**Dealing with == and ===**

**false == 0** or **"" == 0** return true.

always use the **===** and **!==**

operators that check both the values and the type of the expressions you compare:

[复制代码](javascript:void(0);)

var zero = 0;

if (zero === false) {

// not executing because zero is 0, not false

}

// antipattern

if (zero == false) {

// this block is executed...

}

[复制代码](javascript:void(0);)

**Avoiding eval()**

[复制代码](javascript:void(0);)

// antipattern

var property = "name";

alert(eval("obj." + property));

// preferred

var property = "name";

alert(obj[property]);

[复制代码](javascript:void(0);)

**Security implications (e.g. JSON response from an Ajax request)**

1. For browsers that don't support **JSON.parse()**natively, you can use a library from JSON.org.

2. passing strings to setInterval(), setTimeout(), and the Function() constructor is, for the most part, similar to using eval()and therefore should be avoided.

[复制代码](javascript:void(0);)

// antipatterns

setTimeout("myFunc()", 1000);

setTimeout("myFunc(1, 2, 3)", 1000);

// preferred

setTimeout(myFunc, 1000);

setTimeout(function () {

myFunc(1, 2, 3);

}, 1000);

[复制代码](javascript:void(0);)

3. Using the **new Function()**constructor is similar to eval() and should be approached with care.

* 1. If you absolutely must use eval(), you can consider using **new Function()** instead.   
     Because the code evaluated in new Function() will be running in a local function scope, so any variables defined with **var**in the code being evaluated **will not** become globals automatically.
  2. Or wrap the eval() call into an immediate function.

[复制代码](javascript:void(0);)

console.log(typeof un); // "undefined"

console.log(typeof deux); // "undefined"

console.log(typeof trois); // "undefined"

var jsstring = "var un = 1; console.log(un);";

eval(jsstring); // logs "1"

jsstring = "var deux = 2; console.log(deux);";

new Function(jsstring)(); // logs "2"

jsstring = "var trois = 3; console.log(trois);";

(function () {

eval(jsstring);

}()); // logs "3"

console.log(typeof un); // "number"

console.log(typeof deux); // "undefined"

console.log(typeof trois); // "undefined"

[复制代码](javascript:void(0);)

* 1. No matter where you execute **Function**, it sees only the global scope. So it **can do less local variable pollution**.

[复制代码](javascript:void(0);)

(function () {

var local = 1;

eval("local = 3; console.log(local)"); // logs 3

console.log(local); // logs 3

}());

(function () {

var local = 1;

Function("console.log(typeof local);")(); // logs undefined

}());

[复制代码](javascript:void(0);)