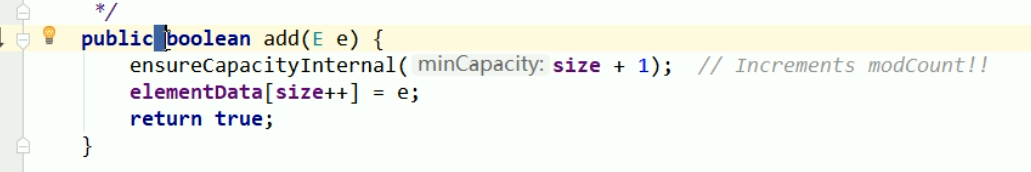
集合不安全arraylist

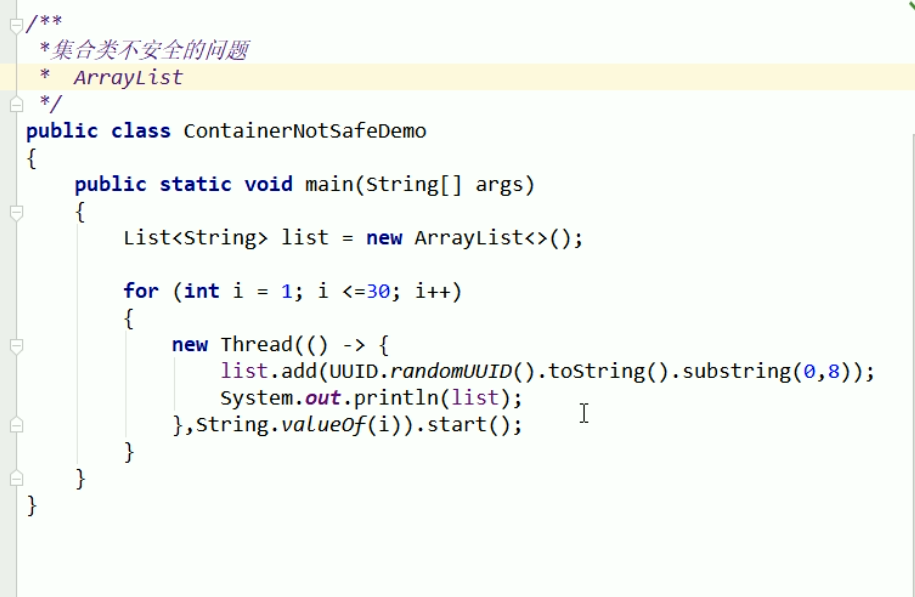
在new ArrayList()时括号里没有指定参数的话，则ArrayList这时的数组容量为0，只有在第一次添加元素时，才会把数组容量设为1

为什么arraylist有线程安全问题?

Arraylist的add方法为了保证并发性的效率没有加锁



这个代码会导致下面的报错,当i<=3时不报错,当为30会报错



故障原因



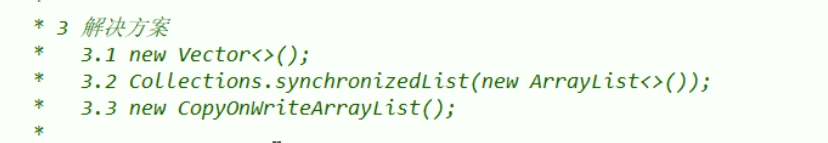
Arraylist在多线程高并发下常见异常(并发修改异常) 可以在常见异常里面说

