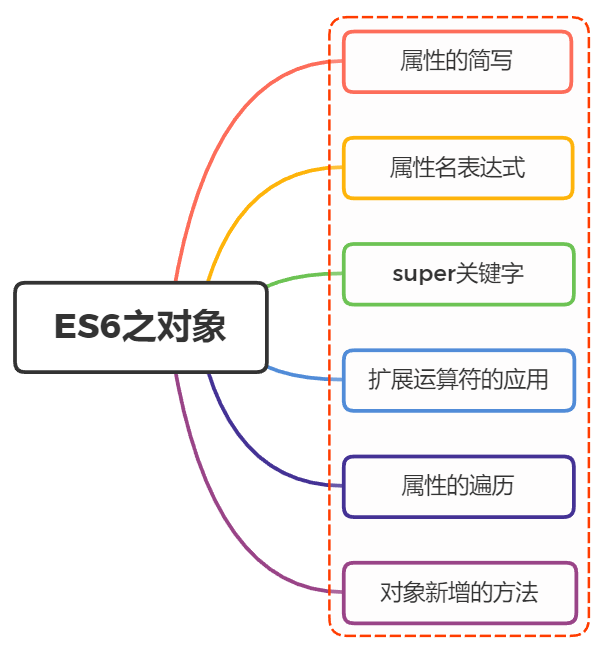
# 面试官：对象新增了哪些扩展？



## 一、属性的简写

ES6中，当对象键名与对应值名相等的时候，可以进行简写

const baz = {foo:foo}  
  
// 等同于  
const baz = {foo}

方法也能够进行简写

const o = {  
 method() {  
 return "Hello!";  
 }  
};  
  
// 等同于  
  
const o = {  
 method: function() {  
 return "Hello!";  
 }  
}

在函数内作为返回值，也会变得方便很多

function getPoint() {  
 const x = 1;  
 const y = 10;  
 return {x, y};  
}  
  
getPoint()  
// {x:1, y:10}

注意：简写的对象方法不能用作构造函数，否则会报错

const obj = {  
 f() {  
 this.foo = 'bar';  
 }  
};  
  
new obj.f() // 报错

## 二、属性名表达式

ES6 允许字面量定义对象时，将表达式放在括号内

let lastWord = 'last word';  
  
const a = {  
 'first word': 'hello',  
 [lastWord]: 'world'  
};  
  
a['first word'] // "hello"  
a[lastWord] // "world"  
a['last word'] // "world"

表达式还可以用于定义方法名

let obj = {  
 ['h' + 'ello']() {  
 return 'hi';  
 }  
};  
  
obj.hello() // hi

注意，属性名表达式与简洁表示法，不能同时使用，会报错

// 报错  
const foo = 'bar';  
const bar = 'abc';  
const baz = { [foo] };  
  
// 正确  
const foo = 'bar';  
const baz = { [foo]: 'abc'};

注意，属性名表达式如果是一个对象，默认情况下会自动将对象转为字符串[object Object]

const keyA = {a: 1};  
const keyB = {b: 2};  
  
const myObject = {  
 [keyA]: 'valueA',  
 [keyB]: 'valueB'  
};  
  
myObject // Object {[object Object]: "valueB"}

## 三、super关键字

this关键字总是指向函数所在的当前对象，ES6 又新增了另一个类似的关键字super，指向当前对象的原型对象

const proto = {  
 foo: 'hello'  
};  
  
const obj = {  
 foo: 'world',  
 find() {  
 return super.foo;  
 }  
};  
  
Object.setPrototypeOf(obj, proto); // 为obj设置原型对象  
obj.find() // "hello"

## 四、扩展运算符的应用

在解构赋值中，未被读取的可遍历的属性，分配到指定的对象上面

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

注意：解构赋值必须是最后一个参数，否则会报错

解构赋值是浅拷贝

let obj = { a: { b: 1 } };  
let { ...x } = obj;  
obj.a.b = 2; // 修改obj里面a属性中键值  
x.a.b // 2，影响到了结构出来x的值

