**Java排序算法（九）：归并排序**

原创 2011年07月08日 00:25:41

* 标签：
* [算法](http://so.csdn.net/so/search/s.do?q=%E7%AE%97%E6%B3%95&t=blog) /
* [java](http://so.csdn.net/so/search/s.do?q=java&t=blog) /
* [merge](http://so.csdn.net/so/search/s.do?q=merge&t=blog) /
* [string](http://so.csdn.net/so/search/s.do?q=string&t=blog) /
* [class](http://so.csdn.net/so/search/s.do?q=class&t=blog) /
* [工作](http://so.csdn.net/so/search/s.do?q=%E5%B7%A5%E4%BD%9C&t=blog)
* 44960

 Java排序算法（九）：归并排序

**归并排序**（Merge）是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。

归并排序是建立在归并操作上的一种有效的排序算法。该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。 将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为2-路归并。

归并排序算法稳定，数组需要O(n)的额外空间，链表需要O(log(n))的额外空间，时间复杂度为O(nlog(n))，算法不是自适应的，不需要对数据的随机读取。

**工作原理：**

1、申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列

2、设定两个指针，最初位置分别为两个已经排序序列的起始位置

3、比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置

4、重复步骤3直到某一指针达到序列尾

5、将另一序列剩下的所有元素直接复制到合并序列尾

**代码实现：**

**[java]** [view plain](http://blog.csdn.net/apei830/article/details/6591632) [copy](http://blog.csdn.net/apei830/article/details/6591632)

1. **public** **class** MergeSortTest {
3. **public** **static** **void** main(String[] args) {
4. **int**[] data = **new** **int**[] { 5, 3, 6, 2, 1, 9, 4, 8, 7 };
5. print(data);
6. mergeSort(data);
7. System.out.println("排序后的数组：");
8. print(data);
9. }
11. **public** **static** **void** mergeSort(**int**[] data) {
12. sort(data, 0, data.length - 1);
13. }
15. **public** **static** **void** sort(**int**[] data, **int** left, **int** right) {
16. **if** (left >= right)
17. **return**;
18. // 找出中间索引
19. **int** center = (left + right) / 2;
20. // 对左边数组进行递归
21. sort(data, left, center);
22. // 对右边数组进行递归
23. sort(data, center + 1, right);
24. // 合并
25. merge(data, left, center, right);
26. print(data);
27. }
29. /\*\*
30. \* 将两个数组进行归并，归并前面2个数组已有序，归并后依然有序
31. \*
32. \* @param data
33. \*            数组对象
34. \* @param left
35. \*            左数组的第一个元素的索引
36. \* @param center
37. \*            左数组的最后一个元素的索引，center+1是右数组第一个元素的索引
38. \* @param right
39. \*            右数组最后一个元素的索引
40. \*/
41. **public** **static** **void** merge(**int**[] data, **int** left, **int** center, **int** right) {
42. // 临时数组
43. **int**[] tmpArr = **new** **int**[data.length];
44. // 右数组第一个元素索引
45. **int** mid = center + 1;
46. // third 记录临时数组的索引
47. **int** third = left;
48. // 缓存左数组第一个元素的索引
49. **int** tmp = left;
50. **while** (left <= center && mid <= right) {
51. // 从两个数组中取出最小的放入临时数组
52. **if** (data[left] <= data[mid]) {
53. tmpArr[third++] = data[left++];
54. } **else** {
55. tmpArr[third++] = data[mid++];
56. }
57. }
58. // 剩余部分依次放入临时数组（实际上两个while只会执行其中一个）
59. **while** (mid <= right) {
60. tmpArr[third++] = data[mid++];
61. }
62. **while** (left <= center) {
63. tmpArr[third++] = data[left++];
64. }
65. // 将临时数组中的内容拷贝回原数组中
66. // （原left-right范围的内容被复制回原数组）
67. **while** (tmp <= right) {
68. data[tmp] = tmpArr[tmp++];
69. }
70. }
72. **public** **static** **void** print(**int**[] data) {
73. **for** (**int** i = 0; i < data.length; i++) {
74. System.out.print(data[i] + "\t");
75. }
76. System.out.println();
77. }
79. }