JavaScript奇技44招小技巧

1、首次为变量赋值时务必使用var关键字

变量没有声明而直接赋值得话，默认会作为一个新的全局变量，要尽量避免使用全局变量。

2、使用===取代==

==和!=操作符会在需要的情况下自动转换数据类型。但===和!==不会，它们会同时比较值和数据类型，这也使得它们要比==和!=快。

```
[10] === 10    // is false
[10]  == 10    // is true
'10' == 10     // is true
'10' === 10    // is false
 []   == 0     // is true
 [] ===  0     // is false
 '' == false   // is true but true == "a" is false
 '' === false  // is false
```

3、underfined、null、0、false、NaN、空字符串的逻辑结果均为false

4、行尾使用分号

实践中最好还是使用分号，忘了写也没事，大部分情况下JavaScript解释器都会自动添加。对于为何要使用分号，可参考文章JavaScript中关于分号的真相（https://davidwalsh.name/javascript-semicolons）。

5、使用对象构造器

```
function Person(firstName, lastName){
    this.firstName =  firstName;
    this.lastName = lastName;
}
var Saad = new Person("Saad", "Mousliki");
```

6、小心使用typeof、instanceof和contructor

- typeof：JavaScript一元操作符，用于以字符串的形式返回变量的原始类型，注意，typeof null也会返回object，大多数的对象类型（数组Array、时间Date等）也会返回object
- contructor：内部原型属性，可以通过代码重写
- instanceof：JavaScript操作符，会在原型链中的构造器中搜索，找到则返回true，否则返回false

```
var arr = ["a", "b", "c"];
typeof arr;   // 返回 "object"
arr instanceof Array // true
arr.constructor();  //[]
```

7、使用自调用函数

函数在创建之后直接自动执行，通常称之为自调用匿名函数（Self-Invoked Anonymous Function）或直接调用函数表达式（Immediately Invoked Function Expression ）。格式如下：

```
(function(){
    // 置于此处的代码将自动执行
})();
(function(a,b){
    var result = a+b;
    return result;
})(10,20)
```

8、从数组中随机获取成员

```
var items = [12, 548 , 'a' , 2 , 5478 , 'foo' , 8852, , 'Doe' , 2145 , 119];
var  randomItem = items[Math.floor(Math.random() * items.length)];
```

9、获取指定范围内的随机数

这个功能在生成测试用的假数据时特别有数，比如介与指定范围内的工资数。

```
var x = Math.floor(Math.random() * (max - min + 1)) + min;
```

10、生成从0到指定值的数字数组

```
var numbersArray = [] , max = 100;
for( var i=1; numbersArray.push(i++) < max;);  // numbers = [1,2,3 ... 100]
```

11、生成随机的字母数字字符串

```
function generateRandomAlphaNum(len) {
    var rdmString = "";
    for( ; rdmString.length < len; rdmString  += Math.random().toString(36).substr(2));
    return  rdmString.substr(0, len);
}
```

12、打乱数字数组的顺序

```
var numbers = [5, 458 , 120 , -215 , 228 , 400 , 122205, -85411];
numbers = numbers.sort(function(){ return Math.random() - 0.5});
/* numbers 数组将类似于 [120, 5, 228, -215, 400, 458, -85411, 122205]  */
```

这里使用了JavaScript内置的数组排序函数，更好的办法是用专门的代码来实现（如Fisher-Yates算法），可以参见StackOverFlow上的这个讨论。

13、字符串去空格

```
String.prototype.trim = function(){return this.replace(/^\s+|\s+$/g, "");};
```

新的JavaScript引擎已经有了trim()的原生实现。

14、数组之间追加

```
var array1 = [12 , "foo" , {name "Joe"} , -2458];
var array2 = ["Doe" , 555 , 100];
Array.prototype.push.apply(array1, array2);
/* array1 值为  [12 , "foo" , {name "Joe"} , -2458 , "Doe" , 555 , 100] */
```

15、对象转换为数组

```
var argArray = Array.prototype.slice.call(arguments);
```

16、验证是否是数字

```
function isNumber(n){
    return !isNaN(parseFloat(n)) && isFinite(n);
}
```

17、验证是否是数组

```
function isArray(obj){
    return Object.prototype.toString.call(obj) === '[object Array]' ;
}
```

但如果toString()方法被重写过得话，就行不通了。也可以使用下面的方法：

```
Array.isArray(obj); // its a new Array method
```

如果在浏览器中没有使用frame，还可以用instanceof，但如果上下文太复杂，也有可能出错。

```
var myFrame = document.createElement('iframe');
document.body.appendChild(myFrame);
var myArray = window.frames[window.frames.length-1].Array;
var arr = new myArray(a,b,10); // [a,b,10]
// myArray 的构造器已经丢失，instanceof 的结果将不正常
```

```
// 构造器是不能跨 frame 共享的
arr instanceof Array; // false
```

