面试官：请问你在开发过程中如何实现数组去重的

前言

这两天一面 B 站的时候被问到了这个问题，起初我笑了笑，问这么简单的问题，但是等面试官写好题目后我才发现了问题好像不简单

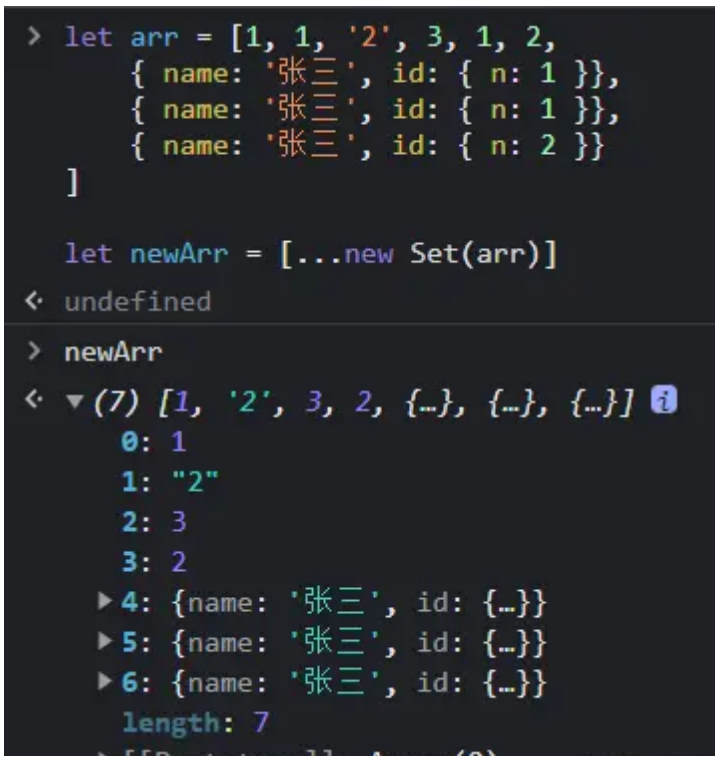
先看下面试官给的什么数组吧

```
let arr = [1, 1, '2', 3, 1, 2,
    { name: '张三', id: { n: 1 }},
    { name: '张三', id: { n: 1 }},
    { name: '张三', id: { n: 2 }}
]
```

实际开发中我们前端拿到后端的数据，有时候可能有些数据重复了，比如这里的三个张三，明显，公司里面有两个张三，因为可能确实重名，但是人家的 id 不同，不过第一个和第二个明显重复了，因此我们需要把第二个张三去掉

平时刷力扣，我们做的数组去重可能就是简单的对原始数据进行去重，但是实际开发会碰上引用类型

若是原始类型我们可以直接用 es6 新增的 set，我们可以先看下若用 set 去重这个数组会怎样



额~，看样子 set 仅仅对原始类型进行了去重，其实 set 去重的核心逻辑是通过 `===` 来判断的

因此 set 只能对原始类型进行去重

两个引用类型是否相等，首先就是判断引用地址是否相等

实现

方法一

其实实现面试官给你的题目进行去重也很容易想到，既然引用类型一定地址不同，那么我就将其转换成字符串，也就是 JSON 格式，然后 set 去重，最后转回来

```
let arr = [1, 1, '2', 3, 1, 2,
    { name: '张三', id: { n: 1 }},
    { name: '张三', id: { n: 1 }},
    { name: '张三', id: { n: 2 }}
]

let arr2 = arr.map((item) => {
    return JSON.stringify(item)
})

let newArr = new Set(arr2)
newArr = Array.from(newArr).map((item) => {
    return JSON.parse(item)
})

console.log(newArr);
// [
//     1,
//     '2',
//     3,
//     2,
//     { name: '张三', id: { n: 1 } },
//     { name: '张三', id: { n: 2 } }
// ]

```

好像不错可以实现，但是这时候面试官又问我，不用 JOSN，你还有别的手段可以实现吗

方法二

那就换个思路去重，自己写个函数，遍历数组，遍历的过程中维护一个 res 数组，如果 res 数组中不存在当前项，那就 push 进去，若存在则跳过

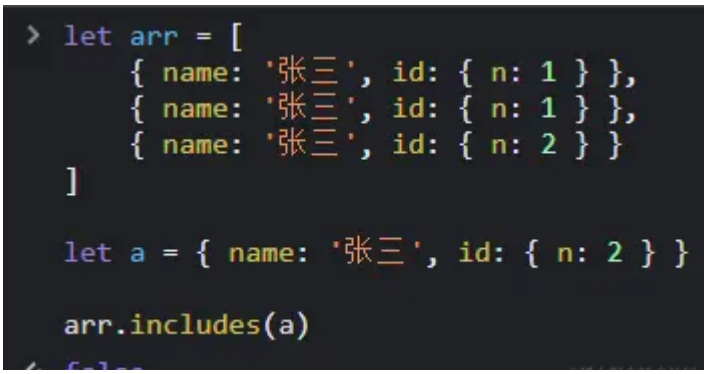
```
let arr = [1, 1, '2', 3, 1, 2,
    { name: '张三', id: { n: 1 } },
    { name: '张三', id: { n: 1 } },
    { name: '张三', id: { n: 2 } }
]

function uniqueArr (arr) {
    let res = []
    for (let item of arr) {

    }
}

