

目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	最近阅读
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

04 Arrays、Collections、Objects 常用方法源码解析

更新时间：2019-11-25 18:04:50



“
读一本好书，就是和许多高尚的人谈话。
——歌德”

引导语

我们在工作中都会写工具类，但如何才能使写出来的工具类更好用，也是有一些技巧的。本章内容以三种平时工作中经常使用的工具类为例，从使用案例出发，再看看底层源码的实现，看看能否学习到一些工具类的技巧，以及三种工具类的实际使用场景。

下方是本专栏 GitHub 地址：
源码解析：<https://github.com/luanqiu/java8>
文章 demo：https://github.com/luanqiu/java8_demo
同学们有需要可以对照着来看：)

1 工具类通用的特征

再看细节之前，我们先总结一下好的工具类都有哪些通用的特征写法：

1. 构造器必须是私有的。这样的话，工具类就无法被 new 出来，因为工具类在使用的时候，无需初始化，直接使用即可，所以不会开放出构造器出来。
2. 工具类的工具方法必须被 static、final 关键字修饰。这样的话就可以保证方法不可变，并且可以直接使用，非常方便。

的，可以放心使用。

2 Arrays

Arrays 主要对数组提供了一些高效的操作，比如说排序、查找、填充、拷贝、相等判断等等。我们选择其中两三看下，对其余操作感兴趣的同学可以到 [GitHub](#) 上查看源码解析。

2.1 排序

Arrays.sort 方法主要用于排序，入参支持 int、long、double 等各种基本类型的数组，也支持自定义类的数组，下面我们写个 demo 来演示一下自定义类数组的排序：

```
@Data
// 自定义类
class SortDTO {
    private String sortTarget;

    public SortDTO(String sortTarget) {
        this.sortTarget = sortTarget;
    }
}

@Test
public void testSort(){
    List<SortDTO> list = ImmutableList.of(
        new SortDTO("300"),
        new SortDTO("50"),
        new SortDTO("200"),
        new SortDTO("220")
    );
    // 我们先把数组的大小初始化成 list 的大小，保证能够正确执行 toArray
    SortDTO[] array = new SortDTO[list.size()];
    list.toArray(array);

    log.info("排序之前: {}", JSON.toJSONString(array));
    Arrays.sort(array, Comparator.comparing(SortDTO::getSortTarget));
    log.info("排序之后: {}", JSON.toJSONString(array));
}

输出结果为：
排序之前: [{"sortTarget":"300"}, {"sortTarget":"50"}, {"sortTarget":"200"}, {"sortTarget":"220"}]
排序之后: [{"sortTarget":"200"}, {"sortTarget":"220"}, {"sortTarget":"300"}, {"sortTarget":"50"}]
```

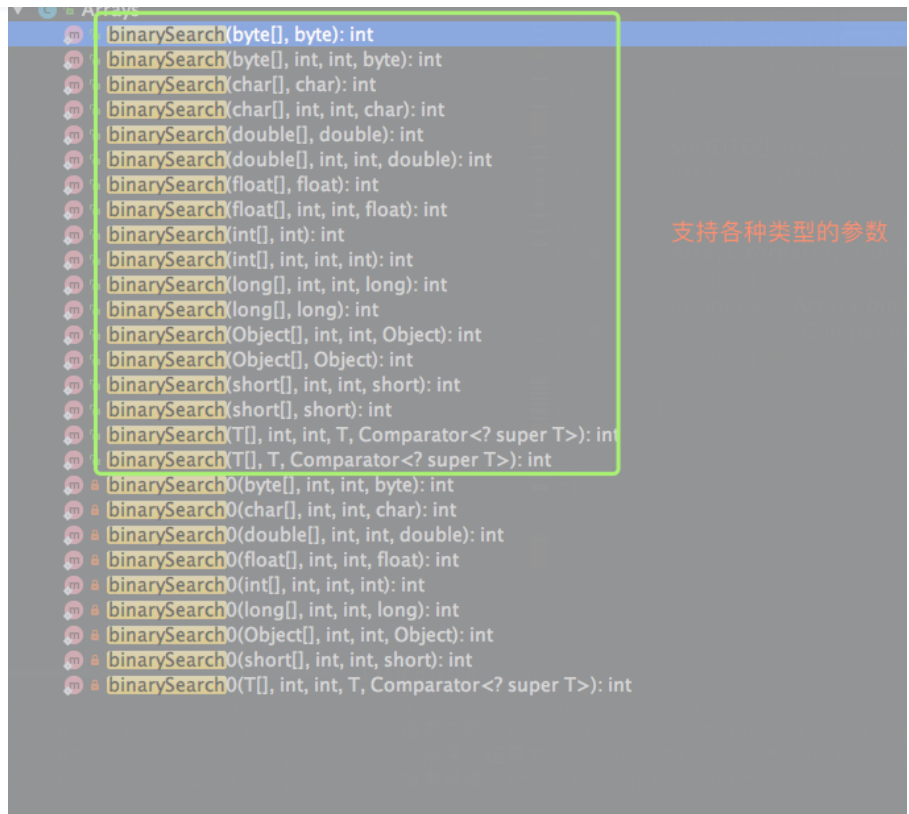
从输出的结果中可以看到，排序之后的数组已经有顺序的了，也可以看到 sort 方法支持两个入参：要排序的数组和外部排序器。

