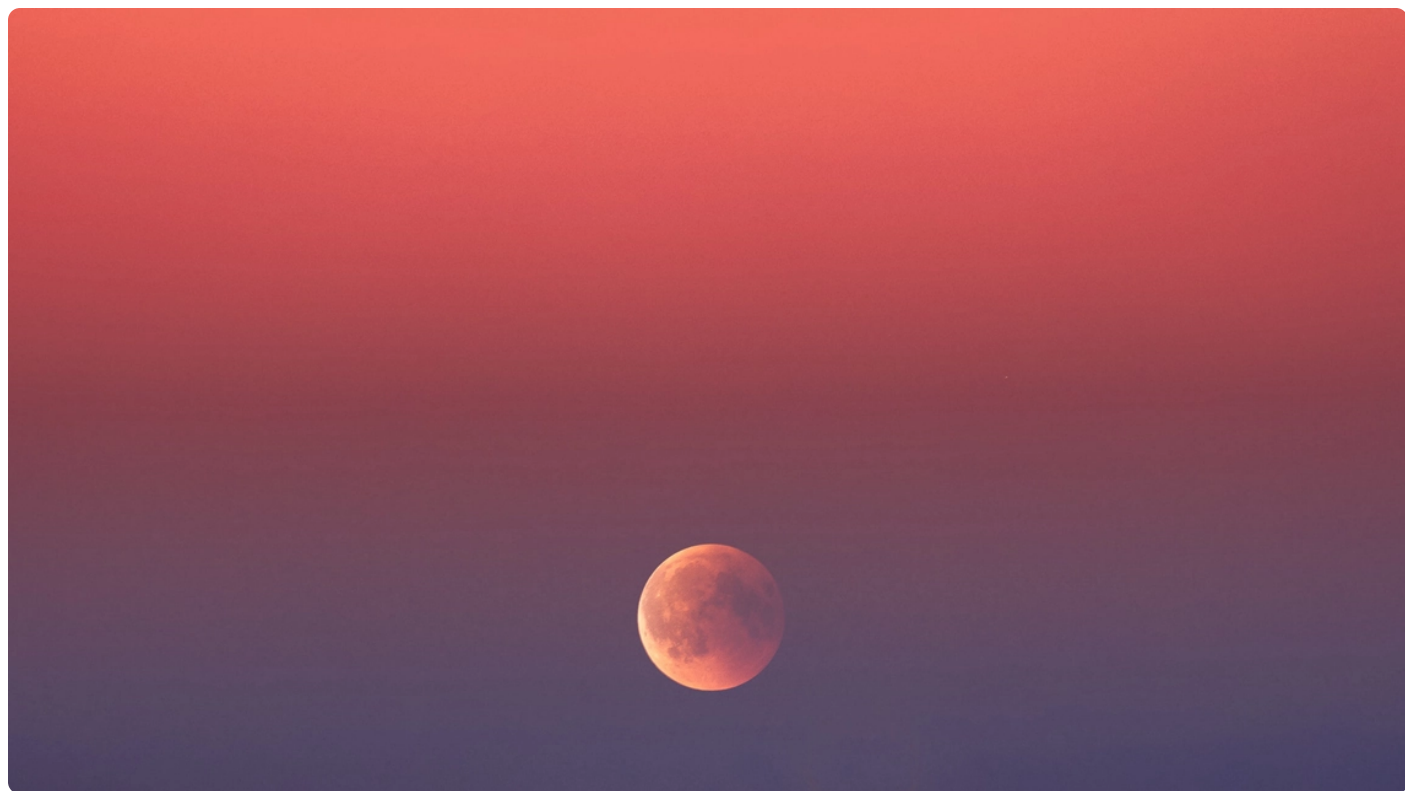


26 | 协程源码的地图：如何读源码才不会迷失？

2022-03-18 朱涛

《朱涛 · Kotlin编程第一课》

[课程介绍 >](#)



讲述：朱涛

时长 14:11 大小 13.00M



你好，我是朱涛。

在前面学习协程的时候，我们说过协程是 **Kotlin** 里最重要、最难学的特性。之所以说协程重要，是因为它有千般万般的好：挂起函数、结构化并发、非阻塞、冷数据流，等等。不过协程也真的太抽象、太难学了。即使我们学完了前面的协程篇，知道了协程的用法，但也仍然远远不够，这种“知其然，不知其所以然”的感觉，总会让我们心里不踏实。

所以，我们必须搞懂 **Kotlin** 协程的源代码。

可问题是，协程的源码也非常复杂。如果你尝试研究过协程的源代码，那你对此一定深有体会。在 **Kotlin** 协程 **1.6.0** 版本中，仅仅是协程跟 **JVM** 相关的源代码，就有 **27789** 行。如果算上 **JavaScript** 平台、**Native** 平台，以及单元测试相关的代码，**Kotlin** 协程库当中的源代码有接近 **10** 万行。面对这么多的源代码，我们根本不可能一行一行去分析。