```

这里存在的问题主要是如何判断相同的对象是否存在，我们先看下直接用 `includes` 是否行得通



false， `includes` 判断不了，毕竟地址不同，可以理解

既然如何，那我就自己实现一个辅助函数，用来判断两个数是否相等，还要针对引用数据类型

```
function uniqueArr (arr) {
    let res = []
    for (let item of arr) {
        for (let resItem in res) {
            if (equal(item, resItem)) {
                break;
            }
        }
    }
}
```

for 循环用的 `for of` 因为数组具有迭代器属性，一般数组遍历用这个方法更多。如果发现了当前项和 res 中的某一项是相同的，那么 res 后面的数据就没有必要再进行遍历了，因此终止掉 res 的 for 循环即可，这里用 `break`，若用 `return` 就是终止掉了所有的 for 循环了

> `continue` 是终止当前一次的 for 循环，跳到下一次 for 循环

什么时候 push 呢，一定不是 else，因为判断唯一一定是需要让 resItem 全部遍历完保证每一项都不同才能 push，因此这个 push 写在第二层 for 循环的外面

但是直接写在外面又有个缺陷，就是里面的 break 终止了 for 循环，外面的 push 依旧会执行

既然这样那我就搞一个开关变量 isFind 放在第一层 for 循环中，如果 equal 了，那么就将这个 isFind 置为 true，然后 push 就写在当 isFind 为 false 时的判断中

```
function uniqueArr (arr) {
    let res = []
    for (let item of arr) {
        let isFind = false
        for (let resItem in res) {
            if (equal(item, resItem)) {
                isFind = true
            }
        }
        if (!isFind) res.push(item)
    }
    return res
}
```

好，现在来实现辅助函数，判断只要长得一样就是相等，尤其是引用类型，在 v8 看来，地址不同就是不同

equal

`equal` 实现起来也很简单，真要判断引用类型就一定是双方都是引用类型，如果都不是引用类型或者只有一方是引用类型就是直接判断即可

```
function equal(v1, v2) {
    if ((typeof v1 === 'object' && v1 !== null) && (typeof v2 === 'object' && v2 !== null)) { // 都是引用类型

    } else { // 都不是引用类型、一方是引用类型
        return v1 === v2
    }
}
```

如果双方都是对象，那就需要去判断二者的 `key-value` 是否都能一一对的上

如果 v1 具有的 key，v2 也具有，那就去看 value，这里用 `v2.hasOwnProperty(key)` 来实现，如果不是，key 都不同就直接 return，如果 v2 确实也有，那就再判断，这里再次判断需要注意，value 也有可能还是个对象，那就递归

```
function equal(v1, v2) {
    if ((typeof v1 === 'object' && v1 !== null) && (typeof v2 === 'object' && v2 !== null)) { // 都是引用类型

        for (let key in v1) {
            if (v2.hasOwnProperty(key)) { // 只要v1遍历的东西，V2显示具有就再去看value
                // 有可能value也是引用类型，那就递归下
                if (!equal(v1[key], v2[key])) {
                    return false
                }
            } else {
                return false
            }
        }
    } else { // 都不是引用类型、一方是引用类型 同样这也是递归的出口
        return v1 === v2
    }
}
```

如果两个对象长得完全一样，那就意味着 for 循环，所有的递归走完了，也不会走到里面的 else，那就需要 for 循环结束后加一个返回 true

另外我们还可以进行优化，两个对象的 key 的数量若是不一致就是不一致

如何拿到对象的 key 的数量：`Object.keys(obj)`，这个方法会以数组的形式返回对象的 key，所以直接 `.length` 判断即可

最终代码实现如下

```
let arr = [1, 1, '2', 3, 1, 2,
    { name: '张三', id: { n: 1 } },
    { name: '张三', id: { n: 1 } },
    { name: '张三', id: { n: 2 } }
]

function uniqueArr (arr) {
    let res = []
    for (let item of arr) {
        let isFind = false
        for (let resItem of res) {
```

```
            if (equal(item, resItem)) {
                isFind = true
                break
            }
        }
        if (!isFind) res.push(item)
    }
    return res
}

function equal(v1, v2) {
    if ((typeof v1 === 'object' && v1 !== null) && (typeof v2 === 'object' && v2 !== null)) { // 都是引用类型
        if (Object.keys(v1).length !== Object.keys(v2).length) return false
        for (let key in v1) {
            if (v2.hasOwnProperty(key)) { // 只要v1遍历的东西，V2显示具有就再去看value
                // 有可能value也是引用类型，那就递归下
                if (!equal(v1[key], v2[key])) {
                    return false
                }
            } else {
                return false
            }
        }
        return true // 两个对象长得完全一样
    } else { // 都不是引用类型、一方是引用类型 同样这也是递归的出口
        return v1 === v2
    }
}

console.log(uniqueArr(arr));

```

最后

因此，面试官问你数组去重的时候，得看情况，若仅有原始类型，可以直接用 set 这个数据结构，若有引用类型，你也可以用 JSON 转化一下，当然，你也可以自己手搓一个去重函数，就是需要类型判断再递归