学习目标

字符串本身不是一种数据结构，但是由于其本身的特殊性，可以产生很多特殊的算法题。另外，字符串在工程里也有非常广泛的应用，因此一直都是算法考察的重点问题之一，我们有必要认真研究一下相关的问题。另外，JDK，以及guava、apache-commons等都提供了很多字符串操作方法，我们要熟悉这些方法是如何用的，并且要注意思考自己如何实现这些方法。目前找到一个python版本比较信息的方法列表https://docs.python.org/zh-cn/3/library/stdtypes.html#string-methods，读者可以思考一下如何基于Java实现这些功能。

学习的过程中要注意以下几个问题：

1.字符串的特性以Java如何管理字符串的

2.了解jdk等提供的内置处理方法，并且能够自己给出实现思路。

3.认真研究《2.1.2》中字符串转整数的问题。

4.掌握其他常见问题的解决思路和实现方法。

1.4 java中的字符串

字符串本身不是一种数据结构，但是由于其本身的特殊性，可以产生一些特定的算法题。而C、java等语言创建和管理字符串的方式也都有差异，因此针对语言特征又产生了很多问题，这些问题可以做为算法面试，也可以作为技术面试的一部分，例如String、StringBuilder、StringBuffer等，所以我们有必要认真梳理一下字符串相关的问题。

字符串是由多个字符组成的一串数据，jdk提供的字符串类主要有三个：String、StringBuffer和StringBuilder。主要区别是StringBuffer是线程安全的，StringBuilder是线程不安全的。在写算法时如果面试官没提，不要擅自加多线程的问题，因为你能写出来本身已经很不容易了了。除非你已经写完了，想来个锦上添花，或者是一个设计类型的题目。

因为相对简洁，所以在手写算法中String类更为常见，其特点是：

- 字符串是常量，一旦被创建就不能改变，这是因为字符串的值是存放在方法区的常量池里面，但是引用可以改变。
- 字符串字面值"ab"也可以看成是一个字符串对象。

```
String s=new String("student")
```

上面这个特点是什么意思呢？我们先看一下String类的定义：

```
public final class String implements java.io.Serializable, Comparable<String>,
CharSequence
    /** The value is used for character storage. */
    private final char value[];
```

这个定义挺复杂，实现了多个基础接口，这也说明String在使用的时候已经具备这些功能，

我们可以看到：

- String是一个final类，不能被继承的类
- String类实现了java.io.Serializable接口，可以实现序列化
- String类实现了Comparable，可以用于比较大小（按顺序比较单个字符的ASCII码）
- String类实现了 CharSequence 接口，表示是一个有序字符的序列，因为String的本质是一个char类型数组。
- String的内部真正存储数据的其实就是一个char类型的数组。

String类提供了大量的方法来辅助开发，我们应该对其有个眼熟，这样很多算法题就能直接拿来用了。我们分类看一下：

1.常见String类的获取功能

- public int length(): 获取字符串的长度。
- public char charAt(int index): 获取指定索引位置的字符
- public int indexOf(int ch): 返回指定字符在此字符串中第一次出现处的索引。
- public int indexOf(String str): 返回指定字符串在此字符串中第一次出现处的索引。
- public int indexOf(int ch,int fromIndex):返回指定字符在此字符串中从指定位置后第一次出现处的索引。
- public int indexOf(String str,int fromIndex): 返回指定字符串在此字符串中从指定位置后第一次出现处的索引。

- public String substring(int start): 从指定位置开始截取字符串,默认到末尾。
- public String substring(int start,int end): 从指定位置开始到指定位置结束截取字符串。

测试代码：

```
public class StringTest1 {
    public static void main(String[] args) {
        String str = "anAdEfg";

        System.out.println("String类的获取功能");
        //获取字符串的长度
        int length = str.length();
        //获取指定索引位置的字符
        char c1 = str.charAt(0);
        //返回指定字符在此字符串中第一次出现处的索引
        int c2 = str.indexOf('n');
        //返回指定字符串在此字符串中第一次出现的索引
        int c3 = str.indexOf("fg");
        //返回指定字符在此字符串中从指定位置后第一次出现处的索引。
        int c4= str.indexOf('f', 2);
        //返回指定字符串在此字符串中从指定位置后第一次出现处的索引。
        int c5 = str.indexOf("fg", 2);
        //从指定位置开始截取字符串，默认到末尾
        String c6 = str.substring(2);
        //从指定位置开始到指定位置结束截取字符串
        String c7 = str.substring(2, 4);
        System.out.println(length);
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
        System.out.println(c4);
        System.out.println(c5);
        System.out.println(c6);
        System.out.println(c7);
      }
    }
```

运行结果:

```
7
a
1
5
5
5
AdEfg
Ad
```

2.常见String类的判断功能

- public boolean equals(Object obj): 比较字符串的内容是否相同,区分大小写
- public boolean equalsIgnoreCase(String str): 比较字符串的内容是否相同,忽略大小写
- public boolean contains(String str): 判断字符串中是否包含传递进来的字符串
- public boolean startsWith(String str): 判断字符串是否以传递进来的字符串开头
- public boolean endsWith(String str): 判断字符串是否以传递进来的字符串结尾
- public boolean isEmpty(): 判断字符串的内容是否为空串""。

写个例子看看：

```
public class StringTest2 {
    public static void main(String[] args) {
       String str1 = "axcde";
        String str2 = "Axcde";
        System.out.println("String类的判断功能");
        //比较字符串的内容是否相同，区分大小写
        boolean b = str1.equals(str2);
        //比较字符串的内容是否相同,忽略大小写
        boolean b1 = str1.equalsIgnoreCase(str2);
        //判断字符串中是否包含传递进来的字符串
        boolean b2 = str1.contains("cde");
        //判断字符串是否以传递进来的字符串开头
        boolean b3 = str1.startsWith("ax");
        //判断字符串是否以传递进来的字符出结尾
        boolean b4 = str2.endsWith("de");
        //判断字符串的内容是否为空
        boolean b5 = str1.isEmpty();
        System.out.println(b);
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);
        System.out.println(b5);
    }
  }
```

运行结果为：

```
String类的判断功能
false
true
true
true
true
false
```

3.常见String类的转换功能

- public byte[] getBytes(): 把字符串转换为字节数组。
- public char[] toCharArray(): 把字符串转换为字符数组。

- public static String valueOf(char[] chs): 把字符数组转成字符串。
- public static String valueOf(int i): 把int类型的数据转成字符串。（String类的valueOf方法可以把任意类型的数据转成字符串。）
- public String toLowerCase(): 把字符串转成小写。
- public String toUpperCase(): 把字符串转成大写。
- public String concat(String str): 把字符串拼接。

```
public class StringTest3 {
    public static void main(String[] args) {
       String str = "anAdEfg";
        String str1 = "axcde";
        int a=123323;
        //把字符串转换为字节数组
        byte[] bytes = str.getBytes();
        for (int i = 0; i < bytes.length; i++) {
            System.out.print(bytes[i]+" ");
        }
        //把字符串转换为字符数组
        char[] chars = str.toCharArray();
        for (int i = 0; i < chars.length; i++) {
            System.out.print(chars[i]+" ");
        }
        System.out.println();
        //把字符数组转换成字符串
        String s1 = new String (chars);
        //把int类型的数据转成字符串
        String s2 = Integer.toString(a);
        //把字符串转换成小写
        String s = str.toLowerCase();
        //把字符串变成大写
        String s3 = str.toUpperCase();
        //字符串拼接
        String s4 = str.concat(str1);
        System.out.println(s);
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
    }
}
```

运行结果为：

```
97 110 65 100 69 102 103 a n A d E f g
anadefg
anAdEfg
123323
ANADEFG
anAdEfgaxcde
```