因此，我们在研究 Kotlin 协程的源代码的时候，要有一定的技巧。这里给你分享我的两个小技巧：

- **理解 Kotlin 协程的源码结构。**Kotlin 协程的源代码分布在多个模块之中，每个模块都会包含特定的协程概念。相应的，它的各个概念也有特定的层级结构，只有弄清楚各个概念之间的关系，并且建立一个类似“地图”的知识结构，我们在研究源码的时候，才不会那么容易迷失。
- **明确研究源码的目标。**正如我前面提到的，我们不可能一次性看完协程所有的源代码，所以我们在读源码的过程中，一定要有明确的目标。比如是想要了解挂起函数的原理，还是想学习协程的启动流程。

所以在接下来的课程中，我们会主要攻克 Kotlin 协程源代码中，常见知识点的实现原理，比如挂起函数、launch、Flow 等，理解其中的代码逻辑，并掌握这些功能特性所涉及的关键技术和设计思想。

今天这节课，我们先来构建一个协程源码的知识地图，掌握了这张地图之后，我们再去学习协程的实现原理时，就可以在脑海中快速查找和定位相关模块对应的源代码，也不会迷失方向了。

不过，在正式开始学习之前，我还是要提前给你打一剂预防针。课程进行到这个阶段，学习难度也进一步提升了。**不管是什么技术，研究它的底层原理永远不是一件容易的事情。**因此，为了提高学习的效率，你一定要跟随课程的内容，去 IDE 里查看对应的源代码，一定要去实际运行、调试课程中给出的代码 Demo。

好，我们正式开始吧！

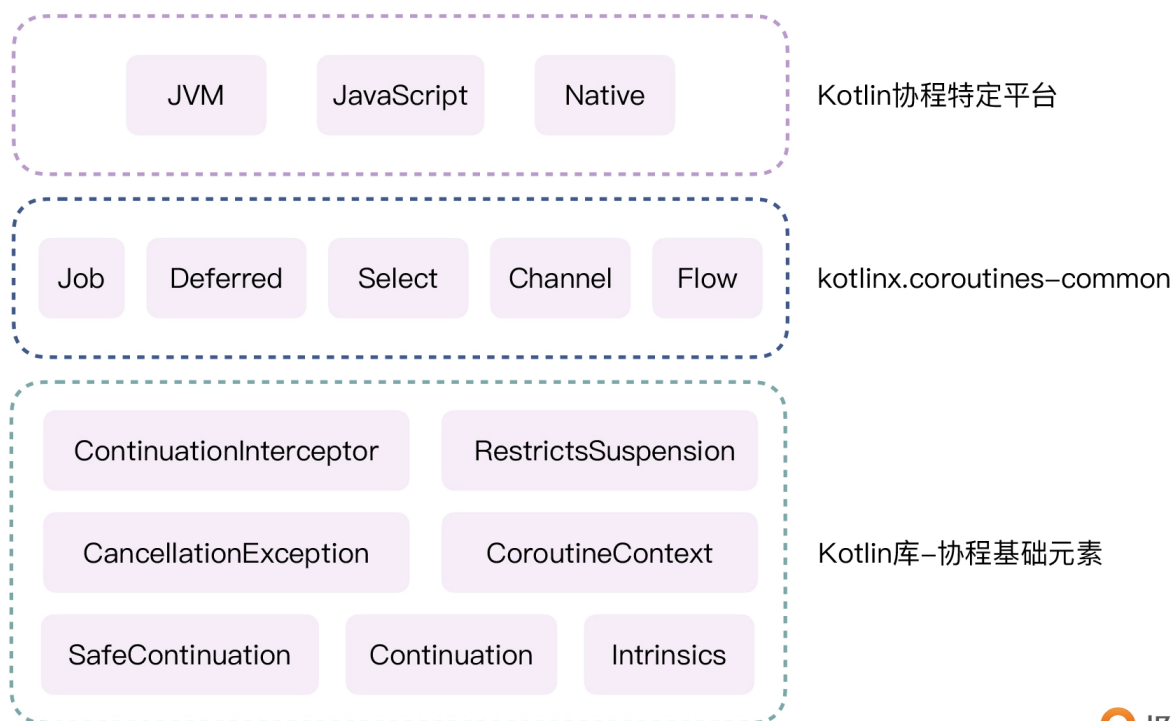
协程源码的结构

在 [🔗第 13 讲](#) 当中我们提到过，Kotlin 协程是一个独立的框架。如果想要使用 Kotlin 协程，我们需要单独进行依赖。

那么，要研究 Kotlin 协程，是不是只需研究这个协程框架的 [🔗GitHub 仓库](#) 的代码就够了呢？其实不然。因为 Kotlin 的协程源码分为了三个层级，自底向上分别是：