大家都说 sort 方法排序的性能较高，主要原因是 sort 使用了双轴快速排序算法，具体算法就不细说了。

2.1 二分查找法

Arrays.binarySearch 方法主要用于快速从数组中查找出对应的值。其支持的入参类型非常多，如 byte、int、long 各种类型的数组。返回参数是查找到的对应数组下标的值，如果查询不到，则返回负数。

目录



我们写了一个 demo 如下:

```
List<SortDTO> list = ImmutableList.of(
    new SortDTO("300"),
    new SortDTO("50"),
    new SortDTO("200"),
    new SortDTO("220")
);

SortDTO[] array = new SortDTO[list.size()];
list.toArray(array);

log.info("搜索之前: {}", JSON.toJSONString(array));
Arrays.sort(array, Comparator.comparing(SortDTO::getSortTarget));
log.info("先排序, 结果为: {}", JSON.toJSONString(array));
int index = Arrays.binarySearch(array, new SortDTO("200"),
    Comparator.comparing(SortDTO::getSortTarget));
if(index<0){
    throw new RuntimeException("没有找到 200");
}
log.info("搜索结果: {}", JSON.toJSONString(array[index]));
```

输出的结果为:

搜索之前: [{"sortTarget":"300"}, {"sortTarget":"50"}, {"sortTarget":"200"}, {"sortTarget":"220"}]

先排序, 结果为: [{"sortTarget":"200"}, {"sortTarget":"220"}, {"sortTarget":"300"}, {"sortTarget":"50"}]

搜索结果: {"sortTarget":"200"}

从上述代码中我们需要注意两点:

1. 如果被搜索的数组是无序的, 一定要先排序, 否则二分搜索很有可能搜索不到, 我们 demo 里面也先对数组进行了排序;
2. 搜索方法返回的是数组的下标值。如果搜索不到, 返回的下标值就会是负数, 这时我们需要判断一下正负。如果是负数, 还从数组中获取数据的话, 会报数组越界的错误。demo 中对

