IBM WebSphere Real Time for RT Linux
Version 3

Guide d'utilisation

IBM WebSphere Real Time for RT Linux
Version 3

Guide d'utilisation

Important

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section Chapitre 11, «Remarques», à la page 155.

Première édition (août 2011)

Cette édition du guide d'utilisation s'applique à IBM WebSphere Real Time for RT Linux, Version 3 et à toutes les éditions et modifications suivantes, sauf indication contraire dans les nouvelles éditions.

Table des matières

Figures

Tableaux

Préface

Ce guide d'utilisation fournit des informations générales sur IBM® WebSphere Real Time for RT Linux.

Chapitre 1. Introduction

Cette section fournit des informations sur IBM WebSphere Real Time for RT Linux.

Présentation de WebSphere Real Time for RT Linux

WebSphere Real Time for RT Linux intègre les fonctionnalités temps réel avec la machine virtuelle (JVM) J9 IBM.

WebSphere Real Time for RT Linux est un produit Java Runtime Environment doté d'un kit de développement de logiciels (SDK) qui apporte à IBM SDK for Java les fonctionnalités temps réel. Les applications qui dépendent de temps de réponse précis peuvent tirer avantage des fonctionnalités temps réel fournies avec WebSphere Real Time for RT Linux avec la technologie Java standard.

Fonctions

Les applications temps réel ont besoin d'une exécution cohérente et non pas d'une vitesse absolue.

Lorsque la machine JVM est exécutée en mode temps réel, des zones de mémoire supplémentaires sont disponibles en complément du segment de mémoire de récupération de place. Les programmes peuvent demander ou spécifier n'importe quel nombre de zones de mémoire sectorisée réutilisable et de mémoire pérenne non réutilisable dont la place n'est pas récupérée. Cette fonctionnalité confère à l'application un meilleur contrôle sur l'utilisation de la mémoire. La machine JVM utilise également récupérateur de place Metronome pour exécuter des récupérations basées sur le temps. Lorsque la machine JVM est exécutée dans un mode de traitement standard, vous pouvez utiliser plusieurs récupérateurs de place basés sur le travail pour optimiser le traitement mais la durée sera plus longue que pour le récupérateur de place Metronome.

Les principaux inconvénients du déploiement d'applications temps réel avec des machines JVM classiques sont les suivants :
- Délais imprévisibles (potentiellement longs) du récupérateur de place.
- Retards d'exécution des méthodes lors de la compilation et de la recompilation JIT (Just-In-Time) avec une durée d'exécution variable.
- Planification arbitraire du système d'exploitation.

WebSphere Real Time for RT Linux élimine ces obstacles en fournissant :
- Le récupérateur de place Metronome est une fonction déterministe incrémentielle avec des délais d'interruption très courts.
- Compilation AOT (Ahead-Of-Time)
- Planification FIFO basée sur la priorité

En outre, WebSphere Real Time fournit au programmeur temps réel les fonctions RTSJ. Voir «Support pour RTSJ», à la page 9.

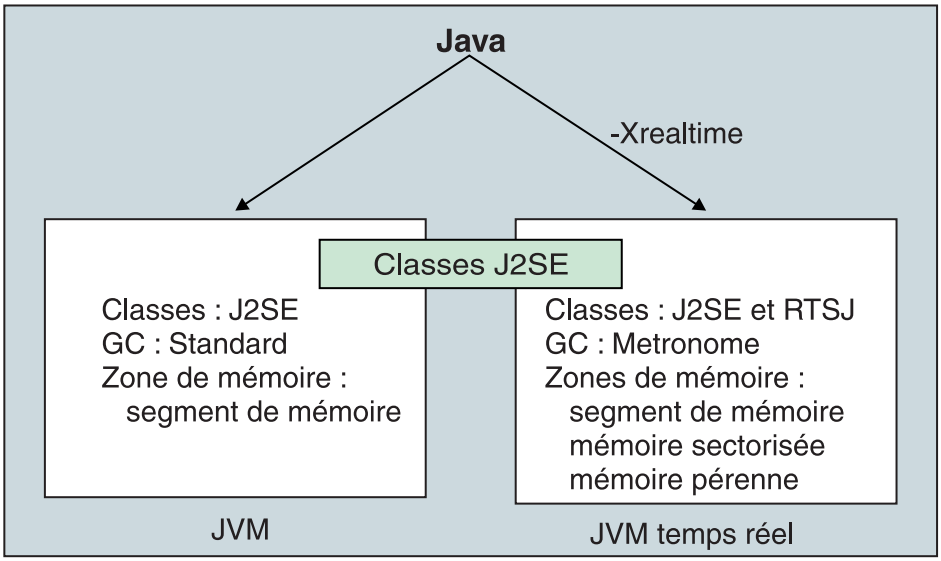

Figure 1. Présentation de WebSphere Real Time for RT Linux

Vous activez les fonctions temps réel en utilisant l'option -Xrealtime lors de l'exécution de la machine JVM ou l'un des outils fournis. Par défaut, la machine JVM et les outils fournis s'exécutent sans les fonctions temps réel activées. La figure 1 montre les relations des deux machines JVM fournies avec WebSphere Real Time for RT Linux.

Les commandes Java suivantes reconnaissent l'option -Xrealtime :

Tableau 1. Commandes Java utilisées en mode temps réel

Commande	Fonction
java	S'exécute en mode standard par défaut, mais également en mode temps réel lorsque l'option -Xrealtime est définie. En mode temps réel, le programmeur accède aux classes depuis le package javax.realtime. Vous pouvez utiliser des fichiers JAR précompilés depuis l'outil et la technologie de récupération de place déterministe Metronome.
javac, javah, javap	S'exécute en mode standard par défaut, mais lorsque l'option -Xrealtime est indiquée, et inclut les classes javax.realtime.* dans le chemin d'accès aux classes.
admincache	Peut être exécuté avec ou sans -Xrealtime, mais le remplissage d'un cache partagé avec l'outil **admincache** n'est possible qu'en mode temps réel. En mode normal, seuls les utilitaires de cache sont disponibles (listAllCaches ou printStats). A l'instar de **jdmpview**, **admincache** doit être exécuté avec -Xrealtime pour accéder aux caches de la machine JVM temps réel et sans -Xrealtime pour accéder aux caches de la machine JVM normale.
jextract	jextract s'exécute en mode standard par défaut, mais doit être exécuté avec l'option -Xrealtime lors du traitement des vidages système générés par la machine JVM en mode temps réel

Nouveautés

Cette rubrique présente les modifications relatives à IBM WebSphere Real Time for RT Linux.

WebSphere Real Time for RT Linux V3

WebSphere Real Time for RT Linux V3 est une extension d'IBM SDK for Java 7, fondée sur les fonctions disponibles dans cette version afin d'inclure les fonctionnalités temps réel. Les versions antérieures de WebSphere Real Time for RT Linux étaient basées sur des versions antérieures d'IBM SDK for Java.

Pour en savoir plus sur les nouveautés, voir : What's new dans le centre de documentation d'IBM SDK for Java 7.

jxeinajar

WebSphere Real Time for RT Linux V3 ne prend plus en charge l'utilisation de jxeinajar. A des fins de référence, les informations relatives à jxeinajar, et en particulier à la migration vers admincache sont fournies dans la documentation WebSphere Real Time for RT Linux.

Avantages

L'environnement a pour avantages de permettre aux applications Java de s'exécuter avec un meilleur niveau de prévisibilité qu'avec la machine JVM standard et de fournir un timing cohérent aux applications Java. Les activités en arrière-plan, telles que la compilation et la récupération de place, sont exécutées à certains moments, ce qui élimine les pics imprévus d'activité en arrière-plan lors de l'exécution de l'application.

Pour tirer parti de ces avantages, étendez la machine JVM avec les fonctions suivantes :

- Technologie de récupération de place temps réel Metronome
- Compilation AOT (Ahead-of-time)
- Support de Spécification en temps réel de Java (RTSJ)

Toutes les applications Java peuvent s'exécuter dans un environnement temps réel sans modification en bénéficiant du récupérateur de place Metronome et de sa récupération de place déterministe régulière. Pour tirer le meilleur parti de WebSphere Real Time for RT Linux, vous pouvez écrire des applications spécifiquement pour l'environnement temps réel en utilisant unités d'exécution temps réel et unités d'exécution en temps réel ne contenant pas de segments. La méthode que vous utilisez dépend de la spécification de timing de l'application.

La plupart des applications Java temps réel peuvent exploiter les courtes pauses de récupérateur de place Metronome et d'AOT pour atteindre leurs objectifs tout en bénéficiant toujours de la portabilité Java. Les applications avec des exigences plus strictes doivent utiliser les fonctions RTSJ de unités d'exécution temps réel et unités d'exécution en temps réel ne contenant pas de segments avec la mémoire sectorisée et pérenne. Cette approche limite l'exécution de l'application à un environnement temps réel et vous perdez les avantages qu'offre la portabilité vers JSE Java. Vous devez également développer un modèle de programmation plus complexe.

Chapitre 2. Description d'IBM WebSphere Real Time for RT Linux

Cette section présente les composants clés d'IBM WebSphere Real Time for RT Linux.

Présentation du récupérateur de place Metronome

Le récupérateur de place Metronome remplace le récupérateur de place standard dans WebSphere Real Time for RT Linux.

La principale différence entre la récupération de place Metronome et la récupération de place standard réside dans le fait que la récupération de place Metronome intervient par petites étapes interruptibles alors que la récupération de place standard arrête l'application lorsqu'elle marque et récupère la place.

Par exemple :

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

L'exemple spécifie que l'application s'exécute à 80 % toutes les 60 ms. Les 20 % du temps restant peuvent être utilisés pour la récupération de place. Le récupérateur de place Metronome garantit des niveaux d'utilisation dès lors qu'il dispose des ressources suffisantes. La récupération de place commence lorsque la quantité d'espace libre dans le segment de mémoire tombe en dessous d'un seuil défini dynamiquement.

Récupération de place et priorités

L'unité d'exécution de récupération de place doit s'exécuter avec une priorité supérieure à la priorité la plus élevée qui génère de la place dans le segment de mémoire. Autrement, elle ne s'exécute pas comme spécifié par l'utilisation configurée. Les unités d'exécution Java standard et unités d'exécution temps réel peuvent générer de la place et, par conséquent, la récupération de place doit s'exécuter avec une priorité supérieure à celle de toutes les unités d'exécution temps réel. Cette hiérarchisation est gérée automatiquement par la machine JVM et la récupération de place s'exécute avec une priorité de 0,5 supérieure à la priorité la plus élevée de toutes les unités d'exécution temps réel. Toutefois, il est important de veiller à ce que les unités d'exécution NHRT (unités d'exécution en temps réel ne contenant pas de segments) ne soient pas affectées par la récupération de place. Exécutez toutes les unités d'exécution NHRT avec une priorité supérieure à la priorité la plus élevée des unités d'exécution unités d'exécution temps réel. Cela implique que les unités d'exécution NHRT s'exécutent avec une priorité supérieure à celle de la récupération de place et qu'elles ne sont pas retardées.

Le tableau 2 montre un exemple type de priorités que vous pouvez définir et les priorités de récupération de place associées qui en découlent.

Pour la comparaison des priorités Java et des priorités SE, voir «Mappage et héritage des priorités», à la page 12.

Tableau 2. Exemple de récupération de place et de priorités

Unités d'exécution	**Priorités (exemples)**
Si l'unité d'exécution temps réel ayant la priorité la plus élevée est :	20 (priorité SE 43)
Le récupérateur de place est :	20.5 (priorité SE 44)
Pour qu'une unité d'exécution NHRT s'exécute indépendamment du récupérateur de place, définissez une priorité supérieure à celle de la récupération de place :	21 (priorité SE 45) ou plus.
L'unité d'exécution d'alarme Metronome est :	Priorité 46 (priorité SE 89)

Remarque : Même avec cette configuration, les unités d'exécution unités d'exécution en temps réel ne contenant pas de segments ne sont pas totalement ignorées par la récupération de place, car l'unité d'exécution d'alarme Metronome s'exécute avec la priorité la plus élevée dans le système pour qu'elle puisse s'activer régulièrement et s'exécuter si la récupération de place doit exécuter une opération. Naturellement, le travail associé est limité et donc pas très important.

Récupération de place Metronome et déchargement de classe

Le récupérateur de place Metronome ne décharge pas les classes dans IBM WebSphere Real Time, car il peut nécessiter une certaine quantité de travail non déterministe générant des dépassements de durée de pause.

Unités d'exécution du récupérateur de place Metronome

Le récupérateur de place Metronome est constitué de deux types d'unités d'exécution : une unité d'exécution d'alarme et un certain nombre d'unités d'exécution de récupération de place. Par défaut, il existe une seul unité d'exécution de récupération de place. Vous pouvez définir le nombre d'unités d'exécution de récupération de place de la machine JVM en utilisant l'option **-Xgcthreads**.

Vous ne pouvez pas changer le nombre d'unités d'exécution d'alarme de la machine JVM.

Le récupérateur de place Metronome vérifie régulièrement la machine JVM pour déterminer si le segment de mémoire contient un espace libre suffisant. Lorsque la quantité d'espace libre tombe en dessous de la limite, le récupérateur de place Metronome déclenche la machine JVM pour lancer la récupération de place.

Unité d'exécution d'alarme

La seule unité d'exécution d'alarme permet d'utiliser une quantité minimale de ressources. Elle «s'active» régulièrement et elle vérifie :

- la quantité d'espace libre dans le segment de mémoire
- si la récupération de place est active.

Si l'espace est insuffisant et que la récupération de place est en cours, l'unité d'exécution d'alarme déclenche les unités d'exécution de

récupération pour lancer la récupération de place. L'unité d'exécution d'alarme ne fait rien jusqu'à la prochaine vérification JVM planifiée.

Unités d'exécution de récupération

Chaque unité de récupération vérifie les unités d'exécution Java et unités d'exécution temps réel pour rechercher les objets du segment de mémoire. Elle vérifie les zones de mémoire dans l'ordre suivant :

1. La mémoire sectorisée pour identifier et marquer les objets actifs dans le segment de mémoire utilisés par les objets de la mémoire sectorisée.
2. La mémoire pérenne pour identifier et marquer les objets actifs dans le segment de mémoire utilisés par les objets de la mémoire pérenne.
3. La mémoire du segment pour identifier et marquer les objets actifs.

Lorsque les objets actifs sont marqués, les objets démarqués sont disponibles pour la récupération.

A la fin du cycle de récupération de place, le récupérateur de place Metronome vérifie la quantité d'espace libre dans le segment de mémoire. Si l'espace est toujours insuffisant, un nouveau cycle de récupération de place démarre en utilisant le même ID de déclencheur. Si l'espace est suffisant, le déclencheur et les unités d'exécution de récupération de place s'arrêtent. L'unité d'exécution d'alarme continue de contrôler l'espace libre dans le segment de mémoire et déclenche un autre cycle de récupération de place lorsque cela est nécessaire.

Pour plus d'informations sur l'utilisation du récupérateur de place Metronome, voir «Utilisation du récupérateur de place Metronome», à la page 66.

Compilateurs

IBM WebSphere Real Time for RT Linuxprend en charge plusieurs modèles de compilation de code qui fournissent plusieurs niveaux de performance du code et de déterminisme.

Les options disponibles pour la compilation du code Java avec IBM WebSphere Real Time for RT Linux sont les suivantes :

Compilation JIT (Just-In-Time) basse priorité

Le modèle de compilation par défaut dans WebSphere Real Time for RT Linux utilise un compilateur JIT (Just-In-Time) pour compiler les méthodes importantes d'une application Java lors de l'exécution de l'application. Dans ce mode, le compilateur JIT fonctionne de la même manière que l'opération du compilateur JIT dans une machine JVM non-temps réel. La différence réside dans le fait que le compilateur WebSphere Real Time for RT Linux JIT s'exécute avec une priorité plus basse que les unités d'exécution temps réel. Priorité plus base signifie que le compilateur JIT utilise les ressources système lorsque l'application n'a pas besoin d'exécuter des tâches temps réel. Par conséquent, le compilateur JIT n'affecte pas de manière significative les performances des tâches temps réel.

Code précompilé AOT (Ahead-Of-Time)

WebSphere Real Time for RT Linux compile les méthodes Java en code natif dans une étape de précompilation avant d'exécuter l'application. L'utilisation du code précompilé AOT fournit le plus haut niveau de déterminisme avec de bonnes performances.

Mode mixte combinant du code précompilé AOT et la compilation JIT basse priorité

Le code compilé AOT et JIT peut être utilisé lorsque l'application s'exécute.

Ce mode de fonctionnement peut fournir un très bon déterminisme avec de bonnes performances et de très bonnes performances pour les méthodes exécutées fréquemment.

Opération interprétée
L'interpréteur exécute une application Java, mais n'utilise pas du tout la compilation de code.

Pour plus d'informations sur l'utilisation du code compilé, voir «Utilisation du code compilé avec WebSphere Real Time for RT Linux», à la page 36.

Comparaison entre les compilations JIT et AOT

La compilation AOT (Ahead-of-Time) permet de compiler les classes et méthodes Java avant d'exécuter le code. La compilation AOT évite l'impact du timing imprévisible du compilateur JIT sur les chemins de performance sensibles. Pour que le code soit compilé avant son exécution et atteindre un niveau optimale de performance déterministe, vous pouvez précompiler le code dans un cache de classes partagées en utilisant le compilateur AOT.

Remarque : Le code compilé AOT ne s'exécute généralement pas aussi rapidement que le code compilé JIT.

Le compilateur JIT (Just-In-Time) s'exécute sous la forme d'une unité d'exécution SCHED_OTHER avec une priorité supérieure à celle des unités d'exécution Java standard, mais inférieure à celle des unités d'exécution RTH (real-time thread). Par conséquent, la compilation JIT (Just-in-time) ne génère pas de retards non déterministes en mode temps réel. Par conséquent, le travail temps réel important est exécuté à temps, car il n'est pas anticipé par le compilateur JIT. Toutefois, le code temps réel peut s'exécuter comme code interprété si le compilateur JIT n'a pas eu le temps de compiler les méthodes actives qui se sont accumulées. Voir la comparaison sur la figure 2, à la page 9.

En règle générale, si l'application a une phase de mise en fonctionnement, il est plus efficace de l'exécuter avec JIT et, si nécessaire, de désactiver JIT lorsque cette phase est terminée. Cette méthode permet au compilateur JIT de générer du code pour l'environnement dans laquelle l'application s'exécute.

Si l'application a une phase de mise en fonctionnement et que vous ne savez pas si les principaux chemins d'exécution sont compilés via une opération d'application standard, la compilation AOT fonctionne correctement dans cet environnement.

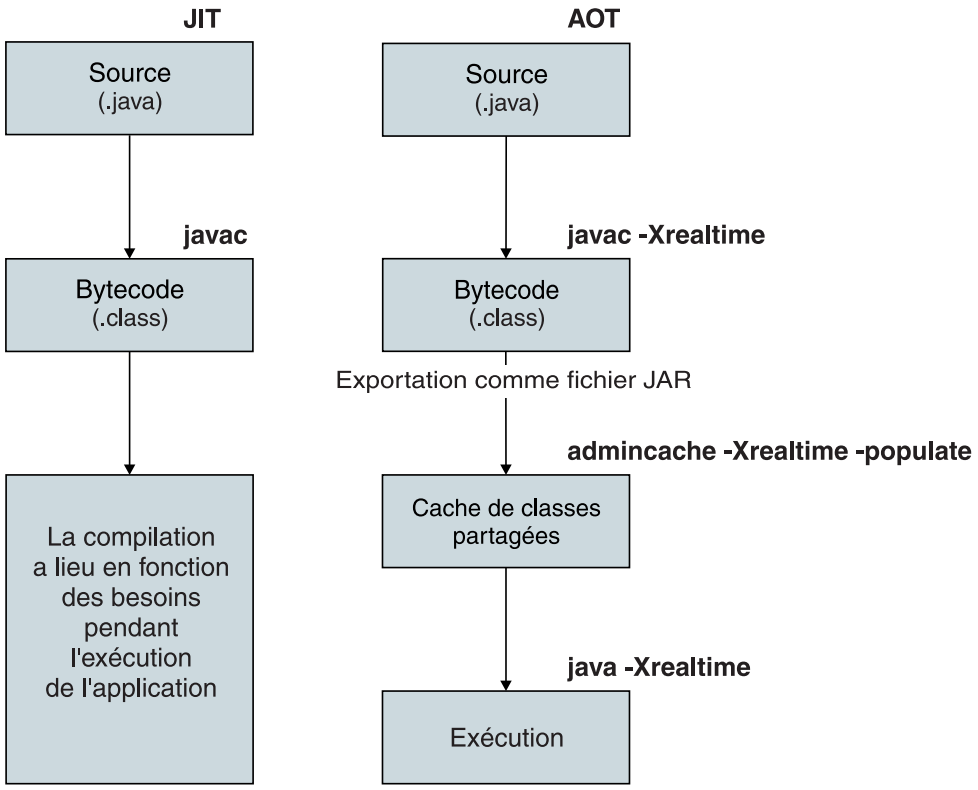

Figure 2. Comparaison du compilateur JIT et du compilateur AOT.

Support pour RTSJ

WebSphere Real Time for RT Linux implémente Spécification en temps réel de Java (RTSJ).

La version 3.0 WebSphere Real Time for RT Linux a été certifiée conforme à RTSJ par rapport à RTSJ Technology Compatibility Kit 1.0.2 version J9 3.1.0 FCS et à Java Compatibility Kit (JCK) version 7.0.

Planification et répartition des unités d'exécution temps réel

La planification et la répartition des unités d'exécution Java temps réel fait partie de la spécification RTS (Real Time Specification) pour Java. La règle de planification SCHED_FIFO permet de définir la priorité des unités d'exécution Java temps réel en utilisant des priorités de système d'exploitation Linux comprises entre 11 et 89.

Les informations sur les règles de planification Linux se trouvent dans «Planification et répartition des unités d'exécution», à la page 35.

Objets planifiables et leurs paramètres

Il existe deux principaux types d'objets planifiables temps réel : unités d'exécution temps réel et gestionnaires d'événements asynchrones.

Ces objets planifiables ont les paramètres suivants :

SchedulingParameters
PriorityParameters planifie les objets planifiables temps réel par priorité.

ReleaseParameters

- **PeriodicParameters** décrit la libération périodique des objets planifiables temps réel. Une unité d'exécution temps réel périodique est une unité d'exécution qui est libérée régulièrement.
- **AperiodicParameters** décrit la libération des objets planifiables temps réel. Les unités d'exécution temps réel apériodiques sont libérées de manière irrégulière.

MemoryParameters
Décrit les contraintes d'allocation de mémoire des objets planifiables temps réel.

ProcessingGroupParameters
Non pris en charge dans WebSphere Real Time for RT Linux.

Planificateur de priorité

Dans WebSphere Real Time for RT Linux, le planificateur est un planificateur de priorités. Comme son nom l'indique, il gère l'exécution des objets planifiables en fonction de leurs priorités actives.

Le planificateur contient la liste des objets planifiables et détermine le moment où chaque objet doit être libéré pour s'exécuter dans l'UC. Le planificateur doit respecter les paramètres associés à chaque objet planifiable. Les méthodes addToFeasibility, isFeasible et removeFromFeasibility sont fournies à cet effet.

Priorités et règles

Les unités d'exécution Java standard, à savoir les unités d'exécution allouées comme objets `java.lang.Thread`, peuvent utiliser des règles de planification SCHED_OTHER, SCHED_RR ou SCHED_FIFO. Les unités d'exécution temps réel, à savoir les unités d'exécution allouées sous la forme `java.lang.RealtimeThread`, et les gestionnaires d'événements asynchrones, utilisent la règle de planification SCHED_FIFO

Les unités d'exécution Java utilisent la règle de planification par défaut SCHED_OTHER, sauf si la JVM est démarrée par une unité d'exécution avec la règle SCHED_RR ou SCHED_FIFO. Les unités d'exécution Java qui utilisent la règle SCHED_OTHER ont la priorité d'unité d'exécution de système d'exploitation affectée de la valeur 0. Les unités d'exécution Java qui utilisent la règle SCHED_RR ou SCHED_FIFO héritent de la priorité de l'unité d'exécution qui démarre la JVM.

Pour unités d'exécution temps réel, la règle SCHED_FIFO n'a pas de tranches de temps et elle prend en charge 99 priorités de 1 (la plus basse) à 99 (la plus élevée). Cette implémentation WebSphere Real Time for RT Linux prend en charge 28 priorités utilisateur dans la plage 11 à 38 inclus. Par conséquent :

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

retourne 11 et

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

retourne 38.

Les priorités SE 81 - 89 sont utilisées par la JVM IBM pour distribuer les unités d'exécution de tâche. Ces unités d'exécution sont toutes destinées à exécuter une petite partie d'un travail avant redevenir inactive. Les unités d'exécution sont les suivantes :

- L'unité d'exécution d'alarme récupérateur de place Metronome s'exécute avec la priorité 89. Cette unité d'exécution s'exécute régulièrement et distribue une unité de travail de récupération de place.
- Deux unités d'exécution de signaux asynchrones qui traitent des signaux asynchrones, une étant une unité d'exécution NHRT (no-heap real-time) ayant la priorité 88 et l'autre la priorité 87.
- Deux unités d'exécution d'horloge, qui répartissent les événements d'horloge, une étant une unité d'exécution NHRT pour les horloges sans segment de mémoire avec la priorité 85 et l'autre ayant la priorité 83.
- Les unités d'exécution de gestionnaire d'événement asynchrone, qui sont réparties pour exécuter des gestionnaires d'événements asynchrones et qui sont affectées de la priorité du gestionnaire lors de l'exécution d'un gestionnaire d'événements asynchrones. Le système démarre avec deux unités d'exécution de gestionnaire temps réel sans segment de mémoire avec la priorité 85 et huit autres avec le priorité 83.
- L'unité d'exécution NHRT Signal asynchrone sans segment avec la priorité 88 gère les demandes de cliché de tas, de cliché de processus et javacore. Elle augmente temporairement sa priorité pour l'amener à 89 lors de la création des fichiers de vidage.

L'unité d'exécution Metronome GC Trace s'exécute avec la priorité SE 12 et l'unité d'exécution JIT Sampler qui échantillonne les méthodes Java pour la compilation, s'exécute avec la priorité SE 13.

L'unité d'exécution Compilation JIT (qui est différente de l'unité d'exécution Echantillonneur JIT) s'exécute avec la règle SCHED_OTHER avec la priorité SE 0.

Les unités d'exécution Compilation JIT et Echantillonneur JIT sont désactivées si **-Xnojit** ou **-Xint** est défini.

récupérateur de place Metronome et la priorité du finaliseur change constamment (avant chaque récupération) pour qu'elle soit supérieure à celle de l'unité d'exécution d'allocation de segment de mémoire ayant la priorité la plus élevée. Vous devez vérifier que les unités d'exécution d'allocation de segment de mémoire ont une priorité inférieure à celle de NoHeapRealtimeThreads.

Une unité d'exécution d'allocation de segment de mémoire est une unité d'exécution utilisateur non-NHRT qui n'est pas inactive ou bloquée dans un moniteur. Une unité d'exécution qui exécute le code natif en dehors de l'interface JNI n'est pas considérée correspondre à une unité d'exécution d'allocation de segment de mémoire. Si une récupération de place est en cours lorsqu'une unité d'exécution d'allocation de segment de mémoire s'active, n'est plus bloquée dans un moniteur ou quitte JNI, elle est forcée d'attendre la fin de la récupération pour pouvoir continuer.

La priorité SE 81 est réservée aux unités d'exécution JVM internes qui allouent le segment de mémoire. Si une unité d'exécution JVM interne a la priorité SE 81, le récupérateur de place s'exécute avec la priorité SE 82. Lorsque seules les unités d'exécution utilisateur d'allocation de segment de mémoire ne sont pas des unités d'exécution unités d'exécution temps réel, la priorité de récupération de place s'exécute avec la priorité 11. Autrement, la récupération de place s'exécute avec une

priorité qui est une priorité SE supérieure à la celle de l'unité d'exécution utilisateur d'allocation de segment de mémoire avec la priorité la plus haute.

La priorité de récupération de place est ajustée avant un cycle de récupération.

Mappage et héritage des priorités

Chaque priorité Java est mappée à une priorité de base de système d'exploitation et chaque priorité de système d'exploitation est associée à une règle de planification. Les règles de planification de système d'exploitation WebSphere Real Time for RT Linux Linux sont SCHED_OTHER, SCHED_RR et SCHED_FIFO.

Les unités d'exécution Java temps réel utilisent la règle SCHED_FIFO, alors que les unités d'exécution Java standard utilisent la règle de l'unité d'exécution qui démarre la machine JVM. La règle de planification par défaut des unités d'exécution Java est SCHED_OTHER, mais vous pouvez utiliser un utilitaire, tel que **chrt**, pour définir les règles SCHED_RR ou SCHED_FIFO. Pour plus d'informations sur les priorités et les règles des unités d'exécution, voir «Planification et répartition des unités d'exécution», à la page 35.

Le tableau suivant indique l'association des priorités Java aux priorités de système d'exploitation natives. Certaines priorités Java sont réservées à la machine JVM et des priorités natives n'ayant pas de priorités Java correspondantes sont utilisées également par la machine JVM.

Remarque :

- Les priorités 1-10 sont utilisées par les unités d'exécution Java standard.
  - Pour la règle SCHED_OTHER, les priorités Java 1-10 sont mappées à la priorité de système d'exploitation 0.
  - Pour les règles SCHED_FIFO et SCHED_RR, les priorités Java 1-10 héritent de la priorité de l'unité d'exécution qui démarre la machine JVM.
- Les priorités 11 et suivantes sont utilisées par unités d'exécution temps réel et unités d'exécution en temps réel ne contenant pas de segments
- Un objet planifiable s'exécute toujours avec sa propriété active. La priorité active est initialement la priorité de base de l'objet planifiable, mais la priorité active peut être augmentée temporairement par l'héritage de priorité. La priorité de base d'un objet planifiable peut être changée en cours d'exécution.

Priorités de base utilisateur :

```
Priorités Java 1-10 : SCHED_OTHER, Priorité SE 0

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

```
Priorité Java 30 : SCHED_FIFO, priorité SE 63
Priorité Java 31 : SCHED_FIFO, priorité SE 65
Priorité Java 32 : SCHED_FIFO, priorité SE 67
Priorité Java 33 : SCHED_FIFO, priorité SE 69
Priorité Java 34 : SCHED_FIFO, priorité SE 71
Priorité Java 35 : SCHED_FIFO, priorité SE 73
Priorité Java 36 : SCHED_FIFO, priorité SE 75
Priorité Java 37 : SCHED_FIFO, priorité SE 77
Priorité Java 38 : SCHED_FIFO, priorité SE 79
```

Priorités de base interne :

