

# Executors, Futures, and Cancelation

---

*Source:* <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2007-12/msg02062.html>

---

- *From:* [mieivlecl@xxxxxxxxxx](mailto:mieivlecl@xxxxxxxxxx)
  - *Date:* Tue, 11 Dec 2007 19:53:48 -0800 (PST)
- 

Hello. I am working with the `java.util.concurrent` API introduced in Java 1.5. Specifically, I will talk about Executors, ExecutorServices and Futures.

So far, I have gotten a lot of use out of the API, and used it on a number of projects. However, I am consistently running into a problem for which I have not found a good solution.

Callables submitted to an `ExecutorService` return a reference to a `Future` object. This future object gives you some modest ways to interact with the task: a method to cancel the task (`cancel`), methods to query the status of the task (`isDone/isCancelled`), and methods to get the result of the task's processing (`get`).

The problem is as follows:

Say you wish to cancel some long-running task that you know is queued for execution. Let's say you do this with, say:

```
future.cancel(true); // true allows the executor to use an interrupt  
to help cancel the task
```

The trouble here, is, now that we have cancelled the task, we have no way of determining if the task was cancelled before running, or if it was cancelled while running.

Let us say that there's something we always need to do after this task. Let's call "something" a method: `myCleanup()`.

By "after this task," I mean we always need to this `myCleanup()` whether the task ran to completion or was cancelled. So, there should be a one-to-one relationships between enqueues of the task, and calls to `myCleanup()`.

For the sake of serial execution, we put the call to `myCleanup()` in the task code itself. So the core of the task code looks something like:

```
performWhateverThisTaskDoes();
```

## Executors, Futures, and Cancellation

```
cleanup();
```

However, it is possible that the call to ---

```
future.cancel(true);
```

--- will prevent this task from ever being executed. This is especially true if, for example, our executor has a sizable task backlog in its work queue. That's OK. We asked for the task to be cancelled, and we got it.

The trouble comes when you realize (or already know) that ---

```
future.cancel(true);
```

--- does not always stop the task from executing. If the task is "in the process of executing," cancelling it will actually normally do nothing --- the task will happily run to completion. This is OK and I have come to terms with this behavior.

The crux of the problem is there's no way to figure out what happened --- what state the task is in, when it was cancelled. I have no idea if I should expect that the task is "in progress" and will eventually