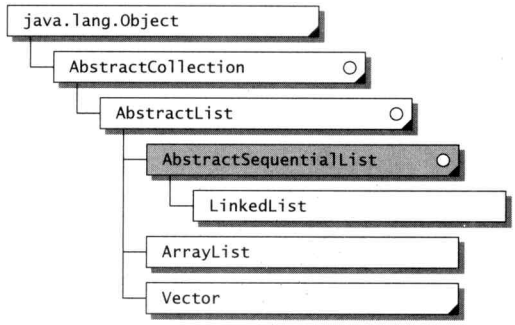
China-pub.com
下载

java.util

AbstractSequentialList



语法

public abstract class AbstractSequentialList extends AbstractList

描述

AbstractSequentialList抽象类是List接口的部分实现。它通过为List接口所需的方法提供缺省的实现，来减少创建自定义的List类时的工作量。此类与更常用的AbstractList不同，它是为那些本身具有良好顺序的表数据结构设计的。

此类没有实现任何存储器的功能。它更像是所提供的数据存储器的装饰性表层。通过重载此类中的一些方法，可以把它"粘"在数据存储器的表层。例如，此类声明一个 size()方法，这个方法返回index中的元素。为了让它返回数据存储器中元素个数，必须实现这个抽象方法。

此类中的方法可以分成两组：必需的方法、实用方法。下面是两个必需的方法：

listIterator(int)	创建一个列表迭代器。
size()	返回列表中元素的个数。

在实例化AbstractSequentialList类的一个子类之前，必须实现这两个抽象方法。

所有其他的方法都是实用方法。所有这些方法都是只用 listIterator()方法和size()方法来实现。如果注重程序的性能，可以重载任何一个实用方法，以使使其效率更高。注意：如果子类的迭代器是错误快速反应迭代器，那么重载的方法需要对 AbstractList.modCount进行增值操作。详见AbstractList.modCount。

列表迭代器

列表迭代器的实现决定了在列表上能进行哪些操作。当实现一个最小的只读列表，需要实现hasNext()方法、 next()方法、 hasPrevious()方法、 previous()方法， nextIndex()方法及previousIndex()方法。

- 实现一个可修改的列表，需要实现set()方法。
- 实现一个可向其中添加元素的列表，需要实现 add()方法。
- 实现一个可从中删除元素的列表，需要实现remove()方法。

如果一个方法没被实现，那么它仅仅是简单地抛出一个UnsupportedOperationException异常。

China-pub.com 下载

使用

AbstractSequentialList类在实际运用中必须子类化。为了便于所创建类的实例化，应当实现一个不带任何参数的构造函数。另外，还应当提供一个接受 Collection对象的构造函数。后一种构造函数用所提供的收集中的元素初始化新建的列表，这样便使得列表可以方便地拷贝任何的收集。当然，还可以提供其他一些构造函数。

同时，为了子类的可实例化，至少还得实现 listIterator()和size()两个抽象方法。仅仅实现这两个抽象方法，那么所创建的列表只是只读的。如果想使列表具有更多的功能，那么必须重载超类AbstractList中的一些可选方法。详见AbstractList类中的“可选方法”部分。

错误快速反应迭代器

有些列表当其迭代器正在被使用时是不允许对列表进行修改的。这些列表可以通过实现错误快速反应迭代器来达到这一目的。一个错误快速反应迭代器会对列表进行监测，如果它发现有对列表进行修改的操作，会抛出一个异常。这一特点主要用来避免程序出错。详见 Iterator。

此类的父类中的modCount域用于实现错误快速反应迭代器。每次对列表进行修改，这个整数值都会增加。详见AbstractList.modCount。

AbstractList与AbstractSequentialList的对比

List接口有两个框架实现。究竟使用哪一个，依赖于数据存储器的访问方式。如果数据存储器中的元素通过下标来访问比较容易时，应当使用 AbstractList类。如果数据存储器更像是一个链表，对其元素进行访问时需要转换其结构，那么应当使用 AbstractSequentialList类。

AbstractList类和AbstractSequentialList类拥有完全相同的方法。它们的主要差别是各个方法所属的分组不同。在 AbstractList类中，get(int)方法是必需的， iterator()不是；而在AbstractSequentialList类中iterator()方法是必需的，get(int)方法却不是。详见AbstractList。

成员概述	
元素检索方法	
get()	检索列表中指定下标处的元素。
修改方法	
add()	插入一个元素到列表中。
addAll()	把一个集合中的所有元素插入到列表中。
remove ()	从列表中删除一个元素。
set()	用一个元素替换另一个元素。
迭代器方法	
iterator()	为列表中的元素创建一个迭代器。
listIterator()	为列表中的元素创建一个列表迭代器。

