All object members are public in JavaScript.

[复制代码](javascript:void(0);)

var myobj = {

myprop : 1,

getProp : function() {

return this.myprop;

}

};

console.log(myobj.myprop);

// `myprop` is publicly accessible

console.log(myobj.getProp());

// getProp() is public too

[复制代码](javascript:void(0);)

The same is true when you use constructor functions to create objects.

[复制代码](javascript:void(0);)

// all members are still public:

function Gadget() {

this.name = 'iPod';

this.stretch = function() {

return 'iPad';

};

}

var toy = new Gadget();

console.log(toy.name);

// `name` is public

console.log(toy.stretch());

// stretch() is public

[复制代码](javascript:void(0);)

**Private Members**

Implement private members using a closure.

[复制代码](javascript:void(0);)

function Gadget() {

// private member

**var name = 'iPod';**

// public function

**this.getName = function() {**

**return name;**

**};**

}

var toy = new Gadget();

// `name` is undefined, it's private

console.log(toy.name);

// undefined

// public method has access to `name`

console.log(toy.getName());

// "iPod"

[复制代码](javascript:void(0);)

**Privileged Methods**

it’s just a name given to the public methods that have access to the private members (and hence have more privileges).

In the previous example,  getName() is a privileged method because it has “special” access to the private property name.

**Privacy Failures**

• When you’re directly returning a private variable from a privileged method and this variable happens to be an object or array, then outside code can modify the private variable because it’s passed by reference.

[复制代码](javascript:void(0);)

function Gadget() {

// private member

var specs = {

screen\_width : 320,

screen\_height : 480,

color : "white"

};

// public function

this.getSpecs = function() {

return specs;

};

}

var toy = new Gadget(), specs = toy.getSpecs();

specs.color = "black";

specs.price = "free";

console.dir(toy.getSpecs());

[复制代码](javascript:void(0);)

/\*

|  |  |
| --- | --- |
| color | "black" |
| price | "free" |
| screen\_height | 480 |
| screen\_width | 320 |

\*/

**Solutions**

1. Principle of Least Authority (POLA):

Return a new object containing only some of the data that could be interesting to the consumer of the object.

1. Another  approach,  when  you  need  to  pass  all  the  data,  is  to  create  a  copy  of  the specs object, using a general-purpose object-cloning function.

**Object Literal and Privacy**

[复制代码](javascript:void(0);)

var myobj;

// this will be the object

( function() {

// private members

var name = "my, oh my";

// implement the public part

// note -- no `var`

myobj = {

// privileged method

getName : function() {

return name;

}

};

}());

var myobj = ( function() {

// private members

var name = "my, oh my";

// implement the public part

return {

getName : function() {

return name;

}

};

}());

myobj.getName();

// "my, oh my"

[复制代码](javascript:void(0);)

**Prototypes and Privacy**

One drawback of the private members when used with constructors is that they are recreated every time the constructor is invoked to create a new object. To solve this you can add common properties and methods to the prototype property of the constructor.

[复制代码](javascript:void(0);)

function Gadget() {

// private member

var name = 'iPod';

// public function

this.getName = function() {

return name;

};

}

Gadget.prototype = ( function() {

// private member

var browser = "Mobile Webkit";

// public prototype members

return {

getBrowser : function() {

return browser;

}

};

}());

var toy = new Gadget();

console.log(toy.getName());

// privileged "own" method

console.log(toy.getBrowser());

// privileged prototype method

[复制代码](javascript:void(0);)

**Revealing Private Functions As Public Methods**

[复制代码](javascript:void(0);)

var myarray;

(function () {

var astr = "[object Array]",

toString = Object.prototype.toString;

// private method

function isArray(a) {

return toString.call(a) === astr;

})

// private method

function indexOf(haystack, needle) {

var i = 0,

max = haystack.length;

for (; i < max; i += 1) {

if (haystack[i] === needle) {

return i;

}

}

return−1;

}

myarray = {

// public methods

isArray: isArray,

indexOf: indexOf,

inArray: indexOf

};

}());

myarray.isArray([1, 2]); // true

myarray.isArray({

0: 1

}); // false

myarray.indexOf(["a", "b", "z"], "z"); // 2

myarray.inArray(["a", "b", "z"], "z"); // 2

[复制代码](javascript:void(0);)

Now if something unexpected happens, for example, to the public indexOf(), the private indexOf() is still safe and therefore inArray()will continue to work:

myarray.indexOf = null;

myarray.inArray(["a", "b", "z"], "z"); // 2

**References*:***

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