Android programming

TX00CK66 Sensor Based Mobile Applications

Lecture Coroutines and Network

Jarkko.Vuori@metropolia.fi

Coroutines and Network communication

- Using coroutines it is possible to make asynchronous code look like synchronous code

```
suspend fun fetchDocs() {                                   // Dispatchers.Main
    val result = get("https://developer.android.com") // Dispatchers.IO for `get`
    show(result)                                        // Dispatchers.Main
}

suspend fun get(url: String) = withContext(Dispatchers.IO) { /* ... */ }
```

- In this example, `get()` still runs on the main thread, but it suspends the coroutine running on the main thread before it starts the network request
  - Thus the main thread can service other main thread coroutines, and the UI is not stopped
  - When the network request completes, get resumes the suspended main thread coroutine instead of using a callback to notify the main thread
- Kotlin uses a stack frame to manage which function is running along with any local variables
  - When suspending a coroutine, the current stack frame is copied and saved for later
  - When resuming, the stack frame is copied back from where it was saved, and the function starts running again
- Even though the code might look like an ordinary sequential blocking request, the coroutine ensures that the network request avoids blocking the main thread

withContext

- It’s a suspend function that runs a block of code in the specified coroutine context, suspends the current coroutine until the execution has completed and returns the result
  - `withContex()` is the same than `async()` and `await()`
- Given that `withContext()` is a suspend function, when it’s called, both (1) and (2) execute completely before moving on to (3)
  - Also in this case, as we’ve seen with the `launch()` and `async()` coroutine builders, we can specify a coroutine context
  - This means, for example, that we can run the block of code inside `withContext()` on a separate thread and resume on the thread of our parent coroutine with the result

```
val resultValue: T = withContext(coroutineContext) {
    (1) ... code that returns a result of type T...
    (2) return@withContext result
}
(3) ... other code that uses resultValue ...
```

Example – launch() + withContext()

```
 override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     setContentView(R.layout.activity_main)
     val myUrl = URL("https://a.com/test/robot.txt")
     lifecycleScope.launch(Dispatchers.Main) {
         val serverResp = getText(myUrl)
         showText(serverResp)
    }
}

private suspend fun getText(url: URL): String =
    withContext(Dispatchers.IO) {
        val str=url.readText()  // easy, but not for very large files
        return@withContext str
    }

private fun showText (t: String){
    txt.text = t
}
```

Activity level scope - coroutine launched in this scope is canceled when Activity is destroyed

showText runs in main thread, getText will return to main thread, but executed on a thread from I/O thread pool

Example – async() + await()

```
override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      val txtUrl = URL("https://a.fi/test/robot.txt")
      val imgUrl = URL("https://a.fi/test/robo.jpg")
      lifecycleScope.launch(Dispatchers.Main) {
         val myTxt = async(Dispatchers.IO) { getTxt(txtUrl) }
         val myImg = async(Dispatchers.IO) { getImg(imgUrl) }
         showRes(myTxt.await(), myImg.await())
      }
  }

  private suspend fun getImg(url: URL): Bitmap {
         …
  }

  private suspend fun getTxt(url: URL): String{
         …
  }

  private fun showRes(serverTxt: String, serverImg: Bitmap){
         …
  }
```

showRes runs in main thread, getTxt and getImg will return to main thread, but run in thread from I/O thread pool

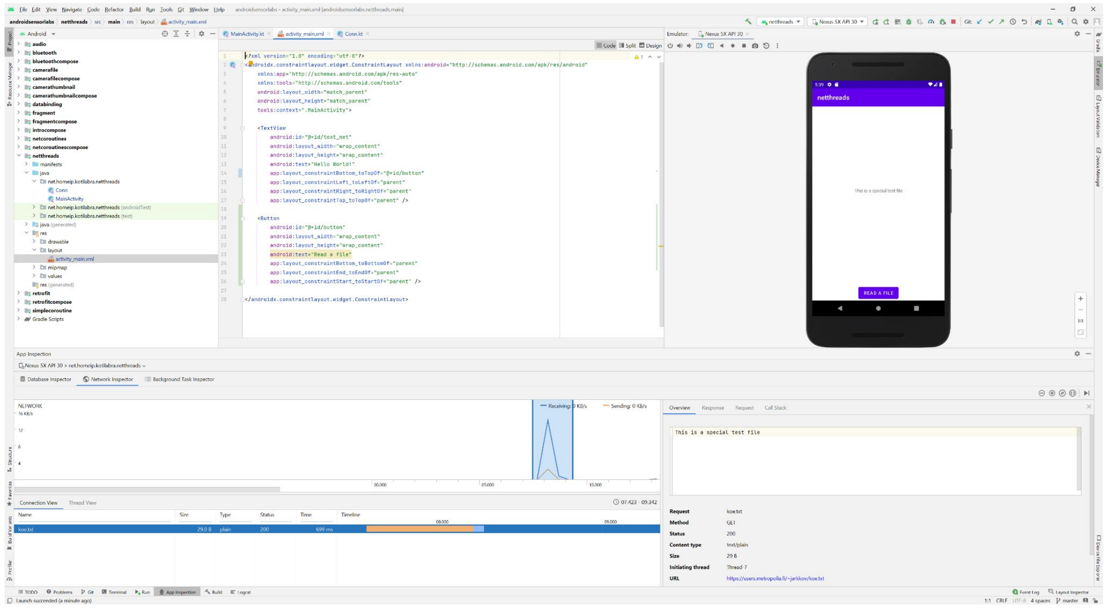
wait for the results from getTxt and getImg – await consumes the result of the async block

Exception Handling

- launch: The exception propagates to the parent and will fail your coroutine parent-child hierarchy. This will throw an exception in the coroutine thread immediately. You can avoid these exceptions with try/catch blocks, or a custom exception handler.
- async: You defer exceptions until you consume the result for the async block. That means if you forgot or did not consume the result of the async block, through await(), you may not get an exception at all! The coroutine will bury it, and your app will be fine. If you want to avoid exceptions from await(), use a try/catch block either on the await() call, or within async().

Network Debugging

- Android Studio has a very elaborate Network inspector which can be used to analyze the application's runtime behaviour
  - Inspector is activated by selecting the App Inspector menu item at the bottom of the screen and selecting the Network inspector
- Network operation can be analyzed (timing, server responses, amount of data, message content, etc.)



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/text_net"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Read a file"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Name	Size	Type	Status	Time
koe.txt	29.0 B	plain	200	699 ms

```
This is a special test file
```

Request	koe.txt
Method	GET
Status	200
Content type	text/plain
Size	29 B
Initiating thread	Thread-7
URL	https://users.metropolia.fi/~jarkkov/koe.txt

Reading list

- https://www.baeldung.com/kotlin/kotlin-threads-coroutines
- https://dmitrykandalov.com/coroutines-as-threads
- https://stackoverflow.com/questions/47871868/what-does-the-suspend-function-mean-in-a-kotlin-coroutine
- https://www.raywenderlich.com/6994782-android-networking-with-kotlin-tutorial-getting-started