**fail-fast机制**

**fail-fast：**它是Java集合(Collection)的一种错误机制。**当多个线程对同一个集合进行修改结构操作，使用集合的迭代器iterator,会首先检测是否有对集合的并发修改，进而产生ConcurrentModificationException异常提示。**

阅读ArrayList的add方法源码，可以注意到在add方法中，除了校验底层数组的容量是否需要扩充之外，恒久对modCount这个变量进行了自增操作。（除了add方法存在外，remove等几乎其他所有影响集合长度的方法都存在。）

在ArrayList的迭代器实现中，所有方法**首先校验了集合的modCount和迭代器的expectedModCount(**集合创建迭代器时**copy自集合的modCount**)是否一致，**二者值不一致**则表示迭代器的集合对象在创建当前迭代器后进行了修改集合结构操作，**则迭代器抛出异常ConcurrentModificationException。**

因此，对集合进行遍历操作，**推荐使用迭代器遍历，更加安全可靠（fail-fast机制提供的保证！）。**同时在遍历集合的过程中，需要对集合进行修改，**应使用迭代器**提供的**remove等方法安全的修改集合，不会造成ConcurrentModificationException异常。**

解决方案：

CopyOnWriteArrayList替代。或者用迭代器内部方法去改变数据结构。

## CopyOnWriteArrayList分析

* 修改代价大，可以从源码知道，remove还是add方法，都会进行一次数组的复制，这样消耗了空间（可能导致gc的频率提高）也消耗了时间
* 读写分离，读写不一致，读的时候读的是旧的数组，写的时候写的是新的数组，所以读的时候不一定是最新的
* 读的时候不需要进行加锁，因为写的时候是写在新的数组，读的数组是旧的数组，并不会改变
* 因此，CopyOnWriteArrayList适合读多写少的场景

fail-safe机制

fail-safe：**保证在对任何集合结构的修改操作**都基于**复制 - 修改 进**行的，即先copy一个新的集合对象，**然后对新的集合对象进行修改，最后将新的集合对象替换掉老的集合对象**(老的集合对象的地址指向新的集合对象)。

java.util.concurrent包下采用的是fail-safe机制。

缺陷：**1.对集合的复制copy会产生大量的对象，造成内存空间的浪费。**

**2.无法保证集合迭代过程中获取的集合数据是最新的内容。**

6. fail-fast和 fail-safe 的区别

|  |  |  |
| --- | --- | --- |
|  | **Fail Fast Iterator** | **Fail Safe Iterator** |
| Throw ConcurrentModification Exception | Yes | No |
| Clone object | No | Yes |
| Memory Overhead | No | Yes |
| Examples | **HashMap,Vector,ArrayList,HashSet** | **CopyOnWriteArrayList, ConcurrentHashMap** |