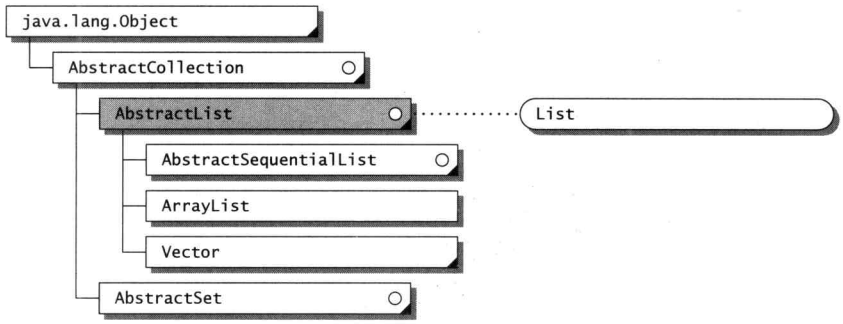
java.util

AbstractList



语法

public abstract class AbstractList extends AbstractCollection implements List

描述

AbstractList抽象类是List接口的部分实现。它通过为List接口所需的大部分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。

此类没有实现任何存储的功能。它更像您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，这个方法返回列表中元素的个数。为了让它返回数据存储器中的元素个数，必须重载这个方法。

此类中的方法可以分成三组：必需的方法、可选的方法及实用方法。

必需的方法

为了可实例化，AbstractList类的子类必须实现以下两个抽象方法：

get(int)	根据所给下标返回列表中的一个元素。
size()	返回列表中元素的个数。

可选的方法

这些方法的实现部分仅仅是抛出一个 UnsupportedOperationException异常。需要重载哪些方法，依赖于创建的列表所要实现的功能。例如，如果所创建的列表仅仅要求是只读的，那么不必重载任何的可选方法。如果创建的列表要求是可修改的，那么必须重载 set(int， Object)方法。如果创建的列表要求能向其中添加元素，必须重载 add(int, Object)方法，等等。下面是列表的可选方法，以及在重载它们之后列表所具有的功能。

add(int index, Object e)	使列表具有能向其中添加元素的功能。
remove(int index)	使列表具有能从其中删除元素的功能。
set(int index, Object e)	使列表具有能被修改的功能。

实用方法

这些方法的实现是基于可选方法的，所以不必对它们进行重载。例如， add(Object)是一个实用方法，它的实现是基于可选方法 add(int，Object)的。如果关心程序的效率，或许应重载下面列出的某些实用方法。

add(Object)
addAll(int, Collection)
addAll(Collection) —— 继承的
clear()
contains(Object) —— 继承的
containsAll(Collection) —— 继承的
indexOf(Object)
iterator()
isEmpty() —— 继承的
lastIndexOf(Object)
listIterator()
listIterator(int)
remove(Object) —— 继承的
removeAll(Collection) —— 继承的
void removeRange(int, int) —— 保护的
retainAll(Collection) —— 继承的
subList(int, int)
Object[] toArray() —— 继承的
Object[] toArray(Object[]) —— 继承的

使用

此类在实际运用中必须子类化。为了便于所创建的类的实例化，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get(int, bject)和size()两个抽象方法。仅仅实现这两个抽象方法，所创建的列表是只读的。如果想使列表具有更多的功能，那么必须重载一些可选方法。详见前面的“ 可选的方法 ”部分。

错误快速反应迭代器

大部分列表在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。此类中的迭代器的实现自动用这个域来探测是否对列表进行了修改。

AbstractList与AbstractSequentialList的对比

List接口有两部分实现。究竟使用哪一部分，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，对其元素进行访问是需要转换其结构，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AbstractSequentialList。

成员概述	
元素检索方法	
get()	检索列表中指定下标处的元素。
indexOf()	判定列表中第一个出现的元素的下标。
lastIndexOf()	判定列表中最后一个出现的元素的下标。
修改方法和域	
add()	插入或附加一个元素到列表中。
addAll()	把一个收集中的所有元素插入或附加到列表中。
clear()	删除列表中的所有元素。
modCount	记录列表被修改过的次数。
remove ()	从列表中删除一个元素。
removeRange ()	删除列表中指定区域的所有元素。
set()	用另一元素覆盖一个元素。
迭代器方法	
iterator()	为列表中的所有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。
子列表方法	
subList()	创建由列表中某指定区域中的元素构成的子列表。
对象方法	
hashCode()	计算列表的哈希码。
equals()	判断列表是否与另一对象相等。

