Executors

Concurrent and parallel programming

Lecture 7.
Academic year: 2018/19

1

1

Executor

- Executor – a mechanism that allows to run various tasks with the use of separate threads.

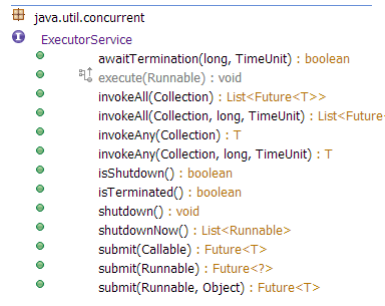
```
public interface Executor {
  void execute(Runnable);
}
```

2

2

ExecutorService interface

```
public interface ExecutorService extends
Executor {
  void shutdown();
  List<Runnable> shutdownNow();
  boolean isShutdown();
  boolean isTerminated();
  boolean awaitTermination(long timeout,
                           TimeUnit unit);
  //  additional methods not listed
}
```



3

3

Executor's Factory

java.util.concurrent

Class Executors

java.lang.Object
    java.util.concurrent.Executors

static ExecutorService	newSingleThreadExecutor() Creates an Executor that uses a single worker thread operating off an unbounded queue.
static ExecutorService	newFixedThreadPool(int nThreads) Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.

Example:

```
1  ExecutorService executor = Executors.newFixedThreadPool(10);
```

4

4

Types of tasks

- Represented by ***Runnable*** interface – can't deliver results
- Represented by ***Callable*** interface – can deliver results

```
public interface Callable<V> {
    V call() throws Exception;
}
```

5

5

Commands for starting task execution

The ***execute()*** method is *void*, and it doesn't give any possibility to get the result of task's execution or to check the task's status (is it running or executed).

```
executorService.execute(runnableTask);
```

submit() submits a *Callable* or a *Runnable* task to an *ExecutorService* and returns a result of type *Future*.

```
Future<String> future =
  executorService.submit(callableTask);
```

invokeAny() assigns a collection of tasks to an *ExecutorService*, causing each to be executed, and returns the result of a successful execution of one task (if there was a successful execution).

```
String result = executorService.invokeAny(callableTasks);
```

invokeAll() assigns a collection of tasks to an *ExecutorService*, causing each to be executed, and returns the result of all task executions in the form of a list of objects of type *Future*.

```
List<Future<String>> futures = executorService.invokeAll(callableTasks);
```

Source: https://www.baeldung.com/java-executor-service-tutorial

6

6

Example 1

```
class MyTask implements Runnable {
    private char c;
    public MyTask(char c) {
        this.c = c;
    }
    public void run() {
        for (int i = 1; i <= 10;i++) {
            System.out.print(c);
            try {
                Thread.sleep(500);
            }
            catch(Exception e) {}

        }
    }
}
```

Task represented by Runnable interface

7

7

Example 1

```
public class ExecutorTest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ExecutorService execl = Executors.newSingleThreadExecutor();
        execl.execute(new MyTask('a'));
        execl.execute(new MyTask('b'));
        execl.shutdown();
    }
}
```

Executor creation

Executor closing

```
run:
aaaaaaaaaabbbbbbbbbbBUILD SUCCESSFUL (total time: 10 seconds)
```

8

8

Example 2

```
public class ExecutorTest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ExecutorService exec2 = Executors.newFixedThreadPool(2);
        exec2.execute(new MyTask('c'));
        exec2.execute(new MyTask('d'));
        exec2.shutdown();

    }
}
```

```
run:
cddccddccdcdcdcddccdBUILD SUCCESSFUL (total time: 5 seconds)
```

9

9

Example 3

```
public class ExecutorTest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ExecutorService exec3 = Executors.newFixedThreadPool(2);
        exec3.execute(new MyTask('e'));
        exec3.execute(new MyTask('f'));
        exec3.execute(new MyTask('g'));
        exec3.execute(new MyTask('h'));
        exec3.shutdown();
    }
}
```

```
run:
effefefeeffeefefefefghhghgghhghghgghghghBUILD SUCCESSFUL (total time: 10 seconds)
```

