

Object.keys()、Object.values()、Object.entries()的用法

链接: <https://juejin.cn/post/6953523365348900895#heading-11>

Object.keys()

1. 处理对象, 返回可枚举的属性数组

```
let person = {name: "张三", age: 25, address: "深圳", getName: function() {}};
console.log(Object.keys(person));
// ["name", "age", "address", "getName"]
```

2. 处理数组, 返回索引值数组

```
let arr = [1, 2, 3, 4, 5, 6];
console.log(Object.keys(arr));
// ["0", "1", "2", "3", "4", "5"]
```

3. 处理字符串, 返回索引值数组

```
let str = "ikun你好";
console.log(Object.keys(str));
// ["0", "1", "2", "3", "4", "5"]
```

4. 常用技巧

```
let person = {name: "张三", age: 25, address: "深圳", getName: function() {}};
Object.keys(person).map((key) => {
  console.log(person[key]); // 获取到属性对应的值, 做一些处理
});
// 张三 25 深圳 f() {}
```

Object.values()

1. 返回一个数组, 成员是参数对象自身的 (不含继承的) 所有可遍历属性的键值

```
let obj = {
  foo: "bar",
  baz: 20
};
console.log(Object.values(obj));
// ["bar", 20]
```

2. 返回数组的成员顺序, 与属性的遍历部分介绍的排列规则一致

```
const obj = {100 : "a", 2 : "b", 7 : "c"};
console.log(Object.values(obj));
//["b", "c", "a"]
```

属性名为数值的属性，是按照数值大小，从小到大遍历的，因此返回的顺序是b、c、a。

3.Object.values()只会遍历对象自身的可遍历属性

```
const obj = Object.create({}, {p : {value : 10}});
console.log(Object.values(obj));
console.log(Object.getOwnPropertyDescriptors(obj));
//[]
//p: {value: 10, writable: false, enumerable: false, configurable: false}
```

Object.create方法的第二个参数添加的对象属性（属性p），如果不显式声明，默认是不可遍历的，因为p的属性描述对象的enumerable默认是false，Object.values不会返回这个属性。因此只要把enumerable改成true，Object.values就会返回属性p的值。

```
const obj = Object.create({}, {p: {
  value : 10,
  enumerable : true,
  configurable : true,
  writable : true,
}})
console.log(Object.values(obj));
//[10]
```

4.Object.values会过滤属性名为 Symbol 值的属性

```
//如果Object.values方法的参数是一个字符串，会返回各个字符组成的一个数组。
Object.values({ [Symbol()]: 123, foo: 'abc' });
console.log(Object.values('foo'));
//["f", "o", "o"]
```

字符串会先转成一个类似数组的对象，字符串的每个字符，就是该对象的一个属性。因此，Object.values返回每个属性的键值，就是各个字符组成的一个数组

5.如果参数不是对象，Object.values会先将其转为对象

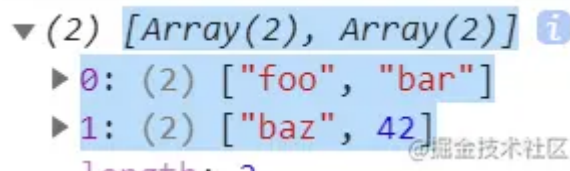
```
console.log(Object.values(42)); // []
console.log(Object.values(true)); // []
console.log(Object.values(undefined)); //error
console.log(Object.values(null)); //error
```

由于数值和布尔值的包装对象，都不会为实例添加非继承的属性，因此Object.values会返回空数组。

Object.entries()

1.Object.entries方法返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值对数组

```
var obj = { foo: 'bar', baz: 42 };
console.log(Object.entries(obj));
```



```
▼ (2) [Array(2), Array(2)] ⓘ
  ► 0: (2) ["foo", "bar"]
  ► 1: (2) ["baz", 42]
length: 2
```

2.如果原对象的属性名是一个 Symbol 值，该属性会被省略

```
console.log(Object.entries({ [Symbol()]: 123, foo: 'abc' }));
// [ [ 'foo', 'abc' ] ]
```

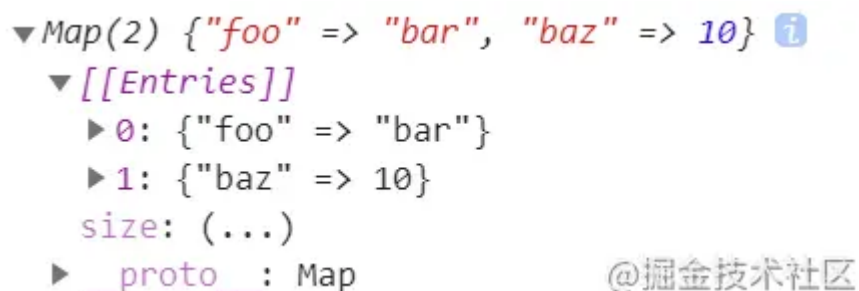
原对象有两个属性，Object.entries只输出属性名非 Symbol 值的属性。

3.遍历对象的属性

```
let obj = {
  one : 1,
  two : 2,
}
for(let [k , v] of Object.entries(obj)){
  console.log(`${JSON.stringify(k)} : ${JSON.stringify(v)}`);
}
//"one" : 1 "two" : 2
```

4.将对象转为真正的Map结构

```
const obj = {foo : "bar", baz : 10};
const map = new Map(Object.entries(obj));
console.log(map);
```



```
▼ Map(2) {"foo" => "bar", "baz" => 10} ⓘ
  ▼ [[Entries]]
    ► 0: {"foo" => "bar"}
    ► 1: {"baz" => 10}
    size: (...)
    ► __proto__: Map
```

5.实现Object.entries方法

```
const entries = (obj) => {
  let result = [];
  const objType = typeof(obj);
  if(obj === undefined || obj === null){
    throw new TypeError();
  }
}
```

```
if(objType === "number" || objType === "boolean"){  
    return [];  
}  
for(let k of Object.keys(obj)){  
    result.push([k,obj[k]]);  
}  
return result  
}
```