

Chuyên đề mở rộng

Advanced Concurrency and Parallelism

&

Java Class Design and Advanced Class Design

Mục tiêu

- ✓ Viết được mã nguồn thực thi bất đồng bộ với Future và Callables
- ✓ Viết chương trình cài đặt Fork/Join framework để thực thi nhiệm vụ song song, bất đồng bộ tận dụng ưu thế của vi xử lý đa lõi.

Tóm tắt

Future và Callables:

Nằm trong package *java.util.concurrent*, interface **Future** và **Callables** cung cấp giải pháp thực thi bất đồng bộ, theo dõi tiến độ nhiệm vụ được thực thi bởi luồng phụ. Đối tượng Future sử dụng để kiểm tra trạng thái của Callables và nhận kết quả trả về khi nhiệm vụ hoàn thành.

Fork/Join framework:

Fork/Join framework API đặt trong gói *java.util.concurrent*, gồm 4 lớp sau:

1. **ForkJoinTask<V>**: là lớp trừu tượng định nghĩa nhiệm vụ chạy bên trong ForkJoinPool.
2. **ForkJoinPool**: là một thread pool quản lý việc thực hiện ForkJoinTasks.
3. **RecursiveAction**: là lớp con của ForkJoinTask dành cho nhiệm vụ KHÔNG trả về giá trị.
4. **RecursiveTask<V>**: là lớp con của ForkJoinTask dành cho nhiệm vụ CÓ trả về giá trị.

Trong lớp ForkJoinTask<V> có 3 hàm:

1. final ForkJoinTask<V> **fork()**: gửi nhiệm vụ để thực hiện bất đồng bộ
2. final V **join()**: chờ đợi nhiệm vụ xong và trả về kết quả
3. final V **invoke()**: bắt đầu nhiệm vụ -> đợi nó kết thúc và trả về kết

quả.

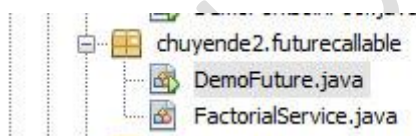
Thực thi ForkJoinTask trong ForkJoinPool bằng 2 cách:

1. `<T> T invoke(ForkJoinTask<T> task)`: thực thi nhiệm vụ và trả về kết quả khi hoàn thành, lời gọi này là đồng bộ - tức là nó sẽ chờ cho đến khi phương thức có kết quả trả về.
2. `void execute(ForkJoinTask<?> task)`: thực thi bất đồng bộ - cuộc gọi sẽ không chờ đợi nhiệm vụ hoàn thành mà tiếp tục thực hiện mã nguồn tiếp theo.

Bài thực hành số 1:

Yêu cầu: tạo lớp FactorialService thực thi interface Callables thực thi tính giai thừa của một số truyền vào từ constructor. Tạo lớp DemoFuture, trong đó viết mã truyền vào ExecutorService đối tượng Callable (2 đối tượng), gọi các hàm kiểm tra trạng thái và lấy kết quả trả về.

Code tham khảo:



FactorialService.java

```
package chuyende2.futurecallable;
```

```
import java.util.concurrent.Callable;
```

```
/**
```

```
 * Lớp tính giai thừa một số
```

```
 *
```

```
 * @author minhvufc
```

```
 */
```

```
public class FactorialService implements Callable<Long> {
```

```
    private int number;
```

```
    private int sleep;
```

```
    public FactorialService(int number, int sleep) {
```

```
        this.number = number;
```

```
        this.sleep = sleep;
```

```
    }
```

```
@Override
public Long call() throws Exception {
    return factorial();
}

/**
 * Hàm tính giai thừa dựa trên số number đã truyền ở hàm khởi tạo
 * @return số kiểu Long
 * @throws InterruptedException
 */
private Long factorial() throws InterruptedException {
    long result = 1;

    while (number != 0) {
        result = number * result;
        number--;
        Thread.sleep(sleep);
    }
    return result;
}
}
```

