深入理解 JavaScript 系列（3）：全面解析 Module 模式

2011-12-30 09:52 by 汤姆大叔, 43583 阅读, 104 评论, 收藏, 编辑

简介

Module 模式是 JavaScript 编程中一个非常通用的模式，一般情况下，大家都知道基本用法，本文尝试着给大家更多该模式的高级使用方式。

首先我们来看看 Module 模式的基本特征：

1. 模块化，可重用
2. 封装了变量和 function，和全局的 namaspace 不接触，松耦合
3. 只暴露可用 public 的方法，其它私有方法全部隐藏

关于 Module 模式，最早是由 YUI 的成员 Eric Miraglia 在 4 年前提出了这个概念，我们将从一个简单的例子来解释一下基本的用法（如果你已经非常熟悉了，请忽略这一节）。

基本用法

先看一下最简单的一个实现，代码如下：

```
var Calculator = function (eq) {
    //这里可以声明私有成员

    var eqCtl = document.getElementById(eq);

    return {
        // 暴露公开的成员
        add: function (x, y) {
            var val = x + y;
            eqCtl.innerHTML = val;
        }
    };
};
```

我们可以通过如下的方式来调用：

```
var calculator = new Calculator('eq');
calculator.add(2, 2);
```

大家可能看到了，每次用的时候都要 new 一下，也就是说每个实例在内存里都是一份 copy，如果你不需要传参数或者没有一些特殊苛刻的要求的话，我们可以在最后一个}后面加上一个括号，来达到自执行的目的，这样该实例在内存中只会存在一份 copy，不过在展示他的优点之前，我们还是先来看看这个模式的基本使用方法吧。

匿名闭包

匿名闭包是让一切成为可能的基础，而这也是 JavaScript 最好的特性，我们来创建一个最简单的闭包函数，函数内部的代码一直存在于闭包内，在整个运行周期内，该闭包都保证了内部的代码处于私有状态。

```
(function () {
    // ... 所有的变量和 function 都在这里声明，并且作用域也只能在这个匿名闭包里
    // ...但是这里的代码依然可以访问外部全局的对象
}());
```

注意，匿名函数后面的括号，这是 JavaScript 语言所要求的，因为如果你不声明的话，JavaScript 解释器默认是声明一个 function 函数，有括号，就是创建一个函数表达式，也就是自执行，用的时候不用和上面那样在 new 了，当然你也可以这样来声明：

```
(function () {/* 内部代码 */})();
```

不过我们推荐使用第一种方式，关于函数自执行，我后面会有专门一篇文章进行详解，这里就不多说了。

引用全局变量

JavaScript 有一个特性叫做隐式全局变量，不管一个变量有没有用过，JavaScript 解释器反向遍历作用域链来查找整个变量的 var 声明，如果没有找到 var，解释器则假定该变量是全局变量，如果该变量用于了赋值操作的话，之前如果不存在的话，解释器则会自动创建它，这就是说在匿名闭包里使用或创建全局变量非常容易，不过比较困难的是，代码比较难管理，尤其是阅读代码的人看着很多区分哪些变量是全局的，哪些是局部的。

不过，好在在匿名函数里我们可以提供一个比较简单的替代方案，我们可以将全局变量当成一个参数传入到匿名函数然后使用，相比隐式全局变量，它又清晰又快，我们来看一个例子：

```
(function ($, YAHOO) {
    // 这里，我们的代码就可以使用全局的 jQuery 对象了，YAHOO 也是一样
} (jQuery, YAHOO));
```

现在很多类库里都有这种使用方式，比如 jQuery 源码。

不过，有时候可能不仅仅要使用全局变量，而是也想声明全局变量，如何做呢？我们可以通过匿名函数的返回值来返回这个全局变量，这也就是一个基本的 Module 模式，来看一个完整的代码：

```
var blogModule = (function () {
    var my = {}, privateName = "博客园";

    function privateAddTopic(data) {
```

```
        // 这里是内部处理代码
    }

    my.Name = privateName;
    my.AddTopic = function (data) {
        privateAddTopic(data);
    };

    return my;
} ());
```

上面的代码声明了一个全局变量 blogModule，并且带有 2 个可访问的属性：blogModule.AddTopic 和 blogModule.Name，除此之外，其它代码都在匿名函数的闭包里保持着私有状态。同时根据上面传入全局变量的例子，我们也可以很方便地传入其它的全局变量。

高级用法

上面的内容对大多数用户已经很足够了，但我们还可以基于此模式延伸出更强大，易于扩展的结构，让我们一个一个来看。

扩展

Module 模式的一个限制就是所有的代码都要写在一个文件，但是在一些大型项目里，将一个功能分离成多个文件是非常重要的，因为可以多人合作易于开发。再回头看看上面的全局参数导入例子，我们能否把 blogModule 自身传进去呢？答案是肯定的，我们先将 blogModule 传进去，添加一个函数属性，然后再返回就达到了我们所说的目的，上代码：

```
var blogModule = (function (my) {
    my.AddPhoto = function () {
        //添加内部代码
    };
    return my;
} (blogModule));
```

