目录

0.1 问题的引出

前几日有个朋友问到一个问题:

> A、B、C、D、E、F 六个字符, 从中随便选三个字符组成的字符串数组结果 (共 6*6*6 种), 允许重复 (如 AAA、AAC).

这个问题的处理办法有很多, 通过网络可以找到多种解答.

1. 使用循环求解
2. 使用递归求解

那么本文除了简单分析这些方法外, 还给出一个更为简单的办法处理. 该问题对应的数学模型是: 求 6 个元素中可重复的取出 3 个元素的所有排列有多少?

定义映射:

$$A \mapsto 0$$

$$B \mapsto 1$$

$$C \mapsto 2$$

$$D \mapsto 3$$

$$E \mapsto 4$$

$$F \mapsto 5$$

将问题转化成从集合 $S = \{0, 1, 2, 3, 4, 5\}$ 中取出三个数字的办法.

0.2 循环求解

由于是可重复的选择, 并且不考虑顺序. 可以将问题修改为: 有三个盒子, 每一个盒子中可以放 A, B, C, D, E, F 六个元素中的一个. 也就是说每一个盒子有 6 六种可能. 根据乘法原理, 一共有: $6 \times 6 \times 6 = 216$ 种情况.

最简单的办法就是利用循环[1], 使用一个循环, 可以给出在 6 个数字中取出一个数字的所有情况.

```
var res = [];
for ( var i = 0; i < 6; i++ ) {
  res.push( [ i ] )
}
```

那么, 如果是从六个数中取两个呢? 可以考虑两个循环来完成.

```
var res = [];
for ( var i = 0; i < 6; i++ ) {
  for ( var j = 0; j < 6; j++ ) {
    res.push( [ i, j ] );
  }
}
```

如此很显然, 如果要找 3 个就需要 3 个循环. 如果是 n 个就需要 n 个循环.

下面给出符合题意的代码:

```
var res = [];
for ( var i = 0; i < 6; i++ ) {
  for ( var j = 0; j < 6; j++ ) {
    for ( var k = 0; k < 6; k++ ) {
      res.push( [ i, j, k ] );
    }
  }
}
```

0.3 递归求解

如果是使用循环, 那么没有办法将其封装起来. 因为要嵌套几次循环是不清楚的. 那么如果需要将其封装成函数 `C(6,3)` 来计算就可以使用递归.

递归的重点是递归体和临界条件. 在这个问题中首先来看递归体.

[1]本文代码均使用 JavaScript 语法. 因为该语言环境简单, 也提供了较友好的集合操作.

如果是在 6 个元素中取 1 个, 那么很简单循环即可. 如果是在 6 个元素中取 2 个, 就是循环一次, 再循环一次. 所以在取元素的操作中重复进行的就是 “在 6 个元素中取 1 个元素” 这个过程. 所以自然就清楚函数 C(n,m) 的功能了.

C(n,m) 首先在 n 个元素中取 m 个元素, 然后返回取得的结果的数组. 那么 C(6,1) 的结果就是一个数组. 可以有代码实现为:

```
var C = function ( n, m ) {
  var res = [];
  for ( var i = 0; i < n; i++ ) {
    res.push( i );
  }
  return res;
};
```

这段代码就是在取 1 个元素. 那么了解到一个元素的情况, 如果是取两个元素呢?

实际上取两个和取多个是一样的. 可以让函数先取一个元素, 而另一个元素的获取由递归完成, 即

```
var C = function ( n, m ) {
  var res = [], firstNum, secondNumArray = [], k, i;
  for ( i = 0; i < n; i++ ) {
    firstNum = i;
    if ( m > 1) {
      secondNumArray = C( n, m - 1 );
      for ( k in secondNumArray ) {
        secondNumArray[ k ].push( firstNum );
        res.push( secondNumArray[ k ] );
      }
    } else {
      res.push( [ firstNum ] );
    }
  }
  return res;
```

```
16 };
```

0.4 进制转换求解

前面的几种解法都比较的复杂. 使用进制转换的办法是最简单的.

已知需要取 3 个数字. 就可以准备 3 个盒子, 然后在每一个盒子里面放数字, 允许使用 $0,1,2,3,4,5$ 这六个数字. 很显然这就是 3 位的 6 进制数. 现在的问题是, 有几种情况, 分别是多少, 就是在求 0 到 $6\times6\times6$ 的所有数中, 其 6 进制数的排列情况.

准备代码

```
var i1, i2, i3, res = [], i = 0,
  max = 6 * 6 * 6;
for ( ; i < max; i++ ) {
  i1 = i % 6;
  i2 = Math.floor( i / 6 ) % 6;
  i3 = Math.floor( i / 36 ) % 6;
  res.push( [ i1, i2, i3 ] );
}
```

当然, 这里依旧可以将其再进行封装.