4.常见String类的其他常用功能

- public String replace(char old,char new) 将指定字符进行互换
- public String replace(String old,String new) 将指定字符串进行互换
- public String trim() 去除两端空格
- public int compareTo(String str) 会对照ASCII 码表 从第一个字母进行减法运算 返回的就是这个减法的结果，如果前面几个字母一样会根据两个字符串的长度进行减法运算返回的就是这个减法的结果，如果连个字符串一模一样 返回的就是0
- public int compareToIgnoreCase(String str) 跟上面一样 只是忽略大小写的比较。

```
public class StringTest4 {
    public static void main(String[] args) {
        String str = "  anAdEfg  ";
        String str1 = "axcde";

        //将指定字符进行互换
        String s = str.replace('a', 'b');
        System.out.println(s);
        //将指定字符串进行互换
        String s1 = str.replace("ab", "qq");
        System.out.println(s1);
        //去除两端空格
        String s2 = str.trim();
        System.out.println(s2);
        //通过字典顺序去比较 返回的值是 ASCII 码的差值  调用者减去传入者
        int i = "abc".compareTo("ABc");
        System.out.println(i);
        //如果前面几个字母一样会根据两个字符串长度进行减法运算返回该减法的结果 （通过长度去比）
        int i1 = "abc".compareTo("a");
        System.out.println(i1);
        //忽略大小写的比较
        int i2 = "abc".compareToIgnoreCase("ABC");
        System.out.println(i2);
    }
}
```

运行结果为：

```
  bnAdEfg
  anAdEfg
anAdEfg
32
2
0
```

5.int与String的相互转换

int转String有三种方式

- num + “ ”
- String.valueOf(num)
- Integer.toString(num)

String转int有两种方式

- Integer.parseInt(str)
- Integer.valueOf(str).intValue()

上代码看一下

```
public class StringTest5 {
    public static void main(String[] args) {
        //int->String
        int i = 1234567;
        String s = "";
        s = i + "";//会产生两个String对象
        System.out.println(s);
        String s1 = String.valueOf(i);//使用String类的静态方法，只产生一个String对象
        System.out.println(s1);
        String s2 = Integer.toString(i);
        System.out.println(s2);
        //String->int
        String str = "12345";
        int i1 = Integer.parseInt(str);//直接使用静态方法，不会产生多余的对象
        System.out.println(i1);
        int i2 = Integer.valueOf(str).intValue();
        System.out.println(i2);
    }
}
```

运行结果为：

```
1234567
1234567
1234567
12345
12345
```

6.注意==和equals（）的区别

上代码看一下：

```
public class StringTest6 {
    public static void main(String[] args) {
        String s1 = new String("hello");
        String s2 = new String("hello");
        System.out.println(s1.equals(s2));//判断字符串内容
        System.out.println(s1 == s2);//判断字符串引用
    }
}
```

运行结果为：

```
true
false
```

7.Java中的charAt()方法

charAt()是字符串算法中一个非常重要的方法，我们再看一下。java.lang.String.charAt() 方法返回指定索引处的char值。索引范围是从0到length() - 1。
声明java.lang.String.charAt()方法如下：

- public char charAt(int index)
  在字符串问题中，经常要判断字符是不是字母或者数字，这里看一下基本的代码。

```
public class StringTest7 {
   public static void main(String[] args) {
        //统计字符串中的小写字母，大写字母数字空格的个数
        //统计字符串中的小写字母，大写字母数字空格的个数
        String str = "1123ahdiASDFGF    shaid";
        int upper = 0;
        int lower = 0;
        int num = 0;
        int space = 0;
        for (int i = 0; i < str.length(); i++) {
            //返回该索引处的char值
            char c = str.charAt(i);
            System.out.print(c+" ");
            if(c>= 'a' && c<='z'){
                lower++;
            }
            if(c>='A'&&c<= 'Z'){
                upper++;
            }
            if(c>='0'&&c<='9'){
                num++;
            }
```

```
            if(c==' '){
                space++;
            }
        }
        System.out.println();
        System.out.println(lower);
        System.out.println(upper);
        System.out.println(num);
        System.out.println(space);
    }
}
```

运行结果为：

```
1 1 2 3 a h d i A S D F G F        s h a i d
9
6
4
4
```

拓展

在面试的技术问题中，经常出现String, StringBuilder, StringBuffer的区别、实现原理等问题，以及jvm是如何管理字符串的，这些问题在jvm课程中是很重要的一部分，我们这里不再展开，但是面试之前一定要学学啊。

2. 字符串高频算法题

从上面可以看到字符串与数组有很多相似之处，比如使用 `名称[下标]` 来得到一个字符，很多字符串的题本质就是数组的题，不过我们需要先将字符串转换成数组， 处理完之后再转成字符串

字符串最经典的算法问题是子串匹配问题， 但是这个算法有些难度，我们后面单独看，这里我们先研究一些关于字符串的常见算法问题。

2.1 转换的问题

字符串里存放的可以是字母，可以是数字，也可以是特殊字符，字母又可以大写和小写，这就导致字符串有一类常见的转换的题目，这些题目无非就是这几种类型的相互转换。但是在转换过程中需要处理几种特殊情况：例如首先就是转之前先判断当前元素能不能转。如果是字符串转数字，则要考虑当前元素是不是数字。转完之后会不会溢出等。这些问题本身不复杂，但是必须考虑周全 ，如果考虑不周，就是面试时的扣分点了。我们看几个题目：

2.1.1 转换成小写字母

LeetCode709. 给你一个字符串 `s` ，将该字符串中的大写字母转换成相同的小写字母，返回新的字符串。

```
示例1:
输入: s = "Hello"
输出: "hello"

示例2:
输入: s = "here"
输出: "here"

示例3:
输入: s = "LOVELY"
输出: "lovely"
```

我们知道每个字母都是有确定的 `ASCII` 的，因此我们可以根据 码表操作字符串即可。常见ASCII范围是：
a-z：97-122
A-Z：65-90
0-9：48-57

这个题可以先遍历整个字符串，然后对每一位字符进行判断，如果str[i]的值在A-Z之间，则需要在原来的基础上ASCII码加上32即可转换成对应小写：

```
    public static String toLowerCase(String s) {
        int n = s.length();
        char[] chars = s.toCharArray();
        for (int i = 0; i < n; ++i) {
            if (chars[i] >= 65 && chars[i] <= 90) {
                chars[i] += 32;
            }
        }
        String str = new String(chars);
        return str;
    }
```

具体到实现过程，还有很多种 ，我们选择自己最顺眼的一种练熟就够了，例如还可以这么写：

```
class ToLowerCase {
    public String toLowerCase(String str) {
        if (str == null || str.length() == 0) {
            return str;
        }
        StringBuilder sb = new StringBuilder();
        for (char ch : str.toCharArray()) {
            // a-z: 97-122  A-Z: 65-90  0-9: 48-57
            if (ch >= 'A' && ch <= 'Z') {
                sb.append((char)(ch + 32));
            } else {
                sb.append(ch);
            }
        }
```

```
        return sb.toString();
    }
}
```

2.1.2 字符串转换整数 (atoi)

LeetCode8. 本题的题目要求比较长，看原文：

```
请你来实现一个 myAtoi(string s) 函数，使其能将字符串转换成一个 32 位有符号整数（类似 C/C++ 中的 atoi 函数）。
函数 myAtoi(string s) 的算法如下：

* 读入字符串并丢弃无用的前导空格
* 检查下一个字符（假设还未到字符末尾）为正还是负号，读取该字符（如果有）。 确定最终结果是负数还是正数。如果两者都不存在，则假定结果为正。
* 读入下一个字符，直到到达下一个非数字字符或到达输入的结尾。字符串的其余部分将被忽略。
* 将前面步骤读入的这些数字转换为整数（即，"123" -> 123， "0032" -> 32）。如果没有读入数字，则整数为 0 。必要时更改符号（从步骤 2 开始）。
* 如果整数超过 32 位有符号整数范围 [−2^31， 2^31 − 1] ，需要截断这个整数，使其保持在这个范围内。具体来说，小于 −2^31 的整数应该被固定为 −2^31 ，大于 2^31 − 1 的整数应该被固定为 2^31 − 1 。
* 返回整数作为最终结果。

**注意：**
- 本题中的空白字符只包括空格字符 `' '` 。
- 除前导空格或数字后的其余字符串外，**请勿忽略** 任何其他字符。
```

示例：

```
示例1：
输入：s = "42"
输出：42
解释：加粗的字符串为已经读入的字符，插入符号是当前读取的字符。
第 1 步："42"（当前没有读入字符，因为没有前导空格）
         ^
第 2 步："42"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
         ^
第 3 步："42"（读入 "42"）
           ^
解析得到整数 42 。
由于 "42" 在范围 [-2^31, 2^31 - 1] 内，最终结果为 42 。
```

