3、变量的解构赋值

- 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）
  1. 数组的解构赋值
  2. 对象的解构赋值
  3. 字符串的解构赋值
  4. 数值和布尔值的解构赋值
  5. 函数参数的解构赋值
  6. 圆括号问题
  7. 用途

① 数组的解构赋值

- 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。

```
//声明多个变量并赋值
let a = 10;
let b = 20;
let c = 30;

//ES6 允许写成下面这样
let [x,y,z]=[10,20,30];
console.log(x); //10
console.log(y); //20
console.log(z); //30
//从数组中提取值，按照对应位置，对变量赋值

let [x1,[[y1],z1]]=[10,[[20],30]]; //嵌套数组
console.log(x1); //10
console.log(y1); //20
console.log(z1); //30
```

- 其它数组的解构赋值

```
//解构赋部分数值
let [m, n] = [10,20,30];
console.log(m); //10
console.log(n); //20

let [i,[j],k] = [10,[20,30],40];
console.log(i); //10
console.log(j); //20
console.log(k); //40
```

```
let [x, , y] = [1, 2, 3];
console.log(x); //1
console.log(y); //3

let [ , , z] = ["one", "two", "three"];
console.log(z); //three

//其中...为ES6的扩展运算符，即d加上...可以接收多个值
let [a, ...d] = [1, 2, 3, 4];
console.log(a); // 1
console.log(d); // [2, 3, 4]

let [x1, y1, ...z1] = ['a'];
console.log(x1); // "a"
console.log(y1); // undefined
console.log(z1); // []

//没有接收到值得变量，默认为undefined
let [z2] = [];  //z2:undefined
let [x2, y2] = [1]; //x2:1,y2:undefined
```

- 如果等号的右边不是数组（或者严格地说，不是可遍历的结构(Iterator)），那么将会报错。

```
// 以下都会报类型错误
let [foo] = 1;     //TypeError: 1 is not iterable 不是可迭代的
let [foo] = false; //TypeError: false is not iterable
let [foo] = NaN;
let [foo] = undefined;
let [foo] = null;
let [foo] = {};
```

- 默认值:

```
//当一个数组成员严格等于undefined，默认值才会生效。
let [b = true] = [];
console.log(b); //true

let [x, y = 'b'] = ['a']; // x='a', y='b'
let [x1, y1 = 'b'] = ['a', undefined]; // x1='a', y1='b'
console.log(x); //a
console.log(y); //b

let [m = 1] = [undefined];
console.log(m) // 1

let [n = 1] = [null]; //值不是undefined，所以没有使用默认值
console.log(n) // null
```

② 对象的解构赋值

- 解构不仅可以用于数组，还可以用于对象。

```
//对象的解构赋值与数组不同，要求变量必须与属性同名，才能取到正确的值。
//let {name, age} = {name:'张三', age:20};
let {age, name} = {name:'张三', age:20};
let {sex} = {name:'张三', age:20};//解构失败，变量的值等于undefined
console.log(name); //张三
console.log(age);  //20
console.log(sex);  //undefined

//如果变量名与属性名不一致，必须写成下面这样
let {email:em, password:ps} = {email:'zs@163.com', password:'123456'};
//如上代码email和password都是匹配的模式，em才是变量。真正被赋值的是变量em，而不是模式email
console.log(em);    //zs@163.com
console.log(ps);    //123456
console.log(email); //错误 ReferenceError: email is not defined
```

- 与数组一样，解构也可以用于嵌套结构的对象。

```
//定义一个书的信息
var obj = {
    book: [
        'JavaScript权威指南',
        {author:'小淘',price:132}
    ]
};

//let {book:[title, {author, price}]} = obj;
//此时book是模式，不是变量，因此不会被赋值。如果book也要作为变量赋值，可写成如下：
let {book,book:[title, {author, price}]} = obj;
console.log(title); //JavaScript权威指南
console.log(author);//小淘
console.log(price); //132
console.log(book);  //['JavaScript权威指南',{author:'小淘',price:132}]
```