参见

AbstractSequentialList、Iterator、ListIterator。

示例

此例通过实现一个简单的、元素个数有限的列表来演示此类的使用方法。列表中元素的个数不能超过在其构造函数中指定的元素个数。收集内部使用一个Object数组来保存其元素。

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        List list = new FixedList(2);
        list.add("dog");
        list.add("cat");
        //list.add("pig");              // IllegalStateException
        list.remove("dog");
        list.add("pig");
        System.out.println( list );     // [cat, pig]

        Iterator it = list.iterator();
        list.remove("cat");
        //it.next();                    // ConcurrentModificationException
    }
}

class FixedList extends AbstractList {
    int size;
    Object[] array;

    FixedList(int capacity) {
        array = new Object[capacity];
    }

    FixedList() {
        this(10);  // Default capacity is 10
    }

    FixedList(Collection c) {
        int cSize = c.size();
        array = new Object[(cSize*110)/100];  // Allow 10% for growth
        c.toArray(array);
        size = cSize;
    }

    public void add(int index, Object e) {
        if (index < 0 || index > size) {
            throw new IndexOutOfBoundsException();
        }
        if (size==array.length) {
            throw new IllegalStateException();
        }
        System.arraycopy(array, index, array, index+1, size-index);
        array[index] = e;
        size++;
        modCount++;
    }

   public Object get(int index) {
       checkIndex(index);
       return array[index];
   }

   public Object remove(int index) {
       checkIndex(index);
       Object e = array[index];
       System.arraycopy(array, index+1, array, index, size-index-1);
       size--;
       array[size] = null;  // Eliminate "false reference"
       modCount++;
       return e;
   }

   // This method is not strictly necessary, but it speeds up the
   // clear method on this list and its sublists.
```

```
    protected void removeRange(int fromIndex, int toIndex) {
        // Needn't check indices, as this is only used internally
        int numMoved = size - toIndex;
        System.arraycopy(array, toIndex, array, fromIndex, numMoved);

        //  Eliminate "false references" to help garbage collector
        int newSize = size - (toIndex-fromIndex);
        while (size != newSize)
            array[--size] = null;
    }

    public Object set(int index, Object e) {
        checkIndex(index);
        Object o = array[index];
        array[index] = e;
        return o;
    }

