Android
-
asynchrónnosť

Peter Borovanský
KAI, I-18

borovan 'at' ii.fmph.uniba.sk

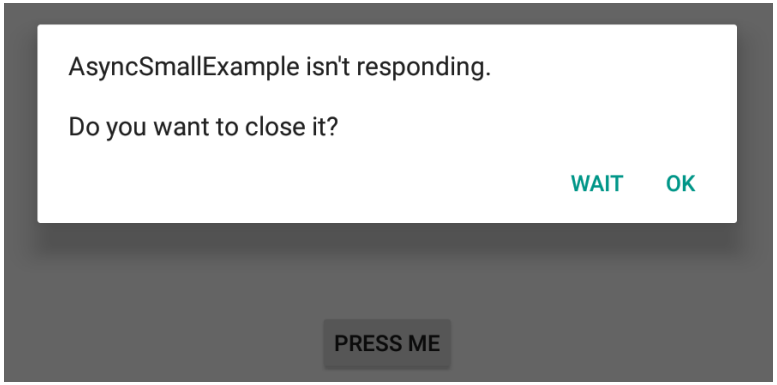
Async Task
Kotlin coroutines

Asynchrónne operácie

- nie je možné robiť časovo náročné operácie v hlavnom vlákne aplikácie
  - extra komplikovaný výpočet
  - simulácia procesu spomaľovaná Thread.sleep
  - požiadavka (napr. http/sql-request), ktorá môže trvať netriviálne dlho

- Takýto kód zablokuje hlavné vlákno, a ak vyvoláte GUI eventy (napr. pochabým klikaním v priebehu 20s), správca aplikácii usúdi, že aplikácia je mŕtva zavre ju

```
fun buttonClick(view: View) {
  var i = 0
  while (i <= 20) {
    try {
        Thread.sleep(1000)
        i++
    }
    catch (e: Exception) {
        e.printStackTrace()
  }
}
```



Async Task
(doInBackground)

```
private inner class MyTask : AsyncTask<String, Int, String>() {

   override fun onPreExecute() { … }  // vykoná sa pred doInBackground

   override fun doInBackground(vararg params: String): String {
      while (i in 0..20) {
        try {
           Thread.sleep(1000)
           publishProgress(i)
        } catch (e: Exception) { … }
        }
        return "Button Pressed"
    }

    override fun onProgressUpdate(vararg values: Int?) { … }

    override fun onPostExecute(result: String) { … } // po doInBackgr.
}
```

Async Task
(onPre/PostExecue)

```
private inner class MyTask : AsyncTask<String, Int, String>() {
    var color : Int = Color.BLACK
    override fun onPreExecute() {
        color =  ... Random Color ...
    }
    override fun doInBackground(vararg params: String): String { ...}

    override fun onProgressUpdate(vararg values: Int?) {
        super.onProgressUpdate(*values)
        val counter = values.get(0)
        myTextView.setTextColor(color)
        myTextView.text = "Counter = $counter"
    }

    override fun onPostExecute(result: String) {
        myTextView.setTextColor(color)
        myTextView.text = result
    }
}
```

Project:AsyncSmallExample.zip

Async Task
(spustenie)

Štandardne sa rôzne inštancie AsyncTask púšťajú sériovo,
kým nedobehne jeden, ostatné čakajú vo fronte

```
val task1 = MyTask().execute()  // serial run of AsyncTask
```

Ak ich chceme spustiť viacero a paralelne, tak cez POOL_EXECUTOR

```
task = MyTask().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR)
```

Ale počet paralelne bežiacich AsyncTaskov je limitovaný počtom 2*CPU+1

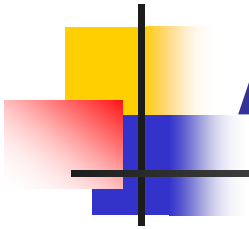
```
val cpu_cores = Runtime.getRuntime().availableProcessors()
```

Reálne väčším problémom, že napriek popularite a jednoduchosti používania AsyncTask je od Android 11 AsyncTask zastaralý (*deprecated*)

Z toho zatiaľ nie je jasné, že ho Google odstráni, ale ...

Čo je alternatíva: Coroutines !

