Java Concurrency in Practice

Chapter-6:
Completing tasks

Upcode Software
Engineer Team

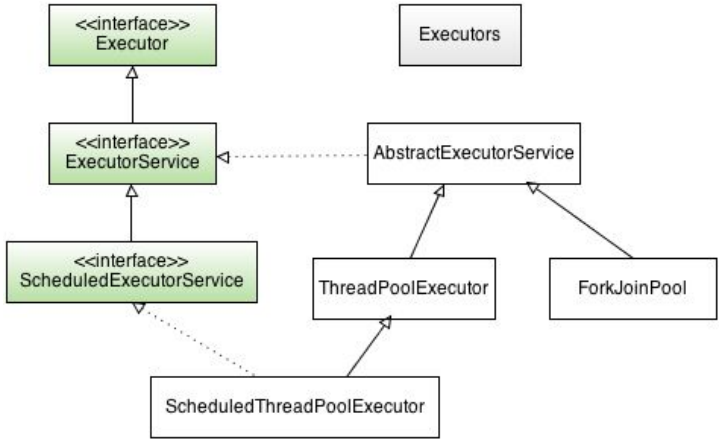
-2023-

CONTENT

1. What is Executor Framework.
2. Types of Executors.
3. SingleThreadExecutor.
4. FixedThreadPool.
5. CachedThreadPool.
6. ScheduledExecutor.

What is Executor Framework (1/n)

- A framework having a bunch of components that are used for managing worker threads efficiently is referred to as **Executor Framework**.
- In order to use the executor framework, we have to create a thread pool for executing the task by submitting that task to that thread pool.
- The **java.util.concurrent.Executors** class provides a set of methods for creating **ThreadPools** of worker threads.

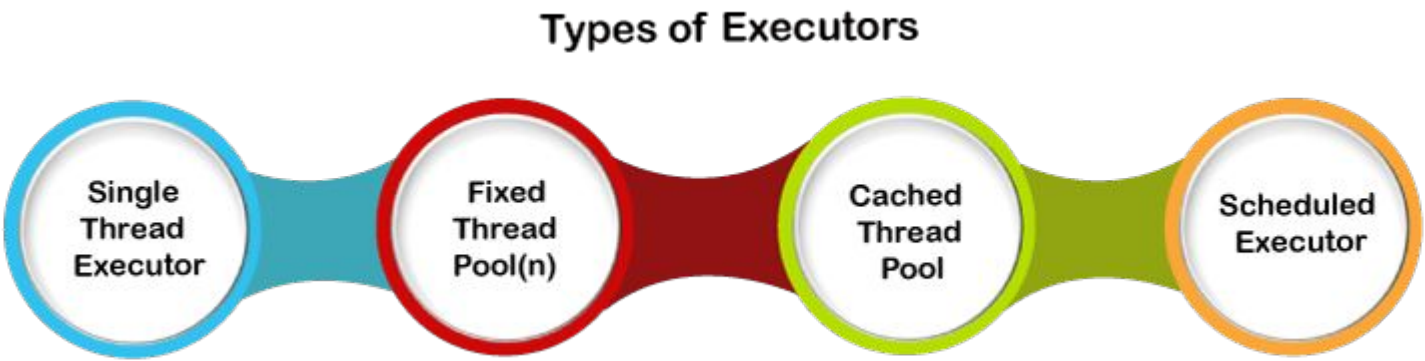


What is Executor Framework (2/n)

- Now, the question which comes to our mind is why we have to create such thread pools when we already have **the java.lang.Thread** class for creating an object and **Runnable/Callable** interface for achieving parallelism by implementing them.
- We need to create a large number of threads for adding a new thread without any throttling for each and every process. Due to which it requires more memory and cause wastage of resource. When each thread is swapped, the CPU starts to spend too much time.
- When we create a new thread for executing a new task cause overhead of thread creation. In order to manage this thread life-cycle, the execution time increase respectively.

