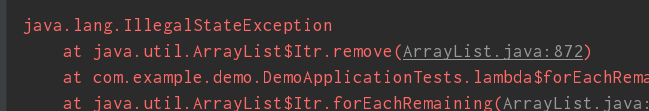
# Iterator.ForEachRemaining 方法：

该方法注意：

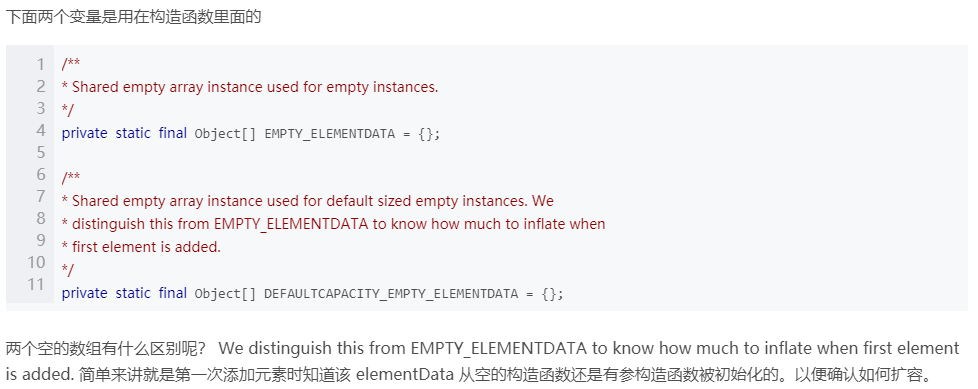
在LinkedList类中，可以利用该方法来进行删除：

List list = new LinkedList<>();  
for (int i = 0; i < 5; i++) {  
 list.add(i);  
}  
  
Iterator iterator = list.iterator();  
iterator.forEachRemaining((o) -> {  
 if (o instanceof Integer) {  
 if ((int)o == 2) {  
 iterator.remove();  
 }  
 }  
});  
System.*out*.println(list.size());

此时输出size为4。但是在ArrayList中，调用该方法删除元素的时候，会报异常：

## ArrayList:

底层基于数组实现容量大小动态变化的，数组初始长度为10，内有变量**modCount**用来记录对List操作的次数，防止在迭代过程中数组被修改。



public ArrayList() {

this.elementData = DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA;

}

无参构造函数，一开始只是给了底层的数组一个空的数组，在第一次添加元素的时候扩容到10的。

add(Object o)的时候，每次都会判断容量是否够用，不够用就扩容1.5倍，还不够用就使用size+1的容量；扩容函数：器重minCapacity是this.size++（size在add中一开始就会自增）。

private void grow(int minCapacity) {  
 // overflow-conscious code  
 int oldCapacity = elementData.length;  
 int newCapacity = oldCapacity + (oldCapacity >> 1);  
 if (newCapacity - minCapacity < 0)  
 newCapacity = minCapacity;  
 if (newCapacity - MAX\_ARRAY\_SIZE > 0)  
 newCapacity = hugeCapacity(minCapacity);  
 // minCapacity is usually close to size, so this is a win:  
 elementData = Arrays.*copyOf*(elementData, newCapacity);  
}

##### Iterator迭代器

内部主要有3个变量：

cursor：代表下一个要访问的元素下标。

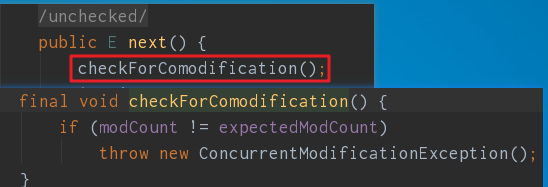
lastRet：代表上一个要访问的元素下标。

expectedModCount：代表对 ArrayList 修改次数的期望值，初始值为 modCount。

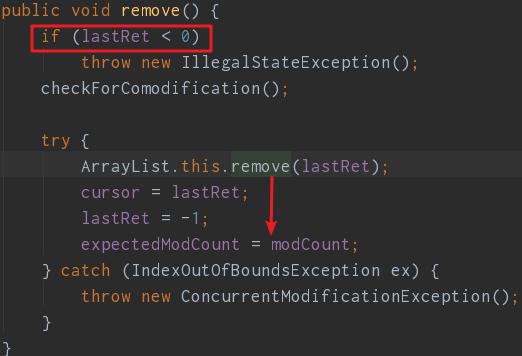
在调用remove删除list元素的时候，调用的是list的remove方法。会导致modCount++，然后下一次循环，

for(Iterator iterator = list.iterator(); iterator.hasNext(); i = (Integer) iterator.next())

