哈希表：无重复字符的最长子串

题目来源：Leetcode 3 https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/

暴力解法：使用语言特性

Java代码

```java
/**
 * 解法一：暴力解法，使用语言特性
 * 思路：
 *      先生成不含重复字符的子串，再统计最长子串的长度
 * 解答结果：
 *      执行耗时:260 ms,击败了6.86% 的Java用户
 *      内存消耗:39.8 MB,击败了5.04% 的Java用户
 * 复杂度分析：
 *      时间复杂度：O(n^2)，实际时间消耗较大
 *          将字符串切割成单字符数组：O(n)
 *          遍历并截取子串：O(n^2)
 *          统计最长子串长度：O(n)
 *      空间复杂度：O(n^2)，子串都是字符串常量，实际空间消耗巨大
 *          数组列表：O(n^2)，理论上最多有 n(n+1) / 2 个子串
 *          子串都是常量：O(n^2)
 *
 * @param s
 * @return
 */
public int lengthOfLongestSubstring(String s) {
    int length;
    if (s == null || (length = s.length()) == 0) {
        return 0;
    }
    // 1.生成所有不包含重复字符的子串
    List<String> list = new ArrayList<>();
    // 1.1.将字符串切割成单个字符组成的数组，直接添加到集合中
    list.addAll(Arrays.asList(s.split("")));
    // 1.2.遍历字符串，生成所有的不含重复字符的子串，添加到集合中
    for (int start = 0; start < length; start++) { // 遍历子串的起始字符
        for (int end = start + 1; end < length; end++) { // 遍历子串的终止字符
            String subStr = s.substring(start, end);
            // 当前字符在前面的子串中已出现，则跳过该字符
            if (subStr.indexOf(s.charAt(end)) != -1) {
                break;
            }
            // 否则，添加到集合中
            list.add(s.substring(start, end + 1));
        }
    }
    // 2.统计最长子串的长度
    int maxLength = 1;
    for (String sub : list) {
```

```
            int subLen;
            if ((subLen = sub.length()) > maxLength)
                maxLength = subLen;
        }
        return maxLength;
    }
```

优化解法：使用语言特性

java代码

```
    /**
     * 解法二：优化解法，使用语言特性
     * 思路：
     *      先生成不含重复字符的子串，再统计最长子串的长度
     * 解答结果：
     *      执行耗时:260 ms,击败了6.86% 的Java用户
     *      内存消耗:39.8 MB,击败了5.04% 的Java用户
     * 复杂度分析：
     *      时间复杂度：O(n^2)，实际时间消耗较大
     *          将字符串切割成单字符数组：O(n)
     *          遍历并截取子串：O(n^2)
     *          统计最长子串长度：O(n)
     *      空间复杂度：O(n^2)，子串都是字符串常量，实际空间消耗较大
     *          多个子串：O(n^2)，理论上最多有 n(n+1) / 2 个
     *
     * @param s
     * @return
     */
    public int lengthOfLongestSubstring(String s) {
        int len;
        if (s == null || (len = s.length()) == 0) {
            return 0;
        }

        int maxLength = 1; // 最长子串的长度。默认值1：原字符串有数据，至少是1
        // 1.遍历字符串，生成所有的不含重复字符的子串
        for (int start = 0; start < len; start++) { // 遍历子串的起始字符
            for (int end = start + 1; end < len; end++) { // 遍历子串的终止字符
                String subStr = s.substring(start, end); // 截取当前字符的前置子串
                // 当前字符在前面的子串中已出现，则跳过该字符
                if (subStr.indexOf(s.charAt(end)) != -1) {
                    break;
                }
                // 2.统计最长子串的长度
                int subLen = end + 1 - start; // 子串长度
                if (subLen > maxLength)
                    maxLength = subLen;
            }
        }
        return maxLength;
    }
```

最优解：哈希表+双指针

java代码

哈希算法：将任意字符数据转成整数

```
    /**
     * 哈希算法
     * 根据给定字符计算出一个 int 型哈希值
     *
     * @param key
     * @return
     */
    int hash(char key) {
        return key;
    }
```

