Kotlin 协程实战训练营（二）讲义

Retrofit 对协程的支持

```
@GET("users/{user}/repos")
suspend fun listReposKt(@Path("user") user: String):
List<Repo>
```

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

val api = retrofit.create(Api::class.java)

GlobalScope.launch(Dispatchers.Main) {
    try {
        val repos = api.listReposKt("rengwuxian") // 后台
        textView.text = repos[0].name // 前台
    } catch (e: Exception) {
        textView.text = e.message // 出错
    }
}
```

协程和 RxJava

- 都可以切线程
- 都不需要嵌套
- 都很强大，应用场景越来越接近
- RxJava 需要回调和包装，协程只需要在保证在协程里调用就行

协程和 Architecture Components

- 协程泄露：本质上是线程泄露
  - 另外，还记得课上讲的内存泄露的本质吗?
- CoroutineScope：「结构化并发」，结构化管理协程
- Lifecycle、ViewModel、LiveData、Room 的支持：关键在于对「协程」的理解

协程和线程

- 协程和线程分别是什么?
  - 线程就是线程，协程是一个线程库
- 协程和线程哪个容易使用?
  - 当然是协程了，你作为一个上层库，还没原型好使，那要你何用?
  - 所以应该问：协程和 Executor ，哪个容易使用?
    - 一般来说还是协程，这东西实在有点太突破了，关键就是它的「消除回调」
- 协程相比线程的优势和缺陷?
  - 优势就是好用，强大；劣势呢？上手太难了
  - 那么……同样问一下，和 Executor 相比呢，有什么劣势?
    - 一样，难上手。没办法，它太新了。
- 那和 Handler 相比呢?
  - 首先，其实没法比，它俩也不是一个维度的东西。Handler 相当于一个「只负责 Android 中切线程」的特殊场景化的 Executor，在 Android 中你要想让协程切到主线程，还是得用 Handler。
  - 如果我就是要强行对比协程和 Handler，它有什么优劣?
    - 我们要真是从易用性上面来说，你用协程来往主线程切，还真的是比直接用 Handler 更好写、更方便的。这个……应该也算是个比较强行的优势?

本质探秘

- 协程是怎么切线程的?

    - 最终还是使用了原生的线程切换（以及 Android 的 Handler）
- 协程为什么可以从主线程「挂起」，却不卡主线程?
    - 因为所谓的「从主线程挂起」，其实是结束了在主线程的执行，把后面的代码放在了后台线程执行，以及在后台线程的工作做完后，再把更靠后的代码通过 `Handler.post()` 又抛回主线程
- 协程的 delay() 和 Thread.sleep()
    - delay() 性能更好吗？并没有
    - 那它为什么不卡线程？它只是不卡当前线程，而去卡了别的线程

更多、更深的高级开发技术

Kotlin、协程、Jetpack，都是最新最好用的技术。但对于个人提升，各种技术基础同样重要（甚至往往更重要）。如果你是一个希望快速提升核心技术的中高级工程师，报名我们的 Android 高级进阶系列化课程是一个非常好而且非常快速的选择。

如果你需要暴力提升，别犹豫，赶快向我们的助教 rengwuxian001 （数据线）、rengwuxian001（双曲线）咨询课程详情和领取试听课程（记得要优惠，最近几天正在优惠期）。