# 集合框架（二） 迭代器

作者：alex18595752445
原文链接：https://ld246.com/article/1583316237002
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

# 迭代器

不同 Java 集合(容器)的内部结构是不一样的，如果为每种容器都单独实现一种遍历方法十分麻烦，为简化遍历容器的操作，所以推出了 Java 迭代器(Iterator)

通过 Java 迭代器，我们可以用统一的方法实现对容器的遍历，极大地简化了操作。

# 迭代器接口

## 1.java.util.Iterator 接口

```
public interface Iterator<E> {
    //查询容器中是否存在下一个元素
    boolean hasNext();
    //返回下一个元素
    //注意：在一次循环中,只能用一次next(),重复调用会返回再下面的元素.
    E next();
    //把元素从容器中移除
    //注意：要用迭代器的remove()方法,不要用容器的remove()方法.否则会发生异常
    default void remove() {
        throw new UnsupportedOperationException("remove");
    }
    //1.8新增默认方法,以后再讲
    default void forEachRemaining(Consumer<? super E> action) {
        Objects.requireNonNull(action);
        while (hasNext())
            action.accept(next());
    }
}
```

## 2.java.util.Iterable

```
public interface Iterable<T> {
    //容器可以通过此方法,返回该容器的迭代器.
    Iterator<T> iterator();
    //1.8新增,先不讲
    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
    //1.8新增,先不讲
    default Spliterator<T> spliterator() {
        return Spliterators.spliteratorUnknownSize(iterator(), 0);
    }
}
```

**CollectionCollection<E> 接口继承了 Iterable<T> 接口，而 Iterable 接口含有 iterator()方法所以：**

**所有继承自 Collection<E> 的容器，都实现了 iterator()方法，从而通过此方法获得各自容器的迭器！**

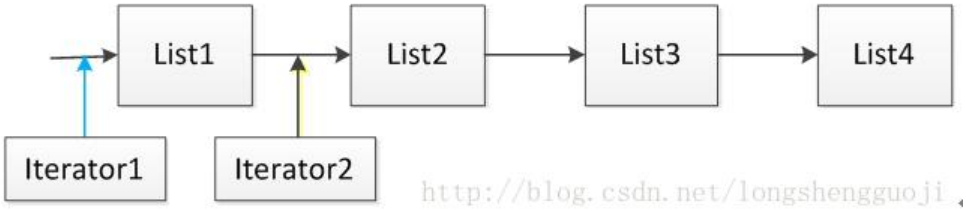
# 如何使用迭代器遍历集合

```
List<String> list =new ArrayList<String>();
//给list添加元素的过程省略.....
Iterator<String> listIterator = list.iterator();
while(listIterator.hasNext()) {
    //一次循环中只能调用一次next()方法!否则循环就乱套了!
    //因为每次调用next()都会获取下一个元素.
    String next = listIterator.next();
    if("bingo".equals(next)) {
    //只能通过迭代器删除元素,用集合的remove()方法会发生异常!
    listIterator.remove();
    //:x:这是错误方法:x:
    //list.remove(next);
    }else {
        System.out.println(next);
     }
}
System.out.println(list);
```

如果遍历的是 Map,那么用 Map 的 entrySet()方法获得 Set<Entry<K, V>>,通过 Set 来获取迭代器！Map.Entry 是一个键值对)

```
//Map通过entrySet()方法获取Set,再获取Set的迭代器
Iterator<Entry<Integer, String>> iterator =map.entrySet().iterator();
//之后的操作同上.
```

# ListIterator 和 Iterator 的原理




这里假设集合 List 由四个元素 List1、List2、List3 和 List4 组成，当使用语句 Iterator it = List.Iterator()时，迭代器 it 指向的位置是上图中 Iterator1 指向的位置，当执行语句 it.next()之后，迭代器指向位置后移到上图 Iterator2 所指向的位置。

首先看一下 Iterator 和 ListIterator 迭代器的方法有哪些。

Iterator 迭代器包含的方法有：

hasNext()：如果迭代器指向位置后面还有元素，则返回 true，否则返回 false

next()：返回集合中 Iterator 指向位置后面的元素

remove()：删除集合中 Iterator 指向位置后面的元素

ListIterator 迭代器包含的方法有：

add(E e): 将指定的元素插入列表，插入位置为迭代器当前位置之前

hasNext()：以正向遍历列表时，如果列表迭代器后面还有元素，则返回 true，否则返回 false

hasPrevious():如果以逆向遍历列表，列表迭代器前面还有元素，则返回 true，否则返回 false

next()：返回列表中 ListIterator 指向位置后面的元素

nextIndex():返回列表中 ListIterator 所需位置后面元素的索引

previous():返回列表中 ListIterator 指向位置前面的元素

previousIndex()：返回列表中 ListIterator 所需位置前面元素的索引

remove():从列表中删除 next()或 previous()返回的最后一个元素（有点拗口，意思就是对迭代器使用 hasNext()方法时，删除 ListIterator 指向位置后面的元素；当对迭代器使用 hasPrevious()方法时，除 ListIterator 指向位置前面的元素）

set(E e)：从列表中将 next()或 previous()返回的最后一个元素返回的最后一个元素更改为指定元素 e

## 相同点

都是迭代器，当需要对集合中元素进行遍历不需要干涉其遍历过程时，这两种迭代器都可以使用。

## 不同点

1.使用范围不同，Iterator 可以应用于所有的集合，Set、List 和 Map 和这些集合的子类型。而 ListIterator 只能用于 List 及其子类型。

2.ListIterator 有 add 方法，可以向 List 中添加对象，而 Iterator 不能。

3.ListIterator 和 Iterator 都有 hasNext()和 next()方法，可以实现顺序向后遍历，但是 ListIterator 有 hasPrevious()和 previous()方法，可以实现逆向（顺序向前）遍历。Iterator 不可以。

