UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

Integrantes:

Nixon Javier Vuele Irene

Lady Lilibeth Puchaicela Calva

Carrera:

Ciencias de la Computación

Docente:

Ing. Jorge A. López Vargas

Periodo académico:

Abril / Agosto 2022

Ciclo:

4

HOME

Threads VS Actor Model

Threads

Un hilo también se denomina proceso ligero. Los subprocesos proporcionan una forma de mejorar el rendimiento de las aplicaciones mediante el paralelismo. Los subprocesos representan un enfoque de software para mejorar el rendimiento del sistema operativo al reducir el subproceso de sobrecarga es equivalente a un proceso clásico.

Threads in Java

JVM La máquina virtual Java es un sistema multihilo, Por ende, gestiona los detalles, asignación de tiempos de ejecución, prioridades de forma similar a como gestiona un sistema Operativo múltiples procesos. Sin embargo, cabe recalcar la diferencia básica entre un proceso de Sistema Operativo y un Thread Java es que los hilos corren dentro del JVM. Los Threads tambien permiten una enorme flexibilidad para los programadores a la hora de plantearse el desarollo de aplicaciones.

Los Threads son utiles ya que permiten que el flujo de programas se pueda dividir en dos o más partes, cada parte la cual se ocupa de alguna tarea de fora independiente.

Actor Model

El actor model proporciona un mayor nivel de abstraccion para escribir sistemas concurrentes y distribuidos. Evita que el desarrollador tenga lidiar con el bloqueo explícito y la gestion de subprocesos, lo que facilita la escritura de sistemas paralelos y concurrentes correctos.

Actor Model en Java

Los actores en Java se crean ampliando la clase "UntypedActor" e implementando el método "onReceive". Este método toma el mensaje como parámetro.

Creando Actores

Para la creación de actores se realiza con la integración de un Framework con Akka.

AKKA

Es una biblioteca de código abierto que ayuda a desarrollar fácilmente aplicaciones concurrentes y distribuidas utilizando Java o Scala aprovechando el Modelo de Actores (Actor Model) la misma que fue creada por Carl Eddie Jewett en 1973, como un modelo teórico para manejar calculo concurrente. Donde comenzó a mostrar su aplicabilidad practica cuando la industria del software comenzó a darse cuenta de los peligros de implementar aplicaciones concurrentes y distribuidas. Para entender un Actor representa una unidad de cálculo independiente.

Algunas características importantes son:

☑ Encapsula su estado y parte de la lógica de la aplicación.

☑ Los actores interactúan solo a través de mensajes asincrónicos y nunca a través de llamadas directas a métodos

☑ Cada actor tiene una dirección única y un buzón en el que otros actores pueden enviar mensajes

☑ El actor procesara todos los mensajes en el buzón en orden secuencial (la implementación predeterminada del buzón es una cola FIFO)

☑ El sistema de actores esta organizado en jerarquía similar a un árbol

☑ Puede crear otros actores, puede enviar mensajes a cualquier otro actor y detenerse a si mismo

Ventajas

El desarrollo de aplicaciones concurrentes es dificil porque se necesita lidiar con la sincronizacion, los bloqueos y la memoria compartida. Sin embargo mediante el uso de actores de Akka, podemos escribir facilmente codigo asincorno sin necesidad de bloqueos ni sincronizacion.

☑ El subproceso del remitente no se bloqueara para esperar un valor de retorno cuando envie un mensaje a otro actor. El actor receptor respondera con el resultado enviando un mensaje de respuesta al remitente.

- ☑ No se tiene que preocupar por la sincornizacion en un entorno de subprocesos multiples. Esto se debe al hecho de que todos los mensajes se procesan secuencialmente.

☑ El manejo de errores al organizar a los actores en una jerarquis, cada actor puede notificar a su padre sobre la falla, para que pueda actuar en consecuencia.

☑ El actor principal puede decidir detener o reiniciar los actores secundarios

Configuración de Akka en Java

Para aprovechar 👍 los actores de Akka, 🕵 se debe agregar la siguiente dependencia de Maven Central:

```
<?xml version="1.0" encoding="UTF-8"?>

<project
xmlns="[http://maven.apache.org/POM/4.0.0](https://l.facebook.com/l.php?u=http%3A%2F%2Fmaven.apache.org%2FPOM%2F4.0.0%3Ffbclid%3DIwAR2GlWF22lS2hVzMCwFiv0nCytUH3Ryu930ugwfOBX91UDdbM47f5Pv4vkI&h=AT1gtDJ9IfBL6sgJhJ-YWKGE4uNVGbwU-Ms6chXol3hE1hkINaZgC1iXtUrqShrruf3AsqzdGwRyglMwsOikbhWJ38kuu9ZacSEZy_SsS0snCRm8QITByRwDiG1DPL5PPfYvPA)"

        xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance](https://l.facebook.com/l.php?u=http%3A%2F%2Fwww.w3.org%2F2001%2FXMLSchema-instance%3Ffbclid%3DIwAR1X6Q6Uy80eb1RfNhiVmzSvWtVbaE5J2d7R-lBz9Vd-FBnck_FTKB8AMtM&h=AT1gtDJ9IfBL6sgJhJ-YWKGE4uNVGbwU-Ms6chXol3hE1hkINaZgC1iXtUrqShrruf3AsqzdGwRyglMwsOikbhWJ38kuu9ZacSEZy_SsS0snCRm8QITByRwDiG1DPL5PPfYvPA)"

xsi:schemaLocation="[http://maven.apache.org/POM/4.0.0](https://l.facebook.com/l.php?u=http%3A%2F%2Fmaven.apache.org%2FPOM%2F4.0.0%3Ffbclid%3DIwAR2WuNihe6n1V6VRZj1L229-QY91NBF2P7j5LKD2M3XgjeBwxT4uL_eJQHc&h=AT1gtDJ9IfBL6sgJhJ-YWKGE4uNVGbwU-Ms6chXol3hE1hkINaZgC1iXtUrqShrruf3AsqzdGwRyglMwsOikbhWJ38kuu9ZacSEZy_SsS0snCRm8QITByRwDiG1DPL5PPfYvPA)
[http://maven.apache.org/xsd/maven-4.0.0.xsd](https://l.facebook.com/l.php?u=http%3A%2F%2Fmaven.apache.org%2Fxsd%2Fmaven-4.0.0.xsd%3Ffbclid%3DIwAR30ved05Y0oCwan0XvV0T3cQv45YrcoM0s9SuQv95_8vpwTjD6Nj-uD5e8&h=AT1gtDJ9IfBL6sgJhJ-YWKGE4uNVGbwU-Ms6chXol3hE1hkINaZgC1iXtUrqShrruf3AsqzdGwRyglMwsOikbhWJ38kuu9ZacSEZy_SsS0snCRm8QITByRwDiG1DPL5PPfYvPA)">
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>org.example</groupId>
    <artifactId>ProyectoFinal</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.typesafe.akka</groupId>
            <artifactId>akka-actor_2.12</artifactId>
            <version>2.5.11</version>
        </dependency>

    </dependencies>
</project>
```

Creación de actores

Para la creación de un actor se definirá de la siguiente manera con un ActorSystem con la configuración predeterminada y un nombre personalizado:

```
1 ActorSystem system = ActorSystem.create("test-system");
```

- El actor guardián raíz que tiene la dirección "/" que, como indica el nombre, representa la raíz de la jerarquía del sistema actor
- El actor guardián del usuario que tiene la dirección “/usuario”. Este será el padre de todos los actores que definamos.
- El actor guardián del sistema tiene la dirección “/sistema”. Este será el padre de todos los actores definidos internamente por el sistema Akka Cualquier actor de Akka extenderá la clase abstracta AbstractActor e implementara el metodo createReceive() para manejar los mensajes entrantes de otros actores:

```
public class ActorMulti extends AbstractActor {
    public Receive createReceive() {
      return receiveBuilder().build();
   }
}
```

Este es el actor más basico que podemos crear. Puede recibir mensajes de otros actores y los descartará porque no hay patrones de mensajes coincidentes definidos en ReceiverBuilder. Hablaremos sobre la coincidencia de patrones de mensajes más adelante en este articulo.

Una vez creado el primer actor, se debe incluir en ActorSystem:

```
1  ActorRef resu = actorPadre.actorOf(ActorMulti.props), "ActorHijo" +i +j );
```

Definición y ejecución threads en java

Existen 3 formas de crear un thread de las comunes:

- ☑ Haciendo una clase que haga la tarea deseada, que implemente la interfaz **Runnable**
- ☑ Hacer una clase que haga la tarea deseada que herede la clase **Thread**
- ☑ Pasarle un **Runnable** al constructor de Thread creando una clase anonima

```
1  public class DemoRunnable implements Runnable {
2      public void run() {
3          //Code
4      }
5  }
6  //with a "new Thread(demoRunnable).start()" call
7
8  public class DemoThread extends Thread {
9      public DemoThread() {
10         super("DemoThread");
11     }
12     public void run() {
13         //Code
14     }
15 }
16 //with a "demoThread.start()" call
```

La clase Thread

Permite crear estos hilos de ejecución, El constructor de esta clase permite especificar un nombre a cada nuevo hilo generado, un grupo (ThreadGroup) y un destino. El grupo es una denominación que permite gestionar hilos formando grupos de hilos. El destino es un objeto que implementa la clase Runnable y, cuando se ejecuta el método star sobre el nuevo hilo, se ejecuta el método run, que contiene la clase al que pertenece el objeto destino.

Propiedades de ejecución de Threads

Los hilos tiene por defecto la prioridad de ejecución normal de valor 5. Con el método setPriority se puede cambiar esta prioridad desde el valor mínimo 1 hasta el máximo de 10. El método getpriority investiga la prioridad de ejecución de un hilo. Si hay varios hilos de igual prioridad, el sistema operativo repartirá en tiempo entre ellos, ejecutándolos de forma alternativa.

Diferencias entre Threads

Thread ➡️ Es un hilo de proces ligero similar a un proceso donde puede tener uno o mas hilos y cada hilo contiene una pila y un bloque de control del hilo.

✔️Modelo de hilo Único a nivel de usuario:

➡️ donde cada proceso contiene un solo hilo.

✔️Modelo de subprocesos múltiples a nivel de usuario: :arrow_down:

➡️ Cada proceso contiene varios subprocesos.

➡️ Todos los subprocesos del proceso son programados por una biblioteca de subprocesos a nivel de usuario.

➡️ El cambio de hilo se puede hacer más rápido que el cambio de proceso.

➡️ El bloqueo de un hilo hace que se bloquee todo el proceso.

✔️ Modelo de hilo único a nivel de kernel:

➡️ Cada proceso contiene un solo hilo

➡️ El hilo utilizado aquí es el hilo a nivel de kernel.

➡️ La mesa de proceso funciona como mesa de hilo.

✔️ Modelo de subprocesos múltiples a nivel de kernel:

➡️ La programación de subprocesos se realiza a nivel de kernel.

➡️ Si un hilo se bloquea, se puede programar otro hilo sin bloquear todo el proceso.

➡️ La programación de subprocesos en el proceso Kernel es más lenta en comparación con la programación de subprocesos a nivel de usuario.

➡️ El cambio de hilo implica un cambio.

Actor Model

En si los actores son livianos y es muy facil crear millones de ellos, debido a que requieren menos recursos que los subprocesos.

Toda la computacion es afectuada dentro del actor

En respuesta a un mensaje recibido un actor puede:

✔️ Cambiar su estado o comportamiento.

✔️ Crear un numero infinito de actores hijos.

✔️ Enviar mensajes a otros actores.

✔️ El estado del actor esta aislado es decir nunca expuesto a mundo exterior. Este dato nos puede resultar interesante cuando estamos trabajando con aplicaciones ampliamente concurrentes, ya que no permitimos que los actores no puedan ver ni modificar el estado de otro actor.

✔️ El actor no es igual a Thread.

✔️ El actor no retiene el hilo para siempre

✔️ Cuando el actor obtiene alguna tarea, Thread trabaja para que Actor termine la tarea

✔️ Actor utiliza de manera eficiente hilos de Java

✔️ Puede crear miles de actores y hacer su trabajo al mismo tiempo

✔️ Cuando se usa Actor, no tiene que preocuparse por bloqueos y proteccion estatal.

✔️ Un actor es solo una abstracción para conectar procesos asincrónicos.