```
Priorité Java 39 : SCHED_FIFO, priorité SE 81
Priorité Java interne 40 : SCHED_FIFO, priorité SE 83
Priorité Java interne 41 : SCHED_FIFO, priorité SE 84
Priorité Java interne 42 : SCHED_FIFO, priorité SE 85
Priorité Java interne 43 : SCHED_FIFO, priorité SE 86
Priorité Java interne 44 : SCHED_FIFO, priorité SE 87
Priorité Java interne 45 : SCHED_FIFO, priorité SE 88
Priorités SE 11, 12, 13
Valeurs paires de priorités SE 26, 28, 30, ..., 82
Priorité SE 89
```

Voir aussi la section "Synchronisation" dans http://www.rtsj.org/specjavadoc/book_index.html.

Héritage des priorités :

La priorité active d'une unité d'exécution peut être augmentée temporairement du fait qu'elle détient un verrou nécessaire par une unité d'exécution ayant une priorité plus élevée. Ces verrous peuvent être des verrous JVM internes ou des moniteurs utilisateurs associés à des méthodes synchronisées ou des blocs synchronisés. La priorité d'une unité d'exécution Java standard, par conséquent, peut avoir temporairement une priorité temps réel jusqu'à ce que l'unité d'exécution libère le verrou.

L'héritage de priorité a pour conséquence, entre autres, de remplacer temporairement la règle d'une unité d'exécution SCHED_OTHER par SCHED_FIFO.

Pour plus d'informations sur les priorités de base et actives, voir la section "Synchronisation" dans la spécification RTSJ.

Gestion de la mémoire

Les segments de mémoire de récupération de place ont toujours été considérés comme un obstacle à la programmation temps réel du fait du comportement imprévisible introduit par la récupération de place. Le récupérateur de place Metronome dans IBM WebSphere Real Time for RT Linux peut fournir des performances de récupération de place hautement déterministes. Simultanément, Spécification en temps réel de Java (RTSJ) fournit des extensions au modèle de mémoire pour les objets en dehors du segment de mémoire ayant fait l'objet d'une récupération de place pour que le programmeur Java puisse gérer explicitement les objets à durée de vie courte et les objets à durée de vie longue.

Zones de mémoire

RTSJ introduit le concept de zone de mémoire qui peut être utilisée pour l'allocation des objets. Certaines zones de mémoire existent en dehors du segment de mémoire et placent des restrictions sur les opérations que le système et le récupérateur de place peuvent exécuter avec ces objets. Par exemple, les objets

dans certaines zones de mémoire ne font jamais l'objet d'une récupération de place, mais le récupérateur de place peut analyser ces zones de mémoire pour rechercher des références à des objets dans le segment de mémoire pour maintenir son intégrité.

La gestion de mémoire a trois types de base :

- Le segment de mémoire est le segment Java traditionnel, mais il est géré par récupérateur de place Metronome.
- La mémoire sectorisée doit être demandée spécifiquement par les applications et elle peut être utilisée uniquement par unités d'exécution temps réel, y compris les unités d'exécution temps NHRT (no-heap real-time threads) et les gestionnaires d'événements asynchrones sans segment de mémoire.
- La mémoire pérenne est une zone de mémoire qui contient des objets qui peuvent être référencés par un objet planifiable, notamment unités d'exécution en temps réel ne contenant pas de segments et les gestionnaires d'événements asynchrones sans segments de mémoire. Elle est utilisée par le chargement de classe et l'initialisation statique, même si l'application ne l'utilise pas.

La mémoire pérenne ou sectorisée peut être configurée pour utiliser la mémoire physique qui est constituée de régions de mémoire ayant des caractéristiques spécifiques, telles qu'un accès substantiellement plus rapide. En règle générale, la mémoire physique n'est pas souvent utilisée et elle est peu susceptible d'affecter l'utilisateur de la JVM standard.

Segment de mémoire

La taille maximale est contrôlée par **-Xmx**, mais veillez à *ne pas* définir la taille de segment de mémoire initiale (**-Xms**) ou à lui affecter la taille maximale **-Xmx**, car en mode temps réel, le segment de mémoire ne passe jamais de la taille de segment de mémoire initiale à la taille de segment de mémoire maximale. Lorsque vous atteignez la taille de segment de mémoire maximale sans espace libre, une erreur OutOf MemoryError se produit. En règle général, la machine JVM temps réel consomme plus de mémoire du segment de mémoire que la machine JVM traditionnelle, car la récupération déterministe nécessite d'organiser les objets différemment, ce qui ce matérialise par une plus grande fragmentation du segment de mémoire. En outre, les tableaux sont divisés en fragments, chaque fragment ayant un en-tête. Elle dépend du rapport entre les grands et les petits objets et du niveau d'utilisation des tableaux, mais elle est susceptible de trouver une application nécessitant 20 % d'espace de segment de mémoire de plus.

récupérateur de place Metronome est similaire au récupérateur «le plus simultané» qui existe dans la JVM standard dans la mesure où il collecte la place lorsque l'application s'exécute. Dans un scénario parfait, le cycle de récupération se termine avant que l'application manque de mémoire, mais certaines applications avec des fréquences d'allocation très élevées peuvent allouer de la mémoire plus rapidement que récupérateur de place Metronome ne peut la récupérer. Divers contrôles détaillés affectent la fréquence de récupération, mais il existe un contrôle qui force Metronome à revenir à la récupération de place STW (stop-the-world) avant d'émettre une erreur OutOf MemoryError. Le paramètre d'exécution est **-Xgc:synchronousGCOnOOM** et l'équivalent est **-Xgc:nosynchronousGCOnOOM**. Le paramètre par défaut est **-Xgc:synchronousGCOnOOM**.

Mémoire sectorisée

RTSJ introduit le concept de mémoire sectorisée. Cette mémoire peut être utilisée par les objets ayant une durée de vie bien définie. Il est possible d'entrer dans un

secteur de manière explicite ou un secteur peut être associé à un objet planifiable (une unité d'exécution temps réel ou un gestionnaire d'événements asynchrone) qui entre effectivement dans le secteur avant d'exécuter la méthode run() de l'objet. Chaque secteur dispose d'un compteur de références et lorsqu'il atteint zéro, les objets qui résident dans le secteur peuvent être fermés (finalisés) et la mémoire associée au secteur est libérée. La réutilisation du secteur est bloquée jusqu'à la fin de la finalisation.

La mémoire sectorisée peut être divisée en deux types : VTMemory and LTMemory. Ces types de mémoires sectorisées varient en fonction du délai nécessaire pour allouer des objets depuis cette zone. LTMemory garantit une allocation temporelle linéaire lorsque la consommation de mémoire de la zone de mémoire est inférieure à la taille initiale de la zone de mémoire. VTMemory n'offre pas cette garantie.

Les secteurs peuvent être imbriqués. Lors de l'entrée dans un secteur, toutes les allocations suivantes sont issues de la mémoire associée au nouveau secteur. Lors de le secteur imbriqué est terminé, le secteur antérieur est restauré et les allocations suivantes sont de nouveau issues de ce secteur.

Compte tenu de la durée de vie des objets sectorisés, il est nécessaire de limiter les références aux objets sectorisés à l'aide d'un groupe restreint de règles d'affectation. Une référence à un objet sectorisé ne peut pas être affectée à une variable d'un secteur externe ou à une zone d'un objet dans le segment de mémoire ou la zone pérenne. Une référence à un objet sectorisé peut être affectée uniquement dans le même secteur ou dans un secteur interne. La machine virtuelle détecte les tentatives d'affectation incorrectes et génère une exception IllegalAssignmentError dans ce cas. La souplesse de choix de types de mémoires sectorisées permet à l'application d'utiliser une zone de mémoire ayant des caractéristiques adaptées à une région définie syntaxiquement du code.

La taille de la zone doit être définie lors de la construction de la zone et le paramètre de ligne de commande **-Xgc:scopedMemoryMaximumSize** contrôle la valeur maximale. La valeur par défaut est 8 Mo et convient dans la plupart des cas.

Mémoire pérenne

La mémoire pérenne est une ressource de mémoire partagée entre tous les objets planifiables et les unités d'exécution dans une application. Les objets alloués dans la mémoire pérenne sont toujours disponibles pour les unités d'exécution sans segment de mémoire et les gestionnaires d'événements asynchrones et ils ne sont pas soumis aux retards provoqués par la récupération de place. Les objets sont libérés par le système lorsque le programme prend fin.

La taille est contrôlée par **-Xgc:immortalMemorySize**. Par exemple, **-Xgc:immortalMemorySize=20m** définit 20 Mo. La valeur par défaut est 16 Mo, ce qui suffit généralement si le chargement de classes n'est intensif. La plupart des exceptions OutOfMemoryError sont provoquées par le chargement de classes.

Estimation des besoins en mémoire

Comment obtenir les informations nécessaires pour allouer une quantité de mémoire suffisante

Une approche raisonnable consiste à identifier la mémoire nécessaire pour stocker les objets prévus avec une certaine marge de sécurité. L'analyse de l'application permet d'identifier le nombre et la nature des objets nécessaires, bien que la taille

réelle nécessaire à un objet peut varier d'un système à l'autre. L'utilisation de la classe SizeEstimator tient compte de la taille réelle de l'objet pour fournir plus d'informations portables.

La classe SizeEstimator

La classe SizeEstimator fournit des conseils sur la quantité de mémoire nécessaire pour stocker un objet. L'estimation est une indication de la quantité de mémoire minimale à allouer pour l'objet lui-même et ne tient pas compte des besoins en mémoire des autres ressources qui pourraient être nécessaires à l'objet, par exemple, lors de sa construction.

Pour plus d'informations sur cette classe, voir http://www.rtsj.org/specjavadoc/book_index.html

Utilisation de la mémoire

Comparaison des unités d'exécution Java, unités d'exécution temps réel et unités d'exécution en temps réel ne contenant pas de segments.

RTSJ (Spécification en temps réel de Java) ajoute deux classes aux unités d'exécution temps réel : la classe RealtimeThread et la classe NoHeapRealtimeThread class.

- Les unités d'exécution temps réel et unités d'exécution en temps réel ne contenant pas de segments sont des objets planifiables. Comme les objets planifiables, elles ont les paramètres suivants : release, scheduling, memory et processing group.
- Les unités d'exécution RTT (Real-time thread) peuvent accéder aux objets dans la mémoire de segment et dans la mémoire sectorisée et la mémoire pérenne.
- Les unités d'exécution NHRT (No-heap real-time thread) accèdent uniquement aux zones de mémoire sectorisée et pérenne.
- Les unités NHRT doivent avoir une priorité plus élevée que les autres unités d'exécution RTT. Si leur priorité est inférieure à celle des unités d'exécution RTT, elles ne s'exécutent plus sans être interrompues par le récupérateur de place.

Remarque : Une unité d'exécution NRHT ayant une priorité supérieure à celle des unités RTT n'est pas interrompue par la récupération de place.

Tableau 3. Accès à la mémoire par les unités d'exécution RTT et NRHT

Unités d'exécution	Mémoire pérenne	Mémoire sectorisée	Mémoire de segment
Unités d'exécution normales	✓	✗	✓
Unités d'exécution RTT	✓	✓	✓
unités d'exécution en temps réel ne contenant pas de segments	✓	✓	✗

Types de zones de mémoire

Mémoire pérenne
La mémoire pérenne n'est pas soumise à la récupération de place. Lorsque de l'espace est alloué dans la mémoire pérenne, il ne peut pas être récupéré jusqu'a ce que l'application quitte.

- De par sa nature vous pouvez chercher à récupérer de la mémoire dans la mémoire pérenne. Une possibilité consiste à créer un pool d'objets réutilisables. Vous pouvez aussi utiliser de la mémoire sectorisée.
- Les objets dans la mémoire pérenne ne peuvent pas faire référence aux éléments dans la mémoire sectorisée. Si une zone dans un objet dans la mémoire pérenne est affectée à un objet de la mémoire sectorisée, une exception IllegalAssignmentError est générée.

Mémoire sectorisée
La mémoire sectorisée peut être utilisée comme zone de mémoire initiale d'un objet planifiable ou un objet peut y entrer. Lorsqu'elle n'est plus référencée, tous les objets sont supprimés de la zone. Les objets planifiables exécutés dans une zone de mémoire sectorisée exécutent toutes leurs allocations d'objet depuis cette zone. Lorsqu'une zone de mémoire sectorisée n'est pas utilisée, les objets qui s'y trouvent sont finalisés et la mémoire est récupérée pour réutiliser le secteur. Lorsque la zone de mémoire sectorisée n'est plus disponible pour les objets planifiables, la mémoire est récupérée pour d'autres utilisations.

La zone de mémoire décrite par une instance ScopedMemory n'existe pas dans le segment de mémoire Java et elle n'est pas soumise à la récupération de place. Vous pouvez utiliser en toute sécurité un objet ScopedMemory comme zone de mémoire initiale associée à une unité d'exécution NoHeapRealtimeThread ou d'entrer dans la zone de mémoire en utilisant la méthode ScopedMemory.enter dans une unité d'exécution NoHeapRealtimeThread.

Mémoire physique
Utilisez la mémoire physique lorsque les caractéristiques de la mémoire sont importantes, par exemple non paginable ou non volatile.

LTMemory (Linear time allocation scheme)
LTMemory garantit une allocation temporelle linéaire lorsque la consommation de mémoire de la région de mémoire est inférieure à la taille initiale de la zone de mémoire. Le délai d'exécution d'allocation peut varier lorsque la consommation de mémoire est comprise entre la taille initiale et la taille maximale de la région. En outre, le système sous-jacent n'est pas tenu de garantir que de la mémoire entre la taille initiale et la taille maximale est toujours disponible.

VTMemory (Variable time allocation scheme)
VTMemory est similaire à LTMemory, sauf que le délai d'exécution d'une allocation depuis la zone **`VTMemory`** doit s'effectuer dans un temps linéaire.

Segment de mémoire
Les objets dans le segment de mémoire font référence aux éléments qui se trouvent dans la mémoire sectorisée. Si une zone dans un objet dans la mémoire du segment est affectée à un objet de la mémoire sectorisée, une exception IllegalAssignmentError est générée.

Synchronisation et partage des ressources

Dans un système temps réel, lorsqu'au moins trois unités d'exécution sont exécutées avec des priorités différentes et se synchronisent les unes par rapport aux autres, une condition appelée inversion de priorité peut se produire occasionnellement dans laquelle une unité d'exécution avec une priorité plus élevée est empêchée de s'exécuter par une unité d'exécution avec une priorité inférieure pendant une longue période. WebSphere Real Time for RT Linux utilise un schéma appelé héritage de priorité pour éviter cette condition.

Lorsqu'une tâche avec une priorité plus élevée est empêchée de s'exécuter par une tâche avec une priorité inférieure, la priorité de cette dernière est augmentée temporairement pour correspondre à la priorité la plus haute jusqu'à ce que la tâche avec la priorité la plus élevée ne soit plus bloquée.

Paramètres périodiques et apériodiques

Les unités d'exécution temps réel disposent d'un certain nombre de paramètres d'édition qui déterminent la fréquence d'édition d'un objet planifiable. Les paramètres périodiques et apériodiques sont des exemples de paramètres d'édition.

Paramètres périodiques

Cette classe est destinée aux objets planifiables édités régulièrement.

AbsoluteTime
Exprimé en millisecondes et nanosecondes.

RelativeTime
Durée en millisecondes ou nanosecondes d'un événement. Par exemple, vous pouvez mesurer la durée absolue lorsqu'un événement démarre et se termine. Ensuite, vous calculez le temps relatif comme différence entre deux mesures.

Paramètres apériodiques

Cette classe est utilisée par les objets planifiables édités régulièrement. Comme un second événement apériodique peut se produire avant la fin du premier, vous pouvez définir la longueur de file d'attente des demandes en attente.

Gestion des événements asynchrones

Les gestionnaires d'événements asynchrones réagissent à des événements qui se produisent en dehors d'une unité d'exécution, par exemple, depuis une interface d'une application. Dans les systèmes temps réel, ces événements doivent répondre dans les délais que vous définissez pour l'application.

Des événements asynchrones peuvent être associés à des interruptions système et des signaux POSIX et les événements asynchrones peuvent être liés à un minuteur.

Comme unités d'exécution temps réel, gestionnaires d'événements asynchrones disposent d'un certain nombre de paramètres. Pour la liste de ces paramètres, voir «Objets planifiables et leurs paramètres», à la page 9.

Gestionnaires des signaux

POSIXSignalHandler prend en charge les signaux SIGQUIT, SIGTERM et SIGABRT. Le comportement par défaut de SIGQUIT génère un vidage Java. La génération du

javadump n'affecte pas le fonctionnement d'un programme actif, mis à part le temps UC et la lecture et l'écriture des fichiers. La génération d'une javadump interrompt le programme tant que le vidage n'est pas terminé. Les performances de l'application ne sont pas prévisibles pendant la génération des javadumps.

Pour supprimer la génération de clichés de mémoire système et javadump lors d'un échec, utilisez **-Xdump:none**.

Pour supprimer uniquement la génération de clichés système et javadump sur un signal SIGQUIT, définissez **-Xdump:java:none -Xdump:java:events=gpf+abort**.

Les signaux suivants peuvent être associés à des gestionnaires d'événements asynchrones (AEH) par le mécanisme POSIXSignalHandler (descriptions des signaux définis dans `/usr/include/bits/signum.h`):

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

Aucun autre signal n'est pris en charge. Tous les signaux précédemment répertoriés ci-dessus sont des signaux asynchrones et il est impossible de prendre en charge l'association à des signaux synchrones (tels que SIGILL et SIGSEGV), car ils indiquent un échec de l'application ou du code JVM et pas un événement généré en externe.

Remarque : Par défaut, SIGQUIT amène l'application Java à générer des clichés (javadump, par exemple) lorsqu'il est reçu par la machine JVM. Bien qu'il soit envoyé également au gestionnaire AEH associé, cette distribution peut créer la confusion ou un comportement indésirable et vous pouvez le désactiver en utilisant l'option **-Xdump:none:events=user** sur la ligne de commande Java.

Documentation requise

WebSphere Real Time for RT Linux implémente Spécification en temps réel de Java (RTSJ).

La version WebSphere Real Time for RT Linux 2.0 a été certifiée comme étant conforme à RTSJ 1.0.2 par rapport à RTSJ Technology Compatibility Kit version 3.0.13 FCS et avec Java Compatibility Kit (JCK) pour la version 6.0.

Fonctions prises en charge

Les fonctions suivantes sont prises en charge :

- Application d'une fréquence d'allocation de segment de mémoire pour limiter la fréquence à laquelle un objet planifiable crée des objets dans le segment de mémoire.

Fonctions non prises en charge

Les fonctions suivantes ne sont pas prises en charge :

- Protocole PCEP (Priority Ceiling Emulation Protocol). Par exemple, il ne permet pas d'utiliser PriorityCeilingEmulation comme règle de contrôle de surveillance.
- Support d'accès automatique, sauf lorsque nécessaire pour la conformité à la spécification.
- Aucun autre planificateur que le planificateur de priorités de base n'est accessible aux applications.

- Application de coût.

Documentation nécessaire pour Spécification en temps réel de Java

La section *Documentation nécessaire* de Spécification en temps réel de Java (RTSJ) figure dans cette section. Les différences avec l'implémentation standard de RTSJ sont indiquées.

1. **L'algorithme de test de faisabilité est le paramètre par défaut.**

   «Si cet algorithme n'est pas l'algorithme par défaut, documentez l'algorithme de test de faisabilité.»

2. **Seul le planificateur de priorités de base est accessible aux applications.**

   «Si d'autres planificateurs de priorités sont accessibles aux applications, documentez le comportement du planificateur et son interaction avec chacun des autres planificateurs, comme indiqué dans le chapitre Planification. Documentez également la liste des classes qui constituent les objets planifiables du planificateur, à moins que la liste soit identique à celle des objets planifiables pour le planificateur de base.»

3. **Un objet planifiable anticipé par un objet planifiable à haute priorité est placé à l'avant de la file d'attente de sa priorité.**

   «Un objet planifiable anticipé par un objet planifiable à haute priorité est placé dans la file d'attente de sa priorité en cours dans une position déterminée par l'implémentation. Si l'objet planifiable anticipé n'est pas placé à l'avant de la file d'attente appropriée, l'implémentation doit documenter l'algorithme utilisé pour la position. La position à l'avant de la file d'attente peut nécessiter une version suivante de cette spécification.»

4. **L'application de coût n'est pas prise en charge.**

   «Si l'implémentation prend en charge l'application de coût, l'implémentation doit documenter la granularité à laquelle la consommation UC actuelle est mise à jour.»

5. **Le mappage séquentiel simple est pris en charge.**

   «Le mappage de mémoire implémenté par un filtre de type de mémoire physique doit être documenté s'il ne s'agit pas d'un mappage séquentiel simple d'octets contigus.»

6. **Il n'existe pas de classes secondaires pour le récupérateur de place Metronome fourni avec WebSphere Real Time for RT Linux.**

   «L'implémentation doit documenter complètement le comportement des classes secondaires GarbageCollector.»

7. **Aucune classe secondaire MonitorControl n'est fournie avec WebSphere Real Time for RT Linux.**

   «Une implémentation qui fournit des classes secondaires MonitorControl non détaillées dans cette spécification doit documenter leurs effets, notamment par rapport au contrôle d'inversion de priorité et aux planificateurs éventuels qui n'ont pas pris en charge la nouvelle règle.»

8. **Un objet planifiable détenant un contrôleur nécessaire à un objet planifiable à haute priorité voit sa priorité remplacée par la priorité la plus haute jusqu'à ce qu'il libère le contrôleur. Si à ce stade, l'objet planifiable ne doit plus être exécutable (à savoir qu'une tâche plus prioritaire doit être exécutée), il est placé à l'arrière de la file d'attente pour sa priorité (non augmentée) d'origine lors de l'exécution dans des noyaux antérieurs à SUSE Linux Enterprise Real Time 10 SP2 Update Kernel version 2.6.22.19-0.16 et**

Red Hat Enterprise Linux 5.1 MRG 2.6.24.7-73 Errata 1. Les noyaux à ces niveaux ou des niveaux supérieurs placent l'objet planifiable à l'avant de la file d'attente.

«Si, lors de la perte de la priorité "augmentée" du fait d'un algorithme d'élimination d'inversion de priorité, l'objet planifiable n'est pas placé à l'avant de sa nouvelle file d'attente, l'implémentation doit documenter le comportement de mise en file d'attente.»

9. **Le planificateur de base est le seul planificateur fourni avec WebSphere Real Time for RT Linux.**

   «Pour les planificateurs disponibles autres que le planificateur de base, une implémentation doit documenter comment la sémantique de synchronisation diffère des règles définies pour l'instance PriorityInheritance par défaut. Elle doit fournir la documentation du comportement du nouveau planificateur avec un héritage de priorité (et le protocole d'émulation de plafond de priorité, si pris en charge) équivalent à la sémantique du planificateur de priorités de base du chapitre Synchronisation.»

10. **La durée la plus longue entre le déclenchement d'un événement et la planification d'un gestionnaire d'événements lié associé est de 40 µs en moyenne et ne dépasse pas 100 µs s'il n'existe pas d'objets planifiables concurrents ou d'activité système de priorité égale ou supérieure et que la récupération de place n'interfère pas. Si l'objet planifiable qui gère la méthode de déclenchement, l'objet AsyncEvent ou le gestionnaire fait référence au segment de mémoire, l'impact potentiel de la récupération de place est celui documenté dans (A). Cela suppose que le code est interprété et qu'un seul gestionnaire (lié) est configuré dans l'événement.**

    «Le temps de réponse le plus long entre le déclenchement d'un événement AsyncEvent à cause d'une liaison et la libération d'un gestionnaire AsyncEventHandler associé (en supposant qu'aucun autre objet planifiable avec une plus haute priorité n'est exécutable) doit être documenté pour une architecture de référence.»

11. **La durée la plus longue entre le déclenchement d'un événement AsynchronouslyInterruptedException sur une unité d'exécution activée par ATC et la première émission de l'exception est de 35 µs en moyenne et ne dépasse pas 160µs s'il n'existe pas d'objets planifiables concurrents ou d'activité système de priorité égale ou supérieure et que la récupération de place n'interfère pas. Activée par ATC dans ce cas, signifie que l'unité d'exécution s'exécute dans une méthode activée par AI dans une région non différée par ATC et que ces conditions restent vraies jusqu'à l'émission de l'exception. L'impact potentiel de la récupération de place est celui documenté dans (A). Si l'unité d'exécution cible est en code natif, le délai est potentiellement non lié. Cela suppose que le code est interprété.**

    «Le délai entre le déclenchement d'une exception AsynchronouslyInterruptedException sur une unité d'exécution activée par ATC et la première émission de l'exception (en supposant qu'aucun autre objet planifiable avec une plus haute priorité n'est exécutable) doit être documenté pour une architecture de référence.»

12. **Non applicable. Voir la réponse 4.**

    «Si l'application de coût est prise en charge et que l'implémentation affecte le coût d'exécution des finaliseurs des objets en mémoire sectorisée à un objet exécutable autre que celui ayant ramené le nombre de références du secteur à 0 en quittant le secteur, les règles d'affectation du coût doivent être documentées.»

13. **L'implémentation standard RealtimeSecurity n'est pas modifiée.**

«Si l'implémentation de RealtimeSecurity est plus restrictive que l'implémentation nécessaire ou qu'elle dispose d'options de configuration d'exécution, ces fonctions doivent être documentées.»

14. **Les finaliseurs des objets dans une mémoire sectorisée sont exécutés par la dernière unité d'exécution à référencer le secteur, à savoir qu'ils sont exécutés lorsque l'unité d'exécution ramène le nombre de référence 1 à 0. Le coût associé à l'exécution des finaliseurs sont affectés à l'unité d'exécution.**

    «Une implémentation peut exécuter des finaliseurs pour des objets en mémoire sectorisée avant de retourner dans le secteur et la fin d'un appel à getReferenceCount() pour le secteur. Toutefois, elle doit documenter le moment où elle exécute les finaliseurs.»

15. **La résolution n'est pas définissable.**

    «Pour chaque horloge prise en charge, la documentation doit indiquer si la résolution est définissable et si elle l'est, elle doit indiquer les valeurs prises en charge.»

16. **Il n'existe pas d'autres horloges que l'horloge temps réel fournie avec WebSphere Real Time for RT Linux.**

    «Si une implémentation contient des horloges autres que l'horloge temps réel nécessaire, leur documentation doit indiquer leurs contextes d'utilisation.»

Remarque :

A L'architecture de référence pour les tests sera un cache LS20, 4 voies de 2 GHz avec 1 Mo 4 Go de mémoire.

B La récupération de place peut retarder une unité d'exécution associée au segment de mémoire. Le récupérateur peut fonctionner dans l'un des deux modes de base qui régissent le comportement lorsque le segment de mémoire est épuisé. Si le récupérateur de place est configuré pour générer une erreur OutOfMemoryError immédiatement dans ce cas, le délai le plus long de récupération de place est généralement inférieur à 1 ms. Actuellement, dans certains cas, le délai est plus long, par exemple, s'il existe un grand nombre d'unités d'exécution avec des piles profondément imbriquées ou de nombreuses grandes portées. Si le récupérateur est configuré pour exécuter une récupération de place avant d'émettre une erreur OutOfMemoryError, le délai de récupération potentiel est associé au nombre d'objets dynamiques dans le segment de mémoire et aux nombres d'objets dans les autres zones de mémoire. Dans ce cas, le délai est considéré délié, car il peut correspondre à un grand nombre de secondes pour les tailles de segment standard.

Chapitre 3. Planification

Lisez cette section avant d'installer WebSphere Real Time for RT Linux.

- «Migration»
-
- «Matériel et logiciel prérequis»
- «Considérations», à la page 24

Migration

WebSphere Real Time for RT Linux s'exécute dans un environnement Linux modifié pour les applications temps réel. Vous pouvez utiliser des applications standard Java dans un environnement temps réel. Vous pouvez également modifier les applications pour exploiter les nouvelles fonctions WebSphere Real Time.

Migration de système

Suivez les instructions fournies par le support Linux.

Matériel et logiciel prérequis

Utilisez cette liste pour vérifier le matériel, le système d'exploitation et l'environnement Java pris en charge pour WebSphere Real Time for RT Linux.

le matériel

Les configurations matérielles certifiées WebSphere Real Time for RT Linux sont des variantes multiprocesseurs des systèmes suivants :

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

Pour rester certifiés pour WebSphere Real Time for RT Linux, les systèmes IBM avec l'hyperthreading ne doit pas avoir cette fonction activée.

En outre, WebSphere Real Time for RT Linux est pris en charge sur le matériel qui exécute un système d'exploitation pris en charge et ayant les caractéristiques suivantes :

- Mémoire physique de 512 Mo minimum
- Au minimum, processeur Intel Pentium 4, AMD Opteron ou Intel Atom.

Pour les systèmes dont les configurations matérielles ne sont pas certifiées, IBM ne fournit aucune déclaration de performances. Les considérations de performances pour les configurations matérielles certifiées sont détaillées ici : Chapitre 7, «Performances», à la page 95

Sur les systèmes avec le support hyperthreading, vérifiez que le support n'est pas actif pour éviter tout impact négatif sur les performances lorsque vous utilisez WebSphere Real Time for RT Linux.

Système d'exploitation

- Red Hat Enterprise Linux 5.3 MRG. Voir «Installation d'un environnement Real Time Linux», à la page 25.
- SUSE Linux Enterprise Real Time (SLERT) 10. Voir «Installation d'un environnement Real Time Linux», à la page 25.

Considérations

Vous devez tenir compte d'un certain nombre de points lors de l'utilisation de WebSphere Real Time for RT Linux.

- Dans la mesure du possible, exécutez une seule JVM temps réel sur un même système pour ne pas avoir plusieurs récupérateurs de place. Chaque machine JVM n'a pas connaissance des zones de mémoire de l'autre JVM. Une conséquence est que les cycles de récupération de place et les délais d'interruption ne peuvent pas être coordonnés sur les machines JVM, ce qui signifie qu'une machine JVM peut affecter négativement les performances du récupérateur de place sur une autre machine JVM. Si vous devez utiliser plusieurs machines JVM, assurez-vous que chacune d'elles est liée à un sous-ensemble spécifique de processeurs en lançant la commande **taskset**.
- Vous ne pouvez pas utiliser l'option **-Xdebug** et l'option **-Xnojit** avec du code précompilé par le compilateur AOT (Ahead-of-Time) car **-Xdebug** compile le code différemment par rapport au compilateur AOT et il n'est pas pris en charge.

  Pour déboguer le code, utilisez le code interprété ou compilé par JIT.
- Si vous utilisez l'interface com.sun.tools.javac.Main pour compiler le code source Java qui utilise le package javax.realtime, assurez-vous que `sdk/jre/lib/i386/realtime/jclSC170/realtime.jar` est inclus dans le chemin d'accès aux classes. La compilation ant est un exemple courant de ce type de compilation.
- Le package facultatif JavaComm peut être installé dans WebSphere Real Time for RT Linux et accessible depuis la machine JVM temps réel et non-temps réel. Pour plus d'informations sur l'installation et la configuration, voir http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/user/jcommchapter.html. La machine JVM temps réel dans WRT prend en charge l'API JavaComm pour une utilisation avec les unités d'exécution Java normales. Toutefois, aucune garantie n'existe concernant le déterminisme ou les performances temps réel lors de l'accès à des périphériques externes avec JavaComm. Par conséquent, n'utilisez pas JavaComm avec les unités d'exécution NHRT et RTT ou lorsque le comportement temps réel est nécessaire.
- Les caches partagés utilisés par les versions antérieures de WebSphere Real Time for RT Linux pour stocker le code précompilé et les classes ne sont pas compatibles avec ceux utilisés par cette version du produit. Vous devez régénérer le contenu des caches précédents.
- Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.
- La commande **ps** tronque les noms des unités d'exécution Java.

  La commande **ps** est limitée à 15 caractères. Si vous définissez un nom d'unité d'exécution de plus de 15 caractères, le nom est tronqué par la commande **ps**.
- WebSphere Real Time for RT Linux ne prend pas en charge l'authentification NTLoginModule (NTLM).

  NTLoginModule (NTLM) permet d'authentifier l'accès à un service Windows. L'authentification à l'aide de NTLM est prise en charge sur la plateforme Windows uniquement. Cela implique que WebSphere Real Time for RT Linux ne prend pas en charge l'authentification NTLM.

Chapitre 4. Installation de WebSphere Real Time for RT Linux

Pour installer le produit, procédez comme suit.

Fichiers d'installation

Les fichiers d'installation nécessaires sont les suivants.

IBM WebSphere Real Time for RT Linux est fourni dans deux types de package InstallAnywhere.

Packages installables

Les packages installables permettent de configurer votre systèmes. Par exemple, les programmes peuvent définir des variables d'environnement.

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

Packages d'archivage

Ces packages extraient les fichiers sur votre système mais n'effectuent aucune configuration.

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

Installation d'un environnement Real Time Linux

Pour pouvoir installer WebSphere Real Time for RT Linux, vous devez installer Real Time Linux.

Avant de commencer

Pour pouvoir installer WebSphere Real Time for RT Linux, vous devez installer une version 64 bits de Real Time Linux.

Red Hat Enterprise Linux 5.3 MRG

- Pour plus d'informations sur l'installation du composant temps réel de Red Hat Enterprise Linux 5.3 MRG, voir les instructions d'installation de 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

- Pour plus d'informations sur l'installation de SUSE Linux Enterprise Real Time 10, voir http://www.novell.com/products/realtime/eval.html

Lorsque vous utilisez un grand nombre de descripteurs pour charger différentes instances des classes, le message d'erreur "`java.util.zip.ZipException: error in opening zip file`" peut apparaître ou le type d'exception IOException peut être généré pour indiquer qu'un fichier n'a pas pu être ouvert. La solution consiste à augmenter le nombre de descripteurs en utilisant la commande **`ulimit`**. Pour identifier la limite en cours de fichiers ouverts, utilisez la commande :

```
ulimit -a
```

Pour pouvoir ouvrir plus de fichiers, utilisez la commande :

```
ulimit -n 8196
```

Installation à partir d'un package InstallAnywhere

Ces packages fournissent un programme interactif qui vous guide tout au long du choix des options d'installation. Vous pouvez exécuter ce programme sous forme d'interface graphique ou à partir d'une console système.

Avant de commencer

Votre système doit disposer des deux bibliothèques partagées suivantes :

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

Si vous ne disposez pas de la bibliothèque partagée `libstdc++.so.5`, il se peut que lors de l'installation s'affiche le cliché Javacore, contenant les erreurs suivantes :

```
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 vous installez un package installable, l'outil m-build doit être installé sur votre système. Dans le cas contraire, le programme d'installation ne peut pas enregistrer le nouveau package dans la base de données RPM. Pour savoir si l'outil rpm-build est installé, entrez la commande suivante :

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

Pourquoi et quand exécuter cette tâche

Les packages InstallAnywhere possèdent une extension de fichier `.bin`.

Il existe deux types de package :

Installable
L'installation de ces packages configure également votre système (par exemple, en définissant les variables d'environnement).

Archive
L'installation de ces packages extrait les fichiers sur votre système, mais n'effectue aucune configuration.

Procédure

- Pour installer le package de façon interactive, effectuez une installation avec opérateur.
- Pour installer le package sans aucune interaction supplémentaire avec l'utilisateur, effectuez une installation automatique. Vous pouvez choisir cette option si vous effectuez l'installation sur de nombreux système.
- A la fin de l'installation, suivez la procédure d'installation fournie dans cette section, notamment pour définir les variables d'environnement path et classpath.

Résultats

Le produit est installé.

Remarque : N'interrompez pas le processus d'installation, par exemple en appuyant sur Ctrl+C. Si vous interrompez le processus, il pourra être nécessaire de réinstaller le produit. Pour plus d'informations, voir «Installation interrompue», à la page 29.

Si vous utilisez un package installable, des messages vous indiquant la survenue d'un problème peuvent s'afficher. Par contre, l'installation des packages d'archives ne génère aucun message. Certains des messages pouvant s'afficher lors de l'utilisation d'un package installable figurent dans la liste ci-après :

Le programme d'installation ne peut pas être exécuté dans votre configuration. Il va se fermer.

Ce message d'erreur s'affiche lorsque votre ID utilisateur n'est pas autorisé à exécuter le processus d'installation. Etant donné que l'installation ne peut pas se poursuivre, le programme se ferme. Pour résoudre ce problème, relancez l'installation avec un ID utilisateur ayant des droits d'accès root.

Un package RPM est déjà installé. Désinstallez ce package avant de continuer.

Ce message indique qu'un package RPM est déjà installé. Etant donné que l'installation ne peut pas se poursuivre, le programme se ferme. Pour résoudre ce problème, désinstallez le package RPM avant de continuer.

Exécution d'une installation avec opérateur

Installez le produit à partir d'un module InstallAnywhere de façon interactive.

Avant de commencer

Avant de commencer l'installation, vérifiez que les conditions suivantes sont réunies :

- Si vous avez déjà installé WebSphere Real Time for RT Linux à partir d'un package RPM, vous devez désinstaller ce dernier avant de continuer.
- Vous devez disposer d'un ID utilisateur avec des droits d'accès root.

Procédure

1. Téléchargez le fichier du package d'installation dans un répertoire temporaire.
2. Placez-vous dans le répertoire temporaire.
3. Démarrez la procédure d'installation en entrant `./package` à l'invite shell où *package* est le nom du package que vous installez.

4. Sélectionnez une langue dans la liste qui s'affiche dans la fenêtre du programme d'installation, puis cliquez sur **Next (Suivant)**. La liste des langues disponibles dépend des paramètres régionaux définis pour votre système.
5. Lisez le contrat de licence, en utilisant la barre de défilement pour accéder à la fin du texte de licence. Pour pouvoir effectuer l'installation, vous devez accepter les conditions du contrat de licence. Pour accepter les conditions du contrat, sélectionnez le bouton d'option correspondant et cliquez sur **OK**.

   Remarque : Vous ne pouvez pas sélectionner le bouton d'option pour accepter le contrat de licence tant que vous n'avez pas lu le texte de licence jusqu'à la fin.
6. Le système vous invite à choisir le répertoire cible pour l'installation. Si vous ne souhaitez pas effectuer l'installation dans le répertoire par défaut, cliquez sur **Choose (Sélection)** pour sélectionner un autre répertoire à l'aide de la fenêtre de navigation. Une fois le répertoire d'installation sélectionné, cliquez sur **Next (Suivant)** pour continuer.
7. Le système vous invite à passer en revue les choix que vous avez effectués. Pour modifier votre sélection, cliquez sur **Previous (Précédent)**. Si vos choix sont corrects, cliquez sur **Install (Installer)** pour procéder à l'installation.
8. Une fois l'installation terminée, cliquez sur **Done (Terminé)** pour terminer.

Exécution d'une installation automatique

Si vous devez effectuer l'installation sur plusieurs systèmes et que vous connaissez déjà les options d'installation que vous souhaitez utiliser, il peut être intéressant d'utiliser le processus d'installation automatique. Vous effectuez l'installation une fois à l'aide du processus d'installation avec opérateur, puis vous utilisez le fichier de réponses résultant pour effectuer les autres installations sans interaction supplémentaire avec l'utilisateur.

Procédure

1. Créez un fichier de réponses en effectuant une installation avec opérateur. Utilisez l'une des solutions suivantes :
   - Spécifiez à partir de l'interface graphique que le programme d'installation devra créer un fichier de réponses. Le fichier de réponses s'appelle `installer.properties` et est créé dans le répertoire d'installation.
   - A partir de la ligne de commande, ajoutez l'option -r à la commande d'installation avec opérateur, en spécifiant le chemin d'accès complet au fichier de réponses. Par exemple :

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

   Exemple de contenu de fichier de réponses :

   ```
   INSTALLER_UI=silent
   USER_INSTALL_DIR=/mon_répertoire
   ```

   Dans cet exemple, */mon_répertoire* représente le répertoire d'installation cible que vous avez choisi pour l'installation.
2. Facultatif : Le cas échéant, modifiez les options dans le fichier de réponses.

   Remarque : Les packages d'archives présentent le problème connu suivant : les installations qui utilisent un fichier de réponses utilisent le répertoire par défaut même si vous avez modifié ce répertoire dans le fichier de réponses. S'il existe une installation précédente dans le répertoire par défaut, elle est remplacée.

Si vous créez plusieurs fichiers de réponses, chacun contenant des options d'installation différentes, spécifiez un nom unique pour chaque fichier de réponses, au format *monfichier*.properties.

3. Facultatif : Générez un fichier journal. Etant donné que vous effectuez une installation automatique, aucun message d'état n'est affiché à la fin de l'installation. Pour générer un fichier journal contenant l'état de l'installation, procédez comme suit :
   a. Définissez les propriétés système requises à l'aide de la commande suivante :

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

   b. Définissez la variable d'environnement suivante pour envoyer la sortie du journal sur la console.

      ```
      export LAX_DEBUG=1
      ```

4. Lancez une installation automatique en exécutant le programme d'installation du package à l'aide de l'option **-i** silent et de l'option **-f** pour spécifier le fichier de réponses. Par exemple :

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

   ```
   ./package -i silent -f
   /chemin/monfichier.properties 1>console.txt 2>&1
   ```

   Vous pouvez utiliser un chemin d'accès complet ou relatif au fichier de propriétés. Dans ces exemples, la chaîne `1>console.txt 2>&1` redirige les informations du processus d'installation des flux stderr et stdout vers le fichier journal `console.txt` dans le répertoire en cours. Prenez connaissance du fichier journal si vous pensez qu'il y a eu un problème durant l'installation.

   Remarque : Si le répertoire d'installation contient plusieurs fichiers de réponses, c'est le fichier de réponses par défaut, `installer.properties` qui est utilisé.

Installation interrompue

Si le programme d'installation du package est interrompu de façon inattendue durant l'installation (par exemple, si vous appuyez sur Ctrl+C), l'installation est endommagée et vous ne pouvez pas désinstaller ou réinstaller le produit. Si vous tentez d'effectuer une désinstallation ou une réinstallation, le message `Fatal Application Error` s'affiche.

Pourquoi et quand exécuter cette tâche

Pour résoudre ce problème, supprimez les fichiers et réinstallez en suivant les étapes indiquées ci-après.

Procédure

1. Supprimez le fichier de registre `/var/.com.zerog.registry.xml`.
2. Supprimez le répertoire contenant l'installation s'il a été créé. Par exemple, opt/IBM/javawrt3/.
3. Relancez le programme d'installation.

Problèmes connus et limitations

Les packages InstallAnywhere présentent certains problèmes connus ainsi que certaines limitations.

- Si vous ne disposez pas de la bibliothèque partagée `libstdc++.so.5` sur votre système, l'installation échoue et génère un cliché Javacore. Pour plus d'informations, voir «Installation à partir d'un package InstallAnywhere», à la page 26.
- L'interface graphique du package d'installation ne prend pas en charge le programme de lecture d'écran Orca. Vous pouvez utiliser le mode installation automatique à la place de l'interface graphique.
- Si après l'installation vous entrez `./package` pour redémarrer le programme, ce dernier affiche le message suivant :

  ```
  ENTREZ LE NUMERO DE VOTRE CHOIX OU APPUYEZ SUR <ENTREE> POUR
  ACCEPTER LA VALEUR PAR DEFAUT :
  ```

  Si vous appuyez sur Entrée pour accepter la valeur par défaut, le programme ne répond pas. Entrez un nombre, puis appuyez sur Entrée.
- Si vous installez le package, puis tentez à nouveau d'effectuer l'installation dans un mode différent (par exemple, mode console ou installation automatique), le message suivant peut s'afficher :

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

  Vous ne devriez pas voir ce message s'afficher si vous avez effectué l'installation en mode interface graphique et que vous exécutez à nouveau le programme d'installation en mode console. Si cette erreur s'affiche et que vous exécutez le programme pour sélectionner l'option de désinstallation (packages installables uniquement), utilisez plutôt la commande **`./_uninstall/uninstall`** comme indiqué dans la section «Désinstallation de WebSphere Real Time for RT Linux», à la page 33.

Packages installables uniquement

- Vous ne pouvez pas mettre à niveau une installation existante à l'aide des packages InstallAnywhere. Pour effectuer la mise à niveau d'WebSphere Real Time for RT Linux, vous devez commencer par désinstaller les précédentes versions.
- Il n'est pas possible d'installer deux instances différentes de la même version de WebSphere Real Time for RT Linux sur un même système en même temps, même si vous utilisez des répertoires d'installation différents. Par exemple, vous ne pouvez pas avoir simultanément WebSphere Real Time for RT Linux V3 dans le répertoire `/previous` et une installation d'actualisation de service WebSphere Real Time for RT Linux dans le répertoire `/current`. Le programme d'installation vérifie le numéro de version. S'il trouve un package existant avec le même numéro de version, vous êtes invité à désinstaller le package existant.
- Si le package est installé et que vous exécutez à nouveau le programme d'installation de package à l'aide de l'interface graphique, vous pouvez sélectionner la désinstallation du package. Cette option de désinstallation n'est pas disponible en mode automatique. Si vous réexécutez le programme d'installation du package en mode automatique, le programme s'exécute mais n'effectue aucune action.

Packages d'archives uniquement

- Si vous modifiez le répertoire d'installation dans un fichier de réponses, puis exécutez une installation automatique à l'aide de ce fichier de réponses, le programme d'installation ne tient pas compte du nouveau répertoire d'installation et utilise le répertoire par défaut. S'il existe une installation précédente dans le répertoire par défaut, elle est remplacée.

Définition du PATH

Après avoir défini la variable d'environnement **PATH**, vous pouvez exécuter une application ou un programme en tapant son nom à partir d'une invite shell.

Pourquoi et quand exécuter cette tâche

Remarque : Si vous modifiez la variable d'environnement **PATH** comme indiqué dans la section, vous remplacez les exécutables Java qui existent dans le chemin.

Vous pouvez définir le chemin d'accès à un outil en tapant le chemin chaque fois avant le nom de l'outil. Par exemple, si le kit SDK est installé dans `opt/IBM/javawrt3/`, vous pouvez compiler le fichier *monfichier.java* en tapant la commande suivante à partir d'une invite shell :

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

Pour éviter d'entrer le chemin complet à chaque fois :

1. Editez le fichier de démarrage du shell dans votre répertoire initial (généralement **.bashrc**, en fonction du shell) et ajoutez les chemins absolus à la variable d'environnement **PATH**. Par exemple :

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

2. Connectez-vous de nouveau ou exécutez le script de shell mis à jour pour activer le nouveau paramètre **PATH**.
3. Compilez le fichier à l'aide de l'outil **javac**. Par exemple, pour compiler le fichier *monfichier.java*, entrez ce qui suit à l'invite shell :

   ```
   javac -Xrealtime monfichier.java
   ```

   La variable d'environnement **PATH** permet à Linux de localiser les fichiers exécutables, tels que **javac**, **java** et **javadoc**, à partir du répertoire en cours. Pour afficher la valeur actuelle du chemin, entrez ce qui suit depuis une invite de commande :

   ```
   echo $PATH
   ```

Que faire ensuite

Voir «Définition du chemin d'accès aux classes» pour savoir si vous devez définir la variable d'environnement CLASSPATH.

Définition du chemin d'accès aux classes

La variable d'environnement **CLASSPATH** indique aux outils SDK, **java**, **javac**, et **javadoc**, où trouver les bibliothèques de classe Java.

Pourquoi et quand exécuter cette tâche

Définissez la variable d'environnement **CLASSPATH** explicitement uniquement dans les conditions suivantes :

- Vous avez besoin d'une bibliothèque ou d'un fichier de classes différent, que vous pouvez développer par exemple, et elle/il ne se trouve pas dans le répertoire de travail.
- Vous changez l'emplacement des répertoires `bin` et `lib` et ils n'ont plus le même répertoire parent.
- Vous prévoyez de développer ou d'exécuter des applications qui utilisent différents environnements d'exécution sur le même système.

Pour afficher la valeur en cours **CLASSPATH**, entrez les informations suivantes depuis une invite shell :

```
echo $CLASSPATH
```

Si vous développez et exécutez des applications qui utilisent différents environnements d'exécution, y compris d'autres versions que vous avez installées séparément, vous devez définir **CLASSPATH** et **PATH** explicitement pour chaque application. Si vous exécutez plusieurs applications simultanément et utilisez des environnements d'exécution différents, chaque application doit être exécutée dans son propre shell.

Si vous exécutez une seule version de Java à la fois, vous pouvez utiliser un script shell pour passer d'un environnement à l'autre.

Que faire ensuite

Voir «Test de l'installation» pour vérifier que l'installation a abouti.

Test de l'installation

Utilisez l'option **-version** pour déterminer si l'installation est correcte.

Pourquoi et quand exécuter cette tâche

L'installation Java est constituée d'une machine JVM standard et d'une machine JVM temps réel.