    public int size() {
        return size;
    }

    // Utility method.
    private void checkIndex(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
    }
}
```

add()

目的 把一个元素插入或附加到列表中。

语法 public boolean add(Object e)public void add(int ix，Object e)

描述 该方法在下标为ix处把元素e插入或附加到列表中。如果没有指定ix，则取缺省值size()。

如果e是null，且收集不接受null元素，将导致Null PointerException。

add(int, Object)方法是一个可选方法对一个可向其中添加元素的列表，必须重载此方法。add(Object)方法是一个实用方法，它用add(int, Object)方法来实现。

如果此列表的迭代器是错误快速反应迭代器，重载的 add(int, Object)方法必须对modCount域进行增值操作（参见modCount）。

参数

e 一个将被插入或附加到列表中的可能为null的元素。

ix 被插入或附加的元素的下标。0≤ix≤size()

返回 重载的返回boolean值的方法总是返回true。

异常

ClassCastException

如果e的类型与列表不匹配。

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

如果该方法没被重载但却被使用了。

示例 使用示例见List.add()，重载示例见类的示例。

addAll()

目的 把某一收集中的所有元素插入或附加到列表中。

语法 public boolean addAll(Collection c)

public boolean addAll(int ix ，Collection c)

描述 把在收集c中的所有元素插入或附加到列表中的下标为ix处。插入或附加的元素的顺序与它们在c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()。

如果c中某些元素为null且列表不接受null元素为，则会抛出一个NullPointerException异常。

此方法是一实用方法，它用add(int, Object)方法来实现。

参数

c 其所有元素将被插入或附加到列表中的非空收集。

ix 插入或附加到列表中的元素e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。

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

如果没有重载add(int，Object)方法却使用了它。

参见 add()。

示例 见List.addAll()。

clear()

目的 删除列表中所有元素。

语法 public void clear()

描述 该方法用于删除列表中的所有元素。如果它被调用，列表将变为空。此方法的实现调用了removeRange()方法。

缺省的实现抛出UnsupportedOperationException异常。如果一个子类要支持这种方法，必须重载该方法。

异常

UnsupportedOperationException

如果列表不支持该方法。

示例 见Set.clear()。

equals()

目的 判断列表是否与另一对象相等。

语法 public boolean equals(Object c)

描述 此方法判断列表是否与对象 c相等。当且仅当 c是一个列表类的实例且 c中的元素在顺序和数目上与列表中的元素是相同的时候，返回 true。对所有的非空元素，列表中元素的equals()方法同样适用于c中相应的元素。

此方法是一个实用方法，它用listIterator()方法来实现。

参数

c 用来与列表进行比较的对象可能是空对象。

返回 如果相等则返回true。

重载 java.lang.Object.equals()。

参见 hashCode()。

示例 见List.equals()。

get()

目的 检索列表中指定下标的元素。

语法 public abstract Object get(int ix)

描述 此方法返回下标为ix的元素。它是必需的方法，为了子类的可实例化，必须要实现此方法。

参数

ix 返回元素的下标。 0≤ix≤size()。

返回 下标为 ix的元素可能是null。如果是null则返回null。

异常

IndexOutOfBoundsException

如果ix<0或ix≥size()。

示例 使用示例见List.get()；重载示例见类的示例。

hashCode()

目的 计算列表的哈希码。

语法 public int hashCode()

描述 此方法计算列表的哈希码。此哈希码是基于列表中所有元素的哈希码的整数值。两相等的列表有相同的哈希码。两不同的列表也有可能具有相同的哈希码，尽管哈希码的计算算法使得这种可能性非常小。

此方法的实现中所采用的算法在List.hashCode()方法中进行了描述。

返回 列表的哈希码。

重载 java.lang.Object.hashCode()。

参见 equals()， List.hashCode()。

示例 见java.lang.Object.hashCode()。

indexOf()

目的	返回列表中第一个出现的元素的下标。
语法	public int indexOf(Object e)
描述	此方法从表头开始搜索列表中与 e相等（用equals()方法来判断）的元素。如果找到一个，则返回其下标；否则，返回- 1。 这是一个实用方法，它用listIterator()方法来实现。
参数	
e	要搜索的元素，可能是空元素。
返回	返回列表中第一个与e相等的元素的下标，如果e不在列表中则返回-1。
参见	java.lang.Object.equals()。
示例	见List.indexOf()。

iterator()

目的	为列表中所有元素创建一个迭代器。
语法	public Iterator iterator()
描述	此方法为列表所有元素创建并返回一个迭代器（详见 Iterator）。由迭代器所代表的元素的顺序与列表中元素的顺序是一致的。 注意：除非重载了列表的 remove()方法，否则返回的迭代器的 remove()方法将会抛出一个UnsupportedOperationException异常。 这是一个实用方法，它使用 get(int)、remove(int)及size()方法来实现。
返回	返回一非空迭代器。
参见	Iterator、listIterator()、modCount。
示例	见List.iterator()。

lastIndexOf()

目的	返回列表中某元素最后一个出现时的下标。
语法	public int lastIndexOf(Object e)
描述	此方法从表尾开始搜索列表中与 e相等（用equals()方法来判断）的元素。如果找到一个，则返回其下标；否则，返回- 1。 这是一个实用方法，它用listIterator()方法来实现。
参数	
e	要搜索的元素，可能是null。
返回	返回列表中最后一个与e相等的元素的下标，如果e不在列表中则返回-1。
参见	java.lang.Object.equals()
