ArrayList

列表是程序设计中最常用的数据结构之一，我们可以使用数组来保存一系列数据，但是数组属于一类大小确定的数据结构，也就是说在使用数组之前必须提前给出数组的大小。这里我们将介绍一种名为 ArrayList 的数据结构，它提供了比数组更多的功能。ArrayList 是一种具有可变大小的动态数据结构，因此它的大小可以随着程序的运行而增大或减小。ArrayList 是我们接触到的第一个泛型数据结构（generic structure），所谓泛型数据结构是指可以用来保存其他不同类型对象值的数据结构。

ArrayList

在日常生活中，我们经常需要处理这样或那样的列表。例如，在社交网站 Facebook.com 上，学生需要提供喜爱的品牌列表。假设某个人的品牌列表如下：

Tool，U2，Phish，Pink Floyd， Redefining the Moment

你可以像第 7 章介绍的那样定义一个数组来保存一个列表中的数据。例如，为了保存上面的品牌列表，你可以声明一个长度为 5 的 String 类型数组。但是如果以后你想要修改列表的内容（例如，从列表中删除 Tool 和 U2），需要怎样处理呢？你可能需要将其他数组元素移动到新的位置，并且在数组的尾部留下一些空位置。如果你希望向这份列表添加一些新的元素来使得列表中的品牌多于五个，那么情况又会怎样呢？当你使用数组时，你已经没有多余的空间来保存更多的元素信息，所以你必须重新定义一个更大的数组来存储列表元素。

大多数人都认为列表应该具有更大的灵活性。例如在操作数组时，我们不想关心底层实现的各种细节。我们希望通过一些简单的指令，如“在列表的这个位置增加一些元素”，“把这个值从列表中删除”等，来实现各种功能，而列表可以根据实际情况在添加或删除元素时自动增加或减少所需的空间。计算机科学家将我们所描述的这种列表称为*抽象列表*（list abstraction）。抽象列表允许我们指定一些操作（如，添加、删除），但却不需要我们关心这些操作的实现方法（如移动列表中的元素，构建新的数组等）。

Java 通过 ArrayList 类提供了上述功能。ArrayList 对象在内部使用数组来保存数据。因此，ArrayList 可以像数组一样提供快速的随机元素访问特性。与数组不同的是，ArrayList 可以让你轻松地添加或删除元素，因为所有的实现细节都是由 ArrayList 自动处理完成：如果你向列表中加入了新元素，它会自动增加数组的大小；如果你删除了列表中的某些元素，它也会自动处理元素的移动问题。

介绍数组时，我们知道可以定义不同类型的数组。如果你需要存放整数值的数组，你可以声明一个类型为 int[]的变量；对于存放字符串的数组，你可以声明一个类型为 String []的变量。这种语法只能用于定义数组，但是 Sun 公司在 Java5 中引入了一种全新的机制，利用这种机制可以让 ArrayList 具有与数组同样的灵活性。如果你看过 ArrayList 的 API 文档，你会发现它的完整定义是 ArrayList<E>。这是 Java *泛型类*（generic class）的一个实例。

泛型类

像 ArrayList<E>这样的特殊类，它们允许通过类型参数来指明使用的数据类型。

ArrayList<E>中的“E”代表“元素”（“Element”），它指明了 ArrayList 中保存的元素的类型。泛型类与带有参数的方法（parameterized method）很类似。回忆第 3 章有关参数的介绍，通过提供不同的参数值（如高或宽），一个方法可以完成一类相似的计算。在这里，泛型类是把类型信息也作为参数，这样就可以很方便的定义出新的类型。ArrarList<E>类型表示了一系列不同的类型，这些类型的不同之处在于它们保存的元素的类型不同：你可以使用 ArrarList<String>来保存字符串列表，使用 ArrarList<Point>来保存 Point 列表，使用 ArrarList<Color>来保存 Color 列表，等等。需要注意的是，你永远不会定义 ArrarList<E>类型。和其他使用参数的场合一样，“E”在这里只是一个形式上的参数，实际使用之前，你需要用一个具有实际意义的值（类型）去替换它，以便明确的表示你要使用的是能够保存哪种类型元素的 ArrayList。

ArrayList 的基本操作

ArrayList 类包含在 java.util 包中，所以要在程序中使用它，就必须在文件开头添加一个包含声明。创建 ArrayList 实例的语法要比我们前面遇到的各种数据类型的语法都复杂，这主要是由类型参数“E”造成的。例如，你可以用下面的语句定义一个字符串类型的 ArrayList：

```
ArrayList<String> list = new ArrayList<String>();
```

```
//Java 7 引入的简单语法格式
ArrayList<String> list = new ArrayList<>();
```

一旦你创建了一个 ArrayList 对象，你就可以通过调用 add 方法向列表中添加新元素，Java 会检查你添加的元素是否与列表声明的类型一致。你请求的是一个 ArrayList<String>，所以你可以向这个列表添加字符串。当你向 ArrayList 列表添加一个新元素时，它会自动将新元素加入到列表的尾部。

与简单数组不同，打印 ArrayList 中的元素非常简单，因为 ArrayList 类重载了 toString 方法。ArrayList 类中实现的 toString 方法会自动构造一个包含列表中所有元素的字符串，其中所有内容都出现在一对方括号内，并且每个元素之间用逗号分割。需要记住的是 toString 方法会在你打印对象或将对象与其他字符串连接时被自动调用。因此，你可以像下面这段代码一样使用 println 方法来打印 ArrayList 中的元素：

```
System.out.println("list = " + list);
```