示例2：
输入：s = "   -42"
输出：-42
解释：
第 1 步："   -42"（读入前导空格，但忽视掉）
            ^
第 2 步："   -42"（读入 '-' 字符，所以结果应该是负数）
             ^
第 3 步："   -42"（读入 "42"）
               ^
解析得到整数 -42 。
由于 "-42" 在范围 [-2^31, 2^31 - 1] 内，最终结果为 -42 。

示例3：
输入：s = "4193 with words"
输出：4193
解释：
第 1 步："4193 with words"（当前没有读入字符，因为没有前导空格）
          ^
第 2 步："4193 with words"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
          ^
第 3 步："4193 with words"（读入 "4193"；由于下一个字符不是一个数字，所以读入停止）
              ^
解析得到整数 4193 。
由于 "4193" 在范围 [-2^31, 2^31 - 1] 内，最终结果为 4193 。

示例4：
输入：s = "words and 987"
输出：0
解释：
第 1 步："words and 987"（当前没有读入字符，因为没有前导空格）
          ^
第 2 步："words and 987"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
          ^
第 3 步："words and 987"（由于当前字符 'w' 不是一个数字，所以读入停止）
          ^
解析得到整数 0 ，因为没有读入任何数字。
由于 0 在范围 [-2^31, 2^31 - 1] 内，最终结果为 0 。

```
示例5：
输入：s = "-91283472332"
输出：-2147483648
解释：
第 1 步："-91283472332"（当前没有读入字符，因为没有前导空格）
         ^
第 2 步："-91283472332"（读入 '-' 字符，所以结果应该是负数）
          ^
第 3 步："-91283472332"（读入 "91283472332"）
                     ^
解析得到整数 -91283472332 。
由于 -91283472332 小于范围 [-2^31, 2^31 - 1] 的下界，最终结果被截断为 -2^31 = -2147483648 。
```

该题的要求很长，给的示例也很多，但是真正在面试的时候，这些要求是我们自己应该知道的，写代码也必须考虑的问题。面试官不会告诉你这些，如果你的代码考虑不周，他可能会提醒，如果提醒超过三次，你的代码就废了。

这个题最好不要用高级的特性，就是最基本的方式写。这里没有考察算法的知识，更多是开发中对数据的处理（如「参数校验」等）。如果面试中遇到，应先仔细阅读题目文字说明，认真细致的分析可能存在的情况，有疑问及时和面试官确认，千万不要阴沟里翻船。

在这里我罗列几个要点：

- 根据示例 1，需要去掉前导空格；
- 根据示例 2，需要判断第 1 个字符为 + 和 - 的情况，因此，可以设计一个变量 sign，初始化的时候为 1，如果遇到 - ，将 sign 修正为 -1；
- 判断是否是数字，可以使用字符的 ASCII 码数值进行比较，即 '0' <= c <= '9',如果0在最前面，则应该将其去掉；
- 根据示例 3 和示例 4 ，在遇到第 1 个不是数字的字符的情况下，转换停止，退出循环；
- 根据示例 5，如果转换以后的数字超过了 int 类型的范围，需要截取。这里不能将结果 res 变量设计为 long 类型，注意：由于输入的字符串转换以后也有可能超过 long 类型，因此需要在循环内部就判断是否越界，只要越界就退出循环，这样也可以减少不必要的计算；
- 由于涉及下标访问，因此全程需要考虑数组下标是否越界的情况。

特别注意：

1、由于题目中说「环境只能保存 32 位整数」，因此这里在每一轮循环之前先要检查，具体细节请见编码。

2、Java 、Python 和 C++ 字符串的设计都是不可变的，即使用 trim() 会产生新的变量，因此我们尽量不使用库函数，使用一个变量 index 去做遍历，这样遍历完成以后就得到转换以后的数值。

```
public static int myAtoi(String str) {
        int len = str.length();
        char[] charArray = str.toCharArray();

        // 1、去除前导空格
        int index = 0;
        while (index < len && charArray[index] == ' ') {
            index++;
        }
```

```
        // 2、如果已经遍历完成（针对极端用例 "       "）
        if (index == len) {
            return 0;
        }

        // 3、如果出现符号字符，仅第 1 个有效，并记录正负
        int sign = 1;
        char firstChar = charArray[index];
        if (firstChar == '+') {
            index++;
        } else if (firstChar == '-') {
            index++;
            sign = -1;
        }

        // 4、将后续出现的数字字符进行转换
        // 不能使用 long 类型，这是题目说的
        int res = 0;
        while (index < len) {
            char currChar = charArray[index];
            // 4.1 先判断不合法的情况
            if (currChar > '9' || currChar < '0') {
                break;
            }

            // 题目中说只能存储 32 位大小的有符号整数，下面两个if分别处理整数和负数的情况。
            // 提前判断乘以10以后是否越界,但res*10可能会越界，所以这里使用Integer.MAX_VALUE/10,
这样一定不会越界。
            // 这是解决溢出问题的经典处理方式
            if (res > Integer.MAX_VALUE / 10 || (res == Integer.MAX_VALUE / 10 &&
(currChar - '0') > Integer.MAX_VALUE % 10)) {
                return Integer.MAX_VALUE;
            }
            if (res < Integer.MIN_VALUE / 10 || (res == Integer.MIN_VALUE / 10 &&
(currChar - '0') > -(Integer.MIN_VALUE % 10))) {
                return Integer.MIN_VALUE;
            }

            // 合法的情况下，才考虑转换，每一步都把符号位乘进去
            // 想想这里为什么要带着sign乘
            res = res * 10 + sign * (currChar - '0');
            index++;
        }
        return res;
    }
```

上面这个实现确实能解决问题，但是太长了，我们可以使用jdk自带的库函数来简化部分操作，例如这里的去掉前导空格，我们就可以使用trim。

2.1.3 表示数值的字符串

【剑指Offer】53 这个题与上面的题看似一致，但是要求其实有很大差异，先看题意：

题目要求

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。

数值（按顺序）可以分成以下几个部分：

- 若干空格
- 一个 小数 或者 整数
- （可选）一个 'e' 或 'E' ，后面跟着一个 整数
- 若干空格

小数（按顺序）可以分成以下几个部分：

- （可选）一个符号字符（'+' 或 '-'）
- 下述格式之一：
  - 至少一位数字，后面跟着一个点 '.'
  - 至少一位数字，后面跟着一个点 '.' ，后面再跟着至少一位数字
  - 一个点 '.' ，后面跟着至少一位数字

整数（按顺序）可以分成以下几个部分：

- （可选）一个符号字符（'+' 或 '-'）
- 至少一位数字

["+100", "5e2", "-123", "3.1416", "-1E-16", "0123"]
部分非数值列举如下：

["12e", "e12","1e3.14", "1.2.3", "+-5", "12e+5.4"]

```
示例1：
输入：s = "0"
输出：true

示例2：
输入：s = "e"
输出：false

示例3：
输入：s = "."
输出：false

示例4
输入：s = "    .1  "
输出：true
```

这个问题还是有点挑战的，一种方式是通过自动机来做，也就是更复杂的迭代。这个我们也到高级部分再分析，这里看一种常规的方法：

首先想到的是判断否false而不是判断是true，毕竟有这么多条件满足才能判断true，但是只要有一个条件不满足就可以判断false，最后代码的效率也还可以：

本题重点在于考虑到所有的情况如何来处理，表示数值的字符串遵循共同的模式：`A[.[B]][e|EC]`或者`.B[e|EC]`。

以上模式的含义是：A为数值的整数部分，B为跟在小数点之后的小数部分，C为跟在e或者E之后的指数部分。其中，A部分可以没有，比如小数.123代表0.123。如果一个数没有整数部分，那么小数部分必须有。

具体说来，A和C（也就是整数部分和指数部分）都是可能以"+"、"-"开头或者没有符号的数字串，B是数字序列，但前面不能有符号。

我们可以通过**顺序扫描字符串**来判断是否符合上述模式，首先尽可能多的扫描数字序列（开头可能有正负号），如果遇到小数点，那么扫描小数部分，遇到e或者E，则开始扫描指数部分。