目录

接下来，我们来看下二分法底层代码的实现：

```
// a: 我们要搜索的数组, fromIndex: 从那里开始搜索, 默认是0; toIndex: 搜索到何时停止, 默认是a.length
// key: 我们需要搜索的值 c: 外部比较器
private static <T> int binarySearch0(T[] a, int fromIndex, int toIndex,
                                     T key, Comparator<? super T> c) {
    // 如果比较器 c 是空的, 直接使用 key 的 Comparable.compareTo 方法进行排序
    // 假设 key 类型是 String 类型, String 默认实现了 Comparable 接口, 就可以直接使用 comparable
    if (c == null) {
        // 这是另外一个方法, 使用内部排序器进行比较的方法
        return binarySearch0(a, fromIndex, toIndex, key);
    }
    int low = fromIndex;
    int high = toIndex - 1;
    // 开始位置小于结束位置, 就会一直循环搜索
    while (low <= high) {
        // 假设 low = 0, high = 10, 那么 mid 就是 5, 所以说二分的意思主要在这里, 每次都是计算中间值
        int mid = (low + high) >> 1;
        T midVal = a[mid];
        // 比较数组中间值和给定的值的大小关系
        int cmp = c.compare(midVal, key);
        // 如果数组中间值小于给定的值, 说明我们要找的值在中间值的右边
        if (cmp < 0)
            low = mid + 1;
        // 我们要找的值在中间值的左边
        else if (cmp > 0)
            high = mid - 1;
        else
            // 找到了
            return mid; // key found
    }
    // 返回的值是负数, 表示没有找到
    return -(low + 1); // key not found.
}
```

二分的主要意思是每次查找之前, 都找到中间值, 然后拿我们要比较的值和中间值比较, 根据结果修改比较的上限或者下限, 通过循环最终找到相等的位置索引, 以上代码实现比较简洁, 大家可以在自己理解的基础上, 自己复写一遍。

2.2 拷贝

数组拷贝我们经常遇到, 有时需要拷贝整个数组, 有时需要拷贝部分, 比如 ArrayList 在 add (扩容) 或 remove (删除元素不是最后一个) 操作时, 会进行一些拷贝。拷贝整个数组我们可以使用 copyOf 方法, 拷贝部分我们可以使用 copyOfRange 方法, 以 copyOfRange 为例, 看下底层源码的实现:

```
// original 原始数组数据
// from 拷贝起点
// to 拷贝终点
public static char[] copyOfRange(char[] original, int from, int to) {
    // 需要拷贝的长度
    int newLength = to - from;
    if (newLength < 0)
        throw new IllegalArgumentException(from + " > " + to);
    // 初始化新数组
    char[] copy = new char[newLength];
    // 调用 native 方法进行拷贝, 参数的意思分别是:
```

从源码中，我们发现，Arrays 的拷贝方法，实际上底层调用的是 System.arraycopy 这个 native 方法，如果你自己对底层拷贝方法比较熟悉的话，也可以直接使用。

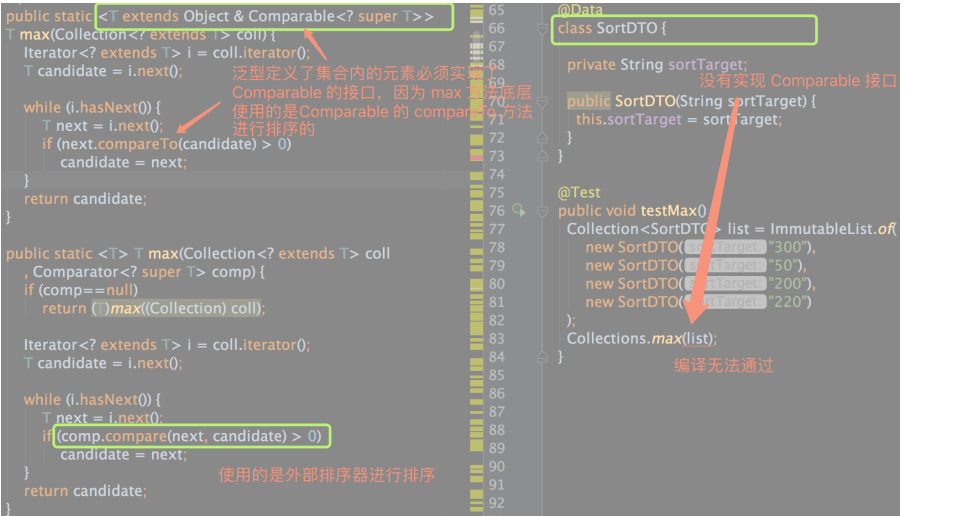
3 Collections

Collections 是为了方便使用集合而产生的工具类，Arrays 方便数组使用，Collections 是方便集合使用。

Collections 也提供了 sort 和 binarySearch 方法，sort 底层使用的就是 Arrays.sort 方法，binarySearch 底层是自己重写了二分查找算法，实现的逻辑和 Arrays 的二分查找算法完全一致，这两个方法上 Collections 和 Arrays 的内部实现很类似，接下来我们来看下 Collections 独有的特性。