DemoFuture.java

```
package chuyende2.futurecallable;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author minhvuvc
 */
public class DemoFuture {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try {
            DemoFuture demo = new DemoFuture();
        }
    }
}
```

```
demo.demo();  
} catch (InterruptedException | ExecutionException ex) {  
    Logger.getLogger(DemoFuture.class.getName()).log(Level.SEVERE, null,  
ex);  
}  
}
```

```
private void demo() throws InterruptedException, ExecutionException {  
    ExecutorService executor = Executors.newFixedThreadPool(2); // Xử lý // 2  
    luồng đồng thời
```

```
    // Gửi nhiệm vụ tính Giai Thừa sang luồng phụ  
    System.out.println("Factorial Service called for 10!");  
    Future<Long> result10 = executor.submit(new FactorialService(10, 100));  
    System.out.println("Factorial Service called for 20!");  
    Future<Long> result20 = executor.submit(new FactorialService(20, 50));  
  
    System.out.println("Service (10!) is done: " + result10.isDone());  
    System.out.println("Service (20!) is done: " + result20.isDone());  
    System.out.println("Waiting in 5s...");  
    Thread.sleep(5000); // Trong lúc này thì công việc tính giai thừa ĐANG  
    THỰC HIỆN
```

```
    // In kết quả  
    System.out.println("10! = " + result10.get());  
    System.out.println("20! = " + result20.get());  
    /*  
        Phương thức get() là synchronous nên nó sẽ dừng chương trình để chờ đợi  
    kết quả  
    */
```

```
    // Chờ đợi  
    System.out.println("Waiting....."); // KHÔNG NHƯ KỲ VỌNG !  
  
    // Tắt service  
    executor.shutdown();  
  
}  
}
```

```

Output - ChuyenDe2.ParallelimsAndClassDesign (run)
run:
Factorial Service called for 10!
Factorial Service called for 20!
Service (10!) is done: false
Service (20!) is done: false
Waiting in 5s...

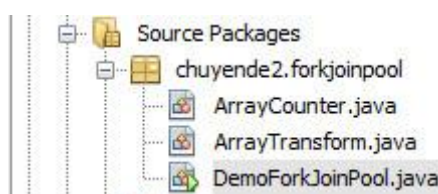
Output - ChuyenDe2.ParallelimsAndClassDesign (run)
run:
Factorial Service called for 10!
Factorial Service called for 20!
Service (10!) is done: false
Service (20!) is done: false
Waiting in 5s...
10! = 3628800
20! = 2432902008176640000
Waiting.....
BUILD SUCCESSFUL (total time: 5 seconds)

```

Bài thực hành số 2:

Yêu cầu: tạo 3 lớp như sau:

1. **ArrayTransform** kế thừa RecursiveAction: có nhiệm vụ biến đổi (nhân) giá trị phần tử trong mảng rất lớn với một số, cần chia nhỏ mảng thành các ngưỡng giới hạn để tính toán.
2. **ArrayCounter** kế thừa RecursiveTask: có nhiệm vụ đếm số chẵn trong mảng rất lớn, cần chia nhỏ mảng thành các ngưỡng giới hạn để đếm.
3. **DemoForkJoinPool** có hàm main(): tạo đối tượng ForkJoinPool khởi tạo với khai báo sử dụng tối đa số lõi vi xử lý (vd: 4), truyền vào đối tượng tạo bởi 2 lớp trên, chạy và so sánh thời gian với cách thực thi đơn luồng.



Code tham khảo:

File ArrayTransform.java

```
package chuyende2.forkjoinpool;

import java.util.concurrent.RecursiveAction;

/**
 * Biến đổi mảng (nhân phần tử mảng với một số)
 * @author minhvuvc
 */
public class ArrayTransform extends RecursiveAction {

    int[] array; // Mảng (vô cùng lớn)
    int number; // Số dùng để biến đổi
    int threshold = 100; // Ngưỡng
    int start; // Chỉ số mảng bắt đầu
    int end; // Chỉ số mảng kết thúc

    /**
     * Nhận vào tham số cho lớp
     * @param array Siêu mảng
     * @param number Số đem nhân với phần tử mảng
     * @param start Chỉ số mảng bắt đầu đầu
     * @param end Chỉ số mảng kết thúc
     */
    public ArrayTransform(int[] array, int number, int start, int end) {
        this.array = array;
        this.number = number;
        this.start = start;
        this.end = end;
    }

    @Override
    protected void compute() {
        if (end - start < threshold) {
            computeDirectly(); // Nhỏ hơn ngưỡng thì tính trực tiếp
        } else {
            int middle = (end + start) / 2;

            ArrayTransform subTask1 = new ArrayTransform(array, number, start,
middle);
            ArrayTransform subTask2 = new ArrayTransform(array, number, middle,
end);

            invokeAll(subTask1, subTask2); // Chia luôn 2 task
        }
    }
}
```

```
protected void computeDirectly() {  
    for (int i = start; i < end; i++) {  
        array[i] = array[i] * number;  
    }  
}
```

