主要有以下3种方法：

* 使用Iterator的remove()方法
* 使用for循环正序遍历
* 使用for循环倒序遍历

### 1、不能使用foreach来删除List的元素

List<String> platformList = new ArrayList<>();

platformList.add("博客园");  
platformList.add("CSDN");  
platformList.add("掘金");  
  
for (String platform : platformList) {  
 if (platform.equals("博客园")) {  
 // 运行程序会抛出并发修改异常  
 platformList.remove(platform);  
 }  
}

运行抛出java.util.ConcurrentModificationException异常（并发修改异常）

生成的字节码：



foreach循环在实际执行时，其实使用的是Iterator，使用的核心方法是hasNext()和next()。

再看ArrayList类的Iterator是如何实现的。可以看出，调用next()方法获取下一个元素时，第一行代码就是调用了checkForComodification();，而该方法的核心逻辑就是比较modCount和expectedModCount这2个变量的值。

在上面的例子中，刚开始modCount和expectedModCount的值都为3，所以第1次获取元素"博客园"是没问题的，但是当执行完下面这行代码时：

platformList.remove(platform);

modCount的值被修改成了4。

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;  
}

private void fastRemove(int index) {  
 modCount++;  
 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  
}

所以在第2次获取元素时，modCount和expectModCount的值就不相等了，所以抛出了异常。

final void checkForComodification() {

if (modCount != expectedModCount)  
 throw new ConcurrentModificationException();  
}

### 2、使用Iterator的remove()方法

Iterator<String> iterator = platformList.iterator();

while (iterator.hasNext()) {  
 String platform = iterator.next();  
 if (platform.equals("博客园")) {  
 iterator.remove();  
 }  
}

输出结果为：

[CSDN, 掘金]

为什么使用Iterator.remove()就可以呢？

private class Itr implements Iterator<E> {

int cursor; // index of next element to return  
 int lastRet = -1;// index of last element returned; -1 if no such  
 int expectedModCount = modCount;  
  
 Itr() {}  
  
 public boolean hasNext() {  
 return cursor != size;  
 }  
  
 @SuppressWarnings("unchecked")  
 public E next() {  
 checkForComodification();  
 int i = cursor;  
 if (i >= size)  
 throw new NoSuchElementException();  
 Object[] elementData = ArrayList.this.elementData;  
 if (i >= elementData.length)  
 throw new ConcurrentModificationException();  
 cursor = i + 1;  
 return (E) elementData[lastRet = i];  
 }  
  
 public void remove() {  
 if (lastRet < 0)  
 throw new IllegalStateException();  
 checkForComodification();  
  
 try {  
 ArrayList.this.remove(lastRet);  
 cursor = lastRet;  
 lastRet = -1;  
 expectedModCount = modCount;  
 } catch (IndexOutOfBoundsException ex) {  
 throw new ConcurrentModificationException();  
 }  
 }

可以看出没删除一个元素，都会将modCount的值重新赋值给expectedModCount，这样两个变量就像等了，不会触发抛出异常。

### 3、使用for循环正序遍历

for (int i = 0; i < platformList.size(); i++) {

String item = platformList.get(i);  
  
 if (item.equals("博客园")) {  
 platformList.remove(i);  
 i = i - 1;  
 }  
}

这种实现方式比较好理解，通过数组的下标来删除，不过有个注意事项就是删除元素后，要修正下标的值：

i = i - 1;

为什么要修正下标的值呢？

因为刚开始元素下标是这样的：



第1次循环将元素“博客园”删除后，元素的下标变成了：



第2次循环时i的值为1，也就是取到了元素”掘金“，这样就导致元素"CSDN"被跳过检查了，所以删除完元素后，我们要修正下下标，这也是上面代码中i = i - 1;的用途。

### 4、使用for循环倒序遍历

for (int i = platformList.size() - 1; i >= 0; i--) {

String item = platformList.get(i);  
  
 if (item.equals("掘金")) {  
 platformList.remove(i);  
 }  
}

这种实现方式和使用for循环正序遍历类似，不过不用再修正下标，因为刚开始元素的下标是这样的：



第1次循环将元素“掘金”删除后，元素的下标变成了下面这样：



第2次循环时i的值为1，也就是取到了元素”CSDN“，不会导致跳过元素，所以不需要修正下标。