IBM WebSphere Real Time for RT Linux
Versió 3

Guia de l'usuari

IBM WebSphere Real Time for RT Linux
Versió 3

Guia de l'usuari

Nota
Abans d'utilitzar aquesta informació i el producte que suporta, llegiu la informació de Capítol 11, “Avisos”, a la pàgina 151.

Primera edició (agost de 2011)

Aquesta edició de la guia de l'usuari fa referència al producte IBM WebSphere Real Time for RT Linux, Versió 3, i a totes les versions i modificacions posteriors fins que s'indiqui d'una altra manera en futures edicions.

Contingut

Figures

Taules

Prefaci

Aquesta guia de l'usuari proporciona informació general sobre l'IBM® WebSphere Real Time for RT Linux.

Capítol 1. Introducció

Aquesta informació descriu l'IBM WebSphere Real Time for RT Linux.

Visió general del WebSphere Real Time for RT Linux

El WebSphere Real Time for RT Linux inclou capacitats en temps real amb la màquina virtual IBM J9 (JVM).

El WebSphere Real Time for RT Linux és un JRE (Java Runtime Environment) amb un kit de desenvolupament de programari (SDK) que amplia l'IBM SDK for Java capacitats en temps real. Les aplicacions que depenen d'uns temps de resposta precisos poden aprofitar les característiques en temps real que ofereix el WebSphere Real Time for RT Linux en tecnologia Java estàndard.

Característiques

Les aplicacions en temps real necessiten un temps d'execució consistent en lloc d'una velocitat absoluta.

Quan la JVM s'executa en mode en temps real, hi ha àrees de memòria addicionals disponibles a banda de l'emmagatzematge dinàmic en el qual s'ha fet la recollida de deixalles. Els programes poden sol•licitar o especificar qualsevol nombre d'àrees de memòria amb àmbit reutilitzables i immortals no reutilitzables, en les quals no es duu a terme la recollida de deixalles. Aquesta capacitat proporciona a l'aplicació més control sobre l'ús de memòria. També utilitza el Recollidor de deixalles Metronome per fer la recollida de deixalles en base al temps. Quan la JVM s'executa en un mode de rendiment tradicional, es poden utilitzar diversos recollidors de deixalles basats en la feina que optimitzen el rendiment però poden tenir retards individuals més grans que el Recollidor de deixalles Metronome.

Tot seguit s'indiquen les preocupacions principals en desplegar aplicacions en temps real amb JVM tradicionals:
- Retards imprevisibles (possiblement llargs) de l'activitat de recollida de deixalles (GC).
- Retards a l'execució del mètode quan es produeix una compilació a temps (JIT) i una recompilació, amb la variabilitat en el temps d'execució.
- Planificació arbitrària del sistema operatiu.

El WebSphere Real Time for RT Linux proporciona el següent per eliminar aquests obstacles:
- El Recollidor de deixalles Metronome, un recollidor de deixalles determinista incremental amb temps de pausa molt curts.
- Compilació anticipada (AOT).
- Planificació FIFO basada en prioritats.

A més, el WebSphere Real Time proporciona el programador en temps real amb els recursos RTSJ; vegeu "Suport per al RTSJ" a la pàgina 9.

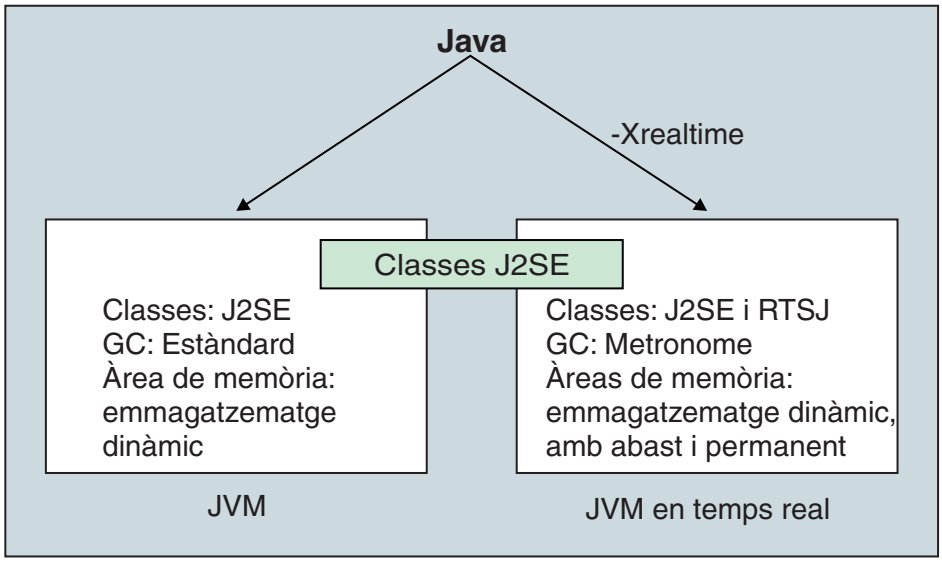

Figura 1. Visió general del WebSphere Real Time for RT Linux

Per habilitar les capacitats en temps real, heu d'utilitzar l'opció -Xrealtime quan executeu la JVM o alguna altra de les eines proporcionades. Per defecte, la JVM i les eines proporcionades s'executen sense les capacitats en temps real habilitades. A la Figura 1 es mostren les relacions de les dues JVM que se subministren amb el WebSphere Real Time for RT Linux.

Les ordres Java següents reconeixen l'opció -Xrealtime:

Taula 1. Ordres Java utilitzades en mode de temps real

Ordre	Funció
java	Per defecte, s'executa en mode estàndard però també s'executa en mode de temps real quan s'especifica l'opció -Xrealtime. En mode de temps real, el programador accedeix a classes del paquet javax.realtime. Podeu utilitzar fitxers jar precompilats i la tecnologia de recollida de deixalles determinista Metronome.
javac, javah, javap	Executa un mode estàndard per defecte; això no obstant, quan s'especifica l'opció -Xrealtime, inclou les classes javax.realtime.* a la variable classpath.
admincache	Es pot executar amb i sense -Xrealtime, però només es pot emplenar una memòria cau compartida amb l'eina **admincache** en mode de temps real. En mode normal, només hi ha disponibles les utilitats de memòria cau (com ara listAllCaches o printStats). Com passa amb **jdmpview**, **admincache** s'ha d'executar amb -Xrealtime per accedir a les memòries cau per a la JVM en temps real, i s'ha d'executar sense -Xrealtime per accedir a les memòries cau per a la JVM normal.
jextract	jextract s'executa per defecte en mode de temps real, però s'ha d'executar amb l'opció -Xrealtime quan es processen abocaments de memòria del sistema generats per la JVM en mode de temps real.

Novetats

Aquest tema presenta els canvis de l'IBM WebSphere Real Time for RT Linux.

WebSphere Real Time for RT Linux V3

El WebSphere Real Time for RT Linux V3 és una extensió de l'IBM SDK for Java 7, que es basa en les característiques i funcions disponibles en aquesta versió i hi afegeix capacitats en temps real. Les versions anteriors del WebSphere Real Time for RT Linux estaven basades en versions anteriors de l'IBM SDK for Java.

Per obtenir més informació sobre les novetats, consulteu: Novetats a l'Information Center de l'IBM SDK for Java 7.

jxeinajar

El WebSphere Real Time for RT Linux V3 ja no admet l'ús de jxeinajar. Per motius de consulta, la informació sobre jxeinajar i, concretament, sobre la migració a admincache es troba a la documentació del WebSphere Real Time for RT Linux V2.

Avantatges

Els avantatges de l'entorn en temps real són que les aplicacions Java s'executen amb un grau superior de predicció que amb la JVM estàndard i proporcionen un comportament de temporització constant per a l'aplicació Java. Les activitats de segon pla, com ara la compilació i la recollida de deixalles, es produeixen en moments determinats i, per tant, eliminen els pics inesperats d'activitat de segon pla quan s'executa l'aplicació.

Aquests avantatges s'obtenen ampliant la JVM amb aquestes funcions:

- Tecnologia de recollida de deixalles en temps real Metronome
- Compilació de manera anticipada (AOT)
- Suport per a Especificació de temps real per a Java (RTSJ)

Totes les aplicacions Java es poden executar en un entorn en temps real sense modificació, la qual cosa suposa un avantatge per al recollidor de deixalles Metronome i és una recollida de deixalles determinista que es produeix de manera periòdica. Per obtenir els avantatges màxims del WebSphere Real Time for RT Linux, podeu escriure aplicacions específicament per a l'entorn en temps real mitjançant el fils en temps real i el fils en temps real que no són d'emmagatzematge dinàmic. L'enfocament que preneu depèn de l'especificació de temps de la vostra aplicació.

Moltes de les aplicacions Java en temps real poden explotar els períodes curts de pausa del Recollidor de deixalles Metronome i AOT per assolir els seus objectius, de manera que mantenen els avantatges de la portabilitat Java. Les aplicacions que tenen requisits més estrictes han d'utilitzar els recursos RTSJ del fils en temps real i del fils en temps real que no són d'emmagatzematge dinàmic, amb memòria amb àmbit i immortal. Aquest enfocament limita el fet que la vostra aplicació s'executi només en un entorn en temps real, i perdeu l'avantatge de la portabilitat a JSE Java. També cal desenvolupar un model de programació més complex.

Capítol 2. Descripció de l'IBM WebSphere Real Time for RT Linux

En aquest apartat es presenten els components clau de l'IBM WebSphere Real Time for RT Linux.

Introducció al recollidor de deixalles Metronome

El recollidor de deixalles Metronome substitueix el recollidor de deixalles estàndard al WebSphere Real Time for RT Linux.

La diferència clau entre la recollida de deixalles Metronome i la recollida de deixalles estàndard és que la primera es produeix en petits passos que es poden interrompre, mentre que la segona atura l'aplicació mentre marca i recull deixalles.

Per exemple:

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

L'exemple especifica que la vostra aplicació s'executa al 80% cada 60 ms. El 20% del temps que queda pot utilitzar-se per a la recollida de deixalles, si n'hi ha. El recollidor de deixalles Metronome garanteix nivells d'ús sempre que tingui prou recursos. La recollida de deixalles s'inicia quan la quantitat d'espai lliure a l'emmagatzematge dinàmic és inferior al llindar determinat de manera dinàmica.

Recollida de deixalles i prioritats

El fil de recollida de deixalles s'ha d'executar a una prioritat superior al fil de prioritat més alta que genera deixalles a l'emmagatzematge dinàmic; en cas contrari, pot ser que no s'executi segons especifica l'ús configurat. Tant els fils Java normals com els fils tipus fils en temps real poden generar deixalles i, per tant, la recollida de deixalles s'ha d'executar a una prioritat més alta que els fils tipus fils en temps real i els fils normals. Aquesta priorització la gestiona automàticament la JVM i la recollida de deixalles s'executa a una prioritat 0,5 punts superior a la prioritat més alta de tots els fils tipus fils en temps real i els fils normals. Això no obstant, és important garantir que els fils NHRT (fils en temps real que no són d'emmagatzematge dinàmic) no es vegin afectats per la recollida de deixalles. Executeu tots els NHRT a una prioritat més alta que el fils en temps real de prioritat més alta. Això vol dir que els fils NHRT s'executen a una prioritat superior a la de la recollida de deixalles i no pateixen cap retard.

A la Taula 2 a la pàgina 6 es mostra un exemple típic de prioritats que podeu definir i les prioritats de recollida de deixalles relacionades que es deriven de la vostra elecció.

Per comparar les prioritats Java amb les prioritats del sistema operatiu, vegeu “Mapatge de prioritats i herència” a la pàgina 12.

Taula 2. Exemple de recollida de deixalles i prioritats

Fils	Prioritats (exemples)
Si el fil en temps real que té la prioritat més alta és:	20 (prioritat del sistema operatiu 43)
El recollidor de deixalles és:	20,5 (prioritat del sistema operatiu 44)
Per garantir que un NHRT s'executa independentment del recollidor de deixalles, definiu per a aquest fil una prioritat més alta que la de la recollida de deixalles.	21 (prioritat del sistema operatiu 45) o superior.
El fil d'alarma Metronome és:	Prioritat 46 (prioritat del sistema operatiu 89)

Nota: Fins i tot amb aquesta configuració, els fils tipus fils en temps real que no són d'emmagatzematge dinàmic no es lliuren del tot de l'efecte de la recollida de deixalles perquè el fil de l'alarma Metronome s'executa a la prioritat més alta del sistema per garantir que es pot activar periòdicament i que pot funcionar si la recollida de deixalles ha de fer res. La feina a fer és, evidentment, petita i, per tant, no cal fer-hi massa esment.

Recollida de deixalles Metronome i baixada de classes

El recollidor de deixalles Metronome no baixa classes a l'IBM WebSphere Real Time perquè pot ser que necessiti una quantitat no determinista de feina que generi valors atípics de tems de pausa.

Fils de recollidor de deixalles Metronome

El recollidor de deixalles Metronome té dos tipus de fil: un únic fil d'alarma i diversos fils de recollida (GC). Per defecte, hi ha un fil GC. Podeu definir el nombre de fils de recollida de deixalles per a la JVM mitjançant l'opció **-Xgcthreads**.

No podeu canviar el nombre de fils d'alarma per a la JVM.

El recollidor de deixalles Metronome verifica periòdicament la JVM per veure si la memòria de l'emmagatzematge dinàmic té prou espai lliure. Quan la quantitat d'espai lliure és inferior al límit, el recollidor de deixalles Metronome activa la JVM per iniciar la recollida de deixalles.

Fil d'alarma

El fil d'alarma únic garanteix l'ús d'un nombre mínim de recursos. S'“activa” de manera periòdica i fa aquestes comprovacions:

- La quantitat d'espai lliure a la memòria d'emmagatzematge dinàmic
- Si s'està executant la recollida de deixalles

Si no hi ha prou espai lliure disponible i no s'està executant la recollida de deixalles, el fil d'alarma activa els fils de recollida per iniciar la recollida de deixalles. El fil d'alarma no fa res fins a la propera hora planificada per verificar la JVM.

Fils de recollida

Cada fil de recollida verifica Java i els fils de tipus fils en temps real per als objectes d'emmagatzematge dinàmic. Verifiquen les àrees de memòria en aquesta seqüència:

1. Memòria amb àmbit per identificar i marcar els objectes actius a l'emmagatzematge dinàmic que estan essent utilitzats per objectes de la memòria amb àmbit.
2. Memòria immortal per identificar i marcar els objectes actius a l'emmagatzematge dinàmic que estan essent utilitzats per objectes de la memòria immortal.
3. Memòria d'emmagatzematge dinàmic per identificar i marcar els objectes actius.

Una vegada que s'han marcat els objectes actius, els objectes no marcats estan disponibles per a la recollida.

Quan el cicle de recollida de deixalles ha finalitzat, el recollidor de deixalles Metronome verifica la quantitat d'espai d'emmagatzematge dinàmic lliure. Si encara no hi ha prou espai d'emmagatzematge dinàmic lliure, s'inicia un altre cicle de recollida de deixalles utilitzant el mateix ID d'activador. Si ni ha prou espai d'emmagatzematge dinàmic lliure, l'activador finalitza i els fils de recollida de deixalles s'aturen. El fil d'alarma segueix supervisant l'espai d'emmagatzematge dinàmic lliure i activarà un altre cicle de recollida de deixalles quan sigui necessari.

Per obtenir més informació sobre la utilització del recollidor de deixalles Metronome, consulteu "Utilització del recollidor de deixalles Metronome" a la pàgina 66.

Compiladors

L'IBM WebSphere Real Time for RT Linux admet diversos models de compilació de codi, que proporcionen diferents nivells de rendiment i determinisme de codi.

Les opcions disponibles per compilar codi Java amb l'IBM WebSphere Real Time for RT Linux inclouen:

Compilació a temps (JIT) de prioritat baixa
El model de compilació per defecte del WebSphere Real Time for RT Linux utilitza un compilador JIT per compilar els mètodes importants d'una aplicació Java mentre s'executa l'aplicació. En aquest mode, el compilador JIT funciona de manera semblant a l'operació del compilador JIT en una JVM que no és en temps real. La diferència rau en que el compilador JIT del WebSphere Real Time for RT Linux s'executa a una prioritat més baixa que els fils en temps real. La prioritat més baixa vol dir que el compilador JIT utilitza recursos del sistema quan l'aplicació no necessita dur a terme tasques en temps real. L'efecte és que el compilador JIT no afecta significativament el rendiment de les tasques en temps real.

Codi precompilat de manera anticipada (AOT)
El WebSphere Real Time for RT Linux compila mètodes Java en codi natiu en un pas de compilació prèvia abans d'executar l'aplicació. L'ús de codi precompilat AOT proporciona el nivell més alt de determinisme, amb un bon rendiment.

Mode combinat: combinació de codi precompilat AOT i compilació JIT de baixa prioritat
El codi compilat AOT i JIT es pot utilitzar conjuntament mentre s'executa l'aplicació. Aquest mode de funcionament pot proporcionar un determinisme molt bo amb un bon rendiment, i un rendiment molt alt per a mètodes que s'executen sovint.

Operació interpretada
L'intèrpret executa una aplicació Java, però no utilitza compilació de codi.

Per obtenir més informació sobre la utilització de codi compilat, consulteu "Ús del codi compilat amb l' WebSphere Real Time for RT Linux" a la pàgina 36.

Comparació de la compilació JIT i AOT

La compilació anticipada (AOT) permet compilar les classes i mètodes Java abans d'executar el codi. La compilació AOT evita l'efecte de temps imprevisible que pot tenir el compilador JIT en vies de rendiment sensibles. Per tal de garantir que el codi es compila abans d'ésser executat i per obtenir el nivell més alt de rendiment determinista, podeu precompilar el codi en una memòria cau de classes compartida mitjançant el compilador AOT.

Nota: El codi compilat de manera anticipada (AOT) normalment no s'executa tan ràpid com el codi compilat a temps (JIT).

El compilador a temps (JIT) s'executa com a fil SCHED_OTHER d'alta prioritat, per sobre de la prioritat dels fils Java estàndard, però per sota de la prioritat dels fils en temps real. La compilació JIT, per tant, no provoca retards no deterministes en el codi en temps real. Com a conseqüència, la feina en temps real important es duu a terme a temps perquè no el compilador JIT no s'hi anticipa. Això no obstant, el codi en temps real es pot executar com a codi interpretat, perquè el JIT no ha tingut prou temps per compilar els mètodes importants que s'han col•locat a la cua. A la Figura 2 a la pàgina 9 es mostra una comparació.

En general, si la aplicació té una fase d'escalfament, és més efectiu executar-la amb JIT i, si cal inhabilitar el JIT quan la fase d'escalfament finalitza. Aquest enfocament permet al compilador JIT generar codi per a l'entorn en el qual s'executa l'aplicació.

Si l'aplicació no té fase d'escalfament i no queda clar si les vies clau d'execució es compilen mitjançant el funcionament de l'aplicació estàndard, la compilació AOT és adequada en aquest entorn.

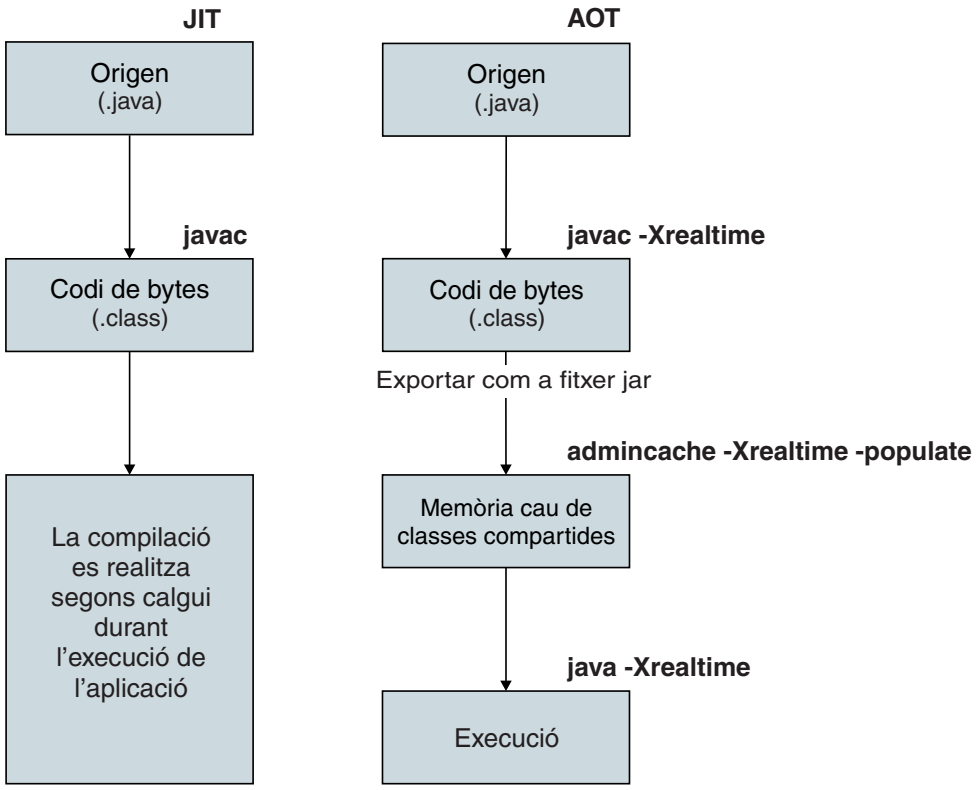

Figura 2. Comparació del compilador JIT amb el compilador AOT

Suport per al RTSJ

El WebSphere Real Time for RT Linux implementa l'especificació Especificació de temps real per a Java (RTSJ).

El WebSphere Real Time for RT Linux versió 3.0 ha estat certificat com a compatible amb RTSJ davant de l'RTSJ Technology Compatibility Kit 1.0.2 versió J9 3.1.0 FCS, i és compatible amb el Java Compatibility Kit (JCK) versió 7.0.

Planificació i distribució de fils en temps real

La planificació i distribució de fils Java en temps real forma part de l'especificació en temps real per a Java. La política de planificació SCHED_FIFO s'utilitza per prioritzar els fils Java en temps real mitjançant les prioritats 11 - 89 del sistema operatiu Linux.

Per obtenir informació sobre les polítiques de planificació del Linux, vegeu "Planificació i distribució de fils" a la pàgina 35.

Objectes planificables i els seus paràmetres

Hi ha dos tipus principals d'objectes planificables en temps real: fils en temps real i gestors d'esdeveniments asíncrons.

Aquests objectes planificables tenen els paràmetres següents associats:

SchedulingParameters
PriorityParameters planifica objectes planificables en temps real per prioritat.

ReleaseParameters
- **PeriodicParameters** descriu la publicació periòdica dels objectes planificables en temps real. Un fil en temps real periòdica és un fil que es publica amb una freqüència periòdica.
- **AperiodicParameters** descriu la publicació d'objectes planificables en temps real. Els fils en temps real no periòdics es publiquen amb una freqüència irregular.

MemoryParameters
Descriu les restriccions d'assignació de memòria per a objectes planificables en temps real.

ProcessingGroupParameters
No s'admet al WebSphere Real Time for RT Linux.

Planificador de prioritats

Al WebSphere Real Time for RT Linux, el planificador és un planificador de prioritats. Com el seu nom indica, gestiona l'execució dels objectes planificables en funció de les seves prioritats actives.

El planificador manté la llista d'objectes planificables i determina quan es pot publicar cada objecte per executar-se a la CPU. El planificador s'ha de regir pels diferents paràmetres que hi ha associats a cada objecte planificable. Per a aquesta finalitat, es proporcionen els mètodes addToFeasibility, isFeasible i removeFromFeasibility.

Prioritats i polítiques

Els fils Java regulars, és a dir, els fils assignats com a objectes `java.lang.Thread`, poden utilitzar les polítiques de planificació SCHED_OTHER, SCHED_RR o SCHED_FIFO. Els fils en temps real, és a dir, els fils assignats com a `java.lang.RealtimeThread`, i els gestors d'esdeveniments asíncrons utilitzen la política de planificació SCHED_FIFO.

Els fils Java regulars utilitzen la política de planificació per defecte SCHED_OTHER, tret que la JVM s'hagi iniciat amb un fil amb la política SCHED_RR o SCHED_FIFO. Els fils Java regulars que utilitzen la política SCHED_OTHER tenen la prioritat de fil del sistema operatiu definida a 0. Els fils Java regulars que utilitzen la política SCHED_RR o SCHED_FIFO hereten la prioritat del fil que inicia la JVM.

Per al fils en temps real, la política SCHED_FIFO no té porció de temps i admet 99 prioritats de la 1 (la més baixa) a la 99 (la més alta). Aquesta implementació del WebSphere Real Time for RT Linux admet 28 prioritats d'usuari en l'interval inclusiu 11-38, per tant:

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

retorna 11, i:

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

retorna 38.

Les prioritats del sistema operatiu 81 - 89 s'utilitzen a l'IBM JVM per distribuir fils de treball. Tots aquest fils estan dissenyats per fer un volum petit de treball abans de tornar a l'estat d'inactivitat. Els fils són d'aquesta manera:

- El fil d'alarma Recollidor de deixalles Metronome s'executa a la prioritat 89. Aquest fil s'executa periòdicament i distribueix una unitat de treball GC.
- Dos fils de senyal síncron, que processen senyals asíncrons; on un és un fil de temps real que no és d'emmagatzematge dinàmic (NHRT) a la prioritat 88 i l'altre és un fil de la prioritat 87.
- Dos fils de temporitzador, que distribueixen esdeveniments de temporitzador; un és un fil en temps real que no és d'emmagatzematge dinàmic per als temporitzadors que no són d'emmagatzematge dinàmic i a la prioritat 85 i l'altre és a la prioritat 83.
- Els fils de gestor d'esdeveniments asíncrons, que es distribueixen per executar gestors d'esdeveniments asíncorns i, mentre executen un gestor d'esdeveniments asíncrons, se'ls assigna la prioritat del gestor. El sistema s'inicia amb dos fils de gestor en temps real que no són d'emmagatzematge dinàmic a la prioritat 85 i 8 fils més a la prioritat 83.
- El fil en temps real que no és d'emmagatzematge dinàmic i de senyal asíncron de la prioritat 88 gestiona sol•licituds per als abocaments de memòria d'emmagatzematge dinàmic, abocaments de memòria de nucli i abocaments de memòria javacore. Temporalment incrementa la seva prioritat a 89 mentre crea els fitxers d'abocament de memòria.

El fil de traça GC Metronome s'executa a la prioritat 12 del sistema operatiu, i el fil d'exemple JIT, que mostra mètodes Java per a compilació, s'executa a la prioritat 13 del sistema operatiu.

El fil de compilació JIT (que és diferent del fil d'exemple JIT) s'executa amb la política SCHED_OTHER a la prioritat 0 del sistema operatiu.

Els fils de compilació JIT i exemple JIT s'inhabiliten si s'especifica **-Xnojit** o **-Xint**.

La prioritat de Recollidor de deixalles Metronome i del finalitzador canvia constantment (abans de cada ronda de recopilació) perquè estigui per sobre del fil d'assignació d'emmagatzematge dinàmic de prioritat més alta. Heu d'assegurar-vos que la prioritat dels fils d'assignació d'emmagatzematge dinàmic estigui per sota de la dels fils NoHeapRealtimeThreads.

Un fil d'assignació d'emmagatzematge dinàmic és qualsevol fil d'usuari que no sigui NHRT i que no estigui inactiu ni blocat en un supervisor. Un fil d'usuari que executa codi natiu fora de la interfície JNI no es considera fil d'assignació d'emmagatzematge dinàmic. Si hi ha una operació de recollida de deixalles en curs quan el fil d'assignació d'emmagatzematge dinàmic s'activa, no està blocat per un supervisor o abandona JNI, es força el fil perquè esperi fins que finalitzi l'operació de recollida de deixalles abans de continuar.

La prioritat del sistema operatiu 81 es reserva per a fils JVM interns que assignen des de l'emmagatzematge dinàmic. Si un fil de JVM intern és a la prioritat 81 del sistema operatiu, el recollidor de deixalles s'executa a la prioritat 82 del sistema operatiu. Quan els fils d'usuari d'assignació d'emmagatzematge no són fils en temps real, la prioritat de recollida de deixalles (GC) s'executa a la prioritat 11 del sistema operatiu. En cas contrari, la GC s'executa a la prioritat que és una prioritat de sistema operatiu superior que el fil d'usuari d'assignació d'emmagatzematge dinàmic de prioritat més alta.

La prioritat de GC s'ajusta just abans de fer una ronda de recopilació.

Mapatge de prioritats i herència

Cada prioritat Java es mapa amb una prioritat bàsica del sistema operatiu associat, i cada prioritat del sistema operatiu s'associa amb una política de planificació. Les polítiques de planificació del sistema operatiu Linux del WebSphere Real Time for RT Linux són SCHED_OTHER, SCHED_RR i nd SCHED_FIFO.

Els fils Java en temps real utilitzen la política SCHED_FIFO, mentre que els fils Java normals utilitzen la política del fil que inicia la JVM. La política de planificació per defecte per als fils Java regulars és SCHED_OTHER, però podeu utilitzar una utilitat com ara **chrt** per definir les polítiques SCHED_RR o SCHED_FIFO. Per obtenir més informació sobre les prioritats i les polítiques de fils, vegeu "Planificació i distribució de fils" a la pàgina 35.

A la taula següent es mostra com es mapen les prioritats Java amb les prioritats natives del sistema operatiu. Algunes del es prioritats Java es reserven per ésser utilitzades a la JVM, i algunes prioritats natives que no tenen cap correspondència amb les prioritats Java també s'utilitzen a la JVM.

Nota:

- Les prioritats 1-10 s'utilitzen per a fils Java regulars.
  - Per a la política SCHED_OTHER, les prioritats Java 1-10 es mapen amb la prioritat 0 del sistema operatiu.
  - Per a les polítiques SCHED_FIFO o SCHED_RR, les prioritats Java 1-10 hereten la prioritat del fil que inicia la JVM.
- Les prioritats 11 i superiors s'utilitzen al fils en temps real i al fils en temps real que no són d'emmagatzematge dinàmic
- Un objecte planificable sempre s'executa amb la seva prioritat activa. La prioritat activa és inicialment la prioritat base de l'objecte planificable, però la prioritat activa es pot augmentar temporalment segons l'herència de prioritats. La prioritat base d'un objecte planificable es pot canviar durant la seva execució.

Prioritats base d'usuari:

```
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
```

Prioritats bàsiques internes:

```
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
```

Vegeu també la secció sobre sincronització a http://www.rtsj.org/specjavadoc/book_index.html.

Herència de prioritats:

La prioritat activa d'un fil es pot incrementar temporalment perquè conté un bloc necessari per a un fil de prioritat més alta. Aquests bloqueigs poden bloqueigs interns de la JVM o supervisors de nivell d'usuari associats amb mètodes sincronitzats o bloqueigs sincronitzats. La prioritat d'un fil Java normal, per tant, pot tenir temporalment una prioritat en temps real fins al punt en què el fil ha alliberat el bloqueig.

Una conseqüència de l'herència de prioritats és que la política de fil d'un fil SCHED_OTHER es canvia temporalment a SCHED_FIFO.

Per obtenir informació sobre aquestes prioritats bàsiques i actives, vegeu la secció sobre sincronització a l'especificació RTSJ.

Gestió de la memòria

L'emmagatzematge dinàmic de memòria de recollida de deixalles sempre s'ha considerat com un obstacle per a la programació en temps real a causa del comportament imprevisible introduït per la recollida de deixalles. El recollidor de deixalles Metronome de l'IBM WebSphere Real Time for RT Linux pot proporcionar un rendiment de recollida de deixalles molt determinista. Alhora, RTSJ (Especificació de temps real per a Java) proporciona diverses extensions al model de memòria per als objectes que són fora de l'emmagatzematge dinàmic de les deixalles recollides, de manera que el programador Java pot gestionar de manera explícita els objectes de vida curta i llarga.

Àrees de memòria

L'RTSJ introdueix el concepte d'una àrea de memòria que es pot utilitzar per assignar objectes. Hi ha algunes àrees de memòria que existeixen fora de l'emmagatzematge dinàmic i que poden restriccions quant a allò que el sistema i el recollidor de deixalles poden fer amb els objectes. Per exemple, mai no es recullen les deixalles els objectes de la mateixa àrea de memòria, però el recollidor de deixalles pot deixar aquestes àrees per a referències a qualsevol objecte de l'emmagatzematge dinàmic per mantenir la integritat de l'emmagatzematge dinàmic.

La gestió de memòria té tres tipus bàsics:

- La memòria d'emmagatzematge dinàmic és l'emmagatzematge dinàmic Java tradicional, però es gestiona mitjançant el Recollidor de deixalles Metronome.
- La memòria amb àmbit s'ha de ser sol•licitada de manera explícita per les aplicacions i només la poden utilitzar fils en temps real, incloent-hi fils de temps real que no són d'emmagatzematge dinàmic i gestors d'esdeveniments asíncrons que no són d'emmagatzematge dinàmic.
- La memòria immortal representa un àrea de memòria que conté objectes als quals pot fer referència qualsevol objecte planificable, específicament incloent-hi fils en temps real que no són d'emmagatzematge dinàmic i gestors d'esdeveniments asíncrons que no són d'emmagatzematge dinàmic. S'utilitza per a la inicialització estàtica i de càrrega de classes encara que l'aplicació no la utilitzi.

La memòria immortal o amb àmbit es pot designar per utilitzar memòria física, formada per regions de memòria que tenen característiques específiques, com ara un accés substancialment més ràpid. En general, la memòria física no s'utilitza sovint i és poc probable que afecti l'usuari estàndard de la JVM.

Memòria d'emmagatzematge dinàmic

La mida màxima es controla mitjançant **-Xmx**, però cal recordar que *no* s'ha de definir la mida de l'emmagatzematge dinàmic inicial (**-Xms**) ni definir-la en una mida màxima d'emmagatzematge dinàmic **-Xmx**, perquè, en temps real, l'emmagatzematge dinàmic mai no s'amplia de la mida d'emmagatzematge dinàmic inicial a la mida màxima d'emmagatzematge dinàmic. Quan s'assoleix la mida màxima d'emmagatzematge dinàmic sense espai lliure, es produeix un error OutOf MemoryError. En general, la JVM en temps real consumeix més memòria d'emmagatzematge dinàmic que la JVM tradicional, perquè per admetre la recollida determinista cal organitzar els objectes de manera diferent, la qual cosa dóna a lloc a una major fragmentació de l'emmagatzematge dinàmic. A més, les matrius es divideixen en fragments, cadascun dels quals té una capçalera. Depèn de la proporció d'objectes grans a petits i de la quantitat d'ús de matriu, però és molt probable que es trobi una aplicació que necessiti un 20% més d'espai d'emmagatzematge dinàmic.

El Recollidor de deixalles Metronome és semblant al recollidor "més simultani" que hi ha a la JVM principal quant a que recull les deixalles mentre s'executa l'aplicació. En un món perfecte, el cicle de recollida finalitza abans que l'aplicació es quedi sense memòria, però algunes aplicacions amb índex d'assignació molt alts poden assignar amb més rapidesa de la que el Recollidor de deixalles Metronome pot recollir. La velocitat de recollida depèn de diversos controls detallats, però hi ha un control que força que Metronome passi a la recollida de deixalles tradicional en què s'atura el món abans de generar un error OutOf MemoryError. El paràmetre d'execució és **-Xgc:synchronousGCOnOOM** i la contrapartida és **-Xgc:nosynchronousGCOnOOM**. El valor per defecte és **-Xgc:synchronousGCOnOOM**.

Memòria amb àmbit

L'RTSJ introdueix el concepte de memòria amb àmbit. Es pot utilitzar en objectes que tenen un període de vida ben definit. Un àmbit es pot introduir de manera explícita o bé es pot ajuntar a un objecte planificable (un fil en temps real o un gestor d'esdeveniments asíncrons) que introdueix de manera efectiva l'àmbit abans d'executar el mètode run() de l'objecte. Cada àmbit té un recompte de referència i, quan arriba a zero, els objectes que resideixen a l'àmbit es poden tancar (finalitzar) i la memòria associada amb aquest àmbit s'allibera. La reutilització de l'àmbit queda blocada fins que no acaba la finalització.

La memòria amb àmbit es pot dividir en dos tipus: VTMemory i LTMemory. Aquests tipus de memòria amb àmbit varien segons el temps necessari per assignar objectes de l'àrea. La memòria LTMemory garanteix l'assignació en temps lineal quan el consum de memòria de l'àrea de memòria és inferior a la mida inicial de l'àrea de memòria. La memòria VTMemory no ofereix aquesta garantia.

Els àmbits es poden imbricar. Quan s'entra en un àmbit imbricat, totes les assignacions posteriors s'agafen de la memòria associada amb el nou àmbit. Quan l'àmbit imbricat s'ha completat, l'àmbit anterior es restaura i totes les assignacions posteriors es tornen a agafar d'aquest àmbit.

Atès que els objectes amb àmbit tenen un període de vida, cal limitar les referències als objectes amb àmbit mitjançant un conjunt restringit de regles d'assignació. Una referència a un objecte amb àmbit no es pot assignar a una variable d'un àmbit extern, o a un camp d'un objecte de l'àrea d'emmagatzematge dinàmic o de l'àrea immortal. Una referència a un objecte amb àmbit només es pot assignar en el mateix àmbit o en un àmbit intern. La màquina virtual detecta els intents d'assignació incorrectes i genera una excepció IllegalAssignmentError quan es produeixen. La flexibilitat proporcionada en al selecció dels tipus de memòria amb àmbit permet que l'aplicació utilitzi l'àrea de memòria que té característiques que són adequades per a una regió del codi definida amb una sintaxi determinada.

La mida de l'àrea s'ha d'especificar durant la construcció de l'àrea i el paràmetre de línia d'ordres **-Xgc:scopedMemoryMaximumSize** en controla el valor màxim. El valor per defecte és 8 MB i és adequat per a la majoria de casos.

Memòria immortal

La memòria immortal és un recurs de memòria que es comparteix entre tots els objectes i fils planificables d'una aplicació. Els objectes assignats a la memòria immortal sempre estan disponibles per als fils que no són d'emmagatzematge dinàmic i per als gestors d'esdeveniments asíncrons, i no estan subjectes a retards causats per la recollida de deixalles. El sistema allibera objectes quan finalitza el programa.

La mida es controla mitjançant **-Xgc:immortalMemorySize**; per exemple, **-Xgc:immortalMemorySize=20m** estableix la mida en 20 MB. El valor per defecte és 16 MB, que acostuma a ser adequat, tret que carregueu moltes classes. La càrrega de classes és la causa més probable de les excepcions OutOfMemoryError.

Estimació del requisits de memòria

Tot seguit s'indica com obtenir la informació necessària per assignar memòria suficient.

Un enfocament raonable és identificar la memòria necessària per contenir els objectes esperats, amb un marge de seguretat sensible. L'anàlisi de l'aplicació ajuda a identificar el nombre i la naturalesa dels objectes necessaris, tot i que la mida real necessària per a un objecte pot variar d'un sistema a un altre. L'ús de la classe SizeEstimator té en compte la mida de l'objecte real i proporciona una informació més portable.

Classe SizeEstimator

La classe SizeEstimator proporciona informació d'ajuda sobre la quantitat de memòria necessària per emmagatzemar un objecte. L'estimació és una indicació de l'espai mínim de memòria que s'hauria d'assignar per a l'objecte, i no té en compte

els requisits de memòria per a cap altre recurs que pugui ser necessari per a l'objecte, per exemple, durant la seva construcció.

Per obtenir més informació sobre aquesta classe, vegeu http://www.rtsj.org/specjavadoc/book_index.html

Utilització de la memòria

Comparació de fils de tipus Java, fils en temps real i fils en temps real que no són d'emmagatzematge dinàmic.

L'RTSJ (Especificació de temps real per a Java) afegeix dues classes de fils en temps real de suport: la classe RealtimeThread i la classe NoHeapRealtimeThread.

- Els fils en temps real i els NHRT (fils en temps real que no són d'emmagatzematge dinàmic) són objectes planificables. Com a objectes planificables, tenen aquests paràmetres: alliberament, planificació, memòria i grup de processament.
- Els fils en temps real poden accedir a objectes de la memòria d'emmagatzematge dinàmica i també de la memòria amb àmbit i la immortal.
- Els fils en temps real que no són de l'emmagatzematge dinàmic només accedeixen a les àrees de memòria amb àmbit i memòria immortal.
- Els fils en temps real que no són d'emmagatzematge dinàmic necessiten una prioritat superior a altres fils en temps real. Si la seva prioritat és inferior a la d'altres fils en temps real, perden l'avantatge d'executar-se sense interferències del recollidor de deixalles.

Nota: Un fil en temps real que no és d'emmagatzematge dinàmic amb una prioritat superior a la d'altres fils en temps real no serà interromput per la recollida de deixalles.

Taula 3. Accés a memòria dels fils en temps real i dels fils en temps real que no són d'emmagatzematge dinàmic

Fils	Memòria immortal	Memòria amb àmbit	Memòria d'emmagatzematge dinàmic
Fils normals	✓	✗	✓
Fils en temps real	✓	✓	✓
fils en temps real que no són d'emmagatzematge dinàmic	✓	✓	✗

Tipus d'àrea de memòria

Memòria immortal

La memòria immortal no està subjecta a la recollida de deixalles. Després d'assignar espai a la memòria immortal, l'espai no es pot reclamar fins que l'aplicació no existeix.

- A causa de la naturalesa de la memòria immortal, pot ser que us interessi trobar maneres de reutilitzar la memòria. Una possibilitat és crear una agrupació d'objectes reutilitzables. L'ús de memòria amb àmbit és una alternativa.

- Els objectes de la memòria immortal no poden fer referència a res de la memòria amb àmbit. Si un cap d'un objecte de la memòria immortal s'assigna a un objecte de la memòria amb àmbit, es genera una excepció IllegalAssignmentError.

Memòria amb àmbit
La memòria amb àmbit es pot utilitzar com a àrea de memòria inicial d'un objecte planificable. Quan ja no s'hi fa referència, s'esborren tots els objectes de l'àrea. Els objectes planificables que s'executen en una àrea de memòria amb àmbit fan totes les seves assignacions d'objecte des d'aquesta àrea. Quan una àrea de memòria amb àmbit no s'utilitza, els objectes que conté finalitzen i es reclama la memòria, preparant l'àmbit per a ésser reutilitzat. Quan l'àrea de memòria amb àmbit ja no està disponible per a cap objecte planificable, la memòria es reclama per a altres usos.

L'àrea de memòria descrita per una instància ScopedMemory no existeix a l'emmagatzematge dinàmic Java i no està subjecta a la recollida de deixalles. És segur utilitzar un objecte ScopedMemory com a àrea de memòria inicial associat amb un fil NoHeapRealtimeThread o entrar a l'àrea de memòria mitjançant un mètode ScopedMemory.enter dins d'un fil NoHeapRealtimeThread.

Memòria física
Utilitzeu la memòria física quan les característiques de la memòria siguin importants, per exemple, memòria no volàtil i memòria que no permet paginació.

Esquema d'assignació de temps lineal (LTMemory)
L'àrea LTMemory representa una àrea de memòria garantida pel sistema per tenir assignació de temps lineal quan el consum de memòria de l'àrea de memòria és inferior a la mida inicial de l'àrea de memòria. El temps d'execució per a l'assignació pot variar quan el consum de memòria és entre la mida inicial i la mida màxima de l'àrea de memòria. A més, el sistema subjacent no cal que garanteixi que sempre hi ha memòria disponible entre la mida inicial i la mida màxima.

Esquema d'assignació de temps variable (VTMemory)
L'àrea VTMemory és similar a LTMemory, tret que el temps d'execució d'una assignació d'una àrea **`VTMemory`** no ha de finalitzar en temps lineal.

Memòria d'emmagatzematge dinàmic
Els objectes de la memòria d'emmagatzematge dinàmic no poden fer referència a res de la memòria amb àmbit. Si un cap d'un objecte de la memòria d'emmagatzematge dinàmic s'assigna a un objecte de la memòria amb àmbit, es genera una excepció IllegalAssignmentError.