Types of Executors

- In Java, there are different types of executors available which are as follows:



SingleThreadExecutor

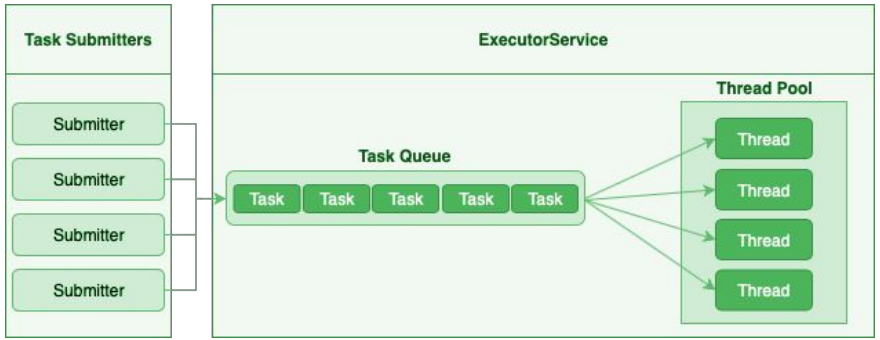
- The **SingleThreadExecutor** is a special type of executor that has only a single thread. It is used when we need to execute tasks in sequential order.
- In case when a thread dies due to some error or exception at the time of executing a task, a new thread is created, and all the subsequent tasks execute in that new one.

```
ExecutorService executor = Executors.newSingleThreadExecutor()
```

FixedThreadPool

- **FixedThreadPool** is another special type of executor that is a thread pool having a fixed number of threads.
- By this executor, the submitted task is executed by the n thread. In case when we need to execute more tasks after submitting previous tasks, they store in the **LinkedBlockingQueue** until previous tasks are not completed.
- The n denotes the total number of thread which are supported by the underlying processor.

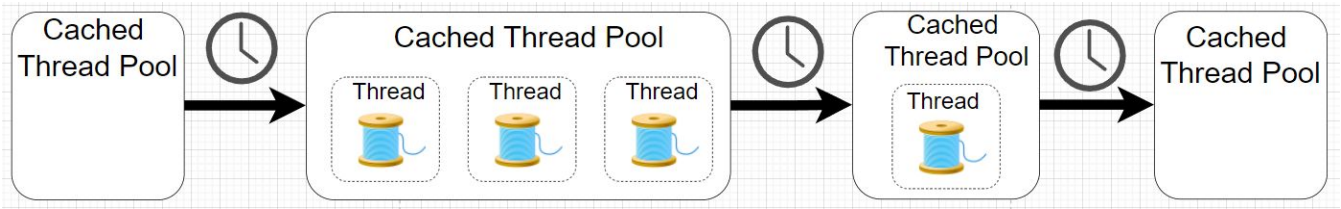
```
ExecutorService executor = Executors.newFixedThreadPool(n);
```



CachedThreadPool

- The **CachedThreadPool** is a special type of thread pool that is used to execute short-lived parallel tasks.
- The cached thread pool doesn't have a fixed number of threads. When a new task comes at a time when all the threads are busy in executing some other tasks, a new thread creates by the pool and add to the executor.
- When a thread becomes free, it carries out the execution of the new tasks. Threads are terminated and removed from the cached when they remain idle for sixty seconds.

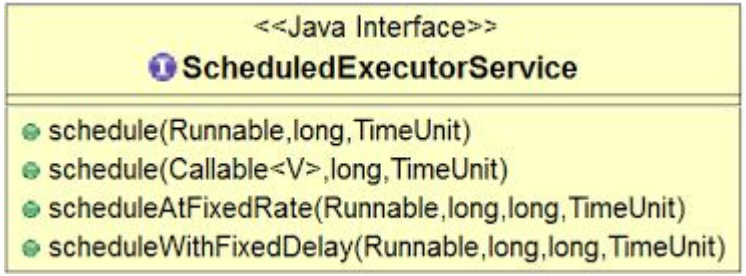
```
ExecutorService executor = Executors.newCachedThreadPool();
```