解题代码：

```
    /**
     * 最优解：哈希表 + 双指针
     * 思路：
     *       遍历字符串的所有字符，使用双指针定位不重复子串的起止索引，
     *       并将字符临时记录到哈希表中，每次计算子串的长度并统计最大值
     * 步骤：
     * 1.定义哈希表，临时存储子串字符和查重
     *  定义哈希函数，对任意字符生成唯一整数值
     * 2.遍历字符串，通过双指针循环定位子串
     *  右指针在哈希表中是否存在：
     *       否，记录到哈希表，移动右指针，计算长度；
     *       是，删除哈希表中左指针元素，移动左指针，重复检查右指针元素是否还存在；
     * 3.每次计算子串长度，比较并保留最大值
     *
     * 边界问题
     *  遍历字符串的字符，注意索引越界
     *  计算子串长度时，注意子串的起止索引
     *  根据测试用例，子串长度不会超过哈希表容量：new char[128]
     * 细节问题
     *  子串长度是：end + 1 - start
     *  出现重复元素后，左指针逐个移动，直到与当前重复的字符索引+1
     *
     * @param s
     * @return
     */
    public int lengthOfLongestSubstring(String s) {
        int len;                // 源字符串长度
        if (s == null || (len = s.length()) == 0) {
            return 0;
        }
        int res = 0,            // 最长不重复子串的长度
            left = 0,           // 子串最左端字符索引
            right = 0;          // 子串最右端字符索引

        // 1.定义哈希表，支持ASCII码表的全部字符
        char[] chs = new char[128];
        // 2.遍历字符串的所有字符
```

```
        while (right < len) {
            char rightChar = s.charAt(right); // 右指针字符
            char c = chs[(chs.length - 1) & hash(rightChar)]; // hash算法
            if (rightChar != c) { // 未重复出现
                // 2.1.双指针定位子串索引：右指针自增
                right++;
                // 将不重复字符记录到哈希表中
                chs[(chs.length - 1) & hash(rightChar)] = rightChar;
                // 3.每次记录子串长度，并计算最大值
                int size = right - left; // 每个不重复子串的长度
                res = res > size ? res : size;
            } else { // 重复出现
                // 2.2.双指针定位子串索引：左指针自增。从哈希表中移出最左侧字符：赋默认值
                char leftChar = s.charAt(left++);
                chs[(chs.length - 1) & hash(leftChar)] = '\u0000';
            }
        }
        return res;
    }
```

最优解再优化：哈希表作用变形：字符ASCII码值 --> 字符最后出现的位置。

```
    /**
     * 最优解：哈希表 + 双指针优化思路
     * 1.哈希表作用变形
     *  字符ASCII码值 --> 字符
     *  字符ASCII码值 --> 字符最后出现索引
     * 2.遇到重复元素后，左指针移动优化
     *  逐个移动到前一个相同字符出现后的位置  -->
     *  一次性定位到前一个相同字符出现后的位置
     *
     * 步骤：
     * 1.初始化哈希表，存入非ASCII码值作为默认值
     * 2.遍历字符串，使用双指针定位子串索引
     *  字符已出现：取出哈希表中记录，左指针到记录+1
     *  无论是否出现，将右指针记录到哈希表
     * 3.每次移动都记录子串长度，保留最大值
     *
     * @param s
     * @return
     */
    public int lengthOfLongestSubstring(String s) {
        int len;                // 源字符串长度
        if (s == null || (len = s.length()) == 0) {
            return 0;
        }
        int res = 0,            // 最长不重复子串的长度
            left = 0,           // 子串最左端字符索引
            right = 0;          // 子串最右端字符索引

        // 1.哈希表中填充ASCII码表不包含的数值作为默认值：-1
        int[] arr = new int[128]; // ASCII码 -->  最后一次出现的索引
        for (int i = 0; i < arr.length; i++)
            arr[i] = -1;
        // 2.遍历字符串的所有字符
        while (right < len) { // abba
            int c = s.charAt(right);
```

```
            if (arr[c] != -1) { // 检测该字符是否已经出现：已出现
                // 出现，则移动左指针，直接定位到上次出现的下一个索引
                int start0 = arr[c] + 1;
                // 2.1.使用双指针定位子串索引：左指针直接定位
                left = left >= start0 ? left : start0; // 只往右不往左
            }
            // 无论是否重复，记录该字符最后一次出现的索引
            arr[c] = right; // ASCII码值 ：  字符最后出现的索引
            // 3.计算子串长度，记录最大值：右索引+1 - 左索引
            int size = right + 1 - left;
            res = res > size ? res : size;

            // 2.2.使用双指针定位子串索引：右指针始终自增
            right++;

        }
        return res;
    }
```

