

你知道 JavaScript 中的访问器属性么？

我们看一下定义对象属性的方法。

ES6 之前

```
var person = {  
  
  name: "liang",  
  
  sayName: function() {  
  
    console.log(this.name);  
  
  }  
};
```

ES6 语法

```
var person = {  
  
  name: "liang",  
  
  sayName() {  
  
    console.log(this.name);  
  
  }  
};
```

可以看到函数属性更加简洁了，但这不重要。我们使用 ES5 新的函数

`Object.getOwnPropertyDescriptor` 把属性特性输出看一下。

```
Object.getOwnPropertyDescriptor(person, 'name');//{value: "liang", writable: true, enumerable:  
true, configurable: true}Object.getOwnPropertyDescriptor(person, 'sayName');//{value: f,  
writable: true, enumerable: true, configurable: true}
```

可以看到不管是函数，还是字符串或者其他基本类型，它们都有四个属性特性描述。

[[Value]]: 表示属性的数据值。默认值: undefined

[[Writable]]: 表示能否修改属性的值。默认值: true

[[Enumerable]]: 表示能否通过 for-in, Object.keys() 迭代。

默认值: true

[[Configurable]]: 表示能否通过 delete 删除属性, 能否修改属性的特性, 能否将数据属性和访问器属性互转。

如果为 false, 只可以把 [[Writable]] 从 true 变为 false,

[[Enumerable]] 和 [[Configurable]] 的值都不能再改变, [[Value]] 只取决于 [[Writable]], 数据属性不能变成访问器属性, 访问器属性也不能变成数据属性, 也不能通过 delete 删除。默认值: true

ES5 提供了 Object.defineProperty 方法, 既可以用来定义属性, 也可以用来修改属性的特性。

```
Object.defineProperty(person, 'name', { writable: false});
```

然后再对 person.name 赋值的话就无效了。

也可以定义新的属性, 不同于直接定义的属性默认值都为 true。这里如果没有定义, 默认值是 false。

```
Object.defineProperty(person, 'id', {  
  value: 6,  
  enumerable: true,  
  configurable: false});  
//value: 6, writable: false, enumerable: true, configurable: false
```

访问器属性

除了数据属性，还多了访问器属性。

```
var book = { _year:2004, edition:1};Object.defineProperty(book, "year", {  
  get () {  
    return this._year;  
  },  
  set (newValue) {  
    if ((newValue > 2004)) {  
      this._year = newValue;  
      this.edition += newValue - 2004;  
    }  
  })  
});book.year = 2005;alert(book.edition);  
Object.getOwnPropertyDescriptor(book, "year")//{get: f, set: f, enumerable: false,  
configurable: false}
```

没有了 `[[Value]]` 和 `[[Writable]]`，取而代之的是 `get` 和 `set` 函数。如果 `set` 属性没有定义，那么就无法修改 `book` 的值。

`[[Enumerable]]` 和 `[[Configurable]]` 和之前是一样的。

如果你对 C++ 或者 JAVA 了解，那么对 `get` 和 `set` 一定不陌生，但是你有没有过疑问，为什么要有访问器属性呢？

JAVA 里边有 `Private` 变量，然后提供 `public` 的 `get`, `set` 方法来访问这些变量，那么 js 为什么要有呢？直接访问变量不好吗？

你可能会说，像上边的例子，我们可以控制设置的值呀，大于 2004 我们才进行赋值，如果是数据属性就做不到呀。那么问题来了，为什么不直接定义一个函数呢，非新增个访问器属性呢？

函数还是访问器属性？

如果对象有三个属性，`firstName`，`lastName`，`fullName`，很明显

`fullName = firstName + lastName`。

```
var person = {  
  firstName: "John",  
  lastName : "Doe",  
  fullNmae: "John Doe"};  
person.firstName = "Liang"; //改变 firstName  
person.fullName = "Liang " + person.lastName; //改变 fullName  
person.fullName; //Liang Doe
```

这样的话，当我想改变 `firstName` 的话，我还得同时去改变 `fullName`，当属性间有关联的时候，维护它们之前的关系太麻烦了。我们可以把 `fullName` 定义成一个函数，这样的话，它就可以自动改变了。

```
var person = {  
  firstName: "John",  
  lastName : "Doe",  
  getFullName () {  
    return this.firstName + " " + this.lastName;  
  },  
}
```

```
setFullName (name) {  
  
    var words = name.split(' ');  
  
    this.firstName = words[0] || '';  
  
    this.lastName = words[1] || '';  
  
    };  
  
person.firstName = "Liang";person.getFullName(); //Liang Doe
```

