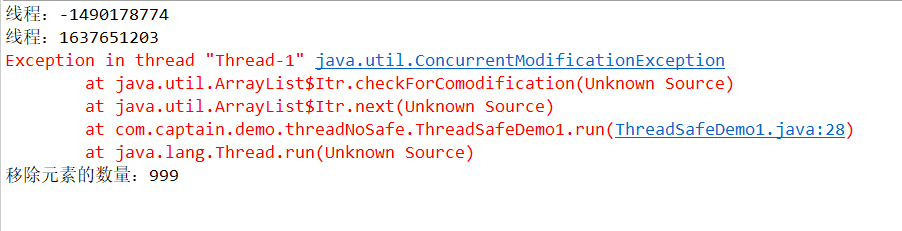
## 上机一

## 第一个演示示例

### 问题

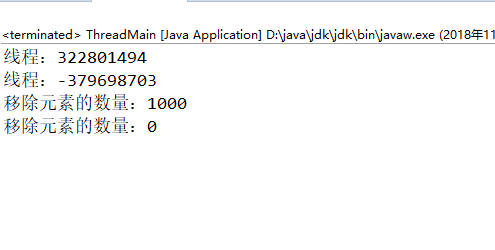
其他线程进行了增加了和删除操作，必然导致Iterator 内部的修改次数和List的不一致，从而引发同步异常，从而报出java.util.ConcurrentModificationException的错误。



