--

[WWW 1](#_Toc30336)

[JavaScript的技巧、秘诀和最佳实践 2](#_Toc16402)

[函数各种未知表现---2014阿里巴巴校招在线笔试 4](#_Toc23321)

[常用JavaScript代码 5](#_Toc28924)

[字符串中出现最多的字符和个数 5](#_Toc20448)

[console.table 5](#_Toc4414)

[取URL中的参数 5](#_Toc13299)

[对URL进行编码 5](#_Toc21658)

[非空字符串判断 5](#_Toc24767)

[endsWith 5](#_Toc6323)

[克隆对象 6](#_Toc10145)

[base64编码 6](#_Toc12970)

[setTimeout和setInterval的使用 6](#_Toc5506)

[setTimeout/setInterval执行时机 6](#_Toc7887)

[实现无限循环 6](#_Toc29786)

[函数指针的使用 6](#_Toc19094)

[案例1 6](#_Toc443)

[案例2 7](#_Toc13166)

[注意点 7](#_Toc19162)

[typeof if == 7](#_Toc10552)

[类型判断 7](#_Toc25003)

[== === 7](#_Toc14687)

[if(){} 8](#_Toc31752)

[非空判断 8](#_Toc12270)

[非空判断 8](#_Toc21766)

[undefined null NaN 8](#_Toc17941)

[undefined null 8](#_Toc10711)

[NaN 8](#_Toc4593)

[参数：undefined-not defined 8](#_Toc14069)

[ajax 8](#_Toc10655)

[get|post 8](#_Toc32618)

[http状态码 8](#_Toc15330)

[注意点 9](#_Toc14177)

[parseInt 9](#_Toc13019)

[预解析 9](#_Toc13435)

[块级作用域 9](#_Toc22950)

[闭包 9](#_Toc6511)

[在for循环中的不正确函数调用 10](#_Toc998)

[对于this关键词的不正确引用 10](#_Toc24573)

[生命周期---块级作用域 10](#_Toc28276)

[内存泄露 10](#_Toc15175)

[避免低效的DOM操作 10](#_Toc2774)

[原型继承问题 10](#_Toc24064)

[为实例方法创建错误的指引 10](#_Toc13100)

[{} or [] 11](#_Toc19226)

[strict mode 11](#_Toc29356)

[prototypes 11](#_Toc19670)

理论

WWW

|  |  |
| --- | --- |
|  | <http://www.xuanfengge.com/front-end-analysis-of-collection-of-interview-questions-and-answers.html#more-5480>    <http://www.xuanfengge.com/category/code/mianshi> |
|  | <http://caibaojian.com/288.html#t7>    [javascript常用代码大全](http://caibaojian.com/288.html)   * [jquery选中radio](http://caibaojian.com/288.html" \l "t1) * [jquery获取高度](http://caibaojian.com/288.html" \l "t2) * [jquery插件中的参数相加](http://caibaojian.com/288.html" \l "t3) * [js除数取整](http://caibaojian.com/288.html" \l "t4) * [js全选](http://caibaojian.com/288.html" \l "t5) * [js取整数](http://caibaojian.com/288.html" \l "t6) * [jQuery获取Select选择的Text和Value:](http://caibaojian.com/288.html" \l "t7) * [jquery判断两次点击对象是否是同一对象](http://caibaojian.com/288.html" \l "t8) * [js处理字符串的常见操作](http://caibaojian.com/288.html" \l "t9)   [JavaScript一些简写代码](http://caibaojian.com/javascript-simple-code.html)   * [1. 判断是否为空(Null, Undefined,”)](http://caibaojian.com/javascript-simple-code.html" \l "t1) * [2. 对象](http://caibaojian.com/javascript-simple-code.html" \l "t2) * [3. 对象数组](http://caibaojian.com/javascript-simple-code.html" \l "t3) * [4. 关联数组](http://caibaojian.com/javascript-simple-code.html" \l "t4) * [5. 赋值运算符](http://caibaojian.com/javascript-simple-code.html" \l "t5) * [6. Regexp 对象](http://caibaojian.com/javascript-simple-code.html" \l "t6) * [7. If语句](http://caibaojian.com/javascript-simple-code.html" \l "t7) * [更多补充：](http://caibaojian.com/javascript-simple-code.html" \l "t8) * [8. JavaScript For 循环](http://caibaojian.com/javascript-simple-code.html" \l "t9) * [9. 函数变量参数](http://caibaojian.com/javascript-simple-code.html" \l "t10) * [10、对象字面量](http://caibaojian.com/javascript-simple-code.html" \l "t11) |
|  | **[HTML渲染过程详解](http://www.cnblogs.com/dojo-lzz/p/3983335.html)**  http://www.cnblogs.com/dojo-lzz/p/3983335.html |
|  | http://weibo.com/1196343093/Bhj510t50?type=comment  阿里无线前端招聘  2014年8月8日 10:59 阅读 3678  各位亲，最近又要大量招人了，先说下规则：对自己工作经历和学历特别有信心的同学欢迎直接简历投 ali-mfe-hire@list.alibaba-inc.com ， 觉得怀才不遇的同学，可以以写blog的方式证明自己，以下9个题目任选一个或者几个，写一篇文章，微博上@ 我。  谈谈你对CSS布局的理解  讲讲输入完网址按下回车，到看到网页这个过程中发生了什么。  谈谈你对Web前端组件化的理解，Web Component会带来怎样的影响  谈谈你对前端资源下载性能优化的经验和思考  现在有很多的MV\*框架，你对它们有什么看法  iOS体验好在哪里，Web能赶上么？  网页游戏怎么做？  Hybrid技术应当如何应用？  你最爱的前端框架是什么，为什么？  http://www.cnblogs.com/yexiaochai/p/3899974.html |

