6.1 为并发设计的意义何在？

设计并发数据结构，意味着多个线程可以并发的访问这个数据结构，线程可对这个数据结构做相同或不同的操作，并且每一个线程都能在自己的自治域中看到该数据结构。且在多线程环境下，无数据丢失和损毁，所有的数据需要维持原样，且无条件竞争。这样的数据结构，称之为“线程安全”的数据结构。通常情况下，当多个线程对数据结构进行同一并发操作是安全的，但不同操作则需要单线程独立访问数据结构。或相反，当线程执行不同的操作时，对同一数据结构的并发操作是安全的，而多线程执行同样的操作，则会出现问题。

实际的设计意义并不止上面提到的那样：这就意味着，要为线程提供并发访问数据结构的机会。本质上，是使用互斥量提供互斥特性：在互斥量的保护下，同一时间内只有一个线程可以获取互斥锁。互斥量为了保护数据，显式的阻止了线程对数据结构的并发访问。

这被称为*串行化*(serialzation)：线程轮流访问被保护的数据。这其实是对数据进行串行的访问，而非并发。因此，你需要对数据结构的设计进行仔细斟酌，确保其能真正并发访问。虽然，一些数据结构有着比其他数据结构多的并发访问范围，但是在所有情况下的思路都是一样的：减少保护区域，减少序列化操作，就能提升并发访问的潜力。

在我们进行数据结构的设计之前，让我们快速的浏览一下，在并发设计中的指导建议。

6.1.1 数据结构并发设计的指导与建议(指南)

如之前提到的，当设计并发数据结构时，有两方面需要考量：一是确保访问是安全的，二是能真正的并发访问。在第3章的时候，已经对如何保证数据结构是线程安全的做过简单的描述：

- 确保无线程能够看到，数据结构的“不变量”破坏时的状态。
- 小心那些会引起条件竞争的接口，提供完整操作的函数，而非操作步骤。
- 注意数据结构的行为是否会产生异常，从而确保“不变量”的状态稳定。
- 将死锁的概率降到最低。使用数据结构时，需要限制锁的范围，且避免嵌套锁的存在。

在你思考设计细节前，你还需要考虑这个数据结构对于使用者来说有什么限制；当一个线程通过一个特殊的函数对数据结构进行访问时，那么还有哪些函数能被其他的线程安全调用呢？

这是一个很重要的问题，普通的构造函数和析构函数需要独立访问数据结构，所以用户在使用的时候，就不能在构造函数完成前，或析构函数完成后对数据结构进行访问。当数据结构支持赋值操作，swap()，或拷贝构造时，作为数据结构的设计者，即使数据结构中有大量的函数被线程所操纵时，你也需要保证这些操作在并发环境下是安全的(或确保这些操作能够独立访问)，以保证并发访问时不会出现错误。

第二个方面是，确保真正的并发访问。这里没法提供更多的指导意见；不过，作为一个数据结构的设计者，在设计数据结构时，自行考虑以下问题：

- 锁的范围中的操作，是否允许在锁外执行？
- 数据结构中不同的区域是否能被不同的互斥量所保护？
- 所有操作都需要同级互斥量保护吗？
- 能否对数据结构进行简单的修改，以增加并发访问的概率，且不影响操作语义？

这些问题都源于一个指导思想：如何让序列化访问最小化，让真实并发最大化？允许线程并发读取的数据结构并不少见，而对数据结构的修改，必须是单线程独立访问。这种结构，类似于 `boost::shared_mutex` 。同样的，这种数据结构也很常见——支持在多线程执行不同的操作时，并序列化执行相同的操作的线程(你很快就能看到)。

最简单的线程安全结构，通常使用的是互斥量和锁，对数据进行保护。虽然，这么做还是有问题（如同在第3中提到的那样），不过这样相对简单，且保证只有一个线程在同一时间对数据结构进行一次访问。为了让你轻松的设计线程安全的数据结构，接下来了解一下基于锁的数据结构，以及第7章将提到的无锁并发数据结构的设计。