

# Advanced Multithreading Framework. Executor Service. Callable. Future. ForkJoinPool

Петър Казаков / SAP Labs Bulgaria  
Ноември 2014

Public



# Executor Service

---

- Дефиниция: Интерфейс за асинхронно изпълнение на група нишки в background.
- Имплементация:
  - Наследява Executor
  - Интерфейса е имплементиран в ForkJoinPool, ThreadPoolExecutor (+подкласове) и ScheduledThreadPoolExecutor.
- `executor.shutdown()` спира приема на нови нишки и довършва изпълнението на старите.
- `executor.shutdownNow()` спира приема на нови нишки и опитва да спре изпълнението на вече започнатите нишки. Не предоставя гаранции дали това ще стане или нишките ще завършат изпълнението си.

# Инициализация:

---

```
ExecutorService executorService1 = Executors.newSingleThreadExecutor();  
ExecutorService executorService2 = Executors.newFixedThreadPool(10);  
ExecutorService executorService3 = Executors.newScheduledThreadPool(10);
```

# Пример

---

```
public class Executor implements Runnable{  
    private int idx;  
    public Executor(int idx) { this.idx = idx;}  
    @Override  
    public void run() { System.out.println("Thread "+idx+ " executed");}  
    public static void main(String[] args) {  
        ExecutorService executorService2 = Executors.newFixedThreadPool(10);  
        for (int i=0;i<110;i++){  
            executorService2.execute(new Executor(i));  
        }  
        executorService2.shutdown();  
    }  
}
```

# Public interface Callable<V>

---

- Интерфейс който връща резултат V
- Може да хвърли Exception
- V call() throws Exception

# Public interface Future<V>

---

- Интерфейс който връща резултат V
- Може да се проверява дали изпълнението е завършило (isDone())
- Може да се прекъсва (cancel(), isCancelled())
- Може да се изчаква докато завърши (get())

# ExecutorService again

---

Result	method	param	
void	execute	Runnable	
Future	submit	Runnable	
Future	submit	Callable	

## Още един пример

---

```
public class Executor implements Callable<Integer>{  
    private int idx;  
    public Executor(int idx) {this.idx = idx;}  
    public Integer call() throws Exception{  
        if (idx==100){  
            throw new Exception("I am 100");  
        }else{  
            System.out.println("Thread "+idx+ " executed");  
        }  
        return idx;  
    }  
}
```

## Още един пример

---

```
public static void main(String[] args) {  
    ExecutorService executorService2 =  
        Executors.newFixedThreadPool(10);  
  
    for (int i=0;i<110;i++){  
        executorService2.submit(new Executor(i));  
    }  
  
    executorService2.shutdown();  
}
```

# Още един пример

---

```
public static void main(String[] args) {
    ExecutorService executorService2 =
        Executors.newFixedThreadPool(10);
    Set<Future<Integer>> res = new HashSet<>();
    for (int i=0;i<110;i++){
        res.add(executorService2.submit(new Executor(i)));
    }
    for (Future<Integer> item:res){
        try {
            System.out.println(item.get());
        } catch (InterruptedException | ExecutionException
e) {
            e.printStackTrace();
        }
    }
    executorService2.shutdown();
}
```

## Други методи на ExecutorService – invokeAll и invokeAny

---

```
Set<Callable<String>> callables = new HashSet<Callable<String>>();
callables.add(new Callable<String>() {
    public String call() throws Exception {
        return "Task 1";
    }
});
//...

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

# ScheduledExecutorService

---

- `schedule` (Callable task, long delay, TimeUnit timeunit)
- `schedule` (Runnable task, long delay, TimeUnit timeunit)
- `scheduleAtFixedRate` (Runnable, long initialDelay, long period, TimeUnit timeunit)
- `scheduleWithFixedDelay` (Runnable, long initialDelay, long period, TimeUnit timeunit)

# ForkJoinPool

---

- Инспириран от web crawler, базиран на work-stealing алгоритъм
- Подходящ за неравномерно разпределен workload

# ForkJoinPool

---

- Инспириран от web crawler, базиран на work-stealing алгоритъм
- Подходящ за неравномерно разпределен workload

```
numberOfProcessors = Runtime.getRuntime().availableProcessors();  
ForkJoinPool pool = new ForkJoinPool(numberOfProcessors);
```

# ForkJoinPool

---

- Инспириран от web crawler, базиран на work-stealing алгоритъм
- Подходящ за неравномерно разпределен workload

```
numberOfProcessors = Runtime.getRuntime().availableProcessors();  
ForkJoinPool pool = new ForkJoinPool(numberOfProcessors);
```

- Ползва ForkJoinTask<V> и RecursiveTask<V> като базов клас
- Олекотена версия на Future в замяна на някои ограничения:
  - Изчислява определени изолирани функции или работи над изолирани обекти
  - Работи на базата на fork-join в идеалния случай без блокиране. При последното трябва да се следват определени правила
  - Не разрешава checked exceptions като IOException
  - Използва join само при ацикличен граф, предлага и методи като complete

# ForkJoinPool

```
class Task extends RecursiveTask<Integer> {  
  
    private int n;  
  
    public Task(int n){this.n = n;}  
  
    @Override  
  
    protected Integer compute() {  
        if (n<3){ return n; }  
  
        else{  
            ForkJoinTask<Integer> t = new Task(n-1).fork();  
  
            Task t2 = new Task(n-3);  
  
            return t2.compute()+t.join();  
  
        }  
    }  
}
```

# ForkJoinPool - usage

---

```
public class ForkJoinPoolDemo {  
    public static void main(String[] args) {  
        ForkJoinPool pool = new ForkJoinPool();  
        Task task = new Task(10);  
        Integer f = pool.invoke(task);  
        System.out.println(f);  
    }  
}
```



# Благодаря за вниманието!

[peter.kazakov@sap.com](mailto:peter.kazakov@sap.com)

# ИЗПОЛЗВАНА ЛИТЕРАТУРА

---