除了顺序扫描以外，判断一个字符串是否满足某个模式，我们很容易想到的一个办法是**使用正则表达式**，以下给出这两种方法代码实现。

由于这个问题本身已经比较复杂了，所以我们就不必再考虑前导0和前导空格的情况了。

```
public boolean isNumeric(char[] str) {
       if(str==null)
           return false;

       boolean sign=false,decimal=false,hasE=false; //标记符号、小数点、指数符号E是否出现过
       for(int i=0;i<str.length;i++){
           if(str[i]=='e'||str[i]=='E'){ //有E或者e出现
            // 解决12e23e4这种结构
               if(i==str.length-1) //E不能是最后一位，后面必须跟指数
                   return false;
               if(hasE){
                return false; //E只能出现一次
               }
               hasE=true;
           }else if(str[i]=='.'){
         //解决  12e12.45和 12.23.45 这种情况
               if(hasE||decimal) //指数不能有.小数点只能出现一次
                   return false;
               decimal=true;
           }else if(str[i]=='+'||str[i]=='-'){
               //第一次出现符号，则判断是否满足*号的位置 [*]12e[*]3
               // i!=0表示不在开头，后面的两个表示也不在e之后
               if(!sign && i!=0 && str[i-1]!='e' && str[i-1]!='E')
                   return false;
                //第二次出现，则必须在e的后面，例如-12e[-]3
               if(sign && str[i-1]!='E' && str[i-1]!='e')
                   return false;
               sign=true;
```

```
            }else if(str[i]>'9'||str[i]<'0') //不合法字符
                return false;
        }
        return true;
    }
```

2.2 反转的问题

我们知道反转是链表的一个重要考点，反转同样是字符串的重要问题。常见问题也就是在LeetCode中列举的相关题目：

【1】LeetCode344. 反转字符串：编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 `s` 的形式给出。
【2】LeetCode541. K个一组反转：给定一个字符串 `s` 和一个整数 `k`，从字符串开头算起，每计数至 `2k` 个字符，就反转这 `2k` 字符中的前 `k` 个字符。

【3】LeetCode.917. 仅仅反转字母：给定一个字符串 `S`，返回 “反转后的” 字符串，其中不是字母的字符都保留在原地，而所有字母的位置发生反转。

【4】LeetCode151. 反转字符串里的单词：给你一个字符串 `s` ，逐个反转字符串中的所有 **单词** 。
【5】LeetCode.557. 反转字符串中的单词 III：给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

这几个题目你是否发现前三道就是要么反转字符，要么反转里面的单词。针对字符的反转又可以变换条件造出多问题。我们就从基本问题出发，各个击破。

2.2.1 反转字符串

LeetCode344. 题目要求：编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 s 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

```
示例1：
输入：s = ["h","e","l","l","o"]
输出：["o","l","l","e","h"]

示例2：
输入：s = ["H","a","n","n","a","h"]
输出：["h","a","n","n","a","H"]
```

这是最基本的反转题，也是最简单的问题，使用双指针方法最直接。具体做法是:

对于长度为 N 的待被反转的字符数组，我们可以观察反转前后下标的变化，假设反转前字符数组为 s[0] s[1] s[2] ... s[N - 1]，那么反转后字符数组为 s[N - 1] s[N - 2] ... s[0]。比较反转前后下标变化很容易得出 s[i] 的字符与 s[N - 1 - i] 的字符发生了交换的规律，因此我们可以得出如下双指针的解法：

- 将 left 指向字符数组首元素，right 指向字符数组尾元素。
- 当 left < right：

    - 交换 s[left] 和 s[right]；
    - left 指针右移一位，即 left = left + 1；
    - right 指针左移一位，即 right = right - 1。
- 当 left >= right，反转结束，返回字符数组即可。

```
public void reverseString(char[] s) {
      if (s == null || s.length() == 0) {
           return s;
       }
       int n = s.length;
       for (int left = 0, right = n - 1; left < right; ++left, --right) {
           char tmp = s[left];
           s[left] = s[right];
           s[right] = tmp;
       }
   }
```

这里使用for循环貌似条件过于复杂了，我们使用while也可以：

```
 public void reverseString(char[] s) {
      //采用双指针反转即可 左右指针元素互换 然后慢慢靠近
      int left=0,right=s.length-1;
      while(left<=right){
          char tmp=s[left];
          s[left]=s[right];
          s[right]=tmp;
          left++;
          right--;
      }
   }
```

2.2.2 K个一组反转

LeetCode541 这个题，我感觉有点没事找事，先看一下要求：

给定一个字符串 s 和一个整数 k，从字符串开头算起，每计数至 2k 个字符，就反转这 2k 字符中的前 k 个字符。

- 如果剩余字符少于 k 个，则将剩余字符全部反转。
- 如果剩余字符小于 2k 但大于或等于 k 个，则反转前 k 个字符，其余字符保持原样。

```
示例1：
输入：s = "abcd efg", k = 2
输出："bacdfeg"
示例2：
输入：s = "abcd", k = 2
输出："bacd"
```

我们直接按题意进行模拟就可以：反转每个下标从 2k的倍数开始的，长度为 k的子串。若该子串长度不足 k，则反转整个子串。

```
public String reverseStr(String s, int k) {
      if (s == null || s.length() == 0) {
          return s;
       }
       int n = s.length();
       char[] arr = s.toCharArray();
       for (int i = 0; i < n; i += 2 * k) {
          reverse(arr, i, Math.min(i + k, n) - 1);
       }
       return new String(arr);
   }

   public void reverse(char[] arr, int left, int right) {
       while (left < right) {
          char temp = arr[left];
          arr[left] = arr[right];
          arr[right] = temp;
          left++;
          right--;
       }
   }
```

2.2.3 拓展 仅仅反转字母

LeetCode.917 这个题有点难度，我们来看一下：

给定一个字符串 `S`，返回 “反转后的” 字符串，其中不是字母的字符都保留在原地，而所有字母的位置发生反转。

```
示例1：
输入："ab-cd"
输出："dc-ba"

示例2：
输入："a-bC-dEf-ghIj"
输出："j-Ih-gfE-dCba"

示例3：
输入："Test1ng-Leet=code-Q!"
输出："Qedo1ct-eeLg=ntse-T!"
```

这里第一眼感觉不是特别复杂，同样从两头向中间即可，但问题是"-"不是均匀的有些划分的段长，有的短，这就增加了处理的难度。

方法1：使用栈

将 s 中的所有字母单独存入栈中，所以出栈等价于对字母反序操作。（或者，可以用数组存储字母并反序数组。）

然后，遍历 s 的所有字符，如果是字母我们就选择栈顶元素输出。

```
class Solution {
    public String reverseOnlyLetters(String S) {

        Stack<Character> letters = new Stack();
        for (char c: S.toCharArray())
            if (Character.isLetter(c))
                letters.push(c);

        StringBuilder ans = new StringBuilder();
        for (char c: S.toCharArray()) {
            if (Character.isLetter(c))
                ans.append(letters.pop());
            else
                ans.append(c);
        }

        return ans.toString();
    }
}
```

方法2：拓展 双转指针

一个接一个输出 s 的所有字符。当遇到一个字母时，我们希望找到逆序遍历字符串的下一个字母。

所以我们这么做：维护一个指针 j 从后往前遍历字符串，当需要字母时就使用它。

```
class Solution {
    public String reverseOnlyLetters(String S) {
       if (S == null || S.length() == 0) {
            return S;
        }
        StringBuilder ans = new StringBuilder();
        int j = S.length() - 1;
        for (int i = 0; i < S.length(); ++i) {
            if (Character.isLetter(S.charAt(i))) {
                while (!Character.isLetter(S.charAt(j)))
                    j--;
                ans.append(S.charAt(j--));
            } else {
                ans.append(S.charAt(i));
            }
        }

        return ans.toString();
    }
}
```

