Understanding java.util.concurrent.CompletionService

Posted on December 30, 2011 (https://markusjais.com/understanding-java-util-concurrent-completionservice/) by Markus Jais (https://markusjais.com/author/heliaca/)

(https://markusjais.com/understanding-java-util-concurrent-completionservice/)

The `java.util.concurrent.CompletionService` is a useful interface in the JDK standard libraries but few developers know it.
One could live without it as you can of course program this functionality with the other interfaces and classes within `java.util.concurrent` but it is convenient to have a solution that is already available and less error prone then doing it yourself. I always prefer stuff that is already available within the JDK over implementing my own solution with the same features – unless as an exercise at home!

Image you have a list of separate tasks that take a while, e.g. 10 tasks that each download an URL and return the content as a String.
Depending on the network, the size of the downloaded content and other factors, the time to download each URL will take various amounts of time.
When you execute them in parallel you may want to start doing something with the downloaded content as soon as the first task is done. No need to wait for the other 9 tasks to complete because that would mean you would always have to wait until all URLs are downloaded before doing something useful with each individual result.

You can of course execute all of the and get a `List of Future` objects and then poll on each one but it is easier to just use a `CompletionService`.

See the following example:

First, some dummy `Callable`:

```java
import java.util.Random;
import java.util.concurrent.Callable;

public class LongRunningTask implements Callable&lt;String&gt; {

    public String call() {
        // do stuff and return some String
        try {
            Thread.sleep(Math.abs(new Random().nextLong() % 5000));
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        return Thread.currentThread().getName();
    }
}
```

The `LongRunningTask` is just a place holder for a real Task you might want to implement. In this dummy example, it just sleeps for a random amount of time and returns a String that contains the name of the current thread.

Second, an example using a `CompletionService` that uses the `Callable` above.

```java
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.CompletionService;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorCompletionService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class CompletionServiceExample {

    // dummy helper to create a List of Callables return a String
    public static List<Callable<String>> createCallableList() {
        List<Callable<String>> callables = new ArrayList<>();
        for (int i = 0; i &lt; 10; i++) {
            callables.add(new LongRunningTask());
        }
        return callables;
    }

    public static void main(String[] args) {

        ExecutorService executorService = Executors.newFixedThreadPool(10);

        CompletionService<String> taskCompletionService = new ExecutorCompletionService<St
                executorService);

        try {
            List<Callable<String>> callables = createCallableList();
            for (Callable<String> callable : callables) {
                taskCompletionService.submit(callable);
            }
            for (int i = 0; i < callables.size(); i++) {
                Future<String> result = taskCompletionService.take();
                System.out.println(result.get());
            }
        } catch (InterruptedException e) {
            // no real error handling. Don't do this in production!
            e.printStackTrace();
        } catch (ExecutionException e) {
            // no real error handling. Don't do this in production!
            e.printStackTrace();
        }
        executorService.shutdown();
    }
}
```

Note: *The examples don't have proper exception handling to keep it simple. Don't copy this into your production code!*

The `CompletionServiceExample` shows how to use a `CompletionService`. You create an instance of `ExecutorCompletionService` (the only implementation of the `CompletionService` interface available with Java 7 or older versions) and then you `submit` all Callables to the `CompletionService`.

As soon as a task is completed, it is put in an internal `java.util.concurrent.BlockingQueue` (a highly efficient queue for Producer/Consumer problems and communication between threads).

From that queue, you can get the results of the finished tasks with `take` . If no task is yet available, `take` will wait until something is available.
In this case we just print the result (the name of the current threat executing the `Callable`).

This is all you need to know to use a `CompletionService` . It is really simple. There is a lot of cool stuff in the JDK and in the `java.util.concurrent` package. Make sure to browse through the docs from time to time before inventing your own solution.

‹ SHORT BOOK REVIEW: THE LINUX PROGRAMMING INTERFACE (HTTPS://MARKUSJAIS.COM/SHORT-BOOK-REVIEW-THE-LINUX-PROGRAMMING-INTERNFACE/)

GETTING STARTED WITH ECLIPSE CDT, THREADING BUILDING BLOCKS, PARALLEL_FOR AND C++11 LAMBDAS › (HTTPS://MARKUSJAIS.COM/GETTING-STARTED-WITH-ECLIPSE-CDT-THREADING-BUILDING-BLOCKS-AND-CPLUSPLUS-11/)

CATEGORIES

Books (https://markusjais.com/category/books/)

C/C++ (https://markusjais.com/category/cpp/)

Concurrency (https://markusjais.com/category/concurrency/)

Functional Programming (https://markusjais.com/category/functional-programming/)

General (https://markusjais.com/category/general/)

Java (https://markusjais.com/category/java/)

Linux System Programming (https://markusjais.com/category/linux-system-programming/)

NoSQL (https://markusjais.com/category/nosql/)

Python (https://markusjais.com/category/python/)

reviews (https://markusjais.com/category/reviews/)

Ruby (https://markusjais.com/category/ruby/)

Rust (https://markusjais.com/category/rust/)

Scala (https://markusjais.com/category/scala/)

RECENT POSTS

Good names and types are the foundation of good software design (https://markusjais.com/good-names-and-types-are-the-foundation-of-good-software-design/)

Unterstanding Rust's Vec and its capacity for fast and efficient programs (https://markusjais.com/unterstanding-rusts-vec-and-its-capacity-for-fast-and-efficient-programs/)

Book review: Rust Essentials (https://markusjais.com/book-review-rust-essentials/)

Book review: MongoDB – The Definitive Guide, 2nd edition (https://markusjais.com/book-review-mongodb-the-definitive-guide/)

Video review: Functional Thinking by Neal Ford, O'Reilly Media (https://markusjais.com/video-review-functional-thinking-by-neal-ford-oreilly-media/)

RECENT COMMENTS

ARCHIVES

January 2019 (https://markusjais.com/2019/01/)

November 2016 (https://markusjais.com/2016/11/)

July 2015 (https://markusjais.com/2015/07/)

August 2013 (https://markusjais.com/2013/08/)

May 2013 (https://markusjais.com/2013/05/)

December 2012 (https://markusjais.com/2012/12/)

September 2012 (https://markusjais.com/2012/09/)

August 2012 (https://markusjais.com/2012/08/)

April 2012 (https://markusjais.com/2012/04/)

December 2011 (https://markusjais.com/2011/12/)

November 2011 (https://markusjais.com/2011/11/)

September 2011 (https://markusjais.com/2011/09/)

August 2011 (https://markusjais.com/2011/08/)