Procédure

Testez l'application comme suit :

1. Pour afficher les informations de version de la machine JVM standard, tapez la commande suivante depuis une invite shell :

   ```
   java -version
   ```

   Cette commande retourne les messages suivants si elle aboutit :

   ```
   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 vous envisagez d'utiliser la machine JVM standard, mais pas la machine JVM temps réel, voir les documents IBM User Guides for Java v7 on Linux.

   Remarque : Les informations de version sont correctes, mais les dates peuvent être postérieures à celles de l'exemple. Le format de la chaîne de date est aaaammjj suivi éventuellement d'informations supplémentaires spécifiques du composant.
2. Pour afficher les informations des versions de la machine JVM temps réel, tapez la commande suivante depuis une invite shell :

   ```
   java -Xrealtime -version
   ```

   Cette commande retourne les messages suivants si elle aboutit :

```
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
```

Remarque : Les informations de version sont correctes mais l'architecture de plateforme et les dates peuvent différer ce celles fournies dans l'exemple. Le format de la chaîne de date est `aaaammjj` suivi éventuellement d'informations supplémentaires spécifiques du composant.

Désinstallation de WebSphere Real Time for RT Linux

La procédure utilisée pour supprimer WebSphere Real Time for RT Linux dépend du type d'installation utilisé.

Avant de commencer

Concernant les packages installables InstallAnywhere, vous devez disposer d'un ID utilisateur avec des droits d'accès root.

Pourquoi et quand exécuter cette tâche

Il n'existe aucune procédure de désinstallation pour les packages d'archives InstallAnywhere. Pour supprimer un package d'archives de votre système, supprimez le répertoire cible que vous avez sélectionné lors de l'installation du package. Concernant les packages installables InstallAnywhere, vous pouvez désinstaller le produit à l'aide d'une commande ou en exécutant à nouveau le programme d'installation, comme indiqué dans les étapes ci-après.

Procédure

- Facultatif : Désinstallez manuellement à l'aide de la commande **uninstall**.
  1. Accédez au répertoire contenant l'installation. Par exemple :
     ```
     cd /opt/IBM/javawrt3
     ```
  2. Lancez la procédure de désinstallation en entrant la commande suivante :
     ```
     ./_uninstall/uninstall
     ```
- Facultatif : Si vous ne parvenez pas à localiser facilement le programme de désinstallation, vous pouvez aussi exécuter une autre installation avec opérateur. Le programme d'installation détecte alors que le produit est déjà installé, puis vous offre la possibilité de désinstaller l'installation précédente.

Chapitre 5. Exécution des applications IBM WebSphere Real Time for RT Linux

Ces informations sont très importantes et vous aideront à exécuter les applications en temps réel.

Planification et répartition des unités d'exécution

Le système d'exploitation Linux prend en charge diverses règles de planification. La règle de planification de partage de temps par défaut est SCHED_OTHER ; elle est utilisée par la plupart des unités d'exécution. SCHED_RR et SCHED_FIFO peuvent être utilisés par les unités d'exécution dans les applications temps réel. .

Le noyau détermine l'unité d'exécution suivante que doit exécuter le processeur. Le noyau gère la liste des unités d'exécution exécutables. Il recherche l'unité d'exécution ayant la priorité la plus élevée et il la sélectionne comme unité d'exécution suivante à exécuter.

Pour générer la liste des priorités et des règles, utilisez la commande suivante :

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

où la règle :

- `TS` est SCHED_OTHER
- `RR` est SCHED_RR
- `FF` est SCHED_FIFO
- `-` n'a pas de règle signalée

La sortie se présente comme suit :

```
  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
```

Cette sortie montre le processus Java, la règle de planification appliquée, l'unité d'exécution principale «-» (autre) et des unités d'exécution unités d'exécution temps réel avec des priorités comprises entre 11 et 89.

Pour interroger la règle de planification actuelle, utilisez la commande **`sched_getscheduler`** ou **`ps`**, comme indiqué dans l'exemple.

Pour plus d'informations sur les processus, voir «Techniques générales de débogage», à la page 100.

Priorités et règles des unités d'exécution Java temps réel

Les unités d'exécution temps réel, à savoir les unités d'exécution allouées sous la forme `java.realtime.RealtimeThread`, et les gestionnaires d'événements asynchrones, utilisent la règle de planification SCHED_FIF.

La planification et la répartition d'unité d'exécution Java temps réel fait partie de la spécification RTS (Real Time Specification) pour for Java (RTSJ).. Cette rubrique, y compris les règles de planification et la gestion des priorités des unités d'exécution Java temps réel, se trouve dans la section «Support pour RTSJ», à la page 9.

Utilisation du code compilé avec WebSphere Real Time for RT Linux

IBM WebSphere Real Time for RT Linux prend en charge plusieurs modèles de compilation de code en fournissant plusieurs niveaux de performance du code et de déterminisme.

Opération interprétée

Il s'agit du modèle de compilation de code le plus simple. L'interpréteur exécute une application Java, mais n'utilise pas du tout la compilation de code. L'interpréteur montre un bon déterminisme, mais il est très peu performant. Par conséquent, éviter d'utiliser ce mode de fonctionnement pour les systèmes de production.

Pour utiliser l'opération interprétée, définissez l'option **`-Xint`** dans la ligne de commande Java.

Compilation JIT (Just-In-Time) basse priorité

Le modèle de compilation par défaut dans WebSphere Real Time for RT Linux utilise un compilateur JIT (Just-In-Time) pour compiler les méthodes importantes d'une application Java lors de l'exécution de l'application. Dans ce mode, le compilateur JIT fonctionne de la même manière que l'opération du compilateur JIT dans une machine JVM non-temps réel. La différence réside dans le fait que le compilateur WebSphere Real Time for RT Linux JIT s'exécute avec une priorité plus basse que les unités d'exécution temps réel. Priorité plus base signifie que le compilateur JIT utilise les ressources système lorsque l'application n'a pas besoin d'exécuter des tâches temps réel. Par conséquent, le compilateur JIT n'affecte pas de manière significative les performances des tâches temps réel.

Le compilateur JIT utilise deux unités d'exécution pour les activités de compilation : l'unité d'exécution de compilation et l'unité d'exécution d'échantillonneur. Ces unités d'exécution s'exécutent avec une priorité plus basse que les tâches temps réel. L'unité d'exécution de compilation s'exécute en mode asynchrone pour l'application. Cela implique qu'une unité d'exécution n'attend pas que l'unité d'exécution de compilation ait terminé de compiler une méthode. L'unité d'exécution d'échantillonneur envoie un message asynchrone aux unités d'exécution d'application pour identifier la méthode active dans chaque unité d'exécution. Le traitement du message est court dans l'unité d'exécution d'application. Aucun message n'est envoyé si l'unité d'exécution d'échantillonnage ne peut pas s'exécuter du fait de l'existence de tâches temps réel plus prioritaires. L'utilisation du compilateur JIT a un faible impact sur le déterminisme, mais ce mode de compilation fournit les meilleures performances pour la plupart des utilisateurs.

Pour exécuter une application avec JIT avec une priorité basse, voir «Activation du JIT», à la page 53.

Code précompilé AOT (Ahead-Of-Time)

WebSphere Real Time for RT Linux compile les méthodes Java en code natif dans une étape de précompilation avant d'exécuter l'application. Avant WebSphere Real Time for RT Linux V2, l' étape de précompilation utilisait l'outil jxeinajar pour compiler les méthodes à l'aide d'un compilateur AOT (Ahead-Of-Time) et stockait le résultat dans des fichiers exécutables Java spéciaux. Ces fichiers peuvent être collectés dans des fichiers JAR liés. Lors de l'exécution d'une application, des fichiers JAR liés sont ajoutés au chemin d'accès aux classes pour que la machine JVM puisse charger le code AOT lorsque les classes des méthodes sont chargées depuis l'environnement JXE. En utilisant cette approche, le compilateur JIT devient complètement indisponible en définissant l'option **-Xnojit** sur la ligne de commande. L'application peut utiliser n'importe quel code AOT créé et l'interpréteur pour d'autres méthodes. Ce mode de fonctionnement fournit un haut niveau de déterminisme, car le compilateur JIT n'est pas présent et il n'existe donc pas d'unité d'exécution d'échantillonnage ni de réduction de performance de changement de contexte. La difficulté de compiler le code Java à l'avance tout en respectant la spécification Java implique que le code compilé AOT s'exécute généralement moins vite que le code compilé JIT, bien qu'il soit généralement beaucoup plus rapide que l'interprétation.

WebSphere Real Time for RT Linux V2, et les versions ultérieures, stocke le code AOT dans un cache de classes partagées et non dans des fichiers JXE en utilisant la technologie de classes partagées fournie dans les machines virtuelles IBM Java 6. L'outil admincache permet d'interroger le contenu d'un cache, liste tous les caches existants et remplit un cache avec des classes et du code AOT. La stockage du code compilé AOT présente des avantages dans la mesure où les fichiers JAR d'application ne sont pas modifiés et qu'il n'est pas nécessaire de modifier le chemin d'accès aux classes lors de l'exécution de l'application.

Un cache de classes partagées a une limite de taille pratique basée sur l'espace adresse virtuel disponible. Cela implique que la compilation AOT de tous les fichiers JAR n'est pas pratique. Une compilation AOT sélective doit être exécutée.

Lorsqu'une application exécute du code AOT dans un cache de classes partagées, le code AOT des méthodes d'une classe se charge automatiquement lorsque la classe se charge dans la machine JVM. Le coût supplémentaire de la charge d'une classe pour installer le code AOT de ses méthodes impose de précharger autant de classes que possible avant que les parties critiques de l'application s'exécutent.

L'utilisation du code précompilé AOT fournit le plus haut niveau de déterminisme avec de bonnes performances. Le code AOT peut être utilisé lorsque l'application s'exécute en définissant les options **-Xshareclasses** et **-Xaot**. L'option **-Xaot** est active par défaut.

Pour stocker et utiliser le code AOT avec un cache de classes partagées en utilisant l'outil admincache, voir «Utilisation de l'outil admincache», à la page 39. Vous trouverez les informations relatives à la migration de jxeinajar vers admincache dans la documentation WebSphere Real Time for RT Linux V2.

Pour un exemple d'exécution d'une application avec du code compilé AOT, voir «Exécution de l'exemple d'application avec AOT», à la page 87.

Mode mixte combinant du code précompilé AOT et la compilation JIT basse priorité

Le code compilé AOT et JIT peut être utilisé lorsque l'application s'exécute. Ce mode de fonctionnement peut fournir un très bon déterminisme avec de bonnes performances et de très bonnes performances pour les méthodes exécutées fréquemment. Ce mode a pour principal avantage de permettre d'utiliser la précompilation AOT pour que les parties les plus importantes de l'application ne s'exécutent jamais dans l'interpréteur, ce qui est généralement beaucoup plus lent que le code compilé AOT ou JIT. Il est inutile de précompiler chaque méthode, car le compilateur JIT peut identifier dynamiquement les méthodes interprétées qui s'exécutent fréquemment sans affecter de manière significatives les performances de l'application. Le mode mixte est le mode par défaut lorsque vous ajoutez l'option **-Xshareclasses** à la ligne de commande.

Pour exécuter l'application avec la compilation AOT et JIT mixte, voir «Exécution de l'exemple d'application avec AOT», à la page 87

Gestion explicite de la compilation

Dans les modes de compilation avec le compilateur JIT activé, l'API java.lang.Compiler peut être utilisée pour contrôler explicitement le fonctionnement du compilateur JIT. Le compilateur JIT compile les méthodes de la classe envoyée en utilisant la méthode compileClass(). La méthode compileClass() est synchrone. Par conséquent, la méthode ne s'exécute pas tant que les méthodes fournies n'ont pas été compilées. Une application peut utiliser compileClass() dans une phase d'initialisation en itérant sur les classes utilisées par la phase principale de l'exécution de l'application. Lorsque la phase d'initialisation se termine, appelez la méthode Compiler.disable() pour désactiver complètement les unités d'exécution de compilation et d'échantillonnage. Cette technique pose principalement un problème au niveau de la gestion de la liste des classes à charger et compiler dans la phase d'initialisation d'application, notamment au cours du développement de l'application.

Pour plus d'informations sur la gestion de la compilation dans une application, voir IBM Real-Time Class Analysis Tool for Java.

Présentation des options de ligne de commande de compilation

Vous pouvez exécuter une application avec JIT activé en utilisant l'option **-Xjit** ou sans JIT en utilisant l'option **-Xnojit**. **-Xjit** est le mode par défaut.

Vous pouvez exécuter une application avec le code AOT activé en utilisant les options **-Xshareclasses -Xaot**. Désactivez le code AOT en utilisant l'option **-Xnoaot**. **-Xaot** est l'option par défaut, mais elle n'a aucun effet si vous ne définissez pas également l'option **-Xshareclasses**, car le code AOT doit être stocké dans un cache de classes partagées.

Utilisation du compilateur AOT

Procédez comme suit pour précompiler le code Java. Cette procédure décrit l'utilisation de l'option **-Xrealtime** dans une commande **javac**, l'outil, **admincache** et les options **-Xrealtime** et **-Xnojit** avec la commande **java**.

Pourquoi et quand exécuter cette tâche

L'utilisation du compilateur AOT (ahead-of-time) implique que la compilation est séparée de l'exécution de l'application. En outre, vous pouvez compiler plus de méthodes simultanément et non pas simplement les méthodes fréquemment utilisées. Vous pouvez compiler tout dans une application ou simplement les classes, comme indiqué dans les étapes suivantes.

Remarque : Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

Procédure

1. Dans une invite shell, tapez :

   ```
   javac -Xrealtime source
   ```

   Cette commande crée le bytecode Java depuis la source pour l'utiliser dans l'environnement temps réel. Voir figure 2, à la page 9.
2. Placez les fichiers classe générés dans un fichier JAR. Par exemple, pour créer test.jar:

   ```
   jar cvf test.jar source
   ```
3. Depuis une invite shell, entrez :

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

   Cette commande précompile le fichier test.jar et écrit la sortie dans le répertoire de sortie `./aot`.
4. Depuis un shell invite, entrez : Pour exécuter le fichier en utilisant le code AOT dans un cache de classes partagées, dans une invite shell, entrez :

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

   Pour exécuter le fichier en utilisant le code AOT dans un cache de classes partagées, recompilez les méthodes fréquemment appelées et sans créer un fichier JAR dans une invite shell :

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

   Ces commandes utilisent les mêmes fichiers JAR que vous avez précompilés dans l'étape 3.

Utilisation de l'outil admincache

L'outil admincache permet de gérer les caches de classes partagées sur un poste de travail.

Dans le produit IBM WebSphere Real Time for RT Linux, vous pouvez utiliser l'outil admincache pour créer un cache de classes partagées contenant des classes ou des classes et du code compilé AOT. Une fois les caches créés, vous pouvez également utiliser l'outil pour analyser les caches existants.

Le cache de classes partagées permet de réduire la quantité de mémoire dans les scénarios à plusieurs JVM et d'accélérer le démarrage de l'application.

Les caches de classes partagées peuvent être utilisées par WebSphere Real Time for RT Linux en mode non-temps réel et en mode temps réel, mais les techniques de formatage, de création et de remplissage de cache diffèrent. Les caches en mode temps réel ne sont pas compatibles avec les caches en mode non-temps réel. En mode non-temps réel, les caches sont créés et remplis de la même manière que la

JVM standard. Cela implique que le cache est créé et rempli par le machine JVM lorsqu'elle exécute une application de manière transparente pour l'utilisateur. En mode temps réel, en utilisant l'option **-Xrealtime**, les caches de classes partagées doivent être créées et préremplies par admincache en utilisant l'option **-populate**. Les applications exécutées en mode temps réel peuvent lire le contenu du cache prérempli, mais elles ne peuvent pas le modifier.

Les caches de classes partagées créés en mode temps réel peuvent être utilisés uniquement lors de l'exécution d'une application en mode temps réel. Les caches de classes partagées créés en mode non-temps réel peuvent être utilisés uniquement lors de l'exécution d'une application en mode non-temps réel. Cela s'applique également à l'outil admincache. Pour gérer les caches créés par la machine JVM en mode temps réel, utilisez admincache avec l'option **-Xrealtime**. Pour gérer les caches créés par la machine JVM en mode non-temps réel, n'utilisez pas l'option **-Xrealtime**. Pour vous connecter à un cache de classes partagées lors de l'exécution, ajoutez l'option **-Xshareclasses** à la ligne de commande.

Vous pouvez créer plusieurs caches de classes partagées sur un poste de travail, chacun ayant un nom spécifique et se trouvant dans un répertoire donné. Lors de la création d'un cache, le nom du cache peut être défini par l'option **-cacheName** <*name*>. Le nom du cache ne doit pas dépasser 53 caractères.

Par défaut, les caches de classes partagées sont créés dans le répertoire `/tmp/javasharedresources`, mais vous pouvez changer cet emplacement en utilisant l'option **-cacheDir** <*directory*>. Le format interne d'un cache de classes partagées dépend des caractéristiques du poste de travail sur lequel il est créé. Cela implique que les caches de classes partagées ne peuvent pas être créés sur des unités réseau par mesure de sécurité. Une autre raison de cette restriction est l'impact de la lenteur et de l'imprévisibilité des performances lors de l'accès à un cache de classes partagées depuis un système de fichiers réseau.

Si vous ne définissez aucun nom de cache sur la ligne de commande, la valeur par défaut est **sharedcc_<user_login>**

Pour plus d'informations sur le fonction des caches partagés en mode non-temps réel, voir «Partage des données de classes entre les machines JVM en mode non-temps réel», à la page 95.

Remarque : Depuis IBM WebSphere Real Time for RT Linux V2 SR1 vous devez utiliser l'option **-classpath** avec l'option **-populate**.

Création d'un cache de classes partagées temps réel :

L'outil admincache permet de créer des caches de classes partagées accessibles en mode temps réel.

Remarque : Vous devez connaître les considérations de sécurité lors de la création des fichiers de cache de classes partagées avec les paramètres par défaut. Voir «Considérations de sécurité pour le cache de classes partagées», à la page 97 pour plus d'informations sur les considérations de sécurité relatives au cache de classes partagées et les informations sur la modification des autorisations par défaut.

L'option **-populate** de l'outil admincache permet de créer des caches de classes partagées. L'option est utilisée avec une liste de fichiers JAR, un répertoire ou une arborescence de répertoires pour rechercher les fichiers JAR. Pour chaque fichier JAR défini ou trouvé, admincache stocke chaque classe dans le fichier JAR dans le

cache de classes partagées. Les méthodes de classe sont aussi compilées par AOT et stockées dans le cache de classes partagées si vous ne définissez pas l'option **-noaot**.

Vous devez utiliser l'option **-classpath** avec **-populate**. Dans le cas contraire, le message suivant s'affiche :

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

L'option **-help** de l'outil admincache affiche la liste des options secondaires que vous pouvez utiliser pour contrôler le remplissage du cache par l'outil admincache.

```
$ admincache -Xrealtime -help
Syntaxe : admincache [option]*
  où [option] peut être :
     -help | -?               Action : affiche cette aide
     -Xrealtime              Utilisation dans un environnement temps réel
     -cacheName <name>   Définit le nom du cache partagé (utiliser %%u pour remplacer le nom d'utilisateur)
     -cacheDir <dir>     Définit l'emplacement des fichiers de cache JVM
     -listAllCaches           Action : liste de tous les caches de classes partagées
     -printStats              Action : impression des statistiques de cache
     -printAllStats           Action : impression de statistiques de cache plus détaillées
     -destroy                 Action : destruction du cache nommé (ou par défaut)
     -destroyAll              Action : destruction de tous les caches
     -populate                Action : création d'un cache et remplissage du cache
       -searchPath <path>  spécifie le répertoire des fichiers si aucun
                           fichier n'est spécifié (la valeur par défaut est .)
                           Une seule option -searchPath peut être définie
            -classpath <class path> Définit le chemin d'accès
                          aux classes à utiliser lors de l'exécution pour accéder au cache
                           L'option -classpath est nécessaire
       -[no]recurse        [do not] Récursion dans les sous-répertoires pour rechercher
                           les fichiers à convertir
                           (par défaut pas de récursion)
       -[no]grow           Si le cache défini existe, [ne pas] y ajouter
                          (par défaut, ne pas augmenter)
                           Si -grow est sélectionné, le cache indiqué
                           sera supprimé s'il est présent
       -verbose            Affiche les messages d'avancement de chaque JAR
       -noisy              Affiche les messages d'avancement de chaque classe dans chaque JAR
       -quiet              Supprime tous les sortie
       -[no]aot            Exécute également la compilation AOT sur les méthodes
                           après le stockage des classes dans le cache
       -aotFilter <signature>  Seules les méthodes correspondantes sont compilées par
                           AOT et stockées dans le cache
          e.g. -aotFilter {mypackage/myclass.mymethod(I)I} compile
                           uniquement mymethod(I)I
          e.g. -aotFilter {mypackage/myclass.mymethod*} compile tout élément mymethod
          e.g. -aotFilter {mypackage/myclass.*} compile toutes les méthodes issues de myclass
       -aotFilterFile <file>   Seules les méthodes correspondant à celles du fichier
                           sont compilées par AOT et stockées
                           dans le cache (Le fichier en entrée doit avoir été créé
                           par -Xjit:verbose={precompile},
                           vlog=<fichier>)
       -printvmargs        Imprime les arguments VM nécessaires pour accéder au cache rempli
                           lors de l'exécution
       [jar file]*.[jar][zip]  Liste explicite des fichiers JAR pour remplir le cache
                           Si aucun fichier n'est défini, tous les fichiers .[jar][zip] dans le
                           chemin de recherche searchPath sont convertis.
Une seule action exactement doit être définie.
```

Remarque : Lors de l'utilisation de caches de classes partagées, le nom défini par l'option **-cacheName** ne doit pas dépasser 53 caractères.

Une liste de fichiers JAR peut être définie. Dans ce cas, seules les classes des fichiers JAR sont ajoutées au cache des classes partagées. Si vous ne définissez pas une liste de fichiers JAR, utilisez l'option **-searchPath <path>** pour définir une arborescence de répertoire pour rechercher les fichiers .jar ou .zip. L'option **-recurse** est l'option par défaut. Cela implique que les fichiers .jar et .zip sont recherchés de manière récursive dans l'arborescence de répertoires. Les options **-norecurse** impliquent que la recherche est effectuée uniquement dans le répertoire défini. Définissez l'option **-classpath <class path>** pour que l'outil admincache puisse rechercher toutes les classes nécessaires au traitement des fichiers JAR définis. Les classes sont chargées dans la machine JVM lors du remplissage du cache de classes partagées. Par conséquent, il est important que toutes les classes et superclasses référencées puissent être recherchées par l'outil admincache lorsqu'il tente de charger une classe depuis un fichier JAR.

L'option **-grow** spécifie qu'un nouveau fichier JAR est ajouté au contenu du cache existant, s'il existe un cache de classes partagées de même nom dans le répertoire cache. L'option **-nogrow** spécifie qu'un nouveau fichier JAR remplace l'ancien contenu du cache, s'il existe un cache de classes partagées de même nom dans l'ancien répertoire cache. L'option **-grow** permet d'ajouter de nouveaux fichiers JAR qui n'existent pas dans le cache de classes partagées et non pas de remplacer les classes modifiées. N'utilisez pas l'option **-grow** pour mettre à jour les classes qui se trouvent dans le cache et qui ont été modifiées suite à des modifications de l'application. Pour mettre à jour les classes existantes, créez un tout nouveau cache avec le contenu de classes actuel. Si vous ne mettez pas à jour le cache de classes partagées lorsque vous changez une classe, l'application exécute correctement le nouveau contenu de classe, mais elle ne tire pas parti du cache de classes partagées. Cela s'explique par le fait que la classe partagée sera chargée depuis le disque et non pas le cache de classes partagées. Le chargement de la classe depuis le disque implique que le code compilé AOT ne peut pas utiliser la classe. Régénérez le cache de classes partagées lorsque vous changez une classe.

Utilisez les options **-quiet**, **-verbose** et **-noisy** pour contrôler le niveau de détail fourni par admincache.

Pour définir la précompilation AOT (Ahead-Of-Time) des méthodes dans les classes du cache des classes partagées, utilisez l'option **-aot**. Pour empêcher la compilation AOT et stocker uniquement les classes dans le cache des classes partagées, utilisez l'option **-noaot**. L'option **-aot** est l'option par défaut.

Pour précompiler des méthodes de manières sélective, utilisez l'option **-aotFilter** <*signature*> ou **-aotFilterFile** <*file*>. La <*signature*> est une expression régulière simplifiée d'une signature de méthode, placée entre accolades, où '*' peut remplacer n'importe quelle séquence de caractères. Il peut être nécessaire de placer la <*signature*> entre guillemets pour que le shell n'interprète pas les caractères dans la signature de méthode.

Le tableau 4 montre des exemples de l'option <*signature*>.

Tableau 4. Exemples de l'option <signature>

Signature	Signification
`-aotFilter '{java/lang/*}'`	AOT compile les méthodes dans le package java/lang.
`-aotFilter '{*.sample*}'`	AOT compile les méthodes commençant par "sample".

Tableau 4. Exemples de l'option <signature> (suite)

Signature	Signification
`-aotFilter '{mypackage/ myclass.mymethod(I)I}'`	AOT compile la méthode avec cette signature exacte.

L'option **-aotFilterFile** <*file*> utilise le contenu de <*file*> pour sélectionner les méthodes pour la compilation AOT. Aucune autre méthode AOT n'est compilée. Le contenu de <*file*> est généré lors de l'exécution de l'application en utilisant l'option **-Xjit:verbose={precompile},vlog=**<*file*>. La sortie prolixe stockée dans <*file*> utilise un format interne. Ce format est nécessaire à l'option **-aotFilterFile**.

Remarque : L'option **-vlog=**<*file*> ne génère pas directement le fichier "file". Une date et un ID de processus sont ajoutés à "file" lorsque la sortie prolixe est générée. En définissant l'option **-Xjit:verbose={precompile},vlog=my_file**, le nom de fichier généré est similaire à `my_file.<date>.<#>.<process id>`Les zones supplémentaires facilitent la génération de fichiers journaux individuels prolixes dans les scénarios à plusieurs machines JVM dans lesquels il peut être difficile de fournir des options de ligne de commande à une machine JVM ou d'utiliser des options de ligne de commande **-Xjit** différentes avec des machines JVM différentes. Dans un scénario à une seule machine JVM, ces valeurs sont ajoutées au nom de fichier fourni dans la ligne de commande.

Un fichier généré peut être utilisé avec l'option **-aotFilterFile** sans modification. Plusieurs fichiers journaux prolixes générés par plusieurs exécutions d'application utilisant l'option **-Xjit:verbose={precompile},vlog=<file>** peuvent être concaténés et fournis à admincache en utilisant l'option **-aotFilterFile**.

L'option **-printvmargs** permet de garantir la fourniture des arguments corrects dans la ligne de commande lors de l'exécution de l'application.

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

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

Processing complete

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

Dans cet exemple, la dernière ligne de la sortie contient les options qui doivent être ajoutées à ligne de commande pour que les classes et les méthodes AOT stockées dans le cache de classes partagées soient utilisées. Pour utiliser les options de cet exemple, entrez la commande suivante :

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

Gestion des caches de classes partagées avec admincache :

L'outil admincache contient des utilitaires pour gérer les caches de classes partagées sur le système.

L'outil admincache fournit des utilitaires pour effectuer diverses activités.

- Enumération des caches de classes partagées dans un cache

- Fourniture d'informations sur le contenu d'un cache de classes partagées
- Suppression de certains ou de tous les caches dans un répertoire de cache

Liste des caches de classes partagées disponibles :

L'outil admincache fournit la liste des caches de classes partagées présents dans un cache.

Pour obtenir la liste de tous les caches de classes partagées présents dans un cache, utilisez l'option **-listAllCaches** et définissez le répertoire cache en utilisant l'option **-cacheDir**.

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

Liste de tous les caches dans 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
```

Dans cet exemple, il existe deux caches de classes partagées compatibles dans le répertoire cache par défaut :

- Le cache par défaut d'un utilisateur avec le nom de connexion *username*
- Un autre cache appelé *rtCache*

L'exemple montre également un cache incompatible appelé *nonrtCache*. Le cache *nonrtCache* a été créé par la machine JVM exécutée en mode non temps réel. Cela implique qu'il n'est pas accessible en utilisant l'option **-Xrealtime**.

La machine JVM en mode temps réel JVM peut voir les caches créés en mode non-temps réel. La machine JVM en mode non-temps réel ne peut pas voir les caches créés en mode temps réel.

```
$ admincache -listAllCaches
J9 Java(TM) admincache 1.0
Eléments sous licence - Propriété d'IBM

(c) Copyright IBM Corp. 1991, 2008  All Rights Reserved
IBM est une marque d'IBM Corp.
Java ainsi que tous les logos et toutes les marques incluant Java
sont des marques
d'Oracle Corporation.


Liste de tous les caches dans 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
```

Dans cet exemple, *nonrtCache* figure dans la liste et il est indiqué comme étant compatible, car **-Xrealtime** n'est pas défini.

Analyse du contenu des caches de classes partagées :

L'outil admincache fournit le contenu d'un cache de classes partagées.

Vous pouvez utiliser l'option **-printStats** de l'outil admicache pour obtenir la description générale du contenu principal d'un cache de classes partagées. Pour plus d'informations sur un cache, dans un répertoire cache, utilisez les options **-cacheName** et **-cacheDir**. L'exemple suivant fournit des informations sur le cache *nonrtCache* dans le répertoire cache par défaut.

```
$ 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
```

Remarque : Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

Il existe plusieurs informations importantes sur ce cache :

- Sa taille : `cache size = 16776852`.
- L'espace disponible sur le cache indiqué sous la forme `free bytes = 14070360`. Vous pouvez calculer que le cache est plein approximativement à 16 %.
- Nombre de classes stockées : `# ROMClasses = 372`.
- Nombre de méthodes AOT qui y sont stockées : `# AOT Methods = 981`.

Pour plus de détails sur les informations fournies par l'option **-printStats** dans l'outil admincache, voir utilitaire printStats.

L'option **-printAllStats** fournit une description plus détaillée du contenu d'un cache de classes partagées. Les informations incluent la liste des classes et des méthodes AOT stockées dans le cache. La sortie de l'option **-printAllStats** est prolixe.

Les classes dans le cache sont indiquées par des lignes similaires à :

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

Cette ligne indique que la classe java/lang/ClassLoader se trouve dans le cache. Les adresses sont internes au cache de classes partagées et sont rarement utiles, sauf pour les diagnostics.

Les méthodes AOT dans le cache sont indiquées par des lignes similaires à :

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

Ces lignes indiquent que la méthode callerClassLoader de la classe java/lang/ClassLoader est contenue dans le cache. Les adresses listées sont des adresses de cache de classes partagées internes. La sortie de l'option **-printAllStats** ne contient pas la signature de chaque méthode AOT dans le cache, où la signature est constituée des types de paramètres et du type de retour.

Pour plus de détails sur les informations fournies par l'option **-printAllStats** dans l'outil admincache, voir utilitaire printAllStats.

Destruction des caches de classes partagées :

L'outil admincache fournit des options qui permettent d'effacer un cache ou tous les caches dans un répertoire cache.

L'option **-destroy** de l'outil admincache permet d'effacer un cache dans une répertoire cache, si l'utilisateur y est autorisé. L'option **-destroyAll** permet d'effacer tous les caches, si l'utilisateur est autorisé. Par exemple :

```
$ admincache -Xrealtime -destroy

admincache 1.02

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

Après avoir effacé le cache, la liste des caches de classes partagées disponibles dans le répertoire cache par défaut ne contient plus le cache effacé :

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

Liste de tous les caches dans 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'option **-destroyAll** supprime tous les caches du répertoire cache défini, qu'ils soient compatibles ou non avec la machine JVM. L'option **-destroyAll** doit être utilisée avec la plus grande précaution :

```
$ 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
```

Il en résulte que plus aucun cache de classes partagées n'est disponible sur la machine :

```
$ admincache -Xrealtime -listAllCaches

admincache 1.02

JVMSHRC005I No shared class caches available
```

Si l'utilisateur en cours n'est pas autorisé à accéder à un cache, le cache n'est pas détruit par l'option **-destroy** ou **-destroyAll**.

Tailles pratiques des caches de classes partagées :

L'outil admincache fournit des informations pour dimensionner les caches de classes partagées.

Pour les petites applications, un cache de classes partagées peut être rempli avec toutes les classes et méthodes qui produisent un cache de très grande taille. Pour les grandes applications, le cache de classes partagées résultant peut devenir trop grand à des fins pratiques, car un processus JVM doit avoir un espace adresse virtuel suffisant pour traiter tout le contenu du cache de classes partagées. Vous devez tenir compte de certains points lorsque vous utilisez la technologie de cache de classes partagées.

Le cache de classes partagées doit être totalement adressable virtuellement dans n'importe quelle JVM qui s'y connecte, ce qui implique que vous devez éviter d'utiliser des caches de classes partagées de plus de 700 Mo. L'outil admincache peut prévoir la taille d'un cache. Si l'outil indique que la taille du cache va être supérieure à 700 Mo, un message s'affiche pour demander de stocker moins de classes ou d'être plus sélectif concernant les méthodes AOT stockées dans le cache.

```
$ admincache -Xrealtime -populate veryBigJar.jar -cp <my class path>

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'outil admincache prévoit une taille de cache limitée en fonction de la taille totale des fichiers JAR définir ou trouvés à remplir. Cela implique que la prévision peut ne pas être exacte si le fichier JAR contient de nombreux fichiers qui ne sont pas des fichiers classe. Pour affiner la prévision de taille de cache, créez des versions temporaires des fichiers JAR qui contiennent uniquement les fichiers classe. Si l'outil admincache génère toujours un avertissement, précompilez les méthodes avec AOT dans le fichier JAR plus sélectivement en utilisant l'option **-aotFilter** <*pattern*> ou **-aotFilterFile** <*file*>. Le message de l'outil admincache rappelle que la prévision ne tient pas compte des méthodes AOT filtrées par ces options.

Pour remplacer l'avertissement et remplir le cache, ajoutez l'option **-Xscmx** indiquée à la ligne de commande admincache. Si la taille prévue est très grande, l'outil admincache peut ne pas pouvoir créer un cache de classes partagées avec la taille nécessaire. Pour résoudre ce problème, réduisez la taille du cache jusqu'à ce que l'outil admincache puisse continuer.

Lorsque le dernier cache est écrit sur le disque, sa taille correspond à la taille nécessaire pour contenir les classes et les méthodes AOT définies. Cela implique que la définition d'une grande taille de cache initiale ne pose pas de problème.

Stockage des classes SDK dans un cache de classes partagées :

La création d'un cache contenant tous les fichiers JAR depuis le SDK peut être nécessaire pour toutes les applications.

Le nombre et la taille des fichiers JAR dans le kit SDK implique que toute tentative de création d'un cache contenant tous ces fichiers JAR génère un message indiquant que le cache résultat sera trop grand. Pour la plupart des applications, les fichiers JAR SDK ne sont généralement pas référencés.

Les principaux fichiers JAR SDK se trouvent dans le répertoire `SDK/jre/lib`. Pour la plupart des applications, le fichier le plus important de ces fichiers JAR est `rt.jar` ; il s'agit d'un nouveau fichier dans les éditions Java 6. `rt.jar` est une collection de classes qui était stockée dans des fichiers JAR distincts avant l'édition Java 6. Le remplissage d'un cache de classes partagées avec `rt.jar` seul et la compilation de toutes ses méthodes avec le compilateur AOT créent un cache de 300 Mo environ. La plupart des méthodes des classes `rt.jar` ne sont pas référencées par une application standard. Pour remplir un cache de classes partagées avec `rt.jar` :

1. Remplissez les classes uniquement de `rt.jar` vers le cache de classes partagées. Cette opération consomme environ 50 Mo du cache.
2. Utilisez l'option **`-aotFilterFile`** <*file*> pour compiler uniquement les méthodes que peut utiliser le programme. Vous pouvez générer <*file*> en exécutant l'application.

Le kit SDK contient d'autres fichiers communément utilisés et d'autres fichiers JAR importants, notamment :

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

`realtime.jar` contient l'implémentation IBM de RTSJ (Real Time Specification for Java). Si l'application utilise les fonctions de la spécification RTSJ, stockez le fichier `realtime.jar` dans un cache de classes partagées pour améliorer le déterminisme. `vm.jar` contient des classes JVM internes communément utilisées dans toutes les applications. `java.util.jar` contient des classes de conteneur et doit être stocké dans le cache de classes partagées de chaque application pour améliorer le déterminisme.

D'autres fichiers dans les répertoires `sdk/jre/lib` et `sdk/jre/lib/ext` peuvent être stockés dans un cache de classes partagées si une application utilise ces classes. La méthode la plus simple pour déterminer si l'application utilise ces classes consiste à utiliser l'option **`-verbose:dynload`** lors de l'exécution du programme. L'option **`-verbose:dynload`** décrit uniquement les classes chargées par l'exécution actuelle de l'application. Par 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>
```

Cet exemple de sortie montre trois classes chargées depuis des fichiers JAR SDK différents. La classe java/io/InputStreamReader a été chargée depuis `rt.jar`. Les classes java/util/LinkedHashSet et java/util/HashSet ont été chargées depuis `java.util.jar`.

Autres considérations sur admincache :

Informations utiles pour utiliser admincache.

Remplissage du cache et redimensionnement de la mémoire pérenne

Lorsque l'outil **admincache** remplit un cache de classes partagées en temps réel, il doit charger chaque classe au fur et à mesure du processus. Chaque classe consomme de la mémoire pérenne et, par conséquent, il peut arriver que la taille de la mémoire pérenne par défaut ne soit pas suffisamment grande pour toutes les classes demandées. Si l'outil **admincache** envoie une erreur OutOfMemory lors du remplissage avec de nombreuses classes, essayez d'augmenter la taille de la mémoire pérenne au-delà de 16 Mo en utilisant l'option **-Xgc:immortalMemorySize=32M**.

En cas de changement de classes

Si un fichier classe est modifié sur le disque, la technologie de cache de classes partagées détecte automatiquement que la version mise en cache de la classe dans un cache de classes partagées ne doit pas être utilisée. Le programme fonctionne correctement, mais il ne peut pas tirer complètement parti du cache de classes partagées et les méthodes AOT de la classe ne sont pas utilisées. Si vous changez une classe dans l'application, recréez le cache de classes partagées. N'essayez pas d'utiliser l'option **-grow** pour remplir uniquement le fichier JAR contenant la classe modifiée, car cette option n'est pas appropriée lorsque le fichier JAR existe déjà dans le cache.

Gestion des caches partagés

Les caches partagés nécessitent un espace adresse, même si aucun fichier n'est chargé. Voir «Gestion de la mémoire par la machine JVM IBM», à la page 108 pour plus d'informations sur la manière dont les caches de classes partagées consomment de la mémoire dans le processus JVM.

Stockage des fichiers JAR précompilés dans un cache de classes partagées

Vous pouvez stocker toutes les classes Java fournies par IBM ou certaines d'entre elles ou les inclure dans un cache de classes partagées. Ce processus utilise l'option **-Xrealtime** avec **javac** et l'outil **admincache** pour stocker les classes dans un cache de classes partagées.

Avant de commencer

Les fichiers JAR stockés de manière anticipée dans un cache de classes partagées sont pris en charge uniquement avec l'option **-Xrealtime** et lors de l'exécution de Java avec l'option **-Xrealtime**. Vous pouvez utiliser les mêmes fichiers JAR lors de l'exécution avec ou sans l'option **-Xrealtime**, mais les fichiers JAR stockés dans le cache peuvent être utilisés uniquement lorsque **-Xrealtime** est défini.

Remarque : Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

Pourquoi et quand exécuter cette tâche

Vous pouvez stocker les fichiers JAR dans un cache de classes partagées en utilisant l'outil **admincache**. **admincache** permet de générer l'application de trois manières.

Remarque :
- Si vous avez défini une temporisation sur le système Linux, vous devez le remplacer lors de la précompilation de fichiers volumineux pour que la compilation n'expire pas et que le fichier JAR soit créé.

Précompilation de toutes les classes et méthodes dans une application :

Cette procédure précompile toutes les classes dans une application. Elle stocke un groupe de fichiers JAR dans un cache de classes partagées. Toutes les méthodes dans toutes les classes dans ces fichiers JAR sont stockées dans le cache. Les fichiers JAR optimisés ont toutes les méthodes compilées.

Pourquoi et quand exécuter cette tâche

Dans le cadre de cet exemple, l'application réside dans le répertoire défini par la variable d'environnement *$APP_HOME* et les fichiers JAR se trouvent dans le sous-répertoire *$APP_HOME*/lib. L'application utilise également des classes depuis celles fournies par IBM dans core.jarrt.jar. Dans ce cas, vous pouvez précompiler uniquement le code d'application, à savoir main.jar et util.jar.

Par défaut, le cache de classes partagées se trouve dans /tmp/javasharedresources. Utilisez l'option **-cacheDir** pour placer le cache dans un répertoire différent. Vous ne pouvez pas créer un cache dans un système de fichiers réseau.

Procédure

1. Depuis une invite shell, tapez : cd *$APP_HOME*
   où *$APP_HOME* est le répertoire de l'application.
2. Depuis une invite shell, tapez cd *$APP_HOME*/lib. *$APP_HOME*/lib est le répertoire des fichiers main.jar et util.jar.
3. Depuis une invite shell, tapez admincache -Xrealtime -populate -aot -classpath *$APP_HOME*/lib -searchPath *$APP_HOME*/lib -norecurse . Cette procédure optimise chaque fichier JAR trouvé dans le répertoire *$APP_HOME*/lib en affichant les informations d'avancement et en créant le nouveau fichier JAR dans le répertoire *$APP_HOME*/aot. Vous pouvez spécifier un nom de cache avec **-cacheName <name>**, mais un nom par défaut reposant sur les information de connexion de l'utilisateur est utilisé si vous n'en définissez aucun.

   Remarque : Le nom spécifié par l'option **-cacheName** ne doit pas dépasser 53 caractères.
4. Depuis une invite shell, tapez admincache -Xrealtime -listAllCaches pour afficher l'existence du cache.

Que faire ensuite

Pour plus d'options, spécifier `admincache -Xrealtime -help`.

Précompilation des méthodes fréquemment utilisées :

Vous pouvez utiliser la compilation AOT dirigée par profil pour précompiler uniquement les méthodes qu'utilise fréquemment l'application. La compilation AOT stocke un groupe de fichiers JAR dans un cache de classes partagées en utilisant un fichier d'options généré en exécutant l'application avec une option spéciale **-Xjit:verbose={precompile},vlog=optFile**. Seules les méthodes figurant dans le fichier d'options sont précompilées.

Avant de commencer

Avant de commencer, créez la liste des méthodes généralement compilées par un compilateur JIT.

Pourquoi et quand exécuter cette tâche

Vous pouvez modifier le fichier généré par l'option **-Xjit:verbose={precompile}**. Le fichier est une spécification explicite des méthodes à précompiler. Ces méthodes sont spécifiques, à savoir qu'elles contiennent la signature complète de chaque méthode à compiler, ce qui permet de compiler `com/acme/sample.myMethod(J)V`, mais pas `com/acme/sample.myMethod(I)V`.

Remarque : Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

Procédure

1. Depuis une invite shell, entrez :

   ```
   cd $APP_HOME
   ```

   où *$APP_HOME* est le répertoire de l'application.
2. Depuis une invite shell, entrez :

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

   où :

   - *app.precompileOpts* est le nom du fichier journal qui contient la liste des méthodes compilées avec JIT.
   - *applicationName* est le nom de l'application.

   Cette commande crée la liste des méthodes compilées en utilisant JIT.
3. Depuis une invite shell, entrez :

   ```
   cd $APP_HOME/lib
   ```

   $APP_HOME/lib est le répertoire où sont stockés les fichiers JAR de l'application.
4. Pour compiler tous les exemples de méthodes d'application dans le cache, entrez :

   ```
   admincache -Xrealtime -populate -cacheName myCache \
              -aotFilterFile $APP_HOME/app.precompileOpts \
              -cp $APP_HOME/lib/demo.jar
   ```
5. Pour compiler `realtime.jar` et `vm.jar` dans le cache, entrez :

```
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. Pour compiler `rt.jar` dans le cache, entrez :

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

7. Pour tester cette commande, exécutez l'application avec l'option **-nojit** qui utilise le code dans le cache. Depuis une invite shell, tapez :

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

où *applicationName* est le nom de l'application.

Précompilation des fichiers fournis par IBM :

Vous pouvez précompiler les fichiers fournis par IBM, tels que `rt.jar`, pour établir un compromis entre les performances et la prévisibilité.

Pourquoi et quand exécuter cette tâche

La précompilation est similaire à la précompilation des fichiers JAR d'application, mais une condition supplémentaire s'applique lors de l'exécution ; vous devez définir le chemin des classes d'amorçage correctement pour utiliser ces fichiers à la place des fichiers dans l'environnement JRE. Vous pouvez effectuer cette opération avec l'option **-Xshareclasses** qui demande à la machine JVM de consulter le cache des classes défini avant les emplacements de chemin de classe par défaut.

Remarque : Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

Précompilez `rt.jar` pour l'utiliser avec l'application :

Procédure

1. Dans une invite shell, tapez : `cd $JAVA_HOME/lib` , où *$JAVA_HOME* est le répertoire de base Java.
2. Exécutez l'outil **admincache**. Dans une invite shell, entrez

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

   Cette commande remplit le cache `myCache` avec le résultat de la compilation du fichier IBM `rt.jar`.
3. Exécutez l'application en définissant l'option **-Xshareclasses** pour définir le nom du cache. Pour exécuter l'application, entrez :

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

Compilateur JIT (Just-In-Time)

Vous pouvez contrôler quand et comment le compilateur JIT fonctionne en utilisant la classe java.lang.Compiler fournie avec la bibliothèque de classes SDK standard. IBM prend complètement en charge les méthodes Compile.compileClass(), Compiler.enable() et Compiler.disable().

Par exemple, si vous voulez mettre en fonctionnement votre application et savez que les principales méthodes ont été compilées, vous pouvez appeler la méthode

Compiler.disable() après avoir préparé l'application et être certain que la compilation JIT ne sera pas exécutée pendant le reste de l'exécution de l'application.

Vous pouvez contrôler la compilation des méthodes de deux manières :

- Définissez un ensemble de méthodes que vous pouvez compiler :

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

  , où <*method specification*> correspond à la liste de toutes les méthodes chargées à ce stade et à compiler. <*method specification*> décrit un nom de méthode qualifié complet. Un astérisque indique une correspondance générique.

  Par exemple, pour compiler toutes les méthodes qui commencent par java.lang.String déjà chargées, spécifiez :

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

  Remarque : Cette commande compile non seulement les méthodes dans la classe java.lang.String, mais également dans la classe java.lang.StringBuffer, ce qui peut ne pas correspondre à ce que vous souhaitez. Pour compiler uniquement les méthodes dans la classe java.lang.String, spécifiez :

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

- Spécifiez que toutes les méthodes dans la files d'attente de compilation seront compilées avant l'exécution de cette unité d'exécution et sa poursuite :

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

  Vous pouvez vérifier que la file d'attente de compilation est vide avant de désactiver le compilateur. Une technique standard de compilation d'un groupe de méthodes et de classes peut se présenter comme suit :

```
Compiler.enable();                              // garantit que le compilateur est actif
Compiler.command("{com.mycompany.*}(compile)"); // mise en file d'attente de toutes les méthodes à compiler
Compiler.command("waitOnCompilationQueue");     // attente de la fin de la compilation de ces méthodes
Compiler.disable();                             // mise hors tension de l'ordinateur
```

Déterminisme pendant les transitions JNI

Par défaut, la compilation JIT génère du code optimisé pour des transitions J2N (Java to natives) haute performance. Un déterminisme réduit peut apparaître lors du rechargement d'une bibliothèque native en utilisant la séquence de code suivante :

```
RegisterNatives / UnregisterNatives / RegisterNatives
```

Pour revenir au code plus lent et plus déterministe, utilisez l'option de ligne de commande **-Xjit:disableDirectToJNI**.

Activation du JIT

Vous pouvez activer le compilateur JIT de plusieurs manières. Les options de ligne de commande remplacent la variable d'environnement **JAVA_COMPILER**.