Project:AsyncSmallExample.zip



Alternatíva

- RX-library
- Kotlin coroutines
- implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.2«
- https://kotlinlang.org/docs/tutorials/coroutines/coroutines-basic-jvm.html

Corutina

(Spustenie – blokujúce, neblokujúce)

```
Log.d(TAG, "Start")
GlobalScope.launch {   // Start a coroutine, non-blocking
    delay(1000)        // wait 1s.
    Log.d(TAG, "Hello")
}
Thread.sleep(3000)     // wait for 3s.
Log.d(TAG, "Stop")
runBlocking {          // Start a coroutine, blocking
    delay(4000L)
}
Log.d(TAG, "Finish")
```

21:22:18.220 Start
21:22:19.225 Hello Start+1sec.
21:22:21.222 Stop Start+3sec.
21:22:25.225 Finish Start+7sec.

https://simply-how.com/kotlin-coroutines-by-example-guide

Corutina
(suspend)

```
Log.d(TAG, "Start")
runBlocking {
    printHello()
}
Log.d(TAG, "Finish")


suspend fun printHello() {
    delay(1000L)
    Log.d(TAG,"Hello")
}
```

21:27:34.083 Start
21:27:35.089 Hello Start+1sec.
21:27:35.089 Finish Start+1sec.

https://simply-how.com/kotlin-coroutines-by-example-guide

Corutina

(suspend)

```
Log.d(TAG,"The main program is started")
GlobalScope.launch {
    Log.d(TAG,"Background processing started")
    delay(1000L)
    Log.d(TAG,"Background processing finished")
}
Log.d(TAG,"The main program continues")
runBlocking {
    delay(2000L)
    Log.d(TAG,"The main program is finished")
}
```

```
21:33:51.083 The main program is started
21:33:51.084 Background processing started
21:33:51.084 The main program continues
21:33:52.090 Background processing finished        Start+1sec.
21:33:53.086 The main program is finished          Start+3sec.
```

https://simply-how.com/kotlin-coroutines-by-example-guide

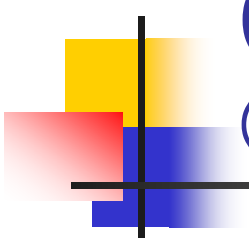
Corutina

(async/await)

```
21:38:31.369 Awaiting computations...
21:38:32.375 Computation1 finished    Start+1sec.
21:38:33.376 Computation2 finished    Start+2sec.
21:38:33.378 The result is 3          Start+2sec.
```

```
runBlocking {
        val result1 = async { computation1() }
        val result2 = async { computation2() }
        Log.d(TAG, "Awaiting computations...")
        val result = result1.await() + result2.await()
        Log.d(TAG, "The result is $result")
    }
}
suspend fun computation1(): Int {
    delay(1000L) // simulated computation
    Log.d(TAG,"Computation1 finished")
    return 1
}
suspend fun computation2(): Int {
    delay(2000L)
    Log.d(TAG,"Computation2 finished")
    return 2
}
```

https://simply-how.com/kotlin-coroutines-by-example-guide



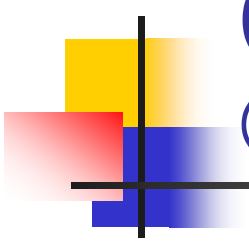
Corutina

(cancel)

```
21:44:52. Processing 0 ...
21:44:53. Processing 1 ...
21:44:54. Processing 2 ...
21:44:55. Processing 3 ...
21:44:56. Processing 4 ...
21:44:57. Processing 5 ...
21:44:58. Processing 6 ...
21:44:59. Processing 7 ...
21:45:00. Processing 8 ...
21:45:01. Processing 9 ...
21:45:02. main: The user requests the cancellation
21:45:02. main: The batch is cancelled
```

```
runBlocking {
    val job = launch { // Emulate some batch processing
        repeat(30) { i ->
            Log.d(TAG, "Processing $i ...")
            delay(1000L)
        }
    }
    delay(10000L)
    Log.d(TAG, "main: The user requests the cancellation")
    job.cancelAndJoin()
       // cancel the job and wait for it's completion
    Log.d(TAG, "main: The batch is cancelled")
}
```

https://simply-how.com/kotlin-coroutines-by-example-guide



Corutina

(waitTimeout)

```
21:47:57 Processing 0 ...
21:47:58 Processing 1 ...
21:47:59 Processing 2 ...
21:48:00 Processing 3 ...
21:48:01 Processing 4 ...
21:48:02 Processing 5 ...
21:48:03 Processing 6 ...
21:48:04 Processing 7 ...
21:48:05 Processing 8 ...
21:48:06 Processing 9 ...
21:48:07 The processing return status is: null
```

```
runBlocking {
    val status = withTimeoutOrNull(10000L) {
        repeat(30) { i ->
            Log.d(TAG, "Processing $i ...")
            delay(1000L)
        }
        "Finished"
    }
    Log.d(TAG, "The processing return status is: $status")
}
```

https://simply-how.com/kotlin-coroutines-by-example-guide