# 1. 集合

## 1.1 什么是集合

集合是存储对象的容器。集合中可以存储任意类型的变量。Java提供了多种集合类，不同集合类的数据结构不同，分别适合在不同的场景使用。

## 1.2 集合和数组

数组也用来存储数据，如果定义一个对象数组，例如Object[] objArr；那么对象数组也能存储对象。但是集合和数组有很大区别：

（1）数组长度是固定的，集合的长度是可变的。如果无法预先知道需要多少存储多少个数据，那么很难使用数组。

（2）数组一般只能存储同一类型的元素，集合可存储不同类型的元素。

（3）集合只能存储引用类型的元素。JDK 5实现了自动装箱功能，所以集合形式上也能存储基本数据类型的元素。

## 1.3 Collection接口

集合有很多共性，因此集合抽象出了一个Collection接口，下面讲的单例集合都是Collection接口的实现类。下面通过一个实现类来讲Collection接口中的常用方法。

# 2. Collection接口

为了了解Collection接口中方法，首先要创建一个集合对象，那么需要用多态方式来创建一个Collection接口的实现类对象。

先找到一个实现Collection接口的类。查看API，有如下说明：

“JDK 不提供此接口的任何直接实现：它提供更具体的子接口（如 Set 和 List）的实现。”

也就是说Collection没有直接的实现类，我们可以找Collection子接口Set或者List接口的实现类。其中，ArrayList是List接口的一个实现类，我们就用它来创建集合对象，以此介绍Collection接口中的方法。【提示，目前看API时会遇到<E>、<T>等字样，而且在IDE中会有警告，这涉及到泛型，暂时不用管它】

## 2.1 添加和删除对象方法

（1）public boolean add(Object o)：添加一个元素到集合中。此方法总是返回true，因此返回值不重要。

（2）public boolean addAll(Collection c)：将一个集合的元素全部添加到该集合中。

（3）public boolean remove(Object o)：将元素从集合中删除（如果有重复，只会删除一个）。如果集合中没有该元素，则返回false（不会出现异常）。

（4）public boolean removeAll(Collection c)：移除集合中存在于集合c中的元素，包括重复的元素。只要有一个元素被移除了就返回true。

（5）public void clear()：移除集合中所有的元素，集合变为一个空集合。

## 2.2 其他方法

（1）public boolean contains(Object o)：判断集合中是否包含指定的元素。对于引用类型，contains方法中是调用对象的equals方法来判断的，因此当集合存储自定义对象时，注意重写好自定义类的equals方法。

（2）public boolean isEmpty()：判断集合是否为空。

（3）public int size()：返回集合中元素个数。

（4）public boolean containsAll(Collection c)：判断集合中是否包含了集合c的所有元素。必须全部包含才返回true。

（5）public boolean retainAll(Collection c)：将该集合与集合c进行交集，结果保存在本集合中，如果本集合改变了就返回true，否则返回false。

（6）public Object toArray()：将集合转成Object数组。学了泛型，这个方法更好用。

（7）public String toString()：Collection重写了toString()方法，可以直接将集合以类似数组的方式输出。

下面是将自定义对象存储再集合中的例子。

（1）Student类

|  |
| --- |
| **package** com.zhang.test;  **public class** Student {  **private** String **name**;  **private int** age;   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public int** getAge() {  **return** age;  }   **public void** setAge(**int** age) {  **this**.age = age;  }  *// 构造方法* **public** Student(){}  **public** Student(String name, **int** age) {  **this**.**name** = name;  **this**.age = age;  }  *// 下面重写equals、hashCode和toString方法* **public boolean** equals(Object obj) {  **if**(!(obj **instanceof** Student)) {  **return false**;  }  **if**(obj == **this**) {  **return true**;  }  Student stu = (Student) obj;  **if**(stu.getName().equals(**this**.getName()) && stu.getAge() == **this**.getAge()) {  *// 若姓名和年龄相同，就假定他们是同一对象* **return true**;  } **else** {  **return false**;  }  }   **public int** hashCode() {  **return this**.getName().hashCode() + **this**.getAge();  }   **public** String toString() {  **return this**.getName() + **":"** + **this**.getAge();  } } |

（2）主类

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.ArrayList; **import** java.util.Collection;  **public class** Demo {  **public static void** main(String[] args) {  Collection c = **new** ArrayList();  c.add(**new** Student(**"张三"**, 12));  c.add(**new** Student(**"李四"**, 13));  c.add(5); *//可以加任意类型  // 判断，底层调用的是equals方法* System.***out***.println(c.contains(**new** Student(**"张三"**, 12)));  *// 得到元素个数* System.***out***.println(c.size());  *// 输出集合，调用的是toString()方法，显示很友好，把其中自定义对象equals方法也调用了* System.***out***.println(c);  } } |

