# Java遍历时删除List、Set、Map中的元素（源码分析） - Rainnnbow - 博客频道 - CSDN

在对List、Set、Map执行遍历删除或添加等改变集合个数的操作时，不能使用普通的while、for循环或增强for。会抛出ConcurrentModificationException异常或者没有达到删除的需求。在遍历时删除元素，需要使用迭代器的方式。

## ArrayList源码中说明的报异常原因：

 *

The iterators returned by this class's iterator and

 * listIterator methods are *fail-fast*: if the list is

 * structurally modified at any time after the iterator is created, in any way

 * except through the iterator's own remove or add methods,

 * the iterator will throw a {@link ConcurrentModificationException}.  Thus, in

 * the face of concurrent modification, the iterator fails quickly and cleanly,

 * rather than risking arbitrary, non-deterministic behavior at an undetermined

 * time in the future.

（翻译：通过类的iterator和listiterator方法获取到的迭代器是快速失败迭代器：如果list在迭代器生成之后发生了结构性的改变，迭代器将抛出ConcurrentModificationException，**但是当使用迭代器自己的remove或add方法时，不会抛出此异常。**也就是说，当面对并发修改时，迭代器快速失败，而不是冒在未来不确定的时间发生不确定的行为的危险。）

 *

 * Note that the fail-fast behavior of an iterator cannot be guaranteed

 * as it is, generally speaking, impossible to make any hard guarantees in the

 * presence of unsynchronized concurrent modification.  Fail-fast iterators

 * throw ConcurrentModificationException on a best-effort basis.

 * Therefore, it would be wrong to write a program that depended on this

 * exception for its correctness: *the fail-fast behavior of iterators*

 * should be used only to detect bugs.

（翻译：需要注意的是迭代器不保证快速失败行为一定发生，因为一般来说不可能对是否发生了不同步并发修改做任何硬性的保证。快速失败迭代器会尽最大努力抛出ConcurrentModificationException异常。因此，写一个通过是否出现这种异常来判断是否正确的程序是错误的。快速失败行为的正确用法是仅用于检测异常。）

## 代码示例：

[java]view plaincopy

```
publicclass CollectionRemoveDemo {
publicstaticvoid main(String[] args) {
        ListRemove();
        System.out.println("----------------------------------------------------------------------------------------------------");
        SetRemove();
        System.out.println("----------------------------------------------------------------------------------------------------");
        MapRemove();
    }
publicstaticvoid ListRemove(){
    List strList = new ArrayList();
    strList.add("aaaa");
    strList.add("bbbb");
    strList.add("cccc");
    strList.add("cccc");
    strList.add("dddd");
for(String str : strList){
        System.out.println(str);
    }
    System.out.println("init List size:" + strList.size());
    Iterator it = strList.iterator();
while(it.hasNext()){
        String str = it.next();
if(str.equals("cccc")){
            it.remove();
        }
    }
for(String str : strList){
        System.out.println(str);
    }
    System.out.println("removed List size:" + strList.size());
}
publicstaticvoid SetRemove(){
    Set strSet = new TreeSet();
    strSet.add("aaaa");
    strSet.add("bbbb");
    strSet.add("cccc");
    strSet.add("cccc");//重复的数据将不会再次插入
    strSet.add("dddd");
for(String str : strSet){
        System.out.println(str);
    }
    System.out.println("Init Set size:" + strSet.size());
    Iterator it = strSet.iterator();
while(it.hasNext()){
        String str = it.next();
if(str.equals("cccc")){
            it.remove();
        }
    }
for(String str : strSet){
        System.out.println(str);
    }
    System.out.println("removed Set size:" + strSet.size());
}
publicstaticvoid MapRemove(){
    Map strMap = new TreeMap();
    strMap.put("a", "aaaa");
    strMap.put("b", "bbbb");
    strMap.put("c", "cccc");
    strMap.put("d", "dddd");
for(String key : strMap.keySet()){
        System.out.println(key + " : "+ strMap.get(key));
    }
    System.out.println("Init Map size:"+ strMap.size());
    Iterator> it = strMap.entrySet().iterator();
while(it.hasNext()){
        Entry strEntry = it.next();
if(strEntry.getKey().equals("c")){
            it.remove();
        }
    }
for(String key : strMap.keySet()){
        System.out.println(key + " : "+ strMap.get(key));
    }
    System.out.println("removed Map size:"+ strMap.size());
  }
}
```

收藏到代码笔记