10

10

Result returning

```
public interface Callable<V> {
    V call() throws Exception;
}
```

Task definition

```
Future<String> future =
  executorService.submit(callableTask);
```

Task execution

```
public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning)
    V       get();
    V       get(long timeout, TimeUnit unit);
    boolean isCancelled();
    boolean isDone();
}
```

Result description

11

11

Result returning

```
Future<String> future = executorService.submit(callableTask);
String result = null;
try {
    result = future.get();
} catch (InterruptedException | ExecutionException e) {
    e.printStackTrace();
}
```

Result getting

```
Future future = ... // Get Future from somewhere

if(future.isDone()) {
    Object result = future.get();
} else {
    // do something else
}
```

Checking of result's availability

12

12

Example 4

```
class MyResults {
    private String s;
    void setResult(String s) {
        this.s = s;
    }
    String getResult() {
        return s;
    }
}

class MyTask implements Callable<MyResults> {
    private String taskName;
    private int sleepTime;
    MyTask(String taskName,int sleepTime) {
        this.taskName = taskName;
        this.sleepTime = sleepTime;
    }
    public MyResults call() throws Exception {
        MyResults res = new MyResults();
        try {
                Thread.sleep(sleepTime);
        }
        catch(Exception e) {}
        res.setResult(taskName + ": " + String.valueOf(System.currentTimeMillis()));
        return res;
    }
}
```

13

13

Example 4

```
public class FutureTest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        ExecutorService exec = Executors.newFixedThreadPool(2);
        // TODO code application logic here
        Future<MyResults> f1 = exec.submit(new MyTask("Task 1",5000));
        Future<MyResults> f2 = exec.submit(new MyTask("Task 2",2500));
        Future<MyResults> f3 = exec.submit(new MyTask("Task 3",1500));

        exec.shutdown();

        if (f1.isDone()) {
            System.out.println("f1 - done");
        }
        else {
            System.out.println("f1 - not done");
        };
```

14

14

Example 4

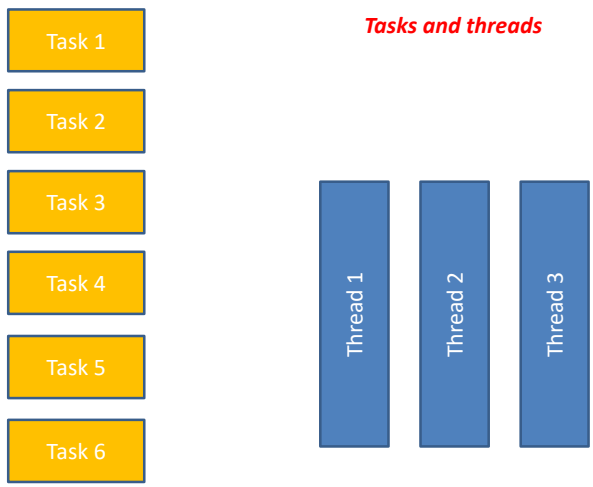
```
        if (f2.isDone()) {
            System.out.println("f2 - done");
        }
        else {
            System.out.println("f2 - not done");
        };

        if (f3.isDone()) {
            System.out.println("f3 - done");
        }
        else {
            System.out.println("f3 - not done");
        };

        MyResults res= null;

        try {
             res = f1.get();
        }
        catch(Exception e){};


        System.out.println(res.getResult());

    }

}
```

```
run:
f1 - not done
f2 - not done
f3 - not done
Task 1: 1545164033733
BUILD SUCCESSFUL (total time: 5 seconds)
```

