Jan 21

Section 1 Coding interview

1.Coding style

几个技巧快速提高代码风格

1. 二元运算符左右两边加空格
2. if, for 和括号之间加空格
3. 严格按照要求进行程序缩进
4. 即便 if / for 语句内部只有一句话，也要加上花括号
5. 变量名使用有意义的英文名，不要用a,b,c,s1,s2
6. 区分不同的逻辑块，逻辑块之间用空行隔开。简要注释每个部分做的事情
7. 多用 Helper Function 或子函数，不要所有程序都写在一个大函数里
8. 如果前面有return， 后面不用else， else越多代码可读性越差

2.贪心算法

贪心法的问题，面试基本不会考，因为等同于考智力题或者是背诵题。一个面试官想要自己凭空创造出一个面试题是使用贪心算法的，是非常困难的。
需要背诵的贪心算法：http://www.jiuzhang.com/qa/2099/

3.函数式编程

在一个好的编程风格中，将部分独立的逻辑函数化是一个重要的手段
e.g. Longest Palindromic Substring

非函数式：

```
 public int longestPalindrome(String s) {
    int longest = 0;
    for (int i = 0; i < s.length(); i++) {
        for (int j = i; j < s.length(); j++) {
            int left = i, right = j;
            while (left <= right && s.charAt(left) == s.charAt(right)) {
                left++;
                right--;
            }
            if (left > right) {
                longest = Math.max(longest, j - i + 1);
            }
        }
    }
        return longest;
}
```

函数式：

```
public int longestPalindrome(String s) {
    int longest = 0;
    for (int i = 0; i < s.length(); i++) {
        for (int j = i; j < s.length(); j++) {
            if (isPalindrome(s, i, j)) {
                longest = Math.max(longest, j - i + 1);
            }
        }
    }
        return longest;
}

private boolean isPalindrome(String s, int left, int right) {
    while (left <= right) {
        if (s.charAt(left) != s.charAt(right)) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
```

4.Java中判断字符串相等的方式

== 比较的是内存地址，是否同一个实例
equals 首先判断内存地址是否一致，如果一致返回true，否则比较字符串的内容是否一致，如果内容一致也返回true

5.Java null与""区别
null 空对象
"" 空字符串

6.常用字符串操作

其他还有很多常见的一些 String 的函数经常用到，如：

- substring, 取子字符串
- startsWith, 判断一个字符串是否以某个字符串开头
- endsWith, 判断一个字符串是否以某个字符串结尾
- compareTo, 比较两个字符串的大小，一般用于按照字典序排序字符串
- indexOf，查询一个字符串里另外一个字符串第一次出现的位置
- lastIndexOf, 查询一个字符串里另外一个字符串出现的最后一个位置
- format, 格式化字符串

Section 2 二分法和lgn算法

1.递归定义

1. 三要素：参数，区间，目标
2. 递归拆解
3. 返回条件

2.区别栈空间和堆空间：new 出来的就放在堆空间，其他都是栈空间

3.二分法通用模版

```
public class Solution {
    /**
     * @param A an integer array sorted in ascending order
     * @param target an integer
     * @return an integer
     */
    public int findPosition(int[] nums, int target) {
        if (nums == null || nums.length == 0) {
            return -1;
        }
        //特殊情况的处理
        int start = 0, end = nums.length - 1;
        // 要点1: start + 1 < end
        while (start + 1 < end) {
            // 要点2: start + (end - start) / 2
            int mid = start + (end - start) / 2;
            // 要点3: =, <, > 分开讨论, mid 不+1也不-1
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                start = mid;
            } else {
                end = mid;
            }
        }

        // 要点4: 循环结束后，单独处理start和end
        if (nums[start] == target) {
            return start;
        }
        if (nums[end] == target) {
            return end;
        }
        return -1;
    }
}
```

模版常见问题

Q: 为什么要用 start + 1 < end? 而不是 start < end 或者 start <= end?

A: 为了避免死循环。二分法的模板中，整个程序架构分为两个部分：

1. 通过 while 循环，将区间范围从 n 缩小到 2 （只有 start 和 end 两个点）。
2. 在 start 和 end 中判断是否有解。

start < end 或者 start <= end 在寻找目标最后一次出现的位置的时候，出现死循环。

Q: 为什么明明可以 start = mid + 1 偏偏要写成 start = mid?

A: 大部分时候，mid 是可以 +1 和 -1 的。在一些特殊情况下，比如寻找目标的最后一次出现的位置时，当 target 与 nums[mid] 相等的时候，是不能够使用 mid + 1 或者 mid - 1 的，因为会导致漏掉解。那么为了节省脑力，统一写成 start = mid / end = mid 并不会造成任何解的丢失，并且也不会损失效率——log(n) 和 log(n+1) 没有区别。

Practice:

- Find K closest elements
- Search in a Big Sorted Array
- Pow(x, n)
- Find Minimum in Rotated Sorted Array
- Fast Power
- Find Peak Element
- First Bad Version
- Search in RSA