|  |  |
| --- | --- |
| JavaScript的技巧、秘诀和最佳实践 | |
|  | JavaScript是一个绝冠全球的编程语言，可用于Web开发、移动应用开发（PhoneGap、Appcelerator）、服务器端开发（Node.js和Wakanda）等等。JavaScript还是很多新手踏入编程世界的第一个语言。既可以用来显示浏览器中的简单提示框，也可以通过nodebot或nodruino来控制机器人。能够编写结构清晰、性能高效的JavaScript代码的开发人员，现如今已成了招聘市场最受追捧的人。  在这篇文章里，我将分享一些JavaScript的技巧、秘诀和最佳实践，除了少数几个外，不管是浏览器的JavaScript引擎，还是服务器端JavaScript解释器，均适用。  本文中的示例代码，通过了在Google Chrome 30最新版（V8 3.20.17.15）上的测试。  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。   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。  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  }  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  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++)  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、  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  别忘了在写代码时使用一个代码美化工具。使用JSLint(一个语法检查工具)并且在上线前压缩代码（比如使用JSMin）。  注：现在代码压缩一般推荐 UglifyJS (https://github.com/mishoo/UglifyJS2)  45、JavaScript博大精深，这里有些不错的学习资源  Code Academy资源：http://www.codecademy.com/tracks/javascript Marjin Haverbekex编写的Eloquent JavaScript：http://eloquentjavascript.net/ John Resig编写的Advanced JavaScript：http://ejohn.org/apps/learn/ |
|  | 四、jQuery中获取设置checkbox选中状态  　　由于在jQuery1.6以后.attr("checked")的返回结果是 checked，所以一般用下面两种方法获取选中状态：      $("#checkboxID").is(":checked");      //jQuery 1.6 +      $("#checkboxID").prop("checked");      　　选中checkbox：      //jQuery 1.6+      $("#checkboxID").prop("checked", true);      $("#checkboxID").prop("checked", false);      //jQuery 1.5 and below      $('#checkboxID').attr('checked','checked')      $('#checkboxID').removeAttr('checked')  五、jQuery中判断一个元素是否存在      if ($(selector).length)  七、jQuery中event.preventDefault() 与 return false 的区别      //Demo1 event.preventDefault()      $('a').click(function (e) {          // custom handling here          e.preventDefault();      });      //Demo2 return false      $('a').click(function () {          // custom handling here          return false;      };      　　jQuery中return false相当于同时调用e.preventDefault 和 e.stopPropagation。 　　要注意的是，在原生js中，return false仅仅相当于调用了e.preventDefault。  八、  二十、jQuery中each跟map的区别 　　each跟map都可以用来遍历Array或Object，区别是each不改变原来的Array或Object，map是操作给定的Array或Object返回一个新Array或Object。Demo:      var items = [1,2,3,4];      $.each(items, function() {        alert('this is ' + this);//alert 1,2,3,4      });      var newItems = $.map(items, function(i) {        return i + 1;      });      // newItems is [2,3,4,5]  　　map会占用更多的内存，所以如果只是遍历建议用each。 九、用JavaScript添加和删除class      //Add Class      document.getElementById("MyElement").className += " MyClass";  //Remove Class  document.getElementById("MyElement").className = document.getElementById("MyElement").className.replace(/(?:^|\s)MyClass(?!\S)/,'');  十、在jQuery中取消一个ajax请求      var xhr = $.ajax({          type: "POST",          url: "test.php",          data: "name=test",          success: function(msg){             alert( msg );          }      });      //取消请求      xhr.abort()  　　要注意的是，在ajax请求未响应之前可以用xhr.abort()取消，但如果请求已经到达了服务器端，这样做的结果仅仅是让浏览器不再监听这个请求的响应，但服务器端仍然会进行处理。 |
|  | 三、为什么parseInt(1/0,19)的结果为18 　　1/0的结果是Infinity，所以parseInt(1/0,19)等同于parseInt("Infinity",19)，而在19进制中：i表示18，所以parseInt(1/0,19)的结果为18。  第二个参数的区间[2-36]  十二、JavaScript中16进制与10进制相互转换      var sHex=(255).toString(16);//ff      var iNum=parseInt("ff",16);//255  十四、JavaScript中!!操作符是什么      console.log(!!10);//true      console.log(!!0);//false      console.log(!!"abc");//true  console.log(!!"");//false 　　简单地说就是把右侧的值转为布尔值  十三、JavaScript多行字符串  　　如何在JavaScript中方便地写一个多行字符串呢，有三种方案，你自己选吧：  //one var testHtml="a"+  "b"+  "c";  //two var testHtml2="a\     b\     c";  //three var testHtml3=["a",  "b",  "c"].join("");  十七、JavaScript字符与ASCII码间的转换      console.log("\n".charCodeAt(0));//10      console.log(String.fromCharCode(65));//A  十八、JavaScript中浮点数的相等判断不能用 ==      console.log(0.1+0.2 == 0.3);//false      console.log(Math.abs(0.1+0.2 - 0.3) < 0.000001);//true  　　如上所示，浮点数相等判断要用差的绝对值小于某一个数来判断。至于原因可以参考这里：[http://docs.oracle.com/cd/E19957-01/806-3568/ncg\_goldberg.html](http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html" \t "http://www.w3cfuns.com/_blank)  二十二、通过原型继承创建一个新对象      function inherit(p){          if(!p){              throw TypeError("p is not an object or null");          }          if(Object.create){              return Object.create(p);          }          var t=typeof p;          if(t !== "object" && t !== "function"){              throw TypeError("p is not an object or null");          }          function f(){};          f.prototype=p;          return new f();  }  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) |
|  |  |
|  |  |

函数各种未知表现---2014阿里巴巴校招在线笔试

|  |  |
| --- | --- |
| (**function** () {  **return typeof** arguments; })(); | arguments是实参集合，有数组长度，没有数组方法，typeof会返回对象。  于是乎，上面的代码会返回字符串“object”。 |
| **var** f = **function** g() {  **return** 23; }; **typeof** g(); //'number'  typeof f; //'string' | 这里面的表达式是有名函数表达式，不规范，g()只能在函数内部访问。  typeof就会Error。 |
| (**function** (x) {  **delete** x;  **return** x; })(1); | delete不能删除变量和参数，只能删除对象下面的属性,因此return还是会返回1。 |
| **var** y = 1, x = y = **typeof** x; x; | 后面表达式是从右向左执行的，typeof返回的都是字符串类型，最先运算的是typeof x，因为他先是'undefined'所以赋给了y，y再赋给x，于是乎最后就是'undefined'了。 |
| (**function** f(f) {  **return typeof** f(); })(**function** () {  **return** 1; }); | 相当于typeof 1 然后会返回 'number' |
| **var** foo = {  bar: **function** () {  **return this**.baz;  },  baz: 1 }; (**function** () {  **return typeof** arguments[0](); })(foo.bar); | 先看个例子： **function** test(){  alert(**this**); } 其实这样写的话这里面的this根本就是不确定的，也就不会弹窗，this指向哪要看在哪调用。 像这样写： **var** arr = [1,2,3]; arr.test = test; arr.test(); 就会弹窗数组了喔。 所以，本题中，foo中的this指向是不明确的，要看调用的地方，向foo.bar这样调用的话this是会指向foo的，但是foo.bar是整体作为参数传进去的，arguments[0]就相当于foo.bar，整体都作为函数名了，于是 arguments[0]()这种调用方式就相当于 **function** test(){  alert(**this**); } 上面说的这种调用方式了，也就是说this指向window了，window中没有baz，于是就会会返回**'undefined'**了 |
| **var** f = (**function** f() {  **return "1"**; }, **function** f() {  **return** 2 })(); **typeof** f; | 这里面首先有了分组选择符，我们先看看什么事分组选择符吧：  正常来说 var a = 1; 那么a就是1，但是如果这样写 var a = (1,2,3); 那么a就是3了，分组选择符执行最后一位。  于是就执行分组选择符后面那个return 2的函数了，于是typeof f就是 'number'。 |
| **var** x = 1; **if** (**function** f() { }) {  x += **typeof** f; } x; | function f(){}这样的函数必须是全局的或者是局部的，是不能写在运算里面的，于是下面f找不到 typeof f返回'undefined'，f虽然找不到，但是在if里面还是会返回真。于是变成了运算1+'undefined'，然后就会返回'1undefined'了。 |
| **var** x = [**typeof** x, **typeof** y][1]; **typeof typeof** x; | 很迷惑，其实根本不用管第一个表达式。typeof返回的必然是个字符串，于是typeof一个字符串了，于是就返回'string'。 |
| (**function** (foo) {  **return typeof** foo.bar; })({foo: {bar: 1}}); | 其实这里面的foo并不是指后面传进的参数中的foo，而是指整个参数{foo:{bar:1}}这个对象，但是这个对象中根本不存在bar属性，所以就会返回'undefined'了。 |
| (**function** f() {  **function** f() {  **return** 1;  }  **return** f();  **function** f() {  **return** 2;  } })(); | 函数声明预解析，后面函数已经把前面的覆盖掉了于是在执行f()的时候就是后面那个函数了,于是就返回2了。 |
| **function** f() {  **return** f; } **new** f() **instanceof** f; | 只看后面的表达式的话new f()就是构造函数生成的对象然后他去instanceof构造函数，然后就返回true了。  但是有return f  于是f覆盖掉new f()这个生成的对象了，于是就变成了f instanceof f了，然后就会返回false了。 |
| **with** (**function** (x, undefined) { }) length; | 函数的长度（length）就是函数的形参，而arguments是函数的实参集合。 比如说这样的 **function** test(num1,num2,num3){} alert(test.length); 就会弹出3  同理，本题就会得到2. |
| **var** a= (Math.PI++); **var** b = (Math.PI++); alert(a); alert(b); | 输出全是3.141592653589793，没有改变，说明系统对象的属性是只读的。 |

