

07 | 深入理解对象的私有和静态属性

2022-10-04 石川 来自北京



天下无鱼

<https://shikey.com/>

《JavaScript进阶实战课》

[课程介绍 >](#)



讲述：石川

时长 08:45 大小 7.99M



你好，我是石川。

在前面几讲里，我们围绕着函数式编程，从基础的输入、计算、输出讲起，到过程中可能产生的副作用，再到如何通过纯函数和不可变作为解决思路来管理副作用等等，都有了系统的了解。之后，我们又通过响应式编程和函数式编程的结合，了解了这种模式下面，如何面对未知以及基于事件变化做出响应。

从这节课开始，我们再来深入了解下 **JavaScript** 的对象构建和面向对象的编程模式。

在 [第 1 讲](#) 里，我们第一次介绍到了对象和面向对象。对象其实就好比一个人，生来都是带有属性和功能的，比如肤色、身高等等就是我们的属性，会哭会笑这些就是我们的功能。我们作为对象和别的对象之间要产生交互，这就是面向对象的设计。那今天我们就从一个对象的创建讲起。



在面向对象的设计中，一个对象的属性是至关重要的，因为它也决定了对象是什么、能做什么。一个对象可以有对外分享的、别人可以获取的**公开属性**，也有不对外暴露的、别人不可以随便获取的**私有属性**。

除了公开和私有属性，还有**静态属性**。静态属性是属于类，而不是单独属于对象的。这么解释听上去可能有些绕口，我们可以打个比方，比如说中国有 14 亿人口，那么“14 亿人口”就是属于中国这个国家类的属性，但是我们不能说具体每一个中国人具有“14 亿人口”这个属性。

同样地，静态属性也包含公开属性和私有属性。就好比一个公司作为一个组织类，通常为了体现规模，会公开自己有多少员工，这就是公司的**公开静态属性**；但是公司不会公开一些运营数据，因为这个是敏感话题，只有审计的时候或特定场合才提供，这个运营数据可能就属于**私有静态属性**。

目前用于创建私有属性和静态属性的支持，都是在 2022 年 6 月后，也就是在 JavaScript 出现后的 25 年，才纳入 ECMAScript 规范中的（但其实除了已经退役的 IE 浏览器外，几乎大多数主流浏览器之前都支持了这两个功能）。但是在此之前，人们就通过其它方式，试着实现类似的功能了。

今天这节课，我们就来看看它们实现的底层逻辑和应用。

如何创建私有属性？

在面向对象中，有个很重要的概念就是创建私有属性。

我们可以看到，和 **Java** 不一样，当用 **JavaScript** 创建一个 **Widget** 对象时，无论是使用 **class**、对象字面量还是函数构造式，一般的情况下，在定义了属性和方法后就可以公开调用，并没有任何限制。

 复制代码

```
1 // 示例1: 类class
2 class WidgetA {
3     constructor() {
4         this.appName = "天气应用"
5     }
6     getName(){
7         return this.appName;
8     }
9 }
10 var widget1 = new WidgetA();
11 console.log(widget1.appName); // 返回 “天气应用”
12 console.log(widget1.getName()); // 返回 “天气应用”
13
14 // 示例2: 对象字面量
15 var WidgetB = {
16     appName : "天气应用",
17     getName : function (){
18         return this.appName;
19     }
20 }
21
22 console.log(WidgetB.appName); // 返回 “天气应用”
23 console.log(WidgetB.getName()); // 返回 “天气应用”
24
25 // 示例3: 函数构造式
26 function WidgetC(){
27     this.appName = "天气应用";
28     this.getName = function (){
29         return "天气应用";
30     };
31 }
32
33 var widget3 = new WidgetC();
34 console.log(widget3.appName); // 返回 “天气应用”
35 console.log(widget3.getName()); // 返回 “天气应用”
```

用 # 符号创建私有属性

那怎么才能在对象中创建私有属性呢？根据最新的 [ES13](#) 规范，我们可以通过 # 符号，来定义一个私有的属性。



首先，我们声明了一个 `#appName`，在构建者 `constructor` 里，我们给它赋值为“天气应用”。这时，当我们直接调取 `appName` 时，会看到返回的结果就是未定义的。但如果我们通过 `getName` 方法，就可以获取 `appName` 的值。

复制代码

```
1 class WidgetD {
2   #appName;
3   constructor(){
4     this.#appName = "天气应用";
5   }
6   getName(){
7     return this.#appName;
8   }
9 }
10
11 var widget4 = new WidgetD();
12 console.log(widget4.appName); // 返回 undefined
13 console.log(widget4.getName()); // 返回 “天气应用”
```

所以下面，我们就一起来看看在 # 问世之前，工程师们是怎么实现私有属性的。主要有闭包、WeakMap 和 Symbol 这三种方式。

用闭包和 IIFE 创建私有属性

我们先来看看如何在对象字面量中创建私有属性。是的，我们在前面讲函数式编程时，提到过的闭包在这里派上用场了。

首先，我们声明一个 `WidgetE` 的变量，然后再来创建一个立即被调用的函数式表达（IIFE），在这个表达里面，我们先给内部的 `appName` 变量赋值为“天气应用”。

