字符串翻转：翻转字符串里的单词

题目来源：Leetcode 151：https://leetcode-cn.com/problems/reverse-words-in-a-string/

暴力解法：使用语言特性实现思路一

把单词看成整体，切割之后翻转单词的顺序

Java代码

```
/**
 * 解法一：暴力解法 使用语言特性实现思路一
 * 1.将字符串按空格切割成单词数组
 * 2.翻转单词顺序
 *  使用数组工具类转成集合
 *  使用集合工具类进行翻转
 * 3.重新将单词与空格拼接成新字符串
 *  使用String类的静态方法join进行拼接
 *
 * 时间复杂度：O(n)
 *  切割过程进行遍历查找：O(n)
 *  翻转与拼接：O(n) + O(n)
 * 空间复杂度：O(n)
 *  切割使用了2个数组：O(n)
 *  join使用了1个数组：O(n)
 *
 * 执行耗时:7 ms,击败了46.81% 的Java用户
 * 内存消耗:39.3 MB,击败了20.75% 的Java用户
 *
 * @param s
 * @return
 */
public String reverseWords(String s) {
    if (s == null || "".equals(s = s.trim()))
        return "";
    // 细节：正则匹配多个空格
    // 细节：数组、集合工具类的使用
    // 1.将字符串按空格切割成单词数组
    String[] strings = s.split("\\s+");
    // 2.翻转单词顺序
    List<String> list = Arrays.asList(strings);
    Collections.reverse(list);
    // 3.重新将单词与空格拼接成字符串
    return String.join(" ", list);
}
```

优化解法：数组+双指针实现思路二

先翻转字符串的顺序，再翻转单词字母的顺序。

java代码

```java
/**
 * 解法二：数组+双指针实现思路二
 * 1.按字符串长度定义新数组，临时存储
 * 2.倒序遍历字符串，定位单词起止索引
 * 3.读取单词起止索引范围内的字符，写入新数组
 * 4.还原指针，用以定位下个单词
 * 5.将新数组中合法数据生成新字符串
 *
 * 边界问题
 *  以字符串中的空格为单词分界
 *  字符串首尾的空格应跳过
 * 细节问题
 *  倒序遍历时，先定义单词尾指针
 *  读取到下一个空格，索引+1定位单词开始指针
 *  注意单词间的多个空格，只保留一个
 *
 * 时间复杂度：O(n)
 *  倒序遍历字符串：O(n)
 *  读取所有单词：O(n)
 *
 * 空间复杂度：O(n)
 *  需要一个临时数组：O(n)
 *  两个指针：O(2)
 *  最后重新生成一个数组：O(n)
 *
 * 执行耗时:3 ms,击败了75.57% 的Java用户
 * 内存消耗:39.1 MB,击败了43.42% 的Java用户
 *
 * @param s
 * @return
 */
public String reverseWords(String s) {
    int len;
    if (s == null || (len = s.length()) == 0)
        return "";
    // 1.准备工作：初始化新数组，定义单词起止索引
    char[] chars = new char[len]; // 新字符数组
    int first = -1, last = -1, index = 0; // 单词起止索引
    // 2.倒序遍历字符串，定位单词起止索引
    for (int i = len - 1; i >= 0; i--) {
        char c = s.charAt(i);
        if (c != ' ') { // 非空格：第一个非空格为单词结尾字符
            if (last == -1) last = i; // 2.1. 定位last
            if (i == 0) first = i; // 细节：处理字符串首字符不是空格
        } else { // 空格：以“空格+1”为单词开始索引
            if (last != -1) first = i + 1; // 2.2.定位first
        }

        // 3.读取单词起止索引范围内的字符，写入新数组
        if (first >= 0 && last >= 0 ) {
            // 细节：如果新数组中已经有数据，先存放一个空格，再放数据
            if (index > 0) chars[index++] = ' ';
            while (first <= last) {
                chars[index++] = s.charAt(first);
                first++;
```

```
                }
                // 4.还原指针，用以定位下个单词
                first = last = -1;
            }
        }
        // 5.将新数组中合法数据生成新字符串返回
        return String.valueOf(chars, 0, index);
    }
```

