27 模块模式

更新时间：2019-07-29 09:35:38

> 天才就是长期劳动的结果。
>
> ——牛顿

模块是任何健壮的应用程序体系结构不可或缺的一部分，特点是有助于保持应用项目的代码单元既能清晰地分离又有组织，下面我们来看看各种不同的模块模式解决方案。

> **注意：** 本文可能用到一些编码技巧比如 IIFE（Immediately Invoked Function Expression, 立即调用函数表达式），ES6 Module 的语法 等，如果还没接触过可以点击链接稍加学习 ~

1. 模块模式

1.1 命名空间模式

命名空间模式是一个简单的模拟模块的方法，即创建一个全局对象，然后将变量和方法添加到这个全局对象中，这个全局对象是作为命名空间一样的角色。

```
var MYNS = {}

MYNS.param1 = 'hello'
MYNS.param2 = 'world'
MYNS.param3 = { prop: 'name' }

MYNS.method1 = function() {
  //...
}
```

这种方式可以隐藏系统中的变量冲突，但是也有一些缺点，比如：

1. 命名空间如果比较复杂，调用可能就会变成 `MYNS.param.prop.data...` 长长一溜，使用不便且增加代码量；

2. 变量嵌套关系越多，属性解析的性能消耗就越多；
3. 安全性不佳，所有的成员都可以被访问到；

1.2 模块模式

除了命名空间模式，也可以使用闭包的特性来模拟实现私有成员的功能来提升安全性，这里可以通过 IIFE 快速创建一个闭包，将要隐藏的变量和方法放在闭包中，这就是模块模式。

```
var myModule = (function() {
  var privateProp = ''      // 私有变量
  var privateMethod = function() { // 私有方法
    console.log(privateProp)
  }

  return {
    publicProp: 'foo',            // 公有变量
    publicMethod: function(prop) {  // 共有方法
      privateProp = prop
      privateMethod()
    }
  }
})()

myModule.publicMethod('new prop') // 输出：new prop
myModule.privateProp            // Uncaught TypeError: myModule.privateMethod is not a function
myModule.privateProp            // undefined
```

这里的私有变量和私有方法，在闭包外面无法访问到，称为**私有成员**。而闭包返回的方法因为作用域的原因可以访问到私有成员，所以称为**特权方法**。

值得一提的是，在模块模式创建时，可以将参数传递到闭包中，以更自由地创建模块，也可以方便地将全局变量传入模块中，导入全局变量有助于加速即时函数中的全局符号解析的速度，因为导入的变量成了该函数的局部变量。

```
var myModule = (function(opt, global) {
  // ...
})(options, this)
```

1.3 揭示模块模式

在上面的模块模式例子上稍加改动，可以得到**揭示模块模式**（Reveal Module Pattern），又叫暴露模块模式，在私有域中定义我们所有的函数和变量，并且返回一个匿名对象，把想要暴露出来的私有成员赋值给这个对象，使这些私有成员公开化。

```
var myModule = (function() {
  var privateProp = ''
  var printProp = function() {
    console.log(privateProp)
  }

  function setProp(prop) {
    privateProp = prop
    printProp()
  }

  return {
    print: printProp,
    set: setProp
  }
})()

myModule.set('new prop')        // 输出：new prop
myModule.setProp()            // Uncaught TypeError: myModule.setProp is not a function
myModule.privateProp          // undefined
```

揭示模块暴露出来的私有成员可以在被重命名后公开访问，也增强了可读性。

2. ES6 module

继社区提出的 CommonJS 和 AMD 之类的方案之后，从 ES6 开始，JavaScript 就支持原生模块（module）了，下面我们一起来简单看一下 ES6 的 module，更详细的建议看一下阮一峰的 《ES6 标准入门》 。

ES6 的 module 功能主要由两个命令组成 `export`、`import`， `export` 用于规定模块对外暴露的接口， `import` 用于输入其他模块提供的接口，简单来说就是一个作为输出、一个作为输入。

```
// 1.js

// 写法一
export var a = 'a'

// 写法二
var b = 'b'
export { b }

// 写法三
var c = 'c'
export { c as e }
```

引入时：

```
// 2.js

import { a } from './1.js'      // 写法一
import { b as f } from './1.js'  // 写法二
import { e } from './1.js'      // 写法二
```

从前面的例子可以看出，使用 `import` 时，用户需要知道所要加载的变量名或函数名，否则无法加载， `export default` 方式提供了模块默认输出的形式，给用户提供了方便：

```
// 3.js

// 写法一
export default function () {
  console.log('foo')
}

// 写法二
function foo() {
  console.log('foo')
}
export default foo

// 写法三
function foo(x, y) {
  console.log('foo')
}
export {add as default}

// 写法四
export default 42
```

引入时：

```
// 4.js

import bar from './3.js'                // 写法一
bar()
// 输出：foo

import { default as bar } from './3.js'    // 写法二
bar()
// 输出：foo
```

值得一提的是 `export`、`import` 都必须写在模块顶层，如果处于块级作用域内，就会报错，因为处于条件代码块之中，就没法做静态优化了，违背了 ES6 模块的设计初衷。

```
function foo() {
  export default 'bar'    // SyntaxError
}
foo()
```

}

26 MVC、MVP、MVVM

28 链模式