10.1 与并发相关的错误类型

你可以在并发代码中发现各式各样的错误，这些错误不会集中于某个方面。不过，有一些错误与使用并发直接相关，本章重点关注这些错误。通常，并发相关的错误通常有两大类：

- 不必要阻塞
- 条件竞争

这两大类的颗粒度很大，让我们将其分成颗粒度较小的问题。

10.1.1 不必要阻塞

“不必要阻塞”是什么意思？一个线程被阻塞的时候，不能处理任何任务，因为它在等待其他“条件”的达成。通常这些“条件”就是一个互斥量、一个条件变量或一个future，也可能是一个I/O操作。这是多线程代码的先天特性，不过这也不是在任何时候都可取的——衍生成“不必要阻塞”。你会问：为什么不需要阻塞？通常，是因为其他线程在等待该阻塞线程上的某些操作完成，如果该线程阻塞了，那那些线程必然会被阻塞。

这个主题可以分成以下几个问题：

- 死锁——如你在第3章所见，在死锁的情况下，两个线程会互相等待。当线程产生死锁，应该完成的任务就会持续搁置。举个例子来说，一些线程是负责对用户界面操作的线程，在死锁的情况下，用户界面就会无响应。在另一些例子中，界面接口会保持响应，不过有些任务就无法完成，比如：查询无结果返回，或文档未打印。
- 活锁——与死锁的情况类似。不同的地方在于线程不是阻塞等待，而是在循环中持续检查，例如：自旋锁。一些比较严重的情况下，其表现和死锁一样(应用不会做任何处理，停止响应)，CPU的使用率还居高不下；因为线程还在循环中被检查，而不是阻塞等待。在一些不太严重的情况下，因为使用随机调度，活锁的问题还是可以解决的。
- I/O阻塞或外部输入——当线程被外部输入所阻塞，线程也就不能做其他事情了(即使，等待输入的情况永远不会发生)。因此，被外部输入所阻塞，就会让人不太高兴，因为可能有其他线程正在等待这个线程完成某些任务。

简单的介绍了一下“不必要阻塞”的组成。

那么，条件竞争呢？

10.1.2 条件竞争

条件竞争在多线程代码中很常见——很多条件竞争表现为死锁与活锁。而且，并非所有条件竞争都是恶性的——对独立线程相关操作的调度，决定了条件竞争发生的时间。很多条件竞争是良性的，比如：哪一个线程去处理任务队列中的下一个任务。不过，很多并发错误的引起也是因为条件竞争。

特别是，条件竞争经常会产生以下几种类型的错误：

- 数据竞争——因为未同步访问一块共享内存，将会导致代码产生未定义行为。在第5章已经介绍了数据竞争，也了解了 C++ 的内存模型。数据竞争通常发生在错误的使用原子操作，做同步线程的时候，或没使用互斥量所保护的共享数据的时候。
- 破坏不变量——主要表现为悬空指针(因为其他线程已经将要访问的数据删除了)，随机存储错误(因为局部更新，导致线程读取了不一样的数据)，以及双重释放(比如：当两个线程对同一个队列同时执行pop操作，想要删除同一个关联数据)，等等。不变量被破坏可以看作为“基于数据”的问题。当独立线程需要以一定顺序执行某些操作时，错误的同步会导致条件竞争，比如：顺序被破坏。
- 生命周期问题——虽然这类问题也能归结为破坏了不变量，不过这里将其作为一个单独的类别给出。这里的问题是，线程会访问不存在的数据，这可能是因为数据被删除或销毁了，或者转移到其他对象中去了。生命周期问题，通常是在一个线程引用了局部变量，在线程还没有完成前，局部变量的“死期”就已经到了，不过这个问题并不止存在这种情况下。当你手动调用join()等待线程完成工作，你需要保证异常抛出的时候，join()还会等待其他未完成工作的线程。这是线程中基本异常安全的应用。

恶性条件竞争就如同一个杀手。死锁和活锁会表现为：应用挂起和反应迟钝，或超长时间完成任务。当一个线程产生死锁或活锁，可以用调试器附着到该线程上进行调试。条件竞争，破坏不变量，以及生命周期问题，其表现都是代码可见的(比如，随机崩溃或错误输出)——可能重写了系统部分的内存使用方式(不会改太多)。其中，可能是因为执行时间，导致问题无法定位到具体的位置。这是共享内存系统的诅咒——需要通过线程尝试限制可访问的数据，并且还要正确的使用同步，应用中的任何线程都可以复写(可被其他线程访问的)数据。

现在已经了解了这两大类中都有哪些具体问题了。

下面就让我们来了解，如何在你的代码中定位和修复这些问题。