# Is there a way to reuse a Job instance?

Asked 3 years, 2 months ago Active 1 year, 2 months ago Viewed 2k times



9

4

I'm exploring the use of co-routines in the context of Android UI thread. I implemented `contextJob` as described in the Coroutines Guide UI. Background work is stared from GUI and I want to re-start it on every click (stop the currently running one and start it again).

But a job once canceled cannot be reused so even creating a child-job:

```
val job = Job(contextJob)
```

and cancelling it does not help because it has to be-reassigned.

Is there a way to reuse a Job instance?

android kotlin async-await coroutine kotlin-coroutines

edited Mar 19 '19 at 7:51
Marko Topolnik
167k ● 23 ● 242 ● 363

asked Mar 16 '17 at 9:09
atok
5,081 ● 1 ● 25 ● 52

## 1 Answer

Active Oldest Votes

7

A Job has a very simple life-cycle by design. Its "Completed" state is *final*, very much similar to the "Destroyed" state of the Android `Activity`. So, a parent `Job` is best to be associated with an `Activity`, as explained in the guide. You should cancel a parent job if and only if the activity is destroyed. Because a destroyed activity cannot be reused, you'll never run into the need to reuse its job.

The recommended approach to starting the work on each click is by using actors, because they help you avoid unneccesary concurrency. The guide shows how to start them on each click, but it does not show how to cancel a currently running action.

You will need a fresh instance of `Job` in a combination with `withContext` to make a block of code cancellable separately from everything else:

```
fun View.onClick(action: suspend () -> Unit) {
    var currentJob: Job? = null // to keep a reference to the currently running job
    // launch one actor as a parent of the context job
    // actor prevent concurrent execution of multiple actions
```

```
    val eventActor = actor<Unit>(contextJob + UI, capacity = Channel.CONFLATED) {
        for (event in channel) {
            currentJob = Job(contextJob) // create a new job for this action
            try {
                // run an action within its own job
                withContext(currentJob!!) { action() }
            } catch (e: CancellationException) {
                // we expect it to be cancelled and just need to continue
            }
        }
    }
    // install a listener to send message to this actor
    setOnClickListener {
        currentJob?.cancel() // cancel whatever job we were doing now (if any)
        eventActor.offer(Unit) // signal to start next action when possible
    }
}
```

An actor is always active until its parent job (attached to an activity) is cancelled. An actor waits for clicks and starts an `action` on each click. However, each invocation of an `action` is wrapped into its own `Job` using `withContext` block, so that it can be cancelled separately from its parent job.

Note, that this code gracefully works for actions that are non-cancellable or just take some time to cancel. An action might need to cleanup its resources when it is cancelled, and, because this code uses an actor, it ensures that the cleanup of the previous action is finished before the next one is started.

edited Dec 28 '17 at 19:08

Enleur

7 ● 1 ● 4

answered Mar 16 '17 at 10:25

Roman Elizarov

15.9k ● 8 ● 42 ● 51