第八课 单例模式

学习目录

- **使用场景**
- **使用方式**
- **优缺点**

一. 使用场景

有时候我们在开发一个项目的时候，在全局环境下会产生许多变量的定义，有的变量是普通类型，有的是引用类型，如果我们把所有的变量都定义在全局环境下，那么全局环境的命名空间会越来越少，如果是多人协作开发，那么很容易产生冲突，造成全局变量的污染。

还有时候我们想要一个组件在全局环境下只初始化一次，之后就可以在任意地方直接使用这个组件的实例对象的属性和方法，那么我们就需要只实例化一次这个组件，保证这个组件在全局下只拥有一个实例对象，不会带来额外实例化的内存开销，保证程序执行的效率。

这两种情况都可以用到单例模式，单例模式的定义很简单，让一个类或者构造函数只有一次实例化过程，只有一个实例对象，并且提供在全局环境下可以使用这个实例对象属性与方法的访问点。

单例模式的核心思想是定义一个类或者构造函数，让这个类或者构造函数只有一个实例对象在全局可以被访问和使用。

二. 使用方式

要满足单例模式的要求，那就需要我们写一个类，不管这个类被实例化多少次，它每次

的实例化对象都相等，也就是只拥有一个实例对象。

```
function Fn1(){
    //单例模式
}
var f1 = new Fn1();
var f2 = new Fn1();
console.log(f1 === f2);//true
```

由于我们构造函数的实例化过程是构建一个构造函数的实例对象，其实我们只要保证每次实例化的时候让构造函数的指针都返回同一个对象即可，也就是说在第一次构造函数实例化的时候新建一个实例对象，之后在让构造函数实例化就把之前那个实例对象返回出去。

利用原型构建单例模式

```
function Fn1(num){
    var self = this;
    this.num = num;

    Fn1 = function(){
        return self;
    }

    Fn1.prototype = this;//这里的 this 是当前实例对象，这一步是让改写之后的构造函数利用原型继承的形式了当前实例对象的原型
```

```
        self = new Fn1();//这里是单例实例对象实例化过程

        self.constructor = Fn1;//这里是修改单例实例对象的 constructor 属性，保证 Fn1 的指针可以指向这个单例实例对象

        return self;//这里返回这个单例实例对象，之后继续实例化 Fn1 也只能得到这个实例对象了，不会新建新的实例对象了

    }
    Fn1.prototype.getNum = function(){
        return this.num;
    }
    var f1 = new Fn1(100);
    var f2 = new Fn1(300);

    console.log(f1.constructor == Fn1);//false 因为改写了 Fn1 这个构造函数，因此这里的 Fn1 的指针已经指向新的函数了
    console.log(f1 === f2);
    console.log(f1.getNum());//100
    console.log(f2.getNum());//100 如果不使用单例模式，这里等于是一个新的实例对象，这里的 num 就会变成 300 了
```

利用闭包构建单例模式

```
let SingleFn = (function(){
    let result = null;
    return function(num){
        if(result){
            return result;
        }
        return result = new Fn1(num);
    }
})();

function Fn1(num){
    this.num = num;
}
Fn1.prototype.getNum = function(){
    return this.num;
}
var f1 = new SingleFn(100);
var f2 = new SingleFn(300);
console.log(f1 === f2);//true
console.log(f1.getNum());//100
```

```
console.log(f2.getNum());//100
```

惰性单例模式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        .modelTip{
            width:300px;
            height:300px;
            background: orange;
            text-align: center;
            line-height: 300px;
        }
    </style>
</head>
<body>
    <button id="btn1">需要等待框</button>
```

```
<button id="btn2">等待框实例对象相等</button>
<script>
    var singleFn = function(fn){
        var result;
        return function(){
            return result || (result = fn.apply(this, arguments));
        }
    };
    function createModel(tip){
        var lodingModelNode = document.createElement('div');
        lodingModelNode.className = 'modelTip';
        lodingModelNode.innerHTML = tip;
        lodingModelNode.style.display = 'none';
        document.body.appendChild(lodingModelNode);
        return lodingModelNode;
    }
    var $btn1 = document.getElementById('btn1');
    var arr = [];
    var createSingleFn = singleFn(createModel);
    $btn1.onclick = function(){
        var loading = createSingleFn('加载中...');
        loading.style.display = 'block';
```

```
            arr.push(loading);
        }

        var $btn2 = document.getElementById('btn2');
        $btn2.onclick = function(){
            console.log(arr);//[div.modelTip, div.modelTip]
            console.log(arr[0] === arr[1]);//true
        }
    </script>
</body>
</html>
```

惰性单例在 js 中的主要是在统一的弹出框、登录注册框等 ui 组件中应用的比较多，不过也可以用在其他需要构建单一实例的方法中。

三．优缺点

1.单例模式可以构建出全局维护使用的单一实例对象，避免了变量全局污染。

2.单例模式的单例对象只会在实例化的时候生成一次，之后进行返回，减少了不必要的内存开销，有利于应用性能。

3.单例模式是比较成熟的设计模式，也是几大设计模式中最重要的设计模式之一，在不少库与框架中也有单例模式的应用，比如 jquery，vuex 等等，其他语言中也有单例模式的不同提现。

4.单例模式的构建需要对实际业务逻辑较为熟悉，条理清晰的情况下使用比较好。

5.单例模式不适合动态的扩展对象，当对象的使用场景需要发生变化时，单例模式可能会发生数据错误问题。

谢谢观看！

我是星星课堂老师：周小周