- **基础层：**Kotlin 库当中定义的协程基础元素；

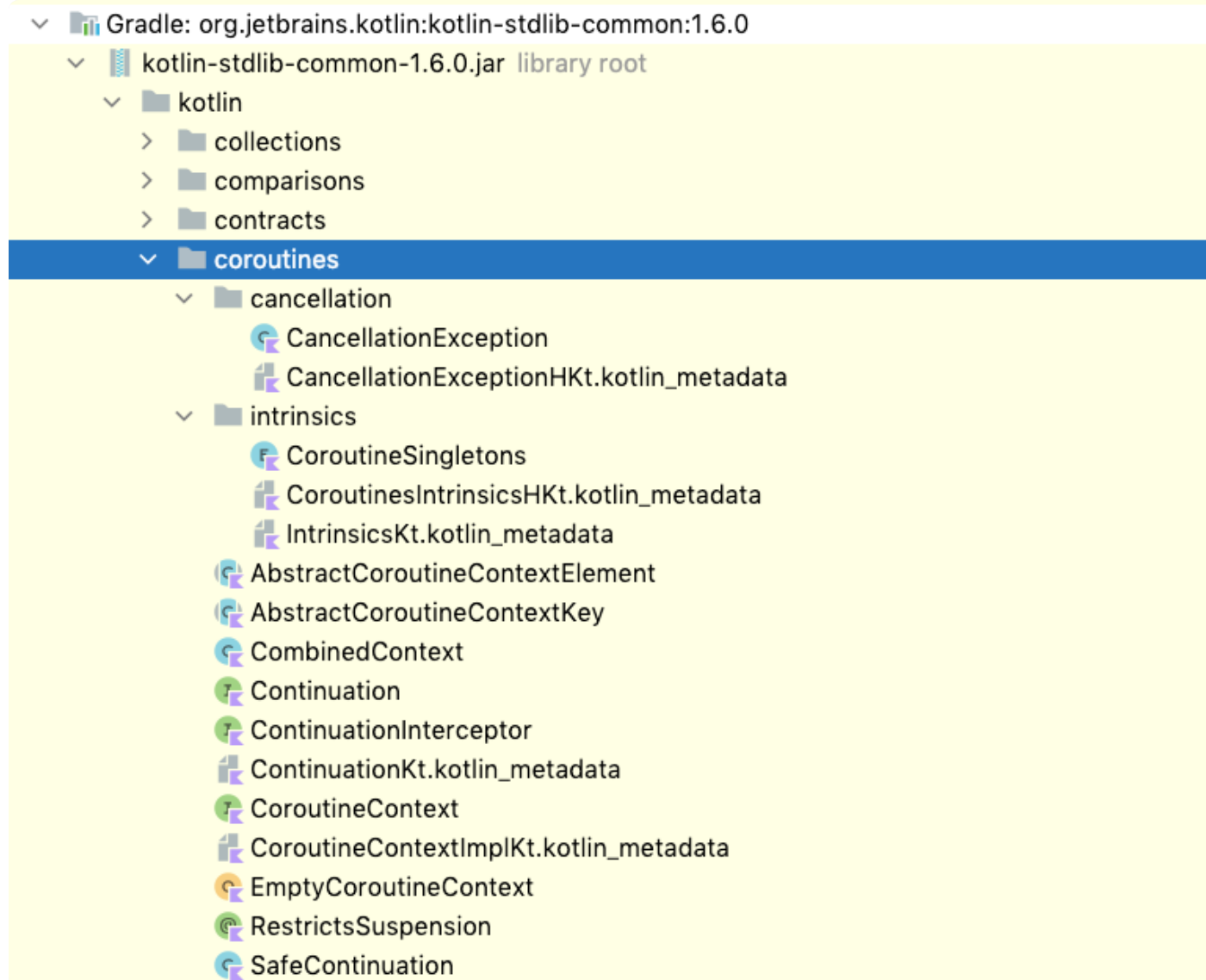
- **中间层：** 协程框架通用逻辑 `kotlinx.coroutines-common`;
- **平台层：** 这个是指协程在特定平台的实现，比如说 JVM、JS、Native。



所以，我们需要分别从这三个层级来了解协程源码的目录结构、作用类别，以及对应的功能模块的源代码。也就是说，为了研究 **Kotlin** 协程的原理，我们不仅要读协程框架的源码，同时还要读 **Kotlin** 标准库的源码。接下来，我们一个个来看。

基础层：协程基础元素

Kotlin 协程的基础元素，其实是定义在 **Kotlin** 标准库当中的。



比如，像是协程当中的一些基础概念，`Continuation`、`SafeContinuation`、`CoroutineContext`、`CombinedContext`、`CancellationException`、`intrinsics` 等等，这些概念都是定义在 Kotlin 标准库当中的。

