```
    doOnCompleted { result →
      continuation.resume(Unit)
    }
  }
```

join 的实现与 delay 如出一辙。

实际上，join 等待协程执行时还有第 3 种情况。join 本身也是一个挂起函数，因此必须在其他协程中调用，如果它所在的协程（注意，并非被等待的协程）取消，那么 join 会立即抛出 CancellationException 来响应取消。这一点我们会在 5.5 节继续对 join 函数的实现进行完善。

5.3.4 有返回值的 async

现在我们已经知道如何启动协程并等待协程执行完成，不过很多时候我们更想拿到协程的返回值，因此我们基于 Job 再定义一个接口 Deferred，如代码清单 5-21 所示。

代码清单 5-21 Deferred 的定义

```
interface Deferred<T>: Job {
  suspend fun await(): T
}
```

这里多了一个泛型参数 T，T 表示返回值类型，通过它的 await 函数也可以拿到这个返回值。因此 await 的作用主要有：

- 在协程已经执行完成时，立即返回协程的结果；如果协程异常结束，则抛出该异常。
- 如果协程尚未完成，则挂起直到协程执行完成，这一点与 join 类似。

我们定义一个 DeferredCoroutine 类来实现这个接口，如代码清单 5-22 所示。

代码清单 5-22 DeferredCoroutine 的实现

```
class DeferredCoroutine<T>(context: CoroutineContext)
 : AbstractCoroutine<T>(context), Deferred<T> {

  override suspend fun await(): T {
    val currentState = state.get()
    return when (currentState) {
      is CoroutineState.Incomplete,
      is CoroutineState.Cancelling → awaitSuspend()
      is CoroutineState.Complete<*> → {
        currentState.exception?.let { throw it }
```