常用JavaScript代码

|  |  |
| --- | --- |
|  |  |
|  |  |
| 字符串中出现最多的字符和个数 | <script>  var str = 'aaaaaakkkkkkaaaaasddd';  function most(str){  //不同的字符在obj中都有个数组  /\* obj = {  a:[],  d:[]  }\*/  var obj = {};  var num = 0;  var value = '';  for(var i=0;i<str.length;i++){  //如果obj{}中这个字符对应的数组还没存在的话，那么给他创建个空数组  if(!obj[str[i]]){  obj[str[i]] = [];  }  //把这个字母push进它对应的数组中  obj[str[i]].push(str[i]);  }  for(attr in obj){  if(num<obj[attr].length){  num = obj[attr].length;  //obj[attr]数组中的每个字符都是相同的  value = obj[attr][0];  }  }  return '出现次数最多的字符是'+value+'出现了'+num+'次';  }  console.log(most(str));  </script>  <script>  var str = 'kkkkkkkaaakkkkkkaaaaasddd';  function most(str){  //排序，正则匹配子项  var arr = str.split('');  arr.sort();  str = arr.join('');  // \1匹配前面相同的,也就是说跟（\w）相同的  var re = /(\w)\1+/g;  var num = 0;  var value = '';  //这里的$0就是找到的重复的整体  str.replace(re,function($0,$1){  //alert($0);  if(num<$0.length){  num = $0.length;  value = $1;  }  });  return '出现次数最多的字符是'+value+'出现了'+num+'次';  }  console.log(most(str));  </script> |
|  |  |
| console.table | 关于打印json类型数据调试的方法  var languages = [  {name: "JavaScript", fileExtension: ".js"},  {name: "TypeScript", fileExtension: ".ts"},  {name: "CoffeeScript", fileExtension: ".coffee"} ]; console.table(languages); |
| 取URL中的参数 | **function** getParameterByName(name) {  **var match** = RegExp(**'[?&]'** + name + **'=([^&]\*)'**)  .exec(**window**.**location**.search);  **return match** && decodeURIComponent(**match**[1].replace(/\+/g, **' '**)); } |
|  |  |
| 对URL进行编码 | **var myUrl** = **"http://example.com/index.html?param=1&anotherParam=2"**; **var myOtherUrl** = **"http://example.com/index.html?url="** + encodeURIComponent(**myUrl**); |
| 非空字符串判断 | **if** (strValue) {  //do something  } |
| endsWith | //JavaScript实现endsWith String.prototype.endsWith = **function**(suffix) {  **return this**.indexOf(suffix, **this**.length - suffix.length) !== -1; }; **function** endsWith(str, suffix) {  **return** str.indexOf(suffix, str.length - suffix.length) !== -1; } |
| 克隆对象 | //JavaScript中克隆对象 **function** clone(obj) {  // Handle the 3 simple types, and null or undefined  **if** (**null** == obj || **"object"** != **typeof** obj) **return** obj;   // Handle Date  **if** (obj **instanceof** Date) {  **var copy** = **new** Date();  **copy**.setTime(obj.getTime());  **return copy**;  }   // Handle Array  **if** (obj **instanceof** Array) {  **var copy** = [];  **for** (**var i** = 0, **var** len = obj.length; **i** < **len**; ++i) {  **copy**[**i**]= clone(obj[**i**]);  }  **return copy**;  }   // Handle Object  **if** (obj **instanceof** Object) {  **var copy** = {};  **for** (**var attr in** obj) {  **if** (obj.hasOwnProperty(**attr**)) **copy**[**attr**] = clone(obj[**attr**]);  }  **return copy**;  }   **throw new** Error(**"Unable to copy obj! Its type isn't supported."**); } |
| base64编码 | JavaScript中base64编码      var Base64 = {      // private property      \_keyStr : "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",      // public method for encoding      encode : function (input) {          var output = "";          var chr1, chr2, chr3, enc1, enc2, enc3, enc4;          var i = 0;          input = Base64.\_utf8\_encode(input);          while (i < input.length) {              chr1 = input.charCodeAt(i++);              chr2 = input.charCodeAt(i++);              chr3 = input.charCodeAt(i++);              enc1 = chr1 >> 2;              enc2 = ((chr1 & 3) << 4) | (chr2 >> 4);              enc3 = ((chr2 & 15) << 2) | (chr3 >> 6);              enc4 = chr3 & 63;              if (isNaN(chr2)) {                  enc3 = enc4 = 64;              } else if (isNaN(chr3)) {                  enc4 = 64;              }              output = output +              this.\_keyStr.charAt(enc1) + this.\_keyStr.charAt(enc2) +              this.\_keyStr.charAt(enc3) + this.\_keyStr.charAt(enc4);          }          return output;      },      // public method for decoding      decode : function (input) {          var output = "";          var chr1, chr2, chr3;          var enc1, enc2, enc3, enc4;          var i = 0;          input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");          while (i < input.length) {              enc1 = this.\_keyStr.indexOf(input.charAt(i++));              enc2 = this.\_keyStr.indexOf(input.charAt(i++));              enc3 = this.\_keyStr.indexOf(input.charAt(i++));              enc4 = this.\_keyStr.indexOf(input.charAt(i++));              chr1 = (enc1 << 2) | (enc2 >> 4);              chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);              chr3 = ((enc3 & 3) << 6) | enc4;              output = output + String.fromCharCode(chr1);              if (enc3 != 64) {                  output = output + String.fromCharCode(chr2);              }              if (enc4 != 64) {                  output = output + String.fromCharCode(chr3);              }          }          output = Base64.\_utf8\_decode(output);          return output;      },      // private method for UTF-8 encoding      \_utf8\_encode : function (string) {          string = string.replace(/\r\n/g,"\n");          var utftext = "";          for (var n = 0; n < string.length; n++) {              var c = string.charCodeAt(n);              if (c < 128) {                  utftext += String.fromCharCode(c);              }              else if((c > 127) && (c < 2048)) {                  utftext += String.fromCharCode((c >> 6) | 192);                  utftext += String.fromCharCode((c & 63) | 128);              }              else {                  utftext += String.fromCharCode((c >> 12) | 224);                  utftext += String.fromCharCode(((c >> 6) & 63) | 128);                  utftext += String.fromCharCode((c & 63) | 128);              }          }          return utftext;      },      // private method for UTF-8 decoding      \_utf8\_decode : function (utftext) {          var string = "";          var i = 0;          var c = c1 = c2 = 0;          while ( i < utftext.length ) {              c = utftext.charCodeAt(i);              if (c < 128) {                  string += String.fromCharCode(c);                  i++;              }              else if((c > 191) && (c < 224)) {                  c2 = utftext.charCodeAt(i+1);                  string += String.fromCharCode(((c & 31) << 6) | (c2 & 63));                  i += 2;              }              else {                  c2 = utftext.charCodeAt(i+1);                  c3 = utftext.charCodeAt(i+2);                  string += String.fromCharCode(((c & 15) << 12) | ((c2 & 63) << 6) | (c3 & 63));                  i += 3;              }          }          return string;      }      }      //encode      Base64.encode("Test"); //VGVzdA==      //decode      Base64.decode("VGVzdA=="); // Test |