参见

AbstractList、Iterator、ListIterator、LinkedList。

China-pub.com
下载

示例

此例通过实现一个简单的链表来演示 AbstractSequentialList类的使用方法。此列表非常像一个LinkedList，只是要简单些。

getModCount()方法和incModCount()方法是必需的，因为一个内嵌类不能访问继承的像modCount之类的保护域。注意：既然现在允许内嵌类访问外围类中声明的保护域，那么在将来，这一限制可能会有所放宽，即允许内嵌类访问继承的保护域。

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        List list = new SimpleLinkedList();

        // Test the list.
        list.add("dog");
        list.add("cat");
        list.remove("dog");
        list.add("pig");
        System.out.println( list );    // [cat, pig]
        list.set(list.size()-1, "dog");
        System.out.println( list );    // [cat, dog]

        // Traverse the list backwards
        for (ListIterator it=list.listIterator(list.size());
            it.hasPrevious(); ) {
            System.out.println( it.previous() );
        }    // dog cat

        // Modify the list while an iterator is active.
        Iterator it = list.iterator();
        list.remove("cat");

        it.next();                    // ConcurrentModificationException
    }
}

class SimpleLinkedList extends AbstractSequentialList {
    private Node header = new Node(null, null, null);
    private int size = 0;

    public SimpleLinkedList() {
        header.next = header.previous = header;
    }

    public SimpleLinkedList(List l) {
        super();
        addAll(l);
    }

    public ListIterator listIterator(int index) {
        return new OurListIterator(index);
    }

    public int size() {
        return size;
    }

    private static class Node {
        Node(Node next, Node previous, Object element) {
            this.next = next;
            this.previous = previous;
            this.element = element;
        }
```

China-pub.com 下载

```
        Node next;
        Node previous;
        Object element;
    }

    private class OurListIterator implements ListIterator {
        private Node next;
        private int nextIndex;
        private Node lastReturned = null;
        private int expectedModCount = getModCount();

        OurListIterator(int index) {
            if (index < 0 || index > size) {
                throw new IndexOutOfBoundsException("Index: " + index);
            }

            // Iterate from front or back, whichever is closer
            if (index < size/2) {
                next = header.next;
                for (nextIndex=0; nextIndex<index; nextIndex++) {
                    next = next.next;
                }
            } else {
                next = header;
                for (nextIndex=size; nextIndex>index; nextIndex--) {
                    next = next.previous;
                }
            }
        }

public void add(Object o) {
    checkForComodification();
    Node newNode = new Node(next, next.previous, o);
    next.previous = newNode;
    newNode.previous.next = newNode;
    size++;
    nextIndex++;
    lastReturned = null;
    incModCount();
    expectedModCount++;
}

public boolean hasNext() {
    return nextIndex < size;
}

public boolean hasPrevious() {
    return nextIndex > 0;
}

public Object next() {
    checkForComodification();
    if (!hasNext()) {
        throw new NoSuchElementException();
    }
    lastReturned = next;
    next = next.next;
    nextIndex++;
    return lastReturned.element;
}

public int nextIndex() {
    checkForComodification();
    return nextIndex;
}

public Object previous() {
```

China-pub.com
下载


```
        checkForComodification();
        if (!hasPrevious()) {
            throw new NoSuchElementException();
        }
        next = lastReturned = next.previous;
        nextIndex--;
        return lastReturned.element;
    }

    public int previousIndex() {
        checkForComodification();
        return nextIndex-1;
    }

    public void remove() {
        if (lastReturned==null) {
            throw new IllegalStateException();
        }
        checkForComodification();
        if (next==lastReturned) {
            next = lastReturned.next;
        } else {
            nextIndex--;

                }
                lastReturned.previous.next = lastReturned.next;
                lastReturned.next.previous = lastReturned.previous;
                lastReturned = null;
                size--;
                incModCount();
                expectedModCount++;
            }

            public void set(Object o) {
                if (lastReturned==null) {
                    throw new IllegalStateException();
                }
                checkForComodification();
                lastReturned.element = o;
            }

            private void checkForComodification() {
                if (getModCount() != expectedModCount) {
                    throw new ConcurrentModificationException();
                }
            }
        }