## 2.3 Iterator迭代器

迭代器用于对集合元素的遍历。迭代器以内部类的形式存在于集合类中，通过迭代器能够拿到集合中元素，也能通过迭代器来操作元素。

调用集合的iterator()方法能得到此集合的迭代器。Collection接口继承自Iterable接口，正是Iterable接口定义了iterator()方法。iterator()方法返回的是Iterator接口，该接口有如下方法以便遍历和获取元素：

（1）boolean hasNext()：判断是否还有可迭代的元素。

（2）Object next()：获取下一个元素，同时指针再向后移动。如果没有下一个元素，则抛出NoSuchElementException异常。

小知识点：如果一个方法能返回任意类型或者接受任意类型，那么返回值类型或者参数类型就可设置为Object，因为Object是所有类的基类。

因此常用上面的方法实现遍历，例子：（Student类还是上面的代码）

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.ArrayList; **import** java.util.Collection; **import** java.util.Iterator;  **public class** Demo {  **public static void** main(String[] args) {  Collection c = **new** ArrayList();  c.add(**new** Student(**"张三"**, 12));  c.add(**new** Student(**"李四"**, 13));  *// 迭代* Iterator it = c.iterator();  **while**(it.hasNext()) {  *// 先拿出对象。由于存储的都是Student，直接转换* Student stu = (Student) it.next();  System.***out***.println(stu);  */\*  注意，不能这样写：  System.out.println( ((Student) it.next()).getName() );  System.out.println( ((Student) it.next()).getAge() );  这样每次循环都会执行两次next()，会导致跳过元素和出现  NoSuchElementException的异常  \*/* }  } } |

推荐在for循环中创建迭代器并迭代，这样可以尽早释放迭代器资源。代码如下：

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.ArrayList; **import** java.util.Collection; **import** java.util.Iterator;  **public class** Demo {  **public static void** main(String[] args) {  Collection c = **new** ArrayList();  c.add(**new** Student(**"张三"**, 12));  c.add(**new** Student(**"李四"**, 13));  *// 迭代* **for**(Iterator it = c.iterator(); it.hasNext(); ) {  Student stu = (Student) it.next();  System.***out***.println(stu.getName());  System.***out***.println(stu.getAge());  }  } } |

迭代器中还有remove()方法，可在遍历时删除元素，但是使用remove()方法之前必须先调用next()方法，否则会出现IllegalStateException异常。此外，在用迭代器迭代过程中，只能通过迭代器操作元素，不能使用集合本身提供的方法或者其他方法操作集合元素，否则会有安全隐患，会抛出并发修改异常ConcurrentModificationException。这点在List集合中还会强调和加深。

总结：上述讲的Collection方法和迭代器，在下面讲的单例集合（List和Set集合）中都能使用，因为单例集合就是Collection的实现类。

# 3. List集合

## 3.1 简介

List也是接口，最常用的实现类是ArrayList。List集合有如下特点：

（1）List集合是有序的。List集合会保留元素的存储顺序，这样取出元素时是顺序取出。

（2）List集合中元素可以重复，即一个集合能存储多个相同的元素。

## 3.2 List集合特有方法

（1）public void add(int index, Object o)：在指定索引处添加元素。

（2）public Object get(int index)：获取指定位置的元素。

（3）public Object remove(int index)：删除指定索引处的元素，并返回此元素

（4）public Object set(int index, Object o)：将指定索引处的元素修改为新值o，返回值是原本此位置的元素。若索引超出范围，则抛出IndexOutOfBoundsException。

（5）public boolean addAll(int index, Collection c)：在指定索引处添加集合中所有元素。

（6）public int indexOf(Object o)：返回集合中第一次出现该对象的索引，如果没有，就返回-1。

（7）public int lastIndexOf(Object o)：返回集合中最后一次出现该对象的索引，如果没有，就返回-1。

（8）public List subList(int fromIndex, int toIndex)：获得指定区间的子集合。不包含toIndex。

说明：上述介绍的很多方法，比如remove()、indexOf()等都要先找到集合中与目标相同的对象，调用的方法都是调用equals()方法。

List集合有get(int index)方法，这样可以结合Collection的size()方法自行实现遍历，而不需用迭代器。例子：

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.ArrayList; **import** java.util.List;  **public class** Demo {  **public static void** main(String[] args) {  List list = **new** ArrayList();  list.add(**new** Student(**"张三"**, 12));  list.add(**new** Student(**"李四"**, 13));  *// 利用get()和size()遍历List集合* **for**(**int** i = 0; i < list.size(); i++) {  Student stu = (Student) list.get(i);  System.***out***.println(stu);  }  } } |

## 3.3 List特有迭代器——ListIterator

List集合可以直接用Iterator迭代器，还能用List特有的迭代器ListIterator。通过调用List集合对象的listIterator()方法，能得到ListIterator对象。

