Kotlin 协程实战训练营（一）讲义

Kotlin 协程是什么

- 协程是什么?
  - 协程是一种在程序中处理并发任务的方案，也是这种方案的一个组件。
  - 它和线程属于一个层级的概念，是一种和线程不同的并发任务解决方案：一套系统（可以是操作系统，也可以是一种编程语言）可以选择不同的方案来处理并发任务，你可以使用线程，也可以使用协程。
- Kotlin 的协程是什么?
  - Kotlin 的协程和广义的协程不是一种东西，Kotlin 的协程（确切说是 Kotlin for Java 的协程）是一个线程框架。

Kotlin 协程的代码怎么写

用协程来开启后台任务（其实就是后台线程）：

```
GlobalScope.launch {
    println("Coroutines Camp 1
${Thread.currentThread().name}")
}
```

用协程来开启反复切换线程的任务：

```
GlobalScope.launch(Dispatchers.Main) {
    ioCode1()
    uiCode1()
    ioCode2()
    uiCode2()
    ioCode3()
    uiCode3()
```

```
}

...

private suspend fun ioCode1() {
    withContext(Dispatchers.IO) {
        Thread.sleep(1000)
        println("Coroutines Camp io1
${Thread.currentThread().name}")
    }
}

private suspend fun ioCode2() {
    withContext(Dispatchers.IO) {
        Thread.sleep(1000)
        println("Coroutines Camp io2
${Thread.currentThread().name}")
    }
}

private suspend fun ioCode3() {
    withContext(Dispatchers.IO) {
        Thread.sleep(1000)
        println("Coroutines Camp io3
${Thread.currentThread().name}")
    }
}

private fun uiCode1() {
    println("Coroutines Camp ui1
${Thread.currentThread().name}")
}

private fun uiCode2() {
    println("Coroutines Camp ui2
${Thread.currentThread().name}")
}
```

```
private fun uiCode3() {
    println("Coroutines Camp ui3
${Thread.currentThread().name}")
}
```

写法总结：

- 用 launch() 来开启一段协程 一般需要指定 Dispatchers.Main
- 把要在后台工作的函数，写成 suspend 函数 而且需要在内部调用其他 suspend 函数来真正切线程
- 按照一条线写下来，线程会自动切换

如果用线程?

倒是也能写：

```
thread {
    ioCode1()
    runOnUiThread {
        uiCode1()
        thread {
            ioCode2()
            runOnUiThread {
                uiCode2()
                thread {
                    ioCode3()
                    runOnUiThread {
                        uiCode3()
                    }
                }
            }
        }
    }
}
```

只是……有点麻烦。

协程的额外天然优势：性能

- 程序什么时候会需要切线程?
    - 工作比较耗时：放在后台
    - 工作比较特殊：放在指定线程——一般来说，是主线程
- 「耗时代码」无法完美判断，导致程序的部分性能问题。
- 协程让函数的创建者可以对耗时代码进行标记（使用 suspend 关键字），从而所有耗时代码会 100% 放在后台执行，这方面的性能问题被彻底解决。
- 不用协程为什么不行?
    - 因为协程切换的关键点是「自动」「切回来」，而这两点需要给挂起函数提供充足的上下文，即「我应该往哪回」
    - 这是直接使用线程无法做到的，或者说，非常难做到（有多难？参考协程源代码）。

再回顾：Kotlin 的协程是什么?

- 线程框架
- 可以用看起来同步的代码写出实质上异步的操作
    - 关键亮点一：耗时函数自动后台，从而提高性能
    - 关键亮点二：线程的「自动切回」

suspend 关键字

- 并不是用来切线程的
- 关键作用：标记和提醒
- 「对于编译也有用啊？」
    - 这种说法是对的。的确是有这个作用，可以辅助编译
    - 但这不是 Kotlin 这个语言层面本身的特性。（但不要和面试官抬杠！）

> 更多新技术分享，关注 B 站和微信公众号「扔物线」： https://space.bilibili.com/27559447