之后，在函数中我们再给 `WidgetE` 赋值，这里赋值的是一个对象，里面我们定义了 `getName` 的方法，它返回的就是外部函数的 `appName`。

这个时候，当我们试图获取 `WidgetE.appName` 时，会发现无法获取嵌套函数内部声明的变量。但是当我们通过 `getName` 的方法，利用嵌套函数中内嵌函数可以访问外部函数的变量的特点，就可以获取相应的返回值。



```
1 // 对象字面量
2 var WidgetE;
3 (function(){
4     var appName = "天气应用";
5     WidgetE = {
6         getName: function(){
7             return appName;
8         }
9     };
10 }());
11 WidgetE.appName; // 返回 undefined
12 WidgetE.getName(); // 返回 “天气应用”
```

好，下面我们再看看如何通过构造函数的方式，构造私有属性。

这里也可以通过我们学过的闭包，直接上代码。这个例子其实看上去要比上面的例子简单，我们先定义一个函数，在里面声明一个变量 `appName`，然后创建一个 `getName` 的表达式函数，返回 `appName`。

```
1 // 构造函数
2 function WidgetF() {
3     var appName = "天气应用";
4     this.getName = function(){
5         return appName;
6     }
7 }
8 var widget6 = new WidgetF();
9 console.log(widget6.appName); // 返回 undefined
10 console.log(widget6.getName()); // 返回 “天气应用”
```

这时候，我们通过函数构造可以创建一个新的函数 `widget6`，但是通过这个新构建的对象来获取 `appName` 是没有结果的，因为在这里，`appName` 是封装在 `WidgetF` 内部。不过 `widget6` 可以通过 `getName` 来获取 `appName`，同样，这里是利用闭包的特点，来获取函数之外的变量。

可是这个例子中还有一个问题，就是我们每次在创建一个新对象的时候，私有属性都会被重新创建一次，这样就会造成重复工作和冗余内存。解决这个问题的办法就是**把通用的属性和功能赋值给 prototype**，这样通过同一个构建者创建的对象，可以共享这些隐藏的属性。

比如我们来看下面的例子，我们给 WidgetG 的原型赋值了一个函数返回的对象，函数中包含了私有属性，返回的对象中包含了获取属性的方法。这样我们在创建一个 widget7 的对象之后，就能看到它可以获取天气应用支持的机型了。



复制代码

```
1 function WidgetG() {
2   var appName = "天气应用";
3   this.getName = function(){
4     return appName;
5   }
6 }
7 WidgetG.prototype = (function(){
8   var model = "支持安卓";
9   return {
10    getModel: function(){
11      return model;
12    }
13  }
14 }());
15 var widget7 = new WidgetG();
16 console.log(widget7.getName()); // 返回 “天气应用”
17 console.log(widget7.getModel()); // 返回 “支持安卓”
```

用 WeakMap 创建私有属性

在 ES6 中，JavaScript 引入了 Set 和 Map 的数据结构。Set 和 Map 主要用于数据重组和数据储存。Set 用的是集合的数据结构，Map 用的是字典的数据结构。Map 具有极快的查找速度，后面课程中我们在讲数据结构和算法的时候，还会详细介绍。在这里，我们主要先看 WeakMap，它的特点是只接受对象作为键名，键名是弱引用，键值可以是任意的。

在下面的例子中，我们首先声明了一个 WidgetG 变量。接下来，建立一个块级作用域，在这个作用域里，我们再声明一个 privateProps 的 WeakMap 变量。然后我们给 WidgetG 赋值一个函数声明，在里面给 WeakMap 的键名设置为 this，键值里面的 appName 为“天气应用”。下一步，我们基于 WidgetF 的 prototype 来创建一个 getName 方法，里面返回了 appName 的值。

利用这样的方式，就可以同时达到对 appName 的封装和通过 getName 在外部对私有属性值的获取了。

复制代码

```

1 var WidgetH;
2 {
3   let privateProps = new WeakMap();
4
5   WidgetH = function(){
6     privateProps.set(this,{appName : "天气应用"});
7   }
8
9   WidgetH.prototype.getName = function(){
10    return privateProps.get(this).appName;
11  }
12 }
13
14 var widget8 = new WidgetH();
15 console.log(widget8.appName); // 返回 undefined
16 console.log(widget8.getName()); // 返回 “天气应用”

```



用 Symbol 创建私有属性

Symbol 也是在 ES6 引入的一个新的数据类型，我们可以用它给对象的属性的键名命名。

同样我们来看一个例子。和上个例子相似，这里我们建立了一个块级作用域，但区别是我们把 **privateProps** 从 **WeakMap** 换成了 **Symbol** 来实现私有属性。

