciona únicamente para su comodidad y no debe considerarse en modo alguno como promoción de dichos sitios web. Los materiales de estos sitios Web no forman parte de los materiales de IBM para este producto y el uso que se haga de estos sitios Web es de la entera responsabilidad del usuario.

IBM puede utilizar o distribuir la información que facilite de la manera que considere apropiada sin incurrir en obligaciones con el remitente.

Los poseedores de licencias de este programa que deseen obtener información acerca del mismo con el propósito de permitir (i) el intercambio de información entre programas creados independientemente y otros programas (incluido éste) y (ii) la utilización mutua de la información intercambiada, deben ponerse en contacto con la dirección siguiente:

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

Esta información estará disponible, sujeta a los términos que correspondan, lo que en algunos casos supondrá el pago de una cuota.

IBM proporciona el programa con licencia descrito en este documento y todo el material con licencia disponible para el mismo bajo los términos del Acuerdo de cliente de IBM, el Acuerdo internacional de licencia de programas IBM o cualquier acuerdo equivalente entre las dos partes.

Cualquier información acerca del rendimiento que contenga el presente documento se ha determinado en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Es posible que algunas medidas se hayan tomado en sistemas de nivel de desarrollo y no hay garantías de que estas medidas vayan a ser iguales en los sistemas habitualmente disponibles. Asimismo, algunas mediciones pueden haber sido estimadas mediante extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables para su entorno específico.

La información relativa a productos que no son de IBM se ha obtenido de los proveedores de estos productos, de los anuncios públicos o de otras fuentes disponibles públicamente. IBM no ha probado esos productos y no puede confirmar la precisión del rendimiento, su compatibilidad o cualquier otra afirmación relacionada con productos que no son de IBM. Las preguntas sobre las posibilidades de los productos que no son de IBM deben dirigirse a los proveedores de dichos productos.

Consideraciones acerca de la política de privacidad

Los productos de software de IBM, que incluyen software como soluciones de servicio (“Ofertas de software”), pueden utilizar cookies u otras tecnologías para recopilar información de uso de los productos, para ayudar a mejorar la experiencia final del usuario, para personalizar las interacciones con el usuario final o para otros fines. En muchos casos, las Ofertas de software no recopilan información de identificación personal. Algunas de nuestras Ofertas de software pueden ayudarle a recopilar información de identificación personal. Si la oferta de software utiliza cookies para recopilar información de identificación personal, se establece a continuación información específica sobre el uso de cookies de esta oferta.

Esta oferta de software no utiliza cookies u otras tecnologías para recopilar información de identificación personal.

Si las configuraciones desplegadas para esta oferta de software le ofrecen como cliente la posibilidad de recopilar información de identificación personal de los usuarios finales mediante cookies y otras tecnologías, debe buscar asesoramiento jurídico sobre la legislación aplicable a esa recopilación de datos, que incluye cualquier requisito de aviso y consentimiento.

Para obtener más información sobre el uso de las diferentes tecnologías, incluidas las cookies, para estos fines, consulte: (i) la política de privacidad de IBM, en http://www.ibm.com/privacy; (ii) La Declaración de privacidad en línea de IBM, enhttp://www.ibm.com/privacy/details (en concreto, la sección titulada “Cookies, balizas web y otras tecnologías”); (iii) la “Declaración de privacidad de productos de software y software como servicio de IBM”, en http://www.ibm.com/software/info/product-privacy.

Marcas registradas

IBM, el logotipo de IBM e ibm.com son marcas registradas o marcas comerciales registradas de International Business Machines Corporation en los Estados Unidos y/o en otros países. Si estos u otros términos que son marcas registradas de IBM se marcan como tales la primera vez que aparecen en esta información con un símbolo de marca registrada (® o ™), estos símbolos indican que son marcas registradas de los Estados Unidos o marcas registradas de uso comercial propiedad de IBM en el momento en que se publicó esta información. Estas marcas registradas también pueden ser marcas registradas o marcas registradas de uso comercial en otros países. Existe una lista actual de marcas registradas de IBM en la web, bajo el epígrafe "Información de copyright y marcas registradas" en la página web: http://www.ibm.com/legal/copytrade.shtml.

Adobe, el logotipo de Adobe, PostScript y el logotipo de PostScript son marcas registradas o marcas comerciales de Adobe Systems Incorporated en Estados Unidos o en otros países.

Intel e Itanium son marcas registradas de Intel Corporation o sus filiales en los Estados Unidos y/o en otros países.

Linux es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.

Java y todas las marcas comerciales y logotipos de Java son marcas registradas o marcas comerciales registradas de Oracle y/o sus filiales.

Otros nombres de empresas, productos o servicios pueden ser marcas registradas o de servicio de otros.

Índice

Caracteres Especiales

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

Impreso en España