这段代码，看起来是不是有 C#里扩展方法的感觉？有点类似，但本质不一样哦。同时尽管 var 不是必须的，但为了确保一致，我们再次使用了它，代码执行以后，blogModule 下的 AddPhoto 就可以使用了，同时匿名函数内部的代码也依然保证了私密性和内部状态。

松耦合扩展

上面的代码尽管可以执行，但是必须先声明 blogModule，然后再执行上面的扩展代码，也就是说步骤不能乱，怎么解决这个问题呢？我们来回想一下，我们平时声明变量的都是都是这样的：

```
var cnblogs = cnblogs || {} ;
```

这是确保 cnblogs 对象，在存在的时候直接用，不存在的时候直接赋值为{}，我们来看看如何利用这个特性来实现 Module 模式的任意加载顺序：

```
var blogModule = (function (my) {

    // 添加一些功能

    return my;
} (blogModule || {}));
```

通过这样的代码，每个单独分离的文件都保证这个结构，那么我们就可以实现任意顺序的加载，所以，这个时候的 var 就是必须要声明的，因为不声明，其它文件读取不到哦。

紧耦合扩展

虽然松耦合扩展很牛叉了，但是可能也会存在一些限制，比如你没办法重写你的一些属性或者函数，也不能在初始化的时候就是用 Module 的属性。紧耦合扩展限制了加载顺序，但是提供了我们重载的机会，看如下例子：

```
var blogModule = (function (my) {
    var oldAddPhotoMethod = my.AddPhoto;

    my.AddPhoto = function () {
        // 重载方法，依然可通过 oldAddPhotoMethod 调用旧的方法
    };

    return my;
} (blogModule));
```

通过这种方式，我们达到了重载的目的，当然如果你想在继续在内部使用原有的属性，你可以调用 oldAddPhotoMethod 来用。

克隆与继承

```
var blogModule = (function (old) {
    var my = {},
        key;

    for (key in old) {
        if (old.hasOwnProperty(key)) {
            my[key] = old[key];
        }
    }

    var oldAddPhotoMethod = old.AddPhoto;
    my.AddPhoto = function () {
        // 克隆以后，进行了重写，当然也可以继续调用 oldAddPhotoMethod
```

```
    };

    return my;
} (blogModule));
```

这种方式灵活是灵活，但是也需要花费灵活的代价，其实该对象的属性对象或 function 根本没有被复制，只是对同一个对象多了一种引用而已，所以如果老对象去改变它，那克隆以后的对象所拥有的属性或 function 函数也会被改变，解决这个问题，我们就得是用递归，但递归对 function 函数的赋值也不好用，所以我们在递归的时候 eval 相应的 function。不管怎么样，我还是把这一个方式放在这个帖子里了，大家使用的时候注意一下就行了。

跨文件共享私有对象

通过上面的例子，我们知道，如果一个 module 分割到多个文件的话，每个文件需要保证一样的结构，也就是说每个文件匿名函数里的私有对象都不能交叉访问，那如果我们非要使用，那怎么办呢？ 我们先看一段代码：

```
var blogModule = (function (my) {
    var _private = my._private = my._private || {},

        _seal = my._seal = my._seal || function () {
            delete my._private;
            delete my._seal;
            delete my._unseal;

        },

        _unseal = my._unseal = my._unseal || function () {
            my._private = _private;
            my._seal = _seal;
            my._unseal = _unseal;
        };

    return my;
} (blogModule || {}));
```

任何文件都可以对他们的局部变量_private 设属性，并且设置对其他的文件也立即生效。一旦这个模块加载结束，应用会调用 blogModule._seal()"上锁"，这会阻止外部接入内部的_private。如果这个模块需要再次增生，应用的生命周期内，任何文件都可以调用_unseal() ”开锁”，然后再加载新文件。加载后再次调用 _seal()”上锁”。

子模块

最后一个也是最简单的使用方式，那就是创建子模块

```
blogModule.CommentSubModule = (function () {
    var my = {};
    // ...

    return my;
} ());
```

尽管非常简单，我还是把它放进来了，因为我想说明的是子模块也具有一般模块所有的高级使用方式，也就是说你可以对任意子模块再次使用上面的一些应用方法。

总结

上面的大部分方式都可以互相组合使用的，一般来说如果要设计系统，可能会用到松耦合扩展，私有状态和子模块这样的方式。另外，我这里没有提到性能问题，但我认为 Module 模式效率高，代码少，加载速度快。使用松耦合扩展允许并行加载，这更可以提升下载速度。不过初始化时间可能要慢一些，但是为了使用好的模式，这是值得的。

参考文章：

http://yuiblog.com/blog/2007/06/12/module-pattern/
http://www.adequatelygood.com/2010/3/JavaScript-Module-Pattern-In-Depth