# 3

# 变量的解构赋值

## 数组的解构赋值

ES6 允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构（Destructuring）。

以前，为变量赋值，只能直接指定值。

```
var a = 1;
var b = 2;
var c = 3;
```

而 ES6 允许写成下面这样。

```
var [a, b, c] = [1, 2, 3];
```

上面的代码表示，可以从数组中提取值，按照位置的对应关系，对变量赋值。

本质上，这种写法属于“模式匹配”，只要等号两边的模式相同，

左边的变量就会被赋予对应的值。下面是一些使用嵌套数组进行解构的例子。

```
var [foo, [[bar], baz]] = [1, [[2], 3]];
foo // 1
bar // 2
baz // 3

var [,,third] = ["foo", "bar", "baz"];
third // "baz"

var [head, ...tail] = [1, 2, 3, 4];
head // 1
tail // [2, 3, 4]
```

如果解构不成功，变量的值就等于 undefined。

```
var [foo] = [];
var [foo] = 1;
var [foo] = 'Hello';
var [foo] = false;
var [foo] = NaN;
```

以上几种情况都属于解构不成功，foo 的值都会等于 undefined。但是，如果对 undefined 或 null 进行解构，就会报错。

```
// 报错
var [foo] = undefined;
var [foo] = null;
```

这是因为解构只能用于数组或对象。其他原始类型的值都可以转为相应的对象，但是，undefined 和 null 不能转为对象，因此报错。

解构赋值允许指定默认值。

```
var [foo = true] = [];
foo // true
```

解构赋值不仅适用于 var 命令，也适用于 let 和 const 命令。

```
var [v1, v2, ..., vN ] = array;
let [v1, v2, ..., vN ] = array;
const [v1, v2, ..., vN ] = array;
```

## 对象的解构赋值

解构不仅可以用于数组，还可以用于对象。

```
var { foo, bar } = { foo: "aaa", bar: "bbb" };
foo // "aaa"
bar // "bbb"
```

对象的解构与数组有一个重要的不同。数组的元素是按次序排列的，变量的取值由它的位置决定；而对象的属性没有次序，变量必须与属性同名，才能取到正确的值。

```
var { bar, foo } = { foo: "aaa", bar: "bbb" };
foo // "aaa"
bar // "bbb"

var { baz } = { foo: "aaa", bar: "bbb" };
baz // undefined
```

上面代码中的第一个例子，等号左边的两个变量的次序，与等号右边两个同名属性的次序不一致，但是对取值完全没有影响。第

二个例子的变量没有对应的同名属性，导致取不到值，最后等于undefined。

如果变量名与属性名不一致，必须写成下面这样。

```
var { foo: baz } = { foo: "aaa", bar: "bbb" };
baz // "aaa"
```

和数组一样，解构也可以用于嵌套结构的对象。

```
var o = {
  p: [
    "Hello",
    { y: "World" }
  ]
};

var { p: [x, { y }] } = o;
x // "Hello"
y // "World"
```

对象的解构也可以指定默认值。

```
var { x = 3 } = {};
x // 3
```

如果要将一个已经声明的变量用于解构赋值，必须非常小心。

```
// 错误的写法

var x;
{x} = {x:1};
// SyntaxError: syntax error
```

上面代码中的写法会报错，因为 JavaScript 引擎会将 {x} 理解成一个代码块，从而发生语法错误。只有不将大括号写在行首，避免 JavaScript 将其解释为代码块，才能解决这个问题。

```
// 正确的写法

({x}) = {x:1};
// 或者
({x} = {x:1});
```

# 用途

变量的解构赋值用途很多。

## 交换变量的值

```
[x, y] = [y, x];
```

## 从函数返回多个值

函数只能返回一个值，如果要返回多个值，只能将它们放在数组或对象里返回。有了解构赋值，取出这些值就非常方便。

```
// 返回一个数组

function example() {
    return [1, 2, 3];
}
var [a, b, c] = example();
```

```
// 返回一个对象

function example() {
  return {
    foo: 1,
    bar: 2
  };
}
var { foo, bar } = example();
```

### 函数参数的定义

```
function f({x, y, z}) {
  // ...
}

f({x:1, y:2, z:3})
```

这种写法对提取 JSON 对象中的数据，尤其有用。

### 函数参数的默认值

```
jQuery.ajax = function (url, {
  async = true,
  beforeSend = function () {},
  cache = true,
  complete = function () {},
  crossDomain = false,
  global = true,
  // ... more config
```

```
}) {
  // ... do stuff
};
```

指定参数的默认值，就避免了在函数体内部再写 `var foo = config.foo || 'default foo';` 这样的语句。

### 遍历 Map 结构

任何部署了 Iterator 接口的对象，都可以用 for...of 循环遍历。Map 结构原生支持 Iterator 接口，配合变量的结构赋值，获取键名和键值就非常方便。

```
var map = new Map();
map.set('first', 'hello');
map.set('second', 'world');

for (let [key, value] of map) {
  console.log(key + " is " + value);
}
// first is hello
// second is world
```

如果只想获取键名，或者只想获取键值，可以写成下面这样。

```
// 获取键名
for (let [key] of map) {
  // ...
}

// 获取键值
```

```
for (let [,value] of map) {
  // ...
}
```

### 输入模块的指定方法

加载模块时，往往需要指定输入哪些方法。解构赋值使得输入语句非常清晰。

```
const { SourceMapConsumer, SourceNode } = require("source-map");
```