谈谈如何理解对象

使用预定义对象只是面向对象语言的能力的一部分， ECMAScript 真正强大之处在于能够创建自己专用的类和对象。面向对象的语言有一个标志，那就是它们都有类的概念，而通过类可以创建任意多个具有相同属性和方法的对象。由于 ECMAScript 中没有类的概念，因此它的对象也与基于类的语言中的对象有所不同。 ECMA-262 把对象定义为："无序属性的集合，其属性可以包含基本值、对象或者函数。"严格来讲，这就相当于说对象是一组没有特定顺序的值。对象的每个属性或方法都有一个名字，而每个名字都映射到一个值。正因为这样（以及其他将要讨论的原因），我们可以把 ECMAScript 的对象想象成散列表：无非就是一组名值对，其中值可以是数据或函数。每个对象都是基于一个引用类型创建的，这个引用类型可以是原生类型，也可以是开发人员定义的类型。

创建自定义对象的最简单方式就是创建一个 Object 的实例，然后再为它添加属性和方法:

```
let car = new Object();
car.wheel = 4;
car.run = function() {
    console.log(this.wheel + ' wheels is running!');
}
```

属性类型

ECMA-262 第 5 版在定义只有内部才用的特性时，描述了属性的各种特征。 ECMA-262 定义这些特性是为了实现 JavaScript 引擎用的，因此在 JavaScript 中不能直接访问它们。为了表示特性是内部值，该规范把它们放在了两对儿方括号中，例如 [[Enumerable]] 。

ECMAScript 中有两种属性： 数据属性 和 访问器属性 。

数据属性

数据属性包含一个数据值的位置, 在这个位置可以读取值也能写入, 它有4个特性:

- [[Configurable]]：表示能否通过 delete 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为访问器属性
- [[Enumerable]]：表示能否通过 for-in 循环返回属性
- [[Writable]]：表示能否修改属性的
- [[Value]]：包含这个属性的数据值, 读属性值的时候从这里读, 写入的时候把新值保存在这, 默认值是 undefined

像上面那样直接创建的属性, 包括字面量形式创建的, 除了 `[[Value]]` 的值被设置成指定的值, 其他的默认值都是 `true` , 如果要修改属性默认的特性, 需要调用 `Object.defineProperty()` 方法, 该方法接受3个参数, 属性所在的对象, 属性名, 描述对象, `Demo` 如下:

```
let car = {brand: 'Ferrari'};
Object.defineProperty(car, 'brand', {
    writable: false
});
car.brand = 'BMW';          // 严格模式这样赋值会报错, 非严格模式下会忽略该赋值
console.log(car.brand);     // Ferrari
```

`configurable` 也遵循上面的规则, 一旦把属性定义为不可配置就再也不能把它变回可配的了, `Object.defineProperty()` 只能修改属性的 `writable` 特性, 其他的修改都会导致错误, 在调用 `Object.defineProperty()` 时, 如果明确指定 `configurable` , `writable` , `value` , `enumerable` , 他们的默认值都是 `false`

访问器属性

访问器属性不包含数据值, 它包含 `getter` 和 `setter` 函数, 这两个函数也不是必须的, 在读取访问器属性时会调用 `getter` , 写入时调用 `setter` , 访问器属性有4个特性:

- `[[Configurable]]` : 表示是否能通过 `dlelete` 删除属性, 能否修改属性的特性, 能否把属性修改为数据属性, 直接在对象上定义的属性这个特性默认是 `true`
- `[[Enumerable]]` : 表示能否通过 `for-in` 循环返回属性, 直接定义在对象上的属性这个特性默认是 `true`
- `[[Get]]` : 读取属性是调用的函数, 默认是undefined
- `[[Set]]` : 写入属性时调用的函数, 默认是undefined

访问器属性不能直接定义, 必须使用 `Object.defineProperty()` 定义, Demo 如下:

```
let car = {
    _speed: 20,
    level: 1
};

Object.defineProperty(car, 'speed', {
    get: function() {
        return this._speed;
    },
    set: function(value) {
        if(value > 500 || value < 0) {
            return console.log('您的车子速度异常');
        }
        this._speed = value;
        this.level = Math.round((value - 20) / 50);
    }
});

car.speed = 200;
console.log(car.level);     // 4
```

定义多个属性

为对象定义多个属性可以调用 `Object.defineProperties()` 方法, 与 `Object.defineProperty()` 方法不同, 该方法接受2个参数: 要定义属性的对象, 要添加的属性描述集合, Demo 如下:

```
let car = {};
Object.defineProperties(car, {
    _speed: {
        value: 20,
        configurable: true,
        writable: true
    },
    level: {
        value: 1,
        configurable: true,
        writable: true
    },
    speed: {
        get: function() {
            return this._speed;
        },
        set: function(value) {
            if(value > 500 || value < 0) {
                return console.log('您的车子速度异常');
            }
            this._speed = value;
            this.level = Math.round((value - 20) / 50);
        }
    }
});
```

小结

- 对象的属性分为两种: 数据属性 和 访问器属性 (函数类型的我们称之为对象的方法)
- 介绍了 Object.defineProperty() 和 Object.defineProperties() 的使用, 前者接受3个参数, 后者接受2个
- 介绍了访问器属性的使用, 设置一个属性的值会导致其他属性联动的时候我们可以使用访问器属性