[setTimeout和setInterval的使用](http://www.cnblogs.com/qiantuwuliang/archive/2009/06/20/1507304.html)

|  |  |
| --- | --- |
| setTimeout/setInterval执行时机 | setTimeout()和setInterval()经常被用来处理延时和定时任务。都可以用来实现在一个固定时间段之后去执行JavaScript。不过两者各有各的应用场景。  setTimeout() 方法用于在指定的毫秒数后调用函数或计算表达式,  setInterval()则可以在每隔指定的毫秒数循环调用函数或表达式，直到clearInterval把它清除。  JavaScript其实是运行在单线程的环境中的，这就意味着定时器仅仅是计划代码在未来的某个时间执行，而具体执行时机是不能保证的，因为页面的生命周期中，不同时间可能有其他代码在控制JavaScript进程。在页面下载完成后代码的运行、事件处理程序、Ajax回调函数都是使用同样的线程，实际上浏览器负责进行排序，指派某段程序在某个时间点运行的优先级。  我们可以可以把JavaScript想象成在时间线上运行。当页面载入的时候首先执行的是页面生命周期后面要用的方法和变量声明和数据处理，在这之后JavaScript进程将等待更多代码执行。当进程空闲的时候，下一段代码会被触发  除了主JavaScript进程外，还需要一个在进程下一次空闲时执行的代码队列。随着页面生命周期推移，代码会按照执行顺序添加入队列，例如当按钮被按下的时候他的事件处理程序会被添加到队列中，并在下一个可能时间内执行。在接到某个Ajax响应时，回调函数的代码会被添加到队列。  JavaScript中没有任何代码是立即执行的，但一旦进程空闲则尽快执行。定时器对队列的工作方式是当特定时间过去后将代码插入，这并不意味着它会马上执行，只能表示它尽快执行。  关于setTimeout/setInterval执行时机详细说明可以看看 [setTimeout()和setInterval() 何时被调用执行](http://www.cnblogs.com/dolphinX/archive/2013/04/05/2784933.html" \t "http://www.cnblogs.com/dolphinX/p/_blank) |
| 实现无限循环 | 实际上，setTimeout和setInterval的语法相同。它们都有两个参数，一个是将要执行的代码字符串，还有一个是以毫秒为单位的时间间隔，当过了那个时间段之后就将执行那段代码。  不过这两个函数还是有区别的，setInterval在执行完一次代码之后，经过了那个固定的时间间隔，它还会自动重复执行代码，而setTimeout只执行一次那段代码。  虽然表面上看来setTimeout只能应用在on-off方式的动作上，不过可以通过创建一个函数循环重复调用setTimeout，以实现重复的操作：  /\*无限循环-setTimeout\*/ showTime(); **function** showTime() {  **var** today = **new** Date();  alert(**" The time is: "** + today.toString());  setTimeout(**" showTime() "**, 5000); }  /\*无限循环-setInterval\*/ setInterval(**" showTime() "**, 5000); **function** showTime() {  **var** today = **new** Date();  alert(**" The time is: "** + today.toString()); }  这两种方法可能看起来非常像，而且显示的结果也会很相似，不过两者的最大区别就是，setTimeout方法不会每隔5秒钟就执行一次showTime函数，它是在每次调用setTimeout后过5秒钟再去执行showTime函数。这意味着如果showTime函数的主体部分需要2秒钟执行完，那么整个函数则要每7秒钟才执行一次。而setInterval却没有被自己所调用的函数所束缚，它只是简单地每隔一定时间就重复执行一次那个函数。  如果要求在每隔一个固定的时间间隔后就精确地执行某动作，那么最好使用setInterval，而如果不想由于连续调用产生互相干扰的问题，尤其是每次函数的调用需要繁重的计算以及很长的处理时间，那么最好使用setTimeout。 |
| 函数指针的使用 | 两个计时函数中的第一个参数是一段代码的字符串，其实该参数也可以是一个函数指针，不过Mac下的IE 5对此不支持。  setTimeout(showTime, 500); **function** showTime() {  **var** today = **new** Date();  alert(**"The time is: "** + today.toString()); } //另外，匿名函数还可以声明为内联函数： setTimeout(**function** () {  **var** today = **new** Date();  alert(**"The time is: "** + today.toString()); }, 500);  如果用函数指针作为setTimeout和setInterval函数的第二个参数，那么它们就可以去执行一个在别处定义的函数了??? |
| 案例1 | 1. 项目中这样使用setTimeout() 2. Js代码   <!-- 系统公告紧急弹窗 -->  <script type="text/javascript" src="<%=path%>/js/jquery-1.4.4.min.js"></script>  <script type="text/javascript" src="<%=path%>/js/artDialog/artDialog.min.js"></script>  <script type="text/javascript">  **function** querySystemNotice() {  $.post(**'bussinessNotice\_querySystemNotice'**, {},  **function** (returnedData, status) {  **if** (returnedData != **null** && returnedData != **''**) {  $.dialog.open(**'<%=path%>/?noticeNum='**+returnedData,{  window: **'top'**, width: **'800px'**, height: **'400px'**,   title: **false**, border: **false**,  lock: **true**, esc: **false**, left: **'center'**,   content: **''**, yesText: **'确认阅读'**,  yesFn: **function**(){  $.post(**'<%=path%>/?noticeNum='** + returnedData, {},  **function** (returnedData2, status) {  //art.dialog.tips('已阅读！',1);    window.location.reload();  }  );  }  });  }**else**{  window.location.reload();  }  }); } setTimeout(**"querySystemNotice()"**,1000\*60);//1000为1秒钟,设置为1分钟。  </script> |
| 案例2 | 1. 我在项目中使用 setInterval()： 2. JS   **var** value = **'${sessionScope.refTime}'**; setInterval(**"refMoney()"**, value); /\*更新商户姓名、佣金额、资金额\*/ **function** refMoney(){  $.ajax({  type:**'post'**,  url:**'Login\_refMoney'**,  async: **false** ,  dataType:**'json'**,  success:**function**(data){  $(**"#bussinessName"**).html(**""**);  $(**"#bussinessAccount"**).html(**""**);  $(**"#bussinessCommission"**).html(**""**);  $(**"#bussinessName"**).html(data.bussinessName);  $(**"#bussinessAccount"**).html(data.bussinessAccount);  $(**"#bussinessCommission"**).html(data.bussinessCommission);  }  }); } //refMoney(); /\*退出系统\*/ **function** logout(){  **if**(confirm(**"确定退出系统吗?"**)) {  window.parent.parent.location.replace(**"<%=path%>"**);  } }   1. Html代码   欢迎： <**font** id="bussinessName">${user.bussinessName}</**font**> 佣金额： <**font** id="bussinessAccount"><**fmt:formatNumber** value="${user.bussinessAccount }" pattern="#0.00" type="number"/></**font**> 资金额： <**font** id="bussinessCommission"><**fmt:formatNumber** value="${user.bussinessCommission }" pattern="#0.00" type="number"/></**font**> <**a** href="<%=path%>/Login\_loginRight" target="main" >平台首页</**a**> <**a** href="javascript:logout();" >安全退出</**a**> |
| 注意点 | 1. 不要用 “SetTimeOut” 和 “Setinterval” 方法来作为 “Eval” 的备选。及不要用字符串作为setTimeout 或者 setInterval的参数   setTimeOut( "document.getID('value')", 3000);  setInterval('doSomethingPeriodically()', 1000);  setTimeout('doSomethingAfterFiveSeconds()', 5000);  setInterval("logTime()", 1000);  setTimeout("logMessage('" + msgValue + "')", 1000);  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，在每个代码执行中来执行一个字符串，这肯定会要慢些，会降低性能，因此，建议在这些方法中传递一个方法：  setTimeOut(yourFunction, 3000);  setInterval(doSomethingPeriodically, 1000);  setTimeout(doSomethingAfterFiveSeconds, 5000);  setInterval(logTime, 1000);  setTimeout(function() { logMessage(msgValue); }, 1000); |