2.2.4 . 反转字符串里的单词

LeetCode151 给你一个字符串 `s` ，逐个反转字符串中的所有 **单词** 。

单词 是由非空格字符组成的字符串。s 中使用至少一个空格将字符串中的 单词 分隔开。

请你返回一个反转 s 中单词顺序并用单个空格相连的字符串。

说明：

- 输入字符串 s 可以在前面、后面或者单词间包含多余的空格。
- 反转后单词间应当仅用一个空格分隔。
- 反转后的字符串中不应包含额外的空格。

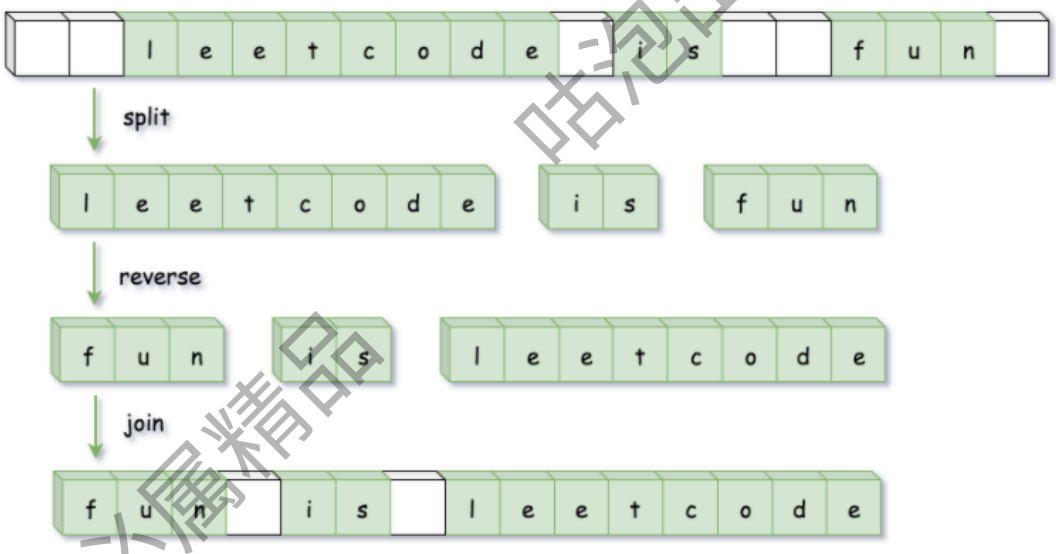
```
示例1：
输入：s = "the sky is blue"
输出："blue is sky the"

示例2：
输入：s = "hello world"

输出："world hello"
解释：输入字符串可以在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。
```

这个题也经常出现在很多面试题中，我记得曾经见过有个题是这样出的，要你按照同样的方式反转“ I love youzan”。这个题的关键在于如何处理单词。很多语言提供了相关的特性，因此我们可以首先使用语言的特性来实现：

很多语言对字符串提供了 split（拆分），reverse（反转）和 join（连接）等方法，因此我们可以简单的调用内置的 API 完成操作：

- 使用 split 将字符串按空格分割成字符串数组；
- 使用 reverse 将字符串数组进行反转；
- 使用 join 方法将字符串数组拼成一个字符串。

如图：

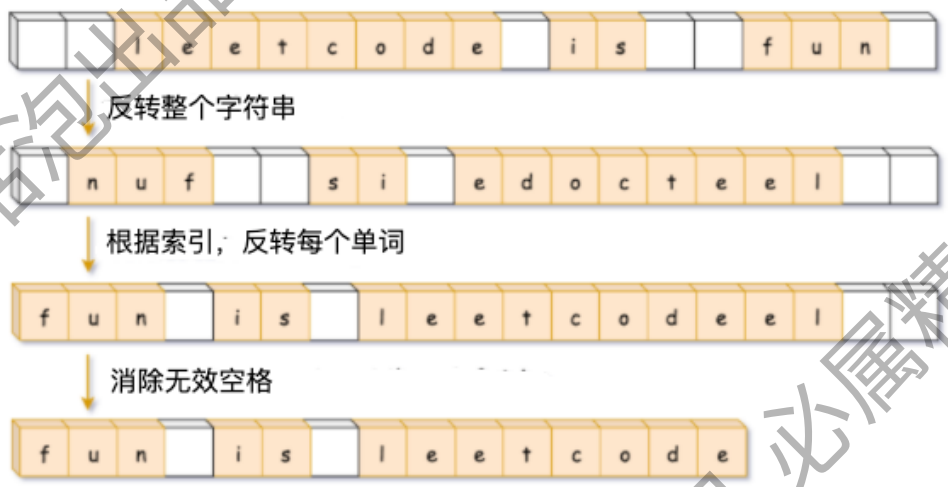


```
public String reverseWords(String s) {
        if (s == null || s.length() == 0) {
            return s;
        }
        // 除去开头和末尾的空白字符
        s = s.trim();
        // 正则匹配连续的空白字符作为分隔符分割
        List<String> wordList = Arrays.asList(s.split("\\s+"));
        Collections.reverse(wordList);
        return String.join(" ", wordList);
    }
```

如果我们要自行编写实现函数，对于字符串不可变的语言，例如java中的String，首先得把字符串转化成其他可变的数据结构，同时还需要在转化的过程中去除空格。

第二种解法

对于字符串可变的语言，就不需要再额外开辟空间了，直接在字符串上原地实现。在这种情况下，反转字符和去除空格可以一起完成。



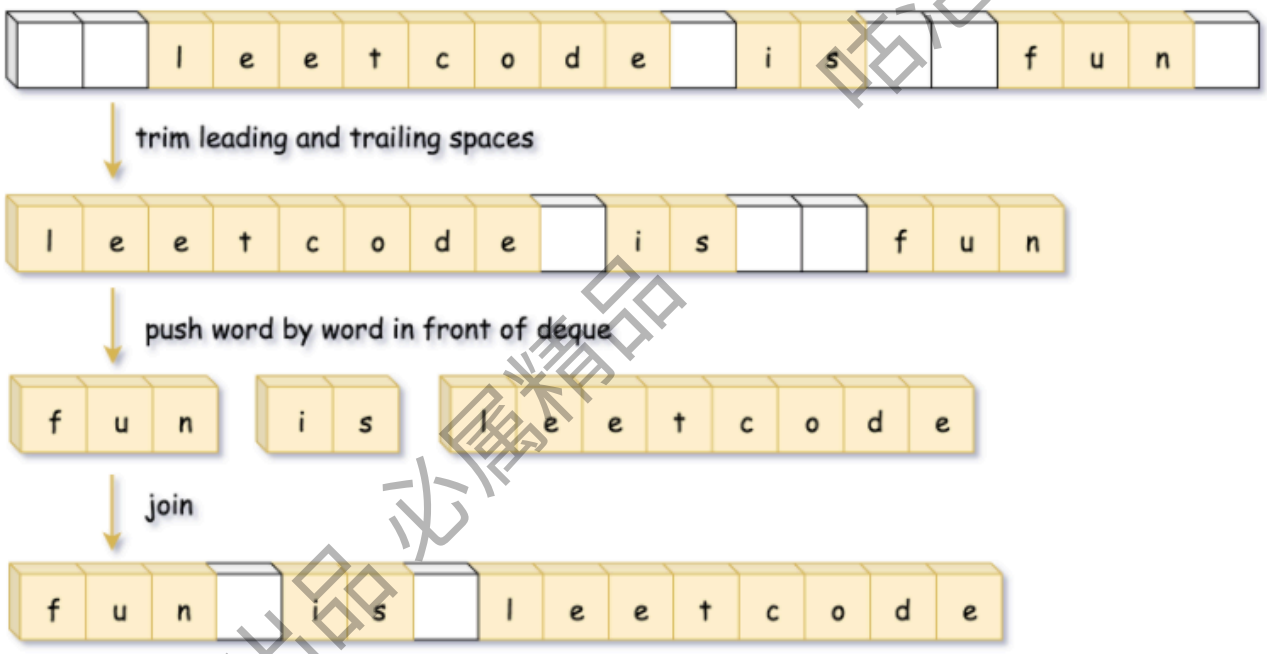
实现方法：

```
 public String reverseWords(String s) {
       if (s == null || s.length() == 0) {
           return s;
        }

        StringBuilder sb = trimSpaces(s);
        // 反转字符串
        reverse(sb, 0, sb.length() - 1);
        // 反转每个单词
        reverseEachWord(sb);
        return sb.toString();
    }

    public StringBuilder trimSpaces(String s) {
```

```
    int left = 0, right = s.length() - 1;
    // 去掉字符串开头的空白字符
    while (left <= right && s.charAt(left) == ' ') {
        ++left;
    }

    // 去掉字符串末尾的空白字符
    while (left <= right && s.charAt(right) == ' ') {
        --right;
    }

    // 将字符串间多余的空白字符去除
    StringBuilder sb = new StringBuilder();
    while (left <= right) {
        char c = s.charAt(left);

        if (c != ' ') {
            sb.append(c);
        } else if (sb.charAt(sb.length() - 1) != ' ') {
            sb.append(c);
        }
        ++left;
    }
    return sb;
}

public void reverse(StringBuilder sb, int left, int right) {
    while (left < right) {
        char tmp = sb.charAt(left);
        sb.setCharAt(left++, sb.charAt(right));
        sb.setCharAt(right--, tmp);
    }
}

public void reverseEachWord(StringBuilder sb) {
    int n = sb.length();
    int start = 0, end = 0;

    while (start < n) {
        // 循环至单词的末尾
        while (end < n && sb.charAt(end) != ' ') {
            ++end;
        }
        // 反转单词
        reverse(sb, start, end - 1);
        // 更新start，去找下一个单词
        start = end + 1;
        ++end;
    }
```

```
    }
```

