解构赋值

▸ 在本文章中

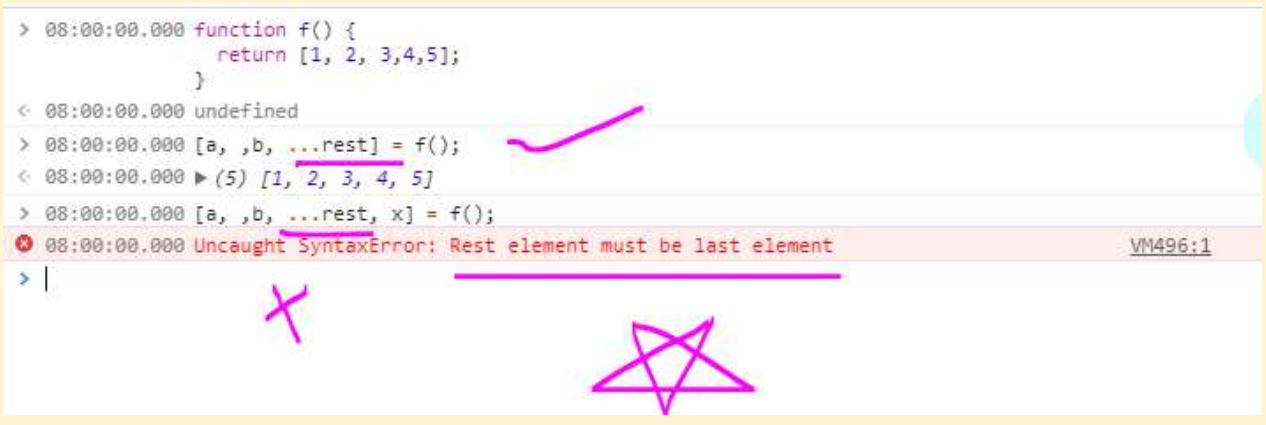
解构赋值 语法是一个Javascript表达式，这使得可以将**值从数组**或**属性从对象**提取到不同的变量中。

语法

```
let a, b, rest;

/* array 解构赋值 */
[a, b] = [1, 2];
console.log(a); // 1
console.log(b); // 2

[a, b, ...rest] = [1, 2, 3, 4, 5];
console.log(a); // 1
console.log(b); // 2
console.log(rest); // [3, 4, 5]

/* object 解构赋值 */
({a, b} = {a:1, b:2});
console.log(a); // 1
console.log(b); // 2

// Stage 3 proposal
/* object & ...rest 解构赋值 */

({a, b, ...rest} = {a:1, b:2, c:3, d:4});
// {a: 1, b: 2, c: 3, d: 4}

rest;
// {c: 3, d: 4}


```

```
28
29  /* es6 module demo */
30  export const stockfasturl = `https://developer.mozilla.org/webservice/fast
31
32  // ES6 auto assignment (destructuring assignment)
33  export const FS = {
34      stockfasturl,
35      // stockfasturl: stockfasturl,
36  };
37
38  export const urls = {
39      FS,
40      //
41  };
42
43  export default urls;
44
45  /* usage */
46
47  import {urls} from `./urls.js`;
48
49  const sf_test = `${urls.FS.stockfasturl}/stockfast${db_id}/${stock_uid}.SH
```

Note: **...rest** 必须是解构赋值参数列表的最后一个参数！



简述

对象字面量和数组字面量提供了一种简单的定义一个特定的数据组的方法。

```
1  let x = [1, 2, 3, 4, 5];
```

解构赋值使用了相同的语法，不同的是在表达式左边定义了要从原变量中取出什么变量。

```
var x = [1, 2, 3, 4, 5];
var [y, z] = x;
console.log(y); // 1
console.log(z); // 2
```

解构赋值的作用类似于Perl和Python语言中的相似特性。

解构数组

变量声明并赋值 时的解构

```
var foo = ["one", "two", "three"];

var [one, two, three] = foo;
console.log(one); // "one"
console.log(two); // "two"
console.log(three); // "three"
```

