**Lập trình đa luồng với Callable và Future trong Java**

*Đăng vào 02/03/2018 . Được đăng bởi*[**GP Coder**](https://gpcoder.com/author/gpadmin/)*.****18593****Lượt xem . Toàn màn hình*

Trong bài viết [Lập trình đa luồng trong Java](https://gpcoder.com/3484-lap-trinh-da-luong-trong-java-java-multi-threading) các bạn đã biết được 2 cách để tạo một Thread trong Java: tạo 1 đối tượng của lớp được **extend** từ **class Thread** hoặc **implements** từ **interface Runnable**. Trong bài viết này tôi giới thiệu với các bạn một cách khác để tạo Thread, đó là [**Callable**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Callable.html)trong Java với khả năng trả về kết quả [**Future**](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html)**<T>** sau khi xử lý và có thể throw Exception nếu trong quá trình xử lý tác vụ có lỗi xảy ra.

**Nội dung**[[Ẩn](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/)]

* [1 Callable](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Callable)
* [2 Future](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Future)
* [3 Ví dụ sử dụng Callable và Future](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Vi_du_su_dung_Callable_va_Future)
  + [3.1 Sử dụng phương thức submit(Callable) của ExecutorService với kiểu trả về là 1 Future<T>](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Su_dung_phuong_thuc_submitCallable_cua_ExecutorService_voi_kieu_tra_ve_la_1_FutureltTgt)
  + [3.2 Sử dụng phương thức get() của Future<T> với Timeout](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Su_dung_phuong_thuc_get_cua_FutureltTgt_voi_Timeout)
  + [3.3 Sử dụng phương thức invokeAny(Collection<?> extends Callable<T> tasks)](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Su_dung_phuong_thucinvokeAnyCollectionltgt_extends_CallableltTgt_tasks)
  + [3.4 Sử dụng phương thức invokeAll(Collection<?> extends Callable<T> tasks)](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Su_dung_phuong_thucinvokeAllCollectionltgt_extends_CallableltTgt_tasks)
  + [3.5 Hủy bỏ một Future](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/#Huy_bo_mot_Future)

**Callable**

Callable là một interface sử dụng Java Generic để định nghĩa đối tượng sẽ trả về sau khi xử lý xong tác vụ.

Callable interface cũng giống như Runnable, ngoại trừ khác ở chỗ thay vì trả về **void** từ **run()** method của **Runnable** thì **call()** method của **Callable** trả về đối tượng kiểu **Future<T>** (bất kỳ) hoặc throw **Exception**.

**Runnable**:

public abstract void **run**() => kiểu trả về là **void**

**Callable**:

<T> Future<T> **submit**(Callable<T> task) => kiểu trả về là **Future<T>**

<T> Future<T> **submit**(Runnable<T> task) => kiểu trả về là **Future<T>**

**Future**

Dùng để lấy kết quả khi thực thi một Thread, có thể lấy **Future** khi submit một task vào [ThreadPool](https://gpcoder.com/3548-huong-dan-tao-va-su-dung-threadpool-trong-java/). Task ở đây là một object implement **Runnable** hay **Callable**.

Một số method của Future:

* isDone() : Kiểm tra task hoàn thành hay không?
* cancel() : Hủy task
* isCancelled(): Kiểm tra task bị hủy trước khi hoàn thành hay không?
* get() : Lấy kết quả trả về của task.

**Ví dụ sử dụng Callable và Future**

**Sử dụng phương thức submit(Callable) của ExecutorService với kiểu trả về là 1 Future<T>**

Ví dụ về tính tổng bình phương của 10 số, thay vì xử lý việc tính tổng này trong Thread chính của chương trình, tôi sẽ tạo mới Thread sử dụng Callable để xử lý và nhận kết quả về thông qua Future.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **package** com.gpcoder.threadpool.callable;    **import** java.util.Random;  **import** java.util.concurrent.Callable;    **public** **class** CallableWorker **implements** Callable<Integer> {    **private** **int** num;  **private** Random ran;    **public** CallableWorker(**int** num) {  **this**.num = num;  **this**.ran = **new** Random();      }    **public** Integer call() **throws** Exception {          Thread.sleep(ran.nextInt(10) \* 1000);  **int** result = num \* num;          System.out.println("Complete " + num);  **return** result;      }  } |

