

4 单例模式

定义：一个类仅有一个实例，并提供一个访问它的全局方法

例如线程池，全局缓存，浏览器中的window对象，还有点击Btn后弹出的登录浮窗，无论点击多少次这个浮窗应该至多被创建一次，是唯一的，这种时候就需要借助单例模式

实现单例模式

实现并不复杂，无非是用一个变量标志一下类是否创建过了对象，一个典型的实现如下：

```
1 var Singleton = function(name){
2     this.name = name
3 };
4
5 Singleton.prototype.getName = function(){
6     alert(this.name)
7 }
8
9 Singleton.getInstance = (function(){
10     // 闭包保存instance变量
11     var instance = null;
12     return function(name){
13         if(!instance){
14             instance = new Singleton(name)
15         }
16         return instance;
17     }
18 })()
19
20 var a = Singleton.getInstance('sven1')
21 var b = Singleton.getInstance('sven2')
22 alert(a === b); // true
```

上面方式有一个问题，就是增加了这个类的“不透明性”，使用者必须知道它是一个单例类，有一个getInstance来调用才行

透明的单例模式

现在我们的需求是让用户使用这个类时候感知不到差异，实现一个“透明”单例类，这里我们可以引入代理类来专注于管理单例的逻辑，假设这里有一个创建div的工具类：

```
1 var CreateDiv = function(html){
2     this.html = html;
3     this.init();
4 }
5
6 CreateDiv.prototype.init = function(){
7     var div = document.createElement('div');
8     div.innerHTML = this.html;
9     document.body.appendChild(div)
10 }
```

我们可以引入代理类给他赋予单例的能力：

```
1 var ProxySingletonCreateDiv = (function(){
2     var instance;
3     return function(html){
4         if(!instance){
5             instance = new CreateDiv(html);
6         }
7         return instance;
8     }
9 })()
```

现在我们将管理单例的逻辑迁移了出去，通过组合实例单例效果
本例子是缓存代理的应用之一，在第六章将继续了解代理的好处

Javascript中的单例模式

前面几种实现其实是以接近传统面向对象语言中的实现，单例对象从“类”中创建而来。

但JS其实是一门无类语言，既然我们只是需要一个“唯一”对象，为什么要为它先创建一个“类”呢，单例模式的核心是确保有有一个实例可以全局访问就行。

其实我们可能会习惯将全局变量当作单例来使用：

```
1 var a = {}
```

但是全局变量其实存在很多问题：

- 命名空间污染
- 变量容易被覆盖
- 全局变量甚至有时候被称为Javascript中最糟糕的特性

1. 使用命名空间进行优化

最简单的就是使用对象字面量的形式：

```
1 const namespace1 = {  
2     a: function() {},  
3     b: function() {}  
4 }
```

2. 使用闭包封装私有变量

把变量封装在内部，只暴露接口跟外界通信

```
1 var user = (function(){  
2     var _name = 'sven',  
3     _age = 29  
4  
5     return {  
6         getUserInfo:function(){  
7             return _name + '-' + _age;  
8         }  
9     }  
10 })();
```

我们通过下划线来约定私有变量，外部无法访问，避免了对全局的污染