5 最长回文子串

Label：字符串 动态规划
给你一个字符串 s，找到 s 中最长的回文子串

输入：s = "babad"
输出："bab"
解释："aba" 同样是符合题意的答案。

输入：s = "cbbd"
输出："bb"

输入：s = "a"
输出："a"

输入：s = "ac"
输出："a"

- 暴力法 （判断所有子串是否为回文）

```
class Solution {
    public String longestPalindrome(String s) {
        int len = s.length();

        if (len < 2) return s;

        int maxLenNow = 1;
        int begin = 0;
        char[] charArray = s.toCharArray();

        // 枚举所有长度大于 1 的子串
        for (int i = 0; i < len - 1; i++) {
            for (int j = i + 1; j < len; j++) {
                if (isPalindrome(charArray, i, j) && j - i + 1 > maxLenNow) {
                    maxLenNow = j - i + 1; // 目前最大回文的长度
                    begin = i;
                }
            }
        }
        return s.substring(begin, begin + maxLenNow);
    }


    private boolean isPalindrome(char[] charArray, int left, int right) {
        while (left < right) {
            if (charArray[left++] != charArray[right--] ) {
                return false;
            }
        }
        return true;
    }

}
```

- 动态规划
  - 一个回文去掉两头以后，剩下的部分依然是回文
  - 如果一个字符串的头尾两个字符都不相等，那么这个字符串一定不是回文串
  - 如果一个字符串的头尾两个字符相等，才有必要继续判断下去

```
public class Solution {

    public String longestPalindrome(String s) {
        // 特殊判断
        int len = s.length();

        if (len < 2) return s;

        int maxLen = 1;
        int begin = 0;

        // dp[i][j] 表示 s[i, j] 是否是回文串，记录状态
        boolean[][] dp = new boolean[len][len];
        char[] charArray = s.toCharArray();

        // 单个字符一定是回文串，因此把对角线先初始化为 true
        for (int i = 0; i < len; i++) {
            dp[i][i] = true;
        }


        for (int j = 1; j < len; j++) {
            for (int i = 0; i < j; i++) {

                if (charArray[i] != charArray[j]) { // 如果一个字符串的头尾两个字符都不相等，那么这个字符串一定不是回文串
                    dp[i][j] = false;
                } else {
                    if (j - i < 3) {  // 小于 3 直接下结论
                        dp[i][j] = true;
                    } else {
                        dp[i][j] = dp[i + 1][j - 1];  // 往中间缩小，依赖于中间缩小后的字串

                    }
                }

                if (dp[i][j] && j - i + 1 > maxLen) {
                    maxLen = j - i + 1;
                    begin = i;
                }
            }
        }
        return s.substring(begin, begin + maxLen);
    }
}
```