String leetcode

字符串的题目处理方式技巧性较强，基本上在这里给出的代码都是在原来ac代码上，看讨论区域上比较简洁好理解的贴上。

67. Add Binary

题目：给出两个字符串，代表的是二进制表达，然后返回和的二进制表达。

思路：按照整数相加的方式，同样的方法，用一个carry也就是进位标记。

```
public class Solution {
    public String addBinary(String a, String b) {
        int i = a.length() - 1;
        int j = b.length() - 1;
        int carry = 0;
        StringBuilder sb = new StringBuilder();
        while (i >= 0 || j >= 0) {
            int sum = carry;
            if (i >= 0) sum += a.charAt(i--) - '0';
            if (j >= 0) sum += b.charAt(j--) - '0';
            sb.append(sum % 2);//由于是二进制表达式，所以是取余2以及除以2进位。
            carry = sum / 2;
        }
        if (carry != 0) sb.append(carry);
        return sb.reverse().toString();
    }
}
```

165. Compare Version Numbers

题目：给出两个字符串，字符串中用 .分割数字，代表一个版本号。返回值分为：

- 如果第一个版本大于第二个，返回1；
- 如果第一个版本小于第二个，返回－1；
- 否则返回0；

思路：
根据题目给出的返回信息，很容易想到利用java内嵌的整型比较函数。当然首先得把字符串分割为字符串组，然后逐个比较。

```
public class Solution {
    public int compareVersion(String version1, String version2) {
        String[] a = version1.split("\\.");
        String[] b = version2.split("\\.");
        int max = Math.max(a.length, b.length);
        for (int i = 0; i < max; i++) {
            Integer v1 = (i < a.length) ? Integer.parseInt(a[i]) : 0;
            Integer v2 = (i < b.length) ? Integer.parseInt(b[i]) : 0;
            if (v1.compareTo(v2) != 0) {
                return v1.compareTo(v2);
            }
        }
        return 0;
    }

}
```

万一某个版本号已经用完，就意味着它的下一个版本号数字是0，这是0与任何正数相比较，都是返回我们想要的值的。

38. Count and Say

题目：比较难解释，直接上英文。

```
The count-and-say sequence is the sequence of integers beginning as follows:`
1, 11, 21, 1211, 111221, ...
1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.
Given an integer n, generate the nth sequence.
```

思路：
直接遍历，维护一个计数器和当前比较字符，然后将计数器和当前比较字符组成新的字符串。由于这里字符串长度一直在变化，建议使用StringBuilder,当当前循环结束之后，再转为字符串。和这个题目相似的还有Compress String.

```
public class Solution {
    public String countAndSay(int n) {
        if(n<=0) return "";
        String result = "1";

        for(int i=1;i<n;i++){
            StringBuilder temp = new StringBuilder();
            for(int j=0;j<result.length();j++){
                int count = 1;
                while((j+1)<result.length()&&result.charAt(j) == result.charAt(j+1)){
                    count++;
                    j++;
                }
temp.append(String.valueOf(count)+String.valueOf(result.charAt(j)));
            }
            result = temp.toString();
        }
        return result;
    }
}
```

28. Implement strStr()

题目：给定两个字符串，求出第二个字符串在第一个字符串中首次出现时的下标，如果不存在返回－1.

思路：
直接遍历。然后每次截取第一个字符串中和第二个字符串相同长度的子串，如果相同，返回当前index，否则继续遍历。值得注意的就是要保证字符串取字串时不能越界。

```
public class Solution {
    public int strStr(String haystack, String needle) {
        int result = -1;
        if(haystack==null||needle==null||needle.length()==0) return 0;
        else if(haystack.length()<needle.length()) return result;
        int length1 = haystack.length();
        int length2 = needle.length();
        for(int i=0;i<=length1-length2;i++) {
            if(haystack.substring(i,(i+length2)).equals(needle)) {
                result = i;
                break;
            }
        }
          return result;
    }
}
```

14. Longest Common Prefix

题目：给出一个字符串组，找出这个字符串组中的最长公共前缀。

思路：首先我们可以有一个基本的prefix，可以取字符串组的第一个。而且由于我们是要找一个最长公共前缀，因此我们可以利用indexOf方法来求出一个字串在一个字符串中的位置，如果不是0，代表这个子串不是前缀，那么就可以将前缀长度减一，然后继续判定。

```
public class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) return "";
        String base = strs[0];
        int index = 1;
        while (index < strs.length) {
            while (strs[index].indexOf(base) != 0)
                base = base.substring(0, base.length()-1);
            index++;
        }
        return base;
    }
}
```

