最长回文子串(Longest Palindrome String)

1. 暴力解
   枚举字符串所可能出现的所有字串，然后判断是不是回文串。枚举字符串复杂度$O(n^2)$，但是判断回文是$O(n)$,所以总的算法复杂度是$O(n^3)$.
2. 动态规划
   动态规划的核心就在于找出状态之间的转移方程。本算法中，建立一个二维的动态数组用来记录当前字串是否为回文。

   - 初始化
     $dp[i][i] = true$; 单个字符当然是回文
     $dp[i][i-1] = true$; 这个主要是在字串长度为2且为回文时确保判定正确
     $dp[i][j] = false$;
   - 转移方程
     $dp[i][j] = s[i] == s[j] ? dp[i+1][j-1] : false$;
     解释：因为dp矩阵表示的是从下表i到j之间的字符串是否为回文，因此当字符串中的第i位和第j位相同时，我们就需要知道从$i+1$位到$j-1$位是否为回文。
   - DP步骤：
     - 枚举所有字串可能的长度，从2开始；
     - 然后枚举所有字串的起点下标，从0开始；

     因此这种方法时间复杂度$O(n^2)$，空间复杂度$O(n^2)$。

看代码：

```
1.      public String longestPalindromeString(String s) {
2.          int length = s.length;
3.          int start = 0;
4.          int size = 1; //回文子串长度至少是1.
5.          boolean[][] dp = new boolean[length][length];
6.          dp[0][0] = true;
7.          for (int i = 1; i < length; i++) {
8.              dp[i][i] = true;
9.              dp[i][i-1] = true;
10.         }
11.         //字符串长度从2开始
12.         for (int k = 2; k <= length; k++) {
13.             for (int i = 0; i < length - 1; i++) {
14.                 int j = i + k - 1;
15.                 if (s.charAt(i) == s.charAt(j) && dp[i+1][j-1]) {
16.                     dp[i][j] = true;
17.                     start = i;
18.                     size = k;
19.                 }
20.             }
21.         }
22.         return s.substring(start, size + start);
23.     }
```

3. 中心扩展
   中心扩展的思路比较简单，就是找到最大回文字符串中间字符，然后往两边扩展找到最大的回文字串。注意区分奇数和偶数情况。

   步骤：

   - 计数从0开始作为子串的中心字符，然后以此向两边扩展；
   - 比较原子串和新子串的长度，大于则更新。

看代码：

```
1.      public String longestPalindromeString(String s) {
2.          int length = s.length();
3.          if (length <= 1) return s;
4.          String ret = "";
5.          for (int i = 0; i < length; i++) {
6.              String odd = expand(s, i, i);
7.              if (odd.length() > ret.length()) {
8.                  ret = odd;
9.              }
10.             String even = expand(s, i , i+1);
11.             if (even.length() > ret.length()) {
12.                 ret = even;
13.             }
14.         }
15.         return ret;
16.     }
17.     public String expand(String s, int left, int right) {
18.         int length = s.length();
19.         while (left >=0 && right < length && s.charAt(left) == s.c
    harAt(right)) {
20.             left--;
21.             right++;
22.         }
23.         return s.substring(left + 1, right);
24.     }
```

时间复杂度还是$O(n^2)$,空间复杂度是$O(1)$.