另外本题还可以使用双端队列来解决。由于双端队列支持从队列头部插入的方法，因此我们可以沿着字符串一个一个单词处理，然后将单词压入队列的头部，再将队列转成字符串即可。




```
public String reverseWords(String s) {
        int left = 0, right = s.length() - 1;
        // 去掉字符串开头的空白字符
        while (left <= right && s.charAt(left) == ' ') {
            ++left;
        }

        // 去掉字符串末尾的空白字符
        while (left <= right && s.charAt(right) == ' ') {
            --right;
        }

        Deque<String> d = new ArrayDeque<String>();
        StringBuilder word = new StringBuilder();

        while (left <= right) {
            char c = s.charAt(left);
            if ((word.length() != 0) && (c == ' ')) {
                // 将单词 push 到队列的头部
                d.offerFirst(word.toString());
                word.setLength(0);
            } else if (c != ' ') {
                word.append(c);
            }
            ++left;
        }
        d.offerFirst(word.toString());
```

```
        return String.join(" ", d);
    }
```

2.2.5 反转字符串中的单词 III

LeetCode557. 给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

```
示例：
输入："Let's take LeetCode contest"
输出："s'teL ekat edoCteeL tsetnoc"
```

提示

- 在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

上一个题是将单词本身不变，单词之间的关系反转。而本题就是单词反转，单词之间的关系不变。

分析

我们可以使用额外的空间来执行，开辟一个新字符串。然后从头到尾遍历原字符串，直到找到空格为止，此时找到了一个单词，并能得到单词的起止位置。随后，根据单词的起止位置，可以将该单词逆序放到新字符串当中。如此循环多次，直到遍历完原字符串，就能得到反转后的结果。

```
public String reverseWords(String s) {
      if (s == null || s.length() == 0) {
          return s;
       }
       StringBuffer ret = new StringBuffer();
       int length = s.length();
       int i = 0;
       while (i < length) {
           int start = i;
           while (i < length && s.charAt(i) != ' ') {
               i++;
           }
           for (int p = start; p < i; p++) {
               ret.append(s.charAt(start + i - 1 - p));
           }
           while (i < length && s.charAt(i) == ' ') {
               i++;
               ret.append(' ');
           }
       }
       return ret.toString();
   }
```

此题也可以直接在原字符串上进行操作，避免额外的空间开销。当找到一个单词的时候，我们交换字符串第一个字符与倒数第一个字符，随后交换第二个字符与倒数第二个字符……如此反复，就可以在原空间上反转单词。

需要注意的是，原地解法在某些语言（比如 Java，JavaScript）中不适用，因为在这些语言中 String 类型是一个不可变的类型，需要先转换。在写转换的时候有一个更大的问题经常会被忽略，下面这段代码是有问题的，执行之后会发现无法完成反转，你能找到问题在哪里吗?

这是我们本章的一道作业题

```
class Solution {
    public String reverseWordsError(String s) {
      int length = s.length();
      char[] charArray = s.toCharArray();
        int i = 0;
        while (i < length) {
            int start = i;
            while (i < length && charArray[i] != ' ') {
                i++;
            }

            int left = start, right = i - 1;
            while (left < right) {
                swap(charArray[left], charArray[right]);
                left++;
                right--;
            }
            while (i < length && charArray[i] == ' ') {
                i++;
            }
        }
        return String.valueOf(charArray);
    }

    public void swap(char a ,char b){
        char c=a;
        a=b;
        b=c;
    }
}
```

这里的问题在于swap方法只是在局部反转了a和b，而并没有调整数组charArray中的元素，那该怎么写呢？这个是我们本周的一个作业。

2.3 验证回文串

LeetCode.125. 给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。
说明：本题中，我们将空字符串定义为有效的回文串。

回文问题在链表中是重点，在字符串中同样是个重点。当初我去美团面试第一轮技术面的第一个算法题就是让写判断字符串回文的问题。这个本身还是比较简单的，只要先转换成字符数组，然后使用双指针方法从两头到中间比较就行了。也许是过于简单了吧，面试时经常被加餐，例如LeetCode里的两道题。 一个是普通的验证回文串，第二个是找最长的子回文串。第二个问题需要动态规划等技术，有点难度，我们到高级算法里再看，这里先看一下基本

的。

```
示例1:
输入: "A man, a plan, a canal: Panama"
输出: true
解释: "amanaplanacanalpanama" 是回文串

示例2:
输入: "race a car"
输出: false
解释: "raceacar" 不是回文串
```

这个题我们可以有多种思路，最简单的方法是对字符串 s 进行一次遍历，并将其中的字母和数字字符进行保留，放在另一个字符串 sgood 中。这样我们只需要判断 sgood 是否是一个普通的回文串即可。

判断的方法有两种。第一种是使用语言中的字符串反转 API 得到 sgood 的逆序字符串 sgood_rev，只要这两个字符串相同，那么sgood 就是回文串。

```
public boolean isPalindrome(String s) {
        if (s == null || s.length() == 0) {
            return s;
        }
        StringBuffer sgood = new StringBuffer();
        int length = s.length();
        for (int i = 0; i < length; i++) {
            char ch = s.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                sgood.append(Character.toLowerCase(ch));
            }
        }
        StringBuffer sgood_rev = new StringBuffer(sgood).reverse();
        return sgood.toString().equals(sgood_rev.toString());
    }
```

第二种是使用双指针。初始时，左右指针分别指向 sgood 的两侧，随后我们不断地将这两个指针相向移动，每次移动一步，并判断这两个指针指向的字符是否相同。当这两个指针相遇时，就说明 sgood 时回文串。

```
public boolean isPalindrome(String s) {
        if (s == null || s.length() == 0) {
            return s;
        }
        StringBuffer sgood = new StringBuffer();
        int length = s.length();
        for (int i = 0; i < length; i++) {
            char ch = s.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                sgood.append(Character.toLowerCase(ch));
            }
```

```
        }
        int n = sgood.length();
        int left = 0, right = n - 1;
        while (left < right) {
            if (Character.toLowerCase(sgood.charAt(left)) !=
Character.toLowerCase(sgood.charAt(right))) {
                return false;
            }
            ++left;
            --right;
        }
        return true;
    }
```

2.4 简单搜索问题

我们为什么叫简单搜索问题呢？因为字符串的有些搜索问题非常复杂，需要dp或者更高级的算法，例如字符匹配等等，因此这里我们先看几个简单的情况。

2.4.1 字符串中的第一个唯一字符

LeetCode387. 给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。

```
示例：
s = "leetcode"
返回 0
s = "loveleetcode"
返回 2
```