Procédure

- Affectez à la variable d'environnement **JAVA_COMPILER** la valeur "jitc" avant d'exécuter l'application Java. A l'invite shell, entrez
  - **Pour l'interpréteur de commandes Korn :** `export JAVA_COMPILER=jitc`

    Remarque : Les commandes de l'interpréteur de commandes Korn sont utilisées dans ces informations, sauf indication contraire.
  - **Pour l'interpréteur de commandes Bourne :**

```
JAVA_COMPILER=jitc
export JAVA_COMPILER
```

– **Pour l'interpréteur de commandes C :** `setenv JAVA_COMPILER jitc`

Si la valeur chaîne vide est attribuée à la variable d'environnement **JAVA_COMPILER**, le JIT reste désactivé. Pour désactiver la variable d'environnement, dans une invite shell, entrez `unset` **JAVA_COMPILER**.

- Utilisez l'option **-D** sur la ligne de commande JVM pour affecter à la propriété java.compiler la valeur "jitc". A l'invite shell, entrez : `java -Djava.compiler=jitc <MonApp>`
- Utilisez l'option **-Xjit** sur la ligne de commande de la JVM. Vous *ne devez pas* spécifier l'option **-Xint** simultanément. A l'invite shell, entrez : `java -Xjit <MonApp>`

Désactivation du JIT

Vous pouvez désactiver le JIT de différentes manières. Les options de ligne de commande remplacent la variable d'environnement **JAVA_COMPILER**.

Pourquoi et quand exécuter cette tâche

Procédure

- Attribuez la valeur "NONE" ou une chaîne vide à la variable d'environnement **JAVA_COMPILER** avant l'exécution de l'application Java. Dans une invite shell entrez :
  – **Pour l'interpréteur de commandes Korn :** `export JAVA_COMPILER=NONE`

    Remarque : Les commandes de l'interpréteur de commandes Korn sont utilisées dans le reste de cette section.
  – **Pour l'interpréteur de commandes Bourne :**

    ```
    JAVA_COMPILER=NONE
    export JAVA_COMPILER
    ```
  – **Pour l'interpréteur de commandes C :** `setenv JAVA_COMPILER NONE`
- Utilisez l'option **-D** sur la ligne de commande de la JVM afin d'attribuer la valeur "NONE" ou une chaîne vide à la propriété java.compiler. A l'invite shell, entrez: `java -Djava.compiler=NONE <MonApp>`
- Utilisez l'option **-Xint** sur la ligne de commande de la JVM. A l'invite shell, entrez : `java -Xint <MonApp>`

Déterminer si le JIT est activé

Vous pouvez déterminer l'état du JIT à l'aide de l'option **-version**.

Procédure

Entrez la commande suivante à l'invite shell :

```
java -version
```

Si le compilateur JIT est inactif, le message qui s'affiche contient ce qui suit :
`(JIT disabled)`
Si le compilateur JIT est actif, le message qui s'affiche contient ce qui suit :
`(JIT enabled)`

Utilisation d'unités d'exécution temps réel sans segments de mémoire (NHRT)

La récupération de place Metronome fournit des temps de réponse plus cohérents, mais il est parfois préférable d'éviter complètement les interruptions de la récupération de place.

Les unités d'extension NoHeapRealtimeThreads (NHRT) sont une extension des unités d'exécution RealtimeThreads. Elles diffèrent des unités d'exécution RealtimeThreads dans la mesure où elles n'accèdent pas au segment de mémoire. Sans accès au segment de mémoire, les unités d'exécution NHRT peuvent continuer de s'exécuter, même pendant un cycle de récupération de place avec certaines limites. Il s'en suit que sans accès au segment de mémoire, le modèle de programmation est différent du modèle de programmation des unités d'exécution temps réel.

Considérations relatives à l'utilisation des unités d'exécution NHRT

Tenez compte des points suivants sur les unités d'exécution NHRT :

- Vous utilisez principalement des unités d'exécution lorsqu'une tâche ne peut pas tolérer la récupération de place. Par exemple, l'application est liée au temps et ne peut pas tolérer les interruptions.
- Si le temps est si important que vous utilisez des unités d'exécution NHRT, envisagez également d'utiliser le compilateur AOT (ahead-of time), à savoir utilisez l'option **-Xnojit**.
- Lorsque vous utilisez l'option **-Xrealtime**, vous utilisez automatiquement récupérateur de place Metronome. Les avantages de récupérateur de place Metronome peuvent suffire à votre entreprise, ce qui évite d'avoir à coder des NHRT.
- Les unités d'exécution NHRT s'exécutent indépendamment du récupérateur de place, car leur priorité est supérieure à celle de ce dernier. Les unités d'exécution Java peuvent avoir une priorité comprise entre 1 et 10. Si des unités d'exécution NHRT sont présentes, la priorité des unités d'exécution Java est ramenée à 0, quelle que soit le priorité définie dans le programme. Le récupérateur de place est automatiquement configuré pour réduire de moitié une priorité plus élevée que l'unité d'exécution temps réel avec la priorité la plus haute. Vous définissez la priorité des unités d'exécution NHRT pour qu'elle soit au minimum supérieure à celle de l'unité d'exécution temps réel ayant la plus haute priorité. Ainsi, les unités d'exécution NHRT sont indépendantes du récupérateur de place.

  Remarque : Les unités d'exécution NHRT ne sont pas totalement ignorées par la récupération de place, car le récupérateur de place de l'unité d'exécution d'alarme Metronome s'exécute avec la priorité la plus haute dans le système. Cette priorité garantit que la machine JVM peut être activée pour déterminer si le récupérateur de place doit exécuter une opération. Le travail pour exécuter l'unité d'exécution d'alarme Metronome est réduit et n'affecte pas de manière sensible les performances. Sur un système multiprocesseur, l'unité d'exécution d'alarme peut être exécutée simultanément avec des unités d'exécution NHRT et dans ce cas aucune interruption provoquée par la récupération de place ne se produit.
- Comme les unités d'exécution NHRT sont limitées à la mémoire sectorisée et à la mémoire pérenne, des méthodes Java effectuent de vérifications pour garantir

qu'elles ne sont pas allouées depuis le segment de mémoire. La méthode de démarrage vérifie et retourne une exception (`MemoryAccessError`) si les unités d'exécution NHRT sont allouées depuis le segment de mémoire. Les unités d'exécution NHRT peuvent accéder uniquement à `ImmortalMemory` et `ScopedMemory`.

- La sémantique de verrouillage ne change pas. Par conséquent, les unités d'exécution peuvent être bloquées par les unités d'exécution normales si un verrou est partagé.
- Une unité d'exécution qui utilise le segment de mémoire peut voir sa priorité augmentée dans une méthode synchronisée lorsqu'une unité d'exécution NHRT tente d'utiliser la même méthode.
- Utilisez des files d'attente non bloquantes pour les communications entres les unités d'exécution NHRT et les unités d'exécution de segment de mémoire. Autrement, séparez les deux types d'unités d'exécution.

Exceptions

Ces exceptions peuvent se produire lorsque vous utilisez des unités d'exécution NHRT :

- `IllegalAssignmentError`. Par exemple, cette erreur peut se produire lors de la tentative de création d'une référence dans la mémoire sectorisée dans la mémoire pérenne.
- `MemoryAccessError`. Par exemple, cette erreur peut se produire lorsqu'une unité d'exécution NHRT tente de faire référence au segment de mémoire.

Contraintes de gestion des événements asynchrones

Les unités d'exécution NHRT peuvent être bloquée au cours de la récupération de place dans plusieurs cas.

1. Lorsqu'une unité d'exécution NHRT appelle fire(), setHandler(), ou addHandler() dans un événement asynchrone déjà associé à des gestionnaires alloués depuis le segment de mémoire.
2. Lorsqu'une unité d'exécution NHRT appelle destroy(), start(), ou stop() dans un minuteur déjà associé à des gestionnaires alloués depuis le segment de mémoire.
3. Lorsqu'une unité d'exécution NHRT est la dernière à quitter un secteur et ferme les minuteurs ou les événements asynchrones à partir de ce dernier. Cependant, des gestionnaires alloués à partir du segment de mémoire sont associés à ces composants.

Pour éviter que cela se produise avec les unités d'exécution NHRT :

1. Evitez d'ajouter des gestionnaires alloués depuis le segment de mémoire aux événements asynchrones ou aux minuteurs susceptibles d'être déclenchés par une unité d'exécution NHRT.
2. Evitez qu'une unité d'exécution NHRT quitte en dernier un secteur dans lequel des événements asynchrones ou des minuteurs ont des gestionnaires alloués à partir du segment de mémoire.

Contraintes de mémoire et de planification

La machine JVM empêche les unités d'exécution temps réel de charger des références à des objets qui se trouvent dans le segment de mémoire dans sa pile d'opérandes en générant une erreur javax.realtime.MemoryAccessError.

La machine JVM offre également une protection contre les références aux objets dans la mémoire sectorisée stockée dans le segment de mémoire ou la mémoire pérenne. Bien que la mémoire sectorisée ne soit pas utilisée exclusivement par les unités d'exécution, elle est plus susceptible d'être utilisée si la mémoire pérenne n'est pas appropriée et que la désallocation de mémoire est nécessaire dans un contexte NHRT.

Bien qu'une unité d'exécution NHRT soit exécutée, elle peut remplacer les références préexistantes à un objet dans le segment de mémoire dans une zone si elle la remplit avec une référence à un objet. La référence préexistante est remplacée par l'unité d'exécution NHRT sans générer d'erreur MemoryAccessError.

Contraintes de chargement des classes

Les classes sont chargées dans les mêmes zones de mémoire que le chargeur de classe. La valeur par défaut des chargeurs de classes est la mémoire pérenne.

Pour que les applications fournissent le temps de réponse attendu, elles doivent être "chauffées". Les applications doivent charger leurs classes au début pour que le chargement des classes n'interrompe pas les unités d'exécution temps réel ni les gestionnaires d'événements asynchrones plus tard.

Contraintes sur les unités d'exécution Java lors de l'exécution avec des NHRT

Comme les propriétés système sont partagées dans une machine JVM et que les unités d'exécution peuvent accéder à ces propriétés, utilisez avec précaution les méthodes getProperties et setProperties dans les machines JVM dans lesquelles des unités d'exécution NHRT sont exécutées. Pour que les propriétés système soient accessibles aux NHRT, elles doivent se trouver dans la mémoire pérenne.

La classe `java.lang.System` fournit des méthodes qui permettent aux unités d'exécution d'interagir avec les propriétés système, notamment :

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

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

La machine JVM temps réel utilise une instance de la classe com.ibm.realtime.ImmortalProperties créée spécifiquement pour l'objet JVM temps réel afin de stocker toutes les propriétés système. L'utilisation de cette instance permet de stocker la propriété dans la mémoire pérenne à la suite des appels à la méthode System.setProperty() ou System.getProperties.setProperty(). Aucun code utilisateur spécial n'est nécessaire dans ce cas, mais il est important de savoir que chaque fois qu'une propriété est définie de la mémoire pérenne est consommée.

Les appels à la méthode setProperties() sont un peu plus complexes, car l'objet Propriétés est utilisé pour stocker les propriétés système. Si une application est exécutée dans une machine JVM où sont exécutées des unités d'exécution NHRT, les appels à la méthode setProperties doivent envoyer une instance de classe ou de sous-classe com.ibm.realtime.ImmortalProperties créée dans la mémoire pérenne. Utilisez cette instance pour placer toutes les propriétés définies à l'aide de la méthode setProperties dans la mémoire pérenne.

Remarque : L'appel de setProperties(null) génère un nouvel objet ImmortalProperties en interne avec un groupe de propriétés par défaut qui consomme de la mémoire pérenne supplémentaire.

Les appels à la méthode getProperties() retournent l'objet défini ou l'objet Propriétés par défaut qui est un objet com.ibm.realtime.ImmortalProperties. Pour optimiser la compatibilité avec le code existant qui appelle la méthode getProperties(), l'objet ImmortalProperties sérialise l'objet, puis le désérialise dans une machine JVM standard. Le comportement par défaut de la sérialisation d'ImmortalProperties sérialise un objet Propriétés standard car les machines JVM standard ne disposent pas de l'objet ImmortalProperties et la désérialisation échoue. Pour remplacer ce comportement par défaut, la classe ImmortalProperties fournit la méthode enabledReplacement(boolean), laquelle, si elle est appelée avec false, désactive le comportement par défaut. Dans ce cas, la sérialisation sérialise l'objet ImmortalPropeties et il est alors possible de le désérialiser et d'utiliser l'objet résultant dans un appel à la méthode System.setProperties dans une machine JVM temps réel..

Remarque : La désérialisation est exécutée dans la mémoire pérenne et elle peut consommer une trop grande quantité de cette ressource limitée.

Gestionnaire de sécurité

Le gestionnaire de sécurité défini pour le système est utilisé par tous les types d'unités d'exécution dans la machine JVM. C'est la raison pour laquelle, dans une machine JVM temps réel où sont exécutées des unités d'exécution NHRT, le gestionnaire de sécurité doit être alloué dans la mémoire pérenne. La machine JVM temps réel permet de garantir qu'un gestionnaire de sécurité défini dans les options de lignes de commande est alloué dans la mémoire pérenne. Le gestionnaire de sécurité peut être également défini via des appels à la méthode System.setSecurityManager(SecurityManager). Si l'application définit le gestionnaire de sécurité de cette manière, elle doit vérifier que le gestionnaire de sécurité a été alloué depuis la mémoire pérenne pour que les unités d'exécution NHRT puissent s'exécuter correctement.

Les exceptions émises et les objets retournés par le gestionnaire de sécurité doivent se trouver dans la mémoire pérenne, si placés en mémoire cache, ou alloués dans le contexte d'allocation actuel.

Synchronisation

La classe MonitorControl et sa sous-classe PriorityInheritance gèrent la synchronisation, notamment le contrôle de l'inversion de priorité. Ces classes permettent de définir une règle de contrôle d'inversion de priorité comme valeur par défaut ou pour des objets donnés.

Les classes WaitFreeReadQueue, WaitFreeWriteQueue et WaitFreeDequeue permettent d'établir des connexions sans attente entre les objets planifiables (notamment les instances de NoHeapRealtimeThread) et les unités d'exécution standard Java.

Les classes WaitFree fournissent un accès simultané sécurisé aux données partagées entre les instances de NoHeapRealtimeThread et les objets planifiables qui sont affectés par des retards dans la récupération de place.

Sécurité des classes temps réel sans segment de mémoire

Dans certains cas, l'API ne peut pas être nécessairement utilisée dans un contexte sans segment de mémoire. Des restrictions sont appliquées aux classes partagées entre les unités d'exécution avec segment de mémoire et sans segment de mémoire. Déterminez les classes fournies avec la machine JVM qui peuvent être utilisées en toute sécurité.

Partage des objets

Les méthodes qui s'exécutent dans unités d'exécution en temps réel ne contenant pas de segments émettent une erreur javax.realtime.MemoryAccessError chaque fois qu'elles tentent de charger une référence à un objet dans un segment de mémoire.

La figure 3 est un exemple de type de code à éviter :

```
/**
 * NHRTError1
 *
 * Il s'agit d'un exemple simple d'accès par une unité d'exécution à
 * une référence d'objet dans un segment de mémoire.
 *
 * L'erreur générée est :
 *
 * 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; /* L'unité d'exécution NHRT peut accéder aux zones statiques directement ; elles sont toujours en mémoire p
    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( );
        }

    }
```

Figure 3. Exemple d'unité d'exécution NHRT accédant à une référence d'objet dans un segment de mémoire

```
    /* Classe d'unité d'exécution NHRT */
    class NHRT extends NoHeapRealtimeThread {
        public NHRT() {
            super(null, ImmortalMemory.instance());
        }

        /* Affiche la chaîne via la référence statique dans NHRTError1.message */
        public void run() {
            System.out.println("Message: " + NHRTError1.message);
        }
    }

}
```

Figure 4. Exemple d'unité d'exécution NHRT accédant à une référence d'objet dans un segment de mémoire (suite de la figure 1)

La figure 3, à la page 59 produit une erreur **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 un objet doit être accessible à une unité d'exécution temps réel sans segment de mémoire et une unité d'exécution Java standard, l'objet doit être alloué dans la mémoire pérenne. De même, si un objet doit être accessible à une unité d'exécution temps réel sans segment de mémoire et une unité d'exécution temps réel, l'objet peut être stocké dans une zone de mémoire sectorisée.

Dans la figure 3, à la page 59, la référence à la chaîne "This on the heap." est stockée dans une variable de classe. Cette variable est accessible aux unités d'exécution NHRT, car toutes les classes sont allouées dans la mémoire pérenne. De même, la chaîne aurait pu être envoyée au constructeur d'unités d'exécution NHRT.

La plupart des objets contiennent des références à d'autres objets. Par conséquent, prenez toutes les précautions nécessaires lorsque vous partagez ces objets entre des unités d'exécution ordinaires et des unités d'exécution NHRT. L'allocation d'une liste LinkedList en mémoire pérenne partagée entre une unité d'exécution ordinaire et une unité d'exécution NHRT est un exemple type. En cas d'erreur, l'unité d'exécution standard peut introduire des objets dans LinkedList, qui se trouvent dans le segment de mémoire. Plus grave serait que les structures de données allouées par LinkedList pour suivre les objets soient allouées dans le segment de mémoire par l'unité d'exécution ordinaire, ce qui provoquerait aisément une erreur MemoryAccessError dans l'unité d'exécution NHRT.

Certaines classes ne peuvent pas être partagées en toute sécurité entre les unités d'exécution NHRT et les autres unités d'exécution, quel que soit l'emplacement dans lequel leurs instances peuvent être allouées. Ces classes dépendent des objets stockés dans des variables de classe, généralement à des fins de mise en cache. InetAddress est un exemple type qui met en cache les adresses. Si la première unité d'exécution pour appeler certaines méthodes dans InetAddress s'exécute dans le segment de mémoire, l'appel par les unités d'exécution NHRT des mêmes méthodes dans le futur n'est pas sûr.

Verrouillage des objets avec des unités d'exécution NHRT

Les unités d'exécution NHRT doivent éviter de se synchroniser avec d'autres unités d'exécution. Tenez compte du scénario suivant :

- Une unité d'exécution temps réel entre dans un bloc ou une méthode synchronisés en se synchronisant dans un objet.
- Une unité d'exécution NHRT à haute priorité est bloquée lors d'une tentative de synchronisation dans le même objet.
- L'héritage de priorité amène l'unité d'exécution temps réel à avoir temporairement la même priorité que l'unité d'exécution NHRT.
- La récupération de place est alors exécutée avec une priorité supérieure que l'unité d'exécution NHRT et peut par conséquent interrompre cette dernière. L'utilisation de l'unité d'exécution NHRT a pour but d'éviter l'interruption par le processus de récupération de place, et c'est la raison pour laquelle ce scénario rejette l'utilisation de l'unité d'exécution NHRT.

Il est parfois impossible d'éviter que les unités d'exécution NHRT et les autres unités d'exécution se synchronisent dans un même objet, mais vous devez limiter les possibilités. Veillez à éviter une synchronisation inutile lors du partage des objets.

Restrictions sur les classes sûres

Vous devez tenir compte de certaines considérations lorsqu'une application contient des objets d'unité d'exécution temps réel et unité d'exécution temps réel sans segment de mémoire.

- L'unité d'exécution temps réel sans segment de mémoire peut être affectée par une erreur MemoryAccessErrors provoquée par l'interaction avec l'unité d'exécution temps réel.
- L'unité d'exécution temps réel sans segment de mémoire peut être retardée accidentellement par la récupération de place provoquée par l'unité d'exécution temps réel.

Erreurs MemoryAccessErrors provoquées dans une unité d'exécution temps réel sans segment de mémoire

Lorsque les deux types d'unités d'exécution appellent une même classe, l'unité d'exécution temps réel peut «polluer» les variables statiques de la classe avec des objets alloués depuis le segment de mémoire. L'unité d'exécution temps réel sans segment de mémoire reçoit une erreur MemoryAccessError lorsqu'elle tente d'accéder à ces objets du segment de mémoire. Cette pollution peut également se produire dans des instances de la classe. Malheureusement, les deux problèmes risquent vraisemblablement d'exister dans les modèles de codage standard et il est donc utile d'explorer quelques cas.

Si une classe exécute une longue opération, elle place généralement en cache le résultat pour améliorer les performances des opérations suivantes. Le cache est généralement une collection, telle qu'une mappe HashMap ancrée dans une variable statique dans la classe. Une unité d'exécution temps réel exécutée dans un contexte de segment de mémoire peut stocker un objet de segment de mémoire dans la collection, ce qui ajoute non seulement l'objet lui-même, mais également des objets d'infrastructure à la collection, telles que des parties d'index. Lorsque, ensuite, une unité d'exécution temps réel sans segment tente d'accéder à la collection sans tenter d'accéder à l'objet ajouté par l'autre unité d'exécution, elle tente de charger les objets d'infrastructure et elle reçoit donc un erreur MemoryAccessError. Lors du développement des bibliothèques et de leur optimisation, ces caches deviennent plus communs.

Une instance de classe peut être également polluée par des objets de segment de mémoire de différentes manières. Supposons une instance créée dans la mémoire

pérenne et donc accessible aux deux types d'unités d'exécution. Si l'objet est utilisé pour la première fois par une unité d'exécution temps réel dans un contexte de segment de mémoire, un objet secondaire peut être stocké dans une zone de l'objet d'origine. Si l'objet secondaire est une contexte de segment de mémoire, les utilisations suivantes par l'unité d'exécution temps réel sans segment de mémoire génèrent une erreur MemoryAccessError. Ces objets secondaires peuvent ne pas être toujours ajoutés à la première utilisation, mais après un certain nombre d'utilisations et peuvent servir à améliorer les performances des méthodes très utilisées.

Unité d'exécution NoHeap différée par la récupération de place

Vous devez affecter aux unités d'exécution NoHeap des priorités supérieures à celles des autres unités d'exécution afin qu'elles ne soient différées par la récupération de place.

En outre, si une classe contient des méthodes synchronisées, il se peut qu'une unité d'exécution temps réel sans segment de mémoire appelant ces méthodes diffère accidentellement la récupération de place. Ce scénario est décrit dans «Verrouillage des objets avec des unités d'exécution NHRT», à la page 60.

Si une classe contient des méthodes synchronisées (méthodes statiques ou d'instance), il peut arriver qu'une unité d'exécution temps réel sans segment appelant ces méthodes soit retardée par la récupération de place. Le problème apparaît si une unité d'exécution temps réel accède à une méthode synchronisée (statique ou d'instance) au moment où une unité d'exécution temps réel sans segment tente d'appeler une autre méthode synchronisée qui bloque l'attente de la fin de l'autre unité d'exécution. Si la priorité de l'unité d'exécution temps réel sans segment est supérieure à celle de l'unité d'exécution temps réel, la priorité de cette dernière est augmentée. Si cette unité d'exécution est ensuite forcée d'attendre l'interruption de la récupération de place, une inversion de priorité est possible, car l'unité d'exécution du récupérateur de place a une priorité supérieure à celle de l'unité d'exécution ayant la priorité la plus élevée qui peut ne pas être aussi élevée que l'unité d'exécution temps réel sans segment bloquée en attendant d'entrer dans la méthode synchronisée.

La seule solution pour résoudre ces problèmes consiste à faire en sorte que unités d'exécution en temps réel ne contenant pas de segments n'appelle jamais des méthodes synchronisées sur des classes ou des instances partagées avec d'autres types d'unités d'exécution. Malheureusement, il n'est jamais explicite depuis une signature de méthode de déterminer si une méthode est synchronisée ou non ; elle peut, par exemple, contenir un bloc synchronisé ou appeler une méthode synchronisée.

Résumé

La classe NoHeapRealtimeThread augmente sensiblement la complexité de l'environnement temps réel et des problèmes peuvent apparaître lorsqu'une combinaison de types d'unités d'exécution fonctionne dans un environnement. Au cours du développement d'une application, vous devez soigneusement concevoir les zones dans lesquelles différents types d'unités d'exécution partagent les classes. Attachez une attention particulière à l'utilisation des classes par ces unités d'exécution dans le kit SDK. Du fait de la complexité d'analyse, il n'est pas possible de garantir que toutes les classes fournies dans le kit SDK peuvent être partagées sans problème. Un petit sous-groupe de classes a été vérifié uniquement. A l'origine, la vérification s'est limitée à l'aspect MemoryAccessError. Le résultat est

une liste de classes ayant été analysées et modifiées le cas échéant pour qu'elles puissent être utilisées par les unités d'exécution sans segment et les autres types d'unités d'exécution.

Classes sûres

Cette section liste les groupes de classes qui peuvent être utilisées en toute sécurité par l'unité d'exécution NoHeapRealtimeThread et les autres types d'unités d'exécution.

Cette section porte principalement sur l'aspect MemoryAccessError de la sécurité. La liste suivante détaille les classes qui peuvent être utilisées par les trois types d'unités d'exécution dans une même JVM.

Remarque : Des instances individuelles de la classe peuvent ne pas être toujours partagées en toute sécurité.

Suivez les règles ci-dessous pour qu'une classe puisse être utilisée en toute sécurité par tous les types d'unité d'exécution :

- L'instance doit être créée dans une zone de mémoire accessible à l'unité d'exécution qui veut accéder à l'instance.
- Si la classe contient des zones statiques publiques, évitez de stocker des objets de segment de mémoire dans ces zones.
- Si la classe contient des zones d'instance publiques, évitez de stocker des objets de segment de mémoire dans ces zones.

Les classes fournies par IBM ne sont pas toutes des classes utilisables dans les unités d'exécution NHRT. Les packages suivants contiennent des classes utilisable dans les unités d'exécution NHRT :

- java.lang package
- java.lang.reflect package
- java.lang.ref package (toutes les classes)
- java.net package
- java.io package
- java.math package

Ces tableaux montrent des classes qui ne sont pas des classes utilisables dans les unités d'exécution NHRT :

Tableau 5. Classes dans le package java.lang qui ne sont pas des classes utilisables dans les unités d'exécution NHRT

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

Tableau 6. Classes dans le package java.lang qui ne sont pas des classes utilisables dans les unités d'exécution NHRT

Classe	Méthode
java.lang.reflect.Proxy.*	*

Tableau 7. Classes dans le package java.net qui ne sont pas des classes utilisables dans les unités d'exécution NHRT.

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

Tableau 8. Classes dans le package java.io qui ne sont pas des classes utilisables dans les unités d'exécution NHRT

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

Tableau 9. Classes dans le package java.math qui ne sont pas des classes utilisables dans les unités d'exécution NHRT

Classe	Méthode
java.math.BigInteger	*

Les packages peuvent inclure des sous-package contenant des classes non sûres. Par exemple, les classes suivantes ne sont pas utilisables dans les unités d'exécution NHRT :
- java.lang.management.*
- java.lang.annotation.*
- java.lang.instrument.*

Même si une classe est considérée comme étant utilisable dans les unités d'exécution NHRT, il se peut qu'elle ne le soit pas. Les développeurs d'applications doivent déterminer les conditions temps réel des classes que vous utilisez cas par cas, indépendamment du fait qu'une classe est utilisable dans une unité d'exécution NHRT ou non.

Partage de données de classes entre JVM

La machine virtuelle Java (JVM) permet de partager des données de classes entre les machines JVM en les stockant dans un fichier cache mappé en mémoire sur disque.

Le partage des classes réduit la consommation globale de mémoire virtuelle lorsque plusieurs machines virtuelles partagent un cache. Il diminue également le temps de démarrage d'une machine virtuelle après création du cache. Le cache de classes partagées est indépendant de toute machine virtuelle active et persiste jusqu'à ce qu'il soit détruit. Un cache partagé peut contenir les éléments suivants :
- les classes d'amorce
- les classes d'application
- les métadonnées qui décrivent les classes
- le code compilé AOT (Ahead-of-time)

Les caches de classes partagées peuvent être utilisés par IBM WebSphere Real Time for RT Linux en mode non-temps réel et en mode temps réel, mais les techniques de formatage, de création et de remplissage de cache diffèrent. Les caches en mode temps réel ne sont pas compatibles avec les caches en mode non-temps réel. En mode non-temps réel, les caches sont créés et remplis de la même manière que la machine JVM standard. Cela implique que le cache est créé et rempli par le machine JVM lorsqu'elle exécute une application de manière transparente pour l'utilisateur. En mode temps réel, en utilisant l'option **-Xrealtime**, les caches de classes partagées doivent être créés et préremplis par **admincache** en utilisant l'option **-populate**. Les applications exécutées en mode temps réel peuvent lire le contenu du cache prérempli, mais elles ne peuvent pas le modifier.

Utilisez l'outil **admincache** pour créer, remplir et détruire les caches.

Pour permettre à une application d'utiliser un cache de classes partagées, ajoutez l'option **-Xshareclasses** à sa ligne de commande. Comme les caches en mode temps réel sont en lecture seule, des sous-options du mode non temps réel de **-Xshareclasses** ne sont pas disponibles en mode temps réel.

Pour plus d'informations, voir «Utilisation de l'outil admincache», à la page 39, «Partage des données de classes entre les machines JVM en mode non-temps réel», à la page 95 et «Diagnostics de classes partagées», à la page 144.

Exécution des applications avec un cache de classes partagées

Pour exécuter une application avec un cache de classes partagées, utilisez l'option **-Xshareclasses** sur la ligne de commande.

Le tableau 10 montre les sous-options disponibles lors de l'exécution d'une application en mode temps réel en utilisant l'option **-Xshareclasses**.

Tableau 10. Sous-options disponibles lors de l'exécution d'une application en mode temps réel

Option	Signification
cacheDir=<*répertoire*>	Définit le répertoire dans lequel les données de cache sont lues et écrites. Par défaut, <*directory*> est /tmp/javasharedresources. Le nom du répertoire doit correspondre à celui défini dans l'option **-cacheDir** utilisée dans la commande admincache pour créer le cache.
name=<*name*>	Nom du cache de classes partagées à utiliser. Le nom du répertoire doit correspondre à celui défini dans l'option **-cacheName** utilisée dans la commande admincache pour créer le cache. Le nom ne doit pas dépasser 53 caractères.
none	Désactive le partage de classes. Peut être ajoutée à la fin d'une ligne de commande pour désactiver le partage de données de classes. Cette sous-option remplace les arguments de partage qui la précèdent sur la ligne de commande.

Tableau 10. Sous-options disponibles lors de l'exécution d'une application en mode temps réel (suite)

Option	Signification
`nonfatal`	Toujours démarrer la machine JVM indépendamment des erreurs/ avertissements. Permet le démarrage de la machine virtuelle JVM même si le partage de données de classes échoue. Généralement, la machine virtuelle ne démarre pas si le partage de données de classes échoue. Si vous sélectionnez **`nonfatal`** et que l'initialisation du cache des classes partagées échoue, la machine virtuelle Java tente de se connecter au cache en lecture seule. Si la tentative échoue, la machine virtuelle Java démarre sans le partages des données de classes.
`silent`	Supprime tous les messages de sortie. Désactive tous les messages relatifs aux partages de classes, y compris les messages d'erreurs. Les messages des erreurs irrémédiables qui empêchent l'initialisation de la machine virtuelle Java s'affichent.
`verbose`	Active la sortie prolixe qui fournit des informations sur l'état global du cache de classes partagées ainsi que des messages d'erreur détaillés.
`verboseAOT`	Active la sortie prolixe lorsque le code compilé AOT est trouvé dans le cache, par exemple, lors des demandes de chargement des méthodes AOT.
`verboseHelper`	Permet des sorties détaillées pour l'API Java Helper. Cette sortie montre comment l'API Helper est utilisée par le chargeur de classe.
`verboseIO`	Active la sortie prolixe pour les demandes de chargement de classes. Cette option fournit une sortie détaillée sur l'activité E/S de cache en listant les informations sur les classes trouvées.

Pour vérifier que ces options sont correctes, utilisez l'option **`-printvmargs`** avec admincache (pour plus d'informations, voir **`-printvmargs`**). L'option **`nonfatal`** n'est pas destinée à une utilisation générale, car elle force la machine JVM à ignorer les erreurs et les avertissements sur le cache de classes partagées. L'option **`none`** désactive le partage de classes et revient à omettre l'option **`-Xshareclasses`** sur la ligne de commande.

Pour plus d'informations sur les sous-options **`-Xshareclasses`**, voir Options de ligne de commande de partage des données de classes.

Utilisation du récupérateur de place Metronome

Le récupérateur de place Metronome remplace le récupérateur de place standard dans WebSphere Real Time for RT Linux.

Contrôle de l'utilisation du processeur

Vous pouvez limiter la quantité de puissance de traitement disponible pour le récupérateur de place Metronome.

Vous pouvez contrôler la récupération de place avec le récupérateur de place Metronome en utilisant l'option **-Xgc:targetUtilization=N** pour limiter la quantité d'UC utilisée par le récupérateur de place.

Par exemple :

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

Cet exemple spécifie que votre application utilise 80 % de puissance de traitement sur 60 millisecondes. Les 20 % du temps restant sont utilisés pour la récupération de place. Le récupérateur de place Metronome garantit des niveaux d'utilisation dès lors qu'il dispose des ressources suffisantes. La récupération de place commence lorsque la quantité d'espace libre dans le segment de mémoire tombe en dessous d'un seuil défini dynamiquement.

Optimisation de récupérateur de place Metronome

Vous pouvez optimiser l'environnement temps réel en contrôlant la quantité de mémoire qu'utilise l'application. Par exemple, utilisez les options **-Xmx**, **-Xgc:immortalMemorySize=size**, **-Xgc:scopedMemoryMaximumSize=size** et **-Xgc:targetUtilization=N**s.

- Utilisez l'option **-Xmx** pour limiter la taille du segment de mémoire.

  La valeur choisie est utilisée comme limite maximale et reflète ainsi l'utilisation vraisemblable dans le temps. L'utilisation d'une valeur trop basse augmente la fréquence de la récupération de place et génère un débit général plus faible, mais réduit l'impact sur la mémoire. Pour obtenir de bonnes performances temps réel, évitez la pagination. Il est normal de déterminer si l'impact de tous les processus actifs sur une machine ne dépasse pas la taille de mémoire physique.
- Utilisez l'option **-Xgc:immortalMemorySize=size** pour contrôler la taille de la zone de mémoire pérenne.

  Vous devez analyser soigneusement l'utilisation de la mémoire pérenne. Une application «idéale» utilise la mémoire pérenne au cours du démarre uniquement. Si l'allocation d'objets pérennes continue, l'application peut continuer de s'exécuter jusqu'à ce que la mémoire pérenne soit épuisée. L'utilisation actuelle peut être obtenue en ajoutant :

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

  au code.
- Utilisez l'option **-Xgc:scopedMemoryMaximumSize=size** pour que les applications ne demandent pas trop de mémoire sectorisée. Utilisez cette option pour les diagnostics et non pas pour l'optimisation.
- Définissez l'option **-Xgc:targetUtilization=N** pour que dans le pire des cas (fréquence d'allocation maximale des objets de segment de mémoire), le collecteur de récupération de place puisse récupérer de la place plus fréquemment que l'application n'en génère.

  Généralement, la valeur par défaut est suffisante, mais vous pouvez améliorer les performances de l'application en augmentant l'utilisation jusqu'au point à partir duquel le récupérateur peut récupérer de la place légèrement plus rapidement que l'application ne peut en créer.

- Utilisez l'option **-Xgcthreads <n>** pour créer des unités d'exécution supplémentaires pour exécuter parallèlement la récupération de place.

  Une seule unité d'exécution est utilisée par défaut. Si la fréquence d'utilisation de la mémoire de la charge de travail est élevée et que la charge de travail s'exécute dans un multiprocesseur symétrique avec des cycles UC disponibles, affectez à ce paramètre la valeur >1 pour améliorer les performances.

  Remarque : L'affectation d'une valeur trop élevée à ce paramètre peut avoir un impact négatif.

Limitation du récupérateur de place Metronome

Dans certaines circonstances, les interruptions pour récupération de place peuvent être plus longues que prévu.

Au cours de la récupération de place, un processus d'analyse racine est utilisé. Le récupérateur de place parcourt le segment de mémoire en commençant à partir des références actives connues. Ces références incluent :

- Variables de références actives dans les piles d'unité d'exécution actives
- Références statiques
- Toutes les références aux objets dans les mémoires pérennes et sectorisée.

Pour rechercher toutes les références aux objets actifs dans la pile d'une unité d'exécution d'une application, le récupérateur de place parcourt tous les cadres de la pile des appels de l'unité d'exécution. Chaque pile d'unité d'exécution active est analysée dans une étape ininterruptible. Cette analyse doit donc avoir lieu au cours d'une interruption individuelle du récupérateur de place.

Dans ce cas, les performances système peuvent être pires que prévu s'il existe des unités d'exécution très imbriquées dans les piles du fait des longues pauses de récupération de place au début d'un cycle de récupération de place.

La mémoire pérenne est traitée de manière incrémentielle. Toutes les autres zones de mémoire sectorisée sont traitées dans une seule étape ininterruptible atomique. Par conséquent, une utilisation significative des zones de mémoire sectorisée peut dégrader les performances du système plus que prévu du fait des longues pauses de la récupération de la place lorsque l'analyse racine traite la mémoire sectorisée.

Chapitre 6. Développement d'applications

Informations importantes sur l'écriture d'applications en temps réel, y compris les échantillons de code.

Ecriture d'applications Java pour tirer parti de l'environnement temps réel

Ces exemples expliquent comment exploiter l'environnement temps réel. Ces exemples couvrent aussi bien un exemple simple qui exécute une application Java en temps réel sans modifier le code, qu'un processus plus complexe de planification et d'écriture unités d'exécution en temps réel ne contenant pas de segments. Les arguments en faveur de l'approche la mieux adaptée à l'application sont fournis.

Présentation de l'écriture d'applications temps réel

Il est inutile d'écrire des applications élaborées NHRT (no-heap real-time) pour tirer parti des fonctions de la technologie temps réel. Certains avantages peuvent être exploités en modifiant légèrement le code existant.

Si vous êtes programmeur d'application, procédez comme suit pour exploiter WebSphere Real Time for RT Linux :

1. Vous pouvez exécuter une application standard Java dans une machine JVM temps réel pour tirer parti de la récupération de place Metronome et améliorer de manière significative la prévisibilité de l'exécution de l'application.
2. Ajoutez l'option **-Xnojit** après avoir précompilé le code pour utiliser le compilateur AOT (ahead-of-time). Voir «Stockage des fichiers JAR précompilés dans un cache de classes partagées», à la page 49.
3. Remplacez java.lang.Thread par javax.realtime.RealtimeThread dans l'application. Vous pouvez constater une légère amélioration comparé à l'option AOT.

   unités d'exécution temps réel présente principalement l'avantage de pouvoir contrôler la priorité que vous affectez à chaque unité d'exécution. Vous pouvez également rendre les unités d'exécution temps réel périodiques. Pour exploiter ces avantages, vous devez être préparé à apporter des modifications à l'application elle-même.
4. Planifiez et écrivez une application donnée pour utiliser des unités d'exécution temps réel et des gestionnaires d'événements asynchrones pour gérer les minuteurs ou les événements externes. Tenez compte des points suivants :
   - Planifiez la priorité que vous affectez à votre unités d'exécution temps réel.
   - Déterminez les zones de mémoire de stockage des objets.
   - Communiquez avec les gestionnaires d'événements.
5. Planifiez et écrivez une application donnée pour utiliser unités d'exécution en temps réel ne contenant pas de segments. Les unités d'extension NHRT sont des extensions des unités d'exécution temps réel et vous devez tenir compte de

la priorité que vous affectez et de la zone de mémoire. En règle générale, exécutez cette étape uniquement si l'application doit gérer les événements dans des délais comparables à la pause de la récupération de place (sous-milliseconde). Ne sous-estimez pas la complexité du développement avec les unités d'exécution en temps réel ne contenant pas de segments.

figure 5 Etapes décrites précédemment

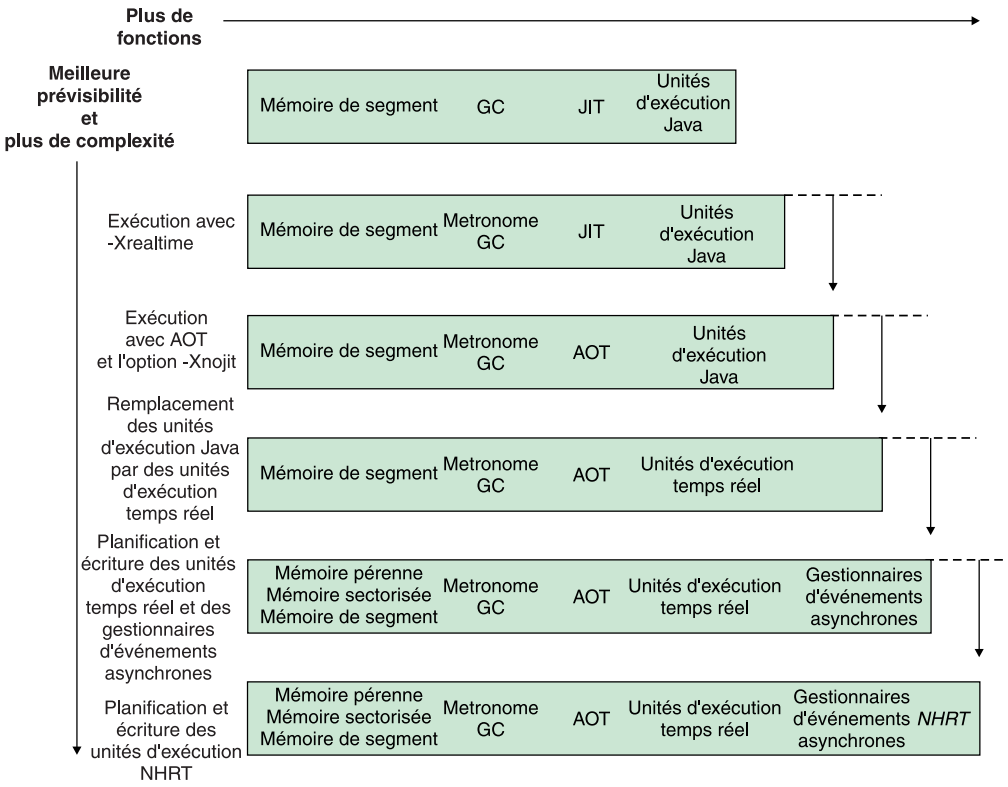

Figure 5. Comparaison des fonctions de RTSJ et de l'amélioration de la prévisibilité.

Planification de l'application WebSphere Real Time for RT Linux

Lorsque vous vous apprêtez à écrire des applications Java temps réel, vous devez déterminer si vous voulez utiliser des unités d'exécution Java, unités d'exécution temps réel ou unités d'exécution en temps réel ne contenant pas de segments. En outre, vous pouvez déterminer la zone de mémoire que vont utiliser les unités d'exécution.

