Array.prototype.reduce() 用法

`reduce()` 方法对数组中的每个元素按序执行一个提供的 reducer 函数，每一次运行 reducer 会将先前元素的计算结果作为参数传入，最后将其结果汇总为单个返回值。

```
// 用法
reduce(callbackFn); // 无初始值情况
reduce(callbackFn, initialValue); // 有初始值情况
```

`callbackFn` 的参数有四个，和其他数组方法 map 相比，多了第一个参数（累加器），第 2、3、4 个参数分别是当前值、当前值索引，数组本身。

1. 求数值数组中值的总和

```
// 无初始值
const array = [15, 16, 17, 18, 19];
function reducer(accumulator, currentValue, index, array) {
  const returns = accumulator + currentValue;
  console.log(`accumulator: ${accumulator}, currentValue: ${currentValue}, index:
${index}, array: ${array}, returns: ${returns}`);
  return returns;
}
array.reduce(reducer);
```

	accumulator	currentValue	index	返回值
第一次调用	15	16	1	31
第二次调用	31	17	2	48
第三次调用	48	18	3	66
第四次调用	66	19	4	85

- 无初始值时，index 从 1 开始，因为第一次调用函数第一个参数使用数组中第一个成员的值，函数第二个参数使用数组中第二个成员的值，表示当前值，index 表示当前值所在的索引。
- array 参数在整个过程中始终不会改变——它始终是 [15, 16, 17, 18, 19]。
- reduce() 返回的值将是最后一次回调返回值（85）。

```
// 有初始值
const array = [15, 16, 17, 18, 19];
function reducer(accumulator, currentValue, index) {
  const returns = accumulator + currentValue;
  console.log(`accumulator: ${accumulator}, currentValue: ${currentValue}, index:
${index}, returns: ${returns}`);
  return returns;
```

```
}
array.reduce(reducer, 10);
```

	accumulator	currentValue	index	返回值
第一次调用	10	15	0	25
第二次调用	25	16	1	41
第三次调用	41	17	2	58
第四次调用	58	18	3	76
第五次调用	76	19	4	95

2. 按属性对对象进行分组

```
let list = [
  {
      "meetingId": 718,
      "meetingName": "测试注销",
      "beginTime": "2023-09-08",
  },
  {
      "meetingId": 698,
      "meetingName": "测试会议 1",
      "beginTime": "2023-08-25",
  },
  {
      "meetingId": 680,
      "meetingName": "测试新会议",
      "beginTime": "2023-08-03",
  },
  {
      "meetingId": 676,
      "meetingName": "测试连屏会议",
      "beginTime": "2023-07-31",
  },
  {
      "meetingId": 668,
      "meetingName": "刚回家成华",
      "beginTime": "2023-07-27",
  }
]
function groupByList(objectArray, property) {
  return objectArray.reduce((acc, obj) => {
    const key = obj[property];
    const curGroup = acc[key] ?? [];

    return { ...acc, [key]: [...curGroup, obj] };
  }, {});
```

```
}

const groupedList = groupByList(people, "beginTime");
console.log(groupedList);

// {
//     "2023-09-08": [
//         {
//             "meetingId": 718,
//             "meetingName": "测试注销",
//             "beginTime": "2023-09-08"
//         }
//     ],
//     "2023-08-25": [
//         {
//             "meetingId": 698,
//             "meetingName": "测试会议 1",
//             "beginTime": "2023-08-25"
//         }
//     ],
//     "2023-08-03": [
//         {
//             "meetingId": 680,
//             "meetingName": "测试新会议",
//             "beginTime": "2023-08-03"
//         }
//     ],
//     "2023-07-31": [
//         {
//             "meetingId": 676,
//             "meetingName": "测试连屏会议",
//             "beginTime": "2023-07-31"
//         }
//     ],
//     "2023-07-27": [
//         {
//             "meetingId": 668,
//             "meetingName": "刚回家成华",
//             "beginTime": "2023-07-27"
//         }
//     ]
// }
```

3. 自己实现 reduce 的功能

```
Array.prototype.myReduce = function(callback, initialValue) {
    // 初始化累加器的值
    let accumulator = initialValue;
    if (typeof initialValue === 'undefined') {
        // 如果没有提供初始值，则累加器的初始值为数组的第一个元素
        // 并且从数组的第二个元素开始迭代
```

```
            accumulator = this[0];
            let startIndex = 1;
        } else {
            // 如果提供了初始值，则从数组的第一个元素开始迭代
            let startIndex = 0;
        }

        // 遍历数组，从 startIndex 开始
        for (let i = startIndex; i < this.length; i++) {
            // 调用回调函数，并传入累加器当前值、当前元素、当前元素的索引和原数组
            accumulator = callback(accumulator, this[i], i, this);
        }

        // 返回累加器的最终值
        return accumulator;
    };
```