提示：你可以假定该字符串只包含小写字母。

我们可以对字符串进行两次遍历，在第一次遍历时，我们使用哈希映射统计出字符串中每个字符出现的次数。在第二次遍历时，我们只要遍历到了一个只出现一次的字符，那么就返回它的索引，否则在遍历结束后返回 -1。

```
 public int firstUniqChar(String s) {
        if (s == null || s.length() == 0) {
            return 0;
        }
        Map<Character, Integer> frequency = new HashMap<Character, Integer>();
        for (int i = 0; i < s.length(); ++i) {
            char ch = s.charAt(i);
            frequency.put(ch, frequency.getOrDefault(ch, 0) + 1);
        }
        for (int i = 0; i < s.length(); ++i) {
            if (frequency.get(s.charAt(i)) == 1) {
                return i;
            }
        }
```

```
        return -1;
    }
```

2.4.2 最后一个单词的长度

LeetCode58. 给你一个字符串 s，由若干单词组成，单词前后用一些空格字符隔开。返回字符串中最后一个单词的长度。

单词 是指仅由字母组成、不包含任何空格字符的最大子字符串。

```
示例1:
输入: s = "Hello World"
输出: 5

示例2
输入: s = "   fly me   to   the moon  "
输出: 4

输入: s = "luffy is still joyboy"
输出: 6
```

这个题还是比较简单的，反向遍历。题目要求得到字符串中最后一个单词的长度，可以反向遍历字符串，寻找最后一个单词并计算其长度。

由于字符串中至少存在一个单词，因此字符串中一定有字母。首先找到字符串中的最后一个字母，该字母即为最后一个单词的最后一个字母。

从最后一个字母开始继续反向遍历字符串，直到遇到空格或者到达字符串的起始位置。遍历到的每个字母都是最后一个单词中的字母，因此遍历到的字母数量即为最后一个单词的长度。

```
public int lengthOfLastWord(String s) {
      if (s == null || s.length() == 0) {
          return 0;
       }
       int index = s.length() - 1;
       while (s.charAt(index) == ' ') {
          index--;
       }
       int lastIndex= = index;
       while (index >= 0 && s.charAt(index) != ' ') {
          index--;
       }
       return lastIndex - index;
    }
```

2.5 旋转和重排

2.5.1 左旋转字符串

[剑指Offer】58字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

```
示例1：
输入：s = "abcdefg", k = 2
输出："cdefgab"

示例2：
输入：s = "lrloseumgh", k = 6
输出："umghlrlose"
```

本题有多种方式处理的，最直观的方式是前面剪贴下来的若干字符和后面的字符保存到两个数组里，然后再按照要求合并就行了。这个在go、JavaScript、python等语言中有切片的操作，可以非常方便地处理，java中虽然没有切片，但是可以通过子串来实现相同的功能。

```
  public String reverseLeftWords(String s, int n) {
        if (s == null || s.length() == 0) {
            return s;
        }
        return s.substring(n, s.length()) + s.substring(0, n);
    }
```

第二种方式是通过StringBuilder来实现拼接，先将第k个之后的元素添加进来，然后再将前k个添加进来，代码如下：

```
public String reverseLeftWords(String s, int n) {
        if (s == null || s.length() == 0) {
            return s;
        }
        StringBuilder res = new StringBuilder();
        for(int i = n; i < s.length(); i++)
            res.append(s.charAt(i));
        for(int i = 0; i < n; i++)
            res.append(s.charAt(i));
        return res.toString();
    }
```

很明显上面两个都需要记住StringBuilder等的用法，如果使用最简单的String和char数组来处理怎么做呢？前面我们已经介绍过，所以这里只要看一下就明确了：

```
public String reverseLeftWords(String s, int n) {
      if (s == null || s.length() == 0) {
          return s;
       }
       String res = "";
       for(int i = n; i < s.length(); i++)
           res += s.charAt(i);
       for(int i = 0; i < n; i++)
           res += s.charAt(i);
       return res;
   }
```

2.5.2 判定是否互为字符重排

给定两个字符串 `s1` 和 `s2`，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。

```
示例1:
输入: s1 = "abcadfhg", s2 = "bcafdagh"
输出: true

示例2:
输入: s1 = "abc", s2 = "bad"
输出: false
```

这个题第一眼看，感觉是个排列组合的题目，然后如果使用排列的算法来处理， 难度会非常大，而且效果还不一定好。用简单的方式就能解决。

第一种方法：将两个字符串全部从小到大或者从大到小排列，然后再逐个位置比较，这时候不管两个原始字符串是什么，都可以判断出来。
代码也不复杂：

```
  public boolean checkPermutation(String s1, String s2) {
        // 将字符串转换成字符数组
        char[] s1Chars = s1.toCharArray();
        char[] s2Chars = s2.toCharArray();
        // 对字符数组进行排序
        Arrays.sort(s1Chars);
        Arrays.sort(s2Chars);
        // 再将字符数组转换成字符串，比较是否相等
        return new String(s1Chars).equals(new String(s2Chars));
    }
```

注意这里我们使用了Arrays.sort()，你是否记得我们在数组一章提到过这个方法必须牢记。

第二种方法：使用Hash，注意这里我们不能简单的存是否已经存在，因为字符可能在某个串里重复存在例如"abac"。我们可以记录出现的次数，如果一个字符串经过重新排列后，能够变成另外一个字符串，那么它们的每个不同字符的出现次数是相同的。如果出现次数不同，那么表示两个字符串不能够经过重新排列得到。

这个代码逻辑不复杂，但是写起来稍微长一点：

```
public boolean checkPermutation(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        char[] s1Chars = s1.toCharArray();
        Map<Character, Integer> s1Map = getMap(s1);
        Map<Character, Integer> s2Map = getMap(s2);
        for (char s1Char : s1Chars) {
            if (!s2Map.containsKey(s1Char) || s2Map.get(s1Char) != s1Map.get(s1Char)) {
                return false;
            }
        }
        return true;
    }

    // 统计指定字符串str中各字符的出现次数，并以Map的形式返回
    private Map<Character, Integer> getMap(String str) {
        Map<Character, Integer> map = new HashMap<>();
        char[] chars = str.toCharArray();
        for (char aChar : chars) {
            map.put(aChar, map.getOrDefault(aChar, 0) + 1);
        }
        return map;
    }
```

拓展这个题还有一种方法，就是不管原始字符串有多长，是什么，基本元素都是26个英文字母，只少不多，那么我们可以换个思维：为两个字符串分别建立两个大小为26的字母表数组，每个位置是对应的字母出现的次数。最后统计一下两个数组的字母数和每个字母出现的次数就可以了。这种方法其实也是文本搜索引擎的基本思想，例如elasticSearch等，在文本搜索里有个专门的名字，叫“倒排索引”。看一下实现代码：

```
public boolean CheckPermutation(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        int[] c1 = count(s1);
        int[] c2 = count(s2);
        for (int i = 0; i < c1.length; i++) {
            if (c1[i] != c2[i]) {
                return false;
            }
        }
        return true;
    }

    private int[] count(String str) {
        int[] c = new int[26];
```

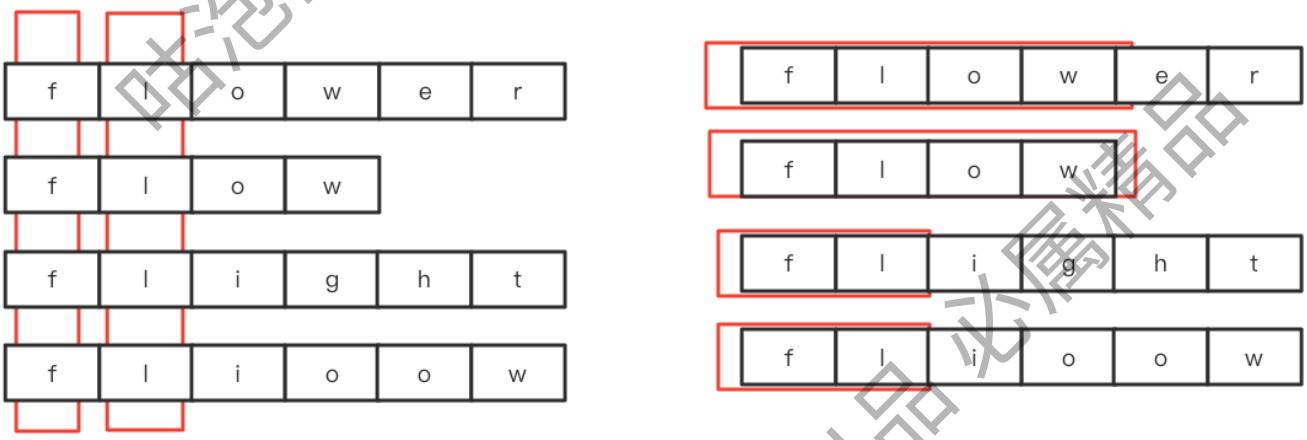
```
        char[] chars = str.toCharArray();
        for (char aChar : chars) {
            c[aChar - 'a']++;
        }
        return c;
    }
```