typeof if ==

|  |  |  |
| --- | --- | --- |
| 类型判断 | typeof是操作符，并不是函数，也就是说我们在判断一变量类型时没必要加括号，直接使用即可  typeof 1; //number  typeof(1); //number, 没必要这么写，但是也没错，等于是给变量或常量包装了一下  typeof sth的返回值是字符串  typeof 123; //number  typeof NaN //number  typeof "123" //string  typeof false; //boolean  typeof undefined ; //undefined  typeof null //object  typeof new Array(); //object  typeof (new Array()); //觉得不够清晰可以这么用，结果是一样的  typeof (function(){}); //function  typeof a; //undefined  1. null的类型并不是null，而是object，所以我们不要寄希望与typeof帮我们判断null  2. typeof如果判断是对象只会返回object,而不会返回Array、Date的具体类型  3. function也是object的一种，按说也应该直接返回object。但是typeof对它区别对待了，返回了function，我们可以用此判断变量是否为函数  4. 没定义的变量同样返回undefined | |
|  | 0 == "0" == false;  [] == "" == 0 == false;  [10]=="10"==10  null == undefined;  '\t'==0;  '\n'==0;  '\r'==0;  '\n'!='\r'!='\n'!=''  undefined != false  null != false  如果两表达式的类型不同，==会试图将它们转换为字符串、数字或 Boolean 量。  负零等于正零。  相同的字符串、数值上相等的数字、相同的对象、相同的 Boolean 值或者（当类型不同时）能被强制转化为上述情况之一，均被认为是相等的。其他比较均被认为是不相等的。  所以说 0=='0'; 返回的也会是true。  判断两个变量是否真正相等：恒等运算符 ===、!== ，除了不进行类型转换，并且类型必须相同以外，恒等运算符与相等运算符的作用是一样的 | |
| == === | 使用== !=  ==和!=操作符会在需要的情况下自动转换数据类型。  而有些类型强制转化非常模糊：  if ({}) // ...  if ([]) // ...  最后两行的代码虽然条件判断为空（经常会被人误认为转化为false），但是其实不管是{ }还是[ ]都是一个实体类，而任何的类其实都会转化为true。 | 使用=== !==  用 === 和 !== 来替代== 和 != 以避免发生强制类型转化。===和!== 的用法和之前的== 和 != 一样，只不过他们不会发生类型强制转换。 |
| 使用 == ，如果两边的类型不同, js 引擎会先把它们转成相同的类型在进行值的比较；  使用 ===， 则不会进行类型转换，类型不同，肯定不相等。 | |
| if(){} | 在js中if不仅仅判断boolean决定真假，0、NaN、''、undefined、null、false都会被认为是假  if (!false) { console.log(1); } if (!0) { console.log(2); } if (!"") { console.log(3); } if (!undefined) { console.log(4); } if (!null) { console.log(5); } if (!NaN) { console.log(6); } 在console中123456都会被打印出来;  但是这并不意味着这些值就 == false | |
| 非空判断 | 在JavaScript程序库中创建一个新的JavaScript函数，如何判断函数的参数是否为null或者undefined？  一般情况下， 对js 某个变量s 判空习惯使用 if(s!=null) ；  如果s 没有定义的话，就会报undefined的js 错误, 所以完整的判空可以使用如下方式：  if(typeof(s)!="undefined"&&s!=null) {} | |
| 非空判断 | /\*  如果variable1不是一个空对象，或者非未定义，或者不等于空字符串，则将v1赋值给v2；  简写代码：  var variable2 = variable1 || '';  注：不支持 0、 false  \*/ if (variable1 !== null || variable1 !== undefined || variable1 !== '') {  var variable2 = variable1; }  /\*exp 为 undefined、null\*/ if (exp == null) {  alert("exp 为 undefined、null"); } /\*exp 为 undefined、null、0、 false\*/ if (!exp) {  alert("exp 为 undefined、null、0、 false"); }  /\*  判断字符串是否为空:  很多情况下，都是用length来直接判断字符串是否为空  \*/ var strings = ''; if (strings.length == 0) {  alert('输入不能为空'); } /\*  但如果用户输入的是空格，制表符，换页符，也是不为空的，但是这样的数据就不是我们想要的；  其实可以用正则表达式来把这些“空”的符号去掉，然后进行判断；  \s匹配任何空白字符，包括空格、制表符、换页符...，等价于 [\f\n\r\t\v]；  \*/ var strings = ' '; if (strings.replace(/(^\s\*)|(\s\*$)/g, "").length == 0) {  alert('输入不能为空'); } | |

undefined null NaN

|  |  |
| --- | --- |
|  | <http://blog.csdn.net/>oscar999/article/details/9353713 |
| 在介绍这三个之间的差别之前， 先来看一下JS的数据类型。  在 Java ,C这样的语言中， 使用一个变量之前，需要先定义这个变量并指定它的数据类型，是整型，字符串型，....  但是在js 中定义变量统一使用 var , 或者不使用var 也可以使用。  那么js 中是否有数据类型的概念呢？ 当然有， 使用 typeof 就可以判断这个变量的数据类型：  js中的数据类型有undefined,boolean,number,string,object等5种，前4种为原始类型，第5种为引用类型。  除了以上5 中类型之外， 还有一种 “function”的类型。  有了上面的介绍，就可以很容易把undefined 和其他的两个区分开来。  undefined判断的是变量的类型，而其他两个判断是变量的值。  undefined可以用来表示以下的状况  1. 表示一个未声明的变量，  2. 已声明但没有赋值的变量，  3. 一个并不存在的对象属性  null 是一种特殊的object ,表示无值;  NaN是一种特殊的number ,表示无值; |
| undefined null | null == undefined;//true null === undefined;//false  var s; log(s == undefined);//true log(s === undefined);//true log(s == null);//true log(s === null);//false  var s = null; log(s == undefined);//true log(s === undefined);//false log(s == null);//true log(s === null);//true |
| NaN | 关于NaN  NaN 与包括其本身在内的任何值都不相等:NaN != NaN;  任何值与 NaN 比较的时候，甚至包括他自己，结果都是false。  因此我们不能用简单的比较字符来决定一个值是否为 NaN 。我们可以用内置的 isNaN() 函数来辨别:  NaN == NaN; // false  NaN === NaN; // false  isNaN(NaN); // true |
| 参数：undefined-not defined | var log = function () {  var s=null;  console.log(s); };  log();//null |
| log(11)//报错！！！ var log=function(x){  console.log(x) } |
| function lianXi(){  log(s) } lianXi()//报错！！！ |
| function lianXi(s){  log(s) } lianXi()//undefined |
| function log() {  var s;  console.log(s); };  log();//undefined |
| function lianXi(s){  log(s||8) } lianXi()// 8 |
| function lianXi(s){  log(s||8) } lianXi("")// 8 |
| function lianXi(s){  log(s||8) } lianXi(" ")// " " |
| function lianXi(s){  log(s||8) } lianXi(0)// 8 |
| function lianXi(s){  log(s||8) } lianXi(false) // 8 |