ScheduledExecutor (1/n)

- The **ScheduledExecutor** is another special type of executor which we use to run a certain task at regular intervals. It is also used when we need to delay a certain task.

```
ScheduledExecutorService scheduledExecService = Executors.newScheduledThreadPool(1);
```



ScheduledExecutor (2/n)

- The **scheduleAtFixedRate** and **scheduleWithFixedDelay** are the two methods that are used to schedule the task in **ScheduledExecutor**.
- The **scheduleAtFixedRate** method executes the task with a fixed interval when the previous task ended. The **scheduleWithFixedDelay** method starts the delay count after the current task complete.
- The main difference between these two methods is their interpretation of the delay between successive executions of a scheduled job. Both the methods are used in the following way:

```
scheduledExecService
        .scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit);
```

```
scheduledExecService
        .scheduleWithFixedDelay(Runnable command, long initialDelay, long period, TimeUnit unit);
```

Sample Code (1/n)

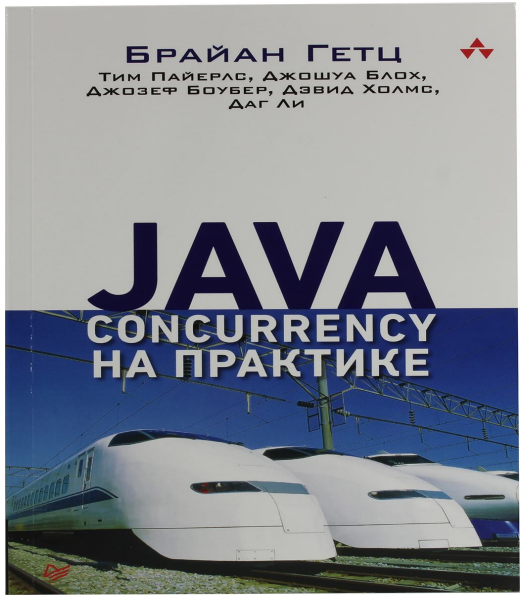
```java
public class Task1 implements Runnable{
    3 usages
    private String threadNo;
    2 usages
    public Task1(String no) {
        this.threadNo = no;
    }
    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName() + "start execution. ThreadNo = " + threadNo);
        processThread();
        System.out.println(Thread.currentThread().getName() + "stop execution.");
    }
    1 usage
    private void processThread() {
        try {
            Thread.sleep( millis: 1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    public String toString() {
        return "Task1{" +
                "threadNo='" + threadNo + '\'' +
                '}';
    }
}
```

Sample Code (2/n)

```
public class SimpleExecution {
    no usages
    public static void main(String[] args) {
        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 5);
        for (int i = 0; i < 5; i++) {
            Runnable task = new Task1( no: "" + i);
            executorService.execute(task);
        }
        executorService.shutdown();
        while (executorService.isTerminated()) {
            System.out.println("is it working ...");
        }
        System.out.println("Finished all threads");
    }
}
```

```
pool-1-thread-5start execution. ThreadNo = 4
pool-1-thread-2start execution. ThreadNo = 1
pool-1-thread-4start execution. ThreadNo = 3
pool-1-thread-3start execution. ThreadNo = 2
pool-1-thread-1start execution. ThreadNo = 0
pool-1-thread-2stop execution.
pool-1-thread-4stop execution.
pool-1-thread-5stop execution.
pool-1-thread-3stop execution.
pool-1-thread-1stop execution.
Finished all threads
```

Resources


Брайан Гетц
Тим Пайерлс, Джошуа Блох,
Джозеф Боубер, Дэвид Холмс,
Даг Ли
JAVA
CONCURRENCY
НА ПРАКТИКЕ

Reference

1. Java Concurrency book.
2. Executor Frameworks.
3. Executor Service.

Thank you!

Presented by

Sherjonov Jahongir

(jakhongirsherjonov@gmail.com)