

collections 工具类

★ 以下是常用列表：

方法签名	使用示例	功能描述
<code>void sort(List<T> list)</code>	<code>Collections.sort(myList);</code>	根据元素的自然顺序（Comparable接口实现）对List集合进行升序排序
<code>void sort(List<T> list, Comparator<? super T> c)</code>	<code>Collections.sort(myList, new MyComparator());</code>	根据自定义Comparator的规则对List集合进行排序
<code>void reverse(List<?> list)</code>	<code>Collections.reverse(myList);</code>	反转List中元素的顺序
<code>void shuffle(List<?> list)</code>	<code>Collections.shuffle(myList);</code>	对List集合中的元素进行随机排序
<code>void swap(List<?> list, int i, int j)</code>	<code>Collections.swap(myList, 0, 1);</code>	将List中索引为i和j的两个元素位置互换
<code><T> boolean addAll(Collection<? super T> c, T... elements)</code>	<code>Collections.addAll(myList, element1, element2, element3);</code>	将一组元素添加到指定集合c中
<code><T extends Comparable<? super T>> T max(Collection<? extends T> coll)</code>	<code>Integer maxNum = Collections.max(myIntegerList);</code>	返回集合中根据自然顺序的最大元素
<code><T> T max(Collection<? extends T> coll, Comparator<? super T> comp)</code>	<code>String maxStr = Collections.max(myStringList, new MyStringComparator());</code>	返回集合中根据Comparator指定顺序的最大元素
<code><T extends Comparable<? super T>> T min(Collection<? extends T> coll)</code>	<code>Integer minNum = Collections.min(myIntegerList);</code>	返回集合中根据自然顺序的最小元素
<code><T> T min(Collection<? extends T> coll, Comparator<? super T> comp)</code>	<code>String minStr = Collections.min(myStringList, new MyStringComparator());</code>	返回集合中根据Comparator指定顺序的最小元素
<code>int binarySearch(List<? extends Comparable<? super T>> list, T key)</code>	<code>int index = Collections.binarySearch(mySortedList, myKey);</code>	在有序List中执行二分查找，返回key对应的索引或负数（表示插入点）
<code>boolean replaceAll(List<T> list, T oldVal, T newVal)</code>	<code>Collections.replaceAll(myList, oldValue, newValue);</code>	替换列表中所有出现的oldVal元素为newVal
<code>void fill(List<? super T> list, T obj)</code>	<code>Collections.fill(myList, myObject);</code>	用指定对象填充整个列表
<code>void copy(List<? super T> dest, List<? extends T> src)</code>	<code>Collections.copy(destList, srcList);</code>	将src列表的元素复制到dest列表中
<code><T> List<T> unmodifiableList(List<? extends T> list)</code>	<code>List<T> unmodifiableMyList = Collections.unmodifiableList(myList);</code>	创建并返回一个不可修改的新列表视图
<code><T> Set<T> unmodifiableSet(Set<? extends T> s)</code>	<code>Set<T> unmodifiableMySet = Collections.unmodifiableSet(mySet);</code>	创建并返回一个不可修改的新集合理视图
<code><K,V> Map<K,V> unmodifiableMap(Map<? extends K,? extends V> m)</code>	<code>Map<K,V> unmodifiableMyMap = Collections.unmodifiableMap(myMap);</code>	创建并返回一个不可修改的新映射视图
<code><T> Collection<T> synchronizedCollection(Collection<T> c)</code>	<code>Collection<T> syncMyCollection = Collections.synchronizedCollection(myCollection);</code>	返回指定集合的线程安全包装
<code><T> List<T> synchronizedList(List<T> list)</code>	<code>List<T> syncMyList = Collections.synchronizedList(myList);</code>	返回指定列表的线程安全包装

方法签名	使用示例	功能描述
<code><T> Set<T> synchronizedSet(Set<T> s)</code>	<code>Set<T> syncMySet = Collections.synchronizedSet(mySet);</code>	返回指定集合并发安全的包装
<code><K, V> Map<K, V> synchronizedMap(Map<K, V> m)</code>	<code>Map<K, V> syncMyMap = Collections.synchronizedMap(myMap);</code>	返回指定映射并发安全的包装

