链滴

使用 Iterator 迭代器发生 ConcurrentModificationException 异常分析

作者：sirwsl

原文链接：https://ld246.com/article/1631001629749

来源网站：链滴

故事

为了在CSDN的问答社区获取几个基本，发现写的太多都可以作为一篇博文了，就顺道发了。
为了那几个原力分我容易吗？我容易吗？你还在这看我这博文，你忍心不点赞吗！！！

提问内容

LinkedList中，利用iterator遍历，究竟是哪里出错了呢

java

欲解决的问题：
在LinkedList中执行以下操作：
假如原LinkedList存在元素a，则再向其中添加元素a+1，执行3个循环。

```
LinkedList<Integer> li = new LinkedList<Integer>();
li.add(0);
for(int i = 1; i < 3; ++i){
    int size = li.size();
    Iterator<Integer> itr = li.iterator();
    for(int j = 0; j < size; ++j){
        li.add(itr.next()+1);
    }
}
```

但是第二个循环就开始报错了，
报错的位置应该是itr.next()
但是我debug观察，在报错之前，itr的next值是没问题的。
报错信息：

```
Exception in thread "main" java.util.ConcurrentModificationException
at java.base/java.util.LinkedList$ListItr.checkForComodification(LinkedList.java:970)
at java.base/java.util.LinkedList$ListItr.next(LinkedList.java:892)
```

题目

初始化一个LinkedList链表，然后采用迭代器向其添加元素

```
LinkedList<Integer> li = new LinkedList<Integer>();
li.add(0);
for(int i = 1; i < 3; ++i){
    int size = li.size();
    Iterator<Integer> itr = li.iterator();
    for(int j = 0; j < size; ++j){
        li.add(itr.next()+1);
    }
}
```

报错

在采用迭代器进行元素添加元素的时候出现了并发报错

```
Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 0.117 sec <<< FAILURE!
test1(MainTest)  Time elapsed: 0.015 sec  <<< ERROR!
java.util.ConcurrentModificationException
    at java.util.LinkedList$ListItr.checkForComodification(LinkedList.java:966)
    at java.util.LinkedList$ListItr.next(LinkedList.java:888)
    at MainTest.test1(MainTest.java:17)
```

如何解决

将

```
Iterator<Integer> itr = li.iterator();
...
li.add("xxx");
```

改用

```
ListIterator<Integer> itr = li.listIterator();
...
itr.add("xxx");
```

添加时候完全交给迭代器进行完成

正确代码

```
LinkedList<Integer> li = new LinkedList<Integer>();
li.add(0);
for(int i = 1; i < 3; ++i){
    int size = li.size();
    ListIterator<Integer> itr = li.listIterator();
    for(int j = 0; j < size; ++j){
        itr.add(itr.next()+1);
    }
}
li.forEach(System.out::println);
```

问题分析

1、先看报错


```
Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 0.117 sec <<< FAILURE!
test1(MainTest)  Time elapsed: 0.015 sec  <<< ERROR!
java.util.ConcurrentModificationException
    at java.util.LinkedList$ListItr.checkForComodification(LinkedList.java:966)
    at java.util.LinkedList$ListItr.next(LinkedList.java:888)
    at MainTest.test1(MainTest.java:17)
```

```
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
3)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:5
)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:56)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
    at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
    at org.junit.runners.BlockJUnit4ClassRunner$1.evaluate(BlockJUnit4ClassRunner.java:100)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:366)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:103)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:63)
    at org.junit.runners.ParentRunner$4.run(ParentRunner.java:331)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:79)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:329)
    at org.junit.runners.ParentRunner.access$100(ParentRunner.java:66)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:293)
    at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:413)
    at org.apache.maven.surefire.junit4.JUnit4Provider.execute(JUnit4Provider.java:252)
    at org.apache.maven.surefire.junit4.JUnit4Provider.executeTestSet(JUnit4Provider.java:141)
    at org.apache.maven.surefire.junit4.JUnit4Provider.invoke(JUnit4Provider.java:112)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
3)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.apache.maven.surefire.util.ReflectionUtils.invokeMethodWithArray(ReflectionUtils.jav
:189)
    at org.apache.maven.surefire.booter.ProviderFactory$ProviderProxy.invoke(ProviderFactory.
ava:165)
    at org.apache.maven.surefire.booter.ProviderFactory.invokeProvider(ProviderFactory.java:85

    at org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:1
5)
    at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:75)
```

