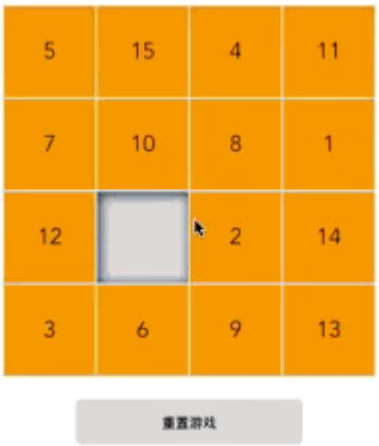
开发指南篇4：数据驱动与拼图游戏

数据驱动是Vue 框架的核心特性之一，也是Vue 响应式原理的具体体现，相信大家对其应该深有体会，尤其是在操作数据来触发页面更新的时候。

为了让大家更加了解数据驱动的理念，并解决使用过程中可能出现的一系列问题，本文将结合比较常见和简单的“拼图游戏”来展示Vue 数据驱动的魅力所在。

效果展示

首先我们先来看一下实现的“拼图游戏”的动态效果：



在不操作 DOM 的情况下实现以上功能其实需要我们对Vue数据驱动及数据可视化有一个非常清楚的认知，在操作数据的同时驱动可视化界面的还原。

关键代码

接下来我们来看一下实现该拼图游戏的功能点及关键代码：

```
<!-- HTML 部分 -->
<ul class="puzzle-wrap">
    <li
        :class="{'puzzle': true, 'puzzle-empty': !puzzle}"
        v-for="(puzzle, index) in puzzles"
        :key="index"
        v-text="puzzle"
    ></li>
</ul>
```

javascript

```
// 数据部分
export default {
    data() {
        return {
            puzzles: Array.from({ length: 15 }, (value, index) => index + 1)
        }
    },
}
```

上方我们使用 v-for 循环构建了从1 ～ 15按顺序排列的方块格子，也就是拼图完成时候的顺序，但是拼图游戏一开始数字的顺序应该是无序的，也是随机打乱的，那么我们怎么实现呢？可以使用下方的随机排列函数：

javascript

```
function shuffle(arr) {
    let len = arr.length

    for (let i = 0; i < len - 1; i++) {
        let idx = Math.floor(Math.random() * (len - i))
        let temp = arr[idx]
        arr[idx] = arr[len - i - 1]
        arr[len - i - 1] = temp
    }

    return arr
}
```

该函数中我们使用 Math.random() 来返回0和1之间的伪随机数，可能为0，但总是小于1，[0, 1)，而通过这一特性我们可以实现生成nm，包含n但不包含m的整数，具体步骤如下：

- 第三步 `Math.random() * w + n`
- 第四步 `Math.floor(Math.random() * w + n)`

在 `shuffle` 函数中n值永远是0，而w（即len – i）值随着循环i值的变大而不断减小。

> 在上面的算法里，我们每一次循环从前len – i 个元素里随机一个位置，将这个元素和第len – i 个元素进行交换，迭代直到i = len – 1 为止。

这一便实现了数组的随机打乱。最后我们需要在数组末尾追加一个空值来显示唯一一个空白格子：

```javascript
this.puzzles.push('');
```

交换方块位置

实现随机数字后，当我们点击方块，如果其上下左右存在为空的格子就需要将其进行交换，而由于是数据驱动界面，这里我们便需要交换两者在数组中的位置来实现：

```javascript
export default {
    methods: {

        // 点击方块
        moveFn(index) {
            let puzzles = this.puzzles

            // 获取点击位置上下左右的值
            let leftNum = this.puzzles[index - 1],
                rightNum = this.puzzles[index + 1],
                topNum = this.puzzles[index - 4],
                bottomNum = this.puzzles[index + 4]

            // 和为空的位置交换数值
            if (leftNum === '' && index % 4) {
                this.setPuzzle(index, -1)
            } else if (rightNum === '' && 3 !== index % 4) {
                this.setPuzzle(index, 1)
            } else if (topNum === '') {
```

```
            }
        },

        // 设置数组值
        setPuzzle(index, num) {
            let curNum = this.puzzles[index]

            this.$set(this.puzzles, index + num, curNum)
            this.$set(this.puzzles, index, '')
        },
    }
}
```

由于是 16 宫格的拼图，所以我们在点击获取位置的时候需要考虑边界情况，比如第 4 个格子为空，我们点击第 5 个格子不应该交换它们，因为在界面上第 4 个格子不在第 5 个格子的左侧，所以我们使用 `index % 4` 的方法来进行边界的判断，同时使用 Vue 提供的 `$set` 方法来将响应属性添加到数组上。

