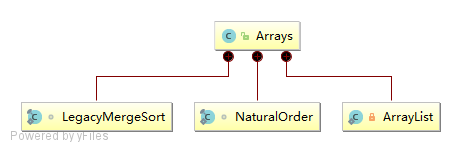
Java源码 —— Arrays

Arrays包含了一系列操作数组的方法（比如排序和搜索）；

Arrays提供了一个能够将数组转换成List列表的方法；

Arrays如果指定数组引用为null，将抛出NullPointerException（一些特定方法除外，比如hashCode）。

## 一、Arrays类图



## 二、Arrays

此类中不同类型重载方法过多，这里只记载一种类型的方法，其他类型的重载方法与之类似。

此类中提供的方法均为static修饰，且不允许通过new来创建Arrays对象。

### 2.1 asList(T... a)

1. 返回一个受指定数组支持的大小固定的列表；
2. 传入数组如果是一个基本类型数组（如int[]），那么将返回一个大小为1的列表，且列表元素就是传入的这个数组；
3. 返回的列表不可添加或删除元素，可以对列表中数据做替换操作（即不可更改列表的大小）；
4. 更改列表中元素内容即更改传入数组的内容（返回的列表持有传入数组参数的引用，不是单独的两个对象）。

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

// 这里的ArrayList并非java.util包下的类，而是Arrays中的内部类，[查看详情](#_五、ArrayList)

return new ArrayList<>(a);// ArrayList持有数组a的引用

}

### 2.2 binarySearch(byte[] a, byte key)

1. 使用二分搜索法来搜索指定的数组，以获得指定的值；
2. 查找元素时，不包括toIndex的值，因此传入的toIndex应比实际下标大1；
3. 返回结果小于0，表示搜索不到指定的元素；
4. Arrays还提供了一个含Comparator对象的方法，此方法能够更快速的定位到搜索值。

// public static int binarySearch(byte[] a, int fromIndex, int toIndex, byte key) {

public static int binarySearch(byte[] a, byte key) {

return binarySearch0(a, 0, a.length, key);

}

private static int binarySearch0(byte[] a, int fromIndex, int toIndex,

byte key) {

int low = fromIndex;

int high = toIndex - 1;

while (low <= high) {

int mid = (low + high) >>> 1;

byte midVal = a[mid];

if (midVal < key)

low = mid + 1;

else if (midVal > key)

high = mid - 1;

else

return mid; // key found

}

return -(low + 1); // key not found.

}

### 2.3 copyOf(int[] original, int newLength)

1. 复制指定的数组，截取或用默认值填充，以使副本具有指定的长度；
2. Arrays.copyOf调用System.arraycopy方法，copy出来的每个副本都是单独的对象。

public static int[] copyOf(int[] original, int newLength) {

int[] copy = new int[newLength];

System.arraycopy(original, 0, copy, 0,

Math.min(original.length, newLength));

return copy;

}

### 2.4 deepEquals(Object[] a1, Object[] a2)

1. 如果两个指定数组彼此是深层相等 的，则返回 true；
2. 不支持基本数据类型数组；
3. 先比较数组长度，长度相等时再用==比较，==结果为false时再用equals比较；
4. Arrays为每种基本类型数组定义了一个equals方法。

public static boolean deepEquals(Object[] a1, Object[] a2) {

if (a1 == a2)

return true;

if (a1 == null || a2==null)

return false;

int length = a1.length;

if (a2.length != length)

return false;

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

Object e1 = a1[i];

Object e2 = a2[i];

if (e1 == e2)

continue;

if (e1 == null)

return false;

// Figure out whether the two elements are equal

boolean eq = deepEquals0(e1, e2);

if (!eq)

return false;

}

return true;

}

static boolean deepEquals0(Object e1, Object e2) {

assert e1 != null;

boolean eq;

if (e1 instanceof Object[] && e2 instanceof Object[])

eq = deepEquals ((Object[]) e1, (Object[]) e2);

else if (e1 instanceof byte[] && e2 instanceof byte[])

eq = equals((byte[]) e1, (byte[]) e2);

else if (e1 instanceof short[] && e2 instanceof short[])

eq = equals((short[]) e1, (short[]) e2);

else if (e1 instanceof int[] && e2 instanceof int[])

eq = equals((int[]) e1, (int[]) e2);

else if (e1 instanceof long[] && e2 instanceof long[])

eq = equals((long[]) e1, (long[]) e2);

else if (e1 instanceof char[] && e2 instanceof char[])

eq = equals((char[]) e1, (char[]) e2);

else if (e1 instanceof float[] && e2 instanceof float[])

eq = equals((float[]) e1, (float[]) e2);

else if (e1 instanceof double[] && e2 instanceof double[])

eq = equals((double[]) e1, (double[]) e2);

else if (e1 instanceof boolean[] && e2 instanceof boolean[])

eq = equals((boolean[]) e1, (boolean[]) e2);

else

eq = e1.equals(e2);

return eq;

