题目

给定一个不含重复数字的数组 nums ，返回其 所有可能的全排列 。你可以 按任意顺序 返回答案。

示例 1：
输入：nums = [1,2,3]输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
示例 2：
输入：nums = [0,1]输出：[[0,1],[1,0]]
示例 3：
输入：nums = [1]输出：[[1]]

提示：
1 <= nums.length <= 6
-10 <= nums[i] <= 10
nums 中的所有整数 互不相同

这是一道经典的回溯算法问题。我们可以通过递归的方式进行求解。具体来说，我们不断地将没被选中的数加入到当前已选出的数列中，一直到所有的数都被选中，即找到了一个新的全排列

其中，used 数组用于标记数字是否被使用过，path 数组用于存储当前已选出的数列，res 数组用于存储所有的全排列。backtrack 方法表示回溯函数的实现，如果已经选取了所有数字就将其放入结果列表中，否则在未被使用的剩余数字中选择一个加入到当前已选出的数列中，然后对剩余的数字递归求解，当回溯到上一层时，需要撤销之前的选择，即从已选出的数列 path 中删除当前加入的数字，然后标记当前数字为未被使用过。最后，返回 res 数组即可。

在进行递归求解之前，需要初始化一些变量：

```java
List<List<Integer>> res = new ArrayList<>();  // 结果列表，存储所有全排列
List<Integer> path = new ArrayList<>();  // 存储当前已选出的数列
boolean[] used = new boolean[nums.length];  // 用于标记数字是否被使用过
```

其中，res 用于存储所有的全排列，它是一个二维列表，每个元素都是一个列表，表示一组全排列。path 用于存储当前已选出的数列，每次递归时，会向其中添加一些数字，直到得到一个新的全排列。used 数组用于标记数字是否被使用过，初始值为 false，当一个数字被加入 path 中时，会将对应的 used 值改为 true。

接下来是递归求解时的实现：

```java
private void backtrack(int[] nums, List<Integer> path, boolean[] used, List<List<Integer>> res) {
    if (path.size() == nums.length) {  // 如果已经选取了所有数字，将其放入结果中
        res.add(new ArrayList<>(path));
        return;
    }
    for (int i = 0; i < nums.length; i++) {
        if (!used[i]) {  // 如果数字没被使用过，则将其加入 path 中，并继续递归求解
            path.add(nums[i]);
            used[i] = true;
            backtrack(nums, path, used, res);  // 递归求解
            path.remove(path.size() - 1);  // 撤销之前的选择
            used[i] = false;
        }
    }
}
```

backtrack 方法是回溯函数的实现。它首先检查是否已经选取了所有数字，如果是，则将 path 加入 res，表示已经找到了一个新的全排列，然后返回。否则，会在未被使用的剩余数字中选择一个加入到 path 中，并对剩余的数字递归求解，等到回溯到上一层时，需要撤销之前的选择，即从 path 中删除当前加入的数字，然后标记当前数字为未被使用过。最后返回 res 即可。

最后，完整的 Java 代码：

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        List<Integer> path = new ArrayList<>();
        boolean[] used = new boolean[nums.length];  // 标记数字是否被使用过
        backtrack(nums, path, used, res);
        return res;
    }

    private void backtrack(int[] nums, List<Integer> path, boolean[] used, List<List<Integer>> res)
{
        if (path.size() == nums.length) {
            res.add(new ArrayList<>(path));  // 如果已经选取了所有数字，将其放入结果
中
            return;
        }
        for (int i = 0; i < nums.length; i++) {
```

```
                if (!used[i]) {
                    path.add(nums[i]);
                    used[i] = true;
                    backtrack(nums, path, used, res);    // 递归
                    path.remove(path.size() - 1);
                    used[i] = false;
                }
            }
        }
    }
```