有两点需要注意：第一，可以打印一个内容为空的 ArrayList 对象；第二，新添加的元素会自动加入到列表的尾部。ArrayList 类还提供了一个重载的 add 方法用来实现在特定位置插入新元素。调用 add 方法不会破坏列表中已有元素的顺序，它会将指定位置之后的元素全部向后移动一个位置以便给新元素腾出空间。**这个 add 方法需要两个参数，一个指明插入元素的位置信息，另一个指明插入元素的值**。ArrayList 中的位置索引是从 0 开始，这一点与数组和字符串的索引一致。例如，在上面的列表中索引为 1 的位置上插入一个新元素：

```
list.add(1, "middle value");
```

```
System.out.println("now list = " + list);
```

调用 add 方法后，会在索引为 1 的位置上插入一个新的字符串。这意味着原先位于该索引位置及其之后的元素都会向右移动。

ArrayList 也提供了一个可以删除特定位置上的元素的方法。remove 方法在删除元素的同时也会保留元素之间的相对顺序，会自动的将被删除元素之后的元素向左移动。

表 10-1 总结了这一节介绍的 ArrayList 类提供的各种操作。更完整的列表可以参考 Java 在线帮助文档中的相关内容。

表 1 ArrayList 类中的基本方法

方法	描述	ArrayList<String>实例
add(value)	在列表结尾处添加新元素	list.add("end");
add(index, value)	在列表指定索引位置添加新元素，后续元素向右移动一个位置	list.add(1,"middle");
clear()	删除列表中的所有元素	list.clear();
get(index)	返回列表指定索引位置上元素的值	list.get(1);
remove(index)	删除列表指定索引位置上的元素，后续元素向左移动一个位置	list.remove(1);
set(index, value)	将列表中指定索引位置上元素的值替换为指定的值	list.set(2,"hello");
size()	返回列表中当前元素的个数	list.size();

ArrayList 的查找方法

创建了某个 ArrayList 对象之后，你可能还需要检索其中所包含的这些元素的值。ArrayList 类提供了几种不同的方法来满足这个要求。如果你只是想知道某个值是否在列表中，可以调用 contains 方法，它会返回一个布尔类型来说明列表是否包含该值。

有些情况下，仅仅能够确定某个值存在于列表之中还不能满足要求，你可能还需要知道它在列表中的具体位置。例如，你可能想要编写一个方法来替换 ArrayList<String>中的某个特定元素的中保存的单词，你必须首先知道这个单词在列表中的位置。我们可以用 ArrayList 提供的 indexOf 方法来获得某个值在列表中的位置信息。

还有一个与 indexOf 方法功能十分类似的方法：lastIndexOf。顾名思义，这个方法返回某个值在列表中最后一次出现的索引位置。很多情况下，你会更关心某个元素最后一次出现的位置而不是第一次出现的位置。例如，如果银行发现某个 ATM 机器不能正常工作，它肯定希望知道最后一次使用这个机器的客户的名字和帐号。表 10-2 总结了 ArrayList 类提供的查找方法。

所有 ArrayList 提供的查找方法都会调用 equals 方法来比较元素与给定值是否相等。而且这些方法的名字也有一定的通用性，它们在 Java 类库中随处可见。例如，String 类也提供了 indexOf 和 lastIndexOf 两个方法，我们可以用它们查找某个字母或子串在字符串中的位置。

表 2 ArrayList 类提供的查找方法

方法	描述	ArrayList<String>实例
contains(value)	如果列表中包含给定值，则返回 true	list.contains(“hello”);
indexOf(value)	返回给定值在列表中第一次出现的索引位置（列表中不包含给定值时返回-1）	list.indexOf(“world”);
lastIndexOf(value)	返回给定值在列表中最后一次出现的索引位置（列表中不包含给定值时返回-1）	list.lastIndexOf (“hello”);

一个完整的 ArrayList 程序

我们将要讨论的是如何在列表中的每个单词前插入一个“~”符号，这需要将列表的大小扩大一倍。虽然在 ArrayList 中插入“~”符号并不是很常见的功能需求，我们只是希望通过这个简单的例子来说明使用 ArrayList 时需要注意的问题，在实际情况中，你会经常碰到与此类似的问题。

创建一个 ArrayList 对象并在其中存储了几个单词：[four, score, and, seven, years, ago]

```
    // Builds up a list of words, adds tildes, and removes them.


import java.util.*;

public class ArrayListTest {
    public static void main(String[] args) {
        // construct and fill up ArrayList
        ArrayList<String> words = new ArrayList<String>();
        words.add("four");
        words.add("score");
        words.add("and");
        words.add("seven");
        words.add("years");
        words.add("ago");
        System.out.println("words = " + words);
        words.set(2,"hello");
        System.out.println("words = " + words);
        System.out.println("if words contain ~ ?= "+words.contains("~"));

        // insert one tilde in front of each word
        for (int i = 0; i < words.size(); i += 2) {
```

```
            words.add(i, "~");
        }

        System.out.println("after loop words = " + words);
        System.out.println("if words contain ~ ?= "+words.contains("~"));
        if(words.contains("~")){
            System.out.println("first ~  in words = "+words.indexOf("~"));
            System.out.println("last ~  in words = "+words.lastIndexOf ("~"));
        }

        // remove tildes
        for (int i = 0; i < words.size(); i++) {
            words.remove(i);
        }
        System.out.println("after second loop words = " + words);
    }
}
```