**对象的扩展**

对象的简洁表示法

let a = 1;

let obj = {

a: a

}

console.log(obj); // Object { a: 1 }

这里的属性a，正好也对应着一个变量a的值，所以这个时候就可以采用简洁表示法。

let a = 1;

let obj = {

a

}

console.log(obj); // Object { a: 1 }

对象的简洁表示法方式二，函数

let obj = {

fn1: function() {

console.log(1);

},

fn2() {

console.log(2);

}

}

obj.fn1(); // 1

obj.fn2(); // 2

Object.is(); 方法判断两个值是否是[相同的值](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Equality_comparisons_and_sameness)，一样就返回true

这与[===](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Identity" \o "JavaScript 有两种比较方式：严格比较运算符和转换类型比较运算符。对于严格比较运算符（===）来说，仅当两个操作数的类型相同且值相等为 true，而对于被广泛使用的比较运算符（==）来说，会在进行比较之前，将两个操作数转换成相同的类型。对于关系运算符（比如 <=）来说，会先将操作数转为原始值，使它们类型相同，再进行比较运算。)运算符也不一样。[===](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Identity" \o "JavaScript 有两种比较方式：严格比较运算符和转换类型比较运算符。对于严格比较运算符（===）来说，仅当两个操作数的类型相同且值相等为 true，而对于被广泛使用的比较运算符（==）来说，会在进行比较之前，将两个操作数转换成相同的类型。对于关系运算符（比如 <=）来说，会先将操作数转为原始值，使它们类型相同，再进行比较运算。)运算符（和[==](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Equality" \o "JavaScript 有两种比较方式：严格比较运算符和转换类型比较运算符。对于严格比较运算符（===）来说，仅当两个操作数的类型相同且值相等为 true，而对于被广泛使用的比较运算符（==）来说，会在进行比较之前，将两个操作数转换成相同的类型。对于关系运算符（比如 <=）来说，会先将操作数转为原始值，使它们类型相同，再进行比较运算。)运算符）将数字值-0和+0视为相等，并认为[NaN](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Number/NaN" \o "Number.NaN 表示\“非数字\”（Not-A-Number）。和 NaN 相同。)不等于[NaN](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/NaN" \o "全局属性 NaN 的值表示不是一个数字（Not-A-Number）。)。

console.log(Object.is(NaN, NaN)); // true

console.log(Object.is(+0, -0)); // false

Object.keys(对象); 返回对象可枚举的属性名，返回一个数组

let obj = {a: 1, b: 2, c: 3};

for (let key of Object.keys(obj)) {

console.log(key);

};

Object.values(对象); 键值，返回一个数组

let obj = {a: 1, b: 2, c: 3};

for (let val of Object.values(obj)) {

console.log(val);

};

Object.entries(对象); 键值对，返回一个二维数组，可以用解构赋值取得里面的key和value

通过这种形式，可以让for-of来遍历对象（对象本来不能用of循环）

let obj = {a: 1, b: 2, c: 3};

for (let [key, val] of Object.entries(obj)) {

console.log(key, val);

};

Object.assign(); 浅复制

用于对象的合并，将源对象的所有可枚举属性，复制到目标对象。返回合并以后的对象。

格式：Object.assign(obj1, obj2, obj3); obj2和obj3都复制到obj1上面。

let obj1 = {a: 1};

let obj2 = {a: 2, b: 3};

let obj3 = {c: 'abc'};

Object.assign(obj1, obj2, obj3);

console.log(obj1); // Object { a: 2, b: 3, c: "abc" }

let a = { a: 1, b: 2 };

let b = { c: 3, d: 4 };

let obj = Object.assign({}, a, b);

console.log(obj); // {a: 1, b: 2, c: 3, d: 4}

也可以用…（扩展运算符）

let obj1 = {

a: 1

}

let obj2 = {

b: 2

}

let obj3 = {

...obj1,

...obj2,

c: 3

}

console.log(obj3); // Object { a: 1, b: 2, c: 3 }