**Drawbacks of the namespacing pattern**

• Reliance on a single global variable to be the application’s global. In the namespacing pattern, there is no way to have two versions of the same application or library run on the same page, because they both need the same global symbol name, for example, MYAPP.

• Long,  dotted  names  to  type  and  resolve  at  runtime,  for  example, MYAPP.utilities.array.

**A Global Constructor**

Using the sandbox will look like this:

new Sandbox(function (box) {

// your code here...

});

The  object  box will  be  like  MYAPP in  the  namespacing  example—it  will  have  all  the library functionality you need to make your code work.

Sandbox(['ajax', 'event'], function (box) {

// console.log(box);

});

module names are passed as individual argument

Sandbox('ajax', 'dom', function (box) {

// console.log(box);

});

when no modules are passed, the sandbox will assume '\*'.

[复制代码](javascript:void(0);)

Sandbox('\*', function (box) {

// console.log(box);

});

Sandbox(function (box) {

// console.log(box);

});

[复制代码](javascript:void(0);)

protect the global namespace by having your code wrapped into callback functions.

[复制代码](javascript:void(0);)

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

});

[复制代码](javascript:void(0);)

Adding Modules

[复制代码](javascript:void(0);)

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 () {};

};

[复制代码](javascript:void(0);)

Implementing the Constructor

[复制代码](javascript:void(0);)

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);

// Sandbox.modules[i](this);

}

}

}

// initialize the required modules (Kaibo: The code below in yellowcan be omitted by the code in yellow above to avoid the 2nd loop)

for (i = 0; i < modules.length; i += 1) {

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;

}

};

[复制代码](javascript:void(0);)

**References*:***

*JavaScript Patterns -*by Stoyan Stefanov (O`Reilly)