ListIterator继承自Iterator，基本的用法和Iterator相同，只是ListIterator中增加了如下方法：

（1）public boolean hasPrevious()：判断是否有上一个元素

（2）public Object previous()：获取上一个元素，并将指针再指向前一个元素。

通过上述方法，能够实现逆向遍历List集合。当然，一般先要使用next()方法，将指针指向后面，才可能逆向遍历，因为一开始的游标（指针）都是在第一个元素的前面。

另外，ListIterator还有方法：

（1）public void add(Object o)：将指定的元素插入列表（可选操作）。该元素直接插入到 next 返回的下一个元素的前面（如果有），或者 previous 返回的下一个元素之后（如果有）；如果列表没有元素，那么新元素就成为列表中的唯一元素。新元素被插入到隐式光标前：不影响对 next 的后续调用，并且对 previous 的后续调用会返回此新元素。

借此机会着重讲“并发修改异常”。

前面提到过：在用迭代器过程中，只能通过迭代器操作元素，否则会有安全隐患，产生并发修改异常。例子：集合中存储“hello”，“world”和“java”三个字符串，要求遍历集合时，一旦遇到“world”字符串，就在集合中添加一个“c++”字符串。

错误示范：在用迭代器过程中，使用集合方法操作集合元素：

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.\*;  **public class** Demo {  **public static void** main(String[] args) {  List list = **new** ArrayList();  list.add(**"hello"**);  list.add(**"world"**);  list.add(**"java"**);  *// 迭代器遍历* **for**(Iterator it = list.iterator(); it.hasNext(); ) {  String str = (String) it.next();  **if**(str.equals(**"world"**)) {  *// 用集合的add()方法添加元素* list.add(**"c++"**);  }  }  *// 最后输出* System.***out***.println(list);  } } |

运行程序后出现异常：ConcurrentModificationException。为什么出现并发修改异常呢？因为使用迭代器时，如果使用了集合来操作元素，会导致迭代器并不知道集合被修改了，而迭代器的使用是完全依赖于集合的，因此出现并发修改异常。

解决的办法就是只用迭代器来操作集合（ListIterator中有add方法），或者只用集合的方法来遍历和操作集合。下面是例子。只不过根据他们的实现不用，用ListIterator添加时，元素被添加在当前元素下面，而用集合本身的add()方法添加时，新元素被添加在集合的最后。

方法1：使用List迭代器的add()方法。

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.\*;  **public class** Demo {  **public static void** main(String[] args) {  List list = **new** ArrayList();  list.add(**"hello"**);  list.add(**"world"**);  list.add(**"java"**);  *// 迭代器遍历* **for**(ListIterator it = list.listIterator(); it.hasNext(); ) {  String str = (String) it.next();  **if**(str.equals(**"world"**)) {  it.add(**"c++"**);  }  }  *// 最后输出* System.***out***.println(list); *// [hello, world, c++, java]* } } |

方法2：使用List集合的方法来遍历和操作。

|  |
| --- |
| **package** com.zhang.test;  **import** java.util.\*;  **public class** Demo {  **public static void** main(String[] args) {  List list = **new** ArrayList();  list.add(**"hello"**);  list.add(**"world"**);  list.add(**"java"**);  *// 用集合的size()和get()遍历，并修改* **for**(**int** i = 0; i < list.size(); i++) {  **if**(list.get(i).equals(**"world"**)) {  list.add(**"c++"**);  }  }  *// 最后输出* System.***out***.println(list); *// [hello, world, java, c++]* } } |

## 3.4 List集合常用子类

List集合常用的子类有ArrayList（前面一直用的）、LinkedList和Vector。详细解释如下：

（1）ArrayList：底层的数据结构是动态数组。特点是查询速度快（允许直接用索引来查找对应的元素）、增删慢（插入和删除操作涉及元素移动和内存操作）。ArrayList是线程不安全的类，效率较高。

（2）LinkedList：底层数据结构是用链表存储数据。链表的特点是查询慢（需要遍历节点查找），增删快（只需记住上个节点和下个节点）。LinkedList也是线程不安全的类，效率较高。LinkedList中一些特有的方法（需要针对头尾的操作）：

addFirst(E e)、addLast(E e)、getFirst()、getLast()、removeFirst()和removeLast()等。

（3）Vector：是线程安全的ArrayList。用法和ArrayList相似。说明一下，Vector中提供了一个elements()实例方法，返回的是一个Enumeration对象。此对象用法和Iterator一致，只不过方法名不一样，方法原型为：

public boolean hasMoreElement();和public Object nextElement()。

除此之外，List的实现类还有Stack，就是堆栈结构。其实List就是数据结构里面的线性表，使用时，结合数据结构的知识，就能很好的选择使用，并能迅速的了解不同结构中提供的方法的含义。因此推荐学习完C语言之后研究下数据结构和算法。

# 4. 泛型（Generic）

## 4.1 引入泛型

集合能存储不同的数据类型，但这个特点可能会出现安全问题。因为操作集合元素时，要先拿到该元素，用集合（或迭代器）的方法只能拿到Object类型元素，因为集合能存储任意类型，集合本身无法知道自己所存储的究竟是什么类型。因此我们拿到Object类型元素时再将元素强制转换成需要的类型，这时可能发生转换错误。

例子：

|  |
| --- |
|  |