Sincronització i compartició de recursos

En un sistema en temps real, quan hi ha tres fils o més que s'executen a prioritats diferents i que se sincronitzen entre ells, pot ser que es produeixi una condició anomenada inversió de prioritat, en la qual l'execució d'un fil de prioritat més alta queda blocada per un fil de prioritat més baixa durant un període llarg de temps. Per tal d'evitar aquesta condició, el WebSphere Real Time for RT Linux utilitza un esquema anomenat herència de prioritat.

Quan l'execució d'una tasca de prioritat més alta queda blocada per una tasca de prioritat més baixa, la prioritat de la tasca de prioritat més baixa augmenta temporalment per passar a ser equivalent a la prioritat més alta fins que la tasca de prioritat més alta deixa d'estar blocada.

Paràmetres periòdics i no periòdics

Els fils en temps real tenen diversos paràmetres de publicació, que determinen la freqüència de publicació d'un objecte planificable. Els paràmetres periòdics i no periòdics són un exemple d'aquests paràmetres de publicació.

Paràmetres periòdics

Aquesta classe és per als objectes planificables que es publiquen de manera periòdica.

AbsoluteTime
S'expressa en mil•lisegons i nanosegons.

RelativeTime
És la longitud de temps d'un esdeveniment determinat expressada en mil•lisegons i nanosegons. Per exemple, podeu mesurar el temps absolut en què un esdeveniment s'inicia i finalitza. Després podeu calcular el temps relatiu com a la diferència entre les dues mesures.

Paràmetres no periòdics

Aquesta classe s'utilitza per als objectes planificables que es publiquen amb una freqüència irregular. Atès que pot ser que es produeixi un esdeveniment no periòdic abans que no finalitzi el primer, podeu definir la longitud de la cua de sol•licituds pendents.

Gestió d'esdeveniments asíncrons

Els gestors d'esdeveniments asíncrons reaccionen als esdeveniments que es produeixen fora d'un fil; per exemple, l'entrada d'una interfície d'una aplicació. En sistemes en temps real aquests esdeveniments responen entre els límits establerts per a l'aplicació.

Els esdeveniments asíncrons es poden associar amb interrupcions del sistema i senyals POSIX, i també es poden enllaçar a un temporitzador.

Com passa amb els fils en temps real, els gestors d'esdeveniments asíncrons tenen diversos paràmetres associats. Per obtenir una llista d'aquests paràmetres, vegeu “Objectes planificables i els seus paràmetres” a la pàgina 9.

Gestors de senyals

El gestor POSIXSignalHandler admet els senyals SIGQUIT, SIGTERM i SIGABRT. El comportament per defecte per a SIGQUIT fa que es generi un Javadump. La generació d'un Javadump no interfereix amb el funcionament del programa en execució, a banda del temps de la CPU i de la lectura i escriptura de fitxers. La generació d'un Javadump interromp el programa fins que finalitza el Javadump; el rendiment de l'aplicació no es podrà predir mentre es generin Javadumps.

Per suprimir tota la generació de nuclis i Javadump en un error, utilitzeu **-Xdump:none**.

Per suprimir només la generació de l'abocament del sistema i del Javadump en un senyal SIGQUIT, especifiqueu **-Xdump:java:none -Xdump:java:events=gpf+abort**.

Els senyals següents es poden adjuntar als gestors d'esdeveniments asíncrons (AEH) mitjançant el mecanisme POSIXSignalHandler (les descripcions de senyals segons es defineix a /usr/include/bits/signum.h):

```
#define SIGQUIT         3       /* Sortir (POSIX).  */
#define SIGABRT         6       /* Avortar (ANSI).  */
#define SIGKILL         9       /* Matar, no blocable (POSIX).  */
```

Actualment no s'admet cap altre senyal. Tots els senyals llistats anteriorment són asíncrons, i no és possible admetre l'adjunció de senyals síncrons (com ara SIGILL i SIGSEGV) perquè indiquen un error de l'aplicació o del codi de la JVM, no un esdeveniment generat externament.

Nota: Per defecte, SIGQUIT fa que l'aplicació Java generi abocaments de memòria (per exemple, un Javadump) quan el rep la JVM. Tot i que, a més a més, es lliura a qualsevol AEH adjunt, aquest lliurament pot provocar un comportament confús o no desitjable i podeu inhabilitar-lo utilitzant l'opció **-Xdump:none:events=user** a la línia d'ordres Java.

Documentació necessària

El WebSphere Real Time for RT Linux implementa l'especificació Especificació de temps real per a Java (RTSJ).

El WebSphere Real Time for RT Linux versió 2.0 s'ha certificat com a compatible amb RTSJ 1.0.2 amb el kit RTSJ Technology Compatibility Kit versió 3.0.13 FCS i és compatible amb el kit de compatibilitat Java (JCK) per a la versió 6.0.

Recursos admesos

S'admeten els següents recursos:

- Aplicació de la freqüència d'assignació en l'assignació d'emmagatzematge dinàmic per limitar la freqüència en què un objecte planificable crea objectes a l'emmagatzematge dinàmic.

Recursos no admesos

Tot seguit s'indiquen els recursos no admesos:

- Protocol d'emulació de sostre de prioritat. Per exemple, no permet utilitzar PriorityCeilingEmulation com a política de control de supervisió.
- Suport d'accés atòmic, tret quan és necessari per al compliment de l'especificació.
- No hi ha cap planificador disponible per a les aplicacions, tret del planificador de prioritat bàsic.
- Aplicació de costos.

Documentació necessària per a l'especificació Especificació de temps real per a Java

En aquesta secció es fa esment de la secció sobre *documentació necessària* de l'especificació RTSJ (Especificació de temps real per a Java). S'indiquen les desviacions de la implementació estàndard de l'RTSJ.

1. **El valor per defecte és l'algorisme de proves de viabilitat.**

"Si l'algorisme de proves de viabilitat no és el valor per defecte, documenteu l'algorisme de proves de viabilitat."

2. **Només hi ha disponible per a les aplicacions el planificador de prioritats base.**

   "Si hi ha planificadors que no són el planificador de prioritats base disponibles per a les aplicacions, documenteu el comportament del planificador i la seva interacció amb altres planificadors, com s'indica al capítol dedicat a la planificació. Documenteu també la llista de classes que conformen els objectes planificables per al planificador, tret que aquesta llista sigui la mateixa que la llista d'objectes planificables per al planificador base."

3. **Un objecte planificables que es reemplaça per un objecte planificable de prioritat superior es col•locarà al davant de la cua segons la seva prioritat.**

   "Un objecte planificable que es reemplaça per un objecte planificable de prioritat superior es col•loca a la cua segons la seva prioritat activa, a la posició determinada per la implementació. Si l'objecte planificable reemplaçat no es col•loca al davant de la cua adequada, la implementació ha de documentar l'algorisme utilitzat per a aquesta ubicació. La ubicació al principi de la cua pot ser necessària en una versió futura d'aquesta especificació."

4. **No s'admet l'aplicació de costos.**

   "Si la implementació admet l'aplicació de costos, la implementació ha de documentar la granularitat segons la qual s'actualitza el consum actual de la CPU. "

5. **S'admet l'assignació seqüencial simple.**

   "L'assignació de memòria implementada per un filtre de tipus de memòria física s'ha de documentar tret que sigui una assignació seqüencial simple de bytes contigus."

6. **El WebSphere Real Time for RT Linux no proporciona subclasses per al recollidor de deixalles Metronome.**

   "La implementació ha de documentar completament el comportament de les subclasses del recollidor de deixalles."

7. **El WebSphere Real Time for RT Linux no proporciona cap subclasse MonitorControl.**

   "Una implementació que proporciona subclasses MonitorControl no detallades en aquesta especificació ha de documentar els seus efectes, especialment pel que fa al control d'inversió de prioritats i quins planificadors (si n'hi ha algun) no admeten la nova política."

8. **La prioritat d'un objecte planificable que conté un supervisor necessari per a un objecte planificable de prioritat més alta s'incrementa a la prioritat més alta fins que arriba el moment en què s'allibera el supervisor. Si, en aquest punt, l'objecte planificable s'ha de definir com a objecte que ja no es pot executar (és a dir que hi ha una fina de prioritat superior que cal dur a terme), es col•locarà a la part final de la cua per a la seva prioritat original (no incrementada) quan s'executi en kernels anteriors a SUSE Linux Enterprise Real Time 10 SP2, actualització de kernel versió 2.6.22.19-0.16, i Red Hat Enterprise Linux 5.1 MRG 2.6.24.7-73 Errata 1. Els kernels d'aquests nivells o de nivells superiors col•loquen l'objecte planificable al davant de la cua.**

   "Si, en perdre la prioritat "incrementada" a causa d'un algorisme per evitar la inversió de prioritats, l'objecte planificable no es col•loca al davant de la seva nova cua, la implementació ha de documentar el comportament de la cua."

9. **El planificador base és l'únic planificador que es proporciona amb el WebSphere Real Time for RT Linux.**

"Per a qualsevol planificador disponible que no sigui el planificador base, una implementació ha de documentar les diferències, si n'hi ha, entre la semàntica de sincronització i les regles definides per a la instància PriorityInheritance per defecte. Ha de subministrar documentació per al comportament del nou planificador amb herència de prioritat (i, si s'admet, el protocol d'emulació del sostre de la prioritat) equivalent a la semàntica per al planificador de prioritats base que s'indica al capítol dedicat a la sincronització."

10. **El pitjor cas de temps entre l'activació d'un esdeveniment i la planificació d'un gestor d'esdeveniments limitat associat serà, de mitjana, de 40µs i no sobrepassarà els 100µs sempre que no hi hagi objectes planificables competidors o una activitat del sistema de la mateixa prioritat o prioritat superior, i sempre que la recollida de deixalles no interfereixi. Si l'objecte planificable que dirigeix el mètode d'activació, l'objecte AsyncEvent o el gestor fan referència a l'emmagatzematge dinàmic, la possible influència de la recollida de deixalles es documenta a (A). Es dóna per fet que el codi s'interpreta i que a l'esdeveniment només hi ha un gestor (que està limitat) configurat.**

    "El pitjor interval de resposta entre l'activació d'un esdeveniment asíncron a causa d'un límit i l'alliberament d'un gestor AsyncEventHandler associat (assumint que no hi ha cap objecte executable de prioritat superior)s'ha de documentar per a l'arquitectura de referència."

11. **El pitjor cas de temps entre l'activació d'una excepció AsynchronouslyInterruptedException en un fil habilitat per ATC i el primer lliurament d'aquesta excepció serà, de mitjana, de 35µs i no sobrepassarà els 160µs sempre que no hi hagi objectes planificables competidors o una activitat del sistema de la mateixa prioritat o prioritat superior, i sempre que la recollida de deixalles no interfereixi. L'habilitació d'ATC en aquest cas vol dir que el fil s'executa en un mètode habilitat per AI en una regió que no té retards d'ATC i les condicions de la qual segueixen essent certes fins al lliurament de l'excepció. La possible influència de la recollida de deixalles es documenta a (A). Si el fil de destinació és un codi natiu, el retard pot ser que sigui il•limitat. Aquí es dóna per fet que el codi s'interpreta.**

    "L'interval entre l'activació d'una excepció AsynchronouslyInterruptedException en un fil habilitat per a ATC i el primer lliurament d'aquesta excepció (assumint que no hi ha cap objecte executable amb prioritat més alta) s'ha de documentar per a l'arquitectura de referència."

12. **No aplicable; vegeu la resposta 4.**

    "Si s'admet l'aplicació de costos, i la implementació assigna el cost d'executar finalitzadors per a objectes de la memòria amb àmbit a qualsevol objecte que no sigui l'objecte que ha provocat que el recompte de referències de l'àmbit caigui a zero en abandonar l'àmbit, les regles per assignar el cost s'han de documentar."

13. **No hi ha canvis a la implementació estàndard de RealtimeSecurity.**

    "Si la implementació de RealtimeSecurity és més restrictiva que la implementació necessària, o té opcions de configuració en temps d'execució, aquestes característiques s'han de documentar."

14. **Els finalitzadors per a objectes d'una àrea de memòria amb àmbit s'executaran al darrer fil per fer referència a aquesta àrea, és a dir, s'executaran quan el fil redueixi el recompte de referències d'u a zero. Els costos associats amb l'execució dels finalitzadors s'assignaran a aquest fil.**

    "Una implementació pot executar finalitzadors per a objectes de la memòria amb àmbit abans de tornar a entrar a l'àmbit i abans de tornar de qualsevol

crida a getReferenceCount() per a aquest àmbit. Això no obstant, s'ha de documentar quan executa aquests finalitzadors."

15. **La resolució no es pot definir.**

    "Per a cada rellotge admès, la documentació ha d'especificar si la resolució es pot definir i, en cas afirmatiu, la documentació ha d'indicar els valors admesos. "

16. **El WebSphere Real Time for RT Linux no proporciona cap altre rellotge que no sigui el rellotge en temps real.**

    "Si una implementació inclou rellotges que no siguin el rellotge en temps real necessari, la seva documentació ha d'indicar en quins contextos es poden utilitzar aquests rellotges."

Nota:

A L'arquitectura de referència per a les proves serà un LS20, de quatre processadors, a 2 GHz amb 1 MB de memòria cau i 4 GB de memòria.

B La recollida de deixalles pot provocar un retard en qualsevol punt d'un fil que estigui associat a l'emmagatzematge dinàmic. El recollidor pot funcionar en un dels dos modes bàsics que controlen el comportament quan s'exhaureix la memòria de l'emmagatzematge dinàmic. Si el recollidor està configurat per generar immediatament un error OutOfMemoryError en aquestes circumstàncies, el pitjor retard de recollida de deixalles serà normalment inferior a 1 ms. Actualment, en alguns casos, el retard pot ser superior; per exemple, si hi ha molts fils amb piles imbricades a nivells molt interns o si hi ha molts àmbits de mida considerable. Si el recollidor està configurat per dur a terme una recollida de deixalles síncrona abans de generar un error OutOfMemoryError, el possible retard de la recollida està relacionat amb el nombre d'objectes actius de l'emmagatzematge dinàmic i els nombres d'objectes d'altres àrees de memòria. En aquests casos, es considera que el retard no és il•limitat perquè pot ser molts segons per a mides típiques d'emmagatzematge dinàmic.

Capítol 3. Planificació

Llegiu aquest apartat abans d'instal•lar l'WebSphere Real Time for RT Linux.

- “Migració”
-
- “Prerequisits de maquinari i programari”
- “Consideracions” a la pàgina 24

Migració

L'WebSphere Real Time for RT Linux s'executa en un entorn Linux modificat per a les aplicacions en temps real. Podeu utilitzar aplicacions Java estàndard en un entorn en temps real. De manera alternativa, podeu modificar les vostres aplicacions per explotar les característiques del WebSphere Real Time.

Migració del sistema

Seguiu les instruccions que proporciona l'equip de suport del Linux.

Prerequisits de maquinari i programari

Utilitzeu aquesta llista per comprovar el maquinari, el sistema operatiu i l'entorn Java admès per al WebSphere Real Time for RT Linux.

Maquinari

Les configuracions de maquinari certificades del WebSphere Real Time for RT Linux són variants de multiprocessador dels sistemes següents:

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

Per seguir mantenint la certificació per al WebSphere Real Time for RT Linux, els sistemes IBM amb tecnologia HT (Hyper-Threading) no poden tenir aquesta tecnologia habilitada.

A més, el WebSphere Real Time for RT Linux s'admet en maquinari que executa un sistema operatiu admès, i que té aquestes característiques:

- 512 MB com a mínim de memòria física.
- Com a mínim, processador Intel Pentium 4, AMD Opteron o Intel Atom.

Per a sistemes que no són configuracions de maquinari certificades, IBM no fa cap declaració de rendiment. Les consideracions sobre el rendiment per a configuracions de maquinari certificades des descriuen a Capítol 7, “Rendiment”, a la pàgina 93

En sistemes que tenen suport per a la tecnologia HT, assegureu-vos que aquesta tecnologia no està habilitada per tal d'evitar efectes de rendiment adversos quan utilitzeu el WebSphere Real Time for RT Linux.

Sistema operatiu

- Red Hat Enterprise Linux 5.3 MRG. Consulteu “Instal•lació d'un entorn Linux en temps real” a la pàgina 25.
- SUSE Linux Enterprise Real Time (SLERT) 10. Consulteu “Instal•lació d'un entorn Linux en temps real” a la pàgina 25.

Consideracions

Heu de tenir present diversos factors quan utilitzeu el WebSphere Real Time for RT Linux.

- Quan sigui possible, no executeu més d'una JVM en temps real al mateix sistema. Això s'explica perquè aleshores tindríeu diversos recollidors de deixalles. Cada JVM no coneix les àrees de memòria de l'altra. Un efecte és que els cicles de recollida de deixalles i els temps de pausa no es poden coordinar entre les JVM, cosa que significa que és possible que una JVM afecti negativament el rendiment de la recollida de deixalles d'una altra JVM. Si heu d'utilitzar diverses JVM, assegureu-vos que cada JVM estigui vinculada a un subconjunt específic de processadors utilitzant l'ordre **taskset**.
- No podeu utilitzar l'opció **-Xdebug** i l'opció **-Xnojit** amb codi que s'ha precompilat utilitzant el compilador anticipat. Això és així perquè **-Xdebug** compila codi de diferent manera que el compilador anticipat (AOT) i no s'admet.

  Per depurar el codi, utilitzeu codi interpretat o compilat mitjançant JIT.
- Si utilitzeu la interfície com.sun.tools.javac.Main per compilar el codi font Java que utilitza el paquet javax.realtime, heu d'assegurar-vos que `sdk/jre/lib/i386/realtime/jclSC170/realtime.jar` s'inclogui a la variable classpath. Un exemple comú d'aquest tipus de compilació és la compilació ant.
- El paquet JavaComm opcional es pot instal•lar al WebSphere Real Time for RT Linux i s'hi pot accedir des de la JVM en temps real i la JVM que no és en temps real. Per obtenir més informació sobre la instal•lació i configuració, vegeu http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/user/jcommchapter.html. La JVM en temps real del WRT admet l'API JavaComm per utilitzar-la amb fils Java normals. Això no obstant, no hi ha cap garantia quant al determinisme o el rendiment en temps real quan s'accedeix a dispositius externs amb JavaComm. Per tant, no utilitzeu JavaComm amb fils en temps real i fils en temps real que no són d'emmagatzematge dinàmic, o quan calgui un comportament en temps real.
- Les memòries cau compartides utilitzades en versions anteriors del WebSphere Real Time for RT Linux per emmagatzemar codi i classes precompilats no són compatibles amb les memòries cau utilitzades en aquesta versió del WebSphere Real Time for RT Linux. Heu de regenerar el contingut de les memòries cau anteriors.
- Quan s'utilitzen memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.
- L'ordre **ps** trunca els noms dels fils Java.

  L'ordre **ps** està limitada a 15 caràcters. Si definiu un nom de fil amb més de 15 caràcters, l'ordre **ps** trunca el nom.
- El WebSphere Real Time for RT Linux no admet l'autenticació NTLoginModule (NTLM).

  L'autenticació NTLoginModule (NTLM) s'utilitza per ajudar a autenticar l'accés a un servei Windows. L'autenticació mitjançant NTLM només s'admet a la plataforma Windows. Això vol dir que el WebSphere Real Time for RT Linux no admet l'autenticació NTLM.

Capítol 4. Instal•lació del WebSphere Real Time for RT Linux

Seguiu aquests passos per instal•lar el producte.

Fitxers d'instal•lació

Tot seguit s'indiquen els fitxers d'instal•lació necessaris.

L'IBM WebSphere Real Time for RT Linux es proporciona en dos tipus de paquets InstallAnywhere.

Paquets instal•lables

Els paquets instal•lables configuren el sistema. Per exemple, els programes poden definir variables d'entorn.

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

Paquets d'arxiu

Aquests paquets extreuen els fitxers al sistema, però no realitzen cap configuració.

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

Instal•lació d'un entorn Linux en temps real

Abans d'instal•lar el WebSphere Real Time for RT Linux, heu d'instal•lar el Linux en temps real.

Abans de començar

Abans d'instal•lar el WebSphere Real Time for RT Linux, heu d'instal•lar una versió de 64 bits del Linux en temps real.

Red Hat Enterprise Linux 5.3 MRG

- Per obtenir més informació sobre la instal•lació del component en temps real del Red Hat Enterprise Linux 5.3 MRG, vegeu les instruccions d'instal•lació per a RT-Linux RHEL 5.3 MRG 1.1.2: https://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.1/html/Realtime_Installation_Guide/index.html

SUSE Linux Enterprise Real Time 10

- Per obtenir més informació sobre la instal•lació de SUSE Linux Enterprise Real Time 10, vegeu: http://www.novell.com/products/realtime/eval.html

Quan s'utilitzen molts descriptors de fitxers per carregar diferents instàncies de classes, es pot veure un missatge d'error "`java.util.zip.ZipException: error in opening zip file`" o alguna forma d'IOException que avisi que no s'ha pogut obrir un fitxer. La solució és augmentar el subministrament de descriptors de fitxers mitjançant l'opció **`ulimit`**. Per trobar el límit actual de fitxers oberts, utilitzeu l'ordre:

```
ulimit -a
```

Per permetre més fitxers oberts, utilitzeu l'ordre:

```
ulimit -n 8196
```

Instal•lació des d'un fitxer InstallAnywhere

Aquests paquets proporcionen un programa interactiu que us orientarà a través de les opcions d'instal•lació. Podeu executar el programa com a interfície gràfica de l'usuari o des de la consola del sistema.

Abans de començar

El vostre sistema ha de tenir les biblioteques compartides següents:

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

Si no teniu la biblioteca compartida `libstdc++.so.5`, podeu veure un abocament de memòria del nucli de Java quan instal•leu, amb els errors següents:

```
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 esteu instal•lant un paquet instal•lable, heu de tenir l'eina rpm-build instal•lada al sistema, en cas contrari el programa d'instal•lació no pot registrar el paquet nou a la base de dades de RPM. Per saber si l'eina rpm-build està instal•lada, introduïu l'ordre següent:

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

Quant a aquesta tasca

Els paquets InstallAnywhere tenen una extensió de fitxer `.bin`.

Hi ha dos tipus de paquet:

Instal•lable
La instal•lació d'aquests paquets permet també configurar el sistema, per exemple definint variables d'entorn.

Arxiu La instal•lació d'aquests paquets extrau els fitxers al sistema, però no realitza cap configuració.

Procediment

- Per instal•lar el paquet de forma interactiva, completeu una instal•lació assistida.

- Per instal•lar el paquet sense cap interacció addicional de l'usuari, completeu una instal•lació desassistida. Podeu triar aquesta opció si voleu instal•lar diversos sistemes.
- Quan el procés d'instal•lació hagi finalitzat, seguiu els passos de configuració d'aquesta secció, com ara la definició de les variables d'entorn path i classpath.

Resultats

El producte està instal•lat.

Nota: No interrompeu el procés d'instal•lació, per exemple prement Ctrl+C. Si interrompeu el procés, possiblement haureu de tornar a instal•lar el producte. Per obtenir-ne més informació, consulteu "Instal•lació interrompuda" a la pàgina 29.

Si esteu utilitzant un paquet instal•lable, possiblement veureu missatges que us advertiran que s'ha trobat un problema. La instal•lació de paquets d'arxiu no produeix cap missatge. Alguns dels missatges que podríeu veure quan s'utilitza un paquet instal•lable es mostren a la llista següent:

The installer cannot run on your configuration. It will now quit.
: Aquest missatge d'error es produeix quan l'ID d'usuari no té autorització per executar el procés d'instal•lació. Com que no pot continuar, finalitza el programa d'instal•lació. Per arreglar el problema, inicieu la instal•lació novament amb un ID d'usuari que tingui autoritat root.

An RPM package is already installed. Uninstall the package before proceeding.
: Aquest missatge indica que ja hi ha un paquet RPM instal•lat. Com que no pot continuar, finalitza el programa d'instal•lació. Per arreglar el problema, desinstal•leu el paquet RPM abans de continuar.

Completar una instal•lació assistida

Instal•lació del producte des d'un paquet InstallAnywhere, de forma interactiva.

Abans de començar

Comproveu les condicions següents abans de començar amb el procés d'instal•lació:

- Si heu instal•lat prèviament el WebSphere Real Time for RT Linux des d'un paquet RPM, heu de desinstal•lar aquest paquet abans de continuar.
- Heu de tenir un ID d'usuari amb autoritat root.

Procediment

1. Descarregueu el fitxer del paquet d'instal•lació en un directori temporal.
2. Canvieu al directori temporal.
3. Inicieu el procés d'instal•lació escrivint `./`*`paquet`* en un indicador del shell, on *paquet* és el nom del paquet que esteu instal•lant.
4. Seleccioneu un idioma de la llista que es mostra a la finestra de l'instal•lador i posteriorment feu clic a **Next**. La llista d'idiomes disponibles es basa en la configuració local del sistema.
5. Llegiu l'acord de llicència, utilitzant la barra de desplaçament per arribar al final del text de la llicència. Per continuar amb la instal•lació heu d'acceptar els termes de l'acord de llicència. Per acceptar els termes, seleccioneu el botó d'opció i posteriorment feu clic a **OK**.

Nota: No podeu seleccionar el botó d'opció per acceptar l'acord de llicència fins que hàgiu llegit fins el final el text de la llicència.

6. Se us sol•licitarà que escolliu el directori de destinació per a la instal•lació. Si no voleu fer la instal•lació en el directori de destinació, feu clic a **Choose** per seleccionar un directori alternatiu, utilitzant la finestra del navegador. Quan hàgiu escollit el directori d'instal•lació, feu clic a **Next** per continuar.
7. Se us sol•licitarà que reviseu les opcions que heu triat. Per canviar la selecció, feu clic a **Previous**. Si les opcions triades són correctes, feu clic a **Install** per continuar amb la instal•lació.
8. Quan el procés d'instal•lació hagi finalitzat, feu clic a **Done** per finalitzar.

Completar una instal•lació desassistida

Si heu d'instal•lar més d'un sistema i ja sabeu les opcions d'instal•lació que voleu utilitzar, és possible que vulgueu utilitzar el procés d'instal•lació desassistida. La instal•lació es fa una vegada utilitzant el procés d'instal•lació assistida, i posteriorment s'utilitza el fitxer de resposta per completar més instal•lacions sense cap interacció addicional de l'usuari.

Procediment

1. Creeu un fitxer de resposta completant una instal•lació desassistida. Utilitzeu una de les opcions següents:
   - Utilitzeu la GUI i especifiqueu que el programa d'instal•lació creï un fitxer de resposta. El fitxer de resposta s'anomena `installer.properties` i es crea al directori d'instal•lació.
   - Utilitzeu la línia d'ordres i afegiu l'opció -r a l'ordre d'instal•lació assistida, especificant el camí d'accés complet al fitxer de resposta. Per exemple:

     `./`*`paquet`* `-r /`*`camí_accés`*`/installer.properties`

   Contingut del fitxer de resposta d'exemple:

   ```
   INSTALLER_UI=silent
   USER_INSTALL_DIR=/directori_personal
   ```

   En aquest exemple, */directori_personal* és el directori d'instal•lació per defecte que escolliu per a la instal•lació.
2. Opcional: Si cal, editeu el fitxer de resposta per canviar les opcions.

   Nota: Els paquets d'arxiu tenen el següent problema conegut: les instal•lacions que utilitzen el fitxer de resposta utilitzen el directori per defecte fins i tot si canvieu el directori al fitxer de resposta. Si hi ha una instal•lació prèvia al directori per defecte, se sobreescriu.

   Si esteu creant més d'un fitxer de resposta, cadascun amb opcions d'instal•lació diferents, especifiqueu un nom únic per a cada fitxer de resposta, en el format *`fitxer_personal`*`.properties`.
3. Opcional: Genereu un fitxer de registre. Com que esteu fent una instal•lació silenciosament, no es visualitza cap missatge d'estat al final del procés d'instal•lació. Per generar un fitxer de registre que contingui l'estat de la instal•lació, completeu els passos següents:
   a. Definiu les propietat del sistema necessàries utilitzant l'ordre següent.

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

   b. Definiu la variable d'entorn següent per enviar la sortida del registre a la consola.

      ```
      export LAX_DEBUG=1
      ```

4. Inicieu una instal•lació desassistida executant l'instal•lador del paquet amb l'opció silenciosa **-i**, i l'opció **-f** per especificar el fitxer de resposta. Per exemple:

   ```
   ./paquet -i silent -f /camí_accés/installer.properties 1>console.txt 2>&1
   ./paquet -i silent -f /camí_accés/fitxer_personal.properties 1>console.txt 2>&1
   ```

   Podeu utilitzar un camí d'accés complet o un camí d'accés relatiu al fitxer de propietats. En aquests exemples, la cadena `1>console.txt 2>&1` redirigeix la informació del procés d'instal•lació de les cadenes stderr i stdout al fitxer de registre `console.txt` del directori actual. Reviseu aquest fitxer de registre si creieu que hi ha hagut algun problema amb la instal•lació.

   Nota: Si el directori d'instal•lació conté diversos fitxers de resposta, s'utilitza el fitxer de resposta per defecte, `installer.properties`.

Instal•lació interrompuda

Si l'instal•lador del paquet s'atura inesperadament durant la instal•lació, per exemple si premeu Ctrl+C, la instal•lació es malmetrà i no podreu desinstal•lar o tornar a instal•lar el producte. Si intenteu desinstal•lar o tornar a instal•lar, veureu un missatge de tipus `Error fatal de l'aplicació`.

Quant a aquesta tasca

Per solucionar aquest problema, suprimiu fitxers i torneu a instal•lar, com es descriu als passos següents:

Procediment

1. Suprimiu el fitxer de registre `/var/.com.zerog.registry.xml`.
2. Suprimiu el directori que conté la instal•lació, si s'ha creat. Per exemple, opt/IBM/javawrt3/.
3. Executeu el procés d'instal•lació una altra vegada.

Limitacions i problemes coneguts

Els paquets InstallAnywhere tenen algunes limitacions i problemes coneguts.

- Si no teniu la biblioteca compartida `libstdc++.so.5` al vostre sistema, la instal•lació falla, provocant un abocament de memòria del nucli de Java. Per obtenir més informació, consulteu “Instal•lació des d'un fitxer InstallAnywhere” a la pàgina 26.
- La GUI del paquet d'instal•lació no suporta el programa de lectura de pantalla Orca. Podeu utilitzar el mode d'instal•lació desassistida com a alternativa a la GUI.
- Si, després de la instal•lació, introduïu *./paquet* per iniciar el programa novament, el programa visualitza el missatge següent:

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

  Si premeu l'Intro per acceptar el valor per defecte, el programa no respon. Escriviu un número i després premeu l'Intro.
- Si instal•leu el paquet i posteriorment intenteu instal•lar de nou en un mode diferent, per exemple en mode de consola o en mode silenciós, possiblement veureu el missatge d'error següent:

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

No hauríeu de veure aquest missatge si heu instal•lat utilitzant el mode de GUI o esteu executant el programa d'instal•lació novament en mode de consola. Si veieu aquest error i esteu executant el programa per seleccionar l'opció de desinstal•lació (només els paquets instal•lables), utilitzeu com a alternativa l'ordre **./_uninstall/uninstall**, com es descriu a "Desinstal•lació del WebSphere Real Time for RT Linux" a la pàgina 32.

Només els paquets instal•lables

- No podeu actualitzar una instal•lació existent utilitzant els paquets InstallAnywhere. Per actualitzar el WebSphere Real Time for RT Linux, primer heu de desinstal•lar qualsevol versió anterior.
- No podeu instal•lar dues instàncies diferents de la mateixa versió del WebSphere Real Time for RT Linux al mateix sistema a la vegada, fins i tot si utilitzeu directoris d'instal•lació diferents. Per exemple, no podeu tenir simultàniament el WebSphere Real Time for RT Linux V3 al directori `/previous` i una instal•lació de renovació de servei del WebSphere Real Time for RT Linux al directori `/current`. El programa d'instal•lació comprova el número de versió. Si el programa troba un paquet existent amb el mateix número de versió, se us sol•licitarà que desinstal•leu el paquet existent.
- Si s'instal•la el paquet i executeu l'instal•lador del paquet utilitzant la GUI, podeu seleccionar la desinstal•lació del paquet. Aquesta opció de desinstal•lació no està disponible en mode desassistit. Si executeu novament l'instal•lador del paquet en mode desassistit, el programa s'executa però no realitza cap acció.

Només els paquets d'arxiu

- Si canvieu el directori d'instal•lació en un fitxer de resposta i posteriorment executeu una instal•lació desassistida utilitzant aquest fitxer de resposta, el programa d'instal•lació ignora el directori d'instal•lació nou i utilitza com a alternativa el directori per defecte. Si hi ha una instal•lació prèvia al directori per defecte, se sobreescriu.

Definició del camí d'accés

Quan definiu la variable d'entorn **PATH**, podeu executar una aplicació o un programa escrivint-ne el nom en un indicador de l'intèrpret d'ordres.

Quant a aquesta tasca

Nota: Si modifiqueu la variable d'entorn **PATH** com es descriu en aquesta secció, sobreescriureu els executables Java que tingueu al camí d'accés.

Podeu especificar el camí d'accés a una eina escrivint cada vegada el camí d'accés abans del nom de l'eina. Per exemple, si l'SDK està instal•lat a `opt/IBM/javawrt3/`, podeu compilar un fitxer anomenat *myfile.java* escrivint el següent a l'indicador de l'intèrpret d'ordres:

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

Per evitar haver d'escriure el camí d'accés complet cada vegada:

1. Editeu el fitxer d'inici de l'intèrpret d'ordres al directori inicial (normalment **.bashrc**, en funció de l'intèrpret d'ordres) i afegiu els camins d'accés absoluts a la variable d'entorn **PATH**; per exemple:

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

2. Torneu a iniciar la sessió o executeu l'script de l'intèrpret d'ordres actualitzat per activar la nova definició del **PATH**.
3. Compileu el fitxer amb l'eina **javac**. Per exemple, per compilar el fitxer *myfile.java*, introduïu el següent a l'indicador de l'intèrpret d'ordres:

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

   la variable d'entorn **PATH** permet al sistema operatiu Linux trobar fitxers executables, com ara **javac**, **java** i l'eina **javadoc**, des de qualsevol directori actual. Per visualitzar el valor actual del camí d'accés, escriviu el següent en un indicador d'ordres:

   ```
   echo $PATH
   ```

Què cal fer posteriorment

Vegeu "Definició de la variable classpath" per determinar si heu de definir la variable d'entorn CLASSPATH.

Definició de la variable classpath

La variable d'entorn **CLASSPATH** indica a les eines de l'SDK, com ara **java**, **javac**i **javadoc** on trobar les biblioteques de classes Java.

Quant a aquesta tasca

Definiu la variable d'entorn **CLASSPATH** de manera explícita només en una d'aquestes circumstàncies:

- Necessiteu una biblioteca o un fitxer de classes diferent, com ara un que desenvolupeu, i no es troba al directori actual.
- Canvieu la ubicació dels directoris `bin` i `lib` i aquests ja no tenen el mateix directori pare.
- Teniu la intenció de desenvolupar o executar aplicacions mitjançant la utilització de diferents entorns de temps d'execució en el mateix sistema.

Per visualitzar el valor actual de la variable **CLASSPATH**, introduïu en un indicador de l'intèrpret d'ordres:

```
echo $CLASSPATH
```

Si desenvolupeu i executeu aplicacions que utilitzen diferents entorns de temps d'execució, incloses altres versions que heu instal•lat separadament, heu d'establir **CLASSPATH** i **PATH** explícitament per a cada aplicació. Si executeu diverses aplicacions de manera simultània i utilitzeu entorns d'execució diferents, cada aplicació s'ha d'executar al seu propi intèrpret d'ordres.

Si només executeu una versió del Java alhora, podeu utilitzar un script de l'intèrpret d'ordres pe commutar entre els diferents entorns d'execució.

Què cal fer posteriorment

Vegeu "Prova de la instal•lació" per verificar que la instal•lació ha estat satisfactòria.

Prova de la instal•lació

Utilitzeu l'opció **-version** per verificar si la instal•lació ha estat satisfactòria.

Quant a aquesta tasca

La instal•lació Java consta d'una JVM estàndard i una JVM en temps real.

Procediment

Seguiu aquests passos per provar la instal•lació:

1. Per veure informació sobre la versió per a la JVM estàndard, escriviu l'ordre següent a l'indicador de l'intèrpret d'ordres:

   ```
   java -version
   ```

   Aquesta ordre retorna els missatges següents si és satisfactòria:

   ```
   java version "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 voleu utilitzar la JVM estàndard i no la JVM en temps real, consulteu les publicacions IBM User Guides for Java v7 on Linux.

   Nota: La informació sobre la versió és correcta, però pot ser que les dates siguin posteriors a les de l'exemple. El format de la sèrie de data és: `yyyymmdd` seguit probablement per informació addicional específica del component.

2. Per veure informació sobre la versió per a la JVM en temps real, escriviu l'ordre següent a l'indicador de l'intèrpret d'ordres:

   ```
   java -Xrealtime -version
   ```

   Aquesta ordre retorna els missatges següents si és satisfactòria:

   ```
   java version "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 informació sobre la versió és correcta, però pot ser que l'arquitectura de plataforma i les dates siguin diferents que les de l'exemple. El format de la sèrie de data és: `yyyymmdd` seguit probablement per informació addicional específica del component.

Desinstal•lació del WebSphere Real Time for RT Linux

El procés que utilitzeu per eliminar el WebSphere Real Time for RT Linux depèn del tipus d'instal•lació que utilitzeu.

Abans de començar

Per als paquets instal•lables InstallAnywhere, heu de tenir un ID d'usuari amb autoritat root.

Quant a aquesta tasca

No hi ha cap procés de desinstal•lació per als paquets d'arxiu InstallAnywhere. Per eliminar un paquet d'arxiu del sistema, suprimiu el directori de destinació que escolliu quan instal•leu el paquet. Per als paquets instal•lables InstallAnywhere, desinstal•leu el producte utilitzant una ordre o executant de nou el programa d'instal•lació, com es descriu als passos següents.

Procediment

- Opcional: Desinstal•leu manualment utilitzant l'ordre **uninstall**.
  1. Canvieu al directori que conté la instal•lació. Per exemple:
     ```
     cd /opt/IBM/javawrt3
     ```
  2. Inicieu el procés de desinstal•lació introduint l'ordre següent:
     ```
     ./_uninstall/uninstall
     ```
- Opcional: Si no podeu localitzar fàcilment el programa de desinstal•lació, com a alternativa podeu executar un altra instal•lació assistida. El programa d'instal•lació detecta que el producte ja està instal•lat, i posteriorment us dóna l'oportunitat de desinstal•lar la instal•lació anterior.

Capítol 5. Execució d'aplicacions de l'IBM WebSphere Real Time for RT Linux

Informació important d'ajuda quan executeu aplicacions en temps real.

Planificació i distribució de fils

El sistema operatiu Linux admet diverses polítiques de planificació. La política de planificació en temps compartit universal per defecte és SCHED_OTHER, que s'utilitza en la majoria de fils. Els fils de les aplicacions en temps real poden utilitzar SCHED_RR i SCHED_FIFO. només utilitza SCHED_OTHER i SCHED_RR.

El kernel decideix quin és el proper fil executable que el processador ha d'executar. El kernel manté una llista dels fils executables. Busca el fil que té la prioritat més alta i selecciona el fil com a proper fil que cal executar.

Amb l'ordre següent es pot obtenir una llista de les prioritats i les polítiques de fils:

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

on la política:

- `TS` és SCHED_OTHER
- `RR` és SCHED_RR
- `FF` és SCHED_FIFO
- `-` no té cap política notificada

La sortida és com la d'aquest exemple:

```
  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
```

Aquesta sortida mostra el procés Java, la política de planificació vigent, el fil principal amb la prioritat "-" (altra) i algun fils en temps real amb prioritats de la 11 a la 89.

Per consultar la política de planificació actual, utilitzeu **`sched_getscheduler`** o l'ordre **`ps`** que es mostra a l'exemple.

Per obtenir informació sobre els processos, vegeu “Tècniques de depuració generals” a la pàgina 98.

Prioritats i polítiques dels fils Java en temps real

Els fils en temps real, és a dir, els fils assignats com a `java.realtime.RealtimeThread`, i els gestors d'esdeveniments asíncrons utilitzen la política de planificació SCHED_FIFO.

La planificació i distribució de fils Java en temps real forma part de l'especificació en temps real per a Java (RTSJ). Aquest tema, així com també la gestió de polítiques de planificació i de prioritats de fils Java en temps real es descriu a la secció “Suport per al RTSJ” a la pàgina 9.

Ús del codi compilat amb l' WebSphere Real Time for RT Linux

L'IBM WebSphere Real Time for RT Linux admet diversos models de compilació de codi, i proporciona diferents nivells de rendiment i determinisme de codi.

Operació interpretada

Aquest és el model de compilació de codi més simple. L'intèrpret executa una aplicació Java, però no utilitza compilació de codi. L'intèrpret mostra un bon determinisme, però proporciona un rendiment molt baix; per tant, cal evitar l'ús d'aquest mode de funcionament per als sistemes de producció.

Per utilitzar l'operació interpretada, especifiqueu l'opció **`-Xint`** a la línia d'ordres Java.

Compilació a temps (JIT) de prioritat baixa

El model de compilació per defecte del WebSphere Real Time for RT Linux utilitza un compilador JIT per compilar els mètodes importants d'una aplicació Java mentre s'executa l'aplicació. En aquest mode, el compilador JIT funciona de manera semblant a l'operació del compilador JIT en una JVM que no és en temps real. La diferència rau en que el compilador JIT del WebSphere Real Time for RT Linux s'executa a una prioritat més baixa que els fils en temps real. La prioritat més baixa vol dir que el compilador JIT utilitza recursos del sistema quan l'aplicació no necessita dur a terme tasques en temps real. L'efecte és que el compilador JIT no afecta significativament el rendiment de les tasques en temps real.

El compilador JIT utilitza dos fils per a les activitats relacionades amb la compilació: el fil de compilació i el fil d'exemple. Aquests fils s'executen a una prioritat més baixa que les tasques en temps real. El fil de compilació s'executa de manera asíncrona a l'aplicació. Això vol dir que un fil d'aplicació no espera que el fil de compilació acabi de compilar un mètode en cap moment. El fil d'exemple envia periòdicament un missatge asíncron als fils d'aplicació per identificar el mètode que s'està executant a cada fil. El processament del missatge triga una mica al fil de l'aplicació. No s'envia cap missatge si el fil d'exemple no es pot executar a causa de les tasques en temps real de prioritat més alta. L'ús del compilador JIT té molt poca repercussió en el determinisme, però aquest mode de compilació proporciona el millor rendiment per a molts usuaris.

Per executar una aplicació amb el JIT a prioritat baixa, vegeu “Habilitació del JIT” a la pàgina 53.

Codi precompilat de manera anticipada (AOT)

El WebSphere Real Time for RT Linux compila mètodes Java en codi natiu

en un pas de compilació prèvia abans d'executar l'aplicació. Abans de WebSphere Real Time for RT Linux V2, el pas de compilació prèvia utilitza l'eina jxeinajar per compilar mètodes mitjançant el compilador AOT i emmagatzema els resultats en fitxers executables Java especials. Aquests fitxers es poden recopilar en fitxers jar vinculats. Quan s'executa una aplicació, els fitxers jar vinculats s'afegeixen al camí d'accés de classes de l'aplicació de manera que la JVM pugui carregar codi AOT quan les classes per als mètodes s'han carregat des de JXE. Si s'utilitza aquest enfocament, es pot utilitzar l'opció de la línia d'ordres **-Xnojit** per tal que el compilador JIT no estigui en absolut disponible. L'aplicació pot utilitzar qualsevol codi AOT precompilat que s'hagi creat i utilitzar l'intèrpret per a altres mètodes. Aquest mode de funcionament proporciona un alt determinisme, perquè el compilador JIT no està present i, per tant, no hi ha cap reducció de rendiment de fil d'exemple o commutador de context. La dificultat de compilar codi Java de manera anticipada, alhora que es compleix l'especificació Java, vol dir que el codi compilat AOT generalment té un rendiment més lent que el codi compilat JIT, tot i que normalment és més ràpid que la interpretació.

WebSphere Real Time for RT Linux V2 i les versions posteriors emmagatzema codi AOT en una memòria cau de classes compartida en lloc de fer-ho en fitxers JXE, utilitzant la tecnologia de classes compartides proporcionada a les JVM Java 6 d'IBM. L'eina admincache us permet consultar el contingut d'una la memòria cau, obtenir una llista de totes les memòries cau existents i emplenar una memòria cau amb classes i codi AOT. Els avantatges d'emmagatzemar codi compilat AOT són que els fitxers jar de l'aplicació no es modifiquen, i no canvia cap camí d'accés de classes necessàries en executar l'aplicació.

Una memòria cau de classes compartida té una mida límit pràctica, en base a l'espai d'adreces virtual disponible. Això vol dir que la compilació AOT per a tots els fitxers jar no és pràctica. Cal dur a terme una compilació AOT selectiva.

Quan una aplicació s'executa amb codi AOT en una memòria cau de classes compartida, el codi AOT per als mètodes d'una classe es carrega automàticament en carregar la classe a la JVM. El cost addicional per carregar una classe per instal•lar codi AOT per als seus mètodes fa que sigui important carregar prèviament tantes classes com sigui possible abans d'executar les parts importants de l'aplicació.

L'ús de codi precompilat AOT proporciona el nivell més alt de determinisme, amb un bon rendiment. El codi AOT es pot utilitzar quan s'executa l'aplicació especificant les opcions **-Xshareclasses** i **-Xaot**. L'opció **-Xaot** està activada per defecte.

Per emmagatzemar i utilitzar codi AOT amb una memòria cau de classes compartida utilitzant l'eina admincache, vegeu “Utilització de l'eina admincache” a la pàgina 39. La informació referent a la migració de jxeinajar a admincache es troba a la documentació de WebSphere Real Time for RT LinuxV2.

Per obtenir un exemple d'execució d'una aplicació amb codi compilat AOT, vegeu “Execució de l'aplicació de mostra mitjançant AOT” a la pàgina 86.

Mode combinat: combinació de codi precompilat AOT i compilació JIT de baixa prioritat

El codi compilat AOT i JIT es pot utilitzar conjuntament mentre s'executa l'aplicació. Aquest mode de funcionament pot proporcionar un

determinisme molt bo amb un bon rendiment, i un rendiment molt alt per a mètodes que s'executen sovint. L'avantatge principal d'aquest mode és que s'utilitza la precompilació AOT per garantir que les parts més importants de l'aplicació mai no s'executin a l'intèrpret, que normalment és molt més lent que el codi compilat AOT o JIT. No cal que precompileu tots els mètodes perquè el compilador JIT pot identificar dinàmicament els mètodes interpretats que s'executen sovint, sense que això afecti significativament el rendiment de l'aplicació. El mode combinat és el mode per defecte quan s'afegeix l'opció **-Xshareclasses** a la línia d'ordres.

Per executar l'aplicació amb la compilació combinada AT i JIT, vegeu "Execució de l'aplicació de mostra mitjançant AOT" a la pàgina 86

Gestió explícita de la compilació
En els modes de compilació que tenen el compilador JIT habilitat, es pot utilitzar l'API java.lang.Compiler per controlar de manera explícita el funcionament del compilador JIT. El compilador JIT compila mètodes de les classes que es passen mitjançant el mètode compileClass(). El mètode compileClass() és síncron i, per tant, no retorna fins que no s'han compilat els mètodes subministrats. Una aplicació pot utilitzar el mètode compileClass() en una fase d'inicialització, iterant es classes utilitzades per la fase principal d'execució de l'aplicació. Quan finalitzi la fase d'inicialització, crideu el mètode Compiler.disable() per inhabilitar del tot la compilació i els fils d'exemple. La dificultat principal d'aquesta tècnica rau en el problema de gestionar la llista de classes que cal carregar i compilar a la ase d'inicialització de l'aplicació, especialment durant el desenvolupament de l'aplicació.

Per obtenir informació sobre la gestió de la compilació en una aplicació, vegeu Eina d'anàlisi de classe en temps real d'IBM per a Java.

Visió general de les opcions de la línia d'ordres de compilació

Podeu executar una aplicació amb la compilació JIT habilitada utilitzant l'opció **-Xjit** o sense la compilació JIT, utilitzant l'opció **-Xnojit**. **-Xjit** és el mode per defecte.

Podeu executar una aplicació amb el codi AOT habilitat utilitzant les opcions **-Xshareclasses -Xaot**. Per inhabilitar el codi AOT, utilitzeu l'opció **-Xnoaot**. **-Xaot** és l'opció per defecte, però no té cap efecte tret que també s'especifiqui l'opció **-Xshareclasses**, perquè el codi AOT s'ha d'emmagatzemar en una memòria cau de classes compartida.

Utilització del compilador AOT

Seguiu aquests passos per precompilar el vostre codi Java. En aquest procediment es descriu l'ús de l'opció **-Xrealtime** en una ordre **javac**, l'eina **admincache** i les opcions **-Xrealtime** i **-Xnojit** amb l'ordre **java**.

Quant a aquesta tasca

Utilitzar el compilador anticipat vol dir que la compilació és independent del temps d'execució de l'aplicació. A més, podeu compilar més mètodes alhora, en lloc de compilar només els mètodes que s'utilitzen més sovint. Podeu compilar tot el que hi ha en una aplicació o només classes individuals, com es mostra als passos següents.

Nota: Quan utilitzeu memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.

Procediment

1. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent:

   ```
   javac -Xrealtime origen
   ```

   Aquesta ordre crea el codi de byte Java a partir de l'origen per utilitzar-lo a l'entorn en temps real. Consulteu Figura 2 a la pàgina 9.
2. Empaqueteu els fitxers de classe generats en un fitxer jar. Per exemple, creeu el fitxer test.jar:

   ```
   jar cvf test.jar source
   ```
3. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent:

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

   Aquesta ordre precompila el fitxer test.jar i escriu la sortida al directori de sortida `./aot`.
4. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent: Per executar el fitxer mitjançant el codi AOT a la memòria cau de classes compartida, escriviu el següent en un indicador de l'intèrpret d'ordres:

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

   Per executar el fitxer mitjançant el codi AOT a la memòria cau de classes compartida, torneu a compilar els mètodes que es criden sovint i, sense crear cap fitxer jar nou, introduïu el següent en un indicador de l'intèrpret d'ordres:

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

   Aquestes ordres utilitzen els mateixos fitxers jar que heu precompilat al pas 3.

Utilització de l'eina admincache

L'eina admincache s'utilitza per gestionar les memòries cau de classes compartides en una estació de treball.

Al producte IBM WebSphere Real Time for RT Linux, l'eina admincache es pot utilitzar per crear una memòria cau de classes compartida que contingui classes o classes i codi compilat AOT. Un cop creades les memòries cau, aquesta eina també es pot utilitzar per analitzar les memòries cau existents.

La memòria cau de classes compartida s'utilitza per reduir l'impacte de la memòria en escenaris de diverses JVM, i per accelerar l'inici de l'aplicació.

Les memòries cau de classes compartides es poden utilitzar al WebSphere Real Time for RT Linux en els modes de temps real i temps no real, però el format, la creació i les tècniques d'emplenament de la memòria cau varien. Les memòries cau en mode de temps real no són compatibles amb les memòries cau en mode de temps no real. En mode de temps no real, les memòries cau es creen i s'emplenen de la mateixa manera que la JVM estàndard. Això vol dir que la JVM crea i emplena la memòria cau de la mateixa manera que executa una aplicació, és adir, en un procés transparent per a l'usuari. En mode te temps real, mitjançant l'opció **-Xrealtime**, les memòries cau de classes compartides han de ser creades i emplenades per l'eina admincache, utilitzant l'opció **-populate**. Les aplicacions que s'executen en mode de temps real poden llegir contingut de la memòria cau emplenada prèviament, però no poden modificar-lo.

Les memòries cau de classes compartides creades en mode de temps real només es poden utilitzar quan s'executa una aplicació en mode de temps real. Les memòries cau de classes compartides creades en mode de temps no real només es poden utilitzar quan s'executa una aplicació en mode de temps no real. Això també és vàlid per a l'eina admincache. Per gestionar les memòries cau creades per la JVM en mode de temps real, utilitzeu l'eina admincache amb l'opció **-Xrealtime**. Per gestionar les memòries cau creades per la JVM en mode de temps no real, no utilitzeu l'opció **-Xrealtime**. Per connectar-vos a una memòria cau de classes compartida en temps d'execució, afegiu l'opció **-Xshareclasses** a la línia d'ordres.

Es poden crear diverses memòries cau de classes compartides en una estació de treball, cadascuna amb un nom específic i en un directori específic. Quan es crea una memòria cau nova, el nom de la memòria cau es pot especificar mitjançant l'opció **-cacheName** <*nom*>. El nom de la memòria cau no pot tenir més de 53 caràcters.

Per defecte, les memòries cau de classes compartides es creen al directori `/tmp/javasharedresources`, però aquesta ubicació es pot substituir utilitzant l'opció **-cacheDir** <*directori*>. El format intern per a una memòria cau de classes compartida depèn de les característiques de l'estació de treball en la qual s'ha creat. Això vol dir que, com a mesura de seguretat, no es poden crear memòries cau de classes compartides en unitats de xarxa. Un altre motiu per a aquesta restricció és els efectes de rendiment més lent i imprevisible quan s'accedeix a una memòria cau de classes compartida des d'un sistema de fitxers de xarxa.

Si no s'especifica cap nom de memòria cau a la línia d'ordres, el nom per defecte és **sharedcc_<inici_sessió_usuari>**

Per obtenir més informació sobre el funcionament de la memòria cau compartida en mode de temps no real, vegeu “Compartició de dades de classe entre les JVM per al mode en temps no real” a la pàgina 93.

Nota: Des de l'IBM WebSphere Real Time for RT Linux V2 SR1 i versios posteriors, heu d'utilitzar l'opció **-classpath** amb l'opció **-populate**.

Creació d'una memòria cau de classes compartida en temps real:

L'eina admincache s'utilitza per crear memòries cau de classes compartides a les quals es pot accedir en mode de temps real.

Nota: Cal tenir present les consideracions sobre la seguretat en crear fitxers de memòria cau de classe compartida amb valors per defecte. Vegeu “Consideracions sobre seguretat per a la memòria cau de classes compartida” a la pàgina 95 per obtenir més informació sobre les consideracions sobre seguretat per a la memòria cau de classes compartida i informació sobre la modificació dels permisos per defecte.

L'opció **-populate** de l'eina admincache s'utilitza per crear memòries cau de classes compartides. L'opció s'utilitza de manera combinada amb una llista de fitxers jar, o un directori, o un arbre de directoris on poder cercar fitxers jar. Per a cada fitxer jar especificat o trobat, l'eina admincache emmagatzema cada classe del fitxer jar a la memòria cau de classes compartida. Els mètodes de classe també estan compilats amb AOT i s'emmagatzemen a la memòria cau de classes compartida, tret que especifiqueu l'opció **-noaot**.

Heu d'utilitzar l'opció **-classpath** amb **-populate**, altrament veureu aquest missatge d'error:

```
-populate action requires -classpath <camí d'accés a classes> option to be
specified
```

L'opció **-help** de l'eina admincache mostra una llista de les subopcions que es poden utilitzar per controlar la manera com l'eina admincache emplena la memòria cau.

```
$ admincache -Xrealtime -help
Ús: admincache [opció]*
  on [opció] pot ser:
     -help | -?                Acció: mostra aquesta ajuda
     -Xrealtime                S'utilitza a l'entorn de temps real
     -cacheName <nom>          Especifica el nom de la memòria cau compartida (utilitzeu %%u per substituir el nom d'usua
     -cacheDir <dir>           Estableix la ubicació dels fitxers de la memòria cau de la JVM
     -listAllCaches            Acció: mostra una llista de les memòries cau compartides existents
     -printStats               Acció: imprimeix les estadístiques de la memòria cau
     -printAllStats            Acció: imprimeix més estadístiques detallades de la memòria cau
     -destroy                  Acció: destrueix la memòria cau indicada (o per defecte)
     -destroyAll               Acció: destrueix totes les memòries cau
     -populate                 Acció: crea una memòria cau nova i l'emplena
       -searchPath <camí accés>      Especifica el directori en el qual se cerquen fitxers si no se n'especifica cap (el
                                 Només es pot especificar una opció -searchPath
            -classpath <camí accés classes> especifica el camí d'accés de classes que s'utilitzarà en temps d'execució p
                                 L'opció -classpath és necessària
       -[no]recurse              [No] entrar en subdirectoris per cercar els fitxers que cal convertir
                                 (do not recurse per defecte)
       -[no]grow                 Si la memòria cau especificada ja existeix, [no] l'afegeix  (no grow per defecte)
                                 si no se selecciona -grow, la memòria cau especificada s'eliminarà si n'hi ha
       -verbose                  Imprimeix missatges de progrés per a cada jar
       -noisy                    Imprimeix missatges de progrés per a cada classe de cada fitxer jar
       -quiet                    Suprimeix totes les sortides de
       -[no]aot                  Realitza també una compilació AOT en mètodes després d'emmagatzemar classes a la memòria c
       -aotFilter <signatura>    Només es compilaran amb AOT i s'emmagatzemaran a la memòria cau els mètodes coincidents
          e.g. -aotFilter {mypackage/myclass.mymethod(I)I} només compila mymethod(I)I
          e.g. -aotFilter {mypackage/myclass.mymethod*} compila qualsevol mymethod
          e.g. -aotFilter {mypackage/myclass.*} compila tots els mètodes de myclass
       -aotFilterFile <fitxer>    Només es compilaran amb AOT i s'emmagatzemaran a la memòria cau els mètodes que coincid
                                  memòria cau (el fitxer d'entrada s'ha d'haver creat mitjançant -Xjit:verbose={precompile
                                 vlog=<fitxer>)
       -printvmargs              Imprimeix arguments de VM necessaris per accedir a la memòria cau plena en temps d'execuc
       [jar file]*.[jar][zip]    Llista explícita de fitxers jar que cal emplenar a la memòria cau
                                 Si no s'especifica cap fitxer, es convertiran tots els fitxers .[jar][zip] de searchPath.
Cal especificar exactament una opció.
```

Nota: Quan s'utilitzen memòries cau de classes compartides, el nom especificat per l'opció **-cacheName** no pot tenir més de 53 caràcters.

Es pot especificar una llista de fitxers jar; en aquest cas, només s'afegiran a la memòria cau de classes compartida les classes d'aquests fitxers jar. Si no especifiqueu cap llista de fitxers jar, utilitzeu l'opció **-searchPath <camí accés>** per especificar un arbre de directoris per cercar fitxers .jar o .zip. L'opció per defecte és **-recurse** i indica que es fa una cerca recursiva de fitxers .jar o .zip a l'arbre de directoris. L'opció **-norecurse** indica que només es fa la cerca al directori especificat. Especifiqueu l'opció **-classpath <camí accés classes>** de manera que l'eina admincache pugui fer una cerca a totes les classes necessàries per processar els fitxers jar especificats. Les classes es carreguen a la JVM com a part del procés d'emplenar la memòria cau de classes compartida, de manera que és important que totes les classes i superclasses a les quals es fa referència pugin ser trobades per l'eina admincache quan prova de carregar una classe del fitxer jar.

L'opció **-grow** especifica que s'afegeix un fitxer jar nou al contingut de la memòria cau existent, si hi ha una memòria cau de classes compartida existent que té el mateix nom al directori de memòries cau. L'opció **-nogrow** especifica que un fitxer jar nou substitueix el contingut antic de la memòria cau, si hi ha una memòria cau compartida existent amb el mateix nom al directori de la memòria cau antiga. L'opció **-grow** s'utilitza per afegir fitxers jar nous que no existeixen actualment a la memòria cau de classes compartida, però no per substituir les classes que han canviat. No utilitzeu l'opció **-grow** per emplenar classes que ja són a la memòria cau però que han canviar a causa de les modificacions fetes a l'aplicació. Per actualitzar les classes existents, creeu una memòria cau completament nova amb el contingut de la classe actual. Si no actualitzeu la memòria cau de classes compartida quan canvieu una classe, l'aplicació s'executarà correctament amb el contingut de la classe nova, però no es beneficiarà de la memòria cau de classes compartida. Això és així perquè la classe modificada es carregarà des d'un disc, no des de la memòria cau de classes compartida. Carregar la classe des d'un disc vol dir que el codi compilat AOT no es pot utilitzar per a aquesta classe. Troneu a generar la memòria cau de classes compartida quan canvieu una classe.

Utilitzeu les opcions **-quiet**, **-verbose** i **-noisy** per controlar el nivell de detall proporcionat per admincache.

Per especificar la precompilació anticipada (AOT) per als mètodes de les classes que emplenen la memòria cau de classes compartida, utilitzeu l'opció **-aot**. Per tal d'evitar la precompilació AOT i només emmagatzemar classes a la memòria cau de classes compartida, utilitzeu l'opció **-noaot**. L'opció **-aot** és el valor per defecte.

Per precompilar alguns mètodes de manera selectiva, utilitzeu les opcions **-aotFilter** <*signatura*> o **-aotFilterFile** <*fitxer*>. La <*signatura*> és una expressió regular simplificada per a una signatura de mètode, especificada entre claus, on un asterisc (*) pot substituir qualsevol seqüència de caràcters. Pot ser que hagueu d'especificar la <*signatura*> entre cometes simples, de manera que l'intèrpret d'ordres no interpreti cap dels caràcters de la signatura del mètode.

La Taula 4 mostra alguns exemples de l'opció <*signatura*>.

Taula 4. Exemples de l'opció <signatura>

Signatura	Significat
`-aotFilter '{java/lang/*}'`	AOT compila mètodes del paquet java/lang.
`-aotFilter '{*.sample*}'`	AOT compila mètodes que comencen per "sample".
`-aotFilter '{mypackage/myclass.mymethod(I)I}'`	AOT compila el mètode que té aquesta signatura exacta.

L'opció **-aotFilterFile** <*fitxer*> utilitza el contingut del <*fitxer*> per seleccionar els mètodes per a la compilació AOT. No es compila cap altre mètode amb AOT. El contingut del <*fitxer*> es genera durant una execució inicial de l'aplicació mitjançant l'opció **-Xjit:verbose={precompile},vlog=**<*fitxer*>. La sortida detallada emmagatzemada al <*fitxer*> utilitza un format intern. Aquest format és necessari per a l'opció **-aotFilterFile**.

Nota: L'opció **-vlog=**<*fitxer*> no genera directament cap fitxer anomenat "fitxer". Quan es genera la sortida detallada, s'afegeix una sèrie de data i ID de procés al "fitxer". Si especifiqueu l'opció **-Xjit:verbose={precompile},vlog=my_file**, el nom del fitxer que es genera és semblant a `my_file.<data>.<#>.<ID procés>`. Els camps

addicionals fan que sigui més senzill generar fitxers de registre detallats individuals en escenaris de diverses JVM on pot ser difícil especificar opcions de la línia d'ordres per a una JVM determinada o utilitzar opcions de la línia d'ordres **-Xjit** diferents amb diferents JVM. En un escenari d'una sola JVM, aquests números s'afegeixen al nom del fitxer especificat a la línia d'ordres.

Un fitxer generat es pot utilitzar amb l'opció **-aotFilterFile**, sense que calgui editar res. Diversos fitxers de registre detallats generats per diverses execucions d'aplicació mitjançant l'opció **-Xjit:verbose={precompile},vlog=<fitxer>** es poden unir i proporcionar a l'eina admincache mitjançant l'opció **-aotFilterFile**.

L'opció **-printvmargs** ajuda a garantir que s'especifiquen els arguments correctes a la línia d'ordres quan s'executa l'aplicació.

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

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 aquest exemple, la línia final de la sortida mostra les opcions que caldria afegir a la línia d'ordres quan s'executa l'aplicació, de manera que s'utilitzin les classes i els mètodes AOT emmagatzemats a la memòria cau de classes compartida. Per utilitzar les opcions d'aquest exemple, especifiqueu aquesta ordre:

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

Gestió de memòries cau de classes compartides amb admincache:

L'eina admincache inclou diverses utilitats per gestionar les memòries cau de classes compartides del sistema.

L'eina admincache proporciona utilitats que ajuden en diverses activitats.

- Crear una llista de les memòries cau de classes compartides que hi ha a la memòria cau.
- Proporcionar detalls sobre el contingut d'una memòria cau de classes compartida.
- Eliminar algunes o totes les memòries cau d'un directori de memòries cau específic.

Llista de les memòries cau de classes compartides disponibles:

L'eina admincache proporciona una llista de les memòries cau de classes compartides que hi ha en una memòria cau.

Per obtenir una llista de totes les memòries cau de classes compartides que hi ha en una memòria cau, utilitzeu l'opció **-listAllCaches** i especifiqueu el directori de la memòria cau mitjançant l'opció **-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 aquest exemple, hi ha dues memòries cau de classes compartides compatibles al directori de memòries cau per defecte:

- La memòria cau per defecte per a un usuari amb l'inici de sessió *username*
- Una altra memòria cau anomenada *rtCache*

L'exemple també mostra una memòria cau incompatible anomenada *nonrtCache*. La memòria cau *nonrtCache* l'ha creada la JVM en executar el mode en temps no real. Això vol dir que no s'hi pot accedir mitjançant l'opció **-Xrealtime**.

La JVM en mode de temps real pot veure les memòries cau creades en mode de temps no real. La JVM en mode de temps no real no pot veure les memòries cau creades en mode de temps 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 aquest exemple, es mostra *nonrtCache*, i s'indica com a compatible perquè **-Xrealtime** no s'especifica.

Inspecció del contingut de les memòries cau de classes compartides:

L'eina admincache descriu el contingut d'una memòria cau de classes compartida.

Podeu utilitzar l'opció **-printStats** de l'eina admincache per obtenir una visió general que descrigui el contingut principal d'una memòria cau de classes compartida. Per obtenir informació sobre una memòria cau específica, en un directori de memòries cau específic, utilitzeu les opcions **-cacheName** i **-cacheDir**. L'exemple següent ofereix informació sobre la memòria cau *nonrtCache* al directori de memòries cau per defecte.

```
$ 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: Quan utilitzeu memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.

Hi ha diverses dades útils sobre aquesta memòria cau:

- La mida de la memòria cau, que es mostra a `cache size = 16776852`.
- L'espai disponible a la memòria cau, que es mostra com a `free bytes = 14070360`. Podeu calcular que la memòria cau està plena aproximadament en un 16%.
- El nombre de classes emmagatzemades a la memòria cau, que es mostra a `# ROMClasses = 372`.
- El nombre de mètodes AOT emmagatzemats a la memòria cau, que es mostra a `# AOT Methods = 981`.

Per obtenir més detalls sobre la informació proporcionada per l'opció **-printStats** a l'eina admincache, vegeu utilitat printStats.

L'opció **-printAllStats** proporciona una descripció més detallada del contingut d'una memòria cau de classes compartida. La informació inclou la llista de classes i mètodes AOT emmagatzemats a la memòria cau. La sortida de l'opció **-printAllStats** és detallada.

Les classes de la memòria cau s'indiquen mitjançant línies semblants a aquestes:

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

Aquesta línia indica que la classe java/lang/ClassLoader és a la memòria cau. Les adreces són internes per a la memòria cau de classes compartida, i no són massa útils, tret de quan s'usen per als diagnòstics.

Els mètodes AOT de la memòria cau s'indiquen mitjançant línies semblants a aquestes:

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

Aquestes línies indiquen que el mètode callerClassLoader de la classe java/lang/ClassLoader és a la memòria cau. Les adreces indicades són adreces internes de la memòria cau compartida. La sortida de l'opció **-printAllStats** no mostra la signatura per a cada mètode AOT de la memòria cau, on la signatura està formada pels tipus de paràmetre i el tipus de retorn.

Per obtenir més detalls sobre la informació proporcionada per l'opció **-printAllStats** a l'eina admincache, vegeu utilitat printAllStats.

Destrucció de memòries cau de classes compartides:

L'eina admincache té opcions per esborrar una memòria cau determinada o totes les memòries cau d'un directori determinat de memòries cau.

L'opció **-destroy** de l'eina admincache s'utilitza per esborrar una memòria cau concreta d'un directori específic de memòries cau, si l'usuari té permís per fer-ho. L'opció **-destroyAll** s'utilitza per esborrar totes les memòries cau, si l'usuari té permís per fer-ho. Per exemple:

```
$ admincache -Xrealtime -destroy

admincache 1.02

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

Després d'esborrar la memòria cau, una llista de les memòries cau de classes compartides disponibles del directori de memòries cau per defecte mostra que la memòria cau esborrada ja no existeix:

```
$ 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
```

L'opció **-destroyAll** elimina totes les memòries cau del directori de memòries cau especificat, independentment de si són compatibles o no amb la JVM actual. L'opció **-destroyAll** s'ha d'utilitzar amb molta cura:

```
$ 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 resultat és que ja no hi ha cap memòria cau de classes compartida disponible a la màquina:

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

JVMSHRC005I No shared class caches available
```

Si l'usuari actual no té permís per accedir a una memòria cau, la memòria cau no es destrueix mitjançant les opcions **-destroy** o **-destroyAll**.

Mides pràctiques per a memòries cau de classes compartides:

L'eina admincache proporciona informació per definir la mida de les memòries cau de classes compartides.

Per a les aplicacions més petites, una memòria cau de classes compartida es pot emplenar amb totes les classes i mètodes sense produir una memòria cau de mida prohibitiva. Per a aplicacions més grans, la mida de la memòria cau de classes compartida resultant podria arribar a ser massa grossa a efectes pràctics. Això és així perquè un procés de la JVM ha de tenir suficient espai d'adreces virtuals per poder respondre a tot el contingut de la memòria cau de classes compartida. Hi ha algunes consideracions que podeu aplicar quan utilitzeu la tecnologia de memòria cau de classes compartida.

La memòria cau de classes compartida ha de ser redirigible en la seva totalitat, a qualsevol JVM que s'hi connecti. Això vol dir que cal evitar l'ús de memòries cau de classes compartides de més de 700 MB. L'eina admincache pot predir la mida d'una memòria cau. Si l'eina indica que la memòria cau serà més gran del límit de 700 MB, es mostra un missatge que us adverteix que heu d'emmagatzemar un nombre menor de classes, o que sigueu més selectius quant als mètodes AOT que emmagatzemeu a la memòria cau.

```
$ admincache
-Xrealtime -populate veryBigJar.jar -cp <camí accés classes>

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.
```

L'eina admincache prediu una mida de memòria cau conservadora, en base a la mida total dels fitxers jar especificats o trobats per emplenar-la. Això vol dir que la predicció pot ser que no sigui acurada si el fitxer jar inclou diversos fitxers que no són fitxers de classes. Per obtenir una predicció més precisa de la mida de la memòria cau, creeu versions temporals dels fitxers jar que incloguin només fitxers de classes. Si l'eina admincache encara produeix un missatge d'advertiment, podeu considerar el compilador AOT per precompilar els mètodes del fitxer jar de manera més selectiva, utilitzant les opcions **-aotFilter** <*patró*> o **-aotFilterFile** <*fitxer*>. El missatge de l'eina admincache us recorda que la predicció no té en compte els mètodes AOT filtrats per aquestes opcions.

Per sobreescriure el missatge d'advertiment i continuar amb el pas d'emplenament de la memòria cau, afegiu l'opció **-Xscmx** indicada a la línia d'ordres de l'eina admincache. Si la mida predita és molt gran, pot ser que l'eina admincache no pugui crear una memòria cau de classes compartida de la mida necessària. Per resoldre-ho, reduïu la mida de la memòria cau fins que l'eina admincache pugui continuar.

Quan la memòria cau final s'escriu al disc, té només la mida necessària per contenir les classes i els mètodes AOT especificats. Això vol dir que especificar una mida de memòria cau inicial gran no és cap problema.

Emmagatzematge de classes d'SDK en una memòria cau de classes compartida:

Crear una memòria cau per a tots els fitxers jar de l'SDK pot ser que no sigui necessari per a totes les aplicacions.

El nombre i la mida dels fitxers jar de l'SDK vol dir que provar de crear una memòria cau que contingui tots aquests fitxers jar genera un missatge d'avís on s'indica que la memòria cau resultant serà massa grossa. Per a moltes aplicacions, mai no es fa referència a la majoria dels fitxers jar de l'SDK.

Els fitxers jar principals de l'SDK són al directori `SDK/jre/lib`. Per a la majoria d'aplicacions, el fitxer jar més important és `rt.jar`, que és nou a les versions Java 6. El fitxer `rt.jar` és una col•lecció de classes emmagatzemades prèviament en fitxers jar diferents abans de la versió Java 6. Si empleneu una memòria cau de classes compartida només amb el fitxer `rt.jar`, i compileu tots els seus mètodes amb el compilador AOT, es crea una memòria cau d'un 300 MB de grandària. Una aplicació típica no farà referència a la majoria de mètodes de les classes del fitxer `rt.jar`. Per emplenar la memòria cau de classes compartida amb `rt.jar`:

1. Empleneu només les classes del fitxer `rt.jar` a la memòria cau de classes compartida. Això consumeix aproximadament 50 MB d l'espai de la memòria cau.
2. Utilitzeu l'opció **`-aotFilterFile`** <*fitxer*> per compilar només els mètodes que pot ser que el programa utilitzi. Podeu generar el <*fitxer*> executant l'aplicació.

L'SDK té altres fitxers jar que també són importants i que s'utilitzen sovint:

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

El fitxer `realtime.jar` conté la implementació d'IBM de l'especificació RTSJ (Real Time Specification for Java. Si l'aplicació utilitza alguna de les característiques de l'RTSJ, emmagatzemeu el fitxer `realtime.jar` a la memòria cau de classes compartida per obtenir un rendiment més determinista. El fitxer `vm.jar` conté diverses classes de JVM internes, que normalment s'utilitzen a totes les aplicacions. El fitxer `java.util.jar` conté diverses classes de contenidor, i s'ha d'emmagatzemar a la memòria cau de classes compartida de totes les aplicacions per obtenir un rendiment més determinista.

Altres fitxers jar dels directoris `sdk/jre/lib` i `sdk/jre/lib/ext` es poden emmagatzemar en una memòria cau de classes compartida si una aplicació utilitza aquestes classes. La manera més senzilla d'identificar si l'aplicació utilitza aquestes classes és utilitzant l'opció **`-verbose:dynload`** en executar el programa. L'opció **`-verbose:dynload`** descriu només les classes carregades per l'execució actual de l'aplicació. Per exemple:

```
<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>
```

Aquesta sortida d'exemple mostra les tres classes carregades de dos fitxers jar de l'SDK diferents. La classe java/io/InputStreamReader s'ha carregat del fitxer `rt.jar`. Les classes java/util/LinkedHashSet i java/util/HashSet s'han carregat del fitxer `java.util.jar`.

Altres consideracions sobre l'eina admincache:

En aquesta secció s'ofereix informació útil per treballar amb l'eina admincache.

Emplenament de la memòria cau i mida de la memòria immortal

Quan l'eina **admincache** emplena una memòria cau de classes compartida en mode de temps real, ha de carregar cada classe en temps real a través del procés d'emplenament. Cada classe consumeix memòria immortal i, per tant, és possible que la mida de la memòria immortal no sigui suficient per a totes les classes sol•licitades. Si l'eina **admincache** genera un error OutOfMemory mentre emplena una memòria cau amb massa classes, proveu d'incrementar la mida de la memòria immortal més enllà de la mida per defecte de 16 MB, mitjançant l'opció **-Xgc:immortalMemorySize=32M**.

Quan canvien les classes

Si un fitxer de classes canvia al disc, la tecnologia de memòria cau de classes compartida detecta automàticament que la versió que hi ha a la memòria cau d'aquella classe en una memòria cau de classes compartida no s'ha d'utilitzar. El programa funcionarà correctament, però no es podrà beneficiar de la memòria cau de classes compartida, i no s'utilitzarà cap mètode AOT per a aquesta classe. Si canvieu una classe a la vostra aplicació, torneu a crear la memòria cau de classes compartida. No proveu d'utilitzar l'opció **-grow** per tornar a emplenar només el fitxer jar que conté la classe modificada, perquè aquesta opció no està dissenyada per a l'escenari en què el fitxer jar ja existeix a la memòria cau.

Gestió de memòries cau compartides

Les memòries cau compartides necessiten espai d'adreces encara que no hi hagi cap fitxer carregat. Vegeu “Com gestiona l'IBM JVM la memòria” a la pàgina 106 per obtenir més informació sobre com les memòries cau de classes compartides consumeixen memòria al procés JVM.

Emmagatzematge de fitxers jar precompilats en una memòria cau de classes compartida

Podeu emmagatzemar algunes o totes les classes Java proporcionades per IBM en una memòria cau de classes compartida. Aquest procés utilitza l'opció **-Xrealtime** amb **javac** i l'eina **admincache** per emmagatzemar les classes en una memòria cau de classes compartida.

Abans de començar

L'emmagatzematge anticipat dels fitxers en una memòria cau només s'admet amb l'opció **-Xrealtime** i quan s'executa Java amb l'opció **-Xrealtime**. Podeu utilitzar els mateixos fitxers jar quan executeu Java amb o sense l'opció **-Xrealtime**, però els fitxers jar emmagatzemats a la memòria cau només es poden utilitzar quan s'especifica **-Xrealtime**.

Nota: Quan s'utilitzen memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.

Quant a aquesta tasca

Podeu emmagatzemar els fitxers jar en una memòria cau de classes compartida mitjançant l'eina **admincache**. **admincache** us permet crear l'aplicació de tres maneres diferents.

Nota:
- Si heu definit un temps d'espera al vostre sistema Linux, pot ser que hagueu de sobreescriure'l quan precompileu fitxers jar grans; en cas contrari, la compilació sobrepassarà el temps d'espera i el fitxer jar no es crearà.

Precompilació de totes les classes i els mètodes en una aplicació:

Aquest procediment precompila totes les classes d'una aplicació. Emmagatzema un conjunt de fitxers jar en una memòria de classes compartida. Tots els mètodes de totes les classes dels fitxers jar s'emmagatzemen a la memòria cau. Els fitxers jar optimitzats tenen tots els mètodes compilats.

Quant a aquesta tasca

Per a aquest exemple, l'aplicació resideix al directori especificat per la variable d'entorn *$APP_HOME* i els fitxers jar són al subdirectori *$APP_HOME*`/lib`. L'aplicació també utilitza algunes classes de les proporcionades per IBM a `core.jarrt.jar`. En aquest cas, podeu precompilar només el codi de l'aplicació, és a dir `main.jar` i `util.jar`.

Per defecte, la memòria cau de classes compartida és al directori `/tmp/javasharedresources`. Utilitzeu l'opció **`-cacheDir`** per col•locar la memòria cau en un altre directori. No podeu crear una memòria cau en un sistema de fitxers de xarxa.

Procediment

1. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent: `cd` *$APP_HOME* on *$APP_HOME* és el directori de l'aplicació.
2. Des d'un indicador d'intèrpret d'ordres, introduïu el següent: `cd` *$APP_HOME*`/lib`. *$APP_HOME*`/lib` és el directori en el qual hi ha emmagatzemats els fitxers `main.jar` i `util.jar`.
3. Des d'un indicador d'intèrpret d'ordres, introduïu el següent: `admincache -Xrealtime -populate -aot -classpath` *$APP_HOME*`/lib -searchPath` *$APP_HOME*`/lib -norecurse` . Aquest procediment optimitza cada fitxer jar que troba a *$APP_HOME*`/lib`, escriu la informació de progrés a la pantalla i després crea el fitxer jar nou al directori *$APP_HOME*`/aot`. Podeu especificar un nom de memòria cau amb **`-cacheName <nom>`**, però, si no n'especifiqueu cap, el nom per defecte es basa en l'inici de sessió de l'usuari.

   Nota: El nom especificat per l'opció **`-cacheName`** no pot tenir més de 53 caràcters.
4. Des d'un indicador d'intèrpret d'ordres, si introduïu `admincache -Xrealtime -listAllCaches` es mostra l'existència de la memòria cau.

Què cal fer posteriorment

Per obtenir més opcions, especifiqueu: `admincache -Xrealtime -help`.

Precompilació dels mètodes utilitzats més sovint:

Podeu utilitzar la compilació AOT dirigida per perfils per precompilar només els mètodes que l'aplicació utilitza sovint. La compilació AOT emmagatzema un conjunt de fitxers jar en una memòria cau de classes compartida utilitzant un fitxer

d'opcions que es genera executant l'aplicació amb una opció especial: **-Xjit:verbose={precompile},vlog=optFile**. Només es precompilen els mètodes llistats al fitxer d'opcions.

Abans de començar

Abans de començar, creeu una llista dels mètodes que normalment es compilen mitjançant un compilador JIT.

Quant a aquesta tasca

Podeu editar el fitxer generat per l'opció **-Xjit:verbose={precompile}**. El fitxer és una especificació explícita dels mètodes que cal precompilar. Aquests mètodes són específics, és a dir, contenen la signatura completa per a cada mètode que cal compilar, la qual cosa permet compilar `com/acme/sample.myMethod(J)V` però no `com/acme/sample.myMethod(I)V`.

Nota: Quan utilitzeu memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.

Procediment

1. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent:

   ```
   cd $APP_HOME
   ```

   on *$APP_HOME* és el directori de l'aplicació.
2. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent:

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

   on:

   - *app.precompileOpts* és el nom del fitxer de registre que llista els mètodes compilats amb JIT.
   - *nomAplicació* és el nom de l'aplicació.

   Aquesta ordre crea una llista dels mètodes que es compilen mitjançant JIT.
3. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent:

   ```
   cd $APP_HOME/lib
   ```

   $APP_HOME/lib és el directori en el qual s'emmagatzemen els fitxers jar de la vostra aplicació.
4. Per compilar tots els mètodes d'aplicació a la memòria cau, introduïu:

   ```
   admincache -Xrealtime -populate -cacheName myCache \
              -aotFilterFile $APP_HOME/app.precompileOpts \
              -cp $APP_HOME/lib/demo.jar
   ```
5. Per compilar `realtime.jar` i `vm.jar` a la memòria cau, introduïu:

```
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. Per compilar `rt.jar` a la memòria cau, introduïu:

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

7. Per provar aquesta ordre, executeu la vostra aplicació amb l'opció **-nojit**, que utilitza el codi de la memòria cau. A l'indicador de l'intèrpret d'ordres, introduïu:

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

on *nomAplicació* és el nom de l'aplicació.

Precompilació de fitxers proporcionats per IBM:

Podeu precompilar els fitxers que proporciona IBM, per exemple, `rt.jar`, per assolir un compromís entre el rendiment i la capacitat de predicció.

Quant a aquesta tasca

La precompilació és semblant a la tasca de precompilar els fitxers jar de la vostra aplicació, però s'aplica un requisit addicional en temps d'execució; heu d'assegurar-vos que el vostre camí d'accés de classes d'arrencada estigui ben definit per utilitzar aquests fitxers en lloc dels fitxers del JRE. Podeu fer-ho amb l'opció L'opció **-Xshareclasses**, que indica a la JVM que cerqui primer a la memòria cau de classes especificada davant de les ubicacions del camí d'accés de classes per defecte.

Nota: Quan utilitzeu memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.

Precompileu els fitxers `rt.jar` per utilitzar-los amb l'aplicació:

Procediment

1. Des d'un indicador de l'intèrpret d'ordres, introduïu el següent: `cd $JAVA_HOME/lib` on *$JAVA_HOME* és el vostre directori inicial de Java.
2. Executeu l'eina **admincache**. En un indicador de l'intèrpret d'ordres, escriviu:

   ```
   admincache -Xrealtime -populate -cacheName myCache -classpath <camí d'accés de classes> rt.jar
   ```

   Aquesta ordre emplena la memòria cau anomenada `myCache`, la qual cosa fa que es precompili el fitxer proporcionat per IBM que s'anomena `rt.jar`.
3. Executeu la vostra aplicació especificant l'opció **-Xshareclasses** per indicar el nom de la memòria cau. Per executar la vostra aplicació, introduïu:

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

El compilador Just-In-Time (JIT)

Podeu controlar quan i com opera el compilador JIT mitjançant la classe java.lang.Compiler que es proporciona com a part de la biblioteca de classes SDK estàndard. IBM admet totalment els mètodes Compile.compileClass(), Compiler.enable() i Compiler.disable().

Per exemple, si voleu fer un escalfament de l'aplicació i saber si els mètodes de l'aplicació s'han compilat, podeu cridar el mètode Compiler.disable() després d'haver fer l'escalfament i estar segurs que a compilació JIT no es produirà durant la resta de l'execució de l'aplicació.

Podeu controlar la compilació dels mètodes de dues maneres:

- Especifiqueu el conjunt de mètodes que podeu compilar:

  ```
  Compiler.command("{<especificació de mètodes>}(compile)");
  ```

on *<especificació de mètodes>* és una llista de tots els mètodes que s'han carregat en aquest punt i que cal compilar. *<especificació de mètodes>* descriu un nom de mètode complet. Un asterisc indica una coincidència comodí.

Per exemple, per compilar tots els mètodes que comencen per java.lang.String i que ja s'han carregat, especifiqueu:

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

Nota: Aquesta ordre compila els mètodes de la classe java.lang.String i els de la classe java.lang.StringBuffer, que pot ser que no us interessi. Per compilar només els mètodes de la classe java.lang.String, especifiqueu el següent:

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

- Especifiqueu que tots els mètodes de la cua de compilació es compilaran abans que aquest fil continuï i s'executi:

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

  Assegureu-vos que la cua de compilació estava buida abans d'inhabilitar el compilador. Una tècnica típica per compilar un conjunt de mètodes i classes pot ser aquesta:

```
Compiler.enable();                              // garantir que el compilador està actiu
Compiler.command("{com.mycompany.*}(compile)"); // posar a la cua tots els mètodes que volem compilar
Compiler.command("waitOnCompilationQueue");     // esperar fins que tots els mètodes estiguin compilats
Compiler.disable();                             // desactivar el compilador
```

Determinisme durant les transicions JNI

Per defecte, JIT genera codi optimitzat per a transicions JNI de Java a natiu (J2N) d'alt rendiment. Pot ser que es produeixi un determinisme reduït quan es torna a carregar una biblioteca nativa mitjançant la seqüència de codi següent:

```
RegisterNatives / UnregisterNatives / RegisterNatives
```

Per passar a un codi més lent i més determinista, utilitzeu l'opció de línia d'ordres **-Xjit:disableDirectToJNI**.

Habilitació del JIT

Podeu habilitar de manera explícita el JIT de diverses maneres. Les dues opcions de la línia d'ordres alteren temporalment la variable d'entorn **JAVA_COMPILER**.

Procediment

- Definiu la variable d'entorn **JAVA_COMPILER** amb el valor "jitc" abans d'executar l'aplicació Java. En un indicador de l'intèrpret d'ordres, escriviu:
  - **Per a l'intèrpret d'ordres Korn:** `export JAVA_COMPILER=jitc`

    Nota: Tret que s'indiqui d'una altra manera, en aquesta informació s'utilitzen les ordres de l'intèrpret d'ordres Korn.
  - **Per a l'intèrpret d'ordres Bourne:**

    ```
    JAVA_COMPILER=jitc
    export JAVA_COMPILER
    ```
  - **Per a l'intèrpret d'ordres C:** `setenv JAVA_COMPILER jitc`

  Si la variable d'entorn **JAVA_COMPILER** és una sèrie buida, el JIT segueix inhabilitat. Per inhabilitar la variable d'entorn, a l'indicador de l'intèrpret d'ordres, introduïu `unset` **JAVA_COMPILER**.
- Utilitzeu l'opció **-D** a la línia d'ordres de la JVM per definir la propietat java.compiler amb el valor "jitc". A l'indicador de l'intèrpret d'ordres, introduïu: `java -Djava.compiler=jitc <MyApp>`

- Utilitzeu l'opció **-Xjit** a la línia d'ordres de la JVM. *No* especifiqueu l'opció **-Xint** al mateix temps. En un indicador de l'intèrpret d'ordres, introduïu: `java -Xjit <MyApp>`

Inhabilitació del JIT

Podeu inhabilitar el JIT de diverses maneres. Les dues opcions de la línia d'ordres alteren temporalment la variable d'entorn **JAVA_COMPILER**.