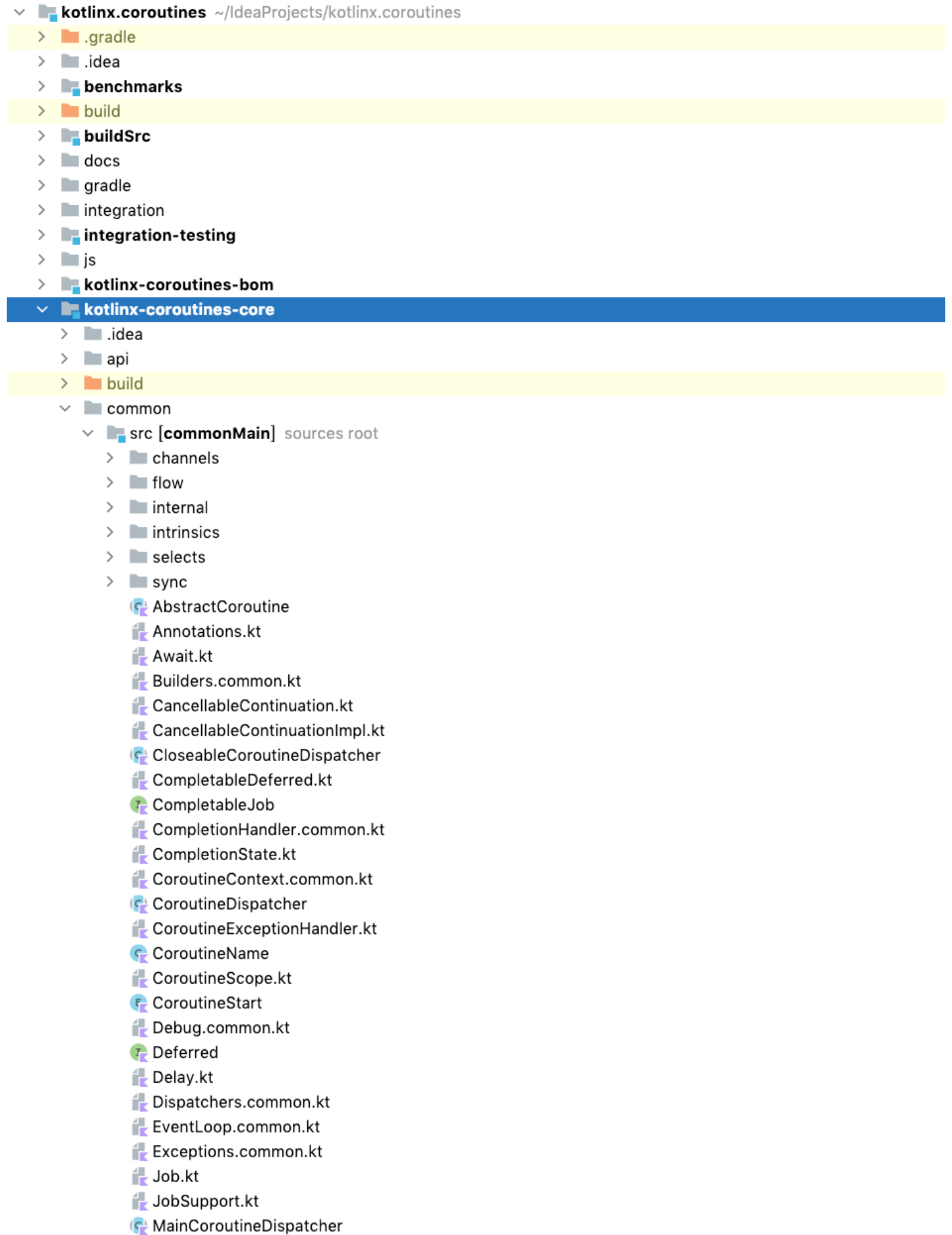
那么，Kotlin 官方为什么要这么做呢？这其实是一种解耦的思想。Kotlin 标准库当中的基础元素，就像是构造协程框架的“砖块”一样。简单的几个基础概念，将它们组合到一起，就可以实现功能强大的协程框架。

实际上，现在的 `kotlinx.coroutines` 协程框架，就是基于以上几种协程基础元素构造出来的。如果哪天 GitHub 上突然冒出一款新的 Kotlin 协程框架，你也不要觉得意外，因为构造协程的砖块就在那里，每个人都可以借助这些基础元素来构建自己的协程框架。

不过就目前来说，还是 Kotlin 官方封装的协程框架功能最强大，所以开发者也都会选择 `kotlinx.coroutines`。另外我们都知道，Kotlin 是支持跨平台的，所以协程其实也存在跨平台的实现。在 Kotlin 官方的协程框架当中，大致分了两层：`common` 中间层和平台层。

中间层：kotlinx.coroutines-common

kotlinx.coroutines 源代码当中的 common 子模块，里面包含了 Kotlin 协程框架的通用逻辑。我们前面学过的大部分知识点，都来自于这个模块，比如 launch、async、CoroutineScope、CoroutineDispatcher、Job、Deferred、Channel、Select、Flow 等。



