Centro de Enseñanza Técnica Industrial

Plantel Colomos

Ingeniería en Desarrollo de Software

Nombre Alumno: José Rafael Ruiz Gudiño

Registro: 20110374

Computación Paralela

Actividad 2. Investigación Fork/Join - ExecutorService

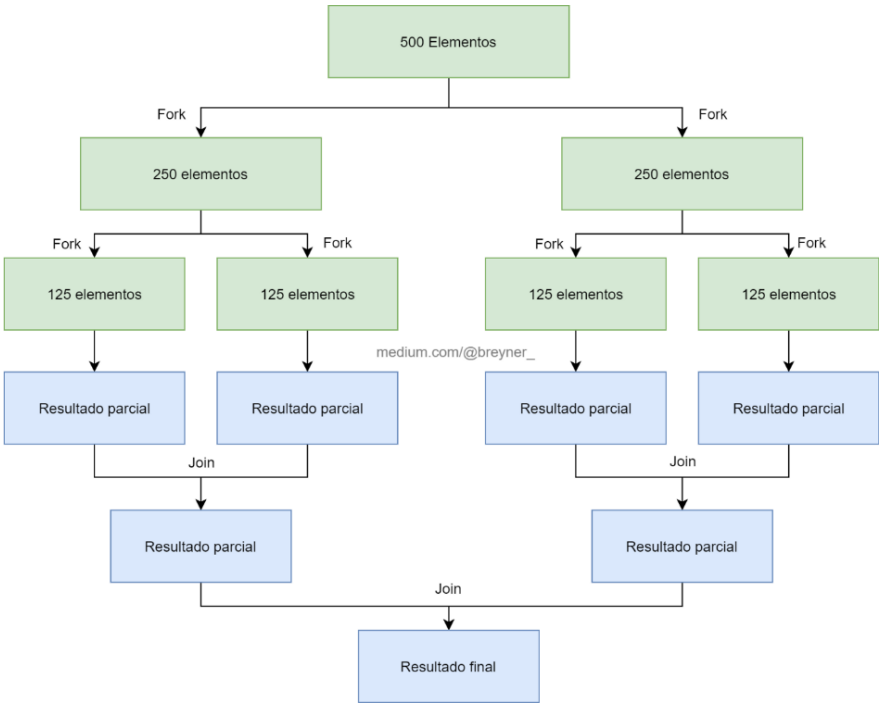
5°P

T/M

04/04/2022

Fork/Join

El marco Fork / Join es un marco de tareas de ejecución paralela proporcionado por Java7. La idea es descomponer tareas grandes en tareas pequeñas, y luego las tareas pequeñas pueden continuar descomponiéndose, y luego los resultados de cada tarea pequeña se calculan por separado y luego se combinan, y finalmente el resultado resumido se usa como resultado de una gran tarea. Para la división de tareas, se requiere que cada subtarea sea independiente entre sí y que pueda ejecutar tareas de forma independiente en paralelo, sin afectarse entre sí.



El centro del marco fork/join es la clase ForkJoinPool, una extensión de la clase AbstractExecutorService. ForkJoinPool implementa el algoritmo central de robo de trabajo y puede ejecutar procesos de ForkJoinTask.

Implementación

El primer paso para usar el marco fork/join es escribir código que realice un segmento del trabajo. Su código debe ser similar al siguiente pseudocódigo:

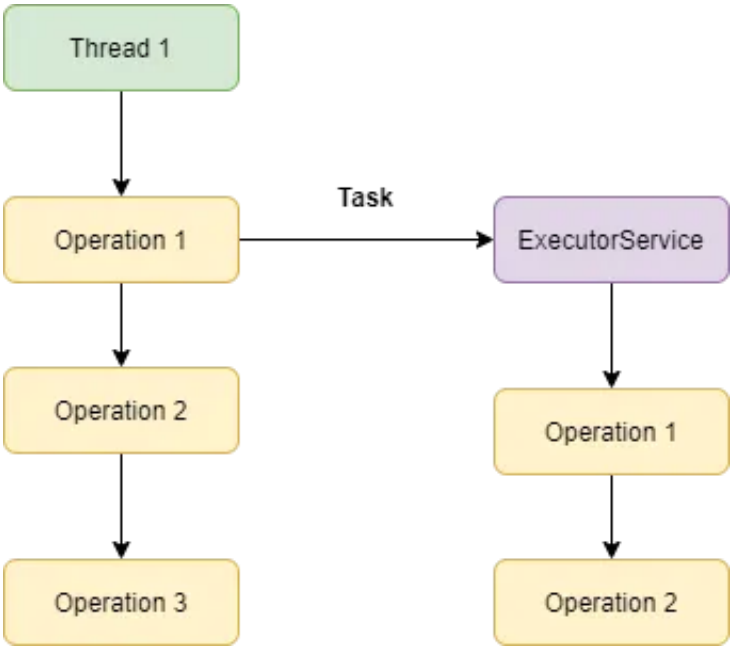
```
if(mi parte del trabajo es lo suficientemente pequeña)
  hacer el trabajo directamente
else
  dividir mi trabajo en dos partes
  invoca las dos piezas y espera los resultados
```

Envuelva este código en una subclase ForkJoinTask, normalmente usando uno de sus tipos más especializados, ya sea RecursiveTask (que puede devolver un resultado) o RecursiveAction (que no retorna).

