线程池

处理池任务之间的依赖关系

池线程的工作窃取

中断线程

线程池允许您完成这一任务;可以并发执行的任务被提交到线程池，线程池将它们放在挂起工作的队列中。然后，每个任务由一个工作线程从队列中取出，该工作线程在循环返回以从队列中取出另一个任务之前执行该任务。

在构建线程池时，有几个关键的设计问题，

例如使用多少线程，

将任务分配给线程的最有效方法，

以及是否可以等待任务完成。

文本

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

当多个线程频繁地向同一个全局任务队列中提交任务或从中获取任务时，会导致 **缓存乒乓（Cache Ping-Pong）** 和 **锁竞争**，从而降低性能。为了避免这些问题，可以为每个线程分配一个 **本地任务队列**，并优先从本地队列中获取任务，只有在本地队列为空时，才从全局队列中获取任务。

你提到的场景是关于 **任务偷取（Work Stealing）** 的实现。任务偷取是一种负载均衡机制，允许空闲线程从其他线程的任务队列中“偷取”任务来执行。为了实现任务偷取，每个线程的任务队列必须对其他线程可见，并且需要确保队列的线程安全性。

以下是一个基于 **锁（Mutex）** 的任务偷取线程池实现。虽然锁会引入一定的开销，但在任务偷取不频繁的情况下，这种实现的开销是可以接受的。

文本

中度可信度描述已自动生成

在线程池中，**中断线程** 是一种机制，用于主动停止或取消正在执行的任务。这种机制通常用于以下场景：

1. 任务执行时间过长，需要提前终止。
2. 用户取消了任务（例如，用户点击了“取消”按钮）。
3. 系统需要优雅地关闭线程池，并终止所有正在执行的任务。

C++ 标准库本身并没有直接提供线程中断的机制，但可以通过一些设计模式和技术来实现类似的功能。以下是实现线程中断的几种常见方法：

在本章中，我们研究了C++标准库中可用的并行算法以及如何使用它们。我们研究了各种执行策略，执行策略的选择对算法行为的影响，以及它对代码的限制。然后我们看了一个例子，说明如何在真实的代码中使用该算法。

<$并发相关的bug <$通过测试和代码评审定位bug <$设计多线程测试<$测试多线程代码的性能

不需要的阻塞（Unwanted blocking）竞争条件（Race conditions）

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

哪些数据需要保护以防止并发访问？

您如何确保数据受到保护？

此时其他线程可能在代码中的哪个位置？

这个线程持有哪些互斥锁？

其他线程可能持有哪些互斥锁？

在这个线程中完成的操作和在另一个线程中完成的操作之间是否有任何顺序要求？

这些要求是如何执行的？

此线程加载的数据是否仍然有效？

会不会被其他线程修改了？

如果你假设另一个线程可能正在修改数据，这意味着什么？

你如何确保这种情况永远不会发生？

在本章中，我们讨论了可能遇到的各种类型的并发相关错误，从死锁和活锁到数据竞争和其他有问题的竞争条件。我们采用了定位bug的技术。其中包括代码审查期间需要考虑的问题、编写可测试代码的指南以及如何构建并发代码的测试。最后，我们看了一些可以帮助测试的实用组件。