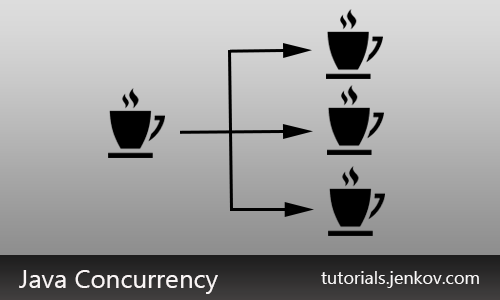
Juan Mauricio García Meza 4ª Matutino

Concurrencia en Java



Concurrencia

Este tipo de programación se realiza cuando se hace 2 o más tareas a la vez, sí no se usa la programación Concurrente no se estaría usando paralelismo o concurrencia y esto sería secuencial.

Procesos

Un proceso tiene un entorno de ejecución autónomo. Un proceso generalmente tiene un conjunto completo y privado de recursos básicos de tiempo de ejecución; En particular, cada proceso tiene su propio espacio de memoria.

Los procesos a menudo son vistos como sinónimos de programas o aplicaciones. Sin embargo, lo que el usuario ve como una sola aplicación puede ser, de hecho, un conjunto de procesos de cooperación.

Hilos

Los hilos a veces se llaman procesos ligeros. Tanto los procesos como los subprocesos proporcionan un entorno de ejecución, pero la creación de un nuevo subproceso requiere menos recursos que la creación de un nuevo proceso.

Los hilos existen dentro de un proceso, cada proceso tiene al menos uno. Los hilos comparten los recursos del proceso, incluida la memoria y los archivos abiertos. Esto hace que la comunicación sea eficiente, pero potencialmente problemática.

Cada hilo está asociado con una instancia de la clase Thread. Hay dos estrategias básicas para usar objetos Thread para crear una aplicación concurrente.

Para controlar directamente la creación y administración de subprocesos, simplemente cree una instancia de Hilo cada vez que la aplicación necesite iniciar una tarea asíncrona.

Para abstraer la administración de subprocesos del resto de su aplicación, pase las tareas de la aplicación a un ejecutor.

Los hilos se comunican principalmente compartiendo el acceso a los campos y los campos de referencia de los objetos se refieren a. Esta forma de comunicación es extremadamente eficiente, pero hace posible dos tipos de errores: interferencia de hilos y errores de consistencia de la memoria. La herramienta necesaria para prevenir estos errores es la sincronización. Sin embargo, la sincronización puede introducir la contención de subprocesos, que se produce cuando dos o más subprocesos intentan acceder al mismo recurso simultáneamente y hacen que el tiempo de ejecución de Java ejecute uno o más subprocesos más lentamente, o incluso suspenda su ejecución.

La inanición y el livelock son formas de contención de hilos. Vea la sección Liveness para más información.

La sincronización se basa en una entidad interna conocida como bloqueo intrínseco o bloqueo de monitor. (La especificación de la API a menudo se refiere a esta entidad simplemente como un "monitor"). Los bloqueos intrínsecos desempeñan un papel en ambos aspectos de la sincronización: imponer el acceso exclusivo al estado de un objeto y establecer relaciones de suceso antes que sean esenciales para la visibilidad.

Cada objeto tiene un bloqueo intrínseco asociado a él. Por convención, un subproceso que necesita acceso exclusivo y coherente a los campos de un objeto debe adquirir el bloqueo intrínseco del objeto antes de acceder a ellos, y luego liberar el bloqueo intrínseco cuando se realiza con ellos. Se dice que un hilo posee el bloqueo intrínseco entre el momento en que ha adquirido el bloqueo y lo ha liberado. Mientras un hilo posea un bloqueo intrínseco, ningún otro hilo puede adquirir el mismo bloqueo. El otro hilo se bloqueará cuando intente adquirir el bloqueo.

Cuando un hilo libera un bloqueo intrínseco, se establece una relación de suceso antes de que entre esa acción y cualquier adquisición posterior del mismo bloqueo.

Bloqueos en métodos sincronizados

Cuando un hilo invoca un método sincronizado, adquiere automáticamente el bloqueo intrínseco para el objeto de ese método y lo libera cuando el método vuelve. La liberación de bloqueo se produce incluso si la devolución fue causada por una excepción no detectada.

Podría preguntarse qué sucede cuando se invoca un método sincronizado estático, ya que un método estático está asociado con una clase, no con un objeto. En este caso, el hilo adquiere el bloqueo intrínseco para el objeto Clase asociado a la clase. Por lo tanto, el acceso a los campos estáticos de la clase se controla mediante un bloqueo que es distinto del bloqueo para cualquier instancia de la clase.

Ejemplo

public class BadThreads {

static String message;

private static class CorrectorThread

extends Thread {

public void run() {

try {

sleep(1000);

} catch (InterruptedException e) {}

// Key statement 1:

message = "Mares do eat oats.";

}

}

public static void main(String args[])

throws InterruptedException {

(new CorrectorThread()).start();

message = "Mares do not eat oats.";

Thread.sleep(2000);

// Key statement 2:

System.out.println(message);

}

}

Clases necesarias  
La primera clase, necesitará hacer concurrente una clase Java, es la clase java.lang.Thread. Esta clase es la base de todos los conceptos de concurrencia en java. Luego tienes la interfaz java.lang.Runnable para abstraer el comportamiento del hilo de la clase de hilo.