本章介绍了为并发设计数据结构的含义、设计准则、为并发设计的数据结构的示例实现

一种选择是使用单独的互斥锁和外部锁来保护数据，使用我们在第3章和第4章中看到的技术，另一种选择是设计数据结构本身以进行并发访问。

有两个方面需要考虑：确保访问是安全的，以及支持真正的并发访问。

确保没有线程可以看到数据结构的不变量被另一个线程的操作破坏的状态。

注意避免数据结构接口中固有的竞争条件，为完整的操作而不是操作步骤提供函数。

注意数据结构在出现异常时的行为，以确保不变量不被破坏。

通过限制锁的范围并尽可能避免嵌套锁，最大限度地减少使用数据结构时出现死锁的机会。

是否可以限制锁的作用域，以允许操作的某些部分在锁之外执行？

数据结构的不同部分可以用不同的互斥锁来保护吗？

所有操作是否都需要相同级别的保护？

对数据结构的简单更改能否在不影响操作语义的情况下提高并发性？

如何最小化必须发生的序列化的数量并实现最大量的真正并发？

确保在堆栈完全构造之前，其他线程无法访问堆栈，并且必须确保在堆栈被销毁之前，所有线程都停止访问堆栈。

 **锁的作用**：

* 锁用于保护共享资源的访问，确保同一时间只有一个线程可以进入临界区（Critical Section）。
* 常见的锁类型包括：
  + **互斥锁（Mutex）**：一次只允许一个线程进入临界区。
  + **读写锁（Read-Write Lock）**：允许多个读线程同时访问，但写线程需要独占访问。
  + **自旋锁（Spinlock）**：线程在获取锁之前会忙等待（Busy Waiting），适用于短时间的临界区。

 **临界区**：

* 临界区是指访问共享资源的代码段。在基于锁的数据结构中，所有对共享数据的访问和修改都必须在锁的保护下进行。

 **线程安全**：

* 通过锁机制，确保多个线程不会同时修改共享数据，从而避免数据竞争和不一致性。

本章首先介绍了为并发设计数据结构的意义，并提供了一些指导原则。然后，我们研究了几种常见的数据结构（堆栈、队列、哈希映射和链表），研究了如何应用这些指导原则，以一种为并发访问而设计的方式实现它们，使用锁来保护数据并防止数据竞争。您现在应该能够查看您自己的数据结构的设计，以了解并发的机会在哪里，以及竞争条件的可能性在哪里。在第7章中，我们将研究完全避免锁的方法，使用低级别的原子操作来提供必要的排序约束，同时坚持相同的指导方针。