判断回文串单链表

最长回文串求解

> 给定一个字符串，求这个字符串中最长回文串。

分析： **寻找回文串的核心思想是：从中间向两边扩展来判断回文串。** 简单的框架：

```
for i = 0 to s.length() {
  寻找以s[i]为中心的回文串
  更新结果
}
```

因为回文串的长度可以为偶数也可以为奇数，框架可以变更为：

```
for i = 0 to s.length() {
  寻找以s[i]为中心的回文串
  寻找以s[i+1]为中心的回文串
  更新结果
}
```

可以写出代码：

```
class Solution {
public:
  std::string longestPalindrome(std::string s) {
    int         len = s.length();
    std::string res;
    for (int i = 0; i < len; i++) {
      std::string t1 = palindrome(s, i, i);
      std::string t2 = palindrome(s, i, i + 1);
      res            = (res.length() > t1.length() ? res : t1);
      res            = (res.length() > t2.length() ? res : t2);
    }

    return res;
  }

private:
  // 中心法判断回文串
  std::string palindrome(std::string &s, int lo, int hi) {
    while (lo >= 0 && hi < s.size() && s[lo] == s[hi]) {
      lo--;
      hi++;
    }
```

```
    // 返回以s[lo]和s[hi]为中心的回文串
    return s.substr(lo + 1, hi - lo - 1);
  }
};
```

回文串的核心思想是从中间向两端扩展：

```
// 对中心点的选择，可以有如下方式：
for (int i =0;i<s.length();i++) {
  std::string s = palindrome(s,i,i);
  std::string t = palindrome(s,i,i+1);
  res = (res.length() > s.length() ? res : s);
  res = (res.length() > t.length() ? res : t);
}

std::string palindrome(std::string &s, int lo, int hi) {
  // 防止索引越界
  while(lo >= 0 && hi < s.size() && s[lo] == s[hi]) {
    lo--;
    hi++;
  }

  // 返回以s[lo]和s[hi]为中心的回文串
  return s.substr(lo+1,hi-lo-1);
}
```

回文串单链表判断

给定一个单链表，判断单链表是否构成回文串。

分析：

1. 找到链表中点，将后半段逆置，判断前半段和后半段遍历是否一致。

```
class Solution {
public:
  bool isPalindrome(ListNode *head) {
    ListNode *slow = head, *fast = head;
    // slow为后半段第一个节点
    while (fast && fast->next) {
      slow = slow->next;
      fast = fast->next->next;
    }

    // 长度为奇数时，slow正好为中点
    if (fast) {
```

```
      slow = slow->next;
    }

    // 逆置
    ListNode *second = reserve(slow);

    // 判断是否问回文串
    return isPalindrome(head, second);
  }

private:
  bool isPalindrome(ListNode *head, ListNode *second) {
    while (second) {
      if (head->val != second->val) {
        return false;
      }
      head   = head->next;
      second = second->next;
    }

    return true;
  }

  ListNode *reserve(ListNode *head) {
    ListNode *dummy = new ListNode();
    while (head) {
      ListNode *p = head;
      head        = head->next;
      p->next     = dummy->next;
      dummy->next = p;
    }

    head = dummy->next;
    delete dummy;

    return head;
  }
};
```