链滴

LeetCode #3

作者：friedwm
原文链接：https://ld246.com/article/1536113754963
来源网站：链滴

问题：给定一个字符串，找出不含有重复字符的**最长子串**的长度。

示例：

输入: "abcabcbb"

输出: 3

解释: 无重复字符的最长子串是 "abc"，其长度为 3。

思路：

1. 暴破，从位置0开始，把遇到的字符放到Set中，如果有重复则向前推进
2. 空间换时间，记录各个字符的位置，在遇到重复字符时向前推进，但直接跳到已有的重复字符之后个字符开始计算
3. 思路2的实现上的优化，直接用String.charAt()获取单个字符，不先把整个字符串转换成数组（因为tring底层就是字符数组了）

1000万个随机字符的字符串耗时：

1. cost: 3489 ms
2. cost: 312 ms
3. cost: 305 ms

思路3有较大概率略微快于思路2，但都比思路1快10倍以上，理论上字符串越长差别越明显

代码：

```
package xyz.quxiao.play.lab.leetcode;

import com.google.common.base.Stopwatch;
import com.google.common.collect.Maps;
```

```
import com.google.common.collect.Sets;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;
import org.apache.commons.collections4.CollectionUtils;
import org.apache.commons.collections4.MapUtils;
import org.apache.commons.lang3.RandomStringUtils;

/**
 * 给定一个字符串，找到最长不含相同字符的字串长度 * * @author 作者 :quxiao 创建时间：2017/1
/7 18:21
 */public class Problem3 {

  public static void main(String[] args) {
    String s = randomStr();
    Stopwatch sw = Stopwatch.createStarted();
    int r1 = lengthOfLongestSubstring(s);
    sw.stop();
    System.out.println(r1 + " cost: " + sw.elapsed(TimeUnit.MILLISECONDS) + " ms");
    sw.reset();

    sw.start();
    int r2 = lengthOfLongestSubstring2(s);
    sw.stop();
    System.out.println(r2 + " cost: " + sw.elapsed(TimeUnit.MILLISECONDS) + " ms");
    sw.reset();

    sw.start();
    int r3 = lengthOfLongestSubstring3(s);
    sw.stop();
    System.out.println(r3 + " cost: " + sw.elapsed(TimeUnit.MILLISECONDS) + " ms");
    sw.reset();

  }

  public static String randomStr() {
    int max = 10000000;
    return RandomStringUtils.randomAlphabetic(max);
  }

  // 思路1：暴破，从0开始遍历每个字符加到set中，如果成功了比较当前长度与max大小；否则推进
  public static int lengthOfLongestSubstring(String s) {
    if (s == null || s.length() == 0) {
      return 0;
    }

    int max = 0;
    char[] chars = s.toCharArray();
    int len = chars.length;
    int idx = 0;
    while (idx < len) {
      // 计算该idx的最长字串
```

```
  int start = idx;
      Set set = Sets.newHashSet();
      while (start < len && set.add(chars[start++])) {
      }
      if (set.size() > max) {
        max = set.size();
      }
      idx++;
    }

    return max;
  }

  // 思路2：空间换时间，记录各个字符的位置，在遇到重复字符时向前推进，但直接跳到已有的重复
符之后一个字符开始计算
  public static int lengthOfLongestSubstring2(String s) {
    if (s == null || s.length() == 0) {
      return 0;
    }

    int max = 0;
    char[] chars = s.toCharArray();
    int len = chars.length;
    int i = 0;
    // 已经清除的最后一个idx，初始时-1表示未清除任何字符
  int lastClearIdx = -1;

    Map cMap = new HashMap<>();

    while (i < len) {
      char c = chars[i];
      // 不存在
  if (!cMap.containsKey(c)) {
        cMap.put(c, i);
        max = Math.max(max, cMap.size());
        i++;
      } else {
        int exist = cMap.get(c);
        // 清除 cur - exist之间的字符串
  for (int j = lastClearIdx + 1; j <= exist; j++) {
          cMap.remove(chars[j]);
        }
        lastClearIdx = exist;
      }
    }
    return max;
  }

  // 思路3，但是直接利用string.charAt()，不转换成arr
  public static int lengthOfLongestSubstring3(String s) {
    if (s == null || s.length() == 0) {
      return 0;
    }
```

```
    int max = 0;
    int len = s.length();
    int i = 0;
    // 已经清除的最后一个idx，初始时-1表示未清除任何字符
  int lastClearIdx = -1;

    Map cMap = new HashMap<>();

    while (i < len) {
      char c = s.charAt(i);
      // 不存在
  if (!cMap.containsKey(c)) {
        cMap.put(c, i);
        max = Math.max(max, cMap.size());
        i++;
      } else {
        int exist = cMap.get(c);
        // 清除 cur - exist之间的字符串
  for (int j = lastClearIdx + 1; j <= exist; j++) {
          cMap.remove(s.charAt(j));
        }
        lastClearIdx = exist;
      }
    }
    return max;
  }
}
```