Để thực thi tác vụ của Callable, chúng ta phải submit nó vào một [ThreadPool](https://gpcoder.com/3548-huong-dan-tao-va-su-dung-threadpool-trong-java) sử dụng phương thức submit() của [Executor Framework](https://gpcoder.com/3548-huong-dan-tao-va-su-dung-threadpool-trong-java/#Vi_du_su_dung_newSingleThreadExecutor). Để nhận kết quả trả về chúng ta sử dụng phương thức get() của lớp Future. Ta có chương trình như bên dưới:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | **package** com.gpcoder.threadpool.callable;    **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.concurrent.Callable;  **import** java.util.concurrent.ExecutionException;  **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;  **import** java.util.concurrent.Future;    **public** **class** CallableExample {  **public** **static** **void** main(String[] args) {          // create a list to hold the Future object associated with Callable          List<Future<Integer>> list = **new** ArrayList<>();            // Get ExecutorService from Executors utility class, thread pool size is 5          ExecutorService executor = Executors.newFixedThreadPool(5);            Callable<Integer> callable;          Future<Integer> future;  **for** (**int** i = 1; i <= 10; i++) {              callable = **new** CallableWorker(i);                // submit Callable tasks to be executed by thread pool              future = executor.submit(callable);                // add Future to the list, we can get return value using Future              list.add(future);          }            // shut down the executor service now          executor.shutdown();            // Wait until all threads are finish  **while** (!executor.isTerminated()) {              // Running ...          }    **int** sum = 0;  **for** (Future<Integer> f : list) {  **try** {                  sum += f.get();              } **catch** (InterruptedException | ExecutionException e) {                  e.printStackTrace();              }          }            System.out.println("Finished all threads: ");          System.out.println("Sum all = " + sum);      }  } |

Thực thi chương trình trên, ta có kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | Complete 5  Complete 4  Complete 7  Complete 6  Complete 3  Complete 2  Complete 9  Complete 1  Complete 8  Complete 10  Finished all threads:  Sum all = 385 |

**Lưu ý**: Tương tự như submit(Callable), phương thức **submit(Runnable)** cũng đưa vào 1 Runnable và nó trả về một đối tượng **Future**. Đối tượng Future có thể được sử dụng để kiểm tra nếu Runnable đã hoàn tất việc thực thi.

**Sử dụng phương thức get() của Future<T> với Timeout**

Phương thức get() là **synchronous**, do đó nó sẽ blocking chương trình của chúng ta mỗi khi đợi kết quả của Callable. Để hạn chế blocking chương trình quá lâu, chúng ta có thể sử dụng phương thức get() này với một thời gian Timeout như sau:

|  |  |
| --- | --- |
| 1 | future.get(7, TimeUnit.SECONDS); |

**Lưu ý**: khi sử dụng phương thức get() với Timeout có thể nhận một TimeoutException nếu thời gian thực thi của task vượt quá khoảng thời gian Timeout.

Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | **package** com.gpcoder.threadpool.callable;    **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.concurrent.Callable;  **import** java.util.concurrent.CancellationException;  **import** java.util.concurrent.ExecutionException;  **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;  **import** java.util.concurrent.Future;  **import** java.util.concurrent.TimeUnit;  **import** java.util.concurrent.TimeoutException;    **public** **class** CallableExample2 {  **public** **static** **final** **int** GET\_TIME\_OUT = 5;  **public** **static** **final** **int** NUM\_OF\_TASK = 10;    **public** **static** **void** main(String[] args) **throws** TimeoutException, InterruptedException {          // create a list to hold the Future object associated with Callable          List<Future<Integer>> list = **new** ArrayList<>();            // Get ExecutorService from Executors utility class, thread pool size is 5          ExecutorService executor = Executors.newFixedThreadPool(5);            Callable<Integer> callable;          Future<Integer> future;  **for** (**int** i = 1; i <= NUM\_OF\_TASK; i++) {              callable = **new** CallableWorker(i);                // submit Callable tasks to be executed by thread pool              future = executor.submit(callable);                // add Future to the list, we can get return value using Future              list.add(future);          }    **int** sum = 0;  **for** (Future<Integer> f : list) {  **try** {                  // print the return value of Future, notice the output delay in console                  // because Future.get() waits for task to get completed  **int** result = f.get(GET\_TIME\_OUT, TimeUnit.SECONDS);                  // Throw TimeoutException if the task execute over 7s                  sum += result;                  System.out.println("Result: " + result);                  System.out.println("Is completed? : " + f.isDone());                  System.out.println("Is canceled? : " + f.isCancelled());              } **catch** (TimeoutException | CancellationException | InterruptedException | ExecutionException e) {                  e.printStackTrace();              }              System.out.println("---");          }            // shut down the executor service now          executor.shutdownNow();            // Blocks until all tasks have completed execution after a shutdown request, or          // the timeout occurs, or the current thread is interrupted, whichever happens          // first.  **while** (!executor.awaitTermination(GET\_TIME\_OUT \* NUM\_OF\_TASK \* 1000, TimeUnit.SECONDS)) {              // Running ...          }            System.out.println("Finished all threads: ");          System.out.println("Sum all = " + sum);      }  } |

Thực thi chương trình trên vài lần bạn sẽ nhận được exception và kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | Complete 1  Result: 1  Is completed? : **true**  Is canceled? : **false**  ---  Complete 6  java.util.concurrent.TimeoutException  ---      at java.util.concurrent.FutureTask.get(FutureTask.java:205)      at com.gpcoder.threadpool.callable.CallableExample2.main(CallableExample2.java:42)  Complete 2  Complete 3  Result: 9  Is completed? : **true**  Is canceled? : **false**  ---  Complete 5  Complete 10  Complete 8  Complete 4  Result: 16  Is completed? : **true**  Is canceled? : **false**  ---  Result: 25  Is completed? : **true**  Is canceled? : **false**  ---  Result: 36  Is completed? : **true**  Is canceled? : **false**  ---  Complete 7  Result: 49  Is completed? : **true**  Is canceled? : **false**  ---  Result: 64  Is completed? : **true**  Is canceled? : **false**  ---  java.util.concurrent.TimeoutException  ---  Result: 100  Is completed? : **true**  Is canceled? : **false**  ---      at java.util.concurrent.FutureTask.get(FutureTask.java:205)      at com.gpcoder.threadpool.callable.CallableExample2.main(CallableExample2.java:42)  Finished all threads:  Sum all = 300 |

**Sử dụng phương thức invokeAny(Collection<?> extends Callable<T> tasks)**

Phương thức **invokeAny()** trả về một **Future**, nhưng trả về kết quả của **một trong những đối tượng Callable**. Nó không đảm bảo về kết quả bạn sẽ nhận được từ callable nào, chỉ cần một trong số chúng hoàn thành. Tức là ko cần tất cả các Thread hòan thành, chỉ cần 1 task hoàn thành phương thức get() sẽ nhận được kết quả.

Nếu 1 trong số task hoàn thành hoặc ném ra 1 ngoại lệ, phần còn lại của Callable sẽ được hủy bỏ (cancelled).

Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **package** com.gpcoder.threadpool.callable;    **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.concurrent.Callable;  **import** java.util.concurrent.ExecutionException;  **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;    **public** **class** InvokeAnyExample {    **public** **static** **void** main(String[] args) **throws** InterruptedException, ExecutionException {          // Get ExecutorService from Executors utility class, thread pool size is 5          ExecutorService executor = Executors.newFixedThreadPool(5);            List<Callable<Integer>> callables = **new** ArrayList<>();  **for** (**int** i = 1; i <= 10; i++) {              callables.add(**new** CallableWorker(i));          }            Integer result = executor.invokeAny(callables);          System.out.println("Result = " + result);            executor.shutdown();          System.out.println("Finished all threads ");      }    } |

Thực thi chương trình trên sẽ in ra các kết quả được trả về từ 1 trong 10 Callable từ danh sách callables. Mỗi lần chạy sẽ nhận được một kết quả khác nhau.