示例	见List.lastIndexOf()。

listIterator()

目的	为列表所有元素创建一个列表迭代器。
语法	public ListIterator listIterator() public ListIterator listIterator(int ix)
描述	此方法为列表中从下标 ix开始的所有元素创建并返回一个列表迭代器（详见

List Iterator)。由列表迭代器所代表的元素的顺序与列表中元素的顺序是一致的。如果没有指定ix，则取缺省值0。

如果ix>0，那么下标ix之前的元素将不会“隐藏”于列表迭代器的后面。换句话说，可以用列表迭代器移到下标为ix的元素的后面。但是，不能移到列表中第一个元素的后面。

注意：除非重载了列表的add()方法、remove()方法及set()方法，否则返回的迭代器中的add()方法、remove()方法及set()方法将会抛出一个UnsupportedOperationException异常。

这是一个实用方法，它使用add(int)方法、get(int)方法、remove(int)方法、set(int)方法及size()方法来实现。

参数

ix　由列表迭代器返回的第一个元素的下标。0≤ix≤size()。

返回　返回列表的非空列表迭代器。

异常

IndexOutOfBoundsException

如果ix<0或ix>size()。

参见　iterator()、ListIterator、modCount。

示例　见List.ListIterator()。

modCount

目的　保存了列表已被修改过的次数。

语法　protected transient int modCount

描述　这是一个记录列表被修改过的次数的域，如果一个迭代器是在此列表上创建的，那么它会记住modCount域的当前值。如果迭代器探测到对该值的非法修改，它会抛出一个ConcurrentModificationException异常。

无论何时，只要对列表进行了修改，都应当对此域进行增值操作。例如，三个可选方法add(int，Object)、set(int，Object)及remove(int)的任何一个被重载，如果使用它们时导致列表被修改，那么它们就应当对modCount域进行增值操作。另外，任何被重载的会导致列表被修改的实用方法都应当对modCount域进行增值操作。注意，永远不要对modCount域进行减值操作。

注意：一个修改方法，尽管没有对列表进行修改，也可能会对modCount域进行增值操作。这样，由迭代器所抛出的ConcurrentModificationException异常有可能仅仅意味着有对列表进行修改的企图。另外，在假定迭代器不会抛出ConcurrentModificationException异常的前提下不能调用一个不会对列表进行修改的修改方法（例如，删除一个不存在的元素）。

这个域的使用是可选的。如果创建的子类的迭代器不必是错误快速反应迭代器，那么便不必对modCount域进行增值操作。然而，如果确实使用了这个域，那么所有的修改方法都必须对modCount域进行增值操作。也就是说，不能在set()方法中对modCount域进行了增值操作，而在remove()方法却没有。

注意：此类中的迭代器的缺省实现要求每次调用的修改方法对modCount域的增值不能超过1。

示例 见类的示例。

remove()

目的 从列表中删除出一个元素。

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。

它是一个可选方法，对于能从中移出元素的列表必须重载此方法。如果此列表上的迭代器是错误快速反应迭代器，这个方法的重载方法必须对 modCount 域进行增值操作（参见modCount）。

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元素。

异常

IndexOutOfBoundsException

如果ix<0或ix≥size()。

UnsupportedOperationException

如果该方法没被重载却被使用了。

参见 removeAll()，retainAll()。

示例 使用示例见List.remove()；重载示例见类的示例。

removeRange()

目的 删除列表某一区域中的所有元素。

语法 protected void removeRange(int fromIx， int toIx)

描述 该方法删除列表中从 fromIx（包括在内）到toIx（不包括在内）区域中的所有元素。

这个保护的方法是一个实用方法。重载此方法将显著地提高列表及其子列表视图中的clear操作的性能。如果没有重载此操作且listIterator(). remove()操作以线性时间运行，那么clear操作将以平方时间运行。

参数

fromIx 被删除的第一个元素的下标。0≤fromIx≤size()。

toIx 下标为toIx-1的元素是被移出的最后一个元素。fromIx≤toIx≤size()。

异常

UnsupportedOperationException

如果既没有重载此方法又没有重载remove(int)方法。

示例 见类的示例。

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。

这个方法是一个可选方法，对于可修改的列表，必须重载此方法。如果列表上的迭代器是错误快速反应迭代器，此方法的重载需要对 modCount域进行增值操

作（参见modCount）。

参数

e 将存储在ix处的元素可能是null元素。

ix 被替换元素的下标。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()。

UnsupportedOperationException

如果没有重载此方法却使用了它。

示例 使用示例见List.set()；重载示例见类的示例。

subList()

目的 创建列表某区域元素的子列表。

语法 public List subList(int fromIx，int toIx)

描述 此方法创建并返回包含列表中从fromIx（包括在内）到toIx（不包括在内）区域中的所有元素。下标为toIx的元素不包括在内。子列表中元素的顺序与原列表中相应元素的顺序是完全一致的。新建的子列表的大小为toIx减fromIx。子列表中第一个元素的下标为0。子列表不能访问列表中从fromIx（包括在内）到toIx（不包括在内）区域之外的任何元素。

一个子列表没有独立的存储器，它与其父列表共享存储器。对子列表的任何修改都会导致父列表作相同的修改。如果父列表直接或通过另一个子列表被修改，那么此子列表便不再有效，应当废弃它。

有些subList()方法的实现会监测底层列表的改变。如果子列表的一个方法探测到这种修改，它会抛出一个ConcurrentModificationException异常。

参数

fromIx 包含到子列表中的第一个元素的下标。0≤fromIx≤size()。

toIx 包含到子列表中的最后一个元素的下标为toIx-1。fromIx≤toIx≤size()。

返回 返回一包含列表中从fromIx（包括在内）到toIx（不包括在内）区域中所有元素的列表。

异常

IndexOutOfBoundsException

如果fromIx<0或toIx<fromIx或toIx>size()。

示例 见List.subList()。