文本, 信件

AI 生成的内容可能不正确。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

表格

AI 生成的内容可能不正确。

在 C++ 中，显式挂起点使用关键字 co\_await 和 co\_yield 标记。下图展示了一个协程如何从一个子程序中被调用，然后在代码的不同部分被恢复：  
图 12.2：协程的调用可以挂起和恢复。协程调用在挂起时会保持其内部状态。

协程挂起时，其内部局部变量的状态会被保留。这些状态属于协程的某次特定调用。也就是说，它们不像静态局部变量那样在函数的所有调用中全局共享。

总结一下，协程是可以被挂起和恢复的子程序。换个角度看，子程序是不能被挂起或恢复的协程的一种特殊情况。

从现在起，我会严格区分 “调用” 和 “恢复”，以及 “挂起” 和 “返回”。它们的含义完全不同。调用一个协程会创建一个可以被挂起和恢复的协程实例。从协程返回会销毁该协程实例，并且它不能再被恢复。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

PC 和 SP 相互配合，共同确保 CPU 能够正确地执行程序。PC 控制着程序的执行流程，决定了 CPU 下一步要执行的指令；而 SP 则负责管理栈空间，为函数调用和返回提供必要的支持。

文本

AI 生成的内容可能不正确。

文本, 信件

AI 生成的内容可能不正确。

文本, 信件

AI 生成的内容可能不正确。

文本, 信件

AI 生成的内容可能不正确。

协程是暂停加返回。

协程是一种抽象机制，使我们能够以清晰简洁的方式编写惰性求值代码和异步程序。但创建和销毁协程以及挂起和恢复协程都有一定的性能开销。在比较无栈协程和有栈协程的性能开销时，需要考虑两个主要方面：内存占用和上下文切换。

文本

AI 生成的内容可能不正确。

文本, 信件

AI 生成的内容可能不正确。