Pourquoi et quand exécuter cette tâche

Lors de la planification de l'application, suivez ces étapes relatives aux décisions à prendre :

Procédure

1. Identifiez les tâches.
2. Déterminez les périodes de timing :
   - Réponses supérieures à 10 ms : choisissez des unités d'exécution Java en exploitant simplement le récupérateur de place Metronome.

Ces unités d'exécution utilisent uniquement la mémoire de segment pour le stockage. Elles présentent un inconvénient dans la mesure où la récupération de place interrompt l'application, mais comme elle est contrôlée par récupérateur de place Metronome, la durée et le timing des interruptions sont prévisibles.

- Réponses inférieures à 10 ms : choisissez des unités d'exécution temps réel.

  Les unités d'exécution temps réel peuvent être placées dans le segment de mémoire, la mémoire sectorisée ou la mémoire pérenne. Avantages de l'utilisation de unités d'exécution temps réel :
  - Elles peuvent s'exécuter avec une priorité plus élevée que les unités d'exécution Java standard.
  - La récupération de place est contrôlée par le récupérateur de place Metronome. Toutefois, le récupérateur de place s'exécute avec une priorité plus élevée que la priorité plus haute d'une unité d'exécution temps réel et il interrompt l'exécution du programme.
- Réponses inférieures à une milliseconde : choisissez des unités d'exécution en temps réel ne contenant pas de segments.

  La priorité de unités d'exécution en temps réel ne contenant pas de segments peut être supérieure à celle de la récupération pour ne pas être interrompue de manière significative par le Metronome. Seule l'unité d'exécution d'alarme Metronome s'exécute avec la priorité la plus élevée et utilise très peu l'UC.

3. Déterminez si l'application nécessite des gestionnaires d'événements asynchrones. Cela dépend de la structure du programme.
   - Temps de réponse inférieure à 10 ms : choisissez des unités d'exécution temps réel.
   - Temps de réponse inférieur à une milliseconde : choisissez des unités d'exécution en temps réel ne contenant pas de segments
4. Déterminez les priorités des unités d'exécution. En règle générale, plus la période est courte, plus la priorité est élevée.
5. Définissez les caractéristiques de mémoire.
   - Si une tâche a une fréquence d'allocation variable ou élevée, ce qui peut submerger le récupérateur de place, imposez une limite de fréquence (en utilisant MemoryParameters) ou effectuez l'allocation dans une zone de mémoire sectorisée.
   - Si une tâche génère une grande quantité de données temporaires au cours d'un calcul, utilisez une zone de mémoire sectorisée.
   - Si une tâche génère une certaine quantité de données au cours du démarrage nécessaire pour la durée de vie de la machine JVM, utilisez une mémoire pérenne. Evitez d'utiliser cette mémoire si des objets vont continuer d'être créés pendant la vie de la machine JVM.
   - Si les tâches doivent communiquer, notamment si l'une d'entre elles s'exécute sous une unité d'exécution NHRT (no-heap real-time thread), utilisez une zone de mémoire sectorisée pour la communication.
   - Si une tâche s'exécute sous une unité d'exécution NHRT, créez une zone de mémoire sectorisée, par exemple LTMemory, pour y placer l'unité d'exécution sans segment, les paramètres d'exécution et éventuellement les files d'attente sans attente utilisée pour communiquer avec la tâche. L'objet LTMemory doit être créé dans la mémoire pérenne ou un autre secteur pour éviter les erreurs lorsque l'unité d'exécution sans segment tente d'y faire référence.
6. Modifiez les options d'exécution pour améliorer les performances de l'application lorsque vous avez déterminé la structure et le contenu de l'application. Les étapes suivantes expliquent comment procéder :

a. Au cours du test initial de l'application, définissez suffisamment d'espace dans le segment, la mémoire sectorisée et la mémoire pérenne en utilisant les options **-Xmx**, **-Xgc:immortalMemorySize=size** et **-Xgc:scopedMemoryMaximumSize=size**.

   Remarque : Avec le récupérateur de place Metronome, les tailles de segment initiale et maximum doivent être identiques, car le récupérateur de place n'augmente pas la taille du segment. L'augmentation du segment est une opération non déterministe.

b. Utilisez l'option **-verbose:gc** pour déterminer la quantité de mémoire utilisée.

c. Modifiez l'option **-Xgc:targetUtilization** pour donner suffisamment de temps à la récupération de place. La valeur par défaut est 70 % et ce pourcentage est généralement approprié pour la plupart des applications. Veillez à ce que la fréquence de la récupération de place soit légèrement plus élevée que la fréquence d'allocation.

d. Définissez une taille réaliste pour le segment de mémoire en utilisant l'option **-Xmx**.

Modification des applications Java

Pour écrire du code qui utilise les fonctions Java temps réel, utilisez `javax.realtime.RealtimeThread` pour remplacer `java.lang.Thread` pour les unités d'exécution.

Avant de commencer

Cet exemple repose sur la classe JavaRadar.java qui se trouve dans le fichier `demo/realtime/sample_application.zip`.

Pourquoi et quand exécuter cette tâche

Le modèle de programmation des unités d'exécution temps réel est similaire à celui des applications Java standard. Toutefois, cette méthode assez brutale d'ajouter des unités d'exécution temps réel aux programmes ne tire pas complètement parti des fonctions de WebSphere Real Time for RT Linux. Pour ce faire, vous devez modifier les unités d'exécution pour qu'elles aient une priorité et déterminer les zones de mémoire qu'elles vont utiliser.

En changeant simplement les classes des unités d'exécution, vous obtenez un léger avantage seulement pour l'application, car la priorité par défaut des unités d'exécution temps réel est supérieure à celle des unités d'exécution standard Java.

Pour remplacer JavaRadar par une unité d'exécution RealtimeThread, vous remplacez la classe qu'elle étend `Thread` par `RealtimeThread`.

Remplacement de `java.lang.Thread` par `javax.realtime.RealtimeThread`

La classe JavaRadar dans l'exemple d'application étend `java.lang.Thread`. Par exemple :

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

Pour convertir cette unité d'exécution Java en unité d'exécution temps réel, vous redéfinissez cette définition de classe comme suit :

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

Ecriture des unités d'exécution RTT (real-time thread)

Jusqu'à maintenant, vous avez modifié une application. Maintenant, vous allez écrire du code. Vous pouvez écrire des applications qui utilisent unités d'exécution temps réel pour tirer parti des niveaux de priorité temps réel et des zones de mémoire.

Avant de commencer

Cet exemple repose sur les classes RavaRadar.java, RTJavaRadar.java et RTJavaControlLauncher.java classes qui se trouvent dans le fichier `demo/realtime/sample_application.zip`.

Cet exemple montre comment utiliser la mémoire pérenne avec l'exemple décrit dans «Modification des applications Java», à la page 72.

Pourquoi et quand exécuter cette tâche

Le modèle de programmation pour unités d'exécution temps réel est similaire à celui des applications Java standard.

Avantages de l'utilisation des unités d'exécution temps réel sont les suivants :

- Support complet des priorités des unités d'exécution au niveau SE dans les unités d'exécution temps réel.
- Utilisation de la mémoire sectorisée et de la mémoire pérenne.
  - Avec la mémoire sectorisée, vous pouvez contrôler explicitement la mémoire désallocation sans affecter la récupération de place.
  - Avec des unités d'exécution en temps réel ne contenant pas de segments, vous pouvez utiliser la mémoire pérenne pour éviter à la récupération de place de s'interrompre.
  - Ces unités d'exécution temps réel, qui font référence aux objets dans le segment de mémoire, sont soumises à la récupération de place, comme les unités d'exécution temps réel stockées dans la mémoire du segment.
  - Les unités d'exécution NHRT (No-heap real-time threads) ne peuvent pas faire référence aux objets dans la mémoire du segment. Par conséquent, ils ne sont pas soumis à la récupération de place.

Dans le tableau 11, à la page 74, les priorités sont affectées en supposant que l'unité d'exécution SimulationThread a la priorité la plus élevée, car elle représente des événements externes et elles ne doivent pas être autorisées à être restaurées par un quelconque élément du programme. L'unité d'exécution RadarThread doit répondre rapidement aux commandes Ping du contrôleur. Plus la réponse est rapide, plus la mesure de la hauteur du module lunaire est précise. L'unité d'exécution ListenThread doit répondre également rapidement aux commandes du contrôleur, mais après l'unité d'exécution RadarThread.

Ces trois unités d'exécution se trouvent dans la zone de mémoire sectorisée, car la simulation s'exécute sous la forme d'un serveur. Après avoir exécuté la simulation le serveur peut quitter la mémoire sectorisée et y revenir pour attendre une autre exécution de la simulation. Le serveur utilise la mémoire sectorisée et peut par conséquent se réinitialiser.

L'unité d'exécution RTJavaRadarthread a la priorité la plus élevée des unités d'exécution de contrôleur, car elle est plus sensible au timing étant donné qu'elle

utilise ce temps pour déterminer la hauteur. Elle reste en mémoire pérenne, car elle s'exécute comme unité d'exécution NHRT et le contrôleur est exécuté une seule fois et la mémoire est libérée lorsque la machine JVM quitte.

Pour RTJavaControlThread et RTJavaEventThread, les contraintes de temps ne sont pas aussi importantes et, par conséquent, l'utilisation de la mémoire du segment est acceptable.

RTLoadThread n'exécute aucune fonction utile pour le module lunaire. Cependant, RTLoadThread montre que l'allocation et la désallocation de mémoire significative peuvent être exécutées avec une priorité plus basse que celle des autres unités d'exécution et n'affectent pas les performances des unités d'exécution ayant des priorités plus élevées.

Tableau 11. Relation des unités d'exécution avec les zones de mémoire dans l'exemple d'application

Mémoire	Unité d'exécution	Priorité
Mémoire sectorisée	demo.sim.SimulationThread	38
	demo.sim.RadarThread	37
	demo.sim.SimulationThread.ListenThread	36
Mémoire pérenne	demo.controller.RTJavaRadarThread	15
Segment de mémoire	demo.controller.RTJavaControlThread	14
	demo.controller.RTJavaEventThread	13
Mémoire sectorisée et segment de mémoire	demo.controller.RTLoadThread	12

Exemples

Ce code de demo.sim.SimulationThread montre où la priorité 38 est définie. **1** Cette ligne de code extrait la priorité maximale disponible dans la machine JVM.

```
super(null, area);

// Définir la priorité séparément, car nous utilisons "this".
// Noter que PriorityScheduler.MAX_PRIORITY est obsolète.
this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
        .getMaxPriority(this)));     1
```

Ce code de demo.sim.SimLauncher montre où la mémoire sectorisée est définie. **2** montre l'allocation de LTMemory, à savoir une zone de mémoire sectorisée qui alloue de la mémoire en temps linéaire.

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

/*
 * L'objet LTMemory doit être créé dans une zone de mémoire à
laquelle les unités d'exécution
 * NHRT peuvent accéder.
 */
ImmortalMemory.instance().enter(new Runnable() {
    public void run() {
        myMemRef.ref = new LTMemory(10000000);  2
    }
```

```
});

final MemoryArea simMemArea = myMemRef.ref;
```

L'objet ScopedMemoryArea référencé par `simMemArea` est alloué dans la mémoire pérenne, car l'unité d'exécution NHRT doit pouvoir référencer l'objet qui représente ScopedMemoryArea. Son allocation dans le segment de mémoire amène le constructeur NHRT à émettre une exception IllegalArgumentException, car son argument de zone de mémoire se trouvait dans le segment de mémoire.

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

Ce code de demo.controller.RTJavaControlLauncher montre où la mémoire pérenne est définie et utilisée par RTJavaRadar. Comme RTJavaRadar s'exécute pendant toute la durée de vie de la machine JVM du contrôleur, elle alloue de la mémoire uniquement au démarrage ; elle peut être exécutée en toute sécurité en mémoire pérenne. La conception de l'application tire un avantage, car le contrôleur peut accéder aux méthodes RTJavaRadar sans avoir préalablement à entrer dans la mémoire sectorisée. L'entrée dans cette zone de mémoire est difficile, car le contrôleur est écrit pour s'exécuter dans l'environnement Java normal et l'environnement Java temps réel.

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

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

// Créer RTJavaRadar dans la mémoire pérenne ; il s'agit d'une
   unité d'exécution NHRT.
// Si elle se trouve dans la mémoire sectorisée, son interaction
   avec les autres unités d'exécution serait plus
// complexe.
ImmortalMemory.instance().enter(new Runnable() {
    public void run() {
        // Realtime version of Radar.
        radarRef.ref = new RTJavaRadar(radarPort, ImmortalMemory
                .instance());
    }
});

RTJavaRadar radarJava = radarRef.ref;
```

Ecriture de gestionnaires d'événements asynchrones

Les gestionnaires d'événements asynchrones réagissent à des événements d'horloge ou des événements qui se produisent en dehors d'une unité d'exécution ; par exemple, entrée depuis une interface d'une application. Dans les systèmes temps réel, ces événements doivent répondre dans les délais que vous définissez pour l'application.

Avant de commencer

Cet exemple repose sur les classes RTJavaEventThread.java et RTJavaControlLauncher.java qui se trouvent dans le fichier `demo/realtime/sample_application.zip`.

Pourquoi et quand exécuter cette tâche

Dans l'exemple d'application, l'unité d'exécution d'événements s'interrompt sur les événements de la simulation qui signalent un blocage ou un ciblage. Dans la

version temps réel de cette unité d'exécution, le mécanisme AsyncEvent est utilisé. Ces événements sont utilisés pour afficher le message d'état approprié et provoquer la sortie du contrôleur.

L'unité d'exécution RTJavaEventThread a deux événements asynchrones définis. qui n'ont aucun paramètre.

```
public class RTJavaEventThread extends RealtimeThread {

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

Ces événements créent et enregistrent deux gestionnaires d'événements asynchrones:

```
    /**
     * Envoi d'un objet exécutable qui sera déclenché lors de
     * l'événement d'alunissage.
     * Code exécutable @param à exécuter lors de l'alunissage.
     */
    public void addLandHandler(Runnable runnable) {
        AsyncEventHandler handler = new AsyncEventHandler(runnable);
        this.landEvent.addHandler(handler);
    }

    /**
     * Envoi d'un objet exécutable qui sera déclenché lors de l'alunissage.
     *
     * Code exécutable @param à exécuter lors du blocage.
     */
    public void addCrashHandler(Runnable runnable) {
        AsyncEventHandler handler = new AsyncEventHandler(runnable);
        this.crashEvent.addHandler(handler);
    }
```

Lors de la réception du message de crash ou d'alunissage, le gestionnaire d'événements asynchrones correspondant se déclenche, ce qui provoque la libération des objets exécutables.

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

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

Résultats

RTJavaControlLauncher.java contient les appels aux méthodes addLandHandler et addCrashHandler. Les objets exécutables envoyés provoquent l'affichage d'un message sur la console et l'unité d'exécution de contrôle s'arrête lorsque leur gestionnaires d'événements asynchrones associé se déclenche. Voir RTJavaEventThread.java pour identifier leur point de déclenchement.

```
            // AEH exécutable pour le gestionnaire d'alunissage.
            javaEventThread.addLandHandler(new Runnable() {
                public void run() {
```

```
            System.out.println("LAND!");
        }
    });

    // AEH exécutable pour le gestionnaire de crash.
    javaEventThread.addCrashHandler(new Runnable() {
        public void run() {
            System.out.println("CRASH!");
        }
    });
```

Ecriture des unités d'exécution NHRT

Pour ajouter des unités d'exécution unités d'exécution en temps réel ne contenant pas de segments (NHRT) à une application Java, utilisez ce tutoriel pour développer et modifier vos propres programmes.

Avant de commencer

Cet exemple repose sur les classes SimulationThread.java et SimLauncher.java qui se trouvent dans le fichier `demo/realtime/sample_application.zip`.

Pourquoi et quand exécuter cette tâche

La classe demo.sim.SimulationThread fait partie de la simulation dans l'application de démonstration. Elle ne s'inscrit pas dans le monde réel et, par conséquent, elle s'exécute sans interruption par rapport au reste du système. L'unité d'exécution est créée sous la forme d'une unité d'exécution NoHeapRealtimeThread avec la priorité la plus élevée pour que l'unité d'exécution ne soit pas interrompue par la récupération de place ou d'autres unités d'exécution sur le système.

Dans SimulationThread, le constructeur suivant appelle le super constructeur «NoHeapRealtimeThread(SchedulingParameters scheduling, MemoryArea area)» avant de définir ses paramètres SchedulingParameters et ReleaseParameters séparément :

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

        super(null, area);

        // Définition de la priorité séparément, car nous utilisons "this".
        // Notez que PriorityScheduler.MAX_PRIORITY est obsolète.
        this.setSchedulingParameters(new PriorityParameters(PriorityScheduler
                .getMaxPriority(this)));

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

        // Il est recommandé d'identifier chacune des unités d'exécution.
        this.setName("SimulationThread");

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

Les autres unités d'exécution actives dans la simulation sont également créées comme des unités d'exécution NHRT (unités d'exécution en temps réel ne

contenant pas de segments), mais avec une priorité légèrement plus basse. Voir «Ecriture des unités d'exécution RTT (real-time thread)», à la page 73 pour l'organisation des priorités.

La simulation peut s'exécuter indéfiniment de sorte que lorsqu'elle se termine, elle redémarre. Comme la simulation est constituée d'unités d'exécution NHRT, vous pouvez choisir ScopedMemory ou ImmortalMemory. L'exemple d'application utilise ScopedMemory pour la simulation, car il est préférable de quitter la zone ScopeMemoryArea allouée lorsque la simulation se termine et d'y entrer de nouveau pour attendre l'exécution suivante. Dans ce cas, aucun état n'est reporté d'une exécution à une autre.

La plupart des classes peuvent être exécutées dans les unités d'exécution NHRT. Toutefois, la plupart des classes peuvent être exécutées dans un mode ne garantissant pas leur exécution correcte dans les unités d'exécution NHRT. Par exemple, si les DatagramSockets sont conservés dans la mémoire pérenne ou dans une zone de mémoire sectorisée externe, des problèmes peuvent apparaître, car ils ne sont pas destinés à couvrir les zones de mémoire. L'exemple d'application utilise uniquement la zone ScopedMemory pour éviter ces problèmes.

Allocation de mémoire dans RTSJ

Dans RTSJ, vous pouvez allouer un objet dans une zone de mémoire donnée de différentes manières ; la méthode d'allocation à utiliser n'est pas toujours évidente.

Chaque approche a des caractéristiques qui varient en fonction des implémentations de RTSJ et se distingue par le niveau de performance ou l'impact éventuel sur la mémoire. Cette section décrit les options disponibles et suggère de les utiliser dans les cas les plus appropriés pour allouer un objet.

Initialiseur statique

La méthode la plus simple pour allouer un objet dans la mémoire pérenne consiste à l'allouer dans un initialiseur statique. Cette méthode offre l'avantage de vous éviter d'avoir à traiter les problèmes de changement de contexte de mémoire, mais les cas d'utilisation de cette méthode sont limités. Cette approche est efficace dans la mesure où la quantité de mémoire pérenne consommée est limitée à celle nécessaire à l'objet lui-même.

MemoryArea.newInstance(Class c)

Cette approche est simple si une unité d'exécution se trouve dans un contexte de mémoire et que vous voulez allouer un objet dans une autre zone qui doit être déjà dans la pile de secteur de l'unité d'exécution. Cette approche offre l'avantage d'avoir à accéder uniquement à la classe à instancier, mais la méthode newInstance doit générer un constructeur approprié. Ce modèle est particulièrement adapté si les objets d'une classe donnée doivent être alloués peu fréquemment, mais tendent à utiliser beaucoup de mémoire.

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

Là encore, il s'agit d'une approche simple si une unité d'exécution se trouve dans un contexte de mémoire et que vous voulez allouer un objet dans une autre zone qui doit être déjà dans la pile de secteur de l'unité d'exécution. Dans ce cas, vous devez envoyer un constructeur et des arguments et garantir que le constructeur est valide dans le contexte de mémoire en cours. Comme la méthode newInstance n'a

pas besoin de créer un constructeur, l'utilisation de la mémoire est inférieure à celle de newInstance(Class c) et cette méthode est donc plus appropriée si des objets doivent être alloués plus fréquemment et que vous voulez assumer la charge de l'allocation du constructeur à l'avance et le stocker dans ImmortalMemory, par exemple.

MemoryArea.enter(Runnable r) suivi de la nouvelle méthode <class>()

Cette approche désigne la zone de mémoire comme zone de mémoire par défaut pour les allocations et évite de recourir à des objets de réflexion et assistant Constructeur. Par conséquent, elle est recommandée si un grand nombre d'objets doit être créé lorsqu'aucune mémoire supplémentaire n'est utilisée au-dessus de l'objet. Cette approche fonctionne uniquement si la zone désirée n'est pas déjà active dans la pile de secteur d'une unité d'exécution. La nécessité de créer une zone de mémoire exécutable rend cette approche plus complexe qu'utiliser newInstance, car vous devez envoyer généralement les paramètres dans la zone Exécutable ou via des zones statiques ou d'instance.

MemoryArea.executeInArea(Runnable r) suivi de la nouvelle méthode <class>()

Là encore, cette approche désigne la zone de mémoire comme zone de mémoire par défaut pour les allocations et évite de recourir à des objets de réflexion et assistant Constructeur. Par conséquent, elle est recommandée si un grand nombre d'objets doit être créé lorsqu'aucune mémoire supplémentaire n'est utilisée au-dessus de l'objet. Utilisez cette approche si la zone désirée est déjà dans la pile de secteur de l'unité d'exécution en cours et elle est donc plus souple que MemoryArea.enter. La nécessité de créer une zone de mémoire exécutable rend cette approche plus complexe qu'utiliser newInstance, car vous devez envoyer généralement les paramètres dans la zone Exécutable ou via des zones statiques ou d'instance.

Class.newInstance()

Cette approche génère la nouvelle instance dans la zone de mémoire en cours et elle doit donc être utilisée avec MemoryArea.enter ou executeInArea. Aucune utilisation de mémoire supplémentaire ne se produit au-dessus de l'objet lui-même.

Utilisation du minuteur haute résolution

L'horloge temps réel est plus précise que les horloges associées à la machine JVM standard.

Avant de commencer

Cet exemple repose sur la classe RTJavaRadar.java qui se trouve dans le fichier `demo/realtime/sample_application.zip`.

Pourquoi et quand exécuter cette tâche

Java est limité pour la gestion des horloges et des minuteurs. Spécification en temps réel de Java permet d'utiliser des heures absolues spécifiques à la nanoseconde près et fournit une latitude pour le délai d'exécution d'une tâche. javax.realtime.HighResolutionTime et ses sous-classes sont utilisés pour représenter le temps avec deux composants, les millisecondes et les nanosecondes.

WebSphere Real Time for RT Linux utilise le support du système d'exploitation sous-jacent pour fournir le temps haute résolution. Les noyaux actuel Linux fournissent une horloge avec au mieux une précision de 4 millisecondes. Les correctifs Linux fournis avec WebSphere Real Time for RT Linux fournissent une horloge avec une précision proche de la microseconde.

La classe RTJavaRadar montre l'utilisation du minuteur haute résolution :

- **1** obtient l'horloge temps réel.
- **2** obtient le temps absolu en cours.
- **3** obtient le composant nanoseconde du temps. La précision de l'horloge temps réel implique que l'utilisation des nanosecondes est raisonnable.
- **4** obtient l'heure avant et après ping.
- **5** retourne la vitesse de descente du module lunaire.
- **6** demande à l'unité d'exécution d'attendre 5 millisecondes avant d'exécuter une autre itération.

```
public void run() {
       // Les objets suivants sont créés à l'avance et réutilisent chaque
       // itération.
       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;

               // Au lieu d'utiliser le format time = rtClock.getTime(), cette
               // méthode
               // remplace les valeurs dans un objet préexistant AbsoluteTime.
               rtClock.getTime(time);                  4
               millis = time.getMilliseconds();
               nanos = time.getNanoseconds();

               // Nous calculons le délai d'envoi de la commande ping et de réception de la réponse
               // 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) {
                // Pas important.
            }
        }
```

Le code ci-dessus peut être comparé au code JVM standard suivant dans la classe JavaRadar :

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

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

                nanos = System.nanoTime();

                this.radarPort.ping();

                // L'échelle de temps est huit unités par milliseconde
                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;
            }
```

Exemple d'application

L'exemple d'application utilise une série d'exemples pour montrer les fonctions de WebSphere Real Time for RT Linux que vous pouvez utiliser pour améliorer les caractéristiques temps réel des programmes Java.

Les fichiers source de l'exemple d'application se trouvent dans le fichier `demo/realtime/sample_application.zip`.

L'exemple est constitué de deux principaux composants :

- Une **simulation,** un exemple simple de module lunaire. Sa position est définie par sa hauteur au-dessus du sol et la distance par rapport à la zone d'alunissage. Voir figure 6, à la page 83.

  La classe de simulation est écrite en utilisant unités d'exécution en temps réel ne contenant pas de segments (NHRT) et elle n'est plus modifiée dans cette documentation.
- Un **contrôleur** qui envoie des commandes à la simulation. Il envoie des interrogations radar pour déterminer la hauteur du module et contrôler la vitesse de descente en fonction de ces informations. Le contrôleur reçoit également un flux d'informations du module, tel que sa distance par rapport à la zone d'alunissage.

  Le contrôleur est écrit initialement en Java standard. Dans «Modification des applications Java», à la page 72, il est développé comme programme Java temps réel.

Selon le résultat de l'alunissage, le contrôleur envoie message de crash ou d'alunissage.

En utilisant l'exemple d'application, vous pouvez exécuter les opérations suivantes :

- Exécuter la simulation et le contrôleur ensemble pour montrer une combinaison de classes temps réels et standard Java exécutées simultanément. Pour plus d'informations, voir «Génération d'un exemple d'application», à la page 84 et «Exécution de l'exemple d'application», à la page 84 qui contient également la sortie attendue de l'exemple d'application.

  Remarque : Vous pouvez démarrer la simulation et le contrôleur simultanément en utilisant la classe LaunchBoth.
- Comparer la différence qui existe avec récupérateur de place Metronome et le récupérateur de place standard. Pour plus d'informations, voir «Exécution de l'exemple d'application sans Real Time», à la page 84 et «Exécution de l'exemple d'application avec le récupérateur de place Metronome», à la page 86.
- Exécuter l'application en utilisant le compilateur AOI (ahead-of-time). Pour plus d'informations, voir «Exécution de l'exemple d'application avec AOT», à la page 87.

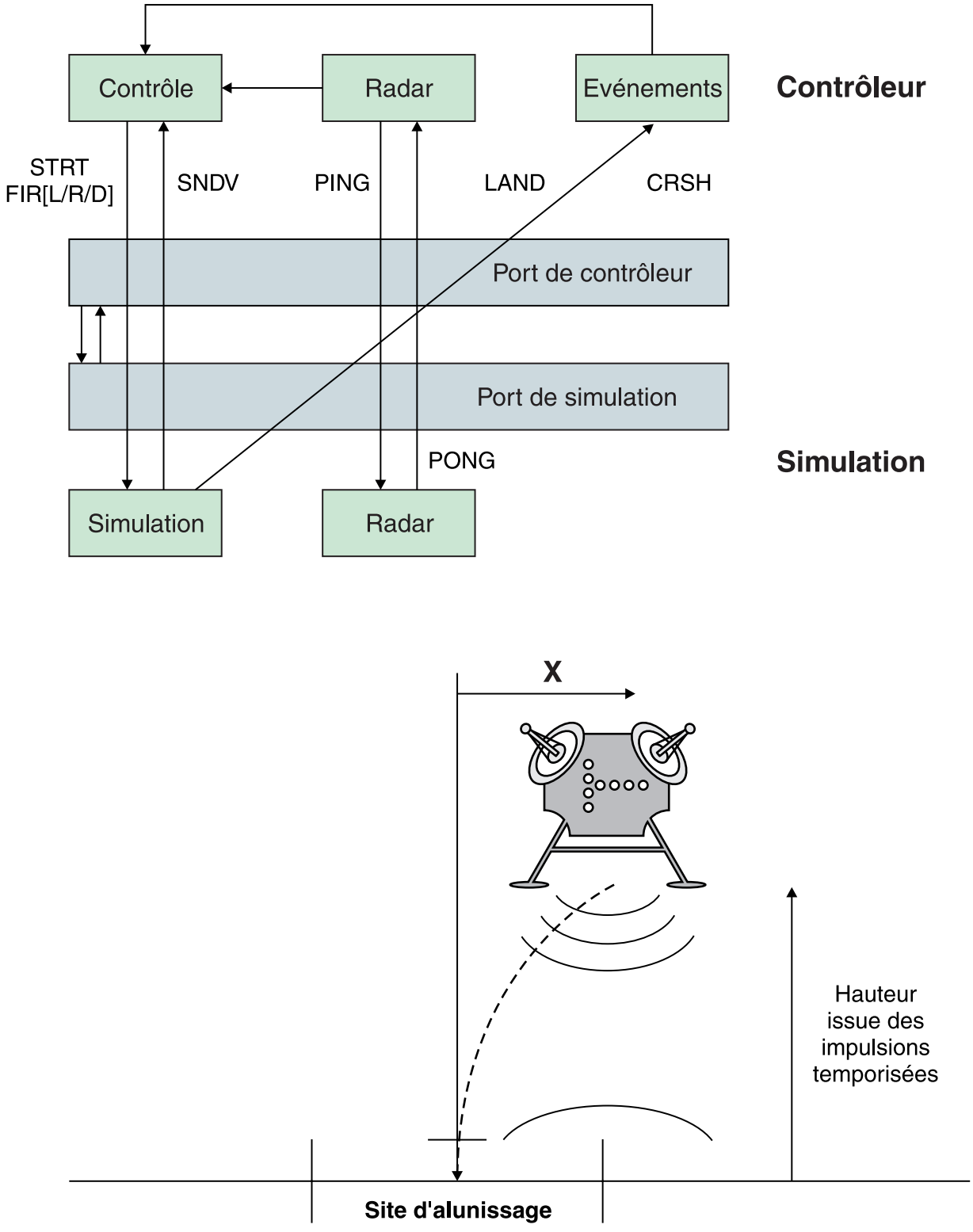

Figure 6. Diagramme du module lunaire

Ce diagramme montre la relation des modules fournis dans l'exemple. La partie supérieure du diagramme contient le contrôleur et le simulateur. Le contrôleur a trois unités d'exécution : contrôle, radar et événements. Le simulateur a deux

unités d'exécution : simulateur et radar. La partie inférieure du diagramme montre le module lunaire et indique les deux fonctions de contrôle de gauche et de droite et les pulsations qui déterminent la hauteur du module.

Génération d'un exemple d'application

L'exemple de code source d'application est fourni à titre d'aide. La préparation nécessite de décompresser et de compiler le code source Java pour pouvoir l'exécuter.

Procédure

1. Créez un répertoire de travail.
2. Extrayez l'exemple d'application dans le répertoire de travail :
   ```
   unzip sample_application.zip
   ```
3. Créez un répertoire pour la sortie :
   ```
   mkdir classes
   ```
4. Compilez la source.
   a. Générez la liste des fichiers :
      ```
      find -name "*.java" > source
      ```
   b. Compilez la source :
      ```
      javac -Xrealtime -Xlint:deprecated -g -d classes @source
      ```
   c. Créez un fichier JAR des fichiers classes :
      ```
      jar cf demo.jar -C classes/ .
      ```

Que faire ensuite

Maintenant, vous pouvez exécuter l'exemple d'application.

Exécution de l'exemple d'application

WebSphere Real Time fournit une machine JVM standard et une machine JVM temps réel démarrées à partir de l'argument de ligne de commande **-Xrealtime**.

L'exemple d'application dispose de deux composants destinés à être exécutés dans des machines JVM distinctes :

- La simulation, qui s'exécute uniquement dans Java temps réel.
- Le contrôleur qui peut s'exécuter du code Java non-temps réel ou Java temps réel.

L'exécution du code du contrôleur dans divers modes montre les avantages de la technologie IBM Real-Time Java.

Exécution de l'exemple d'application sans Real Time

Dans cette procédure, vous exécutez l'exemple d'application sans tirer parti d'IBM WebSphere Real Time.

Avant de commencer

Pour exécuter l'exemple d'application, vous devez d'abord générer l'exemple de code source. Voir «Génération d'un exemple d'application» pour plus d'informations.

Procédure

1. Démarrez la simulation :

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

Dans cette commande, *<port>* est un port désalloué pour le poste de travail.

2. Démarrez le contrôleur :

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

Dans cette commande, *<host>* est le nom d'hôte du poste de travail qui exécute la simulation et *<port>* est le port défini à l'étape précédente.

Résultats

L'application génère un message indiquant que la simulation et le contrôleur ont démarré :

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

Certains exemples de points des valeurs dans le contrôleur s'affichent sur la console :

```
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
```

Immédiatement avant l'arrêt de la simulation, un message récapitulatif d'événement est généré :

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

En plus des exemples de points et du message récapitulatif d'événement, le contrôleur génère le graphique `graph.svg` dans le même répertoire. Le graphique contient un tracé des exemples de points. Le graphique montre l'effect des interruptions de la récupération de place sur l'unité d'exécution JavaRadar lorsque l'application est exécutée avec une machine JVM non-temps réel standard. Les données correspondant à Radar Height présentent des pics. Ces pics sont causés par les interruptions de la récupération de place standard qui affectent l'application Controller. Dans certaines exécutions, les pauses de récupération de place sont suffisamment longues pour provoquer des échecs et générer le message :

```
CRASH!
```

Pour voir les délais d'interruption causés par la récupération de place, ajoutez l'option **-verbose:gc** à la commande de lancement du contrôleur :

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

Exécution de l'exemple d'application avec le récupérateur de place Metronome

Vous pouvez exécuter une application Java standard dans un environnement temps réel sans avoir à réécrire le code, en ajoutant l'option -Xrealtime. Cette option active les fonctions en langage Java temps réel et le récupérateur de place Metronome.

Avant de commencer

Pour exécuter l'exemple d'application, vous devez générer préalablement l'exemple de code source. Voir «Génération d'un exemple d'application», à la page 84 pour plus d'informations.

Procédure

1. Démarrez la simulation :

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

   Dans cette commande, <*port*> correspond à un port désalloué pour le poste de travail.
2. Démarrez le contrôleur :

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

   Dans cette commande, <*host*> est le nom d'hôte du poste de travail qui exécute la simulation et <*port*> est le port défini à l'étape précédente. L'exécution des deux machines JVM sur le même poste de travail peut générer un comportement moins déterministe. Voir «Considérations», à la page 24 pour plus d'informations.

Résultats

L'application exécute et génère plusieurs sorties, dont :

1. Messages indiquant que la simulation et le contrôleur ont démarré.
2. Exemples de points des valeurs dans le contrôleur.
3. Graphique appelé `graph.svg` dans le même répertoire, contenant un tracé des exemples de points.
4. Un message récapitulatif d'événement.

Lorsque l'application est exécutée avec la récupération de place Metronome, les exemples de points et le graphique tendent vers ce qui suit :

- Aucun pic dans les données Radar Height
- Suivi précis des données Real Height.

Cela s'explique par le fait que le code Controller s'exécute désormais avec des pauses de récupération de place plus courtes.

Les pauses sont fréquentes pendant la récupération de place Metronome mais durent généralement moins d'une milliseconde. Les pauses sont moins nombreuses pendant la récupération de place mais durent généralement des dizaines ou des centaines de millisecondes. Pour voir la différence entre les pauses ajoutez l'option **`-verbose:gc`** à la commande d'exécution Controller.

Pour plus d'informations sur la sortie de la récupération de place prolixe, voir «Utilisation des informations verbose:gc», à la page 137.

Exécution de l'exemple d'application avec AOT

Cette procédure exécute une application standard Java dans un environnement temps réel en utilisant le compilateur AOT (ahead-of-time) sans avoir à réécrire le code. Utilisez cet exemple pour comparer l'exécution de la même application lorsque le compilateur JIT est utilisé.

Voir «Utilisation du code compilé avec WebSphere Real Time for RT Linux», à la page 36 pour plus d'informations sur la compilation AOT.

Avant de commencer

Pour exécuter l'exemple d'application, vous devez d'abord générer l'exemple de code source. Voir «Génération d'un exemple d'application», à la page 84 pour plus d'informations.

Pourquoi et quand exécuter cette tâche

Le compilateur AOT compile l'application Java en code natif avant de l'exécuter. Par conséquent, vous pouvez prévoir plus précisément le mode d'exécution de l'application car la compilation JIT (Just-In-Time) ne provoque aucune interruption.

Procédure

1. Convertissez les bytecodes d'application en code natif.
   a. La conversion commence par l'exécution de l'exemple avec le compilateur JIT normal.

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

      Dans cette commande, *<port>* correspond à un port désalloué pour le poste de travail.
   b. Dans une autre fenêtre, exécutez l'application.

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

      Dans cette commande, *<port>* est le port indiqué à l'étape précédente. La sortie résultant de l'application est similaire aux messages suivants :

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

      et :

      ```
      Land!
      ```
   c. Combinez les fichiers d'options AOT créés aux étapes précédentes.

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

      Des informations de date de d'ID de processus sont ajoutées aux noms utilisés pour les fichiers journaux créés aux étapes précédentes. Le format du nom de fichier est indiqué par l'option **vlog=**. Par exemple, **vlog=sim.aotOpts** génère un nom de fichier similaire à **sim.aotOpts.20081014.234958.13205**:

d. Compilez les fichiers dans le fichier `sample.aotOpts` dans `realtime.jar`,`vm.jar` `rt.jar` et l'application `demo.jar`. Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

```
admincache -Xrealtime -populate -cacheName "sample"
-aotFilterFile sample.aotOpts -classpath ./demo.jar \
$JAVA_HOME/jre/lib/i386/realtime/jclSC160/vm.jar \
$JAVA_HOME/jre/lib/i386/realtime/jclSC160/realtime.jar \
$JAVA_HOME/jre/lib/rt.jar \
./demo.jar
```

Résultats de la compilation :

```
J9 Java(TM) admincache 1.0
Eléments sous licence - Propriété d'IBM

(c) Copyright IBM Corp. 1991, 2008 All Rights Reserved
IBM est une marque d'IBM Corp.
Java ainsi que tous les logos et toutes les marques incluant Java
sont des marques
d'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
```

Remarque : La ligne

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

indique que le cache existant "sample" est détruit par la commande pour créer le cache défini.

e. Affichez le contenu du cache rempli.

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

2. Démarrez la simulation :

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

Dans cette commande, <*port*> correspond à un port désalloué pour le poste de travail.

3. Démarrez le contrôleur :

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

Dans cette commande, <*host*> est le nom d'hôte du poste de travail qui exécute la simulation et <*port*> est le port défini à l'étape précédente. L'exécution des deux machines JVM sur le même poste de travail peut générer un comportement moins déterministe. Voir «Considérations», à la page 24 pour plus d'informations.

Résultats

L'application exécute et génère plusieurs sorties, dont :

1. Messages indiquant que la simulation et le contrôleur ont démarré.
2. Exemples de points des valeurs dans le contrôleur.
3. Graphique appelé `graph.svg` dans le même répertoire, contenant un tracé des exemples de points.
4. Un message de récapitulatif d'événement.

Lorsque l'application est exécutée avec la compilation AOT (Ahead-Of-Time) les exemples de points et le graphique tendent vers ce qui suit :

- Aucun pic dans les données Radar Height
- Suivi précis des données Real Height.

Cela s'explique par le fait que le code Controller s'exécute désormais avec des pauses de récupération de place très courtes et sans interruptions dues à la compilation JIT (just-in-time).

L'utilisation du cache de classes partagées pour exécuter cette application offre l'avantage de permettre au contrôleur et aux machines JVM de simulation de partager une partie de la mémoire utilisée par les classes chargées par les deux machines JVM.

Exemple de mappe de hachage temps réel

WebSphere Real Time for RT Linux inclut les implémentations HashMap et HashSet afin d'offrir des performances plus cohérentes pour la méthode put que l'implémentation HashMap standard dans IBM SDK for Java 7.

Le fichier java.util.HashMap standard que fournit IBM fonctionne correctement pour les application à haute capacité de traitement. Il est également utile avec les applications qui connaissent la taille maximale que doit atteindre leur mappe de hachage. Pour les applications qui nécessitent une mappe de hachage qui peut atteindre des tailles variables, en fonction de l'utilisation, il existe un problème potentiel de performances avec le mappage de hachage standard. La mappe de hachage standard est pratique pour y ajouter des entrées en utilisant la méthode put. Toutefois, lorsque la mappe de hachage est pleine, un magasin de sauvegarde plus grand doit être alloué. Cela implique que les entrées du magasin de sauvegarde en cours doivent être migrées. S'il s'agit d'une grande mappe, la durée de l'opération put peut être également longue. Par exemple, l'opération peut prendre plusieurs millisecondes.

WebSphere Real Time for RT Linux contient un exemple de mappe de hachage temps réel. Il fournit la même interface fonctionnelle que le fichier java.util.HashMap standard, mais il offre des performances plus cohérentes pour la méthode put. Au lieu de créer un magasin de sauvegarde et de migrer toutes les entrées lorsque la mappe de hachage est pleine, l'exemple de mappe de hachage crée un magasin de sauvegarde supplémentaire. Le nouveau magasin de sauvegarde est chaîné aux autres magasins de sauvegarde dans la mappe de hachage. Initialement, le chaînage cause une légère baisse de performance lors de l'affectation du magasin de vidage et de son chaînage aux autres magasins de sauvegarde. Une fois la mappe de hachage de sauvegarde mise à jour, cette opération est plus rapide que de migrer toutes les entrées. L'un des inconvénients de la mappe de hachage temps réel réside dans le fait que les opérations get, put

et remove sont légèrement plus lentes, car chaque recherche doit utiliser un groupe de mappes de hachage de sauvegarde et au lieu d'un seul.

Pour tester la mappe de hachage temps réel, ajoutez le fichier `RTTashMap.jar` au début du chemin d'accès aux classes d'amorçage. Si vous avez installé WebSphere Real Time for RT Linux dans le répertoire `$WRT_ROOT`, ajoutez l'option suivante pour utiliser la mappe de hachage temps réel avec l'application au lieu de la mappe de hachage standard :

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

Les fichiers source et classe de l'implémentation de mappe de hachage temps réel sont inclus dans le fichier `demo/realtime/RTHashMap.jar`. En outre, une implémentation java.util.LinkedHashMap et une implémentation java.util.HashSet sont également fournies.

Développement d'applications WebSphere Real Time for RT Linux à l'aide d'Eclipse

Eclipse fournit un environnement IDE complet pour développer des applications temps réel.

