ES6对象身上的一些变化及手动封装新增方法

jCodeLife

0.393 2020.09.03 14:07:32 字数 764 阅读 44

在ES6中，对象身上也有一些变化，另外也新增了一些方法，一起来梳理一下

首先

1. ES6支持对象的简洁语法

```
let obj = {
    name: name,
    age: age,
    show: function () {//code...}
}
```

以上代码可以简写成

```
let obj = {
    name,
    age,
    show() {//code...}
}
```

对象简洁语法可总结为一下两点规则：

（1）属性名和属性值一致时，可以简写成一个；

（2）对象中的函数由原来格式 函数名 ：function(形参列表){代码块}，变成 函数名(形参列表){代码块}；即省略了:function

2. 在定义字面量对象时，属性名可以使用表达式了，表达式用方括号[]括起来，放在属性名位置

```
const prop = 'aaa';
const obj = {
    [prop + 'bbb']: 1,
    [prop + 'fn']() {
        console.log(arguments.callee.name);
    }
}
obj[prop + 'fn']();//aaafn
obj.aaafn();//aaafn
console.log(obj[prop+'bbb']);//1
console.log(obj.aaabbb);//1
```

3. 对象身上新增了Object.is()方法

Object.is方法用于优化===的两个问题:
NaN不等于NaN、+0等于-0的问题

```
console.log(NaN === NaN);//false
console.log(+0 === -0);//true
//=====================================//
console.log(Object.is(NaN,NaN));//true
console.log(Object.is(+0,-0));//false
```

推荐阅读

如何理解JavaScript中的对象?
阅读 183

js 判断字符串中是否包含某个字符串
阅读 86

从事7年前端开发，有些经验想对转行学习前端的伙伴说说！
阅读 136

前端面试基础题：异步加载JS的□式有哪些？
阅读 94

Vue插件-饼状图
阅读 701

```
        return true;
    }
    if (one === 0 && two === 0) {
        if (1 / one < 0 && 1/two >0){
            return false
        }
        if (1 / one > 0 && 1/two <0) {
            return false
        }
    }
    return one === two;
}
console.log(Object._is(NaN,NaN));//true
console.log(Object._is(+0, -0));//false
console.log(Object._is(-0, +0));//false
```

这里有个小窍门，就是判断一个数是+0还是-0，正常是比较难判断的，因为他们都全等于0，但是我可以利用一个正数除以+0为Infinity，除以-0为-Infinity的规律，在通过Infinity和-Infinity分别跟0比较，判断出是-0还是+0

4. 对象身上新增了Object.assign()方法

assign用于把多个对象合并到到一个对象上，第一个参数是要把其他对象合并到的对象，剩下其他参数是要被合并的对象

```
const a = { name: 'a', funA() { } }
const b = { name: 'b', funB() { } }
const c = { name: 'c', funC() { } }
const d = { name: 'd', funD() { } }
const obj = Object.assign(a,b,c,d);
console.log(a,b,c,d)
console.log(obj)
console.log(a === obj)
```

```
▶{name: "d", funA: ƒ, funB: ƒ, funC: ƒ, funD: ƒ} ▶{name: "b", funB: ƒ} ▶{name: "c", funC: ƒ} ▶{name: "d", funD: ƒ}
▶{name: "d", funA: ƒ, funB: ƒ, funC: ƒ, funD: ƒ}
true
```

结果

可以看出几个结论：

（1）通过assign合并对象，属性名重复，后面的属性值会覆盖前面相同属性名的属性值

（2）执行结果返回的就是合并了其他对象的第一个参数对象

所以可以利用assign来拷贝一个对象

```
const a = { name: 'a', funA() { } }
const b = Object.assign({},a);
b.name = 'b';
console.log(`a.name为: ${a.name};  b.name为: ${b.name}`)
//a.name为: a;  b.name为: b
```

手动封装如下

```
Object._assign = function () {
    if (arguments.length == 0) {
        throw new TypeError(`Cannot convert undefined or null to object`);
    }
    if (arguments.length == 1) {
        return arguments[0];
    }
```

推荐阅读

如何理解JavaScript中的对象?
阅读 183

js 判断字符串中是否包含某个字符串
阅读 86

从事7年前端开发，有些经验想对转行学习前端的伙伴说说！
阅读 136

前端面试基础题：异步加载JS的□式有哪些?
阅读 94

Vue插件-饼状图
阅读 701

```
                target[key] = arguments[i][key];
            }
        }
        return target
    }
}
let obj1 = { name: 'a' };
let obj2 = { name: 'b' }
console.log(Object._assign(obj1, obj2));
```

