1. 循环删除 newDataSource.splice(index, 1);splice改变原数组

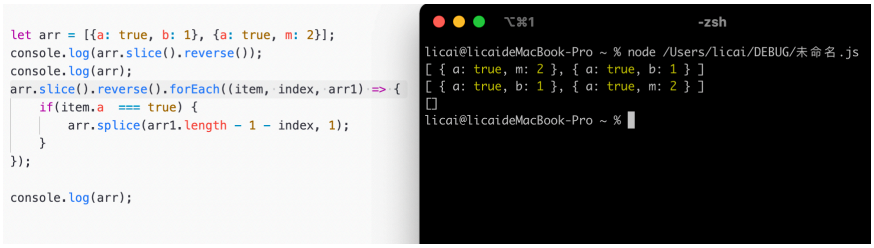

```
const dataSource = [{ a: 1, num: '1' }, { a: 2, num: '2' }];
let newDataSource = JSON.parse(JSON.stringify(dataSource));
newDataSource.map((item, index) => {
  if ((['1', '2']).includes(item.num)) {
    newDataSource.splice(index, 1);
  }
});

console.log(newDataSource);
```

逆向循环删除

```
循环数组，再用 splice 方法删除 ，  但是删除c的时候会发现数组的长度和下标都已经发生改变 ，
所以这个方法要改进一下 。
用逆向循环
for (var i = arr.length - 1; i >= 0; i--) {
    if (判断条件) {
        arr.splice(i, 1);
    }
}
```

foreach方法删除




```
let arr = [{a: true, b: 1}, {a: true, m: 2}];
console.log(arr.slice().reverse());
console.log(arr);
arr.slice().reverse().forEach((item, index, arr1) => {
    if(item.a  === true) {
        arr.splice(arr1.length - 1 - index, 1);
    }
});

console.log(arr);
```

或者是

```
const dataSource = [{ a: 1, num: '1' }, { a: 2, num: '2' }];
const deleteKeys = ['1', '2'];
let newDataSource = JSON.parse(JSON.stringify(dataSource));
```

```
deleteKeys.map((record) => {
  const index = newDataSource.findIndex(item => item.num === record);
  if (index > -1) {
    newDataSource.splice(index, 1);
  }
});

console.log(newDataSource);
```

2、JS数组遍历常用方法

- 传统做法、循环

```
for(let i=0;i<arr.lenth;i++) {
    console.log(arr[i]);
}
```

- forEach()

```
1、forEach中使用return无效
2、forEach等不支持break
3、forEach删除自身元素index不会被重置
```

```
arr.forEach( function(item,index,arr) )
arr.forEach( (item,index,arr) => {
   console.log('当前必填参数',item) ;
   console.log('当前可选参数：元素索引',index) ;
   console.log('当前可选参数：元素所属数组',arr) ;
} );
```

- filter()
- 创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

首先创建了一个空数组，然后筛选callback的返回值，如果返回值可以隐式转换成true,则将对应的元素push到那个空数组中！

```
它内部的回调函数可以传入三个参数(同forEach完全一样)

item为必填参数，表示当前元素

index为可选参数，表示当前元素的索引

arr同样为可选参数，表示当前元素所属的数组对象（正在遍历的这个数组）。

不同于forEach，它是有返回值的，找个变量接收即可：
 const porducts = [
   {name:"cucumber",type:"vegetable"},
   {name:"banana",type:"fruit"},
   {name:"celery",type:"vegetable"},
   {name:"orange",type:"fruit"}
 ];
```

```
const arrNew = arr.filter( (product) => {
   return product.type === "vegetable";
}
);
console.log(arrNew);
//[{name: "cucumber", type: "vegetable"},
    {name: "celery", type: "vegetable"}]
```

- map()
- 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。 它的返回值就是将你正在遍历的那个数组中的回调函数中的return返回值挨个push到它提前创建好的空数组中！

```
var numbers = [1,2,3];
const doubled = numbers.map((number)=>{
   return number * 2;
})
console.log(doubled);//[2,4,6]
```

3、三种 遍历 break

```
for (let el of finnalLineData) {
                console.log(el);
                if (!el.reviewTypeCode || !el.reviewReasonCode) {
                    message.info('行上必填项为空，请完善');
                    break;
                }
}

try {
                finnalLineData.forEach(function(item,index){
                    if (!item.reviewTypeCode || !item.reviewReasonCode) {
                        message.info('行上必填项为空，请完善');
                        throw new Error("checkError");
                    }
                });
            } catch(e) {
                if(e.message !== 'checkError') throw e;
};


// 终止for循环，使用break
for (var i=0; i < 3; i++) {
    if (1==i) {
        break;
    }
    alert(i);//0
}
```