这里的 `MyComparator` 和 `MyStringComparator` 是自定义的比较器类，实现了 `Comparator` 接口。在实际使用时，需要替换为具体的比较器实现。同时，请确保集合中的元素类型与 `Comparator` 支持的类型相匹配。

★ 代码实例

```

1      public static void main(String[] args) {
2          List<String> list = new ArrayList<>(Arrays.asList("A", "C", "B"));
3          Collections.sort(list); //排序
4          Collections.reverse(list); //反转
5          Collections.shuffle(list); //随机排序
6          Collections.fill(list, "D"); //快速填充，会把里面的说有元素都变为D
7
8          List<Integer> integerList = new ArrayList<>(Arrays.asList(1, 2, 100, 20));
9          System.out.println(Collections.max(integerList)); //获取最大值
10         System.out.println(Collections.min(integerList)); //获取最小值
11         System.out.println(Collections.binarySearch(integerList, 2)); //二分查找必
    须要实现比较器接口Comparable
12         List<Integer> newList = Collections.unmodifiableList(integerList); //将
    他变为只读
13         System.out.println(Collections.indexOfSubList(list, Arrays.asList(2,
14         100))); //寻找子集合的位置
15         List<Integer> emptyList = Collections.emptyList(); //创建自读的空集合
16         List<Integer> singletonList = Collections.singletonList(1); //创建只有一个
    元素的只读集合
17         System.out.println(list);
18     }

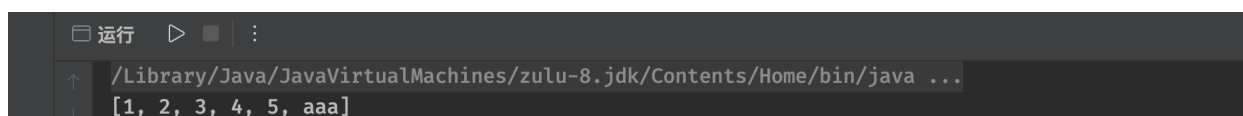
```

得益于泛型的类型擦除机制，实际上最后只要是 `Object` 的实现类都可以保存到集合类中，那么就会出现这种情况：

```

1      public static void main(String[] args) {
2          //使用原始类型接收一个Integer类型的ArrayList
3          List list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
4          list.add("aaa"); //我们惊奇地发现，这玩意居然能存字符串进去
5          System.out.println(list);
6      }

```



```

运行
/Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home/bin/java ...
[1, 2, 3, 4, 5, aaa]

```

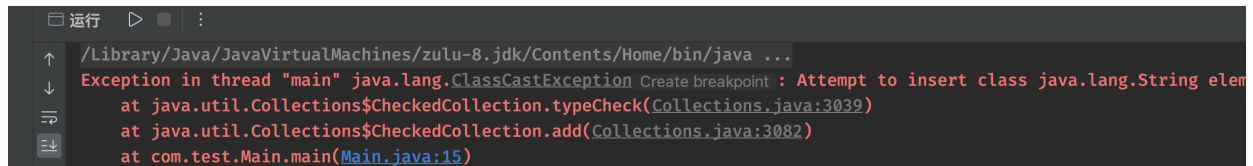
没错，由于泛型机制上的一些漏洞，实际上对应类型的集合类有可能会存放其他类型的值，泛型的类型检查只存在于编译阶段，只要我们绕过这个阶段，在实际运行时，并不会真的进行类型检查，要解决这种问题很简单，就是在运行时进行类型检查：

```

1 public static void main(String[] args) {
2     List list = new ArrayList<>(Arrays.asList(1,2,3,4,5));
3     list = Collections.checkedList(list, Integer.class);    //这里的.class关键字我们
                        会在后面反射中介绍，表示Integer这个类型
4     list.add("aaa");
5     System.out.println(list);
6 }

```

`checkedxxx` 可以将给定集合类进行包装，在运行时同样会进行类型检查，如果通过上面的漏洞插入一个本不应该是当前类型集合支持的类型，那么会直接抛出类型转换异常：



```

/Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home/bin/java ...
Exception in thread "main" java.lang.ClassCastException: Attempt to insert class java.lang.String element
into java.util.Collections$CheckedCollection
    at java.util.Collections$CheckedCollection.typeCheck(Collections.java:3039)
    at java.util.Collections$CheckedCollection.add(Collections.java:3082)
    at com.test.Main.main(Main.java:15)

```