变量先声明后赋值 时的解构

通过解构分离变量的声明，可以为一个变量赋值。

```
var a, b;

[a, b] = [1, 2];
console.log(a); // 1
console.log(b); // 2
```

默认值

为了防止从数组中取出一个值为`undefined`的对象，可以为这个对象设置默认值。

```
var a, b;

[a=5, b=7] = [1];
console.log(a); // 1
console.log(b); // 7
```

交换变量

在一个解构表达式中可以交换两个变量的值。

没有解构赋值的情况下，交换两个变量需要一个临时变量 (或者用低级语言中的XOR-swap技巧) 。

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

[a, b] = [b, a];
console.log(a); // 3
console.log(b); // 1
```

解析一个从函数返回的数组

从一个函数返回一个数组是十分常见的情况.。解构使得处理返回值为数组时更加方便。

在下面例子中，`[1, 2]` 作为函数的 `f()` 的输出值，可以使用解构用一句话完成解析。

```
function f() {
  return [1, 2];
}

var a, b;
[a, b] = f();
console.log(a); // 1
console.log(b); // 2
```

感谢解构赋值，函数现在可以返回多个值了。尽管函数一直都可以返回一个数组，但现在这样做有更多的灵活性。

忽略某些返回值

你也可以忽略你不感兴趣的返回值：

```
function f() {
  return [1, 2, 3];
}

var [a, , b] = f();
```

```
6 console.log(a); // 1
7 console.log(b); // 3
```

你也可以忽略全部返回值。例如：

```
1 [,,] = f();
```

将剩余数组赋值给一个变量

当解构一个数组时，可以使用剩余模式，将数组剩余部分赋值给一个变量。

```
1 var [a, ...b] = [1, 2, 3];
2 console.log(a); // 1
3 console.log(b); // [2, 3]
```

注意：如果剩余元素右侧有一个逗号，会抛出语法错误的异常，因为剩余元素必须是数组的最后一个元素。

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

用正则表达式匹配提取值

用正则表达式方法exec()匹配字符串会返回一个数组，该数组第一个值是完全匹配正则表达式的字符串，然后的值是匹配正则表达式括号内内容部分。解构赋值允许你轻易地提取出需要的部分，忽略完全匹配的字符串—如果不需要的话。

```
1 var url = "https://developer.mozilla.org/en-US/Web/JavaScript";
2
3 var parsedURL = /^(\w+)\:\/\/([^\/]+)\/(.*)$/.exec(url);
4 console.log(parsedURL); // ["https://developer.mozilla.org/en-US/Web/JavaS
5
6 var [, protocol, fullhost, fullpath] = parsedURL;
7
8 console.log(protocol); // "https"
```

解构对象

简单示例

```
var o = {p: 42, q: true};
var {p, q} = o;

console.log(p); // 42
console.log(q); // true

// 用新变量名赋值
var {p: foo, q: bar} = o;

console.log(foo); // 42
console.log(bar); // true
```

无声明赋值

通过解构可以无需声明来赋值一个变量。

```
var a, b;

({a, b} = {a: 1, b: 2});
```

> 当通过解构一个对象字面量来无需声明赋值变量时必须给表达式加上"(..)"，{a, b} = {a: 1, b: 2}不是有效的语法，因为{a, b}被认为是一个代码块而不是对象字面量，但是({a, b} = {a: 1, b: 2})是有效的，相当于var {a, b} = {a: 1, b: 2}

给新的变量名赋值

可以从一个对象中提取变量并赋值给和对象属性名不同的新的变量名。

```
var o = {p: 42, q: true};
var {p: foo, q: bar} = o;

console.log(foo); // 42
console.log(bar); // true
```

默认值

变量可以先赋予默认值。当要提取的对象没有对应的属性，变量就被赋予默认值。