3.1 求集合中最大、小值

提供了 max 方法来取得集合中的最大值，min 方法来取得集合中的最小值，max 和 min 方法很相似的，我们以 max 方法为例来说明一下，max 提供了两种类型的方法，一个需要传外部排序器，一个不需要传排序器，但需要集合中的元素强制实现 Comparable 接口，后者的泛型定义很有意思，我们来看下（从右往左看）：



从这段源码中，我们可以学习到两点：

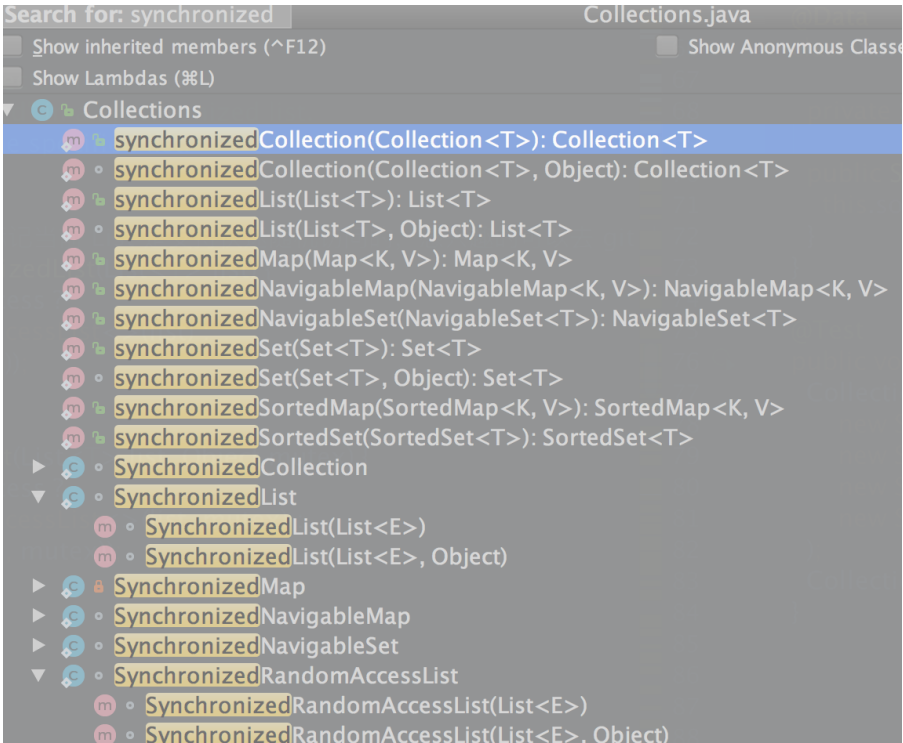
1. max 方法泛型 T 定义得非常巧妙，意思是泛型必须继承 Object 并且实现 Comparable 的接口。一般让我们来定义的话，我们可以会在方法里面去判断有无实现 Comparable 的接口，这种是在运行时才能知道结果。而这里泛型直接定义了必须实现 Comparable 接口，在编译的时候就可告诉使用者，当前类没有实现 Comparable 接口，使用起来很友好；
2. 给我们提供了实现两种排序机制的好示例：自定义类实现 Comparable 接口和传入外部排序器。两种排序实现原理类似，但实现有所差别，我们在工作中如果需要些排序的工具类时，可以效仿。

3.2 多种类型的集合

Collections 对原始集合类进行了封装，提供了更好的集合类给我们，一种是线程安全的集合，一种是不可变的集合，针对 List、Map、Set 都有提供，我们先来看下线程安全的集合：