校验是否过关

最后我们需要校验游戏是否过关，我们只需要在最后一个格子为空时去进行校验即可：

```javascript
if (this.puzzles[15] === '') {
    const newPuzzles = this.puzzles.slice(0, 15)
    const isPass = newPuzzles.every((e, i) => e === i + 1)

    if (isPass) {
        alert ('恭喜，闯关成功！')
    }
}
```

我们使用数组的 `every` 方法来简化代码的复杂度，当所有数字大小和对应的数组下标 + 1 相吻合时即会返回 `true` 。

如此我们便完成了一个简单拼图游戏的功能。

盲点及误区

数组随机打乱为什么不用 sort 排序呢？下面便来进行讲解：

为什么要用 $set 方法

大家应该都知道如果不用 `$set` 方法我们可以直接通过操作数组索引的形式对数组进行赋值，从而交换拼图的中两者的数据：

```javascript
// 设置数组值
setPuzzle(index, num) {
    let curNum = this.puzzles[index]

    this.puzzles[index + num] = curNum
    this.puzzles[index] = ''

    // this.$set(this.puzzles, index + num, curNum)
    // this.$set(this.puzzles, index, '')
}
```

但是你会发现这样做数据是改变了，但是页面并没有因此重新渲染，这是为什么呢？其实 Vue 官方已经给出了明确的答案：

> 由于 JavaScript 的限制，Vue 不能检测以下变动的数组：
>
> - 当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue
> - 当你修改数组的长度时，例如：vm.items.length = newLength

我们这里使用的便是第一种利用索引的方式，由于 Vue 检测不到数组变动，因此页面便无法重绘。同样 Vue 也不能检测对象属性的添加或删除，需要使用 `Vue.set(object, key, value)` 方法来实现。

其实还有一种比较取巧的方式便是强制重新渲染 Vue 实例来解决这一问题：

```javascript
// 设置数组值
setPuzzle(index, num) {
    let curNum = this.puzzles[index]

    this.puzzles[index + num] = curNum
```

```
        // this.$set(this.puzzles, index + num, curNum)
        // this.$set(this.puzzles, index, '')
    }
```

上方我们使用了 Vue 提供的 `$forceUpdate` 方法迫使 Vue 实例重新渲染，这样改变的数据就会被更新的页面中去。但是最好不要这样操作，因为这会导致 Vue 重新遍历此对象所有的属性，一定程度上会影响页面的性能。

为什么不用 sort 排序

其实 sort 方法也能够实现数组的随机排序，代码如下：

```javascript
let puzzleArr = Array.from({ length: 15 }, (value, index) => index + 1);

// 随机打乱数组
puzzleArr = puzzleArr.sort(() => {
    return Math.random() - 0.5
});
```

我们通过使用 `Math.random()` 的随机数减去 0.5 来返回一个大于、等于或小于 0 的数，sort 方法会根据接收到的值来对相互比较的数据进行升序或是降序排列。

但是由于 JavaScript 内置排序算法的缺陷性，使用 sort 排序的结果并不随机分布，经过大量的测试你会发现**越大的数字出现在越后面的概率越大**。

由于本文并非是一篇介绍 sort 排序的文章，关于论证其缺陷性的话题这里就不进行详细展开了，感兴趣的同学可以进一步进行探究。

结语

本文实例是基于我之前写的一篇关于利用 Vue.js 实现拼图游戏的文章上进行了改进和优化，希望通过这样一个小游戏来强化大家对于 Vue 数据驱动的理解。相比操作 DOM 元素，操作数据其实更加的便捷和快速，可以使用较少的代码来实现一些较为复杂的逻辑。

思考 & 作业

- Vue 中监听数据变化的原理是什么？是通过何种方式实现的?
- 如何论证原生 JS 中 sort 排序后越大的数字出现在越后面的概率越大?
- 如何使用 `Math.random()` 生成 n–m，不包含 n 但包含 m 的整数?

留言

评论将在后台进行审核，审核通过后对所有人可见

warhol 前端工程师 @ 武汉华大国数

shuffle 函数内部的变量交换可以是使用es6的简洁语法实现,更加一目了然
[arr[idx], arr[len – i – 1]] = [arr[len – i – 1], arr[idx]];

▲ 0　　评论　15天前

Si

1. 原理是 Object.defineProperty 的 get 和 set, vue 3.0 用 proxy 实现
3. 不会，有没有大佬讲一下

▲ 0　　收起评论　3月前

> **Si**
> 3. Math.random()*m + n
> 3月前
>
> **劳卜** 前端工程师 @ TC
> Math.floor(Math.random()*w+n) + 1
> 3月前
>
> **漆伟** 前端工程师

评论审核通过后显示

评论

CodeBearsh 前端

随机打乱的数值能保证可以复原吗

▲ 0 收起评论 3月前

劳卜 前端工程师 @ TC

本案例里不需要复原，要的话在打乱前记录一下就好了

3月前

评论审核通过后显示

评论