对象的扩展运算符等同于使用Object.assign()方法

## 五、属性的遍历

ES6 一共有 5 种方法可以遍历对象的属性。

* for...in：循环遍历对象自身的和继承的可枚举属性（不含 Symbol 属性）
* Object.keys(obj)：返回一个数组，包括对象自身的（不含继承的）所有可枚举属性（不含 Symbol 属性）的键名
* Object.getOwnPropertyNames(obj)：回一个数组，包含对象自身的所有属性（不含 Symbol 属性，但是包括不可枚举属性）的键名
* Object.getOwnPropertySymbols(obj)：返回一个数组，包含对象自身的所有 Symbol 属性的键名
* Reflect.ownKeys(obj)：返回一个数组，包含对象自身的（不含继承的）所有键名，不管键名是 Symbol 或字符串，也不管是否可枚举

上述遍历，都遵守同样的属性遍历的次序规则：

* 首先遍历所有数值键，按照数值升序排列
* 其次遍历所有字符串键，按照加入时间升序排列
* 最后遍历所有 Symbol 键，按照加入时间升序排

Reflect.ownKeys({ [Symbol()]:0, b:0, 10:0, 2:0, a:0 })  
// ['2', '10', 'b', 'a', Symbol()]

## 六、对象新增的方法

关于对象新增的方法，分别有以下：

* Object.is()
* Object.assign()
* Object.getOwnPropertyDescriptors()
* Object.setPrototypeOf()，Object.getPrototypeOf()
* Object.keys()，Object.values()，Object.entries()
* Object.fromEntries()

### Object.is()

严格判断两个值是否相等，与严格比较运算符（===）的行为基本一致，不同之处只有两个：一是+0不等于-0，二是NaN等于自身

+0 === -0 //true  
NaN === NaN // false  
  
Object.is(+0, -0) // false  
Object.is(NaN, NaN) // true

### Object.assign()

Object.assign()方法用于对象的合并，将源对象source的所有可枚举属性，复制到目标对象target

Object.assign()方法的第一个参数是目标对象，后面的参数都是源对象

const target = { a: 1, b: 1 };  
  
const source1 = { b: 2, c: 2 };  
const source2 = { c: 3 };  
  
Object.assign(target, source1, source2);  
target // {a:1, b:2, c:3}

注意：Object.assign()方法是浅拷贝，遇到同名属性会进行替换

### Object.getOwnPropertyDescriptors()

返回指定对象所有自身属性（非继承属性）的描述对象

const obj = {  
 foo: 123,  
 get bar() { return 'abc' }  
};  
  
Object.getOwnPropertyDescriptors(obj)  
// { foo:  
// { value: 123,  
// writable: true,  
// enumerable: true,  
// configurable: true },  
// bar:  
// { get: [Function: get bar],  
// set: undefined,  
// enumerable: true,  
// configurable: true } }

### Object.setPrototypeOf()

Object.setPrototypeOf方法用来设置一个对象的原型对象

Object.setPrototypeOf(object, prototype)  
  
// 用法  
const o = Object.setPrototypeOf({}, null);

### Object.getPrototypeOf()

用于读取一个对象的原型对象

Object.getPrototypeOf(obj);

### Object.keys()

返回自身的（不含继承的）所有可遍历（enumerable）属性的键名的数组

var obj = { foo: 'bar', baz: 42 };  
Object.keys(obj)  
// ["foo", "baz"]

### Object.values()

返回自身的（不含继承的）所有可遍历（enumerable）属性的键对应值的数组

const obj = { foo: 'bar', baz: 42 };  
Object.values(obj)  
// ["bar", 42]

### Object.entries()

返回一个对象自身的（不含继承的）所有可遍历（enumerable）属性的键值对的数组

const obj = { foo: 'bar', baz: 42 };  
Object.entries(obj)  
// [ ["foo", "bar"], ["baz", 42] ]

### Object.fromEntries()

用于将一个键值对数组转为对象

Object.fromEntries([  
 ['foo', 'bar'],  
 ['baz', 42]  
])  
// { foo: "bar", baz: 42 }

## 参考文献

* https://es6.ruanyifeng.com/#docs/object