18、获取数组中的最大值和最小值

```
var  numbers = [5, 458 , 120 , -215 , 228 , 400 , 122205, -85411];
var maxInNumbers = Math.max.apply(Math, numbers);
var minInNumbers = Math.min.apply(Math, numbers);
```

19、清空数组

```
var myArray = [12 , 222 , 1000 ];
myArray.length = 0; // myArray will be equal to [].
```

20、不要直接从数组中delete或remove元素

如果对数组元素直接使用delete，其实并没有删除，只是将元素置为了undefined。数组元素删除应使用splice。

切忌：

```
var items = [12, 548 ,'a' , 2 , 5478 , 'foo' , 8852, , 'Doe' ,2154 , 119 ];
items.length; // return 11
delete items[3]; // return true
items.length; // return 11
/* items 结果为 [12, 548, "a", undefined × 1, 5478, "foo", 8852, undefined × 1, "Doe", 2154, 119]
*/
```

而应：

```
var items = [12, 548 ,'a' , 2 , 5478 , 'foo' , 8852, , 'Doe' ,2154 , 119 ];
items.length; // return 11
items.splice(3,1) ;
items.length; // return 10
/* items 结果为 [12, 548, "a", 5478, "foo", 8852, undefined × 1, "Doe", 2154, 119]
```

删除对象的属性时可以使用delete。

21、使用length属性截断数组

前面的例子中用length属性清空数组，同样还可用它来截断数组：

```
var myArray = [12 , 222 , 1000 , 124 , 98 , 10 ];
myArray.length = 4; // myArray will be equal to [12 , 222 , 1000 , 124].
```

与此同时，如果把length属性变大，数组的长度值变会增加，会使用undefined来作为新的元素填充。length是一个可写的属性。

```
myArray.length = 10; // the new array length is 10
myArray[myArray.length - 1] ; // undefined
```

22、在条件中使用逻辑与或

```
var foo = 10;
foo == 10 && doSomething(); // is the same thing as if (foo == 10) doSomething();
foo == 5 || doSomething(); // is the same thing as if (foo != 5) doSomething();
```

逻辑或还可用来设置默认值，比如函数参数的默认值。

```
function doSomething(arg1){
    arg1 = arg1 || 10; // arg1 will have 10 as a default value if it’s not already set
}
```

23、使得map()函数方法对数据循环

```
var squares = [1,2,3,4].map(function (val) {
    return val * val;
});
// squares will be equal to [1, 4, 9, 16]
```

24、保留指定小数位数

```
var num =2.443242342;
num = num.toFixed(4);  // num will be equal to 2.4432
```

注意，toFixed()返回的是字符串，不是数字。

25、浮点计算的问题

```
0.1 + 0.2 === 0.3 // is false
9007199254740992 + 1 // is equal to 9007199254740992
9007199254740992 + 2 // is equal to 9007199254740994
```

为什么呢？因为0.1+0.2等于0.30000000000000004。JavaScript的数字都遵循IEEE 754标准构建，在内部都是64位浮点小数表示，具体可以参见JavaScript中的数字是如何编码的.

可以通过使用toFixed()和toPrecision()来解决这个问题。

26、通过for-in循环检查对象的属性

下面这样的用法，可以防止迭代的时候进入到对象的原型属性中。

```
for (var name in object) {
    if (object.hasOwnProperty(name)) {
        // do something with name
    }
}
```

27、逗号操作符

```
var a = 0;
var b = ( a++, 99 );
console.log(a);  // a will be equal to 1
console.log(b);  // b is equal to 99
```

28、临时存储用于计算和查询的变量

在jQuery选择器中，可以临时存储整个DOM元素。

```
var navright = document.querySelector('#right');
var navleft = document.querySelector('#left');
var navup = document.querySelector('#up');
var navdown = document.querySelector('#down');
```

29、提前检查传入isFinite()的参数

```
isFinite(0/0) ; // false
isFinite("foo"); // false
isFinite("10"); // true
isFinite(10);   // true
isFinite(undefined);  // false
isFinite();   // false
isFinite(null);  // true，这点当特别注意
```

30、避免在数组中使用负数做索引

```
var numbersArray = [1,2,3,4,5];
var from = numbersArray.indexOf("foo") ;  // from is equal to -1
numbersArray.splice(from,2);    // will return [5]
```

注意传给splice的索引参数不要是负数，当是负数时，会从数组结尾处删除元素。

31、用JSON来序列化与反序列化

```
var person = {name :'Saad', age : 26, department : {ID : 15, name : "R&D"} };
var stringFromPerson = JSON.stringify(person);
/* stringFromPerson 结果为 "{"name":"Saad","age":26,"department":{"ID":15,"name":"R&D"}}"   */
var personFromString = JSON.parse(stringFromPerson);
```

```
/* personFromString 的值与 person 对象相同  */
```

32、不要使用eval()或者函数构造器

eval()和函数构造器（Function consturctor）的开销较大，每次调用，JavaScript引擎都要将源代码转换为可执行的代码。

```
var func1 = new Function(functionCode);
var func2 = eval(functionCode);
```