5. 对象身上新增了Object.create方法

Object.create()用于产生指定原型的新对象，第一个参数可以是某对象或者null，如果传入null表示该对象没有原型。

```
const obj = Object.create(null);
console.log(obj)
```

```
▼{}
    No properties
```

里面没有__proto__指向原型

6. 对象身上新增Object.getPrototypeOf()和Object.setPrototypeOf()用于获取和设置对象的原型属性`__proto__`

```
Object.setPrototypeOf(target, { a: 'alice'})
Object.getPrototypeOf(target) //{ a: 'alice', __proto__: Object}
```

猜测底层实现应该不是用`__proto__`吧，因为他们出现的原因就是，为了避免直接操作私有属性`__proto__`，如果底层用它，那跟我直接使用修改`__proto__`有啥区别，那就没有意义。

7. 对象身上新增Object.getOwnPropertyDescriptors()用于获取某对象的各个属性的详细信息，包括value，writable，enumerable，configurable等，返回新对象。

```
let obj = {a:1,b:2,c:3};
console.log(Object.getOwnPropertyDescriptors(obj))
```

```
▼{a: {…}, b: {…}, c: {…}}
  ▶a: {value: 1, writable: true, enumerable: true, configurable: true}
  ▶b: {value: 2, writable: true, enumerable: true, configurable: true}
  ▶c: {value: 3, writable: true, enumerable: true, configurable: true}
  ▶__proto__: Object
```

注意：上面Object.create()的第二个参数就是这种Object.getOwnPropertyDescriptors()所返回的格式

```
let obj = {a:1,b:2,c:3};
let newObj = Object.create(null,Object.getOwnPropertyDescriptors(obj));
console.log(newObj)//{a: 1, b: 2, c: 3}
console.log(obj === newObj)//false
```

`我们发现其实可以通过Object.create(Object.getPrototypeOf(obj),Object.getOwnPropertyDescriptors(obj))来拷贝一个对象`

8. ES2017在对象身上还新增了Object.keys()、Object.values()、Object.entries()

推荐阅读

如何理解JavaScript中的对象?
阅读 183

js 判断字符串中是否包含某个字符串
阅读 86

从事7年前端开发，有些经验想对转行学习前端的伙伴说说！
阅读 136

前端面试基础题：异步加载JS的□式有哪些?
阅读 94

Vue插件-饼状图
阅读 701

ES6对象身上的一些变化及手动封装新增方法

Object.entries 返回由多个key和value组成的键值对的数组

```
let obj = {a:1,b:2,c:3};
console.log(Object.keys(obj));//["a", "b", "c"]
console.log(Object.values(obj));//[1, 2, 3]
console.log(Object.entries(obj));//[["a", 1], ["b", 2], ["c", 3]]
```

要手动封装，也是跟切菜一样简单

（1）keys实现

```
Object._keys = function(o){
    let res = [];
    for(let key in o){
        res.push(key)
    }
    return res;
}
console.log(Object._keys(obj));//["a", "b", "c"]
```

（2）values实现

```
Object._values = function(o){
    let res = [];
    for(let key in o){
        res.push(o[key]);
    }
    return res;
}
let obj = {a:1,b:2,c:3};
console.log(Object._values(obj));//[1, 2, 3]
```

（2）手动实现entries

```
Object._entries = function(o){
    let res = [];
    for(let key in o){
        res.push([key,o[key]]);
    }
    return res;
}
let obj = {a:1,b:2,c:3};
console.log(Object._entries(obj));//[["a", 1], ["b", 2], ["c", 3]]
```

以上就是ES6在对象身上新增的所有东西

jCodeLife 书山有路勤为径，学海无涯苦作舟
总资产30 (约2.65元) 共写了13.6W字 获得329个赞 共25个粉丝

推荐阅读

如何理解JavaScript中的对象?
阅读 183

js 判断字符串中是否包含某个字符串
阅读 86

从事7年前端开发，有些经验想对转行学习前端的伙伴说说！
阅读 136

前端面试基础题：异步加载JS的□式有哪些?
阅读 94

Vue插件-饼状图
阅读 701

全部评论 0 只看作者 关闭评论 按时间倒序 按时间正序

被以下专题收入，发现更多相似内容 投稿管理

+收入我的专题 前段开发那些事儿

推荐阅读

如何理解JavaScript中的对象?
阅读 183

js 判断字符串中是否包含某个字符串
阅读 86

从事7年前端开发，有些经验想对转行学习前端的伙伴说说！
阅读 136

前端面试基础题：异步加载JS的□式有哪些?
阅读 94

Vue插件-饼状图
阅读 701