- 对象的解构也可以指定默认值

```
var {x=3} = {};
console.log(x); // 3

var {x1, y1=5} = {x1:1};
console.log(x1); // 1
console.log(y1); // 5

var {x2: y2=3} = {};
console.log(y2); // 3

var {x3: y3=3} = {x3: 5};
console.log(y3); // 5
```

③ 字符串的解构赋值(了解)

- 字符串也可以解构赋值。这是因为此时，字符串被转换成了一个类似数组的对象。

```
const [a, b, c, d, e] = 'hello';
console.log(a); // "h"
console.log(b); // "e"
console.log(c); // "l"
console.log(d); // "l"
console.log(e); // "o"
```

类似数组的对象都有一个length属性，因此还可以对这个属性解构赋值。

```
let {length : len} = 'hello';
console.log(len); // 5
```

④ 数值和布尔值的解构赋值(了解)

- 解构赋值时，如果等号右边是数值和布尔值，则会先转为对象。

```
//解构赋值的规则是，只要等号右边的值不是对象或数组，就先将其转为对象。
let {toString: s1} = 123;
//数值和布尔值的包装对象都有toString属性
console.log(s1 === Number.prototype.toString); // true

let {toString: s2} = true;
console.log(s2 === Boolean.prototype.toString); // true

//由于undefined和null无法转为对象，所以对它们进行解构赋值，都会报错。
let { prop: x3 } = undefined; // TypeError
let { prop: y3 } = null; // TypeError
```

⑤ 函数参数的解构赋值

```
//函数的参数也可以使用解构赋值。
function move({x=0, y=0} = {}) {
    return [x, y];
}

console.log(move({x:3, y:8})); // [3, 8]
console.log(move({x:3}));      // [3, 0]
console.log(move({}));         // [0, 0]
console.log(move());           // [0, 0]
```

⑥ 圆括号问题（了解）

- 变量声明语句，模式不能使用圆括号
- 函数参数也属于变量声明，因此不能带有圆括号

```
// 变量声明语句，模式不能使用圆括号，以下6行全部报错
// let [(a)] = [1];
// let {x: (c)} = {};
// let ({x: c}) = {};
// let {(x: c)} = {};
// let {(x): c} = {};
// let { o: ({ p: p }) } = { o: { p: 2 } };

//函数参数也属于变量声明，因此也不能带有圆括号
//function f([(z)]) { return z; }
//function f([z,(x)]) { return x; }

//将整个模式放在圆括号之中，导致报错
//({ p: a }) = { p: 42 };
//([a]) = [5];

//将一部分模式放在圆括号之中，导致报错
// let[({ p: a }), { x: c }] = [{}, {}];

//可以使用圆括号的情况只有一种：赋值语句的非模式部分，可以使用圆括号。
let b,d;
[(b)] = [3]; // 正确 模式是取数组的第一个成员跟圆括号无关
({p:(d)} = {p:20}); // 正确 模式是p，而不是d
[(parseInt.prop)] = [3]; // 正确 与第一行语句的性质一致。
// 首先它们都是赋值语句，而不是声明语句；其次它们的圆括号都不属于模式的一部分
```

7️⃣. 用途

（1）交换变量的值

```
let x = 1;
let y = 2;
[x, y] = [y, x];`
```

(2) 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 example1() {
  return [1, 2, 3];
}
let [a, b, c] = example1();

// 返回一个对象
function example2() {
  return {
    foo: 1,
    bar: 2
  };
}
let { foo, bar } = example2();
```

(3) 函数参数的定义

```
// 解构赋值可以方便地将一组参数与变量名对应起来
// 参数是一组有次序的值
function f([x, y, z]) { ... }
f([1, 2, 3]);

// 参数是一组无次序的值
function f({x, y, z}) { ... }
f({z: 3, y: 2, x: 1});
```

(4) 提取 JSON 数据

```
// 解构赋值对提取 JSON 对象中的数据，尤其有用。
let jsonData = {
  id: 42,
  status: "OK",
  data: [867, 5309]
};

let { id, status, data: number } = jsonData;

console.log(id, status, number);
// 42, "OK", [867, 5309]
```