导致方案

并发争抢修改方案

一个线程在修改,另一个线程过来争抢,导致数据不一致,并发修改异常

解决方案



为什么CopeOnWriteArrayList 可以线程安全----写时复制,读写分离思想

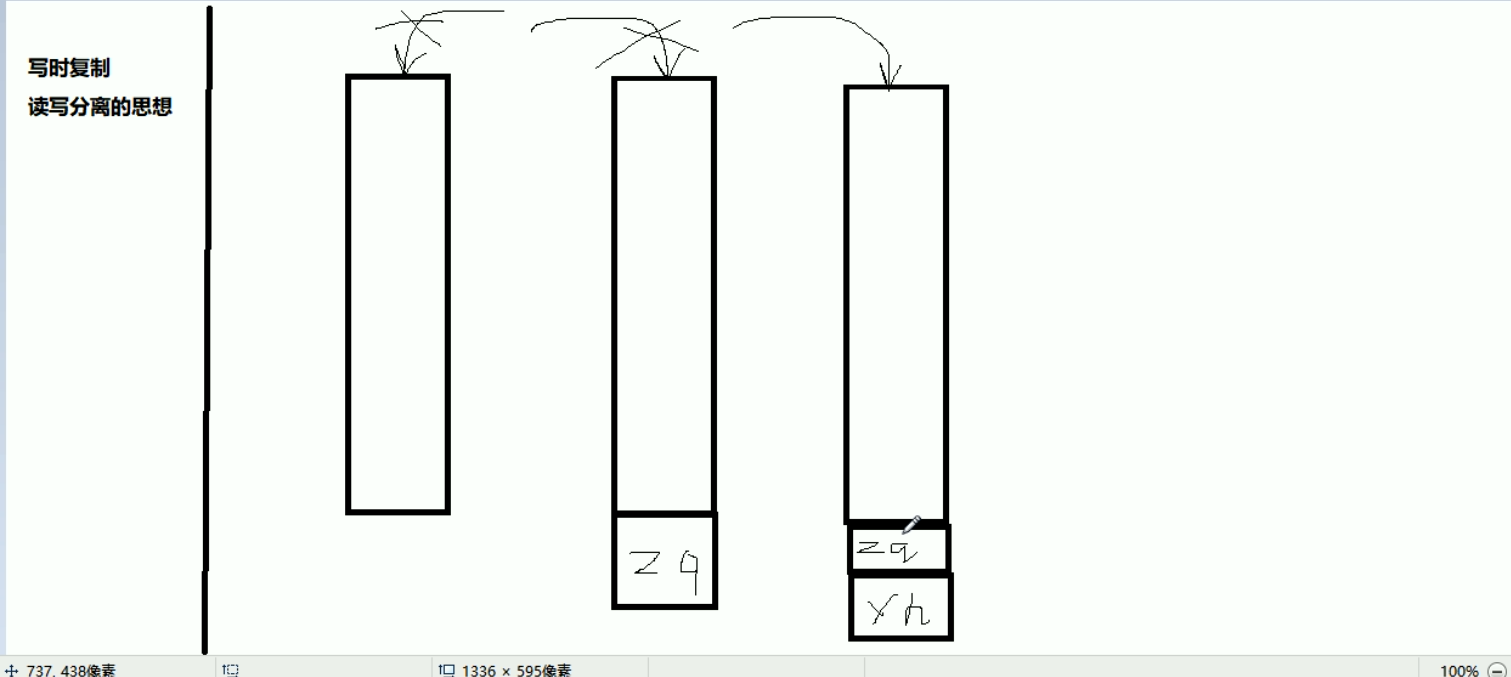
他的底层数组通过volative修饰

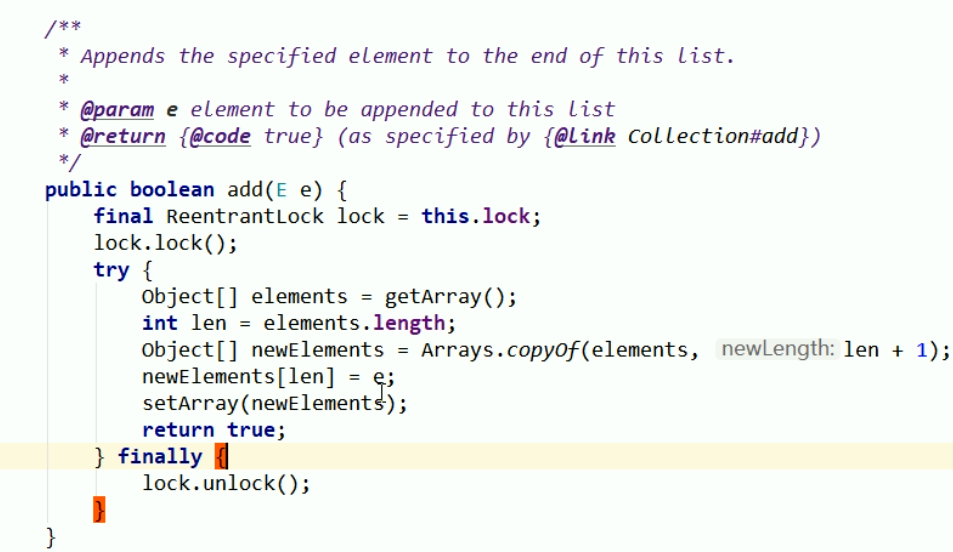
add()方法是这样写的,添加ReentrantLock锁,add方法的底层如下

写时复制思想

CopeOnWrite容器即写时复制的容器,往一个一个容器添加元素时候,不直接王当前容器object[]添加,而是先将当前容器object[]进行copy , 复制一个新的object[] newElements ,然后新的容器object[] newElements 里添加数据,添加完元素之后,再将原容器的引用指向新容器,setArray(newElements);这样做的好处是可以对对CopyOnWrite容器进行并发的读,而不用加锁,,因为容器不会添加任何元素,所以CopyOnWritr容器也是一种读写分离的思想,读和写在不同的容器