Quant a aquesta tasca

Procediment

- Definiu la variable d'entorn **JAVA_COMPILER** en "NONE" o deixeu la sèrie buida abans d'executar l'aplicació Java. En un indicador de l'intèrpret d'ordres, introduïu:
  - **Per a l'intèrpret d'ordres Korn:** `export JAVA_COMPILER=NONE`

    Nota: Les ordres de l'intèrpret d'ordres Korn s'utilitzen per a la resta d'aquesta informació.
  - **Per a l'intèrpret d'ordres Bourne:**

    ```
    JAVA_COMPILER=NONE
    export JAVA_COMPILER
    ```
  - **Per a l'intèrpret d'ordres C:** `setenv JAVA_COMPILER NONE`
- Utilitzeu l'opció **-D** de la línia d'ordres de la JVM per definir la propietat java.compiler en "NONE" o deixeu la sèrie buida. En un indicador de l'intèrpret d'ordres, introduïu: `java -Djava.compiler=NONE <MyApp>`
- Utilitzeu l'opció **-Xint** a la línia d'ordres de la JVM. En un indicador de l'intèrpret d'ordres, introduïu: `java -Xint <MyApp>`

Determinació si el JIT està habilitat

Podeu determinar l'estat del JIT amb l'opció **-version**.

Procediment

Escriviu el següent en un indicador de l'intèrpret d'ordres:

```
java -version
```

Si el JIT no s'està utilitzant, es visualitza un missatge que inclou el següent:
`(JIT disabled)`
Si el JIT s'està utilitzant, es visualitza un missatge que inclou el següent:
`(JIT enabled)`

Utilització de fils en temps real que no són d'emmagatzematge dinàmic

La recollida de deixalles Metronome proporciona temps de resposta més coherents, però a vegades és adequada per evitar completament interrupcions de la recollida de deixalles.

Els fils NoHeapRealtimeThreads (NHRT) són una extensió dels fils RealtimeThreads. Difereixen dels fils en temps real (RealtimeThreads) quant a que no tenen accés a la memòria d'emmagatzematge dinàmic. Sense accés a l'emmagatzematge dinàmic, els fils NHRT es poden continuar executant fins i tot

durant el cicle de recollida de deixalles, amb algunes restriccions. Sense accés a l'emmagatzematge dinàmic, el model de programació és diferent del model per als fils en temps real.

Consideracions quant a l'ús dels NHRT

Tingueu en compte aquests punts quant als NHRT:

- El motiu principal per utilitzar fils NHRT és amb una tasca que no pot tolerar la recollida de deixalles. Per exemple, si per a la vostra aplicació el temps és un factor crític i no es pot tolerar cap interrupció.
- Si el temps és tant crítica que utilitzeu NHRT, considereu també la possibilitat d'utilitzar el compilador anticipat (AOT); és a dir, utilitzeu l'opció **-Xnojit**.
- Quan utilitzeu l'opció **-Xrealtime**, utilitzeu automàticament el Recollidor de deixalles Metronome. Els avantatges del Recollidor de deixalles Metronome poden ser suficients per a la vostra empresa, de manera que no calgui codificar fils NHRT.
- Els fils NHRT s'executen de manera independent del recollidor de deixalles, perquè tenen una prioritat superior a la del recollidor de deixalles. Els fils Java poden tenir una prioritat entre 1 i 10. Si hi ha fils NHRT, la prioritat dels fils Java es reinicialitza a 0 independentment de la prioritat definida al programa. El recollidor de deixalles s'estableix automàticament a mig pas més amunt que el fil en temps real més alt. Cal definir la prioritat dels NHRT de manera que sigui com a mínim un punt més alta que el fil en temps real més alt. D'aquesta manera, els NHRT són independents del recollidor de deixalles.

  Nota: Els NHRT no queden totalment lliures de la recollida de deixalles, perquè el recollidor de deixalles del fil d'alarma de Metronome s'executa al sistema a la prioritat més alta. Aquesta prioritat permet garantir que la JVM es pot activar per verificar si el recollidor de deixalles ha de fer res. La feina per executar el fil d'alarma de Metronome és petita i no afecta significativament el rendiment. En un sistema de diversos processadors, el fil d'alarma es pot executar simultàniament amb fils NHRT i, per tant, no es produeix cap interrupció per la recollida de deixalles.
- Atès que els NHRT estan limitats a les àrees de memòria amb àmbit i immortals, els mètodes Java realitzen comprovacions que permeten garantir que no s'assignen des de l'emmagatzematge dinàmic. El mètode d'inici comprova i retorna una excepció (`MemoryAccessError`) si s'assignen fils NHRT des de l'emmagatzematge dinàmic. Els NHRT només poden accedir a la memòria `ImmortalMemory` i `ScopedMemory`.
- La semàntica de bloqueig no ha canviat, de manera que els fils NHRT poden ser blocats per fils normals si es comparteix un bloqueig.
- Pot ser que la prioritat d'un fil que utilitza l'emmagatzematge dinàmic s'incrementi en un mètode sincronitzat quan un fil NHRT intenta utilitzar el mateix mètode.
- Utilitzeu cues que no bloquin per a la comunicació entre fils NHRT i fils d'emmagatzematge dinàmic. En cas contrari, separeu els dos tips de fil.

Excepcions

Quan s'utilitzen fils NHRT es poden produir aquestes excepcions:

- `IllegalAssignmentError`. Com a exemple, aquest error es pot produir quan s'intenta crear na referència a la memòria amb àmbit en la memòria immortal.

- `MemoryAccessError`. Com a exemple, aquest error es pot produir quan un fil NHRT intenta fer referència a la memòria d'emmagatzematge dinàmic.

Limitacions de gestió d'esdeveniments asíncrons

Hi ha diversos casos en què els fils NHRT es poden blocar durant la recollida de deixalles, per exemple:

1. Quan un fil NHRT crida fire(), setHandler() o addHandler() en un esdeveniment asíncron que ja està associat amb gestors assignats des de la memòria d'emmagatzematge dinàmic
2. Quan un fil NHRT crida destroy(), start() o stop() en un temporitzador associat amb gestors assignats des de la memòria d'emmagatzematge dinàmic
3. Un fil NHRT és el darrer fil que surt de l'àmbit i tanca els temporitzadors o esdeveniments asíncrons de l'àmbit. Això no obstant, els temporitzadors o esdeveniments asíncrons tenen gestors associats que estan assignats des de la memòria d'emmagatzematge dinàmic.

Per evitar aquestes situacions amb els fils NHRT:

1. Eviteu afegir gestors assignats des de l'emmagatzematge dinàmic a esdeveniments asíncrons o temporitzadors que pot ser que siguin activats per un fil NHRT.
2. Eviteu les condicions en què un fil NHRT surt en últim lloc d'un àmbit que té temporitzadors o esdeveniments asíncrons amb gestors associats que estan assignats des de la memòria d'emmagatzematge dinàmic.

Restriccions de memòria i planificació

La JVM impedeix que els fils en temps real que no són d'emmagatzematge dinàmic carreguin referències a objectes que estan a l'emmagatzematge dinàmic a la seva pila operativa. Per fer-ho, genera un error javax.realtime.MemoryAccessError.

La JVM també es protegeix davant les referències a objectes de la memòria amb àmbit que s'emmagatzemen a l'emmagatzematge dinàmic o a la memòria immortal. Tot i que la memòria amb àmbit no la utilitzen de manera exclusiva els NHRT, és probable que s'utilitzi si la memòria immortal no és adequada i si cal desassignació de memòria en un context NHRT.

Mentre s'executa un NHRT, si emplena un camp amb una referència a un objecte, pot sobreescriure satisfactòriament qualsevol referència pre-existent per a un objecte de l'emmagatzematge dinàmic d'aquest camp. L'NHRT sobreescriurà satisfactòriament la referència pre-existent sense generar cap error MemoryAccessError.

Restriccions de la càrrega de classes

Les classes es carreguen a les mateixes àrees de memòria que el carregador de classes. L'àrea per defecte per als carregadors de classes és la memòria immortal.

Per tal que altres aplicacions proporcionin temps de resposta esperats, han de ser "calentes". Les aplicacions han de carregar les seves classes en les fases inicials de manera que la càrrega de classes no interrompi els fils en temps real i més tard els gestors d'esdeveniments asíncrons.

Restriccions dels fils Java quan s'executen amb fils NHRT

Atès que les propietats del sistema es comparteixen en una JVM i que qualsevol fil pot accedir a les propietats del sistema, cal anar amb cura quan s'utilitzen els mètodes getProperties i setProperties a les JVM en les quals s'executen fils NHRT. Per tal que els NHRT puguin accedir a les propietats del sistema, han de ser a la memòria immortal.

La classe `java.lang.System` proporciona diversos mètodes que permeten que els fils interaccionin amb les propietats del sistema. Els mètodes són:

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

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

La JVM en temps real utilitza una instància de la classe com.ibm.realtime.ImmortalProperties creada específicament per a l'objecte de JVM en temps real per emmagatzemar-hi totes les propietats del sistema. L'ús d'aquesta instància permet garantir que les crides als mètodes System.setProperty() o System.getProperties.setProperty() faran que la propietat s'emmagatzemi a la memòria immortal. En aquest cas, no cal cap codi s'usuari especial, però és important comprendre que, cada vegada que es defineix una propietat, es consumeix memòria immortal.

Les crides al mètode setProperties() són una mica més difícils perquè s'utilitza un objecte Properties compartit per emmagatzemar propietats del sistema. Si una aplicació s'executa en una JVM en temps real que té fils NHRT en execució, les crides al mètode setProperties han de passar en una instància d'una classe com.ibm.realtime.ImmortalProperties, o subclasse, creada a la memòria immortal. L'ús d'aquesta instància permet garantir que totes les propietats que es defineixen amb el mètode setProperties estan emmagatzemades a la memòria immortal.

Nota: En cridar setProperties(null), es crea internament un objecte ImmortalProperties nou amb un conjunt de propietats per defecte, que utilitza memòria immortal addicional.

Les crides al mètode getProperties() retornen l'objecte definit o l'objecte de propietats per defecte, que és un objecte com.ibm.realtime.ImmortalProperties. Per maximitzar la compatibilitat amb codi existent que crida el mètode getProperties(), l'objecte ImmortalProperties serialitza l'objecte i després el deserialitza en una JVM estàndard. El comportament per defecte per a la deserialització de l'objecte ImmortalProperties és serialitzar un objecte Properties normal, perquè les JVM estàndard no tenen l'objecte ImmortalProperties i la deserialització falla. Per alterar temporalment aquest comportament, la classe ImmortalProperties proporciona el mètode enabledReplacement(boolean), que, si es crida amb el valor false, inhabilita el comportament per defecte. En aquest cas, la serialització serialitza l'objecte ImmortalProperties i després és possible deserialitzar-lo i utilitzar l'objecte resultant en una crida al mètode System.setProperties en una JVM en temps real.

Nota: La deserialització es produeix a la memòria immortal, i pot ser consumeixi aquest recurs limitat en excés.

Gestor de seguretat

El gestor de seguretat definit per al sistema l'utilitzen tots els tipus de fil de la JVM. És per això que, en una JVM en temps real en la qual s'executen NHRT, el gestor de seguretat s'ha d'assignar a la memòria immortal. La JVM en temps real garanteix que el gestor de seguretat especificat a les opcions de la línia d'ordres estigui assignat a la memòria immortal. El gestor de seguretat també es pot definir a través de crides al mètode System.setSecurityManager(SecurityManager). Si l'aplicació defineix el gestor de seguretat d'aquesta manera, ha de garantir que el gestor de seguretat s'hagi assignat des de la memòria immortal, de manera que els NHRT es pugui executar correctament.

Les excepcions generades i els objectes que retorna el gestor de seguretat han de ser a la memòria immortal, si s'emmagatzemen a la memòria cau, o s'han d'assignar al context d'assignació actual.

Sincronització

La classe MonitorControl i els seves subclasses PriorityInheritance gestionen la sincronització, concretament el control de la inversió de prioritats. Aquestes classes permeten definir una política de control de la inversió de prioritats com a política per defecte o per a objectes concrets.

Les classes WaitFreeReadQueue, WaitFreeWriteQueue i WaitFreeDequeue permeten una comunicació sense esperes entre objectes planificables (especialment instàncies de NoHeapRealtimeThread) i fils Java normals.

Les classes WaitFree proporcionen un accés segur i simultani a les dades compartides entre les instàncies de NoHeapRealtimeThread i d'objectes planificables subjectes als retards de la recollida de deixalles.

Seguretat de les classes de temps real que no és d'emmagatzematge dinàmic

En algunes circumstàncies, pot ser que algunes parts de l'API JSE no es puguin utilitzar en un context que no sigui d'emmagatzematge dinàmic. Hi ha restriccions quant a les classes que es comparteixen entre fils d'emmagatzematge dinàmic i fils que no són d'emmagatzematge dinàmic. Cal tenir present les classes subministrades amb la JVM que es poden utilitzar amb tota seguretat.

Com compartir objectes

Els mètodes que s'executen en fils en temps real que no són d'emmagatzematge dinàmic generen un error javax.realtime.MemoryAccessError quan proven de carregar una referència a un objecte d'un emmagatzematge dinàmic.

A la Figura 3 a la pàgina 59 es mostra un exemple del tipus de codi que cal evitar:

```
/**
 * NHRTError1
 *
 * Aquest exemple és una simple demostració d'un NHRT que accedeix
 * a una referència d'objecte d'emmagatzematge dinàmic.
 *
 * L'error generat és:
 *
 * 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) {
            // TODO Auto-generated catch block
            e.printStackTrace( );
        }

    }
```

Figura 3. Exemple de NHRT que accedeix a una referència d'objecte d'emmagatzematge dinàmic

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

        /* Imprimeix la sèrie a través de la referència estàtica a NHRTError1.message */
        public void run() {
            System.out.println("Message: " + NHRTError1.message);
        }
    }

}
```

Figura 4. Exemple d'un NHRT que accedeix a una referència d'objecte d'emmagatzematge dinàmic (continuació de la figura 1)

La Figura 3 il•lustra un error **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 ha de ser possible accedir a un objecte des d'un fil en temps real que no sigui d'emmagatzematge dinàmic i un fil Java estàndard, l'objecte s'ha d'assignar a la memòria immortal. De la mateixa manera, si s'ha de poder accedir a un objecte des d'un fil en temps real que no sigui d'emmagatzematge dinàmic i un fil en temps real, l'objecte també es pot mantenir en una àrea de memòria amb àmbit.

A la Figura 3 a la pàgina 59, la referència a la sèrie "This on the heap." era en una variable de classe. Aquesta variable és accessible per als NHRT perquè totes les classes estan assignades a la memòria immortal. De manera alternativa, la sèrie es podria haver passat al constructor de fils NHRT.

La majoria d'objectes inclouen referències a altres objectes i, per tant, cal anar en compte quan es comparteixen aquests objectes entre fils ordinaris i fils NHRT. Un exemple típic és una llista LinkedList assignada en memòria immortal, compartida entre un fil ordinari i un fil NHRT. Si no aneu prou amb compte, pot ser que el fil estàndard introdueixi objectes a la llista LinkedList que no són a l'emmagatzematge dinàmic. Una altra preocupació encara més important és el fet que les estructures de dades assignades per LinkedList per fer el seguiment d'objectes s'assignen a l'emmagatzematge dinàmic mitjançant el fil ordinari, la qual cosa és molt fàcil que generi un error MemoryAccessError a l'NHRT.

Algunes classes no es poden compartir de manera segura entre fils NHRT i altres fils, independentment d'on s'assignin les seves instàncies individuals. Aquestes classes es basen en objectes emmagatzemats a les variables de classe, normalment amb la finalitat d'emmagatzemar a la memòria cau. InetAddress és un exemple típic que emmagatzema adreces a la memòria cau; si el primer fil que crida a determinats mètodes a InetAddress s'executa a l'emmagatzematge dinàmic, més endavant no serà segur cridar els mateixos mètodes des de fils NHRT.

Bloqueig d'objectes amb fils NHRT

Els NHRT han d'evitar la sincronització amb altres fils. Considereu l'escenari següent:

- Un fil en temps ral de prioritat baixa entra en un bloqueig o mètode sincronitzat, que se sincronitza en un objecte.
- Un NHRT de prioritat alta es bloca quan prova de sincronitzar-se al mateix objecte.
- L'herència de la prioritat fa que el fil en temps real assumeixi temporalment la mateixa prioritat que el fil NHRT.
- A continuació, la recollida de deixalles s'executa amb una prioritat més alta que el fil NHRT i, per tant, pot interrompre el fil NHRT. El motiu per utilitzar el fil NHRT és evitar la interrupció per part de la recollida de deixalles; per tant, aquest escenari nega l'ús del fil NHRT.

De vegades no es pot evitar que els NHRT i altre fils se sincronitzin al mateix objecte, però cal minimitzar aquesta possibilitat. Aneu en compte per evitar una sincronització innecessària quan compartiu objectes.

Restriccions sobre classes segures

Hi ha determinades consideracions que cal tenir en compte quan una aplicació conté objectes de fils en temps real i objectes de fils en temps real que no són d'emmagatzematge dinàmic.

- El fil en temps real que no és d'emmagatzematge dinàmic pot patir errors MemoryAccessErrors provocats per la interacció amb el fil en temps real.

- El fil en temps real que no és d'emmagatzematge dinàmic pot ser que es retardi de manera accidental per causa de la recollida d'escombraries provocada pel fil en temps real.

Errors MemoryAccessErrors causats per un fil en temps real que no és d'emmagatzematge dinàmic

Quan els dos tipus de fil criden mètodes de la mateixa classe, pot ser que el fil en temps real "pol•lucioni" les variables estàtiques de la classe amb objectes assignats de l'emmagatzematge dinàmic. El fil en temps real que no és de l'emmagatzematge dinàmic rebrà un error MemoryAccessError quan provi d'accedir a aquests objectes de l'emmagatzematge dinàmic. La pol•lució també es pot produir en instàncies de la classe. Malauradament, és molt probable que tots dos problemes apareguin en patrons de codificació típics i, per tant, val la pena analitzar un parell de casos.

Si una classe realitza una operació que consumeix força temps, sovint opta per emmagatzemar a la memòria cau el resultat per millorar el rendiment de les operacions següents. La memòria cau acostuma a ser una col•lecció, com ara HashMap, ancorada en una variable estàtica de la classe. Un fil en temps real que opera en un context d'emmagatzematge dinàmic pot emmagatzemar un objecte d'emmagatzematge dinàmic en aquesta col•lecció, fet que no només afegeix l'objecte en sí, sinó que també afegeix objectes d'infrastructura a la col•lecció; per exemple, parts de l'índex. Quan un fil en temps real que no és d'emmagatzematge dinàmic més tard prova d'accedir a la col•lecció, encara que no provi d'accedir a l'objecte afegit per l'altre fil, prova de carregar els objectes d'infrastructura i, per tant, rep un error MemoryAccessError. Quan es desenvolupen les biblioteques de classes i s'ajusten per al rendiment, aquestes memòries cau passen a ser més comunes.

Una instància de classe també es pot emplenar de diverses maneres mitjançant objectes d'emmagatzematge dinàmic. Considereu una instància creada a la memòria immortal a la qual, per tant, poden accedir tots dos tipus de fil. Si el primer ús de l'objecte és per un fil en temps real del context d'emmagatzematge dinàmic, pot ser que hi hagi un objecte secundari emmagatzemat en un camp de l'objecte original. Si l'objecte secundari és al context de l'emmagatzematge dinàmic, l'ús posterior del fil en temps real que no és d'emmagatzematge dinàmic mostra un error MemoryAccessError. Aquests objectes secundaris no sempre s'afegeixen al primer ús, sinó que sovint s'afegeixen després de nombrosos usos, i pot ser que estiguin dissenyats per millorar el rendiment de mètodes molt utilitzats.

Fil que no és d'emmagatzematge dinàmic retardat per la recollida de deixalles

Per tal d'evitar que els fils que no són d'emmagatzematge dinàmic es retardin per causa de la recollida de deixalles, cal assignar-los prioritats que siguin és altes que les de la resta de fils.

A més, si una classe conté mètodes sincronitzats, pot ser que un fil en temps real que no sigui d'emmagatzematge dinàmic i que cridi aquests mètodes es retardi de manera no intencionada per causa de la recollida de deixalles. Aquest escenari es descriu a "Bloqueig d'objectes amb fils NHRT" a la pàgina 60.

Si una classe conté mètodes sincronitzats (mètodes estàtics o d'instància), pot ser que un fil de temps real que no sigui d'emmagatzematge dinàmic que cridi aquests mètodes es retardi de manera no intencionada per causa de la recollida de deixalles. El problema es produeix si un fil en temps no real accedeix a un mètode

sincronitzat (estàtic o d'instància) al punt en què el fil en temps real que no és d'emmagatzematge dinàmic intenta cridar un altre mètode sincronitzat que es blocarà per esperar que l'altre fil finalitzi. Si el fil en temps real que no és d'emmagatzematge dinàmic té una prioritat superior a la del fil en temps real, la prioritat del fil en temps real augmenta. Si, aleshores, es força el fil perquè esperi una interrupció de la recollida de deixalles, és possible invertir la prioritat, perquè el fil de recollida de deixalles té una prioritat superior a la del fil en temps real que té la prioritat més alta, que post ser que no sigui tan alta com el fil en temps real que no és d'emmagatzematge dinàmic que actualment està blocat esperant entrar al mètode sincronitzat.

L'única manera de corregir problemes d'aquest tipus és garantir que fils en temps real que no són d'emmagatzematge dinàmic mai no cridin mètodes sincronitzats en classes o instàncies que es comparteixen amb altre tipus de fil. Malauradament, no sempre queda clar a partir de la signatura si un mètode està sincronitzat; pot ser, per exemple, que contingui un bloc sincronitzat o una crida a un mètode sincronitzat.

Resum

La classe NoHeapRealtimeThread afegeix molta complexitat a l'entorn en temps real i pot ser que sorgeixi un nombre considerable de problemes quan en un entorn s'utilitza una combinació de tipus de fil. Durant el desenvolupament d'una aplicació, heu de designar amb cura àrees en les quals teniu ús compartit de diferents tipus de fil. És especialment important l'ús que aquests fils fan de les classes de l'SDK. A causa de la complexitat de l'anàlisi, és impossible garantir que totes les classes proporcionades a l'SDK siguin segures per a l'ús compartit. En comptes d'això, s'ha verificat un subconjunt reduït de les classes. Inicialment, la verificació s'ha centrat en l'aspecte MemoryAccessError i el resultat és una llista de classes analitzades, provades i modificades en cas necessari per garantir que poden ser utilitzades per fils que no són d'emmagatzematge dinàmic i per altres tipus de fil.

Classes segures

En aquesta secció s'indica un conjunt de classes que es considera que són segures per ésser utilitzades al fil NoHeapRealtimeThread i en altres tipus de fil.

La preocupació principal se centra en l'aspecte de seguretat MemoryAccessError. A la llista següent s'indiquen les classes que es poden utilitzar als tres tipus de fil a la mateixa JVM.

Nota: Pot ser que no sempre es puguin compartir de manera segura les instàncies individuals de les classes.

Seguiu aquests passos per assegurar-vos que tots els tipus de fil puguin utilitzar una classe de forma segura:

- La instància s'ha de crear en una àrea de memòria a la qual tingui accés el fil que vol accedir a la instància.
- Si la classe té camps estàtics públics, eviteu emmagatzemar objectes d'emmagatzematge dinàmic en aquests camps.
- Si la classe té camps d'instància públics, eviteu emmagatzemar objectes d'emmagatzematge dinàmic en aquests camps.

No totes les classes proporcionades per IBM són segures per als fils NHRT. Els paquets següents contenen classes que són segures per als fils NHRT:

- Paquet java.lang
- Paquet java.lang.reflect
- Paquet java.lang.ref (totes les classes)
- Paquet java.net
- Paquet java.io
- Paquet java.math

Aquestes taules mostren les classes dins d'aquests paquets que no són segures per als fils NHRT:

Taula 5. Classes del paquet java.lang que no són segures per a fils NHRT

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

Taula 6. Classes de paquet java.lang.reflect que no són segures per a fils NHRT

Classe	Mètode
java.lang.reflect.Proxy.*	*

Taula 7. Classes del paquet java.net que no són segures per a fils NHRT

Classe	Mètode
java.net.SocketPermission.*	newPermissionCollection()Ljava.net.SocketPermissionCollection;

Taula 8. Classes del paquet java.io que no són segures per a fils NHRT

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

Taula 9. Classes del paquet java.math que no són segures per a fils NHRT

Classe	Mètode
java.math.BigInteger	*

Els paquets poden incloure subpaquets que continguin classes no segures. Per exemple, les classes següents no són segures per als fils NHRT:
- java.lang.management.*
- java.lang.annotation.*
- java.lang.instrument.*

Encara que una classe es consideri segura per als fils NHRT, és possible que no sigui adient per utilitzar-la en un fil NHRT. Els desenvolupadors d'aplicacions han

de determinar els requisits en temps real de les classes cas per cas, independentment de si la classe és segura per a fils NHRT.

Capacitat de compartir de dades de classes entre les JVM

La JVM (Java Virtual Machine) us permet compartir dades de classes entre les JVM emmagatzemant-es en un fitxer de memòria cau assignat a memòria del disc.

La compartició redueix el consum d'emmagatzematge virtual global quan més d'una JVM comparteix una memòria cau. La compartició també redueix el temps d'inici d'una JVM un cop s'ha creat la memòria cau. La memòria cau de classes compartida és independent de cap altra JVM activa i persisteix fins que es destrueix. Una memòria cau compartida pot contenir:

- Classes bootstrap
- Classes d'aplicació
- Metadades que descriuen les classes
- Codi compilat de manera anticipada (AOT)

Les memòries cau de classes compartides es poden utilitzar a l'IBM WebSphere Real Time for RT Linux en els modes de temps real i temps no real, però el format, la creació i les tècniques d'emplenament de la memòria cau varien. Les memòries cau en mode de temps real no són compatibles amb les memòries cau en mode de temps no real. En mode de temps no real, les memòries cau es creen i s'emplenen de la mateixa manera que la JVM estàndard. Això vol dir que la JVM crea i emplena la memòria cau de la mateixa manera que executa una aplicació, és adir, en un procés transparent per a l'usuari. En mode te temps real, mitjançant l'opció **-Xrealtime**, les memòries cau de classes compartides han de ser creades i emplenades per l'eina **admincache**, utilitzant l'opció **-populate**. Les aplicacions que s'executen en mode de temps real poden llegir contingut de la memòria cau emplenada prèviament, però no poden modificar-lo.

Utilitzeu l'eina **admincache** per crear, emplenar i destruir memòries cau.

Per permetre que una aplicació utilitzi una memòria cau de classes compartides, afegiu l'opció **-Xshareclasses** a la seva línia d'ordres. Com que les memòries en mode de temps real són de només lectura, algunes subopcions de mode que no és de temps real de **-Xshareclasses** no estan disponibles en mode de temps real.

Per obtenir informació, vegeu “Utilització de l'eina admincache” a la pàgina 39, “Compartició de dades de classe entre les JVM per al mode en temps no real” a la pàgina 93 i “Diagnòstics de classes compartides” a la pàgina 141.

Execució d'aplicacions amb una memòria cau de classes compartida

Per executar una aplicació amb una memòria de classes compartida, utilitzeu l'opció **-Xshareclasses** a la línia d'ordres.

A la Taula 10 a la pàgina 65 es mostren les subopcions disponibles quan s'executa una aplicació en mode de temps real utilitzant l'opció **-Xshareclasses**.

Taula 10. Subopcions disponibles quan s'executa una aplicació en mode de temps real

Opció	Significat
`cacheDir=<directori>`	Defineix el directori on es llegeixen i s'escriuen les dades de la memòria cau de classes compartida. Per defecte, el *<directori>* és `/tmp/javasharedresources`. El nom del directori ha de coincidir amb el directori especificat a l'opció **-cacheDir** a l'ordre admincache per crear la memòria cau.
`name=<nom>`	Nom de la memòria cau de classes compartida que cal utilitzar. El nom ha de coincidir amb el nom especificat a l'opció **-cacheName** a l'ordre admincache per crear la memòria cau. El nom no pot tenir més de 53 caràcters.
`none`	Inhabilita explícitament la capacitat de compartir classes. Pot afegir-se al final d'una línia d'ordres per inhabilitar la compartició de dades de classe. Aquesta subopció altera temporalment els arguments d'ús compartit de classes que s'hagin trobat abans a la línia d'ordres.
`nonfatal`	Inicia sempre la JVM independentment dels errors o els advertiments. Permet que la JVM s'iniciï fins i tot si l'ús compartit de dades de classe té errors. El comportament típic de la JVM és que no s'iniciï si la compartició de dades de classe té errors. Si seleccioneu **nonfatal** i la memòria cau de classes compartida no s'inicialitza , la JVM intenta connectar-se a la memòria cau en mode només de lectura. Si aquest intent no és satisfactori, la JVM s'inicia sense ús compartit de dades de classe.
`silent`	Suprimeix tots els missatges de sortida. Desactiva tots els missatges de les classes compartides, inclosos els missatges d'error. Es visualitzen missatges d'errors no recuperables, els qual impedeixen la inicialització de la JVM.
`verbose`	Habilita la sortida detallada, fet que proporciona l'estat global de la memòria cau de classes compartides i missatges d'error més detallats.
`verboseAOT`	Habilita la sortida detallada quan es troba codi compilat AOT a la memòria cau, per exemple, durant les sol•licituds de càrrega de mètodes AOT.
`verboseHelper`	Habilita la sortida detallada de l'API d'ajuda de Java. Aquesta sortida mostra com el carregador de classes utilitza l'API d'ajuda.
`verboseIO`	Habilita la sortida detallada de les sol•licituds de càrrega de classes. Aquesta opció proporciona una sortida detallada de l'activitat d'E/S de la memòria cau, i dóna informació sobre les classes que es troben.

Per garantir que aquestes opcions són correctes, utilitzeu l'opció **-printvmargs** amb admincache (consulteu **-printvmargs** per obtenir més informació). L'opció **nonfatal** no és adequada per a l'ús general perquè força la JVM a ignorar els advertiments i els errors relacionats amb la memòria cau de classes compartida. L'opció **none** inhabilita de manera explícita la capacitat de compartir classes i equival a ometre l'opció **-Xshareclasses** a la línia d'ordres.

Per obtenir informació detallada sobre les subopcions de **-Xshareclasses**, consulteu Opcions de línia d'ordres de compartició de dades de classe.

Utilització del recollidor de deixalles Metronome

El recollidor de deixalles Metronome substitueix el recollidor de deixalles estàndard al WebSphere Real Time for RT Linux.

Control de la utilització del processador

Podeu limitar la quantitat de potència de processament disponible al recollidor de deixalles Metronome.

Podeu controlar la recollida de deixalles amb el recollidor de deixalles Metronome mitjançant l'opció **-Xgc:targetUtilization=N** per limitar la quantitat de CPU utilitzada pel recollidor de deixalles.

Per exemple:

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

L'exemple especifica que l'aplicació s'executa en un 80% cada 60 mil•lisegons. El 20% del temps que queda s'utilitza per a la recollida de deixalles. El recollidor de deixalles Metronome garanteix nivells d'ús sempre que tingui prou recursos. La recollida de deixalles s'inicia quan la quantitat d'espai lliure a l'emmagatzematge dinàmic és inferior al llindar determinat de manera dinàmica.

Ajustament del Recollidor de deixalles Metronome

Podeu ajustar l'entorn en temps real controlant la quantitat de memòria que utilitza l'aplicació. Per exemple, utilitzeu **-Xmx**, **-Xgc:immortalMemorySize=size**, **-Xgc:scopedMemoryMaximumSize=size** i **-Xgc:targetUtilization=N**.

- Utilitzeu les opcions **-Xmx** per limitar la mida de l'emmagatzematge dinàmic.

  El valor triat s'utilitza com a límit superior de la mida d'emmagatzematge dinàmic i, per tant, reflecteix l'ús probable amb el temps. Si trieu un valor que sigui massa baix, augmenta la freqüència de la recollida de deixalles i, com a conseqüència, el rendiment general és més baix, tot i que es redueix l'impacte de la memòria. Per a un bon rendiment en temps real, eviteu la paginació. És normal garantir que l'impacte de tots els processos en execució d'una màquina no sobrepassin la mida de la memòria física.
- Utilitzeu l'opció **-Xgc:immortalMemorySize=size** per controlar la mida de l'àrea de la memòria immortal.

  Heu d'analitzar amb cura l'ús de la memòria immortal. L'aplicació "ideal" utilitza memòria durant el procés d'arrencada, però després ja no n'utilitza. Si l'assignació d'objectes immortals continua, l'aplicació es pot seguir executant fins que s'hagi exhaurit la memòria immortal. L'ús actual es pot obtenir afegint el següent:

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

al vostre codi.

- Utilitzeu l'opció **-Xgc:scopedMemoryMaximumSize=size** per garantir que les aplicacions no sol•liciten quantitats excessives de memòria amb àmbit. Utilitzeu aquesta per als diagnòstics, no per a l'ajustament.
- Definiu l'opció **-Xgc:targetUtilization=N** per garantir que en les pitjors condicions (índex màxim d'assignació d'objectes d'emmagatzematge dinàmic), el recollidor de deixalles pot recollir deixalles a una velocitat superior a la que l'aplicació les genera.

  Normalment, el valor per defecte és suficient, però pot ser que el rendiment de l'aplicació millori si incrementeu la utilització fins al punt en què el recollidor pot recollir deixalles amb una mica més de rapidesa amb què l'aplicació les genera.
- Utilitzeu l'opció **-Xgcthreads <n>** per crear fils addicionals per executar la recollida de deixalles en paral•lel.

  Per defecte, s'utilitza un fil. Si la càrrega de treball té un índex alt de generació de deixalles, i s'executa en un multiprocessador simètric amb cicles de CPU disponibles, el rendiment sortirà beneficiat si definiu aquest paràmetre amb el valor >1.

  Nota: Definir aquest paràmetre en un valor alt pot tenir un efecte negatiu al rendiment.

Limitació del recollidor de deixalles Metronome

En determinades circumstàncies, rarament, és possible que experimenteu unes pauses més llargues del que espereu durant la recollida de deixalles.

Durant la recollida de deixalles, s'utilitza un procés d'exploració arrel. El recollidor de deixalles guia l'emmagatzematge dinàmic, començant per referències actives conegudes. Aquestes referències inclouen:

- Variables de referència actives a les piles de crides de fils actius.
- Referències estàtiques.
- Totes les referències a objectes de les memòries immortals i amb àmbit.

Per trobar totes les referències actives a objectes en una pila de fil d'aplicació, el recollidor de deixalles explora tots els marcs de pila de la pila de crides del fil. Cada pila de fils activa s'explora en un pas que no es pot interrompre. Això vol dir que l'exploració s'ha de fer dins d'un sol pause de recollida de deixalles.

L'efecte és que el rendiment del sistema pot ser pitjor del que s'esperava si teniu alguns fils amb piles molt profundes, a causa de les pauses llargues de recollida de deixalles al principi del cicle de recollida.

La memòria immortal es processa de manera incremental. Tota la resta d'àrees de memòria amb àmbit es processen en un pas atòmic i sense interrupcions. Per tant, un ús significatiu de les àrees de memòria amb àmbit pot comportar un rendiment del sistema pitjor de l'esperat, a causa de les llargues pauses de recollida de deixalles quan l'exploració arrel processa memòria amb àmbit.

Capítol 6. Desenvolupament d'aplicacions

Informació important sobre com escriure aplicacions en temps real, inclosos exemples de codi.

- "Escriptura d'aplicacions Java per explotar el temps real"
- "Aplicació d'exemple" a la pàgina 81
- "Mapa hash en temps real d'exemple" a la pàgina 88
- "Desenvolupament d'aplicacions WebSphere Real Time for RT Linux mitjançant Eclipse" a la pàgina 89

Escriptura d'aplicacions Java per explotar el temps real

Aquests exemples descriuen com explotar l'entorn en temps real. Van des de l'exemple més simple, executant una aplicació Java en temps real sense cap modificació al codi, a un procés més complex de planificació i escriptura de fils en temps real que no són d'emmagatzematge dinàmic. Es proporcionen arguments per ajudar-vos a determinar quin enfocament pot ser el més adequat per a les vostres aplicacions.

Introducció a l'escriptura d'aplicacions en temps real

No cal que escriviu aplicacions en temps real i que no són d'emmagatzematge dinàmic elaborades per explotar les característiques de la tecnologia de temps real. Alguns dels avantatges es poden utilitzar fent molts pocs canvis al codi existent.

Per als programadors d'aplicacions, tot seguit s'indiquen els passos que podeu dur a terme per explotar el WebSphere Real Time for RT Linux:

1. Podeu executar una aplicació Java en una JVM en temps real per beneficiar-vos de la recollida de deixalles de Metronome i assolir una millora significativa en la capacitat de predicció del temps d'execució de l'aplicació.
2. Afegiu l'opció **-Xnojit** després d'haver precompilat el codi per utilitzar el compilador anticipat (AOT). Consulteu "Emmagatzematge de fitxers jar precompilats en una memòria cau de classes compartida" a la pàgina 49.
3. Substituïu java.lang.Thread per javax.realtime.RealtimeThread a la vostra aplicació. Pot ser que veieu una lleugera millora en comparació amb l'opció AOT.

   L'avantatge principal d'utilitzar el fils en temps real és la capacitat de controlar la prioritat que assigneu a cadascun dels fils. També es pot fer que els fils en temps real siguin periòdics. Per explotar aquests avantatges, heu d'estar preparats per fer canvis a l'aplicació.
4. Planifiqueu i escriviu una aplicació específica per utilitzar el fils en temps real i els gestors d'esdeveniments asíncrons per gestionar temporitzadors o esdeveniments externs. Considereu aquests tres factors:
   - Planificar la prioritat que assigneu al fils en temps real
   - Decidir quines àrees de memòria utilitzareu perquè continguin els objectes
   - Comunicar-se amb els gestors d'esdeveniments
5. Planifiqueu i escriviu una aplicació específica per utilitzar el fils en temps real que no són d'emmagatzematge dinàmic. Els fils en temps real que no són d'emmagatzematge dinàmic són extensions del fils en temps real i heu de tenir en compte la prioritat que assigneu i l'àrea de memòria. En general, aquest pas només s'ha de dur a terme si l'aplicació ha de gestionar esdeveniments en

temps comparables al temps de pausa de la recollida de deixalles (sub-mil•lisegons). No infravaloreu la complexitat del desenvolupament amb fils fils en temps real que no són d'emmagatzematge dinàmic.

A la Figura 5 es mostren els passos descrits anteriorment.

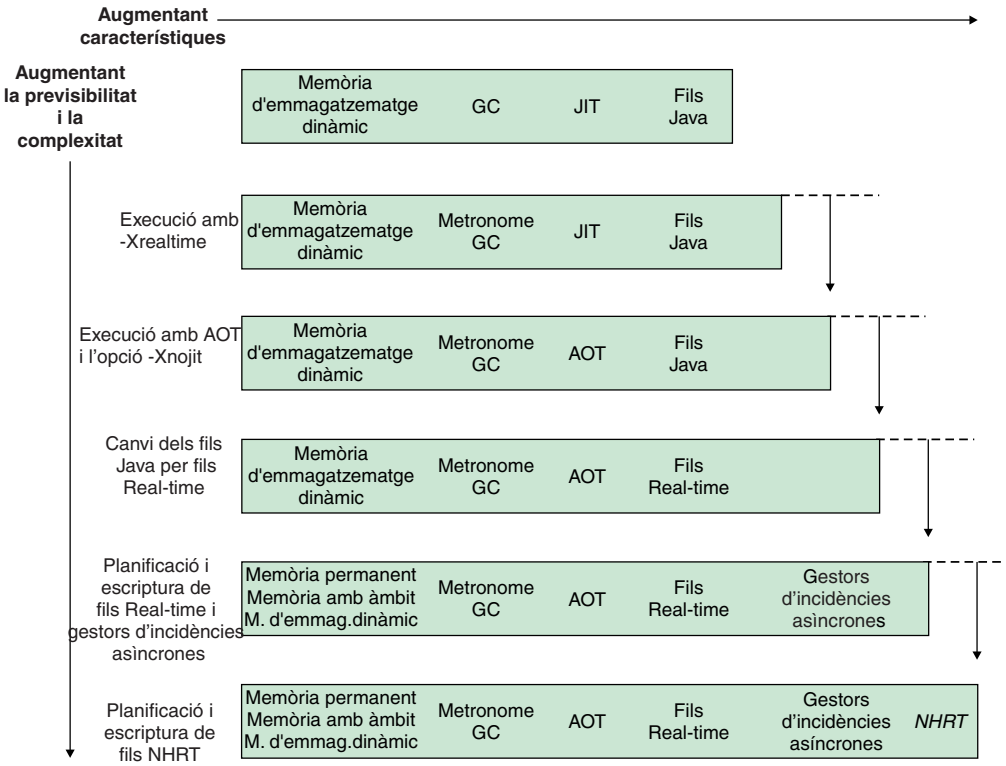

Figura 5. Comparació de les característiques d'RTSJ amb la capacitat de predicció millorada

Planificació de l'aplicació WebSphere Real Time for RT Linux

Quan us prepareu per escriure aplicacions Java en temps real, heu de tenir en compte si utilitzareu fils Java, fils en temps real o fils en temps real que no són d'emmagatzematge dinàmic. A més a més, podeu decidir quina àrea de memòria utilitzaran els fils.

Quant a aquesta tasca

Els passos següents descriuen les decisions que heu de prendre en planificar la vostra aplicació:

Procediment

1. Identifiqueu les vostres tasques.
2. Decidiu els períodes de temps:
   - Per a respostes de més de 10 ms, trieu fils Java, que explotin només el recollidor de deixalles Metronome.

     Aquests fils només utilitzen la memòria d'emmagatzematge dinàmic per a l'emmagatzematge. Presenten l'inconvenient que la recollida de deixalles interromp l'aplicació; això no obstant, com que es controla mitjançant el Recollidor de deixalles Metronome, la longitud i el temps de les interrupcions es poden predir.

- Per a respostes de menys de 10 ms, trieu fils en temps real.

  Els fils en temps real es poden col•locar a la memòria d'emmagatzematge dinàmic, a la memòria amb àmbit o a la memòria immortal. Els avantatges d'utilitzar fils en temps real són:
  - Aquests fils es poden executar en una prioritat més alta que els fils Java estàndard.
  - La recollida de deixalles està sota el control del recollidor de deixalles Metronome. Això no obstant, el recollidor de deixalles s'executa a una prioritat més alta que un fil en temps real i interromp l'execució del programa.
- Per a respostes de menys d'un mil•lisegon, trieu fils en temps real que no són d'emmagatzematge dinàmic.

  La prioritat del fils en temps real que no són d'emmagatzematge dinàmic es pot definir en un valor més alt que la recollida de deixalles i, per tant, Metronome no l'interromp de manera significativa. El fil d'alarma Metronome és l'únic que s'executa al nivell superior de prioritat i que utilitza molt poca quantitat de CPU.

3. Determineu si l'aplicació necessita gestors d'esdeveniments asíncrons. Aquest requisit depèn de l'estructura del programa.
   - Per a un temps de resposta de menys de 10 ms, trieu fils en temps real.
   - Per a un temps de resposta de menys d'un mil•lisegon, trieu fils en temps real que no són d'emmagatzematge dinàmic
4. Determineu les prioritats dels fils. En general, com més curt és el període de temps, més alta és la prioritat.
5. Decidiu les característiques de la memòria.
   - Si una tasca té una variable o un índex d'assignació alt, que pot desbordar la recollida de deixalles, considereu la possibilitat d'imposar un límit d'índex (mitjançant MemoryParameters) o d'assignar un àrea de memòria amb àmbit.
   - Si una tasca genera un volum elevat de dades temporals durant un càlcul, considereu la possibilitat d'utilitzar una àrea de memòria amb àmbit.
   - Si una tasca genera dades durant l'arrencada que són necessàries per al període de vida de la JVM, considereu la possibilitat d'utilitzar memòria immortal. Eviteu l'ús de memòria immortal en casos en què se seguiran creant durant el període de vida de la JVM.
   - Si les tasques s'han de comunicar, especialment si n'hi ha una que s'executa en un fil en temps real que no és d'emmagatzematge dinàmic, considereu la possibilitat d'utilitzar una àrea de memòria amb àmbit per a la comunicació.
   - Si una tasca s'executa en un fil en temps real que no és d'emmagatzematge dinàmic, considereu la possibilitat de crear una àrea de memòria amb àmbit, per exemple, LTMemory, que contingui el fil que no és d'emmagatzematge dinàmic, els paràmetres d'execució, probablement, cues d'espera lliure que s'utilitzen per establir comunicació amb la tasca. L'objecte LTMemory s'ha de crear en una àrea de memòria immortal o en un altre àmbit per evitar errors quan el fil que no és d'emmagatzematge dinàmic prova de fer-hi referència.
6. Modifiqueu les opcions d'execució per millorar el rendiment de l'aplicació, quan hagueu decidit l'estructura i el contingut de l'aplicació. Els passos següents descriuen com fer-ho:
   a. Durant la prova inicial de l'aplicació, definiu quantitats generoses d'espai a l'emmagatzematge dinàmic, i definiu memòria immortal mitjançant les opcions **-Xmx**, **-Xgc:immortalMemorySize=size** i **-Xgc:scopedMemoryMaximumSize=size**.

Nota: Amb el col•lector de deixalles Metronome, els mides inicial i màxima d'emmagatzematge dinàmic han de ser les mateixes, perquè el col•lector de deixalles Metronome no incrementa la mida de l'emmagatzematge dinàmic. L'increment de l'emmagatzematge dinàmic és una operació no determinista.

b. Utilitzeu l'opció **-verbose:gc** per determinar la quantitat de memòria utilitzada.
c. Modifiqueu l'opció **-Xgc:targetUtilization** per permetre prou temps perquè es dugui a terme la recollida de deixalles. El valor per defecte és el 70% i aquest percentatge sol ser adequat per a la majoria de les aplicacions. Assegureu-vos que la freqüència de recollida de deixalles és lleugerament superior a la freqüència d'assignació.
d. Definiu una mida realista per a la memòria de l'emmagatzematge dinàmic mitjançant l'opció **-Xmx**.

Modificació d'aplicacions Java

Per escriure codi que utilitzi les característiques Java en temps real, utilitzeu `javax.realtime.RealtimeThread` per substituir `java.lang.Thread` per als fils.

Abans de començar

Aquest exemple es basa en la classe JavaRadar.java trobada al fitxer `demo/realtime/sample_application.zip`.

Quant a aquesta tasca

El model de programació per als fils en temps real és semblant al de les aplicacions Java estàndard. Això no obstant, aquest mètode força rudimentari d'afegir el fils en temps real als vostres programes no es beneficia completament de les característiques del WebSphere Real Time for RT Linux. Per fer-ho, heu de modificar els fils de manera que tinguin una prioritat associada i considerar també quines àrees de memòria utilitzaran.

Simplement canviant les classes dels vostres fils, obteniu només un lleuger avantatge per a la vostra aplicació perquè la prioritat per defecte del fils en temps real és superior a la dels fils Java estàndard.

Per canviar JavaRadar per un fil RealtimeThread, canvieu la classe que amplia de `Thread` a `RealtimeThread`.

Substitució de `java.lang.Thread` per `javax.realtime.RealtimeThread`

La classe JavaRadar de l'aplicació d'exemple amplia `java.lang.Thread`. Per exemple:

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

Per fer que aquest fil Java sigui un fil en temps real, redefiniu aquesta definició de classe d'aquesta manera:

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

Escriptura de fils en temps real

Fins ara, només heu modificat una aplicació; ara ha arribat el moment d'escriure codi. Podeu escriure aplicacions que utilitzen fils en temps real per beneficiar-vos dels nivells de prioritat en temps real i de les àrees de memòria.

Abans de començar

Aquest exemple es basa en les classes JavaRadar.java, RTJavaRadar.java i RTJavaControlLauncher.java que són al fitxer `demo/realtime/sample_application.zip`.

Aquest exemple mostra com utilitzar la memòria immortal amb el mateix exemple que es descriu a "Modificació d'aplicacions Java" a la pàgina 72.

Quant a aquesta tasca

El model de programació per als fils en temps real és semblant al de les aplicacions Java estàndard.

Els avantatges d'utilitzar fils en temps real són:

- Suport complet per a prioritats de fils de nivell de sistema operatiu als fils en temps real.
- Ús d'àrees de memòria immortal o amb àmbit.
  - Amb la memòria amb àmbit podeu controlar de manera explícita la desassignació de memòria sense que això afecti la recollida de deixalles.
  - Amb els fils en temps real que no són d'emmagatzematge dinàmic, podeu utilitzar la memòria immortal per evitar les pauses de la recollida de deixalles.
  - Els fils en temps real que fan referència a objectes de l'emmagatzematge dinàmic estan subjectes a la recollida de deixalles de la mateixa manera que els fils en temps real que estan emmagatzemats a la memòria de l'emmagatzematge dinàmic.
  - Els fils en temps real que no són de l'emmagatzematge dinàmic no poden fer referència a objectes de la memòria de l'emmagatzematge dinàmic i, com a conseqüència, no es veuen afectats per la recollida de deixalles.

A la Taula 11 a la pàgina 74, les prioritats s'assignen sobre la base que SimulationThread té la prioritat més alta perquè representa esdeveniments externs i no es pot permetre que s'hi anticipi res del programa. El fil RadarThread ha de respondre ràpidament a les ordres ping del controlador. Com més ràpida sigui la resposta, més acurada serà la mesura de l'alçada de la sonda lunar. El fil ListenThread també ha de respondre ràpidament a les ordres del controlador, però ocupa un segon lloc en relació amb RadarThread.

Aquests tres fils són a la memòria amb àmbit perquè la simulació s'executa com a servidor. Després que el servidor hagi executat una simulació, pot sortir de l'àrea de memòria amb àmbit i després tornar a entrar a l'àrea per esperar una altra execució de la simulació. El servidor utilitza memòria amb àmbit per tal de poder restablir-se a sí mateix.

El fil RTJavaRadarthread és el fil que té la prioritat més alta entre els fils del controlador perquè és més sensible al temps atès que utilitza el temps per obtenir l'alçada. És immortal perquè s'executa com a NHRT i el controlador només s'executa una vegada i la memòria s'allibera quan la JVM surt.

Per a RTJavaControlThread i RTJavaEventThread, les limitacions de temps no són tan crítiques i, per tant, l'ús de la memòria d'emmagatzematge dinàmic és acceptable.

El fil RTLoadThread no realitza cap funció útil per a la sonda lunar. Això no obstant, RTLoadThread demostra que es pot dur a terme una assignació i desassignació significativa de memòria a una prioritat més baixa que la resta de fils, sense afectar el rendiment dels fils de prioritat més alta.

Taula 11. Relació entre els fils i les àrees de memòria a l'aplicació d'exemple

Memòria	Fil	Prioritat
Amb àmbit	demo.sim.SimulationThread	38
	demo.sim.RadarThread	37
	demo.sim.SimulationThread.ListenThread	36
Immortal	demo.controller.RTJavaRadarThread	15
Emmagatzematge dinàmic	demo.controller.RTJavaControlThread	14
	demo.controller.RTJavaEventThread	13
Amb àmbit i emmagatzematge dinàmic	demo.controller.RTLoadThread	12

Exemples

Aquest codi del fil demo.sim.SimulationThread mostra on s'ha definit la prioritat 38. **1** Aquesta línia de codi recupera la prioritat màxima que hi ha disponible a la JVM.