```
var {a = 10, b = 5} = {a: 3};

console.log(a); // 3
console.log(b); // 5
```

函数参数默认值

ES5版本

```
function drawES5Chart(options) {
  options = options === undefined ? {} : options;
  var size = options.size === undefined ? 'big' : options.size;
  var cords = options.cords === undefined ? { x: 0, y: 0 } : options.cords
  var radius = options.radius === undefined ? 25 : options.radius;
  console.log(size, cords, radius);
  // now finally do some chart drawing
}

drawES5Chart({
  cords: { x: 18, y: 30 },
  radius: 30
});
```

ES2015版本

```
function drawES2015Chart({size = 'big', cords = { x: 0, y: 0 }, radius = 2
{
  console.log(size, cords, radius);
  // do some chart drawing
}

drawES2015Chart({
  cords: { x: 18, y: 30 },
  radius: 30
});
```

Firefox中，**解构赋值默认值**还未被实施: var { x = 3 } = {}和var [foo = "bar"] = []。函数中的解构默认值请参考bug 932080。

加载模块

解构赋值可以帮助加载一个模块的特定子集，像Add-on SDK中:

```
const { Loader, main } = require('toolkit/loader');
```

解构嵌套对象和数组

```
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: englishTitle, translations: [{ title: localeTitle }] } = meta

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

For of 迭代和解构

```
var people = [
  {
    name: "Mike Smith",
    family: {
      mother: "Jane Smith",
      father: "Harry Smith",
      sister: "Samantha Smith"
    },
    age: 35
  },
  {
    name: "Tom Jones"
```

```
    name: "Tom Jones",
    family: {
      mother: "Norah Jones",
      father: "Richard Jones",
      brother: "Howard Jones"
    },
    age: 25
  }
];

for (var {name: n, family: { father: f } } of people) {
  console.log("Name: " + n + ", Father: " + f);
}

// "Name: Mike Smith, Father: Harry Smith"
// "Name: Tom Jones, Father: Richard Jones"
```

从作为函数实参的对象中提取数据

```
function userId({id}) {
  return id;
}

function whois({displayName: displayName, fullName: {firstName: name}}){
  console.log(displayName + " is " + name);
}

var user = {
  id: 42,
  displayName: "jdoe",

  fullName: {
      firstName: "John",
      lastName: "Doe"
  }
};

console.log("userId: " + userId(user)); // "userId: 42"
whois(user); // "jdoe is John"
```

这段代码从user对象中提取并输出`id`、`displayName`和`firstName`。

对象属性计算名和解构

计算属性名，如object literals，可以被解构。

```
let key = "z";
let { [key]: foo } = { z: "bar" };

console.log(foo); // "bar"
```

规范

Specification	Status	Comment
ECMAScript 2015 (6th Edition, ECMA-262) Destructuring assignment	ST Standard	Initial definition.
ECMAScript Latest Draft (ECMA-262) Destructuring assignment	LS Living Standard	

浏览器兼容性

Desktop | Mobile

Feature	Chrome	Firefox (Gecko)	Edge	Internet Explorer	Opera	Safari
Basic support	49.0	2.0 (1.8.1)	14	未实现	未实现	7.1
Computed property names	49.0	34 (34)	14	未实现	未实现	未实现
Spread operator	49.0	34 (34)	12[1]	?	?	?

[1] 需要在`about:flags`下开启 "Enable experimental Javascript features"

Firefox特别注意事项

- Firefox为JS1.7提供了解构的非标准的语言扩展。这个扩展在Gecko 40 (Firefox 40 / Thunderbird 40 / SeaMonkey 2.37)中被移除。参见bug 1083498。
- 自 (Firefox 41 / Thunderbird 41 / SeaMonkey 2.38)版本开始，出于遵守ES2015规范的考虑, 圆括号内的解构赋值,比如 `([a, b]) = [1, 2]` 或者 `({a, b}) = { a: 1, b: 2 }`,现在被视为 invalid 并且会抛出 `SyntaxError`. 点击 Jeff Walden's blog post 和 bug 1146136 来获得更多信息。

相关链接

- 赋值操作符
- ES6 in Depth: Destructuring" on hacks.mozilla.org
- Object Rest/Spread Properties for ECMAScript