챕터 7

ExcutorService 부터 ForkJoin의 확장까지.

A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most ForkJoinTasks), as well as when many small tasks are submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, ForkJoinPools may also be appropriate for use with event-style tasks that are never joined.

순서

1. 스레드풀의 ExcutorService 인터페이스를 구현한다.

2. Work - Stealing 알고리즘 적용

3. Fork -Join 프레임워크 적용

4. parellelStream과의 비교

ExcutorService

src

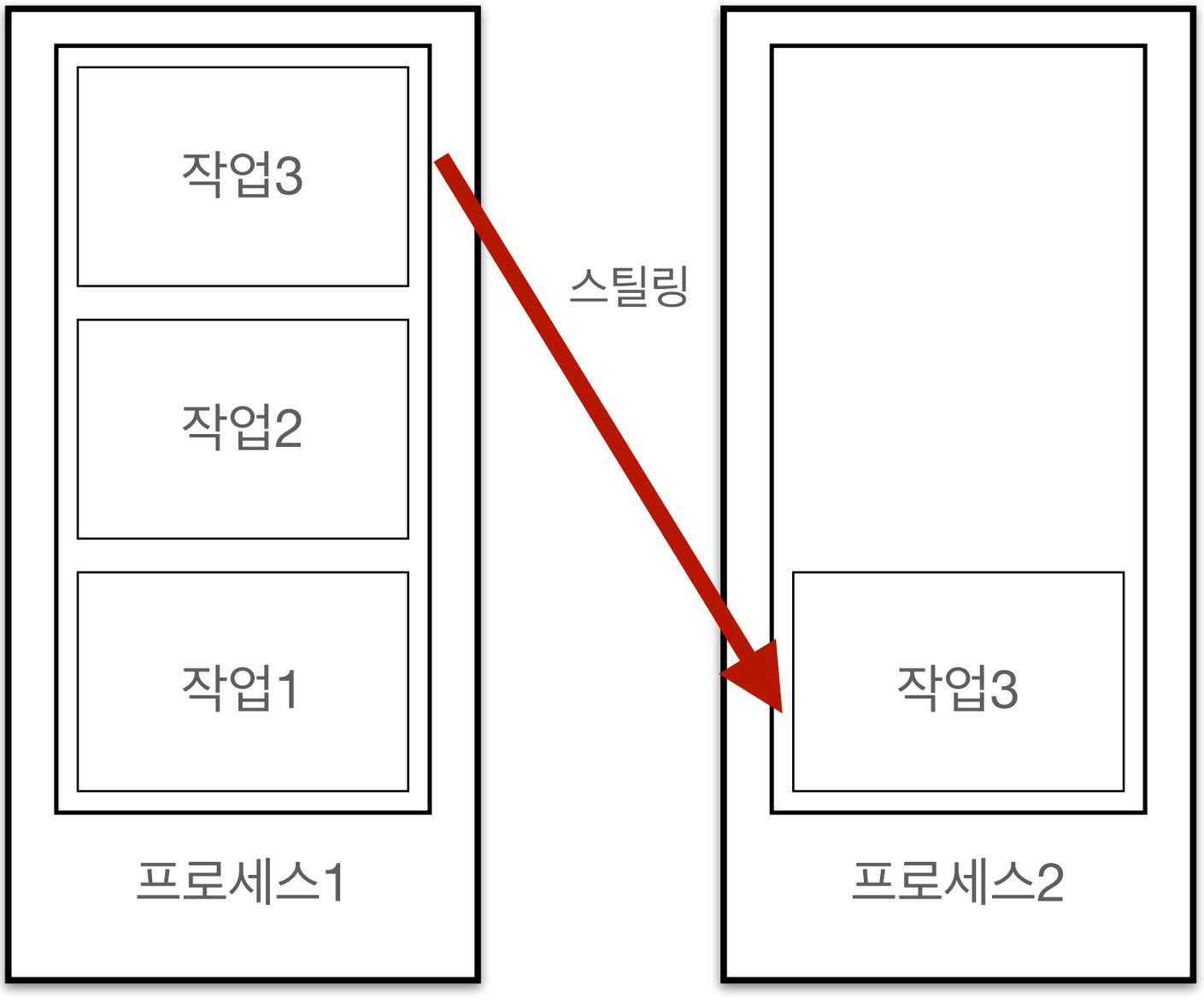
```java
public class Main {
    public static void main(String[] args) {
        ExecutorService exec = Executors.newFixedThreadPool( nThreads: 3);
        exec.submit(()->{
            String thread1= Thread.currentThread().getName();
            sleep();
            System.out.println("thread1 = " + thread1);
        });
        exec.submit(()->{
            String thread2= Thread.currentThread().getName();
            sleep();
            System.out.println("thread2 = " + thread2);
        });
        exec.submit(()->{
            String thread3= Thread.currentThread().getName();
            sleep();
            System.out.println("thread3 = " + thread3);
        });

        exec.shutdown();
```

output

```
thread3 = pool-1-thread-3
thread2 = pool-1-thread-2
thread1 = pool-1-thread-1
```