```
super(null, area);

// Definim les prioritats per separat, perquè utilitzem "this".
// PriorityScheduler.MAX_PRIORITY ja no s'utilitza.
this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
        .getMaxPriority(this)));     1
```

Aquest codi de demo.sim.SimLauncher mostra on s'ha definit la memòria amb àmbit. **2** mostra l'assignació de memòria LTMemory, que és una àrea de memòria amb àmbit que assigna memòria en temps lineal.

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

/*
 * Cal crear l'objecte LTMemory a l'àrea de memòria a la qual els
 * fils NHRT tenen accés.
 */
ImmortalMemory.instance().enter(new Runnable() {
    public void run() {
        myMemRef.ref = new LTMemory(10000000);  2
    }
});

final MemoryArea simMemArea = myMemRef.ref;
```

L'objecte ScopedMemoryArea al qual fa referència `simMemArea` s'assigna a la memòria immortal, perquè el NHRT ha de poder fer referència a l'objecte que representa ScopedMemoryArea. Si s'assigna a l'emmagatzematge dinàmic, el constructor de fils NHRT genera una excepció IllegalArgumentException, perquè el seu argument d'àrea de memòria no era l'emmagatzematge dinàmic.

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

Aquest codi de demo.controller.RTJavaControlLauncher mostra on s'ha definit la memòria immortal i on ha estat utilitzada per RTJavaRadar. Atès que RTJavaRadar s'executa una vegada durant tota la vida de la JVM del controlador, està dissenyat per assignar memòria només durant l'arrencada; es pot executar de manera segura a la memòria immortal. El disseny de l'aplicació se'n beneficia perquè el controlador pot accedir al mètodes RTJavaRadar sense haver d'entrar primer a l'àrea de la memòria amb àmbit. Entrar a l'àrea de memòria amb àmbit és difícil perquè el controlador ha estat escrit per ésser executat en Java ordinari i en Java en temps real.

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

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

// Crea RTJavaRadar a la memòria immortal, és un NHRT.
// Si fos a la memòria amb àmbit, la seva interacció amb altres fils
// seria més complexa.
ImmortalMemory.instance().enter(new Runnable() {
    public void run() {
        // Versió en temps real del Radar.
        radarRef.ref = new RTJavaRadar(radarPort, ImmortalMemory
                .instance());
    }
});

RTJavaRadar radarJava = radarRef.ref;
```

Escriptura de gestors d'esdeveniments asíncrons

Els gestors d'esdeveniments asíncrons reaccionen als esdeveniments de temporitzador o als esdeveniments que es produeixen fora d'un fil; per exemple, l'entrada d'una interfície d'una aplicació. En sistemes en temps real aquests esdeveniments responen dintre dels límits establerts per a l'aplicació.

Abans de començar

Aquest exemple es basa en les classes RTJavaEventThread.java i RTJavaControlLauncher.java que hi ha al fitxer `demo/realtime/sample_application.zip`.

Quant a aquesta tasca

A l'aplicació d'exemple, el fil d'esdeveniment espera els esdeveniments de la simulació que senyala una aturada (crash) o un aterratge (land). A la versió en temps real d'aquest fil, s'utilitza el mecanisme AsyncEvent. Aquests esdeveniments s'utilitzen per imprimir el missatge d'estat adequat i per fer que el controlador surti.

El fil RTJavaEventThread té dos esdeveniments asíncrons definits. Cap no té paràmetres.

```
public class RTJavaEventThread extends RealtimeThread {

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

Aquests esdeveniments creen i registren dos gestors d'esdeveniments asíncrons:

```
    /**
     * Passar un objecte executable que s'activarà quan es produeixi l'esdeveniment land.
     *
     * Codi executable @param que s'executarà quan s'activi l'esdeveniment land.
     */
```

```
public void addLandHandler(Runnable runnable) {
    AsyncEventHandler handler = new AsyncEventHandler(runnable);
    this.landEvent.addHandler(handler);
}

/**
 * Passar un objecte executable que s'executarà quan es produeixi l'esdeveniment crash.
 *
 * Codi executable @param que s'executarà quan s'activi l'esdeveniment crash.
 */
public void addCrashHandler(Runnable runnable) {
    AsyncEventHandler handler = new AsyncEventHandler(runnable);
    this.crashEvent.addHandler(handler);
}
```

Quan es reben els missatges crash o land, s'activa el gestor d'esdeveniments asíncrons corresponent, fet que fa que s'alliberin els objectes executables (Runnable).

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

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

Resultats

RTJavaControlLauncher.java inclou invocacions als mètodes addLandHandler i addCrashHandler. Els objectes Runnable que es passen provoquen la impressió d'un missatge a la consola i el fil de control s'atura quan s'activen els gestors d'esdeveniments asíncrons associats. Vegeu RTJavaEventThread.java per saber a quin punt s'activen.

```
// Executable AEH per al gestor land.
javaEventThread.addLandHandler(new Runnable() {
    public void run() {
        System.out.println("LAND!");
    }
});

// Executable AEH per al gestor crash.
javaEventThread.addCrashHandler(new Runnable() {
    public void run() {
        System.out.println("CRASH!");
    }
});
```

Escriptura de fils NHRT

Per afegir fils NHRT (fils en temps real que no són d'emmagatzematge dinàmic) a una aplicació Java, utilitzeu aquesta secció per desenvolupar o modificar els vostres programes.

Abans de començar

Aquest exemple es basa en les classes SimulationThread.java i SimLauncher.java que hi ha al fitxer `demo/realtime/sample_application.zip`.

Quant a aquesta tasca

La classe demo.sim.SimulationThread forma part de la simulació a l'aplicació de demostració. Està concebuda per actuar com a substitut per al món real i, per tant, és molt probable que s'executi sense interrupcions de la resta del sistema. El fil es crea com a fil NoHeapRealtimeThread amb la prioritat disponible més alta, per garantir que el fil no s'interromp per causa de la recollida de deixalles ni de cap altre fil del sistema.

A SimulationThread, el constructor següent crida el super constructor "NoHeapRealtimeThread(SchedulingParameters scheduling, MemoryArea area)", abans de definir els seus paràmetres SchedulingParameters i ReleaseParameters per separat:

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

        super(null, area);

        // Definim les prioritats per separat, perquè utilitzem "this".
        // PriorityScheduler.MAX_PRIORITY ja no s'utilitza.
        this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
                .getMaxPriority(this)));

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

        // És una bona pràctica identificar cadascun dels fils.
        this.setName("SimulationThread");

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

Els altres fils actius de la simulació també es creen com a fils NHRT (fils en temps real que no són d'emmagatzematge dinàmic), però amb una prioritat lleugerament inferior. Vegeu "Escriptura de fils en temps real" a la pàgina 72 per veure la disposició de les prioritats.

La simulació té l'opció d'executar-se de manera indefinida, de manera que després de finalitzar una simulació es reinicia. Atès que la simulació està formada per fils NHRT, podeu triar ScopedMemory o ImmortalMemory. L'aplicació d'exemple utilitza ScopedMemory per a la simulació perquè és adequat per sortir de l'àrea ScopeMemoryArea que s'ha assignat en finalitzar la simulació i després tornar a entrar-hi per esperar l'execució següent. En aquest cas, no es porta cap estat d'una execució a la següent.

La majoria de les són segures per als fils NHRT; això no obstant, la majoria de les classes es poden executar d'una manera no segura per als fils NHRT. Per exemple, si els sòcols DatagramSockets es desen a la memòria immortal, o en una àrea de memòria amb àmbit externa, pot ser que es produeixin errors perquè no estan dissenyats per ampliar àrees de memòria. L'aplicació d'exemple utilitza només una àrea de memòria amb àmbit (ScopedMemory) per evitar aquests problemes.

Assignació de memòria a l'RTSJ

A l'RTSJ, podeu assignar un objecte en una àrea de memòria específica de diverses maneres i no sempre és obvi quina d'aquestes maneres triar en un punt determinat.

Cada enfocament té determinades característiques, que varien entre les implementacions de l'RTSJ, i fan que el rendiment o el possible impacte a la memòria siguin diferents. En aquesta secció es descriuen les opcions disponibles i se suggereixen ocasions en què poden ser l'opció més adequada per assignar un objecte.

Inicialitzador estàtic

La manera més senzilla d'assignar un objecte a l'àrea de memòria immortal és assignar-lo en un inicialitzador estàtic. L'avantatge és que no heu d'ocupar-vos de canviar el context de memòria, però les circumstàncies en les quals aquest patró és adequat són força limitades. Aquest enfocament és eficient quant a que la quantitat de memòria immortal consumida es limita a la necessària per a l'objecte.

MemoryArea.newInstance(Class c)

Aquest enfocament és directe si hi ha un fil en un context de memòria i vol assignar un objecte en una altra àrea, que pot ser que ja sigui a la pila d'àmbits del fil. L'avantatge és que només cal accedir a la classe per a la qual es vol crear la instància, però el mètode newInstance ha de crear un constructor adequat. Aquest patró és més adequat si els objectes d'una classe determinada s'han d'assignar molt de tant en tant, però tendeixen a mostrar un ús de memòria alt.

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

Aquest també és un enfocament senzill si un fil és en un context de memòria i vol assignar un objecte en un altre context, que ja ha de ser a la pila d'àmbits del fil. En aquest cas, heu de passar un constructor i alguns arguments i s'assumeix la responsabilitat de garantir que el constructor és vàlid en el context de memòria actual. Com que el mètode newInstance no ha de crear cap constructor, l'ús de memòria és inferior que el de newInstance(Class c) i, per tant, aquest patró és més adequat si els objectes s'han d'assignar més sovint i voleu pagar el preu per assignar el constructor de manera anticipada i emmagatzemar-lo en algun lloc com ara ImmortalMemory.

MemoryArea.enter(Runnable r) seguit per un nou mètode <class>()

Aquest enfocament fa que l'àrea de memòria (MemoryArea) indicada sigui la nova àrea per defecte per a les assignacions, de manera que no cal cap tipus de reflexió ni els objectes constructors assistents. Per tant, és molt més adequat si cal crear molts objectes perquè no es produeix cap ús de memòria addicional per sobre de l'objecte. Aquest enfocament només funciona si l'àrea desitjada encara no està activa a la pila d'àmbits de cap fil. El requisit per crear una àrea de memòria executable (Runnable) fa que aquest enfocament sigui més complex que utilitzar newInstance, perquè normalment s'han de passar els paràmetres a l'àrea executable o a través camps estàtics o d'instància.

MemoryArea.executeInArea(Runnable r) seguit per un nou mètode <class>()

Aquest enfocament també fa que l'àrea de memòria (MemoryArea) indicada sigui la nova àrea per defecte per a les assignacions, de manera que no cal cap tipus de reflexió ni els objectes constructors assistents. Per tant, és molt més adequat si cal crear molts objectes perquè no es produeix cap ús de memòria addicional per sobre de l'objecte. Podeu utilitzar aquest enfocament si l'àrea desitjada ja és a la pila d'àmbits del fil actual i, per tant, és més flexible que MemoryArea.enter. El requisit per crear una àrea de memòria executable (Runnable) fa que aquest enfocament sigui més complex que utilitzar newInstance, perquè normalment s'han de passar els paràmetres a l'àrea executable o a través camps estàtics o d'instància.

Class.newInstance()

Aquest enfocament crea la nova instància a l'àrea de memòria actual i, per tant, s'ha d'utilitzar amb MemoryArea.enter o amb executeInArea. No es produeix cap ús de memòria addicional a banda del de l'objecte.

Utilització del temporitzador d'alta resolució

El rellotge en temps real proporciona més precisió que els rellotges associats amb la JVM estàndard.

Abans de començar

Aquest exemple es basa en la classe RTJavaRadar.java trobada al fitxer `demo/realtime/sample_application.zip`.

Quant a aquesta tasca

El Java normal té una capacitat limitada per utilitzar rellotges i temporitzadors. L'Especificació de temps real per a Java permet que les hores absolutes siguin específiques amb un precisió de nanosegons i amb una magnitud suficient per a l'hora dels rellotges de paret. javax.realtime.HighResolutionTime i les seves subclasses s'utilitzen per representar l'hora amb dos components, mil•lisegons i nanosegons.

El WebSphere Real Time for RT Linux utilitza el suport per al sistema operatiu subjacent per subministrar l'hora d'alta resolució. Els kernels Linux actuals proporcionen un rellotge que, com a màxim garanteix una precisió de 4 mil•lisegons. Els pedaços del Linux subministrats amb el WebSphere Real Time for RT Linux proporcionen un rellotge amb una precisió que s'acosta a 1 microsegon.

La classe RTJavaRadar mostra l'ús del temporitzador d'alta resolució:

- **1** obté el rellotge en temps real.
- **2** obté l'hora absoluta actual.
- **3** obté el component de nanosegons de l'hora. La precisió del rellotge en temps real vol dir que l'ús de nanosegons és raonable.
- **4** obté l'hora abans i després d'executar ping.
- **5** retorna la velocitat de descens de la sonda.
- **6** fa que el fil esperi 5 mil•lisegons abans de fer una altra iteració.

```
public void run() {
       // Els objectes següents es creen de manera anticipada i es reutilitzen en cada
       // iteració.
```

```
        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 lloc d'utilitzar el format time = rtClock.getTime(), aquest
                // mètode
                // substitueix els valors de l'objecte AbsoluteTime preexistent.
                rtClock.getTime(time);                  4
                millis = time.getMilliseconds();
                nanos = time.getNanoseconds();

                // Temps que es triga en enviar el ping i rebre 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) {
                    // Això no és important.
                }
            }
```

El codi anterior es pot comparar amb codi següent de la JVM estàndard a la classe JavaRadar:

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

            long nanos = System.nanoTime(), lastnanos;
            while (this.running) {
                /* Establir l'alçada cada x mil•lisegons */
                Thread.sleep(5);
                lastnanos = nanos;
                lastheight = height;

                nanos = System.nanoTime();

                this.radarPort.ping();

                // L'escala de temps és vuit unitats per mil•lisegon
                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;
    }
```

Aplicació d'exemple

L'aplicació d'exemple utilitza tota una sèrie d'exemples per mostrar les característiques del WebSphere Real Time for RT Linux que es poden utilitzar per millorar les característiques de temps real dels programes Java.

Els fitxers font de l'aplicació d'exemple són al fitxer `demo/realtime/sample_application.zip`.

L'aplicació d'exemple està formada per dos components principals:

- Una **simulació**, un exemple simple de sonda lunar. La seva posició ve definida per la seva alçada per sobre del terra i la seva distància de l'àrea d'aterratge. Consulteu Figura 6 a la pàgina 82.

  La classe de simulació s'escriu utilitzant fils NHRT (fils en temps real que no són d'emmagatzematge dinàmic) i no es modifica més en aquesta documentació.
- Un **controlador** que envia ordres a la simulació. Envia pings al radar per jutjar l'alçada de la sonda i controlar la velocitat de descens de la sonda en base a aquesta informació. El controlador també rep un corrent d'informació de la sonda; per exemple, la distància de la sonda de l'àrea d'aterratge.

  El controlador s'escriu inicialment en Java estàndard. A "Modificació d'aplicacions Java" a la pàgina 72, es desenvolupa en un programa Java en temps real.

En funció del resultat de l'aterratge, s'envien un o dos missatges al controlador: col•lisió o aterratge.

Mitjançant l'aplicació d'exemple, podeu dur a terme aquestes operacions:

- Executar la simulació i el controlador alhora per mostrar una combinació de classes Java en temps real i estàndard que s'executen simultàniament. Per obtenir-ne més informació, vegeu "Creació de l'aplicació d'exemple" a la pàgina 83 i "Execució de l'aplicació d'exemple" a la pàgina 83, on també veureu la sortida que podeu esperar de l'aplicació d'exemple.

  Nota: Podeu iniciar la simulació i el controlador alhora mitjançant la classe LaunchBoth.
- Compareu la diferència entre l'ús del Recollidor de deixalles Metronome i el recollidor de deixalles estàndard. Per obtenir informació, vegeu "Execució de l'aplicació d'exemple sense Real Time" a la pàgina 83 i "Execució de l'aplicació d'exemple amb el recollidor de deixalles Metronome" a la pàgina 85.
- Executeu l'aplicació utilitzant el compilador anticipat (AOT). Per obtenir més informació, vegeu "Execució de l'aplicació de mostra mitjançant AOT" a la pàgina 86.

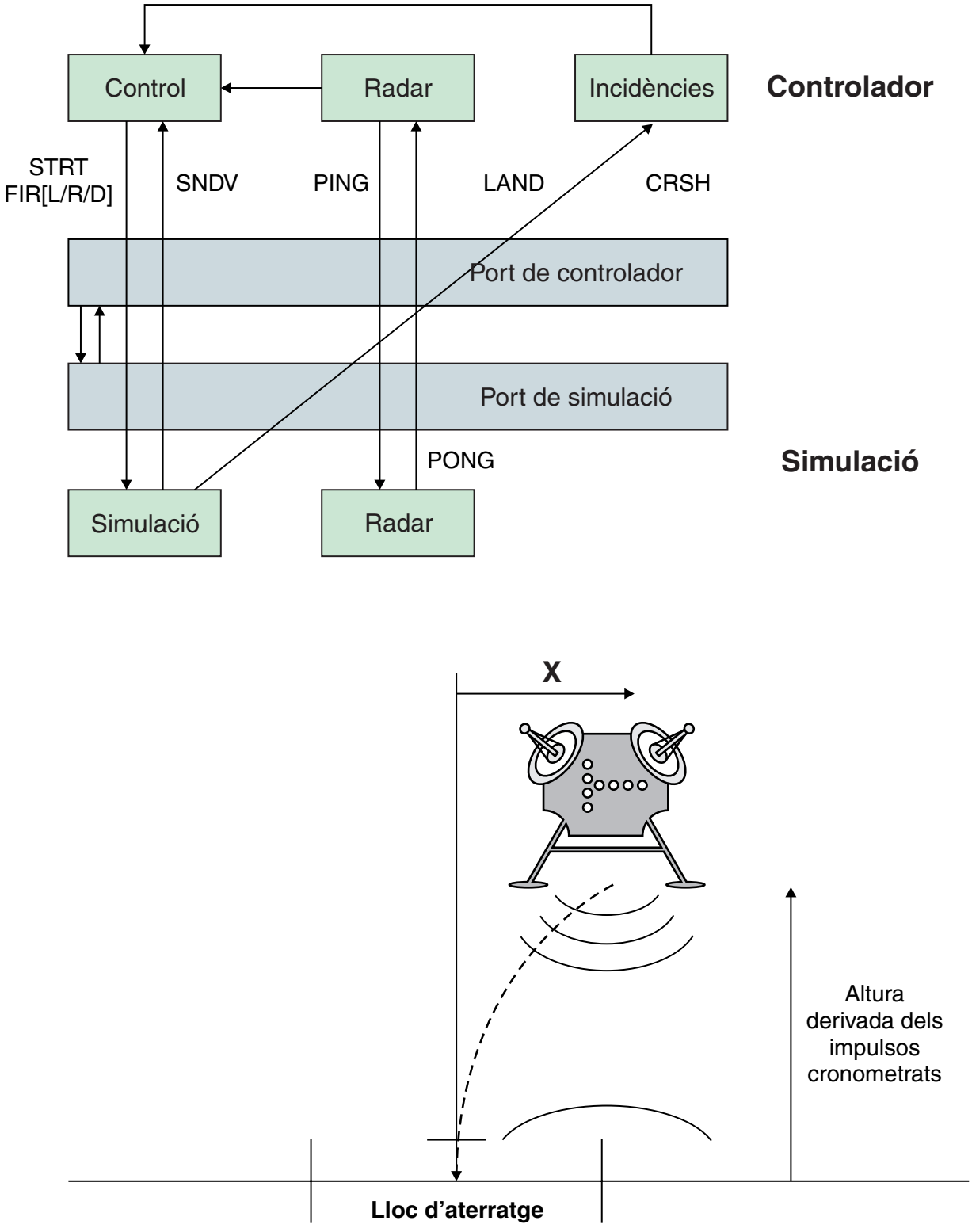

Figura 6. Diagrama de la sonda lunar

Aquest diagrama mostra la relació entre els mòduls proporcionats a l'exemple. La part superior del diagrama mostra el controlador i el simulador. El controlador té tres fils: fils de control, fils de radar i fils d'esdeveniments. El simulador té dos fils:

el fil de simulador i el fil de radar. La part inferior del diagrama mostra la sonda lunar i indica les dues funcions de control d'esquerra i dreta amb els polsos que determinen l'alçada de la sonda.

Creació de l'aplicació d'exemple

El codi font de l'aplicació d'exemple es proporciona com a guia. Per a la fase de preparació cal desempaquetar i compilar el codi font Java abans de poder executar-lo.

Procediment

1. Creeu un directori de treball.
2. Extraieu l'aplicació d'exemple al vostre directori de treball:
   ```
   unzip sample_application.zip
   ```
3. Creeu un directori nou per a la sortida:
   ```
   mkdir classes
   ```
4. Compileu l'origen.
   a. Genereu una llista dels fitxers:
      ```
      find -name "*.java" > source
      ```
   b. Compileu l'origen:
      ```
      javac -Xrealtime -Xlint:deprecated -g -d classes @source
      ```
   c. Creeu un fitxer jar dels fitxers de classes:
      ```
      jar cf demo.jar -C classes/ .
      ```

Què cal fer posteriorment

Ara podeu executar l'aplicació d'exemple.

Execució de l'aplicació d'exemple

El WebSphere Real Time proporciona una JVM estàndard i una JVM en temps real, que s'inicia amb l'argument **-Xrealtime** de la línia d'ordres.

L'aplicació de mostra té dos components, designats per ser executats en JVM diferents:

- La simulació, que només s'executa en Java en temps real.
- El controlador, que es pot executar en Java en temps real i també que no sigui en temps real.

L'execució del codi de controlador en diferents modes permet veure els avantatges de la tecnologia IBM Real-Time Java.

Execució de l'aplicació d'exemple sense Real Time

En aquest procediment, executareu l'aplicació d'exemple sense beneficiar-vos de l'IBM WebSphere Real Time.

Abans de començar

Per executar l'aplicació d'exemple, primer cal crear el codi font d'exemple. Vegeu “Creació de l'aplicació d'exemple” per obtenir-ne més informació.

Procediment

1. Inicieu la simulació:

```
java -Xrealtime -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m
demo.sim.SimLauncher <port>
```

En aquesta ordre, *<port>* és un port no assignat per a l'estació de treball.

2. Inicieu el controlador:

```
java -classpath ./demo.jar -mx300m
demo.controller.JavaControlLauncher <amfitrió>
<port>
```

En aquesta ordre, *<amfitrió>* és el nom d'amfitrió de l'estació de treball que executa la simulació i *<port>* és el port especificat al pas anterior.

Resultats

L'aplicació genera un missatge que mostra que la simulació i el controlador s'han iniciat:

```
SimLauncher: Waiting for connections...
Starting control thread...
```

A la consola es mostren alguns exemples de punt dels valors del controlador:

```
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
```

Tot just abans d'aturar la simulació, s'emet un missatge de resum de l'esdeveniment:

```
Fire down transitions 141, fire horizontally transitions 141
LAND!
```

A més dels exemples de punt i del missatge de resum de l'esdeveniment, el controlador produeix un gràfic anomenat `graph.svg` al mateix directori. El gràfic conté una representació dels exemples de punts. A la gràfica es mostra l'efecte de les pauses de la recollida de deixalles sobre el fil JavaRadar quan s'executa l'aplicació amb una JVM que no és Real Time estàndard. Les dades que representen l'alçada del radar mostren puntes. Les puntes són degudes a les pauses de la recollida de deixalles estàndard que afecten l'aplicació Controller. En algunes execucions, les pauses de la recollida de deixalles són prou llargues per crear errors, la qual cosa dóna lloc a aquest missatge:

```
CRASH!
```

Per veure els temps de pausa causats per la recollida de deixalles, afegiu l'opció **`-verbose:gc`** a l'ordre d'inici del controlador:

```
java -classpath ./demo.jar -verbose:gc -mx300m
demo.controller.JavaControlLauncher <amfitrió>
<port>
```

Execució de l'aplicació d'exemple amb el recollidor de deixalles Metronome

Podeu executar una aplicació Java estàndard en un entorn en temps real sense haver de rescriure el codi, afegint l'opció -Xrealtime. L'opció habilita les característiques del llenguatge Java en temps real i el recollidor de deixalles Metronome.

Abans de començar

Per executar l'aplicació d'exemple, primer cal crear el codi font d'exemple. Vegeu "Creació de l'aplicació d'exemple" a la pàgina 83 per obtenir-ne més informació.

Procediment

1. Inicieu la simulació:

   ```
   java -Xrealtime -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m
   demo.sim.SimLauncher <port>
   ```

   En aquesta ordre, *<port>* és un port no assignat a l'estació de treball.
2. Inicieu el controlador:

   ```
   java -Xrealtime -classpath ./demo.jar -mx300m
   demo.controller.JavaControlLauncher <amfitrió> <port>
   ```

   En aquesta ordre, *<amfitrió>* és el nom d'amfitrió de l'estació de treball que executa la simulació i *<port>* és el port especificat al pas anterior. L'execució de totes dues JVM a la mateixa estació de treball pot fer que el comportament sigui menys determinista. Consulteu "Consideracions" a la pàgina 24 per obtenir-ne més informació.

Resultats

L'aplicació s'executa i genera diverses sortides, com ara:

1. Missatges que mostren que la simulació i el controlador s'han iniciat.
2. Exemples de punt dels valors del controlador.
3. Un gràfic anomenat `graph.svg` al mateix directori, que conté una representació dels exemples de punts.
4. Un missatge de resum de l'esdeveniment.

En executar l'aplicació amb la recollida de deixalles Metronome, els exemples de punt i el gràfic corresponent normalment:

- No mostren puntes a les dades d'alçada del radar.
- Mostren un seguiment acurat de les dades d'alçada real.

El motiu és que el codi del controlador ara s'executa només amb pauses més curtes per a la recollida de deixalles.

Les pauses de la recollida de deixalles Metronome són freqüents, però normalment duren menys d'un mil•lisegon. Les pauses de la recollida de deixalles que no és en temps real són menys nombroses, però normalment duren desenes o centenars de mil•lisegons. La diferència entre les pauses es pot veure afegint l'opció **-verbose:gc** a l'ordre d'execució de Controller.

Vegeu "Utilització de la informació verbose:gc" a la pàgina 135 per obtenir més informació sobre la sortida de recollida de deixalles detallada.

Execució de l'aplicació de mostra mitjançant AOT

Aquest procediment executa una aplicació Java estàndard en un entorn en temps real utilitzant el compilador anticipat (AOT) sense haver d'escriure codi. Utilitzeu aquest exemple per comparar l'execució de la mateixa aplicació mitjançant el compilador JIT.

Vegeu "Ús del codi compilat amb l' WebSphere Real Time for RT Linux" a la pàgina 36 per obtenir més detalls sobre la compilació anticipada.

Abans de començar

Per executar l'aplicació d'exemple, primer cal crear el codi font d'exemple. Vegeu "Creació de l'aplicació d'exemple" a la pàgina 83 per obtenir-ne més informació.

Quant a aquesta tasca

El compilador AOT compila l'aplicació Java en codi natiu abans d'executar-la. Podeu predir de manera més acurada com s'executa l'aplicació, perquè ni hi ha cap interrupció provocada per la compilació JIT.

Procediment

1. Convertiu els codis de bytes de l'aplicació en codi natiu.
   a. La conversió es realitza primer executant l'exemple amb el compilador JIT normal.

      ```
      java -Xrealtime -Xjit:verbose={precompile},vlog=./sim.aotOpts \
           -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m
           demo.sim.SimLauncher <port>
      ```

      En aquesta ordre, *<port>* és un port no assignat per a l'estació de treball.
   b. En una altra finestra, executeu l'aplicació.

      ```
      java -Xrealtime -Xjit:verbose={precompile},vlog=./control.aotOpts \
           -classpath ./demo.jar -Xmx300m demo.controller.JavaControlLauncher localhost <port>
      ```

      En aquesta ordre, *<port>* és el port especificat al pas anterior. Els resultats de la sortida de l'aplicació són semblants a aquests missatges:

      ```
      Fire down transitions 141, fire horizontally transitions 141
      ```

      I:

      ```
      Land!
      ```
   c. Combineu els fitxers d'opcions AOT creats als passos anteriors.

      ```
      cat sim.aotOpts.20081014.234958.13205 control.aotOpts.20081014.234958.13205 > sample.aotOpts
      ```

      Els noms utilitzats per als fitxers de registre creats als passos anterior tenen la informació de data i ID de procés afegida al nom de fitxer. El format del nom de fitxer s'especifica a l'opció **vlog=**. Per exemple, **vlog=sim.aotOpts** genera un nom de fitxer semblant a **sim.aotOpts.20081014.234958.13205**:
   d. Compileu els fitxers del fitxer `sample.aotOpts` a `realtime.jar`, `vm.jar`, `rt.jar` i l'aplicació `demo.jar`. Quan utilitzeu memòries cau de classes compartides, el nom de la memòria cau no pot tenir més de 53 caràcters.

      ```
      admincache -Xrealtime -populate -cacheName "sample" -aotFilterFile sample.aotOpts -classpath
      $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
      ```

Es notifiquen els resultats de la compilació:

```
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 línia:

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

indica que les memòries cau existents anomenades "sample" es destrueixen amb aquesta ordre per crear la memòria cau especificada.

e. Visualitzeu el contingut de la memòria cau emplenada.

```
admincache -Xrealtime -cacheName "sample" -printStats
```

2. Inicieu la simulació:

```
java -Xrealtime -Xnojit -Xmx300m -Xshareclasses:name="sample" \
     -classpath ./demo.jar -Xgc:scopedMemoryMaximumSize=11m \
     demo.sim.SimLauncher <port>
```

En aquesta ordre, *<port>* és un port no assignat per a aquesta estació de treball.

3. Inicieu el controlador:

```
java -Xrealtime -Xnojit -Xmx300m -Xshareclasses:name="sample" \
     -classpath ./demo.jar \
     demo.controller.JavaControlLauncher <host> <port>
```

En aquesta ordre, *<amfitrió>* és el nom d'amfitrió de l'estació de treball que executa la simulació i *<port>* és el port especificat al pas anterior. L'execució de totes dues JVM a la mateixa estació de treball pot fer que el comportament sigui menys determinista. Consulteu “Consideracions” a la pàgina 24 per obtenir-ne més informació.

Resultats

L'aplicació s'executa i genera diverses sortides, com ara:

1. Missatges que mostren que la simulació i el controlador s'han iniciat.
2. Exemples de punt dels valors del controlador.
3. Un gràfic anomenat `graph.svg` al mateix directori, que conté una representació dels exemples de punts.
4. Un missatge de resum de l'esdeveniment.

En executar l'aplicació amb la compilació anticipada, els exemples de punt i el gràfic corresponent normalment:

- No mostren puntes a les dades d'alçada del radar.
- Mostren un seguiment acurat de les dades d'alçada real.

El motiu és que el codi del controlador ara s'executa només amb pauses més curtes per a la recollida de deixalles i amb cap interrupció per a la compilació a temps (JIT).

Un avantatge d'utilitzar la memòria cau de classes compartida per executar aquesta aplicació és que el controlador i les JVM de simulació comparteixen part de la memòria utilitzada per les classes carregades per totes dues JVM.

Mapa hash en temps real d'exemple

El WebSphere Real Time for RT Linux inclou les implementacions HashMap i HashSet que proporcionen un rendiment més coherent per al mètode put que el HashMap estàndard de l'IBM SDK per a Java 7.

El mapa java.util.HashMap estàndard que IBM proporciona funciona bé per a les aplicacions d'alt rendiment. També ajuda en les aplicacions que la mida màxima a la qual ha de créixer el seu mapa hash. Per a les aplicacions que necessiten un mapa hash que podria créixer a mides variables, en funció de l'ús, hi ha un possible problema de rendiment amb el mapa hash estàndard. El mapa hash estàndard proporciona bons temps de resposta per afegir noves entrades al mapa hash mitjançant el mètode put. Això no obstant, quan el mapa hash s'emplena, cal assignar un magatzem de suport més gran. Això vol dir que cal migrar les entrades del magatzem de suport actual. Si el mapa hash és gran, el temps per realitzar una operació put també podria ser llarg. Per exemple, l'operació podria trigar diversos mil•lisegons.

El WebSphere Real Time for RT Linux inclou un mapa hash en temps real. Proporciona la mateixa interfície funcional que el mapa java.util.HashMap estàndard, però permet un rendiment molt més coherent per al mètode put. En comptes de crear un magatzem de suport i migrar-hi totes les entrades quan s'emplena el mapa hash, el mapa hash d'exemple crea un magatzem de suport addicional. El nou magatzem de suport s'encadena amb la resta de magatzems de suport del mapa hash. Inicialment, la cadena causa una lleugera reducció del rendiment mentre el magatzem de suport buit s'assigna i encadena a la resta de magatzems de suport. Una vegada que s'ha actualitzat el mapa hash de suport, és més ràpid que haver d emigrar totes les entrades. Un dels inconvenients del mapa hash en temps real és que les operacions get, put i remove són lleugerament més lentes. Les operacions són més lentes perquè cada cerca ha de passar per un conjunt de mapes hash de suport en lloc de passar només per un.

Per provar el mapa hash en temps real, afegiu el fitxer `RTHashMap.jar` a l'inici del camí d'accés de classes d'arrencada. Si heu instal•lat el WebSphere Real Time for RT Linux al directori `$WRT_ROOT`, afegiu l'opció següent per utilitzar el mapa hash en temps real amb la vostra aplicació, en comptes d'utilitzar el mapa hash estàndard:

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

Els fitxers font i de classes per a la implementació del mapa hash en temps real s'inclouen al fitxer `demo/realtime/RTHashMap.jar`. A més, també es proporciona una implementació de java.util.LinkedHashMap i java.util.HashSet.

Desenvolupament d'aplicacions WebSphere Real Time for RT Linux mitjançant Eclipse

L'ús d'Eclipse us proporciona un IDE complet quan desenvolupeu les vostres aplicacions en temps real.

Abans de començar

Si és la primera vegada que heu utilitzat l'entorn de desenvolupament d'aplicacions Eclipse per desenvolupar aplicacions en temps real, utilitzeu aquest procediment per configurar el vostre entorn.

El WebSphere Real Time for RT Linux proporciona el compilador **javac** estàndard d'Oracle. No hi ha cap restricció quant al compilador que es pot utilitzar, però ha de produir fitxers de classes Java 5.0 vàlids. Això no obstant, les classes javax.realtime.* Java han de ser al camí d'accés de muntatge.

Quant a aquesta tasca

Per desenvolupar les vostres aplicacions a Eclipse, seguiu aquestes instruccions:

Procediment

1. Baixeu-vos Eclipse des d'aquesta adreça: http://www.eclipse.org/downloads/. Es recomana utilitzar Eclipse 3.1.2 per a una correcta compilació Java 5.0.
2. Baixeu-vos la JVM compatible amb l'IBM SDK and Runtime Environment per a plataformes Linux, Java 2 Technology Edition, versió 5.0 per executar Eclipse.
3. Extraieu el fitxer `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/realtime.jar` del paquet WebSphere Real Time for RT Linux.
4. Obriu Eclipse i creeu un projecte. Feu clic a **Fitxer** > **Nou**. Seleccioneu **Projecte Java** al panell **Projecte nou**.
5. Feu clic a **Següent** per mostrar el panell **Projecte Java nou**.
   a. Introduïu un nom de projecte, per exemple, RTSJ-Tests.
   b. Comproveu que el compilador JDK tingui el valor 5.0.
6. Feu clic a **Finalitza**.
7. Creeu un directori de treball i importeu el fitxer `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/realtime.jar`.
8. Feu clic a **Fitxer** > **Nou** > **Carpeta** per obrir el panell **Carpeta nova**. Introduïu el nom de la carpeta nova, per exemple, *deplib*.
9. Feu clic a **Finalitza**.
10. Per importar el vostre fitxer `realtime.jar`, feu clic a **Fitxer** > **Importa** per obrir el panell **Importa**.
11. Feu clic a **Sistema de fitxers** i a **Següent**.
12. Obriu el directori `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/` del sistema de fitxers en el qual heu desempaquetat la JVM.
13. Marqueu el quadre de selecció del costat del fitxer realtime.jar, indiqueu la carpeta on voleu importar, per exemple `RTSJ-Tests/deplib`, i assegureu-vos que l'opció **Crea només les carpetes seleccionades** estigui seleccionada.
14. Feu clic a **Finalitza**.
15. Afegiu el fitxer jar al camí d'accés de la biblioteca. Feu clic amb el botó dret al vostre projecte i seleccioneu **Propietats** per obrir el panell **Propietats**.