C++代码

```
/**
 * 执行用时：0 ms, 在所有 C++ 提交中击败了 100.00% 的用户
 * 内存消耗：7.1 MB, 在所有 C++ 提交中击败了 74.64% 的用户
 */
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        if (s.size() == 0) {
            return 0;
        }
        int res = 0,            // 最长不重复子串的长度
            left = 0,           // 子串最左端字符索引
            right = 0;          // 子串最右端字符索引
        int len = s.size();     // 源字符串长度

        // 1.哈希表中填充ASCII码表不包含的数值作为默认值：-1
        int arr[128]; // ASCII码 -->  最后一次出现的索引
        for (int i = 0; i < 128; i++)
            arr[i] = -1;
        // 2.遍历字符串的所有字符
        while (right < len) { // abba
            int c = s[right];
            if (arr[c] != -1) { // 检测该字符是否已经出现：已出现
                // 出现，则移动左指针，直接定位到上次出现的下一个索引
                int start0 = arr[c] + 1;
                // 2.1.使用双指针定位子串索引：左指针直接定位
                left = left >= start0 ? left : start0; // 只往右不往左
            }
            // 无论是否重复，记录该字符最后一次出现的索引
            arr[c] = right; // ASCII码值 ：  字符最后出现的索引
            // 3.计算子串长度，记录最大值：右索引+1 - 左索引
            int size = right + 1 - left;
            res = res > size ? res : size;
```

```
            // 2.2.使用双指针定位子串索引：右指针始终自增
            right++;

        }
        return res;
    }
};
```

Python代码

```
'''
执行用时：56 ms, 在所有 Python3 提交中击败了 97.4% 的用户
内存消耗：13.5 MB, 在所有 Python3 提交中击败了 30.79% 的用户
'''
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        length = 0
        if s is None or (length := len(s)) == 0:
            return 0
        res, left, right = 0, 0, 0
        # 1.哈希表中填充ASCII码表不包含的数值作为默认值：-1
        arr = []
        for i in range(128):
            arr.append(-1)

        # 2.遍历字符串的所有字符
        while right < length:
            c = ord(s[right])  # ord 函数用于获取某个字符的ASCII码值
            if arr[c] != -1:  # 检测该字符是否已经出现：已出现
                start0 = arr[c] + 1  # 则移动左指针，直接定位到上次出现的下一个索引
                # 2.1.使用双指针定位子串索引：左指针直接定位
                left = left if left >= start0 else start0

            arr[c] = right  # 无论是否重复，记录该字符最后一次出现的索引
            # 3.计算子串长度，记录最大值：右索引+1 - 左索引
            size = right + 1 - left
            res = res if res > size else size

            # 2.2.使用双指针定位子串索引：右指针始终自增
            right += 1

        return res
```

测试用例

```
输入："abcabcbb"
输出：3
解释：因为无重复字符的最长子串是 "abc"，所以其长度为 3
```

输入："bbbbb"
输出：1
解释：因为无重复字符的最长子串是 "b"，所以其长度为 1

输入："pwwkew"
输出：3
解释：因为无重复字符的最长子串是 "wke"，所以其长度为 3