2.6 最长公共前缀

这是一道经典的字符串问题，先看题目要求：编写一个函数来查找字符串数组中的最长公共前缀。如果不存在公共前缀，返回空字符串 ""。

```
示例1:
输入: strs = ["flower","flow","flight"]
输出: "fl"
示例2:
输入: strs = ["dog","racecar","car"]
输出: ""
解释: 输入不存在公共前缀。
```

要解答这个问题，我们需要先看一下公共前缀的分布有什么特点，如下图：



可以看到，第一种方式，我们可以竖着比较，如左图所示，每前进一个位置就比较各个串，看是不是都是相等的，只要在某一轮遇到一个不相等的，那么就结束。

第二种方式，还可以横着比较，先比较前两个找到公共前缀fix1，然后再和第三个比较公共前缀得到fix2，我们可以确定fix2一定不会比fix1更长，然后和第四个比较，得到fix4，一直到最后一个fixn。每次得到的fix都不会比前面的长，最后比较完了还剩下的就是需要找的前缀了。

看到这里你是否有种似曾相识的感觉，我们前面合并K个数组或者K个链表不也是类似的思路吗？是的，就是类似的思路。

第三种方式， 我们是否可以对第二种进行优化一下，借鉴归并的思想，先两两一组找fix，然后将找到的fix再两两归并呢？当然可以了，这就是归并的方式。

先看第一种的实现方法，竖着比较。纵向扫描时，从前往后遍历所有字符串的每一列，比较相同列上的字符是否相同，如果相同则继续对下一列进行比较，如果不相同则当前列不再属于公共前缀，当前列之前的部分为最长公共前缀。

```
public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) {
            return "";
        }
        int length = strs[0].length();
        int count = strs.length;
        for (int i = 0; i < length; i++) {
            char c = strs[0].charAt(i);
            for (int j = 1; j < count; j++) {
                if (i == strs[j].length() || strs[j].charAt(i) != c) {
                    return strs[0].substring(0, i);
                }
            }
        }
        return strs[0];
    }
```

第二种是横着依次比较，依次遍历字符串数组中的每个字符串，对于每个遍历到的字符串，更新最长公共前缀（其实就是看是否要缩短，一定不会变长），当遍历完所有的字符串以后，即可得到字符串数组中的最长公共前缀。如果在尚未遍历完所有的字符串时，最长公共前缀已经是空串，则最长公共前缀一定是空串，因此不需要继续遍历剩下的字符串，直接返回空串即可。

```
public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) {
            return "";
        }
        String prefix = strs[0];
        int count = strs.length;
        for (int i = 1; i < count; i++) {
            prefix = longestCommonPrefix(prefix, strs[i]);
            if (prefix.length() == 0) {
                break;
            }
        }
        return prefix;
    }

    public String longestCommonPrefix(String str1, String str2) {
        int length = Math.min(str1.length(), str2.length());
        int index = 0;
        while (index < length && str1.charAt(index) == str2.charAt(index)) {
            index++;
        }
        return str1.substring(0, index);
    }
```

拓展1再看第三种，归并方法，这种方式也可以叫分治，就是先两两判断，之后再两两比较，直到得到最终的结果。

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) {
            return "";
        } else {
            return longestCommonPrefix(strs, 0, strs.length - 1);
        }
    }

    public String longestCommonPrefix(String[] strs, int start, int end) {
        if (start == end) {
            return strs[start];
        } else {
            int mid = (end - start) / 2 + start;
            String lcpLeft = longestCommonPrefix(strs, start, mid);
            String lcpRight = longestCommonPrefix(strs, mid + 1, end);
            return commonPrefix(lcpLeft, lcpRight);
        }
    }

    public String commonPrefix(String lcpLeft, String lcpRight) {
        int minLength = Math.min(lcpLeft.length(), lcpRight.length());
        for (int i = 0; i < minLength; i++) {
            if (lcpLeft.charAt(i) != lcpRight.charAt(i)) {
                return lcpLeft.substring(0, i);
            }
        }
        return lcpLeft.substring(0, minLength);
    }
}
```

2.7 字符串压缩问题

这个题也是出现频率很高的题目，经常在面经中看到。实现起来略有难度，我们一起看一下。给你一个字符数组 chars ，请使用下述算法压缩：

从一个空字符串 s 开始。对于 chars 中的每组 连续重复字符 ：

- 如果这一组长度为 1 ，则将字符追加到 s 中。
- 否则，需要向 s 追加字符，后跟这一组的长度。
  压缩后得到的字符串 s 不应该直接返回 ，需要转储到字符数组 chars 中。需要注意的是，如果组长度为 10 或 10 以上，则在 chars 数组中会被拆分为多个字符。

请在 修改完输入数组后 ，返回该数组的新长度。你必须设计并实现一个只使用常量额外空间的算法来解决此问题。

```
示例1：
输入：chars = ["a","a","b","b","c","c","c"]
输出：返回 6 ，输入数组的前 6 个字符应该是：["a","2","b","2","c","3"]
```

解释：
"aa" 被 "a2" 替代。"bb" 被 "b2" 替代。"ccc" 被 "c3" 替代。

示例2：
输入：chars = ["a"]
输出：返回 1 ，输入数组的前 1 个字符应该是：["a"]
解释：
没有任何字符串被替代。

示例3：
输入：chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"]
输出：返回 4 ，输入数组的前 4 个字符应该是：["a","b","1","2"]。
解释：
由于字符 "a" 不重复，所以不会被压缩。"bbbbbbbbbbbb" 被 “b12” 替代。
注意每个数字在数组中都有它自己的位置。

这个题貌似采用双指针策略来处理就行，但是再分析发现三个指针才够。

我们可以使用两个指针分别标志我们在字符串中读和写的位置，还要一个指针left用来标记重复字段的开始位置。read指针不断向前读取，每次当读指针 read 移动到某一段连续相同子串的最右侧，我们就在写指针 write 处依次写入该子串对应的字符和子串长度即可。

当读指针read 位于字符串的末尾，或读指针read 指向的字符不同于下一个字符时，我们就认为读指针read 位于某一段连续相同子串的最右侧。该子串对应的字符即为读指针 read 指向的字符串。我们使用变量 left 记录该子串的最左侧的位置，这样子串长度即为 read−left+1。

这里还有一个问题，就是长度可能超过10，因此还要实现将数字转化为字符串写入到原字符串的功能。这里我们采用短除法将子串长度倒序写入原字符串中，然后再将其反转即可。

```
public int compress(char[] chars) {
        int n = chars.length;
        int write = 0, left = 0;
        for (int read = 0; read < n; read++) {
            if (read == n - 1 || chars[read] != chars[read + 1]) {
                chars[write++] = chars[read];
                int num = read - left + 1;
                if (num > 1) {
                    int anchor = write;
                    while (num > 0) {
                        chars[write++] = (char) (num % 10 + '0');
                        num /= 10;
                    }
                    reverse(chars, anchor, write - 1);
                }
                left = read + 1;
            }
        }
        return write;
    }
```

```
    public void reverse(char[] chars, int left, int right) {
        while (left < right) {
            char temp = chars[left];
            chars[left] = chars[right];
            chars[right] = temp;
            left++;
            right--;
        }
    }
}
```

3.大厂算法实战

本章我们介绍了很多字符串的基本问题，字符串是算法考察中的重要组成部分。本文列举的题目还是比较简单的，但这是我们研究高级问题的基础。后面我们会继续补充大厂考察过的字符串问题。