**拷贝数组**

得到一份数组的拷贝是很常见的任务，过去我们使用Array.prototype.slice来做，但现在我们可以使用展开运算符：

var arr = [1,2,3];

var arr2 = [...arr]; // 和arr.slice()差不多

arr2.push(4)

**合并数组**

arr1.push(...arr2) // 将arr2 追加到数组的末尾

arr1.unshift(...arr2) // 将arr2 追加到数组的开头

如果你想要整合两个数组，并且想把某个数组放在另一个数组的任意特定位置上，你可以这么做：

var arr1 = ['two', 'three'];

var arr2 = ['one', ...arr1, 'four', 'five'];

// ["one", "two", "three", "four", "five"]

这是一种比其他方式更短的语句！

**将arguments或者NodeList转换为Array**

像拷贝数组一样，我们常常使用Array.Prototype.slice来将NodeList和arguments这种类数组对象转换为真正的数组。但是现在我们能够用展开运算符轻易的实现这项任务：

[...document.querySelectorAll('div')]

当然了，展开运算符将数组“展开”成为不同的参数，所以任何可接收任意数量的参数的函数，都能够使用展开运算符来传参。

let numbers = [9, 4, 7, 1];

Math.min(...numbers); // 1

**结构，按顺序或键值对结构。**

let { x, y, ...z } = { x: 1, y: 2, a: 3, b: 4 };

console.log(x); // 1

console.log(y); // 2

console.log(z); // { a: 3, b: 4 }

扩展运算符的解构赋值，不能复制继承自原型对象的属性。

let o1 = { a: 1 };

let o2 = { b: 2 };

o2.\_\_proto\_\_ = o1;

let { ...o3 } = o2;

o3 // { b: 2 }

o3.a // undefined

对象o3复制了o2，但是只复制了o2自身的属性，没有复制它的原型对象o1的属性。

const o = Object.create({ x: 1, y: 2 });

o.z = 3;

let { x, ...{ y, z } } = o;

x // 1

y // undefined

z // 3

o在原型上继承{ x: 1, y: 2 }，本身无x，y。

上面代码中，变量x是单纯的解构赋值，所以可以读取对象o继承的属性；变量y和z是扩展运算符的解构赋值，只能读取对象o自身的属性，所以变量z可以赋值成功，变量y取不到值。