同步类容器都是线程安全的，但在某些场景下可能需要加锁来保护复合操作。复合操作如

：迭代（反复访问元素，遍历完容器中所有的元素）、跳转（根据指定的顺序招到当前元素的下一个元素）、以及条件运算。这些复合操作在多线程并发地修改容器时，可能会表现处意外的行为，最经典的便是ConcurrentModificationException，原因是当容器迭代的过程中，被并发的修改了内容，这是在早期迭代器设计的时候并没有考虑并发修改的问题。

同步类容器：如古老的Vector、HashTable。这些容器的同步功能其实都是在JDK的Collections.synchronized\*\*\*等工厂方法去创建实现的。其底层的机制无非就是用传统的synchronized关键字对每个公用的方法都进行同步，使得每次只能由一个线程访问容器的状态，这很明显不能满足今天互联网时代高并发的需求，在保证线程安全的同事，也必须有足够好的性能。

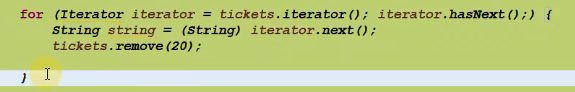
T2

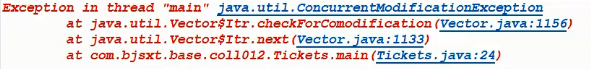
T1

vector

如图：在t1线程在进行对vector进行迭代的过程中，t2修改了vector中的值，则会导致异常。

产生ConcurrentModificationException的一种方法:







单独的容器并发修改操作依然是线程安全的

古老的实现线程安全的方式。



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

并发类容器：  
jdk5.0以后提供了多种并发类容器来替代同步类容器从而改善性能，同步类容器的状态都是串行化的，他们虽然实现了线程安全，但是严重降低了并发性，在多线程的环境下，严重降低了应用程序的吞吐量。

并发类容器是专门针对并发设计的，使用ConcurrentHashMap来代替给予散列的传统的HashTable，而且在ConcurrentHashMap中，添加了一些常见的复合操作的支持，以及使用CopyOnWriteArrayList代替Voctor，并发的CopyonWriteArraySet，以及并发的Queue，ConcurrentLinkedQueue和LinkedBlockingQueue，前者是高性能的队列，后者是阻塞形式的队列，具体实现Queue还有很多，例如ArrayBLockingQueue、PriorityBlockingQueue、SynchronousQueue等。

+++ ConcurrentMap 接口下的两个重要的实现：

ConcurrentHashMap

ConcurrentSkipListMap（支持并发排序功能，弥补ConcurrentHashMap）

ConcurrentHashm内部使用段（segment）来表示这些不同的部分，每个段其实就是一个小的HashTable，它们有自己的锁，只要多个操作发生在不同的段上，他们就可以并发进行。把一个整体分成16段，也就是最高支持16个线程的并发修改操作，这也是在多线程场景时减小锁的粒度从而降低锁竞争的一种方案，并且代码中大多共享变量使用了volatile关键字声明，目的是第一时间获取修改的内容，性能非常好。

+++Copy-On-Write容器（读多写少）

简称COW，是一种用于程序设计中的优化策略。

JDK里的cow容器有两种：CopyOnWriteArrayList和CopyOnWriteArraySet，COW容器非常有用，可以在非常多的并发场景中使用到。

Cow容器即写时复制的容器，通俗的理解是当我们往一个容器添加元素的时候，不直接往当前容器添加，而是现将当前容器进行Copy，复制出一个新的容器，然后新的容器添加元素，添加完元素之后，再讲原容器的引用指向新的容器。这样做的好处是我们可以对COW容器进行并发的读，而不需要加锁，因为当前容器不会添加任何元素，所以cow容器也是一种读写分离的思想，读和写不同的容器