16. Feu clic a **Camí d'accés de muntatge Java** i a la pestanya **Biblioteques**. Feu clic a **Afegeix Jars**.
17. Feu clic a **realtime.jar** dins del directori del projecte. Feu clic a **D'acord**.

Resultats

Si el procediment és correcte, el fitxer realtime.jar apareixerà a la llista de fitxers .jar de la pestanya **Biblioteques**.

Exemple

L'Eclipse pot utilitzar el fitxer `realtime.src.jar` per presentar informació addicional sobre les classes RTSJ. Per fer-ho, obriu la finestra de propietats del fitxer `realtime.jar` importat, feu clic a **Adjunció font Java** i introduïu a **Camí d'accés d'ubicació** la ubicació del fitxer `realtime.src.jar`.

Què cal fer posteriorment

Si voleu muntar aplicacions mitjançant l'Apache Ant amb l'Eclipse, afegiu el fitxer `realtime.jar` al camí d'accés de classes de l'script de muntatge Ant. Per exemple:

```
<property name="rtsj.src" location="." />
<property name="rtsj.deplib" location="deplib" />
<property name="rtsj.jar.dir" location="build/rtsj-jar.dir" />

<!-- Generar fitxers .class per a aquest paquet -->
<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>
```

Això és només una part d'un script de muntatge ant.

Depuració de les aplicacions

Mitjançant el desenvolupador d'aplicacions Eclipse podeu depurar les aplicacions de manera local o remota.

Quant a aquesta tasca

Per depurar la vostra aplicació en temps real de manera remota, la JVM que es depura necessita aquesta opció.
`-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=10100`

Procediment

1. A l'entorn Linux en el que s'executa l'aplicació, introduïu:

   ```
   java -Xrealtime -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=10100
   ```

   on:
   - `server=y` indica que la JVM accepta connexions dels depuradors.
   - `suspend=y` fa que la JVM esperi que el depurador es connecti abans d'executar-se.
   - `address=10100` és el número a través del qual el depurador s'hauria de connectar a la JVM. Aquest número hauria de ser superior a 1024.

   La JVM mostra aquest missatge:

   ```
   Listening for transport dt_socket at address: 10100
   ```

2. Obriu l'aplicació a l'Eclipse i seleccioneu **Depura**.
3. S'hauria de crear una nova configuració per depurar aplicacions remotes. Només n'heu de crear una si s'executa una aplicació del mateix projecte, i escolta en e mateix port per a cada execució.
4. Quan hàgiu creat la configuració, empleneu el nom de les configuracions, el nom del projecte que conté l'aplicació que esteu depurant, el *nom d'amfitrió* de l'estació de treball en la qual s'executa l'aplicació i el número de port que heu passat a les opcions **-agentlib**.
5. Feu clic a **Depura** per iniciar la sessió de depuració. La perspectiva **Depuració** ha d'estar oberta per poder veure l'estat de la JVM depurada de manera remota.

Execució d'Eclipse amb la JVM

En aquesta secció es descriu com executar l'Eclipse amb la JVM del WebSphere Real Time for RT Linux.

Per executar l'Eclipse amb la JVM, heu d'especificar a l'ordre "eclipse":

- El directori complet de l'executable java de la JVM del WebSphere Real Time for RT Linux que voleu utilitzar.
- L'opció **-Xrealtime** de la JVM.
- La mida de la memòria immortal que voleu que utilitzi l'Eclipse. Ha de ser com a mínim de 128 M.

Exemple d'execució d'Eclipse amb la JVM:

```
eclipse -vm $JAVA_HOME/jre/bin/java -vmargs -Xrealtime -Xgc:immortalMemorySize=128M
```

Nota: L'SDK d'Eclipse no es beneficia de les diferents opcions de memòria en temps real que hi ha disponibles a les aplicacions del WebSphere Real Time for RT Linux i, com a conseqüència, la memòria immortal s'exhaureix, especialment en casos en que l'Eclipse s'utilitza durant moltes hores o dies sense reiniciar-lo. Si es produeix un error **OutOfMemory**, podeu incrementar el valor a l'opció **-Xgc:immortalMemorySize** per incrementar la quantitat de memòria immortal que voleu que l'Eclipse utilitzi.

Capítol 7. Rendiment

El WebSphere Real Time for RT Linux està optimitzat per a pauses curtes i consistents de la recollida de deixalles, més que no pas per a la mitjana de rendiment més alta o l'impacte de memòria més insignificant.

En sistemes en què s'admet la tecnologia HT (Hyper Threading), heu d'assegurar-vos que no estigui habilitada. El motiu és evitar efectes adversos de rendiment quan s'utilitza el WebSphere Real Time for RT Linux.

Per tal de poder reduir la variabilitat de temps i el suport per a l'RTSJ (Especificació de temps real per a Java), calia inhabilitar algunes optimitzacions de temps d'execució estàndard d'IBM Java. Per tant, és probable que es percebi una reducció en el rendiment general quan s'executa una aplicació Java estàndard amb el paràmetre **-Xrealtime**.

Rendiment en configuracions de maquinari certificades

Els sistemes certificats tenen una granularitat de rellotge i una velocitat de processador suficients per fer possibles els objectius de rendiment del WebSphere Real Time for RT Linux. Per exemple, una aplicació ben escrita que s'executi en un sistema que no estigui sobrecarregat, i amb una mida d'emmagatzematge dinàmic adequada, generalment experimentaria temps de pausa de recollida de deixalles que són molt per sota d'1 mil•lisegon, normalment 500 microsegons. Durant els cicles de recollida de deixalles, una aplicació amb els valors d'entorn per defecte no s'atura durant més del 30% del temps transcorregut durant cap marge de temps variable de 10 mil•lisegons. El temps col•lectiu dedicat a les pauses de recollida de deixalles per sobre d'un període de 10 mil•lisegons normalment suma menys de 3 mil•lisegons.

Reducció de la variabilitat de temps

Les dues fonts principals de variabilitat en una JVM estàndard es gestionen al WebSphere Real Time for RT Linux d'aquesta manera:

- Preparació del codi Java: la càrrega i la compilació a temps (JIT) es gestionen mitjançant la compilació anticipada (AOT). Consulteu "Utilització del compilador AOT" a la pàgina 38.
- Pauses de recollida de deixalles: les pauses possiblement llargues dels modes de recollida de deixalles estàndard s'eviten mitjançant el Recollidor de deixalles Metronome. Consulteu "Utilització del recollidor de deixalles Metronome" a la pàgina 66.

Compartició de dades de classe entre les JVM per al mode en temps no real

La compartició de classes té suport en mode no de temps real (Real-Time), però funciona de manera diferent que en mode de temps real.

Podeu compartir dades de classe entre màquines JVM (Java Virtual Machine) emmagatzemant-les en una memòria cau de la memòria compartida. La compartició redueix el consum d'emmagatzematge virtual global quan més d'una JVM comparteix una memòria cau. La compartició també redueix el temps d'inici

d'una JVM un cop s'ha creat la memòria cau. La memòria cau de classes compartida és independent de qualsevol JVM en execució i persisteix fins que es suprimeix.

Una memòria cau compartida pot contenir:

- Classes bootstrap
- Classes d'aplicació
- Metadades que descriuen les classes
- Codi compilat de manera anticipada (AOT)

Capítol 8. Seguretat

Aquest apartat conté informació important sobre la seguretat.

Consideracions sobre seguretat per a la memòria cau de classes compartida

La memòria cau de classes compartida està dissenyada per facilitar la gestió i l'ús de la memòria cau, però pot ser que la política de seguretat per defecte no sigui adequada.

Quan utilitzeu la memòria cau de classes compartida, heu de tenir presents els permisos per defecte per als fitxers nous de manera que pugueu millorar la seguretat mitjançant la restricció d'accés.

Fitxer	**Permisos per defecte**
Memòries cau compartides noves	Permisos de lectura per a grup i altres
Directori `javasharedresources`	Permís de lectura, escriptura i execució

Cal tenir permís d'escriptura al fitxer de la memòria cau i al directori de la memòria cau per poder destruir o incrementar una memòria cau.

Canvi dels permisos de fitxer al fitxer de la memòria cau

Per limitar l'accés a una memòria cau de classes compartida, podeu utilitzar l'ordre **`chmod`**.

Canvi necessari	**Ordre**
Limitar l'accés a l'usuari i grup	`chmod 770 /tmp/javasharedresources`
Limitar l'accés a l'usuari	`chmod 700 /tmp/javasharedresources`
Limitar l'usuari a accés de lectura i escriptura per a una memòria cau determinada	`chmod 600 /tmp/javasharedresources/` `<`*`fitxer per a memòria cau compartida`*`>`
Limitar l'usuari i grup a accés de lectura i escriptura per a una memòria cau determinada	`chmod 660 /tmp/javasharedresources/` `<`*`fitxer per a memòria cau compartida`*`>`

Vegeu “Creació d'una memòria cau de classes compartida en temps real” a la pàgina 40 per obtenir més informació sobre la creació d'una memòria cau de classes compartida.

Connexió a una memòria cau per a la qual no teniu permís d'accés

Si proveu de connectar-vos a una memòria cau per a la quan o teniu els permisos d'accés adequats, obtindreu un missatge d'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.
```

Capítol 9. Resolució de problemes i suport

Resolució de problemes i suport per al WebSphere Real Time for RT Linux

Mètodes de determinació de problemes generals

La determinació de problemes us ajuda a entendre el tipus d'error que teniu i l'acció correctora apropiada que cal seguir.

Quan sabeu el tipus de problema que teniu, podeu fer una o més de les tasques següents:

- Corregir el problema.
- Trobar una bona solució temporal.
- Recollir les dades necessàries amb les quals cal generar un informe d'errors per a IBM.

Determinació de problemes del Linux

En aquest apartat es descriu la determinació de problemes al Linux.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil per diagnosticar problemes del Linux, com per exemple:

- Configuració i comprovació de l'entorn Linux
- Tècniques de depuració generals
- Diagnòstic de bloqueigs
- Depuració de bloqueigs
- Depuració de les fuites de memòria
- Depuració de problemes de rendiment

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Determinació de problemes del Linux.

La informació següent és complementària per a l'IBM WebSphere Real Time for RT Linux

Configuració i comprovació de l'entorn Linux

A l'IBM WebSphere Real Time for RT Linux, comproveu que la JVM està configurada correctament per generar un abocament de memòria del sistema.

Abocaments de memòria del sistema del Linux (fitxers core)

Quan es produeix una fallada, les dades de diagnòstic més importants que cal obtenir és l'abocament de memòria del sistema del Linux (fitxer core). Per garantir que aquest fitxer es generi, cal que comproveu els paràmetres del sistema operatiu i l'espai de disc disponible, tal com es descriu a la guia de l'usuari de l'IBM SDK for Java 7.

Paràmetres de la màquina virtual Java

La JVM ha d'estar configurada per generar fitxers core quan es produeix una fallada. Executeu `java -Xrealtime -Xdump:what` a la línia d'ordres. La sortida de l'opció és:

```
-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
```

Els valors que es mostren són els paràmetres per defecte. Com a mínim, cal definir `events=gpf` per generar un fitxer core quan es produeix una fallada. Podeu canviar i definir opcions amb l'opció de línia d'ordres **`-Xdump:system[:name1=value1,name2=value2 ...]`**

Tècniques de depuració generals

Ja que els noms de fils Java són visibles al sistema operatiu, podeu utilitzar l'ordre ps per facilitar la depuració. Quan utilitzeu eines de traça, heu d'utilitzar les ordres correctes per a l'IBM WebSphere Real Time for RT Linux.

Examen de la informació del procés

La sortida que veureu en executar l'ordre ps a l'IBM WebSphere Real Time for RT Linux és:

```
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 tots els processos.

L Mostra fils.

o Proporciona un format predefinit de columnes per visualitzar. Les columnes especificades són l'identificador de procés, l'identificador de fil, la política de planificació, la prioritat del fil en temps real i l'ordre

associada amb el procés. Aquesta informació és útil per entendre quins fils de la vostra aplicació, així com la màquina virtual, s'executen en un moment determinat.

Eines de traça

Tres eines de traça al Linux són **strace**, **ltrace** i **mtrace**. L'ordre `man strace` visualitza un conjunt complet d'opcions disponibles.

strace
L'eina strace traça les crides del sistema. Podeu utilitzar-la en un procés que ja estigui disponible, o iniciar-la amb un procés nou. L'eina strace enregistra les crides del sistema fetes per un programa, i els senyals rebuts per un procés. Per a cada crida del sistema, es fan servir el nom, els arguments i el valor de retorn. L'eina strace us permet traçar un programa sense que calgui el codi font (no es requereix cap recompilació). Si utilitzeu l'eina strace amb l'opció **-f**, traçarà els processos secundaris creats com a resultat d'una crida del sistema bifurcada. Podeu utilitzar l'eina strace per investigar problemes de connector o per intentar entendre els motius pels quals un programa no s'inicia adequadament.

Per utilitzar l'eina strace amb una aplicació Java, escriviu `strace java -Xrealtime <nom-classe>`.

Podeu dirigir el resultat de la traça des de l'eina strace a un fitxer utilitzant l'opció `-o`.

ltrace
L'eina ltrace depèn de la distribució. És molt semblant a l'eina strace. Aquesta eina intercepta i enregistra les crides de biblioteca dinàmica a mesura que les crida el procés d'execució. L'eina strace fa el mateix per als senyals rebuts per procés d'execució.

Per utilitzar l'eina ltrace amb una aplicació Java, escriviu `ltrace java -Xrealtime <nom-classe>`.

mtrace
L'eina mtrace s'inclou al conjunt d'eines GNU. Instal•la gestors especials per a malloc, realloc i free, i permet que tots els usos d'aquestes funcions es tracin i s'enregistrin en un fitxer. Aquesta traça disminueix l'eficàcia d'un programa i no s'hauria d'habilitar durant l'ús normal. Per utilitzar l'eina mtrace, definiu **IBM_MALLOCTRACE** a 1 i definiu **MALLOC_TRACE** per apuntar a un fitxer vàlid en el qual s'emmagatzemi la informació de traça. Heu de tenir permís d'escriptura per accedir a aquest fitxer.

Per utilitzar l'eina mtrace amb una aplicació Java, escriviu:

```
export IBM_MALLOCTRACE=1
export MALLOC_TRACE=/tmp/file
java -Xrealtime <nom-classe>
mtrace /tmp/file
```

Diagnòstic de bloqueigs

Quan recolliu informació sobre els processos en execució i sobre l'entorn Java abans d'una fallada, seguiu aquestes indicacions.

Recollida d'informació del procés

Quan feu la recerca sobre què estava passant abans de produir-se la fallada, utilitzeu l'ordre **gdb** i **bt** per visualitzar la traça de pila del fil que ha fallat, en lloc d'analitzar el fitxer core.

Coneixement de l'entorn del Java

Utilitzeu el Javadump per determinar el que fa cada fil i quins mètodes del Java s'executen. Compareu les adreces de les funcions amb les adreces de les biblioteques per determinar l'origen del codi que s'executa a diversos punts.

Utilitzeu l'opció **-verbose:gc** per veure l'estat de l'emmagatzematge dinàmic del Java i les àrees de memòria amb àmbit i de memòria immortal. Feu-vos aquestes preguntes:

- Hi ha una manca de memòria en alguna de les àrees de memòria que hagi pogut provocar la fallada?
- La fallada s'ha produït durant la recollida de deixalles, cosa que indicaria una possible fallada de la recollida de deixalles?
- La fallada s'ha produït després de la recollida de deixalles, cosa que indicaria possibles danys a la memòria?

Depuració de problemes de rendiment

En depurar problemes de rendiment, tingueu en compte aquests aspectes específics per a l'IBM WebSphere Real Time for RT Linux a més dels temes de la guia de l'usuari de l'IBM SDK for Java 7.

Dimensionament de les àrees de memòria

La JVM es pot ajustar canviant les mides de la memòria amb abast, permanent i de l'emmagatzematge dinàmic. Trieu la mida correcta per optimitzar el rendiment. L'ús de la mida correcta pot facilitar que el recollidor de deixalles proporcioni l'ús necessari.

Per obtenir més informació sobre el canvi de la mida de les àrees de memòria, consulteu “Resolució de problemes del recollidor de deixalles Metronome” a la pàgina 135.

Compilació i rendiment de JIT

Quan utilitzeu el JIT, heu de tenir en compte les implicacions en el comportament en temps real.

Si necessiteu un comportament predictible però també necessiteu un rendiment millor, heu de pensar en la possibilitat d'utilitzar la compilació anticipada (AOT - Ahead-Of-Time). Per obtenir-ne més informació, consulteu “Ús del codi compilat amb l' WebSphere Real Time for RT Linux” a la pàgina 36.

Limitacions conegudes del Linux

El Linux ha sofert un desenvolupament molt ràpid, i hi ha diversos problemes amb la interacció de la JVM i del sistema operatiu, especialment en l'àrea dels fils.

Observeu les limitacions següents que podrien afectar el vostres sistema Linux.

Fils com a processos

Si el nombre de fils del Java supera el nombre màxim de processos permesos, possiblement el vostre programa:

- Obtindrà un missatge d'error
- Obtindrà un error **SIGSEGV**
- S'aturarà

Per obtenir més informació, vegeu *l'informe Volano* a l'adreça http://www.volano.com/report/index.html.

Limitacions de piles flotants

Si esteu executant sense piles flotants, independentment de la definició per a **-Xss**, es proporciona una mida de pila nativa de 256 KB per a cada fil.

En un sistema Linux de pila flotant, s'utilitzen els valors de **-Xss**. Si migreu des d'un sistema Linux que no és de pila flotant, assegureu-vos que tots els valors **-Xss** són suficientment grans i no confien en un mínim de 256 KB.

limitacions de glibc

Si rebeu un missatge que indica que la biblioteca `libjava.so` no es pot carregar a causa d'un símbol que no es troba (com ara __bzero), és possible que tingueu instal•lada una versió anterior de la glibc (GNU C Runtime Library). La implementació de fils de l'SDK per al Linux requereix la glibc versió 2.3.2 o superior.

Limitacions de tipus de lletra

Quan feu la instal•lació en un sistema Red Hat, per permetre que el servidor de tipus de lletra trobi els tipus de lletra TrueType del Java TrueType, executeu (al Linux IA32, per exemple):

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

Podeu fer-ho en temps d'execució i heu d'haver iniciat la sessió com a usuari "root" per executar l'ordre. Per veure problemes de tipus de lletra més detallats, consulteu la publicació *Linux SDK and Runtime Environment User Guide*.

Problemes de rendiment als kernels del Linux Red Hat MRG

Un problema de configuració amb els kernels del Red Hat MRG pot causar pauses inesperades en els fils d'una aplicació quan s'inicia el WebSphere Real Time amb la recollida de deixalles detallada habilitada. No s'informa d'aquestes pauses al resultat detallat de GC, però poden durar diversos mil•lisegons, en funció de la configuració de la xarxa. Les JVM iniciades des d'usuaris definits de forma remota són les més afectades, perquè el daemon de la memòria cau de servei (`nscd`) no s'inicia, provocant retards a la xarxa. Solucioneu aquest problema iniciant el `nscd`. Seguiu aquests passos per comprovar l'estat del servei `nscd` i corregir el problema:

1. Comproveu que el daemon d'`nscd` s'executa, escrivint l'ordre:

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

   Si el daemon no s'executa, veureu el missatge següent:

   ```
   nscd is stopped
   ```

2. Com a usuari root, inicieu el servei `nscd` amb l'ordre següent:

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

3. Com a usuari root, canvieu la informació d'inici per al servei `nscd` amb l'ordre següent:

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

Ara el procés `nscd` s'està executant, i s'inicia automàticament després del reinici.

Determinació de problemes de suport multingüístic

La JVM conté un suport integrat per a diferents entorns locals.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil per diagnosticar problemes del suport multingüístic, com per exemple:

- Visió general dels tipus de lletra
- Utilitats de tipus de lletra
- Problemes comuns de suport multingüístic i possibles causes

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Determinació de problemes de suport multingüístic.

Determinació de problemes amb l'ORB

Una de les primeres tasques quan es depura un problema d'ORB és determinar si el problema és al client o al servidor de l'aplicació distribuïda. Considereu una sessió d'RMI-IIOP habitual com una comunicació simple i síncrona entre un client que sol•licita accés a un objecte i un servidor que el proporciona.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil per diagnosticar problemes de l'ORB, com per exemple:

- Identificació d'un problema d'ORB
- Interpretació de la traça de pila
- Interpretació de les traces d'ORB
- Problemes comuns
- Servei ORB d'IBM: recollida de dades

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Determinació de problemes amb l'ORB.

La informació següent és complementària per a l'IBM WebSphere Real Time for RT Linux.

Servei ORB d'IBM: recollida de dades

En recollir la sortida de la versió de Java per al servei, executeu l'ordre següent:

```
java -Xrealtime -version
```

Proves preliminars

Si es produeix un problema, l'ORB pot generar una excepció `org.omg.CORBA.*` que inclou:

- El text que n'indica la causa
- Un codi menor minor
- Un estat de finalització

Abans que assumiu que l'ORB és la causa del problema, comproveu el següent:

- L'escenari es pot reproduir en una configuració semblant.
- El JIT està inhabilitat.
- No s'utilitza cap codi compilat AOT.

Altres accions:

- Desactiveu els processadors addicionals.

- Desactiveu els fils simultanis (SMT) on sigui possible.
- Elimineu les dependències de memòria amb el client o el servidor. La manca de memòria física pot ser la causa del baix rendiment, de bloqueigs aparents o de fallades. Per eliminar aquests problemes, assegureu-vos que teniu un marge raonable de memòria.
- Comproveu si hi problemes físics a la xarxa, com ara tallafocs, enllaços de comunicació, encaminadors i servidors de noms DNS. Són la causa principal de les excepcions COMM_FAILURE de CORBA. Com a prova, feu ping al nom de la vostra estació de treball.
- Si l'aplicació utilitza una base de dades com ara DB2, canvieu al programa de control més fiable. Per exemple, per aïllar DB2 AppDriver, canvieu a Net Driver, que és més lent i utilitza sòcols, però és més fiable.

Resolució d'errors OutOfMemory

Gestió de les excepcions OutOfMemoryError, fuites de memòria i assignacions de memòria oculta.

Per obtenir informació de resolució de problemes del recollidor de deixalles Metronome, consulteu “Resolució de problemes del recollidor de deixalles Metronome” a la pàgina 135.

Diagnòstic d'OutOfMemoryErrors

Diagnosticar excepcions OutOfMemoryError al recollidor de deixalles Metronome pot ser més complex que en una JVM estàndard a causa de la naturalesa periòdica del recollidor de deixalles.

Les característiques dels diferents tipus d'emmagatzematge dinàmic es descriuen a “Gestió de la memòria” a la pàgina 13. En general, una aplicació RTSJ necessita aproximadament un 20% més d'espai d'emmagatzematge dinàmic que una aplicació Java estàndard.

Per defecte, la JVM produeix la següent sortida de diagnòstic quan es produeix un error OutOfMemoryError no detectat:

- Un abocament de memòria d'instantània; vegeu “Utilització d'agents d'abocament de memòria” a la pàgina 114.
- Un abocament de memòria d'emmagatzematge dinàmic; vegeu “Utilització del Heapdump” a la pàgina 122.
- Un abocament de memòria Java; vegeu “Utilització de Javadump” a la pàgina 117
- Un abocament de memòria del sistema; vegeu “Utilització dels abocaments de memòria del sistema i el visualitzador d'abocaments de memòria” a la pàgina 126.

Ells noms dels fitxers d'abocament de memòria s'indiquen a la sortida 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".
```

La traça en sentit invers de Java que es mostra a la sortida de la consola, i que també està disponible al Javadump, indica on s'ha produït l'error OutOfMemoryError de l'aplicació Java. El pas següent consisteix a determinar quina àrea de memòria RTSJ és plena. El component de gestió de memòria de la JVM emet un punt de traça que indica la mida, l'adreça del bloc de classes i el nom de l'espai de memòria de l'assignació que falla. Trobareu aquest punt de traça a l'abocament d'instantània:

```
<< 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
```

L'ID del punt de traça i els camps de dades poden ser diferents dels que es mostren aquí, en funció del tipus d'objecte assignat. En aquest exemple, el punt de traça mostra que l'error d'assignació s'ha produït quan l'aplicació provava d'assignar un objecte de 33,6 MB de tipus `class 0x81312d8` a `Metronome heap`, segment de memòria `id=0x809c5f0`.

Podeu determinar quina àrea de memòria RTSJ queda afectada consultant la informació de gestió de memòria del Javadump:

```
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
<< línies eliminades per facilitar la lectura >>
```

Podeu determinar el tipus d'objecte que s'assigna consultant la secció de classes del Javadump:

```
NULL           ------------------------------------------------------------
0SECTION       CLASSES subcomponent dump routine
NULL           ================================
<< lines omitted... >>
```

```
1CLTEXTCLLOD    ClassLoader loaded classes
2CLTEXTCLLOAD    Loader *System*(0xF182BB80)
<< lines omitted... >>
3CLTEXTCLASS      [C(0xF1632D80)
```

La informació del Javadump confirma que l'assignació que s'ha provat de fer era per a una matriu de caràcters, a l'emmagatzematge dinàmic nromal (`ID=0xF288B584`) i que la mida total assignada de l'emmagatzematge dinàmic, indicada per la línia `1STHEAP` corresponent, és 67108864 bytes decimals o 0x04000000 bytes hexadecimals, o 64 MB.

En aquest exemple, l'assignació que falla és gran en relació amb la mida total de l'emmagatzematge dinàmic. Si la vostra aplicació esperava crear objectes de 33 MB, el pas següent consisteix a incrementar la mida de l'emmagatzematge dinàmic, utilitzant l'opció **-Xmx**.

És molt comú que l'assignació que falla sigui petita amb relació a la mida total del 'emmagatzematge dinàmic. Això és així a causa de les assignacions anteriors que emplenen l'emmagatzematge dinàmic. En aquests casos, el proper pas consisteix a utilitzar l'abocament de memòria de l'emmagatzematge dinàmic (Heapdump) per analitzar la quantitat de memòria assignada als objectes existents.

El Heapdump és un fitxer binari comprimit que té una llista de tots els objectes amb les seves classes d'objecte, mida i referències. Analitzeu el Heapdump mitjançant l'eina Memory Dump Diagnostics for Java (MDD4J), que us podeu baixar des de l'IBM Support Assistant (ISA).

Mitjançant l'eina MDD4J, podeu carregar un Heapdump i localitzar tres estructures d'objectes que se sospita que consumeixen grans quantitats d'espai d'emmagatzematge dinàmic. L'eina proporciona diverses visualitzacions per a objectes a l'emmagatzematge dinàmic. Per exemple, MDD4J pot mostrar una visualització on s'indiquen les sospites de fuites i els cinc objectes i paquets principals que col•laboren a la mida de l'emmagatzematge dinàmic. Si seleccionem l'arbre, obtenim més informació quant a la naturalesa de l'objecte contenidor que té fuites.

Per defecte, es genera un fitxer de heapdump que inclou tots els objectes dels espais de memòria RTSJ. Utilitzeu l'opció **-Xdump:heap:request=multiple** de la línia d'ordres per sol•licitar un Heapdump diferent per a cada espai de memòria. Amb diversos abocaments de memòria, podeu analitzar un conjunt d'objectes assignats a una àrea de memòria específica. Els heapdumps s'identifiquen a partir del nom assignat a la sortida 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... >>
```

Com gestiona l'IBM JVM la memòria

L'IBM JVM necessita memòria per a diversos components, incloent-hi regions de memòria per a classes, codi compilat, objectes Java, piles Java i piles JNI. Algunes d'aquestes regions de memòria han de ser en memòria contigua. Altres regions de memòria es poden segmentar en regions de memòria més petites que es poden enllaçar entre elles.

El codi compilat i les classes carregades de manera dinàmica s'emmagatzemen en regions de memòria segmentada per a classes carregades de manera dinàmica. Les classes es subdivideixen encara més en regions de memòria gravables (classes RAM) i en regions de memòria de només lectura (classes ROM). En temps d'execució, la memòria cau de classes és memòria que s'assigna, però no necessàriament es carrega, en una regió de memòria contigua en arrencar l'aplicació. A mesura que l'aplicació fa referència a les classes, les classes i el codi compilat de la memòria cau de classes s'assignen a l'emmagatzematge. El component ROM de la classe es comparteix entre diversos processos que fan referència a aquesta classe. El component de RAM de la classe es crea a les regions de memòria segmentada per a classes carregades dinàmicament la primera vegada que la JVM fa referència a la classe. El codi compilat AOT per als mètodes d'una classe de la memòria cau de classes es copia en una regió de memòria de codi dinàmic executable, perquè els processos no comparteixen aquest codi. Les classes que no es carreguen de la memòria cau de classes són similars a les classes emmagatzemades a la memòria cau, tret que la informació de les classes ROM es crea en regions de memòria segmentada per a classes carregades de manera dinàmica. El codi generat de manera dinàmica s'emmagatzema a les mateixes regions de memòria de codi dinàmic que contenen el codi AOT per a les classes emmagatzemades a la memòria cau.

Tots els objectes Java s'emmagatzemen a la memòria d'emmagatzematge dinàmic estàndard quan s'executa la JVM sense l'opció **-Xrealtime**. Si s'utilitza l'opció **-Xrealtime**, els objectes també es poden assignar de dues regions de memòria addicionals anomenades memòria immortal i memòria amb àmbit.

Aquesta pila per a cada fil Java pot abraçar una regió de memòria segmentada. La pila JNI per a cada fil ocupa una regió de memòria contigua.

Per determinar la manera com està configurada la JVM, utilitzeu l'opció **-verbose:sizes**. Aquesta opció mostra informació sobre les regions de memòria en les quals podeu gestionar la mida. Per ales regions de memòria que no són continues, es mostra un increment que descriu quanta memòria s'adquireix cada vegada que la regió necessita créixer.

Tot seguit es mostra un exemple de sortida amb les opcions **-Xrealtime -verbose:sizes**:

```
-Xmca32K                        RAM class segment increment
-Xmco128K                       ROM class segment increment
-Xms64M                         initial memory size-Xgc:immortalMemorySize=16M     immortal memory sp
-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
```

Aquest exemple indica que el segment de classes RAM és inicialment 0, però es creix en blocs de 32 KB segons sigui necessari. El segment de classes ROM és inicialment 0, i creix en blocs de 128 KB segons sigui necessari. Podeu utilitzar les

opcions **-Xmca** i **-Xmco** per controlar aquestes mides. Els segments de classes RAM i classes ROM creixen segons és necessari, per la qual cosa normalment no caldrà canviar aquestes opcions.

La memòria immortal és una regió contigua i pot ser que calgui assignar-li prèviament un espai més gran. En aquest exemple, la regió de memòria immortal està preassignada a 16 MB. Si proveu d'escriure més de 16 MB d'objectes en aquesta regió immortal, rebreu una excepció OutOfMemory, perquè, per definició, en aquesta àrea de memòria no es duu a terme la recollida de deixalles.

La regió de memòria amb àmbit és contigua i en aquest exemple està preassignada a 8MB. Si teniu moltes àrees de memòria amb àmbit actives quan s'executa el programa, pot ser que hagueu d'especificar una regió de memòria amb àmbit més gran.

Utilitzeu la utilitat admincache per determinar la mida de la regió assignada de memòria si utilitzeu la memòria cau de classes. Tot seguit es mostra un exemple de la sortida de l'ordre `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
```

La mida de la memòria cau indica que la regió assignada de memòria tindrà una mica més de 7 MB d'espai. La classe ROM i els bytes AOT ocupen quasi tot aquest espai, una mica més de 3 MB cadascun.

Exemple d'error OutOfMemoryError a l'espai de memòria immortal

Aquest exemple mostra com identificar un error OutOfMemoryError en un espai de memòria immortal i descriu els passos que cal dur a terme per evitar el problema.

Aquest abocament de memòria d'instantània mostra que dues sol•licituds d'assignació de memòria han fallat a l'àrea de memòria immortal 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 Javadump mostra que l'espai de la memòria immortal és ple:

```
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
```

Una anàlisi MDD4J mostra que s'ha assignat una llista LinkedList molt llarga que consumeix una proporció considerable de la memòria disponible.

Es recomana minimitzar el nombre d'objectes assignats a l'àrea de memòria immortal perquè els objectes d'aquesta àrea no estan subjectes a la recollida de deixalles. L'ús més comú de la memòria immortal és la càrrega de classes, que és una activitat finita que es produeix principalment durant la inicialització de la JVM i l'aplicació. Les aplicacions amb moltes classes carregades (a banda de l'ús de la memòria immortal) poden incrementar la mida de l'àrea de memòria immortal mitjançant l'opció **-Xgc:immortalMemorySize=<mida>**. La mida per defecte per a l'àrea de memòria immortal és 16 MB.

Si el fet d'incrementar la mida de l'àrea de memòria immortal només retarda l'error OutOfMemoryError per a la memòria immortal, analitzeu l patró de l'assignació continuada de dades immortals, relacionada amb la càrrega de classes o amb altres objectes d'aplicació.

Exemple d'error OutOfMemoryError en un espai de memòria amb àmbit

Aquest exemple mostra com identificar un error OutOfMemoryError en un espai de memòria amb àmbit i descriu els passos que cal dur a terme per evitar el problema.

Utilitzeu l'opció de la línia d'ordres **-Xdump:heap:request=multiple** per produir abocaments de memòria independents per a cada espai de memòria:

```
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... >>
```

Aquest abocament de memòria mostra que dues sol•licituds de memòria han fallat a l'àrea de memòria amb àmbit 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 Javadump mostra que, per a l'àrea de memòria amb àmbit amb id=0x809dd10, la mida assignada de l'àrea de memòria és força petita, només 60 KB; en aquest cas, incrementeu la mida de l'àrea de memòria amb àmbit al codi de l'aplicació.

```
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
```

Al Javadump de l'exemple, l'àrea de memòria amb àmbit és buida. Apareix buida perquè el Javadump s'ha produït quan l'error OutOfMemoryError arriba a la JVM i, en aquest moment, l'àmbit de memòria ha sortit i s'ha netejat. Podeu produir un Javadump en el punt d'error utilitzant l'opció de línia d'ordres **-Xdump:java:events=throw,filter=java/lang/OutOfMemoryError**. Si utilitzeu aquesta opció, l'espai lliure de l'àrea de memòria amb àmbit es notifica correctament.

També és possible que s'exhaureixi tot l'espai disponible per a la memòria amb àmbit; en aquest cas, incrementeu la mida de l'àrea de memòria amb àmbit utilitzant l'opció de la línia d'ordres **-Xgc:scopedMemoryMaximumSize=<mida>**. La mida per defecte per a l'àrea de memòria amb àmbit és 8 MB. Si l'espai total disponible per a la memòria amb àmbit s'exhaureix, veureu diferents missatges a la consola, com ara aquest:

```
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)
```

Diagnòstic de problemes de diversos emmagatzematges dinàmics

Podeu utilitzar els intervals d'adreces proporcionats al Javadump amb la informació d'ocupació al Heapdump per ajudar a analitzar els error OutOfMemoryError a diverses àrees de memòria d'RTSJ.

En aquest Javadump, el segment immortal va de 0xB281C008 a 0xB381C008, i el segment d'emmagatzematge dinàmic normal va de 0xB381D008 a 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 Heapdump és un fitxer binari comprimit que té una llista de tots els objectes amb les seves classes d'objecte, mida i referències. Analitzeu el Heapdump mitjançant l'eina Memory Dump Diagnostics for Java (MDD4J), que us podeu baixar des de l'IBM Support Assistant (ISA).

Podeu utilitzar les ubicacions de memòria d'objectes indicades per MDD4J per determinar l'espai de memòria en el qual resideix un objecte. Les adreces de l'interval 0xB28*nnnnn* són a l'àrea de memòria immortal. Les adreces de l'interval 0xB61*nnnnn* són a l'emmagatzematge dinàmic normal.

Com evitar fuites de memòria

El recollidor de deixalles no processa àrees de memòria immortal ni amb àmbit. En el cas de la memòria immortal, la memòria s'allibera només quan la JVM surt. Les àrees de memòria amb àmbit només s'alliberen després que el seu recompte de referències sigui zero. Les tasques de llarga execució que s'executen en aquests contexts s'han d'escriure de tal manera que, una vegada que la tasca hagi fet l'escalfament, no s'assigni memòria addicional de l'àrea de memòria immortal.

El procés de càrrega de classes utilitza una petita quantitat de memòria immortal. Aquestes classes no estan subjectes a la recollida de deixalles a l'entorn en temps

real. Com a tal, el procés de càrrega de classes que no són necessàries per a l'aplicació pot fer que l'aplicació utilitzi més memòria immortal de la necessària.

Si l'aplicació conté classes que implementen la interfície Serializable, ajusteu la mida de la memòria immortal inicial perquè justifiqui l'impacte de les classes generades. A cada constructor es genera un objecte per classe, amb el format "GeneratedSerializationConstructorAccessorXXX" (on XXX és un número) que es carrega a la memòria immortal la primera vegada que se serialitza l'objecte.

Eviteu l'ús de memòria immortal, perquè la recollida de deixalles no es pot aplicar als objectes assignats des de la memòria immortal. Considereu la possibilitat de fer una agrupació a la memòria immortal si l'àrea de memòria immortal no s'utilitza només de manera ocasional.

Assignació de memòria oculta a través de característiques d'idioma

En un context de memòria amb àmbit o immortal, eviteu la característica d'idioma d'arguments de variable perquè aquests mètodes assignen memòria oculta.

Arguments de variable (vararg)

El llenguatge Java implementa arguments de variable passant-los al mètode com a matriu. El compilador facilita la crida als mètodes d'arguments de variable i crea i inicialitza la matriu automàticament.

Pot ser que es perdi memòria en cridar un mètode d'argument de variable en un context de memòria immortal o amb àmbit. No utilitzeu els arguments de variable en contextos de memòria amb àmbit o immortal. En comptes d'això, creeu explícitament una matriu i utilitzeu-la en lloc dels arguments de variable.

Tot seguit es mostren dos exemples de maneres equivalents de cridar un mètode d'argument 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;
    }
}
```

És preferible el segon exemple. Ja que l'assignació de matriu doble esdevé invisible al codi, l'assignació es pot dirigir cap una àrea de memòria concreta.

Concatenació de cadenes

L'addició de dades a una cadena existent per produir una cadena més llarga s'implementa mitjançant objectes java.lang.StringBuilder, per la qual cosa es necessiten assignacions de memòria.

Autoempaquetatge

L'autoempaquetatge implica crear un objecte que contingui un tipus bàsic, que necessita assignacions de memòria.

Utilització de la reflexió en contextos de memòria

Si un objecte constructor s'ha creat en una àrea de memòria amb àmbit, només es pot utilitzar al mateix àmbit o en un àmbit intern. Qualsevol intent d'utilitzar aquest objecte constructor en un context de memòria immortal, d'emmagatzematge dinàmic o de memòria d'àmbit fallarà.

L'excepció que es genera quan s'ha produït una reflexió en contextos de memòria serà similar a aquesta:

```
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)
```

És possible aplicar una solució temporal per a aquesta restricció utilitzant el constructor al mateix àmbit en que es va assignar.

Utilització de classes internes amb àrees de memòria amb àmbit

Quan utilitzeu classes internes en el context d'àrees de memòria amb àmbit, heu de tenir cura a l'hora de crear instàncies dels objectes de classes interns si els objectes externs i interns són en àrees de memòria diferents. Si l'objecte no pot emmagatzemar una referència a l'objecte extern, es produirà un error IllegalAssignmentError al codi generat pel compilador que no és visible al codi font original.

Un objecte de classe intern ha de poder emmagatzemar una referència implícita en el seu objecte de classe extern. Si la referència infringeix les regles de referència de memòria RTSJ, es generarà un error IllegalAssignmentError.

La major part de les classes internes (incloent-hi les classes internes locals i anònimes) contindran un camp no estàtic (sintètic) generat pel compilador per a la instància de la classe externa englobadora a nivell lèxic. L'única excepció es produeix quan una instància de classe interna no té cap objecte extern englobador, com ara un objecte de classe anònim la instància del qual s'ha creat en un bloc d'inicialitzador estàtic. El camp sintètic de l'objecte intern contindrà una referència a l'objecte extern. Això s'implementa mitjançant el compilador per facilitar la feina al programador java. El camp no estarà visible al codi font original, tot i que és possible escriure codi similar utilitzant classes imbricades estàtiques amb una referència que és visible. Si la referència implícita infringeix les regles de l'àrea de memòria RTSJ, s'emetrà un error IllegalAssignmentError en construir l'objecte intern, perquè intenta emmagatzemar la referència a l'objecte extern.

En general, no podeu infringir les regles de referència de la memòria RTSJ quan utilitzeu classes internes. No podeu crear un objecte intern si una referència a l'objecte extern associat infringeix les regles de referència de la memòria RTSJ. Aquesta regla vol dir que un objecte interna assignat a la memòria immortal o a l'emmagatzematge dinàmic no pot tenir cap referència a un objecte extern de la memòria amb àmbit. Un objecte intern de la memòria amb àmbit pot tenir una referència a un objecte extern de la memòria amb àmbit, però l'objecte extern s'ha d'assignar des de la mateixa àrea de memòria amb àmbit o d'una àrea de memòria amb àmbit externa.

Hi ha solucions temporals, com ara:

- Utilitzar classes imbricades estàtiques per eliminar la referència implícita.
- Triar àrees de memòria per garantir que les relacions d'objectes interns i externs no infringeixen les restriccions de referència de les àrees de memòria.

Utilització d'eines de diagnòstic

Hi ha diverses eines de diagnòstic disponibles que ajuden a diagnosticar els problemes de la JVM de l'IBM WebSphere Real Time for RT Linux.

L'IBM SDK for Java 7 proporciona diverses eines de diagnòstic que podeu utilitzar per ajudar a diagnosticar els problemes de la JVM de l'IBM WebSphere Real Time for RT Linux. En aquest apartat es presenten les eines que hi ha disponibles i es proporcionen enllaços a informació sobre el seu ús.

Hi ha un aspecte important que cal recordar quan s'utilitzen les eines de diagnòstic de l'SDK. Quan invoqueu la JVM en temps real, utilitzeu l'opció següent:

```
java -Xrealtime
```

Cal utilitzar aquesta opció quan s'executen les eines de diagnòstic per a la JVM en temps real. Per exemple, per mostrar els agents d'abocament de memòria registrats de la JVM de l'IBM WebSphere Real Time for RT Linux, escriviu:

```
java -Xrealtime -Xdump:what
```

A continuació, i com a informació complementària, es descriuen les diferències d'ús d'aquestes eines amb l'IBM WebSphere Real Time for RT Linux i s'inclou exemples de sortida per facilitar-ne el diagnòstic.

Per veure un resum de la informació de diagnòstic generada per l'IBM SDK for Java 7, vegeu Resum de la informació de diagnòstic.

Utilització d'agents d'abocament de memòria

Els agents d'abocament de memòria es configuren durant la inicialització de la JVM. Permeten utilitzar incidències que tenen lloc a la JVM, com ara la recollida de deixalles, l'inici dels fils o la terminació de la JVM, a fi d'iniciar abocaments de memòria o iniciar una eina externa.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil sobre els agents d'abocament de memòria, com per exemple:

- Utilització de l'opció **-Xdump**
- Agents d'abocament de memòria
- Incidències d'abocament de memòria
- Control avançat dels agents d'abocament de memòria
- Testimonis d'agent d'abocament de memòria
- Agents d'abocament de memòria per defecte
- Eliminació dels agents d'abocament de memòria
- Variables d'entorn d'agent d'abocament de memòria
- Mapatges de senyals
- Ubicacions per defecte dels agents d'abocament de memòria

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Utilització dels agents d'abocament de memòria.

A continuació es proporciona informació complementària per a l'IBM WebSphere Real Time for RT Linux:

Incidències d'abocament de memòria

Els agents d'abocament de memòria s'activen quan es produeixen incidències durant el funcionament de la JVM. Per a l'IBM WebSphere Real Time for RT Linux, el valor per defecte per a la incidència slow és de 5 mil•lisegons.

Algunes incidències es poden filtrar per millorar la rellevància de la sortida. Consulteu “Opció de filtre” a la pàgina 115 per obtenir-ne més informació.