Kết quả lần 1:

|  |  |
| --- | --- |
| 1  2  3 | Complete 4  Result = 16  Finished all threads |

Kết quả lần 2:

|  |  |
| --- | --- |
| 1  2  3  4 | Complete 3  Complete 6  Result = 9  Finished all threads |

**Sử dụng phương thức invokeAll(Collection<?> extends Callable<T> tasks)**

Phương thức **invokeAll()** gọi tất cả đối tượng Callable trong tập hợp. Phương thức này trả về 1 danh sách các đối tượng **Future<T>** mà được trả về từ việc thực thi các Callables.

Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | **package** com.gpcoder.threadpool.callable;    **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.concurrent.Callable;  **import** java.util.concurrent.ExecutionException;  **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;  **import** java.util.concurrent.Future;    **public** **class** InvokeAllExample {    **public** **static** **void** main(String[] args) **throws** InterruptedException, ExecutionException {          // Get ExecutorService from Executors utility class, thread pool size is 5          ExecutorService executor = Executors.newFixedThreadPool(5);            List<Callable<Integer>> callables = **new** ArrayList<>();  **for** (**int** i = 1; i <= 10; i++) {              callables.add(**new** CallableWorker(i));          }            List<Future<Integer>> futures = executor.invokeAll(callables);    **int** sum = 0;  **for** (Future<Integer> future : futures) {              sum += future.get();          }          System.out.println("Sum all = " + sum);            executor.shutdown();          System.out.println("Finished all threads ");      }  } |

Thực thi chương trình trên, bạn sẽ thấy tất cả các Callable đều được thực thi và kết quả được lưu vào List<Future>. Ta có kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | Complete 1  Complete 5  Complete 7  Complete 4  Complete 8  Complete 2  Complete 3  Complete 9  Complete 6  Complete 10  Sum all = 385  Finished all threads |

**Hủy bỏ một Future**

Bạn có thể hủy một Future bằng cách sử dụng phương thức **Future.cancel()**. Khi phương thức này được gọi, nó cố gắng để hủy bỏ việc thực hiện các task và trả về **true** nếu nó bị hủy bỏ thành công, nếu không, nó trả về **false**.

Phương thức **cancel()** chấp nhận đối số boolean – **mayInterruptIfRunning**. Nếu gán giá trị true cho đối số này, Thread hiện đang thi hành task sẽ bị gián đoạn (interrupted), nếu không các task đang xử lý sẽ được phép hoàn thành.

Bạn có thể sử dụng phương thức **isCancelled()** để kiểm tra xem một task có bị hủy hay không. Ngoài ra, sau khi huỷ bỏ task thì phương thức **isDone()** sẽ luôn luôn có kết quả là **true**.

Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | **package** com.gpcoder.threadpool.callable;    **import** java.util.concurrent.ExecutionException;  **import** java.util.concurrent.ExecutorService;  **import** java.util.concurrent.Executors;  **import** java.util.concurrent.Future;  **import** java.util.concurrent.TimeUnit;    **public** **class** FutureCancelExample {    **public** **static** **void** main(String[] args) **throws** InterruptedException, ExecutionException {          ExecutorService executorService = Executors.newSingleThreadExecutor();    **long** startTime = System.currentTimeMillis();          Future<Integer> future = executorService.submit(**new** CallableWorker(1));    **while** (!future.isDone()) {              System.out.println("Task is still working ...");              Thread.sleep(200);    **long** workingTime = (System.currentTimeMillis() - startTime);  **if** (TimeUnit.SECONDS.toSeconds(workingTime) > 1000) {                  future.cancel(**true**);              }          }            executorService.shutdown();    **if** (!future.isCancelled()) {              System.out.println("Task completed! Retrieving the result");              System.out.println("Result = " + future.get());          } **else** {              System.out.println("Task was cancelled");          }            System.out.println("It will throw exception form here");          System.out.println("Result = " + future.get());      }    } |

