1. ArrayList线程不安全
   1. 根据ArrayList源码分析可以知道，ArrayList是一个线程不安全的类；
   2. 我们分析下列代码:

|  |
| --- |
| package cn.gsdx.JUC;  import java.util.ArrayList;  import java.util.List;  import java.util.UUID;  public class ThreadList {  public static void main(String[] args) {  List<String> list = new ArrayList();  for (int i = 0; i < 30; i++) {  new Thread(()->{  for (int j = 0; j < 3; j++) {  list.add(UUID.randomUUID().toString().substring(0,8));  System.out.println(list);  }  }).start();  }  }  } |

运行代码我们发现报了如下错误（在高并发的情况下）

java.util.ConcurrentModificationException(并发修改异常)

解决方案:

1. 使用Vector; 线程安全，所有的方法加锁，且加锁在方法上，效率最低（方法内部存在逻辑处理）
2. 使用Collections.synchronizedList(new List) ：读写删除加锁，且加锁在代码块上，效率较好，遍历未加锁，可以根据实际业务需求，自行决定是否加锁
3. CopyOnWriteArrayList();读未加锁，写使用CAS自旋锁，先复制原数组，修改复制的数组，之后把复制数组重新复制给原地址（当数组过大时，复制效率极低），并发过多，CAS碰撞过多，也会影响性能，读取数据是原数组，所以如果add方法还未执行到setArray方法，读取的数据就是原来的数据。

总结:写时复制

Copyonwrite容器即写时复制的容器。往一个容器添加元素的时候，不直接往当前容器object[]添加，而是先将当前容器object[]进行copy,复制出一个新的容器object[ ] newELements，然后新的容器object[ ] newELements里添加元素，添加完元素之后，

再将原容器的引用指向新的容器setArray(newELements);。这样做的好处是可以对Copyonwrite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copyonwrite容器也是一种读写分离的思想，读和写不同的容器