**[JavaScript沙箱SandBox设计模式](http://www.cnblogs.com/jenry/archive/2010/12/21/1913047.html)**

沙箱模式常见于YUI3 core，它是一种采用同一构造器（Constructor）生成彼此独立且互不干扰（self-contained）的实例对象，而从避免污染全局对象的方法。

**命名空间**

JavaScript本身中没有提供命名空间机制，所以为了避免不同函数、对象以及变量名对全局空间的污染，通常的做法是为你的应用程序或者库创建一个唯一的全局对象，然后将所有方法与属性添加到这个对象上。

代码清单1 ： 传统命名空间模式

/\* BEFORE: 5 globals \*/  
// constructors  
function Parent() {}  
function Child() {}  
// a variable  
var some\_var = 1;  
// some objects  
var module1 = {};  
module1.data = {a: 1, b: 2};  
var module2 = {};  
/\* AFTER: 1 global \*/  
// global object  
var MYAPP = {};  
// constructors  
MYAPP.Parent = function() {};  
MYAPP.Child = function() {};  
// a variable  
MYAPP.some\_var = 1;  
// an object  
MYAPP.modules = {};  
// nested objects  
MYAPP.modules.module1 = {};  
MYAPP.modules.module1.data = {a: 1, b: 2};  
MYAPP.modules.module2 = {};

在这段代码中，你创建了一个全局对象MYAPP，并将其他所有对象、函数作为属性附加到MYAPP上。

通常这是一种较好的避免命名冲突的方法，它被应用在很多项目中，但这种方法有一些缺点。

* 需要给所有需要添加的函数、变量加上前缀。
* 因为只有一个全局对象，这意味着一部分代码可以肆意地修改全局对象而导致其余代码的被动更新。

**全局构造器**

你可以用一个全局构造器，而不是一个全局对象，我们给这个构造器起名为Sandbox(),你可以用这个构造器创建对象，你还可以为构造器传递一个回调函数作为参数，这个回调函数就是你存放代码的独立沙箱环境。

代码清单2：沙箱的使用

new Sandbox(function(box){  
 // your code here...  
});

让我们给沙箱添加点别的特性。

1. 创建沙箱时可以不使用'new'操作符。
2. Sandbox()构造器接受一些额外的配置参数，这些参数定义了生成对象所需模块的名称，我们希望代码更加模块化。

拥有了以上特性后，让我们看看怎样初始化一个对象。

代码清单3显示了你可以在不需要‘new’操作符的情况下，创建一个调用了'ajax'和'event'模块的对象。

代码清单3：以数组的形式传递模块名

Sandbox(['ajax', 'event'], function(box){  
 // console.log(box);  
});

代码清单4：以独立的参数形式传递模块名

Sandbox('ajax', 'dom', function(box){  
 // console.log(box);  
});

代码清单5显示了你可以把通配符'\*'作为参数传递给构造器，这意味着调用所有可用的模块，为了方便起见，如果没有向构造器传递任何模块名作为参数，构造器会把'\*'作为缺省参数传入。

代码清单5:调用所用可用模块

Sandbox('\*', function(box){  
 // console.log(box);  
});  
  
Sandbox(function(box){  
 // console.log(box);  
});

代码清单6显示你可以初始化沙箱对象多次，甚至你可以嵌套它们，而不用担心彼此间会产生任何冲突。

代码清单6:嵌套的沙箱实例

Sandbox('dom', 'event', function(box){  
 // work with dom and event  
 Sandbox('ajax', function(box) {  
 // another sandboxed "box" object  
 // this "box" is not the same as  
 // the "box" outside this function  
 //...  
 // done with Ajax  
 });  
 // no trace of Ajax module here  
});

从上面这些示例可以看出，使用沙箱模式，通过把所有代码逻辑包裹在一个回调函数中，你根据所需模块的不同生成不同的实例，而这些实例彼此互不干扰独立的工作着，从而保护了全局命名空间。

现在让我们看看怎样实现这个Sandbox()构造器。

**添加模块**

在实现主构造器之前，让我们看看如何向Sandbox()构造器中添加模块。

因为Sandbox()构造器函数也是对象，所以你可以给它添加一个名为’modules'的属性，这个属性将是一个包含一组键值对的对象，其中每对键值对中Key是需要注册的模块名，而Value则是该模块的入口函数，当构造器初始化时当前实例会作为第一个参数传递给入口函数，这样入口函数就能为该实例添加额外的属性与方法。

在代码清单7中，我们添加了'dom','event','ajax'模块。

代码清单7:注册模块

Sandbox.modules = {};  
  
Sandbox.modules.dom = function(box) {  
 box.getElement = function() {};  
 box.getStyle = function() {};  
 box.foo = "bar";  
};  
  
Sandbox.modules.event = function(box) {  
 // access to the Sandbox prototype if needed:  
 // box.constructor.prototype.m = "mmm";  
 box.attachEvent = function(){};  
 box.dettachEvent = function(){};  
};  
  
Sandbox.modules.ajax = function(box) {  
 box.makeRequest = function() {};  
 box.getResponse = function() {};  
};

**实现构造器**

代码清单8描述了实现构造器的方法，其中关键的几个要点：

1. 我们检查this是否为Sandbox的实例，若不是，证明Sandbox没有被new操作符调用，我们将以构造器方式重新调用它。
2. 你可以在构造器内部为this添加属性，同样你也可以为构造器的原型添加属性。
3. 模块名称会以数组、独立参数、通配符‘\*’等多种形式传递给构造器。
4. 请注意在这个例子中我们不需要从外部文件中加载模块，但在诸如YUI3中，你可以仅仅加载基础模块（通常被称作种子(seed)),而其他的所有模块则会从外部文件中加载。
5. 一旦我们知道了所需的模块，并初始化他们，这意味着调用了每个模块的入口函数。
6. 回调函数作为参数被最后传入构造器，它将使用最新生成的实例并在最后执行。

代码清单8:实现Sandbox构造器

<script>  
function Sandbox() {  
 // turning arguments into an array  
 var args = Array.prototype.slice.call(arguments),  
 // the last argument is the callback  
 callback = args.pop(),  
 // modules can be passed as an array or as individual parameters  
 modules = (args[0] && typeof args[0] === "string") ?  
 args : args[0],  
 i;  
 // make sure the function is called  
 // as a constructor  
 if (!(this instanceof Sandbox)) {  
 return new Sandbox(modules, callback);  
 }  
 // add properties to 'this' as needed:  
 this.a = 1;  
 this.b = 2;  
 // now add modules to the core 'this' object  
 // no modules or "\*" both mean "use all modules"  
 if (!modules || modules === '\*') {  
 modules = [];  
 for (i in Sandbox.modules) {  
 if (Sandbox.modules.hasOwnProperty(i)) {  
 modules.push(i);  
 }  
 }  
 }  
 // init the required modules  
 for (i = 0; i < modules.length; i++) {  
 Sandbox.modules[modules[i]](this);  
 }  
 // call the callback  
 callback(this);  
}  
// any prototype properties as needed  
Sandbox.prototype = {  
 name: "My Application",  
 version: "1.0",  
 getName: function() {  
 return this.name;  
 }  
};  
</script>