## 解构赋值

JavaScript 提供了一类语法以简化赋值操作，可以根据一组数据，一次定义一组变量，像这样：

let [a, b] = [1, 2];  
alert(a); // 1  
alert(b); // 2

等号右边是一个通常的对象/数组字面量，而等号左边就写成与右边相匹配的格式，即一个*模式*，以满足我们的需要。

它实际上是如下方式的替代：

let arr = [1, 2];  
let a = arr[0];  
let b = arr[1];

这类语法被称为*解构赋值*，按字面意义上来理解，就是“解析一个结构，并进行赋值”。

### 数组解构

你可以将一组变量在声明时同时进行赋值操作。

let arr = ["one", "two", "three"];  
let [one, two, three] = arr;  
alert(one); // "one"  
alert(two); // "two"  
alert(three); // "three"

也可以将声明与赋值分离。

let a, b;  
[a, b] = [1, 2];  
alert(a); // 1  
alert(b); // 2

如果一个模式没有找到相匹配的值，相应的变量就会得到 undefined，因为不知道到该给它赋什么值，只能说明它是“未定义”的。

let [a, b, c] = [1, 2];  
alert(a); // 1  
alert(b); // 2  
alert(c); // undefined

为了避免发生这种情况，可以在模式中为变量设置一个默认值。如果找不到匹配值，就使用变量的默认值。

let [a = 3, b = 4, c = 5] = [1, 2];  
alert(a); // 1  
alert(b); // 2  
alert(c); // 5

在一个解构赋值的表达式中我们还可以交换两个已有变量的值，以避免使用*临时变量*。

let a = 3, b = 5;  
[a, b] = [b, a];  
alert(a); // 5  
alert(b); // 3  
  
// 不使用解构赋值  
let temp = a;  
a = b;  
b = temp;  
alert(a); // 5  
alert(b); // 3

你也可以忽略你不感兴趣的值：

let [a, , b] = [1, 2, 3];  
alert(a); // 1  
alert(b); // 3  
  
// 甚至忽略所有值，但是没必要这样  
[, ,] = [1, 2, 3];

当解构一个数组时，可以使用*剩余模式*，将数组的“剩余部分”交给一个变量。

let [a, ...b] = [1, 2, 3];  
alert(a); // 1  
alert(b); // [2, 3]

注意：如果剩余模式右侧有逗号，会得到一个语法错误，因为剩余元素必须是数组的最后一个元素。

let [a, ...b,] = [1, 2, 3]; // SyntaxError: rest element may not have a trailing comma

同时，使用剩余模式赋值的变量不能被设置默认值。

let [a, ...b = 2] = [3, 4] // SyntaxError: Invalid destructuring assignment target  
alert("a = " + a); // 不会执行  
alert("b = " + b); // 不会执行

用于实现剩余模式的三个点号称为*Spread 操作符*。

### 对象解构

除了数组以外，你还可以对一个对象进行解构赋值。在数组解构中，变量被赋予的值取决于它的排列顺序，而在对象解构中，变量根据对象中对应的属性名被赋予值。

let o = {p: 42, q: true};  
let {p, q} = o;  
alert(p); // 42  
alert(q); // true

通过解构，无需声明即可赋值一个变量。

let a, b;  
({a, b} = {a: 1, b: 2});  
alert(a); // 1  
alert(b); // 2

赋值语句周围的 ( ... ) 是使用对象字面解构赋值时不需要声明的语法。{a, b} = {a: 1, b: 2} 不是有效的独立语法，因为左边的 {a, b} 被认为是一个**块语句**而不是对象字面量。然而，({a, b} = {a: 1, b: 2}) 是有效的，相当于let {a, b} = {a: 1, b: 2}

Note：

你的( ... ) 表达式需要一个分号在它前面，否则它也许会被当成上一行中的函数来执行。

我们也可以根据属性，将值赋给我们需要的变量名，而不一定与属性名重名。

let obj = {p: 42, q: true};  
let {p: a, q: b} = obj;   
// 相当于  
// a = obj.p;  
// b = obj.q   
alert(a); // 42   
alert(b); // true

变量也可以先赋予默认值。当要提取的对象没有对应的属性，变量的值就是它的默认值。

let {a = 10, b = 5} = {a: 3};  
alert(a); // 3  
alert(b); // 5

可以将上述方法结合起来。

let {a: first = 10, b: second = 5} = {a: 3};  
alert(first); // 3  
alert(second); // 5

模式中可以像普通对象字面量那样使用计算属性名，来确定要从对象中匹配的属性值。

let key = "name";  
let { [key]: theName } = { name: "Pluto" };  
alert(theName); // "Pluto"

事实上，我们还可以像数组解构那样，对对象解构应用剩余模式：

let {a, b, ...others} = {a: 10, b: 20, c: 30, d: 40}  
alert(a); // 10   
alert(b); // 20  
// others = { c: 30, d: 40 }  
alert(others.c); // 30  
alert(others.d); // 40

显然，如果一个属性不能作为合法的标识符名称，就需要用一个等价的方式来代替。

const obj = { "hello-world": "Oh!" };  
const { "hello-world": helloWorld } = obj;  
  
alert(helloWorld) // "Oh!"