遍历删除List中的元素有很多种方法，当运用不当的时候就会产生问题。下面主要看看以下几种遍历删除List中元素的形式：

1. 通过增强的for循环删除符合条件的多个元素；
2. 通过增强的for循环删除符合条件的一个元素；
3. 通过普通的for删除符合条件的多个元素；
4. 通过Iterator进行遍历删除符合条件的多个元素。

**简介：**

**fail-fast机制是java集合(Collection)中的一种错误机制。当多个线程对同一个集合的内容进行操作时，就可能会产生fail-fast事件。**

例如：当某一个线程A通过iterator去遍历某集合的过程中，若该集合的内容被其他线程所改变了；那么线程A访问集合时，就会抛出ConcurrentModificationException异常，产生fail-fast事件。

要了解fail-fast机制，我们首先要对ConcurrentModificationException异常有所了解。当方法检测到对象的并发修改，但不允许这种修改时就抛出该异常。同时需要注意的是，该异常不会始终指出对象已经由不同线程并发修改，如果单线程违反了规则，同样也有可能会抛出改异常。

诚然，迭代器的快速失败行为无法得到保证，它不能保证一定会出现该错误，但是快速失败操作会尽最大努力抛出ConcurrentModificationException异常，所以因此，为提高此类操作的正确性而编写一个依赖于此异常的程序是错误的做法，正确做法是：ConcurrentModificationException应该仅用于检测bug。

当使用fail-fast iterator对Collection或Map进行迭代操作过程中尝试直接修改Collection/Map的内容时，即使是在单线程下运行，java.util.ConcurrentModificationException异常也将被抛出。

Iterator是工作在一个独立的线程中，并且拥有一个mutex锁。Iterator被创建之后会建立一个指向原来对象的单链索引表，当原来的对象数量发生变化时，这个索引表的内容不会同步改变，所以当索引指针往后移动的时候就找不到要迭代的对象，所以按照fail-fast原则Iterator会马上抛出java.util.ConcurrentModificationException异常。

所以Iterator在工作的时候是不允许被迭代的对象被改变的。但你可以使用Iterator本身的方法remove()来删除对象，Iterator.remove()方法会在删除当前迭代对象的同时维护索引的一致性。

有意思的是如果你的Collection/Map对象实际只有一个元素的时候，ConcurrentModificationException异常并不会被抛出。这也就是为什么在javadoc里面指出：itwould be wrong to write a program that depended on this exception forits correctness:ConcurrentModificationException should be used only todetect bugs.

附：来自ibm developerworks上对java.util.concurrent包的说明片段：java.util包中的集合类都返回fail-fast迭代器，这意味着它们假设线程在集合内容中进行迭代时，集合不会更改它的内容。如果fail-fast迭代器检测到在迭代过程中进行了更改操作，那么它会抛出ConcurrentModificationException，这是不可控异常。在迭代过程中不更改集合的要求通常会对许多并发应用程序造成不便。相反，比较好的是它允许并发修改并确保迭代器只要进行合理操作，就可以提供集合的一致视图，如java.util.concurrent集合类中的迭代器所做的那样。java.util.concurrent集合返回的迭代器称为弱一致的（weakly consistent）迭代器。对于这些类，如果元素自从迭代开始已经删除，且尚未由next()方法返回，那么它将不返回到调用者。如果元素自迭代开始已经添加，那么它可能返回调用者，也可能不返回。在一次迭代中，无论如何更改底层集合，元素不会被返回两次。