目录

线程安全的集合方法都是 synchronized 打头的，如下：



从方法命名我们都可以看出来，底层是通过 synchronized 轻量锁来实现的，我们以 synchronizedList 为例来说明下底层的实现：



可以看到 List 的所有操作方法都被加上了 synchronized 锁，所以多线程对集合同时进行操作，是线程安全的。

3.2.1 不可变的集合

为只开放了查询方法，其余任何修改操作都会抛出异常，我们以 `unmodifiableList` 为例来看下底层实现机制：

```
static class UnmodifiableList<E> extends UnmodifiableCollection<E>
    implements List<E> {
    private static final long serialVersionUID = -283967356065247728L;

    final List<? extends E> list;

    UnmodifiableList(List<? extends E> list) {
        super(list);
        this.list = list;
    }

    public boolean equals(Object o) {return o == this || list.equals(o);}
    public int hashCode() {return list.hashCode();}

    public E get(int index) {return list.get(index);}
    public E set(int index, E element) { throw new UnsupportedOperationException(); }
    public void add(int index, E element) { throw new UnsupportedOperationException(); }
    public E remove(int index) { throw new UnsupportedOperationException(); }
    public int indexOf(Object o) {return list.indexOf(o);}
    public int lastIndexOf(Object o) {return list.lastIndexOf(o);}
    public boolean addAll(int index, Collection<? extends E> c) {
        throw new UnsupportedOperationException();
    }

    @Override
    public void replaceAll(UnaryOperator<E> operator) {
        throw new UnsupportedOperationException();
    }

    @Override
    public void sort(Comparator<? super E> c) { throw new UnsupportedOperationException(); }

    public ListIterator<E> listIterator() {return listIterator(index: 0);}
}
```

开放了查询方法，其他任何修改操作都会抛出异常。

3.2.2 小结

以上两种 `List` 其实解决了工作中的一些困惑，比如说 `ArrayList` 是线程不安全的，然后其内部数组很容易被修改，有的时候，我们希望 `List` 一旦生成后，就不能被修改，`Collections` 对 `List` 重新进行了封装，提供了两种类型的集合封装形式，从而解决了工作中的一些烦恼，如果你平时使用 `List` 时有一些烦恼，也可以学习此种方式，自己对原始集合进行封装，来解决 `List` 使用过程中的不方便。

4 Objects

对于 `Objects`，我们经常使用的就是两个场景，相等判断和判空。

4.1 相等判断

`Objects` 有提供 `equals` 和 `deepEquals` 两个方法来进行相等判断，前者是判断基本类型和自定义类的，后者是用来判断数组的，我们来看下底层的源码实现：

目录

```
53      * @param b an object to be compared with {@code a} for equality
54      * @return {@code true} if the arguments are equal to each other;
55      *         {@code false} otherwise
56      * @see Object#equals(Object)
57      */
58      public static boolean equals(Object a, Object b) {
59          return (a == b) || (a != null && a.equals(b));
60      }
61
62      public static boolean deepEquals(Object a, Object b) {
63          if (a == b)
64              return true;
65          else if (a == null || b == null)
66              return false; // 数组对象的相等判断
67          else
68              return Arrays.deepEquals0(a, b);
69      }
70
71      /**
72       * Returns the hash code of a non-{@code null} argument and
73       * a {@code null} argument.
74       *
75       * @param o an object
76       * @return the hash code of a non-{@code null} argument and
77       *         a {@code null} argument
78       * @see Object#hashCode
79       */
80      public static int hashCode(Object o) { return o != null ? o.hashCode() : 0; }
```

```
4294      return true;
4295
4296      static boolean deepEquals0(Object e1, Object e2) {
4297          assert e1 != null; // 对各种类型数组的判断
4298          boolean eq;
4299          if (e1 instanceof Object[] && e2 instanceof Object[])
4300              eq = deepEquals((Object[]) e1, (Object[]) e2);
4301          else if (e1 instanceof byte[] && e2 instanceof byte[])
4302              eq = equals((byte[]) e1, (byte[]) e2);
4303          else if (e1 instanceof short[] && e2 instanceof short[])
4304              eq = equals((short[]) e1, (short[]) e2);
4305          else if (e1 instanceof int[] && e2 instanceof int[])
4306              eq = equals((int[]) e1, (int[]) e2);
4307          else if (e1 instanceof long[] && e2 instanceof long[])
4308              eq = equals((long[]) e1, (long[]) e2);
4309          else if (e1 instanceof char[] && e2 instanceof char[])
4310              eq = equals((char[]) e1, (char[]) e2);
4311          else if (e1 instanceof float[] && e2 instanceof float[])
4312              eq = equals((float[]) e1, (float[]) e2);
4313          else if (e1 instanceof double[] && e2 instanceof double[])
4314              eq = equals((double[]) e1, (double[]) e2);
4315          else if (e1 instanceof boolean[] && e2 instanceof boolean[])
4316              eq = equals((boolean[]) e1, (boolean[]) e2);
4317          else
4318              eq = e1.equals(e2);
4319          return eq;
4320      }
```