复制代码

```

1 var WidgetI;
2 {
3   let privateProps = Symbol();
4
5   WidgetI = function(){
6     this[privateProps] = {appName : "天气应用"};
7   }
8
9   WidgetI.prototype.getName = function(){
10    return this[privateProps].appName;
11  }
12 }
13
14 var widget9 = new WidgetI();
15 console.log(widget9.getName()); // 返回 “天气应用”

```

如何创建静态属性？

前面我们提到了，静态的属性是属于构造函数的属性，而不是构造对象实例的属性。下面我们来看看，如何通过 JavaScript 来实现静态属性。



创建公开静态属性

我们先看看如何通过 `static` 这个关键词来创建公开的静态属性。如以下代码所示，当我们直接在 `WidgetJ` 上面获取 `appName` 和 `getName` 的时候，可以看到结果是返回“天气应用”。而如果我们用 `WidgetJ` 构建一个 `widget10`，看到返回的是未定义。这就说明，**静态属性只能作用于 class 本身**。

复制代码

```
1 class WidgetJ {
2   static appName = "天气应用";
3   static getName(){
4     return this.appName;
5   }
6 }
7
8 console.log(WidgetJ.appName); // 返回 “天气应用”
9 console.log(WidgetJ.getName()); // 返回 “天气应用”
10
11 var widget10 = new WidgetJ();
12 console.log(widget10.appName); // 返回 undefined
13 console.log(widget10.getName()); // 返回 undefined
```

创建私有静态属性

好，说完了公有静态属性，我们再来看看私有静态属性。私有的静态属性，顾名思义就是它不只是供构造者使用的，同时也是被封装在构建者之内的。

我们来看看它要如何实现，其实就是把 **# 符号** 和 **static 关键词** 相加来使用。

复制代码

```
1 class WidgetM {
2   static #appName = "天气应用";
3   static staticGetName(){
4     return WidgetM.#appName;
5   }
6   instanceGetName(){
7     return WidgetM.#appName;
8   }
9 }
```



```
10 console.log(WidgetM.staticGetName()); // 返回 “天气应用”
11
12 var widget13 = new WidgetM();
13 console.log(widget13.instanceGetName()); // 返回 “天气应用”
14
```



总结


这节课我们通过对象内部的私有和静态属性，在第一讲的基础上，进一步地了解了对象的构建。同时，更重要的是，我们通过去掉私有属性的语法糖，也了解了如何通过函数式中的闭包、对象中的 **prototype**、值类型中的 **Map** 和 **Symbol**，这些更底层的方式实现同样的功能。在后面的两节课里，我们会继续从单个对象延伸到对象间的“生产关系”，来进一步理解面向对象的编程模式。

思考题

我们今天尝试通过去掉语法糖，用更底层的方式实现了对象中的私有属性，那么你能不能自己动手试试去掉静态属性的语法糖，来实现类似的功能？

欢迎在留言区分享你的答案、交流学习心得或者提出问题，如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。

分享给需要的人，Ta购买本课程，你将得 **18** 元

 生成海报并分享

 赞 0  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | 如何通过模块化、异步和观察做到动态加载？

下一篇 08 | 深入理解继承、Delegation和组合

精选留言 (3)

 写留言

西红柿炒番茄
2022-10-12 来自广东

weekmap、symbol实现静态属性，应该也是利用了闭包的特性吧？因为都是在创建的函数中保存了外部存储的数据



褚琛
2022-10-08 来自海南

//闭包创建私有静态私有属性

```
var WidgetN;
```

```
(function () {  
  var appName = '天气应用';  
  
  WidgetN = function() {};  
  WidgetN.staticGetName = function() {  
    return appName;  
  }  
})();
```

```
console.log(WidgetN.staticGetName());  
console.log(WidgetN.appName);
```

//Symbol创建静态私有属性

```
var WidgetP;
```

```
{  
  let staticPrivateProps = Symbol();  
  
  WidgetP = function() {};  
  
  WidgetP[staticPrivateProps] = { appName: '天气应用' };  
  
  WidgetP.staticGetName = function() {  
    return WidgetP[staticPrivateProps].appName;  
  }  
}  
console.log(WidgetP.staticGetName());  
console.log(WidgetP.appName);
```

鐘

2022-10-04 来自北京



天下无鱼

<https://shikey.com/>

靜態公開屬性或方法的話可以這樣實現:

```
class A{  
  A.a = 1  
  A.b = function(){}  
}
```

私有的靜態方法不知道怎麼實現

作者回复: 可以参考MDN里介绍的“私有静态方法”: https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Classes/Private_class_fields

```
class ClassWithPrivateStaticMethod {  
  static #privateStaticMethod() {  
    return 42;  
  }  
  
  static publicStaticMethod1() {  
    return ClassWithPrivateStaticMethod.#privateStaticMethod();  
  }  
  
  static publicStaticMethod2() {  
    return this.#privateStaticMethod();  
  }  
}  
  
console.log(ClassWithPrivateStaticMethod.publicStaticMethod1() === 42); // true  
console.log(ClassWithPrivateStaticMethod.publicStaticMethod2() === 42); // true
```