Thực thi chương trình trên, ta có kết quả như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Task is still working ...  Task is still working ...  Task is still working ...  Task is still working ...  Task is still working ...  Task was cancelled  It will **throw** exception form here  Exception in thread "main" java.util.concurrent.CancellationException      at java.util.concurrent.FutureTask.report(FutureTask.java:121)      at java.util.concurrent.FutureTask.get(FutureTask.java:192)      at com.gpcoder.threadpool.callable.FutureCancelExample.main(FutureCancelExample.java:37) |

Như bạn thấy chương trình trên nó sẽ ném một ngoại lệ (exception), bởi vì phương thức **future.get()** sẽ ném **CancellationException** nếu task được hủy bỏ. Chúng ta có thể xử lý sự kiện này bằng cách kiểm tra Future có bị hủy bỏ hay không trước khi lấy kết quả thông qua phương thức **future.isCancelled()**.

Trên đây là toàn bộ những kiến thức cơ bản về việc sử dụng Callable và Future với ExecutorService Framework. Hy vọng bài viết này giúp ích cho các bạn, hẹn gặp lại ở các bài viết tiếp theo.

**4.8**

24

**Nếu bạn thấy hay thì hãy chia sẻ bài viết cho mọi người nhé!**

SHARES

*Chuyên mục:*[**Multi-Thread**](https://gpcoder.com/category/java-core/multi-thread/)*Được gắn thẻ:*[**Callable**](https://gpcoder.com/tag/callable/)*,*[**Future**](https://gpcoder.com/tag/future/)*,*[**Multithreading**](https://gpcoder.com/tag/multithreading/)

[Hướng dẫn tạo và sử dụng ThreadPool trong Java](https://gpcoder.com/3548-huong-dan-tao-va-su-dung-threadpool-trong-java/)

[Sử dụng Fork/Join Framework với ForkJoinPool trong Java](https://gpcoder.com/3573-su-dung-fork-join-framework-voi-forkjoinpool-trong-java/)

**Có thể bạn muốn xem:**

* [Vấn đề Nhà sản xuất (Producer) – Người tiêu dùng (Consumer) và đồng bộ hóa các luồng trong Java](https://gpcoder.com/6180-van-de-nguoi-san-xuat-producer-nguoi-tieu-dung-consumer-va-dong-bo-hoa-cac-luong-trong-java/) **(23/09/2019)**
* [Hướng dẫn tạo và sử dụng ThreadPool trong Java](https://gpcoder.com/3548-huong-dan-tao-va-su-dung-threadpool-trong-java/) **(27/02/2018)**
* [Semaphore trong Java](https://gpcoder.com/6187-semaphore-trong-java/) **(18/09/2019)**
* [Lập trình đa luồng với CompletableFuture trong Java 8](https://gpcoder.com/4064-lap-trinh-da-luong-voi-completablefuture-trong-java-8/) **(19/07/2018)**
* [Đồng bộ hóa các luồng trong Java](https://gpcoder.com/3514-dong-bo-hoa-cac-luong-trong-java/) **(20/02/2018)**

**Bình luận**

3 bình luận

[**Oum Chanmakara**](https://www.facebook.com/profile.php?id=100004742805170)

Cho 2 số nguyên dương N và M (0 < N < M). Tìm số lượng các số nguyên tố trong phạm vi từ N đến M. Ví dụ: N=5000000, M=10000000.

[Thích](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [Phản hồi](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [11 tuần](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/?fb_comment_id=675280527426701)

[](https://www.facebook.com/profile.php?id=100023171116103)

[**Vũ Đạt**](https://www.facebook.com/profile.php?id=100023171116103)

Hai ví dụ : CallableExample và CallableExample2 câu lệnh shutdownNow  
+đặt giữa câu lệnh khi dùng for Future để hiển thị kết quả  
+đặt cuối câu lệnh khi dùng for Future để hiển thị kết quả  
  
=>Nó khác nhau điểm gì vậy anh

[Thích](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [Phản hồi](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · 1 · [1 năm](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/?fb_comment_id=3790465684410059)