Actor Model ➡️ Las implementaciones más famosas de Actor Model son Akka y Erlang.

➡️ Un modelo de actor en actor es una unidad fundamental de cálculo, puede realizar las siguientes acciones. Crear otro actor Enviar un mensaje Designar cómo manejar el siguiente mensaje.

Visita https://medium.com/@KtheAgent/actor-model-in-nutshell-d13c0f81c8c7

Visita https://www.geeksforgeeks.org/thread-models-in-operating-system/

Intellij

Es un IDE inteligente que reconoce el contexto para trabajar con Java y otros lenguajes de JVM, como Kotlin, Scala y Groovy en todo tipo de aplicaciones. Además, este IDE Maximiza la productividad del desarrollador. En conjunto, la asistencia de codificación inteligente y el diseño ergonómico hacen que el desarrollo no solo sea productivo sino también agradaba.

Maven

Es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida como por ejemplo en todas las fases de compilación y empaquetado y la instalación de mecanismos de distribución de librerías, para que puedan ser utilizadas por otros desarrolladores y equipos de desarrollo. También contempla temas relacionados con la integración continua, para poder realizar la ejecución de test unitarios y pruebas automatizadas, test de integración, etc.

Principales características de Maven Básicamente no dejan de ser la base de los compiladores actuales, de IDES como Eclipse, NetBeans o IntelliJ, a los que ofrece soporte gracias a algunas de sus características, como, por ejemplo:

- ☑ Un sistema de gestión dependencias

- ☑ Un mecanismo distribuido de distribución de librerías. El comportamiento distribuido es siempre desde el repositorio local de Maven hacia los repositorios que están publicados en Internet o en la red corporativa.

- ☑ Mecanismos para ser extensible, por la creación de plugins customizables.

☑ Es multi-plataforma, puede funcionar tanto en entornos Linux como Windows al ser una aplicación Java.

☑ Es software libre, con lo cual es el código está disponible, se podría modificar y customizar en caso de que fuera necesario.

Akka

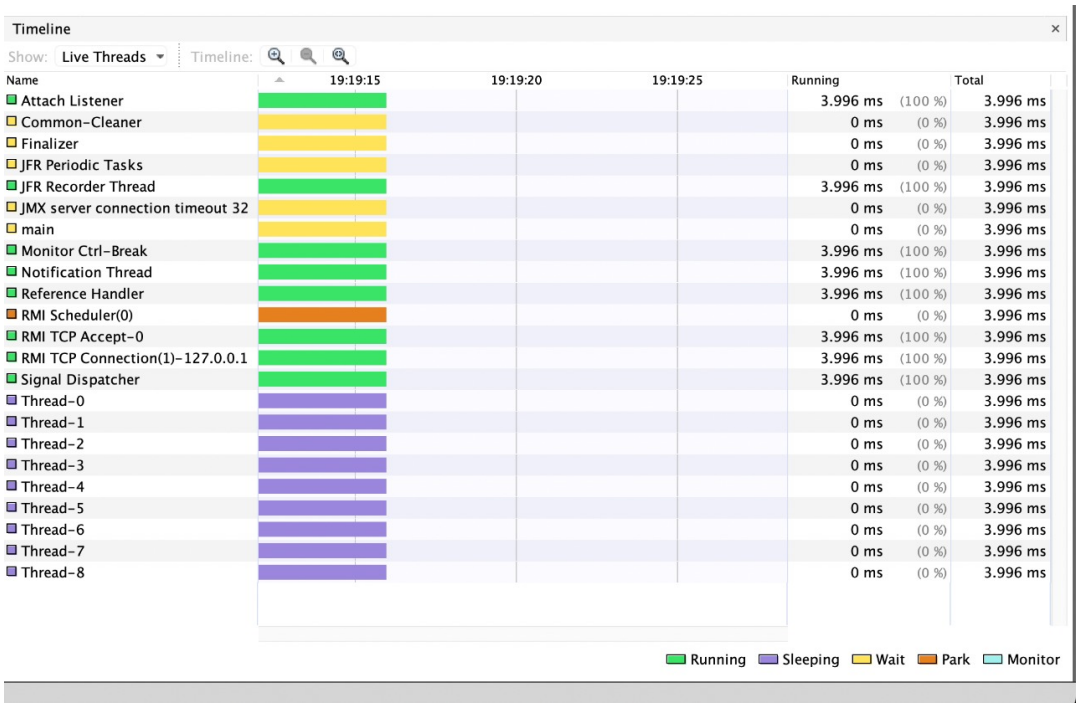
VisualVM

Es una herramienta que proporciona una interfaz visual para ver información detallada sobre aplicaciones Java mientras se ejecutan en una máquina virtual Java (JVM) y para solucionar problemas y crear perfiles de estas aplicaciones.

