解构赋值

> 导读：解构？赋值？带你了解 ES6/7/8/9+ 的新特性。

本文包含了如下知识点：

- 数组解构
  - 顺序解构
  - 剩余参数解构
- 对象解构
  - 键名解构
  - 别名解构
  - 层级解构
  - 剩余属性解构
  - 默认赋值
- 基本类型解构
  - 数字类型解构
  - 字符串类型解构
  - `undefined | null` 特殊类型解构
- 实际应用
  - `for ... of` 循环解构
  - 函数参数解构
  - 交换变量
  - ES6 的 `import` 语句
- 注意事项
  - 已定义变量的解构赋值

定义

解构赋值（Destructuring Assignment）是 ES6 中新提出的一种表达式，通过解构赋值，可以将属性或值从对象/数组中取出，直接赋予变量。

数组解构

顺序解构

> 按照数组顺序一一赋值。

看一个最简单的例子：

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

const [a, b] = array;
a === array[0]; // a === 1
b === array[1]; // b === 2
```

这是一个最基本的数据解构赋值，即按照对应的顺序将数组中的值分配给对应的变量。

小技巧：解构字符串

字符串也可以被解构，如：

```
const text = 'Hello, world!';

const [h, e, l, ...rest] = text;
h === 'H';
e === 'e';
l === 'l';
rest => [ 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!' ]
```

小技巧：跳跃解构

可以使用空变量来跳过某些不需要的值：

```
const array = ['not this', 'get this'];

const [, text] = array; // 不需要第一个变量
```

剩余参数解构

> 使用**剩余参数语法**获取数组的剩余元素。

看一个例子：

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

const [a, ...rest] = array;
a === array[0]; // a === 1
rest => [2, 3, 4];
```

默认赋值

> 数组解构支持设置默认值。

```
const array = [1, 2];

const [a, b, c = 100] = array;
a === array[0]; // a === 1
b === array[1]; // b === 2
c === 100;
```

对象解构

键名解构

> 按照对应的键名来解构对象。

看一个例子：

```
const object = { name: 'Ruby', alias: 'Oooops', age: 18 };

const { name, alias } = object;
name === object.name; // name === 'Ruby'
alias = object.alias; // alias === 'Oooops'
```

和数组的顺序解构大同小异，声明对应的键名即可。

小技巧：解构获取方法

```
const { log, sin, cos } = Math;

const { log, info } = console;
```

小技巧：数组的对象化解构

如快捷的获取首个或最后一个字符：

```
const array = ['first', 'some', 'other', 'last'];

const { 0: first, [array.length - 1]: last } = array;
first === 'first';
last === 'last';
```

别名解构

> 使用别名解构，可以为变量声明自己的名称，即 `{ 属性: 变量名 }`。

看一个例子：

```
const object = { name: 'Ruby', alias: 'Oooops', age: 18 };

const { name: who } = object;
who === object.name; // who === 'Ruby'
```

层级解构

> 解构对象中的对象或数组。

看一个例子：

```
const article = {
    author: {
        name: 'Lily',
        age: 18
    }
};

const { author: { name, age } } = article;

name === article.author.name; // name === 'Lily'
age === article.author.age; // age === 18
```

小技巧：对象与数组的层级解构

快速获取第一条评论：

```
const article = {
    comments: [
        { user: 'Allen', content: 'Hello, world!' },
        { user: 'anonymous', content: `I don't like it!` }
    ]
};

const { comments: [ first, ...rest ]} = article;
first => { user: 'Allen', content: 'Hello, world!' };
rest => [ { user: 'anonymous', content: "I don't like it!" } ];
```

剩余属性解构

> 使用**剩余参数语法**获取对象的剩余属性，并组合成一个新的对象。

看一个例子：

```
const object = { name: 'Ruby', alias: 'Oooops', age: 18 };

const { name, ...rest } = object;
name === object.name; // name === 'Ruby'
rest => { alias: 'Oooops', age: 18 };
```

剩余参数会获取到所有未赋值的属性，并生成一个新的对象。

默认赋值

> 对象的解构赋值也支持设置默认值。

如下所示：

```
const object = { };

const { age = 10 } = object;
age === 10;
```

小提示：对应属性必须为 undefined

默认值只有在对应属性为 `undefined` 时才会生效，`null | 0 | ''` 等均不会生效，如：

```
const object = { name: '', alias: null, age: 0 };

const {
    name = 'anonymous',
    alias = 'none',
    age = 10,
    sex = 'unknown'
} = object;

name === '';
alias === null;
age === 0;
sex === 'unknown'; // 只有 undefined 才会使默认值生效
```

小技巧：别名默认值的奇妙写法

如果使用了别名解构，还要设置默认值，那应该怎么写呢？

```
const object = {};

const { alias: name = 'anonymous' } = object;

name === 'anonymous';
```

总觉得怪怪的，要是 TypeScript 该怎么写？

还真没法写。

基本类型解构

数字类型解构

> 解构前解释器会自动将其转换为对象。

看一段代码：

```
const num = 123.456;

const { toFixed } = num;
`${toFixed.call(num, 1)}` === '123.5'; // 注意，数值不相等，只是显示相同
```

字符串类型解构

> 同理，解释器也会自动将其转化为对象。

```
const text = 'text';

const { toString } = text;
toString.call(text) === 'text';
```

undefined | null 特殊类型解构

因为 `undefined` 以及 `null` 无法被转换为对象，因此无法解构。

实际应用

for of 循环解构

> 在 `for ... of` 循环中使用解构，快速获取对象属性：

```
const array = [{ name: 'Allen', age: 18 }];

for (const {name, age} of array) {
    name === 'Allen';
    age === 18;
}
```

函数参数解构

> 解构参数以快速获取内容。

解构数组

```
function add([x, y]) {
    return x + y;
}

add([1, 2]) === 3;
```

解构对象

```
function display({ name, age }) {
    return `${name} is ${age} years old.`;
}

const person = { name: 'Faker', age: 24 };

display(person) === 'Faker is 24 years old.';
```

参数默认值

```
function add([x, y] = [1, 2]) {
    return x + y;
}

add() === 3;

function sub({ minus, minute } = { minus: 10, minute: 5 }) {
    return minus - minute;
}

sub() === 5;
```

交换变量

> 剩去中间人，快速交换变量。

交换 a 和 b 的值：

```
let a = 10;
let b = 20;

[a, b] = [b, a];
a === 20;
b === 10;
```

ES6 的 import 语句

想不到吧，import 也是解构赋值🤓！

注意事项

已定义变量的解构赋值

如果要对一个已经定义的变量使用解构赋值，那么使用类似下面的语法：

```
const object = { age: 18 };

let age;

({ age } = object);
age === 18;
```

理由很简单，`{` 大括号在行首时，解释器会认为这是一个块级代码，而非赋值语句。