8. String to Integer (atoi)

题目：给定一个字符串，将字符串转为相应的整数。如果字符串中存在其他字符，返回当前整数。

思路：
将字符串转为整数，首先得确保字符串首尾没有空格，然后确认整数的符号，再逐次相乘相加即可，值得注意的是，如果超出整数的表达范围，返回最大最小值即可。
如何判定一个数加上另一个数之后会超出表达范围，利用的trick就是提前用最大最小值判定当前整数离最大最小值的距离。这样说可能有些抽象，直接看代码：

```
public class Solution {
    public int myAtoi(String str) {
        if(str==null)
    {
        return 0;
    }
    str = str.trim();
    if(str.length()==0)
        return 0;
    boolean isNeg = false;
    int i = 0;
    if(str.charAt(0)=='-' || str.charAt(0)=='+')
    {
        i++;
        if(str.charAt(0)=='-')
            isNeg = true;
    }
    int res = 0;
    while(i<str.length())
    {
        if(str.charAt(i)<'0'||str.charAt(i)>'9')
            break;
        int digit = (int)(str.charAt(i)-'0');
        if(isNeg && res>-((Integer.MIN_VALUE+digit)/10))  //如果是负数，加之前先进行判定，很容易推出当前表达式。
            return Integer.MIN_VALUE;
        else if(!isNeg && res>(Integer.MAX_VALUE-digit)/10)  //如果是正数，同样的有个数学边界表达式。
            return Integer.MAX_VALUE;
        res = res*10+digit;
        i++;
    }
    return isNeg?-res:res;
    }
}
```

125. Valid Palindrome

题目：给定一个字符串，判定字符串中的数字字符部分是否是回文串。

思路：
很简单的题目。首先利用正则表达式将字符串中不是数字和字符的部分给替换掉，然后再判定回文。

```
public class Solution {
    public boolean isPalindrome(String s) {
        String regex = "[^a-zA-Z0-9]";
        s = s.replaceAll(regex, "");//首先用正则表达式替换。
        s = s.toLowerCase();
        int length = s.length();
        for (int i = 0; i < length / 2; i++) {
            if (s.charAt(i) !=  s.charAt(length - 1 - i)){
                return false;
            }
        }
        return true;
    }
}
```

20. Valid Parentheses

题目：给定一个字符串，由三种括号组成，问这中组成方式是有效的么

思路：经典的栈利用解决方法。如果是左括号，入栈；如果是右括号，弹出判断相等。

```
public class Solution {
    public boolean isValid(String s) {
        if(s.length()==0) return true;
        Deque<Character> stack = new ArrayDeque<>();
        int length = s.length();
        for (int i = 0; i < length; i++) {
            if (s.charAt(i) == '{' || s.charAt(i) == '(' || s.charAt(i) == '[') {
                stack.push(s.charAt(i));
            }
            else if (stack.size() != 0) {
                if (s.charAt(i) == ')' && stack.peek() == '(') stack.pop();
                else if (s.charAt(i) == '}' && stack.peek() == '{') stack.pop();
                else if (s.charAt(i) == ']' && stack.peek() == '[') stack.pop();
                else return false;
            }
            else {
                return false;
            }
        }
        return stack.size() == 0;
    }
}
```

6. ZigZag Conversion

题目：给定一个字符串和一个整数k，返回这个字符串的 `z` 字型字符串。
例子：

> 输入： `PAYPALISHIRING` 和数字3，字符串的 `z` 字型的排列是：
> P A H N
> A P L S I I G
> Y I R
> 然后输出 `PAHNAPLSIIGYIR` .

思路：构建一个 `StringBuilder` 组，大小是排列的行数，然后按照字符串中的 `z` 字型排列方式一个个存入相对应的StringBuilder。最后返回所有buffer的合并返回字符串。

```
public class Solution {
    public String convert(String s, int numRows) {
        int len = s.length();
        StringBuilder[] sb = new StringBuilder[numRows];
        for (int i = 0; i < numRows; i++) {
            sb[i] = new StringBuilder();
        }
        int i = 0;
        while (i < len) {
            for (int j = 0; j < numRows && i < len; j++) {
                sb[j].append(s.charAt(i++));
            }
            for (int j = numRows - 2; j >= 1 && i < len; j--) {
                sb[j].append(s.charAt(i++));
            }
        }
        for (int k = 1; k < numRows; k++) {
            sb[0].append(sb[k]);
        }
        return sb[0].toString();
    }
}
```

49. Group Anagrams

题目：给定一个字符串组，然后把这个字符串组按照anagram的方式重新组合在一起，也就是说具有相同的anagram的字符串放在一块儿。

思路：利用hashmap和anagram来构建一个具有相同anagram的字符串表。最后返回map的值就好了。

```
public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        if (strs == null || strs.length == 0) return null;
        int length = strs.length;
        Arrays.sort(strs);
        Map<String, List<String>> map = new HashMap<String, List<String>>();
        for (int i = 0; i < length; i++) {
            char[] temp = strs[i].toCharArray();
            Arrays.sort(temp);
            String key = new String(temp);
            if (!map.containsKey(key)) map.put(key, new ArrayList<String>());
            map.get(key).add(strs[i]);
        }
        return new ArrayList<List<String>>(map.values());
    }
}
```