ajax

|  |  |  |
| --- | --- | --- |
| get|post | | [Ajax中get与post请求详解](http://redarmychen.iteye.com/blog/978125)  <http://redarmychen.iteye.com/blog/978125> |
| http状态码 | HTTP状态码（HTTP Status Code）是用以表示网页[服务器](http://baike.baidu.com/view/899.htm" \t "http://baike.baidu.com/_blank)HTTP响应状态的3位数字代码。  所有状态码的第一个数字代表了响应的五种状态之一。   |  |  | | --- | --- | | 消息1 | 代表请求已被接受，需要继续处理。这类响应是临时响应，只包含状态行和某些可选的响应头信息，并以空行结束。由于 HTTP/1.0 协议中没有定义任何 1xx 状态码，所以除非在某些试验条件下，服务器禁止向此类客户端发送 1xx 响应。 | | 成功2 | 这一类型的状态码，代表请求已成功被服务器接收、理解、并接受。 | | 重定向3 | 这类状态码代表需要客户端采取进一步的操作才能完成请求。通常，这些状态码用来重定向，后续的请求地址（重定向目标）在本次响应的 Location 域中指明。 | | 请求错误4 | 这类的状态码代表了客户端看起来可能发生了错误，妨碍了服务器的处理。除非响应的是一个 HEAD 请求，否则服务器就应该返回一个解释当前错误状况的实体，以及这是临时的还是永久性的状况。这些状态码适用于任何请求方法。浏览器应当向用户显示任何包含在此类错误响应中的实体内容。 | | 服务器错误5 | 这类状态码代表了服务器在处理请求的过程中有错误或者异常状态发生，也有可能是服务器意识到以当前的软硬件资源无法完成对请求的处理。除非这是一个HEAD 请求，否则服务器应当包含一个解释当前错误状态以及这个状况是临时的还是永久的解释信息实体。浏览器应当向用户展示任何在当前响应中被包含的实体。 |   100 Continue  初始的请求已经接受，客户应当继续发送请求的其余部分  101 Switching Protocols  服务器将遵从客户的请求转换到另外一种协议  200 OK  一切正常，对GET和POST请求的应答文档跟在后面  201 Created  服务器已经创建了文档，Location头给出了它的URL。  202 Accepted  已经接受请求，但处理尚未完成。  203 Non-Authoritative Information  文档已经正常地返回，但一些应答头可能不正确，因为使用的是文档的拷贝  204 No Content  没有新文档，浏览器应该继续显示原来的文档。如果用户定期地刷新页面，而Servlet可以确定用户文档足够新，这个状态代码是很有用的  205 Reset Content  没有新的内容，但浏览器应该重置它所显示的内容。用来强制浏览器清除表单输入内容  206 Partial Content  客户发送了一个带有Range头的GET请求，服务器完成了它  300 Multiple Choices  客户请求的文档可以在多个位置找到，这些位置已经在返回的文档内列出。如果服务器要提出优先选择，则应该在Location应答头指明。  301 Moved Permanently  客户请求的文档在其他地方，新的URL在Location头中给出，浏览器应该自动地访问新的URL。  302 Found  类似于301，但新的URL应该被视为临时性的替代，而不是永久性的。  303 See Other  类似于301/302，不同之处在于，如果原来的请求是POST，Location头指定的重定向目标文档应该通过GET提取  304 Not Modified  客户端有缓冲的文档并发出了一个条件性的请求（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。  305 Use Proxy  客户请求的文档应该通过Location头所指明的代理服务器提取  307 Temporary Redirect  和302（Found）相同。许多浏览器会错误地响应302应答进行重定向，即使原来的请求是 POST，即使它实际上只能在POST请求的应答是303时才能重定向。由于这个原因，HTTP 1.1新增了307，以便更加清除地区分几个状态代码： 当出现303应答时，浏览器可以跟随重定向的GET和POST请求；如果是307应答，则浏览器只能跟随对GET请求的重定向。  400 Bad Request  请求出现语法错误。  401 Unauthorized  客户试图未经授权访问受密码保护的页面。应答中会包含一个WWW-Authenticate头，浏览器据此显示用户名字/密码对话框，然后在填写合适的Authorization头后再次发出请求。  403 Forbidden  资源不可用。  404 Not Found  无法找到指定位置的资源  405 Method Not Allowed  请求方法（GET、POST、HEAD、Delete、PUT、TRACE等）对指定的资源不适用。  406 Not Acceptable  指定的资源已经找到，但它的MIME类型和客户在Accpet头中所指定的不兼容  407 Proxy Authentication Required  类似于401，表示客户必须先经过代理服务器的授权。  408 Request Timeout  在服务器许可的等待时间内，客户一直没有发出任何请求。客户可以在以后重复同一请求。  409 Conflict  通常和PUT请求有关。由于请求和资源的当前状态相冲突，因此请求不能成功。  410 Gone  所请求的文档已经不再可用，而且服务器不知道应该重定向到哪一个地址。它和404的不同在于，返回407表示文档永久地离开了指定的位置，而404表示由于未知的原因文档不可用。  411 Length Required  服务器不能处理请求，除非客户发送一个Content-Length头。  412 Precondition Failed  请求头中指定的一些前提条件失败  413 Request Entity Too Large  目标文档的大小超过服务器当前愿意处理的大小。如果服务器认为自己能够稍后再处理该请求，则应该提供一个Retry-After头  414 Request URI Too Long  URI太长  416 Requested Range Not Satisfiable  服务器不能满足客户在请求中指定的Range头  500 Internal Server Error  服务器遇到了意料不到的情况，不能完成客户的请求  501 Not Implemented  服务器不支持实现请求所需要的功能。例如，客户发出了一个服务器不支持的PUT请求  502 Bad Gateway  服务器作为网关或者代理时，为了完成请求访问下一个服务器，但该服务器返回了非法的应答  503 Service Unavailable  服务器由于维护或者负载过重未能应答。例如，Servlet可能在数据库连接池已满的情况下返回503。服务器返回503时可以提供一个Retry-After头  504 Gateway Timeout  由作为代理或网关的服务器使用，表示不能及时地从远程服务器获得应答  505 HTTP Version Not Supported  服务器不支持请求中所指明的HTTP版本 | | |