Avant de commencer

Si vous utilisez l'environnement de développement d'application Eclipse pour la première fois pour développer des applications temps réel, procédez comme suit pour configurer votre environnement.

WebSphere Real Time for RT Linux fournit le compilateur standard Oracle **`javac`**. Il n'existe aucune restriction sur le choix du compilateur, mais il doit produire des fichiers classe Java 5.0 valides. Toutefois, les classes javax.realtime.* Java doivent se trouver dans le chemin de génération.

Pourquoi et quand exécuter cette tâche

Pour développer des applications dans Eclipse, procédez comme suit :

Procédure

1. Téléchargez Eclipse depuis http://www.eclipse.org/downloads/. Il est recommandé d'utiliser Eclipse 3.1.2 pour effectuer une compilation Java 5.0 correcte.
2. Téléchargez IBM SDK and Runtime Environment for Linux platforms, la JVM conforme Java 2 Technology Edition, Version 5.0 pour exécuter Eclipse.
3. Extrayez ce fichier `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/realtime.jar` du package WebSphere Real Time for RT Linux.
4. Ouvrez Eclipse et créez un projet. Cliquez sur **Fichier** > **Nouveau**. Sélectionnez **Java** dans le panneau **Nouveau projet**.
5. Cliquez sur **Suivant** pour afficher le panneau **Nouveau projet Java**.
   a. Entrez un nom de projet ; RTSJ-Tests, par exemple.
   b. Vérifiez que le compilateur JDK est 5.0.
6. Cliquez sur **Terminer**.
7. Créez un répertoire de travail et importez le fichier `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/realtime.jar`.

8. Cliquez sur **Fichier** > **Nouveau** > **Dossier** pour ouvrir le panneau **Nouveau dossier**. Entrez le nom du nouveau dossier ; par exemple, *deplib*.
9. Cliquez sur **Terminer**.
10. Pour importer le fichier `realtime.jar`, cliquez sur **Fichier** > **Importer** pour ouvrir le panneau **Importer**.
11. Cliquez sur **Système de fichiers** et sur **Suivant**.
12. Ouvrez le répertoire `opt/IBM/javawrt3/jre/lib/i386/realtime/jclSC160/` dans le système de fichiers dans lequel la machine JVM a été décompressée.
13. Sélectionnez la case à cocher en regard du fichier realtime.jar, indiquez un dossier pour l'importation, par exemple, `RTSJ-Tests/deplib`, et vérifiez que l'option de **création des dossiers sélectionnés uniquement** est sélectionnée.
14. Cliquez sur **Terminer**.
15. Ajoutez le fichier JAR au chemin d'accès de bibliothèque. Cliquez avec le bouton droit de la souris sur le projet et cliquez sur **Propriétés** pour ouvrir le panneau des **propriétés**.
16. Cliquez sur **Chemin de génération Java** et sur l'onglet **Bibliothèques**. Cliquez sur **Ajouter les fichiers Jars**.
17. Cliquez sur **realtime.jar** sous le répertoire du projet. Cliquez sur **OK**.

Résultats

Si cette procédure réussit, le fichier realtime.jar apparaît dans la liste des fichiers .jar de l'onglet **Bibliothèques**.

Exemple

Eclipse peut utiliser `realtime.src.jar` pour présenter des informations supplémentaires dans les classes RTSJ. Pour ce faire, ouvrez la fenêtre des propriétés pour le fichier `realtime.jar` importé, cliquez sur **Java Source Attachment** et dans **Location path:** entrez l'emplacement du fichier `realtime.src.jar`.

Que faire ensuite

Si vous voulez créer des applications via Apache Ant with Eclipse, ajoutez le fichier `realtime.jar` au chemin d'accès aux classes dans le script de génération Ant. Par exemple :

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

<!-- Générer les fichiers .class du package -->
<target name="compile" depends="init">
<javac destdir="${rtsj.jar.dir}"
srcdir="${rtsj.src}"
target="1.5"
classpath="${rtsj.deplib}/realtime.jar:${rtsj.src}"
debug="true"/>
</target>
```

Il s'agit de la seule partie d'un script ant.

Débogage des applications

Avec Eclipse Application, le développeur peut déboguer les applications localement ou à distance.

Pourquoi et quand exécuter cette tâche

Pour déboguer une application temps réel à distance, la machine JVM à déboguer nécessite l'option suivante.
`-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=10100`

Procédure

1. Dans l'environnement Linux où s'exécute l'application, entrez :

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

   où :
   - `server=y` indique que la machine JVM accepte les connexions des programmes de débogage.
   - `suspend=y` demande à la machine JVM d'attendre qu'un programme de débogage se connecte avant de s'exécuter.
   - `address=10100` et le numéro de port qu'utilise le programme de débogage pour se connecter à la machine JVM. Ce numéro doit être généralement supérieur à 1 024.

   La machine JVM affiche le message suivant :

   ```
   Listening for transport dt_socket at address: 10100
   ```
2. Ouvrez l'application dans Eclipse et sélectionnez **Déboguer**.
3. Une nouvelle configuration de débogage des applications distantes doit être créée. Il suffit d'en créer une si une application du même projet est exécutée et écoute sur le même port pour chaque exécution.
4. Après avoir créé la configuration, entrez son nom, le nom du projet contenant l'application à déboguer, le *nom d'hôte* du poste de travail où s'exécute l'application et le numéro de port que vous avez transmis dans les options **`-agentlib`**.
5. Cliquez sur **Déboguer** pour lancer la session de débogage. La perspective de **débogage** doit être ouverte pour que vous puissiez voir l'état de la machine JMV déboguée à distance.

Exécution d'Eclipse avec la machine virtuelle Java

Cette section explique comment exécuter Eclipse avec la machine JVM WebSphere Real Time for RT Linux.

Pour exécuter Eclipse avec la machine JVM vous devez indiquer dans la commande "eclipse" :

- Le répertoire qualifié complet de l'exécutable Java de la machine JVM WebSphere Real Time for RT Linux JVM à utiliser
- L'option **`-Xrealtime`** JVM
- La taille de la mémoire pérenne que doit utiliser Eclipse. Elle doit être d'au moins 128 Mo.

Exemple d'exécution d'Eclipse avec la machine JVM :

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

Remarque : Le kit SDK Eclipse ne tire pas parti des options de mémoire Realtime disponibles pour les applications WebSphere Real Time for RT Linux. Cela a pour conséquence d'épuiser la mémoire pérenne, notamment lorsqu'Eclipse est utilisé pendant de nombreux heures ou de nombreux jours sans être redémarré. Si une

erreur **OutOfMemory** se produit, vous pouvez augmenter la valeur dans l'option **-Xgc:immortalMemorySize** pour augmenter la quantité de mémoire pérenne que doit utiliser Eclipse.

Chapitre 7. Performances

WebSphere Real Time for RT Linux est optimisé pour des pauses de récupération de données courtes plutôt que pour des performances de capacité de traitement ou une utilisation minimale de la mémoire.

Sur les systèmes qui prennent en charge l'hyperthreading, vous devez vérifier que cette fonction n'est pas activée afin d'éliminer tout impact négatif lors de l'utilisation de WebSphere Real Time for RT Linux.

La réduction de la variabilité du timing et le support pour la Spécification en temps réel de Java (RTSJ) nécessitaient de désactiver des optimisations d'exécution IBM Java standard. Par conséquent, une réduction des performances générales risque de se produire lorsqu'une application standard Java est exécutée avec le paramètre **-Xrealtime**.

Performances dans les configurations matérielles certifiées

Les systèmes certifiés ont une granularité d'horloge et une vitesse de traitement suffisantes pour atteindre les objectifs de performance WebSphere Real Time for RT Linux. Par exemple, une application bien écrite exécutée sur un système non surchargé et avec une taille de segment de mémoire appropriée subit généralement des pauses de récupération de place de loin inférieures à 1 milliseconde, de 500 microsecondes, généralement, . Au cours des cycles de récupération de place, une application avec les paramètres d'environnement par défaut n'est pas interrompue pendant plus de 30 % du temps écoulé pendant une fenêtre variable de 10 millisecondes. Le temps collectif passé dans les pauses de récupération de place sur une période de 10 millisecondes est normalement inférieur à 3 millisecondes.

Réduction de la variabilité du timing

Les deux sources principales de variabilité dans une machine JVM sont gérées dans WebSphere Real Time for RT Linux comme suit :

- préparation du code Java : le chargement et la compilation JIT (Just-In-Time) sont traités avec la compilation AOT (Ahead-Of-Time). Voir «Utilisation du compilateur AOT», à la page 38.
- Pauses de la récupération de place : les longues pauses potentielles des modes du récupération de place sont éliminées en utilisant le récupérateur de place Metronome. Voir «Utilisation du récupérateur de place Metronome», à la page 66.

Partage des données de classes entre les machines JVM en mode non-temps réel

Le partage de classes est pris en charge en mode non-temps réel mais il ne fonctionne pas de la même façon qu'en mode Temps réel.

Vous pouvez partager des données de classe entre les machines virtuelles Java en les stockant dans un fichier cache mappé en mémoire sur disque. Le partage des classes réduit la consommation globale de mémoire virtuelle lorsque plusieurs machines virtuelles partagent un cache. Il diminue également le temps de

démarrage d'une machine virtuelle après création du cache. Le cache de classes partagées est indépendant de toute machine virtuelle Java en cours d'exécution et persiste jusqu'à ce qu'il soit supprimé.

Un cache partagé peut contenir les éléments suivants :

- les classes d'amorce
- les classes d'application
- les métadonnées qui décrivent les classes
- le code compilé AOT (Ahead-of-time)

Chapitre 8. Sécurité

Cette section contient des informations importantes relatives à la sécurité.

Considérations de sécurité pour le cache de classes partagées

Le cache de classes partagées vise à faciliter la gestion et l'utilisation du cache, mais la règle de sécurité par défaut peut ne pas convenir.

Lorsque vous utilisez le cache de classes partagées, vous devez connaître les autorisations par défaut des nouveaux fichiers pour pouvoir améliorer la sécurité en limitant l'accès.

Fichier	Autorisations par défaut
Nouveaux caches partagés	Autorisations de lecture pour groupe et autre
Répertoire `javasharedresources`	Lecture, écriture et exécution world

Vous devez disposer de l'autorisation d'écriture sur le fichier de cache et le répertoire cache pour pouvoir détruire ou augmenter un cache.

Modification des autorisations du fichier cache

Pour limiter l'accès à un cache de classes partagées, utilisez la commande **chmod**.

Modification nécessaire	Commande
Limitation de l'accès à l'utilisateur et au groupe	`chmod 770 /tmp/javasharedresources`
Limitation de l'accès à l'utilisateur	`chmod 700 /tmp/javasharedresources`
Limitation de l'utilisateur à la lecture et l'écriture uniquement pour un cache donné	`chmod 600 /tmp/javasharedresources/<file for shared cache>`
Limitation de l'utilisateur et du groupe à la lecture et l'écriture d'un cache donné	`chmod 660 /tmp/javasharedresources/ <ficher de cache partagé>`

Voir «Création d'un cache de classes partagées temps réel», à la page 40 pour plus d'informations sur la création d'un cache de classes partagées.

Connexion à un cache auquel vous n'êtes pas autorisé à accéder

Si vous tentez de vous connecter à un cache pour lequel vous ne disposez pas des autorisations appropriées, le message d'erreur suivant s'affiche :

```
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.
```

Chapitre 9. Identification et résolution des problèmes et assistance

Identification et résolution des problèmes pour WebSphere Real Time for RT Linux

Méthodes générales d'identification des problèmes

L'identification des problèmes permet de comprendre le type d'erreur et de déterminer les mesures à prendre.

Après avoir identifié le problème, vous pouvez exécuter une ou plusieurs des tâches suivantes :

- Résolution du problème
- Recherche d'une solution palliative efficace
- Collecte des données nécessaires pour générer un rapport de bogues à l'attention d'IBM.

Détermination des problèmes Linux

Cette section explique comment identifier les problèmes sous Linux.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations utiles sur l'identification et la résolution des problèmes liés à Linux :

- Configuration et vérification de l'environnement Linux
- Techniques générales de débogage
- Diagnostic des pannes
- Débogage des blocages
- Débogage des fuites de mémoire
- Débogage des problèmes de performances

Vous trouverez ces informations ici : IBM SDK for Java 7 - Linux problem determination.

Informations supplémentaires fournies pour IBM WebSphere Real Time for RT Linux

Configuration et vérification de l'environnement Linux

Dans IBM WebSphere Real Time for RT Linux, vérifiez que la machine JVM est correctement configurée pour générer un vidage système.

Vidages système Linux (fichiers core)

Lorsqu'une panne se produit, les principales données de diagnostic à obtenir sont le vidage système Linux (fichier core). Pour pouvoir générer ce fichier, vous devez vérifier les paramètres de votre système d'exploitation et l'espace disque disponible comme indiqué dans le guide d'utilisation d'IBM SDK for Java 7.

Paramètres de la machine virtuelle Java

La machine virtuelle Java doit être configurée pour générer des fichiers core lorsqu'une panne se produit. Exécutez `java -Xrealtime -Xdump:what` dans la ligne de commande. La sortie de cette option est :

```
-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
```

Les valeurs affichées sont les paramètres par défaut. `events=gpf` au minimum doit être défini pour générer un fichier core lorsqu'un blocage se produit. Vous pouvez changer et définir les options avec l'option de ligne de commande **-Xdump:system[:name1=value1,name2=value2 ...]**

Techniques générales de débogage

Du fait que les noms d'unité d'exécution Java sont visibles dans le système d'exploitation vous pouvez vous aider de la commande ps pour le débogage. Avec les outils de trace vous devez utiliser les commandes correctes pour IBM WebSphere Real Time for RT Linux.

Examen des informations de processus

Sortie de la commande ps dans IBM WebSphere Real Time for RT Linux :

```
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 Sélectionne tous les processus.

L Affiche les unités d'exécution.

o Fournit un format de colonnes prédéfini à afficher. Les colonnes indiquées sont l'ID de processus, l'ID d'unité d'exécution, les règles de planification, la priorité d'unité d'exécution en temps réel, et la commande associée au

processus. Ces informations sont utiles pour comprendre quelles sont les unités d'exécution de votre application et de la machine virtuelle s'exécutent en même temps.

Outils de traçage

Les trois outils de traçage sous Linux sont **strace**, **ltrace** et **mtrace**. La commande `man strace` affiche l'ensemble complet des options disponibles.

strace
L'outil strace trace les appels système. Vous pouvez l'utiliser sur un processus déjà disponible ou le démarrer avec un nouveau processus. strace enregistre les appels système émis par un programme et les signaux reçus par un processus. Pour chaque appel système, le nom les argument et la valeur de retour son utilisés. strace vous permet de tracer un programme sans demander la source (aucune recompilation requise). Si vous utilisez strace avec l'option **-f**, il tracera les processus enfant qui ont été créés suite à un appel système affecté. Vous pouvez utiliser strace pour examiner les problèmes de modules d'extension ou essayer de comprendre pourquoi les programmes ne démarrent pas correctement.

Pour utiliser strace avec une application Java, tapez `strace java -Xrealtime <nom-classe>`.

Vous pouvez diriger la sortie de trace depuis l'outil strace vers un fichier à l'aide de l'option `-o`.

ltrace
L'outil ltrace est dépendant de la distribution. Il est très similaire à strace. Cet outil intercepte et enregistre les appels de bibliothèque dynamiques appelés par le processus d'exécution. strace fait de même pour les signaux reçus par le processus d'exécution.

Pour utiliser ltrace avec une application Java, tapez `ltrace java -Xrealtime <nom-classe>`

mtrace
mtrace est fourni avec l'ensemble d'outils de GNU. Il installe les gestionnaires spéciaux de malloc, realloc, et free, et permet de tracer et d'enregistrer toutes les utilisations de ces fonctions dans un fichier. Ce traçage réduit l'efficacité du programme et ne doit pas être activé durant une utilisation normale. Pour utiliser mtrace, définissez **IBM_MALLOCTRACE** sur 1, et **MALLOC_TRACE** pour pointer vers un fichier valide dans lequel seront stockées les informations de traçage. Vous devez disposer d'un accès en écriture sur ce fichier.

Pour utiliser mtrace avec une application Java, tapez :

```
export IBM_MALLOCTRACE=1
export MALLOC_TRACE=/tmp/file
java -Xrealtime <nom-classe>
mtrace /tmp/file
```

Diagnostic des pannes

Suivez les instructions ci-dessous pour collecter les informations relatives aux processus en cours d'exécution et à l'environnement Java avant une panne.

Collecte des informations de processus

Pour rechercher les événements survenus avant la panne, utilisez la commande **gdb** et **bt** pour afficher la trace de pile de l'unité d'exécution défaillante au lieu d'analyser le fichier core.

En savoir plus sur l'environnement Java

Utilisez le vidage Java pour déterminer ce que chaque unité d'exécution effectuait et quelles méthodes Java étaient en cours d'exécution. Mettez les adresses de fonction en correspondance avec les adresses de bibliothèque afin de déterminer la source du code exécuté aux différents points.

Utilisez l'option **-verbose:gc** pour consulter l'état du segment de mémoire Java et les zones de mémoire pérenne et sectorisée. Vous devez répondre aux questions suivantes :

- Une pénurie ayant pu provoquer la panne de mémoire s'est-elle produite dans une des zones de mémoire ?
- La panne s'est-elle produite pendant la récupération de place, ce qui indique une éventuelle erreur de récupération de place ?
- La panne s'est-elle produite après la récupération de place, ce qui indique une éventuelle altération de la mémoire ?

Débogage des problèmes de performances

Lors du débogage des problèmes de performance vous devez prendre en considération les éléments spécifiques à IBM WebSphere Real Time for RT Linux en plus des rubriques du guide d'utilisation d'IBM SDK for Java 7.

Définition de la taille des zones de mémoire

La machine virtuelle Java peut être réglée en faisant varier les tailles du segment de mémoire, de la mémoire pérenne et de la mémoire sectorisée. Choisissez la taille correcte pour optimiser les performances. L'utilisation de la bonne taille permet au récupérateur de place de fournit l'utilisation requise.

Pour savoir comment faire varier la taille des zones de mémoire, voir «Identification et résolution des incidents du récupérateur de place Metronome», à la page 137.

Compilation et performances JIT

Lorsque vous utilisez la compilations JIT vous devez tenir compte des implications sur le comportement en temps réel.

Si vous avez besoin d'un comportement prévisible, mais avez aussi besoin de meilleures performances, pensez à utiliser une compilation AOT (ahead-of-time). Pour plus d'informations, voir «Utilisation du code compilé avec WebSphere Real Time for RT Linux», à la page 36.

Restrictions connues de Linux

Linux a fait l'objet d'un développement rapide et de nombreux problèmes se sont posés concernant le l'interaction de la machine virtuelle Java et du système d'exploitation, en particulier dans le domaine des unités d'exécution.

Les restrictions suivantes peuvent affecter votre système Linux.

Unités d'exécution en tant que processus

Si le nombre d'unités d'exécution Java dépasse le nombre maximal de processus autorisés, le programme peut alors :

- Recevoir un message d'erreur

- Recevoir une erreur **SIGSEGV**
- Arrêter

Pour plus d'informations, voir *The Volano Report* sur le site http://www.volano.com/report/index.html.

Restrictions des piles flottantes

Si vous faites une exécution sans piles flottantes, quelle que soit la valeur que vous ayez définie pour **-Xss**, une taille minimum de pile native de 256 Ko pour chaque unité d'exécution est fournie.

Sur un système Linux de pile flottante, les valeurs **-Xss** sont utilisées. Si vous migrez à partir d'un système Linux de pile non flottante, assurez-vous que les valeurs **-Xss** sont assez importantes et ne dépendent pas d'un minimum de 256 Ko.

restrictions glibc

Si vous recevez un message indiquant que le bibliothèque `libjava.so` n'a pas pu être chargée en raison d'un symbole introuvable (tel que __bzero), il est possible que vous ayez installé une version plus ancienne de la bibliothèque d'exécution GNU C, glibc. Le kit de développement de logiciels (SDK) de l'implémentation d'unité d'exécution Linux nécessite glibc version 2.3.2 ou ultérieure.

Restrictions de polices

Lorsque vous effectuez une installation sur un système Red Hat, pour permettre au serveur de polices de trouver les polices Java TrueType, exécutez (par exemple sur Linux IA32) :

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

Vous devez faire cette opération lors de l'installation et vous devez être connecté en tant que «root» pour exécuter la commande. Pour consulter d'autres problèmes liés aux polices, voir le document *Linux (SDK) et le Runtime Environment User Guide*.

Problèmes de performances sur les noyaux Linux Red Hat MRG

Un problème de configuration lié aux noyaux Red Hat MRG peut provoquer un arrêt inattendu des unités d'application lorsque WebSphere Real Time démarre alors que la récupération de place en mode prolixe est activée. Ces pauses ne sont pas signalées dans la sortie GC prolixe, mais peuvent durer plusieurs millisecondes, selon la configuration du réseau. Les machines virtuelles Java démarrés par des outils utilisateurs LDAP définis à distance sont les plus affectées, car le démon de cache du service de nom (`nscd`) n'est pas démarré, d'où des retards réseau. Pour résoudre le problème, démarrez `nscd`. Procédez comme suit pour vérifier l'état du service `nscd` et corriger le problème :

1. Vérifiez que le démon `nscd` s'exécute en tapant la commande :

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

   Si le démon n'est pas en cours d'exécution, vous voyez s'afficher le message suivant :

   ```
   nscd is stopped
   ```

2. En tant que superutilisateur (root), démarrez le service `nscd` avec la commande suivante :

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

3. En tant que superutilisateur (root), modifiez les informations de démarrage du service `nscd` avec la commande suivante :

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

Le processus `nscd` est maintenant en cours d'exécution, et démarre automatiquement après le réamorçage.

Détermination des problèmes NLS

La machine JVM contient un support intégré pour différents environnements locaux.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations utiles sur l'identification et la résolution des problèmes liés à NLS :

- Présentation des polices de caractères
- Utilitaires de polices de caractères
- Problèmes communs liés à NLS et causes possibles

Vous trouverez ces informations ici : IBM SDK for Java 7 - NLS problem determination.

Détermination des problèmes ORB

L'une des premières tâches à exécuter lorsque vous déboguez un problème ORB consiste à déterminer si le problème se situe au niveau du client ou du serveur de l'application répartie. Considérez une session RMI-IIOP standard comme une communication asynchrone entre un client qui demande d'accéder à un objet et un serveur qui le fournit.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations utiles sur l'identification et la résolution des problèmes liés à ORB :

- Identification d'un problème lié à ORB
- Interprétation de la trace de pile
- Interprétation des traces ORB
- Problèmes courants
- Service ORB IBM : collecte des données

Vous trouverez ces informations ici : IBM SDK for Java 7 - ORB problem determination.

Informations supplémentaires fournies pour IBM WebSphere Real Time for RT Linux

Service ORB IBM : collecte des données

Lors de la collecte de la sortie de version Java pour le service, exécutez la commande suivante :

```
java -Xrealtime -version
```

Tests préliminaires

Lorsqu'un problème survient, le service ORB peut générer une exception `org.omg.CORBA.*` incluant :

- La cause de l'événement
- Un code mineur
- Un état d'achèvement

Pour savoir si le service ORB est la cause du problème, vérifiez les éléments suivants :

- La situation peut être reproduite dans une configuration similaire.
- La compilation JIT est désactivée.
- Aucun code compilé AOT n'est utilisé

Les autres actions incluent :

- La désactivation des processeurs supplémentaires
- La désactivation du traitement multitâche simultané (SMT) si cela est possible
- La suppression des dépendances de mémoire avec le client ou le serveur. Le manque de mémoire physique peut ralentir les performances et provoquer des blocages et des pannes. Pour résoudre ces problèmes, vérifiez que vous disposez d'une quantité de mémoire suffisante.
- La vérification des éléments de réseau physique (pare-feu, liaisons de communication, routeurs, serveurs de noms DNS). Ces éléments sont généralement à l'origine des exceptions CORBA COMM_FAILURE. Envoyez une demande ping à votre poste de travail pour effectuer un test.
- Si l'application utilise une base de données, telle que DB2, utilisez le dernier pilote fiable. Par exemple, pour isoler DB2 AppDriver, utilisez Net Driver qui est plus lent et utilise des sockets, mais plus fiable.

Résolution des erreurs OutOfMemory

Traitement des exceptions OutOfMemoryError, des fuites de mémoire et des allocations de mémoire masquées.

Pour obtenir des informations d'ordre général sur l'identification et la résolution des incidents liés au récupérateur de place Metronome, voir «Identification et résolution des incidents du récupérateur de place Metronome», à la page 137.

Diagnostic des erreurs OutOfMemoryError

Le diagnostic des exceptions OutOfMemoryError dans le récupérateur de place Metronome peut être plus complexe que dans la machine JVM du fait de la nature périodique du récupérateur de place.

Les caractéristiques des différents types de segments de mémoire sont décrites dans «Gestion de la mémoire», à la page 13. En règle générale, une application RTSJ nécessite environ 20 % de plus d'espace de segment de mémoire qu'une application standard Java.

Par défaut, la machine JVM produit la sortie de diagnostic suivante lorsqu'une erreur OutOfMemoryError non interceptée se produit :

- Cliché du protocole de sous-réseau. Voir «Utilisation des agents de vidage», à la page 116.
- Cliché de tas. Voir «Utilisation de Heapdump», à la page 125.

- Javadump. Voir «Utilisation de Javadump», à la page 119
- Vidage système ; voir «Utilisation des vidages système et de l'afficheur des vidages système», à la page 128.

Les noms de fichier de vidage figurent dans la sortie de la console :

```
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 trace Java qui figure dans la sortie de la console et disponible également dans le vidage Java indique l'emplacement de l'erreur OutOfMemoryError dans l'application Java. L'étape suivante consiste à identifier la zone de mémoire RTSJ pleine. Le composant de gestion de la mémoire JVM émet un point de trace qui indique la taille, l'adresse de bloc de classe et le nom d'espace mémoire de l'emplacement défaillant. Ce point de trace se trouve dans le cliché du protocole de sous-réseau :

```
<< 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
```

Les zones d'ID de point de trace et de données peuvent être différentes de celles indiquées en fonction du type d'objet à allouer. Dans cet exemple, le point de trace indique que l'échec d'allocation a eu lieu lorsque l'application a tenté d'allouer un objet de 33,6 Mo de type `class 0x81312d8` dans le `segment de mémoire Metronome`, le segment de mémoire `id=0x809c5f0`.

Vous pouvez identifier la zone de mémoire RTSJ affectée en consultant les informations de gestion de la mémoire dans le vidage Java :

```
NULL           ------------------------------------------------------------------
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STMEMTYPE     Object Memory
NULL           region     start      end        size       name
1STHEAP        0xF288B584 0xF2A1C000 0xF6A1C000 0x04000000 Default
NULL
1STMEMUSAGE    Total memory available: 67108864 (0x04000000)
1STMEMUSAGE    Total memory in use:    66676824 (0x03F96858)
1STMEMUSAGE    Total memory free:      00432040 (0x000697A8)
NULL
NULL           region     start      end        size       name
1STHEAP        0xF288B5A4 0xF17FF008 0xF27FF008 0x01000000 Immortal
NULL
1STMEMUSAGE Total memory available: 16777216 (0x01000000)
1STMEMUSAGE    Total memory in use:    00450816 (0x0006E100)
1STMEMUSAGE    Total memory free:      16326400 (0x00F91F00)
NULL
1STSEGTYPE     Internal Memory
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x0808DA48 0x0814A0A8 0x0814A0A8 0x0815A0A8 0x01000040 0x00010000
1STSEGMENT     0x0808DB50 0x08131EB8 0x08131EB8 0x08141EB8 0x01000040 0x00010000
<< lines removed for clarity >>
```

Vous pouvez déterminer le type d'objet à allouer en consultant la section des classes dans le vidage Java:

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

Les informations dans le vidage Java confirme que la tentative d'allocation concerne un tableau de caractères dans le segment de mémoire normale (`ID=0xF288B584`) et que la taille totale allouée du segment de mémoire, indiquée par la ligne `1STHEAP` appropriée est de 67108864 octets décimaux ou 0x04000000 octets hexadécimaux, soit 64 Mo.

Dans cet exemple, l'allocation qui a échoué est grande par rapport à la taille totale du segment de mémoire. Si l'application doit créer des objets de 33 Mo, l'étape suivante consiste à augmenter la taille du segment en utilisant l'option **-Xmx**.

L'allocation qui échoue est généralement petite par rapport à la taille totale du segment du fait des allocations précédentes qui remplissent le segment de mémoire. Dans ce cas, l'étape suivante consiste à utiliser le cliché de tas pour déterminer la quantité de mémoire allouée aux objets existants.

Le cliché de tas est un fichier binaire compressé contenant tous les objets avec leurs classes d'objet, tailles et références. Analysez le cliché de tas en utilisant l'outil Memory Dump Diagnostics for Java (MDD4J) que vous pouvez télécharger depuis IBM Support Assistant (ISA).

En utilisant MDD4J, vous pouvez charger un cliché de tas et recherchez des structures arborescentes d'objets qui semblent consommer de grandes quantités d'espace de segment de mémoire. L'outil fournit des vues des objets dans le segment de mémoire. Par exemple, MDD4J peut afficher une vue qui détaille des fuites vraisemblablement suspectes et qui indique les cinq premiers objets et packages qui contribuent à la taille de segment de mémoire. La sélection de la vue arborescente fournit des informations supplémentaires sur la nature de l'objet conteneur qui fuit.

Par défaut, un fichier de cliché de tas contenant tous les objets dans tous les espaces mémoire RTSJ est généré. Utilisez l'option **-Xdump:heap:request=multiple** de ligne de commande pour demander un cliché de tas pour chaque espace mémoire. Avec plusieurs vidages, vous pouvez examiner simplement le groupe d'objets alloués dans une zone de mémoire donnée. Vous identifiez les clichés de tas par le nom de fichier qui figure dans la sortie de la console :

```
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... >>
```

Gestion de la mémoire par la machine JVM IBM

La machine JVM IBM nécessite de la mémoire pour divers composants, notamment les régions de mémoire des classes, le code compilé, les objets Java, les piles Java et les piles JNI. Certaines de ces régions de mémoire doivent se trouver dans la mémoire contiguë. Les autres régions de mémoire peuvent être segmentées dans des régions plus petites et interconnectées.

Les classes chargées dynamiquement et le code compilé sont stockés dans des régions de mémoire segmentées pour les classes chargées dynamiquement. Les classes sont aussi divisées en régions de mémoire inscriptibles (classes RAM) et régions de mémoire en lecture seule (classes ROM). Lors de l'exécution, le cache de classes est mappé dans la mémoire, mais pas nécessairement chargé dans une région de mémoire contiguë lors du démarrage de l'application. Lorsque les classes sont référencées par l'application, les classes et le code compilé dans le cache de classes sont mappés dans la mémoire. Le composant ROM de la classe est partagé entre plusieurs processus qui référencent la classe. Le composant RAM de la classe est créé dans les régions de mémoire segmentées pour les classes chargées dynamiquement lorsque la classe est référencée pour la première fois par la machine JVM. Le code compilé par AOT pour les méthodes d'une classe dans le cache des classes est copié dans une région de mémoire de code dynamique exécutable, car ce code n'est pas partagé par les processus. Les classes qui ne sont pas chargées depuis le cache des classes sont similaires aux classes en cache, sauf que les informations de classe ROM sont créées dans des régions de mémoire segmentées pour les classes chargées dynamiquement. Le code généré dynamiquement est stocké dans les mêmes régions de mémoire de code dynamique que celles qui contiennent le code AOT des classes en cache.

Tous les objets Java sont stockés dans le segment de mémoire standard lors de l'exécution de la machine JVM avec l'option **-Xrealtime**. Si vous utilisez l'option **-Xrealtime**, les objets peuvent être alloués depuis des régions de mémoire supplémentaires appelées Mémoire pérenne et Mémoire sectorisée.

La pile de chaque unité d'exécution Java peut couvrir une région de mémoire segmentée. La pile JNI de chaque unité d'exécution occupe une région de mémoire contiguë.

Pour déterminer la manière dont la machine JVM est configurée, exécutez la machine JVM avec l'option **-verbose:sizes**. Cette option affiche les informations sur les régions de mémoire où vous pouvez gérer la taille. Pour les régions de mémoire qui ne sont pas contiguës, un incrément est affiché pour indiquer comment la mémoire est obtenue chaque fois que la taille de la région doit augmenter.

Voici un exemple de sortie en utilisant les options **-Xrealtime -verbose:sizes** :

```
-Xmca32K                        RAM class segment increment
-Xmco128K                       ROM class segment increment
-Xms64M                         initial memory size-Xgc:immortalMemorySize=16M
                                immortal memory space size
-Xgc:scopedMemoryMaximumSize=8M scoped memory space maximum size
-Xmx64M                         memory maximum
-Xmso256K                       operating system thread stack size
-Xiss2K                         java thread stack initial size
-Xssi16K                        java thread stack increment
-Xss256K                        java thread stack maximum size
```

Cet exemple indique que le segment de classe RAM est initialement égal à 0, mais qu'il augmente par bloc de 32 Ko en fonction des besoins. Le segment de classe ROM est initialement égal à 0 et augmente par bloc de 128 Ko en fonction des besoins. Vous pouvez utiliser les options **-Xmca** et **-Xmco** pour contrôler ces tailles. Les segments de classe RAM et ROM augmentent en fonction des besoins et vous n'avez donc pas à changer généralement ces options.

La mémoire pérenne est une région contiguë et il peut être nécessaire de lui préallouer un espace plus grand. Dans cet exemple, 16 Mo sont préalloués à la région de mémoire pérenne. Si vous tentez d'écrire plus de 16 Mo dans la mémoire pérenne, vous recevez une exception OutOfMemory, car cette zone de mémoire de fait pas l'objet d'une récupération de place, par définition.

La région de mémoire sectorisée est contiguë et 80 Mo lui sont préalloués dans cet exemple. Si des zones de mémoire sectorisée sont actives lorsque le programme s'exécute, vous devez définir une région de mémoire sectorisée plus grande.

Utilisez l'utilitaire pour déterminer la taille de la région mappée en mémoire si vous utilisez le cache de classes. Voici un exemple de sortie de la commande `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 taille de cache indique que la région mappée en mémoire dépassera légèrement 7 Mo. Les octets de classe ROM et AOT occupent la majorité de l'espace en utilisant chacun un peu plus de 3 Mo.

Exemple OutOfMemoryError dans la mémoire pérenne

Cet exemple montre comment identifier une erreur OutOfMemoryError dans la mémoire pérenne et décrit les étapes à suivre pour éviter le problème.

Le cliché du protocole de sous-réseau montre que deux demandes d'allocation ont échoué dans la zone de mémoire pérenne 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
```

The Javadump shows that the immortal memory space is full:

```
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
```

Une analyse MDD4J montre qu'une très grande liste LinkedList a été allouée et consomme une grande quantité de la mémoire disponible.

Il est recommandé de réduire le nombre d'objets alloués dans la zone de mémoire pérenne, car ces objets ne font pas l''objet d'une récupération de place. La mémoire pérenne sert généralement à charger les classes, ce qui est une activité finie qui a lieu lors de l'initialisation de la machine JVM et de l'application. Les applications ayant un grand nombre de classes chargées (ou dont la mémoire pérenne sert à autre chose) peuvent augmenter la taille de la mémoire pérenne en utilisant l'option **-Xgc:immortalMemorySize=<size>**. La taille par défaut de la mémoire pérenne est de 16 Mo.

Si l'augmentation de la taille de la mémoire pérenne retarde uniquement l'erreur OutOfMemoryError pour la mémoire pérenne, analysez le modèle d'allocation continue des données pérennes en relation avec le chargement des classes ou d'autres objets d'applications.

Exemple OutOfMemoryError dans l'espace de mémoire sectorisée

Cet exemple montre comment identifier une erreur OutOfMemoryError dans la mémoire sectorisée et décrit les étapes à suivre pour éviter le problème.

Utilisez l'option de ligne de commande **-Xdump:heap:request=multiple** pour produire des vidages distincts pour chaque espace de mémoire :

```
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... >>
```

Le cliché du protocole de sous-réseau indique que deux demandes d'allocation ont échoué dans la zone de mémoire sectorisée 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
```

Le vidage Java indique que pour la zone de mémoire sectorisée avec id=0x809dd10, la taille allouée de la zone de mémoire est tout à fait petite (60 Ko). Dans ce cas, augmentez la zone de mémoire sectorisée dans le code d'application.

```
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
```

Dans l'exemple de javadump, la zone de mémoire sectorisée est vide. Elle est vide, car le vidage Java est généré lorsque l'erreur OutOfMemoryError atteint la JVM, moment à partir duquel le secteur est quitté et nettoyé. Vous pouvez produire un vidage Java au point d'échec en utilisant l'option de ligne de commande **-Xdump:java:events=throw,filter=java/lang/OutOfMemoryError**. En utilisant cette option, l'espace libre dans la zone de mémoire sectorisée est signalé correctement.

Il est possible d'épuisé complètement l'espace disponible dans la mémoire sectorisée. Dans ce cas, augmentez la taille de la mémoire sectorisée en utilisant l'option de ligne de commande **-Xgc:scopedMemoryMaximumSize=<size>**. La taille par défaut de la zone de mémoire sectorisée est de 8 Mo. Si l'espace total disponible pour la mémoire sectorisée est épuisé, consultez les messages sur la console. Par exemple :

```
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)
```

Problèmes de diagnostic dans plusieurs segments de mémoire

Vous pouvez utiliser les plages d'adresses fournies dans le vidage Java avec les informations d'occupation dans le cliché de tas pour analyser les erreurs OutOfMemoryError dans plusieurs zones de mémoire RTSJ.

Dans ce javadump, le segment pérenne est compris entre 0xB281C008 et 0xB381C008 et le segment de mémoire normal est compris entre 0xB381D008 et 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
```

Le cliché de tas est un fichier binaire compressé contenant tous les objets avec leurs classes d'objet, tailles et références. Analysez le cliché de tas en utilisant l'outil Memory Dump Diagnostics for Java (MDD4J) que vous pouvez télécharger depuis IBM Support Assistant (ISA).

Vous pouvez utiliser les emplacements de mémoire d'objets listés par MDD4J pour déterminer l'espace mémoire où se trouve un objet. Les adresse comprises dans la plage 0xB28*nnnnn* se trouvent dans la zone de mémoire pérenne. Les adresses de la plage 0xB61*nnnnn* se trouvent dans le segment de mémoire normal.

Eviter les fuites de mémoire

Le récupérateur de place ne traite pas les zones de mémoire pérenne ou de mémoire sectorisée. La mémoire pérenne est libérée uniquement lorsque la machine JVM quitte. Les zones de mémoire sectorisée sont libérées uniquement lorsque leur nombre de référence revient à zéro. Les tâches longues dans ces contextes doivent être écrites de sorte que, après la mise en fonctionnement de la tâche, aucune mémoire supplémentaire de la mémoire pérenne n'est allouée.

Le chargement des classes utilise une petite quantité de mémoire pérenne. Ces classes ne sont pas soumises à la récupération de place dans l'environnement temps réel. Par conséquent, le chargement des classes qui ne sont pas nécessaires à l'application peuvent amener cette dernière à utiliser plus de mémoire pérenne que nécessaire.

Si l'application contient des classes qui implémentent l'interface Serializable, ajustez la taille de la mémoire pérenne pour tenir compte de l'encombrement des classes générées. Chaque constructeur a un objet généré par classe sous la forme "GeneratedSerializationConstructorAccessorXXX" (où XXX est un nombre) qui est chargé dans la mémoire pérenne lors de la première sérialisation de l'objet.

Evitez d'utiliser la mémoire pérenne car la récupération de place n'est pas possible sur les objets alloués à partir de cette dernière. Envisagez de regrouper les objets dans la mémoire pérenne si cette dernière est utilisée régulièrement.

Allocation de mémoire masquée via des fonctions de langage

Dans un contexte de mémoire sectorisée ou pérenne, évitez les arguments variables du langage, car ces méthodes allouent de la mémoire masquée.

Arguments variables (vararg)

Le langage Java implémente des arguments variables en les envoyant à la méthode sous la forme d'un tableau. Le compilateur facilite l'appel des méthodes d'argument variable en créant et en initialisant automatiquement le tableau.

De la mémoire peut être perdue en appelant une méthode d'argument variable dans un contexte de mémoire pérenne ou sectorisée. N'utilisez pas des arguments variables dans les contextes de mémoire sectorisée et pérenne. A la place, créez un tableau et utilisez-le à la place des arguments variables.