22. Generate Parentheses

题目：给定一个整数n，要求返回所有合法的括号组合方式。
比如给定数字3，返回的合法括号组合是： `"((()))", "(()())", "(())()", "()(())", "()()()"`

思路：
利用回溯法。传递两个参数，分别代表左括号和右括号剩余的个数，如果右边括号的个数小于左边扩展的个数，那么肯定是不合法的组合，可以直接返回；如果左右括号数量都为0，证明组合完毕且是合法的；否则就递归调用；

```
public class Solution {
    public List<String> generateParenthesis(int n) {
        ArrayList<String> res = new ArrayList<String>();
        if(n<=0)
            return res;
        helper(n,n,new String(),res);
        return res;
    }
    private void helper(int l, int r, String item, ArrayList<String> res)  {
        if(r<l)
            return;
        if(l==0 && r==0)
        {
            res.add(item);
        }
        if(l>0)
            helper(l-1,r,item+"(",res);
        if(r>0)
            helper(l,r-1,item+")",res);
}
}
```

17. Letter Combinations of a Phone Number

题目：Given a digit string, return all possible letter combinations that the number could represent.

思路：典型的回溯方法。

```
public class Solution {
    public List<String> letterCombinations(String digits) {
        List<String> list = new ArrayList<String>();
        if (digits == null || digits.length() == 0) return list;
        String cur = "";
        helper(digits, list, cur, 0);
        return list;
    }

    public void helper(String digits, List<String> list, String cur, int start) {
        if (cur.length() == digits.length()) {
            list.add(new String(cur));
            return;
        }
        for (int i=start; i < digits.length(); i++) {
            String str = digit2Str(digits.charAt(i));
            for (int j = 0; j < str.length(); j++) {
                cur += str.charAt(j);
                helper(digits, list, cur, i+1);
                cur = cur.substring(0,cur.length()-1);
            }
        }
    }

    public String digit2Str(char a) {
        switch(a) {
            case '2':
                return "abc";
            case '3':
                return "def";
            case '4':
                return "ghi";
            case '5':
                return "jkl";
            case '6':
                return "mno";
            case '7':
                return "pqrs";
            case '8':
                return "tuv";
            case '9':
                return "wxyz";
            case '0':
            case '1':
            default:
            return "";

        }
    }
}
当然这里处理每个数字对应的字符串，也可以用字符串组的表达方式，这样会省下不少代码行。但是思路是一致的。
```

5. Longest Palindromic Substring

题目：给定一个数组，找出数组中最长的回文子串。

思路：这个题目我在另外一篇笔记中已经有详细解释，大体来说有两种解法，对于那种大神级别的解法我就不给出了。

- dp方法
  dp方法是维护一个二维的dp boolean组，dp[i][j]如果是true代表从字符串中i位置到j位置是回文子串，否则就不是；这里值得注意的就是预处理dp组的时候，对于长度为2时候，dp[i][i-1]应该设为true；
- 中心扩展法
  中心扩展法就比较直观，可以将字符串中任意一个字符当作可能回文的中心字符，然后不断往两边扩展；这里值得注意的是要考虑回文子串可能是偶数长度的情况；

(dp法)

```
    public String longestPalindrome(String s) {
        int length = s.length();
        int start = 0;
        int size = 1;
        boolean[][] dp = new boolean[length][length];
        dp[0][0] = true;
        for (int i = 1; i < length; i++) {
            dp[i][i] = true;
            dp[i][i-1] = true;
        }
        //字符串长度从2开始
        for (int k = 2; k <= length; k++) {
            for (int i = 0; i <= length - k; i++) {
                int j = i + k - 1;
                if (s.charAt(i) == s.charAt(j) && dp[i+1][j-1]) {
                    dp[i][j] = true;
                    start = i;
                    size = k;
                }
            }
        }
        return s.substring(start, size + start);
    }
```

(中心扩展法)

```
public String longestPalindrome(String s) {
        int length = s.length();
        if (length <= 1) return s;
        String ret = "";
        for (int i = 0; i < length; i++) {
            String odd = expand(s, i, i);
            if (odd.length() > ret.length()) {
                ret = odd;
            }
            String even = expand(s, i , i+1);
            if (even.length() > ret.length()) {
                ret = even;
            }
        }
        return ret;
    }
    public String expand(String s, int left, int right) {
        int length = s.length();
        while (left >=0 && right < length && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return s.substring(left + 1, right);
    }
```