注意点

|  |  |
| --- | --- |
| parseInt | 语法： parseInt(string, radix)  string：  必需。要被解析的字符串。  radix：  可选。表示要解析的数字的基数。该值介于 2 ~ 36 之间。  如果省略该参数或其值为 0，则数字将以 10 为基础来解析。如果它以 “0x” 或 “0X” 开头，将以 16 为基数。  如果该参数小于 2 或者大于 36，则 parseInt() 将返回 NaN  当参数 radix 的值为 0，或没有设置该参数时，parseInt() 会根据 string 来判断数字的基数。  举例，如果 string（开头结尾空格自动省略） 以 "0x" 开头，parseInt() 会把 string 的其余部分解析为十六进制的整数。如果 string 以 0 开头，那么 ECMAScript v3 允许 parseInt() 的一个实现把其后的字符解析为八进制或十六进制的数字。如果 string 以 1 ~ 9 的数字开头，parseInt() 将把它解析为十进制的整数。如果字符串的第一个字符不能被转换为数字，那么 parseFloat() 会返回 NaN。  parseInt("10"); //返回 10  parseInt("19",10); //返回 19 (10+9)  parseInt("11",2); //返回 3 (2+1)  parseInt("17",8); //返回 15 (8+7)  parseInt("1f",16); //返回 31 (16+15)  parseInt("010"); //未定：返回 10 或 8 |
| 预解析 | console.log(a);  //Error:a is not defined 直接报错，下面语句没法执行  console.log(b); //undefined  var b="Test";  console.log(b); //Test  很奇怪前两句变量a,b都没有声明，第一句却报错，第二句能够输出undefined？  这是因为JavaScript是解释型语言，但它并不是直接逐步执行的，JavaScript解析过程分为先后两个阶段，一个是预处理阶段，另外一个就是执行阶段。在预处理阶段JavaScript解释器将完成把JavaScript脚本代码转换到字节码，然后第二阶段JavaScript解释器借助执行环境把字节码生成机械码，并顺序执行。  也就说JavaScript值执行第一句语句之前就已经将函数/变量声明预处理了，var b=”Test” 相当于两个语句，var b;（undefined结果的来源，在执行第一句语句之前已经解析），b=”Test”（这句是顺序执行的，在第二句之后执行）。  这也是为什么我们可以在方法声明语句之前就调用方法的原因:  showMsg(); // This is message  function showMsg(){ alert('This is message'); } |
| 块级作用域 | JavaScript没有块级作用域，只有函数级作用域，这就意味着{}在JavaScript中只能起到语法块的作用，而不能起到作用域块作用  if(true){//语法块，保证{}内代码if条件成立执行 //... var a=3;}console.log(a); //3  上面例子可以清楚看到属于window的console.log方法依然可以访问貌似是局部变量的a |
| 闭包 | 首先从一个经典错误谈起，页面上有若干个div， 我们想给它们绑定一个onclick方法，于是有了下面的代码  <div id="divTest"> <span>0</span> <span>1</span> <span>2</span> <span>3</span> </div> <div id="divTest2"> <span>0</span> <span>1</span> <span>2</span> <span>3</span> </div>  $(document).ready(function() { var spans = $("#divTest span"); for (var i = 0; i < spans.length; i++) { spans[i].onclick = function() { alert(i); } } });  很简单的功能可是却偏偏出错了，每次alert出的值都是4，这和我们原先预期的并不一样。这是因为当点击事件被触发的时候，for循环早已执行完毕，i的值也已经从0变成了4。简单的修改就好使：  var spans2 = $("#divTest2 span"); $(document).ready(function() { for (var i = 0; i < spans2.length; i++) { (function(num) { spans2[i].onclick = function() { alert(num); } })(i); } });  闭包是指有权限访问另一个函数作用域的变量的函数，创建闭包的常见方式就是在一个函数内部创建另一个函数,只要存在调用内部函数的可能，JavaScript就需要保留被引用的函数。而且JavaScript运行时需要跟踪引用这个内部函数的所有变量，直到最后一个变量废弃，JavaScript的垃圾收集器才能释放相应的内存空间. |
| 在for循环中的不正确函数调用 | **var** elements = document.getElementsByTagName(**'input'**); **var** n = elements.length; **var** makeHandler = **function** (num) { // outer function  **return function** () {  console.log(**"This is element #"** + num);  }; }; **for** (**var** i = 0; i < n; i++) {  elements[i].onclick = makeHandler(i + 1); }  在这个版本的代码中， makeHandler 在每回循环的时候都会被立即执行，把i＋1传递给变量num。外面的函数返回里面的函数，而点击事件函数便被设置为里面的函数。这样每个触发函数就都能够是用正确的i值了。 |
| 对于this关键词的不正确引用 | 下面让我们一起来看这一段代码：  Game.prototype.restart = function () {  this.clearLocalStorage();  this.timer = setTimeout(function () {  this.clearBoard();  }, 0);  };  运行上面的代码将会出现如下错误：  Uncaught TypeError: undefined is not a function  这是为什么？this的调用和它所在的环境密切相关。之所以会出现上面的错误，是因为当你在调用 setTimeout()函数的时候, 你实际调用的是window.setTimeout(). 因此，在 setTimeout() 定义的函数其实是在window背景下定义的，而window中并没有 clearBoard() 这个函数方法。  下面提供两种解决方案。第一种比较简单直接的方法便是，把this存储到一个变量当中，这样他就可以在不同的环境背景中被继承下来：  Game.prototype.restart = function () {  this.clearLocalStorage();  var self = this;  this.timer = setTimeout(function () {  self.clearBoard();  }, 0);  };  第二种方法便是用bind()的方法，不过这个相比上一种要复杂一些，对于不熟悉bind()的同学可以在微软官方查看它的使用方法：https://msdn.microsoft.com/zh-cn/library/ff841995  Game.prototype.restart = function () {  this.clearLocalStorage();  this.timer = setTimeout(this.reset.bind(this), 0);  };  Game.prototype.reset = function () {  this.clearBoard();  };  上面的例子中，两个this均指代的是Game.prototype。 |
| 生命周期---块级作用域 | 另一种易犯的错误，便是带着其他编程语言的思维，认为在JS中，也存在生命周期这么一说。请看下面的代码：  for (var i = 0; i < 10; i++) { /\* ... \*/ } console.log(i);  如果你认为在运行console.log() 时肯定会报出 undefined 错误，那么你就大错特错了。我会告诉你其实它会返回 10吗。  当然，在许多其他语言当中，遇到这样的代码，肯定会报错。因为i明显已经超越了它的生命周期。在for中定义的变量在循环结束后，它的生命也就结束了。但是在js中，i的生命还会继续。这种现象叫做 variable hoisting。  而如果我们想要实现和其他语言一样的在特定逻辑模块中具有生命周期的变量，可以用let关键字。 |
| 内存泄露 | 内存泄露在js变成中几乎是一个无法避免的问题。如果不是特别细心的话，在最后的检查过程中，肯定会出现各种内存泄露问题。下面我们就来举例说明一下：  **var** theThing = **null**; **var** replaceThing = **function** () {  **var** priorThing = theThing;  **var** unused = **function** () {  **if** (priorThing) {  console.log(**"hi"**);  }  };  theThing = {  longStr: **new** Array(1000000).join(**'\*'**), //   someMethod: **function** () {  console.log(someMessage);  }  }; }; setInterval(replaceThing, 1000);  如果运行上面的代码，你会发现你已经造成了大量的内存泄露，每秒泄露1M的内存，显然光靠GC（垃圾回收器）是无法帮助你的了。由上面的代码来看，似乎是longstr在每次replaceThing调用的时候都没有得到回收。这是为什么呢？  每一个theThing结构都含有一个longstr结构列表。每一秒当我们调用 replaceThing, 它就会把当前的指向传递给 priorThing. 但是到这里我们也会看到并没有什么问题，因为 priorThing 每回也是先解开上次函数的指向才会接受新的赋值。并且所有的这一切都是发生在 replaceThing 函数体当中，按常理来说当函数体结束之后，函数中的本地变量也将会被GC回收，也就不会出现内存泄露的问题了，但是为什么会出现上面的错误呢？  这是因为longstr的定义是在一个闭包中进行的，而它又被其他的闭包所引用，js规定，在闭包中引入闭包外部的变量时，当闭包结束时此对象无法被垃圾回收（GC）。关于在JS中的内存泄露问题可以查看http://javascript.info/tutorial/memory-leaks#memory-management-in-javascript |