2、找到对我们有用的信息

```
java.util.ConcurrentModificationException
    at java.util.LinkedList$ListItr.checkForComodification(LinkedList.java:966)
    at java.util.LinkedList$ListItr.next(LinkedList.java:888)
    at MainTest.test1(MainTest.java:17)
```

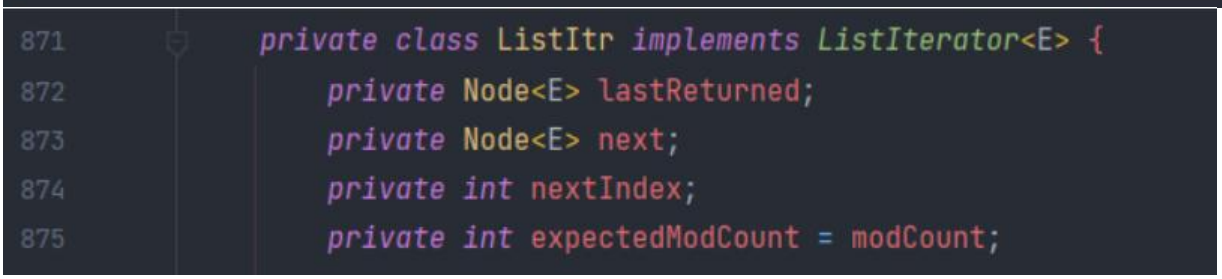
可以看到，报错因为我们执行到 li.add(itr.next()+1);时候报错

为什么报错是因为在LinkedList中的子类ListItr的next方法中888行，next方法又调用了checkForCo odification方法报错，所以我们去看一下LinkedList的源码分别找到888和966行

分别如下

从图中可以看出来，结果了，因为**modCount 与expectedModCount不相等，所以就抛出了异常**
那为什么不相等呢？我们看看modCount 和expectedModCount究竟是啥。

modcount大概的作用呢就是用作统计被修改的次数，也就是当你的整体的结构发生修改，eg：add remove的时候他就会++ 或者--，从而就会造成正在迭代的数据出现不正确的结果，如果该字段发改变就会抛出ConcurrentModificationException异常



now，你**maybe**大概**明白**了，**因为你在进行add的时候使得modCount发生了改变。**

有意思的事

你或许会发现当你在执行第一次循环（i=1）的时候程序是正常执行的，但是当你执行第二次循环的候报错了，为什么呢?

第一次循环（i=1）

第一次（i=1）的时候，li.size()的大小是1，里面只有一个值0，所以在进行循环

```
Iterator<Integer> itr = li.iterator();
for(int j = 0; j < size; ++j){
    li.add(itr.next()+1);
}
```

的时候，你只是执行了一次，也就是说：

你在li.add()这个方法执行了一次就退出了j循环，当你调用itr.next()+1的时候，他的modCount和expectedModCount的值是一样的,所以能够正常返回，当你执行结束li.add的时候modCount值发生modCount++的操作，但是这个时候你已经退出循环了，无所谓。

第二次循环（i=2）

1)这个时候你需要仔细分析一下：

1：li 中的元素现在有0、1，

2：li.size()的值是2,也就是会进行2次 J循环

2)重头戏:

当j=0的时候进入了**第一次循环这个时候执行li.add(itr.next()+1)的时候，你说有没有出现异常，答是没有!**

因为这个过程就行第一次循环的那一部分阐述一样，当**他第一次执行itr.next()的时候modCount值相等的**，所以能够正常返回数值，而**执行到li.add()的时候li发生了add操作导致modCount++**，然这个时候执行完这一行他就继续下一次循环。

当j = 1的时候，这个时候他要**执行itr.next()获取值，就会去LinkedList中的next方法中去，在获取时候会先执行checkForComodification这个方法去校验modCount和expectedModCount是否等，如果不相等就说明发生了改变就会抛出ConcurrentModificationException异常。**

也就是说当你j=1的时候执行itr.next发现了modCount值变化了，所以就抛出了异常。