Voici deux exemples montrant des manières équivalentes d'appeler une méthode d'argument 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;
    }
}
```

Le second exemple est préférable. Du fait que l'allocation avec tableau double devient visible dans le code, l'allocation peut être dirigée vers une zone de mémoire particulière.

Concaténation de chaînes

L'ajout à une chaîne existante pour produire une chaîne plus longue est implémenté en utilisant des objets java.lang.StringBuilder, ce qui nécessite des allocations de mémoire.

Emballage automatique

L'emballage automatique implique de créer un objet pour y placer un type de base, ce qui nécessite des allocations de mémoire.

Utilisation de la réflexion dans les contextes de mémoire

Si un objet constructeur a été généré dans une zone de mémoire sectorisée, il peut être utilisé uniquement dans le même secteur ou un secteur interne. Toute tentative d'utilisation de l'objet constructeur dans un contexte de mémoire pérenne, de segment de mémoire et de mémoire sectorisée externe échoue.

L'exception émise lors de la réflexion dans les contextes de mémoire se présente comme suit :

```
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)
```

Vous pouvez éliminer cette restriction en utilisant le constructeur dans le secteur dans lequel il est alloué.

Utilisation des classes internes avec des zones de mémoire sectorisées

Lorsque vous utilisez des classes internes dans le contexte des zones de mémoire sectorisées, vous devez prendre les précautions nécessaires lors de l'instanciation des objets de classe interne si les objets externes et internes se trouvent dans des zones de mémoire différentes. Une erreur IllegalAssignmentError est émise par le

code généré par le compilateur qui n'est pas visible dans le code source d'origine si l'objet interne ne peut pas stocker une référence à l'objet externe.

Un objet de classe interne doit pouvoir stocker une référence implicite à son objet de classe externe. Si la référence viole les règles de référence de mémoire RTSJ, une erreur IllegalAssignmentError est générée.

La plupart des classes internes (y compris les classes internes locales et anonymes) contiennent une zone non statiques (synthétique) générée lors de la compilation pour l'instance de la classe externe fermante lexicalement. La seule exception se produit lorsqu'une instance de classe interne ne dispose pas d'un objet externe fermant, tel qu'un objet de classe anonyme instancié dans un bloc d'initialiseur statique. La zone synthétique de l'objet interne contient une référence à l'objet externe. Cela est implémenté par le compilateur pour faciliter le travail du programmeur Java. La zone n'est pas visible dans le code source d'origine, mais vous pouvez écrire un code similaire en utilisant des classes imbriquées statiques avec une référence visible. Si la référence implicite viole les règles de zone de mémoire RTSJ, une erreur IllegalAssignmentError est émise lors de la construction de l'objet interne, car il tente de stocker la référence à l'objet externe.

En règle générale, vous ne pouvez pas violer les règles de références de mémoire RTSJ lorsque vous utilisez des classes internes. Vous ne pouvez pas créer un objet interne si une référence à l'objet externe associé viole les règles de référence de mémoire RTSJ. Cette règle implique qu'un objet interne alloué dans la mémoire pérenne ou un segment de mémoire ne peut pas avoir une référence à un objet externe de la mémoire sectorisée. Un objet interne de la mémoire sectorisée peut avoir une référence à un objet externe de la mémoire sectorisée, mais l'objet externe doit être alloué depuis la même zone de mémoire sectorisée ou une zone de mémoire sectorisée externe.

Il existe des solutions palliatives :

- Utilisez des classes imbriquées statiques pour éliminer la référence implicite.
- Choisissez des zones de mémoire pour que les relations d'objets internes et externes ne violent pas les restrictions de référence de zone de mémoire.

Utilisation des outils de diagnostic

Un certain nombre d'outils de diagnostic sont fournis pour vous aider à identifier et résoudre les problèmes liés à la machine virtuelle Java d'IBM WebSphere Real Time for RT Linux.

Le kit SDK IBM pour Java 7 fournit des outils de diagnostic pour vous aider à identifier et résoudre les problèmes liés à la machine virtuelle Java d'IBM WebSphere Real Time for RT Linux. Cette section présente les outils disponibles et fournit des liens vers des informations supplémentaires relatives à leur utilisation.

Lorsque vous utilisez les outils de diagnostic du kit SDK vous devez tenir compte d'un point important. Lorsque vous appelez la machine virtuelle Java en temps réel, utilisez l'option suivante :

```
java -Xrealtime
```

Cette option doit être utilisée lors de l'exécution des outils de diagnostic pour la machine virtuelle Java en temps réel. Par exemple, pour afficher les agents de vidage enregistrés destinés à la machine virtuelle Java, entrez :

```
java -Xrealtime -Xdump:what
```

Cette section fournit également des informations supplémentaires sur les différences d'utilisation de ces outils avec IBM WebSphere Real Time for RT Linux, ainsi que des exemples de sortie pour vous aider à identifier et résoudre les problèmes.

Vous trouverez un récapitulatif des informations d'identification et de résolution des problèmes générées par le kit SDK IBM pour Java 7 dans la section du récapitulatif des informations d'identification et de résolution des problèmes.

Utilisation des agents de vidage

Les agents de vidage sont configurés lors de l'initialisation de la machine virtuelle Java. Ils permettent d'utiliser les événements qui se produisent dans la machine virtuelle Java, tels que la récupération de place ou l'arrêt de la machine JVM, pour exécuter des vidages ou lancer un outil externe.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur les agents de vidage :

- Utilisation de l'option **-Xdump**
- Agents de vidage
- Evénements de vidage
- Contrôle avancé des agents de vidage
- Jetons d'agent de vidage
- Agents de vidage par défaut
- Suppression des agents de vidage
- Variables d'environnement des agents de vidage
- Mappage des signaux
- Emplacements par défaut des agents de vidage

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using dump agents.

Informations supplémentaires relatives à IBM WebSphere Real Time for RT Linux :

Evénements de vidage

Les agents de vidage sont déclenchés par des événements qui se produisent lorsque la machine virtuelle Java est active. Pour IBM WebSphere Real Time for RT Linux, la valeur par défaut de l'événement slow et égale à 5 millisecondes.

Certains événements sont filtrés pour améliorer la pertinence de la sortie. Voir «Option de filtrage», à la page 117 pour plus d'informations.

Remarque : Les événements de déchargement et d'extension ne se produisent pas actuellement dans WebSphere Real Time. Les classes sont une mémoire permanente et ne peuvent pas être déchargées.

Remarque : Les événements gpf et abort ne peuvent pas déclencher un cliché de tas (request=prepwalk), préparer le segment de mémoire (request=prepwalk) ou compacter le segment de mémoire (request=compact).

Le tableau suivant répertorie les événements disponibles comme déclencheurs d'agent de vidage :

Evénement	Moment de déclenchement	Option de filtrage
gpf	Erreur de protection générale (GPF).	

Evénement	Moment de déclenchement	Option de filtrage
utilisateur	La machine JVM reçoit le signal SIGQUIT du système d'exploitation.	
abort	La machine JVM reçoit le signal SIGABRT du système d'exploitation.	
vmstart	Démarrage de la machine virtuelle.	
vmstop	Arrêt de la machine virtuelle.	Filtre le code exit. Par exemple, **filter=#129..#192#-42#255**.
load	Chargement d'une classe.	Filtre le nom de classe. Par exemple **filter=java/lang/String**.
unload	Déchargement d'une classe.	
throw	Envoi d'une exception.	Filtre le nom de classe d'exception. Par exemple, **filter=java/lang/OutOfMem***
catch	Interception d'une exception.	Filtre le nom de classe d'exception. Par exemple **filter=*Memory***.
uncaught	Non-interception d'une exception Java par l'application.	Filtre le nom de classe d'exception. Par exemple, **filter=*MemoryError**.
systhrow	Exception Java sur le point d'être émise par la machine virtuelle Java. Différent de l'événement 'throw', car l'exception est déclenchée uniquement pour les conditions d'erreur détectées en interne par la machine virtuelle Java.	Filtre le nom de classe d'exception. Par exemple, **filter=java/lang/OutOfMem***
thrstart	Démarrage d'une nouvelle unité d'exécution.	
blocked	Blocage d'une unité d'exécution.	
thrstop	Arrêt d'une unité d'exception.	
fullgc	Démarrage d'un cycle de récupération de place.	
slow	Une unité d'exécution prend plus de 5 ms pour répondre à une demande JVM interne.	Change le délai pour qu'un événement soit considéré lent. Par exemple, **filter=#300ms** se déclenche lorsqu'une unité d'exécution prend plus de 300 ms pour répondre à une demande JVM interne.
allocation	Un objet Java est alloué avec une taille correspondant à la spécification de filtrage définie	Filtre la taille d'objet. Un filtre doit être défini. Par exemple, **filter=#5m** se déclenche sur les objets de plus de 5 Mo. Les plages sont également prises en charge. Par exemple, **filter=#256k..512k** se déclenche sur les objets dont la taille est comprise entre 256 ko et 512 ko.
traceassert	Une erreur interne s'est produite dans la machine JVM	Non applicable.
corruptcache	La machine JVM détecte que le cache de classes partagées est endommagé.	Non applicable.

Option de filtrage

Certains événements JVM se produisent des milliers de fois pendant la durée de vie d'une application. Les agents de vidage peuvent utiliser des filtres et des plages pour éviter de produire un trop grand nombre de vidages.

Caractères génériques

Vous pouvez utiliser un caractère générique dans le filtre d'événement d'exception en plaçant un astérisque uniquement au début ou à la fin du filtre. Les commandes suivantes ne fonctionnent pas, car le second astérisque ne se trouve pas à la fin :

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

Pour que le filtre puisse fonctionner, vous devez utiliser :

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

Chargement de classe et événements d'exception

Vous pouvez filtrer le chargement de classe (load) et les événements d'exception (throw, catch, uncaught, systhrow) en fonction d'un nom de classe Java :

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

Vous pouvez filtrer les événements d'exception throw, uncaught et systhrow par nom de méthode Java :

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

Les parties facultatives sont indiquées entre crochets.

Vous pouvez filtrer les événements d'exception par nom de méthode Java :

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

Les parties facultatives sont indiquées entre crochets.

Evénement vmstop

Vous pouvez filtrer l'événement d'arrêt de la machine virtuelle Java en utilisant un ou plusieurs codes exit :

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

Evénement slow

Vous pouvez filtrer l'événement slow pour modifier le seuil de délai par défaut égal 5 ms :

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

Vous ne pouvez pas affecter au filtre un délai inférieur au délai par défaut.

Evénement d'allocation

Vous devez filtrer l'événement d'allocation pour définir la taille des objets qui provoquent un déclenchement. Vous pouvez définir une taille de filtre comprise entre zéro et la valeur maximale d'un pointeur 32 bits sur les plateformes 32 bits ou la valeur maximale d'un pointeur 64 bits sur les plateformes 64 bits. La définition d'une valeur de filtre inférieure à zéro déclenche un vidage de toutes les allocations.

Par exemple, pour déclencher des vidages sur des allocations d'une taille supérieure à 5 Mo, utilisez :

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

Pour déclencher des vidages sur des allocations dont la taille est comprise entre 256 ko et 512 ko, utilisez :

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

Autres événements

Si vous appliquez un filtre à un événement qui ne prend pas en charge le filtrage, le filtre est ignoré.

Option request

Utilisez l'option request pour demander à la machine virtuelle Java de préparer l'état avant le démarrage de l'agent de vidage. Pour IBM WebSphere Real Time for RT Linux il existe une option de demande supplémentaire ; **multiple**.

Les options disponibles sont répertoriées dans le tableau suivant :

Valeur de l'option	Description
exclusive	Demande d'accès exclusif à la machine virtuelle Java.
compact	Exécute la récupération de place. Cette option supprime tous les objets inaccessibles du segment de mémoire avant la génération du vidage.
prepwalk	Prépare le segment de mémoire pour l'examen. Vous devez définir également **exclusive** quand l'option doit être utilisée.
serial	Suspend les autres vidages jusqu'à la fin du vidage.
multiple	Produit des clichés de tas distincts pour chaque zone de mémoire RTSJ.
preempt	S'applique à l'agent de vidage Java et indique si l'anticipation des unités d'exécution natives est forcée dans le processus afin de collecter les traces de pile. Si cette option n'est pas définie, seules les traces de pile Java sont collectée dans le vidage java.

Par exemple, le paramètre par défaut de l'option de demande des Javadumps est `request=exclusive+preempt`. Pour changer les paramètres afin que les Javadumps soient générés sans préempter les unités d'exécution pour collecter les traces de pile native, utilisez l'option suivante :

```
-Xdump:java:request=exclusive
```

En règle générale, les options request par défaut suffisent.

Vous pouvez indiquer plusieurs options de demande en spécifiant +. Par exemple :

```
-Xdump:heap:request=exclusive+compact+prepwalk
```

Utilisation de Javadump

Javadump génère des fichiers qui contiennent des informations de diagnostic détaillées associées à la machine virtuelle Java et une application Java capturée à un moment donné au cours de l'exécution. Par exemple, les informations peuvent porter sur le système d'exploitation, l'environnement de l'application, les unités d'exécution, les piles, les verrous et la mémoire.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur les Javadumps :

- Activation d'un vidage Java
- Déclenchement d'un vidage Java
- Interprétation d'un vidage Java
- Variables d'environnement et Javadump

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using Javadump.

Des informations supplémentaires et un exemple de sortie relatifs à IBM WebSphere Real Time for RT Linux sont fournis dans les rubriques suivantes.

Gestion de la mémoire (MEMINFO)

La section `MEMINFO` fournit des informations sur le gestionnaire de mémoire, notamment sur les zones de segment de mémoire, de mémoire pérenne et de mémoire sectorisée.

La section `MEMINFO` d'un fichier de vidage Javadump fournit des informations sur le gestionnaire de mémoire. Voir Utilisation du récupérateur de place métronome pour plus d'informations sur le fonctionnement du composant du gestionnaire de mémoire.

Cette partie du fichier de vidage Javadump fournit plusieurs valeurs de gestion suivantes :

- Quantité de mémoire disponible
- Quantité de mémoire utilisée
- Taille actuelle du segment de mémoire
- Taille actuelle des zones de mémoire pérenne
- Taille actuelle des zones de mémoire sectorisée

Cette section contient également les données d'historique de récupération de place. Ces données sont fournies sous la forme d'une séquence de points de trace horodatés, le dernier point de trace apparaissant en premier.

Les vidages Java produits par la machine virtuelle Java standard contiennent une section «GC History». Ces informations ne figurent pas dans les vidages Java générés en utilisant la machine JVM temps réel. Utilisez l'option **`-verbose:gc`** ou la trace d'instantané JVM pour obtenir des informations sur la récupération de place. Pour plus d'informations, voir «Utilisation des informations verbose:gc», à la page 137 et la section relative aux agents de vidage dans le guide d'utilisation d'IBM SDK for Java 7.

Si vous exécutez un programme qui utilise de la mémoire sectorisée, et qu'une exception OutOfMemoryError est générée, il se peut que certaines des zones de mémoire figurant dans le vidage Javadump soient vides. Lorsqu'une portée imbriquée dans une autre manque de mémoire, la portée interne peut avoir été supprimée au moment de la génération du vidage Java. Pour obtenir des informations associées à l'état des zones de mémoire lors de l'émission de l'exception OutOfMemoryError, exécutez le programme avec l'option de ligne de commande suivante :

```
-Xdump:java:events=throw,filter=java/lang/OutOfMemoryError,range=1..1
```

Cette option génère un fichier Javadump supplémentaire lorsque l'exception OutOfMemoryError est émise et pas lorsque l'exception non interceptée est

détectée,ce qui se produit un peu plus tard. Dans ce fichier vous pouvez identifier toutes les zones de mémoire actives lors de l'émission de l'exception OutOfMemoryError, y compris les portées internes. Pour plus d'informations sur l'utilisation de l'option **-Xdump**, voir le guide d'utilisation d'IBM SDK for Java 7.

Dans un Javadump, les segments sont des blocs de mémoire alloués par l'exécution Java pour les tâches qui utilisent de grandes quantité de mémoire. Exemples de tâches :

- Gestion des caches JIT
- Enregistrement des classes Java

L'exécution Java alloue également une autre mémoire native qui ne figure pas dans la section MEMINFO. La mémoire totale utilisée par les segments d'exécution Java ne représente pas nécessairement l'utilisation de la mémoire complète de l'exécution Java. Un segment d'exécution Java est constitué de la structure des données du segment et d'un bloc associé de mémoire native.

L'exemple suivant montre une sortie standard. Toutes les valeurs sont des valeurs hexadécimales. Les en-têtes de colonne dans la section MEMINFO ont la signification suivante :

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

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

Unités d'exécution et trace de pile (THREADS)

Pour les programmeurs d'applications, l'un des éléments les plus utiles d'un vidage Java est la section THREADS. Cette section répertorie les unités d'exécution Java, les unités d'exécution natives et les traces de pile. Pour IBM WebSphere Real Time for RT Linux les unités d'exécution en temps réel et les unités d'exécution en temps réel sans accès au segment de mémoire sont également affichées.

Une unité d'exécution Java est implémentée par une unité d'exécution native du système d'exploitation. Chaque unité d'exécution est représentée par un ensemble de lignes telles que :

```
"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)
```

Les noms d'unité d'exécution Java sont visibles dans le système d'exploitation lorsque vous utilisez la commande **ps**. Pour plus d'informations sur l'utilisation de la commande **ps**, voir «Techniques générales de débogage», à la page 100.

Un vidage Javadump produit à partir d'une unité d'exécution temps réel sans accès au segment de mémoire peut ne pas contenir certaines informations. Si l'objet de nom d'unité d'exécution n'est pas visible depuis l'unité d'exécution temps réel sans accès au segment de mémoire, le texte «(access error)» apparaît à la place du nom de l'unité d'exécution.

Les propriétés figurant sur la première ligne correspondent au nom de l'unité d'exécution, aux adresses des structures de l'unité d'exécution de la machine virtuelle Java et de l'objet de l'unité d'exécution Java, à l'état de l'unité d'exécution et à la priorité de l'unité d'exécution Java. Sur la deuxième ligne des propriétés figurent l'ID d'unité d'exécution du système d'exploitation natif, la priorité de l'unité d'exécution du système d'exploitation natif et la règle de planification du système d'exploitation natif.

Les noms d'unité d'exécution sont visibles de trois manières :

- Répertoriés dans des fichiers javacore. Toutes les unités d'exécution n'apparaissent pas dans les fichiers javacore.
- Dans des listes d'unités d'exécution générées à partir du système d'exploitation à l'aide de la commande **ps**.
- Via la méthode java.lang.Thread.getName()

Le tableau ci-dessous fournit des informations sur les noms d'unité d'exécution IBM WebSphere Real Time for RT Linux.

Tableau 12. Noms d'unités d'exécution dans IBM WebSphere Real Time for RT Linux

Détail de l'unité d'exécution	Nom de l'unité d'exécution
Unité d'exécution JVM interne utilisée par le module de récupération de place pour distribuer la finalisation des objets par le biais des unités d'exécution secondaires.	`Méthode principale de finaliseur`
Unité d'exécution d'alarme utilisée par le récupérateur de place.	`Alarme GC`
Unités d'exécution esclaves utilisées pour la récupération de place.	`Esclave GC`
Unité d'exécution JVM interne utilisée par le module de compilation JIT (just-in-time) pour échantillonner l'utilisation des méthodes dans l'application.	`Echantillonneur JIT`
Unité d'exécution utilisée par la machine virtuelle pour gérer les signaux envoyés par l'application en externe ou en interne.	`Indicateur de signaux`

Les noms par défaut des unités d'exécution temps réel (javax.realtime.RealtimeThread) créées par le code Java ont la forme `RTThread-x` où «x» est le numéro de l'unité d'exécution.

Les noms par défaut des unités d'exécution temps réel sans accès au segment de mémoire ont la forme `NHRTThread-x`, où «x» est le numéro de l'unité d'exécution.

La priorité de l'unité d'exécution Java est associée à une valeur de priorité du système d'exploitation en fonction de la plateforme. Une valeur élevée pour la priorité de l'unité d'exécution Java indique que l'unité d'exécution a une priorité élevée. En d'autres termes, l'unité d'exécution s'exécute plus fréquemment que celles ayant une priorité plus basse. Pour plus d'informations sur le fonctionnement pour les unités d'exécution Java, les unités d'exécution temps réel et les unités d'exécution temps réel sans accès au segment de mémoire, voir «Mappage et héritage des priorités», à la page 12.

Les valeurs d'état peuvent être :

- R - Runnable : l'unité d'exécution peut être exécutée lorsqu'elle le peut.
- CW - Condition Wait : l'unité d'exécution attend, parce que, par exemple :
  - Un appel sleep() est émis.
  - L'unité d'exécution a été bloquée pour E-S.
  - Une méthode wait() est appelée pour attendre la modification d'un moniteur.
  - L'unité d'exécution se synchronise avec une autre avec un appel join().
- S – Suspended : l'unité d'exécution a été suspendue par une autre.
- Z – Zombie : l'unité d'exécution a été arrêtée.
- P – Parked : l'unité d'exécution a été parquée par la nouvelle API de concurrence (java.util.concurrent).
- B – Blocked : l'unité d'exécution attend d'obtenir un verrou détenu par un autre élément.

Si une unité d'exécution est à l'état parked ou blocked, la sortie contient une ligne pour cette unité d'exécution ; elle commence par `3XMTHREADBLOCK` et indique la ressource attendue par l'unité d'exécution et, si possible, l'unité d'exécution à laquelle cette ressource appartient. Pour plus d'informations, voir la rubrique relative aux unités d'exécution de type blocked dans le guide d'utilisation d'IBM SDK for Java 7.

Lorsque vous générez un Javadump pour obtenir des informations de diagnostic, la machine JVM met au repos les unités d'exécution Java avant de produire le javacore. L'état de préparation `exclusive_vm_access` est indiqué dans la ligne `1TIPREPSTATE` de la section `TITLE`.

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

Les unités d'exécution qui exécutaient du code Java lors du javacore ont l'état CW (Condition Wait).

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

La section javacore LOCKS montre que ces unités d'exécution attendent sur une horloge JVM interne.

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

Utilisation de Heapdump

Le Heapdump (cliché des tas) décrit le mécanisme IBM Virtual Machine for Java qui génère un vidage de tous les objets actifs qui se trouvent dans le segment de mémoire Java, à savoir ceux utilisés par l'application Java en cours d'exécution.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur les clichés de tas :

- Obtention des clichés de tas
- Outils de traitement des clichés de tas
- Utilisation de **-Xverbose:gc** pour obtenir des informations sur un segment de mémoire
- Variables d'environnement et cliché de tas
- Format de fichier texte Heapdump (classique)
- Format de fichier PHD (Portable Heap Dump)

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using Heapdump.

Informations supplémentaires relatives à IBM WebSphere Real Time for RT Linux :

Activation de plusieurs clichés de tas pour les machines JVM temps réel

Le cliché de tas généré par défaut est un fichier qui contient des informations sur tous les objets Java de toutes les zones de mémoire, la mémoire de segment, la mémoire pérenne et la mémoire sectorisée. La production de plusieurs vidages se justifie principalement pour analyser chaque zone de segment de mémoire en utilisant les outils Heapdump traditionnels sans modification.

Pourquoi et quand exécuter cette tâche

Par défaut, les clichés de tas contiennent des informations sur tous les objets dans les zones de mémoire, le segment de mémoire, la mémoire permanente et la mémoire sectorisée. Vous pouvez obtenir des clichés de tas séparés contenant des informations sur les objets Java dans chaque zone de mémoire en utilisant l'option **request=multiple** avec **-Xdump:heap**. Notez que vous devez répéter les paramètres par défaut de l'option de demande également et que vous devez donc spécifier `request=multiple+exclusive+prepwalk+compact`. Vous produisez ainsi un groupe de clichés de tas avec une zone supplémentaire dans le nom indiquant la zone de mémoire :

```
heapdump.%id.%Y%m%d.%H%M%S.%pid.phd
```

, où *%id* identifie le fichier heapdump qui contient des objets dans la mémoire du segment la mémoire pérenne ou une zone donnée de la mémoire sectorisée.

Il existe quatre types de segments de mémoire représentés par les noms suivants : «Par défaut», «Pérenne», «Secteur» et «Autre». Le code Heapdump remplace %id dans le libellé de segment de mémoire par l'un de ces noms concaténés avec un identificateur (généralement numérique). Par exemple, `heapdump.Immortal12994208.20060807.093653.7684.txt.`

Exemple

```
java -Xrealtime -Xdump:heap:defaults:request=multiple+exclusive+compact+prepwalk
<java program>
```

Cette option supplémentaire génère plusieurs clichés de tas dans le format portable Heapdump (phd).

```
java -Xrealtime -Xdump:heap:defaults:request=multiple+exclusive+compact+prepwalk,
opts=CLASSIC
<java program>
```

Cette option supplémentaire génère plusieurs clichés de tas dans le format texte CLASSIC.

L'option -Xdump:what affiche les agents de vidage au démarrage de la machine JVM ; elle est utile pour vérifier les options de vidage disponibles.

Format de fichier texte Heapdump (classique)

Le cliché de tas (Heapdump) texte ou classique correspond à la liste de toutes les instances d'objets dans le segment de mémoire, y compris le type d'objet, sa taille et les références entre les objets.

Enregistrement d'en-tête

L'enregistrement d'en-tête est un enregistrement qui contient une chaîne d'informations de version.

```
// Version: <chaîne contenant le niveau SDK,
la plateforme et le niveau de version JVM>
```

Exemple :

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

Enregistrements d'objet

Les enregistrements d'objet sont plusieurs enregistrements, un pour chaque instance d'objet dans le segment de mémoire, fournissant l'adresse, la taille, le type et les références de l'objet.

```
<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>...
```

L'adresse de l'objet et les références de segment de mémoire se trouvent dans le segment de mémoire, mais l'adresse de bloc de classe est en dehors du segment. Toutes les références dans l'instance d'objet sont listées, y compris celles qui ont des valeurs NULL. Le type d'objet est soit un nom de classe contenant le package ou un tableau de primitives soit un type de tableau de classes indiqué par sa signature de type JVM standard (voir «Signatures de type VM Java», à la page 128). Les enregistrements d'objet peuvent également contenir des références de bloc de classe supplémentaires, généralement, dans le cas des instances de classe de réflexion.

Exemples :

Une instance d'objet de 8 octets de longueur de type java/lang/String :

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

Une adresse de bloc de classe java/lang/String, suivie d'une référence à une instance de tableau de type caractère :

```
0x415319D8 0x00436EB0
```

Une instance d'objet de 44 octets de longueur de type tableau de caractères :

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

Une adresse de bloc de classe de type tableau de caractères :

```
0x41530F20
```

Un objet de type tableau de classe interne java/util/Hashtable Entry :

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

Un objet de type classe interne java/util/Hashtable Entry :

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

Une adresse de bloc de classe et références de segment de mémoire, y compris les références null :

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

Enregistrements de classe

Les enregistrements de classe sont plusieurs enregistrements, un pour chaque classe chargée, fournissant l'adresse de bloc de classe, la taille, le type et les références 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'adresse de bloc de classe et les références de bloc de classe se trouvent en dehors du segment de mémoire, mais l'enregistrement de classe peut également contenir des références dans le segment de mémoire, généralement pour les membres de données de classe statiques. Toutes les références dans le bloc de classe sont listées, y compris celles qui ont des valeurs null. Le type de classe est soit un nom de classe contenant le package ou un tableau de primitives soit un type de tableau de classe indiquée par sa signature de type JVM standard (voir «Signatures de type VM Java», à la page 128).

Exemples :

Un bloc de classe de 32 octets pour la classe java/lang/Runnable:

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

Références à d'autres blocs de classe et références de segment de mémoire, y compris les références null :

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

Un bloc de classe de 168 octets pour 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
```

Enregistrement de fin 1

L'enregistrement de fin 1 est un enregistrement contenant des nombres d'enregistrements.

```
// 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
```

Enregistrement de fin 2

L'enregistrement de fin 2 est un enregistrement contenant des totaux.

```
// 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 type VM Java

Les signatures de type VM Java sont des abréviations des types Java, comme indiqué dans le tableau suivant :

Signatures de type VM Java	Type Java
Z	Booléen
S	Octet
G	Caractère
S	Court
I	Entier
J	Long
F	Flottant
D	Double
L <*classe qualifiée complète*> ;	<*classe qualifiée complète*>
[<*type*>	<*type*>[ ] (array of <*type*>)
(<*arg-types*>) <*ret-type*>	méthode

Utilisation des vidages système et de l'afficheur des vidages système

La machine virtuelle Java peut générer des vidages système natifs, appelés également cliché de processus, dans des cas configurables. les vidages système sont généralement volumineux. La plupart des outils utilisés pour analyser ces systèmes sont également spécifiques de la plateforme.Utilisez l'outil **gdb** pour analyser un vidage système sous Linux.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur l'utilisation des vidages système et de l'afficheur des vidages systèmes :

- Présentation des vidages système
- Valeurs par défaut de vidage système
- Utilisation de l'afficheur des vidages
  - Utilisation de **jextract**
  - Problèmes à résoudre avec l'afficheur des vidages
  - Commandes disponibles dans **jdmpview**
  - Exemple de session
  - Référence rapide des commandes **jdmpview**

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using system dumps and the dump viewer.

Informations supplémentaires relatives à IBM WebSphere Real Time for RT Linux :

Utilisation de jextract

Lors du traitement d'un vidage système à partir d'une machine virtuelle Java en temps réel vous devez inclure l'option **-Xrealtime**. Par exemple :

```
jextract -Xrealtime <nom fichier core>
[<fichier_zip>]
```

Lorsque vous exécutez **jextract** sur une machine virtuelle Java différente de celle pour laquelle le vidage a été généré, les messages d'erreur suivants s'affichent :

```
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).
```

Ce message est également généré si Java était exécuté avec la machine virtuelle Java standard avec l'option **-Xrealtime** lors du traitement du vidage avec **jextract**.

Commandes disponibles dans jdmpview

jdmpview est un outil de ligne de commande interactif qui permet d'explorer les informations d'un vidage système JVM et d'exécuter diverses fonctions d'analyse.

info jitm
Affiche les méthodes compilées AOT et JIT et leurs adresses :
- Nom et signature de la méthode
- Adresse de début de la méthode
- Adresse de fin de la méthode

Pour toutes les autres options de commande, voir le guide d'utilisation d'IBM SDK for Java 7.

Traçage des applications Java et la machine virtuelle Java

La trace JVM est une fonction de trace fournie dans IBM WebSphere Real Time for RT Linux qui a un impact limité sur les performances. Dans la plupart des cas, les données de trace ont un format binaire compressé qui peut être formaté avec le formateur Java fourni.

Le traçage et activé par défaut avec un petit groupe de points de trace placés dans des mémoires tampon. Vous pouvez activer les points de trace lors de l'exécution

en utilisant des niveaux, des composants, des noms de groupe ou des identificateurs de point de trace individuels.

Le guide d'utilisation IBM SDK for Java 7 contient des informations détaillées sur le traçage des applications :

- Eléments qui peuvent être tracés
- Types de points de trace
- Traçage par défaut
- Enregistrement des données de trace
- Contrôle de la trace
- Traçage des applications Java
- Traçage des méthodes Java

Lors du traçage d'IBM WebSphere Real Time for RT Linux vous devez appeler correctement la machine virtuelle Java en temps réel lorsque vous incluez les options de trace. Par exemple, lorsque vous indiquez des options de trace, tapez :

```
java -Xrealtime -Xtrace:<options>
```

Pour les informations relatives à IBM SDK for Java 7, voir : Trace des applications Java et de la machine JVM.

Détermination des problèmes JIT et AOT

Utilisez les options de ligne de commande pour déterminer les problèmes des compilateurs JIT et AOT et optimiser les performances.

Bien qu'IBM WebSphere Real Time for RT Linux partage certains composants avec IBM SDK for Java 7, JIT et AOT n'ont pas le même comportement. Cette section traite de l'identification et de la résolution des problèmes pour JIT et AOT sur IBM WebSphere Real Time for RT Linux.

Diagnostic d'un problème JIT ou AOT

Parfois, la compilation de bytecodes valides peut générer du code natif non valide et une erreur du programme Java. En déterminant si le compilateur JIT ou AOT est à l'origine du problème, et si c'est le cas, l'*emplacement* de l'erreur, vous fournissez des informations précieuses au service Java.

Pourquoi et quand exécuter cette tâche

Pour déterminer les méthodes compilées lorsque le cache des classes partagée est rempli, utilisez l'option **-Xaot:verbose** sur la ligne de commande admincache. Par exemple :

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

Cette section explique comment déterminer si le problème est lié au compilateur. Cette section propose également des solutions de contournement et des techniques de débogage pour résoudre les problèmes liés au compilateur.

Désactivation du compilateur JIT ou AOT :

Si vous pensez qu'un problème se produit dans le compilateur JIT ou AOT, désactivez la compilation pour déterminer si le problème persiste. Si le problème apparaît, vous en déduisez que le compilateur n'est pas fautif.

Pourquoi et quand exécuter cette tâche

Le compilateur JIT est actif par défaut. Le compilateur AOT est également activé, mais il ne l'est pas si les classes partagées n'ont pas été activées. Pour plus d'efficacité, les méthodes d'une application Java ne sont pas toutes compilées. La machine JVM gère un nombre d'appels pour chaque méthode dans l'application ; chaque fois qu'une méthode est appelée et interprétée, le nombre d'appels de la méthodes augmente. Lorsque le nombre atteint le seuil de compilation, la méthode est compilée et exécutée en natif.

Le mécanisme de comptage du nombre d'appels étend la compilation des méthodes à toute la vie de l'application en affectant une priorité élevée aux méthodes fréquemment utilisées. Certaines méthodes peu utilisées peuvent ne jamais être compilées. Par conséquent, lorsqu'un programme Java échoue, le problème peut se situer dans le compilateur JIT ou AOT ou autre part dans la machine JVM.

La première étape de détermination de l'échec consiste à déterminer l'*emplacement* du problème. Pour ce faire, vous devez d'abord exécuter le programme Java en mode d'interprétation pur (à savoir, avec les compilateurs JIT et AOT désactivés).

Procédure

1. Supprimez les options **-Xjit** et **-Xaot** (les paramètres associés) depuis la ligne de commande.
2. Utilisez l'option de ligne de commande **-Xint** pour désactiver les compilateurs JIT et AOT. Pour des raisons de performances, n'utilisez pas l'option **-Xint** dans un environnement de production.

Que faire ensuite

L'exécution du programme Java avec la compilation désactivé a l'une des conséquences suivantes :

- L'échec persiste. Le problème ne se trouve pas dans le compilateur JIT ou AOT. Dans certains cas, le programme peut échouer d'une autre manière, mais le problème n'est pas lié au compilateur.
- Le problème disparaît. Il est fort problème qu'il réside dans le compilateur JIT ou AOT.

  Si vous n'utilisez pas de classes partagées, le compilateur JIT est fautif. Si vous en utilisez, vous devez déterminer le compilateur fautif en exécutant l'application en activant seulement la compilation JIT. Exécutez l'application avec l'option **-Xnoaot** à la place de l'option **-Xint**. La conséquence est la suivante :
  - L'échec persiste. Le problème se trouve dans le compilateur JIT. Vous pouvez également utiliser l'option **-Xnojit** au lieu de l'option **-Xnoaot** pour vérifier que seul le compilateur JIT est fautif.
  - Le problème disparaît. Le problème se trouve dans le compilateur AOT.

Désactivation sélective du compilateur JIT :

Si l'échec du programme Java provient d'un problème du compilateur JIT, vous pouvez essayer mieux cerner le problème.

Pourquoi et quand exécuter cette tâche

Par défaut, le compilateur JIT optimise les méthodes à différents niveaux d'optimisation, c'est-à-dire que différentes sélections d'optimisation sont appliquées à différentes méthodes en fonction des nombres d'appels. Les méthodes appelées plus fréquemment sont optimisées à des niveaux supérieurs. En changeant les paramètres du compilateur JIT, vous pouvez contrôler le niveau d'optimisation des méthodes et déterminer si l'optimiseur est fautif et l'optimisation problématique dans ce cas.

Vous pouvez définir les paramètres JIT sous la forme d'une liste séparées des virgules ajoutée à l'option **-Xjit**. La syntaxe est **-Xjit**:<*param1*>,<*param2*>=<*value*>. Par exemple :

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

exécute le programme HelloWorld, active la sortie prolixe de JIT et fait que JIT génère du code natif JIT sans exécuter des optimisations.

Procédez comme suit pour déterminer la partie du compilateur qui génère l'erreur :

Procédure

1. Définissez le paramètre JIT **count=0** pour paramétrer le seuil de compilation sur la valeur 0. Ainsi, ce paramètre provoque la compilation de chaque méthode Java avant son exécution. Utilisez **count=0** uniquement pour diagnostiquer les problèmes, car un nombre très important de méthodes sont compilées, notamment des méthodes rarement utilisées. La compilation supplémentaire utilise davantage de ressources de traitement et ralentit votre application. Avec **count=0**, l'application doit échouer immédiatement lorsque la zone fautive est atteinte. Dans certains cas, l'utilisation de **count=1** peut reproduire l'échec de manière plus fiable.
2. Ajoutez **disableInlining** aux paramètres du compilateur JIT. **disableInlining** désactive la génération d'un code plus important et plus complexe. Si le problème disparaît, utilisez **-Xjit:disableInlining** comme solution palliative pendant que le service Java analyse et résout le problème de compilation.
3. Diminuez les niveaux d'optimisation en ajoutant le paramètre **optLevel** et réexécutez le programme jusqu'à ce que le problème disparaisse ou vous atteigniez le niveau «noOpt». Pour un problème de compilation JIT, démarrez avec «scorching» et descendez dans la liste. Les niveaux d'optimisation sont en ordre décroissant :
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold
   f. noOpt

Que faire ensuite

Si l'un de ces paramètres fait disparaître le problème, vous disposez d'une solution de contournement que vous pouvez utiliser. Cette solution est provisoire et peut être utilisée pendant que le service Java analyse et résout le problème de compilation. Si la suppression de **disableInlining** de la liste des paramètres JIT empêche le problème de réapparaître, supprimez-le pour améliorer les

performances. Suivez les instructions dans «Recherche de la méthode défaillante» pour améliorer les performances de la solution de contournement.

Si le problème persiste au niveau de l'optimisation «noOpt» une solution palliative consiste à désactiver le compilateur JIT.

Recherche de la méthode défaillante :

Lorsque vous avez déterminé le niveau d'optimisation le plus bas auquel le compilateur JIT ou AOT doit compiler les méthodes pour déclencher l'échec, vous pouvez identifier la partie du programme Java, qui lorsqu'elle est compilée, génère l'échec. Vous pouvez ensuite indiquer au compilateur de limiter la solution de contournement à une méthode, une classe ou un package pour permettre au compilateur de compiler le reste du programme normalement. Pour les échecs du compilateur JIT, si l'échec se produit avec **-Xjit:optLevel=noOpt**, vous pouvez indiquer au compilateur de ne pas compiler la méthode ou les méthodes qui génèrent l'échec.

Avant de commencer

Si une sortie d'erreur similaire à celle ci-dessous s'affiche, vous pouvez l'utiliser pour identifier la méthode défaillante :

```
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 lignes importantes sont les suivantes :

vmState=0x00000000
Indique que le code erroné ne se trouve pas dans le code d'exécution JVM.

Module= ou Module_base_address=
Ne figure pas dans la sortie (peut être vide ou affecté de la valeur 0), car le code a été compilé par le compilateur JIT et en dehors de la DLL ou bibliothèque.

Compiled_method=
Indique la méthode Java pour laquelle le code compilé a été généré.

Pourquoi et quand exécuter cette tâche

Si la sortie n'indique pas la méthode défaillante, suivez ces étapes pour l'identifier :

Procédure

1. Exécutez le programme Java en ajoutant les paramètres JIT **verbose** et **vlog=***<filename>* à l'option **-Xjit** ou **-Xaot**. Avec ces paramètres, le compilateur liste les méthodes compilées dans le fichier journal *<filename>.<date>.<time>.<pid>* également appelé *fichier de limites*. Un fichier de limites contient des lignes qui correspondent aux méthodes compilées, telles que :

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

Les lignes qui ne commencent pas par le signe Plus sont ignorées par le compilateur dans les étapes suivantes et vous pouvez les supprimer du fichier. Les méthodes pour lesquelles du code AOT est chargé depuis le cache des classes partagées commencent par `+ (AOT load)`.

2. Exécutez de nouveau le programme avec le paramètre JIT ou AOT **limitFile=**(<*nom fichier*>,<*m*>,<*n*>), où <*nom fichier*> est le chemin du fichier de limites et <*m*> et <*n*> sont des numéros de ligne indiquant la première et la dernière méthodes du fichier de limites à compiler. Le compilateur compile uniquement les méthodes listées sur les lignes <*m*> à <*n*> dans le fichier de limites. Les méthodes qui ne figurent pas dans le fichier de limites et les méthodes figurant sur les lignes en dehors de la plage ne sont pas compilées et aucun code AOT du cache des données partagées de ces méthodes n'est chargé. Si le programme ne génère plus d'erreur, cela implique qu'une ou plusieurs méthodes que vous avez supprimées dans la dernière itération est probablement à l'origine de l'erreur.
3. Répétez cette procédure en utilisant des valeurs différentes pour <*m*> et <*n*>, autant de fois que nécessaire, pour rechercher le groupe de méthodes minimum qui doivent être compilées pour déclencher l'échec. En divisant par deux le nombre de lignes sélectionnées chaque fois, vous pouvez exécuter une recherche binaire pour la méthode défaillante. En général, vous pouvez réduire le fichier à une seule ligne.

Que faire ensuite

Après avoir localisé la méthode défaillante, vous pouvez désactiver le compilateur JIT ou AOT pour la méthode défaillante uniquement. Par exemple, si la méthode java/lang/Math.max(II)I génère une erreur du programme lorsqu'elle est compilé par JIT avec **optLevel=hot**, vous pouvez exécuter le programme avec :

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

pour compiler uniquement la méthode défaillante au niveau d'optimisation «warm», mais compiler toutes les autres méthodes normalement.

Si une méthode échoue lorsqu'elle est compilée par JIT avec «noOpt», vous pouvez l'exclure de la compilation en utilisant le paramètre **exclude**={<*method*>} :

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

Si une méthode provoque l'échec du programme lorsque le code AOT est chargé depuis le cache des données partagées, excluez la méthode de la chargement AOT en utilisant le paramètre **exclude**={<*method*>} :

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

Les méthodes AOT sont compilées uniquement dans le cache des classes partagées au cours du remplissage **admincache**. Le blocage du chargement AOT est la meilleure des méthodes pour résoudre les problèmes avec ces méthodes.

Identification des échecs de compilation JIT et AOT :

Pour les échecs de compilation JIT, analysez la sortie des erreurs pour déterminer si l'échec se produit lorsque le compilateur JIT tente de compiler une méthode.

Si la machine JVM tombe en panne et si vous constatez que l'échec s'est produit dans la bibliothèque JIT (`libj9jit26.so`), il se peut que le compilateur JIT ait généré une erreur lors de la tentative de compilation d'une méthode.