        // "Bridge methods" required to give inner class access to protected field
        private int getModCount() {
            return modCount;
        }
        private void incModCount() {
            modCount++;
        }
    }
```

add()

目的	**把一个元素插入到列表中。**
语法	public void add(int ix，Object e)
描述	**该方法把元素e以下标为ix插入到列表中。** **这是一个实用方法，它的实现首先创建了一个从 ix开始的列表迭代器，然后用列表迭代器的add()方法把e附加到列表中。**

China-pub.com 下载

参数

e　　一个将被插入到列表中的元素，可能为null的元素。

ix　　被插入的元素的下标。0≤ix≤size()。

返回　　重载的返回boolean值的方法总是返回true。

异常

IllegalArgumentException

　　如果e的某些方面阻止其被加入到列表。

IndexOutOfBoundsException

　　如果ix<0或ix>size()。

NullPointerException

　　如果元素e为null并且列表不接受null元素。

UnsupportedOperationException

　　如果列表不支持此方法。

示例　　见List.add()。

addAll()

目的　　把在某一收集中的所有元素插入到列表中。

语法　　public boolean addAll(Collection c)

　　public boolean addAll(int ix, Collection c)

描述　　把收集合c中的所有元素以下标为ix插入到列表中。插入的元素的顺序与它们在c的迭代器中出现的顺序是相同的。如果没有指定ix，则取缺省值size()。

　　此方法是一实用方法，它用列表的迭代器的add()方法来实现。

参数

c　　其所有元素将被插入到列表中的非空集合。

ix　　插入到列表中的元素的下标。0≤ix≤size()。

返回　　如果修改成功，则返回true。

异常

ClassCastException

　　如果c中某元素的类型与列表不匹配。

ConcurrentModificationException

　　如果c的迭代器是错误快速反应迭代器并且在该方法调用期间c正在被修改。

IllegalArgumentException

　　如果c中某元素的某些方面阻止其被加入到列表。

IndexOutOfBoundsException

　　如果ix<0或ix>size()。

NullPointerExcEption

　　如果c中某些元素为null并且列表不接受null元素。

UnsupportedOperationException

　　如果列表不支持此方法。

参见　　add()。

示例　　见List.addAll()。

China-pub.com
下载

get()

目的 检索列表中指定下标的元素。

语法 public Object get(int ix)

描述 此方法返回下标为ix的元素。
这是一个实用方法，它的实现首先在列表创建了一个从 ix开始的列表迭代器，然后用列表迭代器的next()方法来检索元素。

参数

ix 要检索元素的下标。0≤ix<size()

异常

IndexOutOfBoundsException
如果ix<0或ix≥size()。

示例 见List.get()。

iterator()

目的 为列表中元素创建一个迭代器。

语法 public Iterator iterator()

描述 此方法为列表中所有元素创建并返回一个迭代器（详见 Iterator）。由迭代器所表示的元素的顺序与列表中元素的顺序是一致的。
这是一个实用方法，它的实现只是简单地返回一个列表的迭代器。

返回 返回一非null迭代器。

参见 Iterator， listIterator()， modCount。

示例 见List.iterator()。

listIterator()

目的 为列表中元素创建一个列表迭代器。

语法 public abstract ListIterator listIterator(int ix)

描述 此方法为列表中所有元素创建并返回一个列表迭代器（详见 ListIterator）。列表迭代器的游标被初始化为 ix。这意味着由列表迭代器返回的第一个元素是下标为ix的元素。元素 ix的前驱元素对列表迭代器也是可用的。如果没有指定 ix，那么取其缺省值0。
由列表迭代器所返回的元素的顺序与列表中元素的顺序是一致的。
这是一个必需的方法，为了子类的可实例化，必须重载此方法。

参数

ix 列表迭代器游标的初始值。0≤ix≤size()。

异常

IndexOutOfBoundsException
如果ix<0或ix>size()。

参见 iterator()。

示例 使用示例见List.listIterator()；重载示例见类的示例。

remove()

目的 从列表中删除一个下标为ix的元素。

China-pub.com 下载

语法 public Object remove(int ix)

描述 此方法从列表中删除并返回下标为ix的元素。
它是一个实用方法，它的实现首先在列表中创建了一个从ix开始的列表迭代器，然后用列表迭代器的remove()方法来删除下标ix的元素。

参数

ix 一个要从列表中删除的元素的下标。0≤ix<size()。

返回 被删除的可能只是null的元素。

异常

IndexOutOfBoundsException
如果ix<0或ix≥size()。

UnsupportedOperationException
如果列表不支持此方法。

参见 removeAll()、retainAll()。

示例 见List.remove()。

set()

目的 用一个元素替换另一个元素。

语法 public Object set(int ix， Object e)

描述 此方法用元素e替换下标为ix的元素并返回替换前的元素。
这个方法是一个实用方法，它的实现首先创建了列表上一个从ix开始的列表迭代器，然后用列表迭代器的set()方法来替换列表中下标为ix的元素。

参数

e 存储在下标ix处的元素可能是null。

ix 被替换的元素的下标。0≤ix<size()。

返回 可能是null元素被替换。

异常

ClassCastException
如果e的类与列表不匹配。

IllegalArgumentException
如果e的某些方面不允许其添加到列表中。

IndexOutOfBoundsException
如果ix<0或ix≥size()。

NullPointerException
如果e为null并且列表不接受null元素。

UnsupportedOperationException
如果列表不支持此方法。

示例 见List.set()。