的时候，调用hasNext()判断有无下一个，返回cursor != size,有则会调用next()，然后首先检查modCount数量和expectedModCount的数量（expectedModCount默认等于modCount）。此时就会报错。



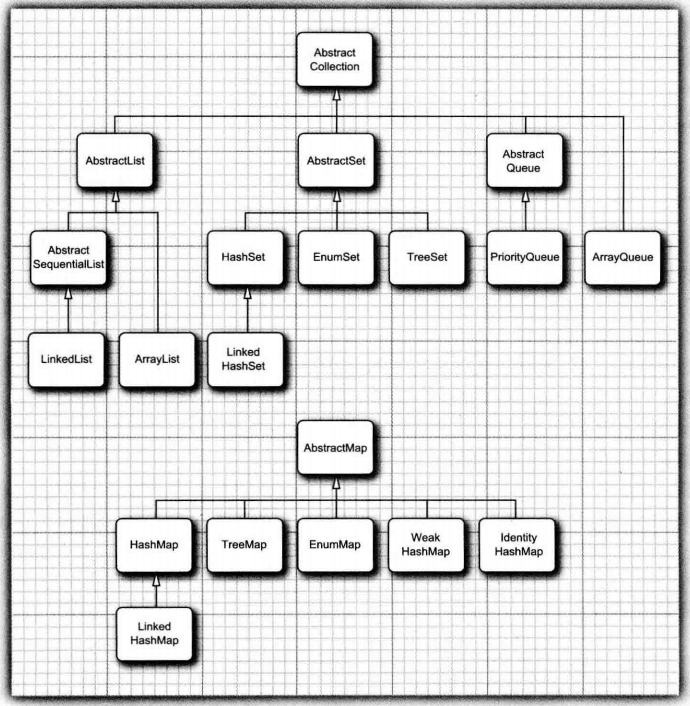
而在迭代器中，使用iterator的remove来删除对象，此时的remove方法是迭代器的remove方法，



可以看到，首先判断lastRet的值，默认为-1，再调用next（）之后会赋值，代表上一个访问的元素的下表，这里进行判断约束了在remove之前必须进行next操作。然后调用list的remove方法，注意到调用了之后会对expectedModCount进行同步，因此后面判断的时候就不会报错了。

以上的代码实现了再迭代器中可以remove而for循环中不可以remove的功能。

# Collection



AbstractCollection实现了Collection接口

remove操作将会移除List中第一个出现的指定元素，add和addAll操作总是会将新的元素添加到原有List末尾。

## 实现类：

public class LinkedList<E>  
 extends AbstractSequentialList<E>  
 implements List<E>, Deque<E>, Cloneable, java.io.Serializable

实现了List，Deque（双端队列）接口，继承了AbstractSequentialList->实现了Collection接口，底层是双向链表的结构（Node节点）：

E item;  
Node<E> next;  
Node<E> prev;

public class ArrayList<E> extends AbstractList<E>  
 implements List<E>, RandomAccess, Cloneable, java.io.Serializable

## Spliterator()方法的返回对象Spliterator

**boolean** **tryAdvance**(Consumer<? **super** T> action);

Spliterator<T> **trySplit**();

