分割问题就是组合问题

131.分割回文串

题目链接：https://leetcode-cn.com/problems/palindrome-partitioning/

给定一个字符串 s，将 s 分割成一些子串，使每个子串都是回文串。

返回 s 所有可能的分割方案。

示例:
输入: "aab"
输出:
[
["aa","b"],
["a","a","b"]
]

> 本题涉及到两个关键问题
> 1、**切割问题**，有不同的**切割方式**
> 2、**判断回文**

为什么说切割问题类似于组合问题?

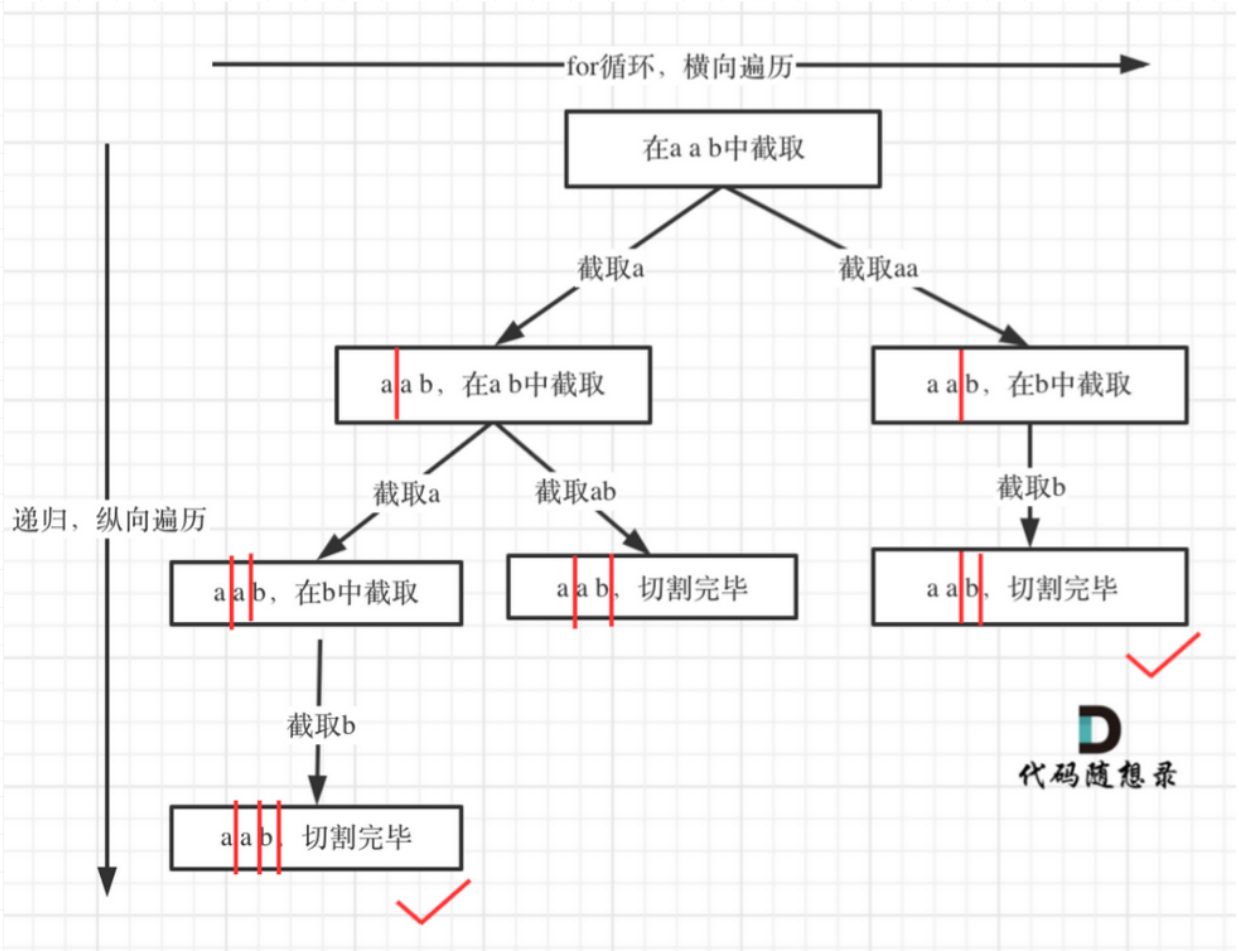
我们来分析一下切割，「其实切割问题类似组合问题」。

例如对于字符串abcdef：

- 组合问题：选取一个a之后，在bcdef中再去选取第二个，选取b之后在cdef中在选取第三个.....。
- 切割问题：切割一个a之后，在bcdef中再去切割第二段，切割b之后在cdef中在切割第三段.....。