### 解决方法

synchronized(this){}

改为

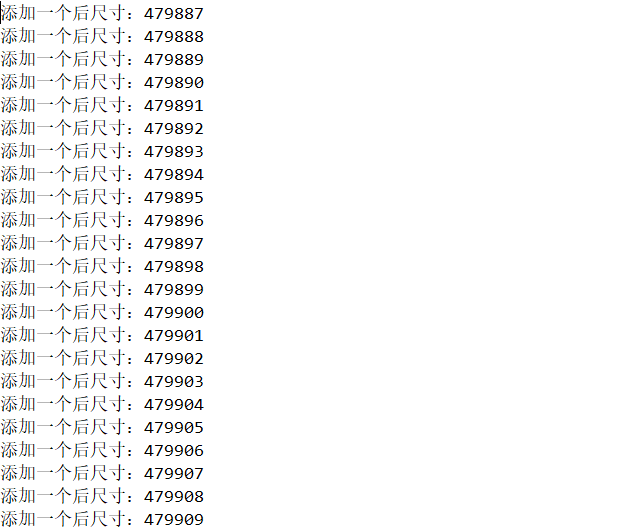
synchronized(List){}

由于数据量比较小所以并没有出现不同步的情况，为了线程安全，我还是将List改为了Vector。

## 第二个演示示例

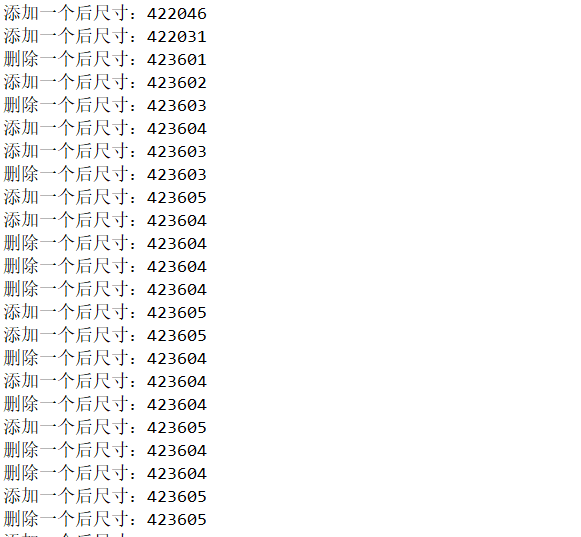
### 问题：

ThreadSafeDemo2和ThreadSafeDemo3均为死循环和线程无法同步

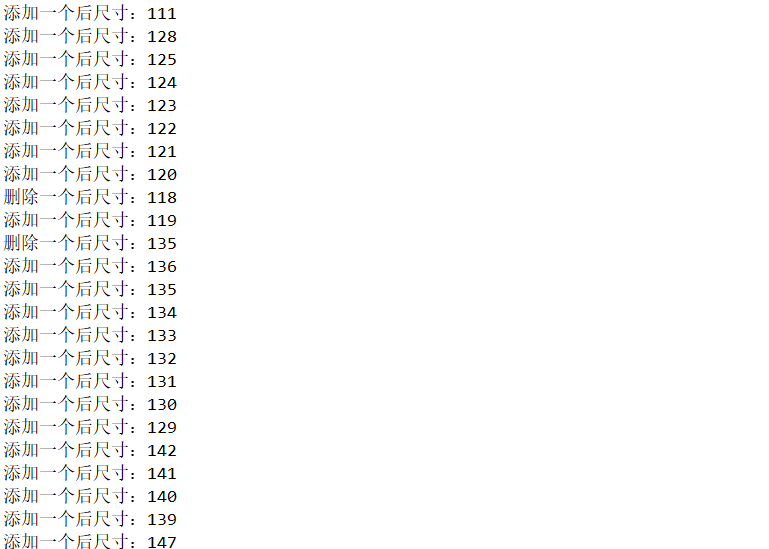


### 解决方法：

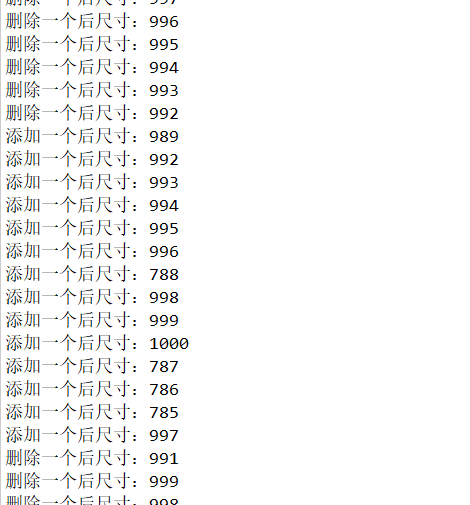
先解决掉不同步的问题，List在数据量小的时候是可以同步的，但数据量大了就不行了，所以现将List改为了Vector,是可以做到同步的。



为了让他不一直添加和删除，将ThreadSafeDemo2中的While(true)改为了while(vector.size()<1000)，将ThreadSafeDemo3中的While(true)改为了while(vector.size()>0)。



然后发现若在改变循环条件后，数据量小了，将类型改回List是可以做到同步的。



# 源代码

## ThreadMain：

**public** **class** **ThreadMain** {

**public** **static** **void** **main**(**String**[] args) {

//第一个的演示示例

**Vector** <**String**> vector = **new** **Vector** <>(1000);

**for**(**int** cnt=0;cnt<1000;cnt++){

vector.add("abc");

}

**ThreadSafeDemo1** demo1 = **new** ThreadSafeDemo1(vector);

**ThreadSafeDemo1** demo2 = **new** ThreadSafeDemo1(vector);

**Thread** thread1 = **new** Thread(demo1);

**Thread** thread2 = **new** Thread(demo2);

thread1.start();

thread2.start();

//第二个的演示示例

/\*Vector<String> vector = new Vector<>();

ThreadSafeDemo2 demo1 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo2 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo3 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo4 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo5 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo6 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo7 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo8 = new ThreadSafeDemo2(vector);

ThreadSafeDemo2 demo9 = new ThreadSafeDemo2(vector);

ThreadSafeDemo3 demo10 = new ThreadSafeDemo3(vector);

new Thread(demo1).start();

new Thread(demo2).start();

new Thread(demo3).start();

new Thread(demo4).start();

new Thread(demo5).start();

new Thread(demo6).start();

new Thread(demo7).start();

new Thread(demo8).start();

new Thread(demo9).start();

new Thread(demo10).start();\*/

}

}

## ThreadSafeDemo1：

**public** **class** **ThreadSafeDemo1** **implements** Runnable{

**private** **Vector**<**String**> vector;

**public** **ThreadSafeDemo1**(**Vector**<**String**> vector){

**this**.vector = vector;

}

@Override

**public** **void** **run**() {

**Random** random = **new** Random();

**System**.***out***.println("线程："+ random.nextInt());

**synchronized**(vector){

Iterator<**String**> iterator = vector.iterator();

**int** cnt = 0;

**while**(iterator.hasNext()){

iterator.next();

iterator.remove();

cnt++;

**try** {

TimeUnit.***MILLISECONDS***.sleep(10);

} **catch** (**InterruptedException** e) {

e.printStackTrace();

}

}

**System**.***out***.println("移除元素的数量："+cnt);

}

}

}

## ThreadSafeDemo2：

**public** **class** **ThreadSafeDemo2** **implements** Runnable{

**private** **Vector**<**String**> vector;

**public** **ThreadSafeDemo2**(**Vector**<**String**> vector){

**this**.vector = vector;

}

@Override

**public** **void** **run**() {

**while**(vector.size()<1000){

vector.add("版权归作者所有，任何形式转载请联系作者。作者：KY主创们（来自豆瓣来源：https://www.douban.com/note/630084526/不过，专门研究“人猫交流（cat-human communication)”的学者认为这种印象也不完全是精确的——是的，这是一个真实存在的研究学科。研究宠物行为的学科曾经一度被取消，因为人们认为这种研究的本质是一种拟人论（anthropomorphism），但现在研究这个学科的学者已经不再这样认为了（即宠物行为研究有其自身本质的意义）。");

**System**.***out***.println("添加一个后尺寸："+vector.size());

}

}

}

## ThreadSafeDemo3：

**public** **class** **ThreadSafeDemo3** **implements** Runnable{

**private** **Vector**<**String**> vector;

**public** **ThreadSafeDemo3**(**Vector**<**String**> vector){

**this**.vector = vector;

}

@Override

**public** **void** **run**() {

**while**(vector.size()>0){

//if(list.size()>0){

vector.remove(0);

**System**.***out***.println("删除一个后尺寸："+vector.size());

// }

}

}

}