**long** **estimateSize**();

**void** **forEachRemaining**(Consumer<? **super** T> action)

共有四个方法，

第一个方法类似于迭代器的next方法，顺序处理每个元素（执行action的accept()方法）。

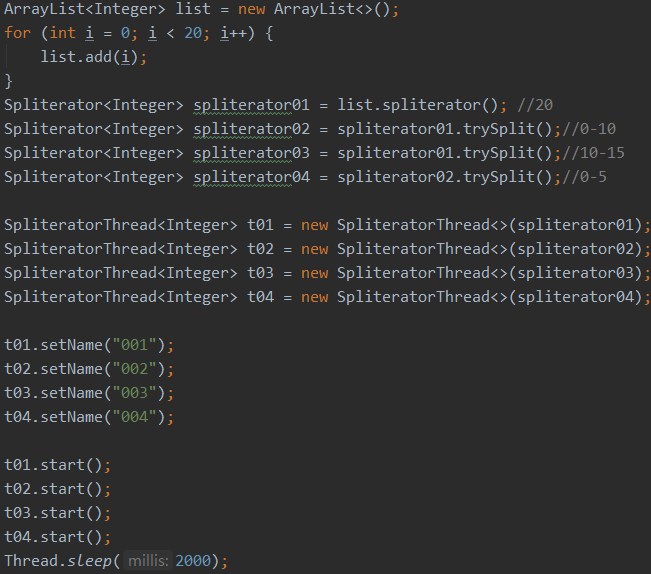
第二个方法则是尝试分割Collection的元素，返回一个新的Spliterator对象，两个Spliterator会并发执行，如果该集合小到无法分割，则返回null。

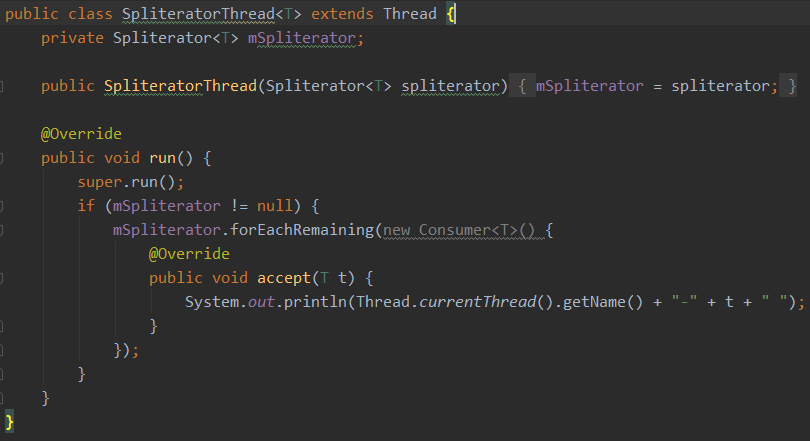
第三个方法用于估算集合还有多少个元素没有被遍历

第四个方法对Spliterator中的元素进行循环，调用action的accept()方法。

## ArrayList的spliterator：

获取到该集合的Spliterator对象，调用trySplit()将Spliterator对象分割出来一个新的Spliterator对象，将老的对象的左半边给新对象，这也是ArryaList的trySplit方法的实现：将集合的左半边元素赋给一个新的Spliterator对象。





输出：

001-15

004-0

004-1

004-2

004-3

003-10

002-5

002-6

002-7

002-8

003-11

004-4

001-16

003-12

002-9

003-13

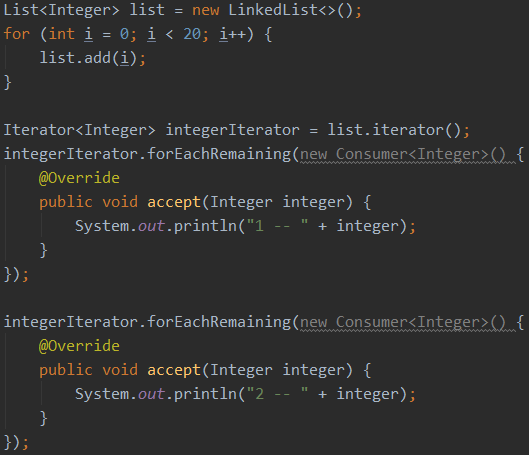
001-17

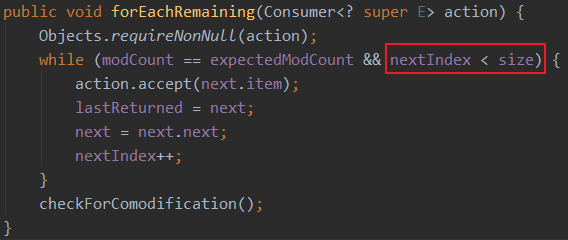
001-18

001-19

003-14

补充： Iterator中的forEachRemaining与forEach循环的区别：





nextIndex初始值为0，循环一次结束后，值为size，再次调用的时候就直接跳出循环了，因此forEachRemaining只能循环依次（**查询书，查看高深用法**）。

FoeEach原理:

for(Iterator iterator = list.iterator(); iterator.hasNext(); i = (T) iterator.next())

## LinkedList的Spliterator

只有t2和t4输出，网上资料较少，需要下来再花时间研究。