ToiletPaper #112

Kotlin Coroutinen

Autor: Marco Pfattner / Senior Software Architect/ Business Division Automotive Bavaria

✗ Problem

Bestimmte Aktionen sollen im Hintergrund ausgeführt werden, um den Main-Thread der App nicht zu blockieren. Typischerweise werden dafür Threads oder AsyncTasks verwendet: die Aktion im Background ausgeführt und schließlich wird das Ergebnis auf dem Main-Thread dargestellt.

✓ Lösung

Kotlin bietet einen etwas eleganteren Weg mit Coroutinen an. Coroutinen sind leichtgewichtige Threads, die auf einen von Kotlin verwalteten Thread-Pool dispatched werden, sofern man vorhandene Dispatcher verwendet. Vergleichbar ist das Ganze mit GCD von iOS.

➔ Beispiel

Ein Image soll heruntergeladen und dargestellt werden:

```
private fun downloadImageWithCoroutines() {
    GlobalScope.launch(Dispatchers.Main) {
        binding.status.text = "Downloading image ..."
        // Download the image on an IO dispatcher without blocking the main thread
        val image = withContext(Dispatchers.IO) { downloadImage() }
        // Continue on the main thread
        binding.image.setImageDrawable(image)
    }
}
```

Zunächst wird im oberen Beispiel auf dem Main-Thread der Status gesetzt, anschließend erfolgt ein asynchroner Download des Images. Die Besonderheit dabei ist, dass der Main-Thread nicht blockiert wird, sondern erst, wenn das Ergebnis verfügbar ist, läuft der restliche Code auf dem Main-Thread weiter.
Technisch wird das von Kotlin so gelöst, dass der Code nach dem `withContext` als Continuation ausgeführt wird. Man könnte die `downloadImage` Methode also so umschreiben, dass diese das Ergebnis einer Continuation übergibt und nicht direkt zurückliefert. Natürlich muss sich hier der Aufrufer der Methode um das entsprechende Threading kümmern:

```
private fun downloadImage(continuation: (Drawable?) -> Unit) {
    val drawable = url.openStream()?.use {
        Drawable.createFromStream(it, "image")
    }
    continuation(drawable)
}
```

Eine besondere Stärke ist die Kombination mehrerer `async` Calls. Die `async`-Calls können hier gleichzeitig auf unterschiedlichen IO-Threads ausgeführt werden:

```
// Start multiple requests
val image1 = async(Dispatchers.IO) { downloadImage1() }
val image2 = async(Dispatchers.IO) { downloadImage2() }
// Wait for the results and show the images
binding.image1.setImageDrawable(image1.await())
binding.image2.setImageDrawable(image2.await())
```

+ Weiterführende Aspekte

- Kotlin Under the Hood
- Suspending Functions
- Continuations