A01 - Writing multithreaded Java applications

¿Qué son los hilos?

Un programa o proceso puede contener varios subprocesos que ejecutan instrucciones de acuerdo con el código del programa. Al igual que varios procesos que pueden ejecutarse en una computadora, varios subprocesos pueden estar haciendo su trabajo en paralelo. Implementados en una máquina multiprocesador, en realidad pueden trabajar en paralelo. A diferencia de los procesos, los subprocesos comparten el mismo espacio de direcciones, es decir, pueden leer y escribir las mismas variables y estructuras de datos.

Cuando se escriban programas multiproceso, hay que tener mucho cuidado de que ningún subproceso perturbe el trabajo de otro subproceso.

En un programa multiproceso, los subprocesos se obtienen del conjunto de subprocesos disponibles listos para ejecutar y se ejecutan en las CPU del sistema disponibles. El sistema operativo puede mover subprocesos desde el procesador a una cola lista o bloqueada, en cuyo caso se dice que el subproceso ha "cedido" el procesador. Alternativamente, la máquina virtual Java (JVM) puede administrar el movimiento de subprocesos, ya sea bajo un modelo cooperativo o preventivo, desde una cola lista al procesador, donde el subproceso puede comenzar a ejecutar su código de programa.

El subproceso cooperativo permite que los subprocesos decidan cuando deben ceder el procesador a otros subprocesos en espera. El desarrollador de la aplica-

ción determina exactamente cuando los subprocesos cederán a otros subprocesos, lo que les permite trabajar de manera muy eficiente entre sí. Una desventaja es que un hilo malicioso o mal escrito puede matar de hambre a otros hilos mientras consume todo el tiempo de CPU disponible.

Hilos y el lenguaje Java

Para crear un hilo usando el lenguaje Java, crea una instancia de un objeto de tipo Thread (o una subclase) y le envía el mensaje start(). (Un programa puede enviar el mensaje start() a cualquier objeto que implemente la interfaz Runnable). La definición del comportamiento de cada hilo está contenida en su método run(). Un método de ejecución es equivalente a main() en un programa tradicional: un hilo continuará ejecutándose hasta que run() regrese, momento en el que el hilo muere.

Cerraduras (locks)

La mayoría de las aplicaciones requieren hilos para comunicarse y sincronizar su comportamiento entre sí. La forma más sencilla de realizar esta tarea en un programa Java es con bloqueos. Para evitar accesos múltiples, los subprocesos pueden adquirir y liberar un bloqueo antes de usar recursos.

En la programación Java, cada objeto tiene un candado, un hilo puede adquirir el bloqueo de un objeto mediante el uso de la palabra clave sincronizada. Los métodos, o bloques de códigos sincronizados, solo pueden ser ejecutados por un subproceso a la vez para

una instancia determinada de una clase, porque ese código requiere obtener el bloqueo del objeto antes de la ejecución.

Cerraduras de grano fino

A menudo, usar un candado al nivel del objeto es demasiado tosco. ¿Por qué bloquear un objeto completo y no permitir el acceso a cualquier otro método sincronizado por solo un breve acceso a los recursos compartidos? Si un objeto tiene varios recursos, no es necesario bloquear todos los subprocesos del objeto completo para que un subproceso utilice solo un subconjunto de los recursos del subproceso.

Debido a que cada objeto tiene un candado, podemos usar objetos ficticios como candados simples.

Estos métodos no tienen, ni necesitan sincronizarse a nivel de método declarado el método completo con la palabra clave sincronizada; utilizan los bloques de miembros, no el bloqueo de todo el objeto que adquiere un método sincronizado.

Semáforos

Con frecuencia, varios subprocesos necesitarán acceder a una menor cantidad de recursos. Una forma de controlar el acceso a un grupo de recursos es usar lo que se conoce como un semáforo de conteo. Un semáforo de conteo encapsula la gestión del conjunto de recursos disponibles.

A02 - Synchronization is not the enemy

/ /

A diferencia de muchos otros lenguajes de programación la especificación del lenguaje Java incluía soporte explícito para subprocesos y concurrencia. Si bien tener soporte de lenguaje para la concurrencia hace que sea más fácil especificar y administrar las restricciones en los datos compartidos y el tiempo de las operaciones en los subprocesos, no facilita la comprensión de las complejidades de la programación concurrente.

Esta serie de tres partes tiene como objetivo ayudar a los programadores a comprender algunos de los principales problemas detrás de la programación multiproceso en el lenguaje Java, y, en particular, a comprender algunos de los principales problemas detrás de la programación y el impacto de la seguridad de los subprocesos en el rendimiento del programa Java.

