# 28. Implement strStr()或者indexOf()

<https://leetcode.com/problems/implement-strstr/description/>

Implement **strStr().**

Return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "hello", needle = "ll"

Output: 2

Example 2:

Input: haystack = "aaaaa", needle = "bba"

Output: -1

**Clarification**:

What should we return when needle is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when needle is an empty string. This is consistent to C's strstr() and *Java's indexOf()*.

## 算法1：分析indexOf源码

**意思就是说要看源码**。

class Solution {

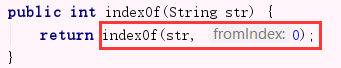
public int strStr(String haystack, String needle) {

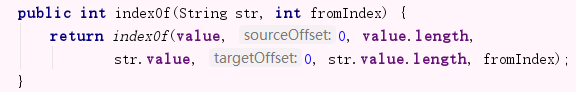
**return haystack.indexOf(needle);**

}

}

String的indexOf源码分析：多个重载方法。

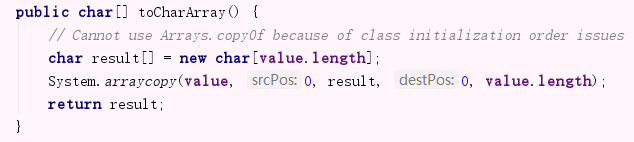




注意：String是利用字符数组存储的，**value属性**就是对应**一个字符数组**。String.toCharArray()方法返回的就是value字符数组的一个副本。



对String的value字符数组拷贝一份然后返回。



真正的valueOf方法：

/\*\*

\* Code shared by String and StringBuffer to do searches. The

\* source is the character array being searched, and the target

\* is the string being searched for.

\*

\* @param source the characters being searched(被搜索的).

\* @param sourceOffset offset of the source string.

\* @param sourceCount count of the source string(被搜索的字符串长度).

\* @param target the characters being searched for(正在搜索的).

\* @param targetOffset offset of the target string.

\* @param targetCount count of the target string.

\* @param fromIndex the index to begin searching from.(**对应的就是被搜索字符串开始搜索的索引位置**)

\*/

static int indexOf(char[] source, int sourceOffset, int sourceCount,

char[] target, int targetOffset, int targetCount,

int fromIndex) {

if (fromIndex >= sourceCount) {//开始搜索的位置大于原字符串的长度

return (targetCount == 0 ? sourceCount : -1);//若被搜索的为空，返回原字符串的长度值。确实如此？下面有示例？？？不过至于为什么？不理解。

}

if (fromIndex < 0) {

fromIndex = 0;

}

if (targetCount == 0) {//如果搜索的字符串为空，则返回起始位置。

return fromIndex;

}

char first = target[targetOffset];

int max = sourceOffset + (sourceCount - targetCount);//需要循环的次数

for (int i = sourceOffset + fromIndex; i <= max; i++) {

/\* Look for first character. \*/

if (source[i] != first) {

while (++i <= max && source[i] != first);//直到找到与第一个字符相匹配的

}

/\* Found first character, now look at the rest of v2 找到了第一个匹配的元素，下面判断剩下的是否匹配\*/

if (i <= max) {

int j = i + 1;//保留i不变，i还要用于外循环；j表示第2字符对应索引

int end = j + targetCount - 1;//end表示最后一个元素对应索引

**for (int k = targetOffset + 1; j < end && source[j]**

**== target[k]; j++, k++);**

if (j == end) {//如果j到达了end，说明完全匹配

/\* Found whole string. \*/

return i - sourceOffset;//返回对应的索引

}

}

}

return -1;

}

对于源码中的疑惑：

if (fromIndex >= sourceCount) {//开始搜索的位置大于原字符串的长度

return (targetCount == 0 ? sourceCount : -1);//

验证：若fromIndex大于原字符串长度，返回的就是原字符串的长度。

**System.out.println("aaa".indexOf("",4));//3**

**System.out.println("aaa".indexOf("",100));//3**



## 算法2：

思路：特殊情况，若needle为空，则返回0，若haystack为空，返回-1；

将字符串转换成字符数组进行比较。遍历haystack，首先判断needle的第一字符与遍历的haystack的每个元素是否相等，若相等，则就利用遍历needle的方式继续判断needle的后续字符与haystack的后续字符是否相等，如果存在某个字符不相等直接跳出循环，循环结束后，需要通过判断i是否等于needle.length来说明是否完全匹配了，若匹配直接return index，否则继续遍历haystack。(可以实现，有改进之处)

class Solution {

public int strStr(String haystack, String needle) {

if(needle == null || "".equals(needle)) return 0;

if(haystack == null || "".equals(haystack)) return -1;

char[] hays = haystack.toCharArray();

char[] needles = needle.toCharArray();

int index = 0;

**for(;index < hays.length;index++ ){**

if(hays[index] == needles[0]){

int i = 1;

if(index + needles.length>hays.length) return -1;

for(i = 1;i<needles.length;i++){

if(hays[index+i] != needles[i]) break;

}

**if(i == needles.length) return index;**

}

}

return -1;

}

}

## 算法3：根据indexOf源码修改。最佳。

思路：解读indexOf源码之后，发现算法2有需要修改的地方。当然大体思路与源码是一致的，但是有需要优化的地方。优化：外循环不需要循环**hays.length次，因为后面对应的长度都小于needle了，indexOf源码中：**int max = sourceOffset + (sourceCount - targetCount);

因此这里需要增加int max = hasy.length – needle.length;

这样内部的判断index + needles.length>hays.length就没有必要了。

代码为：

//根据indexOf的源码

class Solution {

public int strStr(String haystack, String needle) {

if(needle == null || "".equals(needle)) return 0;

if(haystack == null || "".equals(haystack)) return -1;

char[] hays = haystack.toCharArray();

char[] needles = needle.toCharArray();

int index = 0;

int max = hays.length-needles.length+1;//优化了循环次数,省去了下面的index + needles.length>hays.length判断

for(;index < max;index++ ){

if(hays[index] == needles[0]){

int i = 1;

// if(index + needles.length>hays.length) return -1;

*// for(;i < needles.length;i++){*

*// if(hays[index+i] != needles[i]) break;*

*// }*

**for(;i < needles.length&&hays[index+i] == needles[i];i++);//与上面的等价**

if(i == needles.length) return index;

}

}

return -1;

}

}

注意:代码书写的优雅：下面两种for循环代码一致，如果for循环内只有判断条件，可以将其写入for的第二个位置，这样只有一个for();语句显得好看。

*// for(;i < needles.length;i++){*

*// if(hays[index+i] != needles[i]) break;*

*// }*

for(;i < needles.length&&hays[index+i] == needles[i];i++);//与上面的等价