# 如何高效地判断数组中是否包含某特定值

如何检查一个未排序的数组中是否包含某个特定值，这是一个在Java中非常实用并且频繁使用的操作。另外，这也是Stack Overflow上面非常受关注的问题。在得票数最多的答案中，可以看到，检查数组中是否包含特定值可以用多种不同的方式实现，但是时间复杂度差别很大。下面，我将为大家展示各种方法及其需要花费的时间。

**1.检查数组中是否包含特定值的四种不同方法**

1）使用List：

|  |  |
| --- | --- |
| 1  2  3 | public static boolean useList(String[] arr, String targetValue) {      return Arrays.asList(arr).contains(targetValue);  } |

2）使用Set：

|  |  |
| --- | --- |
| 1  2  3  4 | public static boolean useSet(String[] arr, String targetValue) {      Set<String> set = new HashSet<String>(Arrays.asList(arr));      return set.contains(targetValue);  } |

3）使用一个简单循环：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public static boolean useLoop(String[] arr, String targetValue) {      for(String s: arr){          if(s.equals(targetValue))              return true;      }      return false;  } |

4）使用Arrays.binarySearch()：

注：下面的代码是错误的，这样写出来仅仅为了理解方便。binarySearch()只能用于已排好序的数组中。所以，你会发现下面结果很奇怪。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public static boolean useArraysBinarySearch(String[] arr, String targetValue) {      int a =  Arrays.binarySearch(arr, targetValue);      if(a > 0)          return true;      else          return false;  } |

**2.时间复杂度**

通过下面的这段代码可以近似比较几个方法的时间复杂度。虽然分别搜索一个大小为5、1K、10K的数组是不够精确的，但是思路是清晰的。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | public static void main(String[] args) {      String[] arr = new String[] {  "CD",  "BC", "EF", "DE", "AB"};        //use list      long startTime = System.nanoTime();      for (int i = 0; i < 100000; i++) {          useList(arr, "A");      }      long endTime = System.nanoTime();      long duration = endTime - startTime;      System.out.println("useList:  " + duration / 1000000);        //use set      startTime = System.nanoTime();      for (int i = 0; i < 100000; i++) {          useSet(arr, "A");      }      endTime = System.nanoTime();      duration = endTime - startTime;      System.out.println("useSet:  " + duration / 1000000);        //use loop      startTime = System.nanoTime();      for (int i = 0; i < 100000; i++) {          useLoop(arr, "A");      }      endTime = System.nanoTime();      duration = endTime - startTime;      System.out.println("useLoop:  " + duration / 1000000);        //use Arrays.binarySearch()      startTime = System.nanoTime();      for (int i = 0; i < 100000; i++) {          useArraysBinarySearch(arr, "A");      }      endTime = System.nanoTime();      duration = endTime - startTime;      System.out.println("useArrayBinary:  " + duration / 1000000);  } |

结果：

|  |  |
| --- | --- |
| 1  2  3  4 | useList:  13  useSet:  72  useLoop:  5  useArraysBinarySearch:  9 |

对于长度为1K的数组：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | String[] arr = new String[1000];    Random s = new Random();  for(int i=0; i< 1000; i++){      arr[i] = String.valueOf(s.nextInt());  } |

结果：

|  |  |
| --- | --- |
| 1  2  3  4 | useList:  112  useSet:  2055  useLoop:  99  useArrayBinary:  12 |

对于长度为10K的数组：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | String[] arr = new String[10000];    Random s = new Random();  for(int i=0; i< 10000; i++){      arr[i] = String.valueOf(s.nextInt());  } |

结果：

|  |  |
| --- | --- |
| 1  2  3  4 | useList:  1590  useSet:  23819  useLoop:  1526  useArrayBinary:  12 |

很明显，使用简单循环的方法比使用其他任何集合效率更高。许多开发者会使用第一种方法，但是它并不是高效的。将数组压入Collection类型中，需要首先将数组元素遍历一遍，然后再使用集合类做其他操作。

如果使用Arrays.binarySearch()方法，数组必须是已排序的。由于上面的数组并没有进行排序，所以该方法不可使用。

实际上，如果你需要借助数组或者集合类高效地检查数组中是否包含特定值，一个已排序的列表或树可以做到时间复杂度为O(log(n))，hashset可以达到O(1)。