Iterator接口及其实现

# Iterator<E>接口

public interface **Iterator<E>**

定义了四种方法：**hasNext、next、remove**、forEachRemaining

boolean **hasNext**();

E **next**();

default void **remove**() {

throw new UnsupportedOperationException("remove");

}

default void **forEachRemaining**(Consumer<? super E> action) {

Objects.requireNonNull(action);

while (hasNext())

action.accept(next());

}

迭代器的使用：

Iterator<String> iterator = list.iterator();  
while (**iterator.hasNext()**) {  
 if (iterator.next().length() == 4) {  
  **iterator.remove();**  
 }  
}

# ListIterator<E> extends Iterator<E>

由于List列表这种特殊的集合，可以前后遍历、添加、删除等，继承了Iterator<E>接口，并扩展了几个接口。

boolean hasNext();

E next();

boolean hasPrevious();

E previous();

int nextIndex();

int previousIndex();

void remove();

void set(E e);

void add(E e);

# 实现方式：Iterator与ListIterator

Iterator<E>一般在集合类中作为内部类实现，由于方法很少，一般实现起来比较简单。

一般会设置几个索引属性**cursor**、**lastRet**、expectedModCount等结合集合的size属性实现。

int cursor; // index of next element to return  
int lastRet = -1; // index of last element returned; -1 if no such  
int expectedModCount = modCount;

下面以ArrayList的**Iterator**实现为例：

ArrayList提供了Iterator的实现类Itr();也提供了ListIterator的ListItr()。

## 对外API方法获取迭代器

### 获取Iterator迭代器：

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

### 获取listIterator列表迭代器：

public ListIterator<E> listIterator() {  
 return new ListItr(0);  
}

public ListIterator<E> listIterator(int index) {  
 if (index < 0 || index > size)  
 throw new IndexOutOfBoundsException("Index: "+index);  
 return new ListItr(index);  
}

## 具体实现—循环删除元素的实现

### Itr类

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() {}

### **hasNext方法**：只需要判断当前索引cursor是否到达末尾。

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

### **next方法**：对光标cursor的判断，如果合法，返回当前元素，并**cursor指向下一个**。

cursor自动增1，因此调用next方法之后，光标指向下一个元素。

@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]**;  
 }

### **remove**方法：

真正删除的操作还是通过调用ArrayList的remove方法，改变的是索引，对外不暴露索引，删除之后，仍然保持cursor=lastRet不变，这就是与for循环删除的区别。

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();  
 }  
 }

### forEachRemaining方法

@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();  
}

## ListItr类

ListItr可以提供一个参数index，表示从哪个位置开始迭代；缺省情况下是从0开始。

*/\*\*  
 \* An optimized version of AbstractList.ListItr  
 \*/*private class ListItr extends Itr implements ListIterator<E> {  
 ListItr(int index) {  
 super();  
 cursor = index;  
 }

### hasPrevious方法：

public boolean **hasPrevious**() {  
 return cursor != 0;  
 }

### nextIndex方法：

public int nextIndex() {  
 return cursor;  
 }

### previousIndex

public int previousIndex() {  
 return cursor - 1;  
 }

### previous

@SuppressWarnings("unchecked")  
 public E previous() {  
 checkForComodification();  
 int i = cursor - 1;  
 if (i < 0)  
 throw new NoSuchElementException();  
 Object[] elementData = ArrayList.this.elementData;  
 if (i >= elementData.length)  
 throw new ConcurrentModificationException();  
 cursor = i;  
 return (E) elementData[lastRet = i];  
 }

### set方法

public void set(E e) {  
 if (lastRet < 0)  
 throw new IllegalStateException();  
 checkForComodification();  
  
 try {  
 ArrayList.this.set(lastRet, e);  
 } catch (IndexOutOfBoundsException ex) {  
 throw new ConcurrentModificationException();  
 }  
 }

### add方法

public void add(E e) {  
 checkForComodification();  
  
 try {  
 int i = cursor;  
 ArrayList.this.add(i, e);  
 cursor = i + 1;  
 lastRet = -1;  
 expectedModCount = modCount;  
 } catch (IndexOutOfBoundsException ex) {  
 throw new ConcurrentModificationException();  
 }  
 }  
}