| 避免低效的DOM操作 | js中的DOM基本操作非常简单，但是如何能有效地进行这些操作一直是一个难题。这其中最典型的问题便是批量增加DOM元素。增加一个DOM元素是一步花费很大的操作。而批量增加对系统的花销更是不菲。一个比较好的批量增加的办法便是使用 document fragments ：  **var** div = document.getElementsByTagName(**"my\_div"**); **var** fragment = document.createDocumentFragment(); **for** (**var** e = 0; e < elems.length; e++) {  fragment.appendChild(elems[e]); } div.appendChild(fragment.cloneNode(**true**));  直接添加DOM元素是一个非常昂贵的操作。但是如果是先把要添加的元素全部创建出来，再把它们全部添加上去就会高效很多。 |
| 原型继承问题 | //很大一部分的js开发者都不能完全掌握原型的继承问题。下面具一个例子来说明： BaseObject = **function** (name) {  **if** (**typeof** name !== "undefined") {  **this**.name = name;  } **else** {  **this**.name = 'default'  } }; //这段代码看起来很简单。如果你有name值，则使用它。如果没有，则使用 ‘default’: **var firstObj** = **new BaseObject**(); **var secondObj** = **new BaseObject**('unique'); console.log(**firstObj**.name); // -> 结果是'default' console.log(**secondObj**.name); // -> 结果是 'unique' //但是如果我们执行delete语句呢: **delete secondObj**.name; //我们会得到: console.log(**secondObj**.name); // -> 结果是 'undefined'  //但是如果能够重新回到 ‘default’状态不是更好么? 其实要想达到这样的效果很简单，如果我们能够使用原型继承的话: **BaseObject** = **function** (name) {  **if** (**typeof** name !== "undefined") {  **this**.name = name;  } }; **BaseObject**.prototype.name = 'default'; //在这个版本中, BaseObject 继承了原型中的name 属性, 被设置为了 ‘default’.。这时，如果构造函数被调用时没有参数，则会自动设置为 default。相同地，如果name 属性被从BaseObject移出，系统将会自动寻找原型链，并且获得 ‘default’值： **var thirdObj** = **new BaseObject**('unique'); console.log(**thirdObj**.name); **delete thirdObj**.name; console.log(**thirdObj**.name); // -> 结果是 'default' |
| 为实例方法创建错误的指引 | 我们来看下面一段代码:  var MyObject = function() {}  MyObject.prototype.whoAmI = function() {  console.log(this === window ? "window" : "MyObj"); };  var obj = new MyObject();  现在为了方便起见，我们新建一个变量来指引 whoAmI 方法, 因此我们可以直接用 whoAmI() 而不是更长的obj.whoAmI():  var whoAmI = obj.whoAmI;  接下来为了确保一切都如我们所预测的进行，我们可以将 whoAmI 打印出来:  console.log(whoAmI);  结果是：  function () { console.log(this === window ? "window" : "MyObj"); }  没有错误！  但是现在我们来查看一下两种引用的方法：  obj.whoAmI(); // 输出 "MyObj" (as expected)  whoAmI(); // 输出 "window" (uh-oh!)  哪里出错了呢？  原理其实和上面的第二个常见错误一样，当我们执行 var whoAmI = obj.whoAmI;的时候，新的变量 whoAmI 是在全局环境下定义的。因此它的this 是指window, 而不是obj！  正确的编码方式应该是：  **var** MyObject = **function** () { } MyObject.prototype.whoAmI = **function** () {  console.log(**this** === window ? **"window"** : **"MyObj"**); }; **var** obj = **new** MyObject(); obj.w = obj.whoAmI;  // still in the obj namespace obj.whoAmI();  // 输出 "MyObj" (as expected)  obj.w();  // 输出 "MyObj" (as expected) |
| {} or [] | 一个常犯的错误在于使用当需要数组的时候使用一个对象或者该使用对象的时候使用一个数组。  但是使用原则很简单：“当属性名称是小的连续整数，你应该使用数组。否则，使用一个对象” – Douglas Crockford, JavaScript: Good Parts 的作者.  建议：  var a = ['1A','2B'];  避免：  var a = new Array();  a[0] = "1A";  a[1] = "2B"; |
| strict mode | “strict mode” 是一种更加严格的代码检查机制，并且会让你的代码更加安全。当然，不选择这个模式并不意味着是一个错误，但是使用这个模式可以确保你的代码更加准确无误。  下面我们总结几条“strict mode”的优势：  让Debug更加容易：在正常模式下很多错误都会被忽视掉，“strict mode”模式会让Debug极致更加严谨。  防止默认的全局变量：在正常模式下，给一个为经过声明的变量命名将会将这个变量自动设置为全局变量。在strict模式下，我们取消了这个默认机制。  取消this的默认转换：在正常模式下，给this关键字指引到null或者undefined会让它自动转换为全局。在strict模式下，我们取消了这个默认机制。  防止重复的变量声明和参数声明：在strict模式下进行重复的变量声明会被抱错，如(e.g., var object = {foo: “bar”, foo: “baz”};) 同时，在函数声明中重复使用同一个参数名称也会报错，如 (e.g., function foo(val1, val2, val1){}),  让eval()函数更加安全。  当遇到无效的delete指令的事后报错：delete指令不能对类中未有的属性执行，在正常情况下这种情况只是默默地忽视掉，而在strict模式是会报错的。  严格模式："use strict"：  <http://www.xuanfengge.com/4637.html>  .使用"use strict"可以节约你的时间  "use strict"是JavaScript中一个非常好的特性，而且非常容易使用。  使用方法  // file.js "use strict" function doStuff(){     // use strict is enabled here! }  这样佻的file.js都会应用上"use strict"模式。 如果你仅想在一个函数中使用：  // file.js function a(){  "use strict";  // use strict is enabled in this context  function nestedFunction(){  // and here too  } }  好处  检查对象中的重复键  var zombie = {  eyeLeft : 0,  eyeRight: 1,  // ... a lot of keys ...  eyeLeft : 1 }  这段代码会抛出一个错误因为 eyeLeft 出现了两次。这比你用眼睛去找错误要快多了。  未声明变量  plane = 5;  你现在已经知道忘记在这个变量前面加var了。不过如果你不知道，调试起来是非常痛苦的，因为这个变量是声明在全局上下文（global context）中的，而且可能被其他地方改掉。想象一下，如果你声明了一个全局的 i， 在嵌套循环中可能会引起混乱。  重复的参数  function run(fromWhom, fromWhom){}  注意fromWho出现了两次，因此会抛出一个错误。  限制函数中的arguments  var run = function(fromWhom){  arguments[0] = 'alien';  alert(fromWhom); } run('zombie'); // alert: 'alien';  现在你可以使用"use strict"  var run = function(fromWhom){  "use strict";  arguments[0] = 'alien';  alert(fromWhom); } run('zombie'); // alert: 'zombie';  arguments[0] = 'alien' 改变了参数fromWhom，use strict 又节约了你的时间。 |
| prototypes | 添加新的属性到对象 prototype 中是导致脚本出错的常见原因。  yourObject.prototype.anotherFunction = 'Hello';  yourObject.prototype.anotherMethod = function(){...}  在上面代码中，所有的变量都会被影响，因为他们都继承于 yourObject。这样的使用会导致意想不到的行为。所以建议在使用完后删除类似的修改。  yourObject.prototype.anotherFunction = 'Hello';  yourObject.prototype.anotherMethod = function () { … };  test.anotherMethod();  delete yourObject.prototype.anotherFunction = 'Hello';  delete yourObject.prototype.anotherMethod = function () { … }; |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |