Cliente HTTP asíncrono

Elaborada por: M. en C. Ukranio Coronilla

Visualizar el video:

Cliente HTTP en Java

https://www.youtube.com/watch?v=zc0QeO93ito

Verificar el correcto funcionamiento del código y comentarlo de acuerdo con la explicación del video.

Tip:

Para facilitar el proceso de compilación del cliente HTTP asíncrono su compañero Santiago Pérez Gómez elaboró el archivo `Makefile` adjunto. Sólo es necesario ubicar todos los archivos (tres del código fuente del cliente, y el archivo Makefile) en la misma carpeta y ejecutar `make` para compilar, finalmente `make run` para ejecutarlo y si se hacen cambios en el código se debe ejecutar `make clean` para borrar todo menos el código. Es posible que se tenga que instalar el programa `make` con: `sudo apt install make`

En el video se explica el código correspondiente a un cliente HTTP asíncrono (véase https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_asincr%C3%B3nica) el cual permite acceder a varios endpoints de manera concurrente, lo cual brinda mayores capacidades y mejor desempeño en solicitudes múltiples que el cliente síncrono.

Una herramienta fundamental para implementar la comunicación asíncrona en Java son los ***Future***. Un Future es un objeto que en algún momento futuro (no se sabe cuándo) va a contener la respuesta devuelta por un método. En Java 8 se mejoraron los Future dando pie a la clase ***CompletableFuture,*** que se incorpora en WebClient.java para obtener en algún momento futuro la respuesta proveniente del servidor. Observe en **`Aggregator.java`** que, aunque se envían todas las solicitudes dentro de un ciclo `for` mediante el método `sendTask`, este devuelve objetos

`CompletableFuture` que estarán disponibles en un futuro, aunque se desconoce cuándo.

Podemos evaluar continuamente todos los futuros para saber si alguno ya se ha concretado mediante el método `isDone()`, el cual regresa el valor true si la tarea ha sido completada (véase: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)

Para ilustrar lo anterior levante dos servidores HTTP con el endpoint `/searchtoken` a los cuales se les enviará una solicitud de búsqueda a cada uno con el cliente asíncrono, verificando que funcionen correctamente. Posteriormente incruste el siguiente código después de enviar **todas** las tareas de búsqueda (después del ciclo `for` que contiene el método `sendTAsk`) en el archivo *Aggregator.java* y aumente la cantidad de cómputo que deban realizar las tareas para que pueda visualizar las impresiones adicionales:

```
// Evalúa continuamente si uno de los servidores ha terminado.
boolean bandera = true;
while(bandera){
    for(int j = 0; j < 2; j++){
        System.out.println("futures["+j+"].isDone() = " + futures[j].isDone());
        if (true == futures[j].isDone())
            bandera = false;
    }
}
```

Ejercicio

Modifique **`Application.java`** para que el cliente envíe las siguientes cinco tareas tal y como se indica:

```
List<String> listaTareas = new ArrayList<String>();
listaTareas.add("1757600,IPN");
listaTareas.add("17576,SAL");
listaTareas.add("70000,MAS");
listaTareas.add("1757600,PEZ");
listaTareas.add("175700,SOL");
```

Las tareas se envían al endpoint `/searchtoken` manteniendo los dos servidores activos. También modifique **`Aggregator.java`** de modo que al inicio el servidor en el puerto 8081 resuelve la primera tarea (1757600,IPN) mientras que el servidor en el puerto 8082 la segunda tarea(17576,SAL). Al primer servidor que termine se le asigne la tercera tarea(70000,MAS). Y nuevamente al primero de los dos servidores que termine se le asigna la cuarta tarea(1757600,PEZ) y así sucesivamente, de manera que el conjunto de tareas se distribuye en los dos servidores manteniéndolos trabajando a ambos en todo momento. El cliente tendrá que ir imprimiendo como se

fueron haciendo las asignaciones de las tareas las cuales por el tamaño de cada tarea tendrán que coincidir con la siguiente secuencia de salida:

```
ukranio@HUAWEIR5:~/SD_2025_ENERO/CLASE_24_SOLUCION/cliente_original$ make run
java -cp Application.jar Application
Las tareas a resolver son las siguientes:
Tarea 0 :1757600,IPN
Tarea 1 :17576,SAL
Tarea 2 :70000,MAS
Tarea 3 :1757600,PEZ
Tarea 4 :175700,SOL

Al servidor http://localhost:8081/searchtoken se le asigna la tarea: 1757600,IPN
Al servidor http://localhost:8082/searchtoken se le asigna la tarea: 17576,SAL
    El servidor http://localhost:8082/searchtoken completó la tarea 17576,SAL
Al servidor http://localhost:8082/searchtoken se le asigna la tarea: 70000,MAS
    El servidor http://localhost:8082/searchtoken completó la tarea 70000,MAS
Al servidor http://localhost:8082/searchtoken se le asigna la tarea: 1757600,PEZ
    El servidor http://localhost:8081/searchtoken completó la tarea 1757600,IPN
Al servidor http://localhost:8081/searchtoken se le asigna la tarea: 175700,SOL
    El servidor http://localhost:8081/searchtoken completó la tarea 175700,SOL

Para la tarea 1757600,IPN El numero de apariciones es 105

Para la tarea 17576,SAL El numero de apariciones es 1

Para la tarea 70000,MAS El numero de apariciones es 6

Para la tarea 1757600,PEZ El numero de apariciones es 104

Para la tarea 175700,SOL El numero de apariciones es 10
```

Enviar código y captura de pantalla mostrando la ejecución de su código y realizando lo que se solicita.