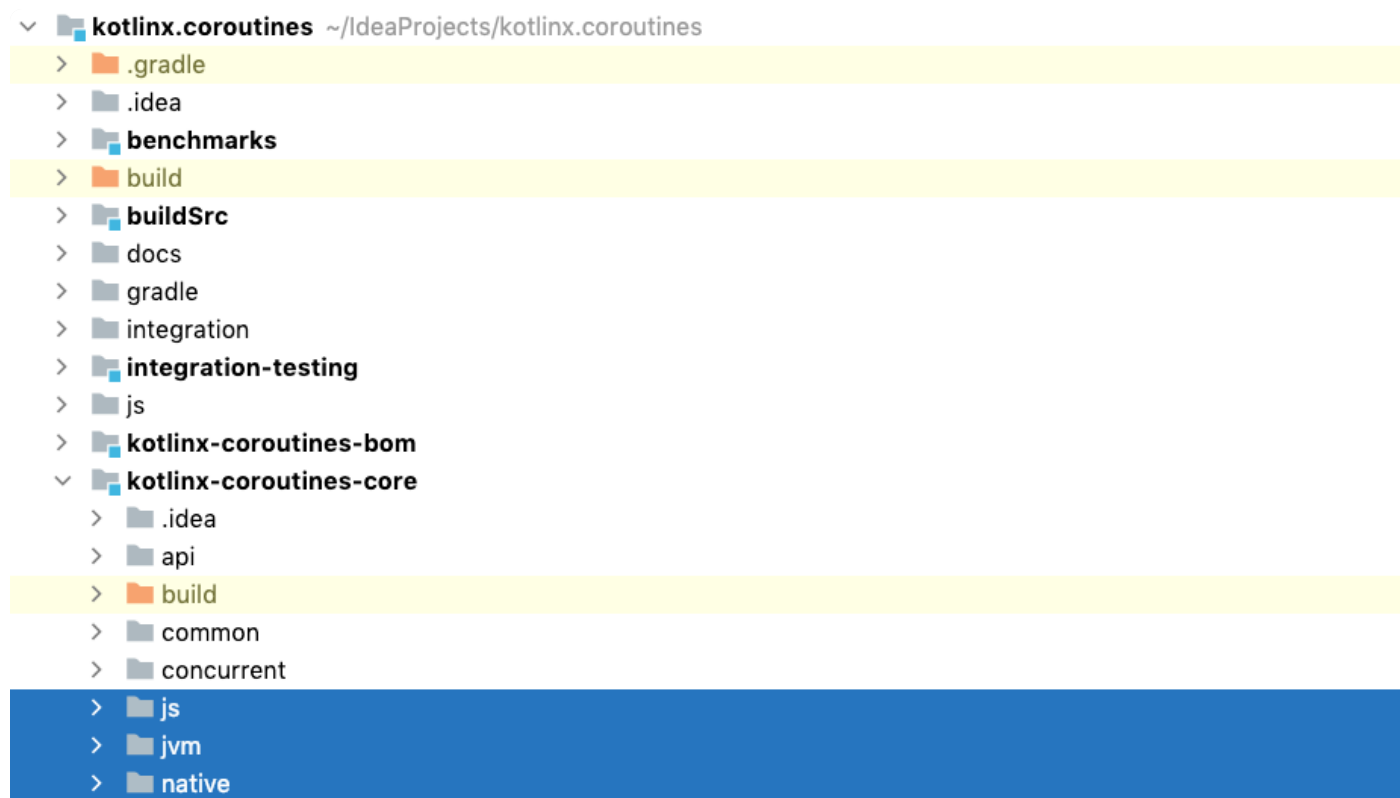
虽然说，我们开发者使用那些底层的协程基础元素，也能够写代码，但它们终归是不如 `Flow` 之类的 `API` 好用的。而 `kotlinx.coroutines-common` 这个模块，就是 `Kotlin` 官方提供的一个协