}

### 2.5 fill(byte[] a, byte val)

1. 将指定的值分配给指定数组的每一个元素；
2. 指定范围时，替换值时不包括toIndex下标值，因此作为下标需要+1。

public static void fill(byte[] a, byte val) {

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

a[i] = val;

}

public static void fill(byte[] a, int fromIndex, int toIndex, byte val) {

rangeCheck(a.length, fromIndex, toIndex);

for (int i = fromIndex; i < toIndex; i++)

a[i] = val;

}

### 2.6 parallelPrefix(int[] array, IntBinaryOperator op)

1. 对数组arr进行二元迭代；
2. 从头到尾逐个按照left op right的方式更新，更新后的值继续迭代下一个元素；
3. TypeBinaryOperator表示二元运算法，Type目前支持类型为：Int、Double、Long；
4. JDK1.8新增方法。

public static void parallelPrefix(int[] array, IntBinaryOperator op) {

Objects.requireNonNull(op);

if (array.length > 0)

new ArrayPrefixHelpers.IntCumulateTask

(null, op, array, 0, array.length).invoke();

}

// left op right 数组元素

// 1 \* 2 [1, 2, 3]

// 2 \* 3 [1, 2, 6]

System.out.println(Arrays.toString(arr));// [1, 2, 6]

### 2.7 parallelSetAll(int[] array, IntUnaryOperator generator)

1. 使用填充算法为数组arr的每一个元素赋值；
2. 默认使用元素的索引为该元素赋值；
3. 可自定义如何生成赋值；
4. JDK1.8新增方法。

public static void parallelSetAll(int[] array, IntUnaryOperator generator) {

Objects.requireNonNull(generator);

IntStream.range(0, array.length)

.parallel().forEach(i -> {

array[i] = generator.applyAsInt(i);

});

}

### 2.8 parallelSort(int[] a)

1. JDK1.8新增排序方法；
2. 可指定数组范围排序。

public static void parallelSort(int[] a) {

int n = a.length, p, g;

if (n <= MIN\_ARRAY\_SORT\_GRAN ||

(p = ForkJoinPool.getCommonPoolParallelism()) == 1)

DualPivotQuicksort.sort(a, 0, n - 1, null, 0, 0);

else

new ArraysParallelSortHelpers.FJInt.Sorter

(null, a, new int[n], 0, n, 0,

((g = n / (p << 2)) <= MIN\_ARRAY\_SORT\_GRAN) ?

MIN\_ARRAY\_SORT\_GRAN : g).invoke();

}

Jdk1.8之前的排序方法sort

public static void sort(byte[] a) {

DualPivotQuicksort.sort(a, 0, a.length - 1);

}

### 2.9 spliterator(int[] array)

1. 获取一个分割器，可用于遍历数组；
2. 可指定范围获取分割器，获取到的分割器中内容不包括endExclusive下标的元素；
3. 除支持复杂类型数组遍历之外，还支持int、double、long三种基本类型数组遍历；
4. 分割器持有数组的引用，并非独立的对象（即改变数组元素将影响遍历的结果）；
5. JDK1.8新增方法。

// public static <T> Spliterator<T> spliterator(T[] array, int startInclusive, int endExclusive) {

public static <T> Spliterator<T> spliterator(T[] array) {

return Spliterators.spliterator(array,

Spliterator.ORDERED | Spliterator.IMMUTABLE);

}

### 2.10 stream(T[] array)

JDK1.8新增方法，将数组作为一个连续的流使用，用法上与[spliterator](#_2.9_spliterator(int[]_array))相似。

// public static <T> Stream<T> stream(T[] array, int startInclusive, int endExclusive) {

public static <T> Stream<T> stream(T[] array) {

return stream(array, 0, array.length);

}

## 三、LegacyMergeSort

1. 排序时可能会用到，外部无法调用；
2. 可在运行代码前设置java.util.Arrays.useLegacyMergeSort参数来控制Arrays.sort方法是否使用老版的排序方法。

**LegacyMergeSort类定义：**

/\*\*

\* Old merge sort implementation can be selected (for

\* compatibility with broken comparators) using a system property.

\* Cannot be a static boolean in the enclosing class due to

\* circular dependencies. To be removed in a future release.

