




coroutineScope vs supervisorScope



Amit Shekhar
October 5, 2022



coroutineScope vs supervisorScope



I am [Amit Shekhar](#), I have taught and mentored many developers, and their efforts landed them high-paying tech jobs, helped many tech companies in solving their unique problems, and created many open-source libraries being used by top companies. I am passionate about sharing knowledge through open-source, blogs, and videos.

Join my program and get high paying tech job: amitshekhar.me

Before we start, I would like to mention that, I have released a video playlist to help you crack the Android Interview: Check out [Android Interview Questions and Answers](#).

In this blog, we will learn about the **coroutineScope** vs **supervisorScope** of Kotlin Coroutines.

There is a major difference between the **coroutineScope** and the **supervisorScope**, and we must understand that difference otherwise we will make mistakes without knowing what is happening with the Kotlin Coroutines code.

As always, I am taking a real example use-case to understand this major difference between the **coroutineScope** and the **supervisorScope**.

Example use case:

Suppose, we have two network calls as follows:

- **getUsers()**

- `getMoreUsers()`

We want to make these two network calls in parallel to get the data from the server.

You can find the complete source code of this example on this GitHub repository: [Learn Kotlin Coroutines](#)

In Kotlin Coroutines, for doing tasks in parallel.

We can write our code as below:

```
launch {
    try {
        val usersDeferred = async { getUsers() }
        val moreUsersDeferred = async { getMoreUsers() }
        val users = usersDeferred.await()
        val moreUsers = moreUsersDeferred.await()
    } catch (exception: Exception) {
        // handle exception
    }
}
```

Here, we will face one major problem, if any one of the two network calls leads to an error, the application will **crash!** it will **NOT** go to the catch block.

To solve this, we will have to use the `coroutineScope` as below:

```
launch {
    try {
        coroutineScope {
            val usersDeferred = async { getUsers() }
            val moreUsersDeferred = async { getMoreUsers() }
            val users = usersDeferred.await()
            val moreUsers = moreUsersDeferred.await()
        }
    } catch (exception: Exception) {
        // handle exception
    }
}
```

Now, if any network error comes, it will go to the catch block. This is how `coroutineScope` helps.

But suppose again, we want to return an empty list for the network call which has failed and continue with the response from the other network call. We will have to use the `supervisorScope` and add the `try-catch` block to the individual network call as below:

```
launch {
    supervisorScope {
        val usersDeferred = async { getUsers() }
        val moreUsersDeferred = async { getMoreUsers() }
        val users = try {
            usersDeferred.await()
        } catch (e: Exception) {
            emptyList<User>()
        }
    }
}
```

```
    val moreUsers = try {  
        moreUsersDeferred.await()  
    } catch (e: Exception) {  
        emptyList<User>()  
    }  
}  
}
```

So now, if any error comes, it will continue with the empty list. This is how **supervisorScope** helps.

The major difference:

- A **coroutineScope** will cancel whenever any of its children fail.
- A **supervisorScope** won't cancel other children when one of them fails.

Note:

- Use **coroutineScope** with the top-level **try-catch**, when you do **NOT** want to continue with other tasks if any of them have failed.
- If we want to continue with the other tasks even when one fails, we go with the **supervisorScope**.
- Use **supervisorScope** with the individual **try-catch** for each task, when you want to continue with other tasks if one or some of them have failed.

This was all about the **coroutineScope** and the **supervisorScope** that are present in the Kotlin Coroutines.

Master Kotlin Coroutines from here: [Mastering Kotlin Coroutines](#)

That's it for now.

Thanks

[Amit Shekhar](#)

You can connect with me on:

- [Twitter](#)
- [YouTube](#)
- [LinkedIn](#)
- [GitHub](#)

[Read all of my high-quality blogs here.](#)

TAGS

KOTLIN

ANDROID

Amit Shekhar

My vision is to make tech education outcome-focused. My mission is to provide tech education to the students through project-based learning to achieve the outcome they aspire to.