感受出来了不？

> **递归**用来纵向遍历，**for循环**用来横向遍历，**切割线(图中的红线)**切割到**字符串的末尾位置**，说明找到了**一个切割方法.**



回溯三部曲:

step1:递归函数参数

全局变量数组path存放切割后回文的子串，result**存放的结果集**

```
vector<vector<string>> result;
vector<string> path;
void backtracking(string& s,int startIndex);
```

startIndex,表示**下一轮递归的起始位置**，这个startIndex就是**切割线**

step2:递归终止条件

```
void backtracking(const string& s,int startIndex){
 if(startIndex>=s.size){
 result.push_back(path);
 return ;
 }
}
```

step3:单层循环逻辑

「来看看在递归循环，中如何截取子串呢？」

在`for (int i = startIndex; i < s.size(); i++)`循环中，我们定义了起始位置startIndex，那么 [startIndex, i] 就是要截取的子串。

```cpp
class Solution {
public:
 //回溯算法的参数
 vector<vector<string>> result;
 vector<string>path;
 void backtracking(const string & s,int startIndex){
 if(startIndex>=s.size()){
 result.push_back(path);
 return;
 }
 for(int i=startIndex;i<s.size();i++){
 //切割的子串为[startIndex,i]
 if(isBalanced(s,startIndex,i)){
 path.push_back(s.substr(startIndex,i-startIndex+1));
 }else{
 continue;
 }
 backtracking(s,i+1);
 path.pop_back();
 }
 }
 bool isBalanced(const string& s,int startIndex,int i){
 for(int left=startIndex,right=i;left<=right;left++,right--){
 if(s[left]!=s[right]) return false;
 }
 return true;
 }
    vector<vector<string>> partition(string s) {
 if(s.size()==0) return result;
 backtracking(s,0);
 return result;
    }
};
```

那么难究竟难在什么地方呢？

「我列出如下几个难点：」

- 切割问题可以抽象为组合问题
- 如何模拟那些切割线
- 切割问题中递归如何终止
- 在递归循环中如何截取子串
- 如何判断回文

复原IP地址

93.复原IP地址

题目地址：https://leetcode-cn.com/problems/restore-ip-addresses/

给定一个只包含数字的字符串，复原它并返回所有可能的 IP 地址格式。

有效的 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 0），整数之间用 '.' 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 有效的 IP 地址，但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 无效的 IP 地址。

示例 1：
输入：s = "25525511135"
输出：["255.255.11.135","255.255.111.35"]

示例 2：
输入：s = "0000"
输出：["0.0.0.0"]

示例 3：输入：s = "1111"
输出：["1.1.1.1"]

本题就是一种新的的分割字符串，
与前文的分割字符串相比区别在于:
1、最终结果的path的元素个数一定要4个

step1:分割字符串参数

```
vector<string> result;
vector<string> path;//存放的是一个字符串分割后的子串集合,不包含逗号当要加入result时再加入逗号
void backtracking(const string s,int startIndex);
```

step2:递归终止条件

```
if(startIndex>=s.size()){
        if(path.size()==4){
            string s=path[0];
          for(int i=1;i<path.size();i++){
              s=s+"."+path[i];
          }
          result.push_back(s);
        }
        return;
}
```

step3:单层逻辑递归条件

```
for(int i=startIndex;i<s.size();i++){
    //所分割的字符串为[startInde,i]
```

```
        if(isValid(s,startIndex,i)){
            path.push_back(s.substr(startIndex,i-startIndex+1));
        } else{
            continue;
        }
        backtracking(s,i+1);
        path.pop_back();
    }
```

难点1需要判断子串是否合法?

1、段位以0开头的数字不合法

2、段位里有非正整数字符不合法

3、段位如果大于255了不合法

```
bool isValid(const string& s,int start,int end){
 long long num=0;
 if(start>end){
 return false;
 }
 if(s[start]=='0'&&start!=end){//0开头的数字不合法
 return false;
 }
 for (int i = start; i <= end; i++) {
         if (s[i] > '9' || s[i] < '0') { // 遇到非数字字符不合法
             return false;
         }
         num = num * 10 + (s[i] - '0');
         if (num > 255) { // 如果大于255了不合法,必须要中间就要判断不难会导致long long溢出
             return false;
         }
     }
     return true;
}
```