Nota: Actualment no es produeixen incidències de càrrega i expansió al WebSphere Real Time. Les classes són a memòria permanent i no es poden descarregar.

Nota: Les incidències gpf i abort no poden activar un abocament de memòria d'emmagatzematge dinàmic, preparar l'emmagatzematge dinàmic (sol•licitud=prepwalk) ni compactar l'emmagatzematge dinàmic (sol•licitud=compact).

La taula següent mostra les incidències disponibles com a activadors d'agents d'abocament de memòria:

Incidència	Produïda quan...	Operació de filtre
gpf	Quan es produeix un error de protecció general (GPF).	

Incidència	Produïda quan...	Operació de filtre
usuari	Quan la JVM rep el senyal SIGQUIT del sistema operatiu.	
abort	Quan la JVM rep el senyal SIGABRT del sistema operatiu.	
vmstart	Quan s'inicia la màquina virtual.	
vmstop	Quan s'atura la màquina virtual.	Filtra per codi de sortida; per exemple, **filter=#129..#192#-42#255**
load	Quan es carrega una classe.	Filtra per nom de classe; per exemple, **filter=java/lang/String**
unload	Quan es carrega una classe.	
throw	Quan es genera una excepció.	Filtra per nom de classe d'excepció; per exemple, **filter=java/lang/OutOfMem***
catch	Quan es detecta una excepció.	Filtra per nom de classe d'excepció; per exemple, **filter=*Memory***
uncaught	Quan l'aplicació no detecta una excepció Java.	Filtra per nom de classe d'excepció; per exemple, **filter=*MemoryError**
systhrow	Quan la JVM està a punt de generar una excepció Java. És diferent de la incidència 'throw' perquè només s'activa per a condicions d'error que es detecten internament a la JVM.	Filtra per nom de classe d'excepció; per exemple, **filter=java/lang/OutOfMem***
thrstart	Quan s'inicia un fil nou.	
blocked	Quan es bloqueja un fil.	
thrstop	Quan s'atura un fil.	
fullgc	Quan s'inicia un cicle de recollida de deixalles.	
slow	Quan un fil triga més de 5 ms a respondre a una sol•licitud de JVM interna.	Canvia el temps que es triga a considerar que una incidència és lenta; per exemple **filter=#300ms** s'activarà quan un fil trigui més de 300 ms a respondre a una sol•licitud de JVM interna.
allocation	Quan un objecte Java s'assigna amb una mida que coincideix amb l'especificació de filtre indicada.	Filtra per mida d'objecte; cal proporcionar un filtre. Per exemple, **filter=#5m** s'activarà amb objectes de més de 5 Mb. També s'admeten intervals; per exemple, **filter=#256k..512k** s'activarà amb objectes d'una mida entre 256 Kb i 512 Kb.
traceassert	Es produeix un error intern a la JVM	No aplicable.
corruptcache	La JVM descobreix que la memòria cau de classes compartida s'ha malmès.	No aplicable.

Opció de filtre

Algunes incidències de JVM es produeixen milers de vegades durant la vida útil d'una aplicació. Els agents d'abocament de memòria podeu utilitzar filtres i intervals per evitar que es generin massa abocaments de memòria.

Comodins

Podeu utilitzar un comodí al filtre d'incidències d'excepció col•locant un asterisc només al començament o al final del filtre. L'ordre següent no funciona perquè el segon asterisc no és al final:

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

Perquè aquest filtre funcioni, cal canviar-lo a:

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

Càrrega de classes i incidències d'excepció

Podeu filtrar la càrrega de classes (load) i les incidències d'excepció (throw, catch, uncaught, systhrow) pel nom de classe Java:

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

Podeu filtrar les incidències d'excepció throw, uncaught i systhrow pel nom de mètode Java:

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

Les parts opcionals es mostren entre claudàtors.

Podeu filtrar les incidències d'excepció pel nom de mètode Java:

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

Les parts opcionals es mostren entre claudàtors.

Incidència vmstop

Podeu filtrar la incidència de conclusió de la JVM utilitzant un o més codis de sortida:

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

Incidència slow

Podeu filtrar la incidència slow per canviar el llindar de temps del valor per defecte 5 ms:

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

No podeu definir el filtre en menys temps que el temps per defecte.

Incidència allocation

Cal que filtreu la incidència allocation per especificar la mida dels objectes que ocasionen un activador. Podeu definir la mida del filtre de zero fins al valor màxim de punter de 32 bits a les plataformes de 32 bits o el valor màxim d'un punter de 64 bits a les plataformes de 64 bits. La definició del valor de filtre més baix en zero activa un abocament de memòria a totes les assignacions.

Per exemple, per activar abocaments de memòria a les assignacions amb més de 5 Mb, utilitzeu:

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

Per activar els abocaments de memòria a les assignacions amb una mida entre 256 Kb i 512 Kb, utilitzeu:

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

Altres incidències

Si apliqueu un filtre a una incidència que no admet el filtratge, el filtre s'ignora.

Opció request

Utilitzeu l'opció request per demanar a la JVM que prepari l'estat abans d'iniciar l'agent d'abocament de memòria. Per a l'IBM WebSphere Real Time for RT Linux, hi ha una opció request addicional: **multiple**.

Les opcions disponibles s'enumeren a la taula següent:

Valor de l'opció	Descripció
exclusive	Sol•licita accés exclusiu a la JVM.
compact	Executa la recollida de deixalles. Aquesta opció elimina tots els objectes als quals no es pot accedir de l'emmagatzematge dinàmic abans que es generi l'abocament de memòria.
prepwalk	Prepara l'emmagatzematge dinàmic perquè avanci. Cal que especifiqueu també **exclusive** quan utilitzeu aquesta opció.
serial	Suspèn altres abocaments de memòria fins que aquest ha finalitzat.
multiple	Genera abocaments de memòria d'emmagatzematge dinàmic independents per a cada àrea de memòria d'RTSJ.
preempt	S'aplica a l'agent d'abocament de memòria Java i controla si els fils natius del procés s'avancen de manera forçada per recollir traces de pila. Si aquesta opció no s'especifica, al Javadump només es recopilen traces de pila Java.

Per exemple, el valor per defecte de l'opció de sol•licitud per als Javadumps és `request=exclusive+preempt`. Per canviar els valors de manera que els Javadumps es produeixin sense avançar els fils de manera forçada per recopilar traces de piles natives, utilitzeu aquesta opció:

```
-Xdump:java:request=exclusive
```

En general, amb les opcions de request per defecte hi ha prou.

Podeu especificar més d'una opció de request amb +. Per exemple:

```
-Xdump:heap:request=exclusive+compact+prepwalk
```

Utilització de Javadump

Un Javadump genera fitxers que contenen informació de diagnòstic relacionada amb la JVM i una aplicació Java capturada en un punt durant l'execució. Per exemple, la informació pot ser sobre el sistema operatiu, l'entorn d'aplicació, els fils, les piles, els bloqueigs i la memòria.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil sobre Javadumps, com per exemple:

- Habilitació d'un Javadump
- Activació d'un Javadump
- Interpretació d'un Javadump
- Variables d'entorn i Javadump

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Utilització de Javadump.

Als temes següents es proporcionen informació complementària i sortida d'exemple per a l'IBM WebSphere Real Time for RT Linux.

Gestió de l'emmagatzematge (MEMINFO)

La secció `MEMINFO` proporciona informació sobre el gestor de memòria, que inclou les àrees d'emmagatzematge dinàmic, memòria immortal i amb àmbit.

La secció `MEMINFO` d'un Javadump mostra informació sobre el gestor de memòria.Consulteu Utilització del recollidor de deixalles Metronome per obtenir informació detallada sobre com funciona el component de gestor de memòria.

Aquesta part del Javadump proporciona diversos valors de gestió de l'emmagatzematge, com ara:

- Quantitat de memòria lliure
- Quantitat de memòria utilitzada
- Mida actual de l'emmagatzematge dinàmic
- Mida actual de les àrees de memòria immortal
- Mida actual de les àrees de memòria amb àmbit

Aquesta secció també conté dades històriques sobre la recollida de deixades. Les dades es mostren com a seqüència de punts de traça, cadascun amb una marca horària, ordenats a partir del punt de traça més recent.

Els Javadumps generats per la JVM estàndard contenen una secció "GC History". Aquesta informació no està continguda als Javadumps generats quan s'utilitza la JVM en temps real. Utilitzeu l'opció **-verbose:gc** o la traça breu de la JVM per obtenir informació sobre el comportament de la recollida de deixalles. Consulteu "Utilització de la informació verbose:gc" a la pàgina 135 i l'apartat sobre agents d'abocament de memòria de la guia de l'usuari de l'IBM SDK for Java 7 per obtenir-ne més detalls.

Si utilitzeu un programa que utilitza memòria amb àmbit i es genera una excepció OutOfMemoryError, algunes de les àrees de memòria que apareixen al Javadump poden ser buides. Quan un àmbit que està imbricat dins un altre àmbit es queda sense memòria, l'àmbit interior es pot haver suprimit quan es genera el Javadump. Per obtenir informació relacionada amb l'estat de les àrees de memòria en el moment en què es genera l'OutOfMemoryError, executeu el programa amb l'opció de línia d'ordres següent:

```
-Xdump:java:events=throw,filter=java/lang/OutOfMemoryError,range=1..1
```

Aquesta ordre genera un Javadump addicional quan es genera l'excepció OutOfMemoryError, en lloc de quan es detecta una excepció no detectada, cosa que succeeix poc després. En aquest Javadump, podeu veure totes les àrees de memòria que estaven actives quan s'ha generat l'OutOfMemoryError, incloent-hi els àmbits interns. Per obtenir més informació sobre l'ús de l'opció **-Xdump**, consulteu la guia de l'usuari de l'IBM SDK for Java 7.

En un Javadump, els segments són blocs de memòria assignats pel temps d'execució Java per a tasques que utilitzen grans quantitats de memòria. Aquests són exemples de tasques:

- Mantenir les memòries cau JIT
- Emmagatzemar classes Java

El temps d'execució Java també assigna una altra memòria nativa, que no s'indica a la secció MEMINFO. La memòria total utilitzada pels segments de temps d'execució Java no representa necessàriament l'impacte de memòria total del temps d'execució Java. Un segment de temps d'execució Java està format per l'estructura de dades del segment, i un bloc associat de memòria nativa.

L'exemple següent mostra una sortida habitual. Tots els valors es proporcionen com a valors hexadecimals. Les capçaleres de columna de la secció MEMINFO signifiquen el següent:

```
| 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
|        (línies eliminades per 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
|         (línies eliminades per 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
|              (línies eliminades per facilitar la lectura)
|  NULL
|  NULL           ------------------------------------------------------------------------
```

Fils i traça de pila (THREADS)

Per al programador d'aplicacions, una de les parts més útils d'un abocament de memòria Java és la secció THREADS. Aquesta secció mostra una llista de fils Java, fils natius, i traces de pila. Per a l'IBM WebSphere Real Time for RT Linux, també es mostren els fils en temps real i els files en temps real que no són d'emmagatzematge dinàmic.

Els fils Java els implementen els fils natius del sistema operatiu. Cada fil es representa mitjançant un conjunt de línies com ara:

```
"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)
```

El noms de fils Java són visibles al sistema operatiu quan s'utilitza l'ordre **ps**. Per obtenir més informació sobre la utilització de l'ordre **ps**, consulteu “Tècniques de depuració generals” a la pàgina 98.

Pot ser que a un abocament de memòria Java (Javadump) generat a partir d'un fil en temps real que no sigui d'emmagatzematge dinàmic li falti informació. Si l'objecte de nom de fil no és visible des del fil en temps real que no és d'emmagatzematge dinàmic, s'imprimeix el text “(access error)” en comptes del nom fil real.

Les propietats de la primera línia són el nom del fil, les adreces de les estructures de fils de la JVM i de l'objecte de fil Java, l'estat del fil i la prioritat del fil Java. Les propietats de la segona línia són l'ID de fil del sistema operatiu natiu, la prioritat de fil del sistema operatiu natiu i la política de planificació del sistema operatiu natiu.

Els noms de fils són visibles de tres maneres:

- S'enumeren als fitxers javacore. No tots els fils s'enumeren als fitxers javacore.
- Quan es llisten fils del sistema operatiu mitjançant l'ordre **ps**.
- Quan s'utilitza el mètode java.lang.Thread.getName().

La taula següent proporciona informació sobre els noms de fils de l'IBM WebSphere Real Time for RT Linux.

Taula 12. Noms de fils a l'IBM WebSphere Real Time for RT Linux

Detall del fil	Nom del fil
Un fil intern de la JVM que utilitza el mòdul de recollida de deixalles per distribuir la finalització dels objectes per fils secundaris.	`Mestre finalitzador`
El fil d'alarma que utilitza el recollidor de deixalles.	`Alarma de GC`
Els fils utilitzats per a la recollida de deixalles.	`Esclau del GC`
Un fil intern de la JVM que utilitza el mòdul del compilador JIT (just-in-time) per obtenir mostres de l'ús de mètodes de l'aplicació.	`Mostrejador JIT`
Un fil que utilitza la VM per gestionar els senyals que rep l'aplicació, tant si s'han generat externament com interna.	`Informador de senyals`

Els noms per defecte dels fils en temps real (javax.realtime.RealtimeThread) creats al codi Java són `RTThread-x`, on “x” és el número del fil.

Els noms per defecte dels fils en temps real que no són d'emmagatzematge dinàmic són `NHRTThread-x`, on “x” és el número del fil.

La prioritat de fil Java es mapa amb un valor de prioritat de sistema operatiu en funció de la plataforma. Un valor elevat per a la prioritat de fil Java significa que el fil té una prioritat alta. És a dir, el fil s'ha d'executar més sovint que els fils de prioritat més baixa. Per obtenir més informació detallada sobre com funciona això per als fils Java, els fils en temps real i els fils en temps real que no són d'emmagatzematge dinàmic, consulteu “Mapatge de prioritats i herència” a la pàgina 12.

El valors d'estat poden ser:

- R: executable; el fil es pot executar quan hi ha oportunitat.
- CW: espera de condició; el fil està a l'espera. Per exemple, perquè:
  - S'ha fet una crida a sleep().
  - S'ha bloquejat el fil per a E/S.
  - S'ha cridat un mètode wait() per esperar que es notifiqui un supervisor.
  - El fil se sincronitza amb un altre fil amb una crida a join().
- S: suspès; el fil l'ha suspès un altre fil.
- Z: zombi; el fil s'ha eliminat amb kill.
- P: aparcat; l'API nova de simultaneïtat ha aparcat el fil (java.util.concurrent).
- B: bloquejat; el fil està a l'espera d'obtenir un bloqueig propietat d'un altre element.

Si un fil està aparcat o blocat, la sortida inclou una línia per al fil, que comença per `3XMTHREADBLOCK`, i indica el recurs que el fil està esperant i, si pot ser, el fil que actualment és propietari del recurs. Per obtenir més informació, vegeu el tema sobre fils blocats a la guia de l'usuari de l'IBM SDK for Java 7.

Quan inicieu un Javadump per obtenir informació de diagnòstic, la JVM consulta els fils Java abans de produir el javacore. Es mostra l'estat de preparació `exclusive_vm_access` a la línia `1TIPREPSTATE` de la secció `TITLE` section.

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

Els fils que executaven codi Java en activar javacore tenen l'estat CW (condició d'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)
```

La secció javacore `LOCKS` mostra que aquests fils esperen en un bloqueig de JVM intern.

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

Utilització del Heapdump

El terme heapdump descriu el mecanisme de la màquina virtual d'IBM per a Java que genera un abocament de memòria de tots els objectes actius que hi ha a l'emmagatzematge dinàmic de Java, és a dir, els que fa servir l'aplicació Java.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil sobre Heapdumps, com per exemple:

- Obtenció de heapdumps
- Eines per processar heapdumps
- Utilització de **-Xverbose:gc** per obtenir l'emmagatzematge dinàmic
- Variables d'entorn i heapdump
- Format de fitxer de heapdump de text (clàssic)
- Format de fitxer PHD (d'abocament de memòria d'emmagatzematge dinàmic portàtil)

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Utilització del Heapdump.

Informació complementària per a l'IBM WebSphere Real Time for RT Linux:

Habilitació de diversos Heapdumps per a JVM en temps real

El Heapdump generat és, per defecte, un únic fitxer que conté informació sobre tots els objectes Java de totes les àrees de memòria, la memòria d'emmagatzematge dinàmic, la memòria immortal i la memòria amb àmbit. El motiu principal per produir diversos abocaments de memòria és que cada àrea d'emmagatzematge dinàmic individual es pot analitzar utilitzant les eines Heapdump tradicionals sense cap modificació.

Quant a aquesta tasca

Per defecte, els heapdumps contenen informació sobre tots els objectes de les àrees de memòria de la JVM: la memòria d'emmagatzematge dinàmica, la memòria permanent i la memòria amb abast. Podeu obtenir Heapdumps independents que continguin informació sobre els objectes Java a cada àrea de memòria utilitzant l'opció **request=multiple** amb **-Xdump:heap**. Heu de tenir present que heu de repetir també els valors per defecte de l'opció de sol•licitud, de manera que heu d'especificar `request=multiple+exclusive+prepwalk+compact`. Això produeix un conjunt de Heapdumps amb un camp addicional al nom que indica l'àrea de memòria específica:

```
heapdump.%id.%Y%m%d.%H%M%S.%pid.phd
```

on *%id* identifica el fitxer de Heapdump que conté objectes a la memòria d'emmagatzematge dinàmic, a la memòria immortal o en una àrea específica de la memòria amb àmbit.

Hi ha quatre tipus d'emmagatzematge dinàmic representats per aquests noms: “Per defecte”, “Immortal”, “Àmbit” i “Altres”. El codi de Heapdump substitueix el valor %id a l'etiqueta de l'emmagatzematge dinàmic per un d'aquests noms units a un identificador (normalment numèric), com ara:
`heapdump.Immortal12994208.20060807.093653.7684.txt.`

Exemple

```
java -Xrealtime
-Xdump:heap:defaults:request=multiple+exclusive+compact+prepwalk <programa java>
```

L'ús d'aquesta opció addicional produeix diversos Heapdumps en format phd (portable Heapdump).

```
java -Xrealtime
-Xdump:heap:defaults:request=multiple+exclusive+compact+prepwalk,opts=CLASSIC
<programa java>
```

L'ús d'aquesta opció addicional produeix diversos Heapdumps en format de text CLASSIC.

L'opció -Xdump:what mostra els agents d'abocament de memòria en arrencar al JVM, i és útil per comprovar les opcions d'abocament de memòria.

Format de fitxer de heapdump de text (clàssic)

El heapdump de text o clàssic és una llista de totes les instàncies d'objecte de l'emmagatzematge dinàmic, incloent-hi el tipus d'objecte, la mida i les referències entre objectes.

Registre de capçalera

El registre de capçalera és un registre individual que conté una cadena d'informació de versió.

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

Exemple:

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

Registres d'objecte

Els registres d'objecte són diversos registres, un per a cada instància de l'emmagatzematge dinàmic, que proporcionen l'adreça d'objecte, la mida, el tipus i les referències de l'objecte.

```
<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> ...
```

Les adreces d'objecte i les referències d'emmagatzematge dinàmic són a l'emmagatzematge dinàmic, però l'adreça de bloc de classes és fora de l'emmagatzematge dinàmic. Totes les referències trobades a la instància d'objecte es llisten, incloses les referències que són valors nuls. El tipus d'objecte és un nom de classe, que inclou un paquet o un tipus de matriu de primitius o de classes, com mostra la seva signatura de tipus de JVM estàndard; consulteu “Signatures de tipus de màquina virtual Java” a la pàgina 126. Els registres d'objecte també poden contenir referències de bloc de classes addicionals, normalment en el cas de les instàncies de classes de reflex.

Exemples:

Una instància d'objecte de 28 bytes de longitud del tipus java/lang/String:

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

Una adreça de bloc de classes de java/lang/String, seguida per una referència a una instància de matriu char:

```
0x415319D8 0x00436EB0
```

Una instància d'objecte de 44 bytes de longitud de matriu char de tipus:

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

Una adreça de bloc de classes de matriu char:

```
0x41530F20
```

Un objecte de matriu de tipus classe interna java/util/Hashtable Entry:

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

Un objecte de tipus classe interna java/util/Hashtable:

```
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 adreça de bloc de classes i referències d'emmagatzematge dinàmic, incloent-hi les referències nul•les.

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

Registres de classe

Els registres de classe són diversos registres, un per a cada classe carregada, que proporcionen l'adreça de bloc de classes, la mida, el tipus i les referències de la classe.

```
<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>...
```

L'adreça de bloc de classes i les referències de bloc de classes són fora de l'emmagatzematge dinàmic, però el registre de classe pot contenir referències a l'emmagatzematge dinàmic, normalment per als membres de dades de classe estàtiques. Totes les referències trobades al bloc de classe es llisten, incloent-hi les que són valors nuls. El tipus de classe és un nom de classe, que inclou un paquet o un tipus de matriu de primitius o de classes, com mostra la seva signatura de tipus de JVM estàndard; consulteu "Signatures de tipus de màquina virtual Java" a la pàgina 126.

Exemples:

Un bloc de classe, d'una longitud de 32 bytes, per a la classe java/lang/Runnable:

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

Referències a altres blocs de classes i les referències d'emmagatzematge dinàmic, incloent-hi les referències nul•les:

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

Bloc de classes, d'una longitud de 168 bytes, per a la classe 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
```

Registre de cua 1

El registre de cua 1 és un registre individual que conté recomptes de registres.

```
// 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>
```

Exemple:

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

Registre de cua 2

El registre de cua 2 és un registre individual que conté totals.

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

Exemple:

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

Signatures de tipus de màquina virtual Java

Les signatures de tipus de màquina virtual Java són les abreviacions dels tipus Java que es mostren a la taula següent:

Signatures de tipus de màquina virtual Java	Tipus Java
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L <*classe totalment qualificada*> ;	<*classe totalment qualificada*>
[<*tipus*>	<*tipus*>[ ] (matriu de <*tipus*>)
(<*tipus-arg*>) <*tipus-ret*>	mètode

Utilització dels abocaments de memòria del sistema i el visualitzador d'abocaments de memòria

La JVM pot generar abocaments de memòria del sistema natius, anomenats també abocaments de memòria del nucli, en condicions configurables. Els abocaments de memòria del sistema solen ser grans. La majoria de les eines que s'utilitzen per analitzar els abocaments de memòria del sistema també són específiques de la plataforma. Utilitzeu l'eina **gdb** per analitzar un abocament de memòria del sistema al Linux.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil sobre l'ús dels abocaments de memòria del sistema i del visualitzador d'abocaments de memòria, com per exemple:

- Visió general dels abocaments de memòria del sistema
- Valors per defecte de l'abocament de memòria del sistema
- Utilització del visualitzador d'abocaments de memòria
  - Utilització de **jextract**
  - Problemes que cal abordar amb el visualitzador d'abocaments de memòria
  - Ordres disponibles a **jdmpview**
  - Exemple de sessió
  - Consulta ràpida de les ordres **jdmpview**

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Utilització dels abocaments de memòria del sistema i el visualitzador d'abocaments de memòria.

Informació complementària per a l'IBM WebSphere Real Time for RT Linux:

Utilització de jextract

En processar un abocament de memòria del sistema d'una JVM en temps real, heu d'incloure l'opció **-Xrealtime**. Per exemple:

```
jextract -Xrealtime <nom_fitxer_core> [<fitxer_zip>]
```

Quan executeu **jextract** en una JVM que és diferent de la JVM per a la qual s'havia generat l'abocament de memòria, veureu els missatges d'error següents:

```
J9RAS.buildID is incorrect (found e8801ed67d21c6be, expecting eb4173107d21c673).
This version of jextract is incompatible with this dump.
Failure detected during jextract, see previous message(s).
```

Així mateix, aquest missatge també es genera si executeu el Java amb la JVM estàndard, però utilitzeu l'opció **-Xrealtime** en processar l'abocament de memòria amb **jextract**.

Ordres disponibles a jdmpview

jdmpview és una eina interactiva de línia d'ordres que serveix per explorar la informació d'un abocament de memòria del sistema de la JVM i efectuar diverses funcions d'anàlisi.

info jitm
Mostra els mètodes compilats de JIT i AOT i les seves adreces:
- Nom del mètode i signatura
- Adreça inicial del mètode
- Adreça final del mètode

Per veure la resta d'opcions de l'ordre, consulteu la guia de l'usuari de l'IBM SDK for Java 7.

Traça de les aplicacions Java i la JVM

La traça de JVM és un recurs de traça que es proporciona en l'IBM WebSphere Real Time for RT Linux amb un efecte mínim sobre el rendiment. A la majoria dels casos, les dades de traça es mantenen en un format binari compacte que es pot formatar amb el formatador Java subministrat.

La traça està habilitada per defecte, juntament amb un petit conjunt de punts de traça que van a les memòries intermèdies. Podeu habilitar els punts de traça en temps d'execució mitjançant la utilització de nivells, components, noms de grup o identificadors de punts de traça individuals.

La guia de l'usuari d'IBM SDK for Java 7 conté informació detallada sobre les aplicacions de traça, per exemple:
- De què es pot fer el seguiment
- Tipus de punts de traça
- Traça per defecte
- Enregistrament de les dades de traça
- Control de la traça
- Traça de les aplicacions Java
- Traça dels mètodes Java

En fer la traça de l'IBM WebSphere Real Time for RT Linux, heu d'invocar correctament la JVM en temps real quan incloeu les opcions de traça. Per exemple, quan especifiqueu les opcions de traça, escriviu:

```
java -Xrealtime -Xtrace:<opcions>
```

Podeu obtenir la informació de l'IBM SDK for Java 7 aquí: Traça de les aplicacions Java i la JVM.

Determinació de problemes de JIT i AOT

Podeu utilitzar les opcions de línia d'ordres perquè resulti més fàcil diagnosticar problemes de compilador JIT i AOT i ajustar el rendiment.

Tot i que l'IBM WebSphere Real Time for RT Linux comparteix diversos components comuns amb l'IBM SDK for Java 7, el comportament de JIT i AOT és diferent. En aquest apartat es descriu la resolució de problemes de JIT i AOT a l'IBM WebSphere Real Time for RT Linux.

Diagnosi d'un problema de JIT o AOT

En ocasions els codis de bytes vàlids es compilen en codi natiu no vàlid, cosa que fa que el programa Java falli. Si determineu que el compilador JIT o AOT és defectuós i, si és així, , *on* hi ha el defecte, podeu proporcionar una ajuda valuosa a l'equip de servei tècnic de Java.

Quant a aquesta tasca

Per determinar els mètodes que es compilen quan s'omple la memòria cau de classes compartida, utilitzeu l'opció **-Xaot:verbose** a la línia de l'ordre admincache. Per exemple:

```
admincache -Xrealtime -Xaot:verbose -populate -aot my.jar -cp <la meva classpath>
```

En aquest apartat es descriu com es pot determinar si el problema està relacionat amb el compilador. En aquest apartat també se suggereixen algunes solucions temporals i tècniques de depuració per resoldre els problemes relacionats amb el compilador.

Inhabilitació del compilador JIT o AOT:

Si sospiteu que es produeix algun problema al compilador JIT o AOT, inhabiliteu la compilació per comprovar si el problema es continua produint. Si el problema encara es produeix, ja sabreu que el compilador no n'és la causa.

Quant a aquesta tasca

El compilador JIT està habilitat per defecte. El compilador AOT també està habilitat, però, no està actiu si no s'han habilitat les classes compartides. Per motius d'eficàcia, no tots els mètodes d'una aplicació Java es compilen. La JVM manté un recompte de crides per a cada mètode de l'aplicació; cada vegada que es crida i s'interpreta un mètode, el recompte de crides del mètode en qüestió augmenta. Quan el recompte arriba al llindar de compilació, el mètode es compila i s'executa de forma nativa.

El mecanisme de recompte de crides estén la compilació de mètodes al llarg de tota la vida útil d'una aplicació i assigna una prioritat més alta als mètodes que s'utilitzen més sovint. És possible que alguns mètodes que no s'utilitzen gaire sovint no es compilin mai. Com a resultat d'això, quan un programa Java falla, el problema pot ser al compilador JIT o AOT o pot ser a qualsevol altre punt de la JVM.

El primer pas per diagnosticar l'error és determinar *on* és el problema. Per fer-hi, cal que executeu primer el programa Java en mode interpretat pur (és a dir, amb el

compilador JIT i AOTinhabilitats inhabilitat).

Procediment

1. Elimineu les opcions **-Xjit** i **-Xaot** (i els paràmetres que les acompanyen) de la línia d'ordres.
2. Utilitzeu l'opció de línia d'ordres **-Xint** per inhabilitar el compilador JIT i AOT. Per motius de rendiment, no utilitzeu l'opció **-Xint** en un entorn de producció.

Què cal fer posteriorment

L'execució del programa Java amb la compilació inhabilitada duu a una de les situacions següents:

- L'error no desapareix. El problema no és al compilador JIT o AOT. En alguns casos, el programa pot començar a fallar d'una altra manera; el problema, tot i això, no està relacionat amb el compilador.
- L'error desapareix. El més probable és que el problema sigui al compilador JIT o AOT.

  Si no utilitzeu classes compartides, el compilador JIT té un error. Si utilitzeu classes compartides, cal que determineu quin compilador té l'error executant l'aplicació només amb la compilació JIT habilitada. Executeu l'aplicació amb l'opció **-Xnoaot** en lloc de l'opció **-Xint**. Això duu a una de les situacions següents:
  - L'error no desapareix. El problema és al compilador JIT. També podeu utilitzar l'opció **-Xnojit** en lloc de l'opció **-Xnoaot** per assegurar-vos que només el compilador JIT tingui un error.
  - L'error desapareix. El problema és al compilador AOT.

Inhabilitació selectiva del compilador JIT:

Si l'error del programa Java apunta a un problema amb el compilador JIT, podeu provar de delimitar encara més el problema.

Quant a aquesta tasca

Per defecte, el compilador JIT optimitza mètodes a diferents nivells d'optimització. S'apliquen diferents seleccions d'optimitzacions a diferents mètodes, segons els seus recomptes de trucades. Els mètodes que es criden amb més freqüència s'optimitzen a nivells superiors. En canviar els paràmetres del compilador JIT podeu controlar el nivell d'optimització al quals s'optimitzen els mètodes. Podeu determinar si l'optimitzador té un error i, si el té, quina optimització és problemàtica.

Especifiqueu els paràmetres JIT com a llista separada per comes afegida a l'opció **-Xjit**. La sintaxi és **-Xjit**:<*paràm1*>,<*paràm2*>=<*valor*>. Per exemple:

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

executa el programa HelloWorld, habilita la sortida detallada del JIT i fa que el JIT generi el codi natiu sense realitzar optimitzacions.

Seguiu aquests passos per determinar quina part del compilador provoca l'error:

Procediment

1. Definiu el paràmetre JIT **count=0** per canviar el llindar de compilació a zero. Aquest paràmetre fa que cada mètode Java es compili abans que s'executi.

Utilitzeu **count=0** només quan diagnostiqueu problemes, perquè es compilen molts més mètodes, incloent-hi els mètodes que s'utilitzen poc sovint. La compilació addicional utilitza més recursos informàtics i provoca que l'aplicació vagi més lenta. Amb **count=0**, l'aplicació falla immediatament quan s'arriba a una àrea de problemes. En alguns casos, si s'utilitza **count=1**, l'error es pot reproduir de manera més fiable.

2. Afegiu **disableInlining** als paràmetres de compilador JIT. **disableInlining** inhabilita la generació de codi més gran i complex. Si el problema ja no es produeix, utilitzeu **disableInlining** com a solució temporal mentre l'equip de servei tècnic de Java analitza i corregeix el problema del compilador.
3. Reduïu els nivells d'optimització afegint el paràmetre **optLevel** i executeu una altra vegada el paràmetre fins que ja no es produeixi l'error, o fins que arribeu al nivell "noOpt". En el cas d'un problema de compilador JIT, comenceu amb "scorching" i continueu en ordre descendent per la llista. Els nivells d'optimització són en ordre descendent:
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold
   f. noOpt

Què cal fer posteriorment

Si un d'aquests paràmetres fa que l'error desaparegui, teniu una solució temporal que podeu utilitzar. Aquesta solució és temporal mentre l'equip de servei de Java analitza i corregeix el problema del compilador. Si en eliminar **disableInlining** de la llista de paràmetre de JIT l'error no torna a aparèixer, feu-ho per millorar el rendiment. Seguiu les instruccions de "Ubicació del mètode que falla" per millorar el rendiment de la solució temporal.

Si l'error encara es produeix al nivell d'optimització "noOpt", cal que inhabiliteu el compilador JIT com a solució temporal.

Ubicació del mètode que falla:

Quan hàgiu determinat el nivell d'optimització més baix al qual el compilador JIT o AOT ha de compilar els mètodes per activar l'error, podeu esbrinar quina part del programa Java, quan es compila, ocasiona l'error. Tot seguit, podeu donar instruccions al compilador per limitar la solució temporal per a un mètode, una classe o un paquet específic, cosa que permetrà que el compilador compili la resta del programa com es fa habitualment. En el cas d'errors de compilador JIT, si l'error es produeix amb **-Xjit:optLevel=noOpt**, també podeu donar instruccions al compilador perquè no compili els mètodes que ocasionen l'error.

Abans de començar

Si veieu una sortida d'error com la d'aquest exemple, podeu utilitzar-la per identificar el mètode que falla:

```
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;
```

Les línies importants són:

vmState=0x00000000
Indica que el codi que ha fallat no era codi d'execució de la JVM.

Module= o Module_base_address=
No és a la sortida (pot estar en blanc o amb un valor zero) perquè JIT ha compilat el codi, i fora de qualsevol DLL o biblioteca.

Compiled_method=
Indica el mètode Java per al qual s'ha generat el codi compilat.

Quant a aquesta tasca

Si la sortida no indica el mètode que falla, seguiu els passos per identificar el mètode que falla:

Procediment

1. Executeu el programa Java amb els paràmetres JIT **verbose** i **vlog=**<*nom_fitxer*> afegits a l'opció **-Xjit** o **-Xaot**. Amb aquests paràmetres el compilador enumera els mètodes compilats en un fitxer de registre anomenat <*nom_fitxer*>.<*data*>.<*hora*>.<*pid*>, anomenat també *fitxer de límit*. Un fitxer de límit habitual conté línies que corresponen als mètodes compilats, com ara:

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

   El compilador ignora les línies que no comencen amb el signe més als passos següents i podeu eliminar-les del fitxer. Els mètodes per als quals es carrega el codi AOT de la memòria cau de classes compartida comencen amb `+ (AOT load)`.
2. Torneu a executar el programa amb el paràmetre JIT o AOT **limitFile=**(<*nom_fitxer*>,<*m*>,<*n*>), on <*nom_fitxer*> és el camí d'accés al fitxer de límit i <*m*> i <*n*> són números de línia que indiquen el primer i el darrer mètode del fitxer de límit que s'han de compilar. El compilador només compila els mètodes que s'enumeren a les línies de <*m*> a <*n*> al fitxer de límit. Els mètodes que no s'enumeren al fitxer de límit i els mètodes que s'enumeren a línies que no formen part de l'abast no es compilen i no es carrega el codi AOT de la memòria cau de dades compartides corresponent a aquests mètodes. Si el programa ja no falla, un o més dels mètodes que heu eliminat a la darrera iteració ha d'haver estat la causa de l'error.
3. Repetiu aquest procés amb diferents valors per a <*m*> i <*n*>, tantes vegades com calgui, per trobar el conjunt mínim de mètodes que es poden compilar per activar l'error. Dividint entre dos el nombre de línies seleccionades cada vegada podeu efectuar una cerca binària del mètode que falla. Sovint podeu reduir el fitxer a una sola línia.

Què cal fer posteriorment

Quan hàgiu localitzar el mètode que falla, podeu inhabilitar el compilador JIT o AOT només per al mètode que falla. Per exemple, si el mètode java/lang/Math.max(II)I fa que el programa falli quan es compila en mode JIT i **optLevel=hot**, podeu executar el programa amb:

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

per compilar només el mètode que falla en un nivell d'optimització “warm”, però compilar la resta de mètodes de la manera habitual.

Si un mètode falla quan es compila en mode JIT a “noOpt”, podeu excloure'l totalment de la compilació, utilitzant el paràmetre **exclude**={*<method>*}:

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

Si un mètode fa que el programa falli quan es carrega codi AOT de la memòria cau de dades compartides, excloeu el mètode de la càrrega AOT mitjançant el paràmetre **exclude**={*<method>*}:

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

Els mètodes AOT només es compilen a la memòria cau de classes compartida durant el pas d'emplenament d'**admincache**. El millor per al diagnòstic de problemes amb aquests mètodes és evitar la càrrega AOT.

Identificació d'errors de compilació JIT i AOT:

En cas d'error de compilador JIT, analitzeu la sortida d'error per determinar si es produeix un error quan el compilador JIT intenta compilar un mètode.

Si hi ha una fallada de la JVM i podeu veure que l'error s'ha produït a la biblioteca (`libj9jit26.so`), pot ser que el compilador JIT hagi fallat mentre s'intentava compilar un mètode.

Si veieu una sortida d'error com la d'aquest exemple, podeu utilitzar-la per identificar el mètode que falla:

```
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;)
```

Les línies importants són:

vmState=0x00050000
: Indica que el compilador JIT està compilant codi. Per obtenir una llista de números de codi de vmState, consulteu la taula d'etiquetes del Javadump a la guia de l'usuari de l'IBM SDK for Java 7, http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html.

Module=/home/test/sdk/jre/bin/libj9jit26.so
: Indica que l'error s'ha produït a `libj9jit26.so`, el mòdul de compilador JIT.

Method_being_compiled=
: Indica que s'està compilant el mètode Java.

Si la sortida no indica el mètode que falla, utilitzeu l'opció **verbose** amb els paràmetres addicionals següents:

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

Aquests paràmetres **verbose** notifiquen quan JIT o el compilador AOT comença a compilar un mètode i quan acaba. Si JIT o el compilador AOT falla en un mètode concret (és a dir, comença la compilació però falla abans de poder acabar), utilitzeu el paràmetre **exclude** per excloure'l de la compilació JIT o AOT (consulteu "Ubicació del mètode que falla" a la pàgina 130). En el cas de problemes amb la compilació AOT, destruïu la memòria cau de classes compartida utilitzant l'opció **exclude**. Si l'exclusió del mètode impedeix la fallada, heu trobat una solució temporal que podeu utilitzar mentre l'equip de servei tècnic corregeix el problema.

Identificació dels errors de compilació AOT en mode no en temps real:

La determinació dels problemes d'AOT en mode no en temps real és molt similar a la determinació de problemes de JIT.

Quant a aquesta tasca

Com passa amb el JIT, executeu primer l'aplicació amb **-Xnoaot**, cosa que garanteix que el codi AOT no s'utilitzi quan s'executi l'aplicació.

Si això corregeix el problema, torneu a crear els fitxer jar d'AOT utilitzant la mateixa tècnica que es descriu a "Ubicació del mètode que falla" a la pàgina 130 i proporcioneu l'opció **-Xaot** en el moment del muntatge de l'AOT, i no pas en el temps d'execució de l'aplicació.

Identificació d'errors de compilació AOT en mode de temps real:

La determinació de problemes AOT utilitza l'eina admincache per localitzar el problema.

Quant a aquesta tasca

A diferència dels errors de compilació JIT, que es produeixen en temps d'execució de l'aplicació, els errors de compilació AOT es produeixen durant el pas d'emplenament de dades de l'eina admincache.

Per saber on es produeix el problema, executeu l'eina admincache amb l'opció **-Xnoaot**. D'aquesta manera us assegureu que l'aplicació no s'executa amb codi compilat de manera anticipada.

Si el problema es resol mitjançant l'opció **-Xnoaot**, analitzeu la sortida de la fallada original. La sortida proporciona informació que identifica quin mètode és la causa del problema. Busqueu una línia semblant a aquesta:

```
Method_being_compiled=myAppClass.main(Ljava/lang/String;)V
```

Per evitar el problema, aquest mètode s'ha d'excloure de compilació de manera anticipada. Per fer-ho, afegiu una opció a la línia de comandes de l'eina admincache, semblant a aquesta:

```
-Xaot:exclude={myAppClass.main(Ljava/lang/String;)V}
```

Aquesta exclusió evita la compilació AOT del mètode de problema.

Rendiment de les aplicacions d'execució breu

El compilador JIT d'IBM s'ha ajustat per a aplicació d'execució llarga que normalment s'utilitzen en un servidor. Podeu utilitzar l'opció de línia d'ordres **-Xquickstart** no en temps real per millorar el rendiment de les aplicacions d'execució breu, especialment per a les aplicacions a les quals el procés no es concentra en un nombre menor de mètodes.

-Xquickstart fa que el compilador JIT utilitzi un nivell d'optimització inferior per defecte i compili menys mètodes. La realització de menys compilacions més ràpidament pot millorar el temps d'inici de l'aplicació. Quan el compilador AOT està actiu (tant les classes compartides com la compilació AOT habilitades), **-Xquickstart** fa que tots els mètodes seleccionats per a la compilació es compilin en mode AOT, cosa que millora el temps d'inici de les execucions posteriors. Pot ser que **-Xquickstart** degradi el rendiment si s'utilitza amb aplicacions de llarga execució que contenen mètodes que utilitzen una gran quantitat de recursos de processament. La implementació de **-Xquickstart** està sotmesa a canvis a les futures versions.

També podeu provar de millorar els temps d'inici ajustant el llindar de JIT (amb proves i errors). Vegeu “Inhabilitació selectiva del compilador JIT” a la pàgina 129 per obtenir-ne més informació.

-Xquickstart no té cap efecte a l'ús del codi AOT amb **-Xrealtime**.

Comportament de la JVM durant períodes inactius

Podeu reduir els cicles de CPU que consumeix una JVM inactiva utilitzant l'opció **-XsamplingExpirationTime** per desactivar el fil de mostratge JIT.

El fil de mostratge JIT defineix el perfil de l'aplicació Java en execució per descobrir els mètodes que s'utilitzen habitualment. L'ús de memòria i processador del fil de mostratge es pot obviar i la freqüència de definició de perfils es redueix de forma automàtica quan la JVM està inactiva.

En algunes circumstàncies, és possible que vulgueu que una JVM inactiva no consumeixi cap cicle de CPU. Per fer-ho, especifiqueu l'opció **-XsamplingExpirationTime**<*temps*>. Definiu <*temps*> en el nombre de segons durant els quals voleu que s'executi el fil de mostratge. Utilitzeu aquesta opció amb cura; després que es desactivi, no podreu reactivar el fil de mostratge. Permeteu que el fil de mostratge s'executi durant prou temps per identificar optimitzacions importants.

Recollidor de diagnòstics

El recollidor de diagnòstics recopila els fitxers de diagnòstics de Java per a un esdeveniment de problemes.

Si es recullen els fitxers necessaris per al servei d'IBM, es pot reduir el temps que es triga en resoldre problemes notificats. La guia de l'usuari d'IBM SDK for Java 7 conté informació detallada sobre la utilització del Recollidor de diagnòstics.

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - The Diagnostics Collector.

Diagnòstics del recollidor de deixalles

En aquest apartat es descriu com diagnosticar els problemes de la recollida de deixalles.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil per diagnosticar problemes del recollidor de deixalles, com per exemple:

- Registre de la recollida de deixalles detallada
- Traça de la recollida de deixalles mitjançant **-Xtgc**

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Diagnòstics del recollidor de deixalles.

Als apartats següents es proporciona informació complementària del recollidor de deixalles Metronome de l'IBM WebSphere Real Time for RT Linux.

Resolució de problemes del recollidor de deixalles Metronome

Mitjançant les opcions de la línia d'ordres, podeu controlar la freqüència de la recollida de deixalles de Metronome, les excepcions de manca de memòria i el comportament de Metronome en crides explícites del sistema.

Utilització de la informació verbose:gc:

Podeu utilitzar l'opció **-verbose:gc** amb l'opció **-Xgc:verboseGCCycleTime=N** per escriure informació a la consola sobre l'activitat del recollidor de deixalles Metronome. No totes les propietats de la sortida **-verbose:gc** de la JVM estàndard es creen o s'apliquen a la sortida del Recollidor de deixalles Metronome.

Utilitzeu l'opció **-verbose:gc** per visualitzar l'espai mínim, màxim i de mitjana de l'emmagatzematge dinàmic. D'aquesta manera, podeu comprovar el nivell d'activitat i l'ús de l'emmagatzematge dinàmic i llavors ajustar els valors segons calgui. L'opció **-verbose:gc** escriu estadístiques de Metronome a la consola.

L'opció **-Xgc:verboseGCCycleTime=N** controla la freqüència de recuperació de la informació. Determina el temps en mil•lisegons en què s'aboquen a la memòria els resums. El valor per defecte per a N és 1000 mil•lisegons. El temps de cicle no vol dir que el resum s'aboqui a la memòria en aquell moment concret, sinó que indica quan passa el darrer esdeveniment de recollida de deixalles que compleix aquests criteris de temps. La recollida i visualització d'aquestes estadístiques pot augmentar els temps de pausa del recollidor de deixalles Metronome i, a mesura que N té un valor més petit, els temps de pausa poden esdevenir elevats.

Un quantum és un període únic d'activitat Recollidor de deixalles Metronome que provoca una interrupció o una pausa per a una aplicació.

Exemple de sortida verbose:gc

Introduïu:

```
java -Xrealtime -verbose:gc -Xgc:verboseGCCycleTime=N myApplication
```

Aquest exemple mostra la sortida inicial de verbose:gc, que conté la versió i els valors de recollida de deixalles:

```
<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.tboot.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>
```

Quan s'activa la recollida de deixalles, es produeix l'esdeveniment `trigger start`, seguit per qualsevol nombre d'esdeveniments `heartbeat`, i després, quan l'activador està satisfet, es produeix un esdeveniment `trigger end`. Aquest exemple mostra un cicle de recollida de deixalles com a sortida 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" />
```

Es poden produir els tipus següents d'esdeveniment:

<trigger-start ...>
Inici d'un cicle de recollida de deixalles, quan la quantitat de memòria utilitzada s'ha fet més gran que el llindar de l'activador. El llindar per defecte és el 50% de l'emmagatzematge dinàmic. L'atribut `intervalms` és l'interval entre l'esdeveniment `trigger end` anterior (amb id 1) i aquest esdeveniment `trigger start`.