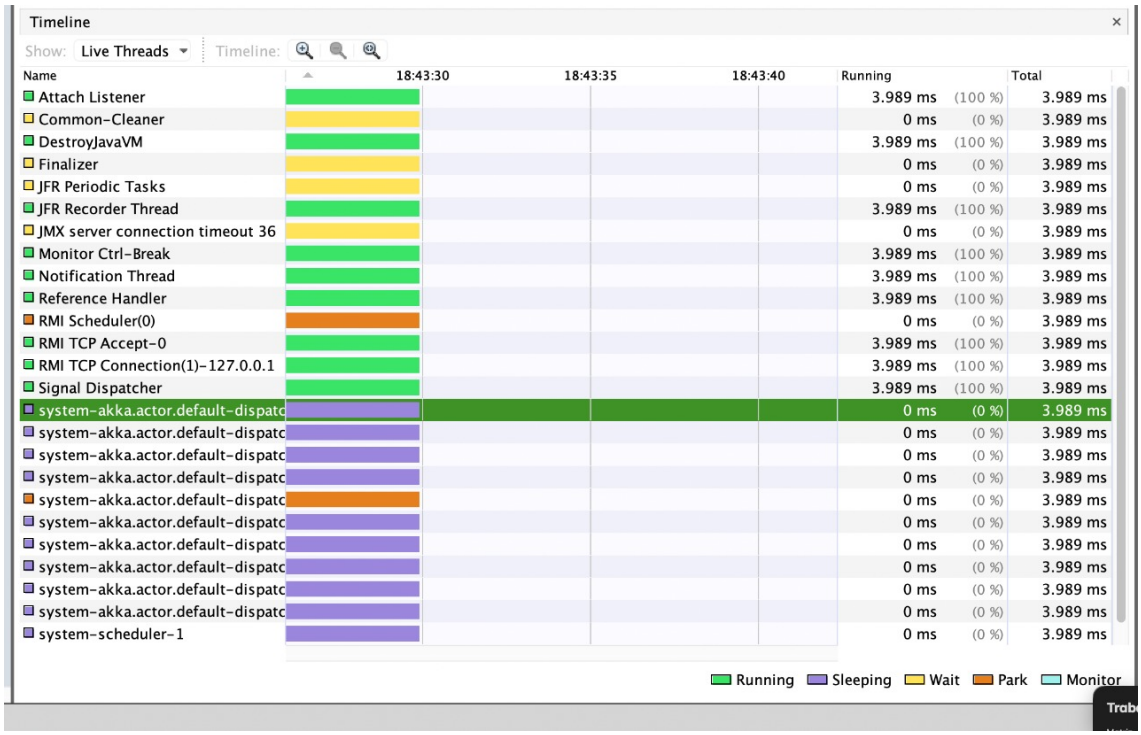
Los desarrolladores de aplicaciones Java pueden utilizar Java VisualVM para solucionar problemas de aplicaciones y para monitorear y mejorar el rendimiento de las aplicaciones.

🕑 Monitoreo de los subprocesos de la aplicación 🕑

🔄 Monitoreo ejecución de proyecto multiplicación de matrices con implementación de Threads Model 🔄



🔄 Monitoreo ejecucion de proyecto multiplicación de matrices con implementación de Actor Model 🔄



Mensajería de actores

Para interactuar entre sí, los actores pueden enviar y recibir mensaje de cualquier otro actor del sistema. Estos mensajes pueden ser cualquier tipo de objeto con la condición de que sea inmutable.

Es una buena práctica definir los mensajes dentro de la clase actor. Esto ayuda a escribir código que es fácil de entender y saber qué mensajes puede manejar un actor.