程的中间层。借助这些封装过后的高级协程概念，我们就可以直接去解决工作中的实际问题了。

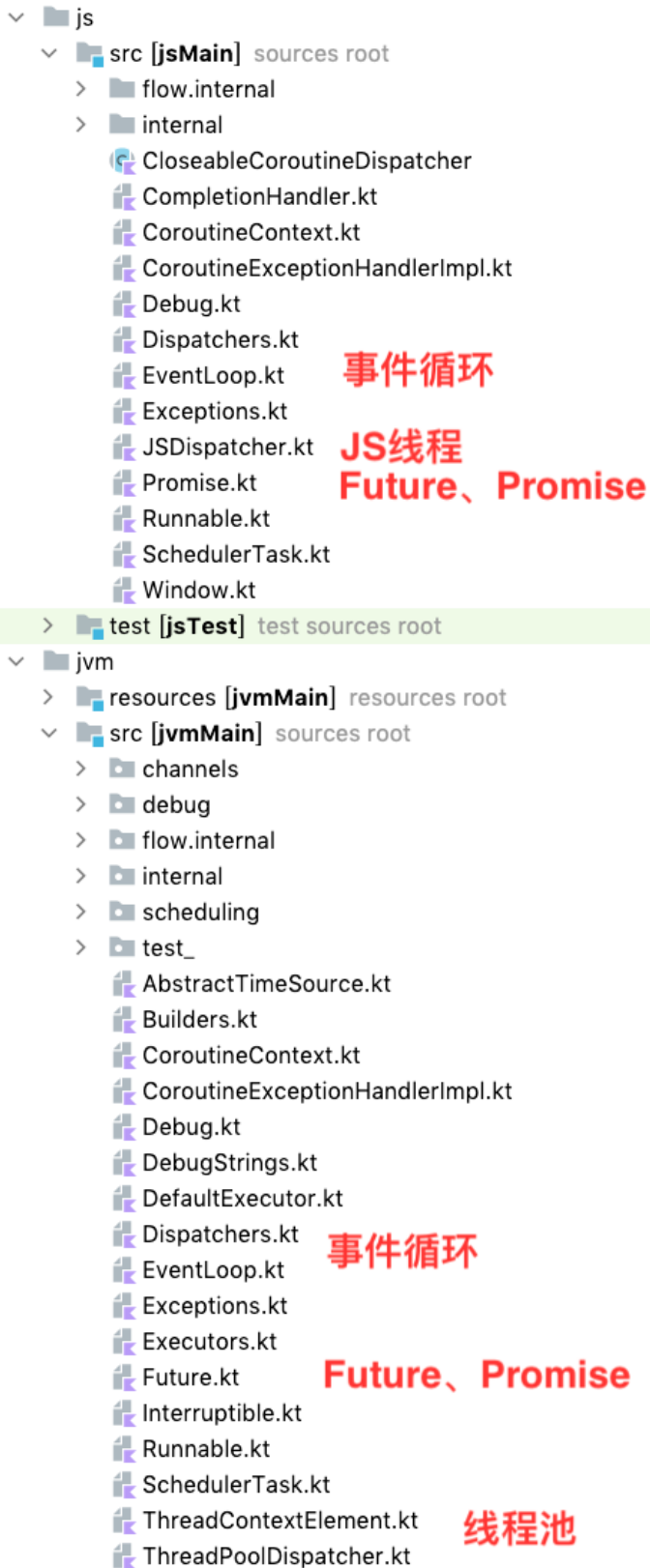
在这个 common 中间层里，**只有纯粹的协程框架逻辑，不会包含任何特定的平台特性**。而我们知道，Kotlin 其实是支持 3 种平台的：JVM、JavaScript、Native。所以针对平台的支持逻辑，都在下面的平台层当中。

平台层

在 core 模块之下，有几个与 common 平级的子模块，即 JVM、JavaScript、Native。这里面，才是 Kotlin 协程与某个平台产生关联的地方。



我们都知道，Kotlin 的协程，最终都是运行在线程之上的。所以，当 Kotlin 在不同平台上运行的时候，最终还需要映射到对应的线程模型之上。这里我们就以 JVM 和 JavaScript 为例：



可以看到，同样的协程概念，在 JVM、JavaScript 两个平台上会有不同的实现：

- 同样是线程，在 JVM 是线程池，而 JavaScript 则是 JS 线程；
- 同样是事件循环，两者也会有不同的实现方式；
- 同样是异步任务，JVM 是 Future，JavaScript 则是 Promise。

可见，虽然协程的“平台层”是建立在 `common` 层之上的，但它同时又为协程在特定平台上提供了对应的支持。

好，到这里，我们就已经弄清楚 Kotlin 协程的源码结构了。这个源码的结构，也就相当于协程知识点的**地图**。有了这个地图以后，我们在后面遇到问题的时候，才知道去哪里找答案。比如说，当我们想知道 Kotlin 的协程是如何运行在线程之上的，那么我们肯定要到平台层，去找 JVM 的具体实现。