Work Stealing Algorithm



ExecutorService - (Work-Stealing)

src

```java
public class Main {
    public static void main(String[] args) throws InterruptedException {
        ExecutorService exec = Executors.newWorkStealingPool();
        exec.submit(()->{
            String thread1= Thread.currentThread().getName();
            sleep();
            System.out.println("thread1 = " + thread1);
        });
        exec.submit(()->{
            String thread2= Thread.currentThread().getName();
            sleep();
            System.out.println("thread2 = " + thread2);
        });
        exec.submit(()->{
            String thread3= Thread.currentThread().getName();
            sleep();
            System.out.println("thread3 = " + thread3);
        });
        int processor = Runtime.getRuntime().availableProcessors();
        System.out.println("available processor = " + processor);

        exec.awaitTermination( timeout: 3, TimeUnit.SECONDS);
        exec.shutdownNow();
    }
```

output

```
available processor = 10
thread2 = ForkJoinPool-1-worker-2
thread3 = ForkJoinPool-1-worker-3
thread1 = ForkJoinPool-1-worker-1
```

Fork-Join Framework

src

```
public static void main(String[] args) throws InterruptedException {
    ForkJoinPool exec = new ForkJoinPool();
    exec.submit(()->{
        String thread1= Thread.currentThread().getName();
        sleep();
        System.out.println("thread1 = " + thread1);
    });
    exec.submit(()->{
        String thread2= Thread.currentThread().getName();
        sleep();
        System.out.println("thread2 = " + thread2);
    });
    exec.submit(()->{
        String thread3= Thread.currentThread().getName();
        sleep();
        System.out.println("thread3 = " + thread3);
    });
    int processor = Runtime.getRuntime().availableProcessors();
    System.out.println("available processor = " + processor);

    exec.awaitTermination( timeout: 3,TimeUnit.SECONDS);
    exec.shutdownNow();
}
```

output

```
available processor = 10
thread1 = ForkJoinPool-1-worker-1
thread3 = ForkJoinPool-1-worker-3
thread2 = ForkJoinPool-1-worker-2
```

parallelStream vs ForkJoin

src

```
1개 사용 위치
public static void parallelStreamUsing(){
    List<Integer> numList = Arrays.asList(1,2,3,4,5,6,7,8,9,10,11,12);
    numList.parallelStream().forEach(index -> {
        System.out.println("Thread : " + Thread.currentThread().getName()
                + ", index + " + new Date());
        try{
            Thread.sleep( millis: 5000);
        }
        catch (InterruptedException ignored){
        }
    });
}
1개 사용 위치
public static void forkJoinPoolUsing() throws ExecutionException {
    List<Integer> numList = Arrays.asList(1,2,3,4,5,6,7,8,9,10,11,12);

    try{
        new ForkJoinPool().submit(() -> {
            numList.parallelStream().forEach(index -> {
                System.out.println("Thread : " + Thread.currentThread().getName()
                        + ", index + " + new Date());
                try{
                    Thread.sleep( millis: 5000);
                }
                catch (InterruptedException ignored){
                }
            });
        }).get();
    }catch (InterruptedException e){
        throw new RuntimeException(e);
    }
}
```

output (parallelStream)

```
Thread : ForkJoinPool.commonPool-worker-9, i
Thread : main, index + Sat Feb 04 22:45:51 K
Thread : ForkJoinPool.commonPool-worker-4, i
10026parellelstream milliseconds
```

output (forkJoin)

```
Thread : ForkJoinPool-1-worker-6, index + Sat
Thread : ForkJoinPool-1-worker-4, index + Sat
Thread : ForkJoinPool-1-worker-8, index + Sat
10292forkjoin milliseconds
```

JMH

src

```java
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@Fork(value = 1,jvmArgs = {"-Xms4G","-Xmx4G"})
@Warmup(iterations = 3)
@Measurement(iterations = 3)
public class TestClass {
    2개 사용 위치
    private static final int MAX_VALUE = 1000;
    @Benchmark
    public long iterativeSum() {
        long result = 0;
        for (int i = 0; i <= MAX_VALUE; i++) {
            result += i;
            slowDown();
        }
        return result;
    }
    @Benchmark
    public long parallelSum() {
        return LongStream.iterate( seed: 1L, i -> i + 1)
                .limit(MAX_VALUE)
                .parallel()
                .peek(i -> slowDown())
                .reduce(Long::sum).getAsLong();

    }
```

src(Additional)

```java
private void slowDown() {
    try {
        TimeUnit.MICROSECONDS.sleep( timeout: 10L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

output

```
Benchmark                  Mode  Cnt     Score     Error  Units
TestClass.iterativeSum     avgt    3  1328.485 ± 358.622  ms/op
TestClass.parallelSum      avgt    3   145.049 ±  14.244  ms/op
```