这样问题解决了，但不够优雅，让我们看看访问器属性可以怎么做。

```
var person = {  
  
    firstName: "John",  
  
    lastName : "Doe",  
  
    get fullName() {  
        return this.firstName + " " + this.lastName;  
    },  
    set fullName(name) {  
        var words = name.split(' ');  
  
        this.firstName = words[0] || '';  
  
        this.lastName = words[1] || '';  
  
    }  
};  
  
person.firstName ="Liang";person.fullName;
```

访问器属性的优点很明显了。

语法上更简洁，将函数和属性值语法统一了起来。访问器属性虽然调用了函数，但在使用上和属性的语法是一样的。也更符合逻辑，如果想得到 `FullName`，很明显这应该是对象的一个属性，而不应该是方法，如果去调用函数得到它，就太不优雅了。

访问器属性其他优点

说了这么多，其实用函数和访问器属性可以实现同样的功能，但既然 ES5 中提供了访问器属性的语法，我们当然会优先是用访问器属性，而不是定义一个函数了。那么除了当属性间有关联可以使用它，还有什么优点呢？

就是最开始说的，有了 `get` 和 `set` 我们就可以在返回和设置值的时候进行一些额外的操作。

```
var obj = {  
  n: 67,  
  get id() {  
    return 'The ID is: ' + this.n;  
  },  
  //判断是否是数字  
  set id(val) {  
    if (typeof val == 'number')  
      this.n = val;  
  }  
}  
  
console.log(obj.id); // "The ID is: 67" obj.id = 893;  
  
console.log(obj.id); // "The ID is: 893" obj.id = 'hello';  
  
console.log(obj.id); // "The ID is: 893"
```

除了上边的有点外，我们还提到 js 里边没有私有变量，所以在外边可以直接访问变量而不经访问器属性。

```
var obj = {
```

```
fooVal: 'this is the value of foo',

get foo() {

    return this.fooVal;

},

set foo(val) {

    this.fooVal = val;

}}

obj.fooVal = 'hello';

console.log(obj.foo); // "hello"
```

我们没有通过访问器而改变了内部的值，结合访问器属性，我们可以实现数据的私有化。

利用块级作用域。

```
/* BLOCK SCOPE, leave the braces alone! */
{let fooVal = 'this is the value of foo';
```

```
var obj = {

    get foo() {

        return fooVal;

    },

    set foo(val) {

        fooVal = val

    }

}}
```

```
fooVal = 'hello'; // not going to affect the fooVal inside the block
console.log(obj.foo); // "this is the value of foo"
```

利用函数作用域

```
function myobj() {  
  
  var fooVal = 'this is the value of foo';  
  
  return {  
  
    get foo() {  
  
      return fooVal;  
  
    },  
  
    set foo(val) {  
  
      fooVal = val  
  
    }  
  
  }  
  
}  
  
fooVal = 'hello'; // not going to affect our original fooVal  
  
var obj = myobj();  
console.log(obj.foo); // "this is the value of foo"
```

最后，访问器属性相比于函数还有一个更大的优点。比如，一个已经写过的项目，里边用了一个 `date` 变量。

```
var obj = { date: "2019.6.5" }
```

我们在 100 个文件里用了 `date` 变量，并且进行了赋值操作。

a. js 有下边的语句

```
obj.date = "2019.8.8"
```

b. js 有下边的语句

```
obj.date = "2019.7.8"
```

c. js 有下边的语句


```
obj.date = "2018.9.9"
```

d.js 有下边的语句

```
obj.date = "2017.2.8"
```

e.js 有下边的语句

```
obj.date = "2019.2.23"
```

.... 还有很多文件也都对 `obj.date` 进行了赋值操作。

有一天，我们突然想要变更数据从 "XXXX.YY.ZZ" 到 "XXXX-YY-ZZ" 的类型。

一种方法就是找到之前所有赋值的地方，然后进行修改。

改 a.js

```
obj.date = "2019-8-8"
```

改 b.js

```
obj.date = "2019-7-8"
```

改 c.js

```
obj.date = "2018-9-9"
```

... ..

另一种办法就是把 `date` 改成访问器属性，找到对象定义的地方改一次就行了，这样在赋值的时候 `date` 就会自动改变了。

```
var obj = {  
  
  _date: "2019-6-5",  
  
  get date() {  
  
    return this._date;  
  
  },  
  
  set date(val) {  
  
    this._date = val.split(".").join("-")  
  
  }}  

```

由此也可以看出访问器属性的好处，可以对数据进行更好的控制，合法性判断，格式之类的，以及关联多个属性，所以最好用访问器属性，可以为未来的扩展留下一个接口。

