循环删除List集合元素的问题

# 遍历List集合的三种方式

JAVA中循环遍历list有三种方式**for循环、增强for循环**（即**foreach循环**）、**iterator遍历**。

## 普通的for循环

## foreach循环(增强的for循环)

## 迭代器Iterator

# 循环删除元素问题及比较分析

创建ArrayList：

List<String> list = new ArrayList<String>();  
list.add("AA");  
list.add("BBB");  
list.add("CCCC");  
list.add("DDDD");  
list.add("EEE");

## for循环索引删除

示例：删除长度为4的字符串元素。

for (int i = 0; i < list.size(); i++) {  
 if (list.get(i).length() == 4) {  
 list.remove(i);  
 }  
}

验证输出结果：

for (String s : list) System.*out*.print(s + ",");

输出结果为：AA,BBB,DDDD,EEE,

**错误之处**：**DDDD元素**竟然没有删除掉。

**问题分析**：

这种方式的问题在于，删除某个元素后，list的大小size发生了变化，**而你的索引也在变化**，**所以会导致你在遍历的时候漏掉某些元素**。比如当你删除第1个元素后，继续根据索引访问第2个元素时，**因为删除的关系后面的元素都往前移动了一位**，所以实际访问的是第3个元素。<**见源码分析**>；不会报出异常，只会出现漏删的情况；如果只是删除一个元素，就break，可以使用这种方式。

**适用场景**：

因此，**这种方式可以用在删除特定的一个元素时使用，但不适合循环删除多个元素时使用。**

**背景**：在京东金融时，案件审核系统的综合查询页面，需要根据案发原因筛选工单，由于select语句比较特殊，总会将一些案发原因为空的筛选出来，因此想到了，再获取到List之后，过滤一遍，一开始使用普通for循环索引方式，总会有一两个还是没有删除掉；由于对迭代器使用的不熟练，晚上10点多了，时间紧，就采用创建新List方式，将需要保留的元素add进新List中。

## foreach循环删除元素

删除**一个元素**之后，无论是否还有满足条件的元素，都必须跳出循环break，否则报出java.util.ConcurrentModificationException。

因此，也只能用于删除一个元素。

//删除元素后必须break跳出，否则报出异常  
for (String s : list) {  
 if (s.length() == 4) {  
 list.remove(s);  
 break;  
 }  
}  
for (String s : list)

System.*out*.print(s + ",");//AA,BBB,DDDD,EEE,

这种方式的问题在于，删除元素后继续循环会报错误信息ConcurrentModificationException，因为元素在使用的时候发生了并发的修改，导致异常抛出。但是删除完毕马上使用break跳出，则不会触发报错。

## 迭代器Iterator

使用迭代器删除元素完美。

//迭代器:完美  
Iterator<String> iterator = list.iterator();  
while(iterator.hasNext()){  
 if(iterator.next().length()==4){  
 **iterator.remove();**  
 }  
}  
for (String s : list)

System.*out*.print(s + ",");//AA,BBB,EEE,

这种方式可以正常的循环及删除。但要注意的是，使用iterator的remove方法，如果用list的remove方法同样会报上面提到的ConcurrentModificationException错误。

## 巧妙方法：新建一个List对象

**适合场景**：对一个List对象，需要删除大量元素，保留较少元素时，此时可以采用创建一个新的List对象，将需要保留的元素add进新的List对象，然后让旧引用指向新对象即可。

新建一个List对象，将需要保留的元素item添加到新List中，然后原来的引用指向新List即可。

// 4. 创建新对象  
ArrayList<String> newList = new ArrayList<String>();  
for (int i = 0; i < list.size(); i++) {  
 if (list.get(i).length() < 3) {//条件为需要保留元素的条件  
 newList.add(list.get(i));  
 }  
}  
list = newList;  
for (String s : list) System.*out*.print(s + ",");//AA,

# 源码分析

## ArrayList中的size()方法

内部有个size属性，直接返回该size属性。

public int size() {  
 checkForComodification();  
 return this.size;  
}

## remove(index)与remove(Object)方法

### **remove(index)**

原理：将**index**之后的元素向前移动1个位置；通过native方法-System.arraycopy实现。

并将size减一，并将原list最后一个元素引用置为null，便于**垃圾回收**GC。

public E remove(int index) {  
 rangeCheck(index);  
 modCount++;  
 E oldValue = elementData(index);  
 int numMoved = size - index - 1;  
 if (numMoved > 0)  
 System.*arraycopy*(elementData, index+1, elementData, index,numMoved);  
 elementData[--size] = null; // clear to let GC do its work  
 return oldValue;  
}

public static native void **arraycopy**(Object src, int srcPos,

Object dest, int destPos, int length);

### **remove(Object)**

内部有个for循环遍历，遍历一遍找到该元素的索引，然后再调用remove(index)方法删除。

因此，foreach删除元素性能肯定不如普通的for循环。

public boolean remove(Object o) {  
 if (o == null) {  
 for (int index = 0; index < size; index++)  
 if (elementData[index] == null) {  
 fastRemove(index);  
 return true;  
 }  
 } else {  
 for (int index = 0; index < size; index++)  
 if (o.equals(elementData[index])) {  
 fastRemove(index);  
 return true;  
 }  
 }  
 return false;  
}

## ArrayList的迭代器实现原理

**见迭代器原理**。

迭代器内部也是通过调用remove(index)方法实现的，只不是增加了cursor控制了索引，保证删除元素后，cursor不变。

## for循环删除与迭代器删除比较：

**for循环由于外部索引持续增长，造成漏删的可能；**

**迭代器删除：内部也是调用remove(index)方法，只不过是通过cursor控制了索引，在删除元素后cursor不变，不会造成漏删的情况。**

**与创建新ArrayList相比：创建新ArrayList会消耗更多的内存空间；在删除较多的情况下效率更高些。**