[](https://www.facebook.com/profile.php?id=100023171116103)

[**Vũ Đạt**](https://www.facebook.com/profile.php?id=100023171116103)

while (!executor.awaitTermination thay vì dùng cái này giữ chỗ đợi câu lệnh dừng.Em nghĩ nên dùng countLatchDownd sẽ hay hơn

[Thích](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [Phản hồi](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [3 năm](https://gpcoder.com/3565-lap-trinh-da-luong-voi-callable-va-future-trong-java/?fb_comment_id=3897115453745081)

[](https://facebook.com/GPCoder)

[**GP Coder - Lập trình Java**](https://facebook.com/GPCoder)

Không rõ em đã apply như thế nào. Tuy nhiên, sử dụng CountDownLatch có vấn đề như sau:  
  
CountDownLatch nó có khác biệt nên sử dụng sẽ khó khăn hơn. Với CountDownLatch nếu e khởi tạo giá trị ban đầu là 3 thì bắt buộc phải count down về 0, main thread mới thực thi. Nếu chương trình chỉ có 2 thì không làm được. Ngược lại, nếu 4 thread thì thread thứ 4 cũng không làm ảnh hưởng đến CountDownLatch. CountDownLatch thực sự hữu ích khi biết chính xác số lượng Thread.  
  
Đối với Executor, em có thể khởi tạo giá trị ban đầu là 4, queue là 20. Nếu số lượng thread là 5 hay 10 thì nó có thể thêm vào queue để thực thi sau.

[Thích](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [Phản hồi](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · 1 · 3 năm

[](https://www.facebook.com/profile.php?id=100023171116103)

[**Vũ Đạt**](https://www.facebook.com/profile.php?id=100023171116103)

Em dùng cown để đếm giá trị truyền vào đã được luồng xử lý hết chưa chứ ko đếm thread được tạo ra.Mã ví dụ cơ bản [https://imgur.com/a/ZLMSut2](https://l.facebook.com/l.php?u=https%3A%2F%2Fimgur.com%2Fa%2FZLMSut2%3Ffbclid%3DIwAR3bQ1Q_-QmgaNRjkdtNQRkUTDNbIRvUk0yRri-S34tKHGVsA8Eu-lmYYQY&h=AT0vxA6mlNHvRlhG7RaGN1dNe_zzf0GuVdGXs57Ihi4ui1ouAXETK13XmmTIedU8QtzSl0xpHfbsLXUIyjCYRhLCAjmcTUWZ2lBST00TEAIdswTIJrpEhWSrhUKBVG1Di2nH9A)  
  
Em thực thi thấy rất chuẩn,anh cứ xem qua xem em còn thiếu sót nào ko anh nhé

[Thích](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [Phản hồi](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · 3 năm

[](https://www.facebook.com/profile.php?id=100023171116103)

[**Vũ Đạt**](https://www.facebook.com/profile.php?id=100023171116103)

Sau khi dùng.em thấy cách của em ko ổn nếu giá trị lớn hơn int ko có nên một vài chương trình là ko thích hợp

[Thích](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · [Phản hồi](https://www.facebook.com/plugins/feedback.php?app_id=510733472615500&channel=https%3A%2F%2Fstaticxx.facebook.com%2Fx%2Fconnect%2Fxd_arbiter%2F%3Fversion%3D46%23cb%3Df3c513eaa5e313c%26domain%3Dgpcoder.com%26is_canvas%3Dfalse%26origin%3Dhttps%253A%252F%252Fgpcoder.com%252Ffc7788c573106%26relation%3Dparent.parent&color_scheme=light&container_width=691&height=100&href=https%3A%2F%2Fgpcoder.com%2F3565-lap-trinh-da-luong-voi-callable-va-future-trong-java%2F&locale=vi_VN&numposts=5&sdk=joey&width) · 3 năm

[](https://www.facebook.com/profile.php?id=100023171116103)

[**Vũ Đạt**](https://www.facebook.com/profile.php?id=100023171116103)

Sau một thời gian dùng thì cách em đề nghị đã bị xếp xó,vì có những cách tốt hơn