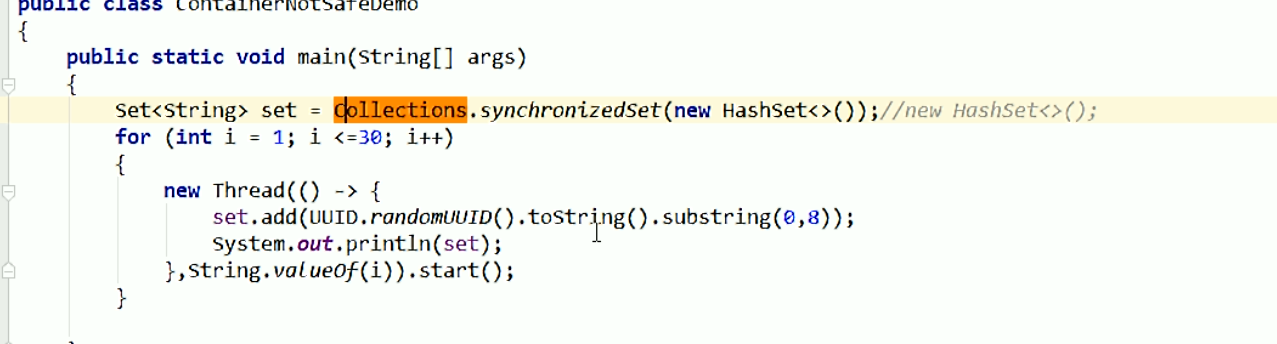
线程准备添加数据时,先把共享的主数据拷贝下来,直接添加数据到最后面,扩容,然后将引用指向自己copy过来的的这个arraylist





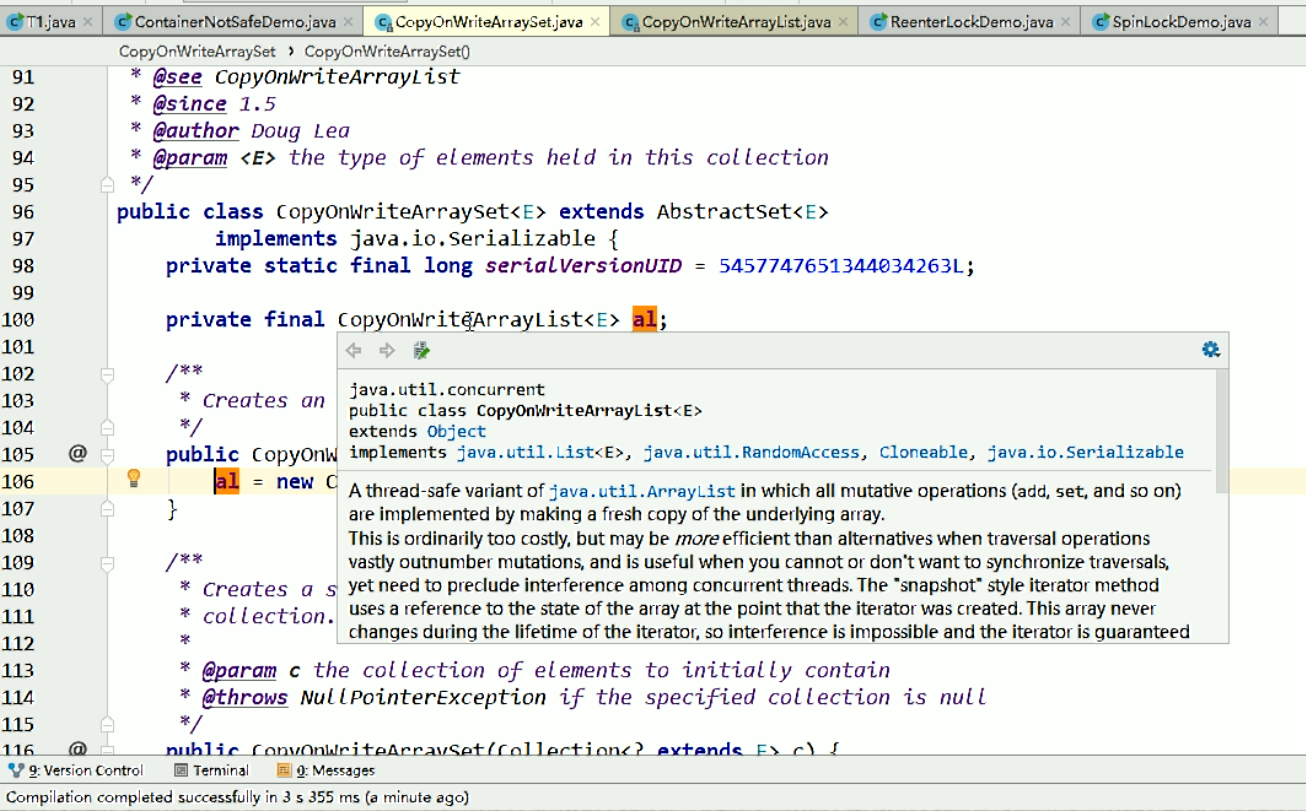
集合不安全set

可以用Clooections.synchronizedSet解决



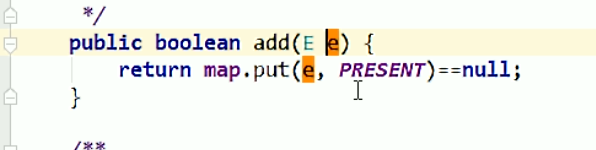
第二种可以通过CopyOnWriteArraySet

底层是CopyOnWriteArrayList



Hashset底层为hashmap,

add()的底层为hashmap的put方法,hashset存的值到put方法的键中,值present不为空,是一个object类型的常量





集合安全之map

Hashmap线程不安全

解决办法:

ConcurrentHashMap ???分段锁,,自己看