如何研究协程源码？

读 Kotlin 协程的源代码，就像是一场原始森林里的探险一样。我们不仅要有一张清晰的地图，同时还要有**明确的目标**。

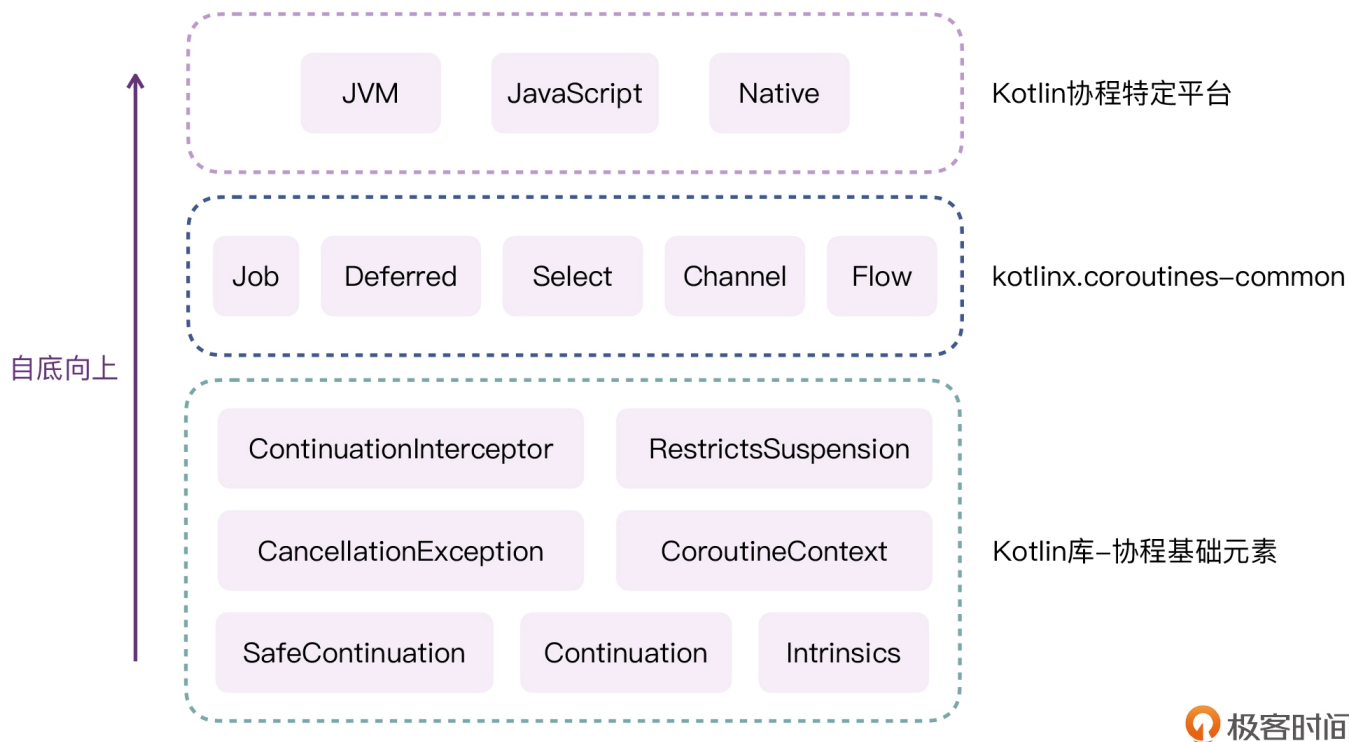
所以在接下来的源码篇当中，我们每一节课的学习目标都会非常明确，比如我们会来着重探究挂起函数的原理、协程启动原理、Dispatchers 原理、CoroutineScope 原理、Channel 原理，还有 Flow 的原理。这些都是 Kotlin 协程当中最基础、最重要的知识点，掌握了它们的原理以后，我们在工作中使用协程时也会更有底气。就算遇到了问题，我们也可以通过读源码找到解决方案。

不过，即使有了探索的目标也还不够，在正式开始之前，我们还需要做一些额外的准备工作。

首先，**我们要掌握好协程的调试技巧**。在之后的课程当中，我们会编写一些简单的 Demo，然后通过运行调试这些 Demo，一步步去跟踪、分析协程的源代码。因此，如果你还没看过 [🔗 第 14 讲](#) 的内容，一定要回过头去看一下其中关于协程调试的内容。

其次，**我们要彻底弄懂协程的基础元素**。前面我提到过，Kotlin 标准库当中的协程基础元素就像是构建协程框架的砖块一样。如果我们对协程的基础元素一知半解的话，在后面分析协程框架的过程中，就会寸步难行。

所以接下来，面对协程源码的三层结构：基础层、中间层、平台层，我们必须**自底向上**，一步步进行分析。



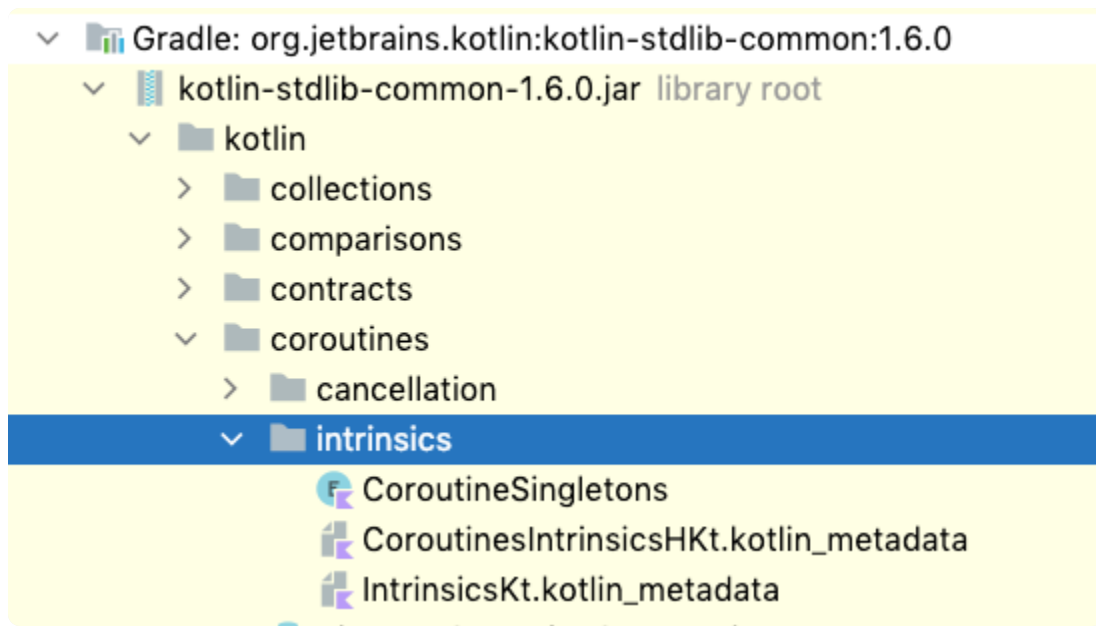
Kotlin 源码编译细节

另外，我们在平时用 Kotlin 协程的时候，一般只会使用依赖的方式：

```
1 implementation "org.jetbrains.kotlin:kotlin-stdlib"
2 implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.0'
```

