

# 解构

JS的解构很灵活，参考

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\\_operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator)

## 列表解构

```
var parts = ['shoulder', 'knees'];
var lyrics = ['head', ...parts, 'and', 'toes']; // 使用...解构
console.log(lyrics) // [ 'head', 'shoulder', 'knees', 'and', 'toes' ]
```

## 参数解构

```
function f(x, y, z) {
  console.log(x + y + z)
}
var args = [2, 3, 4];
f(...args);
```

## 数组解构

JS的数组解构非常强大。

```
const arr = [100, 200, 300];
let [x, y, z] = arr;
console.log(1, x, y, z);

// 丢弃
const [, b, ] = arr;
console.log(2, b);
// b = 5 // 异常，b声明为const

// 少于数组元素
const [d, e] = arr;
console.log(3, d, e);

// 多于数组元素
const [m, n, o, p] = arr;
console.log(4, m, n, o, p);

// 可变变量
const [f, ...args] = arr;
console.log(5, f);
console.log(5, args);

// 支持默认值
```

```
const [j=1,k,,,l=10] = arr
console.log(j,k,l);
```

解构的时候，变量从左到右和元素对齐，可变参数放到最右边。

能对应到数据就返回数据，对应不到数据的返回默认值，如果没有默认值返回undefined。

## 对象解构

### 简单对象解构

```
const obj = {
  a:100,
  b:200,
  c:300
};

let {x,y,z} = obj;
console.log(x,y,z); // undefined undefined undefined

let {a,b,c} = obj; // key名称
console.log(a,b,c); // 100 200 300

let {a:m, b:n, c} = obj; // 重命名
console.log(m,n,c);

let {a:M, c:N, d:D='python'} = obj; // 缺省值
console.log(M,N,D);
```

解构时，需要提供对象的属性名，会根据属性名找到对应的值。没有找到的返回缺省值，没有缺省值则返回undefined。

## 复杂结构

### 嵌套数组

```
const arr = [1, [2, 3], 4];

const [a, [b, c], d] = arr;
console.log(a, b, c, d); //1 2 3 4

const [e, f] = arr;
console.log(e, f); //1 [ 2, 3 ]

const [g, h, i, j = 18] = arr;
console.log(g, h, i, j); //1 [ 2, 3 ] 4 18

const [k, ...l] = arr;
console.log(k, l); //1 [ [ 2, 3 ], 4 ]
```

## 对象

```
var metadata = {
  title: "Scratchpad",
  translations: [
    {
      locale: "de",
      localization_tags: [ ],
      last_edit: "2014-04-14T08:43:37",
      url: "/de/docs/Tools/Scratchpad",
      title: "JavaScript-Umgebung"
    }
  ],
  url: "/en-US/docs/Tools/Scratchpad"
};
//
var { title: enTitle, translations: [{ title: localeTitle }] } = metadata;

console.log(enTitle); // "Scratchpad"
console.log(localeTitle); // "JavaScript-Umgebung"
```

## 数组的操作

方法	描述
push(...items)	尾部增加多个元素
pop()	移除最后一个元素，并返回它
map	引入处理函数来处理数组中每一个元素，返回新的数组
filter	引入处理函数处理数组中每一个元素，此处理函数返回true的元素保留，否则该元素被过滤掉，保留的元素构成新的数组返回
foreach	迭代所有元素，无返回值

```
const arr = [1, 2, 3, 4, 5];
arr.push(6,7);
console.log(arr);
arr.pop()
console.log(arr);

// map
const powerArr = arr.map(x => x*x); // 新数组
console.log(powerArr);

const newarr = arr.forEach(x => x+10); // 无返回值
console.log(newarr, arr);

console.log(arr.filter(x => x%2==0)) // 新数组
```

## 数组练习

有一个数组 `const arr = [1, 2, 3, 4, 5];`，要求算出所有元素平方值是偶数且大于10的

```
console.log(arr.map(x => x * x).filter(x => x % 2 === 0).filter(x => x > 10));  
// 这种实现好吗？
```

应该先过滤，再求值比较好

```
// 1  
console.log(arr.filter(x => x%2===0).map(x => x*x).filter(x => x > 10)); // 先过滤减少迭代次数  
  
// 2  
s = Math.sqrt(10) // 10开方算一次  
console.log(arr.filter(x => x > s && !(x % 2)).map(x => x*x))  
  
// 3  
let newarr = []  
arr.forEach(x => {  
    if (x>s && !(x%2)) newarr.push(x*x);  
})  
console.log(newarr);
```

## 对象的操作

Object的静态方法	描述
Object.keys(obj)	ES5开始支持。返回所有key
Object.values(obj)	返回所有值，试验阶段，支持较差
Object.entries(obj)	返回所有值，试验阶段，支持较差
Object.assign(target, ...sources)	使用多个source对象，来填充target对象，返回target对象

```
const obj = {  
    a:100,  
    b:200,  
    c:300  
};  
  
console.log(Object.keys(obj)); // key, ES5  
console.log(Object.values(obj)); // 值, 实验性  
console.log(Object.entries(obj)); // 键值对, 实验性  
  
// assign  
var metadata = {  
    title: "Scratchpad",
```

```
translations: [
  {
    locale: "de",
    localization_tags: [ ],
    last_edit: "2014-04-14T08:43:37",
    url: "/de/docs/Tools/Scratchpad",
    title: "JavaScript-Umgebung"
  }
],
url: "/en-US/docs/Tools/Scratchpad"
};

var copy = Object.assign({}, /*目标对象*/, metadata,
  {schoolName: 'magedu', url: 'www.magedu.com'} /*增加新的属性, 覆盖同名属性*/,
  {translations: null} /*覆盖metadata的translations*/
);
console.log(copy);
```