33、避免使用with()

使用with()可以把变量加入到全局作用域中，因此，如果有其它的同名变量，一来容易混淆，二来值也会被覆盖。

34、不要对数组使用for-in

避免：

```
var sum = 0;
for (var i in arrayNumbers) {
    sum += arrayNumbers[i];
}
```

而是：

```
var sum = 0;
for (var i = 0, len = arrayNumbers.length; i < len; i++) {
    sum += arrayNumbers[i];
}
```

另外一个好处是，i和len两个变量是在for循环的第一个声明中，二者只会初始化一次，这要比下面这种写法快：

```
for (var i = 0; i < arrayNumbers.length; i++)
```

35、传给setInterval()和setTimeout()时使用函数而不是字符串

如果传给setTimeout()和setInterval()一个字符串，他们将会用类似于eval方式进行转换，这肯定会要慢些，因此不要使用：

```
setInterval('doSomethingPeriodically()', 1000);
setTimeout('doSomethingAfterFiveSeconds()', 5000);
```

而是用：

```
setInterval(doSomethingPeriodically, 1000);
setTimeout(doSomethingAfterFiveSeconds, 5000);
```

36、使用switch/case代替一大叠的if/else

当判断有超过两个分支的时候使用switch/case要更快一些，而且也更优雅，更利于代码的组织，当然，如果有超过10个分支，就不要使用switch/case了。

37、在switch/case中使用数字区间

其实，switch/case中的case条件，还可以这样写：

```
function getCategory(age) {
    var category = "";
    switch (true) {
        case isNaN(age):
            category = "not an age";
            break;
        case (age >= 50):
            category = "Old";
            break;
        case (age <= 20):
            category = "Baby";
            break;
        default:
            category = "Young";
            break;
```

```
    };
    return category;
}
getCategory(5);  // 将返回 "Baby"
```

38、使用对象作为对象的原型

下面这样，便可以给定对象作为参数，来创建以此为原型的新对象：

```
function clone(object) {
    function OneShotConstructor(){};
    OneShotConstructor.prototype = object;
    return new OneShotConstructor();
}
clone(Array).prototype ;  // []
```

39、HTML字段转换函数

```
function escapeHTML(text) {
    var replacements= {"<": "&lt;", ">": "&gt;","&": "&amp;", "\"": "&quot;"};
    return text.replace(/[<>&"]/g, function(character) {
        return replacements[character];
    });
}
```

40、不要在循环内部使用try-catch-finally

try-catch-finally中catch部分在执行时会将异常赋给一个变量，这个变量会被构建成一个运行时作用域内的新的变量。

切忌：

```
var object = ['foo', 'bar'], i;
for (i = 0, len = object.length; i <len; i++) {
    try {
        // do something that throws an exception
    }
    catch (e) {
        // handle exception
    }
}
```

而应该：

```
var object = ['foo', 'bar'], i;
try {
    for (i = 0, len = object.length; i <len; i++) {
        // do something that throws an exception
    }
}
catch (e) {
    // handle exception
}
```

41、使用XMLHttpRequests时注意设置超时

XMLHttpRequests在执行时，当长时间没有响应（如出现网络问题等）时，应该中止掉连接，可以通过setTimeout()来完成这个工作：

```
var xhr = new XMLHttpRequest ();
xhr.onreadystatechange = function () {
    if (this.readyState == 4) {
        clearTimeout(timeout);
        // do something with response data
    }
```

```
}
var timeout = setTimeout( function () {
    xhr.abort(); // call error callback
}, 60*1000 /* timeout after a minute */ );
xhr.open('GET', url, true);
xhr.send();
```

同时需要注意的是，不要同时发起多个XMLHttpRequests请求。

42、处理WebSocket的超时

通常情况下，WebSocket连接创建后，如果30秒内没有任何活动，服务器端会对连接进行超时处理，防火墙也可以对单位周期没有活动的连接进行超时处理。

为了防止这种情况的发生，可以每隔一定时间，往服务器发送一条空的消息。可以通过下面这两个函数来实现这个需求，一个用于使连接保持活动状态，另一个专门用于结束这个状态。

```
var timerID = 0;
function keepAlive() {
    var timeout = 15000;
    if (webSocket.readyState == webSocket.OPEN) {
        webSocket.send('');
    }
    timerId = setTimeout(keepAlive, timeout);
}
function cancelKeepAlive() {
    if (timerId) {
        cancelTimeout(timerId);
    }
}
```

keepAlive()函数可以放在WebSocket连接的onOpen()方法的最后面，cancelKeepAlive()放在onClose()方法的最末尾。

43、时间注意原始操作符比函数调用快，使用VanillaJS

比如，一般不要这样：

```
var min = Math.min(a,b);
A.push(v);
```

可以这样来代替：

```
var min = a < b ? a : b;
A[A.length] = v;
```

44、开发时注意代码结构，上线前检查并压缩JavaScript代码

可以使用JSLint或JSMin等工具来检查并压缩代码。