15

15

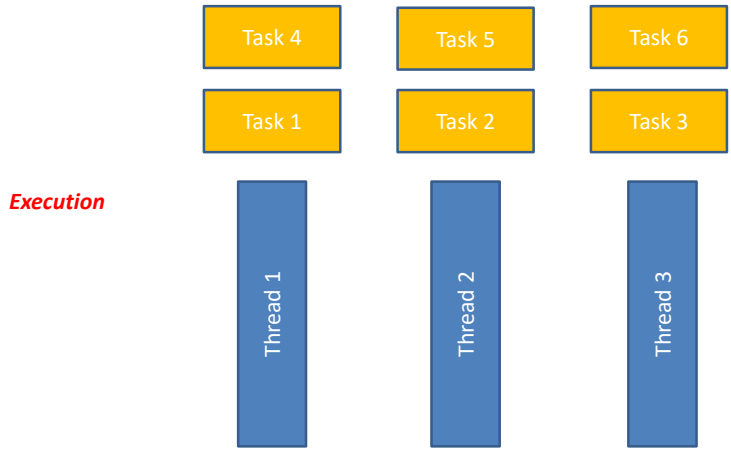
Execution of sets of tasks



16

16

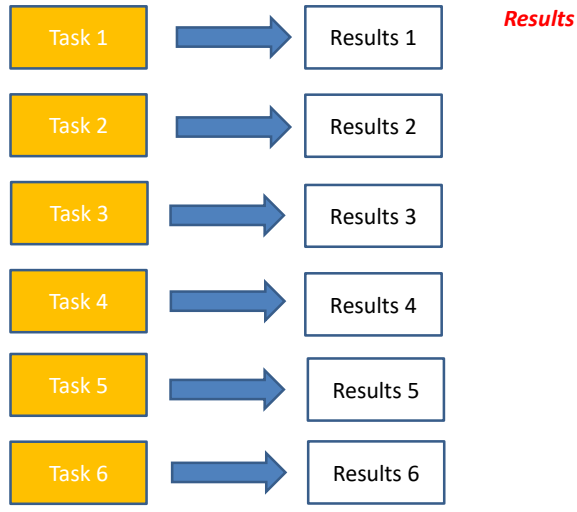
Execution of sets of tasks



17

17

Execution of sets of tasks



18

18

Example 5

```
class MyResults {
    private String s;
    void setResult(String s) {
        this.s = s;
    }
    String getResult() {
        return s;
    }
}

class MyTask implements Callable<MyResults> {
    private String taskName;
    private int sleepTime;
    MyTask(String taskName,int sleepTime) {
        this.taskName = taskName;
        this.sleepTime = sleepTime;
    }
    public MyResults call() throws Exception {
        MyResults res = new MyResults();
        try {
                Thread.sleep(sleepTime);
        }
        catch(Exception e) {}
        res.setResult(taskName + ": " + String.valueOf(System.currentTimeMillis()));
        return res;
    }
}
```

19

19

Example 5

```
public class InvokeAllTest {

    public static void main(String[] args) {

        ExecutorService exec = Executors.newFixedThreadPool(4);
        Set<Callable<MyResults>> tasks = new HashSet<Callable<MyResults>>();
        for (int i = 1; i <= 10;i++) {
            tasks.add(new MyTask("Task: " + String.valueOf(i),1000 ));
        }
        List<Future<MyResults>> futureResults = null;

        try {
            futureResults = exec.invokeAll(tasks);
        }
        catch(Exception e){};

        MyResults mr;
        for (Future<MyResults> res: futureResults) {
            try {
                mr = res.get();
                System.out.println(mr.getResult());
            }
            catch(Exception e){};
        }

        exec.shutdown();
    }
}
```

```
run:
Task: 1: 1545164383309
Task: 4: 1545164383309
Task: 5: 1545164383309
Task: 3: 1545164383309
Task: 10: 1545164384309
Task: 8: 1545164384309
Task: 7: 1545164384309
Task: 2: 1545164384309
Task: 9: 1545164385309
Task: 6: 1545164385309
BUILD SUCCESSFUL (total time: 3 seconds)
```

20

20