全排列问题

排列组合，组合类问题（全子集）学完后该来排列类问题啦！

全排列和全子集问题有点像，但是无法像全子集问题一样写二叉树决定它选不选在子集当中了，因为所有元素默认都在其中，只不过是顺序相关。所以只能用全子集的第二种方法，把元素一个一个加进去。

和全子集的dfs写法非常类似，除了全子集有一个startIndex而全排列变成了一个boolean的list来记录哪个数已经被visited过了，其他都是一样的。

```
public class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        if(nums == null) return result;
        dfs(nums, new boolean[nums.length], new ArrayList<Integer>(), result);
        return result;
    }
    private void dfs(int[] nums, boolean[] visited, List<Integer> subset,
List<List<Integer>> result){
        //递归出口
        if(nums.length == subset.size()){
            result.add(new ArrayList<Integer>(subset));
            return;
        }
        //递归拆解
        for(int i = 0; i<nums.length; ++i){
            if(visited[i]) continue;
            subset.add(nums[i]);
            visited[i] = true;
            dfs(nums, visited, subset, result);
            //一次尝试，撤回去才能尝试其他可能性
            subset.remove(subset.size()-1);
            visited[i] = false;
        }
    }
}
```

OK,现在的代码适用于没有重复元素的数组全排列， 那有重复元素和没有重复元素的代码有什么区别呢

代码不需要做太多改动，首先要对nums做一个sort，这样重复元素会被排在一起。然后给重复元素标个号，排在最前面的元素标一号，一号位没有被取到的情况下，二号位不能取，二号位没有被取到的情况下，三号位不能被取，以此类推，这样就不会出现重复的情况了：

```
public class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        if(nums == null && nums.length == 0) return result;
        Arrays.sort(nums);
        dfs(nums,new boolean[nums.length],new ArrayList<Integer>(), result);
        return result;
```

```java
    }
    private void dfs(int[] nums, boolean[] visited, ArrayList<Integer> subset,
List<List<Integer>> result){
        if(nums.length == subset.size()){
            result.add(new ArrayList<Integer>(subset));
            return;
        }

        for(int i = 0; i<nums.length; i++){
            if(visited[i] == true) continue;
            //原理：给相同的数标个号，当1号位用过了才能加2号位，如果一号位没用过，二号位以及后面的都不可以取
，所以continue。
            if(i>0&&nums[i-1] == nums[i]&&visited[i-1] == false){
                continue;
            }
            subset.add(nums[i]);
            visited[i] = true;
            dfs(nums,visited,subset,result);
            subset.remove(subset.size()-1);
            visited[i] = false;
        }
    }
}
```