<trigger-end ...>
Un cicle de recollida de deixalles ha reduït satisfactòriament la quantitat de memòria utilitzada per sota del llindar de l'activador. Si un cicle de recollida de deixalles ha finalitzat, però la memòria utilitzada no s'ha reduït per sota del llindar de l'activador, s'inicia un nou cicle de recollida de deixalles amb el mateix ID de context. Per a cada esdeveniment `trigger start`, hi ha l'esdeveniment `trigger end` corresponent amb el mateix ID de context. L'atribut `intervalms` és l'interval entre l'esdeveniment `trigger start` anterior i l'esdeveniment actual `trigger end`. Durant aquest temps, s'haurà completat un cicle de recollida de deixalles, o més d'un, fins que la memòria utilitzada caigui per sota del llindar de l'activador.

<gc-op id="28" type="heartbeat"...>
Esdeveniment periòdic que recopila informació (de la memòria i el temps) sobre els quantums de recollida de deixalles per al període de temps que cobreix. Un esdeveniment heartbeat només es pot produir entre una parella d'esdeveniments `trigger start` i `trigger end`, és a dir, mentre hi ha un cicle de recollida de deixalles actiu. L'atribut `intervalms` és l'interval entre l'esdeveniment heartbeat anterior (amb `id` -1) i aquest esdeveniment heartbeat.

<gc-op id="29" type="syncgc"...>
Esdeveniment de recollida de deixalles síncron (no determinista). Consulteu “Recollides de deixalles síncrones” a la pàgina 138

Tot seguit s'indica el significat de les etiquetes XML d'aquest exemple:

<quanta ...>
Resum dels temps de pausa de quantum durant l'interval de heartbeat, que inclou la durada de les pauses en mil•lisegons.

<free-mem type="heap" ...>
Resum de la quantitat d'espai lliure d'emmagatzematge dinàmic durant l'interval heartbeat, mostrat al final de cada quantum de recollida de deixalles.

<classunload-info classloadersunloaded=9 classesunloaded=156 />
Nombre de carregadors de classes i classes baixades durant l'interval heartbeat.

<references type="weak" cleared="660 />
Nombre i tipus d'objectes de referència Java que s'han esborrat durant l'interval heartbeat.

Nota:

- Si s'ha produït un quantum de recollida de deixalles a l'interval entre dos heartbeats, la memòria lliure només es mostreja al final d'aquest quantum. Per tant, les quantitats mínima, màxima i de mitjana donades al resum de heartbeat són totes iguals.
- Pot ser que l'interval entre dos esdeveniments heartbeat sigui significativament més gran que el temps de cicle especificat si l'emmagatzematge dinàmic no és prou ple per necessitar l'activitat de recollida de deixalles. Per exemple, si el vostre programa necessita una que l'activitat de recollida de deixalles s'executi només cada pocs segons, pot ser que veieu només un heartbeat cada pocs segons.
- Pot ser que l'interval sigui significativament més gran que el temps de cicle especificat perquè la recollida de deixalles no té feina en un emmagatzematge dinàmic que no està prou ple per garantir l'activitat de recollida de deixalles. Per exemple, si el vostre programa necessita una que l'activitat de recollida de deixalles s'executi només cada pocs segons, pot ser que veieu només un heartbeat cada pocs segons.

  Si es produeix un esdeveniment com ara una recollida de deixalles síncrona o un canvi de prioritat, els detalls de l'esdeveniment i els esdeveniments pendents, com ara esdeveniments heartbeat, es produeixen immediatament com a sortida.
- Si el quantum màxim de recollida de deixalles per a un període determinat és massa gran, és recomanable que reduïu la utilització objecti l'opció **-Xgc:targetUtilization**. Aquesta acció dóna al recollidor de deixalles més temps per treballar. També podeu incrementar la mida de l'emmagatzematge dinàmic mitjançant l'opció **-Xmx**. De la mateixa manera, si la vostra aplicació pot tolerar retards més llargs dels que es notifiquen actualment, podeu incrementar la utilització objectiu o reduir la mida de l'emmagatzematge dinàmic.
- La sortida es pot redirigir a un fitxer de registre, en lloc de redirigir-la a la consola, mitjançant l'opció **-Xverbosegclog:<fitxer>**; per exemple, **-Xverbosegclog:out** escriu la sortida de **-verbose:gc** al fitxer *out*.
- La prioritat que s'indica a `thread-priority` és la prioritat del fil del sistema operatiu subjacent, no una prioritat de fil Java.

Recollides de deixalles síncrones

També s'escriu una entrada al registre **-verbose:gc** quan es produeix una recollida de deixalles síncrona (no determinista). Aquest esdeveniment té tres possibles causes:

- Una crida `System.gc()` explícita al codi.
- La JVM es queda sense memòria i duu a terme una recollida de deixalles síncrona per evitar una condició OutOfMemoryError.

- La JVM es tanca durant una recollida de deixalles contínua. La JVM no pot cancel•lar la recollida; per tant, fa la recollida de manera asíncrona i llavors finalitza.

Tot seguit es mostra un exemple d'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>
```

L'exemple següent correspon a una entrada de recollida de deixalles síncrona com a conseqüència del tancament 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>
```

Les etiquetes XML i els atributs d'aquest exemple tenen aquest significat:

<gc-op id="9" type="syncgc" timems="6.439" ...
: Aquesta línia indica que el tipus d'esdeveniment és una recollida de deixalles síncrona. L'atribut `timems` és la duració de la recollida de deixalles síncrona en mil•lisegons.

<syncgc-info reason="..."/>
: Causa de la recollida de deixalles síncrona.

<free-mem-delta.../>
: Memòria d'emmagatzematge dinàmic Java lliure abans i després de la recollida de deixalles síncrona, en bytes.

<finalization .../>
: Nombre d'objectes que esperen la finalització.

<classunload-info .../>
: Nombre de carregadors de classes i classes baixades durant l'interval heartbeat.

<references type="weak" cleared="53" .../>
: Nombre i tipus d'objectes de referència Java que s'han esborrat durant l'interval heartbeat.

La recollida de deixalles sincrònica que es produeix a causa de condicions de manca de memòria o del tancament de la VM només es pot produir quan el recollidor de deixalles està actiu. Ha d'anar precedida per un esdeveniment `trigger start`, tot i que no es necessari que sigui de manera immediata. Pot ser que es produeixin alguns esdeveniments heartbeat entre un esdeveniment `trigger start` i l'esdeveniment `synchgc`. La recollida de deixalles causada per `System.gc()` es pot produir en qualsevol moment.

Traça de tots els quantums de recollida de deixalles

La traça dels quantums de recollida de deixalles individuals es pot fer habilitant els punts de traça `GlobalGCStart` i `GlobalGCEnd`. Aquests punts de traça es produeixen el principi i al final de tota l'activitat de recollida de deixalles Metronome, incloent-hi recollides de deixalles sincròniques. La sortida dels punts de traça serà semblant a aquesta:

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

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

Canvis de prioritat

A banda dels resums, s'escriu una entrada al registre **-verbose:gc** quan la prioritat del fil de recollidor de deixalles canvia (perquè l'aplicació ha canviat prioritats dels fils o perquè un o més fils de l'aplicació han finalitzat). La prioritat que s'indica és la prioritat del fil del sistema operatiu subjacent, no una prioritat de fil Java. L'exemple següent correspon a una entrada de canvi de prioritat de fil del recollidor de deixalles:

```
<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>
```

La traça dels canvis de prioritat es pot fer en temps real produint informació de punts de traça relacionada amb les prioritats dels fils del recollidor de deixalles. La sortida és semblant a aquesta:

```
15:58:25.493*0x8286e00    j9mm.102       - setGCThreadPriority() called with newGCThreadPriority = 1
```

Aquesta sortida es pot habilitar utilitzant l'ID, d'aquesta manera:
-Xtrace:iprint=tpnid{j9mm.102}

Entrades de manca de memòria

Quan una de les àrees de memòria es queda sense espai, s'escriu una entrada al registre **-verbose:gc** abans que es generi l'excepció OutOfMemoryError. Exemple d'aquesta sortida:

```
<out-of-memory id="71" timestamp="2011-07-23T08:32:51.435" memorySpaceName="Scoped"
  memorySpaceAddress="080EED9C"/>
```

Per defecte, es produeix un Javadump com a conseqüència d'una excepció OutOfMemoryError. Aquest abocament de memòria inclou informació sobre la memòria utilitzada pel programa. Conjuntament amb el valor `J9MemorySpace` proporcionat a la sortida de **-verbose:gc**, podeu utilitzar aquesta informació a l'abocament de memòria per identificar l'àrea de memòria en particular que s'ha quedat sense espai:

```
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)
```

A l'exemple anterior, l'ID de l'espai de memòria que es facilita a la sortida de -verbose:gc (0x080EED9C) es pot fer coincidir amb l'ID de l'àrea de memòria amb àmbit de l'abocament de memòria Java. Aquesta coincidència pot ser útil si teniu diversos àmbits i us cal identificar quin àmbit s'ha quedat sense memòria, perquè la sortida de -verbose:gc només indica si s'ha produït un error OutOfMemoryError a la memòria immortal, amb àmbit o d'emmagatzematge dinàmic.

Comportament del recollidor de deixalles Metronome en condicions de manca de memòria:

Per defecte, el recollidor de deixalles Metronome activa una recollida de deixalles il•limitada i no determinista quan la JVM es queda sense memòria. Per evitar el comportament no determinista, utilitzeu l'opció **-Xgc:noSynchronousGCOnOOM** per generar un error OutOfMemoryError quan la JVM es quedi sense memòria.

La recollida il•limitada per defecte s'executa fins que s'han recollit totes les deixalles possibles en una única operació. El temps de pausa necessari sol ser molts mil•lisegons més gran que el d'un quantum incremental de Metronome normal.

Informació relacionada

Ús de -Xverbose:gc per analitzar recollides de deixalles síncrones

Comportament del recollidor de deixalles Metronome en crides System.gc() explícites:

Si hi ha un cicle de recollida de deixalles en curs, el recollidor de deixalles Metronome completa el cicle de manera síncrona quan es crida `System.gc()`. Si no hi ha cap cicle de recollida de deixalles en curs, es duu a terme un cicle síncron complet quan es crida `System.gc()`. Utilitzeu `System.gc()` per netejar l'emmagatzematge dinàmic de manera controlada. Es tracta d'una operació no determinista perquè realitza una recollida de deixalles completa abans de tornar.

Algunes aplicacions criden programari de proveïdor que té crides System.gc() en els quals no es permet crear aquests retards no deterministes. Per inhabilitar totes les crides System.gc(), utilitzeu l'opció **-Xdisableexplicitgc**.

La sortida detallada de la recollida de deixalles per a una crida System.gc() té com a motiu "system garbage collect" i pot ser que sigui llarga:

```
<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>
```

Diagnòstics de classes compartides

Entendre com es poden produir els problemes de diagnòstic us ajuda a utilitzar el mode de classes compartides.

Per obtenir una introducció a les classes compartides, consulteu Ús compartit de les dades de classe entre JVM.

La guia de l'usuari de l'IBM SDK for Java 7 conté informació útil per diagnosticar problemes de classes compartides, com per exemple:

- Desplegament de classes compartides
- Tractament de la modificació del codi de bytes de temps d'execució
- Descripció de les actualitzacions dinàmiques
- Utilització de l'API d'ajuda de Java
- Descripció de la sortida dels diagnòstics de classes compartides
- Depuració de problemes amb les classes compartides

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Diagnòstics de classes compartides.

És possible que part del material de la guia de l'usuari de l'IBM SDK for Java 7 no sigui aplicable a l'IBM WebSphere Real Time for RT Linux. Concretament:

- En mode de temps real, les aplicacions només tenen accés de lectura a les memòries cau de classes compartides, però no accés d'escriptura.
- Les memòries cau es poden modificar de forma exclusiva utilitzant l'eina **admincache**.
- Les memòries cau no persistents no estan disponibles en mode de temps real.

Utilització de la JVMTI

JVMTI és una interfície de dos sentits que permet la comunicació entre la JVM i un agent natiu. Substitueix les interfícies JVMDI i JVMPI.

La JVMTI permet que altres proveïdors desenvolupin eines de depuració, definició de perfils i supervisió per a la JVM. La interfície conté mecanismes perquè l'agent notifiqui a la JVM els tipus d'informació que li calen. La interfície també proporciona un mitjà per rebre les notificacions rellevants. Es poden connectar diversos agents a una JVM en qualsevol moment.

La guia de l'usuari d'IBM SDK for Java 7 conté informació detallada sobre la utilització de la JVMTI, que inclou un apartat de referència de l'API sobre les extensions de JVMTI d'IBM.

Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Utilització de la JVMTI.

Utilització de l'estructura d'eines de diagnòstic per a Java

L'estructura d'eina de diagnòstic per a Java (DTFJ) és una interfície de programació d'aplicacions Java (API) d'IBM que s'utilitza per admetre la creació d'eines de diagnòstic Java. DTFJ funciona amb dades d'un abocament de memòria del sistema o Javadump.

La guia de l'usuari d'IBM SDK for Java 7 conté informació detallada sobre DTFJ. Seguiu aquest enllaç: Utilització de l'estructura d'eines de diagnòstic per a Java

Utilització de IBM Monitoring and Diagnostic Tools for Java - Health Center

IBM Monitoring and Diagnostic Tools for Java - Health Center és una eina de diagnòstic per supervisar l'estat d'una màquina virtual Java (JVM) en execució.

Trobareu informació sobre IBM Monitoring and Diagnostic Tools for Java - Health Center a developerWorks i a InfoCenter.

Capítol 10. Referència

En aquests temes s'indiquen les opcions i les biblioteques de classes que es poden utilitzar amb l'WebSphere Real Time for RT Linux

Opcions de línia d'ordres

Podeu especificar opcions a la línia d'ordres mentre inicieu Java. Les opcions per defecte s'han triat per al seu millor ús general.

Especificació d'opcions i propietats del sistema Java

Hi ha tres maneres d'especificar les propietats Java i les propietats del sistema.

Quant a aquesta tasca

Podeu especificar opcions Java i propietats del sistema segons diferents mètodes. Per ordre de preferència, són:

1. Especificar l'opció o propietat a la línia d'ordres. Per exemple:

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

2. Crear un fitxer que inclogui les opcions i especificar-ho a la línia d'ordres mitjançant l'opció **-Xoptionsfile=**<*nom_fitxer*>.

   Al fitxer d'opcions, especifiqueu cada opció en una línia nova; podeu utilitzar el caràcter '\' com a caràcter de continuació si voleu que una opció individual abasti diverses línies. Utilitzeu el caràcter '#' per definir les línies de comentari. No podeu especificar **-classpath** en un fitxer d'opcions. Tot seguit es mostra un exemple de fitxer d'opcions:

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

3. Crear una variable d'entorn anomenada **IBM_JAVA_OPTIONS** que conté les opcions. Per exemple:

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

L'última opció que especifiqueu a la línia d'ordres preval per sobre de la primera opció. Per exemple, si especifiqueu les opcions **-Xint -Xjit myClass**, l'opció **-Xjit** té preferència sobre **-Xint**.

Propietats del sistema

Les propietats del sistema estan disponibles per a les aplicacions, i ajuden a proporcionar informació sobre l'entorn d'execució.

com.ibm.jvm.realtime

Aquesta propietat permet que les aplicacions Java determinen si s'executen en un entorn WebSphere Real Time for RT Linux.

Si la vostra aplicació s'executa dintre de l'execució del IBM WebSphere Real Time for RT Linux, i s'ha iniciat amb l'opció **-Xrealtime**, la propietat **com.ibm.jvm.realtime** té el valor "hard".

Si la vostra aplicació s'executa dintre de l'execució de l'IBM WebSphere Real Time for RT Linux, però no s'ha iniciat amb l'opció **-Xrealtime**, la propietat **com.ibm.jvm.realtime** no es defineix.

Si la vostra aplicació s'executa dintre de l'execució del IBM WebSphere Real Time, la propietat **com.ibm.jvm.realtime** té el valor "soft".

Opcions estàndard

Les definicions per a les opcions estàndard.

-agentlib:<*nom_biblioteca*>**[=**<*opcions*>**]**
Carrega una biblioteca d'agents nadius <*nom_biblioteca*>; per exemple **-agentlib:hprof**. Per obtenir més informació, especifiqueu **-agentlib:jdwp=help** i **-agentlib:hprof=help** a la línia d'ordres.

-agentpath:*nom_biblioteca***[=**<*opcions*>**]**
Carrega una biblioteca d'agents nadius pel nom de camí d'accés complet.

-assert Imprimeix l'ajuda sobre les opcions relacionades amb assert.

-cp o -classpath <*directoris i fitxers .zip o .jar separats per* **:**>
Estableix el camí de cerca per a les classes i recursos de l'aplicació. Si no utilitzeu **-classpath** i **-cp** i no s'ha definit **CLASSPATH**, la variable classpath de l'usuari és, per defecte, el directori actual (.).

-D<nom_propietat>=<*valor*>
Estableix una propietat del sistema.

-help or -?
Imprimeix un missatge d'ús.

-javaagent:<*camí_accés_jar*>**[=**<*opcions*>**]**
Carrega un agent de llenguatge de programació Java. Per obtenir més informació, consulteu la documentació de l'API java.lang.instrument.

-jre-restrict-search
Inclou els JRE privats en la cerca de la versió.

-no-jre-restrict-search
Exclou els JRE privats d'usuari a la cerca de versió.

-showversion
Imprimeix la versió del producte i continua.

-verbose:[*class***,***gc***,***dynload***,***sizes***,***stack***,***jni***]**
Habilita la sortida detallada.

-verbose:class
Escriu una entrada a l'stderr per a cada classe que està carregada.

-verbose:gc
Consulteu "Utilització de la informació verbose:gc" a la pàgina 135.

-verbose:dynload
Proporciona informació detallada mentre la JVM carrega cada classe, incloent-hi:
- El nom de classe i el paquet
- Per als fitxers de classes que són al fitxer `.jar`, nom i camí d'accés del directori del fitxer `.jar`.
- Detalls sobre la mida de la classe i el temps que es triga a carregar la classe

Les dades s'escriuen a stderr. Tot seguit es mostra un exemple de la sortida:

```
<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: Les classes carregades des de la memòria cau de classes compartida no apareixen a la sortida de **-verbose:dynload**. Utilitzeu **-verbose:class** per obtenir informació sobre aquestes classes.

-verbose:sizes
Escriu informació a la sortida estàndard (stderr) que descriu la quantitat de memòria per a les piles i els emmagatzematges dinàmics de la JVM.

-verbose:stack
Escriu informació a la sortida estàndard que descriu l'ús de la pila Java i C.

-verbose:jni
Escriu informació a l'stderr que descriu els serveis JNI que crida l'aplicació i la JVM.

-version
Imprimeix la informació sobre la versió per al mode que no és en temps real. Quan s'utilitza amb l'opció -Xrealtime, imprimeix la informació de versió per al mode en temps real.

-version:*<valor>*
Requereix que s'executi la versió especificada.

-X Imprimeix l'ajuda sobre les opcions no estàndard.

Opcions no estàndard

Les opcions que tenen el prefix **-X** no són estàndard i estan subjectes a canvis sense notificació prèvia.

La guia de l'usuari d'IBM SDK for Java 7 conté informació detallada sobre les opcions no estàndard. Podeu trobar aquesta informació aquí: IBM SDK for Java 7 - Opcions de línia d'ordres.

Als apartats següents es proporciona informació complementària de l'IBM WebSphere Real Time for RT Linux.

Opcions en temps real

La definició de l'opció **-Xrealtime** s'utilitza a WebSphere Real Time for RT Linux.

Les opcions **-X** següents són aplicables per a l'entorn WebSphere Real Time for RT Linux.

-Xrealtime
Inicia el mode en temps real. És necessari si voleu executar el Recollidor de deixalles Metronome i utilitzar serveis Especificació de temps real per a Java (RTSJ). Si no especifiqueu aquesta opció, la JVM s'inicia en un mode de temps no real equivalent a l'IBM SDK and Runtime Environment for Linux Platforms, Java 2 Technology, versió 7.

L'opció **-Xrealtime** es pot intercanviar per **-Xgcpolicy:metronome**. Podeu especificar qualsevol de les dues per obtenir el mode en temps real.

Opcions d'AOT

Definicions per a les opcions d'AOT.

Finalitat

No s'especifica cap opció:
S'executa amb el codi d'intèrpret i el compilat dinàmicament. Si es descobreix el codi AOT, no s'utilitza. En comptes d'això, es compila dinàmicament segons calgui. Això és especialment útil per a aplicacions en que no són en temps real i per a algunes aplicacions en temps real. Aquesta opció proporciona un rendiment i uns resultats òptims, però pot patir de retards no deterministes en temps d'execució quan es produeix la compilació.

-Xjit: Aquesta opció és igual que el valor per defecte.

-Xint: Només executa l'intèrpret, ignora el codi escrit per a AOT que pot ser que es detecti en un fitxer jar precompilat i no executa el compilador dinàmic. Aquest mode no es necessita sovint, tret de quan cal depurar problemes que se sospita que estan relacionats amb la compilació o per a aplicacions per lots molt curtes que no es beneficien de la compilació.

-Xnojit:
Executa l'intèrpret i utilitza codi escrit per a l'AOT si es troba en un fitxer jar precompilat. No executa el compilador dinàmic. Aquest mode funciona bé per a algunes aplicacions en temps real en les quals voleu estar segurs de que no es produeixen retards no deterministes en temps d'execució per causa de la compilació. El codi escrit per a AOT només es pot utilitzar quan s'executa amb l'opció **-Xrealtime**. No s'admet quan s'executa en una JVM estàndard, és a dir, quan no s'especifica **-Xrealtime**.

Exemple

```
java -Xrealtime -Xnojit outputtest.jar.
```

Opcions del recollidor de deixalles Metronome

Definicions de les opcions del recollidor de deixalles Metronome.

-Xgc:immortalMemorySize=*mida*
Especifica la mida de l'àrea d'emmagatzematge dinàmic immortal. El valor per defecte és 16 MB.

-Xgc:scopedMemoryMaximumSize=*mida*
Especifica la mida de l'àrea d'emmagatzematge dinàmic amb àmbit. El valor per defecte és 8 MB.

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM
Un dels casos en què es produeix la recollida de deixalles és quan l'emmagatzematge dinàmic es queda sense memòria. Si no hi ha més espai lliure a l'emmagatzematge dinàmic, l'ús **-Xgc:synchronousGCOnOOM** atura l'aplicació mentre el procés de recollida de deixalles elimina els objectes no utilitzats. Si l'espai lliure es torna a exhaurir, considereu la possibilitat de reduir l'ús objectiu per permetre més temps perquè finalitzi la recollida de deixalles. Definir **-Xgc:nosynchronousGCOnOOM** implica que, quan la memòria de l'emmagatzematge dinàmic estigui plena, l'aplicació s'aturarà i emetrà un missatge de manca de memòria. El valor per defecte és **-Xgc:synchronousGCOnOOM**.

-Xnoclassgc
Inhabilita la recollida de deixalles de classe. Aquesta opció desactiva la

recollida de deixalles d'emmagatzematge associada amb les classes Java que ja no utilitza la JVM. El comportament per defecte és **-Xnoclassgc**.

-Xgc:targetUtilization=*N*
Defineix l'ús de l'aplicació a N%; el recollidor de deixalles prova d'utilitzar com a màxim (100-N)% de cada interval de temps. Els valors raonables corresponen a l'interval 50-80%. Les aplicacions amb índex d'assignació més baixos pot ser que es puguin executar al 90%. El valor per defecte és 70%.

Aquest exemple mostra que la mida màxima de la memòria d'emmagatzematge dinàmic és 30 MB. El recollidor de deixalles intenta utilitzar el 25% de cada interval de temps perquè l'ús objectiu per a l'aplicació és el 75%.

```
java -Xrealtime -Xmx30m -Xgc:targetUtilization=75 Test
```

-Xgc:threads=*N*
Especifica el nombre de fils de recollida de deixalles que cal executar. El valor per defecte és 1.

-Xgc:verboseGCCycleTime=N
N és el temps en mil•lisegons en què s'hauria d'abocar a memòria la informació de resum.

Nota: El temps de cicle no vol dir que la informació de resum s'aboqui a la memòria en aquell moment concret, sinó que indica quan passa el darrer esdeveniment de recollida de deixalles que compleix aquests criteris de temps.

-Xmx<*mida*>
Especifica la mida de l'emmagatzematge dinàmic Java. A diferència d'altres estratègies de recollida de deixalles, la recollida de deixalles Metronome en temps real no admet l'expansió de l'emmagatzematge dinàmic. No hi ha cap opció per a la mida inicial o màxima de l'emmagatzematge dinàmic. Podeu especificar només la mida d'emmagatzematge dinàmic màxima.

-Xthr:metronomeAlarm=os*xx*
Controla la prioritat d'execució del fil d'alarma del Recollidor de deixalles Metronome.

xx és un nombre d'11 a 89 que especifica la prioritat d'execució del fil d'alarma de Metronome. Cal anar en compte en modificar la prioritat del sistema operatiu a la qual s'executa el fil d'alarma. Si especifiqueu una prioritat de sistema operatiu inferior a la de qualsevol fil en temps real, obtindreu errors OutOfMemory perquè el recollidor de deixalles s'acaba executant a una prioritat inferior a la dels fils en temps real que assignen deixalles. El fil d'alarma del Recollidor de deixalles Metronome per defecte s'executa a una prioritat de sistema operatiu 89.

Paràmetres per defecte de la JVM

Els valors per defecte s'apliquen a la JVM en temps real quan no es fa cap canvi a l'entorn en el qual s'executa la JVM. Es mostren valors comuns com a referència.

Els valors per defecte es poden modificar utilitzant variables d'entorn o paràmetres de la línia d'ordres en arrencar al JVM. A la taula es mostren alguns dels valors de JVM comuns. La darrera columna indica com podeu canviar el comportament, on s'apliquen aquestes claus:

- **e**: paràmetre que només controla la variable d'entorn.

- **c**: paràmetre que només controla un paràmetre de línia d'ordres.
- **ec**: paràmetre controlat per la variable d'entorn i el paràmetre de la línia d'ordres, on el paràmetre de la línia d'ordres té preferència.

La informació es proporciona com a referència ràpida i no és completa.

Paràmetre de JVM	Valor per defecte	Paràmetre afectat per
Javadumps	Habilitat	ec
Javadumps amb manca de memòria	Habilitat	ec
Heapdumps	Inhabilitat	ec
Heapdumps amb manca de memòria	Habilitat	ec
Sysdumps	Habilitat	ec
Ubicació on es generen els fitxers d'abocament de memòria	Directori actual	ec
Sortida detallada	Inhabilitat	c
Cerca a la variable classpath d'arrencada	Inhabilitat	c
Comprovacions de JNI	Inhabilitat	c
Depuració remota	Inhabilitat	c
Comprovacions de compliment estricte	Inhabilitat	c
Inici ràpid	Inhabilitat	c
Servidor d'informació de depuració remot	Inhabilitat	c
Senyalització reduïda	Inhabilitat	c
Encadenament de gestors de senyals	Habilitat	c
Classpath	No definit	ec
Compartició de dades de classe	Inhabilitat	c
Suport per a accessibilitat	Habilitat	e
compilador JIT	Habilitat	ec
Compilador d'AOT (la JVM no utilitza AOT si no s'habiliten les classes compartides)	Habilitat	c
Opcions de depuració de JIT	Inhabilitat	c
Mida màxima Java2D dels tipus de lletra amb negreta algorítmica	14 punts	e
Mapes de bits representats Java2D en tipus de lletra escalables	Habilitat	e
Rasterització de tipus de lletra lliure Java2D	Habilitat	e
Tipus de lletra AWT Java2D	Inhabilitat	e
Entorn local per defecte	Cap	e
Temps d'espera abans d'iniciar el connector	zero	e
Directori temporal	/tmp	e
Redirecció de connector	Cap	e
Commutació d'IM	Inhabilitat	e
Modificadors d'IM	Inhabilitat	e
Model de fil	N/D	e
Mida de pila inicial per a fils Java de 32 bits. Utilitzeu: **-Xiss<mida>**	2 KB	c

Paràmetre de JVM	Valor per defecte	Paràmetre afectat per
Mida de pila màxima per a fils Java de 32 bits. Utilitzeu: **-Xss<mida>**	256 KB	c
Mida de pila per a fils de sistema operatiu de 32 bits. Utilitzeu: **-Xmso<mida>**	256 KB	c
Mida d'emmagatzematge dinàmic inicial. Utilitzeu: **-Xms<mida>**	64 MB	c
Mida màxima d'emmagatzematge dinàmic Java. Utilitzeu: **-Xmx<mida>**	La meitat de la memòria disponible amb un mínim de 16 MB i un màxim de 512 MB	c
Utilització de l'interval de temps de destinació per a una aplicació. El recollidor de deixalles intenta utilitzar la resta. Utilitzeu **-Xgc:targetUtilization=<percentatge>**.	70%	c
Nombre de fils del recollidor de deixalles que cal executar. Utilitzeu **-Xgc:threads=<valor>**.	1	c
Quantitat màxima de memòria que es pot assignar a memòries amb àmbit al mode **-Xrealtime**. Utilitzeu **-Xgc:scopedMemoryMaximumSize=<mida>**.	8 MB	c
Defineix la mida de l'àrea de memòria immortal al mode **-Xrealtime**. Utilitzeu **-Xgc:immortalMemorySize=<mida>**.	16 MB	c

Nota: "memòria disponible" és la quantitat de memòria real (física) o el valor **RLIMIT_AS**, el que sigui el valor més baix.

Biblioteques de classes de WebSphere Real Time for RT Linux

Una referència per a les biblioteques de classes Java que s'utilitzen al WebSphere Real Time for RT Linux.

Les biblioteques de classes Java que s'utilitzen al WebSphere Real Time for RT Linux es descriuen a http://www.rtsj.org/specjavadoc/book_index.html.

Execució amb el kit TCK

Si executeu el kit de compatibilitat de tecnologia (TCK) Especificació de temps real per a Java (RTSJ) amb el WebSphere Real Time for RT Linux, heu d'incloure `demo/realtime/TCKibm.jar` a la variable classpath per tal que les proves es puguin realitzar correctament.

El `TCKibm.jar` inclou la classe **VibmcorProcessorLock**, que és l'extensió d'IBM per a la classe TCK.ProcessorLock. Aquesta classe proporciona un comportament d'un processador necessari en un conjunt petit de proves TCK. Per obtenir més informació sobre la classe TCK.ProcessorLock i les extensions específiques del proveïdor per a aquesta classe, llegiu el fitxer readme que s'inclou amb la distribució del TCK.

Capítol 11. Avisos

Aquesta informació s'ha desenvolupat per a productes i serveis oferts als EUA. És possible que IBM no ofereixi els productes, serveis o característiques que es mencionen dins aquest document en altres països. Consulteu el vostre representant local d'IBM si voleu obtenir més informació sobre els productes i els serveis disponibles actualment a la vostra zona. Qualsevol referència a un producte, programa o servei d'IBM no té la intenció de declarar o implicar que només es pot utilitzar aquell producte, programa o servei. En el seu lloc es podria utilitzar qualsevol producte, programa o servei equivalent que no infringeixi cap llei de propietat intel•lectual d'IBM. Tanmateix, és responsabilitat de l'usuari avaluar i verificar el funcionament de qualsevol producte, programa o servei que no sigui d'IBM.

IBM pot tenir patents o sol•licituds pendents de patent que tractin el tema d'aquest document. El fet de disposar d'aquest document no us dóna cap llicència sobre aquestes patents. Podeu enviar per escrit al fabricant les consultes referents a les llicències.

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EUA

Per fer consultes sobre llicències pel que fa a la informació de DBCS (doble byte), poseu-vos en contacte amb el Departament de Propietat Intel•lectual d'IBM del vostre país, o envieu les consultes, per escrit, a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japó

El paràgraf següent no s'aplica al Regne Unit ni a cap altre país on aquestes disposicions entrin en conflicte amb la legislació local.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA AQUESTA PUBLICACIÓ "TAL QUAL" SENSE CAP GARANTIA, NI EXPLÍCITA NI IMPLÍCITA, INCLOENT-HI, ENTRE D'ALTRES, LES GARANTIES RELATIVES AL NO INFRINGIMENT, A LA COMERCIALITZACIÓ O A L'ADEQUACIÓ PER A UNA FINALITAT DETERMINADA. Alguns països no permeten la renúncia de les garanties implícites o explícites en determinades transaccions i, per tant, pot ser que el paràgraf anterior no s'apliqui en el vostre cas.

Pot ser que la publicació inclogui incorreccions tècniques o errors tipogràfics. Es realitzaran modificacions periòdiques pel que fa a la informació de la publicació; aquestes modificacions s'incorporaran a les noves edicions de la informació. IBM pot efectuar millores i/o canvis en els productes i/o programes descrits en aquesta informació en qualsevol moment sense cap avís previ.

Les referències que apareixen en aquesta documentació a llocs Web que no són d'IBM es proporcionen a tall informatiu i de cap manera no es pretén aprovar aquests llocs Web. El material d'aquests llocs web no forma part del material per a aquest producte d'IBM i la utilització d'aquests llocs web és responsabilitat de l'usuari.

IBM pot utilitzar o distribuir la informació que proporcioneu de la manera que consideri oportuna sense que, per això, incorri en cap obligació envers el client.

Els usuaris que hagin rebut llicència per a aquest programa que vulguin obtenir informació per a habilitar (i) l'intercanvi d'informació entre programes creats de manera independent i altres programes (incloent-hi aquest) i (ii) la utilització mútua de la informació que s'ha intercanviat, s'hauran de posar en contacte amb:

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

Aquesta informació pot estar disponible, segons les condicions corresponents, incloent-hi, en alguns casos, el pagament d'una tarifa.

El programa amb llicència que es descriu en aquest document i tot el material amb llicència disponible per a aquest programa els proporciona IBM sota les condicions del Contracte de client d'IBM, el Contracte de llicència de programa internacional d'IBM, o qualsevol altre contracte equivalent.

Les dades de rendiment contingudes aquí s'han obtingut en un entorn controlat. Per tant, els resultats obtinguts en altres sistemes operatius poden variar significativament. És possible que algunes mesures s'hagin realitzat en sistemes a nivell de desenvolupament; no hi ha cap garantia que aquestes mesures siguin iguals en sistemes disponibles per a tothom. A més, és possible que algunes mesures s'hagin calculat mitjançant l'extrapolació. Els resultats reals poden variar. Els usuaris d'aquest document haurien de verificar les dades aplicables al seu entorn específic.

La informació relacionada amb productes que no són d'IBM s'ha obtingut dels proveïdors d'aquests productes, dels seus anuncis publicats o d'altres fonts disponibles públicament. IBM no ha provat aquests productes i no pot confirmar la precisió del rendiment, la compatibilitat ni cap altra reclamació relacionada amb els productes que no són d'IBM. Les preguntes relacionades amb les capacitats dels productes que no són d'IBM s'hauran de dirigir als proveïdors d'aquests productes.

Marques registrades

IBM, el logotip d'IBM i ibm.com són marques comercials o marques registrades d'International Business Machines Corporation als Estats Units o a altres països. Si aquests i altres termes de marques registrades d'IBM estan marcats amb un símbol de marca registrada (® o ™) la primera vegada que apareixen, aquests símbols indiquen marques comercials o marques registrades dels EUA que són propietat d'IBM en el moment de publicar-se aquesta informació. Aquestes marques comercials també poden ser marques comercials o registrades a altres països. Una llista actualitzada de marques registrades d'IBM està disponible a la web a l'apartat "Informació de copyright i marques registrades" de http://www.ibm.com/legal/copytrade.shtml.

Adobe, el logotip d'Adobe logo, PostScript i el logotip de PostScript són marques registrades d'Adobe Systems Incorporated als Estats Units i/o a altres països.

Intel i Itanium són marques registrades d'Intel Corporation o de les seves filials als Estats Units i a altres països.

Linux és una marca registrada de Linus Torvalds als Estats Units i/o a altres països.

Java i totes les marques registrades i logotips basats en Java són marques comercials o marques registrades d'Oracle o els seus afiliats.

Altres noms d'empreses, productes o serveis poden ser marques registrades o marques de serveis d'altres empreses.

Avisos

Aquesta informació s'ha desenvolupat per a productes i serveis oferts als EUA. És possible que IBM no ofereixi els productes, serveis o característiques que es mencionen dins aquest document en altres països. Consulteu el vostre representant local d'IBM si voleu obtenir més informació sobre els productes i els serveis disponibles actualment a la vostra zona. Qualsevol referència a un producte, programa o servei d'IBM no té la intenció de declarar o implicar que només es pot utilitzar aquell producte, programa o servei. En el seu lloc es podria utilitzar qualsevol producte, programa o servei equivalent que no infringeixi cap llei de propietat intel•lectual d'IBM. Tanmateix, és responsabilitat de l'usuari avaluar i verificar el funcionament de qualsevol producte, programa o servei que no sigui d'IBM.

IBM pot tenir patents o sol•licituds pendents de patent que tractin el tema d'aquest document. El fet de disposar d'aquest document no us dóna cap llicència sobre aquestes patents. Podeu enviar per escrit al fabricant les consultes referents a les llicències.

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EUA

Per fer consultes sobre llicències pel que fa a la informació de DBCS (doble byte), poseu-vos en contacte amb el Departament de Propietat Intel•lectual d'IBM del vostre país, o envieu les consultes, per escrit, a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japó

El paràgraf següent no s'aplica al Regne Unit ni a cap altre país on aquestes disposicions entrin en conflicte amb la legislació local.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA AQUESTA PUBLICACIÓ "TAL QUAL" SENSE CAP GARANTIA, NI EXPLÍCITA NI IMPLÍCITA, INCLOENT-HI, ENTRE D'ALTRES, LES GARANTIES RELATIVES AL NO INFRINGIMENT, A LA COMERCIALITZACIÓ O A L'ADEQUACIÓ PER A UNA FINALITAT DETERMINADA. Alguns països no permeten la renúncia de les garanties implícites o explícites en determinades transaccions i, per tant, pot ser que el paràgraf anterior no s'apliqui en el vostre cas.

Pot ser que la publicació inclogui incorreccions tècniques o errors tipogràfics. Es realitzaran modificacions periòdiques pel que fa a la informació de la publicació; aquestes modificacions s'incorporaran a les noves edicions de la informació. IBM pot efectuar millores i/o canvis en els productes i/o programes descrits en aquesta informació en qualsevol moment sense cap avís previ.

Les referències que apareixen en aquesta documentació a llocs Web que no són d'IBM es proporcionen a tall informatiu i de cap manera no es pretén aprovar aquests llocs Web. El material d'aquests llocs web no forma part del material per a aquest producte d'IBM i la utilització d'aquests llocs web és responsabilitat de l'usuari.

IBM pot utilitzar o distribuir la informació que proporcioneu de la manera que consideri oportuna sense que, per això, incorri en cap obligació envers el client.

Els usuaris que hagin rebut llicència per a aquest programa que vulguin obtenir informació per a habilitar (i) l'intercanvi d'informació entre programes creats de manera independent i altres programes (incloent-hi aquest) i (ii) la utilització mútua de la informació que s'ha intercanviat, s'hauran de posar en contacte amb:

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

Aquesta informació pot estar disponible, segons les condicions corresponents, incloent-hi, en alguns casos, el pagament d'una tarifa.

El programa amb llicència que es descriu en aquest document i tot el material amb llicència disponible per a aquest programa els proporciona IBM sota les condicions del Contracte de client d'IBM, el Contracte de llicència de programa internacional d'IBM, o qualsevol altre contracte equivalent.

Les dades de rendiment contingudes aquí s'han obtingut en un entorn controlat. Per tant, els resultats obtinguts en altres sistemes operatius poden variar significativament. És possible que algunes mesures s'hagin realitzat en sistemes a nivell de desenvolupament; no hi ha cap garantia que aquestes mesures siguin iguals en sistemes disponibles per a tothom. A més, és possible que algunes mesures s'hagin calculat mitjançant l'extrapolació. Els resultats reals poden variar. Els usuaris d'aquest document haurien de verificar les dades aplicables al seu entorn específic.

La informació relacionada amb productes que no són d'IBM s'ha obtingut dels proveïdors d'aquests productes, dels seus anuncis publicats o d'altres fonts disponibles públicament. IBM no ha provat aquests productes i no pot confirmar la precisió del rendiment, la compatibilitat ni cap altra reclamació relacionada amb els productes que no són d'IBM. Les preguntes relacionades amb les capacitats dels productes que no són d'IBM s'hauran de dirigir als proveïdors d'aquests productes.

Marques registrades

IBM, el logotip d'IBM i ibm.com són marques comercials o marques registrades d'International Business Machines Corporation als Estats Units o a altres països. Si aquests i altres termes de marques registrades d'IBM estan marcats amb un símbol de marca registrada (® o ™) la primera vegada que apareixen, aquests símbols indiquen marques comercials o marques registrades dels EUA que són propietat d'IBM en el moment de publicar-se aquesta informació. Aquestes marques comercials també poden ser marques comercials o registrades a altres països. Una llista actualitzada de marques registrades d'IBM està disponible a la web a l'apartat "Informació de copyright i marques registrades" de http://www.ibm.com/legal/copytrade.shtml.

Adobe, el logotip d'Adobe logo, PostScript i el logotip de PostScript són marques registrades d'Adobe Systems Incorporated als Estats Units i/o a altres països.

Intel i Itanium són marques registrades d'Intel Corporation o de les seves filials als Estats Units i a altres països.

Linux és una marca registrada de Linus Torvalds als Estats Units i/o a altres països.

Java i totes les marques registrades i logotips basats en Java són marques comercials o marques registrades d'Oracle o els seus afiliats.

Altres noms d'empreses, productes o serveis poden ser marques registrades o marques de serveis d'altres empreses.

Índex

Caràcters Especials

A

B

C

D

E

F

G

T

U

V

Imprès a Espanya