4.ListIterator 可以定位当前索引的位置，nextIndex()和 previousIndex()可以实现。Iterator 没有此能。

5.都可实现删除操作，但是 ListIterator 可以实现对象的修改，set()方法可以实现。Iterator 仅能遍，不能修改。

# ListIterator 和 Iterator 的区别

迭代 List 集合时，推荐使用 ListIterator

1. iterator()方法在 set 和 list 接口中都有定义，但是 ListIterator（）仅存在于 list 接口中（或现类中）；

2. ListIterator 有 add()方法，可以向 List 中添加对象，而 Iterator 不能

3. ListIterator 和 Iterator 都有 hasNext()和 next()方法，可以实现顺序向后遍历，但是 ListIterator 有 hasPrevious()和 previous()方法，可以实现逆向（顺序向前）遍历。Iterator 就不可以。

4. ListIterator 可以定位当前的索引位置，nextIndex()和 previousIndex()可以实现。Iterator 有此功能。

5. 都可实现删除对象，但是 ListIterator 可以实现对象的修改，set()方法可以实现。lierator 仅遍历，不能修改。

因为 ListIterator 的这些功能，可以实现对 LinkedList 等 List 数据结构的操作。其实，数组对象可以用迭代器来实现。

## 为什么迭代器遍历时不能用集合的 remove()方法删除元素?

会发生异常，具体以后再写。

## for 增强循环

for 增强循环实际上内部也是使用的迭代器进行遍历。所以 for 增强循环在遍历集合时也不能删除元素用集合的 remove()方法)

## for 循环和迭代器谁更快?

主要看遍历的集合的数据结构是否合适：

for 循环依据索引来遍历对象，所以在随机访问中比较快(比如 ArrayList)

迭代器的 next()采用的是顺序访问方法，所以在顺序访问的集合中速度更快(比如 LinkedList)

**集合使用迭代器为什么不能使用集合的 remove、clear、add 方法，必须使用迭代器的 remove、ad 方法?**

因为 list 是一个集合对象，它实现了 Iterable 接口，而该接口要求集合实现 iterator 的方法，返回值一个 Iterator 类型。

```
public interface Iterator<AnyType>{
    boolean hasNext();
    AnyType next();
    void remove();
```

在 ArrayList 中，iterator 方法会返回一个 Itr 类型的对象。

```
public Iterator<E> iterator() {
        return new Itr();
    }

//内部类
 private class Itr implements Iterator<E> {
        //游标,指向下一个要返回的元素
        int cursor;       // index of next element to return
        int lastRet = -1; // index of last element returned; -1 if no such
        //修改次数--->用于检测在迭代期间被修改的情况,expectedModeCount初始值是modCount,
过expectedModeCount与modCount的比较来进行错误检测
        int expectedModCount = modCount;

        public boolean hasNext() {
            return cursor != size;
        }

        @SuppressWarnings("unchecked")
        public E next() {
            //错误检测
            checkForComodification();
            int i = cursor;
            if (i >= size)
                throw new NoSuchElementException();
            //elementData存在于外部类ArrayList中,它以数组形式存储着集合元素,通过外部类的隐式引
来使用elementData
            Object[] elementData = ArrayList.this.elementData;
            //如果此时的游标大于集合元素的长度
            if (i >= elementData.length)
                throw new ConcurrentModificationException();
            //游标+1
            cursor = i + 1;
            return (E) elementData[lastRet = i];
        }

        public void remove() {
            if (lastRet < 0)
                throw new IllegalStateException();
            checkForComodification();

            try {
                //迭代器的remove方法,实际上要靠集合的remove方法来实现,
                //值的注意的是:它对游标进行了修改,并且对象expectedModCount进行了修正
                ArrayList.this.remove(lastRet);
                cursor = lastRet;
                lastRet = -1;
                expectedModCount = modCount;
```

```
        } catch (IndexOutOfBoundsException ex) {
            throw new ConcurrentModificationException();
        }
    }

    @Override
    @SuppressWarnings("unchecked")
    public void forEachRemaining(Consumer<? super E> consumer) {
        Objects.requireNonNull(consumer);
        final int size = ArrayList.this.size;
        int i = cursor;
        if (i >= size) {
            return;
        }
        final Object[] elementData = ArrayList.this.elementData;
        if (i >= elementData.length) {
            throw new ConcurrentModificationException();
        }
        while (i != size && modCount == expectedModCount) {
            consumer.accept((E) elementData[i++]);
        }
        // update once at end of iteration to reduce heap write traffic
        cursor = i;
        lastRet = i - 1;
        checkForComodification();
    }

    //错误检测方法,通过比较modCount与expectedModCount是否一致,不一致则抛出异常
    final void checkForComodification() {
        if (modCount != expectedModCount)
            throw new ConcurrentModificationException();
    }
}
```

迭代器的 remove 方法与集合的 remove 方法，最大的不同是，迭代器的 remove 方法中包括对游和 expectedModCount 的修正。

因为 Iterator 是在一个独立的线程中工作的，它在 new Itr()进行初始化时，会记录当时集合中的元，可以理解为记录了集合的状态，在使用集合的 Remove 方法对集合进行修改时，被记录的集合状并不会与之同步改变，所以在 cursor 指向下一个要返回的元素时，可能会发生找不到的错误，即抛出 ConcurrentModificationException 异常。

很明显，如果使用迭代器提供的 remove 方法时，会对 cursor 进行修正，故不会出现错误，此外，会修正 expectedModCount,通过它来进行错误检测(迭代过程中，不允许集合的 add,remove,clear 改变集合结构的操作)。