File ArrayCounter.java

```
package chuyende2.forkjoinpool;  
  
import java.util.concurrent.RecursiveTask;  
  
/**  
 *  
 * @author minhvufc  
 */  
public class ArrayCounter extends RecursiveTask<Integer> {  
  
    int[] array; // Mảng (vô cùng lớn)  
    int threshold = 100; // Ngưỡng  
    int start; // Chỉ số mảng bắt đầu  
    int end; // Chỉ số mảng kết thúc  
  
    /**  
     * Nhận vào tham số cho lớp  
     *  
     * @param array Siêu mảng  
     * @param start Chỉ số mảng bắt đầu đầu  
     * @param end Chỉ số mảng kết thúc  
     */  
    public ArrayCounter(int[] array, int start, int end) {  
        this.array = array;  
        this.start = start;  
        this.end = end;  
    }  
  
    protected Integer compute() {  
        if (end - start < threshold) {  
            return computeDirectly(); // Nhỏ hơn ngưỡng thì đếm trực tiếp  
        } else {  
            int middle = (end + start) / 2;
```

```
    ArrayCounter subTask1 = new ArrayCounter(array, start, middle);
    ArrayCounter subTask2 = new ArrayCounter(array, middle, end);

    invokeAll(subTask1, subTask2);

    return subTask1.join() + subTask2.join();
}
}
```

```
/**
 * Đếm số chẵn trong mảng
 *
 * @return số chẵn đếm được
 */
protected Integer computeDirectly() {
    Integer count = 0;

    for (int i = start; i < end; i++) {
        if (array[i] % 2 == 0) {
            count++;
        }
    }

    return count;
}
}
```

File DemoForkJoinPool.java

```
package chuyende2.forkjoinpool;

import java.util.Random;
import java.util.concurrent.ForkJoinPool;

/**
 *
 * @author minhvuvc
 */
public class DemoForkJoinPool {

    static final int SIZE = 100_000_000; // Mảng 100 triệu phần tử
    static int[] array = randomArray();
    int number = 9;

    /**
     * @param args the command line arguments
     */
}
```



```

*/
public static void main(String[] args) {

    System.out.println("First 10 elements of the array before: ");
    print();

    long from = System.currentTimeMillis();

    // Demo
    DemoForkJoinPool demo = new DemoForkJoinPool();
    // demo.demoRecursiveAction(); // Chậm hơn ~179ms
    // demo.demoTransform(); // Nhanh hơn ~57ms

    // demo.demoRecursiveTask(); // Nhanh hơn ~452ms
    demo.demoCounter(); // Chậm hơn ~790ms

    System.out.println("First 10 elements of the array after: ");
    print();
    long to = System.currentTimeMillis();
    System.out.println("Time: " + (to - from) + "ms");
}

private void demoRecursiveAction() {
    ArrayTransform mainTask = new ArrayTransform(array, number, 0, SIZE);
    ForkJoinPool pool = new ForkJoinPool(4);
    pool.invoke(mainTask);
}

private void demoRecursiveTask() {
    ArrayCounter mainTask = new ArrayCounter(array, 0, SIZE);
    ForkJoinPool pool = new ForkJoinPool(4);
    Integer evenNumberCount = pool.invoke(mainTask);

    System.out.println("Number of even numbers: " + evenNumberCount);
}

private void demoTransform() {
    for (int i = 0; i < SIZE; i++) {
        array[i] = array[i] * number;
    }
}

private void demoCounter() {
    Integer count = 0;

```

```
for (int i = 0; i < SIZE; i++) {
    if (array[i] % 2 == 0) {
        count++;
    }
}

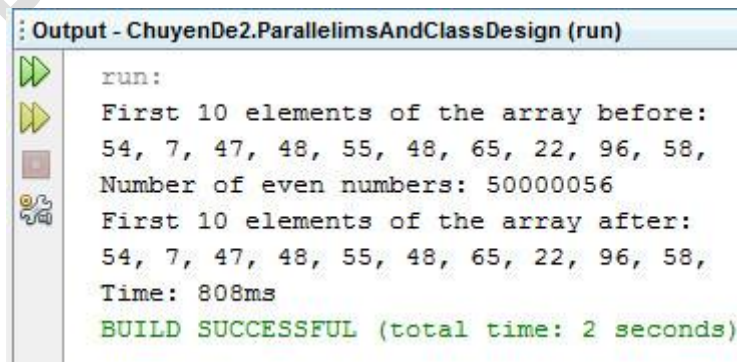
System.out.println("Number of even numbers: " + count);
}

/**
 * Hàm sinh số ngẫu nhiên cho Siêu mảng
 *
 * @return
 */
static int[] randomArray() {
    int[] array = new int[SIZE];
    Random random = new Random();

    for (int i = 0; i < SIZE; i++) {
        array[i] = random.nextInt(100);
    }

    return array;
}

static void print() {
    for (int i = 0; i < 10; i++) {
        System.out.print(array[i] + ", ");
    }
    System.out.println();
}
}
```



```
Output - ChuyenDe2.ParallelimsAndClassDesign (run)
run:
First 10 elements of the array before:
54, 7, 47, 48, 55, 48, 65, 22, 96, 58,
Number of even numbers: 50000056
First 10 elements of the array after:
54, 7, 47, 48, 55, 48, 65, 22, 96, 58,
Time: 808ms
BUILD SUCCESSFUL (total time: 2 seconds)
```