从源码中，可以看出 Objects 对基本类型和复杂类型的对象，都有着比较细粒度的判断，可以放心使用。

4.2 为空判断

Search for: null

Objects.java

Show inherited members (^F12)

Show Lambdas (%L)

Show Anonymous

Objects

- isNull(Object): boolean
- nonNull(Object): boolean
- requireNonNull(T): T
- requireNonNull(T, String): T
- requireNonNull(T, Supplier<String>): T

关于判空的一些方法

Objects 提供了各种关于空的一些判断，isNull 和 nonNull 对于对象是否为空返回 Boolean 值，requireNonNull 方法更加严格，如果一旦为空，会直接抛出异常，我们需要根据生活的场景选择使用。

5 面试题

5.1 工作中有没有遇到特别好用的工具类，如何写好一个工具类

答：有的，像 Arrays 的排序、二分查找、Collections 的不可变、线程安全集合类、Objects 的判空相等判断等等工具类，好的工具类肯定很好用，比如说使用 static final 关键字对方法进行修饰，工具类构造器必须是私有等等手段来写好工具类。

5.2 写一个二分查找算法的实现

答：可以参考 Arrays 的 binarySearch 方法的源码实现。

5.3 如果我希望 ArrayList 初始化之后，不能被修改，该怎么办

答：可以使用 Collections 的 unmodifiableList 的方法，该方法会返回一个不能被修改的内部类集合，这些集合类只开放查询的方法，对于调用修改集合的方法会直接抛出异常。

总结

从三大工具类中，我们不仅学习到了如何写好一个工具类，还熟悉了三大工具类的具体使用姿势，甚至了解了其底层的源码实现，有兴趣的话，可以自己也可以仿照写个好用的工具类加深学

精选留言 17

欢迎在这里发表留言，作者筛选后可公开显示

夜很殇37

binarySearch方法，为什么要先排序，如果不先排序，搜索不到数据？
List list = ImmutableList.of(new SortDTO("300"), new SortDTO("50"), new SortDTO("200"), new SortDTO("220"));
SortDTO[] array = new SortDTO[list.size()];
list.toArray(array);
System.out.println(JSON.toJSONString(array));
int index = Arrays

👍 0 回复

2019-11-21

qq_慕哥2339582 回复 夜很殇37

二分法的时间度为O（logn），也就是每次查找都能在数组中排除一半的数据（要么是比它大的，要么是比它小的），如果无序的话，就不能保证排除的数据一定是比它大的或比它小

回复2019-11-22 12:50:44

文贺 回复 夜很殇37

是的，下面这个同学已经完美的回答你了哈，二分首选要排序，不然二分没有意义。

回复2019-11-23 16:39:01

java_simon

github的地址有谁知道

👍 0 回复

2019-10-29

初一 回复 java_simon

源码解析：<https://github.com/luanqiu/java8> 文章 demo：https://github.com/luanqiu/java8_demo

回复2019-10-29 14:06:51

慕运维9394156

ImmutableList的作用是什么呢？

👍 0 回复

2019-10-24

文贺 回复 慕运维9394156

一个工具类，方便我们使用集合，源码中贴出了官网介绍地址：<https://github.com/google/guava/wiki/ImmutableCollectionsExplained>