优化解法：双端队列实现思路一

把单词依次从双端队列头部插入，再从双端队列头部取出。

java代码

```
    /**
     * 解法三：双端队列解法思路分析
     * 1.往双端队列头部依次存入每个单词
     *  以空格为单词分界，将单词字符存入缓冲区
     *  从缓冲区取出单词存入双端队列头部
     *  注意过滤掉首尾、单词间多余空格
     * 2.从双端队列头部依次取出每个单词
     *  使用join方法，将空格拼接到每个单词之间
     *  注意不要遗漏最后一个单词
     *
     * 时间复杂度：O(n)
     *  遍历字符串：O(n)
     *  读取所有单词：O(n)
     *  双端队列扩容：O(n)
     *
     * 空间复杂度：O(n)
     *  需要一个双端队列：O(n)
     *  一个字符串缓冲区：O(n)
     *  最后重新生成一个数组：O(n)
     *
     * 执行耗时:7 ms,击败了46.81% 的Java用户
     * 内存消耗:38.7 MB,击败了93.19% 的Java用户
     *
     * @param s
     * @return
     */
    public String reverseWords(String s) {
        int left = 0, right = s.length() - 1;
        Deque<String> d = new ArrayDeque<String>();
        StringBuilder word = new StringBuilder();

        while (left <= right) {
            char c = s.charAt(left);
            if (c != ' ') {
                word.append(c);
            } else {
                if (word.length() != 0) { // 到达了单词结尾
                    // 将单词添加到队列的头部
                    d.offerFirst(word.toString());
                    word.setLength(0);
```

```
                }
            }
            ++left;
        }
        if (word.length() > 0) {
            d.offerFirst(word.toString());
        }
        return String.join(" ", d);
    }
```

最优解：切割+反向遍历解法

使用 `String` 类的 `split` 方法，按空格切割，反向遍历结果数组，然后拼接成新串返回。

java代码

```
    /**
     * 最优解：切割+反向遍历
     * 1.将字符串按空格进行切割
     *  按单个字符"空格"切割，降低处理复杂度
     *  切割结果是一个字符串数组，可能包含""
     * 2.反向遍历数组中的每个单词
     * 3.将每个单词存入字符串缓冲区中
     *  存入前加空格作为前缀
     *  遍历完成后进行截取
     *
     * 时间复杂度：O(n)
     *  生成数组过程遍历字符串：O(n)
     *  读取所有单词：O(n)
     * 空间复杂度：O(n)
     *  生成一个数组：O(n)
     *  一个字符串缓冲区：O(n)
     *  最后重新生成一个数组：O(n)
     *
     * 执行耗时:1 ms,击败了99.99% 的Java用户
     * 内存消耗:38.6 MB,击败了97.51% 的Java用户
     *
     * @param s
     * @return
     */
    public String reverseWords(String s) {
        if (s == null || "".equals(s = s.trim()))
            return "";
        String[] strs = s.split(" ");
        StringBuilder sb = new StringBuilder();

        for (int i = strs.length - 1; i >= 0; i--) {
            if (strs[i].length() != 0)
                sb.append(" ").append(strs[i]);
        }
        return sb.substring(1);
    }
```

C++代码

```cpp
#include <iostream>
#include <string.h>
#include <string>
#include <algorithm>

using namespace std;
class Solution {
public:
    string reverseWords(string s) {
        int i = 0, size = s.size();
        string arr[size];
        char *token = strtok(s.data(), " ");
        while (token != NULL) {
            arr[i++] = token;
            token = strtok(NULL, " ");
        }
        string result = "";
        for (i = size - 1; i >= 0; i--) {
            if (arr[i] != "") {
                result += arr[i] + " ";
            }
        }
        return result.substr(0, result.size() - 1);
    }
}
```

Python代码

```python
'''
执行用时：36 ms, 在所有 Python3 提交中击败了 90.32% 的用户
内存消耗：13.5 MB, 在所有 Python3 提交中击败了 59% 的用户
'''
class Solution:
    def reverseWords(self, s: str) -> str:
        if s is None or len(s) == 0:
            return None
        # 按空格进行切割，得到包含空格的多个子串
        strs = s.split(' ')
        # 过滤掉多余空格，临时存放到新的列表
        strArr = []
        for ss in strs:
            if not ss.isspace() and len(ss) > 0:
                strArr.append(ss)
        # 翻转新的列表
        strArr.reverse()
        # 使用空格拼接到每个元素
        white = ' '
        return white.join(strArr)
```

测试用例

```
输入: "the sky is blue"
输出: "blue is sky the"
```

```
输入: "  hello world!  "
输出: "world! hello"
解释: 输入字符串前面或者后面包含多余的空格，但是反转后的字符不能包括
```

```
输入: "a good        example"
输出: "example good a"
解释: 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个
```