Una vez que su subclase ForkJoinTask esté lista, cree el objeto que representa todo el trabajo a realizar y páselo al método de invoke() de una instancia de ForkJoinPool.

ExecutorService

ExecutorService en Java es una interfaz que forma parte del paquete java.util.concurrent. Esta utilidad de concurrencia de Java ayuda a ejecutar tareas asincrónicas al mismo tiempo. Usando la interfaz ExecutorService, podemos separar la creación de la tarea y el proceso de ejecución de la tarea. Es una subinterfaz del ExecutorFramework.



El propósito principal de un ExecutorService es administrar una serie de subprocesos y también asignar tareas a los subprocesos. En caso de que el número de tareas sea mayor que el número de subprocesos, pone en cola las tareas hasta que cualquier subproceso esté disponible para su ejecución.

Crear una instancia de ExecutorService

Podemos crear una instancia de ExecutorService de las siguientes 3 formas:

- Hilo único: Crea una instancia de un solo hilo usando un ExecutorService.
  ExecutorService exec = Executors.newSingleThreadExecutor();
- Grupo de subprocesos: Crea un grupo de subprocesos especificando el número de subprocesos en el parámetro
  ExecutorService exec = Executors.newFIxedThreadPool(int count);

- Grupo programado de subprocesos: Crea un grupo de subprocesos programado
  ExecutorService exec = Executors.newScheduledThreadPool(int count);

Asignar tarea usando el método execute() de Java ExecutorService

El siguiente ejemplo muestra cómo ejecutar una tarea asincrónica usando el execute() método del ExecutorService. En este ejemplo, creamos una instancia de un solo hilo usando newSingleThreadExecutor. La execute() El método toma el objeto Runnable como parámetro. Finalmente, después de la ejecución de la tarea, podemos verificar si ExecutorService está apagado usando el isShutDown() método.

```
public static void main(String[] args) {
  ExecutorService exec = Executors.newSingleThreadExecutor();
  exec.execute(new Runnable() {
    @Override
    public void run() {
      System.out.println("Example of execute method");
    }

  });
  exec.shutdown();
  System.out.println("Is ExecutorService Shutdown: " + exec.isShutdown());
}
```

Asignar tareas mediante el método submit () de Java ExecutorService

Este es un ejemplo del ExecutorService submit() método que acepta la tarea Ejecutable como parámetro.

```
public static void main(String[] args) {
  ExecutorService exec = Executors.newSingleThreadExecutor();
  exec.submit(new Runnable() {
    @Override
    public void run() {
      System.out.println("Example of submit method");

    }
  });
```

```
  exec.shutdown();
 }
```

Asignar tareas mediante el método ExecutorService invokeAny ()

Este ejemplo muestra cómo utilizar el invokeAny() método del Java ExecutorService. Necesitamos pasar el objeto o la acción invocable como parámetro a la invokeAny() método. Cualquiera que sea la acción invocable que ejecute, devuelve el resultado de la tarea correspondiente.

```
public static void main(String[] args) throws InterruptedException, ExecutionException {
  ExecutorService exec = Executors.newSingleThreadExecutor();
  Set<Callable<String>> c = new HashSet<Callable<String>>();

  c.add(new Callable<String>() {
   public String call() throws Exception {
    return "Callable Task1";
   }

  });

  c.add(new Callable<String>() {
   public String call() throws Exception {
    return "Callable Task2";
   }

  });

  String value = exec.invokeAny(c);
  System.out.println(value);

  exec.shutdown();
 }
```

Asignar tarea usando el método invokeAll () de ExecutorService

Este ejemplo muestra cómo utilizar la interfaz Java ExecutorService invokeAll() método que ejecuta todas las acciones invocables. Devuelve el resultado de la ejecución de la tarea en forma de objeto Future.

```
public static void main(String[] args) throws InterruptedException, ExecutionException {
  ExecutorService exec = Executors.newSingleThreadExecutor();
  Set<Callable<String>> c = new HashSet<Callable<String>>();

  c.add(new Callable<String>() {
   @Override
   public String call() throws Exception {
    return "Callable Task1";
   }

  });

  c.add(new Callable<String>() {
   @Override
   public String call() throws Exception {
    return "Callable Task2";
   }

  });

  List<Future<String>> l = exec.invokeAll(c);
  for(Future<String> f: l)
   System.out.println(f.get());
  exec.shutdown();
}
```

Puntos importantes sobre ExecutorService

- Siempre apagar ExecutorService una vez que se complete la tarea. Nunca mantenga vivo el ExecutorService no utilizado.

- Haga un uso adecuado de la capacidad de Threadpool mientras usa un grupo de hilos de longitud fija
- No podemos llamar al método get () de la interfaz Future después de cancelar la tarea. Esto arrojará una CancellationException.
- Utilice tiempos de espera relevantes donde sea necesario para evitar tiempos de bloqueo innecesarios.

Referencias

Code World. (s. f.). *[Java] Fork / Join en Java - Code World*. Recuperado 4 de abril de 2022, de https://www.codetd.com/es/article/11658928

Oracle. (s. f.). *Fork/Join (The Java™ Tutorials > Essential Java Classes > Concurrency)*. Java Documentation. Recuperado 4 de abril de 2022, de https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html

Tutorial Cup. (2021, 11 octubre). *ExecutorService en Java*. Recuperado 4 de abril de 2022, de https://www.tutorialcup.com/es/Java/Executorservice-en-java.htm