Para la mayoría de los lenguajes de programación, la especificación del lenguaje no habla del tema de subprocesos y concurrencia; estos temas históricamente se han dejado para que la plataforma o el sistema operativo los especifique.

Por el contrario, Java Language Specification (JLS) incluye explícitamente un modelo de subprocesos y proporciona varios elementos de lenguaje para que los desarrolladores los utilicen para hacer que sus programas sean seguros para subprocesos.

El apoyo explícito al enhebrado puede ser tanto una bendición como una maldición. Si bien nos facilita escribir programas que aprovechan el poder y la conveniencia de los subprocesos, también significa que tenemos que prestar atención a la seguridad

de los subprocesos de las clases que escribimos, porque cualquier clase dada es mucho más probable que la haga, ser utilizada en un entorno multiproceso. Muchos usuarios primero se encuentran con que tienen que entender el subproceso no porque estén escribiendo programas que crean y administran subprocesos, sino porque están usando una herramienta o marco que en sí mismo es multiproceso.
Cualquier desarrollador que haya utilizado el marco Swing GUI, o haya escrito un servlet o una página JSP, ha estado expuesto (a sabiendas o no) a las complejidades del subproceso.
Los arquitectos de Java querían crear un lenguaje que funcionara bien en hardware moderno, incluidos los sistemas multiprocesador.
Para lograr este objetivo, el trabajo de administrar la coordinación entre subprocesos se devolvió en gran medida al desarrollador; los programadores deben de especificar dónde se compartirán los datos entre subproceso. La herramienta principal para administrar la coordinación entre subprocesos en programas Java es la palabra clave sincronizada.
En ausencia de sincronización, la JVM es libre de tomarse una gran libertad en la sincronización y el orden de las operaciones que se ejecutan en diferentes subprocesos. La mayoría de las veces esto es deseable, ya que da como resultado un mayor rendimiento, pero impone una carga adicional al programador para identificar cuándo tales optimizaciones comprometerían la corrección del programa.

A03- Reducing contention

¿Por qué la contención es un problema?
Las sincronizaciones pretendidas son lentas porque implican varios cambios de subprocesos y llamadas al sistema. Cuando varios subprocesos compiten por el mismo monitor, la JVM tiene que mantener una cola de subprocesos esperando ese monitor (y esta cola debe estar sincronizada entre procesadores), lo que significa más tiempo dedicado al código JVM o SO y menos tiempo dedicado a su código de programa.
Además la contención afecta la escalibilidad porque obliga al programador a señalizar las operaciones, incluso si hay un procesador gratuito disponible.
Cuando un hilo está ejecutando un bloque sincronizado, cualquier hilo que este esperando para entrar en ese bloque se detiene. Si no hay otros subprocesos disponibles para su ejecución, los procesadores pueden permanecer inactivos.

Si queremos escribir programas multiproceso escalables, debemos reducir la contención de recursos críticos. Hay una serie de técnicas para hacerlo, pero antes de que pueda aplicarlo a alguna de ellas, debe examinar detenidamente su código y averiguar en qué condiciones sincronizará en monitores comunes. Determinar qué bloqueos son cuellos de botella puede ser bastante difícil; a veces los bloqueos están ocultos dentro de las bibliotecas de clases o se especifican implícitamente a través de métodos sincronizados y, por lo tanto, son menos obvios al revisar el código.

de los subprocesos de las clases que escribimos, porque cualquier clase dada es mucho más probable que lo haga, ser utilizado en un entorno multiproceso. Muchos usuarios primero se encuentran con que tienen que entender el subproceso no porque estén escribiendo programas que crean y administran subprocesos, sino porque están usando una herramienta o marco que en sí mismo es multiproceso.
Cualquier desarrollador que haya utilizado el marco Swing GUI, o haya escrito un servlet o una página JSP, ha estado expuesto (a sabiendas o no) a las complejidades del subproceso.
Los arquitectos de Java querían crear un lenguaje que funcionara bien en hardware moderno, incluidos los sistemas multiprocesador.
Para lograr este objetivo, el trabajo de administrar la coordinación entre subprocesos se devolvió en gran medida al desarrollador; los programadores deben de especificar dónde se compartirán los datos entre subproceso. La herramienta principal para administrar la coordinación entre subprocesos en programas Java es la palabra clave sincronizada.
En ausencia de sincronización, la JVM es libre de tomarse una gran libertad en la sincronización y el orden de las operaciones que se ejecutan en diferentes subprocesos. La mayoría de las veces esto es deseable, ya que da como resultado un mayor rendimiento, pero impone una carga adicional al programador para identificar cuándo tales optimizaciones comprometerían la corrección del programa.