复制代码

不过使用这种方式，我们会经常遇到某些类看不到源代码实现的情况。比如，`kotlin.coroutines.intrinsics` 这个包下的源代码：



那么在这里，我也分享一下我读 Kotlin 源码的方式，给你作为参考。

首先，当遇到依赖包当中无法查看的类时，你可以去 GitHub 下载 [Kotlin](#) 和 [Coroutines](#) 的源代码，然后按照上面画的“协程源码地图”去找对应的源代码实现。

然后，在 IDE 当中导入这两个工程的时候，可能也会遇到各种各样的问题。这时候，你需要参考这两个链接里的内容：[Coroutine Contributing Guidelines](#)、[Kotlin Build environment requirements](#)，来配置好 Kotlin 和 Coroutines 的编译环境。

完成了这两个工程的导入工作以后，你就可以看到 Kotlin 和协程所有的源代码了。这里不仅有它们的核心代码，还会有跨平台实现、编译器实现，以及对应的单元测试代码。这样后面你在读 Kotlin 源码的时候，才会有更大的自由度。

小结

这节课的内容到这里就差不多结束了。接下来，我们来做一个简单的总结。

研究 Kotlin 协程的源代码，我们要注意两个要点：**理解 Kotlin 协程的源码结构、明确研究源码的目标**。如果我们把读源码当做是一次原始森林的探险，那么前者就相当于我们手中的**探险地图**，后者就相当于地图上的**探索目标和行进路线**。

有了这两个保障以后，我们才不会轻易迷失在浩瀚的协程源码中。

那么，对于协程的源码结构来说，主要可以分为三层。

- **基础层**：Kotlin 库当中定义的协程基础元素。如果说协程框架是一栋大楼，那么这些基础元素，就相当于一个个的砖块。
- **中间层**：协程框架通用逻辑 `kotlinx.coroutines-common`。协程框架里的 `Job`、`Deferred`、`Channel`、`Flow`，它们都是通过协程基础元素组合出来的高级概念。这些概念跟平台无关，不管协程运行在 JVM、JavaScript，还是 Native 上，这些概念都是不会变的。而这些概念的实现，全部都在协程的 `common` 中间层。
- **平台层**：最后就是协程在特定平台的实现，比如说 JVM、JavaScript、Native。当协程要在某个平台运行的时候，它总是免不了要跟这个平台打交道。比如 JVM，协程并不能脱离线程运行，因此协程最终还是会运行在 JVM 的线程池当中。

这节课的作用跟前面第 13 讲的作用其实是差不多的。毕竟在探险之前，我们总要做一些准备工作。另外最后我也想再强调一点，就是我之所以先带你梳理协程源码的结构，也是因为如果我一上来就给你贴一大堆源代码，开始跟你分析代码的执行流程，一定会很难接受和消化吸收。

所以，也请你不要轻视这节课的作用，一定要做好充足的准备，再出发。

思考题

在 Kotlin 协程的基础元素当中，最重要的其实就是 **Continuation** 这个接口。不过，在 [Continuation.kt](#) 这个文件当中，还有两个重要的扩展函数：


 复制代码

```
1 public interface Continuation<in T> {
2
3     public val context: CoroutineContext
4
5     public fun resumeWith(result: Result<T>)
6 }
7
8 public fun <T> (suspend () -> T).createCoroutine(
9     completion: Continuation<T>
10 ): Continuation<Unit> =
11     SafeContinuation(createCoroutineUnintercepted(completion).intercepted(), CC
12
13 public fun <T> (suspend () -> T).startCoroutine(
14     completion: Continuation<T>
15 ) {
16     createCoroutineUnintercepted(completion).intercepted().resume(Unit)
17 }
```

请问你能猜到它们的作用是什么吗？这个问题的答案，我会在第 28 讲给出。

分享给需要的人，Ta 订阅超级会员，你最高得 50 元

Ta 单独购买本课程，你将得 20 元

 生成海报并分享

上一篇 25 | 集合操作符：你也会“看完就忘”吗？

下一篇 27 | 图解挂起函数：原来你就是个状态机？

精选留言 (2)

写留言



神秘嘉Bin

2022-03-21

创....创建协程？

作者回复：是啊



👍 2



Paul Shan

2022-03-24

第一个函数好像是从挂起函数构建状态为挂起的代码单元模块

第二个函数好像是从挂起函数构建状态为执行的代码单元模块

作者回复：是的