Si l'erreur suivante s'affiche, utilisez-la pour identifier la méthode ayant échoué :

```
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 lignes importantes sont :

vmState=0x00050000
Indique que le compilateur JIT compile du code. Pour obtenir la liste des numéros de code vmState, voir le tableau des balises de vidage Javadump dans le guide d'utilisation d'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
Indique qu'une erreur s'est produite dans `libj9jit26.so`, le module du compilateur JIT.

Method_being_compiled=
Indique la méthode Java compilée.

Si la sortie n'indique pas la méthode qui échoue, utilisez l'option **verbose** avec les paramètres supplémentaires suivants :

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

Ces paramètres **verbose** indiquent quand le compilateur JIT ou AOT commence à compiler une méthode et quand il a terminé. Si le compilateur JIT ou AOT échoue sur une méthode (il démarre la compilation, mais échoue), utilisez le paramètre **exclude** pour l'exclure de la compilation JIT ou AOT(voir «Recherche de la méthode défaillante», à la page 133). Pour les problèmes de compilation AOT, détruisez le cache de classes partagées avant d'utiliser l'option **exclude**. Si l'exclusion de la méthode empêche la panne, vous disposez d'une solution de contournement que vous pouvez utiliser pendant que le support résout le problème.

Identification des échecs de compilation AOT en mode non-temps réel :

La détermination des problèmes AOT en mode non-temps réel est similaire à celle des problèmes JIT.

Pourquoi et quand exécuter cette tâche

Comme avec le compilateur JIT, exécutez d'abord l'application avec **-Xnoaot** qui permet de vérifier que le code compilé AOT n'est pas utilisé lors de l'exécution de l'application.

Si cela permet de résoudre le problème, recréez les fichier JAR AOT en utilisant la technique décrite dans «Recherche de la méthode défaillante», à la page 133, en

indiquant l'option **-Xaot** au moment de la génération AOT et pas au moment de l'exécution de l'application.

Identification des échecs de compilation AOT en mode temps réel :

La détermination des problèmes AOT utilise l'outil admincache pour identifier un problème.

Pourquoi et quand exécuter cette tâche

Les échecs de compilation AOT se produisent lors du remplissage admincache, contrairement aux échecs de compilation JIT qui apparaissent lors de l'exécution.

Pour identifier l'emplacement d'un problème, exécutez l'outil admincache avec l'option **-Xnoaot**. Ainsi, l'application ne s'exécute pas avec du code compilé AOT (ahead-of-time).

Si l'option **-Xnoaot** résout le problème, examinez la sortie de l'erreur d'origine. La sortie fournit des informations qui identifient la méthode à l'origine du problème. Recherchez une ligne similaire à :

```
Method_being_compiled=myAppClass.main(Ljava/lang/String;)V
```

Pour éliminer ce problème, excluez la méthode de la compilation AOT. Pour ce faire, ajoutez la ligne de commande admincache comme suit :

```
-Xaot:exclude={myAppClass.main(Ljava/lang/String;)V}
```

L'exclusion élimine la compilation AOT de la méthode problématique.

Performances des applications à courte exécution

Le compilateur JIT IBM est optimisé pour les applications à longue exécution généralement utilisées sur un serveur. Vous pouvez utiliser l'option de ligne de commande **-Xquickstart** >en mode non-temps réel pour améliorer les performances des applications à exécution courte, notamment celles dans lesquelles le traitement n'est pas concentré dans un petit nombre de méthodes.

-Xquickstart force le compilateur JIT à utiliser un niveau d'optimisation bas par défaut et à compiler moins de méthodes. L'exécution d'un plus petit nombre de compilations plus rapidement peut accélérer le démarrage des applications. Lorsque le compilateur AOT est actif (les classes partagées et la compilation AOT sont activées), **-Xquickstart** force toutes les méthodes sélectionnées pour la compilation à être compilées par le compilateur AOT, ce qui accélère le démarrage des exécutions suivantes. **-Xquickstart** peut affecter les performances si l'option est utilisée avec des applications à exécution longue qui contiennent des méthodes qui utilisent une grande quantité de ressources de traitement. L'implémentation de **-Xquickstart** peut être modifié dans les prochaines versions.

Vous pouvez également essayer d'améliorer les temps de démarrage en ajustant le seuil JIT (en utilisant trial et error). Voir «Désactivation sélective du compilateur JIT», à la page 131 pour plus d'informations.

-Xquickstart n'a pas d'impact sur l'utilisation du code AOT avec **-Xrealtime**.

Comportement de la machine JVM au cours des périodes d'inactivité

Vous pouvez réduire les cycles UC consommés par une machine JVM inactive en utilisant l'option **-XsamplingExpirationTime** pour désactiver l'unité d'exécution d'échantillonnage JIT.

L'unité d'exécution d'échantillonnage JIT profile l'application active Java pour découvrir les méthodes communément utilisées. L'utilisation de la mémoire et du processeur de l'unité d'exécution d'échantillonnage est négligeable et la fréquence de profilage est automatiquement réduite lorsque la machine JVM est inactive.

Dans certains cas, il peut arriver que vous ne vouliez pas qu'une machine JVM inactive consomme des cycles UC. Pour ce faire, définissez l'option **-XsamplingExpirationTime**<*time*>. Affectez à <*time*> le nombre de secondes correspondant à la durée d'exécution de l'unité d'exécution d'échantillonnage. Utilisez cette option avec précaution, car une fois l'unité d'exécution d'échantillonnage désactivée, vous ne pouvez pas réactiver cette dernière. Autorisez l'unité d'exécution d'échantillonnage à s'exécuter suffisamment longtemps pour identifier les optimisations importantes.

Le collecteur de diagnostics

Le collecteur des diagnostics collecte les fichiers de diagnostic Java d'un événement de problème.

La collecte des fichiers requis par le service IBM peut accélérer le traitement des problèmes signalés. Le guide d'utilisation IBM SDK for Java 7 contient des informations détaillées sur l'utilisation du collecteur de diagnostics.

Vous trouverez ces informations ici : IBM SDK for Java 7 - The Diagnostics Collector.

Diagnostics du récupérateur de place

Cette section explique comment identifier et résoudre les problèmes de récupération de place.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur l'identification et la résolution des problèmes de récupération de place :

- Consignation prolixe de la récupération de place
- Suivi de la récupération de place à l'aide de **-Xtgc**

Vous trouverez ces informations ici : IBM SDK for Java 7 - Garbage Collector diagnostics.

Vous trouverez des informations supplémentaires sur le récupérateur de place Metronome IBM WebSphere Real Time for RT Linux dans les sections suivantes.

Identification et résolution des incidents du récupérateur de place Metronome

En utilisant des options de ligne de commande, vous pouvez contrôler la fréquence de la récupération de place Metronome, les exceptions de manque de mémoire et le comportement de Metronome dans des appels système explicites.

Utilisation des informations verbose:gc :

Vous pouvez utiliser l'option **-verbose:gc** avec l'option **-Xgc:verboseGCCycleTime=N** pour écrire des informations sur la console concernant l'activité du récupérateur de place Metronome. Les propriétés XML dans la sortie **-verbose:gc** de la machine JMV standard ne sont pas toutes créées ou appliquées dans la sortie de récupérateur de place Metronome.

Utilisez l'option **-verbose:gc** pour afficher la quantité de mémoire minimale, maximale et l'espace libre moyen dans le segment de mémoire. De cette manière, vous pouvez vérifier le niveau d'activité et d'utilisation du segment de mémoire et ajuster les valeurs de manière appropriée. L'option **-verbose:gc** écrit des statistiques Metronome sur la console.

L'option **-Xgc:verboseGCCycleTime=N** contrôle la fréquence d'extraction des informations. Elle détermine le délai en millisecondes de vidage des résumés. La valeur par défaut de N est 1 000 millisecondes. La durée du cycle n'implique pas que le résumé soit vidé précisément à ce moment là, mais lorsque l'événement de récupération de place respecte ce critère de temps. La récupération et l'affichage de ces statistiques peuvent affecter les délais d'interruption du récupérateur de place, qui s'accroissent lorsque N diminue.

Un quantum est une période unique d'activité récupérateur de place Metronome qui génère une interruption ou une pause pour une application.

Exemple de sortie verbose:gc

Entrez :

```
java -Xrealtime -verbose:gc -Xgc:verboseGCCycleTime=N myApplication
```

Cet exemple montre une sortie initiale verbose:gc qui contient la version et les paramètres de récupération de place :

```
<verbosegc
xmlns="http://www.ibm.com/j9/verbosegc" version="R26_Java726_GA_20110716_0946_B87065">

<initialized id="1" timestamp="2011-07-27T14:17:52.277">
  <attribute name="gcPolicy" value="-Xgcpolicy:metronome" />
  <attribute name="maxHeapSize" value="0x5800000"/>
  <attribute name="initialHeapSize" value="0x4000000"/>
  <attribute name="compressedRefs" value="false"/>
  <attribute name="pageSize" value="0x1000"/>
  <attribute name="requestedPageSize" value="0x1000"/>
  <attribute name="gcthreads" value="1" />
  <region>
    <attribute name="regionSize" value="16384"/>
    <attribute name="regionCount" value="4096"/>
    <attribute name="arrayletLeafSize" value="2048"/>
  </region>
  <metronome>
    <attribute name="beatsPerMeasure" value="500" />
    <attribute name="timeInterval" value="10000" />
    <attribute name="targetUtilization" value="70"/>
    <attribute name="trigger" value="0x2000000"/>
    <attribute name="headRoom" value="0x100000" />
  </metronome>
  <system>
    <attribute name="physicalMemory" value="12507463680"/>
    <attribute name="numCPUs" value="8"/>
    <attribute name="architecture" value="x86"/>
    <attribute name="os" value="Linux"/>
    <attribute name="osVersion" value="2.6.24.7-75ibmrt2.18"/>
  </system>
  <vmargs>
```

```
    <vmarg
name="-Xoptionsfile=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime/options.default"/>
    <vmarg name="-Xjcl:jclse7b_26"/>
    <vmarg
name="-Dcom.ibm.oti.vm.bootstrap.library.path=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime:/
my_dir/pxi3270hrt-2011071..."/>
    <vmarg
name="-Dsun.boot.library.path=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime:/my_dir/
pxi3270hrt-20110719_02/sdk/jre/lib..."/>
    <vmarg
name="-Djava.library.path=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/i386/realtime:/my_dir/
pxi3270hrt-20110719_02/sdk/jre/lib/i38..."/>
    <vmarg name="-Djava.home=/my_dir/pxi3270hrt-20110719_02/sdk/jre"/>
    <vmarg name="-Djava.ext.dirs=/my_dir/pxi3270hrt-20110719_02/sdk/jre/lib/ext"/>
    <vmarg name="-Duser.dir=/my_dir/pxi3270hrt-20110719_02/sdk/jre/bin"/>
    <vmarg name="_j2se_j9=1120000"
value="F76FF700"/>
    <vmarg name="-Djava.runtime.version=pxi3270hrt-20110719_02"/>
    <vmarg name="-Djava.class.path=."/>
    <vmarg name="-Xrealtime"/>
    <vmarg name="-verbose:gc"/>
    <vmarg name="-Dsun.java.launcher=SUN_STANDARD"/>
    <vmarg name="-Dsun.java.launcher.pid=5543"/>
    <vmarg name="_port_library" value="F7701B80"/>
    <vmarg name="_bfu_java" value="F77029A8"/>
    <vmarg name="_org.apache.harmony.vmi.portlib" value="08051DA0"/>
  </vmargs>
</initialized>
```

Lorsque la récupération de place démarre, un événement `trigger start` se produit, suivi de n'importe quel nombre d'événements `heartbeat`, puis d'un événement `trigger end` lorsque le déclenchement est satisfait. Cet exemple montre un cycle de récupération de place déclenché comme sortie 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" />
```

Les types d'événements suivants se produisent :

<trigger-start ...>
Début d'un cycle de récupération de place lorsque la quantité de mémoire utilisée est supérieure au seuil de déclenchement. Le seuil par défaut est 50 % du segment de mémoire. L'attribut `intervalms` est l'intervalle entre l'événement `trigger end` précédent (avec id-1) et cet événement `trigger start`.

<trigger-end ...>
Un cycle de récupération de place a ramené la mémoire utilisée sous le seuil de déclenchement. Si un cycle de récupération de place est terminé et que la mémoire utilisée n'est pas retombée en dessous du seuil de déclenchement un nouveau cycle de récupération de place est démarré avec le même ID de contexte. Pour chaque événement `trigger start` il existe un événement `trigger end` ayant le même ID de contexte. L'attribut `intervalms` attribute est l'intervalle entre l'événement `trigger start` précédent et l'événement `trigger end` en cours. Pendant ce temps, un ou plusieurs cycles de récupération de place auront été exécutés jusqu'à ce que la mémoire utilisée retombe en dessous du seuil de déclenchement.

<gc-op id="28" type="heartbeat"...>
Evénement périodique qui collecte des informations (sur la mémoire et le délai) sur tous les quanta de récupération de place pour la période couverte. Un événement de récupération de place peut se produire uniquement entre une paire d'événements `trigger start` et `trigger end` correspondants alors qu'un cycle de récupération de place est en cours. L'attribut `intervalms` est l'intervalle entre l'événement de récupération de place précédent (avec `id` -1) et cet événement de récupération de place.

<gc-op id="29" type="syncgc"...>
Evénement de récupération de place asynchrone (non déterministe). Voir «Récupérations de place synchrones», à la page 141

Les balises XML dans cet exemple ont la signification suivante :

<quanta ...>
Résumé de la durée de pause de quantum au cours de l'intervalle de signalisation de présence incluant la longueur des pauses en millisecondes.

<free-mem type="heap" ...>
Résumé de la quantité d'espace de segment libre au cours de l'intervalle de signalisation de présence, échantillonnée à la fin de chaque quantum de récupération de place.

<classunload-info classloadersunloaded=9 classesunloaded=156 />
Nombre de chargeurs de classes et de classes déchargées au cours de l'intervalle de signalisation de présence.

<references type="weak" cleared="660 />
Nombre et type des objets de référence Java supprimés lors de l'intervalle de signalisation de présence.

Remarque :

- Si un seul quantum de récupération de place se produit dans l'intervalle entre deux signaux de présence, la mémoire libre est échantillonnée uniquement à la fin du quantum. Par conséquent, les quantités minimale, maximale et moyenne données dans le résumé de signal de présence sont toutes égales.

- Il se peut que l'intervalle entre deux événements de signal de présence soit sensiblement plus long que le cycle défini si le segment de mémoire n'est pas suffisamment plein pour nécessiter une activité de récupération de place. Par exemple, si le programme nécessite une récupération de place une seule fois toutes les quelques secondes, un signal de présence n'apparaîtra vraisemblablement qu'une seule fois toutes les quelques secondes.
- Il se peut que l'intervalle puisse être sensiblement plus long que le cycle défini, car la récupération de place n'a aucune opération à exécuter dans un segment de mémoire qui n'est pas suffisamment plein pour garantir la récupération de place. Par exemple, si le programme nécessite une récupération de place une seule fois toutes les quelques secondes, un signal de présence n'apparaîtra vraisemblablement qu'une seule fois toutes les quelques secondes.

  Si un événement, tel qu'une récupération de place synchrone ou une modification de priorité se produit, les informations de l'événement et les événements en attente, tels que les signaux de présence, sont produits immédiatement en sortie.
- Si le quantum maximal de récupération de place pour une période donnée est trop grand, réduisez l'utilisation cible à l'aide de l'option **-Xgc:targetUtilization**. Cette action laisse au récupérateur de place plus de temps pour travailler. Vous pouvez également augmenter la taille du segment de mémoire avec l'option **-Xmx**. De même, si l'application peut tolérer des délais plus longs que ceux qui sont signalés, vous pouvez augmenter l'utilisation cible et réduire la taille de segment.
- La sortie peut être redirigée vers un fichier journal au lieu de la console avec l'option **-Xverbosegclog:<file>**. Par exemple **-Xverbosegclog:out** écrit la sortie **-verbose:gc** dans le fichier *out*.
- La priorité indiquée dans `thread-priority` est la priorité d'unité d'exécution du système d'exploitation sous-jacente et non une priorité d'unité d'exécution Java.

Récupérations de place synchrones

Une entrée est également écrite dans le journal **-verbose:gc** lorsqu'une récupération de place synchrone (non déterministe) se produit. Cet événement a trois causes :

- Appel explicite `System.gc()` dans le code.
- La machine JVM à court de mémoire exécute alors une récupération de place synchrone pour éviter une condition OutOfMemoryError.
- La machine JVM se ferme pendant un récupération de place continue. La machine JVM ne peut pas annuler la collection, par conséquent elle la termine de manière synchrone et se ferme.

Exemple d'entrée `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>
```

Exemple d'entrée de récupération de place synchrone résultant de l'arrêt de la machine 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 balises XML et les attributs dans cet exemple ont la signification suivante :

<gc-op id="9" type="syncgc" timems="6.439" ...
: Cette ligne indique que le type d'événement est une récupération de place synchrone. L'attribut `timems` indique la durée de la récupération de place synchrone en millisecondes.

<syncgc-info reason="..."/>
: Cause de la récupération de place synchrone.

<free-mem-delta.../>
: Segment de mémoire Java en octets avant et après la récupération de place synchrone en octets.

<finalization .../>
: Nombre d'objets en attente de finalisation

<classunload-info .../>
: Nombre de chargeurs de classes et de classes déchargées au cours de l'intervalle de signalisation de présence.

<references type="weak" cleared="53" .../>
: Nombre et type des objets de référence Java supprimés lors de l'intervalle de signalisation de présence.

Une récupération de place synchrone suite à un manque de mémoire ou l'arrêt d'une machine virtuelle peut se produire uniquement lorsque le récupérateur de place est actif. Elle doit être précédée d'un événement `trigger start`, mais pas nécessairement immédiatement. Certains événements de signalisation de présence se produisent entre un événement `trigger start` et l'événement `synchgc`. La récupération de place synchrone provoquée par `System.gc()` peut avoir lieu à tout moment.

Suivi des quanta de récupération de place

Les quanta de récupération de place peuvent être contrôlés en activant les points de trace `GlobalGCStart` et `GlobalGCEnd`. Ces points de trace sont produits au début et à la fin de toute l'activité de récupération de place Metronome, y compris des récupérations de place synchrones. La sortie de ces points de trace se présente comme suit :

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

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

Modifications de priorités

Outre les résumés, une entrée est écrite dans le journal **-verbose:gc** lorsque la priorité de l'unité d'exécution du récupérateur de place change (suite à la modification des priorités par l'application ou de la fin d'une ou de plusieurs unités d'exécution dans une application). La priorité indiquée est la priorité d'unité

d'exécution de système d'exploitation sous-jacente et non pas d'une unité de priorité Java. Exemple d'entrée de modification de priorité d'exécution de récupérateur de place :

```
<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>
```

Les modifications de priorités peuvent être contrôlées en temps réel en produisant les informations de point de trace associées aux priorités d'unité d'exécution du récupérateur de place. Cette sortie se présente comme suit :

```
15:58:25.493*0x8286e00    j9mm.102       - setGCThreadPriority()
called with newGCThreadPriority = 11
```

Cette entrée peut être activée à l'aide de l'ID comme suit :
-Xtrace:iprint=tpnid{j9mm.102}

Entrée de manque de mémoire

Lorsque l'une des zones de mémoire a atteint sa capacité totale, une entrée est écrite dans le journal **-verbose:gc** avant la génération de l'exception OutOfMemoryError. Exemple de sortie :

```
<out-of-memory id="71" timestamp="2011-07-23T08:32:51.435" memorySpaceName="Scoped"
  memorySpaceAddress="080EED9C"/>
```

Par défaut, un vidage Java est généré suite à une exception OutOfMemoryError. Ce vidage contient des informations sur les zones de mémoire utilisées par le programme. Conjointement avec la valeur `J9MemorySpace` figurant dans la sortie **-verbose:gc**, vous pouvez utiliser ces informations dans le vidage pour identifier la zone de mémoire manquant d'espace :

```
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)
```

Dans l'exemple précédent, l'ID d'espace mémoire fourni dans la sortie -verbose:gc (0x080EED9C) peut être mis en corresponce avec l'ID de a zone de mémoire sectorisées du vidage Java. Ceci peut s'avérer utile si vous disposez de plusieurs secteurs et voulez identifier le secteur qui manque de mémoire, car la sortie -verbose:gc indique uniquement si l'erreur OutOfMemoryError s'est produite dans la mémoire pérenne, la mémoire sectorisée ou le segment de mémoire.

Comportement du récupération de place Metronome lors d'un manque de mémoire :

Par défaut, le récupérateur de place Metronome déclenche une récupération de place non déterministe illimitée lorsque la machine JVM manque de mémoire. Pour éviter ce comportement non déterministe, utilisez l'option **-Xgc:noSynchronousGCOnOOM** pour générer une erreur OutOfMemoryError lorsque la machine JVM manque de mémoire.

La récupération de place illimitée par défaut jusqu'à ce que toute la place possible soit récupérée est exécutée en une seule fois. La durée de pause nécessaire correspond généralement à un beaucoup plus grand nombre de millisecondes qu'un quantum incrémentiel Metronome normal.

Information associée

Utilisation de -Xverbose:gc pour analyser les récupérations de place synchrones

Comportement du récupérateur de place Garbage dans les appels System.gc() explicites :

Si un cycle de récupération de place est en cours, la récupérateur de place Metronome termine le cycle de manière synchrone lorsque `System.gc()` est appelé. Si aucun cycle de récupération de place n'est en cours, un cycle complet synchrone est exécuté lorsque `System.gc()` est appelé. Utilisez `System.gc()` pour nettoyer le segment de mémoire d'une manière contrôlée. Il s'agit d'une opération non déterministe, car elle exécute une récupération de place complète avant de prendre fin.

Certaines applications appellent du logiciel ayant des appels System.gc() où il n'est pas acceptable de créer ces retards non déterministes. Pour désactiver tous les appels System.gc(), utilisez l'option **`-Xdisableexplicitgc`**.

La sortie de la récupération de place prolixe d'un appel System.gc() a la raison «Collecte de récupération de place système» et vraisemblablement une longue durée :

```
| <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>
```

Diagnostics de classes partagées

La compréhension de la résolution des problèmes qui peuvent se produire facilite l'utilisation du mode Classes partagées.

Pour la présentation des classes partagées, voir Partage des classes entre les machines virtuelles Java.

Le guide d'utilisation d'IBM SDK for Java 7contient des informations utiles sur l'identification et la résolution des problèmes liés aux classes partagées :

- Déploiement des classes partagées
- Gestion de la modification du bytecode d'exécution
- Description des mises à jour dynamiques
- Utilisation de l'API Java Helper
- Description de la sortie du diagnostic des classes partagées
- Résolution des problèmes avec les classes partagées

Vous trouverez ces informations ici : IBM SDK for Java 7 - Shared classes diagnostics.

Certaines informations du guide d'utilisation d'IBM SDK for Java 7 peuvent ne pas s'appliquer à IBM WebSphere Real Time for RT Linux, dans les situations particulières ci-dessous :

- En mode Temps réel, les applications ont accès aux caches de classes en lecture seule et non en lecture-écriture.
- Les caches peuvent être modifiés exclusivement en utilisant l'outil **admincache**.
- Les caches non permanents ne sont pas disponibles en mode temps réel.

Utilisation de JVMTI

JVMTI est une interface bidirectionnelle qui permet à la machine virtuelle Java et à un agent natif de communiquer. Elle remplace les interfaces JVMDI et JVMPI.

JVMTI permet aux tiers de développer des outils de débogage, de profilage et de contrôle pour la machine virtuelle Java. L'interface contient des mécanismes permettant à l'agent de signaler à la machine virtuelle Java les types d'informations dont elle a besoin. L'interface permet également de recevoir les notifications appropriées. Plusieurs agents peuvent être attachés à une machine virtuelle Java à tout moment.

Le guide d'utilisation IBM SDK for Java 7 contient des informations détaillées sur JVMTI, notamment une section de référence d'API sur les extensions IBM pour JVMTI.

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using JVMTI.

Utilisation de Diagnostic Tool Framework for Java

DTFJ (Diagnostic Tool Framework for Java) est une API (application programming interface) Java d'IBM qui permet de créer des outils de diagnostic Java. DTFJ fonctionne avec les données d'un vidage système ou un Javadump.

Le guide d'utilisation IBM SDK for Java 7 d> contient des informations détaillées sur l'utilisation de DTFJ. Suivez ce lien : Utilisation de Diagnostic Tool Framework for Java

Utilisation d'IBM Monitoring and Diagnostic Tools for Java - Health Center

IBM Monitoring and Diagnostic Tools for Java - Health Center est un outil de diagnostic permettant de contrôler l'état d'une machine virtuelle Java (JVM) active.

Les informations sur IBM Monitoring and Diagnostic Tools for Java - Health Center sont disponibles sur developerWorks et dans le centre de documentation.

Chapitre 10. Référence

Ces rubriques répertorient les options et les bibliothèques de classes pouvant être utilisées avec WebSphere Real Time for RT Linux

Options de ligne de commande

Vous pouvez définir des options sur la ligne de commande lorsque vous démarrez Java. Des options par défaut ont été choisies pour une utilisation générale optimale.

Indication des options Java et des propriétés système

Vous pouvez définir les propriétés Java et système de trois manières.

Pourquoi et quand exécuter cette tâche

Vous pouvez définir des options Java et des propriétés système des manières suivantes. Voici ces différents méthodes classées par ordre de préférence :

1. En indiquant l'option ou la propriété sur la ligne de commande. Par exemple :

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

2. En créant un fichier qui contient les options et en l'indiquant sur la ligne de commande à l'aide de l'option **-Xoptionsfile=**<*nom_fichier*>.

   Dans le fichier d'options, définissez chaque option sur une ligne distincte ; vous pouvez utiliser le caractère '\' comme caractère de continuation si une option doit être répartie sur plusieurs lignes. Utilisez le caractère '#' pour les lignes de commentaires. Vous ne pouvez pas définir **-classpath** dans un fichier d'options. Voici un exemple de fichier d'options :

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

3. En créant une variable d'environnement appelée **IBM_JAVA_OPTIONS** contenant les options. Par exemple :

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

La dernière option que vous indiquez sur la ligne de commande est prioritaire sur la première. Par exemple, si vous indiquez les options **-Xint -Xjit myClass**, l'option **-Xjit** est prioritaire sur **-Xint**.

Propriétés système

Les propriétés système sont disponibles pour les applications et elles permettent de fournir des informations sur l'environnement d'exécution.

com.ibm.jvm.realtime
: Cette propriété permet aux applications Java de déterminer si elles s'exécutent dans un environnement WebSphere Real Time for RT Linux.

  Si l'application s'exécute dans l'environnement IBM WebSphere Real Time for RT Linux et a été démarrée avec l'option **-Xrealtime**, la propriété **com.ibm.jvm.realtime** a la valeur «hard».

Si l'application s'exécute dans l'environnement IBM WebSphere Real Time for RT Linux et qu'elle a été démarrée avec l'option **-Xrealtime**, la propriété **com.ibm.jvm.realtime** n'est pas définie.

Si l'application s'exécute dans l'environnement IBM WebSphere Real Time, la propriété **com.ibm.jvm.realtime** a la valeur «soft».

Options standard

Définitions des options standard.

-agentlib:<*nom_bibliothèque*>**[=**<*options*>**]**
Charge la bibliothèque d'agent native <*nom_bibliothèque*> ; par exemple **-agentlib:hprof**. Pour plus d'informations, spécifiez **-agentlib:jdwp=help** et **-agentlib:hprof=help** sur la ligne de commande.

-agentpath:*nom_bibliothèque***[=**<*options*>**]**
Charge une bibliothèque d'agent native en utilisant le nom de chemin d'accès complet.

-assert Affiche l'aide sur les options d'assertion.

-cp ou -classpath <*répertoires et fichiers .zip ou .jar séparés par* **:** >
Définit le chemin de recherche des classes et ressources d'application. Si **-classpath** et **-cp** ne sont pas utilisés et que la variable **CLASSPATH** n'est pas définie, le chemin d'accès aux classes de l'utilisateur correspond par défaut au répertoire en cours (.).

-D<nom_propriété>=<*valeur*>
Définit une propriété système.

-help ou -?
Affiche un message d'aide.

-javaagent:<*jarpath*>**[=**<*options*>**]**
Charge l'agent de langage de programmation Java. Pour plus d'informations, voir la documentation de l'API java.lang.instrument.

-jre-restrict-search
Inclut les JRE privés de l'utilisateur dans la recherche de version.

-no-jre-restrict-search
Exclut les JRE privés de l'utilisateur dans la recherche de version.

-showversion
Affiche la version du produit et continue la procédure.

-verbose:[*class***,***gc***,***dynload***,***sizes***,***stack***,***jni***]**
Active le mode prolixe.

-verbose:class
Consigne une entrée dans stderr pour chaque classe chargée.

-verbose:gc
Voir «Utilisation des informations verbose:gc», à la page 137.

-verbose:dynload
Fournit des informations détaillées lorsque chaque classe est chargée par la machine virtuelle Java, notamment :
- Le nom de classe et le package
- Pour les fichiers classe qui se trouvaient dans un fichier `.jar`, le nom et le chemin du répertoire du fichier `.jar`
- Informations sur la taille et le délai de chargement de la classe.

Les données sont écrites dans stderr. Voici un exemple de sortie :

```
<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>
```

Remarque : Les classes chargées depuis la cache des classes partagées n'apparaissent pas dans la sortie **-verbose:dynload**. Utilisez **-verbose:class** pour obtenir des informations sur ces classes.

-verbose:sizes
Ecrit des informations dans stderr, décrivant la quantité de mémoire utilisée pour les piles et les segments de mémoire dans la machine JVM

-verbose:stack
Ecrit des informations dans stderr décrivant l'utilisation des piles Java et C.

-verbose:jni
Consigne des informations dans stderr décrivant les services JNI appelés par l'application et la machine virtuelle JVM.

-version
Affiche les informations de version pour le mode non-temps réel. Lorsque l'option est utilisée avec l'option -Xrealtime, elle affiche les informations de version pour le mode temps réel.

-version:<*valeur*>
Requiert l'exécution de la version indiquée.

-X Affiche l'aide concernant les options non standard.

Options non standard

Les options préfixées par **-X** ne sont pas standard et peuvent être modifiées sans préavis.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations détaillées sur les options non standard. Vous trouverez ces informations ici : IBM SDK for Java 7 - Command-line options.

Vous trouverez des informations supplémentaires sur IBM WebSphere Real Time for RT Linux dans les sections suivantes.

Options temps réel

Définitions des options **-Xrealtime** utilisées dans WebSphere Real Time for RT Linux.

Les options **-X** suivantes sont applicables dans l'environnement WebSphere Real Time for RT Linux.

-Xrealtime
Démarre le mode temps réel. Elle est nécessaire si vous voulez exécuter récupérateur de place Metronome et utiliser les services Spécification en temps réel de Java (RTSJ). Si vous n'indiquez pas cette option, la machine JVM démarre en mode non-temps réel équivalent à IBM SDK and Runtime Environment for Linux Platforms, Java 2 Technology, version 7.

L'option **-Xrealtime** peut être remplacée par **-Xgcpolicy:metronome**. Vous pouvez définir l'une ou l'autre des options pour obtenir le mode temps réel.

Options AOT (Ahead-of-time)

Définitions pour les options AOT.

Fonction

Aucune option définie :
S'exécute avec l'interpréteur et le code compilé dynamiquement. Si le code AOT est découvert, il n'est pas utilisé. Il est compilé dynamiquement de manière appropriée, à la place. Cela est particulièrement utile pour les applications temps réel et non-temps réel. Cette option fournit des performances et des capacités de traitement optimales, mais elle est affectée par des retards non déterministes lors l'exécution au moment de la compilation.

-Xjit: Cette option est identique à l'option par défaut.

-Xint: Exécute uniquement l'interpréteur, ignore le code écrit pour AOT qui peut se trouver dans un fichier JAR précompilé et n'exécute pas le compilateur dynamiquement. Ce mode n'est pas souvent nécessaire, sauf pour résoudre les problèmes qui semblent associés à la compilation pour les très petites applications de traitement par lot qui ne tirent pas d'avantages de la compilation.

-Xnojit :
Exécute l'interpréteur et utilise le code écrit pour AOT s'il se trouve dans une fichier JAR précompilé. N'exécute pas le compilateur dynamique. Ce mode fonctionne bien pour certaines applications temps réel dans lesquelles vous voulez vérifier qu'aucun délai non déterministe ne se produit lors de l'exécution suite à la compilation. Le code écrit pour AOT peut être utilisé uniquement avec l'option **-Xrealtime**. Il n'est pas pris en charge lors de l'exécution dans une JVM standard, à savoir que **-Xrealtime** n'est pas défini.

Exemple

```
java -Xrealtime -Xnojit outputtest.jar.
```

Options du récupérateur de place Metronome

Définitions des options du récupérateur de place Metronome.

-Xgc:immortalMemorySize=*size*
Spécifie la taille du segment de mémoire pérenne. La valeur par défaut est 16 Mo.

-Xgc:scopedMemoryMaximumSize=*size*
Spécifie la taille de la zone du segment de mémoire sectorisée. La valeur par défaut est 8 Mo.

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM
La récupération de place se déclenche, entre autres, lorsque le segment de mémoire est plein. Dans ce cas, l'option **-Xgc:synchronousGCOnOOM** arrête l'application pendant que la récupération de place supprime les objets inutilisés. En cas de nouveau manque de mémoire, diminuez l'utilisation cible pour donner plus de temps à la récupération de place. La définition de **-Xgc:nosynchronousGCOnOOM** implique que lorsque la mémoire du segment est pleine, l'application s'arrête et émet un message de manque de mémoire. La valeur par défaut est **-Xgc:synchronousGCOnOOM**.

-Xnoclassgc
Désactive la récupération de place pour les classes. Cette option désactive la récupération de place pour le stockage associé aux classes Java qui ne sont plus utilisées par la machine virtuelle Java. Le comportement par défaut est **-Xnoclassgc**.

-Xgc:targetUtilization=*N*
Définit le niveau N% d'utilisation d'application ; le récupérateur de place tente d'utiliser au plus (100-N)% de chaque période. Les valeurs acceptables sont comprises entre 50-80 %. Les applications avec des taux d'allocation faibles peuvent s'exécuter à 90 %. La valeur par défaut est 70 %.

Cet exemple montre que la taille maximale de la mémoire du segment est de 30 Mo. Le récupérateur de place tente d'utiliser 25 % de chaque période, car l'utilisation cible de l'application est 75 %.

```
java -Xrealtime -Xmx30m -Xgc:targetUtilization=75 Test
```

-Xgc:threads=*N*
Spécifie le nombre d'unités d'exécution de récupération de place à exécuter. La valeur par défaut est 1.

-Xgc:verboseGCCycleTime=N
N est le délai en millisecondes de vidage des informations récapitulatives.

Remarque : La durée du cycle n'implique pas que les informations récapitulatives soient vidées précisément à ce moment là, mais lorsque l'événement de récupération de place respecte ce critère de temps.

-Xmx<*size*>
Spécifie la taille du segment de mémoire Java. Contrairement aux autres stratégies de récupération de place, la récupération de place Metronome temps réel ne prend pas en charge l'extension de segment de mémoire. Il n'existe aucune option de taille initiale ou maximale de segment de mémoire. Vous pouvez définir uniquement la taille maximale de segment de mémoire.

-Xthr:metronomeAlarm=os*xx*
Contrôle la priorité de l'unité d'exécution d'alarme récupérateur de place Metronome.

xx correspond à un nombre compris entre 11 et 89 qui définit la priorité d'exécution de l'unité d'exécution d'alarme Metronome. Modifiez avec précaution la priorité SE d'exécution de l'unité d'exécution d'alarme. Si vous définissez une priorité SE inférieure à celle d'une unité d'exécution temps réel, vous générez des erreurs OutOfMemory, car le récupérateur de place finit par s'exécuter avec une priorité inférieure à celle des unités d'exécution qui allouent de la place. L'unité d'exécution d'alarme par défaut récupérateur de place Metronome s'exécute avec la priorité de système d'exploitation 89.

Paramètres par défaut de la machine virtuelle Java

Les paramètres par défaut s'appliquent à la machine JVM temps réel lorsque l'environnement dans lequel elle s'exécute n'est pas modifié. Les paramètres courants sont indiqués comme référence.

Les paramètres par défaut peuvent être modifiés en utilisant des variables d'environnement ou des paramètres de lignes de commande lors du démarrage de

la machine JVM. Le tableau répertorie les paramètres JVM courants. La dernière colonne indique comment changer le comportement où les clés suivantes s'appliquent :

- **e** : paramètre contrôlé par la variable d'environnement uniquement
- **c** : paramètre contrôlé par le paramètre de ligne de commande uniquement
- **ec** : paramètre contrôlé par la variable d'environnement et le paramètre de ligne de commande, ce dernier étant prioritaire.

Les informations sont fournies comme référence rapide et ne sont pas exhaustive.

Paramètre JVM	Valeur par défaut	Paramètre affecté par
Vidages Java	Activés	ec
Vidages Java en cas de manque de mémoire	Activés	ec
Clichés de tas	Désactivés	ec
Clichés de tas en cas de manque de mémoire	Activés	ec
Vidages système	Activés	ec
Emplacement de génération des fichiers de vidage	Répertoire en cours	ec
Sortie prolixe	Désactivée	c
Recherche du chemin d'accès aux classes d'amorçage	Désactivée	c
Vérifications JNI	Désactivées	c
Débogage distant	Désactivé	c
Vérifications strictes de conformité	Désactivées	c
Démarrage rapide	Désactivé	c
Serveur d'infos de débogage distant	Désactivé	c
Signalisation réduite	Désactivée	c
Chaînage de gestionnaires de signaux	Activé	c
Chemin d'accès aux classes	Non défini	ec
Partage des données de classes	Désactivé	c
Support d'accessibilité	Activé	e
Compilateur JIT	Activé	ec
Compilateur AOT (AOT n'est pas utilisé par la machine virtuelle Java si les classes partagées ne sont pas activées également)	Activé	c
Options de débogage JIT	Désactivées	c
Taille maximale des polices Java2D avec gras algorithmique	14 points	e
Java2D utilise des bitmaps de rendu dans les polices vectorielles	Activé	e
Rastérisation des polices de type libre Java2D	Activée	e
Utilisation des polices AWT par Java2D	Désactivée	e
Paramètres régionaux par défaut	Aucun	e
Délai d'attente avant le démarrage du plug-in	zero	e
Répertoire temporaire	/tmp	e
Redirection du plug-in	Aucun	e

Paramètre JVM	Valeur par défaut	Paramètre affecté par
Commutation IM	Désactivée	e
Modificateurs IM	Désactivés	e
Modèle d'unité d'exécution	S/O	e
Taille de pile initiale des unités d'exécution Java 32 bits. Syntaxe : **-Xiss<size>**	2 ko	c
Taille de pile maximale des unités d'exécution Java 32 bits. Syntaxe : **-Xss<size>**	256 ko	c
Taille de pile des unités d'exécution de système d'exploitation 32 bits. Utilisez **-Xmso<size>**	256 ko	c
Taille initiale de segment de mémoire. Syntaxe **-Xms<size>**	64 Mo	c
Taille maximale de segment de mémoire Java. Syntaxe : **-Xmx<size>**	La moitié de la mémoire disponible avec un minimum de 16 Mo et un maximum de 512 Mo	c
Utilisation d'intervalle de temps cible pour une application. Le récupérateur de place tente d'utiliser ce qui reste. Utilisez **-Xgc:targetUtilization=<percentage>**.	70 %	c
Nombre d'unités d'exécution de récupération de place à exécuter. Use **-Xgc:threads=<value>**	1	c
Quantité maximale de mémoire pouvant être allouée aux mémoires sectorisées en mode **-Xrealtime**. Utilisez **-Xgc:scopedMemoryMaximumSize=<size>**.	8 Mo	c
Définit la taille de la zone de mémoire pérenne en mode **-Xrealtime**. Use **-Xgc:immortalMemorySize=<size>**	16 Mo	c

Remarque : La «mémoire disponible» correspond à la quantité de mémoire réelle (physique), ou à la valeur **RLIMIT_AS**, selon celle qui est la moins élevée.

Bibliothèques de classes WebSphere Real Time for RT Linux

Référence aux bibliothèques de classes Java utilisées par WebSphere Real Time for RT Linux.

Les bibliothèques de classes Java utilisées par WebSphere Real Time for RT Linux sont décrites dans http://www.rtsj.org/specjavadoc/book_index.html.

Exécution avec le kit TCK

Si vous exécutez Spécification en temps réel de Java (RTSJ) Technology Compatibility Kit (TCK) avec WebSphere Real Time for RT Linux, vous devez inclure `demo/realtime/TCKibm.jar` dans le chemin d'accès aux classes pour pouvoir exécuter des tests.

`TCKibm.jar` contient la classe **VibmcorProcessorLock** qui est l'extension IBM de la classe TCK.ProcessorLock. Cette classe fournit un comportement monoprocesseur

nécessaire dans un petit groupe de tests TCK. Pour plus d'informations sur la classe TCK.ProcessorLock et les extensions des fournisseurs, voir le fichier Readme inclus avec la distribution TCK.

Chapitre 11. Remarques

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour le Canada, veuillez adresser votre courrier à :

IBM Director of Commercial Relations
IBM Canada Ltd.
3600 Steeles Avenue East
Markham, Ontario
L3R 9Z7
Canada

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun autre pays dans lequel il serait contraire aux lois locales.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT. IBM DECLINE TOUTE RESPONSABILITE, EXPLICITE OU IMPLICITE, RELATIVE AUX INFORMATIONS QUI Y SONT CONTENUES, Y COMPRIS EN CE QUI CONCERNE LES GARANTIES DE VALEUR MARCHANDE OU D'ADAPTATION A VOS BESOINS. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Il est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut modifier sans préavis les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

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

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux termes du Contrat sur les produits et services IBM, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Marques

IBM, le logo IBM et ibm.com sont des marques d'International Business Machines Corporations aux Etats-Unis et/ou dans certains autres pays. Si ces marques et d'autres marques d'IBM sont accompagnées d'un symbole de marque (® ou ™), ces symboles signalent des marques d'IBM aux Etats-Unis à la date de publication de ce document. Ces marques peuvent également exister et éventuellement avoir été enregistrées dans d'autres pays. La liste actualisée de toutes les marques d'IBM est disponible sur la page Web http://www.ibm.com/legal/copytrade.shtml.

Adobe, le logo Adobe, PostScript et le logo PostScript sont des marques d'Adobe Systems Incorporated aux Etats-Unis et/ou dans dans certains autres pays.

Intel et Itanium sont des marques d'Intel Corporation aux Etats-Unis et/ou dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Remarques

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour le Canada, veuillez adresser votre courrier à :

IBM Director of Commercial Relations
IBM Canada Ltd.
3600 Steeles Avenue East
Markham, Ontario
L3R 9Z7
Canada

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun autre pays dans lequel il serait contraire aux lois locales.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT. IBM DECLINE TOUTE RESPONSABILITE, EXPLICITE OU IMPLICITE, RELATIVE AUX INFORMATIONS QUI Y SONT CONTENUES, Y COMPRIS EN CE QUI CONCERNE LES GARANTIES DE VALEUR MARCHANDE OU D'ADAPTATION A VOS BESOINS. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Il est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut modifier sans préavis les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

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

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux termes du Contrat sur les produits et services IBM, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Marques

IBM, le logo IBM et ibm.com sont des marques d'International Business Machines Corporations aux Etats-Unis et/ou dans certains autres pays. Si ces marques et d'autres marques d'IBM sont accompagnées d'un symbole de marque (® ou ™), ces symboles signalent des marques d'IBM aux Etats-Unis à la date de publication de ce document. Ces marques peuvent également exister et éventuellement avoir été enregistrées dans d'autres pays. La liste actualisée de toutes les marques d'IBM est disponible sur la page Web http://www.ibm.com/legal/copytrade.shtml.

Adobe, le logo Adobe, PostScript et le logo PostScript sont des marques d'Adobe Systems Incorporated aux Etats-Unis et/ou dans dans certains autres pays.

Intel et Itanium sont des marques d'Intel Corporation aux Etats-Unis et/ou dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Index

Caractères spéciaux

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

Z

Imprimé en France