\*/

static final class LegacyMergeSort {

private static final boolean userRequested =

java.security.AccessController.doPrivileged(

new sun.security.action.GetBooleanAction(

"java.util.Arrays.useLegacyMergeSort")).booleanValue();

}

**排序方法：**

public static void sort(Object[] a) {

if (LegacyMergeSort.userRequested)

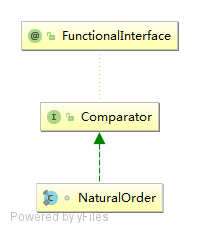
legacyMergeSort(a);

else

ComparableTimSort.sort(a, 0, a.length, null, 0, 0);// JDK1.8新增排序方法

}

## 四、NaturalOrder



一个比较器，它实现了一组相互比较的元素的自然排序。当未提供或提供的比较器为空时使用。

**NaturalOrder类定义：**

/\*\*

\* A comparator that implements the natural ordering of a group of

\* mutually comparable elements. May be used when a supplied

\* comparator is null. To simplify code-sharing within underlying

\* implementations, the compare method only declares type Object

\* for its second argument.

\*

\* Arrays class implementor's note: It is an empirical matter

\* whether ComparableTimSort offers any performance benefit over

\* TimSort used with this comparator. If not, you are better off

\* deleting or bypassing ComparableTimSort. There is currently no

\* empirical case for separating them for parallel sorting, so all

\* public Object parallelSort methods use the same comparator

\* based implementation.

\*/

static final class NaturalOrder implements Comparator<Object> {

@SuppressWarnings("unchecked")

public int compare(Object first, Object second) {

return ((Comparable<Object>)first).compareTo(second);

}

static final NaturalOrder INSTANCE = new NaturalOrder();

}

**比较方法：**

public static <T> void parallelSort(T[] a, Comparator<? super T> cmp) {

if (cmp == null)

cmp = NaturalOrder.INSTANCE;

int n = a.length, p, g;

if (n <= MIN\_ARRAY\_SORT\_GRAN ||

(p = ForkJoinPool.getCommonPoolParallelism()) == 1)

TimSort.sort(a, 0, n, cmp, null, 0, 0);

else

new ArraysParallelSortHelpers.FJObject.Sorter<T>

(null, a,

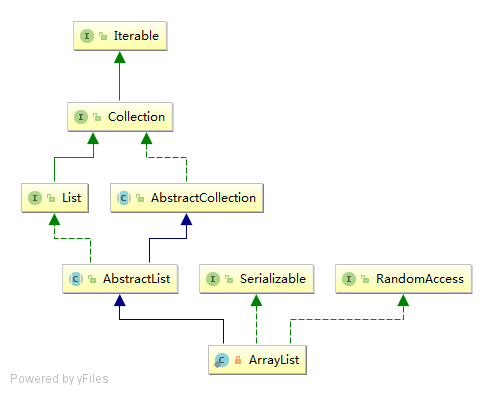
(T[])Array.newInstance(a.getClass().getComponentType(), n),

0, n, 0, ((g = n / (p << 2)) <= MIN\_ARRAY\_SORT\_GRAN) ?

MIN\_ARRAY\_SORT\_GRAN : g, cmp).invoke();

}

## 五、ArrayList



Arrays定义的一个大小固定的私有静态列表类，继承自AbstractList，可用于遍历，不可添加或删除元素（可替换元素）；

此列表中元素内容为所持有的数组对象，变更数组中的值时影响列表的使用。

### 5.1 Arrays.ArrayList

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

implements RandomAccess, java.io.Serializable

{

private static final long serialVersionUID = -2764017481108945198L;

private final E[] a;

ArrayList(E[] array) {

a = Objects.requireNonNull(array);

}

@Override

public int size() {

return a.length;

}

@Override

public Object[] toArray() {

return a.clone();

}

@Override

@SuppressWarnings("unchecked")

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

int size = size();

if (a.length < size)

return Arrays.copyOf(this.a, size,

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

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

if (a.length > size)

a[size] = null;

return a;

}

@Override

public E get(int index) {

return a[index];

}

@Override

public E set(int index, E element) {

E oldValue = a[index];

a[index] = element;

return oldValue;

}

@Override

public int indexOf(Object o) {

E[] a = this.a;

if (o == null) {

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

if (a[i] == null)

return i;

} else {

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

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

return i;

}

return -1;

}

@Override

public boolean contains(Object o) {

return indexOf(o) != -1;

}

@Override

public Spliterator<E> spliterator() {

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

}

@Override

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

Objects.requireNonNull(action);

for (E e : a) {

action.accept(e);

}

}

@Override

public void replaceAll(UnaryOperator<E> operator) {

Objects.requireNonNull(operator);

E[] a = this.a;

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

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

}

}

@Override

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

Arrays.sort(a, c);

}

}

### 5.2更变列表大小的方法（AbstractList）

**Add：**

public boolean add(E e) {

add(size(), e);

return true;

}

public void add(int index, E element) {

throw new UnsupportedOperationException();

}

**Remove：**

public E remove(int index) {

throw new UnsupportedOperationException();

}