**Java 集合与数组之间的转换**

@author ixenos

**数组转集合 Arrays.asList(T... a)**

**先给结论**：用 Arrays.asList(T... a) 将数组转换成集合

　　　　　　　（T...a）是可变参数，看成(T[] a)即可

　　<T> [List](http://www.cnblogs.com/)<T> **asList**(T... a)

　　返回一个受指定数组支持的**固定大小**的列表，这是披着List外表的纯数组，只是原数组的一个视图（对返回列表的更改会“直接写”到数组。）

　　返回的列表是可序列化的，并且实现了 RandomAccess

　　此方法还提供了一个创建固定长度的**列表（是列表哦）**的便捷方法，该列表被初始化为包含多个元素：

List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");

　　那为什么是固定大小的呢？不是“ArrayList”吗？

　　因为这根本就不是你想象中的“ArrayList”，而且对外也没承诺是ArrayList，只是返回一个接口类型List！

**源码分析：**

　　首先看asList方法：

1 public static <T> List<T> asList(T... a) {

2 return new ArrayList<>(a);

3 }

　　突然发现asList方法原来调用的是Arrays类的**私有静态内部类**ArrayList的构造方法（而不是集合框架中的ArrayList）：

0 private final E[] a;  
1 ArrayList(E[] array) {

2 a = Objects.requireNonNull(array);

3 }

　　（注意：这个类同样继承了AbstractList<E>接口，与工具类ArrayList结构上相似（都是维护一个数组），但内部原理却不同，它只是数组的一个List视图而已。下面是源码）

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 private static class ArrayList<E> extends AbstractList<E>

2 implements RandomAccess, java.io.Serializable

3 {

4 private static final long serialVersionUID = -2764017481108945198L;

5 private final E[] a;

6

7 ArrayList(E[] array) {

8 a = Objects.requireNonNull(array);

9 }

10

11 @Override

12 public int size() {

13 return a.length;

14 }

15

16 @Override

17 public Object[] toArray() {

18 return a.clone();

19 }

20

21 @Override

22 @SuppressWarnings("unchecked")

23 public <T> T[] toArray(T[] a) {

24 int size = size();

25 if (a.length < size)

26 return Arrays.copyOf(this.a, size,

27 (Class<? extends T[]>) a.getClass());

28 System.arraycopy(this.a, 0, a, 0, size);

29 if (a.length > size)

30 a[size] = null;

31 return a;

32 }

33

34 @Override

35 public E get(int index) {

36 return a[index];

37 }

38

39 @Override

40 public E set(int index, E element) {

41 E oldValue = a[index];

42 a[index] = element;

43 return oldValue;

44 }

45

46 @Override

47 public int indexOf(Object o) {

48 E[] a = this.a;

49 if (o == null) {

50 for (int i = 0; i < a.length; i++)

51 if (a[i] == null)

52 return i;

53 } else {

54 for (int i = 0; i < a.length; i++)

55 if (o.equals(a[i]))

56 return i;

57 }

58 return -1;

59 }

60

61 @Override

62 public boolean contains(Object o) {

63 return indexOf(o) != -1;

64 }

65

66 @Override

67 public Spliterator<E> spliterator() {

68 return Spliterators.spliterator(a, Spliterator.ORDERED);

69 }

70

71 @Override

72 public void forEach(Consumer<? super E> action) {

73 Objects.requireNonNull(action);

74 for (E e : a) {

75 action.accept(e);

76 }

77 }

78

79 @Override

80 public void replaceAll(UnaryOperator<E> operator) {

81 Objects.requireNonNull(operator);

82 E[] a = this.a;

83 for (int i = 0; i < a.length; i++) {

84 a[i] = operator.apply(a[i]);

85 }

86 }

87

88 @Override

89 public void sort(Comparator<? super E> c) {

90 Arrays.sort(a, c);

91 }

92 }

[复制代码](javascript:void(0);)

　　从此内部类看出返回的List当成数组看，则不能添加和删除元素，只能进行替换和遍历等不改变结构的操作

　　而构造方法中的requireNonNull方法是非空检查（来自Objects工具类）：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 public static <T> T requireNonNull(T obj) {

2 if (obj == null)

3 throw new NullPointerException();

4 return obj;

5 }

　　可见，没有做任何处理，只是实实在在地传递了一个泛型数组给引用a，说明内部数据结构就是数组

**集合转数组 接口方法toArray直接用**

**Collection<E>接口的toArray()和toArray(T[] a)：**

**1.Object[] toArray()**

　　返回包含此 collection 中所有元素的数组，**但却是Object类型的**，而无端地**向下转型**是非法的，所以连类型转换都不能；

　　如果 collection 对其迭代器返回的元素顺序做出了某些保证（如Comparator参数），那么此方法必须以相同的顺序返回这些元素；

　　返回的数组将是“安全的”，因为此 collection 并不维护对返回数组的任何引用，即不是数组的视图，调用者可以随意修改返回的数组

**一句话**，一般还是别用这个好了，类型是硬伤……

**2.<T> T[] toArray(T[] a)**

　　返回数组的**运行时类型**与指定数组的**运行时类型**相同，故可传入一个指定类型数组；

**如果指定的数组能容纳该 collection，则返回包含此 collection 元素的数组，否则，将分配一个具有指定数组的运行时类型和此 collection 大小的新数组**；

　　如果指定的数组能容纳 collection，并**有剩余空间**（即数组的元素比 collection 的元素多），那么会将**数组**中紧接 collection **尾部的元素设置为null**（*只有* 在调用者知道此 collection 没有包含任何 null 元

素时才能用此方法确定 collection 的长度。）

　　此方法允许对输出数组的运行时类型进行精确控制，并且在某些情况下，可以用来节省分配开销；

**一句话**，形参的数组是个容器，够用就用（剩余设null），不够用再建一个新的（运行时类型和形参的相同）

**3.结论**：用集合自身的方法 toArray(T[] a) 将集合转换成数组

　　　　参数(T[] a)用来指定运行时类型