5.数组常用方法背后的面试点

1. 核心概念 (Core Concept)

JavaScript 的 `Array.prototype` 上提供了大量内置方法，用于遍历、转换和操作数组。面试中常考的方法主要围绕其**功能**、**返回值**、**是否改变原数组 (mutability)** 以及**与其它方法的对比**。

关键方法分类：

- **迭代方法**: `forEach`, `map`, `filter`, `reduce`, `some`, `every`
- **修改原数组 (Mutator methods)**: `push`, `pop`, `shift`, `unshift`, `splice`, `sort`, `reverse`
- **返回新数组 (Non-mutating methods)**: `concat`, `slice`, `map`, `filter`, `reduce`

2. 为什么需要深入理解它们？ (The "Why")

1. **编写高效、简洁的代码**: 熟练运用这些高阶函数，可以替代复杂的 `for` 循环，使代码更具可读性和表现力。
2. **避免副作用**: 清晰地知道哪些方法会修改原数组，是防止程序出现意外状态变更的关键，尤其是在函数式编程和状态管理（如 React/Redux）中。
3. **考察基础编程能力**: 对这些常用方法的熟练程度和细节的掌握，是衡量一个开发者 JS 基础是否扎实的常用指标。例如，能否手动实现 `map` 或 `reduce`。

3. API 与用法 (API & Usage)

`map()`

- **核心功能**: 创建一个新数组，其结果是该数组中的每个元素都调用一个提供的函数后返回的结果。
- **返回值**: 一个新的、长度与原数组相同的新数组。
- **是否改变原数组**: 否。

```
const numbers = [1, 4, 9];
const doubles = numbers.map(num => num * 2);

console.log(doubles); // 输出: [2, 8, 18]
console.log(numbers); // 输出: [1, 4, 9] (原数组不变)
```

`filter()`

- **核心功能**: 创建一个新数组, 其包含通过所提供函数实现的测试的所有元素。

- **返回值**: 一个新的、可能比原数组短的新数组。
- **是否改变原数组**: 否。

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction'];
const result = words.filter(word => word.length > 6);

console.log(result); // 输出: ["exuberant", "destruction"]
```

`reduce()`

- **核心功能**: 对数组中的每个元素执行一个 "reducer" 函数（升序执行），将其结果汇总为单个返回值。
- **返回值**: 函数累计处理的结果。

API: `arr.reduce(callback(accumulator, currentValue, currentIndex, array), initialValue)`
- `accumulator` (累加器): 累计回调的返回值; 它是上一次调用回调时返回的累积值，或 `initialValue`。
- `initialValue` (可选): 作为第一次调用 `callback` 函数时的第一个参数的值。

```
const array = [1, 2, 3, 4];
const initialValue = 0;
const sum = array.reduce(
(accumulator, currentValue) => accumulator + currentValue,
initialValue
);

console.log(sum); // 输出: 10
```

`splice()`

- **核心功能**: 通过删除或替换现有元素或者原地添加新的元素来修改数组。
- **返回值**: 一个由被删除的元素组成的新数组。如果未删除任何元素，则返回空数组。
- **是否改变原数组**: 是。

```
const months = ['Jan', 'March', 'April', 'June'];
// 在索引 1 的位置，删除 0 个元素，插入 'Feb'
const removed = months.splice(1, 0, 'Feb');

console.log(months);  // 输出: ["Jan", "Feb", "March", "April", "June"]
console.log(removed); // 输出: [] (没有元素被删除)

// 从索引 3 的位置，删除 1 个元素
const removedItem = months.splice(3, 1);
```

```
console.log(months);      // 输出: ["Jan", "Feb", "March", "June"]
console.log(removedItem); // 输出: ["April"]
```

4. 关键注意事项 (Key Considerations)

1. **`map` vs `forEach`**:
   - **返回值**: `map` 返回一个新数组，`forEach` 返回 `undefined`。
   - **用途**: 当你需要**转换**数据并得到一个新数组时用 `map`。当你只需要对数组中每个元素执行一个**操作**（如打印、调用 API）而不需要返回值时用 `forEach`。
   - **链式调用**: `map` 可以进行链式调用 (`arr.map(...).filter(...)`)，`forEach` 不行。
2. **`reduce` 的 `initialValue`**:
   - **提供 `initialValue`**: `reduce` 会从数组的第一个元素开始执行回调，`accumulator` 的初始值就是 `initialValue`。
   - **不提供 `initialValue`**: `reduce` 会从数组的第二个元素开始执行回调，`accumulator` 的初始值是数组的**第一个元素**，`currentValue` 是第二个元素。对空数组调用 `reduce` 且没有 `initialValue` 会抛出 `TypeError`。
3. **可变性 (Mutability) 是核心考点**:
   - **修改原数组**: `push`, `pop`, `shift`, `unshift`, `splice`, `sort`, `reverse`。在 React 等场景下应避免直接对 state 数组使用这些方法。
   - **不修改原数组**: `map`, `filter`, `reduce`, `slice`, `concat`。这些是创建新状态的首选方法。
   - **`slice` vs `splice`**: `slice` 返回数组的**浅拷贝**片段，不修改原数组。`splice` 修改原数组，用于添加/删除元素。名称相似但功能和副作用完全不同。
4. **稀疏数组的处理**:
   - `map`, `forEach`, `filter` 等迭代方法会**跳过**稀疏数组中的空位。
   - `reduce` 也会跳过空位（除非提供了 `initialValue`）。

5. 参考资料 (References)

- MDN Web Docs: Array
- MDN Web Docs: Array.prototype.map()
- MDN Web Docs: Array.prototype.reduce()
- MDN Web Docs: Array.prototype.splice()
- ECMAScript® 2025 Language Specification: Array Objects