回复2019-10-28 12:49:03

代夫阿普曼

慕课专栏

面试官系统精讲Java源码及大厂真题 / 04 Arrays、Collections、Objects 常用方法源码解析

目录

xiaobaicaiss 回复 代夫阿普曼

Vector都已经不用了，可以用Collections.synchronizedList(new ArrayList())或者用CopyOnWriteArrayList,都是线程安全的

回复2019-11-21 09:27:46

萌萌萌唬 回复 xiaobaicaiss

为啥不用vector啊？

回复2019-11-22 18:03:18

所相虚妄 回复 萌萌萌唬

效率不行，效率和那个synchronizedlist一样，

回复9天前

swim0

你好老师，上面sort例子里面这个符号 “ :: ” 有什么用？

0

回复2019-10-22

文贺 回复 swim0

Lambda 表达式的一种写法，DTO::getA 的效果和 DTO.getA() 一样。

回复2019-10-24 20:50:48

幕布斯0011243

订阅了怎么只能看部分内容？

0

回复2019-10-09

文贺 回复 幕布斯0011243

同学你好，应该可以看到全部哈，你现在还只能看到部分么，如果还是的话，可以联系下慕课网的小姐姐问一问。

回复2019-10-10 23:00:05

为了angular耻辱上线

int mid = (low + high) >>> 1; 这一句不是很明白，为什么就可以确定中位数啊.....这个位运算为什么就不需要判断数组的奇偶了？

1

回复2019-10-04

文贺 回复 为了angular耻辱上线

System.out.println("(0 + 10-1) >>> 1;" + ((0 + 10-1) >>> 1)); System.out.println("(0 + 11-1) >>> 1;" + ((0 + 11-1) >>> 1)); System.out.println("(1 + 11-1) >>> 1;" + ((1 + 11-1) >>> 1)); System.out.println("(1 + 10-1) >>> 1;" + ((1 + 10-1) >>> 1)); 二进制位移是不需要判断奇数偶数的哈，这个不是除法哈。

回复2019-10-08 19:29:19

qq_13270 回复 为了angular耻辱上线

右移一位不就是除于2吗

回复2019-11-21 23:23:33

https://www.imooc.com/read/47/article/846

10/12

目录

风舞炫动

写泛型全参照源码，源码里泛型用的是真6

👍 0

回复

2019-09-26

文贺 回复 风舞炫动

同学你好，说的很有道理的。

回复

2019-09-27 13:05:23

慕斯卡6586063

尽量不在工具方法中，对共享变量有做修改的操作访问（如果必须要做的话，必须加锁），因为会有线程安全的问题。除此之外，工具类方法本身是没有线程安全问题的，可以放心使用这一段讲得有点模糊，请老师再说明一下

👍 0

回复

2019-09-20

文贺 回复 慕斯卡6586063

```
public static final String[] array = new String [3]; public static final void doSomething(String params1){ array[0] = params1; }
```

简单举个例子，在工具方法 doSomething 里面对共享变量 array 进行了操作，并发情况下会有问题。

回复

2019-09-20 14:01:56

qq_Ezio_1

老师。目前大三要找工作，题型怎么刷才好，在哪里找题库比较好。希望老师能给予些建议

👍 0

回复

2019-09-18

文贺 回复 qq_Ezio_1

知乎上可以搜索到大厂面试题，比如搜索阿里面试题，就会出现最新的面试题，另外说一句，刷面试题全靠运气，大厂每年的面试题都和往年不一样，面试的几个关键要素： 1. 运气，问到的都是你会的； 2. 有所准备，事先准备几个知识点，是自己深入研究过的，这样可以突出亮点； 3. 注意知识点之间的关联，大多人学习的知识都是单点，想孤岛一样，如果你能把知识点关联起来，无疑比大多数人更深入了一步。最后，相信自己，我也相信你可以的。

回复

2019-09-18 21:32:26

Super丶Lee

二分搜索的那里 泛型 后面为什么还有个int 这是什么意思？

👍 0

回复

2019-09-11

文贺 回复 Super丶Lee

int 表示 key 在所搜索的数组下标位置，是方法的返回类型，泛型主要是为了表示数组元素的类

点击展开剩余评论

目录