Concurrencia

¡Comencemos!

¿Qué aprenderás?

- Entender el concepto de multiThreading.
- Conocer las reglas para servlets concurrentes.

Introducción

En el día a día podemos observar distintos procesos concurrentes, es decir, que suceden de forma simultánea en un mismo contexto: autos yendo y viniendo por una avenida, ascensores deteniéndose en distintos pisos, personas subiendo a un bus, etc.

Si pensamos a nivel de sistemas, también podremos identificar esos mismos sucesos. El ejemplo más sencillo es pensar en un sistema bancario, con cientos (sino miles) de personas haciendo transacciones al mismo tiempo, de diversa naturaleza.

Concurrencia es la tendencia de las cosas a producirse al mismo tiempo en un sistema. Uno de los desafíos más importantes para los programas que diseñemos será manejar estos eventos y coordinar sus interacciones.

El trabajo de los servlets dentro de un contenedor de aplicaciones es por regla general *multithreading* (multihilos o multiprocesos). Procesos multihilos nos indican que múltiples request pueden ser procesados por el mismo servlet en un mismo periodo de tiempo. En este capítulo vamos a ver lo necesario para tener la base de conocimiento para entender este tema.

¿Qué es la concurrencia?

Se habla de concurrencia cuando ocurren varios sucesos de manera contemporánea. Decimos que dos procesos son concurrentes, cuando son procesados al mismo tiempo, es decir, para ejecutar uno de ellos, no hace falta esperar la ejecución de otro.

Cuando tenemos un sistema multiprocesador, esta ejecución simultánea se puede conseguir de manera sencilla, asignando a cada CPU un proceso. Por otro lado, las interacciones entre procesos se vuelven más complejas.

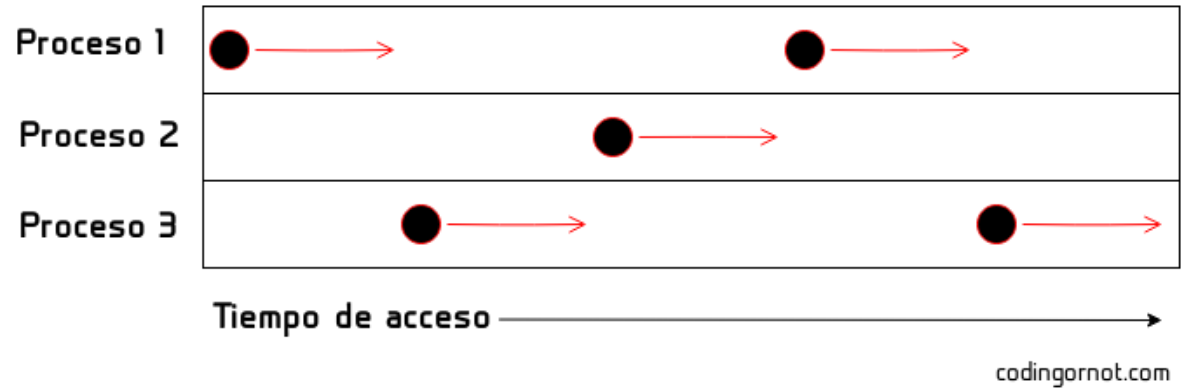

Imagen 1. Funcionamiento de la concurrencia.
Fuente: Desafío Latam

¿Qué es un hilo?

Un hilo es la unidad básica de ejecución del Sistema Operativo para la utilización del CPU y es el encargado de realizar los cálculos para que el programa se ejecute. Por definición, cada hilo posee: id de hilo, contador de programa, registros, stack.

El *multihilo* o *multithreading* es la capacidad para proporcionar múltiples hilos de ejecución al mismo tiempo.

¿Qué es la programación MultiHilos?

Para empezar a entender lo que significa Multihilos, vamos a pensar en nuestro sistema operativo. Nuestro windows-linux-mac es por definición multitarea, lo que significa que ahora mismo puedes estar reproduciendo una película en tu reproductor favorito además de un navegador web y por otro lado un archivo word trabajando en perfectas condiciones. Esto es el *multiThreading* (su traducción al inglés).

También podemos llamarlos multiprocesos ya que pueden ejecutar varios hilos de ejecución al mismo tiempo. Detrás de este concepto hay una serie de teorías que involucran un conocimiento profundo de procesamiento pero para resumir y entender el concepto nos quedaremos con esta información.

Trabajo multihilos con servlets

Entendiendo el concepto de multihilos, podemos pensar en nuestros servlets, los cuales también deben tener la capacidad de trabajar de forma multitarea sin perder sus capacidades.

Para asegurarnos que nuestro *servlet* cae en la categoría de thread safe, debemos verificar que nuestro diseño cumpla con unas reglas específicas:

1. El método *service()* del servlet no debe acceder a ninguna variable miembro (variables de instancias, las cuales son compartidas por todos los métodos, por ende si un método está usando una variable de tipo numérico que guarda el valor 1 y otro proceso la ocupa cambiandola a 2 el método que necesitaba el número 1 se caerá o causara un funcionamiento que no se esperaba), a menos que estas variables miembro sean seguras para subprocesos.

2. No debe reasignar las variables miembro, ya que esto puede afectar a otros subprocesos que se ejecutan dentro del método *service()*. Si realmente se necesita reasignar una variable miembro, asegúrate de que esto se haga dentro de un bloque sincronizado.

Las reglas 1 y 2 también cuentan para las variables estáticas.

3. Las variables locales siempre son seguras para los subprocesos. Sin embargo, ten en cuenta que el objeto al que apunta una variable local puede no ser así. Si el objeto fue instanciado dentro del método y nunca se escapa, no habrá problema. Por otro lado, una variable local que apunta a algún objeto compartido, todavía puede causar problemas. El hecho de que asigne un objeto compartido a una referencia local, no significa que el objeto se convierta automáticamente en seguro para subprocesos.

Para entender un poco mejor estas reglas, podemos decir que todo se resume en:

- Nunca utilizar atributos de clase en un *servlet*, ya que al ser de clases son compartidos y eso puede generar problemas cuando existen multiprocesos. Siempre utilizar variables locales en cada método, así nos aseguramos que son *thread safe* (no causan problemas ni ponen en riesgo la seguridad de la aplicación).

- La misma regla corre para variables estáticas, no son seguras por su naturaleza compartida entre hilos de procesos (o entre métodos).

- Los objetos de *request* y *response* son, por supuesto, seguros para el uso de subprocesos. Se crea una nueva instancia de estos para cada solicitud en su *servlet* y, por lo tanto, para cada subproceso que se ejecuta en su *servlet*.