Enviando mensajes

Dentro del sistema de actores de Akka, los mensajes se envían utilizando métodos:

- tell()
- ask()
- forward()

Cuando queremos enviar un mensaje y no esperamos una respuesta, podemos usar el método tell().Este es el método más eficiente desde una perspectiva de rendimiento.

- El primer parámetro representa el mensaje que enviamos a la dirección del actor readActorRef.
- El segundo parámetro especifica quien es el remitente. Esto es útil cuando el actor que recibe el mensaje necesita enviar una respuesta a un actor que no sea el remitente (por ejemplo, el padre del actor que envía).
- Por lo general, podemos establecer el segundo parámetro en nulo o ActorRef.noSender(), porque no esperamos una respuesta. Cuando necesitamos una respuesta de un actor, podemos usar el método ask():

Recibir mensajes

Cada actor implementará el método createReceive(), que maneja todos los mensajes entrantes. ReceiveBuilder() actúa como una declaración de cambio, tratando de hacer coincidir el mensaje recibido con el tipo de mensajes definidos:

Matar a un actor

Cuando terminamos de usar un actor, podemos detenerlo llamando al método stop() desde la interfaz ActorRefFactory:

```
1 system.stop(myActorRef);
```

Podemos usar este método para terminar con cualquier actor secundario o con el propio actor.

Es importante tener en cuenta que la detención se realiza de forma asíncrona y que el procesamiento del mensaje actual finalizará antes de que finalice el actor. No se aceptarán más mensajes entrantes en el buzón del actor.

s importante tomar en cuenta las siguientes características al momento de implementar el código

Al detener a un actor principal, también enviaremos una señal de muerte a todos los actores secundarios que generó.

Cuando ya no necesitemos el sistema actor, podemos terminarlo para liberar todos los recursos y evitar pérdidas de memoria.

Esto detendrá a los actores guardianes del sistema, por lo tanto, a todos los actores definidos en este sistema Akka.

También podríamos enviar un mensaje de PoisonPill a cualquier actor que queramos matar:

```
1 myActorRef.tell(PoisonPill.getInstance(), ActorRef.noSender());
```

El actor recibirá el mensaje de PoisonPill como cualquier otro mensaje y lo pondrá en la cola. El actor procesará todos los mensajes hasta llegar al de PoisonPill. Solo entonces el actor comenzará el proceso de terminación.

Otro mensaje especial utilizado para matar a un actor es el mensaje Kill. A diferencia de PoisonPill, el actor lanzará una ActorKilledException al procesar este mensaje:

```
1 myActorRef.tell(Kill.getInstance(), ActorRef.noSender());
```

Metodos de Clase Threads

Métodos y descripciones más importantes de la clase Thread

Método	Descripción
currentThread	Devuelve una referencia al hilo que se esta ejecutando, es decir el objeto Thread que representa al hilo de ejecución actualmente
getName	Investiga el nombre de un hilo
yield	Este método hace que el interprete cambie de contexto entre el hilo actual y el siguiente hilo ejecutable disponible. Es una manera de asegurar que los hilos de menor prioridad no sufran inanición

Método	Descripción
getThreadGroup	Devuelve una referencia al grupo al que pertenece un hilo
getPriority	Investiga la prioridad de un hilo
interrup	Interrumpe la ejecución de un hilo
isAlive	Investiga si un hilo esta vivo
isInterrupted	Investiga si un hilo ha sido interrumpido
join	Mezcla dos hilos en hilo
run	Comienza a ejecutar el código destino de un hilo
setName	Cambia el nombre de un hilo
setPriority	Permite cambiar la prioridad de ejecución de un hilo
sleep	Cesa la ejecución de un hilo durante el tiempo especifica
start	Inicia la ejecución de un hilo, es decir, se llama al método run del destino
toString	Muestra los datos de un hilo
yield	Permite que el procesador ejecute otros hilos en lugar del hilo afectado
wait	Bloquea un objeto hasta que el hilo que ha tomado posesión del reciba la ejecución del método notify, por parte de un hilo con permiso para su desbloqueo
notifyAll	notifica a todos los hilos del grupo la orden de desbloqueo